

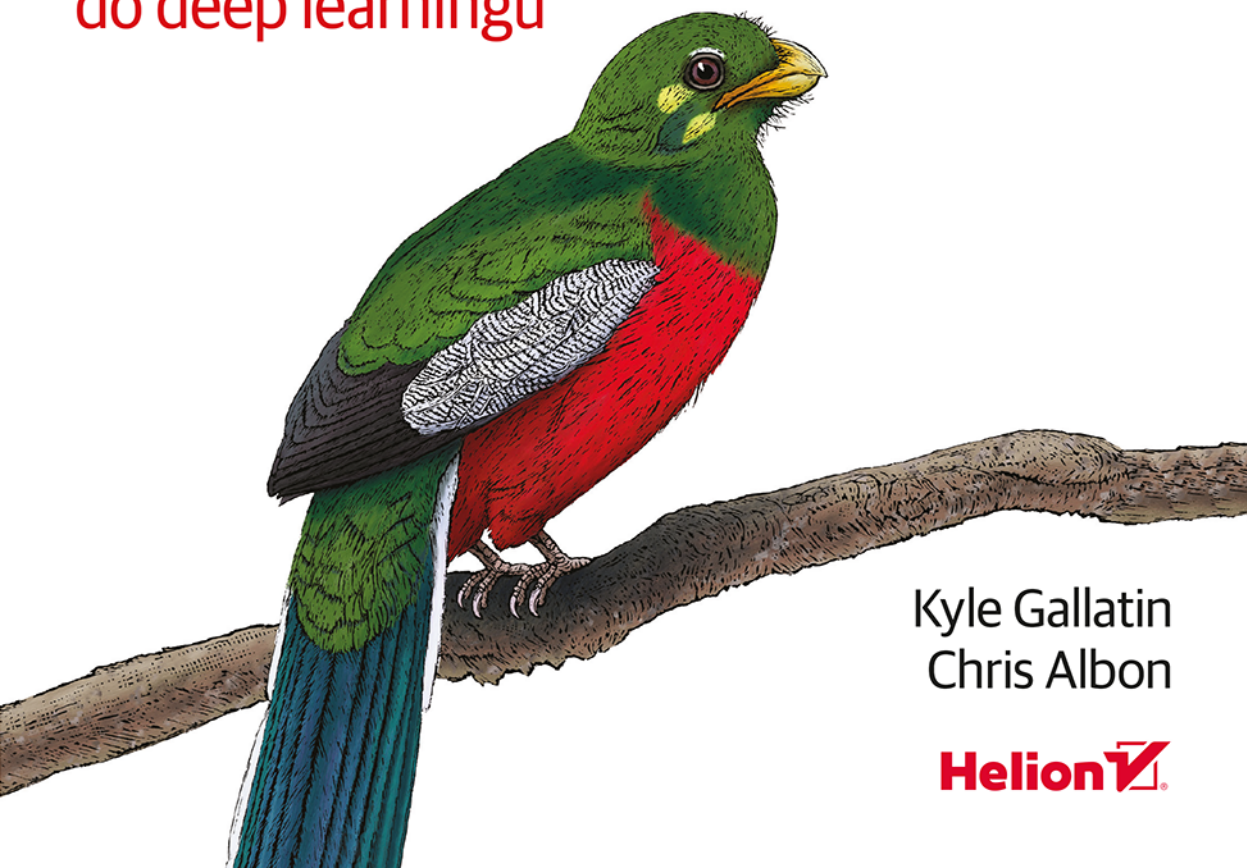
O'REILLY®

Wydanie II

Uczenie maszynowe w Pythonie

Receptury

Od przygotowania danych
do deep learningu



Kyle Gallatin
Chris Albon

Helion 

Tytuł oryginału: Machine Learning with Python Cookbook: Practical Solutions
from Preprocessing to Deep Learning, 2nd Edition

Tłumaczenie: Robert Górczyński

ISBN: 978-83-289-0811-6

© 2024 Helion S.A.

Authorized Polish translation of the English edition of *Machine Learning with Python Cookbook*,
2nd Edition ISBN 9781098135720 © 2023 Kyle Gallatin.

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.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/uczma2>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<https://ftp.helion.pl/przyklady/uczma2.zip>

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)

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

Wprowadzenie	11
1. Wektor, macierz i tablica	15
1.0. Wprowadzenie	15
1.1. Tworzenie wektora	15
1.2. Tworzenie macierzy	16
1.3. Tworzenie macierzy rzadkiej	17
1.4. Wstępna alokacja tablicy NumPy	19
1.5. Pobieranie elementów	19
1.6. Opisywanie macierzy	21
1.7. Przeprowadzanie operacji na wszystkich elementach	22
1.8. Znajdowanie wartości maksymalnej i minimalnej	23
1.9. Obliczanie średniej, wariancji i odchylenia standardowego	24
1.10. Zmiana kształtu tablicy	24
1.11. Transponowanie wektora lub macierzy	25
1.12. Spłaszczanie macierzy	26
1.13. Znajdowanie rzędu macierzy	28
1.14. Pobieranie przekątnej macierzy	28
1.15. Obliczanie śladu macierzy	29
1.16. Obliczanie iloczynu skalarnego	30
1.17. Dodawanie i odejmowanie macierzy	31
1.18. Mnożenie macierzy	32
1.19. Odwracanie macierzy	33
1.20. Generowanie liczb losowych	34
2. Wczytywanie danych	36
2.0. Wprowadzenie	36
2.1. Wczytywanie przykładowego zbioru danych	36
2.2. Tworzenie symulowanego zbioru danych	38

2.3.	Wczytywanie pliku CSV	41
2.4.	Wczytywanie pliku Excela	42
2.5.	Wczytywanie pliku JSON	43
2.6.	Wczytywanie pliku Parquet	44
2.7.	Wczytywanie pliku Avro	45
2.8.	Wykonywanie zapytań do bazy danych SQLite	46
2.9.	Wykonywanie zapytań do zdalnej bazy danych SQL	47
2.10.	Wczytywanie danych z Google Sheets	48
2.11.	Wczytywanie danych z kubełka S3	49
2.12.	Wczytywanie danych nieposiadających struktury	50
3.	Przygotowywanie danych	52
3.0.	Wprowadzenie	52
3.1.	Tworzenie ramki danych	53
3.2.	Opisywanie danych	54
3.3.	Poruszanie się po ramce danych	56
3.4.	Pobieranie wierszy na podstawie pewnych warunków	58
3.5.	Sortowanie wartości	59
3.6.	Zastępowanie wartości	60
3.7.	Zmiana nazwy kolumny	61
3.8.	Znajdowanie wartości minimalnej, maksymalnej, sumy, średniej i liczby elementów w kolumnie	63
3.9.	Znajdowanie unikatowych wartości	64
3.10.	Obsługa brakujących wartości	65
3.11.	Usuwanie kolumn	67
3.12.	Usuwanie wiersza	68
3.13.	Usuwanie powielonych wierszy	70
3.14.	Grupowanie wierszy według wartości	71
3.15.	Grupowanie wierszy według czasu	73
3.16.	Agregowanie operacji i danych statystycznych	75
3.17.	Iterowanie przez kolumnę	77
3.18.	Wywoływanie funkcji dla wszystkich elementów kolumny	78
3.19.	Wywoływanie funkcji dla grupy	79
3.20.	Konkatenacja obiektów typu DataFrame	79
3.21.	Złączanie obiektów typu DataFrame	81
4.	Obsługa danych liczbowych	85
4.0.	Wprowadzenie	85
4.1.	Przeskalowywanie cechy	85
4.2.	Standaryzowanie cechy	87
4.3.	Normalizowanie obserwacji	88

4.4.	Generowanie cech wielomianowych i interakcji	90
4.5.	Transformacja cech	92
4.6.	Wykrywanie elementów odstających	93
4.7.	Obsługa elementów odstających	95
4.8.	Dyskretyzacja cech	97
4.9.	Grupowanie obserwacji przy użyciu klastra	98
4.10.	Usuwanie obserwacji, w których brakuje wartości	100
4.11.	Uzupełnianie brakujących wartości	102
5.	Obsługa danych kategoryzujących	105
5.0.	Wprowadzenie	105
5.1.	Kodowanie nominalnych cech kategoryzujących	106
5.2.	Kodowanie porządkowych cech kategoryzujących	108
5.3.	Kodowanie słowników cech	110
5.4.	Wstawianie brakujących wartości klas	112
5.5.	Obsługa niezerównoważonych klas	114
6.	Obsługa tekstu	118
6.0.	Wprowadzenie	118
6.1.	Oczyszczanie tekstu	118
6.2.	Przetwarzanie i oczyszczanie danych HTML	121
6.3.	Usuwanie znaku przestankowego	122
6.4.	Tokenizacja tekstu	123
6.5.	Usuwanie słów o małym znaczeniu	124
6.6.	Stemming słów	125
6.7.	Oznaczanie części mowy	126
6.8.	Rozpoznawanie nazwanych jednostek	127
6.9.	Kodowanie tekstu za pomocą modelu worka słów	129
6.10.	Określanie wagi słów	131
6.11.	Używanie wektorów tekstu do obliczania podobieństwa tekstu w zapytaniu wyszukiwania	133
6.12.	Używanie klasyfikatora analizy sentymentu	134
7.	Obsługa daty i godziny	136
7.0.	Wprowadzenie	136
7.1.	Konwertowanie ciągu tekstowego na datę	136
7.2.	Obsługa stref czasowych	138
7.3.	Pobieranie daty i godziny	139
7.4.	Podział danych daty na wiele cech	140
7.5.	Obliczanie różnicy między datami	141
7.6.	Kodowanie dni tygodnia	142

7.7.	Tworzenie cechy opóźnionej w czasie	143
7.8.	Użycie okien wpływającego czasu	144
7.9.	Obsługa brakujących danych w serii danych zawierających wartości daty i godziny	146
8.	Obsługa obrazów	149
8.0.	Wprowadzenie	149
8.1.	Wczytywanie obrazu	149
8.2.	Zapisywanie obrazu	152
8.3.	Zmiana wielkości obrazu	153
8.4.	Kadrowanie obrazu	154
8.5.	Rozmywanie obrazu	155
8.6.	Wyostrażanie obrazu	157
8.7.	Zwiększanie kontrastu	158
8.8.	Izolowanie kolorów	160
8.9.	Progowanie obrazu	162
8.10.	Usuwanie tła obrazu	164
8.11.	Wykrywanie krawędzi	166
8.12.	Wykrywanie narożników w obrazie	168
8.13.	Tworzenie cech w uczeniu maszynowym	171
8.14.	Użycie histogramu koloru jako cech	173
8.15.	Użycie wytrenowanych embeddingów jako cech	176
8.16.	Wykrywanie obiektów za pomocą OpenCV	178
8.17.	Klasyfikowanie obrazów za pomocą PyTorch	180
9.	Redukcja wymiarowości za pomocą wyodrębniania cech	182
9.0.	Wprowadzenie	182
9.1.	Redukowanie cech za pomocą głównych składowych	183
9.2.	Redukowanie cech, gdy dane są liniowo nierozłączne	185
9.3.	Redukowanie cech przez maksymalizację rozłączności klas	187
9.4.	Redukowanie cech za pomocą rozkładu macierzy	190
9.5.	Redukowanie cech w rzadkich danych	191
10.	Redukcja wymiarowości za pomocą wyboru cech	194
10.0.	Wprowadzenie	194
10.1.	Progowanie wariancji cechy liczbowej	195
10.2.	Progowanie wariancji cechy binarnej	196
10.3.	Obsługa wysoce skorelowanych cech	197
10.4.	Usuwanie nieistotnych dla klasyfikacji cech	199
10.5.	Rekurencyjne eliminowanie cech	201

11. Ocena modelu	204
11.0. Wprowadzenie	204
11.1. Modele sprawdzianu krzyżowego	204
11.2. Tworzenie modelu regresji bazowej	208
11.3. Tworzenie modelu klasyfikacji bazowej	209
11.4. Ocena prognoz klasyfikatora binarnego	211
11.5. Ocena progowania klasyfikatora binarnego	214
11.6. Ocena prognoz klasyfikatora wieloklasowego	217
11.7. Wizualizacja wydajności klasyfikatora	219
11.8. Ocena modelu regresji	221
11.9. Ocena modelu klasteryzacji	223
11.10. Definiowanie niestandardowych współczynników oceny modelu	224
11.11. Wizualizacja efektu wywieranego przez wielkość zbioru uczącego	226
11.12. Tworzenie raportu tekstowego dotyczącego współczynnika oceny	228
11.13. Wizualizacja efektu wywieranego przez zmianę wartości hiperparametrów	229
12. Wybór modelu	232
12.0. Wprowadzenie	232
12.1. Wybór najlepszych modeli przy użyciu wyczerpującego wyszukiwania	233
12.2. Wybór najlepszych modeli za pomocą przeszukiwania losowego	235
12.3. Wybór najlepszych modeli z wielu algorytmów uczenia maszynowego	237
12.4. Wybór najlepszych modeli na etapie przygotowywania danych	239
12.5. Przyspieszanie wyboru modelu za pomocą równoległości	241
12.6. Przyspieszanie wyboru modelu przy użyciu metod charakterystycznych dla algorytmu	242
12.7. Ocena wydajności po wyborze modelu	244
13. Regresja liniowa	246
13.0. Wprowadzenie	246
13.1. Wyznaczanie linii	246
13.2. Obsługa wpływu interakcji	248
13.3. Wyznaczanie zależności nieliniowej	250
13.4. Redukowanie wariancji za pomocą regularyzacji	252
13.5. Redukowanie cech za pomocą regresji metodą LASSO	254
14. Drzewa i lasy	256
14.0. Wprowadzenie	256
14.1. Trenowanie klasyfikatora drzewa decyzyjnego	256
14.2. Trenowanie regresora drzewa decyzyjnego	258
14.3. Wizualizacja modelu drzewa decyzyjnego	259
14.4. Trenowanie klasyfikatora losowego lasu	261

14.5. Trenowanie regresora losowego lasu	263
14.6. Ocena losowego lasu za pomocą estymatora błędu out-of-bag	264
14.7. Identyfikacja ważnych cech w losowych lasach	265
14.8. Wybór ważnych cech w losowym lesie	267
14.9. Obsługa niezerównoważonych klas	268
14.10. Kontrolowanie wielkości drzewa	270
14.11. Poprawa wydajności za pomocą wzmocnienia	271
14.12. Wytrenowanie modelu XGBoost	272
14.13. Poprawianie wydajności w czasie rzeczywistym za pomocą LightGBM	274
15. Algorytm k najbliższych sąsiadów	276
15.0. Wprowadzenie	276
15.1. Wyszukiwanie najbliższych sąsiadów obserwacji	276
15.2. Tworzenie klasyfikatora k najbliższych sąsiadów	278
15.3. Ustalanie najlepszej wielkości sąsiedztwa	280
15.4. Tworzenie klasyfikatora najbliższych sąsiadów opartego na promieniu	281
15.5. Wyszukiwanie przybliżonych najbliższych sąsiadów	282
15.6. Ocena przybliżonych najbliższych sąsiadów	285
16. Regresja logistyczna	288
16.0. Wprowadzenie	288
16.1. Trenowanie klasyfikatora binarnego	288
16.2. Trenowanie klasyfikatora wieloklasowego	290
16.3. Redukcja wariancji poprzez regularyzację	291
16.4. Trenowanie klasyfikatora na bardzo dużych danych	292
16.5. Obsługa niezerównoważonych klas	293
17. Maszyna wektora nośnego	295
17.0. Wprowadzenie	295
17.1. Trenowanie klasyfikatora liniowego	295
17.2. Obsługa liniowo nierozdzielnych klas przy użyciu funkcji jądra	298
17.3. Określanie prognozowanego prawdopodobieństwa	301
17.4. Identyfikacja wektorów nośnych	303
17.5. Obsługa niezerównoważonych klas	304
18. Naiwny klasyfikator bayesowski	306
18.0. Wprowadzenie	306
18.1. Trenowanie klasyfikatora dla cech ciągłych	307
18.2. Trenowanie klasyfikatora dla cech dyskretnych lub liczebnych	309
18.3. Trenowanie naiwnego klasyfikatora bayesowskiego dla cech binarnych	310
18.4. Kalibrowanie prognozowanego prawdopodobieństwa	311

19. Klasteryzacja	313
19.0. Wprowadzenie	313
19.1. Klasteryzacja za pomocą k średnich	313
19.2. Przyspieszanie klasteryzacji za pomocą k średnich	316
19.3. Klasteryzacja za pomocą algorytmu meanshift	316
19.4. Klasteryzacja za pomocą algorytmu DBSCAN	318
19.5. Klasteryzacja za pomocą łączenia hierarchicznego	319
20. Tensory w PyTorch	321
20.0. Wprowadzenie	321
20.1. Utworzenie tensora	321
20.2. Utworzenie tensora z poziomu NumPy	322
20.3. Utworzenie tensora rzadkiego	323
20.4. Wybór elementów tensora	324
20.5. Opisanie tensora	325
20.6. Przeprowadzanie operacji na elementach tensora	327
20.7. Wyszukiwanie wartości minimalnej i maksymalnej	327
20.8. Zmiana kształtu tensora	328
20.9. Transponowanie tensora	329
20.10. Spłaszczanie tensora	330
20.11. Obliczanie iloczynu skalarnego	330
20.12. Mnożenie tensorów	331
21. Sieci neuronowe	333
21.0. Wprowadzenie	333
21.1. Używanie silnika Autograd frameworka PyTorch	334
21.2. Przygotowywanie danych dla sieci neuronowej	336
21.3. Projektowanie sieci neuronowej	337
21.4. Trenowanie klasyfikatora binarnego	341
21.5. Trenowanie klasyfikatora wieloklasowego	343
21.6. Trenowanie regresora	345
21.7. Generowanie prognoz	347
21.8. Wizualizacja historii trenowania	349
21.9. Redukcja nadmiernego dopasowania za pomocą regularyzacji wagi	352
21.10. Redukcja nadmiernego dopasowania za pomocą techniki wcześniejszego zakończenia procesu uczenia	354
21.11. Redukcja nadmiernego dopasowania za pomocą techniki porzucenia	357
21.12. Zapisywanie postępu modelu uczącego	359
21.13. Dostrajanie sieci neuronowej	361
21.14. Wizualizacja sieci neuronowej	364

22. Sieci neuronowe dla danych pozbawionych struktury	367
22.0. Wprowadzenie	367
22.1. Wytrenowanie sieci neuronowej na potrzeby klasyfikacji obrazów	368
22.2. Wytrenowanie sieci neuronowej na potrzeby klasyfikacji tekstu	370
22.3. Dostrajanie wytrenowanego modelu na potrzeby klasyfikacji obrazu	372
22.4. Dostrajanie wytrenowanego modelu na potrzeby klasyfikacji tekstu	375
23. Zapisywanie, wczytywanie i udostępnianie wytrenowanych modeli	377
23.0. Wprowadzenie	377
23.1. Zapisywanie i wczytywanie modelu biblioteki scikit-learn	377
23.2. Zapisywanie i wczytywanie modelu biblioteki TensorFlow	379
23.3. Zapisywanie i wczytywanie modelu PyTorch	380
23.4. Udostępnianie modeli scikit-learn	382
23.5. Udostępnianie modeli TensorFlow	384
23.6. Udostępnianie modeli PyTorch za pomocą Seldon	386

Przygotowywanie danych

3.0. Wprowadzenie

Przygotowywanie danych (ang. *data wrangling*) to dość często używane pojęcie, najczęściej w celu opisanego procesu przekształcenia niezmodyfikowanych danych na postać czystego i zorganizowanego formatu informacji gotowych do użycia. Dla nas to tylko jeden — choć zarazem niezwykle ważny — krok na etapie wstępnego przetwarzania danych.

Najczęściej wykorzystywaną strukturą stosowaną do przygotowywania danych jest tzw. ramka danych, która jest intuicyjna w użyciu i jednocześnie niezwykle elastyczna. Ramka danych ma postać tabelaryczną, co oznacza, że została oparta na wierszach i kolumnach, podobnie jak dane przechowywane w arkuszu kalkulacyjnym. Oto przykład ramki danych utworzonej na podstawie informacji o pasażerach Titanica:

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Umieszczenie wczytanych informacji w ramce danych.
dataframe = pd.read_csv(url)

# Wyświetlenie pierwszych pięciu wierszy.
dataframe.head(5)
```

Oto tabela pokazująca przykładowe dane wejściowe wczytane z pliku CSV:

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.00	female	1	1
1	Allison, Miss Helen Loraine	1st	2.00	female	0	1
2	Allison, Mr Hudson Joshua Creighton	1st	30.00	male	0	0
3	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	female	0	1
4	Allison, Master Hudson Trevor	1st	0.92	male	1	0

Trzeba zwrócić uwagę na trzy ważne kwestie dotyczące tej ramki danych.

Po pierwsze, każdy wiersz ramki danych odpowiada jednej obserwacji (tutaj jednemu pasażerowi), natomiast każda kolumna przedstawia jedną cechę — płeć, wiek itd. Na podstawie informacji zawartych w pierwszej obserwacji można powiedzieć, że panna Elisabeth Walton Allen podróżowała pierwszą klasą, w chwili katastrofy Titanica miała 29 lat, była kobietą i przeżyła katastrofę.

Po drugie, każda kolumna zawiera tytuł (na przykład `Name`, `PCl ass` i `Age`), zaś każdy wiersz ma numer indeksu (na przykład 0 dla panny Elisabeth Walton Allen, której dopisało szczęście). Dzięki tym danym można pobierać, a następnie przeprowadzać operacje na obserwacjach i cechach.

Po trzecie, dwie kolumny, `Sex` i `SexCode`, zawierają te same informacje, ale zapisane w różnych formatach. W kolumnie `Sex` płeć kobiety została zapisana jako `female`, natomiast w kolumnie `SexCode` kobieta została oznaczona liczbą całkowitą 1. Ponieważ chcemy, aby wszystkie cechy były uni-katowe, trzeba będzie usunąć jedną z tych kolumn.

W rozdziale przedstawię różne techniki przeznaczone do przeprowadzania operacji na ramkach danych za pomocą biblioteki `pandas`. Celem tych działań jest przygotowanie zbioru obserwacji o doskonałej strukturze, gotowego do dalszego wstępnego przetwarzania.

3.1. Tworzenie ramki danych

Problem

Chcesz utworzyć nową ramkę danych.

Rozwiązanie

Biblioteka `pandas` oferuje wiele metod przeznaczonych do tworzenia nowych ramek danych, czyli obiektów typu `DataFrame`. Jedną z nich pozwala na łatwe utworzenie ramki danych za pomocą słownika Pythona. W słowniku każdy klucz jest nazwą kolumny, zaś wartość listą. Poszczególne elementy słownika odpowiadają wierszom.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie słownika.
dictionary = {
    "Name": ['Jacky Jackson', 'Steven Stevenson'],
    "Age": [38, 25],
    "Driver": [True, False]
}

# Utworzenie obiektu typu DataFrame.
dataframe = pd.DataFrame(dictionary)

# Wyświetlenie obiektu DataFrame.
dataframe
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	Age	Driver
0	Jacky Jackson	38	True
1	Steven Stevenson	25	False

Po utworzeniu obiektu `DataFrame` można na jego końcu dodawać nowe dane, używając do tego listy wartości, jak pokazałem w kolejnym fragmencie kodu.

```
# Dodanie kolumny opisującej kolor oczu.
dataframe["Eyes"] = ["Brown", "Blue"]

# Wyświetlenie obiektu DataFrame.
dataframe
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	Age	Driver	Eyes
0	Jacky Jackson	38	True	Brown
1	Steven Stevenson	25	False	Blue

Analiza

Można odnieść wrażenie, że biblioteka `pandas` oferuje niemal nieograniczoną liczbę sposobów na tworzenie obiektu typu `DataFrame`. W rzeczywistości do utworzenia pustego obiektu `DataFrame`, a następnie wypełnienia go danymi niemal nigdy nie dochodzi. Zamiast tego obiekt jest tworzony na podstawie rzeczywistych danych wczytanych z innego źródła, na przykład pliku CSV lub bazy danych.

3.2. Opisywanie danych

Problem

Chcesz wyświetlić pewne cechy charakterystyczne obiektu `DataFrame`.

Rozwiązanie

Jedną z najprostszych czynności, którą można wykonać po wczytaniu danych, jest wyświetlenie kilku pierwszych wierszy za pomocą metody `head()`.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Wyświetlenie dwóch pierwszych wierszy.
dataframe.head(2)
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.00	female	1	1
1	Allison, Miss Helen Loraine	1st	2.00	female	0	1

Można też sprawdzić liczbę istniejących wierszy i kolumn.

```
# Wyświetlenie wielkości danych.  
dataframe.shape
```

```
(1313, 6)
```

Metoda `describe()` dostarcza opisowe dane statystyczne dotyczące kolumn zawierających dane liczbowe.

```
# Wyświetlenie pewnych danych statystycznych.  
dataframe.describe()
```

Oto dane po wykonaniu przedstawionego polecenia:

	Age	Survived	SexCode
Count	756.000000	1313.000000	1313.000000
Mean	30.397989	0.342727	0.351866
Std	14.259049	0.474802	0.477734
Min	0.170000	0.000000	0.000000
25%	21.000000	0.000000	0.000000
50%	28.000000	0.000000	0.000000
75%	39.000000	1.000000	1.000000
Max	71.000000	1.000000	1.000000

Z kolei metoda `info()` wyświetla zwięzłe podsumowanie dotyczące wskazanego obiektu `DataFrame`:

```
# Wyświetlenie informacji.  
dataframe.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1313 entries, 0 to 1312  
Data columns (total 6 columns):  
#   Column      Non-Null Count  Dtype  
---  -  
0   Name        1313 non-null  object  
1   PClass      1313 non-null  object  
2   Age         756 non-null   float64  
3   Sex         1313 non-null  object  
4   Survived    1313 non-null  int64  
5   SexCode     1313 non-null  int64  
dtypes: float64(1), int64(2), object(3)  
memory usage: 61.7+ KB
```

Analiza

Po wczytaniu danych warto się zapoznać z ich strukturą i rodzajem przechowywanych w nich informacji. W idealnym świecie bezpośrednio przeglądane byłyby rzeczywiste dane. Jednak w większości sytuacji te dane mogą się składać z dziesiątek tysięcy, setek tysięcy lub nawet milionów wierszy i kolumn. Dlatego też trzeba pobrać jedynie próbki pozwalające na wyświetlanie niewielkich wycinków i obliczanie podsumowania danych statystycznych dotyczących tych informacji.

W omawianym tutaj rozwiązaniu użyłem niewielkiego zbioru danych przedstawiającego pasażerów Titanica znajdujących się na pokładzie w trakcie jego ostatniego rejsu. Metoda `head()` pozwala na przejrzanie kilku pierwszych (domyślnie pięciu) wierszy danych. Ewentualnie można użyć metody `tail()` do wyświetlenia kilku ostatnich wierszy. Wynikiem działania metody `shape()` jest liczba wierszy i kolumn przechowywanych we wskazanym obiekcie `DataFrame`. Z kolei metoda `describe()` dostarcza wybrane podstawowe dane statystyczne dotyczące wszystkich kolumn zawierających dane liczbowe. Natomiast metoda `info()` wyświetla pewną liczbę użytecznych informacji na temat wskazanego obiektu `DataFrame`, m.in. typy indeksu i kolumn danych, liczbę wartości innych niż `null` oraz poziom użycia pamięci.

Warto w tym miejscu dodać, że podsumowanie danych statystycznych nie zawsze przedstawia prawdziwy obraz przetwarzanych danych. Na przykład biblioteka `pandas` traktuje kolumny `Survived` i `SexCode` jako liczbowe, ponieważ zawierają one dane w postaci zer i jedynek. Jednak w omawianym przykładzie te wartości liczbowe przedstawiają kategorie. Dlatego też jeśli wartość kolumny `Survived` wynosi 1, to oznacza, że dany pasażer przeżył katastrofę Titanica. Pewne wygenerowane wartości w podsumowaniu danych statystycznych nie mają więc sensu — przykładem jest tutaj odchylenie standardowe dla kolumny `SexCode` określającej płeć pasażera.

3.3. Poruszanie się po ramce danych

Problem

Musisz wybrać pojedyncze dane lub wycinek ramki danych.

Rozwiązanie

Do pobrania jednego lub dwóch wierszy wartości użyj metody `loc()` lub `iloc()`.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Pobranie pierwszego wiersza.
dataframe.iloc[0]
```

```

Name      Allen, Miss Elisabeth Walton
PClass    1st
Age       29
Sex       female
Survived   1
SexCode    1
Name: 0, dtype: object

```

Dwukropek może zostać wykorzystany do zdefiniowania wycinka składającego się z żądanych wierszy. W kolejnym poleceniu pokazałem przykład wycinka obejmującego wiersze od drugiego do czwartego.

```

# Pobranie trzech wierszy.
dataframe.iloc[1:4]

```

Oto dane po wykonaniu przedstawionego polecenia:

	Name	PClass	Age	Sex	Survived	SexCode
1	Allison, Miss Helen Loraine	1st	2.00	female	0	1
2	Allison, Mr Hudson Joshua Creighton	1st	30.00	male	0	0
3	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	female	0	1

Dwukropek może również zostać użyty do pobrania wszystkich wierszy do pewnego miejsca. W następnym przykładzie pokazałem przykład wycinka obejmującego wiersze od początku do czwartego włącznie.

```

# Pobranie wierszy do czwartego włącznie.
dataframe.iloc[:4]

```

Oto dane po wykonaniu przedstawionego polecenia:

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.00	female	1	1
1	Allison, Miss Helen Loraine	1st	2.00	female	0	1
2	Allison, Mr Hudson Joshua Creighton	1st	30.00	male	0	0
3	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	female	0	1

Ramki danych nie muszą być indeksowane liczbowo. Indeksami może być dowolna wartość unikatowa w poszczególnych wierszach. Na przykład jako indeksu można użyć nazwiska pasażera, a następnie pobierać wiersze za pomocą tego nazwiska.

```

# Zdefiniowanie indeksu.
dataframe = dataframe.set_index(dataframe['Name'])

# Wyświetlenie podanego wiersza.
dataframe.loc['Allen, Miss Elisabeth Walton']

Name      Allen, Miss Elisabeth Walton
PClass    1st
Age       29
Sex       female
Survived   1
SexCode    1
Name: Allen, Miss Elisabeth Walton, dtype: object

```

Analiza

Wszystkie wiersze w obiekcie typu `DataFrame` biblioteki `pandas` muszą mieć indeks o unikatowej wartości. Domyślnie tym indeksem jest liczba całkowita wskazująca położenie danego wiersza w obiekcie. Można jednak zdefiniować zupełnie inny indeks. Może nim być unikatowy ciąg tekstowy lub liczba. Aby pobierać poszczególne wiersze i wycinki, biblioteka `pandas` oferuje dwie wymienione tutaj metody:

- `loc()` okazuje się użyteczna, gdy indeksem obiektu `DataFrame` jest etykieta, czyli na przykład ciąg tekstowy;
- `iloc()` działa przez wyszukanie położenia w obiekcie `DataFrame`. Dlatego też wywołanie `iloc[0]` zwróci pierwszy wiersz niezależnie od tego, jaką postać ma jego indeks.

Warto nabrać wprawy w pracy z obiema omówionymi tutaj metodami, ponieważ bardzo pomagają one podczas oczyszczania danych.

3.4. Pobieranie wierszy na podstawie pewnych warunków

Problem

Chcesz pobrać wiersze ramki danych na podstawie pewnych warunków.

Rozwiązanie

Ten problem można niezwykle łatwo rozwiązać za pomocą biblioteki `pandas`. Spójrz na przykład pokazujący, jak pobrać informacje o wszystkich kobietach znajdujących się na *Titanicu*.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Wyświetlenie dwóch pierwszych wierszy, w których kolumna 'Sex' ma wartość 'female'.
dataframe[dataframe['Sex'] == 'female'].head(2)
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.00	female	1	1
1	Allison, Miss Helen Loraine	1st	2.00	female	0	1

Zatrzymaj się na chwilę i przyjrzyj formatowi użytemu w rozwiązaniu. Mamy tutaj polecenie warunkowe, `dataframe['Sex'] == 'female'`, opakowane konstrukcją `dataframe[]`. W ten sposób

nakazujemy bibliotece pandas pobranie wszystkich wierszy w ramce danych, których wartością `dataframe['Sex']` jest `'female'`. W wyniku otrzymujemy serię wartości boolowskich.

Równie łatwo można zastosować większą liczbę wyrażeń warunkowych. Na przykład zobacz, jak pobrać wiersze wszystkich pasażerów przedstawiające kobiety w wieku przynajmniej 65 lat.

```
# Filtrowanie wierszy.  
dataframe[(dataframe['Sex'] == 'female') & (dataframe['Age'] >= 65)]
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	PClass	Age	Sex	Survived	SexCode
73	Crosby, Mrs Edward Gifford (Catherine Elizabeth...	1st	69.0	female	1	1

Analiza

Warunkowe pobieranie i filtrowanie danych to jedno z podstawowych zadań wykonywanych podczas przygotowywania danych. Bardzo rzadko zależy nam na tym, by otrzymać niezmodyfikowane dane pochodzące ze źródła. Najczęściej będziesz zainteresowany jedynie ich podzbiorem. Na przykład być może chcesz otrzymać informacje o sklepach znajdujących się w podanych województwach lub rekordy pacjentów w określonym wieku.

3.5. Sortowanie wartości

Problem

Musisz posortować ramkę danych według wartości kolumny.

Rozwiązanie

Skorzystaj z oferowanej przez bibliotekę pandas funkcji `sort_values()`.

```
# Wczytanie biblioteki.  
import pandas as pd  
  
# Utworzenie adresu URL.  
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'  
  
# Wczytanie danych.  
dataframe = pd.read_csv(url)  
  
# Sortowanie ramki danych według kolumny Age i wyświetlenie dwóch pierwszych wierszy.  
dataframe.sort_values(by=["Age"]).head(2)
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	PClass	Age	Sex	Survived	SexCode
763	Dean, Miss Elizabeth Gladys (Millvena)	3rd	0.17	female	1	1
751	Danbom, Master Gilbert Sigvard Emanuel	3rd	0.33	male	0	0

Analiza

Podczas analizy danych i ich eksploracji bardzo często użyteczne okazuje się sortowanie ramki danych według określonej kolumny bądź zbioru kolumn. Argument `by` funkcji `sort_values()` pobiera listę kolumn, według których ma być sortowany obiekt `DataFrame`. Następnie operacja sortowania jest przeprowadzana w kolejności nazw kolumn wymienionych na liście.

Domyślnie argument `ascending` ma wartość `True`, więc sortowanie odbywa się w kolejności od najmniejszej do największej wartości. Jeżeli chcesz na przykład najpierw wyświetlić najstarszych pasażerów, argumentowi `ascending` powinienś przypisać wartość `False`.

3.6. Zastępowanie wartości

Problem

Musisz zastąpić wartość w ramce danych.

Rozwiązanie

Oferowana przez bibliotekę `pandas` metoda `replace()` pozwala na bardzo łatwe wyszukiwanie i zastępowanie wartości. Spójrz na przykład pokazujący, jak prosta jest operacja zastąpienia wystąpienia słowa `female` w kolumnie `Sex` wszystkich wierszy słowem `Woman`.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Zastąpienie wartości, wyświetlenie dwóch pierwszych wierszy.
dataframe['Sex'].replace("female", "Woman").head(2)

0    Woman
1    Woman
Name: Sex, dtype: object
```

Można też zastąpić wiele wartości jednocześnie.

```
# Zastąpienie słów "female" i "male" słowami "Woman" i "Man".
dataframe['Sex'].replace(["female", "male"], ["Woman", "Man"]).head(5)
```

```
0    Woman
1    Woman
2     Man
3    Woman
4     Man
Name: Sex, dtype: object
```

Operację wyszukiwania i zastępowania można przeprowadzać w całym obiekcie DataFrame. Wystarczy wskazać całą ramkę danych zamiast pojedynczej kolumny.

```
# Zastąpienie wartości, wyświetlenie dwóch pierwszych wierszy.
dataframe.replace(1, "One").head(2)
```

Oto dane po wykonaniu przedstawionego polecenia:

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.00	female	One	One
1	Allison, Miss Helen Loraine	1st	2.00	female	0	One

Metoda `replace()` akceptuje również wyrażenie regularne.

```
# Zastąpienie wartości, wyświetlenie dwóch pierwszych wierszy.
dataframe.replace(r"1st", "First", regex=True).head(2)
```

Oto dane po wykonaniu przedstawionego polecenia:

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	First	29.00	female	1	1
1	Allison, Miss Helen Loraine	First	2.00	female	0	1

Analiza

Metoda `replace()` to proste, choć jednocześnie oferujące potężne możliwości narzędzie przeznaczone do zastępowania wartości. Pozwala na podanie argumentu w postaci wyrażenia regularnego.

3.7. Zmiana nazwy kolumny

Problem

Chcesz zmienić nazwę kolumny w obiekcie DataFrame biblioteki pandas.

Rozwiązanie

Aby zmienić nazwę kolumny, użyj metody `rename()`.

```
# Wczytanie biblioteki.
import pandas as pd
```

```
# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Zmiana nazwy kolumny, wyświetlenie dwóch pierwszych wierszy.
dataframe.rename(columns={'PClass': 'Passenger Class'}).head(2)
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	Passenger Class	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.0	female	1	1
1	Allison, Miss Helen Loraine	1st	2.0	female	0	1

Metoda `rename()` może przyjąć argument w postaci słownika, za pomocą którego istnieje możliwość jednoczesnej zmiany nazw wielu kolumn.

```
# Zmiana nazwy kolumn, wyświetlenie dwóch pierwszych wierszy.
dataframe.rename(columns={'PClass': 'Passenger Class', 'Sex': 'Gender'}).head(2)
```

Oto dane po wykonaniu przedstawionego polecenia:

	Name	Passenger Class	Age	Gender	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.0	female	1	1
1	Allison, Miss Helen Loraine	1st	2.0	female	0	1

Analiza

Użycie metody `rename()` wraz z argumentem w postaci słownika dla parametru `columns` jest preferowanym rozwiązaniem podczas zmiany nazw kolumn, ponieważ pozwala na przeprowadzenie operacji na dowolnej liczbie kolumn. Jeżeli chcesz jednocześnie zmienić nazwy wszystkich kolumn, ten użyteczny fragment kodu tworzy słownik wraz z kluczami w postaci dotychczasowych nazw kolumn i wartościami w postaci pustych ciągów tekstowych.

```
# Wczytanie biblioteki.
import collections

# Utworzenie słownika.
column_names = collections.defaultdict(str)

# Utworzenie kluczy.
for name in dataframe.columns:
    column_names[name]

# Wyświetlenie słownika.
column_names

defaultdict(str,
            {'Age': '',
             'Name': ''})
```

```
'PClass': '',  
'Sex': '',  
'SexCode': '',  
'Survived': ''})
```

3.8. Znajdowanie wartości minimalnej, maksymalnej, sumy, średniej i liczby elementów w kolumnie

Problem

Chcesz odszukać wartość minimalną, maksymalną, sumę, średnią lub liczbę elementów w kolumnie.

Rozwiązanie

Biblioteka pandas zawiera pewne wbudowane metody, `min()`, `max()`, `mean()`, `sum()` i `count()`, przeznaczone do obliczania niektórych najczęściej używanych danych statystycznych.

```
# Wczytanie biblioteki.  
import pandas as pd  
  
# Utworzenie adresu URL.  
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'  
  
# Wczytanie danych.  
dataframe = pd.read_csv(url)  
  
# Obliczenie danych statystycznych.  
print('Maksimum:', dataframe['Age'].max())  
print('Minimum:', dataframe['Age'].min())  
print('Średnia:', dataframe['Age'].mean())  
print('Suma:', dataframe['Age'].sum())  
print('Liczba elementów:', dataframe['Age'].count())  
  
Maksimum: 71.0  
Minimum: 0.17  
Średnia: 30.397989417989415  
Suma: 22980.879999999997  
Liczba elementów: 756
```

Analiza

Poza danymi statystycznymi, które wykorzystałem w omawianym rozwiązaniu, biblioteka pandas oferuje również znalezienie wariancji (`var()`), odchylenia standardowego (`std()`), kurtozy (`kurt()`), współczynnika skośności (`skew()`), błędu standardowego średniej arytmetycznej (`sem()`), dominanty (`mode()`), mediany (`median()`), liczby wystąpień poszczególnych wartości oraz wielu innych wartości.

Co więcej, wymienione metody można stosować dla całego obiektu `DataFrame`.

```
# Wyświetlenie liczby elementów w kolumnach.  
dataframe.count()  
  
Name      1313
```

```
PClass      1313
Age         756
Sex         1313
Survived    1313
SexCode     1313
dtype: int64
```

3.9. Znajdowanie unikatowych wartości

Problem

Chcesz pobrać wszystkie unikatowe wartości w kolumnie.

Rozwiązanie

Za pomocą metody `unique()` można wyświetlić tablicę wszystkich unikatowych wartości w kolumnie.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Pobranie unikatowych wartości.
dataframe['Sex'].unique()

array(['female', 'male'], dtype=object)
```

Z kolei metoda `value_counts()` wyświetla wszystkie unikatowe wartości i podaje liczbę wystąpień każdej z nich.

```
# Wyświetlenie liczby elementów w kolumnach.
dataframe['Sex'].value_counts()

male      851
female    462
Name: Sex, dtype: int64
```

Analiza

Metody `unique()` i `value_counts()` okazują się użyteczne podczas przeprowadzania operacji i analizowania kolumn kategoryzujących. Bardzo często w takich kolumnach znajdują się klasy wymagające obsługi na etapie przygotowywania danych. Na przykład w zbiorze danych o pasażerach Titanica kolumna `PClass` określa, którą klasą podróżowała dana osoba. Na Titanicu były trzy klasy; po użyciu metody `value_counts()` będzie można dostrzec problem.

```
# Wyświetlenie liczby elementów w kolumnach.
dataframe['PClass'].value_counts()
```

```
3rd    711
1st    322
2nd    279
*        1
Name: PClass, dtype: int64
```

Wprawdzie zgodnie z oczekiwaniami praktycznie wszyscy pasażerowie należeli do jednej z tych trzech klas, ale jeden pasażer miał klasę *. Do obsługi takich problemów istnieje wiele strategii, które przedstawię w rozdziale 5. W tym miejscu wystarczy wiedzieć, że te „dodatkowe” klasy są często spotykane w danych kategoryzujących i nie powinny być ignorowane.

Jeżeli zachodzi potrzeba ustalenia po prostu liczby unikatowych wartości, wówczas można skorzystać z metody `nunique()`.

```
# Wyświetlenie liczby unikatowych wartości.
dataframe['PClass'].nunique()

4
```

3.10. Obsługa brakujących wartości

Problem

Chcesz ustalić, czy w obiekcie `DataFrame` brakuje jakichkolwiek wartości.

Rozwiązanie

Metody `isnull()` i `notnull()` zwracają wartość boolowską, określającą, czy brakuje danej informacji.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Pobranie i wyświetlenie dwóch wierszy, w których brakuje wartości.
dataframe[dataframe['Age'].isnull()].head(2)
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	PClass	Age	Sex	Survived	SexCode
12	Aubert, Mrs Leontine Pauline	1st	NaN	female	1	1
13	Barkworth, Mr Algernon H	1st	NaN	male	1	0

Analiza

Brakujące wartości to często pojawiający się problem podczas przygotowywania danych, a wielu programistów nie zdaje sobie sprawy z trudności podczas pracy z niekompletnymi danymi. Biblioteka pandas używa znanej z NumPy wartości NaN (ang. *not a number* — to nie jest liczba) do zaznaczenia braku wartości. Trzeba w tym miejscu koniecznie dodać, że wartość NaN nie została w pełni natywnie zaimplementowana w bibliotece pandas. Dlatego też jeśli chcesz zastąpić wszystkie ciągi tekstowe małe wartością NaN, wynikiem takiej operacji będzie błąd.

```
# Próba zastąpienia wartości przez NaN.
```

```
dataframe['Sex'] = dataframe['Sex'].replace('male', NaN)
```

```
-----  
NameError
```

```
Traceback (most recent call last)
```

```
<ipython-input-7-5682d714f87d> in <module>()  
1 # Próba zastąpienia wartości przez NaN.
```

```
----> 2 dataframe['Sex'] = dataframe['Sex'].replace('male', NaN)
```

```
NameError: name 'NaN' is not defined  
-----
```

Aby zapewnić pełną funkcjonalność NaN, najpierw trzeba zaimportować bibliotekę NumPy.

```
# Wczytanie biblioteki.
```

```
import numpy as np
```

```
# Zastąpienie wartości przez NaN.
```

```
dataframe['Sex'] = dataframe['Sex'].replace('male', np.nan)
```

Bardzo często zdarza się, że do oznaczenia brakującej obserwacji w zbiorze danych używa się pewnej wartości, na przykład NONE, -999 lub kropki. Metoda `read_csv()` biblioteki pandas zawiera parametr pozwalający na określenie wartości używanych do wskazania brakujących wartości.

```
# Wczytanie danych, przypisanie brakujących wartości.
```

```
dataframe = pd.read_csv(url, na_values=[np.nan, 'NONE', -999])
```

Istnieje również możliwość użycia funkcji `fillna()` do przypisania brakujących wartości kolumny. W kolejnym fragmencie kodu pokazałem, jak można użyć funkcji `isna()` do znalezienia wierszy, w których wartością kolumny Age jest null, a następnie zastąpić ją wartością przedstawiającą średni wiek pasażera Titanica.

```
# Pobranie pojedynczego rekordu, w którym brakuje wartości kolumny Age.
```

```
null_entry = dataframe[dataframe["Age"].isna()].head(1)
```

```
print(null_entry)
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	PClass	Age	Sex	Survived	SexCode
12	Aubert, Mrs Leontine Pauline	1st	NaN	female	1	1

```
# Przypisanie kolumnie o wartości null wartości przedstawiającej średni wiek pasażera Titanica.
```

```
null_entry.fillna(dataframe["Age"].mean())
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	PClass	Age	Sex	Survived	SexCode
12	Aubert, Mrs Leontine Pauline	1st	30.397989	female	1	1

3.11. Usuwanie kolumn

Problem

Chcesz usunąć kolumnę z obiektu DataFrame.

Rozwiązanie

Najlepszym sposobem na usunięcie kolumny jest użycie metody `drop()` wraz z parametrem `axis=1` (np. określającym oś kolumny).

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Usunięcie kolumny.
dataframe.drop('Age', axis=1).head(2)
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	PClass	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	female	1	1
1	Allison, Miss Helen Loraine	1st	female	0	1

Lista nazw kolumn może zostać użyta jako argument główny, co pozwala na jednoczesne usunięcie wielu kolumn.

```
# Usunięcie kolumn.
dataframe.drop(['Age', 'Sex'], axis=1).head(2)
```

Oto dane po wykonaniu przedstawionego polecenia:

	Name	PClass	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	1	1
1	Allison, Miss Helen Loraine	1st	0	1

Jeżeli kolumna nie ma nazwy, co się czasami zdarza, można ją usunąć przez podanie indeksu kolumny za pomocą atrybutu `dataframe.columns`, jak pokazałem w kolejnym poleceniu.

```
# Usunięcie kolumny.  
dataframe.drop(dataframe.columns[1], axis=1).head(2)
```

Oto dane po wykonaniu przedstawionego polecenia:

	Name	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	29.0	female	1	1
1	Allison, Miss Helen Loraine	2.0	female	0	1

Analiza

Metoda `drop()` jest przeznaczona do usunięcia kolumny. Alternatywną metodą jest `del`, na przykład `del dataframe['Age']`, która działa w większości przypadków, ale nie jest zalecana ze względu na sposób jej wywoływania w bibliotece pandas (związane z tym szczegóły wykraczają poza zakres tematyczny książki).

Szczerze odradzam używanie argumentu `inplace=True`. Wiele metod biblioteki pandas zawiera parametr `inplace`, którego wartość `True` oznacza bezpośrednią edycję obiektu `DataFrame`. To może prowadzić do problemów w bardziej złożonych potokach przetwarzania danych, ponieważ obiekt `DataFrame` będzie traktowany jako modyfikowalny (formalnie rzecz biorąc, należy go za taki uznać). Gorąco zachęcam do używania `DataFrame` jak obiektu niemodyfikowalnego. Spójrz na przedstawiony tutaj przykład.

```
# Utworzenie nowego obiektu typu DataFrame.  
dataframe_name_dropped = dataframe.drop(dataframe.columns[0], axis=1)
```

W tym przykładzie nie mamy do czynienia ze zmianą obiektu typu `DataFrame`, ale z utworzeniem nowego obiektu tego typu będącego alternatywną wersją `dataframe` o nazwie `dataframe_name_dropped`. Jeżeli będziesz traktować `DataFrame` jako obiekt niemodyfikowalny, unikniesz wielu kłopotów.

3.12. Usuwanie wiersza

Problem

Chcesz usunąć jeden lub większą liczbę wierszy z obiektu `DataFrame`.

Rozwiązanie

Użyj wyrażenia boolowskiego do utworzenia nowego obiektu typu `DataFrame` niezawierającego wierszy, które chciałeś usunąć.

```
# Wczytanie biblioteki.  
import pandas as pd
```

```
# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Usunięcie wierszy, wyświetlenie dwóch pierwszych wierszy danych wyjściowych.
dataframe[dataframe['Sex'] != 'male'].head(3)
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.0	female	1	1
1	Allison, Miss Helen Loraine	1st	2.0	female	0	1
3	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	female	0	1

Analiza

Mimo że formalnie rzecz biorąc, możliwe jest użycie metody `drop()`, na przykład `dataframe.drop([0, 1], axis=0)`, aby usunąć dwa pierwsze wiersze, to jednak znacznie praktyczniejszym rozwiązaniem będzie umieszczenie wyrażenia boolowskiego w wywołaniu `dataframe[]`. Dzięki temu można wykorzystać potężne możliwości wyrażen boolowskich do usunięcia pojedynczego wiersza lub (co bardziej prawdopodobne) jednocześnie wielu wierszy.

Wyrażenia boolowskiego można używać, aby w łatwy sposób usunąć jeden wiersz dopasowany dzięki unikatowej wartości.

```
# Usunięcie wiersza, wyświetlenie dwóch pierwszych wierszy danych wyjściowych.
dataframe[dataframe['Name'] != 'Allison, Miss Helen Loraine'].head(2)
```

Oto dane po wykonaniu przedstawionego polecenia:

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.0	female	1	1
2	Allison, Mr Hudson Joshua Creighton	1st	30.0	male	0	0

Pojedynczy wiersz można również usunąć, podając jego indeks, jak pokazałem w kolejnym poleceniu.

```
# Usunięcie wiersza, wyświetlenie dwóch pierwszych wierszy danych wyjściowych.
dataframe[dataframe.index != 0].head(2)
```

Oto dane po wykonaniu przedstawionego polecenia:

	Name	PClass	Age	Sex	Survived	SexCode
1	Allison, Miss Helen Loraine	1st	2.0	female	0	1
2	Allison, Mr Hudson Joshua Creighton	1st	30.0	male	0	0

3.13. Usuwanie powielonych wierszy

Problem

Z obiektu `DataFrame` chcesz usunąć powielone wiersze.

Rozwiązanie

Możesz skorzystać z metody `drop_duplicates()`, ale powinieneś pamiętać o dostępnych parametrach.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Usunięcie powielonych wierszy, wyświetlenie dwóch pierwszych wierszy danych wyjściowych.
dataframe.drop_duplicates().head(2)
```

Oto dane po wykonaniu przedstawionego kodu:

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.0	female	1	1
1	Allison, Miss Helen Loraine	1st	2.0	female	0	1

Analiza

Uważny czytelnik dostrzeże, że przedstawione tutaj rozwiązanie w rzeczywistości nie spowodowało usunięcia żadnych wierszy.

```
# Wyświetlenie liczby wierszy.
print("Liczba wierszy w początkowym obiekcie DataFrame:", len(dataframe))
print("Liczba wierszy po przeprowadzeniu operacji:", len(dataframe.drop_duplicates()))

Liczba wierszy w początkowym obiekcie DataFrame: 1313
Liczba wierszy po przeprowadzeniu operacji: 1313
```

Domyślnie działanie metody `drop_duplicates()` polega na usunięciu jedynie wierszy idealnie dopasowanych we wszystkich kolumnach. W takim przypadku każdy wiersz w użytym tutaj obiekcie typu `DataFrame` jest faktycznie unikatowy. Jednak bardzo często zachodzi potrzeba sprawdzenia pod kątem powielonych wierszy tylko podzbioru kolumn. Wówczas należy skorzystać z parametru `subset`.

```
# Usunięcie powielonych wierszy.
dataframe.drop_duplicates(subset=['Sex'])
```

Oto dane po wykonaniu przedstawionego polecenia:

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.0	female	1	1
2	Allison, Mr Hudson Joshua Creighton	1st	30.0	male	0	0

Spójrz uważnie na wygenerowane dane wyjściowe: metoda `drop_duplicates()` uznaje za duplikaty przeznaczone do usunięcia dwa dowolne wiersze o takiej samej wartości kolumny `Sex`. Po wykonaniu tej operacji mamy obiekt typu `DataFrame`, zawierający tylko dwa wiersze przedstawiające kobietę i mężczyznę. Być może zadajesz sobie pytanie, dlaczego metoda `drop_duplicates()` pozostawiła akurat te dwa wiersze. Odpowiedź okazuje się prosta: metoda pozostawia pierwsze wystąpienie powielonego wiersza i usuwa pozostałe. To zachowanie można kontrolować za pomocą parametru `keep`, jak pokazałem w kolejnym przykładzie.

```
# Usunięcie powielonych wierszy.  
dataframe.drop_duplicates(subset=['Sex'], keep='last')
```

Oto dane po wykonaniu przedstawionego polecenia:

	Name	PClass	Age	Sex	Survived	SexCode
1307	Zabour, Miss Tamini	3rd	NaN	female	0	1
1312	Zimmerman, Leo	3rd	29.0	male	0	0

Istnieje metoda o nazwie `duplicated()`, podobna do omawianej tutaj `drop_duplicates()`. Wartością zwrótną metody `duplicated()` jest seria wartości boolowskich, wskazujących, czy wiersz jest powielony. Użycie tej metody jest dobrym rozwiązaniem, jeśli nie chcesz usunąć powielonych wierszy.

```
dataframe.duplicated()  
  
0      False  
1      False  
2      False  
3      False  
4      False  
...  
1308   False  
1309   False  
1310   False  
1311   False  
1312   False  
Length: 1313, dtype: bool
```

3.14. Grupowanie wierszy według wartości

Problem

Chcesz pogrupować poszczególne wiersze według pewnej wartości współdzielonej.

Rozwiązanie

Metoda `groupby()` to jedna z najpotężniejszych funkcji oferowanych przez bibliotekę `pandas`.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Grupowanie wierszy według wartości kolumny 'Sex'
# i obliczenie średniej w poszczególnych grupach.
dataframe.groupby('Sex').mean(numeric_only=True)
```

Oto dane po wykonaniu przedstawionego kodu:

Sex	Age	Survived	SexCode
Female	29.396424	0.666667	1.0
Male	31.014338	0.166863	0.0

Analiza

Użycie metody `groupby()` powoduje, że dane zaczynają nabierać kształtu. Bardzo często zdarza się, że chcemy grupować poszczególne wiersze obiektu `DataFrame` przedstawiające osobę lub zdarzenie według pewnego kryterium, aby następnie wygenerować wybrane dane statystyczne. Na przykład wyobraź sobie istnienie obiektu typu `DataFrame`, którego wiersze przedstawiają wartość pojedynczych transakcji we wszystkich lokalach danej sieci restauracji, a Twoim zadaniem jest obliczenie wartości wszystkich transakcji w poszczególnych lokalach. To zadanie można wykonać poprzez zgrupowanie wierszy określających poszczególne restauracje, a następnie obliczenie sumy całej grupy.

Początkujący mają tendencję do używania metody `groupby()` w pokazany tutaj sposób, a następnie są zdziwieni otrzymanym wynikiem.

```
# Grupowanie wierszy.
dataframe.groupby('Sex')

<pandas.core.groupby.DataFrameGroupBy object at 0x10efacf28>
```

Dlaczego przedstawione polecenie nie zwróciło bardziej użytecznego wyniku? Wywołanie metody `groupby()` trzeba po prostu połączyć z pewną operacją, która ma być przeprowadzona na każdej grupie, na przykład z obliczeniem zagregowanych danych statystycznych, takich jak średnia, mediana, suma itd. Kiedy mówi się o grupowaniu, często słyszy się określenie „grupowanie według płci”, ale to nie do końca odpowiednie stwierdzenie. Aby grupowanie było użyteczne, konieczne jest grupowanie pewnych wartości i następnie przeprowadzenie operacji na poszczególnych grupach.

```
# Grupowanie wierszy, a następnie ich zliczenie.
dataframe.groupby('Survived')['Name'].count()
```

```
Survived
0      863
1      450
Name: Name, dtype: int64
```

Zwróć uwagę na dodanie `Name` po wywołaniu metody `groupby()`. Wynika to z tego, że podsumowanie określonych danych statystycznych ma sens jedynie dla pewnych typów danych. Na przykład obliczanie średniego wieku według płci ma sens, natomiast obliczanie sumy lat według płci nie ma żadnego sensu. W omawianym przypadku zostały zgrupowane dane dotyczące osób, które przeżyły i nie przeżyły katastrofy Titanica, a następnie zliczana jest liczba pasażerów w poszczególnych grupach.

Istnieje również możliwość grupowania według pierwszej kolumny, a następnie według drugiej, jak pokazałem w kolejnym fragmencie kodu.

```
# Grupowanie wierszy, obliczenie średniej.
dataframe.groupby(['Sex', 'Survived'])['Age'].mean()

Sex      Survived
female  0          24.901408
        1          30.867143
male    0          32.320780
        1          25.951875
Name: Age, dtype: float64
```

3.15. Grupowanie wierszy według czasu

Problem

Musisz pogrupować pojedyncze wiersze według czasu.

Rozwiązanie

Użyj oferowanej przez bibliotekę `pandas` metody `resample()` do grupowania wierszy według czasu.

```
# Wczytanie bibliotek.
import pandas as pd
import numpy as np

# Utworzenie przedziału czasu.
time_index = pd.date_range('06/06/2017', periods=100000, freq='30S')

# Utworzenie obiektu typu DataFrame.
dataframe = pd.DataFrame(index=time_index)

# Utworzenie kolumny losowo wybranych wartości.
dataframe['Sale_Amount'] = np.random.randint(1, 10, 100000)

# Grupowanie wierszy według tygodnia, obliczenie sumy dla danego tygodnia.
dataframe.resample('W').sum()
```

Oto dane po wykonaniu przedstawionego kodu:

	Sale_Amount
2017-06-11	86423
2017-06-18	101045
2017-06-25	100867
2017-07-02	100894
2017-07-09	100438
2017-07-16	10297

Analiza

Wykorzystywany dość często w rozdziale zbiór danych dotyczących pasażerów Titanica nie ma kolumny z wartościami czasu. Dlatego też w tej recepturze wygenerowałem prosty obiekt typu DataFrame, którego wiersze przedstawiają pojedyncze transakcje. Znamy datę i godzinę oraz wyrażoną w złotych wartość każdej transakcji. (Te dane nie są zbyt rzeczywiste, ponieważ transakcje odbyły się w odstępach 30 sekund i wartość każdej z nich jest kwotą wyrażoną w pełnych złotych, ale na potrzeby omawianego tutaj rozwiązania jest to w zupełności wystarczające).

Niezmodyfikowane dane w trzech pierwszych wierszach przedstawiają się następująco:

```
# Wyświetlenie trzech pierwszych wierszy.  
dataframe.head(3)
```

Oto dane po wykonaniu przedstawionego polecenia:

	Sale_Amount
2017-06-06 00:00:00	7
2017-06-06 00:00:30	2
2017-06-06 00:01:00	7

Zwróć uwagę na to, że data i godzina poszczególnych transakcji to indeks w obiekcie typu DataFrame. Metoda `resample()` wymaga indeksu dla wartości przedstawiających datę i godzinę.

Za pomocą tej metody można grupować wiersze według różnych przedziałów czasu, a następnie przeprowadzać pewne obliczenia w tych grupach, jak pokazałem w dwóch kolejnych poleceniach.

```
# Grupowanie według dwóch tygodni, obliczenie średniej.  
dataframe.resample('2W').mean()
```

Oto dane po wykonaniu przedstawionego polecenia:

	Sale_Amount
2017-06-11	5.001331
2017-06-25	5.007738

	Sale_Amount
2017-07-09	4.993353
2017-07-23	4.950481

```
# Grupowanie według miesiąca, obliczenie liczby wierszy.
dataframe.resample('M').count()
```

Oto dane po wykonaniu przedstawionego polecenia:

	Sale_Amount
2017-06-30	72000
2017-07-31	28000

Zwróć uwagę na wygenerowanie dwóch elementów danych wyjściowych indeksu daty i godziny, pomimo że grupowanie odbywa się według — odpowiednio — tygodni i miesięcy. Domyślnie metoda `resample()` zwraca etykietę prawej „krawędzi” (ostatnia etykieta) grupy daty i godziny. To zachowanie można kontrolować za pomocą parametru `label`.

```
# Grupowanie według miesiąca, obliczenie liczby wierszy.
dataframe.resample('M', label='left').count()
```

Oto dane po wykonaniu przedstawionego polecenia:

	Sale_Amount
2017-05-31	72000
2017-06-30	28000

Zobacz również

- Lista aliasów biblioteki pandas związanych z wartościami daty i godziny na stronie https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html.

3.16. Agregowanie operacji i danych statystycznych

Problem

Chcesz przeprowadzić agregację operacji w poszczególnych kolumnach (lub w zbiorze kolumn) ramki danych.

Rozwiązanie

Skorzystaj z oferowanej przez bibliotekę pandas metody `agg()`. Zobacz, jak łatwo można pobrać wartość minimalną każdej kolumny.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Pobranie wartości minimalnej każdej kolumny.
dataframe.agg("min")

Name      Abbing, Mr Anthony
PClass    *
Age        0.17
Sex        female
Survived   0
SexCode    0
dtype: object
```

Czasami zachodzi potrzeba zastosowania określonych funkcji dla pewnego zbioru kolumn.

```
# Wartość średnia kolumny Age oraz wartości minimalna i maksymalna kolumny Sex.
dataframe.agg({"Age":["mean"], "SexCode":["min", "max"]})
```

Oto dane po wykonaniu przedstawionego kodu:

	Age	SexCode
mean	30.397989	NaN
min	NaN	0.0
max	NaN	1.0

Funkcje agregacji można stosować także dla grup, aby w ten sposób otrzymać znacznie dokładniejsze i bardziej opisowe dane statystyczne.

```
# Liczba pasażerów w poszczególnych klasach, którzy przeżyli i nie przeżyli katastrofy Titanica.
dataframe.groupby(
    ["PClass", "Survived"]).agg({"Survived":["count"]})
).reset_index()
```

Oto dane po wykonaniu przedstawionego kodu:

	PClass	Survived	Count
0	*	0	1
1	1st	0	129
2	1st	1	193
3	2nd	0	160
4	2nd	1	119
5	3rd	0	573
6	3rd	1	138

Analiza

Funkcje agregacji okazują się szczególnie użyteczne podczas eksploracyjnej analizy danych, ponieważ pozwalają poznać informacje o różnych podpopulacjach danych i relacjach zachodzących między zmiennymi. Dzięki grupowaniu danych i zastosowaniu agregacji danych statystycznych można poznać wzorce w danych, które okażą się przydatne podczas wykonywania zadań związanych z uczeniem maszynowym bądź w trakcie procesu wyodrębniania cech. Wprawdzie wykresy graficzne również są pomocne, ale często wystarczające będzie posiadanie takich konkretnych, opisowych danych statystycznych, które pomogą w jeszcze lepszym zrozumieniu danych.

Zobacz również

- Dokumentacja funkcji `agg()` biblioteki `pandas`
<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.agg.html>.

3.17. Iterowanie przez kolumnę

Problem

Chcesz przeprowadzić iterację przez wszystkie elementy kolumny i wykonać na nich jakieś działanie.

Rozwiązanie

Kolumnę w strukturze danych biblioteki `pandas` można potraktować jak każdą inną sekwencję Pythona.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Wyświetlenie wielkimi literami imienia i nazwiska dwóch pierwszych pasażerów.
for name in dataframe['Name'][0:2]:
    print(name.upper())

ALLEN, MISS ELISABETH WALTON
ALLISON, MISS HELEN LORAINE
```

Analiza

Poza pętlami (często określanymi mianem pętli `for`) można również użyć listy składanej, jak pokazałem w kolejnym przykładzie.

```
# Wyświetlenie wielkimi literami imienia i nazwiska dwóch pierwszych pasażerów.
[name.upper() for name in dataframe['Name'][0:2]]

['ALLEN, MISS ELISABETH WALTON', 'ALLISON, MISS HELEN LORAINE']
```

Wprawdzie rozwiązaniem awaryjnym jest pętla `for`, ale bardziej zgodnym ze stylem „Pythonic” podejściem jest użycie oferowanej przez bibliotekę `pandas` metody `apply()`, co przedstawię w następnej recepturze.

3.18. Wywoływanie funkcji dla wszystkich elementów kolumny

Problem

Chcesz wywołać pewną funkcję dla wszystkich elementów kolumny.

Rozwiązanie

Użyj metody `apply()`, aby dla każdego elementu kolumny wywołać funkcję wbudowaną lub niestandardową.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Utworzenie funkcji.
def uppercase(x):
    return x.upper()

# Wywołanie funkcji, wyświetlenie dwóch pierwszych wierszy.
dataframe['Name'].apply(uppercase)[0:2]

0    ALLEN, MISS ELISABETH WALTON
1    ALLISON, MISS HELEN LORAINÉ
Name: Name, dtype: object
```

Analiza

Metoda `apply()` sprawdza się doskonale podczas oczyszczania i przygotowywania danych. Bardzo często tworzy się funkcje przeznaczone do przeprowadzania pewnych użytecznych operacji (rozdzielenia imienia i nazwiska, konwersji ciągu tekstowego na liczbę itd.), a następnie wywołuje się tę funkcję dla każdego elementu kolumny.

3.19. Wywoływanie funkcji dla grupy

Problem

Masz utworzone za pomocą metody `groupby()` grupy wierszy i chcesz w nich wywołać pewną funkcję.

Rozwiązanie

Najlepszym rozwiązaniem będzie połączenie wywołań metod `groupby()` i `apply()`.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie adresu URL.
url = 'https://raw.githubusercontent.com/chrisalbon/sim_data/master/titanic.csv'

# Wczytanie danych.
dataframe = pd.read_csv(url)

# Grupowanie wierszy, wywołanie funkcji w grupach.
dataframe.groupby('Sex').apply(lambda x: x.count())
```

Oto dane po wykonaniu przedstawionego kodu:

Sex	Name	PClass	Age	Sex	Survived	SexCode
female	462	462	288	462	462	462
male	851	851	468	851	851	851

Analiza

W recepturze 3.18 wspomniałem o metodzie `apply()`. Ta metoda okazuje się szczególnie użyteczna, gdy chcesz wywołać pewną funkcję w grupie elementów. Dzięki połączeniu wywołań metod `groupby()` i `apply()` można wygenerować niestandardowe dane statystyczne lub wykonywać dowolne funkcje w poszczególnych grupach.

3.20. Konkatenacja obiektów typu DataFrame

Problem

Chcesz przeprowadzić konkatenację dwóch obiektów typu `DataFrame`.

Rozwiązanie

Do przeprowadzenia konkatenacji obiektów typu `DataFrame` na podstawie wierszy użyj metody `concat()` wraz z argumentem `axis=0`.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie obiektu typu DataFrame.
data_a = {'id': ['1', '2', '3'],
          'first': ['Alex', 'Amy', 'Allen'],
          'last': ['Anderson', 'Ackerman', 'Ali']}
dataframe_a = pd.DataFrame(data_a, columns = ['id', 'first', 'last'])

# Utworzenie obiektu typu DataFrame.
data_b = {'id': ['4', '5', '6'],
          'first': ['Billy', 'Brian', 'Bran'],
          'last': ['Bonder', 'Black', 'Balwner']}
dataframe_b = pd.DataFrame(data_b, columns = ['id', 'first', 'last'])

# Konkatencja obiektów DataFrame na podstawie wierszy.
pd.concat([dataframe_a, dataframe_b], axis=0)
```

Oto dane po wykonaniu przedstawionego kodu:

	id	first	last
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner

Konkatencja na podstawie kolumny odbywa się po użyciu argumentu `axis=1` w wywołaniu metody `concat()`.

```
# Konkatencja obiektów DataFrame na podstawie kolumn.
pd.concat([dataframe_a, dataframe_b], axis=1)
```

Oto dane po wykonaniu przedstawionego kodu:

	id	first	last	id	first	last
0	1	Alex	Anderson	4	Billy	Bonder
1	2	Amy	Ackerman	5	Brian	Black
2	3	Allen	Ali	6	Bran	Balwner

Analiza

Konkatencja to nie jest słowo, które zbyt często pada w informatyce i programowaniu. Dlatego też nie przejmuj się, jeśli wcześniej go nie słyszałeś. Nieformalna definicja **konkatenacji** to połączenie dwóch obiektów. W omawianym tutaj rozwiązaniu pokazałem połączenie dwóch małych obiektów typu `DataFrame` za pomocą parametru `axis`, wskazującego, czy dwa obiekty mają być ułożone jeden na drugim, czy obok siebie.

3.21. Złączanie obiektów typu DataFrame

Problem

Chcesz złączyć ze sobą dwa obiekty typu DataFrame.

Rozwiązanie

Do przeprowadzenia złączenia wewnętrznego należy użyć metody `merge()` wraz z parametrem `on` wskazującym kolumnę, na podstawie której zostanie przeprowadzona operacja.

```
# Wczytanie biblioteki.
import pandas as pd

# Utworzenie obiektu typu DataFrame.
employee_data = {'employee_id': ['1', '2', '3', '4'],
                  'name': ['Amy Jones', 'Allen Keys', 'Alice Bees',
                           'Tim Horton']}
dataframe_employees = pd.DataFrame(employee_data, columns = ['employee_id',
                                                            'name'])

# Utworzenie obiektu typu DataFrame.
sales_data = {'employee_id': ['3', '4', '5', '6'],
              'total_sales': [23456, 2512, 2345, 1455]}
dataframe_sales = pd.DataFrame(sales_data, columns = ['employee_id',
                                                      'total_sales'])

# Złączenie obiektów typu DataFrame.
pd.merge(dataframe_employees, dataframe_sales, on='employee_id')
```

Oto dane po wykonaniu przedstawionego kodu:

	employee_id	name	total_sales
0	3	Alice Bees	23456
1	4	Tim Horton	2512

Domyślnie metoda `merge()` przeprowadza złączenie wewnętrzne. Jeżeli chcesz przeprowadzić złączenie zewnętrzne, możesz to określić za pomocą parametru `how`.

```
# Złączenie obiektów typu DataFrame.
pd.merge(dataframe_employees, dataframe_sales, on='employee_id', how='outer')
```

Oto dane po wykonaniu przedstawionego polecenia:

	employee_id	name	total_sales
0	1	Amy Jones	NaN
1	2	Allen Keys	NaN
2	3	Alice Bees	23456.0
3	4	Tim Horton	2512.0

	employee_id	name	total_sales
4	5	NaN	2345.0
5	6	NaN	1455.0

Ten sam parametr może zostać użyty do określenia złączenia lewego i prawego.

```
# Złączenie obiektów typu DataFrame.
pd.merge(dataframe_employees, dataframe_sales, on='employee_id', how='left')
```

Oto dane po wykonaniu przedstawionego polecenia:

	employee_id	name	total_sales
0	1	Amy Jones	NaN
1	2	Allen Keys	NaN
2	3	Alice Bees	23456.0
3	4	Tim Horton	2512.0

Istnieje również możliwość podania w poszczególnych obiektach typu DataFrame nazwy kolumny, na podstawie której odbędzie się łączenie.

```
# Złączenie obiektów typu DataFrame.
pd.merge(dataframe_employees,
         dataframe_sales,
         left_on='employee_id',
         right_on='employee_id')
```

Jeżeli zamiast złączenia na podstawie dwóch kolumn chcesz przeprowadzić złączenie na podstawie indeksów poszczególnych obiektów typu DataFrame, parametry `left_on` i `right_on` możesz zastąpić parametrami `right_index=True` i `left_index=True`.

Oto dane po wykonaniu przedstawionego kodu:

	employee_id	name	total_sales
0	3	Alice Bees	23456
1	4	Tim Horton	2512

Analiza

Bardzo często dane konieczne do użycia są skomplikowane i nie zawsze będą dostępne w postaci jednego bloku. W rzeczywistych projektach zwykle mamy oddzielne zbiory danych pochodzące z wielu plików bądź zapytań do różnych baz danych. Aby zebrać razem wszystkie dane, można najpierw umieścić w oddzielnych obiektach typu DataFrame dane pochodzące z poszczególnych źródeł, a następnie połączyć je ze sobą w jeden obiekt DataFrame.

Ten proces powinien być znany programistom, którzy mają doświadczenie w pracy z SQL, czyli popularnym językiem zapytań pozwalającym między innymi na operacje łączenia danych (w SQL są one określane mianem **złączeń**). Wprowadzicie konkretne parametry wykorzystywane przez bibliotekę

pandas będą inne, ale ogólna zasada działania złączenia jest taka sama jak w innych narzędziach i językach programowania.

W trakcie operacji złączenia należy zwrócić uwagę na trzy aspekty. Po pierwsze, konieczne jest podanie dwóch obiektów typu `DataFrame` przeznaczonych do złączenia. W omawianym rozwiązaniu mają one nazwy `dataframe_employees` i `dataframe_sales`. Po drugie, trzeba podać nazwę kolumny (lub kolumn), na podstawie której zostanie przeprowadzone złączenie. To będzie kolumna, której wartość jest współdzielona przez dwa obiekty `DataFrame`. Na przykład w omawianym przykładzie oba obiekty mają kolumnę o nazwie `employee_id`. Aby połączyć oba obiekty, trzeba w nich dopasować wartości wymienionej kolumny. Jeżeli obie kolumny mają tę samą nazwę, można użyć parametru `on`. Natomiast w przeciwnym razie należy skorzystać z parametrów `left_on` lub `right_on`.

Który obiekt `DataFrame` jest uznawany za lewy, a który za prawy? To proste — lewym jest obiekt `DataFrame` podany jako pierwszy w wywołaniu metody `merge()`, natomiast prawym jest obiekt `DataFrame` podany jako drugi. Ta koncepcja pojawia się ponownie w następnym zestawie parametrów.

Trzeci aspekt złączenia jest jednocześnie najtrudniejszy — to konieczność wskazania typu przeprowadzanej operacji złączenia. Do określenia typu używamy parametru `how`. Metoda `merge()` obsługuje cztery podstawowe typy złączeń.

Wewnętrzne

Zwracane są jedynie wiersze dopasowane w obu obiektach `DataFrame`. W omawianym przykładzie zwrócony będzie każdy wiersz, którego wartość `employee_id` występuje zarówno w obiekcie `dataframe_employees`, jak i `dataframe_sales`.

Zewnętrzne

Zwracane są wszystkie wiersze w obu obiektach `DataFrame`. Jeżeli wiersz istnieje tylko w jednym obiekcie, wówczas brakujące wartości są zastępowane przez `NaN`. W omawianym przykładzie zwrócone będą wszystkie wiersze znajdujące się w obiektach `dataframe_employees` i `dataframe_sales`.

Lewe

Z lewego obiektu `DataFrame` zwracane są wszystkie wiersze, natomiast z prawego tylko te, które zostały dopasowane do wierszy w lewym obiekcie. Brakujące wartości są zastępowane przez `NaN`. W omawianym przykładzie zwrócone będą wszystkie wiersze obiektu `dataframe_employees`, zaś z obiektu `dataframe_sales` tylko te, których wartość `employee_id` została dopasowana do wartości tej samej kolumny w `dataframe_employees`.

Prawe

Z prawego obiektu `DataFrame` zwracane są wszystkie wiersze, natomiast z lewego tylko te, które zostały dopasowane do wierszy w prawym obiekcie. Brakujące wartości są zastępowane przez `NaN`. W omawianym przykładzie zwrócone będą wszystkie wiersze obiektu `dataframe_sales`, zaś z obiektu `dataframe_employees` tylko te, których wartość `employee_id` została dopasowana do wartości tej samej kolumny w `dataframe_sales`.

Nie przejmuj się, jeżeli to wyjaśnienie nie jest do końca jasne. Zachęcam Cię do eksperymentów z parametrem `how` w kodzie i obserwacji efektu, jaki ten parametr ma na dane wyjściowe zwracane przez metodę `merge()`.

Zobacz również

- Artykuł *A Visual Explanation of SQL Joins* opublikowany na stronie <https://blog.codinghorror.com/a-visual-explanation-of-sql-joins/>.
- Dokumentacja biblioteki pandas dotycząca złączeń na stronie https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html.

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

Długo szukałam książki, która spójnie przedstawiałaby algorytm ANN, hiperpłaszczyzny i wybór cech za pomocą losowego lasu. I wtedy pojawiła się ta pozycja!

Vicki Boykis, inżynier uczenia maszynowego w Duo

W ciągu ostatnich lat techniki uczenia maszynowego rozwijały się z niezwykłą dynamiką, rewolucjonizując pracę w różnych branżach. Obecnie do uczenia maszynowego najczęściej używa się Pythona i jego bibliotek. Znajomość najnowszych wydań tych narzędzi umożliwia efektywne tworzenie wyrafinowanych systemów uczących się.

Oto zaktualizowane wydanie popularnego przewodnika, dzięki któremu skorzystasz z ponad dwustu sprawdzonych receptur bazujących na najnowszych wydaniach bibliotek Pythona. Wystarczy, że skopiujesz i dostosujesz kod do swoich potrzeb. Możesz też go uruchamiać i testować za pomocą przykładowego zbioru danych. W książce znajdziesz receptury przydatne do rozwiązywania szerokiego spektrum problemów, od przygotowania i wczytania danych aż po trenowanie modeli i korzystanie z sieci neuronowych. W ten sposób wyjdziesz poza rozważania teoretyczne czy też matematyczne koncepcje i zaczniesz tworzyć aplikacje korzystające z uczenia maszynowego.

Poznaj receptury dotyczące:

- pracy z danymi w wielu formatach, z bazami i magazynami danych
- redukcji wymiarowości, jak również oceny i wyboru modelu
- regresji liniowej i logistycznej, drzew i lasów, a także najbliższych sąsiadów
- maszyn wektorów nośnych (SVM), naiwnej klasyfikacji bayesowskiej i klasteryzacji
- udostępniania wytrenowanych modeli za pomocą wielu frameworków

Kyle Gallatin jest inżynierem oprogramowania na platformie uczenia maszynowego w Etsy. Przez wiele lat pracował jako analityk danych, naukowiec i inżynier uczenia maszynowego.

Chris Albon jest dyrektorem do spraw uczenia maszynowego w Wikimedia Foundation.

Helion
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!



ISBN 978-83-289-0811-6



9 788328 908116

Cena: 89,00 zł