WGU D209 Predictive Modeling

Task 1 - Classification Analysis

Ednalyn C. De Dios

August 9, 2023

Part I - Research Question

A1. Propose one relevant question and choose a classification method.

> ***Which customers are most likely to churn?***

We will endeavor to answer the above question by using the KNN (K-nearest neighbor) method.

A2. Define one goal of the data analysis.

The ultimate goal is to reduce operating costs. We will build a KNN model to predict which customers are likely to churn. The organization will benefit from knowing which customers are most likely to churn or discontinue service. This will inform the decisions of stakeholders in matters where customer retention is involved. For example, it takes a lot more money to acquire new customers than to retain existing ones. Knowing which customers are at a high risk of churn will provide the organization with advanced warnings to prevent churn. This will surely reduce the operating costs of the organization.

This goal is within the scope of the scenario and is represented in the available data.

Part II - Method Justification

B1. Explain how the classification method you chose analyzes the selected data set. Include expected outcomes.

I chose KNN because it is non-parametric, which means it does not make any assumptions about data distribution. Furthermore, it also does not make any generalizations about the data but simply checks the neighboring data points to determine which class they belong in.

It looks at the distance between data points and determines that the close the data points are, the more similar they are. (Sharma, 2021).

I expect the model to show the relationship between the target variable "churn" and the k-neighbors. I also expect a summary of the model's performance, such as accuracy and AUC scores.

B2. Summarize one assumption of the chosen classification method.

The core assumption of KNN is that "The closer two given points are to each other, the more related and similar they are." (Neptune.ai, 2023).

B3. List the packages or libraries you have chosen for Python or R, and justify how each item on the list supports the analysis.

The packages/libraries I've chosen are:

- sys: for running the executable file (d209_task1.py)
- random: set random seed for reproducibility of the experiment
- pandas: for manipulating dataframes
- numpy: for performing mathematical computations
- matplotlib and seaborn: for visualizations
- Image: display image from a URL
- StandardScaler: to scale the data set
- train_test_split: to split the data into train and test sets
- KNeighborsClassifier: machine learning model
- roc_curve, confusion_matrix, roc_auc_score: display metrics
- GridSearchCV: to optimize the model

Part III - Data Preparation

C1. Describe one data preprocessing goal relevant to the classification method from part A1.

One preprocessing goal is to encode categorical variables with values using dummy variables. Most machine learning models require the use of numerical data. For binary columns, we will change "Yes" to 1 and "No" to 0. Similarly, we will convert True to 1 and False to 0. We will utilize the get_dummies() method to accomplish this goal.

C2. Identify the initial data set variables that you will use to analyze the classification question from part A1, and classify each variable as continuous or categorical.

Independent Variables:

- Children, Continuous, Quantitative
- Age, Continuous, Quantitative
- Income, Continuous, Quantitative
- Marital, Categorical, Qualitative
- Gender, Categorical, Qualitative
- Outage_sec_perweek, Continuous, Quantitative
- Email, Continuous, Quantitative
- Contacts, Continuous, Quantitative
- Yearly_equip_failure, Categorical, Qualitative
- Techie, Categorical, Qualitative
- Contract, Categorical, Qualitative
- Port_modem, Categorical, Qualitative
- Tablet, Categorical, Qualitative
- InternetService, Categorical, Qualitative
- Phone, Categorical, Qualitative
- Multiple, Categorical, Qualitative
- OnlineSecurity, Categorical, Qualitative
- OnlineBackup, Categorical, Qualitative
- DeviceProtection, Categorical, Qualitative
- TechSupport, Categorical, Qualitative
- StreamingTV, Categorical, Qualitative
- StreamingMovies, Categorical, Qualitative
- PaperlessBilling, Categorical, Qualitative
- PaymentMethod, Categorical, Qualitative
- Tenure, Continuous, Quantitative
- MonthlyCharge, Continuous, Quantitative
- Bandwidth_GB_Year, Continuous, Quantitative

Dependent/Target Variable:

- Churn, Categorical, Qualitative

C3. Explain each of the steps used to prepare the data for the analysis. Identify the code segment for each step.

- Load and view the data

  > use pandas to read the csv use head or tail methods use info()

- Perform data cleaning

  > remove duplicates, missing values, outliers

- Remove irrelevant data for the analysis

  > use the drop method

- Pre-process data

  > encode categorical data, scale the features

In [1]:
```
# setting the random seed for reproducibility
import random
random.seed(493)

# for manipulating dataframes
import pandas as pd
import numpy as np

# for visualizations
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")
from IPython.display import Image

# for modeling
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

# to print out all the outputs of the cell
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# set display options
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)
```

In [2]:
```
# read the csv file
df = pd.read_csv('churn_clean.csv')
```

```
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CaseOrder             10000 non-null  int64  
 1   Customer_id           10000 non-null  object 
 2   Interaction           10000 non-null  object 
 3   UID                   10000 non-null  object 
 4   City                  10000 non-null  object 
 5   State                 10000 non-null  object 
 6   County                10000 non-null  object 
 7   Zip                   10000 non-null  int64  
 8   Lat                   10000 non-null  float64
 9   Lng                   10000 non-null  float64
 10  Population            10000 non-null  int64  
 11  Area                  10000 non-null  object 
 12  TimeZone              10000 non-null  object 
 13  Job                   10000 non-null  object 
 14  Children              10000 non-null  int64  
 15  Age                   10000 non-null  int64  
 16  Income                10000 non-null  float64
 17  Marital               10000 non-null  object 
 18  Gender                10000 non-null  object 
 19  Churn                 10000 non-null  object 
 20  Outage_sec_perweek    10000 non-null  float64
 21  Email                 10000 non-null  int64  
 22  Contacts              10000 non-null  int64  
 23  Yearly_equip_failure  10000 non-null  int64  
 24  Techie                10000 non-null  object 
 25  Contract              10000 non-null  object 
 26  Port_modem            10000 non-null  object 
 27  Tablet                10000 non-null  object 
 28  InternetService       7871 non-null   object 
 29  Phone                 10000 non-null  object 
 30  Multiple              10000 non-null  object 
 31  OnlineSecurity        10000 non-null  object 
 32  OnlineBackup          10000 non-null  object 
 33  DeviceProtection      10000 non-null  object 
 34  TechSupport           10000 non-null  object 
 35  StreamingTV           10000 non-null  object 
 36  StreamingMovies       10000 non-null  object 
 37  PaperlessBilling      10000 non-null  object 
 38  PaymentMethod         10000 non-null  object 
 39  Tenure                10000 non-null  float64
 40  MonthlyCharge         10000 non-null  float64
 41  Bandwidth_GB_Year     10000 non-null  float64
 42  Item1                 10000 non-null  int64  
 43  Item2                 10000 non-null  int64  
 44  Item3                 10000 non-null  int64  
 45  Item4                 10000 non-null  int64  
 46  Item5                 10000 non-null  int64  
 47  Item6                 10000 non-null  int64  
 48  Item7                 10000 non-null  int64  
 49  Item8                 10000 non-null  int64  
```

```
dtypes: float64(7), int64(16), object(27)
memory usage: 3.8+ MB
```

Out[2]:

	CaseOrder	Customer_id	Interaction	UID	City	S
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	

In [3]:

```python
# select rows that are duplicated based on all columns
dup = df[df.duplicated()]

# find out how many rows are duplicated
dup.shape
```

Out[3]: (0, 50)

In [4]:

```python
def show_missing(df):
    """
    Takes a dataframe and returns a dataframe with stats
    on missing and null values with their percentages.
    """
    null_count = df.isnull().sum()
    null_percentage = (null_count / df.shape[0]) * 100
    empty_count = pd.Series(((df == ' ') | (df == '')).sum())
    empty_percentage = (empty_count / df.shape[0]) * 100
    nan_count = pd.Series(((df == 'nan') | (df == 'NaN')).sum())
    nan_percentage = (nan_count / df.shape[0]) * 100
    dfx = pd.DataFrame({'num_missing': null_count, 'missing_percentage': null_perce
                        'num_empty': empty_count, 'empty_percentage': empty_percen
                        'nan_count': nan_count, 'nan_percentage': nan_percentage})
    return dfx

show_missing(df)
```

Out[4]:

	num_missing	missing_percentage	num_empty	empty_percentage	na
CaseOrder	0	0.00	0	0.0	
Customer_id	0	0.00	0	0.0	
Interaction	0	0.00	0	0.0	
UID	0	0.00	0	0.0	
City	0	0.00	0	0.0	
State	0	0.00	0	0.0	
County	0	0.00	0	0.0	
Zip	0	0.00	0	0.0	
Lat	0	0.00	0	0.0	
Lng	0	0.00	0	0.0	
Population	0	0.00	0	0.0	
Area	0	0.00	0	0.0	
TimeZone	0	0.00	0	0.0	
Job	0	0.00	0	0.0	
Children	0	0.00	0	0.0	
Age	0	0.00	0	0.0	
Income	0	0.00	0	0.0	
Marital	0	0.00	0	0.0	
Gender	0	0.00	0	0.0	
Churn	0	0.00	0	0.0	
Outage_sec_perweek	0	0.00	0	0.0	
Email	0	0.00	0	0.0	
Contacts	0	0.00	0	0.0	
Yearly_equip_failure	0	0.00	0	0.0	
Techie	0	0.00	0	0.0	
Contract	0	0.00	0	0.0	
Port_modem	0	0.00	0	0.0	
Tablet	0	0.00	0	0.0	
InternetService	2129	21.29	0	0.0	
Phone	0	0.00	0	0.0	

	num_missing	missing_percentage	num_empty	empty_percentage	na
Multiple	0	0.00	0	0.0	
OnlineSecurity	0	0.00	0	0.0	
OnlineBackup	0	0.00	0	0.0	
DeviceProtection	0	0.00	0	0.0	
TechSupport	0	0.00	0	0.0	
StreamingTV	0	0.00	0	0.0	
StreamingMovies	0	0.00	0	0.0	
PaperlessBilling	0	0.00	0	0.0	
PaymentMethod	0	0.00	0	0.0	
Tenure	0	0.00	0	0.0	
MonthlyCharge	0	0.00	0	0.0	
Bandwidth_GB_Year	0	0.00	0	0.0	
Item1	0	0.00	0	0.0	
Item2	0	0.00	0	0.0	
Item3	0	0.00	0	0.0	
Item4	0	0.00	0	0.0	
Item5	0	0.00	0	0.0	
Item6	0	0.00	0	0.0	
Item7	0	0.00	0	0.0	
Item8	0	0.00	0	0.0	

In [5]:

```
# handle missing values
df['InternetService'].value_counts()
```

Out[5]:

```
InternetService
Fiber Optic    4408
DSL            3463
Name: count, dtype: int64
```

In [6]:

```
# fill missing values with None as in no service
df = df.fillna("None")

df['InternetService'].value_counts()
show_missing(df)
```

Out[6]:
```
InternetService
Fiber Optic    4408
DSL            3463
None           2129
Name: count, dtype: int64
```

Out[6]:

	num_missing	missing_percentage	num_empty	empty_percentage	na
CaseOrder	0	0.0	0	0.0	
Customer_id	0	0.0	0	0.0	
Interaction	0	0.0	0	0.0	
UID	0	0.0	0	0.0	
City	0	0.0	0	0.0	
State	0	0.0	0	0.0	
County	0	0.0	0	0.0	
Zip	0	0.0	0	0.0	
Lat	0	0.0	0	0.0	
Lng	0	0.0	0	0.0	
Population	0	0.0	0	0.0	
Area	0	0.0	0	0.0	
TimeZone	0	0.0	0	0.0	
Job	0	0.0	0	0.0	
Children	0	0.0	0	0.0	
Age	0	0.0	0	0.0	
Income	0	0.0	0	0.0	
Marital	0	0.0	0	0.0	
Gender	0	0.0	0	0.0	
Churn	0	0.0	0	0.0	
Outage_sec_perweek	0	0.0	0	0.0	
Email	0	0.0	0	0.0	
Contacts	0	0.0	0	0.0	
Yearly_equip_failure	0	0.0	0	0.0	
Techie	0	0.0	0	0.0	
Contract	0	0.0	0	0.0	
Port_modem	0	0.0	0	0.0	
Tablet	0	0.0	0	0.0	
InternetService	0	0.0	0	0.0	
Phone	0	0.0	0	0.0	

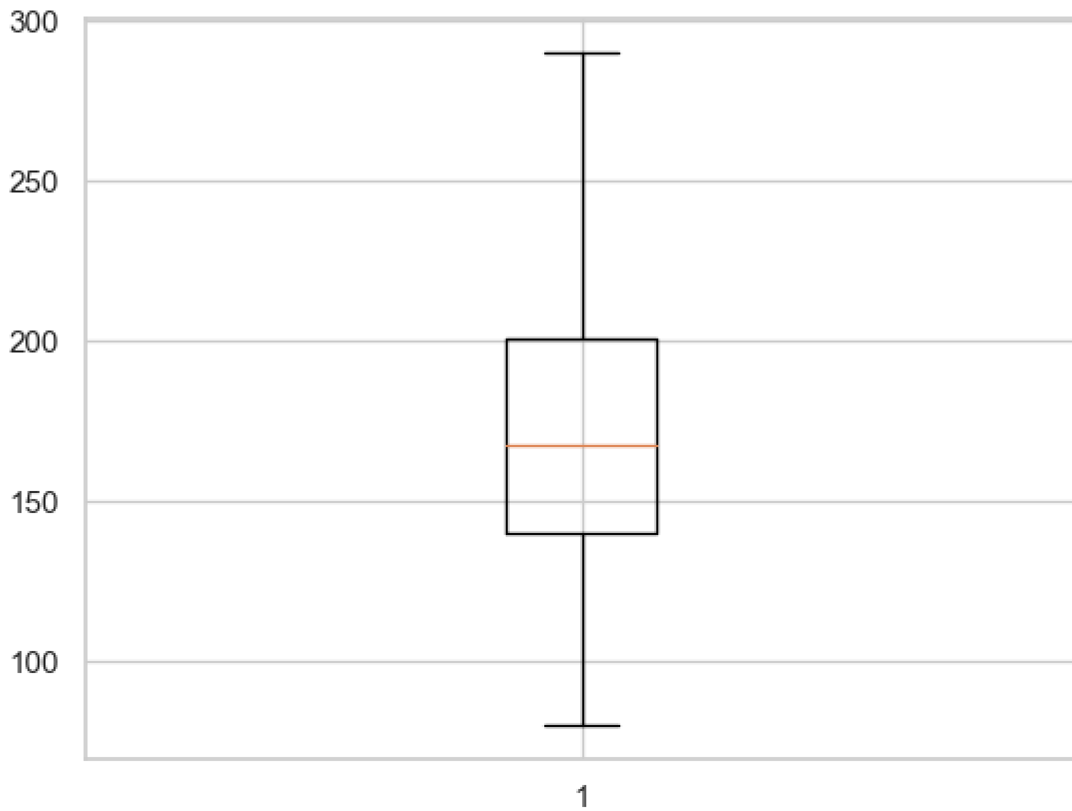
	num_missing	missing_percentage	num_empty	empty_percentage	na
Multiple	0	0.0	0	0.0	
OnlineSecurity	0	0.0	0	0.0	
OnlineBackup	0	0.0	0	0.0	
DeviceProtection	0	0.0	0	0.0	
TechSupport	0	0.0	0	0.0	
StreamingTV	0	0.0	0	0.0	
StreamingMovies	0	0.0	0	0.0	
PaperlessBilling	0	0.0	0	0.0	
PaymentMethod	0	0.0	0	0.0	
Tenure	0	0.0	0	0.0	
MonthlyCharge	0	0.0	0	0.0	
Bandwidth_GB_Year	0	0.0	0	0.0	
Item1	0	0.0	0	0.0	
Item2	0	0.0	0	0.0	
Item3	0	0.0	0	0.0	
Item4	0	0.0	0	0.0	
Item5	0	0.0	0	0.0	
Item6	0	0.0	0	0.0	
Item7	0	0.0	0	0.0	
Item8	0	0.0	0	0.0	

In [7]:

```
# remove outliers
plt.boxplot(df['MonthlyCharge'])
fig = plt.figure(figsize =(10, 7))
```

Out[7]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x25b83a55a60>,
  <matplotlib.lines.Line2D at 0x25b83a55d00>],
 'caps': [<matplotlib.lines.Line2D at 0x25b83a55fa0>,
  <matplotlib.lines.Line2D at 0x25b83a6c280>],
 'boxes': [<matplotlib.lines.Line2D at 0x25b83a557c0>],
 'medians': [<matplotlib.lines.Line2D at 0x25b83a6c520>],
 'fliers': [<matplotlib.lines.Line2D at 0x25b83a6c7c0>],
 'means': []}
```



<Figure size 1000x700 with 0 Axes>

In [8]:
```
# remove irrelevant data
df.drop(columns=['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State',
       'County', 'Zip', 'Lat', 'Lng', 'Population', 'Area', 'TimeZone', 'Job',
        'Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6', 'Item7', 'Item8'],
        inplace=True)
```

In [9]:
```
# encode categorical data

# assemble list of categorical columns to generate dummy variables for
dummy_columns = ['Marital',
                 'Gender',
                 'Techie',
                 'Contract',
                 'Port_modem',
                 'Tablet',
                 'InternetService',
                 'Phone',
                 'Multiple',
                 'OnlineSecurity',
                 'OnlineBackup',
                 'DeviceProtection',
                 'TechSupport',
                 'StreamingTV',
                 'StreamingMovies',
                 'PaperlessBilling',
                 'PaymentMethod'
                ]
```

In [10]:
```python
def dummify(df, column):
    """
    Takes a dataframe and column to return a dataframe with
    dummy variables appended.
    """
    dummy = pd.get_dummies(df[column], prefix=column, prefix_sep='_',)
    return pd.concat([df, dummy], axis=1)
```

In [11]:
```python
dummified = df.copy()

# loop through all the columns tp generate dummy for
for col in dummy_columns:
    dummified = dummify(dummified, col)
```

In [12]:
```python
dummified.head()
```

Out[12]:

	Children	Age	Income	Marital	Gender	Churn	Outage_sec_perweek	Email	Contac
0	0	68	28561.99	Widowed	Male	No	7.978323	10	
1	1	27	21704.77	Married	Female	Yes	11.699080	12	
2	4	50	9609.57	Widowed	Female	No	10.752800	9	
3	1	48	18925.23	Married	Male	No	14.913540	15	
4	0	83	40074.19	Separated	Male	Yes	8.147417	16	

In [13]:
```python
# drop original columns we generated dummies for
dummified.drop(columns=dummy_columns, inplace=True)
dummified.head()
```

Out[13]:

	Children	Age	Income	Churn	Outage_sec_perweek	Email	Contacts	Yearly_equip_fail
0	0	68	28561.99	No	7.978323	10	0	
1	1	27	21704.77	Yes	11.699080	12	0	
2	4	50	9609.57	No	10.752800	9	0	
3	1	48	18925.23	No	14.913540	15	2	
4	0	83	40074.19	Yes	8.147417	16	2	

In [14]:
```
# move target variable at the end of the dataframe
df = dummified[['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email',
       'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge',
       'Bandwidth_GB_Year', 'Marital_Divorced', 'Marital_Married',
       'Marital_Never Married', 'Marital_Separated', 'Marital_Widowed',
       'Gender_Female', 'Gender_Male', 'Gender_Nonbinary', 'Techie_No',
       'Techie_Yes', 'Contract_Month-to-month', 'Contract_One year',
       'Contract_Two Year', 'Port_modem_No', 'Port_modem_Yes', 'Tablet_No',
       'Tablet_Yes', 'InternetService_DSL', 'InternetService_Fiber Optic',
       'InternetService_None', 'Phone_No', 'Phone_Yes', 'Multiple_No',
       'Multiple_Yes', 'OnlineSecurity_No', 'OnlineSecurity_Yes',
       'OnlineBackup_No', 'OnlineBackup_Yes', 'DeviceProtection_No',
       'DeviceProtection_Yes', 'TechSupport_No', 'TechSupport_Yes',
       'StreamingTV_No', 'StreamingTV_Yes', 'StreamingMovies_No',
       'StreamingMovies_Yes', 'PaperlessBilling_No', 'PaperlessBilling_Yes',
       'PaymentMethod_Bank Transfer(automatic)',
       'PaymentMethod_Credit Card (automatic)',
       'PaymentMethod_Electronic Check', 'PaymentMethod_Mailed Check', 'Churn']]
```

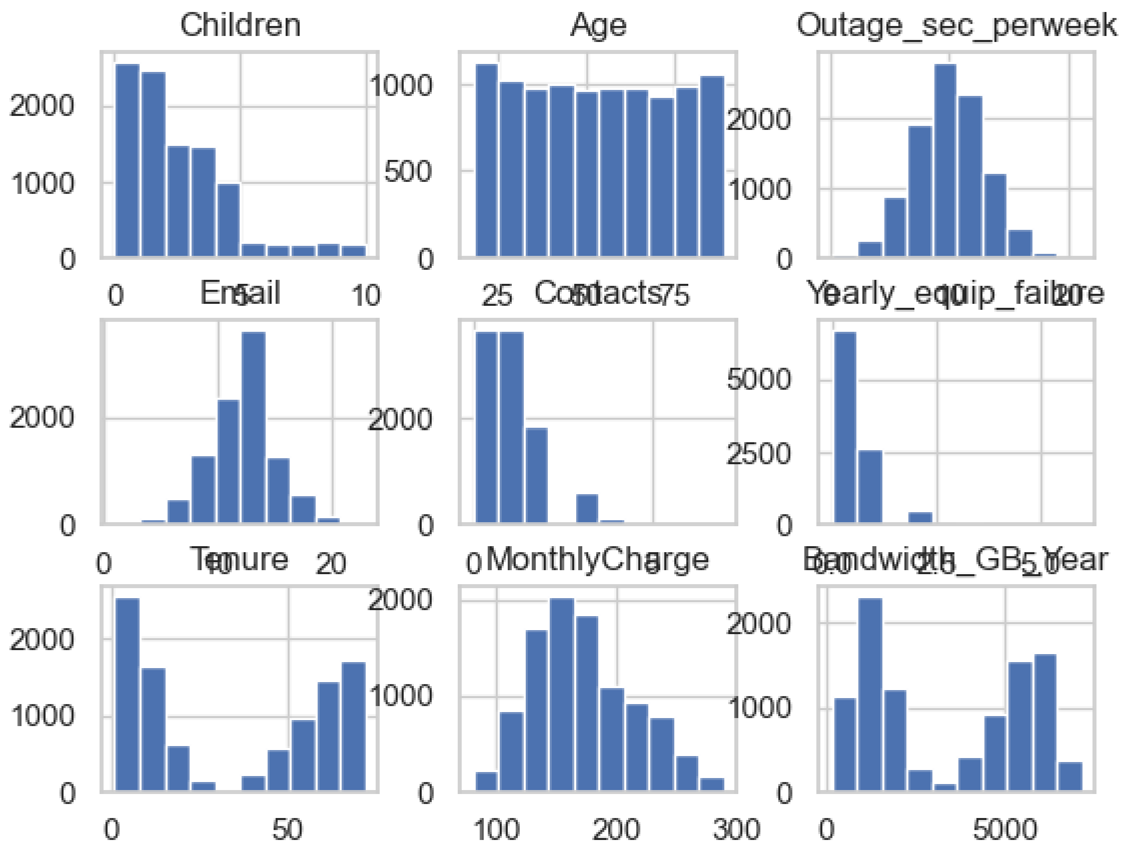
In [15]:
```
# replace True with 1's and False with 0's
df = df.replace(True, 1)
df = df.replace(False, 0)

# replace 'Yes' with 1's and 'No' with 0's
df['Churn'] = df['Churn'].replace('Yes', 1)
df['Churn'] = df['Churn'].replace('No', 0)

df.head()
```

Out[15]:

	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	
0	0	68	28561.99	7.978323	10	0	1	6.
1	1	27	21704.77	11.699080	12	0	1	1.
2	4	50	9609.57	10.752800	9	0	1	15.
3	1	48	18925.23	14.913540	15	2	0	17.
4	0	83	40074.19	8.147417	16	2	1	1.

In [16]:
```
# make historgrams and save the plot
df[['Children',
    'Age',
    'Outage_sec_perweek',
    'Email',
    'Contacts',
    'Yearly_equip_failure',
    'Tenure',
    'MonthlyCharge',
    'Bandwidth_GB_Year'
   ]].hist()
```

Out[16]:
```
array([[<Axes: title={'center': 'Children'}>,
        <Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Outage_sec_perweek'}>],
       [<Axes: title={'center': 'Email'}>,
        <Axes: title={'center': 'Contacts'}>,
        <Axes: title={'center': 'Yearly_equip_failure'}>],
       [<Axes: title={'center': 'Tenure'}>,
        <Axes: title={'center': 'MonthlyCharge'}>,
        <Axes: title={'center': 'Bandwidth_GB_Year'}>]], dtype=object)
```



In [17]:
```
# scale the data
scaler = StandardScaler()

# apply scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Children', 'Age', 'Outage_sec_perweek', 'Email', 'Contacts','Yearly_eq
df[num_vars] = scaler.fit_transform(df[num_vars])

df.head()
```

Out[17]:

	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_f
0	-0.972338	0.720925	28561.99	-0.679978	-0.666282	-1.005852	0.9
1	-0.506592	-1.259957	21704.77	0.570331	-0.005288	-1.005852	0.9
2	0.890646	-0.148730	9609.57	0.252347	-0.996779	-1.005852	0.9
3	-0.506592	-0.245359	18925.23	1.650506	0.986203	1.017588	-0.6
4	-0.972338	1.445638	40074.19	-0.623156	1.316700	1.017588	0.9

C4. Provide a copy of the cleaned data set.

In [18]:

```
# save the prepared data set
df.to_csv('churn_prepared1.csv', index=False)
```

Part IV - Analysis

D1. Split the data into training and test data sets and provide the file(s).

In [19]:

```
df.head()
```

Out[19]:

	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_f
0	-0.972338	0.720925	28561.99	-0.679978	-0.666282	-1.005852	0.9
1	-0.506592	-1.259957	21704.77	0.570331	-0.005288	-1.005852	0.9
2	0.890646	-0.148730	9609.57	0.252347	-0.996779	-1.005852	0.9
3	-0.506592	-0.245359	18925.23	1.650506	0.986203	1.017588	-0.6
4	-0.972338	1.445638	40074.19	-0.623156	1.316700	1.017588	0.9

In [20]:

```
# split the dataframe between independent and dependent variables
X = df.drop('Churn',axis= 1)
y = df[['Churn']]

X.head()
y.head()
```

Out[20]:

	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_f
0	-0.972338	0.720925	28561.99	-0.679978	-0.666282	-1.005852	0.9
1	-0.506592	-1.259957	21704.77	0.570331	-0.005288	-1.005852	0.9
2	0.890646	-0.148730	9609.57	0.252347	-0.996779	-1.005852	0.9
3	-0.506592	-0.245359	18925.23	1.650506	0.986203	1.017588	-0.6
4	-0.972338	1.445638	40074.19	-0.623156	1.316700	1.017588	0.9

Out[20]:

	Churn
0	0
1	1
2	0
3	0
4	1

In [21]:

```
# split train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_si
```

In [22]:

```
X_train.to_csv('X_train1.csv')
X_test.to_csv('X_test1.csv')
y_train.to_csv('y_train1.csv')
y_test.to_csv('y_test1.csv')
```

D2. Describe the analysis technique you used to appropriately analyze the data. Include screenshots of the intermediate calculations you performed.

Initially, I used a very simple KNN classifier with default parameters. The accuracy was abysmal (68%) and the AUC score even more so. In fact, we're better off guessing (50% AUC) than use this model (48%)! With the help of GridSearchCV() I was able to determine the optimal number of k-neighbors. My optimized model increased to an accuracy of 84% and a whopping AUC score of 91%.

- Best number of k-neighbors is 19
- Weights: Uniform
- Algorithm: Auto
- n_jobs: to use all cpu
- K-folds: 5 fold

The best number of k-neighbors is 19 as determined when using GridSearchCV(). The number of k-folds we used is 5, and we're using all cpu's to run the jobs. We're also using the default parameters of "uniform" for weights and "auto" for the algorithm.

In the end, the parameter that made the most difference is the number of k-neighbors.

D3. Provide the code used to perform the classification analysis from part D2.

In [23]:

```
# create model
knn0 = KNeighborsClassifier()

# fit the model
knn0.fit(X_train, y_train['Churn'])

# make predictions
y_pred = knn0.predict(X_test)
```

Out[23]:

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

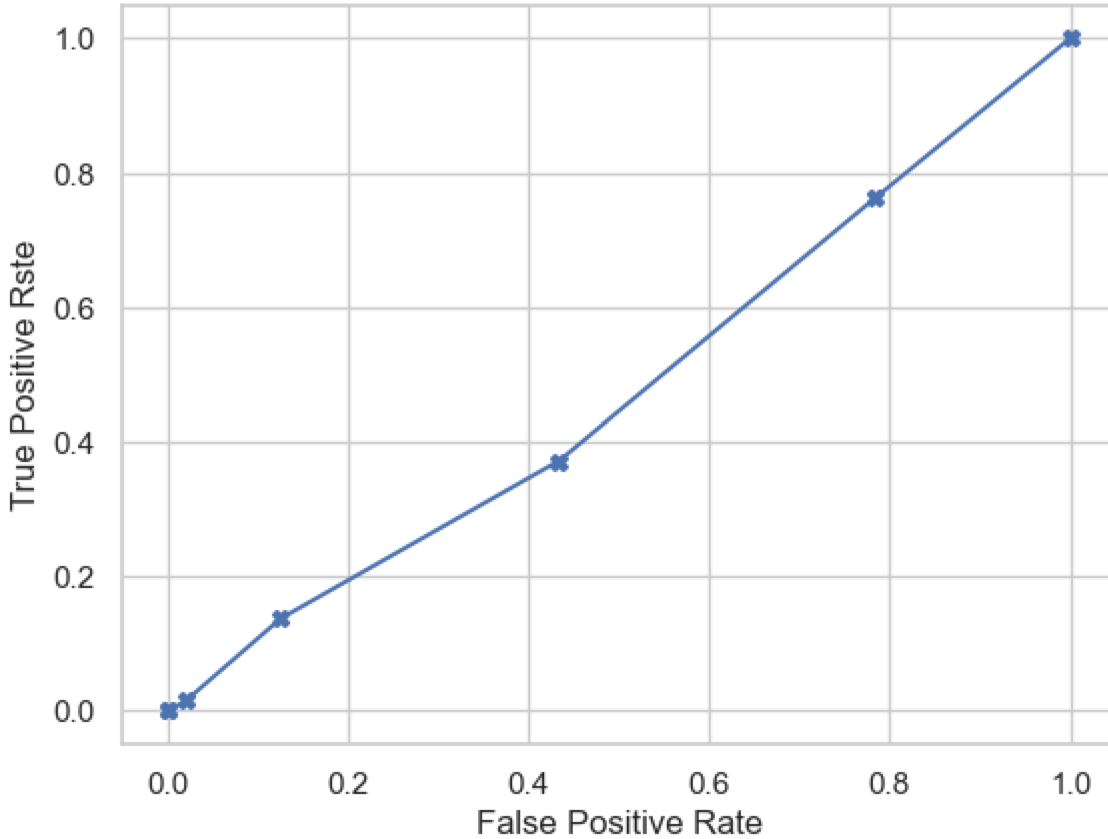
In [24]:

```
# print the accuracy score
print("=====================================================")
print("Accuracy score: " + str(knn0.score(X_test, y_test)))
print("=====================================================")
print("Confusion matrix: \n" + str(confusion_matrix(y_test, y_pred)))
print("-----------------------------------------------------")
y_predicted_proba = knn0.predict_proba(X_test)[:,1]
print("AUC score: " + str(roc_auc_score(y_test, y_predicted_proba)))
```

```
=====================================================
Accuracy score: 0.679
=====================================================
Confusion matrix: 
[[1285  185]
 [ 457   73]]
-----------------------------------------------------
AUC score: 0.47577589526376596
```

In [25]:

```
def viz_roc(model, X_test, y_test):
    probs = model.predict_proba(X_test)
    fpr, tpr, _ = roc_curve(y_test, probs[:,1])
    plt.plot(fpr, tpr, marker="X")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rste")
    plt.show()

viz_roc(knn0, X_test, y_test)
```



In [26]:
```
steps = [('scaler', StandardScaler()),
         ('knn', KNeighborsClassifier())]

pipeline = Pipeline(steps)
```

In [27]:
```
# identify optimal value for hyperparameter tuning
parameters = {'knn__n_neighbors':np.arange(1,30)}
```

In [28]:
```
knn1 = GridSearchCV(estimator=pipeline,
                    param_grid = parameters,    # parameters to try
                    n_jobs=-1,                  # use all cpu
                    cv=5)                       # use 5 fold cross validation
```

In [29]:
```
knn1.fit(X_train, y_train)
```

```
C:\Users\Dd\OneDrive\Documents\_github\d209-data-mining\v399\lib\site-packages\sklea
rn\neighbors\_classification.py:215: DataConversionWarning: A column-vector y was pa
ssed when a 1d array was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  return self._fit(X, y)
```

Out[29]:

```
▸ GridSearchCV
▸ estimator: Pipeline
    ▸ StandardScaler
  ▸ KNeighborsClassifier
```

In [30]:

```
print("Best params: " + str(knn1.best_params_))
```

```
Best params: {'knn__n_neighbors': 19}
```

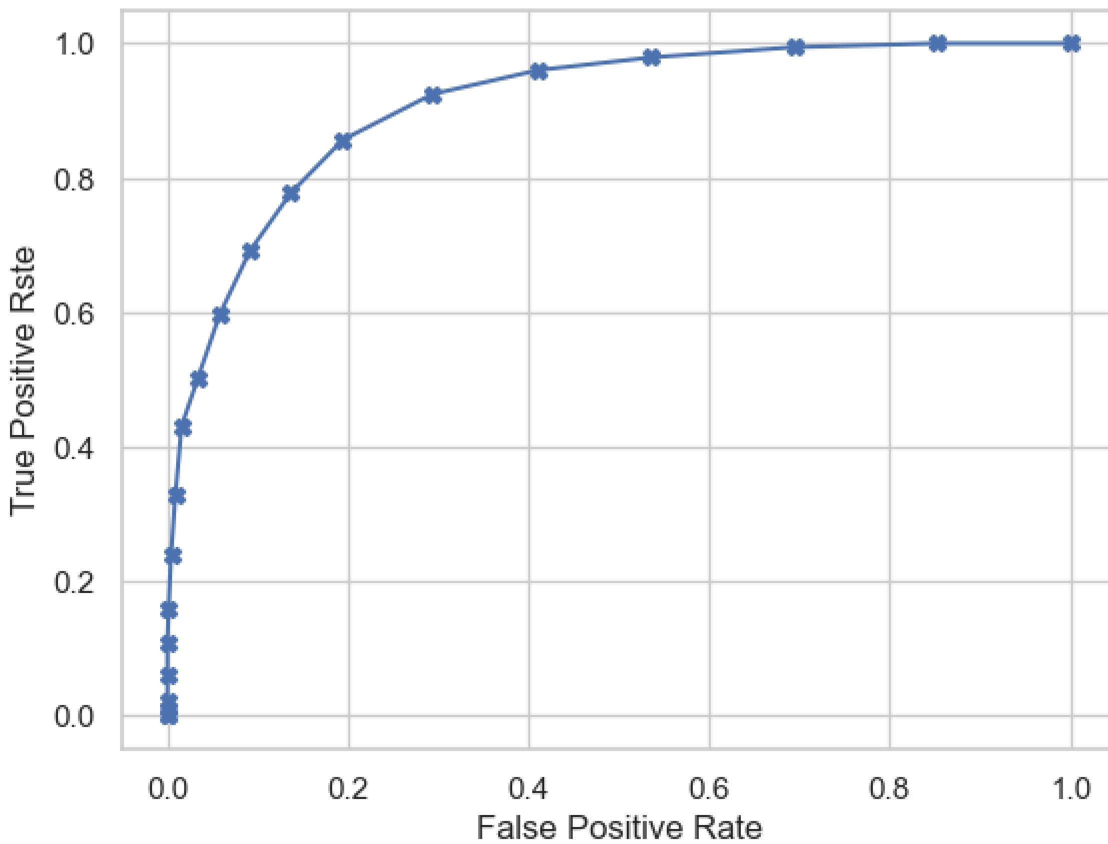
In [31]:

```
# print the accuracy score
print("====================================================")
print("Accuracy score: " + str(knn1.score(X_test, y_test)))
print("====================================================")
print("Confusion matrix: \n" + str(confusion_matrix(y_test, y_pred)))
print("----------------------------------------------------")
y_predicted_proba = knn1.predict_proba(X_test)[:,1]
print("AUC score: " + str(roc_auc_score(y_test, y_predicted_proba)))
```

```
====================================================
Accuracy score: 0.843
====================================================
Confusion matrix:
[[1285  185]
 [ 457   73]]
----------------------------------------------------
AUC score: 0.909960210499294
```

In [32]:

```
viz_roc(knn1, X_test, y_test)
```



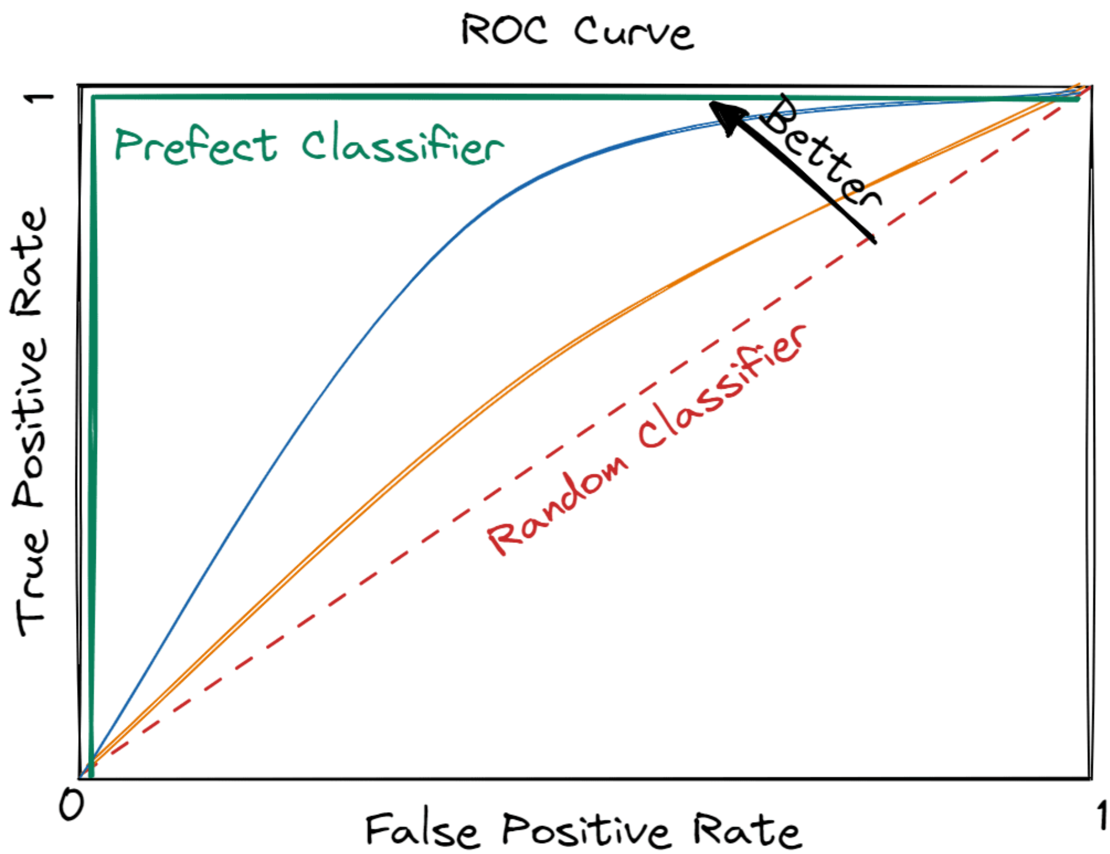
E1. Explain the accuracy and the area-under-the-curve (AUC) of your classification model.

As mentioned, the accuracy and AUC scores of the initial model are 68% and 48%, respectively, while the optimized model has an accuracy of 84% and an AUC score of 91%. This means that the initial was able to predict correctly 68% of the time, while the optimized model was able to predict correctly 84% of the time. Conversely, we're better off looking at AUC because "it calibrates the trade-off between sensitivity and specificity at the best-chosen threshold" (Chugh, 2022).

E2. Discuss the results and implications of your classification analysis.

An untrained, no-skill predictor has an AUC of 50%. As the ROC plot shows, the curve nearly hugs the upper left corner of the graph which depicts the AUC score of 91%, which basically means 9 points less than perfect! This has great implications because this means that the organization can predict which customers will churn and be correct approximately 91% of the time. Practically, our model can power an advanced warning system for churn candidates. The marketing department can use this warning system to inform who to target for mitigation purposes, increasing the retention rate for the organization.

In [33]:
```
Image(url='https://www.kdnuggets.com/wp-content/uploads/chugh_metric_accuracy_auc_2
```
Out[33]:



E3. Discuss one limitation of your data analysis.

One limitation of our analysis lies in the use of the KNN classifier itself. KNN does not work well with large datasets because it would exponentially increase the time it would take to calculate between distances (Soni, 2020).

E4. Recommend a course of action for the real-world organizational situation from part A1 based on your results and implications discussed in part E2.

Based on the optimized KNN model's favorable result, I recommend that the organization begin the effort to productionize this model and deploy it at scale.

F. Provide a Panopto video recording that includes a demonstration of the functionality of the code used for the analysis and a summary of the programming environment.

URL: https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=5d2c6b6e-dfb9-4ca5-af2b-b05a00378fca

G. Record the web sources used to acquire data or segments of third-party code to support the analysis. Ensure the web sources are reliable.

- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.ne
- https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html

H. Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

- https://medium.com/data-science-on-customer-churn-data/k-nearest-neighbors-knn-on-customer-churn-data-40e9b2bb9266
- https://neptune.ai/blog/knn-algorithm-explanation-opportunities-limitations
- https://www.kdnuggets.com/2022/10/metric-accuracy-auc.html
- https://medium.com/@anuuz.soni/advantages-and-disadvantages-of-knn-ee06599b9336

In [34]:
```
print("Successful run!")
```

```
Successful run!
```