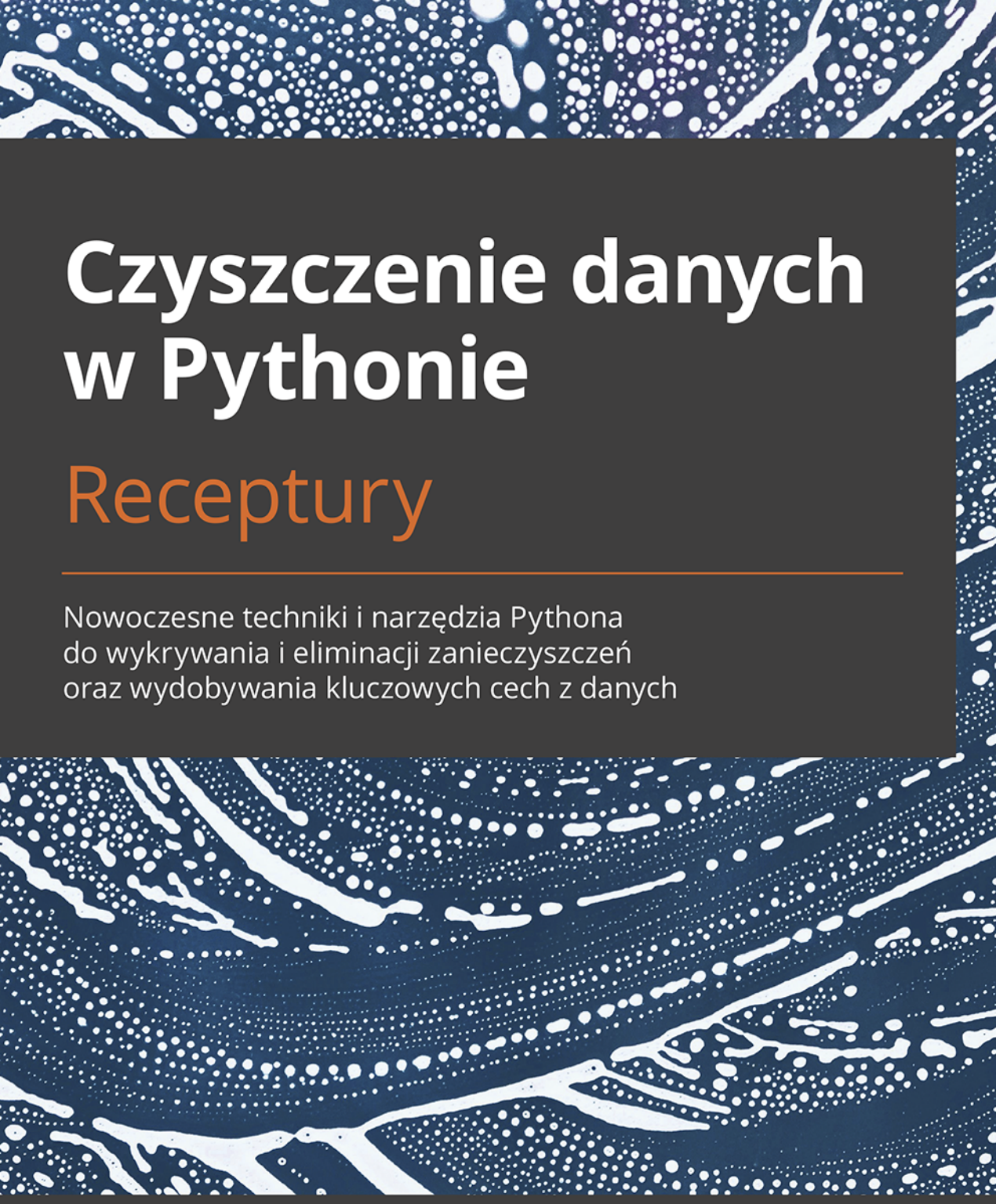

Czyszczenie danych w Pythonie

Receptury

Nowoczesne techniki i narzędzia Pythona
do wykrywania i eliminacji zanieczyszczeń
oraz wydobywania kluczowych cech z danych

Michael Walker

Helion

Tytuł oryginału: Python Data Cleaning Cookbook: Modern techniques and Python tools to detect and remove dirty data and extract key insights

Tłumaczenie: Filip Kamiński

ISBN: 978-83-283-8029-5

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/czydap.zip

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

Printed in Poland.

Spis treści

ROZDZIAŁ 1

Oczyszczanie danych podczas importowania danych tabelarycznych do pandas

Naukowe dystrybucje języka **Python** (Anaconda, WinPython, Canopy itd.) dostarczają analitykom imponującą gamę narzędzi do manipulacji danymi oraz ich eksploracji i wizualizacji. Jednym z najważniejszych narzędzi jest biblioteka *pandas*. Biblioteka ta została opracowana przez Wesa McKinneya w 2008 roku, ale popularność zyskała dopiero po 2012 roku. Obecnie *pandas* to podstawowa biblioteka do analizy danych w Pythonie. W tej książce będę z niej intensywnie korzystał. Skorzystam też z popularnych pakietów, takich jak NumPy, Matplotlib i SciPy.

Podstawowym obiektem danych w *pandas* jest ramka danych (`DataFrame`), która przedstawia dane w postaci tabeli z wierszami i kolumnami. Ten sposób przechowywania danych jest podobny do innych formatów, które omówię w tym rozdziale. Ramka danych pozwala również na indeksowanie, które sprawia, że wybieranie, łączenie i przekształcanie danych jest stosunkowo proste. Przekonasz się o tym, przeglądając receptury pokazane w tej książce.

Zanim będziemy mogli skorzystać z tej wspaniałej funkcjonalności, konieczne będzie przeniesienie danych do ramki *pandas*. Dane docierają do nas w wielu różnych formatach. Otrzymujemy je w postaci plików CSV lub Excela, tabel z baz SQL, dane mogą być zapisane w formatach pakietów statystycznych (takich jak SPSS, Stata, SAS lub R) lub mogą pochodzić ze źródeł innych niż tabelaryczne, takich jak pliki JSON lub strony WWW.

W tej recepturze przyjrzymy się sposobom importowania danych tabelarycznych. W szczególności omówię następujące zagadnienia:

- Importowanie plików CSV.
- Importowanie plików z Excela.
- Importowanie danych z baz danych SQL.
- Importowanie danych z pakietów SPSS, Stata i SAS.
- Importowanie danych z R.
- Przechowywanie danych tabelarycznych.

Wymagania techniczne

Kody i dane użyte w tym rozdziale są dostępne pod adresem *https://ftp.helion.pl/przyklady/czydap.zip*.

Importowanie danych z plików CSV

Metoda `read_csv` z biblioteki *pandas* może służyć do odczytywania **plików z wartościami oddzielonymi przecinkami** (CSV — ang. *comma separated values*). Metoda ta pozwala załadować zawartość pliku do pamięci w postaci ramki danych *pandas*. W tej recepturze wczytam plik CSV i rozwiążę kilka typowych problemów związanych z tą operacją, takich jak tworzenie sensownych nazw kolumn, parsowanie dat i usuwanie wierszy ze znacznymi brakami.

Surowe dane są często przechowywane w postaci plików CSV. W tym formacie do oddzielenia kolejnych wierszy służy znak powrotu karetki umieszczany na końcu każdego wiersza. Do oddzielenia kolumn służy przecinek, który umieszcza się pomiędzy kolejnymi wartościami. Zamiast przecinka można też zastosować inny separator, np. tabulator. Jeżeli separator występuje naturalnie jako element przechowywanej wartości (co dzieje się czasem na przykład w przypadku przecinka), to wartość taką można otoczyć cudzysłowami.

Wszystkie dane przechowywane w pliku CSV są typu znakowego, niezależnie od ich logicznego typu. Dzięki temu niezbyt duże pliki CSV można łatwo przeglądać w edytorze tekstu. Metoda `read_csv` pozwala na inteligentne odgadnięcie typu danych w każdej kolumnie. Proces ten wymaga wsparcia ze strony użytkownika; należy upewnić się, że typy ustalone przez tę metodę są prawidłowe.

Przygotuj się

Utwórz katalog dla tego rozdziału i umieść w nim nowy skrypt lub notatnik programu **Jupyter Notebook**. Następnie utwórz podkatalog *dane* i umieść w nim plik *landtempssample.csv*.

Możesz też po prostu pobrać wszystkie pliki z serwera FTP wydawnictwa Helion. Oto kilka początkowych wierszy z tego pliku CSV:

```
locationid,year,month,temp,latitude,longitude,stnelev,station,countryid,country
USS0010K01S,2000,4,5.27,39.9,-110.75,2773.7,INDIAN_CANYON,US,United States
CIO00085406,1940,5,18.04,-18.35,-70.333,58.0,ARICA,CI,Chile
USC00036376,2013,12,6.22,34.3703,-91.1242,61.0,SAINT_CHARLES,US,United States
ASN00024002,1963,2,22.93,-34.2833,140.6,65.5,BERRI_IRRIGATION,AS,Australia
ASN00028007,2001,11,,-14.7803,143.5036,79.4,MUSGRAVE,AS,Australia
```

Ten zestaw danych został pobrany ze zintegrowanej bazy danych Global Historical Climatology Network. Baza ta jest udostępniana przez United States National Oceanic and Atmospheric Administration. Znajdziesz ją pod adresem *https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/global-historical-climatology-network-monthly-version-4*. Wykorzystany przeze mnie plik to tylko 100 000 wierszy wybranych z pełnego zestawu danych.

Jak to zrobić...

Wczytamy plik CSV do ramki danych *pandas*. Skorzystamy w tym celu z kilku bardzo przydatnych opcji funkcji `read_csv`:

1. Zaimportuj bibliotekę *pandas* i skonfiguruj środowisko, tak aby ułatwić przeglądanie wyników:

```
import pandas as pd
pd.options.display.float_format = '{:,.2f}'.format
pd.set_option('display.width', 85)
pd.set_option('display.max_columns', 8)
```

2. Wczytaj plik z danymi, ustal nowe nazwy nagłówków i przekształć kolumny z datami.

   Przekazanie 1 w parametrze `skiprows` pozwoli Ci pominąć pierwszy wiersz. Aby utworzyć w ramce danych kolumnę typu `datetime`, za pomocą argumentu `parse_dates` przekaż metodzie listę kolumn, które zawierają składniki daty (miesiąc, rok). Aby zmniejszyć zużycie pamięci podczas procesu importu, ustaw parametr `low_memory` na `True`:

```
>>> landtemps = pd.read_csv('dane/landtempssample.csv',
...     names=[
'ID_stacji','rok','miesiąc','średnia_temperatura','szerokość_geograficzna',
...
'długość_geograficzna','wysokość','stacja','ID_państwa','nazwa_państwa'],
...     skiprows=1,
...     parse_dates=[['miesiąc','rok']],
...     low_memory=False)
>>> type(landtemps)
<class 'pandas.core.frame.DataFrame'>
```

3. Rzuć okiem na dane.

   Spójrz na kilka pierwszych wierszy. Wyświetl typy danych we wszystkich kolumnach, a także liczbę wierszy i kolumn:

```
>>> landtemps.head(7)
  miesiąc_rok    ID_stacji  ...  ID_państwa  nazwa_państwa
0  2000-04-01  USS0010K01S  ...          US  United States
1  1940-05-01  CIO00085406  ...          CI          Chile
2  2013-12-01  USC00036376  ...          US  United States
3  1963-02-01  ASN00024002  ...          AS      Australia
4  2001-11-01  ASN00028007  ...          AS      Australia
5  1991-04-01  USW00024151  ...          US  United States
6  1993-12-01  RSM00022641  ...          RS         Russia

[7 rows x 9 columns]
>>> landtemps.dtypes
miesiąc_rok               datetime64[ns]
ID_stacji                         object
średnia_temperatura              float64
szerokość_geograficzna           float64
długość_geograficzna             float64
wysokość                         float64
stacja                            object
ID_państwa                        object
nazwa_państwa                     object
dtype: object
>>> landtemps.shape
(100000, 9)
```

4. Nadaj kolumnie z datami lepszą nazwę i przejrzyj statystyki podsumowujące średnią miesięczną temperaturę:

```
>>> landtemps.rename(columns={'miesiąc_rok':'data_pomiaru'}, inplace=True)
>>> landtemps.dtypes
data_pomiaru              datetime64[ns]
ID_stacji                         object
średnia_temperatura              float64
szerokość_geograficzna           float64
długość_geograficzna             float64
wysokość                         float64
stacja                            object
ID_państwa                        object
nazwa_państwa                     object
dtype: object
>>> landtemps.średnia_temperatura.describe()
count   85,554.00
mean        10.92
std         11.52
min        -70.70
25%          3.46
50%         12.22
75%         19.57
max         39.95
Name: średnia_temperatura, dtype: float64
```

5. Znajdź brakujące wartości w każdej kolumnie.

   Użyj `isnull`. Funkcja ta zwraca `True` dla każdej brakującej wartości w każdej kolumnie i `False` w przeciwnym wypadku. Aby określić liczbę braków w każdej kolumnie, połącz wywołanie `isnull` z wywołaniem `sum`. (Podczas pracy z wartościami logicznymi funkcja `sum` traktuje `True` jako `1`, a `False` jako `0`. Łączenie wywołań w łańcuchy omówię w punkcie „Zobacz również...” w tej recepturze):

```
>>> landtemps.isnull().sum()
data_pomiaru                  0
ID_stacji                     0
średnia_temperatura       14446
szerokość_geograficzna        0
długość_geograficzna          0
wysokość                      0
stacja                        0
ID_państwa                    0
nazwa_państwa                 5
dtype: int64
```

6. Usuń wiersze, w których występują braki w kolumnie `średnia_temperatura`.

   Za pomocą parametru `subset` funkcji `dropna` usuń wiersze, w których brakuje wartości w kolumnie `średnia_temperatura`. Ustaw argument `inplace` na `True`. Pozostawienie wartości `False` spowoduje wyświetlenie ramki danych bez tych wierszy, ale wprowadzone zmiany nie zostaną zachowane. Do sprawdzenia liczby wierszy i kolumn wykorzystaj atrybut `shape`:

```
>>> landtemps.dropna(subset=['średnia_temperatura'], inplace=True)
>>> landtemps.shape
(85554, 9)
```

To wszystko! Importowanie danych z plików CSV w *pandas* jest aż tak proste.

Jak to działa...

Prawie wszystkie receptury w tej książce korzystają z biblioteki *pandas*. Aby uprościć odwołania w kodzie, importuję bibliotekę pod zwyczajową nazwą `pd`. Do wyświetlania wartości zmiennoprzecinkowych w czytelny sposób wykorzystuję parametr `float_format` oraz metodę `set_option`. Pozwala to zagwarantować, że wyjście z terminala będzie wystarczająco szerokie i pomieści wartości kilku zmiennych.

Duża część pracy została wykonana w pierwszej linii kodu z *kroku 2*. Do załadowania danych do ramki (o nazwie `landtemps`) i umieszczenia jej w pamięci służy funkcja `read_csv`. Oprócz nazwy pliku w parametrze `names` przekazałem listę preferowanych nazw kolumn. W powyższym wywołaniu poleciłem funkcji `read_csv` pominięcie pierwszego wiersza w pliku CSV (`skiprows = 1`). Wiersz ten zawiera oryginalne nazwy kolumn. Jeśli się go nie pominie, to funkcja `read_csv` potraktuje ten nagłówek jak zwykłe dane.

Funkcja `read_csv` rozwiązuje również problem z konwersją dat. Za pomocą parametru `parse_dates` można ją poprosić o przekonwertowanie kolumn `miesiąc` i `rok` na poprawną datę.

Krok 3. składa się z kilku standardowych sprawdzeń danych. Wykorzystałem `head(7)`, aby wyświetlić na ekranie wartości wszystkich kolumny dla pierwszych 7 wierszy. Za pomocą atrybutu `dtypes` ramki danych wyświetlam typy danych we wszystkich kolumnach. Każda kolumna ma typ zgodny z oczekiwaniami. W *pandas* dane znakowe są typu `object`. Jest to typ danych, który dopuszcza istnienie wartości różnych rodzajów. Atrybut `shape` zwraca krotkę, której pierwszym elementem jest liczba wierszy w ramce (w tym przypadku `100 000`), a drugim liczba kolumn (`9`).

Parsowanie kolumn `miesiąc` i `rok` za pomocą `read_csv` dało w efekcie nową kolumnę `miesiąc_rok`. W *kroku 4.* wywołałem metodę `rename` i za jej pomocą nadałem tej kolumnie bardziej czytelną nazwę. Aby w pamięci zastąpić starą nazwę kolumny nową, konieczne jest przekazanie tej metodzie parametru `inplace = True`. Metoda `describe` pozwala obliczyć statystyki podsumowujące dla kolumny `średnia_temperatura`.

Zwróć uwagę, że ze 100 000 wierszy znajdujących się w tej ramce danych 85 554 zawiera prawidłowe wartości średniej temperatury (atrybut `shape`). Potwierdza to ustalenie liczby brakujących wartości w *kroku 5.* (`landtemps.isnull().sum()`): 100 000 – 85 554 = 14 446.

W *kroku 6.* usuwam wszystkie wiersze, w których `średnia_temperatura` to `NaN`. Wartość `NaN` (ang. *not a number* — nie-liczba) reprezentuje w *pandas* brak wartości. Argument `subset` jest używany do wskazania nazwy kolumny, w której ma nastąpić sprawdzenie braków. Atrybut `shape` zmiennej `landtemps` wskazuje, że po zmianach w ramce pozostają 85 554 wiersze. Biorąc pod uwagę obliczenia z poprzedniego akapitu, wartość ta nie jest w żaden sposób zaskakująca.

Zobacz również...

Jeśli wczytywany plik zawiera separator inny niż przecinek (na przykład tabulator), to informacje o jego użyciu można przekazać funkcji `read_csv` w parametrze `sep`. Podczas tworzenia ramki danych tworzony jest również indeks. W rezultatach wywołania funkcji `head` i `sample` wartości indeksu są wyświetlane jako pierwsze z lewej. W argumencie funkcji `head` i `sample` można określić dowolną liczbę wierszy. Wartość domyślna to `5`.

Ustawienie `low_memory` na `True` powoduje, że funkcja `read_csv` przetwarza plik we fragmentach. Ułatwia to pracę z dużymi plikami w systemach z mniejszą ilością pamięci. Z drugiej strony po pomyślnym zakończeniu wywołania `read_csv` do pamięci komputera i tak zostanie załadowana pełna ramka danych.

Instrukcja `landtemps.isnull().sum()` jest przykładem łączenia wywołań funkcji w łańcuch. Wywołanie `isnull` zwraca ramkę danych zawierającą wartości `True` i `False`. Wartości te to wyniki sprawdzeń, czy w poszczególnych elementach ramki nie znajdują się wartości `null`. Funkcja `sum` przyjmuje tę ramkę danych i sumuje wartości `True` w każdej kolumnie. Podczas sumowania wartości `True` są traktowane jako `1`, a wartości `False` jako `0`. Ten sam wynik można otrzymać również, wykonując dwa poniższe kroki:

```
>>> checknull = landtemps.isnull()
>>> checknull.sum()
```

Nie istnieje sztywna reguła, który mówi nam, kiedy łączyć ze sobą wywołania metod, a kiedy nie. Moim zdaniem łączenie wywołań jest pomocne, gdy myślę o moim działaniu jako o jednym kroku, którego techniczna realizacja składa się z dwóch lub więcej etapów. Łączenie wywołań ma również tę zaletę, że nie tworzy ono dodatkowych niepotrzebnych obiektów.

Zbiór danych użyty w tej recepturze to tylko próbka z pełnej bazy danych pomiarów temperatur lądu, która zawiera prawie 17 milionów rekordów. Jeśli Twój komputer poradzi sobie z większą ilością danych, to pełny plik (po jego pobraniu z bazy i zapisaniu w katalogu *dane* pod nazwą *landtemps.csv*) możesz wczytać za pomocą następującego kodu:

```
>>> landtemps = pd.read_csv('dane/landtemps.csv',
...
names=['ID_stacji','rok','miesiąc','średnia_temperatura','szerokość_geograficzna',
...       'długość_geograficzna','wysokość','stacja','ID_państwa','nazwa_państwa'],
...     skiprows=1,
...     parse_dates=[['miesiąc','rok']],
...     low_memory=False)
```

Funkcja `read_csv` może odczytać skompresowany plik *.zip*. W tym celu należy przekazać jej nazwę archiwum i informacje o użytym rodzaju kompresji.

Co dalej?

W kolejnych recepturach w tym i w dalszych rozdziałach wykorzystamy indeksy, aby usprawnić nawigację po wierszach i scalanie.

Dane wykorzystane w tej recepturze różnią się od surowych danych, które możesz pobrać z Global Historical Climatology Network. Przed stworzeniem tego kodu surowe dane zostały przeze mnie przetworzone. Proces modyfikacji opisałem w rozdziale 8., „Rozwiązywanie problemów z danymi podczas łączenia ramek danych".

Importowanie plików z Excela

Metoda `read_excel` z biblioteki *pandas* służy do importowania danych z arkuszy Excela oraz wczytywania ich do pamięci urządzenia w postaci ramki danych. W tej recepturze zaimportujemy plik Excela i rozwiążemy kilka typowych problemów podczas pracy z takimi plikami, w tym obecność zbędnych informacji w nagłówku i stopce, wybieranie określonych kolumn, usuwanie pustych wierszy i odwoływanie się do określonych arkuszy.

Pomimo tabelarycznej struktury Excela, która zachęca do organizacji danych w wiersze i kolumny, arkusze kalkulacyjne nie są zbiorami danych i nie wymagają od użytkowników przechowywania danych w takim formacie. Nawet jeśli niektóre dane spełniają te założenia, to w wierszach lub kolumnach przed lub po danych często znajdują się dodatkowe informacje. Ponadto zastosowane typy danych nie zawsze są tak jasne, jak dla osoby, która utworzyła arkusz. Wszystkie te problemy są doskonale znane każdemu, kto kiedykolwiek walczył

z importowaniem wiodących zer. Ponadto Excel nie wymaga, aby wszystkie dane w kolumnie były tego samego typu oraz aby nazwy kolumn były odpowiednie do użycia w języku programowania, takim jak Python.

Na szczęście funkcja `read_excel` oferuje wiele możliwości radzenia sobie z bałaganem w danych pochodzących z Excela. Te opcje ułatwiają pomijanie wierszy i wybieranie określonych kolumn oraz pobieranie danych z określonego arkusza lub arkuszy.

Przygotuj się

Plik *GDPpercapita.xlsx*, a także kod tej receptury znajdziesz w archiwum pobranym ze strony wydawnictwa Helion. Kod zakłada, że plik z Excela znajduje się w podfolderze *dane*. Początek tego pliku przedstawia rysunek 1.1.

Dataset: Metropolitan areas							
Variables	GDP per capita						
Unit	US Dollar						
Year	2001	2002	2003	2004	2005	2006	2007
Metropolitan areas							
AUS: Australia	..	..	..	..	..	..	..
AUS01: Greater Sydney	43313	44008	45424	45837	45423	45547	45880
AUS02: Greater Melbourne	40125	40894	41602	42188	41484	41589	42316
AUS03: Greater Brisbane	37580	37564	39080	40762	42976	44475	44635

Rysunek 1.1. Początek zbioru danych

Koniec pliku przedstawiono na rysunku 1.2.

USA162: Tuscaloosa	35370	36593	38907	41846	44774	44298	46190
USA164: Linn	53047	51751	54894	58660	60195	58244	61742
USA165: Lafayette (IN)	38057	38723	39173	40412	40285	40879	41717
USA167: Weber	34592	34997	35587	35776	37613	41213	41554
USA169: Cass	44597	46856	49043	49134	49584	50417	51596
USA170: Benton (AR)	41988	44687	45296	47799	49260	47329	45503
Data extracted on 05 May 2020 10:55 UTC (GMT) from OECD.Stat							

Rysunek 1.2. Koniec zbioru danych

> Ten zbiór danych pochodzi z Organizacji Współpracy Gospodarczej i Rozwoju. Zbiór jest dostępny publicznie pod adresem *https://stats.oecd.org/*.

Jak to zrobić...

Zaimportujemy plik Excela do *pandas* i wykonamy wstępne oczyszczanie danych:

1. Zaimportuj bibliotekę *pandas*:

```
>>> import pandas as pd
```

2. Wczytaj plik Excela zawierający wartości PKB *per capita*.

   Wybierz arkusz z danymi, których potrzebujesz, pomiń zbędne kolumny i wiersze. Użyj parametru `sheet_name`, aby określić nazwę arkusza. Aby pominąć pierwsze cztery rzędy (pierwszy rząd jest ukryty) i jeden końcowy, ustaw wartość `skiprows` na `4` i `skipfooter` na `1`. Za pomocą `usecols` pobierz dane z kolumny `A` i kolumn od `C` do `T` (kolumna `B` jest pusta). Zastosuj metodę `head`, aby wyświetlić kilka pierwszych wierszy:

```
>>> percapitaGDP = pd.read_excel("dane/GDPpercapita.xlsx",
...   engine='openpyxl',
...   sheet_name="OECD.Stat export",
...   skiprows=4,
...   skipfooter=1,
...   usecols="A,C:T")

>>> percapitaGDP.head()
                         Year   2001  ...   2017   2018
0          Metropolitan areas    NaN  ...    NaN    NaN
1              AUS: Australia     ..  ...     ..     ..
2       AUS01: Greater Sydney  43313  ...  50578  49860
3    AUS02: Greater Melbourne  40125  ...  43025  42674
4     AUS03: Greater Brisbane  37580  ...  46876  46640

[5 rows x 19 columns]
```

3. Użyj metody `info` ramki danych, aby wyświetlić typy danych i liczbę wartości niezerowych (`non-null`):

```
>>> percapitaGDP.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 702 entries, 0 to 701
Data columns (total 19 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Year    702 non-null    object
 1   2001    701 non-null    object
 2   2002    701 non-null    object
 3   2003    701 non-null    object
 4   2004    701 non-null    object
 5   2005    701 non-null    object
 6   2006    701 non-null    object
 7   2007    701 non-null    object
 8   2008    701 non-null    object
 9   2009    701 non-null    object
 10  2010    701 non-null    object
 11  2011    701 non-null    object
 12  2012    701 non-null    object
 13  2013    701 non-null    object
 14  2014    701 non-null    object
 15  2015    701 non-null    object
 16  2016    701 non-null    object
 17  2017    701 non-null    object
 18  2018    701 non-null    object
dtypes: object(19)
memory usage: 104.3+ KB
```

4. Zmień nazwę kolumny `Year` na `metro` i usuń spacje.

   Nadaj odpowiednią nazwę kolumnie, która zawiera nazwy obszarów metropolitalnych. W niektórych wierszach istnieją dodatkowe spacje przed nazwą obszaru. W innych dodatkowe spacje występują po nazwie obszaru. Do sprawdzenia, czy spacje występują przed nazwą, możesz wykorzystać `startswith(' ')`. Za pomocą `any` możesz sprawdzić, czy taka sytuacja ma miejsce chociaż jeden raz. Do zbadania spacji końcowych można wykorzystać `endswith(' ')`. Aby usunąć zarówno początkowe, jak i końcowe spacje, skorzystaj z `strip`:

```
>>> percapitaGDP.rename(columns={'Year':'metro'}, inplace=True)
>>> percapitaGDP.metro.str.startswith(' ').any()
True
>>> percapitaGDP.metro.str.endswith(' ').any()
True
>>> percapitaGDP.metro = percapitaGDP.metro.str.strip()
```

5. Zmień typ kolumny na liczbowy.

   Przejdź pętlą przez wszystkie kolumny, które zawierają wartości PKB z lat 2001 – 2018, i zmień ich typ danych z `object` na `float`. Wymuszaj konwersję, nawet jeżeli w kolumnie występują dane znakowe (w tym przykładzie `..`). Chcemy, aby w miejscach, w których występują znaki, pojawiły się braki w wartościach. Zmień nazwy kolumn z PKB z poszczególnych lat, tak aby lepiej odzwierciedlały one znajdujące się w nich dane:

```
>>> for col in percapitaGDP.columns[1:]:
...   percapitaGDP[col] = pd.to_numeric(percapitaGDP[col], errors='coerce')
...   percapitaGDP.rename(columns={col:'pcGDP'+col}, inplace=True)
...
>>> percapitaGDP.head()
                     metro  pcGDP2001  ...  pcGDP2017  pcGDP2018
0        Metropolitan areas        NaN  ...        NaN        NaN
1            AUS: Australia        NaN  ...        NaN        NaN
2     AUS01: Greater Sydney      43313  ...      50578      49860
3  AUS02: Greater Melbourne      40125  ...      43025      42674
4   AUS03: Greater Brisbane      37580  ...      46876      46640

[5 rows x 19 columns]
>>> percapitaGDP.dtypes
metro         object
pcGDP2001    float64
pcGDP2002    float64

Wyjście skrócono, aby oszczędzić miejsce

pcGDP2017    float64
pcGDP2018    float64
dtype: object
```

6. Użyj metody `describe`, aby wygenerować statystyki dla wszystkich wartości liczbowych w ramce danych:

```
>>> percapitaGDP.describe()
       pcGDP2001  pcGDP2002  ...  pcGDP2017  pcGDP2018
count        424        440  ...        445        441
```

```
mean       41264      41015  ...      47489      48033
std        11878      12537  ...      15464      15720
min        10988      11435  ...       2745       2832
25%        33139      32636  ...      37316      37908
50%        39544      39684  ...      45385      46057
75%        47972      48611  ...      56023      56638
max        91488      93566  ...     122242     127468

[8 rows x 18 columns]
```

7. Usuń wiersze, w których brakuje wszystkich wartości PKB *per capita*.

   Wykorzystaj parametr `subset` metody `dropna` do sprawdzenia kolumn od drugiej do ostatniej (numeracja kolumn rozpoczyna się od zera). Za pomocą argumentu `how` określ, że wiersz zostanie usunięty tylko wtedy, gdy brakuje w nim wartości we wszystkich kolumnach określonych w argumencie `subset`. Liczbę wierszy i kolumn w wynikowej ramce danych wyświetl za pomocą atrybutu `shape`:

```
>>> percapitaGDP.dropna(subset=percapitaGDP.columns[1:], how="all",
inplace=True)

>>> percapitaGDP.describe()
       pcGDP2001  pcGDP2002  ...  pcGDP2017  pcGDP2018
count        424        440  ...        445        441
mean       41264      41015  ...      47489      48033
std        11878      12537  ...      15464      15720
min        10988      11435  ...       2745       2832
25%        33139      32636  ...      37316      37908
50%        39544      39684  ...      45385      46057
75%        47972      48611  ...      56023      56638
max        91488      93566  ...     122242     127468

[8 rows x 18 columns]

>>> percapitaGDP.head()
                    metro  pcGDP2001  ...  pcGDP2017  pcGDP2018
2     AUS01: Greater Sydney      43313  ...      50578      49860
3  AUS02: Greater Melbourne      40125  ...      43025      42674
4   AUS03: Greater Brisbane      37580  ...      46876      46640
5      AUS04: Greater Perth      45713  ...      66424      70390
6   AUS05: Greater Adelaide      36505  ...      40115      39924

[5 rows x 19 columns]
>>> percapitaGDP.shape
(480, 19)
```

8. Ustaw jako indeks kolumnę `metro`.

   Przed ustawieniem indeksu sprawdź, czy w kolumnie tej znajduje się 480 prawidłowych i unikatowych wartości:

```
>>> percapitaGDP.metro.count()
480
>>> percapitaGDP.metro.nunique()
480
>>> percapitaGDP.set_index('metro', inplace=True)
>>> percapitaGDP.head()
```

```
                          pcGDP2001  pcGDP2002  ...  pcGDP2017  pcGDP2018
metro                                           ...
AUS01: Greater Sydney         43313      44008  ...      50578      49860
AUS02: Greater Melbourne      40125      40894  ...      43025      42674
AUS03: Greater Brisbane       37580      37564  ...      46876      46640
AUS04: Greater Perth          45713      47371  ...      66424      70390
AUS05: Greater Adelaide       36505      37194  ...      40115      39924

[5 rows x 18 columns]

>>> percapitaGDP.loc['AUS02: Greater Melbourne']
pcGDP2001   40125
pcGDP2002   40894
...
pcGDP2017   43025
pcGDP2018   42674
Name: AUS02: Greater Melbourne, dtype: float64
```

Zaimportowałeś dane Excela do ramki danych *pandas* i usunąłeś część bałaganu, który panował w arkuszu kalkulacyjnym.

Jak to działa...

Dzięki możliwości pominięcia niektórych wierszy i kolumn najczęściej udaje się uzyskać potrzebne dane już w *kroku 2*. Po ich pobraniu nadal pozostaje jednak szereg problemów do rozwiązania: funkcja `read_excel` interpretuje wszystkie wartości PKB jako dane znakowe, w danych znajduje się wiele wierszy, które nie zawierają przydatnych danych, a nazwy kolumn nie opisują danych we właściwy sposób. Ponadto kolumna z nazwami obszarów miejskich mogłaby posłużyć jako indeks, ale są w niej spacje na początku i na końcu nazw. Co więcej, może w niej brakować wartości lub niektóre nazwy mogą występować w niej więcej niż raz.

Funkcja `read_excel` poszukuje nazw kolumn w nagłówku powyżej danych. Ponieważ w arkuszu Excela słowo *Year* występuje w kolumnie z nazwami obszarów (pojawia się ono nad danymi), kolumna z nazwami obszarów w ramce danych została błędnie nazwana `Year`. W *kroku 4.* zmieniłem nazwę tej kolumny na `metro`. Aby rozwiązać problem z początkowymi i końcowymi spacjami, wywołałem `strip`. Gdyby spacje występowały tylko na początku (na końcu), do ich usunięcia można by zastosować `lstrip` (`rstrip`). Dobrym pomysłem jest założenie, że w danych znakowych mogą znajdować się początkowe lub końcowe spacje. Dane tego typu warto oczyścić zaraz po ich imporcie.

Autorzy arkusza kalkulacyjnego wykorzystali `..` do reprezentowania brakujących danych. Ponieważ są to w rzeczywistości prawidłowe dane znakowe, *pandas* przypisuje zawierającym je kolumnom typ `object` (w ten sposób *pandas* traktuje kolumny zawierające dane znakowe lub mieszane). Konwersję na typ liczbowy wymuszam w *kroku 5*. Takie wymuszenie powoduje również, że oryginalne wartości `..` są zastępowane przez `NaN` (nie-liczbę) — wartość stosowaną w *pandas* do reprezentacji braku liczb. Działanie takie jest zgodne z naszymi oczekiwaniami.

Dzięki możliwości łatwego iterowania po kolumnach ramki danych wszystkie problemy związane z kolumnami zawierającymi PKB *per capita* możemy naprawić zaledwie w kilku wierszach. Zapis `[1:]` pozwala przejść pętlą od drugiej do ostatniej kolumny. Wewnątrz pętli zmieniam typy wartości w kolumnach na typ liczbowy. Równocześnie zmieniam też nazwy kolumn na bardziej odpowiednie.

Istnieje kilka powodów, dla których warto zmienić nazwy kolumn z PKB. Zmiana pomaga nam zapamiętać, jakie dane znajdują się w tych kolumnach. Po połączeniu danych z innymi danymi odnoszącymi się do obszarów metropolitalnych nie będziemy musieli martwić się o konflikty w nazwach kolumn. Zmiana umożliwi również dostęp do poszczególnych serii danych za pomocą atrybutów (możliwość tę omówię bardziej szczegółowo w punkcie „Zobacz również..." tej receptury).

Użycie metody `describe` w *kroku 6.* pokazuje nam, że tylko między 420 a 480 wierszy zawiera prawidłowe wartości PKB *per capita*. Po usunięciu w *kroku 7.* wszystkich wierszy, w których brakuje wartości dla wszystkich kolumn PKB, otrzymujemy ramkę danych zawierającą 480 wierszy. Wartość ta jest zgodna z naszymi przewidywaniami.

Zobacz również...

Gdy mamy już ramkę danych, znajdujące się w niej kolumny możemy traktować jako coś więcej niż tylko kolumny. Za pomocą atrybutów (takich jak `percapitaGPA.metro`) lub notacji z nawiasami (`percapitaGPA['metro']`) możemy uzyskać funkcjonalność znaną z obiektów typów `Series`. Zastosowanie każdej z metod pozwala na użycie metod pracujących na łańcuchach znaków, takich jak `str.startswith`, oraz metod liczących, takich jak `nunique`. Zauważ, że oryginalne nazwy kolumn w postaci `20##` nie pozwalają na takie odwołania, ponieważ zaczynają się od cyfry. Wywołanie `percapitaGDP.pcGDP2001.count()` działa, a `percapitaGDP.2001.count()` powoduje błąd składniowy, ponieważ `2001` nie jest w Pythonie prawidłowym identyfikatorem (zaczyna się od liczby).

Pandas zawiera wiele funkcji do manipulacji łańcuchami znaków i wiele operacji ułatwiających pracę z seriami danych. Wiele z nich wypróbujemy w kolejnych recepturach. Ta receptura pokazała te, które uważam za najbardziej przydatne podczas importowania danych Excela.

Co dalej?

Istnieją dobre powody, aby rozważyć zmianę kształtu danych. Zamiast 18 kolumn z wartościami PKB *per capita* w poszczególnych latach dla każdego obszaru metropolitalnego powinniśmy mieć 18 wierszy z wartościami w dwóch kolumnach: rok i PKB *per capita*. Receptury dotyczące przekształcania danych można znaleźć w rozdziale 9., „Porządkowanie i przekształcanie danych".

Importowanie danych z baz SQL

W tej recepturze wykorzystamy API *pymssql* i *mysql* do pobrania danych z odpowiednio: bazy MicrosoftSQL Server i MySQL (która obecnie należy do Oracle). Dane z takich źródeł mają zazwyczaj dobrą strukturę, ponieważ bazy danych zostały zaprojektowanie z myślą o obsłudze wielu odbywających się równocześnie transakcji, które dokonywane są przez członków korporacji i współpracujące z nimi podmioty. Każda transakcja może być powiązana zależnościami z inną.

Chociaż tabele z systemów biznesowych mają bardziej niezawodną strukturę niż dane z plików CSV i plików Excela, to istnieje mniejsze prawdopodobieństwo, że ich logika będzie zawarta w jednej tabeli. Aby poznać ich pełne znaczenie, musisz zazwyczaj wiedzieć, jak dane z jednej tabeli odnoszą się do danych z innej. Podczas pobierania danych należy zachować te relacje poprzez m.in. zapewnienie integralności kluczy głównych i obcych. Ponadto dobrze zorganizowane tabele mogą być skomplikowane. Często stosuje się w nich wyrafinowane schematy kodowania, które determinują wartości poszczególnych pól. Dodatkowo same schematy kodowania mogą się zmieniać w czasie. Na przykład kody przynależności etnicznej pracowników sieci sklepów detalicznych w 1998 roku mogły być inne niż w 2020. W tabelach często występują też wartości kodujące braki, takie jak wartość `99999`, którą *pandas* zinterpretuje jako prawidłową wartość.

Ponieważ znaczna część tej logiki to logika biznesowa zaimplementowana w procedurach składowania (ang. *stored procedures*) lub w innych aplikacjach, po wyjęciu danych z większego systemu jest ona tracona. Część tego, co zostanie utracone, trzeba będzie zrekonstruować podczas przygotowywania danych do analizy. Prawie zawsze wiąże się to z łączeniem danych z wielu tabel, dlatego ważne jest, aby zachować taką możliwość. Zmiany mogą też polegać na dodaniu części utraconej po wczytaniu tabeli SQL logiki do ramki danych *pandas*. W tej recepturze pokażę przykład takiego działania.

Przygotuj się

W tej recepturze zakładam, że masz zainstalowane interfejsy API *pymssql* i *mysql*. Ich instalacja za pomocą narzędzia `pip` jest stosunkowo prosta. W terminalu lub w interpreterze PowerShell (w systemie Windows) zainstalujesz je za pomocą poleceń `pip install pymssql` i `pip install mysql-connector-python`.

> Zbiór danych użyty w tym przepisie jest dostępny publicznie pod adresem *https://archive.ics.uci.edu/ml/machine-learning-databases/00320/*.

Jak to zrobić...

Tabele z baz SQL Server i MySQL można zaimportować do ramki danych w następujący sposób:

1. Zaimportuj *pandas*, NumPy, *pymssql* i *mysql*.

   W tym kroku zakładam, że masz zainstalowane interfejsy API *pymssql* i *mysql*:

   ```
   >>> import pandas as pd
   >>> import numpy as np
   >>> import pymssql
   >>> import mysql.connector
   ```

2. Użyj API *pymssql* i funkcji `read_sql` do pobrania i załadowania danych z SQL Server.

   Wybierz kolumny z bazy, wykorzystaj aliasy do poprawy nazw kolumn (na przykład `fedu AS fathereducation`). Połącz się z bazą, przekazując dane logowania funkcji `pymssql.connect`. Ramkę danych utworzysz za pomocą funkcji `read_sql`, której musisz przekazać zapytanie (`query`) i obiekt reprezentujący połączenie (`conn`). Zamknij połączenie z bazą i zwróć je do puli na serwerze:

   ```
   >>> query = "SELECT studentid, school, sex, age, famsize,\
   ... medu AS mothereducation, fedu AS fathereducation,\
   ... traveltime, studytime, failures, famrel, freetime,\
   ... goout, g1 AS gradeperiod1, g2 AS gradeperiod2,\
   ... g3 AS gradeperiod3 From studentmath"
   >>>
   >>> server = "pdcc.c9sqqzd5fulv.us-west-2.rds.amazonaws. com"
   >>> user = "pdccuser"
   >>> password = "pdccpass"
   >>> database = "pdcctest"
   >>>
   >>> conn = pymssql.connect(server=server,
   ... user=user, password=password, database=database)
   >>>
   >>> studentmath = pd.read_sql(query,conn)
   >>> conn.close()
   ```

3. Sprawdź typy danych i spójrz na kilka pierwszych wierszy:

   ```
   >>> studentmath.dtypes
   studentid          object
   school             object
   sex                object
   age                 int64
   famsize            object
   mothereducation     int64
   fathereducation     int64
   traveltime          int64
   studytime           int64
   failures            int64
   famrel              int64
   freetime            int64
   goout               int64
   ```

```
gradeperiod1        int64
gradeperiod2        int64
gradeperiod3        int64
dtype: object

>>> studentmath.head()
  studentid school  ... gradeperiod2  gradeperiod3
0       001     GP  ...            6             6
1       002     GP  ...            5             6
2       003     GP  ...            8            10
3       004     GP  ...           14            15
4       005     GP  ...           10            10

[5 rows x 16 columns]
```

4. (Alternatywnie) Zastosuj API `mysql` i funkcję `read_sql`, aby pobrać dane z bazy MySQL.

   Aby pobrać dane i załadować je do ramki danych *pandas*, utwórz połączenie (obiekt, który je reprezentuje) do bazy MySQL i przekaż je do funkcji `read_sql`. (W obu bazach danych umieściłem te same dane dotyczące ocen uczniów z matematyki. Z tego powodu w tym przykładzie możemy skorzystać z tej samej instrukcji `SELECT`, co w poprzednim kroku):

```
>>> host = "pdccmysql.c9sqqzd5fulv.us-west-2.rds.amazonaws.com"
>>> user = "pdccuser"
>>> password = "pdccpass"
>>> database = "pdccschema"

>>> connmysql = mysql.connector.connect(host=host,
... database=database,user=user,password=password)

>>> studentmath = pd.read_sql(sqlselect,connmysql)
>>> connmysql.close()
```

5. Zmień kolejność kolumn, ustal indeks i sprawdź brakujące wartości.

   Przesuń oceny do lewej części ramki danych, umieść je zaraz po kolumnie `studentid`. Kolumnę `freetime` przesuń w prawo i umieść ją zaraz po kolumnach `traveltime` i `studytime`. Upewnij się, że każdy wiersz zawiera identyfikator oraz że identyfikatory te są unikalne. Ustaw `studentid` jako indeks:

```
>>> newcolorder = ['studentid', 'gradeperiod1', 'gradeperiod2',
...   'gradeperiod3', 'school', 'sex', 'age', 'famsize',
...   'mothereducation', 'fathereducation', 'traveltime',
...   'studytime', 'freetime', 'failures', 'famrel',
...   'goout']
>>> studentmath = studentmath[newcolorder]
>>> studentmath.studentid.count()
395
>>> studentmath.studentid.nunique()
395
>>> studentmath.set_index('studentid', inplace=True)
```

6. Użyj funkcji `count` ramki danych do sprawdzenia brakujących wartości:

```
>>> studentmath.count()
gradeperiod1        395
gradeperiod2        395
gradeperiod3        395
school              395
sex                 395
age                 395
famsize             395
mothereducation     395
fathereducation     395
traveltime          395
studytime           395
freetime            395
failures            395
famrel              395
goout               395
dtype: int64
```

7. Zastąp zakodowane wartości bardziej znaczącymi odpowiednikami.

   Utwórz słownik określający sposób zastąpienia wartości w kolumnach. Do zmiany wykorzystaj funkcję `replace`:

```
>>> setvalues={"famrel":{1:"1:bardzo zły",2:"2:zły",3:"3:średni",
...      4:"4:dobry",5:"5:świetny"},
...    "freetime":{1:"1:bardzo niski",2:"2:niski",3:"3:średni",
...      4:"4:wysoki",5:"5:bardzo wysoki"},
...    "goout":{1:"1:bardzo niski",2:"2:niski",3:"3:średni",
...      4:"4:wysoki",5:"5:bardzo wysoki"},
...    "mothereducation":{0:np.nan,1:"1:k-4",2:"2:5-9",
...      3:"3:wykształcenie średnie",4:"4:wykształcenie wyższe"},
...    "fathereducation":{0:np.nan,1:"1:k-4",2:"2:5-9",
...      3:"3:wykształcenie średnie",4:"4:wykształcenie wyższe"}}

>>> studentmath.replace(setvalues, inplace=True)
>>> setvalueskeys = [k for k in setvalues]
```

8. Zmień typ kolumn, w których zmieniłeś wartości, na `category`. Sprawdź, czy nie zaszły zmiany w użyciu pamięci:

```
>>> studentmath[setvalueskeys].memory_usage(index=False)
famrel              3160
freetime            3160
goout               3160
mothereducation     3160
fathereducation     3160
dtype: int64

>>> for col in studentmath[setvalueskeys].columns:
...     studentmath[col] = studentmath[col].astype('category')
...
>>> studentmath[setvalueskeys].memory_usage(index=False)
famrel              595
freetime            595
goout               595
```

```
mothereducation    587
fathereducation    587
dtype: int64
```

9. Oblicz procentowy udział wartości w kolumnie `famrel`.

   Aby obliczyć procentowy udział wartości, skorzystaj z funkcji `value_counts` z argumentem `normalize` o wartości `True`:

```
>>> studentmath['famrel'].value_counts(sort=False, normalize=True)
>>> studentmath['famrel'].value_counts(sort=False, normalize=True)
1:bardzo zły   0.02
2:zły          0.05
3:średni       0.17
4:dobry        0.49
5:świetny      0.27
Name: famrel, dtype: float64
```

10. Skorzystaj z `apply`, aby obliczyć procentowe udziały w wielu kolumnach:

```
>>> studentmath[['freetime','goout']].\
...   apply(pd.Series.value_counts, sort=False, normalize=True)
                 freetime  goout
1:bardzo niski       0.05   0.06
2:niski              0.16   0.26
3:średni             0.40   0.33
4:wysoki             0.29   0.22
5:bardzo wysoki      0.10   0.13
>>> studentmath[['mothereducation','fathereducation']].\
...   apply(pd.Series.value_counts, sort=False, normalize=True)
                          mothereducation  fathereducation
1:k-4                                0.15             0.21
2:5-9                                0.26             0.29
3:wykształcenie średnie              0.25             0.25
4:wykształcenie wyższe               0.33             0.24
```

W powyższych krokach pobrałem tabelę z bazy danych SQL, załadowałem pobrane dane do *pandas* i wykonałem wstępne sprawdzenie i oczyszczenie danych.

Jak to działa...

Ponieważ dane z systemów biznesowych mają zwykle lepszą strukturę niż pliki CSV lub Excela, w ich przypadku nie musimy wykonywać czynności takich jak pomijanie wierszy lub zajmowanie się różnymi typami danych w kolumnie. Jednak nawet w przypadku takich danych przed rozpoczęciem analizy eksploracyjnej konieczne będą pewne modyfikacje. Często w danych znajduje się więcej kolumn, niż potrzeba, niektóre z nazw są nieintuicyjne lub dane nie są uporządkowane w sposób najlepszy z punktu widzenia analizy. Informacje o znaczeniu wielu wartości nie są przechowywane w tabeli, aby uniknąć błędów przy wprowadzaniu danych i zaoszczędzić miejsce w pamięci. Na przykład na potrzeby informacji o wykształceniu matki w bazie przechowywana jest liczba 3, a nie fraza `wykształcenie wyższe`. Dobrym pomysłem jest odtworzenie bardziej znaczącego kodowania na jak najwcześniejszym etapie procesu oczyszczania danych.

Do pobrania danych z bazy SQL potrzebny będzie obiekt reprezentujący połączenie, który uwierzytelni nas na serwerze, oraz łańcuch znaków zawierający zapytanie SQL. Obiekt ten należy przekazać do funkcji `read_sql`, która umożliwia pobranie danych i ich wczytanie do ramki danych. Do pobrania danych zwykle stosuję instrukcję `SELECT`, dzięki której mogę oczyścić nazwy kolumn już na etapie pobierania danych. Czasami na tym etapie zmieniam również kolejność kolumn, ale w tym przykładzie robię to w dalszej części receptury.

W *kroku 5.* po potwierdzeniu, że każdy wiersz zawiera unikalną wartość atrybutu `studentid`, wykorzystuję ten atrybut jako indeks. Jest to szczególnie ważne podczas pracy z danymi biznesowymi, ponieważ w ich przypadku prawie zawsze będziemy musieli scalić pobrane dane z innymi danymi w systemie. Chociaż indeks nie jest wymagany w tym konkretnym przypadku, jego ustawienie pozwoli lepiej przygotować się do trudnego zadania łączenia danych. Ustalenie indeksu prawdopodobnie zwiększy również wydajność procesu łączenia danych.

Za pomocą funkcji `count` można sprawdzić, czy w danych nie brakuje jakichś wartości. W przypadku braku danych funkcja ta zwróci wartość `395` (liczba wierszy) dla każdej kolumny. Niestety sytuacja taka jest zbyt piękna, aby mogła być prawdziwa. W tabeli mogą istnieć wartości, które reprezentują brak — to znaczy prawidłowe liczby, które mimo wszystko oznaczają brak wartości, np. `-1`, `0`, `9` lub `99`. Taką możliwością zajmuję się w następnym kroku.

Krok 7. przedstawia przydatną technikę zastępowania wartości w wielu kolumnach. Tworzę w nim słownik odwzorowujący oryginalne wartości w każdej kolumnie na nowe. Do zamiany wartości służy metoda `replace`. Aby zmniejszyć ilość miejsca zajmowanego przez nowe, znaczące wartości, zmieniam typ danych w tych kolumnach na `category`. Zmiany dokonuję poprzez wygenerowanie listy kluczy ze słownika `setvalues`. Wywołanie `setvalueskeys = [k for k in setvalues]` zwraca listę `[famrel, freetime, goout, mothereducation, fathereducation]`. Następnie przechodzę pętlą przez te pięć kolumn i za pomocą metody `astype` zmieniam typ danych na `category`. Zauważ, że ilość miejsca w pamięci, jaką zajmują te kolumny, zostaje znacznie zmniejszona.

Na koniec sprawdzam poprawność przypisania nowych wartości za pomocą funkcji `value_counts`, która pozwala wyświetlić procentowy udział poszczególnych wartości. Korzystam z `apply`, ponieważ chcę wywołać `value_counts` na wielu kolumnach. Aby uniknąć sortowania wyniku według częstości, ustawiam wartość parametru `sort` na `False`.

Metoda `replace` ramki danych jest również przydatnym narzędziem do radzenia sobie z logicznymi brakami wartości, które nie zostaną rozpoznane podczas pobierania danych przez funkcję `read_sql`. Zera w kolumnach `mothereducation` i `fathereducation` wydają się należeć do tej kategorii. Problem ten rozwiązuję poprzez wskazanie w słowniku `setvalues`, że zera w tych kolumnach powinny zostać zastąpione przez `NaN`. Ważne jest, aby zająć się tego rodzaju brakami zaraz po pierwszym imporcie danych. Tego typu braki nie zawsze są oczywiste i mogą znacząco wpłynąć na wszystkie późniejsze prace.

Użytkownicy pakietów takich jak SPPS, SAS i R zauważą różnicę pomiędzy tym podejściem a etykietami wartości w SPSS i R oraz `format proc` w SAS. W *pandas* do zwiększenia czytelności wartości konieczne są zmiany w rzeczywistych danych. Zużycie pamięci możemy zmniejszyć, zmieniając typ takich kolumn na `category` (w języku R jest podobnie, w nim do tego celu służą tzw. czynniki [ang. *factors*]).

Zobacz również...

Przeniosłem dane o ocenach bliżej początku ramki danych. Uważam, że pomocne jest umieszczenie potencjalnych zmiennych docelowych lub zależnych w skrajnych lewych kolumnach. Dzięki temu interesujące wartości będące przedmiotem badania są widoczne „od razu". Pomocne jest również umieszczanie obok siebie podobnych kolumn. W tym przykładzie osobnicze zmienne demograficzne (`sex`, `age`) znajdują się obok siebie, podobnie jak zmienne związane z rodziną (`mothereducation`, `fathereducation`) oraz sposobem spędzania czasu przez uczniów (`traveltime`, `studytime` i `freetime`).

Zamiast funkcji `replace` w *kroku 7.* możesz zastosować `map`. Przed pojawieniem się *pandas* 19.2 wywołanie `map` było znacznie wydajniejsze. Od wersji 19.2 różnica w wydajności jest znacznie mniejsza. Jeśli pracujesz z bardzo dużym zbiorem danych, różnica może być nadal wystarczająca, aby rozważyć użycie `map`.

Co dalej?

Receptury zawarte w rozdziale 8., „Rozwiązywanie problemów z danymi podczas łączenia ramek danych", zawierają szczegółowe informacje na temat łączenia danych. W rozdziale 4., „Identyfikacja brakujących i odstających wartości w podzbiorach danych", przyjrzymy się bliżej relacjom pomiędzy dwoma i większą liczbą zmiennych. W kolejnych recepturach w tym rozdziale pokażę Ci, jak wykorzystać te same podejścia podczas pracy z danymi z pakietów SPSS, SAS i R.

Importowanie danych z SPSS, Stata i SAS

Do wczytania danych z trzech popularnych pakietów statystycznych wykorzystam pakiet *pyreadstat*. Główną jego zaletą jest możliwość importu danych z pakietów statystycznych bez utraty metadanych, takich jak nazwy zmiennych i etykiety.

Często zdarza się, że trafiające do Ciebie dane zawarte w plikach SPSS, Stata i SAS to dane przetworzone w takim stopniu, że nie występują już w nich problemy znane z plików CSV i Excela oraz baz danych SQL. W plikach tych zazwyczaj nie występują znane z plików CSV i Excel kolumny o nieprawidłowych nazwach, niepoprawne typy danych i braki w wartościach, a same dane nie są oderwane od związanej z nimi logiki biznesowej (np. poprzez zastosowanie specyficznego kodowania, które często pojawia się w pracy z bazami SQL). Gdy ktoś lub jakaś firma udostępnia nam plik z jednego z tych pakietów, to często do danych dodaje etykiety zmiennych (ang. *variable label*) i etykiety wartości kategorialnych (ang. *value label*). Na przykład hipotetyczna kolumna o nazwie `ocena_prezentacji` ma przypisaną etykietę `ogólny poziom zadowolenia z prezentacji`. Wartości w tej kolumnie pochodzą z zakresu `1` – `5`, przy czym `1` oznacza bardzo niezadowolony, a `5` bardzo zadowolony.

Zachowanie tych metadanych podczas importowania danych do *pandas* jest wyzwaniem. W bibliotece tej nie ma dokładnego odpowiednika etykiet zmiennych i wartości, a wbudowane narzędzia do importowania danych z SAS, Stata i SAS nie importują metadanych. W tej recepturze wykorzystam *pyreadstat* do wczytania informacji o zmiennych i etykietach wartości oraz pokażę Ci kilka technik reprezentowania tych informacji w *pandas*.

Przygotuj się

W tej recepturze zakładam, że masz zainstalowany pakiet *pyreadstat*. Jeśli nie, to możesz go zainstalować za pomocą narzędzia `pip`. W terminalu lub w PowerShellu (w systemie Windows) wpisz `pip install pyreadstat`. Do uruchomienia kodu potrzebne będą pliki z danymi z SPSS, Stata i SAS.

Będziemy pracowali z danymi z United States National Longitudinal Survey of Youth (NLS).

> National Longitudinal Survey of Youth („narodowe badania przekrojowe młodzieży") to badanie przeprowadzone przez Biuro Statystyk Pracy Stanów Zjednoczonych. Badanie rozpoczęto w 1997 roku, a jego przedmiotem była grupa osób urodzonych w latach 1980 – 1985. Przez kolejne lata, aż do 2017 roku, powtarzano badania na tej grupie osób. Na potrzeby tej receptury z setek zmiennych dostępnych w podanym badaniu wyodrębniłem 42 zmienne dotyczące ocen, zatrudnienia, dochodów i postaw wobec rządu. Oddzielne pliki dla SPSS, Stata i SAS można pobrać z repozytorium. Kompletne wyniki badania można pobrać ze strony *https://www.nlsinfo.org/*.

Jak to zrobić...

Zaimportujemy dane z SPSS, Stata i SAS. Zachowamy metadane, takie jak etykiety wartości:

1. Zaimportuj *pandas*, NumPy i *pyreadstat*.

   W tym kroku zakładam, że masz zainstalowany pakiet *pyreadstat*:

   ```
   >>> import pandas as pd
   >>> import numpy as np
   >>> import pyreadstat
   ```

2. Pobierz dane z SPSS.

   Przekaż ścieżkę i nazwę pliku do metody `read_sav` z pakietu *pyreadstat*. Wyświetl kilka pierwszych wierszy i rozkład wartości. Zauważ, że nazwy kolumn i etykiety wartości nie mają opisowego charakteru oraz że wywołanie `read_sav` zwraca zarówno ramkę danych, jak i obiekt meta:

   ```
   >>> nls97spss, metaspss = pyreadstat.read_sav('dane/nls97.sav')
   >>> nls97spss.dtypes
   R0000100    float64
   R0536300    float64
   R0536401    float64
   ...
   ```

```
U2962900    float64
U2963000    float64
Z9063900    float64
dtype: object

>>> nls97spss.head()
   R0000100  R0536300  ...  U2963000  Z9063900
0       1.0       2.0  ...       NaN      52.0
1       2.0       1.0  ...       6.0       0.0
2       3.0       2.0  ...       6.0       0.0
3       4.0       2.0  ...       6.0       4.0
4       5.0       1.0  ...       5.0      12.0

[5 rows x 42 columns]
>>> nls97spss['R0536300'].value_counts(normalize=True)
1.0    0.51191
2.0    0.48809
Name: R0536300, dtype: float64
```

3. Wykorzystaj metadane do poprawy etykiet kolumn i wartości.

   Obiekt `metaspss` zwrócony przez funkcję `read_sav` zawiera etykiety kolumn i etykiety wartości z pliku SPSS. Za pomocą słownika `variable_value_labels` zmapuj wartości na etykiety w jednej kolumnie (`R0536300`). (Działanie takie nie zmienia danych, poprawia tylko sposób ich wyświetlenia w wywołaniu `value_counts`). Do rzeczywistego ustalenia etykiet wykorzystaj metodę `set_value_labels`:

```
>>> metaspss.variable_value_labels['R0536300']
{0.0: 'No Information', 1.0: 'Male', 2.0: 'Female'}

>>> nls97spss['R0536300'].\
...   map(metaspss.variable_value_labels['R0536300']).\
...   value_counts(normalize=True)
Male      0.51191
Female    0.48809
Name: R0536300, dtype: float64

>>> nls97spss = pyreadstat.set_value_labels(nls97spss, metaspss,
↪formats_as_category=True)
```

4. Wykorzystaj etykiety kolumn z metadanych do zmiany nazwy kolumn.

   Aby zastosować etykiety kolumn z `metaspss` w ramce danych, wystarczy przypisać je do nazw kolumn w ramce. Uporządkuj nieco nazwy kolumn: zmień litery na małe, zamień spacje na podkreślenia i usuń wszystkie pozostałe znaki niealfanumeryczne:

```
>>> nls97spss.columns = metaspss.column_labels
>>> nls97spss['KEY!SEX (SYMBOL) 1997'].value_counts(normalize=True)
Male      0.51191
Female    0.48809
Name: KEY!SEX (SYMBOL) 1997, dtype: float64

>>> nls97spss.dtypes
PUBID - YTH ID CODE 1997                 float64
KEY!SEX (SYMBOL) 1997                  category
```

```
KEY!BDATE M/Y (SYMBOL) 1997                    float64
KEY!BDATE M/Y (SYMBOL) 1997                    float64
CV_SAMPLE_TYPE 1997                           category
KEY!RACE_ETHNICITY (SYMBOL) 1997              category
...
HRS/WK R WATCHES TELEVISION 2017              category
HRS/NIGHT R SLEEPS 2017                        float64
CVC_WKSWK_YR_ALL L99                           float64
dtype: object

>>> nls97spss.columns = nls97spss.columns.\
...    str.lower().\
...    str.replace(' ','_').\
...    str.replace('[^a-z0-9_]', '')
>>> nls97spss.set_index('pubid__yth_id_code_1997', inplace=True)
```

5. Uprość proces, ustal etykiety wartości już w momencie importu.

Etykiety można dodać już w początkowym wywołaniu funkcji `read_sav`. Wystarczy ustawić wartość parametru `apply_value_formats` na `True`. Takie podejście eliminuje potrzebę późniejszego wywoływania funkcji `set_value_labels`:

```
>>> nls97spss, metaspss = pyreadstat.read_sav('dane/nls97.sav', apply_
↪value_formats=True, formats_as_category=True)
>>> nls97spss.columns = metaspss.column_labels
>>> nls97spss.columns = nls97spss.columns.\
...   str.lower().\
...   str.replace(' ','_').\
...   str.replace('[^a-z0-9_]', '')
```

6. Wyświetl wszystkie kolumny i kilka wierszy:

```
>>> nls97spss.dtypes
pubid__yth_id_code_1997                        float64
keysex_symbol_1997                            category
keybdate_my_symbol_1997                        float64
keybdate_my_symbol_1997                        float64
cv_sample_type_1997                           category
keyrace_ethnicity_symbol_1997                 category
...
hrsnight_r_sleeps_2017                         float64
cvc_wkswk_yr_all_l99                           float64
dtype: object

>>> nls97spss.head()
   pubid__yth_id_code_1997 keysex_symbol_1997  ...  hrsnight_r_sleeps_2017  \
0                      1.0             Female  ...                     NaN
1                      2.0               Male  ...                     6.0
2                      3.0             Female  ...                     6.0
3                      4.0             Female  ...                     6.0
4                      5.0               Male  ...                     5.0

   cvc_wkswk_yr_all_l99
0                  52.0
1                   0.0
2                   0.0
```

```
3                   4.0
4                  12.0

[5 rows x 42 columns]
```

7. Na jednej z kolumn wywołaj metodę `frequencies` i ustaw indeks:

```
>>> nls97spss.govt_responsibility__provide_jobs_2006.\
...   value_counts(sort=False)
Definitely should be        454
Definitely should not be    300
Probably should be          617
Probably should not be      462
Name: govt_responsibility__provide_jobs_2006, dtype: int64

>>> nls97spss.set_index('pubid__yth_id_code_1997', inplace=True)
```

8. Zaimportuj dane ze Stata, zastosuj etykiety wartości i popraw nagłówki kolumn. Zastosuj te same metody co w przypadku danych z SPSS:

```
>>> nls97stata, metastata = pyreadstat.read_dta('dane/nls97.dta', apply_
↪value_formats=True, formats_as_category=True)
>>> nls97stata.columns = metastata.column_labels
>>> nls97stata.columns = nls97stata.columns.\
...     str.lower().\
...     str.replace(' ','_').\
...     str.replace('[^a-z0-9_]', '')
>>> nls97stata.dtypes
pubid__yth_id_code_1997                      float64
keysex_symbol_1997                          category
keybdate_my_symbol_1997                      float64
keybdate_my_symbol_1997                      float64
...
hrsnight_r_sleeps_2017                       float64
cvc_wkswk_yr_all_l99                         float64
dtype: object
```

9. Wyświetl kilka wierszy i wywołaj funkcję `frequency`:

```
>>> nls97stata.head()
   pubid__yth_id_code_1997 keysex_symbol_1997  ...  hrsnight_r_sleeps_2017  \
0                      1.0             Female  ...                    -5.0
1                      2.0               Male  ...                     6.0
2                      3.0             Female  ...                     6.0
3                      4.0             Female  ...                     6.0
4                      5.0               Male  ...                     5.0

   cvc_wkswk_yr_all_l99
0                  52.0
1                   0.0
2                   0.0
3                   4.0
4                  12.0

[5 rows x 42 columns]
>>> nls97stata.govt_responsibility__provide_jobs_2006.\
  value_counts(sort=False)
-5.0                        1425
```

```
-4.0                        5665
-2.0                          56
-1.0                           5
Definitely should be         454
Definitely should not be     300
Probably should be           617
Probably should not be       462
Name: govt_responsibility__provide_jobs_2006, dtype: int64
```

10. Dopisz brakujące wartości w danych ze Stata i ustaw indeks:

```
>>> nls97stata.min()
pubid__yth_id_code_1997          1.0
keybdate_my_symbol_1997          1.0
keybdate_my_symbol_1997       1980.0
...
cv_ba_credits_l1_2011           -5.0
cv_bio_child_hh_2017            -5.0
cv_bio_child_nr_2017            -5.0
hrsnight_r_sleeps_2017          -5.0
cvc_wkswk_yr_all_l99            -4.0
dtype: float64

>>> nls97stata.replace(list(range(-9,0)), np.nan, inplace=True)

>>> nls97stata.min()
pubid__yth_id_code_1997          1.0
keybdate_my_symbol_1997          1.0
keybdate_my_symbol_1997       1980.0
...
cv_bio_child_hh_2017             0.0
cv_bio_child_nr_2017             0.0
hrsnight_r_sleeps_2017           0.0
cvc_wkswk_yr_all_l99             0.0
dtype: float64
>>> nls97stata.set_index('pubid__yth_id_code_1997', inplace=True)
>>>
```

11. Pobierz dane z SAS. Użyj pliku katalogowego SAS dla etykiet wartości:

Etykiety wartości są przechowywane w pliku katalogowym SAS (ang. *catalog file*). Przekazanie ścieżki i nazwy pliku katalogowego powoduje pobranie etykiet wartości i zastosowanie ich na danych:

```
>>> nls97sas, metasas = pyreadstat.read_sas7bdat('dane/nls97.sas7bdat',
↪catalog_file='dane/nlsformats3.sas7bcat', formats_as_category=True)
>>> nls97sas.columns = metasas.column_labels
>>>
>>> nls97sas.columns = nls97sas.columns.\
...     str.lower().\
...     str.replace(' ','_').\
...     str.replace('[^a-z0-9_]', '')
>>>
>>> nls97sas.head()
   pubid__yth_id_code_1997 keysex_symbol_1997  ...  hrsnight_r_sleeps_2017  \
0                      1.0             Female  ...                     NaN
1                      2.0               Male  ...                     6.0
```

```
2                    3.0            Female  ...                    6.0
3                    4.0            Female  ...                    6.0
4                    5.0              Male  ...                    5.0

   cvc_wkswk_yr_all_l99
0                  52.0
1                   0.0
2                   0.0
3                   4.0
4                  12.0

[5 rows x 42 columns]
>>> nls97sas.keysex_symbol_1997.value_counts()
Male      4599
Female    4385
Name: keysex_symbol_1997, dtype: int64
>>> nls97sas.set_index('pubid__yth_id_code_1997', inplace=True)
```

Dzięki tej recepturze zaimportujesz dane z SPSS, SAS i Stata bez utraty metadanych.

Jak to działa...

Metody `read_sav`, `read_dta` i `read_sas7bdat` z pakietu *pyreadstat* pozwalają wczytywać dane z plików SPSS, Stata i SAS. Metody te działają w podobny sposób. Podczas wczytywania danych wartości możliwe jest wykorzystanie etykiet wartości. W przypadku plików SPSS i Stata (*kroki 5.* i *8.*) wartości można zastąpić etykietami za pomocą parametru `apply_value_formats = True`. W przypadku plików SAS (*krok 11.*) należy podać ścieżkę i nazwę pliku katalogowego. Do zmiany typu danych w kolumnach, w których zostaną użyte etykiety wartości, wykorzystaj argument `functions_as_category = True`. Obiekt metadanych zawiera nazwy kolumn z pakietu statystycznego oraz ich etykiety, które można przypisać do kolumn w ramce danych w dowolnym momencie (`nls97spss.columns = metaspss.column_labels`). Możliwy jest nawet powrót do oryginalnych nazw kolumn po przypisaniu etykiet. Wystarczy nazwom kolumn w ramce danych przypisać ich nazwy z metadanych (`nls97spss.columns = metaspss.column_names`).

W *kroku 3.* odczytałem dane z pliku SPSS, ale nie wykorzystałem etykiet wartości. W kroku tym wyświetliłem słownik tylko dla jednej zmiennej (`metaspss.variable_value_labels['R0536300']`), ale możesz również wyświetlić wartości dla wszystkich zmiennych (`metaspss.variable_value_labels`). Gdy upewnisz się, że etykiety mają sens, wartości możesz zastąpić etykietami za pomocą funkcji `set_value_labels`. Warto zastosować to podejście, gdy nie znasz dobrze danych i chcesz sprawdzić etykiety przed ich zastosowaniem.

Etykiety kolumn z obiektu metadanych są często lepszym wyborem niż oryginalne nazwy kolumn. Nazwy kolumn mogą być dość zagadkowe, zwłaszcza gdy zawartość pliku SPSS, Stata lub SAS pochodzi z jakiegoś dużego badania (tak jak w tym przykładzie). Zazwyczaj etykiety kolumn nie są idealnymi kandydatami na ich nazwy. Etykiety zawierają czasami spacje, niezbyt pomocnie użytą kapitalizację i znaki niealfanumeryczne. Aby zamienić litery na małe, zastąpić spacje podkreśleniami i usunąć znaki inne niż alfanumeryczne, warto połączyć ze sobą kilka wywołań funkcji pracujących na łańcuchach znaków.

W przypadku plików tego rodzaju obsługa brakujących wartości nie zawsze jest prosta, ponieważ istnieje wiele powodów, dla których brakuje danych. Jeśli plik zawiera wyniki ankiety, brakująca wartość może wynikać z pominięcia pola w ankiecie, braku odpowiedzi respondenta, niepoprawnie udzielonej odpowiedzi i tak dalej. W badaniu NLS występuje dziewięć możliwych wartości braków (od `-1` do `-9`). Import danych z SPSS powoduje automatycznie zastąpienie tych wartości wartością `NaN`, podczas gdy import ze Stata powoduje ich zachowanie. (W przypadku importu z SPSS również można zachować te wartości, w takim przypadku należy skorzystać z parametru `user_missing = True`). W przypadku danych ze Stata musimy jawnie zażądać zamian wartości od `-1` do `-9` na wartość `NaN`. Można to zrobić za pomocą funkcji `replace`, której należy przekazać listę liczb całkowitych od `-9` do `-1` (`list(range(-9,0))`).

Zobacz również...

Być może zauważyłeś podobieństwa pomiędzy tą i poprzednią recepturą dotyczące sposobu ustalania etykiet wartości. Funkcja `set_value_labels` jest podobna do metody `replace`, którą wykorzystałem do ustawienia etykiet wartości w poprzedniej recepturze. Metodzie `replace` przekazałem słownik, który pozwolił zmapować wartości w kolumnach na odpowiadające im etykiety wartości. Funkcja `set_value_labels` w tej recepturze robi zasadniczo to samo, ale w roli słownika wykorzystuje pole `variable_value_labels` obiektu metadanych.

W jednym aspekcie dane z pakietów statystycznych często nie mają tak dobrej struktury jak bazy danych SQL. Ponieważ pakiety statystyczne mają na celu ułatwienie analizy, to często naruszają one reguły normalizacji baz danych. W danych tego typu często występuje niejawna struktura relacyjna, której odtworzenie może być konieczne na pewnym etapie prac. Odtworzenie struktury relacyjnej polega na cofnięciu operacji *spłaszczenia* danych. Przetwarzane zbiory mogą na przykład łączyć dane z poziomu indywidualnych jednostek z informacjami o zdarzeniach, na przykład wizyty osób i szpitale, niedźwiedzie brunatne i daty ich przebudzenia z hibernacji. W przypadku niektórych analiz wymagana będzie zmiana kształtu danych.

Co dalej?

Pakiet *pyreadstat* jest dobrze udokumentowany (*https://github.com/Roche/pyreadstat*). Pakiet umożliwia wybór kolumn na wiele różnych sposobów i obsługuje braki w danych. Ze względu na ograniczenie miejsca nie mogłem sobie pozwolić na ich prezentację w tej recepturze.

Importowanie danych z R

Do wczytania danych z R zastosuję pakiet *pyreadr*. Ponieważ pakiet ten nie pozwala przechwycić metadanych, stworzę kod, który zrekonstruuje etykiety wartości (czynniki z R) i nazwy kolumn. Kod ten będzie podobny do kodu z receptury „Importowanie danych z baz SQL".

Pakiet statystyczny R jest pod wieloma względami podobny do połączenia Pythona i *pandas* (przynajmniej w zakresie, w którym ich funkcjonalności się pokrywają). Oba rozwiązania zawierają świetne narzędzia do przygotowywania i analizy danych. Niektórzy analitycy danych korzystają zarówno z R, jak i z Pythona. W zależności od swoich preferencji część prac związanych z przetwarzaniem danych analitycy wykonują w Pythonie, a do analiz statystycznych stosują R (lub odwrotnie). Obecnie brakuje narzędzi do odczytu danych zapisanych w postaci plików *rds* lub *rdata* w Pythonie. Dlatego analitycy często zapisują dane jako pliki CSV, które następne wczytują w Pythonie. W tej recepturze skorzystam z pakietu *pyreadr*, który nie wymaga instalacji R. Twórca tego pakietu jest też twórcą pakietu *pyreadstat*.

Kiedy otrzymujemy plik z R lub pracujemy z takim, który sami stworzyliśmy, możemy liczyć na to, że będzie on dość dobrze skonstruowany (przynajmniej w porównaniu z plikami CSV i Excela). Każda kolumna będzie zawierała dane tylko jednego typu, nazwy kolumn będą zgodne ze składnią Pythona, a wszystkie wiersze będą miały tę samą strukturę. Może jednak zajść potrzeba przywrócenia części logiki kodowania, tak jak zrobiłem to podczas pracy z bazami SQL.

Przygotuj się

W tej recepturze zakładam, że masz zainstalowany pakiet *pyreadr*. Jeśli nie, to możesz go zainstalować za pomocą narzędzia `pip`. W terminalu lub w PowerShellu (Windows) wpisz `pip install pyreadr`. Do uruchomienia poniższego kodu potrzebny będzie plik *rds* z R.

W tym przepisie ponownie wykorzystam dane z National Longitudinal Survey.

Jak to zrobić...

Zaimportujemy dane z R bez utraty ważnych metadanych:

1. Zaimportuj *pandas*, NumPy, *pprint* i pakiet *pyreadr*:

```
>>> import pandas as pd
>>> import numpy as np
>>> import pyreadr
>>> import pprint
```

2. Pobierz dane z R.

   Aby wczytać dane z R do ramki danych *pandas*, metodzie `read_r` przekaż ścieżkę i nazwę pliku. Funkcja `read_r` może zwrócić jeden lub więcej obiektów. Podczas wczytywania pliku *rds* (w przeciwieństwie do pliku *rdata*) funkcja zwróci jeden obiekt, dostępny pod kluczem `None`. Aby uzyskać ramkę danych, należy się do niego odwołać w sposób pokazany poniżej:

```
>>> nls97r = pyreadr.read_r('dane/nls97.rds')[None]
>>> nls97r.dtypes
R0000100    int32
R0536300    int32
...
U2962800    int32
U2962900    int32
```

```
U2963000    int32
Z9063900    int32
Length: 42, dtype: object

>>> nls97r.head(10)
   R0000100  R0536300  ...  U2963000  Z9063900
0         1         2  ...        -5        52
1         2         1  ...         6         0
2         3         2  ...         6         0
3         4         2  ...         6         4
4         5         1  ...         5        12
5         6         2  ...         6         6
6         7         1  ...        -5         0
7         8         2  ...        -5        39
8         9         1  ...         4         0
9        10         1  ...         6         0

[10 rows x 42 columns]
```

3. Stwórz słowniki dla etykiet wartości i nazw kolumn.

   Wczytaj słownik zawierający mapowanie wartości na etykiety i utwórz listę preferowanych nazw kolumn:

```
>>> with open('dane/nlscodes.txt', 'r') as reader:
...     setvalues = eval(reader.read())
...
>>> pprint.pprint(setvalues)
{'R0536300': {0.0: 'No Information', 1.0: 'Male', 2.0: 'Female'},
 'R1235800': {0.0: 'Oversample', 1.0: 'Cross-sectional'},
 'S8646900': {1.0: '1. Definitely',
              2.0: '2. Probably ',
              3.0: '3. Probably not',
              4.0: '4. Definitely not'},
...
>>> newcols = ['personid','gender','birthmonth','birthyear',
...   'sampletype',  'category','satverbal','satmath',
...   'gpaoverall','gpaeng','gpamath','gpascience','govjobs',
...   'govprices','govhealth','goveld','govind','govunemp',
...   'govinc','govcollege','govhousing','govenvironment',
...   'bacredits','coltype1','coltype2','coltype3','coltype4',
...   'coltype5','coltype6','highestgrade','maritalstatus',
...   'childnumhome','childnumaway','degreecol1',
...   'degreecol2','degreecol3','degreecol4','wageincome',
...   'weeklyhrscomputer','weeklyhrstv',
...   'nightlyhrssleep','weeksworkedlastyear']
```

4. Ustaw etykiety wartości, obsłuż braki oraz zmień typ wybranych kolumn na `category`.

   Wykorzystaj słownik `setvalues` i zastąp istniejące wartości etykietami. Zastąp wszystkie wartości od `-9` do `-1` wartością `NaN`:

```
>>> nls97r.replace(setvalues, inplace=True)
>>> nls97r.head()
   R0000100 R0536300  ...  U2963000  Z9063900
0         1   Female  ...        -5        52
1         2     Male  ...         6         0
```

```
2         3   Female  ...         6         0
3         4   Female  ...         6         4
4         5     Male  ...         5        12

[5 rows x 42 columns]

>>> nls97r.replace(list(range(-9,0)), np.nan, inplace=True)

>>> for col in nls97r[[k for k in setvalues]].columns:
...     nls97r[col] = nls97r[col].astype('category')
...
>>> nls97r.dtypes
R0000100     float64
R0536300    category
R0536401     float64
R0536402     float64
R1235800    category
              ...
U2857300    category
U2962800    category
U2962900    category
U2963000     float64
Z9063900     float64
Length: 42, dtype: object
```

5. Nadaj kolumnom znaczące nazwy:

```
>>> nls97r.columns = newcols
>>> nls97r.dtypes

personid                float64
gender                 category
birthmonth              float64
birthyear               float64
sampletype             category
                          ...
wageincome             category
weeklyhrscomputer      category
weeklyhrstv            category
nightlyhrssleep         float64
weeksworkedlastyear     float64
Length: 42, dtype: object
```

W tej recepturze pokazałem, jak zaimportować plik z danymi z R do *pandas* oraz jak ustawić etykiety wartości w powstałej ramce danych.

Jak to działa...

Wczytywanie danych z R do ramki danych w *pandas* za pomocą *pyreadr* jest dość proste. Funkcji `read_r` należy przekazać jedynie nazwę pliku. Ponieważ funkcja w jednym wywołaniu `read_r` może zwrócić wiele obiektów, musimy określić, który obiekt nas interesuje. Podczas odczytu pliku *rds* (w przeciwieństwie do pliku *rdata*) zwracany jest tylko jeden obiekt. Znajduje się on pod kluczem `None`.

W *kroku 3.* wczytałem słownik, który mapuje wartości zmiennych na etykiety oraz listę preferowanych nazw kolumn. W *kroku 4.* ustaliłem etykiety wartości. W kolumnach, w których zastosowałem etykiety wartości, zmieniłem też typ danych na `category`. Zmianę przeprowadziłem w pętli, która przechodzi przez klucze ze słownika `setvalues` (`[k for k in setvalues]`).

W *kroku 5.* zmieniłem nazwy kolumn na bardziej intuicyjne. Zwróć uwagę, że istotna jest kolejność kolumn. Etykiety wartości należy zastosować przed zmianą nazw kolumn, ponieważ słownik `setvalues` jest oparty na oryginalnych nazwach kolumn.

Główną zaletą związaną z użyciem *pyreadr* do wczytania plików z R do ramki danych *pandas* jest to, że nie musimy najpierw konwertować danych R do pliku CSV. Po zaimplementowaniu procesu wczytywania w Pythonie i zmianie danych w R wystarczy po prostu ponownie uruchomić kod. Jest to szczególnie pomocne, gdy na swojej maszynie nie masz zainstalowanego tego języka.

Zobacz również...

Użycie pakietu *pyreadr* może zwrócić wiele ramek danych. Jest to przydatne, gdy w jednym pliku *rdata* zapisuje się kilka obiektów R. Aby wczytać je wszystkie, wystarczy tylko jedno wywołanie.

Pakiet *pprint* to poręczne narzędzie, które poprawia sposób wyświetlania zawartości słowników w Pythonie.

Co dalej?

Instrukcje i przykłady użycia *pyreadr* są dostępne pod adresem *https://github.com/ofajardo/pyreadr*.

Pliki *feather* to stosunkowo nowy format, który może być odczytany zarówno w R, jak i w Pythonie. Format ten omówię w następnej recepturze.

Do zaimportowania danych z R zamiast *pyreadr* można wykorzystać *rpy2*. Pakiet ten oferuje więcej możliwości niż *pyreadr*, ale wymaga zainstalowania R. Pakiet automatycznie rozpozna czynniki z R i ustawi je jako wartości w ramce danych. Spójrz na poniższy kod:

```
>>> import rpy2.robjects as robjects
>>> from rpy2.robjects import pandas2ri
>>> pandas2ri.activate()
>>> readRDS = robjects.r['readRDS']
>>> nls97withvalues = readRDS('dane/nls97withvalues.rds')

>>> nls97withvalues
      R0000100 R0536300  ...    U2963000  Z9063900
1            1   Female  ... -2147483648        52
2            2     Male  ...           6         0
3            3   Female  ...           6         0
```

```
4           4   Female  ...           6         4
5           5     Male  ...           5        12
...       ...      ...  ...         ...       ...
8980     9018   Female  ...           4        49
8981     9019     Male  ...           6         0
8982     9020     Male  ... -2147483648        15
8983     9021     Male  ...           7        50
8984     9022   Female  ...           7        20

[8984 rows x 42 columns]
```

W wyniku pojawiają się zaskakujące wartości `-2147483648`. Są one efektem interpretacji braków danych w kolumnach typu liczbowego przez funkcję `readRDS`. Globalne zastąpienie tej liczby wartością `NaN` byłoby dobrym krokiem — oczywiście po potwierdzeniu, że nie jest to prawidłowa wartość.

Przechowywanie danych tablicowych

Dane utrwala się poprzez ich kopiowanie z pamięci na dysk lokalny lub sieciowy z kilku powodów. Po pierwsze, aby mieć do nich dostęp bez konieczności powtarzania kroków, które wykorzystano do ich wygenerowania. Po drugie, aby udostępniać dane innym osobom lub wykorzystać je w innym oprogramowaniu. W tej recepturze zapiszemy dane przechowywane w ramce w różnych formatach plików (CSV, Excel *pickle* i *feather*).

Innym ważnym, ale czasami pomijanym powodem, dla którego zapisujemy dane, jest chęć zachowania pewnego ich fragmentu na potrzeby dalszej analizy. Być może przed zakończeniem analizy dane muszą zostać dokładnie zbadane przez inne osoby. Dla analityków, którzy pracują z danymi operacyjnymi w średnich i dużych firmach, proces ten jest częścią codziennej pracy związanej z oczyszczaniem danych.

Oprócz powodów, dla których utrwalamy dane, decyzje dotyczące tego, kiedy i jak serializować dane, są kształtowane przez kilka innych czynników, takich jak aktualny stan projektu, zasoby sprzętowe i oprogramowanie maszyn wykorzystywanych do zapisywania i ładowania danych oraz rozmiar samego zbioru. Podczas zapisu danych analitycy muszą być dużo bardziej świadomi tego, co robią, w porównaniu z zapisem tekstu w edytorze za pomocą kombinacji *Ctrl+S*.

Po zapisie dane są przechowywane w sposób niezależny od logiki, która została wykorzystana do ich stworzenia. Kwestię tę uważam za jedno z największych zagrożeń dla integralności całej analizy. Często wczytujemy dane, które zapisaliśmy w przeszłości (tydzień, miesiąc, a może rok temu?), zapominając o tym, jak zdefiniowaliśmy zmienną i jak jest ona powiązana z innymi. Jeżeli jesteś na etapie oczyszczania danych, to nie warto zajmować się ich zapisem (o ile Twoja stacja robocza i sieć poradzą sobie z obciążeniem związanym z ciągłym odtwarzaniem procesu przygotowywania danych). Dobrym pomysłem jest utrwalanie danych dopiero po osiągnięciu kamieni milowych w danym projekcie.

Poza pytaniem, *kiedy* utrwalać dane, pojawia się pytanie, *jak* to robić. Jeśli dane będą przechowywane w celu ich ponownego wykorzystania w tym samym oprogramowaniu, to warto zapisać je w formacie binarnym natywnym dla danego oprogramowania. W przypadku narzędzi takich jak SPSS, SAS, Stata i R nie jest to trudne, problem pojawia się w przypadku danych z *pandas*. W pewnym sensie jest to jednak dobra wiadomość. Mamy wiele możliwości zapisu, od plików CSV i Excela po wykorzystanie *pickle* i *feather*. W tej recepturze pokażę, jak zapisać dane we wszystkich tych formatach.

Przygotuj się

Jeżeli jeszcze tego nie zrobiłeś, to musisz zainstalować pakiet *feather*. Możesz to zrobić, wpisując w terminalu lub w PowerShellu (Windows) `pip install pyarrow`. Jeżeli w Twoim folderze z plikami z tego rozdziału nie ma jeszcze katalogu *widoki*, to do uruchomienia poniższego kodu będziesz musiał go utworzyć.

> Ten zestaw danych został pobrany ze zintegrowanej bazy danych Global Historical Climatology Network. Baza ta jest udostępniana przez United States National Oceanic and Atmospheric Administration. Znajdziesz ją pod adresem *https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/global-historical-climatology-network-monthly-version-4*. Wykorzystany przeze mnie plik to tylko 100 000 wierszy wybranych z pełnego zestawu danych.

Jak to zrobić...

Wczytamy plik CSV w *pandas*, a następnie zapiszemy powstałą ramkę danych jako plik *pickle* i plik *feather*. Podzbiory otrzymane z tych danych zapiszemy również w formatach CSV i Excel:

1. Zaimportuj *pandas* i PyArrow oraz dostosuj sposób wyświetlania danych.

   Pakiet PyArrow jest niezbędny do przeprowadzenia zapisu danych w formacie *feather*:

   ```
   >>> import pandas as pd
   >>> import pyarrow
   >>> pd.options.display.float_format = '{:,.2f}'.format
   >>> pd.set_option('display.width', 68)
   >>> pd.set_option('display.max_columns', 3)
   ```

2. Wczytaj plik CSV z temperaturami gruntu, pomiń wiersze z brakującymi wartościami i ustaw indeks:

   ```
   >>> landtemps = pd.read_csv('dane/landtempssample.csv',
   ...
   names=['ID_stacji','rok','miesiąc','średnia_temperatura','szerokość_geograficzna',
   ...
   'długość_geograficzna','wysokość','stacja','ID_państwa','nazwa_państwa'],
   ...     skiprows=1,
   ...     parse_dates=[['miesiąc','rok']],
   ```

```
...     low_memory=False)
>>> landtemps.rename(columns={'miesiąc_rok':'data_pomiaru'}, inplace=True)
>>> landtemps.dropna(subset=['średnia_temperatura'], inplace=True)

>>> landtemps.dtypes
data_pomiaru              datetime64[ns]
ID_stacji                         object
średnia_temperatura              float64
szerokość_geograficzna           float64
długość_geograficzna             float64
wysokość                         float64
stacja                            object
ID_państwa                        object
nazwa_państwa                     object
dtype: object
>>> landtemps.set_index(['data_pomiaru','ID_stacji'], inplace=True)
```

3. Zapisz skrajne wartości temperatury w plikach CSV i Excela.

   Za pomocą metody `quantile` wybierz wiersze z odstającymi wartościami (kwantyle 0,001 i 0,999):

```
>>> extremevals = landtemps[(landtemps.średnia_temperatura <
landtemps.średnia_temperatura.quantile(.001)) | (landtemps.średnia_temperatura
↪> landtemps.średnia_temperatura.quantile(.999))]
>>> extremevals.shape
(171, 7)
>>> extremevals.sample(7)
                          średnia_temperatura  ...  nazwa_państwa
data_pomiaru ID_stacji                         ...
1918-07-01   PKXLT869883                34.49  ...       Pakistan
2010-08-01   KU000405820                38.20  ...         Kuwait
2017-06-01   SAM00041140                34.80  ...   Saudi Arabia
1980-04-01   AYXLT433578               -35.70  ...     Antarctica
1971-12-01   RSM00025325               -35.38  ...         Russia
1990-01-01   CA002403854               -34.97  ...         Canada
1992-11-01   RSM00024266               -40.56  ...         Russia

[7 rows x 7 columns]
>>> extremevals.to_excel('widoki/tempext.xlsx')
>>> extremevals.to_csv('widoki/tempext.csv')
```

4. Zapisz dane w formacie *pickle* i *feather*.

   Przed zapisem w formacie *feather* należy zresetować indeks:

```
>>> landtemps.to_pickle('dane/landtemps.pkl')
>>> landtemps.reset_index(inplace=True)
>>> landtemps.to_feather("dane/landtemps.ftr")
```

5. Wczytaj pliki *pickle* i *feather*, które właśnie zapisałeś.

   Zauważ, że w przypadku formatu *feather* indeks został zachowany:

```
>> landtemps = pd.read_pickle('dane/landtemps.pkl')
>>> landtemps.head(2).T
data_pomiaru               2000-04-01  1940-05-01
ID_stacji                  USS0010K01S CI000085406
średnia_temperatura               5.27       18.04
```

```
szerokość_geograficzna           39.9      -18.35
długość_geograficzna           -110.75     -70.333
wysokość                        2773.7        58.0
stacja                     INDIAN_CANYON       ARICA
ID_państwa                            US          CI
nazwa_państwa              United States       Chile

>>> landtemps = pd.read_feather("dane/landtemps.ftr")
>>> landtemps.head(2).T
                                       0                    1
data_pomiaru           2000-04-01 00:00:00  1940-05-01 00:00:00
ID_stacji                      USS0010K01S          CI000085406
średnia_temperatura                  5.27                18.04
szerokość_geograficzna               39.9               -18.35
długość_geograficzna               -110.75               -70.333
wysokość                          2773.7                 58.0
stacja                       INDIAN_CANYON                ARICA
ID_państwa                              US                   CI
nazwa_państwa                United States                Chile
```

W powyższych krokach pokazałem, jak zapisywać ramki danych *pandas* w dwóch różnych formatach (*pickle*, *feather*).

Jak to działa...

Zapisywanie danych w *pandas* jest stosunkowo proste. Ramki danych oferują metody `to_csv`, `to_excel`, `to_pickle` i `to_feather`. Format *pickle* pozwala zachować też indeks.

Zobacz również

Zaletą jest to, że pliki *csv* nie są obciążone wieloma informacjami nadmiarowymi. Wadą tego formatu jest jego powolność i utrata ważnych metadanych, takich jak typy danych. (Funkcja `read_csv` często, ale nie zawsze potrafi określić typ danych podczas ponownego wczytywania pliku). Pliki *pickle* przechowują metadane, ale ich zapis może stanowić duże obciążenie w przypadku niewystarczających zasobów. Zapis w formacie *feather* zużywa mniej zasobów i można go wykorzystać zarówno w R, jak i w Pythonie. Jego użycie wiąże się jednak z utratą indeksu. Autorzy *feather* nie gwarantują też długoterminowego wsparcia.

Być może zauważyłeś, że poza wskazówką na temat wstrzymania się z zapisem do momentu osiągnięcia kamienia milowego nie rekomenduję żadnego sposobu zapisu danych. Wybór metody przechowywania danych to typowy przykład, w którym należy znaleźć właściwe narzędzie do właściwego zadania. Osobiście stosuję pliki CSV lub Excela, gdy chcę udostępnić kolegom fragment danych do przeanalizowania. Korzystam z *feather* w projektach w Pythonie — szczególnie gdy pracuję na maszynie z mniejszą ilością pamięcią RAM i przestarzałym procesorem lub w projektach, w których stosuję także R. Na końcowych etapach projektu stosuję format *pickle*.

Prawdziwą wartość mają tylko oczyszczone i spójne dane!

Przetwarzanie dużych ilości danych daje wiedzę, która leży u podstaw istotnych decyzji podejmowanych przez organizację. Pozwala to na uzyskiwanie znakomitych efektów: techniki wydobywania wiedzy z danych stają się coraz bardziej wyrafinowane. Podstawowym warunkiem sukcesu jest uzyskanie odpowiedniej jakości danych. Wykorzystanie niespójnych i niepełnych informacji prowadzi do podejmowania błędnych decyzji. Konsekwencją mogą być straty finansowe, stwarzanie konkretnych zagrożeń czy uszczerbek na wizerunku. A zatem oczyszczanie jest wyjątkowo ważną częścią analizy danych.

Ta książka jest praktycznym zbiorem gotowych do użycia receptur, podanych tak, aby maksymalnie ułatwić proces przygotowania danych do analizy. Omówiono tu takie kwestie dotyczące danych jak importowanie, ocena ich jakości, uzupełnianie braków, porządkowanie i agregacja, a także przekształcanie. Poza zwięzłym omówieniem tych zadań zaprezentowano najskuteczniejsze techniki ich wykonywania za pomocą różnych narzędzi: pandas, NumPy, Matplotlib czy SciPy. W ramach każdej receptury wyjaśniono skutki podjętych działań. Cennym uzupełnieniem jest zestaw funkcji i klas zdefiniowanych przez użytkownika, które służą do automatyzacji oczyszczania danych. Umożliwiają one też dostrojenie procesu do konkretnych potrzeb.

W książce znajdziesz receptury, dzięki którym:

- wczytasz i przeanalizujesz dane z różnych źródeł
- uporządkujesz dane, poprawisz ich błędy i uzupełnisz braki
- efektywnie skorzystasz z bibliotek Pythona
- zastosujesz wizualizacje do analizy danych
- napiszesz własne funkcje i klasy do automatyzacji procesu oczyszczania danych

Michael Walker — jest analitykiem danych. Od ponad trzydziestu lat zajmuje się tym zagadnieniem w różnych instytucjach edukacyjnych. Od 2006 roku prowadzi na wyższych uczelniach zajęcia z analizy danych, metod badawczych, statystyki i programowania. Poza tym tworzy raporty dla fundacji i sektora publicznego, a także publikuje analizy w czasopismach naukowych.

ISBN 978-83-283-8029-5
9 788328 380295
Cena: 79,00 zł

Packt