SINGLE LINEAR REGRESSION

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import *

import os
os.getcwd()

data1 = pd.read_csv("/Users/pinkslayer/Desktop/SAT to GPA.csv”)

data1

scalerx = StandardScaler()
scalery = StandardScaler()
scalerx.fit(data1['SAT Score'].values.reshape(-1,1))
scalery.fit(data1['GPA'].values.reshape(-1,1))
newX = scalerx.transform(data1['SAT Score'].values.reshape(-1,1))
newY = scalery.transform(data1['GPA'].values.reshape(-1,1))
print(newX, newY)

lin = LinearRegression()
xtrain, xtest, ytrain, ytest = train_test_split(newX, newY, test_size=0.3, random_state=3)
lin.fit(xtrain, ytrain)
print(f"Slope = ",lin.coef_[0][0],"Intercept: ", lin.intercept_[0])
preds = lin.predict(xtest)

print("The Equation is: \nY =",lin.coef_[0][0],'* X +',lin.intercept_[0])


plt.xlabel('Scaled SAT')
plt.ylabel('Scaled GPA')
plt.scatter(xtrain, ytrain, c='green')
plt.scatter(xtest, ytest, c='blue')
plt.plot
plt.plot(xtest, preds, c='red')
plt.show()

## Optional , If they ask to predict a new random value

x = [[1324]]
pred_single = lin.predict(scalerx.transform(x))
print("Scaled output for 1324", pred_single[0][0])
print("Unscaled output for 1324", scalery.inverse_transform(pred_single)[0][0])

mse= mean_squared_error(ytest, preds)
mae = mean_absolute_error(ytest, preds)
rmse = pow(mse, 0.5)
print("Mean Squared Error: ", mse)
print("Mean absolute Error: ", mae)
print("Root Mean Squared Error: ", rmse)
```

MULTIPLE LINEAR REGRESSION

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
import seaborn as sns
%matplotlib inline

import os
os.getcwd()

data = pd.read_csv('/Users/pinkslayer/Downloads/Datasets/Advertising.csv')

data = data.drop('Unnamed: 0', axis=1)
data

scalerx = StandardScaler()
scalery = StandardScaler()

scalerx.fit(data[['TV', 'radio', 'newspaper']])
scalery.fit(data['sales'].values.reshape(-1,1))
newX = scalerx.transform(data[['TV', 'radio', 'newspaper']])
newY = scalery.transform(data['sales'].values.reshape(-1,1))
newX

lin = LinearRegression()
xtrain, xtest, ytrain, ytest = train_test_split(newX, newY, test_size=0.3, random_state=3)
lin.fit(xtrain, ytrain)
print("Slopes = ",lin.coef_,"Intercept: ", lin.intercept_[0])
print("Slope of TV: ",lin.coef_[0][0],"\nSlope of Radio: ",lin.coef_[0][1],"\nSlope of Newspaper: 
",lin.coef_[0][2])
preds=lin.predict(xtest)


print("The Equation is: \nY =",lin.coef_[0][0],'* X1',lin.coef_[0][1],'* X2',lin.coef_[0][2],'* X3 
+',lin.intercept_[0])

sns.pairplot(data, x_vars=['TV', 'radio', 'newspaper'], y_vars='sales', height=7, aspect=0.7, 
kind='reg')

## Optional , If they ask to predict a new random value

x = [[230, 37, 69]]
pred_single = lin.predict(scalerx.transform(x))
print("Scaled output ", pred_single)
print("Unscaled output ", scalery.inverse_transform(pred_single))

mse= mean_squared_error(ytest, preds)
mae = mean_absolute_error(ytest, preds)
rmse = pow(mse, 0.5)
print("Mean Squared Error: ", mse)
print("Mean absolute Error: ", mae)
print("Root Mean Squared Error: ", rmse)
```

Decision trees

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
import seaborn as sns
%matplotlib inline

import os
os.getcwd()

data = pd.read_csv('/Users/pinkslayer/Downloads/Datasets/Play_Tennis.csv')

data

le = LabelEncoder()
data['outlook']= le.fit_transform(data['outlook'])
data['temp']= le.fit_transform(data['temp'])
data['humidity']= le.fit_transform(data['humidity'])
data['windy']= le.fit_transform(data['windy'])

data['play']= le.fit_transform(data['play'])
data


X = data[['outlook', 'temp', 'humidity', 'windy']]
Y = data['play']
new_data=[[2,0,1,1]]

entropyModel = DecisionTreeClassifier(criterion='entropy')
entropyModel.fit(X,Y)

plt.figure(figsize=(10,9))
tree.plot_tree(entropyModel, filled=True, feature_names=['outlook', 'temperature', 'humidity',
'windy'], class_names='play')
plt.show()

output = entropyModel.predict(new_data)
if output[0] == 1:
   print("Yes")
else:
   print("No")

giniModel = DecisionTreeClassifier(criterion='gini')
giniModel.fit(X,Y)

plt.figure(figsize=(13,11))
tree.plot_tree(giniModel,filled=True, feature_names=['outlook', 'temperature', 'humidity', 'windy'],
class_names='play')
plt.show()


output = giniModel.predict(new_data)
if output[0] == 1:
```

```
    print("Yes")
else:
    print(“No")

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.datasets import make_classification
from sklearn.svm import SVC

X, y = make_classification(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = SVC(random_state=0)
clf.fit(X_train, y_train)
SVC(random_state=0)


predictions = clf.predict(X_test)
cm = confusion_matrix(y_test, predictions, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()

predictions = clf.predict(X_test)
precision, recall, _ = precision_recall_curve(y_test, predictions)
disp = PrecisionRecallDisplay(precision=precision, recall=recall)
disp.plot()
plt.show()
```

LOGISTIC REGRESSION

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
import seaborn as sns
%matplotlib inline

import os
os.getcwd()

data=pd.read_csv('/Users/pinkslayer/Downloads/Datasets/diabetes3.csv')
data

labels = data[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age']]
outcome = data[‘Outcome']

xtrain, xtest, ytrain, ytest = train_test_split(labels, outcome, test_size=0.25, random_state=35)
reg = LogisticRegression(solver='liblinear') ##Used liblinear as its efficient for small datasets
reg.fit(xtrain, ytrain)
print("slopes: ", reg.coef_, "intercept", reg.intercept_)
preds = reg.predict(xtest)

cm = confusion_matrix(ytest, preds)
```

```
print(cm)
cfd=ConfusionMatrixDisplay(cm, display_labels=reg.classes_)
cfd.plot()
plt.show()

acc = accuracy_score(ytest,preds)
rec = recall_score(ytest, preds)
pre = precision_score(ytest, preds)
f1 = f1_score(ytest, preds)
auc = roc_auc_score(ytest, preds)
print(f"{acc=},{rec=}, {pre=}, {f1=}, {auc=}")
fprs,tprs,thresh = roc_curve(ytest, preds)
plt.plot(fprs, tprs)
plt.show()
```

K-MEANS:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

import os
os.getcwd()


data = pd.read_csv('/Users/pinkslayer/Downloads/Datasets/ColourXY.csv')
print(data)
print("\nUnique values: ", data[‘color'].unique())

plt.xlabel('X')
plt.ylabel('Y')
plt.title('Dataset')
plt.scatter(data['x'], data['y'],c = data['color'], cmap ='rainbow')
plt.show()

features = data[['x', 'y']]
features

wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(features, data['color'])
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method Graph')
plt.xlabel('Number of clusters(k)')
plt.ylabel('wcss_list')
plt.show()


kmeans = KMeans(n_clusters=3)
plt.title('Clusters=3')
plt.xlabel('X')
plt.ylabel('Y')
plt.scatter(data['x'], data['y'],c = kmeans.fit_predict(features, data['color']), cmap ='rainbow')
plt.show()
```

KNN

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import *



import os
os.getcwd()

data = pd.read_csv('/Users/pinkslayer/Downloads/Datasets/balance-scale2.csv')
data

features = data[['L-Weight', 'L-Distance', 'R-Weight', 'R-Distance']]
target = data['Class']
xtrain, xtest, ytrain, ytest = train_test_split(features, target, test_size=0.3, random_state=235)

accuracy=[]
for i in range(1,24):

    model = KNeighborsClassifier(n_neighbors=i)

    model.fit(xtrain, ytrain)

    preds=model.predict(xtest)

    print(f"Accuracy of {i} th neighbour : ",accuracy_score(ytest, preds))
    accuracy.append(accuracy_score(ytest, preds))

plt.figure(figsize=(15,8))
plt.scatter(range(1,24),accuracy)
plt.plot(range(1,24), accuracy)
plt.xticks(range(1,24))
plt.show

model = KNeighborsClassifier(n_neighbors=21)
model.fit(xtrain, ytrain)
preds=model.predict(xtest)

cm = confusion_matrix(ytest, preds)
print(cm)
cmd = ConfusionMatrixDisplay(cm, display_labels=model.classes_)
cmd.plot()
plt.show()
```

MULTINOMIAL NAIVE BAYES

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
import seaborn as sns
%matplotlib inline

import os
os.getcwd()

data = pd.read_csv('/Users/pinkslayer/Downloads/Datasets/balance-scale.data')
data

data.rename(columns={'1': 'LW', '1.1': 'LD', '1.2': 'RW', '1.3': 'RD'}, inplace=True)
print(data)
data.info()

features = data[['x', 'y']]
features

le = LabelEncoder()
labels=data[['LW', 'LD', 'RW', 'RD']]
data['Ble'] = le.fit_transform(data['B'])
outcome = data['Ble']
trainx, testx, trainy, testy = train_test_split(labels, outcome, test_size=0.25, random_state=29538)

model = GaussianNB()
model.fit(trainx, trainy)
preds = model.predict(testx)
proba = model.predict_proba(testx)

cm = confusion_matrix(testy, preds)
print(cm)
cmd = ConfusionMatrixDisplay(cm, display_labels=model.classes_)
cmd.plot()
plt.show()
testx.shape


acc = accuracy_score(testy,preds)
rec = recall_score(testy, preds , average='micro')
pre = precision_score(testy, preds, average='micro')
f1 = f1_score(testy, preds, average='micro')
mcc = matthews_corrcoef(testy, preds)
print(f"{acc=},{rec=}, {pre=}, {f1=}, {mcc=}”)

fpr={}
tpr={}
thresh={}
auc={}

def get_auc(fprs, tprs):
    fpr = fprs[::-1]
    tpr = tprs[::-1]
```

```
    x1,y1 = fpr[0], tpr[0]
    auc=0.0
    diffs = [fpr[i] - fpr[i-1] for i in range(1, len(fpr))]
    for x,y in zip(diffs, tpr[1:]):
        auc+=(x*y1)
        auc+=(x*(y-y1)/2)
        y1=y
    return auc


nclass = 3
for i in range(nclass):
    fpr[i], tpr[i], thresh[i] = roc_curve(testy, proba[:,i], pos_label=i)
    auc[i] = get_auc(fpr[i], tpr[i])


print("AUC's:")
print("balanced vs rest: ", -auc[0])
print("left vs rest: ", -auc[1])
print("right vs rest: ", -auc[2])



plt.plot(fpr[0], tpr[0], linestyle='-', color='red', label='balanced vs rest')
plt.plot(fpr[1], tpr[1], linestyle='-', color='yellow', label='left vs rest')
plt.plot(fpr[2], tpr[2], linestyle='-', color='blue', label='right vs rest')
plt.plot([[0, 0], [1, 1]], linestyle='--', color="black")
plt.title('ROC Curve')
plt.legend(loc='best')
plt.show()
```

MLP

```
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
import seaborn as sns
%matplotlib inline

import os
os.getcwd()

data = pd.read_csv('/Users/pinkslayer/Downloads/Datasets/HR_comma_sep.csv')
data

le = LabelEncoder()
data['dept'] = le.fit_transform(data['sales'])
data['salary_encoded'] = le.fit_transform(data['salary'])
print(data['dept'].unique())
print(data[‘salary_encoded'].unique())

features =
data[['satisfaction_level','last_evaluation','number_project','average_montly_hours','time_spend_c
ompany','Work_accident','promotion_last_5years','dept','salary_encoded']]
```

```
outcome = data['left']
xtrain, xtest, ytrain, ytest = train_test_split(features, outcome, test_size=0.3, random_state=325)

clf = MLPClassifier(hidden_layer_sizes=(6,5),
                    random_state=5,
                    verbose=False,
                    learning_rate_init=0.01)

clf.fit(xtrain,ytrain)

testX = [[0.37,0.32,2,188,3,0,0,7,0]]  #Employee will leave

testpred=clf.predict(testX)
print("testpred is :", testpred)

if testpred == 0:
  print("Employee will stay")
else:
  print("Employee will leave")


# clf = MLPClassifier(hidden_layer_sizes=(6,5),
#                     random_state=5,
#                     verbose=False,
#                     learning_rate_init=0.01)


N_TRAIN_SAMPLES = xtrain.shape[0]
N_EPOCHS = 25
N_BATCH = 128
N_CLASSES = np.unique(ytrain)

scores_train = []
scores_test = []

epoch = 0
while epoch < N_EPOCHS:

    random_perm = np.random.permutation(xtrain.shape[0])
    mini_batch_index = 0


    while True:

        indices = random_perm[mini_batch_index:mini_batch_index + N_BATCH]

        clf.partial_fit(xtrain.iloc[indices], ytrain.iloc[indices], classes=N_CLASSES)

        mini_batch_index += N_BATCH

        if mini_batch_index >= N_TRAIN_SAMPLES:
            break

    scores_train.append(clf.score(xtrain, ytrain))

    scores_test.append(clf.score(xtest, ytest))

    epoch += 1
```

```
fig, ax = plt.subplots(2, sharex=True, sharey=True)
ax[0].plot(scores_train)
ax[0].set_title('Train')
ax[1].plot(scores_test)
ax[1].set_title('Test')
fig.suptitle("Accuracy over epochs", fontsize=14)
plt.show()

preds = clf.predict(xtest)
cm = confusion_matrix(ytest, preds)
cmd = ConfusionMatrixDisplay(cm, display_labels=clf.classes_)
cmd.plot()
plt.show()

acc = accuracy_score(ytest,preds)
rec = recall_score(ytest, preds)
pre = precision_score(ytest, preds)
f1 = f1_score(ytest, preds)
auc = roc_auc_score(ytest, preds)
print(f"{acc=},{rec=}, {pre=}, {f1=}, {auc=}")
fprs,tprs,thresh = roc_curve(ytest, preds)
plt.plot(fprs, tprs)
```