3 Reading data

3.1 Types of data - structured and unstructured

Reading data is the first step to extract information from it. Data can exist broadly in two formats:

1. Structured data, and
2. Unstructured data.

Structured data is typically stored in a tabular form, where rows in the data correspond to “observations” and columns correspond to “variables”. For example, the following dataset contains 5 observations, where each observation (or row) consists of information about a movie. The variables (or columns) contain different pieces of information about a given movie. As all variables for a given row are related to the same movie, the data below is also called relational data.

	Title	US Gross	Production Budget	Release Date	Major Genre	Creative Type	Rotten Tomatoes Rating	I R
0	The Shawshank Redemption	28241469	25000000	Sep 23 1994	Drama	Fiction	88	9
1	Inception	285630280	160000000	Jul 16 2010	Horror/Thriller	Fiction	87	9
2	One Flew Over the Cuckoo's Nest	108981275	4400000	Nov 19 1975	Comedy	Fiction	96	8
3	The Dark Knight	533345358	185000000	Jul 18 2008	Action/Adventure	Fiction	93	8
4	Schindler's List	96067179	25000000	Dec 15 1993	Drama	Non-Fiction	97	8

Unstructured data is data that is not organized in any pre-defined manner. Examples of unstructured data can be text files, audio/video files, images, Internet of Things (IoT) data, etc. Unstructured data is relatively harder to analyze as most of the analytical methods and tools are oriented towards structured data.

However, an unstructured data can be used to obtain structured data, which in turn can be analyzed. For example, an image can be converted to an array of pixels - which will be structured data. Machine learning algorithms can then be used on the array to classify the image as that of a dog or a cat.

In this course, we will focus on analyzing structured data.

3.2 Reading a *csv* file with *Pandas*

Structured data can be stored in a variety of formats. The most popular format is *data_file_name.csv*, where the extension *csv* stands for comma separated values. The variable values of each observation are separated by a comma in a *.csv* file. In other words, the **delimiter** is a comma in a *csv* file. However, the comma is not visible when a *.csv* file is opened with Microsoft Excel.

3.2.1 Using the *read_csv* function

We will use functions from the *Pandas* library of *Python* to read data. Let us import *Pandas* to use its functions.

```
import pandas as pd
```

Note that *pd* is the acronym that we will use to call a *Pandas* function. This acronym can be anything as desired by the user.

The function to read a *csv* file is read_csv(). It reads the dataset into an object of type Pandas DataFrame. Let us read the dataset *movie_ratings.csv* in Python.

```
movie_ratings = pd.read_csv('movie_ratings.csv')
```

The built-in python function `type` can be used to check the datatype of an object:

```
type(movie_ratings)
```

```
pandas.core.frame.DataFrame
```

Note that the file *movie_ratings.csv* is stored at the same location as the python script containing the above code. If that is not the case, we'll need to specify the location of the file as in the following code.

```
movie_ratings = pd.read_csv('D:/Books/DataScience_Intro_python/r
```

Note that forward slash is used instead of backslash while specifying the path of the data file. Another option is to use two consecutive backslashes instead of a single forward slash.

3.2.2 Specifying the working directory

In case we need to read several datasets from a given location, it may be inconvenient to specify the path every time. In such a case we can change the current working directory to the location where the datasets are located.

We'll use the *os* library of *Python* to view and/or change the current working directory.

```
import os #Importing the 'os' library
os.getcwd() #Getting the path to the current working directory
```

```
C:\Users\username\STAT303-1\Quarto
Book\DataScience_Intro_python
```

The function *getcwd()* stands for get current working directory.

Suppose the dataset to be read is located at `'D:\Books\DataScience_Intro_python\Datasets'`. Then, we'll use the function *chdir* to change the current working directory to this location.

```
os.chdir('D:/Books/DataScience_Intro_python/Datasets')
```

Now we can read the dataset from this location without mentioning the entire path as shown below.

```
movie_ratings = pd.read_csv('movie_ratings.csv')
```

3.2.3 Data overview

Once the data has been read, we may want to see what the data looks like. We'll use another *Pandas* function head() to view the first few rows of the data.

```
movie_ratings.head()
```

	Title	US Gross	Worldwide Gross	Production Budget	Release Date	MPAA Rating	Source	Major Genre
0	Opal Dreams	14443	14443	9000000	Nov 22 2006	PG/PG-13	Adapted screenplay	Drama

1	Major Dundee	14873	14873	3800000	Apr 07 1965	PG/PG-13	Adapted screenplay	Western/Music
2	The Informers	315000	315000	18000000	Apr 24 2009	R	Adapted screenplay	Horror/Thriller
3	Buffalo Soldiers	353743	353743	15000000	Jul 25 2003	R	Adapted screenplay	Comedy
4	The Last Sin Eater	388390	388390	2200000	Feb 09 2007	PG/PG-13	Adapted screenplay	Drama

3.2.3.1 Row Indices and column names (axis labels)

The bold integers on the left are the indices of the DataFrame. Each index refers to a distinct row. For example, the index *2* correponds to the row of the movie *The Informers*. By default, the indices are integers starting from 0. However, they can be changed (to even non-integer values) if desired by the user.

The bold text on top of the DataFrame refers to column names. For example, the column *US Gross* consists of the gross revenue of a movie in the US.

Collectively, the indices and column names are referred as **axis labels**.

3.2.3.2 Shape of DataFrame

For finding the number of rows and columns in the data, you may use the shape() function.

```
#Finding the shape of movie_ratings dataset
movie_ratings.shape
```

```
(2228, 11)
```

The *movie_ratings* dataset contains 2,228 observations (or rows) and 11 variables (or columns).

3.2.4 Summary statistics

3.2.4.1 Numeric columns summary

The Pandas function of the DataFrame class, describe() can be used very conviniently to print the summary statistics of numeric columns of the data.

```
#Finding summary statistics of movie_ratings dataset
movie_ratings.describe()
```

Table 3.1: Summary statistics of numeric variables

	US Gross	Worldwide Gross	Production Budget	IMDB Rating	IMDB Votes	Release Year
count	2.228000e+03	2.228000e+03	2.228000e+03	2228.000000	2228.000000	2228.0000
mean	5.076370e+07	1.019370e+08	3.816055e+07	6.239004	33585.154847	2002.0053
std	6.643081e+07	1.648589e+08	3.782604e+07	1.243285	47325.651561	5.524324
min	0.000000e+00	8.840000e+02	2.180000e+02	1.400000	18.000000	1953.0000
25%	9.646188e+06	1.320737e+07	1.200000e+07	5.500000	6659.250000	1999.0000
50%	2.838649e+07	4.266892e+07	2.600000e+07	6.400000	18169.000000	2002.0000
75%	6.453140e+07	1.200000e+08	5.300000e+07	7.100000	40092.750000	2006.0000
max	7.601676e+08	2.767891e+09	3.000000e+08	9.200000	519541.000000	2039.0000

Answer the following questions based on the above table.

Do majority of the movies have a higher IMDB rating than the average IMDB rating of a movie?

No

Yes

What is approximately the average profit of a movie? Profit = Worldwide gross - Production budget

\$127 million

\$64 million

\$17 million

Can you find the maximum profit of a movie?

No

Yes

Suppose the movies are arrange in decreasing order of IMDB ratings. What is the maximum IMDB rating of a movie that is not in the top 25% movies?

Type numeric answer here:

3.2.4.2 Summary statistics across rows/columns

The Pandas DataFrame class has functions such as sum() and mean() to compute sum over rows or columns of a DataFrame.

Let us compute the mean of all the numeric columns of the data:

```
movie_ratings.mean(axis = 0)
```

```
US Gross             5.076370e+07
Worldwide Gross      1.019370e+08
Production Budget    3.816055e+07
IMDB Rating          6.239004e+00
IMDB Votes           3.358515e+04
dtype: float64
```

The argument `axis=0` denotes that the mean is taken over all the rows of the DataFrame. For computing a statistic across columns the argument `axis=1` will be used.

If mean over a subset of columns is desired, then those column names can be subset from the data. For example, let us compute the mean IMDB rating, and mean IMDB votes of all the movies:

```
movie_ratings[['IMDB Rating','IMDB Votes']].mean(axis = 0)
```

```
IMDB Rating        6.239004
IMDB Votes     33585.154847
```

```
dtype: float64
```

3.2.5 Practice exercise 1

Read the file *Top 10 Albums By Year.csv*. This file contains the top 10 albums for each year from 1990 to 2021. Each row corresponds to a unique album.

3.2.5.1

Print the first 5 rows of the data.

```
album_data = pd.read_csv('./Datasets/Top 10 Albums By Year.csv')
album_data.head()
```

	Year	Ranking	Artist	Album	Worldwide Sales	CDs	Tracks	Album Length	Hours	Minu
0	1990	8	Phil Collins	Serious Hits... Live!	9956520	1	15	1:16:53	1.28	76.88
1	1990	1	Madonna	The Immaculate Collection	30000000	1	17	1:13:32	1.23	73.53
2	1990	10	The Three Tenors	Carreras Domingo Pavarotti In Concert 1990	8533000	1	17	1:07:55	1.13	67.92
3	1990	4	MC Hammer	Please Hammer Don't Hurt Em	18000000	1	13	0:59:04	0.98	59.07
4	1990	6	Movie Soundtrack	Aashiqui	15000000	1	12	0:58:13	0.97	58.22

3.2.5.2

How many rows and columns are there in the data?

```
album_data.shape
```

```
(320, 12)
```

There are 320 rows and 12 columns in the data

3.2.5.3

Print the summary statistics of the data, and answer the following questions:

1. What proportion of albums have 15 or lesser tracks? Mention a range for the proportion.
2. What is the mean length of a track (in minutes)?

```
album_data.describe()
```

	Year	Ranking	CDs	Tracks	Hours	Minutes	Seconds
count	320.000000	320.00000	320.000000	320.000000	320.000000	320.000000	320.00000
mean	2005.500000	5.50000	1.043750	14.306250	0.941406	56.478500	3388.7156
std	9.247553	2.87678	0.246528	5.868995	0.382895	22.970109	1378.2098
min	1990.000000	1.00000	1.000000	6.000000	0.320000	19.430000	1166.0000
25%	1997.750000	3.00000	1.000000	12.000000	0.740000	44.137500	2648.2500
50%	2005.500000	5.50000	1.000000	13.000000	0.860000	51.555000	3093.5000
75%	2013.250000	8.00000	1.000000	15.000000	1.090000	65.112500	3906.7500
max	2021.000000	10.00000	4.000000	67.000000	5.070000	304.030000	18242.000

At least 75% of the albums have 15 tracks or lesser tracks since the 75th percentile value of the number of tracks is 15. However, albums between those having 75th percentile value for the number of tracks and those having the maximum number of tracks can also have 15 tracks. Thus, the proportion of albums having 15 or lesser tracks = [75%-99.99%].

```
print("Mean length of a track =",56.478500/14.306250, "minutes"
```

```
Mean length of a track = 3.9478200087374398 minutes
```

3.2.6 Creating new columns from existing columns

New variables (or columns) can be created based on existing variables, or with external data (we'll see adding external data later). For example, let us create a new variable `ratio_wgross_by_budget`, which is the ratio of `Worldwide Gross` and `Production Budget` for each movie:

```
movie_ratings['ratio_wgross_by_budget'] = movie_ratings['Worldw
```

The new variable can be seen at the right end of the updated DataFrame as shown below.

```
movie_ratings.head()
```

	Title	US Gross	Worldwide Gross	Production Budget	Release Date	MPAA Rating	Source	Major Genre
0	Opal Dreams	14443	14443	9000000	Nov 22 2006	PG/PG-13	Adapted screenplay	Drama
1	Major Dundee	14873	14873	3800000	Apr 07 1965	PG/PG-13	Adapted screenplay	Western/Music
2	The Informers	315000	315000	18000000	Apr 24 2009	R	Adapted screenplay	Horror/Thriller
3	Buffalo Soldiers	353743	353743	15000000	Jul 25 2003	R	Adapted screenplay	Comedy
4	The Last Sin Eater	388390	388390	2200000	Feb 09 2007	PG/PG-13	Adapted screenplay	Drama

3.2.7 Datatype of variables

Note that in Table 3.1 (summary statistics), we don't see `Release Date`. This is because the datatype of `Release Data` is not `numeric`.

The datatype of each variable can be seen using the dtypes() function of the DataFrame class.

```
#Checking the datatypes of the variables
movie_ratings.dtypes
```

```
Title                object
US Gross              int64
Worldwide Gross       int64
Production Budget     int64
Release Date         object
MPAA Rating          object
Source               object
Major Genre          object
Creative Type        object
IMDB Rating         float64
IMDB Votes            int64
dtype: object
```

Often, we wish to convert the datatypes of some of the variables to make them suitable for analysis. For example, the datatype of `Release Date` in the DataFrame *movie_ratings* is `object`. To perform numerical computations on this variable, we'll need to convert it to a `datetime` format. We'll use the Pandas function to_datetime() to covert it to a `datetime` format. Similar functions such as to_numeric(), to_string() etc., can be used for other conversions.

```
pd.to_datetime(movie_ratings['Release Date'])
```

```
0      2006-11-22
1      1965-04-07
2      2009-04-24
3      2003-07-25
4      2007-02-09
          ...
2223   2004-07-07
2224   1998-06-19
2225   2010-05-14
2226   1991-06-14
2227   1998-01-23
Name: Release Date, Length: 2228, dtype: datetime64[ns]
```

We can see above that the function `to_datetime()` converts *Release Date* to a `datetime` format.

Now, we'll update the variable `Release Date` in the DataFrame to be in the `datetime` format:

```
movie_ratings['Release Date'] = pd.to_datetime(movie_ratings['R
```

```
movie_ratings.dtypes
```

```
Title                        object
US Gross                      int64
Worldwide Gross               int64
Production Budget             int64
Release Date         datetime64[ns]
MPAA Rating                  object
Source                       object
Major Genre                  object
Creative Type                object
IMDB Rating                 float64
IMDB Votes                    int64
dtype: object
```

We can see that the datatype of *Release Date* has changed to `datetime` in the updated DataFrame, *movie_ratings*. Now we can perform computations on `Release Date`. Suppose we wish to create a new variable `Release_year` that consists of the year of release of the movie. We'll use the attribute `year` of the datetime module to extract the year from `Release Date`:

```
#Extracting year from Release Date
movie_ratings['Release Year'] = movie_ratings['Release Date'].d
```

```
movie_ratings.head()
```

	Title	US Gross	Worldwide Gross	Production Budget	Release Date	MPAA Rating	Source	Major Genre
0	Opal Dreams	14443	14443	9000000	2006-11-22	PG/PG-13	Adapted screenplay	Drama
1	Major Dundee	14873	14873	3800000	1965-04-07	PG/PG-13	Adapted screenplay	Western/Music
2	The Informers	315000	315000	18000000	2009-04-24	R	Adapted screenplay	Horror/Thriller
3	Buffalo Soldiers	353743	353743	15000000	2003-07-25	R	Adapted screenplay	Comedy
4	The Last Sin Eater	388390	388390	2200000	2007-02-09	PG/PG-13	Adapted screenplay	Drama

As year is a numeric variable, it will appear in the numeric summary statistics with the `describe()` function, as shown below.

```
movie_ratings.describe()
```

	US Gross	Worldwide Gross	Production Budget	IMDB Rating	IMDB Votes	Release Year
count	2.228000e+03	2.228000e+03	2.228000e+03	2228.000000	2228.000000	2228.0000
mean	5.076370e+07	1.019370e+08	3.816055e+07	6.239004	33585.154847	2002.0053
std	6.643081e+07	1.648589e+08	3.782604e+07	1.243285	47325.651561	5.524324
min	0.000000e+00	8.840000e+02	2.180000e+02	1.400000	18.000000	1953.0000
25%	9.646188e+06	1.320737e+07	1.200000e+07	5.500000	6659.250000	1999.0000
50%	2.838649e+07	4.266892e+07	2.600000e+07	6.400000	18169.000000	2002.0000

75%	6.453140e+07	1.200000e+08	5.300000e+07	7.100000	40092.750000	2006.0000
max	7.601676e+08	2.767891e+09	3.000000e+08	9.200000	519541.000000	2039.0000

3.2.8 Practice exercise 2

3.2.8.1

Why is `Worldwide Sales` not included in the summary statistics table printed in Practice exercise 1?

```
album_data.dtypes
```

```
Year                int64
Ranking             int64
Artist             object
Album              object
Worldwide Sales    object
CDs                 int64
Tracks              int64
Album Length       object
Hours             float64
Minutes           float64
Seconds             int64
Genre              object
dtype: object
```

`Worldwide Sales` is not included in the summary statistics table printed in Practice exercise 1 because its data type is `object` and not `int` or `float`

3.2.8.2

Update the DataFrame so that `Worldwide Sales` is included in the summary statistics table. Print the summary statistics table.

Hint: Sometimes it may not be possible to convert an object to numeric(). For example, the object 'hi' cannot be converted to a numeric() by the python compiler. To avoid getting an error, use the *errors* argument of to_numeric() to force such conversions to NaN (missing value).

```
album_data['Worldwide Sales'] = pd.to_numeric(album_data['Worldw
album_data.describe()
```

	Worldwide

	Year	Ranking	Sales	CDs	Tracks	Hours	Minute
count	320.000000	320.00000	3.190000e+02	320.000000	320.000000	320.000000	320.000
mean	2005.500000	5.50000	1.071093e+07	1.043750	14.306250	0.941406	56.4785
std	9.247553	2.87678	7.566796e+06	0.246528	5.868995	0.382895	22.9701
min	1990.000000	1.00000	1.909009e+06	1.000000	6.000000	0.320000	19.4300
25%	1997.750000	3.00000	5.000000e+06	1.000000	12.000000	0.740000	44.1375
50%	2005.500000	5.50000	8.255866e+06	1.000000	13.000000	0.860000	51.5550
75%	2013.250000	8.00000	1.400000e+07	1.000000	15.000000	1.090000	65.1125
max	2021.000000	10.00000	4.500000e+07	4.000000	67.000000	5.070000	304.030

3.2.8.3

Create a new column that computes the average worldwide sales per year for each album, assuming that the worldwide sales are as of 2022. Print the first 5 rows of the updated DataFrame.

```
album_data['mean_sales_per_year'] = album_data['Worldwide Sales
album_data.head()
```

	Year	Ranking	Artist	Album	Worldwide Sales	CDs	Tracks	Album Length	Hours	Minu
0	1990	8	Phil Collins	Serious Hits... Live!	9956520.0	1	15	1:16:53	1.28	76.88
1	1990	1	Madonna	The Immaculate Collection	30000000.0	1	17	1:13:32	1.23	73.53
2	1990	10	The Three Tenors	Carreras Domingo Pavarotti In Concert 1990	8533000.0	1	17	1:07:55	1.13	67.92
3	1990	4	MC Hammer	Please Hammer Don't Hurt Em	18000000.0	1	13	0:59:04	0.98	59.07
4	1990	6	Movie Soundtrack	Aashiqui	15000000.0	1	12	0:58:13	0.97	58.22

3.2.9 Reading a sub-set of data: `loc` and `iloc`

Sometimes we may be interested in working with a subset of rows and columns of the data, instead of working with the entire dataset. The indexing operators loc and iloc provide a convenient way of selecting a subset of desired rows and columns. The operator `loc` uses axis labels (row indices and column names) to subset the data, while `iloc` uses the position of rows or columns, where position has values 0,1,2,3,…and so on, for rows from top to bottom and columns from left to right. In other words, the first row has position 0, the second row has position 1, the third row has position 2, and so on. Similarly, the first column from left has position 0, the second column from left has position 1, the third column from left has position 2, and so on.

Let us read the file *movie_IMDBratings_sorted.csv*, which has movies sorted in the descending order of their IMDB ratings.

```
movies_sorted = pd.read_csv('./Datasets/movie_IMDBratings_sorte
```

The argument `index_col=0` assigns the first column of the file as the row indices of the DataFrame.

```
movies_sorted.head()
```

	Title	US Gross	Worldwide Gross	Production Budget	Release Date	MPAA Rating	Source	Ma
Rank								
1	The Shawshank Redemption	28241469	28241469	25000000	Sep 23 1994	R	Adapted screenplay	Dra
2	Inception	285630280	753830280	160000000	Jul 16 2010	PG/PG-13	Original Screenplay	Ho
3	The Dark Knight	533345358	1022345358	185000000	Jul 18 2008	PG/PG-13	Adapted screenplay	Act
4	Schindler's List	96067179	321200000	25000000	Dec 15 1993	R	Adapted screenplay	Dra
5	Pulp Fiction	107928762	212928762	8000000	Oct 14 1994	R	Original Screenplay	Dra

Let us say, we wish to subset the title, worldwide gross, production budget, and IMDB rating of top 3 movies.

```
# Subsetting the DataFrame by loc - using axis labels
movies_subset = movies_sorted.loc[1:3,['Title','Worldwide Gross
movies_subset
```

Rank	Title	Worldwide Gross	Production Budget	IMDB Rating
1	The Shawshank Redemption	28241469	25000000	9.2
2	Inception	753830280	160000000	9.1
3	The Dark Knight	1022345358	185000000	8.9

```
# Subsetting the DataFrame by iloc - using index of the positio
movies_subset = movies_sorted.iloc[0:3,[0,2,3,9]]
movies_subset
```

Rank	Title	Worldwide Gross	Production Budget	IMDB Rating
1	The Shawshank Redemption	28241469	25000000	9.2
2	Inception	753830280	160000000	9.1
3	The Dark Knight	1022345358	185000000	8.9

Let us find the movie with the maximum `Worldwide Gross`.

We will use the argmax() function of the Pandas Series class to find the position of the movie with the maximum worldwide gross, and then use the position to find the movie.

```
position_max_wgross = movies_sorted['Worldwide Gross'].argmax()
```

```
movies_sorted.iloc[position_max_wgross,:]
```

```
Title                          Avatar
US Gross                    760167650
Worldwide Gross            2767891499
Production Budget           237000000
Release Date              Dec 18 2009
MPAA Rating                 PG/PG-13
Source              Original Screenplay
Major Genre            Action/Adventure
Creative Type                 Fiction
```

```
IMDB Rating                          8.3
IMDB Votes                        261439
Name: 59, dtype: object
```

Avatar has the highest worldwide gross of all the movies. Note that the `:` indicates that all the columns of the DataFrame are selected.

Key differences between `loc` and `iloc` in pandas

- **Indexing Type:**
  - loc uses labels (names) for indexing.
  - iloc uses integer positions for indexing.
- **Inclusion of Endpoints:**
  - In a loc slice, both endpoints are included.
  - In an iloc slice, the last endpoint is excluded.

Examples: Assuming you have a DataFrame like this:

```
data = {'A': [1, 2, 3, 4, 5],
        'B': [10, 20, 30, 40, 50],
        'C': [100, 200, 300, 400, 500]}

df = pd.DataFrame(data, index=['row1', 'row2', 'row3', 'row4',
df.style.set_properties({'text-align': 'right'})
df
```

	A	B	C
row1	1	10	100
row2	2	20	200
row3	3	30	300
row4	4	40	400
row5	5	50	500

```
# using 'loc'
df.loc['row2':'row4', 'B']
```

```
row2    20
row3    30
row4    40
Name: B, dtype: int64
```

```
# using 'iloc'
df.iloc[1:4, 1]
```

```
row2    20
row3    30
row4    40
Name: B, dtype: int64
```

Note that in the `loc` example, both 'row2' and 'row4' are included in the result, whereas in the `iloc` example, the rows at integer positions 1, 2 and 3 are included, but the row at position 4 is excluded.

3.2.10 Practice exercise 3

3.2.10.1

Find the album having the highest worldwide sales per year, and its artist.

```
album_data.iloc[album_data['mean_sales_per_year'].argmax(),:]
```

```
Year                      2021
Ranking                      1
Artist                   Adele
Album                       30
Worldwide Sales        4485025.0
CDs                          1
Tracks                      12
Album Length           0:58:14
Hours                     0.97
Minutes                  58.23
Seconds                   3494
Genre                      Pop
mean_sales_per_year    4485025.0
Name: 312, dtype: object
```

'30' has the highest worldwide sales and its artist is Adele.

3.2.10.2

Subset the data to include only Hip-Hop albums. How many Hip_Hop albums are there?

```
hiphop_albums = album_data.loc[album_data['Genre']=='Hip Hop',:]
print("There are",hiphop_albums.shape[0], "hip-hop albums")
```

```
There are 42 hip-hop albums
```

3.2.10.3

Which album amongst hip-hop has the higest mean sales per year per track, and who is its artist?

```
hiphop_albums.loc[:,'mean_sales_per_year_track'] = hiphop_albums
hiphop_albums.iloc[hiphop_albums['mean_sales_per_year_track'].a
```

```
Year                                2021
Ranking                                6
Artist                        Cai Xukun
Album                                 迷
Worldwide Sales                3402981.0
CDs                                    1
Tracks                                11
Album Length                     0:24:16
Hours                                0.4
Minutes                            24.27
Seconds                             1456
Genre                           Hip Hop
mean_sales_per_year            3402981.0
mean_sales_per_year_track    309361.909091
Name: 318, dtype: object
```

迷 has the higest mean sales per year per track amongst hip-hop albumns, and its artist is Cai Xukun.

3.3 Reading other data formats - txt, html, json

Although *csv* is a very popular format for structured data, data is found in several other formats as well. Some of the other data formats are *txt, html* and *json*.

3.3.1 Reading *txt* files

The *txt* format offers some additional flexibility as compared to the *csv* format. In the *csv* format, the delimiter is a comma (or the column values are separated by a comma). However, in a *txt* file, the delimiter can be anything as desired by the user. Let us read the file *movie_ratings.txt*, where the variable values are separated by a tab character.

```
movie_ratings_txt = pd.read_csv('movie_ratings.txt',sep='\t')
```

We use the function *read_csv* to read a *txt* file. However, we mention the tab character (r") as a separator of variable values.

Note that there is no need to remember the argument name - *sep* for specifying the delimiter. You can always refer to the read_csv() documentation to find the relevant argument.

3.3.2 Practice exercise 4

Read the file *bestseller_books.txt*. It contains top 50 best-selling books on amazon from 2009 to 2019. Identify the delimiter without opening the file with Notepad or a text-editing software. How many rows and columns are there in the dataset?

Solution:

```
#Reading some lines with 'error_bad_lines=False' to identify the
bestseller_books = pd.read_csv('./Datasets/bestseller_books.txt
bestseller_books.head()
```

```
b'Skipping line 6: expected 1 fields, saw 2\nSkipping line 10:
expected 1 fields, saw 3\nSkipping line 16: expected 1 fields,
saw 5\nSkipping line 17: expected 1 fields, saw 4\nSkipping
line 20: expected 1 fields, saw 3\nSkipping line 29: expected 1
fields, saw 2\nSkipping line 33: expected 1 fields, saw
2\nSkipping line 40: expected 1 fields, saw 2\nSkipping line
41: expected 1 fields, saw 2\nSkipping line 42: expected 1
fields, saw 3\nSkipping line 43: expected 1 fields, saw
3\nSkipping line 44: expected 1 fields, saw 2\nSkipping line
60: expected 1 fields, saw 4\nSkipping line 61: expected 1
fields, saw 3\nSkipping line 63: expected 1 fields, saw
2\nSkipping line 64: expected 1 fields, saw 2\nSkipping line
70: expected 1 fields, saw 3\nSkipping line 71: expected 1
fields, saw 2\nSkipping line 72: expected 1 fields, saw
2\nSkipping line 73: expected 1 fields, saw 2\nSkipping line
80: expected 1 fields, saw 4\nSkipping line 82: expected 1
fields, saw 2\nSkipping line 94: expected 1 fields, saw
4\nSkipping line 95: expected 1 fields, saw 2\nSkipping line
96: expected 1 fields, saw 2\nSkipping line 101: expected 1
fields, saw 3\nSkipping line 119: expected 1 fields, saw
3\nSkipping line 130: expected 1 fields, saw 2\nSkipping line
131: expected 1 fields, saw 2\nSkipping line 132: expected 1
fields, saw 2\nSkipping line 133: expected 1 fields, saw
2\nSkipping line 148: expected 1 fields, saw 3\nSkipping line
149: expected 1 fields, saw 3\nSkipping line 150: expected 1
```

```
fields, saw 3\nSkipping line 154: expected 1 fields, saw
3\nSkipping line 155: expected 1 fields, saw 2\nSkipping line
156: expected 1 fields, saw 3\nSkipping line 157: expected 1
fields, saw 2\nSkipping line 158: expected 1 fields, saw
2\nSkipping line 159: expected 1 fields, saw 2\nSkipping line
177: expected 1 fields, saw 4\nSkipping line 178: expected 1
fields, saw 2\nSkipping line 179: expected 1 fields, saw
2\nSkipping line 183: expected 1 fields, saw 3\nSkipping line
209: expected 1 fields, saw 2\nSkipping line 215: expected 1
fields, saw 3\nSkipping line 224: expected 1 fields, saw
3\nSkipping line 230: expected 1 fields, saw 2\nSkipping line
241: expected 1 fields, saw 2\nSkipping line 247: expected 1
fields, saw 2\nSkipping line 248: expected 1 fields, saw
2\nSkipping line 249: expected 1 fields, saw 2\nSkipping line
250: expected 1 fields, saw 2\nSkipping line 251: expected 1
fields, saw 2\nSkipping line 252: expected 1 fields, saw
2\nSkipping line 253: expected 1 fields, saw 2\nSkipping line
254: expected 1 fields, saw 2\nSkipping line 259: expected 1
fields, saw 3\nSkipping line 273: expected 1 fields, saw
2\nSkipping line 274: expected 1 fields, saw 2\nSkipping line
275: expected 1 fields, saw 2\nSkipping line 276: expected 1
fields, saw 2\nSkipping line 277: expected 1 fields, saw
2\nSkipping line 278: expected 1 fields, saw 2\nSkipping line
279: expected 1 fields, saw 2\nSkipping line 280: expected 1
fields, saw 2\nSkipping line 281: expected 1 fields, saw
2\nSkipping line 282: expected 1 fields, saw 2\nSkipping line
292: expected 1 fields, saw 4\nSkipping line 293: expected 1
fields, saw 4\nSkipping line 295: expected 1 fields, saw
7\nSkipping line 296: expected 1 fields, saw 7\nSkipping line
297: expected 1 fields, saw 2\nSkipping line 302: expected 1
fields, saw 3\nSkipping line 315: expected 1 fields, saw
3\nSkipping line 321: expected 1 fields, saw 2\nSkipping line
346: expected 1 fields, saw 3\nSkipping line 347: expected 1
fields, saw 3\nSkipping line 365: expected 1 fields, saw
2\nSkipping line 408: expected 1 fields, saw 2\nSkipping line
420: expected 1 fields, saw 2\nSkipping line 421: expected 1
fields, saw 2\nSkipping line 430: expected 1 fields, saw
2\nSkipping line 434: expected 1 fields, saw 2\nSkipping line
446: expected 1 fields, saw 2\nSkipping line 448: expected 1
fields, saw 2\nSkipping line 449: expected 1 fields, saw
4\nSkipping line 451: expected 1 fields, saw 3\nSkipping line
458: expected 1 fields, saw 2\nSkipping line 459: expected 1
fields, saw 2\nSkipping line 460: expected 1 fields, saw
2\nSkipping line 465: expected 1 fields, saw 2\nSkipping line
470: expected 1 fields, saw 2\nSkipping line 471: expected 1
fields, saw 2\nSkipping line 476: expected 1 fields, saw
2\nSkipping line 495: expected 1 fields, saw 2\nSkipping line
496: expected 1 fields, saw 2\nSkipping line 497: expected 1
```

```
fields, saw 2\nSkipping line 512: expected 1 fields, saw
5\nSkipping line 513: expected 1 fields, saw 2\nSkipping line
515: expected 1 fields, saw 2\nSkipping line 517: expected 1
fields, saw 3\nSkipping line 518: expected 1 fields, saw
3\nSkipping line 519: expected 1 fields, saw 3\nSkipping line
520: expected 1 fields, saw 3\nSkipping line 521: expected 1
fields, saw 3\nSkipping line 525: expected 1 fields, saw
3\nSkipping line 533: expected 1 fields, saw 3\nSkipping line
534: expected 1 fields, saw 3\n'
```

	;Unnamed: 0;Name;Author;User Rating;Reviews;Price;Year;Genre
0	0;0;10-Day Green Smoothie Cleanse;JJ Smith;4.7...
1	1;1;11/22/63: A Novel;Stephen King;4.6;2052;22...
2	2;2;12 Rules for Life: An Antidote to Chaos;Jo...
3	3;3;1984 (Signet Classics);George Orwell;4.7;2...
4	5;5;A Dance with Dragons (A Song of Ice and Fi...

```
#The delimiter seems to be ';' based on the output of the above
bestseller_books = pd.read_csv('./Datasets/bestseller_books.txt
bestseller_books.head()
```

	Unnamed: 0	Unnamed: 0.1	Name	Author	User Rating	Reviews	Price	Year	Genre
0	0	0	10-Day Green Smoothie Cleanse	JJ Smith	4.7	17350	8	2016	Non Fiction
1	1	1	11/22/63: A Novel	Stephen King	4.6	2052	22	2011	Fiction
2	2	2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7	18979	15	2018	Non Fiction
3	3	3	1984 (Signet Classics)	George Orwell	4.7	21424	6	2017	Fiction
4	4	4	5,000 Awesome Facts (About Everything!) (Natio...	National Geographic Kids	4.8	7665	12	2019	Non Fiction

```
#The file read with ';' as the delimited is correct
print("The file has",bestseller_books.shape[0],"rows and",bestse
```

```
The file has 550 rows and 9 columns
```

Alternatively, you can use the argument `sep = None`, and `engine = 'python'`. The default engine is C. However, the 'python' engine has a 'sniffer' tool which may identify the delimiter automatically.

```
bestseller_books = pd.read_csv('./Datasets/bestseller_books.txt
bestseller_books.head()
```

	Unnamed: 0	Unnamed: 0.1	Name	Author	User Rating	Reviews	Price	Year	Genre
0	0	0	10-Day Green Smoothie Cleanse	JJ Smith	4.7	17350	8	2016	Non Fiction
1	1	1	11/22/63: A Novel	Stephen King	4.6	2052	22	2011	Fiction
2	2	2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7	18979	15	2018	Non Fiction
3	3	3	1984 (Signet Classics)	George Orwell	4.7	21424	6	2017	Fiction
4	4	4	5,000 Awesome Facts (About Everything!) (Natio...	National Geographic Kids	4.8	7665	12	2019	Non Fiction

3.3.3 Reading HTML data

The *Pandas* function *read_html* searches for tabular data, i.e., data contained within the *<table>* tags of an html file. Let us read the tables in the GDP per capita page on Wikipedia.

```
#Reading all the tables from the Wikipedia page on GDP per capi
tables = pd.read_html('https://en.wikipedia.org/wiki/List_of_co
```

All the tables will be read and stored in the variable named as *tables*. Let us find the datatype of the variable *tables*.

```
#Finidng datatype of the variable – tables
type(tables)
```

```
list
```

The variable - tables is a list of all the tables read from the HTML data.

```
#Number of tables read from the page
len(tables)
```

```
6
```

The in-built function *len* can be used to find the length of the list - *tables* or the number of tables read from the Wikipedia page. Let us check out the first table.

```
#Checking out the first table. Note that the index of the first
tables[0]
```

	0	1	2
0	.mw-parser-output .legend{page-break-inside:av...	$20,000 - $30,000 $10,000 - $20,000 $5,000 - $...	$1,000 - $2,500 $500 - $1,000 <$500 No data

The above table doesn't seem to be useful. Let us check out the second table.

```
#Checking out the second table. Note that the index of the firs
tables[1]
```

	Country/Territory	UN Region	IMF[4][5]		United Nations[6]		World Bank[7]	
	Country/Territory	UN Region	Estimate	Year	Estimate	Year	Estimate	Year
0	Liechtenstein*	Europe	—	—	180227	2020	169049	2019
1	Monaco*	Europe	—	—	173696	2020	173688	2020
2	Luxembourg*	Europe	135046	2022	117182	2020	135683	2021
3	Bermuda*	Americas	—	—	123945	2020	110870	2021
4	Ireland*	Europe	101509	2022	86251	2020	85268	2020

...	...	...	...	...	...	...	...	...
212	Central AfricanRepublic*	Africa	527	2022	481	2020	477	2020
213	Sierra Leone*	Africa	513	2022	475	2020	485	2020
214	Madagascar*	Africa	504	2022	470	2020	496	2020
215	South Sudan*	Africa	393	2022	1421	2020	1120	2015
216	Burundi*	Africa	272	2022	286	2020	274	2020

217 rows × 8 columns

The above table contains the estimated GDP per capita of all countries. This is the table that is likely to be relevant to a user interested in analyzing GDP per capita of countries. Instead of reading all tables of an HTML file, we can focus the search to tables containing certain relevant keywords. Let us try searching all table containing the word 'Country'.

```
#Reading all the tables from the Wikipedia page on GDP per capi
tables = pd.read_html('https://en.wikipedia.org/wiki/List_of_cou
```

The *match* argument can be used to specify the keywords to be present in the table to be read.

```
len(tables)
```

1

Only one table contains the keyword - 'Country'. Let us check out the table obtained.

```
#Table having the keyword – 'Country' from the HTML page
tables[0]
```

	Country/Territory	UN Region	IMF[4][5]		United Nations[6]		World Bank[7]	
	Country/Territory	**UN Region**	**Estimate**	**Year**	**Estimate**	**Year**	**Estimate**	**Year**
0	Liechtenstein*	Europe	—	—	180227	2020	169049	2019
1	Monaco*	Europe	—	—	173696	2020	173688	2020
2	Luxembourg*	Europe	135046	2022	117182	2020	135683	2021
3	Bermuda*	Americas	—	—	123945	2020	110870	2021

4	Ireland*	Europe	101509	2022	86251	2020	85268	2020
...	...	...	...	...	...	...	...	...
212	Central AfricanRepublic*	Africa	527	2022	481	2020	477	2020
213	Sierra Leone*	Africa	513	2022	475	2020	485	2020
214	Madagascar*	Africa	504	2022	470	2020	496	2020
215	South Sudan*	Africa	393	2022	1421	2020	1120	2015
216	Burundi*	Africa	272	2022	286	2020	274	2020

217 rows × 8 columns

The argument *match* helps with a more focussed search, and helps us discard irrelevant tables.

3.3.4 Practice exercise 5

Read the table(s) consisting of attendance of spectators in FIFA worlds cup from this page. Read only those table(s) that have the word *'attendance'* in them. How many rows and columns are there in the table(s)?

```
dfs = pd.read_html('https://en.wikipedia.org/wiki/FIFA_World_Cu
                   match='attendance')
print(len(dfs))
data = dfs[0]
print("Number of rows =",data.shape[0], "and number of columns='
```

```
1
Number of rows = 22 and number of columns= 9
```

3.3.5 Reading JSON data

JSON stands for JavaScript Object Notation, in which the data is stored and transmitted as plain text. A couple of benefits of the JSON format are:

1. Since the format is text only, JSON data can easily be exchanged between web applications, and used by any programming language.
2. Unlike the *csv* format, JSON supports a hierarchical data structure, and is easier to integrate with APIs.

The JSON format can support a hierachical data structure, as it is built on the following two data structures (*Source: technical documentation*):

- A collection of name/value pairs. In various languages, this is realized as an

object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

The *Pandas* function read_json converts a JSON string to a Pandas DataFrame. The function *dumps()* of the *json* library converts a Python object to a JSON string.

Lets read the JSON data on Ted Talks.

```
tedtalks_data = pd.read_json('https://raw.githubusercontent.com/
```

```
tedtalks_data.head()
```

	id	speaker	headline	URL	description	transcript_U
0	7	David Pogue	Simplicity sells	http://www.ted.com/talks/view/id/7	New York Times columnist David Pogue takes aim...	http://www.te
1	6	Craig Venter	Sampling the ocean's DNA	http://www.ted.com/talks/view/id/6	Genomics pioneer Craig Venter takes a break fr...	http://www.te
2	4	Burt Rutan	The real future of space exploration	http://www.ted.com/talks/view/id/4	In this passionate talk, legendary spacecraft ...	http://www.te
3	3	Ashraf Ghani	How to rebuild a broken state	http://www.ted.com/talks/view/id/3	Ashraf Ghani's passionate and powerful 10-minu...	http://www.te

4	5	Chris Bangle	Great cars are great art	http://www.ted.com/talks/view/id/5	American designer Chris Bangle explains his ph...	http://www.te

3.3.6 Practice exercise 6

Read the movies dataset from here. How many rows and columns are there in the data?

```
movies_data = pd.read_json('https://raw.githubusercontent.com/v
print("Number of rows =",movies_data.shape[0], "and number of c
```

```
Number of rows = 3201 and number of columns= 16
```

3.3.7 Reading data from web APIs

API, an acronym for Application programming interface, is a way of transferring information between systems. Many websites have public APIs that provide data via JSON or other formats. For example, the IMDb-API is a web service for receiving movies, serial, and cast information. API results are in the JSON format and include items such as movie specifications, ratings, Wikipedia page content, etc. One of these APIs contains ratings of the top 250 movies on IMDB. Let us read this data using the IMDB API.

We'll use the *get* function from the python library *requests* to request data from the API and obtain a response code. The response code will let us know if our request to pull data from this API was successful.

```
#Importing the requests library
import requests as rq
```

```
# Downloading imdb top 250 movie's data
url = 'https://imdb-api.com/en/API/Top250Movies/k_v6gf8ppf' #URI
response = rq.get(url) #Requesting data from the API
response
```

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

We have a response code of 200, which indicates that the request was successful.

The response object's JSON method will return a dictionary containing JSON parsed into native Python objects.

```
movie_data = response.json()
```

```
movie_data.keys()
```

```
dict_keys(['items', 'errorMessage'])
```

The *movie_data* contains only two keys. The *items* key seems likely to contain information about the top 250 movies. Let us convert the values of the *items* key (which is list of dictionaries) to a dataframe, so that we can view it in a tabular form.

```
#Converting a list of dictionaries to a dataframe
movie_data_df = pd.DataFrame(movie_data['items'])
```

```
#Checking the movie data pulled using the API
movie_data_df.head()
```

	id	rank	title	fullTitle	year	image
0	tt0111161	1	The Shawshank Redemption	The Shawshank Redemption (1994)	1994	https://m.media-amazon.com/images/M/MV5BMDFkYT...
1	tt0068646	2	The Godfather	The Godfather (1972)	1972	https://m.media-amazon.com/images/M/MV5BM2MyNj..
2	tt0468569	3	The Dark Knight	The Dark Knight (2008)	2008	https://m.media-amazon.com/images/M/MV5BMTMxNT..
3	tt0071562	4	The Godfather Part II	The Godfather Part II (1974)	1974	https://m.media-amazon.com/images/M/MV5BMWMwM(

4	tt0050083	5	12 Angry Men	12 Angry Men (1957)	1957	https://m.media-amazon.com/images/M/MV5BMWU4N2

```
#Rows and columns of the movie data
movie_data_df.shape
```

```
(250, 9)
```

This API provides the names of the top 250 movies along with the year of release, IMDB ratings, and cast information.

3.4 Writing data

The *Pandas* function *to_csv* can be used to write (or export) data to a *csv* or *txt* file. Below are some examples.

Example 1: Let us export the movies data of the top 250 movies to a *csv* file.

```
#Exporting the data of the top 250 movies to a csv file
movie_data_df.to_csv('movie_data_exported.csv')
```

The file *movie_data_exported.csv* will appear in the working directory.

Example 2: Let us export the movies data of the top 250 movies to a *txt* file with a semi-colon as the delimiter.

```
movie_data_df.to_csv('movie_data_exported.txt',sep=';')
```

Example 3: Let us export the movies data of the top 250 movies to a *JSON* file.

```
with open('movie_data.json', 'w') as f:
    json.dump(movie_data, f)
```