PANDAS

- Python library used for **manipulation & analyzing data.**
- It has functions for **analyzing**, **cleaning**, **exploring** & **manipulating** data.

- **Pandas Series :**

    - Is a column in a table which holds a 1-D array of data of any datatype.
    - **Series() :** converts 1D array into column.
    - Create **labels** for the data passed through the **series** using **index** parameter inside the **series.**
    - Using the labels you have created, you can access the values. If the **labels** are not created then it'll be labeled with their index number, and can be accessed through that itself.
    - You can pass a **dictionary** as input to the **series.** When you pass a **dictionary**, then the **key** becomes **labels** to the **values.**

- **DataFrames :**

    - Datasets in Pandas are usually multi-dimensional tables, called **DataFrames.**
    - **Series** is like a column, a **DataFrame** is a whole table.
    - DataFrame is a 2-dimensional data structure, like a 2-d array or table with rows and columns.
    - **DataFrame()** : method is used to load the data into the object.
    - **loc[ ]** - attribute used to locate one or more rows. For getting more rows pass LIST of index values.
    - One row - Pandas Series. >1 row - DataFrame.
    - Here also you can name the index just like **Series** using the **index** parameter.
    - You can also give custom column names by using **names** parameter just like **index** parameter.
    - When you use named index, then while locating rows use those names for it.
    - For creating DataFrame, we can pass either **Dictionary** or **Tuple** directly or through the variable into **DataFrame()** function.
        - When the column names are specified while sending the above data, we have to pass the **columns** parameter inside **DataFrame()** .
    - To **transpose** a dataframe, use **<**DataFrame**>.T**
    - Loading **CSV** file into dataframe :
        - **read_csv(<**file_name**>)**
        - If the file is missing the header, then we should pass **header=None, names=['<**name1**>','<**name2**>','<**name3**>'....]** inside **DataFrame().**

    - **skip(<**Number_of_rows_to_skip**>) :** function used to skip the rows from top.
    - **header(<**Row_index**>) :** Used to set the row of the file as header using the index of the row.
    - **to_string() :** method to print **entire** dataframe else the pandas return only the first 5 and last 5 rows of the file.
    - **options.display.max_rows :** used to **check** and **change** the max rows that the system can display. More than that it'll display only headers and first 5 and last 5 rows.
- Loading **JSON** file into dataframe :
    - **read_json(<**file_name**>)**
- **unique() :** gives you unique elements in that column.
- Columns can be given as arguments. Or as index values.

```
[movieDataFrame.imdb_rating,
```

```
movieDataFrame[movieDataFrame.industry=='Hollywood']
```

- Loading **EXCEL** file into dataframe :
    - **read_excel(<**excelFileName**>,<**SheetName**>)**

- **Analyzing DataFrames :**

    - Viewing Data :
        - **head(<**value**>)** - method is used to get the headers and the specified numbers of rows starting from top. If the value is not specified then it will return the top 5 rows.
        - **tail(<**value**>)** - method also returns the headers but the rows from the bottom. If the value is not specified then 5 rows from bottom will be returned.
        - **sample(<**value**>)** - method which returns random rows of specific numbers.
        - **info()** - method returns more info regarding the dataset.
        - **describe() -** Gives you the numerical statistics values of the columns with integer data from the DataFrame.

- **Cleaning Data :**

    - Means fixing bad data in the dataset.
    - Bad data :
        1. Empty cells.

2. Data in wrong format.
3. Wrong data.
4. Removing Duplicates.

1. **Empty Cells :**
    - Empty cells might give you wrong results when you analyze the data.
    - Methods to handle empty cells :
        - Remove Rows :
            - Removing the entire rows which contain empty cells.
            - Since the datasets are big, removing some rows doesn't affect much.
            - **dropna()** : method is used to **remove the rows** that contain empty cells.
            - By default it returns a new DataFrame and will not change the original DataFrame.
            - If you want to make changes to the original DataFrame, then pass **inplace = True** argument within **dropna().**

        - Replace Empty Values :
            - We can insert a new value in place of empty cells, by doing so we can avoid deleting the entire rows.
            - **fillna()** : method is used to **replace** all the empty cells with a value.
            - Should pass the value inside **fillna()**.
            - By default it returns a new DataFrame and will not change the original DataFrame.
            - If you want to make changes to the original DataFrame, then pass **inplace = True** argument within **dropna().**
            - We can pass **method** parameter inside **fillna()**, where it'll take **ffill/bfill** as value.
                - **ffill :** carry forward the previous valid data to fill an empty cell.
                - **bfill :** use next valid data to fill the empty cell.
            - **replace(<**which_value**>,<**with_what_value**>)** : method used to replace the **<**which_value**>** with **<**with_what_value**>**. To show it as **Nan** call **numpy.nan**
            - **<**which_value**> :** this can be a list of values or a dictionary with all the columns and values or old value to new value.

        - Replace Only For Specified Columns :

- To do that, specify the column_name for the DataFrame.

- Replace using Mean, Median or Mode :
  - Common way of replacing the empty cells is to calculate the mean, median or mode value of the column.
  - **mean(), median(), mode(),** or **interpolate()**

Mean = the average value (the sum of all values divided by number of values).

Median = the value in the middle, after you have sorted all values ascending.

Mode = the value that appears most frequently.

In short, interpolation is a process of determining the unknown values that lie in between the known data points.

2. **Data in Wrong Format :**
   - Cells with data of wrong format can make it difficult or even impossible to analyze data.
   - Two options to fix it :
     1. Remove the rows
     2. Convert all the cells in the columns into the same format.

3. **Wrong Data :**
   - "Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".
   - Sometimes you can spot wrong data by looking at the data set, because you have an expectation of what it should be.
   - Two options :
     1. Replacing the wrong the data with correct one or according to the boundaries set for the values
     2. Removing the rows. [**drop()**]

4. **Removing Duplicates :**
   - Same data registered more than one time.
   - **duplicated() :** method used to identify the duplicate data in the datasets.
   - This returns **True** for the index if it is a duplicate, else **False** for that index.
   - **drop_duplicates() :** used to remove the duplicates from the DataFrame.

- **Data Correlations :**

    - Finding Relationships : **corr()** method is used to calculate the relationships between columns in DataFrame.
    - **corr()** ignores the 'non-numeric' columns.
    - Result of the **corr()** will be a table which includes numbers. Number will be between **-1 to 1**.

What is a good correlation? It depends on the use, but I think it is safe to say you have to have at least `0.6` (or `-0.6`) to call it a good correlation.

    - Perfect Correlation [**1**]: each column always has a perfect relationship with itself. Each time a value went up in the first column, the other one went up as well.
    - Good Correlation : [For +ve value] If you increase one value, the other will probably increase as well.
      [For -ve value] If you increase one value, the other will probably go down.
    - Bad Correlation : Meaning that if one value goes up does not mean that the other will.

- **groupby() :** method used to group the entire data frame based on the value of a column which is mentioned inside the function.
    - It returns a **DataFrameGroupBy** object which will contain all the grouped data with condition as their **key.**
    - **get_group() :** by passing the column, key in this method, you can access that data.

- **concat() :** provide the list of DataFrames that you want to concatenate.
    - Can provide **keys** for each DataFrame and later can access using the same

- **merge() :** merges DataFrames.
    - Pass **on** parameter on which column you want to merge the DataFrames
    - Pass **how** parameter on which what kind of JOIN it has to perform
        - By default, **inner**
        - We have : **left, right, outer, inner, cross**
- **merge()** is used for joining data frames based on a **common column or index**, **concat()** is used for concatenating data frames either **vertically or horizontally**.

- **<**DataFrame**>.pivot(index = <**Values to be on X-axis**>, columns = <**Values to be on Y-axis**>, values = <**Which values to display**>) :**
  - Used to reshape the DataFrame

- **Pivot table :** used to summarize and aggregate the data inside the dataframe.
  - **<**DataFrame**>.pivot_table(index = <**Values to be on X-axis**>, columns = <**Values to be on Y-axis**>, aggfunc = <**Operation to do**>)**

- **melt() :** Pandas method used to transform or reshape the data in DataFrame.
  - This function is useful to massage a DataFrame into a format where one or more columns are **identifier variables (id_vars)**, while all other columns, considered measured **variables (value_vars)**, are “unpivoted” to the row axis, leaving just two non-identifier columns, **‘variable’** and **‘value’**

```
pandas.melt(frame, id_vars=None, value_vars=None, var_name=None,
value_name='value', col_level=None, ignore_index=True)
```

- **stack() :** Pandas method used to reshape the DataFrame which has multi-level headers. When you call the function, it transforms the DataFrame with the innermost header as the row headers.

```
DataFrame.stack(level=-1, dropna=_NoDefault.no_default,
sort=_NoDefault.no_default, future_stack=False) #
```

  - By default, **dropna** will be **True.**
  - **level :** Should give which level to be taken as row headers from the column headers.

```
import pandas as pd
df = pd.read_excel("stocks.xlsx",header=[0,1])
df
```

	Price			Price to earnings ratio (P/E)		
Company	Facebook	Google	Microsoft	Facebook	Google	Microsoft
2017-06-05	155	955	66	37.10	32.0	30.31
2017-06-06	150	987	69	36.98	31.3	30.56
2017-06-07	153	963	62	36.78	31.7	30.46
2017-06-08	155	1000	61	36.11	31.2	30.11
2017-06-09	156	1012	66	37.07	30.0	31.00

```
df.stack()
```

	Company	Price	Price to earnings ratio (P/E)
2017-06-05	Facebook	155	37.10
	Google	955	32.00
	Microsoft	66	30.31
2017-06-06	Facebook	150	36.98
	Google	987	31.30
	Microsoft	69	30.56
2017-06-07	Facebook	153	36.78
	Google	963	31.70
	Microsoft	62	30.46
2017-06-08	Facebook	155	36.11
	Google	1000	31.20
	Microsoft	61	30.11
2017-06-09	Facebook	156	37.07
	Google	1012	30.00
	Microsoft	66	31.00

```
df.stack(level=0)
```

	Company	Facebook	Google	Microsoft
2017-06-05	Price	155.00	955.0	66.00
	Price to earnings ratio (P/E)	37.10	32.0	30.31
2017-06-06	Price	150.00	987.0	69.00
	Price to earnings ratio (P/E)	36.98	31.3	30.56
2017-06-07	Price	153.00	963.0	62.00
	Price to earnings ratio (P/E)	36.78	31.7	30.46
2017-06-08	Price	155.00	1000.0	61.00
	Price to earnings ratio (P/E)	36.11	31.2	30.11
2017-06-09	Price	156.00	1012.0	66.00
	Price to earnings ratio (P/E)	37.07	30.0	31.00

- **crosstab()** : used to get the frequency distribution of the data inside the DataFrame.

```
pandas.crosstab(index, columns, values=None, rownames=None, colnames=None,
aggfunc=None, margins=False, margins_name='All', dropna=True,
normalize=False)                                                    [source]
```

- **Index :** Row side headers
- **Columns :** Column headers
- **Margins :** Gives you the total of each column and row

```
pd.crosstab(df.Nationality,df.Handedness)
```

Handedness	Left	Right
Nationality		
Bangadesh	2	0
China	2	1
India	2	1
USA	1	3

```
pd.crosstab(df.Sex,df.Handedness)
```

Handedness	Left	Right
Sex		
Female	2	3
Male	5	2

Multi Index Column and Rows

```
pd.crosstab(df.Sex, [df.Handedness,df.Nationality], margins=True)
```

Handedness	Left				Right			All
Nationality	Bangadesh	China	India	USA	China	India	USA	
Sex								
Female	1	1	0	0	1	0	2	5
Male	1	1	2	1	0	1	1	7
All	2	2	2	1	1	1	3	12

- **Pandas Plotting :**

    - **plot()** is used for creating the diagrams.
    - **PyPlot** submodule of the Matplotlib library is used to visualize the diagrams.

    - **Scatter Plot :**
        - This needs x-axis and y-axis
        - Pass **kind = 'scatter'** inside **plot().**
        - Should pass column names against the x & y axis which you want to plot inside the **plot().**

    - **Histogram :**
        - Pass **kind='hist'** inside **plot().**
        - Histogram needs only one column.
        - It shows the frequency of each interval.