From : Manya Sajwan (BSC 3rd year)

Roll : 21535

Q1

In [ ]:
```
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd
```

In [ ]:
```
iris = load_iris(as_frame=True)
X = iris["data"]
y = iris["target"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_
```

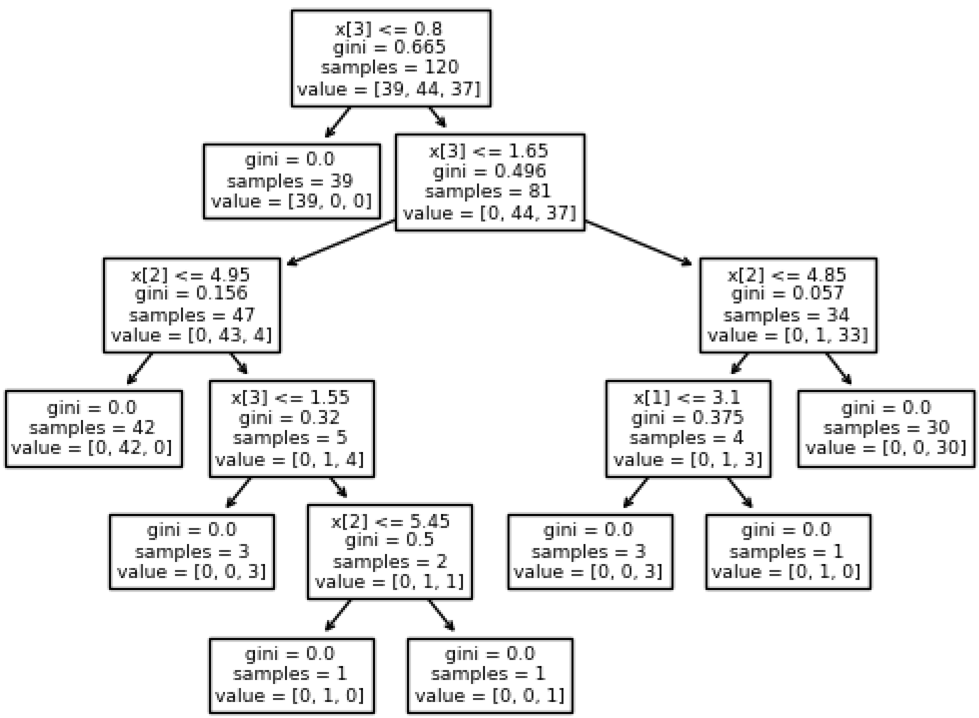
In [ ]:
```
dt = tree.DecisionTreeClassifier()
model = dt.fit(X_train, y_train)
```

In [ ]:
```
preds = model.predict(X_test)
print(f"Accuracy : {accuracy_score(y_test, preds)*100}%")
```

```
Accuracy : 96.66666666666667%
```

In [ ]:
```
tree.plot_tree(model)
```

```
Out[ ]: [Text(0.4, 0.9166666666666666, 'x[3] <= 0.8\ngini = 0.665\nsamples = 120\nvalue =
[39, 44, 37]'),
 Text(0.3, 0.75, 'gini = 0.0\nsamples = 39\nvalue = [39, 0, 0]'),
 Text(0.5, 0.75, 'x[3] <= 1.65\ngini = 0.496\nsamples = 81\nvalue = [0, 44, 37]'),
 Text(0.2, 0.5833333333333334, 'x[2] <= 4.95\ngini = 0.156\nsamples = 47\nvalue =
[0, 43, 4]'),
 Text(0.1, 0.4166666666666667, 'gini = 0.0\nsamples = 42\nvalue = [0, 42, 0]'),
 Text(0.3, 0.4166666666666667, 'x[3] <= 1.55\ngini = 0.32\nsamples = 5\nvalue =
[0, 1, 4]'),
 Text(0.2, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.4, 0.25, 'x[2] <= 5.45\ngini = 0.5\nsamples = 2\nvalue = [0, 1, 1]'),
 Text(0.3, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.5, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.8, 0.5833333333333334, 'x[2] <= 4.85\ngini = 0.057\nsamples = 34\nvalue =
[0, 1, 33]'),
 Text(0.7, 0.4166666666666667, 'x[1] <= 3.1\ngini = 0.375\nsamples = 4\nvalue =
[0, 1, 3]'),
 Text(0.6, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.8, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.9, 0.4166666666666667, 'gini = 0.0\nsamples = 30\nvalue = [0, 0, 30]')]
```



Q2

In [ ]:
```
from sklearn.datasets import load_iris
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
import numpy as np
import pandas as pd
```

In [ ]:
```
iris = load_iris(as_frame=True)
X = iris["data"]
y = iris["target"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_
```

In [ ]:
```
bayes = GaussianNB()
model = bayes.fit(X_train, y_train)
```

In [ ]:
```
preds = model.predict(X_test)
print(f"Accuracy : {accuracy_score(y_test, preds)*100}%")
confusion_matrix(y_test, preds)
```

```
Accuracy : 96.66666666666667%
```

Out[ ]:
```
array([[11,  0,  0],
       [ 0,  5,  1],
       [ 0,  0, 13]], dtype=int64)
```

Q3

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

In [ ]:
```

```

In [ ]:
```
X = load_iris(as_frame=True)["data"]
y = load_iris(as_frame=True)["target"]
X.columns
```

Out[ ]:
```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)'],
      dtype='object')
```

In [ ]:
```
def normalise(feature, df):
    mean = df[feature].mean()
    sd = df[feature].std()
    df[feature] = (df[feature] - mean) / sd

normalise("sepal length (cm)", X)
normalise("sepal width (cm)", X)
normalise("petal length (cm)", X)
normalise("petal width (cm)", X)
X
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	-0.897674	1.015602	-1.335752	-1.311052
1	-1.139200	-0.131539	-1.335752	-1.311052
2	-1.380727	0.327318	-1.392399	-1.311052
3	-1.501490	0.097889	-1.279104	-1.311052
4	-1.018437	1.245030	-1.335752	-1.311052
...	...	...	...	...
145	1.034539	-0.131539	0.816859	1.443994
146	0.551486	-1.278680	0.703564	0.919223
147	0.793012	-0.131539	0.816859	1.050416
148	0.430722	0.786174	0.930154	1.443994
149	0.068433	-0.131539	0.760211	0.788031

150 rows × 4 columns

In [ ]:

```
X_train, X_temp, y_train, y_temp = train_test_split(X , y,random_state=100, test_si

X_validate, X_test, y_validate, y_test = train_test_split(X_temp , y_temp,random_st
```

In [ ]:

```

```

In [ ]:

```
def report_show(x,y, k):
    knn = KNeighborsClassifier(n_neighbors=k, weights="distance")
    knn.fit(X_train, y_train)
    preds = knn.predict(x)
    print(f"Accuracy  : {accuracy_score(y , preds)}")
    print(f"ConfusionMatrix\n{confusion_matrix(y , preds)}")
```

In [ ]:

```
# k = 4
print("Validation")
report_show(X_validate, y_validate, 4)
print("Testing")
report_show(X_test, y_test, 4)
```

```
Validation
Accuracy  : 0.9411764705882353
ConfusionMatrix
[[6 0 0]
 [0 4 0]
 [0 1 6]]
Testing
Accuracy  : 1.0
ConfusionMatrix
[[6 0 0]
 [0 4 0]
 [0 0 7]]
```

In [ ]:

```python
# k = 5
print("Validation")
report_show(X_validate, y_validate, 5)
print("Testing")
report_show(X_test, y_test, 5)
```

```
Validation
Accuracy  : 0.9411764705882353
ConfusionMatrix
[[6 0 0]
 [0 4 0]
 [0 1 6]]
Testing
Accuracy  : 1.0
ConfusionMatrix
[[6 0 0]
 [0 4 0]
 [0 0 7]]
```

In [ ]:

```python
# k = 8
print("Validation")
report_show(X_validate, y_validate, 2)
print("Testing")
report_show(X_test, y_test, 1)
```

```
Validation
Accuracy  : 0.9411764705882353
ConfusionMatrix
[[6 0 0]
 [0 4 0]
 [0 1 6]]
Testing
Accuracy  : 1.0
ConfusionMatrix
[[6 0 0]
 [0 4 0]
 [0 0 7]]
```

best k = 4

Q4

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

In [ ]:
```
df = pd.read_csv("../data/salary.csv")
y = df["Salary"]
X = df["YearsExperience"]
df.head()
```

Out[ ]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X , y,random_state=100, test_si
```

In [ ]:
```
reg = LinearRegression()
reg.fit(np.array(X_train).reshape(-1,1) , y_train)
```

Out[ ]:

```
▾ LinearRegression
LinearRegression()
```

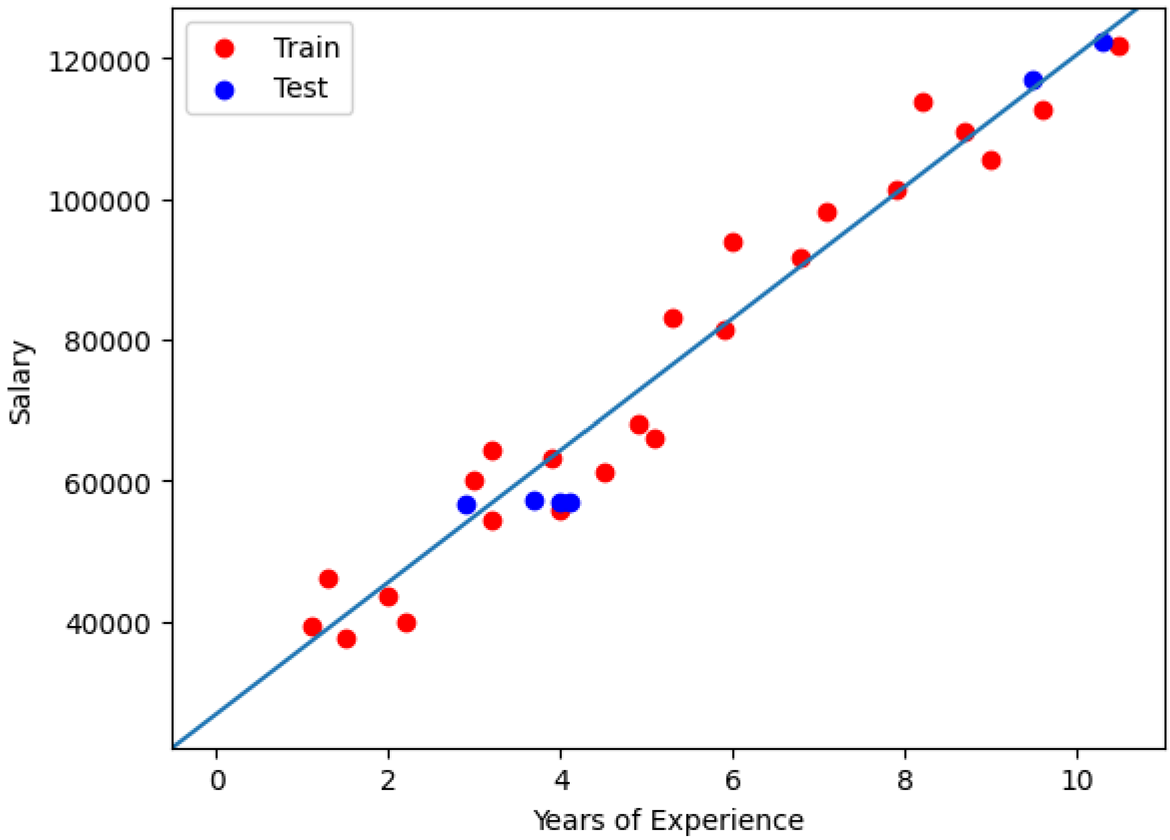
In [ ]:
```
preds = reg.predict(np.array(X_test).reshape(-1,1))
print(f"MSE : {mean_squared_error(y_test, preds)}")
print(f"R2 score : {r2_score(y_test, preds)}")
```

```
MSE : 24477109.08965574
R2 score : 0.9720725422361338
```

In [ ]:
```
plt.scatter(X_train, y_train , color='red', label="Train")
plt.scatter(X_test, y_test , color='blue', label="Test")
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.legend()

intercept = reg.intercept_
```

```python
slope = reg.coef_[0]
plt.axline((0 , intercept),slope=slope)
```

Out[ ]: <matplotlib.lines.AxLine at 0x297af11d6d0>



Q5

In [ ]:
```
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
import pandas as pd
```

In [ ]:
```
obj = fetch_california_housing()
data = obj["data"]
```

In [ ]:
```
for i in range(8):
    for j in range(i+1, 8):
        print(f"Correlations of column {i} with column {j} ")
        print(np.corrcoef(data[:,i], data[:,j])[0,1])
```

```
Correlations of column 0 with column 1
-0.11903398993785665
Correlations of column 0 with column 2
0.3268954316412956
Correlations of column 0 with column 3
-0.062040133836099076
Correlations of column 0 with column 4
0.004834345627652915
Correlations of column 0 with column 5
0.01876624796696885
Correlations of column 0 with column 6
-0.0798091274597188
Correlations of column 0 with column 7
-0.015175865414173956
Correlations of column 1 with column 2
-0.15327742256198937
Correlations of column 1 with column 3
-0.07774728275376118
Correlations of column 1 with column 4
-0.2962442397735358
Correlations of column 1 with column 5
0.01319135663602974
Correlations of column 1 with column 6
0.011172673530605408
Correlations of column 1 with column 7
-0.10819681311244811
Correlations of column 2 with column 3
0.8476213257130447
Correlations of column 2 with column 4
-0.07221284865893354
Correlations of column 2 with column 5
-0.004852294991781336
Correlations of column 2 with column 6
0.1063889654862552
Correlations of column 2 with column 7
-0.027540053873544787
Correlations of column 3 with column 4
-0.06619740232676065
Correlations of column 3 with column 5
-0.006181201268673116
Correlations of column 3 with column 6
0.0697211298887421
Correlations of column 3 with column 7
0.0133443896399991
Correlations of column 4 with column 5
0.06986273036567671
Correlations of column 4 with column 6
-0.1087847473776677
Correlations of column 4 with column 7
0.09977322287464561
Correlations of column 5 with column 6
0.0023661822637503493
Correlations of column 5 with column 7
0.0024758163767050613
Correlations of column 6 with column 7
-0.9246644339150403
```

In [ ]:
```
X = data
y = obj.target
```

Out[ ]: 20640

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X , y,random_state=100, test_si
reg = LinearRegression()
reg.fit(X_train, y_train)
```

Out[ ]:
```
LinearRegression
LinearRegression()
```

In [ ]:
```
preds = reg.predict(X_test)
print(f"MSE : {mean_squared_error(y_test, preds)}")
print(f"R2 score : {r2_score(y_test, preds)}")
```

```
MSE : 0.5088933351158983
R2 score : 0.6223138107295262
```

In [ ]:
```
print(reg.coef_, reg.intercept_)
```

```
[ 4.33432793e-01  9.22564691e-03 -1.06547768e-01  6.46494007e-01
 -7.07960568e-06 -3.45850134e-03 -4.23282369e-01 -4.37465774e-01] -37.20562128878796
```

Q6

In [ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

In [ ]:
```
data = pd.read_csv("../data/advertising.csv", index_col="ID")
data
```

Out[ ]:

	TV	Radio	Newspaper	Sales
ID				
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9
...	...	...	...	...
196	38.2	3.7	13.8	7.6
197	94.2	4.9	8.1	9.7
198	177.0	9.3	6.4	12.8
199	283.6	42.0	66.2	25.5
200	232.1	8.6	8.7	13.4

200 rows × 4 columns

In [ ]:
```
data.corr()
```

Out[ ]:

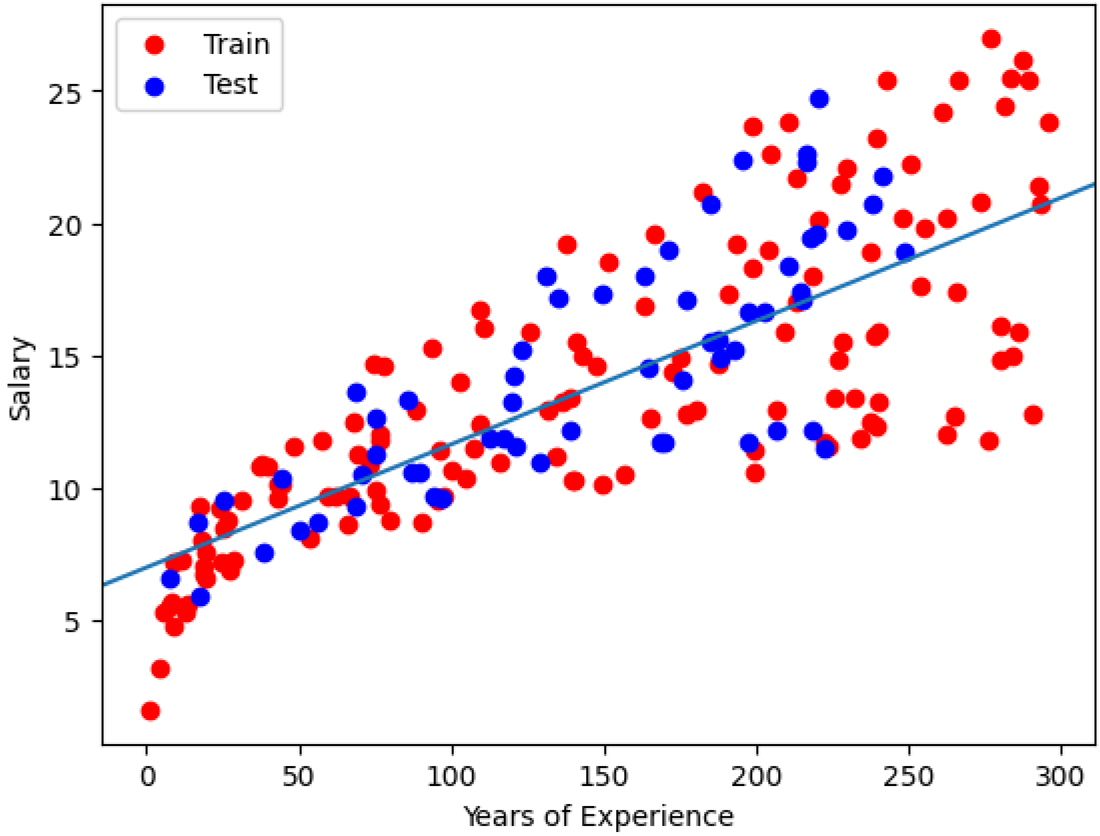
	TV	Radio	Newspaper	Sales
TV	1.000000	0.054809	0.056648	0.782224
Radio	0.054809	1.000000	0.354104	0.576223
Newspaper	0.056648	0.354104	1.000000	0.228299
Sales	0.782224	0.576223	0.228299	1.000000

In [ ]:
```
def sales_regression(feature):
    X = data[feature]
    y = data["Sales"]

    X_train, X_test, y_train, y_test = train_test_split(X , y,random_state=100, tes

    reg = LinearRegression()
    reg.fit(np.array(X_train).reshape(-1,1) , y_train)

    preds = reg.predict(np.array(X_test).reshape(-1,1))
    print(f"MSE : {mean_squared_error(y_test, preds)}")
    print(f"R2 score : {r2_score(y_test, preds)}")

    plt.scatter(X_train, y_train , color='red', label="Train")
    plt.scatter(X_test, y_test , color='blue', label="Test")
    plt.xlabel(feature)
    plt.ylabel("Sales")
    plt.legend()

    intercept = reg.intercept_
    slope = reg.coef_[0]
    plt.axline((0 , intercept),slope=slope)
```

1. TV and Sales

In [ ]:
```
sales_regression("TV")
```

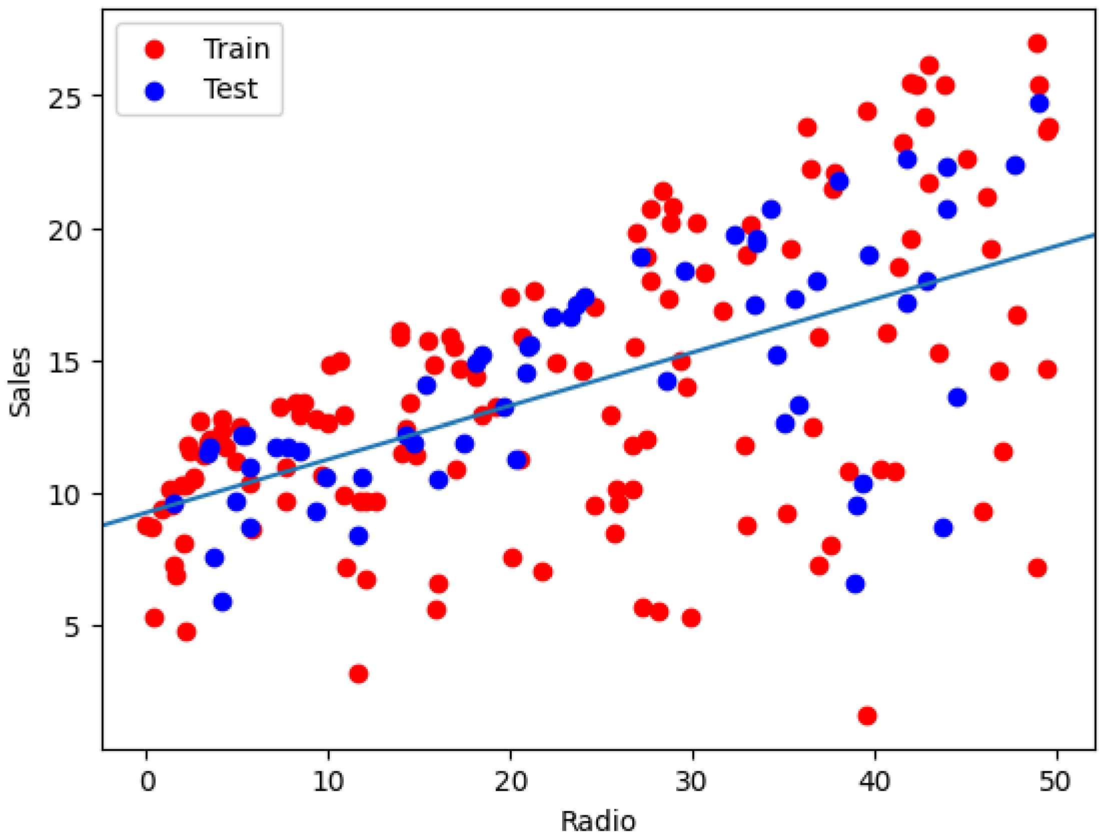
```
MSE : 7.975798532854851
R2 score : 0.5942987267783302
```



2. Radio and Sales

In [ ]:
```
sales_regression("Radio")
```

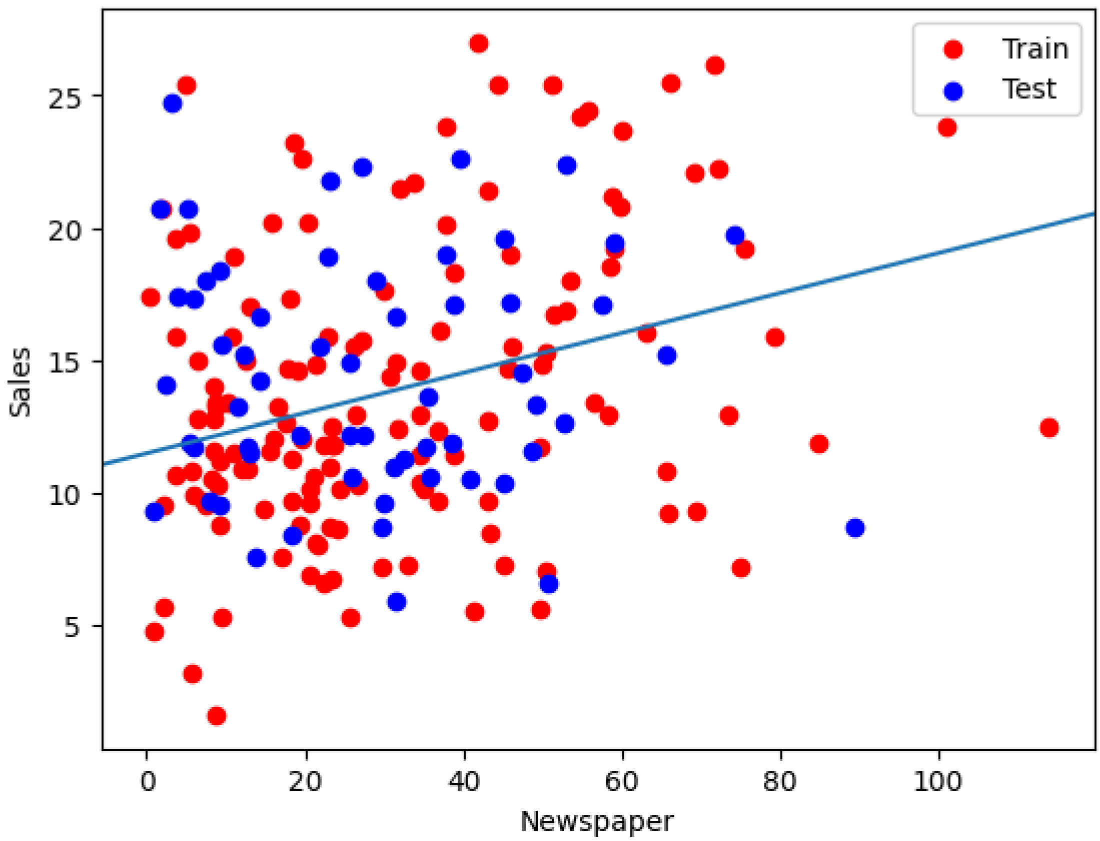
```
MSE : 11.388611592147727
R2 score : 0.4207007355904727
```



3. Newspaper and Sales

In [ ]:
```
sales_regression("Newspaper")
```

```
MSE : 22.78312971627622
R2 score : -0.15889897366292205
```


Train
Test
Sales
Newspaper
25
20
15
10
5
0
20
40
60
80
100

Q7

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

In [ ]:
```
data = pd.read_csv("../data/advertising.csv", index_col="ID")
data
```

Out[ ]:

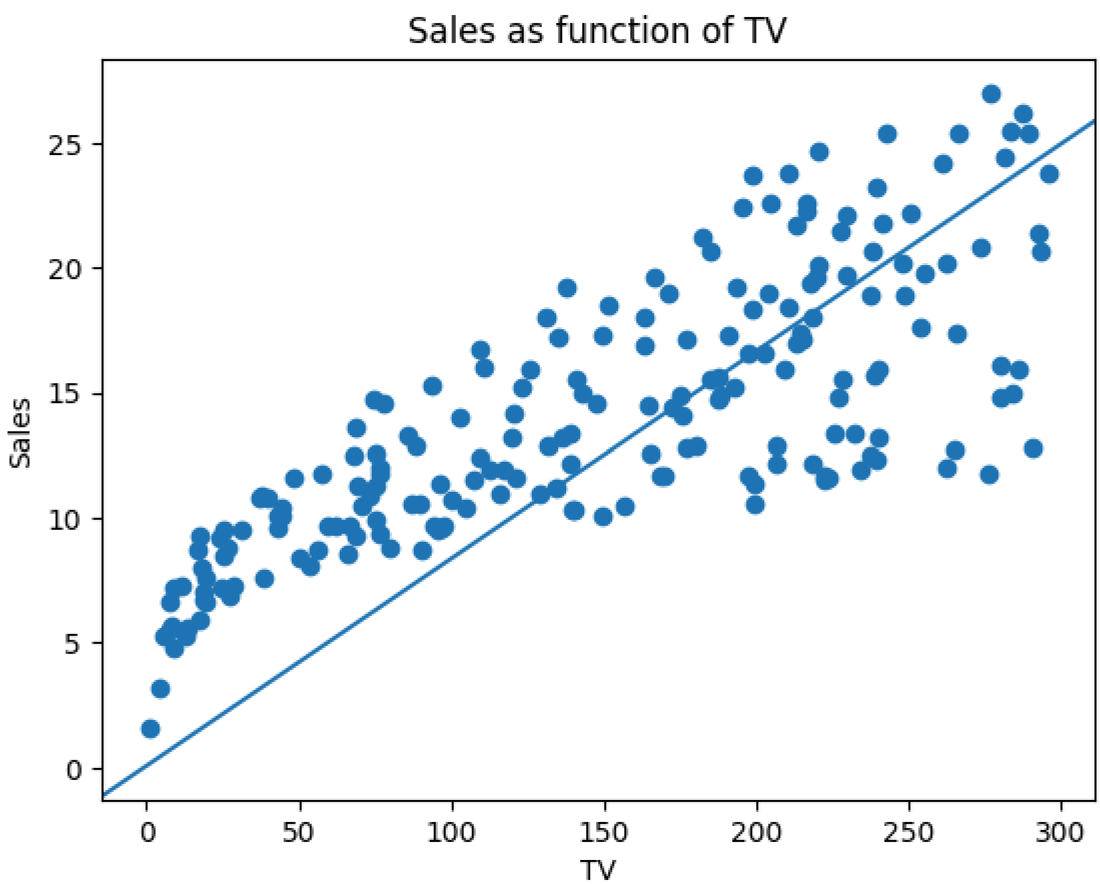
	TV	Radio	Newspaper	Sales
ID				
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9
...	...	...	...	...
196	38.2	3.7	13.8	7.6
197	94.2	4.9	8.1	9.7
198	177.0	9.3	6.4	12.8
199	283.6	42.0	66.2	25.5
200	232.1	8.6	8.7	13.4

200 rows × 4 columns

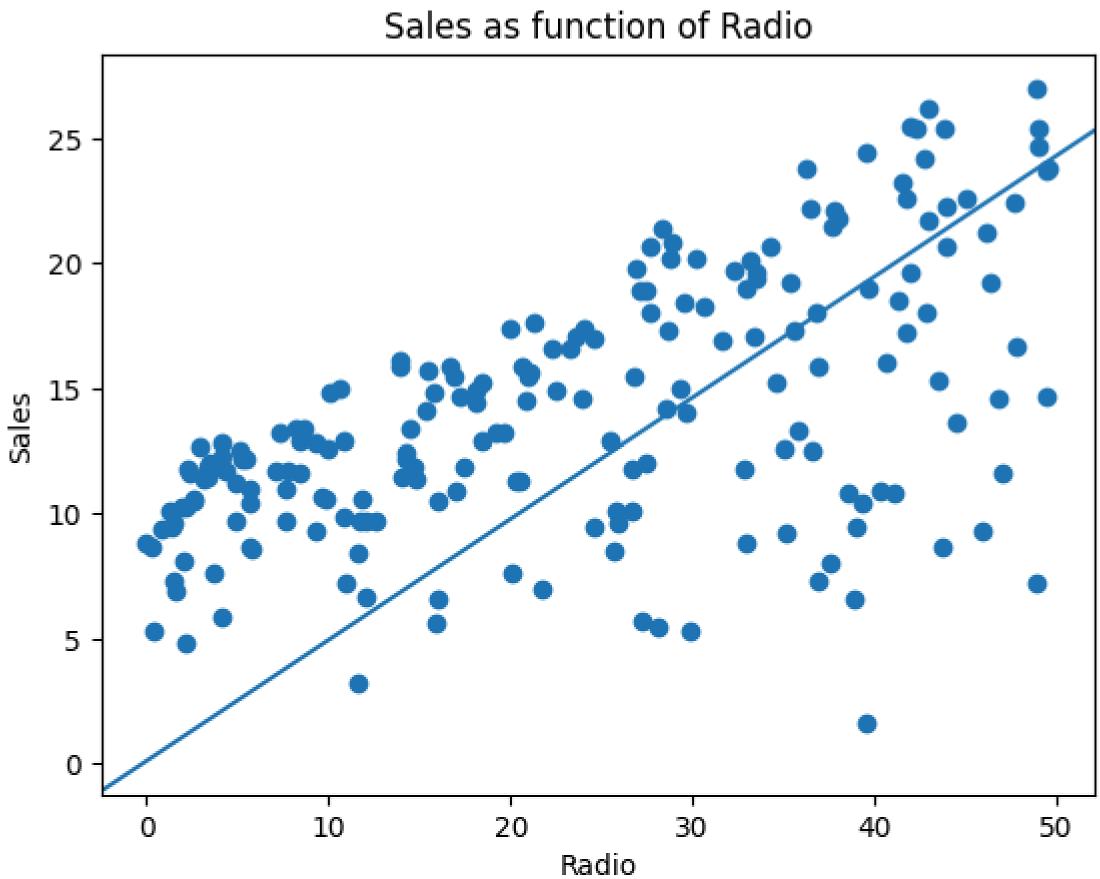
In [ ]:
```
def sales_regression_gradient_descent(feature):
    X = data[feature]
    y = data["Sales"]


    # starting params
    m = 0
    c = 0
    L = .0001 # ;earning param
    n = 1000 # iterations


    for i in range(n):
        y_pred = X*m + c

        D_m = -2/n * (X * ( y - y_pred)).sum()
```

```
        D_c = -2/n *  ( y - y_pred).sum()

        m -= L * D_m
        c -= L * D_c



    # plotting
    plt.scatter(X,y)
    plt.axline((0,c) , slope=m)
    plt.xlabel(feature)
    plt.ylabel("Sales")
    plt.title(f"Sales as function of {feature}")
```

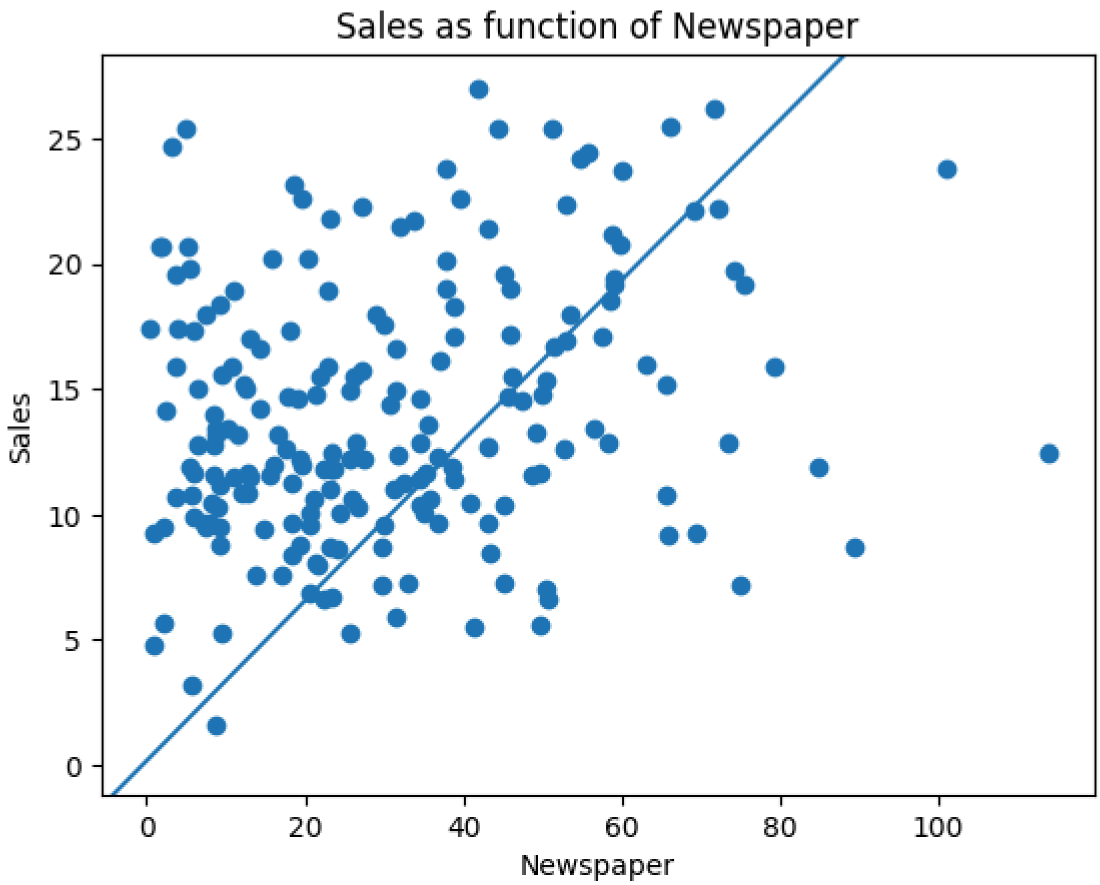
In [ ]:
```
sales_regression_gradient_descent("TV")
```



In [ ]:
```
sales_regression_gradient_descent("Radio")
```




```
In [ ]: sales_regression_gradient_descent("Newspaper")
```


Sales as function of Newspaper
Sales
Newspaper
0
5
10
15
20
25
0
20
40
60
80
100

Q8

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
```

Load Data

In [ ]:
```
data = load_breast_cancer()
X = data["data"]
y = data["target"]
```

Split data

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

Train model

In [ ]:
```
logReg = LogisticRegression(solver="liblinear")
logReg.fit(X_train, y_train)
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression(solver='liblinear')
```

In [ ]:
```
preds = logReg.predict(X_test)

print(classification_report(y_test, preds))
print(confusion_matrix(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       0.94      0.93      0.93        69
           1       0.95      0.96      0.96       102

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

[[64  5]
 [ 4 98]]
```

Q9

In [ ]:
```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.datasets import load_iris
```

Load Data

In [ ]:
```
iris = load_iris(as_frame=True)
X = iris["data"]
y = iris["target"]
```

Splitting data

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

Training model

In [ ]:
```
logReg = LogisticRegression()
logReg.fit(X_train, y_train)

logReg.predict_proba(X_test)
```

Out[ ]:
```
array([[1.72598811e-05, 4.90291644e-02, 9.50953576e-01],
       [9.69214414e-01, 3.07854618e-02, 1.24183591e-07],
       [7.50624085e-09, 1.62768540e-03, 9.98372307e-01],
       [9.82927219e-01, 1.70727393e-02, 4.21211089e-08],
       [7.15478659e-04, 4.25835699e-01, 5.73448822e-01],
       [7.18446986e-07, 1.64090909e-02, 9.83590191e-01],
       [9.85298821e-01, 1.47011559e-02, 2.35223546e-08],
       [9.64044750e-01, 3.59551647e-02, 8.52166111e-08],
       [7.83270592e-05, 1.42220683e-01, 8.57700990e-01],
       [9.66368158e-01, 3.36317326e-02, 1.09810718e-07],
       [9.82411080e-01, 1.75888853e-02, 3.47154490e-08],
       [7.66166173e-05, 7.14842482e-02, 9.28439135e-01],
       [9.61534401e-01, 3.84653649e-02, 2.33560855e-07],
       [9.71162735e-01, 2.88372120e-02, 5.27061771e-08],
       [8.16888878e-05, 1.04016764e-01, 8.95901547e-01],
       [9.14086083e-03, 9.27237076e-01, 6.36220634e-02],
       [9.83601240e-03, 9.73039428e-01, 1.71245596e-02],
       [6.90673755e-04, 4.94946398e-01, 5.04362928e-01],
       [6.40120032e-08, 6.50415317e-03, 9.93495783e-01],
       [3.21021679e-05, 8.23861355e-02, 9.17581762e-01],
       [7.51176227e-06, 6.01994490e-02, 9.39793039e-01],
       [9.55549701e-01, 4.44501372e-02, 1.61797376e-07],
       [2.99934586e-04, 2.92468479e-01, 7.07231586e-01],
       [9.64060645e-01, 3.59391950e-02, 1.60373489e-07],
       [2.48009786e-02, 9.55663962e-01, 1.95350595e-02],
       [6.05370394e-04, 2.63945058e-01, 7.35449572e-01],
       [5.53391900e-03, 9.11003802e-01, 8.34622789e-02],
       [9.35689796e-01, 6.43099620e-02, 2.42434932e-07],
       [3.45452200e-03, 8.48057454e-01, 1.48488024e-01],
       [9.16733722e-06, 3.24799711e-02, 9.67510862e-01]])
```

Predictions and metrics

In [ ]:
```python
preds = logReg.predict(X_test)

print("Confusion Matrix")
confusion_matrix(y_test, preds)
```

```
Confusion Matrix
```

Out[ ]:
```
array([[11,  0,  0],
       [ 0,  5,  1],
       [ 0,  0, 13]], dtype=int64)
```

In [ ]:
```python
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      0.83      0.91         6
           2       0.93      1.00      0.96        13

    accuracy                           0.97        30
   macro avg       0.98      0.94      0.96        30
weighted avg       0.97      0.97      0.97        30
```

Q10

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
```

In [ ]:
```
data = pd.read_csv("../data/advertising.csv", index_col="ID")
data
```

Out[ ]:

ID	TV	Radio	Newspaper	Sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9
...	...	...	...	...
196	38.2	3.7	13.8	7.6
197	94.2	4.9	8.1	9.7
198	177.0	9.3	6.4	12.8
199	283.6	42.0	66.2	25.5
200	232.1	8.6	8.7	13.4

200 rows × 4 columns

In [ ]:
```
class Linear_Regression_Gradient_Descent:

    def __init__(self):
        self.slope = 0
        self.intercept = 0


    def fit(self, X , y , L=.0001 , n = 1000 ):
        # starting params
        m = 0
        c = 0
        L = .0001 # ;earning param
        n = 1000 # iterations
```

```
        for i in range(n):
            y_pred = X*m + c

            D_m = -2/n * (X * ( y - y_pred)).sum()
            D_c = -2/n *  ( y - y_pred).sum()

            m -= L * D_m
            c -= L * D_c

        self.slope = m
        self.intercept = c

    def predict(self, X):
        return self.slope*X + self.intercept
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(data["Radio"] , data["Sales"],r
```

In [ ]:
```
model = Linear_Regression_Gradient_Descent()
model.fit(X_train, y_train)
```

In [ ]:
```
preds = model.predict(X_test)
print(f"MSE : {mean_squared_error(y_test, preds)}")
print(f"R2 score : {r2_score(y_test, preds)}")
```

```
MSE : 34.02575687759605
R2 score : -0.730772515216807
```

In [ ]:
```
print("Line")
print(f"Slope : {model.slope}")
print(f"Intercept : {model.intercept}")
```

```
Line
Slope : 0.47989071486978124
Intercept : 0.09238662391493048
```

Q11

In [ ]:
```
from sklearn.datasets import load_wine
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.decomposition import PCA
import numpy as np
import pandas as pd
```

In [ ]:
```
wine = load_wine()
X = wine["data"]
y = wine["target"]
```

In [ ]:
```
p = PCA(n_components=5)
p.fit(X)
X = p.transform(X)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_
```

In [ ]:
```
gauss = GaussianNB()
gauss.fit(X_test, y_test)


# train Accuracy
preds_train = gauss.predict(X_train)
print(f"Training Accuracy : {accuracy_score(y_train, preds_train)}")

# Testing Accuracy
preds_test = gauss.predict(X_test)
print(f"Testing Accuracy : {accuracy_score(y_test, preds_test)}")
```

```
Training Accuracy : 0.8739495798319328
Testing Accuracy : 0.9661016949152542
```

Q12

In [ ]:
```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
```

In [ ]:
```
data = load_iris()
X = data["data"]
y = data["target"]
```

In [ ]:
```
p = PCA(n_components=2)
X = pd.DataFrame(p.fit_transform(X), columns=["X0" , "X1"])
X
```

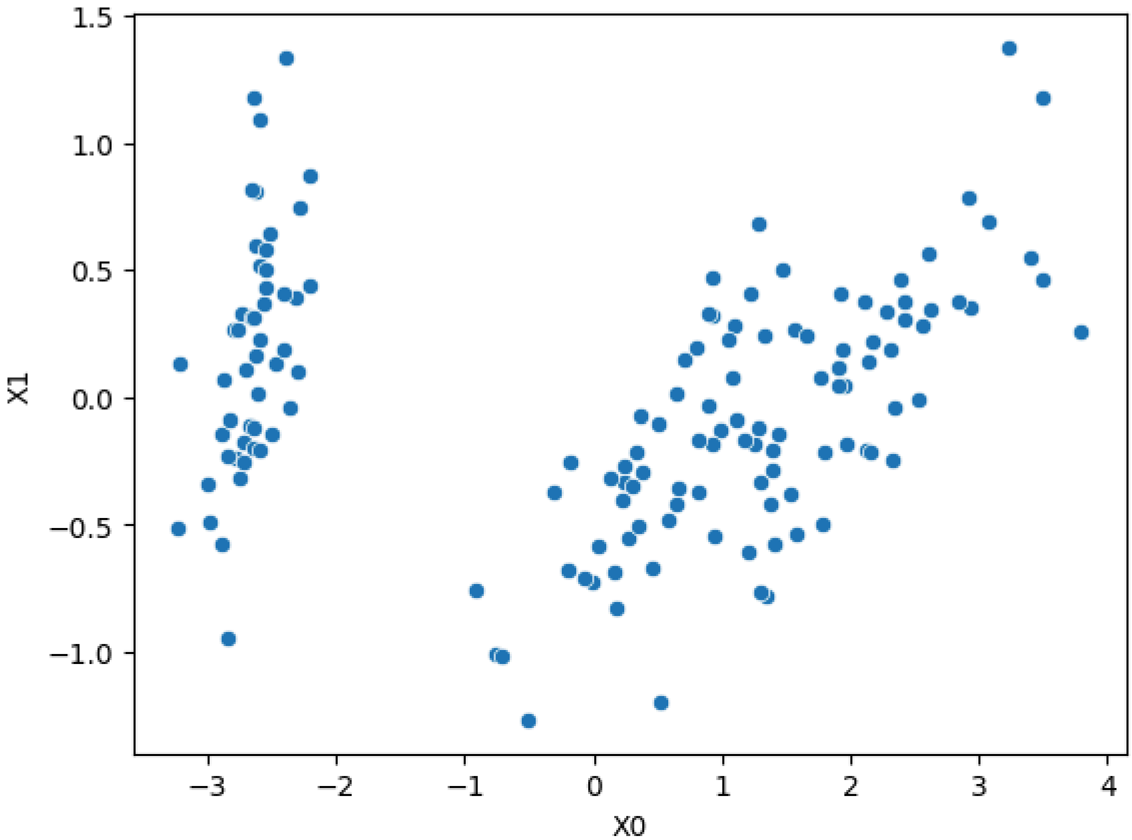
Out[ ]:

	X0	X1
0	-2.684126	0.319397
1	-2.714142	-0.177001
2	-2.888991	-0.144949
3	-2.745343	-0.318299
4	-2.728717	0.326755
...	...	...
145	1.944110	0.187532
146	1.527167	-0.375317
147	1.764346	0.078859
148	1.900942	0.116628
149	1.390189	-0.282661

150 rows × 2 columns

In [ ]:
```
sns.scatterplot(data = X, x="X0" , y="X1")
```

Out[ ]:
```
<Axes: xlabel='X0', ylabel='X1'>
```



In [ ]:
```
kmeans = KMeans(n_clusters=3 ,random_state = 0, n_init='auto')
kmeans.fit(X)
```

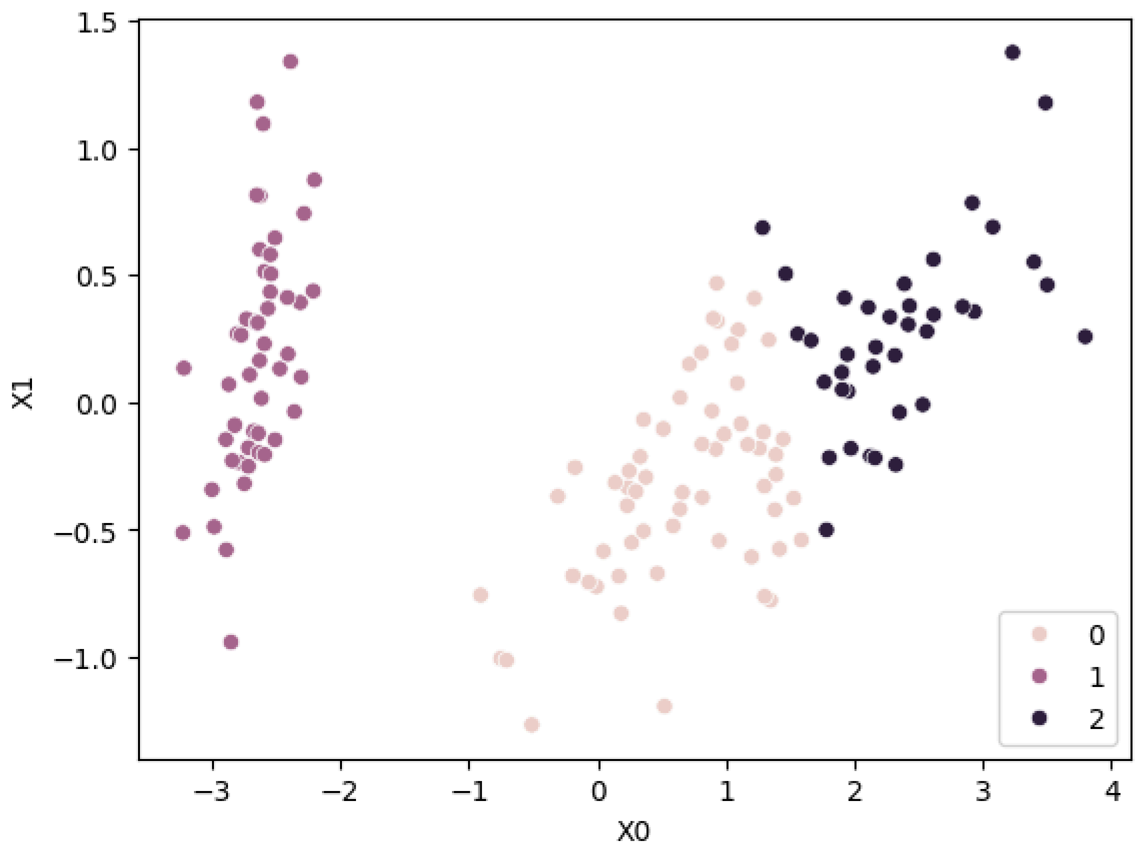
Out[ ]:

▾ KMeans

```
KMeans(n_clusters=3, random_state=0)
```

In [ ]:
```
sns.scatterplot(data = X, x="X0" , y="X1", hue = kmeans.labels_)
```

Out[ ]: <Axes: xlabel='X0', ylabel='X1'>


X1
X0
0
1
2

Q13

In [ ]:
```
from sklearn.linear_model import Perceptron
import pandas as pd
```

In [ ]:
```
df = pd.DataFrame(
    [[0,        0,      0,      0],
    [0, 0,      1,      0],
    [0, 1,      0,      0],
    [0, 1,      1,      0],
    [1, 0,      0,      0],
    [1, 0,      1,      0],
    [1, 1,      0,      0],
    [1, 1,      1,      1]] , columns=["A" , "B" , "C" , "Y"]
)
df
```

Out[ ]:

	A	B	C	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

In [ ]:
```
X = df.drop(["Y"], axis=1)
y = df["Y"]
```

In [ ]:
```
p = Perceptron()
p.fit(X,y)
p.score(X,y)
```

Out[ ]:
```
0.75
```

Q14

In [ ]:
```
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
import numpy as np
import pandas as pd
```

In [ ]:
```
data = load_iris()
X = data["data"]
y = data["target"]
```

In [ ]:
```
hierarchical_cluster = AgglomerativeClustering(n_clusters=3 )
labels = hierarchical_cluster.fit_predict(X)

labels
```

Out[ ]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,
       2, 0, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int64)
```

Q15

Loading the modules

In [ ]:
```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

Load digits dataset

In [ ]:
```
digits = load_digits()
X = digits.data
y = digits.target
```

In [ ]:
```
# Normalize the features
X = X / 255.0

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# One-hot encode the labels
num_classes = len(np.unique(y))
y_train_onehot = tf.one_hot(y_train, depth=num_classes)
y_test_onehot = tf.one_hot(y_test, depth=num_classes)
```

Define the architecture of the neural network

In [ ]:
```
input_dim = X_train.shape[1]
output_dim = num_classes
hidden_dim = 64


# Define the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(hidden_dim, input_dim=input_dim, activation='relu'),
    tf.keras.layers.Dense(output_dim, activation='softmax')
])
```

```
c:\Users\user\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\la
yers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
t to a layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

ANN Training

In [ ]:

```
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['a

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.1)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", accuracy)
```

```
Epoch 1/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.1298 - loss: 2.2954 - val_accur
acy: 0.4524 - val_loss: 2.2680
Epoch 2/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5135 - loss: 2.2612 - val_accur
acy: 0.6429 - val_loss: 2.2274
Epoch 3/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6894 - loss: 2.2162 - val_accur
acy: 0.7857 - val_loss: 2.1710
Epoch 4/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7762 - loss: 2.1640 - val_accur
acy: 0.8254 - val_loss: 2.1051
Epoch 5/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7622 - loss: 2.0983 - val_accur
acy: 0.8095 - val_loss: 2.0240
Epoch 6/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7787 - loss: 2.0158 - val_accur
acy: 0.8254 - val_loss: 1.9347
Epoch 7/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7817 - loss: 1.9354 - val_accur
acy: 0.8571 - val_loss: 1.8327
Epoch 8/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7890 - loss: 1.8362 - val_accur
acy: 0.8492 - val_loss: 1.7269
Epoch 9/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8094 - loss: 1.7265 - val_accur
acy: 0.8810 - val_loss: 1.6254
Epoch 10/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8309 - loss: 1.6218 - val_accur
acy: 0.8651 - val_loss: 1.5210
Epoch 11/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8121 - loss: 1.5271 - val_accur
acy: 0.8810 - val_loss: 1.4179
Epoch 12/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8380 - loss: 1.4032 - val_accur
acy: 0.8730 - val_loss: 1.3153
Epoch 13/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8099 - loss: 1.3507 - val_accur
acy: 0.8889 - val_loss: 1.2287
Epoch 14/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8314 - loss: 1.2427 - val_accur
acy: 0.8651 - val_loss: 1.1473
Epoch 15/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8232 - loss: 1.1766 - val_accur
acy: 0.8730 - val_loss: 1.0782
Epoch 16/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8405 - loss: 1.0889 - val_accur
acy: 0.8730 - val_loss: 1.0054
Epoch 17/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8684 - loss: 1.0074 - val_accur
acy: 0.8810 - val_loss: 0.9466
Epoch 18/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8530 - loss: 0.9671 - val_accur
acy: 0.9048 - val_loss: 0.8938
Epoch 19/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8559 - loss: 0.9229 - val_accur
```

```
acy: 0.9048 - val_loss: 0.8442
Epoch 20/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8689 - loss: 0.8571 - val_accur
acy: 0.9048 - val_loss: 0.7980
Epoch 21/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8698 - loss: 0.8319 - val_accur
acy: 0.9048 - val_loss: 0.7589
Epoch 22/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8850 - loss: 0.7715 - val_accur
acy: 0.9048 - val_loss: 0.7170
Epoch 23/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8800 - loss: 0.7246 - val_accur
acy: 0.9048 - val_loss: 0.6873
Epoch 24/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8875 - loss: 0.6987 - val_accur
acy: 0.9048 - val_loss: 0.6566
Epoch 25/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8985 - loss: 0.6450 - val_accur
acy: 0.9048 - val_loss: 0.6327
Epoch 26/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8925 - loss: 0.6506 - val_accur
acy: 0.9048 - val_loss: 0.6021
Epoch 27/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9026 - loss: 0.6189 - val_accur
acy: 0.9048 - val_loss: 0.5784
Epoch 28/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9006 - loss: 0.5966 - val_accur
acy: 0.9048 - val_loss: 0.5587
Epoch 29/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8957 - loss: 0.6022 - val_accur
acy: 0.9127 - val_loss: 0.5347
Epoch 30/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9186 - loss: 0.5373 - val_accur
acy: 0.9048 - val_loss: 0.5173
Epoch 31/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9025 - loss: 0.5541 - val_accur
acy: 0.9048 - val_loss: 0.5020
Epoch 32/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9144 - loss: 0.5234 - val_accur
acy: 0.9127 - val_loss: 0.4836
Epoch 33/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9203 - loss: 0.4877 - val_accur
acy: 0.9127 - val_loss: 0.4697
Epoch 34/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9194 - loss: 0.4826 - val_accur
acy: 0.9206 - val_loss: 0.4513
Epoch 35/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9131 - loss: 0.4829 - val_accur
acy: 0.9127 - val_loss: 0.4411
Epoch 36/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9260 - loss: 0.4503 - val_accur
acy: 0.9206 - val_loss: 0.4254
Epoch 37/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9231 - loss: 0.4320 - val_accur
acy: 0.9206 - val_loss: 0.4154
Epoch 38/50
```

```
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9249 - loss: 0.4360 - val_accur
acy: 0.9206 - val_loss: 0.4041
Epoch 39/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9181 - loss: 0.4391 - val_accur
acy: 0.9206 - val_loss: 0.3929
Epoch 40/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9354 - loss: 0.4066 - val_accur
acy: 0.9206 - val_loss: 0.3836
Epoch 41/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9448 - loss: 0.3809 - val_accur
acy: 0.9206 - val_loss: 0.3721
Epoch 42/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9439 - loss: 0.3568 - val_accur
acy: 0.9206 - val_loss: 0.3668
Epoch 43/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9446 - loss: 0.3577 - val_accur
acy: 0.9206 - val_loss: 0.3507
Epoch 44/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9363 - loss: 0.3561 - val_accur
acy: 0.9206 - val_loss: 0.3477
Epoch 45/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9327 - loss: 0.3597 - val_accur
acy: 0.9206 - val_loss: 0.3392
Epoch 46/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9308 - loss: 0.3619 - val_accur
acy: 0.9206 - val_loss: 0.3316
Epoch 47/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9326 - loss: 0.3507 - val_accur
acy: 0.9206 - val_loss: 0.3249
Epoch 48/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9389 - loss: 0.3197 - val_accur
acy: 0.9206 - val_loss: 0.3182
Epoch 49/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9428 - loss: 0.3198 - val_accur
acy: 0.9206 - val_loss: 0.3127
Epoch 50/50
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9499 - loss: 0.3049 - val_accur
acy: 0.9206 - val_loss: 0.3054
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9282 - loss: 0.2927
Test Accuracy: 0.9222221970558167
```