Day 3: Programming with Python

Pandas Basics

`Pandas` is one of the most used open-source Python libraries to work with structured tabular data for analysis. Pandas is widely used for data science, machine learning, and many more. It offers data structures and operations for manipulating numerical tables and time series.

In this unit we will see some of the basic functionalities of Pandas like how we can create Pandas data structures, view information on the data and columns, working with index, and how to combine different pandas data structures.

0. Import Pandas

The Pandas website where you can find everything about Pandas is here: https://pandas.pydata.org/.

We use import pandas as `pd` to import Pandas. The alias `pd` is commonly used for Pandas.

In [1]:

```
import pandas as pd
```

1. Pandas datatypes

Key points:

- **pd.Series()** - Creates a Series
- **pd.DataFrame()** - Creates a DataFrame

There are two main data structures in Pandas, `Series` and `DataFrames`.

`Series` is a one-dimensional data structure. It holds data of many types including objects, floats, strings and integers. `DataFrame` is a more complex data structure and is designed to contain data with several dimensions.

1.1 Series

To create a Series we call the `Series()` function and pass as an argument an array containing the values to be included in it.

Let's create our first Series from a list (listItems) that contains the numbers [1,3,6,9].

In [2]:

```
listItems = [1,3,6,9]
s = pd.Series(listItems)
print (s)
```

```
0    1
```

```
1    3
2    6
3    9
dtype: int64
```

The Series has a `dtype` , that refers to the type of the data.

In the above example, we see that the `dtype` is integer. This is because our list contained only integers.

Let's now try to create a Series from a list that contains the numbers [1.1,3.1,6.1,9.1] and let's check the if the type of the data has changed. What do you notice?

In [63]:

```
listItems = [1.1,3.1,6.1,9.1]
s = pd.Series(listItems)
print(s)
```

```
float64
```

Let's create a Series that contains ["Anna", "John", "Mark"]. What is going to be the type of the Series?

In [4]:

```
listItems = ["Anna", "John", "Mark"]
s = pd.Series(listItems)
print(s)
```

```
0    Anna
1    John
2    Mark
dtype: object
```

1.2 Pandas DataFrames

The `DataFrame` is a tabular data structure and is similar to a spreadsheet. It is a 2-dimensional data structure where each column contains data of the same type. DataFrames are great for representing real data: rows correspond to instances, and columns correspond to features of these instances.

For example, in case of movies, we have movies as rows and different metadata (title, release year, duration etc.) for each movie in the columns.

MovieID	title	year	duration
0	The Pianist	2002	150
1	Gladiator	2000	155
2	The Godfather	1972	177
3	Inception	2010	148
4	Titanic	1997	195

Create DataFrame from existing data structure

There are different ways to create a DataFrame. It can be created by typing the values, or from an existing data structure (list, dictionary) or imported from a file.

To create a DataFrame we use the `pd.DataFrame()` function.

Let's create a simple DataFrame from a single list, [1,3,6,9]

In [5]:

```
listItems = [1,3,6,9]
pd.DataFrame(listItems)
```

Out[5]:

	0
0	1
1	3
2	6
3	9

Let's create a second DataFrame from a dictionary. The dictionary contains the title, release year and duration in minutes of some movies.

In [6]:

```
data = {'title': ['The Pianist', 'Gladiator', 'The Godfather', 'Inception',
                  'Titanic'],
                    'year':[2002, 2000, 1972, 2010, 1997],
                   'duration':[150, 155, 177, 148, 195]}

data
```

Out[6]:

```
{'title': ['The Pianist',
  'Gladiator',
  'The Godfather',
  'Inception',
  'Titanic'],
 'year': [2002, 2000, 1972, 2010, 1997],
 'duration': [150, 155, 177, 148, 195]}
```

Now we create the DataFrame from the dictionary

In [7]:

```
df = pd.DataFrame(data)
print(df)
```

```
           title  year  duration
0    The Pianist  2002       150
1      Gladiator  2000       155
2  The Godfather  1972       177
3      Inception  2010       148
4        Titanic  1997       195
```

2. Index

Key points:

- **pd.Series(list, index)** - Creates a Series and assigns index
- **series.index = list** - Assigns index to Series
- **pd.DataFrame(dict, index)** - Creates a DataFrame and assigns index
- **df.index** - Returns the index of the DataFrame
- **df.values** - Returns the values of the DataFrame

Until now, every time we created a Series from a list there was an additional column on the left. This axis is called `index` and is added into the Series and DataFrames. Index is like an address for the stored data, and can be used to access any data point across the Series or DataFrame.

The index can be specified at the creation of the Series or the DataFrame. If we do not specify any index during the construction of the Pandas object then, by default, Pandas will assign numerical values increasing from 0 as labels.

In [8]:

```
s = pd.Series(listItems)
s
```

Out[8]:

```
0    1
1    3
2    6
3    9
dtype: int64
```

Let's create a Series that contains grades of a student (listItems) regarding the following 4 courses ['Math', 'Physics','Chemistry', 'History']. We can set the courses as index with the parameter `index`

In [9]:

```
s = pd.Series(listItems, index = ['Math', 'Physics','Chemistry', 'History'])
print(s)
```

```
Math         1
Physics      3
Chemistry    6
History      9
dtype: int64
```

The index can also be defined later with the `s.index` attribute.

In [10]:

```
s = pd.Series(listItems)
s.index = ['Math', 'Physics','Chemistry', 'History']
print(s)
```

```
Math         1
Physics      3
Chemistry    6
History      9
dtype: int64
```

In a similar way we define the index in DataFrames.

In [11]:

```
movies = ['The Pianist', 'Gladiator', 'The Godfather', 'Inception',
          'Titanic', 'Moulin Rouge', 'La La Land', 'The Notebook']
df = pd.DataFrame({'year':[2002, 2000, 1972, 2010, 1997, 2001, 2016, 2004],
                  'duration':[150, 155, 177, 148, 195, 130, 128, 124]},
                  index=movies)

print(df)
```

```
               year  duration
The Pianist    2002       150
Gladiator      2000       155
The Godfather  1972       177
Inception      2010       148
Titanic        1997       195
Moulin Rouge   2001       130
La La Land     2016       128
The Notebook   2004       124
```

We can use the `df.values` and the `df.index` attributes to view the value and index arrays

In [12]:

```
df.values
```

Out[12]:

```
array([[2002,  150],
       [2000,  155],
       [1972,  177],
       [2010,  148],
       [1997,  195],
       [2001,  130],
       [2016,  128],
       [2004,  124]])
```

In [13]:

```
df.index
```

Out[13]:

```
Index(['The Pianist', 'Gladiator', 'The Godfather', 'Inception', 'Titanic',
       'Moulin Rouge', 'La La Land', 'The Notebook'],
      dtype='object')
```

3. Knowing my data

Key points:

- **df.info()** - Prints a concise summary of a DataFrame
- **df.shape** - Returns a tuple representing the dimensionality of the DataFrame
- **df.columns** - Returns the column labels of the DataFrame
- **df.head(n)** - Returns the first n rows (default is 5)
- **df.tail(n)** - Returns the last n rows (default is 5)

Let's see how we can know more about the data

In [14]:

```
df = pd.DataFrame({'year':[2002, 2000, 1972, 2010, 1997, 2001, 2016, 2004],
                   'duration':[150, 155, 177, 148, 195, 130, 128, 124]},
                   index=movies)

print(df)
```

```
               year  duration
The Pianist    2002       150
Gladiator      2000       155
The Godfather  1972       177
Inception      2010       148
Titanic        1997       195
Moulin Rouge   2001       130
La La Land     2016       128
The Notebook   2004       124
```

There are some functions to get some basic information regarding our DataFrame.

We use the `info()` to output some general information about the DataFrame, the `shape` to get its dimensions, and the `columns` attribute to get the column labels of the DataFrame. Note that `shape` and `columns` are attributes and not functions and that is why they do not have parentheses.

With the `info()` we can see the columns of the DataFrame, the number of non-null values and the type of the data per column.

In [15]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8 entries, The Pianist to The Notebook
```

```
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   year      8 non-null      int64
 1   duration  8 non-null      int64
dtypes: int64(2)
memory usage: 192.0+ bytes
```

With the `shape` we get the dimensions of the DataFrame

In [16]:

```
df.shape
```

Out[16]:

```
(8, 2)
```

With the `columns` we get the names of the columns

In [17]:

```
df.columns
```

Out[17]:

```
Index(['year', 'duration'], dtype='object')
```

We can get the first rows of the DataFrame with the `head()` function. The `head()` function shows the first five rows if there is no argument or will show the mumber of rows specified in the argument. The function `tail()` shows the last entries of the DataFrame.

In [18]:

```
df.head()
```

Out[18]:

	year	duration
The Pianist	2002	150
Gladiator	2000	155
The Godfather	1972	177
Inception	2010	148
Titanic	1997	195

Viewing the first 3 rows

In [19]:

```
df.head(3)
```

Out[19]:

	year	duration
The Pianist	2002	150
Gladiator	2000	155
The Godfather	1972	177

To view the last rows we use the `tail()` function

In [20]:

```
df.tail()
```

Out[20]:

	year	duration
Inception	2010	148
Titanic	1997	195

	year	duration
Moulin Rouge	2001	130
La La Land	2016	128
The Notebook	2004	124

4. Working with columns

Key points:

- **df.column** - Returns the data of a column
- **df['column']** - Returns the data of a column
- **df[['column1', 'column2']]** - Returns the data of multiple columns
- **df['new_column'] = list** - Creates a new column and assigns data to it
- **del df['column']** - Deletes a column

For now we will work with only the DataFrame we created before.

In [21]:

```
df = pd.DataFrame({'year':[2002, 2000, 1972, 2010, 1997, 2001, 2016, 2004],
                   'duration':[150, 155, 177, 148, 195, 130, 128, 124]},
                   index=movies)

print(df)
```

```
               year  duration
The Pianist    2002       150
Gladiator      2000       155
The Godfather  1972       177
Inception      2010       148
Titanic        1997       195
Moulin Rouge   2001       130
La La Land     2016       128
The Notebook   2004       124
```

To select a column, we use the column name.

In [22]:

```
df.year
```

Out[22]:

```
The Pianist      2002
Gladiator        2000
The Godfather    1972
Inception        2010
Titanic          1997
Moulin Rouge     2001
La La Land       2016
The Notebook     2004
Name: year, dtype: int64
```

In [23]:

```
df['year']
```

Out[23]:

```
The Pianist      2002
Gladiator        2000
The Godfather    1972
Inception        2010
Titanic          1997
Moulin Rouge     2001
La La Land       2016
```

```
The Notebook     2004
Name: year, dtype: int64
```

If we want data of more columns we provide a list of the column names

In [24]:

```
df[['year','duration']]
```

Out[24]:

	year	duration
The Pianist	2002	150
Gladiator	2000	155
The Godfather	1972	177
Inception	2010	148
Titanic	1997	195
Moulin Rouge	2001	130
La La Land	2016	128
The Notebook	2004	124

Add a column

Let's say we want to add the column of `genre` in our DataFrame and we know that for the movies the genres are ['drama','action','crime','action','romance']

In [25]:

```
genre = ['drama','action','crime','action','romance',
         'drama', 'romance', 'romance']
df['genre'] = genre
df
```

Out[25]:

	year	duration	genre
The Pianist	2002	150	drama
Gladiator	2000	155	action
The Godfather	1972	177	crime
Inception	2010	148	action
Titanic	1997	195	romance
Moulin Rouge	2001	130	drama
La La Land	2016	128	romance
The Notebook	2004	124	romance

Delete a column

To delete a column, we use the keyword `del` followed by the column to be deleted (e.g., del df['genre'])

In [26]:

```
del df['genre']
df
```

Out[26]:

	year	duration

	year	duration
The Pianist	2002	150
Gladiator	2000	155
The Godfather	1972	177
Inception	2010	148
Titanic	1997	195
Moulin Rouge	2001	130
La La Land	2016	128
The Notebook	2004	124

5. Working with rows

Key points:

- **df.loc['index']** - Returns data of the row with index label index
- **df.loc[['index1', 'index2']]** - Returns data of multiple rows
- **df.loc['index1': 'index2']** - Returns data from row index1 until index2
- **df.loc['index1': 'index2', 'column']** - Returns data of a column and from row index1 until index2
- **df.loc[df['column'] >/</>=/<=/==]** - Returns data of a column based on a condition
- **df.iloc[indexID]** - Returns data of a row with specific index id

5.1. Select a row with loc

We can select a row using the `.loc` that can access group of values using labels. For example the code `df.loc['Gladiator']` will return the data of the row of `Gladiator`

In [27]:

```
df.loc['Gladiator']
```

Out[27]:

```
year        2000
duration     155
Name: Gladiator, dtype: int64
```

Let's say that now we want to return the data for Gladiator and Titanic. The command `df.loc['Gladiator', 'Titanic']` will return an error.

To return more rows we use a list of labels. Note that using `[ [  ] ]` returns a DataFrame.

In [28]:

```
df.loc[['Gladiator', 'Titanic']]
```

Out[28]:

	year	duration
Gladiator	2000	155
Titanic	1997	195

We can also access data with the slice. Let's return the data from `Gladiator` until `Inception`

In [29]:

```
df.loc['Gladiator':'Inception']
```

Out[29]:

	year	duration
Gladiator	2000	155
The Godfather	1972	177
Inception	2010	148

Also, we can select to get only the values of a specific column. The first argument is the rows and the second the columns that we want to access. So we want only the `duration` of the movies from `Gladiator` until `Inception`

In [30]:

```
df.loc['Gladiator':'Inception', 'duration']
```

Out[30]:

```
Gladiator        155
The Godfather    177
Inception        148
Name: duration, dtype: int64
```

5.2 Conditional selection

In different cases we want to select data that meet a specific condition. For example, we want to get the data of movies that last more that 2.5 hours. We can first set the expression `df['duration'] > 150`. This will return a boolen value for every row

In [31]:

```
df['duration'] > 150
```

Out[31]:

```
The Pianist      False
Gladiator         True
The Godfather     True
Inception        False
Titanic           True
Moulin Rouge     False
La La Land       False
The Notebook     False
Name: duration, dtype: bool
```

To get the rows we can pass the expression in the `loc` attribute. Only the rows with that were assigned as `True` will be printed

In [32]:

```
df.loc[df['duration'] > 150]
```

Out[32]:

	year	duration
Gladiator	2000	155
The Godfather	1972	177
Titanic	1997	195

5.3 Select a row with iloc

The `iloc` selects data only by integer index.

As we mentioned before every row and column in a DataFrame has an integer location that is assigned to it. This is even if we have assigned our own index. The integer location is simply the number of rows/columns from the top/left beginning at 0.

To return the third row we use `df.iloc[2]`

In [33]:

```
df.iloc[2]
```

Out[33]:

```
year        1972
duration     177
Name: The Godfather, dtype: int64
```

6. Combining DataFrames

Key points:

- **pd.concat([df1, df2])** - Concatenates two DataFrames
- **pd.merge(df1, df2)** - Merges two DataFrames similar to relational algebra

Pandas provides various facilities for combining together Series or DataFrames with various kinds of set logic for the indexes and relational algebra functionality in the case of merge-type operations.

6.1 Concatenate two DataFrames

The `concat()` function does all of the heavy lifting of performing concatenation operations along an axis while performing optional set logic (union or intersection) of the indexes (if any) on the other axes.

Here is a simple example. Let's say we have our DataFrame from before with movies.

In [34]:

```
df = pd.DataFrame({'year':[2002, 2000, 1972, 2010, 1997],
                   'duration':[150, 155, 177, 148, 195]},
                   index=['The Pianist', 'Gladiator', 'The Godfather',
                          'Inception', 'Titanic'])

df
```

Out[34]:

	year	duration
The Pianist	2002	150
Gladiator	2000	155
The Godfather	1972	177
Inception	2010	148
Titanic	1997	195

We also have data on more movies which maybe they came from a different source and we want to combine these two DataFrames. We have Nomadland (2021) that lasts 108 minutes and Parasite (2019) that lasts 132 minutes

In [35]:

```
df_new = pd.DataFrame({'year': [2021, 2019],
                      'duration': [108, 132]},
                     index=['Nomadland', 'Parasite'])

df_new
```

Out[35]:

	year	duration
Nomadland	2021	108
Parasite	2019	132

In order to concatenate the two DataFrames, we call the `concat()` function. The `concat()` function takes as argument the list of the DataFrames.

In [36]:

```
df_conc = pd.concat([df, df_new])
df_conc
```

Out[36]:

	year	duration
The Pianist	2002	150
Gladiator	2000	155
The Godfather	1972	177
Inception	2010	148
Titanic	1997	195
Nomadland	2021	108
Parasite	2019	132

Let's say we get data on the genres of the movies. We can concatenate the DataFrames again but not we have to set the axis to 1

In [37]:

```
df_new = pd.DataFrame({'genre': ['drama','action','crime','action','romance']
                   index=['The Pianist', 'Gladiator', 'The Godfather',
                          'Inception', 'Titanic'])


df_new
```

Out[37]:

	genre
The Pianist	drama
Gladiator	action
The Godfather	crime
Inception	action
Titanic	romance

In [38]:

```
df_conc = pd.concat([df, df_new])
df_conc
```

Out[38]:

	year	duration	genre
The Pianist	2002.0	150.0	NaN
Gladiator	2000.0	155.0	NaN
The Godfather	1972.0	177.0	NaN
Inception	2010.0	148.0	NaN
Titanic	1997.0	195.0	NaN

	year	duration	genre
The Pianist	NaN	NaN	drama
Gladiator	NaN	NaN	action
The Godfather	NaN	NaN	crime
Inception	NaN	NaN	action
Titanic	NaN	NaN	romance

In [39]:

```
df_conc = pd.concat([df, df_new], axis = 1)
df_conc
```

Out[39]:

	year	duration	genre
The Pianist	2002	150	drama
Gladiator	2000	155	action
The Godfather	1972	177	crime
Inception	2010	148	action
Titanic	1997	195	romance

6.2 Merge two DataFrames

We use the `merge()` function to combine data objects based on one or more keys in a similar way to a relational database. More specifically, `merge()` is most useful when we want to combine rows that share data.

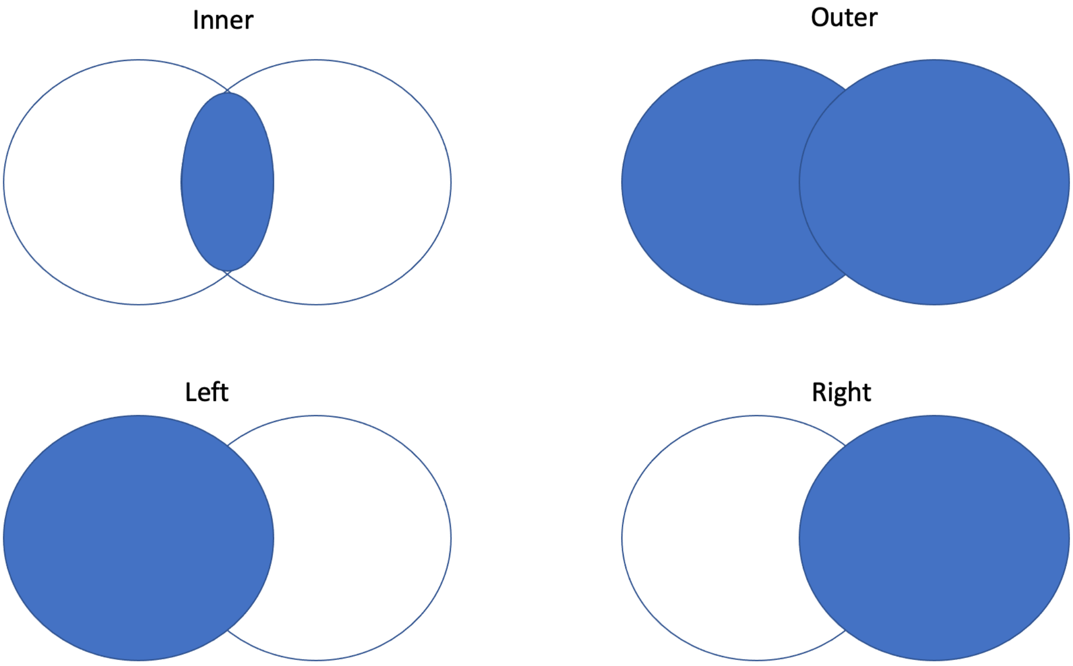
We can achieve both many-to-one and many-to-many joins with `merge()` . In a many-to-one join, one of your DataFrames will have many rows in the merge column that repeat the same values, while the merge column in the other DataFrame will not have repeat values.

In a many-to-many join, both of the merge columns have repeat values. These merges are more complex. This means that, after the merge, we have every combination of rows that share the same value in the key column.

When we use `merge()` , we provide two required arguments: the left DataFrame and the right DataFrame.

After that, we can provide a number of optional arguments to define how the datasets are merged, such as `how` that defines the kind of merge.

![howMerge.png]



Let's see some examples:

The simplest type of merge is the one-to-one join. Let's continue with our examples from above with some modifications. Let's create the `df` as:

In [40]:

```
df = pd.DataFrame({'title':['The Pianist', 'Gladiator', 'The Godfather',
                            'Inception', 'Titanic'],
                   'year':[2002, 2000, 1972, 2010, 1997],
                   'duration':[150, 155, 177, 148, 195]})
print(df)
```

```
           title  year  duration
0    The Pianist  2002       150
1      Gladiator  2000       155
2  The Godfather  1972       177
3      Inception  2010       148
4        Titanic  1997       195
```

And now we know that Gladiator has a score of 8.9 and Inception of 8.5. We first create the `df_new` DataFrame

In [41]:

```
df_new = pd.DataFrame({'title':['Gladiator', 'Inception'],
                       'score':[8.9,8.5]})

print(df_new)
```

```
       title  score
0  Gladiator    8.9
1  Inception    8.5
```

If we try to concatenate the two DataFrames, we will have the following result

In [42]:

```
df_concat = pd.concat([df, df_new])
df_concat
```

Out[42]:

	title	year	duration	score

	title	year	duration	score
0	The Pianist	2002.0	150.0	NaN
1	Gladiator	2000.0	155.0	NaN
2	The Godfather	1972.0	177.0	NaN
3	Inception	2010.0	148.0	NaN
4	Titanic	1997.0	195.0	NaN
0	Gladiator	NaN	NaN	8.9
1	Inception	NaN	NaN	8.5

So the `concat()` method ignores the title and the fact that there are already data for Gladiator and Inception

We have to use `merge()` in this case:

In [43]:

```
df_merge = pd.merge(df, df_new)
print(df_merge)
print()
```

```
       title  year  duration  score
0  Gladiator  2000       155    8.9
1  Inception  2010       148    8.5
```

The `pd.merge()` function recognizes that each DataFrame has a `title` column, and automatically joins using this column as a key. The result of the merge is a new DataFrame that combines the information from the two input DataFrames.

The order of entries in each column is not necessarily maintained: in this case, the order of the title column is different between `df` and `df_new` .

Many-to-one joins

Many-to-one joins are joins in which one of the two key columns contains duplicate entries. For the many-to-one case, the resulting DataFrame preserves those duplicate entries. Consider the following example of a many-to-one join:

In [44]:

```
df_movies = pd.DataFrame({'title': ['Gladiator', 'Amelie','Parasite'],
                          'duration': [155,123,132]})
df_genre = pd.DataFrame({'title': ['Gladiator', 'Gladiator', 'Amelie',
                                  'Titanic'],
                        'genre': ['action', 'drama', 'romance', 'romance']})
print(df_movies)
print()

print(df_genre)
print()

print(pd.merge(df_movies,df_genre))
```

```
       title  duration
0  Gladiator       155
1     Amelie       123
2   Parasite       132

       title   genre
0  Gladiator  action
```

```
1  Gladiator    drama
2     Amelie  romance
3    Titanic  romance

       title  duration    genre
0  Gladiator       155   action
1  Gladiator       155    drama
2     Amelie       123  romance
```

We have many-to-many merge if the key column in both the left and right array contains duplicates.

In the following example, we have a DataFrame showing one or more countries related to the movie and a DataFrame showing the genres of the movies. By performing a many-to-many join, we associate movies with the genres and countries:

In [45]:

```
df_country = pd.DataFrame({'title': ['Gladiator', 'Gladiator', 'Amelie',
                                     'Amelie', 'Parasite'],
                           'country': ['UK', 'USA', 'France', 'Germany',
                                      'South Korea']})
df_genre = pd.DataFrame({'title': ['Gladiator', 'Gladiator', 'Amelie',
                                   'Titanic'],
                         'genre': ['action', 'drama', 'romance',
                                  'romance']})
print(df_country)
print()

print(df_genre)
print()

print(pd.merge(df_genre,df_country))
```

```
       title      country
0  Gladiator           UK
1  Gladiator          USA
2     Amelie       France
3     Amelie      Germany
4   Parasite  South Korea

       title    genre
0  Gladiator   action
1  Gladiator    drama
2     Amelie  romance
3    Titanic  romance

       title    genre  country
0  Gladiator   action       UK
1  Gladiator   action      USA
2  Gladiator    drama       UK
3  Gladiator    drama      USA
4     Amelie  romance   France
5     Amelie  romance  Germany
```

The how arguement

The `how` argument defines the type of merge to be performed. This argument is by default to `inner`. Let's say we have the `df_country` and `df_genre` from before

Let's see the following example:

In [46]:

```
print(df_country)
print()
print(df_genre)
print()
```

```
       title      country
0  Gladiator           UK
1  Gladiator          USA
2     Amelie       France
3     Amelie      Germany
4   Parasite  South Korea

       title    genre
0  Gladiator   action
1  Gladiator    drama
2     Amelie  romance
3    Titanic  romance
```

Let's compare the inner and outer intersections.

In [47]:

```
print(pd.merge(df_country, df_genre, how='inner'))
print()

print(pd.merge(df_country, df_genre, how='outer'))
print()
```

```
       title  country    genre
0  Gladiator       UK   action
1  Gladiator       UK    drama
2  Gladiator      USA   action
3  Gladiator      USA    drama
4     Amelie   France  romance
5     Amelie  Germany  romance

       title      country    genre
0  Gladiator           UK   action
1  Gladiator           UK    drama
2  Gladiator          USA   action
3  Gladiator          USA    drama
4     Amelie       France  romance
5     Amelie      Germany  romance
6   Parasite  South Korea      NaN
7    Titanic          NaN  romance
```

Summary

In this unit, we covered basic functions of Pandas including how to create Pandas Series and DataFrames, and how to access the data.

In addition, we covered the main joining functions of Pandas, namely `concat()` and `merge()` .

Although these functions operate quite similar to each other, there are some fundamental differences among them.

Pandas `concat()` can be used for joining multiple DataFrames through both columns or rows. It is considered to be the most efficient method of joining DataFrames.

`Merge()` function performs joins similar to SQL. With the help of `merge()` we can merge values using a common column found in two DataFrames.

Exercises

1. Read the IMDB_movies.csv file and print all the basic information for the data, names of colums, shape and the first 5 rows

In [48]:

```
movies = pd.read_csv("../data/IMDB_movies.csv")

print(movies.columns)
print(movies.info())
print(movies.index)

print(movies.head())
```

```
Index(['imdb_title_id', 'title', 'original_title', 'year', 'date_published',
       'genre', 'duration', 'country', 'language', 'director', 'writer',
       'production_company', 'actors', 'description', 'avg_vote', 'votes',
       'budget', 'usa_gross_income', 'worlwide_gross_income', 'metascore',
       'reviews_from_users', 'reviews_from_critics'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 22 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   imdb_title_id          10000 non-null  object 
 1   title                  10000 non-null  object 
 2   original_title         10000 non-null  object 
 3   year                   10000 non-null  int64  
 4   date_published         10000 non-null  object 
 5   genre                  10000 non-null  object 
 6   duration               10000 non-null  int64  
 7   country                10000 non-null  object 
 8   language               9803 non-null   object 
 9   director               9999 non-null   object 
 10  writer                 9966 non-null   object 
 11  production_company     9971 non-null   object 
 12  actors                 9999 non-null   object 
 13  description            9921 non-null   object 
 14  avg_vote               10000 non-null  float64
 15  votes                  10000 non-null  int64  
 16  budget                 1880 non-null   object 
 17  usa_gross_income       451 non-null    object 
 18  worlwide_gross_income  392 non-null    object 
 19  metascore              227 non-null    float64
 20  reviews_from_users     9821 non-null   float64
 21  reviews_from_critics   9236 non-null   float64
dtypes: float64(4), int64(3), object(15)
memory usage: 1.7+ MB
None
RangeIndex(start=0, stop=10000, step=1)
  imdb_title_id                        title               original_title  \
0     tt0000009                   Miss Jerry                   Miss Jerry   
1     tt0000574  The Story of the Kelly Gang  The Story of the Kelly Gang   
2     tt0001892               Den sorte drøm               Den sorte drøm   
3     tt0002101                    Cleopatra                    Cleopatra   
4     tt0002130                    L'Inferno                    L'Inferno   

   year date_published                      genre  duration          country  
\
0  1894     1894-10-09                    Romance        45              USA   
1  1906     1906-12-26    Biography, Crime, Drama        70        Australia   
2  1911     1911-08-19                      Drama        53  Germany, Denmark   
3  1912     1912-11-13             Drama, History       100              USA   
4  1911     1911-03-06  Adventure, Drama, Fantasy        68            Italy   
```

```
  language                             director  ...  \
0     None                      Alexander Black  ...
1     None                         Charles Tait  ...
2      NaN                            Urban Gad  ...
3  English                   Charles L. Gaskill  ...
4  Italian  Francesco Bertolini, Adolfo Padovan  ...

                                              actors  \
0  Blanche Bayliss, William Courtenay, Chauncey D...
1  Elizabeth Tait, John Tait, Norman Campbell, Be...
2  Asta Nielsen, Valdemar Psilander, Gunnar Helse...
3  Helen Gardner, Pearl Sindelar, Miss Fielding, ...
4  Salvatore Papa, Arturo Pirovano, Giuseppe de L...

                                         description avg_vote votes   budget
\
0  The adventures of a female reporter in the 1890s.      5.9   154      NaN
1  True story of notorious Australian outlaw Ned ...      6.1   589   $ 2250
2  Two men of high rank are both wooing the beaut...      5.8   188      NaN
3  The fabled queen of Egypt's affair with Roman ...      5.2   446  $ 45000
4  Loosely adapted from Dante's Divine Comedy and...      7.0  2237      NaN

   usa_gross_income worlwide_gross_income metascore reviews_from_users  \
0               NaN                   NaN       NaN                1.0
1               NaN                   NaN       NaN                7.0
2               NaN                   NaN       NaN                5.0
3               NaN                   NaN       NaN               25.0
4               NaN                   NaN       NaN               31.0

   reviews_from_critics
0                   2.0
1                   7.0
2                   2.0
3                   3.0
4                  14.0

[5 rows x 22 columns]
```

2. Assume you have the following dictionary with some passengers of a flight. Create a DataFrame from the dictionary and assign pass_id as index. Print the basic information on the DataFrame and the index and the first 7 rows

In [49]:

```
passengers = {"age":[23, 25, 78, 12, 56, 33, 67, 78, 34, 64], "priority":['y'
pass_id=[101,102,103,104,105,106,107,108,109,110]
```

In [50]:

```
df = pd.DataFrame(passengers, index=pass_id)
```

In [51]:

```
print(df.columns)
print(df.info())
print(df.index)

print(df.head(7))
```

```
Index(['age', 'priority'], dtype='object')
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10 entries, 101 to 110
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       10 non-null     int64
 1   priority  10 non-null     object
dtypes: int64(1), object(1)
memory usage: 240.0+ bytes
None
Int64Index([101, 102, 103, 104, 105, 106, 107, 108, 109, 110], dtype='int64')
     age priority
101   23        y
102   25        y
103   78        n
104   12        n
105   56        n
106   33        y
107   67        y
```

3. Select the column of age

In [52]:

```
df.age
```

Out[52]:

```
101    23
102    25
103    78
104    12
105    56
106    33
107    67
108    78
109    34
110    64
Name: age, dtype: int64
```

4. Select the rows with id 105, 106 and 109

In [53]:

```
df.loc[[105, 106, 109]]
```

Out[53]:

	age	priority
105	56	n
106	33	y
109	34	n

5. Get the rows of passengers that are older than 40

In [54]:

```
df.loc[df['age'] > 40]
```

Out[54]:

	age	priority
103	78	n
105	56	n
107	67	y
108	78	n
110	64	y

6. Assume you get additional passengers from a different source (df2). Concatenate the two DataFrames and name the new DataFrame

df3

In [55]:
```
passengers2 = {"age":[54, 65, 12],
                "priority":['y','y','n']}
pass_id=[201, 202, 203]
```

In [56]:
```
df2 = pd.DataFrame(passengers2, index=pass_id)
```

In [57]:
```
df3 = pd.concat([df, df2])
df3.tail()
```

Out[57]:

	age	priority
109	34	n
110	64	y
201	54	y
202	65	y
203	12	n

7. Assume that for some passengers we have also some data for the flight they booked. Merge the two DataFrames (df3 and df_flights) including all the rows (outer) (assign the result to a new DataFrame df_merge).

In [58]:
```
pass_id = [101, 102, 202, 205, 210, 211]
flights = ['KL211','HV543', 'FR3090', 'KL4345','KL4345', 'FR3090']
df_flights = pd.DataFrame({'flight': flights}, index=pass_id)
```

In [59]:
```
print(df3)
print(df_flights)
df_merge = pd.merge(df3,df_flights, right_index=True,
                    left_index=True, how='outer')
print(df_merge)
```

```
     age priority
101   23        y
102   25        y
103   78        n
104   12        n
105   56        n
106   33        y
107   67        y
108   78        n
109   34        n
110   64        y
201   54        y
202   65        y
203   12        n
     flight
101   KL211
102   HV543
202  FR3090
205  KL4345
210  KL4345
211  FR3090
```

```
      age priority  flight
101  23.0        y   KL211
102  25.0        y   HV543
103  78.0        n     NaN
104  12.0        n     NaN
105  56.0        n     NaN
106  33.0        y     NaN
107  67.0        y     NaN
108  78.0        n     NaN
109  34.0        n     NaN
110  64.0        y     NaN
201  54.0        y     NaN
202  65.0        y  FR3090
203  12.0        n     NaN
205   NaN      NaN  KL4345
210   NaN      NaN  KL4345
211   NaN      NaN  FR3090
```

8. Now you have information on flights delayed. Merge the two DataFrames (delayed with df_merge). Make sure you keep the passengers id

In [60]:

```
delayed = pd.DataFrame({'flight': ['KL211','HV543', 'FR3990',
                                   'KL4345','KL4335', 'FR3090'],
                        'delay': ['y', 'n', 'y', 'n', 'y', 'y']})

df_merge_delayed = pd.merge(df_merge.reset_index(), delayed, how = 'left').se

df_merge_delayed
```

Out[60]:

	age	priority	flight	delay
index				
101	23.0	y	KL211	y
102	25.0	y	HV543	n
103	78.0	n	NaN	NaN
104	12.0	n	NaN	NaN
105	56.0	n	NaN	NaN
106	33.0	y	NaN	NaN
107	67.0	y	NaN	NaN
108	78.0	n	NaN	NaN
109	34.0	n	NaN	NaN
110	64.0	y	NaN	NaN
201	54.0	y	NaN	NaN
202	65.0	y	FR3090	y
203	12.0	n	NaN	NaN
205	NaN	NaN	KL4345	n
210	NaN	NaN	KL4345	n
211	NaN	NaN	FR3090	y

9. Set the priority to 'n' for the passenger with id 205 and 210

In [61]:

```
df_merge_delayed.loc[[205 ,210], ['priority']] = 'n'
```

In [62]:

```
df_merge_delayed.loc[[205, 210]]
```

Out[62]:

```
        age   priority   flight   delay
index
  205   NaN          n   KL4345       n
  210   NaN          n   KL4345       n
```

In [ ]: