PERFORMANCE ASSESSMENT

Task 2 | Predictive Analysis

KAILI HAMILTON

Masters of Science in Data Analytics, Western Governors University

Course: D209 Data Mining I

Instructor: Dr. Festus Elleh

Program Mentor: Krissy Bryant

December, 2023

TABLE OF CONTENTS

A1

My research question for this performance assessment is, “Using the Random Forest Regressor (RF) prediction method, can we predict the tenure of a customer?”

A2

One goal of the data analysis is to build a supervised machine learning model using random forests to predict the tenure of its customers and identify key features that influence customer tenure so that the company can take measures to maintain their customers longer.

B1

I used random forests (RF) as my predictive method I used in my analysis. “Random Forests (RF) are an ensemble machine learning method for classification and regression. My analysis a regression problem because the target variable, tenure, is a numerical data type. RF constructs multiple decision trees at the training time and outputs the mean of the predictions for regression. Each of the trees makes its own individual prediction. These predictions are then averaged to produce a single result. The averaging makes an RF better than a single decision tree, improving its accuracy and reducing overfitting. A prediction from the RF Regressor is an average of the trees’ predictions in the forest” (Elleh, nd). Hyperparameter tuning is performed using GridSearchCV and the best features for prediction will be selected via SelectKBest.

Expected outcomes include building an algorithm that will predict a customer’s tenure based on selected key features and measuring the accuracy of the algorithm using MSE, mean square error.

B2

One assumption of Random Forests (RF) is that RF uses bootstrap aggregation, where samples are taken randomly with replacement, which assumes that sampling is representative of the whole data set. “[This] is also an assumption of the decision tree on which the random forest is built. The tree-based model depends heavily on the training dataset used to build the model” (Elleh, nd). In other words, diverse subsets of data help to reduce overfitting and improve generalization.

B3

Listed below are the packages and libraries I used in my analysis along with their justification (Elleh, nd).

Libraries and Packages	**Justification**
Pandas	Import data into data frames and data manipulation
NumPy	Provides array objects for calculation
Seaborn	For visualizations
Matplotlib	For visualizations
from sklearn.impute import SimpleImputer	Impute missing data
from sklearn.feature_selection import SelectKBest, f_classif	Select k best features
from sklearn.preprocessing import scale	Scale the features
from sklearn.preprocessing import StandardScaler	Scale the features
From sklearn.model_selection import train_test_split	Split the data into training and testing sets
From sklearn.ensemble import RandomForestRegressor	Random forest regression fitting a number of decision trees
From sklearn.model_selection import cross_val_score	Cross validation score
From sklearn.model_selection import GridSearchCV	Grid search cross validation to find optimal k
From sklearn.pipeline import Pipeline	Using a pipeline to scale the data and fit the model
From sklearn.metrics import make_scorer	Scoring function to use in GridSearchCV
From sklearn.metrics import accuracy_score	Provides an accuracy score for the model
From sklearn.metrics import mean_squared_error as MSE	Calculate the MSE metric
From sklearn.metrics import r2_score	Calculate R^2

C1

One data preprocessing goal relevant to the prediction method from part A1 is as follows. Prepare the data to be ready for Random Forest Regression by: imputing missing data, check for outliers, encoding categorical variables, looking for correlation in the data, scaling the numerical data types by standardizing, and selecting features using selectKbest to improve model accuracy (Elleh, nd).

C2

Here I identify the initial set of variables that I used to perform the analysis for the question for part A1, as well as their types. “Tenure” is the target variable. All others are feature variables.

Categorical Variables	Numeric Variables
Martial_Widowed	Tenure
Churn	Bandwidth_GB_Year
InternetService_DSL	Population
Port_modem	

C3

The steps I took to prepare the data for analysis, including the code segment used for each step, are outlined below (Elleh, nd).

1. **Load and view data**. After importing some preliminary libraries and packages, I read in the data as a data frame and viewed the head.

```
import pandas as pd #dataframes
import numpy as np #arrays
import seaborn as sns #visualization
import matplotlib as plt #visualization
```

```
from sklearn.impute import SimpleImputer #impute missing values
from sklearn.feature_selection import SelectKBest, f_classif #Feature selection
```

```
df= pd.read_csv('churn_clean.csv')
df.head()
```

2. **Evaluate the data structures and data types.** I did some exploratory data analysis by looking at the data structures, such as the shape and summary statistics and value counts on the varaibles. I also examined the data types. I also removed extraneous columns such as “CaseOrder”, “CustomerID”, “Job”, and “Items 1-8” as well as other demographic information.

```
#drop columns: customer IDs and demographics
df.drop(columns=['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State', 'County', 'Zip', 'Lat', 'Lng', 'Area', 'TimeZone'], inplace=True)
df.head()
```

```
#drop other columns

df.drop(columns=['Email', 'Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6', 'Item7', 'Item8'], inplace=True)
df.head()
df.shape
```

```
(10000, 29)
```

```
df.dtypes
```

```
df.describe()
```

```
df.describe(include=object)
```

```
# drop "Job" variable - too many unique observations to encode with dummy variables
df.drop(columns='Job', inplace=True)
df.head()
```

```
df['PaymentMethod'].value_counts()
```

```
df['InternetService'].unique()
```

```
array(['Fiber Optic', 'DSL', nan], dtype=object)
```

3. **Remove null values, missing data, outliers**. Next, I explored the data for outliers and removed the most extreme of those. When I null and/or missing values I discovered that there was NaN values in the "InternetService" variable. The options for internet service are "Fiber Optic", "DSL", and "none". The null values here represented customers who did not subscribe to "Fiber Optic" nor "DSL." Instead of imputing the missing values here, I took care of these when creating dummy variables, since customers who have "None" will be inherently encoded in the dummies.

```
outliers = find_outliers_IQR(df['Income'])
print("number of outliers: " + str(len(outliers)))
print('max outlier value: ' + str(outliers.max()))
print('min outlier value: ' + str(outliers.min()))
```

```
number of outliers: 336
max outlier value: 258900.7
min outlier value: 104362.5
```

```
outliers = find_outliers_IQR(df['Outage_sec_perweek'])
print("number of outliers: " + str(len(outliers)))
print('max outlier value: ' + str(outliers.max()))
print('min outlier value: ' + str(outliers.min()))
```

```
number of outliers: 76
max outlier value: 21.20723
min outlier value: 0.09974694
```

```
df.drop(df[df['Income'] > 200000].index, inplace=True)

df.drop(df[df['Population'] > 100000].index, inplace=True)
```

```
df.isnull().sum() #missing values
```

```
df['InternetService'].value_counts() #2129 missing values from "InternetService"
```

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

4. **Covert categorical variables into dummy variables.** I created dummy variables for nominal encoding. For ordinal encoding I re-expressed “No” as “0” and “Yes” as “1”. I include code samples for the nominal categorical variable, “InternetService” and for the ordinal categorical variables, such as “Churn”.

```
#get dummy variables on "InternetService"
df = pd.get_dummies(df, columns=['InternetService'])
df.head()
```

```
#remove the space in column title
df.rename(columns={'InternetService_Fiber Optic' : 'InternetService_FiberOptic'}, inplace=True)
df.head()
```

```
df[['InternetService_FiberOptic', 'InternetService_DSL']].dtypes
```

```
InternetService_FiberOptic    bool
InternetService_DSL           bool
dtype: object
```

```
df['InternetService_DSL'].replace({
    False : 0,
    True : 1
}, inplace=True)
```

```
df['InternetService_FiberOptic'].replace({
    False : 0,
    True : 1
}, inplace=True)
```

```
df[['InternetService_DSL', 'InternetService_FiberOptic']].dtypes
```

```
df['Churn'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['Techie'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['Port_modem'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['Tablet'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['Phone'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['Multiple'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['OnlineSecurity'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['OnlineBackup'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['DeviceProtection'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['TechSupport'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['StreamingTV'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['StreamingMovies'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['PaperlessBilling'].replace({'No' : 0, 'Yes' : 1}, inplace=True)
df['Churn'].replace({'No' : 0, 'Yes' : 1}, inplace=True)

df.head()
```

5. **Drop redundant variables from the dummy variables created.** To remove redundancy from data, I removed one of the dummy variables created out of each encoding of a nomial categorical variable. The information from the dummy variable that was removed is inherently encoded in the other dummy variables. For example, the variable "Marital" had 5 unique values. I dropped the dummy variable "Married_NeverMarried" because the other 4 dummies would have 0s indicated that this customer's marital status was "never married".

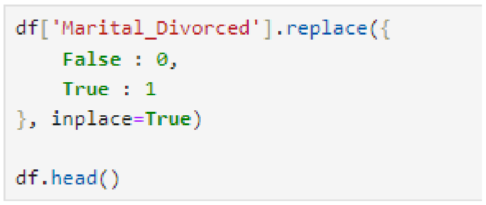
```
df['Marital'].unique()
```

```
array(['Widowed', 'Married', 'Separated', 'Never Married', 'Divorced'],
      dtype=object)
```

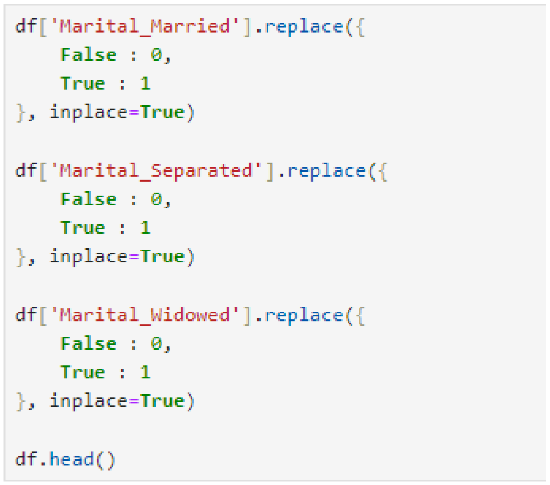
```
df = pd.get_dummies(df, columns=['Marital'])
df.head()
```

```
#remove the space in column title
df.rename(columns={'Marital_Never Married' : 'Marital_NeverMarried'}, inplace=True)
df.head()
```

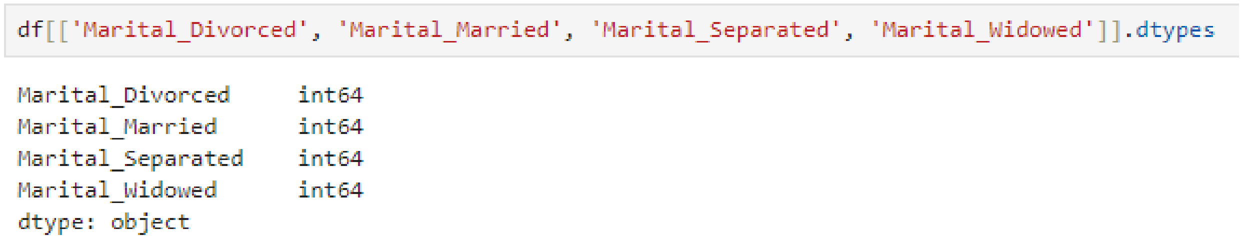
```
#drop "Never Married" variable -- info is inherently encoded in the other martial options

df.drop(columns=['Marital_NeverMarried'], inplace=True)
df.head()
```



```
df['Marital_Divorced'].replace({
    False : 0,
    True : 1
}, inplace=True)

df.head()
```



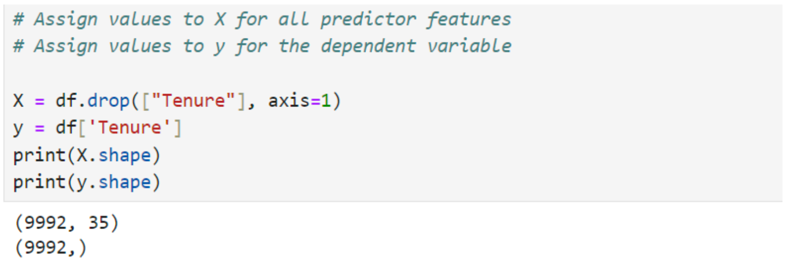
```
df['Marital_Married'].replace({
    False : 0,
    True : 1
}, inplace=True)

df['Marital_Separated'].replace({
    False : 0,
    True : 1
}, inplace=True)

df['Marital_Widowed'].replace({
    False : 0,
    True : 1
}, inplace=True)

df.head()
```



```
df[['Marital_Divorced', 'Marital_Married', 'Marital_Separated', 'Marital_Widowed']].dtypes
```

```
Marital_Divorced     int64
Marital_Married      int64
Marital_Separated    int64
Marital_Widowed      int64
dtype: object
```

6. **Feature selection.** I used SelectKBest from the scikit learn library to obtain the initial variables used in the machine learning model.



```
# Assign values to X for all predictor features
# Assign values to y for the dependent variable

X = df.drop(["Tenure"], axis=1)
y = df['Tenure']
print(X.shape)
print(y.shape)
```

```
(9992, 35)
(9992,)
```

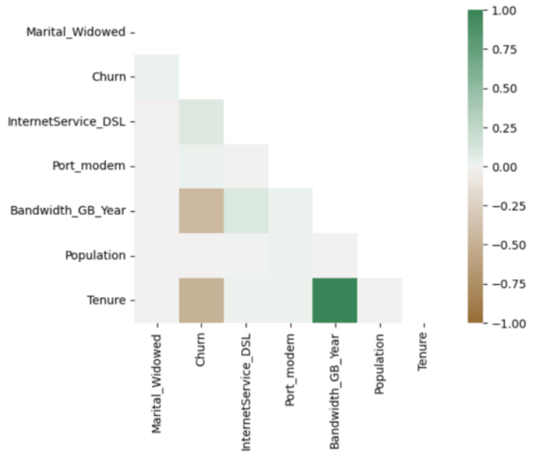
```
feature_names = X.columns

# Initialize the class and call fit_transform

# from sklearn.feature_selection import SelectKBest (already imported)

skbest = SelectKBest(score_func = f_classif, k='all')
X_new = skbest.fit_transform(X, y)
X_new.shape
```

```
# Finding P-values to select statistically significant features

p_values = pd.DataFrame({'Feature': X.columns, 'p_value':skbest.pvalues_}).sort_values('p_value')
p_values[p_values['p_value'] < .05]

features_to_keep = p_values['Feature'][p_values['p_value'] < .05]

# Print the name of the selected features

features_to_keep
```

```
27        Marital_Widowed
4                   Churn
22    InternetService_DSL
9              Port_modem
21       Bandwidth_GB_Year
0              Population
Name: Feature, dtype: object
```

7. **Look for correlation in the data**. I created a heatmap to explore multicollinearity in the data. I see that “Tenure” and “Bandwidth_GB_Year” are highly correlated.

```
df_corr = df_3.corr()


mask = np.triu(np.ones_like(df_3.corr(), dtype=bool))


axis_corr = sns.heatmap(
    df_corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(50, 500, n=500),
    mask=mask,
    square=True
)
```

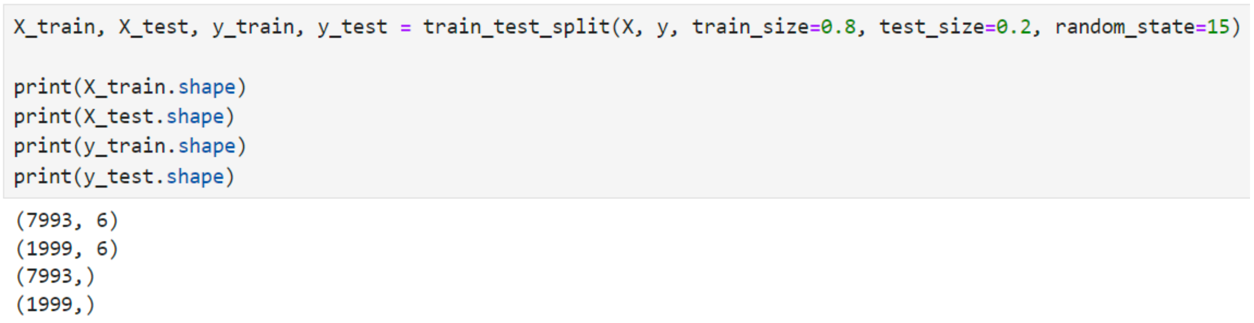


C4

A copy of the cleaned and prepared data set is submitted as a CSV file titled ‘prepared_clean_data_RF_prediction_D209 PA2.csv’.

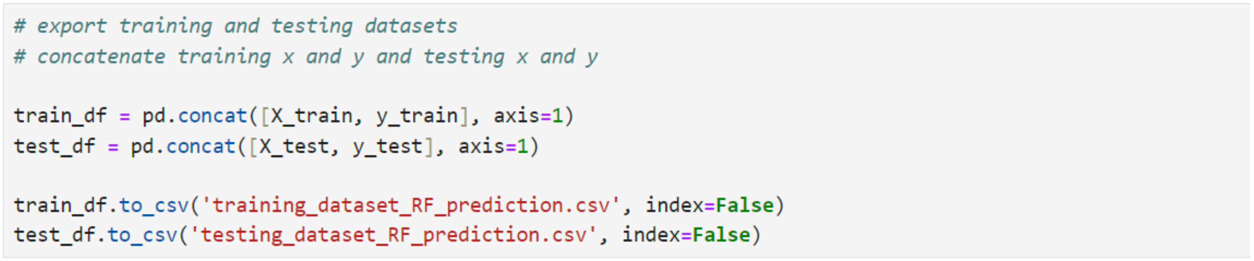
D1

The data was split into X_train, X_test, y_train, and y_test data sets. When I exported the training and testing sets, I concatenated the X_train and y_train sets together as well as the X_test and y_test sets, Two files from part D1 are included in the submission of this performance assessment as 'training_dataset_RF_prediction.csv' and 'testing_dataset_RF_prediction.csv'.



```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, random_state=15)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(7993, 6)
(1999, 6)
(7993,)
(1999,)
```



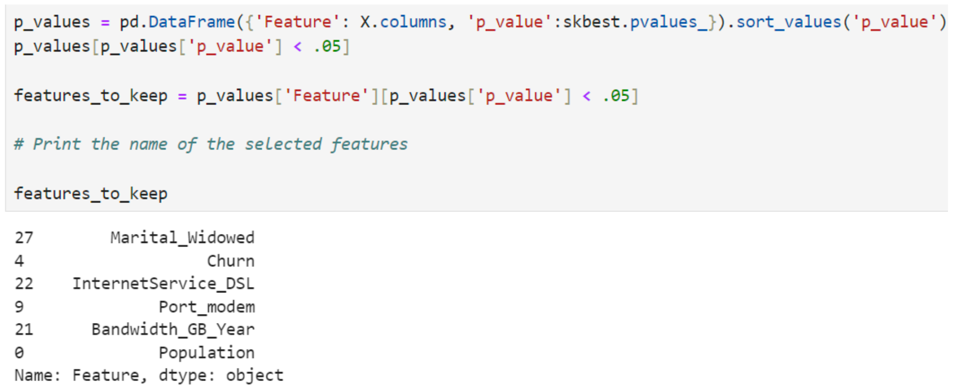
```
# export training and testing datasets
# concatenate training x and y and testing x and y

train_df = pd.concat([X_train, y_train], axis=1)
test_df = pd.concat([X_test, y_test], axis=1)

train_df.to_csv('training_dataset_RF_prediction.csv', index=False)
test_df.to_csv('testing_dataset_RF_prediction.csv', index=False)
```

D2

After preprocessing the data, I analyzed the data using Random Forest Regressor as the prediction method. "Random Forests (RF) are an ensemble machine learning method for classification and regression. My analysis a regression problem because the target variable, tenure, is a numerical data type. RF constructs multiple decision trees at the training time and outputs the mean of the predictions for regression. Each of the trees makes its own individual prediction. These predictions are then averaged to produce a single result. The averaging makes an RF better than a single decision tree, improving its accuracy and reducing overfitting. A prediction from the RF Regressor is an average of the trees' predictions in the forest" (Elleh, nd). Hyperparameter tuning is performed using GridSearchCV and the best features for prediction will be selected via SelectKBest.

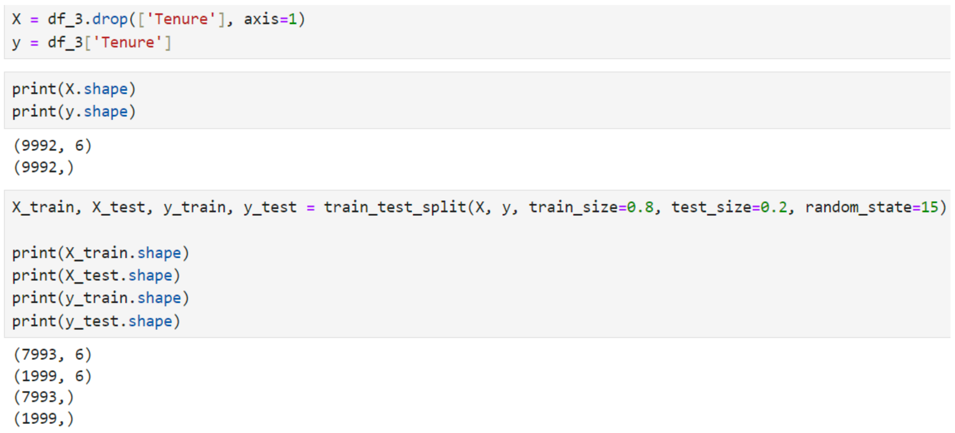
Here I will describe the analysis technique I used to analyze the data and include screenshots of intermediate calculations.

1. **Test train split.** I defined the feature variables as X and the target variable as y. The selected features from SelectKBest are defined as X and the target variable, "Tenure", is defined as y. Then, these sets were split into training and testing sets using an 80/20 split.



```
p_values = pd.DataFrame({'Feature': X.columns, 'p_value':skbest.pvalues_}).sort_values('p_value')
p_values[p_values['p_value'] < .05]

features_to_keep = p_values['Feature'][p_values['p_value'] < .05]

# Print the name of the selected features

features_to_keep
```

```
27        Marital_Widowed
4                   Churn
22    InternetService_DSL
9              Port_modem
21      Bandwidth_GB_Year
0              Population
Name: Feature, dtype: object
```



```
X = df_3.drop(['Tenure'], axis=1)
y = df_3['Tenure']
```

```
print(X.shape)
print(y.shape)
```

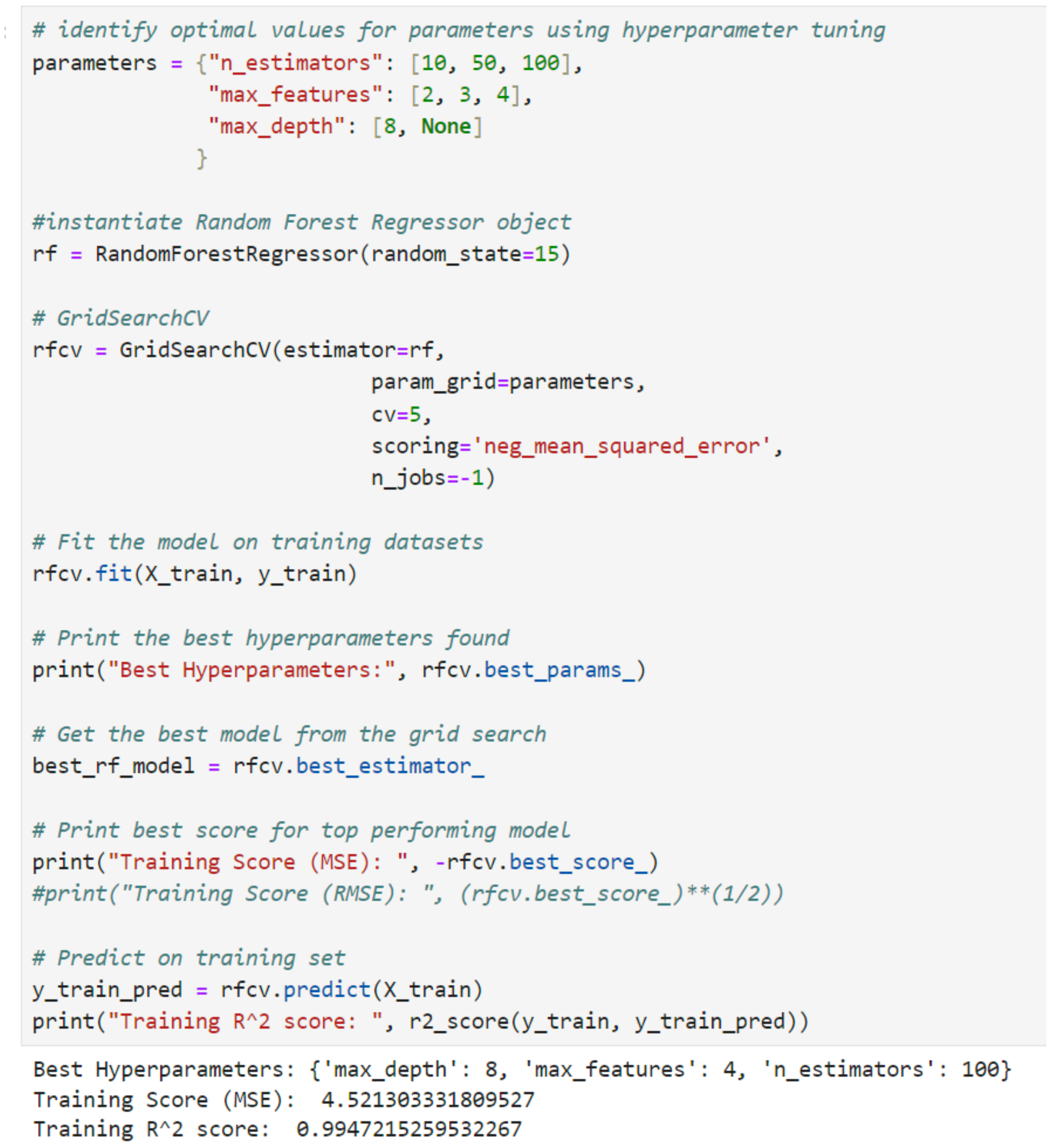
```
(9992, 6)
(9992,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, random_state=15)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(7993, 6)
(1999, 6)
(7993,)
(1999,)
```

2. **Fit the model.** I identified optimal values for parameters using hyperparameter tuning, instantiated the Random Forest Regressor object, applied GridSearchCV object to tune the hyperparameters, fit the model on the training datasets, and evaluated model performance using mean squared error (MSE) and the coefficient of determination (R^2). For the training data $MSE = 4.52$ and $R^2 = 0.99$.

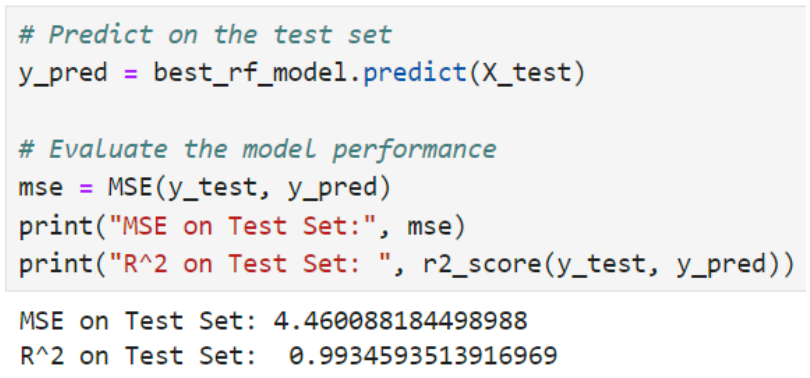
   It is not necessary to scale the features when using random forests because they are not sensitive to the variance in the data (Kawerk, nd).



```
# identify optimal values for parameters using hyperparameter tuning
parameters = {"n_estimators": [10, 50, 100],
              "max_features": [2, 3, 4],
              "max_depth": [8, None]
             }

#instantiate Random Forest Regressor object
rf = RandomForestRegressor(random_state=15)

# GridSearchCV
rfcv = GridSearchCV(estimator=rf,
                    param_grid=parameters,
                    cv=5,
                    scoring='neg_mean_squared_error',
                    n_jobs=-1)

# Fit the model on training datasets
rfcv.fit(X_train, y_train)

# Print the best hyperparameters found
print("Best Hyperparameters:", rfcv.best_params_)

# Get the best model from the grid search
best_rf_model = rfcv.best_estimator_

# Print best score for top performing model
print("Training Score (MSE): ", -rfcv.best_score_)
#print("Training Score (RMSE): ", (rfcv.best_score_)**(1/2))

# Predict on training set
y_train_pred = rfcv.predict(X_train)
print("Training R^2 score: ", r2_score(y_train, y_train_pred))
```

```
Best Hyperparameters: {'max_depth': 8, 'max_features': 4, 'n_estimators': 100}
Training Score (MSE):  4.521303331809527
Training R^2 score:  0.9947215259532267
```

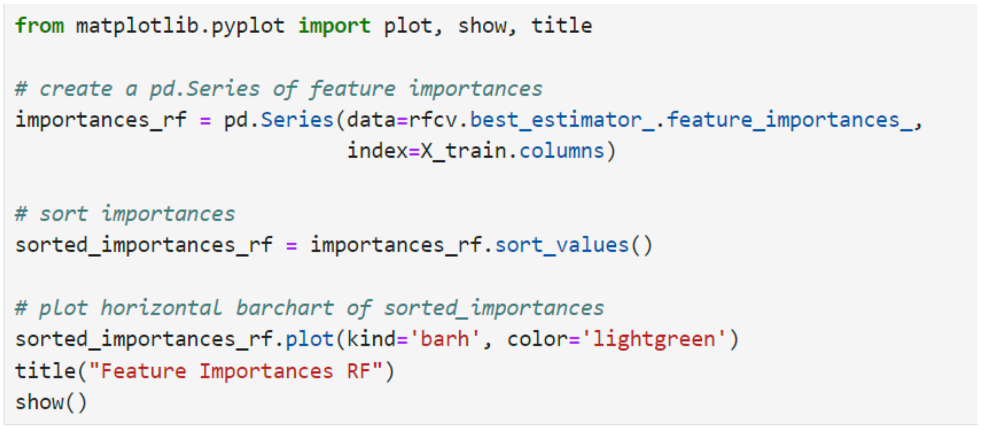
3. **Predict on testing dataset.** Next, I predict on the testing dataset and evaluate the model performance by calculating $MSE = 4.46$ and $R^2 = 0.99$.



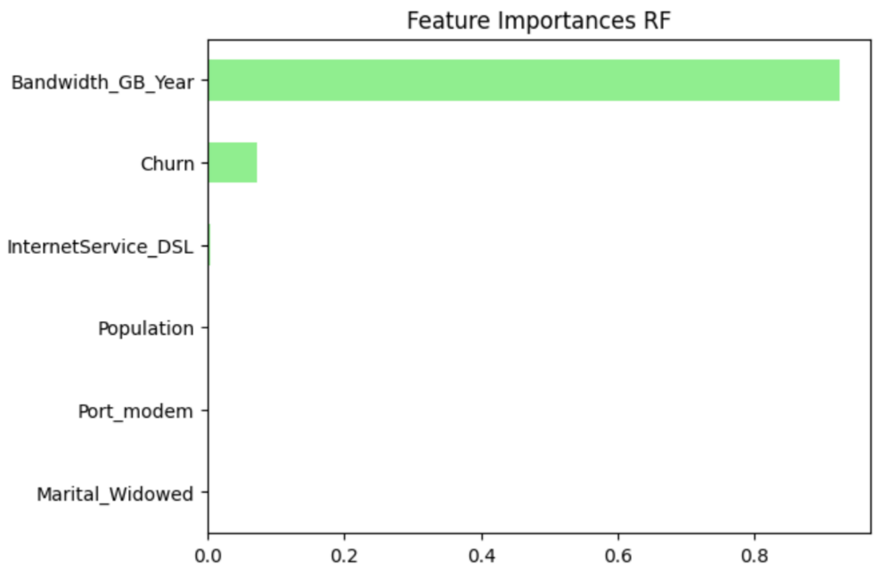
```
# Predict on the test set
y_pred = best_rf_model.predict(X_test)

# Evaluate the model performance
mse = MSE(y_test, y_pred)
print("MSE on Test Set:", mse)
print("R^2 on Test Set: ", r2_score(y_test, y_pred))
```

```
MSE on Test Set: 4.460088184498988
R^2 on Test Set:  0.9934593513916969
```

4. **Feature Importance.** Lastly, I determined which features were of most importance in predicting “Tenure”. “Bandwidth_GB_Year” is by far the most important feature. The next most important feature is “Churn”.



```python
from matplotlib.pyplot import plot, show, title

# create a pd.Series of feature importances
importances_rf = pd.Series(data=rfcv.best_estimator_.feature_importances_,
                           index=X_train.columns)

# sort importances
sorted_importances_rf = importances_rf.sort_values()

# plot horizontal barchart of sorted_importances
sorted_importances_rf.plot(kind='barh', color='lightgreen')
title("Feature Importances RF")
show()
```



D3

The code used to perform the predictive analysis explained in part D2 is included as a Jupyter Notebook file in the submission of this performance assessment.

E1

Here I will explain the mean squared error (MSE) and the accuracy of my prediction method.

MSE is the difference between the predicted values and the actual values. It determines how good the estimate is based on the algorithm analysis. The lower the value for MSE, the more accurately a model is at predicting values (Elleh, nd). The MSE for both the training and testing sets are about equal. The training $MSE = 4.52$ while the testing $MSE = 4.46$, both are relatively low values and indicate that the model is good at predicting value for customer tenure.

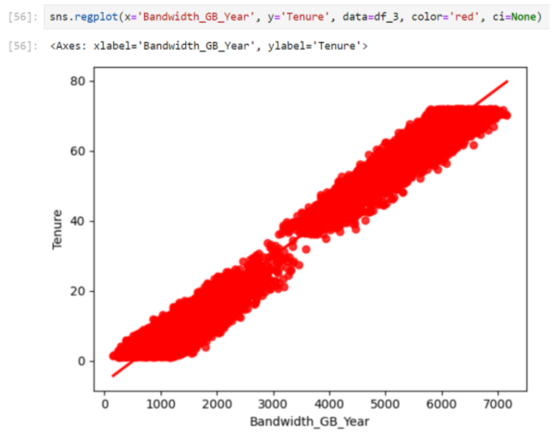
R^2 is called the coefficient of determination. It is the metric that measures the relationship between the residual sum of squares and the total sum of squares. It tells how well the model fits the data. The higher the R^2 value, the better the model fits the data. $R^2 = 0.99$ for both the training and testing data, indicating that 99% of the variation is explained by the model.

E2

The results and implications of my predictive analysis are discussed here.

The model fits the data well. MSE = 4.46 for the testing data indicating that the model is good at predicting value for customer tenure. Also, $R^2 = 0.99$ for the testing data, indicating that 99% of the variation can be explained by the model. I used hyperparameter tuning to find the optimal value for n_estimators, max_features, and max_depth: 100, 4, and 8 respectively.

“Bandwidth_GB_Year” is the most important feature in predicting the tenure of a customer, with “Churn” as a distant second. The implication is that more data a customer uses in a year, the more tenured the customer becomes. The company should spend their efforts to increase the amount of data customers use per year and ensure quality products so customers will want to spend time using data.


```
[56]: sns.regplot(x='Bandwidth_GB_Year', y='Tenure', data=df_3, color='red', ci=None)
[56]: <Axes: xlabel='Bandwidth_GB_Year', ylabel='Tenure'>
```

E3

One limitation of my data analysis is as follows. Random forests are powerful machine learning models that have the ability to predict non-linear data. “However, a random forest cannot extrapolate. It can only make predictions that are an average of previously observed labels. This could be a problem when the training and prediction inputs differ in their range” (Elleh, nd).

E4

Based on the analysis and the model’s accuracy, the company can use this random forest model to predict customers’ tenure. A recommended course of action is that since “Bandwidth_GB_Year” is the most important feature in predicting the tenure of a customer, the company should spend their efforts to increase the amount of data customers use per year and ensure quality products so customers will want to spend time using data. Also, the company should explore why customers use less data and figure out how to increase their usage.

F

A Panopto video recording is provided in the submission of this performance assessment.

G

Web sources used to acquire segments of code to support the application:

Elleh, Festus. “Task 2: Predictive Analysis – Decision Trees, Random Forests, Advanced Regression (Lasso or Ridge).” Data Mining I – D209, nd.

Kawerk, Elie, et. al. “Machine Learning with Tree-Based Models in Python” Datacamp, app.datacamp.com/learn/courses/machine-learning-with-tree-based-models-in-python

H

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

Elleh, Festus. “Task 2: Predictive Analysis – Decision Trees, Random Forests, Advanced Regression (Lasso or Ridge).” Data Mining I – D209, nd.

Kawerk, Elie, et. al. “Machine Learning with Tree-Based Models in Python” Datacamp, app.datacamp.com/learn/courses/machine-learning-with-tree-based-models-in-python

I

Professional communication is demonstrated in the content and presentation of my Performance Assessment.