I

Data Wrangling

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Data Collection

Data collection is a crucial step in the process of obtaining valuable insights and making informed decisions. In today's interconnected world, data can be found in a multitude of sources, ranging from traditional files such as .csv, .html, .txt, .xlsx, .html, and .json, to databases powered by SQL, websites hosting relevant information, and APIs (Application Programming Interfaces) offered by companies. To efficiently gather data from these diverse sources, various tools can be employed. These tools encompass an array of technologies, including web scraping frameworks, database connectors, data extraction libraries, and specialized APIs, all designed to facilitate the collection and extraction of data from different sources. By leveraging these tools, organizations can harness the power of data and gain valuable insights to drive their decision-making processes.

Python offers a rich ecosystem of packages for data collection. Some commonly used Python packages for data collection include: including:

- Pandas: Pandas is a powerful library for data manipulation and analysis. It provides data structures and functions to efficiently work with structured data, making it suitable for data collection from CSV files, Excel spreadsheets, and SQL databases.
- BeautifulSoup: Beautiful Soup is a Python library for web scraping. It helps parse HTML and XML documents, making it useful for extracting data from websites.
- Requests: Requests is a versatile library for making HTTP requests. It simplifies the process of interacting with web services and APIs, allowing data retrieval from various sources.
- mysql-connector-python, psycopg2, and sqlite3: These libraries are Python connectors for MySQL, PostgreSQL, and sqlite databases, respectively. They enable data collection by establishing connections to these databases, executing queries, and retrieving data.

DOI: 10.1201/9781003462781-1

- Yahoo Finance: The Yahoo Finance library provides an interface to access financial data from Yahoo Finance. It allows you to fetch historical stock prices, company information, and other financial data.

These are just a few examples of Python packages commonly used for data collection. We will cover them in detail with tutorials and case studies. Depending on the specific data sources and requirements, there are many more packages available to facilitate data collection in Python.

1.1 COLLECT DATA FROM FILES

Storing data in different file formats allows for versatility and compatibility with various applications and tools.

- CSV (Comma-Separated Values): CSV files store tabular data in plain text format, where each line represents a row, and values are separated by commas (or other delimiters). CSV files are simple, human-readable, and widely supported. They can be easily opened and edited using spreadsheet software or text editors. However, CSV files may not support complex data structures, and there is no standardized format for metadata or data types. Pandas provides the read_csv() function, allowing you to read CSV files into a DataFrame object effortlessly. It automatically detects the delimiter, handles missing values, and provides convenient methods for data manipulation and analysis.
- TXT (Plain Text): TXT files contain unformatted text with no specific structure or metadata. TXT files are lightweight, widely supported, and can be easily opened with any text editor. However, TXT files lack a standardized structure or format, making it challenging to handle data that requires specific organization or metadata. Pandas offers the read_csv() function with customizable delimiters to read text files with structured data. By specifying the appropriate delimiter, you can read text files into a DataFrame for further analysis.
- XLSX (Microsoft Excel): XLSX is a file format used by Microsoft Excel to store spreadsheet data with multiple sheets, formatting, formulas, and metadata. XLSX files support complex spreadsheets with multiple tabs, cell formatting, and formulas. They are widely used in business and data analysis scenarios. However, XLSX files can be large, and manipulating them directly can be memory-intensive. Additionally, XLSX files require software like Microsoft Excel to view and edit. Pandas provides the read_excel() function, enabling the reading of XLSX files into DataFrames. It allows you to specify the sheet name, range of cells, and other parameters to extract data easily.
- JSON (JavaScript Object Notation): JSON is a lightweight, human-readable data interchange format that represents structured data as key-value pairs, lists, and nested objects. JSON is easy to read and write, supports complex nested structures, and is widely used for data interchange between systems. However, JSON files can be larger than their equivalent CSV representations, and handling

complex nested structures may require additional processing. Pandas provides the read_json() function to read JSON data directly into a DataFrame. It handles both simple and nested JSON structures, allowing for convenient data exploration and analysis.

- XML (eXtensible Markup Language): XML files store structured data using tags that define elements and their relationships. XML is designed to be self-descriptive and human-readable. XML files provide a flexible and extensible format for storing structured data. They are widely used for data interchange and can represent complex hierarchical structures. However, XML files can be verbose and have larger file sizes compared to other formats. Parsing XML files can be more complex due to the nested structure and the need for specialized parsing libraries. Pandas provides the read_xml() function to directly read XML files into a DataFrame. It provides several options for handling different XML structures, such as extracting data from specific tags, handling attributes, and parsing nested elements.
- HTML (Hypertext Markup Language): HTML files are primarily used for structuring and presenting content on the web. They consist of tags that define the structure and formatting of the data. HTML files provide a rich structure for representing web content and can include images, links, and other multimedia elements. However, HTML files are designed for web display, so extracting structured data from them can be more complex due to the presence of non-tabular content and formatting tags. Pandas provides the read_html() function, which can extract tabular data from HTML tables into a DataFrame.

1.1.1 Tutorial – Collect Data from Files

We may have stored data in multiple types of files, such as text, csv, excel, xml, html, etc. We can load them into dataframes.

```
import pandas as pd
```

1.1.1.1 CSV

We have done this when we learned pandas. You can get the path of your csv file, and feed the path to the function `read_csv`.

Default setting A lot cases, default setting will do the job.

```
df = pd.read_csv('/content/ds_salaries.csv')
```

```
df.head()
```

```
   Unnamed: 0  work_year experience_level employment_type  \
0           0       2020               MI              FT
1           1       2020               SE              FT
```

```
2           2       2020               SE              FT
3           3       2020               MI              FT
4           4       2020               SE              FT

                   job_title  salary salary_currency  salary_in_usd  \
0             Data Scientist   70000             EUR          79833
1  Machine Learning Scientist  260000             USD         260000
2          Big Data Engineer   85000             GBP         109024
3       Product Data Analyst   20000             USD          20000
4   Machine Learning Engineer  150000             USD         150000

  employee_residence  remote_ratio company_location company_size
0                 DE             0                DE            L
1                 JP             0                JP            S
2                 GB            50                GB            M
3                 HN             0                HN            S
4                 US            50                US            L
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 607 entries, 0 to 606
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Unnamed: 0          607 non-null    int64
 1   work_year           607 non-null    int64
 2   experience_level    607 non-null    object
 3   employment_type     607 non-null    object
 4   job_title           607 non-null    object
 5   salary              607 non-null    int64
 6   salary_currency     607 non-null    object
 7   salary_in_usd       607 non-null    int64
 8   employee_residence  607 non-null    object
 9   remote_ratio        607 non-null    int64
 10  company_location    607 non-null    object
 11  company_size        607 non-null    object
dtypes: int64(5), object(7)
memory usage: 57.0+ KB
```

Customize setting You can manipulate arguments for your specific csv file

```
df = pd.read_csv('/content/ds_salaries.csv', header = None)
df.head()
```

```
    0          1                2                3   \
0  NaN  work_year  experience_level  employment_type
1  0.0       2020               MI               FT
2  1.0       2020               SE               FT
3  2.0       2020               SE               FT
4  3.0       2020               MI               FT

                          4       5               6             7   \
```

```
0                   job_title  salary  salary_currency  salary_in_usd
1              Data Scientist   70000             EUR          79833
2  Machine Learning Scientist  260000             USD         260000
3           Big Data Engineer   85000             GBP         109024
4        Product Data Analyst   20000             USD          20000

                    8            9                 10            11
0  employee_residence  remote_ratio  company_location  company_size
1                  DE             0                DE             L
2                  JP             0                JP             S
3                  GB            50                GB             M
4                  HN             0                HN             S
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 608 entries, 0 to 607
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       607 non-null    float64
 1   1       608 non-null    object
 2   2       608 non-null    object
 3   3       608 non-null    object
 4   4       608 non-null    object
 5   5       608 non-null    object
 6   6       608 non-null    object
 7   7       608 non-null    object
 8   8       608 non-null    object
 9   9       608 non-null    object
 10  10      608 non-null    object
 11  11      608 non-null    object
dtypes: float64(1), object(11)
memory usage: 57.1+ KB
```

```
df = pd.read_csv('/content/ds_salaries.csv', header = None, skiprows=1)
df.head()
```

```
   0     1   2   3                           4       5    6       7   8   9   \
0   0  2020  MI  FT              Data Scientist   70000  EUR   79833  DE   0
1   1  2020  SE  FT  Machine Learning Scientist  260000  USD  260000  JP   0
2   2  2020  SE  FT           Big Data Engineer   85000  GBP  109024  GB  50
3   3  2020  MI  FT        Product Data Analyst   20000  USD   20000  HN   0
4   4  2020  SE  FT   Machine Learning Engineer  150000  USD  150000  US  50

   10 11
0  DE  L
1  JP  S
2  GB  M
3  HN  S
4  US  L
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 607 entries, 0 to 606
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       607 non-null    int64
 1   1       607 non-null    int64
 2   2       607 non-null    object
 3   3       607 non-null    object
 4   4       607 non-null    object
 5   5       607 non-null    int64
 6   6       607 non-null    object
 7   7       607 non-null    int64
 8   8       607 non-null    object
 9   9       607 non-null    int64
 10  10      607 non-null    object
 11  11      607 non-null    object
dtypes: int64(5), object(7)
memory usage: 57.0+ KB
```

```
df = pd.read_csv('/content/ds_salaries.csv', header = None,
      skiprows=1, skipfooter=300)
df.head()
```

```
   0     1   2   3                           4       5    6       7   8   9   \
0   0  2020  MI  FT              Data Scientist   70000  EUR   79833  DE   0
1   1  2020  SE  FT  Machine Learning Scientist  260000  USD  260000  JP   0
2   2  2020  SE  FT           Big Data Engineer   85000  GBP  109024  GB  50
3   3  2020  MI  FT        Product Data Analyst   20000  USD   20000  HN   0
4   4  2020  SE  FT   Machine Learning Engineer  150000  USD  150000  US  50

   10 11
0  DE  L
1  JP  S
2  GB  M
3  HN  S
4  US  L
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307 entries, 0 to 306
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       307 non-null    int64
 1   1       307 non-null    int64
 2   2       307 non-null    object
 3   3       307 non-null    object
 4   4       307 non-null    object
 5   5       307 non-null    int64
 6   6       307 non-null    object
```

```
 7   7       307 non-null    int64
 8   8       307 non-null    object
 9   9       307 non-null    int64
 10  10      307 non-null    object
 11  11      307 non-null    object
dtypes: int64(5), object(7)
memory usage: 28.9+ KB
```

1.1.1.2 TXT

If the txt follows csv format, then it can be read as a csv file

```
df = pd.read_csv('/content/ds_salaries.txt')
df
```

```
     Unnamed: 0  work_year experience_level employment_type  \
0             0       2020               MI              FT
1             1       2020               SE              FT
2             2       2020               SE              FT
3             3       2020               MI              FT
4             4       2020               SE              FT
..          ...        ...              ...             ...
602         602       2022               SE              FT
603         603       2022               SE              FT
604         604       2022               SE              FT
605         605       2022               SE              FT
606         606       2022               MI              FT

                    job_title  salary salary_currency  salary_in_usd  \
0                Data Scientist   70000             EUR          79833
1    Machine Learning Scientist  260000             USD         260000
2              Big Data Engineer   85000             GBP         109024
3          Product Data Analyst   20000             USD          20000
4     Machine Learning Engineer  150000             USD         150000
..                          ...     ...             ...            ...
602               Data Engineer  154000             USD         154000
603               Data Engineer  126000             USD         126000
604                Data Analyst  129000             USD         129000
605                Data Analyst  150000             USD         150000
606                AI Scientist  200000             USD         200000

    employee_residence  remote_ratio company_location company_size
0                   DE             0               DE            L
1                   JP             0               JP            S
2                   GB            50               GB            M
3                   HN             0               HN            S
4                   US            50               US            L
..                 ...           ...              ...          ...
602                 US           100               US            M
603                 US           100               US            M
604                 US             0               US            M
605                 US           100               US            M
606                 IN           100               US            L

[607 rows x 12 columns]
```

1.1.1.3 Excel

```
df = pd.read_excel('/content/ds_salaries.xlsx')
```

```
df.head()
```

```
   Unnamed: 0  work_year experience_level employment_type  \
0           0       2020               MI              FT
1           1       2020               SE              FT
2           2       2020               SE              FT
3           3       2020               MI              FT
4           4       2020               SE              FT

                    job_title  salary salary_currency  salary_in_usd  \
0              Data Scientist   70000             EUR          79833
1  Machine Learning Scientist  260000             USD         260000
2            Big Data Engineer   85000             GBP         109024
3         Product Data Analyst   20000             USD          20000
4   Machine Learning Engineer  150000             USD         150000

  employee_residence  remote_ratio company_location company_size
0                 DE             0               DE            L
1                 JP             0               JP            S
2                 GB            50               GB            M
3                 HN             0               HN            S
4                 US            50               US            L
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 607 entries, 0 to 606
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Unnamed: 0          607 non-null    int64
 1   work_year           607 non-null    int64
 2   experience_level    607 non-null    object
 3   employment_type     607 non-null    object
 4   job_title           607 non-null    object
 5   salary              607 non-null    int64
 6   salary_currency     607 non-null    object
 7   salary_in_usd       607 non-null    int64
 8   employee_residence  607 non-null    object
 9   remote_ratio        607 non-null    int64
 10  company_location    607 non-null    object
 11  company_size        607 non-null    object
dtypes: int64(5), object(7)
memory usage: 57.0+ KB
```

1.1.1.4 json

```
df = pd.read_json('/content/ds_salaries.json')
df.head()
```

```
   FIELD1  work_year experience_level employment_type  \
0       0       2020               MI              FT
1       1       2020               SE              FT
2       2       2020               SE              FT
3       3       2020               MI              FT
4       4       2020               SE              FT

                   job_title  salary salary_currency  salary_in_usd  \
0             Data Scientist   70000             EUR          79833
1  Machine Learning Scientist  260000             USD         260000
2           Big Data Engineer   85000             GBP         109024
3        Product Data Analyst   20000             USD          20000
4   Machine Learning Engineer  150000             USD         150000

  employee_residence  remote_ratio company_location company_size
0                 DE             0                DE            L
1                 JP             0                JP            S
2                 GB            50                GB            M
3                 HN             0                HN            S
4                 US            50                US            L
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 607 entries, 0 to 606
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   FIELD1              607 non-null    int64
 1   work_year           607 non-null    int64
 2   experience_level    607 non-null    object
 3   employment_type     607 non-null    object
 4   job_title           607 non-null    object
 5   salary              607 non-null    int64
 6   salary_currency     607 non-null    object
 7   salary_in_usd       607 non-null    int64
 8   employee_residence  607 non-null    object
 9   remote_ratio        607 non-null    int64
 10  company_location    607 non-null    object
 11  company_size        607 non-null    object
dtypes: int64(5), object(7)
memory usage: 57.0+ KB
```

1.1.1.5 XML

```
df = pd.read_xml('/content/ds_salaries.xml')
df.head()
```

```
   FIELD1  work_year experience_level employment_type  \
0       0       2020               MI              FT
1       1       2020               SE              FT
2       2       2020               SE              FT
3       3       2020               MI              FT
4       4       2020               SE              FT

                   job_title  salary salary_currency  salary_in_usd  \
0             Data Scientist   70000             EUR          79833
1  Machine Learning Scientist  260000             USD         260000
2           Big Data Engineer   85000             GBP         109024
3        Product Data Analyst   20000             USD          20000
4   Machine Learning Engineer  150000             USD         150000

  employee_residence  remote_ratio company_location company_size
0                 DE             0               DE            L
1                 JP             0               JP            S
2                 GB            50               GB            M
3                 HN             0               HN            S
4                 US            50               US            L
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 607 entries, 0 to 606
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   FIELD1              607 non-null    int64
 1   work_year           607 non-null    int64
 2   experience_level    607 non-null    object
 3   employment_type     607 non-null    object
 4   job_title           607 non-null    object
 5   salary              607 non-null    int64
 6   salary_currency     607 non-null    object
 7   salary_in_usd       607 non-null    int64
 8   employee_residence  607 non-null    object
 9   remote_ratio        607 non-null    int64
 10  company_location    607 non-null    object
 11  company_size        607 non-null    object
dtypes: int64(5), object(7)
memory usage: 57.0+ KB
```

1.1.1.6 HTM

```
df = pd.read_html('/content/ds_salaries.htm')[0]
df.head()
```

```
   FIELD1  work_year experience_level employment_type  \
0       0       2020               MI              FT
1       1       2020               SE              FT
2       2       2020               SE              FT
3       3       2020               MI              FT
4       4       2020               SE              FT

                    job_title  salary salary_currency  salary_in_usd  \
0              Data Scientist   70000             EUR          79833
1  Machine Learning Scientist  260000             USD         260000
2            Big Data Engineer   85000             GBP         109024
3         Product Data Analyst   20000             USD          20000
4   Machine Learning Engineer  150000             USD         150000

  employee_residence  remote_ratio company_location company_size
0                 DE             0               DE            L
1                 JP             0               JP            S
2                 GB            50               GB            M
3                 HN             0               HN            S
4                 US            50               US            L
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 607 entries, 0 to 606
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   FIELD1              607 non-null    int64
 1   work_year           607 non-null    int64
 2   experience_level    607 non-null    object
 3   employment_type     607 non-null    object
 4   job_title           607 non-null    object
 5   salary              607 non-null    int64
 6   salary_currency     607 non-null    object
 7   salary_in_usd       607 non-null    int64
 8   employee_residence  607 non-null    object
 9   remote_ratio        607 non-null    int64
 10  company_location    607 non-null    object
 11  company_size        607 non-null    object
dtypes: int64(5), object(7)
memory usage: 57.0+ KB
```

1.1.2 Documentation

It is always good to have a reference of the read files functions in pandas. You can find it via https://pandas.pydata.org/docs/reference/io.html

1.2 COLLECT DATA FROM THE WEB

Collecting data from the web is essential for various reasons:

- Access to vast amounts of information: The web contains an immense amount of data on diverse topics. By collecting data from the web, you can tap into this vast information pool and gain insights that can inform decision-making, research, analysis, and more.
- Real-time and up-to-date data: The web provides a platform for the dissemination of real-time and up-to-date information. By collecting data from the web, you can stay informed about the latest news, trends, market updates, social media activity, and other dynamic sources of information.
- Competitive intelligence: Collecting data from the web allows you to monitor your competitors, track their activities, analyze their strategies, and gain insights into the market landscape. This can help you make informed decisions and stay ahead in a competitive environment.
- Research and analysis: Web data collection is crucial for research, analysis, and data-driven insights. By collecting data from diverse sources, you can validate hypotheses, perform statistical analysis, conduct sentiment analysis, and uncover patterns or trends that can enhance understanding and drive informed decision-making.

The web has many websites, including structured websites, semi-structured websites, and unstructured websites, that differ in terms of their organization and consistency.

- Structured Websites: Structured websites have a well-defined and organized format, making it easy to locate specific information. They often follow a consistent layout and have clearly defined sections. Structured websites generally pose fewer challenges for data collection as the information is neatly organized. However, occasional variations in page layouts or changes in website structure can introduce some level of complexity. To collect data from structured websites, you can utilize libraries like Beautiful Soup or lxml in Python. These libraries enable you to parse the HTML structure of the web pages and extract desired data using specific tags or CSS selectors.
- Semi-Structured Websites: Semi-structured websites contain a mixture of structured and unstructured data. While certain sections might be organized, others may have varying formats or lack consistent organization. The main challenge with semi-structured websites is the inconsistency in data presentation. The lack of uniformity in structure and formatting requires additional effort to identify and extract the relevant data. Similar to structured websites, libraries like Beautiful Soup or lxml can help parse and extract data from semi-structured websites. However, you may need to employ additional techniques such as regular expressions or data cleaning procedures to handle variations in data presentation.

- Unstructured Websites: Unstructured websites lack a clear organization or predefined structure. They may have free-form text, multimedia content, and unorganized data scattered across multiple pages. Unstructured websites pose the most significant challenges for data collection due to the absence of consistent structure. The data may be embedded within paragraphs, images, or other non-tabular formats, requiring sophisticated techniques for extraction. For unstructured websites, natural language processing (NLP) techniques and machine learning algorithms can be employed to extract relevant information. These methods involve parsing the web content, identifying patterns, and applying text processing algorithms to extract structured data.

In summary, structured websites provide a clear structure, making data collection relatively straightforward. Semi-structured websites introduce some variability, requiring careful handling of inconsistencies. Unstructured websites present the most significant challenges, necessitating advanced techniques such as NLP and machine learning to extract structured information. Python libraries like Beautiful Soup, lxml, and NLP frameworks can assist in parsing and extracting data from these different types of websites, adapting to their specific characteristics and complexities.

1.2.1 Tutorial – Collect Data from Web

```
import pandas as pd
```

1.2.1.1 Wiki

Some websites maintains structured data, which is easy to read

```
table = pd.read_html('https://en.wikipedia.org/wiki/
                        List_of_countries_by_GDP_(nominal)#Table')
```

```
for i in table:
  print(type(i))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

```
for i in table:
  print(i.columns)
```

```
Int64Index([0], dtype='int64')
Int64Index([0, 1, 2], dtype='int64')
MultiIndex([( 'Country/Territory', 'Country/Territory'),
            (         'UN Region',         'UN Region'),
```

```
            (        'IMF[1][13]',          'Estimate'),
            (        'IMF[1][13]',              'Year'),
            (    'World Bank[14]',          'Estimate'),
            (    'World Bank[14]',              'Year'),
            ('United Nations[15]',          'Estimate'),
            ('United Nations[15]',              'Year')],
           )
...
Int64Index([0, 1], dtype='int64')
```

```
df = table[2]
df.head()
```

```
 Country/Territory UN Region IMF[1][13]           World Bank[14]             \
 Country/Territory UN Region   Estimate       Year      Estimate       Year
0             World         -  101560901       2022      96513077       2021
1     United States  Americas   25035164       2022      22996100       2021
2             China      Asia   18321197  [n 1]2022      17734063  [n 3]2021
3             Japan      Asia    4300621       2022       4937422       2021
4           Germany    Europe    4031149       2022       4223116       2021

  United Nations[15]
            Estimate       Year
0           85328323       2020
1           20893746       2020
2           14722801  [n 1]2020
3            5057759       2020
4            3846414       2020
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 217 entries, 0 to 216
Data columns (total 8 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   (Country/Territory, Country/Territory)  217 non-null    object
 1   (UN Region, UN Region)                  217 non-null    object
 2   (IMF[1][13], Estimate)                  217 non-null    object
 3   (IMF[1][13], Year)                      217 non-null    object
 4   (World Bank[14], Estimate)              217 non-null    object
 5   (World Bank[14], Year)                  217 non-null    object
 6   (United Nations[15], Estimate)          217 non-null    object
 7   (United Nations[15], Year)              217 non-null    object
dtypes: object(8)
memory usage: 13.7+ KB
```

1.2.1.2 Web Scraping

Some websites are semi-structured, which has metadata, such as labels, classes, etc, so we can look into their source code, and do web scraping.

> Note: You need to have a basic understanding of html, xml, in order to understand the source code and collect data from these websites.

Note: Some websites prevent users from scraping or scraping rapidly.

The first thing we'll need to do to scrape a web page is to download the page. We can download pages using the Python `requests` library.

The requests library will make a `GET` request to a web server, which will download the HTML contents of a given web page for us. There are several different types of requests we can make using requests, of which `GET` is just one. If you want to learn more, check out our API tutorial.

Let's try downloading a simple sample website, https://dataquestio.github.io/web-scraping-pages/simple.html.

Download by requests We'll need to first import the requests library, and then download the page using the requests.get method:

```
import requests

page = requests.get("https://dataquestio.github.io/
                        web-scraping-pages/simple.html")
page
```

```
<Response [200]>
```

After running our request, we get a Response object. This object has a status_code property, which indicates if the page was downloaded successfully:

```
page.status_code
```

```
200
```

A status_code of 200 means that the page downloaded successfully. We won't fully dive into status codes here, but a status code starting with a 2 generally indicates success, and a code starting with a 4 or a 5 indicates an error.

We can print out the HTML content of the page using the content property:

```
page.content
```

```
b'<!DOCTYPE html>\n<html>\n    <head>\n        <title>A simple example
    page</title>\n    </head>\n    <body>\n        <p>Here is some
    simple content for this page.</p>\n    </body>\n</html>'
```

Parsing by BeautifulSoup As you can see above, we now have downloaded an HTML document.

We can use the `BeautifulSoup` library to parse this document, and extract the text from the p tag.

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(page.content, 'html.parser')
```

We can now print out the HTML content of the page, formatted nicely, using the prettify method on the BeautifulSoup object.

```
print(soup.prettify())
```

```
<!DOCTYPE html>
<html>
 <head>
  <title>
   A simple example page
  </title>
 </head>
 <body>
  <p>
   Here is some simple content for this page.
  </p>
 </body>
</html>
```

This step isn't strictly necessary, and we won't always bother with it, but it can be helpful to look at prettified HTML to make the structure of the and where tags are nested easier to see.

Finding Tags Finding all instances of a tag at once What we did above was useful for figuring out how to navigate a page, but it took a lot of commands to do something fairly simple. If we want to extract a single tag, we can instead use the find_all method, which will find all the instances of a tag on a page.

if we are looking for the title, we can look for `<title>` tag

```
soup.find_all('title')
```

```
[<title>A simple example page</title>]
```

```
for t in soup.find_all('title'):
  print(t.get_text())
```

```
A simple example page
```

If we are looking for text, we can look for `<p>` tag

```
for t in soup.find_all('p'):
  print(t.get_text())
```

```
Here is some simple content for this page.
```

If you instead only want to find the first instance of a tag, you can use the find method, which will return a single BeautifulSoup object:

```
soup.find('p').get_text()
```

```
{"type":"string"}
```

Searching for tags by class and id:

Classes and ids are used by CSS to determine which HTML elements to apply certain styles to. But when we're scraping, we can also use them to specify the elements we want to scrape.

Let's try another page.

```
page = requests.get("https://dataquestio.github.io/
        web-scraping-pages/ids_and_classes.html")
soup = BeautifulSoup(page.content, 'html.parser')
soup
```

```
<html>
<head>
<title>A simple example page</title>
</head>
<body>
<div>
<p class="inner-text first-item" id="first">
                First paragraph.
            </p>
<p class="inner-text">
                Second paragraph.
            </p>
</div>
<p class="outer-text first-item" id="second">
<b>
                First outer paragraph.
            </b>
</p>
<p class="outer-text">
<b>
                Second outer paragraph.
            </b>
</p>
</body>
</html>
```

Now, we can use the find_all method to search for items by class or by id. In the below example, we'll search for any p tag that has the class outer-text:

```
soup.find_all('p', class_='outer-text')
```

```
[<p class="outer-text first-item" id="second">
 <b>
```

```
                First outer paragraph.
            </b>
 </p>, <p class="outer-text">
 <b>
                Second outer paragraph.
            </b>
 </p>]
```

In the below example, we'll look for any tag that has the class outer-text:

```
soup.find_all(class_="outer-text")
```

```
[<p class="outer-text first-item" id="second">
 <b>
                First outer paragraph.
            </b>
 </p>, <p class="outer-text">
 <b>
                Second outer paragraph.
            </b>
 </p>]
```

We can also search for elements by id:

```
soup.find_all(id="first")
```

```
[<p class="inner-text first-item" id="first">
                First paragraph.
            </p>]
```

1.2.2 Case Study – Collect Weather Data from Web

1.2.2.1 Downloading Weather Data

We now know enough to proceed with extracting information about the local weather from the National Weather Service website!

The local weather of Boulder, CO is: https://forecast.weather.gov/MapClick.php?lat=40.0466&lon=-105.2523#.YwpRBy2B1f0

Time to Start Scraping!

We now know enough to download the page and start parsing it. In the below code, we will:

- Download the web page containing the forecast.
- Create a BeautifulSoup class to parse the page.
- Find the div with id seven-day-forecast, and assign to seven_day
- Inside seven_day, find each individual forecast item. Extract and print the first forecast item.

```
import requests
from bs4 import BeautifulSoup

page = requests.get("https://forecast.weather.gov/
        MapClick.php?lat=40.0466&lon=-105.2523#.YwpRBy2B1f0")
soup = BeautifulSoup(page.content, 'html.parser')
seven_day = soup.find(id="seven-day-forecast")
forecast_items = seven_day.find_all(class_="tombstone-container")
print(forecast_items)
```

```
[<div class="tombstone-container">
<p class="period-name">Today<br/><br/></p>
<p><img alt="Today: Sunny...>
<p class="period-name">Tonight<br/><br/></p>
<p><img alt="Tonight: Mostly clear...>
...
```

```
tonight = forecast_items[0]
print(tonight.prettify())
```

```
<div class="tombstone-container">
 <p class="period-name">
  Today
  <br/>
  <br/>
 </p>
 <p>
  <img alt="Today: Sunny, with a high near 88.
    Northwest wind 9 to 13 mph,
    with gusts as high as 21 mph. "
    class="forecast-icon" src="newimages/medium/few.png"
    title="Today: Sunny, with a high near 88.
    Northwest wind 9 to 13 mph,
    with gusts as high as 21 mph. "/>
 </p>
 <p class="short-desc">
  Sunny
 </p>
 <p class="temp temp-high">
  High: 88 °F
 </p>
</div>
```

1.2.2.2 *Extracting Information of Tonight*

As we can see, inside the forecast item tonight is all the information we want. There are four pieces of information we can extract:

- The name of the forecast item – in this case, Tonight.
- The description of the conditions – this is stored in the title property of img.
- A short description of the conditions – in this case, Sunny and hot.
- The temperature hight – in this case, 98 degrees.

We'll extract the name of the forecast item, the short description, and the temperature first, since they're all similar:

```
period = tonight.find(class_="period-name").get_text()
short_desc = tonight.find(class_="short-desc").get_text()
temp = tonight.find(class_="temp").get_text()
print(period)
print(short_desc)
print(temp)
```

```
Today
Sunny
High: 88 °F
```

Now, we can extract the title attribute from the img tag. To do this, we just treat the BeautifulSoup object like a dictionary, and pass in the attribute we want as a key:

```
img = tonight.find("img")
desc = img['title']
print(desc)
```

```
Today: Sunny,
    with a high near 88.
    Northwest wind 9 to 13 mph,
    with gusts as high as 21 mph.
```

1.2.2.3 Extract all Nights!

Now that we know how to extract each individual piece of information, we can combine our knowledge with CSS selectors and list comprehensions to extract everything at once.

In the below code, we will:

Select all items with the class period-name inside an item with the class tombstone-container in seven_day. Use a list comprehension to call the get_text method on each BeautifulSoup object.

```
period_tags = seven_day.select(".tombstone-container .period-name")
periods = [pt.get_text() for pt in period_tags]
periods
```

```
['Today',
 'Tonight',
 'Sunday',
 'SundayNight',
 'Monday',
 'MondayNight',
 'Tuesday',
 'TuesdayNight',
 'Wednesday']
```

As we can see above, our technique gets us each of the period names, in order.

We can apply the same technique to get the other three fields:

```
short_descs = [sd.get_text() for sd in seven_day.select(
    ".tombstone-container .short-desc")]
temps = [t.get_text() for t in seven_day.select(
    ".tombstone-container .temp")]
descs = [d["title"] for d in seven_day.select(
    ".tombstone-container img")]

print(short_descs)
print(temps)
print(descs)
```

```
['Sunny', 'Mostly Clear', 'Sunny thenSlight ChanceT-storms',...]
['High: 88 °F', 'Low: 59 °F', 'High: 88 °F', 'Low: 57 °F', ...]
['Today: Sunny, with a high near 88. Northwest wind 9 to 13 mph...]
```

1.2.2.4 Deal with Data

We can now combine the data into a Pandas DataFrame and analyze it. A DataFrame is an object that can store tabular data, making data analysis easy.

In order to do this, we'll call the DataFrame class, and pass in each list of items that we have. We pass them in as part of a dictionary.

Each dictionary key will become a column in the DataFrame, and each list will become the values in the column:

```
import pandas as pd
weather = pd.DataFrame({
    "period": periods,
    "short_desc": short_descs,
    "temp": temps,
    "desc":descs
})
weather
```

Now let's save it to CSV.

```
weather.to_csv('data/Boulder_Weather_7_Days.csv')
```

1.3 COLLECT DATA FROM SQL DATABASES

Storing data in SQL databases offers several advantages and considerations. The advantages are:

- Advantages of Storing Data in SQL Databases: Structured Storage: SQL databases provide a structured storage model with tables, rows, and columns, allowing for efficient organization and retrieval of data.

- Data Integrity and Consistency: SQL databases enforce data integrity through constraints, such as primary keys, unique keys, and referential integrity, ensuring the accuracy and consistency of the stored data.
- Querying and Analysis: SQL databases offer powerful query languages (e.g., SQL) that enable complex data retrieval, filtering, aggregations, and analysis operations.
- ACID Compliance: SQL databases adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring reliable and transactional data operations.

To collect data from a SQL database, you need to establish a connection to the database server. This typically involves providing connection details such as server address, port, username, and password. Once connected, you can use SQL queries to extract data from the database. Queries can range from simple retrieval of specific records to complex joins, aggregations, and filtering operations. Python provides several libraries for interacting with SQL databases, such as sqlite3, psycopg2, pymysql, and pyodbc. These libraries allow you to establish connections, execute SQL queries, and retrieve the query results into Python data structures for further processing.

1.3.1 Tutorial – Collect Data from SQLite

1.3.1.1 What is SQLite

A file with the .sqlite extension is a lightweight SQL database file created with the SQLite software. It is a database in a file itself and implements a self-contained, full-featured, highly-reliable SQL database engine.

We use SQLite to demonstrate the approach to access SQL databases. They follow similar steps. You just need to setup your account credentials in the `connect` so you can connect the server.

1.3.1.2 Read an SQLite Database in Python

We use a Python package, sqlite3, to deal with SQLite databases. Once the sqlite3 package is imported, the general steps are:

1. Create a connection object that connects the SQLite database.
2. Create a cursor object
3. Create a query statement
4. Execute the query statement
5. Fetch the query result to result
6. If all work is done, close the connection.

We use the built-in SQLite database Chinook as the example here. We connect with the database, and show all the tables it contains.

```
import sqlite3

connection = sqlite3.connect('/content/ds_salaries.sqlite')
cursor = connection.cursor()

query = '''
SELECT name FROM sqlite_master
WHERE type='table';
'''

cursor.execute(query)
results = cursor.fetchall()
results
```

```
[('ds_salaries',)]
```

1.3.1.3 Play with the SQLite Databases

Using SQL statements, you can play with the SQLite Databases and get the data you need.

```
query = '''SELECT *
FROM ds_salaries'''

cursor.execute(query)
results = cursor.fetchall()
results
```

```
[(None,
  'work_year',
  'experience_level',
  'employment_type',
  'job_title',
  'salary',
  'salary_currency',
  'salary_in_usd',
  'employee_residence',
  'remote_ratio',
  'company_location',
  'company_size'),
 (0,
  '2020',
  'MI',
  'FT',
  'Data Scientist',
  '70000',
  'EUR',
  '79833',
  'DE',
  '0',
  'DE',
  'L'),
```

```
 ...,
 (606,
  '2022',
  'MI',
  'FT',
  'AI Scientist',
  '200000',
  'USD',
  '200000',
  'IN',
  '100',
  'US',
  'L')]
```

1.3.1.4 Save Data to CSV Files

Since CSV file is much more convenient to process, we still use pandas to convert and to write to CSV files.

```
import pandas as pd

df = pd.DataFrame(results)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 608 entries, 0 to 607
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       607 non-null    float64
 1   1       608 non-null    object
 2   2       608 non-null    object
 3   3       608 non-null    object
 4   4       608 non-null    object
 5   5       608 non-null    object
 6   6       608 non-null    object
 7   7       608 non-null    object
 8   8       608 non-null    object
 9   9       608 non-null    object
 10  10      608 non-null    object
 11  11      608 non-null    object
dtypes: float64(1), object(11)
memory usage: 57.1+ KB
```

```
df.iloc[0]
```

```
0                    NaN
1              work_year
2       experience_level
3        employment_type
4              job_title
5                 salary
6        salary_currency
```

```
7          salary_in_usd
8     employee_residence
9           remote_ratio
10      company_location
11          company_size
Name: 0, dtype: object
```

```
cols = list(df.iloc[0])
cols
```

```
[nan,
 'work_year',
 'experience_level',
 'employment_type',
 'job_title',
 'salary',
 'salary_currency',
 'salary_in_usd',
 'employee_residence',
 'remote_ratio',
 'company_location',
 'company_size']
```

```
df.columns = cols
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 608 entries, 0 to 607
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   nan                 607 non-null    float64
 1   work_year           608 non-null    object
 2   experience_level    608 non-null    object
 3   employment_type     608 non-null    object
 4   job_title           608 non-null    object
 5   salary              608 non-null    object
 6   salary_currency     608 non-null    object
 7   salary_in_usd       608 non-null    object
 8   employee_residence  608 non-null    object
 9   remote_ratio        608 non-null    object
 10  company_location    608 non-null    object
 11  company_size        608 non-null    object
dtypes: float64(1), object(11)
memory usage: 57.1+ KB
```

```
df.drop(0, inplace = True)
df
```

```
       NaN work_year experience_level employment_type  \
1      0.0      2020               MI              FT
2      1.0      2020               SE              FT
3      2.0      2020               SE              FT
```

```
4      3.0      2020               MI              FT
5      4.0      2020               SE              FT
..     ...       ...              ...             ...
603  602.0      2022               SE              FT
604  603.0      2022               SE              FT
605  604.0      2022               SE              FT
606  605.0      2022               SE              FT
607  606.0      2022               MI              FT

                     job_title  salary salary_currency salary_in_usd  \
1               Data Scientist   70000             EUR         79833
2    Machine Learning Scientist  260000             USD        260000
3             Big Data Engineer   85000             GBP        109024
4          Product Data Analyst   20000             USD         20000
5     Machine Learning Engineer  150000             USD        150000
..                          ...     ...             ...           ...
603               Data Engineer  154000             USD        154000
604               Data Engineer  126000             USD        126000
605                Data Analyst  129000             USD        129000
606                Data Analyst  150000             USD        150000
607                AI Scientist  200000             USD        200000

    employee_residence remote_ratio company_location company_size
1                   DE            0                DE            L
2                   JP            0                JP            S
3                   GB           50                GB            M
4                   HN            0                HN            S
5                   US           50                US            L
..                 ...          ...               ...          ...
603                 US          100                US            M
604                 US          100                US            M
605                 US            0                US            M
606                 US          100                US            M
607                 IN          100                US            L

[607 rows x 12 columns]
```

```
cursor.close()
connection.close()
```

1.3.2 Case Study – Collect Shopping Data from SQLite

Now you have learned how to collect data from a SQLite databse. Let's practice!

The attached `shopping.sqlite` file contains a dummy shopping dataset. Try to use your knowledge of collecting data from a SQL database, and retrieve information from it.

1.3.2.1 Establish the connection

```
import sqlite3

connection = sqlite3.connect('/content/shopping.sqlite')
cursor = connection.cursor()

query = '''
SELECT name FROM sqlite_master
WHERE type='table';
'''

cursor.execute(query)
results = cursor.fetchall()
results
```

```
[('customer_shopping_data',)]
```

1.3.2.2 Retrieve Information from the Database

```
query = '''SELECT *
FROM customer_shopping_data
Limit 3'''

cursor.execute(query)
results = cursor.fetchall()
results
```

```
[('I138884',
  'C241288',
  'Female',
  28,
  'Clothing',
  5,
  1500.4,
  'Credit Card',
  '5/8/2022',
  'Kanyon'),
 ('I317333',
  'C111565',
  'Male',
  21,
  'Shoes',
  3,
  1800.51,
  'Debit Card',
  '12/12/2021',
  'Forum Istanbul'),
 ('I127801',
  'C266599',
  'Male',
  20,
```

```
 'Clothing',
 1,
 300.08,
 'Cash',
 '9/11/2021',
 'Metrocity')]
```

1.3.2.3 Fetch all Records

```
query = '''SELECT *
FROM customer_shopping_data
'''

cursor.execute(query)
results = cursor.fetchall()
```

1.3.2.4 Columns' Names

We learned that the missing columns' names are: `['invoice_no', 'customer_id', 'gender', 'age', 'category', 'quantity', 'price', 'payment_method', 'invoice_date', 'shopping_mall']`.

Combine this information and create a DataFrame of the shopping data, then save it to a CSV file for later use.

```
cols = ['invoice_no',
        'customer_id',
        'gender',
        'age',
        'category',
        'quantity',
        'price',
        'payment_method',
        'invoice_date',
        'shopping_mall']
```

```
import pandas as pd
```

```
df = pd.DataFrame(results, columns= cols)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16029 entries, 0 to 16028
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   invoice_no      16029 non-null  object
 1   customer_id     16029 non-null  object
 2   gender          16029 non-null  object
 3   age             16029 non-null  int64
 4   category        16029 non-null  object
 5   quantity        16029 non-null  int64
```

```
 6   price           16029 non-null  float64
 7   payment_method  16029 non-null  object
 8   invoice_date    16029 non-null  object
 9   shopping_mall   16029 non-null  object
dtypes: float64(1), int64(2), object(7)
memory usage: 1.2+ MB
```

```
df.head()
```

```
  invoice_no customer_id  gender  age  category  quantity    price  \
0    I138884     C241288  Female   28  Clothing         5  1500.40
1    I317333     C111565    Male   21     Shoes         3  1800.51
2    I127801     C266599    Male   20  Clothing         1   300.08
3    I173702     C988172  Female   66     Shoes         5  3000.85
4    I337046     C189076  Female   53     Books         4    60.60

  payment_method invoice_date   shopping_mall
0    Credit Card     5/8/2022          Kanyon
1     Debit Card   12/12/2021  Forum Istanbul
2           Cash    9/11/2021       Metrocity
3    Credit Card   16/05/2021    Metropol AVM
4           Cash   24/10/2021          Kanyon
```

1.3.2.5 Save your retrieve information as a CSV file

```
df.to_csv('/content/shopping.csv')
```

1.4 COLLECT DATA THROUGH APIS

Collecting data through APIs (Application Programming Interfaces) offers several advantages:

- Structured Data: APIs provide structured and standardized data formats, making it easier to consume and integrate into applications or analysis pipelines.
- Real-time and Updated Data: APIs often provide real-time or near-real-time data, allowing you to access the latest information dynamically.
- Controlled Access: APIs allow data providers to control access to their data by implementing authentication mechanisms, usage limits, and access permissions.
- Targeted Data: APIs enable you to request specific data elements or subsets of data, minimizing unnecessary data transfer and processing.
- Automation and Integration: APIs facilitate automated data collection and integration into your workflows or systems.

There are some limitations and considerations too:

- Rate Limits and Usage Restrictions: Some APIs impose rate limits, usage quotas, or require subscription plans for accessing data beyond certain thresholds.

- Data Quality and Reliability: API data quality and reliability depend on the data provider. It's important to verify the accuracy, completeness, and consistency of the data obtained through APIs.
- API Changes and Deprecation: APIs may evolve over time, and changes to endpoints, parameters, or authentication mechanisms can require updates in your data collection code.

Examples of APIs are:

- Yahoo Finance API: The Yahoo Finance API provides access to financial market data, including stock quotes, historical prices, company information, and more. By interacting with the Yahoo Finance API, you can programmatically retrieve financial data for analysis, investment strategies, or market monitoring.
- OpenWeatherMap API: The OpenWeatherMap API offers weather data for various locations worldwide. You can fetch weather conditions, forecasts, historical weather data, and other meteorological information through their API.
- Twitter API: The Twitter API enables access to Twitter's vast collection of tweets and user data. You can use the API to retrieve tweets, monitor hashtags or keywords, analyze sentiment, and gain insights from Twitter's social media data.
- Google Maps API: The Google Maps API provides access to location-based services, including geocoding, distance calculations, routing, and map visualization. It allows you to integrate maps and location data into your applications or retrieve information related to places, addresses, or geographic features.

To collect data through APIs, you need to understand the API's documentation, authentication mechanisms, request formats (often in JSON or XML), and available endpoints. Python provides libraries such as requests and urllib that facilitate making HTTP requests to interact with APIs. You typically send HTTP requests with the required parameters, handle the API responses, and process the returned data according to your needs.

1.4.1 Tutorial – Collect Data from Yahoo

This tutorial will demonstrate how to use Python to retrieve financial data from Yahoo Finance. Using this, we may access historical market data as well as financial information about the company (for example, financial ratios).

1.4.1.1 Installation

```
!pip install yfinance
!pip install yahoofinancials
```

1.4.1.2 Analysis

The yfinance package can be imported into Python programs once it has been installed. We must use the company's ticker as an example in our argument.

A security is given a specific set of letters called a ticker or a stock symbol for trading purposes. For instance:

For Amazon, it is "AMZN" For Facebook, it is "FB" For Google, it is "GOOGL" For Microsoft, it is "MSFT"

```
import yfinance as yahooFinance

# Here We are getting Google's financial information
GoogleInfo = yahooFinance.Ticker("GOOGL")
```

1.4.1.3 Whole Python Dictionary is Printed Here

```
print(GoogleInfo.info)
```

```
{'zip': '94043', 'sector': 'Communication Services', ...
'logo_url': 'https://logo.clearbit.com/abc.xyz', 'trailingPegRatio': 1.3474}
```

The print statement produces a Python dictionary, which we can analyze and use to get the specific financial data we're looking for from Yahoo Finance. Let's take a few financial critical metrics as an example.

The info dictionary contains all firm information. As a result, we may extract the desired elements from the dictionary by parsing it:

We can retrieve financial key metrics like Company Sector, Price Earnings Ratio, and Company Beta from the above dictionary of items easily. Let us see the below code.

```
# display Company Sector
print("Company Sector : ", GoogleInfo.info['sector'])

# display Price Earnings Ratio
print("Price Earnings Ratio : ", GoogleInfo.info['trailingPE'])

# display Company Beta
print(" Company Beta : ", GoogleInfo.info['beta'])
```

```
Company Sector :  Communication Services
Price Earnings Ratio :  1.6200992
 Company Beta :  1.078487
```

There are a ton of more stuff in the information. By printing the informational keys, we can view all of them:

```
# get all key value pairs that are available
for key, value in GoogleInfo.info.items():
    print(key, ":", value)
```

```
zip : 94043
sector : Communication Services
fullTimeEmployees : 174014
longBusinessSummary : Alphabet Inc. ... in Mountain View, California.
city : Mountain View
...
logo_url : https://logo.clearbit.com/abc.xyz
trailingPegRatio : 1.3474
```

We can retrieve historical market prices too and display them. Additionally, we can utilize it to get earlier market data.

We will use historical Google stock values over the past few years as our example. It is a relatively easy assignment to complete, as demonstrated below:

```
# covering the past few years.
# max->maximum number of daily prices available
# for Google.
# Valid options are 1d, 5d, 1mo, 3mo, 6mo, 1y, 2y,
# 5y, 10y and ytd.
print(GoogleInfo.history(period="max"))
```

```
                 Open        High         Low       Close     Volume  \
Date
2004-08-19    2.502503    2.604104    2.401401    2.511011  893181924
2004-08-20    2.527778    2.729730    2.515015    2.710460  456686856
2004-08-23    2.771522    2.839840    2.728979    2.737738  365122512
2004-08-24    2.783784    2.792793    2.591842    2.624374  304946748
2004-08-25    2.626627    2.702703    2.599600    2.652653  183772044
...                ...         ...         ...         ...        ...
2022-08-29  109.989998  110.949997  108.800003  109.419998   21191200
2022-08-30  110.169998  110.500000  107.800003  108.940002   27513300
2022-08-31  110.650002  110.849998  108.129997  108.220001   28627000
2022-09-01  108.279999  110.449997  107.360001  109.739998   28360900
2022-09-02  110.589996  110.739998  107.261597  107.849998   23528231

            Dividends  Stock Splits
Date
2004-08-19          0           0.0
2004-08-20          0           0.0
2004-08-23          0           0.0
2004-08-24          0           0.0
2004-08-25          0           0.0
...               ...           ...
2022-08-29          0           0.0
2022-08-30          0           0.0
2022-08-31          0           0.0
2022-09-01          0           0.0
2022-09-02          0           0.0

[4543 rows x 7 columns]
```

We can pass our own start and end dates.

```
import datetime

start = datetime.datetime(2012,5,31)
end = datetime.datetime(2013,1,30)
print(GoogleInfo.history(start=start, end=end))
```

```
                Open       High        Low      Close     Volume  Dividends  \
Date
2012-05-31  14.732733  14.764765  14.489489  14.536036  118613268          0
2012-06-01  14.309059  14.330581  14.222973  14.288789  122193684          0
2012-06-04  14.269770  14.526777  14.264515  14.479229   97210692          0
2012-06-05  14.400651  14.467718  14.175926  14.274525   93502404          0
2012-06-06  14.426426  14.563814  14.354605  14.528779   83748168          0
...               ...        ...        ...        ...        ...        ...
2013-01-23  18.418167  18.743744  18.413162  18.556055  236127636          0
2013-01-24  18.549549  18.939690  18.531281  18.874125  135172692          0
2013-01-25  18.788038  18.980982  18.775024  18.860611   88946964          0
2013-01-28  18.812813  18.908909  18.715965  18.787037   65018916          0
2013-01-29  18.687437  18.942694  18.682182  18.860861   69814116          0

            Stock Splits
Date
2012-05-31             0
2012-06-01             0
2012-06-04             0
2012-06-05             0
2012-06-06             0
...                  ...
2013-01-23             0
2013-01-24             0
2013-01-25             0
2013-01-28             0
2013-01-29             0

[166 rows x 7 columns]
```

We can simultaneously download historical prices for many stocks:

The code below Pandas DataFrame including the different price data for the requested stocks. We now select the individual stock by printing df.GOOGL to have the historical market data for Google:

```
df = yahooFinance.download("AMZN GOOGL",
    start="2019-01-01", end="2020-01-01",group_by="ticker")
print(df)
print(df.GOOGL)
```

```
[**********************100%**********************]  2 of 2 completed
                AMZN                                                    \
                Open       High        Low      Close  Adj Close     Volume
```

```
Date
2019-01-02  73.260002  77.667999  73.046501  76.956497  76.956497  159662000
2019-01-03  76.000504  76.900002  74.855499  75.014000  75.014000  139512000
2019-01-04  76.500000  79.699997  75.915497  78.769501  78.769501  183652000
2019-01-07  80.115501  81.727997  79.459503  81.475502  81.475502  159864000
2019-01-08  83.234497  83.830498  80.830498  82.829002  82.829002  177628000
...               ...        ...        ...        ...        ...        ...
2019-12-24  89.690498  89.778503  89.378998  89.460503  89.460503   17626000
2019-12-26  90.050499  93.523003  89.974998  93.438499  93.438499  120108000
2019-12-27  94.146004  95.070000  93.300499  93.489998  93.489998  123732000
2019-12-30  93.699997  94.199997  92.030998  92.344498  92.344498   73494000
2019-12-31  92.099998  92.663002  91.611504  92.391998  92.391998   50130000

               GOOGL
                Open       High        Low      Close  Adj Close    Volume
Date
2019-01-02  51.360001  53.039501  51.264000  52.734001  52.734001  31868000
2019-01-03  52.533501  53.313000  51.118500  51.273499  51.273499  41960000
2019-01-04  52.127998  54.000000  51.842999  53.903500  53.903500  46022000
2019-01-07  54.048500  54.134998  53.132000  53.796001  53.796001  47446000
2019-01-08  54.299999  54.667500  53.417500  54.268501  54.268501  35414000
...               ...        ...        ...        ...        ...        ...
2019-12-24  67.510498  67.600502  67.208504  67.221497  67.221497  13468000
2019-12-26  67.327499  68.160004  67.275497  68.123497  68.123497  23662000
2019-12-27  68.199997  68.352501  67.650002  67.732002  67.732002  23212000
2019-12-30  67.840500  67.849998  66.891998  66.985497  66.985497  19994000
2019-12-31  66.789497  67.032997  66.606499  66.969498  66.969498  19514000

[252 rows x 12 columns]
                Open       High        Low      Close  Adj Close    Volume
Date
2019-01-02  51.360001  53.039501  51.264000  52.734001  52.734001  31868000
2019-01-03  52.533501  53.313000  51.118500  51.273499  51.273499  41960000
2019-01-04  52.127998  54.000000  51.842999  53.903500  53.903500  46022000
2019-01-07  54.048500  54.134998  53.132000  53.796001  53.796001  47446000
2019-01-08  54.299999  54.667500  53.417500  54.268501  54.268501  35414000
...               ...        ...        ...        ...        ...        ...
2019-12-24  67.510498  67.600502  67.208504  67.221497  67.221497  13468000
2019-12-26  67.327499  68.160004  67.275497  68.123497  68.123497  23662000
2019-12-27  68.199997  68.352501  67.650002  67.732002  67.732002  23212000
2019-12-30  67.840500  67.849998  66.891998  66.985497  66.985497  19994000
2019-12-31  66.789497  67.032997  66.606499  66.969498  66.969498  19514000

[252 rows x 6 columns]
```

1.4.1.4 Save the Data to CSV

```
df.to_csv('data/FinanceData.csv')
```