# Data Analytics Lab Manual

1.Write a Python program to perform linear search

2.Write a Python program to insert an element into a sorted list

3. Write a python program using object-oriented programming to demonstrate encapsulation, overloading and inheritance

4.Implement a python program to demonstrate

1) Importing Datasets 2) Cleaning the Data 3) Data frame manipulation using Numpy

5.Implement a python program to demonstrate the following using NumPy

- Array manipulation, Searching, Sorting and splitting.
- broadcasting and Plotting NumPy arrays

6. Implement a python program to demonstrate

Data visualization with various Types of Graphs using Numpy

7.Write a Python program that creates a mxn integer arrayand Prints its attributes using matplotlib

8.Write a Python program to demonstrate the generation of linear regression models.

9.Write a Python program to demonstrate the generation of logistic regression models using Python

10.Write a Python program to demonstrate Timeseries analysis with Pandas.

11.Write a Python program to demonstrate Data Visualization using Seaborn.

1.Write a Python program to perform linear search

```python

# 1. Write a Python program to perform linear search

def linearSearch(array, n, x):
# Going through array sequencially
    for i in range(0, n):
        if (array[i] == x):
            return i
    return -1
array = [2, 4, 0, 1, 9]
x = eval(input("enter the element to be searched: "))
# Python eval() function is used to parse an expression string as python expression
# and then execute it.
n = len(array)
result = linearSearch(array, n, x)
if(result == -1):
    print("Element not found")
else:
    print("Element found at index: ", result)
```

```
enter the element to be searched: 0
Element found at index:  2
```

2.Write a Python program to insert an element into a sorted list

2nd.py

C: > Users > Raghu > Documents > python programs > 2nd.py > ...

```
1  # 2. Write a Python program to insert an element into a sorted list
2  #Approach : Python comes with a bisect module whose purpose is to find a position
3  # in list where an element needs to be inserted to keep the list sorted.
4  # Thus we use this module to solve the givenproblem.
5  import bisect
6  def insert(list, n):
7      bisect.insort(list, n)
8      return list
9  list = [1, 2, 4]
10 n = eval(input("enter the value to be inserted "))
11 print(insert(list, n))
```

```
enter the value to be inserted 5
[1, 2, 4, 5]
```

3. Write a python program using object-oriented programming to demonstrate encapsulation, overloading and inheritance

```python
""" 3. Write a python program using object oriented programming
to demonstrate encapsulation, overloading and inheritance """

""" Python doesn't actually prevent someone from accessing internal names. However, do-ing so is
 considered impolite, and may result in fragile code. It should be noted,
 too,that the use of the leading underscore is also used for module names and module-level functions."""
""" You may also encounter the use of two leading underscores ( ) on names within class definitions. For Example: """
class Base:
    def __init__(self):
        self.a = 10
        self._b = 20

    def display(self):
        print(" the values are :")
        print(f"a={self.a} b={self._b}")
class Derived(Base): # Creating a derived class
    def __init__(self):
        Base.__init__(self) # Calling constructor of Base class
        self.d = 30
        def display(self):
            Base.display(self)
            print(f"d={self.d}")

    def __add__(self, ob):
        return self.a + ob.a+self.d + ob.d
#return self.a + ob.a+self.d + ob.d+self.b + ob.b
obj1 = Base()
obj2 = Derived()
obj3 = Derived()
obj2.display()
obj3.display()
print("\n Sum of two objects :",obj2 + obj3)
```

```
 the values are :
a=10 b=20
 the values are :
a=10 b=20

 Sum of two objects : 80
```

4.Implement a python program to demonstrate

1) Importing Datasets 2) Cleaning the Data 3) Data frame manipulation using Numpy

```
import pandas as pd
df=pd.read_csv("C:\\Users\\Apple\\Desktop\\toyota.csv",skiprows=1)
```

```
df
```

| | 0 | 13500 | 23 | 46986 | Diesel | 90 | 1 | 0.1 | 2000 | three | 1165 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 13750 | 23.0 | 72937 | Diesel | 90 | 1.0 | 0 | 2000 | 3 | 1165 |
| **1** | 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| **2** | 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| 3 | 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |
| **4** | 5 | 12950 | 32.0 | 61000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1430** | 1431 | 7500 | NaN | 20544 | Petrol | 86 | 1.0 | 0 | 1300 | 3 | 1025 |
| **1431** | 1432 | 10845 | 72.0 | ?? | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1432** | 1433 | 8500 | NaN | 17016 | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1433** | 1434 | 7250 | 70.0 | ?? | NaN | 86 | 1.0 | 0 | 1300 | 3 | 1015 |
| **1434** | 1435 | 6950 | 76.0 | 1 | Petrol | 110 | 0.0 | 0 | 1600 | 5 | 1114 |

1435 rows × 11 columns

```
import pandas as pd
df=pd.read_csv("C:\\Users\\Apple\\Desktop\\toyota.csv",skiprows=1,names=['a','b','c','d','e','f','g','h'])
```

```
df
```

| | a | b | c | d | e | f | g | h |
|---|---|---|---|---|---|---|---|---|
| 0 | 13500 | 23.0 | 46986 | Diesel | 90 | 1.0 | 0 | 2000 | three | 1165 |
| 1 | 13750 | 23.0 | 72937 | Diesel | 90 | 1.0 | 0 | 2000 | 3 | 1165 |
| 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1431 | 7500 | NaN | 20544 | Petrol | 86 | 1.0 | 0 | 1300 | 3 | 1025 |
| 1432 | 10845 | 72.0 | ?? | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| 1433 | 8500 | NaN | 17016 | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| 1434 | 7250 | 70.0 | ?? | NaN | 86 | 1.0 | 0 | 1300 | 3 | 1015 |
| 1435 | 6950 | 76.0 | 1 | Petrol | 110 | 0.0 | 0 | 1600 | 5 | 1114 |

1436 rows × 8 columns

```
import pandas as pd
df=pd.read_csv("C:\\Users\\Apple\\Desktop\\toyota.csv",nrows=5)
```

```
df
```

| | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | Weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 13500 | 23 | 46986 | Diesel | 90 | 1.0 | 0 | 2000 | three | 1165 |
| 1 | 1 | 13750 | 23 | 72937 | Diesel | 90 | 1.0 | 0 | 2000 | 3 | 1165 |
| 2 | 2 | 13950 | 24 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| 3 | 3 | 14950 | 26 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| 4 | 4 | 13750 | 30 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |

```
import pandas as pd
df=pd.read_excel("C:\\Users\\Apple\\Desktop\\Toyota1.xls")
```

```
df
```

| | Unnamed: 0 | Unnamed: 0.1 | Unnamed: 1 | Unnamed: 2 | Unnamed: 3 | Unnamed: 4 | Unnamed: 5 | Unnamed: 6 | Unnamed: 7 | Unnamed: 8 | Unnamed: 9 | Unnamed: 10 | Unnamed: 11 | Unna |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |
| **1** | 1 | NaN | NaN | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | \ |
| **2** | 2 | NaN | NaN | 0 | 13500 | 23 | 46986 | Diesel | 90 | 1 | 0 | 2000 | three | |
| **3** | 3 | NaN | NaN | 1 | 13750 | 23 | 72937 | Diesel | 90 | 1 | 0 | 2000 | 3 | |
| **4** | 4 | NaN | NaN | 2 | 13950 | 24 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **1433** | 1433 | NaN | NaN | 1431 | 7500 | NaN | 20544 | Petrol | 86 | 1 | 0 | 1300 | 3 | |
| **1434** | 1434 | NaN | NaN | 1432 | 10845 | 72 | ?? | Petrol | 86 | 0 | 0 | 1300 | 3 | |
| **1435** | 1435 | NaN | NaN | 1433 | 8500 | NaN | 17016 | Petrol | 86 | 0 | 0 | 1300 | 3 | |
| **1436** | 1436 | NaN | NaN | 1434 | 7250 | 70 | ?? | NaN | 86 | 1 | 0 | 1300 | 3 | |
| **1437** | 1437 | NaN | NaN | 1435 | 6950 | 76 | 1 | Petrol | 110 | 0 | 0 | 1600 | 5 | |

1438 rows × 14 columns

```
import pandas as pd
import numpy as np
df=pd.read_csv("C:\\Users\\Apple\\Desktop\\Toyota.csv")

new_df=df.replace(-77777,np.NaN)
```

```
df
```

| | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | Weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 13500 | 23.0 | 46986 | Diesel | 90 | 1.0 | 0 | 2000 | three | 1165 |
| **1** | 1 | 13750 | 23.0 | 72937 | Diesel | 90 | 1.0 | 0 | 2000 | 3 | 1165 |
| **2** | 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| **3** | 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| **4** | 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1431** | 1431 | 7500 | NaN | 20544 | Petrol | 86 | 1.0 | 0 | 1300 | 3 | 1025 |
| **1432** | 1432 | 10845 | 72.0 | ?? | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1433** | 1433 | 8500 | NaN | 17016 | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1434** | 1434 | 7250 | 70.0 | ?? | NaN | 86 | 1.0 | 0 | 1300 | 3 | 1015 |
| **1435** | 1435 | 6950 | 76.0 | 1 | Petrol | 110 | 0.0 | 0 | 1600 | 5 | 1114 |

1436 rows × 11 columns

```
weather_data=[
('1/1/2017',32,6,'rain'),
('1/2/2017',35,6,'sunny'),
('1/3/2017',28,2,'snow')
]
```

```
import pandas as pd
df=pd.DataFrame(weather_data,columns=["day","temperature","windspped","event"])
```

```
df
```

| | day | temperature | windspped | event |
|---|---|---|---|---|
| **0** | 1/1/2017 | 32 | 6 | rain |
| **1** | 1/2/2017 | 35 | 6 | sunny |
| **2** | 1/3/2017 | 28 | 2 | snow |

```
weather_data=[
{'day':'1/1/2017','temp':32,'windspped':6,'event':'Rain'},
{'day':'1/2/2017','temp':35,'windspped':7,'event':'sunny'},
{'day':'1/1/2017','temp':28,'windspped':2,'event':'snow'},
]
```

```
import pandas as pd
df=pd.DataFrame(weather_data)
```

```
df
```

| | day | temp | windspped | event |
|---|---|---|---|---|
| 0 | 1/1/2017 | 32 | 6 | Rain |
| 1 | 1/2/2017 | 35 | 7 | sunny |
| 2 | 1/1/2017 | 28 | 2 | snow |

```
import pandas as pd
import numpy as np
df=pd.read_csv("C:\\Users\\Apple\\Desktop\\Toyota.csv")

new_df=df.replace(-99999,np.NaN)
```

```
df
```

| | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | Weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 13500 | 23.0 | 46986 | Diesel | -99999 | 1.0 | 0 | 2000 | three | 1165 |
| 1 | 1 | 13750 | 23.0 | 72937 | Diesel | -99999 | 1.0 | 0 | 2000 | 3 | 1165 |
| 2 | 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| 3 | 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| 4 | 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |

```
import pandas as pd
import numpy as np
df=pd.read_csv("C:\\Users\\Apple\\Desktop\\Toyota.csv")

new_df=df.replace(-99999,np.NaN)
```

```
df
```

| | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | Weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 13500 | 23.0 | 46986 | Diesel | -99999 | 1.0 | 0 | 2000 | three | 1165 |
| **1** | 1 | 13750 | 23.0 | 72937 | Diesel | -99999 | 1.0 | 0 | 2000 | 3 | 1165 |
| **2** | 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| **3** | 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| **4** | 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1431** | 1431 | 7500 | NaN | 20544 | Petrol | 86 | 1.0 | 0 | 1300 | 3 | 1025 |
| **1432** | 1432 | 10845 | 72.0 | ?? | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1433** | 1433 | 8500 | NaN | 17016 | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1434** | 1434 | 7250 | 70.0 | ?? | NaN | 86 | 1.0 | 0 | 1300 | 3 | 1015 |
| **1435** | 1435 | 6950 | 76.0 | 1 | Petrol | 110 | 0.0 | 0 | 1600 | 5 | 1114 |

1436 rows × 11 columns

```
new_df=df.replace(-99,np.NaN)
```

```
new_df
```

| | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | Weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 13500 | 23.0 | 46986 | Diesel | -99999 | 1.0 | 0 | 2000 | three | 1165 |
| **1** | 1 | 13750 | 23.0 | 72937 | Diesel | -99999 | 1.0 | 0 | 2000 | 3 | 1165 |
| **2** | 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| **3** | 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| **4** | 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1431** | 1431 | 7500 | NaN | 20544 | Petrol | 86 | 1.0 | 0 | 1300 | 3 | 1025 |
| **1432** | 1432 | 10845 | 72.0 | ?? | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1433** | 1433 | 8500 | NaN | 17016 | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1434** | 1434 | 7250 | 70.0 | ?? | NaN | 86 | 1.0 | 0 | 1300 | 3 | 1015 |
| **1435** | 1435 | 6950 | 76.0 | 1 | Petrol | 110 | 0.0 | 0 | 1600 | 5 | 1114 |

1436 rows × 11 columns

```
new_df=df.replace([-99999,-88888],np.NaN)
```

```
new_df
```

| | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | Weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 13500 | 23.0 | 46986 | Diesel | -99999 | 1.0 | 0 | 2000 | three | 1165 |
| **1** | 1 | 13750 | 23.0 | 72937 | Diesel | -99999 | 1.0 | 0 | 2000 | 3 | 1165 |
| **2** | 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| **3** | 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| **4** | 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1431** | 1431 | 7500 | NaN | 20544 | Petrol | 86 | 1.0 | 0 | 1300 | 3 | 1025 |
| **1432** | 1432 | 10845 | 72.0 | ?? | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1433** | 1433 | 8500 | NaN | 17016 | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1434** | 1434 | 7250 | 70.0 | ?? | NaN | 86 | 1.0 | 0 | 1300 | 3 | 1015 |
| **1435** | 1435 | 6950 | 76.0 | 1 | Petrol | 110 | 0.0 | 0 | 1600 | 5 | 1114 |

1436 rows × 11 columns

```
new_df=df.replace(
{
'temperature':-99999,
'windspeed':-99999,
'event':'0'
},np.NaN)
```

```
new_df
```

| | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | Weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 13500 | 23.0 | 46986 | Diesel | -99999 | 1.0 | 0 | 2000 | three | 1165 |
| **1** | 1 | 13750 | 23.0 | 72937 | Diesel | -99999 | 1.0 | 0 | 2000 | 3 | 1165 |
| **2** | 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| **3** | 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| **4** | 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1431** | 1431 | 7500 | NaN | 20544 | Petrol | 86 | 1.0 | 0 | 1300 | 3 | 1025 |
| **1432** | 1432 | 10845 | 72.0 | ?? | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1433** | 1433 | 8500 | NaN | 17016 | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1434** | 1434 | 7250 | 70.0 | ?? | NaN | 86 | 1.0 | 0 | 1300 | 3 | 1015 |
| **1435** | 1435 | 6950 | 76.0 | 1 | Petrol | 110 | 0.0 | 0 | 1600 | 5 | 1114 |

1436 rows × 11 columns

```
import pandas as pd
df=pd.DataFrame(weather_data)
new_df=df.replace(
{
'temperature':-99999,
'windspeed':-99999,
'event':'0'
},np.NaN)
```

```
new_df
```

| | day | temp | windspped | event |
|---|---|---|---|---|
| **0** | 1/1/2017 | 32 | 6 | Rain |
| **1** | 1/2/2017 | 35 | 7 | sunny |
| **2** | 1/1/2017 | 28 | 2 | snow |

```
new_df=df.replace(
{
-99999:np.NaN,
'No Event':'Sunny'
})
```

```
new_df
```

| | day | temp | windspped | event |
|---|---|---|---|---|
| **0** | 1/1/2017 | 32 | 6 | Rain |
| **1** | 1/2/2017 | 35 | 7 | sunny |
| **2** | 1/1/2017 | 28 | 2 | snow |

```
new_df=df.replace({'[A-Za-z]':' '},regex=True)
```

```
new_df
```

| | day | temp | windspped | event |
|---|---|---|---|---|
| **0** | 1/1/2017 | 32 | 6 | |
| **1** | 1/2/2017 | 35 | 7 | |
| **2** | 1/1/2017 | 28 | 2 | |

```
new_df=df.replace({
    'temperature':'[A-Za-z]',
    'windspeed':'[A-Za-z]'},'',regex=True)
```

```
new_df
```

| | day | temp | windspped | event |
|---|---|---|---|---|
| **0** | 1/1/2017 | 32 | 6 | Rain |
| **1** | 1/2/2017 | 35 | 7 | sunny |
| **2** | 1/1/2017 | 28 | 2 | snow |

```
df=pd.DataFrame({
'Score':['exceptional','average','good','poor','average','exceptional'],
'student':['rob','maya','parthiv','tom','julian','erica']
})
df.replace(['poor','average','good','exceptional'],[1,2,3,4])
```

| | Score | student |
|---|---|---|
| **0** | 4 | rob |
| **1** | 2 | maya |
| **2** | 3 | parthiv |
| **3** | 1 | tom |
| **4** | 2 | julian |
| **5** | 4 | erica |

```
df.replace(['poor','average','good','exceptional'],[1,2,3,4])
```

| | Score | student |
|---|---|---|
| **0** | 4 | rob |
| **1** | 2 | maya |
| **2** | 3 | parthiv |
| **3** | 1 | tom |
| **4** | 2 | julian |
| **5** | 4 | erica |

```
import pandas as pd
df=pd.read_csv("C:\\Users\\Apple\\Desktop\\Toyota.csv")
```

```
df
```

| | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | Weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 13500 | 23.0 | 46986 | Diesel | -99999 | 1.0 | 0 | 2000 | three | 1165 |
| **1** | 1 | 13750 | 23.0 | 72937 | Diesel | -99999 | 1.0 | 0 | 2000 | 3 | 1165 |
| **2** | 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| **3** | 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| **4** | 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

```
df.head()
```

| | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | Weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 13500 | 23.0 | 46986 | Diesel | -99999 | 1.0 | 0 | 2000 | three | 1165 |
| **1** | 1 | 13750 | 23.0 | 72937 | Diesel | -99999 | 1.0 | 0 | 2000 | 3 | 1165 |
| **2** | 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| 3 | 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| **4** | 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |

```
df.tail()
```

| | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | Weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1431** | 1431 | 7500 | NaN | 20544 | Petrol | 86 | 1.0 | 0 | 1300 | 3 | 1025 |
| **1432** | 1432 | 10845 | 72.0 | ?? | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1433** | 1433 | 8500 | NaN | 17016 | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |
| **1434** | 1434 | 7250 | 70.0 | ?? | NaN | 86 | 1.0 | 0 | 1300 | 3 | 1015 |
| **1435** | 1435 | 6950 | 76.0 | 1 | Petrol | 110 | 0.0 | 0 | 1600 | 5 | 1114 |

```
import pandas as pd
df=pd.read_csv("C:\\Users\\Apple\\Desktop\\Toyota.csv")
```

```
df.columns
```

```
Index(['Unnamed: 0', 'Price', 'Age', 'KM', 'FuelType', 'HP', 'MetColor',
       'Automatic', 'CC', 'Doors', 'Weight'],
      dtype='object')
```

```
df.Price
```

```
0       13500
1       13750
2       13950
3       14950
4       13750
        ...
1431     7500
1432    10845
1433     8500
1434     7250
1435     6950
Name: Price, Length: 1436, dtype: int64
```

```
df.Doors
```

```
0       three
1           3
2           3
3           3
4           3
        ...
1431        3
1432        3
1433        3
1434        3
1435        5
Name: Doors, Length: 1436, dtype: object
```

```
df['KM']
```

```
0       46986
1       72937
2       41711
3       48000
4       38500
        ...
1431    20544
1432       ??
1433    17016
1434       ??
1435        1
Name: KM, Length: 1436, dtype: object
```

```
df['Price'].max()
```

```
32500
```

```
df.describe()
```

| | Unnamed: 0 | Price | Age | MetColor | Automatic | CC | Weight |
|---|---|---|---|---|---|---|---|
| **count** | 1436.000000 | 1436.000000 | 1336.000000 | 1286.000000 | 1436.000000 | 1436.000000 | 1436.00000 |
| **mean** | 717.500000 | 10730.824513 | 55.672156 | 0.674961 | 0.055710 | 1566.827994 | 1072.45961 |
| **std** | 414.681806 | 3626.964585 | 18.589804 | 0.468572 | 0.229441 | 187.182436 | 52.64112 |
| **min** | 0.000000 | 4350.000000 | 1.000000 | 0.000000 | 0.000000 | 1300.000000 | 1000.00000 |
| **25%** | 358.750000 | 8450.000000 | 43.000000 | 0.000000 | 0.000000 | 1400.000000 | 1040.00000 |
| **50%** | 717.500000 | 9900.000000 | 60.000000 | 1.000000 | 0.000000 | 1600.000000 | 1070.00000 |
| **75%** | 1076.250000 | 11950.000000 | 70.000000 | 1.000000 | 0.000000 | 1600.000000 | 1085.00000 |
| **max** | 1435.000000 | 32500.000000 | 80.000000 | 1.000000 | 1.000000 | 2000.000000 | 1615.00000 |

```
df[df.Price>=1436]
```

| | Unnamed: 0 | Price | Age | KM | FuelType | HP | MetColor | Automatic | CC | Doors | Weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 13500 | 23.0 | 46986 | Diesel | -99999 | 1.0 | 0 | 2000 | three | 1165 |
| **1** | 1 | 13750 | 23.0 | 72937 | Diesel | -99999 | 1.0 | 0 | 2000 | 3 | 1165 |
| **2** | 2 | 13950 | 24.0 | 41711 | Diesel | 90 | NaN | 0 | 2000 | 3 | 1165 |
| **3** | 3 | 14950 | 26.0 | 48000 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1165 |
| **4** | 4 | 13750 | 30.0 | 38500 | Diesel | 90 | 0.0 | 0 | 2000 | 3 | 1170 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1431** | 1431 | 7500 | NaN | 20544 | Petrol | 86 | 1.0 | 0 | 1300 | 3 | 1025 |
| **1432** | 1432 | 10845 | 72.0 | ?? | Petrol | 86 | 0.0 | 0 | 1300 | 3 | 1015 |

5.Implement a python program to demonstrate the following using NumPy
i) Array manipulation, Searching, Sorting and splitting.
ii) broadcasting and Plotting NumPy arrays

In [11]:
```
import numpy as np
arr1=([[1,2,3],[4,5,6]])

arr2=([[7,8,9],[9,10,11]])
print("concatenating of two arrays \n ",np.concatenate([arr1,arr2],axis=1))
print("Vertical stacking \n",np.vstack((arr1,arr2)))
print("Horizontal stacking \n",np.hstack((arr1,arr2)))
```

```
concatenating of two arrays
  [[ 1  2  3  7  8  9]
 [ 4  5  6  9 10 11]]
Vertical stacking
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [ 9 10 11]]
Horizontal stacking
 [[ 1  2  3  7  8  9]
 [ 4  5  6  9 10 11]]
```

In [12]:
```
# searching
arr =np.array([1,2,3,4,5,4,4])
x=np.where(arr==4)
print(x)

arr =np.array([6,7,8,9])
x=np.searchsorted(arr,5)
print(x)

arr =np.array([1,3,5,7])
x=np.searchsorted(arr,[2,4,6])
print(x)
```

```
(array([3, 5, 6], dtype=int64),)
0
[1 2 3]
```

In [13]:
```
a =np.array([[1,4],[3,1]])
print("sorted array : ",np.sort(a))
print("\n Flattened sorted array is ",np.sort(a,axis=0))
```

```
a =np.array([[1,4],[3,1]])
print("sorted array : ",np.sort(a))
print("\n Flattened sorted array is ",np.sort(a,axis=0))

x =np.array([3,1,2])
print("\n indices that would sort an array ",np.argsort(x))
print("\n sorted complex number", np.sort_complex([5,3,6,2,1]))
```

```
sorted array :  [[1 4]
 [1 3]]

 Flattened sorted array is  [[1 1]
 [3 4]]

 indices that would sort an array  [1 2 0]

 sorted complex number [1.+0.j 2.+0.j 3.+0.j 5.+0.j 6.+0.j]
```

```
w =  [4 5]

 outer product of v & w is

[[ 4  5]
 [ 8 10]
 [12 15]]
x =  [[1 2 3]
 [4 5 6]]
v =  [1 2 3]

 x + v =  [[2 4 6]
 [5 7 9]]

 transposing this final result
[[ 5  6  7]
 [ 9 10 11]]

 x+np.reshape(w, (2,1))
[[ 5  6  7]
 [ 9 10 11]]
[[ 2  4  6]
 [ 8 10 12]]
```

```
import numpy as np
x=np.arange(9,0)
print(np.split(x,3))
print(np.split(x,[3,5,6,10]))

x=np.arange(9)
np.array_split(x,4)

a=np.array([[1,3,5,7,9,11],[2,4,6,8,10,12]])

print("splitting along horizontal axis into 2 parts\n", np.hsplit(a,2))
print("\n splitting along Vertical axis into 2 parts\n", np.vsplit(a,2))
```

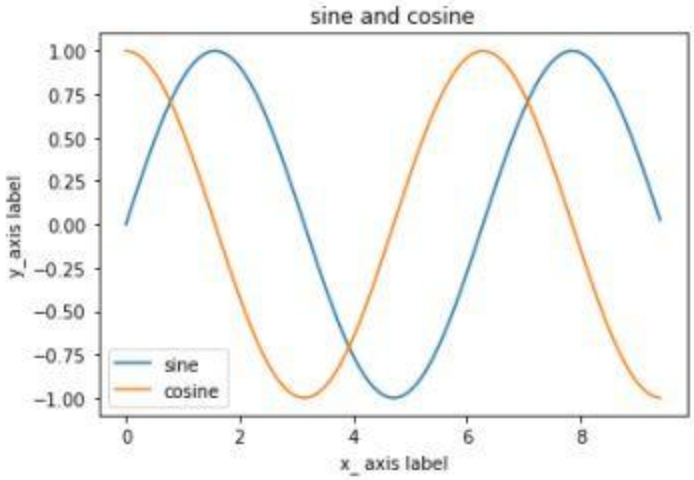
```
[array([], dtype=int32), array([], dtype=int32), array([], dtype=int32)]
[array([], dtype=int32), array([], dtype=int32), array([], dtype=int32), array([], dtype
=int32), array([], dtype=int32)]
splitting along horizontal axis into 2 parts
 [array([[1, 3, 5],
       [2, 4, 6]]), array([[ 7,  9, 11],
       [ 8, 10, 12]])]

 splitting along Vertical axis into 2 parts
 [array([[ 1,  3,  5,  7,  9, 11]]), array([[ 2,  4,  6,  8, 10, 12]])]
```

```
import numpy as np
v=np.array([1,2,3])
w=np.array([4,5])
print("v = ", v)
print("w = ", w)

print("\n outer product of v & w is \n")
print(np.reshape(v,(3,1)) * w)

x=np.array([[1,2,3],[4,5,6]])

print("x = ", x)
print("v = ", v)
```

```
print("\n x + v = ", x+v)

print("\n transposing this final result")
print((x.T+w).T)

print("\n x+np.reshape(w, (2,1))")
print(x+np.reshape(w,(2,1)))

print(x*2)
```

```
v =  [1 2 3]
w =  [4 5]

 outer product of v & w is

[[ 4  5]
 [ 8 10]
 [12 15]]
x =  [[1 2 3]
 [4 5 6]]
v =  [1 2 3]

 x + v =  [[2 4 6]
 [5 7 9]]

 transposing this final result
[[ 5  6  7]
 [ 9 10 11]]

 x+np.reshape(w, (2,1))
[[ 5  6  7]
 [ 9 10 11]]
[[ 2  4  6]
 [ 8 10 12]]
```

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

x=np.arange(0,3 * np.pi, 0.1)
y_sin= np.sin(x)
y_cos= np.cos(x)

plt.plot(x,y_sin)
plt.plot(x,y_cos)
plt.xlabel('x_ axis label')
plt.ylabel('y_axis label')
plt.title('sine and cosine')
plt.legend(['sine' , 'cosine'])
plt.show()
```

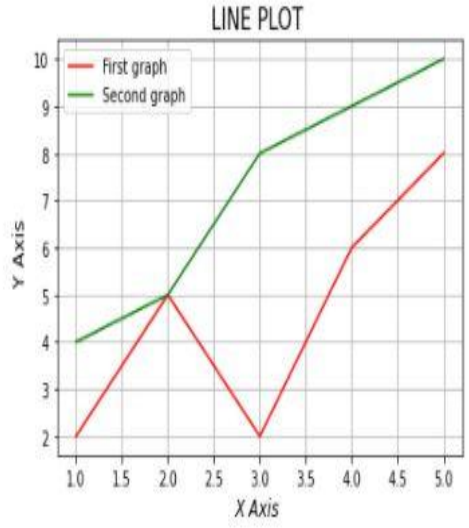

sine and cosine
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y_axis label
sine
cosine
0
2
4
6
8
x_ axis label

6. Implement a python program to demonstrate

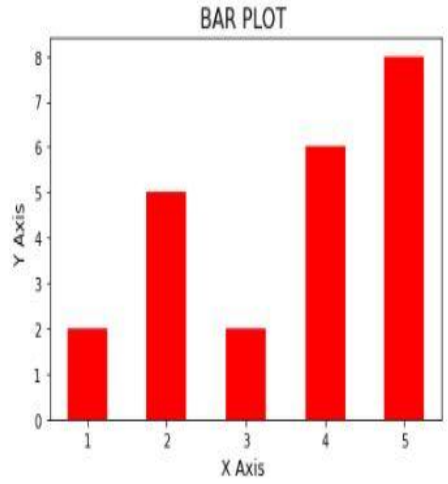
## 6. Implement a python program to demonstrate

## Data visualization with various Types of Graphs using Numpy ¶

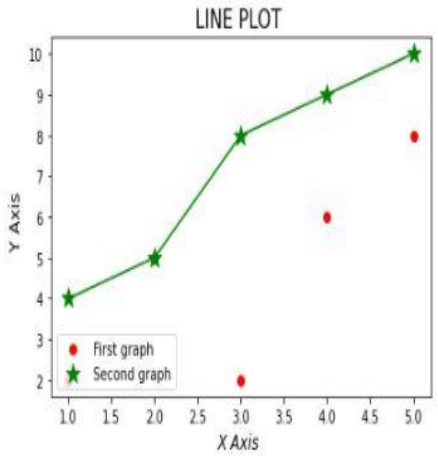
In [4]:
```
####Data Visualization
####Library matplotlib
import matplotlib.pyplot as plt

###Line plot
x1=[1,2,3,4,5]
y1=[2,5,2,6,8]
x2=[1,2,3,4,5]
y2=[4,5,8,9,10]
plt.xlabel("X Axis ",fontsize=12,fontstyle='italic')
plt.ylabel("Y Axis ",fontsize=12)
plt.title("LINE PLOT ",fontsize=15,fontname='DejaVu Sans')
plt.plot(x1,y1,color='red',label='First graph')
plt.plot(x2,y2,color='green',label='Second graph')
plt.legend(loc=2)
plt.grid()
#plt.axis('off')
plt.show()
```



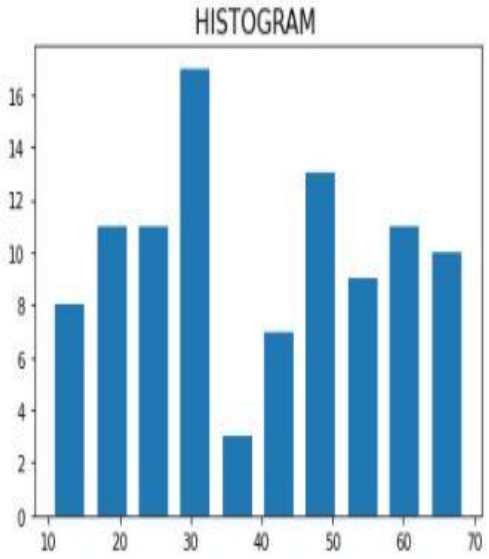
In [5]:
```
####Bar plot
#x=[1,2,3,4,5]
x=['A','B','C','D','E']
y=[20,40,20,60,80]
plt.xlabel("X Axis ",fontsize=12)
plt.ylabel("Y Axis ",fontsize=12)
plt.title("BAR PLOT ", fontsize=15)
plt.bar(x1,y1,color='red',width=0.5)
plt.show()
```



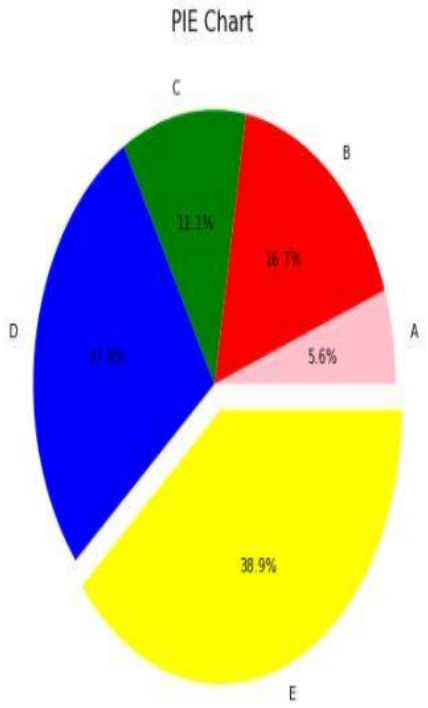
In [25]:
```
####Scatter plot
x1=[1,2,3,4,5]
y1=[2,5,2,6,8]
x2=[1,2,3,4,5]
y2=[4,5,8,9,10]
plt.xlabel("X Axis ",fontsize=12,fontstyle='italic')
plt.ylabel("Y Axis ",fontsize=12)
plt.title("LINE PLOT ",fontsize=15,fontname='DejaVu Sans')
plt.scatter(x1,y1,color='red',label='First graph')
plt.scatter(x2,y2,color='green',s=150,marker="*",label='Second graph')
plt.plot(x2,y2,color='green')
plt.legend(loc=3)
#plt.axis('off')
plt.show()
```



In [27]:
```
####Histogram
import numpy as np
sample=np.random.randint(10,70,100)
plt.title("HISTOGRAM ",fontsize=15,fontname='DejaVu Sans')
plt.hist(sample,rwidth=0.7)
plt.show()
```



In [28]:
```
####Pie Chart
plt.figure(figsize=(7,7))
slices=[10,30,20,50,70]
act=["A","B","C","D","E"]
cols=["pink","red","green","blue","yellow"]
plt.title("PIE Chart ",fontsize=15,fontname='DejaVu Sans')
plt.pie(slices,labels=act,colors=cols,autopct="%1.1f%%",explode=(0,0,0,0,0.1))
plt.show()
```



7.Write a Python program that creates a mxn integer arrayand Prints its attributes using matplotlib

In [2]:
```
##Create M X N integer array and prints its attributes using numpy
import numpy as np
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
print("Printing array")
print()
print(a)
print("Printing numpy array attributes")
print("1)Array dimension: ", a.ndim)
print("2) Array shape:", a.shape)
print("3) Array size: ", a.size)
print("4) Array data type :", a.dtype)
print("5) The length of each array item in bytes is : ",a.itemsize)
```

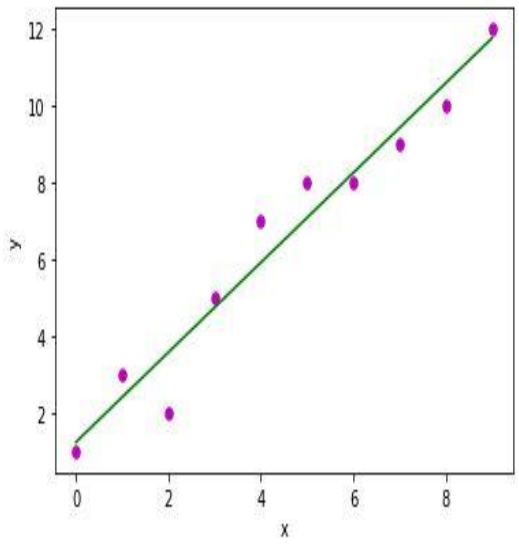
```
Printing array

[[1 2 3]
 [4 5 6]
 [7 8 9]]
Printing numpy array attributes
1)Array dimension:  2
2) Array shape: (3, 3)
3) Array size:  9
4) Array data type : int32
5) The length of each array item in bytes is :  4
```

## 8.Write a Python program to demonstrate the generation of linear regression models.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
               marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")
```

```
    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```



## 9.Write a Python program to demonstrate the generation of logistic regression models using Python.

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

dataset = pd.read_csv('C://Users//Apple//Desktop//User_Data.csv')

x = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.25, random_state = 0)

from sklearn.preprocessing import StandardScaler

sc_x = StandardScaler()
xtrain = sc_x.fit_transform(xtrain)
xtest = sc_x.transform(xtest)

print (xtrain[0:10, :])

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)
y_pred = classifier.predict(xtest)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ytest, y_pred)

print ("Confusion Matrix : \n", cm)
from sklearn.metrics import accuracy_score
```

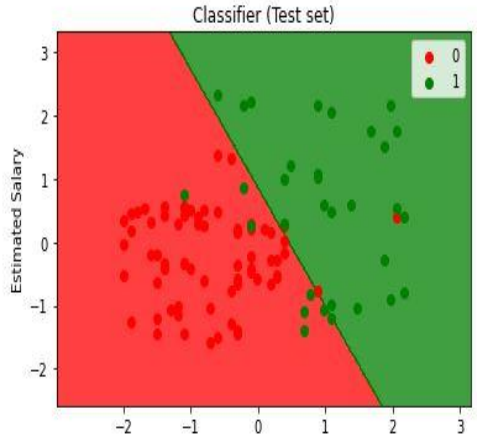
```
print ("Accuracy : ", accuracy_score(ytest, y_pred))

from matplotlib.colors import ListedColormap
X_set, y_set = xtest, ytest
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 = 0.75, 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('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with a single row if yo
u intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with a single row if yo
u intend to specify the same RGB or RGBA value for all points.
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]]
Confusion Matrix :
 [[65  3]
 [ 8 24]]
Accuracy :  0.89
```



## 10.Write a Python program to demonstrate Timeseries analysis with Pandas.

In [1]:
```
import pandas as pd
df = pd.read_csv("aapl.csv")
df.head()
```

Out[1]:

| | Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|---|
| 0 | 7-Jul-17 | 142.90 | 144.75 | 142.90 | 144.18 | 19201712 |
| 1 | 6-Jul-17 | 143.02 | 143.50 | 142.41 | 142.73 | 24128782 |
| 2 | 5-Jul-17 | 143.69 | 144.79 | 142.72 | 144.09 | 21569557 |
| 3 | 3-Jul-17 | 144.88 | 145.30 | 143.10 | 143.50 | 14277848 |
| 4 | 30-Jun-17 | 144.45 | 144.96 | 143.78 | 144.02 | 23024107 |

In [2]:
```
df = pd.read_csv("aapl.csv",parse_dates=["Date"], index_col="Date")
df.tail()
```

Out[2]:

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| 2016-07-15 | 98.92 | 99.30 | 98.50 | 98.78 | 30136990 |
| 2016-07-14 | 97.39 | 98.99 | 97.32 | 98.79 | 38918997 |
| 2016-07-13 | 97.41 | 97.67 | 96.84 | 96.87 | 25892171 |
| 2016-07-12 | 97.17 | 97.70 | 97.12 | 97.42 | 24167463 |
| 2016-07-11 | 96.75 | 97.65 | 96.73 | 96.98 | 23794945 |

In [3]:
```
df.index
```

Out[3]:
```
DatetimeIndex(['2017-07-07', '2017-07-06', '2017-07-05', '2017-07-03',
               '2017-06-30', '2017-06-29', '2017-06-28', '2017-06-27',
               '2017-06-26', '2017-06-23',
               ...
               '2016-07-22', '2016-07-21', '2016-07-20', '2016-07-19',
               '2016-07-18', '2016-07-15', '2016-07-14', '2016-07-13',
               '2016-07-12', '2016-07-11'],
              dtype='datetime64[ns]', name='Date', length=251, freq=None)
```

# What is DatetimeIndex? Benefits of it

(1) Partial Date Index: Select Specific Months Data

In [4]:
```
df.loc['2017-06-30']
```

Out[4]:

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| 2017-06-30 | 144.45 | 144.96 | 143.78 | 144.02 | 23024107 |

In [5]:
```
df.loc["2017-01"]
```

In [5]:
```
df.loc["2017-01"]
```

Out[5]:

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| **2017-01-31** | 121.15 | 121.39 | 120.62 | 121.35 | 49200993 |
| **2017-01-30** | 120.93 | 121.63 | 120.66 | 121.63 | 30377503 |
| **2017-01-27** | 122.14 | 122.35 | 121.60 | 121.95 | 20562944 |
| **2017-01-26** | 121.67 | 122.44 | 121.60 | 121.94 | 26337576 |
| **2017-01-25** | 120.42 | 122.10 | 120.28 | 121.88 | 32586673 |
| **2017-01-24** | 119.55 | 120.10 | 119.50 | 119.97 | 23211038 |
| **2017-01-23** | 120.00 | 120.81 | 119.77 | 120.08 | 22050218 |
| **2017-01-20** | 120.45 | 120.45 | 119.73 | 120.00 | 32597892 |
| **2017-01-19** | 119.40 | 120.09 | 119.37 | 119.78 | 25597291 |
| **2017-01-18** | 120.00 | 120.50 | 119.71 | 119.99 | 23712961 |
| **2017-01-17** | 118.34 | 120.24 | 118.22 | 120.00 | 34439843 |
| **2017-01-13** | 119.11 | 119.62 | 118.81 | 119.04 | 26111948 |
| **2017-01-12** | 118.90 | 119.30 | 118.21 | 119.25 | 27086220 |
| **2017-01-11** | 118.74 | 119.93 | 118.60 | 119.75 | 27588593 |
| **2017-01-10** | 118.77 | 119.38 | 118.30 | 119.11 | 24462051 |
| **2017-01-09** | 117.95 | 119.43 | 117.94 | 118.99 | 33561948 |
| **2017-01-06** | 116.78 | 118.16 | 116.47 | 117.91 | 31751900 |
| **2017-01-05** | 115.92 | 116.86 | 115.81 | 116.61 | 22193587 |
| **2017-01-04** | 115.85 | 116.51 | 115.75 | 116.02 | 21118116 |
| **2017-01-03** | 115.80 | 116.33 | 114.76 | 116.15 | 28781865 |

In [6]:
```
df.loc['2017-06'].head()
```

Out[6]:

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| 2017-06-30 | 144.45 | 144.96 | 143.78 | 144.02 | 23024107 |
| 2017-06-29 | 144.71 | 145.13 | 142.28 | 143.68 | 31499368 |
| 2017-06-28 | 144.49 | 146.11 | 143.16 | 145.83 | 22082432 |
| 2017-06-27 | 145.01 | 146.16 | 143.62 | 143.73 | 24761891 |
| 2017-06-26 | 147.17 | 148.28 | 145.38 | 145.82 | 25692361 |

## average stock price of apple in june month

In [20]:
```
df['2017-06'].Close.mean()
```

```
<ipython-input-20-8e25964a0c31>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  df['2017-06'].Close.mean()
```

Out[20]: 147.8313636363636

In [11]:
```
df['2017'].head(2)
```

```
<ipython-input-11-801a1dbedba0>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  df['2017'].head(2)
```

In [6]:
```
df.loc['2017-06'].head()
```

Out[6]:

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| 2017-06-30 | 144.45 | 144.96 | 143.78 | 144.02 | 23024107 |
| 2017-06-29 | 144.71 | 145.13 | 142.28 | 143.68 | 31499368 |
| 2017-06-28 | 144.49 | 146.11 | 143.16 | 145.83 | 22082432 |
| 2017-06-27 | 145.01 | 146.16 | 143.62 | 143.73 | 24761891 |
| 2017-06-26 | 147.17 | 148.28 | 145.38 | 145.82 | 25692361 |

## average stock price of apple in june month

In [20]:
```
df['2017-06'].Close.mean()
```

```
<ipython-input-20-8e25964a0c31>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  df['2017-06'].Close.mean()
```

Out[20]:
```
147.8313636363636
```

In [11]:
```
df['2017'].head(2)
```

```
<ipython-input-11-801a1dbedba0>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  df['2017'].head(2)
```

In [13]:
```
df['2017-01']
```

```
<ipython-input-13-583ce925afa7>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice
the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  df['2017-01']
```

Out[13]:

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| 2017-01-31 | 121.15 | 121.39 | 120.62 | 121.35 | 49200993 |
| 2017-01-30 | 120.93 | 121.63 | 120.66 | 121.63 | 30377503 |
| 2017-01-27 | 122.14 | 122.35 | 121.60 | 121.95 | 20562944 |
| 2017-01-26 | 121.67 | 122.44 | 121.60 | 121.94 | 26337576 |
| 2017-01-25 | 120.42 | 122.10 | 120.28 | 121.88 | 32586673 |
| 2017-01-24 | 119.55 | 120.10 | 119.50 | 119.97 | 23211038 |
| 2017-01-23 | 120.00 | 120.81 | 119.77 | 120.08 | 22050218 |
| 2017-01-20 | 120.45 | 120.45 | 119.73 | 120.00 | 32597892 |
| 2017-01-19 | 119.40 | 120.09 | 119.37 | 119.78 | 25597291 |
| 2017-01-18 | 120.00 | 120.50 | 119.71 | 119.99 | 23712961 |
| 2017-01-17 | 118.34 | 120.24 | 118.22 | 120.00 | 34439843 |
| 2017-01-13 | 119.11 | 119.62 | 118.81 | 119.04 | 26111948 |
| 2017-01-12 | 118.90 | 119.30 | 118.21 | 119.25 | 27086220 |
| 2017-01-11 | 118.74 | 119.93 | 118.60 | 119.75 | 27588593 |
| 2017-01-10 | 118.77 | 119.38 | 118.30 | 119.11 | 24462051 |
| 2017-01-09 | 117.95 | 119.43 | 117.94 | 118.99 | 33561948 |
| 2017-01-06 | 116.78 | 118.16 | 116.47 | 117.91 | 31751900 |
| 2017-01-05 | 115.92 | 116.86 | 115.81 | 116.61 | 22193587 |
| 2017-01-04 | 115.85 | 116.51 | 115.75 | 116.02 | 21118116 |
| 2017-01-03 | 115.80 | 116.33 | 114.76 | 116.15 | 28781865 |

In [15]:
```
df['2016-07']
```

```
<ipython-input-15-9344db60f0ef>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  df['2016-07']
```

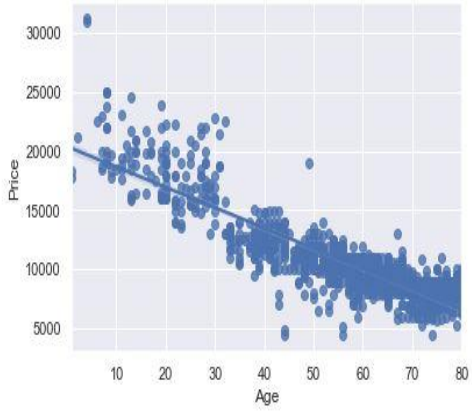
Out[15]:

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| 2016-07-29 | 104.19 | 104.55 | 103.68 | 104.21 | 27733688 |
| 2016-07-28 | 102.83 | 104.45 | 102.82 | 104.34 | 39869839 |
| 2016-07-27 | 104.26 | 104.35 | 102.75 | 102.95 | 92344820 |
| 2016-07-26 | 96.82 | 97.97 | 96.42 | 96.67 | 56239822 |
| 2016-07-25 | 98.25 | 98.84 | 96.92 | 97.34 | 40382921 |
| 2016-07-22 | 99.26 | 99.30 | 98.31 | 98.66 | 28313669 |
| 2016-07-21 | 99.83 | 101.00 | 99.13 | 99.43 | 32702028 |
| 2016-07-20 | 100.00 | 100.46 | 99.74 | 99.96 | 26275968 |
| 2016-07-19 | 99.56 | 100.00 | 99.34 | 99.87 | 23779924 |
| 2016-07-18 | 98.70 | 100.13 | 98.60 | 99.83 | 36493867 |
| 2016-07-15 | 98.92 | 99.30 | 98.50 | 98.78 | 30136990 |
| 2016-07-14 | 97.39 | 98.99 | 97.32 | 98.79 | 38918997 |
| 2016-07-13 | 97.41 | 97.67 | 96.84 | 96.87 | 25892171 |
| 2016-07-12 | 97.17 | 97.70 | 97.12 | 97.42 | 24167463 |
| 2016-07-11 | 96.75 | 97.65 | 96.73 | 96.98 | 23794945 |

## 11. Write a Python program to Demonstrate Data Visualization using Seaborn.

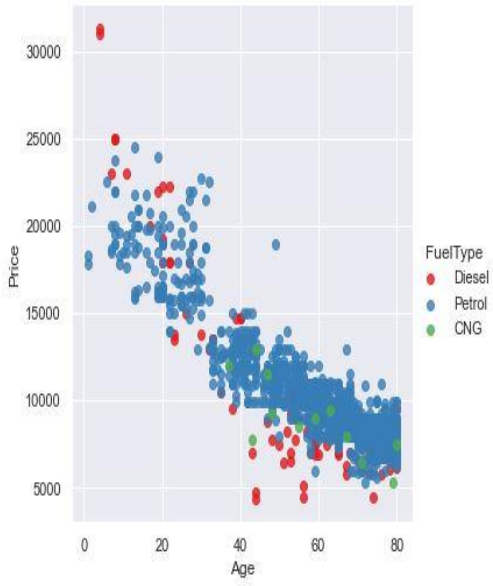
In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

cars_data=pd.read_csv('C://Users//mca//Desktop//Toyota.csv', index_col=0, na_values=["??,????"])
cars_data.dropna(axis=0,inplace=True)
sns.set(style="darkgrid")
sns.regplot(x=cars_data['Age'], y=cars_data['Price'])
```

Out[5]: <AxesSubplot:xlabel='Age', ylabel='Price'>



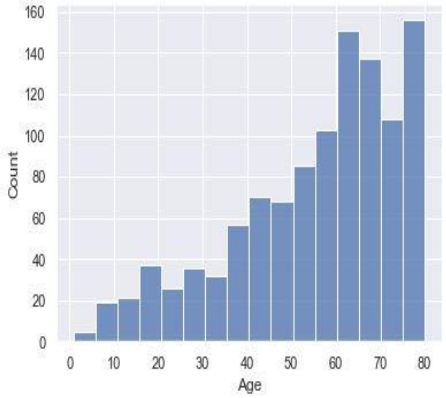
In [7]:
```
#scatter plot of Price VS age by fuel type
sns.lmplot(x='Age',y='Price',data=cars_data, fit_reg=False, hue="FuelType",legend=True, palette='Set1')
```

Out[7]: <seaborn.axisgrid.FacetGrid at 0xac35c488b0>



In [8]:
```
#distribution of the variable 'Age'
sns.histplot(cars_data['Age'])
```

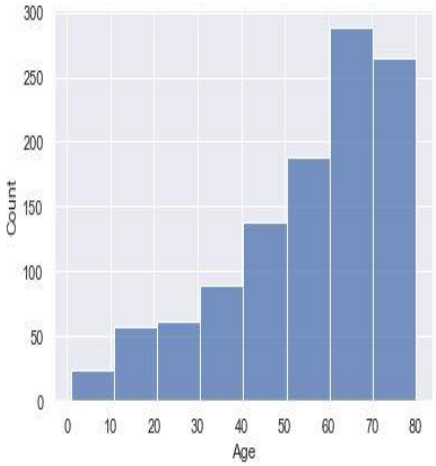
Out[8]: <AxesSubplot:xlabel='Age', ylabel='Count'>



Grouped bar plot

In [9]:
```
sns.histplot(cars_data['Age'],kde=False,bins=8)
```

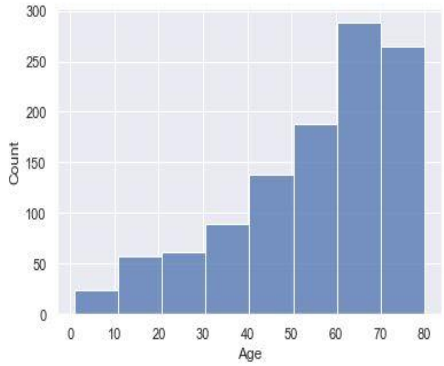
Out[9]: <AxesSubplot:xlabel='Age', ylabel='Count'>




Grouped bar plot

In [9]:
```
sns.histplot(cars_data['Age'],kde=False,bins=8)
```

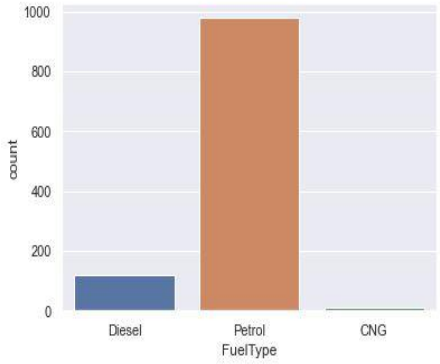
Out[9]: <AxesSubplot:xlabel='Age', ylabel='Count'>



BAR PLOT

In [10]:
```
#Frequency distribution of categorial variable 'FuelType'
sns.countplot(x="FuelType",data=cars_data)
```

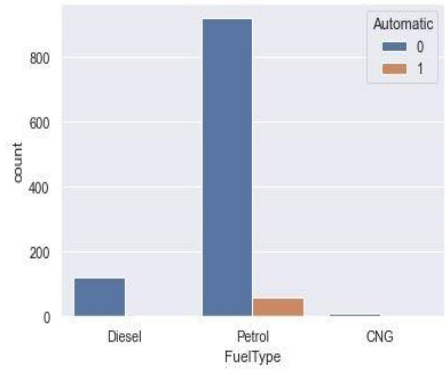
Out[10]: <AxesSubplot:xlabel='FuelType', ylabel='count'>




Grouped bar plot

In [11]:
```
# Grouped bar plot FuelType and Automatic
sns.countplot(x="FuelType",data=cars_data, hue='Automatic')
```

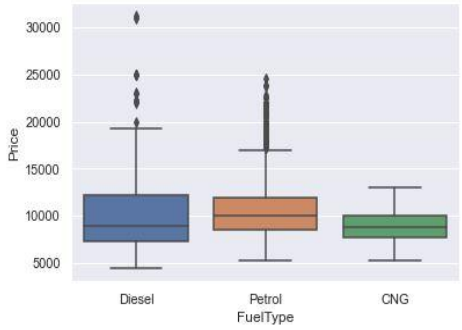
Out[11]: <AxesSubplot:xlabel='FuelType', ylabel='count'>



Box and Whiskers plot-Numerical Variable

In [13]:
```
#frequency distribution of categorical variable 'FuelType'
sns.boxplot(x=cars_data["FuelType"],y=cars_data["Price"])
```

Out[13]: <AxesSubplot:xlabel='FuelType', ylabel='Price'>




In [14]:
```
#frequency distribution of categorical variable 'FuelType'
sns.boxplot(x=cars_data["FuelType"],y=cars_data["Price"],hue="Automatic", data=cars_data)
```

Out[14]: <AxesSubplot:xlabel='FuelType', ylabel='Price'>