Predict K value for Data

Problem Statement- we will be working with the some artificial dataset where labels have been hided due to company's data protection policies and we will targeting and try to predict nearest K value based of rest feature inside the data

Method: K-NN/source of Data: Udemy/Tool: Python

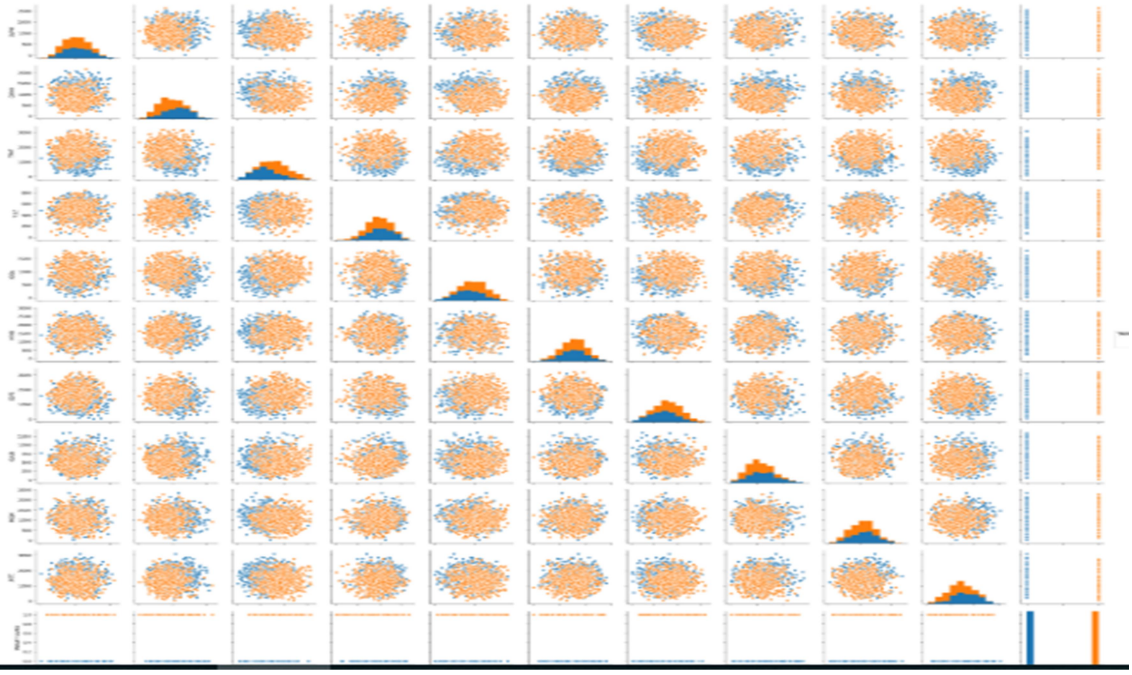
Library: - **Pandas, Matplotlib, sklearn, seaborn, Numpy**

Dataset: head of dataset is as below and Target Colum**: 'Target Class'**

```
df.head()
```

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS	GUUB	MGJM	JHZC	TARGET CLASS
0	1636.670614	817.988525	2565.995189	358.347163	550.417491	1618.870897	2147.641254	330.727893	1494.878631	845.136088	0
1	1013.402760	577.587332	2644.141273	280.428203	1161.873391	2084.107872	853.404981	447.157619	1193.032521	861.081809	1
2	1300.035501	820.518697	2025.854469	525.562292	922.206261	2552.355407	818.676686	845.491492	1968.367513	1647.186291	1
3	1059.347542	1066.866418	612.000041	480.827789	419.467495	685.666983	852.867810	341.664784	1154.391368	1450.935357	0
4	1018.340526	1313.679056	950.622661	724.742174	843.065903	1370.554164	905.469453	658.118202	539.459350	1899.850792	0

since data is artificial will just look via plot and move on prediction.



here is one thing is very important that while using K-NN method we need to standardized our data set otherwise we can't get better result hence will make our data normalize before to proceed with our prediction

Standardize the Variables: sklearn library has powerful ability to convert high variance data into normalize as below:

- from sklearn.preprocessing import StandardScaler..... (import library)
- scaler = StandardScaler()..... (convert comand into scaler)
- scaler.fit(df.drop('TARGET CLASS',axis=1))..... (fiting feature)
- scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))).... (transfrom fearture into scaled version
- df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])....(convert into dataset with old lables)
- df_feat.head().....(check normaize value)

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS	GUUB	MGJM	JHZC
0	1.568522	-0.443435	1.619808	-0.958255	-1.128481	0.138336	0.980493	-0.932794	1.008313	-1.069627
1	-0.112376	-1.056574	1.741918	-1.504220	0.640009	1.081552	-1.182663	-0.461864	0.258321	-1.041546
2	0.660647	-0.436981	0.775793	0.213394	-0.053171	2.030872	-1.240707	1.149298	2.184784	0.342811
3	0.011533	0.191324	-1.433473	-0.100053	-1.507223	-1.753632	-1.183561	-0.888557	0.162310	-0.002793
4	-0.099059	0.820815	-0.904346	1.609015	-0.282065	-0.365099	-1.095644	0.391419	-1.365603	0.787762

kindly note we have not done anything for target variable since that is actual column for which we have to predict nearest K value based on features.

Train Test Split: now it's time to split our data set into training and test so we can fit model.

- from sklearn.model_selection import train_test_split .....(call library for test and split)
- X=df.drop('Target Class', axis=1)....(choose X feature)
- y=df['Target Class').....(choose y feature)
- X_train, X_test, y_train, y_test = rain_test_split(scaled_features,df['TARGET

CLASS'], test_size=0.30)....(data splited via this codes)

Call KNN: now after split dataset we can use sklearn library to call function as below:

- from sklearn.neighbors import KNeighborsClassifier....(library called)
- knn = KNeighborsClassifier(n_neighbors=1).....(covert function as knn to easy to use)
- knn.fit(X_train,y_train).....(fit model to training data)

Predictions and Evaluations: It's time to predict our training models

pred = knn.predict(X_test) ….(call prediction)

from sklearn.metrics import classification_report,confusion_matrix ……(library to read confusion matrix and classification reports)

```
print(confusion_matrix(y_test,pred))
```

```
[[112  40]
 [ 34 114]]
```

```
print(classification_report(y_test,pred))
```

	precision	recall	f1-score	support
0	0.77	0.74	0.75	152
1	0.74	0.77	0.75	148
avg / total	0.75	0.75	0.75	300

So based on classification report accuracy is not too bad for this model

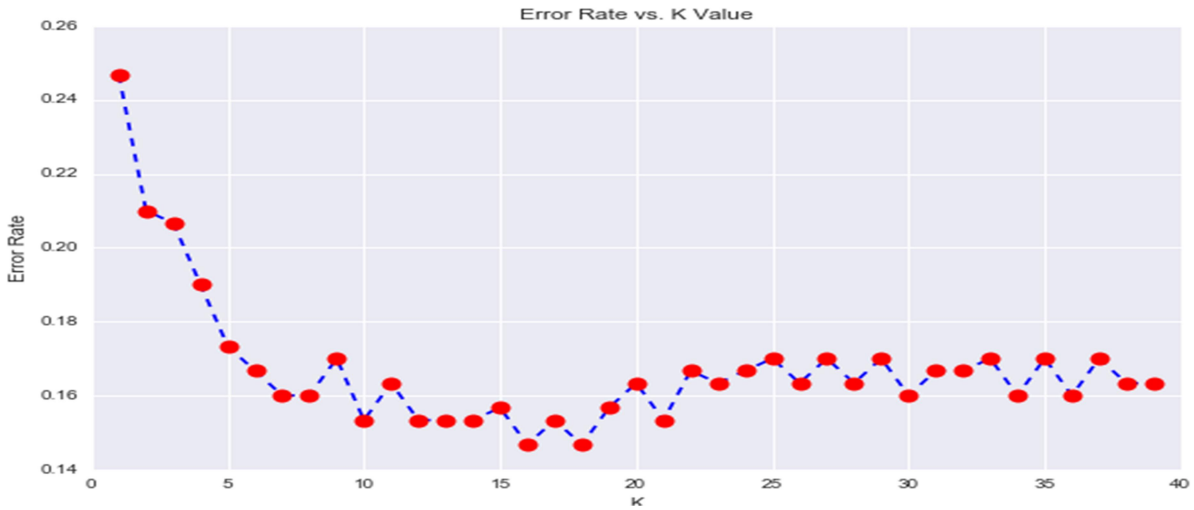
Choosing a K Value: we will create loop to get correct K value for our model is below:

```
error_rate = []
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

Now create the following plot using the information from your for loop.

- plt.figure(figsize=(10,6))
- plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
    - markerfacecolor='red', markersize=10)
- plt.title('Error Rate vs. K Value')
- plt.xlabel('K')
- plt.ylabel('Error Rate')



Observation: this time will choose 30 as new K value since its look more stable in the above graph.

Retrain with new K Value: we will try to retrain our model with new acquire K value and check if model is better than earlier version.

- knn = KNeighborsClassifier(n_neighbors=30)
- knn.fit(X_train,y_train)
- pred = knn.predict(X_test)

Print confusion matrix and classification report to check accuracy of model

- print('WITH K=30')
- print('\n')
- print(confusion_matrix(y_test,pred))
- print('\n')
- print(classification_report(y_test,pred))

```
WITH K=30


[[118  35]
 [ 21 126]]


             precision    recall  f1-score   support

          0       0.85      0.77      0.81       153
          1       0.78      0.86      0.82       147

avg / total       0.82      0.81      0.81       300
```

Final observation: while comparing old model new model with K value 30 is looks better since accuracy has increased with new k value.