Module IV

Pandas

- Pandas is a Python library for **data analysis.**
- Started by Wes McKinney in 2008 out of a need for a powerful and flexible quantitative analysis tool,
- Pandas is a high-level **data manipulation tool**
- It is built on the **Numpy package** and its key **data structure** is called the **DataFrame**.
- **DataFrames** allow you to store and manipulate tabular data in rows of observations and columns of variables.
- pandas has grown into one of the most popular Python libraries.
- It has an extremely active community of contributors.

There are several ways to create a DataFrame.
One way way is to use a dictionary.
For example:

```
dict = {"country": ["Brazil", "Russia", "India", "China", "South Africa"],
       "capital": ["Brasilia", "Moscow", "New Dehli", "Beijing", "Pretoria"],
       "area": [8.516, 17.10, 3.286, 9.597, 1.221],
       "population": [200.4, 143.5, 1252, 1357, 52.98] }

import pandas as pd
brics = pd.DataFrame(dict)
print(brics)
```

11. Create a very first Pandas dataframe (fromCSV)

Read Dataset:

```
Import pandas as pd
pd.read_csv("titanic.csv")
```

Storing dataset in a Variable:

```
Import pandas as pd
titanic = pd.read_csv("titanic.csv")
titanic
```

12. Pandas display option and the methods head() & tail()

```
titanic
or
print(titanic)
<Output : first and last 5 rows>
```

To see the max & min row setting:

```
pd.options.display.max_rows

<output : 60>

pd.options.display.min_rows

<output : 10>
```

To change the min row setting:

```
pd.options.display.min_rows = 20

titanic

<output : display 20 rows if the number of rows are greater than max rows
count>
```

To change the max row setting:

```
pd.options.display.min_rows = 891
titanic

<output : display all 891 rows>
```

Method head() & tail()

```
titanic.head()
<output : display first five rows of the dataset>

To see first 10 rows:

titanic.head(10)
<output : display first 10 rows of the dataset>
```

titanic.tail()
<output : display last five rows of the dataset>

To see last 10 rows:

titanic.tail(10)
<output : display last 10 rows of the dataset>

13. First Data Inspection

Info Method:

titanic.info()

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   survived  891 non-null    int64
 1   pclass    891 non-null    int64
 2   sex       891 non-null    object
 3   age       714 non-null    float64
 4   sibsp     891 non-null    int64
 5   parch     891 non-null    int64
 6   fare      891 non-null    float64
 7   embarked  889 non-null    object
 8   deck      203 non-null    object
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

Describe Method:

titanic.describe()

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
titanic.describe(include = "O")
```

	sex	embarked	deck
count	891	889	203
unique	2	3	7
top	male	S	C
freq	577	644	59

14. Built in Functions & Data Frame Attributes and Methods

Data Frame & Python Built-in Function

```
In [4]: type(titanic)
Out[4]: pandas.core.frame.DataFrame

In [5]: len(titanic)
Out[5]: 891

In [6]: round(titanic, 0)
```

Out[6]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.0	S	NaN
1	1	1	female	38.0	1	0	71.0	C	C
2	1	3	female	26.0	0	0	8.0	S	NaN
3	1	1	female	35.0	1	0	53.0	S	C
4	0	3	male	35.0	0	0	8.0	S	NaN
...	...	...	...	...	...	...	...	...	...

Data Attributes

```
In [9]: titanic.shape
Out[9]: (891, 9)

In [10]: titanic.size
Out[10]: 8019

In [11]: titanic.index
Out[11]: RangeIndex(start=0, stop=891, step=1)

In [12]: titanic.columns
Out[12]: Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
                'embarked', 'deck'],
               dtype='object')
```

DataFrame Methods

```
In [13]: titanic.head()
```

Out[13]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

```
In [14]: titanic.head(n = 2)
```

Out[14]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C

```
In [15]: titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   survived  891 non-null    int64
 1   pclass    891 non-null    int64
 2   sex       891 non-null    object
 3   age       714 non-null    float64
 4   sibsp     891 non-null    int64
 5   parch     891 non-null    int64
 6   fare      891 non-null    float64
 7   embarked  889 non-null    object
 8   deck      203 non-null    object
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

```
In [16]: titanic.min()

Out[16]: survived         0
         pclass           1
         sex         female
         age           0.42
         sibsp            0
         parch            0
         fare             0
         dtype: object
```

Method Chaining

```
In [17]: titanic.mean()

Out[17]: survived     0.383838
         pclass       2.308642
         age         29.699118
         sibsp        0.523008
         parch        0.381594
         fare        32.204208
         dtype: float64
```

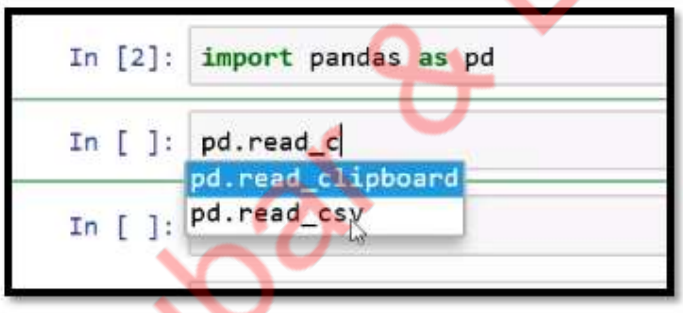
```
In [18]: titanic.mean().sort_values()

Out[18]: parch        0.381594
         survived     0.383838
         sibsp        0.523008
         pclass       2.308642
         age         29.699118
         fare        32.204208
         dtype: float64
```

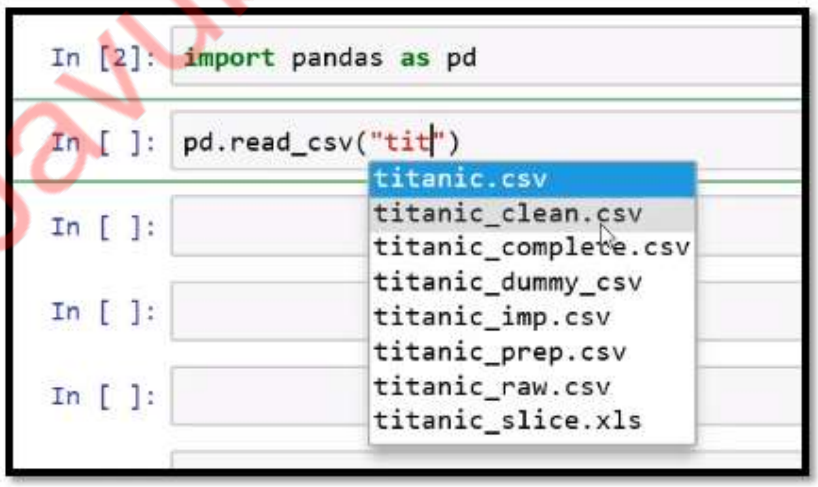
```
In [19]: titanic.mean().sort_values().head(2)

Out[19]: parch       0.381594
         survived    0.383838
         dtype: float64
```

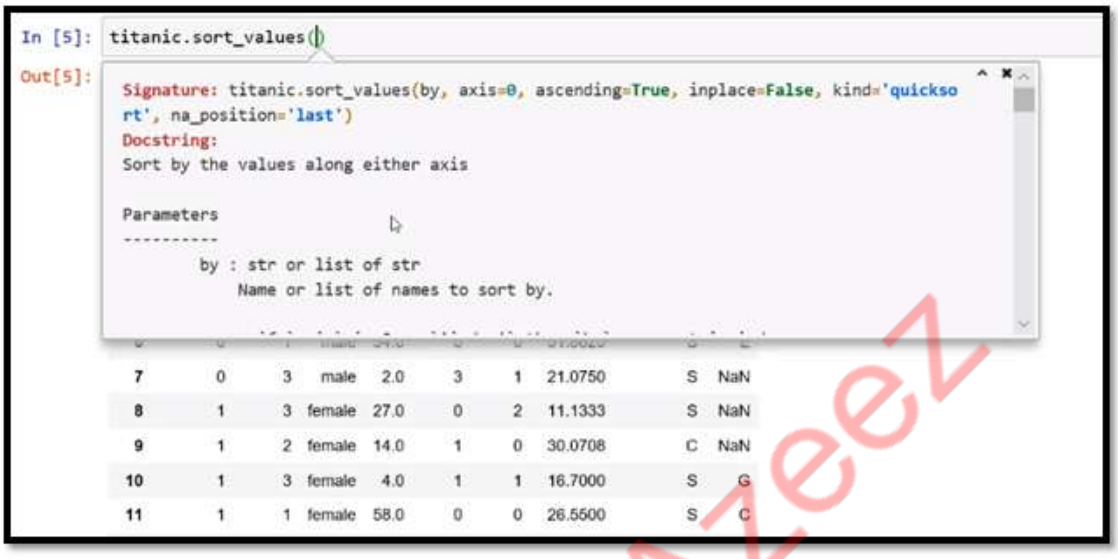
Resources :

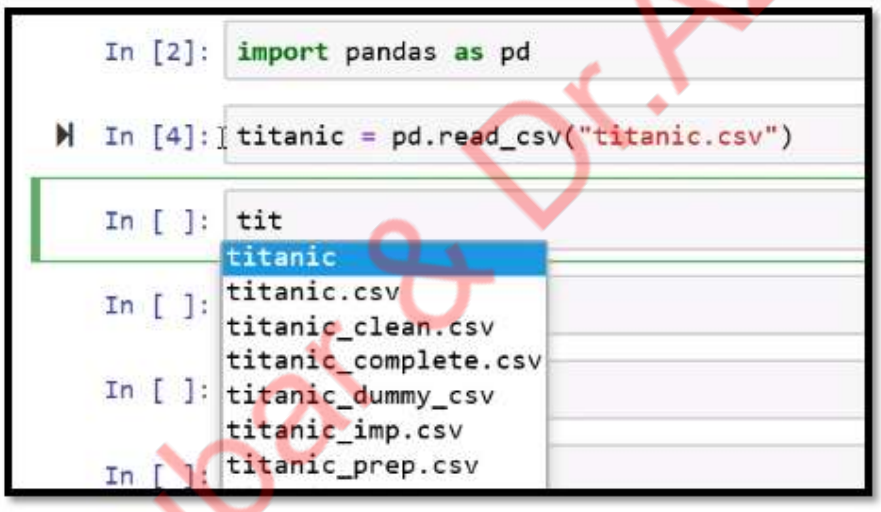
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html

15. Make it easy : TAB Completion and Tooltip





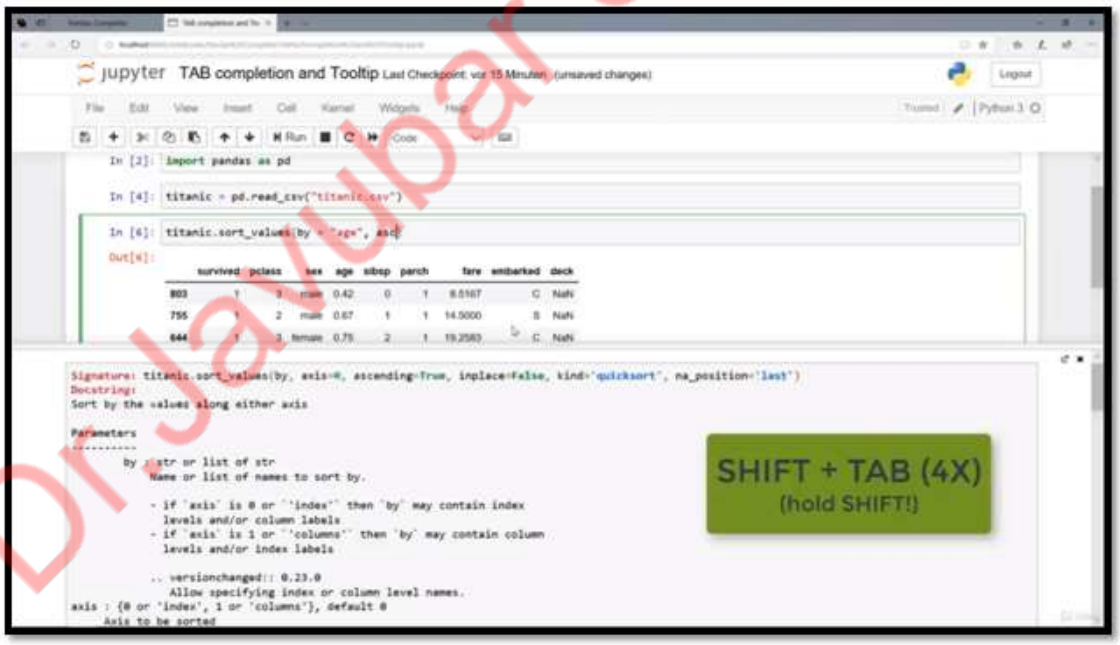


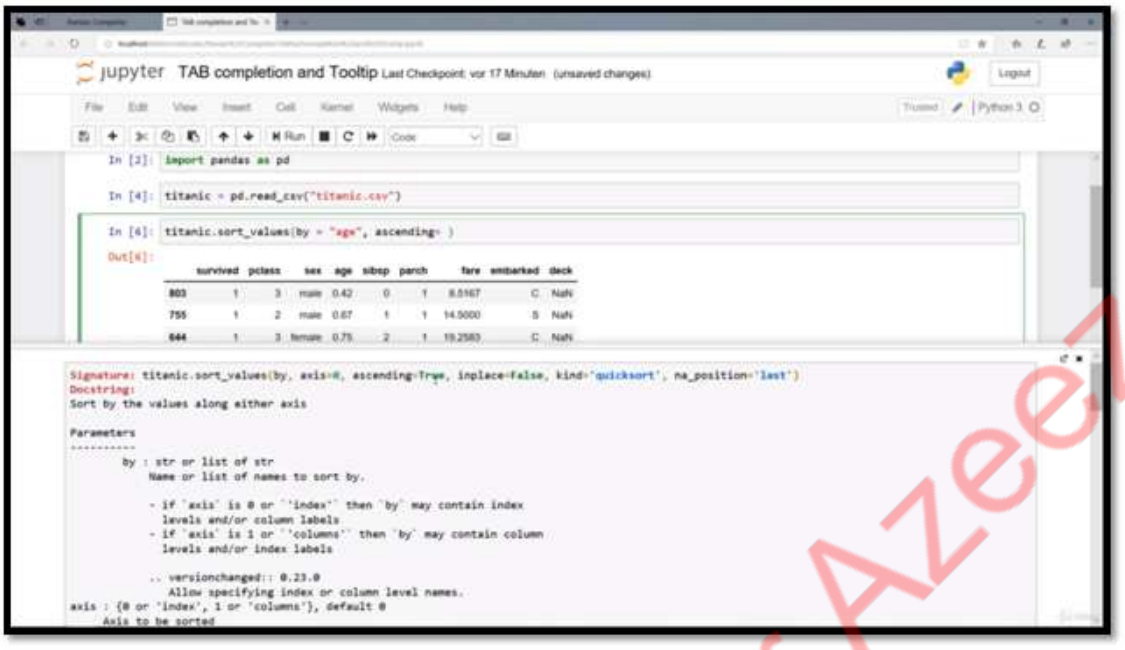


```
In [2]: import pandas as pd

In [4]: titanic = pd.read_csv("titanic.csv")

In [6]: titanic.sort_values(by = "age")
```

Out[6]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
803	1	3	male	0.42	0	1	8.5167	C	NaN
755	1	2	male	0.67	1	1	14.5000	S	NaN
644	1	3	female	0.75	2	1	19.2583	C	NaN
469	1	3	female	0.75	2	1	19.2583	C	NaN
78	1	2	male	0.83	0	2	29.0000	S	NaN
831	1	2	male	0.83	1	1	18.7500	S	NaN
305	1	1	male	0.92	1	2	151.5500	S	C
827	1	2	male	1.00	0	2	37.0042	C	NaN
381	1	3	female	1.00	0	2	15.7417	C	NaN
164	0	3	male	1.00	4	1	39.6875	S	NaN
183	1	2	male	1.00	2	1	39.0000	S	F





18. Selecting Column

Selecting all the columns of a dataset

```
In [2]: import pandas as pd
In [3]: titanic = pd.read_csv("titanic.csv")
In [4]: titanic
```

Out[4]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN
...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	NaN
887	1	1	female	19.0	0	0	30.0000	S	B
888	0	3	female	NaN	1	2	23.4500	S	NaN
889	1	1	male	26.0	0	0	30.0000	C	C
890	0	3	male	32.0	0	0	7.7500	Q	NaN

891 rows × 9 columns

Selection of a single column “age “

```
In [4]: titanic["age"]

Out[4]: 0      22.0
        1      38.0
        2      26.0
        3      35.0
        4      35.0
               ...
        886    27.0
        887    19.0
        888     NaN
        889    26.0
        890    32.0
        Name: age, Length: 891, dtype: float64
```

Finding the pandas series

```
In [5]: type(titanic["age"])

Out[5]: pandas.core.series.Series
```

Selecting two or more column, it require to use list

```
In [7]: titanic[["age", "sex"]]
```

Out[7]:

	age	sex
0	22.0	male
1	38.0	female
2	26.0	female
3	35.0	female
4	35.0	male
...	...	...
886	27.0	male
887	19.0	female
888	NaN	female
889	26.0	male
890	32.0	male

891 rows × 2 columns

Checking the type of the dataframe

```
In [8]: type(titanic[["age", "sex"]])
Out[8]: pandas.core.frame.DataFrame
```

We can also change the column sequence

```
In [7]: titanic[["sex", "age"]]
```

Out[7]:

	age	sex
0	22.0	male
1	38.0	female
2	26.0	female
3	35.0	female
4	35.0	male
...	...	...
886	27.0	male
887	19.0	female
888	NaN	female
889	26.0	male
890	32.0	male

19. SELECTING ONE COLUMN WITH DOT NOTATION

Selecting one Column with "dot notation"

```
In [5]: titanic.age.equals()
Out[5]: 0       22.0
        1       38.0
        2       26.0
        3       35.0
        4       35.0
                ...
        886     27.0
        887     19.0
        888      NaN
        889     26.0
        890     32.0
        Name: age, Length: 891, dtype: float64
```

We can check two panda series by using equal function

Selecting one Column with "dot notation"

```
In [6]: titanic.age.equals(titanic["age"])
Out[6]: True
```

20. Zero Based Indexing and Negative Indexing

Zero-based Indexing

column index positions

row index positions	Player	0 Nationality	1 Club	2 World_Champion	3 Height	4 Goals_2018
0	**Lionel Messi**	Argentina	FC Barcelona	False	1.70	45
1	**Cristiano Ronaldo**	Portugal	Juventus FC	False	1.87	44
2	**Neymar Junior**	Brasil	Paris SG	False	1.75	28
3	**Kylian Mbappe**	France	Paris SG	True	1.78	21
4	**Manuel Neuer**	Germany	FC Bayern	True	1.93	0

Negative Indexing

column index positions

row index positions		Player	-5 0 Nationality	-4 1 Club	-3 2 World_Champion	-2 3 Height	-1 4 Goals_2018
-5	0	**Lionel Messi**	Argentina	FC Barcelona	False	1.70	45
-4	1	**Cristiano Ronaldo**	Portugal	Juventus FC	False	1.87	44
-3	2	**Neymar Junior**	Brasil	Paris SG	False	1.75	28
-2	3	**Kylian Mbappe**	France	Paris SG	True	1.78	21
-1	4	**Manuel Neuer**	Germany	FC Bayern	True	1.93	0

21. Selecting rows with iloc (position based indexing)

Position-based Indexing and Slicing with iloc[]

```
In [1]: import pandas as pd

In [ ]: summer = pd.read_csv("summer.csv")

In [ ]: summer
```

```
In [14]: summer
```

Out[14]:

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver
...	...	...	...	...	...	...	...	...	...
31160	2012	London	Wrestling	Wrestling Freestyle	JANIKOWSKI, Damian	POL	Men	Wg 84 KG	Bronze
31161	2012	London	Wrestling	Wrestling Freestyle	REZAEI, Ghasem Gholamreza	IRI	Men	Wg 96 KG	Gold
31162	2012	London	Wrestling	Wrestling Freestyle	TOTROV, Rustam	RUS	Men	Wg 96 KG	Silver
31163	2012	London	Wrestling	Wrestling Freestyle	ALEKSANYAN, Artur	ARM	Men	Wg 96 KG	Bronze
31164	2012	London	Wrestling	Wrestling Freestyle	LIDBERG, Jimmy	SWE	Men	Wg 96 KG	Bronze

31165 rows × 9 columns

```
In [15]: summer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31165 entries, 0 to 31164
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Year        31165 non-null  int64
 1   City        31165 non-null  object
 2   Sport       31165 non-null  object
 3   Discipline  31165 non-null  object
 4   Athlete     31165 non-null  object
 5   Country     31161 non-null  object
 6   Gender      31165 non-null  object
 7   Event       31165 non-null  object
 8   Medal       31165 non-null  object
dtypes: int64(1), object(8)
memory usage: 2.1+ MB
```

Changing the Index to “Athlete” Column.

```
In [19]: summer = pd.read_csv("summer.csv", index_col = "Athlete")
```

```
In [20]: summer
```

Out[20]:

Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze
MALOKINIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold
CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver
...	...	...	...	...	...	...	...	...
JANIKOWSKI, Damian	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze
REZAEI, Ghasem Gholamreza	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold
TOTROV, Rustam	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver
ALEKSANYAN, Artur	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze

Selecting row using iloc

```
In [30]: summer.iloc[0]
```

```
Out[30]: Year                  1896
         City                Athens
         Sport              Aquatics
         Discipline         Swimming
         Country                 HUN
         Gender                  Men
         Event         100M Freestyle
         Medal                 Gold
         Name: HAJOS, Alfred, dtype: object
```

Selecting last row using negative indexing in iloc

```
In [31]: summer.iloc[-1]
```

```
Out[31]: Year                        2012
         City                      London
         Sport                  Wrestling
         Discipline    Wrestling Freestyle
         Country                       SWE
         Gender                        Men
         Event                   Wg 96 KG
         Medal                     Bronze
         Name: LIDBERG, Jimmy, dtype: object
```

Selecting first three rows using iloc by using list

```
In [33]: summer.iloc[[1,2,3]]
```

Out[33]:

Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze
MALOKINIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold

Selecting first three rows using slicing in iloc

```
In [36]: summer.iloc[1 : 4]
```

Out[36]:

Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze
MALOKINIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold

Using slicing operation, selecting first three rows

```
In [37]: summer.iloc[:3]
```

Out[37]:

Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze

Using slicing operation, selecting last three rows

```
In [38]: summer.iloc[-3:]
```

Out[38]:

Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal
TOTROV, Rustam	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver
ALEKSANYAN, Artur	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze
LIDBERG, Jimmy	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze

22. Slicing rows and columns with iloc(position-based indexing)

Using head & tail to select first and last set of rows

```
In [41]: summer.head(10)
```

Out[41]:

Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze
MALOKINIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold
CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver
CHOROPHAS, Efstathios	1896	Athens	Aquatics	Swimming	GRE	Men	1200M Freestyle	Bronze
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	1200M Freestyle	Gold
ANDREOU, Joannis	1896	Athens	Aquatics	Swimming	GRE	Men	1200M Freestyle	Silver
CHOROPHAS, Efstathios	1896	Athens	Aquatics	Swimming	GRE	Men	400M Freestyle	Bronze
NEUMANN, Paul	1896	Athens	Aquatics	Swimming	AUT	Men	400M Freestyle	Gold

To select a single value from a dataset using iloc.

Out[41]:

Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze
MALOKINIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold
CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver
CHOROPHAS, Efstathios	1896	Athens	Aquatics	Swimming	GRE	Men	1200M Freestyle	Bronze
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	1200M Freestyle	Gold
ANDREOU, Joannis	1896	Athens	Aquatics	Swimming	GRE	Men	1200M Freestyle	Silver
CHOROPHAS, Efstathios	1896	Athens	Aquatics	Swimming	GRE	Men	400M Freestyle	Bronze
NEUMANN, Paul	1896	Athens	Aquatics	Swimming	AUT	Men	400M Freestyle	Gold

```
In [42]: summer.iloc[0, 4]
Out[42]: 'HUN'
```

```
In [43]: summer.iloc[0,:3]
Out[43]: Year          1896
         City        Athens
         Sport     Aquatics
         Name: HAJOS, Alfred, dtype: object
```

```
In [44]: summer.iloc[0,[0,2,5,7]]
```

```
Out[44]: Year          1896
         Sport      Aquatics
         Gender          Men
         Medal          Gold
         Name: HAJOS, Alfred, dtype: object
```

```
In [45]: summer.iloc[34:39,[0,2,5,7]]
```

Out[45]:

Athlete	Year	Sport	Gender	Medal
GARRETT, Robert	1896	Athletics	Men	Silver
KELLNER, Gyula	1896	Athletics	Men	Bronze
LOUIS, Spyridon	1896	Athletics	Men	Gold
VASILAKOS, Kharilaos	1896	Athletics	Men	Silver
DAMASKOS, Evangelos	1896	Athletics	Men	Bronze

```
In [47]: summer.iloc[:, 4]
```

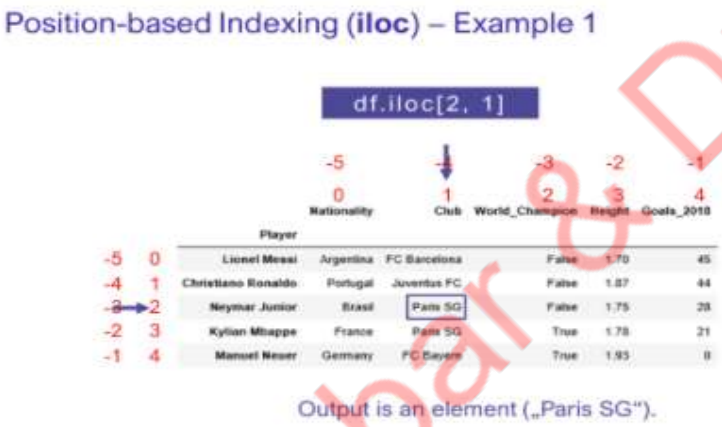
```
Out[47]: Athlete
         HAJOS, Alfred                 HUN
         HERSCHMANN, Otto              AUT
         DRIVAS, Dimitrios             GRE
         MALOKINIS, Ioannis            GRE
         CHASAPIS, Spiridon            GRE
                                       ...
         JANIKOWSKI, Damian            POL
         REZAEI, Ghasem Gholamreza     IRI
         TOTROV, Rustam                RUS
         ALEKSANYAN, Artur             ARM
         LIDBERG, Jimmy                SWE
         Name: Country, Length: 31165, dtype: object
```

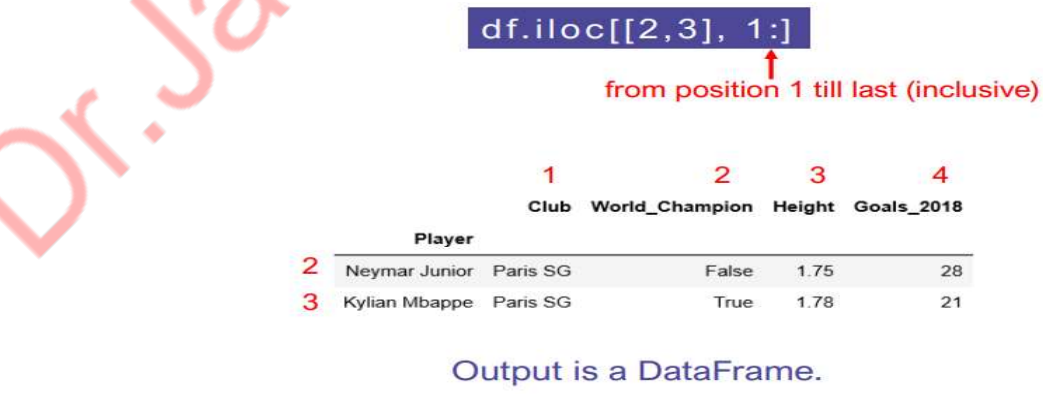
Compare two selecting row outputs by using equals function

```
In [48]: summer.iloc[:, 4].equals(summer.Country)
```

```
Out[48]: True
```

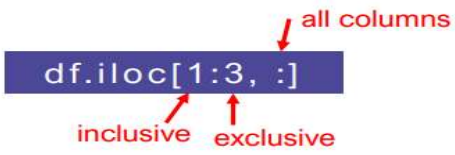
23. Position based Indexing Cheat Sheets





	Player	Club	World_Champion	Height	Goals_2018
2	Neymar Junior	Paris SG	False	1.75	28
3	Kylian Mbappe	Paris SG	True	1.78	21

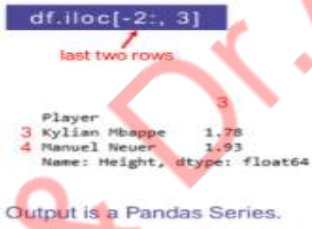
Position-based Indexing (**iloc**) – Example 3



	0	1	2	3	4
	Nationality	Club	World_Champion	Height	Goals_2018
Player					
1 Cristiano Ronaldo	Portugal	Juventus FC	False	1.87	44
2 Neymar Junior	Brasil	Paris SG	False	1.75	28

Output is a DataFrame.

Position-based Indexing (**iloc**) – Example 4



Output is a Pandas Series.

24. LABEL BASED INDEXING AND SLICING WITH LOC[]

Out[31]:

Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze
MALOKINIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold
CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver
...	...	...	...	...	...	...	...	...
JANIKOWSKI, Damian	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze
REZAEI, Ghasem Gholamreza	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold
TOTROV, Rustam	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver
ALEKSANYAN, Artur	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze
LIDBERG, Jimmy	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze

31165 rows × 8 columns

Selecting Rows with loc[]

```
In [32]: summer.iloc[2]

Out[32]: Year                                1896
         City                              Athens
         Sport                            Aquatics
         Discipline                       Swimming
         Country                               GRE
         Gender                                Men
         Event          100M Freestyle For Sailors
         Medal                             Bronze
         Name: DRIVAS, Dimitrios, dtype: object
```

```
In [ ]: summer.loc["DRIVAS, Dimitrios"]
```

```
In [34]: summer.loc["PHELPS, Michael"]
```

Out[34]:

Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal
PHELPS, Michael	2004	Athens	Aquatics	Swimming	USA	Men	100M Butterfly	Gold
PHELPS, Michael	2004	Athens	Aquatics	Swimming	USA	Men	200M Butterfly	Gold
PHELPS, Michael	2004	Athens	Aquatics	Swimming	USA	Men	200M Freestyle	Bronze
PHELPS, Michael	2004	Athens	Aquatics	Swimming	USA	Men	200M Individual Medley	Gold
PHELPS, Michael	2004	Athens	Aquatics	Swimming	USA	Men	400M Individual Medley	Gold
PHELPS, Michael	2004	Athens	Aquatics	Swimming	USA	Men	4X100M Freestyle Relay	Bronze
PHELPS, Michael	2004	Athens	Aquatics	Swimming	USA	Men	4X100M Medley Relay	Gold
PHELPS, Michael	2004	Athens	Aquatics	Swimming	USA	Men	4X200M Freestyle Relay	Gold
PHELPS, Michael	2008	Beijing	Aquatics	Swimming	USA	Men	100M Butterfly	Gold
PHELPS, Michael	2008	Beijing	Aquatics	Swimming	USA	Men	200M Butterfly	Gold
PHELPS, Michael	2008	Beijing	Aquatics	Swimming	USA	Men	200M Freestyle	Gold
PHELPS, Michael	2008	Beijing	Aquatics	Swimming	USA	Men	200M Individual Medley	Gold
PHELPS, Michael	2008	Beijing	Aquatics	Swimming	USA	Men	400M Individual Medley	Gold

e have written a note here.

26. Indexing and Slicing with reindex()

```
In [1]: import pandas as pd
```

```
In [3]: summer = pd.read_csv("summer.csv")
```

```
In [4]: summer
```

Out[4]:

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver
...	...	...	...	...	...	...	...	...	...
31160	2012	London	Wrestling	Wrestling Freestyle	JANIKOWSKI, Damian	POL	Men	Wg 84 KG	Bronze
31161	2012	London	Wrestling	Wrestling Freestyle	REZAEI, Ghasem Gholamreza	IRI	Men	Wg 96 KG	Gold
31162	2012	London	Wrestling	Wrestling Freestyle	TOTROV, Rustam	RUS	Men	Wg 96 KG	Silver
31163	2012	London	Wrestling	Wrestling Freestyle	ALEKSANYAN, Artur	ARM	Men	Wg 96 KG	Bronze
31164	2012	London	Wrestling	Wrestling Freestyle	LIDBERG, Jimmy	SWE	Men	Wg 96 KG	Bronze

31165 rows × 9 columns

```
In [6]: summer.loc[[0,5,30000],["Athlete","Medal"]]
```

Out[6]:

	Athlete	Medal
0	HAJOS, Alfred	Gold
5	CHOROPHAS, Efstathios	Bronze
30000	PAUTARAN, Maryna	Bronze

Reindex

The summer table have rows upto 31165

```
In [10]: summer.reindex (index = [0,5,30000,40000], columns = ["Athlete","Medal", "Age"])
```

Out[10]:

	Athlete	Medal	Age
0	HAJOS, Alfred	Gold	NaN
5	CHOROPHAS, Efstathios	Bronze	NaN
30000	PAUTARAN, Maryna	Bronze	NaN
40000	NaN	NaN	NaN

==========

Importing from CSV and first inspection

```
import pandas as pd
summer = pd.read_csv("summer.csv")
summer
summer.info()
summer.describe()
```

Select one column

```
summer.Medal
summer["Medal"]
```

Select multiple Columns

summer[“Year“,”Medal”]
summer.loc[:,[“Year“,”Medal”]]

Select positional rows

summer.iloc[10:21]

Select positional rows

summer.loc[“LEWIS, Carl”]

==========

Section 4 - Pandas Series and Index Objects

Welcome to the Section on Pandas Series and Pandas Index objects!

In this Section, you will learn how to work with and analyze single columns of a DataFrame (Pandas Series).

We will differentiate between numerical columns (integers or floats) and non-numerical columns (strings).

Analyzing Columns / Pandas Series

non-numerical column | numerical column

Player	Nationality	Club	World_Champion	Height	Goals_2018
Lionel Messi	Argentina	FC Barcelona	False	1.70	45
Christiano Ronaldo	Portugal	Juventus FC	False	1.87	44
Neymar Junior	Brasil	Paris SG	False	1.75	28
Kylian Mbappe	France	Paris SG	True	1.78	21
Manuel Neuer	Germany	FC Bayern	True	1.93	0

Next, you will learn how to work with and analyze Pandas Index objects and how to change single or multiple labels.

We will differentiate between (Row) Index and Column Index.

Analyzing and modifying Pandas Index objects

Player	Nationality	Club	World_Champion	Height	Goals_2018
Lionel Messi	Argentina	FC Barcelona	False	1.70	45
Christiano Ronaldo	Portugal	Juventus FC	False	1.87	44
Neymar Junior	Brasil	Paris SG	False	1.75	28
Kylian Mbappe	France	Paris SG	True	1.78	21
Manuel Neuer	Germany	FC Bayern	True	1.93	0

PANDAS SERIES

FIRST STEP WITH PANDAS SERIES

In [7]:
```
import pandas as pd
```

In [8]:
```
titanic = pd.read_csv("titanic.csv")
```

In [ ]:
```
titanic
```

In [ ]:
```
titanic.info()
```

In [ ]:
```
titanic["age"]
```

In [ ]:
```
type(titanic["age"])
```

In [ ]:
```
titanic["age"].equals (titanic.age)
```

In [13]:
```
age = titanic["age"]
```

In [ ]:
```
age.head()
```

In [ ]:
```
age.tail()
```

In [ ]:
```
age.dtype
```

In [ ]:
```
age.shape
```

In [ ]:
```
len(age)
```

In [ ]:
```
age.index
```

In [ ]:
```
age.info()
```

In [ ]:
```
age.to_frame().info()
```

Analyzing Numerical Series with unique(), nunique() and value_counts()

```
In [ ]: import pandas as pd
```

```
In [ ]: titanic = pd.read_csv("titanic.csv")
```

```
In [ ]: titanic.csv
```

```
In [ ]: age = titanic["age"]
```

```
In [ ]: age
```

```
In [ ]: age.describe()
```

```
In [ ]: age.count()
```

```
In [ ]: age.size
```

```
In [ ]: len(age)
```

```
In [ ]: age.sum()
```

```
In [ ]: sum(age)
```

```
In [ ]: age.mean()
```

```
In [ ]: age.median()
```

```
In [ ]: age.std()
```

```
In [ ]: age.min()
```

```
In [ ]: age.max()
```

```
In [ ]: age.unique()
```

```
In [ ]: len(age.unique())
```

```
In [ ]: age.nunique()
```

```
In [ ]: age.value_counts()
```

Analyzing Non-Numerical Series with unique(), nunique() and value_counts()

```
In [ ]: import pandas as pd
```

```
In [ ]: summer = pd.read_csv("summer.csv")
```

```
In [ ]: summer.head()
```

```
In [ ]: summer.tail()
```

```
In [ ]: summer.info()
```

```
In [ ]: athlete = summer("athlete")
```

```
In [ ]: athlete.head()
```

```
In [ ]: athlete.tail(5)
```

```
In [ ]: tyep(athlete)
```

```
In [ ]: athlete.dtype
```

```
In [ ]: athlete.shape
```

```
In [ ]: athlete.describe()
```

```
In [ ]: athlete.size
```

```
In [ ]: athlete.count()
```

```
In [ ]: athlete.min()
```

```
In [ ]: athlete.unique()
```

```
In [ ]: len(athlete.unique())
```

```
In [ ]: athlete.nunique()
```

```
In [ ]: athlete.value_counts()
```

PANDAS SERIES

Creating Pandas Series Part-1

In [1]:
```
import pandas as pd
```

In [2]:
```
summer = pd.read_csv("summer.csv")
```

In [3]:
```
summer.head()
```

Out[3]:

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver

In [6]:
```
summer["Athlete"]
```

Out[6]:
```
0                    HAJOS, Alfred
1                  HERSCHMANN, Otto
2                 DRIVAS, Dimitrios
3                MALOKINIS, Ioannis
4                CHASAPIS, Spiridon
                     ...
31160            JANIKOWSKI, Damian
31161    REZAEI, Ghasem Gholamreza
31162                TOTROV, Rustam
31163             ALEKSANYAN, Artur
31164                LIDBERG, Jimmy
Name: Athlete, Length: 31165, dtype: object
```

In [7]:
```
summer.Athlete
```

Out[7]:
```
0                    HAJOS, Alfred
1                  HERSCHMANN, Otto
2                 DRIVAS, Dimitrios
3                MALOKINIS, Ioannis
4                CHASAPIS, Spiridon
                     ...
31160            JANIKOWSKI, Damian
31161    REZAEI, Ghasem Gholamreza
31162                TOTROV, Rustam
31163             ALEKSANYAN, Artur
31164                LIDBERG, Jimmy
Name: Athlete, Length: 31165, dtype: object
```

In [8]:
```
summer.iloc[0]
```

Out[8]:
```
Year                    1896
City                  Athens
Sport               Aquatics
Discipline          Swimming
Athlete        HAJOS, Alfred
Country                  HUN
Gender                   Men
Event         100M Freestyle
Medal                   Gold
Name: 0, dtype: object
```

Importing from CSV

In [9]:
```
pd.read_csv("summer.csv", usecols=["Athlete"], squeeze = True)
```

Out[9]:
```
0                    HAJOS, Alfred
1                  HERSCHMANN, Otto
2                 DRIVAS, Dimitrios
3               MALOKINIS, Ioannis
4               CHASAPIS, Spiridon
                      ...
31160           JANIKOWSKI, Damian
31161    REZAEI, Ghasem Gholamreza
31162               TOTROV, Rustam
31163            ALEKSANYAN, Artur
31164               LIDBERG, Jimmy
Name: Athlete, Length: 31165, dtype: object
```

Creating Scratch from pd.Series()

In [12]:
```
pd.Series([10,25,6,36,2])
```

Out[12]:
```
0    10
1    25
2     6
3    36
4     2
dtype: int64
```

In [14]:
```
pd.Series([10,25,6,36,2], index = ["Mon","Tue", "Wed", "Thu","Fri"])
```

Out[14]:
```
Mon    10
Tue    25
Wed     6
Thu    36
Fri     2
dtype: int64
```

In [15]:
```
pd.Series([10,25,6,36,2], index = ["Mon","Tue", "Wed", "Thu","Fri"], name = "sales")
```

Out[15]:
```
Mon    10
Tue    25
Wed     6
Thu    36
Fri     2
Name: sales, dtype: int64
```

Creating Pandas Series Part-2

from Numpy Array

```
In [19]: import pandas as pd
         import numpy as np
```

```
In [22]: sales = np.array([10,25,6,36,2])
         sales
```

```
Out[22]: array([10, 25,  6, 36,  2])
```

```
In [23]: pd.Series(sales)
```

```
Out[23]: 0    10
         1    25
         2     6
         3    36
         4     2
         dtype: int32
```

from List

```
In [24]: sales = [10,25,6,36,2]
```

```
In [25]: pd.Series(sales)
```

```
Out[25]: 0    10
         1    25
         2     6
         3    36
         4     2
         dtype: int64
```

from Dictionary

```
In [27]: dic = {"Mon":10,"Tue":25,"Wed":6,"Thu":36,"Fri":2}
```

```
In [29]: sales = pd.Series(dic)
```

```
In [30]: sales
```

```
Out[30]: Mon    10
         Tue    25
         Wed     6
         Thu    36
         Fri     2
         dtype: int64
```

In [31]:
```
pd.Series(dic, index = ["Fri", "Sat", "Sun", "Mon", "Tue", "Wed"])
```

Out[31]:
```
Fri     2.0
Sat     NaN
Sun     NaN
Mon    10.0
Tue    25.0
Wed     6.0
dtype: float64
```

In [32]:
```
pd.Series(dic, index = [1,2,3,4,5])
```

Out[32]:
```
1   NaN
2   NaN
3   NaN
4   NaN
5   NaN
dtype: float64
```

Indexing and Slicing Pandas Series

In [ ]:
```
import pandas as pd
```

In [ ]:
```
titanic = pd.read_csv("titanic.csv")
```

In [ ]:
```
titanic.head()
```

In [ ]:
```
titanic.tail()
```

In [ ]:
```
age = titanic.age
```

In [ ]:
```
age.head()
```

In [ ]:
```
age.tail()
```

In [ ]:
```
age.index
```

In [ ]:
```
age[0]
```

In [ ]:
```
age[2]
```

In [ ]:
```
age.iloc[-1]
```

In [ ]:
```
age[890]
```

In [ ]:
```
age.iloc[:3]
```

In [ ]:
```
age.loc[:3]
```

Sorting and Introduction to the inplace-parameter

In [ ]:
```
import pandas as pd
```

In [ ]:
```
dic = {1:10, 3:25, 2:6, 4:36, 5:2, 6:0, 7:None}
dic
```

In [ ]:
```
sales = pd.Series(dic)
sales
```

In [ ]:
```
sales.sort_index()
```

In [ ]:
```
sales
```

In [ ]:
```
sales.sort_index(ascending = True, inplace = True)
```

In [ ]:
```
sales
```

In [ ]:
```
sales.sort_values()
```

In [ ]:
```
sales
```

nlargest(), nsmallest(), idxmax() and idxmin()

In [ ]:
```
import pandas as pd
```

In [ ]:
```
titanic = pd.read_csv("titanic.csv")
```

In [ ]:
```
titanic.head()
```

In [ ]:
```
age = titanic.age
```

In [ ]:
```
age
```

In [ ]:
```
age.head()
```

In [ ]:
```
age.sort_values(ascending=False).head(5)
```

In [ ]:
```
age.nlargest()
```

In [ ]:
```
age.nsmallest()
```

In [ ]:
```
age.idxmax()
```

In [ ]:
```
age.idxmin()
```


5.1 Section DataFrame Basics II

In this Section, you will learn how to

1. **filter DataFrames by one and by many conditions.**
2. **perform advanced filtering techniques**
3. **remove rows and columns from a DataFrame**
4. **add new columns to a DataFrame**

DATAFRAME BASICS II ¶

FILTERING DATAFRAMES WITH ONE CONDITION

```
In [ ]: import pandas as pd
In [ ]: titanic = pd.read_csv("titanic.csv")
In [ ]: titanic
In [ ]: titanic.loc[titanic.sex =="male"]
In [ ]: t1 = titanic.loc[titanic.sex =="male"]
In [ ]: t1
```

DATAFRAME BASICS II

FILTERING DATAFRAMES WITH MANY CONDITIONS (AND)

```
In [ ]: import pandas as pd
In [ ]: titanic = pd.read_csv("titanic.csv")
In [ ]: titanic
In [ ]: titanic.loc[titanic.sex =="male"]
In [ ]: mask1 = titanic.sex =="male"
In [ ]: mask2 = titanic.age > 14
In [ ]: mask1
In [ ]: mask2
In [ ]: (mask1 & mask2).head()
```

```
In [ ]: output = titanic.loc[mask1 & mask2]
```

```
In [ ]: output
```

```
In [ ]: output.info()
```

```
In [ ]: output.describe()
```

DATAFRAME BASICS II

FILTERING DATAFRAMES WITH MANY CONDITIONS (OR)

```
In [ ]: import pandas as pd
```

```
In [ ]: titanic = pd.read_csv("titanic.csv")
```

```
In [ ]: titanic
```

```
In [ ]: titanic.loc[titanic.sex =="male"]
```

```
In [ ]: mask1 = titanic.sex =="male"
```

```
In [ ]: mask2 = titanic.age > 14
```

```
In [ ]: mask1
```

```
In [ ]: mask2
```

```
In [ ]: (mask1 & mask2).head()
```

```
In [34]: output = titanic.loc[mask1 | mask2]
```

```
In [ ]: output
```

```
In [ ]: output.info()
```

```
In [ ]: output.describe()
```

DATAFRAME BASICS II

ADVANCED FILTERING WITH BETWEEN(), ISIN() AND ~

In [ ]:
```
import pandas as pd
```

In [ ]:
```
summer = pd.read_csv("summer.csv")
```

In [ ]:
```
summer
```

In [ ]:
```
og_1988 = summer.loc[summer.Year == 1988]
```

In [ ]:
```
og_1988.head()
```

In [ ]:
```
og_1988.tail()
```

In [ ]:
```
og_1988.info()
```

In [ ]:
```
og_since1992 = summer.loc[summer.Year >= 1992]
```

In [ ]:
```
og_since1992
```

In [ ]:
```
og_since1992.head()
```

In [ ]:
```
og_since1992.tail()
```

In [ ]:
```
summer.Year.between(1960,1969).head()
```

In [ ]:
```
og_60s = summer.loc[summer.Year.between(1960,1969, inclusive = True)]
```

In [ ]:
```
og_60s
```

In [ ]:
```
my_favoriate_games = [1972,1996]
```

In [ ]:
```
summer.Year.isin(my_favoriate_games).head()
```

In [ ]:
```
og_72_96 = summer.loc[summer.Year.isin(my_favoriate_games)]
```

```
In [ ]: og_72_96

In [ ]: og_not_72_96 = summer.loc[~summer.Year.isin(my_favoriate_games)]

In [ ]: og_not_72_96

In [ ]: og_not_72_96.Year.unique()

In [ ]: mask1 = summer.Year == 1960

In [ ]: mask2 = summer.Year == 1996

In [ ]: og_or_72_96 = summer.loc[mask1 | mask2]

In [ ]: og_or_72_96

In [ ]: test1 = summer.loc[summer.Year == 1960]

In [ ]: test1

In [ ]: summer

In [ ]: summer.Year.unique()
```

DATAFRAME BASICS II

ADVANCED FILTERING WITH any() AND all()

```
In [ ]: import pandas as pd

In [ ]: titanic = pd.read_csv("titanic.csv")

In [ ]: titanic.head()

In [ ]: titanic.tail()

In [ ]: titanic.sex == "male"

In [ ]: (titanic.sex == "male").any()

In [ ]: (titanic.sex == "male").all()

In [ ]: (titanic.age == 80.0).any()

In [ ]: pd.Series([-1,0.5,1,-0.1,0]).all()

In [ ]: titanic.fare.all()
```

DATAFRAME BASICS II

Removing Columns ¶

In [ ]:
```
import pandas as pd
```

In [ ]:
```
summer = pd.read_csv("summer.csv")
```

In [ ]:
```
summer.head()
```

In [ ]:
```
summer.drop(columns = "Sport")
```

In [ ]:
```
summer.head()
```

In [ ]:
```
summer.drop(columns = ["Sport", "Discipline"])
```

In [ ]:
```
summer.head()
```

In [ ]:
```
summer.drop(columns = ["Sport", "Discipline"], inplace = True)
```

In [ ]:
```
summer.head()
```

In [ ]:
```
summer.loc[:,["Year", "City", "Athlete", "Country"]]
```

DATAFRAME BASICS II

Removing Rows

In [ ]:
```
import pandas as pd
```

In [ ]:
```
summer = pd.read_csv("summer.csv", index_col = "Athlete")
```

In [ ]:
```
summer.head()
```

In [ ]:
```
summer.drop(index = "HAJOS, Alfred")
```

In [ ]:
```
summer.head()
```

In [ ]:
```
summer.drop(index = "HAJOS, Alfred", inplace = True)
```

In [ ]:
```
summer.head()
```

In [ ]:
```
summer.drop(labels = "DRIVAS, Dimitrios", axis = 0, inplace = True)
```

In [ ]:
```
summer.head()
```

In [ ]:
```
summer = pd.read_csv("summer.csv", index_col = "Athlete")
```

In [ ]:
```
summer.head()
```

In [ ]:
```
mask1 = summer.Year == 1996
mask2 = summer.Sport == "Aquatics"
```

In [ ]:
```
summer.loc[~(mask1 | mask2)]
```

DATAFRAME BASICS II

Removing Rows

```
In [ ]: import pandas as pd
```

```
In [ ]: titanic = pd.read_csv("titanic.csv")
```

```
In [ ]: titanic.head()
```

```
In [ ]: titanic["Zeros"] = 0
```

```
In [ ]: titanic.head()
```

```
In [ ]: titanic = pd.read_csv("titanic.csv")
```

```
In [ ]: titanic.head()
```

DATAFRAME BASICS II

Creating columns based on other columns

```
In [ ]: import pandas as pd
```

```
In [ ]: titanic = pd.read_csv("titanic.csv")
```

```
In [ ]: titanic.head()
```

```
In [ ]: 1912-titanic.age
```

```
In [ ]: titanic["YoB"] = 1912-titanic.age
```

```
In [ ]: titanic.head()
```

```
In [ ]: titanic.sibsp + titanic.parch
```

```
In [ ]: titanic["relatives"] = titanic.sibsp + titanic.parch
```

```
In [ ]: titanic.head()
```

```
In [ ]: titanic.drop(columns = ["sibsp", "parch"], inplace = True)
```

```
In [ ]: titanic.head()
```

```
In [ ]: inflation_factor = 10
```

```
In [ ]: titanic.fare*10
```

```
In [ ]: titanic.fare = titanic.fare*10
```

```
In [ ]: titanic.head()
```

DATAFRAME BASICS II

Adding columns with insert()

```
In [ ]: import pandas as pd
```

```
In [ ]: titanic = pd.read_csv("titanic.csv")
```

```
In [ ]: titanic.head()
```

```
In [ ]: titanic["Test"] = "Test"
```

```
In [ ]: titanic.head()
```

```
In [ ]: relatives = titanic.sibsp + titanic.parch
relatives.head()
```

```
In [ ]: titanic.insert(loc = 6, column = "relatives", value = relatives)
```

```
In [ ]: titanic.head()
```

DataFrame Basics II ¶

Exercise 5: Filtering DataFrames & Adding/Removing Rows and Columns

```
In [ ]: #run the cell!
import pandas as pd
```

```
In [ ]: #run the cell!
cars = pd.read_csv("cars.csv")
```

```
In [ ]: #run the cell!
cars
```

40. **Check** for all elements in the column **"origin"** whether they are **equal to "europe"**. **Save** the boolean Series in the variable **mask1! Fill in the gaps!**

```
In [ ]: mask1 = cars.origin == "europe"
mask1
```

41. **Check** for all elements in the column **"mpg"** whether they are **smaller than 20**. **Save** the boolean Series in the variable **mask2!**

In [ ]:
```
mask2 = cars.mpg < 20
```

42. **Filter** the cars DataFrame for **all cars from europe** (use mask1) and **save** the subset in the variable **europe! Fill in the gaps!**

In [ ]:
```
europe  = cars.loc[mask1].copy()
europe
```

Inspect! What is the name of the **second** car?

In [ ]:
```
# run the cell!
europe
```

The second car is a ... peugeot 504!

43. Get some **meta information** on the DataFrame europe. **How many** cars are from europe?

In [ ]:
```
europe.info()
```

... 70 cars are from europe!

44. **Filter** the DataFrame **cars** for all cars **from europe with low fuel efficiency** (mpg lower than 20). Use **mask1** and **mask2!**
Save the subset in the variable **europe_le! Fill in the gaps!**

In [ ]:
```
europe_le = cars.loc[mask1 & mask2].copy()
```

Inspect! What is the **name** of the **least efficient** car from europe?

In [ ]:
```
#run the cell!
europe_le
```

The least efficient european car is ... peugeot 604sl

45. **Filter** the DataFrame cars for all cars with an **mpg between 10 and 15** (both ends inclusive!).
Save the subset in the variable **mpg_10_15! Fill in the gaps!**

In [ ]:
```
mpg_10_15 = cars.loc[cars.mpg.between(10,15)].copy()
```

Inspect! The **first** car is...?

In [ ]:
```
# run the cell!
mpg_10_15
```

The **first** car is... buick skylark 320!

Inspect! How many cars are in the subset?

In [ ]:
```
#run the cell!
mpg_10_15.info()
```

There are ... 68 cars!

46. **Filter** the Dataframe **cars** for all cars that are **not built in the years 73 and 74**, and only the columns **"mpg"** and **"name"!**
Save the subset in the variable **not_73_74**. **Fill in the gaps!**

In [ ]:
```
not_73_74 = cars.loc[~cars.model_year.isin([73, 74]), ["mpg", "name"]].copy()
```

In [ ]:
```
#run the cell!
not_73_74
```

Inspect! How many cars are in this subset?

In [ ]:
```
# run the cell!
not_73_74.info()
```

There are ... 331 cars in the subset!

In [ ]:
```
# run the cell!
cars
```

47. **Drop** the columns **"displacement"** and **"cylinders"** from the DataFrame cars! **Save** the change!

In [ ]:
```
cars.drop(columns = ["displacement", "cylinders"], inplace = True)
```

In [ ]:
```
# run the cell!
cars
```

48. **Drop** all rows/cars from **"usa"** from the DataFrame cars! Use the **loc** operator and **overwrite cars**!

In [ ]:
```
cars = cars.loc[cars.origin != "usa"]
```

In [ ]:
```
# run the cell!
cars
```

49. **Add** the new column **"l_per_100km"** to the DataFrame cars by calculating **235.21/mpg**. **Round** to 2 decimals. **Fill in the gaps!**

In [ ]:
```
cars["l_per_100km"] = (235.21/cars.mpg).round(2)
```

Inspect cars! What is the **"l_per_100km"** value for the **audi 100 ls**?

In [ ]:
```
# run the cell!
cars
```

The "l_per_100km" value for the audi 100 ls is ... 9.80!

Well Done!

Section 6: Manipulating Elements in a DataFrame / Slice +++Important…

If you want to avoid the most commonly made mistakes in Pandas, this is probably the most important Section of the course!

There are two types of mistakes that even experienced Pandas coders are faced with:

1. Intention: Manipulating elements in a DataFrame. But: The code only manipulates these elements in a slice/subset. The original DataFrame remains unchanged.
2. Intention: Manipulating elements in a slice/copy of a DataFrame. But: The code also manipulates these elements in the original DataFrame.

Both types are dangerous and in some cases very hard to detect. In this Section, you will learn how Pandas work under the hood. And you will learn some simple rules to avoid these pitfalls.