D209 Predictive Analysis Performance Assessment, Task 2

Alexa R. Fisher

Western Governors University

Degree: M.S. Data Analytics

Table of Contents

Part I: Research Question

A1. Research Question

The research question for this thesis was, “Is it possible to predict a customer’s tenure using the Decision Tree method and pinpoint the contributing variables for the telecommunication stakeholders?”. This question was assessed using the predictive analysis method called Decision Tree Regression on the telecommunication dataset. This regression technique was described as a supervised non-parametric learning algorithm used for regression tasks (IBM, n.d.). This method used a tree structure that included root nodes, decision nodes, leaf nodes, and the branches connecting them (Kawerk, n.d.). The target variable for this analysis was “Tenure”. This continuous attribute was defined as the customer’s service longevity in a monthly measure. The feature attributes were determined by the SelectKBest feature selection module in Scikit Learn (Pedregosa, 2011). The independent variables included “Churn”, “DeviceProtection”, and “Bandwidth_GB_Year”.

A2. Data Analysis Goal

The main data analysis goal was to help identify a client’s tenure with the telecommunication company. It gave insight to the organization on which customers would have higher longevity based on contributing factors. A machine learning model was created using the decision tree regressor within Python. The decision tree started with a root node asking a specific question, which resulted in a true or false response. For example, our model asked the question “Was the Bandwidth less than or equal to 3345.469 GB per year?” as the root node. The model deemed the “Bandwidth_GB_Year” variable as the strongest predictor for the decision tree. The responses provided branches to decision nodes from the root node until we were left with a leaf

node. The leaf was the final terminating result (Kawerk, n.d.). The model provided information on how long a client's tenure would be. This was based on the customer's churn, service add-ons, and the average amount of data used per year. The analysis can help the stakeholders measure customer loyalty and contribute to the overall lifetime value a customer can provide. The two metrics were key components in the company's retention strategies (Van, 2020).

Part II: Method Justification

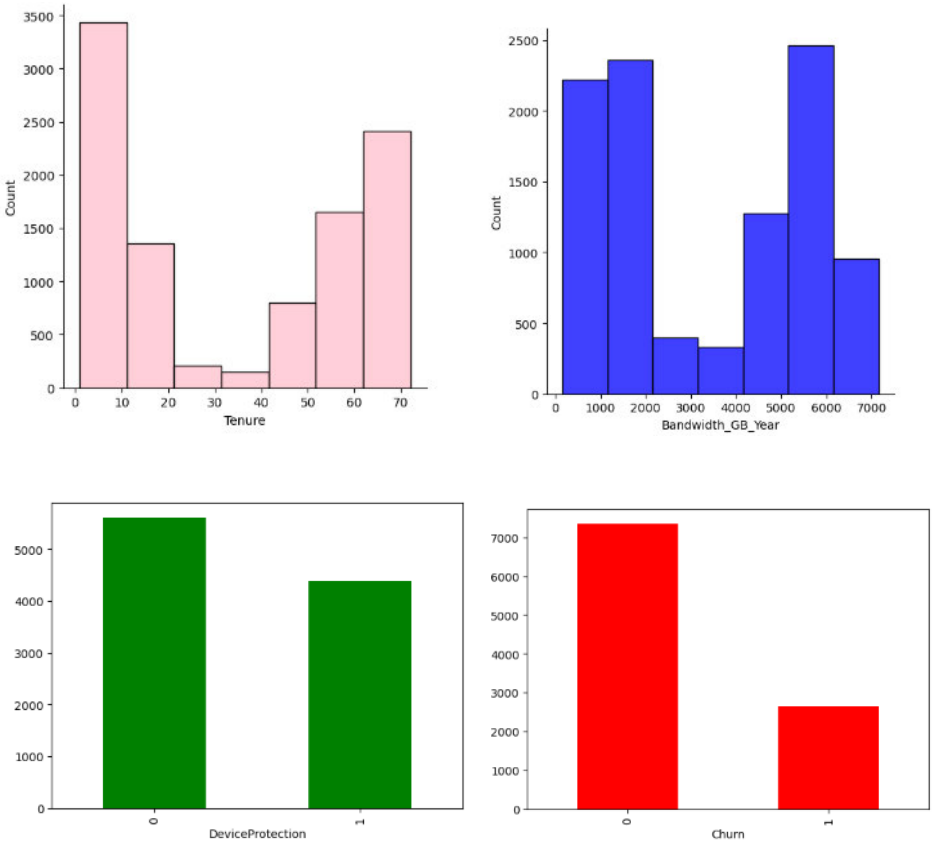
B1. Analysis of Prediction Method

The Decision Tree Regression prediction method was applied to the telecommunication data set. The method provided an output of a real value such as the monthly tenure given the relationship with the feature variables (Kawerk, n.d.). The target variable of "Tenure" and the three independent variables of "Bandwidth_GB_Year", "Churn", and "DeviceProtection" were analyzed in the telecommunication data set. The decision tree method computed the true or false response to the root node. The root node branched into decision nodes based on the response. It continued to split based on the previous result until it was left with a terminal leaf (Coursera, 2022). The decision tree was able to predict a customer's tenure based on the association with the independent variables.

The model provided the stakeholders of the telecommunication company with a high R^- squared statistical measure of 0.97 or 97%. The R-squared value provided a measure of if the model was a good fit. The prediction algorithm utilization gave the organization insight into a customer's monthly duration based on the terminal leaf outcome. This outcome could be used in the company's efforts of retention and provide brainstorming points for sales and marketing (Expert Panel, Forbes Agency Council, 2019).

B2. Summary of Prediction Method Assumption

The Decision Tree prediction method has numerous assumptions. The main assumption was it was non-parametric (IBM, n.d.). This was described as having no distributional assumptions. The method could easily handle skewed, non-linear, bi-modal, or categorical data. It made the model easy to interpret and visualize (IBM, n.d.). In the instance of the telecommunication data set, this assumption could be made based on the various distributions of customers' tenure, churn, bandwidth usage, and service add-on of device protection.



B3. Package and Libraries List

Please see the table below for a listing of the packages and libraries used in the data analysis of the telecommunication data set. The package and libraries' names were noted as well as their usages.

Packages and Libraries	**Usage**
Matplotlib.pyplot	The Pylot module within the Matplotlib library was used to give visualizations. It presented graphical representations of the Decision Trees and HeatMaps.
NumPy	The NumPy library was utilized to work with arrays. This allowed data to be used in calculations and manipulations.
Pandas	The Pandas library was used to analyze data. It allowed the data set to be explored by importing the churn CSV file. It provided a way for the dataframe to be manipulated during the data cleaning and exploration phases.
Seaborn	The Seaborn library was applied to visualize and explore similarly to Pyplot. It was used to provide a visual of the distributions and

	outliers of the variables via bar plots and boxplots.
from sklearn.feature_selection import SelectKBest, f_regression	The SelectKBest, f_regression module of the Scikit Learn Library was used to perform feature selection. The f_regression module calculated the cross-correlation between the variables and then converted it into F-statistics. This statistic was then converted to a p-value (Pedregosa, 2011).
sklearn.metrics import mean_squared_error as MSE	The mean_squared_error module in Scikit Learn's library was used to calculate the average of the square of errors. It provided an average of the total of the square for each difference between the estimated and true values. This metric provided the accuracy metric for the model.
sklearn.model_selection import train_test_split	The train_test_split module from the Scikit Learn library was used to split the initial data set into training and test data sets. This allowed the predictive analysis to be performed. The split provided insight into how well the prediction model would work on unknown values.

sklearn.tree import DecisionTreeRegressor	The DecisionTreeRegressor module in the Scikit Learn library was used to perform predictive analysis on the telecommunication data set. The function computed the tenure of the customers in relation to the bandwidth usage, churn, and device protection add-on.
sklearn.tree import plot_tree	The Scikit Learn library's plot_tree module was utilized to visualize the decision tree of the data set. The decision tree was similar to a flow chart structure with the decisions as well as their result.

Please see below for the code to import packages and libraries:

```
#install libraries and packages to use within environment for analysis
import matplotlib.pyplot as plt #use for visualizations
import numpy as np #work with arrays for computations
import pandas as pd #data manipulation and cleaning, creation of dataframes
import seaborn as sns #visualizations
from sklearn.feature_selection import SelectKBest, f_regression #features selections
from sklearn.metrics import mean_squared_error as MSE # metric for mean squared error
from sklearn.model_selection import train_test_split #splitting data
from sklearn.tree import DecisionTreeRegressor #analysis method, regression tree
from sklearn.tree import plot_tree #plotting decision tree
```

Part III: Data Preparation

C1. Data Preprocessing Goal

The data preprocessing goal pertinent to the prediction method was encoding. The process of encoding the independent variables in the data set allowed for the decision tree

regression model to be used. Machine learning models required all input and output variables to be numeric (Brownlee, 2020). The two encoding methods used on the telecommunication dataset were Ordinal encoding and One-Hot-Encoding. Ordinal encoding was used on binary categorical variables such as "Churn" and "DeviceProtection". The "Yes" value was noted as a "1" and the "No" value was represented as a "0". Prior to the feature selection process, the nominal categorical variables of the data set were encoded via One-Hot-Encoding. This was accomplished using the get_dummies function in Pandas on all categorical data with more than two unique values. The conversion of attributes from categorical to numeric allowed for the variables to be computed in the predictive method, Decision Tree regression. It permitted the data to be fitted to provide predictions based on the target variable of "Tenure".

C2. Identification of Data Set Variables

The variables utilized for the data analysis of the telecommunication data set included the target variable of "Tenure", which was continuous and quantitative. The data set included three independent variables "DeviceProtection", "Bandwidth_GB_Year" and "Churn". The below table noted the variables as well as their data type and data class.

Count of Variables	**Independent Variables**	**Data Type**	**Data Class**
1	Bandwidth_GB_Year	Continuous	Quantitative
2	Churn	Categorical	Qualitative
3	DeviceProtection	Categorical	Qualitative

C3. Data Preparation Steps

The telecommunication data set required numerous data preparation sets before predictive analysis would be completed. It included the following phases: data cleaning, data

exploration, and data wrangling phases. Each of the steps was documented below along with their accompanying code.

Please note all code can also be found on the attached file: AFD209Task2.ipynb

1. The data set was imported into the Python environment. The Pandas function .read_csv() allowed the csv file to be loaded into the dataframe. The .head() function was used to confirm the data set was properly loaded by showing the first five rows of data.

```
#import the telecommunications churn dataset csv file to be used.
#view dataset to ensure proper loading.

df = pd.read_csv('churn_clean.csv')
pd.set_option('display.max_columns', None)
df.head()
```

2. Data exploration was performed on the dataframe by using the shape function. The function was applied to show the total number of rows and columns in the data set.

```
df.shape
```

3. Data exploration was continued by evaluating the dataframe's data types, variable labels, and the count of non-null values. This was accomplished by the .info() function.

```
df.info()
```

4. The unclear variables were renamed based on the telecommunication churn dictionary. The variables for renaming were Items 1 through 8. This process was completed via the .rename() function on each of the specified variables.

```
#renaming unclear variables.
```

```
df = df.rename(columns = { "Item1": "Timely_Respd", "Item2": "Timely_Fixes",
                           "Item3": "Timely_Replc", "Item4": "Reliability", "Item5": "Options",
                           "Item6": "Respect_Resp", "Item7": "Courteous_Exch",
                           "Item8": "Evidence_ActListen"})
```

5. Further data exploration was completed by summarizing the summary statistics. The statistics included the count, mean, standard deviation, minimum, maximum, and quantiles for each continuous variable. The .describe() function compiled this data in a table.

```
#summary statistics
df.describe()
```

6. Data cleaning was processed in three steps. The steps consisted of checking for outliers, missing values, and duplicates. The identification of duplicates was accessed via the .duplicated() function. The .isnull().sum() function was applied to provide the sum of null or missing values for each variable. There were no duplicates or missing values located within the telecommunication data set. The identification and treatment of outliers were accomplished using the IQR method. This method computed the IQR via the subtraction of the first quantile from the third quantile. The lower and upper limits for each variable were calculated utilizing the statistical calculations: Q1 – 1.5 * IQR = lower bound value and Q3 + 1.5 * IQR = upper bound value. The identified outliers were restricted to the nearest upper and lower boundaries by a process called capping. The NumPy .where() function was used to cap the outlier values to the nearest upper or lower boundary for each variable. Boxplots were visualized to show the different distributions and graphical points of the outlier locations for each variable.

```
#finding duplicates
df.duplicated(keep='last')

#finding nulls
df.isnull().sum()

#finding outliers
def find_outliers(df, var):
    q1 = df[var].quantile(0.25)
    q3 = df[var].quantile(0.75)
    IQR = q3 - q1
    lowerbound = q1-(1.5*IQR)
    upperbound = q3+(1.5*IQR)
    outliers = df[var][((df[var] < (lowerbound)) | (df[var] > (upperbound)))]
    return outliers

#running created function on quantitative variables

outliers = find_outliers(df, 'Population')
print("number of outliers in Population: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
print("min outlier value: "+ str(outliers.min()))

outliers = find_outliers(df, 'MonthlyCharge')
print("number of outliers in MonthlyCharge: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
print("min outlier value: "+ str(outliers.min()))

outliers = find_outliers(df, 'Children')
print("number of outliers in Children: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
print("min outlier value: "+ str(outliers.min()))

outliers = find_outliers(df, 'Age')
print("number of outliers in Age: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
print("min outlier value: "+ str(outliers.min()))

outliers = find_outliers(df, 'Income')
print("number of outliers in Income: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
print("min outlier value: "+ str(outliers.min()))

outliers = find_outliers(df, 'Yearly_equip_failure')
print("number of outliers in Yearly_equip_failure: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
print("min outlier value: "+ str(outliers.min()))

outliers = find_outliers(df, 'Outage_sec_perweek')
print("number of outliers in Outage_sec_perweek: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
```

```
print("min outlier value: "+ str(outliers.min()))

outliers = find_outliers(df, 'Email')
print("number of outliers in Email: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
print("min outlier value: "+ str(outliers.min()))

outliers = find_outliers(df, 'Contacts')
print("number of outliers in Contacts: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
print("min outlier value: "+ str(outliers.min()))

outliers = find_outliers(df, 'Bandwidth_GB_Year')
print("number of outliers in Bandwidth_GB_Year: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
print("min outlier value: "+ str(outliers.min()))

#boxplotting all variables showing outliers

boxplot=sns.boxplot(x='Population',data=df)
plt.show()
boxplot=sns.boxplot(x='Children',data=df)
plt.show()
boxplot=sns.boxplot(x='Income',data=df)
plt.show()
boxplot=sns.boxplot(x='Outage_sec_perweek',data=df)
plt.show()
boxplot=sns.boxplot(x='Email',data=df)
plt.show()
boxplot=sns.boxplot(x='Contacts',data=df)
plt.show()
boxplot=sns.boxplot(x='Yearly_equip_failure',data=df)
plt.show()

#treating outliers found.
def find_boundary(df, var):
    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3-Q1
    lower = Q1-(1.5*IQR)
    upper = Q3+(1.5*IQR)
    return lower , upper

lower_pop, upper_pop = find_boundary(df, 'Population' )
print("Upper limit for population is" , upper_pop)
print("Lower limit for population is" , lower_pop)
df.Population = np.where(df.Population > upper_pop, upper_pop,
                         np.where(df.Population < lower_pop, lower_pop,
df.Population))

lower_kid, upper_kid = find_boundary(df, 'Children')
```

```
print("Upper limit for children is" , upper_kid)
print("Lower limit for children is" , lower_kid)
df.Children = np.where(df.Children > upper_kid, upper_kid,
                       np.where(df.Children < lower_kid, lower_kid,
df.Children))

lower_inc, upper_inc = find_boundary(df, 'Income')
print("Upper limit for Income is" , upper_inc)
print("Lower limit for Income is" , lower_inc)
df.Income = np.where(df.Income > upper_inc, upper_inc,
                       np.where(df.Income < lower_inc, lower_inc,
df.Income))

lower_osp, upper_osp = find_boundary(df, 'Outage_sec_perweek')
print("Upper limit for Outage_sec_perweek is" , upper_osp)
print("Lower limit for Outage_sec_perweek is" , lower_osp)
df.Outage_sec_perweek = np.where(df.Outage_sec_perweek >
upper_osp, upper_osp,
                       np.where(df.Outage_sec_perweek < lower_osp,
lower_osp, df.Outage_sec_perweek))

lower_eml, upper_eml = find_boundary(df, 'Email')
print("Upper limit for email is" , upper_eml)
print("Lower limit for email is" , lower_eml)
df.Email = np.where(df.Email > upper_eml, upper_eml,
                       np.where(df.Email < lower_eml, lower_eml, df.Email))

lower_contct, upper_contct = find_boundary(df, 'Contacts')
print("Upper limit for contacts is" , upper_contct)
print("Lower limit for contacts is" , lower_contct)
df.Contacts = np.where(df.Contacts > upper_contct, upper_contct,
                       np.where(df.Contacts < lower_contct, lower_contct,
df.Contacts))

lower_yef, upper_yef = find_boundary(df, 'Yearly_equip_failure')
print("Upper limit for contacts is" , upper_yef)
print("Lower limit for contacts is" , lower_yef)
df.Yearly_equip_failure = np.where(df.Yearly_equip_failure > upper_yef,
upper_yef,
                       np.where(df.Yearly_equip_failure < lower_yef,
lower_yef, df.Yearly_equip_failure))

#re-boxplotting all variables showing outliers

boxplot=sns.boxplot(x='Population',data=df)
plt.show()
boxplot=sns.boxplot(x='Children',data=df)
plt.show()
boxplot=sns.boxplot(x='Income',data=df)
plt.show()
boxplot=sns.boxplot(x='Outage_sec_perweek',data=df)
```

```
plt.show()
boxplot=sns.boxplot(x='Email',data=df)
plt.show()
boxplot=sns.boxplot(x='Contacts',data=df)
plt.show()
boxplot=sns.boxplot(x='Yearly_equip_failure',data=df)
plt.show()
```

7. The categorical variables with a high number of unique values were located using the .nunique() function. The number of unique values within each variable was called cardinality. This was checked on each variable to ensure that variables with a value greater than five were removed. The removal of these variables, as well as unneeded variables, was accomplished using the .drop() function.

```
df.columns
#printing the unique values for each variable to check cardinality

print(f'CaseOrder: {df.CaseOrder.nunique()}')
print(f'Customer_id: {df.Customer_id.nunique()}')
print(f'Interaction: {df.Interaction.nunique()}')
print(f'UID: {df.UID.nunique()}')
print(f'City: {df.City.nunique()}')
print(f'State: {df.State.nunique()}')
print(f'County: {df.County.nunique()}')
print(f'Zip: {df.Zip.nunique()}')
print(f'Lat: {df.Lat.nunique()}')
print(f'Lng: {df.Lng.nunique()}')
print(f'Population: {df.Population.nunique()}')
print(f'Area: {df.Area.nunique()}')
print(f'TimeZone: {df.TimeZone.nunique()}')
print(f'Job: {df.Job.nunique()}')
print(f'Children: {df.Children.nunique()}')
print(f'Age: {df.Age.nunique()}')
print(f'Income: {df.Income.nunique()}')
print(f'Marital: {df.Marital.nunique()}')
print(f'Gender: {df.Gender.nunique()}')
print(f'Churn: {df.Churn.nunique()}')
print(f'Outage_sec_perweek: {df.Outage_sec_perweek.nunique()}')
print(f'Email: {df.Email.nunique()}')
print(f'Contacts: {df.Contacts.nunique()}')
print(f'Yearly_equip_failure: {df.Yearly_equip_failure.nunique()}')
print(f'Techie: {df.Techie.nunique()}')
print(f'Contract: {df.Contract.nunique()}')
print(f'Port_modem: {df.Port_modem.nunique()}')
```

```
print(f'Tablet: {df.Tablet.nunique()}')
print(f'InternetService: {df.InternetService.nunique()}')
print(f'Phone: {df.Phone.nunique()}')
print(f'Multiple: {df.Multiple.nunique()}')
print(f'OnlineSecurity: {df.OnlineSecurity.nunique()}')
print(f'OnlineBackup: {df.OnlineBackup.nunique()}')
print(f'DeviceProtection: {df.DeviceProtection.nunique()}')
print(f'TechSupport: {df.TechSupport.nunique()}')
print(f'StreamingTV: {df.StreamingTV.nunique()}')
print(f'StreamingMovies: {df.StreamingMovies.nunique()}')
print(f'PaperlessBilling: {df.PaperlessBilling.nunique()}')
print(f'PaymentMethod: {df.PaymentMethod.nunique()}')
print(f'Tenure: {df.Tenure.nunique()}')
print(f'MonthlyCharge: {df.MonthlyCharge.nunique()}')
print(f'Bandwidth_GB_Year: {df.Bandwidth_GB_Year.nunique()}')
print(f'Timely_Respd: {df.Timely_Respd.nunique()}')
print(f'Timely_Fixes: {df.Timely_Fixes.nunique()}')
print(f'Timely_Replc: {df.Timely_Replc.nunique()}')
print(f'Reliability: {df.Reliability.nunique()}')
print(f'Options: {df.Options.nunique()}')
print(f'Respect_Resp: {df.Respect_Resp.nunique()}')
print(f'Courteous_Exch: {df.Courteous_Exch.nunique()}')
print(f'Evidence_ActListen: {df.Evidence_ActListen.nunique()}')

#dropping categorical variables with high cardinality and unneeded
variables
df.drop(['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State',
      'County', 'Zip', 'Lat', 'Lng', 'Population', 'TimeZone', 'Job',
      'Email', 'Contacts', 'Yearly_equip_failure',
      'Port_modem', 'Tablet', 'Phone', 'Multiple', 'OnlineSecurity',
'OnlineBackup',
      'StreamingTV', 'StreamingMovies', 'PaperlessBilling',
'PaymentMethod',
      'Timely_Respd', 'Timely_Fixes', 'Timely_Replc',
      'Reliability', 'Options', 'Respect_Resp', 'Courteous_Exch',
      'Evidence_ActListen'], axis=1, inplace=True)
df.info()
df.describe()
```

8. After the removal of high cardinality variables, the categorical variables were re-expressed by encoding methods. The categorical variables with only two values were encoded via ordinal encoding. The nominal categorical variables were re-expressed by one-hot encoding. Ordinal encoding was completed using the cat.codes function. The get_dummies function was used for one-hot encoding. This function created dummy attributes of each unique value found within each feature variable. After the

variable creation, one column from each grouping was removed to reduce multicollinearity.

```
#re-expression of independent variables
#get dummies to complete one hot encoding on nominal categorical variables
df= pd.get_dummies(df, columns=['Area', 'Marital', 'Gender', 'Contract', 'InternetService'], prefix_sep=" " , drop_first=True)

#convert ordinal categorical to numerical

df['Techie']=df['Techie'].astype('category')
df['Techie']=df['Techie'].cat.codes
df['DeviceProtection']=df['DeviceProtection'].astype('category')
df['DeviceProtection']=df['DeviceProtection'].cat.codes
df['TechSupport']=df['TechSupport'].astype('category')
df['TechSupport']=df['TechSupport'].cat.codes
df['Churn']=df['Churn'].astype('category')
df['Churn']=df['Churn'].cat.codes

df.head()
```

9. Feature selection was completed after the encoding process. This was accomplished by using the SelectKBest function. The function returned the independent variables that provided the best “k” value and corresponding p-value. The variables that were deemed not optimal were dropped from the dataframe.

```
##feature selection method SelectKBest
X= df.drop(['Tenure'], axis=1)
y=df['Tenure']

names_features= X.columns
skb = SelectKBest(score_func = f_regression, k='all')
X_skb = skb.fit_transform(X, y)

#p-values of statistically significant features
p_values = pd.DataFrame({'Feature': names_features,
'p_value':skb.pvalues_}).sort_values('p_value')
p_values[p_values['p_value'] < 0.05]
features = p_values['Feature'][p_values['p_value'] < 0.05]
features

df.columns
#dropping unneeded variables
```

```
df.drop(['Children', 'Age', 'Income', 'Outage_sec_perweek', 'Techie',
    'TechSupport', 'MonthlyCharge', 'Area Suburban', 'Area Urban',
'Marital Married',
    'Marital Never Married', 'Marital Separated', 'Marital Widowed',
    'Gender Male', 'Gender Nonbinary', 'Contract One year',
    'Contract Two Year', 'InternetService Fiber Optic',
    'InternetService None'], axis=1, inplace=True)
```

10. After the completion of this process, data exploration was done to visualize the distributions of the remaining variables. This was shown via bar plots, scatterplots, and displots. The scatterplots and bar plots were processed using the Seaborn functions of .displot() and .scatterplot(). The Matplotlib.pyplot .bar()function was used to visualize the grouped "Churn" and "DeviceProtection" variables.

```
#data exploration to show distribution of variables.
sns.displot(df['Tenure'], kde=False, color='pink', bins=7)
sns.displot(df['Bandwidth_GB_Year'], kde=False, color='blue', bins=7)
groupedDeviceProtection = df.groupby(by='DeviceProtection').size()
groupedDeviceProtection
%matplotlib inline
groupedDeviceProtection.plot.bar(color='green')
groupedChurn = df.groupby(by='Churn').size()
groupedChurn
%matplotlib inline
groupedChurn.plot.bar(color='red')

#scatterplot of continuous variables
sns.scatterplot(data=df, x="Bandwidth_GB_Year", y="Tenure",
hue="Churn", style="DeviceProtection")
```

11. Lastly, a correlation matrix and heatmap were presented. The correlation matrix of the dataframe provided the correlation of the variable compared to each other variable. It was computed using the .corr() and then rounded to the neared hundredth place value. The results were visualized on a heatmap.

```
#correlation matrix and heatmap to visualize remaining variables
corr_matrix= df.corr().round(2)
features = corr_matrix.index
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True)
plt.show()
```

C4. Cleaned Data Set

Please see the attached CSV file called AFD209Task2_clean.csv to view the results of the cleaned dataset.

Part IV: Analysis

D1. Splitting Data: Training and Test

The cleaned data set was separated into two individual data sets via the .train_test_split() function from the Scikit Learn Library. The split on the telecommunication data set had a 30 to 70 percent ratio for both the target and feature variables. The target variable, "Tenure" was noted as the "y" value. The feature variables of "Bandwidth_GB_Year", "Churn" and "DeviceProtection" were encompassed as the "X" value. The training set consisted of 7000 entries or 70% of the original data set. The test data set contained the remaining 3000 entries or 30% of the initial data set.

Please see the attached CSV files to view the outcomes of the split datasets: AFD209Task2_xtest.csv; AFD209Task2_ytest.csv; AFD209Task2_xtrain.csv; AFD209Task2_ytrain.csv

D2. Analysis Technique

The predictive analysis technique performed on the telecommunication data set was a Decision Tree for Regression. The Decision Tree for Regression or Regression Tree algorithm was a prediction method, which built models in a tree-like structure (Kawerk, n.d.). The data sets were broken down into smaller subsets as the decision tree was created. This provided a tree with decision and leaf nodes. The decision nodes had various branches until they reached a

terminating leaf node. The leaf node would represent each final decision or tenure value in months (Coursera, 2022). To perform this method, the initial cleaned data set was split into test and training data sets. The .train_test_split() function was utilized to accomplish this objective. The "y" data embodied the target variable of "Tenure". The "X" data was symbolized as the three independent variables of "Bandwidth_GB_Year", "Churn" and "DeviceProtection". Both variables were inputted into the function's parameters along with the test_size value and random_state. The test_size was notated as 0.30, which meant 30% of the initial data was allocated to the test data set. The remaining 70% was assigned to the training data set.

After the separation of the data sets, the Decision Tree regression model was computed on the training and test data sets. This was accomplished by instantiating the model object using the max_depth, min_samples_leaf, and random_state parameters on the DecisionTreeRegressor function. The max_depth was the maximum depth of the tree. The parameter was set to 3 to keep the tree depth small to avoid overfitting (Kawerk, n.d.). The min_samples_leaf was set to 0.1. This was noted as the minimum number of samples a leaf must have.

```
dt= DecisionTreeRegressor(max_depth=3,
                          min_samples_leaf=0.1,
                            random_state=0)
```

The DecisionTreeRegressor was fitted to the training data set by setting the factors of X_train and y_train. The following computation outcome was returned.

```
DecisionTreeRegressor(max_depth=3, min_samples_leaf=0.1, random_state=0)
```

The Decision Tree Regressor model R- squared measurement was calculated for the test and training data set. The test data set had an R-squared of about 97%. This measurement meant that 97% of the variability of the target variable could be explained by the independent variables.

The parameters used on the .score() were X_test and y_test. The following output could be viewed below.

```
0.9741638156752763
```

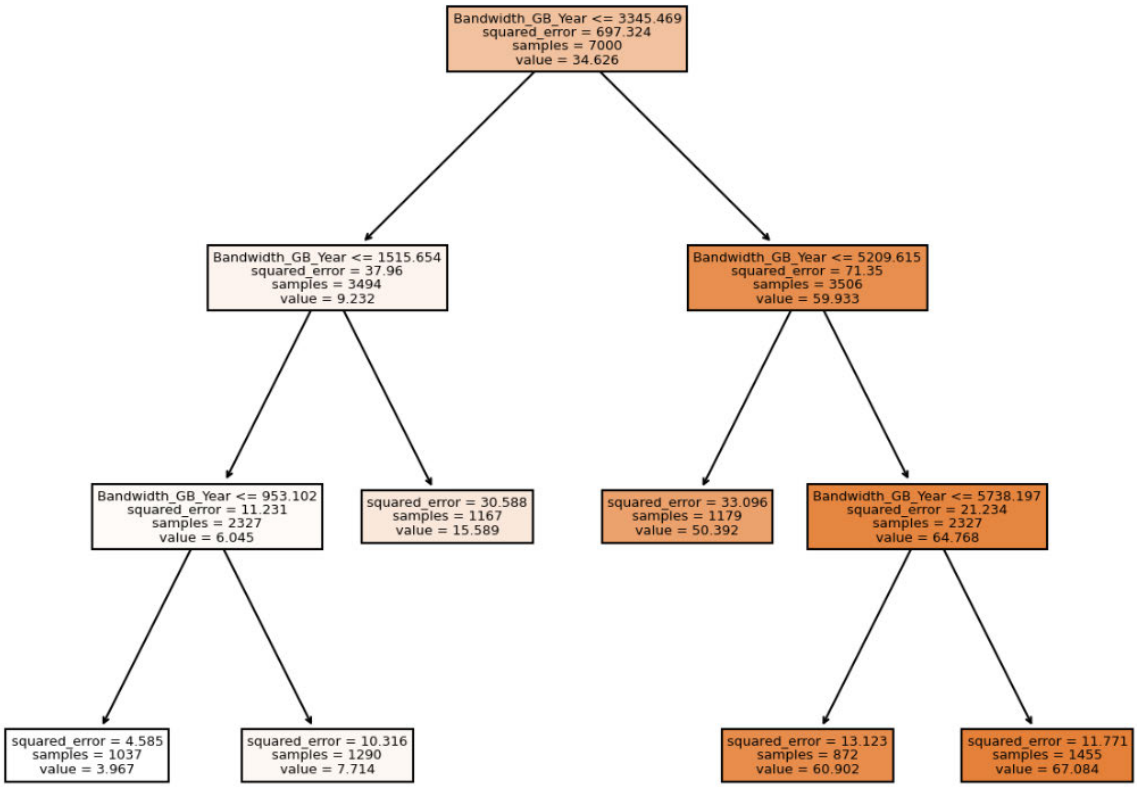
The training data set R- squared was about 98%. The parameters used to provide the following output result were X_train and y_train.

```
0.9751398137443361
```

The .predict() function was used on the feature variables for the test data set to provide predictions for the target value. The following predictions were returned.

```
array([60.90192806,  3.96742232,  7.71446956, ..., 67.08436841,
       50.39166172, 60.90192806])
```

The plot_tree module was another tool used to show the results of the decision tree. It provided the decision tree as a visualization to provide the root node, decision nodes, and ending leaf nodes. The telecommunication data set had the “Bandwidth_GB_Year” as the root node.



After the completion of the predictive method, hyperparameter tuning was applied using the .get_params() function on the Decision Tree Regressor. It provided the best parameters for the telecommunication data set when utilizing Decision Tree Regressor. The outcome was noted below.

```
{'ccp_alpha': 0.0, 'criterion': 'squared_error', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity
_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_state': 1, 'splitter':
'best'}
```

The same process for predictive analysis was completed using these best parameters to compare the results. The decision tree regressor was instantiated and fitted to provide this output:

```
dtbest= DecisionTreeRegressor(ccp_alpha = 0.0, criterion='squared_error',
                              max_depth= None, max_features= None, max_leaf_nodes= None,
                              min_impurity_decrease= 0.0, min_samples_leaf=1, min_samples_split= 2,
                              min_weight_fraction_leaf= 0.0,
                              random_state = 1, splitter= 'best')
```

```
DecisionTreeRegressor(random_state=1)
```

The R-squared value for the tuned training data set was 100% with the output as follows:

```
1.0
```

The R-Squared for the tuned test data set after inputting the parameters of X_test and y_test was about 99%. This is shown below.

```
0.9952823977552193
```

Lastly, the tuned test predictions were computed with the following outputs.

```
array([65.38502 ,  2.477945,  6.184958, ..., 68.6654  , 44.39952 ,
       67.69557 ])
```

The hyperparameter tuning provided a different result compared to the original parameters. It had about a 2 % increase in the coefficient of determination or R-squared measure for the tuned model versus the original model. The tuned model was not visualized in a plot tree as the max_depth parameter was set to None. It would cause the model to run indefinitely until the leaf node was pure (Kawerk, n.d.).

D3. Prediction Analysis Code

Please see the below Python code for the predictive analysis technique.

```
#splitting dataset into training and test data via train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0)

#decisiontree/ regression tree analysis
dt= DecisionTreeRegressor(max_depth=3,
                          min_samples_leaf=0.1,
                           random_state=0)

#fitting Decision Tree on training set
```

```
dt.fit(X_train, y_train)

#R2 coefficient of decision tree training data
dt.score(X_train, y_train)

#R2 coefficient of decision tree test data
dt.score(X_test, y_test)


#predicting target variables on test data.
y_pred=dt.predict(X_test)
y_pred

#plotting Decision Tree
plt.figure(figsize=(10,8), dpi=150)
plot_tree(dt, feature_names=X.columns, filled=True)

#hyperparameter tuning via best params
SEED = 1

dtr= DecisionTreeRegressor(random_state=SEED)
print(dtr.get_params())

dtbest= DecisionTreeRegressor(ccp_alpha = 0.0, criterion='squared_error',
                    max_depth= None, max_features= None, max_leaf_nodes= None,
                    min_impurity_decrease= 0.0, min_samples_leaf=1, min_samples_split=
2,
                    min_weight_fraction_leaf= 0.0,
                    random_state = 1, splitter= 'best')

#fit hyperparameter tuned DT
dtbest.fit(X_train, y_train)

# R2 coefficient of decision tree training data
dtbest.score(X_train, y_train)

# R2 coefficient of decision tree test data
dtbest.score(X_test, y_test)

y_predbest=dtbest.predict(X_test)
y_predbest
```

Part V: Data Summary and Implications

E1. Accuracy and MSE

The prediction model had various evaluation metrics to give insight into if the Decision Tree method was ideal. The accuracy metrics included R-squared, mean squared error (MSE),

and the root mean squared error(RMSE). The R-squared of the model was noted as the relationship between the residual sum of squares and the total sum of squares. It was measured from the values of 0 to 1. The closer the R-squared value was to 1 or 100% deemed the model a better goodness of fit. The .score() method was able to provide the measurement for the training and test data sets (Kawerk, n.d.). The test R-Squared measurement was 97% and the training R-Squared measurement was about 98%.

```
#R2 of decision tree training data
dt.score(X_train, y_train)

#R2 of decision tree test data
dt.score(X_test, y_test)
```

In comparison, the hyperparameter-tuned R-Squared measurements were 2 % higher. The tuned training R-squared value was 100%. This deemed the tuned model a perfect measure of fit on the training set. The tuned training R-Squared measurement was about 99%.

```
#R2 of decision tree tuned training data
dtbest.score(X_train, y_train)

#R2 of decision tree tuned test data
dtbest.score(X_test, y_test)
```

The next evaluation metric was the mean squared error. The mean squared error or MSE was the difference between the predicted values and the actual values (Frost, n.d.). This determined how good the prediction was based on the decision tree algorithm. The lower the MSE, the better the predictions. A perfect model with no error had an MSE of zero. This was not found in the telecommunication data set. The mean squared error value for the model was 18.172.

```
#calculating Mean Squared Error MSE
dt_mse= MSE(y_test, y_pred)
dt_mse
```

In comparison, the tuned model had a mean squared error of 3.318. This determined the hyperparameter-tuned model had better predictions as it had a lower value.

```
#calculating Mean Squared Error MSE on hyperparams.
dtbest_mse= MSE(y_test, y_predbest)
dtbest_mse
```

The last accuracy evaluation metric was Root Mean Squared Error or RMSE. This metric provided the average squared distance between the predicted values by the model and the actual observed values (Bobbitt, 2021). The square root of the variance of residual errors or RMSE for the original model was 4.262.

```
# calculate root mean squared error
dt_rmse= dt_mse**(1/2)
dt_rmse
```

The root mean squared error for the tuned model was 1.821.

```
# calculate root mean squared error
dtbest_rmse= dtbest_mse**(1/2)
dtbest_rmse
```

E2. Results and Implications

The results of the predictive analysis could be condensed in the following notes.

1. Selection of Features
2. Training and Test Data sets
3. Prediction model and MSE
4. Hyperparameter Tuning

The selection of features for the predictive analysis was based on the cross-correlation between the feature variable and the target variable (Pedregosa, 2011). This cross-correlation

was computed and converted to an F-statistic score than a p-value (Pedregosa, 2011). This is completed using the f_regression with the SelectKBest function. The p-value was assigned to the significance level of 0.05. The variables with a p-value of less than or equal to 0.05 were returned. This provided the ideal variables associated with the target variable, “Tenure”. The features that met this threshold were “Bandwidth_GB_Year”, “DeviceProtection” and “Churn”.

The machine learning model, Decision Tree for Regression, required the data set to be split into test and training data sets for analysis. The test data set was utilized to predict the monthly unknown value for the tenure of a customer. The test data set was a sample of 3000 entries or 30% of the original data. The training data set was the remaining 7000 entries or 70% of the original data set. The target variable was represented as the “y” array to use as a class label. The feature variables were assigned to the “X” array for computations.

The Decision Tree regression model provided an MSE score to identify the prediction performance of the model. The mean squared error score provided the difference between the predicted values and the actual values. In the telecommunication data set, the MSE value was 18.172. The use of hyperparameter tuning provided a great view of how the parameters could affect the MSE score of the data. The hyperparameter-tuned model had a better MSE value of 3.318. The predictions had an improvement of 14.862 in comparison to the initial model.

Hyperparameter tuning optimized the data model in all accuracy and evaluation metrics. The usage of the best parameters increased the goodness of fit between the models by 2%. Before the tuning, the model test data set had an R-squared value of 0.97. After the tuning, the goodness of fit measure of R-squared was 0.99. The RMSE value also showed improvement as well. The RMSE value for the initial data set was 4.262. The squared average between the

predicted values of the model and the actual value decreased by about 3 when hyperparameters were tuned. The tuned model had an RMSE of 1.821.

The implication of the predictive analysis, Decision Tree regression was the prediction of the customer's tenure could be adequately forecasted based on using the tuned hyperparameters. The tuned model had a very little distance between the predicted values and the actual values as shown in the root mean squared error calculation outcome. Another conclusion of the analysis was the identification of contributing factors. The selection of the feature variables provided the stakeholders with the key contributors for increased customer longevity (Expert Panel, Forbes Agency Council, 2019). The analysis could supply the telecommunication company with information on the forecasted total months of customer tenure. This would be used to provide sales and marketing tactics on various add-ons and services (Expert Panel, Forbes Agency Council, 2019).

E3. Limitation of Data Analysis

A limitation of the Decision Tree prediction method was the high probability of overfitting. The model was dependent on the depth of the tree. The larger the decision tree depth, the more prone the model was affected by overfitting. The increase of the max_depth parameter would increase the decision nodes and branches of the tree structure (Sharma, 2020).

For example, if the max_depth of the model was set to 100, it would keep splitting the nodes until the tree reached that level or the data set was pure. It would stop at whichever came first. Overfitting would occur if the data set couldn't account for that amount of nodes. It would give a false indication of a perfect model (Sharma, 2020).

E4. Recommendation of Action

The recommended course of action for the telecommunication company was to use the Decision Tree regression to predict the customer tenure. The decision tree regression method should be used after performing pruning on the hyperparameter-tuned model. This would reduce the observance of overfitting.

The Decision Tree regression model responded to the research thesis, "Is it possible to predict a customer's tenure using the Decision Tree method and pinpoint the contributing variables for the telecommunication stakeholders?". The data analysis provided the key contributing factors to customer tenure. The key factors included "Bandwidth_GB_Year", "Churn", and "DeviceProtection".

The model goodness of fit was determined by the R-squared value. The prediction model provided a high R-squared of 97% deeming it a good fit for prediction. The model also included the squared average distance between the predicted variables and the actual variables. The value was shown as the root mean squared error. The value was low at 4.262 for the original model and 1.821 for the tuned model. Both of which determined that the prediction accuracy was ideal.

The stakeholders of the organization can use this information to pinpoint predictions on how long a customer will stay with the company. This will provide insight into what increases customer loyalty (Van, 2020). The key contributing factors could be utilized as areas to increase marketing and sales strategies (Expert Panel, Forbes Agency Council, 2019).

Part VI: Demonstration

F. Panopto Video

The Panopto video provided a summary of the Jupyter Notebook environment and the Python code. The recording demonstrated the code's warning and error-free functionality.

Please see attached Panopto video link. Link Found here:

[redacted]

G. Third-Party Web Sources

There were no third party web sources used to acquire data nor segments of third-party code to support the analysis. All code was original work.

H. References

Bobbitt, Z. (2021, May 10). *How to Interpret Root Mean Square Error (RMSE)*. Retrieved from Statology: https://www.statology.org/how-to-interpret-rmse/

Brownlee, J. (2020, June 12). *Machine Learning Mastery*. Retrieved from Ordinal and One-Hot_Encodings for Categorical Data: https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/

Coursera. (2022, September 27). *Decision Trees in Machine Learning: Two Types*. Retrieved from Coursera: https://www.coursera.org/articles/decision-tree-machine-learning

Expert Panel, Forbes Agency Council. (2019, December 20). *How To Increase Client Retention: 10 Effective Strategies*. Retrieved from Forbes: https://www.forbes.com/sites/forbesagencycouncil/2019/12/20/how-to-increase-client-retention-10-effective-strategies/?sh=3f663798390b

Frost, J. (n.d.). *Mean Squared Error*. Retrieved from Statistics by Jim: https://statisticsbyjim.com/regression/mean-squared-error-mse/

IBM. (n.d.). *What is a Decision Tree?* Retrieved from IBM: https://www.ibm.com/topics/decision-trees

Kawerk, E. (n.d.). *Machine Learning with Tree-Based Models In Python*. Retrieved from Datacamp: https://app.datacamp.com/learn/courses/machine-learning-with-tree-based-models-in-python

Pedregosa, e. a. (2011). *Scikit-learn: Machine Learning in Python*. Retrieved from Scikit-Learn: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html#sklearn.feature_selection.SelectKBest

Sharma, R. (2020, December 10). *Guide to Decision Tree Algorithm: Applications, Pros & Cons & Example*. Retrieved from upGrad blog: https://www.upgrad.com/blog/guide-to-decision-tree-algorithm/

Van, H. L. (2020, July 13). *How to measure customer retention? 7 Key metrics you need to know*. Retrieved from Omniconvert: https://www.omniconvert.com/blog/how-to-measure-customer-retention-7-key-metrics/