1)KNN

```
#!/usr/bin/env python
# coding: utf-8

from sklearn.datasets import load_iris

iris = load_iris()
print(iris.data[:5])
X = iris.data
print("\n", type(X))
print("\n", iris.feature_names)
print("\n", "Printing of target label values or prediction label values")
print(iris.target, "\n")
y = iris.target
print("#Printing of target label or prediction label")
print(iris.target_names, "\n")
print(iris.data.shape, "\n")

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=4)
print(X_train.shape)
print(X_train, "\n")
print(y_train.shape)
print(y_train, "\n")
print(X_test.shape)
print(X_test, "\n")
print(y_test.shape)
print(y_test)

from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
k_range = range(1, 26)
scores = {}
scores_list = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores[k] = metrics.accuracy_score(y_test, y_pred)
    scores_list.append(scores[k])

print("\n", scores_list)
```

```
get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib.pyplot as plt
plt.plot(k_range, scores_list)
plt.xlabel("k value")
plt.ylabel("Testing Accuracy")
```

2)DECISON TREE

```
#!/usr/bin/env python
# coding: utf-8

import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

def importdata():
	balance_data =
pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-s
cale.data', sep=',', header=None)
	print("Dataset Length: ", len(balance_data))
	print("Dataset Shape: ", balance_data.shape)
	print("Dataset: ", balance_data.head())
	return balance_data

def splitdataset(balance_data):
	X = balance_data.values[:, 1:5]
	Y = balance_data.values[:, 0]
	X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=100)
	return X, Y, X_train, X_test, y_train, y_test

def train_using_gini(X_train, X_test, y_train):
	clf_gini = DecisionTreeClassifier(criterion="gini", random_state=100, max_depth=3,
min_samples_leaf=4)
	clf_gini.fit(X_train, y_train)
	return clf_gini

def tarin_using_entropy(X_train, X_test, y_train):
```

```
        clf_entropy = DecisionTreeClassifier(criterion="entropy", random_state=100,
max_depth=3, min_samples_leaf=4)
        clf_entropy.fit(X_train, y_train)
        return clf_entropy

def prediction(X_test, clf_object):
        y_pred = clf_object.predict(X_test)
        print("Predicted values:")
        print(y_pred)
        return y_pred

def cal_accuracy(y_test, y_pred):
    print("Confusion Matrix: ", confusion_matrix(y_test, y_pred))
    print("Accuracy: ", accuracy_score(y_test, y_pred) * 100)
    print("Report: ", classification_report(y_test, y_pred))

def main():
        data = importdata()
        X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
        clf_gini = train_using_gini(X_train, X_test, y_train)
        clf_entropy = tarin_using_entropy(X_train, X_test, y_train)
        print("Results Using Gini Index:")
        y_pred_gini = prediction(X_test, clf_gini)
        cal_accuracy(y_test, y_pred_gini)
        print("Results Using Entropy:")
        y_pred_entropy = prediction(X_test, clf_entropy)
        cal_accuracy(y_test, y_pred_entropy)

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

3)MULTI LAYER FEEDNETWORK

```
#!/usr/bin/env python
# coding: utf-8

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

data = pd.read_csv("/content/breast cancer_week 8 and 9.csv")
del data['Unnamed: 32']
```

```
X = data.iloc[:, 2:].values
y = data.iloc[:, 1].values

from sklearn.preprocessing import LabelEncoder
labelencoder_X_1 = LabelEncoder()
y = labelencoder_X_1.fit_transform(y)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout

classifier = Sequential()

classifier.add(Dense(activation="relu", input_dim=30, units=16, kernel_initializer="uniform"))
classifier.add(Dropout(rate=0.1))

classifier.add(Dense(activation="relu", input_dim=30, units=16, kernel_initializer="uniform"))
classifier.add(Dropout(rate=0.1))

classifier.add(Dense(activation="sigmoid", units=1, kernel_initializer="uniform"))

classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

classifier.fit(X_train, y_train, batch_size=100, epochs=150)
```

4)SVM

```
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
svm.fit(X, y)
```

```
DecisionBoundaryDisplay.from_estimator(
    svm,
    X,
    response_method="predict",
    cmap=plt.cm.Spectral,
    alpha=0.8,
    xlabel=cancer.feature_names[0],
    ylabel=cancer.feature_names[1],
)

plt.scatter(X[:, 0], X[:, 1],
            c=y,
            s=20, edgecolors="k")
plt.show()
```

5)RANDOM FOREST

```
#!/usr/bin/env python
# coding: utf-8

from sklearn.datasets import load_iris
iris = load_iris()

print("\n #Feature Names: \n", "\n", iris.feature_names, "\n", "\n Iris Data:", "\n", iris.data[:30,:], "\n",
"\n TargetNames:", "\n", iris.target_names, "\n", "\n Target:", "\n", iris.target)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size = .30)

from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=100)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result2 = accuracy_score(y_test,y_pred)
print("\n Accuracy: \n",result2)

CNF_MATRIX=confusion_matrix(y_test, y_pred)
print("\n CNFUSION_MATRIX: \n",CNF_MATRIX)

print("\n", classification_report(y_pred,y_test))
```

6)k means

```
#!/usr/bin/env python
# coding: utf-8

from numpy import where
from sklearn.datasets import make_classification
from matplotlib import pyplot

X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0,
n_clusters_per_class=1, random_state=4)
print(X)
print(y)
print("\n",X.shape)
print("\n",y.shape)


print(X)


print(y)


for class_value in range(2):
    row_ix = where(y == class_value)
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
pyplot.show()


from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from matplotlib import pyplot

X, y= make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0,
n_clusters_per_class=1, random_state=4)

model = KMeans(n_clusters=2)
model.fit(X)
yhat = model.predict(X)
clusters = unique(yhat)

for cluster in clusters:
    row_ix = where(yhat == cluster)
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
pyplot.show()
```

7)LINEAR REGRESSION

```
#!/usr/bin/env python
# coding: utf-8

from sklearn.datasets import load_boston
boston = load_boston()

print(boston.data.shape)

print(boston.feature_names, "\n")

print(boston.target, "\n", "\n")

print(boston.DESCR)

import pandas as pd
bos = pd.DataFrame(boston.data)
print(bos.head(), "\n")

bos['PRICE'] = boston.target

X = bos.drop('PRICE', axis=1)
Y = bos['PRICE']

import sklearn
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)

from sklearn.linear_model import LinearRegression
from matplotlib import pyplot as plt

lm = LinearRegression()
lm.fit(X_train, Y_train)

Y_pred = lm.predict(X_test)

plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
plt.show()
```

```
delta_y = Y_test - Y_pred

import seaborn as sns
import numpy as np
sns.set_style('whitegrid')
sns.kdeplot(np.array(delta_y), bw_method=0.05)
plt.show()

sns.set_style('whitegrid')
sns.kdeplot(np.array(Y_pred), bw_method=0.05)
plt.show()
```

8)LOGISTIC REGRESSION

```
#!/usr/bin/env python
# coding: utf-8

import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
import seaborn as sns

data = load_breast_cancer()
print(data.keys())

print(data.feature_names, "\n")

data.target[0:20]
print(data.target[0:20])

print(data.target_names)

df = pd.DataFrame(data.data, columns=data.feature_names)
print(df.size)

print(df.shape)

print(df.head(3))

df['Target'] = data.target
print("\n", "\n", "\n", df['Target'])

print(df.head())

print(df.info())

print(df.dtypes)
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y)

LR = LogisticRegression()

LR.fit(X_train, y_train)

print(LR.score(X_test, y_test))

yhat = LR.predict(X_test)
print(yhat)

from sklearn.metrics import classification_report, confusion_matrix
CM = metrics.confusion_matrix(y_test, yhat, labels=[0, 1])
print(CM)

plt.figure(figsize=(6,6))
sns.heatmap(CM, annot=True)
plt.show()

print(classification_report(y_test, yhat))
```

9)BACK PROPOGATION

```
#!/usr/bin/env python
# coding: utf-8

import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

data = load_iris()

X = data.data
y = data.target

y = pd.get_dummies(y).values
y[:3]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)

learning_rate = 0.1
iterations = 5000
N = y_train.size

input_size = 4
hidden_size = 2
output_size = 3

results_df = pd.DataFrame(columns=["mse", "accuracy"])

np.random.seed(10)

W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))
W2 = np.random.normal(scale=0.5, size=(hidden_size, output_size))

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def mean_squared_error(y_pred, y_true):
    return ((y_pred - y_true)**2).sum() / (2 * y_pred.size)

def accuracy(y_pred, y_true):
    acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
    return acc.mean()

for itr in range(iterations):
    Z1 = np.dot(X_train, W1)
    A1 = sigmoid(Z1)

    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)

    mse = mean_squared_error(A2, y_train)
    acc = accuracy(A2, y_train)
    results_df = pd.concat([results_df, pd.DataFrame({"mse": mse, "accuracy": acc}, index=[0])],
ignore_index=True)

    E1 = A2 - y_train
    dW1 = E1 * A2 * (1 - A2)

    E2 = np.dot(dW1, W2.T)
    dW2 = E2 * A1 * (1 - A1)

    W2_update = np.dot(A1.T, dW1) / N
```

```
    W1_update = np.dot(X_train.T, dW2) / N

    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update

results_df.mse.plot(title="Mean Squared Error")
plt.show()

results_df.accuracy.plot(title="Accuracy")
plt.show()

Z1 = np.dot(X_test, W1)
A1 = sigmoid(Z1)

Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)

acc = accuracy(A2, y_test)
print("Accuracy: {}".format(acc))
```