

Python w analizie danych

PRZETWARZANIE DANYCH ZA POMOCĄ PAKIETÓW
PANDAS I NUMPY ORAZ ŚRODOWISKA IPYTHON



Tytuł oryginału: Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, 2nd Edition

Tłumaczenie: Konrad Matuk

ISBN: 978-83-283-4081-7

© 2018 Helion S.A.

Authorized Polish translation of the English edition of Python for Data Analysis, 2nd Edition ISBN 9781491957660 © 2018 William McKinney

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same."

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/pytand.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/pytand>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Przedmowa	11
1. Wstęp	15
1.1. O czym jest ta książka?	15
Jakie rodzaje danych?	15
1.2. Dlaczego warto korzystać z Pythona w celu przeprowadzenia analizy danych?	16
Python jako spoiwo	16
Rozwiązywanie problemu „dwujęzyczności”	17
Dlaczego nie Python?	17
1.3. Podstawowe biblioteki Pythona	17
NumPy	18
pandas	18
Matplotlib	19
IPython i Jupyter	19
SciPy	20
Scikit-learn	21
statsmodels	21
1.4. Instalacja i konfiguracja	22
Windows	22
Apple (OS X, macOS)	23
GNU, Linux	23
Instalowanie i aktualizowanie pakietów Pythona	24
Python 2 i Python 3	24
Zintegrowane środowiska programistyczne i edytory tekstowe	25
1.5. Społeczność i konferencje	25
1.6. Nawigacja po książce	26
Przykłady kodu	27
Przykładowe dane	27
Konwencje importowania	27
Żargon	27

2. Podstawy Pythona oraz obsługi narzędzi IPython i Jupyter	29
2.1. Interpreter Pythona	30
2.2. Podstawy interpretera IPython	31
Uruchamianie powłoki IPython	31
Uruchamianie notatnika Jupyter Notebook	32
Uzupełnianie poleceń	35
Introspekcja	36
Polecenie %run	37
Wykonywanie kodu ze schowka	39
Skróty klawiaturowe działające w terminalu	39
Polecenia magiczne	40
Integracja pakietu matplotlib	42
2.3. Podstawy Pythona	42
Semantyka języka Python	43
Skalarne typy danych	50
Przepływ sterowania	57
3. Wbudowane struktury danych, funkcje i pliki	61
3.1. Struktury danych i sekwencje	61
Krotka	61
Lista	64
Wbudowane funkcje obsługujące sekwencje	68
Słownik	70
Zbiór	73
Lista, słownik i zbiór — składanie	75
3.2. Funkcje	77
Przestrzenie nazw, zakres i funkcje lokalne	78
Zwracanie wielu wartości	79
Funkcje są obiektami	79
Funkcje anonimowe (lambda)	81
Currying — częściowa aplikacja argumentów	82
Generatory	82
Błędy i obsługa wyjątków	84
3.3. Pliki i system operacyjny	86
Bajty i kodowanie Unicode w plikach	89
3.4. Podsumowanie	91
4. Podstawy biblioteki NumPy: obsługa tablic i wektorów	93
4.1. NumPy ndarray — wielowymiarowy obiekt tablicowy	95
Tworzenie tablic ndarray	96
Typ danych tablic ndarray	98
Działania matematyczne z tablicami NumPy	100

Podstawy indeksowania i przechwytywania części	101
Indeksowanie i wartości logiczne	105
Indeksowanie specjalne	108
Transponowanie tablic i zamiana osi	109
4.2. Funkcje uniwersalne	
— szybkie funkcje wykonywane na poszczególnych elementach tablicy	110
4.3. Programowanie z użyciem tablic	113
Logiczne operacje warunkowe jako operacje tablicowe	115
Metody matematyczne i statystyczne	116
Metody tablic logicznych	117
Sortowanie	118
Wartości unikalne i operacje logiczne	119
4.4. Tablice i operacje na plikach	120
4.5. Algebra liniowa	120
4.6. Generowanie liczb pseudolosowych	122
4.7. Przykład: błędzenie losowe	124
Jednoczesne symulowanie wielu błędzeń losowych	125
4.8. Podsumowanie	126
5. Rozpoczynamy pracę z biblioteką pandas	127
5.1. Wprowadzenie do struktur danych biblioteki pandas	127
Obiekt Series	128
Obiekt DataFrame	131
Obiekty index	137
5.2. Podstawowe funkcjonalności	139
Uaktualnianie indeksu	139
Odrzucanie elementów osi	141
Indeksowanie, wybieranie i filtrowanie	143
Indeksy w postaci liczb całkowitych	147
Działania arytmetyczne i wyrównywanie danych	148
Funkcje apply i map	153
Sortowanie i tworzenie rankingów	154
Indeksy osi ze zduplikowanymi etykietami	157
5.3. Podsumowywanie i generowanie statystyk opisowych	158
Współczynnik korelacji i kowariancja	161
Unikalne wartości, ich liczba i przynależność	163
5.4. Podsumowanie	165
6. Odczyt i zapis danych, formaty plików	167
6.1. Odczyt i zapis danych w formacie tekstowym	167
Wczytywanie części pliku tekstowego	173
Zapis danych w formacie tekstowym	174

Praca z plikami danych rozgraniczonych	176
Dane w formacie JSON	178
XML i HTML — web scraping	179
6.2. Formaty danych binarnych	182
Obsługa formatu HDF5	183
Wczytywanie plików programu Microsoft Excel	185
6.3. Obsługa interfejsów sieciowych	186
6.4. Obsługa baz danych	187
6.5. Podsumowanie	188
7. Czyszczenie i przygotowywanie danych	189
7.1. Obsługa brakujących danych	189
Filtrowanie brakujących danych	191
Wypełnianie brakujących danych	193
7.2. Przekształcanie danych	195
Usuwanie duplikatów	195
Przekształcanie danych przy użyciu funkcji lub mapowania	196
Zastępowanie wartości	197
Zmiana nazw indeksów osi	199
Dyskretyzacja i podział na koszyki	200
Wykrywanie i filtrowanie elementów odstających	202
Permutacje i próbkowanie losowe	203
Przetwarzanie wskaźników i zmiennych zastępczych	204
7.3. Operacje przeprowadzane na łańcuchach	207
Metody obiektu typu string	207
Wyrażenia regularne	209
Wektoryzacja funkcji łańcuchów w pakiecie pandas	212
7.4. Podsumowanie	215
8. Przetwarzanie danych — operacje łączenia, wiązania i przekształcania	217
8.1. Indeksowanie hierarchiczne	217
Zmiana kolejności i sortowanie poziomów	220
Parametry statystyki opisowej z uwzględnieniem poziomu	220
Indeksowanie z kolumnami ramki danych	221
8.2. Łączenie zbiorów danych	222
Łączenie ramek danych w stylu łączenia elementów baz danych	222
Łączenie przy użyciu indeksu	227
Konkatenacja wzdłuż osi	230
Łączenie częściowo nakładających się danych	234

8.3. Zmiana kształtu i operacje osiowe	235
Przekształcenia z indeksowaniem hierarchicznym	236
Przekształcanie z formatu „długiego” na „szeroki”	238
Przekształcanie z formatu „szerokiego” na „długi”	241
8.4. Podsumowanie	243
9. Wykresy i wizualizacja danych	245
9.1. Podstawy obsługi interfejsu pakietu matplotlib	245
Obiekty figure i wykresy składowe	246
Kolory, oznaczenia i style linii	250
Punkty, etykiety i legendy	252
Adnotacje i rysunki	255
Zapisywanie wykresów w postaci plików	257
Konfiguracja pakietu matplotlib	258
9.2. Generowanie wykresów za pomocą pakietów pandas i seaborn	259
Wykresy liniowe	259
Wykresy słupkowe	262
Histogramy i wykresy gęstości	266
Wykresy punktowe	268
Wykresy panelowe i dane katégoryczne	269
9.3. Inne narzędzia przeznaczone do wizualizacji danych w Pythonie	272
9.4. Podsumowanie	272
10. Agregacja danych i operacje wykonywane na grupach	273
10.1. Mechanika interfejsu groupby	274
Iteracja po grupach	277
Wybieranie kolumny lub podzbioru kolumn	278
Grupowanie przy użyciu słowników i serii	279
Grupowanie przy użyciu funkcji	280
Grupowanie przy użyciu poziomów indeksu	280
10.2. Agregacja danych	281
Przetwarzanie kolumna po kolumnie i stosowanie wielu funkcji	282
Zwracanie zagregowanych danych bez indeksów wierszy	285
10.3. Metoda apply — ogólne zastosowanie techniki dziel-zastosuj-połącz	286
Usuwanie kluczy grup	288
Kwantyle i analiza koszykowa	288
Przykład: wypełnianie brakujących wartości przy użyciu wartości charakterystycznych dla grupy	290
Przykład: losowe generowanie próbek i permutacja	292
Przykład: średnie ważone grup i współczynnik korelacji	293
Przykład: regresja liniowa grup	295

10.4. Tabele przestawne i krzyżowe	295
Tabele krzyżowe	298
10.5. Podsumowanie	299
11. Szeregi czasowe	301
11.1. Typy danych i narzędzia przeznaczone do obsługi daty i czasu	302
Konwersja pomiędzy obiektami string i datetime	303
11.2. Podstawy szeregów czasowych	305
Indeksowanie i wybieranie	306
Szeregi czasowe z duplikatami indeksów	309
11.3. Zakresy dat, częstotliwości i przesunięcia	310
Generowanie zakresów dat	310
Częstotliwości i przesunięcia daty	313
Przesuwanie daty	314
11.4. Obsługa strefy czasowej	317
Lokalizacja i konwersja stref czasowych	317
Operacje z udziałem obiektów Timestamp o wyznaczonej strefie czasowej	319
Operacje pomiędzy różnymi strefami czasowymi	320
11.5. Okresy i przeprowadzanie na nich operacji matematycznych	321
Konwersja częstotliwości łańcuchów	322
Kwartalne częstotliwości okresów	323
Konwersja znaczników czasu na okresy (i z powrotem)	325
Tworzenie obiektów PeriodIndex na podstawie tablic	326
11.6. Zmiana rozdzielczości i konwersja częstotliwości	328
Zmniejszanie częstotliwości	329
Zwiększanie rozdzielczości i interpolacja	332
Zmiana rozdzielczości z okresami	333
11.7. Funkcje ruchomego okna	334
Funkcje ważone wykładniczo	337
Binarne funkcje ruchomego okna	338
Funkcje ruchomego okna definiowane przez użytkownika	340
11.8. Podsumowanie	340
12. Zaawansowane funkcje biblioteki pandas	341
12.1. Dane katagoryczne	341
Kontekst i motywacja	341
Typ Categorical w bibliotece pandas	343
Obliczenia na obiektach typu Categorical	345
Metody obiektu katagorycznego	347
12.2. Zaawansowane operacje grupowania	349
Transformacje grup i „nieobudowane” operacje grupowania	349
Zmiana rozdzielczości czasu przeprowadzana przy użyciu grup	353

12.3. Techniki łączenia metod w łańcuch	354
Metoda pipe	355
12.4. Podsumowanie	356
13. Wprowadzenie do bibliotek modelujących	357
13.1. Łączenie pandas z kodem modelu	357
13.2. Tworzenie opisów modeli przy użyciu biblioteki Patsy	360
Przekształcenia danych za pomocą formuł Patsy	362
Patsy i dane kategoryczne	363
13.3. Wprowadzenie do biblioteki statsmodels	366
Szacowanie modeli liniowych	366
Szacowanie procesów szeregów czasowych	369
13.4. Wprowadzenie do pakietu scikit-learn	369
13.5. Dalszy rozwój	373
14. Przykłady analizy danych.....	375
14.1. Dane USA.gov serwisu Bitly	375
Liczenie stref czasowych w czystym Pythonie	376
Liczenie stref czasowych przy użyciu pakietu pandas	378
14.2. Zbiór danych MovieLens 1M	384
Wyznaczenie rozbieżności ocen	388
14.3. Imiona nadawane dzieciom w USA w latach 1880 – 2010	389
Analiza trendów imion	394
14.4. Baza danych USDA Food	402
14.5. Baza danych 2012 Federal Election Commission	406
Statystyki datków z podziałem na wykonywany zawód i pracodawcę	409
Podział kwot datków na koszyki	411
Statystyki datków z podziałem na poszczególne stany	413
14.6. Podsumowanie	414
A. Zaawansowane zagadnienia związane z biblioteką NumPy	415
A.1. Szczegóły budowy obiektu ndarray	415
Hierarchia typów danych NumPy	416
A.2. Zaawansowane operacje tablicowe	417
Zmiana wymiarów tablic	417
Kolejności charakterystyczne dla języków C i Fortran	419
Łączenie i dzielenie tablic	420
Powtarzanie elementów — funkcje tile i repeat	422
Alternatywy indeksowania specjalnego — metody take i put	423
A.3. Rozgłaszanie	424
Rozgłaszanie wzdłuż innych osi	426
Przypisywanie wartości elementom tablicy poprzez rozgłaszanie	428

A.4. Zaawansowane zastosowania funkcji uniwersalnych	429
Metody instancji funkcji uniwersalnych	429
Pisanie nowych funkcji uniwersalnych w Pythonie	431
A.5. Tablice o złożonej strukturze	432
Zagnieżdżone typy danych i pola wielowymiarowe	433
Do czego przydają się tablice o złożonej strukturze?	434
A.6. Jeszcze coś o sortowaniu	434
Sortowanie pośrednie — metody argsort i lexsort	435
Alternatywne algorytmy sortowania	436
Częściowe sortowanie tablic	437
Wyszukiwanie elementów w posortowanej tablicy za pomocą metody <code>numpy.searchsorted</code>	438
A.7. Pisanie szybkich funkcji NumPy za pomocą pakietu Numba	439
Tworzenie obiektów <code>numpy.ufunc</code> za pomocą pakietu Numba	440
A.8. Zaawansowane tablicowe operacje wejścia i wyjścia	441
Pliki mapowane w pamięci	441
HDF5 i inne możliwości zapisu tablic	442
A.9. Jak zachować wysoką wydajność?	442
Dlaczego warto korzystać z sąsiadujących ze sobą obszarów pamięci?	443
B. Dodatkowe informacje dotyczące systemu IPython	445
B.1. Korzystanie z historii poleceń	445
Przeszukiwanie i korzystanie z historii poleceń	445
Zmienne wejściowe i wyjściowe	446
B.2. Interakcja z systemem operacyjnym	447
Polecenia powłoki systemowej i aliasy	447
System tworzenia skrótów do katalogów	448
B.3. Narzędzia programistyczne	449
Interaktywny debugger	449
Pomiar czasu — funkcje <code>%time</code> i <code>%timeit</code>	453
Podstawowe profilowanie — funkcje <code>%prun</code> i <code>%run-p</code>	455
Profilowanie funkcji linia po linii	457
B.4. Wskazówki dotyczące produktywnego tworzenia kodu w środowisku IPython	458
Przeładowywanie modułów	459
Wskazówki dotyczące projektowania kodu	460
B.5. Zaawansowane funkcje środowiska IPython	461
Co zrobić, aby własne klasy były przyjazne dla systemu IPython?	461
Profile i konfiguracja	462
Skorowidz	465

Rozpoczynamy pracę z biblioteką pandas

Większość dalszej części tej książki będzie dotyczyła biblioteki pandas. W jej skład wchodzi struktury danych i narzędzia przeznaczone do przetwarzania danych, które ułatwiają i przyspieszają oczyszczanie danych i analizę w Pythonie. Biblioteka pandas jest często używana w połączeniu z innymi narzędziami przeznaczonymi do przetwarzania danych numerycznych, takimi jak NumPy i SciPy, bibliotekami analitycznymi, takimi jak statsmodels i scikit-learn, a także bibliotekami przeznaczonymi do wizualizacji danych, takimi jak matplotlib. Pakiet pandas przypomina pakiet NumPy — jest nastawiony na przetwarzanie tablic, oferuje wiele funkcji operujących na tablicach i umożliwia przetwarzanie danych bez pętli for.

W bibliotece pandas zastosowano wiele rozwiązań zaczerpniętych z NumPy, ale największą różnicą pomiędzy tymi bibliotekami jest to, że pandas została zaprojektowana z myślą o pracy z danymi w formie tabel lub danymi o charakterze heterogenicznym, a biblioteka NumPy jest zoptymalizowana pod kątem pracy z homogenicznymi tablicami danych liczbowych.

Praca nad otwartym projektem pandas rozpoczęła się w 2010 roku. Od tego czasu biblioteka ta rozwinęła się na tyle, że jest stosowana do rozwiązywania wielu rzeczywistych problemów. Obecnie utrzymuje ją społeczność ponad 800 programistów. Bardzo często są to osoby, które zdecydowały się na współudział w rozwoju tego projektu po tym, jak korzystały z niego podczas codziennej pracy z danymi.

W całej książce importuję bibliotekę pandas przy użyciu następującej konwencji:

```
In [1]: import pandas as pd
```

W związku z tym we wszystkich miejscach kodu, gdzie zobaczysz zapis `pd.`, miej na uwadze, że dany fragment odwołuje się do biblioteki pandas. W związku z tym, że bardzo często będziemy również korzystać z modułów `Series` i `DataFrame`, warto załadować je do swojej lokalnej przestrzeni nazw:

```
In [2]: from pandas import Series, DataFrame
```

5.1. Wprowadzenie do struktur danych biblioteki pandas

Aby móc korzystać z biblioteki pandas, musisz poznać jej dwie główne struktury danych: serie i ramki danych. Nie są to rozwiązania uniwersalne, ale struktury te tworzą solidne i proste w zastosowaniu podstawy wspomagające rozwiązywanie większości problemów.

Obiekt Series

Seria (ang. *Series*) to jednowymiarowy obiekt przypominający tablicę. Składa się on z sekwencji wartości (typy tych wartości są podobne do typów obsługiwanych przez pakiet NumPy) i związanej z danymi tablicy etykiet określanej mianem **indeksu**. Serię najprościej jest utworzyć na podstawie tablicy danych:

```
In [11]: obj = pd.Series([4, 7, -5, 3])
```

```
In [12]: obj
```

```
Out[12]:
```

```
0    4
```

```
1    7
```

```
2   -5
```

```
3    3
```

```
dtype: int64
```

W sesji interaktywnej serie są wyświetlane tak, że po lewej stronie znajduje się indeks, a po prawej wartości odpowiadające poszczególnym elementom indeksu. Nie określiliśmy indeksu danych, ale w takiej sytuacji generowany jest domyślny indeks w postaci liczb całkowitych od 0 do $N-1$, gdzie N jest długością utworzonych danych. W celu wyświetlenia wartości serii skorzystaj z metody `values`, a w celu wyświetlenia indeksu — z metody `index`:

```
In [13]: obj.values
```

```
Out[13]: array([ 4,  7, -5,  3])
```

```
In [14]: obj.index # Przypomina działanie funkcji range(4).
```

```
Out[14]: RangeIndex(start=0, stop=4, step=1)
```

Często będziesz chciał utworzyć obiekt typu `Series` z indeksem identyfikującym każdy element serii za pomocą etykiety:

```
In [15]: obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
```

```
In [16]: obj2
```

```
Out[16]:
```

```
d    4
```

```
b    7
```

```
a   -5
```

```
c    3
```

```
dtype: int64
```

```
In [17]: obj2.index
```

```
Out[17]: Index(['d', 'b', 'a', 'c'], dtype='object')
```

W celu wybrania pojedynczej wartości lub zbioru wartości możesz — w przeciwieństwie do tablic NumPy — korzystać z etykiet umieszczonych w indeksie:

```
In [18]: obj2['a']
```

```
Out[18]: -5
```

```
In [19]: obj2['d'] = 6
```

```
In [20]: obj2[['c', 'a', 'd']]
```

```
Out[20]:
```

```
c    3
```

```
a   -5
```

```
d    6
```

```
dtype: int64
```

W zaprezentowanym przykładzie ['c', 'a', 'd'] stanowi listę indeksów — jak widzisz, indeksy nie muszą być wartościami liczbowymi, mogą być również łańcuchami.

Korzystanie z funkcji biblioteki NumPy, wykonywanie operacji takich jak filtrowanie za pomocą tablicy wartości, mnożenie macierzy lub stosowanie standardowych funkcji matematycznych nie rozwiązuje powiązania pomiędzy indeksem a wartością:

```
In [21]: obj2[obj2 > 0]
Out[21]:
d    6
b    7
c    3
dtype: int64
```

```
In [22]: obj2 * 2
Out[22]:
d    12
b    14
a   -10
c     6
dtype: int64
```

```
In [23]: np.exp(obj2)
Out[23]:
d    403.428793
b   1096.633158
a     0.006738
c    20.085537
dtype: float64
```

Obiekty typu Series można porównać do uporządkowanych słowników o określonej długości — w przypadku obu struktur mamy do czynienia z przypisaniem wartości indeksu do wartości danych. Obiekty Series można stosować w wielu kontekstach, w których używa się słowników:

```
In [24]: 'b' in obj2
Out[24]: True

In [25]: 'e' in obj2
Out[25]: False
```

Jeżeli dysponujesz danymi w formie słownika, to możesz przekształcić go na serię. Wystarczy przekazać go do funkcji Series:

```
In [26]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}

In [27]: obj3 = pd.Series(sdata)

In [28]: obj3
Out[28]:
Ohio    35000
Oregon   16000
Texas    71000
Utah     5000
dtype: int64
```

Jeżeli do funkcji Series przekazujesz tylko słownik, to w wygenerowanej serii posortowane klucze słownika będą pełniły funkcję indeksu. To domyślne zachowanie możesz obejść, przekazując klucze słownika w zaplanowanej przez siebie kolejności:

```
In [29]: states = ['California', 'Ohio', 'Oregon', 'Texas']
```

```
In [30]: obj4 = pd.Series(sdata, index=states)
```

```
In [31]: obj4
Out[31]:
California  NaN
Ohio        35000.0
Oregon      16000.0
Texas       71000.0
dtype: float64
```

W zaprezentowanym przykładzie trzy wartości znajdujące się w słowniku `sdata` zostały wyświetlone w określonej wcześniej kolejności. Nie znaleziono wartości klucza `'California'`, a więc przypisano mu wartość `NaN` (nie-liczba) — w bibliotece `pandas` brakujące wartości są oznaczane właśnie w ten sposób. Klucz `'Utah'` nie został uwzględniony w liście `states`, a więc nie umieszczono go w obiekcie wyjściowym.

Brakujące wartości (brakujące dane) będą czasem również określał mianem wartości `NA`. Funkcje biblioteki `pandas` `isnull` i `notnull` mogą być użyte w celu wykrycia brakujących danych:

```
In [32]: pd.isnull(obj4)
Out[32]:
California    True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

```
In [33]: pd.notnull(obj4)
Out[33]:
California    False
Ohio          True
Oregon        True
Texas         True
dtype: bool
```

Obiekty typu `Series` umożliwiają również korzystanie z tych funkcji jak z ich metod:

```
In [34]: obj4.isnull()
Out[34]:
California    True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

Zagadnienia związane z obsługą brakujących danych opiszę bardziej szczegółowo w rozdziale 7.

Przydatną cechą obiektów typu `Series` jest to, że automatycznie wyrównują one wartości na podstawie indeksu podczas wykonywania operacji arytmetycznych:

```
In [35]: obj3
Out[35]:
Ohio        35000
Oregon      16000
Texas       71000
Utah         5000
dtype: int64

In [36]: obj4
Out[36]:
```

```

California NaN
Ohio      35000.0
Oregon    16000.0
Texas     71000.0
dtype: float64

In [37]: obj3 + obj4
Out[37]:
California NaN
Ohio      70000.0
Oregon    32000.0
Texas    142000.0
Utah      NaN
dtype: float64

```

Równanie danych zostanie później opisane w sposób bardziej szczegółowy. Jeżeli masz doświadczenie w pracy z bazami danych, to możesz to traktować jak operację `join`.

Obiekty `Series` i ich indeksy mają atrybut `name`, który integruje je z innymi kluczowymi elementami biblioteki `pandas`:

```

In [38]: obj4.name = 'population'

In [39]: obj4.index.name = 'state'

In [40]: obj4
Out[40]:
state
California NaN
Ohio      35000.0
Oregon    16000.0
Texas     71000.0
Name: population, dtype: float64

```

Indeks obiektu `Series` może być modyfikowany w miejscu za pomocą operacji przypisania:

```

In [41]: obj
Out[41]:
0      4
1      7
2     -5
3      3
dtype: int64

In [42]: obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']

In [43]: obj
Out[43]:
Bob      4
Steve    7
Jeff    -5
Ryan     3
dtype: int64

```

Obiekt `DataFrame`

Obiekt `DataFrame` (**ramka danych**) jest prostokątną tabelą danych. Zawiera ona uporządkowany zbiór kolumn, a w każdej kolumnie może znaleźć się wartość innego typu (wartość liczbową, łańcuch znaków, wartość logiczną itd.). Ramki danych posiadają indeksy wierszy i kolumn. Można je postrzegać jako słownik obiektów typu `Series` współdzielących ten sam indeks. Wewnętrznie Python nie

przechowuje tego typu danych w formie listy, słownika ani zbioru jednowymiarowych tablic — dane te są przechowywane w formie dwuwymiarowych bloków. Wyjaśnianie wewnętrznej struktury obiektów typu `DataFrame` wykracza poza zakres tematyczny tej książki.



Co prawda obiekt `DataFrame` ma charakter dwuwymiarowy, ale pomimo tego może być używany do reprezentacji danych o większej liczbie wymiarów — służy do tego indeksowanie hierarchiczne (zagadnienie to opiszę w rozdziale 8.), a także zaawansowane funkcje obsługi danych oferowane przez bibliotekę `pandas`.

Obiekty typu `DataFrame` mogą być tworzone na wiele różnych sposobów, ale najczęściej generuje się je na podstawie słownika list o równej długości lub tablic `NumPy`:

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002, 2003],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)
```

Otrzymany obiekt `DataFrame` będzie posiadał automatycznie przypisany indeks (mechanizm ten działa tak samo jak w przypadku obiektu `Series`), a kolumny będą posortowane:

```
In [45]: frame
Out[45]:
```

	pop	state	year
0	1.5	Ohio	2000
1	1.7	Ohio	2001
2	3.6	Ohio	2002
3	2.4	Nevada	2001
4	2.9	Nevada	2002
5	3.2	Nevada	2003

Jeżeli pracujesz w notatniku Jupyter Notebook, to obiekty `dataFrame` będą wyświetlane w formacie tabeli, która może być poprawnie wyświetlana przez przeglądarkę internetową.

Podczas pracy z dużymi ramkami danych warto korzystać z metody `head`, która wybiera tylko pięć pierwszych wierszy:

```
In [46]: frame.head()
Out[46]:
```

	pop	state	year
0	1.5	Ohio	2000
1	1.7	Ohio	2001
2	3.6	Ohio	2002
3	2.4	Nevada	2001
4	2.9	Nevada	2002

Możesz określić kolejność, w jakiej mają być ustawione kolumny obiektu `DataFrame`:

```
In [47]: pd.DataFrame(data, columns=['year', 'state', 'pop'])
Out[47]:
```

	year	state	pop
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4
4	2002	Nevada	2.9
5	2003	Nevada	3.2

Jeżeli podczas tej operacji przekazesz kolumnę, która nie znajduje się w słowniku, to zostanie ona dodana do obiektu `DataFrame`, ale zostanie wypełniona wartościami `NaN`:


```
In [48]: frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
....:                          index=['one', 'two', 'three', 'four',
....:                          'five', 'six'])
```

```
In [49]: frame2
Out[49]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

```
In [50]: frame2.columns
Out[50]: Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

Dostęp do kolumny obiektu DataFrame można uzyskać za pomocą notacji przypominającej notację słownikową lub za pomocą atrybutu (w obu przypadkach zwrócony zostanie obiekt Series):

```
In [51]: frame2['state']
Out[51]:
one      Ohio
two      Ohio
three    Ohio
four    Nevada
five    Nevada
six     Nevada
Name: state, dtype: object
```

```
In [52]: frame2.year
Out[52]:
one      2000
two      2001
three    2002
four     2001
five     2002
six      2003
Name: year, dtype: int64
```



Stosując w środowisku IPython technikę dostępu opartą na atrybucie (np. `frame2.year`), możesz korzystać z automatycznego uzupełniania nazw kolumn (klawisza *Tab*), co znacznie ułatwia pracę.

Składnia `frame2[kolumna]` działa z dowolną nazwą kolumny, a składnia `frame2.kolumna` działa tylko wtedy, gdy nazwa kolumny jest poprawną nazwą zmiennej Pythona.

Zauważ, że zwrócony obiekt Series ma ten sam indeks co obiekt DataFrame, a dodatkowo obu obiektom przypisywane są poprawne atrybuty `name`.

Dostęp do wierszy można uzyskać również za pomocą pozycji lub nazwy i specjalnego atrybutu `loc` (później przedstawię więcej informacji na ten temat):

```
In [53]: frame2.loc['three']
Out[53]:
year      2002
state    Ohio
pop       3.6
debt     NaN
Name: three, dtype: object
```

Kolumny mogą być modyfikowane za pomocą operacji przypisywania. Na przykład do pustej kolumny 'debt' można przypisać wartość skalarną lub tablicę z wieloma wartościami:

```
In [54]: frame2['debt'] = 16.5
```

```
In [55]: frame2
```

```
Out[55]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	16.5
two	2001	Ohio	1.7	16.5
three	2002	Ohio	3.6	16.5
four	2001	Nevada	2.4	16.5
five	2002	Nevada	2.9	16.5
six	2003	Nevada	3.2	16.5

```
In [56]: frame2['debt'] = np.arange(6.)
```

```
In [57]: frame2
```

```
Out[57]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	0.0
two	2001	Ohio	1.7	1.0
three	2002	Ohio	3.6	2.0
four	2001	Nevada	2.4	3.0
five	2002	Nevada	2.9	4.0
six	2003	Nevada	3.2	5.0

Podczas przypisywania list lub tablic do kolumny długość przypisywanej zmiennej musi być równa długości obiektu DataFrame. W przypadku przypisywania obiektu typu Series etykiety tego obiektu zostaną wyrównane zgodnie z indeksem obiektu DataFrame, a we wszystkie dziury zostaną wstawione wartości NaN:

```
In [58]: val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
```

```
In [59]: frame2['debt'] = val
```

```
In [60]: frame2
```

```
Out[60]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN

Przypisywanie nieistniejącej kolumny spowoduje utworzenie nowej kolumny. Słowo kluczowe `del`, podobnie jak w przypadku słowników, jest używane do kasowania kolumn.

W celu zaprezentowania działania słowa kluczowego `del` utworzę nową kolumnę wartości logicznych, w której umieszczę wartość `True` wszędzie tam, gdzie w kolumnie `state` znajduje się łańcuch `'Ohio'`:

```
In [61]: frame2['eastern'] = frame2.state == 'Ohio'
```

```
In [62]: frame2
```

```
Out[62]:
```

	year	state	pop	debt	eastern
one	2000	Ohio	1.5	NaN	True
two	2001	Ohio	1.7	-1.2	True
three	2002	Ohio	3.6	NaN	True

```
four    2001    Nevada    2.4    -1.5    False
five    2002    Nevada    2.9    -1.7    False
six     2003    Nevada    3.2     NaN    False
```



Nowe kolumny nie mogą być tworzone za pomocą składni `frame2.eastern`.

Utworzoną przed chwilą kolumnę można usunąć za pomocą metody `del`:

```
In [63]: del frame2['eastern']
```

```
In [64]: frame2.columns
```

```
Out[64]: Index(['year', 'state', 'pop', 'debt'], dtype='object')
```



Kolumna zwrócona w wyniku indeksowania obiektu `DataFrame` jest widokiem oryginalnych danych, a nie ich kopią. W związku z tym wszelkie operacje modyfikujące zwrócony obiekt `Series` w miejscu będą modyfikowały również obiekt `DataFrame`. Kolumna może zostać skopiowana w sposób jawny za pomocą metody `copy` obiektu `Series`.

Inną popularną formą danych jest zagnieżdżony słownik słowników:

```
In [65]: pop = {'Nevada': {2001: 2.4, 2002: 2.9},
....:         'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
```

Jeżeli zagnieżdżony słownik zostanie przekazany do obiektu `DataFrame`, to biblioteka `pandas` potraktuje klucze zewnętrznego słownika jako kolumny, a klucze wewnętrznego słownika jako indeksy wierszy:

```
In [66]: frame3 = pd.DataFrame(pop)
```

```
In [67]: frame3
```

```
Out[67]:
```

```
      Nevada  Ohio
2000     NaN   1.5
2001     2.4   1.7
2002     2.9   3.6
```

Obiekt `DataFrame` może zostać transponowany (poddany operacji zamiany wierszy z kolumnami) za pomocą składni podobnej do tej, która była używana w przypadku transponowania tablic `NumPy`:

```
In [68]: frame3.T
```

```
Out[68]:
```

```
      2000  2001  2002
Nevada  NaN   2.4   2.9
Ohio     1.5   1.7   3.6
```

Klucze w wewnętrznych słownikach są łączone i sortowane w celu utworzenia indeksu, ale operacja ta nie jest przeprowadzana w przypadku jawnego określenia indeksu:

```
In [69]: pd.DataFrame(pop, index=[2001, 2002, 2003])
```

```
Out[69]:
```

```
      Nevada  Ohio
2001     2.4   1.7
2002     2.9   3.6
2003     NaN   NaN
```

W podobny sposób traktowane są słowniki obiektów `Series`:

```
In [70]: pdata = {'Ohio': frame3['Ohio'][:-1],
....:            'Nevada': frame3['Nevada'][:2]}

In [71]: pd.DataFrame(pdata)
Out[71]:
```

	Nevada	Ohio
2000	NaN	1.5
2001	2.4	1.7

W tabeli 5.1 znajdziesz listę wszystkich obiektów, które możesz przekazać do konstruktora obiektu DataFrame.

Tabela 5.1. Typy danych obsługiwane przez konstruktor ramki danych

Typ	Uwagi
Dwuwymiarowa tablica <i>ndarray</i>	Macierz danych; umożliwia przekazanie dodatkowych etykiet wierszy i kolumn.
Słownik tablic, list lub krotek	Każda sekwencja staje się kolumną ramki danych; wszystkie sekwencje muszą być tej samej długości.
Tablica z rekordami o strukturze zgodnej z NumPy	Dane traktowane tak samo jak w przypadku słownika tablic.
Słownik serii	Każda wartość staje się kolumną; w razie niezdefiniowania indeksu w sposób jawny klucze poszczególnych serii są łączone w unię w celu utworzenia indeksu wierszy ramki danych.
Słownik słowników	Każdy wewnętrzny słownik staje się kolumną; klucze są łączone w unię w celu utworzenia indeksu wierszy (tak samo jak w przypadku słownika serii).
Lista słowników lub serii	Każdy element staje się wierszem ramki danych; unia kluczy słownika lub indeksów serii staje się etykietami kolumn ramki danych.
Lista list lub krotek	Traktowana tak samo jak dwuwymiarowa tablica <i>ndarray</i> .
Inna ramka danych	W razie nieprzekazania indeksów w sposób jawny wczytywane są indeksy ramki danych.
Obiekt NumPy <i>MaskedArray</i>	Obiekt jest traktowany tak samo jak dwuwymiarowa tablica <i>ndarray</i> , ale maskowane wartości są traktowane jako brakujące dane.

Jeżeli elementy `index` (indeks) i `columns` (kolumny) mają zdefiniowane atrybuty `name` (nazwa), to wartości przypisane do tych atrybutów zostaną również wyświetlone wraz z zawartością ramki danych:

```
In [72]: frame3.index.name = 'year'; frame3.columns.name = 'state'

In [73]: frame3
Out[73]:
```

state	Nevada	Ohio
year		
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

Atrybut `values` obiektu DataFrame zwraca dane w postaci dwuwymiarowej tablicy *ndarray*:

```
In [74]: frame3.values
Out[74]:
```

```
array([[ nan,  1.5],
       [ 2.4,  1.7],
       [ 2.9,  3.6]])
```

Jeżeli kolumny ramki danych są danymi różnego typu, to wszystkie dane wszystkich kolumn zostaną umieszczone w tablicy:

```
In [75]: frame2.values
Out[75]:
array([[2000, 'Ohio', 1.5, nan],
       [2001, 'Ohio', 1.7, -1.2],
       [2002, 'Ohio', 3.6, nan],
       [2001, 'Nevada', 2.4, -1.5],
       [2002, 'Nevada', 2.9, -1.7],
       [2003, 'Nevada', 3.2, nan]], dtype=object)
```

Obiekty index

Indeksy (obiekty index) są używane do przechowywania etykiet osi lub innych metadanych, takich jak np. nazwy osi. Tablica lub inna sekwencja etykiet może zostać użyta podczas tworzenia serii lub ramki danych w celu jawnego zdefiniowania indeksu:

```
In [76]: obj = pd.Series(range(3), index=['a', 'b', 'c'])
```

```
In [77]: index = obj.index
```

```
In [78]: index
Out[78]: Index(['a', 'b', 'c'], dtype='object')
```

```
In [79]: index[1:]
Out[79]: Index(['b', 'c'], dtype='object')
```

Indeksy są obiektami niemodyfikowalnymi, a więc użytkownik nie może ich zmieniać:

```
index[1] = 'd' # Błąd typu (TypeError)
```

Niemodyfikowalność sprawia, że współdzielenie obiektów typu `index` pomiędzy strukturami danych jest bezpieczniejsze:

```
In [80]: labels = pd.Index(np.arange(3))
```

```
In [81]: labels
Out[81]: Int64Index([0, 1, 2], dtype='int64')
```

```
In [82]: obj2 = pd.Series([1.5, -2.5, 0], index=labels)
```

```
In [83]: obj2
Out[83]:
0    1.5
1   -2.5
2     0.0
dtype: float64
```

```
In [84]: obj2.index is labels
Out[84]: True
```



Większość problemów analitycznych można rozwiązać bez korzystania z indeksów, ale mechanikę działania indeksów należy zrozumieć, ponieważ niektóre operacje generują dane wyjściowe zawierające indeksy.

Indeksy zachowują się jak tablice, ale również jak zbiór danych o stałym rozmiarze:

```
In [85]: frame3
Out[85]:
state Nevada Ohio
year
2000      NaN  1.5
2001      2.4  1.7
2002      2.9  3.6

In [86]: frame3.columns
Out[86]: Index(['Nevada', 'Ohio'], dtype='object', name='state')

In [87]: 'Ohio' in frame3.columns
Out[87]: True

In [88]: 2003 in frame3.index
Out[88]: False
```

W przeciwieństwie do zbiorów Pythona indeksy pandas mogą zawierać zdublikowane etykiety:

```
In [89]: dup_labels = pd.Index(['foo', 'foo', 'bar', 'bar'])

In [90]: dup_labels
Out[90]: Index(['foo', 'foo', 'bar', 'bar'], dtype='object')
```

Przeprowadzenie operacji wyboru ze zdublowanymi etykietami spowoduje wybranie wszystkich wystąpień danej etykiety.

Każdy indeks obsługuje wiele metod i własności, które mogą przydać się podczas analizy umieszczonych w nim danych. W tabeli 5.2 przedstawiono najczęściej używane metody i własności obiektów typu `index`.

Tabela 5.2. Wybrane metody i własności indeksów

Metoda	Opis
<code>append</code>	Łączy obiekty typu indeks w celu utworzenia nowego indeksu.
<code>difference</code>	Zwraca różnicę zbiorów w postaci indeksu.
<code>insertion</code>	Zwraca efekt operacji wstawiania.
<code>union</code>	Zwraca efekt operacji sumowania.
<code>isin</code>	Generuje tablicę wartości logicznych informujących o tym, czy każda z wartości znajduje się w przekazanym ciągu.
<code>delete</code>	Tworzy nowy indeks po usunięciu elementu znajdującego się pod indeksem <code>i</code> .
<code>drop</code>	Tworzy nowy indeks po usunięciu przekazanych wartości.
<code>insert</code>	Tworzy nowy indeks po wstawieniu elementu pod indeksem <code>i</code> .
<code>is_monotonic</code>	Zwraca <code>True</code> , jeżeli każdy element jest większy od poprzedniego elementu (lub jest mu równy).
<code>is_unique</code>	Zwraca <code>True</code> , jeżeli indeks nie zawiera zdublikowanych wartości.
<code>unique</code>	Tworzy tablicę unikalnych wartości indeksu.

5.2. Podstawowe funkcjonalności

W tym podrozdziale znajdziesz informacje na temat podstawowych mechanizmów obsługi danych umieszczonych w obiektach typu `Series` i `DataFrame`. W kolejnych rozdziałach zgłębię zagadnienia związane z analizą danych i ich przekształcaniem za pomocą biblioteki `pandas`. Książka, którą trzymasz w ręku, nie jest dokładną dokumentacją biblioteki `pandas`. Pisząc ją, chciałem się skupić na najważniejszych możliwościach oferowanych przez ten pakiet. Rzadziej używane funkcje biblioteki `pandas` możesz zgłębić samodzielnie podczas pracy z danymi.

Uaktualnianie indeksu

Jedną z najważniejszych metod obiektów `pandas` jest metoda `reindex` (uaktualnianie indeksu). Umożliwia ona utworzenie nowego obiektu z danymi *dopasowanymi* do nowego indeksu. Przyjrzyj się następującemu przykładowi:

```
In [91]: obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])

In [92]: obj
Out[92]:
d    4.5
b    7.2
a   -5.3
c    3.6
dtype: float64
```

Wywołanie metody `reindex` na serii powoduje zmianę kolejności danych — przystosowanie jej do nowego indeksu i wprowadzenie brakujących wartości w miejsca nowych indeksów:

```
In [93]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])

In [94]: obj2
Out[94]:
a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN
dtype: float64
```

Podczas zmiany indeksów danych o charakterze szeregu (np. szeregów czasowych) konieczne może okazać się przeprowadzenie operacji takich jak interpolacja lub filtrowanie. W celu wykonania takiej operacji należy skorzystać z opcji `method` i np. metody `ffill`, która wypełnia wartości do przodu:

```
In [95]: obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])

In [96]: obj3
Out[96]:
0    blue
2  purple
4  yellow
dtype: object

In [97]: obj3.reindex(range(6), method='ffill')
Out[97]:
0    blue
1    blue
```

```
2 purple
3 purple
4 yellow
5 yellow
dtype: object
```

Metoda `reindex` w przypadku obiektu `DataFrame` może zmieniać kolejność (wierszy) indeksu, kolumn lub obu tych elementów. W przypadku przekazania tylko sekwencji wykonywana jest jedynie operacja uaktualniania indeksów wierszy:

```
In [98]: frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
....:                        index=['a', 'c', 'd'],
....:                        columns=['Ohio', 'Texas', 'California'])
```

```
In [99]: frame
Out[99]:
   Ohio  Texas  California
a      0      1           2
c      3      4           5
d      6      7           8
```

```
In [100]: frame2 = frame.reindex(['a', 'b', 'c', 'd'])
```

```
In [101]: frame2
Out[101]:
   Ohio  Texas  California
a  0.0    1.0           2.0
b  NaN    NaN           NaN
c  3.0    4.0           5.0
d  6.0    7.0           8.0
```

W celu uaktualnienia indeksu kolumn należy skorzystać ze słowa kluczowego `columns`:

```
In [102]: states = ['Texas', 'Utah', 'California']
```

```
In [103]: frame.reindex(columns=states)
```

```
Out[103]:
   Texas  Utah  California
a      1   NaN           2
c      4   NaN           5
d      7   NaN           8
```

W tabeli 5.3 wymieniono więcej argumentów metody `reindex`.

Tabela 5.3. Argumenty uaktualniania indeksu

Argument	Opis
<code>index</code>	W charakterze indeksu używana jest nowa sekwencja. Może być to egzemplarz obiektu <code>index</code> lub dowolny inny sekwencyjny obiekt Pythona. Indeks zostanie użyty bezpośrednio, bez kopiowania.
<code>method</code>	Metoda interpolacji (wypełniania) — <code>ffill</code> wypełnia w przód, a <code>bfill</code> wypełnia wstecz.
<code>fill_value</code>	Wartość, która wypełnia brakujące wartości przypisywane nowym indeksom.
<code>limit</code>	Maksymalny rozmiar przerwy (wyrażony za pomocą liczby elementów) do wypełnienia podczas wypełniania w przód lub w tył.
<code>tolerance</code>	Maksymalny rozmiar przerwy (wyrażony za pomocą odległości bezwzględnej) do wypełnienia w przypadku niedokładnych dopasowań podczas wypełniania w przód lub w tył.

Tabela 5.3. Argumenty uaktualniania indeksu — ciąg dalszy

Argument	Opis
level	Dopasowuje prosty indeks na poziomie wieloindeksu; w przeciwnym wypadku wybiera jego podzbiór.
copy	W przypadku wartości True zawsze kopiuje dane (dotyczy to również sytuacji, w której nowy indeks jest taki sam jak stary); w przypadku wartości False nie kopiuje danych, gdy indeksy są identyczne.

W dalszej części książki przedstawię więcej informacji na temat bardziej zwięzłego uaktualniania indeksów poprzez indeksowanie etykiet metodą `loc`. Wielu użytkowników Pythona korzysta tylko z tej techniki:

```
In [104]: frame.loc[['a', 'b', 'c', 'd'], states]
Out[104]:
```

	Texas	Utah	California
a	1.0	NaN	2.0
b	NaN	NaN	NaN
c	4.0	NaN	5.0
d	7.0	NaN	8.0

Odrzucanie elementów osi

Odrzucanie jednego elementu lub kilku elementów osi jest proste do przeprowadzenia, jeżeli dysponujesz tablicą indeksu lub listą bez tych elementów. Metoda `drop` zwraca nowy obiekt, z którego osi usunięto wybrane wartości (może to się przydać podczas czyszczenia danych i wykonywania operacji logicznych):

```
In [105]: obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])

In [106]: obj
Out[106]:
```

a	0.0
b	1.0
c	2.0
d	3.0
e	4.0

```
dtype: float64

In [107]: new_obj = obj.drop('c')
In [108]: new_obj
Out[108]:
```

a	0.0
b	1.0
d	3.0
e	4.0

```
dtype: float64

In [109]: obj.drop(['d', 'c'])
Out[109]:
```

a	0.0
b	1.0
e	4.0

```
dtype: float64
```

W przypadku ramki danych wartości indeksu mogą być usuwane z dowolnej osi. Aby zademonstrować tę możliwość, najpierw utwórzmy przykładowy obiekt typu `DataFrame`:

```
In [110]: data = pd.DataFrame(np.arange(16).reshape((4, 4)),
.....: index=['Ohio', 'Colorado', 'Utah', 'New York'],
.....: columns=['one', 'two', 'three', 'four'])
```

```
In [111]: data
```

```
Out[111]:
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

Wywołanie metody `drop` z sekwencją etykiet spowoduje odrzucenie wartości z wierszy oznaczonych tymi etykietami (z osi 0):

```
In [112]: data.drop(['Colorado', 'Ohio'])
```

```
Out[112]:
```

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

Wartości znajdujące się w wybranych kolumnach można odrzucać za pomocą argumentu `axis=1` lub `axis='columns'`:

```
In [113]: data.drop('two', axis=1)
```

```
Out[113]:
```

	one	three	four
Ohio	0	2	3
Colorado	4	6	7
Utah	8	10	11
New York	12	14	15

```
In [114]: data.drop(['two', 'four'], axis='columns')
```

```
Out[114]:
```

	one	three
Ohio	0	2
Colorado	4	6
Utah	8	10
New York	12	14

Wiele funkcji modyfikujących rozmiar lub kształt obiektów typu `Series` lub `DataFrame` (przykładem takiej funkcji jest `drop`) może działać *w miejscu* bez zwracania nowego obiektu. Wystarczy skorzystać z argumentu `inplace`:

```
In [115]: obj.drop('c', inplace=True)
```

```
In [116]: obj
```

```
Out[116]:
```

```
a  0.0
b  1.0
d  3.0
e  4.0
dtype: float64
```

Zachowaj ostrożność, korzystając z argumentu `inplace`, ponieważ jego użycie powoduje nieodwracalne kasowanie odrzucanych elementów.

Indeksowanie, wybieranie i filtrowanie

Indeksowanie serii (obiekt[...]) działa analogicznie do indeksowania tablic NumPy, ale zamiast z wartości całkowitoliczbowych musisz korzystać z wartości przypisanych do indeksu serii. Oto ilustrujące to przykłady:

```
In [117]: obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
```

```
In [118]: obj
Out[118]:
a    0.0
b    1.0
c    2.0
d    3.0
dtype: float64
```

```
In [119]: obj['b']
Out[119]: 1.0
```

```
In [120]: obj[1]
Out[120]: 1.0
```

```
In [121]: obj[2:4]
Out[121]:
c    2.0
d    3.0
dtype: float64
```

```
In [122]: obj[['b', 'a', 'd']]
Out[122]:
b    1.0
a    0.0
d    3.0
dtype: float64
```

```
In [123]: obj[[1, 3]]
Out[123]:
b    1.0
d    3.0
dtype: float64
```

```
In [124]: obj[obj < 2]
Out[124]:
a    0.0
b    1.0
dtype: float64
```

Wycinki z etykietami działają nieco inaczej niż w normalnym Pythonie, bo do wycinanego zbioru zaliczany jest również element końcowy:

```
In [125]: obj['b':'c']
Out[125]:
b    1.0
c    2.0
dtype: float64
```

Przypisywanie wartości za pomocą tych metod modyfikuje wybraną sekcję serii:

```
In [126]: obj['b':'c'] = 5
```

```
In [127]: obj
Out[127]:
```

```
a    0.0
b    5.0
c    5.0
d    3.0
dtype: float64
```

Indeks ramki danych umożliwia uzyskanie dostępu do wybranej kolumny lub wybranego ciągu kolumn (należy skorzystać z pojedynczej wartości lub z sekwencji):

```
In [128]: data = pd.DataFrame(np.arange(16).reshape((4, 4)),
.....:                        index=['Ohio', 'Colorado', 'Utah', 'New York'],
.....:                        columns=['one', 'two', 'three', 'four'])
```

```
In [129]: data
Out[129]:
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
In [130]: data['two']
Out[130]:
```

Ohio	1
Colorado	5
Utah	9
New York	13

Name: two, dtype: int64

```
In [131]: data[['three', 'one']]
Out[131]:
```

	three	one
Ohio	2	0
Colorado	6	4
Utah	10	8
New York	14	12

Istnieje kilka specjalnych przypadków takiego indeksowania. Pierwszym z nich jest operacja wycinania lub wybierania danych przy użyciu tablicy wartości logicznych:

```
In [132]: data[:2]
Out[132]:
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

```
In [133]: data[data['three'] > 5]
Out[133]:
```

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

Dla uproszczenia pracy skorzystaliśmy ze składni wybierania danych `data[:2]`. Przekazanie do operatora pojedynczego elementu lub listy `[]` powoduje wybranie kolumn.

Kolejnym ze wspomnianych wcześniej przypadków jest zastosowanie operacji porównywania skalarów:

```
In [134]: data < 5
Out[134]:
```

	one	two	three	four
--	-----	-----	-------	------

Ohio	True	True	True	True
Colorado	True	False	False	False
Utah	False	False	False	False
New York	False	False	False	False

```
In [135]: data[data < 5] = 0
```

```
In [136]: data
```

```
Out[136]:
```

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

Jeśli chodzi o składnię, to w tym przypadku ramka danych zachowuje się jak dwuwymiarowa tablica NumPy.

Wybieranie za pomocą operatorów loc i iloc

Podczas obsługi indeksu w postaci etykiet wierszy ramki danych często przydają się operatory indeksowania `loc` i `iloc`. Pozwalają one na wybranie podzbioru wierszy i kolumn z ramki danych za pomocą notacji podobnej do notacji NymPy przy użyciu etykiet osi (`loc`) lub wartości całkowitoliczbowych (`iloc`).

W pierwszym przykładzie wybieram za pomocą etykiet pierwszy rząd i dwie kolumny:

```
In [137]: data.loc['Colorado', ['two', 'three']]
Out[137]:
```

two	5
three	6

```
Name: Colorado, dtype: int64
```

Teraz wykonam podobne operacje wyboru za pomocą liczb całkowitych i operatora `iloc`:

```
In [138]: data.iloc[2, [3, 0, 1]]
Out[138]:
```

four	11
one	8
two	9

```
Name: Utah, dtype: int64
```

```
In [139]: data.iloc[2]
Out[139]:
```

one	8
two	9
three	10
four	11

```
Name: Utah, dtype: int64
```

```
In [140]: data.iloc[[1, 2], [3, 0, 1]]
Out[140]:
```

	four	one	two
Colorado	7	0	5
Utah	11	8	9

Obie funkcje indeksowania działają poza pojedynczymi etykietami i listami etykiet również z wycinkami:

```
In [141]: data.loc[:, 'Utah', 'two']
Out[141]:
```

Ohio	0
------	---

```

Colorado 5
Utah      9
Name: two, dtype: int64

In [142]: data.iloc[:, :3][data.three > 5]
Out[142]:
      one  two  three
Colorado  0   5     6
Utah      8   9    10
New York 12  13    14

```

Jak widzisz, istnieje wiele sposobów wybierania i przegrupowywania danych umieszczonych w obiektach pakietu pandas. W tabeli 5.4 umieszczono krótkie podsumowanie takich operacji w kontekście ramki danych. W dalszej części tej książki dowiesz się, że istnieje wiele dodatkowych opcji obsługi indeksów hierarchicznych.

Tabela 5.4. Opcje indeksowania obiektu *DataFrame*

Typ	Uwagi
<code>df[wartość]</code>	Wybiera pojedynczą kolumnę lub sekwencję kolumn z ramki danych; specjalne przypadki użycia: tablica wartości binarnych (filtruje wiersze), wycinek (wycina wiersze), ramka danych z wartościami logicznymi (definiuje wartości na podstawie określonego kryterium).
<code>df.loc[wartość]</code>	Wybiera pojedynczy wiersz lub podzbiór wierszy z ramki danych na podstawie etykiet.
<code>df.loc[:, wartość]</code>	Wybiera pojedynczą kolumnę lub podzbiór kolumn na podstawie etykiet.
<code>df.loc[wartość1, wartość2]</code>	Wybiera wiersze oraz kolumny na podstawie etykiet.
<code>df.iloc[miejsce]</code>	Wybiera pojedynczy wiersz lub podzbiór wierszy z ramki danych na podstawie pozycji określonej za pomocą wartości całkowitoliczbowej.
<code>df.iloc[:, miejsce]</code>	Wybiera pojedynczą kolumnę lub podzbiór kolumn na podstawie pozycji określonej za pomocą wartości całkowitoliczbowej.
<code>df.iloc[miejsce_i, miejsce_j]</code>	Wybiera wiersze i kolumny na podstawie pozycji określonej za pomocą wartości całkowitoliczbowej.
<code>df.at[etykieta_i, etykieta_j]</code>	Wybiera pojedynczą wartość skalarną na podstawie etykiety wiersza i kolumny.
<code>df.iat[i, j]</code>	Wybiera pojedynczą wartość skalarną z wiersza i kolumny zdefiniowanych za pomocą pozycji (wartości całkowitoliczbowych).
metoda <code>reindex</code>	Wybiera wiersze lub kolumny na podstawie etykiet.
metody <code>get_value</code> i <code>set_value</code>	Wybór pojedynczej wartości na podstawie etykiet wiersza i kolumny.



Na początku procesu projektowania biblioteki pandas uważałem, że wybieranie kolumny za pomocą notacji ramkadanych[:, kolumna] jest zbyt kłopotliwe i naraża kod na potencjalne błędy, ponieważ wybieranie kolumn jest jedną z najczęściej wykonywanych operacji. Poszedłem na kompromis i upchnąłem techniki obsługi indeksów (etykiet, a także wartości całkowitoliczbowych) do operatora ix. W praktyce doprowadziło to do wielu problematycznych sytuacji z danymi, w których etykiety osi oznaczono etykietami w formie liczb całkowitych, dlatego zespół programistów pakietu pandas zdecydował się na utworzenie operatorów loc i iloc, które miały obsługiwać tylko odpowiednie indeksy w formie etykiet (loc) i liczb całkowitych (iloc).

Operator indeksowania ix jest wciąż dostępny, ale jego używanie jest niezalecane. Odradzam Ci korzystanie z niego.

Indeksy w postaci liczb całkowitych

Praca z obiektami pandas oznaczonymi indeksami w postaci liczb całkowitych jest czymś, co sprawia trudność wielu nowym użytkownikom. Wynika to z różnic składni indeksowania pomiędzy strukturami pandas a wbudowanymi strukturami Pythona — listami i krotkami. Prawdopodobnie nie uważasz, że poniższy kod może wygenerować błąd:

```
ser = pd.Series(np.arange(3.))
ser
ser[-1]
```

Pakiet pandas może rozpocząć indeksowanie od końca, ale ogólnie rzecz biorąc, w wielu sytuacjach może to doprowadzić do powstania błędów. W zaprezentowanym przykładzie indeks składa się z liczb 0, 1 i 2, ale pandas ma problem z określeniem tego, czego chce użytkownik (biblioteka nie wie, czy zastosowano indeksowanie za pomocą etykiet, czy za pomocą numeru pozycji):

```
In [144]: ser
Out[144]:
0    0.0
1    1.0
2    2.0
dtype: float64
```

W przypadku indeksów niemających formy liczb całkowitych nie ma problemu z dwuznacznością:

```
In [145]: ser2 = pd.Series(np.arange(3.), index=['a', 'b', 'c'])

In [146]: ser2[-1]
Out[146]: 2.0
```

W przypadku etykiet osi zawierających wartości liczbowe wybór danych będzie zawsze oparty na etykietach — dla ujednolicenia. W celu zwiększenia precyzji można skorzystać z atrybutu loc (dla etykiet) lub iloc (dla liczb całkowitych):

```
In [147]: ser[:1]
Out[147]:
0    0.0
dtype: float64

In [148]: ser.loc[:1]
Out[148]:
```

```
0    0.0
1    1.0
dtype: float64

In [149]: ser.iloc[:1]
Out[149]:
0    0.0
dtype: float64
```

Działania arytmetyczne i wyrównywanie danych

Ważną cechą pakietu pandas z punktu widzenia niektórych zastosowań jest sposób przeprowadzenia operacji arytmetycznych na obiektach z różnymi indeksami. Jeżeli podczas dodawania któraś z par indeksów nie jest taka sama, to wartości oznaczone takimi indeksami zostaną zaprezentowane w wyniku w postaci unii. Informacja dla osób mających doświadczenie w pracy z bazami danych: działanie to przypomina automatyczne wykonywanie zewnętrznej operacji łączenia (*outer join*) na etykietach indeksu. Przyjrzyj się następującemu przykładowi:

```
In [150]: s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])

In [151]: s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],
.....:                  index=['a', 'c', 'e', 'f', 'g'])

In [152]: s1
Out[152]:
a    7.3
c   -2.5
d    3.4
e    1.5
dtype: float64

In [153]: s2
Out[153]:
a   -2.1
c    3.6
e   -1.5
f    4.0
g    3.1
dtype: float64
```

Po dodaniu tych obiektów otrzymamy:

```
In [154]: s1 + s2
Out[154]:
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

Wewnętrzny mechanizm wyrównywania danych wstawia brakujące wartości w miejsca, w których etykiety się nie nakładają. Brakujące wartości przechodzą dalej do kolejnych operacji matematycznych.

W przypadku obiektu `DataFrame` wyrównaniu podlegają wiersze i kolumny:

```
In [155]: df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),
.....:                       index=['Ohio', 'Texas', 'Colorado'])
```



```
In [156]: df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),
.....:                        index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [157]: df1
Out[157]:
```

	b	c	d
Ohio	0.0	1.0	2.0
Texas	3.0	4.0	5.0
Colorado	6.0	7.0	8.0

```
In [158]: df2
Out[158]:
```

	b	d	e
Utah	0.0	1.0	2.0
Ohio	3.0	4.0	5.0
Texas	6.0	7.0	8.0
Oregon	9.0	10.0	11.0

W wyniku dodania do siebie tych dwóch obiektów powstanie ramka danych, której indeksy i kolumny są uniami kolumn i indeksów dodawanych obiektów:

```
In [159]: df1 + df2
Out[159]:
```

	b	c	d	e
Colorado	NaN	NaN	NaN	NaN
Ohio	3.0	NaN	6.0	NaN
Oregon	NaN	NaN	NaN	NaN
Texas	9.0	NaN	12.0	NaN
Utah	NaN	NaN	NaN	NaN

Kolumny c i e nie występują w obu dodawanych obiektach DataFrame, a więc w obiekcie wyjściowym są one wypełnione brakującymi wartościami. Takie samo zjawisko występuje również wtedy, gdy jakieś etykiety wierszy nie występują w obu obiektach.

Jeżeli dodasz do siebie dwa obiekty DataFrame, w których nie występują żadne wspólne etykiety kolumn i wierszy, to w wyniku znajdą się same wartości null:

```
In [160]: df1 = pd.DataFrame({'A': [1, 2]})

In [161]: df2 = pd.DataFrame({'B': [3, 4]})

In [162]: df1
Out[162]:
```

	A
0	1
1	2

```
In [163]: df2
Out[163]:
```

	B
0	3
1	4

```
In [164]: df1 - df2
Out[164]:
```

	A	B
0	NaN	NaN
1	NaN	NaN

Metody arytmetyczne i wypełnianie wartościami

Podczas wykonywania operacji matematycznych pomiędzy obiektami o różnych indeksach w sytuacji, gdy oś o danej etykiecie znajduje się tylko w jednym obiekcie, możesz wstawić określoną wartość, taką jak np. 0:

```
In [165]: df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),
.....:                      columns=list('abcd'))
```

```
In [166]: df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),
.....:                      columns=list('abcde'))
```

```
In [167]: df2.loc[1, 'b'] = np.nan
```

```
In [168]: df1
```

```
Out[168]:
```

	a	b	c	d
0	0.0	1.0	2.0	3.0
1	4.0	5.0	6.0	7.0
2	8.0	9.0	10.0	11.0

```
In [169]: df2
```

```
Out[169]:
```

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	4.0
1	5.0	NaN	7.0	8.0	9.0
2	10.0	11.0	12.0	13.0	14.0
3	15.0	16.0	17.0	18.0	19.0

Dodanie do siebie tych obiektów spowoduje umieszczenie wartości *NA* w miejscach, w których etykiety obu obiektów nie pokrywają się:

```
In [170]: df1 + df2
```

```
Out[170]:
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	NaN
1	9.0	NaN	13.0	15.0	NaN
2	18.0	20.0	22.0	24.0	NaN
3	NaN	NaN	NaN	NaN	NaN

Używając metody `add` na obiekcie `df1`, mogę przekazać obiekt `df2` oraz argument definiujący wartość używaną do wypełnienia (`fill_value`):

```
In [171]: df1.add(df2, fill_value=0)
```

```
Out[171]:
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	4.0
1	9.0	5.0	13.0	15.0	9.0
2	18.0	20.0	22.0	24.0	14.0
3	15.0	16.0	17.0	18.0	19.0

W tabeli 5.5 wymieniono metody operacji arytmetycznych wykonywanych na obiektach `Series` i `DataFrame`. Każda z metod ma swój odpowiednik rozpoczynający się od litery `r`, który charakteryzuje się odwróconymi argumentami. W związku z tym dwie poniższe metody dają taki sam efekt:

```
In [172]: 1 / df1
```

```
Out[172]:
```

	a	b	c	d
0	inf	1.000000	0.500000	0.333333
1	0.250000	0.200000	0.166667	0.142857
2	0.125000	0.111111	0.100000	0.090909

```
In [173]: df1.rdiv(1)
Out[173]:
```

	a	b	c	d
0	inf	1.000000	0.500000	0.333333
1	0.250000	0.200000	0.166667	0.142857
2	0.125000	0.111111	0.100000	0.090909

Tabela 5.5. Elastyczne metody arytmetyczne

Metoda	Opis
add, radd	Metody służące do dodawania (+).
sub, rsub	Metody służące do odejmowania (-).
div, rdiv	Metody służące do dzielenia (/).
floordiv, rfloordiv	Metody służące do dzielenia całkowitego (/).
mul, rmul	Metody służące do mnożenia (*).
pow, rpow	Metody służące do potęgowania (**).

Podczas uaktualniania indeksu obiektu typu Series lub DataFrame możesz również określić inną wartość używaną do wypełniania brakujących wartości:

```
In [174]: df1.reindex(columns=df2.columns, fill_value=0)
Out[174]:
```

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	0
1	4.0	5.0	6.0	7.0	0
2	8.0	9.0	10.0	11.0	0

Operacje pomiędzy obiektami DataFrame i Series

Operacje arytmetyczne pomiędzy obiektami DataFrame i Series są zdefiniowane podobnie jak w przypadku tablic NumPy o różnej liczbie wymiarów. Przyjrzyj się poniższemu przykładowi — różnicy pomiędzy dwuwymiarową tablicą i jednym z jej wierszy:

```
In [175]: arr = np.arange(12.).reshape((3, 4))
```

```
In [176]: arr
Out[176]:
array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5.,  6.,  7.],
       [ 8.,  9., 10., 11.]])
```

```
In [177]: arr[0]
Out[177]: array([ 0.,  1.,  2.,  3.] )
```

```
In [178]: arr - arr[0]
Out[178]:
array([[ 0.,  0.,  0.,  0.],
       [ 4.,  4.,  4.,  4.],
       [ 8.,  8.,  8.,  8.]])
```

Jak widzisz, podczas odejmowania `arr[0]` od `arr` operacja odejmowania jest wykonywana raz dla każdego wiersza. Zjawisko to określa się mianem **rozgłaszania**. Więcej informacji na jego temat znajdziesz w „Dodatku A”. W podobny sposób działają operacje pomiędzy obiektami DataFrame i Series:

```
In [179]: frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
.....:                        columns=list('bde'),
```

```

.....:                index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [180]: series = frame.iloc[0]

In [181]: frame
Out[181]:
      b      d      e
Utah  0.0   1.0   2.0
Ohio  3.0   4.0   5.0
Texas 6.0   7.0   8.0
Oregon 9.0  10.0  11.0

In [182]: series
Out[182]:
b    0.0
d    1.0
e    2.0
Name: Utah, dtype: float64

```

Domyślnie operacje arytmetyczne pomiędzy obiektami DataFrame i Series dobierają indeks obiektu Series do kolumn obiektu DataFrame i — wykonując operację rozgłaszania — przechodzą w dół wierszy:

```

In [183]: frame - series
Out[183]:
      b      d      e
Utah  0.0   0.0   0.0
Ohio  3.0   3.0   3.0
Texas 6.0   6.0   6.0
Oregon 9.0   9.0   9.0

```

Jeżeli wartość indeksu nie zostanie znaleziona w kolumnie obiektu DataFrame lub w indeksie obiektu Series, to obiekty zostaną przeindeksowane w celu utworzenia unii:

```

In [184]: series2 = pd.Series(range(3), index=['b', 'e', 'f'])

In [185]: frame + series2
Out[185]:
      b      d      e      f
Utah  0.0  NaN   3.0  NaN
Ohio  3.0  NaN   6.0  NaN
Texas 6.0  NaN   9.0  NaN
Oregon 9.0  NaN  12.0  NaN

```

Jeżeli zamiast tego chcesz przeprowadzić operację rozgłaszania po kolumnach, dopasowując wiersze, musisz skorzystać z jednej z metod arytmetycznych, takich jak:

```

In [186]: series3 = frame['d']

In [187]: frame
Out[187]:
      b      d      e
Utah  0.0   1.0   2.0
Ohio  3.0   4.0   5.0
Texas 6.0   7.0   8.0
Oregon 9.0  10.0  11.0

In [188]: series3
Out[188]:
Utah    1.0
Ohio    4.0
Texas    7.0

```

```
Oregon 10.0
Name: d, dtype: float64

In [189]: frame.sub(series3, axis='index')
Out[189]:
```

	b	d	e
Utah	-1.0	0.0	1.0
Ohio	-1.0	0.0	1.0
Texas	-1.0	0.0	1.0
Oregon	-1.0	0.0	1.0

Przekazany numer osi określa *oś, po której ma zostać przeprowadzone dobieranie*. W zaprezentowanym przypadku chciałem przeprowadzić dobieranie po indeksie wiersza obiektu Dataframe (axis='index' lub axis=0), a następnie wykonać rozgłaszanie w poprzek.

Funkcje apply i map

Funkcje uniwersalne pakietu NumPy (metody przetwarzające tablice element po elemencie) działają również z obiektami pandas:

```
In [190]: frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'),
.....:                        index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [191]: frame
Out[191]:
```

	b	d	e
Utah	-0.204708	0.478943	-0.519439
Ohio	-0.555730	1.965781	1.393406
Texas	0.092908	0.281746	0.769023
Oregon	1.246435	1.007189	-1.296221

```
In [192]: np.abs(frame)
Out[192]:
```

	b	d	e
Utah	0.204708	0.478943	0.519439
Ohio	0.555730	1.965781	1.393406
Texas	0.092908	0.281746	0.769023
Oregon	1.246435	1.007189	1.296221

Kolejną często wykonywaną operacją jest przetwarzanie każdego wiersza lub kolumny tablicy jednowymiarowej za pomocą funkcji. Można to zrobić za pomocą metody apply obiektu DataFrame:

```
In [193]: f = lambda x: x.max() - x.min()

In [194]: frame.apply(f)
Out[194]:
```

b	1.802165
d	1.684034
e	2.689627

```
dtype: float64
```

Funkcja f oblicza różnicę pomiędzy maksymalną i minimalną wartością obiektu Series. W zaprezentowanym przykładzie została wywołana dla każdej kolumny obiektu frame. Wynikiem jest obiekt Series, w którym indeksem są etykiety kolumn obiektu frame.

Jeżeli do funkcji apply przekazany zostanie argument axis='columns', to funkcja będzie tym razem wywoływana jednokrotnie dla poszczególnych wierszy:

```
In [195]: frame.apply(f, axis='columns')
Out[195]:
Utah    0.998382
Ohio    2.521511
Texas   0.676115
Oregon  2.542656
dtype: float64
```

Większość standardowych parametrów statystycznych macierzy, takich jak suma (sum) i średnia (mean), można wyznaczyć za pomocą metod obiektu DataFrame, a więc korzystanie z funkcji apply nie jest w ich przypadku konieczne.

Funkcja przekazana do funkcji apply nie musi zwracać wartości skalarnej. Może również zwrócić serię zawierającą wiele wartości:

```
In [196]: def f(x):
.....:     return pd.Series([x.min(), x.max()], index=['min', 'max'])

In [197]: frame.apply(f)
Out[197]:
```

	b	d	e
min	-0.555730	0.281746	-1.296221
max	1.246435	1.965781	1.393406

Istnieje również możliwość użycia funkcji Pythona przetwarzających element po elemencie. Załóżmy, że chcesz uzyskać sformatowany łańcuch na podstawie poszczególnych wartości zmiennoprzecinkowych znajdujących się w obiekcie frame. Możesz to zrobić za pomocą metody applymap:

```
In [198]: format = lambda x: '%.2f' % x

In [199]: frame.applymap(format)
Out[199]:
```

	b	d	e
Utah	-0.20	0.48	-0.52
Ohio	-0.56	1.97	1.39
Texas	0.09	0.28	0.77
Oregon	1.25	1.01	-1.30

Nazwa applymap (zastosuj mapowanie) wynika z tego, że obiekt Series dysponuje metodą map przeznaczoną do wykonywania funkcji przetwarzających dane element po elemencie:

```
In [200]: frame['e'].map(format)
Out[200]:
Utah    -0.52
Ohio     1.39
Texas     0.77
Oregon  -1.30
Name: e, dtype: object
```

Sortowanie i tworzenie rankingów

Sortowanie zbioru danych według określonych kryteriów to kolejny ważny element pakietu pandas. W celu wykonania sortowania leksykograficznego po wierszach lub kolumnach należy skorzystać z metody sort_index, która zwraca nowy, posortowany obiekt:

```
In [201]: obj = pd.Series(range(4), index=['d', 'a', 'b', 'c'])

In [202]: obj.sort_index()
Out[202]:
a    1
```

```
b 2
c 3
d 0
dtype: int64
```

Ramki danych mogą być sortowane po indeksie dowolnej osi:

```
In [203]: frame = pd.DataFrame(np.arange(8).reshape((2, 4)),
.....:                        index=['three', 'one'],
.....:                        columns=['d', 'a', 'b', 'c'])
```

```
In [204]: frame.sort_index()
```

```
Out[204]:
      d  a  b  c
one   4  5  6  7
three 0  1  2  3
```

```
In [205]: frame.sort_index(axis=1)
```

```
Out[205]:
      a  b  c  d
three 1  2  3  0
one   5  6  7  4
```

Dane są domyślnie sortowane w kolejności wzrastającej, ale mogą być również posortowane w kolejności malejącej:

```
In [206]: frame.sort_index(axis=1, ascending=False)
```

```
Out[206]:
      d  c  b  a
three 0  3  2  1
one   4  7  6  5
```

W celu posortowania serii według umieszczonych w niej wartości należy skorzystać z metody `sort_values`:

```
In [207]: obj = pd.Series([4, 7, -3, 2])
```

```
In [208]: obj.sort_values()
```

```
Out[208]:
2  -3
3   2
0   4
1   7
dtype: int64
```

Podczas sortowania serii domyślnie wszelkie brakujące wartości są umieszczane na końcu:

```
In [209]: obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])
```

```
In [210]: obj.sort_values()
```

```
Out[210]:
4  -3.0
5   2.0
0   4.0
2   7.0
1   NaN
3   NaN
dtype: float64
```

Podczas sortowania ramki danych możesz użyć danych z jednej lub z kilku kolumn w roli kluczy sortowania. W tym celu należy przekazać nazwę kolumny (lub nazwy kolumn) do opcji `by` metody `sort_values`:

```
In [211]: frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})
```

```
In [212]: frame
```

```
Out[212]:
```

```
   a  b
0  0  4
1  1  7
2  0 -3
3  1  2
```

```
In [213]: frame.sort_values(by='b')
```

```
Out[213]:
```

```
   a  b
2  0 -3
3  1  2
0  0  4
1  1  7
```

W celu wykonania sortowania po wielu kolumnach należy przekazać listę ich nazw:

```
In [214]: frame.sort_values(by=['a', 'b'])
```

```
Out[214]:
```

```
   a  b
2  0 -3
0  0  4
3  1  2
1  1  7
```

Tworzenie rankingu polega na przypisaniu rangi w postaci wartości od jednego do wartości równej liczbie poprawnych punktów danych tablicy. W celu utworzenia rankingu serii lub ramki danych należy skorzystać z metody `ran`. Metoda ta domyślnie zrywa powiązania, przypisując każdej grupie średnią wartość rangi:

```
In [215]: obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
```

```
In [216]: obj.rank()
```

```
Out[216]:
```

```
0  6.5
1  1.0
2  6.5
3  4.5
4  3.0
5  2.0
6  4.5
dtype: float64
```

Rangi mogą być również przypisywane według kolejności występowania w danych:

```
In [217]: obj.rank(method='first')
```

```
Out[217]:
```

```
0  6.0
1  1.0
2  7.0
3  4.0
4  3.0
5  2.0
6  5.0
dtype: float64
```

W zaprezentowanym przykładzie zamiast korzystać ze średniej rangi równej 6,5 dla elementów 0 i 2, skorzystano z wartości 6 i 7. Wynika to z tego, że etykieta 0 znajduje się przed etykietą 2.

Możliwe jest również tworzenie rankingów w kolejności malejącej:

```
# Wartościom przypisz maksymalną rangę w grupie.  
In [218]: obj.rank(ascending=False, method='max')  
Out[218]:  
0    2.0  
1    7.0  
2    2.0  
3    4.0  
4    5.0  
5    6.0  
6    4.0  
dtype: float64
```

W tabeli 5.6 znajdziesz listę dostępnych metod przełamujących powiązania.

Tabela 5.6. Metody przerywające powiązania z rangami

Metoda	Opis
'average'	Metoda domyślna: przypisuje średnią rangę każdemu elementowi w równej grupie.
'min'	Używa minimalnej rangi dla całej grupy.
'max'	Używa maksymalnej rangi dla całej grupy.
'first'	Przypisuje rangi w kolejności, w jakiej wartości występują w danych.
'dense'	Działa podobnie jak <code>method='min'</code> , ale rangi pomiędzy grupami zawsze wzrastają o 1, a nie o liczbę równą liczbie elementów w grupie.

W przypadku obiektu `DataFrame` możliwe jest określenie rang w wierszach lub kolumnach:

```
In [219]: frame = pd.DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],  
.....:                        'c': [-2, 5, 8, -2.5]})  
  
In [220]: frame  
Out[220]:  
   a    b    c  
0  0  4.3 -2.0  
1  1  7.0  5.0  
2  0 -3.0  8.0  
3  1  2.0 -2.5  
  
In [221]: frame.rank(axis='columns')  
Out[221]:  
   a    b    c  
0  2.0  3.0  1.0  
1  1.0  3.0  2.0  
2  2.0  1.0  3.0  
3  2.0  3.0  1.0
```

Indeksy osi ze zduplikowanymi etykietami

Dotychczas analizowaliśmy tylko przykłady, w których etykiety osi (wartości indeksu) były unikalne. Wiele funkcji pakietu `pandas` (np. funkcja `reindex`) wymaga tego, aby etykiety były unikalne, ale nie jest to obowiązkowe. Oto przykład małej serii ze zduplikowanymi wartościami indeksu:

```
In [222]: obj = pd.Series(range(5), index=['a', 'a', 'b', 'b', 'c'])  
  
In [223]: obj  
Out[223]:
```

```
a 0
a 1
b 2
b 3
c 4
dtype: int64
```

Unikalność etykiet można określić za pomocą własności `is_unique`:

```
In [224]: obj.index.is_unique
Out[224]: False
```

Wybór danych jest jedną z operacji, które działają inaczej z duplikatami. Indeksowanie etykiety przypisanej do wielu elementów zwraca serię. W przypadku pojedynczego elementu przypisanego do etykiety zwracana jest wartość skalarna:

```
In [225]: obj['a']
Out[225]:
a 0
a 1
dtype: int64
```

```
In [226]: obj['c']
Out[226]: 4
```

Może to skomplikować Twój kod, ponieważ typ danych uzyskanych w wyniku indeksowania zależy od tego, czy dana etykieta jest unikalna, czy się powtarza.

Ta sama logika dotyczy również indeksowania wierszy ramki danych:

```
In [227]: df = pd.DataFrame(np.random.randn(4, 3), index=['a', 'a', 'b', 'b'])
```

```
In [228]: df
Out[228]:
```

	0	1	2
a	0.274992	0.228913	1.352917
a	0.886429	-2.001637	-0.371843
b	1.669025	-0.438570	-0.539741
b	0.476985	3.248944	-1.021228

```
In [229]: df.loc['b']
Out[229]:
```

	0	1	2
b	1.669025	-0.438570	-0.539741
b	0.476985	3.248944	-1.021228

5.3. Podsumowywanie i generowanie statystyk opisowych

Obiekty pakietu pandas obsługują zestaw praktycznych metod matematycznych i statystycznych. Większość z nich dostarcza **statystycznych danych podsumowujących** lub służy do **redukcji**. Metody tego typu zwracają pojedynczą wartość (np. sumę lub średnią) wygenerowaną na podstawie serii lub serii wartości z wierszy lub kolumn ramki danych. W porównaniu z podobnymi metodami tablic NumPy charakteryzują się one wbudowanymi procedurami obsługi brakujących danych. Przyjrzyj się następującemu przykładowi małej ramki danych:

```
In [230]: df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],
.....:                      [np.nan, np.nan], [0.75, -1.3]],
.....:                      index=['a', 'b', 'c', 'd'],
.....:                      columns=['one', 'two'])
```

```
In [231]: df
Out[231]:
   one  two
a  1.40 NaN
b  7.10 -4.5
c   NaN NaN
d  0.75 -1.3
```

Wywołanie metody `sum` tej ramki danych zwróci serię zawierającą sumy poszczególnych kolumn:

```
In [232]: df.sum()
Out[232]:
one    9.25
two   -5.80
dtype: float64
```

Przekazanie argumentu `axis='columns'` lub `axis=1` spowoduje zwrócenie sum elementów umieszczonych w poszczególnych kolumnach:

```
In [233]: df.sum(axis='columns')
Out[233]:
a    1.40
b    2.60
c     NaN
d   -0.55
dtype: float64
```

Brakujące wartości nie są brane pod uwagę, dopóki cały wycinek (wiersz lub kolumna) nie zawiera samych wartości typu NA. Funkcję tę można wyłączyć za pomocą opcji `skipna`:

```
In [234]: df.mean(axis='columns', skipna=False)
Out[234]:
a     NaN
b    1.300
c     NaN
d   -0.275
dtype: float64
```

W tabeli 5.7 wymieniono najczęściej używane opcje metod redukujących.

Tabela 5.7. Opcje metod redukujących

Opcja	Opis
<code>axis</code>	Wybór osi, po której ma zostać przeprowadzona redukcja; w przypadku ramki danych 0 powoduje przeprowadzenie operacji na wierszach, a 1 — na kolumnach.
<code>skipna</code>	Wykluczanie brakujących wartości; domyślnie włączone (True).
<code>level</code>	Redukcja grupy o poziom w przypadku hierarchicznego indeksowania osi (wieloindeks).

Niektóre metody, takie jak `idxmin` i `idxmax`, zwracają pośrednie statystyki — takie jak indeksy, pod którymi znajdują się wartości minimalne i maksymalne:

```
In [235]: df.idxmax()
Out[235]:
one  b
two  d
dtype: object
```

Inne metody zwracają **akumulacje**:

```
In [236]: df.cumsum()
Out[236]:
```

	one	two
a	1.40	NaN
b	8.50	-4.5
c	NaN	NaN
d	9.25	-5.8

Istnieją również metody, które nie są reduktorami ani nie generują akumulacji. Przykładem takiej metody jest `describe`. Metoda ta zwraca za jednym zamachem wiele statystyk podsumowujących obiekt:

```
In [237]: df.describe()
Out[237]:
```

	one	two
count	3.000000	2.000000
mean	3.083333	-2.900000
std	3.493685	2.262742
min	0.750000	-4.500000
25%	1.075000	-3.700000
50%	1.400000	-2.900000
75%	4.250000	-2.100000
max	7.100000	-1.300000

W przypadku danych nienumerycznych metoda `describe` zwraca inne parametry statystyczne:

```
In [238]: obj = pd.Series(['a', 'a', 'b', 'c'] * 4)

In [239]: obj.describe()
Out[239]:
```

count	16
unique	3
top	a
freq	8
dtype:	object

W tabeli 5.8 znajdziesz pełną listę statystyk podsumowujących i związanych z nimi metod.

Tabela 5.8. Statystyki opisowe i podsumowujące

Metoda	Opis
count	Liczba wartości innych niż NA.
describe	Zwraca zestaw statystyk podsumowujących serię lub każdą kolumnę ramki danych.
min, max	Określa wartości minimalną i maksymalną.
argmin, argmax	Zwraca indeks (liczbę), pod którą znajduje się wartość minimalna lub maksymalna.
idxmin, idxmax	Zwraca etykietę indeksu, pod którą znajduje się wartość minimalna lub maksymalna.
quantile	Zwraca przykładowy kwantyl w zakresie od 0 do 1.
sum	Oblicza sumę wartości.
mean	Oblicza średnią wartości.
median	Określa medianę wartości (kwantyl 50%).
mad	Oblicza średnią odchylenia bezwzględnego od wartości średniej.
prod	Zwraca iloczyn wszystkich wartości.

Tabela 5.8. Statystyki opisowe i podsumowujące — ciąg dalszy

Metoda	Opis
var	Określa wariancję wartości próbki.
std	Oblicza odchylenie standardowe wartości próbki.
skew	Określa asymetrię (skośność) wartości próbki (trzeci moment).
kurt	Określa asymetrię (skośność) wartości próbki (czwarty moment).
cumsum	Oblicza sumę kumulacyjną wartości.
cummin, cummax	Oblicza kumulacyjne minimum i maksimum.
cumprod	Oblicza kumulacyjny iloczyn wartości.
diff	Oblicza pierwszą różnicę (metoda przydatna podczas pracy z szeregami czasowymi).
pct_change	Oblicza zmianę procentową.

Współczynnik korelacji i kowariancja

Niektóre statystyki podsumowujące, takie jak współczynnik korelacji i kowariancja, są obliczane na podstawie pary argumentów. Skorzystajmy z przykładów ramek danych zawierających dane na temat cen i nakładów aukcji pobranych z serwisu Yahoo! Finance za pomocą dodatkowego pakietu `pandas-datareader`. Jeżeli nie zainstalowałeś go wcześniej, to możesz to zrobić za pomocą polecenia `conda` lub `pip`.

```
conda install pandas-datareader
```

Korzystam z modułu `pandas-datareader` w celu pobrania danych na temat kilku podmiotów giełdowych:

```
import pandas_datareader.data as web
all_data = {ticker: web.get_data_yahoo(ticker)
            for ticker in ['AAPL', 'IBM', 'MSFT', 'GOOG']}
price = pd.DataFrame({ticker: data['Adj Close']
                     for ticker, data in all_data.items()})
volume = pd.DataFrame({ticker: data['Volume']
                      for ticker, data in all_data.items()})
```



Możliwe, że w chwili, gdy czytasz te słowa, serwis Yahoo! Finance już nie istnieje (spółka Yahoo! została przejęta w 2017 roku przez korporację Verizon). Opis aktualnych możliwości pakietu `pandas-datareader` znajdziesz w jego dokumentacji dostępnej w internecie.

Teraz obliczę zmiany procentowe cen. Jest to operacja wykonywana na danych szeregu czasowego, którą opiszę dokładniej w rozdziale 11.:

```
In [242]: returns = price.pct_change()
```

```
In [243]: returns.tail()
```

```
Out[243]:
```

	AAPL	GOOG	IBM	MSFT
Date				
2016-10-17	-0.000680	0.001837	0.002072	-0.003483
2016-10-18	-0.000681	0.019616	-0.026168	0.007690
2016-10-19	-0.002979	0.007846	0.003583	-0.002255
2016-10-20	-0.000512	-0.005652	0.001719	-0.004867
2016-10-21	-0.003930	0.003011	-0.012474	0.042096

Metoda `corr` serii określa współczynnik korelacji nakładających się, wyrównanych za pomocą indeksu wartości umieszczonych w dwóch obiektach typu `Series` (dotyczy to wartości typu innego niż `NA`). Metoda `cov` w podobny sposób oblicza kowariancję:

```
In [244]: returns['MSFT'].corr(returns['IBM'])
Out[244]: 0.49976361144151144

In [245]: returns['MSFT'].cov(returns['IBM'])
Out[245]: 8.8706554797035462e-05
```

`MSFT` jest poprawnym atrybutem Pythona, a więc możemy dokonać wyboru kolumny za pomocą bardziej zwartej składni:

```
In [246]: returns.MSFT.corr(returns.IBM)
Out[246]: 0.49976361144151144
```

Metody `corr` i `cov` ramek danych zwracają pełną macierz współczynników korelacji i kowariancji w formie ramki danych:

```
In [247]: returns.corr()
Out[247]:
```

	AAPL	GOOG	IBM	MSFT
AAPL	1.000000	0.407919	0.386817	0.389695
GOOG	0.407919	1.000000	0.405099	0.465919
IBM	0.386817	0.405099	1.000000	0.499764
MSFT	0.389695	0.465919	0.499764	1.000000

```
In [248]: returns.cov()
Out[248]:
```

	AAPL	GOOG	IBM	MSFT
AAPL	0.000277	0.000107	0.000078	0.000095
GOOG	0.000107	0.000251	0.000078	0.000108
IBM	0.000078	0.000078	0.000146	0.000089
MSFT	0.000095	0.000108	0.000089	0.000215

Metoda `corrwith` obiektu `DataFrame` umożliwia obliczenie korelacji pomiędzy parami elementów kolumn lub wierszy ramki danych i innym obiektem typu `Series` lub `DataFrame`. Przekazanie serii powoduje zwrócenie serii zawierającej współczynniki korelacji obliczone dla każdej kolumny:

```
In [249]: returns.corrwith(returns.IBM)
Out[249]:
```

AAPL	0.386817
GOOG	0.405099
IBM	1.000000
MSFT	0.499764

dtype: float64

Przekazanie ramki danych spowoduje zwrócenie współczynników korelacji pomiędzy kolumnami o takich samych nazwach. W poniższym przykładzie obliczam korelacje pomiędzy procentowymi zmianami ilości:

```
In [250]: returns.corrwith(volume)
Out[250]:
```

AAPL	-0.075565
GOOG	-0.007067
IBM	-0.204849
MSFT	-0.092950

dtype: float64

Przekazanie argumentu `axis='columns'` spowoduje wykonanie operacji wiersz po wierszu. Niezależnie od tego, czy korelacja jest obliczana dla wierszy, czy dla kolumn, przed jej wyznaczeniem punkty danych są wyrównywane na podstawie etykiet.

Unikalne wartości, ich liczba i przynależność

Kolejna grupa metod, którą chciałbym opisać, służy do uzyskiwania informacji dotyczących wartości znajdujących się w jednowymiarowych seriach. Działanie tych metod przedstawię na przykładzie następującej serii:

```
In [251]: obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
```

Na początek przyjrzyjmy się funkcji `unique`, która zwraca tablicę unikalnych wartości znajdujących się w obiekcie `Series`:

```
In [252]: uniques = obj.unique()
```

```
In [253]: uniques
```

```
Out[253]: array(['c', 'a', 'd', 'b'], dtype=object)
```

Unikalne wartości nie zawsze muszą być zwracane w posortowanej kolejności, ale w razie konieczności można je posortować później (`uniques.sort()`). Funkcja `value_counts` zwraca natomiast obiekt `Series` zawierający dane na temat częstotliwości występowania poszczególnych wartości:

```
In [254]: obj.value_counts()
```

```
Out[254]:
```

```
c    3
a    3
b    2
d    1
dtype: int64
```

Dla ułatwienia wartości znajdujące się w obiekcie `Series` są sortowane w kolejności malejącej. Metoda `value_counts` jest jednocześnie wysokopoziomową metodą biblioteki `pandas`, która może być użyta w kontekście dowolnej tablicy lub sekwencji:

```
In [255]: pd.value_counts(obj.values, sort=False)
```

```
Out[255]:
```

```
a    3
b    2
c    3
d    1
dtype: int64
```

Metoda `isin` wykonuje wektorową operację sprawdzania przynależności do zbioru. Można z niej korzystać podczas filtrowania zbioru danych w celu uzyskania podzbioru wartości w formie obiektu `Series` lub kolumny obiektu `DataFrame`:

```
In [256]: obj
```

```
Out[256]:
```

```
0    c
1    a
2    d
3    a
4    a
5    b
6    b
7    c
8    c
dtype: object
```

```
In [257]: mask = obj.isin(['b', 'c'])
```

```
In [258]: mask
```

```

Out[258]:
0    True
1    False
2    False
3    False
4    False
5     True
6     True
7     True
8     True
dtype: bool

In [259]: obj[mask]
Out[259]:
0    c
5    b
6    b
7    c
8    c
dtype: object

```

Metoda `Index.get_indexer` jest związana z metodą `isin`. Zwraca ona tablicę indeksów wygenerowaną na podstawie tablicy, w której mogą znajdować się identyczne wartości, umożliwiając utworzenie kolejnej tablicy z unikalnymi wartościami:

```

In [260]: to_match = pd.Series(['c', 'a', 'b', 'b', 'c', 'a'])

In [261]: unique_vals = pd.Series(['c', 'b', 'a'])

In [262]: pd.Index(unique_vals).get_indexer(to_match)
Out[262]: array([0, 2, 1, 1, 0, 2])

```

Informacje na temat tych metod podsumowano w tabeli 5.9.

Tabela 5.9. Metody umożliwiające określenie unikalnych wartości i liczby powtórzeń wartości oraz przynależności wartości do zbioru

Metoda	Opis
<code>isin</code>	Zwraca tablicę wartości logicznych określającą, czy każda z wartości serii znajduje się w przekazanej sekwencji wartości.
<code>match</code>	Określa wartości indeksów dla każdej wartości znajdującej się w tablicy. Dotyczy to indeksów, pod którymi znajdują się dane wartości w innej tablicy zawierającej unikalne wartości; przydaje się podczas wyrównywania danych i operacji łączenia.
<code>unique</code>	Zwraca tablicę unikalnych wartości wchodzących w skład serii; wartości są zwracane w kolejności, w jakiej zostały zaobserwowane.
<code>value_counts</code>	Zwraca serię, w której indeksem są unikalne wartości, a wartościami — liczby wystąpień poszczególnych wartości; dane są posortowane w kolejności malejącej.

Czasami zachodzi potrzeba obliczenia histogramu wielu spokrewnionych kolumn ramki danych. Oto przykład takiej operacji:

```

In [263]: data = pd.DataFrame({'Qu1': [1, 3, 4, 3, 4],
.....:                       'Qu2': [2, 3, 1, 2, 3],
.....:                       'Qu3': [1, 5, 2, 4, 4]})

In [264]: data
Out[264]:
   Qu1  Qu2  Qu3
0     1     2     1
1     3     3     5
2     4     1     2
3     3     2     4
4     4     3     4

```


0	1	2	1
1	3	3	5
2	4	1	2
3	3	2	4
4	4	3	4

Przekazanie `pandas.value_counts` do metody `apply` tej ramki danych da następujący efekt:

```
In [265]: result = data.apply(pd.value_counts).fillna(0)
```

```
In [266]: result
```

```
Out[266]:
```

	Qu1	Qu2	Qu3
1	1.0	1.0	1.0
2	0.0	2.0	1.0
3	2.0	2.0	0.0
4	2.0	0.0	2.0
5	0.0	0.0	1.0

W zaprezentowanym przykładzie etykiety wierszy wygenerowanego obiektu są unikalnymi wartościami występującymi we wszystkich kolumnach. W poszczególnych kolumnach zwróconego obiektu znajdują się wartości określające liczbę wystąpień danej wartości.

5.4. Podsumowanie

W kolejnym rozdziale opiszę narzędzia biblioteki `pandas` służące do wczytywania (*ładowania*) zbiorów danych, a także zapisywania ich na dysku. W dalszej kolejności zajmę się narzędziami przeznaczonymi do oczyszczania, przetwarzania, analizy i graficznej interpretacji danych.

A

adres URL, 167
algebra liniowa, 20, 93, 120
algorytm
 PCA, 21
 sortowania stabilnego, 436
analiza
 koszykowa, 288, 289
 skupień, 21
 statystyczna, 19, 295, 334, 338, 340, 346
 szeregu czasowego, 22
 trendów, 394
 wariancji, 22, 366
argument, *Patrz*: funkcja argument
arkusz kalkulacyjny, 15
automagia, 41

B

baza
 danych
 MySQL, 239
 SQL, 187, 222
 Olsona, 317
biblioteka, 61
 Beautiful Soup, 179
 Blosc, 183
 h5py, 183, 184, 442
 html5lib, 179
 interfejsu użytkownika, 42
 IPython, 19, 20
 Jupyter, 20
 line_profiler, 457
 LLVM, 439
 lxml, 179

matplotlib, 19, 42, 245, 246, 379
 importowanie, 245
 schemat kolorów, 258, 259
modelująca, 357
Numarray, 94
Numba, 439, 440
Numeric, 94
NumPy, 18, 25, 93, 94, 415
 importowanie, 95
 indeksowanie tablic, 101, 102, 104, 105, 108, 423
 typy danych, 98, 99
pandas, 16, 18, 25, 100, 127, 148, 154, 217, 274, 301, 341
 wizualizacja danych, 259
Patsy, 360, 362
PyTables, 183, 184, 442
pytz, 317
requests, 186
scikit-learn, 16, 21, 357, 369
SciPy, 20
seaborn, 245, 259, 379
statsmodels, 21, 295, 357, 360, 366
TensorFlow, 370
blok try-except, 84
błąd, *Patrz też*: wyjątek
 TypeError, 85
 ValueError, 85
błądzenie losowe, 124, 125
broadcasting, *Patrz*: rozgłaszanie

C

Curry Haskell, 82
currying, 82

czas, 55, 168, 302, 453
specyfikator, 56
strefa czasowa, 317, 319, 320, 376, 378, 381
uniwersalny koordynowany, *Patrz*: czas UTC
UTC, 317, 320

D

dane

agregacja, 281, 285, 309
binarne, 182
brakujące, 189, 370
 filtrowanie, 191
 wypełnianie, 193, 290, 291
 zastępowanie, 197
dyskretyzacja, 200, 201
filtrowanie, 273
grupowanie, *Patrz też*: ramka danych
 grupowanie
 nieobudowane, 352
homogeniczne, 415, 432
indeksowanie, *Patrz*: indeksowanie
kategoryczne, 269
kod kategorii, 342
kodowanie one hot, 348
łączenie, 222, 224, 226, 227, 230
 częściowo nakładających się, 234
modelowanie, 189, 362
nienumeryczne, 363
odczyt, 167, 172, 173
operacja osiowa, 235
podział na koszyki, 200, 201, 288, 289, 345, 411
 krawędź, 329, 331
 OHLC, 331
powtarzające się, 342
poziom, 342
próbka losowa, 292
przekształcanie, 196
reprezentacja
 kategoryczna, 342, 343, 347, 348, 363, 365
 kodowana słownikowo, 342
standaryzacja, 362
średnia ważona, 293
tabelaryczne przekształcenia, 235
typ, 50
ustrukturyzowane, 15, 18
wizualizacja, 19, 245, 272, 273
wyrównywanie, 148
wyśrodkowywanie, 362

zagnieżdżone, 433
zapis, 174
zduplikowane, 195
data, 55, 168, 169, 302, 304
dzień roboczy, 314
przesunięcie, 313, 314, 315
specyfikator, 56
zakres, 310, 311, 312, 313, 314
 z określeniem strefy czasowej, 318
debuger, 449, 450, 451, 452
design matrix, *Patrz*: macierz projektowa
dummy variable, *Patrz*: zmienna fikcyjna

E

estymator jądrowy gęstości, *Patrz*: wykres gęstości

F

facet grid, *Patrz*: wykres panelowy
fancy indexing, *Patrz*: ndarray indeksowanie
 specjalne
feature engineering, *Patrz*: inżynieria cech
format
 bcloz, 183
 długi, 238, 239
 Feather, 183
 HDF5, 167, 182, 183, 185, 442
 hierarchiczny, 183
 HTML, 167, 179
 JSON, 167, 178, 375, 402
 MessagePack, 167, 182
 pickle, 182
 szeroki, 241
 XML, 179, 180
funkcja, *Patrz też*: metoda, polecenie
 %alias, 448
 %cpaste, 39, 41
 %debug, 41
 %hist, 41
 %magic, 41
 %matplotlib, 42
 %page, 42
 %paste, 39, 41
 %pdb, 41
 %prun, 42, 455
 %quickref, 41
 %reset, 41
 %run, 42

%run-p, 455
 %time, 42, 453, 454
 %timeit, 40, 42, 453, 454
 %who, 42
 %xdel, 42
 abs, 112
 add, 113, 440
 add_subplot, 247, 248
 anonimowa, *Patrz: funkcja lambda*
 apply, 153
 arange, 97
 arccos, 112
 arcsin, 112
 arctan, 112
 argmax, 125
 argpartition, 437
 argument
 definiowany za pomocą słowa kluczowego, 78
 kluczowy, 44
 pozycyjny, 44, 78
 array, 96, 97
 asarra, 97
 astype, 100
 beta, 123
 binomial, 123
 bisect.bisect, 66
 bisect.insort, 66
 calue_counts, 341
 ceil, 112
 chisquare, 123
 column_stack, 421
 conatatenate, 230
 concat, 222, 230, 231, 233
 concatenate, 420, 421
 copysign, 113
 cos, 112
 count, 281
 csv.writer, 177
 cut, 200, 202
 date_range, 310, 311, 312
 det, 122
 diag, 122
 divide, 113
 divmod, 111
 dot, 120, 122
 drop_duplicates, 195
 dropna, 190, 191
 dstack, 421
 duplicated, 195
 eig, 122
 empty, 97, 98
 empty_like, 98
 enumerate, 68
 equal, 113
 exp, 110, 112
 eye, 98
 fabs, 112
 factorplot, 269, 271
 figure, 246
 fillna, 190, 193, 194, 197, 290, 291
 first, 281
 float, 84
 floor, 112
 floor_divide, 113
 fmax, 113
 fmin, 113
 frompyfunc, 432
 full, 98
 full_like, 98
 gamma, 21, 123
 get_dummies, 204
 gęstości, 21
 greater, 113
 greater_equal, 113
 groupby, 83, 276, 277, 279, 345
 hstack, 420
 hsplit, 421
 hstack, 421
 identity, 98
 iloc, 203
 in1d, 119
 intersect1d, 119
 inv, 122
 isfinite, 112
 isinf, 112
 isinstance, 46
 isnan, 112
 isnull, 130, 190
 iter, 47
 join, 222, 223
 json.dumps, 178
 json.loads, 178
 json.pandas.read_json, 178
 lambda, 81, 212, 284, 350
 last, 281
 less, 113
 less_equal, 113
 list, 49, 64
 funkcja

load, 120
log, 112
logical_and, 113, 430
logical_not, 112
logical_or, 113
logical_xor, 113
lstsq, 122
magiczna, 39, 40, 41, 447
map, 196, 199
max, 281
maximum, 111, 113
mean, 281, 283
median, 281
melt, 241
merge, 222, 223, 226
min, 281
minimum, 113
mod, 113
model.predict, 372
modf, 111, 112
multiply, 113
normal, 123
not_equal, 113
notnull, 130, 190
ones, 97
ones_like, 97
open, 87
optymalizator, 20
pandas.read_html, 179
pandas.read_pickle, 182
parser, 304
parsowania, 167, 168, 171, 181
partition, 437
patsy.build_design_matrices, 363
PeriodIndex, 239
permutation, 123, 203
pinv, 122
pivot, 241
pivot_table, 295, 297
power, 113
prod, 281
profilowanie, 457
qcut, 201, 202, 345
qr, 122
quantile, 281
rand, 123
randint, 123
randn, 123
random.seed, 123
range, 59, 97
re.compile, 210
read_clipboard, 167
read_csv, 167, 168, 169
 argument, 172, 173
read_excel, 167
read_feather, 168
read_fwf, 167
read_hdf, 167
read_html, 167
read_json, 167
read_msgpack, 167
read_pickle, 167
read_sas, 168
read_sql, 168
read_stata, 168
read_table, 167, 169, 170, 176
 argument, 172, 173
reindex, 157
rename, 199
repeat, 422
replace, 197, 198
resample, 310
reset_index, 222
reversed, 70
rint, 112
row_stack, 421
rozkładu ciągłego, 21
ruchomego okna, 334, 335, 336, 340
 binarna, 338
sample, 203
save, 120
seaborn.barplot, 266
seed, 123
Series, 129
set, 73
set_index, 221
set_trace, 452
setdiff1d, 119
setxor1d, 119
shuffle, 123
sign, 112
sin, 112
solve, 122
sorted, 69
spli, 421
sqrt, 110, 112
square, 112
std, 281, 283

- str, 52, 213, 303
- str.get, 213
- strptime, 56, 303
- subplots, 248
- subtract, 113
- sum, 281
- svd, 122
- tan, 112
- tile, 422, 423
- trace, 122
- tuple, 61
- tworzenie, 77
 - częściowa aplikacja argumentów, 82
- uniform, 123
- union1d, 119
- unique, 119, 341
- uniwersalna, 110, 153, 429
 - dwuargumentowa, 111, 113
 - jednoargumentowa, 111, 112
- value_counts, 381
- var, 281
- vectorize, 440
- vsplit, 421
- vstack, 420, 421
- wartość, 79
- where, 115
- wywołanie, 44
- zeros, 97, 98
- zeros_like, 98
- zip, 69

G

- generator, 82, 83
- GIL, 17
- Global Interpreter Lock, *Patrz*: GIL
- GroupBy, *Patrz*: obiekt grupujący

H

- haszowalność, 72
- histogram, 266, 268
- historia poleceń, 445
- HTML, 20
- Huginin Jim, 94
- Hunter John, 19, 245

I

- indeks, 137, 138, 219, 435
 - hierarchiczny, 170, 288
 - oparty na delcie czasu, 301
 - uaktualnianie, 139, 140, 141
 - w postaci liczb całkowitych, 147
- indeksowanie
 - częściowe, 218
 - hierarchiczne, 217, 218, 236
- instrukcja
 - break, 58
 - except, 84, 85
 - if, 57, 60
 - pass, 59
- interfejs
 - groupby, 273
 - oparty na formułach, 366, 368
 - pakietu matplotlib, 252
 - pyplot, 252
 - sieciowy, 186
 - tablicowy, 366
 - typu, 168
- inżynieria cech, 357
- IPython, 31
- iterator, 82

J

- jądro, 20
- język
 - C, 16
 - C#, 17
 - C++, 16, 17
 - FORTRAN, 16
 - interpretowany, 16, 17, 30
 - Java, 17
 - kompilowany, 17
 - o ścisłej kontroli typów, 46
 - Perl, 16
 - Python, *Patrz*: Python
 - R, 17, 19
 - Ruby, 16
 - SAS, 17
 - skryptowy, 16
- Jupyter Notebook, 20, 25
 - generowanie wykresów, 245
 - plik, 33
 - uruchamianie, 32
 - wykres, 247

K

- kalendarz, 302
- kategoria, 342
- KDE, *Patrz*: wykres gęstości
- kernel density estimate, *Patrz*: wykres gęstości
- klasa
 - defaultdict, 72
 - ExcelFile, 185
 - HDFStore, 183, 184
 - Period, 321
 - PeriodIndex, 321
 - WOM, 314
- kod
 - HTML, *Patrz*: HTML
 - Markdown, 20
 - struktura, 43
 - ze schowka, 39
- komentarz, 44
- kontener, 18
- kowariancja, 161
- krotka, 50, 61, 79
 - element, 62
 - mnożenie, 62
 - przypisywanie zmiennych, 62
 - rozpakowanie, 62, 63
 - tworzenie, 61
 - wycinek, *Patrz*: wycinek
 - zagnieżdżona, 61, 62
- kwartył, 201

L

- las losowy, 21
- liczba
 - całkowita, 51
 - losowa, 93, 123
 - pseudolosowa, 122, 123
 - zmiennoprzecinkowa, 51
- linia regresji, 269
- lista, 49, 50, 64
 - element
 - dodawanie, 64
 - filtrowanie, 75
 - łączenie, 65
 - w formie słownika, 71
 - składanie, 75, 76
 - zagnieżdżone, 76

- sortowanie, 66, 69
- wycinek, *Patrz*: wycinek
- literal łańcuchowy, *Patrz*: łańcuch

Ł

- łańcuch, 50, 51
 - dodawanie, 53
 - formatowanie, 53
 - JSON, 167, 178, 186, 375
 - łączenie, 207
 - podział, 207
 - sparator, 207
 - szablon, 53
 - wektoryzacja, 212
 - wieloliniowy, 51
- łata, 256

M

- macierz, 15
 - faktoryzacja, 21
 - iloczyn skalarny, 120, 121
 - mnożenie, 120
 - projektowa, 360
 - rozkład, 20, 120
 - rzadka, 21
 - tożsamościowa, 98
 - transpozycja, 109
 - wykresów punktowych, 269
- mapa, *Patrz*: słownik
- memory-mapped,
 - Patrz*: plik mapowany w pamięci
- metadane, 98, 137
- metoda, 44, *Patrz też*: funkcja
 - accumulate, 430, 431
 - add_categories, 349
 - agg, 282, 285, 350
 - aggregate, 282, 283
 - all, 118
 - any, 118
 - append, 64, 65, 138
 - apply, 286, 287, 340, 349, 351
 - argmax, 117
 - argmin, 117
 - argsort, 435
 - as_ordered, 349
 - as_unordered, 349
 - asfreq, 322

astype, 99, 100
 Bayesa, 366
 caseload, 209
 cat, 214
 center, 214
 close, 88, 89
 closed, 89
 contains, 212, 214
 conv, 162
 corr, 162
 corrwith, 162
 count, 51, 64, 208, 214
 cumprod, 117
 cumsum, 117
 datetime, 56
 delete, 138
 describe, 282
 difference, 138
 distplot, 268
 drop, 138, 141
 encode, 54
 ends with, 208, 214
 extend, 65
 extract, 214
 fillna, 332
 find, 208
 findall, 210, 211, 212, 214
 finder, 212
 fit, 367
 flush, 89, 441
 format, 53
 get, 72, 214
 grid search, 21
 groupby, 288, 295, 296, 335, 349
 groups, 211
 head, 132
 heapsort, 437
 index, 208
 Index.get_indexer, 164
 insert, 64, 138
 insertion, 138
 intersection, 73
 is_monotonic, 138
 is_unique, 138
 isalnum, 214
 isalpha, 214
 isdecimal, 214
 isdigit, 214
 isin, 138, 163, 164
 islower, 214
 isnumeric, 214
 isupper, 214
 join, 208, 214
 json, 186
 keys, 71
 legend, 252
 len, 214
 lexsort, 435, 436
 ljust, 209
 lower, 209, 214
 lstrip, 209, 214
 łączenie w łańcuch, 354, 355
 magiczna, 35
 match, 164, 210, 211, 212, 214
 max, 117
 mean, 117, 275
 mergesort, 437
 min, 117
 momentów, 366
 mro, 416
 numeryczna, 20
 outer, 430, 431
 pad, 214
 pipe, 355
 pivot, 239
 plot, 259, 260, 261
 plot.bar, 262
 plot.barh, 262
 plot.kde, 267
 plt.legend, 252
 pop, 65, 70, 72
 quicksort, 437
 ravel, 419
 read, 88, 89
 readlines, 89
 reduce, 429, 430, 431
 reduceat, 431
 redukcji, 116
 regplot, 269
 reindex, 139, 140, 141, 332
 remove, 65
 remove_categories, 349
 remove_unused_categories, 349
 rename_categories, 349
 reorder_categories, 349
 repeat, 214
 replace, 208, 214
 resample, 328, 329, 335, 353

- metoda
 - reshape, 417, 418
 - rfind, 208
 - rjust, 209
 - rstrip, 209, 214
 - sample, 292
 - savefig, 257
 - search, 210, 212
 - searchsorted, 438
 - seek, 88, 89, 90
 - set_categories, 349
 - set_xticklabels, 253
 - set_xticks, 253
 - setdefault, 72
 - shift, 314, 315
 - size, 276
 - slice, 214
 - sort, 66, 118, 434
 - klucz sortowania, 66
 - sort_index, 154, 220
 - sort_values, 436
 - split, 207, 209, 212, 214
 - stack, 218, 219, 236, 237
 - startswith, 208, 214
 - statystyczna, 116, 158
 - std, 117
 - strftime, 56, 303
 - strip, 207, 209, 214
 - sub, 211, 212
 - subn, 212
 - subplots_adjust, 250
 - sum, 117
 - summary, 367
 - swapaxes, 110
 - swaplevel, 220
 - take, 342, 382
 - tell, 88, 89
 - to_datetime, 304
 - to_pickle, 182
 - transform, 349, 351
 - transpose, 109
 - tz_convert, 318, 319
 - tz_localize, 318, 319
 - ukrywanie, 35
 - union, 73, 138
 - unique, 138, 164
 - unstack, 218, 219, 236, 237, 238, 241, 381
 - update, 71
 - upper, 209, 214
 - value_counts, 163, 164
 - values, 71
 - var, 117
 - wewnętrzna, 35
 - write, 89
 - writelines, 89
 - xlim, 252, 253
 - xticklabels, 252
 - xticks, 252
- model
 - dostrajanie, 372
 - liniowy, 22, 360, 366
 - obiektowy, 44
 - przestrzeni stanu, 366
 - przeszukiwanie siatkowe, 372
 - przeuczenie, 372
 - regresji liniowej, *Patrz:* model liniowy
 - walidacja krzyżowa, 372
- moduł, 48
 - bisect, 66
 - calendar, 302
 - collections, 72
 - cProfile, 455
 - csv, 176
 - datetime, 55, 302
 - itertools, 83, 84
 - lxml.objectify, 181
 - numpy.linalg, 121
 - numpy.random, 122
 - pickle, 182
 - re, 209
 - time, 302
- MovieLens, 384

N

- NaN, 132
- ndarray, 95, 96, 97, 136, 415, 432
 - indeksowanie, 101, 102, 104, 105, 423
 - specjalne, 108
- krotka
 - kroków, 415
 - kształtu, 415, 418
- kształt, 415, 418
- sortowanie, 118, 119, 434, 435
- transpozycja, 109
- tworzenie, 96, 97, 98
- typ danych, 415
- wskaźnik danych, 415
- wycinek, 102

notacja
?, 37
data.map, 212
NumPy, 358

O

obiekt
 atrybut, 47
 BytesIO, 257
 c_, 421
 data.frame, 19
 DataFrame, 131, 132, 139, 151, 183, 191, 284,
 Patrz też: ramka danych
 indeksowanie, 146
 kolumna, 132, 133, 134
 konwersja na tablicę NumPy, 358
 ranking, 156
 sortowanie, 154
 statystyki, 158
 wiersz, 133
 DateOffset, 320
 datetime, 56, 302, 303, 304, 315
 dostęp oparty na atrybucie, 133
 dtype, 96, 98
 figure, 246
 grupujący, 275, 279, 281, 286, 287
 indeksowanie, 278
 iteracja, 277
 haszowalność, *Patrz:* haszowalność
 index, 137, 199
 introspekcja, 36
 iterowalny, 47, 82
 tworzenie, 82
 kroczący, 415
 memmap, 441
 metoda, 47, *Patrz też:* metoda
 modyfikacja, 50, 137
 efekt uboczny, 50
 ndarray, 18, *Patrz:* ndarray
 odwołanie, 46
 pandas.Categorical, 344
 Patsy, 360, 361
 Period, 321, 322
 PeriodIndex, 322
 tworzenie, 326
 Python pickle, 167
 r_, 421
 Series, 139, 151, 183, 279, 347
 permutacja, 203

 ranking, 156
 sortowanie, 154
 statystyki, 158
 shape, 96
 string, 207, 208
 tablicowy, 18, 95
 timedelta, 302
 Timestamp, 303, 306, 315, 319
 typ, 44, *Patrz:* typ
 wywoływalny, 355
okno ruchome, 334, 335, 336
Oliphant Travis, 94
operator
 &, 73
 |, 73
 binarny, 48, 49
 ewm, 337
 iloc, 145, 147
 indeksowania, 145
 infiksowy mnożenia macierzy, 121
 loc, 133, 145, 147
 matematyczny, 48
 porównywania, 48
 rolling, 335

P

pakiet, *Patrz:* biblioteka
patch, *Patrz:* łata
Pérez Fernando, 19, 20
pętla
 for, 58
 while, 59
pierwiastek, 20
pivot table, *Patrz:* tabela przestawna
pivoting, *Patrz:* dane operacja osiowa
plik
 .py, 48
 binarny, 89, 167, 168, 182, 183, 185
 CSV, 167, 174, 177, 238, 406
 składnia, 176, 177
 Feather, 168, 183
 HDF5, 167, 182, 183, 185
 HTML, 167
 iteracja, 173
 mapowany w pamięci, 441
 otwieranie, 87
 do odczytu, 87, 88
 do zapisu, 87

- plik
 - parsowanie, 181
 - PDF, 257
 - programu Stata, 168
 - SAS, 168
 - SVG, 257
 - tryb tekstowy, 89
 - XLS, 167, 185
 - XML, 180
 - zamykanie, 87, 88
- polecenie, *Patrz też:* funkcja
 - %lprun, 457, 458
 - %pdb, 450
 - %prun, 457
 - %run, 31, 37, 450
 - cat, 169
 - exit, 30
 - ipython, 31
 - powłoki systemowej, 447
 - python, 30
 - systemowe, 447
 - type, 169
- powłoka IPython, *Patrz:* IPython
- protokół iteratora, 47, 82
- przepływ sterowania, 57
- przerwanie KeyboardInterrupt, 38
- przestrzeń nazw, 78, 459
 - pusta, 38
- przetwarzanie
 - sygnałów, 20
 - wsadowe, 95
- Python, 17
 - instalacja, 22, 24
 - interpreter, *Patrz:* interpreter
 - jądro, *Patrz:* jądro
 - konfiguracja, 22, 24
 - ograniczenia, 17
 - semantyka, 43
 - społeczność, 25
 - wydajność, 442
- poziom indeksu, 280
- seria, 279
- słownik, 279
 - tabela przestawna, *Patrz:* tabela przestawna
- indeksowanie, 146, 220, 221
- łączenie, 222, 223, 224, 226, 234
 - przy użyciu indeksu, 227, 230
- permutacja, 203
- ranking, 156
- sortowanie, 155
- statystyki, 158
- typ danych, 136
- regresja
 - grzbietowa, 21
 - Lasso, 21
 - liniowa, 22
 - logistyczna, 21
 - OLS, 295
- resampling, 273
- rozgłaszanie, 93, 113, 151, 424, 425, 426, 428
- rozkład
 - beta, 123
 - chi-kwadrat, 123
 - gamma, 123
 - jednostajny, 123
 - normalny, 116, 123
 - prawdopodobieństwa, 21
- równanie
 - liniowe rzadkie, 21
 - różniczkowe, 20
- rzutowanie, 46

S

R

- ramka danych, 131, 145, 168, 274,
 - Patrz też:* DataFrame
- grupowanie, 274
 - funkcja, 280
 - klucz, 274, 275, 276, 277, 280, 288
 - obiekt grupujący, 275

- sekwencja, 68
- seria, 128, 274, 347
 - etykieta, 128
- indeks, 128
- indeksowanie, 143, 144, 220
- ranking, 156
- sortowanie, 155
- statystyki, 158
- tworzenie, 128
- serializacja, 182
- set, *Patrz:* zbiór
- skrót klawiszowy, 39
 - Ctrl+C, 38, 40
 - Ctrl+D, 30
- skrypt, 16

słownik, 50, 70, 129, 136, 196, 198, 279, 342
 element, 70
 klucz, 72
 łączenie, 71
 składanie, 76
 słowników, 136
słowo kluczowe
 and, 54
 as, 48
 continue, 58
 def, 77
 del, 70
 else, 85
 except, 85
 finally, 85
 for, 58
 global, 78, 79
 if, 57, 60
 in, 65
 is, 49
 is not, 49
 lambda, 81
 map, 196, 197
 not, 65
 or, 54
 while, 59
SPECFUN, 21
statystyka opisowa, 21
stos, *Patrz:* format długi
structured array, *Patrz:* tablica o złożonej strukturze
SVM, 21
symulacja Monte Carlo, 292
system kodowania
 ASCII, 51, 53
 konwersja, 54
 Unicode, 51, 53, 90
 UTF-8, 30, 51, 54, 90
szereg
 czasowy, 15, 22, 301, 305, 333, 366, 369, 394
 częstotliwość, 310, 311, 312, 313, 314, 322,
 323, 326, 328, 329, 332, 333
 eksperymentu, 301
 indeksowanie, 306, 309
 interwał, 301
 okres, 301, 321, 325
 próbkiowanie
 w dół, 328, 329
 w górę, 328, 332

 strefa czasowa, *Patrz:* czas strefa czasowa
 wagi zanikające wykładniczo, 334, 337
 wybieranie, 306, 308
 z duplikatami indeksów, 309
 zmiana rozdzielczości, 310
 znacznik czasu, 301, 325

Ś

środowisko programistyczne, 25, 461

T

tabela, 18
 krzyżowa, 298
 przestawna, 295, 296, 297
tablica, 18, 94
 asocjacyjna, *Patrz:* słownik
 element
 wartość początkowa, 97, 98
 widok, 101
 indeksowanie, 101, 102, 104, 105, 108, 423
 konkatenacja, 230
 kształt, 96
 łączenie, 420
 ndarray, *Patrz:* ndarray
 NumPy, 18, 50
 o złożonej strukturze, 432, 434
 powtarzanie, 422
 replikacja, 422
 spłaszczanie, 419
 transpozycja, 109
 tworzenie, 96, 97, 98
 typ danych, 96, 98
 wartości całkowitoliczbowych, 423
 wartości logicznych, 105, 106, 107
 wielowymiarowa, *Patrz:* macierz
 wymiaru, 342
 zmiana wymiarów, 417
Taylor Jonathan, 21
transformacja Fouriera, 93
tuple, *Patrz:* krotka
typ, 98, 99
 bool, 51, 55
 bytes, 51
 Categorical, 341, 343, 344, 345
 date, 55
 datetime, 55, 302, 303
 datetime.timedelta, 57

typ

- datetime64, 306
- float, 50, 51, 55
- hierarchia, 417
- int, 50, 51, 55
- interfejs, *Patrz*: interfejs typu
- kontrola ścisła, 46
- konwersja, 46
- None, 51
- rozdzielanie, 52
- rzutowanie, 55, 99, 100
- skalarny, 50
- str, 51, 55
- string, 53
- time, 55
- zagnieżdżony, 433

typowanie kaczki, 47

U

uczenie maszynowe, 348, 357, 369, 384

- nadzorowane, 369
- nienadzorowane, 369

układ rzadkich równań liniowych, 21

W

walidacja krzyżowa, *Patrz*: model walidacja krzyżowa

wartość

- False, 54
- logiczna, 54
- NA, 190, 191, 290
- NaN, 189, *Patrz*: NaN
- NaT, 304
- niedostępna, *Patrz*: wartość NA
- None, 50, 55
- null, 51
- OHLC, 331
- True, 54
- zastępcza, 189

Wattenberg Laura, 398

wątek nakładanie się w czasie, 17

wektoryzacja, 100, 113

wiązanie, 45

Wickham Hadley, 183, 274, 389

wielowątkowość, 17, *Patrz też*: GIL

wiersz poleceń, 30

Williams Ashley, 402

współczynnik

- korelacji, 161, 294
- zaniku, 337

wycinek, 67

wydajność, 442

wyjątek, 84, *Patrz też*: błąd

- ValueError, 100
- w konsoli IPython, 86

wykres, 19, 245

- adnotacje, 255, 256
- gęstości, 266, 268
- kolor, 250, 258
- kształty, 256
- legenda, 252, 253, 254
- liniowy, 259
- marginesy, 250, 257
- panelowy, 269, 271
- par, 269
- pudełkowy, 271
- punktowy, 268, 269
- słupkowy, 262, 263, 379
- histogram, *Patrz*: histogram
- styl linii, 250
- tytuł, 253
- zapisywanie, 257
- znacznik, 251, 253

wyrażenie

- generatora, 83
- regex, 209
- regularne, 170, 209, 210
- trójargumentowe, 60

Y

Yahoo! Finance, 161

Z

zapytanie SQL, 168

zbiór, 73, 75

- część wspólna, 73
- domknięcie, 200
- literal, 73
- otwarcie, 200
- różnica, 73
- składanie, 76
- suma, 73
- tworzenie, 73

- zmienna
 - fikcyjna, 348
 - globalna, 78
 - lokalna, 78
 - przypisywanie, 45
 - wiązana, 45
 - wskazująca, 205, 206
 - zakres, 78
 - zastępcza, 204
- znak
 - `#`, *Patrz*: znak kratki
 - `%`, 40
 - `@`, 121
 - `>>>`, *Patrz*: znak gotowości
 - biały, 43
 - usuwanie, 207
 - cudzysłowu
 - podwójny, 51
 - pojedynczy, 51
 - cudzysłowu, 51
 - dwukropka, 43
 - gotowości, 30
 - gwiazdki, 37
 - końca wiersza, 51
 - kratki, 44
 - łańcuch, *Patrz*: łańcuch
 - spacji, 43
 - specjalny, 52
 - system kodowania, *Patrz*: system kodowania
 - średnika, 44
 - tabulacji, 43
 - ukośnika lewego, 52
 - Unicode, 52, 53
 - zapytania, 36

Ż

żądanie HTTP, 178

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

Python: poznaj idealne narzędzie do analizy danych!

Analiza danych stała się samodzielną dyscypliną wiedzy, interesującą specjalistów z wielu branż: analityków biznesowych, statystyków, architektów oprogramowania czy też osoby zajmujące się sztuczną inteligencją. Wydobywanie informacji ze zbiorów danych pozwala na uzyskanie wiedzy niedostępnej w inny sposób. W tym celu dane trzeba odpowiednio przygotować, oczyścić, przetworzyć i oczywiście poddać analizie. Warto również zadbać o ich wizualizację. Do tych wszystkich zadań najlepiej wykorzystać specjalne narzędzia opracowane w języku Python.

To drugie, zaktualizowane i uzupełnione wydanie klasycznego podręcznika napisanego z myślą o analitykach, którzy dotychczas nie pracowali w Pythonie, oraz o programistach Pythona, którzy nie zajmowali się dotąd analizą danych ani obliczeniami naukowymi. Przedstawiono tu możliwości oferowane przez Pythona 3.6 oraz najnowsze funkcje pakietów Pandas i NumPy, a także środowisk IPython i Jupyter. Przy opisie poszczególnych narzędzi analitycznych wyjaśniono ich działanie i zaprezentowano przykłady ich efektywnego i kreatywnego wykorzystania. Ta książka powinna się znaleźć w podręcznej bibliotece każdego analityka danych!

Wes McKinney — jest świetnie znany jako twórca pakietu Pandas, popularnej otwartej biblioteki Pythona przeznaczonej do analizy danych. Zajmuje się językami Python i C++. Jest związany ze środowiskiem analityków pracujących w Pythonie i z Apache Software Foundation, z którą rozwija wiele ciekawych projektów. Obecnie pracuje w Nowym Jorku jako architekt oprogramowania. Często występuje w roli prelegenta na konferencjach. Uwielbia podróże, interesuje się lingwistyką i językami obcymi.

Najważniejsze zagadnienia:

- eksploracja danych za pomocą powłoki IPython i środowiska Jupyter
- korzystanie z pakietów NumPy i Pandas
- tworzenie wizualizacji danych za pomocą pakietu Matplotlib
- praca z danymi regularnych i nieregularnych szeregów czasowych
- rozwiązywanie rzeczywistych problemów analitycznych

  helion.pl  HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	<i>Sprawdź nasze szkolenia!</i>  SZKOLENIA AKADEMIA IT & BUSINESS WWW.SZKOLENIA.HELION.PL	KOD KORZYŚCI <i>Sięgnij po więcej!</i>   ISBN 978-83-283-4081-7  9 788328 340817 INFORMATYKA W NAJLEPSZYM WYDANIU Cena: 89,00 zł
---	---	--