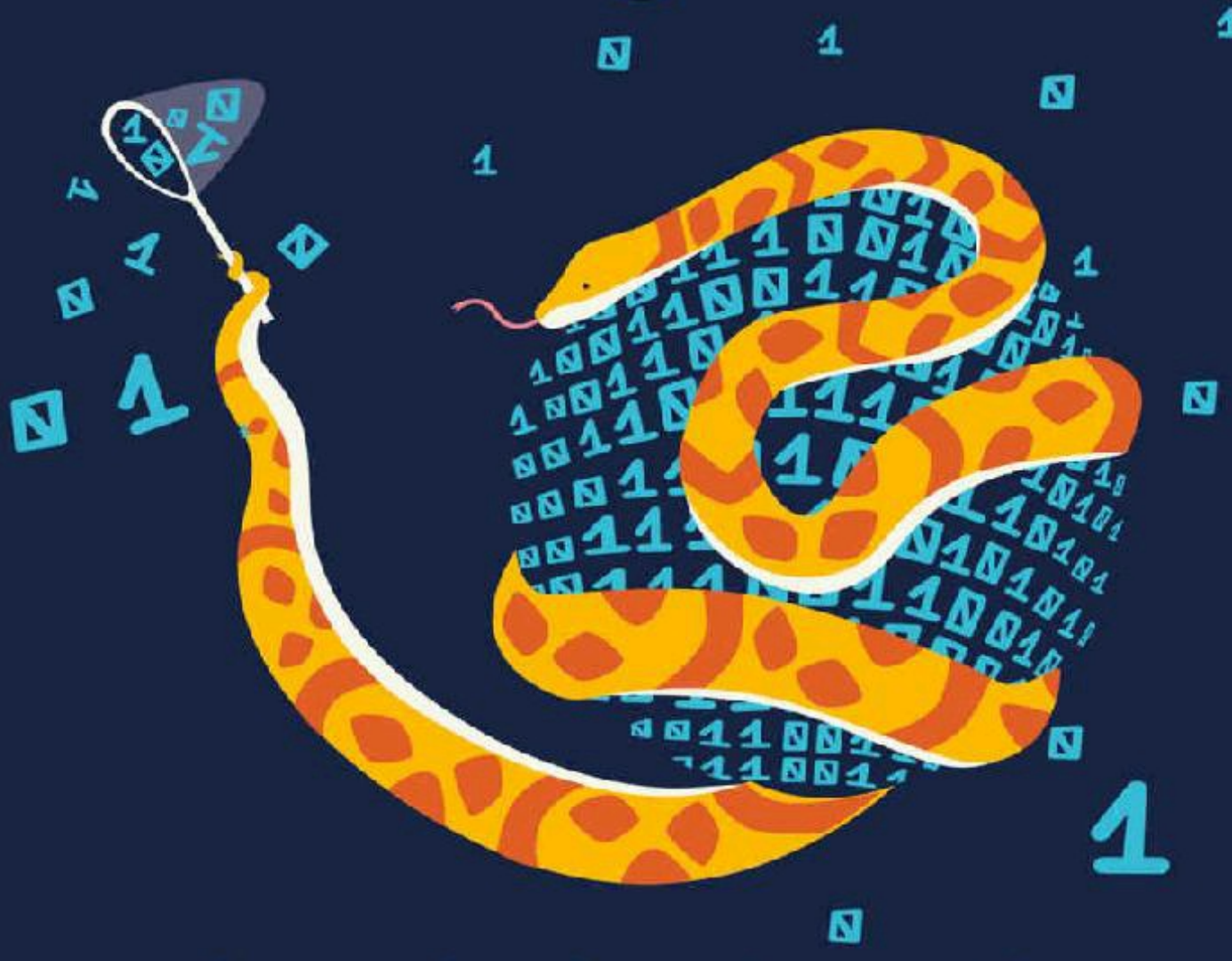
DMITRY ZINOVIEV

Data Science con Python



Dalle stringhe al machine learning, le tecniche essenziali per lavorare sui dati

APOGEO

Data Science con Python
Dalle stringhe al machine learning, le tecniche essenziali per lavorare sui dati

Dmitry Zinoviev

ISBN edizione cartacea: 9788850334148

Alla mia meravigliosa e brillante moglie Anna.

Ai nostri figli:
Eugenia, leggiadra ballerina,
e Roman, romantico giocatore.

Alla mia prima classe
di Data Science dell'estate 2015.

Sono grato al professor Xinxin Jiang (Suffolk University) per i suoi preziosi commenti inclusi nella sezione del libro dedicata alla statistica e a Jason Montojo (uno degli autori di *Practical Programming: An Introduction to Computer Science Using Python 3*), Amirali Sanatinia (Northeastern University), Peter Hampton (Ulster University), Anuja Kelkar (Carnegie Mellon University) e Lokesh Kumar Makani (Skyhigh Networks) per le loro indispensabili revisioni.

Devo parlarvi di scienza, fra poco, giusto per distrarvi dai vostri pensieri.
Marie Corelli, romanziera britannica

Questo libro è stato ispirato da un corso introduttivo alla scienza dei dati in Python che ho tenuto nell'estate del 2015 a un piccolo e selezionato gruppo di studenti della Suffolk University di Boston. Il corso era stato concepito per essere il primo di una sequenza di due corsi, con un'enfasi sull'ottenimento, la "ripulitura", l'organizzazione e la visualizzazione dei dati, con una spruzzata di elementi di statistica, machine learning e analisi dei dati di rete.

Ben presto mi sono reso conto che l'abbondanza di sistemi e moduli Python coinvolti da queste operazioni (database, framework per l'elaborazione del linguaggio naturale, parser JSON e HTML e strutture dati ad alte prestazioni, solo per citarne alcune) sarebbe stata davvero eccessiva non solo per uno studente universitario, ma anche per un professionista esperto. In realtà, devo confessare che, mentre mi occupavo dei miei progetti di ricerca nei campi della scienza dei dati e dell'analisi dei dati di rete, dedicavo molto tempo, fin troppo, a richiamare la funzione `help()` e a sfogliare le discussioni online su Python. Inoltre, devo ammettere di aver avuto anche alcuni momenti imbarazzanti durante la lezione, quando sembravo aver completamente dimenticato il nome di alcune funzioni o di alcuni parametri opzionali.

Nell'ambito del materiale del corso, ho anche compilato una serie di appunti e schemi sui vari argomenti, da usare come riferimento. Tali appunti e schemi si sono evoluti fino a diventare questo libro. Auspicabilmente, il fatto di avere questo libro sulla vostra scrivania vi aiutera a riflettere più sulla scienza dei dati e l'analisi dei dati che sui nomi di funzioni e parametri opzionali.

Informazioni su questo libro

Questo libro si occupa di acquisizione, pulizia, memorizzazione, ricerca, trasformazione, visualizzazione dei dati, con elementi di analisi dei dati avanzata (l'analisi dei dati di rete), statistica e machine learning. Non è un'introduzione alla scienza dei dati o una guida di riferimento generale alla scienza dei dati, anche se troverete una panoramica introduttiva nel Capitolo 1, Che cosa si intende per scienza dei dati?. Presuppongo quindi che abbiate già appreso altrove i metodi tipici della scienza dei dati e della statistica. L'indice analitico che si trova alla fine del libro fa riferimento alle implementazioni Python dei concetti chiave, ma in molti casi scoprirete che tali concetti vi sono già familiari.

Nel Capitolo 2, *Elementi di base di Python per la scienza dei dati*, troverete un riepilogo delle strutture di dati di Python; funzioni per stringhe, file e il Web; espressioni regolari; manipolazione delle liste. Tale capitolo ha lo scopo di rinfrescarvi la memoria su questi argomenti, non di insegnarli. Sono disponibili molti eccellenti testi su Python e una buona conoscenza di tale linguaggio è assolutamente importante per diventare veri esperti di scienza dei dati.

La prima parte del libro mostra la manipolazione di vari tipi di dati testuali, inclusa l'elaborazione di testi strutturati e non strutturati, l'elaborazione di dati numerici con i moduli NumPy e Pandas e l'analisi dei dati di rete. Altri tre capitoli si occupano di vari aspetti dell'analisi: l'uso di database relazionali e non-relazionali, la visualizzazione dei dati e l'analisi predittiva.

Questo libro ha un suo filo logico ma è anche una guida di riferimento. A seconda delle vostre esigenze, potete leggerlo sequenzialmente o saltare direttamente all'indice, trovare la funzione o il concetto che vi interessa e ricercare le descrizioni e gli esempi pertinenti. Nel primo caso, se siete già esperti programmatori Python, potete tranquillamente saltare il Capitolo 2, Elementi di base di Python per la scienza dei dati. Se non pensate di dover interagire con dei database esterni (come MySQL), potete ignorare il Capitolo 4, Utilizzare i database. Infine, il Capitolo 9, Probabilità e

statistica, presuppone che non abbiate alcuna idea della statistica. Se già conoscete l'argomento, saltatene pure le prime due Unità e partite dall'Unità 47, Statistica in Python.

A chi si rivolge questo libro

A questo punto, potreste chiedervi se non sia il caso di far diventare questo libro una presenza stabile nella vostra libreria.

Il libro è rivolto a studenti e laureati, ai docenti di scienza dei dati, ai professionisti alle prime armi nel campo della scienza dei dati (in particolare a coloro che stanno passando da R a Python) e agli sviluppatori alla ricerca di una guida di riferimento che li aiuti a ricordare tutte le funzioni e le opzioni di Python.

Se rientrate in questo breve ritratto, questo libro fa proprio per voi.

Questioni di software

Nonostante alcune controversie sulla transizione da Python 2.7 a Python 3.3 e alle versioni successive, sono decisamente convinto sostenitore della più recente versione di Python. La maggior parte del nuovo software per Python è stata sviluppata per la versione 3.3 e anche la maggior parte del software precedente è stata portata alla versione 3.3. Tenendo conto di questa tendenza, sarebbe poco saggio aggrapparsi a una versione obsoleta, indipendentemente dalla sua popolarità.

Tutti gli esempi in Python di questo libro possono funzionare con i moduli menzionati nella Tabella I.1. Tutti questi moduli, con l'eccezione del modulo community che deve essere installato separatamente (https://pypi.python.org/pypi/python-louvain/0.3) e dello stesso interprete Python, sono inclusi nella distribuzione Anaconda, distribuita da Continuum Analytics e disponibile gratuitamente (https://www.continuum.io).

Tabella I.1 Componenti software usati nel libro.

Pacchetto	Versione usata	Pacchetto	Versione usata
BeautifulSoup	4.3.2	community	0.3
json	2.0.9	html5lib	0.999
matplotlib	1.4.3	networkx	1.10.0
nltk	3.1.0	numpy	1.10.1
pandas	0.17.0	pymongo	3.0.2
pymysql	0.6.2	python	3.4.3
scikit-learn	0.16.1	scipy	0.16.0

Se pensate di provare a usare (o di usare effettivamente, per lavoro) i database, dovrete anche scaricare e installare MySQL (https://www.mysql.com) e MongoDB (https://www.mongodb.com). Entrambi i database sono gratuiti e funzionano perfettamente in Linux, Mac OS e Windows.

Note sugli apici

Python consente di racchiudere le stringhe di caratteri in apici 'singoli', "doppi", "tripli" e perfino """tre doppi apici""" (le ultime due possibilità sono utili per le stringhe multiriga). Tuttavia, quando stampa le stringhe, impiega sempre la notazione a singolo apice, indipendentemente dal numero di apici usati nel programma.

Molti altri linguaggi (C, C++, Java) usano gli apici singoli e doppi in modo molto differente: singoli per i caratteri singoli, doppi per le stringhe di caratteri. Come tributo a questa differenziazione, anche in questo libro useremo i singoli apici per i caratteri e i doppi apici per le stringhe di caratteri.

Il forum del libro

Il forum della community di questo libro si trova sul sito dell'editore inglese The Pragmatic Programmers (https://pragprog.com/book/dzpyds/data-science-essentials-in-python). Qui potete porre domande, inviare commenti e segnalare errori.

Un'altra ottima risorsa per le domande e le risposte (ma non relative in modo specifico a questo libro) è il recente forum Data Science Stack Exchange (https://datascience.stackexchange.com).

Scarica il codice degli esempi

Il codice sorgente di molti esempi presentati nel testo è disponibile sul sito dell'editore originale inglese The Pragmatic Programmers. L'indirizzo per scaricare gli archivi ZIP o TGZ è: https://pragprog.com/titles/dzpyds/source_code (o per comodità http://bit.ly/pp-code-dsp).

Esercitazioni

Alla fine di ogni capitolo si trova una particolare Unità chiamata "Esercitazioni". Questa Unità presenta vari progetti che potete svolgere per vostro conto (o anche insieme ad altri) per rafforzare e "concretizzare" le vostre conoscenze sul materiale teorico proposto.

I progetti contrassegnati con una stella (*) sono i più semplici. Per risolverli vi basterà una buona conoscenza delle funzioni menzionate nei capitoli precedenti. Potete pensare di completare tali progetti in una trentina di minuti. Le soluzioni di tali esercitazioni si trovano nell'Appendice B, *Soluzioni dei progetti di tipo* (*).

I progetti contrassegnati con due stelle (**) sono più complessi. La loro risoluzione può portarvi via un'ora o più, a seconda delle vostre abilità e abitudini di programmazione. I progetti a due stelle richiedeono l'uso di strutture di dati intermedie e di algoritmi ben ragionati.

Infine, i progetti a tre stelle (***) sono i più difficili. Alcuni progetti a tre stelle potrebbero perfino non prevedere una soluzione davvero "perfetta", pertanto non preoccupatevi se non riuscite a trovarla! Ma il fatto di lavorare su questi progetti, migliorerà certamente le vostre abilità di programmatori e vi renderà anche migliori esperti di scienza dei dati. I docenti possono anche considerare i progetti a tre stelle come suggerimenti per assegnazioni di attività pratiche ai loro studenti.

E ora… cominciamo!

Dmitry Zinoviev

dzinoviev@gmail.com

Agosto 2016

Che cosa si intende per scienza dei dati?

È impossibile misurare l'imponderabile.
Kozma Prutkov, autore russo

Sono sicuro che abbiate già almeno un'idea di cosa si intenda con *scienza dei dati*, ma vale la pena di iniziare da qui! La scienza dei dati è la disciplina che consiste nell'estrarre conoscenze dai dati. È un mix di informatica (per le strutture di dati, gli algoritmi, la rappresentazione, la capacità di elaborare grandi quantità di dati e programmazione), statistica (per le regressioni e l'inferenza) e conoscenza del dominio (perché dobbiamo sapere quali domande porre e come interpretare i risultati).

La scienza dei dati, tradizionalmente, si occupa di vari argomenti dissimili, alcuni dei quali potreste forse già conoscere, mentre altri magari li incontrerete per la prima volta in queste pagine.

- *Database*, che forniscono le funzionalità di memorizzazione e integrazione delle informazioni. Troverete informazioni sui database relazionali e sui sistemi di archiviazione dei documenti nel Capitolo 4, *Utilizzare i database*.
- *Analisi del testo* ed *elaborazione del linguaggio naturale*, che ci consentono di eseguire "calcoli sulle parole" trasformando un testo qualitativo in variabili quantitative. Siete interessati agli strumenti per l'analisi del sentiment? Ne parleremo nell'Unità 16, *Elaborazione di testi in linguaggio naturale*.
- *Analisi numerica dei dati* e *data mining*, alla ricerca di schemi coerenti e di relazioni fra le variabili. Questi sono gli argomenti del Capitolo 5, *Usare i dati numerici tabulari*, e del Capitolo 6, *Manipolare serie di dati e frame*.

- *Analisi delle reti complesse*, un argomento tutt'altro che complesso. Riguarda le reti complesse, ovvero collezioni di entità arbitrarie interconnesse. Il Capitolo 7, *Utilizzo dei dati delle reti*, semplifica l'analisi delle reti complesse.
- *Rappresentazione dei dati*, che non deve essere solo gradevole, ma anche utile, in particolare quando occorre convincere gli sponsor a proseguire nell'impegno. Se un'immagine vale più di mille parole, allora il Capitolo 8, *Rappresentazione grafica*, vale da solo il resto del libro.
- *Machine learning* (comprendendo il clustering, gli alberi decisionali, la classificazione e le reti neurali), che tenta di far "pensare" i computer, in modo che possano effettuare previsioni sulla base dei dati del campione. Il Capitolo 10, *Machine Learning*, spiega come fare.
- *Elaborazione delle serie temporali* e, più in generale, *elaborazione dei segnali digitali*, strumenti indispensabili per gli analisti del mercato di borsa, per gli economisti e per i ricercatori nei domini audio e video.
- *Analisi di grandi quantità di dati*; normalmente si fa riferimento all'analisi di una massa di dati non strutturati (testo, audio, video) superiore a 1 TB, prodotti e catturati ad una frequenza elevata. Si tratta di un argomento davvero troppo "grande" per poter essere trattato in questo libro.

Indipendentemente dal tipo di analisi, la scienza dei dati è prima di tutto scienza e solo in seconda battuta "stregoneria". Pertanto, è un processo che segue una sequenza di base piuttosto rigorosa, che inizia con l'acquisizione dei dati e termina con un report sui risultati.

In questo capitolo, esamineremo i processi di base della scienza dei dati: i passi di un tipico studio di analisi dei dati, dove acquisire i dati e la struttura tipica del report di un progetto.

Unità 1 - La tipica sequenza di analisi dei dati

I passi di un tipico studio di analisi dei dati, in genere sono coerenti con la sequenza prevista per qualsiasi altro ambito scientifico.

Una scoperta, in scienza dei dati, inizia con una domanda cui trovare una risposta e con il tipo di analisi da applicare. Il tipo di analisi più semplice è quello *descrittivo*, in cui il dataset viene descritto facendo riferimento alle sue misure aggregate, spesso in una forma visuale. Indipendentemente da come si procederà, bisogna almeno descrivere i dati! Durante l'analisi *esplorativa* dei dati, si cerca di trovare nuove relazioni fra le variabili disponibili. Se abbiamo a disposizione solo pochi dati campione e vogliamo usarli per descrivere una popolazione di maggiori dimensioni, l'analisi *inferenziale* statistica è lo strumento perfetto. Un analista *predittivo* impara dal passato per prevedere il futuro. L'analisi *causale* identifica le variabili che influenzano le altre. Infine, l'analisi dei dati *meccanicistica* esplora esattamente il modo in cui una variabile influenza un'altra variabile.

Tuttavia, la qualità della vostra analisi dipende dalla qualità dei dati utilizzati. Qual è l'aspetto di un dataset ideale? Quali dati conterrebbero la risposta alla vostra domanda in un mondo ideale? A proposito, il dataset ideale potrebbe non esistere affatto o essere difficile, se non impossibile, da ottenere. A seconda delle situazioni, magari un dataset più piccolo o meno ricco di caratteristiche potrebbe funzionare altrettanto bene.

Fortunatamente, trarre dati grezzi dal Web o da un database non è poi così difficile, e Python offre moltissimi strumenti che assistono nel download e nella decifrazione dei dati. Ne riparleremo nell'Unità 2, *Sequenza di acquisizione dei dati*.

In questo mondo imperfetto, i dati perfetti non esistono. Nei dati vi possono essere valori mancanti, valori anomali e altri elementi "non standard". Fra gli esempi di dati "sporchi" vi possono essere date di nascita nel futuro, valori negativi di età e peso, indirizzi email inutilizzabili (noreply@). Una volta ottenuti i dati grezzi, il passo successivo è usare degli

strumenti di “pulizia” dei dati e le vostre conoscenze statistiche per regolarizzare il dataset.

Una volta che nei file abbiamo a disposizione i dati “puliti”, potete utilizzarli per svolgere analisi descrittive ed esplorative. L’output di questo passo spesso include dei grafici a dispersione (Unità 44), istogrammi e riepiloghi statistici (Unità 46). Questi strumenti vi daranno un’idea del significato dei dati, un’intuizione che è indispensabile per le successive ricerche, in particolare se il dataset ha molte dimensioni.

E ora siete solo a un passo dall’iniziare a prevedere il futuro. I vostri strumenti operativi sono i modelli dei dati che, se adeguatamente addestrati, sono in grado di apprendere dal passato per prevedere il futuro. Senza mai dimenticare di misurare la qualità dei modelli realizzati e l’accuratezza delle loro previsioni!

A questo punto dovete togliervi dal capo il cappellino dell’esperto di statistica e di programmatore e indossare quello di esperto del dominio. Avete a disposizione determinati risultati, ma quale significato hanno nel dominio? In altre parole, tali risultati interessano a qualcuno e sono in grado di cambiare le cose? Immaginate di essere un revisore, incaricato di valutare il vostro stesso lavoro. Dove avete ragione, dove avete sbagliato e dove potreste ottenere risultati migliori o differenti, se aveste un’altra *chance*? Dovreste usare altri tipi di dati, svolgere altri tipi di analisi, porre domande differenti o costruire un modello differente? Qualcuno vi porrà queste domande, ed è meglio che siate voi i primi. Iniziate quindi a ricercare le risposte quando siete ancora profondamente immersi nel contesto.

Infine, e non meno importante, dovete produrre un report che spieghi come e perché avete elaborato i dati, quali modelli avete utilizzato e quali conclusioni e previsioni sono possibili. Parleremo di struttura dei report verso la fine di questo capitolo, nell’Unità 3, *Struttura dei report*.

Dato che il suo scopo è quello di fungere da supporto per la scelta delle aree appropriate della scienza dei dati basandosi sul linguaggio Python, il *focus* di questo libro è rivolto principalmente ai primi, meno formalizzati e più creativi passi della tipica sequenza di analisi dei dati: ottenere, ripulire, organizzare e dimensionare i dati. La modellazione dei dati, inclusa la

modellazione dei dati a fini predittivi, è appena accennata (sarebbe sbagliato trascurare del tutto la modellazione, perché è lì che si può individuare la "vera magia" dell'analisi dei dati!). In generale, l'interpretazione, la verifica e la rappresentazione dei risultati sono molto specifiche del dominio e pertanto appartengono a testi più specializzati.

Unità 2 - Sequenza di acquisizione dei dati

L'acquisizione dei dati riguarda tutto ciò che è necessario per ottenere gli artifatti che contengono i dati di input provenienti dalle varie fonti, per estrarre i dati da tali artifatti e per convertirli in rappresentazioni adatte per le successive elaborazioni, come illustrato nella seguente figura.

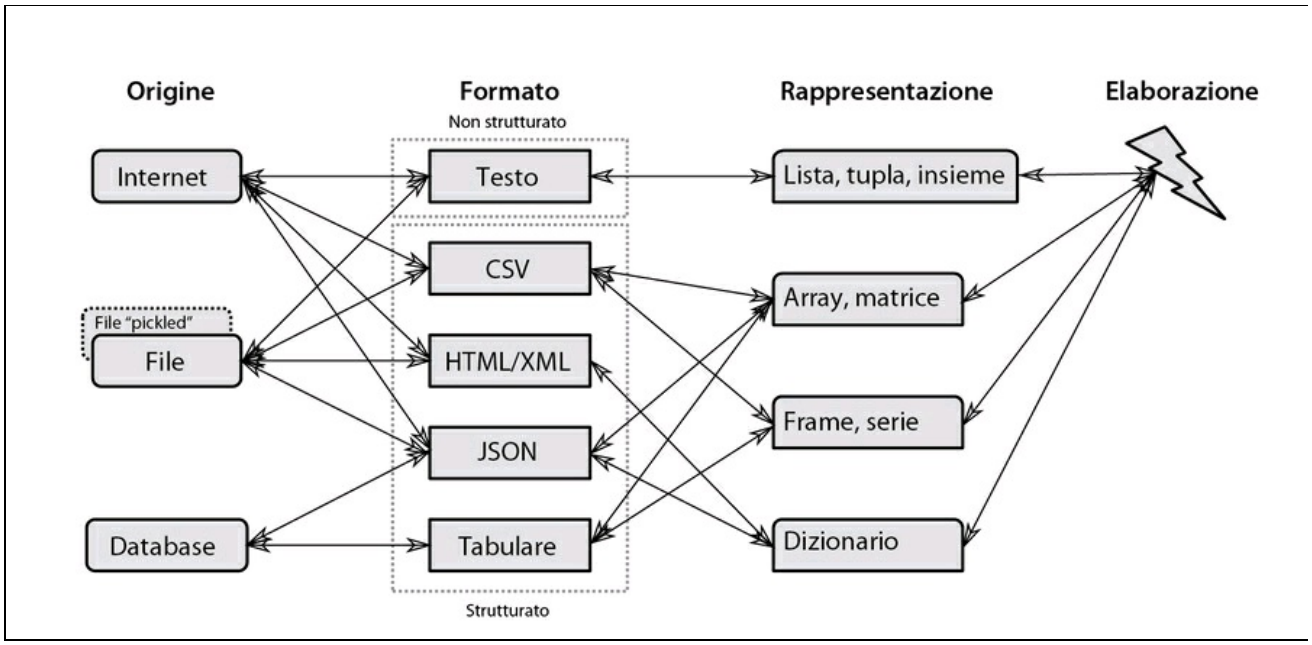


Figura 1.1

Le tre principali fonti di dati sono Internet (in particolare, il World Wide Web), i database e i file locali (magari precedentemente scaricati a mano o usando ulteriore software). Alcuni dei file locali possono essere stati prodotti da altri programmi Python e contenere dati serializzati o "pickled" (vedere l'Unità 12, *Pickling dei dati*).

I formati di dati presenti negli artifatti possono essere i più disparati. Nei prossimi capitoli, vedremo come gestire e manipolare i formati più comuni.

- Testo standard non strutturato in linguaggio naturale (l'inglese, l'italiano, il cinese ecc.).
- Dati strutturati, fra cui:
  - dati tabulari in file CSV (*Comma Separated Values*);

- dati tabulari tratti da database;
- dati a tag in formato HTML (*HyperText Markup Language*) o, in generale, in XML (*eXtensible Markup Language*);
- dati a tag in formato JSON (*JavaScript Object Notation*).

A seconda della struttura originaria dei dati estratti e dello scopo e della natura delle successive elaborazioni, i dati usati negli esempi di questo libro vengono rappresentati usando le strutture di dati native di Python (liste e dizionari) o strutture di dati avanzate che supportano operazioni specializzate (array numpy e frame pandas).

Ho cercato di fare in modo che la sequenza di elaborazione dei dati (ottenimento, pulizia e trasformazione di dati grezzi; analisi descrittiva ed esplorativa dei dati; modellazione dei dati e previsione) fosse completamente automatizzata. Per questo motivo, ho evitato di usare strumenti grafici interattivi, i quali solo raramente sono realizzati per operare in modalità batch, e solo raramente registrano una cronologia delle operazioni. Per favorire la modularità, la riusabilità e la recuperabilità, ho suddiviso questa lunga sequenza in sequenze più piccole, salvando i risultati intermedi in file in formato pickle (Unità 12) o JSON (Unità 15).

L'automazione della sequenza porta, naturalmente, ad avere codice riproducibile: un insieme di script Python che chiunque può eseguire per convertire i dati grezzi originari nei risultati finali descritti nel report, teoricamente senza alcuna ulteriore interazione umana. Altri ricercatori potranno usare il codice riproducibile per convalidare i vostri modelli e i vostri risultati e per applicare ai loro problemi il processo che avete sviluppato.

Unità 3 - Struttura dei report

Il report è ciò che noi, esperti di scienza dei dati, forniamo al committente, al cliente. In genere un report include i seguenti elementi.

- Sintesi (una breve descrizione del progetto).
- Introduzione.
- Metodi usati per l'acquisizione e l'elaborazione dei dati.
- Risultati ottenuti (escludendo i risultati intermedi e insignificanti, da specificare, semmai, nell'Appendice).
- Conclusioni.
- Appendice.

Oltre ai risultati e ai contenuti grafici non essenziali, l'Appendice deve contenere tutto il codice riproducibile usato per elaborare i dati: script ben commentati che possono essere eseguiti senza alcun parametro e interazione da parte dell'utente.

L'ultima ma non meno importante parte che deve essere fornita è rappresentata dai dati grezzi: tutti i file di dati necessari per eseguire il codice in modo riproducibile, a meno che tali file siano stati forniti dal committente e che non siano stati modificati in alcun modo. Un file README in genere spiega la provenienza dei dati e il formato di ogni file di dati allegato.

Considerate questa struttura come una raccomandazione, non come un dogma. Il vostro committente e il vostro buon senso potranno suggerirvi le implementazioni alternative.

Esercitazioni

In questo capitolo introduttivo, abbiamo esaminato i processi che stanno alla base della scienza dei dati: i passi tipici di uno studio di analisi dei dati, dove ottenere i dati e i vari formati dei dati e, infine, la tipica struttura del report di un progetto. La parte rimanente del libro introduce le caratteristiche di Python che sono essenziali per applicare le basi della scienza dei dati, così come i moduli Python che offrono un supporto algoritmico e statistico a un progetto di scienza dei dati di moderata complessità.

Prima di procedere, svolgiamo un semplice progetto per provare a usare Python. Per tradizione, i programmatori sono soliti presentare un nuovo linguaggio di programmazione scrivendo un programma che produce in output le parole “Hello, World!”. E non vi è alcun buon motivo per infrangere questa tradizione.

Hello, World! (*)

Scrivete un programma che produca in output le parole “Hello, World!” (senza gli apici) sulla riga di comando Python.

Elementi di base di Python per la scienza dei dati

> E ho parlato con loro in tutte le lingue di cui avessi avuto almeno la minima infarinatura, alto e basso olandese, latino, francese, spagnolo, italiano e lingua franca, ma del tutto inutilmente.
> *Jonathan Swift*, scrittore e poeta irlandese

Alcune caratteristiche di base del linguaggio Python sono più importanti di altre per l'analisi dei dati. In questo capitolo, tratteremo le più essenziali: funzioni per le stringhe, strutture di dati, manipolazione delle liste, contatori, funzioni per i file e per il Web, espressioni regolari, globbing e pickling dei dati. Imparerete a usare Python per estrarre dati dai file locali e da Internet, memorizzarli in strutture di dati appropriate, individuare i dettagli e gli elementi che rispondono a determinati criteri e serializzare e de-serializzare gli oggetti Python per le successive elaborazioni. Tuttavia, queste funzioni non sono affatto specifiche delle attività di scienza dei dati o di analisi dei dati e si ritrovano in molte altre applicazioni.

È opinione comune che la disponibilità di strumenti di programmazione di alto livello renda ormai obsoleta la programmazione di basso livello. Ma è un equivoco! Dato che la sola distribuzione Anaconda di Python offre più di 350 package Python, perché imparare a separare le stringhe e ad aprire i file? La verità è che, nel mondo, le fonti di dati non standard sono almeno altrettante di quelle che seguono le regole.

Tutti i frame di dati, le serie, i lettori di file CSV e i tokenizer seguono le regole predisposte dai loro sviluppatori. Tuttavia tutti questi sistemi falliscono miseramente quando si incontrano dati che non seguono le regole. E in quel momento vi toccherà soffiare via la polvere da questo volume, togliervi dal capo l'aura del grande esperto di scienza dei dati e indossare le vesti dell'umile, ma utile, programmatore.

Si può dover “scendere” fino al livello delle funzioni per le stringhe, delle quali parliamo nella prossima pagina.

NOTA

In questo libro usiamo il simbolo → per indicare gli output dei vari esempi di codice Python.

Unità 4 - Le funzioni che operano sulle stringhe

Una stringa è una delle unità di base dell'interazione fra il mondo dei computer e il mondo degli esseri umani. Inizialmente, quasi tutti i dati grezzi vengono memorizzati come stringhe. In questa Unità, vedremo come si valutano e manipolano le stringhe di testo.

Tutte le funzioni descritte in questa Unità sono membri della classe interna `str`.

Le funzioni di conversione del *caso* (maiusole, minuscole) restituiscono una copia della stringa originale `s`: `lower()` converte tutti i caratteri in lettere minuscole; `upper()` converte tutti i caratteri in lettere maiuscole; `capitalize()` pone il primo carattere maiuscolo e tutti i successivi caratteri minuscoli. Queste funzioni non alterano i caratteri non alfabetici. Si tratta di funzioni di conversione che rappresentano elementi importanti di normalizzazione, argomento di cui parleremo nell'Unità 16.

Le *funzioni predicato* restituiscono `True` o `False`, a seconda del fatto che la stringa `s` appartenga a una specifica classe: `islower()` controlla se tutti i caratteri alfabetici sono in minuscole; `isupper()` controlla se tutti i caratteri alfabetici sono in maiuscole; `isspace()` controlla se tutti i caratteri sono spazi; `isdigit()` controlla se tutti i caratteri sono cifre decimali nell'intervallo 0-9; `isalpha()` controlla se tutti i caratteri sono alfabetici e negli intervalli `a-z` e `A-Z`. Si possono usare queste funzioni per riconoscere le parole valide, i numeri interi non negativi, i segni di punteggiatura e così via.

Talvolta Python rappresenta i dati di una stringa come semplici array binari, non come stringhe di caratteri, in particolare quando i dati provengono da una fonte esterna: un file, un database o il Web. Python impiega la notazione `b` per gli array binari. Per esempio, `bin = b"Hello"` è un array binario; `s = "Hello"` è una stringa. Rispettivamente, `s[0]` è `'H'` e `bin[0]` è `72`, dove `72` è il codice ASCII del carattere `'H'`. Le funzioni di decodifica convertono un array binario in una stringa di caratteri e viceversa: `bin.decode()` converte un

array binario in una stringa e s.encode() converte una stringa in un array binario. Molte funzioni Python si aspettano che i dati binari siano convertiti in stringhe per le successive elaborazioni.

Il primo passo nell'elaborazione delle stringhe consiste nello sbarazzarsi degli spazi indesiderati (inclusi i codici di fine riga e di tabulazione). Le funzioni lstrip() (*left strip* - elimina a sinistra), rstrip() (*right strip* - elimina a destra) e strip() eliminano gli spazi presenti, rispettivamente, all'inizio, alla fine o all'interno della stringa (strip() non elimina però gli spazi interni, solo quelli in eccesso). Con tutte queste rimozioni, si rischia di rimanere con una stringa vuota!

```
"    Hello, world! \t\t\n".strip()
```

→ 'Hello, world!'

Spesso una stringa è costituita da vari token, separati da delimitatori come spazi e segni di punteggiatura. La funzione split(delim='') suddivide la stringa s in una lista di sottostringhe, sulla base del delimitatore delim. Se il delimitatore non viene specificato, Python suddivide la stringa in base agli spazi e raggruppa insieme tutti gli spazi contigui:

```
"Hello,  world!".split()      # Due spazi!
```

→ ['Hello,', 'world!']

```
"Hello,  world!".split(" ")   # Due spazi!
```

→ ['Hello,', '', 'world!']

```
"www.networksciencelab.com".split(".")
```

→ ['www', 'networksciencelab', 'com']

La funzione gemella join(ls) unisce la lista di stringhe ls a formare un'unica stringa, usando come "collante" l'oggetto stringa. In pratica, con join() potete ricombinare i frammenti:

```
", " .join(["alpha", "bravo", "charlie", "delta"])
```

→ 'alpha, bravo, charlie, delta'

Nell'esempio precedente, join() inserisce la "colla" solo *fra* le stringhe e non davanti alla prima stringa o dopo l'ultima. Il risultato della suddivisione di una stringa con split() e della successiva unione dei frammenti con join() spesso è indistinguibile dalla sostituzione del delimitatore di suddivisione con la "colla":

```
"-" .join("1.617.305.1985".split("."))
```

→ '1-617-305-1985'

Talvolta, infatti, le due funzioni vengono usate insieme per eliminare da una stringa gli spazi indesiderati. Potete ottenere lo stesso effetto con la sostituzione tramite espressioni regolari (di cui parleremo all'interno dell'Unità 10).

```
" ".join("This string\n\r  has    many\t\tspaces".split())
```

→ 'This string has many spaces'

La funzione `find(needle)` restituisce l'indice della prima occorrenza della sottostringa `needle` nell'oggetto stringa oppure `-1` se non trova la sottostringa. Questa funzione distingue fra lettere maiuscole e minuscole e viene usata per trovare un frammento utile (se esiste) all'interno di una stringa.

```
"www.networksciencelab.com".find(".com")
```

→ 21

La funzione `count(needle)` restituisce il numero di occorrenze non sovrapposte della sottostringa `needle` nell'oggetto stringa. Anche questa funzione distingue fra lettere maiuscole e minuscole.

```
"www.networksciencelab.com".count(".")
```

→ 2

Le stringhe sono un importante elemento costitutivo di ogni programma di elaborazione dei dati, ma non sono l'unico (e neanche il più efficiente). Potete anche usare le liste, le tuple, i set e i dizionari per unire insieme stringhe e dati numerici e per svolgere efficienti operazioni di ricerca e ordinamento.

Unità 5 - La struttura dati appropriata

Le strutture composite più comunemente usate per memorizzare i dati in Python sono le liste, le tuple, i set e i dizionari. Sono tutti esempi di *collezioni*.

Python implementa le *liste* sotto forma di array. Il loro tempo di ricerca è lineare, il che le rende una soluzione impraticabile per la memorizzazione di grandi quantità di dati soggetti a ricerche.

Le *tuple* sono liste immutabili. Questo significa che una volta create, non possono più essere modificate. Anch'esse hanno un tempo di ricerca lineare.

A differenza delle liste e delle tuple, i *set* non sono sequenze: ciò significa che gli elementi dei set non hanno un indice. I set possono memorizzare al massimo una copia di ogni elemento e hanno un tempo di ricerca sub-lineare, O(1). Pertanto sono eccellenti per la determinazione dell'appartenenza e per eliminare i duplicati (se convertite in un set una lista contenente duplicati, i duplicati spariscono):

```
myList = list(set(myList))    # Rimuove i duplicati da myList
```

Potete trasformare una lista di dati in un set per accelerare la determinazione dell'appartenenza. Per esempio, supponiamo che `bigList` sia una lista dei primi dieci milioni di numeri interi, rappresentati come stringhe decimali:

```
bigList = [str(i) for i in range(10000000)]
"abc" in bigList      # Richiede 0,2 sec
bigSet = set(bigList)
"abc" in bigSet       # Richiede 15-30 µsec e quindi è 10000 volte più veloce!
```

I *dizionari* creano una mappa fra chiavi e valori. Come chiave può essere usato un qualsiasi oggetto hashable (un numero, un valore booleano, una stringa o una tupla), e chiavi differenti nello stesso dizionario possono appartenere a tipi di dati differenti.

Non vi è alcuna restrizione quanto ai tipi di dati dei valori di un dizionario. Anche i dizionari hanno un tempo di ricerca sublineare, O(1). Sono pertanto eccellenti per le ricerche chiave-valore.

Potete creare un dizionario da una lista di tuple (chiave, valore), e potete usare il costruttore interno `enumerate(seq)` per creare un dizionario in cui la

chiave è il numero sequenziale del rispettivo elemento presente in `seq`:

```
seq = ["alpha", "bravo", "charlie", "delta"]
dict(enumerate(seq))
```

→ {0: 'alpha', 1: 'bravo', 2: 'charlie', 3: 'delta'}

Un altro modo interessante per creare un dizionario da una sequenza di chiavi (`kseq`) e una sequenza di valori (`vseq`) è attraverso un costruttore interno, `zip(kseq, vseq)` (le sequenze devono essere della stessa lunghezza):

```
kseq = "abcd"       # Anche una stringa è una sequenza
vseq = ["alpha", "bravo", "charlie", "delta"]
dict(zip(kseq, vseq))
```

→ {'a': 'alpha', 'c': 'charlie', 'b': 'bravo', 'd': 'delta'}

Python implementa i generatori di liste `enumerate(seq)` e `zip(kseq, vseq)` (e anche il buon vecchio `range()`). I generatori di liste forniscono un'interfaccia iterativa, che consente di utilizzarli nei cicli. A differenza di una vera lista, un generatore di liste produce l'elemento successivo solo su richiesta. I generatori facilitano l'utilizzo di grandi liste e permettono perfino l'esistenza di liste "infinite". Potete trasformare esplicitamente un generatore in una lista, richiamando la funzione `list()`.

Unità 6 - Definizione e manipolazione delle liste

La manipolazione delle liste consiste nell'impiegare un'espressione per trasformare una collezione (non necessariamente una lista) in una lista. Viene usata per applicare la stessa operazione a tutti gli elementi di una lista (o solo a una parte di essi), come convertire tutti gli elementi in lettere maiuscole o elevarli tutti allo stesso esponente.

Il processo di trasformazione ha il seguente aspetto.

1. L'espressione itera su tutta la collezione, visitandone gli elementi.
2. Per ogni elemento viene valutata un'espressione booleana opzionale (default = `True`).
3. Se l'espressione booleana è `True`, l'espressione del ciclo viene valutata per l'elemento corrente e il suo valore viene aggiunto alla lista del risultato.
4. Se l'espressione booleana è `False`, l'elemento viene ignorato.

Ecco alcune semplici manipolazione di liste:

```
# Copia myList; equivale a myList.copy() o myList[:], ma è meno efficiente
[x for x in myList]

# Estrai gli elementi non negativi
[x for x in myList if x >= 0]

# Crea una lista di elevamenti al quadrato
[x**2 for x in myList]

# Crea una lista di reciproci
[1/x for x in myList if x != 0]

# Raccogli tutte le righe non vuote dal file infile,
# eliminando gli spazi in eccesso iniziali e finali
[l.strip() for l in infile if l.strip()]
```

Nell'ultimo esempio, la funzione `strip()` viene valutata per due volte per ogni elemento della lista. Per evitare le duplicazioni, potete usare la manipolazione nidificata delle liste. Quella interna elimina gli spazi e quella esterna elimina le stringhe vuote:

```
[line for line in [l.strip() for l in infile] if line]
```

Racchiudendo la manipolazione di una lista fra parentesi tonde anziché quadre, si ottiene un oggetto generatore di liste:

```
(x**2 for x in myList)    # Restituisce <generator object <genexpr> at 0x...>
```

Spesso il risultato della manipolazione delle liste è una lista di elementi ripetuti: numeri, parole, gruppi di parole e frasi. A volte occorre sapere quali sono gli elementi più oppure meno comuni. La classe Counter, descritta nell'Unità 7, *I contatori*, è un piccolo strumento utile per raccogliere questo genere di informazioni statistiche.

Unità 7 - I contatori

Un contatore è una collezione a dizionario che ha lo scopo di contare gli elementi di un'altra collezione. È definito nel modulo `collections`. Potete passare la collezione da contare al costruttore di `Counter` e poi usare la funzione `most_common(n)` per ottenere una lista degli `n` elementi più frequenti e delle rispettive frequenze (se non fornite `n`, la funzione restituisce una lista di tutti gli elementi).

```
from collections import Counter
phrase = "a man a plan a canal panama"
cntr = Counter(phrase.split())
cntr.most_common()
```

→ [('a', 3), ('canal', 1), ('panama', 1), ('plan', 1), ('man', 1)]

Quest'ultima lista può essere convertita in un dizionario per facilitare le ricerche:

```
cntrDict = dict(cntr.most_common())
```

→ {'a': 3, 'canal': 1, 'panama': 1, 'plan': 1, 'man': 1}

```
cntrDict['a']
```

→ 3

Esamineremo degli strumenti più versatili, basati su `pandas`, nell'Unità 35, *Ordinamento e descrizione dei dati*.

Unità 8 - Manipolazione dei file

Un file è un contenitore non volatile per la memorizzazione a lungo termine dei dati. Una tipica operazione sui file prevede l'apertura di un file, la lettura di dati dal file o la scrittura di dati sul file e infine la chiusura del file. Potete aprire un file per attività di lettura (la modalità di default, denotata con `"r"`), di [sovra]scrittura (`"w"`) o di aggiunta (`"a"`). L'apertura di un file in scrittura distrugge senza alcun preavviso il precedente contenuto del file e l'apertura in lettura di un file inesistente provoca un'eccezione:

```
f = open(name, mode="r")
«lettura del file»
f.close()
```

Python offre un efficiente sostituto di questo paradigma: l'istruzione `with` consente di aprire esplicitamente un file, ma fa in modo che Python lo chiuda automaticamente dopo l'uso, evitandoci così di ricontrollare tutti i file che abbiamo lasciato aperti.

```
with open(name, mode="r") as f:
  «lettura del file»
```

Alcuni moduli, come `pickle` (di cui parleremo nell'Unità 12, *Pickling dei dati*), richiedono che un file venga aperto in modalità binaria (`"rb"`, `"wb"` o `"ab"`). Si dovrebbe usare la modalità binaria anche per leggere e scrivere su file gli array binari. Le seguenti funzioni leggono dati testuali da un file `f` precedentemente aperto:

```
f.read()       # Legge tutti i dati come una stringa o dati binari
f.read(n)      # Legge i primi n byte come una stringa o dati binari
f.readline()   # Legge la prossima riga come una stringa
f.readlines()  # Legge tutte le righe come una lista di stringhe
```

Potete utilizzare queste funzioni come volete e con l'ordine che ritenete necessario. Per esempio, potete leggere la prima stringa, poi cinque byte, poi una riga di testo e infine il resto del file. Il carattere di fine riga non viene rimosso dai risultati restituiti da queste funzioni. In genere, è meglio non usare le funzioni `read()` e `readlines()` se non siete più che sicuri che le dimensioni del file siano ragionevolmente compatte.

Le seguenti funzioni scrivono dati testuali su un file `f` precedentemente aperto:

```
f.write(line)       # Scrive una stringa o dei dati binari
f.writelines(lines) # Scrive una lista di stringhe
```

Queste funzioni non aggiungono un codice di fine riga alla fine delle stringhe scritte: dovrete pensarci voi.

Unità 9 - Andare sul Web

Secondo WorldWideWebSize (http://www.worldwidewebsize.com/), il Web contiene almeno 4,85 miliardi di pagine. Potreste essere interessati ad almeno alcune di esse. Il modulo urllib.request contiene tutte le funzioni necessarie per il download dei dati dal Web. Anche se sarebbe fattibile (ma sconsigliabile) scaricare un singolo dataset a mano, salvarlo in una directory locale e poi analizzarlo usando degli script Python, alcuni progetti di analisi dei dati consigliano l'impiego di download iterativi o ricorsivi automatizzati.

Il primo passo per scaricare qualcosa dal Web consiste nell'aprire l'URL con la funzione urlopen(url) e ottenere l'handle dell'URL. Dopo l'apertura, l'handle dell'URL è simile all'handle di un file di sola lettura: su di esso, per accedere ai dati, potete usare le funzioni read(), readline() e readlines().

A causa della natura dinamica del Web e di Internet, la probabilità di non poter aprire un URL è più elevata rispetto a quello che può accadere con un file su disco. Ricordate pertanto di racchiudere ogni chiamata a una funzione che operi sul Web in un'istruzione di gestione delle eccezioni:

```
import urllib.request
try:
   with urllib.request.urlopen("http://www.networksciencelab.com") as doc:
      html = doc.read()
      # Se la lettura ha successo, la connessione viene chiusa automaticamente
except:
   print("Could not open %s" % doc, file=sys.stderr)
   # Il documento potrebbe non essere accessibile
   # Inserire qui il gestore di errori
```

Se il dataset cui siete interessati viene fornito da un sito web che richiede una forma di autenticazione, urlopen() non funzionerà. Al suo posto si deve usare un modulo che offra la sicurezza SSL (*Secure Sockets Layer*), per esempio OpenSSL.

Il modulo urllib.parse fornisce tutti gli strumenti necessari per il parsing e l'unparsing (costruzione) di indirizzi URL. La funzione urlparse() suddivide un URL in una tupla di sei elementi: schema (per esempio http), indirizzo di rete, percorso nel file system, parametri, query e frammento:

```
import urllib.parse
URL = "http://networksciencelab.com/index.html;param?foo=bar#content"
urllib.parse.urlparse(URL)
```

→ ParseResult(scheme='http', netloc='networksciencelab.com', path='/index.html',
→ params='param', query='foo=bar', fragment='content')

La funzione urlunparse(parts) costruisce un URL valido utilizzando le parti restituite da urlparse(). Se eseguite il parsing di un URL e poi l'unparsing delle parti generate, il risultato può anche essere leggermente diverso dall'URL originale, ma quanto a funzionalità sarà perfettamente equivalente.

Unità 10 - Ricerche a pattern tramite espressioni regolari

Le espressioni regolari sono un potente meccanismo di ricerca, suddivisione e sostituzione di stringhe sulla base della corrispondenza con pattern, modelli. Il modulo `re` offre un linguaggio per la descrizione di pattern e una collezione di funzioni per la ricerca, la suddivisione e la sostituzione di stringhe.

Dal punto di vista di Python, un'espressione regolare è semplicemente una stringa contenente la descrizione di un pattern. Potete rendere ancora più efficiente la ricerca a pattern compilando un'espressione regolare che prevedete di usare più volte:

```
compiledPattern = re.compile(pattern, flags=0)
```

La compilazione migliora notevolmente i tempi di ricerca, senza incidere sulla correttezza. Se volete, potete anche specificare dei flag di controllo della ricerca del pattern, sia al momento della compilazione sia successivamente, al momento dell'esecuzione. I flag più comuni sono `re.I` (che ignora la distinzione fra caratteri maiuscoli e minuscoli) e `re.M` (che attiva la modalità multiriga e consente di usare gli operatori `~` e `$` per individuare rispettivamente l'inizio o la fine di una riga). Se volete combinare più flag, specificateli semplicemente uno dopo l'altro.

Il linguaggio delle espressioni regolari

La seguente tabella riassume, parzialmente, il linguaggio delle espressioni regolari.

Tabella 2.1 Il linguaggio delle espressioni regolari.

Operatori di base	
.	Qualsiasi carattere tranne fine riga (Newline).
a	Il carattere `a`.
ab	La stringa `ab`.
x|y	`x` oppure `y`.
\y	Escape di un carattere speciale `y`, come `^+{}$()[]\-?.*`
Classi di caratteri	

[a-d]	Uno dei caratteri a, b, c o d.
[^a-d]	Un carattere, tranne a, b, c o d.
\d	Una cifra.
\D	Una non cifra.
\s	Uno spazio.
\S	Un non spazio.
\w	Un carattere alfanumerico.
\W	Un carattere non-alfanumerico
Quantificatori	
x*	Zero o più x.
x+	Una o più x.
x?	Zero o una x.
x{2}	Esattamente due x.
x{2,5}	Fra due e cinque x.
Caratteri soggetti a escape	
\n	Newline.
\r	Carriage return.
\t	Tabulazione.
Posizioni	
^	Inizio della stringa.
\b	Limite della parola.
\B	Non limite della parola.
$	Fine della stringa.
Gruppi	
(x)	Cattura di un gruppo.
(?:x)	Non-cattura di un gruppo.

Gli operatori accento circonflesso (^) e trattino (-) quando si trovano dentro o alla fine dell'espressione di una classe di caratteri non hanno alcun particolare significato e rappresentano semplicemente i caratteri ^ e -. I gruppi cambiano l'ordine delle operazioni. Anche le sottostringhe che corrispondono ai gruppi di cattura vengono incluse nella lista dei risultati, se ciò è appropriato.

Notate che le espressioni regolari fanno ampio uso del simbolo backslash (\). Il backslash è un codice di escape in Python. Per poter essere trattato come un normale carattere, deve essere preceduto da un altro

backslash (\\), il che dà origine a espressioni regolari davvero complicate, con un numero sovrabbondante di backslash. Python supporta anche le stringhe grezze, dove i backslash non sono interpretati come caratteri di escape.

Per definire una stringa grezza, specificate il carattere r (da raw) immediatamente prima dell'apice di apertura. Le seguenti due stringhe sono uguali, e nessuna delle due contiene un codice di fine riga:

```
"\\n"
r" \n"
```

Normalmente, vi troverete a scrivere le espressioni regolari come stringhe grezze.

Ora è giunto il momento di provare a usare qualche espressione regolare utile. Lo scopo di questi esempi non è quello di spaventarvi inutilmente, ma di ricordarvi che se la programmazione è difficile, la ricerca di pattern può essere ancora più difficile.

```
r"\w[-\w\./*@\w[-\w/*(\.\w[-\w/*)+"
```

Un indirizzo di posta elettronica.

```
r"<TAG\b[~>/*<(.*?)</TAG>"
```

Un determinato tag HTML con il corrispondente tag di chiusura.

```
r"[-+/?((\d*\.?\d+)\(\d\.))([eE/[-+/?\d+)?"
```

Un numero in virgola mobile.

Si potrebbe pensare di scrivere un'espressione regolare corrispondente a un URL valido, ma questo è un compito davvero improbo. Resistete dunque alla tentazione di farlo e invece usate il modulo urllib.parse, di cui abbiamo parlato in precedenza nell'Unità 9.

Espressioni regolari… irregolari

Le espressioni regolari Python non sono le uniche espressioni regolari disponibili. Il linguaggio Perl impiega delle espressioni regolari con varie altre sintassi e talvolta anche con una semantica differente (ma la stessa potenza espressiva). In alcuni semplici casi (come l'individuazione di nomi di file), potete usare il modulo glob, di cui parleremo nell'Unità 11, *Globbing di nomi di file e altre stringhe*, che gestisce un ulteriore tipo di espressioni regolari.

Ricerca, suddivisione e sostituzione con il modulo re

Una volta scritta e compilata un'espressione regolare, potete usarla per la suddivisione, la ricerca e la sostituzione di sottostringhe. Il modulo `re` offre tutte le funzioni necessarie, e la maggior parte delle funzioni accetta pattern in due forme: grezzi e compilati.

```
re.function(rawPattern, ...)
compiledPattern.function(...)
```

La funzione `split(pattern, string, maxsplit=0, flags=0)` suddivide una stringa in al massimo `maxsplit` sottostringhe in base al `pattern` e restituisce la lista delle sottostringhe (se `maxsplit==0`, vengono restituite tutte le sottostringhe). Potete usarla, fra l'altro, come semplice strumento di riduzione in token per l'analisi delle parole:

```
re.split(r"\W", "Hello, world!")
```

→ ['Hello', '', 'world', '']

```
# Combina tutte le non-lettere adiacenti
re.split(r"\W+", "Hello, world!")
```

→ ['Hello', 'world', '']

La funzione `match(pattern, string, flags=0)` controlla se l'inizio di una stringa corrisponde all'espressione regolare. La funzione restituisce l'oggetto corrispondente o `None` se non viene trovata alcuna corrispondenza. L'oggetto corrispondente, se trovato, ha a disposizione le funzioni `start()`, `end()` e `group()` che restituiscono l'indice iniziale, l'indice finale del frammento e il frammento stesso.

```
mo = re.match(r"\d+", "067 Starts with a number")
```

→ <_sre.SRE_Match object; span=(0, 3), match='067'>

```
mo.group()
```

→ '067'

```
re.match(r"\d+", "Does not start with a number")
```

→ None

La funzione `search(pattern, string, flags=0)` controlla se una qualsiasi parte di una stringa corrisponde all'espressione regolare. La funzione restituisce l'oggetto corrispondente o `None` se non viene trovata alcuna corrispondenza. Usate questa funzione al posto di `match()` se il frammento da individuare non si trova all'inizio della stringa.

```
re.search(r"[a-z]+", "0010010 Has at least one 010 letter 0010010", re.I)
```

→ <_sre.SRE_Match object; span=(8, 11), match='Has'>

```
# Versione con distinzione maiuscole/minuscole
re.search(r"[a-z]+", "0010010 Has at least one 010 letter 0010010")
```

→ <_sre.SRE_Match object; span=(9, 11), match='as'>

La funzione `findall(pattern, string, flags=0)` trova tutte le sottostringhe che corrispondono all'espressione regolare. La funzione restituisce una lista di sottostringhe (lista che, naturalmente, può essere vuota).

```
re.findall(r"[a-z]+", "0010010 Has at least one 010 letter 0010010", re.I)
```

→ ['Has', 'at', 'least', 'one', 'letter']

Gruppi di cattura

Un gruppo senza cattura è semplicemente una parte di un'espressione regolare che `re` tratta come un singolo token. Le parentesi che racchiudono un gruppo senza cattura hanno la stessa funzione delle parentesi delle espressioni aritmetiche. Per esempio, `r"cab+"` individua una sottostringa che inizia con "`ca`", seguita da almeno una "`b`", mentre `r"c(?:ab)+"` individua una sottostringa che inizia con una "`c`", seguita da una o più "`ab`". Notate che non vi sono spazi fra "`(?:`" e il resto dell'espressione regolare.

Un gruppo di cattura, oltre a raggruppare, delinea la sottostringa restituita da `search()` o `findall()`: `r"c(ab)+"` descrive almeno una "`ab`" dopo una "`c`", ma restituisce solo le "`ab`".

La funzione `sub(pattern, repl, string, flags=0)` sostituisce tutte le parti corrispondenti non sovrapposte di una stringa con `repl`. Potete limitare il numero di sostituzioni impiegando il parametro opzionale `count`.

```
re.sub(r"[a-z ]+", "[...]", "0010010 has at least one 010 letter 0010010")
```

→ '0010010[...]010[...]0010010'

Le espressioni regolari sono ottimi strumenti, ma in molti casi (per esempio, quando occorre individuare un nome di file in base all'estensione) sono semplicemente fin troppo potenti, e potete ottenere risultati analoghi col *globbing*, di cui parleremo nella prossima Unità.

Unità 11 - Globbing di nomi di file e altre stringhe

Si dice *globbing* l'operazione che consiste nel ricercare nomi di file e altre stringhe, tramite espressioni regolari semplificate. Un carattere jolly può essere costituito dal segno * (che rappresenta zero o più caratteri) o dal segno ? (che rappresenta esattamente un carattere). Notate che \, + e . non sono codici speciali!

Il modulo glob offre una funzione per l'uso dei caratteri jolly. La funzione restituisce una lista di tutti i nomi di file individuati dal parametro passato:

```
gtob.gtob("*.txt")
```

→ ['public.policy.txt', 'big.data.txt']

Il carattere jolly * individua tutti i nomi di file della directory (cartella) corrente, tranne quelli che iniziano con un punto (.). Per individuare questi nomi speciali di file basta usare la sequenza .*.

Unità 12 - Pickling dei dati

Il modulo pickle implementa la *serializzazione*: il salvataggio di strutture di dati arbitrarie di Python in un file e la loro successiva lettura in un'espressione Python. Potete leggere un'espressione dal file tramite ogni programma Python, ma non con un programma scritto in un altro linguaggio (a meno che in tale linguaggio esista un'implementazione del protocollo pickle).

Il file pickle deve essere aperto (in lettura o scrittura) in modalità binaria:

```
# Salva un oggetto su un file
with open("myData.pickle", "wb") as oFile:
   pickle.dump(object, oFile)

# Ricarica lo stesso oggetto
with open("myData.pickle", "rb") as iFile:
   object = pickle.load(iFile)
```

Potete memorizzare in un file .pickle anche più di un oggetto. La funzione load() restituisce il successivo oggetto del file pickle o lancia un'eccezione nel caso in cui rilevi la fine del file. Potete anche usare il formato pickle per memorizzare i risultati intermedi che probabilmente dovranno essere elaborati da software dotato di accesso a pickle.

Esercitazioni

In questo capitolo, abbiamo imparato a estrarre dati dai file su disco e da Internet, a memorizzarli in appropriate strutture di dati, a estrarre elementi e frammenti che rispondono a determinati pattern e a codificarli per ogni successiva elaborazione. Non vi è nulla di infinito in programmazione, ma c'è un numero infinito di situazioni che richiedono l'estrazione dei dati, con un'ampia varietà di tipi, scopi e complessità. Ecco solo alcuni esempi.

Contatore della frequenza delle parole (*)

Scrivete un programma che scarichi una pagina web richiesta dall'utente e indichi quali sono le dieci parole usate più frequentemente. Il programma dovrà trattare tutte le parole senza distinguere fra maiuscole e minuscole. Per lo scopo di questo esercizio, immaginate che una parola sia descritta dall'espressione regolare `r"\w+"`.

Indicizzatore di file (**)

Scrivete un programma che indicizzi tutti i file di una specifica directory (cartella) indicata dall'utente. Il programma deve costruire un dizionario in cui le chiavi sono tutte le parole univoche di tutti i file (descritte dall'espressione regolare `r"\w+"`, trattando le parole senza distinguere fra lettere maiuscole e minuscole), e il valore di ogni voce è una lista di nomi di file che contengono tale la parola. Per esempio, se la parola `aloha` è menzionata nei file `early-internet.dat` e `hawaiian-travel.txt`, il dizionario dovrà contenere la voce `{..., 'aloha': ['early-internet.dat', 'hawaiian-travel.txt'],...}`.

Il programma poi dovrà salvare i risultati con pickle per eventuali utilizzi futuri.

Estrattore di numeri telefonici (***)

Scrivete un programma che estragga tutti i numeri telefonici da un determinato file di testo. Questo non è un compito semplice, in quanto vi

sono molte decine di convenzioni nazionali per comporre i numeri (vedi https://en.wikipedia.org/wiki/National_conventions_for_writing_telephone_numbers). Riuscite a progettare un'espressione regolare in grado di individuarli tutti?

E se questo compito vi è sembrato facile, cercate di estrarre anche tutti gli indirizzi stradali!

Elaborare i dati testuali

> E chi fu questo Giasone, e perché gli dei gli concedettero il loro favore? Da dove proveniva, e quale fu la sua storia?
> *Omero*, poeta greco

Spesso i dati grezzi provengono sotto forma di documenti testuali: documenti strutturati (file HTML, XML, CSV e JSON) o documenti non strutturati (semplice testo destinato alla lettura prettamente umana). Come si può facilmente immaginare, il testo non strutturato è forse la fonte di dati più difficile da elaborare, perché il software di elaborazione ha il compito di inferire il significato dei dati.

Tutti i tipi di dati appena menzionati sono adatti alla lettura umana (non per niente sono documenti testuali). Se necessario, possiamo aprire qualsiasi file di testo in un semplice editor di testi (il Blocco note in Windows, gedit in Linux, TextEdit su Mac OS) e leggerlo con i nostri occhi e correggerlo con le nostre mani. Se non sono disponibili altri strumenti, possiamo trattare tali documenti testuali come testi, indipendentemente dallo schema di rappresentazione utilizzato, ed esplorarli usando le funzioni Python dedicate alle stringhe (di cui parleremo nell'Unità 4, *Le funzioni che operano sulle stringhe*).

Fortunatamente, Anaconda fornisce molti moduli eccellenti (`BeautifulSoup`, `csv`, `json` e `nltk`) che trasformano il duro lavoro di analisi del testo in un'operazione quasi divertente. Seguendo il principio del Rasoio di Occam, "Non moltiplicare gli elementi più del necessario" (che in realtà fu formulata da John Punch, non da Guglielmo di Occam) eviteremo di reinventare degli strumenti già esistenti. Questo vale non solo per gli strumenti di elaborazione del testo, ma per qualsiasi pacchetto della distribuzione Anaconda.

Iniziamo a elaborare i dati testuali considerando il semplice caso dei dati strutturati. Scopriremo poi come aggiungere una certa struttura al testo non

strutturato, grazie a tecniche di elaborazione del linguaggio naturale.

Unità 13 - Elaborazione di file HTML

Il primo tipo di documento testuale strutturato di cui parleremo è costituito dai file HTML, un linguaggio a codici di formattazione comunemente usato nel Web per la rappresentazione delle informazioni in un formato di facile fruizione. Un documento HTML è costituito da testo e da tag predefiniti, racchiusi fra le parentesi angolari <>; questi ultimi controllano la presentazione e l'interpretazione del testo. I tag possono avere degli attributi. La seguente tabella mostra alcuni tag HTML e i loro attributi.

Tabella 3.1 Tag e attributi HTML fra i più utilizzati.

Tag	Attributi	Scopo
HTML		L'intero documento HTML.
HEAD		Testa del documento.
TITLE		Titolo del documento.
BODY	background, bgcolor	Corpo del documento,
H1, H2, H3 etc.		Titoli di sezione.
I, EM		Enfasi.
B, STRONG		Maggiore enfasi.
PRE		Testo preformattato.
P, SPAN, DIV		Paragrafo, gruppi, divisioni.
BR		Fine riga.
A	href	Collegamento ipertestuale.
IMG	src, width, height	Immagine.
TABLE	width, border	Tabella.
TR		Riga di una tabella.
TH, TD		Titoli di una tabella/dati di una cella.
OL, UL		Lista puntata/numerata.
LI		Elemento di una lista.
DL		Lista descrittiva.
DT, DD		Argomento della descrizione, definizione.
INPUT	name	Campo di input dall'utente.
SELECT	name	Menu.

HTML è un precursore di XML, il quale non è un linguaggio, ma piuttosto una famiglia di linguaggi a codici (markup) che hanno una struttura simile e sono destinati in primo luogo alla lettura automatica dei documenti. Noi utenti definiamo i tag XML e i loro attributi in base alle esigenze.

XML e HTML
Anche se i linguaggi XML e HTML possono essere simili, un tipico documento HTML, in generale, non sarà un documento XML valido, e un documento XML non è un documento HTML.
I tag XML sono specifici dell'applicazione. Qualsiasi stringa alfanumerica può essere un tag, sempre che segua alcune semplici regole (che siano racchiuse fra parentesi angolari e così via). I tag XML non controllano la presentazione del testo, ma solo la sua interpretazione. Il linguaggio XML viene frequentemente usato per realizzare documenti che non sono rivolti principalmente alla visione umana. Un altro linguaggio, XSLT (*eXtensible Stylesheet Language Transformation*), trasforma il codice XML in codice HTML, e un altro linguaggio ancora, CSS (*Cascading Style Sheets*), aggiunge uno stile ai documenti HTML.

Il modulo BeautifulSoup permette il parsing, l'accesso e la modifica di documenti HTML e XML. Potete creare un oggetto BeautifulSoup a partire da una stringa di codice, da un file contenente codici o un URL di un documento situato nel Web:

```
from bs4 import BeautifulSoup
from urllib.request import urlopen

# Costruisce soup1 da una stringa
soup1 = BeautifulSoup("<HTML><HEAD>«headers»</HEAD>«body»</HTML>")

# Costruisce soup2 da un file locale
soup2 = BeautifulSoup(open("myDoc.html"))

# Costruisce soup3 da un documento web
# Ricordate che urlopen() non aggiunge "http://"!
soup3 = BeautifulSoup(urlopen("http://www.networksciencelab.com/"))
```

Il secondo argomento opzionale del costruttore dell'oggetto è il parser del codice, un componente Python che si occupa di estrarre i tag e le entità HTML. BeautifulSoup è dotato di quattro parser preinstallati:

- "html.parser" (il default, molto veloce, non molto indulgente; usato per documenti HTML "semplici");
- "lxml" (molto veloce, indulgente);
- "xml" (solo per file XML);

- "html5lib" (molto lento, estremamente indulgente; usato per documenti HTML con una struttura complessa o per tutti i documenti HTML, se la velocità di parsing non rappresenta un problema).

Quando il soup è pronto, potete stampare il codice originario del documento usando la funzione soup.prettify().

La funzione soup.get_text() restituisce la parte testuale del documento, senza quindi tutti i tag. Potete usare questa funzione per convertire il codice in puro testo quando siete interessati soprattutto ai contenuti del testo.

```
htmlString = '''
   <HTML>
   <HEAD><TITLE>My document</TITLE></HEAD>
   <BODY>Main text.</BODY></HTML>
'''

soup = BeautifulSoup(htmtString)
soup.get_text()
```

→ '\nMy document\nMain text.\n'

Spesso i tag vengono usati per individuare determinati frammenti del file. Per esempio, potreste essere interessati alla prima riga della prima tabella. Il semplice testo non è molto utile per cercare di arrivarci, mentre i tag sì, in particolare se sono dotati di attributi class o id.

BeautifulSoup impiega un approccio coerente a tutte le relazioni verticali e orizzontali fra i tag. Le relazioni sono espresse come attributi degli oggetti tag e somigliano un po' alla gerarchia del file system. Il titolo, soup.title, è un attributo dell'oggetto soup. Il valore dell'oggetto name dell'elemento genitore (parent) del titolo (title) è soup.title.parent.name.string e la prima cella nella prima riga della prima tabella sarà probabilmente soup.body.table.tr.td.

Ogni tag t ha un nome t.name, un valore stringa (t.string con il contenuto originario e una lista di t.stripped_strings da cui sono rimossi gli spazi), un genitore t.parent, un tag successivo t.next e precedente t.prev e zero o più figli t.children (tag contenuti in altri tag).

BeautifulSoup offre l'accesso agli attributi del tag HTML attraverso un'interfaccia a dizionario Python. Se l'oggetto t rappresenta un collegamento ipertestuale (come in <a href="foobar.html">, allora il valore stringa della destinazione del collegamento ipertestuale è t["href"].string. Notate che i tag HTML non distinguono fra lettere minuscole e maiuscole.

Le funzioni soup forse più utili di tutte sono soup.find() e soup.find_all(), che trovano la prima istanza o tutte le istanze di un determinato tag. Ecco alcuni esempi di ricerche:

- Tutte le istanze del tag <H2>:

```
level2headers = soup.find_all("H2")
```

- Tutti i formati grassetti o corsivi:

```
formats = soup.find_all([ "i", "b", "em", "strong"])
```

- Tutti i tag che hanno un determinato attributo (per esempio, id="Nnk3"):

```
soup.find(id="link3")
```

- Tutti i collegamenti ipertestuali e anche l'indirizzo URL di destinazione del primo link, usando la notazione a dizionario o la funzione tag.get():

```
links = soup.find_all("a")
firstLink = links[0]["href"]
# oppure
firstLink = links[0].get("href")
```

A proposito, entrambe le espressioni dell'ultimo esempio non funzionano se l'attributo è assente. Occorre usare la funzione tag.has_attr() per verificare la presenza di un attributo prima di estrarlo. La seguente espressione combina BeautifulSoup e la manipolazione delle liste per estrarre tutti i link e i rispettivi URL ed etichette (utile per l'attraversamento ricorsivo del Web):

```
with urlopen("http://www.networksciencelab.com/") as doc:
   soup = BeautifulSoup(doc)

links = [(link.string, link[ "href" ])
   for link in soup.find_all("a")
   if link.has_attr( "href")]
```

Il valore di links è una lista di tuple:

```
→ [('Network Science Workshop',
→ 'http://www.slideshare.net/DmitryZinoviev/workshop-20212296'),
→ «...»,('Academia.edu',
→ 'https://suffolk.academia.edu/DmitryZinoviev'), ('ResearchGate',
→ 'https://www.researchgate.net/profile/Dmitry_Zinoviev')]
```

La versatilità del formato HTML/XML ha dei punti di forza, ma proprio questa versatilità può essere anche un problema, in particolare quando si ha a che fare con dati tabulari. Fortunatamente, potete memorizzare i dati tabulari all'interno di file CSV, un formato più rigido ma anche più facile da manipolare, che sarà l'argomento della prossima Unità.

Unità 14 - Manipolazione dei file CSV

CSV è un formato per file di testo strutturati utilizzato per memorizzare e trasferire dati tabulari (o prevalentemente tabulari). Risale addirittura al 1972 ed è uno dei formati disponibili in Microsoft Excel, Apache OpenOffice Calc e altri software per fogli di lavoro. Data.gov (https://catalog.data.gov/dataset?res_format=CSV), un sito web governativo statunitense che consente di accedere a molti dati pubblici, offre ben 12.550 dataset in formato CSV.

Un file CSV è costituito da colonne, che rappresentano le variabili, e righe, che rappresentano i record (ma gli esperti di scienza dei dati dotati di basi statistiche spesso usano per i record il termine "osservazioni"). Normalmente i campi di un record sono separati da una virgola, ma in alcuni casi possono essere utilizzati anche altri delimitatori, come le tabulazioni (TSV - *Tab Separated Values*), i due punti, i punti e virgola e le barre verticali. Se dovete preparare un file CSV, utilizzate le virgole, ma preparatevi a trovare anche altri separatori nei file prodotti da coloro che non seguono questa convenzione.

Tenete anche presente che, talvolta, quello che sembra un delimitatore non lo è affatto. Per consentire l'uso del carattere delimitatore in un campo, ovvero come parte del valore della variabile (come in ... ,"Hello, world", ...), racchiudete i campi fra apici.

Per comodità, il modulo Python csv offre un reader e un writer CSV. Entrambi gli oggetti prendono come primo parametro l'handle di un file precedentemente aperto (nell'esempio, il file viene aperto con l'opzione newline='' per evitare di dover gestire le righe). Potete fornire il carattere delimitatore e quello da usare come apice, se necessario, attraverso i parametri opzionali delimiter e quotechar. Altri parametri opzionali controllano il simbolo di escape, il codice di fine riga e così via.

```
with open("nomefile.csv", newline='') as infile:
  reader = csv.reader(infile, delimiter=',', quotechar='"')
```

Il primo record di un file CSV spesso contiene le intestazioni delle colonne e può quindi dover essere trattato in modo differente rispetto al

resto del file. Questa non è una caratteristica tipica del formato CSV, ma semplicemente una pratica comune.

Il reader CSV offre un'interfaccia di iterazione da usare in un ciclo for. L'iteratore restituisce il record successivo come una lista di campi stringa. Il lettore non converte i campi in dati numerici (questo è compito vostro!) e non elimina gli spazi in eccesso, a meno che ciò venga richiesto esplicitamente tramite il parametro opzionale skipinitialspace=True.

Se le dimensioni del file CSV non sono note e sono potenzialmente estese, è meglio non leggere tutti i record in una sola volta. Piuttosto, è meglio adottare un'elaborazione incrementale, iterativa, riga-per-riga: leggere una riga, elaborare la riga, mettere da parte la riga e così via con le successive.

Il writer CSV offre le funzioni writerow() e writerows(). La prima scrive sul file una sequenza di stringhe o numeri a formare un record. I numeri vengono convertiti in stringhe, e questa è sicuramente una preoccupazione in meno. Analogamente, writerows() scrive sul file una lista di sequenze di stringhe o numeri, a formare una collezione di record.

Nel seguente esempio, useremo il modulo csv per estrarre da un file CSV la colonna Answer.Age. Immagineremo che l'indice della colonna sia ignoto, ma che la colonna esista. Una volta ottenuti i numeri, calcoleremo la media e la deviazione standard della variabile age, con un po' di aiuto del modulo statistics.

Innanzitutto, dobbiamo aprire il file e leggere i dati:

```
with open("demographics.csv", newline='') as infile:
    data = list(csv.reader(infile))
```

Esaminiamo data[0], il primo record del file. Dovrebbe contenere l'intestazione di colonna che ci interessa:

```
ageIndex = data[0].index("Answer.Age")
```

Infine, accediamo al campo in questione dei record rimanenti, poi calcoliamo e visualizziamo i valori statistici:

```
ages = [int(row[ageIndex]) for row in data[1:]]
print(statistics.mean(ages), statistics.stdev(ages))
```

I moduli csv e statistics sono strumenti di basso livello e "tecnici". Più avanti, nel Capitolo 6, *Manipolare serie di dati e frame*, vedremo come usare i

frame di dati pandas per un progetto che va ben oltre la semplice esplorazione di alcune colonne.

Unità 15 - Leggere i file JSON

JSON è un formato "leggero" per l'interscambio di dati. A differenza di pickle (di cui abbiamo parlato in precedenza nell'Unità 12), JSON è un formato indipendente dal linguaggio, ma più limitato in termini di rappresentazione dei dati.

NOTA

Molti siti web fra i più noti, come Twitter (https://dev.twitter.com/docs), Facebook (https://developers.facebook.com) e Yahoo! Weather (https://developer.yahoo.com/weather/) forniscono delle interfacce API che usano JSON quale formato di interscambio dei dati.

JSON supporta i seguenti tipi di dati.

- Tipi di dati atomici - stringhe, numeri, true, false, null.
- Array: un array corrisponde a una lista Python; è racchiuso fra parentesi quadre [ ]; gli elementi di un array non devono necessariamente essere dello stesso tipo di dati:[1, 3.14, "a string", true, null]
- Oggetti: un oggetto corrisponde a un dizionario Python; è racchiuso fra parentesi graffe {}; ogni elemento è costituito da una chiave e un valore, separati da un segno di due punti:{"age" : 37, "gender" : "male", "married" : true}
- Qualsiasi combinazione ricorsiva di array, oggetti e tipi di dati atomici (array di oggetti, oggetti i cui valori sono array e così via).

Sfortunatamente, alcuni tipi di dati e strutture Python, come gli insiemi e i numeri complessi, non possono essere memorizzati all'interno di file JSON. Pertanto, prima di esportarli su un file JSON, dovete convertirli in tipi di dati rappresentabili. Potete memorizzare un numero complesso come un array di due numeri e un insieme come un array di elementi.

La memorizzazione di dati complessi su un file JSON si chiama serializzazione. L'operazione opposta si chiama deserializzazione. Python gestisce la serializzazione e deserializzazione JSON tramite le funzioni nel modulo json.

La funzione dump() esporta un oggetto Python rappresentabile su un file di testo precedentemente aperto. La funzione dumps() esporta un oggetto Python rappresentabile su una stringa di testo (per attività di stampa o di

comunicazioni fra processi). Entrambe le funzioni svolgono una serializzazione.

Il valore del pickling
Quando si salvano dei dati su un file JSON, vengono salvati i valori delle variabili; una volta ricaricati, i valori divengono indipendenti. Quando si esegue il pickling di quegli stessi dati, vengono salvati i riferimenti alle variabili originali. Una volta ricaricati, tutti i riferimenti alla stessa variabile continuano a far riferimento alla stessa variabile.

La funzione `loads()` converte una stringa JSON valida in un oggetto Python (in pratica "carica" l'oggetto in Python). Questa conversione è sempre possibile. Analogamente, la funzione `load()` converte il contenuto di un file di testo precedentemente aperto in un oggetto Python. È un errore memorizzare più di un oggetto in un file JSON, ma un file contiene già più di un oggetto, potete leggerlo come contenuto testuale, convertire il testo in un array di oggetti (aggiungendo delle parentesi quadre attorno al testo e delle virgole di separazione fra i singoli oggetti) e usare `loads()` per deserializzare il testo in una lista di oggetti.

Il seguente frammento di codice sottopone un oggetto arbitrario (ma serializzabile) a una sequenza di serializzazioni e deserializzazioni:

```
object = «un oggetto serializzabile»

# Salva un oggetto su un file
with open("data.json", 'V") as out_json:
   json.dump(object, out_json, indent=None, sort_keys=False)

# Carica un oggetto da un file
with open("data.json") as in_json:
   object1 = json.load(in_json)

# Serializza un oggetto su una stringa
json_string = json.dumps(object1)

# Esegue il parsing di una stringa in JSON
object2 = json.loads(json_string)
```

Ecco fatto! Nonostante abbiano attraversato ben quattro conversioni, `object`, `object1` e `object2` hanno ancora lo stesso valore.

In genere, si usa la rappresentazione JSON per memorizzare i risultati finali di un'elaborazione quando si sa che i risultati possono dover essere ulteriormente elaborati o importati da un altro programma.

Unità 16 - Elaborazione di testi in linguaggio naturale

Come regola empirica, qualcosa come l'80 percento di tutti i dati potenzialmente usabili non ha una natura strutturata, comprendendo dati audio, video, immagini (argomenti che non rientrano negli scopi di questo libro) e testuali, scritti in un linguaggio naturale (http://www.informationweek.com/software/information-management/structure-models-and-meaning/d/d-id/1030187). Un testo scritto in un linguaggio naturale non ha tag, non ha delimitatori e non ha tipi di dati, ma può comunque essere una ricchissima fonte di informazioni. Possiamo voler sapere se (e con quale frequenza) determinate parole vengono usate nel testo (riduzione in token, o *tokenizzazione*, di parole e frasi), di quale tipo di testo si tratta (*classificazione del testo*), se il testo contiene un messaggio positivo o negativo (analisi del sentiment), chi o cosa viene menzionato nel testo (*estrazione delle entità*) e così via. Possiamo anche leggere e valutare un testo o due con i nostri occhi, ma per svolgere un'analisi a tappeto del testo occorre ricorrere a un'automazione dell'elaborazione del linguaggio naturale (NLP - *Natural Language Processing*).

Molte funzionalità di elaborazione del linguaggio naturale sono implementate nel modulo Python `nltk` (*Natural Language Toolkit*). Il modulo è organizzato in base alle seguenti suddivisioni: corpus (collezioni di parole ed espressioni), funzioni e algoritmi.

Corpus NLTK

Un corpus è una collezione strutturata o non strutturata di parole o espressioni. Tutti i corpus NLTK sono conservati nel modulo `nltk.corpus`. Ecco alcuni esempi.

- `gutenberg`: contiene diciotto testi in inglese del Progetto Gutenberg, come Moby Dick e la Bibbia.
- `names`: una lista di 8000 nomi maschili e femminili.

- words: una lista delle 235.000 parole e forme più usate in inglese.
- stopwords: una lista delle parole ininfluenti più usate in quattordici lingue. La lista per l'inglese è in stopwords.words("english"). Tali parole vengono normalmente eliminate dal testo per la maggior parte dei tipi di analisi, perché in genere non aggiungono nulla alla comprensibilità dei testi.
- cmudict: un dizionario delle pronunce composto alla Carnegie Mellon University (da cui il nome), con oltre 134.000 voci. Ogni voce di cmudict.entries() è una tupla di una parola e una lista di sillabe. La stessa parola può avere più pronunce differenti. Potete usare questo corpus per identificare le omofonie (parole che hanno pressoché lo stesso suono).

L'oggetto nltk.corpus.wordnet è un'interfaccia che rimanda a un altro corpus, la rete semantica online WordNet (per la quale è richiesto l'accesso a Internet). La rete è una collezione di synset (synonym set), parole contrassegnate con un indicatore "parte del discorso" e un numero sequenziale:

```
wn = nltk.corpus.wordnet # Il reader del corpus
wn.synsets("cat")
```

→ [Synset('cat.n.01'), Synset('guy.n.01'), «altri synset»]

Per ogni synset, potete cercare la definizione, che può anche essere piuttosto sorprendente:

```
wn.synset("cat.n.01").definition()
wn.synset("cat.n.Q2").definition()
```

→ 'feline mammal normalmente having thick soft fur «...»'
→ 'an informal term for a youth o man'

Un synset può avere hypernyms (iperonimi, synset meno specifici) e hyponyms (iponimi, synset più specifici), e in pratica i synset hanno l'aspetto di classi OOP, con le loro sottoclassi e superclassi.

```
wn.synset("cat.n.01").hypernyms()
wn.synset("cat.n.01").hyponyms()
```

→ [Synset('feline.n.01')]
→ [Synset('domestic_cat.n.01'), Synset('wildcat.n.03')]

Infine, potete usare WordNet per calcolare la similarità semantica fra due synset. La similarità è un numero in doppia precisione compreso nell'intervallo [0...1]. Se la similarità è pari a 0, i synset non sono correlati; se è pari a 1, i synset sono del tutto sinonimi.

```
x = wn.synset("cat.n.01")
y = wn.synset("lynx.n.01")
x.path_similarity(y)
```

→ 0.04

Pertanto, quanto sono simili due parole arbitrarie? Cerchiamo tutti i synset per "dog" e "cat" e troviamo le definizioni più semanticamente vicine:

```
[simxy.definition() for simxy in max(
   (x.path_similarity(y), x, y)
      for x in wn.synsets('cat')
      for y in wn.synsets('dog')
      if x.path_similarity(y)   # Cerca la similarità dei synset
   )[1:]]
```

→ ['an informal term for a youth o man', 'informal term for a man']

Oltre a usare i corpus standard, potete anche creare i vostri corpus tramite PlaintextCorpusReader. Il reader considera i file nella directory root che corrispondono al pattern glob.

```
myCorpus = nltk.corpus.PlaintextCorpusReader(root, glob)
```

La funzione fileids() restituisce la lista di file inclusi nel nuovo corpus. La funzione raw() restituisce il testo "grezzo" originale del corpus. La funzione sents() restituisce la lista di tutte le frasi. La funzione words() restituisce la lista di tutte le parole. Nel prossimo paragrafo vedrete come funziona la conversione del testo grezzo in frasi e parole.

```
myCorpus.fileids()
myCorpus.raw()
myCorpus.sents()
myCorpus.words()
```

L'ultima di queste funzioni, insieme a un oggetto Counter (descritto nell'Unità 7) consente di calcolare la frequenza delle parole e identificare le parole impiegate più frequentemente.

ATTENZIONE

Quando installate il modulo NLTK, in realtà installate solo le classi, no i corpus. I corpus sono considerati troppo estesi per poter essere inclusi nella distribuzione. Quando importate il modulo per la prima volta, ricordatevi di richiamare la funzione download() (è necessaria la connessione a Internet) e di installare le parti mancanti, a seconda delle vostre esigenze.

Normalizzazione

La normalizzazione è quella procedura che prepara un testo in linguaggio naturale per le successive elaborazioni. Normalmente prevede i seguenti passi (più o meno in questo ordine).

1. Riduzione in token (tokenizzazione, suddivisione del testo in parole). NLTK offre due tokenizzatori semplici e due più avanzati. Il tokenizzatore di frasi restituisce una lista di frasi, sotto forma di stringhe. Tutti gli altri tokenizzatori restituiscono una lista di parole:
   - word_tokenize(text): tokenizzatore di parole;
   - sent_tokenize(text): tokenizzatore di frasi;
   - regexp_tokenize(text, re): tokenizzatore a espressioni regolari; il parametro re è un'espressione regolare che descrive cosa deve essere considerato "una parola valida".

   A seconda della qualità del tokenizer e della struttura della frase, alcune "parole" potrebbero contenere caratteri non alfabetici. Per le attività che dipendono pesantemente dall'analisi della punteggiatura, come l'analisi del sentiment attraverso le emoticon, occorre impiegare il più avanzato WordPunctTokenizer. Ecco come WordPunctTokenizer.tokenize() e word_tokenize() eseguono il parsing dello stesso testo:

```
from nltk.tokenize import WordPunctTokenizer
word_punct = WordPunctTokenizer()
text = "}Help! :))) :[ ..... :D{"
word_punct.tokenize(text)
```

   → **["}", "Help", "!", ":)))", ":[", ".....", ":", "D", "{"]**

```
nltk.word_tokenize(text)
```

   → **["}", "Help", "!", ":", ")", ")", ")", ":", "[", "...", "..", ":", "D", "{"]**
2. Conversione delle parole in maiuscole o minuscole.
3. Eliminazione delle parole ininfluenti. Usate come riferimento il corpus stopwords e delle liste di parole ininfluenti specifiche dell'applicazione. Ricordate che in stopwords le parole sono in lettere minuscole. In altre parole, nel corpus stopwords non troverete l'articolo inglese "THE" (che è assolutamente una parola ininfluente).
4. Stemming (conversione delle parole ai loro radicali). NLTK fornisce due stemmer: lo stemmer di Porter (meno aggressivo) e lo stemmer di Lancaster (più aggressivo). A causa delle sue regole più aggressive, lo

stemmer di Lancaster produce più radicali omonimi. Entrambi gli stemmer sono dotati della funzione `stem(word)`, che restituisce la radice di `word`:

```
pstemmer = nltk.PorterStemmer()
pstemmer.stem("wonderful")
```

→ **'wonder'**

```
lstemmer = nltk.LancasterStemmer()
lstemmer.stem("wonderful")
```

→ **'wond'**

Applicate questi stemmer solo a singole parole, non a intere frasi.

5. Trasformazione in lemmi. Un meccanismo di riduzione in radicali più lento e conservativo. `WordNetLemmatizer` ricerca i radicali calcolati in WordNet e li accetta solo se esistono sotto forma di parole o formule (per usarlo è necessaria una connessione a Internet). La funzione `lemmatize(word)` restituisce il lemma di `word`.

```
lemmatizer = nltk.WordNetLemmatizer()
lemmatizer.lemmatize("wonderful")
```

→ **'wonderful'**

Anche se tecnicamente non rientra nella sequenza di normalizzazione, il tagging POS (*Part Of Speech*) è un passo importante nella pre-elaborazione dei testi. La funzione `nltk.pos_tag(text)` assegna un tag POS a ogni parola del testo, fornito sotto forma di lista di parole. Il valore restituito è una lista di tuple, in cui il primo elemento di una tupla è la parola originaria e il secondo elemento è un tag.

```
nltk.pos_tag(["beautiful", "world"])
# Un aggettivo (JJ) e un nome (NN)
```

→ **[('beautiful', 'JJ'), ('world', 'NN')]**

Per riepilogare, visualizziamo i dieci radicali di parole (non ininfluenti) più usati nel file `index.html` (notate l'uso di `BeautifulSoup`, precedentemente introdotto nell'Unità 13).

```
from bs4 import BeautifulSoup
from collections import Counter
from nltk.corpus import stopwords
from nltk import LancasterStemmer

# Crea un nuovo stemmer
ls = nltk.LancasterStemmer()

# Legge il file e crea soup
with open("index.html") as infile:
    soup = BeautifulSoup(infile)
```

```
# Estrae e riduce in token il testo
words = nltk.word_tokenize(soup.text)

# Converte il testo in minuscole
words = [w.lower() for w in words]

# Elimina le parole ininfluenti e risale al radicale del resto delle parole
words = [ls.stem(w)
        for w in text
           if w not in stopwords.words("english") and w.isalnum()
       ]

# Presenta le parole
freqs = Counter(words)

print(freqs.most_common(10))
```

Potete considerare questo frammento di codice come il primo passo verso l'estrazione dell'argomento.

Altre procedure di elaborazione del testo

Una discussione approfondita sulle procedure di elaborazione del linguaggio naturale non rientra negli scopi di questo libro, ma ecco un breve riepilogo delle opzioni più interessanti.

- Segmentazione: il riconoscimento delle estremità di una "parola" in un testo che non ha limiti sintattici fra le parole (in un testo cinese, per esempio). Potete applicare la segmentazione a ogni sequenza di caratteri o numerica (per esempio, a una sequenza di acquisti o di frammenti di DNA).
- Classificazione del testo: assegnare un testo a una categoria preimpostata, sulla base di criteri predefiniti. Un caso particolare di classificazione del testo è l'analisi del sentiment, che tipicamente si svolge sulla base dell'analisi della frequenza delle parole a elevato carico emotivo.
- Estrazione delle entità: rilevamento di quelle parole o frasi contenenti determinate proprietà predeterminate, che normalmente fanno riferimento a entità, come persone, luoghi geografici, società, prodotti e marchi.
- Indicizzazione semantica latente: identificazione dei pattern nelle relazioni fra i termini e i concetti contenuti all'interno di una collezione di testi non strutturati, usando tecniche SVD (*Singular*

Value Decomposition). A questo proposito, in statistica la SVD è chiamata anche analisi del componente principale (PCA).

Esercitazioni

A questo punto sapete come estrarre dati utili da un file HTML, XML, CSV o JSON e perfino da un comune file di testo. Sapete cosa sono i tag HTML e XML e la loro struttura e siete in grado di distinguere i tag dai dati e di normalizzare le parole (almeno fino a un certo punto). Sono moltissimi i progetti interessanti che richiedono proprio questo (e anche un po' di pazienza). Cominciamo a fare pratica!

Rilevatore di collegamenti ipertestuali non funzionanti (*)

Scrivete un programma che, dato l'indirizzo URL di una pagina web, indichi i nomi e le destinazioni dei collegamenti ipertestuali non funzionanti presenti nella pagina. In questo esercizio, un collegamento ipertestuale non funziona se un tentativo di aprirlo con `urllib.request.urlopen()` fallisce.

Wikipedia Miner (**)

MediaWiki (un progetto Wikimedia https://www.mediawiki.org/wiki/MediaWiki) offre un'API basata sul formato JSON che permette un accesso sistematico ai dati e metadati di Wikipedia. Scrivete un programma che elenchi i radicali più utilizzati presenti nella pagina [inglese] "Data Science" di Wikipedia.

Suggerimenti per l'implementazione:

- Usate HTTP, non HTTPS.
- Consultate l'esempio sul sito MediaWiki e partite da lì per realizzare questo programma.
- Innanzitutto, estraete l'ID della pagina in base al titolo, poi ottenete la pagina in base al suo ID.
- Esplorate in modo visuale i dati JSON, e in particolare le chiavi, ai diversi livelli gerarchici: al momento della realizzazione di queste pagine, la risposta prevede sei livelli di profondità!

Classificatore del genere musicale (***)

Scrivete un programma che utilizzi Wikipedia per calcolare la similarità semantica fra i diversi generi di musica rock/pop. Iniziate elencando i grandi gruppi (https://en.wikipedia.org/wiki/Category:Rock_music_groups_by_genre) (notate che la lista è gerarchica e contiene varie subcategorie). Elaborate ricorsivamente la lista e i suoi discendenti fino a esaurire i gruppi principali (potete anche limitare l'esplorazione solo a determinate subcategorie, i gruppi rock inglesi, per risparmiare tempo e traffico).

Per ogni gruppo individuato, estraete, se possibile, i generi. Usate come misura della similarità semantica l'indice di Jaccard (https://en.wikipedia.org/wiki/Jaccard_index): per ogni coppia di generi A e B, vale che

$J(A,B) = |A \cap B| / |A \cup B| = |C| / (|A| + |B| - |C|)$

dove |A| e |B| sono il numero di gruppi che elencano, rispettivamente, A o B come loro genere e |C| è il numero di gruppi che elencano sia A sia B. Salvate (*pickle*) i risultati per eventuali utilizzi futuri: probabilmente non vorrete eseguire ripetutamente questo programma!

A proposito, quanti generi esistono e quali sono i generi maggiormente correlati?

Utilizzare i database

> Quali sono le dee?... La decima è Vor, talmente saggia e abile nelle ricerche che nulla le può essere celato.
> *Snorri Sturluson*, storico, poeta e politico islandese

Allora… stiamo studiando (o utilizzando) la scienza dei *dati* (OK!). Sappiamo importare dati grezzi da file su disco, collocandolo in strutture di *dati* Python (OK!). Manca un elemento per fare un terzetto, e tale elemento è l'argomento di questo capitolo: un *database* è il luogo in cui i dati possono essere memorizzati nel lungo periodo.

I database sono componenti importanti nella catena dell'analisi dei dati.

- I dati di input spesso vengono forniti sotto forma di tabelle di database. Occorre pertanto estrarli dal database per tutte le successive elaborazioni.
- I database rappresentano aree di memorizzazione ottimizzate, veloci e non volatili che potete usare per memorizzare i dati grezzi e intermedi e i risultati finali, anche qualora i dati grezzi non fossero stati originariamente memorizzati in un database.
- I database offrono funzionalità ottimizzate di trasformazione dei dati: ordinamento, selezione, unione. Se i dati grezzi o i risultati intermedi si trovano già in un database, potete usare tale database non solo per la memorizzazione, ma anche per l'aggregazione.

In questo capitolo, vedremo come predisporre, configurare, popolare e interrogare database MySQL e MongoDB, uno dei database relazionali più utilizzati e il più noto archivio di documenti (o database NoSQL). Anche se, molto probabilmente, vi troverete a usare questi database preconfigurati e già pre-popolati di dati, ma la conoscenza del funzionamento interno dell'engine di un database avrà l'effetto non solo di migliorare le vostre

competenze come programmatori, ma rappresenterà anche una solida base per l'introduzione di pandas (Capitolo 6).

Unità 17 - Configurazione di un database MySQL

Un database relazionale è una collezione di tabelle ordinate e indicizzate, memorizzate in modo permanente. I database relazionali sono strumenti eccellenti per la memorizzazione di dati tabulari (come i dati presenti nei file CSV), dove una tabella rappresenta un tipo di variabile, le colonne della tabella rappresentano le variabili e le righe rappresentano le osservazioni, o i record.

Non è necessario usare Python per manipolare un database; bisogna però conoscere il linguaggio SQL (*Structured Query Language*) o una sua specifica implementazione, come MySQL, per poter accedere a un database relazionale, dalla sua riga di comando o da un'applicazione Python.

Per interagire dalla riga di comando con un server MySQL in funzione, occorre impiegare un client MySQL, come `mysql`. Tutti i comandi MySQL possono usare indifferentemente lettere maiuscole e minuscole e devono terminare con un punto e virgola.

Per dare avvio a un nuovo progetto che usi un database, usate `mysql` quale amministratore del database (queste operazioni devono essere eseguite una sola volta):

1. Avviate `mysql` dalla riga di comando della shell:

   ```
   c:\myProject> mysql -u root -p
   Enter password:
   Welcome to the MySQL monitor. Commands end with ; or \g.
   «Altro output mysql»
   mysql>
   ```

   Specificate tutte le successive istruzioni al prompt `mysql`.
2. Create un nuovo utente per il database ("`dsuser`") e la relativa password ("`badpassw0rd`"):

   ```
   CREATE USER 'dsuser'@'localhost' IDENTIFIED BY 'badpasswQrd';
   ```
3. Create un nuovo database per il progetto ("dsdb"):

   ```
   CREATE DATABASE dsdb;
   ```
4. Concedete al nuovo utente l'accesso al nuovo database:

```
GRANT ALL ON dsdb.* TO 'dsuser'@'localhost';
```

Ora è giunto il tempo di creare una nuova tabella nel database. Usate lo stesso client mysql, ma questa volta connettetevi come un normale utente del database:

```
c:\myProject> mysql -u dsuser -p dsdb
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
«Altro output mysql»
mysql>
```

In genere una tabella, una volta creata, viene utilizzata più volte. Potete anche modificare, in un secondo tempo, le proprietà di una tabella in base alle esigenze del progetto. Per creare una nuova tabella si usa il comando CREATE TABLE, seguito dal nome della nuova tabella e da una lista di colonne. Per ogni colonna occorre definire il nome e il tipo di dati (in questo ordine). I tipi di dati più comuni in MySQL sono TINYINT, SMALLINT, INT, FLOAT, DOUBLE, CHAR, VARCHAR, TINYTEXT, TEXT, DATE, TIME, DATETIME e TIMESTAMP.

Il seguente comando crea la tabella employee con le colonne empname (testo di lunghezza variabile), salary (numero in virgola mobile) e hired (data). Ogni record della tabella descrive un dipendente.

```
USE dsdb;
CREATE TABLE employee (empname TINYTEXT, salary FLOAT, hired DATE);
```

→ Query OK, 0 rows affected (0.17 sec)

Quando una tabella non è più necessaria, la si può eliminare dal database.

```
DROP TABLE employee;
```

→ Query OK, 0 rows affected (0.05 sec)

Il comando DROP è compatto, elegante e del tutto irreversibile, come il proverbiale latte versato, sul quale, notoriamente, non si piange. Quindi pensateci due o tre volte, prima di lanciare un DROP!

Schema del database

Lo *schema di un database* è la struttura che descrive tutte le tabelle, le colonne, i tipi di dati, gli indici, i vincoli e le relazioni fra le tabelle. Lo schema è un po' come il "buco" in mezzo alla "ciambella": è ciò che rimane del database dopo aver cancellato tutti i dati da tutte le tabelle.

Anche se ciò non è richiesto dallo standard del linguaggio, si dovrebbe sempre aggiungere a ogni record una chiave primaria auto-generata e un

timestamp (un'indicazione oraria) ad aggiornamento automatico dell'ultima modifica apportata (sempre se lo spazio lo consente). La chiave primaria consente di identificare ogni record in modo univoco e accelera le ricerche. Il timestamp dell'ultima modifica aggiunge un'informazione cronologica; la parola riservata NOT NULL conferma che le colonne contrassegnate contengano valori sensati:

```
CREATE TABLE employee (id INT PRIMARY KEY AUTO_INCREMENT,
  updated TIMESTAMP, empname TINYTEXT NOT NULL, salary FLOAT NOT NULL, hired DATE);
```

Se pensate di usare una colonna (una variabile) per attività di ordinamento, ricerca o unione, aggiungete un indice anche a tale colonna:

```
ALTER TABLE employee ADD INDEX(hired);
```

→ Query OK, 0 rows affected (0.22 sec)
→ Records: 0 Duplicates: 0 Warnings: 0

Vale la pena di notare che gli indici migliorano drasticamente i tempi di risposta, ma peggiorano, anche sensibilmente, i tempi di inserimento e cancellazione. Gli indici dovrebbero essere creati solo dopo l'inserimento dei dati nella tabella. Se dovete inserire un nuovo lotto di dati, prima eliminate gli indici esistenti:

```
DROP INDEX hired ON employee;
```

A questo punto potete inserire i nuovi dati e ri-aggiungere gli indici.

Se tutti i valori di una colonna sono univoci (come i codici dei dipendenti o i numeri di parti), aggiungete alla colonna il vincolo UNIQUE. Se il tipo di dati della colonna ha una lunghezza variabile (come VARCHAR, TINYTEXT o TEXT), dovete specificare quanta parte di ogni voce, nella colonna, deve essere univoca:

```
ALTER TABLE employee ADD UNIQUE(empname(255));
```

La chiave primaria deve sempre avere un valore (è NOT NULL), è sempre un indice (INDEX) ed è univoca (UNIQUE).

Unità 18 - Uso di un database MySQL: riga di comando

MySQL supporta cinque operazioni fondamentali: inserimento, cancellazione, modifica, selezione e unione. Esse vengono usate per popolare le tabelle del database e per modificare ed estrarre i dati in esse contenuti. Queste operazioni vengono normalmente svolte dal programma di analisi dei dati, ma per comprenderne il senso, proveremo innanzitutto a utilizzarle dal prompt dei comandi mysql.

Inserimento

Partiamo dall'inizio. Inseriremo un nuovo record in una tabella, poi un altro e un altro ancora, finché la tabella non contiene tutte le osservazioni:

```
INSERT INTO employee VALUES(NULL,NULL,"John Smith",35000,NOW());
```

→ Query OK, 1 row affected, 1 warning (0.18 sec)

I primi due NULL sono semplici valori segnaposto per l'indice e il timestamp. Il server li calcola automaticamente. La funzione NOW() restituisce la data e l'ora correnti, ma per popolare il record viene usata solo la data. Notate che la query ha prodotto un warning, e il motivo è proprio questo troncamento. Osserviamo le descrizioni verbali e i codici degli ultimi warning ed errori:

```
SHOW WARNINGS;
```

```
→ +-------+------+-----------------------------------------+
→ | Level | Code | Message                                 |
→ +-------+------+-----------------------------------------+
→ | Note  | 1265 | Data truncated for column 'hired' at row 1 |
→ +-------+------+-----------------------------------------+
→ 1 row in set (0.00 sec)
```

Se un'operazione di inserimento viola il vincolo UNIQUE, il server la annulla, a meno che sia stata specificata la parola riservata IGNORE, nel qual caso l'inserimento fallisce:

```
INSERT INTO employee VALUES(NULL,NULL,"John Smith",35000,NOW());
```

→ ERROR 1062 (23000): Duplicate entry 'John Smith' for key 'empname'

```
INSERT IGNORE INTO employee VALUES(NULL,NULL,"John Smith",35000,NOW());
```

→ ^ Query OK, 0 rows affected, 1 warning (0.14 sec)

Si potrebbero anche inserire più righe manualmente, ma in genere si preferisce lasciare a Python il compito di svolgere il resto degli inserimenti.

Cancellazione

La cancellazione rimuove dalla tabella tutti i record individuati dal criterio di ricerca. Se non specificate il criterio di ricerca, il server rimuoverà tutti i record:

```
-- Rimuove John Smith se il suo stipendio è basso
DELETE FROM employee WHERE salary<11000 AND empname="John Smith";

-- Rimuove tutti
DELETE FROM employee;
```

Se volete rimuovere solo un determinato record, usate la sua chiave primaria univoca o comunque una condizione di identificazione univoca:

```
DELETE FROM employee WHERE id=387513;
```

Ricordate che la cancellazione è irreversibile!

Modifica

Una modifica aggiorna i valori delle colonne specificate nei record individuati dal criterio di ricerca. Se non specificate il criterio di ricerca, l'operazione riguarderà tutti i record:

```
-- Reset dello stipendio per tutti i neoassunti
UPDATE employee SET salary=35000 WHERE hired=CURDATE();

-- Aumenta lo stipendio di John Smith
UPDATE employee SET salary=salary+1000 WHERE empname="John Smith";

→ Query OK, 1 row affected (0.06 sec)
→ Rows matched: 1 Changed: 1 Warnings: 0
```

E, come immaginate, anche ogni modifica è irreversibile. Proprio come la cancellazione, è un'operazione distruttiva.

Selezione

Il comando SELECT seleziona tutte le colonne richieste in tutti i record individuati dal criterio di ricerca. Se non specificate il criterio di ricerca, otterrete tutti i record, che spesso è molto di più di quello che desideravate:

```
SELECT empname,salary FROM employee WHERE empname="John Smith";

→ +------------+--------+
→ | empname    | salary |
```

```
→ +------------+--------+
→ | John Smith |  36000 |
→ +------------+--------+
→ 1 row in set (0.00 sec)
```

```
SELECT empname,salary FROM employee;
```

```
→ +--------------+--------+
→ | empname      | salary |
→ +--------------+--------+
→ | John Smith   |  36000 |
→ | Jane Doe     |  75000 |
→ | Abe Lincoln  |   0.01 |
→ | Anon I. Muss |  14000 |
→ +--------------+--------+
→ 4 rows in set (0.00 sec)
```

Potete "potenziare" la selezione ordinando, raggruppando, aggregando e filtrando i risultati. Per ordinare i risultati, usate il modificatore ORDER BY: ordinamento su più colonne in senso discendente (DESC) o ascendente (ASC):

```
SELECT * FROM employee WHERE hired>='2000-01-01' ORDER BY salary DESC;
```

```
→ +----+---------------------+--------------+--------+------------+
→ | id | updated             | empname      | salary | hired      |
→ +----+---------------------+--------------+--------+------------+
→ |  4 | 2016-01-09 17:35:11 | Jane Doe     |  75000 | 2011-11-11 |
→ |  1 | 2016-01-09 17:31:29 | John Smith   |  36000 | 2016-01-09 |
→ |  6 | 2016-01-09 17:55:24 | Anon I. Muss |  14000 | 2011-01-01 |
→ +----+---------------------+--------------+--------+------------+
→ 3 rows in set (0.01 sec)
```

Per raggruppare e aggregare i risultati, usate il modificatore GROUP BY e una funzione di aggregazione, come COUNT(), MIN(), MAX(), SUM() o AVG():

```
SELECT (hired>'2001-01-01') AS Recent,AVG(salary)
  FROM employee
  GROUP BY (hired>'2001-01-01');
```

```
→ +--------+--------------------+
→ | Recent | AVG(salary)        |
→ +--------+--------------------+
→ |      0 | 0.009999999776482582 |
→ |      1 |   41666.666666666664 |
→ +--------+--------------------+
→ 2 rows in set (0.00 sec)
```

Quest'ultima istruzione calcola e restituisce lo stipendio medio di ogni gruppo di dipendenti sulla base del fatto che siano stati assunti prima o dopo il 01/01/2001.

Le parole riservate WHERE e HAVING filtrano i risultati della selezione; il server esegue il WHERE prima del raggruppamento e HAVING dopo il raggruppamento.

```
SELECT AVG(salary),MIN(hired),MAX(hired) FROM employee
  GROUP BY YEAR(hired)
  HAVING MIN(hired)>'2001-01-01';
```

```
→ +-------------+-------------+-------------+
→ | AVG(salary) | MIN(hired) | MAX(hired) |
→ +-------------+-------------+-------------+
→ |       44500 | 2011-01-01 | 2011-11-11 |
```

```
→ |      36000 | 2016-01-09 | 2016-01-09 |
→ +-------------+------------+------------+
→ 2 rows in set (0.00 sec)
```

Questa istruzione calcola e visualizza lo stipendio medio e le date estreme di assunzione per ogni gruppo di dipendenti assunti dopo il 01/01/2001.

Unione

L'operazione di unione combina il contenuto di due tabelle sulla base di una o più colonne. MySQL supporta cinque tipi di unione: *interna* (o *straight join*), a sinistra, a destra, esterna e naturale. Quest'ultima può essere a destra o a sinistra. Un'unione interna restituisce le righe aventi almeno una corrispondenza in entrambe le tabelle. Le unioni a sinistra/destra uniscono tutte le righe rispettivamente dalla tabella a sinistra o a destra, anche se non esiste alcuna corrispondenza sull'altro lato. L'unione esterna restituisce le righe con una corrispondenza in una delle due tabelle. Se una tabella non ha una corrispondenza, il server restituisce NULL. Un'unione naturale si comporta come un'unione esterna, tranne per il fatto che coinvolge implicitamente tutte le colonne con gli stessi nomi.

I seguenti comandi creano una nuova tabella con la descrizione del dipendente, aggiungono alla colonna un indice che verrà usato per l'unione ed estraggono il nome e la posizione del dipendente da entrambe le tabelle (la sintassi dell'ultimo esempio è quella di un'unione implicita interna):

```
-- Prepara e popola un'altra tabella
CREATE TABLE position (eid INT, description TEXT);
INSERT INTO position (eid,description) VALUES (6,'Imposter'),
   (1,'Accountant'),(4,'Programmer'),(5,'President');
ALTER TABLE position ADD INDEX(eid);

-- Rilegge i dati uniti
SELECT employee.empname,position.description
   FROM employee,position WHERE employee.id=position.eid
   ORDER BY position.description;

→ +--------------+-------------+
→ | empname      | description |
→ +--------------+-------------+
→ | John Smith   | Accountant  |
→ | Anon I. Muss | Imposter    |
→ | Abe Lincoln  | President   |
→ | Jane Doe     | Programmer  |
→ +--------------+-------------+
→ 4 rows in set (0.00 sec)
```

Unità 19 - Uso di un database MySQL: pymysql

Python impiega un apposito modulo driver per comunicare con MySQL. Molti driver per database, come pymysql, sono disponibili gratuitamente. Per questo esercizio, useremo pymysql, che fa parte di Anaconda. Il driver, quando viene attivato, si connette con il server del database e trasforma le chiamate a funzioni di Python in query per il database e, nella direzione opposta, i risultati restituiti dal database in strutture di dati Python.

La funzione connect() richiede informazioni sul database (il suo nome), i dettagli del server del database (nome host e numero di porta) e l'utente del database (nome e password). Se la connessione ha successo, restituisce l'identificatore di connessione. Il passo successivo consiste nel creare un cursore associato alla connessione con il database:

```
conn = pymysql.connect(host="localftost", port=3306, user="dsuser", passwd="badpasswOrd", db="dsdb")
cur = conn.cursor()
```

La funzione execute() del cursore richiede l'esecuzione di una query e restituisce il numero di righe modificate (0, se la query è non distruttiva). La query è semplicemente una stringa di caratteri, preparata sulla base delle informazioni apprese nell'unità precedente. A differenza di una query MySQL a riga di comando, una query pymysql non richiede il punto e virgola finale.

```
query = '''
SELECT employee.empname,position.description
   FROM employee,position WHERE employee.id=position.eid
   ORDER BY position.description
'''
n_rows = cur.execute(query)
```

Se si invia una query non distruttiva (come SELECT), la funzione cursore fetchall() fornisce tutti i record corrispondenti. La funzione restituisce un generatore che potete convertire in una lista di tuple di colonne:

```
results = list(cur.fetchall())
```

```
→ [('John Smith', 'Accountant'), ('Anon I. Muss', 'Imposter'),
→ ('Abe Lincoln', 'President'), ('Jane Doe', 'Programmer')]
```

Se la query è distruttiva (come UPDATE, DELETE o INSERT), occorre confermarla con un *commit*. Notate che è la connessione, non il cursore, a

offrire la funzione commit().

```
conn.commit()
```

Se non eseguite il commit di una query distruttiva, il server non modificherà le tabelle.

I database relazionali esistono fin dal 1974 (http://www.quickbase.com/articles/timeline-of-database-history). Hanno pertanto una "storia" venerabile e sono ottimi con i dati normalizzati, ovvero dati facili da suddividere in tabelle, colonne e righe. In realtà, qualsiasi dataset può essere normalizzato, ma il costo della normalizzazione può essere proibitivo (in termini di tempo di normalizzazione e di prestazioni delle query). Alcuni tipi di dati (soprattutto documenti testuali, immagini, audio e video clip e strutture irregolari di dati) hanno una naturale resistenza alla normalizzazione. Se è così, non costringeteli; piuttosto, per memorizzarli utilizzate un documento NoSQL, argomento della prossima Unità.

Unità 20 - Gestire gli archivi di documenti: MongoDB

Un *archivio di documenti* (un database NoSQL) è una collezione non volatile di oggetti, spesso sotto forma di documenti con i relativi attributi. Sono state sviluppate varie implementazioni di sistemi per l'archiviazione di documenti. In questa Unità, esamineremo in particolare una di esse, MongoDB, con un'occhiata anche al suo principale concorrente, CouchDB (http://couchdb.apache.org).

MongoDB è un database non relazionale. Un server MongoDB può supportare vari database non correlati. Un database è costituito da una o più collezioni di documenti. Tutti i documenti di una collezione hanno un identificatore univoco.

Un client Python MongoDB è implementato nel modulo Python `pymongo` come un'istanza della classe `MongoClient`. Potete creare un client senza parametri (che funziona in una tipica installazione a server locale) oppure specificando come parametri il nome dell'host e il numero della porta del server o infine specificando come parametro l'URI (*Uniform Resource Identifier*) del server:

```
import pymongo as mongo
# Inizializzazione standard
client1 = mongo.MongoClient()
# Host e porta specificati esplicitamente
client2 = mongo.MongoClient("localhost", 27017)
# Host e porta specificati come un URI
client3 = mongo.MongoClient("mongodb://localhost:27017/")
```

Nel momento in cui il client stabilisce una connessione con il server del database, potete selezionare il database attivo e poi la collezione attiva. Potete usare una notazione a oggetti ("a punto") o a dizionario. Se il database o la collezione selezionati non esistono, il server li creerà:

```
# Due modi per creare/selezionare il database attivo
db = client1.dsdb
db = client1["dsdb"]

# Due modi per creare/selezionare la collezione attiva
people = db.people
people = db[ "people"]
```

`pymongo` rappresenta i documenti MongoDB come dizionari Python. Ogni dizionario che rappresenta un oggetto deve avere la chiave `_id`. Se la chiave

è assente, il server la genera automaticamente.

Un oggetto collezione offre tutte le funzioni necessarie per l'inserimento, la ricerca, la rimozione, l'aggiornamento, la sostituzione e l'aggregazione dei documenti della collezione, e per creare gli indici.

Le funzioni insert_one(doc) e insert_many(docs) inseriscono un documento o una lista di documenti in una collezione. Restituiscono, rispettivamente, gli oggetti InsertOneResult o InsertManyResult, che, a loro volta, forniscono gli attributi inserted_id e inserted_ids. Potete usare questi attributi per conoscere le chiavi del documento per quei documenti che non hanno una chiave esplicita. Se si specifica la chiave _id, questa non cambierà dopo l'inserimento:

```
person1 = {"empname" : "John Smith", "dob" : "1957-12-24"}
person2 = {"_id" : "XVT162", "empname" : "Jane Doe", "dob" : "1964-05-16"}
person_id1 = people.insert_one(person1).inserted_id
```

→ ObjectId('5691a8720f759d05092d311b')

```
# Notate il nuovo campo "_id"!
person1
```

→ {'empname': 'John Smith', 'dob': '1957-12-24',
→ '_id': ObjectId('5691a8720f759d05092d311b')}

```
person_id2 = people.insert_one(person2).inserted_id
```

→"XVT162"

```
persons = [{"empname" : "Abe Lincoln", "dob" : "1809-02-12"},
        {"empname" : "Anon I. Muss"}]
result = people.insert_many(persons)
result.inserted_ids
```

→ [ObjectId('5691a9900f759d05092d311c'),
→ ObjectId('5691a9900f759d05092d311d')]

Le funzioni find_one() e find() forniscono uno o più documenti (che, opzionalmente, hanno una determinata proprietà). find_one() restituisce il documento e find() restituisce un cursore (un generatore) che potete convertire in una lista con la funzione list() o usare come un iteratore in un ciclo for. Se passate un dizionario come parametro a una di queste funzioni, queste restituiranno i documenti i cui valori sono uguali a tutti i valori del dizionario per le rispettive chiavi:

```
everyone = people.find()
list(everyone)
```

→ [{'empname': 'John Smith', 'dob': '1957-12-24',
→ '_id': ObjectId('5691a8720f759d05092d311b')},
→ {'empname': 'Jane Doe', 'dob': '1964-05-16',
→ '_id': 'XVT162'},
→ {'empname': 'Abe Lincoln', 'dob': '1809-02-12',
→ '_id': ObjectId('5691a9900f759d05092d311c')},

```
→ {'empname': 'Anon I. Muss',
→  '_id': ObjectId('5691a9900f759d05092d311d')}]
```

```
list(people.find({"dob" : "1957-12-24"}))
```

```
→ [{'empname': 'John Smith', 'dob': '1957-12-24',
→  '_id': ObjectId('5691a8720f759d05092d311b')}]
```

```
people.find_one()
```

```
→ [{'empname': 'John Smith', 'dob': '1957-12-24',
→  '_id': ObjectId('5691a8720f759d05092d311b')}]
```

```
people.find_one({"empname" : "Abe Lincoln"})
```

```
→ {'empname': 'Abe Lincoln', 'dob': '1809-02-12',
→ '_id': ObjectId('5691a9900f759d05092d311c')}
```

```
people.find_one({"_id" : "XVT162"})
```

```
→ {'empname': 'Jane Doe', 'dob': '1964-05-16',
→ '_id': 'XVT162'}
```

Varie funzioni di raggruppamento e ordinamento offrono funzionalità di aggregazione e ordinamento dei dati. La funzione `sort()` ordina i risultati della query. Richiamata senza argomenti, `sort()` esegue l'ordinamento in base alla chiave `_id` in senso ascendente. La funzione `count()` restituisce il numero di documenti nella query o nell'intera collezione:

```
people.count()
```

```
→ 4
```

```
people.find({"dob": "1957-12-24"}).count()
```

```
→ 1
```

```
people.find().sort("dob")
```

```
→ [{'empname': 'Anon I. Muss',
→  '_id': ObjectId('5691a9900f759d05092d311d')},
→ {'empname': 'Abe Lincoln', 'dob': '1809-02-12',
→  '_id': ObjectId('5691a9900f759d05092d311c')},
→ {'empname': 'John Smith', 'dob': '1957-12-24',
→  '_id': ObjectId('5691a8720f759d05092d311b')},
→ {'empname': 'Jane Doe', 'dob': '1964-05-16',
→  '_id': 'XVT162'}]
```

Le funzioni `delete_one(doc)` e `delete_many(doc)` rimuovono dalla collezione il documento o i documenti identificati dal dizionario `doc`. Per rimuovere tutti i documenti, ma conservando la collezione, usate `delete_many({})` come parametro un dizionario vuoto:

```
result = people.delete_many({"dob" : "1957-12-24"})
result.deleted count
```

```
→ 1
```

CouchDB

Un altro database NoSQL molto noto è CouchDB. A differenza di MongoDB, CouchDB privilegia la disponibilità alla coerenza. Se CouchDB viene replicato

(ovvero se si trova a operare su più computer), i suoi utenti potranno impiegarlo sempre, ma non necessariamente osserveranno gli stessi documenti. Al contrario, se MongoDB viene replicato, tutti gli utenti vedranno esattamente gli stessi documenti, ma per alcuni utenti il database potrebbe non risultare disponibile. Se non prevedete di replicare il database, la scelta fra CouchDB e MongoDB è puramente di natura estetica.

Esercitazioni

Gestire i database è una branca dell'informatica che non può rientrare nei limiti di questo libro. La semplice lettura di questo capitolo non potrà avervi trasformato in amministratori o programmatori di database. Tuttavia, ora sapete come creare un paio di tabelle, come memorizzarvi i dati e come estrarre i dati di cui avete bisogno, e potete farlo in due modi: con o senza SQL.

Indicizzatore di file MySQL (*)

Scrivete un programma Python che, per ogni parola di un determinato file, registri in un database MySQL la parola stessa, il suo numero ordinale nel file (partendo da 1) e il suo contrassegno part-of-speech. Per riconoscere le parole, usate `WordPunctTokenizer` di NLTK (introdotto nell'Unità 16). Immaginate che le parole siano sufficientemente brevi da rientrare nel tipo di dati MySQL `TINYTEXT`. Progettate lo schema del database, create tutte le tabelle necessarie e provate a lavorarci dall'interfaccia prima di iniziare qualsiasi attività di programmazione Python.

Convertitore MySQL - MongoDB (**)

L'istruzione MySQL `DESCRIBE table_name` fornisce i nomi, i tipi di dati, i vincoli, i valori predefiniti e così via per tutte le colonne della tabella. Scrivete un programma Python che trasferisca tutti i dati da una tabella MySQL (indicata dall'utente) a un documento MongoDB. Il programma non deve modificare i valori timestamp.

Usare i dati numerici tabulari

È un errore madornale teorizzare prima ancora di conoscere i fatti.
Sir Arthur Conan Doyle, scrittore britannico

Spesso i dati grezzi sono contenuti all'interno di documenti testuali. Piuttosto spesso, il testo contiene effettivamente i numeri. I fogli di lavoro Excel o CSV e in particolare le tabelle dei database, possono contenere milioni o addirittura miliardi di record numerici. Python è un eccellente strumento per l'elaborazione di testi, ma talvolta non riesce a offrire prestazioni adeguate in campo numerico. È proprio qui che entra in gioco numpy.

NumPy (*Numeric Python*) è un'interfaccia su un'intera famiglia di funzioni, molto efficienti e parallelizzabili, che implementano operazioni numeriche ad alte prestazioni. Il modulo numpy offre una nuova struttura di dati Python, l'array, e tutto un insieme di funzioni specifiche per gli array, così come il supporto dei numeri casuali, dell'aggregazione dei dati, dell'algebra lineare, delle trasformate di Fourier e tante altre cose.

Terabyte di dati?
Se il vostro programma deve accedere a grandi quantità di dati numerici, magari parecchi terabyte, non potete evitare di usare il modulo h5py (http://www.h5py.org/). Tale modulo è un portale sul formato dei dati binari HDF5 che è in grado di operare insieme a molto software indipendente, come IDL e MATLAB. h5py imita i meccanismi di numpy e Python, come gli array e i dizionari. Una volta che saprete usare numpy, potrete procedere con h5py, anche se non ne parleremo in questo libro.

In questo capitolo, impareremo a creare array numpy di varie forme e da varie fonti, a cambiare forma agli array e a prenderne solo una parte, a sommare gli indici degli array e ad applicare funzioni aritmetiche, logiche e di aggregazione ad alcuni o a tutti elementi di un array.

Unità 21 - Creazione di array

Gli array numpy sono più compatti e veloci delle normali liste Python, in particolare nei casi di multidimensionalità. Tuttavia, a differenza delle liste, gli array sono omogenei: non consentono di impiegare elementi aventi tipi di dati differenti.

Vi sono vari modi per creare un array numpy. La funzione array() crea un array a partire da dati che hanno già l'aspetto di un array. I dati possono trovarsi in una lista, una tupla o un altro array. numpy determina il tipo degli elementi dai dati stessi, a meno che utilizziate esplicitamente il parametro dtype. numpy supporta una ventina di tipi di dati, come bool_, int64, uint64, float64 e <U32 (per le stringhe Unicode).

Quando numpy crea un array, non copia i dati dall'origine al nuovo array, crea solo un collegamento e questo per motivi di efficienza. Questo significa che, per default, un array numpy è un po' come una vista sui dati sottostanti, non una loro copia. Se cambia l'oggetto contenente i dati sottostanti, cambiano anche i dati dell'array. Se il collegamento ai dati non fosse il comportamento desiderato (per esempio se i dati sono compatti e maneggevoli), basta passare al costruttore il parametro copy=True.

Le liste sono array; anche gli array sono array

A dispetto del loro nome, le "liste" Python in realtà sono implementate come array, non come liste. Questo significa, fra l'altro, che non viene riservato spazio per i puntatori in avanti, perché non vengono usati. Le grandi liste Python richiedono solo circa il 13 percento in più di spazio rispetto ai "veri" array numpy. Tuttavia, Python è in grado di eseguire alcune semplici operazioni interne, come sum(), da cinque a dieci volte più velocemente sulle liste che sugli array. Pertanto, prima di iniziare un progetto in numpy, domandatevi se davvero avete bisogno delle specifiche funzionalità di numpy.

Ora creiamo il nostro primo array: un piccolo array dei primi dieci numeri interi positivi:

```
import numpy as np
numbers = np.array(range(1, 11), copy=True)
```

→ array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

Le funzioni ones(), zeros() e empty() costruiscono, rispettivamente, degli array di tutti "1", di tutti "0" e di tutte voci non inizializzate. Richiedono un parametro dimensionale obbligatorio: una lista o una tupla delle dimensioni dell'array.

```
ones = np.ones([2, 4], dtype=np.float64)

→ array([[ 1., 1., 1., 1.],
→        [ 1., 1., 1., 1.]])

zeros = np.zeros([2, 4], dtype=np.float64)

→ array([[ 0., 0., 0., 0.],
→        [ 0., 0., 0., 0.]])

empty = np.empty([2, 4], dtype=np.float64)
# Il contenuto dell'array non è necessariamente di zeri!
→ array([[ 0., 0., 0., 0.],
→        [ 0., 0., 0., 0.]])
```

numpy memorizza il numero di dimensioni, la forma e il tipo di dati di un array tramite gli attributi ndim, shape e dtype.

```
ones.shape     # La forma originaria

→ (2, 4)

numbers.ndim   # Equivale a len(numbers.shape)

→ 1

zeros.dtype

→ dtype('float64')
```

La funzione eye(N, M=None, k=0, dtype=np.float) costruisce una matrice N×M costituita da una serie di "1" nella diagonale k-esima, mentre tutto il resto della matrice contiene "0". Il valore k positivo rappresenta le diagonali sopra la diagonale principale. Quando M è None (il default), M=N.

```
eye = np.eye(3, k=1)

→ array([[ 0., 1., 0.],
→        [ 0., 0., 1.],
→        [ 0., 0., 0.]])
```

Quando occorre moltiplicare più matrici, usate una matrice di identità quale valore iniziale dell'accumulatore nella catena delle moltiplicazioni.

Oltre alla "buona vecchia" funzione range(), numpy offre anche modi più efficienti per generare array di numeri ben intervallati: la funzione arange([start,] stop[, step,], dtype=None).

```
np_numbers = np.arange(2, 5, 0.25)

→ array([ 2. , 2.25, 2.5 , 2.75, 3. , 3.25, 3.5 , 3.75, 4. , 4.25, 4.5 , 4.75])
```

Esattamente come range(), il valore start può essere più piccolo di stop (ma allora step deve essere negativo e l'ordine dei numeri nell'array sarà decrescente).

numpy determina il tipo degli elementi in fase di creazione dell'array, ma tale tipo non è immutabile: potete modificarlo successivamente richiamando la funzione astype(dtype, casting="unsafe", copy=True). Nel caso di una riduzione del tipo (conversione in un tipo di dati più specifico), alcune informazioni si perderanno. Ma questo accade per qualsiasi riduzione, non solo in numpy.

```
np_inumbers = np_numbers.astype(np.int)
```

→ array([2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4])

La maggior parte delle operazioni svolte con numpy (come la trasposizione, di cui parleremo nella prossima Unità) restituisce una vista, non una copia dell'array di partenza. Per conservare i dati originari, la funzione copy() crea una copia dell'array. Qualsiasi modifica all'array di partenza non altererà la copia; d'altra parte, se l'array contiene un miliardo di elementi, è meglio riflettere bene, prima di copiarlo.

```
np_inumbers_copy = np_inumbers.copy()
```

Procediamo ora con alcune operazioni più complesse.

Unità 22 - Trasposizione e alterazione

A differenza dei grandi monumenti del passato, gli array numpy non sono scavati nella pietra. È facile cambiare la loro forma e il loro orientamento. Creiamo pertanto un array monodimensionale con alcuni simboli di azioni di borsa e proviamo a manipolarlo:

```
# Alcuni simboli di borsa
sap = np.array(["MMM", "ABT", "ABBV", "ACN", "ACE", "ATVI", "ADBE", "ADT" ])
```

```
→ array('MMM', 'ABT', 'ABBV', 'ACN', 'ACE', 'ATVI', 'ADBE', 'ADT'], dtype='<U4')
```

La funzione reshape(d0, d1, ...) cambia la forma di un array. Gli argomenti definiscono le nuove dimensioni. Il numero totale degli elementi, nella vecchia e nella nuova forma deve coincidere.

```
sap2d = sap.reshape(2, 4)
```

```
→ array([['MMM', 'ABT', 'ABBV', 'ACN'],
→        ['ACE', 'ATVI', 'ADBE', 'ADT']],
→       dtype='<U4')
```

```
sap3d = sap.reshape(2, 2, 2)
```

```
→ array([[['MMM', 'ABT'],
→         ['ABBV', 'ACN']],
→        [['ACE', 'ATVI'],
→         ['ADBE', 'ADT']]],
→       dtype='<U4')
```

Per trasporre un array, non serve neppure richiamare una funzione: il valore dell'attributo T è già la vista trasposta dell'array (per un array monodimensionale, data.T==data; per un array bidimensionale, le righe e le colonne vengono scambiate).

```
sap2d.T
```

```
→ array([['MMM', 'ACE'],
→        ['ABT', 'ATVI'],
→        ['ABBV', 'ADBE'],
→        ['ACN', 'ADT']],
→       dtype='<U4')
```

Sostanzialmente, l'attributo T mostra la matrice scambiando le dimensioni: l'asse 0 ("verticale") diviene l'asse 1 ("orizzontale") e viceversa. La funzione swapaxes() è una versione più generale di T. Esegue la trasposizione di un array multidimensionale scambiando i due assi passati come parametri. Naturalmente, passando gli assi 0 e 1 per un array

bidimensionale si esegue semplicemente la trasposizione dell'array, come abbiamo visto prima.

```
sap3d.swapaxes(1, 2)
```

```
→ array([[['MMM', 'ABBV'],
→        ['ABT', 'ACN']],
→       [['ACE', 'ADBE'],
→        ['ATVI', 'ADT']]],
→      dtype='<U4')
```

La funzione `transpose()` è ancora più generale di `swapaxes()` (nonostante il fatto che il suo nome implichi un'analogia con l'attributo `T`). `transpose()` permuta alcuni o tutti gli assi di un array multidimensionale sulla base del parametro fornito, che deve essere una tupla. Nel seguente esempio, il primo asse rimane "verticale", mentre gli altri due assi vengono scambiati.

```
sap3d.transpose((0, 2, 1))
```

```
→ array([[['MMM', 'ABBV'],
→        ['ABT', 'ACN']],
→       [['ACE', 'ADBE'],
→        ['ATVI', 'ADT']]],
→      dtype='<U4')
```

Naturalmente, il risultato è lo stesso di `swapaxes(1, 2)`.

Unità 23 - Indicizzazione e slicing

Gli array numpy supportano le stesse operazioni di indicizzazione [i] e *slicing* [i:j] delle liste Python. In più, implementano l'indicizzazione booleana: potete usare come indice un array di valori booleani, e il risultato della selezione sarà un array di quegli elementi dell'array di partenza per i quali l'indice booleano è True. Potete usare l'indicizzazione booleana sul lato destro (per una selezione) e sul lato sinistro (per un assegnamento).

Supponete che il fornitore dei vostri dati vi dica che tutti i dati del dataset dirty siano assolutamente non negativi. Questo significa che qualsiasi valore negativo del dataset non è un vero valore, ma un errore, e dovete sostituirlo con qualcosa di sensato, magari uno zero. Questa operazione è chiamata "pulizia" dei dati, *data cleaning*. Per ripulire i dati "sporchi", individuate i valori errati e sostituiteli con delle alternative ragionevoli.

```
dirty = np.array([9, 4, 1, -0.01, -0.02, -0.001])
whos_dirty = dirty <0    # Array booleano, da usare come indice booleano
```

→ array([False, False, False, True, True, True], dtype=bool)

```
dirty[whos_dirty] =0     # Cambia tutti i valori negativi in 0
```

→ array([9, 4, 1, 0, 0, 0])

Potete combinare anche più espressioni booleane, usando gli operatori | (OR logico), & (AND logico) e - (NOT logico). Quali elementi della seguente lista sono compresi fra -0.5 e 0.5? Chiedetelo a numpy!

```
linear = np.arange(-1, 1.1, 0.2)
(linear <= 0.5) & (linear >= -0.5)
```

→ array([False, False, False, True, True, True, True, True, False, False, False],
→ dtype=bool)

C'è booleano e booleano…

Gli operatori relazionali (come < e ==) hanno una precedenza inferiore rispetto agli operatori sui bit &, | e ! che rappresentano gli operatori booleani negli array numpy. Questo comportamento è fonte di confusione, perché gli operatori booleani "normali" di Python, or, and e not, hanno una precedenza inferiore rispetto agli operatori relazionali. Pertanto occorre racchiudere i confronti di array fra parentesi per assicurarsi che numpy li valuti per primi.

Un'altra caratteristica interessante degli array numpy è l'indicizzazione e lo slicing "intelligente", in cui un indice non è uno scalare, ma un array o una lista di indici. Il risultato della selezione è l'array degli elementi individuati dall'indice. Selezioniamo il secondo, il terzo e l'ultimo dei simboli di azioni dalla lista (un esempio di indicizzazione "intelligente").

```
sap[[1, 2, -1]]
```

```
→ array(['ABT', 'BBV', 'ADT'],
→       dtype='<U4')
```

E perché non estrarre dal nuovo array tutte le righe della colonna centrale (un esempio di slicing "intelligente"). In realtà, è possibile farlo in due modi:

```
sap2d[:, [1]]
```

```
→ array([['ABT'],
→        ['ATVI']],
→       dtype='<U4')
```

```
sap2d[:, 1]
```

```
→ array(['ABT', 'ATVI'],
→       dtype='<U4')
```

Python è noto per la sua capacità di offrire più strumenti per la soluzione di problemi simili e per non perdonare la scelta dello strumento sbagliato. Confrontate le due selezioni presentate in precedenza. La prima selezione è una matrice bidimensionale; la seconda è un array monodimensionale. A seconda di ciò che intendevate estrarre, uno dei risultati è sbagliato. Dunque assicuratevi sempre di aver ottenuto quello che desideravate.

Unità 24 - Operazioni vettoriali

Gli array numpy sono molto legati alle operazioni aritmetiche vettoriali con altri array (sempre che abbiano la stessa forma). Per sommare due array, elemento per elemento, senza utilizzare numpy, occorre ricorrere a un ciclo for o alla manipolazione delle liste; con numpy, basta eseguire una somma:

```
a = np.arange(4)
b = np.arange(1, 5)
a + b
```

→ **array([1, 3, 5, 7])**

Le operazioni vettoriali sugli array sono chiamate anche *broadcasting*. Le operazioni vettoriali su due dimensioni sono possibili se sono uguali (come sopra) o se uno di essi è uno scalare (come sotto):

```
a * 5
```

→ **array([0, 5, 10, 15])**

ATTENZIONE

L'operatore asterisco (*) si comporta in modo differente in Python e in numpy.
L'espressione Python "classica" seq * 5 replica la lista seq per cinque volte.
La stessa espressione, in numpy, moltiplica per 5 ogni elemento dell'array seq.

Potete utilizzare insieme varie operazioni aritmetiche sugli array e sugli scalari. Creiamo una matrice diagonale e aggiungiamole un po' di rumore (ma non casuale):

```
noise = np.eye(4) + 0.01 * np.ones((4, ))
```

→ **array([[1.01, 0.01, 0.01, 0.01],**
→ **[0.01, 1.01, 0.01, 0.01],**
→ **[0.01, 0.01, 1.01, 0.01],**
→ **[0.01, 0.01, 0.01, 1.01]])**

Ma come fare per aggiungere un po' di rumore casuale? Parleremo dei generatori di numeri casuali in seguito nell'Unità 47, *Statistica in Python*, ma ecco una piccola anteprima:

```
noise = np.eye(4) + 0.01 * np.random.random([4, 4])
np.round(noise, 2)
```

→ **array([[1.01, 0. , 0.01, 0.],**
→ **[0.01, 1.01, 0. , 0.01],**
→ **[0. , 0. , 1. , 0.],**
→ **[0. , 0. , 0.01, 1.]])**

Qui abbiamo arrotondato la matrice con la *funzione universale* round(): tutti gli elementi con un'unica chiamata a funzione! Parleremo di questo

argomento nell'Unità 25, *Le funzioni universali*.

A proposito, se provate a eseguire più volte l'esempio precedente, otterrete risultati differenti. Questo perché i numeri casuali sono... casuali!

Un'analisi più dettagliata e grafica della generazione di un segnale sinusoidale realistico e rumoroso si trova nell'Unità 30, *Generare un'onda sinusoidale di sintesi*.

Unità 25 - Le funzioni universali

Le funzioni vettoriali universali, o *ufunc*, rappresentano la controparte a funzione delle operazioni vettoriali. Potete applicare le ufunc a tutti gli elementi di un array in una sola chiamata a funzione. numpy offre un buon numero di ufunc; eccone alcune.

- Aritmetiche: add(), multiply(), negative(), exp(), log(), sqrt().
- Trigonometriche: sin(), cos(), hypot().
- Bit-a-bit: bitwise_and(), left_shift().
- Relazionali e logiche: less(), logical_not(), equal().
- maximum() e minimum().
- Funzioni per numeri in virgola mobile: isinf(), isfinite(), floor(), isnan().

Supponiamo di aver registrato in un array monodimensionale chiamato stocks le quotazioni delle azioni di borsa per otto simboli, dopo e prima un weekend:

```
stocks

→ array([ 140.49, 0.97, 40.68, 41.53, 55.7 , 57.21, 98.2 ,
→         99.19, 109.96, 111.47, 35.71, 36.27, 87.85, 89.11,
→         30.22, 30.91])
```

Vogliamo conoscere le quotazioni che hanno perso durante il weekend. Innanzitutto raggrupperemo i prezzi in base al simbolo e trasformeremo l'array di partenza in una matrice 2×8:

```
stocks = stocks.reshape(8, 2).T

→ array([[ 140.49, 40.68, 55.7 , 98.2 , 109.96, 35.71, 87.85, 30.22],
→        [  0.97, 41.53, 57.21, 99.19, 111.47, 36.27, 89.11, 30.91]])
```

Ora possiamo applicare la funzione greater() a entrambe le righe, svolgere un confronto per colonne e trovare il simbolo che ci interessa utilizzando un'indicizzazione booleana:

```
fall = np.greater(stocks[0], stocks[1])

→ array([True, False, False, False, False, False, False, False],
→       dtype=bool)

sap[fall]

→ array(['MMM'],
→       dtype='<U4')
```

A proposito, si dà il caso che *MMM* sia una società russa che applica uno schema Ponzi e che non si ritrova in nessun'altra borsa. Nessuna meraviglia che le sue azioni siano in declino.

Oltre ai numeri "tradizionali", numpy supporta appieno lo standard per numeri in virgola mobile IEEE 754 e offre i simboli per l'infinito positivo (inf) e per i "non-numeri" (nan - not-a-number). Esistono anche all'esterno di numpy come float("inf") e float("nan"). In accordo con la tradizione dei dati scientifici, useremo nan come un segnaposto per i dati mancanti (ne abbiamo parlato nell'Unità 1, *La tipica sequenza di analisi dei dati*).

La funzione universale isnan() è uno strumento eccellente per individuare gli errori nei dati. E anche se nel seguente esempio la sostituzione dei dati mancanti con uno zero probabilmente non è affatto una buona idea, l'abbiamo già fatto in precedenza (nell'Unità 23, *Indicizzazione e slicing*), pertanto ripetiamolo:

```
# Fingiamo che il nuovo MMM sia mancante
stocks[1, 0] = np.nan
np.isnan(stocks)

→ array([[False, False, False, False, False, False, False, False],
→        [ True, False, False, False, False, False, False, False]], dtype=bool)

# Ripariamo il danno; in un modo del tutto brutale
stocks[np.isnan(stocks)] = 0

→ array([[ 140.49, 40.68, 55.7, 98.2 , 109.96, 35.71, 87.85, 30.22],
→        [   0. , 41.53, 57.21, 99.19, 111.47, 36.27, 89.11, 30.91]])
```

Le funzioni universali espandono le possibilità delle funzioni aritmetiche e degli operatori relazionali Python. Le funzioni condizionali sono operatori logici Python "con gli steroidi".

Unità 26 - Le funzioni condizionali

La funzione `where(c, a, b)` è la versione `numpy` dell'operatore ternario `if-else`. Richiede un array booleano (`c`) e altri due array (`a` e `b`) e restituisce l'array `d[i]=a[i] if c[i] else b[i]`. Tutti e tre gli array devono avere la stessa forma.

Le funzioni `any()` e `all()` restituiscono `True` se, rispettivamente, uno qualsiasi o tutti gli elementi di un array sono `True`.

La funzione `nonzero()` restituisce gli indici di tutti gli elementi diversi da zero.

Nell'Unità 25, *Le funzioni universali*, abbiamo registrato alcune quotazioni di azioni in un array `sap`. Per scoprire quali quotazioni sono cambiate sostanzialmente (di più di $1.00 per azione), sostituiamo le variazioni modeste con "0", individuiamo gli elementi diversi da zero e usiamo i loro indici come "indici intelligenti" sull'array dei simboli delle azioni:

```
changes = np.where(np.abs(stocks[1] - stocks[0]) > 1.00, stocks[1] - stocks[0], 0)

→ array([-139.52, 0. , 1.51, 0. , 1.51, 0. , 1.26, 0. ])

sap[np.nonzero(changes)]
→ array(['MMM', 'ABBV', 'ACE', 'ADBE'],
→       dtype='<U4')
```

Si potrebbe ottenere la stessa risposta usando i soli indici booleani:

```
sap[np.abs(stocks[1] - stocks[0]) > 1.00]

→ array(['MMM', 'ABBV', 'ACE', 'ADBE'],
→       dtype='<U4')
```

Ma sarebbe stato meno divertente!

Unità 27 - Aggregazione e ordinamento degli array

L'ordinamento e l'aggregazione dei dati sono gli elementi di base della scienza dei dati. Si parte con grandi quantità di dati e poi li si raffinano gradualmente ordinandoli, calcolandone la media, eseguendo accumulazioni e così via, fino a ottenere un insieme compatto, maneggevole e facilmente interpretabile. numpy offre le funzioni mean(), sum(), std() (deviazione standard), min() e max() che restituiscono le rispettive misure aggregate di un array numpy.

Useremo una combinazione di operazioni vettoriali, funzioni di aggregazione, ufunc e indici booleani (quasi tutto ciò che abbiamo appreso finora!) per estrarre le azioni (dall'Unità 25, *Le funzioni universali*), che sono cambiate, in entrambe le direzioni, di più della media delle azioni controllate:

```
sap[ np.abs(stocks[0] - stocks[1])
   np.mean(np.abs(stocks[0] - stocks[1]))]

→ array(['MMM'],
→       dtype='<U4')
```

Ma, onestamente, mettere insieme le variazioni positive e negative non sembra davvero una buona idea.

Le funzioni cumsum(x) e cumprod(x) calcolano le somme e i prodotti cumulativi: cumsumi=∑1i xi e cumprodi=∏1i xi. Potete usarle come semplici integratori additivi (calcolo dell'interesse semplice) e moltiplicativi (calcolo dell'interesse composto). Ma attenzione: se un elemento dell'array è 0, anche il corrispondente elemento cumprod() e tutti gli elementi successivi saranno a 0.

Confrontiamo il calcolo dell'interesse semplice e composto per 30 anni con un tasso di interesse del 3.75 percento, il tutto in (sostanzialmente) due righe di codice numpy:

Listato 5.1 interest.py.

```
# Questo è un listato parziale
RATE = .0375
TERM = 30
simple = ( RATE * np.ones(TERM)).cumsum()
compound = ((1 + RATE) * np.ones(TERM)).cumprod() - 1
```

`sort()` è forse la funzione più noiosa del modulo. Ordina semplicemente un array in loco (ovvero cancella l'array di partenza) e restituisce `None`. Se l'array di partenza deve essere conservato, eseguitene una copia prima di ordinarlo.

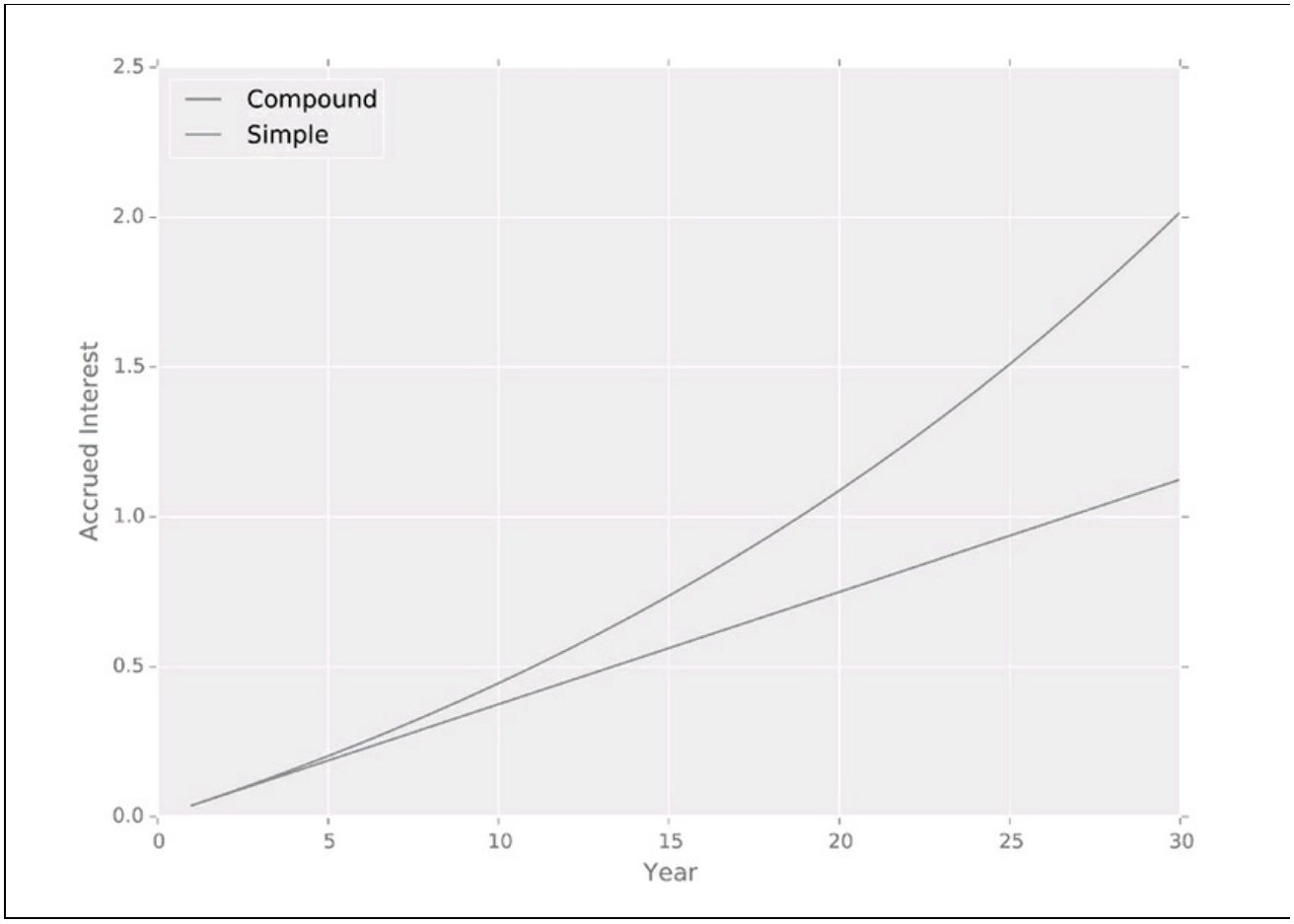


Figura 5.1

Unità 28 - Trattare gli array come se fossero insiemi

In alcuni casi, l'ordine degli elementi di un array è meno importante della composizione dell'array: il fatto che un determinato elemento sia presente o meno nell'array o che tipi di elementi sono presenti nell'array. numpy è anche in grado di trattare gli array come insiemi matematici.

La funzione unique(x) restituisce un array di tutti gli elementi univoci di x. È un ottimo sostituto del modulo Counter (del quale abbiamo parlato nell'Unità 7, *I contatori*), ma in realtà non conta le occorrenze.

Sappiamo che la bioinformatica è una scienza di grande interesse. La bioinformatica si occupa di sequenze di geni, valutando l'ordine dei nucleotidi nel DNA. Proviamo allora a svolgere alcune operazioni di pseudo-bioinformatica e vediamo quanti tipi di nucleotidi esistono in un determinato frammento di DNA:

```
dna = "AGTCCGCGAATACAGGCTCGGT"
dna_as_array = np.array(list(dna))

→ array(['A', 'G', 'T', 'C', 'C', 'G', 'C', 'G', 'A', 'A', 'T', 'A', 'C', 'A',
→ 'G', 'G', 'C', 'T', 'C', 'G', 'G', 'T'],
→       dtype='<U1')

np.unique(dna_as_array)

→ array(['A', 'C', 'G', 'T'],
→       dtype='<U1')
```

Lo sapevate già, vero? (http://www.genomenewsnetwork.org/resources/whats_a_genome/Chp2_1.shtml).

La funzione in1d(needle, haystack) restituisce un array booleano dove un elemento è True se l'elemento corrispondente di needle è in haystack. Gli array needle e haystack non devono necessariamente avere la stessa forma.

```
np.in1d(["MSFT", "MMM", "AAPL"], sap)

array([False, True, False], dtype=bool)
```

Le funzioni union1d() e intersect1d() calcolano l'unione e l'intersezione degli insiemi rappresentati da due array monodimensionali. Anche qui gli array non devono necessariamente avere le stesse dimensioni. Tuttavia, potreste voler continuare a usare gli operatori Python dedicati agli insiemi, & e |, che fra l'altro sono due volte più veloci delle funzioni numpy!

Unità 29 - Salvataggio e lettura di array

Probabilmente vi troverete a usare numpy non proprio da solo, ma insieme a pandas (Capitolo 6, *Manipolare serie di dati e frame*), a networkx (Capitolo 7, *Utilizzo dei dati delle reti*) e agli strumenti di machine learning (Capitolo 10, *Machine Learning*). Si possono creare array numpy dai dati forniti da uno strumento di basso livello di elaborazione dei dati, per fornirli a strumenti di analisi di livello più elevato. È improbabile che dovrete mai salvare o leggere direttamente degli array numpy.

Tuttavia, qualora fosse necessario, numpy offre delle funzionalità per salvare gli array su file .npy (la funzione save(file, arr)) e poi per leggere gli array da tali tipi di file (la funzione load(file)). I file sono in un formato binario e solo numpy può gestirli.

Entrambe le funzioni, seppur inutili, sono molto facili da usare: file può essere l'handle di un file precedentemente aperto o una stringa contenente il nome del file e se non si specifica l'estensione .npy, questa verrà aggiunta automaticamente.

```
# Un modo bizzarro per copiare un array
np.save("sap.npy", sap)
sap_copy = np.load("sap")
```

Un'altra coppia di funzioni, loadtxt() e savetxt(), carica dati tabulari da un file di testo e salva un array su un file di testo. numpy crea automaticamente il file, se necessario, e lo apre. numpy è perfino in grado di espandere e comprimere il file, se il suo nome termina con .gz. Potete controllare il modo in cui numpy gestisce le righe commentate e i delimitatori e salta le righe indesiderate:

```
arr = np.loadtxt(fname, comments="#", delimiter=None, skiprows=0, dtype=float)
np.savetxt(fname, arr, comments="#", delimiter=" ", dtype=float)
```

Unità 30 - Generare un'onda sinusoidale di sintesi

È giunto il tempo di impressionare i vostri amici e fare qualcosa che gli esperti di scienza dei dati normalmente non fanno: generare un'*onda sinusoidale di sintesi*, un segnale periodico che potrebbe essere stato catturato da uno strumento economico, imperfetto e rumoroso con una scala limitata. L'effettiva provenienza del segnale e la natura dello strumento non sono importanti. Magari era un voltmetro connesso a una presa di corrente, un termometro digitale da esterni che ha raccolto dati per qualche giorno o anche un'app per il controllo delle quotazioni di borsa. A proposito, potete usare dei segnali periodici sintetici non solo per impressionare gli amici, ma anche per collaudare nuovi algoritmi di elaborazione dei segnali digitali.

Il codice di generazione è il seguente. numpy dà il meglio nelle porzioni di codice evidenziate. È qui che si svolgono le operazioni vettoriali: si crea un array di numeri interi sequenziali, li si converte in numeri in virgola mobile, si regola il periodo, si calcola il seno, si amplificano i valori, se ne esegue uno scostamento, si aggiunge del rumore gaussiano (vedere l'Unità 45, *Le distribuzioni delle probabilità*) e si tronca il risultato come se fosse stato misurato da uno strumento di range limitato.

Listato 5.2 numpy_sinewave.py.

```
# Importa tutto il necessario
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

# Le costanti definiscono le proprietà:
# il segnale, il rumore e lo "strumento"
SIG_AMPLITUDE = 10; SIG_OFFSET = 2; SIG_PERIOD = 100
NOISE_AMPLITUDE = 3
N_samples = 5 * SIG_PERIOD
INSTRUMENT_RANGE = 9

→ # Costruisce un'onda sinusoidale e le aggiunge del rumore casuale
→ times = np.arange(N_samples).astype(float)
→ signal = SIG_AMPLITUDE * np.sin(2 * np.pi * times / SIG_PERIOD) + SIG_OFFSET
→ noise = NOISE_AMPLITUDE * np.random.normal(size=N_samples)
→ signal += noise
→
→ # Tronca i picchi che cadono all'esterno del range dello strumento
→ signal[signal > INSTRUMENT_RANGE] = INSTRUMENT_RANGE
→ signal[signal < -INSTRUMENT_RANGE] = -INSTRUMENT_RANGE
```

```
# Traccia i risultati
matplotlib.style.use("ggplot")
plt.plot(times, signal)
plt.title("Synthetic sine wave signal")
plt.xlabel("Time")
plt.ylabel("Signal + noise")
plt.ylim(ymin = -SIG_AMPLITUDE, ymax = SIG_AMPLITUDE)

# Salva il grafico
plt.savefig("../images/signal.pdf")
```

La parte non evidenziata del codice mostra l'uso di Matplotlib per visualizzare la forma del segnale.

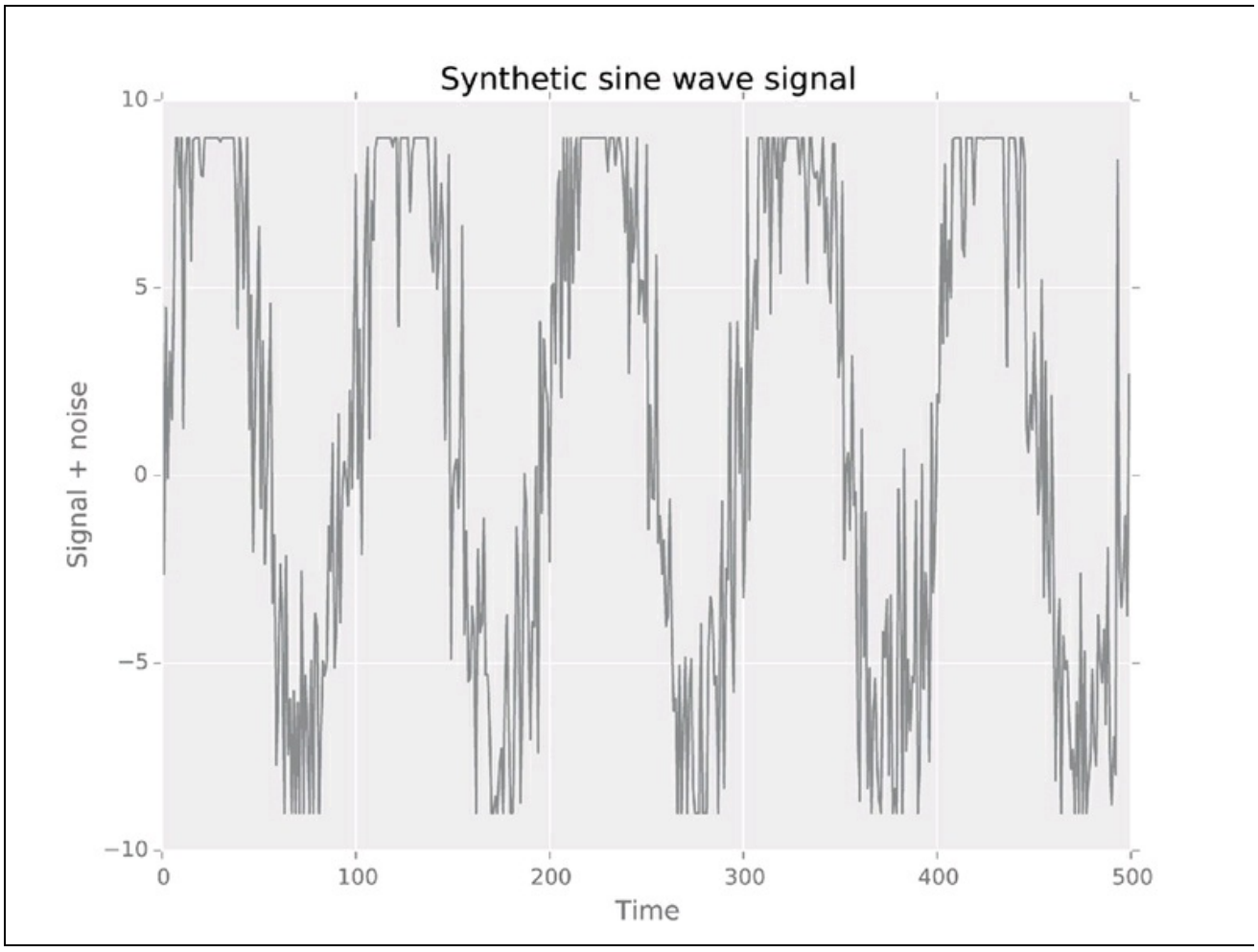


Figura 5.2

Matplotlib, da affiancare a numpy, verrà trattato nel Capitolo 8, *Rappresentazione grafica*.

Esercitazioni

Spero di avervi convinto che numpy è uno strumento eccellente per tutte le attività di elaborazione numerica. È in grado di elaborare senza problemi vettori e matrici; offre l'aritmetica vettoriale, logica e varie altre operazioni; e offre tutto il necessario per cambiare forma, ordinare e aggregare i dati. Fornisce perfino quella strana creatura chiamata nan, un numero che non è un numero. Praticamente non vi è progetto a elevata dose di calcoli che possa fare a meno di numpy.

Sottrazioni fra array (*)

Le somme parziali sono quasi l'equivalente di un integrale. In realtà, il calcolo definisce un integrale come una somma infinita di elementi infinitesimali. Le differenze parziali arri+1-arri sono quasi l'equivalente di una derivata. numpy non fornisce uno strumento per calcolare differenze parziali fra array. Scrivete un programma che, dato un array arr, calcoli la differenza parziale degli elementi dell'array. Immaginate che l'array sia numerico.

HEI Locator (**)

Scaricate il dataset U.S. Higher Education Datasets da www.data.gov/education come un file CSV. Scrivete un programma che fornisca le dieci istituzioni HEI (Higher Education Institution) geograficamente più vicine al punto definito dalla latitudine e longitudine media di tutte le istituzioni del dataset. Calcolate le distanze in gradi. Provate a usare il più possibile numpy per la memorizzazione ed elaborazione dei dati. Ricordate che la prima riga del file CSV contiene le intestazioni delle colonne e che alcuni campi o interi record del file potrebbero essere errati.

Calcolo delle analogie negli Stati USA (***)

Il Census Bureau statunitense offre un riepilogo dei flussi della popolazione fra Stati (scaricate il file XLS più recente da

www.census.gov/hhes/migration/data/acs/state-to-state.html e convertitelo in formato CSV aprendolo in Excel o OpenOffice Calc ed esportandolo in CSV). Scrivete un programma che indichi le dieci coppie di Stati più simili in termini di fenomeni migratori. Considerate due Stati X e Y simili se più di PX/N persone si sono trasferite da X to Y, dove PX/Y è il flusso totale in uscita da X e N è il numero totale di Stati e territori, meno il territorio di origine. Provate a usare il più possibile numpy per la memorizzazione ed elaborazione dei dati. Gli Stati di ogni coppia si trovano, in genere, sulla stessa costa?

Manipolare serie di dati e frame

> Le gallerie d'arte di Parigi contengono la più fine raccolta di cornici che io abbia mai visto.
> *Humphry Davy*, chimico e inventore, originario della Cornovaglia

Gli esperti di scienza dei dati in genere amano avere a disposizione dati in forma tabulare (array, vettori, matrici). I dati tabulari sono molto comodi e possono essere utilizzati elemento per elemento o anche per righe e colonne. Tutti i sistemi, dai supercomputer ai personal computer forniscono operazioni aritmetiche vettoriali che possono essere eseguite su più elementi o anche su tutti gli elementi della tabella (ne abbiamo vista un'implementazione numpy nell'Unità 24, *Operazioni vettoriali*). Tuttavia, numpy non riesce a connettere i dati numerici con i loro attributi: nomi di righe e colonne e indici. Questa mancanza di riferimenti complica il lavoro quando occorre operare su più array numpy.

E qui entra in gioco pandas.

Lo scopo del modulo pandas è quello di aggiungere a Python le astrazioni delle serie e dei frame di dati che rappresentano la base di un linguaggio rivale, R, il linguaggio principe della scienza dei dati. pandas opera sulla base di numpy, del quale estende e, in parte reimplementa le funzionalità.

Un frame di dati pandas è, in pratica, un foglio di lavoro "intelligente": una tabella che riporta etichette per le colonne (le variabili), le righe (le osservazioni) e una ricca collezione di operazioni interne (qui una serie è semplicemente un frame avente una sola colonna). La parte della tabella dedicata ai dati (le celle) è implementata come un array numpy. Molte operazioni (come il cambio di forma dei dati e le funzioni di aggregazione e universali) somigliano alle versioni numpy. Le etichette di riga e di colonna forniscono un accesso comodo e "parlante" alle singole righe e colonne. Inoltre, le etichette di righe e colonne consentono a noi programmatori di combinare più frame unendoli e concatenandoli in senso "verticale" (uno

sopra l'altro) o in "orizzontale" (fianco a fianco). In questo senso, i frame si comportano come le tabelle di un database relazionale. Per rinfrescare la memoria sui database relazionali, tornate a consultare il Capitolo 4, *Utilizzare i database*.

Infine, ma non meno importante, pandas va perfettamente d'accordo con pyplot, un sistema Python per il tracciamento e la visualizzazione dei dati, di cui parleremo nell'Unità 41, *Tracciamento di grafici con PyPlot*. In pratica, pandas offre tutto il necessario per le attività che coinvolgono la scienza dei dati (insieme a tutti gli altri strumenti, naturalmente).

In questo capitolo, inizieremo il nostro viaggio esaminando due container pandas, Series e DataFrame, nell'Unità 31, *Le strutture di dati in Pandas*.

Unità 31 - Le strutture di dati in Pandas

Il modulo pandas aggiunge due nuovi container al già ricco set di strutture di dati Python: Series e DataFrame. Una serie è un vettore monodimensionale, etichettato (ovvero indicizzato). Un frame è una tabella righe e colonne dotate di etichette, un po' come un foglio di lavoro di Excel o una tabella MySQL. Ogni colonna del frame è una serie. Con alcune eccezioni, pandas tratta i frame e le serie allo stesso modo.

I frame e le serie non sono semplici contenitori. Offrono un supporto per varie operazioni di manipolazione dei dati, come:

- indicizzazione monolivello e gerarchica;
- gestione dei dati mancanti;
- operazioni aritmetiche e booleane su intere colonne e tabelle;
- operazioni tipiche da database (come l'unione e l'aggregazione);
- tracciamento grafico di singole colonne e intere tabelle;
- lettura e scrittura di dati da file.

I frame e le serie dovrebbero essere impiegati ogni volta che si ha a che fare con dati tabulari mono- o bidimensionali. In pratica sono strutture troppo comode per non usarle.

Le serie

Una serie è un vettore monodimensionale di dati. Così come gli array numpy (trovate ulteriori informazioni nell'Unità 21), le serie sono omogenee: tutti gli elementi di una serie devono appartenere allo stesso tipo di dati.

Potete creare una semplice serie da qualsiasi sequenza: una lista, una tupla o un array. Usiamo una tupla di dati USA sull'inflazione per mostrare una serie di dati pandas. Ma, innanzitutto, un'avvertenza: ho inserito i dati dell'inflazione, precalcolati, in una tupla per mettere l'accento sulla loro natura immutabile; a parte questo, il seguente esempio non richiede alcuna conoscenza avanzata di economia o finanza.

```
import pandas as pd
# L'ultimo valore è errato, lo correggeremo successivamente!
inflation = pd.Series((2.2, 3.4, 2.8, 1.6, 2.3, 2.7, 3.4, 3.2, 2.8, 3.8, \
   -0.4, 1.6, 3.2, 2.1, 1.5, 1.5))
```

```
→ 0    2.2
→ 1    3.4
→ 2    2.8
→ 3    1.6
→ 4    2.3
→ 5    2.7
→ 6    3.4
→ 7    3.2
→ 8    2.8
→ 9    3.8
→ 10  -0.4
→ 11   1.6
→ 12   3.2
→ 13   2.1
→ 14   1.5
→ 15   1.5
→ dtype: float64
```

La funzione interna `len()` di Python è una sorta di "coltellino svizzero". Funziona anche per le serie:

```
len(inflation)
```

```
→ 16
```

NOTA

La parola *serie* è invariabile al singolare e al plurale. Deriva dal verbo latino *serere*, che significa *unire*, da cui l'italiano *serrare*.

Una semplice serie, come quella appena creata, adotta un indice intero: l'etichetta del primo elemento è 0, quella del secondo è 1 e così via. L'attributo `values` di una serie contiene la lista di tutti i valori della serie; l'attributo `index` fa riferimento all'indice della serie (`index` è un altro tipo di dati `pandas`); l'attributo `index.values` fa riferimento all'array di tutti i valori indice.

```
inflation.values
```

```
→ array([ 2.2, 3.4, 2.8, 1.6, 2.3, 2.7, 3.4, 3.2, 2.8,
→         3.8, -0.4, 1.6, 3.2, 2.1, 1.5, 1.5])
```

```
inflation.index
```

```
→ Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], dtype='int64')
```

```
inflation.index.values
```

```
→ array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
```

Notate come `pandas` ci eviti di "reinventare la ruota" e come basi tutto su array `numpy` .

Piuttosto sorprendente: tutti questi array (e gli attributi della serie che essi rappresentano) sono modificabili. Modificando `values`, `index` e `index.values` si

modificano effettivamente i valori e l'indice della serie. Usiamo questa possibilità per correggere l'ultimo valore dell'inflazione, che è errato:

```
inflation.values[-1] = 1.6
```

Il problema con le serie è il fatto che hanno l'aspetto di array e si comportano anche da array e, sostanzialmente, sono array. Per esempio, è difficile capire a quale anno fa riferimento il primo elemento della serie. Si potrebbe creare un'altra serie di anni e poi mantenere insieme le due serie, ma sappiamo bene che conservando in un programma due serie in parallelo si può andare incontro a veri disastri. Pertanto, creiamo una serie con un indice personalizzato passando un dizionario al costruttore `Series`. Le chiavi del dizionario diverranno gli indici della serie: una parte indivisibile della serie:

```
inflation = pd.Series({1999 : 2.2, «altri elementi», 2014 : 1.6, 2015 : np.nan})
```

→ 1999 2.2
→ «altri elementi»
→ 2014 1.6
→ 2015 NaN

Alternativamente, potete creare un nuovo indice da qualsiasi sequenza e poi collegarlo a una serie esistente:

```
inflation.index = pd.Index(range(1999, 2015))
inflation[2015] = numpy.nan
```

→ 1999 2.2
→ «altri elementi»
→ 2014 1.6
→ 2015 NaN

I valori e gli indici di una serie possono avere un nome, accessibile e assegnabile attraverso appositi attributi denominati. I nomi sono sostanzialmente una forma di documentazione che ricorda a noi stessi e a chiunque legga le istruzioni la natura delle sequenze:

```
inflation.index.name = "Year"
inflation.name = "%"
```

→ Year
→ 1999 2.2
→ «altri elementi»
→ 2014 1.6
→ 2015 NaN
→ Name: %, dtype: float64

Potete osservare l'intera serie, compresi indici e nomi, visualizzandola o semplicemente digitando il suo nome sulla riga di comando in modalità interattiva o richiamando le funzioni `head()` e `tail()`, che restituiscono, rispettivamente, le prime e le ultime cinque righe di una serie:

```
inflation.head()

→ Year
→ 1999 2.2
→ 2000 3.4
→ 2001 2.8
→ 2002 1.6
→ 2003 2.3
→ Name: %, dtype: float64

inflation.tail()

→ Year
→ 2011 3.2
→ 2012 2.1
→ 2013 1.5
→ 2014 1.6
→ 2015 NaN
→ Name: %, dtype: float64
```

Se all'output delle funzioni `head()` e `tail()` preferite un'immagine (che, notoriamente, vale più di mille parole), osservate la figura successiva. Parleremo degli strumenti di tracciamento nel Capitolo 8, *Rappresentazione grafica*.

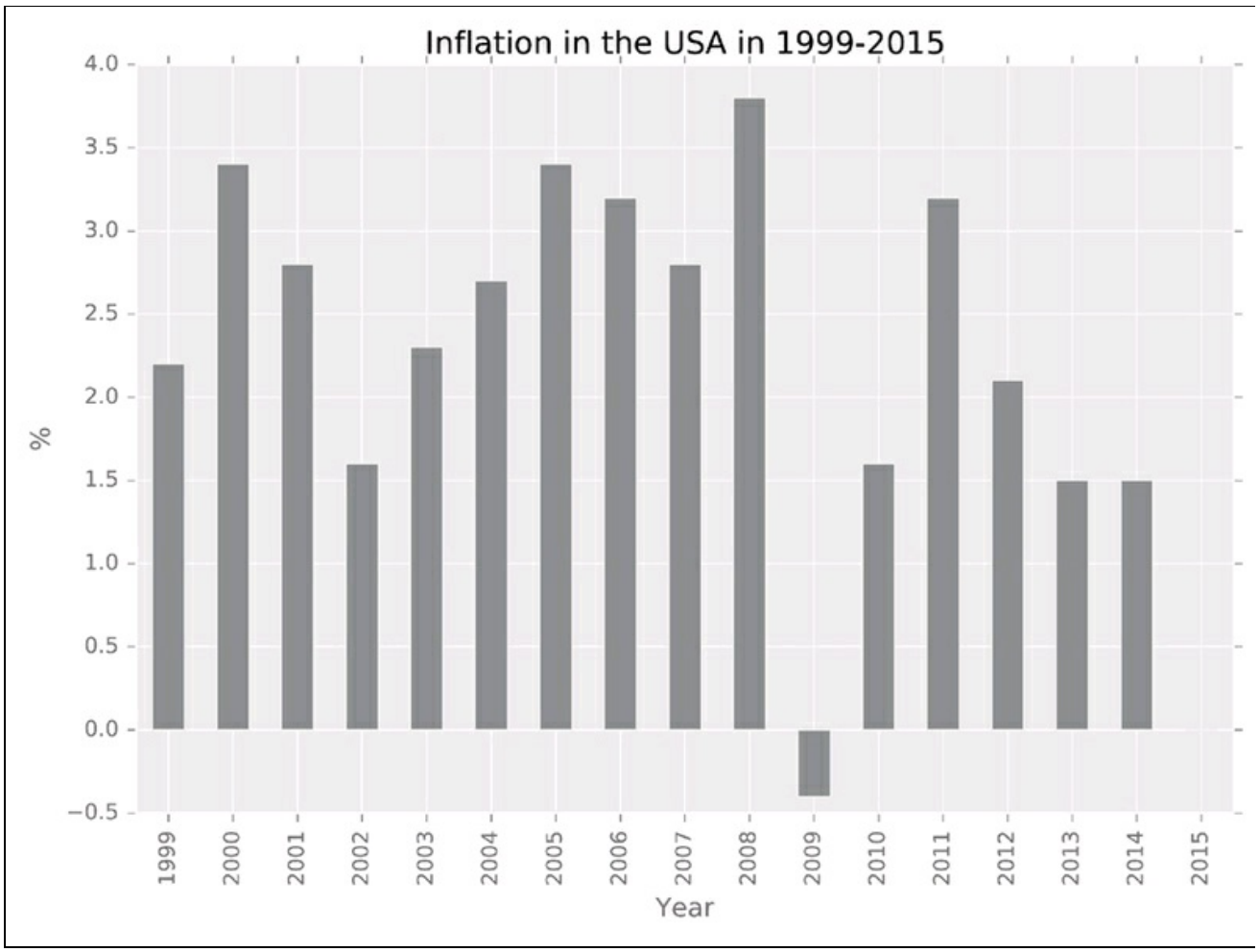


Figura 6.1

Le serie sono ottime per memorizzare le osservazioni di una variabile. Tuttavia, molti dataset contengono più di una variabile. Ecco dove intervengono i frame.

I frame

Un frame di dati è una tabella dotata di righe e colonne etichettate. Potete costruire un frame partendo da un array bidimensionale numpy, da una lista di tuple, da un dizionario Python o da un altro frame. Nel caso di un dizionario, le chiavi fungeranno da nomi per le colonne e i valori (che devono essere sequenze) diverranno i valori delle colonne. Nel caso di un altro frame, pandas copia i nomi delle colonne dal frame di origine al nuovo frame. Nel caso di un array, potete fornire i nomi delle colonne tramite il parametro opzionale columns (una sequenza di nomi di colonne). In base all'approccio numpy, l'indice del frame diviene l'asse 0 ("verticale") e le colonne del frame diviene l'asse 1 ("orizzontale").

Per descrivere i frame, useremo un rapporto del 2011 sull'abuso di alcol negli USA (del NIAAA, *National Institute on Alcohol Abuse and Alcoholism*, https://pubs.niaaa.nih.gov/publications/surveillance92/CONS09.htm). Il rapporto elenca il consumo pro capite di alcol suddiviso per stato, per categoria (birra, vino e superalcolici) e per anno fra il 1977 e il 2009.

NOTA

Il rapporto NIAAA è una fonte di dati di tale qualità che ne ho inclusa una copia pre-elaborata nell'archivio del codice degli esempi allegati a questo libro. Ma ricordate: dopo aver "festeggiato", non mettetevi mai alla guida, neanche di un software di scienza dei dati!

Potete creare un semplice frame con nomi di colonne e un indice passando al costruttore una lista di tuple di righe o altre sequenze della stessa lunghezza. Qui ho usato un lettore di file CSV pandas tratto dall'Unità 37, *I/O su file in Pandas*, per creare il frame alco e poi ho selezionato l'arco di un anno.

```
alco2009 = pd.DataFrame([(1.20, 0.22, 0.58),
                         (1.31, 0.54, 1.16),
                         (1.19, 0.38, 0.74),
                         «altre righe»],
                        columns=( "Beer", "Wine", "Spirits"),
                        index=( "Alabama", "Alaska", «altri stati»))
```

```
→        Beer Wine Spirits
→ Alabama 1.20 0.22    0.58
→ Alaska  1.31 0.54    1.16
→ Arizona 1.19 0.38    0.74
→ «altre righe»
```

Per ottenere lo stesso effetto potete anche usare un dizionario di colonne:

```
→ alco2009 = pd.DataFrame({"Beer" :    (1.20, 1.31, 1.19, «altre righe»),
→                          "Wine" :    (0.22, 0.54, 0.38, «altre righe»),
→                          "Spirits" : (0.58, 1.16, 0.74, «altre righe»)},
→                         index=("Alabama", "Alaska", «altri stati»))
```

L'accesso alle singole colonne del frame può avvenire tramite una notazione a dizionario o a oggetto. Tuttavia, per aggiungere una nuova colonna, occorre usare la notazione a dizionario. Se viene usata la notazione a oggetto, invece, pandas crea un nuovo attributo del frame. Così come con le serie, anche i frame hanno le funzioni head() e tail().

```
alco2009["Wine"].head()
```

```
→ State
→ Alabama    0.22
→ Alaska     0.54
→ Arizona    0.38
→ Arkansas   0.17
→ California 0.55
→ Name: Wine, dtype: float64
```

```
alco2009.Beer.tail()
```

```
→ State
→ Virginia       1.11
→ Washington     1.09
→ West Virginia  1.24
→ Wisconsin      1.49
→ Wyoming        1.45
→ Name: Beer, dtype : float64
```

E come le serie, anche i frame supportano le operazioni vettoriali: potete assegnare un valore a tutte le righe di una colonna con un'unica istruzione di assegnamento. La colonna può anche non esistere; se non esiste, pandas la creerà.

```
alco2009[ "Total"] = 0
```

```
alco2009.head()
```

```
→            Beer Wine Spirits Total
→ State
→ Alabama     1.20 0.22    0.58     0
→ Alaska      1.31 0.54    1.16     0
→ Arizona     1.19 0.38    0.74     0
→ Arkansas    1.07 0.17    0.60     0
→ California  1.05 0.55    0.73     0
```

Qui i totali sono chiaramente errati, ma impareremo a correggerli nell'Unità 36, *Trasformazione dei dati*.

Unità 32 - Cambiare aspetto ai dati

Il principale contributo di pandas alla "causa" dei dati tabulari è l'aggiunta delle etichette: l'associazione di etichette numeriche o testuali alle colonne (nomi di colonne) e alle righe (indici "piatti" e gerarchici). Questa associazione è flessibile: se cambiate la forma dell'array numpy sottostante (con la funzione reshape(), vedi Unità 22) per adattarlo ad altri frame, alcune righe potrebbero diventare colonne e alcune colonne potrebbero diventare righe. Per esempio, se un indice gerarchico di un frame ha due livelli (per esempio, "Year" e "State"), mentre un altro frame ha un indice "piatto" di nome "State", si devono convertire le etichette "Year" in nomi di colonne. In questa Unità parleremo di indicizzazione piatta e gerarchica, di reindicizzazione e di altri modi per riorganizzare le etichette dei dati.

Indicizzazione

L'indice di un frame è una collezione di etichette assegnate alle righe del frame. Le etichette devono avere lo stesso tipo di dati, ma non devono necessariamente essere univoche. Potete fornire un indice impiegando il parametro opzionale index del costruttore DataFrame(). E come nel caso di una serie, è possibile leggere e modificare il nome delle colonne e l'indice attraverso gli attributi index.values e colonne.values.

```
alco2009.columns.values
```

→ array(['Beer', 'Wine', 'Spirits', 'Total'], dtype=object)

```
alco2009.index.values
```

→ array(['Alabama', 'Alaska', 'Arizona', «...»], dtype=object)

Qualsiasi colonna di un frame può diventare un indice: sarà compito delle funzioni reset_index() e set_index(column), rispettivamente, "destituire" l'eventuale indice corrente e dichiarare il nuovo. Entrambe le funzioni restituiscono un nuovo frame, ma se si fornisce il parametro opzionale inplace=True, le funzioni modificano l'intero oggetto frame:

```
alco2009.reset_index().set_index("Beer").head()
        State  Wine  Spirits  Total
```

→ Beer
→ 1.20 Alabama 0.22 0.58 0

```
→ 1.31    Alaska  0.54    1.16    0
→ 1.19    Arizona  0.38    0.74    0
→ 1.07    Arkansas  0.17    0.60    0
→ 1.05 California  0.55    0.73    0
```

L'indice di un frame è un importante strumento per l'accesso alle righe e l'identificazione delle righe. Qualsiasi colonna si usi come indice, è necessario che la scelta sia sensata. Nell'ultimo esempio, la colonna non aveva questo requisito: il consumo di birra è una proprietà, non un identificatore di uno stato.

Una volta attivato l'indice, potete accedere alle singole righe attraverso l'attributo "indice di riga" `ix`, che è come un dizionario della serie di righe, dove la chiave è rappresentata dalle etichette dell'indice. Le colonne del frame fungono da indice per ciascuna serie:

```
alco2009.ix["Nebraska"]
```

```
→ Beer     1.46
→ Wine     0.20
→ Spirits  0.68
→ Total    0.00
→ Name: Nebraska, dtype: float64
```

L'operatore Python `in` controlla se nel frame è presente una riga con una determinata etichetta:

```
"Samoa" in alco2009.index
```

```
→ False
```

La funzione `drop()` restituisce una copia di un frame dalla quale è stata rimossa una riga o una lista di righe. Per rimuovere le righe dal frame originario, occorre passarle il parametro opzionale `inplace=True`.

Reindicizzazione

La reindicizzazione crea un nuovo frame o una nuova serie partendo da un frame o una serie esistenti, selezionando determinate righe, colonne (o entrambe), anche permutate. Sostanzialmente, è una versione "smart" dell'indicizzazione `numpy` (ne abbiamo parlato nell'Unità 23, *Indicizzazione e slicing*); se però `pandas` non trova nel frame originario le etichette delle righe o colonne richieste, crea una o più nuove righe o colonne e le popola con valori `nan`.

Nel prossimo esempio, creiamo una lista di stati il cui nome inizia per "S" (compreso "Samoa", che non è uno stato e non è presente nel frame

alco2009). Poi prendiamo tutte colonne del frame, tranne l'ultima ("Total", che comunque non è adeguatamente inizializzata) e aggiungiamo un'altra colonna chiamata "Water". Infine, estraiamo dal frame originario le righe e le colonne selezionate. Poiché una riga e una colonna non esistono, pandas le crea:

```
s_states = [state for state in alco2009.index if state[0] == 'S'] + ["Samoa"]
drinks = list(alco2009.columns) + ["Water"]
nan_alco = alco2009.reindex(s_states, columns=drinks)
```

```
→                Beer Wine Spirits Water
→ State
→ South Carolina  1.36 0.24    0.77   NaN
→ South Dakota    1.53 0.22    0.88   NaN
→ Samoa            NaN  NaN     NaN   NaN
```

Il parametro opzionale method, che accetta i valori "ffill" (forward fill, riempimento in avanti) e "bfill" (backward fill, riempimento all'indietro) inserisce i valori mancanti. Questo riempimento funziona solo su indici a incremento o decremento monotonico. Parleremo di riempimento di dati nell'Unità 33.

Indicizzazione gerarchica

pandas supporta gli indici gerarchici (multilivello) e i nomi di colonne gerarchici (multilivello). Gli indici multilivello sono detti anche multiindici.

Un indice multilivello è costituito da tre liste.

- Nomi di livelli.
- Tutte le etichette possibili per ogni livello.
- Liste dei valori effettivi per ogni elemento del frame o della serie (le liste hanno la stessa lunghezza, uguale al numero di livelli presenti nell'indice).

Il seguente frame contiene una versione completa del dataset NIAAA, non solo limitata all'anno 2009. Presenta un multiindice sia per stato sia per anno ed è ordinato in base a entrambi gli indici: prima per "State" e poi per "Year".

```
Alco
```

```
→              Beer  Wine  Spirits
→ State    Year
→ Alabama  1977  0.99  0.13     0.84
→          1978  0.98  0.12     0.88
```

```
→         1979  0.98  0.12    0.84
→         1980  0.96  0.16    0.74
→         1981  1.00  0.19    0.73
→ «...»
→ Wyoming  2005  1.21  0.23    0.97
→         2006  1.47  0.23    1.05
→         2007  1.49  0.23    1.10
→         2008  1.54  0.23    1.12
→         2009  1.45  0.22    1.10
```

Spesso, gli indici gerarchici vengono prodotti dalle operazioni di trasformazione dei dati, ma potete anche costruirli esplicitamente. La funzione `MultiIndex.from_tuples()` prende una collezione di tuple con etichette e una lista (opzionale) di nomi di livelli e produce un multiindice. Potete associare il multiindice a un frame o a una serie esistente o passarlo come parametro al costruttore `DataFrame()`:

```
multi = pd.MultiIndex.from_tuples(("Alabama", 1977), ("Alabama", 1978), ("Alabama", 1979),
                                  ...,
                                  ("Wyoming", 2009)),
                                  names=["State", "Year"])
```

```
→ MultiIndex(levels=[['Alabama', 'Alaska', «...», 'Wyoming'],
→                    [1977, 1978, 1979, 1980, «...», 2009]],
→            labels=[[0, 0, 0, 0, 0, 0, 0, 0, «...», 50],
→                    [0, 1, 2, 3, 4, 5, 6, 7, «...», 32]],
→            names=['State', 'Year'])
```

```
alco.index = multi
```

Un multiindice può essere usato allo stesso modo di un indice "piatto". Una selezione parziale (di una delle etichette) produce un frame; una selezione completa produce una serie.

```
alco.ix['Wyoming'].head()
```

```
→       Beer  Wine Spirits
→ Year
→ 1977  1.79  0.21    1.32
→ 1978  1.82  0.22    1.36
→ 1979  1.86  0.22    1.30
→ 1980  1.85  0.24    1.32
→ 1981  1.91  0.24    1.27
```

```
alco.ix['Wyoming', 1999]
```

```
→ Beer    1.41
→ Wine    0.18
→ Spirits 0.84
→ Name: (Wyoming, 1999), dtype: float64
```

`pandas` tratta gli indici multilivello e le colonne in modo coerente, quanto meno fintantoché un livello di indice può diventare un livello di colonna e viceversa.

Stack e pivot

Potete appiattire, in parte o in toto, un indice multilivello, introducendo però nomi di colonne multilivello. Potete appiattire, in parte o in toto, dei nomi di colonne multilivello, introducendo però un multiindice.

La funzione stack() incrementa il numero di livelli nell'indice e, simultaneamente, decrementa il numero di livelli nei nomi di colonne. Rende, pertanto, il frame più "alto" e più "stretto", come illustrato nella Figura 6.2. Se i nomi di colonne sono più "piatti", restituisce una serie. La funzione unstack() fa l'esatto contrario: decrementa il numero di livelli nell'indice e, simultaneamente, incrementa il numero di livelli nei nomi di colonne. Rende, pertanto, il frame più "basso" e più "largo". Se l'indice è già piatto, restituisce una serie.

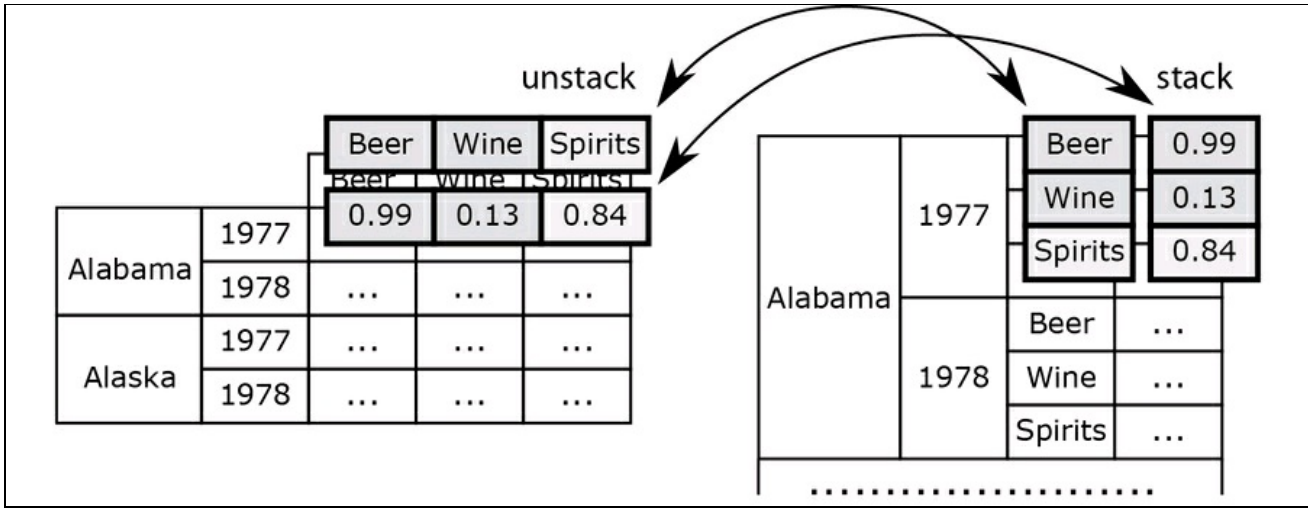


Figura 6.2

```
tall_alco = alco.stack()
tall_alco.index.names += ["Drink"]
tall_alco.head(10)
```

```
→ State    Year Drink
→ Alabama  1977 Beer     0.99
→              Wine     0.13
→              Spirits  0.84
→          1978 Beer     0.98
→              Wine     0.12
→              Spirits  0.88
→          1979 Beer     0.98
→              Wine     0.12
→              Spirits  0.84
→          1980 Beer     0.96
→ dtype: float64
```

Il risultato dell'operazione precedente è una serie con un indice a tre livelli (ho dovuto fornire il nome il terzo livello, mancante: "Drink").

```
wide_alco = alco.unstack()
wide_alco.head()
```

```
→ Beer
→ Year       1977  1978  1979  1980  1981  1982  1983  1984  1985  1986
→ State
→ Alabama     0.99  0.98  0.98  0.96  1.00  1.00  1.01  1.02  1.06  1.09
→ Alaska      1.19  1.39  1.50  1.55  1.71  1.75  1.76  1.73  1.68  1.68
→ Arizona     1.70  1.77  1.86  1.69  1.78  1.74  1.62  1.57  1.67  1.77
→ Arkansas    0.92  0.97  0.93  1.00  1.06  1.03  1.03  1.02  1.03  1.06
→ California  1.31  1.36  1.42  1.42  1.43  1.37  1.37  1.38  1.32  1.36
→
→ [5 rows x 99 columns]
```

Il risultato dell'operazione precedente è un frame con un indice "piatto" nomi di colonne gerarchici a due livelli. Si trovano spesso questi tipi di frame nei file CSV e tabulari in genere. L'operazione di stack consente di renderli più "quadrati" e facili da gestire.

Le operazioni di stack e unstack sono casi speciali di un'operazione più generale: il *pivoting*. La funzione pivot(indice, colonne, valori) converte un frame in un altro frame usando l'indice di una colonna come nuovo indice, le colonne come nuova lista di nomi di colonne e i valori delle colonne come dati.

Nel seguente esempio, i dati alco vengono riorganizzati in un frame "quadrato" che descrive il consumo di vino per anno (nuovo indice piatto) e per stato (nomi di colonne):

```
alco.pivot("Year", "State", "Wine")
```

```
→ State  Alabama  Alaska  Arizona  Arkansas  California  Colorado  Connecticut
→ Year
→ 1977      0.13    0.42     0.34      0.10        0.67      0.36         0.35
→ 1978      0.12    0.45     0.37      0.11        0.68      0.47         0.38
→ 1979      0.12    0.47     0.39      0.10        0.70      0.47         0.40
→ 1980      0.16    0.50     0.36      0.12        0.71      0.47         0.43
→ «...»
→
→ [33 rows x 51 columns]
```

Se indice è None, pandas riutilizza l'indice del frame originario.

Unità 33 - Gestione dei dati mancanti

Solo raramente i dati sono "perfetti". Alcuni valori sono sicuri (e non è necessario preoccuparsene); alcuni sono dubbi (e occorre trattarli con cautela); e alcuni sono semplicemente assenti.

Tradizionalmente, pandas impiega valori numpy nan (ne abbiamo parlato nell'Unità 25) per rappresentare i dati mancanti, probabilmente per evitare ogni confusione con valori "plausibili" e anche perché il suo nome somiglia al simbolo NA ("Not Available", non disponibile) del linguaggio R. pandas offre anche delle funzioni per individuare e ricostruire i valori mancanti.

Vi sono vari motivi per cui i valori possono non essere presenti nelle serie e nei frame: magari non sono mai stati raccolti; magari sono stati raccolti, ma scartati perché inaccettabili; magari il risultato della combinazione di più dataset, ha prodotto un dataset incompleto. Sfortunatamente, non è possibile svolgere una seria analisi dei dati prima di essersi occupati dei valori mancanti. Potete eliminarli o ricostruirli, ovvero sostituirli con valori sensati. Vediamo come fare.

Cancellazione dei dati mancanti

Il modo più semplice per gestire i dati mancanti è fingere di non averli. La funzione dropna() rimuove parzialmente (how="any", il valore di default) o interamente (how="all") le colonne (axis=0, il valore di default) o le righe (axis=1) non valide e restituisce una copia "pulita" del frame. Il parametro opzionale inplace=True modifica il frame originario, invece di crearne una copia.

```
nan_alco.dropna(how="all")

→              Beer  Wine  Spirits  Water
→ State
→ South Carolina  1.36  0.24     0.77    NaN
→ South Dakota    1.53  0.22     0.88    NaN

nan_alco.dropna(how="all", axis=1)

→              Beer  Wine  Spirits
→ State
→ South Carolina  1.36  0.24     0.77
```

```
→ South Dakota   1.53  0.22    0.88
→ Samoa           NaN   NaN     NaN
```

Non potete eliminare un solo valore mancante senza distruggere la griglia del frame. Potete rimuovere solo l'intera riga o colonna contenente la cella "sporca" e il frame risultante, "pulito" , potrebbe, al limite, essere completamente vuoto.

```
nan_alco.dropna()
```

```
→ Empty DataFrame
→ Columns: [Beer, Wine, Spirits, Water]
→ Index: []
```

Ricostruzione dei dati mancanti

Un altro modo per risolvere il problema dei valori mancanti consiste nel ricostruirli. Per ricostruire i valori mancanti occorre sostituirli con alcuni valori "puliti", sensati. Che cosa si intenda con "sensati" dipende, naturalmente, dalla natura dei dati. Solo noi siamo in grado di dire se i sostituti sono corretti o meno.

Due delle tecniche più comunemente utilizzate sono la sostituzione con un valore costante (uno zero, un uno e così via) e la sostituzione con una media tratta dai valori disponibili. Innanzitutto, quindi, occore identificare i valori effettivamente mancanti.

Le funzioni `isnull()` e `notnull()` sono complementari. Restituiscono `True`, rispettivamente, se un valore è `nan` o se non è un `nan`. Notate che in base allo standard IEEE 754 sui valori in virgola mobile, l'espressione `np.nan==np.nan` è `False`, il che rende impossibile qualsiasi confronto diretto!

```
nan_alco.isnull()
```

```
→                Beer   Wine  Spirits  Water
→ State
→ South Carolina  False  False    False   True
→ South Dakota    False  False    False   True
→ Samoa            True   True    True   True
```

```
nan_alco.notnull()
```

```
→                Beer   Wine  Spirits  Water
→ State
→ South Carolina   True   True    True  False
→ South Dakota     True   True    True  False
→ Samoa           False False    False  False
```

Correggiamo la colonna "Spirits" calcolando la media (ricordate che in `numpy` il trattino "-" è l'operatore di negazione):

```
sp = nan_alco['Spirits']      # Seleziona una colonna con dati "sporchi"
clean = sp.notnull()          # Le righe "pulite"
sp[-clean] = sp[clean].mean() # Ricostruisce le righe "sporche" con la media
nan_alco
```

```
→              Beer  Wine  Spirits  Water
→ State
→ South Carolina  1.36  0.24    0.770    NaN
→ South Dakota    1.53  0.22    0.880    NaN
→ Samoa           NaN   NaN    0.825    NaN
```

Dobbiamo inserire le medie colonna per colonna (o riga per riga), mentre le costanti possono essere inserite in tutto il frame. La funzione `fillna(val)`, nella sua forma più semplice, inserisce `val` nei dati assenti. Alternativamente, la funzione propaga l'ultima osservazione valida lungo la colonna (`axis=0`, il valore di default) o lungo la riga (`axis=1`), procedendo in avanti (`method="ffill"`) o all'indietro (`method="bfill"`). La funzione restituisce un nuovo frame o una nuova serie, a meno che si specifichi il parametro `inplace=True`.

```
nan_alco.fillna(0)
```

```
→              Beer  Wine  Spirits  Water
→ State
→ South Carolina  1.36  0.24    0.77     0
→ South Dakota    1.53  0.22    0.88     0
→ Samoa           0.00  0.00    0.00     0
```

```
nan_alco.fillna(method="ffill")
```

```
→              Beer  Wine  Spirits  Water
→ State
→ South Carolina  1.36  0.24    0.77    NaN
→ South Dakota    1.53  0.22    0.88    NaN
→ Samoa           1.53  0.22    0.88    NaN
```

Sostituzione dei valori

Un altro modo per gestire determinati valori "sporchi" consiste nel sostituirli selettivamente con valori "puliti" caso per caso. La funzione `replace(val_or_list,new_val)` sostituisce un valore o una lista di valori con un altro valore o lista di valori. Nel caso delle liste, devono avere la stessa lunghezza. La funzione restituisce un nuovo frame o una nuova serie, a meno che si passi il parametro `inplace=True`.

La funzione `combine_first(pegs)` combina due frame (o due serie). Sostituisce i valori mancanti nell'oggetto frame/serie con i valori corrispondenti nell'oggetto frame/serie passato come argomento. In un certo senso, l'argomento funge da origine dei valori di default.

Unità 34 - Combinare i dati

Una volta che i vostri dati si trovano in una serie o frame, potete aver bisogno di combinarli per prepararli per ulteriori elaborazioni, in quanto alcuni dati potrebbero trovarsi in un frame e altri in un altro frame. pandas offre alcune funzioni per unire e concatenare i frame.

Unione

Unire dei frame è un po' come unire le tabelle di un database: pandas combina le righe aventi indici identici (o valori identici in altre colonne designate) dai frame a sinistra e a destra. Quando esiste una sola corrispondenza nel frame a destra per ogni riga del frame a sinistra, si parla di unione uno-a-uno. Quando invece esiste più di una corrispondenza, si parla di unione uno-a-molti. In tal caso, pandas replica le righe del frame a sinistra in base alla necessità e la replicazione può provocare la duplicazione delle righe (ma vedremo come risolvere questo problema prima della fine di questa Unità). Quando vi sono più corrispondenze per più righe in entrambi i frame, si parla di unione molti-a-molti e, anche in questo caso, pandas replica le righe in base alle necessità e inserisce negli spazi vuoti dei numpy.nan.

Se entrambi i frame hanno una colonna con lo stesso nome (la colonna chiave), potete unire i frame sulla base di tale colonna. Altrimenti potete designare altre colonne che fungeranno da chiavi, come illustrato di seguito:

```
df = pd.merge(df1, df2, on="key")
df = pd.merge(df1, df2, left_on="key.Z", right_on="key2")
```

Useremo i dati del Census Bureau degli Stati Uniti d'America (https://www.census.gov/popest/data/historical/2000s/vintage_2009/state.html) per costruire un frame con la popolazione USA al 1 luglio 2009. Oltre ai dati stato per stato, il frame contiene le osservazioni relative alle regioni East, Northeast, Northwest, Midwest, West, South e degli interi Stati Uniti.

```
population.head()
```

```
→                Population
→ State
```

```
→ Wyoming                544270
→ District of Columbia   599657
→ Vermont                621760
→ North Dakota           646844
→ Alaska                 698473
```

Dato che population e alco2009 sono entrambi indicizzati in base alla colonna "State", potete rimuovere gli indici, unire i frame su tutte le colonne comuni e osservare affiancati i dati sulla popolazione e quelli sul consumo di alcol:

```
df = pd.merge(alco2009.reset_index(),
        population.reset_index()).set_index("State")
df.head()
```

```
→             Beer  Wine  Spirits  Population
→ State
→ Alabama     1.20  0.22     0.58     4708708
→ Alaska      1.31  0.54     1.16      698473
→ Arizona     1.19  0.38     0.74     6595778
→ Arkansas    1.07  0.17     0.60     2889450
→ California  1.05  0.55     0.73    36961664
```

Se due frame hanno delle colonne con lo stesso nome, pandas aggiunge i suffissi "_l" e "_r" ai nomi delle colonne. I suffissi sono controllati dal parametro opzionale suffixes (una tupla di due stringhe). Se volete eseguire l'unione sugli indici piuttosto che sulle colonne in generale, usate i parametri opzionali left_index=True e/o right_index=True. Il risultato della seguente istruzione è identico al precedente, ma l'ordinamento predefinito può essere differente:

```
df = pd.merge(alco2009, population, left_index=True, right_index=True)
df.head()
```

```
→ State                Beer  Wine  Spirits  Population
→ Wyoming              1.45  0.22     1.10      544270
→ District of Columbia 1.26  1.00     1.64      599657
→ Vermont              1.36  0.63     0.70      621760
→ North Dakota         1.63  0.25     1.16      646844
→ Alaska               1.31  0.54     1.16      698473
```

Se entrambi gli indici sono designati come chiavi, potete anche usare join() invece di merge():

```
population.join(alco2009).tail(10)
```

```
→ State          Population  Beer  Wine  Spirits
→ Illinois         12910409  1.22  0.39     0.73
→ Florida          18537969  1.21  0.48     0.92
→ New York         19541453  0.91  0.46     0.69
→ Texas            24782302  1.42  0.28     0.58
→ California       36961664  1.05  0.55     0.73
→ Northeast        55283679   NaN   NaN      NaN
→ Midwest          66836911   NaN   NaN      NaN
→ West             71568081   NaN   NaN      NaN
→ South           113317879   NaN   NaN      NaN
→ United States   307006550   NaN   NaN      NaN
```

Entrambe le funzioni, join() e merge() accettano un parametro opzionale how, i cui valori possono essere left (il valore di default per join()), right, inner (il valore di default per merge()) o outer. Una join a sinistra (left) impiega l'indice del frame a sinistra. Una join a destra (right) impiega l'indice del frame usato come parametro (a destra). Una join esterna (outer) impiega l'unione degli indici. Una join interna (inner) impiega l'intersezione degli indici. I tipi delle join pandas sono coerenti con i tipi delle join MySQL, introdotte in precedenza nell'Unità 18.

Se gli indici dei frame non sono identici, allora le join e i merge left, right e outer introducono delle righe con valori mancanti. Nell'esempio precedente, questo si è verificato dove i totali per Beer, Wine e Spirits non erano noti. Al contrario, le join/merge di tipo inner non introducono mai valori mancanti.

Se due frame hanno colonne con lo stesso nome, occorre fornire i parametri opzionali (stringa) lsuffix e rsuffix. pandas aggiungerà i suffissi ai nomi delle colonne comuni.

La funzione join() accetta anche il parametro opzionale on, che unisce due frame sulla base di una colonna in comune (ma non sulla base di colonne con nomi differenti o su una colonna e un indice).

Concatenamento

La funzione concat() concatena una lista di frame accostandoli fra loro in una delle dimensioni ("verticale", axis=0, il default o "orizzontale", axis=1) e restituisce un nuovo frame:

```
pd.concat([alco2009, population], axis=1).tail()
```

```
→                Beer  Wine  Spirits  Population
→ Washington     1.09  0.51    0.74     6664195
→ West            NaN   NaN     NaN    71568081
→ West Virginia  1.24  0.10    0.45     1819777
→ Wisconsin      1.49  0.31    1.16     5654774
→ Wyoming        1.45  0.22    1.10      544270
```

Se le dimensioni non corrispondono, pandas aggiunge ulteriori righe o colonne e spazi vuoti e valori mancanti.

pandas preserva attentamente tutti gli indici di tutti i frame impilati verticalmente, e ciò può portare ad avere un indice con chiavi duplicate. Potete accettarlo o eliminare i duplicati (come vedremo poco più avanti) o

anche usare il parametro opzionale keys (una lista di stringhe) che aggiunge al nuovo frame un indice di secondo livello, creando pertanto un indice gerarchico. Se concatenate i frame "orizzontalmente" (sulle colonne), il parametro keys crea nomi gerarchici di colonne.

Usiamo i dati del sito web Statistics Canada (http://www.statcan.gc.ca/tables-tableaux/sum-som/l01/cst01/demo02a-eng.htm) per creare un frame contenente la popolazione nelle varie province canadesi nel 2011. Ora possiamo creare un nuovo frame che descrive entrambi i paesi nordamericani in un modo adeguatamente indicizzato (ricordate che il frame USA è due anni più vecchio di quello canadese).

```
pop_na = pd.concat([population, pop_ca], keys=[ "US", "CA"])
pop_na.index.names = ("Country  ", "State")
```

```
→                          Population
→ Country  State
→ US       Wyoming                544270
→          District of Columbia   599657
→          Vermont                621760
→          North Dakota           646844
→          Alaska                 698473
→          «...»
→ CA       Alberta                3790200
→          British Columbia       4499100
→          Yukon                  35400
→          Northwest Territories  43500
→          Nunavut                34200
```

Ricordate che l'indice gerarchico può essere appiattito, se necessario.

Unire o concatenare?

Sia merge() sia concat() combinano due o più frame. Potete usare concat() per combinare frame che hanno contenuti simili (come le popolazioni degli stati USA e delle province canadesi, "mele con mele", per così dire). Potete usare merge() per combinare frame che hanno contenuti complementari (come le popolazioni e i livelli di consumo di alcol, "mele con arance").

Cancellazione dei duplicati

La funzione duplicated([subset]) restituisce una serie booleana che denota se la riga corrispondente contiene duplicati considerando tutte le colonne o un loro sottoinsieme subset (una lista di nomi di colonne). Il parametro opzionale keep controlla se deve essere contrassegnato il primo (first), l'ultimo (last) o tutti (True) i duplicati.

La funzione drop_duplicates() restituisce una copia di un frame o di una serie, dopo aver eliminato i duplicati da tutte le colonne o da un loro sottoinsieme (una lista di nomi di colonne). Il parametro opzionale keep controlla se deve essere eliminato il primo (first), l'ultimo (last) o tutti (True) i duplicati. Il parametro opzionale inplace=True rimuove i duplicati dall'oggetto originario.

Unità 35 - Ordinamento e descrizione dei dati

Il fatto di avere i dati in un frame non è sufficiente. In genere dobbiamo metterli in fila e studiarli. L'"ordinatore" universale di Python, `len()`, e i suoi "assistenti", `min()` e `max()`, sono un buon punto di partenza, ma spesso vogliamo rispondere a domande più articolate delle semplici "Quanti?" e "Quanto?". `pandas` offre varie funzioni per l'ordinamento, la catalogazione, il conteggio, la verifica dell'appartenenza e il calcolo statistico.

Ordinamento e classificazione

Le serie e i frame possono essere ordinati per indice o per valore (o più valori). La funzione `sort_index()` restituisce un frame ordinato in base all'indice (ma non funziona per le serie). L'ordinamento è sempre lessicografico (numerico per i numeri, alfabetico per le stringhe), controllato tramite il parametro `ascending` (valore di default: `True`). L'opzione `inplace=True`, come sempre, chiede che `pandas` ordini il frame originario.

```
population.sort_index().head()
```

```
→            Population
→ State
→ Alabama       4708708
→ Alaska         698473
→ Arizona       6595778
→ Arkansas      2889450
→ California   36961664
```

La funzione `sort_values()` restituisce un frame o una serie ordinata in base ai valori. Nel caso di un frame, il primo parametro è il nome di una colonna o una lista di nomi di colonne e il parametro opzionale `ascending` può essere un valore booleano o una lista di valori booleani (uno per ogni colonna usata per l'ordinamento). Il parametro `na_position` (`first` o `last`) specifica dove collocare i "valori" `nan` (all'inizio o alla fine).

```
population.sort_values("Population").head()
```

```
→                    Population
→ State
→ Wyoming                544270
→ District of Columbia   599657
→ Vermont                621760
```

→ **North Dakota 646844**
→ **Alaska 698473**

Ora, così, sapete che lo Stato meno popoloso degli Stati Uniti è il Wyoming.

La funzione rank() calcola la "classifica" numerica di un frame o di una serie. Se vi sono più valori uguali, la funzione assegna loro un valore intermedio. Il parametro booleano numeric_only limita la classifica ai soli valori numerici. Il parametro na_option (con valori top, bottom o keep) specifica come trattare i "valori" nan: rispettivamente in cima, in fondo o alla posizione che avevano nel frame originario.

```
pop_by_state = population.sort_index()
pop_by_state.rank().head()
```

→ **Population**
→ **State**
→ **Alabama 29**
→ **Alaska 5**
→ **Arizona 38**
→ **Arkansas 20**
→ **California 51**

Ora, con poca fatica, è possibile unire questo output con il frame originario della popolazione e ottenere un unico frame con la popolazione e la posizione in questa "classifica".

Statistica descrittiva

Le funzioni di statistica descrittiva calcolano la somma, sum(), la media, mean(), la mediana, median(), la deviazione standard, std(), il numero, count() e il minimo e massimo, min() e max() di una serie o di ogni colonna di un frame. Ognuna di esse accetta il parametro booleano skipna, che specifica se i "valori" nan devono essere esclusi dall'analisi e il parametro axis, che dice alla funzione come procedere (vertically o horizontally).

```
alco2009.max()
```

→ **Beer 1.72**
→ **Wine 1.00**
→ **Spirits 1.82**
→ **dtype: float64**

```
alco2009.min(axis=1)
```

→ **State**
→ **Alabama 0.22**
→ **Alaska 0.54**
→ **Arizona 0.38**
→ **Arkansas 0.17**

```
→ California   0.55
→ dtype: float64
```

```
alco2009.sum()
```

```
→ Beer      63.22
→ Wine      19.59
→ Spirits   41.81
→ dtype: float64
```

Le funzioni argmax() (per le serie) e idxmax() (per i frame) restituiscono l'indice (la posizione) della prima occorrenza dei valori massimi. È facile ricordarsi di queste due funzioni: sono le uniche due funzioni di pandas che non trattano le serie e i frame in modo coerente.

pandas offre un supporto limitato per la pseudo-integrazione, la pseudo-differenziazione e altri metodi cumulativi. Le funzioni cumsum(), cumprod(), cummin() e cummax() calcolano rispettivamente la somma, il prodotto, il minimo e il massimo cumulativo, partendo dal primo elemento della serie o in ciascuna colonna del frame. Per esempio, in questo modo potete ottenere con cumsum() il consumo cumulativo di alcol nelle Hawaii (o in qualsiasi altro Stato):

```
alco.ix['Hawaii'].cumsum().head()
```

```
→       Beer  Wine  Spirits  Total
→ Year
→ 1977   1.61  0.36     1.26   3.23
→ 1978   2.99  0.82     2.56   6.37
→ 1979   4.59  1.26     3.84   9.69
→ 1980   6.24  1.72     5.05  13.01
→ 1981   7.98  2.16     6.21  16.35
```

La funzione diff() calcola la differenza progressiva fra due elementi consecutivi. Il risultato per la prima riga è indefinito. Con diff() potete calcolare la variazione nel consumo annuo di alcol, magari ancora nelle Hawaii:

```
alco.ix['Hawaii'].diff().head()
```

```
→       Beer  Wine  Spirits         Total
→ Year
→ 1977   NaN   NaN      NaN           NaN
→ 1978 -0.23  0.10     0.04 -9.000000e-02
→ 1979  0.22 -0.02    -0.02  1.800000e-01
→ 1980  0.05  0.02    -0.07 -4.440892e-16
→ 1981  0.09 -0.02    -0.05  2.000000e-02
```

Il nome dell'ultima colonna è fuorviante, dopo aver eseguito una pseudo-differenziazione. Potete cambiarlo in "Δ(Totale)" (variazione rispetto al totale) o qualcos'altro per evitare ogni equivoco in futuro.

Unicità, conteggio, appartenenza

numpy può trattare gli array come insiemi (come discusso nell'Unità 28, *Trattare gli array come se fossero insiemi*). Anche pandas può trattare le serie (ma non i frame) come insiemi. Rispolveriamo l'esempio di pseudo-bioinformatica dell'Unità 28 per svolgere qualche operazione più complessa:

```
dna = "AGTCCGCGAATACAGGCTCGGT"
dna_as_series = pd.Series(list(dna), name="genes")
dna_as_series.head()
```

```
→ 0    A
→ 1    G
→ 2    T
→ 3    C
→ 4    C
→ Name: genes, dtype: object
```

Le funzioni unique() e value_counts() restituiscono, rispettivamente, un array di valori distinti (tratti dalla serie) e un frame con il conteggio per ogni singolo valore (potete confrontarle con Counter, Unità 7). Se la serie contiene "valori" nan, anch'essi vengono inclusi nel conteggio.

```
dna_as_series.unique()
```

```
→ array(['A', 'G', 'T', 'C'], dtype=object)
```

```
dna_as_series.value_counts().sort_index()
```

```
→ A    5
→ C    6
→ G    7
→ T    4
→ Name: genes, dtype: int64
```

La funzione isin() è definita per tutti i principali tipi di dati pandas. Restituisce una serie booleana o un frame delle stesse dimensioni il quale specifica se ogni elemento della serie/del frame è un membro di una determinata collezione. Sappiamo che in una sequenza di DNA sono ammessi solo i nucleotidi A, C, G e T. Possiamo dire se tutti i nostri nucleotidi sono validi?

```
valid_nucs = list("ACGT")
dna_as_series.isin(valid_nucs).all()
```

```
→ True
```

A questo punto, forse vi sentirete ansiosi di mettere alla prova queste nuove conoscenze su numeri reali, per ottenere qualche interessante risultato. Gli strumenti per trasformare i dati sono descritti a partire dalla prossima pagina, nell'Unità 36.

Unità 36 - Trasformazione dei dati

Ora abbiamo tutti gli elementi per affrontare le funzionalità più interessanti di pandas: l'aritmetica vettoriale, le operazioni logiche e altri meccanismi di trasformazione dei dati.

Operazioni aritmetiche

pandas supporta le quattro operazioni aritmetiche (somma, sottrazione, moltiplicazione e divisione) e le funzioni universali (ufunc) numpy (trattate nell'Unità 25, *Le funzioni universali*). Tali operatori e funzioni possono essere usati per combinare frame delle stesse dimensioni e struttura, colonne di frame e serie e anche serie delle stesse dimensioni.

Finalmente possiamo anche correggere la colonna Total del frame alco:

```
alco["Total"] = alco.Wine + alco.Spirits + alco.Beer
alco.head()
```

```
→              Beer  Wine  Spirits  Total
→ State    Year
→ Alabama  1977  0.99  0.13    0.84   1.96
→          1978  0.98  0.12    0.88   1.98
→          1979  0.98  0.12    0.84   1.94
→          1980  0.96  0.16    0.74   1.86
→          1981  1.00  0.19    0.73   1.92
```

Se volete misurare i consumi totali su una scala logaritmica, numpy offre log10(), log() e molte altre:

```
np.log10(alco.Total).head()
```

```
→ State    Year
→ Alabama  1977  0.292256
→          1978  0.296665
→          1979  0.287802
→          1980  0.269513
→          1981  0.283301
→ Name: Total, dtype: float64
```

Tutte le operazioni aritmetiche mantengono l'indicizzazione. Questa funzionalità è chiamata *allineamento dei dati*: se sommate due serie, pandas aggiunge un elemento con l'indice "C" in una serie per denominare l'elemento analogo nell'altra serie. Se questo elemento "di ponte" non esiste, il risultato è un nan. Elaboriamo geneticamente altri due frammenti di DNA rimuovendo i nucleotidi C e T dal frammento originale presentato al

termine dell'Unità precedente e poi contiamo i nucleotidi, per tipo, in entrambi i frammenti:

```
dna = "AGTCCGCGAATACAGGCTCGGT"
dna1 = dna.replace("C", "")
dna2 = dna.replace("T", "")
dna_as_series1 = pd.Series(list(dna1), name="genes") # Togliamo i C
dna_as_series2 = pd.Series(list(dna2), name="genes") # Togliamo i T
dna_as_series1.value_counts() + dna_as_series2.value_counts()
```

```
→ A    10
→ C   NaN
→ G    14
→ T   NaN
→ Name: genes, dtype: float64
```

Controllate i dati. Forse è il momento di eliminare alcuni dati mancanti (Unità 33, *Gestione dei dati mancanti*) prima di svolgere l'aggregazione dei dati?

Aggregazione dei dati

L'aggregazione dei dati è una procedura a tre passi durante la quale i dati vengono suddivisi, aggregati e ricombinati.

1. Nel passo di suddivisione, i dati vengono frammentati sulla base della chiave (o delle chiavi).
2. Nel passo dell'applicazione effettiva, su ogni frammento viene eseguita una funzione di aggregazione (come `sum()` o `count()`).
3. Nel passo di ricombinazione, i risultati calcolati vengono ricombinati in una nuova serie o frame.

La potenza di `pandas` risiede nella funzione `groupby()` e in una ricca collezione di funzioni di aggregazione, che svolgono automaticamente i tre passi elencati, consentendoci di ricevere il risultato in tutta tranquillità.

La funzione `groupby()` suddivide un frame suddividendo le righe in gruppi sulla base dei valori contenuti in una o più chiavi categoriche. La funzione restituisce un generatore di gruppi che potete usare in un ciclo (per accedere al contenuto dei gruppi) o insieme a una funzione di aggregazione.

La lista delle funzioni di aggregazione include: `count()` (restituisce il numero di righe presenti nel gruppo); `sum()` (restituisce la somma delle righe numeriche nel gruppo); `mean()`, `median()`, `std()` e `var()` (che restituiscono le rispettive

misure statistiche del gruppo, ovvero media, mediana, deviazione standard e varianza); min() e max() (restituiscono la riga più piccola e più grande nel gruppo); prod() (restituisce il prodotto delle righe numeriche del gruppo); first() e last() (restituiscono la prima e l'ultima riga del gruppo; queste ultime funzioni hanno senso solo se il frame è ordinato).

Proviamo ad applicare queste funzioni ai dati sul consumo di alcol:

```
# Vogliamo raggruppare i dati in base alla colonna "Year"
alco_noidx = alco.reset_index()
sum_alco = alco_noidx.groupby("Year").sum()
sum_alco.tail()
```

```
→ Year  Beer  Wine  Spirits  Total
→ 2005  63.49 18.06  38.89  120.44
→ 2006  64.37 18.66  40.15  123.18
→ 2007  64.67 19.08  40.97  124.72
→ 2008  64.67 19.41  41.59  125.67
→ 2009  63.22 19.59  41.81  124.62
```

Se per la suddivisione usate più di una colonna, il risultato avrà un multiindice, i cui livelli corrispondono alle colonne utilizzate.

Potete anche accedere al contenuto di ogni gruppo con un'iterazione sui gruppi, in un ciclo for. A ogni iterazione, il generatore restituito dalla funzione groupby() fornisce l'indice e il gruppo di righe corrispondente alla voce (come un frame):

```
for year, year_frame in alco_noidx.groupby("Year"):
  «qualche_operazione(year, year_frame)»
```

Talvolta vorreste poter raggruppare le righe in base a una proprietà calcolabile, invece che su una colonna esistente (o anche più colonne). pandas consente di aggregare i dati attraverso associazioni che fanno uso di dizionari o serie. Consideriamo un dizionario che crei una associazione fra stati e regioni geografiche statunitensi (https://www2.census.gov/geo/docs/maps-data/maps/reg_div.txt).

```
state2reg
```

```
→ {'Idaho': 'West', 'West Virginia': 'South', 'Vermont': 'Northeast', «...»}
```

In tal modo potrete calcolare il consumo medio di alcol per area geografica. Ricordate che il dizionario opera sulle etichette dell'indice della riga, non sui valori, pertanto assegnate la colonna appropriata, quale indice del frame (almeno temporaneamente, per la durata dell'operazione di raggruppamento). I valori del dizionario (che incidentalmente

coincidono con i nomi dei gruppi) formeranno così l'indice del frame restituito:

```
alco2009.groupby(state2reg).mean()

→              Beer     Wine  Spirits
→ Midwest     1.324167 0.265000 0.822500
→ Northeast   1.167778 0.542222 0.904444
→ South       1.207500 0.275625 0.699375
→ West        1.249231 0.470769 0.843846
```

Usando la terminologia attribuita, erroneamente, a Guglielmo di Ockham, frate francescano, filosofo scolastico e teologo inglese (https://en.wikipedia.org/wiki/William_of_Ockham), l'*aggregazione dei dati riunisce le entità* e, pertanto, è una cosa buona. La discretizzazione, al contrario, moltiplica le entità: converte un valore in più categorie. Sarebbe una cosa negativa, se non fosse per il fatto che è estremamente utile.

Discretizzazione

Con il termine discretizzazione si fa riferimento alla conversione di una variabile continua in una variabile discreta (*categorica*), spesso con lo scopo di eseguire rappresentazioni grafiche a istogramma o attività di machine learning (vedi il Capitolo 10, *Machine Learning*).

La funzione `cut()` suddivide il primo parametro (un array o una serie) in categorie semiaperte. Il secondo parametro è un numero di categorie equamente spaziate o una lista di valori estremi delle categorie. Se l'obiettivo è quello di suddividere la sequenza in `N` categorie, potete passarle una lista di `N + 1` valori estremi. Le categorie prodotte da `cut()` appartengono a un tipo di dati ordinale: potete classificarle e confrontarle fra loro.

```
cats = pd.cut(alco2009['Wine'], 3).head()

→ State
→ Alabama    (0.0991, 0.4]
→ Alaska     (0.4, 0.7]
→ Arizona    (0.0991, 0.4]
→ Arkansas   (0.0991, 0.4]
→ California (0.4, 0.7]
→ Name: Wine, dtype: category
→ Categories (3, object): [(0.0991, 0.4] < (0.4, 0.7] < (0.7, 1]]
```

Se preferite specificare le etichette delle categorie, potete passare un altro parametro, opzionale, `labels` (una lista di `N` etichette, un'etichetta per categoria).

```
cats = pd.cut(alco2009['Wine'], 3, labels=("Low", "Moderate", "Heavy'"))
cats.head()
```

```
→ State
→ Alabama          Low
→ Alaska       Moderate
→ Arizona          Low
→ Arkansas         Low
→ California   Moderate
→ Name: Wine, dtype: category
→ Categories (3, object): [Low < Moderate < Heavy]
```

Se specificate `labels=False`, la funzione `cut()` numera le categorie anziché fornire loro un'etichetta e restituisce solo le informazioni di appartenenza alle categorie:

```
cats = pd.cut(alco2009['Wine'], 3, tabets=False).head()
```

```
→ State
→ Alabama      0
→ Alaska       1
→ Arizona      0
→ Arkansas     0
→ California   1
→ Name: Wine, dtype: int64
```

La funzione `qcut()` è simile a `cut()`, ma ragiona in termini di quantili, non di ampiezze delle categorie. Potete pertanto usarla per calcolare i quantili (come la mediana e i quartili).

```
quants = pd.qcut(alco2009['Wine'], 3, labels=("Low", "Moderate", "Heavy"))
quants.head()
```

```
→ State
→ Alabama          Low
→ Alaska         Heavy
→ Arizona      Moderate
→ Arkansas         Low
→ California     Heavy
→ Name: Wine, dtype: category
→ Categories (3, object): [Low < Moderate < Heavy]
```

Un altro modo per discretizzare una variabile con un piccolo numero di valori possibili (quindi una variabile che in qualche modo è già categorica!) consiste nel decomporla in un insieme di variabili *fittizie*, una variabile per ciascun valore possibile.

Una variabile fittizia è una variabile booleana che è `True` per un valore di una variabile categorica e `False` per tutti gli altri valori. Le variabili fittizie vengono usate nella regressione logistica (ne parleremo all'interno dell'Unità 49, *La regressione lineare*) e in altre forme di machine learning, di cui parleremo nel Capitolo 10, *Machine Learning*

La funzione `get_dummies()` converte un array, una serie o un frame in un altro frame con lo stesso indice dell'oggetto originario, in cui ogni colonna

rappresenta una variabile fittizia. Se l'oggetto è un frame, usate il parametro opzionale `columns` (la lista delle colonne da discretizzare).

Ricordate la suddivisione di Stati per regione geografica che abbiamo svolto qualche pagina fa? Ecco un altro modo per considerarla:

```
pd.get_dummies(state2reg).sort_index().head()
```

```
→            Midwest  Northeast  South  West
→ State
→ Alabama          0          0      1     0
→ Alaska           0          0      0     1
→ Arizona          0          0      0     1
→ Arkansas         0          0      1     0
→ California       0          0      0     1
```

Poiché ogni Stato appartiene a una e una sola regione, la somma di tutti i valori di ogni riga è sempre `1`. Questo comportamento vale per qualsiasi insieme di variabili fittizie, non solo per questo esempio sugli Stati.

Mappaggio

Il "mappaggio" è la forma più generale di trasformazione dei dati. Impiega la funzione `map()` per applicare una funzione arbitraria a un solo argomento a ogni elemento della colonna selezionata. La funzione può essere una funzione interna di Python, una funzione aggiunta da un qualsiasi modulo importato, una funzione definita dall'utente o una funzione lambda anonima.

Per esempio, possiamo creare delle abbreviazioni di tre lettere del nome degli Stati Uniti d'America.

```
with_state = alco2009.reset_index()
abbrevs = with_state["State"].map(lambda x: x[:3].upper())
abbrevs.head()
```

```
→ 0   ALA
→ 1   ALA
→ 2   ARI
→ 3   ARK
→ 4   CAL
→ Name: State, dtype: object
```

Naturalmente, il nostro tentativo di creare abbreviazioni univoche di tre lettere non ha avuto successo, ma ciò ci ha permesso di osservare in azione la funzione di mappaggio.

A differenza delle funzioni universali, a elevata ottimizzazione e parallelizzazione, la funzione passata come parametro a `map()` viene eseguita dall'interprete Python e non può essere ottimizzata. Questo rende `map()`

piuttosto inefficiente. Usatela, quindi, solo quando non vi è nessun'altra possibilità alternativa.

Tabulazione incrociata

La tabulazione incrociata calcola le frequenze dei gruppi e restituisce un frame le cui righe e colonne rappresentano i valori di due variabili categoriche (denominate "fattori"). Con il parametro opzionale `margins=True`, la funzione calcola anche i subtotali di righe e colonne.

Il seguente frammento di codice calcola le frequenze congiunte che uno Stato considerato "wine state" (ovvero con un consumo di vino superiore alla media) sia anche un "beer state" (ovvero con un consumo di birra superiore alla media):

```
wine_state = alco2009["Wine"] > alco2009["Wine"].mean()
beer_state = alco2009["Beer"] > alco2009["Beer"].mean()
pd.crosstab(wine_state, beer_state)
```

```
→ Beer  False  True
→ Wine
→ False    14    15
→ True     12    10
```

Se i valori generati nella tabella non sono molto differenti (in questo caso non lo sono), è probabile che i due fattori siano indipendenti. Torneremo su questa osservazione in *Statistica in Python*, nell'Unità 47.

Unità 37 - I/O su file in Pandas

Anche se non avete particolari motivi per usare pandas, non potete fare a meno delle sue funzionalità di input e output su file, che permettono lo scambio di dati fra, da un lato, frame e serie, e, dall'altro, file CSV, file tabulari, file a larghezza fissa, file JSON (ne abbiamo parlato nell'Unità 15, *Leggere i file JSON*), gli Appunti del sistema operativo e così via. Fra le altre cose, pandas supporta:

- l'indicizzazione automatica e l'estrazione del nome delle colonne;
- la determinazione del tipo di dati, la conversione dei dati e il rilevamento dei dati mancanti;
- l'analisi dei valori temporali;
- l'eliminazione dei dati "sporchi" (righe saltate, piè di pagina e commenti, separatori);
- la frammentazione dei dati.

Lettura di file CSV e tabulari

La funzione read_csv() legge un frame da un file CSV designato per nome o dall'handle di un file aperto. Grazie a quasi cinquanta parametri opzionali, questa funzione è un vero e proprio "coltellino svizzero" dei file CSV; è certamente consigliabile usarla al posto di reader() (ne abbiamo parlato nell'Unità 14, *Manipolazione dei file CSV*, ma ora è il momento di "disimpararla"). Ecco alcuni dei parametri opzionali più importanti.

- sep o delimiter: il delimitatore di colonna. read_csv() accetta le espressioni regolari (per esempio, r"\s+" per "un numero qualsiasi di spazi").
- header: il numero della riga da usare per i nomi delle colonne. Usate None se volete specificare una vostra lista di nomi di colonne.
- index_col: il nome della colonna da usare come indice. Se passate False, pandas genererà un indice numerico standard.
- skiprows: il numero di righe da saltare all'inizio del file o una lista di numeri di riga da saltare.

- thousands: il carattere da usare come separatore delle migliaia.
- names: una lista di nomi di colonne.
- na_values: una stringa o una lista di stringhe da usare per i dati mancanti. Se volete usare stringhe differenti per le varie colonne (per esempio, "n/a" per le stringhe e -1 per i dati numerici), potete inserirle in un dizionario, usando come chiavi i nomi delle colonne.

Proviamo a importare una lista di Stati USA e di regioni geografiche (come abbiamo già visto nell'Unità 36, *Trasformazione dei dati*). Il file CSV originale ha una struttura regolare ma sparsa:

```
Northeast,New England,Connecticut
,,Maine
,,Massachusetts
,,New Hampshire
,,Rhode Island
,,Vermont
Northeast,Mid-Atlantic,New Jersey
,,New York
,,Pennsylvania
«altri Stati»
```

Non prevede una riga di intestazioni e ha molte celle vuote, ma sappiamo come finire di compilare sia i nomi di colonne sia i dati vuoti:

```
regions = pd.read_csv("code/regions.csv",
                      header=None,
                      names=("region", "division", "state"))
state2reg_series = regions.ffill().set_index("state")["region"]
state2reg_series.head()
```

```
→ state
→ Connecticut     Northeast
→ Maine           Northeast
→ Massachusetts   Northeast
→ New Hampshire   Northeast
→ Rhode Island    Northeast
→ Name: region, dtype: object
```

state2reg è un dizionario, non una serie, ma pandas sembra avere delle funzioni di conversione letteralmente per ogni occasione:

```
state2reg = state2reg_series.to_dict()
```

```
→ {'Washington': 'West', 'South Dakota': 'Midwest', «altri Stati»}
```

La funzione to_csv() scrive su un file CSV un frame o una serie.

La funzione read_table() legge un frame da un file tabulare designato per nome o dall'handle di un file aperto. Sostanzialmente, si tratta di una read_csv() che usa come separatore standard una tabulazione al posto della virgola.

Chunking

Se volete leggere dei dati tabulari da un grosso file in frammenti, dovete ricorrere alla funzionalità di chunking. Basta passare il parametro chunksize (numero di righe) alla funzione read_csv(). Invece di leggere effettivamente le righe, la funzione restituisce un generatore che potete usare in un ciclo for.

Supponiamo che il file code/regions_clean.csv contenga gli stessi dati del file code/regions.csv, ma senza omissioni (sono specificate tutte le regioni geografiche). Immaginiamo che il file sia davvero esteso e che non vogliate leggerlo tutto in una volta. Il seguente frammento di codice crea un oggetto TextFileReader per l'iterazione e una serie "di accumulo" e poi legge il file CSV cinque righe per volta. Ogni volta, viene estratta la colonna "region" e vengono contati i valori presenti nella colonna. Il conteggio finisce nell' accumulatore. Se nell'accumulatore non esiste ancora una determinata chiave, lo si imposta semplicemente a 0 tramite il parametro opzionale fill_value, per evitare i nan.

```
chunker = pd.read_csv("code/regions_clean.csv", chunksize=5,
                header=None, names=("region", "division", "state"))
accum = pd.Series()
for piece in chunker:
   counts = piece["region"].value_counts()
   accum = accum.add(counts, fill_value=0) accum
```

```
→ Midwest    12
→ Northeast   9
→ South      17
→ West       13
→ dtype: float64
```

Lettura di altri tipi di file

La funzione read_json() tenta di leggere un frame da un file JSON. Poiché i file JSON in genere non sono tabulari ma hanno una struttura gerarchica, non sempre è possibile costringere i dati JSON in un formato rettangolare.

La funzione read_fwf() legge un frame da un file contenente dati a larghezza fissa. La funzione impiega colspecs (una lista di tuple con le posizioni start ed end+1 che individuano una colonna in una riga) o widths (una lista di larghezze delle colonne).

La funzione read_clipboard() legge il testo contenuto negli Appunti del sistema operativo e lo passa a read_table(). Potete usare questa funzione per l'estrazione

di una tabella da una pagina web tramite il passaggio dagli Appunti (con un'operazione di Copia).

Esercitazioni

I frame e le serie pandas sono comodi contenitori per i dati che aggiungono un ulteriore livello di astrazione e coerenza sopra gli array numpy. I frames e le serie sono ottimi strumenti per l'importazione e l'esportazione di dati da file tabulari; per la strutturazione, la ri-strutturazione, l'unione e l'aggregazione dei dati; e per svolgere semplici e complesse operazioni aritmetiche. A differenza dei frame del linguaggio R, le dimensioni dei frame pandas sono limitate solo dalla fantasia, e non dalle dimensioni della memoria RAM del computer.

Trappole per linci (*)

Scrivete un programma che utilizzi i dati annui canadesi relativi alle trappole per linci (http://vincentarelbundock.github.io/Rdatasets/csv/datasets/lynx.csv) e produca il numero totale di trappole per linci per decennio (dieci anni), ordinati in senso inverso (prima il decennio più "produttivo"). Il programma deve scaricare i file di dati nella directory cache, ma solo se il file non esiste ancora in tale directory. Se la directory non esiste, occorre crearla. Il programma deve salvare i risultati come un file CSV nella directory doc. Se la directory non esiste, occorre crearla.

PIL vs. Consumo di alcol (**)

Wikipedia ha una grande quantità di dati su vari aspetti demografici, incluso il consumo di alcol pro capite (https://en.wikipedia.org/wiki/List_of_countries_by_alcohol_consumption_per_capita) e il PIL nei vari paesi e le varie tossicodipendenze pro capite (https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(PPP)_per_capita). Scrivete un programma che utilizzi questi dati per incrociare il livello del PIL (sopra o sotto la media) rispetto ai livelli di consumo di alcol (sopra o sotto la media). Sulla base di tale tabella, queste due misure sembrano mostrare una correlazione?

Meteo vs. Consumo di alcol (***)

Combinate i dati storici del consumo di alcol con i dati storici meteorologici Stato per Stato (degli Stati Uniti d'America). Usate la tabulazione incrociata per stimare se le abitudini, nel bere, sono correlate con la temperatura media locale e le precipitazioni totali. In altre parole, è vero che le persone bevono di più quando piove?

Il sito web del National Climatic Data Center (https://www.ncdc.noaa.gov/cdo-web/) è un buon punto di partenza per questa ricerca sui dati storici meteorologici.

Utilizzo dei dati delle reti

> Guardando verso il basso, passò in rassegna il resto dei suoi capi di abbigliamento, che a tratti facevano pensare alla definizione che può dare un bambino di una rete: una gran quantità di buchi, tenuti insieme da una corda...
> *Morley Roberts*, Autore inglese di novelle e brevi storie

L'analisi dei dati di rete è un'area recente dell'analisi dei dati. La scienza delle reti ha tratto i suoi metodi teorici di analisi in parte dalla matematica (e in particolare dalla teoria dei grafi), in parte dalle scienze sociali e molto dalla sociologia.

Alcuni sono soliti chiamare l'analisi dei dati di rete con una formula più attuale: "analisi delle reti sociali"; e chi siamo noi per mettere in discussione questa scelta?

Dal punto di vista della scienza dei dati, una *rete* è semplicemente una collezione di oggetti interconnessi. In pratica, possiamo trattare tutti i tipi di oggetti numerici e non numerici (testuali) come se fossero reti, sempre che esista un qualsiasi modo per interconneterli.

A seconda del nostro background scientifico e del nostro campo di studio, possiamo chiamare gli oggetti della rete con il nome di "nodi", "vertici" o "attori" e chiamare le connessioni fra di essi "archi" "spigoli" "collegamenti" o "cammini". Possiamo rappresentare le reti in modo grafico e matematico sotto forma di grafi.

In questo capitolo, imparerete a creare reti a partire da dati di rete e anche da dati non provenienti da reti, a conoscere le misurazioni relative alle reti e ad analizzare le reti, e in particolare, a calcolare e interpretare le centralità dei nodi di una rete e la struttura comunitaria. Ci assisteranno il modulo standard di Anaconda networkx e il modulo community, che è necessario installare prima di tentare ogni attività di rilevamento delle comunità (https://pypi.python.org/pypi/python-louvain/0.3).

Unità 38 - Parliamo di grafi

Matematicamente parlando, un grafo è un insieme di nodi connessi da spigoli. Vi sono vari tipi di spigoli, di nodi e grafi, connessi e interpretati in modi differenti. Pertanto, prima di iniziare a utilizzare i grafi, iniziamo con alcune importanti definizioni.

Elementi, tipi e densità dei grafi

Se almeno uno spigolo del grafo è orientato (ovvero, se connette il nodo A al nodo B ma non viceversa), il grafo si dice *orientato* ed è chiamato *digrafo*. Se in un grafo vi sono spigoli paralleli (ovvero, il nodo A può essere connesso al nodo B da più di uno spigolo), il grafo è chiamato *multigrafo*. Uno spigolo che dal nodo A rimanda a se stesso è chiamato *ciclo* (loop). Un grafo senza cicli e spigoli paralleli è chiamato *grafo semplice*.

Possono essere assegnati dei pesi agli spigoli del grafo. Un peso è (normalmente, ma non necessariamente) un numero compreso fra 0 e 1, inclusi gli estremi. Maggiore è il peso, più intensa è la connessione fra i nodi. Un grafo con spigoli pesati è chiamato grafo pesato.

Il peso è un esempio di attributo di uno spigolo. Gli spigoli possono avere anche altri attributi, di tipo numerico, booleano e stringa. Anche i nodi di un grafo possono avere attributi.

Il grado di un nodo di un grafo è definito come il numero di spigoli connessi (incidenti) al nodo. Per un grafo orientato, si distigue fra *grado in ingresso* (numero di spigoli in ingresso verso un nodo) e *grado in uscita* (numero di spigoli che fuoriescono dal nodo).

La densità del grafo d ($0 \leq d \leq 1$) dice quanto si avvicina il grafo a un grafo *completo* (che contiene tutti gli spigoli possibili). Per esempio, per un grafo orientato con e spigoli e n nodi:

$$d = \frac{e}{n(n-1)}$$

Per un grafo non orientato:

$$d = \frac{2e}{n(n-1)}$$

Struttura di un grafo

I grafi e le reti che essi generano sono oggetti interessanti, diversificati e ricchi di sfaccettature. Per comprenderli al meglio, è importante conoscerne la terminologia.

Il concetto di connettività fra nodi può essere espanso osservando i cammini di un grafo. Un *cammino* è una qualsiasi sequenza di spigoli tale per cui la fine di uno spigolo è l'inizio di un altro spigolo. Quando prendiamo l'autobus dal nodo "casa" al nodo "fermata A della metropolitana", poi prendiamo un treno dal nodo "fermata A della metropolitana" al nodo "fermata B della metropolitana" e poi camminiamo dal nodo "fermata B della metropolitana" al nodo "Ufficio", completiamo un cammino del "grafo". Un cammino che non si interseca mai (ovvero che non include lo stesso nodo due volte, a eccezione del primo o ultimo nodo) è chiamato *percorso*. Un percorso chiuso forma un *ciclo*. È proprio grazie a un ciclo che torniamo a casa dopo una giornata di duro lavoro.

Così come possiamo misurare la distanza fra due oggetti nella vita reale, possiamo misurare la distanza fra due nodi qualsiasi del grafo: si tratta semplicemente del più piccolo numero di spigoli (chiamati anche "hop") di ogni percorso di connessione fra i nodi. Questa definizione non funziona nel caso dei grafi pesati, ma talvolta è possibile immaginare che il grafo non sia pesato e contare semplicemente gli spigoli. In un grafo orientato, la distanza da A a B non sempre è uguale alla distanza da B ad A. Addirittura, potremmo essere in grado di andare da A a B, ma non di ritornare indietro! Per esempio possiamo viveve la nostra vita dalla nascita alla morte, ma, a quanto pare, nessuno è mai riuscito a compiere il percorso inverso.

La massima distanza fra due nodi qualsiasi di un grafo è chiamata *diametro* (D) del grafo. A differenza dei cerchi, i grafi non hanno un'area.

Un componente connesso o, semplicemente, un componente, è un insieme di tutti i nodi di un grafo tali per cui esiste un percorso da ogni

nodo dell'insieme a ogni altro nodo dell'insieme. In un grafo orientato, si distingue fra componenti fortemente connessi (connessi da effettivi percorsi) e componenti debolmente connessi (connessi da percorsi in cui gli spigoli sono stati convertiti in spigoli non orientati).

Se un grafo ha più componenti, il componente più grande è chiamato GCC (*Giant Connected Component*). Molto spesso le dimensioni di un GCC sono notevoli. In questo caso, spesso è meglio lavorare con il solo GCC piuttosto che con l'intero grafo per evitare problemi di mancata connettività.

Talvolta due parti di un grafo sono connesse, ma la connessione è così debole che la rimozione di un singolo spigolo spezza il grafo. In questo caso, tale spigolo è chiamato *ponte*.

Una *cricca* è un insieme di nodi tale per cui ogni nodo è connesso direttamente a ogni altro nodo dell'insieme. D'Artagnan e I Tre Moschettieri formavano una cricca e lo stesso vale per ogni altro gruppo che segua il motto "Uno per tutti e tutti per uno!". La cricca più grande di un grafo si chiama *cricca massima*. Se una cricca non può essere estesa aggiungendole un altro nodo, è chiamata *cricca massimale*. Una cricca massima è sempre anche una cricca massimale, ma non sempre vale il contrario. Un grafo completo è una cricca massimale.

Una *stella* è un insieme di nodi tale per cui un nodo è connesso a tutti gli altri nodi dell'insieme, mentre gli altri nodi non sono connessi fra loro. Le stelle si ritrovano comunemente nei sistemi gerarchici multilivello (per esempio nelle aziende, nelle istituzioni militari e in Internet).

Un insieme di nodi direttamente connessi al nodo A forma il *vicinato* ($G(A)$) di A. Questo concetto è un elemento chiave nella tecnica *snowballing*, una tecnica di acquisizione dei dati in cui gli spigoli di un nodo seme selezionato casualmente sono seguiti da quelli dei suoi vicini, poi dai vicini dei vicini (vicinato di secondo livello) e così via.

Il coefficiente di clustering locale (o, semplicemente, coefficiente di clustering) del nodo A è il numero $CC(A)$ di spigoli nel vicinato di A (escludendo gli spigoli direttamente connessi ad A), diviso per il massimo numero possibile di spigoli. In altre parole, è la densità del vicinato di A escluso il solo nodo A. Il coefficiente di clustering di un nodo qualsiasi di

una stella è pari a 0. Il coefficiente di clustering di un nodo qualsiasi di un grafo completo è pari a 1. $cc(A)$ può essere usato per misurare la probabilità che G(A) sia una stella o un grafo completo.

Una *comunità* di rete è un insieme di nodi tale per cui il numero di spigoli che interconnettono tali nodi è molto maggiore del numero di spigoli che attraversano i confini della comunità. La *modularità* ($m \in [-1/2, 1]$) misura la qualità della struttura comunitaria. È definita come la frazione degli spigoli che rientrano nella comunità meno la previsione di tale frazione se gli spigoli fossero distribuiti casualmente. Un'elevata modularità ($m \approx 1$) è sintomo di una rete con comunità dense e chiaramente visibili. Identificare tali comunità è forse il risultato più importante dell'analisi dei dati di rete (ne riparleremo nell'Unità 39, *Sequenza di analisi di una rete*).

Centralità

La centralità misura l'importanza di un nodo in una rete. Vi sono vari tipi di misure di centralità, che valutano vari aspetti importanti. Per comodità, le centralità sono spesso riportate a un intervallo fra 0 (un nodo trascurabile, periferico) e 1 (un nodo importante, centrale).

Locale (degree centrality)

La centralità locale di A è il numero di nodi adiacenti di A, ovvero è il grado di A o la dimensione di G(A). Potete ridurla in scala dividendola per il massimo numero possibile di nodi adiacenti di A, ovvero $n - 1$.

Globale (closeness centrality)

La centralità globale di A è il reciproco della media del più breve percorso LBA da tutti gli altri nodi ad A:

$$cc_A = \frac{n-1}{\sum_{B \neq A} L_{BA}}$$

Globale dei flussi (betweenness centrality)

La centralità globale dei flussi di A è la frazione di tutti i percorsi più brevi fra tutte le coppie di nodi della rete, escluso solo A, che contengono A.

Autovettore (eigenvector centrality)

Centralità ad autovettore di A è definita ricorsivamente come la somma scalare delle centralità ad autovettore di tutti i nodi adiacenti di A:

$$\mathrm{ec}_A = \frac{1}{\lambda}\sum_{B \in G(A)} \mathrm{ec}_B$$

Le ultime due misure di centralità sono costose dal punto di vista computazionale e può essere poco pratico calcolarle per reti di grandi dimensioni.

Unità 39 - Sequenza di analisi di una rete

Ora che siete in possesso di tutte le definizioni e le formule necessarie, vediamo come si svolge l'analisi dei dati di una rete.

Una tipica sequenza di analisi di una rete è costituida dai seguenti passi.

1. Si inizia identificando le entità discrete e le relazioni che le legano. Le entità diverranno i nodi della rete e le relazioni diverranno gli spigoli. Se le relazioni hanno una natura binaria (presente o assente), definiranno direttamente gli spigoli della rete. Se le relazioni sono continue o discrete, ma non binarie, potete trattarle come spigoli pesati o convertirle in spigoli non pesati, ma solo se il valore della relazione raggiunge o supera una certa soglia. Quest'ultima trasformazione è chiamata *campionamento*. La soglia di campionamento viene scelta sulla base di considerazioni empiriche e pragmatiche. Se la soglia è troppo alta, la rete risulterà troppo sparsa e frammentata in tanti piccoli componenti; se invece la soglia è troppo bassa, la rete perderà ogni struttura comunitaria e diverrà un groviglio.
2. Si eseguono vari calcoli sulla rete: densità, numero di componenti, dimensioni del GCC, diametro, centralità, coefficienti di clustering e così via.
3. Vengono identificate le comunità della rete. Se la rete risulta modulare, potete assegnare delle etichette alle comunità, sostituire le comunità con dei "supernodi" e studiare la rete così generata.
4. Infine, così come in qualsiasi altro esperimento di scienza dei dati, i risultati vengono interpretati e viene prodotto un report ricco di grafici.

Il modulo networkx fornisce quasi tutto il necessario per ogni attività di studio delle reti, con un'importante eccezione: le immagini che produce sono, francamente, piuttosto patetiche. Per ottenere migliori funzionalità di

rappresentazione grafica, è meglio ricorrere a Gephi (ne parleremo all'interno dell'Unità 40).

Unità 40 - Parliamo di networkx

Il modulo networkx contiene tutti gli strumenti essenziali per la creazione, la modifica, l'esplorazione, la rappresentazione, l'esportazione e l'importazione di reti. Supporta grafi e multigrafi semplici e orientati. Imparerete a costruire e modificare una rete aggiungendo e rimuovendo nodi, spigoli e attributi, a calcolare varie misure della rete (come la centralità) e a esplorare la struttura comunitaria della rete.

Costruzione e manipolazione di una rete

Usiamo dei dati tratti da Wikipedia (https://en.wikipedia.org/wiki/List_of_countries_and_territories_by_land_borders) per costruire ed esplorare una rete di Paesi, sulla base della presenza (e lunghezza) dei confini terrestri. Il grafo della rete non è orientato; non contiene cicli, né spigoli paralleli.

```
import networkx as nx

borders = nx.Graph()
not_borders1 = nx.DiGraph()     # Solo come riferimento
not_borders2 = nx.MultiGraph()  # Solo come riferimento
```

Potete modificare il grafo di una rete aggiungendo o rimuovendo i singoli nodi o spigoli o anche interi gruppi di nodi o spigoli. Quando si rimuove un nodo, vengono rimossi anche tutti gli spigoli che da esso si dipartono. Quando si aggiunge uno spigolo, vengono aggiunti anche i suoi nodi terminali, a meno che siano già presenti nel grafo. Potete anche etichettare i nodi con numeri o stringhe:

```
borders.add_node("Zimbabwe")
borders.add_nodes_from(["Lugandon", "Zambia", "Portugal", "Kuwait", "Colombia"])
borders.remove_node( "Lugandon")
borders.add_edge("Zambia", "Zimbabwe")
borders.add_edges_from([("Uganda", "Rwanda"), ("Uganda", "Kenya"),
   ("Uganda", "South Sudan"), ("Uganda", "Tanzania"),
   ("Uganda", "Democratic Republic of Congo")])
```

Dopo aver aggiunto tutti i territori e le connessioni fra di essi, si ottiene un grafo facile da interpretare, come quello rappresentato nella seguente figura.

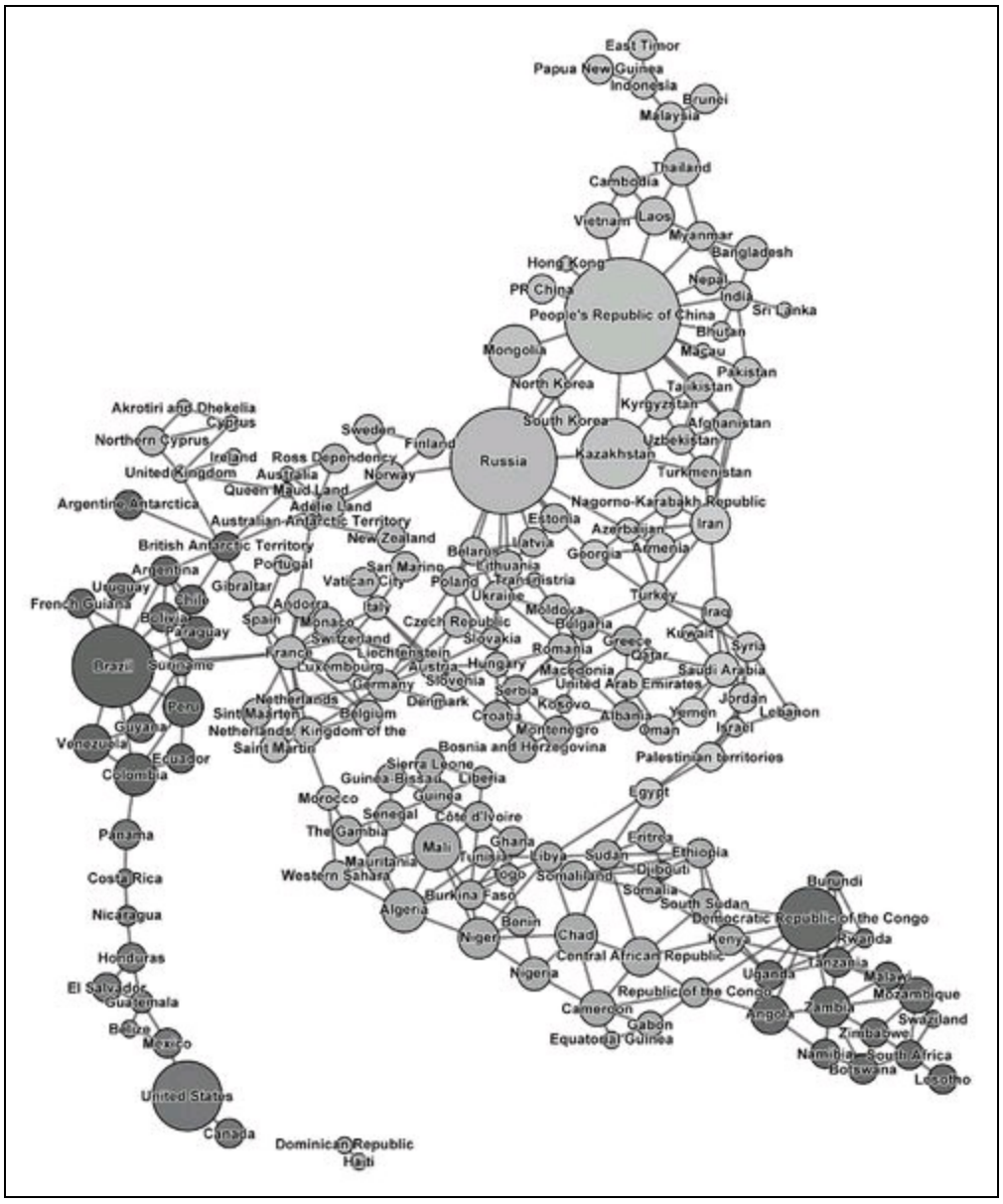


Figura 7.1

Le dimensioni dei nodi del grafo rappresentano la lunghezza totale dei confini terrestri e i vari colori (tonalità di grigio) corrispondono alle comunità della rete (di cui abbiamo parlato nell'Unità 38).

Infine, la funzione `clear()` rimuove da un grafo tutti i nodi e gli spigoli. Probabilmente non la utilizzerete molto spesso.

Esplorazione e analisi di una rete

Per analizzare ed esplorare i dati di rete con `networkx` basta richiamare alcune funzioni e considerare i valori di alcuni attributi. Data l'abbondanza di riferimenti alla funzione interna `len()` presenti in questo libro, non vi

meraviglierete neanche un po' che applicandola a un grafo ne otterrete la "lunghezza" (il numero di nodi):

```
len(borders)

→ 181
```

Le effettive liste di nodi sono disponibili tramite la funzione `nodes()` e gli attributi `node` ed `edge` (quest'ultimo attribute contiene anche i dizionari dello spigolo). Entrambi gli attributi sono read-only, di sola lettura; non potete aggiungere spigoli o nodi modificando gli attributi. Anche se ci provaste, `networkx` non applicherà gli interventi. La lista degli spigoli è disponibile anche attraverso la funzione `edges()`:

```
borders.nodes()

→ ['Iran', 'Palestinian territories', 'Chad', 'Bulgaria', 'France', «...»]

borders.node

→ {{'Iran': {'L': 5440.0}, 'Palestinian territories': {},
→  'Chad': {'L': 5968.0}, 'Bulgaria': {'L': 1808.0}, «...»}
```

I dizionari mostrano gli attributi del nodo (che nel nostro caso sono solo la lunghezza totale dei confini terrestri):

```
borders.edge

→ {'Iran': {'Nagorno-Karabakh Republic': {}, 'Turkey': {}, 'Pakistan': {},
→  'Afghanistan': {}, 'Iraq': {}, 'Turkmenistan': {}, 'Armenia': {},
→  'Azerbaijan': {}}, «...»}
```

L'attributo `edge` è un dizionario di dizionari con una voce per nodo. Contiene anche gli eventuali attributi dello spigolo (per esempio il "peso").

```
borders.edges()[:5]

→ [('Iran', 'Nagorno-Karabakh Republic'), ('Iran', 'Turkey'),
→  ('Iran', 'Pakistan'), ('Iran', 'Afghanistan'), ('Iran', 'Iraq')]
```

Per ottenere la lista dei vicini di un nodo potete usare la funzione `neighbors()`:

```
borders.neighbors("Germany")

→ ['Czech Republic', 'France', 'Netherlands, Kingdom of the', 'Denmark',
→  'Switzerland', 'Belgium', 'Netherlands', 'Luxembourg', 'Poland', 'Austria']
```

Potete anche calcolare il grado totale, il grado in uscita e il grado in entrata di un nodo dalla lunghezza dei suoi vicini o richiamando le funzioni `degree()`, `indegree()` o `outdegree()`. Richiamando queste funzioni senza parametri, esse restituiscono un dizionario dei gradi, indicizzato in base alle etichette dei nodi. Richiamandole invece usando come parametro l'etichetta di un nodo, esse restituiscono il grado di tale nodo.

```
borders.degree( "Poland")
```

→ **7**

```
borders.degree()
```

→ **{'Iran': 8, 'Nigeria': 4, 'Chad': 6, 'Bulgaria': 5, 'France': 14,**
→ **'Lebanon': 2, 'Namibia': 4, «...»}**

Pertanto, quale Paese ha i confini più lunghi?

```
degrees = pandas.DataFrame(list(borders.degree().items()),
   columns=("country", "degree")).set_index("country")
degrees.sort("degree").tail(4)
```

→ **Country**
→ **Brazil 11**
→ **Russia 14**
→ **France 14**
→ **People's Republic of China 17**

Ottima dotazione anche per reti di grandi dimensioni

Molto spesso, capita di dover lavorare con reti di dimensioni enormi (il grafo sociale di Facebook, per esempio, conta ben 1,59 miliardi di nodi). Così come tutto ciò che è implementato in Python puro, anche networkx non si distingue per le elevate prestazioni. Ecco dove entra in gioco NetworKit, un toolkit ad alta efficienza e parallelizzabile per l'analisi dei dati di rete. Gli sviluppatori di NetworKit sostengono che "il rilevamento di comunità in un grafo web con 3 miliardi di spigoli può essere svolto da un server a 16 core in una questione di minuti" (https://networkit.iti.kit.edu): un sogno per la community che ruota attorno a tale modulo. Ancora meglio a rendere ancora più attraente NetworKit vi è l'integrabilità con matplotlib, scipy, numpy, pandas e networkx.

Il coefficiente di clustering è indefinito per i grafi orientati, ma, se necessario, potete trasformare esplicitamente un grafo orientato in un grafo non orientato. La funzione clustering() restituisce un dizionario di tutti i coefficienti di clustering di ogni nodo:

```
nx.clustering(not_borders1)          # Non funziona per una rete orientata!
nx.clustering(nx.Graph(not_borders1)) # Funziona!
nx.clustering(borders)
```

→ **{'Iran': 0.2857142857142857, 'Nigeria': 0.5, 'Chad': 0.4, 'Bulgaria': 0.4,**
→ **'France': 0.12087912087912088, 'Lebanon': 1.0, 'Namibia': 0.5, «...»}**

```
nx.clustering(borders, "Lithuania")
```

→ **0.8333333333333334**

A partire da un grafo, le funzioni connected_components(), weakly_connected_components() e strongly_connected_components() restituiscono un generatore di liste dei rispettivi componenti connessi (sotto forme di liste di etichette del nodo). Potete usare il generatore in un'espressione di iterazione (un ciclo for o un manipolatore di liste) o convertirlo in una lista, grazie alla funzione list(). Se avete bisogno di usare il sottografo definito dalla lista di nodi n del grafo G,

potete estrarlo con la funzione subgraph(G, n). Alternativamente, potete usare una famiglia di funzioni connected_component_subgraphs() e così via, per calcolare i componenti connessi e poi ottenerla dal grafo come un generatore di liste:

```
list(nx.weakly_connected_components(borders))  # Non funziona!
list(nx.connected_components(borders))         # Funziona!
```

→ **[{'Iran', 'Chad', 'Bulgaria', 'Latvia', 'France', 'Western Sahara', «...»}]**

```
[len(x) for x in nx.connected_component_subgraphs(borders)]
```

→ **[179, 2]**

Grafici con Gephi

Gephi è "una piattaforma interattiva di visualizzazione ed esplorazione per ogni genere di rete e sistema complesso" (https://gephi.org). Alcuni lo chiamano "il Paintbrush dell'analisi dei dati di rete". Anche se networkx offre il supporto per la visualizzazione dei grafi (tramite matplotlib, di cui parleremo nell'Unità 41, *Tracciamento di grafici con PyPlot*), Gephi si distingue per la sua versatilità e per il suo feedback istantaneo.

Tutte le funzioni di centralità restituiscono un dizionario di centralità, indicizzato in base alle etichette dei nodi o la centralità del singolo nodo. Questi dizionari sono eccellenti elementi costitutivi per i frame di dati e le serie indicizzate pandas. I seguenti commenti indicano quale Paese ha la più elevata centralità nella rispettiva categoria:

```
nx.degree_centrality(borders)      # People's Republic of China
nx.in_degree_centrality(borders)
nx.out_degree_centrality(borders)
nx.closeness_centrality(borders)   # France
nx.franess_centrality(borders)     # France
nx.eigenvector_centrality(borders) # Russia
```

Gestione degli attributi

Un grafo networkx, con tutti i suoi nodi, spigoli e attributi, è implementato sotto forma di un dizionario. Un grafo ha un'interfaccia a dizionario verso i suoi nodi. Un nodo ha un'interfaccia a dizionario verso i suoi spigoli. Uno spigolo ha un'interfaccia a dizionario verso i suoi attributi. Potete quindi passare i nomi e valori di attributi come parametri opzionali delle funzioni add_node(), add_nodes_from(), add_edge() e add_edges_from():

```
# Attributi degli spigoli
borders["Germany"]["Poland"]["weight"] = 456.0
# Attributi dei nodi
borders.node["Germany"]["area"] = 357168
borders.add_node("Penguinia", area=14000000)
```

Quando le funzioni nodes() ed edges() vengono richiamate con il parametro opzionale data=True, restituiscono una lista di tutti i nodi o spigoli con tutti i loro attributi:

```
borders.nodes(data=True)
```

→ [«...», ('Germany', {'area': 357168}), «...»]

```
borders.edges(data=True)
```

→ [('Uganda', 'Rwanda', {'weight': 169.0}),
→ ('Uganda', 'Kenya', {'weight': 933.0}),
→ ('Uganda', 'South Sudan', {'weight': 435.0}), «...»]

Cricche e struttura comunitaria

Le funzioni find_cliques() e isolates() individuano e isolano le cricche massimali (nodi a grado zero). find_cliques() non è implementata per i grafi orientati (un digrafo viene prima convertito in un grafo non orientato); restituisce un generatore di liste di nodi, come illustrato di seguito:

```
nx.find_cliques(not_borders1)           # Non implementata per i digrafi!
nx.find_cliques(nx.Graph(not_borders1))  # Funziona!
list(nx.find_ctiques(borders))
```

→ [['Iran', 'Nagorno-Karabakh Republic', 'Armenia', 'Azerbaijan'],
→ ['Iran', 'Afghanistan', 'Pakistan'], «...»]

```
nx.isolates(borders)
```

→ ['Penguinia']

Il modulo per il rilevamento delle comunità, community, non fa parte di Anaconda, deve essere installato separatamente. Il modulo non supporta i digrafi.

La funzione best_partition() impiega il metodo di Louvain e restituisce una partizione comunitaria: un dizionario di etichette numeriche di comunità indicizzato in base alle etichette dei nodi. La funzione modularity() restituisce la modularità della partizione:

```
import community
partition = community.best_partition(borders)
```

→ {'Uganda': 0, 'Zambia': 1, 'Portugal': 2, 'Bulgaria': 2, «...»}

```
community.modularity(partition, borders)
```

→ 0.7462013020754683

Se la modularità è troppo bassa («0.5), la rete non presenta una netta struttura comunitaria (o almeno una struttura comunitaria sulla quale contare).

Input e output

Una famiglia di funzioni di I/O è in grado di leggere e scrivere dati di rete da/su file. Dobbiamo solo aprire e chiudere i file (e crearli, se necessario). Alcune funzioni richiedono che i file siano aperti in modalità binaria. La seguente tabella elenca alcune di queste funzioni.

Tabella 7.1 Alcune funzioni di I/O di networkx.

Tipo	Lettura	Scrittura	Estensione dei file
Lista adiacente	read_adjlist(f)	write_adjlist(G, f)	Non specificata
Lista degli spigoli	read_edgelist(f)	write_edgelist(G, f)	Non specificata
GML	read_gml(f)	write_gml(G, f)	.gml
GraphML	read_graphml(f)	write_graphml(G, f)	.graphml
Pajek	read_pajek(f)	write_pajek(G, f)	.net

```
with open("borders.graphml", "wb") as netfile:
   nx.write_pajek(borders, netfile)
with open("file.net", "rb") as netfile:
   borders = nx.read_pajek(netfile)
```

Non tutti i formati di file supportano tutte le funzionalità di una rete. Trovate ulteriori informazioni sui formati di file sul sito web Gephi (https://gephi.org/users/supported-graph-formats/), in modo da poter scegliere il formato di output più adatto ai vostri grafi.

Esercitazioni

L'analisi dei dati di rete è un'attività estremamente contagiosa. Una volta che imparerete a maneggiare le reti, comincerete a trovare reti ovunque, anche in Shakespeare (https://www.slideshare.net/DmitryZinoviev/desdemona-52994413). Ragionando in termini di reti, centralità e comunità potrete rendere ancora più sofisticate le vostre competenze nel campo dell'analisi dei dati, pertanto non esitate a rafforzarle, con la pratica.

Centralità e correlazioni (*)

Scaricate il grafo di una rete sociale di utenti Epinions.com dalla Stanford Large Network Dataset Collection (a cura di J. Leskovec e A. Krevl, http://snap.stanford.edu/data/soc-Epinions1.html), ed estraete la decima comunità più grande. Scrivete un programma che calcoli e visualizzi le correlazioni fra tutte le misurazioni di centralità di rete menzionate in questo capitolo (per divertirvi potete anche aggiungere un coefficiente di clustering). Vi suggerisco di memorizzare tutte le centralità in un frame di dati `pandas`. Se necessario, consultate il calcolo delle correlazioni in `pandas` in Calcolo delle misurazioni statistiche nell'Unità 47.

Esistono commie di centralità fortemente correlate?

Opere di Shakespeare (**)

Tutte le opere di William Shakespeare sono disponibili presso il MIT (http://shakespeare.mit.edu). Scrivete un programma che crei una rete di tutte le sue opere e dei 100 radicali di parole (escluse le parole di stop) più frequentemente usati. Una parola e un'opera sono connessi se il radicale viene usato nell'opera e il peso dello spigolo equivale alla frequenza d'uso. Il programma deve identificare le comunità nella rete e deve restituire la modularità e i nodi membro per ciascuna comunità.

Per quanto conoscete il lavoro di Shakespeare, il partizionamento ottenuto che senso ha?

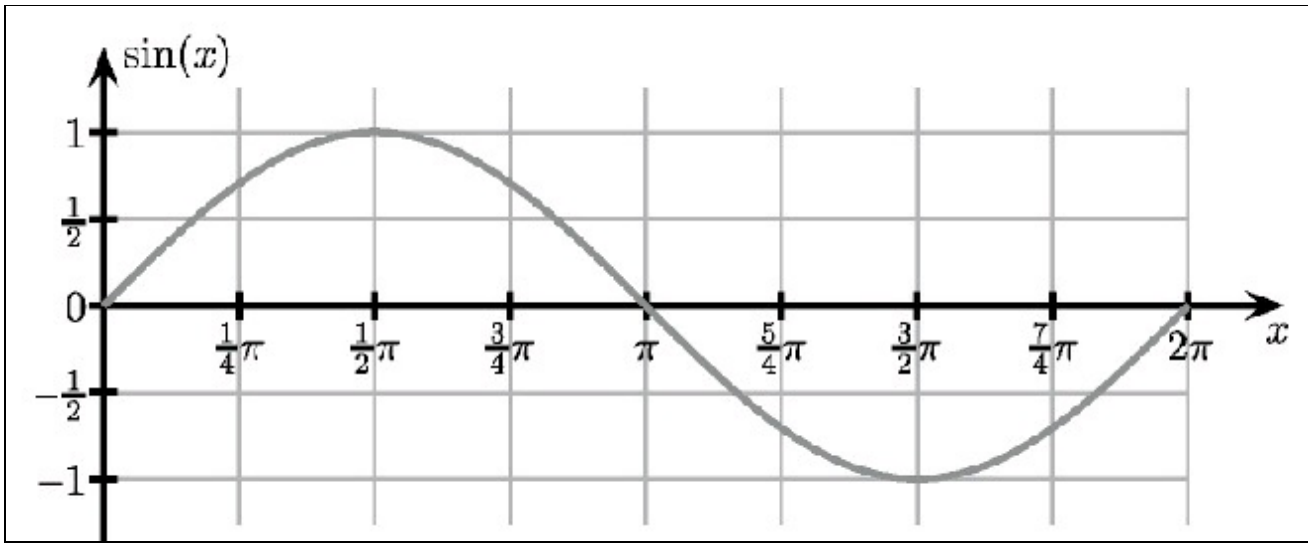
Reti e confini (**)

Usate i dati di Wikipedia, networkx e Gephi per costruire la rete di stati e confini, includendo le dimensioni dei nodi e le comunità (assegnate gli attributi dei nodi che successivamente possano essere utilizzati in Gephi per scegliere le dimensioni e il colore dei nodi).

Rappresentazione grafica

"Sto disegnando i miei disegni e ridisegnando i disegni di altri", replicò Tresham, con un tono misterioso.
William Harrison Ainsworth, storico inglese, scrittore di novelle

La rappresentazione grafica dei dati è un componente essenziale di qualsiasi attività esplorativa o predittiva di analisi dei dati e, probabilmente, anche la parte più importante dell'attività di realizzazione dei report finali. Molto onestamente, a nessuno piace leggere report senza alcuna immagine, anche quando le immagini sono irrilevanti, come questa elegante onda sinusoidale:



Vi sono sostanzialmente tre approcci al tracciamento di grafici da programma. Iniziamo un grafico *incrementale* con un'area vuota e poi aggiungiamo grafici, assi, etichette, legende e così via, usando in modo incrementale delle funzioni specializzate. Infine, presentiamo l'immagine grafica e, opzionalmente, la salviamo in un file. Fra gli esempi di strumenti di tracciamento incrementale vi è la funzione `plot()` del linguaggio R, il modulo `pyplot` del linguaggio Python e il programma di tracciamento a riga di comando `gnuplot`.

I sistemi di tracciamento *monolitico*, invece, passano tutti i parametri necessari, i quali descrivono alla funzione di tracciamento i grafici, gli assi, le etichette, le legende e così via. Tracciamo, completiamo il grafico e salviamo il risultato, in un'unica operazione. Un esempio di strumento di tracciamento monolitico è la funzione xyplot() del linguaggio R.

Infine, gli strumenti a *livelli* rappresentano tutto ciò che deve essere tracciato, il grafico e ogni funzionalità aggiuntiva, come "livelli" virtuali; aggiungiamo pertanto tutti i livelli necessari per tracciare l'oggetto. Un esempio di strumento di tracciamento a livelli è la funzione ggplot() del linguaggio R. Per motivi di compatibilità estetica, il modulo Python matplotlib offre lo stile di tracciamento ggplot.

In questo capitolo, vedremo come si svolge il tracciamento incrementale con pyplot.

Unità 41 - Tracciamento di grafici con PyPlot

Il tracciamento di grafici numpy e pandas è garantito dalle funzionalità del modulo matplotLib e, in particolare, dal sottomodulo pyplot.

Iniziamo la nostra sperimentazione con pyplot riconsiderando il rapporto sul consumo e l'abuso di alcol che all'interno dell'Unità 31 abbiamo convertito in un frame e proviamo a tracciare il consumo di alcol, suddiviso per tipo, nei vari Stati e con il trascorrere del tempo. Sfortunatamente, come accade sempre con i sistemi di disegno incrementali, l'operazione non può essere svolta da un'unica funzione, pertanto vediamo un esempio completo.

Listato 8.1 pyplot-images.py.

```
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd

# Il frame NIAAA sottoposto a pickling
alco = pickle.load(open("alco.pickle", "rb"))
del alco["Total"]
columns, years = alco.unstack().columns.levels

# Le abbreviazioni degli Stati provengono dal file
states = pd.read_csv("states.csv",
                 names=("State", "Standard", "Postal", "Capital"))
states.set_index("State", inplace=True)

# Consumo di alcol per l'anno 2009
frames = [pd.merge(alco[column].unstack(), states,
              left_index=True, right_index=True).sort_values(2009) for column in columns]

# Quanti anni considerare?
span = max(years) - min(years) + 1
```

Il primo frammento di codice importa semplicemente tutti i moduli e frame necessari. Poi combina i dati NIAAA e le abbreviazioni degli Stati in un frame e li suddivide in tre frame distinti, per tipo di bevanda alcolica. Il successivo frammento di codice si occupa del tracciamento effettivo.

Listato 8.2 pyplot-images.py.

```
# Selezionare uno stile appropriato
matplotlib.style.use("ggplot")

STEP = 5
# Traccia ogni frame in un sottografico
for pos, (draw, style, column, frame) in enumerate(zip(
[1]     (plt.contourf, plt.contour, plt.imshow),
       (plt.cm.autumn, plt.cm.cool, plt.cm.spring),
       columns, frames)):
```

```
   # Seleziona il sottografico con 2 righe e 2 colonne
[2] plt.subplot(2, 2, pos + 1)

   # Disegna il frame
[3] draw(frame[frame.columns[:span]], cmap=style, aspect="auto")

   # I raffinamenti
[4] plt.colorbar()
   plt.title(column)
   plt.xlabel( "Year")
   plt.xticks(range(0, span, STEP), frame.columns[:span:STEP])
   plt.yticks(range(0, frame.shape[0], STEP), frame.Postal[::STEP])
   plt.xticks(rotation=-17)
```

Le funzioni imshow(), contour() e contourf() (**[1]**) visualizzano la matrice, rispettivamente, come un'immagine, un grafico a contorno e un grafico a contorno pieno. Non usate queste tre funzioni (o qualsiasi altra funzione di tracciamento) nello stesso subplot, perché sovrimporrebbero nuovi grafici a quelli precedentemente tracciati (a meno che questa non sia esattamente la vostra intenzione, naturalmente. Il parametro opzionale cmap (**[3]**) specifica una tavolozza (mappa di colori) preimpostata per il grafico.

In un grafico potete anche inserire più sottografici dello stesso tipo o di tipi differenti (**[2]**). La funzione subplot(n, m, number) suddivide il grafico principale in n righe virtuali e m colonne virtuali e seleziona il sottografico number. I sottografici sono numerati da 1 e procedono colonna per colonna e poi riga per riga (il sottografico in alto a sinistra è il numero 1, quello alla sua destra è il 2 e così via). Tutti i comandi di tracciamento riguardano solo l'ultimo sottografico selezionato.

Vale la pena di notare che l'origine dell'immagine è collocata all'angolo superiore sinistro e l'asse y procede verso il basso (come sempre accade nella computer graphics), ma l'origine di tutti gli altri grafici è in corrispondenza dell'angolo inferiore sinistro e l'asse y procede verso l'alto (come sempre accade in matematica). Inoltre, per default, un'immagine e un grafico a contorno degli stessi dati hanno un rapporto dimensionale differente, ma potete accomunarli impiegando l'opzione aspect="auto".

Le funzioni colorbar(), title(), xlabel(), ylabel(), grid(), xticks(), yticks() e tick_params() (**[4]**) aggiungono al grafico le rispettive decorazioni (ne riparleremo nell'Unità 43, *Decorazione dei grafici*). La funzione grid() attiva e disattiva la rappresentazione della griglia. La presenza o meno della griglia è pertanto controllata da questa funzione, la quale è controllata dallo stile di tracciamento.

La funzione tight_layout() regola i subgrafici, migliorandone l'aspetto. Date un'occhiata ai seguenti grafici:

Listato 8.3 pyplot-images.py.

```
plt.tight_layout()
plt.savefig("../images/pyplot-all.pdf")
#plt.show()
```

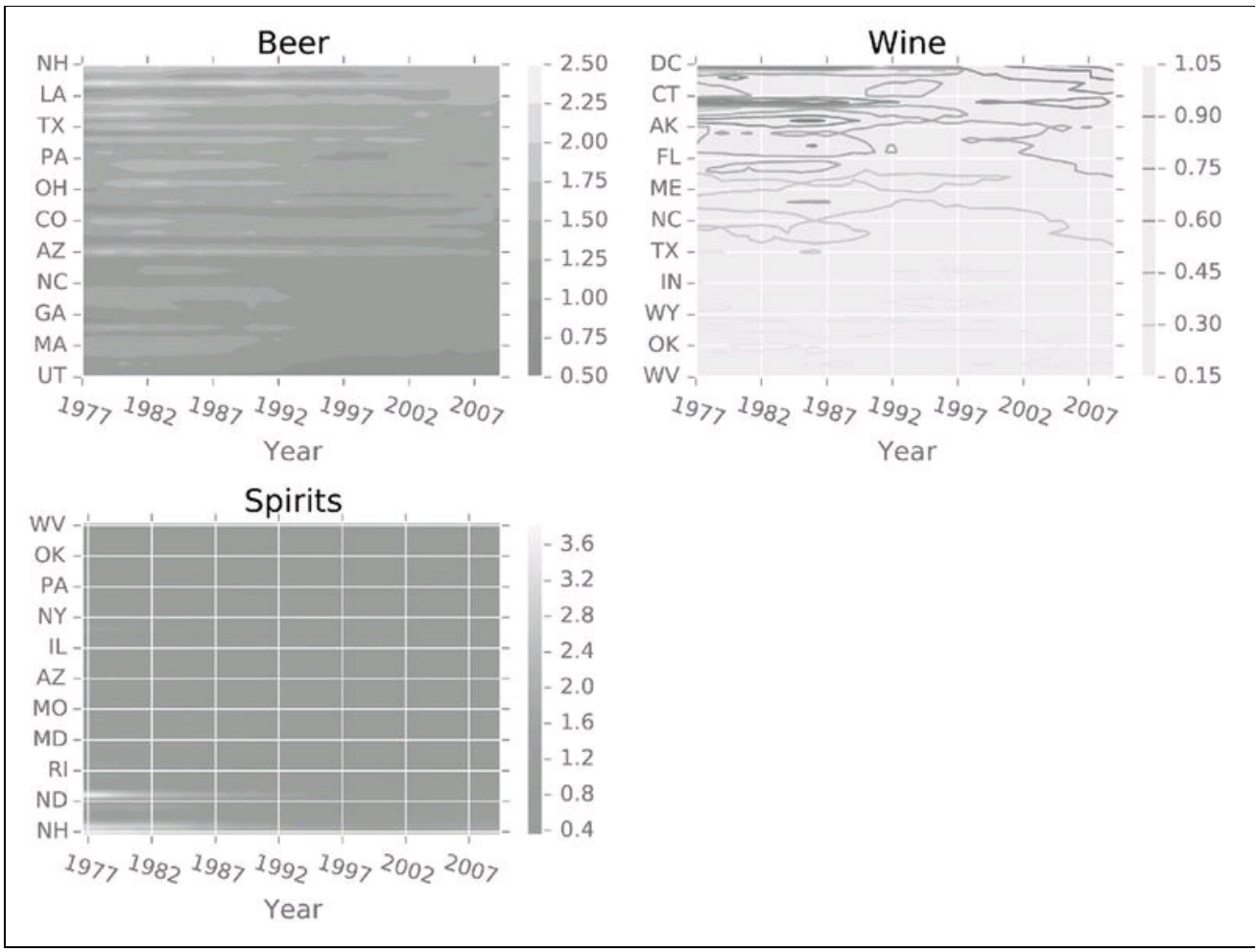


Figura 8.1

La funzione savefig() salva il grafico corrente su un file. La funzione accetta come primo parametro un nome di file o l'handle di un file aperto. Se le passate il nome del file, savefig() tenterà di indovinare il formato del file dall'estensione che avete impiegato. La funzione supporta molti classici formati di file, ma non il formato GIF.

La funzione show() mostra il grafico sullo schermo. Inoltre predispone lo spazio per il grafico, ma se volete semplicemente "ripulire" lo spazio del grafico, richiamate clf().

Unità 42 - Altri tipi di grafici

Oltre ai grafici a contorno e a immagine, pyplot supporta vari altri tipi di grafici: a barre, a scatola e baffi, a istogrammi, a torta, a linee, log e log-log, a dispersione, polari, a passi e così via. La gallery online (matplotlib.org/gallery.html) di pyplot offre molti esempi e la seguente tabella elenca molte delle funzioni grafiche di pyplot.

Tabella 8.1 Alcuni tipi di grafici di pyplot.

Tipo di grafico	Funzione
Grafico a barre verticali	bar()
Grafico a barre orizzontali	barh()
Grafico a scatola e baffi	boxplot()
Grafico a barre di errore	errorbar()
Istogramma (verticale o orizzontale)	hist()
Grafico log-log	loglog()
Grafico log in X	semilogx()
Grafico log in Y	semilogy()
Grafico a torta	pie()
Grafico a linee	plot()
Grafico per date	plot_dates()
Grafico polare	polar()
Grafico a dispersione (con controllo delle dimensioni e del colore dei punti)	scatter()
Grafico a passi	step()

Unità 43 - Decorazione dei grafici

Con pyplot, potete controllare molti aspetti di un grafico.

Potete impostare e modificare la scala degli assi (linear o log, logaritmica) con le funzioni xscale(scale) e yscale(scale) e potete impostare e modificare i limiti degli assi con le funzioni xlim(xmin, xmax) e ylim(ymin, ymax).

Potete impostare e modificare il tipo di carattere, i colori di disegno e di sfondo e lo stile e le dimensioni del testo.

Potete anche aggiungere delle note con annotate(), delle frecce con arrow() e una legenda con legend(). In generale, è opportuno far riferimento alla documentazione di pyplot per l'elenco completo delle funzioni di decorazione e i loro argomenti, ma proviamo almeno ad aggiungere delle frecce, delle note e una legenda a un grafico che ormai dovrebbe essere familiare.

Listato 8.4 pyplot-legend.py.

```
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd

# Pickling del frame
alco = pickle.load(open("alco.pickle", "rb"))

# Selezione dei dati da usare
BEVERAGE = "Beer"
years = alco.index.levels[1]
states = ("New Hampshire", "Colorado", "Utah")

# Selezione dello stile
plt.xkcd()
matplotlib.style.use("ggplot")

# Tracciamento dei grafici
for state in states:
    ydata = alco.ix[state][BEVERAGE]
    plt.plot(years, ydata, "-o")

    # Aggiunta di annotazioni e frecce
    plt.annotate(s="Peak", xy=(ydata.argmax(), ydata.max()),
                 xytext=(ydata.argmax() + 0.5, ydata.max() + 0.1),
                 arrowprops={"facecolor": "black", "shrink": 0.2})

# Aggiunta di etichette e legende
plt.ylabel(BEVERAGE + " consumption")
plt.title( "And now in xkcd...")
plt.legend(states)

plt.savefig("../images/pyplot-legend-xkcd.pdf")
```

Il grafico a tre linee rappresentato nella Figura 8.2 illustra la dinamica del consumo di birra nei tre Stati (in realtà si tratta degli stati con il

consuomo di birra massimo, minimo e mediano).

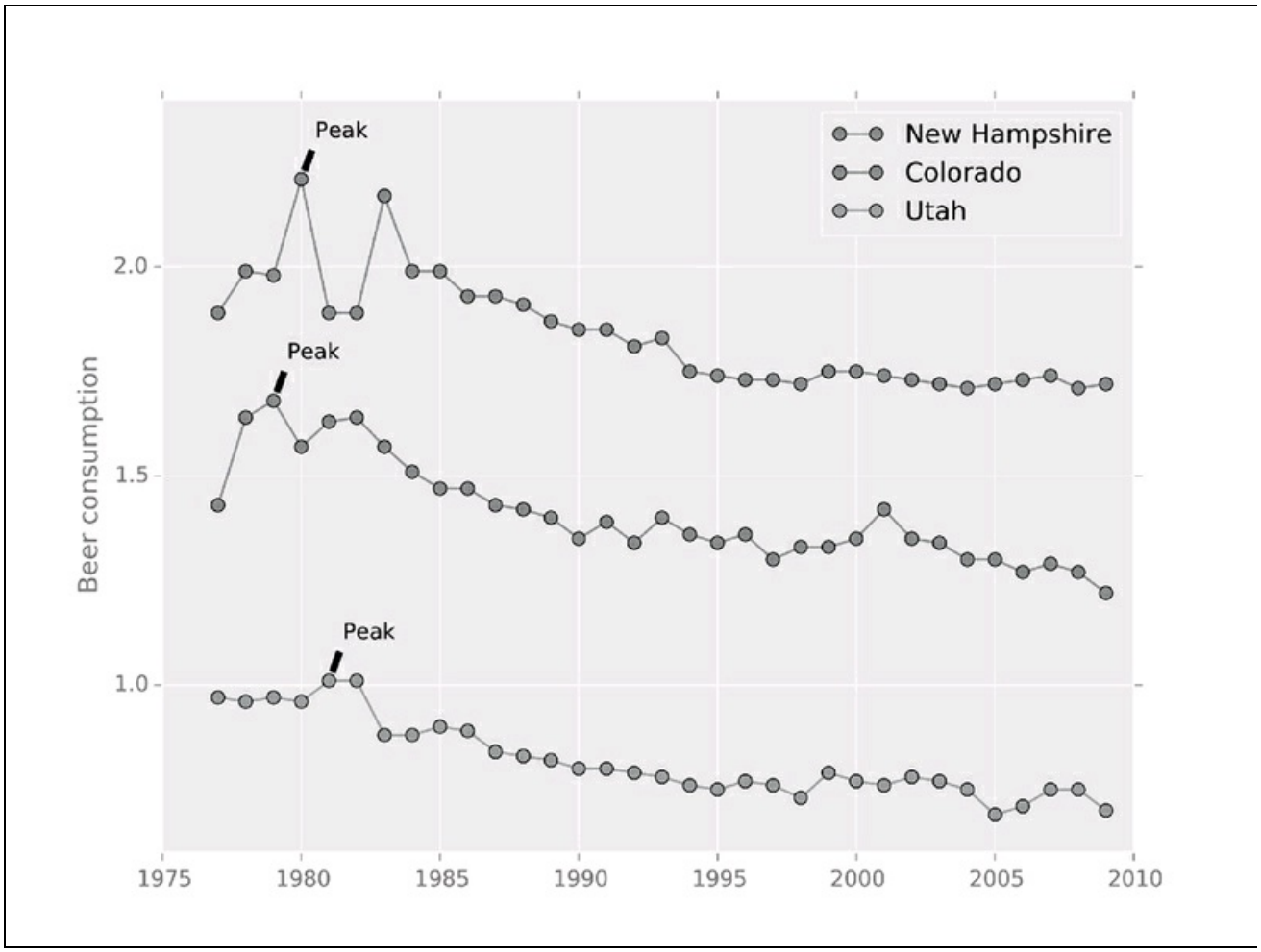


Figura 8.2

NOTA

Se il grafico contiene caratteri Unicode (non latini), potreste dover cambiare il tipo di carattere prima di aggiungere qualsiasi testo, aggiungendo una chiamata a `matplotlib.rc("font", family="Arial")` come prima riga dello script di tracciamento del grafico.

Infine, potete cambiare lo stile di un grafico `pyplot` in stile xkcd (https://xkcd.com) con la funzione `xkcd()`. La funzione influenza solo gli elementi grafici aggiunti dopo la sua chiamata. Non è possibile salvare i grafici come file PostScript, ma in quasi qualsiasi altro formato. Ciononostante, probabilmente vorrete evitare di includere grafici in stile xkcd nelle presentazioni ufficiali, perché il loro aspetto è tutt'altro che "serio" (osservate la Figura 8.3). Certo che, in un grafico sul consumo di alcol, forse le linee ondeggianti e incerte non stonano…

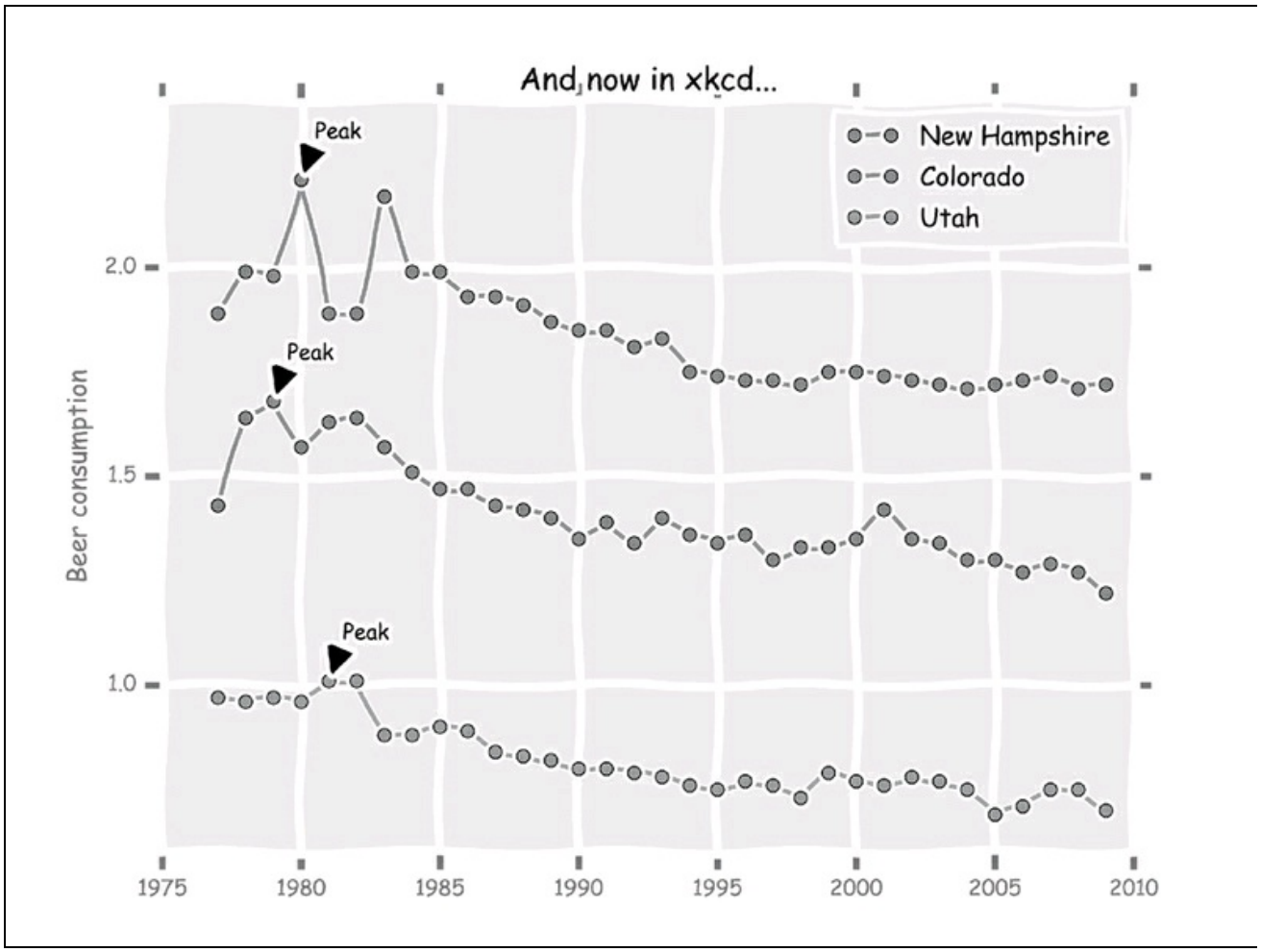


Figura 8.3

Il modulo pyplot è notevole già da solo. Ma funziona ancora meglio se combinato con pandas, come vedremo nell'Unità 44.

Unità 44 - Grafici e Pandas

Frame e serie pandas supportano entrambi il tracciamento di grafici con pyplot. Richiamando la funzione plot() senza parametri, essa traccia dei grafici a linea della serie o di tutte le colonne del frame, comprese le etichette. Specificando i parametri opzionali x e y, la funzione traccia il contenuto della colonna x rispetto a quello della colonna y.

pandas supporta anche altri tipi di grafici tramite il parametro opzionale kind. I valori consentiti per tale parametro sono bar e barh per i grafici a barre, hist per gli istogrammi, box per i grafici "a scatola", kde per i grafici a densità, area per i grafici ad area, scatter per i grafici a dispersione, hexbin per i grafici esagonali e pie per i grafici a torta. Tutti i grafici offrono vari tipi di decorazioni, come legende, barre colorate, dimensioni dei punti (opzione s) e colori (opzione c).

Per esempio, tracciamo un confronto fra il consumo di vino e di birra nello Stato del New Hampshire lungo l'intero arco delle osservazioni NIAAA. Coloreremo ogni punto dei dati in base all'anno dell'osservazione.

Listato 8.5 scatter-plot.py.

```
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd

# Pickling del frame
alco = pickle.load(open("alco.pickle", "rb"))

# Selezione dello stile
matplotlib.style.use("ggplot")

# Traccia un grafico a dispersione
STATE = "New Hampshire"
statedata = alco.ix[STATE].reset_index()
statedata.plot.scatter("Beer", "Wine", c="Year", s=100, cmap=plt.cm.autumn)

plt.title("%s: From Beer to Wine in 32 Years" % STATE)
plt.savefig("../images/scatter-plot.pdf")
```

Nel grafico rappresentato nella Figura 8.4, si può notare come tale Stato, in base alle osservazioni, abbia intrapreso un percorso di passaggio dalla birra al vino. Fortunatamente, non sembra che tale percorso sia stato doloroso, né abbia causato problemi di spopolamento.

Concluderemo questa panoramica sui grafici con il sottomodulo pandas.tools.plotting. Il modulo offre gli strumenti per il tracciamento delle curve di Andrews andrews_curves(), i grafici di correlazione lag_plot() e di autocorrelazione autocorrelation_plot(). Ma, soprattutto, pandas.tools.plotting offre gli strumenti per le matrici disperse.

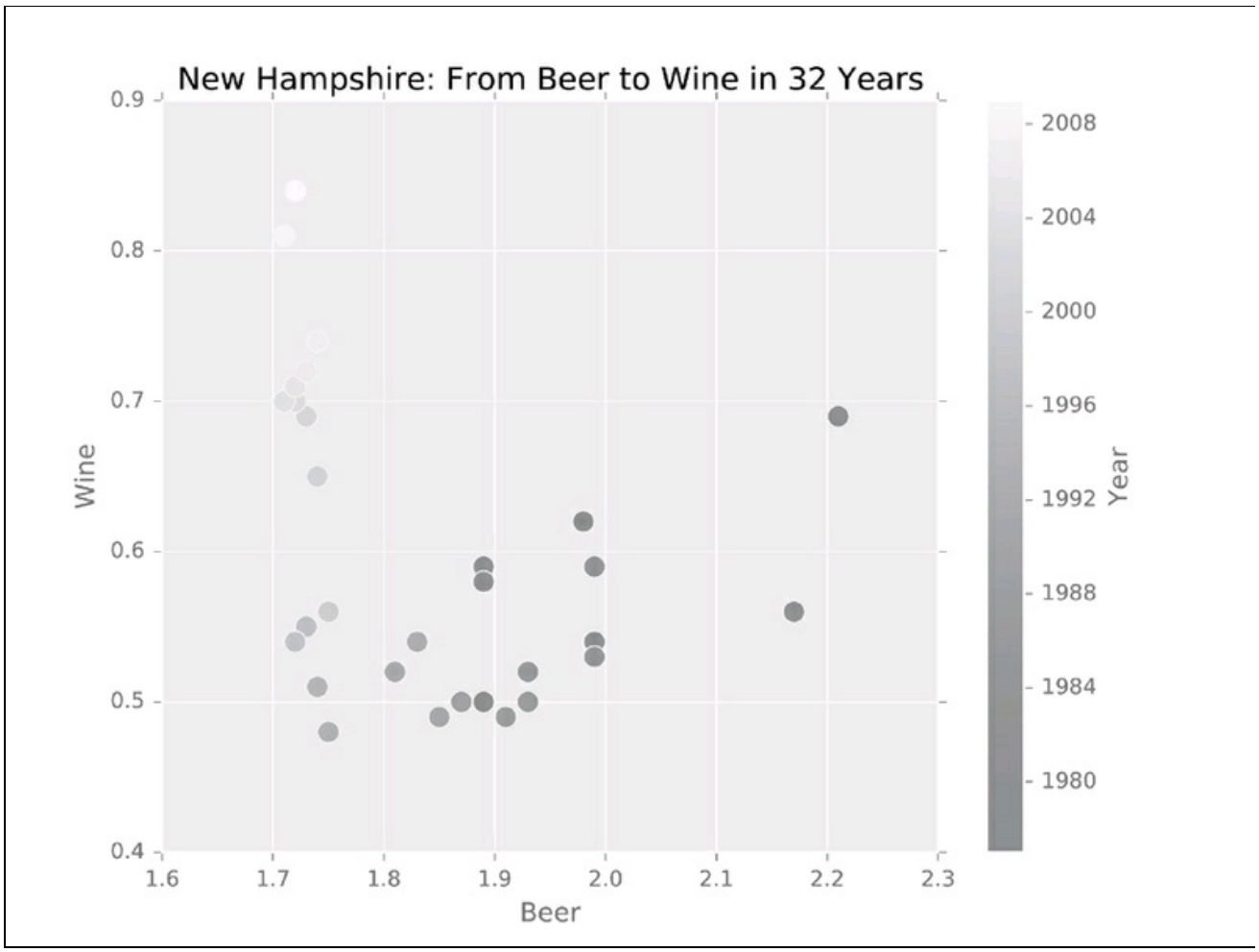


Figura 8.4

Una matrice dispersa è un eccellente strumento esplorativo. È implementato tramite la funzione scatter_matrix(), che mostra degli istogrammi dei dati per ogni colonna della diagonale principale e dei grafici a dispersione a due variabili per ogni combinazione di due colonne, in tutte le altre celle della matrice.

Listato 8.6 scatter-matrix.py.

```
from pandas.tools.plotting import scatter_matrix
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd

# Pickling del frame
alco = pickle.load(open("alco.pickle", "rb"))
```

```
# Selezione dello stile
matplotlib.style.use("ggplot")

# Tracciamento della matrice dispersa
STATE = "New Hampshire"
statedata = alco.ix[STATE].reset_index()
scatter_matrix(statedata[["Wine", "Beer", "Spirits"]],
               s=120, c=statedata["Year"], cmap=plt.cm.autumn)
plt.tight_layout()
plt.savefig("../images/scatter-matrix.pdf")
```

Parliamo ancora del New Hampshire, ed esploriamo ancora le abitudini alcoliche dei suoi abitanti, ma ora nello stesso grafico vediamo tutti e tre i tipi e tutte e sei le coppie di bevande e questo per ognuno dei trentadue anni di osservazione.

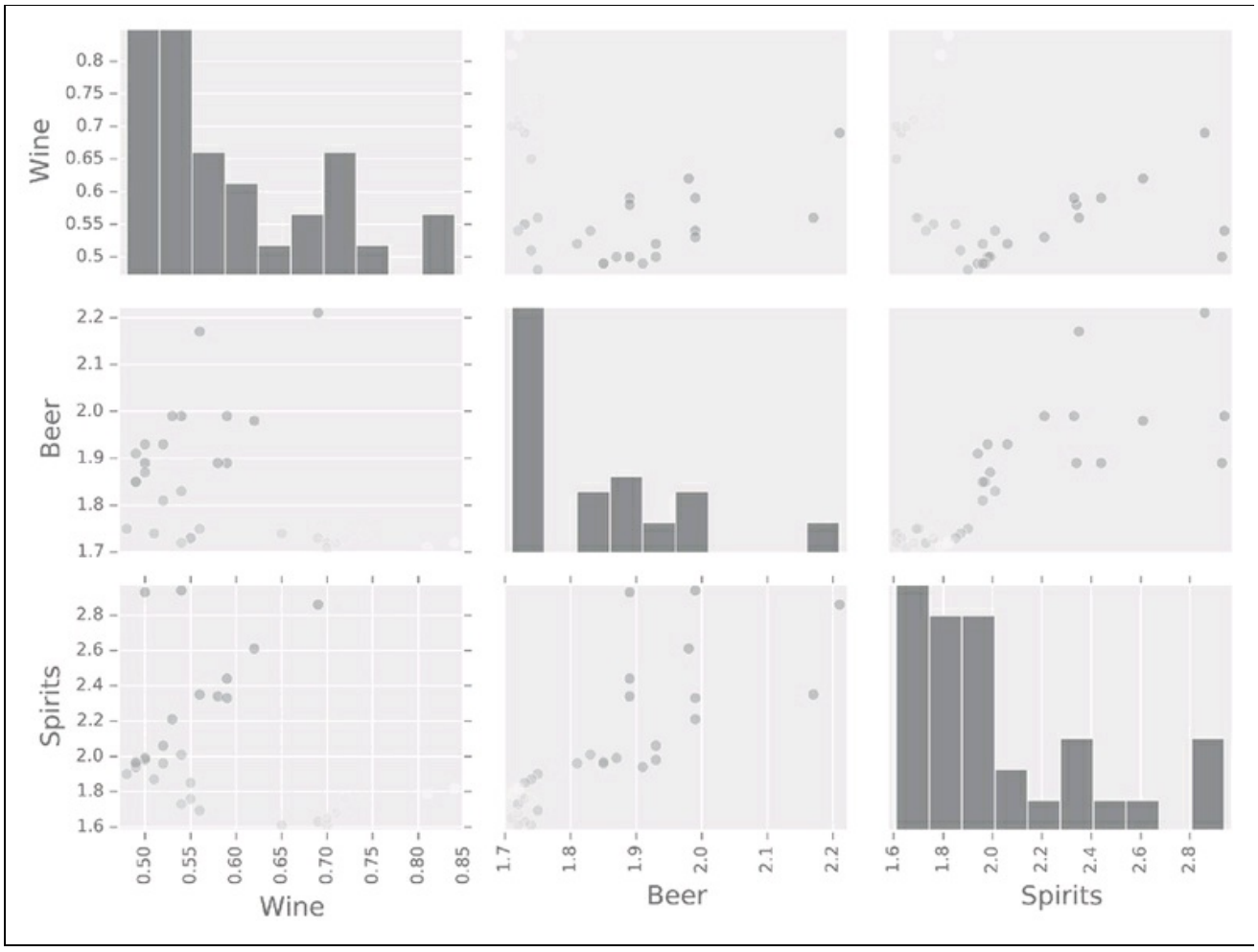


Figura 8.5

Esercitazioni

Il tracciamento dei dati (e, in generale, la visualizzazione dei dati) non è un esercizio futile. Come si sa, infatti, "un'immagine vale più di mille parole". Nella scienza dei dati, un'immagine può "valere" milioni e talvolta miliardi di osservazioni numeriche. La visualizzazione dei dati mette a disposizione molti strumenti di grande efficacia, sia in termini esplorativi (per i vostri occhi) e sia in termini descrittivi (per gli occhi di colleghi e clienti).

American Pie (*)

Scrivete un programma che visualizzi o salvi su un file PDF un grafico a torta degli Stati USA raggruppati per lettera iniziale. Per risolvere questo problema dovete partire da una lista di nomi o abbreviazioni di nomi degli Stati, che potete trarre da un sito web (http://www.stateabbreviations.us).

Popolazione della California (**)

Scrivete un programma che utilizzi i dati del Census Bureau USA (https://www.census.gov/programs-surveys/popest/data/data-sets.html) per visualizzare le variazioni nella popolazione della California (rispetto alla popolazione totale degli Stati Uniti) fra il 2001 e il 2009.

Probabilità e statistica

> Dovrei provare a ridurre questa gran massa di dati statistici ai soli fatti salienti. Troppa statistica e dati per i miei gusti!
> *Mark Twain*, scrittore e umorista americano

La teoria delle probabilità e la statistica studiano i fenomeni casuali, prevalentemente sotto forma di campioni casuali, come numeri casuali e variabili categoriche casuali.

La teoria delle probabilità considera l'origine e la produzione di campioni casuali. Possiamo trarre campioni casuali da appropriate distribuzioni delle probabilità e poi usarli per:

- simulate dati grezzi sintetizzati, da usare per il collaudo di modelli (come abbiamo fatto nell'Unità 30, in *Generare un'onda sinusoidale di sintesi*);
- suddividere i dati grezzi in set di test e di addestramento, come descritto nell'Unità 48, *Progettazione di un esperimento predittivo*;
- supportare gli algoritmi "casuali" del machine learning (come le foreste decisionali casuali, trattate nell'Unità 51, *Sopravvivere nelle foreste decisionali casuali*).

Dal canto suo, la statistica riguarda più lo studio delle proprietà dei campioni casuali precedentemente raccolti. Quasi sempre, i dati grezzi sperimentali hanno un elemento di incertezza e imprevedibilità. Useremo varie misurazioni statistiche per descrivere le osservazioni di una variabile dipendente e le possibili interazioni fra variabili dipendenti e indipendenti.

Il calcolo delle probabilità e la statistica sono aree davvero estese della matematica. È impossibile essere esaustivi in un semplice capitolo di una guida di riferimento. Ma è anche possibile che abbiate già delle conoscenze in questi due campi; questa Unità ha il solo scopo di introdurre, o "rinfrescare" i concetti chiave. Iniziamo con un po' di teoria

delle probabilità, poi procederemo con le definizioni matematiche di varie misurazioni statistiche e concluderemo calcolandole in Python.

Unità 45 - Le distribuzioni delle probabilità

Una distribuzione delle probabilità assegna una probabilità a ogni possibile numero casuale (in una distribuzione discreta) o a un intervallo di numeri casuali (in una distribuzione continua). In altre parole, ci dice se determinati risultati sono più probabili di altri risultati. Fra le distribuzioni delle probabilità più comuni vi sono la distribuzione uniforme (continua e discreta), la distribuzione normale (continua) e la distribuzione binomiale (discreta).

Potete specificare una distribuzione delle probabilità fornendo una funzione di massa della probabilità (PMF, per le distribuzioni discrete) o una funzione di densità della probabilità (PDF, per le distribuzioni continue). Una PDF descrive la probabilità relativa che una variabile casuale assuma un determinato valore e una PMF descrive la probabilità che una variabile casuale discreta sia esattamente uguale a un determinato valore. La Figura 9.1 mostra le PDF (riga superiore) e le PMF (riga inferiore) di alcune distribuzioni delle probabilità particolarmente comuni. Inutile dire che i grafici sono stati prodotti con pyplot, di cui abbiamo parlato nell'Unità 41, *Tracciamento di grafici con PyPlot*.

Osserviamo brevemente i tipi di distribuzioni più interessanti.

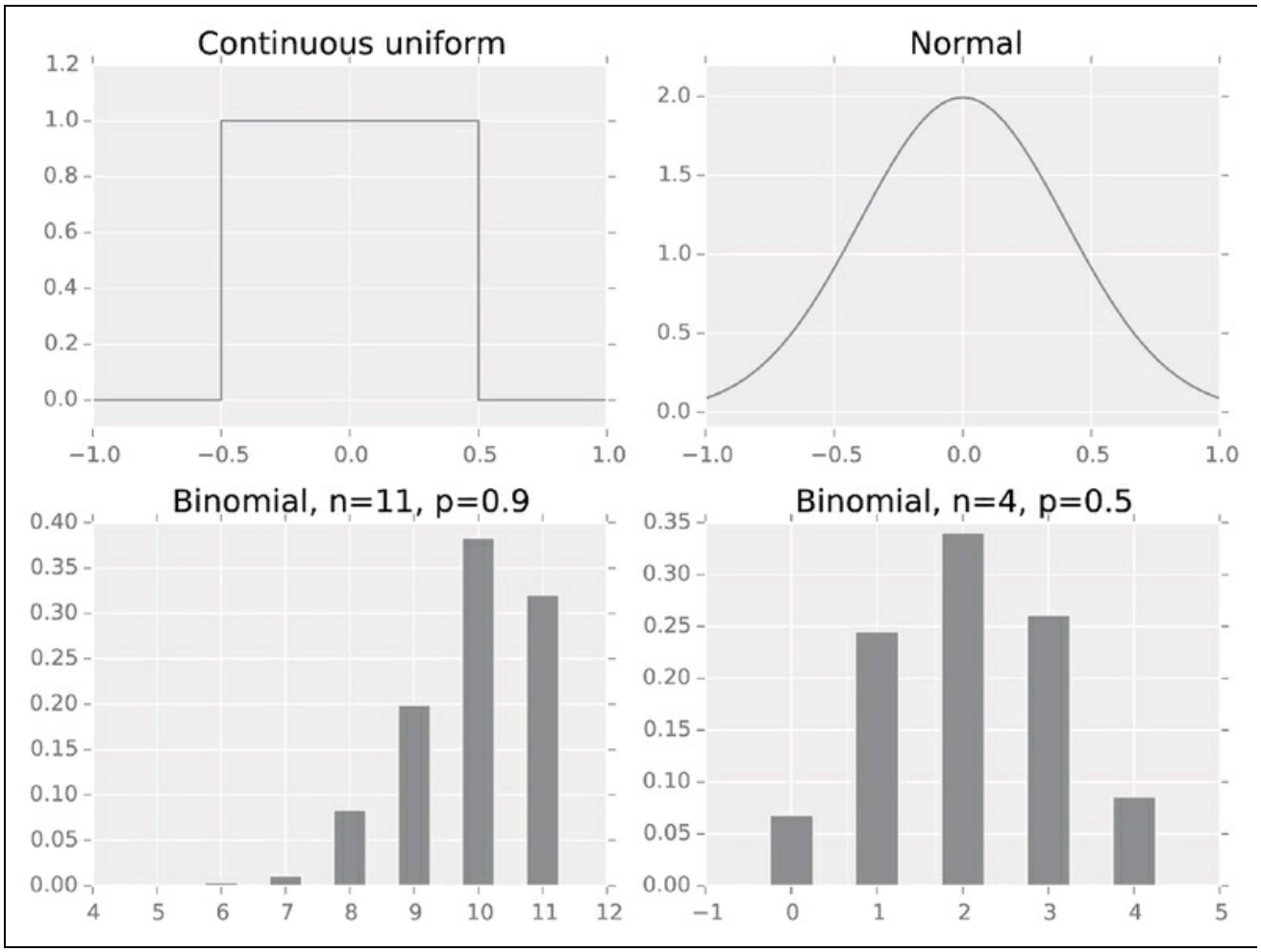


Figura 9.1

Distribuzione uniforme

La *distribuzione uniforme* è una distribuzione nella quale tutti i risultati sono ugualmente possibili. Le distribuzioni uniformi vengono spesso utilizzate per rappresentare variabili casuali di tipo numerico quando tutto ciò che si conosce è l'intervallo dei valori possibili.

Distribuzione normale

La *distribuzione normale*, nota anche con il nome di distribuzione gaussiana o a campana, viene usata per rappresentare variabili casuali di tipo reale, quando sono note la media e la deviazione standard della distribuzione, ma non la distribuzione stessa.

Distribuzione binomiale

La *distribuzione binomiale* è la distribuzione delle probabilità del numero di successi in una sequenza di n > 0 prove binarie indipendenti ("lanci di monete"), ognuno dei quali ha un successo ("testa") con una probabilità $0 \leq p \leq 1$. Il numero atteso di risultati di successo è $n \times p$. Quando n = 1, la distribuzione binomiale diviene una *distribuzione di Bernoulli*.

Una variabile casuale con distribuzione normale è una variabile a distribuzione binomiale con un numero infinito di prove e una probabilità di successo pari a 0.5. In altre parole, una variabile casuale con distribuzione normale è data dall'effetto cumulativo di un numero infinito di cause binarie equiprobabili.

Lunghe code

Alcune distribuzioni delle probabilità (come quella di Pareto, chiamata anche Zipf) ha una "lunga coda" dovuta al fatto che la PDF della distribuzione che si prolunga a destra o a sinistra, somiglia a una creatura dotata di una lunga coda. Una lunga coda significa che anche i campioni che sono lontani dalla media hanno una probabilità diversa da zero e pertanto possono comparire nei nostri dati.

I dati grezzi possono seguire o meno una di queste distribuzioni teoriche. Ma anche se non lo fanno, potete comunque trarre importanti conclusioni su di essi calcolando varie misurazioni statistiche, argomento che affronteremo nella prossima Unità.

Unità 46 - Raccolta delle misurazioni statistiche

Dal punto di vista della scienza dei dati esplorativa (ovvero non inferenziale), la statistica risponde a quattro importanti domande:

Dove sono centrati i dati?

La media del campione è la media di tutte le osservazioni:

$$\bar{x} = \sum x_i / N$$

Potete usare la media del campione per rappresentare l'intero campione quando la distribuzione dei dati è "quasi normale" (a campana) e la deviazione standard è bassa.

Quanto si estendono i dati?

La deviazione standard del campione è la misura della dispersione e viene calcolata come la radice quadrata della deviazione al quadrato dalla media rispetto alla media del campione:

$$s_x = \sqrt{\frac{\sum \left(x_i - \bar{x}\right)^2}{N-1}}$$

Un s_x elevato significa che i dati sono ampiamente dispersi.

Quanto sono asimmetrici i dati?

La *skewness* del campione rappresenta l'asimmetria della distribuzione delle probabilità. Una skewness pari a zero dà una distribuzione simmetrica. In una distribuzione unimodale (una distribuzione con una sola moda), una skewness negativa indica che la coda sul lato sinistro della funzione di densità della probabilità è più lunga di quella sul lato destro.

Due variabili sono correlate?

La covarianza del campione misura quanto due variabili casuali variano nella stessa misura. La covarianza fra X e se stessa è chiamata varianza, ed è semplicemente s^2 (il quadrato della deviazione standard).

$$q_{xy} = \frac{\sum \left(x_i - \bar{x}\right)\left(y_i - \bar{y}\right)}{N-1}$$

Il coefficiente di correlazione di Pearson, chiamato anche coefficiente di correlazione o semplicemente correlazione, è una misura normalizzata della covarianza:

$$r_{xy} = \frac{\sum \left(x_i y_i - N \bar{x} \bar{y} \right)}{(N-1) s_x s_y}$$

La correlazione è sempre compresa nell'intervallo [-1...1]. Osservate la seguente tabella. Un'elevata correlazione significa che le variabili sono correlate. Un'elevata correlazione negativa significa che le variabili sono anti-correlate. Una correlazione pari a zero significa che le variabili non sono correlate linearmente.

Tabella 9.1 Tipi di relazioni lineari fra due variabili.

	r « 0	**r = 0**	**r » 0**
p ≤ 0.01	Anticorrelazione significativa	Nessuna correlazione lineare	Correlazione significativa
p > 0.01		Nessuna correlazione lineare	

Una correlazione, anche significativa, non implica una causalità. Due variabili potrebbero essere molto correlate o anti-correlate perché entrambe sono conseguenze di una comune causa (la variabile di confusione) o anche per semplice coincidenza. Per esempio, si ha un maggior numero di annegamenti quando i giorni sono più lunghi e le notti sono più brevi; ma le persone non annegano perché i giorni sono più lunghi, ma perché in estate, quando incidentalmente i giorni sono più lunghi, aumenta il numero di bagnanti!

Analogamente, una correlazione lineare poco significativa non implica necessariamente una mancanza di relazione fra due variabili: la relazione potrebbe non essere lineare. Quando non si osserva una significativa correlazione lineare, si possono esplorare varie alternative. Considerate, per esempio, altri modelli, come il clustering, descritto nell'Unità 50, *Raggruppamento dei dati con il clustering K-Means*.

Popolazione e campioni

Anche se non siete interessati all'inferenza statistica (l'arte che consiste nel trarre, o "inferire", dai dati campionati le caratteristiche della popolazione), dovete sapere che molto spesso gli esperti di scienza dei dati non possono avere a che fare con

osservazioni relative a un'intera popolazione, ma con un campione di dimensioni limitate. Tutte le misurazioni statistiche trattate in questa Unità non sono veri valori, ma solo stime.

Quando il numero di osservazioni presenti nel campione è basso, la correlazione fra due variabili può risultare ampia, ma non necessariamente significativa. La misura della significatività è chiamata *valore-p*. Un valore-p ≤ 0.01 è considerato accettabile, ma più piccolo è questo valore e meglio è.

Possiamo ora passare a Python per imparare a trarre campioni dalle distribuzioni e a calcolare le principali misurazioni statistiche.

Unità 47 - Statistica in Python

Il supporto di Python per i numeri casuali e la statistica è suddiviso su più moduli: statistics, numpy.random, pandas e scipy.stats.

Generazione di numeri casuali

Il modulo numpy.random offre dei generatori di numeri casuali per tutte le principali distribuzioni di probabilità.

All'inizio di questo libro (Unità 2), abbiamo detto che il codice di analisi dei dati dovrebbe essere riproducibile: chiunque dovrebbe essere in grado di lanciare lo stesso programma con gli stessi dati di input e ottenere gli stessi risultati. Basta avere l'accortezza di inizializzare sempre il seme pseudo-casuale con la funzione seed(). Altrimenti, i generatori produrranno a ogni esecuzione del programma sequenze pseudo-casuali differenti, e in tal caso sarebbe difficile, se non addirittura impossibile, ottenere gli stessi risultati.

```
import numpy.random as rnd
rnd.seed(z)
```

Le seguenti funzioni generano numeri casuali interi e reali a distribuzione uniforme, normale e binomiale. Tutte restituiscono un array numpy avente una certa forma (shape) o determinate dimensioni (size), dove shape è una lista di dimensioni), a meno che il parametro size sia None.

```
rnd.uniform(low=0.0, high=1.0, size=None)
rnd.rand(shape)      # Equivale a uniform(Q.Q, 1.0, None)
rnd.randint(low, high=None, size=None)
rnd.normal(loc=0.0, scale=1.0, size=None)
rnd.randn(shape)     # Equivale a normal(0.0, 1.0, shape)

rnd.binomial(n, p, size=None)
```

La distribuzione binomiale è essenziale quando occorre configurare un esperimento predittivo e suddividere i dati in un set di addestramento e un set di collaudo (parleremo degli esperimenti predittivi nell'Unità 48, *Progettazione di un esperimento predittivo*). Supponiamo che le dimensioni relative del set di addestramento siano p e quelle del set di collaudo 1 - p. Potete preparare una sequenza binomiale, convertirla in una

sequenza booleana di valori True e False e selezionare entrambi gli insiemi da un frame pandas:

```
selection = rnd.binomial(1, p, size=len(dati)).astype(bool)
training = df[selection]
testing = df[-selection]
```

Calcolo delle misurazioni statistiche

Per offrire una soluzione rapida, il modulo statistics, che abbiamo già incontrato nell'Unità 14, offre le funzioni mean() e stdev().

I frame e le serie pandas contengono delle funzioni per calcolare le correlazioni e le covarianze rispetto ad altri frame o serie, ma anche per calcolare tutte le correlazioni e covarianze a coppie fra le colonne del frame (ma non i valori p) e altre misurazioni statistiche.

Riutilizziamo il rapporto NIAAA sull'alcolismo negli Stati Uniti, che abbiamo convertito in un frame in precedenza nell'Unità 31) per esplorare le operazioni statistiche supportate da pandas. Prepariamo due serie e poi calcoliamo la loro correlazione, la covarianza e la skewness:

```
beer_seriesNY = a\co.ix['New York']['Beer']
beer_seriesCA = alco.ix['California']['Beer']

beer_seriesNY.corr(beer_seriesCA)

→ 0.97097785391654789

beer_seriesCA.cov(beer_seriesNY)

→ 0.017438162878787872

[x.skew() for x in (beer_seriesCA, beer_seriesNY)]

→ [0.16457291293004678, 0.32838100586347024]
```

Possiamo applicare le stesse funzioni anche ai frame:

```
frameNY = alco.ix['New York']

frameNY.skew()

→ Beer 0.328381
→ Wine 0.127308
→ Spirits 0.656699
→ dtype: float64

frameNY.corr() # tutte le correlazioni a coppie

→  Beer Wine Spirits
→ Beer 1.000000 0.470690 0.908969
→ Wine 0.470690 1.000000 0.611923
→ Spirits 0.908969 0.611923 1.000000

frameNY.cov() # tutte le covarianze a coppie

→  Beer Wine Spirits
→ Beer 0.016103 0.002872 0.026020
```

→ **Wine 0.002872 0.002312 0.006638**
→ **Spirits 0.026020 0.006638 0.050888**

Le ultime due funzioni restituiscono un frame di tutte le correlazioni o covarianze a coppie.

Possiamo anche calcolare le correlazioni fra una serie e un frame o fra un frame e un altro frame. Per esempio, controlliamo se esiste una correlazione fra il consumo di alcol e la popolazione nello Stato di New York usando i dati del Census Bureau USA (https://www.census.gov/programs-surveys/popest/data/data-sets.html) nel periodo compreso fra il 2000 e il 2009:

```
# Rimuove le ultime due righe: conterranno le stime future
pop_seriesNY = pop.ix["New York"][:-2]

# Converte l'indice da una data a un intero (l'anno)
pop_seriesNY.index = pop_seriesNY.index.str.split().str[-1].astype(int)
frameNY.ix[2000:2009].corrwith(pop_seriesNY)
```

→ **Beer -0.520878**
→ **Wine 0.936026**
→ **Spirits 0.957697**
→ **dtype: float64**

Notate che le righe nel frame e nella serie sono disposte in ordine inverso. Pandas, però, è sufficientemente intelligente da usare gli indici di riga per individuare le righe corrette. Si tratta, naturalmente, del meccanismo di allineamento dei dati in azione. Ulteriori informazioni sull'allineamento dei dati si trovano nell'Unità 36, *Trasformazione dei dati*.

Per stimare la significatività della correlazione, usate la funzione pearsonr() del modulo scipy.stats. Questa funzione restituisce sia la correlazione sia il valore-p. Tuttavia, non si integra con i frame pandas e non supporta l'indicizzazione, pertanto sarà vostro compito allineare gli indici e convertire il risultato di nuovo in un frame.

```
from scipy.stats import pearsonr
# Allinea manualmente gli indici
pop_sorted = pop_seriesNY.sort_index()
alco_10 = alco.ix['New York'][-10:]

# Manipola le liste per calcolare tutte le correlazioni e i valori p
corrs = [(bev,) + pearsonr(alco_10[bev], pop_sorted)
        for bev in alco_10.columns]
# Converte la lista in un frame
pd.DataFrame(corrs, columns=("bev", "r", "p-value")).set_index("bev")
```

→ **r p-value**
→ **bev**
→ **Beer -0.520878 0.122646**
→ **Wine 0.936026 0.000068**
→ **Spirits 0.957697 0.000013**

Vale la pena di notare che il valore-p per la correlazione "beer" è

eccezionalmente alto. In base alla Tabella 9.1, *Tipi di relazioni lineari fra due variabili*, dovremmo concludere che è improbabile che esista una relazione lineare fra la numerosità della popolazione e il consumo di birra.

Ora abbiamo tutto quello che ci serve pe tornare all'esempio della *Tabulazione incrociata*, nell'Unità 36. Sulla base della tabella incrociata, abbiamo immaginato che nel 2009, i consumi di birra e vino potessero essere linearmente indipendenti. La correlazione di Pearson conferma appieno questa ipotesi:

```
alco2009.corr()
```

```
→  Beer Wine Spirits
→ Beer 1.000000 -0.031560 0.452279
→ Wine -0.031560 1.000000 0.599791
→ Spirits 0.452279 0.599791 1.000000
```

Il valore-p molto elevato della correlazione rappresenta il colpo di grazia per l'ipotesi alternativa:

```
pearsonr(alco2009["Wine"], alco2009["Beer"])
```

```
→ (-0.03156048830085 6844, 0.82598481310787297)
```

E il seguente grafico a dispersione spiega il perché: i punti sono del tutto dispersi nello spazio, senza alcun particolare allineamento o schema.

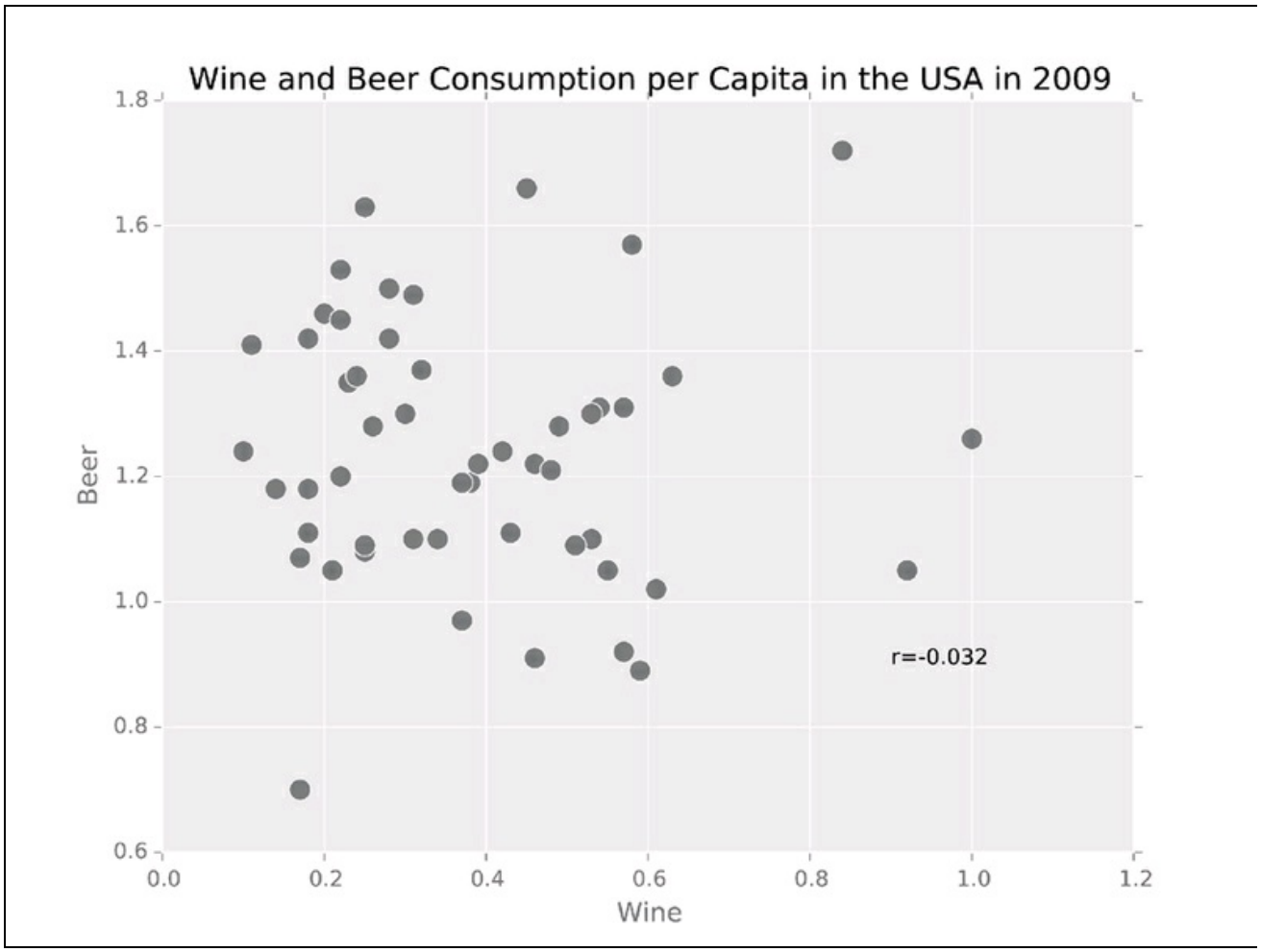

Wine and Beer Consumption per Capita in the USA in 2009
Beer
Wine
r=-0.032
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.0
0.2
0.4
0.6
0.8
1.0
1.2

Figura 9.2

Esercitazioni

Chiamare questo capitolo *Probabilità e statistica* è stato soprattutto un atto di sconsiderata spavalderia. Ci vorrebbe un intero volume, non poche pagine, anche solo per introdurre la teoria delle probabilità e poi un altro volume per trattare la statistica. Se questo è il vostro primo incontro con queste discipline e intendete davvero diventare esperti di scienza dei dati, preparatevi a studiare, e anche molto. In ogni caso, proviamo a svolgere alcuni progetti interessanti. E che Python sia sempre con voi…

S&P 500 nel Ventunesimo secolo (*)

Scrivete un programma che esegua alcune semplici misurazioni statistiche dei valori di chiusura del pacchetto di azioni S&P 500: media, deviazione standard, skewness e correlazione fra i valori di chiusura e il volume degli scambi nel ventunesimo secolo. La correlazione è affidabile? Potete scaricare i dati storici da Yahoo! Finance (https://finance.yahoo.com/q/hp?s=^GSPC+Historical+Prices). Ricordate che il Ventunesimo secolo è iniziato il 1 gennaio 2001.

Nutrienti in Rete (***)

Il database dei nutrienti dello United States Department of Agriculture (USDA) (https://www.ars.usda.gov/Services/docs.htm?docid=25700) contiene informazioni relative a circa 9000 cibi e 150 nutrienti. Supponiamo che due nutrienti siano simili se la loro quantità in tutti i cibi sono fortemente e stabilmente correlati: la correlazione è maggiore di 0.7 e il valore-p è minore di 0.01. Scrivete un programma che utilizzi i dati del file `NUT_data.txt` per costruire una rete di nutrienti simili (a questo proposito potete tornare a consultare l'Unità 40, *Parliamo di networkx*). Ogni nutriente della rete forma un nodo e due nodi sono connessi se due nutrienti sono simili.

La rete presenta una qualsiasi struttura comunitaria? In caso affermativo, quali nutrienti stanno insieme?

Machine Learning

Nella mia fabbrica di Dearborn tutto è meccanizzato.
Henry Ford, industriale americano

Il machine learning è un campo di studio che ricerca e costruisce algoritmi che sono in grado di apprendere e di effettuare previsioni su dati sperimentali. Vi sono due grandi classi di machine learning: con supervisione e senza supervisione.

L'*apprendimento con supervisione* tenta di inferire una funzione predittiva da un insieme di dati di addestramento già etichettati, ovvero un insieme di dati nel quale sappiamo già che ogni osservazione appartiene a una determinata classe (la classificazione è in realtà una parte del dataset). Parleremo della regressione lineare, compresa la regressione logistica, nell'Unità 49, *La regressione lineare*, e delle foreste decisionali casuali nell'Unità 51, *Sopravvivere nelle foreste decisionali casuali*. Per motivi di spazio, dovremo purtroppo ignorare la classificazione bayesiana naïve, le macchine vettoriali, l'analisi discriminante lineare e le reti neurali.

L'apprendimento senza supervisione tenta di trovare il senso nascosto all'interno di dati non etichettati. Alcune delle più note tecniche senza supervisione sono il clustering K-means (di cui parleremo nell'Unità 50, *Raggruppamento dei dati con il clustering K-Means*) e il rilevamento delle comunità (precedentemente trattato nell'Unità 40, *Parliamo di networkx*). Anche il clustering gerarchico e l'analisi del componente principale sono algoritmi senza supervisione, ma qui non abbiamo né il tempo né lo spazio per parlarne.

Entrambi i tipi di strumenti di machine learning possono essere usati per l'analisi esplorativa e predittiva dei dati. Ne troverete le implementazioni Python nel modulo SciKit-Learn e nei suoi sotto-moduli. Se il vostro obiettivo è quello di prevedere qualcosa che non avete visto invece di spiegare qualcosa che avete visto, dovete partire da un esperimento predittivo.

Unità 48 - Progettazione di un esperimento predittivo

L'analisi predittiva dei dati è un vero esperimento scientifico e come tale deve essere organizzato. Non potete sostenere, semplicemente, che il modello individuato per i vostri dati predica qualcosa: una parte importante di ogni esperimento consiste nel verificare e valutare il suo potere predittivo.

Per costruire, applicare e convalidare un modello, occorre seguire quattro passi.

1. Suddividere i dati di input in un set di addestramento e un set di collaudo (con una percentuale consigliata pari a 70:30). Poi mettere da parte i dati del set di collaudo, che non dovranno mai essere impiegati per preparare il modello dei dati.
2. Costruire un modello dei dati usando solo i dati di addestramento.
3. Applicare il nuovo modello ai dati di collaudo.
4. Valutare la qualità del modello con una matrice di confusione o un altro strumento di valutazione della qualità. Se il modello passa il test, è adeguato, altrimenti ripetere i tre passi precedenti.

Una matrice di confusione binaria è una tabella con due righe e due colonne che consente di valutare l'accuratezza di un modello predittivo binario (un modello che predica se una determinata proprietà vale oppure non vale), come illustrato dalla seguente tabella:

Tabella 10.1 Matrice di confusione binaria.

Classificato come		
Positivo	*Negativo*	Quando è…
Vero positivo (TP)	Falso negativo (FN)	*Positivo*
Falso positivo (FP)	Vero negativo (TN)	*Negativo*

Supponiamo di non sapere se un qualche elemento contenuto nel set di collaudo abbia la proprietà considerata e di aver usato il modello per prevedere tale proprietà per ciascun elemento. Chiaramente, questa supposizione vale solo per i modelli di apprendimento con supervisione.

TP fa riferimento al numero di elementi per i quali il modello ha previsto, correttamente, la proprietà come presente (vero positivo); TN fa riferimento al numero di elementi per i quali il modello ha previsto, correttamente, la proprietà come assente (vero negativo); FP fa riferimento al numero di elementi per i quali il modello ha previsto, sbagliando, la proprietà come presente (falso positivo); FN fa riferimento al numero di elementi per i quali il modello ha previsto, sbagliando, la proprietà come assente (falso negativo).

NOTA

Altre tecniche di machine learning con e senza supervisione comprendono la classificazione bayesiana naïve, le macchine vettoriali SVN, analisi discriminante lineare (LDA) e le reti neurali. Alcune di queste sono presenti in SciKit-Learn.

Esistono alcune misure quantitative per valutare il contenuto della matrice.

- L'*accuratezza* è la proporzione degli elementi classificati correttamente:

$$\text{accuratezza} = \frac{TP + TN}{TP + TN + FP + FN}$$

  Come minimo, il modello predittivo deve avere un'elevata accuratezza. In caso contrario, non è … accurato!
- La *precisione* è la proporzione fra i veri positivi e tutti i valori classificati come positivi:

$$\text{precisione} = \frac{TP}{TP + FP}$$

- La *sensibilità* è la proporzione fra i veri positivi e tutti i valori che sono effettivamente positivi:

$$\text{sensibilità} = \frac{TP}{TP + FN}$$

  La sensibilità descrive l'efficacia del modello nel riconosere la proprietà osservata. Se i veri positivi sono rari (per esempio l'incidenza dei casi di cancro fra la popolazione in generale), il modello deve necessariamente essere molto sensibile per poter essere utile.

- La *specificità* è la proporzione fra tutti i veri negativi e tutti i valori effettivamente negativi:

$$\text{specificità} = \frac{TN}{TN + FP}$$

  Un'elevata specificità sta a significare che il modello è efficace nel catturare l'assenza della proprietà.

Molti modelli statistici hanno un'elevata sensibilità ma una bassa specificità, o una bassa sensibilità ma un'elevata specificità, a seconda dei parametri del modello. Dovete scegliere tali parametri sulla base della misura che ritenete più importante. Se un predittore ha bassa specificità e sensibilità, potete invertire i parametri per ottenere un buon predittore.

Se il valore previsto non è binariio (ovvero è categorico o continuo), occorre usare altri strumenti di controllo della qualità, alcuni dei quali verranno trattati nel resto di questo capitolo.

Unità 49 - La regressione lineare

La regressione lineare è una forma di modellazione statistica predittiva che punta a spiegare, in toto o in parte, la varianza di una variabile usando un modello lineare. è una tecnica di modellazione con supervisione: occorre addestrare (con la funzione `fit`) il modello prima di poterlo usare per effettuare una predizione.

Regressione OLS (metodo dei minimi quadrati)

La regressione col metodo dei minimi quadrati (OLS - *Ordinary Least Squares*) mette in relazione delle variabili indipendenti (i predittori) e delle variabili dipendenti (valore previsto). Il modello tratta il valore previsto $reg(x_i)$ come una combinazione lineare dei predittori x_i. Le differenze fra i veri y_i e i valori previsti sono detti residui. Nel caso di un adattamento perfetto, tutti i residui sono pari a zero. La qualità dell'adattamento è data dalla somma SSR dei residui al quadrato, eventualmente pesati (con pesi $w_i > 0$). L'addestramento del modello ha lo scopo di minimizzare SSR:

$$\mathrm{SSR} = \sum_i^N (y_i - \mathrm{reg}(x_i))^2 w_i^2 \rightarrow \min$$

Un'altra misura della qualità dell'adattamento è lo *score* del modello, R^2. Lo score $0 \leq R^2 \leq 1$ mostra quanta varianza è in grado di descrivere il modello adattato. Nel caso di un adattamento perfetto, $R^2 = 1$. Nel caso di un pessimo adattamento, $R^2 \approx 0$.

Il costruttore `LinearRegression()` del modulo `sklearn.linear_model` crea un oggetto per regressione OLS.

La funzione `fit()` prende una matrice $1 \times n$ di predittori. Se il modello ha una variabile indipendente e i predittori formano un vettore, per creare un'altra dimensione si applica lo slicing con l'oggetto `numpy.newaxis`. I veri valori della variabile dipendente vengono passati come un vettore monodimensionale. Per prevedere più di una proprietà, dovrete costruire e adattare più di un modello.

Dopo l'adattamento, potete usare l'oggetto regressione per calcolare i valori previsti (con la funzione predict()) e per controllare la qualità dell'adattamento (con la funzione score()). Gli attributi coef_ e intercept_ contengono i valori dei coefficienti di regressione e dei punti di intercettazione dopo l'adattamento.

Il seguente esempio impiega le serie storiche dellle quotazioni di chiusura S&P 500 tratte da Yahoo! Finance (https://finance.yahoo.com/q/hp?s=^GSPC+Historical+Prices) per costruire, adattare e valutare un modello a regressione lineare. Presuppone che i dati siano stati precedentemente salvati nel file sapXXI.csv.

Iniziate importando tutti i moduli necessari e caricando i dati S&P 500.

Listato 10.1 sap-linregr.py.

```
import numpy, pandas as pd
import matplotlib, matplotlib.pyplot as plt
import sklearn.linear_model as lm

# Carica i dati
sap = pd.read_csv("sapXXI.csv").set_index("Date")
```

Come potete vedere esplorando in modo visuale le quotazioni di chiusura, il loro comportamento generale è tutt'altro che lineare, ma c'è un frammento interessante e quasi lineare, che inizia il 1 gennaio 2009 e si estende fino alla fine del dataset. Usate solo questo frammento per l'adattamento del modello. Sfortunatamente, SciKit-Learn non supporta direttamente le date. Pertanto, prima dovrete convertire i giorni in valori ordinali e poi creare, adattare e valutare il modello di regressione lineare e calcolare i valori S&P 500 previsti. Lo score del modello è pari a 0.95: non male!

Listato 10.2 sap-linregr.py.

```
# Seleziona una parte dall'aspetto linearle
sap.index = pd.to_datetime(sap.index)
sap_linear = sap.ix[sap.index > pd.to_datetime('2009-01-01')]

# Prepara il modello e lo adatta
olm = lm.LinearRegression()
x = numpy.array([x.toordinal() for x in sap_linear.index])[:, numpy.newaxis]
y = sap_linear['Close']
olm.fit(X, y)

# Predice i valori
yp = [olm.predict(x.toordinal())[0] for x in sap_linear.index]

# Valuta il modello

olm_score = olm.score(X, y)
```

Infine, il codice traccia il dataset originario, la linea di predizione e anche lo score del modello. Il risultato è rappresentato nella Figura 10.1.

Listato 10.3 sap-linregr.py.

```
# Seleziona uno stile di grafico
matplotlib.style.use("ggplot")

# Traccia entrambi i dataset
plt.plot(sap_linear.index, y)
plt.plot(sap_linear.index, yp)

# Aggiunge le decorazioni
plt.title("OLS Regression")
plt.xlabel("Year")
plt.ylabel("S&P 500 (closing)")
plt.legend(["Actual", "Predicted"], loc="lower right")
plt.annotate("Score=%.3f" % olm_score,
        xy=(pd.to_datetime('2010-06-01'), 1900))
plt.savefig("../images/sap-linregr.pdf")
```

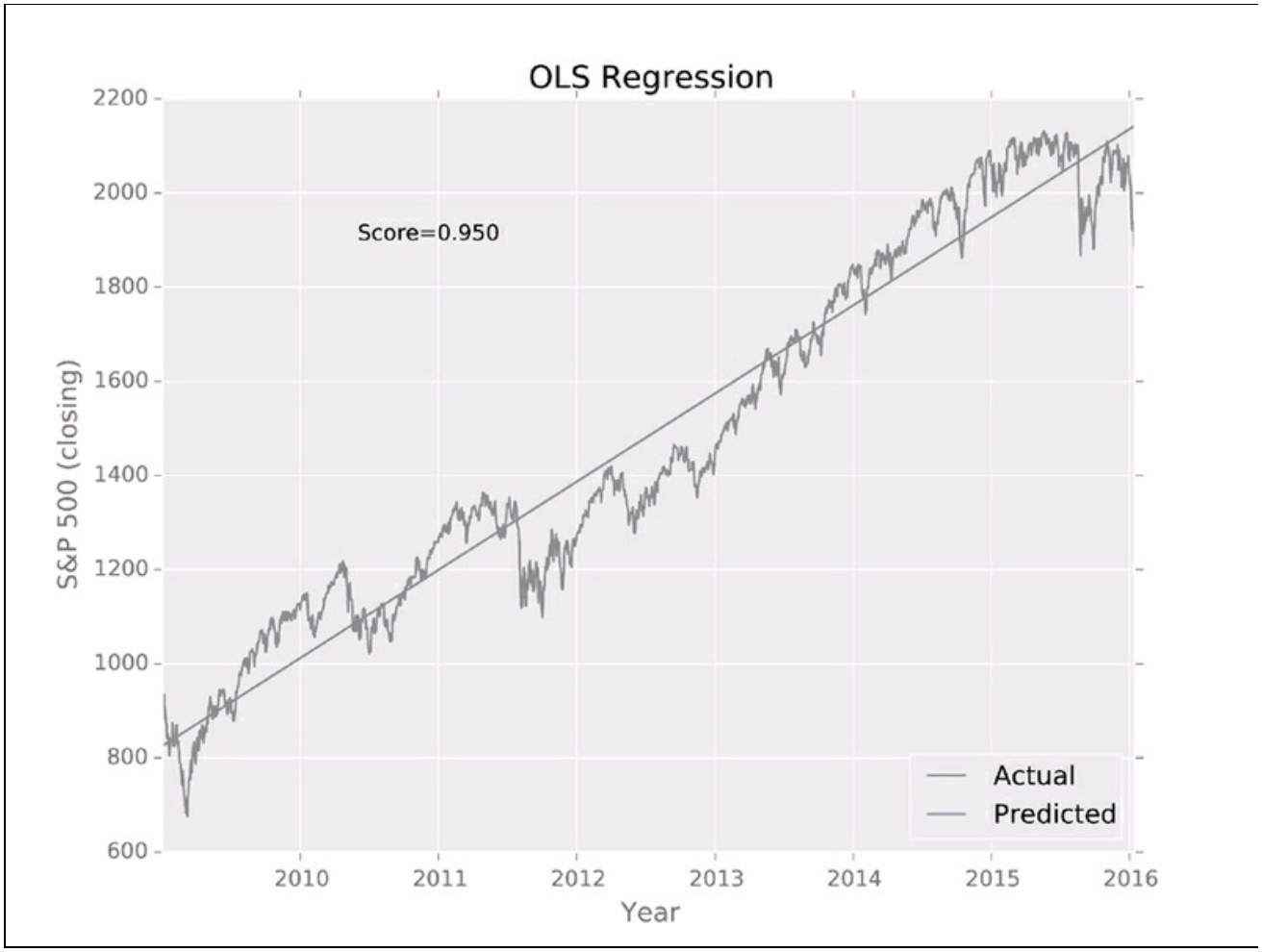


Figura 10.1

Sfortunatamente, SciKit-Learn non calcola il valore-p dell'adattamento. Quindi non è possibile sapere se l'adattamento è significativo o meno.

Se volete aggiungere al modello dei predittori non lineari (per esempio, quadrati, radici quadrate e logaritmi) o anche combinazioni dei predittori

originali, trattate semplicemente queste funzioni e combinazioni come nuove variabili indipendenti.

Regressione ridge (Regolarizzazione di Tichonov)

Se due o più predittori sono molto correlati (il caso della cosiddetta colinearità), l'adattamento della regressione OLS può produrre coefficienti molto grandi. Potete imporre una penalità per limitare la crescita incontrollabile dei coefficienti di regressione. La regressione ridge, un modello lineare generalizzato, impiega il parametro di complessità α per sopprimere i coefficienti. Questa procedura è chiamata regolarizzazione del modello:

$$SSR_{gen} = \sum_{i}^{N}(y_i - \mathrm{reg}(x_i))^2 w_i^2 + \alpha \sum_{i}^{N} \mathrm{coeff}_i^2 \rightarrow \min$$

Con α = 0, la regressione ridge diviene una regressione OLS. Quando α è elevato, la penalità è elevata; l'adattamento del modello fornisce come risultato dei coefficienti più bassi, ma il modello adattato può essere, potenzialmente, di qualità inferiore.

La funzione `Ridge()` crea un oggetto di regressione ridge e prende come parametro il valore α. Una volta creato l'oggetto, potete usarlo esattamente come si fa per la regressione OLS:

```
regr = lm.Ridge(alpha=.5)
regr.fit(X, y)
«...»
```

Regressione logistica (Logit)

Nonostante il nome, la regressione logistica, o *logit*, non è una regressione; è uno strumento di classificazione binaria. Impiega una funzione logistica generalizzata (l'estensione di una funzione logistica, detta anche curva-s o sigmoide, vedi Figura 10.2). La funzione logistica generalizzata è caratterizzata da asintoti superiore e inferiore, dal valore x del punto centrale della curva sigmoide e dalla ripidità della curva.

La funzione `LogisticRegression()` crea un'istanza di un oggetto di regressione logistica. Accetta vari parametri opzionali, dei quali il più importante è `C`.

Il parametro `C` è detto regolarizzazione inversa (l'inverso di α della regressione ridge). Perché i risultati della classificazione abbiano senso, spesso è meglio che sia pari ad almeno 20. Il valore predefinito, C = 1.0, per molte applicazioni pratiche si rivela inaccettabile.

La variabile dipendente `y` può essere un intero, un valore booleano o una stringa.

`sklearn.linear_model` implementa la classificazione attraverso la funzione `predict()`. A differenza dei modelli a regressione lineare (OLS e ridge), i risultati delle previsioni di un modello a regressione logit sono normalmente più utili rispetto ai valori dei coefficienti del modello `coef_` e del punto di intercettazione `intercept_`.

Nel prossimo esempio, useremo i risultati (resi anonimi) dei test svolti da una classe di Computer science composta da quarantatre studenti (disponibile nel file `grades.csv`) per illustrare il funzionamento della regressione logistica. Vediamo se i risultati dei primi due quiz (su un totale di dieci) sono in grado di predire il voto finale di uno studente o almeno di dire se il voto sarà sufficiente (almeno un "C") o insufficiente.

Listato 10.4 logit-example.py.

```
import pandas as pd
from sklearn.metrics import confusion_matrix
import sklearn.linear_model as lm

# Inizializza lo strumento di regressione
clf = lm.LogisticRegression(C=10.0)

# legge i dati e quantizza le lettere dei voti
grades = pd.read_table("grades.csv")
labels = ('F', 'D', 'C', 'B', 'A')
grades["Letter"] = pd.cut(grades["Final score"], [0, 60, 70, 80, 90, 100],
                          labels=labels)
X = grades[["Quiz 1", "Quiz 2"]]
# Adatta il modello, visualizza lo score e la matrice di confusione
clf.fit(X, grades["Letter"])
print("Score=%.3f" % clf.score(X, grades["Letter"]))
cm = confusion_matrix(clf.predict(X), grades["Letter"])
print(pd.DataFrame(cm, columns=labels, index=labels))

→ Score=0.535
→  F D C B A
→ F 0 0 0 0 0
→ D 2 16 6 4 1
→ C 0 1 6 2 2
→ B 0 0 0 1 2
→ A 0 0 0 0 0
```

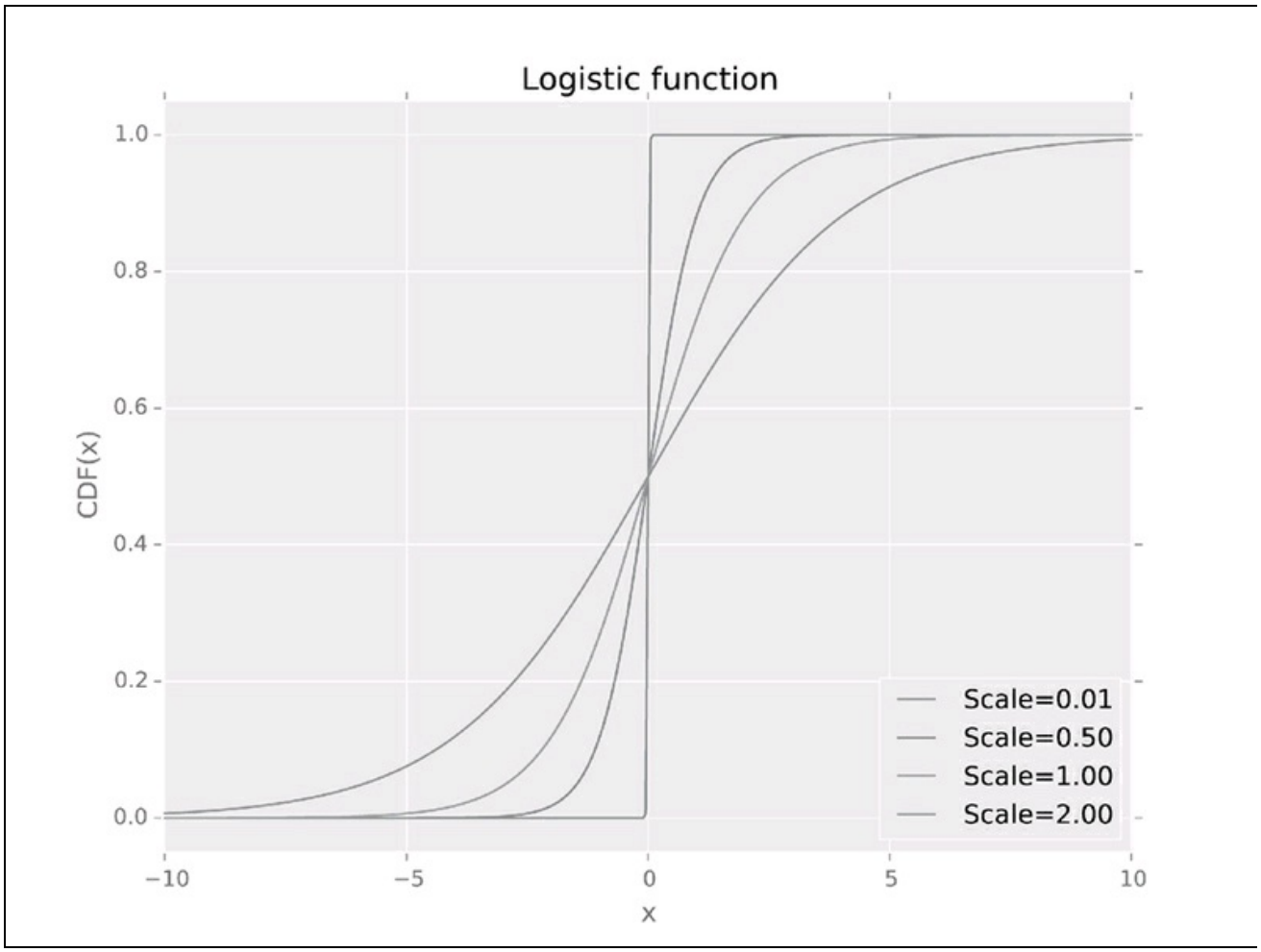


Figura 10.2

Abbiamo usato la funzione `confusion_matrix()` del modulo `sklearn.metrics` per calcolare la matrice di confusione (vedi la Tabella 10.1, *Matrice di confusione binaria*). Lo score del modello non sembra particolarmente accurato: il modello è stato in grado di prevedere correttamente solo circa il 54 percento di tutti i voti. Tuttavia, la matrice di confusione ha quasi tutte le voci diverse da zero o sulla diagonale principale (alla quale appartengono) o nelle vicinanze. Questo significa che il modello è accurato o commette un errore di circa ±1 un voto (una lettera). Per molte applicazioni pratiche, questa accuratezza (84 percento) può essere sufficiente.

Unità 50 - Raggruppamento dei dati con il clustering K-Means

Il clustering è una tecnica di machine learning senza supervisione. In altre parole, non abbiamo bisogno (o, meglio, non ci è possibile) addestrare il modello.

L'obiettivo del clustering è quello di raccogliere dei campioni (rappresentati come vettori n-dimensionali di numeri reali) in gruppi compatti disgiunti dotati di una buona prossimità interna. Perché il clustering funzioni, le dimensioni dei vettori devono avere estensioni ragionevolmente compatibili. Se l'intervallo di una dimensione è molto superiore o molto inferiore rispetto agli intervalli delle altre dimensioni, prima di eseguire il clustering è opportuno modificare la scala delle variabili troppo grandi o troppo piccole.

Il clustering K-means aggrega i campioni in k cluster (da cui il nome) in base al seguente algoritmo.

1. Scegliere casualmente k vettori come centroidi iniziali (i vettori non devono necessariamente essere campioni tratti dal dataset).
2. Assegnare ogni campione al centroide più vicino.
3. Ricalcolare la posizione dei centroidi.
4. Ripetere i passi 2 e 3 finché i centroidi non si spostano più.

Il modulo `sklearn.cluster` implementa l'algoritmo attraverso l'oggetto `KMeans`, che offre le funzioni `fit()` per il clustering vero e proprio, `predict()` per l'assegnamento dei nuovi campioni ai cluster pre-calcolati (etichettatura) e `fit_predict()` per eseguire in contemporanea le operazioni di clustering ed etichettatura.

Il modulo `sklearn.preprocessing` offre la funzione `scale()` per alterare la scala delle variabili. La funzione sottrae il valore minimo da ogni variabile dimensionale e divide la variabile per l'intervallo, mappandola così sul segmento [0...1].

Gli attributi cluster_centers_ e labels_ contengono i vettori che descrivono i centroidi finali e le etichette numeriche assegnate a ogni cluster di campioni. Le ultime etichette non riflettono lo scopo e la composizione dei cluster. Per assegnare ai cluster le etichette appropriate, avete le seguenti scelte:

- applicare l'intelligenza umana (osservare i campioni e trarne un'etichetta generalizzata);
- usare il crowdsourcing (per esempio, gli Human Intelligence Task di Amazon MTurk);
- generare le etichette dai dati (per esempio, designare come etichetta del cluster uno degli attributi del campione più promettente).

In *Calcolo delle misurazioni statistiche*, nell'Unità 47, abbiamo tentato di studiare il consumo di vino e di birra negli Stati Uniti nel 2009 e siamo giunti alla conclusione, controproducente, che i due fattori non avevano una correlazione lineare. Diamo a queste nobili bevande un'altra chance, attraverso clustering. L'operazione e i suoi risultati sono rappresentati nel seguente listato.

Listato 10.5 clusters.py.

```
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd
import sklearn.cluster, sklearn.preprocessing

# Pickling del frame
alco2009 = pickle.load(open("alco2QQ9.pickle", "rb"))

# States" abbreviations
states = pd.read_csv("states.csv",
                names=("State", "Standard", "Postal", "Capital"))
columns = ["Wine", "Beer"]

# Inizializza l'oggetto clustering, adatta il modello
kmeans = sklearn.cluster.KMeans(n_clusters=9)
kmeans.fit(alco2009[columns])
alco2009["Clusters"] = kmeans.labels_
centers = pd.DataFrame(kmeans.cluster_centers_, columns=colonne)

# Selezione dello stile
matplotlib.style.use("ggplot")

# Tracciamento degli Stati e dei centroidi dei cluster
ax = alco2009.plot.scatter(columns[0], columns[1], c="Clusters",
                    cmap=plt.cm.Accent, s=100)
centers.plot.scatter(columns[0], columns[1], color="red", marker="+",
                s=200, ax=ax)

# Aggiunge le abbreviazioni degli Stati quali annotazioni
def add_abbr(state):
   _ = ax.annotate(state["Postal"], state[columns], xytext=(1, 5),
```

```
            textcoords="offset points", size=8,
            color="darkslategrey")

alco2009withStates = pd.concat([alco2009, states.set_index("State")],
  axis=1)

alco2009withStates.apply(add_abbr, axis=1)

# Aggiunge il titolo, salva il grafico
plt.titleC"US States Clustered by Alcohol Consumption")
plt.savefig("../images/clusters.pdf")
```

Vale la pena di notare che la funzione `KMeans()` produce sempre otto cluster, a meno che le si passi il parametro `n_clusters`. È compito vostro e del vostro intuito scegliere il numero di cluster.

Tracciamo nello stesso grafico sia i dati originali (cerchi pieni, contrassegnati dalle abbreviazioni dei nomi di Stati) sia i centroidi dei cluster (croci), come illustrato di seguito.

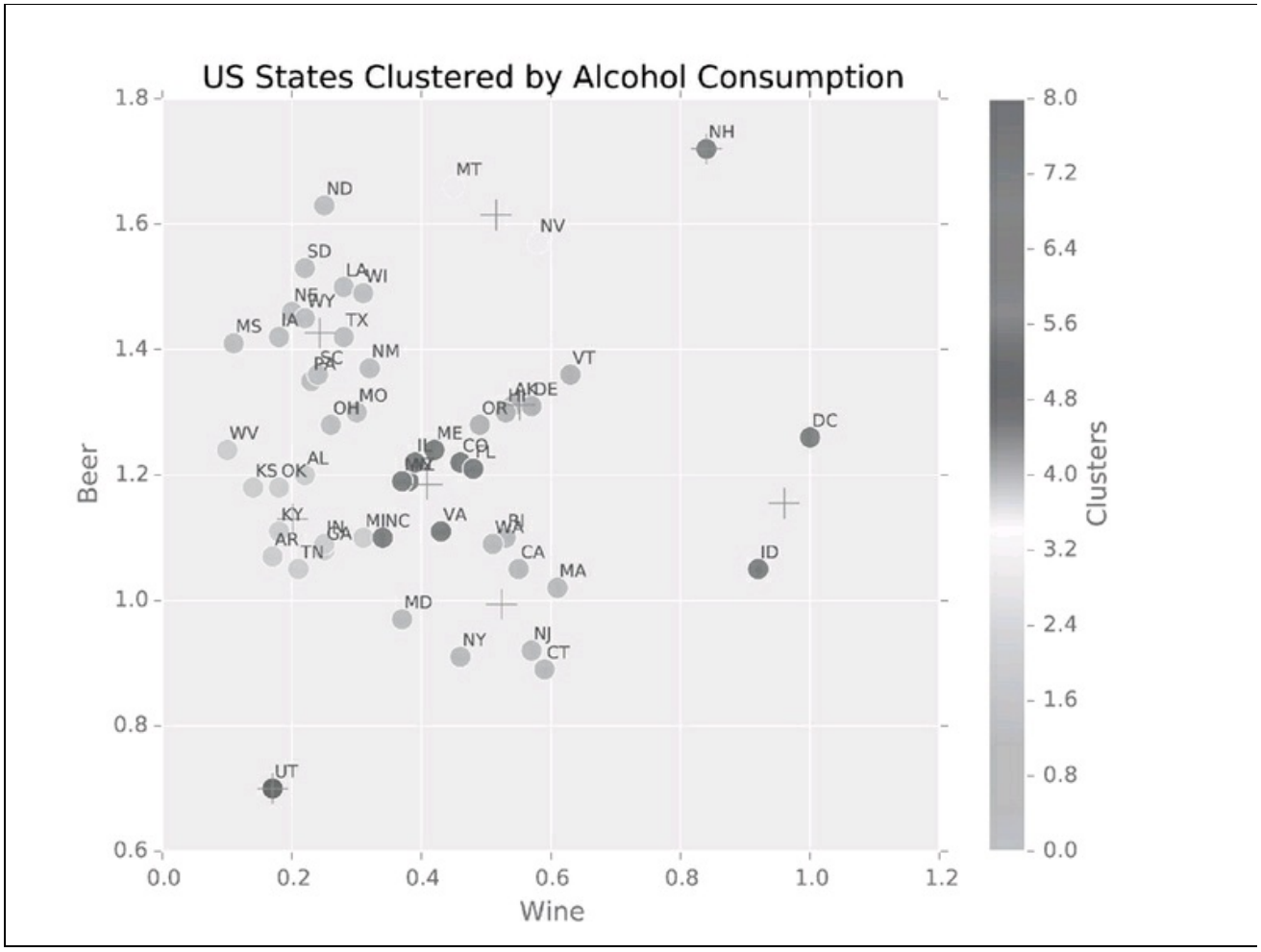


Figura 10.3

`KMeans()` ha fatto un buon lavoro nel riuscire a riconoscere i cluster (individuando anche i bevitori moderati degli stati del NordEst). La

collocazione delle etichette lascia un po' a desiderare, ma questo argomento non può rientrare negli scopi di questo libro.

Celle di Voronoi
L'algoritmo di clustering K-means suddivide lo spazio delle variabili indipendenti in celle di Voronoi, regioni di punti che sono più vicine a un determinato seme (punto dei dati) che a qualsiasi altro seme.

Unità 51 - Sopravvivere nelle foreste decisionali casuali

Gli alberi decisionali sono uno strumento di machine learning con supervisione. Si impiega un grafico ad albero in cui ogni nodo contiene un test su un determinato attributo del dataset e i rami in uscita dal nodo corrispondono ai risultati del test. Se decidete di usare gli alberi, avrete la necessità di addestrarli. L'addestramento consiste nel presentare all'albero vari predittori e le correspondenti etichette (caratteristiche), per consentirgli di regolare in modo appropriato le condizioni di test del nodo.

Un regressore a foresta decisionale casuale impiega un certo numero (ensemble) di alberi decisionali di classificazione su vari sotto-campioni del dataset e calcola la media delle previsioni per migliorarne l'accuratezza. Il modulo sklearn.ensemble offre il costruttore RandomForestRegressor(). L'oggetto regressore offre le funzioni fit(), predict() e così via, le cui sintassi e semantica sono coerenti con quelle degli altri regressori esaminati finora in questo capitolo.

Per mettere alla prova le foreste decisionali casuali useremo il dataset *Hedonic Prices of Census Tracts in the Boston Area*, (http://lib.stat.cmu.edu/datasets/boston) pubblicato per la prima volta da D. Harrison e D. Rubinfeld nel 1978. Il dataset offre 506 osservazioni dei valori mediani delle abitazioni residenziali (mv, l'etichetta dell'esperimento) e quattordici altre variabili (predittori).

Listato 10.6 rfr.py.

```
from sklearn.ensemble import RandomForestRegressor
import pandas as pd, numpy.random as rnd
import matplotlib, matplotlib.pyplot as plt

# Legge i dati, prepara due dataset complementari casuali
hed = pd.read_csv('Hedonic.csv')
selection = rnd.binomial(1, 0.7, size=len(hed)).astype(bool)
training = hed[selection]
testing = hed[-selection]

# Crea gli insiemi regressore e un predittore
rfr = RandomForestRegressor()
predictors_tra = training.ix[:, "crim" : "lstat"]
predictors_tst = testing.ix[:, "crim" : "lstat"]

# Adatta il modello
feature = "mv"
```

```
[1] rfr.fit(predictors_tra, training[feature])

  # Selezione dello stile
  matplotlib.style.use("ggplot")

  # Traccia i risultati della previsione
  plt.scatter(training[feature], rfr.predict(predictors_tra), c="green",
[2]           s=50)
[3] plt.scatter(testing[feature], rfr.predict(predictors_tst), c="red")
  plt.legend(["Training data", "Testing data"], loc="upper left")
  plt.plot(training[feature], training[feature], c="blue")
  plt.title("Hedonic Prices of Census Tracts in the Boston Area")
  plt.xlabel("Actual value")
  plt.ylabel("Predicted value")
  plt.savefig("../images/rfr.pdf")
```

Adatteremo il predittore (**[1]**) usando il set dei dati di addestramento (scelti in modo casuale dall'intero dataset) e poi ne eseguiremo il collaudo sia sul set di addestramento (**[2]**) sia sull'insieme di collaudo che non è stato usato per l'adattamento (**[3]**). Il valore previso mv non è discreto, il che ci impedisce di usare una matrice di confusione per stabilire la qualità del modello. Al contrario, ricorreremo a un'ispezione visuale, la quale suggerisce che la qualità della predizione su entrambi gli insiemi sia quanto meno corretta. Sembra che il modello sia ragionevolmente accurato e non sia soggetto a overfitting.

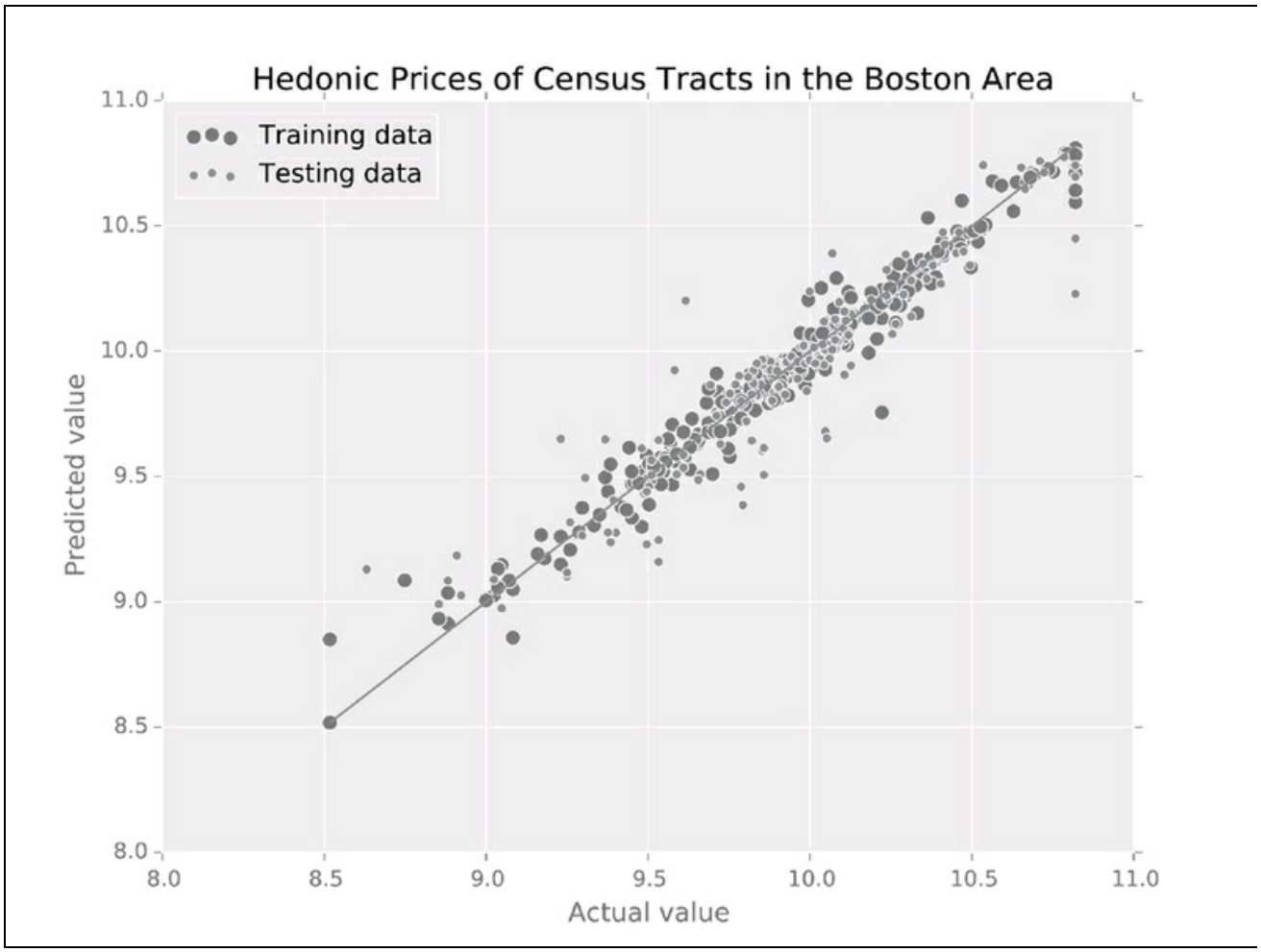

Hedonic Prices of Census Tracts in the Boston Area
Training data
Testing data
Predicted value
Actual value
11.0
10.5
10.0
9.5
9.0
8.5
8.0
8.0
8.5
9.0
9.5
10.0
10.5
11.0

Figura 10.4

Esercitazioni

In questo capitolo, abbiamo solo presentato quella che può essere considerata la punta dell'iceberg del machine learning (e, se la scienza non inganna, la gran parte dell'iceberg sta sott'acqua). Tuttavia, a questo punto avete a disposizione gli strumenti e le conoscenze necessari per elaborare i dati con e senza supervisione. Ora siete in grado di configurare esperimenti sui dati, di natura sia descrittiva (regressione e classificazione) sia predittiva, di verificarne la validità e di trarne conclusioni non banali.

Congratulazioni! A questo punto siete già quasi esperti di scienza dei dati (ma vediamo se siete in grado di completare anche gli ultimi progetti).

MOSN Clusters (*)

Scrivete un programma che suddivida in cluster i siti Massive Online Social Networking in base al numero di utenti registrati e di classificazione Alexa (https://en.wikipedia.org/wiki/List_of_social_networking_websites). Dato che la classificazione e le dimensioni dei siti variano molto, usate una scala logaritmica sia per il clustering sia per la presentazione.

S&P 500 (**)

Scrivete un programma che prenda da Yahoo! Finance (https://finance.yahoo.com/q/hp?s=^GSPC+Historical+Prices) le quotazioni di chiusura storiche del pacchetto S&P 500 nel corso del ventunesimo secolo e che rappresenti le osservazioni annue con un modello a regressione OLS. Se lo score di un modello non è adeguato (inferiore a 0.95), il programma deve suddividere l'intervallo temporale in due metà e procedere ricorsivamente finché lo score del modello OLS non supera la soglia 0.95 o l'intervallo non diventa più breve di una settimana. Il programma dovrà poi tracciare in un solo grafico i prezzi S&P 500 noti e i valori previsti dal modello OLS.

Predittore di metrò (***)

Sviluppate un programma che predica se una città ha o meno una metropolitana. Il programma deve considerare la popolazione, la densità di popolazione, il budget disponibile, le condizioni meteorologiche, i livelli di tassazione e altre variabili. Alcune di esse possono essere acquisite facilmente online, altre no. Usate la regressione logistica e la foresta decisionale casuale e scegliete il metodo più efficace.

Ulteriori letture

> Quando trovate una parte di un libro che davvero vi interessa, iniziate a leggerne le frasi a voce alta.
> *Grenville Kleiser*, autore nordamericano

Se siete arrivati fino a questa pagina, sapete già che questo libro presuppone che sappiate già, più o meno, ciò che volete fare, ma non sapete esattamente come. Vi suggerisco pertanto alcune letture eccellenti, talvolta con sovrapposizioni di argomenti, che entrano più in dettaglio su specifici argomenti.

Se non siete particolarmente esperti di scienza dei dati e conoscete già il linguaggio R o almeno non avete problemi a impararlo, cercate A*n Introduction to Statistical Learning con Applications in R* [JWHT13] e *Practical Data Science with R* [ZM14]. Il primo tratta argomenti di statistica, con elementi di programmazione pratica e il secondo, al contrario, è pratico e contiene alcuni elementi di statistica: insieme sono praticamente perfetti. Un altro libro, *The Elements of Data Analytic Style* [Lee15], si occupa dei vari tipi di modelli di dati, della stesura di report, della creazione di immagini di supporto e della scrittura di codice riproducibile.

Python for Data Analysis [McK12], il classico libro su pandas scritto da Wes McKinney, lo sviluppatore di pandas, contiene tutto il necessario per conoscere pandas e numpy, inclusa l'analisi delle serie temporali finanziarie. Il libro tratta in grande dettaglio molteplici casi di studio.

Natural Language Processing with Python [BKL09] è utile sia come guida al linguaggio Python, sia per l'elaborazione del linguaggio naturale. Va ben oltre la semplice normalizzazione dei testi e il conteggio delle parole e tratta la classificazione del testo, l'analisi della struttura delle frasi e l'analisi semantica; in più presuppone che non conosciate Python e che vogliate procurarvi una copia gratuita online (http://www.nltk.org/book/).

Il *Social Web* è sempre più un archivio, già enorme e in crescita esponenziale, di dati grezzi. *Mining the Social Web* [Rus11] analizza le API (Application Programming Interface) che consentono di usare le mailbox Unix, Twitter, LinkedIn, Google Buzz e Facebook. Offre un'interessante panoramica sui più recenti progressi nel campo dell'elaborazione del linguaggio naturale. Sfortunatamente, nonostante il fatto che sia stato pubblicato solo pochi anni fa, il libro è già abbastanza obsoleto: alcune API sono cambiate e alcuni progetti social (come Google Buzz) non esistono più.

MySQL Crash Course [For05] è un corso lampo sulla configurazione, manutenzione e utilizzo dei database relazionali. Il libro non tratta alcuna API, né per Python né per qualsiasi altro linguaggio di programmazione.

Nel momento in cui viene realizzato questo libro, non esiste alcun libro su Python e l'analisi dei dati di rete. *Network Analysis: Methodological Foundations* [BE05] non è rivolto ai programmatori e, per la verità, è forse un po' troppo teorico. *Social Network Analysis* [KY07] è molto più accessible a chi preferisce tenersi un po' più alla larga dai teoremi, dalle dimostrazioni e da pagine e pagine di formule. Nonostante il nome, il libro rappresenta una buona introduzione alle reti in generale, non solo alle reti dei social.

Infine, *Data Science from Scratch* [Gru15] è una sorta di versione espansa di questo libro. Tratta molto più approfonditamente argomenti come la statistica e il machine learning e sarebbe quindi la naturale prosecuzione di questo libro.

Trovate tutti i dettagli di questi testi nella bibliografia alla fine del libro.

Soluzioni dei progetti di tipo (*)

> In molti casi non dovremmo divagare rispetto alla vera soluzione di un problema; quanto meno per mancanza di dati.
> *Durant Drake*, ricercatore americano nel campo dell'etica

Questa appendice offre le soluzioni per i progetti contrassegnati con una sola stella (*). Le soluzioni suggerite sono implementate nel modo più "Pythonico". Se trovate soluzioni differenti, non preoccupatevi! Come esiste più di un modo per parlare dell'amore e della morte, così esiste più di un modo per risolvere un problema di programmazione.

Capitolo 1
Hello, World!

Scrivete un programma che produca in output le parole "Hello, World!" (senza gli apici) sulla riga di comando Python.

Listato B.1 solution-hello.py.

```
# Onoriamo la tradizione
print("Hello, World!")
```

Capitolo 2
Contatore della frequenza delle parole

Scrivete un programma che scarichi una pagina web richiesta dall'utente e indichi quali sono le dieci parole usate più frequentemente. Il programma dovrà trattare tutte le parole senza distinguere fra maiuscole e minuscole. Per lo scopo di questo esercizio, immaginate che una parola sia descritta dall'espressione regolare `r"\w+"`.

Listato B.2 solution-counter.py.

```
import urllib.request, re
from collections import Counter
```

```
# Comunica con l'utente e con Internet
url = input("Enter the URL: ")
try:
    page = urllib.request.urlopen(url)
except:
    print("Cannot open %s" % url)
    quit()

# Legge e normalizza parzialmente la pagina
doc = page.read().decode().lower()

# Suddivide il testo in parole
words = re.findall(r"\w+", doc)

# Crea un contatore e mostra la risposta
print(Counter(words).most_common(10))
```

Capitolo 3
Rilevatore di collegamenti ipertestuali non funzionanti

Scrivete un programma che, dato l'indirizzo URL di una pagina web, indichi i nomi e le destinazioni dei collegamenti ipertestuali non funzionanti presenti nella pagina. In questo esercizio, un collegamento ipertestuale non funziona se un tentativo di aprirlo con `urllib.request.urlopen()` fallisce.

Listato B.3 solution-broken_link.py.

```
import urllib.request, urllib.parse
import bs4 as BeautifulSoup

# Comunica con l'utente e Internet
base = input("Enter the URL: ")
try:
    page = urllib.request.urlopen(base)
except:
    print("Cannot open %s" % base)
    quit()

# Prepara soup
soup = BeautifulSoup.BeautifulSoup(page)

# Estrae i link come tuple (nome, url)
links = [(link.string, link[ "href" ])
        for link in soup.find_all("a")
        if link.has_attr("href")]

# tenta di aprire ogni link
broken = False
for name, url in links:
    # Combina la base e la destinazione del link
    dest = urllib.parse.urljoin(base, url)
    try:
        page = urllib.request.urlopen(dest)
        page.close()
    except:
        print("Link \"%s\" to \"%s\" is probably broken." % (name, dest))
        broken = True
```

```
# Ottimo!
if not broken:
   print("Page %s does not seem to have broken links." % base)
```

Capitolo 4
Indicizzatore di file MySQL

Scrivete un programma Python che, per ogni parola di un determinato file, registri in un database MySQL la parola stessa, il suo numero ordinale nel file (partendo da 1) e il suo contrassegno part-of-speech. Per riconoscere le parole, usate WordPunctTokenizer di NLTK (introdotto nell'Unità 16). Immaginate che le parole siano sufficientemente brevi da rientrare nel tipo di dati MySQL TINYTEXT. Progettate lo schema del database, create tutte le tabelle necessarie e provate a lavorarci dall'interfaccia prima di iniziare qualsiasi attività di programmazione Python.

La soluzione è formata da due file: uno script MySQL che predispone la tabella e un programma Python che svolge l'indicizzazione.

Listato B.4 solution-mysql_indexer.sql.

```
CREATE TABLE IF NOT EXISTS indexer(id INT PRIMARY KEY AUTO_INCREMENT,
                        ts TIMESTAMP,
                        word TINYTEXT,
                        position INT,
                        pos VARCHAR(8));
```

Listato B.5 solution-mysql_indexer.py.

```
import nltk, pymysql
infilename = input("Enter the name of the file to index: ")

# Modificare questa riga in base alle impostazioni del server MySQL
conn = pymysql.connect(user="dsuser", passwd="badpasswOrd", db="dsbd")
cur = conn.cursor()

QUERY = "INSERT INTO indexer (word,position,pos) VALUES "
wpt = nltk.WordPunctTokenizer()

offset = 1
with open(infilename) as infile:
   # Elabora il testo in modo incrementale, una riga alla volta
   # Una parola non può estendersi su più righe!
   for text in infile:
      # Tokenizza e aggiunge i tag POS
      pieces = enumerate(nltk.pos_tag(wpt.tokenize(text)))

      # Crea una query; non dimenticate l'escape per le parole!
      words = ["(\"%s\",%d,\"%s\")" % (conn.escape_string(w),
                              i + offset,
                              conn.escape_string(pos))
            for (i, (w, pos)) in pieces]

      # Esegue la query
      if words:
         cur.execute(QUERY + ',' .join(words))
```

```
        # fa avanzare il puntatore alla parola
        offset += len(words)

# Commit dei cambiamenti
conn.commit()
conn.close()
```

Capitolo 5
Sottrazioni fra array

Le somme parziali sono quasi l'equivalente di un integrale. In realtà, il calcolo definisce un integrale come una somma infinita di elementi infinitesimali. Le differenze parziali `arri+1-arri` sono quasi l'equivalente di una derivata. `numpy` non fornisce uno strumento per calcolare differenze parziali fra array. Scrivete un programma che, dato un array `arr`, calcoli la differenza parziale degli elementi dell'array. Immaginate che l'array sia numerico.

Listato B.6 solution-difference.py.

```
import numpy as np

# Alcuni dati sintetici di collaudo
array = np.random.binomial(5, 0.5, size=100)

# Differenze parziali: slicing e operazioni vettoriali!
diff = array[1:] - array[:-1]
```

Capitolo 6
Trappole per linci

Scrivete un programma che utilizzi i dati annui canadesi relativi alle trappole per linci (http://vincentarelbundock.github.io/Rdatasets/csv/datasets/lynx.csv) e produca il numero totale di trappole per linci per decennio (dieci anni), ordinati in senso inverso (prima il decennio più "produttivo"). Il programma deve scaricare i file di dati nella directory `cache`, ma solo se il file non esiste ancora in tale directory. Se la directory non esiste, occorre crearla. Il programma deve salvare i risultati come un file CSV nella directory `doc`. Se la directory non esiste, occorre crearla.

Listato B.7 solution-lynx.py.

```
import os, pandas as pd
import urllib.request

# Alcune "costanti"
SRC_HOST = "https://vincentarelbundock.github.io"
FILE = "/lynx.csv"
```

```
SRC_NAME = SRC_HOST + "/Rdatasets/csv/datasets" + FILE CACHE = "cache"
DOC = "doc"

# Prepara le directory, se necessario
if not os.path.isdir(CACHE):
   os.mkdir(CACHE)
if not os.path.isdir(DOC):
   os.mkdir(DOC)

# Controlla se il file è in cache; altrimenti lo salva in cache
if not os.path.isfile(CACHE + FILE):
   try:
      src = urllib.request.urlopen(SRC_NAME)
      lynx = pd.read_csv(src)
   except:
      print("Cannot access %f." % SRC_NAME)
      quit()

   # Crea un frame di dati
   lynx.to_csv(CACHE + FILE)
else:
   lynx = pd.read_csv(CACHE + FILE)

# Aggiunge la colonna "decade"
lynx["decade"] = (lynx['time'] / 10).round() * 10

# Aggrega e ordina
by_decade = lynx.groupby("decade").sum()
by_decade = by_decade.sort_values(by="lynx", ascending=False)

# Salva i risultati
by_decade["lynx"].to_csv(DOC + FILE)
```

Capitolo 7
Centralità e correlazioni

Scaricate il grafo di una rete sociale di utenti Epinions.com dalla Stanford Large Network Dataset Collection (a cura di J. Leskovec e A. Krevl, http://snap.stanford.edu/data/soc-Epinions1.html), ed estraete la decima comunità più grande. Scrivete un programma che calcoli e visualizzi le correlazioni fra tutte le misurazioni di centralità di rete menzionate in questo capitolo (per divertirvi potete anche aggiungere un coefficiente di clustering). Vi suggerisco di memorizzare tutte le centralità in un frame di dati pandas. Se necessario, consultate il calcolo delle correlazioni in pandas in *Calcolo delle misurazioni statistiche* nell'Unità 47.

Esistono coppie di centralità fortemente correlate?

Listato B.8 solution-centrality.py.

```
import networkx as nx, community
import pandas as pd

# Importa la rete
G = nx.read_adjlist(open("soc-Epinions1.txt", "rb"))

# Estrae la struttura comunitaria e la salva come una serie di dati
series partition = pd.Series(community.best_partition(G))
```

```
# Trova l'indice della 10a più grande community
top10 = partition.value_counts().index[9]

# Estrae la decima più grande community
# Ricordate che le etichette dei nodi sono stringhe!
subgraph = partition[partition == top10].index.values.astype('str')
F = G.subgraph(subgraph)

# Calcola le misure della rete
df = pd.DataFrame()
df["degree"] = pd.Series(nx.degree_centrality(F))
df["closeness"] = pd.Series(nx.closeness_centrality(F))
df["betweenness"] = pd.Series(nx.betweenness_centrality(F)
df["eigenvector"] = pd.Series(nx.eigenvector_centrality(F)
df["clustering"] = pd.Series(nx.clustering(F))

# Calcola le correlazioni
print(df.corr())
```

```
→  degree closeness betweenness eigenvector clustering
→ degree 1.000000 0.247377 0.871812 0.738836 0.100259
→ closeness 0.247377 1.000000 0.169449 0.547228 0.024052
→ betweenness 0.871812 0.169449 1.000000 0.527290 -0.015759
→ eigenvector 0.738836 0.547228 0.527290 1.000000 0.143070
→ clustering 0.100259 0.024052 -0.015759 0.143070 1.000000
```

La centralità *degree* ha una forte correlazione lineare con le centralità *betweenness* ed *eigenvector*.

Capitolo 8
American Pie

Scrivete un programma che visualizzi o salvi su un file PDF un grafico a torta degli Stati USA raggruppati per lettera iniziale. Per risolvere questo problema dovete partire da una lista di nomi o abbreviazioni di nomi degli Stati, che potete trarre da un sito web (http://www.stateabbreviations.us).

Listato B.9 solution-states_pie.py.

```
import pandas as pd
import matplotlib, matplotlib.pyplot as plt

def initial(word):
   return word[0]

# Legge il nome degli Stati (usate la fonte che volete)
states = pd.read_csv("states.csv",
                     names=("State", "Standard", "Postal", "Capital"))

# Selezione dello stile
matplotlib.style.use("ggplot")

# Tracciamento del grafico
plt.axes(aspect=1)
states.set_index('Postal').groupby(initial).count()['Standard'].plot.pie()
plt.title( "States by the First Initial")
plt.ylabel("")

plt.savefig("../images/states-pie.pdf")
```

La Figura B.1 mostra il grafico risultante.

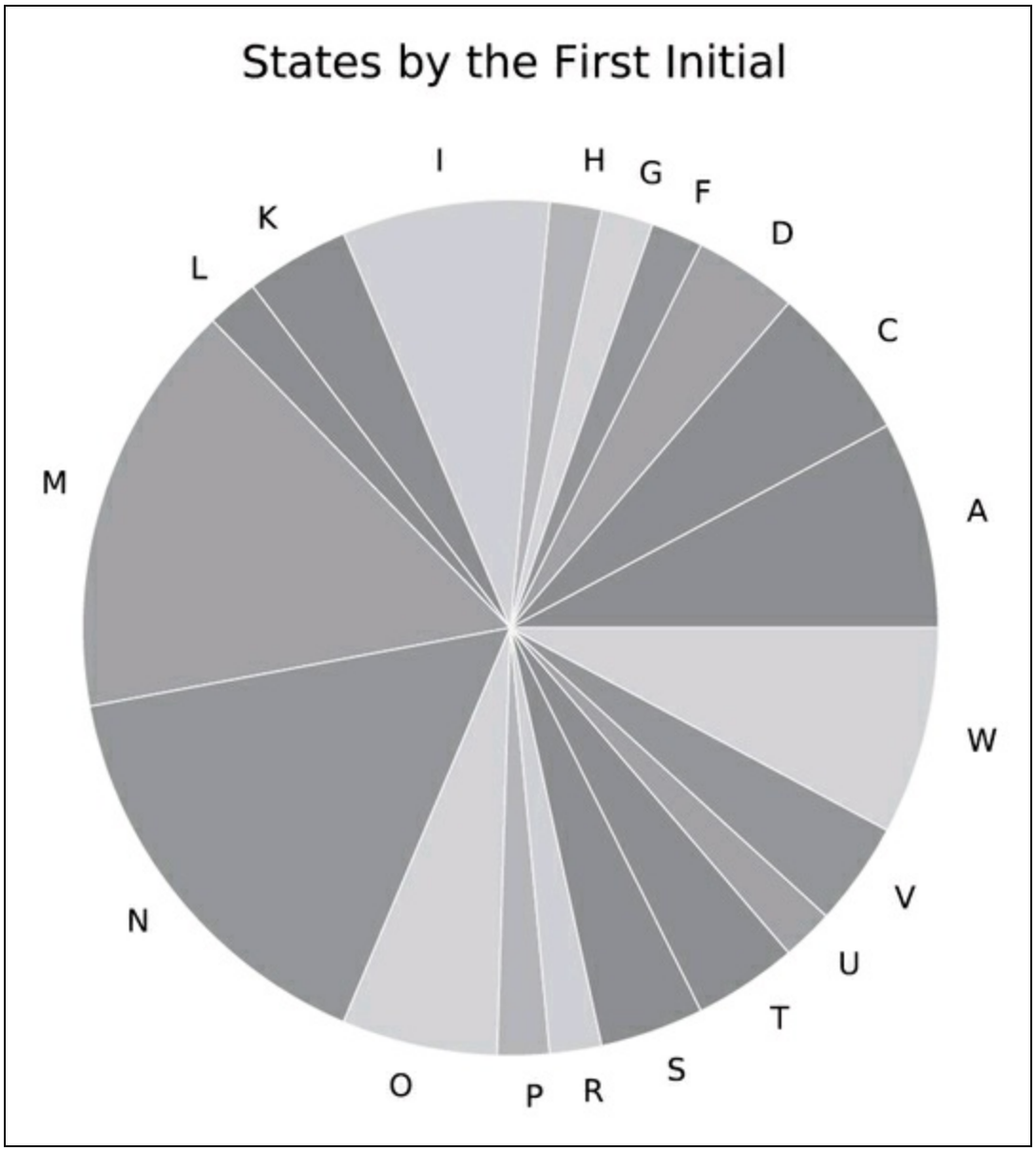


Figura B.1

Capitolo 9
S&P 500 nel Ventunesimo secolo

Scrivete un programma che esegua alcune semplici misurazioni statistiche dei valori di chiusura del pacchetto di azioni S&P 500: media, deviazione standard, skewness e correlazione fra i valori di chiusura e il volume degli scambi nel ventunesimo secolo. La correlazione è affidabile? Potete scaricare i dati storici da Yahoo! Finance (https://finance.yahoo.com/q/hp?s=^GSPC+Historical+Prices). Ricordate che il Ventunesimo secolo è iniziato il 1 gennaio 2001. I dati che scaricherete saranno necessariamente differenti da

quelli usati in questo esempio e pertanto anche le risposte saranno differenti.

Listato B.10 solution-sap.py.

```
import pandas as pd
from scipy.stats import pearsonr

# Legge i dati
sap = pd.read_csv("sapXXI.csv").set_index("Date")

# Calcola e presenta le statistiche
print("Mean:", sap["Close"].mean())
print("Standard deviation:", sap["Close"].std())
print("Skewness:", sap["Close"].skew())
print("Correlation:\n", sap[["Close", "Volume"]].corr()),
    p = pearsonr(sap["Close"], sap["Volume"])
print("p-value:", p)
```

```
→ Mean: 1326.35890044
→ Standard deviation: 332.784759688
→ Skewness: 0.858098114571
→ Correlation:
→  Close Volume
→ Close 1.000000 0.103846
→ Volume 0.103846 1.000000
→ p-value: 1.5301705092e-10
```

La correlazione è molto affidabile, ma anche molto insignificante.

Capitolo 10
Cluster MOSN

Scrivete un programma che suddivida in cluster i siti Massive Online Social Networking in base al numero di utenti registrati e di classificazione Alexa (https://en.wikipedia.org/wiki/List_of_social_networking_websites). Dato che la classificazione e le dimensioni dei siti variano molto, usate una scala logaritmica sia per il clustering sia per la presentazione.

Listato B.11 solution-mosn.py.

```
import pandas as pd, numpy as np
import sklearn.cluster, sklearn.preprocessing
import matplotlib, matplotlib.pyplot as plt

# Legge i dati
mosn = pd.read_csv('mosn.csv', thousands=',',
            names=('Name', 'Description', 'Date', 'Registered Users',
                'Registration', 'Alexa Rank'))
columns = ['Registered Users', 'Alexa Rank']

# Elimina le righe con dati mancanti e zero
good = mosn[np.log(mosn[columns]).notnull().all(axis=1)].copy()

# Esegue il clustering
kmeans = sklearn.cluster.KMeans()
kmeans.fit(np.log(good[columns]))
good["Clusters"] = kmeans.labels_
```

```
# Estrae Facebook
fb = good.set_index('Name').ix['Facebook']['Clusters']

# Selezione dello stile
matplotlib.style.use("ggplot")
# Visualizza i risultati
ax = good.plot.scatter(columns[0], columns[1], c="Clusters",
                       cmap=plt.cm.Accent, s=100)
plt.title("Massive online social networking sites")
plt.xscale("log")
plt.yscale("log")

# Annota i siti principali
def add_abbr(site):
    if site['Clusters'] == fb:
        _ = ax.annotate(site["Name"], site[columns], xytext=(1, 5)
                        textcoords="offset points", size=8,
                        color="darkslategrey")
good.apply(add_abbr, axis=1)

plt.savefig("../images/mosn.png")
```

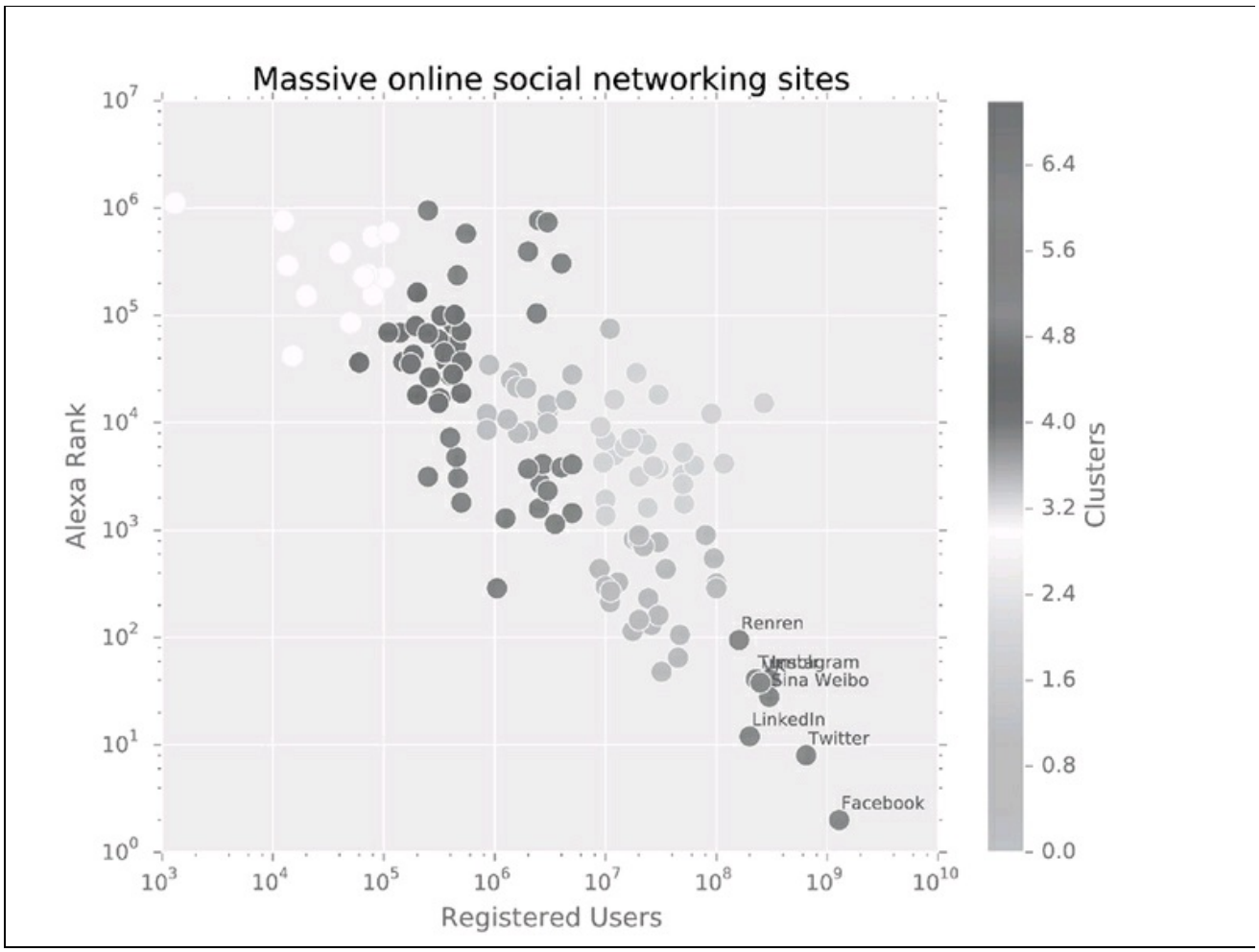


Figura B.2

[BE05] Ulrik Brandes e Thomas Erleback, *Network Analysis: Methodological Foundations*, Springer, New York (NY) 2005.

[BKL09] Steven Bird, Ewan Klein e Edward Loper, *Natural Language Processing with Python*, O'Reilly & Associates, Inc., Sebastopol (CA) 2009.

[For05] Ben Forta, *MySQL Crash Course*, Sams Publishing, Indianapolis (IN) 2005.

[Gru15] Joel Grus, *Data Science from Scratch: First Principles with Python*, O'Reilly & Associates, Inc., Sebastopol (CA) 2015.

[JWHT13] Gareth James, Daniela Witten, Trevor Hastie e Robert Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer, New York (NY) 2013.

[KY07] David Knoke e Song Yang, *Social Network Analysis*, SAGE Publications, Thousand Oaks (CA) 2nd, 2007.

[Lee15] Jeff Leek, *The Elements of Data Analytic Style*, Leanpub, Victoria (BC, Canada) 2015.

[McK12] Wes McKinney, *Python for Data Analysis*, O'Reilly & Associates, Inc., Sebastopol (CA) 2012.

[Rus11] Matthew A. Russell, *Mining the Social Web*, O'Reilly & Associates, Inc., Sebastopol (CA) 2011.

[ZM14] Nina Zumel e John Mount, *Practical Data Science with R*, Manning Publications Co., Greenwich (CT) 2014.