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

```
from google.colab import drive
drive.mount('/content/drive')  # Mount Google Drive
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import pandas as pd
file_path = '/content/drive/MyDrive/ML_SG/Classified Data.csv'
df = pd.read_csv(file_path)
```

```
df.head(4)
```

	Unnamed: 0	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1

Next steps: Generate code with df | View recommended plots | New interactive sheet

Dependent and Independent Fetatures

```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

```
X.head(4)
```

	Unnamed: 0	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
0	0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409
1	1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702
2	2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597
3	3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093

Next steps: Generate code with X | View recommended plots | New interactive sheet

```
X.columns
```

```
Index(['Unnamed: 0', 'WTT', 'PTI', 'EQW', 'SBI', 'LQE', 'QWG', 'FDJ', 'PJF',
       'HQE', 'NXJ'],
      dtype='object')
```

```
y.head(3)
```

	TARGET CLASS
0	1
1	0
2	0

Standardization

```
from sklearn.preprocessing import StandardScaler
```

```
scaler=StandardScaler()
scaled_feature=scaler.fit_transform(X)
```

```
scaled_feature
```

```
array([[-1.73031962, -0.12354188,  0.18590747, ..., -1.48236813,
         -0.9497194 , -0.64331425],
       [-1.72685552, -1.08483602, -0.43034845, ..., -0.20224031,
         -1.82805088,  0.63675862],
       [-1.72339142, -0.78870217,  0.33931821, ...,  0.28570652,
         -0.68249379, -0.37784986],
       ...,
       [ 1.72339142,  0.64177714, -0.51308341, ..., -2.36249443,
         -0.81426092,  0.11159651],
       [ 1.72685552,  0.46707241, -0.98278576, ..., -0.03677699,
          0.40602453, -0.85567   ],
       [ 1.73031962, -0.38765353, -0.59589427, ..., -0.56778932,
          0.3369971 ,  0.01034996]])
```

```
new_df=pd.DataFrame(scaled_feature,columns=X.columns)
```

```
new_df.head(4)
```

	Unnamed: 0	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
0	-1.730320	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.726856	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-1.723391	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	-1.719927	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987

Next steps: Generate code with new_df | View recommended plots | New interactive sheet

train_test_split

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(new_df,y,test_size=0.30,random_state=42)
```

```
X_train
```

	Unnamed: 0	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
541	0.143760	-0.584865	0.042367	0.341874	-1.728983	-0.180807	-1.102021	-0.114799	-0.830409	0.494005	0.310421
440	-0.206114	-0.716170	0.015905	0.295859	-0.263682	1.781141	0.292634	-1.987108	-0.024566	-1.010569	0.594582
482	-0.060622	0.736804	0.674725	0.880257	-0.953838	0.586478	-1.460312	0.569524	0.228368	-1.310556	0.889933
422	-0.268468	1.734803	-1.319249	-0.266479	-0.470776	0.165927	-1.511446	0.127236	-0.247082	0.787180	0.127149
778	0.964753	-0.454654	1.371033	0.217861	2.601246	0.115137	0.756867	-0.428684	2.083538	-0.723318	0.183293
...	...	...	...	...	...	...	...	...	...	...	...
106	-1.363125	-1.978598	0.475777	-0.422636	-1.134506	-0.406097	1.351978	-0.621618	-0.241688	-0.869905	-1.996312
270	-0.795012	0.220902	0.689820	-0.343130	-0.679380	-0.874421	-1.057533	0.706221	-0.509378	1.124947	0.495153
860	1.248809	-1.025521	1.241777	0.358562	0.220358	1.838459	-0.103198	0.325037	0.132569	-1.510792	0.832863
435	-0.223435	0.888357	0.802441	-0.802459	0.036219	-0.411028	-1.040641	-1.206336	0.071199	-1.351282	0.933751
102	-1.376981	0.984286	1.864232	-1.048236	-0.576055	0.283768	-0.629910	0.504642	-0.377265	-0.231939	-0.287136

700 rows × 11 columns

Next steps: Generate code with X_train | View recommended plots | New interactive sheet

knn algorithm

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn=KNeighborsClassifier(n_neighbors=1)
```

```
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

```
y_pred=knn.predict(X_test)
```

```
y_pred
```

```
array([0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0])
```

Prediction and Evaluation

```
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix
```

```
print(accuracy_score(y_pred,y_test))
```

```
0.9266666666666666
```

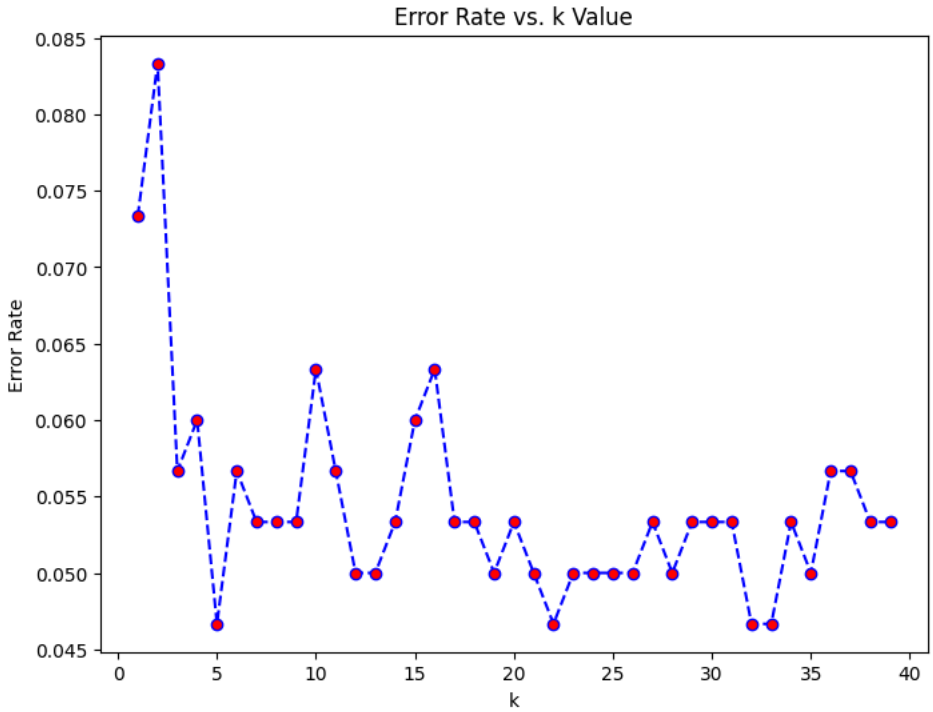
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.93      0.92      0.92       144
           1       0.92      0.94      0.93       156

    accuracy                           0.93       300
   macro avg       0.93      0.93      0.93       300
weighted avg       0.93      0.93      0.93       300
```

Choose K value

```
error_rate=[]
for i in range(1,40):
  knn=KNeighborsClassifier(n_neighbors=i)
  knn.fit(X_train,y_train)
  pred=knn.predict(X_test)
  error_rate.append(np.mean(pred!=y_test))
```

```
plt.figure(figsize=(8,6))
plt.plot(range(1,40), error_rate, color='blue', linestyle='dashed', marker='o', markerfacecolor='red') #x=range(1,40),y=error_rate
plt.title('Error Rate vs. k Value')
plt.xlabel('k')
plt.ylabel('Error Rate')
plt.show()
```




```
#accuracy with k=1
knn=KNeighborsClassifier(n_neighbors=1)
print(accuracy_score(y_pred,y_test))
print('\n')
print(classification_report(y_pred,y_test))
print('\n')
print(confusion_matrix(y_pred,y_test))
```

```
0.9266666666666666


              precision    recall  f1-score   support

           0       0.93      0.92      0.92       144
           1       0.92      0.94      0.93       156

    accuracy                           0.93       300
   macro avg       0.93      0.93      0.93       300
weighted avg       0.93      0.93      0.93       300


[[132  12]
 [ 10 146]]
```

```
#accuracy with the k=23
knn=KNeighborsClassifier(n_neighbors=23)
knn.fit(X_train,y_train)
k_pred=knn.predict(X_test)
print(accuracy_score(k_pred,y_test))
print('\n')
print(confusion_matrix(k_pred,y_test))
print('\n')
print(classification_report(k_pred,y_test))
```

```
0.95


[[135   8]
 [  7 150]]


              precision    recall  f1-score   support

           0       0.95      0.94      0.95       143
           1       0.95      0.96      0.95       157

    accuracy                           0.95       300
   macro avg       0.95      0.95      0.95       300
weighted avg       0.95      0.95      0.95       300
```