Parul™ University

NAAC A++ ACCREDITED UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

BACHELOR OF TECHNOLOGY

MACHINE LEARNING

(303105354)

VI SEMESTER
6B3

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

Laboratory Manual

FACULTY OF ENGINEERING AND TECHNOLOGY
COMPUTER SCIENCE AND ENGINEERING
MACHINE LEARNING LABORATORY(303105354)
B.Tech-CSE, 6th SEMESTER, 3rd YEAR
ENROLLMENT NO. 21030XXXXXXXX

CERTIFICATE

This is to certify that Mr./Ms. **XXXXXXXXXXXXX** with Enrollment No. **21030XXXXXXXXX** has successfully completed her laboratory experiments in **Machine Learning Laboratory (303105354)** From the Department of Computer **Science and Engineering** during the academic year **2025 – 2026**

Date of Submission: …… Staff In Charge: ……..……

Head of Department: …….

TABLE OF CONTENT

Experiment - 1

Aim: Dealing With Data Using Stats Numpy and Pandas Library.

Code:

```
import statistics as st
a=[20, 10, 50, 10, 21, 90]
s=st.mode(a)
print("Mode:",s)
```

```
Mode: 10
```

```
import statistics as st
a=[20, 10, 50, 10, 21, 90]
m=st.mean(a)
print("Mean:",m)
```

```
Mean: 33.5
```

```
pip install numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.25.2)
```

```
import numpy as np
b=np.array([[10, 20, 30, 50, 10],[1, 2, 3, 5, 6]])
print(b)
print("Dim:",b.ndim)
```

```
[[10 20 30 50
  10]
 [ 1  2  3  5  6]]
Dim: 2
```

```
import numpy as np
b=np.array([[10, 20, 30, 50, 10],[1, 2, 3, 5, 6],[1,1,1,1,1]])
print(b)
print("Dim:",b.ndim)
print("Shape:",b.shape)
print("Size:",b.size)
print("Item Size:",b.itemsize)
print("Data type:",b.dtype)
```

```
[[10 20 30 50 10]
 [ 1  2  3  5  6]
 [ 1  1  1  1  1]]
Dim: 2
Shape: (3, 5)
Size: 15
Item Size: 8
Data type: int64
```

```
c=np.empty((5,5))
```

```
array([[4.85494045e-310, 0.00000000e+000, 6.28083585e-038,
        2.16385130e+190, 1.94918964e-153],
       [4.90838520e+252, 1.30304358e-142, 1.06396443e+224,
        1.70100503e+256, 5.49109388e-143],
       [9.30537467e+199, 2.57018365e-057, 1.56067931e+184, 3.53933111e-
        061, 4.85198335e-033],
       [4.40779861e-143, 5.49810101e-143, 6.19651793e-091,
        1.49655354e-076, 1.57712453e-052],
       [4.29600452e-096, 4.82337433e+228, 3.67152456e-062, 6.01346930e-
        154, 1.81816158e-321]])
```

```
d=np.arange(0,100,2
)d
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32,
       34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66,
       68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98])
```

```
e=np.reshape(d,(10,5))
```

```
f=np.reshape(e,(2,5,5))
```

```
print(f)
```

```
print("Last element:",d[10])
```

```
print("ELement upto 10:",d[:10])
```

```
Lement upto 10: [ 0  2  4  6  8 10 12 14 16 18]
```

```
print("Last 5 element:",d[-5:])
```

```
Last 5 element: [90 92 94 96 98]
```

```
print("Alternative Numbers:",d[::2])
```

```
Alternative Numbers: [ 0  4  8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96]
```

```
pip install pandas
```

```
Collecting panda
  Downloading panda-0.3.1.tar.gz (5.8 kB) Preparing metadata (setup.py) ... done
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from panda) (67.7.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from panda) (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->panda) (3.3.2) Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->panda) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from
```

```
requests->panda) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from
requests->panda) (2024.6.2) Building wheels for collected packages: panda
Building wheel for panda (setup.py) ... done
Created wheel for panda: filename=panda-0.3.1-py3-none-any.whl size=7239
sha256=403d903fdbfc2160794713c5b6c16732975823999c068021f Stored in directory:
/root/.cache/pip/wheels/0e/8b/c3/ff9cbde1fffd8071cff8367a86f0350a1ce30a8d31b6a432e9
Successfully built panda
Installing collected
packages: panda Successfully
installed panda-0.3.1
```

```
import pandas as pd
p_dict={'pid':[1,2,3,4,5],
'pid':[1,2,3,4,5],
'pname':['A','B','C','D','E’],'price':[10,20,30,40,50],
'vendor':['V1','V2','V3','V4','V5']}
p_dict
```

```
{'pid': [1, 2, 3, 4, 5],
 'pname': ['A', 'B', 'C', 'D', 'E'],'price': [10, 20, 30, 40, 50],
 'vendor': ['V1', 'V2', 'V3', 'V4', 'V5']}
```

```
grt=pd.DataFrame(p_dict) grt
```

	pid	pname	price	vendor
0	1	A	10	V1
1	2	B	20	V2
2	3	C	30	V3
3	4	D	40	V4

FACULTY OF ENGINEERING AND TECHNOLOGY
COMPUTER SCIENCE AND ENGINEERING
MACHINE LEARNING LABORATORY (303105354)
B.Tech-CSE, 6 SEMESTER, 3rd YEAR
ENROLLMENT NO. 21030XXXXXXXX

Experiment - 2

Aim: Data Analysis and Visualisation on Diwali Sales Database.

Code:

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

df=pd.read_csv('Diwali Sales
Data.csv',encoding='unicode_escape')df
```

	User_ID	Cust_name	Product_ID	Gender	Age Group	Age	Marital_Status	State	Zone	Occupation	Product_Category	O
0	1002903	Sanskriti	P00125942	F	26-35	28	0	Maharashtra	Western	Healthcare	Auto	
1	1000732	Kartik	P00110942	F	26-35	35	1	Andhra Pradesh	Southern	Govt	Auto	
2	1001990	Bindu	P00118542	F	26-35	35	1	Uttar Pradesh	Central	Automobile	Auto	
3	1001425	Sudevi	P00237842	M	0-17	16	0	Karnataka	Southern	Construction	Auto	
4	1000588	Joni	P00057942	M	26-35	28	1	Gujarat	Western	Food Processing	Auto	
...	...	...	...	...	...	...	...	...	...	...	...	
11246	1000695	Manning	P00296942	M	18-25	19	1	Maharashtra	Western	Chemical	Office	
11247	1004089	Reichenbach	P00171342	M	26-35	33	0	Haryana	Northern	Healthcare	Veterinary	
11248	1001209	Oshin	P00201342	F	36-45	40	0	Madhya Pradesh	Central	Textile	Office	
11249	1004023	Noonan	P00059442	M	36-45	37	0	Karnataka	Southern	Agriculture	Office	
11250	1002744	Brumley	P00281742	F	18-25	19	0	Maharashtra	Western	Healthcare	Office	

11251 rows × 15 columns

Next steps: Generate code with df | View recommended plots

```
df.shape
```

(11251, 15)

```
df.head()
```

	User_ID	Cust_name	Product_ID	Gender	Age Group	Age	Marital_Status	State	Zone	Occupation	Product_Category	Orders
0	1002903	Sanskriti	P00125942	F	26-35	28	0	Maharashtra	Western	Healthcare	Auto	1
1	1000732	Kartik	P00110942	F	26-35	35	1	Andhra Pradesh	Southern	Govt	Auto	3
2	1001990	Bindu	P00118542	F	26-35	35	1	Uttar Pradesh	Central	Automobile	Auto	3
3	1001425	Sudevi	P00237842	M	0-17	16	0	Karnataka	Southern	Construction	Auto	2

```
4  1000588  Joni  P00057942  M  26-35 28  1  Gujarat  Western  Food Processing Auto  2
```

Next steps: Generate code with df | View recommended plots

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11251 entries, 0 to 11250
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User_ID          11251 non-null  int64
 1   Cust_name        11251 non-null  object
 2   Product_ID       11251 non-null  object
 3   Gender           11251 non-null  object
 4   Age Group        11251 non-null  object
 5   Age              11251 non-null  int64
 6   Marital_Status   11251 non-null  int64
 7   State            11251 non-null  object
 8   Zone             11251 non-null  object
 9   Occupation       11251 non-null  object
 10  Product_Category 11251 non-null  object
 11  Orders           11251 non-null  int64
 12  Amount           11239 non-null  float64
 13  Status           0 non-null      float64
 14  unnamed1         0 non-null      float64
dtypes: float64(3), int64(4), object(8)
memory usage: 1.3+ MB
```

```
df.drop(['Status', 'unnamed1'], axis=1, inplace=True)
```

```
df.drop(['Status','unnamed1'],axis=1 ,inplace=True)
```

```
pd.isnull(df).sum()
```

```
User_ID           0
Cust_name         0
Product_ID        0
Gender            0
Age Group         0
Age               0
Marital_Status    0
State             0
Zone              0
Occupation        0
Product_Category  0
Orders            0
Amount           12
dtype: int64
```

```
df.dropna(inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>Index: 11239 entries, 0 to 11250
Data columns (total 13 columns):
```

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   User_ID         11239 non-null  int64
 1   Cust_name       11239 non-null  object
 2   Product_ID      11239 non-null  object
 3   Gender          11239 non-null  object
 4   Age Group       11239 non-null  object
 5   Age             11239 non-null  int64
 6   Marital_Status  11239 non-null  int64
 7   State           11239 non-null  object
 8   Zone            11239 non-null  object
 9   Occupation      11239 non-null  object
 10  Product_Category  11239 non-null  object
 11  Orders          11239 non-null  int64
 12  Amount          11239 non-null  float64
dtypes: float64(1), int64(4),
object(8)memory usage: 1.2+ MB
```

```
df['Amount']=df['Amount'].astype('int')
```

```
df['Amount'].d
```

```
types
```

```
dtype('int64')
```

```
df.columns
```

```
Index(['User_ID', 'Cust_name', 'Product_ID', 'Gender',
       'Age Group', 'Age', 'Marital_Status', 'State',
       'Zone', 'Occupation', 'Product_Category','Orders',
       'Amount'],
        dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>Index: 11239 entries, 0 to 11250
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   User_ID         11239 non-null  int64
 1   Cust_name       11239 non-null  object
 2   Product_ID      11239 non-null  object
 3   Gender          11239 non-null  object
 4   Age Group       11239 non-null  object
 5   Age             11239 non-null  int64
 6   Marital_Status  11239 non-null  int64
 7   State           11239 non-null  object
 8   Zone            11239 non-null  object
 9   Occupation      11239 non-null  object
 10  Product_Category  11239 non-null  object
 11  Orders          11239 non-null  int64
 12  Amount          11239 non-null  float64
dtypes: float64(1), int64(4),
object(8)memory usage: 1.2+ MB
```

```
df.describe()
```

	User_ID	Age	Marital_Status	Orders	Amount
count	1.123900e+04	11239.000000	11239.000000	11239.000000	11239.000000
mean	1.003004e+06	35.410357	0.420055	2.489634	9453.610553
std	1.716039e+03	12.753866	0.493589	1.114967	5222.355168
min	1.000001e+06	12.000000	0.000000	1.000000	188.000000
25%	1.001492e+06	27.000000	0.000000	2.000000	5443.000000
50%	1.003064e+06	33.000000	0.000000	2.000000	8109.000000
75%	1.004426e+06	43.000000	1.000000	3.000000	12675.000000
max	1.006040e+06	92.000000	1.000000	4.000000	23952.000000

```
df.describe(include='all')
```

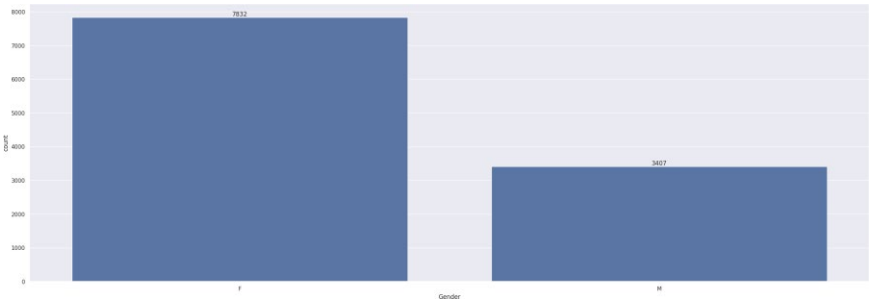
count	1.123900e+04	11239	11239	11239	11239	11239.000000	
unique	NaN	1250	2350	2	7	NaN	N
top	NaN	Vishakha	P00265242	F	26-35	NaN	N
freq	NaN	42	53	7832	4541	NaN	N
mean	1.003004e+06	NaN	NaN	NaN	NaN	35.410357	0.4200
std	1.716039e+03	NaN	NaN	NaN	NaN	12.753866	0.4935
min	1.000001e+06	NaN	NaN	NaN	NaN	12.000000	0.0000
25%	1.001492e+06	NaN	NaN	NaN	NaN	27.000000	0.0000
50%	1.003064e+06	NaN	NaN	NaN	NaN	33.000000	0.0000
75%	1.004426e+06	NaN	NaN	NaN	NaN	43.000000	1.0000
max	1.006040e+06	NaN	NaN	NaN	NaN	92.000000	1.0000

```
df[['Age','Orders','Amount']].describe()
```

	Age	Orders	Amount
count	11239.000000	11239.000000	11239.000000
mean	35.410357	2.489634	9453.610553
std	12.753866	1.114967	5222.355168
min	12.000000	1.000000	188.000000
25%	27.000000	2.000000	5443.000000
50%	33.000000	2.000000	8109.000000
75%	43.000000	3.000000	12675.000000
max	92.000000	4.000000	23952.000000

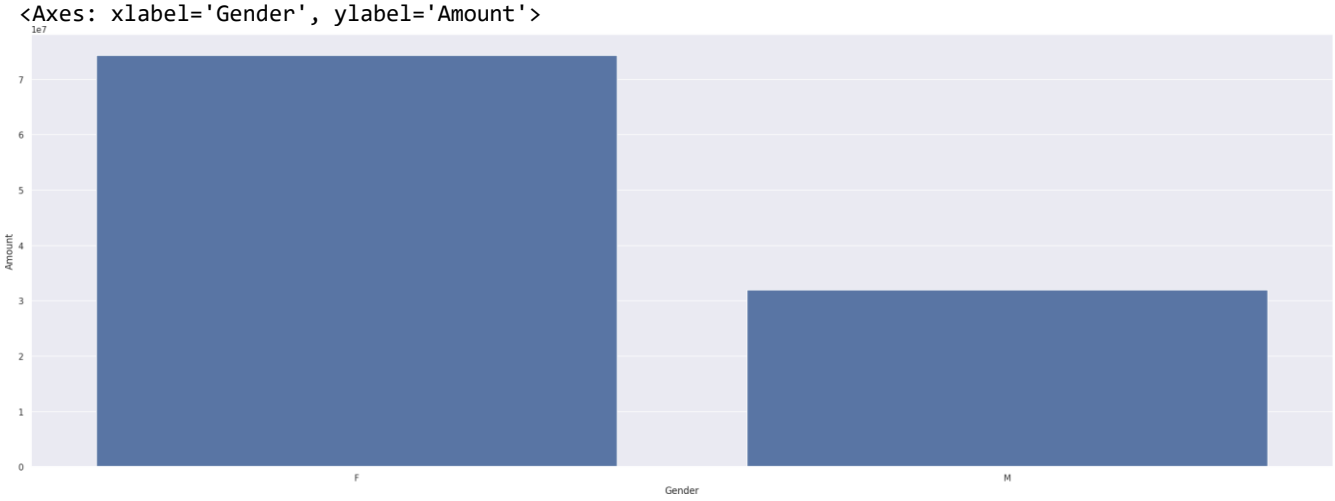
```
#plotting a bar chart for Gnder and it's Count ax=sns.countplot(x='Gender',data=df)
```

```
for bars in ax.containers: ax.bar_label(bars)
```

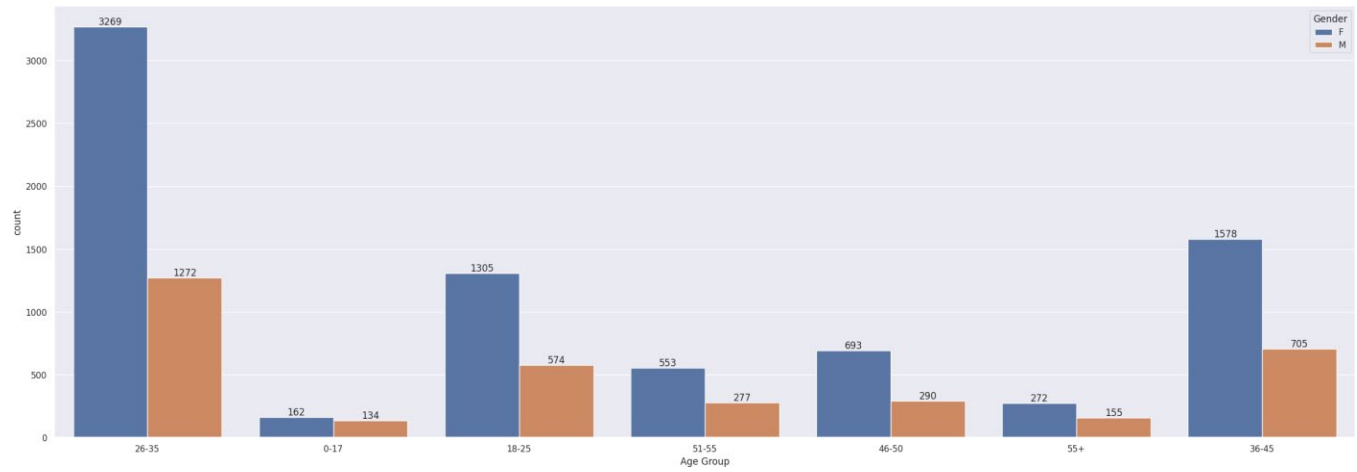



```
sales_gen=df.groupby(['Gender'],as_index=False)['Amount'].sum().sort_values(by='A
mount',ascending=False) sns.barplot(x='Gender',y='Amount',data=sales_gen)
```

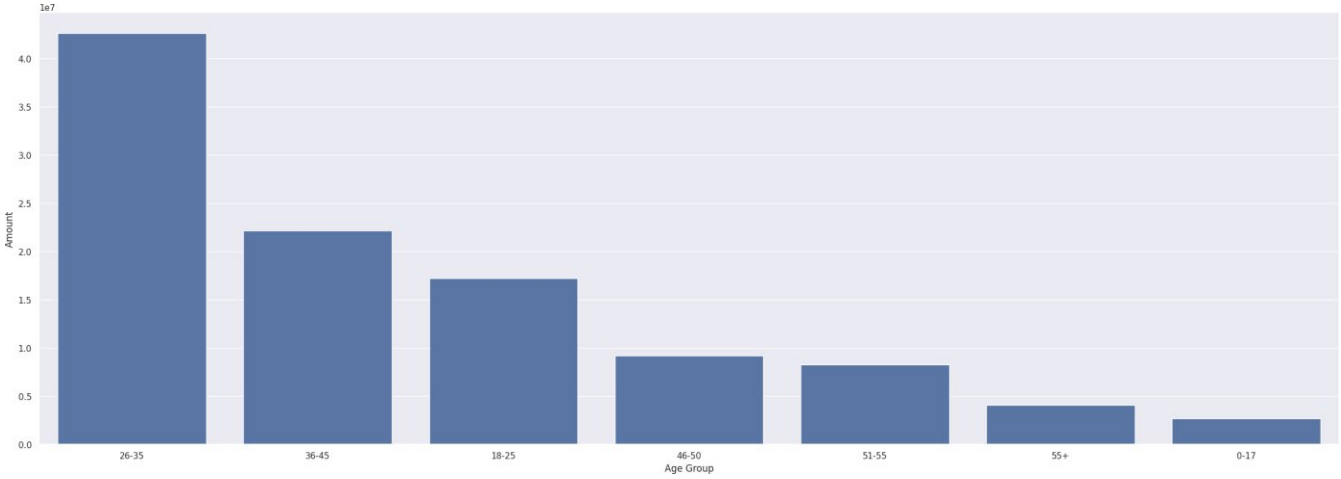
```
<Axes: xlabel='Gender', ylabel='Amount'>
```




```
ax=sns.countplot(data=df,x='Age Group',hue='Gender')
for bars in ax.containers:
ax.bar_label(bars)
```



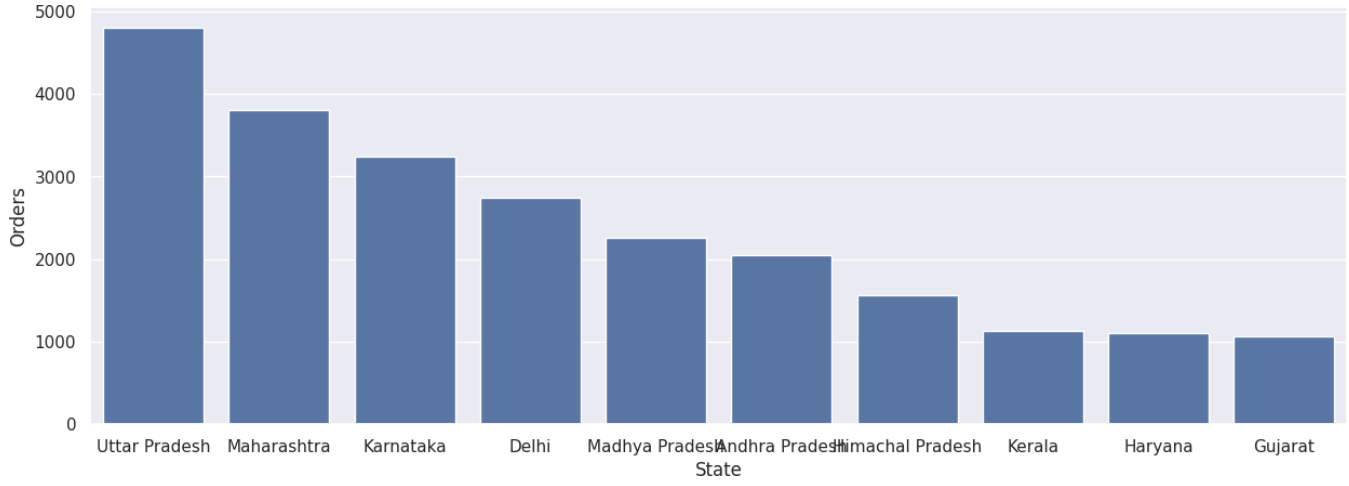

```
sales_age=df.groupby(['Age
Group'],as_index=False)['Amount'].sum().sort_values(by='Amount',ascending=False)
sns.barplot(x='Age Group',y='Amount',data=sales_age)
```



<Axes: xlabel='Age Group', ylabel='Amount'>

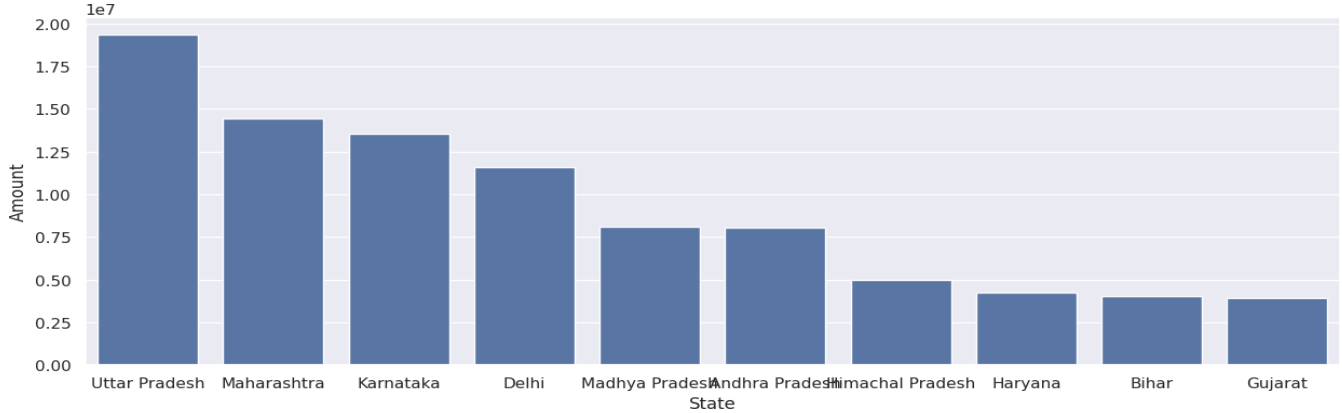
```
sales_states=df.groupby(['State'],as_index=False)['Orders'].sum().sort_values(by='Orders'
,ascending=False).head(10)sns.set(rc={'figure.figsize':(15,5)})
sns.barplot(x='State',y='Orders',data=sales_states
```

<Axes:
xlabel='State',ylabel='Orders'>



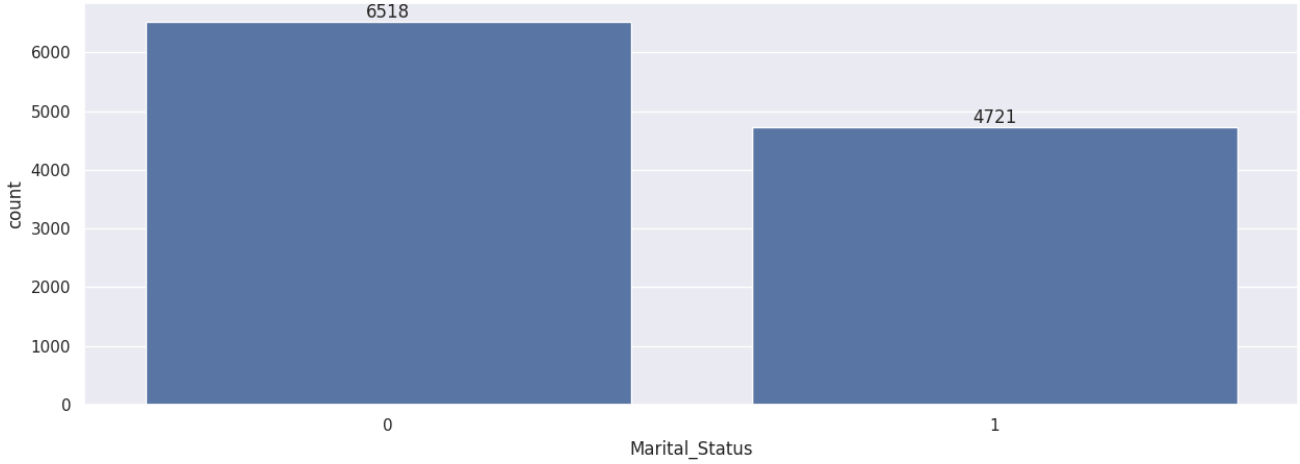

```
sales_state=df.groupby(['State'],as_index=False)['Amount'].sum().sort_values(by='Amount',
ascending=False).head(10)sns.set(rc={'figure.figsize':(15,5)})
sns.barplot(x='State',y='Amount',data=sales_state)
```

<Axes: xlabel='State', ylabel='Amount'>



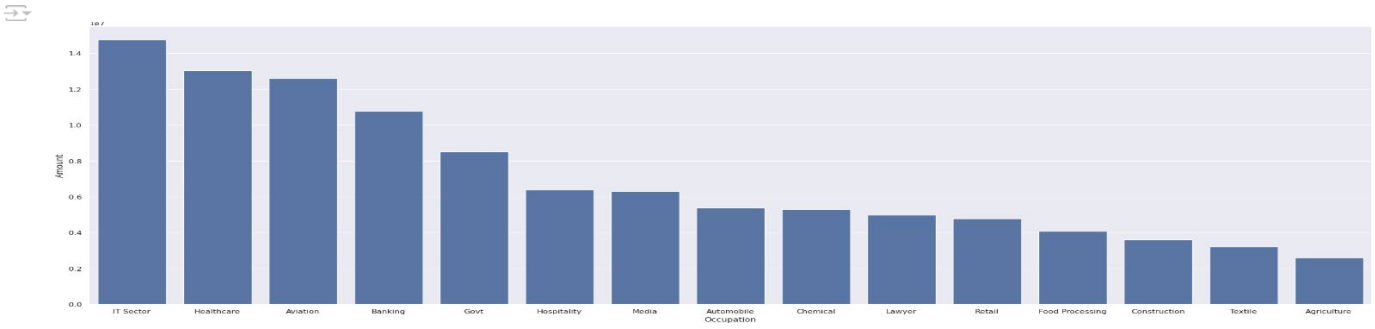

```
ax=sns.countplot(data=df,x='Marital_
Status')
sns.set(rc={'figure.figsize':(7,5)})
```

```
for bars in ax.containers:
    ax.bar_label(bars)
```




```
sns.set(rc={'figure.figsize':(30,10)})
sales_state=df.groupby(['Occupation'],as_index=False)['Amount'].sum().sort_values(by='A
mount',ascending=False)sns.barplot(x='Occupation',y='Amount',data=sales_state)
```

```
<Axes: xlabel='Occupation', ylabel='Amount'>
```



FACULTY OF ENGINEERING AND TECHNOLOGY
COMPUTER SCIENCE AND ENGINEERING
MACHINE LEARNING LABORATORY (303105354)
B.Tech-CSE, 6th SEMESTER, 3rd YEAR
ENROLLMENT NO. 21030XXXXXXXX

Experiment - 3

Aim: Implementation On Linear Regression And Logstic Regression.

Code:

Liner Regression:

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

df=pd.read_csv('placement.csv',encoding='unico
de_escape')
df
```

	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25
3	7.42	3.67
4	6.94	3.57
...	...	...
195	6.93	2.46
196	5.89	2.57
197	7.21	3.24
198	7.63	3.96
199	6.22	2.33

200 rows × 2 columns

Generate code with df | View recommended plots

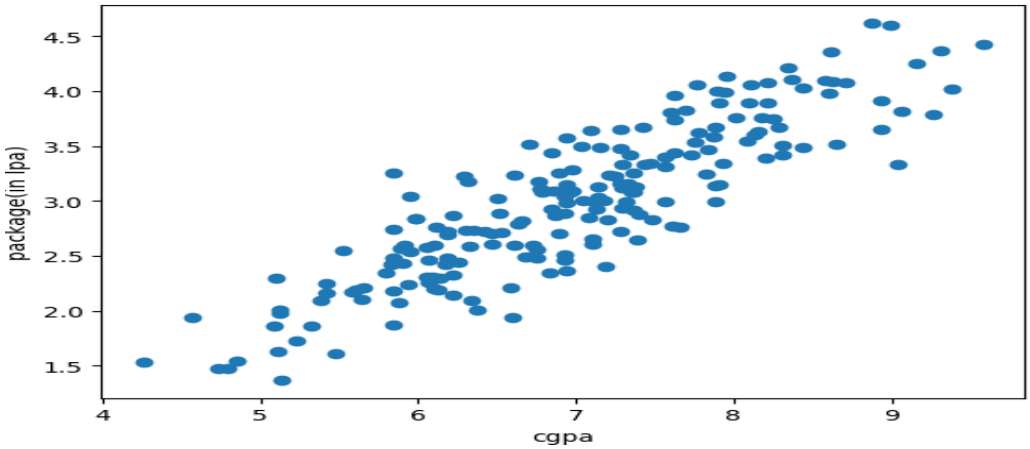
Next steps:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 2 columns):
 #  Column Non-Null Count  Dtype
---  ------ -------------- -----

 0  cgpa   200 non-null float64
 1  package 200 non-null
    float64 dtypes: float64(2)
memory usage: 3.2 KB
```

```
plt.scatter(df['cgpa'],df['package'])
          plt.xlabel('cgpa')
          plt.ylabel('package(in lpa)')
```

```
Text(0, 0.5, 'package(in lpa)')
```




```
x=df.iloc[:,0:1]
y=df.iloc[:,-
1] y
```

```
0     3.26
1     1.98

        2   3.25
        3   3.67
        4   3.57
          ...
      195   2.46
      196   2.57
      197   3.24
      198   3.96
      199   2.33
    Name: package, Length: 200, dtype: float64
```

```
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,random_state=2)

from sklearn.linear_model import
LinearRegression lr=LinearRegression()
lr.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
x_test
```

115	8.35
35	6.87
12	8.94

92	7.90
13	6.93
126	5.91
174	7.32
2	7.82
44	5.09
3	7.42
113	6.94
14	7.73
23	6.19
25	7.28
6	6.73
134	7.20
165	8.21
173	6.75
45	7.87
65	7.60
48	8.63
122	5.12
178	8.15
64	7.36
9	8.31
57	6.60
78	6.59
71	7.47
128	7.93
176	6.29
131	6.37
53	6.47

Next steps: Generate code with x_test | View recommended plots

```
y_test
```

```
112  4.10
29   3.49
182  2.08
199  2.33
193  1.94
85   1.48
10   1.86

35  2.87
```

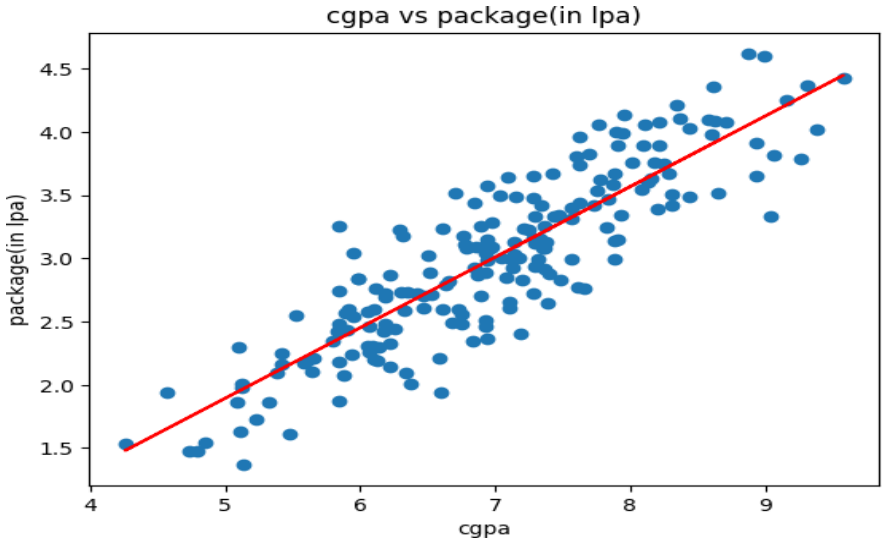
```
12  3.65
92  4.00
13  2.89
126 2.60
174 2.99
2   3.25
44  1.86
3   3.67
113 2.37
14  3.42
23  2.48
25  3.65
6   2.60
134 2.83
165 4.08
173 2.56
45  3.58
65  3.81
48  4.09
122 2.01
178 3.63
64  2.92
9   3.51
57  1.94
78  2.21
71  3.34
128 3.34
176 3.23
131 2.01
53  2.61
Name: package, dtype: float64
```

```
lr.predict(x_test.iloc[0].values.reshape(1,1))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature
  names, but LinearRegression warnings.warn(
array([3.89111601])
```

```
plt.scatter(df['cgpa'],df['package'])
plt.plot(x_train,lr.predict(x_train),color='re
d') plt.title('cgpa vs package(in lpa)')
plt.xlabel('cgpa')
plt.ylabel('package(in lpa)')
```

```
Text(0, 0.5, 'package(in lpa)')
```




```
m=lr.coef_
b=lr.intercept_
m*8.58+b
```

```
array([3.89111601])
```

```
m*9.5+b
```

```
array([4.40443183])
```

```
m*100+b
```

```
array([54.89908542])
```

```
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
y_pred=lr.predict(x_test)
score=r2_score(y_test,y_pred)
print(f'Accuracy score:{score}')
```

```
Accuracy score:0.780730147510384
```

```
y_pred
```

```
array([3.89111601, 3.09324469, 2.38464568, 2.57434935, 1.6537286 ,1.77647803, 2.07219258, 2.93143862,
3.76278706,    2.93701814,
            4.09197872, 3.51170867, 2.97049525, 2.40138424, 3.18809652,3.46707251, 1.94386362, 3.24389172,
2.97607477, 3.41685683,
            2.55761079, 3.16577844, 2.85890486, 3.12114229, 3.68467378,2.8700639 , 3.49497011, 3.34432308,
3.91901361, 1.96060218,
            3.65119666, 3.2104146 , 3.74046898, 2.7863711 , 2.78079158,3.27178932, 3.52844723, 2.61340599,
2.65804215, 2.71383735])
```

```
MSE=mean_squared_error(y_test,y_pred)
print(f'Mean Squared Error:{MSE}')
```

```
Mean Squared Error:0.12129235313495527
```

Multi linear Regression:

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

df =

pd.read_csv('50_Startups.csv'

) df.head()
```

	R&D Spend	Administration	marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

```
df.isnull().sum()
```

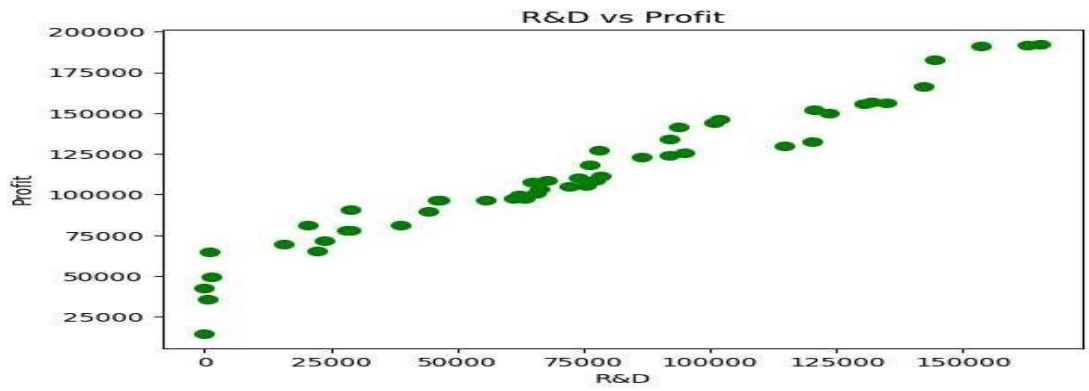
```
R&D Speud       0
Administration  0
Marketing Spend 0
State           0
Profit          0
dtype: int64
```

```
df[”State”].unique()
```

```
array(['New York', 'California', 'Florida'], dtype=object)
```

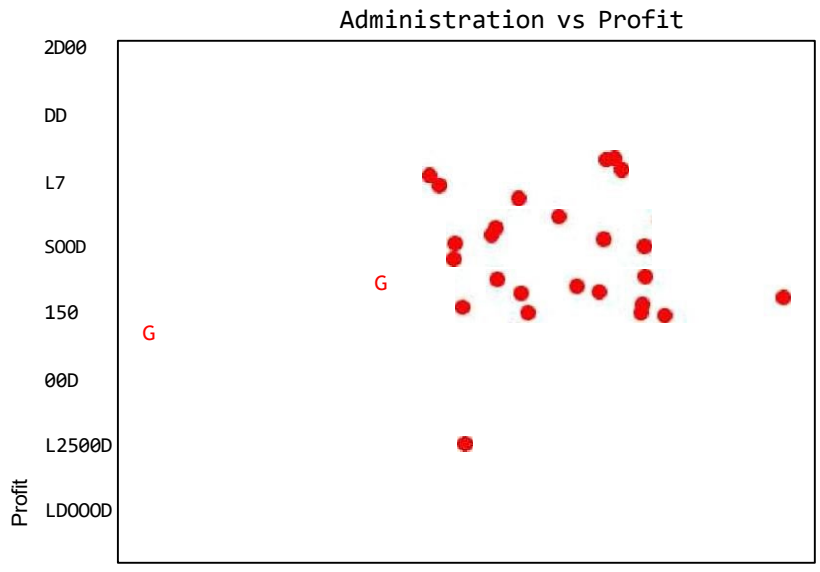
```
#Plot R&D vs
Profit..... x1 =
df.iloc[:, 0].values
y1 = df.iloc[:, -1].values
plt.scatter(x1,y1,colon='Gree
n',s=50) plt.xlabel('R&D')
plt.y1abel('Profit')
```

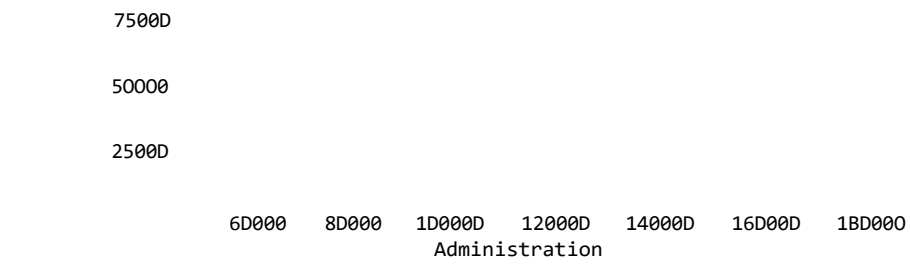
```
plt.title('R&D vs Profit')
plt.show()
```




```
x1 = df.i1oc[:,
1].values
y1 = df.iloc[:, -1].values
plt.scatter(x1,y1,color='Red',
s=50)
plt.x1abe1('Administration')
plt.y1abe1('Profit')
plt.title('Administration vs
Profit')
```

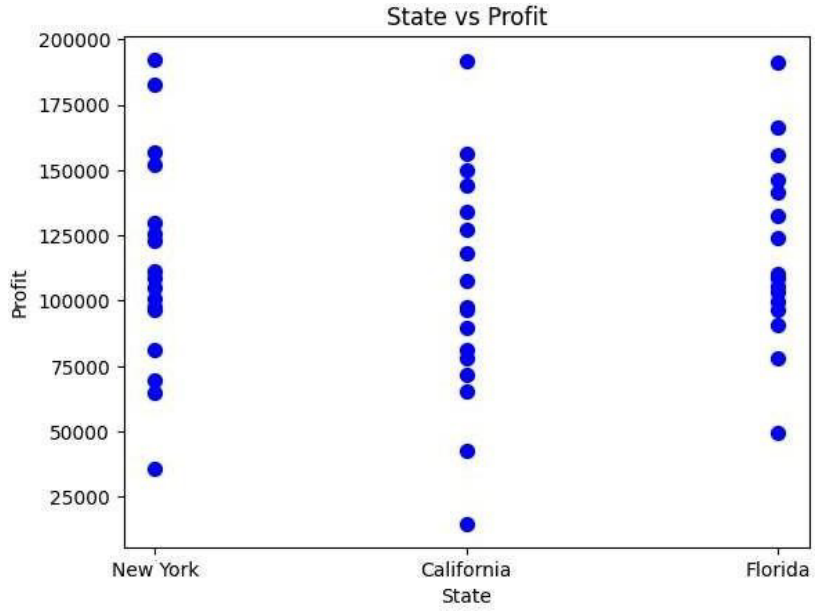
```
plt.show()
```






```
x1 = df.iloc[:,
2].values
y1 = df.i1oc[:, -1].values
plt.scatter(x1,y1,co1or='Black
',s=50) plt.x1abe1('Marketing
Spend') plt.ylabel('Profit')
plt.title('Marketing Spend vs
Profit') plt.show()
```


```
x1 = df.iloc[:, 3].values
y1 = df.iloc[:, -1].values
plt.scatter(x1,y1,colon='Blue',s=50)
plt.x1abel('State')
plt.y1abel('Profit')
plt.title('State vs Profit') plt.show()
```




```
df.State.value_coun

ts()
```

```
State
New York    17
California  17
Florida     16
Name: count, dtype: int64
```

```
df['New York'] = np.where(df['State']=='New York', 1, 0)
df['California'] = np.where(df['State']=='California', 1, 0)
df['Florida'] = np.where(df['State']=='Florida', 1, 0)
df.drop(columns=['State'], axis=1, inplace=True)


df.head()
```

	R&D Spend	Adm1n1strat1on	Narket1ng Spend	Profit	New York	Ca11forn1a	Florida
0	165349.20	136897.80	471784.10	192261.83	1	0	0
1	162597.70	151377.59	443898.53	191792.06	0	1	0
2	153441.51	101145.55	407934.54	191050.39	0	0	1
3	144372.41	118671.85	383199.62	182901.99	1	0	0
4	142107.34	91391.77	3óó168.42	166187.94	0	0	1

```
y='Profit'
x=df.columns.tolist()
x . remove(y)
```

```
['R&D Spend', 'Administration', 'Marketing Spend', 'New
    York','California', 'Florida']
x=df[x].values


y=df[y].values
```

```
array([[1.6534920e+05, 1.3689780e+05, 4.7178410e+05, 1.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [1.6259770e+05, 1.5137759e+05, 4.4389853e+B5, 0.B000000e+B0,
        1.0000000e+00, 0.0000000e+00],
       [1.5344151e+05,  1.0114555e+05,  4.0793454e+05,  0.0000000e+00,
        0.0000000e+00,  1.0000000e+00],
       [1.4437241e+05,  1.1867185e+05,  3.8319962e+B5,  1.0000000e+B0,
        0.0000000e+00, 0.0000000e+00],
       [1.4210734e+05, 9.1391770e+04, 3.6616842e+05, 0.0000000e+00,
        0.0000000e+00, 1.0000000e+00],
       [1.3187690e+05, 9.9814710e+04, 3.6286136e+05, 1.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [1.3461546e+05, 1.4719887e+05, 1.2771682e+05, 8. 8888888e+88,
        1.0000000e+00, 0.0000000e+00],
       [1.3029813e+05, 1.4553006e+05, 3.2387668e+05, 0.0000000e+00,
        0.0000000e+00, 1.0000000e+00],
       [1.2054252e+05, 1.4871895e+05, 3.1161329e+05, 1.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [1.2333488e+05, 1.0867917e+05, 3.0498162e+05, 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       [1.0191308e+05, 1.1059411e+B5, 2.2916095e+05,  0.0000000e+00,
        0.0000000e+00, 1.0000000e+00],
       [1.0067196e+05, 9.1790610e+04, 2.4974455e+05, 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       [9.3863750e+04, 1.2732038e+05, 2.4983944e+05, 0.0000000e+00,
        0.0000000e+00, 1.0000000e+00],
        [9.1992390e+04, 1.3549507e+05, 2. 5266493e+05, 1.0000000e+00, 0.0000000e+00],
       [1.1994324e+05, 1.5654742e+05, 2. 5651292e+05,
        0.0000000e+B0, 1.0000000e+00], 0.0000000e+00, 0.0000000e+00,
        [1.1452361e+05, 1.2261684e+05, 2.6177623e+05, 1.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [7.8013110e+04, 1.2159755e+05, 2.6434606e+B5, 0.00B0000e+B0,
        1.0000000e+00, 0.0000000e+00],
       [9.4657160e+04, 1.4507758e+05, 2.8257431e+05, 1.0000000e+00,
       0.0000000e+00, 0.0000000e+00], 0.0000000e+00, 1.0000000e+00],
       [8.6419700e+04, 1.535l4l1e+05, 0.0000000e+00, 1.0000000e+00,
       0.0000000e+00, 0.0000000e+00],
       [7.6253860e+04, 1.1386730e+05, 2.9866447e+05, 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       [7.8389470e+04, 1.5377343e+05, 2.9973729e+05, 1.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [7.3994560e+04, 1.2278275e+05, 3.0331926e+05, 0.0000000e+00,
        0.0000000e+00, 1.0000000e+00],
       [6.7532530e+04, 1.0575103e+05, 3.0476873e+05, 0.0000000e+00,
        0.0000000e+00, 1.0000000e+00],
       [7.7044010e+04,  9.9281340e+04,  1.4057481e+05,  1.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [6.4664710e+04, 1.3955316e+05, 1.3796262e+05, 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       [7.5328870e+04, 1.4413598e+05, 1.3405007e+05, 0.0000000e+00,
        0.0000000e+00, 1.0000000e+00],
       [7.2107600e+04,  1.2786455e+05,  3.5318381e+05,  1.0000000e+00,
        0.0000000e+00, 0.0000000e+80],
       [6.6851520e+04, 1.8264556e+05, 1.1814820e+B5, 0.0000000e+B0,
        0.0000000e+00, 1.0000000e+00],
```

```
y

     array([192261.83, 191792.06, 191050.39, 182901.99, 166187.94, 156991.12,
            156122.51, 155752. 6  , 152211.77,  149759. 96, 146121. 95, 144259. 4  ,
            141585.52, 134307.35, 132602.65, 129917.04, 126992.93, 125370.37,
            124266.9  , 122776.86, 118474.03, 111313.02, 110352.25, 108733.99,
            108552.04, 107404.34, 105733.54, 105008.31, 103282.38, 101004.64,
             99937.59,  97483.56,  97427.84,  96778.92,  96712.8 ,  96479.51,
             90708.19,  89949.14,  81229.06,  81005.76,  78239.91,  77798.83,
              71498. 49,   69758. 98,   65200. 33,   64926.08,   49496.75,   42559.73,
              35673. 41,   14681. 4  ]


 from sklearn.model_selection import train_test_split
 from sklearn.preprocessing import MinMaxScaler
 X_train,X_test,y_train,y_test =
 train_test_split(x,y,test_size=0.2,random_state=2)
 from sklearn.linear_model import LinearRegnession
 lr = LinearRegnession()

 #Transforming Data
 scaler =
 MinMaxScaler()
 X_train =
 scaler.fit_transform(X_train)
 X_test =
 scaler.fit_transform(X_test

 X_train

  Show hidden output

lr = LinearRegnession()

1r.fit(X_train,y_train)

 -J  * LinearRegression
     LinearRegression()

 lr.predict(X_test)

 array([ 74690.00101189,  47320.3206191 ,  99717.11382077, 155026.3673854 ,127299.64727389, 191081.36739932,
64811.64693661, 55816.26398031,
 85207.83029741, 109464.85584631])


 # Evaluating model
 from sklearn.metrics import n2_scone, mean_squared_error, mean_absolute_error

 y_pred =
 lr.predict(X_test)
 score = r2_score(y_test,
 y_pred)
 print(f'Accuracy Score : {score}')

 -t r Accuracy Score: 0.9782805755154735
```

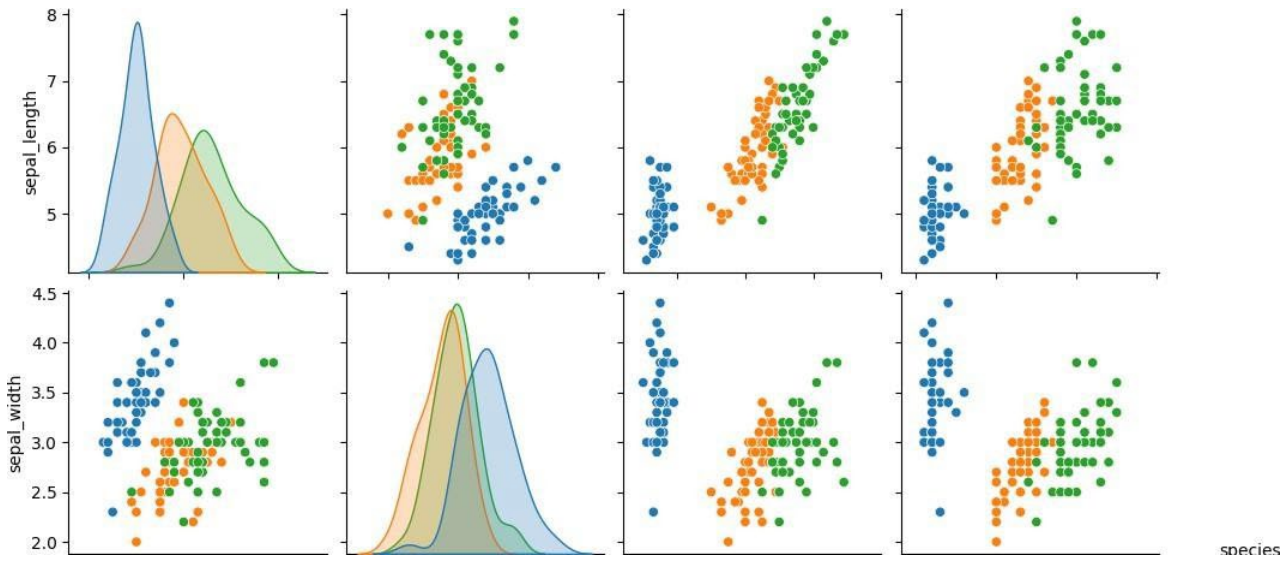
Logistic Regression:

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

```
df=sns.load_dataset('iris')
df.head()
```

	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

```
sns.pairplot(df,hue='species')
plt.show()
```




```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

```
x
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
y
```

```
0          setosa
1          setosa
2          setosa
3          setosa
4          setosa
          ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: species, Length: 150, dtype: object
```

```
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
=42)
```

```
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression()
classifier.fit(x_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
    Increase the number of iterations (max_iter) or scale the data as shown
        in: https://scikit-learn.org/stable/modules/preprocessing.html
    Please also refer to the documentation for alternative solver options:
        https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
      n_iter_i = _check_optimize_result(
```

LogisticRegression

```
classifier.predict(x_test)
```

```
array(['versicolor', 'setosa', 'virginica', 'versicolor',
       'versicolor', 'setosa', 'versicolor', 'virginica',
       'versicolor', 'versicolor',
       'virginica', 'setosa', 'setosa', 'setosa', 'setosa',
       'versicolor', 'virginica', 'versicolor', 'versicolor',
       'virginica', 'setosa',
       'virginica', 'setosa', 'virginica', 'virginica',
       'virginica',
       'virginica', 'virginica', 'setosa', 'setosa', 'setosa',
       'setosa', 'versicolor', 'setosa', 'setosa', 'virginica',
       'versicolor',
       'setosa'],  dtype=object)
```

```
from sklearn.metrics import accuracy_score,classification_report
y_pred=classifier.predict(x_test)
score=accuracy_score(y_test,y_pred)
print(score-0.08)
```

```
0.92
```

Experiment - 4

Aim: Implement the naïve Bayesian Classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering a few test data sets.

Naïve Bayesian Classifier:

```
import pandas as pd


df = pd.read_csv('/content/titanic.csv')
df.head()
```

Show hidden output

```
df.drop(['PassengerId', 'Name', 'SibSp', 'Parch', 'Ticket', 'Cabin', 'Embarked'], axis='columns', inplace=True)
df.head()
```

	Pclass	Sex	Age	Fare	Survived
0	3	male	22.0	7.2500	0
1	1	female	38.0	71.2833	1
2	3	female	26.0	7.9250	1
3	1	female	35.0	53.1000	1
4	3	male	35.0	8.0500	0

```
inputs = df.drop('Survived', axis='columns')
target = df.Survived


inputs.Sex = inputs.Sex.map({'male': 1, 'female': 2})


# dummies = pd.get_dummies(inputs.Sex)      # get dummies method used for mapping
# dummies.head(3)
```

	female	male
0	False	True
1	True	False
2	True	False

```
# inputs = pd.concat([inputs, dummies], axis='columns')      # concat method used for merging
# inputs.head(3)
```

	Pclass	Sex	Age	Fare	female	male
0	3	1	22.0	7.2500	False	True
1	1	2	38.0	71.2833	True	False
2	3	2	26.0	7.9250	True	False

```
# here we are dropping male column as cause of dummy variable trap theory, One column is enough to represent male vs female
# inputs.drop(['Sex', 'male'], axis='columns', inplace=True)
# inputs.head(3)


inputs.columns[inputs.isna().any()]
```

```
Index(['Age'], dtype='object')
```

```
inputs.Age[:10]
```

	Age
0	22.0
1	38.0
2	26.0
3	35.0
4	35.0
5	NaN
6	54.0
7	2.0
8	27.0
9	14.0

```
inputs.Age = inputs.Age.fillna(inputs.Age.mean())
inputs.head()
```

	Pclass	Sex	Age	Fare
0	3	1	22.0	7.2500
1	1	2	38.0	71.2833
2	3	2	26.0	7.9250
3	1	2	35.0	53.1000
4	3	1	35.0	8.0500

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(inputs, target, test_size=0.3)


from sklearn.naive_bayes import GaussianNB
model = GaussianNB()


model.fit(X_train, y_train)
```

```
GaussianNB()
```

```
model.score(X_test, y_test)
```

```
0.7798507462686567
```

```
# calculating the score using cross validation
# from sklearn.model_selection import cross_val_score
# cross_val_score(GaussianNB(), X_train, y_train, cv=5)


from sklearn.naive_bayes import BernoulliNB
model1 = BernoulliNB()


model1.fit(X_train, y_train)
```

```
BernoulliNB()
```

```
model1.score(X_test, y_test)
```

```
0.5895522388059702
```

```
from sklearn.linear_model import LogisticRegression
model2 = LogisticRegression()


model2.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
model2.score(X_test, y_test)
```

```
0.7947761194029851
```

Experiment - 5

Aim: Assuming a set of documents that need to be classified, use the Naïve Bayesian Classifier model to perform this task.

Naïve Bayesian Classifier:

```
import pandas as pd

df = pd.read_csv("/content/spam.csv")
df.head()
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
df.groupby('Category').describe()
```

	Message				
	count	unique	top		freq
Category					
ham	4825	4516	Sorry, I'll call later		30
spam	747	641	Please call our customer service representativ...		4

```
df['spam'] = df['Category'].apply(lambda x: 1 if x=='spam' else 0)
df.head()
```

	Category	Message	spam
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.Message,df.spam, test_size=0.25)

from sklearn.feature_extraction.text import CountVectorizer
v = CountVectorizer()  # to know which word has occured how many times in dataset as it convert it into vector on bases of frequency co
X_train_count = v.fit_transform(X_train.values)
X_train_count.toarray()[:2]
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

```
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(X_train_count,y_train)
```

```
MultinomialNB()
```

```
emails = [
    'Hey mohan, can we get together to watch football game tomorrow?',
    'Upto 20% discount on parking, exclusive offer just for you. Dont miss this reward!'
]
emails_count = v.transform(emails)
model.predict(emails_count)
```

```
array([0, 1])
```

```
X_test_count = v.transform(X_test)
model.score(X_test_count, y_test)
```

```
0.9892318736539842
```

```
# Sklearn Pipeline

from sklearn.pipeline import Pipeline
clf = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('nb', MultinomialNB())
])

clf.fit(X_train, y_train)
```


```
clf.score(X_test, y_test)
```

```
0.9892318736539842
```

Experiment - 6

Aim: Decision tree-based ID3 algorithm.

ID3 algorithm:

```
import pandas as pd

df = pd.read_csv("/content/salaries.csv")
df.head()
```

	company	job	degree	salary_more_then_100k
0	google	sales executive	bachelors	0
1	google	sales executive	masters	0
2	google	business manager	bachelors	1
3	google	business manager	masters	1
4	google	computer programmer	bachelors	0

```
inputs = df.drop('salary_more_then_100k', axis='columns')

target = df['salary_more_then_100k']

from sklearn.preprocessing import LabelEncoder

le_company = LabelEncoder()        # LabelEncoder method to give labels as they don't have continueous values just like dummies, map me
le_job = LabelEncoder()
le_degree = LabelEncoder()

inputs['company_n'] = le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs
```

	company	job	degree	company_n	job_n	degree_n
0	google	sales executive	bachelors	2	2	0
1	google	sales executive	masters	2	2	1
2	google	business manager	bachelors	2	0	0
3	google	business manager	masters	2	0	1
4	google	computer programmer	bachelors	2	1	0
5	google	computer programmer	masters	2	1	1
6	abc pharma	sales executive	masters	0	2	1
7	abc pharma	computer programmer	bachelors	0	1	0
8	abc pharma	business manager	bachelors	0	0	0
9	abc pharma	business manager	masters	0	0	1
10	facebook	sales executive	bachelors	1	2	0
11	facebook	sales executive	masters	1	2	1
12	facebook	business manager	bachelors	1	0	0
13	facebook	business manager	masters	1	0	1
14	facebook	computer programmer	bachelors	1	1	0
15	facebook	computer programmer	masters	1	1	1

```
inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

inputs_n
```

	company_n	job_n	degree_n
0	2	2	0
1	2	2	1
2	2	0	0
3	2	0	1
4	2	1	0
5	2	1	1
6	0	2	1
7	0	1	0
8	0	0	0
9	0	0	1
10	1	2	0
11	1	2	1
12	1	0	0
13	1	0	1
14	1	1	0
15	1	1	1

target

	salary_more_then_100k
0	0
1	0
2	1
3	1
4	0
5	1
6	0
7	0
8	0
9	1
10	1
11	1
12	1
13	1
14	1
15	1

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

model = tree.DecisionTreeClassifier()

model.fit(inputs_n, target)
```

```
DecisionTreeClassifier()
```

```
model.score(inputs_n, target)
```

```
1.0
```

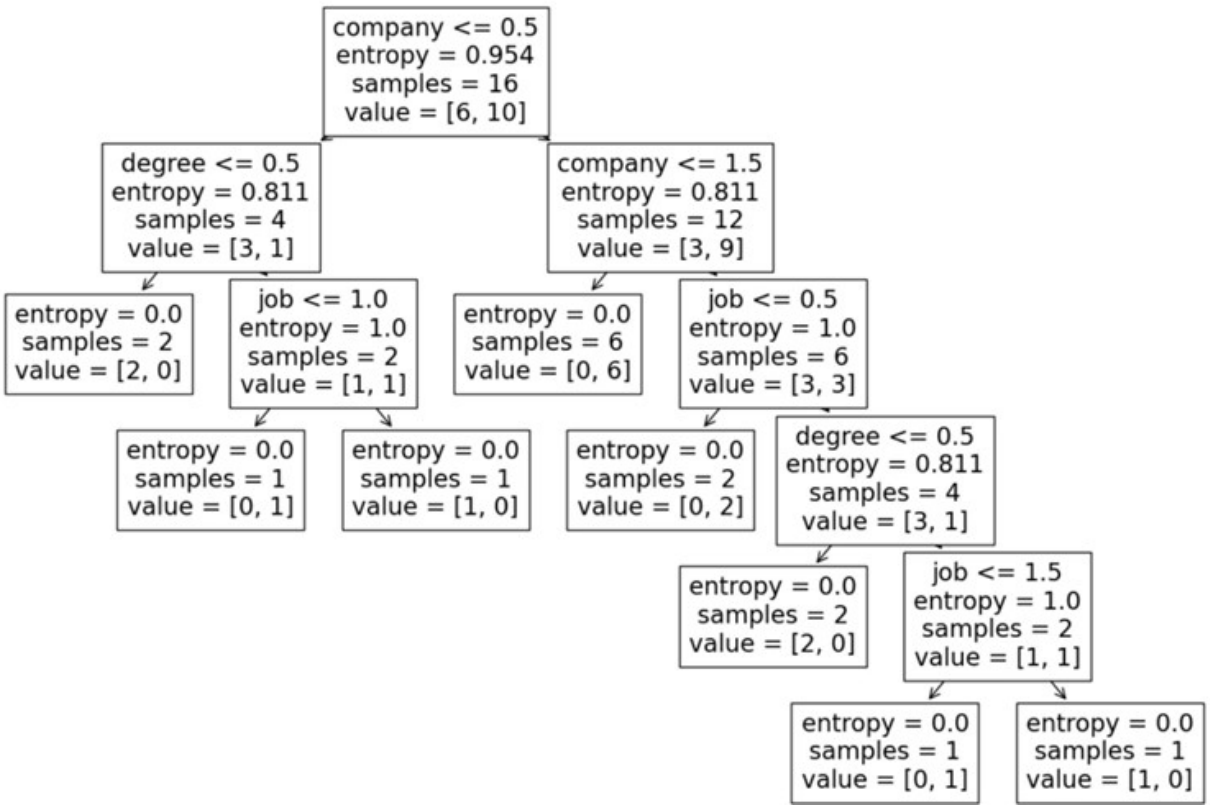
```
model.predict([[2, 1, 0]])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but DecisionTreeClas
  warnings.warn(
array([0])
```

```
model.predict([[2, 1, 1]])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but DecisionTreeClas
  warnings.warn(
array([1])
```

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

model = DecisionTreeClassifier(criterion='entropy')
model.fit(inputs_n, target)

plt.figure(figsize=(15, 10))
plot_tree(model, feature_names=df.columns[:-1])
plt.show()
```



Experiment - 7

Aim: Write a program to implement the K- Nearest Neighbour algorithm.

K- Nearest Neighbour:

```
import pandas as pd
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
import seaborn as sns


import warnings
warnings.filterwarnings('ignore')


cancer_ds = datasets.load_breast_cancer()

x = cancer_ds.data
y = cancer_ds.target


x.shape
```

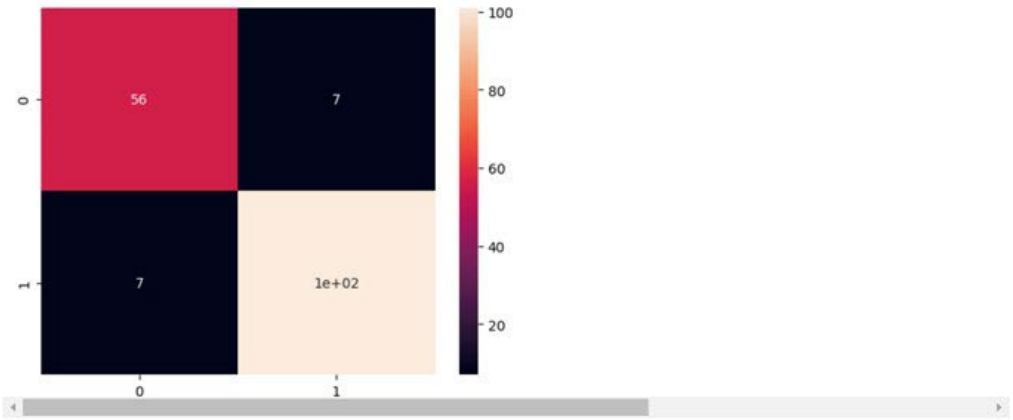
```
(569, 30)
```

```
y[:5]
```

```
array([0, 0, 0, 0, 0])
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)


knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(x_train, y_train)

y_pred = knn_model.predict(x_test)


cm = confusion_matrix(y_test, y_pred)
cls_rpt = classification_report(y_test, y_pred)

print(cm)
sns.heatmap(cm, annot=True)
print(cls_rpt)
```

```
[[ 56   7]
 [  7 101]]
              precision    recall  f1-score   support

           0       0.89      0.89      0.89        63
           1       0.94      0.94      0.94       108

    accuracy                           0.92       171
   macro avg       0.91      0.91      0.91       171
weighted avg       0.92      0.92      0.92       171
```




```
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(x_train, y_train)

y_pred = knn_model.predict(x_test)
```

```
cm = confusion_matrix(y_test, y_pred)
cls_rpt = classification_report(y_test, y_pred)

print(cm)
print(cls_rpt)
```

```
[[ 59   4]
 [  5 103]]
              precision    recall  f1-score   support

           0       0.92      0.94      0.93        63
           1       0.96      0.95      0.96       108

    accuracy                           0.95       171
   macro avg       0.94      0.95      0.94       171
weighted avg       0.95      0.95      0.95       171
```

```
x_train.shape
```

```
(398, 30)
```

```
knn_model = KNeighborsClassifier(n_neighbors=6)
knn_model.fit(x_train, y_train)

y_pred = knn_model.predict(x_test)
```

```
cm = confusion_matrix(y_test, y_pred)
cls_rpt = classification_report(y_test, y_pred)

print(cm)
print(cls_rpt)
```

```
[[ 59   4]
 [  7 101]]
              precision    recall  f1-score   support

           0       0.89      0.94      0.91        63
           1       0.96      0.94      0.95       108

    accuracy                           0.94       171
   macro avg       0.93      0.94      0.93       171
weighted avg       0.94      0.94      0.94       171
```

```
knn_model = KNeighborsClassifier(n_neighbors=9)
knn_model.fit(x_train, y_train)

y_pred = knn_model.predict(x_test)
```

```
cm = confusion_matrix(y_test, y_pred)
cls_rpt = classification_report(y_test, y_pred)

print(cm)
print(cls_rpt)
```

```
[[ 59   4]
 [  3 105]]
              precision    recall  f1-score   support

           0       0.95      0.94      0.94        63
           1       0.96      0.97      0.97       108

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171
```

```
knn_model = KNeighborsClassifier(n_neighbors=11)
knn_model.fit(x_train, y_train)

y_pred = knn_model.predict(x_test)
```

```
cm = confusion_matrix(y_test, y_pred)
cls_rpt = classification_report(y_test, y_pred)

print(cm)
print(cls_rpt)
```

```
[[ 59   4]
 [  2 106]]
              precision    recall  f1-score   support

           0       0.97      0.94      0.95        63
           1       0.96      0.98      0.97       108

    accuracy                           0.96       171
   macro avg       0.97      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

```
# hyper parameter tunning

from sklearn.model_selection import GridSearchCV

knn_model = KNeighborsClassifier()
params = {'n_neighbors':[3, 7, 9, 11, 13, 17, 19, 21]}

gscv = GridSearchCV(knn_model, param_grid=params, scoring='accuracy')
gscv.fit(x_train, y_train)
```


```
gscv.best_score_
```

```
0.9346518987341772
```

```
gscv.best_params_
```

```
{'n_neighbors': 11}
```

```
model = gscv.best_estimator_


y_pred = model.predict(x_test)
cm = confusion_matrix(y_test, y_pred)
cls_rpt = classification_report(y_test, y_pred)

print(cm)
print(cls_rpt)
```

```
[[ 59   4]
 [  2 106]]
              precision    recall  f1-score   support

           0       0.97      0.94      0.95        63
           1       0.96      0.98      0.97       108

    accuracy                           0.96       171
   macro avg       0.97      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

Experiment - 8

Aim: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using K- Means Algorithm.

Clustering using K- Means Algorithm:

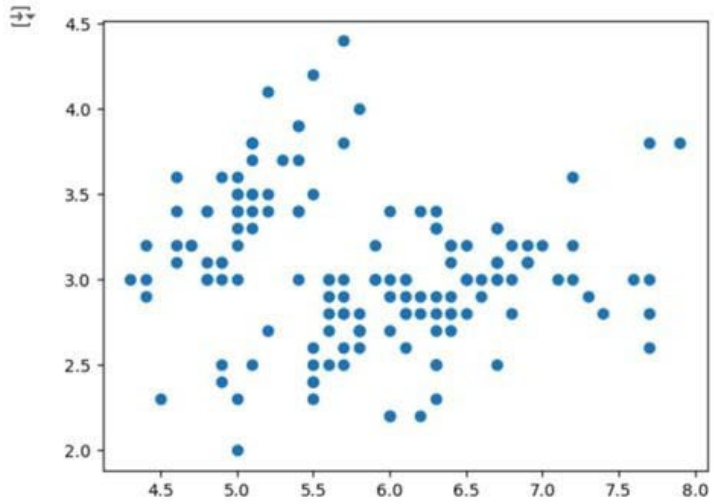
```
# Unsupervised learning
from sklearn import datasets
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt


iris_ds = datasets.load_iris()
x = iris_ds.data    # independent variable
x.shape
```

```
(150, 4)
```

```
# iris_ds


x[:1]     # value of first row
```

```
array([[5.1, 3.5, 1.4, 0.2]])
```

```
plt.scatter(x[:,0], x[:,1])    # (x[:,0], x[:,1]) = (x[all rows, 1st column], x[all rows, 2nd column])
plt.show()
```

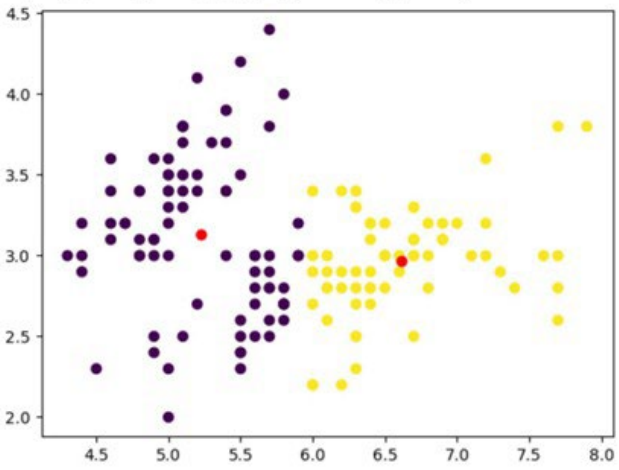



```
cluster_2_model = KMeans(n_clusters=2)
cluster_labels = cluster_2_model.fit_predict(x[:,[0,1]])

cluster_labels[:10], cluster_labels[-10:]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
(array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32),
 array([1, 1, 0, 1, 1, 1, 1, 1, 1, 0], dtype=int32))
```

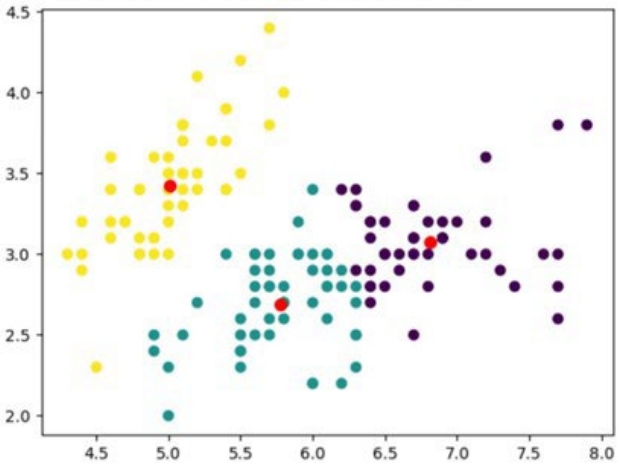
```
cluster_2_model = KMeans(n_clusters=2)
cluster_labels = cluster_2_model.fit_predict(x[:,[0,1]])
plt.scatter(x[:,0], x[:,1], c=cluster_labels)
centroids = cluster_2_model.cluster_centers_
plt.scatter(centroids[:,0], centroids[:,1], c='red')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
```

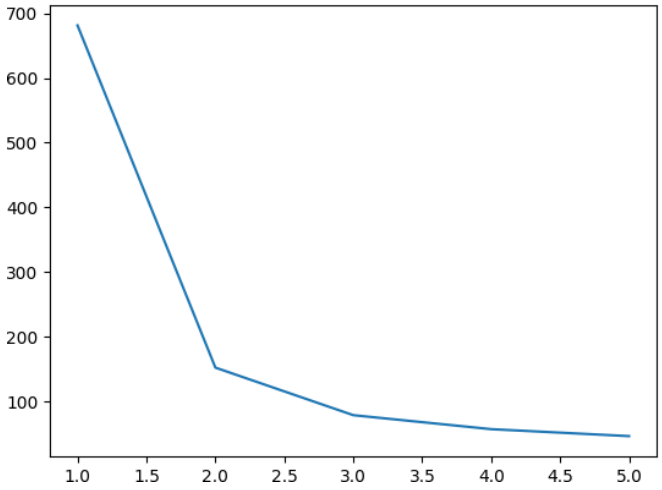



```
cluster_3_model = KMeans(n_clusters=3)
cluster_labels = cluster_3_model.fit_predict(x[:,[0,1]])
plt.scatter(x[:,0], x[:,1], c=cluster_labels)
centroids = cluster_3_model.cluster_centers_
plt.scatter(centroids[:,0], centroids[:,1], s=50, color='red')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
```



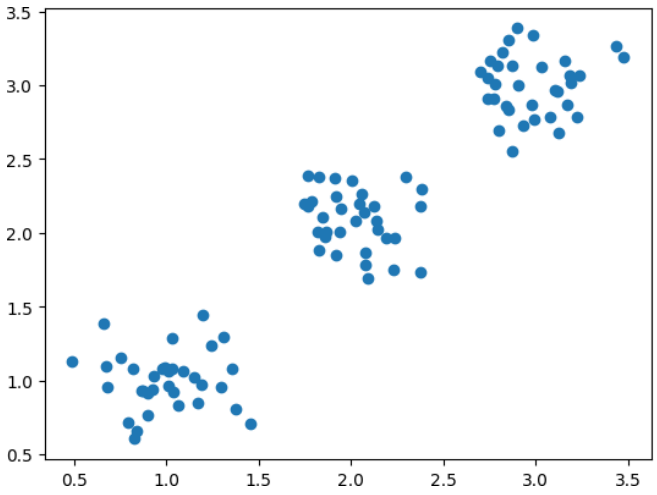

```
k_values = [1, 2, 3, 4, 5]
wcss_values = []

for k_value in k_values:
    model = KMeans(n_clusters=k_value)
    model.fit(x)
    wcss_value = model.inertia_
    wcss_values.append(wcss_value)

plt.plot(k_values, wcss_values)
plt.show()
```




```
from sklearn.metrics import silhouette_score

k_values = [2,3,4,5]
wcss_values = []

for k_value in k_values:
    model = KMeans(n_clusters=k_value)
    cluster_labels = model.fit_predict(x[:,[0, 1]])
    model_score = silhouette_score(x[:,[0,1]], cluster_labels)

    print(f"Cluster score: {model_score} for {k_value}")
```

```
Cluster score: 0.4629549773635977 for 2
Cluster score: 0.4450525692083638 for 3
Cluster score: 0.4206583419478651 for 4
Cluster score: 0.4050931000679723 for 5
```

```
x_data = datasets.make_blobs(n_samples = 100, centers=[[1, 1], [2, 2], [3, 3]], cluster_std = 0.2, random_state = 0)
# x_data[0].shape
plt.scatter(x_data[0][:,0], x_data[0][:,1])
plt.show()
```



Experiment - 9

Aim: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

Naïve Bayes:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import StandardScaler
import warnings

# Suppress warnings
warnings.filterwarnings('ignore')


patient_data = pd.read_csv('/content/heart_statlog_cleveland_hungary_final.csv')


patient_data.shape
```

```
(1190, 12)
```

```
patient_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1190 entries, 0 to 1189
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  1190 non-null   int64  
 1   sex                  1190 non-null   int64  
 2   chest pain type      1190 non-null   int64  
 3   resting bp s         1190 non-null   int64  
 4   cholesterol          1190 non-null   int64  
 5   fasting blood sugar  1190 non-null   int64  
 6   resting ecg          1190 non-null   int64  
 7   max heart rate       1190 non-null   int64  
 8   exercise angina      1190 non-null   int64  
 9   oldpeak              1190 non-null   float64
 10  ST slope             1190 non-null   int64  
 11  target               1190 non-null   int64  
dtypes: float64(1), int64(11)
memory usage: 111.7 KB
```

```
patient_data.head()
```

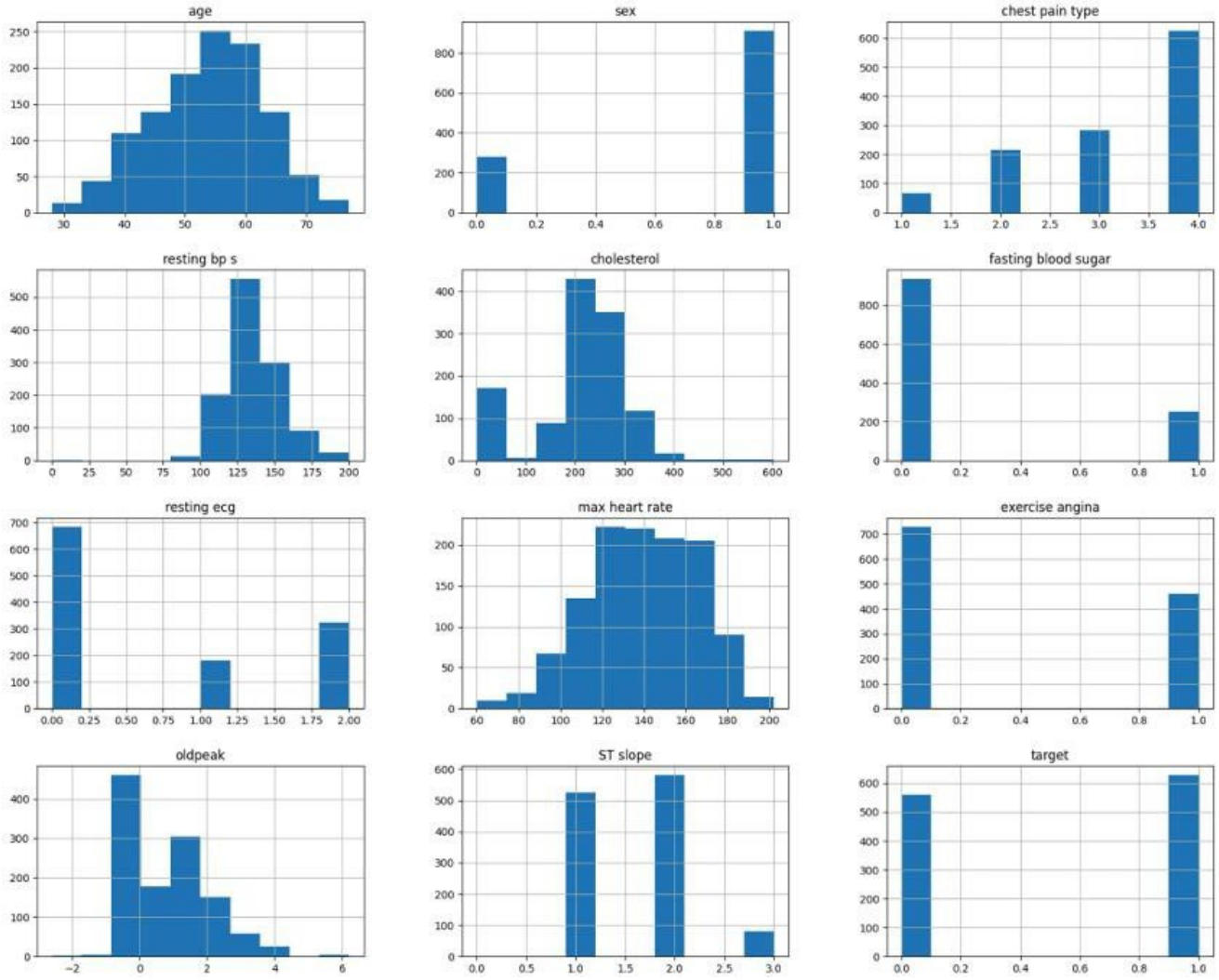
	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope	target
0	40	1	2	140	289	0	0	172	0	0.0	1	0
1	49	0	3	160	180	0	0	156	0	1.0	2	1
2	37	1	2	130	283	0	1	98	0	0.0	1	0
3	48	0	4	138	214	0	0	108	1	1.5	2	1

```
patient_data.describe()
```

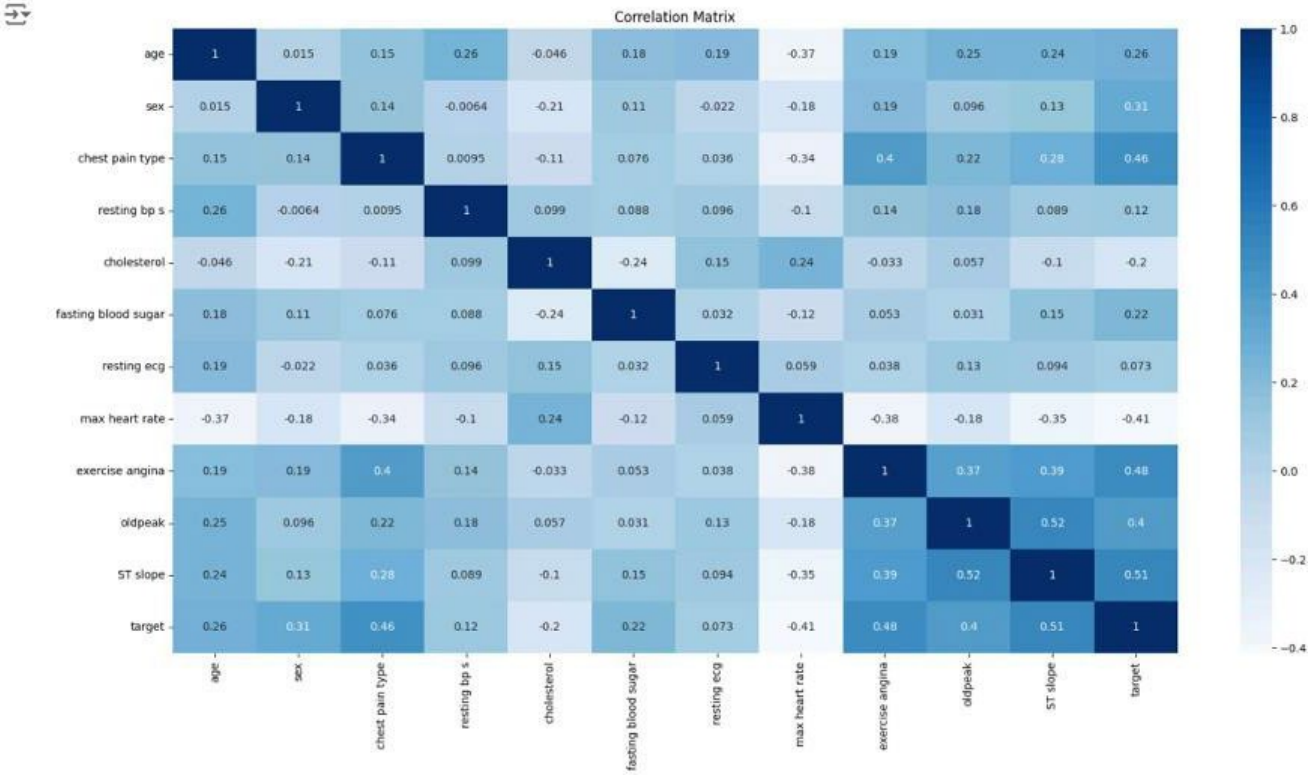
	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak
count	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000
mean	53.720168	0.763866	3.232773	132.153782	210.363866	0.213445	0.698319	139.732773	0.387395	0.922773
std	9.358203	0.424884	0.935480	18.368823	101.420489	0.409912	0.870359	25.517636	0.487360	1.086337
min	28.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	60.000000	0.000000	-2.600000
25%	47.000000	1.000000	3.000000	120.000000	188.000000	0.000000	0.000000	121.000000	0.000000	0.000000
50%	54.000000	1.000000	4.000000	130.000000	229.000000	0.000000	0.000000	140.500000	0.000000	0.600000
75%	60.000000	1.000000	4.000000	140.000000	269.750000	0.000000	2.000000	160.000000	1.000000	1.600000

```
patient_data.hist(figsize=(20, 16))
```

```
array([[<Axes: title={'center': 'age'}>, <Axes: title={'center': 'sex'}>,
        <Axes: title={'center': 'chest pain type'}>],
       [<Axes: title={'center': 'resting bp s'}>,
        <Axes: title={'center': 'cholesterol'}>,
        <Axes: title={'center': 'fasting blood sugar'}>],
       [<Axes: title={'center': 'resting ecg'}>,
        <Axes: title={'center': 'max heart rate'}>,
        <Axes: title={'center': 'exercise angina'}>],
       [<Axes: title={'center': 'oldpeak'}>,
        <Axes: title={'center': 'ST slope'}>,
        <Axes: title={'center': 'target'}>]], dtype=object)
```



```
# Heatmap
corr_matrix = patient_data.corr()

plt.figure(figsize=(20, 10))
sns.heatmap(corr_matrix, annot=True, cmap="Blues")
plt.title("Correlation Matrix")
plt.show()
```




```
# Check for null values
patient_data.isnull().sum()
```

	0
age	0
sex	0
chest pain type	0
resting bp s	0
cholesterol	0
fasting blood sugar	0
resting ecg	0
max heart rate	0
exercise angina	0
oldpeak	0
ST slope	0
target	0

```
patient_data.duplicated().sum()
```

```
272
```

```
patient_data["target"].value_counts()
```

target	count
1	629
0	561

```
# Set target
X_normal = patient_data.drop("target", axis=1)
Y = patient_data["target"]


# Scale data
Scaler = StandardScaler()
X = Scaler.fit_transform(X_normal)
X = pd.DataFrame(X, columns=X_normal.columns)
X.head()
```

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope
0	-1.466728	0.555995	-1.318351	0.427328	0.775674	-0.520929	-0.802672	1.265039	-0.795219	-0.849792	-1.023217
1	-0.504600	-1.798576	-0.248932	1.516587	-0.299512	-0.520929	-0.802672	0.637758	-0.795219	0.071119	0.615583
2	-1.787437	0.555995	-1.318351	-0.117301	0.716489	-0.520929	0.346762	-1.636136	-0.795219	-0.849792	-1.023217
3	-0.611503	-1.798576	0.820487	0.318402	0.035867	-0.520929	-0.802672	-1.244085	1.257515	0.531575	0.615583

Next steps: Generate code with X | View recommended plots | New interactive sheet

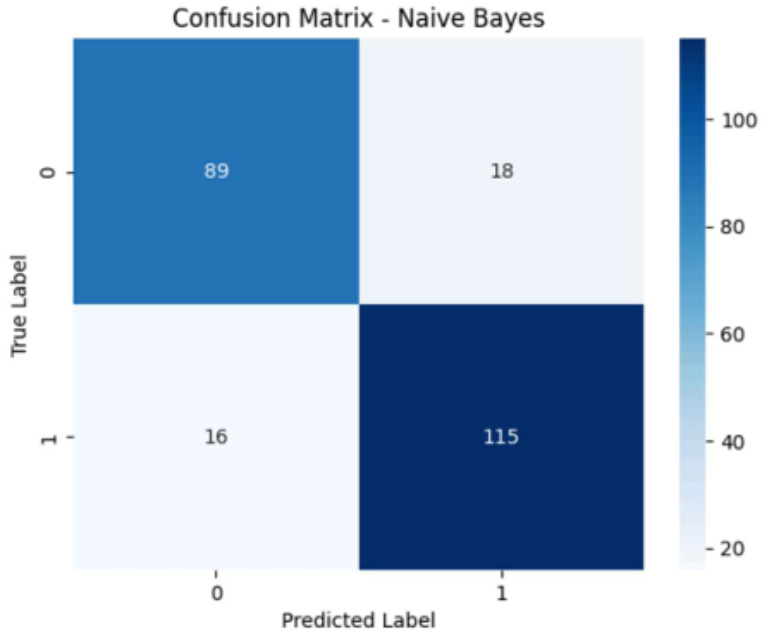
```
Y.value_counts()
```

target	count
1	629
0	561

```
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42
)


# Train and Evaluate model using Naive Bayes model
NB = GaussianNB()

NB.fit(X_train, Y_train)

NB_predict = NB.predict(X_test)

NB_acc_score = accuracy_score(Y_test, NB_predict)

print("Accuracy of Naive Bayes model:", "{:.2f}%".format(NB_acc_score * 100))

print("\nClassification Report:")
nb_cr = classification_report(Y_test, NB_predict)
print(nb_cr)
```

```
NB_cm = confusion_matrix(Y_test, NB_predict)

sns.heatmap(NB_cm, annot=True, fmt="d", cmap="Blues")

plt.title("Confusion Matrix - Naive Bayes")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

plt.show()
```

```
Accuracy of Naive Bayes model: 85.71%

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.83      0.84       107
           1       0.86      0.88      0.87       131

    accuracy                           0.86       238
   macro avg       0.86      0.85      0.86       238
weighted avg       0.86      0.86      0.86       238
```



Experiment – 10

Aim: Compare the various Supervised learning algorithm by using appropriate dataset.

Comparison of various Supervised Learning Algorithms:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import StandardScaler

import warnings

# Suppress warnings
warnings.filterwarnings("ignore")


patient_data = pd.read_csv("/content/heart_statlog_cleveland_hungary_final.csv")


patient_data.shape
```

```
(1190, 12)
```

```
patient_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1190 entries, 0 to 1189
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  1190 non-null   int64  
 1   sex                  1190 non-null   int64  
 2   chest pain type      1190 non-null   int64  
 3   resting bp s         1190 non-null   int64  
 4   cholesterol          1190 non-null   int64  
 5   fasting blood sugar  1190 non-null   int64  
 6   resting ecg          1190 non-null   int64  
 7   max heart rate       1190 non-null   int64  
 8   exercise angina      1190 non-null   int64  
 9   oldpeak              1190 non-null   float64
 10  ST slope             1190 non-null   int64  
 11  target               1190 non-null   int64  
dtypes: float64(1), int64(11)
memory usage: 111.7 KB
```

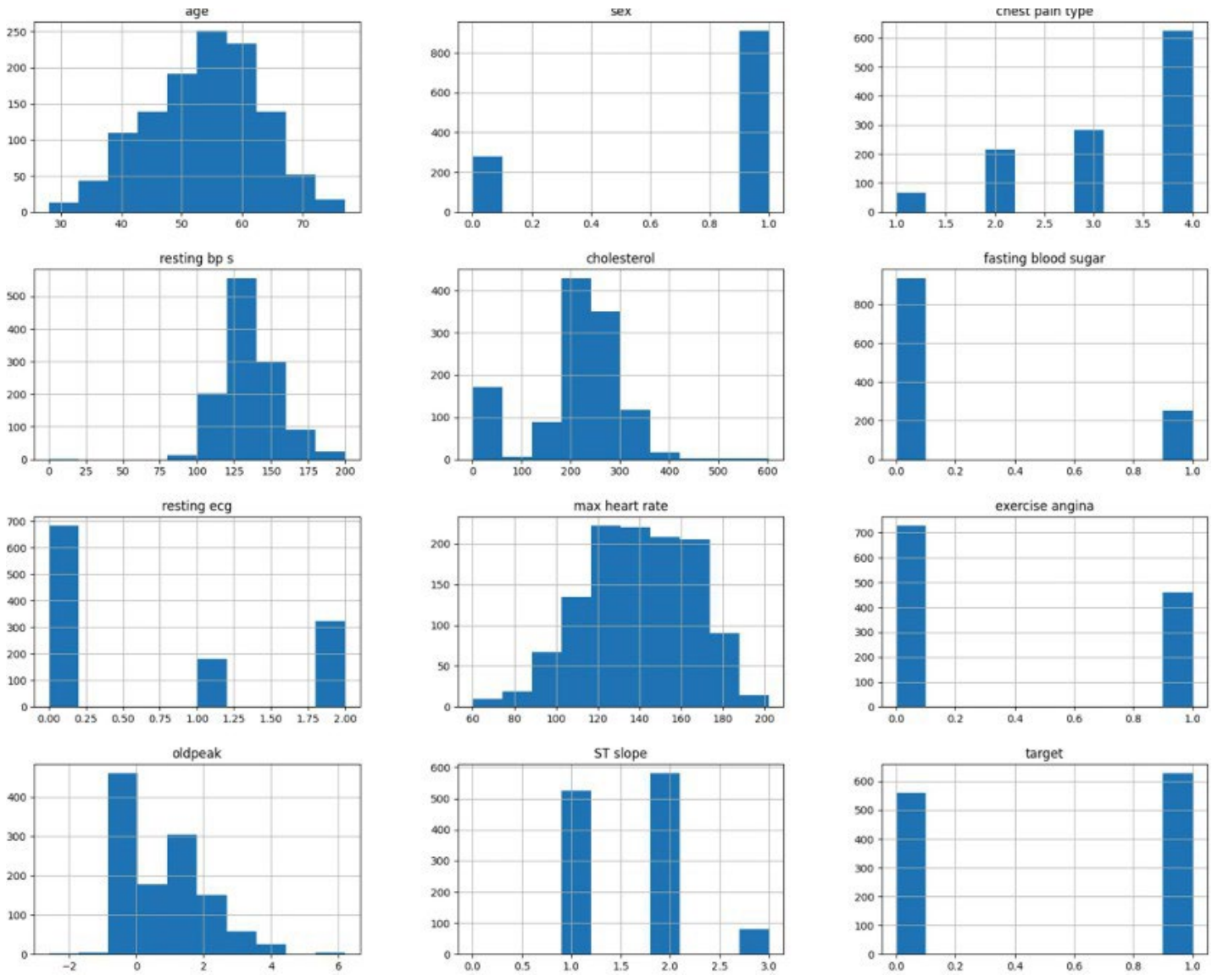
```
patient_data.head()
```

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope	target
0	40	1	2	140	289	0	0	172	0	0.0	1	0
1	49	0	3	160	180	0	0	156	0	1.0	2	1
2	37	1	2	130	283	0	1	98	0	0.0	1	0
3	48	0	4	138	214	0	0	108	1	1.5	2	1

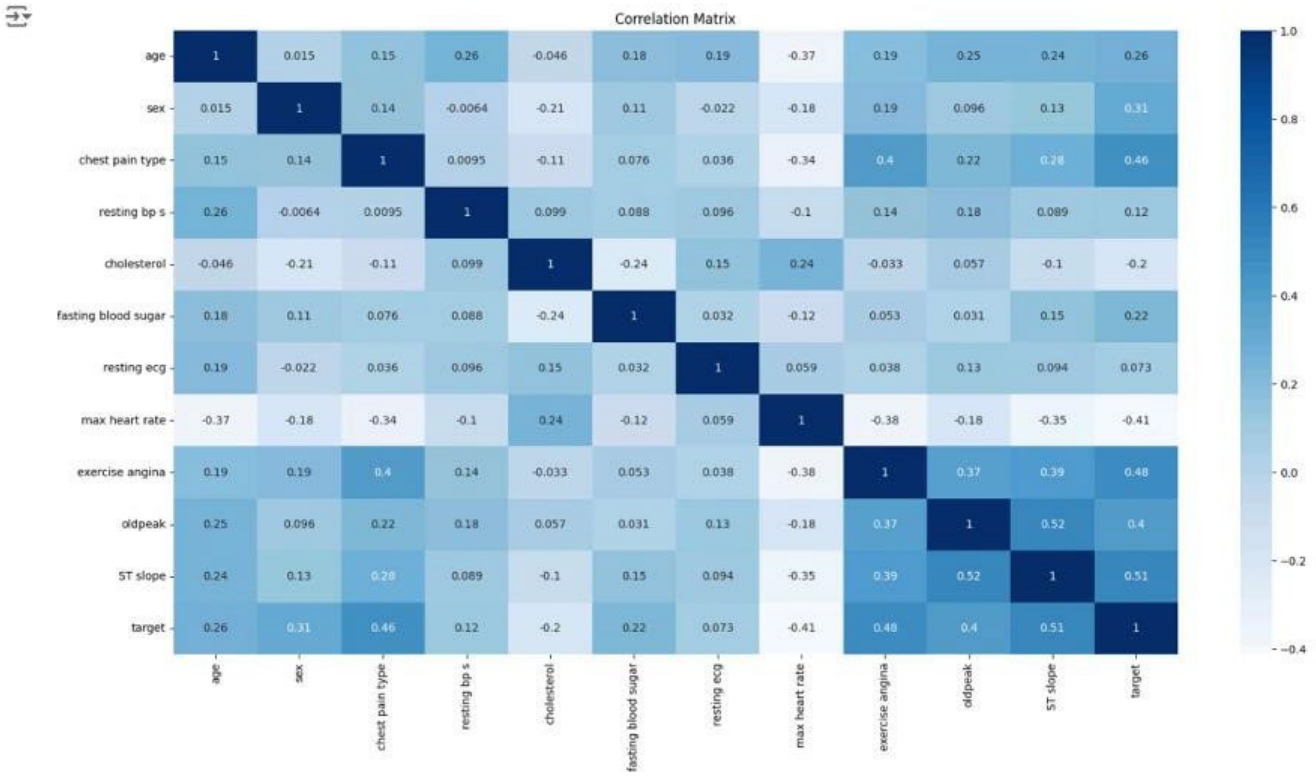
```
patient_data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
age	1190.0	53.720168	9.358203	28.0	47.0	54.0	60.00	77.0
sex	1190.0	0.763866	0.424884	0.0	1.0	1.0	1.00	1.0
chest pain type	1190.0	3.232773	0.935480	1.0	3.0	4.0	4.00	4.0
resting bp s	1190.0	132.153782	18.368823	0.0	120.0	130.0	140.00	200.0
cholesterol	1190.0	210.363866	101.420489	0.0	188.0	229.0	269.75	603.0
fasting blood sugar	1190.0	0.213445	0.409912	0.0	0.0	0.0	0.00	1.0
resting ecg	1190.0	0.698319	0.870359	0.0	0.0	0.0	2.00	2.0
max heart rate	1190.0	139.732773	25.517636	60.0	121.0	140.5	160.00	202.0
exercise angina	1190.0	0.387395	0.487360	0.0	0.0	0.0	1.00	1.0
oldpeak	1190.0	0.922773	1.086337	-2.6	0.0	0.6	1.60	6.2
ST slope	1190.0	1.624370	0.610459	0.0	1.0	2.0	2.00	3.0
target	1190.0	0.528571	0.499393	0.0	0.0	1.0	1.00	1.0

```
patient_data.hist(figsize=(20, 16))
```



```
corr_matrix = patient_data.corr()

plt.figure(figsize=(20, 10))
sns.heatmap(corr_matrix, annot=True, cmap="Blues")
plt.title("Correlation Matrix")
plt.show()
```




```
patient_data.isnull().sum()
```

	0
age	0
sex	0
chest pain type	0
resting bp s	0
cholesterol	0
fasting blood sugar	0
resting ecg	0
max heart rate	0
exercise angina	0
oldpeak	0
ST slope	0
target	0

```
patient_data.duplicated().sum()
```

```
272
```

```
patient_data["target"].value_counts()
```

target	count
1	629
0	561

```
X_normal = patient_data.drop("target", axis=1)
Y = patient_data["target"]


Scaler = StandardScaler()
X = Scaler.fit_transform(X_normal)
X = pd.DataFrame(X, columns=X_normal.columns)
X.head()
```

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope
0	-1.466728	0.555995	-1.318351	0.427328	0.775674	-0.520929	-0.802672	1.265039	-0.795219	-0.849792	-1.023217
1	-0.504600	-1.798576	-0.248932	1.516587	-0.299512	-0.520929	-0.802672	0.637758	-0.795219	0.071119	0.615583
2	-1.787437	0.555995	-1.318351	-0.117301	0.716489	-0.520929	0.346762	-1.636136	-0.795219	-0.849792	-1.023217
3	-0.611503	-1.798576	0.820487	0.318402	0.035867	-0.520929	-0.802672	-1.244085	1.257515	0.531575	0.615583

Next steps: Generate code with X | View recommended plots | New interactive sheet

```
Y.value_counts()
```

target	count
1	629
0	561

```
# 80% of the data will be used for training
# 20% of the data will be used for testing

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42
)
```

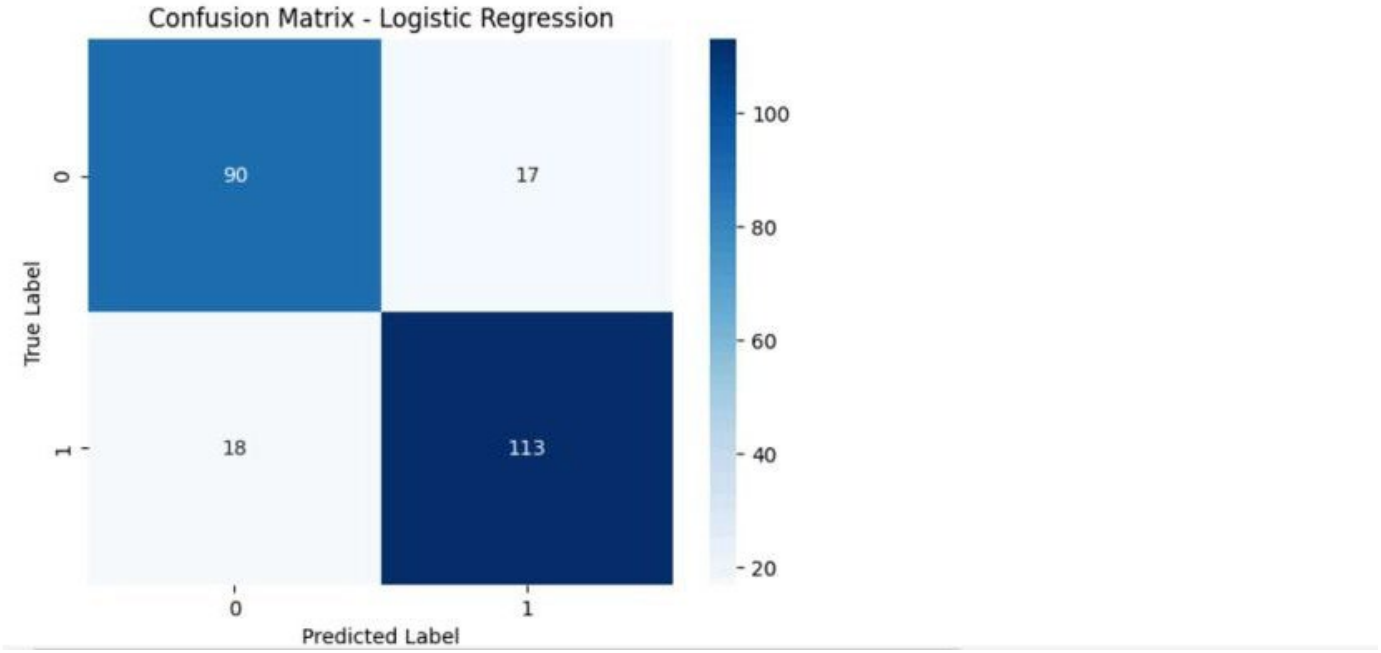
Model training and Evalution using different Supervised Learning Algorithms

1. Logistic Regression

```
# Define the parameter grid for Logistic Regression
logreg_param_grid = {
    "penalty": ["l1", "l2"],
    "C": np.logspace(-3, 3, 7),
    "solver": ["liblinear"],
}
```

```
# Create a Logistic Regression model
logreg_model = LogisticRegression(random_state=42)

# Perform grid search with cross-validation
logreg_grid_search = GridSearchCV(
    logreg_model, logreg_param_grid, cv=5, scoring="accuracy"
)
logreg_grid_search.fit(X_train, Y_train)

# Get the best parameters
best_logreg_params = logreg_grid_search.best_params_
print("Best Hyperparameters for Logistic Regression:", best_logreg_params)

# Train a Logistic Regression model with the best parameters
best_logreg_model = LogisticRegression(random_state=42, **best_logreg_params)
best_logreg_model.fit(X_train, Y_train)

# Make predictions on the test set
logreg_predict = best_logreg_model.predict(X_test)

# Calculate accuracy on the test set
best_logreg_acc = accuracy_score(Y_test, logreg_predict)
print(
    "Best Accuracy of Logistic Regression model:",
    "{:.2f}%".format(best_logreg_acc * 100),
)

# Display classification report
print("\nClassification Report - Logistic Regression:")
lr_cr = classification_report(Y_test, logreg_predict)
print(lr_cr)

# Display confusion matrix
logreg_cm = confusion_matrix(Y_test, logreg_predict)

# Plot the confusion matrix
sns.heatmap(logreg_cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix - Logistic Regression")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Best Hyperparameters for Logistic Regression: {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
Best Accuracy of Logistic Regression model: 85.29%

Classification Report - Logistic Regression:
              precision    recall  f1-score   support

           0       0.83      0.84      0.84       107
           1       0.87      0.86      0.87       131

    accuracy                           0.85       238
   macro avg       0.85      0.85      0.85       238
weighted avg       0.85      0.85      0.85       238
```

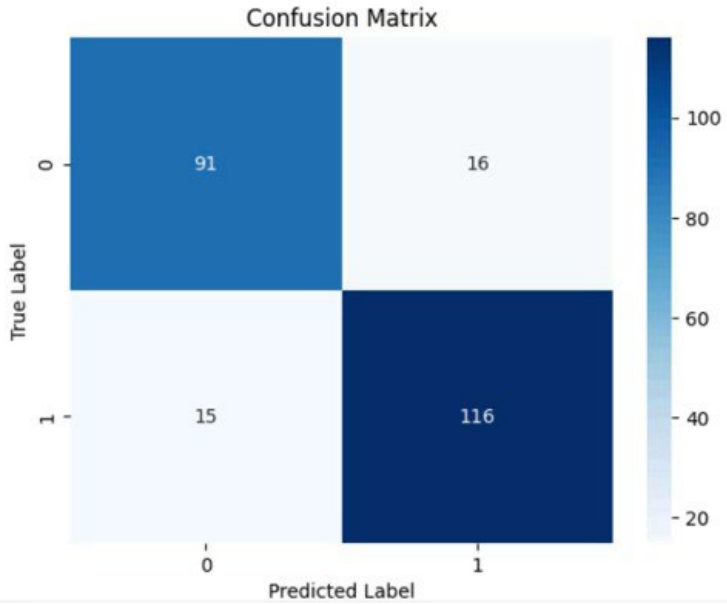


2. Decision Tree

```
# Define the parameter grid
param_grid = {
    "criterion": ["gini", "entropy"],
    "max_depth": np.arange(1, 21),
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "max_features": ["sqrt", "log2", None],
}

# Create a Decision Tree model
DT = DecisionTreeClassifier(random_state=0)

# Perform grid search with cross-validation
grid_search = GridSearchCV(DT, param_grid, cv=5, scoring="accuracy")
grid_search.fit(X_train, Y_train)

best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Train a Decision Tree with the best parameters
best_DT = DecisionTreeClassifier(random_state=0, **best_params)
best_DT.fit(X_train, Y_train)

# Make predictions on the test set
DT_predict = best_DT.predict(X_test)

# Calculate accuracy on the test set
max_dt_acc = accuracy_score(Y_test, DT_predict)
print(
    "Accuracy of Decision Tree with Best Parameters:",
    "{:.2f}%".format(max_dt_acc * 100),
)
```

```
print("\nClassification Report:")
dt_cr = classification_report(Y_test, DT_predict)
print(dt_cr)

# Display confusion matrix
DT_cm = confusion_matrix(Y_test, DT_predict)

# Plot the confusion matrix
sns.heatmap(DT_cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Best Hyperparameters: {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split':
Accuracy of Decision Tree with Best Parameters: 86.97%

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.85      0.85       107
           1       0.88      0.89      0.88       131

    accuracy                           0.87       238
   macro avg       0.87      0.87      0.87       238
weighted avg       0.87      0.87      0.87       238
```

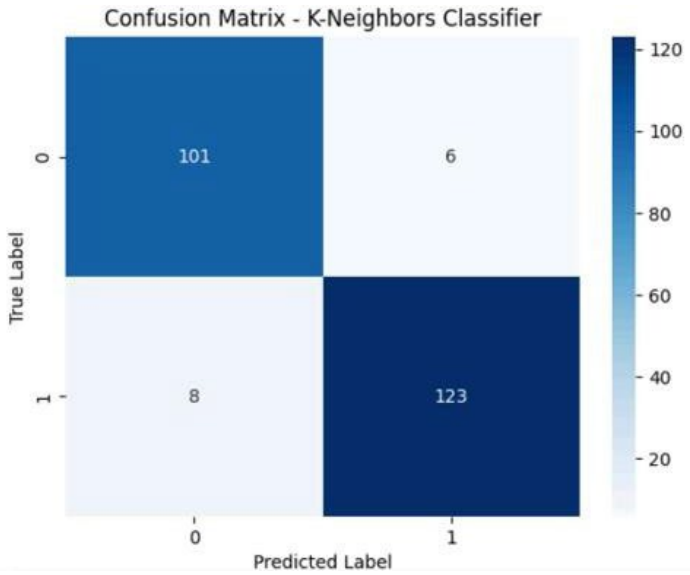


3. KNN

```
  # Define the parameter grid for K-Nearest Neighbors
  knn_param_grid = {
      "n_neighbors": np.arange(1, 21),
      "weights": ["uniform", "distance"],
      "algorithm": ["auto", "ball_tree", "kd_tree", "brute"],
  }
# Create a K-Nearest Neighbors model
KNN = KNeighborsClassifier()

# Perform grid search with cross-validation
knn_grid_search = GridSearchCV(KNN, knn_param_grid, cv=5, scoring="accuracy")
knn_grid_search.fit(X_train, Y_train)
```

```
best_knn_params = knn_grid_search.best_params_
print("Best Hyperparameters for K-Nearest Neighbors:", best_knn_params)

# Train a K-Nearest Neighbors model with the best parameters
best_KNN_model = KNeighborsClassifier(**best_knn_params)
best_KNN_model.fit(X_train, Y_train)

# Make predictions on the test set
KNN_predict = best_KNN_model.predict(X_test)

# Calculate accuracy on the test set
best_KNN_acc = accuracy_score(Y_test, KNN_predict)
print("Best Accuracy of K-Neighbors Classifier:", "{:.2f}%".format(best_KNN_acc * 100))

# Display classification report
print("\nClassification Report - K-Neighbors Classifier:")
knn_cr = classification_report(Y_test, KNN_predict)
print(knn_cr)

# Display confusion matrix
KNN_cm = confusion_matrix(Y_test, KNN_predict)

# Plot the confusion matrix
sns.heatmap(KNN_cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix - K-Neighbors Classifier")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

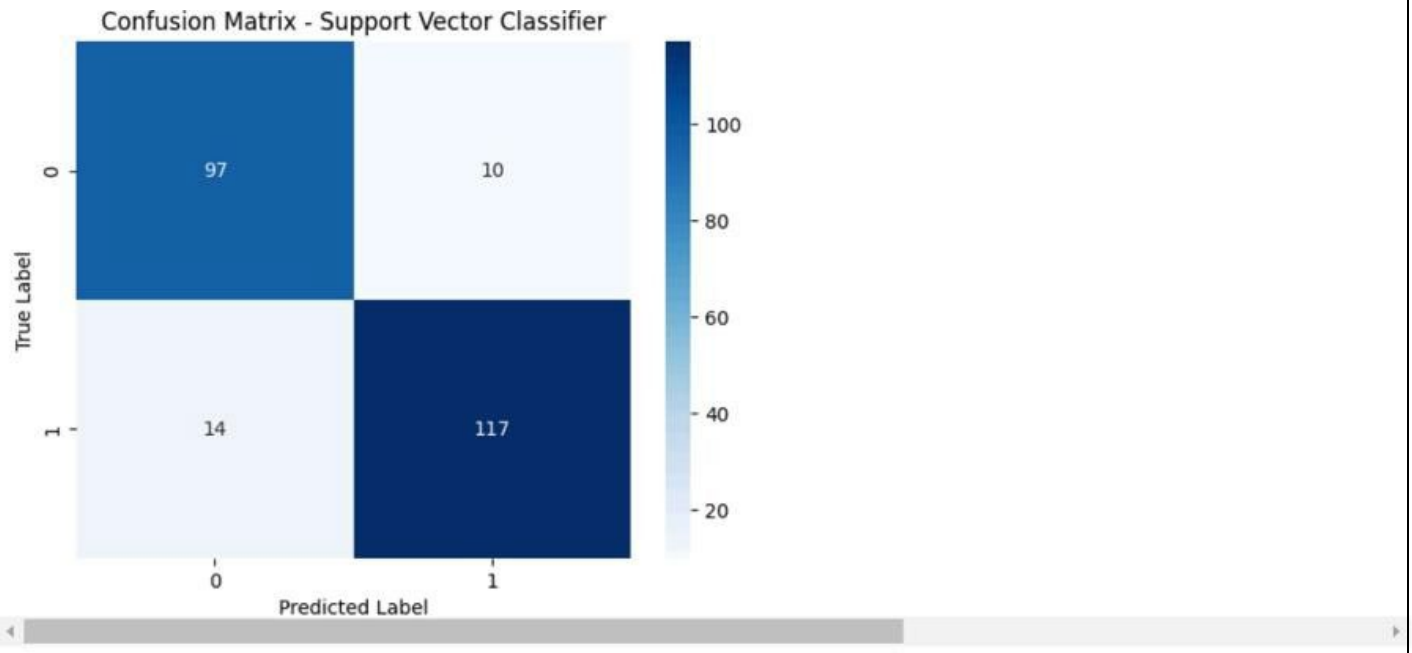
```
Best Hyperparameters for K-Nearest Neighbors: {'algorithm': 'auto', 'n_neighbors': 19, 'weights': 'distance'}
Best Accuracy of K-Neighbors Classifier: 94.12%

Classification Report - K-Neighbors Classifier:
              precision    recall  f1-score   support

           0       0.93      0.94      0.94       107
           1       0.95      0.94      0.95       131

    accuracy                           0.94       238
   macro avg       0.94      0.94      0.94       238
weighted avg       0.94      0.94      0.94       238
```



4. SVM

```
# Define the parameter grid for Support Vector Classifier
svm_param_grid = {
    "C": [0.1, 1, 10, 100],
    "gamma": ["scale", "auto", 0.001, 0.01, 0.1, 1, 10],
    "kernel": ["rbf"],
}

# Create a Support Vector Classifier model
SVM = SVC()

# Perform grid search with cross-validation
svm_grid_search = GridSearchCV(SVM, svm_param_grid, cv=5, scoring="accuracy")
svm_grid_search.fit(X_train, Y_train)

# Get the best parameters
best_svm_params = svm_grid_search.best_params_
print("Best Hyperparameters for Support Vector Classifier:", best_svm_params)

# Train a Support Vector Classifier model with the best parameters
best_SVM_model = SVC(**best_svm_params)
best_SVM_model.fit(X_train, Y_train)

# Make predictions on the test set
SVM_predict = best_SVM_model.predict(X_test)

# Calculate accuracy on the test set
best_SVM_acc = accuracy_score(Y_test, SVM_predict)
print(
    "Best Accuracy of Support Vector Classifier:", "{:.2f}%".format(best_SVM_acc * 100)
)

# Display classification report
print("\nClassification Report - Support Vector Classifier:")
svm_cr = classification_report(Y_test, SVM_predict)
print(svm_cr)

# Display confusion matrix
SVM_cm = confusion_matrix(Y_test, SVM_predict)

# Plot the confusion matrix
sns.heatmap(SVM_cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix - Support Vector Classifier")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

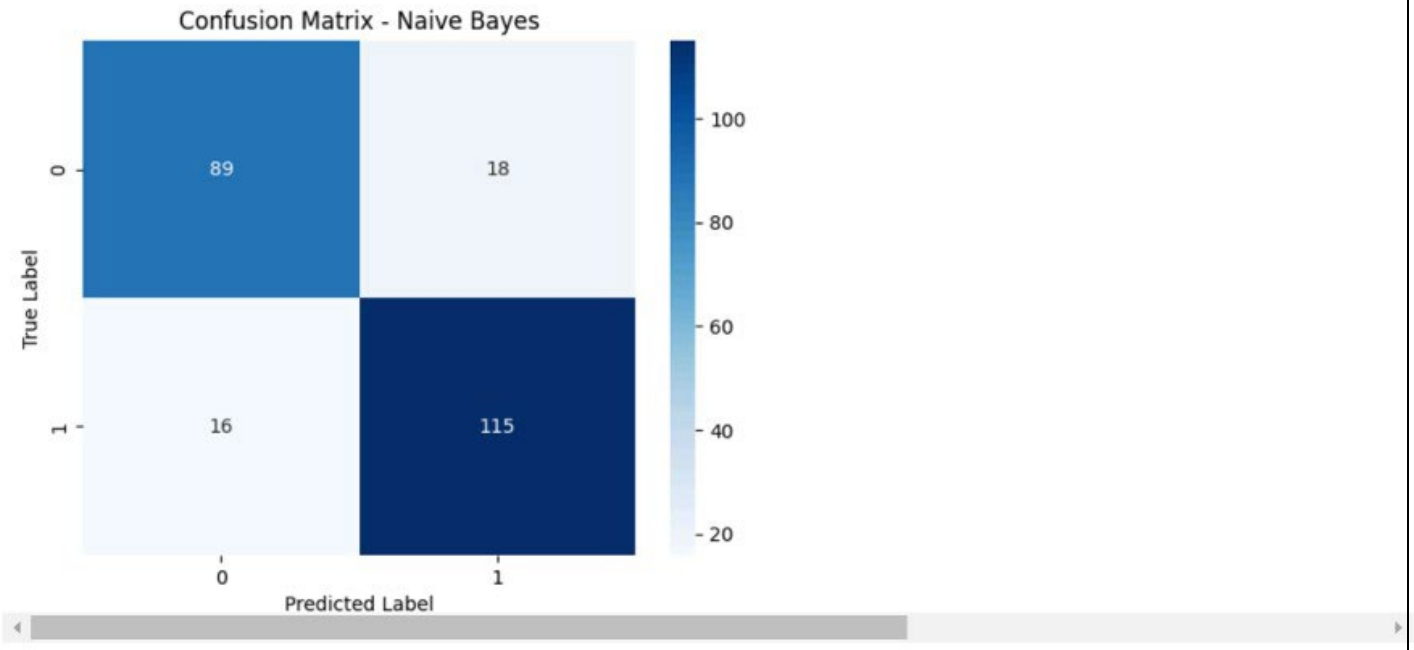
```
Best Hyperparameters for Support Vector Classifier: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Best Accuracy of Support Vector Classifier: 89.92%

Classification Report - Support Vector Classifier:
              precision    recall  f1-score   support

           0       0.87      0.91      0.89       107
           1       0.92      0.89      0.91       131

    accuracy                           0.90       238
   macro avg       0.90      0.90      0.90       238
weighted avg       0.90      0.90      0.90       238
```



5. Naive Bayes

```
NB = GaussianNB()

NB.fit(X_train, Y_train)

NB_predict = NB.predict(X_test)

NB_acc_score = accuracy_score(Y_test, NB_predict)

print("Accuracy of Naive Bayes model:", "{:.2f}%".format(NB_acc_score * 100))

print("\nClassification Report:")
nb_cr = classification_report(Y_test, NB_predict)
print(nb_cr)

NB_cm = confusion_matrix(Y_test, NB_predict)

sns.heatmap(NB_cm, annot=True, fmt="d", cmap="Blues")

plt.title("Confusion Matrix - Naive Bayes")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

plt.show()
```

```
Accuracy of Naive Bayes model: 85.71%

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.83      0.84       107
           1       0.86      0.88      0.87       131

    accuracy                           0.86       238
   macro avg       0.86      0.85      0.86       238
weighted avg       0.86      0.86      0.86       238
```



Comparison

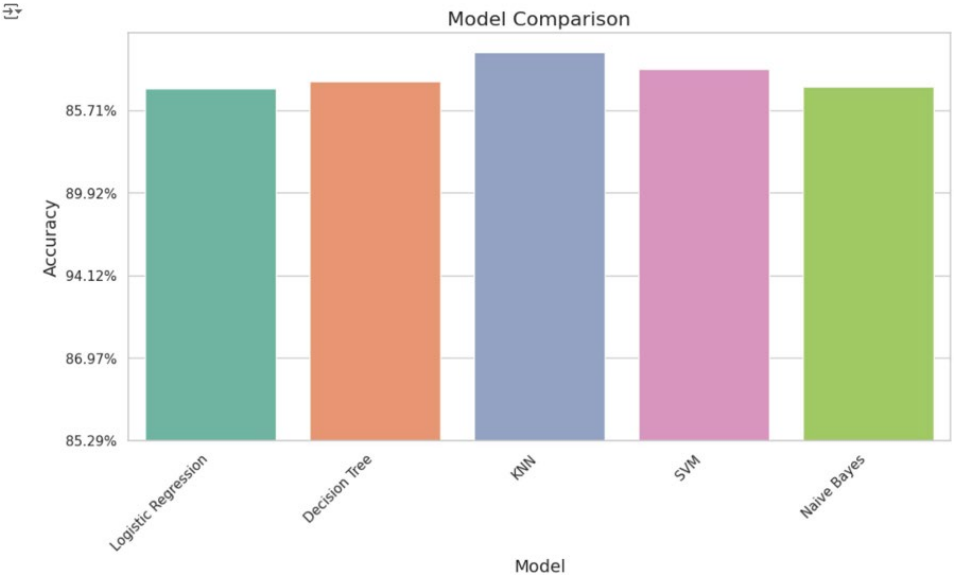
1. Comparison Table

```
# Create a dictionary to store the models and their accuracies
models_accuracy = {
    "Logistic Regression": best_logreg_acc,
    "Decision Tree": max_dt_acc,
    "KNN": best_KNN_acc,
    "SVM": best_SVM_acc,
    "Naive Bayes": NB_acc_score,
}

# Find the model with the highest accuracy
best_model = max(models_accuracy, key=models_accuracy.get)
best_accuracy = models_accuracy[best_model]

comparison = pd.DataFrame(
    {
        "Model": list(models_accuracy.keys()),
        "Accuracy": ["{:.2f}%".format(acc * 100) for acc in models_accuracy.values()],
    }
)

print("Comparison Table:")
print(comparison)

# Print the name and accuracy of the best model
print("\nBest Model:")
print(f"{best_model}: {best_accuracy:.2%}")
```

```
Comparison Table:
                 Model Accuracy
0  Logistic Regression   85.29%
1        Decision Tree   86.97%
2                  KNN   94.12%
3                  SVM   89.92%
4          Naive Bayes   85.71%

Best Model:
KNN: 94.12%
```

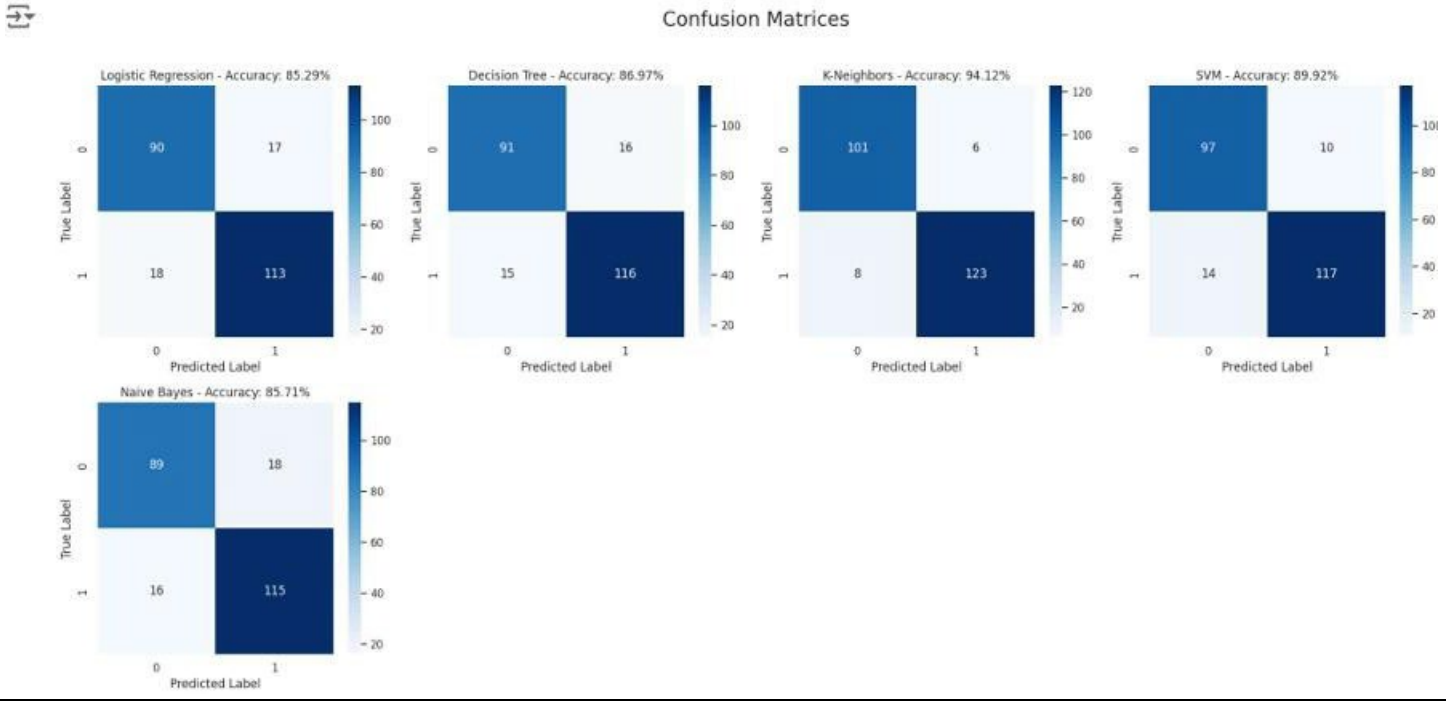
2. Comparison using Bar Plot

```
# Multiply values by 100 and format to 2 decimal places
y_values = [value * 100 for value in models_accuracy.values()]
y_labels = ["{:.2f}%".format(value) for value in y_values]

# Create a color palette with sufficient contrast
colors = sns.color_palette("Set2", len(models_accuracy))

# Create a bar plot
plt.figure(figsize=(12, 6))
sns.set(style="whitegrid")
ax = sns.barplot(x=list(models_accuracy.keys()), y=y_values, palette=colors)

# Set y-axis labels
ax.set_yticklabels(y_labels)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha="right")

plt.title("Model Comparison", fontsize=16)
plt.xlabel("Model", fontsize=14)
plt.ylabel("Accuracy", fontsize=14)

plt.show()
```



3. Comparing Confusion Matrices

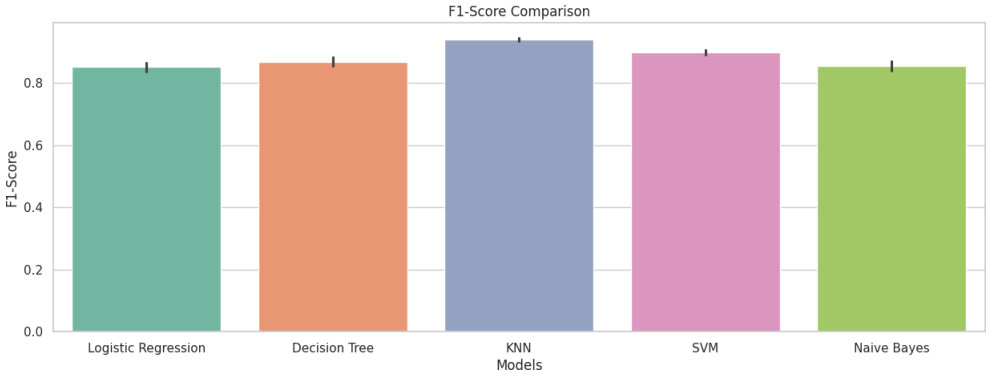
```
num_classifiers = 5
num_rows = (num_classifiers - 1) // 4 + 1
num_cols = min(num_classifiers, 4)

fig, axes = plt.subplots(
    nrows=num_rows, ncols=num_cols, figsize=(5 * num_cols, 5 * num_rows)
)
fig.suptitle("Confusion Matrices", fontsize=20)

classifiers = [
    ("Logistic Regression", logreg_cm, best_logreg_acc),
    ("Decision Tree", DT_cm, max_dt_acc),
    ("K-Neighbors", KNN_cm, best_KNN_acc),
    ("SVM", SVM_cm, best_SVM_acc),
    ("Naive Bayes", NB_cm, NB_acc_score),
]

for (name, cm, acc_score), ax in zip(classifiers, axes.flatten()):
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", ax=ax)

    ax.set_title(f"{name} - Accuracy: {acc_score * 100:.2f}%")
    ax.set_xlabel("Predicted Label")
    ax.set_ylabel("True Label")

for i in range(num_classifiers, num_rows * num_cols):
    fig.delaxes(axes.flatten()[i])

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

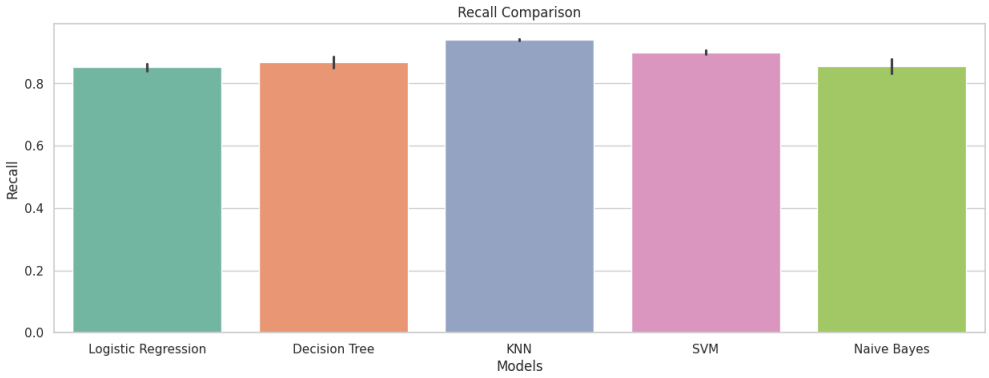


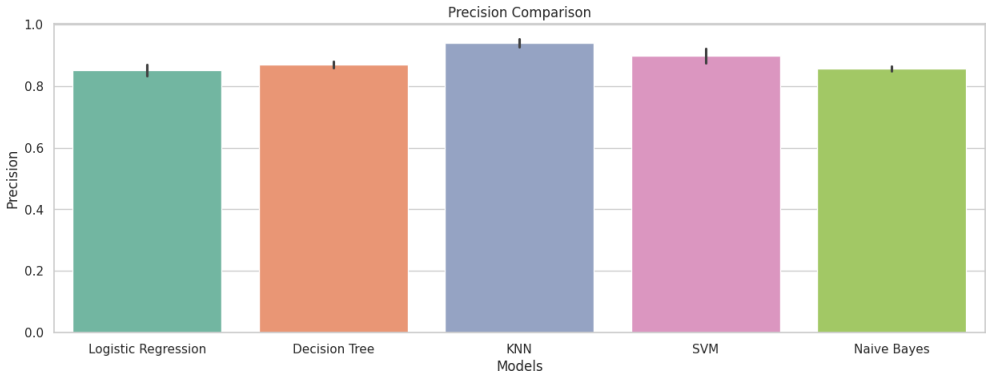
4. Classification Report Comparision Bar Plots

```
# Get classification reports as dictionaries
lr_cr = classification_report(Y_test, logreg_predict, output_dict=True)
dt_cr = classification_report(Y_test, DT_predict, output_dict=True)
knn_cr = classification_report(Y_test, KNN_predict, output_dict=True)
svm_cr = classification_report(Y_test, SVM_predict, output_dict=True)
nb_cr = classification_report(Y_test, NB_predict, output_dict=True)

classification_reports = [lr_cr, dt_cr, knn_cr, svm_cr, nb_cr]

f1_scores = {}
recall_scores = {}
precision_scores = {}

# Store f1-score, recall, and precision scores in lists
for name, cr in zip(
    [
        "Logistic Regression",
        "Decision Tree",
        "KNN",
        "SVM",
        "Naive Bayes",
    ],
    classification_reports,
):
    f1_scores[name] = [
        cr[label]["f1-score"] for label in cr.keys() if label.isnumeric()
    ]
    recall_scores[name] = [
        cr[label]["recall"] for label in cr.keys() if label.isnumeric()
    ]
    precision_scores[name] = [
        cr[label]["precision"] for label in cr.keys() if label.isnumeric()
    ]

# Create pandas dataframes from the lists
df_f1 = pd.DataFrame(
    f1_scores, index=[str(i) for i in range(1, len(f1_scores["Decision Tree"]) + 1)]
)
df_recall = pd.DataFrame(
    recall_scores,
    index=[str(i) for i in range(1, len(recall_scores["Decision Tree"]) + 1)],
)
df_precision = pd.DataFrame(
    precision_scores,
    index=[str(i) for i in range(1, len(precision_scores["Decision Tree"]) + 1)],
)

# Plot accuracy comparison
plt.figure(figsize=(15, 5))
sns.barplot(data=df_f1, palette="Set2")
plt.title("F1-Score Comparison")
plt.xlabel("Models")
plt.ylabel("F1-Score")
plt.show()
```

```
# Plot recall comparison
plt.figure(figsize=(15, 5))
sns.barplot(data=df_recall, palette="Set2")
plt.title("Recall Comparison")
plt.xlabel("Models")
plt.ylabel("Recall")
plt.show()

# Plot precision comparison
plt.figure(figsize=(15, 5))
sns.barplot(data=df_precision, palette="Set2")
plt.title("Precision Comparison")
plt.xlabel("Models")
plt.ylabel("Precision")
plt.show()
```







Conclusion

By comparing various Supervised Learning Algorithm on Heart Disease Dataset. We can say that KNN is the best Supervised Learning Algorithm for these dataset.

Experiment - 11

Aim: Compare the various Unsupervised learning algorithm by using the appropriate datasets.

Comparison of various Unsupervised Learning Algorithms:

```
from sklearn import datasets
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
# Load the Dataset
ds = datasets.load_wine()
x = ds.data    # independent variable
x.shape
```

```
(178, 13)
```

```
import pandas as pd
x_df = pd.DataFrame(x, columns=ds.feature_names)
x_df.head()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_i
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	

Next steps: Generate code with x_df | View recommended plots | New interactive sheet

```
x_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
dtypes: float64(13)
memory usage: 18.2 KB
```

```
x_df.describe()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocy
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.0
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.5
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.5
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.4
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.2
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.5
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.9
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.5

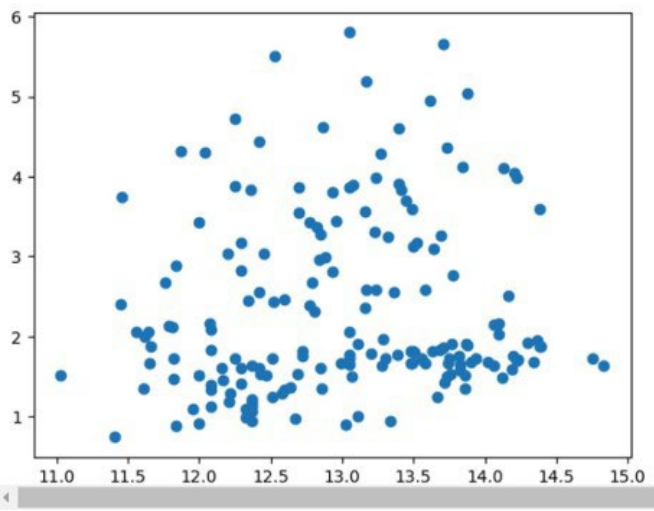
```
x_df.isnull().sum()
```

	0
alcohol	0
malic_acid	0
ash	0
alcalinity_of_ash	0
magnesium	0
total_phenols	0
flavanoids	0
nonflavanoid_phenols	0
proanthocyanins	0
color_intensity	0
hue	0
od280/od315_of_diluted_wines	0
proline	0

```
x_df.duplicated().sum()
```

```
0
```

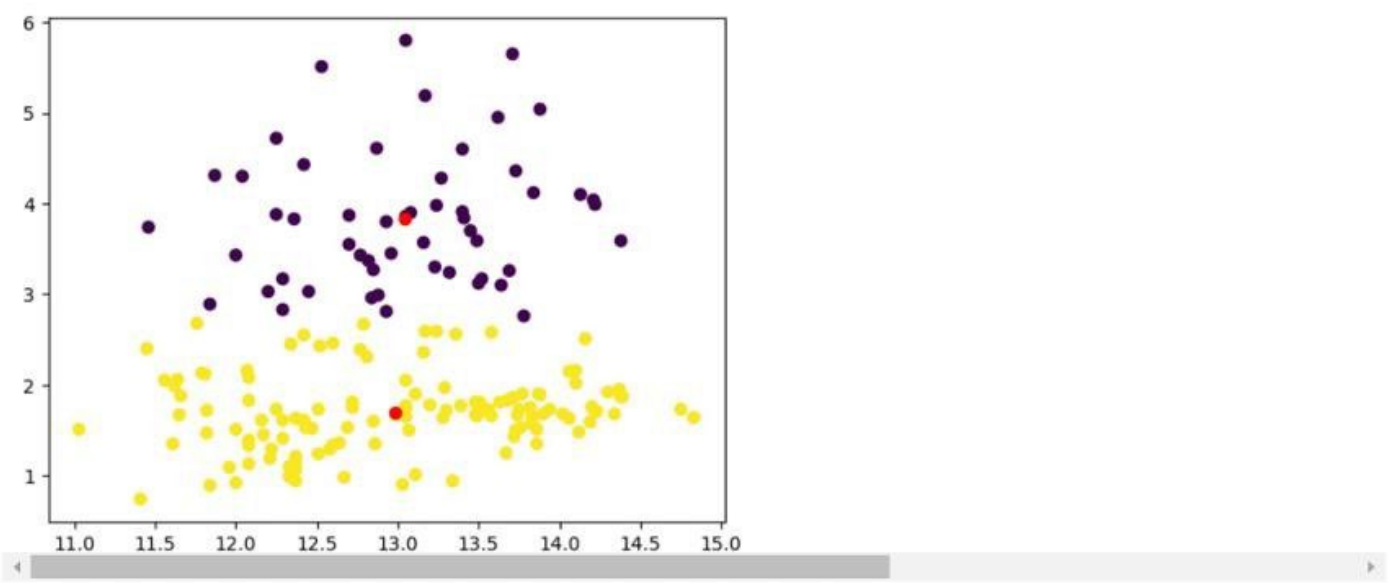
```
plt.scatter(x[:,0], x[:,1])    # (x[:,0], x[:,1]) = (x[all rows, 1st column], x[all rows, 2nd column])
plt.show()
```



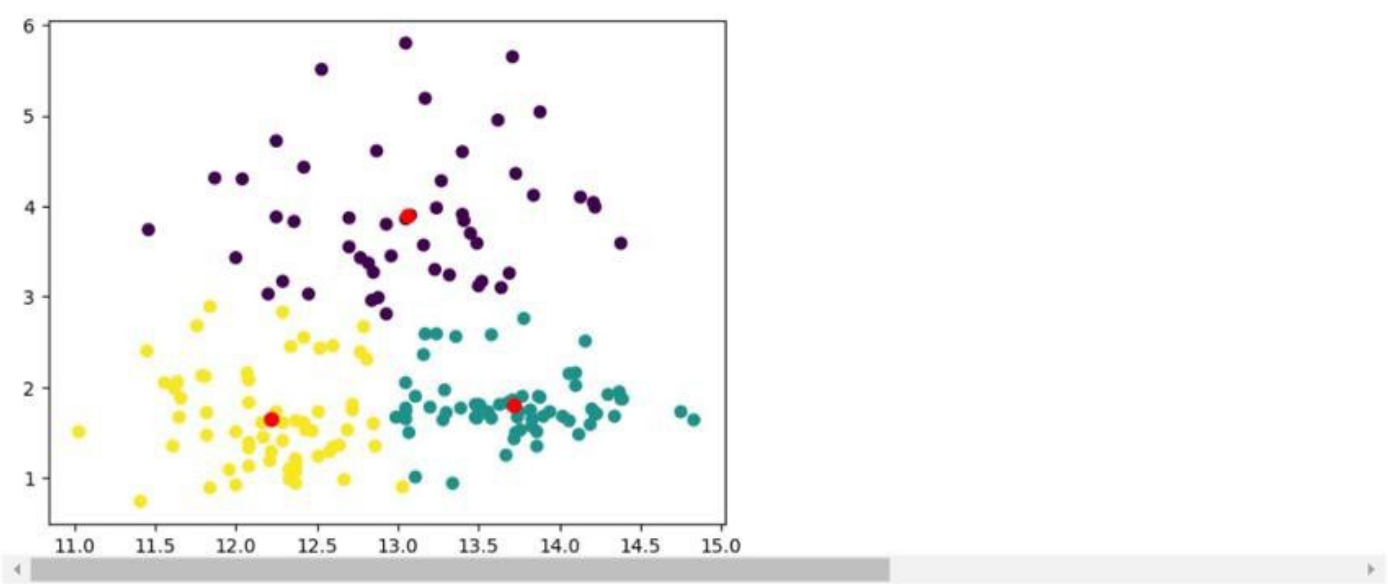
```
cluster_2_model = KMeans(n_clusters=2)
cluster_labels = cluster_2_model.fit_predict(x[:,[0,1]])

cluster_labels[:10], cluster_labels[-10:]
```

```
(array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32),
 array([0, 1, 1, 0, 0, 1, 1, 1, 0, 1], dtype=int32))
```

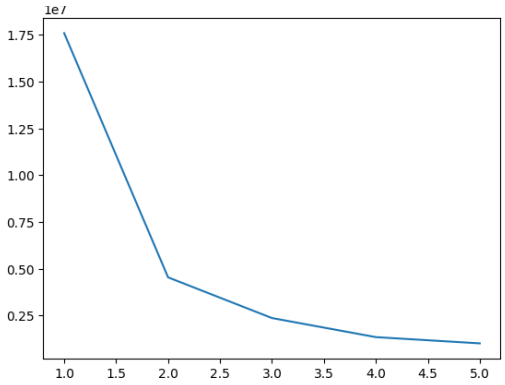
```
cluster_2_model = KMeans(n_clusters=2)
cluster_labels = cluster_2_model.fit_predict(x[:,[0,1]])
plt.scatter(x[:,0], x[:,1], c=cluster_labels)
centroids = cluster_2_model.cluster_centers_
plt.scatter(centroids[:,0], centroids[:,1], c='red')
plt.show()
```




```
cluster_3_model = KMeans(n_clusters=3)
cluster_labels = cluster_3_model.fit_predict(x[:,[0,1]])
plt.scatter(x[:,0], x[:,1], c=cluster_labels)
centroids = cluster_3_model.cluster_centers_
plt.scatter(centroids[:,0], centroids[:,1], s=50, color='red')
plt.show()
```



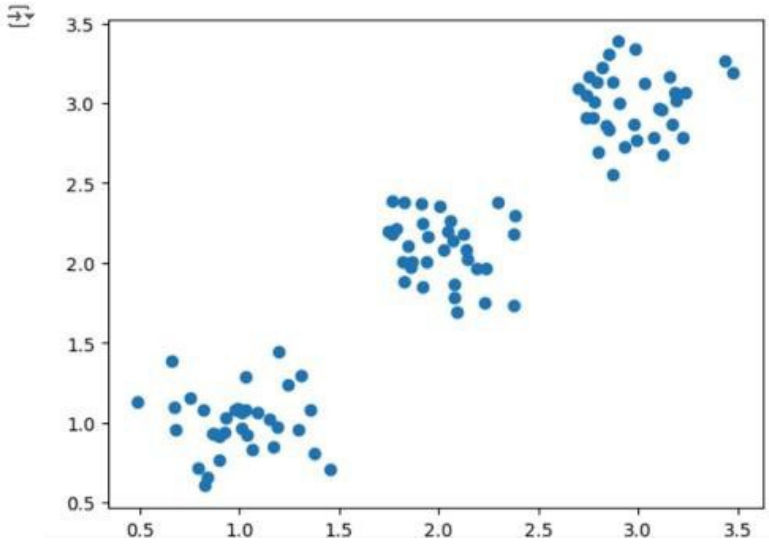
```
# Using Elbow method to find best values of k
k_values = [1, 2, 3, 4, 5]
wcss_values = []

for k_value in k_values:
    model = KMeans(n_clusters=k_value)
    model.fit(x)
    wcss_value = model.inertia_
    wcss_values.append(wcss_value)

plt.plot(k_values, wcss_values)
plt.show()
```




```
from sklearn.metrics import silhouette_score

k_values = [2,3,4,5]
wcss_values = []

for k_value in k_values:
    model = KMeans(n_clusters=k_value)
    cluster_labels = model.fit_predict(x[:,[0, 1]])
    model_score = silhouette_score(x[:,[0,1]], cluster_labels)

    print(f"Cluster score: {model_score} for {k_value}")
```

```
Cluster score: 0.47745578237294 for 2
Cluster score: 0.4806665140821557 for 3
Cluster score: 0.45020918927738124 for 4
Cluster score: 0.37104988795151717 for 5
```

```
x_data = datasets.make_blobs(n_samples = 100, centers=[[1, 1], [2, 2], [3, 3]], cluster_std = 0.2, random_state = 0)
plt.scatter(x_data[0][:,0], x_data[0][:,1])
plt.show()
```



Experiment - 12

Aim: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Backpropagation Algorithm:

```
import tensorflow as tf
from tensorflow.keras import datasets


(x_train, y_train), (x_test, y_test) = datasets.boston_housing.load_data()

print('Training set: ', x_train.shape, y_train.shape)
print('Test set: ', x_test.shape, y_test.shape)
```

```
Training set:  (404, 13) (404,)
Test set:  (102, 13) (102,)
```

```
y_train[:5]
```

```
array([15.2, 42.3, 50. , 21.1, 17.7])
```

```
x_train[0]
```

```
array([  1.23247,   0.     ,   8.14   ,   0.     ,   0.538  ,   6.142  ,
        91.7    ,   3.9769 ,   4.     , 307.     ,  21.     , 396.9    ,
        18.72   ])
```

```
# Build ANN Arch for Regression

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.utils import plot_model

model_reg = Sequential([
    Input((13,)),
    Dense(1, activation='linear')
])

# model_reg.summary()
# plot_model(model_reg, show_shapes=True, show_layer_activations=True)

# Model Training
model_reg.compile(optimizer='sgd', loss='msle', metrics=['accuracy', 'msle'])
model_reg.fit(x_train, y_train, epochs=10)
```

```
Epoch 1/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.0000e+00 - loss: 6.4965 - msle:
Epoch 2/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.0000e+00 - loss: 9.5771 - msle:
Epoch 3/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.0000e+00 - loss: 9.7181 - msle:
Epoch 4/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.0000e+00 - loss: 9.5237 - msle:
Epoch 5/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0000e+00 - loss: 9.4521 - msle:
Epoch 6/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.0000e+00 - loss: 9.6827 - msle:
Epoch 7/10
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0000e+00 - loss: 9.5076 - msle:
Epoch 8/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.0000e+00 - loss: 9.6568 - msle:
Epoch 9/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0000e+00 - loss: 9.7762 - msle:
Epoch 10/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0000e+00 - loss: 9.7529 - msle:
<keras.src.callbacks.history.History at 0x7d5dd8624940>
```

```
# Evaluate the model
model_reg.evaluate(x_test, y_test)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.0000e+00 - loss: 9.6913 - msle: 9.
[9.806747436523438, 0.0, 9.806747436523438]
```

```
# Build ANN Arch for Regression
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.utils import plot_model

model_reg_1 = Sequential()
model_reg_1.add(Dense(50, activation='relu', input_shape=(x_train.shape[1], )))
model_reg_1.add(Dense(25, activation='relu'))
model_reg_1.add(Dense(12, activation='relu'))
model_reg_1.add(Dense(1, activation='linear'))

# model_reg_1.summary()
# plot_model(model_reg_1, show_shapes=True, show_layer_activations=True)

# Model Training
model_reg_1.compile(optimizer='adam', loss='msle', metrics=['accuracy', 'msle'])
model_reg_1.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), batch_size
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarnin
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.0000e+00 - loss: 0.3993 - msle: 0
Epoch 2/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.0000e+00 - loss: 0.1851 - msle: 0.
Epoch 3/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0000e+00 - loss: 0.1372 - msle: 0
Epoch 4/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.0000e+00 - loss: 0.1228 - msle: 0.
Epoch 5/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.0000e+00 - loss: 0.1111 - msle: 0.
Epoch 6/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.0000e+00 - loss: 0.0861 - msle: 0.
Epoch 7/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.0000e+00 - loss: 0.0781 - msle: 0.
Epoch 8/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.0000e+00 - loss: 0.0768 - msle: 0.
Epoch 9/10
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.0000e+00 - loss: 0.0980 - msle: 0
Epoch 10/10
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.0000e+00 - loss: 0.0816 - msle: 0.
<keras.src.callbacks.history.History at 0x7d5dd844fdc0>
```

```
# ANN for binary classification

import pandas as pd
from sklearn.preprocessing import LabelEncoder

bank_ds = pd.read_csv('/content/Bank(1 ).csv', delimiter=';')

le = LabelEncoder()

for col in ['job', 'marital', 'default', 'education', 'housing', 'loan', 'contact', 'mont
    bank_ds[col] = le.fit_transform(bank_ds[col])
bank_ds.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month
0	30	10	1	0	0	1787	0	0	0	19	10
1	33	7	1	1	0	4789	1	1	0	11	8
2	35	4	2	2	0	1350	1	0	0	16	0
3	30	4	1	2	0	1476	1	1	2	3	6
4	59	1	1	1	0	0	1	0	2	5	8

Next steps: Generate code with bank_ds | View recommended plots | New interactive sheet

```
x = bank_ds.iloc[:,:-1].values
y = bank_ds.iloc[:,-1].values

# import numpy as np

# x.shape

# for i in range(x.shape[1]):
#     x[:,i] = x[:,i]/np.max(x[:,i]

from sklearn.preprocessing import StandardScaler

x_std = StandardScaler().fit_transform(x)

x_std[0]
```

```
array([-1.05626965,  1.71680374, -0.24642938, -1.64475535, -0.1307588 ,
        0.12107186, -1.14205138, -0.42475611, -0.72364152,  0.37405206,
        1.48541444, -0.7118608 , -0.57682947, -0.4072183 , -0.32041282,
        0.44441328])
```

```
# ANN

model_bc = Sequential()
model_bc.add(Dense(20, activation='relu', input_shape=(16,)))
model_bc.add(Dense(5, activation='relu'))
model_bc.add(Dense(1, activation='sigmoid')) #binary classification take sigmoid as activ

model_bc.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_bc.fit(x, y, epochs=10, batch_size=128, validation_split=0.2)
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarnin
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
29/29 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5664 - loss: 6.5969 - val_accura
Epoch 2/10
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6788 - loss: 3.1804 - val_accura
Epoch 3/10
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7214 - loss: 2.1814 - val_accura
Epoch 4/10
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7086 - loss: 1.8458 - val_accura
Epoch 5/10
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8008 - loss: 1.3134 - val_accura
Epoch 6/10
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8270 - loss: 1.0517 - val_accura
Epoch 7/10
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8126 - loss: 1.0897 - val_accura
Epoch 8/10
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8322 - loss: 0.8262 - val_accura
Epoch 9/10
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8456 - loss: 0.7307 - val_accura
Epoch 10/10
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8547 - loss: 0.7296 - val_accura
<keras.src.callbacks.history.History at 0x7d5dc4a718a0>
```

```
# prediction of new
pred_proba = model_bc.predict(x)

pred = []

for proba in pred_proba:
    if proba>=0.5:
        pred.append(1)
    else:
        pred.append(0)
```

```
142/142 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

```
from sklearn.metrics import classification_report

cls = classification_report(y, pred)
print(cls)
```

```
              precision    recall  f1-score   support

           0       0.91      0.94      0.92      4000
```

```
              1       0.38      0.29      0.33       521

       accuracy                           0.86      4521
      macro avg       0.64      0.61      0.63      4521
   weighted avg       0.85      0.86      0.86      4521
```

```
from tensorflow.keras import losses
from tensorflow.keras import optimizers


# ANN for multiclass class

from sklearn import datasets
from tensorflow.keras.utils import to_categorical

iris_ds = datasets.load_iris()

x = iris_ds.data
y = iris_ds.target

y_cat = to_categorical(y, num_classes=3) #one-hot encoding

model_mc = Sequential()
model_mc.add(Dense(10, activation='relu', input_shape=(4,)))
model_mc.add(Dense(5, activation='relu'))
model_mc.add(Dense(3, activation='softmax')) #multiclass classification take softma

model_mc.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accur
model_mc.fit(x, y_cat, epochs=4, batch_size=64, validation_split=0.2)
```

```
Epoch 1/4
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarnin
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 219ms/step - accuracy: 0.4236 - loss: 1.1951 - val_accura
Epoch 2/4
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4132 - loss: 1.1791 - val_accurac
Epoch 3/4
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.3924 - loss: 1.1723 - val_accurac
Epoch 4/4
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.4080 - loss: 1.1435 - val_accurac
<keras.src.callbacks.history.History at 0x7d5dc99d5870>
```

```
import numpy as np

x.shape
np.unique(y)
```

```
array([0, 1, 2])
```

```
y_pred_proba = model_mc.predict(x)

y_pred = []
for proba in y_pred_proba:
```

```
    y_pred.append(np.argmax(proba))
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```

```
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        50
           1       0.33      1.00      0.50        50
           2       0.00      0.00      0.00        50

    accuracy                           0.33       150
   macro avg       0.11      0.33      0.17       150
weighted avg       0.11      0.33      0.17       150
```