

O'REILLY®

Wydanie II

Python Data Science

Niezbędne narzędzia do pracy z danymi



Helion 

Jake VanderPlas

Tytuł oryginału: Python Data Science Handbook: Essential Tools for Working with Data, 2nd Edition

Tłumaczenie: Filip Kamiński

ISBN: 978-83-289-0068-4

© 2023 Helion S.A.

Authorized Polish translation of the English edition of *Python Data Science Handbook, 2E*
ISBN 9781098121228 © 2023 Jake VanderPlas

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 wydawca 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 wydawca nie ponoszą również
żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: helion@helion.pl

WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/pydasc>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

Wprowadzenie.....	17
-------------------	----

Część I. Jupyter — coś więcej niż zwykły Python 23

1. Wprowadzenie do IPython oraz Jupytera 25

Uruchamianie powłoki IPython	25
Uruchamianie Jupyter Notebook	25
IPython — pomoc i dokumentacja	26
Dostęp do dokumentacji za pomocą ?	27
Dostęp do kodu źródłowego za pomocą ??	28
Przeglądanie zawartości modułów za pomocą autouzupełniania z tabulatorem	29
Skróty klawiaturowe w powłoce IPython	31
Skróty do nawigacji	31
Skróty do wprowadzania tekstu	31
Skróty związane z historią poleceń	32
Pozostałe skróty	33

2. Funkcje interaktywne 34

Magiczne polecenia IPython	34
Uruchamianie zewnętrznego kodu za pomocą %run	34
Pomiar czasu wykonania za pomocą %timeit	35
Pomoc dotycząca magicznych poleceń ?, %magic i %lsmagic	35
Historia wejścia i wyjścia	36
Obiekty In i Out IPython	36
Symbol podkreślenia i poprzednie wyjścia	37
Wyłączanie wyjścia	38
Inne magiczne polecenia	38
Polecenia IPython i powłoki	38
Krótkie wprowadzenie do powłoki	39
Polecenia powłoki w IPythonie	40
Przekazywanie wartości do i z powłoki	40
Magiczne polecenia związane z powłoką	41

3. Debugowanie i profilowanie	43
Błędy i debugowanie	43
Kontrolowanie wyjątków za pomocą %xmode	43
Debugowanie — gdy lektura śladu nie wystarcza	45
Profilowanie kodu i pomiary czasu jego wykonania	47
Pomiar czasu wykonania fragmentu kodu za pomocą %timeit i %time	48
Profilowanie całych skryptów za pomocą %prun	49
Profilowanie linia po linii za pomocą %lprun	50
Profilowanie pamięci za pomocą %memit i %mprun	51
Więcej materiałów na temat IPython	52
Materiały dostępne w sieci	52
Książki	53

Część II. Wprowadzenie do NumPy **55**

4. Zrozumieć typy danych w Pythonie.....	57
Typ całkowitoliczbowy w Pythonie to coś więcej niż zwykły int	58
Lista w Pythonie to coś więcej niż zwykła lista	59
Tablice o stałym typie w Pythonie	60
Tworzenie tablic z list	61
Tworzenie tablic od podstaw	61
Standardowe typy danych NumPy	63
5. Podstawy pracy z tablicami NumPy	64
Atrybuty tablicy NumPy	64
Indeksowanie tablicy — dostęp do pojedynczych elementów	65
Slicing, czyli sposób na dostęp do podtablic	66
Jednowymiarowe podtablice	66
Wielowymiarowe podtablice	67
Podtablice jako widoki bez kopiowania	68
Kopiowanie tablic	68
Zmiana kształtu tablic	69
Konkatenacja i dzielenie tablic	69
Konkatenacja tablic	69
Dzielenie tablic	70
6. Obliczenia z użyciem tablic NumPy — funkcje uniwersalne	72
Powolność pętli	72
Wprowadzenie do funkcji uniwersalnych	73
Przegląd funkcji uniwersalnych dostępnych w NumPy	74
Arytmetyka tablicowa	74
Wartość bezwzględna	75

Funkcje trygonometryczne	76
Potęgi i logarytmy	76
Funkcje uniwersalne do zastosowań specjalnych	77
Zaawansowane możliwości funkcji uniwersalnych	78
Określanie miejsca zapisu danych wyjściowych	78
Agregacje	78
Metoda outer	79
Więcej materiałów na temat funkcji uniwersalnych	79
7. Agregacje — minimum, maksimum i wszystko pomiędzy nimi.....	80
Sumowanie wartości w tablicy	80
Minimum i maksimum	81
Agregacja w wielu wymiarach	81
Inne funkcje agregujące	82
Przykład: jaki jest średni wzrost prezydenta USA?	83
8. Obliczenia na tablicach — broadcasting	85
Co to jest broadcasting?	85
Zasady broadcastingu	87
Pierwszy przykład	87
Drugi przykład	88
Trzeci przykład	88
Broadcasting w praktyce	89
Centrowanie wartości w tablicy	89
Rysowanie wykresów funkcji dwuwymiarowych	90
9. Porównania, maski i logika boolowska	92
Przykład: sprawdzanie, przez ile dni padało	92
Operatory porównania jako funkcje uniwersalne	93
Praca z tablicami wartości logicznych	95
Zliczanie wpisów	95
Operatory logiczne	96
Tablice wartości logicznych jako maski	97
Słowa kluczowe and i or kontra operatory &/	98
10. Fancy indexing	100
Jak działa fancy indexing?	100
Łączenie różnych metod indeksowania	101
Przykład: wybieranie losowych punktów	102
Modyfikowanie wartości za pomocą fancy indexingu	104
Przykład: podział danych na kubelki	105

11. Sortowanie tablic.....	107
Szybkie sortowanie w NumPy — np.sort i np.argsort	108
Sortowanie wzdłuż wierszy lub kolumn	108
Sortowanie częściowe — partycjonowanie	109
Przykład: metoda k najbliższych sąsiadów	109
12. Dane ustrukturyzowane — ustrukturyzowane tablice NumPy	113
Tworzenie ustrukturyzowanych tablic	114
Bardziej zaawansowane typy złożone	115
Tablice rekordów — ustrukturyzowane tablice z niespodzianką	116
W stronę Pandas	116

Część III. Przekształcanie danych za pomocą Pandas 117

13. Wprowadzenie do obiektów Pandas.....	119
Obiekt typu Series	119
Obiekty typu Series jako uogólnienie tablic NumPy	120
Obiekt typu Series jako szczególny rodzaj słownika	121
Tworzenie obiektów typu Series	121
Obiekt typu DataFrame	122
Ramka danych jako uogólnienie tablicy NumPy	122
Ramka danych jako szczególny rodzaj słownika	123
Tworzenie obiektów typu DataFrame	124
Obiekt typu Index	125
Indeks jako niemutowalna tablica	126
Indeks jako uporządkowany zbiór	126
14. Indeksowanie i wybieranie	127
Wybór danych z obiektów typu Series	127
Obiekt typu Series jako słownik	127
Obiekt typu Series jako jednowymiarowa tablica	128
Indeksatory: loc i iloc	129
Wybór danych z obiektów typu DataFrame	130
Obiekt typu DataFrame jako słownik	130
Obiekt typu DataFrame jako dwuwymiarowa tablica	131
Inne konwencje związane z indeksowaniem	133
15. Operacje na danych w Pandas	134
Funkcje uniwersalne — zachowanie indeksu	134
Funkcje uniwersalne — dopasowanie indeksu	135
Dopasowanie indeksu w obiektach typu Series	135
Dopasowanie indeksu w obiektach typu DataFrame	136
Funkcje uniwersalne — operacje pomiędzy ramkami danych a obiektami typu Series	137

16. Obsługa brakujących danych	139
Kompromisy w konwencjach dotyczących brakujących danych	139
Brakujące dane w Pandas	140
None jako rodzaj wartownika	141
NaN — brakujące dane liczbowe	141
NaN i None w Pandas	142
Nullowalne typy danych w Pandas	143
Praca z wartościami typu null	143
Wykrywanie wartości typu null	144
Usuwanie wartości typu null	144
Uzupełnianie braków	145
17. Indeksowanie hierarchiczne.....	147
Wielokrotnie indeksowane obiekty typu Series	147
Zły sposób	147
Lepszy sposób — MultiIndex z Pandas	148
MultiIndex jako dodatkowy wymiar	149
MultiIndex — metody tworzenia	150
Tworzenie indeksu hierarchicznego z użyciem jawnego konstruktora	151
Nazwy poziomów indeksu hierarchicznego	152
MultiIndex dla kolumn	152
MultiIndex — indeksowanie i slicing	153
Obiekty typu Series z wielokrotnymi indeksami	153
Obiekty typu DataFrame z wielokrotnymi indeksami	155
MultiIndex — zmiana kolejności	156
Posortowane i nieposortowane indeksy	156
Metody stack i unstack	157
Ustawianie i resetowanie indeksu	158
18. Łączenie zbiorów danych — concat i append	159
Przypomnienie: konkatencja tablic NumPy	160
Prosta konkatencja za pomocą pd.concat	160
Zduplikowane indeksy	161
Konkatencja za pomocą złączeń	162
Metoda append	163
19. Łączenie zbiorów danych — merge i join	165
Algebra relacji	165
Rodzaje złączeń	166
Złączenia jeden-do-jednego	166
Złączenia wiele-do-jednego	167
Złączenia wiele-do-wielu	167

Określanie klucza, na podstawie którego ma być wykonane złączenie	168
Słowo kluczowe on	168
Słowa kluczowe left_on i right_on	169
Słowa kluczowe left_index i right_index	169
Wykorzystanie arytmetyki zbiorów w złączeniach	171
Nakładające się nazwy kolumn — słowo kluczowe suffixes	172
Przykład: dane dotyczące stanów USA	173
20. Agregacja i grupowanie	177
Dane na temat planet	177
Prosta agregacja w Pandas	178
Grupowanie — podziel, zastosuj funkcję, połącz	180
Podziel, zastosuj funkcję, połącz	180
Obiekt GroupBy	182
Agregacja, filtrowanie, transformacja, wywoływanie funkcji	183
Określanie sposobu podziału	186
Przykład grupowania	187
21. Tabele przestawne.....	188
Dane na potrzeby przykładu	188
Ręczne tworzenie tabel przestawnych	189
Składnia tabel przestawnych	189
Wielopoziomowe tabele przestawne	190
Dodatkowe opcje tabel przestawnych	190
Przykład: dane dotyczące liczby urodzeń	191
22. Zwektoryzowane operacje na łańcuchach znaków.....	197
Wprowadzenie do pracy z łańcuchami znaków w Pandas	197
Metody pracujące na łańcuchach znaków w Pandas	198
Metody podobne do metod znanych z Pythona	198
Metody wykorzystujące wyrażenia regularne	199
Różne metody	200
Przykład: baza przepisów	202
Prosty system rekomendacji przepisów	204
Jak można rozwinąć ten projekt?	205
23. Praca z szeregami czasowymi.....	206
Daty i godziny w Pythonie	206
Daty i godziny w Pythonie — datetime i dateutil	207
Typowane tablice znaczników czasu — datetime64 z NumPy	207
Daty i godziny w Pandas — najlepsze elementy z obu światów	209
Szeregi czasowe w Pandas — indeksowanie według czasu	210
Struktury danych do przechowywania szeregów czasowych w Pandas	210

Regularne sekwencje dat — <code>pd.date_range</code>	211
Częstotliwości i przesunięcia	212
Ponowne próbkowanie, przesuwanie i okna	213
Ponowne próbkowanie i zmiana częstotliwości	214
Przesunięcia w czasie	217
Ruchome okna	218
Przykład: wizualizacja danych o liczbie rowerów w Seattle	219
Wizualizacja danych	220
Zagłębianie się w dane	222
24. Wysoka wydajność w Pandas — <code>eval</code> i <code>query</code>	226
Dlaczego warto zastosować <code>query</code> i <code>eval</code> — wyrażenia złożone	226
Wydajne operacje z użyciem <code>pandas.eval</code>	227
Operacje na kolumnach z użyciem <code>DataFrame.eval</code>	229
Przypisanie w <code>DataFrame.eval</code>	229
Zmienne lokalne w <code>DataFrame.eval</code>	230
Metoda <code>DataFrame.query</code>	230
Wydajność — kiedy warto korzystać z tych funkcji	231
Materiały dodatkowe	232

Część IV. Wizualizacja z użyciem Matplotlib **233**

25. Wskazówki dotyczące korzystania z Matplotlib	235
Importowanie Matplotlib	235
Ustawianie stylów	235
Czy trzeba używać <code>show()</code> ? Jak wyświetlić wygenerowane wykresy?	235
Rysowanie z poziomu skryptu	236
Rysowanie z poziomu IPython	236
Rysowanie z poziomu notatnika Jupytera	237
Zapisywanie rysunków do pliku	238
Dwa interfejsy w cenie jednego	239
26. Proste wykresy liniowe	242
Dostosowywanie wykresu — kolory i style linii	245
Dostosowywanie wykresu — granice osi	247
Etykietowanie wykresów	249
Pułapki Matplotlib	251
27. Proste wykresy punktowe	252
Tworzenie wykresów punktowych za pomocą <code>plt.plot</code>	252
Tworzenie wykresów punktowych za pomocą <code>plt.scatter</code>	254
<code>plot</code> a <code>scatter</code> — uwaga na temat wydajności	258

Wizualizacja niepewności	258
Słupki błędów	258
Błędy ciągłe	259
28. Wykresy gęstości i wykresy konturowe	262
Wizualizacja trójwymiarowych funkcji	262
Histogramy, kubelki i gęstości	267
Dwuwymiarowe histogramy i podział danych na kubelki	269
plt.hist2d — dwuwymiarowy histogram	269
plt.hexbin — podział na sześciokątne kubelki	270
Jądrowy estymator gęstości	271
29. Dostosowywanie legend	273
Wybór elementów do legendy	276
Legenda opisująca rozmiary punktów	277
Wiele legend	279
30. Dostosowywanie pasków kolorów	281
Dostosowywanie pasków kolorów	281
Wybór mapy kolorów	283
Granice kolorów i wartości spoza zakresu	285
Dyskretne paski kolorów	286
Przykład: odręcznie zapisane cyfry	287
31. Podwykresy	289
plt.axes — manualne tworzenie podwykresów	289
plt.subplot — proste siatki podwykresów	291
plt.subplots — cała siatka za jednym zamachem	292
plt.GridSpec — bardziej skomplikowane układy	294
32. Tekst i adnotacje.....	296
Przykład: wpływ świąt na liczbę urodzeń w Stanach Zjednoczonych	296
Transformacje i położenie tekstu	298
Strzałki i adnotacje	300
33. Dostosowywanie znaczników osi.....	304
Główne i dodatkowe podziały	304
Ukrywanie podziałek lub ich etykiet	306
Zmniejszenie lub zwiększenie liczby podziałek	307
Inne możliwości formatowania podziałek	309
Lokalizatory i formatery — podsumowanie	311

34. Dostosowywanie wykresów — konfiguracja i style	313
Ręczne dostosowywanie wykresów	313
Zmiana ustawień domyślnych — rcParams	315
Arkusze stylów	317
Styl domyślny	318
Styl FiveThirtyEight	318
Styl ggplot	319
Styl z książki Bayesian Methods for Hackers	319
Ciemne tło	320
Rysunki w skali szarości	320
Styl Seaborn	321
35. Wykresy w przestrzeni trójwymiarowej	322
Trójwymiarowe punkty i krzywe	323
Trójwymiarowe wykresy konturowe	324
Wykresy typu wireframe i wykresy powierzchniowe	325
Triangulacja powierzchni	327
Przykład: wizualizacja wstęgi Möbiusa	329
36. Wizualizacje z użyciem pakietu Seaborn	331
Przegląd możliwości pakietu Seaborn	332
Histogramy, jądrowy estymator gęstości i wykresy gęstości	332
Wykresy typu pairplot	334
Grupy histogramów	335
Wykresy typu catplot	336
Wspólne rozkłady prawdopodobieństwa	336
Wykresy słupkowe	338
Przykład: eksploracja danych na temat czasu ukończenia maratonu	340
Materiały dodatkowe	347
Inne biblioteki do wizualizacji danych w Pythonie	347

Część V. Uczenie maszynowe **349**

37. Czym jest uczenie maszynowe?	351
Rodzaje uczenia maszynowego	351
Przykłady problemów uczenia maszynowego	352
Klasyfikacja, czyli przewidywanie dyskretnych etykiet	352
Regresja, czyli przewidywanie ciągłych etykiet	354
Klasteryzacja, czyli ustalanie etykiet w oparciu o nieetykietowane dane	357
Redukcja wymiarowości — wnioskowanie o strukturze danych pozbawionych etykiet	359
Podsumowanie	361

38. Wprowadzenie do Scikit-Learn	362
Reprezentacja danych w Scikit-Learn	362
Macierz cech	363
Tablica wartości docelowych	363
API Estimator	365
Podstawy korzystania z API	366
Przykład uczenia nadzorowanego: prosta regresja liniowa	366
Przykład uczenia nadzorowanego: klasyfikacja irysów	370
Przykład uczenia nienadzorowanego: redukcja wymiarowości w zbiorze Iris	370
Przykład uczenia nienadzorowanego: klasteryzacja irysów	372
Zastosowanie: eksploracja zbioru odręcznie zapisanych cyfr	372
Wczytywanie i wizualizacja danych	374
Przykład uczenia nienadzorowanego: redukcja wymiarowości	375
Klasyfikacja cyfr	376
Podsumowanie	378
39. Hiperparametry i walidacja modelu.....	379
Walidacja modelu	379
Niewłaściwy sposób walidacji modelu	379
Właściwy sposób walidacji modelu, czyli podział danych na zbiór uczący i testowy	380
Walidacja modelu za pomocą walidacji krzyżowej	381
Wybór najlepszego modelu	383
Kompromis pomiędzy obciążeniem a wariancją	384
Krzywe walidacji w Scikit-Learn	386
Krzywe uczenia	389
Walidacja w praktyce — wyszukiwanie w siatce	394
Podsumowanie	395
40. Inżynieria cech.....	396
Cechy o charakterze kategoryjnym	396
Cechy tekstowe	397
Konwersja obrazów na cechy	398
Cechy pochodne	399
Imputacja brakujących danych	401
Potoki przetwarzania w inżynierii cech	402
41. Dogłębne spojrzenie — naiwny klasyfikator Bayesa.....	404
Klasyfikacja bayesowska	404
Naiwny gaussowski klasyfikator Bayesa	405
Naiwny wielomianowy klasyfikator Bayesa	408
Przykład: klasyfikacja tekstu	408
Kiedy korzystać z naiwnego klasyfikatora Bayesa	411

42. Dogłębne spojrzenie — regresja liniowa.....	412
Prosta regresja liniowa	412
Regresja funkcjami bazowymi	415
Wielomianowe funkcje bazowe	415
Gaussowskie funkcje bazowe	416
Regularyzacja	418
Regresja grzbietowa (regularyzacja L_2)	420
Regresja lasso (regularyzacja L_1)	421
Przykład: przewidywanie ruchu rowerowego	422
43. Dogłębne spojrzenie — maszyny wektorów nośnych	428
Motywacje dla maszyn wektorów nośnych	428
Maszyny wektorów nośnych — maksymalizacja marginesu	430
Dopasowywanie maszyny wektorów nośnych	431
Maszyny wektorów nośnych z nieliniowymi granicami — jądro SVM	435
Dostrajanie SVM — zmiekczenie marginesów	438
Przykład: rozpoznawanie twarzy	440
Podsumowanie	444
44. Dogłębne spojrzenie — drzewa decyzyjne i lasy losowe	445
Motywacje dla lasów losowych — drzewa decyzyjne	445
Tworzenie drzewa decyzyjnego	446
Drzewa decyzyjne i nadmierne dopasowanie	448
Zespoły estymatorów — lasy losowe	449
Regresja z użyciem lasów losowych	451
Przykład: wykorzystanie lasu losowego do klasyfikacji cyfr	453
Podsumowanie	456
45. Dogłębne spojrzenie — analiza głównych składowych	457
Wprowadzenie do analizy głównych składowych	457
PCA jako metoda redukcji wymiarowości	461
Wykorzystanie PCA do wizualizacji — odręcznie zapisane cyfry	461
Co reprezentują składowe?	463
Wybór liczby składowych	465
PCA jako metoda filtrowania szumów	466
Przykład: rozpoznawanie twarzy	468
Podsumowanie	470
46. Dogłębne spojrzenie — manifold learning.....	472
Manifold learning — słowo „hello”	473
Skalowanie wielowymiarowe	473
Skalowanie wielowymiarowe jako metoda manifold learningu	477
Osadzenia nieliniowe — gdy zawodzi skalowanie wielowymiarowe	478

Rozmaitości nieliniowe — lokalnie liniowe osadzanie	480
Kilka przemyśleń na temat metod manifold learningu	481
Przykład: mapowanie izometryczne w zbiorze zdjęć twarzy	483
Przykład: wizualizacja struktury w liczbach	487
47. Dogłębne spojrzenie — klasteryzacja za pomocą algorytmu k-średnich	490
Wprowadzenie do algorytmu k-średnich	490
Estymacja-maksymalizacja	492
Przykłady	498
Przykład 1. Algorytm k-średnich w zbiorze digits	498
Przykład 2. Algorytm k-średnich w kompresji kolorów	501
48. Dogłębne spojrzenie — modele mieszanin rozkładów Gaussa	505
Motywacje dla modeli mieszanin rozkładów Gaussa — słabości algorytmu k-średnich	505
Uogólnienie algorytmu EM — modele mieszanin rozkładów Gaussa	509
Wybór rodzaju kowariancji	513
Modele mieszanin rozkładów Gaussa jako narzędzie do szacowania gęstości	513
Przykład: wykorzystanie GMM do generowania nowych danych	517
49. Dogłębne spojrzenie — jądrowy estymator gęstości	521
Motywacje dla jądrowego estymatora gęstości — histogramy	521
Jądrowy estymator gęstości w praktyce	526
Wybór parametru wygładzania za pomocą walidacji krzyżowej	527
Przykład: nie tak naiwny klasyfikator Bayesa	528
Anatomia niestandardowego estymatora	530
Korzystanie z naszego niestandardowego estymatora	531
50. Zastosowanie — potok przetwarzania do wykrywania twarzy.....	534
Cechy HOG	535
HOG w akcji — prosty detektor twarzy	536
1. Stwórz zbiór „pozytywnych” próbek	536
2. Stwórz zbiór „negatywnych” próbek	536
3. Połącz zbiory i wyodrębnij cechy HOG	538
4. Wytrenuj maszynę wektorów nośnych	538
5. Znajdź twarze na nowym zdjęciu	539
Zastrzeżenia i ulepszenia	541
Materiały dodatkowe na temat uczenia maszynowego	542

Praca z szeregami czasowymi

Początkowo Pandas zaprojektowano z myślą o modelowaniu finansowym, więc jak możesz się spodziewać, zawiera on obszerny zbiór narzędzi do pracy z datami, godzinami i danymi indeksowanymi czasowo. Dane dotyczące czasu występują w kilku wariantach, które omówię w tym rozdziale:

Znaczniki czasu

Określone momenty w czasie (np. 4 lipca 2021 r., godz. 7:00).

Przedziały czasu i okresy

Przedział pomiędzy określonym punktem początkowym i końcowym; na przykład czerwiec 2021 r. Okresy to zazwyczaj specjalne przypadki przedziałów, w których każdy przedział ma jednakową długość, a poszczególne przedziały nie zachodzą na siebie (np. 24-godzinne okresy obejmujące dni).

Czasy trwania lub odstępy czasu

Dokładny czas (np. 22,56 sekundy).

W tym rozdziale pokażę Ci, jak pracować z każdym z tych typów daty czy czasu w Pandas. Rozdział ten nie jest w żadnym wypadku kompletnym przewodnikiem po narzędziach do pracy z szeregami czasowymi, które są dostępne w Pythonie lub Pandas. Jest to jedynie ogólny przegląd tego, w jaki sposób użytkownik Pandas powinien podejść do pracy z danymi czasowymi. Zaczę od krótkiego omówienia narzędzi do obsługi dat i godzin dostępnych w Pythonie. Następnie przejdę do bardziej szczegółowego omówienia narzędzi z Pandas. Na koniec pokażę Ci kilka krótkich przykładów pracy z szeregami czasowymi w Pandas.

Daty i godziny w Pythonie

Python oferuje wiele sposobów reprezentacji dat i godzin oraz odstępow i przedziałów czasu. Podczas gdy narzędzia do pracy z szeregami czasowymi dostępne w Pandas zazwyczaj najlepiej sprawdzają się w analizie danych, warto przanalizować ich związek z innymi narzędziami wykorzystywanymi w Pythonie.

Daty i godziny w Pythonie — `datetime` i `dateutil`

Wbudowany w Pythona moduł `datetime` zawiera podstawowe obiekty do pracy z datami i godzinami w tym języku. Moduł ten wraz z modułem `dateutil` możesz wykorzystać do szybkiego wykonania wielu operacji na datach i godzinach. Na przykład do stworzenia daty możesz wykorzystać typ `datetime`:

```
In [1]: from datetime import datetime
        datetime(year=2021, month=7, day=4)
Out[1]: datetime.datetime(2021, 7, 4, 0, 0)
```

Za pomocą modułu `dateutil` możesz przekonwertować daty zapisane w łańcuchach znaków w różnych formatach na obiekty typu `datetime`:

```
In [2]: from dateutil import parser
        date = parser.parse("4th of July, 2021")
        date
Out[2]: datetime.datetime(2021, 7, 4, 0, 0)
```

Gdy masz już obiekt `datetime`, możesz na przykład wyświetlić dzień tygodnia:

```
In [3]: date.strftime('%A')
Out[3]: 'Sunday'
```

W powyższym przykładzie do wyświetlenia dnia tygodnia wykorzystałem jeden ze standardowych kodów formatu ('%A'). Więcej na ich temat znajdziesz w dokumentacji pakietu `datetime` (<https://oreil.ly/AGVR9>) w sekcji `strftime` (<https://oreil.ly/bjdsf>). Informacje o innych przydatnych w pracy z datami narzędziach znajdziesz też w dokumentacji pakietu `dateutils` (<https://oreil.ly/Y5Rwd>). Powiązany z nimi pakietem, którego nazwę warto zapamiętać, jest `pytz` (<https://oreil.ly/DU9Jr>). Pakiet ten zawiera narzędzia do pracy z powodującym najwięcej bólów głowy elementem danych czasowych, jakim jest obecność stref czasowych.

Siła `datetime` i `dateutil` leży w ich elastyczności i łatwej składni. Możesz wykorzystać te obiekty i ich metody, aby z łatwością wykonać prawie każdą operację, która może Cię zainteresować. Narzędzia te nie sprawdzają się wtedy, gdy chcesz pracować z dużymi tablicami dat i godzin. Tak jak tablice NumPy są zoptymalizowaną wersją list liczb zmiennoprzecinkowych, tak samo typowane tablice zakodowanych dat to zoptymalizowane odpowiedniki list obiektów typu `datetime`.

Typowane tablice znaczników czasu — `datetime64` z NumPy

Typ `datetime64` dostępny w NumPy pozwala zapisać daty w postaci 64-bitowych liczb całkowitych i tym samym umożliwia kompaktową reprezentację tablic znaczników czasu i ich wydajną obsługę. Stworzenie obiektu typu `datetime64` wymaga podania daty w określonym formacie wejściowym:

```
In [4]: import numpy as np
        date = np.array('2021-07-04', dtype=np.datetime64)
        date
Out[4]: array('2021-07-04', dtype='datetime64[D]')
```

Po zapisaniu dat we właściwej formie możemy wykonać na nich szybkie zwektoryzowane operacje:

```
In [5]: date + np.arange(12)
Out[5]: array(['2021-07-04', '2021-07-05', '2021-07-06', '2021-07-07',
              '2021-07-08', '2021-07-09', '2021-07-10', '2021-07-11',
              '2021-07-12', '2021-07-13', '2021-07-14', '2021-07-15'],
              dtype='datetime64[D]')
```

Ze względu na jednorodność typu w tablicach NumPy operacje tego rodzaju możemy wykonać znacznie szybciej, niż gdybyśmy pracowali bezpośrednio z obiektami `datetime` z Pythona. Różnica staje się widoczna zwłaszcza podczas pracy z dużymi tablicami (wektoryzując operacji na tablicach omówiłem w rozdziale 6.).

Jedną z cech obiektów typu `datetime64` i pokrewnych im obiektów typu `timedelta64` jest to, że wykorzystują one *bazową jednostkę czasu*. Ponieważ obiekt typu `datetime64` jest zapisywany na 64 bitach, maksymalny możliwy do zapisania w nim przedział czasu to 2^{64} razy bazowa jednostka. Innymi słowy, w typie `datetime64` występuje kompromis między *rozdzielczością czasową* a *maksymalnym przedziałem czasu*.

Na przykład, jeśli chcesz uzyskać rozdzielczość 1 nanosekundy, to w typie tym możesz zapisać zakres 2^{64} nanosekund, czyli nieco mniej niż 600 lat. NumPy wywnioskuje rodzaj użytej jednostki na podstawie danych wejściowych; na przykład w poniższym kodzie jednostką bazową jest jeden dzień.

```
In [6]: np.datetime64('2021-07-04')
Out[6]: numpy.datetime64('2021-07-04')
```

Oto znacznik czasu bazujący na minutach:

```
In [7]: np.datetime64('2021-07-04 12:00')
Out[7]: numpy.datetime64('2021-07-04T12:00')
```

Za pomocą jednego z wielu kodów formatu możesz wymusić użycie konkretnej jednostki bazowej. Na przykład w poniższym kodzie wymusiłem użycie czasu opartego na nanosekundach:

```
In [8]: np.datetime64('2021-07-04 12:59:59.50', 'ns')
Out[8]: numpy.datetime64('2021-07-04T12:59:59.500000000')
```

W tabeli 23.1 (pochodzącej z dokumentacji NumPy) znajdziesz listę kodów formatów wraz ze względnymi i bezwzględnymi przedziałami czasu, które można zapisać przy użyciu danej jednostki bazowej.

W przypadku rzeczywistych danych użyteczną wartością domyślną jest `datetime64[ns]`, która pozwala zakodować użyteczny zakres współczesnych dat z odpowiednio dużą precyzją.

Zauważ, że chociaż typ `datetime64` pozwala ominąć niektóre wady wbudowanego w Pythona typu `datetime`, brakuje w nim wielu wygodnych metod i funkcji znanych z `datetime` oraz `datetime`. Więcej informacji znajdziesz w sekcji na temat typu `datetime64` w dokumentacji NumPy (<https://oreil.ly/XDbck>).

Tabela 23.1. Opis kodów formatów daty i czasu

Kod	Znaczenie	Zakres czasu (względny)	Zakres czasu (bezwzględny)
Y	Rok	$\pm 9,2e18$ lat	[9,2e18 p.n.e. – 9,2e18 n.e.]
M	Miesiąc	$\pm 7,6e17$ lat	[7,6e17 p.n.e. – 7,6e17 n.e.]
W	Tydzień	$\pm 1,7e17$ lat	[1,7e17 p.n.e. – 1,7e17 n.e.]
D	Dzień	$\pm 2,5e16$ lat	[2,5e16 p.n.e. – 2,5e16 n.e.]
h	Godzina	$\pm 1,0e15$ lat	[1,0e15 p.n.e. – 1,0e15 n.e.]
m	Minuta	$\pm 1,7e13$ lat	[1,7e13 p.n.e. – 1,7e13 n.e.]
s	Sekunda	$\pm 2,9e12$ lat	[2,9e9 p.n.e. – 2,9e9 n.e.]
ms	Milisekunda	$\pm 2,9e9$ lat	[2,9e6 p.n.e. – 2,9e6 n.e.]
us	Mikrosekunda	$\pm 2,9e6$ lat	[290301 p.n.e. – 294241 n.e.]
ns	Nanosekunda	± 292 lat	[1678 n.e. – 2262 n.e.]
ps	Pikosekunda	± 106 dni	[1969 n.e. – 1970 n.e.]
fs	Femtosekunda	$\pm 2,6$ godzin	[1969 n.e. – 1970 n.e.]
as	Attosekunda	$\pm 9,2$ sekund	[1969 n.e. – 1970 n.e.]

Daty i godziny w Pandas — najlepsze elementy z obu światów

Pandas bazuje na wszystkich omówionych wcześniej narzędziach i zawiera obiekt typu `Timestamp`, który łączy w sobie łatwość użycia `datetime` i `dateutil` z wydajnym sposobem zapisu i zwektoryzowanym interfejsem `numpy.datetime64`. Z grupy obiektów typu `Timestamp` Pandas potrafi skonstruować `DatetimeIndex`, który można wykorzystać do indeksowania danych w obiektach typu `Series` lub `DataFrame`.

W poniższym kodzie wykorzystam narzędzia z Pandas do odtworzenia treści poprzednich przykładów. Poniżej pokazano, jak za pomocą Pandas można przekształcić datę zapisaną w łańcuchu znaków w obiekt typu `Timestamp` i jak wykorzystać kody formatów do wyświetlenia dnia tygodnia:

```
In [9]: import pandas as pd
        date = pd.to_datetime("4th of July, 2021")
        date
```

```
Out[9]: Timestamp('2021-07-04 00:00:00')
```

```
In [10]: date.strftime('%A')
Out[10]: 'Sunday'
```

Dodatkowo bezpośrednio na tym samym obiekcie możemy wykonać zwektoryzowane operacje w stylu NumPy:

```
In [11]: date + pd.to_timedelta(np.arange(12), 'D')
Out[11]: DatetimeIndex(['2021-07-04', '2021-07-05', '2021-07-06', '2021-07-07',
                        '2021-07-08', '2021-07-09', '2021-07-10', '2021-07-11',
                        '2021-07-12', '2021-07-13', '2021-07-14', '2021-07-15'],
                        dtype='datetime64[ns]', freq=None)
```

W następnym podrozdziale przyjrzymy się bliżej przekształcaniu szeregów czasowych za pomocą narzędzi dostępnych w Pandas.

Szeregi czasowe w Pandas — indeksowanie według czasu

Narzędzia do obsługi szeregów czasowych dostępne w Pandas stają się naprawdę przydatne, gdy chcemy indeksować dane według znaczników czasu. Możemy na przykład skonstruować obiekt typu Series zawierający dane indeksowane za pomocą czasu:

```
In [12]: index = pd.DatetimeIndex(['2020-07-04', '2020-08-04',
                                   '2021-07-04', '2021-08-04'])
        data = pd.Series([0, 1, 2, 3], index=index)
        data
Out[12]: 2020-07-04    0
         2020-08-04    1
         2021-07-04    2
         2021-08-04    3
         dtype: int64
```

Po umieszczeniu danych w obiekcie typu Series możemy go zaindeksować za pomocą dowolnej z metod indeksowania omówionych w poprzednich rozdziałach. W miejsce indeksów wystarczy po prostu wstawić daty:

```
In [13]: data['2020-07-04':'2021-07-04']
Out[13]: 2020-07-04    0
         2020-08-04    1
         2021-07-04    2
         dtype: int64
```

Istnieją również specjalne metody indeksowania dostępne jedynie dla dat, takie jak możliwość przekazania roku w celu uzyskania wycinka zawierającego wszystkie pochodzące z niego dane:

```
In [14]: data['2021']
Out[14]: 2021-07-04    2
         2021-08-04    3
         dtype: int64
```

W dalszej części rozdziału zobaczysz kolejne przykłady użycia dat w roli indeksów, ale najpierw przyjmiemy się bliżej dostępnym w Pandas strukturom danych do przechowywania szeregów czasowych.

Struktury danych do przechowywania szeregów czasowych w Pandas

W tym podrozdziale przedstawię podstawowe struktury danych do pracy z szeregami czasowymi dostępne w Pandas:

- Do przechowywania *znaczników czasu* można wykorzystać typ `Timestamp`. Jak już wspomniałem, jest to zamiennik natywnego typu `datetime` z Pythona oparty na bardziej wydajnym typie `numpy.datetime64`. Powiązany z nim typ indeksu to `DatetimeIndex`.
- Dla *przedziałów czasowych* Pandas zawiera typ `Period`. Pozwala on zapisać interwał o stałej częstotliwości i jest oparty na `numpy.datetime64`. Powiązany z nim typ indeksu to `PeriodIndex`.
- Do przechowywania *czasów trwania lub odstępów czasu* dostępny jest typ `Timedelta`. Jest on wydajniejszym zamiennikiem natywnego typu `datetime.timedelta` z Pythona, bazującym na typie `numpy.timedelta64`. Powiązany z nim typ indeksu to `TimedeltaIndex`.

Najbardziej podstawowymi z tych obiektów są obiekty typów `Timestamp` i `DatetimeIndex`. Chociaż można je utworzyć za pomocą konstruktorów, częściej wykorzystuje się do tego celu funkcję `pd.to_datetime`, która potrafi sparsować dane zapisane w wielu formatach. Przekazanie do `pd.to_datetime` pojedynczej daty daje w wyniku obiekt typu `Timestamp`, a przekazanie serii `dat` — obiekt typu `DatetimeIndex`:

```
In [15]: dates = pd.to_datetime([datetime(2021, 7, 3), '4th of July, 2021',
                                '2021-Jul-6', '07-07-2021', '20210708'])
        dates
Out[15]: DatetimeIndex(['2021-07-03', '2021-07-04', '2021-07-06', '2021-07-07',
                        '2021-07-08'],
                        dtype='datetime64[ns]', freq=None)
```

Dowolny obiekt typu `DatetimeIndex` można przekonwertować na `PeriodIndex` za pomocą funkcji `to_period`, której należy przekazać kod częstotliwości. W poniższym przykładzie wykorzystalem kod `'D'`, aby wskazać, że interesuje mnie częstotliwość dzienna:

```
In [16]: dates.to_period('D')
Out[16]: PeriodIndex(['2021-07-03', '2021-07-04', '2021-07-06', '2021-07-07',
                      '2021-07-08'],
                      dtype='period[D]')
```

Obiekt typu `TimedeltaIndex` powstaje na przykład, gdy odejmujemy od siebie dwie daty:

```
In [17]: dates - dates[0]
Out[17]: TimedeltaIndex(['0 days', '1 days', '3 days', '4 days', '5 days'],
                        > dtype='timedelta64[ns]', freq=None)
```

Regularne sekwencje dat — `pd.date_range`

Pandas zawiera kilka metod ułatwiających tworzenie sekwencji regularnych dat. Są to: `pd.date_range` dla znaczników czasu, `pd.period_range` dla okresów i `pd.timedelta_range` dla odstępów czasu. Wiesz już, że funkcje `range` z Pythona oraz `np.arange` z NumPy przyjmują punkt początkowy, punkt końcowy i opcjonalnie rozmiar kroku oraz zwracają sekwencję wartości. Podobnie funkcja `pd.date_range` przyjmuje datę początkową, datę końcową i opcjonalny kod częstotliwości oraz zwraca regularną sekwencję dat:

```
In [18]: pd.date_range('2015-07-03', '2015-07-10')
Out[18]: DatetimeIndex(['2015-07-03', '2015-07-04', '2015-07-05', '2015-07-06',
                        '2015-07-07', '2015-07-08', '2015-07-09', '2015-07-10'],
                        dtype='datetime64[ns]', freq='D')
```

Zakres dat można określić nie tylko za pomocą punktu początkowego i końcowego, ale również za pomocą punktu początkowego i liczby okresów:

```
In [19]: pd.date_range('2015-07-03', periods=8)
Out[19]: DatetimeIndex(['2015-07-03', '2015-07-04', '2015-07-05', '2015-07-06',
                        '2015-07-07', '2015-07-08', '2015-07-09', '2015-07-10'],
                        dtype='datetime64[ns]', freq='D')
```

Odstępy można modyfikować za pomocą argumentu `freq`, który domyślnie przyjmuje wartość `'D'`. W poniższym kodzie tworzę zakres godzinowych znaczników czasu:

```
In [20]: pd.date_range('2015-07-03', periods=8, freq='H')
Out[20]: DatetimeIndex(['2015-07-03 00:00:00', '2015-07-03 01:00:00',
                        '2015-07-03 02:00:00', '2015-07-03 03:00:00',
                        '2015-07-03 04:00:00', '2015-07-03 05:00:00',
                        '2015-07-03 06:00:00', '2015-07-03 07:00:00'],
                        dtype='datetime64[ns]', freq='H')
```

Do utworzenia regularnych sekwencji wartości typu `Period` lub `Timedelta` można wykorzystać analogiczne funkcje `pd.period_range` i `pd.timedelta_range`. Oto przykład stworzenia miesięcznych okresów:

```
In [21]: pd.period_range('2015-07', periods=8, freq='M')
Out[21]: PeriodIndex(['2015-07', '2015-08', '2015-09',
                      '2015-10', '2015-11', '2015-12',
                      '2016-01', '2016-02'],
                      dtype='period[M'])
```

oraz sekwencja kolejnych czasów trwania różniących się o godzinę:

```
In [22]: pd.timedelta_range(0, periods=6, freq='H')
Out[22]: TimedeltaIndex(['0 days 00:00:00', '0 days 01:00:00', '0 days 02:00:00',
                          '0 days 03:00:00', '0 days 04:00:00', '0 days 05:00:00'],
                          dtype='timedelta64[ns]', freq='H')
```

Zrozumienie tych przykładów wymaga poznania kodów częstotliwości Pandas, które przedstawię w kolejnym podrozdziale.

Częstotliwości i przesunięcia

Podstawowym elementem niezbędnym w pracy z narzędziami do obsługi szeregów czasowych w Pandas są koncepcje **częstotliwości** (ang. *frequency*) i **przesunięcia** (ang. *date offset*). W tabeli 23.2 podsumowano podstawowe kody częstotliwości. Kody te, tak jak kody `'D'` (dzień) i `'H'` (godzina) z poprzednich przykładów, pozwalają określić interesujące nas odstępy w czasie.

Tabela 23.2. Lista kodów częstotliwości dostępnych w Pandas

Kod	Opis	Kod	Opis
D	Rok kalendarzowy	B	Dzień roboczy
W	Tydzień		
M	Koniec miesiąca	BM	Koniec miesiąca obrotowego
Q	Koniec kwartału	BQ	Koniec kwartału obrotowego
A	Koniec roku	BA	Koniec roku obrotowego
H	Godziny	BH	Godziny robocze
T	Minuty		
S	Sekundy		
L	Milisekundy		
U	Mikrosekundy		
N	Nanosekundy		

Miesięczne, kwartalne i roczne częstotliwości reprezentują końce danego okresu. Dodanie sufixu S do dowolnego z nich spowoduje odniesienie się do ich początków (tabela 23.3).

Tabela 23.3. Lista kodów częstotliwości rozpoczynających się od początku danej jednostki

Kod	Opis	Kod	Opis
MS	Początek miesiąca	BMS	Początek miesiąca obrotowego
QS	Początek kwartału	BQS	Początek kwartału obrotowego
AS	Początek roku	BAS	Początek roku obrotowego

Dodatkowo możesz zmienić miesiąc używany do oznaczania dowolnego kodu kwartalnego lub rocznego, dodając do niego trzyliterowy sufix zawierający kod miesiąca:

- Q-JAN, BQ-FEB, QS-MAR, BQS-APR itp.
- A-JAN, BA-FEB, AS-MAR, BAS-APR itp.

W ten sam sposób możesz zmienić punkt podziału częstotliwości tygodniowej, dodając do litery W trzyliterowy kod dnia tygodnia, na przykład W-SUN, W-MON, W-TUE, W-WED itp.

Ponadto kody można łączyć z liczbami, aby określić inne częstotliwości. Na przykład, aby uzyskać częstotliwość 2 godzin i 30 minut, możesz połączyć ze sobą kody reprezentujące godziny (H) i minuty (T):

```
In [23]: pd.timedelta_range(0, periods=6, freq="2H30T")
Out[23]: TimedeltaIndex(['0 days 00:00:00', '0 days 02:30:00', '0 days 05:00:00',
                          '0 days 07:30:00', '0 days 10:00:00', '0 days 12:30:00'],
                          dtype='timedelta64[ns]', freq='150T')
```

Wszystkie te krótkie kody odnoszą się do konkretnych przypadków przesunięcia czasu w szeregach czasowych Pandas, które można znaleźć w module `pd.tseries.offsets`. Na przykład przesunięcie o dzień roboczy można utworzyć w następujący sposób:

```
In [24]: from pandas.tseries.offsets import BDay
         pd.date_range('2015-07-01', periods=6, freq=BDay())
Out[24]: DatetimeIndex(['2015-07-01', '2015-07-02', '2015-07-03', '2015-07-06',
                          '2015-07-07', '2015-07-08'],
                          dtype='datetime64[ns]', freq='B')
```

Więcej informacji na temat częstotliwości i przesunięć znajdziesz w sekcji *DateOffset* dokumentacji Pandas (<https://oreil.ly/J6JHA>).

Ponowne próbkowanie, przesuwanie i okna

Możliwość wykorzystania dat i godzin w roli indeksów do intuicyjnego organizowania i uzyskiwania dostępu do danych jest ważną cechą narzędzi do pracy z szeregami czasowymi dostępnymi w Pandas. Pozwala ona wykorzystać ogólne zalety indeksowanych danych (automatyczne wyrównywanie podczas wykonywania operacji, intuicyjny dostęp do danych i wycinków itp.) podczas pracy z danymi czasowymi. Pandas zawiera też kilka innych operacji stworzonych specjalnie z myślą o szeregach czasowych.

W tym podrozdziale przyjrzymy się kilku z nich. W przykładach wykorzystam niektórych dane o cenach akcji. Ponieważ Pandas został stworzony głównie z myślą o analizie danych finansowych, znajdziesz w nim kilka bardzo specyficznych narzędzi do tego celu. Na przykład pakiet *pandas-datareader* (możesz go zainstalować za pomocą polecenia `pip install pandas-datareader`) potrafi importować dane z różnych źródeł dostępnych w sieci. W poniższym kodzie pobieram za jego pomocą historyczne dane na temat cen akcji spółek indeksowanych przez S&P 500:

```
In [25]: from pandas_datareader import data
import yfinance as yf
yf.pdr_override()

sp500 = data.get_data_yahoo('^GSPC', start='2018-01-01', end='2022-01-01')
sp500.head()
```

Date	High	Low	Open	Close	Volume \
2018-01-02	2695.889893	2682.360107	2683.729980	2695.810059	3367250000
2018-01-03	2714.370117	2697.770020	2697.850098	2713.060059	3538660000
2018-01-04	2729.290039	2719.070068	2719.310059	2723.989990	3695260000
2018-01-05	2743.449951	2727.919922	2731.330078	2743.149902	3236620000
2018-01-08	2748.510010	2737.600098	2742.669922	2747.709961	3242650000

Date	Adj Close
2018-01-02	2695.810059
2018-01-03	2713.060059
2018-01-04	2723.989990
2018-01-05	2743.149902
2018-01-08	2747.709961

Dla uproszczenia wykorzystam jedynie ceny zamknięcia:

```
In [26]: sp500 = sp500['Close']
```

Ceny można zwizualizować za pomocą metody `plot`, którą należy wywołać po przeprowadzeniu wstępnej konfiguracji Matplotlib (część IV). Otrzymany wykres pokazano na rysunku 23.1.

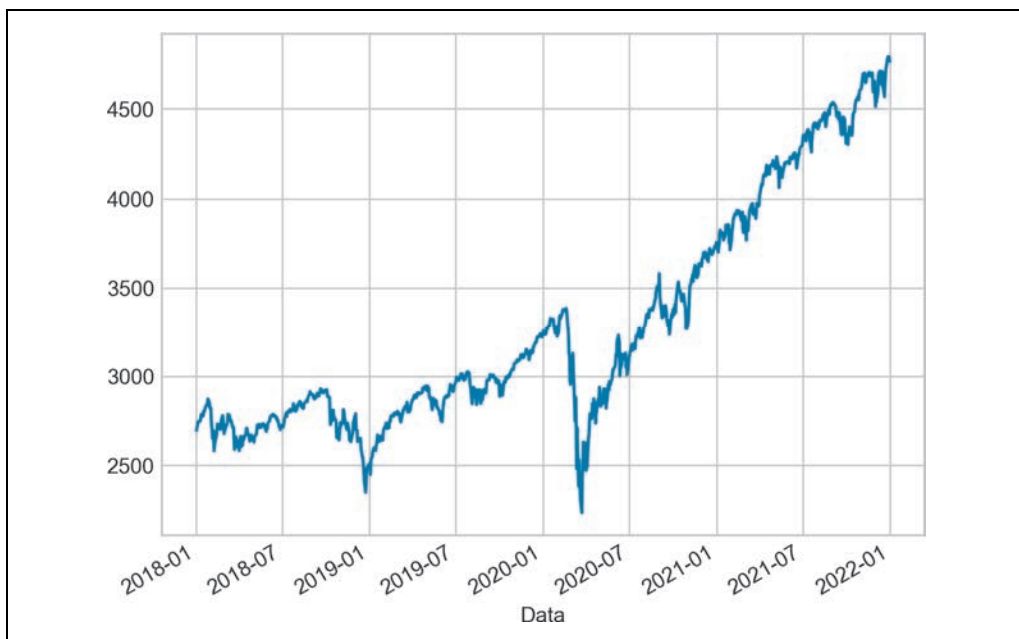
```
In [27]: %matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
sp500.plot();
plt.xlabel('Data');
```

Ponowne próbkowanie i zmiana częstotliwości

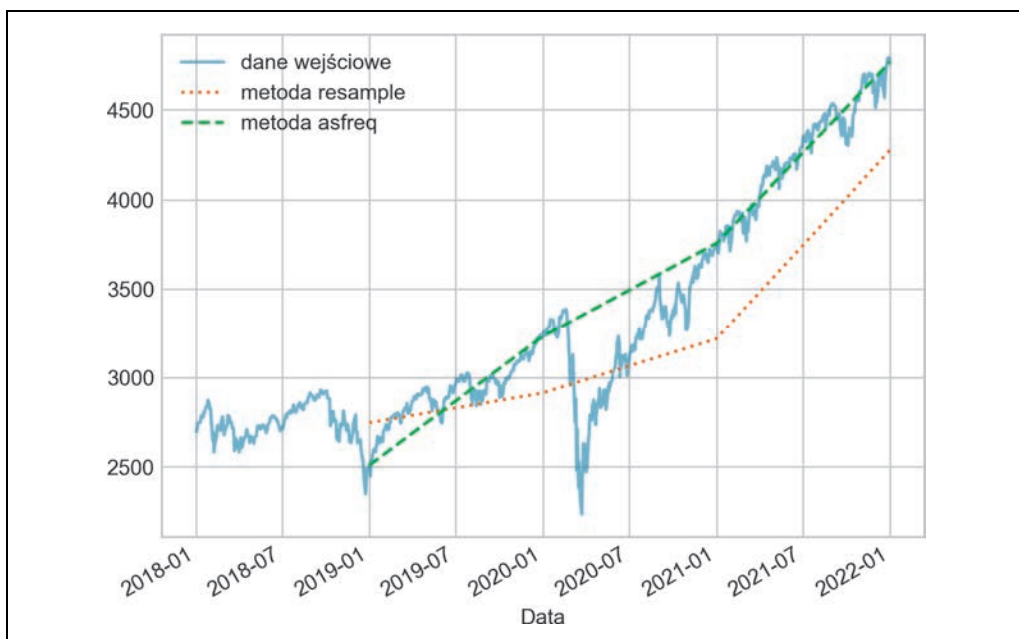
Podczas pracy z szeregami czasowymi często pojawia się potrzeba ponownego próbkowania danych (ang. *resampling*) z wyższą lub niższą częstotliwością. W Pandas można to zrobić za pomocą metody `resample` lub znacznie prostszej w użyciu metody `asfreq`. Podstawowa różnica pomiędzy nimi polega na tym, że `resample` przeprowadza *agregację danych*, a `asfreq` jedynie ich *selekcję*.

Porównajmy, co zwracają te dwie metody w przypadku zmniejszenia częstotliwości (ang. *downsampling*) próbkowania cen zamknięcia z indeksu S&P 500. W poniższym przykładzie dokonam ponownego próbkowania danych na końcu roku obrotowego. Wyniki pokazano na rysunku 23.2.

```
In [28]: sp500.plot(alpha=0.5, style='-')
sp500.resample('BA').mean().plot(style=':')
sp500.asfreq('BA').plot(style='--');
plt.legend(['dane wejściowe', 'metoda resample', 'metoda asfreq'],
           loc='upper left');
plt.xlabel('Data');
```

Rysunek 23.1. Ceny zamknięcia S&P500 w czasie



Rysunek 23.2. Ponowne próbkowanie cen zamknięcia S&P500

Zwróć uwagę na różnicę w wyniku. Metoda resample umieściła w każdym punkcie średnią cenę z poprzedniego roku, natomiast asfreq cenę z końca roku.

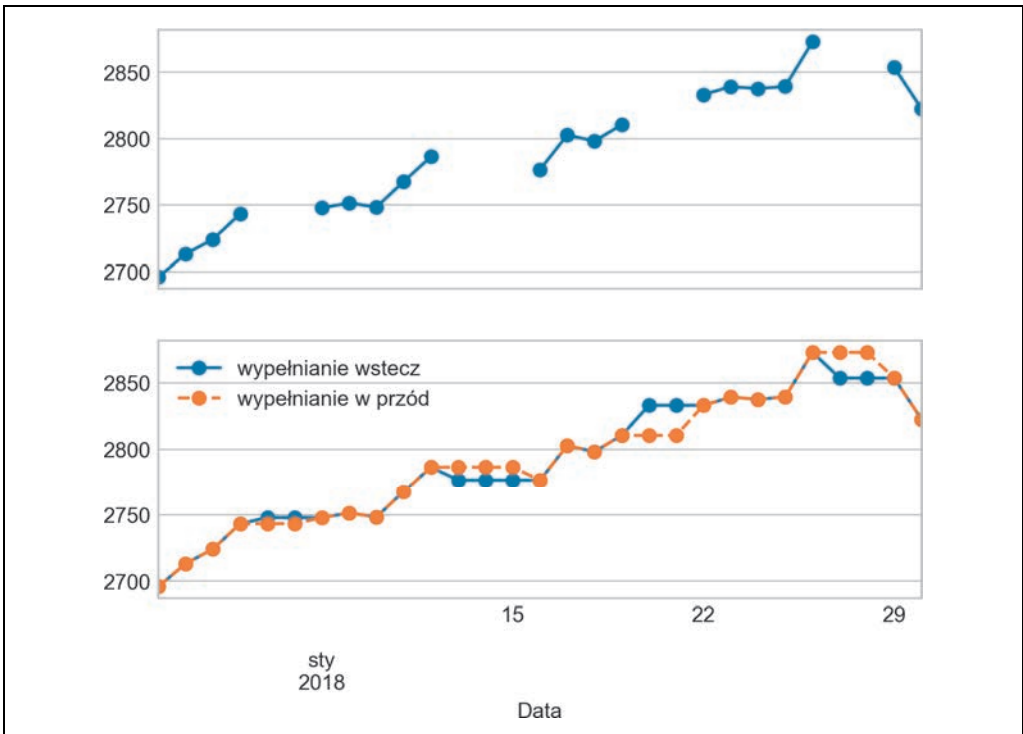
W przypadku zwiększania częstotliwości (ang. *upsampling*) `resample` i `asfreq` działają podobnie, ale `resample` ma więcej opcji. W tym przypadku w nowo powstałych punktach danych obie metody domyślnie umieszczają wartości typu *null*. Tak jak `pd.fillna` (omówiona w rozdziale 16.), `asfreq` również przyjmuje argument `method`, który pozwala określić sposób tworzenia nowych wartości. W poniższym przykładzie przeprowadzę ponowne próbkowanie danych z częstotliwością dzienną (tj. z uwzględnieniem weekendów, oryginalne dane zawierają jedynie ceny z dni roboczych). Wyniki tej operacji pokazano na rysunku 23.3.

```
In[29]: import locale
        locale.setlocale(locale.LC_TIME, "pl_PL")

        fig, ax = plt.subplots(2, sharex=True)
        data = sp500.iloc[:20]

        data.asfreq('D').plot(ax=ax[0], marker='o')

        data.asfreq('D', method='bfill').plot(ax=ax[1], style='-o')
        data.asfreq('D', method='ffill').plot(ax=ax[1], style='--o')
        ax[1].legend(["wypełnianie wstecz", "wypełnianie w przód"]);
        plt.xlabel('Data');
```



Rysunek 23.3. Interpolacja z użyciem metody wypełniania wstecz oraz wypełniania w przód

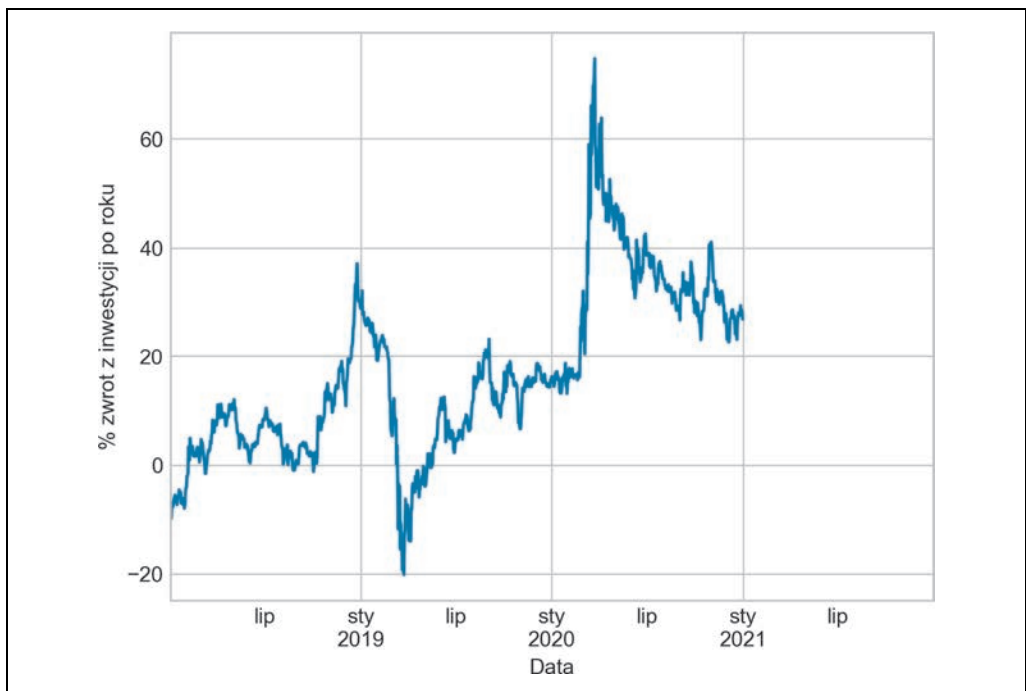
Ponieważ dane na temat indeksu S&P 500 zawierają jedynie ceny w dniach roboczych, pierwszy wykres z rysunku 23.3 zawiera przerwy reprezentujące braki cen w innych dniach. Na dolnym wykresie pokazano różnice między dwiema strategiami uzupełniania braków: wypełnianiem metodą w przód oraz wstecz.

Przesunięcia w czasie

Inną typową dla szeregów czasowych operacją jest przesuwanie danych w czasie. W Pandas służy do tego metoda `shift`, która pozwala przesunąć dane w czasie o określoną liczbę wpisów. W przypadku szeregów czasowych próbkowanych z równomierną częstotliwością metoda ta pozwala na zbadanie trendów w czasie.

W poniższym przykładzie ponownie próbkuję dane z częstotliwością jednego dnia i przesuwam je o 364 dni, aby obliczyć roczny zwrot z inwestycji na S&P 500 w czasie (wyniki pokazano na rysunku 23.4).

```
In [30]: sp500 = sp500.asfreq('D', method='pad')  
  
ROI = 100 * (sp500.shift(-365) - sp500) / sp500  
ROI.plot()  
plt.ylabel('% zwrot z inwestycji po roku');  
plt.xlabel('Data');
```



Rysunek 23.4. Zwrot z inwestycji po roku

Najgorszy roczny zwrot charakteryzuje akcje zakupione w okolicach marca 2019 r. Jest to związane z krachem spowodowanym pandemią koronawirusa, która rozpoczęła się dokładnie rok później. Jak można się spodziewać, najwyższy roczny zwrot dotyczył akcji kupionych w marcu 2020 r. Osoby, które je wtedy tanio kupiły, wykazały się dużą dalekowzrocznością lub miały po prostu szczęście.

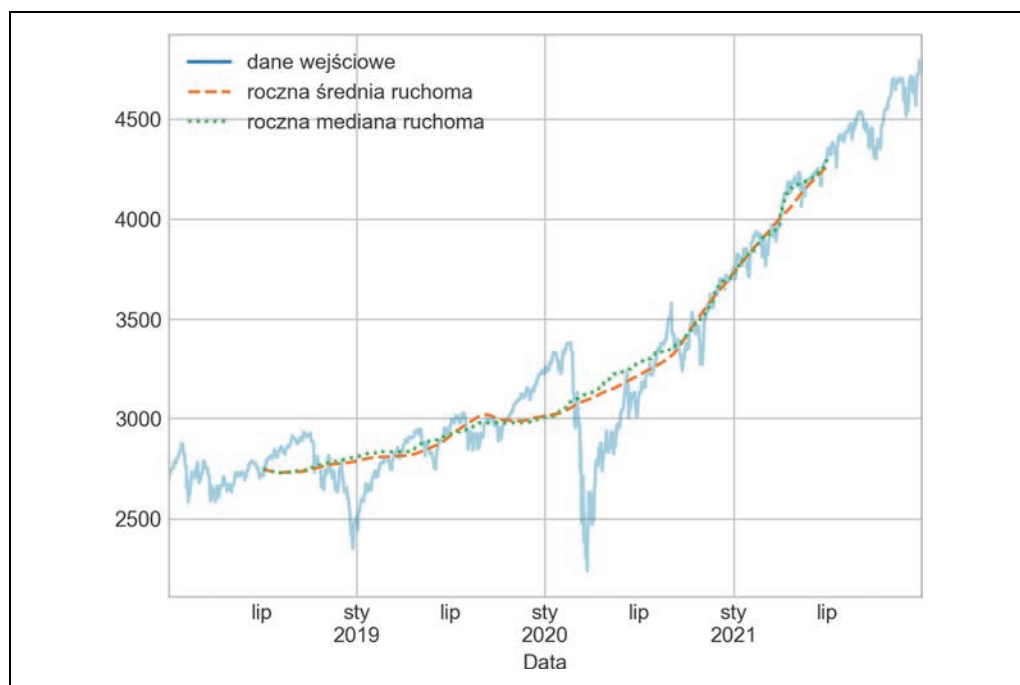
Ruchome okna

Obliczanie statystyk krocących jest trzecim rodzajem operacji specyficznych dla szeregów czasowych, które znajdziemy w Pandas. Statystyki te można obliczyć za pomocą atrybutu `rolling` obiektów typu `Series` i `DataFrame`. Atrybut ten zwraca widok podobny do tego, który widzieliśmy w przypadku operacji grupowania (rozdział 20.). Na tym widoku można wywołać wiele typowych funkcji agregujących.

Możemy na przykład przeanalizować roczną średnią kroczącą i odchylenie standardowe cen akcji (rysunek 23.5).

```
In [31]: rolling = sp500.rolling(365, center=True)

data = pd.DataFrame({'input': sp500,
                    'roczna średnia ruchoma': rolling.mean(),
                    'roczna mediana ruchoma': rolling.median()})
ax = data.plot(style=['-', '--', ':'])
ax.lines[0].set_alpha(0.3)
plt.xlabel('Data');
```



Rysunek 23.5. Statystyki kroczące dla indeksu S&P500

Tak jak w przypadku grupowania, również do przeprowadzenia niestandardowych obliczeń możesz wykorzystać metody `aggregate` i `apply`.

Skąd dowiedzieć się więcej

W tym rozdziale zamieściłem jedynie krótkie podsumowanie niektórych najważniejszych narzędzi Pandas do pracy z szeregami czasowymi. Bardziej szczegółowe omówienie znajdziesz w sekcji *Time series/date functionality* w internetowej dokumentacji Pandas (<https://oreil.ly/uC3pB>).

Innym doskonałym źródłem informacji jest książka *Python w analizie danych. Przetwarzanie danych za pomocą pakietów Pandas i NumPy oraz środowiska IPython*. Wydanie II Wesa McKinneya (Helion). Jest to nieocenione źródło informacji na temat Pandas. W książce tej położono nacisk na wykorzystanie narzędzi do analiz szeregów czasowych w kontekście biznesowym i finansowym. Znajdziesz w niej też dokładniejsze omówienie związanych z nimi szczegółów, w tym kalendarza biznesowego, stref czasowych i pokrewnych zagadnień.

Jak zawsze, możesz również skorzystać z pomocy wbudowanej w IPython, aby zbadać i wypróbować inne opcje omówionych w tym rozdziale funkcji i metod. Moim zdaniem często jest to najlepszy sposób na poznanie nowego pakietu Pythona.

Przykład: wizualizacja danych o liczbie rowerów w Seattle

W ramach bardziej złożonego przykładu pracy z szeregami czasowymi przyjrzymy się liczbie rowerów przejeżdżających przez most Fremont Bridge w Seattle (<https://oreil.ly/6qVBt>). Dane te pochodzą z automatycznego licznika rowerów zainstalowanego pod koniec 2012 roku. Licznik składa się z czujników indukcyjnych zainstalowanych po wschodniej i zachodniej stronie mostu. Godzinowe dane zawierające informacje o liczbie rowerów, które przejechały przez most, można pobrać ze strony <http://data.seattle.gov>. Zbiór danych *Fremont Bridge Bicycle Counter* znajduje się w kategorii *Transportation*.

Plik CSV wykorzystany w tej książce można pobrać w następujący sposób:

```
In [32]: # url = ('https://raw.githubusercontent.com/jakevdp/'
#           'bicycle-data/main/FremontBridge.csv')
# !curl -O {url}
```

Po pobraniu danych możemy wykorzystać Pandas do wczytania zawartości pliku CSV do ramki danych. Za pomocą argumentów `read_csv` określam, że daty mają być sparsowane w sposób automatyczny oraz że indeksem ramki ma być kolumna `Date`:

```
In [33]: data = pd.read_csv('data/FremontBridge.csv', index_col='Date', parse_dates=True)
data.head()
```

```
Out[33]:
```

Date	Fremont Bridge Total	Fremont Bridge East Sidewalk	\
2019-11-01 00:00:00	12.0	7.0	
2019-11-01 01:00:00	7.0	0.0	
2019-11-01 02:00:00	1.0	0.0	
2019-11-01 03:00:00	6.0	6.0	
2019-11-01 04:00:00	6.0	5.0	

```
Fremont Bridge West Sidewalk
```

```
Date
2019-11-01 00:00:00      5.0
2019-11-01 01:00:00      7.0
2019-11-01 02:00:00      1.0
2019-11-01 03:00:00      0.0
2019-11-01 04:00:00      1.0
```

Dla wygody skracam nazwy kolumn:

```
In [34]: data.columns = ['Total', 'East', 'West']
```

Przyjrzyjmy się teraz statystykom podsumowującym te dane:

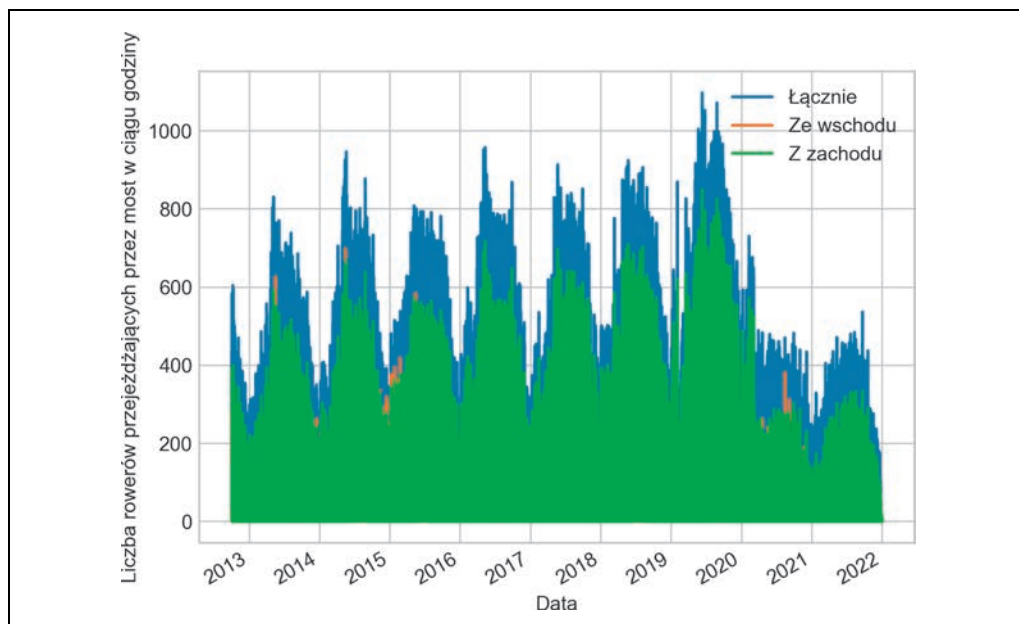
```
In [35]: data.dropna().describe()
Out[35]:
```

	Total	East	West
count	147255.000000	147255.000000	147255.000000
mean	110.341462	50.077763	60.263699
std	140.422051	64.634038	87.252147
min	0.000000	0.000000	0.000000
25%	14.000000	6.000000	7.000000
50%	60.000000	28.000000	30.000000
75%	145.000000	68.000000	74.000000
max	1097.000000	698.000000	850.000000

Wizualizacja danych

Pewien wgląd w ten zbiór danych można uzyskać poprzez wizualizację jego zawartości. Zaczynam od wykreślenia surowych danych (wynik pokazano na rysunku 23.6).

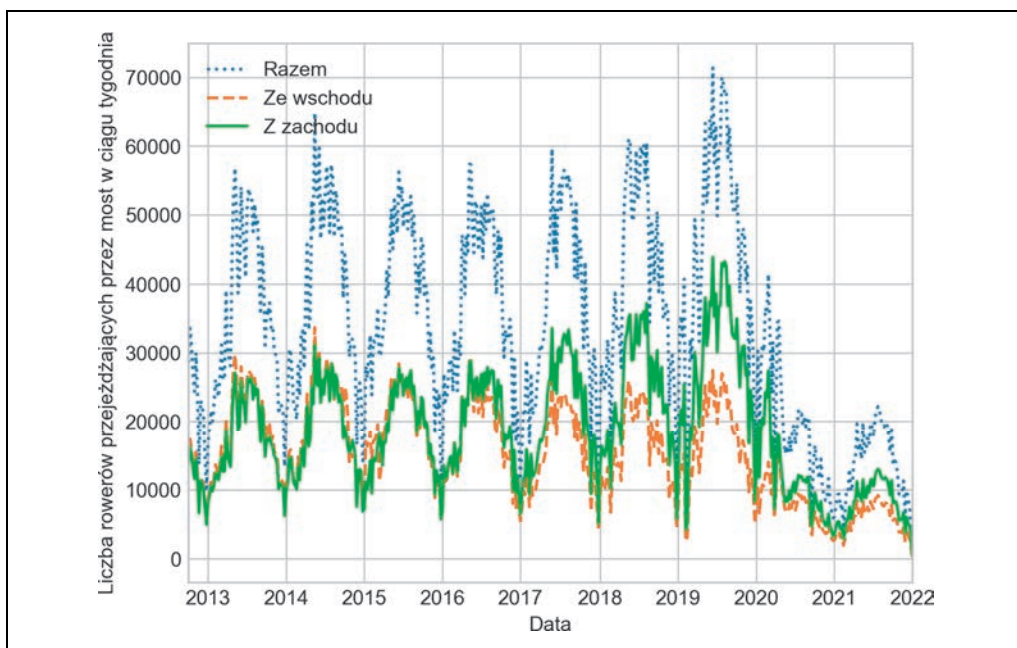
```
In [36]: data.plot()
plt.ylabel('Liczba rowerów przejeżdżających przez most w ciągu godziny');
plt.xlabel('Data');
plt.legend(['Łącznie', 'Ze wschodu', 'Z zachodu']);
```



Rysunek 23.6. Godzinowe liczby rowerów przejeżdżających przez most Fremont Bridge w Seattle

Około 150 000 próbek to zbyt dużo, aby dało się im przyjrzeć na wykresie. Więcej informacji na temat tych danych można uzyskać za pomocą ponownego próbkowania z mniejszą częstotliwością. Na poniższym listingu przeprowadzam ponowne próbkowanie z tygodniową częstotliwością (rysunek 23.7).

```
In [37]: weekly = data.resample('W').sum()
weekly.plot(style=[':', '--', '-'])
plt.ylabel('Liczba rowerów przejeżdżających przez most w ciągu tygodnia');
plt.xlabel('Data');
plt.legend(['Razem', 'Ze wschodu', 'Z zachodu']);
```

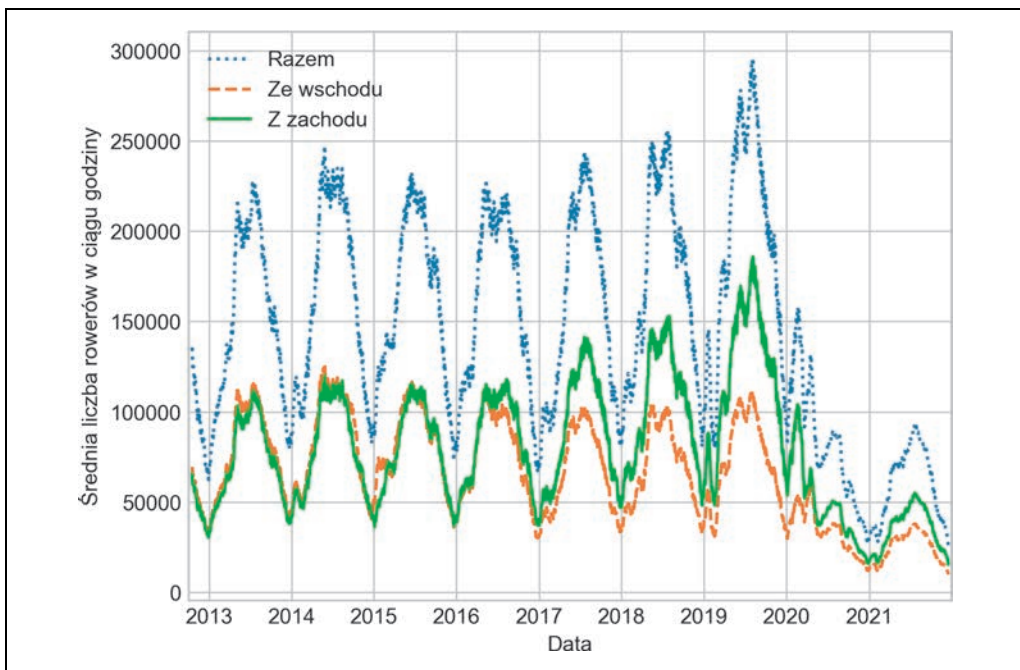


Rysunek 23.7. Tygodniowa liczba przejazdów rowerem przez most Fremont Bridge w Seattle

Ponowne próbkowanie ujawniło pewne trendy. Jak można się było spodziewać, ludzie częściej jeżdżą na rowerze latem niż zimą, a liczba rowerów w każdej porze roku zmienia się z tygodnia na tydzień (prawdopodobnie w zależności od pogody; zajrzyj do rozdziału 42., w którym dokładniej przyjrzymy się tej kwestii). Co więcej, od początku 2020 roku w danych widoczny jest wpływ pandemii COVID-19 na zmianę zwyczajów dotyczących dojazdów do pracy.

Inną opcją, którą warto wykorzystać podczas agregacji danych, jest użycie średniej kroczącej. Można ją obliczyć za pomocą funkcji `pd.rolling_mean`. Za pomocą poniższego kodu zbadam 30-dniową średnią kroczącą. Upewniam się też, że okno zostało wyśrodkowane (wyniki pokazano na rysunku 23.8).

```
In [38]: daily = data.resample('D').sum()
daily.rolling(30, center=True).sum().plot(style=['-', ':', '--'])
plt.ylabel('Średnia liczba rowerów w ciągu godziny');
plt.xlabel('Data');
plt.legend(['Razem', 'Ze wschodu', 'Z zachodu']);
```



Rysunek 23.8. Średnia krocząca z tygodniowych danych na temat liczby rowerów

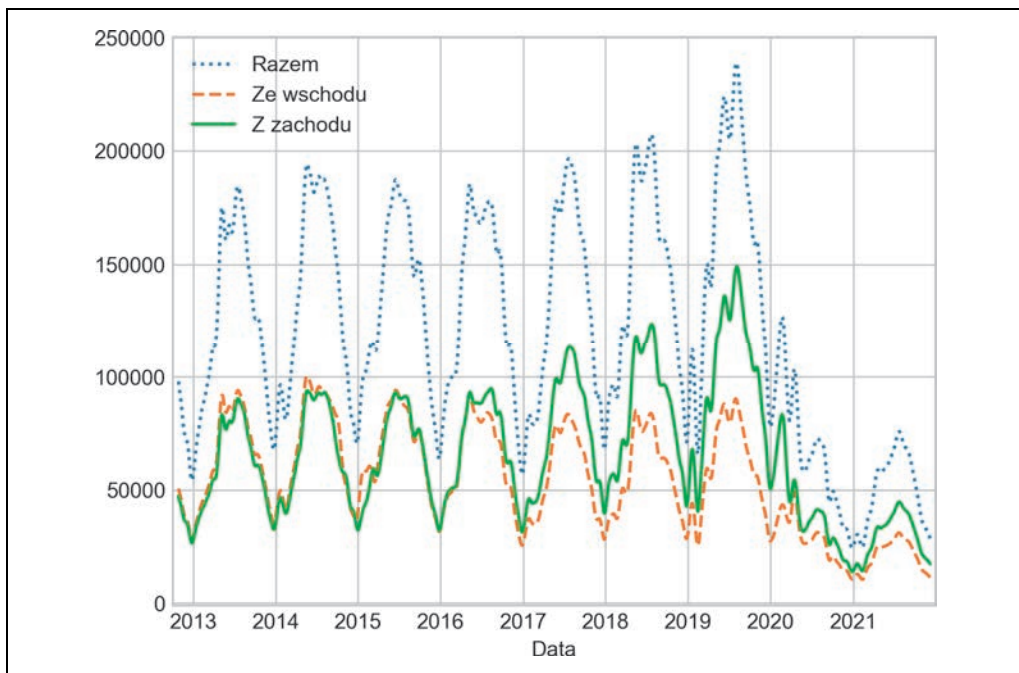
„Postrzępiony” kształt wykresu wynika ze sposobu odcinania wartości przez okno. Gładszą wersję średniej kroczącej można uzyskać, korzystając z innego okna, na przykład okna Gaussa (rezultat pokazano na rysunku 23.9). W poniższym kodzie ustalę zarówno szerokość całego okna (50 dni), jak i szerokość okna Gaussa (10 dni):

```
In [39]: daily.rolling(50, center=True,
            win_type='gaussian').sum(std=10).plot(style=[':', '--', '-']);
            plt.xlabel('Data');
            plt.legend(['Razem', 'Ze wschodu', 'Z zachodu']);
```

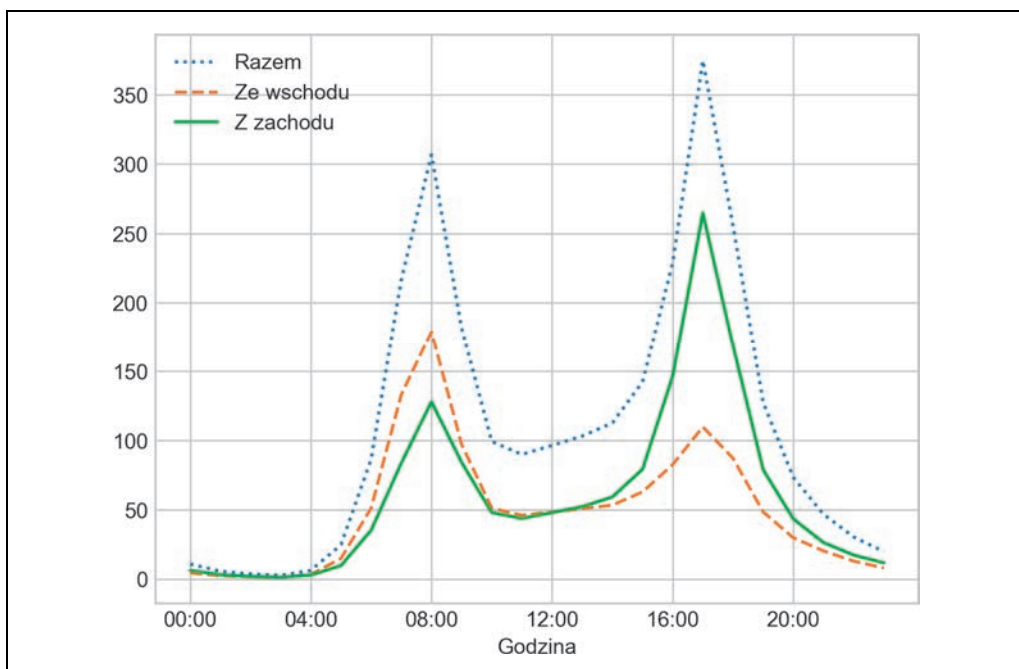
Zagłębianie się w dane

Chociaż wygładzona wersja danych pozwala zaobserwować ich ogólny trend, operacja wygładzania eliminuje znaczną część ich struktury. Moglibyśmy na przykład chcieć spojrzeć na średni ruch w funkcji pory dnia. Możemy to zrobić za pomocą funkcji grupby, którą omówiłem w rozdziale 20. (wyniki grupowania pokazano na rysunku 23.10).

```
In [40]: by_time = data.groupby(data.index.time).mean()
            hourly_ticks = 4 * 60 * 60 * np.arange(6)
            by_time.plot(xticks=hourly_ticks, style=['-', ':', '--']);
            plt.xlabel('Godzina');
            plt.legend(['Razem', 'Ze wschodu', 'Z zachodu']);
```

Rysunek 23.9. Tygodniowe liczby rowerów po zastosowaniu wygładzania za pomocą okna Gaussa

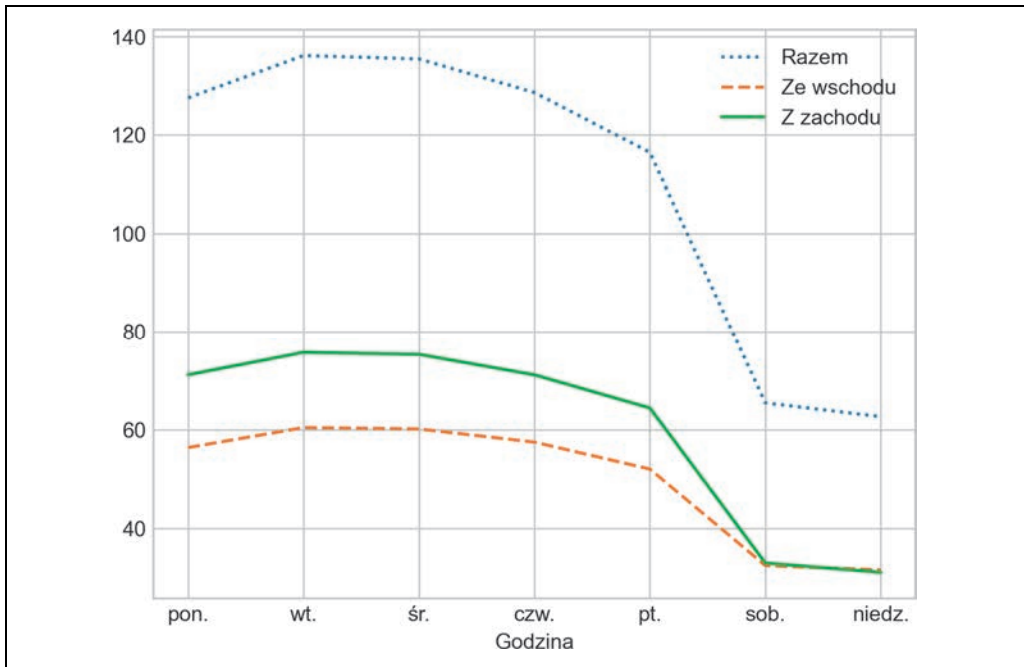


Rysunek 23.10. Średnia liczba rowerów przejeżdżających przez most o określonej godzinie

Ruch godzinowy ma silnie bimodalny charakter i osiąga maksima około 8:00 i 17:00. Jest to prawdopodobnie spowodowane dużą liczbą osób dojeżdżających przez most do pracy. Istnieje również komponent kierunkowy. Z danych wynika, że wschodni chodnik jest częściej wykorzystywany podczas porannych dojazdów do pracy, a zachodni podczas popołudniowych powrotów do domu.

Interesująca może być również zmienność w zależności od dnia tygodnia. Do jej zbadania ponownie wykorzystam operację grupowania (wyniki pokazano na rysunku 23.11).

```
In [41]: by_weekday = data.groupby(data.index.dayofweek).mean()
         by_weekday.index = ['pon.', 'wt.', 'śr.', 'czw.',
                             'pt.', 'sob.', 'niedz.'];
         by_weekday.plot(style=[':', '--', '-']);
         plt.xlabel('Godzina');
         plt.legend(['Razem', 'Ze wschodu', 'Z zachodu']);
```



Rysunek 23.11. Średnia dzienna liczba rowerów

Na rysunku widoczne są duże różnice pomiędzy ruchem w dni powszednie i w ciągu weekendu. Od poniedziałku do piątku przez most przejeżdża około dwa razy więcej rowerzystów niż w soboty i niedziele.

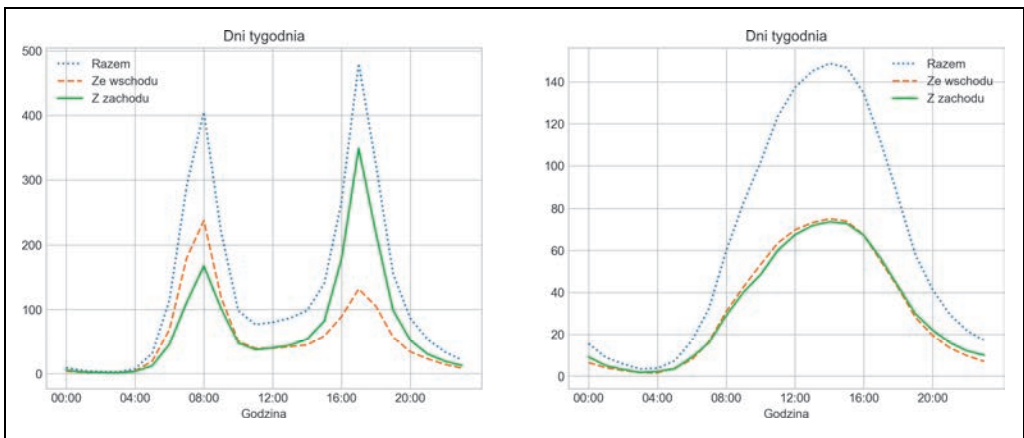
Mając to na uwadze, wykorzystam grupowanie do przyjrzenia się trendom godzinowym w dni powszednie oraz w weekendy. Zaczynam od pogrupowania danych według flag reprezentujących weekend i porę dnia:

```
In [42]: weekend = np.where(data.index.weekday < 5, 'Weekday', 'Weekend')
         by_time = data.groupby([weekend, data.index.time]).mean()
```

Następnie wykorzystam niektóre z narzędzi dostępnych w pakiecie Matplotlib (omawiam je w rozdziale 31.) do umieszczenia dwóch wykresów obok siebie. Otrzymany wykres pokazano na rysunku 23.12.

```
In [43]: import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 2, figsize=(14, 5))
by_time.loc['Weekday'].plot(ax=ax[0], title='Dni tygodnia',
                           xticks=hourly_ticks, style=[':', '--', '-'])
ax[0].set_xlabel('Godzina');
ax[0].legend(['Razem', 'Ze wschodu', 'Z zachodu']);

by_time.loc['Weekend'].plot(ax=ax[1], title='Dni tygodnia',
                             xticks=hourly_ticks, style=[':', '--', '-']);
ax[1].set_xlabel('Godzina');
ax[1].legend(['Razem', 'Ze wschodu', 'Z zachodu']);
```



Rysunek 23.12. Średnia godzinowa liczba rowerów w dni powszednie i weekendy

Wyniki pokazują widoczny w dni robocze bimodalny wzorec dojazdów do pracy oraz jednomodalny wzorec weekendowy związany z rekreacyjną jazdą na rowerze. Interesująca może być dalsza analiza tych danych i zbadanie wpływu pogody, temperatury, pory roku i innych czynników na wzorce dojazdów do pracy. Więcej informacji na ten temat znajdziesz we wpisie *Is Seattle Really Seeing an Uptick In Cycling?* na moim blogu (<https://oreil.ly/j5oEI>). Do jego przygotowania wykorzystałem podzbiór z tego zbioru danych. Do danych tych powrócę raz jeszcze w rozdziale 42., w którym spróbujemy dopasować do nich model.

PROGRAM PARTNERSKI

— GRUPY 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
Helion 

Wydobywaj z danych mądre odpowiedzi na trudne pytania!

Python udostępnia pierwszorzędne narzędzia i biblioteki przeznaczone specjalnie do pracy z danymi. Zdobyły one uznanie wielu naukowców i ekspertów, ceniących ten język za wysoką jakość rozwiązań służących do wydobywania wiedzy z danych. Aby uzyskać najlepsze możliwe efekty, trzeba dobrze poznać zarówno poszczególne biblioteki Pythona, jak i zasady pracy z nimi.

Ta książka stanowi wszechstronne omówienie wszystkich bibliotek Pythona potrzebnych naukowcom i specjalistom pracującym z danymi. Znalazł się tu dokładny opis IPython, NumPy, Pandas, Matplotlib, Scikit-Learn i innych narzędzi. Podręcznik uwzględni przede wszystkim ich aspekty praktyczne, dzięki czemu świetnie się sprawdzi w rozwiązywaniu codziennych problemów z manipulowaniem, przekształcaniem, oczyszczaniem i wizualizacją różnych typów danych, a także jako pomoc podczas tworzenia modeli statystycznych i modeli uczenia maszynowego. Doceni go każdy, kto zajmuje się obliczeniami naukowymi w Pythonie.

To wydanie zawiera jasne przykłady, które pomogą Ci skonfigurować i wykorzystać narzędzia do nauki o danych i uczenia maszynowego.

Anne Bonner, założycielka i dyrektorka generalna Content Simplicity

Nauczysz się:

- pracować w naukowym środowisku obliczeniowym IPython
- korzystać ze specjalistycznych bibliotek przeznaczonych do pracy z danymi
- stosować typy ndarray i DataFrame do przechowywania i przetwarzania danych
- tworzyć różnego rodzaju wizualizacje danych za pomocą Matplotlib

Jake VanderPlas jest inżynierem oprogramowania w Google Research. Współtworzy i rozwija narzędzia do przetwarzania dużych ilości danych, w tym pakiety Scikit-Learn, SciPy, Astropy, Altair i JAX. Jest także twórcą samouczków, często występuje jako prelegent na branżowych konferencjach.

	KOD KORZYŚCI Sięgnij po więcej! ▶	
 helion.pl	ISBN 978-83-289-0068-4	
 HELION SA ul. Kościuszki 1c 44-100 Gliwice tel: 32 250 99 63 helion@helion.pl	 9 788328 900684	
Cena: 129,00 zł		