Assignment 1

Data Wrangling I Perform the following operations using Python on any open source dataset (eg. data.csv)

1. Import all the required Python Libraries.
2. Locate an open source data from the web (eg. https://www.kaggle.com (https://www.kaggle.com)). Provide a clear description of the data and its source (i.e. URL of the web site).
3. Load the Dataset into pandas dataframe.
4. Data Preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.
5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions.
6. Turn categorical variables into quantitative variables in Python

1.Import all the required Python Libraries.

In [58]:
```
import pandas as pd
import numpy as np
```

In [24]:
```
pwd
```

Out[24]: 'C:\\Users\\Admin'

2.Locate an open source data from the web (eg. https://www.kaggle.com (https://www.kaggle.com)).

Provide a clear description of the data and its source (i.e. URL of the web site).

In [59]:
```
df=pd. read_csv("C:\\Users\\Admin\\Desktop\\weather_data.csv")
```

In [60]:
```
print(df)
```

```
          day  temperature  windspeed   event  duration
0  01-01-2017         32.0        6.0    Rain         2
1  01-04-2017          NaN        9.0   Sunny         3
2  01-05-2017         28.0        NaN    Snow         4
3  01-06-2017          NaN        7.0     NaN         5
4  01-07-2017         32.0        NaN    Rain         7
5  01-08-2017          NaN        NaN   Sunny         5
6  01-09-2017          NaN        NaN     NaN         6
7  01-10-2017         34.0        8.0  Cloudy         4
8  01-11-2017         40.0       12.0   Sunny         2
```

In [83]:
```
df.head()
```

Out[83]:

	day	temperature	windspeed	event	duration
0	01-01-2017	32.0	6.0	Rain	2.0
1	01-04-2017	33.2	9.0	Sunny	3.0
2	01-05-2017	28.0	33.2	Snow	4.0
3	01-06-2017	33.2	7.0	NaN	5.0
4	01-07-2017	32.0	33.2	Rain	7.0

In [84]:
```
df.tail()
```

Out[84]:

	day	temperature	windspeed	event	duration
4	01-07-2017	32.0	33.2	Rain	7.0
5	01-08-2017	33.2	33.2	Sunny	5.0
6	01-09-2017	33.2	33.2	NaN	6.0
7	01-10-2017	34.0	8.0	Cloudy	4.0
8	01-11-2017	40.0	12.0	Sunny	2.0

4.Data Preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.

In [ ]:
```
4.a)Function: Describe():
        The  Describe function returns the statistical summary of the data frame or series.
```

In [85]:
```
df.describe()
```

Out[85]:

	temperature	windspeed	duration
count	9.000000	9.000000	9.000000
mean	33.200000	19.422222	4.222222
std	3.098387	13.171729	1.715938
min	28.000000	6.000000	2.000000
25%	32.000000	8.000000	3.000000
50%	33.200000	12.000000	4.000000
75%	33.200000	33.200000	5.000000
max	40.000000	33.200000	7.000000

4.b)Stepes for working with missing data 3.b.1. Identify misssing data 3.b.2. deal with missing data 3.b.3.correct data

4.b.1 identify missing data

A) convert the "?","blackspace" into NaN(non numerical data) function: replace() The replace () method replaces the specified value with another specified value. The replace() method searches the entire Data Frame and replaces every case of specified value.

In [61]:
```
df.replace("  ", np.nan, inplace = True)
df.head(9)
```

Out[61]:

	day	temperature	windspeed	event	duration
0	01-01-2017	32.0	6.0	Rain	2
1	01-04-2017	NaN	9.0	Sunny	3
2	01-05-2017	28.0	NaN	Snow	4
3	01-06-2017	NaN	7.0	NaN	5
4	01-07-2017	32.0	NaN	Rain	7
5	01-08-2017	NaN	NaN	Sunny	5
6	01-09-2017	NaN	NaN	NaN	6
7	01-10-2017	34.0	8.0	Cloudy	4
8	01-11-2017	40.0	12.0	Sunny	2

B) Evaluating missing data The missing values are converted to python's default. Built in function: isnull(),.notnull() 1.isnull()-->we can use isnull() method to check whether a cell contains a numeric value(false) or if data is missing(true)

1. notnull()-->otnull() function detects existing/ non-missing values in the dataframe.

In [51]:
```
missingdata=df.isnull()
```

count missing values in each column

In [63]:
```
df.isnull().sum()
```

Out[63]:
```
day            0
temperature    4
windspeed      4
event          2
duration       0
dtype: int64
```

4.b.2 Deal with missing Data a. Drop data Functions:dropna() b. replace data calculate the mean()

DROPPING NULL OR MISSING VALUES This is the fastest and easiest step to handle missing values. However, it is not generally advised. This method reduces the quality of our model as it reduces sample size because it works by deleting all other observations where any of the variable is missing. The process can be done by: data_name.dropna()

In [64]:
```
df.dropna()
```

Out[64]:

	day	temperature	windspeed	event	duration
0	01-01-2017	32.0	6.0	Rain	2
7	01-10-2017	34.0	8.0	Cloudy	4
8	01-11-2017	40.0	12.0	Sunny	2

It will be observed that of 10 entries will be reduced to 3 just by dropping NaN values.!!! Dropping is only advised to be used if missing values are few (say 0.01–0.5% of our data).

In [65]:
```
#original dataset is not changed
df
```

Out[65]:

	day	temperature	windspeed	event	duration
0	01-01-2017	32.0	6.0	Rain	2
1	01-04-2017	NaN	9.0	Sunny	3
2	01-05-2017	28.0	NaN	Snow	4
3	01-06-2017	NaN	7.0	NaN	5
4	01-07-2017	32.0	NaN	Rain	7
5	01-08-2017	NaN	NaN	Sunny	5
6	01-09-2017	NaN	NaN	NaN	6
7	01-10-2017	34.0	8.0	Cloudy	4
8	01-11-2017	40.0	12.0	Sunny	2

In [67]:
```
#We can drop columns that have at least one NaN in any row by setting the axis argument
df.dropna(axis=1)
```

Out[67]:

	day	duration
0	01-01-2017	2
1	01-04-2017	3
2	01-05-2017	4
3	01-06-2017	5
4	01-07-2017	7
5	01-08-2017	5
6	01-09-2017	6
7	01-10-2017	4
8	01-11-2017	2

In [68]:
```
#The dropna() method has several additional parameters:
df.dropna(how='all')
```

Out[68]:

	day	temperature	windspeed	event	duration
0	01-01-2017	32.0	6.0	Rain	2
1	01-04-2017	NaN	9.0	Sunny	3
2	01-05-2017	28.0	NaN	Snow	4
3	01-06-2017	NaN	7.0	NaN	5
4	01-07-2017	32.0	NaN	Rain	7
5	01-08-2017	NaN	NaN	Sunny	5
6	01-09-2017	NaN	NaN	NaN	6
7	01-10-2017	34.0	8.0	Cloudy	4
8	01-11-2017	40.0	12.0	Sunny	2

In [69]:
```
df.dropna(subset=['event'])
```

Out[69]:

	day	temperature	windspeed	event	duration
0	01-01-2017	32.0	6.0	Rain	2
1	01-04-2017	NaN	9.0	Sunny	3
2	01-05-2017	28.0	NaN	Snow	4
4	01-07-2017	32.0	NaN	Rain	7
5	01-08-2017	NaN	NaN	Sunny	5
7	01-10-2017	34.0	8.0	Cloudy	4
8	01-11-2017	40.0	12.0	Sunny	2

Replace Data Calculate the mean and replace the null value by mean by using fillna() function

In [70]:
```
mean_value = df['temperature'].mean()
```

In [71]:
```
mean_value
```

Out[71]: 33.2

In [72]:
```
df['temperature'] = df['temperature'].fillna(mean_value)
```

In [73]:
```
df
```

Out[73]:

	day	temperature	windspeed	event	duration
0	01-01-2017	32.0	6.0	Rain	2
1	01-04-2017	33.2	9.0	Sunny	3
2	01-05-2017	28.0	NaN	Snow	4
3	01-06-2017	33.2	7.0	NaN	5
4	01-07-2017	32.0	NaN	Rain	7
5	01-08-2017	33.2	NaN	Sunny	5
6	01-09-2017	33.2	NaN	NaN	6
7	01-10-2017	34.0	8.0	Cloudy	4
8	01-11-2017	40.0	12.0	Sunny	2

In [74]:
```
mean_v=df['windspeed'].mean()
```

In [44]:
```
mean_v
```

Out[44]: 8.4

In [75]:
```
df['windspeed'] = df['windspeed'].fillna(mean_value)
```

In [46]:
```
df
```

Out[46]:

	day	temperature	windspeed	event
0	01-01-2017	32.0	8.4	Rain
1	01-04-2017	33.2	8.4	Sunny
2	01-05-2017	28.0	8.4	Snow
3	01-06-2017	33.2	8.4	NaN
4	01-07-2017	32.0	8.4	Rain
5	01-08-2017	33.2	8.4	Sunny
6	01-09-2017	33.2	8.4	NaN
7	01-10-2017	34.0	8.4	Cloudy
8	01-11-2017	40.0	8.4	Sunny

let's say I want to replace all any values with zero and in my day not day but wind speed column I want to replace it with again zero but my event I want to say "No Event" and then print new data frame now as you can see here the temperature and wind speed is replaced with zero as you can see here but the event now I have no event

In [76]:
```
new_df = df.fillna({'temperature':0,
'windspeed':0,
'event':'no_event'})
```

In [77]:
```
new_df
```

Out[77]:

	day	temperature	windspeed	event	duration
0	01-01-2017	32.0	6.0	Rain	2
1	01-04-2017	33.2	9.0	Sunny	3
2	01-05-2017	28.0	33.2	Snow	4
3	01-06-2017	33.2	7.0	no_event	5
4	01-07-2017	32.0	33.2	Rain	7
5	01-08-2017	33.2	33.2	Sunny	5
6	01-09-2017	33.2	33.2	no_event	6
7	01-10-2017	34.0	8.0	Cloudy	4
8	01-11-2017	40.0	12.0	Sunny	2

5.Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions. Function: dtypes()--> to check the data type astype() to change the data type

In [78]:
```
df.dtypes
```

Out[78]:
```
day             object
temperature    float64
windspeed      float64
event           object
duration         int64
dtype: object
```

In [79]:
```
df[["duration"]]=df[["duration"]].astype("float")
```

In [80]:
```
df.head()
```

Out[80]:

	day	temperature	windspeed	event	duration
0	01-01-2017	32.0	6.0	Rain	2.0
1	01-04-2017	33.2	9.0	Sunny	3.0
2	01-05-2017	28.0	33.2	Snow	4.0
3	01-06-2017	33.2	7.0	NaN	5.0
4	01-07-2017	32.0	33.2	Rain	7.0

6.Turn categorical variables into quantitative variables in Python Under this approach, we deploy the simplest way to perform the conversion of all possible Categorical Columns in a data frame to Dummy Columns by using the get_dummies() method of the pandas library.

We can either specify the columns to get the dummies by default it will convert all the possible categorical columns to their dummy columns.

In [81]:
```
# creating a copy of the original data frame
df3 = df.copy()
```

In [82]:
```
# calling the get_dummies method
# the first parameter mentions the
# the name of the data frame to store the
# new data frame in
# the second parameter is the list of
# columns which if not mentioned
# returns the dummies for all
# categorical columns

df3 = pd.get_dummies(df3,
                     columns = ['event'])

display(df3)
```

	day	temperature	windspeed	duration	event_Cloudy	event_Rain	event_Snow	event_Sunny
0	01-01-2017	32.0	6.0	2.0	0	1	0	0
1	01-04-2017	33.2	9.0	3.0	0	0	0	1
2	01-05-2017	28.0	33.2	4.0	0	0	1	0
3	01-06-2017	33.2	7.0	5.0	0	0	0	0
4	01-07-2017	32.0	33.2	7.0	0	1	0	0
5	01-08-2017	33.2	33.2	5.0	0	0	0	1
6	01-09-2017	33.2	33.2	6.0	0	0	0	0
7	01-10-2017	34.0	8.0	4.0	1	0	0	0
8	01-11-2017	40.0	12.0	2.0	0	0	0	1

In [ ]:

Assignment 2:Data Wrangling II Perform the following operations using Python on any open source dataset (eg. data.csv)

1. Scan all variables for missing values and inconsistencies. If there are missing values and/or inconsistencies, use any of the suitable techniques to deal with them.
2. Scan all numeric variables for outliers. If there are outliers, use any of the suitable techniques to deal with them.
3. Apply data transformations on at least one of the variables. The purpose of this transformation should be one of the following reasons: to change the scale for better understanding of the variable, to convert a non-linear relation into a linear one, or to decrease the skewness and convert the distribution into a normal distribution. Reason and document your approach properly.

In [ ]:

```

```

In [ ]:

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

In [26]:

```
pwd
```

Out[26]:

```
'C:\\Users\\Admin'
```

In [27]:

```
df=pd.read_csv("C:\\Users\\Admin\\Desktop\\StudentPerformance.csv")
```

In [28]:

```
print(df)
```

```
    Maths_Score  Reading_Score  Writing_Score  Placement_Score  \
0          70.0           93.0           61.0             82.0
1          77.0           84.0           65.0             88.0
2          69.0           84.0           68.0             93.0
3          72.0           81.0           73.0             91.0
4          78.0           95.0           73.0             96.0
5           NaN           94.0            NaN             80.0
6          69.0           86.0           79.0             91.0
7          76.0           92.0           61.0             79.0
8          79.0           81.0           77.0             80.0
9          65.0           85.0           78.0             76.0
10         66.0           78.0           69.0             94.0
11         75.0            NaN            NaN             90.0
12         75.0           81.0           74.0             88.0
13         94.0           75.0           80.0             83.0
14         69.0           79.0           79.0              NaN
15         60.0           88.0           61.0             81.0
16         79.0           84.0           75.0             76.0
17         68.0           80.0           66.0             89.0
18         65.0           85.0           68.0             92.0
19         63.0           75.0           75.0             84.0
20         71.0           78.0           67.0             83.0
21         67.0           89.0           95.0             78.0
22         74.0           77.0           72.0             81.0
23         64.0           76.0           67.0             82.0
24         61.0           87.0           63.0             98.0
25         76.0           91.0           60.0             88.0
26         67.0           93.0           76.0             90.0
27         93.0           88.0           99.0             91.0
28         62.0           79.0           67.0             86.0

    Club_Join_Date  Placement offer count
0             2020                      2
1             2019                      3
2             2020                      3
3             2019                      3
4             2020                      3
5             2020                      2
6             2018                      3
7             2019                      2
8             2018                      2
9             2020                      2
10            2018                      3
11            2018                      3
12            2019                      3
13            2020                      2
14            2020                      2
15            2018                      2
16            2018                      2
17            2020                      3
18            2020                      3
19            2018                      2
20            2018                      2
21            2019                      2
22            2020                      2
23            2018                      2
24            2020                      3
25            2019                      3
26            2019                      3
27            2019                      3
28            2020                      3
```

In [29]:
```
df
```

Out[29]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70.0	93.0	61.0	82.0	2020	2
1	77.0	84.0	65.0	88.0	2019	3
2	69.0	84.0	68.0	93.0	2020	3
3	72.0	81.0	73.0	91.0	2019	3
4	78.0	95.0	73.0	96.0	2020	3
5	NaN	94.0	NaN	80.0	2020	2
6	69.0	86.0	79.0	91.0	2018	3
7	76.0	92.0	61.0	79.0	2019	2
8	79.0	81.0	77.0	80.0	2018	2
9	65.0	85.0	78.0	76.0	2020	2
10	66.0	78.0	69.0	94.0	2018	3
11	75.0	NaN	NaN	90.0	2018	3
12	75.0	81.0	74.0	88.0	2019	3
13	94.0	75.0	80.0	83.0	2020	2
14	69.0	79.0	79.0	NaN	2020	2
15	60.0	88.0	61.0	81.0	2018	2
16	79.0	84.0	75.0	76.0	2018	2
17	68.0	80.0	66.0	89.0	2020	3
18	65.0	85.0	68.0	92.0	2020	3
19	63.0	75.0	75.0	84.0	2018	2
20	71.0	78.0	67.0	83.0	2018	2
21	67.0	89.0	95.0	78.0	2019	2
22	74.0	77.0	72.0	81.0	2020	2
23	64.0	76.0	67.0	82.0	2018	2
24	61.0	87.0	63.0	98.0	2020	3
25	76.0	91.0	60.0	88.0	2019	3
26	67.0	93.0	76.0	90.0	2019	3
27	93.0	88.0	99.0	91.0	2019	3
28	62.0	79.0	67.0	86.0	2020	3

In [30]:
```
df.isnull()
```

Out[30]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
5	True	False	True	False	False	False
6	False	False	False	False	False	False
7	False	False	False	False	False	False
8	False	False	False	False	False	False
9	False	False	False	False	False	False
10	False	False	False	False	False	False
11	False	True	True	False	False	False
12	False	False	False	False	False	False
13	False	False	False	False	False	False
14	False	False	False	True	False	False
15	False	False	False	False	False	False
16	False	False	False	False	False	False
17	False	False	False	False	False	False
18	False	False	False	False	False	False
19	False	False	False	False	False	False
20	False	False	False	False	False	False
21	False	False	False	False	False	False
22	False	False	False	False	False	False
23	False	False	False	False	False	False
24	False	False	False	False	False	False
25	False	False	False	False	False	False
26	False	False	False	False	False	False
27	False	False	False	False	False	False
28	False	False	False	False	False	False

In [31]:
```
df.isnull().sum()
```

Out[31]:
```
Maths_Score              1
Reading_Score            1
Writing_Score            2
Placement_Score          1
Club_Join_Date           0
Placement offer count    0
dtype: int64
```

In [32]:
```
df.notnull()
```

Out[32]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	True	True	True	True	True	True
1	True	True	True	True	True	True
2	True	True	True	True	True	True
3	True	True	True	True	True	True
4	True	True	True	True	True	True
5	False	True	False	True	True	True
6	True	True	True	True	True	True
7	True	True	True	True	True	True
8	True	True	True	True	True	True
9	True	True	True	True	True	True
10	True	True	True	True	True	True
11	True	False	False	True	True	True
12	True	True	True	True	True	True
13	True	True	True	True	True	True
14	True	True	True	False	True	True
15	True	True	True	True	True	True
16	True	True	True	True	True	True
17	True	True	True	True	True	True
18	True	True	True	True	True	True
19	True	True	True	True	True	True
20	True	True	True	True	True	True
21	True	True	True	True	True	True
22	True	True	True	True	True	True
23	True	True	True	True	True	True
24	True	True	True	True	True	True
25	True	True	True	True	True	True
26	True	True	True	True	True	True
27	True	True	True	True	True	True
28	True	True	True	True	True	True

In [33]:
```
series1=pd.notnull(df["Maths_Score"])
```

In [34]:
```
df[series1]
```

Out[34]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70.0	93.0	61.0	82.0	2020	2
1	77.0	84.0	65.0	88.0	2019	3
2	69.0	84.0	68.0	93.0	2020	3
3	72.0	81.0	73.0	91.0	2019	3
4	78.0	95.0	73.0	96.0	2020	3
6	69.0	86.0	79.0	91.0	2018	3
7	76.0	92.0	61.0	79.0	2019	2
8	79.0	81.0	77.0	80.0	2018	2
9	65.0	85.0	78.0	76.0	2020	2
10	66.0	78.0	69.0	94.0	2018	3
11	75.0	NaN	NaN	90.0	2018	3
12	75.0	81.0	74.0	88.0	2019	3
13	94.0	75.0	80.0	83.0	2020	2
14	69.0	79.0	79.0	NaN	2020	2
15	60.0	88.0	61.0	81.0	2018	2
16	79.0	84.0	75.0	76.0	2018	2
17	68.0	80.0	66.0	89.0	2020	3
18	65.0	85.0	68.0	92.0	2020	3
19	63.0	75.0	75.0	84.0	2018	2
20	71.0	78.0	67.0	83.0	2018	2
21	67.0	89.0	95.0	78.0	2019	2
22	74.0	77.0	72.0	81.0	2020	2
23	64.0	76.0	67.0	82.0	2018	2
24	61.0	87.0	63.0	98.0	2020	3
25	76.0	91.0	60.0	88.0	2019	3
26	67.0	93.0	76.0	90.0	2019	3
27	93.0	88.0	99.0	91.0	2019	3
28	62.0	79.0	67.0	86.0	2020	3

In [35]:
```
ndf=df
ndf.fillna(0)
```

Out[35]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70.0	93.0	61.0	82.0	2020	2
1	77.0	84.0	65.0	88.0	2019	3
2	69.0	84.0	68.0	93.0	2020	3
3	72.0	81.0	73.0	91.0	2019	3
4	78.0	95.0	73.0	96.0	2020	3
5	0.0	94.0	0.0	80.0	2020	2
6	69.0	86.0	79.0	91.0	2018	3
7	76.0	92.0	61.0	79.0	2019	2
8	79.0	81.0	77.0	80.0	2018	2
9	65.0	85.0	78.0	76.0	2020	2
10	66.0	78.0	69.0	94.0	2018	3
11	75.0	0.0	0.0	90.0	2018	3
12	75.0	81.0	74.0	88.0	2019	3
13	94.0	75.0	80.0	83.0	2020	2
14	69.0	79.0	79.0	0.0	2020	2
15	60.0	88.0	61.0	81.0	2018	2
16	79.0	84.0	75.0	76.0	2018	2
17	68.0	80.0	66.0	89.0	2020	3
18	65.0	85.0	68.0	92.0	2020	3
19	63.0	75.0	75.0	84.0	2018	2
20	71.0	78.0	67.0	83.0	2018	2
21	67.0	89.0	95.0	78.0	2019	2
22	74.0	77.0	72.0	81.0	2020	2
23	64.0	76.0	67.0	82.0	2018	2
24	61.0	87.0	63.0	98.0	2020	3
25	76.0	91.0	60.0	88.0	2019	3
26	67.0	93.0	76.0	90.0	2019	3
27	93.0	88.0	99.0	91.0	2019	3
28	62.0	79.0	67.0	86.0	2020	3

In [36]:
```
df['Maths_Score']=df['Maths_Score'].fillna(df['Maths_Score'].mean())
```

In [37]:
```
df
```

Out[37]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70.000000	93.0	61.0	82.0	2020	2
1	77.000000	84.0	65.0	88.0	2019	3
2	69.000000	84.0	68.0	93.0	2020	3
3	72.000000	81.0	73.0	91.0	2019	3
4	78.000000	95.0	73.0	96.0	2020	3
5	71.571429	94.0	NaN	80.0	2020	2
6	69.000000	86.0	79.0	91.0	2018	3
7	76.000000	92.0	61.0	79.0	2019	2
8	79.000000	81.0	77.0	80.0	2018	2
9	65.000000	85.0	78.0	76.0	2020	2
10	66.000000	78.0	69.0	94.0	2018	3
11	75.000000	NaN	NaN	90.0	2018	3
12	75.000000	81.0	74.0	88.0	2019	3
13	94.000000	75.0	80.0	83.0	2020	2
14	69.000000	79.0	79.0	NaN	2020	2
15	60.000000	88.0	61.0	81.0	2018	2
16	79.000000	84.0	75.0	76.0	2018	2
17	68.000000	80.0	66.0	89.0	2020	3
18	65.000000	85.0	68.0	92.0	2020	3
19	63.000000	75.0	75.0	84.0	2018	2
20	71.000000	78.0	67.0	83.0	2018	2
21	67.000000	89.0	95.0	78.0	2019	2
22	74.000000	77.0	72.0	81.0	2020	2
23	64.000000	76.0	67.0	82.0	2018	2
24	61.000000	87.0	63.0	98.0	2020	3
25	76.000000	91.0	60.0	88.0	2019	3
26	67.000000	93.0	76.0	90.0	2019	3
27	93.000000	88.0	99.0	91.0	2019	3
28	62.000000	79.0	67.0	86.0	2020	3

In [38]:
```
ndf.replace(to_replace=np.nan,value=-99)
```

Out[38]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70.000000	93.0	61.0	82.0	2020	2
1	77.000000	84.0	65.0	88.0	2019	3
2	69.000000	84.0	68.0	93.0	2020	3
3	72.000000	81.0	73.0	91.0	2019	3
4	78.000000	95.0	73.0	96.0	2020	3
5	71.571429	94.0	-99.0	80.0	2020	2
6	69.000000	86.0	79.0	91.0	2018	3
7	76.000000	92.0	61.0	79.0	2019	2
8	79.000000	81.0	77.0	80.0	2018	2
9	65.000000	85.0	78.0	76.0	2020	2
10	66.000000	78.0	69.0	94.0	2018	3
11	75.000000	-99.0	-99.0	90.0	2018	3
12	75.000000	81.0	74.0	88.0	2019	3
13	94.000000	75.0	80.0	83.0	2020	2
14	69.000000	79.0	79.0	-99.0	2020	2
15	60.000000	88.0	61.0	81.0	2018	2
16	79.000000	84.0	75.0	76.0	2018	2
17	68.000000	80.0	66.0	89.0	2020	3
18	65.000000	85.0	68.0	92.0	2020	3
19	63.000000	75.0	75.0	84.0	2018	2
20	71.000000	78.0	67.0	83.0	2018	2
21	67.000000	89.0	95.0	78.0	2019	2
22	74.000000	77.0	72.0	81.0	2020	2
23	64.000000	76.0	67.0	82.0	2018	2
24	61.000000	87.0	63.0	98.0	2020	3
25	76.000000	91.0	60.0	88.0	2019	3
26	67.000000	93.0	76.0	90.0	2019	3
27	93.000000	88.0	99.0	91.0	2019	3
28	62.000000	79.0	67.0	86.0	2020	3

In [39]:
```
ndf.dropna()
```

Out[39]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70.0	93.0	61.0	82.0	2020	2
1	77.0	84.0	65.0	88.0	2019	3
2	69.0	84.0	68.0	93.0	2020	3
3	72.0	81.0	73.0	91.0	2019	3
4	78.0	95.0	73.0	96.0	2020	3
6	69.0	86.0	79.0	91.0	2018	3
7	76.0	92.0	61.0	79.0	2019	2
8	79.0	81.0	77.0	80.0	2018	2
9	65.0	85.0	78.0	76.0	2020	2
10	66.0	78.0	69.0	94.0	2018	3
12	75.0	81.0	74.0	88.0	2019	3
13	94.0	75.0	80.0	83.0	2020	2
15	60.0	88.0	61.0	81.0	2018	2
16	79.0	84.0	75.0	76.0	2018	2
17	68.0	80.0	66.0	89.0	2020	3
18	65.0	85.0	68.0	92.0	2020	3
19	63.0	75.0	75.0	84.0	2018	2
20	71.0	78.0	67.0	83.0	2018	2
21	67.0	89.0	95.0	78.0	2019	2
22	74.0	77.0	72.0	81.0	2020	2
23	64.0	76.0	67.0	82.0	2018	2
24	61.0	87.0	63.0	98.0	2020	3
25	76.0	91.0	60.0	88.0	2019	3
26	67.0	93.0	76.0	90.0	2019	3
27	93.0	88.0	99.0	91.0	2019	3
28	62.0	79.0	67.0	86.0	2020	3

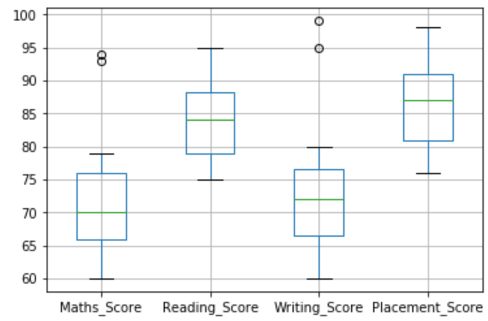
Module 2:Detection of Outlier 1.we can plot the outlier by using Boxplot, Scatterplot

1. Techniques of Detecting outlier a. Z-score b.IQR

Boxplot:

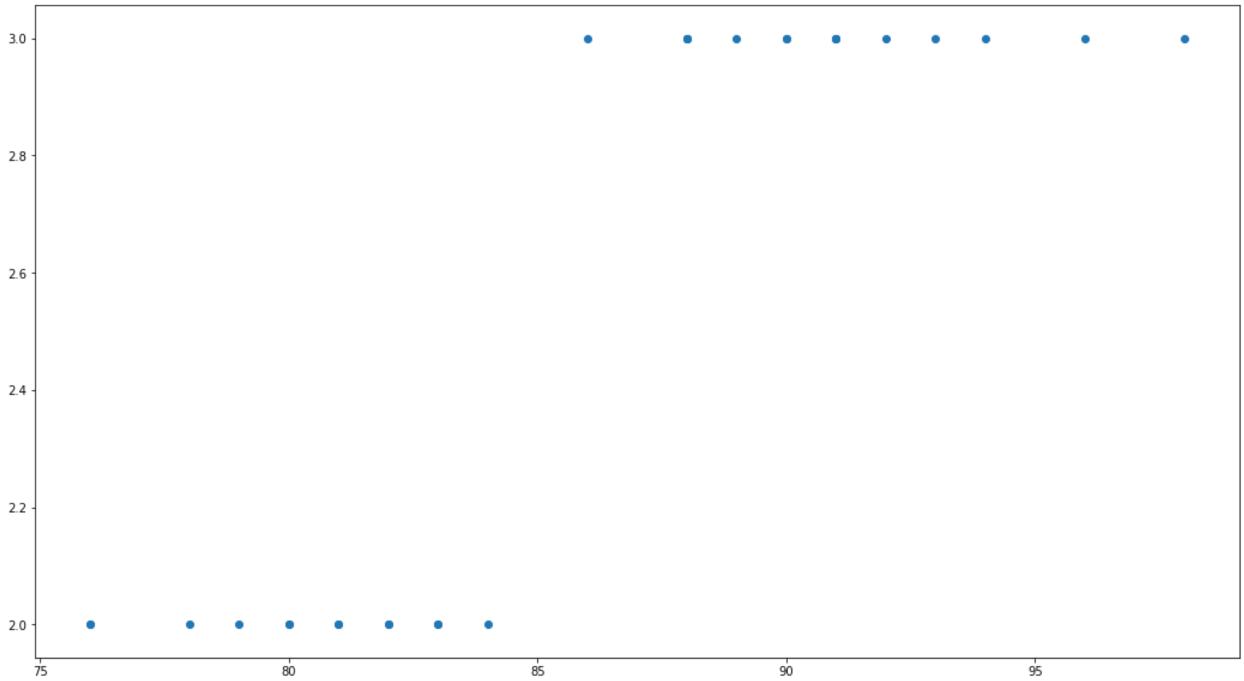
In [40]:
```
# Boxplot--> Summaries sample data using 25th, 50th,75th value
col=['Maths_Score','Reading_Score','Writing_Score','Placement_Score']
df.boxplot(col)
```

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x11240b0>



Scatterplot:It is used when you have paired numerical data, or when your dependent variable has multiple values for each reading independent variable, or when trying to determine the relationship between the two variables. In the process of utilizing the scatter plot, one can also use it for outlier detection.

In [41]:
```
fig,ax=plt.subplots(figsize=(18,10))
ax.scatter(df['Placement_Score'],df['Placement offer count'])
plt.show()
```



In [ ]:

Module 2:Detection of Outlier

1.Outlier visualization- Boxplot, Scatterplot 2.Techniques of Detecting outlier a. Z-score b.IQR

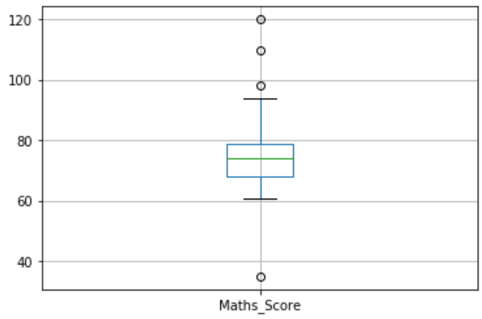
Outlier visualization- Boxplot,

In [124]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [125]:
```
df=pd.read_csv("C:\\Users\\Admin\\Desktop\\StudentPerformance.csv")
```

In [126]:
```
new_df = df

col = ['Maths_Score']
#******Identifying Outliers with Visualization
new_df.boxplot(col) #  outliers are seen in boxplot
```

Out[126]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xb9e2f10>
```



Detecting outlier by using IQR

InterQuantile Range 75%- 25% values in a dataset

Steps

1. Arrange the data in increasing order
2. Calculate first(q1) and third quartile(q3)
3. Find interquartile range (q3-q1) 4.Find lower bound q1*1.5 5.Find upper bound q3*1.5 Percentiles are used in statistics to give you a number that describes the value that a given percent of the values are lower than. numpy.percentile(arr, n)

In [127]:
```
#Identifying Outliers with Interquartile Range (IQR) Calculate and print Quartile 1 and Quartile

q1 = np.percentile(df['Maths_Score'],25)
q3 = np.percentile(df['Maths_Score'],75)
print(q1,q3)
```

```
68.0 79.0
```

In [128]:
```
#Calculate value of IQR (Inter Quartile Range)
IQR = q3-q1

#Calculate and print Upper and Lower Bound to define the outlier base value.
lwr_bound = q1-(1.5*IQR)
upr_bound = q3+(1.5*IQR)
print(lwr_bound, upr_bound)
```

```
51.5 95.5
```

In [129]:
```
index_outliers = np.where((df[col] < lwr_bound) | ( df[col] > upr_bound))
index_outliers
```

Out[129]:
```
(array([ 3, 15, 19, 27], dtype=int32), array([0, 0, 0, 0], dtype=int32))
```

In [130]:
```
df
```

Out[130]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70	93	61	82	2020	2
1	77	84	65	88	2019	3
2	69	84	68	93	2020	3
3	35	81	73	91	2019	3
4	78	95	73	96	2020	3
5	70	94	60	80	2020	2
6	69	86	79	91	2018	3
7	76	92	61	79	2019	2
8	79	81	77	80	2018	2
9	79	85	78	76	2020	2
10	66	78	69	94	2018	3
11	75	60	60	90	2018	3
12	75	81	74	88	2019	3
13	94	75	80	83	2020	2
14	69	79	79	80	2020	2
15	110	88	61	81	2018	2
16	79	84	75	76	2018	2
17	68	80	66	89	2020	3
18	65	85	68	92	2020	3
19	120	75	75	84	2018	2
20	71	78	67	83	2018	2
21	67	89	95	78	2019	2
22	74	77	72	81	2020	2
23	64	76	67	82	2018	2
24	61	87	63	98	2020	3
25	76	91	60	88	2019	3
26	93	93	76	90	2019	3
27	98	88	99	91	2019	3
28	62	79	67	86	2020	3

In [131]:
```
sample_outliers= df[col][(df[col] < lwr_bound) | (df[col] > upr_bound)]
sample_outliers
```

Out[131]:

	Maths_Score
0	NaN
1	NaN
2	NaN
3	35.0
4	NaN
5	NaN
6	NaN
7	NaN
8	NaN
9	NaN
10	NaN
11	NaN
12	NaN
13	NaN
14	NaN
15	110.0
16	NaN
17	NaN
18	NaN
19	120.0
20	NaN
21	NaN
22	NaN
23	NaN
24	NaN
25	NaN
26	NaN
27	98.0
28	NaN

Handling of Outliers:

For removing the outlier, one must follow the same process of removing an entry from the dataset using its exact position in the dataset because in all the above methods of detecting the outliers end result is the list of all those data items that satisfy the outlier definition according to the method used. Below are some of the methods of treating the outliers ● Trimming/removing the outlier ● Quantile based flooring and capping ● Mean/Median imputation

Quantile based flooring and capping

the outlier is capped at a certain value above the 90th percentile value or floored at a factor below the 10th percentile value

In [132]:
```
df1=df
df[col] = np.where(df1[col]< lwr_bound,lwr_bound,df[col])
df[col] = np.where(df1[col]>upr_bound ,upr_bound,df[col])
```

In [133]:
```
df1
```

Out[133]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70.0	93	61	82	2020	2
1	77.0	84	65	88	2019	3
2	69.0	84	68	93	2020	3
3	51.5	81	73	91	2019	3
4	78.0	95	73	96	2020	3
5	70.0	94	60	80	2020	2
6	69.0	86	79	91	2018	3
7	76.0	92	61	79	2019	2
8	79.0	81	77	80	2018	2
9	79.0	85	78	76	2020	2
10	66.0	78	69	94	2018	3
11	75.0	60	60	90	2018	3
12	75.0	81	74	88	2019	3
13	94.0	75	80	83	2020	2
14	69.0	79	79	80	2020	2
15	95.5	88	61	81	2018	2
16	79.0	84	75	76	2018	2
17	68.0	80	66	89	2020	3
18	65.0	85	68	92	2020	3
19	95.5	75	75	84	2018	2
20	71.0	78	67	83	2018	2
21	67.0	89	95	78	2019	2
22	74.0	77	72	81	2020	2
23	64.0	76	67	82	2018	2
24	61.0	87	63	98	2020	3
25	76.0	91	60	88	2019	3
26	93.0	93	76	90	2019	3
27	95.5	88	99	91	2019	3
28	62.0	79	67	86	2020	3

In [146]:
```
df5=pd.read_csv("C:\\Users\\Admin\\Desktop\\StudentPerformance.csv")
df5
```

Out[146]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70	93	61	82	2020	2
1	77	84	65	88	2019	3
2	69	84	68	93	2020	3
3	35	81	73	91	2019	3
4	78	95	73	96	2020	3
5	70	94	60	80	2020	2
6	69	86	79	91	2018	3
7	76	92	61	79	2019	2
8	79	81	77	80	2018	2
9	79	85	78	76	2020	2
10	66	78	69	94	2018	3
11	75	60	60	90	2018	3
12	75	81	74	88	2019	3
13	94	75	80	83	2020	2
14	69	79	79	80	2020	2
15	110	88	61	81	2018	2
16	79	84	75	76	2018	2
17	68	80	66	89	2020	3
18	65	85	68	92	2020	3
19	120	75	75	84	2018	2
20	71	78	67	83	2018	2
21	67	89	95	78	2019	2
22	74	77	72	81	2020	2
23	64	76	67	82	2018	2
24	61	87	63	98	2020	3
25	76	91	60	88	2019	3
26	93	93	76	90	2019	3
27	98	88	99	91	2019	3
28	62	79	67	86	2020	3

In [147]:
```
ninetieth_percentile = np.percentile(df5['Maths_Score'], 90)
ninetieth_percentile
```

Out[147]: 94.80000000000001

In [148]:
```
tenth_percentile = np.percentile(df5['Maths_Score'], 10)
tenth_percentile
```

Out[148]: 63.6

In [150]:
```
df5[col] = np.where(df5[col]>upr_bound ,ninetieth_percentile,df5[col])
df5[col] = np.where(df5[col]<lwr_bound ,tenth_percentile,df5[col])
df5
```

Out[150]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70.0	93	61	82	2020	2
1	77.0	84	65	88	2019	3
2	69.0	84	68	93	2020	3
3	63.6	81	73	91	2019	3
4	78.0	95	73	96	2020	3
5	70.0	94	60	80	2020	2
6	69.0	86	79	91	2018	3
7	76.0	92	61	79	2019	2
8	79.0	81	77	80	2018	2
9	79.0	85	78	76	2020	2
10	66.0	78	69	94	2018	3
11	75.0	60	60	90	2018	3
12	75.0	81	74	88	2019	3
13	94.0	75	80	83	2020	2
14	69.0	79	79	80	2020	2
15	94.8	88	61	81	2018	2
16	79.0	84	75	76	2018	2
17	68.0	80	66	89	2020	3
18	65.0	85	68	92	2020	3
19	94.8	75	75	84	2018	2
20	71.0	78	67	83	2018	2
21	67.0	89	95	78	2019	2
22	74.0	77	72	81	2020	2
23	64.0	76	67	82	2018	2
24	61.0	87	63	98	2020	3
25	76.0	91	60	88	2019	3
26	93.0	93	76	90	2019	3
27	94.8	88	99	91	2019	3
28	62.0	79	67	86	2020	3

Mean/Median imputation: As the mean value is highly influenced by the outliers, it is advised to replace the outliers with the median value

In [137]:
```
#Calculate the median of reading score by using sorted_rscore
median = np.median(new_df[col])
median
```

Out[137]: 74.0

In [138]:
```
#Replace the lower bound and upper bound outliers using median value
for i in index_outliers:
   new_df.at[i,col] = median

new_df
```

Out[138]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	74.0	93	61	82	2020	2
1	77.0	84	65	88	2019	3
2	69.0	84	68	93	2020	3
3	74.0	81	73	91	2019	3
4	78.0	95	73	96	2020	3
5	70.0	94	60	80	2020	2
6	69.0	86	79	91	2018	3
7	76.0	92	61	79	2019	2
8	79.0	81	77	80	2018	2
9	79.0	85	78	76	2020	2
10	66.0	78	69	94	2018	3
11	75.0	60	60	90	2018	3
12	75.0	81	74	88	2019	3
13	94.0	75	80	83	2020	2
14	69.0	79	79	80	2020	2
15	74.0	88	61	81	2018	2
16	79.0	84	75	76	2018	2
17	68.0	80	66	89	2020	3
18	65.0	85	68	92	2020	3
19	74.0	75	75	84	2018	2
20	71.0	78	67	83	2018	2
21	67.0	89	95	78	2019	2
22	74.0	77	72	81	2020	2
23	64.0	76	67	82	2018	2
24	61.0	87	63	98	2020	3
25	76.0	91	60	88	2019	3
26	93.0	93	76	90	2019	3
27	74.0	88	99	91	2019	3
28	62.0	79	67	86	2020	3

Z-Score Z-Score is also called a standard score. This value/score helps to understand how far is the data point from the mean. And after setting up athreshold value one can utilize z score values of data points to define the outliers. Zscore = (data_point -mean) / std. deviation

In [157]:
```
from scipy import stats
df3=pd.read_csv("C:\\Users\\Admin\\Desktop\\StudentPerformance.csv")
df3
```

Out[157]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70	93	61	82	2020	2
1	77	84	65	88	2019	3
2	69	84	68	93	2020	3
3	35	81	73	91	2019	3
4	78	95	73	96	2020	3
5	70	94	60	80	2020	2
6	69	86	79	91	2018	3
7	76	92	61	79	2019	2
8	79	81	77	80	2018	2
9	79	85	78	76	2020	2
10	66	78	69	94	2018	3
11	75	60	60	90	2018	3
12	75	81	74	88	2019	3
13	94	75	80	83	2020	2
14	69	79	79	80	2020	2
15	110	88	61	81	2018	2
16	79	84	75	76	2018	2
17	68	80	66	89	2020	3
18	65	85	68	92	2020	3
19	120	75	75	84	2018	2
20	71	78	67	83	2018	2
21	67	89	95	78	2019	2
22	74	77	72	81	2020	2
23	64	76	67	82	2018	2
24	61	87	63	98	2020	3
25	76	91	60	88	2019	3
26	93	93	76	90	2019	3
27	98	88	99	91	2019	3
28	62	79	67	86	2020	3

In [158]:
```
z = np.abs(stats.zscore(df3['Maths_Score']))
print(z)
```

```
[0.35101647 0.09713663 0.41503834 2.59178199 0.16115851 0.35101647
 0.41503834 0.03311476 0.22518038 0.22518038 0.60710396 0.03090711
 0.03090711 1.18550846 0.41503834 2.20985841 0.22518038 0.47906021
 0.67112583 2.85007713 0.2869946  0.54308209 0.09492898 0.7351477
 0.92721332 0.03311476 1.12148658 1.44159594 0.86319145]
```

In [159]:
```
Upperthreshold = 1.4
Lowerthreshold = 0.18

index_outliers = np.where((z < Lowerthreshold) | ( z > Upperthreshold))

index_outliers
```

Out[159]:
```
(array([ 1,  3,  4,  7, 11, 12, 15, 19, 22, 25, 27], dtype=int32),)
```

In [160]:
```
sample_outliers= z[(z < Lowerthreshold) | (z > Upperthreshold)]
sample_outliers
```

Out[160]:
```
array([0.09713663, 2.59178199, 0.16115851, 0.03311476, 0.03090711,
       0.03090711, 2.20985841, 2.85007713, 0.09492898, 0.03311476,
       1.44159594])
```

In [161]:
```
##Trimming/removing the outlier:  In this technique, we remove the outliers from the dataset. Although it is not a good pract
ice to follow.


new_df2=df3
for i in index_outliers:
  new_df2.drop(i,inplace=True)
new_df2
```

Out[161]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70	93	61	82	2020	2
2	69	84	68	93	2020	3
5	70	94	60	80	2020	2
6	69	86	79	91	2018	3
8	79	81	77	80	2018	2
9	79	85	78	76	2020	2
10	66	78	69	94	2018	3
13	94	75	80	83	2020	2
14	69	79	79	80	2020	2
16	79	84	75	76	2018	2
17	68	80	66	89	2020	3
18	65	85	68	92	2020	3
20	71	78	67	83	2018	2
21	67	89	95	78	2019	2
23	64	76	67	82	2018	2
24	61	87	63	98	2020	3
26	93	93	76	90	2019	3
28	62	79	67	86	2020	3

Module 3

Apply data transformations on at least one of the variables. The purpose of this transformation should be one of the following reasons: to change the scale for better understanding of the variable, to convert a non-linear relation into a linear one, or to decrease the skewness and convert the distribution into a normal distribution.

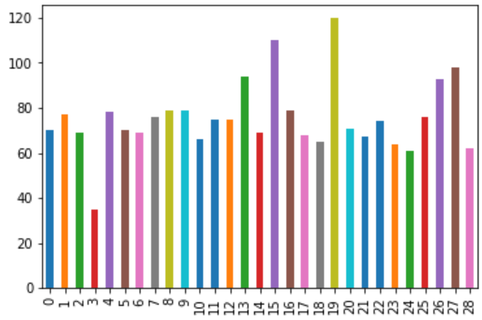
In [162]:
```
df4=pd.read_csv("C:\\Users\\Admin\\Desktop\\StudentPerformance.csv")
df4
```

Out[162]:

	Maths_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement offer count
0	70	93	61	82	2020	2
1	77	84	65	88	2019	3
2	69	84	68	93	2020	3
3	35	81	73	91	2019	3
4	78	95	73	96	2020	3
5	70	94	60	80	2020	2
6	69	86	79	91	2018	3
7	76	92	61	79	2019	2
8	79	81	77	80	2018	2
9	79	85	78	76	2020	2
10	66	78	69	94	2018	3
11	75	60	60	90	2018	3
12	75	81	74	88	2019	3
13	94	75	80	83	2020	2
14	69	79	79	80	2020	2
15	110	88	61	81	2018	2
16	79	84	75	76	2018	2
17	68	80	66	89	2020	3
18	65	85	68	92	2020	3
19	120	75	75	84	2018	2
20	71	78	67	83	2018	2
21	67	89	95	78	2019	2
22	74	77	72	81	2020	2
23	64	76	67	82	2018	2
24	61	87	63	98	2020	3
25	76	91	60	88	2019	3
26	93	93	76	90	2019	3
27	98	88	99	91	2019	3
28	62	79	67	86	2020	3

In [176]:
```
import matplotlib.pyplot as plt

df4['Maths_Score'].plot(kind = 'bar')
```

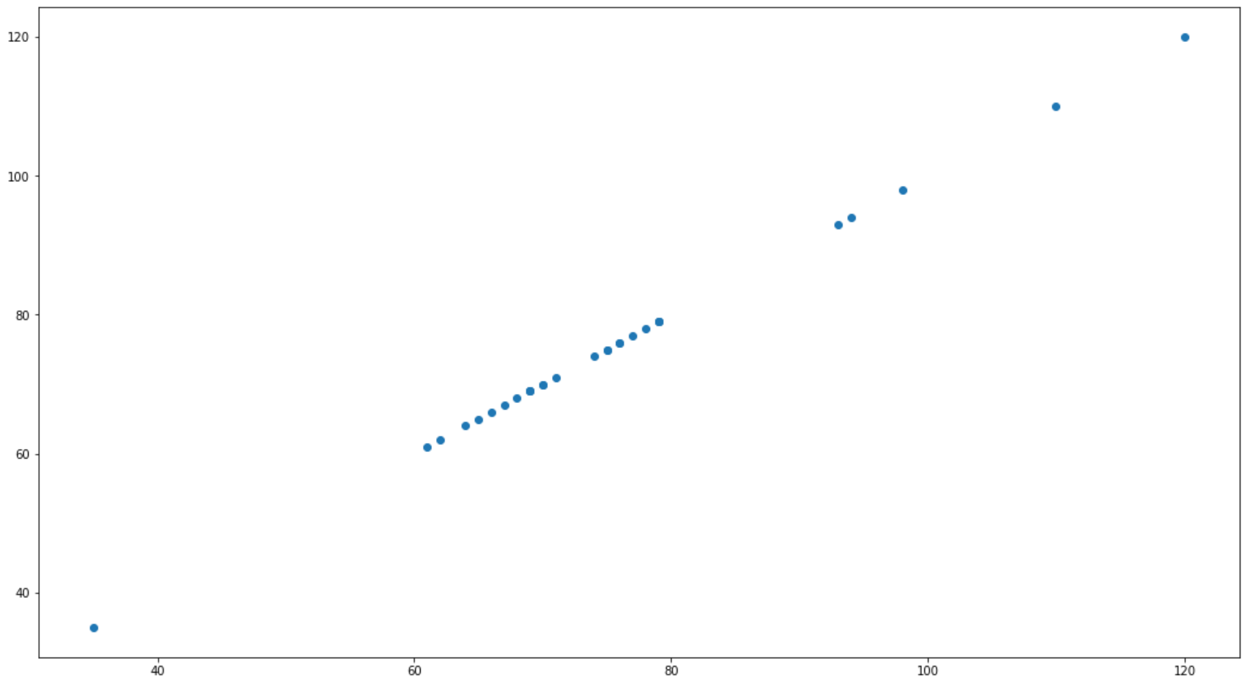
Out[176]: <matplotlib.axes._subplots.AxesSubplot at 0xd169d30>



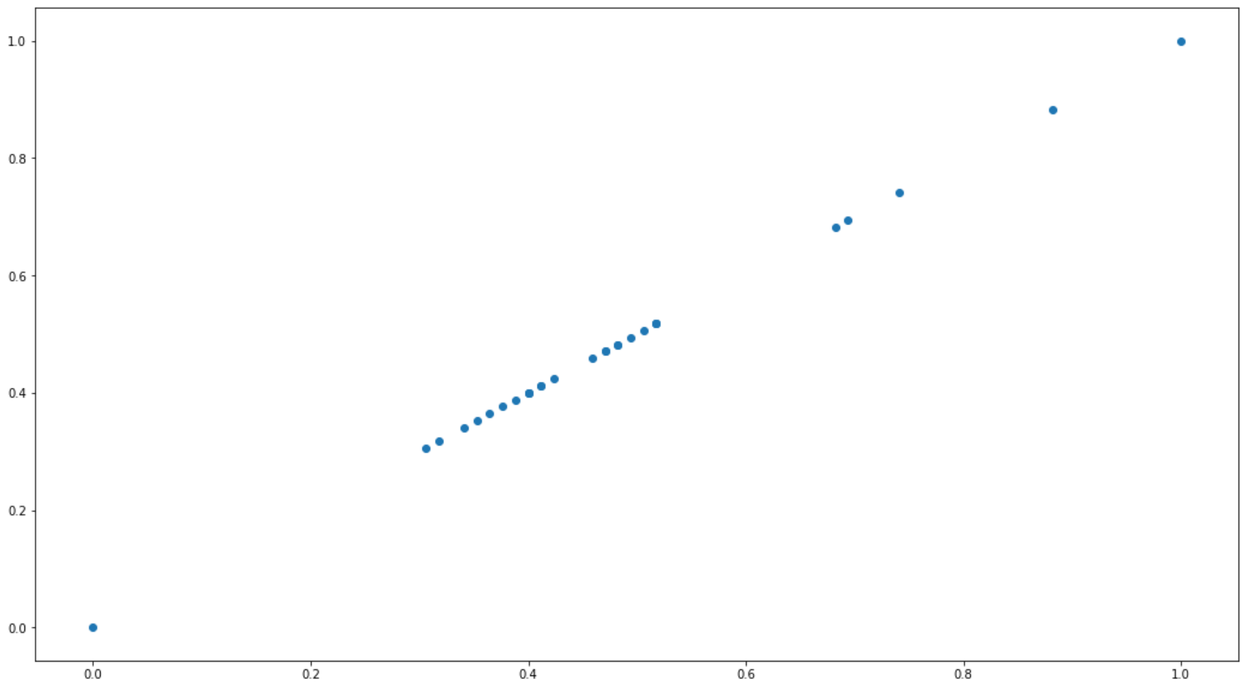
In [173]:
```
df_min_max_scaled = df4.copy()
colu=['Maths_Score']

# apply normalization techniques
df_min_max_scaled[colu] = (df_min_max_scaled[colu] - df_min_max_scaled[colu].min()) / (df_min_max_scaled[colu].max() - df_min_max_scaled[colu].min())

# view normalized data
print(df_min_max_scaled)
```

```
    Maths_Score  Reading_Score  Writing_Score  Placement_Score  \
0      0.411765             93             61               82
1      0.494118             84             65               88
2      0.400000             84             68               93
3      0.000000             81             73               91
4      0.505882             95             73               96
5      0.411765             94             60               80
6      0.400000             86             79               91
7      0.482353             92             61               79
8      0.517647             81             77               80
9      0.517647             85             78               76
10     0.364706             78             69               94
11     0.470588             60             60               90
12     0.470588             81             74               88
13     0.694118             75             80               83
14     0.400000             79             79               80
15     0.882353             88             61               81
16     0.517647             84             75               76
17     0.388235             80             66               89
18     0.352941             85             68               92
19     1.000000             75             75               84
20     0.423529             78             67               83
21     0.376471             89             95               78
22     0.458824             77             72               81
23     0.341176             76             67               82
24     0.305882             87             63               98
25     0.482353             91             60               88
26     0.682353             93             76               90
27     0.741176             88             99               91
28     0.317647             79             67               86

    Club_Join_Date  Placement offer count
0             2020                      2
1             2019                      3
2             2020                      3
3             2019                      3
4             2020                      3
5             2020                      2
6             2018                      3
7             2019                      2
8             2018                      2
9             2020                      2
10            2018                      3
11            2018                      3
12            2019                      3
13            2020                      2
14            2020                      2
15            2018                      2
16            2018                      2
17            2020                      3
18            2020                      3
19            2018                      2
20            2018                      2
21            2019                      2
22            2020                      2
23            2018                      2
24            2020                      3
25            2019                      3
26            2019                      3
27            2019                      3
28            2020                      3
```

In [180]:



In [181]:
```
#fig,ax=plt.subplots(figsize=(18,10))
##plt.show()
```



In [182]:
```
#Skewness
df4.skew(axis = 1, skipna = True)
```

Out[182]:
```
0     2.440742
1     2.440979
2     2.440759
3     2.440198
4     2.439189
5     2.440771
6     2.440408
7     2.440791
8     2.441093
9     2.440993
10    2.441065
11    2.441954
12    2.441064
13    2.439931
14    2.441540
15    2.437675
16    2.441193
17    2.441507
18    2.440855
19    2.436555
20    2.441830
21    2.439648
22    2.441761
23    2.442259
24    2.440098
25    2.440431
26    2.438927
27    2.437555
28    2.441952
dtype: float64
```

In [ ]:

Assignment 3

Part A Perform the following operations on any open source dataset (eg. data.csv) 1. Provide summary statistics (mean, median, minimum, maximum, standard deviation) for a dataset (age, income etc.) with numeric variables grouped by one of the qualitative (categorical) variable. For example, if your categorical variable is age groups and quantitative variable is income, then provide summary statistics of income grouped by the age groups. Create a list that contains a numeric value for each response to the categorical variable.

Commonly used Measures

1. Measure of Central Tendency
2. Measure of Dispertion

Measure of Central Tendency

1. Mean
2. Mode
3. Medain
4. Std Devaition
5. Minimum
6. Maxmimum

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

In [63]:
```
df=pd.read_csv("C:\\Users\\Admin\\Desktop\\Mall_Customers.csv")
df
```

Out[63]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13
15	16	Male	22	20	79
16	17	Female	35	21	35
17	18	Male	20	21	66
18	19	Male	52	23	29
19	20	Female	35	23	98
20	21	Male	35	24	35
21	22	Male	25	24	73
22	23	Female	46	25	5
23	24	Male	31	25	73
24	25	Female	54	28	14
25	26	Male	29	28	82
26	27	Female	45	28	32
27	28	Male	35	28	61
28	29	Female	40	29	31
29	30	Female	23	29	87
...	...	...	...	...	...
170	171	Male	40	87	13
171	172	Male	28	87	75
172	173	Male	36	87	10
173	174	Male	36	87	92
174	175	Female	52	88	13
175	176	Female	30	88	86
176	177	Male	58	88	15
177	178	Male	27	88	69
178	179	Male	59	93	14
179	180	Male	35	93	90
180	181	Female	37	97	32
181	182	Female	32	97	86
182	183	Male	46	98	15
183	184	Female	29	98	88
184	185	Female	41	99	39
185	186	Male	30	99	97
186	187	Female	54	101	24
187	188	Male	28	101	68
188	189	Female	41	103	17
189	190	Female	36	103	85
190	191	Female	34	103	23
191	192	Female	32	103	69
192	193	Male	33	113	8
193	194	Female	38	113	91
194	195	Female	47	120	16
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
199	200	Male	30	137	83

200 rows × 5 columns

In [4]:
```
df.mean()
```

Out[4]:
```
CustomerID                100.50
Age                        38.85
Annual Income (k$)         60.56
Spending Score (1-100)     50.20
dtype: float64
```

In [6]:
```
df.median()
```

Out[6]:
```
CustomerID                100.5
Age                        36.0
Annual Income (k$)         61.5
Spending Score (1-100)     50.0
dtype: float64
```

In [7]:
```
df.std()
```

Out[7]:
```
CustomerID                57.879185
Age                       13.969007
Annual Income (k$)        26.264721
Spending Score (1-100)    25.823522
dtype: float64
```

In [8]:
```
df.min()
```

Out[8]:
```
CustomerID                     1
Genre                     Female
Age                           18
Annual Income (k$)            15
Spending Score (1-100)         1
dtype: object
```

In [9]:
```
df.max()
```

Out[9]:
```
CustomerID                 200
Genre                     Male
Age                         70
Annual Income (k$)         137
Spending Score (1-100)      99
dtype: object
```

In [10]:
```
df["Age"].mean()
```

Out[10]:
```
38.85
```

In [11]:
```
df["Age"].mode()
```

Out[11]:
```
0    32
dtype: int64
```

In [12]:
```
df["Age"].median()
```

Out[12]:
```
36.0
```

In [13]:
```
df["Age"].std()
```

Out[13]:
```
13.969007331558883
```

In [15]:
```
gk=df.groupby(["Genre"])
```

In [17]:
```
gk.first()
```

Out[17]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
Genre				
Female	3	20	16	6
Male	1	19	15	39

part B

Write a Python program to display some basic statistical details like percentile, mean, standard deviation etc. of the species of 'Iris-setosa', 'Iris-versicolor' and 'Iris-versicolor' of iris.csv dataset. Provide the codes with outputs and explain everything that you do in this step.

In [38]:
```
csv_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
```

In [39]:
```
df_iris = pd.read_csv(csv_url, header = None)
```

In [40]:
```
col_names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width','Species']
```

In [41]:
```
df_iris = pd.read_csv(csv_url, names = col_names)
```

In [43]:
```
df_iris
```

Out[43]:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa
20	5.4	3.4	1.7	0.2	Iris-setosa
21	5.1	3.7	1.5	0.4	Iris-setosa
22	4.6	3.6	1.0	0.2	Iris-setosa
23	5.1	3.3	1.7	0.5	Iris-setosa
24	4.8	3.4	1.9	0.2	Iris-setosa
25	5.0	3.0	1.6	0.2	Iris-setosa
26	5.0	3.4	1.6	0.4	Iris-setosa
27	5.2	3.5	1.5	0.2	Iris-setosa
28	5.2	3.4	1.4	0.2	Iris-setosa
29	4.7	3.2	1.6	0.2	Iris-setosa
...	...	...	...	...	...
120	6.9	3.2	5.7	2.3	Iris-virginica
121	5.6	2.8	4.9	2.0	Iris-virginica
122	7.7	2.8	6.7	2.0	Iris-virginica
123	6.3	2.7	4.9	1.8	Iris-virginica
124	6.7	3.3	5.7	2.1	Iris-virginica
125	7.2	3.2	6.0	1.8	Iris-virginica
126	6.2	2.8	4.8	1.8	Iris-virginica
127	6.1	3.0	4.9	1.8	Iris-virginica
128	6.4	2.8	5.6	2.1	Iris-virginica
129	7.2	3.0	5.8	1.6	Iris-virginica
130	7.4	2.8	6.1	1.9	Iris-virginica
131	7.9	3.8	6.4	2.0	Iris-virginica
132	6.4	2.8	5.6	2.2	Iris-virginica
133	6.3	2.8	5.1	1.5	Iris-virginica
134	6.1	2.6	5.6	1.4	Iris-virginica
135	7.7	3.0	6.1	2.3	Iris-virginica
136	6.3	3.4	5.6	2.4	Iris-virginica
137	6.4	3.1	5.5	1.8	Iris-virginica
138	6.0	3.0	4.8	1.8	Iris-virginica
139	6.9	3.1	5.4	2.1	Iris-virginica
140	6.7	3.1	5.6	2.4	Iris-virginica
141	6.9	3.1	5.1	2.3	Iris-virginica
142	5.8	2.7	5.1	1.9	Iris-virginica
143	6.8	3.2	5.9	2.3	Iris-virginica
144	6.7	3.3	5.7	2.5	Iris-virginica
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [62]:
```
# Iris Species are of three types 1. Iris-setosa, 2. Iris-versicolor,3.Iris-virginica

gk=df_iris.groupby('Species')
```

In [52]:
```
gk.first()
```

Out[52]:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
Species				
Iris-setosa	5.1	3.5	1.4	0.2
Iris-versicolor	7.0	3.2	4.7	1.4
Iris-virginica	6.3	3.3	6.0	2.5

In [53]:
```
gk.describe()
```

Out[53]:

	Petal_Length								Petal_Width		...	Sepal_Length		Sepal_Width							
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std	min	25%	50%	75%	max
Species																					
Iris-setosa	50.0	1.464	0.173511	1.0	1.4	1.50	1.575	1.9	50.0	0.244	...	5.2	5.8	50.0	3.418	0.381024	2.3	3.125	3.4	3.675	4.4
Iris-versicolor	50.0	4.260	0.469911	3.0	4.0	4.35	4.600	5.1	50.0	1.326	...	6.3	7.0	50.0	2.770	0.313798	2.0	2.525	2.8	3.000	3.4
Iris-virginica	50.0	5.552	0.551895	4.5	5.1	5.55	5.875	6.9	50.0	2.026	...	6.9	7.9	50.0	2.974	0.322497	2.2	2.800	3.0	3.175	3.8

3 rows × 32 columns

In [56]:
```
#load all rows of Iris-setosa into iris_Set
iris_Set=(df_iris['Species'] == "Iris-setosa")
```

In [57]:
```
#To display basic statistical details like percentile,mean,std deviation etc for Iris-setosa using describe()
print("Iris-setosa")
```

```
Iris-setosa
```

In [58]:
```
print(df_iris[iris_Set].describe())
```

```
       Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
count      50.00000    50.000000     50.000000     50.00000
mean        5.00600     3.418000      1.464000      0.24400
std         0.35249     0.381024      0.173511      0.10721
min         4.30000     2.300000      1.000000      0.10000
25%         4.80000     3.125000      1.400000      0.20000
50%         5.00000     3.400000      1.500000      0.20000
75%         5.20000     3.675000      1.575000      0.30000
max         5.80000     4.400000      1.900000      0.60000
```

In [60]:
```
iris_Vir=(df_iris['Species'] == "Iris-virginica")
print(df_iris[iris_Vir].describe())
```

```
       Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
count      50.00000    50.000000     50.000000     50.00000
mean        6.58800     2.974000      5.552000      2.02600
std         0.63588     0.322497      0.551895      0.27465
min         4.90000     2.200000      4.500000      1.40000
25%         6.22500     2.800000      5.100000      1.80000
50%         6.50000     3.000000      5.550000      2.00000
75%         6.90000     3.175000      5.875000      2.30000
max         7.90000     3.800000      6.900000      2.50000
```

In [61]:
```
iris_Ver=(df_iris['Species'] == "Iris-versicolor")
print(df_iris[iris_Ver].describe())
```

```
       Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
count     50.000000    50.000000     50.000000    50.000000
mean       5.936000     2.770000      4.260000     1.326000
std        0.516171     0.313798      0.469911     0.197753
min        4.900000     2.000000      3.000000     1.000000
25%        5.600000     2.525000      4.000000     1.200000
50%        5.900000     2.800000      4.350000     1.300000
75%        6.300000     3.000000      4.600000     1.500000
max        7.000000     3.400000      5.100000     1.800000
```

In [ ]:

Data Analytics I Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing (https://www.kaggle.com/c/boston-housing)). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset. The objective is to predict the value of prices of the house using the given features.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#Step 2: Import the Boston Housing dataset
from sklearn.datasets import load_boston
boston = load_boston()
```

In [10]:
```
data = pd.DataFrame(boston.data)
```

In [28]:
```
data.shape
```

Out[28]:
```
(506, 14)
```

In [11]:
```
data.columns = boston.feature_names
data.head()
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [12]:
```
data['PRICE'] = boston.target
```

In [13]:
```
data.isnull().sum()
```

Out[13]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

In [14]:
```
x = data.drop(['PRICE'], axis = 1)
y = data['PRICE']
```

In [16]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest =train_test_split(x, y, test_size =0.2,random_state = 0)
```

In [17]:
```
import sklearn
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
model=lm.fit(xtrain, ytrain)
```

In [25]:
```
lm.intercept_
```

Out[25]:
```
38.138692713393205
```

In [26]:
```
lm.coef_
```

Out[26]:
```
array([-1.18410318e-01,  4.47550643e-02,  5.85674689e-03,  2.34230117e+00,
       -1.61634024e+01,  3.70135143e+00, -3.04553661e-03, -1.38664542e+00,
        2.43784171e-01, -1.09856157e-02, -1.04699133e+00,  8.22014729e-03,
       -4.93642452e-01])
```

In [18]:
```
ytrain_pred = lm.predict(xtrain)
ytest_pred = lm.predict(xtest)
```

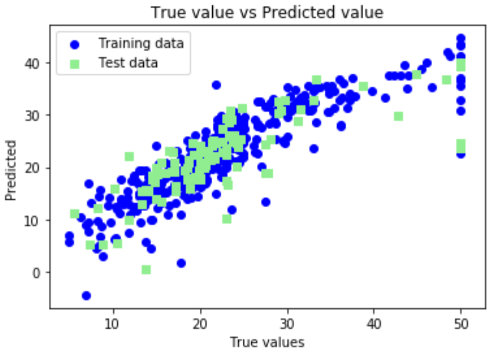
In [19]:
```
df=pd.DataFrame(ytrain_pred,ytrain)
df=pd.DataFrame(ytest_pred,ytest)
```

In [20]:
```
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(ytest, ytest_pred)
print(mse)
mse = mean_squared_error(ytrain_pred,ytrain)
print(mse)
```

```
33.450708967691185
19.330019357349375
```

In [21]:
```
mse = mean_squared_error(ytest, ytest_pred)
```

In [23]:
```
plt.scatter(ytrain ,ytrain_pred,c='blue',marker='o',label='Training data')
plt.scatter(ytest,ytest_pred ,c='lightgreen',marker='s',label='Test data')
plt.xlabel('True values')
plt.ylabel('Predicted')
plt.title("True value vs Predicted value")
plt.legend(loc= 'upper left')
#plt.hlines(y=0,xmin=0,xmax=50)
plt.plot()
plt.show()
```



In [ ]:
```
.
```

Assignment 5

1. Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset.
2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset..

In [10]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [11]:
```
dataset=pd.read_csv("C:\\Users\\Admin\\Desktop\\Social_Network_Ads.csv")
```

In [12]:
```
print(dataset)
```

```
      User ID  Gender  Age  EstimatedSalary  Purchased
0    15624510    Male   19            19000          0
1    15810944    Male   35            20000          0
2    15668575  Female   26            43000          0
3    15603246  Female   27            57000          0
4    15804002    Male   19            76000          0
5    15728773    Male   27            58000          0
6    15598044  Female   27            84000          0
7    15694829  Female   32           150000          1
8    15600575    Male   25            33000          0
9    15727311  Female   35            65000          0
10   15570769  Female   26            80000          0
11   15606274  Female   26            52000          0
12   15746139    Male   20            86000          0
13   15704987    Male   32            18000          0
14   15628972    Male   18            82000          0
15   15697686    Male   29            80000          0
16   15733883    Male   47            25000          1
17   15617482    Male   45            26000          1
18   15704583    Male   46            28000          1
19   15621083  Female   48            29000          1
20   15649487    Male   45            22000          1
21   15736760  Female   47            49000          1
22   15714658    Male   48            41000          1
23   15599081  Female   45            22000          1
24   15705113    Male   46            23000          1
25   15631159    Male   47            20000          1
26   15792818    Male   49            28000          1
27   15633531  Female   47            30000          1
28   15744529    Male   29            43000          0
29   15669656    Male   31            18000          0
..        ...     ...  ...              ...        ...
370  15611430  Female   60            46000          1
371  15774744    Male   60            83000          1
372  15629885  Female   39            73000          0
373  15708791    Male   59           130000          1
374  15793890  Female   37            80000          0
375  15646091  Female   46            32000          1
376  15596984  Female   46            74000          0
377  15800215  Female   42            53000          0
378  15577806    Male   41            87000          1
379  15749381  Female   58            23000          1
380  15683758    Male   42            64000          0
381  15670615    Male   48            33000          1
382  15715622  Female   44           139000          1
383  15707634    Male   49            28000          1
384  15806901  Female   57            33000          1
385  15775335    Male   56            60000          1
386  15724150  Female   49            39000          1
387  15627220    Male   39            71000          0
388  15672330    Male   47            34000          1
389  15668521  Female   48            35000          1
390  15807837    Male   48            33000          1
391  15592570    Male   47            23000          1
392  15748589  Female   45            45000          1
393  15635893    Male   60            42000          1
394  15757632  Female   39            59000          0
395  15691863  Female   46            41000          1
396  15706071    Male   51            23000          1
397  15654296  Female   50            20000          1
398  15755018    Male   36            33000          0
399  15594041  Female   49            36000          1

[400 rows x 5 columns]
```

In [13]:
```
dataset.isnull().sum()
```

Out[13]:
```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

In [14]:
```
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

In [17]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [18]:
```
print(X_train[:3])
print('-'*15)
print(y_train[:3])
print('-'*15)
print(X_test[:3])
print('-'*15)
print(y_test[:3])
```

```
[[    44  39000]
 [    32 120000]
 [    38  50000]]
---------------
[0 1 0]
---------------
[[   30 87000]
 [   38 50000]
 [   35 75000]]
---------------
[0 0 0]
```

In [19]:
```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning: Data with input dtype int6
4 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [20]:
```
print(X_train[:3])
print('-'*15)
print(X_test[:3])
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]]
---------------
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]]
```

In [21]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0, solver='lbfgs' )
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

print(X_test[:10])
```

```
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]
 [-0.80480212  0.27301877]
 [-0.30964085 -0.5677824 ]
 [-1.10189888 -1.43757673]
 [-0.70576986 -1.58254245]
 [-0.21060859  2.15757314]
 [-1.99318916 -0.04590581]
 [ 0.8787462  -0.77073441]]
```

In [22]:
```
print('-'*15)
print(y_pred[:10])
```

```
---------------
[0 0 0 0 0 0 0 1 0 1]
```

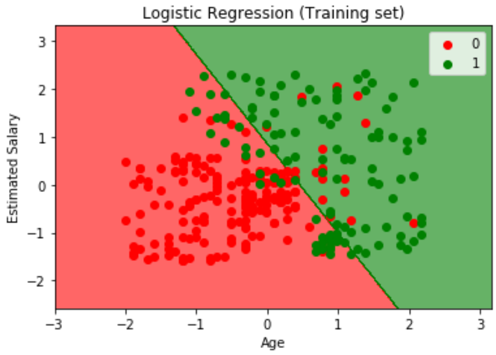
In [23]:
```
print(y_pred[:20])
print(y_test[:20])
```

```
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0]
```

In [25]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[65  3]
 [ 8 24]]
```

In [26]:
```
# Visualizing the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.6, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



In [ ]:

Assignment 6 Data Analytics III 1. Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset. II. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

Step 1: Importing the Libraries #As always, the first step will always include importing the libraries which are the NumPy, Pandas and the Matplotlib.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Step 2: Importing the dataset In this step, we shall import the Iris Flower dataset which is stored in my github repository as IrisDataset.csv and save it to the variable dataset.

In [32]:
```
dataset = pd.read_csv('https://raw.githubusercontent.com/mk-gurucharan/Classification/master/IrisDataset.csv')
```

In [19]:
```
dataset.head()
```

Out[19]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [22]:
```
gk=dataset.groupby('species')
```

In [23]:
```
gk.first()
```

Out[23]:

	sepal_length	sepal_width	petal_length	petal_width
species				
setosa	5.1	3.5	1.4	0.2
versicolor	7.0	3.2	4.7	1.4
virginica	6.3	3.3	6.0	2.5

Step 3: After this, we assign the 4 independent variables to X and the dependent variable 'species' to Y. The first 5 rows of the dataset are displayed.

In [24]:
```
X = dataset.iloc[:,:4].values
y = dataset['species'].values
```

Step 4: Splitting the dataset into the Training set and Test set Once we have obtained our data set, we have to split the data into the training set and the test set. In this data set, there are 150 rows with 50 rows of each of the 3 classes. As each class is given in a continuous order, we need to randomly split the dataset. Here, we have the test_size=0.2, which means that 20% of the dataset will be used for testing purpose as the test set and the remaining 80% will be used as the training set for training the Naive Bayes classification model.

In [25]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

Step 5: Feature Scaling The dataset is scaled down to a smaller range using the Feature Scaling option. In this, both the X_train and X_test values are scaled down to smaller values to improve the speed of the program.

In [26]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Step 6: Training the Naive Bayes Classification model on the Training Set In this step, we introduce the class GaussianNB that is used from the sklearn.naive_bayes library. Here, we have used a Gaussian model, there are several other models such as Bernoulli, Categorical and Multinomial. Here, we assign the GaussianNB class to the variable classifier and fit the X_train and y_train values to it for training purpose.

In [27]:
```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Out[27]:
```
GaussianNB(priors=None)
```

Step 7: Predicting the Test set results Once the model is trained, we use the the classifier.predict() to predict the values for the Test set and the values predicted are stored to the variable y_pred.

In [28]:
```
y_pred = classifier.predict(X_test)
y_pred
```

Out[28]:
```
array(['versicolor', 'setosa', 'setosa', 'virginica', 'virginica',
       'versicolor', 'setosa', 'versicolor', 'virginica', 'virginica',
       'setosa', 'virginica', 'versicolor', 'virginica', 'virginica',
       'versicolor', 'setosa', 'setosa', 'setosa', 'setosa', 'virginica',
       'versicolor', 'versicolor', 'setosa', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'setosa'], dtype='<U10')
```

Step 8: Confusion Matrix and Accuracy This is a step that is mostly used in classification techniques. In this, we see the Accuracy of the trained model and plot the confusion matrix. The confusion matrix is a table that is used to show the number of correct and incorrect predictions on a classification problem when the real values of the Test Set are known. It is of the format

In [29]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
cm
```

```
Accuracy :  1.0
```

Out[29]:
```
array([[10,  0,  0],
       [ 0,  7,  0],
       [ 0,  0, 13]], dtype=int64)
```

From the above confusion matrix, we infer that, out of 30 test set data, 30 were correctly classified . This gives us a high accuracy of 100%

Step 9: Comparing the Real Values with Predicted Values In this step, a Pandas DataFrame is created to compare the classified values of both the original Test set (y_test) and the predicted results (y_pred).

In [30]:
```
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
```

Out[30]:

	Real Values	Predicted Values
0	versicolor	versicolor
1	setosa	setosa
2	setosa	setosa
3	virginica	virginica
4	virginica	virginica
5	versicolor	versicolor
6	setosa	setosa
7	versicolor	versicolor
8	virginica	virginica
9	virginica	virginica
10	setosa	setosa
11	virginica	virginica
12	versicolor	versicolor
13	virginica	virginica
14	virginica	virginica
15	versicolor	versicolor
16	setosa	setosa
17	setosa	setosa
18	setosa	setosa
19	setosa	setosa
20	virginica	virginica
21	versicolor	versicolor
22	versicolor	versicolor
23	setosa	setosa
24	virginica	virginica
25	virginica	virginica
26	virginica	virginica
27	virginica	virginica
28	virginica	virginica
29	setosa	setosa

In [ ]:

Assignment 8Data Visualization I

Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data. Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram

Assignment 9Data Visualization II

1. Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (Column names : 'sex' and 'age') Write observations on the inference from the above statistics.

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

import matplotlib.pyplot as plt
import seaborn as sns
dataset=pd.read_csv("C:\\Users\\Admin\\Desktop\\dataset\\Titanic-Dataset.csv")
```

In [8]:
```
dataset.head()
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

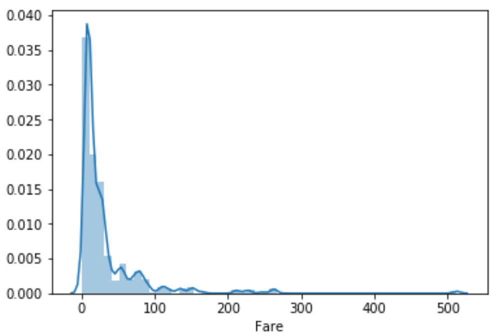
Distributional plots, as the name suggests are type of plots that show the statistical distribution of data. The distplot() shows the histogram distribution of data for a single column. The column name is passed as a parameter to the distplot() function. Let's see how the price of the ticket for each passenger is distributed

In [ ]:
```
Distribution Plots
a. Distplot
b. jointplot
c. Pairplot
d. Rugplot
These plots help us to visualize the distribution of data. We can use these plots to understand the mean, median, range, vari
ance, deviation, etc of the data.
a. Distplot
Dist plot gives us the histogram of the selected continuous variable.
It is an example of a univariate analysis.
We can change the number of bins i.e. number of vertical bars in a histogram
```

In [10]:
```
sns.distplot(dataset['Fare'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidime
nsional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an arr
ay index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

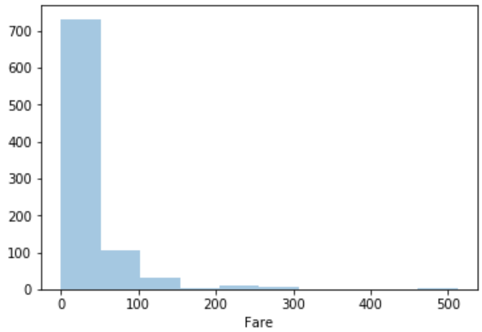
Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x18ab610>
```



OUTPUT:You can see that most of the tickets have been solved between 0-50 dollars. The line that you see represents the kernel density estimation.

In [12]:
```
#Kernal density estimation
#You can also pass the value for the bins parameter in order to see more or less details in the graph.
sns.distplot(dataset['Fare'], kde=False, bins=10)
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x19ef270>

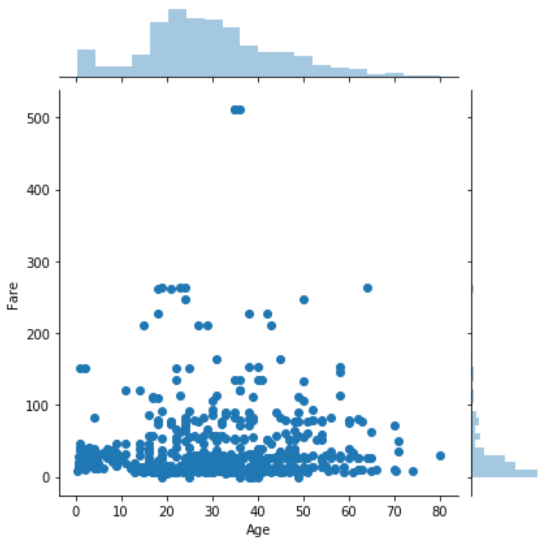


b. Joint Plot It is the combination of the distplot of two variables. It is an example of bivariate analysis. We additionally obtain a scatter plot between the variable to reflecting their linear relationship. We can customize the scatter plot into a hexagonal plot, where, more the color intensity, the more will be the number of observations.

In [14]:
```
sns.jointplot(x='Age', y='Fare', data=dataset)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidime
nsional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an arr
ay index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

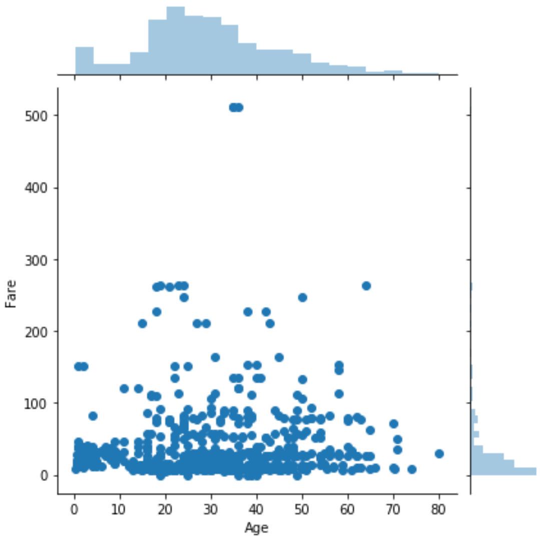
Out[14]: <seaborn.axisgrid.JointGrid at 0x49f0670>



In [16]:
```
sns.jointplot(x = df['Age'], y = df['Fare'], kind = 'scatter')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidime
nsional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an arr
ay index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

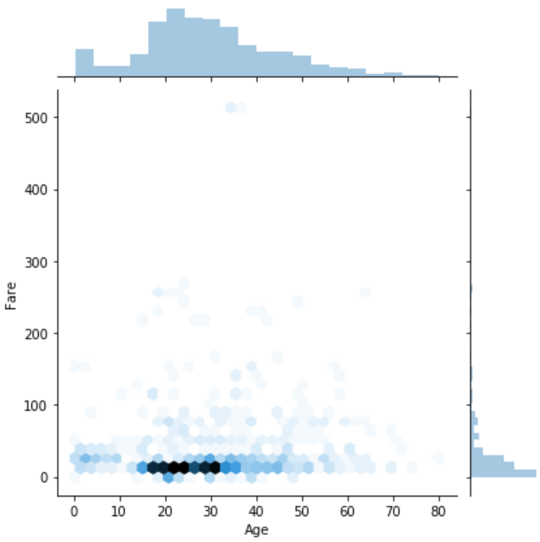
Out[16]: <seaborn.axisgrid.JointGrid at 0x4ac27b0>



In [17]:
```
sns.jointplot(x = df['Age'], y = df['Fare'], kind = 'hex')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidime
nsional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an arr
ay index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

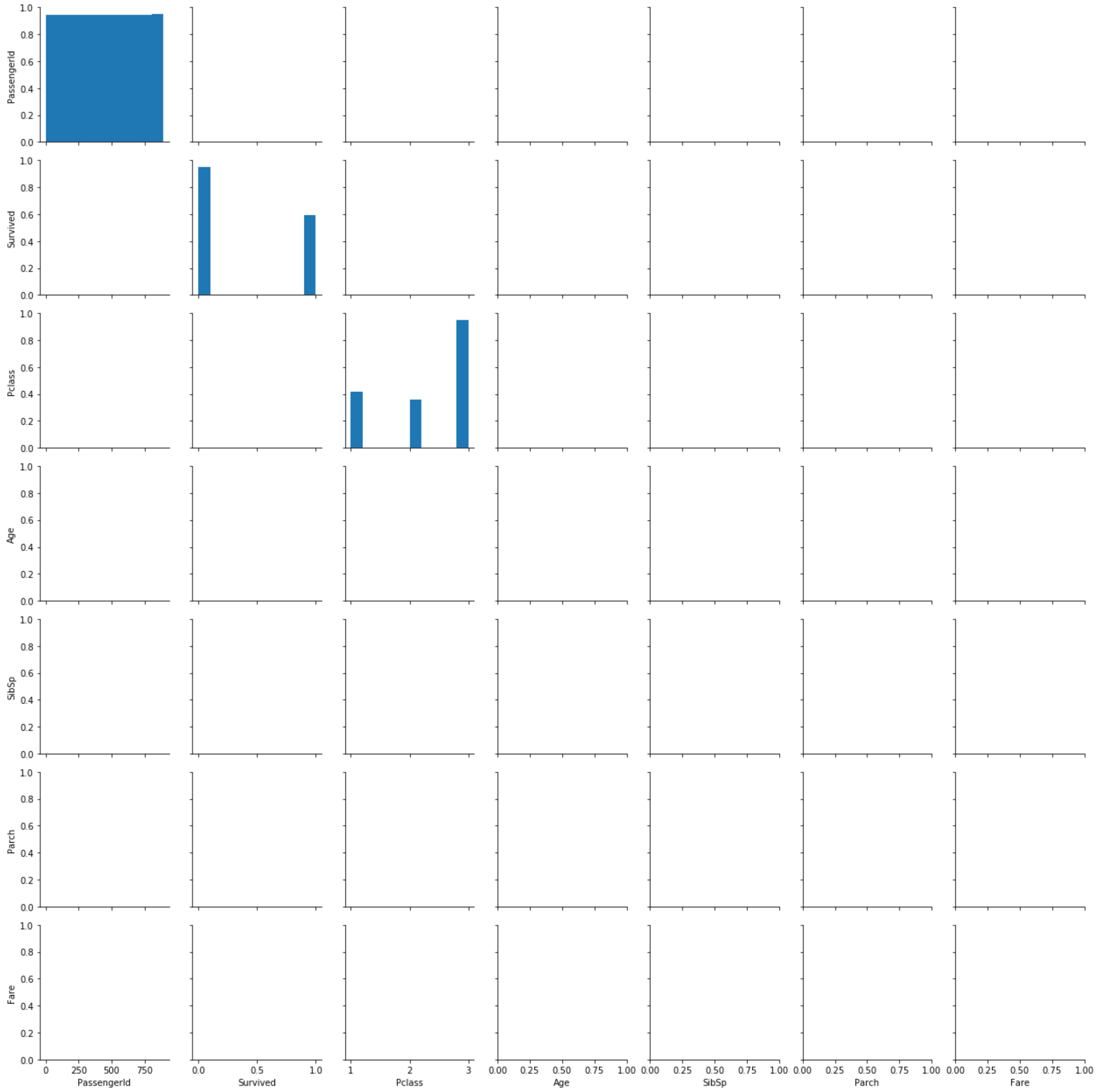
Out[17]: <seaborn.axisgrid.JointGrid at 0xd15d6f0>



We can see that there no appropriate linear relation between age and fare. kind = 'hex' provides the hexagonal plot and kind = 'reg' provides a regression line on the graph.

In [19]:
```
sns.pairplot(dataset)
```

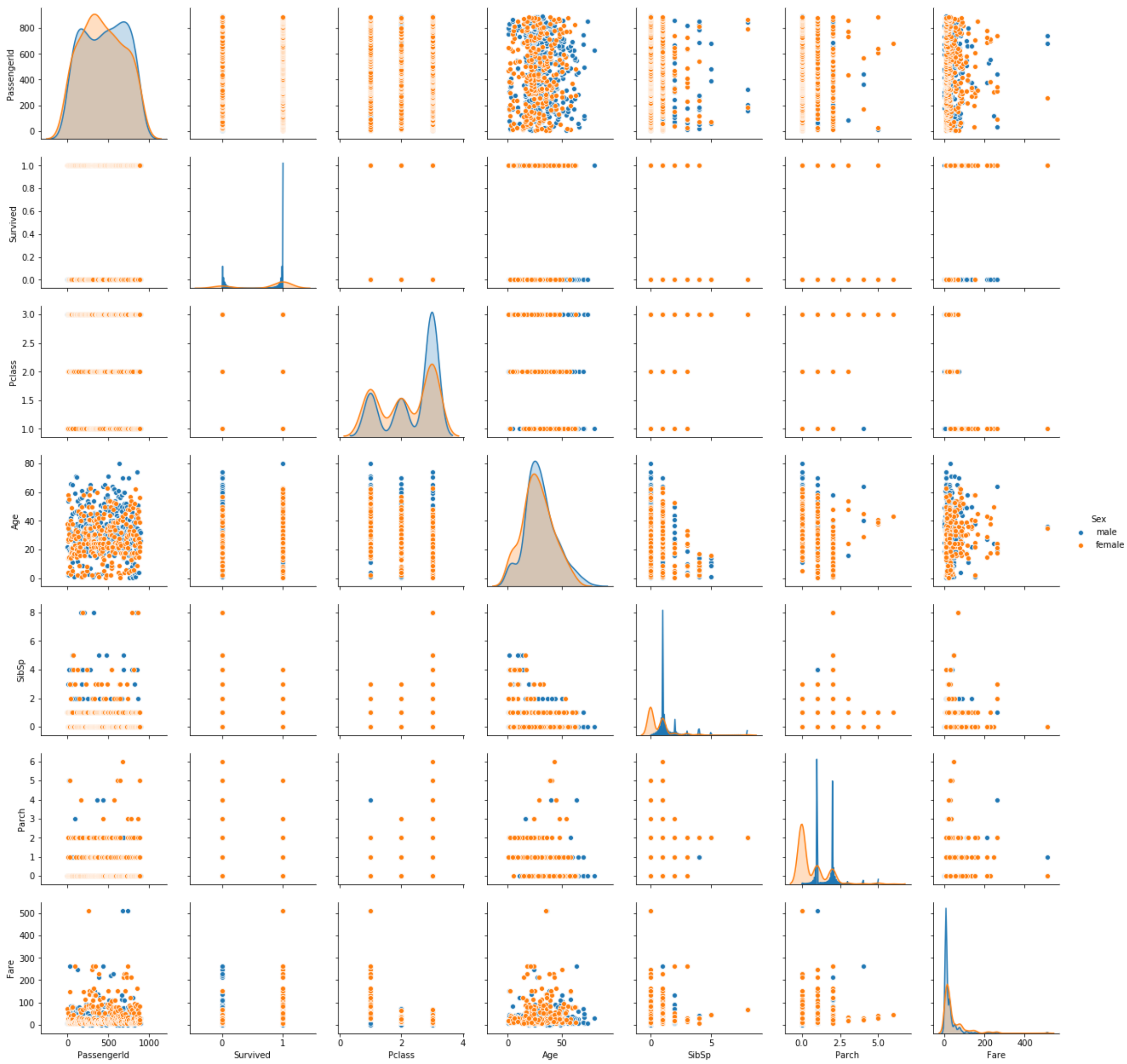
```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\_methods.py:32: RuntimeWarning: invalid value encountered in reduce
  return umr_minimum(a, axis, None, out, keepdims, initial)
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\_methods.py:28: RuntimeWarning: invalid value encountered in reduce
  return umr_maximum(a, axis, None, out, keepdims, initial)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-e506b35fe370> in <module>()
----> 1 sns.pairplot(dataset)

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py in pairplot(data, hue, hue_order, palette, vars, x_vars, y_var
s, kind, diag_kind, markers, height, aspect, dropna, plot_kws, diag_kws, grid_kws, size)
   2105     if grid.square_grid:
   2106         if diag_kind == "hist":
-> 2107             grid.map_diag(plt.hist, **diag_kws)
   2108         elif diag_kind == "kde":
   2109             diag_kws.setdefault("shade", True)

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py in map_diag(self, func, **kwargs)
   1397                     color = fixed_color
   1398 
-> 1399                 func(data_k, label=label_k, color=color, **kwargs)
   1400 
   1401             self._clean_axis(ax)

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\pyplot.py in hist(x, bins, range, density, weights, cumulative, bottom,
histtype, align, orientation, rwidth, log, color, label, stacked, normed, hold, data, **kwargs)
   3135                       histtype=histtype, align=align, orientation=orientation,
   3136                       rwidth=rwidth, log=log, color=color, label=label,
-> 3137                       stacked=stacked, normed=normed, data=data, **kwargs)
   3138     finally:
   3139         ax._hold = washold

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\__init__.py in inner(ax, *args, **kwargs)
   1865                         "the Matplotlib list!)" % (label_namer, func.__name__),
   1866                         RuntimeWarning, stacklevel=2)
-> 1867             return func(ax, *args, **kwargs)
   1868 
   1869         inner.__doc__ = _add_data_doc(inner.__doc__,

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py in hist(***failed resolving arguments***)
   6637             # this will automatically overwrite bins,
   6638             # so that each histogram uses the same bins
-> 6639             m, bins = np.histogram(x[i], bins, weights=w[i], **hist_kwargs)
   6640             m = m.astype(float)  # causes problems later if it's an int
   6641             if mlast is None:

C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\histograms.py in histogram(a, bins, range, normed, weights, density)
    700     a, weights = _ravel_and_check_weights(a, weights)
    701 
--> 702     bin_edges, uniform_bins = _get_bin_edges(a, bins, range, weights)
    703 
    704     # Histogram is an integer or a float array depending on the weights.

C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\histograms.py in _get_bin_edges(a, bins, range, weights)
    353             raise ValueError('`bins` must be positive, when an integer')
    354 
--> 355         first_edge, last_edge = _get_outer_edges(a, range)
    356 
    357     elif np.ndim(bins) == 1:

C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\histograms.py in _get_outer_edges(a, range)
    240         if first_edge > last_edge:
    241             raise ValueError(
--> 242                 'max must be larger than min in range parameter.')
    243         if not (np.isfinite(first_edge) and np.isfinite(last_edge)):
    244             raise ValueError(

ValueError: max must be larger than min in range parameter.
```


PassengerId
Survived
Pclass
Age
SibSp
Parch
Fare

In [21]:
```
sns.pairplot(dataset, hue='Sex')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidime
nsional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an arr
ay index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:448: RuntimeWarning: invalid value encountered in
greater
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:448: RuntimeWarning: invalid value encountered in
less
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
```

Out[21]: <seaborn.axisgrid.PairGrid at 0xefe5730>

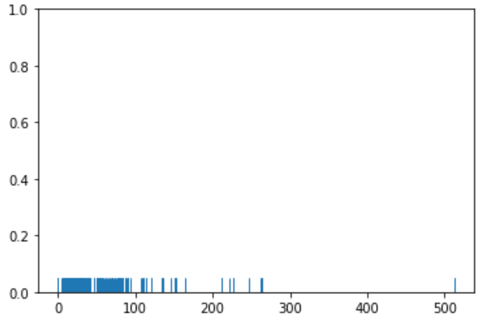


OUTPUT:In the output you can see the information about the males in orange and the information about the female in blue (as shown in the legend). From the joint plot on the top left, you can clearly see that among the surviving passengers, the majority were female.

d.Rug Plot It draws a dash mark instead of a uniform distribution as in distplot. It is an example of a univariate analysis.

In [22]:
```
sns.rugplot(dataset['Fare'])
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0xd6670f0>



OUTPUT:From the output, you can see that as was the case with the distplot(), most of the instances for the fares have values between 0 and 100.

Categorical plots, as the name suggests are normally used to plot categorical data. The categorical plots plot the values in the categorical column against another categorical column or a numeric column. Let's see some of the most commonly used categorical data.

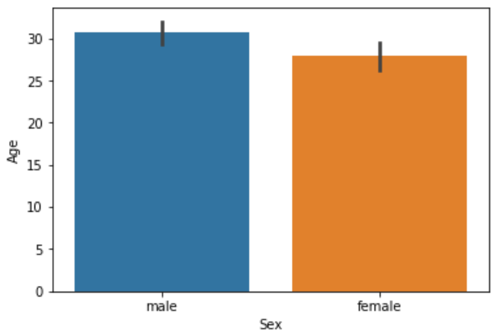
a.The Bar Plot

It is an example of bivariate analysis. On the x-axis, we have a categorical variable and on the y-axis, we have a continuous variable. The barplot() is used to display the mean value for each value in a categorical column, against a numeric column. The first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset. For instance, if you want to know the mean value of the age of the male and female passengers, you can use the bar plot as follows.

In [24]:
```
sns.barplot(x='Sex', y='Age', data=dataset)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidime
nsional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an arr
ay index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0xf6980b0>

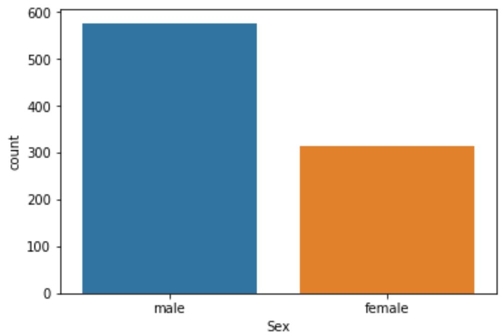


OUTPUT:From the output, you can clearly see that the average age of male passengers is just less than 40 while the average age of female passengers is around 33.

b.Count Plot It counts the number of occurrences of categorical variables. It is an example of a univariate analysis. The count plot is similar to the bar plot, however it displays the count of the categories in a specific column. For instance, if we want to count the number of males and women passenger we can do so using count plot as follows:

In [25]:
```
sns.countplot(x='Sex', data=dataset)
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0xf91a810>

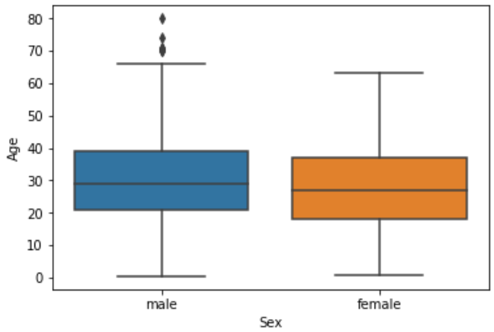


c. Box Plot It is a 5 point summary plot. It gives the information about the maximum, minimum, mean, first quartile, and third quartile of a continuous variable. Also, it equips us with knowledge of outliers. We can plot this for a single continuous variable or can analyze different categorical variables based on a continuous variable.

Now let's plot a box plot that displays the distribution for the age with respect to each gender. You need to pass the categorical column as the first parameter (which is sex in our case) and the numeric column (age in our case) as the second parameter. Finally, the dataset is passed as the third parameter, take a look at the following script:

In [26]:
```
sns.boxplot(x='Sex', y='Age', data=dataset)
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0xf66c730>



OUTPUT:Let's try to understand the box plot for female. The first quartile starts at around 5 and ends at 22 which means that 25% of the passengers are aged between 5 and 25. The second quartile starts at around 23 and ends at around 32 which means that 25% of the passengers are aged between 23 and 32. Similarly, the third quartile starts and ends between 34 and 42, hence 25% passengers are aged within this range and finally the fourth or last quartile starts at 43 and ends around 65.

1. Using hue parameter:

While the points are plotted in two dimensions, another dimension can be added to the plot by coloring the points according to a third variable.

Syntax:

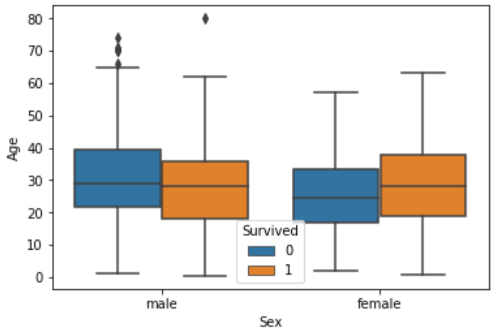
seaborn.boxplot(x, y, hue, data);

if you want to see the box plots of forage of passengers of both genders, along with the information about whether or not they survived, you can pass the survived as value to the hue parameter as shown below:

OUTPUT:Now in addition to the information about the age of each gender, you can also see the distribution of the passengers who survived. For instance, you can see that among the male passengers, on average more younger people survived as compared to the older ones. Similarly, you can see that the variation among the age of female passengers who did not survive is much greater than the age of the surviving female passengers.

In [27]:
```
sns.boxplot(x='Sex', y='Age', data=dataset, hue="Survived")
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0xf8bb4f0>

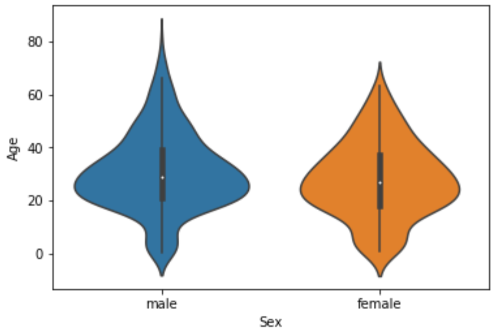


d.VIOLIN PLOT The violin plot is similar to the box plot, however, the violin plot allows us to display all the components that actually correspond to the data point. The violinplot() function is used to plot the violin plot. Like the box plot, the first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset.

In [29]:
```
sns.violinplot(x='Sex', y='Age', data=dataset)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidime
nsional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an arr
ay index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0xf986090>



OUtPUT:You can see from the figure above that violin plots provide much more information about the data as compared to the box plot. Instead of plotting the quartile, the violin plot allows us to see all the components that actually correspond to the data. The area where the violin plot is thicker has a higher number of instances for the age. For instance, from the violin plot for males, it is clearly evident that the number of passengers with age between 20 and 40 is higher than all the rest of the age brackets.

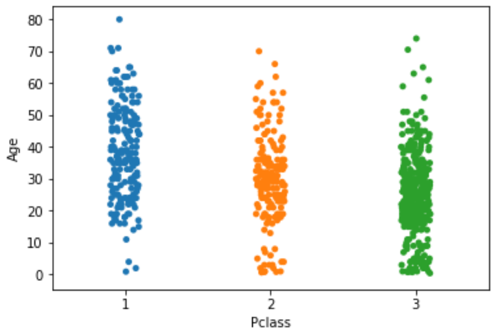
Advanced Plots As the name suggests, they are advanced because they ought to fuse the distribution and categorical encodings.

a.Strip Plot

It's a plot between a continuous variable and a categorical variable. It plots as a scatter plot but supplementarily uses categorical encodings of the categorical variable. The stripplot() function is used to plot the violin plot. Like the box plot, the first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset. Look at the following script:

In [31]:
```
sns.stripplot(y = dataset['Age'], x = dataset['Pclass'])
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x49ad1b0>

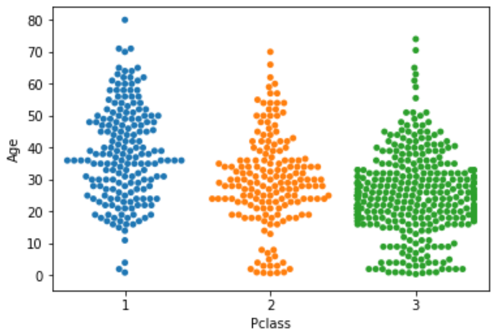


OUTPUT:We can observe that in class 1 and class 2, children around 10 years are not present and the people having age above 60 are mostly accommodated in class 1.

In [ ]:
```
b. Swarm Plot
It is the combination of a strip plot and a violin plot.
Along with the number of data points, it also provides their respective distribution.
```

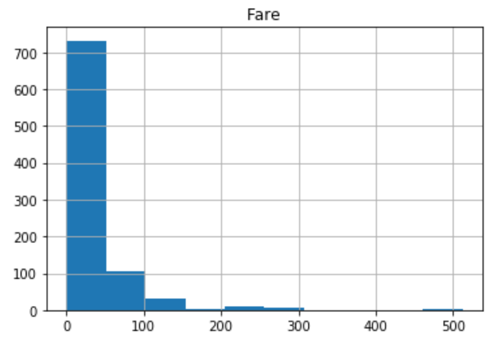
In [34]:
```
sns.swarmplot(y = dataset['Age'], x = dataset['Pclass'])
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0xe35c870>



In [35]:
```
dataset.hist('Fare')
```

Out[35]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0F892A50>]],
      dtype=object)
```



In [ ]:

Assignment 10

Data Visualization III Download the Iris flower dataset or any other dataset into a DataFrame. (eg https://archive.ics.uci.edu/ml/datasets/Iris (https://archive.ics.uci.edu/ml/datasets/Iris)). Scan the dataset and give the inference as:

1. How many features are there and what are their types (e.g., numeric, nominal)?
2. Create a histogram for each feature in the dataset to illustrate the feature distributions.
3. Create a boxplot for each feature in the dataset. Compare distributions and identify outliers.

In [4]:
```
import numpy as np
import pandas as pd
```

In [5]:
```
csv_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
df = pd.read_csv(csv_url, header = None)
col_names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width','Species']
df = pd.read_csv(csv_url, names = col_names)
```

In [6]:
```
df.head()
```

Out[6]:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Q1. How many features are there and what are their types?

In [7]:
```
# to determine the length of lists in a pandas dataframe column
column = len(list(df))
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
Sepal_Length    150 non-null float64
Sepal_Width     150 non-null float64
Petal_Length    150 non-null float64
Petal_Width     150 non-null float64
Species         150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.3+ KB
```

Hence the dataset contains 4 numerical columns and 1 object column

In [9]:
```
np.unique(df['Species'])
```

Out[9]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

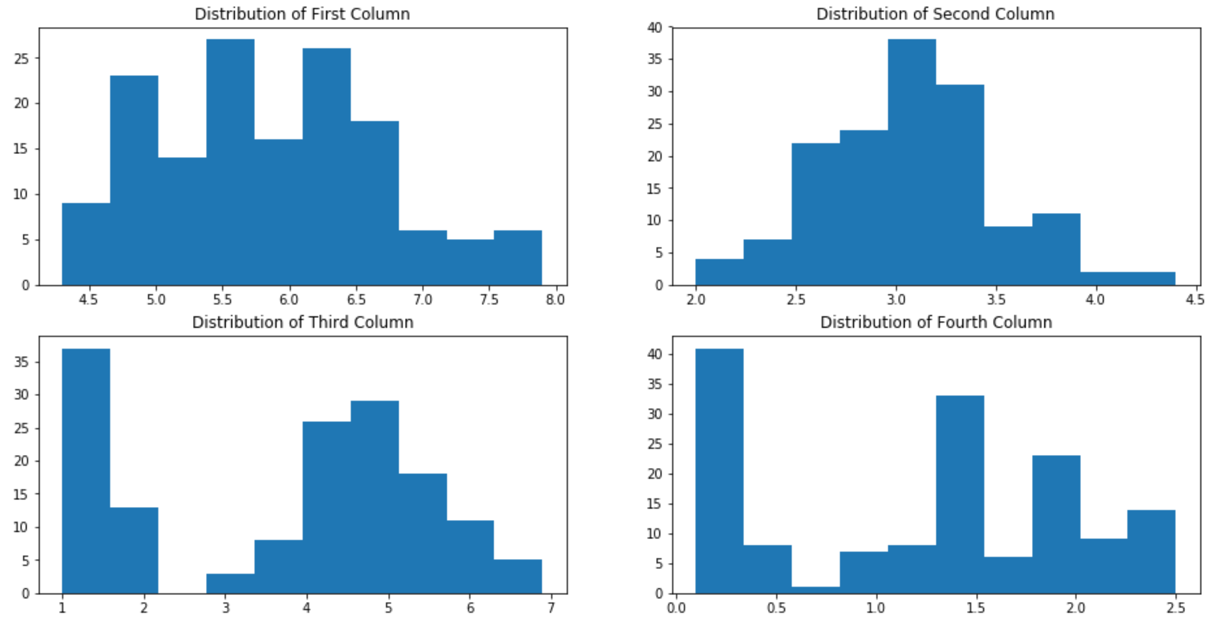
Q2. Data Visualization-Create a histogram for each feature in the dataset to illustrate the feature distributions. Plot each histogram.

The Seaborn library is built on top of Matplotlib and offers many advanced data visualization capabilities.

Though, the Seaborn library can be used to draw a variety of charts such as matrix plots, grid plots, regression plots etc.,

In [ ]:
```
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

In [12]:
```
fig, axes = plt.subplots(2, 2, figsize=(16, 8))


axes[0,0].set_title("Distribution of First Column")
axes[0,0].hist(df["Sepal_Length"]);

axes[0,1].set_title("Distribution of Second Column")
axes[0,1].hist(df["Sepal_Width"]);

axes[1,0].set_title("Distribution of Third Column")
axes[1,0].hist(df["Petal_Length"]);

axes[1,1].set_title("Distribution of Fourth Column")
axes[1,1].hist(df["Petal_Width"]);
```



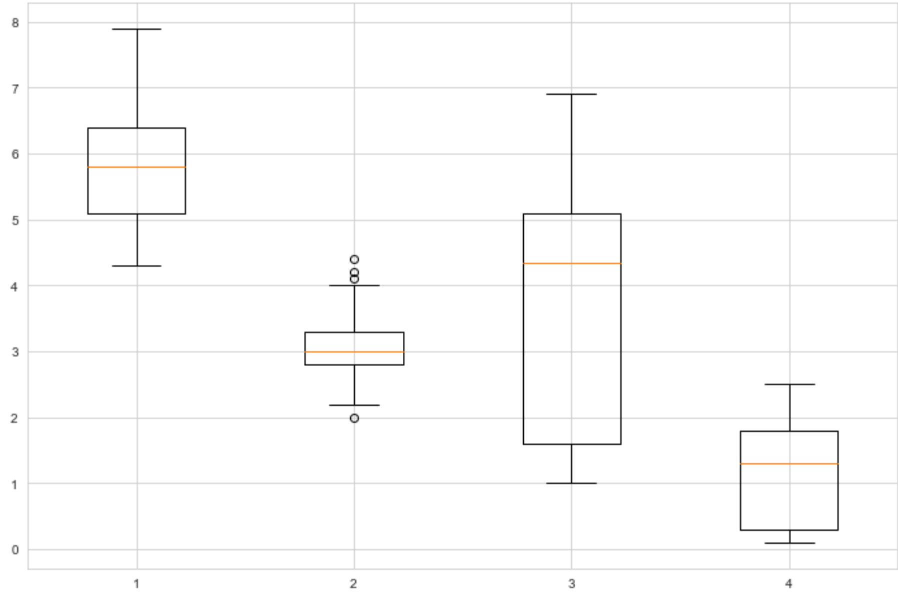
Q4. Create a boxplot for each feature in the dataset. All of the boxplots should be combined into a single plot. Compare distributions and identify outliers.

seaborn.set_style(style=None, rc=None)

Parameters style: dict, or one of {darkgrid, whitegrid, dark, white, ticks} A dictionary of parameters or the name of a preconfigured style.

rc: dict, optional Parameter mappings to override the values in the preset seaborn style dictionaries. This only updates parameters that are considered part of the style definition.

In [13]:
```
data_to_plot = [df["Sepal_Length"],df["Sepal_Width"],df["Petal_Length"],df["Petal_Width"]]

sns.set_style("whitegrid")
# Creating a figure instance
fig = plt.figure(1, figsize=(12,8))

# Creating an axes instance
ax = fig.add_subplot(111)

# Creating the boxplot
bp = ax.boxplot(data_to_plot);
```



In [ ]: