# D209 - Data Mining I

## Performance Assessment - Task 2: Predictive Analysis

## *Telecommunications Churn Data Set (Clean)*

## Part I - Research Question

### A1: Proposal of Question

The central research question addressed by this analysis is to determine:

Can these data be used to reliably predict customer retention (Tenure) via a random forest regression model and, if so, which features are most important?

With regard to hypothesis testing, our null hypothesis () is:

There are no reliable predictors of customer retention (Tenure) that can be observed via a random forest regression model of the available data.

Additionally, our alternate hypothesis () is:

Customer retention (Tenure) can be predicticted with reasonable reliability via a random forest regression model of the available data.

### A2: Defined Goal

The primary goal of the following analysis is to discover whether or not the tenure (Tenure) of a customer can be predicted with a reasonable level of accuracy using the available data in the *Telecommunications Churn* dataset. Additionally, provided the features in the dataset allow for this prediction, the secondary goal will be to identify which features are most useful in predicting tenure. Knowing which features most contribute to the prediction will allow for additional resources to be utilized in gathering precise and timely data and focusing an appropriate proportion of retention efforts in the right direction. This analysis will be conducted using the Python programming language using libraries including pandas, sklearn, plotly, and others to acheive the stated objectives.

## Part II - Method Justification

### B1: Explanation of Prediction Method

The selected prediction method, RandomForestRegressor, is an ensmble method wherein the base estimator is the decision tree regressor. (Pedregosa, et al., 2011) The base decision tree works within the confines of predetermined hyperparameters which dictate a cost function (when decision tree is regressor), maxmimum depth, and a number of other optional parameters. In the case of this analysis, the selected cost function is mean squared error (MSE). The underlying decision tree uses MSE to determine the feature and criteria that will result in a binary split of the training set that minimizes the cost function. If no other hyperparameter is employed, the process is repeated recursively until no split can be made that would reduce the cost function. The hyperparameter max\_depth indicates a condition which, if met, causes the cessation of recursive splitting. The depth of a tree is increased by 1 for each recursive split. So, for example, a tree that splits the initial dataset, then splits the two subsets, then splits those four sub-subsets would have a depth equal to 3. As mentioned, additional hyperparameters offer other criteria to be met in similar fashion. Due to the issues of overfitting and sensitivity, decision trees on their own are often not quite as useful as random forests, which employ an armada of randomized decision trees and aggregate the many trees' outputs to arrive at a result. (Géron, 2019) The RandomForestRegressor is instantiated with a determined number of estimators which is the total number of underlying decision trees. Additionally, the same hyperparameters can be defined such as max\_depth, min\_samples\_leaf, etc. Instead of choosing the feature that best reduces the cost function, a random forest selects a subset of features at random (as the name implies) and then selects the feature among the random subset which best reduces the cost function, which mitigates some of the aforementioned drawbacks of decision trees. Finally, the result is the mean output of all estimators for a given record. The expected outcome of the model is the predicted Tenure (in months) of a given record based on the predictor features of that record. A reasonably reliable model could potentially help identify the most important predictors of tenure.

### B2: Summary of Method Assumptions

The RandomForestRegressor assumes that the sampling of training data is truly representative of the population. Because the model in self-contained within the bounds of the training data, it is range-bound to the training set. Therefore, the model assumes the training data are appropriately representative.

### B3: Packages/Libraries List

The following Python libraries and packages will be utilized in this analysis:

* pandas
* numpy
* plotly.express
* ColumnTransformer from sklearn.compose
* OneHotEncoder from sklearn.preprocessing
* train\_test\_split & RandomizedSearchCV from sklearn.model\_selection
* RandomForestRegressor from sklearn.ensemble
* mean\_squared\_error from sklearn.metrics
* residuals\_plot from yellowbrick.regressor
* feature\_importances from yellowbrick.model\_selection
* dtreeviz.trees

**Pandas**

* The pandas library will be heavily relied upon for the initial import, filtering, and general preparation of the data prior to running our analysis.

**Numpy**

* The numpy library will be used to create arrays for cross-validation and hyperparameter tuning of the model.

**Plotly Express**

* The Plotly Express library will be used to graphically represent the composition of the data and any other similar visual as needed.

**ColumnTransformer**

* The ColumnTransformer package from the sklearn.compose library will be used to handle pre-processing treatment for columns of different dtypes.

**OneHotEncoder**

* The OneHotEncoder package from the sklearn.preprocessing library will be used to tranform categorical features in pre-processing.

**Train\_test\_split**

* The train\_test\_split package from the sklearn.model\_selection library will be used to facilitate the splitting of the sub-selected dataset. It provides a quick and easy way to select a random sample to reserve for testing the model once trained.

**RandomizedSearchCV**

* The RandomizedSearchCV package from the sklearn.model\_selection library will be used to cross-validate the model and assist in hyperparameter tuning.

**RandomForestRegressor**

* The RandomForestRegressor package from the sklearn.ensemble library is the primary estimator of the model.

**Mean\_squared\_error**

* The mean\_squared\_error package from the sklearn.metrics library will be used to evaluate the accuracy model.

**Residuals\_plot**

* The residuals\_plot package from the yellowbrick.regressor library will be used to plot the residuals of the final model.

**Feature\_importances**

* The Feature\_importances package from the yellowbrick.model\_selection library will be used to visualize the most important features of the final model.

**DTreeViz Trees**

* The dtreeviz.trees library will be used to graphically represent an example of the underlying decision trees.

## Part III - Data Preparation

### C1: Data Preprocessing

The process we will need to complete in order to prepare the data for training is relatively minor, given that the raw dataset used in this project has already been cleaned in a prior project. Using the pre-cleaned dataset, we will first partition the data to include only those variables we intend to feed into our model. As described above, our model input will consist of the target feature Tenure (continuous) and will include 26 predictor features (continuous, discrete, and categorical).

Next, we will need to ensure that the data type of each variable is appropriate for that kind of feature. Finally, we will ensure that categorical features are transformed with OneHotEncoder to make it easier for the model to handle and then split the dataset into training and testing sets with proportions of 70% and 30% respectively.

### C2: Dataset Variables

As addressed above, the variables to be used in the model and analysis will consist of the below:

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | DType | DSubtype | Feature Type |
| Tenure | Numeric | Continuous | Target |
| Children | Numeric | Discrete | Predictor |
| Age | Numeric | Discrete | Predictor |
| Income | Numeric | Continuous | Predictor |
| Marital | Categorical | Nominal | Predictor |
| Gender | Categorical | Nominal | Predictor |
| Outage\_sec\_perweek | Numeric | Continuous | Predictor |
| Email | Numeric | Discrete | Predictor |
| Contacts | Numeric | Discrete | Predictor |
| Yearly\_equip\_failure | Numeric | Discrete | Predictor |
| Techie | Categorical | Binary | Predictor |
| Contract | Categorical | Nominal | Predictor |
| Port\_modem | Categorical | Binary | Predictor |
| Tablet | Categorical | Binary | Predictor |
| InternetService | Categorical | Nominal | Predictor |
| Phone | Categorical | Binary | Predictor |
| Multiple | Categorical | Binary | Predictor |
| OnlineSecurity | Categorical | Binary | Predictor |
| OnlineBackup | Categorical | Binary | Predictor |
| DeviceProtection | Categorical | Binary | Predictor |
| TechSupport | Categorical | Binary | Predictor |
| StreamingTV | Categorical | Binary | Predictor |
| StreamingMovies | Categorical | Binary | Predictor |
| PaperlessBilling | Categorical | Binary | Predictor |
| PaymentMethod | Categorical | Nominal | Predictor |
| MonthlyCharge | Numeric | Continuous | Predictor |
| Bandwidth\_GB\_Year | Numeric | Continuous | Predictor |

### C3: Steps for Analysis

The following steps were taken to perform the analysis:

*The steps enumerated correspond to code segments in section D3*

* Step 1 - Load in libraries and dataset
  + This initial step involves importing the necessary libraries and modules as well as reading-in the initial dataset. Finally, the initial dataset is inspected using the .info() method to take a quick glance at all of the features and ensure no NaNs are present.
* Step 2 - Subset data & initial EDA
  + This step takes the initial dataset and selects out the features that will be used in the analysis. Those features are explored using descriptive statistics and some simple visualizations.
* Step 3 - Prepare subset data for analysis
  + The subset data is then prepared by ensuring the correct datatypes are used (setting categorical features to category type), then using ColumnTransformer and OneHotEncoder to encode those categorical features. Finally, the dataset is split using train\_test\_split for training and testing.
* Step 4 - Initial model fit and evaluation
  + A range of hyperparameters is given for the process of hyperparameter tuning. The estimator RandomForestRegressor is then instantiated as well as the RandomizedSearchCV tool for cross-validation. The model is then fit using the training data.
* Step 5 - Revised model fit and evaluation
  + Once the model has been fit, the model is then validated on the training set to assess its accuracy. After this is complete, a secondary model is prepared to be fit using the learnings from the first model.
* Step 6 - Model visualizations
  + Finally, the model results are visualized using the yellowbrick and dtreeviz libraries to provide further intuition and clarity.

### C4: Cleaned Dataset

df.to\_csv('./data/d209\_task2\_cleaned\_dataset.csv')

*Please see attached d209\_task2\_cleaned\_dataset.csv file*

## Part IV - Analysis

### D1: Splitting the Data

We will use the module train\_test\_split from the sklearn library to split the dataset. This tool makes the process of splitting quite simple. The split percentage will be 70/30 with 70% of the total observations utilized in training the model, while the remaining 30% will be reserved for testing the model. Additionally, as described in the code below, a random seed of 42 was chosen for the randomization. Seed 42 was chosen at random as well and was used for sampling previously during EDA visualizations.

# Split data using 70/30 split and seed 42  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

### D2: Output & Intermediate Calculations

This analysis consists primarily of the and determining the optimal values for the model hyperparameters. After first preparing the data for analysis, EDA steps such as preparing descriptive statistics and visualizations of the data (through the use of scatter plots and histograms on the selected features) are used to gain a bit of perspective prior to modelling. After EDA, a preliminary model is instantiated and fit to the training data. The estimator is fed into a hyperparameter tuning tool () to handle both the cross-validation as well as optimization of hyperparameter values. The cross-validator will use five bootstrapped folds of the training set and contain a grid of the parameters n\_estimators, max\_depth, and min\_samples\_leaf with value ranges of 500-2000, 5-40, and 0.01-0.20 respectively. The five folds will be randomly iterated over 200 times each creating a total of 1,000 random fits of the . Once this is done, the CV results will be assessed. The best parameters will be considered along with the descriptive statistics of the top 1/2 (by CV score) of the assessed fits' parameters. Following that, revised hyperparameter ranges will be used and the process will repeat to produce our final model.

# Load in required packages  
import pandas as pd  
import numpy as np  
import plotly.express as px  
from sklearn.compose import ColumnTransformer  
from sklearn.preprocessing import OneHotEncoder  
from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error as MSE  
from yellowbrick.regressor import residuals\_plot  
from yellowbrick.model\_selection import feature\_importances  
from dtreeviz.trees import \*  
  
# Read in initial dataset and check .info()  
df = pd.read\_csv('./data/churn\_clean.csv')  
df.info()

<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 10000 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

# Subset data by features to be included in model  
df = df.iloc[:, np.r\_[14:19, 20:42]]  
df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 10000 entries, 0 to 9999  
Data columns (total 27 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Children 10000 non-null int64   
 1 Age 10000 non-null int64   
 2 Income 10000 non-null float64  
 3 Marital 10000 non-null object   
 4 Gender 10000 non-null object   
 5 Outage\_sec\_perweek 10000 non-null float64  
 6 Email 10000 non-null int64   
 7 Contacts 10000 non-null int64   
 8 Yearly\_equip\_failure 10000 non-null int64   
 9 Techie 10000 non-null object   
 10 Contract 10000 non-null object   
 11 Port\_modem 10000 non-null object   
 12 Tablet 10000 non-null object   
 13 InternetService 10000 non-null object   
 14 Phone 10000 non-null object   
 15 Multiple 10000 non-null object   
 16 OnlineSecurity 10000 non-null object   
 17 OnlineBackup 10000 non-null object   
 18 DeviceProtection 10000 non-null object   
 19 TechSupport 10000 non-null object   
 20 StreamingTV 10000 non-null object   
 21 StreamingMovies 10000 non-null object   
 22 PaperlessBilling 10000 non-null object   
 23 PaymentMethod 10000 non-null object   
 24 Tenure 10000 non-null float64  
 25 MonthlyCharge 10000 non-null float64  
 26 Bandwidth\_GB\_Year 10000 non-null float64  
dtypes: float64(5), int64(5), object(17)  
memory usage: 2.1+ MB

# Show descriptive statistics for numeric data  
df.describe().T

count mean std min \  
Children 10000.0 2.087700 2.147200 0.000000   
Age 10000.0 53.078400 20.698882 18.000000   
Income 10000.0 39806.926771 28199.916702 348.670000   
Outage\_sec\_perweek 10000.0 10.001848 2.976019 0.099747   
Email 10000.0 12.016000 3.025898 1.000000   
Contacts 10000.0 0.994200 0.988466 0.000000   
Yearly\_equip\_failure 10000.0 0.398000 0.635953 0.000000   
Tenure 10000.0 34.526188 26.443063 1.000259   
MonthlyCharge 10000.0 172.624816 42.943094 79.978860   
Bandwidth\_GB\_Year 10000.0 3392.341550 2185.294852 155.506715   
  
 25% 50% 75% max   
Children 0.000000 1.000000 3.000000 10.000000   
Age 35.000000 53.000000 71.000000 89.000000   
Income 19224.717500 33170.605000 53246.170000 258900.700000   
Outage\_sec\_perweek 8.018214 10.018560 11.969485 21.207230   
Email 10.000000 12.000000 14.000000 23.000000   
Contacts 0.000000 1.000000 2.000000 7.000000   
Yearly\_equip\_failure0.000000 0.000000 1.000000 6.000000   
Tenure 7.917694 35.430507 61.479795 71.999280   
MonthlyCharge 139.979239 167.484700 200.734725 290.160419   
Bandwidth\_GB\_Year1236.470827 3279.536903 5586.141370 7158.981530

# Show correlation between continuous features  
dffloats = df[df.columns[df.dtypes == 'float64']]  
dffloats.corr()

Income Outage\_sec\_perweek Tenure MonthlyCharge \  
Income 1.000000 -0.010011 0.002114 -0.003014   
Outage\_sec\_perweek -0.010011 1.000000 0.002932 0.020496   
Tenure 0.002114 0.002932 1.000000 -0.003337   
MonthlyCharge -0.003014 0.020496 -0.003337 1.000000   
Bandwidth\_GB\_Year 0.003674 0.004176 0.991495 0.060406

Bandwidth\_GB\_Year   
Income 0.003674   
Outage\_sec\_perweek 0.004176   
Tenure 0.991495   
MonthlyCharge 0.060406   
Bandwidth\_GB\_Year 1.000000

# Subset numeric features and show histograms  
df\_num\_cols = [x for x in df.describe().columns]  
df\_num = df[df\_num\_cols]  
for i in df\_num.columns:  
 px.histogram(  
 df\_num.sample(500, random\_state=42),  
 x=i,  
 width=500,  
 height=300,  
 nbins=30,  
 template='seaborn'  
 ).show()

|  |  |
| --- | --- |
| Chart  Description automatically generated | Chart, histogram  Description automatically generated |
| Chart, histogram  Description automatically generated | Chart, histogram  Description automatically generated |
| Chart, histogram  Description automatically generated | Chart, bar chart  Description automatically generated |
| Chart  Description automatically generated | Chart, histogram  Description automatically generated |
| Chart, histogram  Description automatically generated | Chart, histogram  Description automatically generated |

# Plot Tenure vs Outage\_sec\_perweek  
px.scatter(df.sample(500, random\_state=42),  
 x='Tenure',  
 y='Outage\_sec\_perweek',  
 width=700,  
 template='seaborn')

Chart, scatter chart

Description automatically generated

# Plot MonthlyCharge vs Outage\_sec\_perweek  
px.scatter(df.sample(500, random\_state=42),  
 x='MonthlyCharge',  
 y='Outage\_sec\_perweek',  
 width=700,  
 template='seaborn')

Chart, scatter chart

Description automatically generated

# Plot Tenure vs Bandwidth\_GB\_Year  
px.scatter(df.sample(500, random\_state=42),  
 x='Bandwidth\_GB\_Year',  
 y='Tenure',  
 width=700,  
 template='seaborn')

Chart, scatter chart

Description automatically generated

# Convert dtypes for categorical features  
df = df.apply(lambda x: x.astype('category') if x.dtype == 'object' else x)  
df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 10000 entries, 0 to 9999  
Data columns (total 27 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Children 10000 non-null int64   
 1 Age 10000 non-null int64   
 2 Income 10000 non-null float64   
 3 Marital 10000 non-null category  
 4 Gender 10000 non-null category  
 5 Outage\_sec\_perweek 10000 non-null float64   
 6 Email 10000 non-null int64   
 7 Contacts 10000 non-null int64   
 8 Yearly\_equip\_failure 10000 non-null int64   
 9 Techie 10000 non-null category  
 10 Contract 10000 non-null category  
 11 Port\_modem 10000 non-null category  
 12 Tablet 10000 non-null category  
 13 InternetService 10000 non-null category  
 14 Phone 10000 non-null category  
 15 Multiple 10000 non-null category  
 16 OnlineSecurity 10000 non-null category  
 17 OnlineBackup 10000 non-null category  
 18 DeviceProtection 10000 non-null category  
 19 TechSupport 10000 non-null category  
 20 StreamingTV 10000 non-null category  
 21 StreamingMovies 10000 non-null category  
 22 PaperlessBilling 10000 non-null category  
 23 PaymentMethod 10000 non-null category  
 24 Tenure 10000 non-null float64   
 25 MonthlyCharge 10000 non-null float64   
 26 Bandwidth\_GB\_Year 10000 non-null float64   
dtypes: category(17), float64(5), int64(5)  
memory usage: 949.6 KB

# Assign X, y  
X = df.drop(['Tenure'], axis=1)  
y = df.Tenure  
  
# Instantiate ColumnTransformer for OneHotEncoding and fit  
ct = ColumnTransformer(  
 [('cat', OneHotEncoder(), X.columns[X.dtypes == 'category'].tolist())],  
 remainder='passthrough')  
X = ct.fit\_transform(X)  
  
# Split the dataset into training and test sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

### D3: Code Execution

#### Step 1 - Load in libraries and dataset

# Load in required packages  
import pandas as pd  
import numpy as np  
import plotly.express as px  
from sklearn.compose import ColumnTransformer  
from sklearn.preprocessing import OneHotEncoder  
from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error as MSE  
from yellowbrick.regressor import residuals\_plot  
from yellowbrick.model\_selection import feature\_importances  
from dtreeviz.trees import \*  
  
# Read in initial dataset and check .info()  
df = pd.read\_csv('./data/churn\_clean.csv')  
df.info()

<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 10000 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

#### Step 2 - Subset data & initial EDA

# Subset data by features to be included in model  
df = df.iloc[:, np.r\_[14:19, 20:42]]  
df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 10000 entries, 0 to 9999  
Data columns (total 27 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Children 10000 non-null int64   
 1 Age 10000 non-null int64   
 2 Income 10000 non-null float64  
 3 Marital 10000 non-null object   
 4 Gender 10000 non-null object   
 5 Outage\_sec\_perweek 10000 non-null float64  
 6 Email 10000 non-null int64   
 7 Contacts 10000 non-null int64   
 8 Yearly\_equip\_failure 10000 non-null int64   
 9 Techie 10000 non-null object   
 10 Contract 10000 non-null object   
 11 Port\_modem 10000 non-null object   
 12 Tablet 10000 non-null object   
 13 InternetService 10000 non-null object   
 14 Phone 10000 non-null object   
 15 Multiple 10000 non-null object   
 16 OnlineSecurity 10000 non-null object   
 17 OnlineBackup 10000 non-null object   
 18 DeviceProtection 10000 non-null object   
 19 TechSupport 10000 non-null object   
 20 StreamingTV 10000 non-null object   
 21 StreamingMovies 10000 non-null object   
 22 PaperlessBilling 10000 non-null object   
 23 PaymentMethod 10000 non-null object   
 24 Tenure 10000 non-null float64  
 25 MonthlyCharge 10000 non-null float64  
 26 Bandwidth\_GB\_Year 10000 non-null float64  
dtypes: float64(5), int64(5), object(17)  
memory usage: 2.1+ MB

# Show descriptive statistics for numeric data  
df.describe().T

count mean std min \  
Children 10000.0 2.087700 2.147200 0.000000   
Age 10000.0 53.078400 20.698882 18.000000   
Income 10000.0 39806.926771 28199.916702 348.670000   
Outage\_sec\_perweek 10000.0 10.001848 2.976019 0.099747   
Email 10000.0 12.016000 3.025898 1.000000   
Contacts 10000.0 0.994200 0.988466 0.000000   
Yearly\_equip\_failure 10000.0 0.398000 0.635953 0.000000   
Tenure 10000.0 34.526188 26.443063 1.000259   
MonthlyCharge 10000.0 172.624816 42.943094 79.978860   
Bandwidth\_GB\_Year 10000.0 3392.341550 2185.294852 155.506715   
  
 25% 50% 75% max   
Children 0.000000 1.000000 3.000000 10.000000   
Age 35.000000 53.000000 71.000000 89.000000   
Income 19224.717500 33170.605000 53246.170000 258900.700000   
Outage\_sec\_perweek 8.018214 10.018560 11.969485 21.207230   
Email 10.000000 12.000000 14.000000 23.000000   
Contacts 0.000000 1.000000 2.000000 7.000000   
Yearly\_equip\_failure0.000000 0.000000 1.000000 6.000000   
Tenure 7.917694 35.430507 61.479795 71.999280   
MonthlyCharge 139.979239 167.484700 200.734725 290.160419   
Bandwidth\_GB\_Year1236.470827 3279.536903 5586.141370 7158.981530

# Show correlation between continuous features  
dffloats = df[df.columns[df.dtypes == 'float64']]  
dffloats.corr()

Income Outage\_sec\_perweek Tenure MonthlyCharge \  
Income 1.000000 -0.010011 0.002114 -0.003014   
Outage\_sec\_perweek -0.010011 1.000000 0.002932 0.020496   
Tenure 0.002114 0.002932 1.000000 -0.003337   
MonthlyCharge -0.003014 0.020496 -0.003337 1.000000   
Bandwidth\_GB\_Year 0.003674 0.004176 0.991495 0.060406   
  
 Bandwidth\_GB\_Year   
Income 0.003674   
Outage\_sec\_perweek 0.004176   
Tenure 0.991495   
MonthlyCharge 0.060406   
Bandwidth\_GB\_Year 1.000000

# Subset numeric features and show histograms  
df\_num\_cols = [x for x in df.describe().columns]  
df\_num = df[df\_num\_cols]  
for i in df\_num.columns:  
 px.histogram(  
 df\_num.sample(500, random\_state=42),  
 x=i,  
 width=500,  
 height=300,  
 nbins=30,  
 template='seaborn'  
 ).show()

|  |  |
| --- | --- |
| Chart  Description automatically generated | Chart, histogram  Description automatically generated |
| Chart, histogram  Description automatically generated | Chart, histogram  Description automatically generated |
| Chart, histogram  Description automatically generated | Chart, bar chart  Description automatically generated |
| Chart  Description automatically generated | Chart, histogram  Description automatically generated |
| Chart, histogram  Description automatically generated | Chart, histogram  Description automatically generated |

# Plot Tenure vs Outage\_sec\_perweek  
px.scatter(df.sample(500, random\_state=42),  
 x='Tenure',  
 y='Outage\_sec\_perweek',  
 width=700,  
 template='seaborn')

Chart, scatter chart

Description automatically generated

# Plot MonthlyCharge vs Outage\_sec\_perweek  
px.scatter(df.sample(500, random\_state=42),  
 x='MonthlyCharge',  
 y='Outage\_sec\_perweek',  
 width=700,  
 template='seaborn')

Chart, scatter chart

Description automatically generated

# Plot Tenure vs Bandwidth\_GB\_Year  
px.scatter(df.sample(500, random\_state=42),  
 x='Bandwidth\_GB\_Year',  
 y='Tenure',  
 width=700,  
 template='seaborn')

Chart, scatter chart

Description automatically generated

#### Step 3 - Prepare subset data for analysis

# Convert dtypes for categorical features  
df = df.apply(lambda x: x.astype('category') if x.dtype == 'object' else x)  
df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 10000 entries, 0 to 9999  
Data columns (total 27 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Children 10000 non-null int64   
 1 Age 10000 non-null int64   
 2 Income 10000 non-null float64   
 3 Marital 10000 non-null category  
 4 Gender 10000 non-null category  
 5 Outage\_sec\_perweek 10000 non-null float64   
 6 Email 10000 non-null int64   
 7 Contacts 10000 non-null int64   
 8 Yearly\_equip\_failure 10000 non-null int64   
 9 Techie 10000 non-null category  
 10 Contract 10000 non-null category  
 11 Port\_modem 10000 non-null category  
 12 Tablet 10000 non-null category  
 13 InternetService 10000 non-null category  
 14 Phone 10000 non-null category  
 15 Multiple 10000 non-null category  
 16 OnlineSecurity 10000 non-null category  
 17 OnlineBackup 10000 non-null category  
 18 DeviceProtection 10000 non-null category  
 19 TechSupport 10000 non-null category  
 20 StreamingTV 10000 non-null category  
 21 StreamingMovies 10000 non-null category  
 22 PaperlessBilling 10000 non-null category  
 23 PaymentMethod 10000 non-null category  
 24 Tenure 10000 non-null float64   
 25 MonthlyCharge 10000 non-null float64   
 26 Bandwidth\_GB\_Year 10000 non-null float64   
dtypes: category(17), float64(5), int64(5)  
memory usage: 949.6 KB

# Assign X, y  
X = df.drop(['Tenure'], axis=1)  
y = df.Tenure  
  
# Instantiate ColumnTransformer for OneHotEncoding and fit  
ct = ColumnTransformer(  
 [('cat', OneHotEncoder(), X.columns[X.dtypes == 'category'].tolist())],  
 remainder='passthrough')  
X = ct.fit\_transform(X)  
  
# Split the dataset into training and test sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

#### Step 4 - Initial model fit and evaluation

# Original model  
# Create grid for RandomizedSearchCV  
n\_estimators = np.arange(500, 2000, 250)  
max\_depth = np.arange(5, 40, 5)  
min\_samp = np.arange(.01, .2, .04)  
  
# Declare grid dictionary  
rs\_grid = {  
 'n\_estimators': n\_estimators,  
 'max\_depth': max\_depth,  
 'min\_samples\_leaf': min\_samp,  
}  
  
# Instantiate base estimator  
rf = RandomForestRegressor(random\_state=42,  
 n\_jobs=-1,  
 bootstrap=True,  
 max\_features='sqrt')  
  
# Instantiate cross-val  
rscv\_rf = RandomizedSearchCV(estimator=rf,  
 param\_distributions=rs\_grid,  
 n\_iter=200,  
 scoring='neg\_mean\_squared\_error',  
 n\_jobs=-1,  
 cv=5,  
 verbose=10,  
 random\_state=42,  
 return\_train\_score=True)  
  
# Fit model  
rscv\_rf.fit(X\_train, y\_train)

Fitting 5 folds for each of 200 candidates, totalling 1000 fits

RandomizedSearchCV(cv=5,  
 estimator=RandomForestRegressor(max\_features='sqrt',  
 n\_jobs=-1, random\_state=42),  
 n\_iter=200, n\_jobs=-1,  
 param\_distributions={'max\_depth': array([ 5, 10, 15, 20, 25, 30, 35]),  
 'min\_samples\_leaf': array([0.01, 0.05, 0.09, 0.13, 0.17]),  
 'n\_estimators': array([ 500, 750, 1000, 1250, 1500, 1750])},  
 random\_state=42, return\_train\_score=True,  
 scoring='neg\_mean\_squared\_error', verbose=10)

# Create dataframe of top 1/2 estimators sorted by score  
dfdict = {}  
for i, x in enumerate(pd.DataFrame(rscv\_rf.cv\_results\_).sort\_values('rank\_test\_score').iloc[:100].params):  
 dfdict[i] = x  
cvdf = pd.DataFrame(dfdict).T.describe()  
cvdf.n\_estimators = cvdf.n\_estimators.astype('int')  
cvdf.min\_samples\_leaf = cvdf.min\_samples\_leaf.round(2)  
cvdf.max\_depth = cvdf.max\_depth.astype('int')  
  
# Declare best parameter values for below  
n\_est = rscv\_rf.best\_params\_['n\_estimators']  
ms = rscv\_rf.best\_params\_['min\_samples\_leaf']  
md = rscv\_rf.best\_params\_['max\_depth']  
  
# Get best estimator and score accuracy of training and test sets  
bestrf = rscv\_rf.best\_estimator\_  
trainscore = bestrf.score(X\_train, y\_train)  
testscore = bestrf.score(X\_test, y\_test)  
  
# Create predictions from test and evaluate RMSE  
y\_pred = bestrf.predict(X\_test)  
rmse\_rf = MSE(y\_test, y\_pred) \*\* 0.5  
  
# Print model results  
print(f'''---Model Results---  
  
Cross Validation Parameter Results:  
----------------------------------  
{cvdf}  
  
Best Estimator Params:  
----------------------  
Best n\_estimators: {n\_est}  
Best min\_samples\_leaf: {ms}  
Best max\_depth: {md}  
  
Best Estimator Scoring:  
----------------------  
Training score: {trainscore.round(4)}  
Testing score: {testscore.round(4)}  
RMSE: {rmse\_rf.round(2)}  
''')

---Model Results---  
  
Cross Validation Parameter Results:  
----------------------------------  
 n\_estimators min\_samples\_leaf max\_depth  
count 100 100.00 100  
mean 1085 0.04 19  
std 417 0.03 10  
min 500 0.01 5  
25% 750 0.01 10  
50% 1000 0.05 20  
75% 1500 0.05 30  
max 1750 0.09 35  
  
Best Estimator Params:  
----------------------  
Best n\_estimators: 500  
Best min\_samples\_leaf: 0.01  
Best max\_depth: 15  
  
Best Estimator Scoring:  
----------------------  
Training score: 0.8238  
Testing score: 0.8159  
RMSE: 11.32

#### Step 5 - Revised model fit and evaluation

# Revised model  
# Create grid for RandomizedSearchCV  
n\_estimators = np.arange(400, 1400, 100)  
max\_depth = np.arange(10, 30, 2)  
  
# Declare grid dictionary  
rs\_grid = {  
 'n\_estimators': n\_estimators,  
 'max\_depth': max\_depth,  
}  
  
# Instantiate base estimator  
rf = RandomForestRegressor(random\_state=42,  
 n\_jobs=-1,  
 bootstrap=True,  
 max\_features='sqrt')  
  
# Instantiate cross-val  
rscv\_rf = RandomizedSearchCV(estimator=rf,  
 param\_distributions=rs\_grid,  
 n\_iter=100,  
 scoring='neg\_mean\_squared\_error',  
 n\_jobs=-1,  
 cv=5,  
 verbose=10,  
 random\_state=42,  
 return\_train\_score=True,  
 )   
  
# Fit model  
rscv\_rf.fit(X\_train, y\_train)

Fitting 5 folds for each of 100 candidates, totalling 500 fits

RandomizedSearchCV(cv=5,  
 estimator=RandomForestRegressor(max\_features='sqrt',  
 n\_jobs=-1, random\_state=42),  
 n\_iter=100, n\_jobs=-1,  
 param\_distributions={'max\_depth': array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28]),  
 'n\_estimators': array([ 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300])},  
 random\_state=42, return\_train\_score=True,  
 scoring='neg\_mean\_squared\_error', verbose=10)

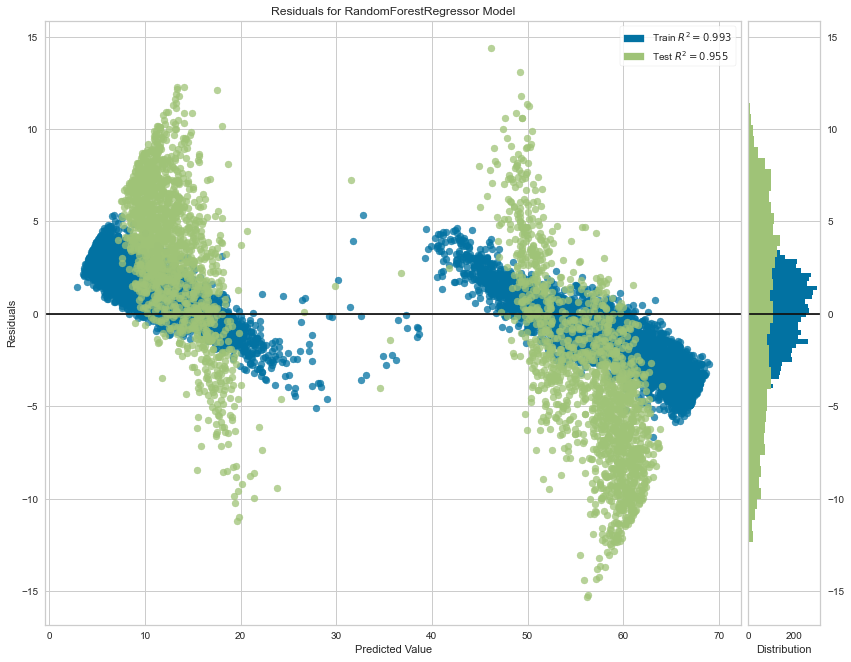
# Create dataframe of top 1/2 estimators sorted by score  
dfdict\_rev = {}  
for i, x in enumerate(pd.DataFrame(rscv\_rf.cv\_results\_).sort\_values('rank\_test\_score').iloc[:50].params):  
 dfdict\_rev[i] = x  
cvdf\_rev = pd.DataFrame(dfdict\_rev).T.describe()  
cvdf\_rev.n\_estimators = cvdf\_rev.n\_estimators.astype('int')  
cvdf\_rev.max\_depth = cvdf\_rev.max\_depth.astype('int')  
  
# Declare best parameter values for below  
n\_est\_rev = rscv\_rf.best\_params\_['n\_estimators']  
md\_rev = rscv\_rf.best\_params\_['max\_depth']  
bestrf\_rev = rscv\_rf.best\_estimator\_  
  
# Get best estimator and score accuracy of training and test sets  
trainscore\_rev = bestrf\_rev.score(X\_train, y\_train)  
testscore\_rev = bestrf\_rev.score(X\_test, y\_test)  
  
# Create predictions from test and evaluate RMSE  
y\_pred = bestrf\_rev.predict(X\_test)  
rmse\_rf\_rev = MSE(y\_test, y\_pred) \*\* 0.5  
  
# Get feature importances from best estimator  
featimp = bestrf\_rev.feature\_importances\_.tolist()  
  
# Print revised model results  
print(f'''---Model Results---  
  
Cross Validation Parameter Results:  
----------------------------------  
{cvdf\_rev}  
  
Best Estimator Params:  
----------------------  
Best n\_estimators: {n\_est\_rev}  
Best max\_depth: {md\_rev}  
  
Best Estimator Scoring:  
----------------------  
Training score: {trainscore\_rev.round(4)}  
Testing score: {testscore\_rev.round(4)}  
RMSE: {rmse\_rf\_rev.round(2)}  
''')

---Model Results---  
  
Cross Validation Parameter Results:  
----------------------------------  
 n\_estimators max\_depth  
count 50 50  
mean 850 24  
std 290 2  
min 400 20  
25% 600 22  
50% 850 24  
75% 1100 26  
max 1300 28  
  
Best Estimator Params:  
----------------------  
Best n\_estimators: 400  
Best max\_depth: 20  
  
Best Estimator Scoring:  
----------------------  
Training score: 0.9931  
Testing score: 0.9551  
RMSE: 5.59

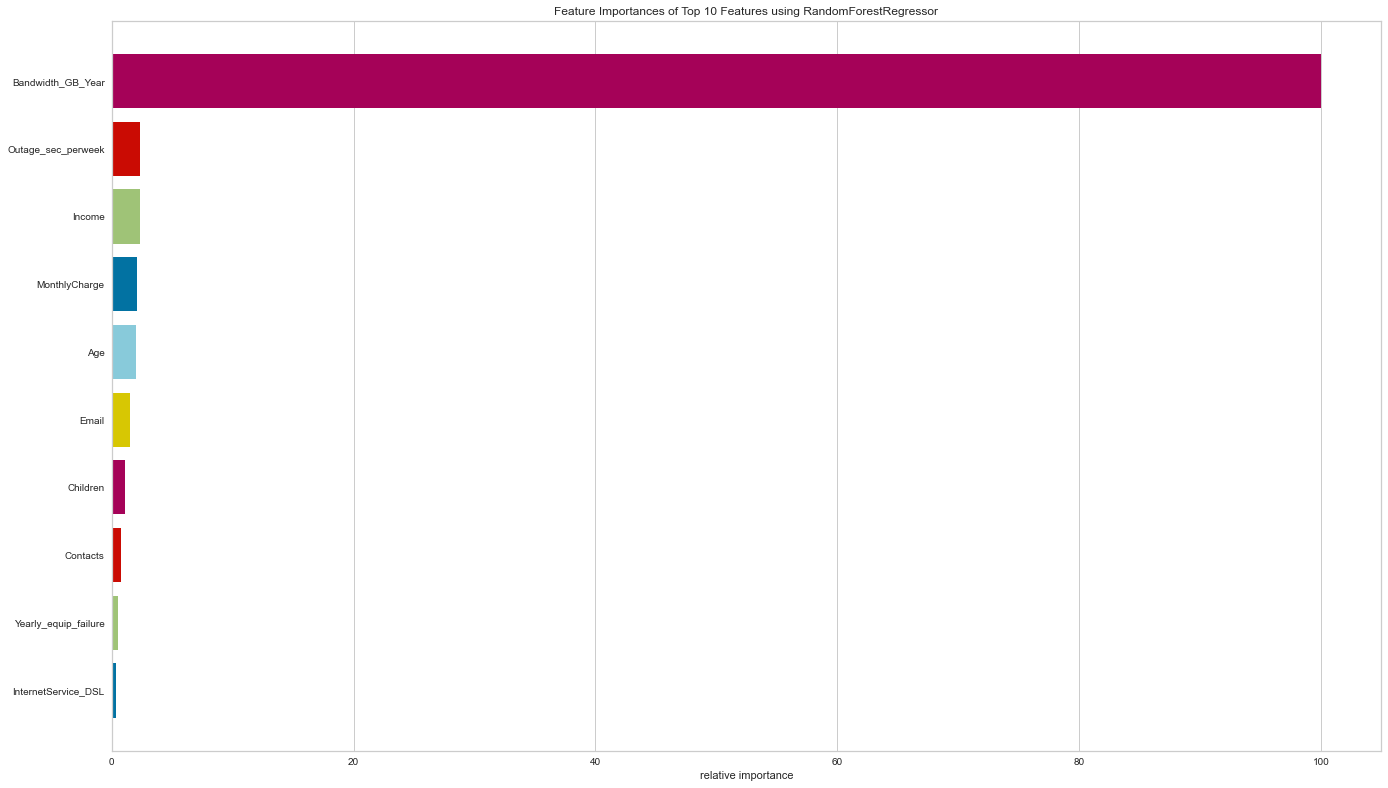
#### 

#### Step 6 - Model visualizations

# PLot residuals for training and test sets  
resplt = residuals\_plot(bestrf\_rev,  
 X\_train,  
 y\_train,  
 X\_test,  
 y\_test,  
 size=(1000,800))



# Replace category name prefix with actual category name  
catnames = df.dtypes[df.dtypes == 'category'].index.tolist()  
ohecats = ct.get\_feature\_names()  
for i, x in enumerate(ohecats):  
 if x[:3] == 'cat':  
 n = int(x[x.find('x')+1:x.rfind('\_')])  
 ohecats[i] = x.replace(x[:x.rfind('\_')], catnames[n])  
  
# Plot top 10 feature importances from best estimator sorted by highest importance  
fi = feature\_importances(bestrf\_rev, X\_train, X\_test, size=(1400,800), labels=ohecats, topn=10, relative=True, dpi=500)  
fi.show()



<AxesSubplot:title={'center':'Feature Importances of Top 10 Features using RandomForestRegressor'}, xlabel='relative importance'>

# Plot example regression decision tree at a depth of 5  
# to illustrate underlying estimator  
dtv = dtreeviz(bestrf\_rev.estimators\_[0],  
 x\_data=X\_train,  
 y\_data=y\_train,  
 target\_name='Tenure',  
 feature\_names=ohecats,  
 title='Decision Tree Regressor Visualization',  
 depth\_range\_to\_display=(0,5),  
 orientation='LR'  
 )  
dtv

*Please see attached ‘dtreeviz.png’ for full size image*

Diagram

Description automatically generated with low confidence

## Part V - Data Summary & Implications

### E1: Accuracy & MSE

The initial model performed quite well. The .score() method the the object defaults to the metric or the coefficient of determination. With a maximum of 1.0, the score for the training set of 0.8238 and 0.8159 on the test set represent a fairly high accuracy for the model. Additionally, the root mean squared error (RMSE) was determined on the test set and resulted in 11.32. In other words, the average error in terms of Tenure (which is denominated in months) is 11.32 months. Considering the fact that the standard deviation of the Tenure feature, as we observed in our EDA process, is approximately 26 months, there is room for improvement, but the model appears to do a decent job. After further hyperparameter tuning, the final model is incredibly accurate with an of 0.9931 on the training set and 0.9551 on the test set. This is an impressive score, particularly considering the cross-validation steps as well as the fact that the score on the training set is so high. Finally, the RMSE for the final model based on the training set is 5.59, or in other words, the average error for the final model is 5.59 months. Based on the results shown here, both models, but the revised model in particular is exceptionally accurate.

### E2: Results & Implications

As mentioned above, our model performed incredibly well in predicting a given client's tenure based on the predictor features identified in C2. As a result, our analysis supports the rejection of the null hypothesis () which states:

There are no reliable predictors of customer retention (Tenure) that can be observed via a random forest regression model of the available data.

Furthermore, our analysis supports the acceptance of the alternate hypothesis () which states:

Customer retention (Tenure) can be predicticted with reasonable reliability via a random forest regression model of the available data.

Therefore, our initial research question (Can these data be used to reliably predict customer retention (Tenure) via a random forest regression model and, if so, which features are most important?) is answered in the affirmative based on the results of our above analysis.

The random forest regression model has done a remarkable job of predicting Tenure. With regard to implications, while the random forest algorithm is considered a "black box" algorithm, we can at least measure the feature importances of our best estimator which gives us a measure of insight into what features are most important in predicting Tenure. When this was measured and visualized above, we note that the feature Bandwidth\_GB\_Year, which is simply the average data in GB a customer uses in a year, was the most important feature by an incredibly large margin. In our EDA, we also noted that the correlation between Tenure and Bandwidth\_GB\_Year was extremely high. The in-depth exploration of the relationship between the two features lies outside of the scope of this analysis, however, it seems prudent to do so as a follow-up. Regardless, the top 5 most important features in predicting Tenure according to this model and in order of importance are:

* Bandwidth\_GB\_Year
* Income
* Outage\_sec\_perweek
* MonthlyCharge
* Age

This provides insight into further analysis that should be done in understanding what factors contribute to a customer's tenure and how the average tenure of a customer can be lengthened.

### E3: Limitations

One limitation of the analysis, which was briefly alluded to previously, is the limits of random forest regression with regard to the sampling data. To reiterate, the random forest regression algorithm predicts outcomes based on mean values accross decision trees using randomized feature selection. The decision tree splits are determined by minimizing the cost function mean squared error with each leaf node (the terminal node of the decision tree) averaging the target feature values of the samples split therein. This being the case, the algorithm, while enormously effective and versatile, lacks extrapolatory power. (Thompson, 2019) Because the model is, of necessity, range-bound to the sample data, it is unable to make predictions outside of that range. This issue becomes particularly limiting with a heteroskedastic sample set. However, for the purposes of this analysis, the random forest regression is an appropriate and effective tool.

### E4: Course of Action

As a result of the analysis performed, it is recommended that the organization further explore the relationship between Tenure and Bandwidth\_GB\_Year. Because, according to this analysis, the bandwidth feature has an overstated importance in comparison to the other features measured, it seems probable that further analysis and understanding of that relationship could reveal key findings that relate to the prediction (and perhaps actions or attributes that could contribute to lengthening) of a customer's Tenure. Additionally, the other important features mentioned above (Income, Outage\_sec\_perweek, MonthlyCharge, and Age) should also be examined further as well as any relationship between them to help uncover additional insights into predicting customer retention.

### H: Sources

Géron, A. (2019). *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems.* Beijing: O'Reilly.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, É. (2011, February 1). Scikit-learn: Machine Learning in Python. *The Journal of Machine Learning Research, 12*, 2825–2830. doi:10.5555/1953048.2078195

Thompson, B. (2019, December 17). *A limitation of Random Forest Regression*. Retrieved May 2022, from Towards Data Science: https://towardsdatascience.com/a-limitation-of-random-forest-regression-db8ed7419e9f