1-KNN(from 3 code)

```
# Step 1: Calculate Euclidean Distance.
# Step 2: Get Nearest Neighbors.
# Step 3: Make Predictions.
# Step 1: Calculate Euclidean Distance
# Example of calculating Euclidean distance
from math import sqrt
# calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1) - 1):
        distance += (row1[i] - row2[i]) ** 2
    return sqrt(distance)
# Step 2: Get Nearest Neighbors
# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors
# Step 3: Make Predictions
# Make a classification prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
```

```
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction
# Test distance function
dataset = [[2.7810836, 2.550537003, 0],
           [1.465489372, 2.362125076, 0],
           [3.396561688, 4.400293529, 0],
           [1.38807019, 1.850220317, 0],
           [3.06407232, 3.005305973, 0],
           [7.627531214, 2.759262235, 1],
           [5.332441248, 2.088626775, 1],
           [6.922596716, 1.77106367, 1],
           [8.675418651, -0.242068655, 1],
           [7.673756466, 3.508563011, 1]]
# 1
row0 = dataset[0]
for row in dataset:
    distance = euclidean_distance(row0, row)
    print(distance)
# 2
neighbors = get_neighbors(dataset, dataset[0], 3)
for neighbor in neighbors:
    print(neighbor)
# 3
prediction = predict_classification(dataset, dataset[0], 3)
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```

2-LDA(from 3 code)

```
# # code for LDA
from sklearn.datasets import load_iris
import numpy as np
from collections import Counter
import seaborn as sn
import matplotlib.pyplot as plt
iris=load_iris()
x=iris.data
y=iris.target
label_num=Counter(y).keys()
class_mean=[]
for i in range(len(label_num)):
    class_mean.append(np.mean(x[y==i],axis=0))
SW=np.zeros((4,4)) # calculate SW
for i in range(len(class_mean)):
    SI=np.zeros((4,4))
    for row in x[y==i]:
        row,c_mean=row.reshape(4,1),class_mean[i].reshape(4,1)
        SI+=1/49*((row-c_mean).dot((row-c_mean).T))
    SW+=SI
# print(SW)
all_mean=np.mean(x,axis=0)
# print(all_mean)
SB=np.zeros((4,4)) # calculate SB
for i in range(len(class_mean)):
    c_mean,all_mean=class_mean[i].reshape(4,1),all_mean.reshape(4,1)
    SB+=(x[y==i].shape[0]*((c_mean-all_mean).dot((c_mean-all_mean).T)))
```

```
# print(SB)
eig_val,eig_vec=np.linalg.eig(np.linalg.inv(SW).dot(SB))
pairs=[[abs(eig_val[i]),eig_vec[:,i]] for i in range(len(eig_val))]
w1=pairs[0][1]
w2=pairs[1][1]
lda1=x.dot(w1)
lda2=x.dot(w2)
# visalise dataset after reduction
cmap_bold = ["darkorange", "c", "darkblue"]
sn.scatterplot(x=lda1,y=lda2,palette=cmap_bold,hue=iris.target_names[y])
plt.show()
```

3-PCA(from 3 code)

```
# #code fpr PCA
from sklearn.datasets import load_iris
import numpy as np
iris=load_iris()
x=iris.data
y=iris.target
mean=[]
# calculate mean from each column
for i in range(x.shape[1]):
    # print(np.mean(x[:,:i]))
    mean.append(np.mean(x[:,i]))
# subtract each column from mean
for i in range(x.shape[1]):
    x[:,i]=x[:,i]-mean[i]
# calculate coviriance matrix
```

```
C=(1/(x.shape[0]-1))*x.T.dot(x)

print(C.shape)

# obtain eign_value and eign_vectors for matrix

eign_value,eign_vectors= np.linalg.eig(C)

# multiply 2 eign_vectores of 2 maximum eign_values to obtain 2-d features from
4-d

pc1=x.dot(eign_vectors[:,0])

pc2=x.dot(eign_vectors[:,1])
```

1-weather(use KNN)

```
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','S
unny', 'Rainy','Sunny','Overcast','Overcast','Rainy'] # first Feature

temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','
Mild','Hot','Mild']# secon features

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','
No']  # Label or target varible

# Import LabelEncoder

from sklearn import preprocessing

#creating labelEncoder

le = preprocessing.LabelEncoder()

# Converting string labels into numbers.

weather_encoded=le.fit_transform(weather)

# converting string labels into numbers

temp_encoded=le.fit_transform(temp)

label=le.fit_transform(play)

#combinig weather and temp into single listof tuples

features=list(zip(weather_encoded,temp_encoded))

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=3)
```

```
# Train the model using the training sets
model.fit(features,label)
#Predict Output
predicted= model.predict(features) # 0:Overcast, 2:Mild
print(predicted)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# confusion matrix
matrix = confusion_matrix(predicted,predicted, labels=[0,1])
print('Confusion matrix : \n',matrix)
# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(predicted,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)
# classification report for precision, recall f1-score and accuracy
matrix = classification_report(predicted,predicted,labels=[1,0])
print('Classification report : \n',matrix)
```

2-bigmart

```
import pandas as pd
df = pd.read_csv('train.csv')
df.head()
print(df.isnull().sum())
mean = df['Item_Weight'].mean() #imputing item_weight with mean
df['Item_Weight'].fillna(mean, inplace =True)


mode = df['Outlet_Size'].mode() #imputing outlet size with mode
df['Outlet_Size'].fillna(mode[0], inplace =True)
df.drop(['Item_Identifier', 'Outlet_Identifier'], axis=1, inplace=True)
```

```
df = pd.get_dummies(df)
from sklearn.model_selection import train_test_split
train , test = train_test_split(df, test_size = 0.3)
x_train = train.drop('Item_Outlet_Sales', axis=1)
y_train = train['Item_Outlet_Sales']


x_test = test.drop('Item_Outlet_Sales', axis = 1)
y_test = test['Item_Outlet_Sales']
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))


x_train_scaled = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train_scaled)


x_test_scaled = scaler.fit_transform(x_test)
x_test = pd.DataFrame(x_test_scaled)
#import required packages
from sklearn import neighbors
model = neighbors.KNeighborsRegressor(n_neighbors = 5)
model.fit(x_train, y_train)
pred=model.predict(x_test)
```

3-bernoli

```
#importing necessary libraries
```

```
import numpy as np
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics  import accuracy_score
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import accuracy_score
#creating dataset with with 0 and 1
X = np.random.randint(2, size=(500,10))
Y = np.random.randint(2, size=(500, 1))
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)
clf = BernoulliNB()
model = clf.fit(X, Y)
y_pred =clf.predict(X_test)
acc_score= accuracy_score(y_test, y_pred)
print(acc_score)
```

4-irisanive

```
# load the iris dataset
from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=1)


# training the model on training set

from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X_train, y_train)


# making predictions on the testing set

y_pred = gnb.predict(X_test)


# comparing actual response values (y_test) with predicted response values
(y_pred)

from sklearn import metrics

print("Gaussian Naive Bayes model accuracy(in %):",
metrics.accuracy_score(y_test, y_pred)*100)
```

5-car

```
import numpy as np

import pandas as pd

from sklearn.naive_bayes import  CategoricalNB

from sklearn.preprocessing import OrdinalEncoder, LabelEncoder

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

sns.set()

car_df = pd.read_csv("car.data",
names=['buying','maint','doors','persons','lug_boot','safety','class'])
```

```
print(car_df.head())
print('=' * 30)
print(car_df.info())
print('=' * 30)
print(car_df.dtypes)
print('=' * 30)
print(car_df.isnull().sum())
print('=' * 30)
print( car_df.shape[0])
print('=' * 30)
#features = car_df.columns.tolist()
#print(list(enumerate(features)))
#for indx, var  in enumerate(features):
   # print(indx)
   # print(var)
def count_plot(df, columns):
    plt.figure(figsize=(15, 10))
    for indx, var  in enumerate(columns):# adds a counter returns an enumerate object (0,buying) (1,maint) (2,doors)
        plt.subplot(2, 3, indx+1)# number of rows ,number of coulumns,index of current plot
        g = sns.countplot(df[var], hue= df['class'])# show the counts of observations in each feature
    plt.tight_layout()
    plt.show()
features = car_df.columns.tolist()
features.remove('class')
print(features)
count_plot(car_df, features)
for var in features:
```

```
    print(car_df[var].value_counts().sort_values(ascending=False) /
car_df.shape[0])
    print('=' * 30)
    print()
encoder = OrdinalEncoder()
data_encoded = encoder.fit_transform(car_df[features])
car_df_encoded = pd.DataFrame(data_encoded, columns=features)
print(car_df_encoded .head())



encoder = LabelEncoder()
target_encoded = encoder.fit_transform(car_df['class'])
car_df_encoded['class'] = target_encoded
print(car_df_encoded['class'].head())
X_train, X_test, y_train, y_test = train_test_split(car_df_encoded.drop('class',
axis=1), car_df_encoded['class'], test_size=0.3)
cnb = CategoricalNB()
cnb.fit(X_train, y_train)
```

6-Kmeans

```
from sklearn.datasets import make_blobs
import pandas as pd
from sklearn.cluster import KMeans
dataset, classes = make_blobs(n_samples=200, n_features=2, centers=4,
cluster_std=0.5, random_state=0)
# make as panda dataframe for easy understanding
df = pd.DataFrame(dataset, columns=['var1', 'var2'])
print(df.head(2))
from yellowbrick.cluster import KElbowVisualizer
model = KMeans()
```

```
visualizer = KElbowVisualizer(model, k=(1,12)).fit(df)
visualizer.show()


kmeans = KMeans(n_clusters=4, init='k-means++', random_state=0).fit(df)
print(kmeans.labels_)
print('=' * 30)
print(kmeans.n_iter_)
print('=' * 30)
print(kmeans.cluster_centers_)
#Get each cluster size
from collections import Counter
print(Counter(kmeans.labels_))
print('=' * 30)
import seaborn as sns
import matplotlib.pyplot as plt
sns.scatterplot(data=df, x="var1", y="var2", hue=kmeans.labels_)
plt.show()
sns.scatterplot(data=df, x="var1", y="var2", hue=kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
marker="X", c="r", s=80, label="centroids")
plt.legend()
plt.show()
```

7-news(naïve)

```
from sklearn.datasets import fetch_20newsgroups
import seaborn as sns
import matplotlib.pyplot as plt
```

```
data = fetch_20newsgroups()

print(data.target_names)

categories = ['talk.religion.misc', 'soc.religion.christian','sci.space',
'comp.graphics']

train = fetch_20newsgroups(subset='train', categories=categories)

test = fetch_20newsgroups(subset='test', categories=categories)

#print(test.target)

#print('=' * 30)

#print(test.data)

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.pipeline import make_pipeline

model = make_pipeline(TfidfVectorizer(), MultinomialNB())

model.fit(train.data, train.target)# train

labels = model.predict(test.data)#test

print(labels)

from sklearn.metrics import confusion_matrix

mat = confusion_matrix(test.target, labels)

sns.heatmap(mat, square=True, annot=True, xticklabels=train.target_names,
yticklabels=train.target_names)

plt.xlabel('true label')

plt.ylabel('predicted label')

plt.show()

def predict_category(s, train=train, model=model):

    pred = model.predict([s])

    return train.target_names[pred[0]]

p=predict_category('sending a payload to the ISS')

print(p)
```

8-ida

```
import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn import datasets

from sklearn.svm import SVC

iris = datasets.load_iris()

X = iris.data

y = iris.target

# 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=0.2,
random_state=0)

from sklearn.preprocessing import StandardScaler



sc = StandardScaler()

X_train = sc.fit_transform(X_train)

print((X_train))

X_test = sc.transform(X_test)

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA



lda = LDA(n_components=1)

X_train = lda.fit_transform(X_train, y_train)

X_test = lda.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
```

```
classifier = SVC()


classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score


cm = confusion_matrix(y_test, y_pred)
print(cm)
```

9-pca

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target
# 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=0.2,
random_state=0)
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
print((X_train))
X_test = sc.transform(X_test)
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train = pca.fit_transform(X_train)
print(X_train)
explained_variance = pca.explained_variance_ratio_
print(explained_variance)
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA


lda = LDA(n_components=1)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
```

10-svm3

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from  sklearn.model_selection import GridSearchCV
bankdata = pd.read_csv("bill_authentication.csv")
bankdata.shape
bankdata.head()
X = bankdata.drop('Class', axis=1)
```

```
y = bankdata['Class']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
from sklearn.svm import SVC
"""svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))"""
grid = {
    'C':[0.01,0.1,1,10],
    'kernel' : ["linear","poly","rbf","sigmoid"],
    'degree' : [1,3,5,7],
    'gamma' : [0.01,1]
}
svm  = SVC ()
gridcv = GridSearchCV(SVC(), grid)




gridcv.fit(X_train,y_train)


# print best parameter after tuning
print(gridcv.best_params_)

# print how our model looks after hyper-parameter tuning
print(gridcv.best_estimator_)
```

11-KNN

```
# Import necessary modules

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

from sklearn.datasets import load_iris

# Loading data

irisData = load_iris()

# Create feature and target arrays

X = irisData.data

y = irisData.target

# Split into training and test set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state=42)

knn =KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before

print(knn.predict(X_test))
```

12-convoution

```
# confusion matrix in sklearn

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

# actual values

actual = [1,0,0,1,0,0,1,0,0,1]

# predicted values

predicted = [1,0,0,1,0,0,0,1,0,0]
```

```
# confusion matrix
matrix = confusion_matrix(actual,predicted, labels=[0,1])
print('Confusion matrix : \n',matrix)


# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)


# classification report for precision, recall f1-score and accuracy
matrix = classification_report(actual,predicted,labels=[1,0])
print('Classification report : \n',matrix)
```