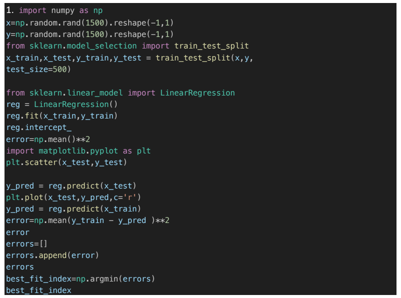
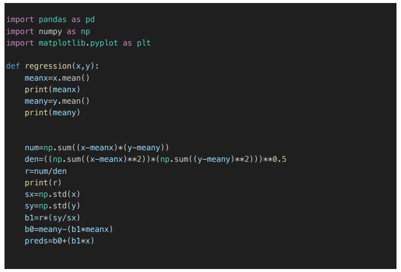
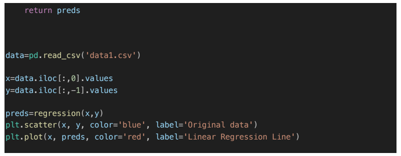


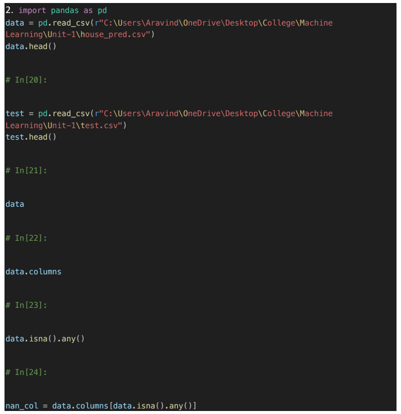
```
1. import numpy as np
x=np.random.rand(1500).reshape(-1,1)
y=np.random.rand(1500).reshape(-1,1)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,
test_size=500)

from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(x_train,y_train)
reg.intercept_
error=np.mean()**2
import matplotlib.pyplot as plt
plt.scatter(x_test,y_test)

y_pred = reg.predict(x_test)
plt.plot(x_test,y_pred,c='r')
y_pred = reg.predict(x_train)
error=np.mean(y_train - y_pred )**2
error
errors=[]
errors.append(error)
errors
best_fit_index=np.argmin(errors)
best_fit_index
```



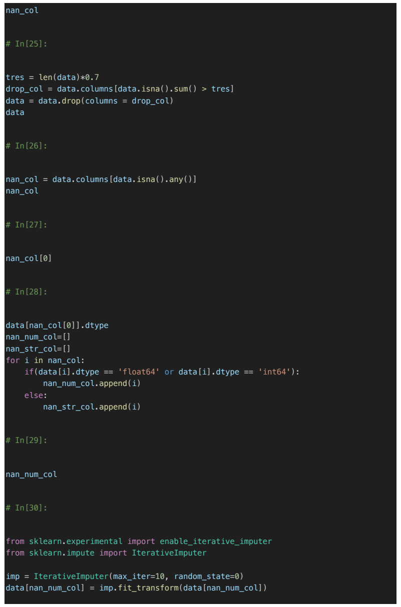
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def regression(x,y):
    meanx=x.mean()
    print(meanx)
    meany=y.mean()
    print(meany)


    num=np.sum((x-meanx)*(y-meany))
    den=((np.sum((x-meanx)**2))*(np.sum((y-meany)**2)))**0.5
    r=num/den
    print(r)
    sx=np.std(x)
    sy=np.std(y)
    b1=r*(sy/sx)
    b0=meany-(b1*meanx)
    preds=b0+(b1*x)
```



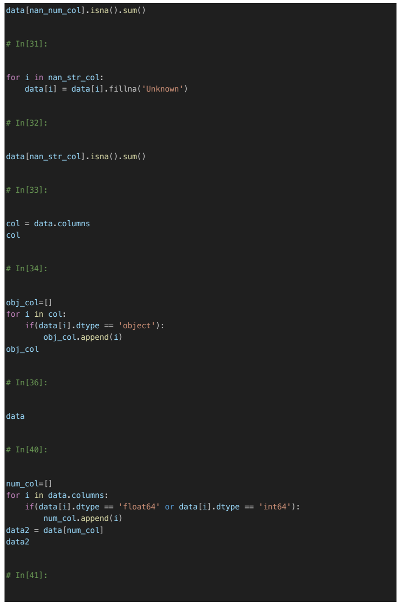
```
    return preds



data=pd.read_csv('data1.csv')

x=data.iloc[:,0].values
y=data.iloc[:,-1].values

preds=regression(x,y)
plt.scatter(x, y, color='blue', label='Original data')
plt.plot(x, preds, color='red', label='Linear Regression Line')
```



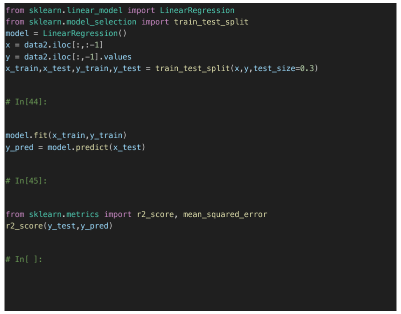
```
2. import pandas as pd
data = pd.read_csv(r"C:\Users\Aravind\OneDrive\Desktop\College\Machine
Learning\Unit-1\house_pred.csv")
data.head()


# In[20]:


test = pd.read_csv(r"C:\Users\Aravind\OneDrive\Desktop\College\Machine
Learning\Unit-1\test.csv")
test.head()


# In[21]:


data


# In[22]:


data.columns


# In[23]:


data.isna().any()


# In[24]:


nan_col = data.columns[data.isna().any()]
```



```
nan_col


# In[25]:


tres = len(data)*0.7
drop_col = data.columns[data.isna().sum() > tres]
data = data.drop(columns = drop_col)
data


# In[26]:


nan_col = data.columns[data.isna().any()]
nan_col


# In[27]:


nan_col[0]


# In[28]:


data[nan_col[0]].dtype
nan_num_col=[]
nan_str_col=[]
for i in nan_col:
    if(data[i].dtype == 'float64' or data[i].dtype == 'int64'):
        nan_num_col.append(i)
    else:
        nan_str_col.append(i)


# In[29]:


nan_num_col


# In[30]:


from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

imp = IterativeImputer(max_iter=10, random_state=0)
data[nan_num_col] = imp.fit_transform(data[nan_num_col])
```

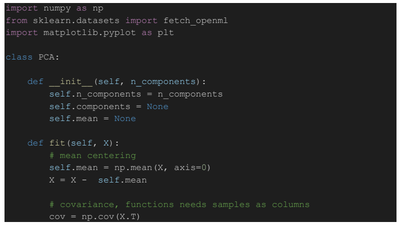


```
data[nan_num_col].isna().sum()


# In[31]:


for i in nan_str_col:
    data[i] = data[i].fillna('Unknown')


# In[32]:


data[nan_str_col].isna().sum()


# In[33]:


col = data.columns
col


# In[34]:


obj_col=[]
for i in col:
    if(data[i].dtype == 'object'):
        obj_col.append(i)
obj_col


# In[36]:


data


# In[40]:


num_col=[]
for i in data.columns:
    if(data[i].dtype == 'float64' or data[i].dtype == 'int64'):
        num_col.append(i)
data2 = data[num_col]
data2


# In[41]:
```

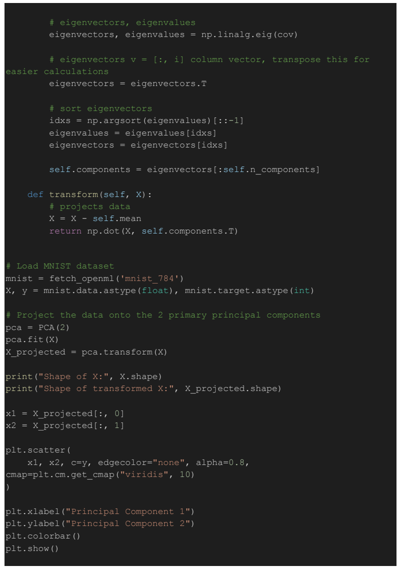


```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
model = LinearRegression()
x = data2.iloc[:,:-1]
y = data2.iloc[:,-1].values
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)


# In[44]:


model.fit(x_train,y_train)
y_pred = model.predict(x_test)


# In[45]:


from sklearn.metrics import r2_score, mean_squared_error
r2_score(y_test,y_pred)


# In[ ]:
```

PCA

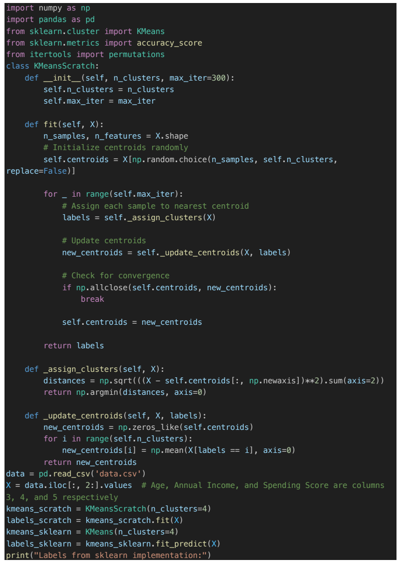


```
import numpy as np
from sklearn.datasets import fetch_openml
import matplotlib.pyplot as plt

class PCA:

    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.mean = None

    def fit(self, X):
        # mean centering
        self.mean = np.mean(X, axis=0)
        X = X -  self.mean

        # covariance, functions needs samples as columns
        cov = np.cov(X.T)
```

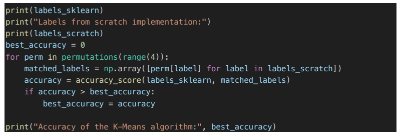


```
        # eigenvectors, eigenvalues
        eigenvectors, eigenvalues = np.linalg.eig(cov)

        # eigenvectors v = [:, i] column vector, transpose this for
easier calculations
        eigenvectors = eigenvectors.T

        # sort eigenvectors
        idxs = np.argsort(eigenvalues)[::-1]
        eigenvalues = eigenvalues[idxs]
        eigenvectors = eigenvectors[idxs]

        self.components = eigenvectors[:self.n_components]

    def transform(self, X):
        # projects data
        X = X - self.mean
        return np.dot(X, self.components.T)


# Load MNIST dataset
mnist = fetch_openml('mnist_784')
X, y = mnist.data.astype(float), mnist.target.astype(int)

# Project the data onto the 2 primary principal components
pca = PCA(2)
pca.fit(X)
X_projected = pca.transform(X)

print("Shape of X:", X.shape)
print("Shape of transformed X:", X_projected.shape)

x1 = X_projected[:, 0]
x2 = X_projected[:, 1]

plt.scatter(
    x1, x2, c=y, edgecolor="none", alpha=0.8,
cmap=plt.cm.get_cmap("viridis", 10)
)

plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar()
plt.show()
```

KMEAN

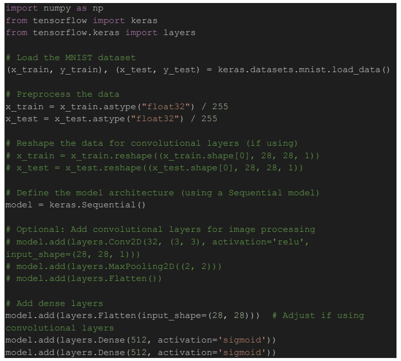


```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
from itertools import permutations
class KMeansScratch:
    def __init__(self, n_clusters, max_iter=300):
        self.n_clusters = n_clusters
        self.max_iter = max_iter

    def fit(self, X):
        n_samples, n_features = X.shape
        # Initialize centroids randomly
        self.centroids = X[np.random.choice(n_samples, self.n_clusters,
replace=False)]

        for _ in range(self.max_iter):
            # Assign each sample to nearest centroid
            labels = self._assign_clusters(X)

            # Update centroids
            new_centroids = self._update_centroids(X, labels)

            # Check for convergence
            if np.allclose(self.centroids, new_centroids):
                break

            self.centroids = new_centroids

        return labels

    def _assign_clusters(self, X):
        distances = np.sqrt(((X - self.centroids[:, np.newaxis])**2).sum(axis=2))
        return np.argmin(distances, axis=0)

    def _update_centroids(self, X, labels):
        new_centroids = np.zeros_like(self.centroids)
        for i in range(self.n_clusters):
            new_centroids[i] = np.mean(X[labels == i], axis=0)
        return new_centroids
data = pd.read_csv('data.csv')
X = data.iloc[:, 2:].values  # Age, Annual Income, and Spending Score are columns
3, 4, and 5 respectively
kmeans_scratch = KMeansScratch(n_clusters=4)
labels_scratch = kmeans_scratch.fit(X)
kmeans_sklearn = KMeans(n_clusters=4)
labels_sklearn = kmeans_sklearn.fit_predict(X)
print("Labels from sklearn implementation:")
```

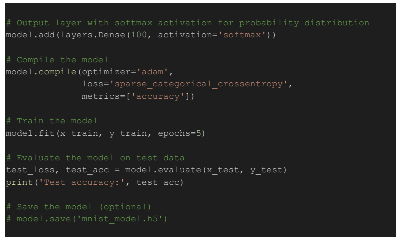


```
print(labels_sklearn)
print("Labels from scratch implementation:")
print(labels_scratch)
best_accuracy = 0
for perm in permutations(range(4)):
    matched_labels = np.array([perm[label] for label in labels_scratch])
    accuracy = accuracy_score(labels_sklearn, matched_labels)
    if accuracy > best_accuracy:
        best_accuracy = accuracy

print("Accuracy of the K-Means algorithm:", best_accuracy)
```

Neural network

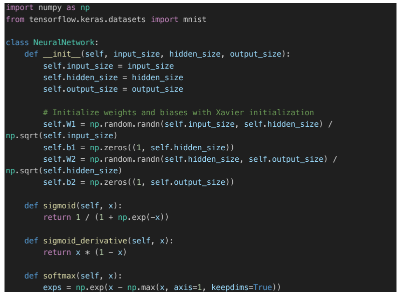


```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Preprocess the data
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

# Reshape the data for convolutional layers (if using)
# x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
# x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))

# Define the model architecture (using a Sequential model)
model = keras.Sequential()

# Optional: Add convolutional layers for image processing
# model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)))
# model.add(layers.MaxPooling2D((2, 2)))
# model.add(layers.Flatten())

# Add dense layers
model.add(layers.Flatten(input_shape=(28, 28)))  # Adjust if using
convolutional layers
model.add(layers.Dense(512, activation='sigmoid'))
model.add(layers.Dense(512, activation='sigmoid'))
```

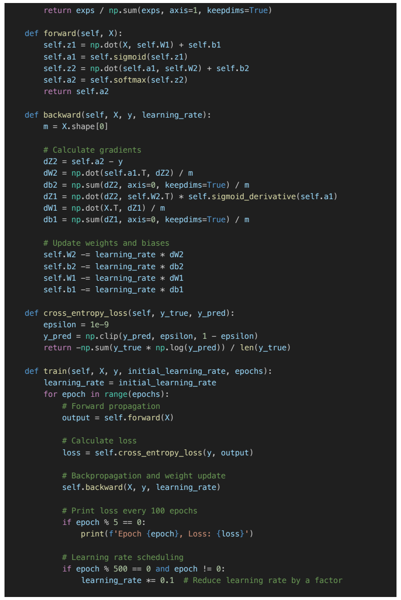


```

# Output layer with softmax activation for probability distribution
model.add(layers.Dense(100, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=5)

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)

# Save the model (optional)
# model.save('mnist_model.h5')
```

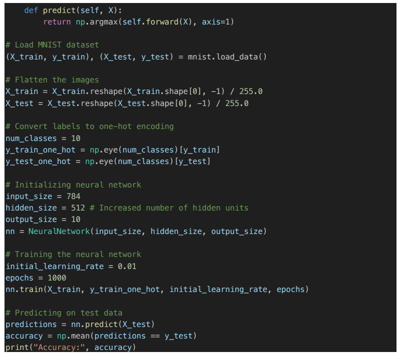
OWN



```
import numpy as np
from tensorflow.keras.datasets import mnist

class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # Initialize weights and biases with Xavier initialization
        self.W1 = np.random.randn(self.input_size, self.hidden_size) /
np.sqrt(self.input_size)
        self.b1 = np.zeros((1, self.hidden_size))
        self.W2 = np.random.randn(self.hidden_size, self.output_size) /
np.sqrt(self.hidden_size)
        self.b2 = np.zeros((1, self.output_size))

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

    def sigmoid_derivative(self, x):
        return x * (1 - x)

    def softmax(self, x):
        exps = np.exp(x - np.max(x, axis=1, keepdims=True))
```

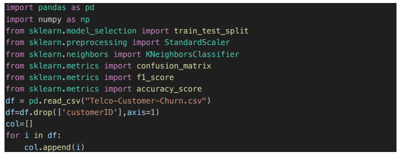


```
        return exps / np.sum(exps, axis=1, keepdims=True)

    def forward(self, X):
        self.z1 = np.dot(X, self.W1) + self.b1
        self.a1 = self.sigmoid(self.z1)
        self.z2 = np.dot(self.a1, self.W2) + self.b2
        self.a2 = self.softmax(self.z2)
        return self.a2

    def backward(self, X, y, learning_rate):
        m = X.shape[0]

        # Calculate gradients
        dZ2 = self.a2 - y
        dW2 = np.dot(self.a1.T, dZ2) / m
        db2 = np.sum(dZ2, axis=0, keepdims=True) / m
        dZ1 = np.dot(dZ2, self.W2.T) * self.sigmoid_derivative(self.a1)
        dW1 = np.dot(X.T, dZ1) / m
        db1 = np.sum(dZ1, axis=0, keepdims=True) / m

        # Update weights and biases
        self.W2 -= learning_rate * dW2
        self.b2 -= learning_rate * db2
        self.W1 -= learning_rate * dW1
        self.b1 -= learning_rate * db1

    def cross_entropy_loss(self, y_true, y_pred):
        epsilon = 1e-9
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
        return -np.sum(y_true * np.log(y_pred)) / len(y_true)

    def train(self, X, y, initial_learning_rate, epochs):
        learning_rate = initial_learning_rate
        for epoch in range(epochs):
            # Forward propagation
            output = self.forward(X)

            # Calculate loss
            loss = self.cross_entropy_loss(y, output)

            # Backpropagation and weight update
            self.backward(X, y, learning_rate)

            # Print loss every 100 epochs
            if epoch % 5 == 0:
                print(f'Epoch {epoch}, Loss: {loss}')

            # Learning rate scheduling
            if epoch % 500 == 0 and epoch != 0:
                learning_rate *= 0.1  # Reduce learning rate by a factor
```

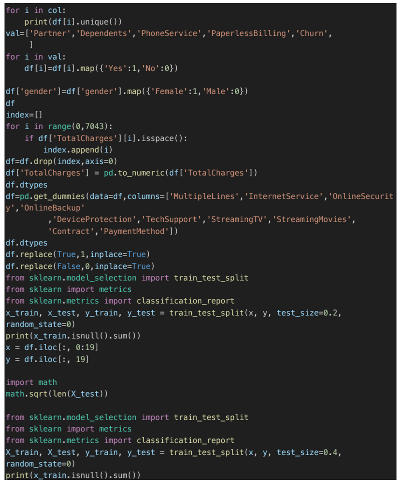


```
    def predict(self, X):
        return np.argmax(self.forward(X), axis=1)

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Flatten the images
X_train = X_train.reshape(X_train.shape[0], -1) / 255.0
X_test = X_test.reshape(X_test.shape[0], -1) / 255.0

# Convert labels to one-hot encoding
num_classes = 10
y_train_one_hot = np.eye(num_classes)[y_train]
y_test_one_hot = np.eye(num_classes)[y_test]

# Initializing neural network
input_size = 784
hidden_size = 512 # Increased number of hidden units
output_size = 10
nn = NeuralNetwork(input_size, hidden_size, output_size)

# Training the neural network
initial_learning_rate = 0.01
epochs = 1000
nn.train(X_train, y_train_one_hot, initial_learning_rate, epochs)

# Predicting on test data
predictions = nn.predict(X_test)
accuracy = np.mean(predictions == y_test)
print("Accuracy:", accuracy)
```

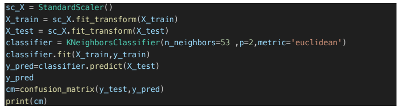
Ex:5



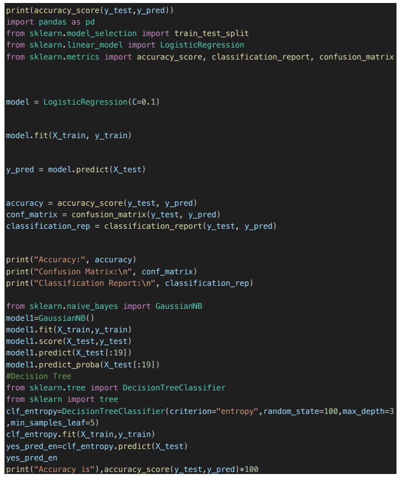
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
df = pd.read_csv("Telco-Customer-Churn.csv")
df=df.drop(['customerID'],axis=1)
col=[]
for i in df:
    col.append(i)
```



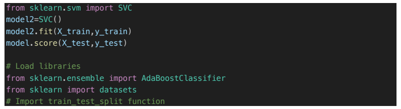
```
for i in col:
    print(df[i].unique())
val=['Partner','Dependents','PhoneService','PaperlessBilling','Churn',
     ]
for i in val:
    df[i]=df[i].map({'Yes':1,'No':0})

df['gender']=df['gender'].map({'Female':1,'Male':0})
df
index=[]
for i in range(0,7043):
    if df['TotalCharges'][i].isspace():
        index.append(i)
df=df.drop(index,axis=0)
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'])
df.dtypes
df=pd.get_dummies(data=df,columns=['MultipleLines','InternetService','OnlineSecurit
y','OnlineBackup'
         ,'DeviceProtection','TechSupport','StreamingTV','StreamingMovies',
         'Contract','PaymentMethod'])
df.dtypes
df.replace(True,1,inplace=True)
df.replace(False,0,inplace=True)
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=0)
print(x_train.isnull().sum())
x = df.iloc[:, 0:19]
y = df.iloc[:, 19]

import math
math.sqrt(len(X_test))

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.4,
random_state=0)
print(x_train.isnull().sum())
```



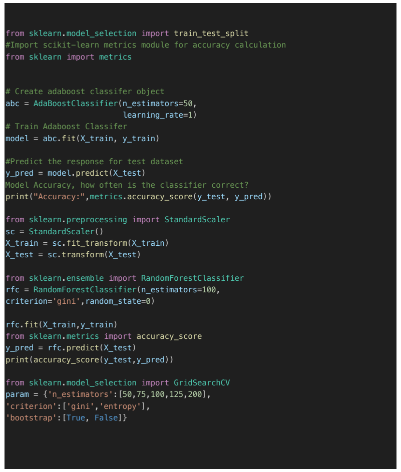
```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
classifier = KNeighborsClassifier(n_neighbors=53 ,p=2,metric='euclidean')
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
y_pred
cm=confusion_matrix(y_test,y_pred)
print(cm)
```



```
print(accuracy_score(y_test,y_pred))
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix



model = LogisticRegression(C=0.1)


model.fit(X_train, y_train)


y_pred = model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)


print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)

from sklearn.naive_bayes import GaussianNB
model1=GaussianNB()
model1.fit(X_train,y_train)
model1.score(X_test,y_test)
model1.predict(X_test[:19])
model1.predict_proba(X_test[:19])
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
clf_entropy=DecisionTreeClassifier(criterion="entropy",random_state=100,max_depth=3
,min_samples_leaf=5)
clf_entropy.fit(X_train,y_train)
yes_pred_en=clf_entropy.predict(X_test)
yes_pred_en
print("Accuracy is"),accuracy_score(y_test,y_pred)*100
```

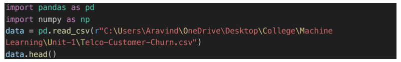


```
from sklearn.svm import SVC
model2=SVC()
model2.fit(X_train,y_train)
model.score(X_test,y_test)

# Load libraries
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets
# Import train_test_split function
```

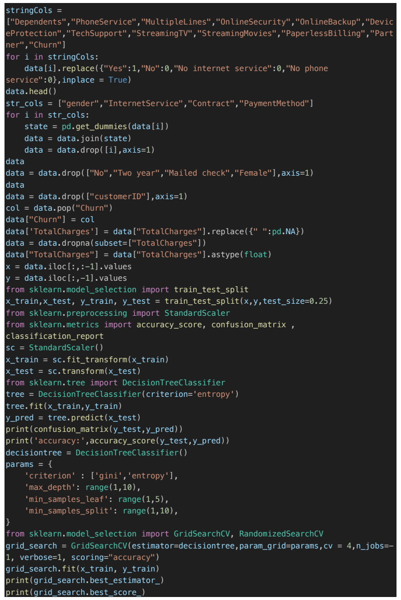


```
from sklearn.model_selection import train_test_split
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics


# Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=50,
                         learning_rate=1)
# Train Adaboost Classifer
model = abc.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = model.predict(X_test)
Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

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

from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=100,
criterion='gini',random_state=0)

rfc.fit(X_train,y_train)
from sklearn.metrics import accuracy_score
y_pred = rfc.predict(X_test)
print(accuracy_score(y_test,y_pred))

from sklearn.model_selection import GridSearchCV
param = {'n_estimators':[50,75,100,125,200],
'criterion':['gini','entropy'],
'bootstrap':[True, False]}
```

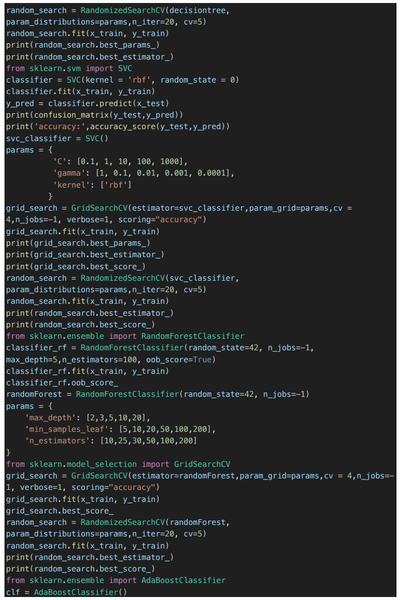
EX:6



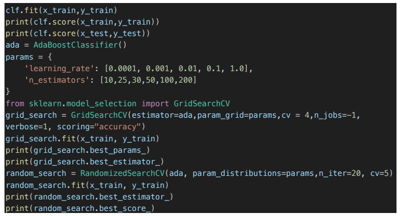
```
import pandas as pd
import numpy as np
data = pd.read_csv(r"C:\Users\Aravind\OneDrive\Desktop\College\Machine
Learning\Unit-1\Telco-Customer-Churn.csv")
data.head()
```



```
stringCols =
["Dependents","PhoneService","MultipleLines","OnlineSecurity","OnlineBackup","Devic
eProtection","TechSupport","StreamingTV","StreamingMovies","PaperlessBilling","Part
ner","Churn"]
for i in stringCols:
    data[i].replace({"Yes":1,"No":0,"No internet service":0,"No phone
service":0},inplace = True)
data.head()
str_cols = ["gender","InternetService","Contract","PaymentMethod"]
for i in str_cols:
    state = pd.get_dummies(data[i])
    data = data.join(state)
    data = data.drop([i],axis=1)
data
data = data.drop(["No","Two year","Mailed check","Female"],axis=1)
data
data = data.drop(["customerID"],axis=1)
col = data.pop("Churn")
data["Churn"] = col
data['TotalCharges'] = data["TotalCharges"].replace({" ":pd.NA})
data = data.dropna(subset=["TotalCharges"])
data["TotalCharges"] = data["TotalCharges"].astype(float)
x = data.iloc[:,:-1].values
y = data.iloc[:,-1].values
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)
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix ,
classification_report
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(criterion='entropy')
tree.fit(x_train,y_train)
y_pred = tree.predict(x_test)
print(confusion_matrix(y_test,y_pred))
print('accuracy:',accuracy_score(y_test,y_pred))
decisiontree = DecisionTreeClassifier()
params = {
    'criterion' : ['gini','entropy'],
    'max_depth': range(1,10),
    'min_samples_leaf': range(1,5),
    'min_samples_split': range(1,10),
}
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
grid_search = GridSearchCV(estimator=decisiontree,param_grid=params,cv = 4,n_jobs=-
1, verbose=1, scoring="accuracy")
grid_search.fit(x_train, y_train)
print(grid_search.best_estimator_)
print(grid_search.best_score_)
```



```
random_search = RandomizedSearchCV(decisiontree,
param_distributions=params,n_iter=20, cv=5)
random_search.fit(x_train, y_train)
print(random_search.best_params_)
print(random_search.best_estimator_)
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print(confusion_matrix(y_test,y_pred))
print('accuracy:',accuracy_score(y_test,y_pred))
svc_classifier = SVC()
params = {
          'C': [0.1, 1, 10, 100, 1000],
          'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
          'kernel': ['rbf']
         }
grid_search = GridSearchCV(estimator=svc_classifier,param_grid=params,cv =
4,n_jobs=-1, verbose=1, scoring="accuracy")
grid_search.fit(x_train, y_train)
print(grid_search.best_params_)
print(grid_search.best_estimator_)
print(grid_search.best_score_)
random_search = RandomizedSearchCV(svc_classifier,
param_distributions=params,n_iter=20, cv=5)
random_search.fit(x_train, y_train)
print(random_search.best_estimator_)
print(random_search.best_score_)
from sklearn.ensemble import RandomForestClassifier
classifier_rf = RandomForestClassifier(random_state=42, n_jobs=-1,
max_depth=5,n_estimators=100, oob_score=True)
classifier_rf.fit(x_train, y_train)
classifier_rf.oob_score_
randomForest = RandomForestClassifier(random_state=42, n_jobs=-1)
params = {
    'max_depth': [2,3,5,10,20],
    'min_samples_leaf': [5,10,20,50,100,200],
    'n_estimators': [10,25,30,50,100,200]
}
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(estimator=randomForest,param_grid=params,cv = 4,n_jobs=-
1, verbose=1, scoring="accuracy")
grid_search.fit(x_train, y_train)
grid_search.best_score_
random_search = RandomizedSearchCV(randomForest,
param_distributions=params,n_iter=20, cv=5)
random_search.fit(x_train, y_train)
print(random_search.best_estimator_)
print(random_search.best_score_)
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier()
```



```
clf.fit(x_train,y_train)
print(clf.score(x_train,y_train))
print(clf.score(x_test,y_test))
ada = AdaBoostClassifier()
params = {
    'learning_rate': [0.0001, 0.001, 0.01, 0.1, 1.0],
    'n_estimators': [10,25,30,50,100,200]
}
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(estimator=ada,param_grid=params,cv = 4,n_jobs=-1,
verbose=1, scoring="accuracy")
grid_search.fit(x_train, y_train)
print(grid_search.best_params_)
print(grid_search.best_estimator_)
random_search = RandomizedSearchCV(ada, param_distributions=params,n_iter=20, cv=5)
random_search.fit(x_train, y_train)
print(random_search.best_estimator_)
print(random_search.best_score_)
```