In [ ]:
```
Name: Bhavesh Waghela
Student Number: N01639685
```

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
churn_df = pd.read_csv('churn_data.csv')
churn_df
```

Out[2]:

	Unnamed: 0	MonthlyCharges	Contract	ServiceUsage	CustomerSupportCalls	Churn
0	0	79.393215	One year	9.246997	10	Yes
1	1	94.367043	One year	10.134628	9	No
2	2	84.248704	Two year	3.111158	8	No
3	3	79.039486	Month-to-month	0.991352	0	No
4	4	68.128932	Two year	0.113940	0	No
...	...	...	...	...	...	...
995	995	38.790873	Two year	22.795105	0	Yes
996	996	76.342998	Two year	8.435893	6	No
997	997	114.457082	Two year	17.608176	3	Yes
998	998	50.578190	One year	22.432958	9	Yes
999	999	90.942703	One year	2.788098	8	No

1000 rows × 6 columns

In [3]:
```
churn_df['Churn'] = churn_df.Churn.eq('Yes').mul(1)
churn_df
```

Out[3]:

	Unnamed: 0	MonthlyCharges	Contract	ServiceUsage	CustomerSupportCalls	Churn
0	0	79.393215	One year	9.246997	10	1
1	1	94.367043	One year	10.134628	9	0
2	2	84.248704	Two year	3.111158	8	0
3	3	79.039486	Month-to-month	0.991352	0	0
4	4	68.128932	Two year	0.113940	0	0
...	...	...	...	...	...	...
995	995	38.790873	Two year	22.795105	0	1
996	996	76.342998	Two year	8.435893	6	0
997	997	114.457082	Two year	17.608176	3	1
998	998	50.578190	One year	22.432958	9	1
999	999	90.942703	One year	2.788098	8	0

1000 rows × 6 columns

In [4]:
```
churn_df = pd.get_dummies(churn_df, columns = ['Contract'])
churn_df
```

Out[4]:

	Unnamed: 0	MonthlyCharges	ServiceUsage	CustomerSupportCalls	Churn	Contract_Month-to-month
0	0	79.393215	9.246997	10	1	0
1	1	94.367043	10.134628	9	0	0
2	2	84.248704	3.111158	8	0	0
3	3	79.039486	0.991352	0	0	1
4	4	68.128932	0.113940	0	0	0
...	...	...	...	...	...	...
995	995	38.790873	22.795105	0	1	0
996	996	76.342998	8.435893	6	0	0
997	997	114.457082	17.608176	3	1	0
998	998	50.578190	22.432958	9	1	0
999	999	90.942703	2.788098	8	0	0

1000 rows × 8 columns

In [5]:
```
churn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Unnamed: 0               1000 non-null   int64
 1   MonthlyCharges           1000 non-null   float64
 2   ServiceUsage             1000 non-null   float64
 3   CustomerSupportCalls     1000 non-null   int64
 4   Churn                    1000 non-null   int32
 5   Contract_Month-to-month  1000 non-null   uint8
 6   Contract_One year        1000 non-null   uint8
 7   Contract_Two year        1000 non-null   uint8
dtypes: float64(2), int32(1), int64(2), uint8(3)
memory usage: 38.2 KB
```

In [6]:
```
churn_df = churn_df.drop(columns=['Unnamed: 0'], axis=1)
churn_df
```

Out[6]:

	MonthlyCharges	ServiceUsage	CustomerSupportCalls	Churn	Contract_Month-to-month	Contract_O ye
0	79.393215	9.246997	10	1	0	
1	94.367043	10.134628	9	0	0	
2	84.248704	3.111158	8	0	0	
3	79.039486	0.991352	0	0	1	
4	68.128932	0.113940	0	0	0	
...	...	...	...	...	...	
995	38.790873	22.795105	0	1	0	
996	76.342998	8.435893	6	0	0	
997	114.457082	17.608176	3	1	0	
998	50.578190	22.432958	9	1	0	
999	90.942703	2.788098	8	0	0	

1000 rows × 7 columns

In [7]:
```
final_df = [churn_df, churn_df]
churn_df = pd.concat(final_df)
display(churn_df)
```

	MonthlyCharges	ServiceUsage	CustomerSupportCalls	Churn	Contract_Month-to-month	Contract_O ye
0	79.393215	9.246997	10	1	0	
1	94.367043	10.134628	9	0	0	
2	84.248704	3.111158	8	0	0	
3	79.039486	0.991352	0	0	1	
4	68.128932	0.113940	0	0	0	
...	...	...	...	...	...	
995	38.790873	22.795105	0	1	0	
996	76.342998	8.435893	6	0	0	
997	114.457082	17.608176	3	1	0	
998	50.578190	22.432958	9	1	0	
999	90.942703	2.788098	8	0	0	

2000 rows × 7 columns

In [98]:
```
from sklearn.model_selection import train_test_split, cross_val_score
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline

X = churn_df.drop(columns=['Churn'], axis=1)
y = churn_df['Churn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
```

In [99]:
```
# Creating a validation Dataset

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_s
```

In [100]:
```
# Adopt cross validation technique.
# This technique splits the entire training set N times with each iteration re

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_r
from scipy.stats import randint

model = DecisionTreeClassifier()
cross_val_score(model, X_train, y_train, cv=7)
```

Out[100]:
```
array([0.75   , 0.78125, 0.7875 , 0.8125 , 0.775  , 0.7875 , 0.83125])
```

In [101]:
```
# Get accuracy on validation dataset

model.fit(X_train, y_train)
print(accuracy_score(y_valid, model.predict(X_valid)))
```

```
0.8321428571428572
```

In [102]:
```
model.feature_importances_
```

Out[102]:
```
array([0.40970096, 0.41509334, 0.12865842, 0.01995602, 0.01103526,
       0.015556  ])
```

In [103]:
```
def sortSecond(val):
    return val[1]
values = model.feature_importances_
features = list(X)
importances = [(features[i], values[i]) for i in range(len(features))]
importances.sort(reverse=True, key=sortSecond)
importances
```

Out[103]:
```
[('ServiceUsage', 0.41509333859706804),
 ('MonthlyCharges', 0.409700955987798),
 ('CustomerSupportCalls', 0.12865842276837866),
 ('Contract_Month-to-month', 0.019956024892302658),
 ('Contract_Two year', 0.01555599894671636),
 ('Contract_One year', 0.01103525880773609)]
```

In [104]:
```
# Considering Feature importance consider only the first 3 features and make a

X_train = X_train[[col[0] for col in importances[:3]]]
X_valid = X_valid[[col[0] for col in importances[:3]]]
cut_clf = DecisionTreeClassifier()
cut_clf.fit(X_train, y_train)
print(accuracy_score(y_valid, cut_clf.predict(X_valid)))
```

```
0.825
```

In [109]:
```
# RandomSearchCV to hyper-tune our model.

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

params_dist = {
    'criterion': ['gini', 'entropy'],
    'max_depth': randint(low=4, high=40),
    'max_leaf_nodes': randint(low=1000, high=20000),
    'min_samples_leaf': randint(low=20, high=100),
    'min_samples_split': randint(low=40, high=200)
}

clf_tuned = DecisionTreeClassifier(random_state=42)
random_search = RandomizedSearchCV(clf_tuned, params_dist, cv=7)
random_search.fit(X_train, y_train)
random_search.best_estimator_
```

Out[109]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=39, max_leaf_nodes=774
2,
                       min_samples_leaf=95, min_samples_split=178,
                       random_state=42)
```

In [110]:
```
best_tuned_clf = random_search.best_estimator_
best_tuned_clf.fit(X_train, y_train)
print(accuracy_score(y_valid, best_tuned_clf.predict(X_valid)))
```

```
0.7035714285714286
```

In [111]:
```
# Classification report based on the Feature Selection and Hyper Parameter Tun

from sklearn import metrics

print(metrics.classification_report(y_test, cut_clf.predict(X_test[[col[0] for
print(metrics.classification_report(y_test, best_tuned_clf.predict(X_test[[col
```

```
              precision    recall  f1-score   support

           0       0.90      0.86      0.88       436
           1       0.67      0.74      0.70       164

    accuracy                           0.83       600
   macro avg       0.78      0.80      0.79       600
weighted avg       0.84      0.83      0.83       600

              precision    recall  f1-score   support

           0       0.73      1.00      0.84       436
           1       0.00      0.00      0.00       164

    accuracy                           0.73       600
   macro avg       0.36      0.50      0.42       600
weighted avg       0.53      0.73      0.61       600


C:\Users\bhave\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bhave\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\bhave\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [112]:
```
# Accuracy With Hyper Parameters

best_tuned_clf.fit(X_train, y_train)

y_pred = best_tuned_clf.predict(X_test[[col[0] for col in importances[:3]]])

print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix: \n', confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.7266666666666667
Confusion Matrix:
 [[436   0]
 [164   0]]
```

In [113]:
```
# Accuracy With Feature Selection

cut_clf.fit(X_train, y_train)

y_pred = cut_clf.predict(X_test[[col[0] for col in importances[:3]]])

print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix: \n', confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.825
Confusion Matrix:
 [[374  62]
 [ 43 121]]
```