**SIMPLE LINEAR REGRESSION**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

 dataset = pd.read\_csv('Salary\_Data.csv')

dataset.head()

 X = dataset.iloc[:,:-1].values  #independent variable array

y = dataset.iloc[:,1].values  #dependent variable vector

 from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=1/3,random\_state=42)

 from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train,y\_train) #actually produces the linear eqn for the data

 y\_pred = regressor.predict(X\_test)

y\_pred

 y\_test

 print('Train Score: ', regressor.score(X\_train, y\_train))

print('Test Score: ', regressor.score(X\_test, y\_test))

 #plot for the TRAIN

 plt.scatter(X\_train, y\_train, color='red') # plotting the observation line

 plt.plot(X\_train, regressor.predict(X\_train), color='blue') # plotting the regression line

 plt.title("Salary vs Experience (Training set)") # stating the title of the graph

 plt.xlabel("Years of experience") # adding the name of x-axis

plt.ylabel("Salaries") # adding the name of y-axis

plt.show() # specifies end of graph

 #plot for the TEST

 plt.scatter(X\_test, y\_test, color='red')

plt.plot(X\_train, regressor.predict(X\_train), color='blue') # plotting the regression line

 plt.title("Salary vs Experience (Testing set)")

 plt.xlabel("Years of experience")

plt.ylabel("Salaries")

plt.show()

**OUTPUT:**



**MULTIPLE LINEAR REGRESSION**

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets

data\_set= pd.read\_csv('50\_Startups.csv')

data\_set

#Extracting Independent and dependent Variable

x= data\_set.iloc[:, :-1].values

y= data\_set.iloc[:, 4].values

print(x)

print(y)

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.compose import ColumnTransformer

# State column

ct = ColumnTransformer([("State", OneHotEncoder(), [3])], remainder = 'passthrough')

X = ct.fit\_transform(x)

 X

 x = X[:, 1:]

 x

 # Splitting the dataset into training and test set.

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=0)

from sklearn.linear\_model import LinearRegression

regressor= LinearRegression()

regressor.fit(x\_train, y\_train)

#Predicting the Test set result;

y\_pred= regressor.predict(x\_test)

print(y\_pred)

print(y\_test)

print('Train Score: ', regressor.score(x\_train, y\_train))

print('Test Score: ', regressor.score(x\_test, y\_test))

**OUTPUT:**



**LOGISTIC REGRESSION**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

dataset = pd.read\_csv('User\_Data.csv')

dataset.head()

# input

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

# output

y = dataset.iloc[:, 4].values

from sklearn.model\_selection import train\_test\_split

xtrain, xtest, ytrain, ytest = train\_test\_split(

        x, y, test\_size = 0.25, random\_state = 42)

from sklearn.preprocessing import StandardScaler

sc\_x = StandardScaler()

xtrain = sc\_x.fit\_transform(xtrain)

xtest = sc\_x.transform(xtest)

print (xtrain[0:10, :])

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 42)

classifier.fit(xtrain, ytrain)

y\_pred = classifier.predict(xtest)

print(y\_pred)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(ytest, y\_pred)

print ("Confusion Matrix : \n", cm)

from sklearn.metrics import accuracy\_score

print ("Accuracy : ", accuracy\_score(ytest, y\_pred))

from matplotlib.colors import ListedColormap

X\_set, y\_set = xtest, ytest

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1,

                               stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1,

                               stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(

             np.array([X1.ravel(), X2.ravel()]).T).reshape(

             X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Classifier (Test set)')

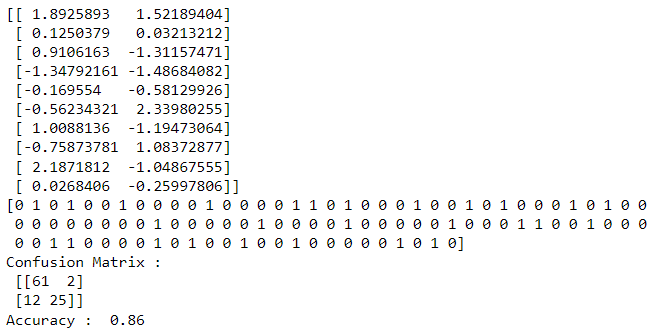
plt.xlabel('Age')

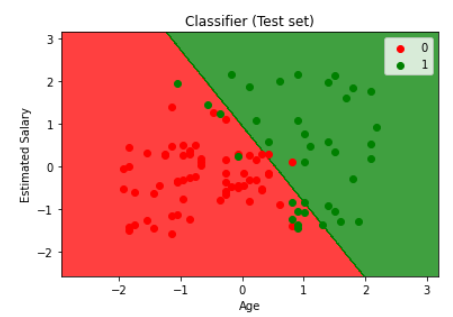
plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

OUTPUT:

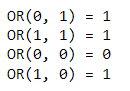




**OR GATE**

import numpy as np  
    
# define Unit Step Function  
def unitStep(v):  
    if v >= 0:  
        return 1  
    else:  
        return 0  
# design Perceptron Model  
def perceptronModel(x, w, b):  
    v = np.dot(w, x) + b  
    y = unitStep(v)  
    return y  
# OR Logic Function  
# w1 = 1, w2 = 1, b = -0.5  
def OR\_logicFunction(x):  
    w = np.array([1, 1])  
    b = -0.5  
    return perceptronModel(x, w, b)  
    
# testing the Perceptron Model  
test1 = np.array([0, 1])  
test2 = np.array([1, 1])  
test3 = np.array([0, 0])  
test4 = np.array([1, 0])  
    
print("OR({}, {}) = {}".format(0, 1, OR\_logicFunction(test1)))  
print("OR({}, {}) = {}".format(1, 1, OR\_logicFunction(test2)))  
print("OR({}, {}) = {}".format(0, 0, OR\_logicFunction(test3)))  
print("OR({}, {}) = {}".format(1, 0, OR\_logicFunction(test4)))

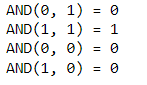
**OUTPUT:**



**AND GATE**

import numpy as np  
    
# define Unit Step Function  
def unitStep(v):  
    if v >= 0:  
        return 1  
    else:  
        return 0  
    
# design Perceptron Model  
def perceptronModel(x, w, b):  
    v = np.dot(w, x) + b  
    y = unitStep(v)  
    return y  
    
# AND Logic Function  
# w1 = 1, w2 = 1, b = -1.5  
def AND\_logicFunction(x):  
    w = np.array([1, 1])  
    b = -1.5  
    return perceptronModel(x, w, b)  
    
# testing the Perceptron Model  
test1 = np.array([0, 1])  
test2 = np.array([1, 1])  
test3 = np.array([0, 0])  
test4 = np.array([1, 0])  
    
print("AND({}, {}) = {}".format(0, 1, AND\_logicFunction(test1)))  
print("AND({}, {}) = {}".format(1, 1, AND\_logicFunction(test2)))  
print("AND({}, {}) = {}".format(0, 0, AND\_logicFunction(test3)))  
print("AND({}, {}) = {}".format(1, 0, AND\_logicFunction(test4)))

**OUTPUT:**



**SVM**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

df = pd.read\_csv('Social\_Network\_Ads.csv')

df.head()

df.shape

X = df.iloc[:, [2,3]]

Y = df.iloc[:, 4]

X.head()

Y.head()

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_Train, X\_Test, Y\_Train, Y\_Test = train\_test\_split(X, Y, test\_size = 0.25, random\_state = 0)

print("Training data : ",X\_Train.shape)

print("Training data : ",X\_Test.shape)

 # Feature Scaling

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_Train = sc\_X.fit\_transform(X\_Train)

X\_Test = sc\_X.transform(X\_Test)

from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random\_state = 0)

classifier.fit(X\_Train, Y\_Train)

# Predicting the test set results

Y\_Pred = classifier.predict(X\_Test)

Y\_Pred

from sklearn import metrics

print('Accuracy Score: with linear kernel')

print(metrics.accuracy\_score(Y\_Test,Y\_Pred))

import matplotlib.pyplot as plt

plt.scatter(X\_Train[:, 0], X\_Train[:, 1],c=Y\_Train)

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.title('Training Data')

plt.show()

import matplotlib.pyplot as plt

plt.scatter(X\_Test[:, 0], X\_Test[:, 1],c=Y\_Test)

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.title('Test Data')

plt.show()

**OUTPUT:**

