Unit 2: Toolboxes for Data Scientists.

* Importing Matplotlib, Numpy and Pandas

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

* DataFrame

- a data structure in Pandas
- It is a tabular data structure with rows and columns
- Rows can be accessed through their index values
- Columns are called Series - a list with several values
- A data frame is usually a dictionary of lists.

* Creating a DataFrame

```
data = {'Name': ['Pooja', 'Preetha', 'Pranav', 'Premnath'],
        'Nationality': ['Indian', 'American', 'Japanese', 'Portuguese'],
        'Age' :[12, 45, 2, 46]}
```

~~Grade~~

```
info = pd.DataFrame(data)        # info is a DataFrame, made of the DataFrame object constructor
```

To specify the order in which columns are to be displayed, add a keyword columns specifying column names in order. If the column keyword is not used, then the columns are arranged alphabetically

i.e `info = pd.DataFrame(data, columns = ['Nationality', 'Age', 'Name'])`

* Reading Data

- Data can be read using the .csv_read function.

```
info = pd.~~csv_read~~('File path',
        read_csv na_values = ':',
                 usecols = [column list])
```

→ na_values is used along with the character in the file that represents non-available data.

→ ~~use cols~~ usecols shows which columns from the csv file have to be read.

* To show the first and last few rows of a table

<dataframevar>.head()

<dataframevar>.tail()

specify a number within the parantheses to get ~~a no.~~ the first / last n rows.

* To find the names of the columns in the table.

<dataframevar>.columns

o/p Index([list of columns], dtype = 'object')

* To find the index range in the table

<dataframevar>.index

o/p RangeIndex(start=0, stop= , step = 1)

* Statistical information on all the numeric values

<dataframevar>.describe()

displays the count, mean, standard deviation, minimum and maximum values and the percentiles (25th, 50th and 75th), of each column / series

* Selecting data

(i) selecting columns

<dataframevar>['colname'].

(ii) selecting rows

<dataframevar>[:]

slice.

Note: The slice corresponds to the position, not the index labels.

→ To select data based on the labels as the reference instead of the positions, use:

<dataframevar>.loc[, cols]
slice

for eg. <dataframevar>.loc[90:94, ['Name','Nationality']]

o/p This would return all the rows in between the specified indices (limits inclusive), and the columns specified.

- loc references the index labels, which means that it returns all the rows labelled between 90 and 94.
- If the index 100 was placed between 90 and 94, then it would be returned as well.

* Filtering Data

- Boolean indexing can be applied to filter data. It uses the result of a Boolean operation over the data, and it returns a mask with True or False for each row.
- The rows marked True in the mask will be selected.

syntax <dataframevar>[<dataframe>['colname'] >= <= != > < value]

for eg. if info is the name of the DataFrame, then

info[info['Name'] >= 5]

- Filtering missing values is done with the isnull() function

for eg. info[info['Name'].isnull()]

Manipulating Data - use aggregate functions like count(), sum(), mean(), median(), max(), min(), prod(), std(), var()

Syntax : <dataframevar>.<aggregatefn()> → axis = 0 or 1

to apply on rows : axis = 1

to apply on columns : axis = 0

* Difference between NaN values in python and pandas

- In python, NaN values propagate through all operations without raising an exception.
- In pandas, operations exclude NaN values representing missing data.

for eg.

Print ("Pandas max function", info['Value'].max()) ⇒ ano.

whereas

print("Python max function", max(edu['Value'])) ⇒ NaN

* Performing mathematical operations over values

- for eg. to display all the value of the column 'Age' divided by 5

var = info['Age'] / 5

var

- use the apply function

var = info['Age'].apply(np.sqrt)

var

* Lambda Functions to perform operations

- If specific operations are to be performed, inline functions can be written.
- These are lambda functions, and they have no name.
- It is only necessary to specify the parameters, and the operation using the lamda keyword.

, . For example, in order to square all the ages in a table, a lambda function would be:

```
var = info['Age'].apply(lambda d: d**2)
var
```

* Adding and Removing Rows and Columns

(i) adding a new column

Syntax: <dataframevar>['colname'] = value

for example: in order to add a new column, with the squared value of ages, then

```
info['SquaredAge'] = info['Age'].apply(lambda d : d**2)
```

(ii) deleting a column

Syntax: <dataframe>.drop('colname', axis = 1, inplace = True)

Note: Changes made to the contents of a data frame (like the drop function) usually return a copy of the modified data. This means that the original DataFrame is kept. If changes are to be made to the original table, set the parameter Inplace to True. (By default, inplace = False)

for eg. To delete the SquaredAge column from the DataFrame

```
info.drop('SquaredAge', axis=1, inplace= True)
```

(iii) Adding Rows

Syntax: <dataframevar> = <dataframevar>.append(dictionary, ignore_index = True)

Note: The ignore_index flag must be set to True, otherwise the index zero is given to the new row, which will raise an error if it already exists.

(iv) Removing a Row

Syntax: <dataframevar>.drop(max(<dataframevar>.index), axis=0 inplace = True)

* Deleting and filling NaN values

Method1: use the drop function

<dataframevar> = <dataframevar>.drop(<dataframevar>['column'].isnull(), axis = 0)

Method2 : filter out NaN values

<dataframevar>[<dataframevar>['col'].isnull()]

Difference: The drop function returns a copy of the DataFrame, while filtering the table of NaN values returns a views.

- **Using the dropna() function :**

syntax: ~~<dataframe~~

var = <dataframevar>.dropna(how = 'any' , subset = ['colname'])

how = 'any' : To erase any row that has a null value

subset = ['colname'] : To restrict deletion to only a set of columns

Filling cells with NaN values with other data

var = <dataframevar>.fillna(value = { 'colname' : value , ... })

Sorting

syntax: <dataframevar>.sort_values(by = 'colname', ascending = True, inplace = True to make permanent changes)

To return to the original order

syntax :

<dataframe>.sort_index(axis = 0 , ascending = True, inplace = True)

* Grouping Data

Syntax : var = <dataframevar>[['col list']].groupby('col').agg.fn()

For example: In order to display the mean values pertaining to each country,

var = info[['GEO','Value']].groupby('GEO').mean()

* Rearranging Data. — pivot tables

- Syntax : var = pd.pivot_table(data, values = ' ',
  index = [' '],
  columns = [' '])

- To access certain entries from the pivot-table

syntax : <pivotvar>.loc[[' row list', ..], [' col list']]

* Ranking data

* Renaming indices

Syntax : <dataframevar>.rename(index = {"old name" : "new name"})

* Plotting bar graphs

To plot a bar graph showing the sum of all the values across the years in descending order.

```
yearsum= pivot_info.sum(axis=1).sort_values(ascending=False)
```

↑ sum uprows

```
yearsum.plot(kind='bar', color='maroon', alpha=1, title="Total values for countries")
```

↓ can be barh as well

↓ handles opacity of bars

plotting a pivot table as a cohole

```
colors=['b','r','g','y','m','c'].
graph= pivot_info.plot(kind='barh',
                       stacked=True,
                       color='colors')
```

```
graph.legend(loc='', bbox_to_anchor(1, .5))
```

↓ set relative position of legend w.r.t plot

can take the values right or left & upper lower center.

↓ to set an absolute position w.r.t the plot, put legend outside graph

NumPy and Matplotlib Functions

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

Function	Use	Syntax
pd.DataFrame( )	To create a data frame	var=pd.DataFrame(<dictionary of lists>,columns=[column list])
pd.csv_read()	To read a csv file na_values=to indicate the character that represents non available data usecols= can choose which columns to read/display in the csv file	var=pd,csv_read('file path', na_values=':',usecols=[column list])
head(n)	To read the first n rows of a data frame, by default reads 5	<dataframevar>head(n)
tail(n)	To read the last n rows of a data frame, by default reads 5	<dataframevar>.tail(n)
columns	To list the names of the columns in the table	<dataframevar>columns Output: Index([list of columns]), dtype='object')]
index	To find the index range in the table	<dataframevar>.index Output: RangeIndex(start=0, end= ,step=)

describe()	Provides statistical information- displays the count, mean, standard deviation, minimum and maximum values and the percentiles (25^{th}, 50^{th} and 75^{th}) of each column/series	<dataframevar>.describe()
Selecting rows and columns	To select the columns in a table	<dataframevar>[colname]
	To select the rows in a table	<dataframevar>[:] (a slice) Note: The slice corresponds to the positions, and not to the index labels <dataframevar>.iloc[slice, column names] This can be used to select columns on the keeping the label names as references, rather than the positions. Slices are limit inclusive.
Filtering	To filter data based on the result of a Boolean condition	<dataframevar>[<datframevar>[colname] </>/<=/>=/!= value]
Filtering null values	To show all the entries that have null values	<dataframevar>[<dataframevar>['colname'] .isnull()]
Aggregate functions	count() sum() min() max() mean() prod() std()	<dataframevar>.<aggregatefunction> (axis=0/1) To apply on rows of each column axis=0 To apply on columns of each row axis=1

	var()	
Mathematical operations	To compute the values of certain entries after performing a mathematical operation on them	var=<dataframevar>[column]/5 (or any other operation) var
apply()	To apply certain functions from the numpy library	var=<dataframevar>.apply(function) For eg. Var=info["Age"].apply(np.sqrt)
Lambda Functions	Single line functions to perform user defined tasks	var=<dataframevar>.apply(lambda param: statement)
Adding a column	To add a new column with values	<dataframevar>[Column Name]=value
drop()- on columns	To delete a column axis=1, since one element per row must be deleted inplace=False by default, makes a copy of the original dataframe. By setting it to True, changes are permanently made in the original data frame.	<dataframevar>.drop(colname,axis=1, inplace=True)_

Adding a row	To add a new entry to the dataframe Set the ignore_index to True, otherwise, the index zero is given to the newly added row, which will raise an error if there is already a row at the zeroth index.	<dataframevar>=<dataframevar>.append (dictionary, ignore_index=True)
Removing a row	To remove the last element/any slice of elements from a dataframe	<dataframevar>.drop(slice,axis=0,inplace =True) #To remove the last row <dataframevar>.drop(max(<dataframevar >. index),axis=0)
Deleting NaN Values	To remove the null values from a column in a data frame	Method 1: var=<dataframevar>.drop(<dataframevar>[column].isnull(),axis=0) Method 2: <dataframevar>[<datframevar>[column] .isnull()] Difference: The drop function returns a copy of DataFrame, the filtering results in a view.
dropna()	To specifically remove null values from rows/columns how='any' : To erase any row that has a null value	var=<dataframevar>.dropna(how='any', subset=[column names])

	subset=[column list] : to restrict the deletion of null values to only a set of columns	
fillna()	To fill certain rows and columns having null values with a specific value fillna() a dictionary of column names and the values to be filled in it as parameters	var=<dataframevar>.fillna(value= {'column name': value, ...})
sort_values()	To sort a dataframe on the basis of values set ascending=False to sort in descending order	var=<dataframevar>.sort_values (by='colname', ascending=True,inplace=True)
sort_index()	To sort a data frame on the basis of index values Use the sort_index() function to revert changes made by the sort_values() function.	var=<dataframevar>sort_index(axis=0, ascending=True,inplace=True)
groupby()	To split the data into groups on	var=<dataframevar>[[columns needed]]. groupby('col').aggregatefunction()

	the basis of a certain column An aggregate function must be used along with the groupby()	
Pivot Tables	To create a new table on the basis of newly specified indices, columns and values	var=pd.pivot_table(data,values='row names to be used', index=[' column list'], columns=[column list'])
Filtering Pivot Tables	To access certain values from a pivot table	<pivotvar>.loc([row list],[column list])
rename()	To rename index names	<dataframevar>.rename(index= {"old name": "new name"}
plot()	To plot graphs using matplotlib Parameters kind= 'bar'/'barh'- To draw vertical and horizontal bar graphs color= a list of colors/a single color for the bars of the graph alpha= a numerical value to handle the opacity of the bars	<dataframvar>,plot(kind,color,alpha,title)

	title=' ' To set the title of the bar graph stacked= True/False- To plot data from multiple columns in the data frame on top of each other	
legend()	The legend function can be used when there are several series shown in a plot, legend is the object that is returned when the plot function is called Parameters loc: To specify the relative position of the legend with respect to the plot Can take the values i. center right ii. center left iii. center upper iv. center lower	<plotvar>.legend(loc, bbox_to_anchor(num1,num2))

	bbox_to_anchor ; To set an absolute position with respect to the plot, to enable one to put the legend outside the plot.	