Python Libraries

**Pandas**

**Pandas** is an open-source Python Library used for high-performance data manipulation and data analysis using its powerful data structures. Python with pandas is in use in a variety of academic and commercial domains, including Finance, Economics, Statistics, Advertising, Web Analytics, and more. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, organize, manipulate, model, and analyse the data.

**NumPy**

**NumPy** is a Python package which stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

**SciPy**

The SciPy library of Python is built to work with NumPy arrays and provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install and are free of charge. NumPy and SciPy are easy to use, but powerful enough to depend on by some of the world's leading scientists and engineers

**Matplotlib**

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.

**Beginning Data Analytics - Week 3**

**Practice Exercise Handout 3.1 for Week 3**

**Instructions**

1. Download **descriptiveS.csv** from moodle into <your> folder
2. Launch jupyter from <your> folder
3. Copy and Paste or type the codes available in the below boxes and run to see the results
4. Copy the codes available in the box only , for practices.
5. Sometime you may have to play run button twice or thrice to see the results.

**Descriptive Statistics in Python**

* pandas and numpy are the libraries.
* We have to load libraries time to time depending upon the type of code we use in python programming

**3.1.1 – Loading data and observing data**

**Your Code**

import pandas as pd

import numpy as np

data1 = pd.read\_csv("descriptiveS.csv")

print(data1)

**Result**

Group Marks

0 Group 1 26.4

1 Group 1 8.4

. . . . . . . . . .

. . . . . . . . . .

**3.1.2 Observing data**

print(data1)

**3.1.3 Observing first 5 top rows with heading**

data1.head()

**3.1.4 Summary Statistics**

The **describe()** function computes a summary of statistics pertaining to the DataFrame columns.

print(data1.describe())

**3.1.5 Showing all statistical information**

print(data1.describe(include="all"))

**Functions & Description**

Let us now understand the functions under Descriptive Statistics in Python Pandas. The following table list down the important functions −

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Function** | **Description** |
| 1 | count() | Number of non-null observations |
| 2 | sum() | Sum of values |
| 3 | mean() | Mean of Values |
| 4 | median() | Median of Values |
| 5 | mode() | Mode of values |
| 6 | std() | Standard Deviation of the Values |
| 7 | min() | Minimum Value |
| 8 | max() | Maximum Value |
| 9 | abs() | Absolute Value |
| 10 | prod() | Product of Values |
| 11 | cumsum() | Cumulative Sum |
| 12 | cumprod() | Cumulative Product |

Now we are testing above table function. (Description of functions are given in the above table)

**Example 3.1.6**

data1. count()

**Example 3.1.7**

data1.sum()

**Example 3.1.8**

data1.mean()

**Example 3.1.9**

data1.median()

**Example 3.1.10**

data1.mode()

**Example 3.1.11**

data1.std()

**Example 3.1.12**

data1.min()

**Example 3.1.13**

data1.max()

**Example 3.1.14**

data1.abs()

**Example 3.1.15**

data1.prod()

**Example 3.1.16**

data1.cumsum()

**Example 3.1.17**

data1.cumprod()

**Example 3.1.18**

**Plotting Bar Chart for “MarksA” column of the table**

x=data1['MarksA']

x.plot.bar()

**Example 3.1.19**

You can also draw bar chart using single line code

data1['MarksA'].plot.bar()

**Function for Drawing various different types of graphs**

* ‘line’ : line plot (default)
* ‘bar’ : vertical bar plot
* ‘barh’ : horizontal bar plot
* ‘hist’ : histogram
* ‘box’ : boxplot
* ‘kde’ : Kernel Density Estimation plot
* ‘density’ : same as ‘kde’
* ‘area’ : area plot
* ‘pie’ : pie plot
* ‘scatter’ : scatter plot
* ‘hexbin’ : hexbin plot

Now we are Testing above table function. (Description of functions are given in above table)

**Example 3.1.20**

data1['MarksA'].plot.line()

**Example 3.1.21**

data1['MarksA'].plot.barh()

**Example 3.1.22**

data1['MarksA'].plot.hist()

**Example 3.1.23**

data1['MarksA'].plot.box()

**Example 3.1.24**

data1['MarksA'].plot.kde()

**Example 3.1.25**

data1['MarksA'].plot.density()

**Example 3.1.26**

data1['MarksA'].plot.area()

**Example 3.1.27**

data1['MarksA'].plot.pie()

**Example 3.1.28**

For scatter plot you need two variables

data1.plot.scatter(x='MarksA',y="MarksB")

**Example 3.1.29**

**Using different color**

data1.plot.scatter(x='MarksA', y="MarksB",color=['red','green'])

**Example 3.1.30**

data1.plot.bar(x='Group', y='MarksA')

**Bar Graph with groups**

**3.2.1**

import pandas as pd

import numpy as np

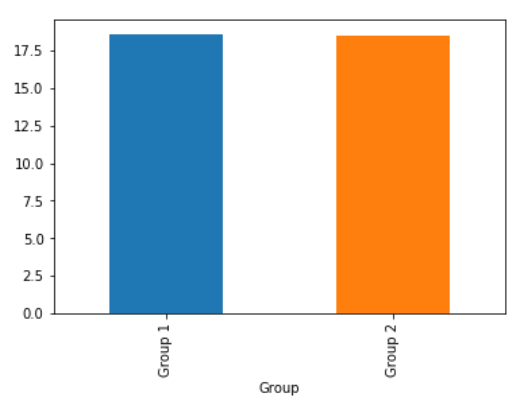
import matplotlib.pyplot as plt

%matplotlib inline

data1 = pd.read\_csv("descriptiveS.csv")

data1.groupby("Group").MarksA.mean().plot.bar()

Results

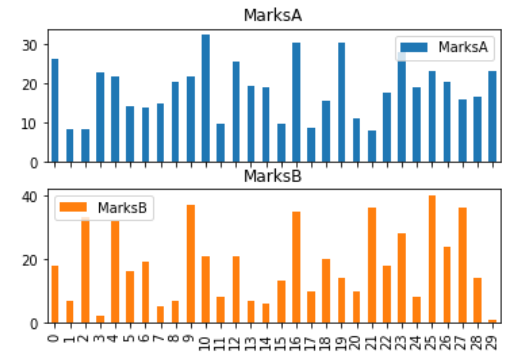


**Generating Subplots**

**3.2.2**

axes = data1.plot.bar(subplots=True)

**Results**

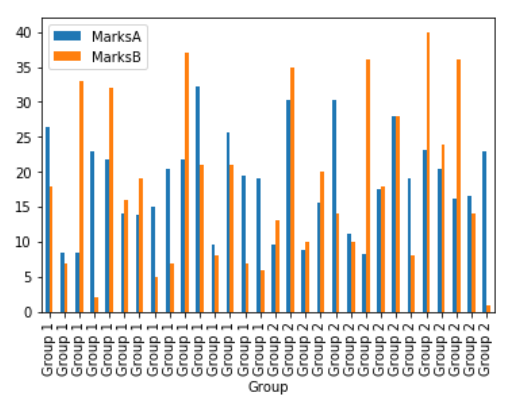
****

**Generating Bar graph –combined**

**3.2.3**

data1.plot(x="Group", y=["MarksA", "MarksB"], kind="bar")

**Results**



**Generating Graph – Grouped**

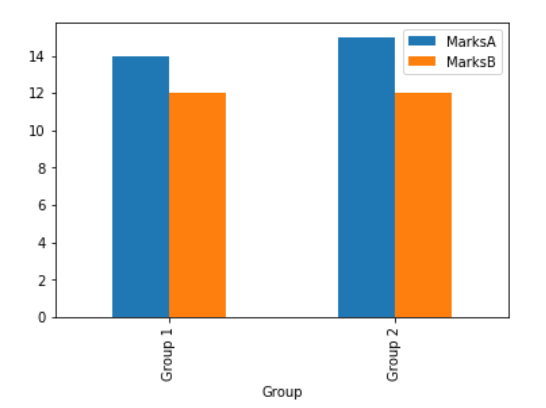
**3.2.4**

import matplotlib.pyplot as plt

import pandas as pd

data1.groupby('Group')['MarksA','MarksB'].nunique().plot(kind='bar')

Results

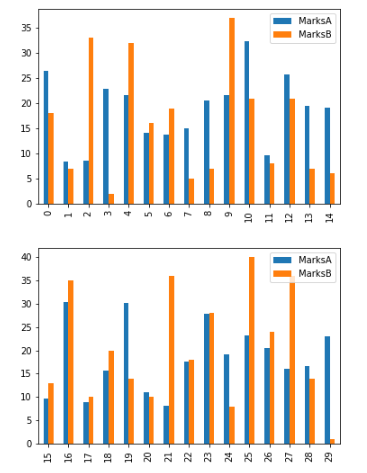


**Generating Graph – Grouped (Another Way)**

**3.2.5**

data1.groupby('Group')['MarksA','MarksB'].plot(kind='bar')

**Results**



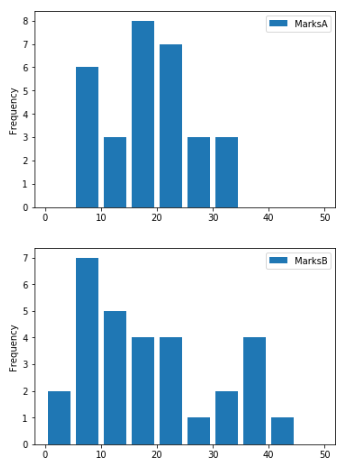
**3.2.6**

**Plotting Histogram**

data1[['MarksA']].plot(kind='hist',bins=[0,5,10,15,20,25,30,35,40,45,50],rwidth=0.8)

data1[['MarksB']].plot(kind='hist',bins=[0,5,10,15,20,25,30,35,40,45,50],rwidth=0.8)

**Results**



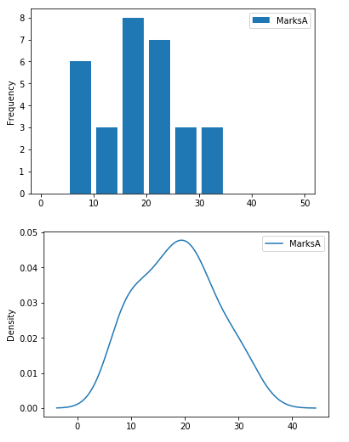
**3.2.7**

**Plotting Histogram and density curve separately**

data1[['MarksA']].plot(kind='hist',bins=[0,5,10,15,20,25,30,35,40,45,50],rwidth=0.8)

data1[['MarksA']].plot(kind='kde')

**Results**



**3.2.13**

import matplotlib.pyplot as plt

%matplotlib inline

plt.style.use('ggplot')

%matplotlib inline

plt.style.use('ggplot')

plt.bar(data1.Group, data1.MarksA, color='green')

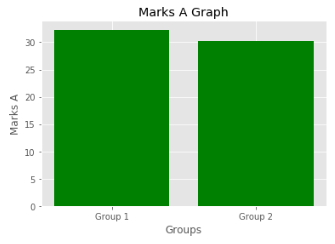
plt.xlabel("Groups")

plt.ylabel("Marks A")

plt.title("Marks A Graph")

plt.show()

Result



Python Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.

Key Features of Pandas

* Fast and efficient DataFrame object with default and customized indexing.
* Tools for loading data into in-memory data objects from different file formats.
* Data alignment and integrated handling of missing data.
* Reshaping and pivoting of date sets.
* Label-based slicing, indexing and subsetting of large data sets.
* Columns from a data structure can be deleted or inserted.
* Group by data for aggregation and transformations.
* High performance merging and joining of data.
* Time Series functionality.

Pandas deals with the following three data structures −

* Series
* DataFrame
* Panel

These data structures are built on top of Numpy array, which means they are fast.

## Series

Series is a one-dimensional array like structure with homogeneous data. For example, the following series is a collection of integers 10, 23, 56, …

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 23 | 56 | 17 | 52 | 61 | 73 | 90 | 26 | 72 |

### **Key Points**

* Homogeneous data
* Size Immutable
* Values of Data Mutable

## DataFrame

DataFrame is a two-dimensional array with heterogeneous data. For example,

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Age** | **Gender** | **Rating** |
| Steve | 32 | Male | 3.45 |
| Lia | 28 | Female | 4.6 |
| Vin | 45 | Male | 3.9 |
| Katie | 38 | Female | 2.78 |

The table represents the data of a sales team of an organization with their overall performance rating. The data is represented in rows and columns. Each column represents an attribute and each row represents a person.

## Data Type of Columns

The data types of the four columns are as follows −

|  |  |
| --- | --- |
| **Column** | **Type** |
| Name | String |
| Age | Integer |
| Gender | String |
| Rating | Float |

### **Key Points**

* Heterogeneous data (Diverse in content)
* Size Mutable (Size changeable)
* Data Mutable (Data changeable)

## Panel

Panel is a three-dimensional data structure with heterogeneous data. It is hard to represent the panel in graphical representation. But a panel can be illustrated as a container of DataFrame.

### **Key Points**

* Heterogeneous data
* Size Mutable
* Data Mutable

## Pandas Series

Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.

he parameters of the constructor are as follows −

|  |  |
| --- | --- |
| **Sr.No** | **Parameter & Description** |
| 1 | **data**  data takes various forms like ndarray, list, constants |
| 2 | **index**  Index values must be unique and hashable, same length as data. Default **np.arrange(n)** if no index is passed. |
| 3 | **dtype**  dtype is for data type. If None, data type will be inferred |
| 4 | **copy**  Copy data. Default False |

## pandas.Series

A pandas Series can be created using the following constructor −

pandas.Series( data, index, dtype, copy)

A series can be created using various inputs like −

* Array
* Dict
* Scalar value or constant

## Create an Empty Series

A basic series, which can be created is an Empty Series.

### **Example 1**

#import the pandas library and aliasing as pd

import pandas as pd

s = pd.Series()

print(s)

s

Its **output** is as follows −

Series([], dtype: float64)

## Create a Series from ndarray

If data is an ndarray, then index passed must be of the same length. If no index is passed, then by default index will be **range(n)** where **n** is array length, i.e., [0,1,2,3…. **range(len(array))-1].**

### **Example 2**

#import the pandas library and aliasing as pd

import pandas as pd

import numpy as np

data = np.array(['a','b','c','d'])

s = pd.Series(data)

print(s)

Its **output** is as follows −

0 a

1 b

2 c

3 d

dtype: object

We did not pass any index, so by default, it assigned the indexes ranging from 0 to **len(data)-1**, i.e., 0 to 3.

Its **output** is as follows −

100 a

101 b

102 c

103 d

dtype: object

We passed the index values here. Now we can see the customized indexed values in the output.

## Create a Series from dict

A **dict** can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index. If **index** is passed, the values in data corresponding to the labels in the index will be pulled out.

## Create a Series from Scalar

If data is a scalar value, an index must be provided. The value will be repeated to match the length of **index**

### **Example 5**

#import the pandas library and aliasing as pd

import pandas as pd

import numpy as np

s = pd.Series(5, index=[0, 1, 2, 3])

print(s)

Its **output** is as follows −

0 5

1 5

2 5

3 5

dtype: int64

## Accessing Data from Series with Position

Data in the series can be accessed similar to that in an **ndarray.**

### **Example 6**

Retrieve the first element. As we already know, the counting starts from zero for the array, which means the first element is stored at zeroth position and so on.

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first element

print(s[0])

Its **output** is as follows −

1

Retrieve the first three elements in the Series. If a : is inserted in front of it, all items from that index onwards will be extracted. If two parameters (with : between them) is used, items between the two indexes (not including the stop index)

### **Example 7**

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first three element

print(s[:3])

Its **output** is as follows −

a 1

b 2

c 3

dtype: int64

### **Example 8**

Retrieve the last three elements.

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the last three element

print(s[-3:])

Its **output** is as follows −

c 3

d 4

e 5

dtype: int64

## Retrieve Data Using Label (Index)

A Series is like a fixed-size **dict** in that you can get and set values by index label.

### **Example 9**

Retrieve a single element using index label value.

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve a single element

print(s['a'])

Its **output** is as follows −

1

### **Example 10**

Retrieve multiple elements using a list of index label values.

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve multiple elements

print(s[['a','c','d']])

Its **output** is as follows −

a 1

c 3

d 4

dtype: int64

### **Example 11**

If a label is not contained, an exception is raised.

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve multiple elements

Print(s['f'])

Its **output** is as follows −

…

KeyError: 'f'

**Data Frame**

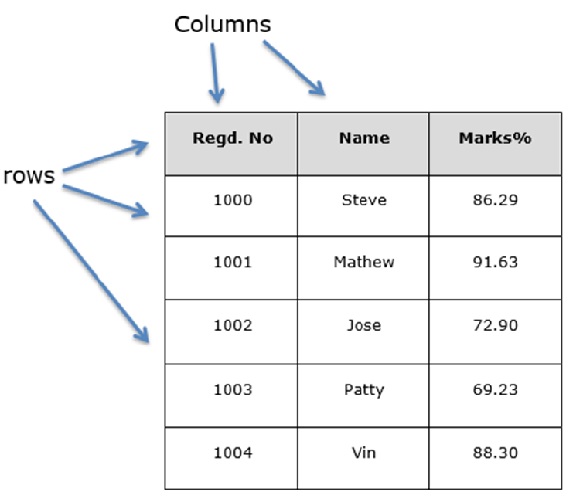
A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

### **Features of DataFrame**

* Potentially columns are of different types
* Size – Mutable
* Labeled axes (rows and columns)
* Can Perform Arithmetic operations on rows and columns

### **Structure**

Let us assume that we are creating a data frame with student’s data.



You can think of it as an SQL table or a spreadsheet data representation.

## pandas.DataFrame

A pandas DataFrame can be created using the following constructor −

pandas.DataFrame( data, index, columns, dtype, copy)

The parameters of the constructor are as follows −

|  |  |
| --- | --- |
| **Sr.No** | **Parameter & Description** |
| 1 | **data**  data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame. |
| 2 | **index**  For the row labels, the Index to be used for the resulting frame is Optional Default np.arange(n) if no index is passed. |
| 3 | **columns**  For column labels, the optional default syntax is - np.arange(n). This is only true if no index is passed. |
| 4 | **dtype**  Data type of each column. |
| 5 | **copy**  This command (or whatever it is) is used for copying of data, if the default is False. |

## Create DataFrame

A pandas DataFrame can be created using various inputs like −

* Lists
* dict
* Series
* Numpy ndarrays
* Another DataFrame

In the subsequent sections of this chapter, we will see how to create a DataFrame using these inputs.

## Create an Empty DataFrame

A basic DataFrame, which can be created is an Empty Dataframe.

### **Example 12**

#import the pandas library and aliasing as pd

import pandas as pd

df = pd.DataFrame()

print(df)

Its **output** is as follows −

Empty DataFrame

Columns: []

Index: []

## Create a DataFrame from Lists

The DataFrame can be created using a single list or a list of lists.

### **Example 13**

import pandas as pd

data = [1,2,3,4,5]

df = pd.DataFrame(data)

print(df)

Its **output** is as follows −

0

0 1

1 2

2 3

3 4

4 5

### **Example 14**

import pandas as pd

data = [['Alex',10],['Bob',12],['Clarke',13]]

df = pd.DataFrame(data,columns=['Name','Age'])

print(df)

Its **output** is as follows −

Name Age

0 Alex 10

1 Bob 12

2 Clarke 13

### **Example 15**

import pandas as pd

data = [['Alex',10],['Bob',12],['Clarke',13]]

df = pd.DataFrame(data,columns=['Name','Age'],dtype=float)

print(df)

Its **output** is as follows −

Name Age

0 Alex 10.0

1 Bob 12.0

2 Clarke 13.0

**Note** − Observe, the **dtype** parameter changes the type of Age column to floating point.

## Create a DataFrame from Dict of ndarrays / Lists

All the **ndarrays** must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.

If no index is passed, then by default, index will be range(n), where **n** is the array length.

### **Example 16**

import pandas as pd

data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}

df = pd.DataFrame(data)

print(df)

Its **output** is as follows −

Age Name

0 28 Tom

1 34 Jack

2 29 Steve

3 42 Ricky

**Note** − Observe the values 0,1,2,3. They are the default index assigned to each using the function range(n).

### **Example 17**

Let us now create an indexed DataFrame using arrays.

import pandas as pd

data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}

df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])

print(df)

Its **output** is as follows −

Age Name

rank1 28 Tom

rank2 34 Jack

rank3 29 Steve

rank4 42 Ricky

**Note** − Observe, the **index** parameter assigns an index to each row.

## Create a DataFrame from List of Dicts

List of Dictionaries can be passed as input data to create a DataFrame. The dictionary keys are by default taken as column names.

### **Example 18**

The following example shows how to create a DataFrame by passing a list of dictionaries.

import pandas as pd

data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

df = pd.DataFrame(data)

print(df)

Its **output** is as follows −

a b c

0 1 2 NaN

1 5 10 20.0

**Note** − Observe, NaN (Not a Number) is appended in missing areas.

### **Example 19**

The following example shows how to create a DataFrame by passing a list of dictionaries and the row indices.

import pandas as pd

data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

df = pd.DataFrame(data, index=['first', 'second'])

print(df)

Its **output** is as follows −

a b c

first 1 2 NaN

second 5 10 20.0

### **Example 20**

The following example shows how to create a DataFrame with a list of dictionaries, row indices, and column indices.

import pandas as pd

data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

#With two column indices, values same as dictionary keys

df1 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b'])

#With two column indices with one index with other name

df2 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b1'])

print(df1)

print(df2)

Its **output** is as follows −

#df1 output

a b

first 1 2

second 5 10

#df2 output

a b1

first 1 NaN

second 5 NaN

**Note** − Observe, df2 DataFrame is created with a column index other than the dictionary key; thus, appended the NaN’s in place. Whereas, df1 is created with column indices same as dictionary keys, so NaN’s appended.

## Create a DataFrame from Dict of Series

Dictionary of Series can be passed to form a DataFrame. The resultant index is the union of all the series indexes passed.

### **Example 21**

import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

print(df)

Its **output** is as follows −

one two

a 1.0 1

b 2.0 2

c 3.0 3

d NaN 4

**Note** − Observe, for the series one, there is no label **‘d’** passed, but in the result, for the **d** label, NaN is appended with NaN.

Let us now understand **column selection, addition**, and **deletion** through examples.

## Column Selection

We will understand this by selecting a column from the DataFrame.

### **Example 22**

import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

print(df ['one'])

Its **output** is as follows −

a 1.0

b 2.0

c 3.0

d NaN

Name: one, dtype: float64

## Column Addition

We will understand this by adding a new column to an existing data frame.

### **Example 23**

import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

# Adding a new column to an existing DataFrame object with column label by passing new series

print ("Adding a new column by passing as Series:")

df['three']=pd.Series([10,20,30],index=['a','b','c'])

print(df)

print ("Adding a new column using the existing columns in DataFrame:")

df['four']=df['one']+df['three']

print(df)

Its **output** is as follows −

Adding a new column by passing as Series:

one two three

a 1.0 1 10.0

b 2.0 2 20.0

c 3.0 3 30.0

d NaN 4 NaN

Adding a new column using the existing columns in DataFrame:

one two three four

a 1.0 1 10.0 11.0

b 2.0 2 20.0 22.0

c 3.0 3 30.0 33.0

d NaN 4 NaN NaN

## Column Deletion

Columns can be deleted or popped; let us take an example to understand how.

### **Example 24**

# Using the previous DataFrame, we will delete a column

# using del function

import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),

'three' : pd.Series([10,20,30], index=['a','b','c'])}

df = pd.DataFrame(d)

print ("Our dataframe is:")

print(df)

# using del function

print ("Deleting the first column using DEL function:")

del df['one']

print(df)

# using pop function

print ("Deleting another column using POP function:")

df.pop('two')

print(df)

Its **output** is as follows −

Our dataframe is:

one three two

a 1.0 10.0 1

b 2.0 20.0 2

c 3.0 30.0 3

d NaN NaN 4

Deleting the first column using DEL function:

three two

a 10.0 1

b 20.0 2

c 30.0 3

d NaN 4

Deleting another column using POP function:

three

a 10.0

b 20.0

c 30.0

d NaN

## Row Selection, Addition, and Deletion

We will now understand row selection, addition and deletion through examples. Let us begin with the concept of selection.

### **Selection by Label**

Rows can be selected by passing row label to a **loc** function.

### **Example 25**

import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

print(df.loc['b'])

Its **output** is as follows −

one 2.0

two 2.0

Name: b, dtype: float64

The result is a series with labels as column names of the DataFrame. And, the Name of the series is the label with which it is retrieved.

We can also retrieve the data by simply typing df

df

### **Selection by integer location**

Rows can be selected by passing integer location to an **iloc** function.

### **Example 26**

import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

df.iloc[2]

Its **output** is as follows −

one 3.0

two 3.0

Name: c, dtype: float64

### **Slice Rows**

Multiple rows can be selected using ‘ : ’ operator.

### **Example 27**

import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

df[2:4]

Its **output** is as follows −

one two

c 3.0 3

d NaN 4

### **Addition of Rows**

Add new rows to a DataFrame using the **append** function. This function will append the rows at the end.

### **Example 28**

import pandas as pd

df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])

df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])

df = df.append(df2)

df

Its **output** is as follows −

a b

0 1 2

1 3 4

0 5 6

1 7 8

### **Deletion of Rows**

Use index label to delete or drop rows from a DataFrame. If label is duplicated, then multiple rows will be dropped.

If you observe, in the above example, the labels are duplicate. Let us drop a label and will see how many rows will get dropped.

### **Example 29**

import pandas as pd

df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])

df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])

df = df.append(df2)

# Drop rows with label 0

df = df.drop(0)

df

Its **output** is as follows −

a b

1 3 4

1 7 8

In the above example, two rows were dropped because those two contain the same label 0.

# **Python Pandas - Basic Functionality**

By now, we learnt about the three Pandas DataStructures and how to create them. We will majorly focus on the DataFrame objects because of its importance in the real time data processing and also discuss a few other DataStructures.

## Series Basic Functionality

|  |  |
| --- | --- |
| **Sr.No.** | **Attribute or Method & Description** |
| 1 | **axes**  Returns a list of the row axis labels |
| 2 | **dtype**  Returns the dtype of the object. |
| 3 | **empty**  Returns True if series is empty. |
| 4 | **ndim**  Returns the number of dimensions of the underlying data, by definition 1. |
| 5 | **size**  Returns the number of elements in the underlying data. |
| 6 | **values**  Returns the Series as ndarray. |
| 7 | **head()**  Returns the first n rows. |
| 8 | **tail()**  Returns the last n rows. |

Let us now create a Series and see all the above tabulated attributes operation.

### **Example 30**

import pandas as pd

import numpy as np

#Create a series with 100 random numbers

s = pd.Series(np.random.randn(4))

s

Its **output** is as follows −

0 0.967853

1 -0.148368

2 -1.395906

3 -1.758394

dtype: float64

### **axes**

Returns the list of the labels of the series.

### **Example 31**

import pandas as pd

import numpy as np

#Create a series with 100 random numbers

s = pd.Series(np.random.randn(4))

print ("The axes are:")

s.axes

Its **output** is as follows −

The axes are:

[RangeIndex(start=0, stop=4, step=1)]

The above result is a compact format of a list of values from 0 to 5, i.e., [0,1,2,3,4].

### **empty**

Returns the Boolean value saying whether the Object is empty or not. True indicates that the object is empty.

### **Example 32**

import pandas as pd

import numpy as np

#Create a series with 100 random numbers

s = pd.Series(np.random.randn(4))

print ("Is the Object empty?")

s.empty

Its **output** is as follows −

Is the Object empty?

False

### **ndim**

Returns the number of dimensions of the object. By definition, a Series is a 1D data structure, so it returns

### **Example 33**

import pandas as pd

import numpy as np

#Create a series with 4 random numbers

s = pd.Series(np.random.randn(4))

print s

print ("The dimensions of the object:")

s.ndim

Its **output** is as follows −

0 0.175898

1 0.166197

2 -0.609712

3 -1.377000

dtype: float64

The dimensions of the object:

1

### **size**

Returns the size(length) of the series.

### **Example 34**

import pandas as pd

import numpy as np

#Create a series with 4 random numbers

s = pd.Series(np.random.randn(2))

print(s)

print ("The size of the object:")

s.size

Its **output** is as follows −

0 3.078058

1 -1.207803

dtype: float64

The size of the object:

2

### **values**

Returns the actual data in the series as an array.

### **Example 35**

import pandas as pd

import numpy as np

#Create a series with 4 random numbers

s = pd.Series(np.random.randn(4))

print(s)

print ("The actual data series is:")

s.values

Its **output** is as follows −

0 1.787373

1 -0.605159

2 0.180477

3 -0.140922

dtype: float64

The actual data series is:

[ 1.78737302 -0.60515881 0.18047664 -0.1409218 ]

### **Head & Tail**

To view a small sample of a Series or the DataFrame object, use the head() and the tail() methods.

**head()** returns the first **n** rows(observe the index values). The default number of elements to display is five, but you may pass a custom number.

### **Example 36**

import pandas as pd

import numpy as np

#Create a series with 4 random numbers

s = pd.Series(np.random.randn(4))

print ("The original series is:")

print(s)

print ("The first two rows of the data series:")

s.head(2)

Its **output** is as follows −

The original series is:

0 0.720876

1 -0.765898

2 0.479221

3 -0.139547

dtype: float64

The first two rows of the data series:

0 0.720876

1 -0.765898

dtype: float64

**tail()** returns the last **n** rows(observe the index values). The default number of elements to display is five, but you may pass a custom number.

### **Example 37**

import pandas as pd

import numpy as np

#Create a series with 4 random numbers

s = pd.Series(np.random.randn(4))

print ("The original series is:")

print(s)

print ("The last two rows of the data series:")

s.tail(2)

Its **output** is as follows −

The original series is:

0 -0.655091

1 -0.881407

2 -0.608592

3 -2.341413

dtype: float64

The last two rows of the data series:

2 -0.608592

3 -2.341413

dtype: float64

## DataFrame Basic Functionality

Let us now understand what DataFrame Basic Functionality is. The following tables lists down the important attributes or methods that help in DataFrame Basic Functionality.

|  |  |
| --- | --- |
| **Sr.No.** | **Attribute or Method & Description** |
| 1 | **T**  Transposes rows and columns. |
| 2 | **axes**  Returns a list with the row axis labels and column axis labels as the only members. |
| 3 | **dtypes**  Returns the dtypes in this object. |
| 4 | **empty**  True if NDFrame is entirely empty [no items]; if any of the axes are of length 0. |
| 5 | **ndim**  Number of axes / array dimensions. |
| 6 | **shape**  Returns a tuple representing the dimensionality of the DataFrame. |
| 7 | **size**  Number of elements in the NDFrame. |
| 8 | **values**  Numpy representation of NDFrame. |
| 9 | **head()**  Returns the first n rows. |
| 10 | **tail()**  Returns last n rows. |

Let us now create a DataFrame and see all how the above mentioned attributes operate.

### **Example 38**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame

df = pd.DataFrame(d)

print ("Our data series is:")

df

Its **output** is as follows −

Our data series is:

Age Name Rating

0 25 Tom 4.23

1 26 James 3.24

2 25 Ricky 3.98

3 23 Vin 2.56

4 30 Steve 3.20

5 29 Smith 4.60

6 23 Jack 3.80

### **T (Transpose)**

Returns the transpose of the DataFrame. The rows and columns will interchange.

### **Example 39**

import pandas as pd

import numpy as np

# Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame

df = pd.DataFrame(d)

print ("The transpose of the data series is:")

df.T

Its **output** is as follows −

The transpose of the data series is:

0 1 2 3 4 5 6

Age 25 26 25 23 30 29 23

Name Tom James Ricky Vin Steve Smith Jack

Rating 4.23 3.24 3.98 2.56 3.2 4.6 3.8

### **axes**

Returns the list of row axis labels and column axis labels.

### **Example 40**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame

df = pd.DataFrame(d)

print ("Row axis labels and column axis labels are:")

df.axes

Its **output** is as follows −

Row axis labels and column axis labels are:

[RangeIndex(start=0, stop=7, step=1), Index([u'Age', u'Name', u'Rating'],

dtype='object')]

### **dtypes**

Returns the data type of each column.

### **Example 41**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame

df = pd.DataFrame(d)

print ("The data types of each column are:")

df.dtypes

Its **output** is as follows −

The data types of each column are:

Age int64

Name object

Rating float64

dtype: object

### **empty**

Returns the Boolean value saying whether the Object is empty or not; True indicates that the object is empty.

### **Example 42**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame

df = pd.DataFrame(d)

print ("Is the object empty?")

df.empty

Its **output** is as follows −

Is the object empty?

False

### **ndim**

Returns the number of dimensions of the object. By definition, DataFrame is a 2D object.

### **Example 43**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame

df = pd.DataFrame(d)

print ("Our object is:")

print(df)

print ("The dimension of the object is:")

df.ndim

Its **output** is as follows −

Our object is:

Age Name Rating

0 25 Tom 4.23

1 26 James 3.24

2 25 Ricky 3.98

3 23 Vin 2.56

4 30 Steve 3.20

5 29 Smith 4.60

6 23 Jack 3.80

The dimension of the object is:

2

### **shape**

Returns a tuple representing the dimensionality of the DataFrame. Tuple (a,b), where a represents the number of rows and **b** represents the number of columns.

### **Example 44**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame

df = pd.DataFrame(d)

print ("Our object is:")

print(df)

print ("The shape of the object is:")

df.shape

Its **output** is as follows −

Our object is:

Age Name Rating

0 25 Tom 4.23

1 26 James 3.24

2 25 Ricky 3.98

3 23 Vin 2.56

4 30 Steve 3.20

5 29 Smith 4.60

6 23 Jack 3.80

The shape of the object is:

(7, 3)

### **size**

Returns the number of elements in the DataFrame.

### **Example 45**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame

df = pd.DataFrame(d)

print ("Our object is:")

print(df)

print ("The total number of elements in our object is:")

df.size

Its **output** is as follows −

Our object is:

Age Name Rating

0 25 Tom 4.23

1 26 James 3.24

2 25 Ricky 3.98

3 23 Vin 2.56

4 30 Steve 3.20

5 29 Smith 4.60

6 23 Jack 3.80

The total number of elements in our object is:

21

### **values**

Returns the actual data in the DataFrame as an **NDarray.**

### **Example 46**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame

df = pd.DataFrame(d)

print ("Our object is:")

print(df)

print ("The actual data in our data frame is:")

df.values

Its **output** is as follows −

Our object is:

Age Name Rating

0 25 Tom 4.23

1 26 James 3.24

2 25 Ricky 3.98

3 23 Vin 2.56

4 30 Steve 3.20

5 29 Smith 4.60

6 23 Jack 3.80

The actual data in our data frame is:

[[25 'Tom' 4.23]

[26 'James' 3.24]

[25 'Ricky' 3.98]

[23 'Vin' 2.56]

[30 'Steve' 3.2]

[29 'Smith' 4.6]

[23 'Jack' 3.8]]

### **Head & Tail**

To view a small sample of a DataFrame object, use the **head()** and tail() methods. **head()** returns the first **n** rows (observe the index values). The default number of elements to display is five, but you may pass a custom number.

### **Example 47**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame

df = pd.DataFrame(d)

print ("Our data frame is:")

print(df)

print ("The first two rows of the data frame is:")

df.head(2)

Its **output** is as follows −

Our data frame is:

Age Name Rating

0 25 Tom 4.23

1 26 James 3.24

2 25 Ricky 3.98

3 23 Vin 2.56

4 30 Steve 3.20

5 29 Smith 4.60

6 23 Jack 3.80

The first two rows of the data frame is:

Age Name Rating

0 25 Tom 4.23

1 26 James 3.24

**tail()** returns the last **n** rows (observe the index values). The default number of elements to display is five, but you may pass a custom number.

### **Example 48**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'Age':pd.Series([25,26,25,23,30,29,23]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame

df = pd.DataFrame(d)

print ("Our data frame is:")

print(df)

print ("The last two rows of the data frame is:")

df.tail(2)

Its **output** is as follows −

Our data frame is:

Age Name Rating

0 25 Tom 4.23

1 26 James 3.24

2 25 Ricky 3.98

3 23 Vin 2.56

4 30 Steve 3.20

5 29 Smith 4.60

6 23 Jack 3.80

The last two rows of the data frame is:

Age Name Rating

5 29 Smith 4.6

6 23 Jack 3.8

**Descriptive Statistics**

A large number of methods collectively compute descriptive statistics and other related operations on DataFrame. Most of these are aggregations like **sum(), mean(),** but some of them, like **sumsum()**, produce an object of the same size. Generally speaking, these methods take an **axis** argument, just like *ndarray.{sum, std, ...},* but the axis can be specified by name or integer

* **DataFrame** − “index” (axis=0, default), “columns” (axis=1)

Let us create a DataFrame and use this object throughout this chapter for all the operations.

### **Example 49**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','David','Gasper','Betina','Andres']),

'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])

}

#Create a DataFrame

df = pd.DataFrame(d)

df

Its **output** is as follows −

Age Name Rating

0 25 Tom 4.23

1 26 James 3.24

2 25 Ricky 3.98

3 23 Vin 2.56

4 30 Steve 3.20

5 29 Smith 4.60

6 23 Jack 3.80

7 34 Lee 3.78

8 40 David 2.98

9 30 Gasper 4.80

10 51 Betina 4.10

11 46 Andres 3.65

### **sum()**

Returns the sum of the values for the requested axis. By default, axis is index (axis=0).

### **Example 50**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','David','Gasper','Betina','Andres']),

'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])

}

#Create a DataFrame

df = pd.DataFrame(d)

df.sum()

Its **output** is as follows −

Age 382

Name TomJamesRickyVinSteveSmithJackLeeDavidGasperBe...

Rating 44.92

dtype: object

Each individual column is added individually (Strings are appended).

### **axis=1**

This syntax will give the output as shown below.

### **Example 51**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','David','Gasper','Betina','Andres']),

'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])

}

#Create a DataFrame

df = pd.DataFrame(d)

df.sum(1)

Its **output** is as follows −

0 29.23

1 29.24

2 28.98

3 25.56

4 33.20

5 33.60

6 26.80

7 37.78

8 42.98

9 34.80

10 55.10

11 49.65

dtype: float64

### **mean()**

Returns the average value

### **Example 52**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','David','Gasper','Betina','Andres']),

'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])

}

#Create a DataFrame

df = pd.DataFrame(d)

df.mean()

Its **output** is as follows −

Age 31.833333

Rating 3.743333

dtype: float64

### **std()**

Returns the Bressel standard deviation of the numerical columns.

### **Example 53**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','David','Gasper','Betina','Andres']),

'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])

}

#Create a DataFrame

df = pd.DataFrame(d)

df.std()

Its **output** is as follows −

Age 9.232682

Rating 0.661628

dtype: float64

## Functions & Description

Let us now understand the functions under Descriptive Statistics in Python Pandas. The following table list down the important functions −

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Function** | **Description** |
| 1 | count() | Number of non-null observations |
| 2 | sum() | Sum of values |
| 3 | mean() | Mean of Values |
| 4 | median() | Median of Values |
| 5 | mode() | Mode of values |
| 6 | std() | Standard Deviation of the Values |
| 7 | min() | Minimum Value |
| 8 | max() | Maximum Value |
| 9 | abs() | Absolute Value |
| 10 | prod() | Product of Values |
| 11 | cumsum() | Cumulative Sum |
| 12 | cumprod() | Cumulative Product |

**Note** − Since DataFrame is a Heterogeneous data structure. Generic operations don’t work with all functions.

* Functions like **sum(), cumsum()** work with both numeric and character (or) string data elements without any error. Though **n** practice, character aggregations are never used generally, these functions do not throw any exception.
* Functions like **abs(), cumprod()** throw exception when the DataFrame contains character or string data because such operations cannot be performed.

## Summarizing Data

The **describe()** function computes a summary of statistics pertaining to the DataFrame columns.

### **Example 54**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','David','Gasper','Betina','Andres']),

'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])

}

#Create a DataFrame

df = pd.DataFrame(d)

df.describe()

Its **output** is as follows −

Age Rating

count 12.000000 12.000000

mean 31.833333 3.743333

std 9.232682 0.661628

min 23.000000 2.560000

25% 25.000000 3.230000

50% 29.500000 3.790000

75% 35.500000 4.132500

max 51.000000 4.800000

This function gives the **mean, std** and **IQR** values. And, function excludes the character columns and given summary about numeric columns. **'include'** is the argument which is used to pass necessary information regarding what columns need to be considered for summarizing. Takes the list of values; by default, 'number'.

* **object** − Summarizes String columns
* **number** − Summarizes Numeric columns
* **all** − Summarizes all columns together (Should not pass it as a list value)

Now, use the following statement in the program and check the output –

### **Example 55**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','David','Gasper','Betina','Andres']),

'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])

}

#Create a DataFrame

df = pd.DataFrame(d)

df.describe(include=['object'])

Its **output** is as follows −

Name

count 12

unique 12

top Ricky

freq 1

Now, use the following statement and check the output –

### **Example 56**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','David','Gasper','Betina','Andres']),

'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])

}

#Create a DataFrame

df = pd.DataFrame(d)

df. describe(include='all')

Its **output** is as follows −

Age Name Rating

count 12.000000 12 12.000000

unique NaN 12 NaN

top NaN Ricky NaN

freq NaN 1 NaN

mean 31.833333 NaN 3.743333

std 9.232682 NaN 0.661628

min 23.000000 NaN 2.560000

25% 25.000000 NaN 3.230000

50% 29.500000 NaN 3.790000

75% 35.500000 NaN 4.132500

max 51.000000 NaN 4.800000

# **Statistical Functions**

Statistical methods help in the understanding and analyzing the behavior of data. We will now learn a few statistical functions, which we can apply on Pandas objects.

## Percent\_change

Series, DatFrames and Panel, all have the function **pct\_change()**. This function compares every element with its prior element and computes the change percentage.

### **Example 57**

import pandas as pd

import numpy as np

s = pd.Series([1,2,3,4,5,4])

print(s.pct\_change())

df = pd.DataFrame(np.random.randn(5, 2))

df.pct\_change()

Its **output** is as follows −

0 NaN

1 1.000000

2 0.500000

3 0.333333

4 0.250000

5 -0.200000

dtype: float64

0 1

0 NaN NaN

1 -15.151902 0.174730

2 -0.746374 -1.449088

3 -3.582229 -3.165836

4 15.601150 -1.860434

By default, the **pct\_change()** operates on columns; if you want to apply the same row wise, then use **axis=1()** argument.

## Covariance

Covariance is applied on series data. The Series object has a method cov to compute covariance between series objects. NA will be excluded automatically.

### **Example 58**

### **Cov Series**

import pandas as pd

import numpy as np

s1 = pd.Series(np.random.randn(10))

s2 = pd.Series(np.random.randn(10))

s1.cov(s2)

Its **output** is as follows −

-0.12978405324

Covariance method when applied on a DataFrame, computes **cov** between all the columns.

## Correlation

Correlation shows the linear relationship between any two array of values (series). There are multiple methods to compute the correlation like pearson(default), spearman and kendall.

### **Example 59**

import pandas as pd

import numpy as np

s1 = pd.Series(np.random.randn(10))

s2 = pd.Series(np.random.randn(10))

s1.corr(s2)

# **Statistics - Measuring Central Tendency**

Mathematically central tendency means measuring the center or distribution of location of values of a data set. It gives an idea of the average value of the data in the data set and also an indication of how widely the values are spread in the data set. That in turn helps in evaluating the chances of a new input fitting into the existing data set and hence probability of success.

There are three main measures of central tendency which can be calculated using the methods in pandas python library.

* Mean - It is the Average value of the data which is a division of sum of the values with the number of values.
* Median - It is the middle value in distribution when the values are arranged in ascending or descending order.
* Mode - It is the most commonly occurring value in a distribution.

## Calculating Mean and Median

The pandas functions can be directly used to calculate these values.

### **Example 60**

import pandas as pd

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','Chanchal','Gasper','Naviya','Andres']),

'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame

df = pd.DataFrame(d)

print ("Mean Values in the Distribution")

print(df.mean())

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Median Values in the Distribution")

print(df.median())

Its **output** is as follows −

Mean Values in the Distribution

Age 31.833333

Rating 3.743333

dtype: float64

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Median Values in the Distribution

Age 29.50

Rating 3.79

dtype: float64

## Calculating Mode

Mode may or may not be available in a distribution depending on whether the data is continous or whether there are values which has maximum frquency. We take a simple distribution below to find out the mode. Here we have a value which has maximum frequency in the distribution.

### **Example 61**

import pandas as pd

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','Chanchal','Gasper','Naviya','Andres']),

'Age':pd.Series([25,26,25,23,30,25,23,34,40,30,25,46])}

#Create a DataFrame

df = pd.DataFrame(d)

print(df.mode())

**Measuring Variance**

In statistics, variance is a measure of how far a value in a data set lies from the mean value. In other words, it indicates how dispersed the values are. It is measured by using standard deviation. The other method commonly used is skewness.

Both of these are calculated by using functions available in pandas library.

## Measuring Standard Deviation

Standard deviation is square root of variance. variance is the average of squared difference of values in a data set from the mean value. In python we calculate this value by using the function std() from pandas library.

### **Example 62**

import pandas as pd

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','Chanchal','Gasper','Naviya','Andres']),

'Age':pd.Series([25,26,25,23,30,25,23,34,40,30,25,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame

df = pd.DataFrame(d)

# Calculate the standard deviation

df.std()

Its **output** is as follows −

Age 7.265527

Rating 0.661628

dtype: float64

## Measuring Skewness

It used to determine whether the data is symmetric or skewed. If the index is between -1 and 1, then the distribution is symmetric. If the index is no more than -1 then it is skewed to the left and if it is at least 1, then it is skewed to the right

### **Example 63**

import pandas as pd

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',

'Lee','Chanchal','Gasper','Naviya','Andres']),

'Age':pd.Series([25,26,25,23,30,25,23,34,40,30,25,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame

df = pd.DataFrame(d)

df.skew()

Its **output** is as follows −

Age 1.443490

Rating -0.153629

dtype: float64

So the distribution of age rating is symmetric while the distribution of age is skewed to the right.

# **Python - Processing CSV Data**

Reading data from CSV(comma separated values) is a fundamental necessity in Data Science. Often, we get data from various sources which can get exported to CSV format so that they can be used by other systems. The Panadas library provides features using which we can read the CSV file in full as well as in parts for only a selected group of columns and rows.

## Input as CSV File

The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named **input1.csv**.

You can create this file using windows notepad by copying and pasting this data. Save the file as **input1.csv** using the save As All files(\*.\*) option in notepad.

### **Example 64**

id,name,salary,start\_date,dept

1,Rick,623.3,2012-01-01,IT

2,Dan,515.2,2013-09-23,Operations

3,Tusar,611,2014-11-15,IT

4,Ryan,729,2014-05-11,HR

5,Gary,843.25,2015-03-27,Finance

6,Rasmi,578,2013-05-21,IT

7,Pranab,632.8,2013-07-30,Operations

8,Guru,722.5,2014-06-17,Finance

## Reading a CSV File

The **read\_csv** function of the pandas library is used read the content of a CSV file into the python environment as a pandas DataFrame. The function can read the files from the OS by using proper path to the file.

### **Example 65**

import pandas as pd

data = pd.read\_csv('input1.csv')

print (data)

When we execute the above code, it produces the following result. Please note how an additional column starting with zero as a index has been created by the function.

id name salary start\_date dept

0 1 Rick 623.30 2012-01-01 IT

1 2 Dan 515.20 2013-09-23 Operations

2 3 Tusar 611.00 2014-11-15 IT

3 4 Ryan 729.00 2014-05-11 HR

4 5 Gary 843.25 2015-03-27 Finance

5 6 Rasmi 578.00 2013-05-21 IT

6 7 Pranab 632.80 2013-07-30 Operations

7 8 Guru 722.50 2014-06-17 Finance

## Reading Specific Rows

The **read\_csv** function of the pandas library can also be used to read some specific rows for a given column. We slice the result from the read\_csv function using the code shown below for first 5 rows for the column named salary.

### **Example 66**

import pandas as pd

data = pd.read\_csv('input1.csv')

# Slice the result for first 5 rows

print (data[0:5]['salary'])

When we execute the above code, it produces the following result.

0 623.30

1 515.20

2 611.00

3 729.00

4 843.25

Name: salary, dtype: float64

## Reading Specific Columns

The **read\_csv** function of the pandas library can also be used to read some specific columns. We use the multi-axes indexing method called **.loc()** for this purpose. We choose to display the salary and name column for all the rows.

### **Example 67**

import pandas as pd

data = pd.read\_csv('input1.csv')

# Use the multi-axes indexing funtion

print (data.loc[:,['salary','name']])

When we execute the above code, it produces the following result.

salary name

0 623.30 Rick

1 515.20 Dan

2 611.00 Tusar

3 729.00 Ryan

4 843.25 Gary

5 578.00 Rasmi

6 632.80 Pranab

7 722.50 Guru

## Reading Specific Columns and Rows

The **read\_csv** function of the pandas library can also be used to read some specific columns and specific rows. We use the multi-axes indexing method called **.loc()** for this purpose. We choose to display the salary and name column for some of the rows.

### **Example 68**

import pandas as pd

data = pd.read\_csv('input1.csv')

# Use the multi-axes indexing funtion

print (data.loc[[1,3,5],['salary','name']])

When we execute the above code, it produces the following result.

salary name

1 515.2 Dan

3 729.0 Ryan

5 578.0 Rasmi

## Reading Specific Columns for a Range of Rows

The **read\_csv** function of the pandas library can also be used to read some specific columns and a range of rows. We use the multi-axes indexing method called **.loc()** for this purpose. We choose to display the salary and name column for some of the rows.

### **Example 69**

import pandas as pd

data = pd.read\_csv('path/input.csv')

# Use the multi-axes indexing funtion

print (data.loc[2:6,['salary','name']])

When we execute the above code, it produces the following result.

salary name

2 611.00 Tusar

3 729.00 Ryan

4 843.25 Gary

5 578.00 Rasmi

6 632.80 Pranab

# **Python - Processing XLS Data**

Microsoft Excel is a very widely used spread sheet program. Its user friendliness and appealing features makes it a very frequently used tool in Data Science. The Panadas library provides features using which we can read the Excel file in full as well as in parts for only a selected group of Data. We can also read an Excel file with multiple sheets in it. We use the **read\_excel** function to read the data from it.

## Input as Excel File

We Create an excel file with multiple sheets in the windows OS. The Data in the different sheets is as shown below.

You can create this file using the Excel Program in windows OS. Save the file as **input2.xlsx**.

### **Example 70**

# Data in Sheet1

id,name,salary,start\_date,dept

1,Rick,623.3,2012-01-01,IT

2,Dan,515.2,2013-09-23,Operations

3,Tusar,611,2014-11-15,IT

4,Ryan,729,2014-05-11,HR

5,Gary,843.25,2015-03-27,Finance

6,Rasmi,578,2013-05-21,IT

7,Pranab,632.8,2013-07-30,Operations

8,Guru,722.5,2014-06-17,Finance

## Reading an Excel File

The **read\_excel** function of the pandas library is used read the content of an Excel file into the python environment as a pandas DataFrame. The function can read the files from the OS by using proper path to the file. By default, the function will read Sheet1.

### **Example 71**

import pandas as pd

data = pd.read\_excel('input2.xlsx')

print (data)

When we execute the above code, it produces the following result. Please note how an additional column starting with zero as a index has been created by the function.

id name salary start\_date dept

0 1 Rick 623.30 2012-01-01 IT

1 2 Dan 515.20 2013-09-23 Operations

2 3 Tusar 611.00 2014-11-15 IT

3 4 Ryan 729.00 2014-05-11 HR

4 5 Gary 843.25 2015-03-27 Finance

5 6 Rasmi 578.00 2013-05-21 IT

6 7 Pranab 632.80 2013-07-30 Operations

7 8 Guru 722.50 2014-06-17 Finance

## Reading Specific Columns and Rows

Similar to what we have already seen in the previous chapter to read the CSV file, the **read\_excel** function of the pandas library can also be used to read some specific columns and specific rows. We use the multi-axes indexing method called **.loc()** for this purpose. We choose to display the salary and name column for some of the rows.

### **Example 72**

import pandas as pd

data = pd.read\_excel('input2.xlsx')

# Use the multi-axes indexing funtion

print (data.loc[[1,3,5],['salary','name']])

When we execute the above code, it produces the following result.

salary name

1 515.2 Dan

3 729.0 Ryan

5 578.0 Rasmi

# **Python Pandas - Visualization**

### **Basic Plotting: plot**

This functionality on Series and DataFrame is just a simple wrapper around the **matplotlib** **libraries plot()** method.

### **Example 73**

import pandas as pd

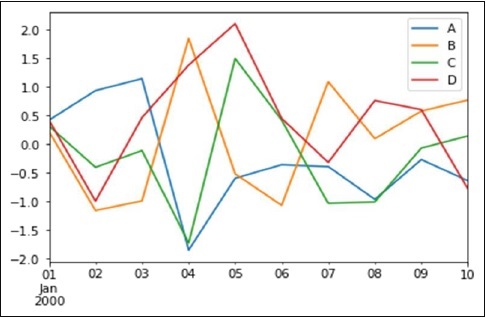
import numpy as np

df = pd.DataFrame(np.random.randn(10,4),index=pd.date\_range('1/1/2000',

periods=10), columns=list('ABCD'))

df.plot()

Its **output** is as follows −



If the index consists of dates, it calls **gct().autofmt\_xdate()** to format the x-axis as shown in the above illustration.

We can plot one column versus another using the **x** and **y** keywords.

Plotting methods allow a handful of plot styles other than the default line plot. These methods can be provided as the kind keyword argument to **plot()**. These include −

* bar or barh for bar plots
* hist for histogram
* box for boxplot
* 'area' for area plots
* 'scatter' for scatter plots

## Bar Plot

Let us now see what a Bar Plot is by creating one. A bar plot can be created in the following way –

### **Example 74**

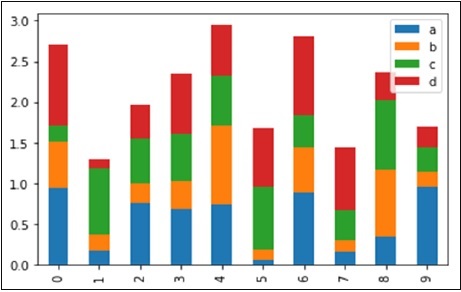
import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])

df.plot.bar()

Its **output** is as follows −



To produce a stacked bar plot, **pass stacked=True** –

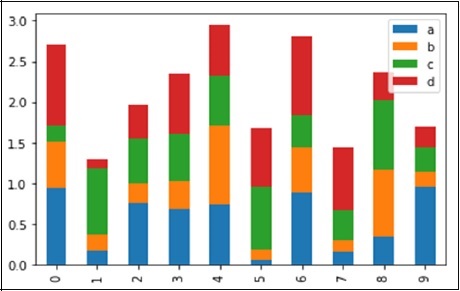
### **Example 75**

import pandas as pd

df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])

df.plot.bar(stacked=True)

Its **output** is as follows −



To get horizontal bar plots, use the **barh** method –

### **Example 76**

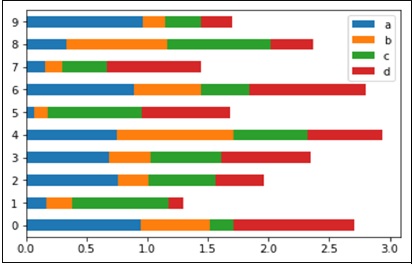
import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])

df.plot.barh(stacked=True)

Its **output** is as follows −



## Histograms

Histograms can be plotted using the **plot.hist()** method. We can specify number of bins.

### **Example 77**

import pandas as pd

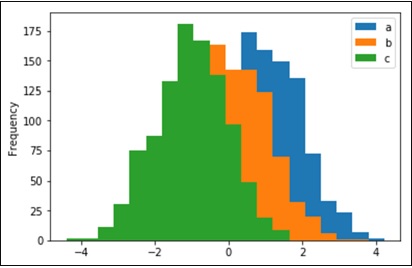
import numpy as np

df = pd.DataFrame({'a':np.random.randn(1000)+1,'b':np.random.randn(1000),'c':

np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])

df.plot.hist(bins=20)

Its **output** is as follows −



To plot different histograms for each column, use the following code –

### **Example 78**

import pandas as pd

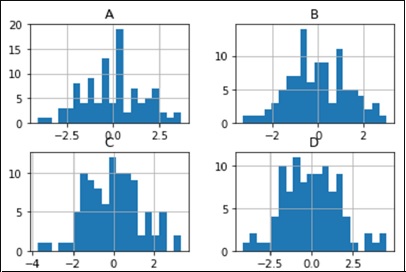
import numpy as np

df=pd.DataFrame({'a':np.random.randn(1000)+1,'b':np.random.randn(1000),'c':

np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])

df.hist(bins=20)

Its **output** is as follows −



## Box Plots

Boxplot can be drawn calling **Series.box.plot()** and **DataFrame.box.plot()**, or **DataFrame.boxplot()** to visualize the distribution of values within each column.

For instance, here is a boxplot representing five trials of 10 observations of a uniform random variable on [0,1).

### **Example 79**

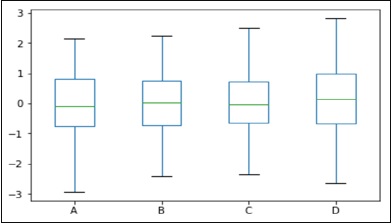
import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D', 'E'])

df.plot.box()

Its **output** is as follows −



## Area Plot

Area plot can be created using the **Series.plot.area()** or the **DataFrame.plot.area()** methods.

### **Example 80**

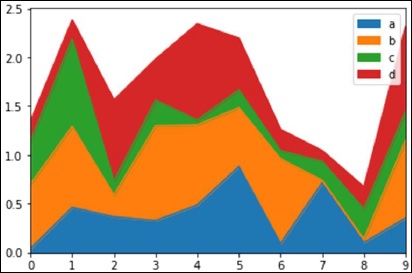
import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])

df.plot.area()

Its **output** is as follows −



## Scatter Plot

Scatter plot can be created using the **DataFrame.plot.scatter()** methods.

### **Example 81**

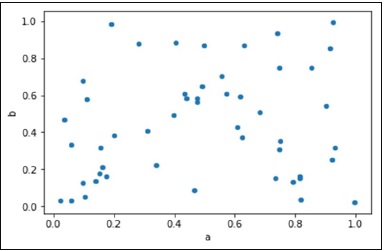
import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.rand(50, 4), columns=['a', 'b', 'c', 'd'])

df.plot.scatter(x='a', y='b')

Its **output** is as follows −



## Pie Chart

Pie chart can be created using the **DataFrame.plot.pie()** method.

### **Example 82**

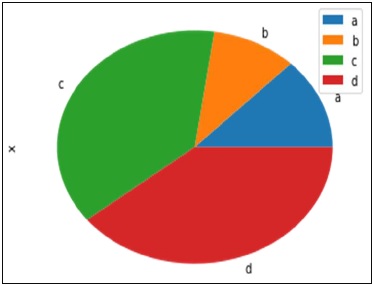
import pandas as pd

import numpy as np

df = pd.DataFrame(3 \* np.random.rand(4), index=['a', 'b', 'c', 'd'], columns=['x'])

df.plot.pie(subplots=True)

Its **output** is as follows −



# **Python - Matplotlib**

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.

Conventionally, the package is imported into the Python script by adding the following statement −

from matplotlib import pyplot as plt

## Matplotlib Example

The following script produces the **sine wave plot** using matplotlib.

### **Example 83**

import numpy as np

import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve

x = np.arange(0, 3 \* np.pi, 0.1)

y = np.sin(x)

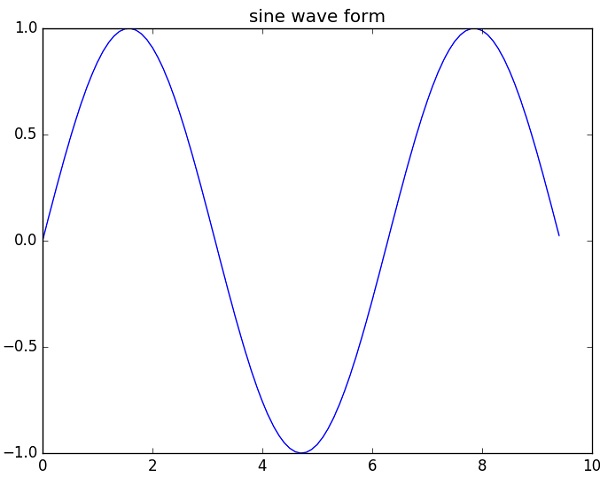
plt.title("sine wave form")

# Plot the points using matplotlib

plt.plot(x, y)

plt.show()

Its **output** is as follows −



**Python Chart Properties**

Python has excellent libraries for data visualization. A combination of **Pandas**, **numpy** and **matplotlib** can help in creating in nearly all types of visualizations charts. In this chapter we will get started with looking at some simple chart and the various properties of the chart.

## Creating a Chart

We use numpy library to create the required numbers to be mapped for creating the chart and the pyplot method in matplotlib to draws the actual chart.

### **Example 84**

import numpy as np

import matplotlib.pyplot as plt

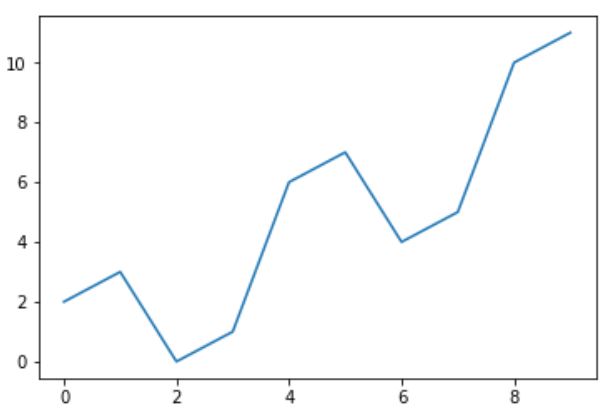
x = np.arange(0,10)

y = x ^ 2

#Simple Plot

plt.plot(x,y)

Its **output** is as follows −



## Labling the Axes

We can apply labels to the axes as well as a title for the chart using appropriate methods from the library as shown below.

### **Example 85**

import numpy as np

import matplotlib.pyplot as plt

x = np.arange(0,10)

y = x ^ 2

#Labeling the Axes and Title

plt.title("Graph Drawing")

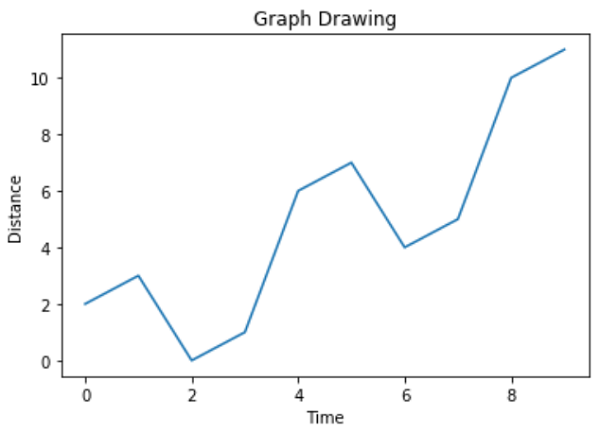
plt.xlabel("Time")

plt.ylabel("Distance")

#Simple Plot

plt.plot(x,y)

Its **output** is as follows −



## Formatting Line type and Colour

The style as well as colour for the line in the chart can be specified using appropriate methods from the library as shown below.

### **Example 86**

import numpy as np

import matplotlib.pyplot as plt

x = np.arange(0,10)

y = x ^ 2

#Labeling the Axes and Title

plt.title("Graph Drawing")

plt.xlabel("Time")

plt.ylabel("Distance")

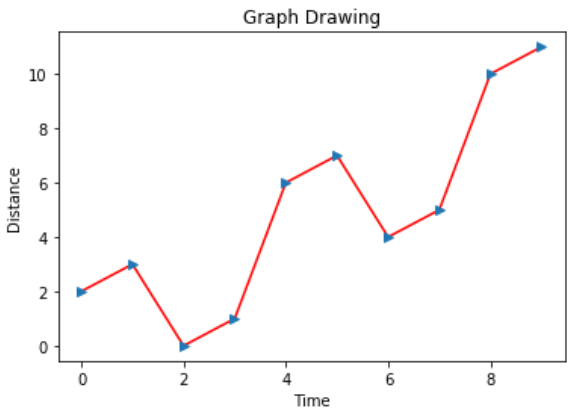
# Formatting the line colors

plt.plot(x,y,'r')

# Formatting the line type

plt.plot(x,y,'>')

Its **output** is as follows −



## Saving the Chart File

The chart can be saved in different image file formats using appropriate methods from the library as shown below.

### **Example 87**

import numpy as np

import matplotlib.pyplot as plt

x = np.arange(0,10)

y = x ^ 2

#Labeling the Axes and Title

plt.title("Graph Drawing")

plt.xlabel("Time")

plt.ylabel("Distance")

# Formatting the line colors

plt.plot(x,y,'r')

# Formatting the line type

plt.plot(x,y,'>')

# save in pdf formats

plt.savefig('timevsdist.pdf', format='pdf')

The above code creates the pdf file in the default path of the python environment.

**Box Plot**

Boxplots are a measure of how well distributed the data in a data set is. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

## Drawing a Box Plot

Boxplot can be drawn calling Series.box.plot() and DataFrame.box.plot(), or DataFrame.boxplot() to visualize the distribution of values within each column.

For instance, here is a boxplot representing five trials of 10 observations of a uniform random variable on [0,1).

### **Example 88**

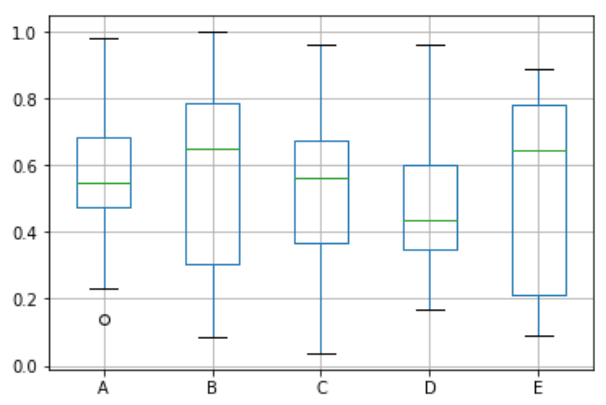
import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D', 'E'])

df.plot.box(grid='True')

Its **output** is as follows –



**HEATMAP**

A heatmap contains values representing various shades of the same colour for each value to be plotted. Usually the darker shades of the chart represent higher values than the lighter shade. For a very different value a completely different colour can also be used.

The below example is a two-dimensional plot of values which are mapped to the indices and columns of the chart.

### **Example 89**

from pandas import DataFrame

import matplotlib.pyplot as plt

data=[{2,3,4,1},{6,3,5,2},{6,3,5,4},{3,7,5,4},{2,8,1,5}]

Index= ['I1', 'I2','I3','I4','I5']

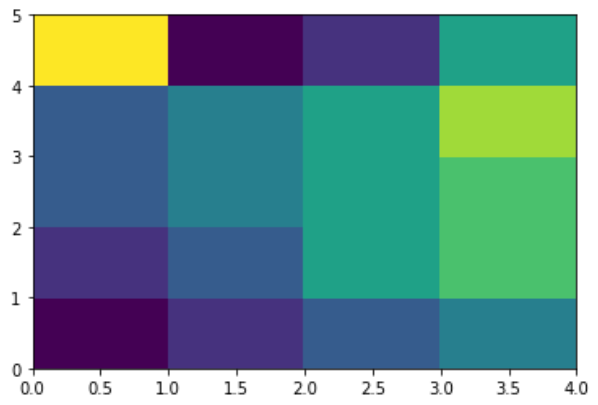
Cols = ['C1', 'C2', 'C3','C4']

df = DataFrame(data, index=Index, columns=Cols)

plt.pcolor(df)

plt.show()

Its **output** is as follows −



**Scatterplots**

**Scatterplots** show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

## Drawing a Scatter Plot

Scatter plot can be created using the DataFrame.plot.scatter() methods.

### **Example 90**

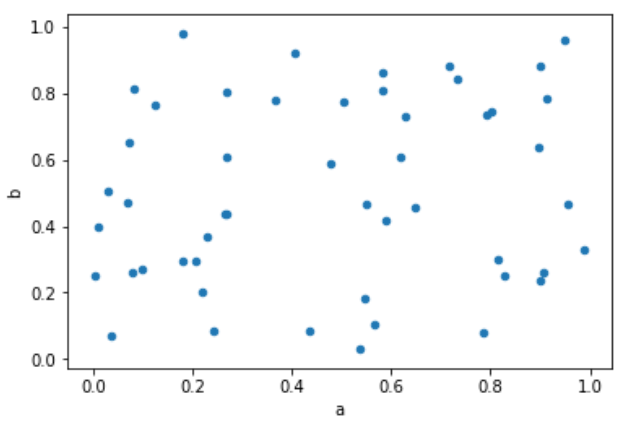
import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.rand(50, 4), columns=['a', 'b', 'c', 'd'])

df.plot.scatter(x='a', y='b')

Its **output** is as follows −



Bubble charts display data as a cluster of circles. The required data to create bubble chart needs to have the xy coordinates, size of the bubble and the colour of the bubbles. The colours can be supplied by the library itself.

## Drawing a Bubble Chart

Bubble chart can be created using the DataFrame.plot.scatter() methods.

### **Example 91**

import matplotlib.pyplot as plt

import numpy as np

# create data

x = np.random.rand(40)

y = np.random.rand(40)

z = np.random.rand(40)

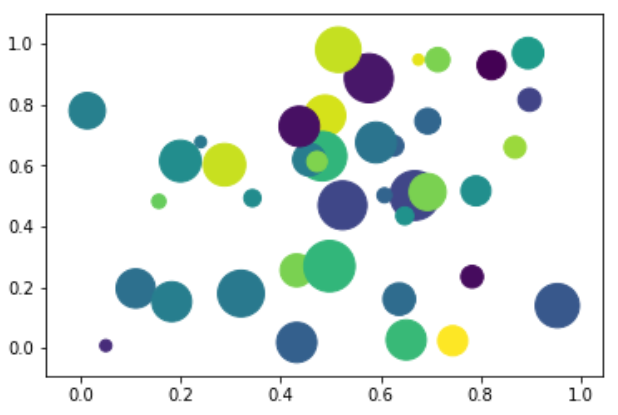
colors = np.random.rand(40)

# use the scatter function

plt.scatter(x, y, s=z\*1000,c=colors)

plt.show()

Its **output** is as follows −



**3D Plot**

Python is also capable of creating 3d charts. It involves adding a subplot to an existing two-dimensional plot and assigning the projection parameter as 3d.

## Drawing a 3D Plot

3dPlot is drawn by mpl\_toolkits.mplot3d to add a subplot to an existing 2d plot.

### **Example 92**

from mpl\_toolkits.mplot3d import axes3d

import matplotlib.pyplot as plt

chart = plt.figure()

chart3d = chart.add\_subplot(111, projection='3d')

# Create some test data.

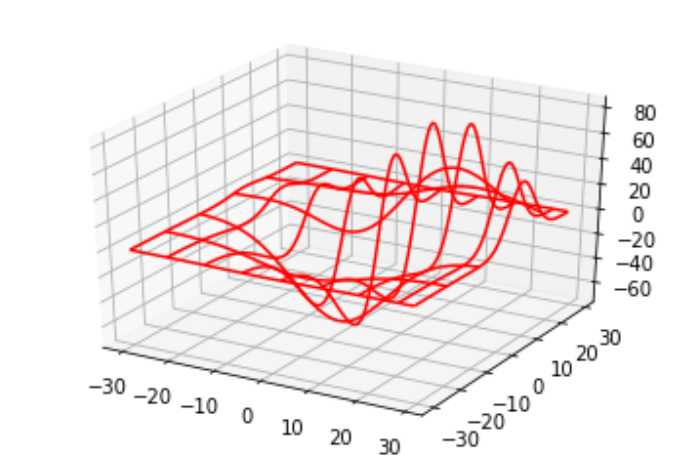
X, Y, Z = axes3d.get\_test\_data(0.08)

# Plot a wireframe.

chart3d.plot\_wireframe(X, Y, Z, color='r',rstride=15, cstride=10)

plt.show()

Its **output** is as follows −



**Time series**

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year.

In the below example we take the value of stock prices every day for a quarter for a particular stock symbol. We capture these values as a csv file and then organize them to a dataframe using pandas library. We then set the date field as index of the dataframe by recreating the additional Valuedate column as index and deleting the old valuedate column.

Sample Data

Below is the sample data for the price of the stock on different days of a given quarter. The data is saved in a file named as stock.csv

ValueDate Price

01-01-2018, 1042.05

02-01-2018, 1033.55

03-01-2018, 1029.7

04-01-2018, 1021.3

05-01-2018, 1015.4

...

...

...

...

23-03-2018, 1161.3

26-03-2018, 1167.6

27-03-2018, 1155.25

28-03-2018, 1154

**Student Performance Analytics**

**Instructions**

1. Download **StudentsPerformance.csv** from moodle into <your> folder
2. Launch jupyter from <your> folder
3. Copy and Paste or type the codes available in the below boxes and run to see the results
4. Copy the codes available in the box only , for practices.
5. Sometime you may have to play run button twice or thrice to see the results.

**Student Performance Analysis**

import numpy as np *# linear algebra*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

import matplotlib.pyplot as plt

import seaborn as sns

In [2]:

%matplotlib inline

plt.style.use('fivethirtyeight')

In [3]:

*#Reading the Data*

df = pd.read\_csv("StudentsPerformance.csv")

df.head()

In [4]:

df.columns

)

In [5]:

df.shape

In [6]:

df.info()

<

: 62.6+ KB

In [7]:

*#Checking for Nulls*

df.isnull().sum()

int64

No Nulls in any column.

In [8]:

df.describe()

In [9]:

*#Adding a column of total score in the dataframe*

df['total score'] = df['math score'] + df['reading score'] + df['writing score']

df.head()

In [10]:

*#Checking data types of the columns*

df.dtypes

: object

In [11]:

*#Correlation among numeric columns*

df.corr()

This shows the students scoring good in one subject, score good overall (in all the subjects)

In [12]:

sns.heatmap(df.corr())

plt.title('Covariance Plot')

In [13]:

*Scatter plot of scores in different subjects to visualize correlation among them*

plt.figure(figsize = (20,8))

plt.subplot(1,3,1)

plt.scatter(df['math score'],df['reading score'])

plt.title('Math scores vs Reading scores')

plt.xlabel('Math score')

plt.ylabel('Reading score')

plt.subplot(1,3,2)

plt.scatter(df['reading score'],df['writing score'])

plt.title('Reading scores vs Writing scores')

plt.xlabel('Reading score')

plt.ylabel('Writing score')

plt.subplot(1,3,3)

plt.scatter(df['writing score'],df['math score'])

plt.title('Writing scores vs Math scores')

plt.xlabel('Writing score')

plt.ylabel('Math score')

plt.show()

The plot shows the same observation as seen from calculating correlation among scores in different subjects. Also reading vs writing scatter plot is much more dense than reading vs maths or writing vs maths which shows higher correalation among them - students better in reading(or writing) are better in writing(or reading).

In [14]:

*#Same plot as above but with regression line using Seaborn*

plt.figure(figsize = (20,8))

plt.subplot(1,3,1)

sns.regplot(x = 'math score', y = 'reading score',data = df)

plt.title('Math scores vs Reading scores')

plt.xlabel('Math score')

plt.ylabel('Reading score')

plt.subplot(1,3,2)

sns.regplot(x = 'reading score', y = 'writing score',data = df)

plt.title('Reading scores vs Writing scores')

plt.xlabel('Reading score')

plt.ylabel('Writing score')

plt.subplot(1,3,3)

sns.regplot(x = 'writing score', y = 'math score',data = df)

plt.title('Writing scores vs Math scores')

plt.xlabel('Writing score')

plt.ylabel('Math score')

plt.show()

In [15]:

*#Scatter plot between individiual subject score and total score with using Seaborn*

plt.figure(figsize = (20,8))

plt.subplot(1,3,1)

sns.regplot(x = 'math score', y = 'total score',data = df)

plt.title('Math scores vs Total scores')

plt.xlabel('Math score')

plt.ylabel('Total score')

plt.subplot(1,3,2)

sns.regplot(x = 'reading score', y = 'total score',data = df)

plt.title('Reading scores vs Total scores')

plt.xlabel('Reading score')

plt.ylabel('Total score')

plt.subplot(1,3,3)

sns.regplot(x = 'writing score', y = 'total score',data = df)

plt.title('Writing scores vs Total scores')

plt.xlabel('Writing score')

plt.ylabel('Total score')

plt.show()

In [16]:

*#Number of Students against Scores in all the 3 subjects*

plt.hist([df['math score'],df['reading score'],df['writing score']], color=['red', 'yellow', 'blue'])

plt.title('Number of Students against Scores')

plt.xlabel('Score')

plt.ylabel('Number of Students')

plt.legend(['Math', 'Reading', 'Writing'])

plt.show()

Number of students with higher score in mathematics have dropped below than reading and writing which may show that is easier to get a higher score in reading and writing than mathematics.

In [17]:

y = ['Math','Reading','Writing']

width = [df['math score'].mean(),df['reading score'].mean(),df['writing score'].mean()]

plt.figure(figsize = (12,2))

plt.barh(y = y,

width = width)

plt.title('Average Scores')

plt.xlabel('Average Score')

plt.ylabel('Subjects')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(round(v,2)), color='blue', va='center', fontweight='bold')

plt.show()

Average score is highest in reading and lowest in mathematics.

Analyzing Scores based on Gender

In [18]:

df\_gender = df.groupby('gender')

In [19]:

*#Number of Females and Males*

y = df\_gender['gender'].count().keys()

width = df\_gender['gender'].count()

plt.figure(figsize = (12,2))

plt.barh(y = y,

width = width)

plt.title('No. of Females and Males')

plt.xlabel('Count')

plt.ylabel('Gender')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

In [20]:

*#Average scores of Females and Males*

y = df\_gender['total score'].mean().keys()

width = df\_gender['total score'].mean()

plt.figure(figsize = (12,2))

plt.barh(y = y,

width = width)

plt.title('Av score of Female and Males')

plt.xlabel('Av. total score out of 300')

plt.ylabel('Gender')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(round(v,2)), color='blue', va='center', fontweight='bold')

plt.show()

Female students have performed better than Male students.

In [21]:

sns.boxplot(x="gender", y="total score", data=df)

In [22]:

sns.swarmplot(x='gender',y='total score',data=df)

sns.violinplot(x='gender',y='total score',data=df, inner=None,color='lightgray')

In [23]:

*#Average scores in individual subjects*

x = df\_gender['gender'].count().keys()

plt.figure(figsize=(12,4))

plt.subplot(1,3,1)

height = df\_gender['math score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Math Score')

plt.xlabel('Gender')

plt.ylabel('Av. Math Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha='center', fontweight='bold')

plt.subplot(1,3,2)

height = df\_gender['reading score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Reading Score')

plt.xlabel('Gender')

plt.ylabel('Av. Reading Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha='center', fontweight='bold')

plt.subplot(1,3,3)

height = df\_gender['writing score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Writing Score')

plt.xlabel('Gender')

plt.ylabel('Av. Writing Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha = 'center', fontweight='bold')

plt.show()

Male students performed better in mathematics than female students but worse in both reading and writing.

Analyzing scores based on Session

In [24]:

df\_session = df.groupby('session')

In [25]:

*#Count of studensts belonging to different session groups*

y = df\_session['session'].count().keys()

width = df\_session['session'].count()

plt.figure(figsize = (12,4))

plt.barh(y = y,

width = width)

plt.title('No. of Students of Different sessions')

plt.xlabel('Count')

plt.ylabel('session')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

In [26]:

*#Average score of students in different session groups*

y = df\_session['total score'].mean().keys()

width = df\_session['total score'].mean()

plt.figure(figsize = (12,4))

plt.barh(y = y,

width = width)

plt.title('Mean Scores of Students of Different Session Groups')

plt.xlabel('Mean score')

plt.ylabel('Session/Ehtnicity')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(round(v,2)), color='blue', va='center', fontweight='bold')

plt.show()

Students belonging to group E performed the best and students of group A performed the worst.

In [27]:

plt.figure(figsize = (12,4))

sns.boxplot(x="session", y="total score", data=df)

In [28]:

plt.figure(figsize=(12, 6))

sns.swarmplot(x='session',y='total score',data=df, hue = 'gender')

sns.violinplot(x='session',y='total score',data=df, inner=None,color='lightgray')

In [29]:

*#Average scores in individual subject based on Session*

x = df\_session['total score'].mean().keys()

plt.figure(figsize=(20,5))

plt.subplot(1,3,1)

height = df\_session['math score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Math Score')

plt.xlabel('session')

plt.ylabel('Av. Math Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha='center', fontweight='bold')

plt.subplot(1,3,2)

height = df\_session['reading score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Reading Score')

plt.xlabel('session')

plt.ylabel('Av. Reading Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha='center', fontweight='bold')

plt.subplot(1,3,3)

height = df\_session['writing score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Writing Score')

plt.xlabel('session')

plt.ylabel('Av. Writing Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha = 'center', fontweight='bold')

plt.show()

Students belonging to group E have performed best in all subjects with students of group A performing the worst.

Analyzing Scores based on Parental Level of Education

In [30]:

df\_parental = df.groupby('parental level of education')

In [31]:

*#Counting students based on the parental level of education*

y = df\_parental['parental level of education'].count().keys()

width = df\_parental['parental level of education'].count()

plt.figure(figsize = (12,4))

plt.barh(y = y,

width = width)

plt.title('No. of Students based on parental level of education')

plt.xlabel('Count')

plt.ylabel('Parental level of education')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

In [32]:

*#Mean score of students based on the parental level of education*

y = df\_parental['total score'].mean().keys()

width = df\_parental['total score'].mean()

plt.figure(figsize = (12,4))

plt.barh(y = y,

width = width)

plt.title('Mean score of Students based on parental level of education')

plt.xlabel('Mean total score')

plt.ylabel('Parental levelof education')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(round(v,2)), color='blue', va='center', fontweight='bold')

plt.show()

This shows that education level of parents effected the students performance.

In [33]:

plt.figure(figsize = (12,4))

sns.boxplot(x="parental level of education", y="total score", data=df)

In [34]:

plt.figure(figsize=(13, 6))

sns.swarmplot(x='parental level of education',y='total score',data=df, hue = 'gender')

sns.violinplot(x='parental level of education',y='total score',data=df, inner=None,color='lightgray')

In [35]:

*#Mean scores of students in individual subjects based on parental level of education*

x = df\_parental['total score'].mean().keys()

plt.figure(figsize=(20,18))

plt.subplot(3,1,1)

height = df\_parental['math score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Math Score')

plt.xlabel('parental level if education')

plt.ylabel('Av. Math Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha='center', fontweight='bold')

plt.subplot(3,1,2)

height = df\_parental['reading score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Reading Score')

plt.xlabel('parental level if education')

plt.ylabel('Av. Reading Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha='center', fontweight='bold')

plt.subplot(3,1,3)

height = df\_parental['writing score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Writing Score')

plt.xlabel('parental level if education')

plt.ylabel('Av. Writing Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha = 'center', fontweight='bold')

plt.show()

Analyzing scores based on finance (sponsored /self-financed)

In [36]:

df\_finance = df.groupby('finance')

In [37]:

*# Counting students according to finance type*

y = df\_finance['finance'].count().keys()

width = df\_finance['finance'].count()

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of Students acc. to finance type')

plt.xlabel('Count')

plt.ylabel('finance')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

In [38]:

*#Mean score of students according to finance type*

y = df\_finance['total score'].mean().keys()

width = df\_finance['total score'].mean()

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('Mean total score of Students in Different Finace categories')

plt.xlabel('Mean total score')

plt.ylabel('finance')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(round(v,2)), color='blue', va='center', fontweight='bold')

plt.show()

Students taking sponsored have performed significantly better than students taking self financed.

In [39]:

plt.figure(figsize = (8,4))

sns.boxplot(x="finance", y="total score", data=df)

In [40]:

plt.figure(figsize=(10, 4))

sns.swarmplot(x='finance',y='total score',data=df, hue = 'gender')

sns.violinplot(x='finance',y='total score',data=df, inner=None,color='lightgray')

In [41]:

*#Mean scores of students in individual subjects based on finance*

x = df\_finance['total score'].mean().keys()

plt.figure(figsize=(12,4))

plt.subplot(1,3,1)

height = df\_finance['math score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Math Score')

plt.xlabel('Gender')

plt.ylabel('Av. Math Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha='center', fontweight='bold')

plt.subplot(1,3,2)

height = df\_finance['reading score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Reading Score')

plt.xlabel('Gender')

plt.ylabel('Av. Reading Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha='center', fontweight='bold')

plt.subplot(1,3,3)

height = df\_finance['writing score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Writing Score')

plt.xlabel('Gender')

plt.ylabel('Av. Writing Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha = 'center', fontweight='bold')

plt.show()

Analyzing students scores based on Test preparation course

In [42]:

df\_test = df.groupby('test preparation course')

In [43]:

*#Count of students based on test preparation course*

y = df\_test['test preparation course'].count().keys()

width = df\_test['test preparation course'].count()

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of Students with and without test preparation course')

plt.xlabel('Count')

plt.ylabel('Test preparation course')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

In [44]:

*#Mean scores of students based on test preparation course*

y = df\_test['total score'].mean().keys()

width = df\_test['total score'].mean()

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('Mean total score of Students with and without a test preparation course')

plt.xlabel('Mean total score')

plt.ylabel('Test preparation course')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(round(v,2)), color='blue', va='center', fontweight='bold')

plt.show()

Students wwho took test preparation course have performed better.

In [45]:

plt.figure(figsize = (8,4))

sns.boxplot(x="test preparation course", y="total score", data=df)

In [46]:

plt.figure(figsize=(8, 4))

sns.swarmplot(x='test preparation course',y='total score',data=df, hue = 'gender')

sns.violinplot(x='test preparation course',y='total score',data=df, inner=

In [47]:

*#Individual scores based on test preparation course*

x = df\_test['total score'].mean().keys()

plt.figure(figsize=(12,4))

plt.subplot(1,3,1)

height = df\_test['math score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Math Score')

plt.xlabel('Test preparation course')

plt.ylabel('Av. Math Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha='center', fontweight='bold')

plt.subplot(1,3,2)

height = df\_test['reading score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Reading Score')

plt.xlabel('Test preparation course')

plt.ylabel('Av. Reading Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha='center', fontweight='bold')

plt.subplot(1,3,3)

height = df\_test['writing score'].mean()

plt.bar(x = x,

height = height)

plt.title('Av. Writing Score')

plt.xlabel('Test preparation course')

plt.ylabel('Av. Writing Score')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(round(v,2)), color='blue', ha = 'center', fontweight='bold')

plt.show()

In [48]:

*#Toppers*

df\_topper = df[(df['math score'] >= 90) & (df['reading score'] >= 90) & (df['writing score'] >= 90)]

In [49]:

df\_topper['gender'].count()

There are 28 students who have scored 90 or above in all the 3 subjects.

In [50]:

df\_topper.sort\_values('total score',ascending=False).head()

In [51]:

*#Students who have failed in all three subjects(less than 40 score)*

df\_fail\_all = df[(df['math score'] < 40) & (df['reading score'] < 40) & (df['writing score'] < 40)]

df\_fail\_all.head()

In [52]:

y = df['gender'].unique().tolist()

x = [0]\*len(y)

for i **in** range(len(y)):

x[i] += df\_fail\_all[df\_fail\_all['gender'] == y[i]]['gender'].count()

width = x

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of students who failed in all 3 subjects based on gender')

plt.xlabel('Count')

plt.ylabel('Gender')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

Though there are more female students who failed but they are more in number and the difference here is not large so nothing significant here.

In [53]:

y = df['session'].unique().tolist()

x = [0]\*len(y)

for i **in** range(len(y)):

x[i] += df\_fail\_all[df\_fail\_all['session'] == y[i]]['gender'].count()

width = x

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of Students who failed in all 3 subjects in Different Session Groups')

plt.xlabel('Count')

plt.ylabel('Session')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

Though group A students have performed the worst as seen earlier there are lesser who failed than students in group B and group C maybe because group A students are lesser in comparison to group B and C. Also same number of group A, D, E students failed in all subjects even though group E students performed the best.

In [54]:

y = df['parental level of education'].unique().tolist()

x = [0]\*len(y)

for i **in** range(len(y)):

x[i] += df\_fail\_all[df\_fail\_all['parental level of education'] == y[i]]['gender'].count()

width = x

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of Students who failed all subjects based on parental education')

plt.xlabel('Count')

plt.ylabel('parental level of education')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

None of the students with parent's having masters or bachelors degree failed in all 3 subjects.

In [55]:

y = df['finance'].unique().tolist()

x = [0]\*len(y)

for i **in** range(len(y)):

x[i] += df\_fail\_all[df\_fail\_all['finance'] == y[i]]['gender'].count()

width = x

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of Students who failed all subjects based on finance')

plt.xlabel('Count')

plt.ylabel('finance')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

Majority of those who failed take self finace. This implies that students with sponsored have performed better.

In [56]:

y = df['test preparation course'].unique().tolist()

x = [0]\*len(y)

for i **in** range(len(y)):

x[i] += df\_fail\_all[df\_fail\_all['test preparation course'] == y[i]]['gender'].count()

width = x

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of students who failed all subjects based on test preparation course')

plt.xlabel('Count')

plt.ylabel('Test preparation course')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

None of the students who failed in all 3 subjects took a test preparation course.

In [57]:

*#Students who failed in atleast 1 subject (less than 40 score in atleast 1 subject)*

df\_fail = df[(df['math score'] < 40) | (df['reading score'] < 40) | (df['writing score'] < 40)]

df\_fail.head()

In [58]:

width = [df\_fail[df\_fail['math score'] < 40]['gender'].count(), df\_fail[df\_fail['reading score']< 40]['gender'] .count(),

df\_fail[df\_fail['writing score'] < 40]['gender'] .count()]

y = ['math','reading','writing']

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of students who failed in individual subjects')

plt.xlabel('Count')

plt.ylabel('Subject')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

In [59]:

y = df['gender'].unique().tolist()

x = [0]\*len(y)

for i **in** range(len(y)):

x[i] += df\_fail[df\_fail['gender'] == y[i]]['gender'].count()

width = x

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of students who failed atleast 1 subject based on gender')

plt.xlabel('Count')

plt.ylabel('gender')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

Female students did performed better on an average but more of them failed atleast one subject than male students.

In [60]:

*#Number of students who failed in individual subjects based on gender*

plt.figure(figsize = (25,10))

x = ['female','male']

plt.subplot(1,3,1)

height = [df\_fail[(df\_fail['gender']=='female') & (df\_fail['math score'] < 40)]['gender'].count(),

df\_fail[(df\_fail['gender']=='male') & (df\_fail['math score'] < 40)]['gender'].count()]

plt.bar(x = x,

height = height)

plt.title('No. of students who failed in maths based on gender')

plt.ylabel('count')

plt.xlabel('gender')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(v), color='blue', ha='center', fontweight='bold')

plt.subplot(1,3,2)

height = [df\_fail[(df\_fail['gender']=='female') & (df\_fail['reading score'] < 40)]['gender'].count(),

df\_fail[(df\_fail['gender']=='male') & (df\_fail['reading score'] < 40)]['gender'].count()]

plt.bar(x = x,

height = height)

plt.title('No. of students who failed in reading based on gender')

plt.ylabel('count')

plt.xlabel('gender')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(v), color='blue', ha='center', fontweight='bold')

plt.subplot(1,3,3)

height = [df\_fail[(df\_fail['gender']=='female') & (df\_fail['writing score'] < 40)]['gender'].count(),

df\_fail[(df\_fail['gender']=='male') & (df\_fail['writing score'] < 40)]['gender'].count()]

plt.bar(x = x,

height = height)

plt.title('No. of students who failed in writing based on gender')

plt.ylabel('Count')

plt.xlabel('gender')

for i,v **in** enumerate(height):

plt.text(i, v, " "+str(v), color='blue', ha='center', fontweight='bold')

plt.show()

Significantly larger number of female students have failed in maths than male students.

In [61]:

y = df['session'].unique().tolist()

x = [0]\*len(y)

for i **in** range(len(y)):

x[i] += df\_fail[df\_fail['session'] == y[i]]['gender'].count()

width = x

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of students who failed atleast 1 subject based on session')

plt.xlabel('Count')

plt.ylabel('Session')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

Students belonging to group D and E though performed best on average but still a significant amount of them failed in atleast one subject as compared to others.

In [62]:

y = df['parental level of education'].unique().tolist()

x = [0]\*len(y)

for i **in** range(len(y)):

x[i] += df\_fail[df\_fail['parental level of education'] == y[i]]['gender'].count()

width = x

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of students who failed atleast 1 subject based on parental level of education')

plt.xlabel('Count')

plt.ylabel('parental level of education')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

None of the students failed in any subject whom parents had a master's degree.

In [63]:

y = df['finance'].unique().tolist()

x = [0]\*len(y)

for i **in** range(len(y)):

x[i] += df\_fail[df\_fail['finance'] == y[i]]['gender'].count()

width = x

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of students who failed atleast 1 subject based acc. to finance')

plt.xlabel('Count')

plt.ylabel('finance')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

Most of the students who failed atleast 1 subject took free/reduced finance.

In [64]:

y = df['test preparation course'].unique().tolist()

x = [0]\*len(y)

for i **in** range(len(y)):

x[i] += df\_fail[df\_fail['test preparation course'] == y[i]]['gender'].count()

width = x

plt.figure(figsize = (10,2))

plt.barh(y = y,

width = width)

plt.title('No. of students who failed atleast 1 subject based on test preparation course')

plt.xlabel('Count')

plt.ylabel('Test preparation course')

for i,v **in** enumerate(width):

plt.text(v, i, " "+str(v), color='blue', va='center', fontweight='bold')

plt.show()

Most of the students who failed atleast 1 subject didn't took a test preparation course.

**Conclusion**

* Female students lag behind male students in maths whereas male students in reading and writing.
* Higher parental education improves score of students.
* Students with self finance have performed worse than students who are sponsored.
* Test preparation course has helped students score more.