Loading Modules and Functions

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix
```

Loading Data

In [ ]:
```
df = pd.read_csv("../processed.csv")
df.dropna(subset=["Initial_Price"],inplace=True)
df.head()
```

Out[ ]:

	Initial_Price	Final_Price	Win_Flag	Mac_Flag	Linux_Flag	Positive_Reviews	Negative_Reviews	Memory_MB	Storage_MB	target
0	52.0	52.0	True	True	False	57.0	7.0	1024	50	1
1	0.0	0.0	True	True	False	53.0	6.0	2048	3072	1
2	0.0	0.0	True	False	False	133.0	69.0	2048	100	0
3	530.0	530.0	True	False	False	22.0	9.0	2048	500	0
4	229.0	229.0	True	True	True	226.0	44.0	2048	1500	1

Normalising Continous Features

In [ ]:
```
def normalise(feature, df):
    mean = df[feature].mean()
    sd = df[feature].std()
    df[feature] = (df[feature] - mean) / sd

normalise("Initial_Price", df)
normalise("Final_Price", df)
normalise("Positive_Reviews", df)
normalise("Negative_Reviews", df)
normalise("Memory_MB", df)
normalise("Storage_MB", df)
df
```

Out[ ]:

	Initial_Price	Final_Price	Win_Flag	Mac_Flag	Linux_Flag	Positive_Reviews	Negative_Reviews	Memory_MB	Storage_MB	target
0	-0.271301	-0.258274	True	True	False	-0.034488	-0.033031	-0.004171	-0.004171	1
1	-0.322070	-0.309572	True	True	False	-0.034609	-0.033196	-0.004171	-0.004171	1
2	-0.322070	-0.309572	True	False	False	-0.032193	-0.022842	-0.004171	-0.004171	0
3	0.195385	0.213271	True	False	False	-0.035545	-0.032703	-0.004171	-0.004171	0
4	-0.098490	-0.083665	True	True	True	-0.029385	-0.026950	-0.004171	-0.004171	1
...	...	...	...	...	...	...	...	...	...	...
57467	-0.239082	-0.225720	True	False	False	-0.036210	-0.033524	-0.004171	-0.004171	-1
57468	0.018669	0.034715	True	True	False	-0.036149	-0.034017	-0.004171	-0.004171	1
57469	-0.161952	-0.147787	True	False	False	-0.035968	-0.034017	-0.004171	-0.004171	1
57470	0.273492	0.292190	True	False	False	-0.036179	-0.034182	-0.004171	-0.004171	1
57471	0.234439	-0.028421	True	False	False	-0.036210	-0.034017	-0.004171	-0.004171	-1

57472 rows × 10 columns

Splitting data to 33% Test , 66% Train

In [ ]:
```
y = df["target"]
X = df.drop(labels=["target"],axis=1)

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.33, random_state=100,shuffle=True)
```

TRIAL - 1 : setting the value of k = 5 (using brute force)

Fitting the model

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=5,weights='distance')
knn.fit(X_train,y_train)
```

Out[ ]:
```
KNeighborsClassifier(weights='distance')
```

Evaluating Model with test data

In [ ]:
```
pred = knn.predict(X_test)

accuracy = accuracy_score(y_test,pred)
print(f"Accuracy : {accuracy_score(y_test, pred)}")
```

```
Accuracy : 0.8594853949172203
```

Important Metrics

In [ ]:
```
classification_report(y_test, pred)
```

Out[ ]:
```
'              precision    recall  f1-score   support\n\n          -1       0.75      0.64      0.69      2123\n           0
0.71      0.72      0.71      3563\n           1       0.91      0.93      0.92     13280\n\n    accuracy
0.86     18966\n   macro avg       0.79      0.76      0.78     18966\nweighted avg       0.86      0.86      0.86     18966
\n'
```

Display Confusion Matrix

In [ ]:
```
confusion_matrix(y_test,pred)
```

Out[ ]:
```
array([[ 1351,   396,   376],
       [  213,  2561,   789],
       [  241,   650, 12389]], dtype=int64)
```

Conclusion

KNN classifier shows a performance of 85% after normalising the continuous attributes. The value of k as 5 turns out to be the best value. Aditionally, another factor to further improve the accuracy in knn classifier is the nearest k=5 points are given a weight.

TRIAL - 2 : setting the value of k = 239 (sqrt(N))

Fitting the model

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=239,weights='distance')
knn.fit(X_train,y_train)
```

Out[ ]:
```
KNeighborsClassifier(n_neighbors=239, weights='distance')
```

Evaluating Model with test data

In [ ]:
```
pred = knn.predict(X_test)

accuracy = accuracy_score(y_test,pred)
print(f"Accuracy : {accuracy_score(y_test, pred)}")
```

```
Accuracy : 0.8252135400189813
```

Important Metrics

In [ ]:
```
classification_report(y_test, pred)
```

Out[ ]:
```
'              precision    recall  f1-score   support\n\n          -1       0.86      0.51      0.64      2123\n           0
0.78      0.45      0.57      3563\n           1       0.83      0.98      0.90     13280\n\n    accuracy
0.83     18966\n   macro avg       0.82      0.65      0.70     18966\nweighted avg       0.82      0.83      0.81     18966
\n'
```

Display Confusion Matrix

In [ ]:
```
confusion_matrix(y_test,pred)
```

Out[ ]:
```
array([[ 1087,   235,   801],
       [   67,  1615,  1881],
       [  113,   218, 12949]], dtype=int64)
```

Conclusion

Setting the value of k as sqrt N is another optimal K value for KNN, gives approximately K = 239 produces a pretty good performance of 82% but hasn't surpassed the performance given by k = 5. Therefore the best performance is shown at K = 5.