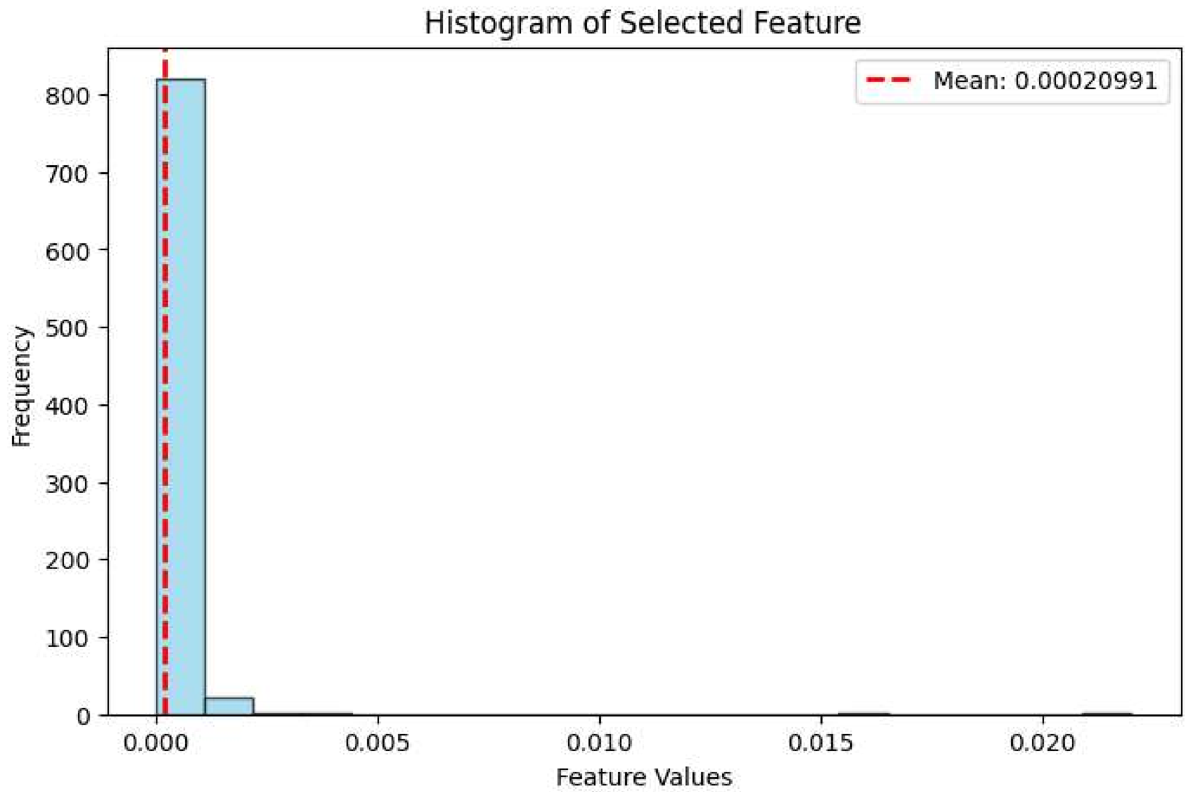
```
# 1
import numpy as np
import pandas as pd
df = pd.read_excel("output.xlsx")
selected_columns = df.iloc[:, :-1]
selected_label = df.iloc[:, -1]
data=selected_columns.to_numpy()
label=selected_label.to_numpy()
print(label)
```

```
[0 0 0 ... 0 0 0]
```

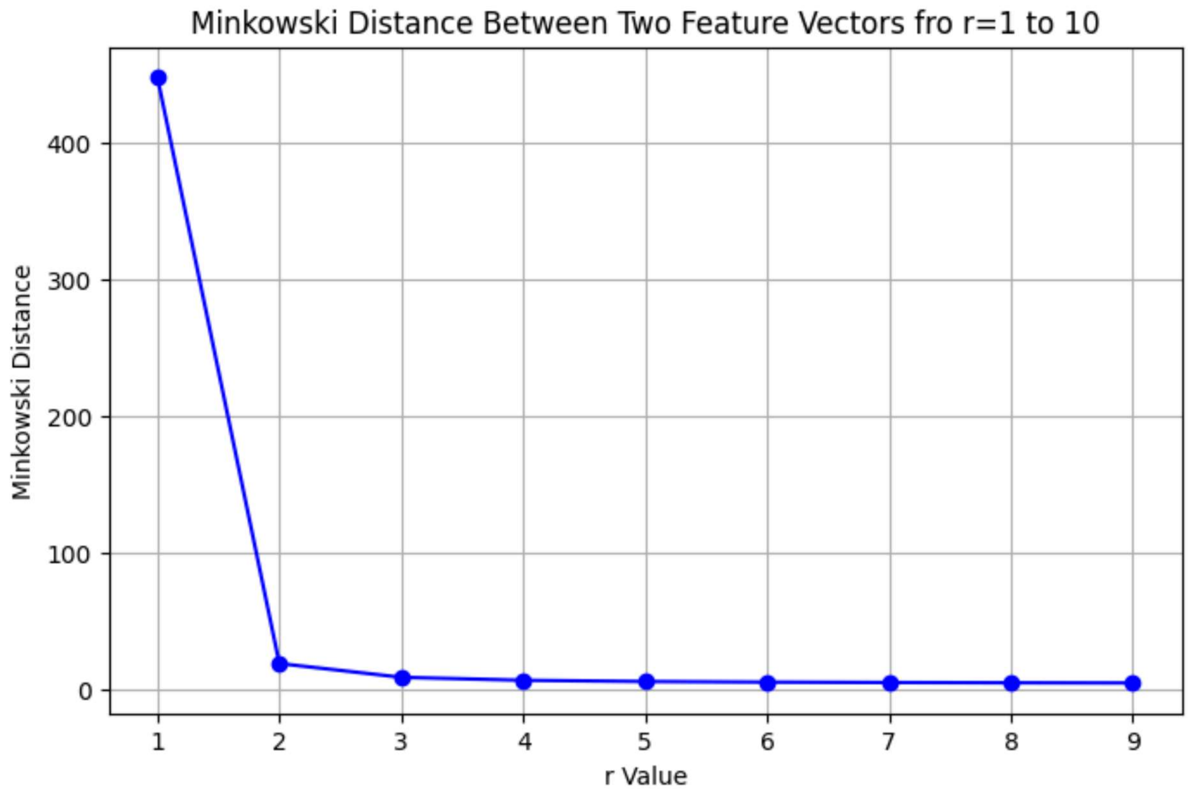
```
class1_data = data[selected_label == 0]
class2_data = data[selected_label == 1]
centroid1=class1_data.mean(axis=0)
centroid2=class2_data.mean(axis=0)
spread1=class1_data.std(axis=0)
spread2=class2_data.std(axis=0)
# print(class1_data)
meann=np.mean(data)
stdd=np.std(data)
print("mean of the data",meann)
print("centroids of the data",centroid1,centroid2)
print("std of data",stdd)
print("spread for the data",spread1,spread2)
print("euclidean dist between the 2 centroids",np.linalg.norm(centroid1 - centroid2))
```

```
mean of the data 0.1534347362439881
centroids of the data [0.00000000e+00 1.85889087e-04 4.63558214e-02 ... 1.82663417e+00
 7.24526712e-01 0.00000000e+00] [0.00000000e+00 2.40055960e-04 3.83625459e-02 ... 1.50626506e+00
 5.91363946e-01 0.00000000e+00]
std of data 0.5576383919594899
spread for the data [0.00000000e+00 3.94259750e-04 4.64719605e-02 ... 5.62379387e-01
 3.02725773e-01 0.00000000e+00] [0.         0.00143054 0.04717773 ... 0.50773967 0.24662965 0.        ]
euclidean dist between the 2 centroids 7.209585037535181
```

```
# 2
import matplotlib.pyplot as plt
feature_column=df.iloc[:, 1]
feature_data=feature_column.to_numpy()
# print(feature_column)
meanf=np.mean(feature_data)
varf=np.var(feature_data)
plt.figure(figsize=(8,5))
plt.hist(feature_data, bins=20, color='skyblue', edgecolor='black', alpha=0.7)
plt.axvline(meanf, color='red', linestyle='dashed', linewidth=2, label=f"Mean: {meanf:.8f}")
plt.xlabel("Feature Values")
plt.ylabel("Frequency")
plt.title("Histogram of Selected Feature")
plt.legend()
plt.show()
```




```
# 3
def minkowski_dist(a,b,r):
  return np.sum(np.abs(a-b)**r)**(1/r)
fv1=df.iloc[1,:-1].to_numpy()
fv2=df.iloc[2,:-1].to_numpy()
dist=[]
r_values=[]
for i in range(1,10):
  dist.append(minkowski_dist(fv1,fv2,i))
  r_values.append(i)

plt.figure(figsize=(8, 5))
plt.plot(r_values, dist, marker='o', linestyle='-', color='b')
plt.xlabel("r Value")
plt.ylabel("Minkowski Distance")
plt.title("Minkowski Distance Between Two Feature Vectors fro r=1 to 10")
plt.grid(True)
plt.show()
```




```
# 4
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, label, test_size=0.3)

# print(y_train,y_test)
```

```
# 5
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(data,label)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
# 6
print(neigh.score(X_test, y_test))
```

```
0.8862745098039215
```

```
# 7
print(neigh.predict(X_test))
print(y_test==neigh.predict(X_test))
```

```
[0 0 1 0 0 1 1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 0 0 1 1 1 0 0 1 0 0 1 1 0 0 1
 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 1 1 1 0 0 0 1 1 1 0 1 1 0
 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 0 0 0 0 0 0 0 1 0
 1 0 1 0 1 0 1 0 1 1 0 1 1 0 1 1 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1
 1 0 0 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 0 1 1 0 0 1 0 0 0 1 1 1 1 0 0 0 1 0
```

```
 1 0 0 1 0 1 1 1 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 1
 0 1 0 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 1]
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True False  True  True
  True  True  True  True  True  True  True  True  True  True False  True
  True  True  True  True  True  True  True  True False False  True  True
  True  True  True False  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True False False  True False
  True  True  True  True  True  True  True  True  True  True  True  True
 False  True  True  True False  True  True  True  True  True  True  True
  True False  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True False
  True  True  True  True  True False  True False  True False  True  True
  True  True  True  True  True  True  True  True  True False  True  True
 False  True  True  True  True  True  True False  True  True  True  True
  True False  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True False  True  True  True
  True  True  True  True  True False  True False  True  True  True  True
  True  True  True  True  True  True False False  True  True  True  True
  True  True  True  True  True  True  True  True  True  True False  True
  True  True  True  True  True  True  True  True  True  True  True False
  True  True  True  True  True  True  True False  True  True False  True
  True False  True]
```

```
# 8
neigh1=KNeighborsClassifier(n_neighbors=1)
neigh1.fit(data,label)
print(neigh1.score(X_test, y_test))
print(neigh.score(X_test, y_test))
```

```
1.0
0.8862745098039215
```

```
# 9 #1 from here lab_04 begins
from sklearn.metrics import confusion_matrix, classification_report
y_train_pred=neigh.predict(X_train)
y_test_pred=neigh.predict(X_test)

train_accuracy=neigh.score(X_train, y_train)
test_accuracy=neigh.score(X_test, y_test)

print("training accuracy:",train_accuracy)
print("testing accuracy:",test_accuracy)

train_conf_matrix=confusion_matrix(y_train,y_train_pred)
test_conf_matrix=confusion_matrix(y_test,y_test_pred)

print("Confusion Matrix for Train Data:\n", train_conf_matrix)
print("Confusion Matrix for Test Data:\n", test_conf_matrix)

precition_train=train_conf_matrix[0][0]/(train_conf_matrix[0][0]+train_conf_matrix[0][1])
precition_test=test_conf_matrix[0][0]/(test_conf_matrix[0][0]+test_conf_matrix[0][1])

print("precition for training data",precition_train)
print("precition for testing data",precition_test)

recall_train=train_conf_matrix[0][0]/(train_conf_matrix[0][0]+train_conf_matrix[1][0])
recall_test=test_conf_matrix[0][0]/(test_conf_matrix[0][0]+test_conf_matrix[1][0])

print("recall for training data",recall_train)
```

```
print("recall for testing data",recall_test)

f1_train=(2*train_conf_matrix[0][0])/(2*train_conf_matrix[0][0]+train_conf_matrix[0][1]+test_conf_matrix[1][(
f1_test=(2*test_conf_matrix[0][0])/(2*test_conf_matrix[0][0]+test_conf_matrix[0][1]+test_conf_matrix[1][0])

print("f1 score for training data",f1_train)
print("f1 score for testing data",f1_test)


if train_accuracy > 0.98 and test_accuracy < 0.75:
    print("The model is Overfitting.")
elif train_accuracy < 0.70 and test_accuracy < 0.70:
    print("The model is Underfitting.")
else:
    print("The model is Generalized well (Regular fit).")
```

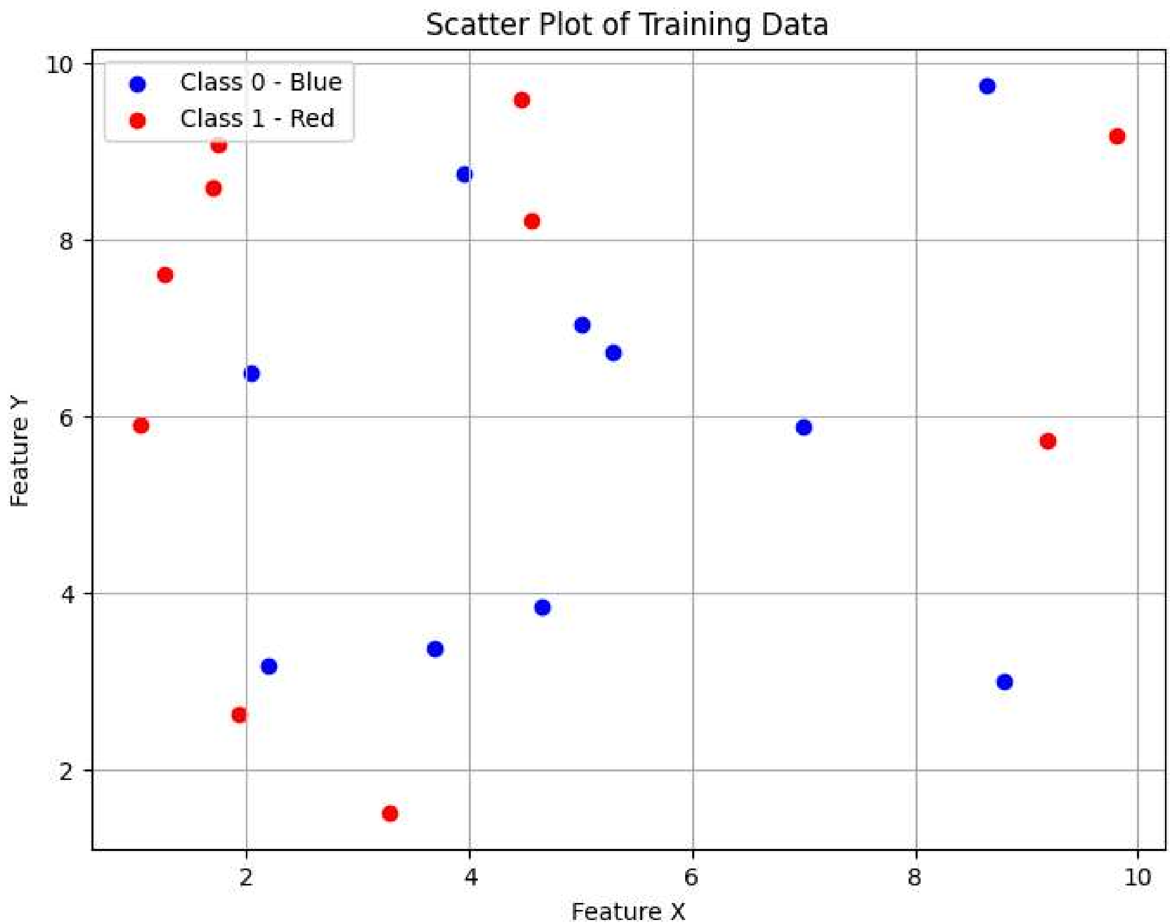
```
training accuracy: 0.9021922428330523
testing accuracy: 0.8862745098039215
Confusion Matrix for Train Data:
 [[275  44]
 [ 14 260]]
Confusion Matrix for Test Data:
 [[129  24]
 [  5  97]]
precition for training data 0.8620689655172413
precition for testing data 0.8431372549019608
recall for training data 0.9515570934256056
recall for testing data 0.9626865671641791
f1 score for training data 0.9181969949916527
f1 score for testing data 0.8989547038327527
The model is Generalized well (Regular fit).
```

```
#2
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error, r2_score
import numpy as np
x = np.array([386, 289, 393, 110, 280, 167, 271, 274, 148, 198])
x_result = np.array([386, 289, 393, 110, 280, 167, 271, 274, 148, 198])
MSE=mean_squared_error(x,x_result)
RMSE=np.sqrt(MSE)
MAPE=mean_absolute_percentage_error(x,x_result)
R2=r2_score(x,x_result)
print("mean square error:",MSE)
print("root mean square error:",RMSE)
print("mean absolute persentage error:",MAPE)
print("R2 score :",R2)
```

```
mean square error: 0.0
root mean square error: 0.0
mean absolute persentage error: 0.0
R2 score : 1.0
```

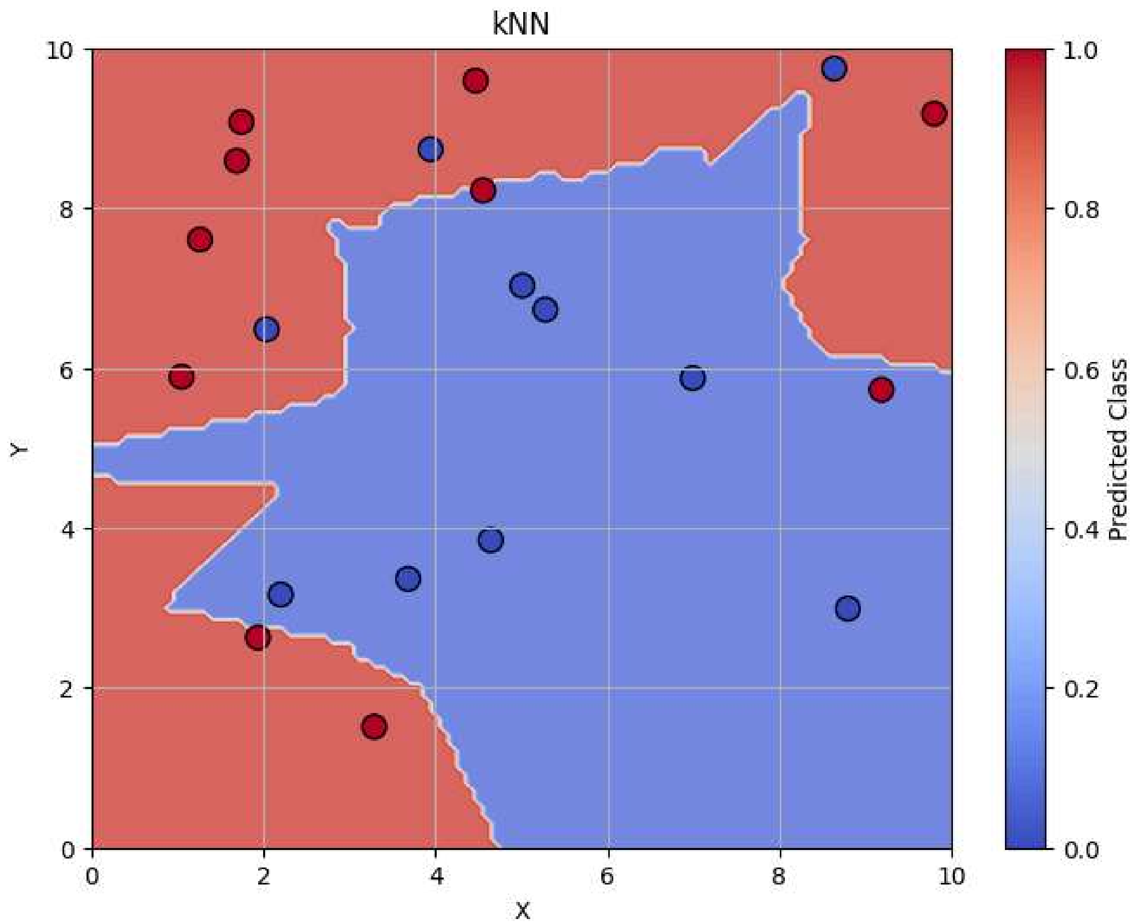
```
#3
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(43)
data=np.random.uniform(1, 10, (20, 2))
classes=np.random.choice([0, 1], size=20)
class_0=data[classes==0]
```

```
class_1=data[classes==1]

plt.figure(figsize=(8,6))
plt.scatter(class_0[:, 0], class_0[:, 1], color='blue', label='Class 0 - Blue')
plt.scatter(class_1[:, 0], class_1[:, 1], color='red', label='Class 1 - Red')
plt.title('Scatter Plot of Training Data')
plt.xlabel('Feature X')
plt.ylabel('Feature Y')
plt.legend()
plt.grid(True)
plt.show()
```



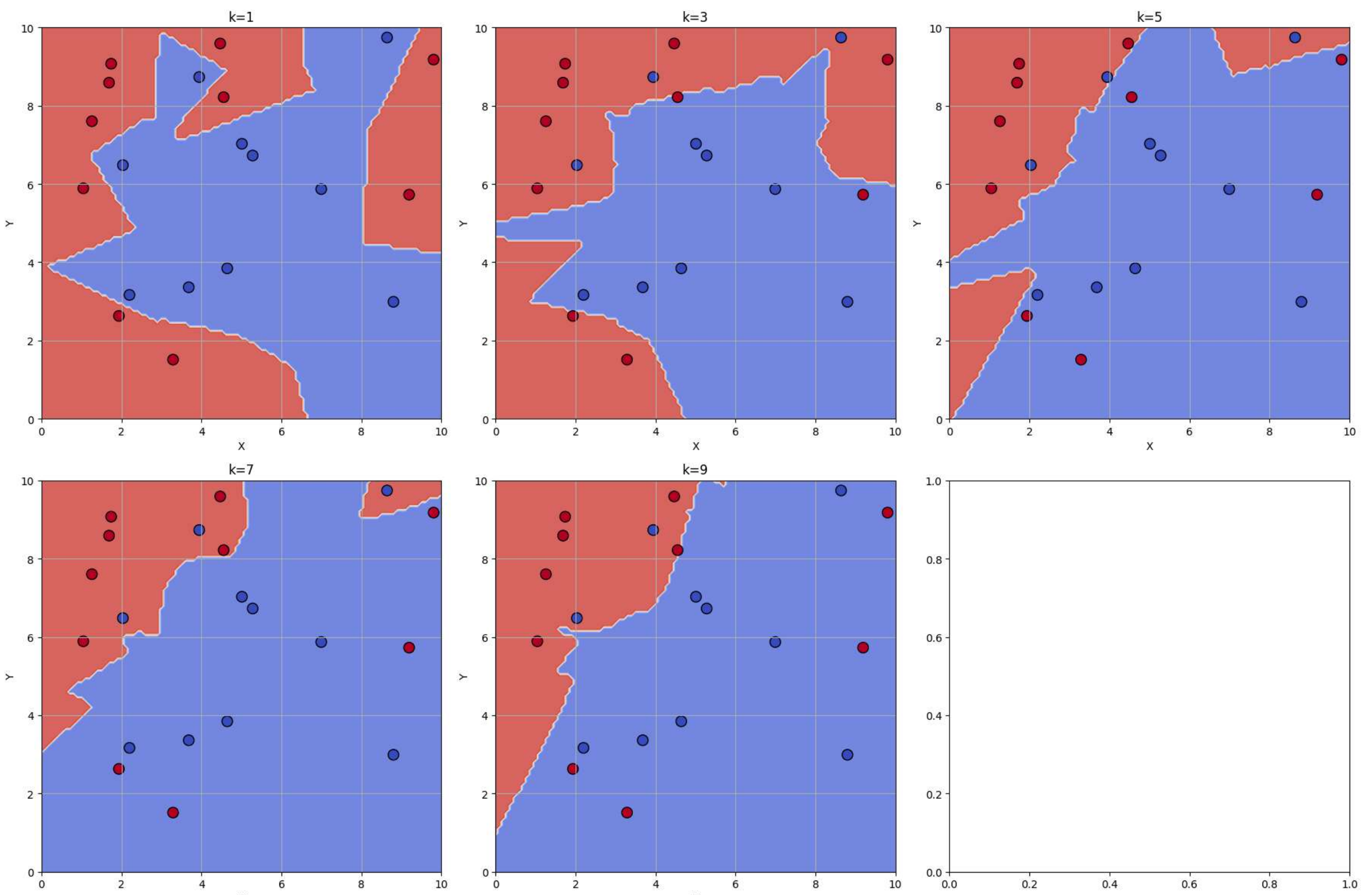

```
#4
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
import itertools

np.random.seed(43)
x_train=np.random.uniform(1,10,(20,2))
y_train=np.random.choice([0,1],size=20)

x_test=np.arange(0,10.1,0.1)
y_test=np.arange(0,10.1,0.1)
x1,y1=np.meshgrid(x_test,y_test)
test_data=np.c_[x1.ravel(),y1.ravel()]

knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train,y_train)
```

```
predicted_labels=knn.predict(test_data)
predicted_labels=predicted_labels.reshape(x1.shape)

plt.figure(figsize=(8, 6))
plt.contourf(x1, y1,predicted_labels,alpha=0.8,cmap='coolwarm')
plt.scatter(x_train[:,0],x_train[:, 1],c=y_train,edgecolor='k',s=100,cmap='coolwarm')
plt.title('kNN')
plt.xlabel('X')
plt.ylabel('Y')
plt.colorbar(label='Predicted Class')
plt.grid(True)
```



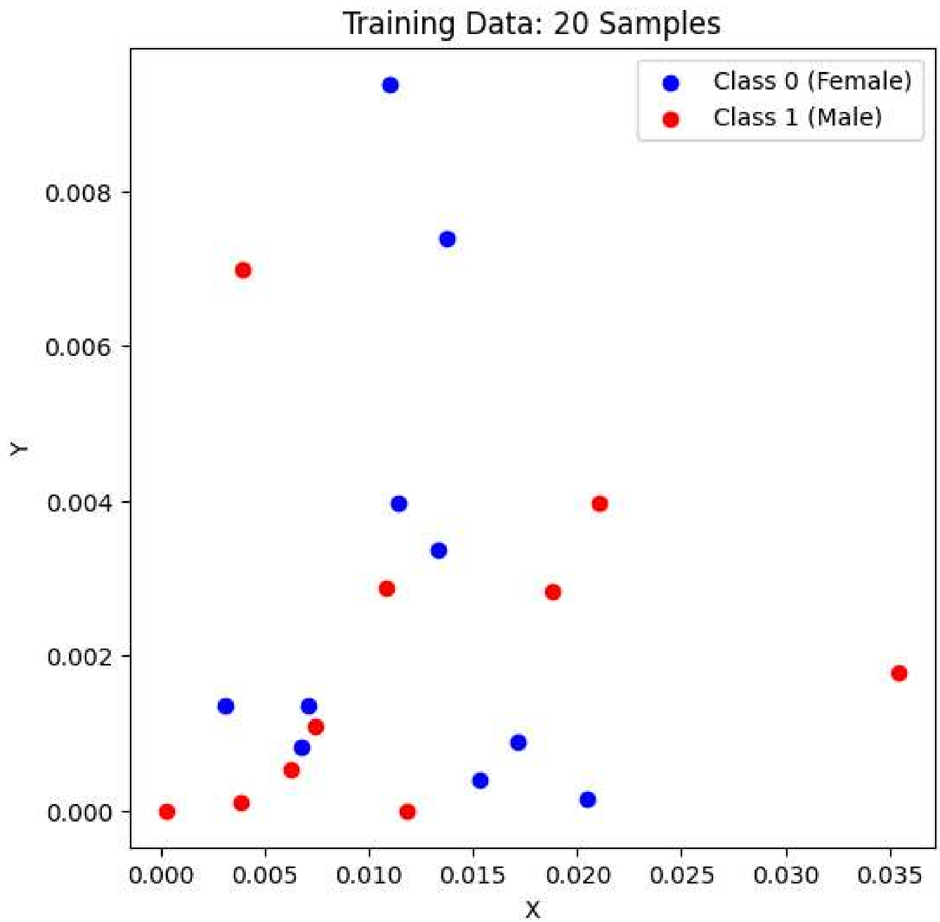

```
# 5
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

np.random.seed(43)

x_train=np.random.uniform(1, 10, (20, 2))
y_train=np.random.choice([0, 1], size=20)

x_test=np.arange(0, 10.1, 0.1)
y_test=np.arange(0, 10.1, 0.1)
x1,y1=np.meshgrid(x_test, y_test)
test_data=np.c_[x1.ravel(), y1.ravel()]

k_values=[1, 3, 5, 7, 9]
```

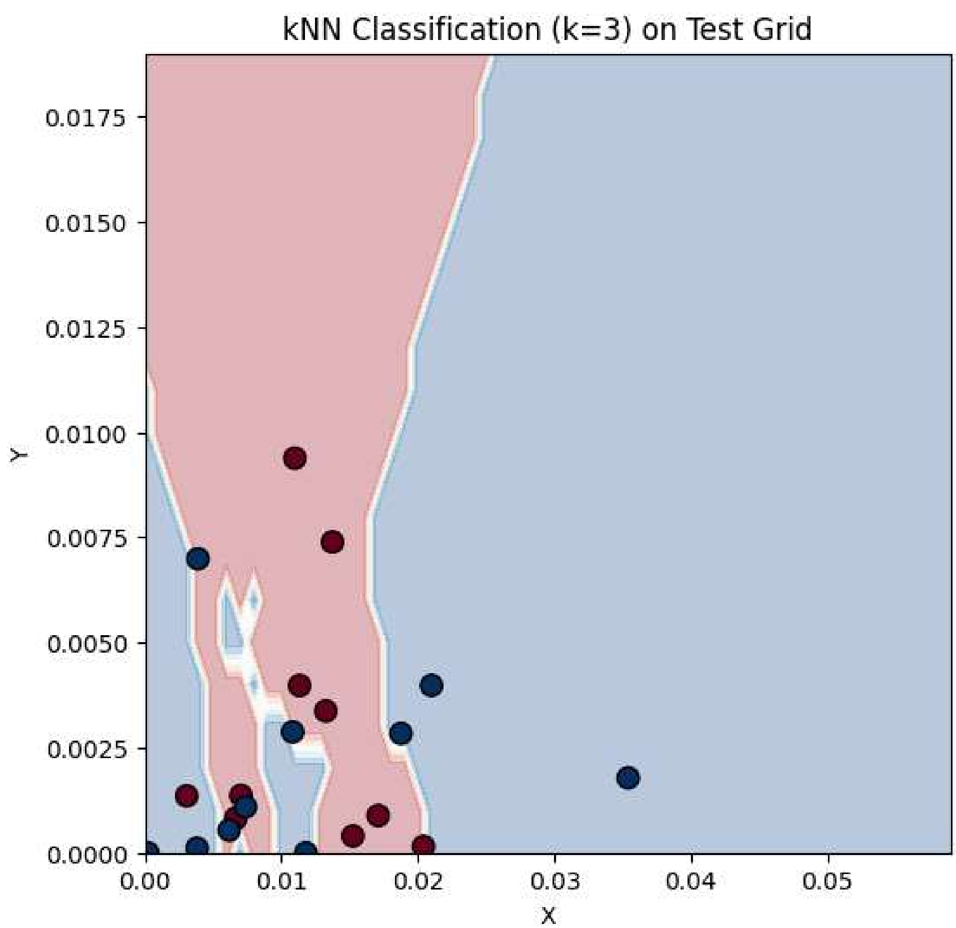
```
fig,axes=plt.subplots(nrows=2,ncols=3,figsize=(18,12))
axes=axes.flatten()

for i,k in enumerate(k_values):
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)

    predicted_labels=knn.predict(test_data)
    predicted_labels=predicted_labels.reshape(x1.shape)

    ax=axes[i]
    ax.contourf(x1,y1,predicted_labels,alpha=0.8,cmap='coolwarm')
    ax.scatter(x_train[:, 0],x_train[:, 1],c=y_train,edgecolor='k',s=100,cmap='coolwarm')
    ax.set_title(f'k={k}')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.grid(True)

plt.tight_layout()
plt.show()
```

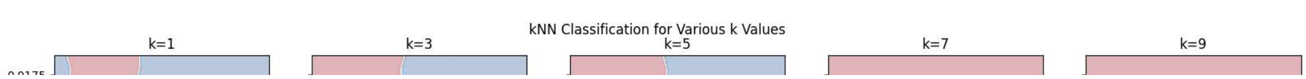


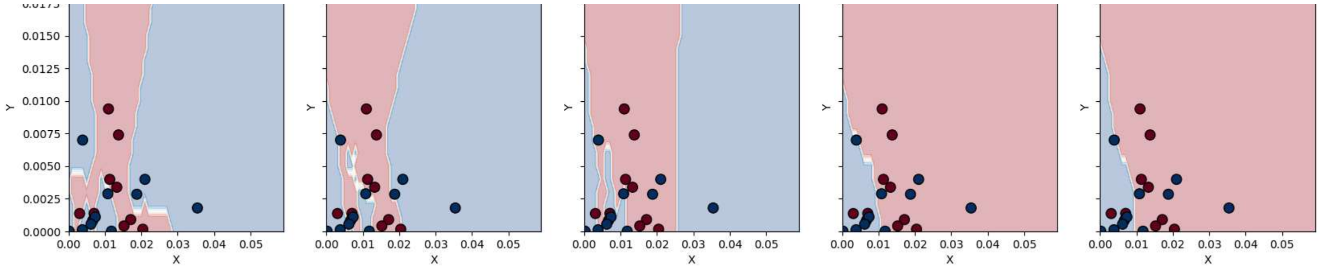
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
df = pd.read_excel("output.xlsx")


#6
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd

np.random.seed(43)
selected_columns=df.iloc[:,:-1]
selected_label=df.iloc[:,-1]
data=selected_columns
label=selected_label
data_female=data[label==0].sample(n=10,random_state=43)
data_male=data[label==1].sample(n=10,random_state=42)
train_set=pd.concat([data_female,data_male])
feat_x=9
feat_y=10
X_train=train_set[[feat_x,feat_y]].values
y_train=label[train_set.index].values
plt.figure(figsize=(6,6))
clr={0:'blue',1:'red'}
for lbl in np.unique(y_train):
    subset=train_set[label[train_set.index]==lbl]
    plt.scatter(subset[feat_x],subset[feat_y],color=clr[lbl],label=f'Class {lbl} ({"Female" if lbl==0 else
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Training Data: 20 Samples')
plt.legend()
plt.show()
x_min,x_max=0,0.06
y_min,y_max=0,0.02
x1,y1=np.meshgrid(np.arange(x_min,x_max,0.001),np.arange(y_min,y_max,0.001))
X_test=np.c_[x1.ravel(),y1.ravel()]
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
y_test_pred=knn.predict(X_test)
Z=y_test_pred.reshape(x1.shape)
plt.figure(figsize=(6,6))
plt.contourf(x1,y1,Z,alpha=0.3,cmap=plt.cm.RdBu)
plt.scatter(X_train[:,0],X_train[:,1],c=y_train,cmap=plt.cm.RdBu,edgecolor='k',s=80)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('kNN Classification (k=3) on Test Grid')
plt.show()
k_vals=[1,3,5,7,9]
fig,axes=plt.subplots(1,len(k_vals),figsize=(20,4),sharex=True,sharey=True)
for ax,k in zip(axes,k_vals):
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train,y_train)
    y_test_pred=knn.predict(X_test)
    Z=y_test_pred.reshape(x1.shape)
    ax.contourf(x1,y1,Z,alpha=0.3,cmap=plt.cm.RdBu)
    ax.scatter(X_train[:,0],X_train[:,1],c=y_train,cmap=plt.cm.RdBu,edgecolor='k',s=80)
    ax.set_title(f'k={k}')
```

```
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
    plt.suptitle('kNN Classification for Various k Values')
    plt.show()
```


Training Data: 20 Samples
Class 0 (Female)
Class 1 (Male)
0.008
0.006
0.004
0.002
0.000
Y
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
X


kNN Classification (k=3) on Test Grid
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
Y
0.00
0.01
0.02
0.03
0.04
0.05
X


kNN Classification for Various k Values
k=1
k=3
k=5
k=7
k=9




```
# 7
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV,train_test_split

np.random.seed(42)
train_data=np.random.uniform(1,10,(20,2))
train_labels=np.random.choice([0,1],size=20)
X_train,X_val,y_train,y_val=train_test_split(train_data,train_labels,test_size=0.2,random_state=42)
knn=KNeighborsClassifier()
param_grid={'n_neighbors':list(range(1,21,2))}
grid_search=GridSearchCV(estimator=knn,param_grid=param_grid,cv=5,scoring='accuracy')
grid_search.fit(X_train,y_train)
best_k=grid_search.best_params_['n_neighbors']
best_score=grid_search.best_score_
print(f"Best k value: {best_k}")
print(f"Best validation accuracy: {best_score:.4f}")
best_knn=grid_search.best_estimator_
val_accuracy=best_knn.score(X_val,y_val)
print(f"Validation accuracy with best k: {val_accuracy:.4f}")
print("---------------------------------------------------------------------------------------------------
print("---------------------------------------------------------------------------------------------------
print("---------------------------------------------------------------------------------------------------
print("---------------------------------------------------------------------------------------------------
print("---------------------------------------------------------------------------------------------------
print("---------------------------------------------------------------------------------------------------
print("---------------------------------------------------------------------------------------------------
print("---------------------------------------------------------------------------------------------------
print("---------------------------------------------------------------------------------------------------
```