Churn Analysis Using k-Nearest Neighbor Classification

Douglas Ehlert

Churn Analysis Using k-Nearest Neighbor Classification

Part 1: Research Question

To maintain profits, it is critical for a telecommunication organization to retain customers for as long as possible. This helps to offset customer acquisition costs and reduced the spend needed on sales and marketing. This report will examine the effects of numerous variables on customer churn. The research question is: which variables have the highest effect on customer churn? The goal of the analysis is to enable the organization to target and interact with customers with the knowledge of what variables affect customer churn; after this analysis, the organization will be able to reduce customer churn.

Part II: Method Justification

K-Nearest Neighbors (KNN) classifier is a simple, yet powerful, tool in order to perform classification. KNN works on the basic assumption that "similar things are always in close proximity. This algorithm works by classifying the data points based on how the neighbors are classified" (*KNN Machine Learning Algorithm Explained - Springboard Blog*, 2019). The model will classify whether a customer will have a positive or false dependent variable, churn, by analyzing the training data set to determine a majority vote by the independent variables. One would expect that if the independent variables in the training dataset have indicated either a positive or negative churn result, similar independent variables will lead to similar churn results in the test dataset. List of imported libraries and packages:

- Pandas
  - Used for data manipulation such as import and of export of .csv files, data wrangling, analysis, and cleaning

- MatPlotLib
    - Used for data visualization (ROC-AUC Curve)
- Label Encoder/OneHotEncoder
    - Used to prep data. (Converting non-integer categorical variables to integers)
- KNeighborsClassifier
    - Used to instantiate the model for analysis
- Train_test_split
    - Used to split the dataset into training and test data
- Roc_auc_score
    - Used to plot the ROC-AUC score
- Sklearn Metrics
    - Used to find the AUC score

Part III: Data Preparation

The initial data preparation goal is to import the dataset and libraries/packages, perform an overview of the data (columns, datatype, etc.), and discover any missing values. The following four pictures show these steps in the code (Jupyter Notebook).

In [1]:
```
#import libraries
import matplotlib.pyplot as plt
import pandas as pd
#import encoder to deal with categorical variables
#import other libraries needed for sklearn module for data prep, kNN model, a
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
```

In [2]:
```
df = pd.read_csv('/Users/dougehlert/Desktop/D209/churn_clean.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	CaseOrder	Customer_id	Interaction	UID	City	Sta
0	1	K409198	aa90260b-4141-4a24-8e36-b04ce1f4f77b	e885b299883d4f9fb18e39c75155d990	Point Baker	
1	2	S120509	fb76459f-c047-4a9d-8af9-e0f7d4ac2524	f2de8bef964785f41a2959829830fb8a	West Branch	
2	3	K191035	344d114c-3736-4be5-98f7-c72c281e2d35	f1784cfa9f6d92ae816197eb175d3c71	Yamhill	
3	4	D90850	abfa2b40-2d43-4994-b15a-989b8c79e311	dc8a365077241bb5cd5ccd305136b05e	Del Mar	
4	5	K662701	68a861fd-0d20-4e51-a587-8a90407ee574	aabb64a116e83fdc4befc1fbab1663f9	Needville	

5 rows × 50 columns

In [4]:
```
dataset = df.drop(['Customer_id', 'Interaction', 'CaseOrder', 'Job','TimeZone
```

In [5]:
```
#This is a SUMMARY STATISTIC of the dataset.
#It shows various metrics for ALL the predictors.
dataset.describe()
```

Out[5]:

	Children	Age	Income	Outage_sec_perweek	Email	Co
count	10000.0000	10000.000000	10000.000000	10000.000000	10000.000000	10000.(
mean	2.0877	53.078400	39806.926771	10.001848	12.016000	0.(
std	2.1472	20.698882	28199.916702	2.976019	3.025898	0.(
min	0.0000	18.000000	348.670000	0.099747	1.000000	0.(
25%	0.0000	35.000000	19224.717500	8.018214	10.000000	0.(
50%	1.0000	53.000000	33170.605000	10.018560	12.000000	1.(
75%	3.0000	71.000000	53246.170000	11.969485	14.000000	2.(
max	10.0000	89.000000	258900.700000	21.207230	23.000000	7.(

In [6]:
```
#Check data for data types.
dataset.dtypes
```

Out[6]:
```
Area                     object
Children                  int64
Age                       int64
Income                  float64
Marital                  object
Gender                   object
Churn                    object
Outage_sec_perweek      float64
Email                     int64
Contacts                  int64
Yearly_equip_failure      int64
Techie                   object
Contract                 object
Port_modem               object
Tablet                   object
InternetService          object
Phone                    object
Multiple                 object
OnlineSecurity           object
OnlineBackup             object
DeviceProtection         object
TechSupport              object
StreamingTV              object
StreamingMovies          object
PaperlessBilling         object
PaymentMethod            object
Tenure                  float64
MonthlyCharge           float64
Bandwidth_GB_Year       float64
Item1                     int64
Item2                     int64
Item3                     int64
Item4                     int64
Item5                     int64
Item6                     int64
Item7                     int64
Item8                     int64
dtype: object
```

In [7]:
```
#confirm there are no null values
dataset.isna().any()
```

Out[7]:
```
Area                    False
Children                False
Age                     False
Income                  False
Marital                 False
Gender                  False
Churn                   False
Outage_sec_perweek      False
Email                   False
Contacts                False
Yearly_equip_failure    False
Techie                  False
Contract                False
Port_modem              False
Tablet                  False
InternetService         False
Phone                   False
Multiple                False
OnlineSecurity          False
OnlineBackup            False
DeviceProtection        False
TechSupport             False
StreamingTV             False
StreamingMovies         False
PaperlessBilling        False
PaymentMethod           False
Tenure                  False
MonthlyCharge           False
Bandwidth_GB_Year       False
Item1                   False
Item2                   False
Item3                   False
Item4                   False
Item5                   False
Item6                   False
Item7                   False
Item8                   False
dtype: bool
```

In [8]:
```
dataset["Churn"].value_counts()
```

Out[8]:
```
No     7350
Yes    2650
Name: Churn, dtype: int64
```

Part III: Data Preparation

Before analysis proceeds, dummy variables had to be created for the categorical variables, that are going to be used in the model, that were not integers. This is necessary for analysis as a KNN classifier would have no way of dealing with variables that consisted of different strings of text. Then the dummy variables are concatenated into the dataset. The target variable (dependent variable) is Churn. The predictor variables (independent variables) are, initially, Area, Children, Interaction, Job, Age, Income, Marital, Gender, Email, Outage_sec_perweek, Contacts, Yearly_equip_failure, Techie, Contract, Port_modem, Tablet, InternetService, Phone, Multiple, OnlineSecurity, OnlineBackup, TechSupport, StreamingTV, StreamingMovies, PaperlessBilling, PaymentMethod, Tenure, MonthlyCharge, Bandwidth_GB_Year, Item1, Item2, Item3, Item4, Item5, Item6, Item7, and Item8. Numerous variables were removed from the initial dataset because they were not pertinent to the analysis at hand. (CaseOrder, TimeZone, Customer_id, Job, Interaction, UID, City, State, County, Zip, Lat,Lng, Population). City, State, County, Zip, Lat, Lng, and Population were removed due existence of the variable Area. These variables would not be considered independent of Area.

The dependent variable, Churn, is assigned to ‘y’ and the independent variables are assigned to ‘X’. The shape of both ‘X’ and ‘y’ is checked to ensure that the length is equal. The cleaned, concatenated dataset, with dummy variables is exported as “ChurnClean2.csv”. Independent, continuous variables include Children, Age, Income, Outage_sec_perweek, Email, Contacts, Yearly_equip_failure, Tenure, MonthlyCharge, and Bandwidth_GB_Year. Independent, categorical variables include Techie, Port_modem, Tablet, Phone, Multiple, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies, PaperlessBilling, Item 1, Item 2, Item 3, Item 4, Item 5, Item 6, Item 7, Item 8, Rural, Suburban,

Urban, Female, Male, Nonbinary, Divorced, Married, Never Married, Separated, Widowed, DSL, Fiber Optic, None, BankTransfer(automatic), CreditCard(automatic), Electronic Check, Mailed Check, Month-to-month, One year, and two year. The dependent variable, which must be categorical for classification models, is Churn.

The following pictures show these steps, in the code, from Jupyter Notebook.

In [9]:
```
#label encoder object
le = LabelEncoder()
# Columns with one or two values can have label encoder run on them.
le_count = 0
for col in dataset.columns[1:]:
    if dataset[col].dtype == 'object':
        if len(list(dataset[col].unique())) <= 2:
            le.fit(dataset[col])
            dataset[col] = le.transform(dataset[col])
            le_count += 1
print('{} columns were label encoded.'.format(le_count))
```

```
13 columns were label encoded.
```

In [10]:
```
# Creating dummy variable for Gender
Gender_cat = pd.get_dummies(dataset['Gender'])

# Check what the dataset 'status' looks like
Gender_cat
```

Out[10]:

	Female	Male	Nonbinary
0	0	1	0
1	1	0	0
2	1	0	0
3	0	1	0
4	0	1	0
...	...	...	...
9995	0	1	0
9996	0	1	0
9997	1	0	0
9998	0	1	0
9999	0	1	0

10000 rows × 3 columns

In [11]:

```
#getdummies will be used to assign values and create new columns to the categ
# Creating dummy variable for Area
Area_cat = pd.get_dummies(dataset['Area'])

# Check what the dataset 'status' looks like
Area_cat
```

Out[11]:

	Rural	Suburban	Urban
0	0	0	1
1	0	0	1
2	0	0	1
3	0	1	0
4	0	1	0
...	...	...	...
9995	1	0	0
9996	1	0	0
9997	1	0	0
9998	0	0	1
9999	0	0	1

10000 rows × 3 columns

In [12]:

```
# Creating dummy variable for Marital
Marital_cat = pd.get_dummies(dataset['Marital'])

# Check what the dataset 'status' looks like
Marital_cat
```

Out[12]:

	Divorced	Married	Never Married	Separated	Widowed
0	0	0	0	0	1
1	0	1	0	0	0
2	0	0	0	0	1
3	0	1	0	0	0
4	0	0	0	1	0
...	...	...	...	...	...
9995	0	1	0	0	0
9996	1	0	0	0	0
9997	0	0	1	0	0
9998	0	0	0	1	0
9999	0	0	1	0	0

10000 rows × 5 columns

In [13]:

```
# Creating dummy variable for InternetService
InternetService_cat = pd.get_dummies(dataset['InternetService'])

# Check what the dataset looks like
InternetService_cat
```

Out[13]:

	DSL	Fiber Optic	None
0	0	1	0
1	0	1	0
2	1	0	0
3	1	0	0
4	0	1	0
...	...	...	...
9995	1	0	0
9996	0	1	0
9997	0	1	0
9998	0	1	0
9999	0	1	0

10000 rows × 3 columns

In [14]:

```
# Creating dummy variable for PaymentMethod
PaymentMethod_cat = pd.get_dummies(dataset['PaymentMethod'])

# Check what the dataset 'status' looks like
PaymentMethod_cat
```

Out[14]:

	Bank Transfer(automatic)	Credit Card (automatic)	Electronic Check	Mailed Check
0	0	1	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	0	1
4	0	0	0	1
...	...	...	...	...
9995	0	0	1	0
9996	0	0	1	0
9997	1	0	0	0
9998	0	1	0	0
9999	0	0	1	0

10000 rows × 4 columns

In [15]:

```
# Creating dummy variable for Contract
Contract_cat = pd.get_dummies(dataset['Contract'])

# Check what the dataset 'status' looks like
Contract_cat
```

Out[15]:

	Month-to-month	One year	Two Year
0	0	1	0
1	1	0	0
2	0	0	1
3	0	0	1
4	1	0	0
...	...	...	...
9995	1	0	0
9996	0	0	1
9997	1	0	0
9998	0	0	1
9999	1	0	0

10000 rows × 3 columns

In [16]:

```
#Combining the dummy variables into the dataset
dataset = pd.concat([dataset, Area_cat], axis = 1)
dataset
```

Out[16]:

	Area	Children	Age	Income	Marital	Gender	Churn	Outage_sec_perweek	Er
0	Urban	0	68	28561.99	Widowed	Male	0	7.978323	
1	Urban	1	27	21704.77	Married	Female	1	11.699080	
2	Urban	4	50	9609.57	Widowed	Female	0	10.752800	
3	Suburban	1	48	18925.23	Married	Male	0	14.913540	
4	Suburban	0	83	40074.19	Separated	Male	1	8.147417	
...	...	...	...	...	...	...	...	...	
9995	Rural	3	23	55723.74	Married	Male	0	9.415935	
9996	Rural	4	48	34129.34	Divorced	Male	0	6.740547	
9997	Rural	1	48	45983.43	Never Married	Female	0	6.590911	
9998	Urban	1	39	16667.58	Separated	Male	0	12.071910	
9999	Urban	1	28	9020.92	Never Married	Male	0	11.754720	

10000 rows × 40 columns

In [17]:

```
#Combining the dummy variables into the dataset
dataset = pd.concat([dataset, Gender_cat], axis = 1)
dataset
```

Out[17]:

	Area	Children	Age	Income	Marital	Gender	Churn	Outage_sec_perweek	Er
0	Urban	0	68	28561.99	Widowed	Male	0	7.978323	
1	Urban	1	27	21704.77	Married	Female	1	11.699080	
2	Urban	4	50	9609.57	Widowed	Female	0	10.752800	
3	Suburban	1	48	18925.23	Married	Male	0	14.913540	
4	Suburban	0	83	40074.19	Separated	Male	1	8.147417	
...	...	...	...	...	...	...	...	...	
9995	Rural	3	23	55723.74	Married	Male	0	9.415935	
9996	Rural	4	48	34129.34	Divorced	Male	0	6.740547	
9997	Rural	1	48	45983.43	Never Married	Female	0	6.590911	
9998	Urban	1	39	16667.58	Separated	Male	0	12.071910	
9999	Urban	1	28	9020.92	Never Married	Male	0	11.754720	

10000 rows × 43 columns

In [18]:

```
#Combining the dummy variables into the dataset
dataset = pd.concat([dataset, Marital_cat], axis = 1)
dataset
```

Out[18]:

	Area	Children	Age	Income	Marital	Gender	Churn	Outage_sec_perweek	Er
0	Urban	0	68	28561.99	Widowed	Male	0	7.978323	
1	Urban	1	27	21704.77	Married	Female	1	11.699080	
2	Urban	4	50	9609.57	Widowed	Female	0	10.752800	
3	Suburban	1	48	18925.23	Married	Male	0	14.913540	
4	Suburban	0	83	40074.19	Separated	Male	1	8.147417	
...	...	...	...	...	...	...	...	...	
9995	Rural	3	23	55723.74	Married	Male	0	9.415935	
9996	Rural	4	48	34129.34	Divorced	Male	0	6.740547	
9997	Rural	1	48	45983.43	Never Married	Female	0	6.590911	
9998	Urban	1	39	16667.58	Separated	Male	0	12.071910	
9999	Urban	1	28	9020.92	Never Married	Male	0	11.754720	

10000 rows × 48 columns

In [19]:

```
#Combining the dummy variables into the dataset
dataset = pd.concat([dataset, InternetService_cat], axis = 1)
dataset
```

Out[19]:

	Area	Children	Age	Income	Marital	Gender	Churn	Outage_sec_perweek	Er
0	Urban	0	68	28561.99	Widowed	Male	0	7.978323	
1	Urban	1	27	21704.77	Married	Female	1	11.699080	
2	Urban	4	50	9609.57	Widowed	Female	0	10.752800	
3	Suburban	1	48	18925.23	Married	Male	0	14.913540	
4	Suburban	0	83	40074.19	Separated	Male	1	8.147417	
...	...	...	...	...	...	...	...	...	
9995	Rural	3	23	55723.74	Married	Male	0	9.415935	
9996	Rural	4	48	34129.34	Divorced	Male	0	6.740547	
9997	Rural	1	48	45983.43	Never Married	Female	0	6.590911	
9998	Urban	1	39	16667.58	Separated	Male	0	12.071910	
9999	Urban	1	28	9020.92	Never Married	Male	0	11.754720	

10000 rows × 51 columns

In [20]:

```
#Combining the dummy variables into the dataset
dataset = pd.concat([dataset, PaymentMethod_cat], axis = 1)
dataset
```

Out[20]:

	Area	Children	Age	Income	Marital	Gender	Churn	Outage_sec_perweek	Er
0	Urban	0	68	28561.99	Widowed	Male	0	7.978323	
1	Urban	1	27	21704.77	Married	Female	1	11.699080	
2	Urban	4	50	9609.57	Widowed	Female	0	10.752800	
3	Suburban	1	48	18925.23	Married	Male	0	14.913540	
4	Suburban	0	83	40074.19	Separated	Male	1	8.147417	
...	...	...	...	...	...	...	...	...	
9995	Rural	3	23	55723.74	Married	Male	0	9.415935	
9996	Rural	4	48	34129.34	Divorced	Male	0	6.740547	
9997	Rural	1	48	45983.43	Never Married	Female	0	6.590911	
9998	Urban	1	39	16667.58	Separated	Male	0	12.071910	
9999	Urban	1	28	9020.92	Never Married	Male	0	11.754720	

10000 rows × 55 columns

In [21]:

```
#Combining the dummy variables into the dataset
dataset = pd.concat([dataset, Contract_cat], axis = 1)
dataset
```

Out[21]:

	Area	Children	Age	Income	Marital	Gender	Churn	Outage_sec_perweek	Er
0	Urban	0	68	28561.99	Widowed	Male	0	7.978323	
1	Urban	1	27	21704.77	Married	Female	1	11.699080	
2	Urban	4	50	9609.57	Widowed	Female	0	10.752800	
3	Suburban	1	48	18925.23	Married	Male	0	14.913540	
4	Suburban	0	83	40074.19	Separated	Male	1	8.147417	
...	...	...	...	...	...	...	...	...	
9995	Rural	3	23	55723.74	Married	Male	0	9.415935	
9996	Rural	4	48	34129.34	Divorced	Male	0	6.740547	
9997	Rural	1	48	45983.43	Never Married	Female	0	6.590911	
9998	Urban	1	39	16667.58	Separated	Male	0	12.071910	
9999	Urban	1	28	9020.92	Never Married	Male	0	11.754720	

10000 rows × 58 columns

In [22]:

```
#drop area, gender, marital
dataset.drop(['Area', 'Gender', 'Marital', 'Contract','PaymentMethod', 'Inter
dataset
```

Out[22]:

	Children	Age	Income	Churn	Outage_sec_perweek	Email	Contacts	Yearly_equip_fa
0	0	68	28561.99	0	7.978323	10	0	
1	1	27	21704.77	1	11.699080	12	0	
2	4	50	9609.57	0	10.752800	9	0	
3	1	48	18925.23	0	14.913540	15	2	
4	0	83	40074.19	1	8.147417	16	2	
...	...	...	...	...	...	...	...	
9995	3	23	55723.74	0	9.415935	12	2	
9996	4	48	34129.34	0	6.740547	15	2	
9997	1	48	45983.43	0	6.590911	10	0	
9998	1	39	16667.58	0	12.071910	14	1	
9999	1	28	9020.92	0	11.754720	17	1	

10000 rows × 52 columns

In [23]:
```
dataset.shape
```

Out[23]: (10000, 52)

In [24]:
```
#Export dataset to save as a checkpoint.
dataset.to_csv(r'churnClean2.csv', index = False)
```

In [25]:
```
y=dataset['Churn'].values
```

In [26]:
```
y.shape
```

Out[26]: (10000,)

In [27]:
```
X=dataset.drop(columns=['Churn'])
```

In [28]:
```
X.shape
```

Out[28]: (10000, 51)

Part IV: Analysis

The dataset was then split into train and test sets to prepare for the kNN Classifier. The data was fit to the training set and prediction run on the 'X_test' data. Next, a KNN score was found, resulting in a score of .72. This essentially means that the classification model is much better than random guessing (random guessing has a kNN score of .50).

In [29]:
```
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.3, random
```

In [30]:
```
knn=KNeighborsClassifier(n_neighbors =6)
```

In [31]:
```
knn.fit(X_train,y_train)
```

Out[31]: KNeighborsClassifier(n_neighbors=6)

In [32]:
```
y_pred=knn.predict(X_test)
```

In [33]:
```
print('Test set predictions:\in{}'.format(y_pred))
```

```
Test set predictions:\in[0 0 0 ... 0 0 0]
```

In [34]:
```
knn.score(X_test, y_test)
```

Out[34]: 0.7203333333333334

In [35]:
```
y_pred=knn.predict(X)
```

In [36]:
```
y_pred_prob=knn.predict_proba(X_test)[:,1]
```

In [37]:
```
roc_auc_score(y_test, y_pred_prob)
```

Out[37]: 0.7372013291689842

In [38]:
```
from sklearn.metrics import roc_curve
from sklearn import metrics
```

In [39]:
```
fpr, tpr, thresholds=roc_curve(y_test, y_pred_prob)
```

Part V: Data Summary and Implications

Lines 36-41 of the code demonstrate the ROC-AUC curve. The model's AUC score 0.737. Using the same criteria as the kNN score, this classifier scores halfway between random guessing and perfection (which would be a AUC score of 1). The implications of this analysis revolve around the fact that this organization can predict whether a customer will churn using the independent variables described in this analysis. One limitation of this analysis is the potential presence of other variables, not in the dataset, that can affect customer churn. This organization should continue to develop programs around these independent variables that will lead to a decrease in customer churn (i.e. higher survey scores, better service, etc.). Eventually, this analysis could be used to predict whether a particular customer will churn, and that customer could be put into a specialized "retention" program to decrease the likelihood of churn. The following screenshots show the Python code discussed in this paragraph and the ROC-AUC Curve.

In [29]:
```
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.3, random
```

In [30]:
```
knn=KNeighborsClassifier(n_neighbors =6)
```

In [31]:
```
knn.fit(X_train,y_train)
```

Out[31]:
```
KNeighborsClassifier(n_neighbors=6)
```

In [32]:
```
y_pred=knn.predict(X_test)
```

In [33]:
```
print('Test set predictions:\in{}'.format(y_pred))
```

```
Test set predictions:\in[0 0 0 ... 0 0 0]
```

In [34]:
```
knn.score(X_test, y_test)
```

Out[34]:
```
0.7203333333333334
```

In [35]:
```
y_pred=knn.predict(X)
```

In [36]:
```
y_pred_prob=knn.predict_proba(X_test)[:,1]
```

In [37]:
```
roc_auc_score(y_test, y_pred_prob)
```

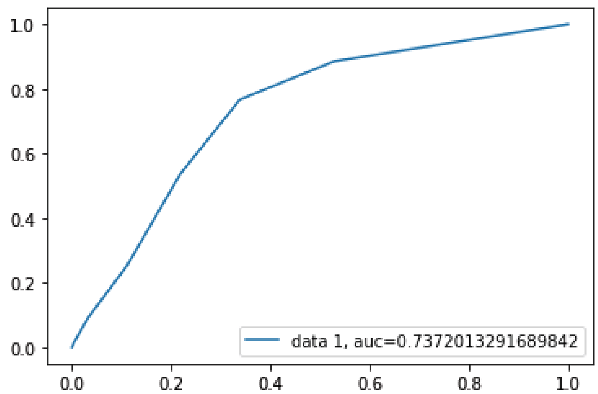
Out[37]:
```
0.7372013291689842
```

In [38]:
```
from sklearn.metrics import roc_curve
from sklearn import metrics
```

In [39]:
```
fpr, tpr, thresholds=roc_curve(y_test, y_pred_prob)
```

In [40]:

```
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_prob)
auc = metrics.roc_auc_score(y_test, y_pred_prob)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

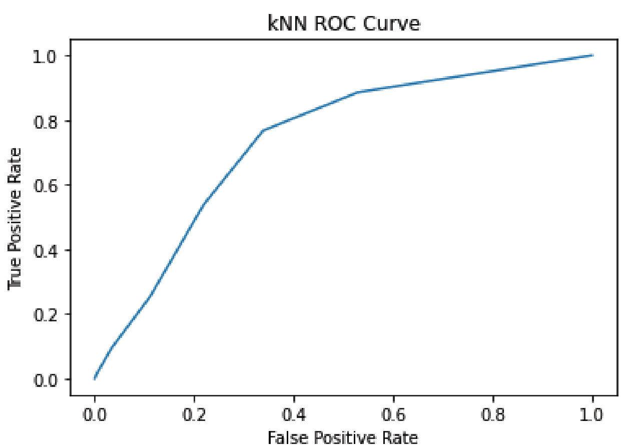


In [41]:

```
plt.plot(fpr, tpr, label='kNN')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('kNN ROC Curve')
plt.show
```

Out[41]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



References

KNN Machine Learning Algorithm Explained - Springboard Blog. (2019, May 29). Springboard Blog. https://in.springboard.com/blog/knn-machine-learning-algorithm-explained/