PYTHON
W DATA SCIENCE

PRAKTYCZNE WPROWADZENIE

YULI VASILIEV

Helion

no starch press

Tytuł oryginału: Python for Data Science: A Hands-On Introduction

Tłumaczenie: Piotr Rajca

ISBN: 978-83-289-1020-1

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

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

Spis treści

4

Korzystanie z danych z plików i API

Uzyskanie dostępu do danych i pobranie ich do skryptu jest pierwszym krokiem analizy danych. W tym rozdziale omówiono kilka sposobów importowania do aplikacji Pythona danych z plików i innych źródeł, a także sposoby eksportowania danych do plików. Dowiesz się w nim, jak uzyskać dostęp do zawartości różnych typów plików, w tym plików przechowywanych lokalnie na komputerze oraz plików dostępnych zdalnie za pośrednictwem żądań HTTP. Opisano w nim także, jak uzyskiwać dane, wysyłając żądania do interfejsów API dostępnych za pośrednictwem adresów URL. Wreszcie dowiesz się, jak wczytywać różne rodzaje danych do obiektów `DataFrame` biblioteki pandas.

Importowanie danych przy użyciu funkcji open() Pythona

Wbudowana funkcja `open()` Pythona służy do otwierania pliku dowolnego typu, pozwalając na wczytanie jego zawartości i jej przetworzenie w skrypcie. Funkcja zwraca obiekt `file`, który zawiera metody umożliwiające dostęp do zawartości pliku i wykonywanie na niej różnych operacji. Jeśli jednak plik zawiera dane w określonych formatach, takich jak CSV, JSON lub HTML, to w celu ich wczytania i manipulowania nimi należy również zaimportować odpowiednią bibliotekę. Przetwarzanie plików tekstowych nie wymaga żadnych specjalnych bibliotek; można po prostu polegać na metodach obiektu `file` zwracanego przez funkcję `open()`.

Pliki tekstowe

Pliki tekstowe (*.txt*) są prawdopodobnie najczęściej spotykanym typem plików. Dla Pythona plik tekstowy jest sekwencją obiektów reprezentujących łańcuchy znaków. Każdy taki obiekt to jeden wiersz zawartości pliku tekstowego, czyli sekwencja znaków zakończona niewyświetlanym znakiem nowego wiersza (`\n`) lub powrotu karetki.

Uwaga

Pojedynczy wiersz pliku tekstowego może być wyświetlany na ekranie w wielu wierszach, co zależy od szerokości okna, ale Python nadal będzie rozumiał go jako jeden wiersz, o ile nie zostanie on podzielony znakami nowego wiersza.

Python ma wbudowane funkcje do pracy z plikami tekstowymi, pozwalające na wykonywanie na nich operacji odczytu, zapisu i dołączania. W bieżącym punkcie rozdziału skoncentrujemy się na tym, jak odczytywać dane z plików tekstowych. Zaczniemy od wpisania poniższego fragmentu w edytorze tekstu i zapisania go w pliku *excerpt.txt*. Pamiętaj, aby dwukrotnie nacisnąć klawisz *ENTER* na końcu pierwszego akapitu, co pozwoli utworzyć pomiędzy kolejnymi akapitami pusty wiersz (ale nie naciskaj klawisza *ENTER*, aby podzielić długie wiersze tekstu).

```
Obecnie roboty mogą rozmawiać z ludźmi za pomocą języka naturalnego i stają się coraz
inteligentniejsze. Mimo to bardzo niewiele osób rozumie, jak działają te roboty lub jak mogą
wykorzystać te technologie we własnych projektach.
Przetwarzanie języka naturalnego (NLP) - gałąź sztucznej inteligencji, która pomaga maszynom
rozumieć ludzki język i reagować na niego - jest kluczową technologią, która leży u podstaw
każdego produktu cyfrowego asystenta.
```

Z naszego punktu widzenia fragment składa się z dwóch akapitów, które zawierają łącznie trzy zdania. Jednak dla Pythona fragment ten zawiera dwa niepuste wiersze tekstu oddzielone od siebie jednym pustym. Oto jak wczytać całą zawartość tego pliku do skryptu Pythona, a następnie ją wyświetlić:

```
path = "/ścieżka/do/pliku/excerpt.txt" ❶
with open(❷ path, ❸ "r") as ❹ f:
   content = f.read() ❺
print(content)
```

Zaczynamy od określenia ścieżki dostępu do pliku ❶. W swoim skrypcie będziesz musiał zastąpić */ścieżkę/do/pliku/excerpt.txt* odpowiednią, faktyczną ścieżką do pliku, której postać będzie zależna od tego, gdzie ten plik zapisałeś. Następnie przekazujemy tę ścieżkę dostępu do funkcji `open()` jako jej pierwszy parametr ❷. Drugi parametr funkcji `open()` kontroluje sposób użycia pliku. Domyślna wartość tego parametru sprawia, że plik zostanie otworzony w trybie odczytu tekstowego, co oznacza, że jego zawartość zostanie otworzona tylko do odczytu (a nie do edycji) i będzie traktowana jako łańcuch znaków. Możemy także jawnie użyć wartości `"r"` dla *odczytu* ❸, jednak nie jest to konieczne. Funkcja `open()` zwraca obiekt `file`, reprezentujący plik otworzony w określonym trybie ❹. Następnie możemy użyć metody `read()` zwróconego obiektu `file`, aby odczytać całą zawartość pliku ❺.

Użycie słowa kluczowego `with` z funkcją `open()` gwarantuje, że obiekt `file` zostanie prawidłowo zamknięty po zakończeniu pracy, nawet jeśli będzie zgłoszony wyjątek. W przeciwnym razie konieczne byłoby wywołanie funkcji `f.close()` w celu zamknięcia obiektu `file` i zwolnienia zużywanych przez niego zasobów systemowych.

Poniższy fragment kodu odczytuje zawartość tego samego pliku */ścieżka/do/pliku/excerpt.txt* wiersz po wierszu, jednak wyświetla na ekranie tylko niepuste wiersze:

```
path = "/ścieżka/do/pliku/excerpt.txt"
with open(path,"r") as f:
  for i, line in enumerate(f): ❶
    if line.strip(): ❷
      print(f"Wiersz {i}: ", line.strip())
```

W tym przykładzie do każdego wyświetlanego wiersza tekstu z pliku dodawany jest jego numer, generowany przy użyciu funkcji `enumerate()` ❶. Następnie używając metody `strip()` ❷, która usuwa wszelkie białe znaki z początku i końca obiektu łańcucha znaków, odrzucamy puste wiersze tekstu. Drugi, pusty wiersz z pliku tekstowego zawiera tylko jeden znak, znak nowego wiersza, który funkcja `strip()` usuwa. Oznacza to, że zawartość tego drugiego wiersza tekstu stanie się pustym łańcuchem znaków, który instrukcja `if` potraktuje jako logiczny fałsz i pominie. A zatem dane wynikowe tego skryptu będą mieć następującą postać (jak widać, nie ma w nich wiersza `1`):

```
Wiersz 0:  Obecnie roboty mogą rozmawiać z ludźmi za pomocą języka naturalnego i stają się
coraz inteligentniejsze. Mimo to bardzo niewiele osób rozumie, jak działają te roboty
lub jak mogą wykorzystać te technologie we własnych projektach.
Wiersz 2:  Przetwarzanie języka naturalnego (NLP) - gałąź sztucznej inteligencji, która
pomaga maszynom rozumieć ludzki język i reagować na niego - jest kluczową technologią,
która leży u podstaw każdego produktu cyfrowego asystenta.
```

Zamiast wyświetlać wiersze tekstu na ekranie, możemy przekazać je do listy, używając w tym celu listy składanej:

```
path = "/ścieżka/do/pliku/excerpt.txt"
with open(path,"r") as f:
  lst = [line.strip() for line in f if line.strip()]
```

W tym przypadku każdy niepusty wiersz tekstu stanie się oddzielnym elementem listy.

Pliki z danymi tabelarycznymi

Plik danych tabelarycznych (ang. *tabular data file*) to plik, w którym dane są podzielone na wiersze. Każdy wiersz zazwyczaj zawiera informacje o kimś lub o czymś, jak pokazano poniżej:

```
Jurek Rumowicz, jurek.rumowicz, sprzedaż
Janina Borewicz, janina.borewicz, sprzedaż
```

Jest to przykład **pliku płaskiego** (ang. *flat file*), najpopularniejszego typu plików z danymi tabelarycznymi. Nazwa pochodzi od struktury: pliki płaskie zawierają rekordy o prostej (płaskiej) strukturze, co oznacza, że rekordy nie zawierają struktur zagnieżdżonych ani podrekordów. Zazwyczaj plik płaski jest plikiem tekstowym w formacie CSV (z wartościami rozdzielonymi przecinkami) lub TSV (z wartościami rozdzielonymi znakami tabulacji), zawierającym rekordy, z których każdy jest zapisany w odrębnym wierszu. W plikach *.csv* wartości w rekordzie są oddzielone przecinkami, podczas gdy pliki *.tsv* jako separatorów używają znaków tabulacji. Oba formaty są szeroko obsługiwane i często używane do przenoszenia danych tabelarycznych między różnymi aplikacjami.

Poniżej został przedstawiony przykład korzystający z danych w formacie CSV, przy czym pierwszy wiersz pliku zawiera nagłówki opisujące zawartość kolejnych wierszy. Opisy nagłówków są używane jako *klucze* do danych zapisanych w kolejnych wierszach. Skopiuj poniższe dane do edytora tekstu i zapisz jako plik *cars.csv*:

```
Rok,Marka,Model,Cena
1997,Ford,E350,3200.00
1999,Chevy,venture,4800.00
1996,Jeep,grand cherokee,4900.00
```

Funkcja `open()` Pythona pozwala otwierać pliki *.csv* w trybie tekstowym. Następnie można wczytać dane do obiektu Pythona, używając do tego celu funkcji z modułu `csv` w sposób przedstawiony poniżej:

```
import csv
path = "/ścieżka/do/pliku/cars.csv"
with open(path, "r") as csv_file: ❶
  csv_reader =   csv.DictReader(csv_file) ❷
```

```
  cars = []
  for row in csv_reader:
    cars.append(dict(row)) ❸
print(cars)
```

Funkcja `open()` zwraca obiekt `file` ❶, który jest przekazywany do funkcji wczytującej dane CSV zdefiniowanej w module `csv`. W tym przypadku stosujemy metodę `DictReader()` ❷ modułu `csv`, która odwzorowuje dane w każdym wierszu w formie słownika, używając nagłówków z pierwszego wiersza jako kluczy słownika. Słowniki te są dołączane do listy ❸. W naszym przykładzie wynikowa lista słowników będzie miała następującą postać:

```
[
 {'Rok': '1997', 'Marka': 'Ford', 'Model': 'E350', 'Cena': '3200.00'},
 {'Rok': '1999', 'Marka': 'Chevy', 'Model': 'venture', 'Cena': '4800.00'},
 {'Rok': '1996', 'Marka': 'Jeep', 'Model': 'grand cherokee', 'Cena': '4900.00'}
]
```

Alternatywnie można użyć metody `reader()` modułu `csv`, aby przekształcić plik *.csv* w listę list, w której wewnętrzna lista będzie reprezentowała wiersz, w tym także wiersz z nagłówkami:

```
import csv
path = "cars.csv"
with open(path, "r") as csv_file:
  csv_reader = csv.reader(csv_file)
  cars = []
  for row in csv_reader:
    cars.append(row)
print(cars)
```

A oto wyniki wygenerowane przez ten skrypt:

```
[
  ['Rok', 'Marka', 'Model', 'Cena'],
  ['1997', 'Ford', 'E350', '3200.00'],
  ['1999', 'Chevy', 'venture', '4800.00'],
  ['1996', 'Jeep', 'grand cherokee', '4900.00']
]
```

Metody `csv.DictReader()` i `csv.reader()` mają opcjonalny parametr `delimiter`, umożliwiający określenie znaku separatora oddzielającego poszczególne pola danych. Parametr ten jest domyślnie ustawiony na przecinek, co jest idealne dla plików *.csv*. Natomiast w przypadku przypisania mu znaku tabulacji, `delimiter = "\t"`, można zamiast tego odczytywać dane oddzielone tabulatorami, zapisywane w plikach *.tsv*.

ĆWICZENIE 5. OTWIERANIE PLIKÓW JSON

Aby wczytać plik JSON, należy otworzyć go w trybie tekstowym za pomocą funkcji `open()`, a następnie używać modułu `json` do dalszego przetwarzania zawartości pliku. Podobnie jak `csv`, także `json` jest wbudowanym pakietem Pythona, więc nie trzeba go instalować osobno. Przykład użycia modułu `json` został przedstawiony w rozdziale 3. przy okazji opisywania konwersji dokumentu JSON na obiekt biblioteki pandas. W tym ćwiczeniu użyjesz modułu `json`, aby zapisać poniższy tekst w pliku *.json*:

```
{"cars":
  [{"Rok": "1997", "Marka": "Ford", "Model": "E350", "Cena": "3200.00"},
   {"Rok": "1999", "Marka": "Chevy", "Model": "venture", "Cena": "4800.00"},
   {"Rok": "1996", "Marka": "Jeep", "Model": "grand cherokee", "Cena": "4900.00"}
]}
```

Otwórz plik do odczytu za pomocą funkcji `open()` i przekaż pobrany obiekt `file` do metody `json.load()`, która przeprowadza deserializację kodu JSON do postaci obiektu Pythona. Z tego obiektu wyodrębnij część zawierającą wiersze samochodów. W pętli przeglądnij wszystkie te wiersze i wyświetl ich zawartość w następujący sposób:

```
Rok: 1997
Marka: Ford
Model: E350
Cena: 3200.00

Rok: 1999
Marka: Chevy
Model: venture
Cena: 4800.00

Rok: 1996
Marka: Jeep
Model: grand cherokee
Cena: 4900.00
```

Pliki binarne

Pliki tekstowe nie są jedynym typem plików, z którymi możemy mieć do czynienia. Istnieją również pliki wykonywalne (*.exe*) i pliki graficzne (*.jpeg*, *.bmp* itd.), które zawierają dane w formacie binarnym, reprezentowane jako sekwencja bajtów. Ponieważ bajty te są zwykle interpretowane jako coś innego niż znaki, nie można otworzyć pliku binarnego w trybie tekstowym, aby uzyskać dostęp do jego zawartości i nią manipulować. W przypadku takich plików należy użyć trybu binarnego funkcji `open()`.

Poniższy przykład pokazuje, jak otworzyć plik obrazu w trybie binarnym. Próba otworzenia takiego pliku w trybie tekstowym doprowadziłaby do wystąpienia błędu. Poniższego kodu można użyć do otworzenia dowolnego pliku *.jpg*:

```
image = "/ścieżka/do/pliku.jpg"
with open(image, "rb") as image_file: ❶
  content =   image_file.read() ❷
print(len(content)) ❸
```

W tym przypadku nakazujemy, by funkcja `open()` otworzyła plik do odczytu w trybie binarnym. Określa to łańcuch `"rb"` przekazany jako drugi parametr funkcji ❶. Obiekt zwrócony przez wywołanie funkcji `open()`, podobnie jak obiekt zwracany w przypadku otwierania pliku w trybie tekstowym, udostępnia metodę `read()`, która umożliwia pobranie zawartości pliku ❷. Tutaj zawartość jest pobierana jako obiekt `bytes`. W tym przykładzie ograniczamy się do wyświetlenia liczby bajtów odczytanych z pliku ❸.

Eksportowanie danych do plików

Po zakończeniu przetwarzania konieczne może być zapisanie danych w pliku, aby można było ich użyć podczas kolejnego wykonania skryptu lub zaimportować do innych skryptów czy aplikacji. Konieczne może być również zapisanie informacji w pliku, aby można było je wyświetlić samemu lub udostępnić innym osobom. Na przykład możesz chcieć rejestrować informacje o błędach i wyjątkach generowanych przez aplikację, by później je przeanalizować.

Z poziomu skryptu Pythona można utworzyć nowy plik i zapisać w nim dane; można także zapisać dane w istniejącym pliku. Przeanalizujemy tutaj przykład tej drugiej możliwości. Wracając do przykładu z punktu „Pliki z danymi tabelarycznymi”, załóżmy, że musimy zmodyfikować wiersz w pliku *cars.csv*, zmieniając cenę określonego samochodu. Przypomnijmy, że dane z pliku *cars.csv* zostały wczytane jako lista słowników i zapisane w zmiennej `cars`. Aby zobaczyć wartości każdego słownika na tej liście, można użyć następującej pętli:

```
for row in cars:
  print(list(row.values()))
```

W ciele pętli dla każdego słownika na liście wywołujemy metodę `values()`, konwertując w ten sposób wartości słownika na obiekt `dict_values`, który z kolei można łatwo skonwertować na listę. Każda lista reprezentuje wiersz z pliku *.csv*, jak pokazano poniżej:

```
['1997', 'Ford', 'E350', '3200.00']
['1999', 'Chevy', 'venture', '4800.00']
['1996', 'Jeep', 'grand cherokee', '4900.00']
```

Załóżmy, że musimy zaktualizować pole `Cena` w drugim wierszu (dla marki Chevy, modelu venture) i zapisać tę zmianę w pliku *cars.csv*. Modyfikację możemy wprowadzić w następujący sposób:

```
to_update = ['1999', 'Chevy', 'venture'] ❶
new_price = '4500.00' ❷
with open('ścieżka/do/pliku/cars.csv', 'w') as csvfile: ❸
  fieldnames = cars[0].keys() ❹
  writer = csv.DictWriter(csvfile, fieldnames=fieldnames) ❺
  writer.writeheader()
  for row in cars: ❻
    if set(to_update).issubset(set(row.values())):
      row['Price'] = new_price
    writer.writerow(row)
```

Przede wszystkim potrzebny jest sposób identyfikacji wiersza, który ma zostać zaktualizowany. W tym celu tworzymy listę o nazwie `to_update`, zawierającą tyle pól wiersza, ile jest koniecznych, aby jednoznacznie zidentyfikować wiersz do zmodyfikowania ❶. Następnie w zmiennej `new_price` ❷ określamy nową wartość pola, które ma zostać zmienione. Potem otwieramy plik do zapisu, przekazując do funkcji `open()` flagę `'w'` ❸. Użyty tutaj tryb spowoduje nadpisanie istniejącej zawartości pliku. Musimy zatem zdefiniować nazwy pól, które mają zostać zapisane w pliku ❹. Są to nazwy kluczy używanych w słowniku reprezentującym wiersz z danymi samochodu.

Korzystając z funkcji `csv.DictWriter()` ❺, tworzymy obiekt, który odwzorowuje słowniki z listy samochodów na wynikowe wiersze tekstu, które mają zostać przekazane do pliku *cars.csv*. W pętli operującej na liście `cars` ❻ sprawdzamy, czy każdy wiersz pasuje do określonego identyfikatora. Jeśli tak, aktualizujemy pole `Cena` obecnie przetwarzanego wiersza. I w końcu, nadal w pętli, zapisujemy każdy wiersz do pliku, używając w tym celu metody `writer.writerow()`.

Oto jak będzie wyglądała zawartość pliku *cars.csv* po wykonaniu tego skryptu:

```
Rok,Marka,Model,Cena
1997,Ford,E350,3200.00
1999,Chevy,venture,4500.00
1996,Jeep,grand cherokee,4900.00
```

Jak widać, są to niemal te same dane co w pierwotnym pliku, jednak wartość pola `Cena` w drugim wierszu została zmieniona.

Dostęp do plików zdalnych i API

Kilka niestandardowych bibliotek Pythona, w tym urllib3 i Requests, umożliwia pobieranie danych ze zdalnego pliku, do którego można się odwołać za pośrednictwem adresu URL. Bibliotek tych można również używać do wysyłania żądań do interfejsów API HTTP (tych, które używają HTTP jako protokołu transportowego), z których wiele zwraca

żądane dane w formacie JSON. Działanie bibliotek urllib3 i Requests bazuje na generowaniu niestandardowych żądań HTTP na podstawie informacji wprowadzonych przez użytkownika.

HTTP (ang. *HyperText Transfer Protocol*), protokół klient/serwer, który stanowi podstawę wymiany danych w sieci, ma strukturę serii żądań i odpowiedzi. Komunikaty HTTP wysyłane przez klienta są *żądaniami*, podczas gdy komunikaty zwracane przez serwer są *odpowiedziami*. Na przykład za każdym razem, gdy klikniesz odnośnik wyświetlony na stronie WWW, przeglądarka, działając jako klient, wysyła żądanie HTTP w celu pobrania odpowiedniej strony z określonego serwera WWW. To samo można zrobić za pomocą skryptu Pythona. Skrypt, działając jako klient, uzyskuje żądane dane w postaci dokumentu JSON lub XML.

Jak działają żądania HTTP

Istnieje kilka rodzajów żądań HTTP. Najpopularniejsze z nich to `GET`, `POST`, `PUT` i `DELETE`. Są one również znane jako *metody żądań HTTP*, *polecenia HTTP* lub po prostu *czasowniki HTTP*. Polecenie HTTP w każdym żądaniu HTTP definiuje akcję, która ma zostać wykonana dla określonego zasobu. Na przykład żądanie `GET` pobiera dane z zasobu, podczas gdy żądanie `POST` przesyła dane do zapisania.

Żądanie HTTP zawiera również *cel* żądania (ang. *target*), przy czym zwykle jest to adres URL, oraz *nagłówki* (ang. *headers*), które są polami przekazującymi dodatkowe informacje. Niektóre żądania zawierają również *treść* (ang. *body*), przechowującą rzeczywiste dane żądania, takie jak informacje podane przez użytkownika w formularzu. Żądania `POST` zazwyczaj zawierają treść, podczas gdy żądania `GET` jej nie mają.

Jako przykład przeanalizujmy następujące żądanie HTTP:

```
GET ❷/api/books?bibkeys=ISBN%3A1718500521&format=json HTTP/1.1 ❶
Host: openlibrary.org ❸
User-Agent: python-requests/2.24.0 ❹
Accept-Encoding: gzip, deflate ❺
Accept: */*
Connection: keep-alive ❻
```

To żądanie używa polecenia HTTP `GET` do pobrania danych z konkretnego serwera (podanego jako `Host` ❸) określonego przy użyciu URI ❷. Pozostałe wiersze zawierają inne nagłówki określające dodatkowe informacje. Nagłówek żądania `User-Agent` identyfikuje aplikację wykonującą żądanie i jej wersję ❹. Nagłówki `Accept` informują, jakie typy treści klient jest w stanie zrozumieć ❺. Nagłówek `Connection` o wartości `keep-alive` ❻ informuje serwer, że powinien nawiązać trwałe połączenie z klientem, co pozwoli na wykonanie kolejnych żądań.

W Pythonie nie trzeba w pełni rozumieć wewnętrznej struktury żądań HTTP, aby je wysyłać i otrzymywać odpowiedzi. Jak się przekonasz w kolejnych punktach rozdziału, biblioteki takie jak Requests i urllib3 pozwalają łatwo i wydajnie manipulować żądaniami HTTP, po prostu wywołując odpowiednią metodę i przekazując do niej wymagane parametry.

Dzięki użyciu biblioteki Requests poprzednie żądanie HTTP można wygenerować w skrypcie Pythona w bardzo prosty sposób:

```
import requests
PARAMS = {'bibkeys':'ISBN:1718500521', 'format':'json'}
requests.get('http://openlibrary.org/api/books', params = PARAMS)
```

Biblioteka Requests została opisana bardziej szczegółowo w dalszej części rozdziału. Na razie zauważmy, że biblioteka ta pozwala uniknąć konieczności ręcznego ustawiania nagłówków żądań. Ustawia ona domyślne wartości w niezauważalny sposób, automatycznie generując w pełni sformatowane żądanie HTTP na podstawie zaledwie kilku wierszy kodu.

Biblioteka urllib3

urllib3 to biblioteka do obsługi adresów URL, która umożliwia docieranie do zasobów dostępnych pod adresami URL, takimi jak interfejsy API HTTP, strony internetowe i pliki, a także manipulowanie nimi. Biblioteka ta została zaprojektowana w celu wydajnego manipulowania żądaniami HTTP i wykorzystuje pule połączeń bezpieczne pod względem działania wielowątkowego, pozwalające zminimalizować zasoby używane po stronie serwera. W porównaniu z biblioteką Requests, która została opisana w następnej kolejności, urllib3 wymaga więcej ręcznej pracy, ale daje również bardziej bezpośrednią kontrolę nad przygotowywanymi żądaniami, co jest przydatne np. w sytuacji, gdy trzeba dostosować zachowanie puli lub jawnie dekodować odpowiedzi HTTP.

Instalowanie biblioteki urllib3

Ponieważ biblioteka urllib3 jest zależnością wielu popularnych pakietów Pythona, takich jak Requests i `pip`, prawdopodobnie już będzie zainstalowana w Twoim środowisku Pythona. Aby to sprawdzić, spróbuj ją zaimportować w interaktywnym interpreterze Pythona. Jeśli zostanie wyświetlony błąd `ModuleNotFoundError`, będziesz musiał zainstalować tę bibliotekę jawnie, używając w tym celu następującego polecenia:

```
$ pip install urllib3
```

Dostęp do plików przy użyciu biblioteki urllib3

Aby zobaczyć, jak korzystając z biblioteki urllib3, wczytać dane z pliku dostępnego pod adresem URL, możemy wykorzystać utworzony wcześniej plik *excerpt.txt*. By udostępnić ten plik za pośrednictwem adresu URL, można umieścić go w katalogu dokumentów serwera HTTP uruchomionego na lokalnym hoście.

Uruchom poniższy kod, zmieniając w nim w razie potrzeby adres URL:

```
import urllib3
http = urllib3.PoolManager() ❶
```

```
r = http.request('GET', 'http://localhost/excerpt.txt') ❷
for i, line in enumerate(r.data.decode('utf-8').split('\n')): ❸
  if line.strip():
    print("Wiersz %i: " %(i), line.strip()) ❹
```

Najpierw tworzymy instancję `PoolManager` ❶, która jest sposobem, w jaki urllib3 wykonuje żądania. Następnie należy wysłać żądanie HTTP do określonego adresu URL za pomocą metody `request()` menedżera `PoolManager` ❷. Metoda `request()` zwraca obiekt `HTTPResponse`. Dostęp do żądanych danych uzyskuje się za pośrednictwem atrybutu `data` tego obiektu ❸. Następnie wyświetlamy niepuste wiersze pobranego pliku, przy czym na początku każdego z nich umieszczamy jego numer ❹.

Generowanie żądań do API przy użyciu urllib3

Biblioteki urllib3 można także użyć do wysyłania żądań do interfejsów API HTTP. W poniższym przykładzie generujemy żądanie skierowane do News API (*https://newsapi.org*), które wyszukuje artykuły z szerokiego zestawu źródeł wiadomości, znajdując te, które są najbardziej odpowiednie dla przesłanego żądania. Podobnie jak wiele innych interfejsów API, także ten wymaga przekazania w każdym żądaniu klucza API. Klucz API dla programistów można uzyskać bezpłatnie na stronie *https://newsapi.org/register* po wypełnieniu prostego formularza rejestracyjnego. Po uzyskaniu klucza API możemy użyć poniższego kodu, aby wyszukać artykuły na temat języka programowania Python:

```
import json
import urllib3
http = urllib3.PoolManager()
r = http.request('GET', 'https://newsapi.org/v2/everything? q=Python ❶
        programming language& ❷apiKey=tu_podaj_swój_klucz_api& ❸ pageSize=5')
articles = json.loads(r.data.decode('utf-8')) ❹
for article in articles['articles']:
  print(article['title'])
  print(article['publishedAt'])
  print(article['url'])
  print()
```

Wyszukiwaną frazę przekazuje się jako parametr `q` zapisany w adresie URL żądania ❶. Jedynym wymaganym parametrem, który należy podać w adresie URL żądania, jest `apiKey` ❷, służący do przekazania klucza API. Istnieje również wiele innych opcjonalnych parametrów, np. można określić źródła wiadomości lub blogi, z których mają pochodzić artykuły. W tym konkretnym przykładzie używamy parametru `pageSize`, aby określić liczbę pobieranych artykułów — w naszym przypadku będzie ich pięć ❸. Pełną listę obsługiwanych parametrów można znaleźć w dokumentacji News API dostępnej pod adresem *https://newsapi.org/docs*.

Atrybut `data` obiektu `HTTPResponse` zwracanego przez metodę `request()` jest dokumentem JSON zapisanym jako obiekt `bytes`. Ten obiekt dekodujemy na łańcuch znaków, który następnie przekazujemy do metody `json.loads()`, żeby skonwertować go na słownik ❹. Aby zobaczyć strukturę danych w tym słowniku, można go po prostu wyświetlić, choć

na powyższym przykładzie ten krok został pominięty. Jeśli spojrzymy na dane wyjściowe, przekonamy się, że w zwróconym dokumencie informacje o artykułach można znaleźć na liście o nazwie `articles`, a każdy rekord na tej liście ma pola `title`, `publishedAt` i `url`.

Korzystając z tych informacji, możemy wyświetlić pobraną listę artykułów w bardziej czytelnym formacie, np. takim jak ten przedstawiony poniżej:

```
How Mojo Hopes to Revamp Python for an AI World
2023-11-19T15:34:00Z
https://developers.slashdot.org/story/23/11/18/2128233/how-mojo-hopes-to-revamp-python-
↪for-an-ai-world

Amazon unleashes Q, an AI assistant for the workplace
2023-11-29T17:13:08Z
https://arstechnica.com/information-technology/2023/11/amazon-unleashes-q-an-ai-assistant-
↪for-the-workplace/

2023's Online 'Advent Calendars' Challenge Programmers With Tips and Puzzles
2023-12-18T12:34:00Z
https://developers.slashdot.org/story/23/12/18/0457249/2023s-online-advent-calendars-
↪challenge-programmers-with-tips-and-puzzles

Onyx, a new programming language powered by WebAssembly
2023-12-01T17:22:01Z
https://wasmer.io/posts/onyxlang-powered-by-wasmer

Programs are games, programming is a game
2023-12-14T15:30:49Z
https://blog.charliemeyer.co/programs-are-games-programming-is-a-game/
```

Ten przykład ilustruje sposób integracji News API z aplikacją Pythona za pomocą bezpośrednich żądań HTTP obsługiwanych przy użyciu biblioteki urllib3. Alternatywą byłoby zastosowanie niestandardowej biblioteki klienckiej Pythona opisanej na stronie *https://newsapi.org/docs/client-libraries/python*.

Biblioteka Requests

Requests to kolejna popularna biblioteka do obsługi adresów URL, która umożliwia łatwe wysyłanie żądań HTTP. Requests w niewidoczny sposób używa biblioteki urllib3 i jeszcze bardziej ułatwia wysyłanie żądań i pobieranie danych. Bibliotekę tę można zainstalować za pomocą polecenia `pip` w następujący sposób:

```
$ pip install requests
```

Czasowniki HTTP są implementowane jako metody biblioteki (np. `requests.get()` dla żądania HTTP `GET`). Poniższy przykład pokazuje, w jaki sposób przy użyciu biblioteki Requests można uzyskać zdalny dostęp do pliku *excerpt.txt*.

```
import requests
r = requests.get('http://localhost/excerpt.txt') ❶
for i, line in enumerate(❷ r.text.split('\n')):
  if line.strip():
    print("Wiersz %i: " %(i), line.strip()) ❸
```

Ten skrypt wykonuje żądanie HTTP `GET`, używając do tego celu metody `requests.get()`, do której przekazywany jest adres URL ❶. Metoda zwraca obiekt `Response`, który w atrybucie `text` ❷ zawiera pobraną zawartość. Żądania automatycznie dekodują pobierane treści, zgadując ich kodowanie, więc nie musimy robić tego ręcznie. Podobnie jak w przedstawionym wcześniej przykładzie korzystającym z biblioteki urllib3, także tu wyświetlane są tylko niepuste wiersze tekstu, przy czym na początku dodawane są ich numery ❸.

> **ĆWICZENIE 6. KORZYSTANIE Z API PRZY UŻYCIU BIBLIOTEKI REQUESTS**
>
> Podobnie jak urllib3, także biblioteka Requests może współdziałać z interfejsami API HTTP. Spróbuj zmodyfikować kod, który wysyła żądanie `GET` do interfejsu API News, tak aby korzystał z biblioteki Requests zamiast urllib3. Zwróć uwagę, że w przypadku biblioteki Requests nie musisz ręcznie dodawać parametrów zapytania do przekazywanego adresu URL. Zamiast tego możesz przekazać parametry jako słownik łańcuchów znaków.

Przenoszenie danych do i z obiektów DataFrame

Biblioteka pandas zawiera szereg metod do wczytywania danych, z których każda jest przeznaczona do pobierania danych zapisanych w określonym formacie bądź pochodzących ze źródła określonego typu. Metody te umożliwiają wczytywanie danych tabelarycznych do obiektów `DataFrame` przy użyciu jednego wywołania, dzięki czemu zaimportowany zestaw danych jest natychmiast gotowy do analizy. pandas ma również metody pozwalające na konwertowanie danych przechowywanych w obiektach `DataFrame` na inne formaty, takie jak JSON. W tym podrozdziale przedstawione zostaną przykłady metod służących do przenoszenia danych do lub z obiektów `DataFrame`. Przyjrzymy się również bibliotece pandas-datareader, która przydaje się do wczytywania do obiektów `DataFrame` danych pochodzących z różnych internetowych źródeł.

Importowanie zagnieżdżonych struktur JSON

Ponieważ JSON stał się *de facto* standardem wymiany danych między aplikacjami, ważne jest, aby dysponować możliwością szybkiego zaimportowania dokumentu JSON i przekonwertowania go na strukturę danych Pythona. W poprzednim rozdziale przedstawiony został

przykład wczytywania do obiektu `DataFrame` prostej, niezagnieżdżonej struktury JSON przy użyciu metody `read_json()` biblioteki pandas. W tym punkcie dowiesz się, jak wczytywać bardziej złożone dokumenty JSON, zawierające zagnieżdżone dane, takie jak te przedstawione poniżej:

```
data = [
  { "Prac":"Piotr Rabanowicz",
    "ObsZam":[ {"NumZam":2608,"Wartosc":35},
               {"NumZam":2617,"Wartosc":35},
               {"NumZam":2620,"Wartosc":139}
    ]},
    { "Prac":"Joanna Broniewicz",
    "ObsZam":[ {"NumZam":2621,"Wartosc":95},
               {"NumZam":2626,"Wartosc":218}
     ]
}]
```

Jak widać, każdy wpis w dokumencie JSON zaczyna się od pary klucz-wartość o prostej strukturze (tej z kluczem `Prac`), po której następuje zagnieżdżona struktura z kluczem `ObsZam`. Możesz przekonwertować tę hierarchiczną strukturę JSON na tabelaryczną ramkę danych za pomocą metody `json_normalize()` biblioteki pandas, która pobiera zagnieżdżoną strukturę i spłaszcza ją, czy też *normalizuje*, do prostej tabeli. Oto jak to zrobić:

```
import json
import pandas as pd
df = pd.json_normalize(❶ data, ❷ "ObsZam", ❸ "Prac").set_index([❹ "Prac","NumZam"])
print(df)
```

Oprócz danych JSON ❶, które mają zostać przetworzone przez metodę `json_normalize()`, w jej wywołaniu wskazujemy również `ObsZam` jako zagnieżdżoną tablicę, która ma zostać spłaszczona ❷, i `Prac` jako pole, które ma zostać wykorzystane jako część złożonego indeksu w wynikowej tabeli ❸. W tym samym wierszu kodu podajemy dwie kolumny, które mają zostać użyte jako indeks: `Prac` i `ObsZam` ❹. W rezultacie zostanie wyświetlony następujący wynikowy obiekt `DataFrame`:

```
                        Wartosc
Prac               NumZam
Piotr Rabanowicz     2608      35
                     2617      35
                     2620     139
Joanna Broniewicz    2621      95
                     2626     218
```

Uwaga

Korzystanie z indeksu dwukolumnowego upraszcza agregowanie danych w grupach. Zagadnieniem obiektów `DataFrame` z indeksami wielokolumnowymi zajmiemy się szczegółowo w rozdziale 6.

Konwersja obiektów DataFrame na format JSON

W praktyce często trzeba wykonać operację odwrotną — zapisać zawartość obiektu `DataFrame` w dokumencie JSON. Poniższy kod konwertuje używany wcześniej obiekt `DataFrame` z powrotem na postać danych JSON, z których został pierwotnie wygenerowany:

```
df = df.reset_index() ❶
json_doc = (df.groupby(['Prac'], as_index=True) ❷
             .apply(lambda x: x[['NumZam','Wartosc']].to_dict('records')) ❸
             .reset_index() ❹
             .rename(columns={0:'ObsZam'}) ❺
             .to_json(orient='records')) ❻
```

Zaczynamy od usunięcia z obiektu `DataFrame` dwukolumnowego indeksu, tak by `Prac` i `NumZam` stały się zwykłymi kolumnami ❶. Następnie używając złożonego wywołania, konwertujemy obiekt `DataFrame` na dokument JSON. Konwersja ta rozpoczyna się od operacji `groupby()`, która grupuje wiersze według kolumny `Prac` ❷. Używamy w tym celu wywołania `groupby()` połączonego z wywołaniem `apply()`, by zastosować funkcję lambda do każdego rekordu w każdej grupie ❸. W wyrażeniu lambda określamy listę pól, które mają się znaleźć w wierszu zagnieżdżonej tablicy powiązanej z każdym rekordem `Prac`. Używamy metody `DataFrame.to_dict()` z parametrem `records`, aby sformatować pola w tablicy w następujący sposób: [{*kolumna*: *wartość*}, ... , {*kolumna*: *wartość*}], gdzie każdy słownik reprezentuje zamówienie powiązane z danym pracownikiem.

W tym momencie dysponujemy obiektem `Series` z indeksem `Prac` i kolumną zawierającą tablicę zamówień powiązanych z pracownikiem. Aby nadać tej kolumnie nazwę (w tym przypadku `ObsZam`), należy przekonwertować obiekt `Series` na `DataFrame`. Jednym z prostych sposobów, by to zrobić, jest użycie `reset_index()` ❹. Oprócz konwersji serii na obiekt `DataFrame`, `reset_index()` zmienia `Prac` z indeksu na zwykłą kolumnę, co będzie ważne podczas konwersji obiektu `DataFrame` na format JSON. Na koniec jawnie określamy nazwę kolumny zawierającej zagnieżdżoną tablicę (`ObsZam`) za pomocą metody `rename()` obiektu `DataFrame` ❺ i przekształcamy zmienioną ramkę danych do formatu JSON ❻.

Zawartość zmiennej `json_doc` będzie wyglądała jak na poniższym przykładzie:

```
[{"Prac":"Joanna Broniewicz",
  "ObsZam":[{"NumZam":2621,"Wartosc":95},
    {"NumZam":2626,"Wartosc":218}
  ]},
 {"Prac":"Piotr Rabanowicz",
  "ObsZam":[{"NumZam":2608,"Wartosc":35},
    {"NumZam":2617,"Wartosc":35},
    {"NumZam":2620,"Wartosc":139}]
}]
```

Aby poprawić czytelność tych danych, można je wyświetlić za pomocą następującego wywołania:

```
print(json.dumps(json.loads(json_doc), indent=2))
```

ĆWICZENIE 7. MANIPULOWANIE ZŁOŻONYMI STRUKTURAMI DANYCH W FORMACIE JSON

Próbka danych JSON użyta w poprzednim punkcie rozdziału miała pojedyncze pole o prostej strukturze (`Prac`) na najwyższym poziomie każdego rekordu. W rzeczywistym dokumencie JSON takich pól może być więcej. Rekordy w tej próbce mają drugie proste pole, `Prac_email`, na najwyższym poziomie:

```
data = [{ "Prac":"Piotr Rabanowicz",
  "Prac_email": "piotr.rabanowicz",
  "ObsZam":[{"NumZam":2608,"Wartosc":35},
            {"NumZam":2617,"Wartosc":35},
            {"NumZam":2620,"Wartosc":139}
   ]},
   {"Prac":"Joanna Broniewicz",
   "Prac_email": "joanna.broniewicz",
   "ObsZam":[ {"NumZam":2621,"Wartosc":95},
            {"NumZam":2626,"Wartosc":218}
    ]
}]
```

Aby wczytać te dane do obiektu `DataFrame`, należy przekazać listę wszystkich pól o prostej strukturze znajdujących się na najwyższym poziomie rekordów jako trzeci parametr wywołania metody `json_normalize()`, jak pokazano tutaj:

```
df = pd.json_normalize(data, "ObsZam", ["Prac","Prac_email"]).set_index(["Prac","Prac
_email","NumZam"])
```

W tym przypadku zawartość obiektu `DataFrame` będzie następująca:

```
                                          Wartosc
Prac              Prac_email              NumZam
Piotr Rabanowicz piotr.rabanowicz          2608       35
                                           2617       35
                                           2620      139
Joanna Broniewicz   joanna.broniewicz      2621       95
                                           2626      218
```

Spróbuj przekonwertować ten obiekt `DataFrame` z powrotem na początkowy dokument JSON, modyfikując operację `groupby` pokazaną w poprzedniej sekcji.

Wczytywanie danych z internetu przy użyciu pandas-datareader

Kilka niestandardowych bibliotek zawiera metody odczytu zgodne z biblioteką pandas i umożliwiające dostęp do informacji z różnych internetowych źródeł danych, takich jak Quandl (*https://data.nasdaq.com*) i Stooq (*https://stooq.com*). Najpopularniejszą z nich jest biblioteka pandas-datareader. W chwili pisania książki biblioteka ta udostępniała 70 metod, z których każda została zaprojektowana do wczytywania obiektów `DataFrame` danych z określonego źródła. Wiele metod biblioteki to opakowania dla finansowych API, pozwalające na łatwe uzyskanie danych finansowych w formacie pandas.

Instalowanie biblioteki pandas-datareader

Bibliotekę pandas-datareader można zainstalować przy użyciu następującego polecenia:

```
$ pip install pandas-datareader
```

Opisy metod służących do wczytywania danych można znaleźć w dokumentacji biblioteki pandas-datareader na stronie *https://pandas-datareader.readthedocs.io/en/latest/remote_data.html*. Można również wydrukować listę dostępnych metod za pomocą funkcji `dir()` Pythona:

```
import pandas_datareader.data as pdr
print(dir(pdr))
```

Pobieranie danych z serwisu Stooq

W poniższym przykładzie użyto metody `get_data_stooq()` w celu uzyskania danych indeksu S&P 500 za określony okres:

```
import pandas_datareader.data as pdr
spx_index = pdr.get_data_stooq('^SPX', '2022-01-03', '2022-01-10')
print(spx_index)
```

Metoda `get_data_stooq()` pobiera dane z serwisu Stooq, darmowej strony internetowej, która dostarcza informacji dotyczących wielu indeksów giełdowych. Jako pierwszy parametr należy podać symbol żądanego indeksu rynkowego. Informacje na temat dostępnych opcji można znaleźć na stronie *https://stooq.com/t*.

Uzyskane dane indeksu S&P 500 będą zazwyczaj wyświetlane w tym formacie:

```
              Open     High      Low    Close      Volume
Date
2022-01-10  4655.34  4673.02  4582.24  4670.29  2668776356
2022-01-07  4697.66  4707.95  4662.74  4677.03  2414328227
```

```
2022-01-06  4693.39  4725.01  4671.26  4696.05  2389339330
2022-01-05  4787.99  4797.70  4699.44  4700.58  2810603586
2022-01-04  4804.51  4818.62  4774.27  4793.54  2841121018
2022-01-03  4778.14  4796.64  4758.17  4796.56  2241373299
```

Kolumna `Date` jest domyślnie ustawiona jako indeks obiektu `DataFrame`.

Podsumowanie

W tym rozdziale dowiedziałeś się, jak pozyskiwać dane z różnych źródeł i wprowadzać je do skryptów Pythona w celu dalszego przetwarzania. W szczególności nauczyłeś się, jak importować dane z plików przy użyciu wbudowanych funkcji Pythona, jak wysyłać żądania HTTP ze skryptów Pythona do internetowych interfejsów API i jak korzystać z metod wczytujących biblioteki pandas w celu pozyskiwania danych zapisywanych w różnych formatach i pochodzących z różnych źródeł. Dowiedziałeś się również, jak eksportować dane do plików i jak konwertować dane z obiektów `DataFrame` na format JSON.


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

PYTHON: TWÓJ NAJLEPSZY SOJUSZNIK W PRZETWARZANIU DANYCH!

Python jest idealnym wyborem dla danologów, którzy chcą w prosty sposób uzyskiwać dostęp do dowolnego rodzaju danych, przetwarzać je i analizować. Służy do tego zarówno bogaty zestaw wbudowanych struktur danych, jak i solidny zbiór przeznaczonych do ich analizy bibliotek open source. Sam język pozwala na tworzenie zwięzłego kodu przy minimalnym nakładzie czasu i wysiłku: jeden wiersz kodu może filtrować, przekształcać i agregować dane.

Tę książkę docenią średnio zaawansowani użytkownicy Pythona, którzy tworzą aplikacje korzystające z osiągnięć nauki o danych. Znajdziesz w niej omówienie możliwości języka, wbudowanych struktur danych Pythona, jak również takich bibliotek jak NumPy, pandas, scikit-learn i matplotlib. Nauczysz się wczytywania danych w różnych formatach, porządkowania, grupowania i agregowana zbiorów danych, a także tworzenia wykresów i map. Poszczególne zagadnienia zostały zilustrowane praktycznymi przykładami tworzenia rzeczywistych aplikacji, takich jak system obsługi taksówek z wykorzystaniem danych lokalizacyjnych, analiza reguł asocjacyjnych dla danych transakcji czy też uczenie maszynowe modelu przewidującego zmiany kursów akcji. Każdy rozdział zawiera interesujące ćwiczenia, które pozwolą Ci nabrać biegłości w stosowaniu opisanych tu technik.

Dzięki tej książce nauczysz się:

- efektywnie korzystać ze struktur danych Pythona
- wyciągać cenne informacje z danych
- posługiwać się danymi: tekstowymi, przestrzennymi, szeregami czasowymi
- korzystać z wielu typów i formatów danych, w tym JSON i CSV
- używać technik uczenia maszynowego do celów przetwarzania języka naturalnego

Yuli Vasiliev jest programistą, autorem książek i konsultantem. Specjalizuje się w wykorzystaniu oprogramowania typu open source, tworzeniu struktur danych i modeli, a także implementacji systemów bazodanowych. Na co dzień posługuje się językiem Python w wersji 3.x.

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

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-289-1020-1
9 788328 910201

Cena: 69,00 zł

no starch press