Project Purpose

This project uses the data that was previously cleaned in the Jupyter Notebook "Data Cleaning". Telecommunications organizations often suffer from a loss of revenue due to customers choosing to terminate their services. As new companies enter the market, many customers choose to leave existing contracts for cheaper services. If there were a way to identify customers who may decide to cancel their account, a company might be able to intercede with special offers and services. In this analysis, I hope to find an answer to the question: "Given a list of customer attributes, can we determine which customers might terminate their services?"

In my last project, I performed a similar analysis using the k-Nearest Neighbors (KNN) algorithm and the "Churn" target variable. I am choosing to investigate the same question, but this time using the Random Forest algorithm and the "Tenure" variable.

My major goal in this analysis is to determine if I should have more confidence in a model derived using KNN or the Random Forest method. As I enter the field of data science, I want to have a good understanding of the tools that I will be using. By investigating the same question using different methods, I will ensure that the organization has the most accurate depiction of which customers may terminate their services.

Explanation of Prediction Method

For this analysis, I am using the Random Forest (RF) algorithm. RF is considered an ensemble method of machine learning, in that it combines multiple base models to create an optimal model using a decision tree as the base estimator. A decision tree for regression analysis is created as the algorithm cycles through the dataset by looking at each feature as if going through a series of if-else structures on a flow chart. It looks at the criteria for a feature, then puts it into one of two categories (or branches) based on whether it fits. It then proceeds through each feature until it reaches the end (leaf), creating branches as it goes. The instance is then labeled as the predominant class (Kawerk, n.d.).

Random Forest is performed by first performing bootstrap sampling on the data. Decision trees are created from multiple bootstrap samples that are different from the training data set. The final decision for the prediction is based on the predominant number of total predictions upon completion (Kawerk, n.d.). In the case of this analysis, the result will be a model that can be used in determining whether a customer might be of terminating their services in the future by applying it to data for active customers.

Assumption

One assumption made of data using the Random Forest method is that there should be no missing values. Having data present allows for better prediction because it is not using estimations (Shruti, 2020).

Preprocessing

My goal in the preprocessing stage is to prepare the data for use by the Random Forest algorithm. All features passed in need to be numerical, so categorical variables will be transformed. Also, any variables that will not be used in the analysis portion will be removed from the data set.

Steps to Prepare the Data

The following steps will be taken to prepare the data for analysis:

1. Import data to pandas dataframe
2. Determine variable types and those which may require further investigation
3. Convert binary variables to yes = 1 and no = 0
4. Investigate potential categorical variables using bar charts, then convert categorical values to dummy variables
5. Drop columns that will not be used in classification
6. Create the arrays for feature and response variables

Step 1

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from yellowbrick.regressor import AlphaSelection, PredictionError, ResidualsPlot
import warnings
warnings.filterwarnings('ignore') # Ignore warning messages for readability
```

In [2]:
```
# Read in data set and view head
df = pd.read_csv('churn_clean.csv')
pd.options.display.max_columns = None
df.head()
```

Out[2]:

	CaseOrder	Customer_id	Interaction	UID	City	State	County	Zip	Lat	Lng	I
0	1	K409198	aa90260b-4141-4a24-8e36-b04ce1f4f77b	e885b299883d4f9fb18e39c75155d990	Point Baker	AK	Prince of Wales-Hyder	99927	56.25100	-133.37571	
1	2	S120509	fb76459f-c047-4a9d-8af9-e0f7d4ac2524	f2de8bef964785f41a2959829830fb8a	West Branch	MI	Ogemaw	48661	44.32893	-84.24080	
2	3	K191035	344d114c-3736-4be5-98f7-c72c281e2d35	f1784cfa9f6d92ae816197eb175d3c71	Yamhill	OR	Yamhill	97148	45.35589	-123.24657	
3	4	D90850	abfa2b40-2d43-4994-b15a-989b8c79e311	dc8a365077241bb5cd5ccd305136b05e	Del Mar	CA	San Diego	92014	32.96687	-117.24798	
4	5	K662701	68a861fd-0d20-4e51-a587-8a90407ee574	aabb64a116e83fdc4befc1fbab1663f9	Needville	TX	Fort Bend	77461	29.38012	-95.80673	

Step 2

In [3]:
```
# View column names
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
```

The target variable, Churn, is categorical.

The continuous variables that will be used in the analysis are:

> Lat, Lng, Timezone, Income, Outage_sec_perweek, Tenure, MonthlyCharge, Bandwidth_GB_Year, Zip, Population, Children, Age, Email, Contacts, Yearly_equip_failure

The categorical variables that will be used in the analysis are:

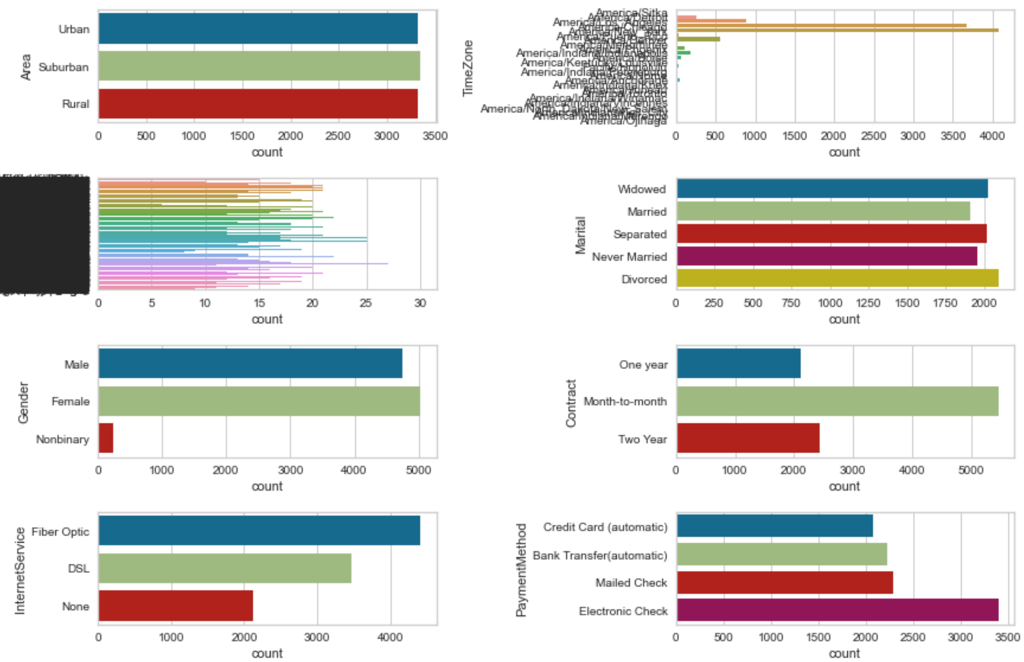
> Area, Gender, Marital, Contract, InternetService, PaymentMethod, Techie, Port_modem, Tablet, Phone, Multiple, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies, PaperlessBilling, Item1, Item2, Item3, Item4, Item5, Item6, Item7, Item8

Step 3

In [4]:
```
# Convert binary variables into yes = 1, no = 0
cols = ['Churn', 'Techie', 'Port_modem', 'Tablet', 'Phone', 'Multiple', 'OnlineSecurity', 'OnlineBackup', 'Dev
iceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'PaperlessBilling']
df[cols] = df[cols].replace(to_replace = ['No', 'Yes'], value = [0, 1])
```

Step 4

In [5]:
```
# View bar charts for potential categorical variables to determine number of categories
figure, axes = plt.subplots(nrows=4, ncols=2, figsize=(15,8))
plt.subplot(4, 2, 1)
sns.countplot(data = df, y = 'Area')
plt.subplot(4, 2, 2)
sns.countplot(data = df, y = 'TimeZone')
plt.subplot(4, 2, 3)
sns.countplot(data = df, y = 'Job')
plt.subplot(4, 2, 4)
sns.countplot(data = df, y = 'Marital')
plt.subplot(4, 2, 5)
sns.countplot(data = df, y = 'Gender')
plt.subplot(4, 2, 6)
sns.countplot(data = df, y = 'Contract')
plt.subplot(4, 2, 7)
sns.countplot(data = df, y = 'InternetService')
plt.subplot(4, 2, 8)
sns.countplot(data = df, y = 'PaymentMethod')
figure.tight_layout()
plt.show();
```



- Timezone and Job seem to have too many possible categories for meaningful separation into categories, so they will be treated as string variables.

In [6]:
```
# Create separate variables for each categorical value, with a 1 if the value is present in that row and 0 if
 not present
df = pd.get_dummies(data=df, columns=['Area', 'Marital', 'Gender', 'Contract', 'InternetService', 'PaymentMeth
od'])
```

<u>Step 5</u>

In [7]:
```
# Drop columns not needed for analysis
drops = ['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State', 'County', 'TimeZone', 'Job']
df = df.drop(drops, axis = 1)
```

In [8]:
```
# View head of clean data set
df.head()
```

Out[8]:

	Zip	Lat	Lng	Population	Children	Age	Income	Churn	Outage_sec_perweek	Email	Contacts	Yearly_equip_failure
0	99927	56.25100	-133.37571	38	0	68	28561.99	0	7.978323	10	0	1
1	48661	44.32893	-84.24080	10446	1	27	21704.77	1	11.699080	12	0	1
2	97148	45.35589	-123.24657	3735	4	50	9609.57	0	10.752800	9	0	1
3	92014	32.96687	-117.24798	13863	1	48	18925.23	0	14.913540	15	2	0
4	77461	29.38012	-95.80673	11352	0	83	40074.19	1	8.147417	16	2	1

Step 6

In [9]:
```
# Create arrays for the features and the response variable
y = df['Tenure']
X = df.drop('Tenure', axis=1)
```

Prepared Data

In [10]:
```
# Save cleaned dataframe to CSV
df.to_csv('churn_clean_data_final.csv', index = False, encoding = 'utf-8')
```

Splitting the Data

In [11]:
```
# Split into training and test set (ref 2)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)
```

Regression Method

To perform the Random Forest analysis, I will first instantiate the random forest classifier algorithm, and fit it to the training data. I will then look at what features are most important to the model, and remove any that fall below 0.1 to decrease the complexity of the model. I will then perform hyperparameter tuning to find the best parameters to create the optimal model for the data.

In [12]:
```
# Instantiate rf (ref 1)
rf = RandomForestRegressor(n_estimators=25, random_state=2)

# Fit rf to the training set
rf.fit(X_train, y_train)
```

Out[12]:
```
RandomForestRegressor(n_estimators=25, random_state=2)
```

In [13]:
```
# View pre-optimization accuracy
print(rf.score(X_test,y_test))
```

```
0.9980818697795567
```

In [14]:
```
# View pre-optimization MSE

# Predict the test set labels (ref 1)
y_pred = rf.predict(X_test)

# Evaluate the test set RMSE
mse_test = MSE(y_test,y_pred)

# Print rmse_test
print('Test set MSE of rf: {:.2f}'.format(mse_test))
```

```
Test set MSE of rf: 1.34
```

In [15]:
```
# Create a pd.Series of features importances (ref 2)
importances = pd.Series(data=rf.feature_importances_,
                        index= X_train.columns)

# Sort importances
importances_sorted = importances.sort_values()

# Draw a horizontal barplot of importances_sorted
plt.figure(figsize=(20,20))
importances_sorted.plot(kind='barh', color='lightgreen')
plt.title('Features Importances')
plt.show()
```

In [16]:
```
# View a ranked list of features
importances_sorted
```

Out[16]:
```
Gender_Nonbinary                         0.000001
Phone                                    0.000007
Marital_Married                          0.000009
Contract_One year                        0.000010
Contract_Two Year                        0.000010
Marital_Divorced                         0.000010
Marital_Never Married                    0.000011
Techie                                   0.000011
Area_Rural                               0.000011
Area_Suburban                            0.000011
Marital_Separated                        0.000011
Contract_Month-to-month                  0.000012
PaymentMethod_Credit Card (automatic)    0.000012
Area_Urban                               0.000012
Marital_Widowed                          0.000012
PaymentMethod_Bank Transfer(automatic)   0.000013
PaymentMethod_Mailed Check               0.000013
Tablet                                   0.000015
PaperlessBilling                         0.000015
Multiple                                 0.000016
Port_modem                               0.000016
PaymentMethod_Electronic Check           0.000017
TechSupport                              0.000018
OnlineBackup                             0.000019
Yearly_equip_failure                     0.000020
Gender_Female                            0.000028
Item2                                    0.000029
InternetService_None                     0.000030
DeviceProtection                         0.000030
Churn                                    0.000031
InternetService_Fiber Optic              0.000031
Item3                                    0.000032
Item1                                    0.000032
Contacts                                 0.000033
Item4                                    0.000033
Item8                                    0.000033
StreamingMovies                          0.000033
Item7                                    0.000034
Item6                                    0.000034
Item5                                    0.000037
OnlineSecurity                           0.000039
Gender_Male                              0.000043
Email                                    0.000069
Zip                                      0.000074
Lng                                      0.000079
Population                               0.000102
Lat                                      0.000102
Income                                   0.000112
Outage_sec_perweek                       0.000114
StreamingTV                              0.000162
Children                                 0.000480
Age                                      0.000741
MonthlyCharge                            0.003989
InternetService_DSL                      0.004624
Bandwidth_GB_Year                        0.988478
dtype: float64
```

In [17]:
```
# Hyperparameter tuning (ref 1)

# Define the dictionary 'params_rf'
params_rf = {'n_estimators':[100,350,500], 'max_features': ['log2','auto','sqrt'],'min_samples_leaf':[2,10,30
]}

# Instantiate grid_rf
grid_rf = GridSearchCV(estimator=rf, param_grid=params_rf, scoring='neg_mean_squared_error', cv=3, verbose=1,
n_jobs=-1)
```

In [18]:
```
# Fit gridsearch results to training data
grid_rf.fit(X_train, y_train)

# Extract the best estimator for optimal model
optimal_rf = grid_rf.best_estimator_
optimal_rf
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done  81 out of  81 | elapsed:  4.6min finished
```

Out[18]:
```
RandomForestRegressor(min_samples_leaf=2, n_estimators=500, random_state=2)
```

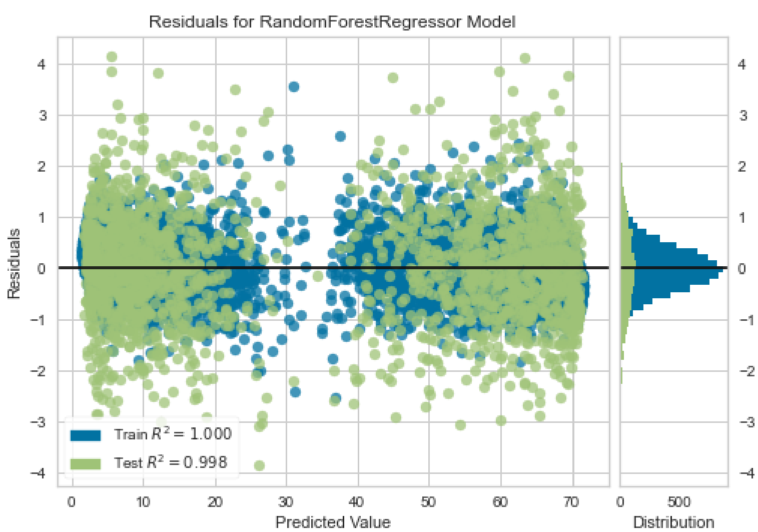
In [19]:
```
# View post-optimization accuracy
print(optimal_rf.score(X_test,y_test))
```

```
0.9982277893451931
```

In [20]:
```
# View post-optimization MSE

# Predict the test set labels (ref 1)
y_pred = optimal_rf.predict(X_test)

# Evaluate the test set RMSE
mse_test = MSE(y_test,y_pred)

# Print rmse_test
print('Test set MSE of rf: {:.2f}'.format(mse_test))
```

```
Test set MSE of rf: 1.23
```

In [21]:
```
# Display residuals plot (ref 4)
visualizer = ResidualsPlot(optimal_rf)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
g = visualizer.poof()
```



Results

The accuracy of the model is very high at nearly 100%. This means of all the samples in the testing set, 99.8% predicted the correct value for tenure. There appeared to be only a slight increase in accuracy after performing hyperparameter tuning.

The mean squared error (MSE) for the model is 1.23, which was slightly lower after optimization. The MSE metric computes the mean of the sum of the squares of the residuals. A low MSE means that most target values are very close to the prediction. The achieved score for the optimized model using the test data is very low which, gives me a high degree of confidence in the model's ability to predict the length of customer tenure.

It appears that the Random Forest model does an excellent job of predicting the length of customer tenure. This means that if we were to use this model with new customer data in the future, we would be likely to predict the duration that a customer will remain with the telecommunications company. To accomplish these predictions, new data will be compared to the existing data and taken back through the Decision Tree process via Random Forest.

The company could then generate a list of customers who may be nearing the end of their tenure to use in focusing customer interaction. By having the ability to predict which customers are in danger of terminating their services, the company may be able to intercede with targeted offers and discounts. This would ensure a better Return on Investment compared to offering these types of benefits to their entire customer base. Resources could be periodically focused in areas where revenue may be lost if a customer decides to end their contract to potentially gain customer loyalty.

Sources

- Kawerk, E. (n.d.). Machine Learning with Tree-Based Models in Python. Retrieved January 25, 2021, from https://learn.datacamp.com/courses/machine-learning-with-tree-based-models-in-python (https://learn.datacamp.com/courses/machine-learning-with-tree-based-models-in-python)
- Shruti, M. (2020, June 24). Introduction to Random Forest in R. Retrieved January 26, 2021, from https://www.simplilearn.com/tutorials/data-science-tutorial/random-forest-in-r (https://www.simplilearn.com/tutorials/data-science-tutorial/random-forest-in-r)

Helpful Sites Used in Coding Project

1. Much of the code for section D comes from the course "Machine Learning with Tree-Based Models in Python" which was included in the D209 official study material.
2. https://www.datacamp.com/community/tutorials/random-forests-classifier-python (https://www.datacamp.com/community/tutorials/random-forests-classifier-python)
3. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)
4. https://www.kaggle.com/kautumn06/yellowbrick-regression-visualizer-examples (https://www.kaggle.com/kautumn06/yellowbrick-regression-visualizer-examples)