

Lora Milam

Western Governors University

D209 Data Mining I

12 July 2022

D209 Performance Assessment Task 2

1 Research Question

Lora Milam Masters Data Analytics (7/4/2022) Program Mentor:d209@wgu.edu

1.1 Question

The questions posed for this assessment will be: what customers are at a high risk of churn? Which customer characteristics are most significant to churn? These questions will be answered by utilizing decision trees.

1.2 Objective

Stakeholders in the company could benefit from further analysis of these fields and the correlation between these fields and the likelihood of a customer being at a high risk of churn. Future findings could provide valuable information for marketing teams to improve the customer's experience and decrease churn rate by utilizing these characteristics and past user experience.

2 Method Justification

2.1 Explanation of Prediction Method

The Decision Tree method creates a tree-like structure of binary splits based on variable values and arrives at a leaf node which is when the classification is assigned to the input. The Decision tree can be adjusted by the hyperparameter options such as maximum branch length, number of trees to be analyzed, minimum number of results per leaf, etc.

The expected outcome is that the test set data points will be categorized(Yes/No) based on data points that were available to the model

2.2 Summary of Assumption

One assumption that this method makes is that attribute value records are distributed recursively(Sharma).

2.3 Packages and Libraries

The packages and/or libraries chosen for this assessment are as follows:

- Pandas
- Numpy
- Matplotlib
- Seaborn
- Scikit-learn

Pandas, Numpy, and Matplotlib provide mathematical and statistical packages to read, score and visualize data. Seaborn provides more in-depth and easy to read graphs, matrices, and plots.

Scikit-learn efficiently provides methods to split, fit, predict and apply metrics for machine learning models.

3 Data Preparation

3.1 Data Preparation Goals

One preprocessing goal for this assessment is to impute numerical dummy variables for features that are populated with character string objects. Specifically, there are a lot of variables that are populated with binary character string objects (yes/no) which can be imputed with numerical objects (1/0).

3.2 Initial Dataset Variables

Variable	Description
CaseOrder:	- Categorical
Customer_id:	- Categorical
Interaction:	- Categorical
UID:	- Categorical
City:	- Categorical
State:	- Categorical
County:	- Categorical
Zip:	- Categorical
Lat:	- Continuous

Lng:	- Continuous
Population:	- Categorical
Area:	- Categorical
Timezone:	- Categorical
Job:	- Categorical
Children:	- Continuous
Age:	- Categorical
Income:	- Continuous
Marital:	- Categorical
Gender:	- Categorical
Churn	- Categorical
Outage_sec_perweek:	- Continuous
Email:	- Continuous
Contacts:	- Continuous
Yearly_equip_failure:	- Continuous
Techie:	- Categorical
Contract:	- Categorical
Port_modem:	- Categorical
Tablet:	- Categorical

InternetService:	- Categorical
Phone:	- Categorical
Multiple:	- Categorical
OnlineSecurity:	- Categorical
OnlineBackup:	- Categorical
DeviceProtection:	- Categorical
TechSupport:	- Categorical
StreamingTV:	- Categorical
StreamingMovies:	- Categorical
PaperlessBilling:	- Categorical
PaymentMethod:	- Categorical
Tenure:	- Continuous
MonthlyCharge:	- Continuous
Bandwidth_GB_Year:	- Continuous
Item1:	- Categorical
Item2:	- Categorical
Item3:	- Categorical
Item4:	- Categorical
Item5:	- Categorical
Item6:	- Categorical

Item7:	- Categorical
Item8:	- Categorical

3.3 Data Preparation Steps

The preparation steps are as follows:

- Read data into Python using the read_csv command provided by Pandas
- Get a better understanding of the input data by using info and describe methods
- Determine if there is a possibility for misspellings, unintuitive variable names and /or missing values
- Determine if there are outliers that may disrupt the model by using histograms and box plots
- Impute missing data with measures of central tendency(mean, median, or mode) or remove outliers that have a significant difference then the standard deviation
- Remove less meaningful variables
- Save cleaned dataset as “churn_clean.csv” for future use.

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

# Visualization
import seaborn as sns
import matplotlib as mlp
import matplotlib.pyplot as plt
%matplotlib inline

# Scikit-learn
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
```

```
In [2]: # Load data set into Pandas dataset
churn_df = pd.read_csv('churn.csv', index_col = 0)

In [3]: # Look at dataset variables
churn_df.columns

Out[3]: Index(['Customer_id', 'Interaction', 'UID', 'City', 'State', 'County', 'Zip',
       'Lat', 'Lng', 'Population', 'Tenure', 'MonthlyCharge'],
       dtype='object')

In [4]: # Look at dimensions of dataset
churn_df.shape

Out[4]: (10000, 49)

In [5]: # Look at some of the entries of dataset
churn_df.head()

Out[5]:
   Customer_id Interaction          UID    City  State  County     Zip      Lat      Lng  Population ... MonthlyCha
CaseOrder
1      K409198 aa90260b-4141-4a24-8e36-b04ce1f477fb e885b299883d4f9fb18e39c75155d990 Point Baker    AK Prince of Wales-Hyder 99927 56.25100 -133.37571 38 ... 172.455
2      S120509 fb76459f-c047-4a9d-8af9-e0f7d4ac2524 f2de8bef964785f41a2959829830fb8a West Branch   MI Ogemaw 48661 44.32893 -84.24080 10446 ... 242.632
3      K191035 344d114c-3736-4be5-98f7-c72c281e2d35 f1784cfa9f6d92ae816197eb175d3c71 Yamhill    OR Yamhill 97148 45.35589 -123.24657 3735 ... 159.947
4      D90850 abfa2b40-2d43-4994-b15a-989b8c79e311 dc8a365077241bb5cd5cc305136b05e Del Mar    CA San Diego 92014 32.96687 -117.24798 13863 ... 119.956
5      K662701 68a861fd-0d20-4e51-a587-8a90407ee574 aabb64a116e83fdc4befc1fbab1663f9 Needville TX Fort Bend 77461 29.38012 -95.80673 11352 ... 149.948
```

5 rows × 49 columns

```
In [6]: # Look at dataframe info
churn_df.info
```

```
Out[6]: <bound method DataFrame.info of
CaseOrder
   ...
```

		Customer_id	Interaction					
1	K409198	aa90260b-4141-4a24-8e36-b04ce1f4f77b						
2	S120509	fb76459f-c047-4a9d-8af9-e0f7d4ac2524						
3	K191035	344d114c-3736-4be5-98e7-c72c281e2d35						
4	D90850	abfa2b40-2d43-4994-b15a-989b8c79e311						
5	K662701	68a861fd-0d20-4e51-a587-8a90407ee574						
...						
9996	M324793	45deb5a2-ae04-4518-bf0b-c82db8dbe44						
9997	D861732	6e96b921-0c09-4993-bbda-a1ac6411061a						
9998	I243405	e8307ddf-9a01-4fff-bc59-4742e03fd24f						
9999	I641617	3775ccfc-0052-4107-81ae-965781ecdf3						
10000	T38070	9de5fb6e-bd33-4995-aec8-f01d0172a499						
		UID	City State					
CaseOrder								
1	e885b299883d4f9fb18e39c75155d990	Point Baker	AK					
2	f2de8bef964785e41a2959829830fb8a	West Branch	MI					
3	f1784cfa9f6d92ae816197eb175d3c71	Yamhill	OR					
4	dc8a365077241bb5cd5cccd305136b05e	Del Mar	CA					
5	aabb64a116e83fdc4befc1fbab1663f9	Needville	TX					
...					
9996	9499fb4de537af195d16d046b79fd20a	Mount Holly	VT					
9997	c09a841117fa81b5c8e19afec2760104	Clarksville	TN					
9998	9e41f212d1e04dca84445019bcc9b41c	Mobeetie	TX					
9999	3elf269b40c235a1038863ecf6b7a0df	Carrollton	GA					
10000	0ea683a03a3cd544aefe8388aab16176	Clarksville	GA					
		County	Zip	Lat	Lng	Population	...	
CaseOrder								
1	Prince of Wales-Hyder	99927	56.25100	-133.37571		38	...	
2	Ogemaw	48661	44.32893	-84.24080		10446	...	
3	Yamhill	97148	45.35589	-123.24657		3735	...	
4	San Diego	92014	32.96687	-117.24798		13863	...	
5	Fort Bend	77461	29.38012	-95.80673		11352	...	
...	
9996	Rutland	5758	43.43391	-72.78734		640	...	
9997	Montgomery	37042	36.56907	-87.41694		77168	...	
9998	Wheeler	79061	35.52039	-100.44180		406	...	
9999	Carroll	30117	33.58016	-85.13241		35575	...	
10000	Habersham	30523	34.70783	-83.53648		12230	...	
		MonthlyCharge	Bandwidth_GB_Year	Item1	Item2	Item3	Item4	Item5
CaseOrder								
1	172.455519	904.536110	5	5	5	3	4	
2	242.632554	800.982766	3	4	3	3	4	
3	159.947583	2054.706961	4	4	2	4	4	
4	119.956840	2164.579412	4	4	4	2	5	
5	149.948316	271.493436	4	4	4	3	4	
...
9996	159.979400	6511.252601	3	2	3	3	4	
9997	207.481100	5695.951810	4	5	5	4	4	
9998	169.974100	4159.305799	4	4	4	4	4	
9999	252.624000	6468.456752	4	4	6	4	3	
10000	217.484000	5857.586167	2	2	3	3	3	
		Item6	Item7	Item8				
CaseOrder								
1	4	3	4					
2	3	4	4					
3	3	3	3					
4	4	3	3					
5	4	4	5					
...					
9996	3	2	3					
9997	5	2	5					
9998	4	4	5					
9999	3	5	4					
10000	3	4	1					

[10000 rows x 49 columns]>

```
In [7]: # Look at stats of dataframe  
churn_df.describe()
```

Out[7]:

	Zip	Lat	Lng	Population	Children	Age	Income	Outage_sec_perweek	Email	Contacts	..
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.0000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	..
mean	49153.319600	38.757567	-90.782536	9756.562400	2.0877	53.078400	39806.926771	10.001848	12.016000	0.994200	..
std	27532.196108	5.437389	15.156142	14432.698671	2.1472	20.698882	28199.916702	2.976019	3.025898	0.988466	..
min	601.000000	17.966120	-171.688150	0.000000	0.0000	18.000000	348.670000	0.099747	1.000000	0.000000	..
25%	26292.500000	35.341828	-97.082812	738.000000	0.0000	35.000000	19224.717500	8.018214	10.000000	0.000000	..
50%	48869.500000	39.395800	-87.918800	2910.500000	1.0000	53.000000	33170.605000	10.018560	12.000000	1.000000	..
75%	71866.500000	42.106908	-80.088745	13168.000000	3.0000	71.000000	53246.170000	11.969485	14.000000	2.000000	..
max	99929.000000	70.640660	-65.667850	111850.000000	10.0000	89.000000	258900.700000	21.207230	23.000000	7.000000	..

8 rows × 22 columns

```
In [8]: # Look at data types of variables  
churn_df.dtypes
```

```
Out[8]: Customer_id          object  
Interaction           object  
UID                   object  
City                  object  
State                 object  
County                object  
Zip                   int64  
Lat                   float64  
Lng                   float64  
Population            int64  
Area                  object  
TimeZone              object  
Job                   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 [9]: # Rename unintuitive variable names
churn_df = churn_df.rename(columns = {'Item1':'TimelyResponse',
                                      'Item2':'Fixes',
                                      'Item3':'Replacements',
                                      'Item4':'Reliability',
                                      'Item5':'Options',
                                      'Item6':'Respectfulness',
                                      'Item7':'Courteous',
                                      'Item8':'Listening'})
```

```
In [10]: # Look at the dataset variables again
churn_df.columns
```

```
Out[10]: Index(['Customer_id', 'Interaction', 'UID', 'City', 'State', 'County', 'Zip',
       'Lat', 'Lng', 'Population', 'Area', 'TimeZone', 'Job', 'Children',
       'Age', 'Income', 'Marital', 'Gender', 'Churn', 'Outage_sec_perweek',
       'Email', 'Contacts', 'Yearly_equip_failure', 'Techie', 'Contract',
       'Port_modem', 'Tablet', 'InternetService', 'Phone', 'Multiple',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'PaperlessBilling', 'PaymentMethod',
       'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'TimelyResponse',
       'Fixes', 'Replacements', 'Reliability', 'Options', 'Respectfulness',
       'Courteous', 'Listening'],
      dtype='object')
```

```
In [11]: # Remove less relevant fields from stats description
churn_df = churn_df.drop(columns=['Customer_id', 'Interaction', 'UID', 'City', 'State',
                                   'County', 'Zip', 'Lat', 'Lng', 'Population', 'Area', 'TimeZone', 'Job',
                                   'Marital'])
```

```
churn_df.shape
```

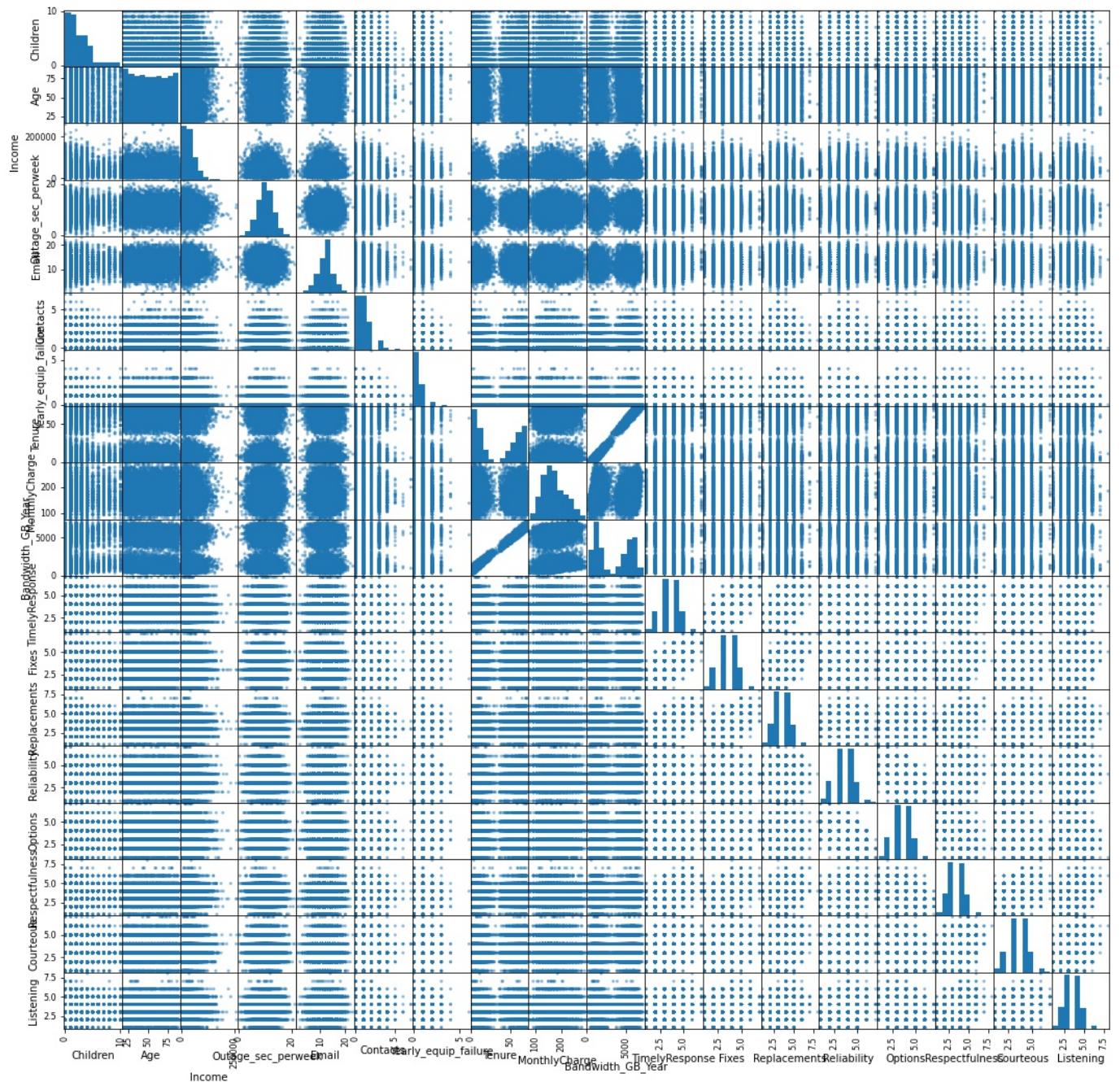
```
Out[11]: (10000, 35)
```

```
In [12]: churn_df.columns
```

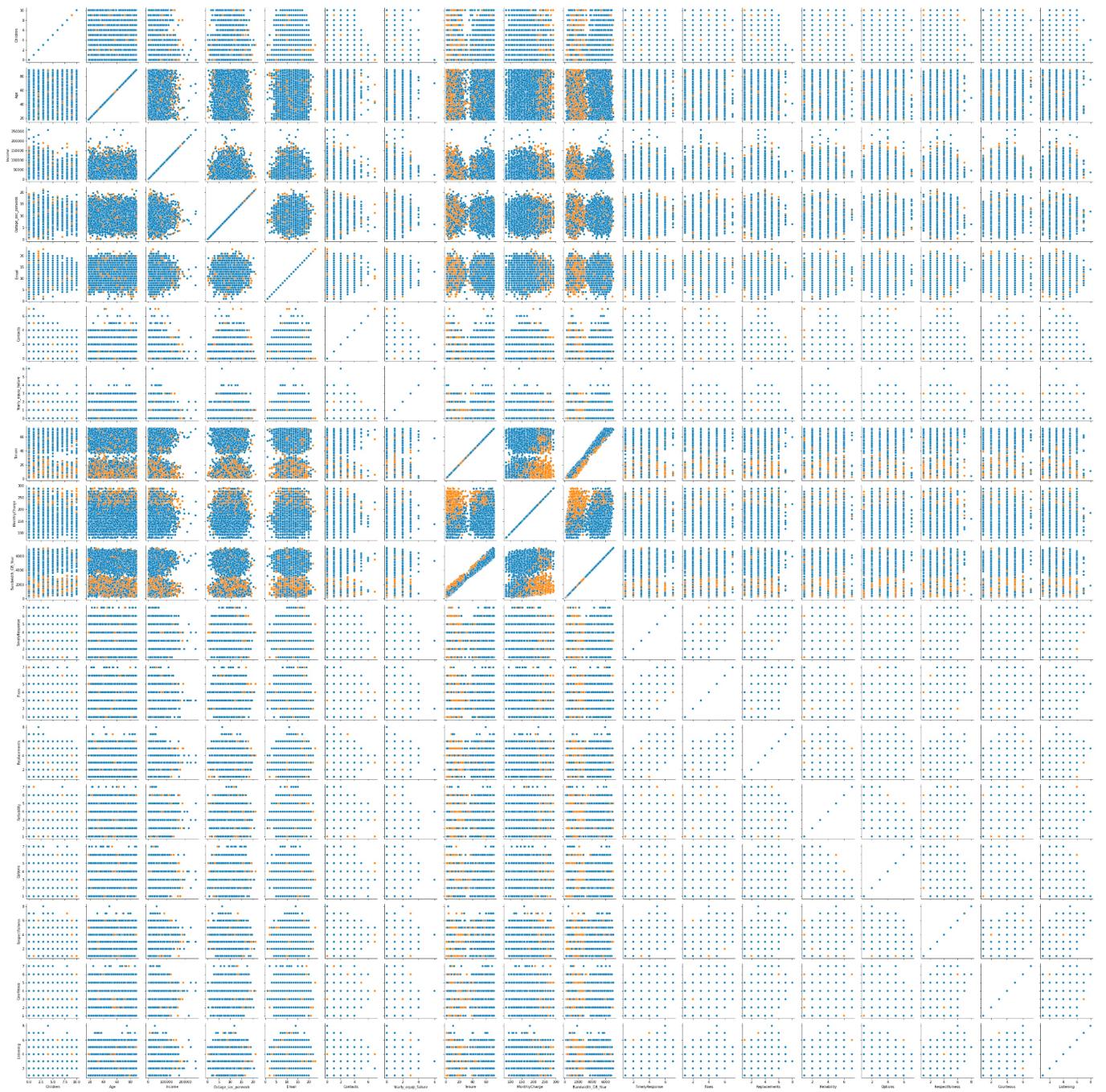
```
Out[12]: Index(['Children', 'Age', 'Income', 'Gender', 'Churn', 'Outage_sec_perweek',
       'Email', 'Contacts', 'Yearly_equip_failure', 'Techie', 'Contract',
       'Port_modem', 'Tablet', 'InternetService', 'Phone', 'Multiple',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'PaperlessBilling', 'PaymentMethod',
       'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'TimelyResponse',
       'Fixes', 'Replacements', 'Reliability', 'Options', 'Respectfulness',
       'Courteous', 'Listening'],
      dtype='object')
```

```
In [13]: # Examine continuous variables by creating histograms
churn_df[['Children', 'Age', 'Income', 'Outage_sec_perweek',
          'Email', 'Contacts', 'Yearly_equip_failure',
          'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'TimelyResponse',
          'Fixes', 'Replacements', 'Reliability', 'Options', 'Respectfulness',
          'Courteous', 'Listening']].hist()
plt.savefig('churn_pypplot.jpg')
plt.tight_layout()
```

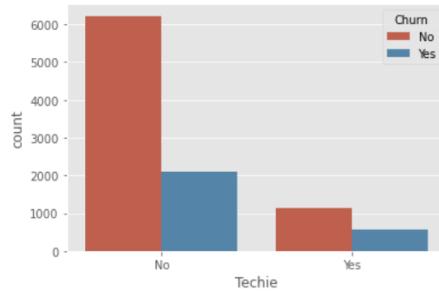
```
In [14]: # Examine a scatter matrix for any potential relationships
churn_numeric = churn_df[['Children', 'Age', 'Income', 'Outage_sec_perweek',
                           'Email', 'Contacts', 'Yearly_equip_failure',
                           'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'TimelyResponse',
                           'Fixes', 'Replacements', 'Reliability', 'Options', 'Respectfulness',
                           'Courteous', 'Listening']]
pd.plotting.scatter_matrix(churn_numeric, figsize = [18,18])
plt.savefig('numeric_scatter_matrix.jpg')
```



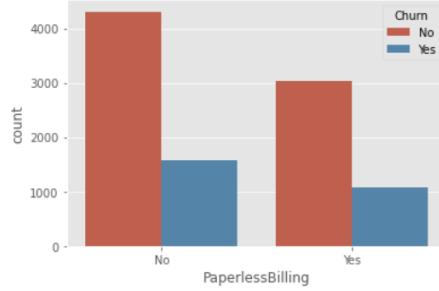
```
In [15]: # Examine a scatter matrix with the target variable as the hue
g = sns.PairGrid(churn_df, hue = "Churn")
g.map(sns.scatterplot)
plt.savefig('churn_scatter_matrix.jpg')
```



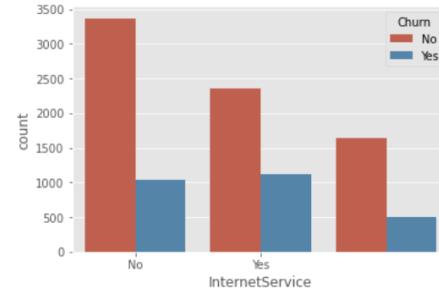
```
In [16]: # Examine binary categoricals using ggplot's countplot
plt.style.use('ggplot')
plt.figure()
sns.countplot(x='Techie', hue='Churn', data=churn_df)
plt.xticks([0,1],['No','Yes'])
plt.show()
```



```
In [17]: plt.figure()
sns.countplot(x='PaperlessBilling', hue='Churn', data=churn_df)
plt.xticks([0,1],['No','Yes'])
plt.show()
```

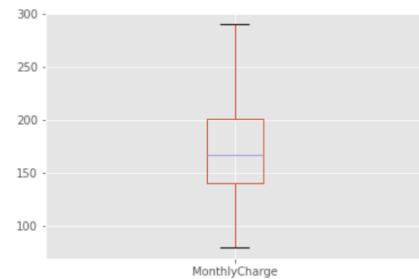


```
In [18]: plt.figure()
sns.countplot(x='InternetService', hue='Churn', data=churn_df)
plt.xticks([0,1],['No','Yes'])
plt.show()
```

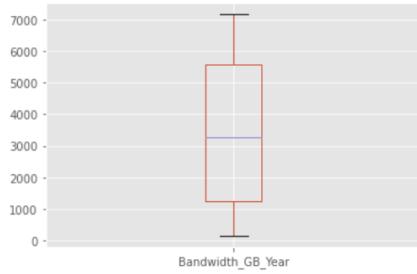


```
In [19]: # Examine boxplots or variables
churn_df.boxplot(column='MonthlyCharge')
```

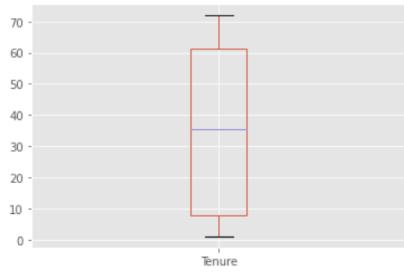
```
Out[19]: <AxesSubplot:
```



```
In [20]: churn_df.boxplot(column='Bandwidth_GB_Year')
Out[20]: <AxesSubplot:>
```



```
In [21]: churn_df.boxplot(column='Tenure')
Out[21]: <AxesSubplot:>
```



Anomalies

- Anomalies have been removed from the dataset. There are no remaining outliers

```
In [22]: # Determine if there is any missing data
data_nulls = churn_df.isnull().sum()
print(data_nulls)
```

```
Children      0
Age          0
Income        0
Gender        0
Churn         0
Outage_sec_perweek  0
Email         0
Contacts       0
Yearly_equip_failure 0
Techie        0
Contract       0
Port_modem     0
Tablet         0
InternetService 0
Phone          0
Multiple        0
OnlineSecurity   0
OnlineBackup     0
DeviceProtection 0
TechSupport      0
StreamingTV      0
StreamingMovies    0
PaperlessBilling 0
PaymentMethod     0
Tenure          0
MonthlyCharge     0
Bandwidth_GB_Year 0
TimelyResponse    0
Fixes           0
Replacements     0
Reliability      0
Options          0
Respectfulness    0
Courteous         0
Listening         0
dtype: int64
```

```
In [23]: # Impute binary categoricals as numerics
data = churn_df['Churn']
churn_dict = {
    "No":0,"Yes":1
}
# Churn variable
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['Churn'] = data
# Techie variable
data = churn_df['Techie']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['Techie'] = data
# Port_modem variable
data = churn_df['Port_modem']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['Port_modem'] = data
# Tablet variable
data = churn_df['Tablet']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['Tablet'] = data
# Phone variable
data = churn_df['Phone']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['Phone'] = data
# Multiple variable
data = churn_df['Multiple']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['Multiple'] = data
# OnlineSecurity variable
data = churn_df['OnlineSecurity']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['OnlineSecurity'] = data
# OnlineBackup variable
data = churn_df['OnlineBackup']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['OnlineBackup'] = data
# DeviceProtection variable
data = churn_df['DeviceProtection']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['DeviceProtection'] = data
# TechSupport variable
data = churn_df['TechSupport']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['TechSupport'] = data
# StreamingTV variable
data = churn_df['StreamingTV']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['StreamingTV'] = data
# StreamingMovies variable
data = churn_df['StreamingMovies']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['StreamingMovies'] = data
# PaperlessBilling variable
data = churn_df['PaperlessBilling']
for k,v in churn_dict.items():
    data = data.replace(k,v)
churn_df['PaperlessBilling'] = data
```

```
In [24]: churn_df.head()
```

```
Out[24]:
```

	Children	Age	Income	Gender	Churn	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Techie	...	MonthlyCharge	Bandwidth_GB_Year
CaseOrder													
1	0	68	28561.99	Male	0	7.978323	10	0	1	0	...	172.455519	904.536110
2	1	27	21704.77	Female	1	11.699080	12	0	1	1	...	242.632554	800.982766
3	4	50	9609.57	Female	0	10.752800	9	0	1	1	...	159.947583	2054.706961
4	1	48	18925.23	Male	0	14.913540	15	2	0	1	...	119.956840	2164.579412
5	0	83	40074.19	Male	1	8.147417	16	2	1	0	...	149.948316	271.493436

5 rows × 35 columns

```
In [25]: # Look at data types of variables
churn_df.dtypes
```

```
Out[25]: Children           int64
Age              int64
Income           float64
Gender            object
Churn             int64
Outage_sec_perweek  float64
Email             int64
Contacts          int64
Yearly_equip_failure  int64
Techie            int64
Contract           object
Port_modem         int64
Tablet             int64
InternetService    object
Phone              int64
Multiple            int64
OnlineSecurity      int64
OnlineBackup         int64
DeviceProtection     int64
TechSupport          int64
StreamingTV          int64
StreamingMovies       int64
PaperlessBilling      int64
PaymentMethod        object
Tenure              float64
MonthlyCharge        float64
Bandwidth_GB_Year      float64
TimelyResponse        int64
Fixes                int64
Replacements          int64
Reliability           int64
Options              int64
Respectfulness          int64
Courteous             int64
Listening             int64
dtype: object
```

```
In [26]: # Create dummy variables for the remaining non-numeric variables:
# Gender, Contract, InternetService, PaymentMethod
# Drop original column
# Exclude redundant values: ex. gender is categorized as male or female, the value 'prefer not to answer' can be identified
# Gender
dmy = pd.get_dummies(churn_df['Gender'])
dmy = dmy.iloc[:, :-1]
churn_df = pd.concat([churn_df, dmy], axis=1)
churn_df = churn_df.drop(columns = 'Gender')

# Contract
dmy = pd.get_dummies(churn_df['Contract'])
churn_df = pd.concat([churn_df, dmy], axis=1)
churn_df = churn_df.drop(columns = 'Contract')

# InternetService
dmy = pd.get_dummies(churn_df['InternetService'])
churn_df = pd.concat([churn_df, dmy], axis=1)
churn_df = churn_df.drop(columns = 'InternetService')

# PaymentMethod
dmy = pd.get_dummies(churn_df['PaymentMethod'])
churn_df = pd.concat([churn_df, dmy], axis=1)
churn_df = churn_df.drop(columns = 'PaymentMethod')
```

```
In [27]: # Move Churn to the end of the dataframe to set as target
Churn = churn_df[['Churn']]
churn_df.pop("Churn")
churn_df['Churn'] = Churn

df = churn_df.columns
print(df)
churn_df.head()

Index(['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts',
       'Yearly_equip_failure', 'Techie', 'Port_modem', 'Tablet', 'Phone',
       'Multiple', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'PaperlessBilling',
       'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'TimelyResponse',
       'Fixes', 'Replacements', 'Reliability', 'Options', 'Respectfulness',
       'Courteous', 'Listening', 'Female', 'Male', 'Month-to-month',
       'One year', 'Two Year', 'DSL', 'Fiber Optic', 'None',
       'Bank Transfer(automatic)', 'Credit Card (automatic)',
       'Electronic Check', 'Mailed Check', 'Churn'],
      dtype='object')

Out[27]:
   Children  Age  Income  Outage_sec_perweek  Email  Contacts  Yearly_equip_failure  Techie  Port_modem  Tablet ... One year  Two Year  DSL  Fiber Optic  None
0         0    68     28561.99        7.978323     10      0           1      0           1      1 ... 1      0      0      1      0
1         1    27     21704.77       11.699080     12      0           1      1           0      1 ... 0      0      0      0      1      0
2         4    50      9609.57       10.752800      9      0           1      1           1      0 ... 0      1      1      0      0      0
3         1    48     18925.23       14.913540     15      2           0      1           0      0 ... 0      1      1      0      0      0
4         0    83     40074.19       8.147417     16      2           1      0           1      0 ... 0      0      0      1      0      0
```

CaseOrder

1	0	68	28561.99	7.978323	10	0	1	0	1	1	...	1	0	0	1	0	
2	1	27	21704.77	11.699080	12	0	1	1	0	1	...	0	0	0	0	1	0
3	4	50	9609.57	10.752800	9	0	1	1	1	0	...	0	1	1	0	0	0
4	1	48	18925.23	14.913540	15	2	0	1	0	0	...	0	1	1	0	0	0
5	0	83	40074.19	8.147417	16	2	1	0	1	0	...	0	0	0	1	0	0

5 rows x 43 columns

```
In [28]: # List variables for analysis
var = (list(churn_df.columns[:1]))
print('Variables for analysis include: \n', var)

Variables for analysis include:
['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Techie', 'Port_modem', 'Tablet', 'Phone', 'Multiple', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'PaperlessBilling', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'TimelyResponse', 'Fixes', 'Replacements', 'Reliability', 'Options', 'Respectfulness', 'Courteous', 'Listening', 'Female', 'Male', 'Month-to-month', 'One year', 'Two Year', 'DSL', 'Fiber Optic', 'None', 'Bank Transfer(automatic)', 'Credit Card (automatic)', 'Electronic Check', 'Mailed Check']
```

```
In [29]: # Save clean dataset
churn_df.to_csv('clean_churn.csv')
```

3.4 Prepared Dataset

This can be found in the attached file named “clean_churn.csv”.

4 Analysis

```
In [30]: # Re-import fully numeric prepared dataset
churn_df = pd.read_csv('clean_churn.csv')

# Set predictor variables and target variable
X = churn_df.drop('Churn',axis=1).values
y = churn_df['Churn'].values
```

```
In [31]: # Import model, splitting method and metrics from sklearn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score, train_test_split
```

4.1 Train/Test Split

The training and test sets can be found in the attached files named “training_set.csv” and “testing_set.csv”.

```
In [32]: # Set the seed
seed = 1313

# Create training and test sets
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = .2, random_state = seed)

In [33]: # Convert training and testing sets to Dataframe for export
X_train_df = pd.DataFrame(X_train, columns = ['x','Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts',
y_train_df = pd.DataFrame(y_train, columns = ['Churn'])
X_test_df = pd.DataFrame(X_test, columns = ['x','Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts'],
y_test_df = pd.DataFrame(y_test, columns = ['Churn'])
X_train_df
```

Out[33]:

x	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure	Techie	Port_modem	...	Listening	Female	Male	Month-to-month
0	1971.0	1.0	86.0	20926.46	3.577497	13.0	0.0	0.0	0.0	...	3.0	1.0	0.0	0.0
1	4101.0	2.0	67.0	39176.98	8.341450	15.0	1.0	0.0	0.0	1.0	...	5.0	1.0	0.0
2	9942.0	1.0	80.0	5821.59	14.678300	9.0	0.0	0.0	0.0	0.0	...	2.0	0.0	1.0
3	2325.0	0.0	58.0	9352.62	13.197550	13.0	3.0	1.0	0.0	0.0	...	3.0	1.0	0.0
4	8066.0	0.0	63.0	20189.82	8.383716	6.0	1.0	0.0	0.0	1.0	...	4.0	0.0	1.0
...
7995	3820.0	0.0	50.0	20931.13	11.411930	12.0	1.0	1.0	0.0	1.0	...	4.0	0.0	1.0
7996	5959.0	9.0	36.0	38900.92	11.352260	7.0	1.0	0.0	1.0	1.0	...	4.0	0.0	1.0
7997	5803.0	1.0	19.0	27844.19	7.227551	8.0	2.0	0.0	0.0	1.0	...	4.0	0.0	1.0
7998	6324.0	4.0	50.0	70152.92	10.304880	8.0	0.0	2.0	0.0	0.0	...	5.0	1.0	0.0
7999	9073.0	4.0	73.0	71134.21	9.067273	18.0	0.0	0.0	1.0	0.0	...	3.0	0.0	1.0

8000 rows × 40 columns

```
In [34]: # Save training and testing sets
training_set = pd.concat([X_train_df,y_train_df], axis = 1)
testing_set = pd.concat([X_test_df, y_test_df], axis = 1)

training_set.pop('x')
testing_set.pop('x')

# Export sets
training_set.to_csv('training_set')
testing_set.to_csv('testing_set')
```

4.2 Analysis Technique

```
In [35]: # Declare the decision tree and parameter grid
dt = DecisionTreeClassifier()
param_grid = {'max_depth' : [1,2,3,4,5,6,7,8,9,10]}

# Instantiate the grid search and fit to the training data
grid_dt = GridSearchCV(estimator=dt, param_grid=param_grid,
                       scoring='neg_mean_squared_error', cv=10, n_jobs=-1)
grid_dt.fit(X_train,y_train)

# Best hyperparameters and sulting MSE
print('Best hyperparameters:', grid_dt.best_params_)
print('Best MSE:', grid_dt.best_score_* -1)

Best hyperparameters: {'max_depth': 5}
Best MSE: 0.12050000000000001

In [36]: # Determine best model and use it to predict
best_dt = grid_dt.best_estimator_
y_pred = best_dt.predict(X_test)

# Make confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
conf_matrix

Out[36]: array([[1362,  108],
   [ 141,  389]])

In [37]: # Test set Accuracy and MSE
print('Accuracy: ',accuracy_score(y_test,y_pred))
print('MSE: ', mean_squared_error(y_test, y_pred))

Accuracy:  0.8755
MSE:  0.1245
```

5 Data Summary and Implications

5.1 Accuracy and MSE

The accuracy of this Decision Tree model is 87.6%. The mean squared error of this Decision Tree model is .125.

5.2 Results and Implications

This Decision Tree model predicts about 87.6% correctly. This means that 12.4% of the time, the model predicts incorrectly. Breaking it down even more, the model predicts 93% of churned customers correctly and 73% of non-churned customers correctly. This means that 7% of all potential churned customers are categorized as non-churned and 27% of all potential non-churned customers are categorized as churned. Though this model is relatively accurate, the inaccurate predictions could cost time and resources by incorrectly depicting where those resources should be directed to. Depending on the implications of marketing costs versus loss of customers, it could be argued that marketing someone who doesn't plan to churn is less impactful than losing a customer due to not marketing them. This model would be a good model to start out with since it has a relatively low inaccuracy concerning potential churn customers.

5.3 Limitations of Analysis

One limitation of this analysis is that Decision Tree models are best used for categorical datasets. Continuous datasets would be placed in “buckets” and would have less precise assumptions.

5.4 Recommended Course of Action

The recommended course of action would be to further analyze the data to determine if there is a better combination of variables(more or less) and/or adjustments of hyperparameters that would further strengthen the model.

6 Supporting Documents

6.1 Video

This can be found within the attached file ‘Panopto Recording’.

6.2 Sources for Third-Party Code

Sharma, A. (2021, February 25). Machine Learning 101: Decision tree algorithm for classification. Analytics Vidhya. Retrieved July 12, 2022, from
<https://www.analyticsvidhya.com/blog/2021/02/machine-learning-101-decision-tree-algorithm-for-classification/#:~:text=Assumptions%20that%20we%20make%20while%20using%20the%20Decision%20tree%3A&text=%2DBased%20on%20attribute%20value%20records,node%20or%20the%20internal%20node.&text=Entropy%3A%20Entropy%20is%20the%20measures,in%20a%20bunch%20of%20examples.>

6.3 Sources

Straw, Eric. (2022). D209 Data Mining I . Salt Lake City ; Western Governors University.

Western Governors University. (n.d.). D207 D208 D209 Medical Data Considerations and Dictionary. Salt Lake City.