BIT34503 Data Science

CHAPTER 4: DATA GATHERING

4.0 DATA GATHERING

4.1 Obtain data from online repositories
4.2 Import data from local file formats (JSON, XML)
4.3 Import data using Web API

4.1 Obtain data from online repositories

- What if you need some reference data for a comparison or adding context? There are a bunch of libraries for downloading public datasets straight into your environment — think of it as pulling from APIs without having to manage all the extra complexity.

a) Pandas_datareader

- Pandas_datareader is a great way to pull data from the internet into your Python environment. It is particularly suited to financial data, but also has some World Bank datasources. To get Zoom's daily share price over the past few years, try the following:

```
from pandas_datareader import data
import datetime as dt
zm = data.DataReader(
    "ZM",
    start='2019-1-1',
    end=dt.datetime.today(),
    data_source='yahoo'
).reset_index()
zm.head()
```

- Running that code gives us the following output:

Out[55]:

	Date	High	Low	Open	Close	Volume	Adj Close
0	2019-04-18	66.000000	60.320999	65.000000	62.000000	25764700	62.000000
1	2019-04-22	68.900002	59.939999	61.000000	65.699997	9949700	65.699997
2	2019-04-23	74.168999	65.550003	66.870003	69.000000	6786500	69.000000
3	2019-04-24	71.500000	63.160000	71.400002	63.200001	4973500	63.200001
4	2019-04-25	66.849998	62.599998	64.739998	65.000000	3863300	65.000000

b) DataCommons

- Datacommons is a project by Google providing access to standardized and cleaned public datasets. The underlying data is represented in a graph format, making it really easy to query and join data from many different datasources e.g. the US Census, World Bank, Wikipedia, Centre for Disease Control and more. Here's a basic example:

```
!pip install datacommons datacommons_pandas --upgrade --quiet
import datacommons_pandas as dc
import pandas as pd
city_dcids = dc.get_property_values(["CDC500_City"], "member",
limit=500)["CDC500_City"]
cdc500_df = dc.build_multivariate_dataframe(
    city_dcids,
    ["Percent_Person_Obesity",  # Prevalence of obesity from CDC
    "Median_Income_Person",
    "Median_Age_Person",
    "UnemploymentRate_Person",  # Unemployment rate from BLS
    "Count_Person_BelowPovertyLevelInThePast12Months",  # Persons living
below the poverty line from Census
    "Count_Person",  # Total population from Census
    ],
)
cdc500_df.info()
```

- Running that code gives us the following:

```
<class 'pandas.core.frame.DataFrame'>
Index: 499 entries, geoId/0107000 to geoId/5613900
Data columns (total 6 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   Percent_Person_Obesity                          499 non-null    float64
 1   Median_Income_Person                            499 non-null    int64
 2   Median_Age_Person                               499 non-null    float64
 3   UnemploymentRate_Person                         494 non-null    float64
 4   Count_Person_BelowPovertyLevelInThePast12Months 499 non-null    int64
 5   Count_Person                                    499 non-null    int64
dtypes: float64(3), int64(3)
memory usage: 27.3+ KB
```

c) PyTrends (Google Trends)

- PyTrends is an unofficial but useful library for querying Google Trends data — here's a simple example:

```
import pandas as pd
from pytrends.request import TrendReq
pytrends = TrendReq()
keywords = ["oat milk", "soy milk", "almond milk"]
pytrends.build_payload(keywords, cat=0, geo='', gprop='') #
Get data from the last 5 years
top_queries = pytrends.interest_over_time()[keywords]
top_queries.head()
```

- Running that code gives us the following output:

```
Out[61]:
```

date	oat milk	soy milk	almond milk
2016-10-09	2	20	49
2016-10-16	3	19	49
2016-10-23	3	17	47
2016-10-30	2	20	42
2016-11-06	2	19	42

d) Kaggle

- Kaggle is a data science community that hosts a lot of datasets and competitions for learning Python. You can download some of these datasets to play around with through their command-line interface (note: you'll need to sign up for a Kaggle account). For example, say we want to download some Zillow economics data, we can run the following commands in our terminal (Jupyter users: replace the $ with ! in your Python code:

```
$ pip install kaggle
$ export KAGGLE_USERNAME=datadinosaur
$ export KAGGLE_KEY=xxxxxxxxxxxxxx
$ kaggle datasets download zillow/zecon
$ unzip zecon.zip
```

This will download a zipped file of the datasets, and then uncompress them. From there, you can open them as local files with Pandas:

```
import pandas as pd
csv_file = "/Users/johnreid/Downloads/Zip_time_series.csv"
df_from_csv = pd.read_csv(csv_file)
df_from_csv.info()
```

4.2 Import data from local file formats (JSON, XML)

- Often the data you need is stored in a local file on your computer. Depending on where you're running your Python environment, you can either specify the filename as a relative or absolute path:

a) CSV files

b) Excel files

c) Text files

d) Multiple files / folders

4.3 Import data using Web API

- APIs
- Sometimes you'll need to access data from a particular platform your company uses, like Hubspot, Twitter or Trello. These platforms often have a public API that you can pull data from, directly inside your Python environment.

- The basic idea is you send a request (which may include query parameters and access credentials) to an endpoint. That endpoint will return a response code plus the data you asked for (hopefully). You'll need to look at the API documentation to understand what data fields are available. The data will usually be returned in JSON format, which allows for deeply-nested data.

a) Without credentials

- Let's do a minimal example using the OpenNotify API, which tracks all the people currently in space:

```
import requestsresponse = requests.get("http://api.open-
notify.org/astros.json")print(response.status_code)
print(response.json())res = pd.DataFrame(response.json()["people"])res.head()
```

- Running that code gives us the following output:

```
200
{'message': 'success', 'number': 10, 'people': [{'craft': 'ISS', 'name': 'Mark Vande Hei'}, {'craft': 'ISS', 'name':
'Oleg Novitskiy'}, {'craft': 'ISS', 'name': 'Pyotr Dubrov'}, {'craft': 'ISS', 'name': 'Thomas Pesquet'}, {'craft': 'I
SS', 'name': 'Megan McArthur'}, {'craft': 'ISS', 'name': 'Shane Kimbrough'}, {'craft': 'ISS', 'name': 'Akihiko Hoshid
e'}, {'craft': 'ISS', 'name': 'Anton Shkaplerov'}, {'craft': 'ISS', 'name': 'Klim Shipenko'}, {'craft': 'ISS', 'nam
e': 'Yulia Pereslid'}]}
```

Out[84]:

	craft	name
0	ISS	Mark Vande Hei
1	ISS	Oleg Novitskiy
2	ISS	Pyotr Dubrov
3	ISS	Thomas Pesquet
4	ISS	Megan McArthur

- The response code tells you the result of your API call — according to Dataquest the most common are:

- 200: Everything went okay, and the result has been returned (if any).
- 301: The server is redirecting you to a different endpoint. This can happen when a company switches domain names, or an endpoint name is changed.
- 400: The server thinks you made a bad request. This can happen when you don't send along the right data, among other things.
- 403: The resource you're trying to access is forbidden: you don't have the right permissions to see it.
- 404: The resource you tried to access wasn't found on the server.
- 503: The server is not ready to handle the request.

b) With credentials & query parameters

- Sometimes you may need more specific information from the API, or have to authenticate. There are several ways to do this, however one of the most common is adding URL parameters to your request.

- Let's assume we have a config.pyfile with our API key in it:

```
personal_api_key = "wouldntyouliketoknow"
```

- Then we create a dictionary for all the parameters (this is a made-up example) and pass it in:

```
import config
import pandas as pd
import requests
parameters = {
   "personal_api_key": config.personal_api_key,
   "date": "2021-09-22"
}
response = requests.get(url, params = parameters)

print(response.status_code)
print(response.json())

res = pd.DataFrame(response.json()["people"])
res.head()
```

If you don’t want to deal with JSON you can try searching for a Python library for that API — these are usually open-source and maintained by the company or third parties.

References

- https://towardsdatascience.com/13-ways-to-access-data-in-python-bac5683e0063

UTHM
Universiti Tun Hussein Onn Malaysia
1993-2023
AMANAH • PROFESIONAL • INOVATIF

Thank you