Task 2: Predictive Analysis – D209

Abhishek Aern

Western Governor University

Contents

Part I - Research Question

A1: Question for analysis

Can Random Forest Algorithm accurately predict the customer tenure based on various factors present in the customer churn dataset? and how can these factors be used to improve customer retention strategies?

A2: Goal of analysis

The goal of the analysis using the Random Forest Algorithm is to identify the key drivers of customer tenure and develop strategies that will help businesses retain customers for longer periods.

I will identify all factors that may influence customer tenure by defining it as a target variable and identifying factors that may influence this. Then Train a Random Forest algorithm on the training set and Evaluate the performance of the model on the testing set. Further analysis can be done to identify which variables are the most important predictors of customer tenure, and how they contribute to the overall prediction accuracy of the Random Forest model.

Part II - Method Justification

B1: Explanation of Random Forests

(R, 2021) Random Forest is a supervised machine learning algorithm that can be used to solve classification and regression problems. It is a popular algorithm for predictive modeling due to its high accuracy and ability to handle large datasets with many features.

Random Forest can avoid overfitting the training data by using multiple decision trees and aggregating their predictions. This helps to improve the generalization performance of the model on new, unseen data. Random Forest is a powerful and flexible algorithm that can be used to accurately predict customer tenure based on a variety of factors. Its ability to handle large datasets and complex relationships, as well as its feature importance analysis, make it a good fit for my analysis.

B2: Assumptions of Random Forests

(Vishalmendekarhere, 2021) Here are some of the assumptions about Random Forests

1. Random Forest assumes that the observations in the dataset are independent of each other.
2. Random Forest does not assume that the target variable follows a normal distribution. This means that the target variable can have any distribution.
3. Random Forest does not assume linearity between the predictors and the target variable. This means that the relationship between the predictors and the target variable can be nonlinear.
4. Random Forest is robust to outliers in the data, meaning that it can handle data points that are far away from the majority of the data.

B3: Packages or Libraries List

Here is a list of packages and libraries for implementing the Random Forests algorithm in Python:

Library name	**Usage**
Pandas	A library for working with data frames and manipulating data
NumPy	A library for working with arrays and numerical operations
Scikit-learn	A machine learning library that provides various tools for classification, regression, and clustering, including Random Forests.
seaborn	For Visualizations like correlation matrices.
matplotlib. pyplot	For Visualization like a scatter plot.
RandomForestRegressor	Random forest regressor that fits several decision trees
GridSearchCV	Performs exhaustive search over the specified parameter values for an estimator (Randomforest in this case and searching for the optimal value of 'n_estimators', 'max_features', 'max_depth').
mean_squared_error, r2_score	For getting the R2 (coefficient of determination) and Mean squared error metric.

train_test_split	The train_test_split function is used from the Scikit-learn library in Python which is commonly used to split a dataset into training and testing sets for machine learning.
plot_tree	To make the random forest decision tree.

Part III - Data Preparation

C1: Data Preprocessing

The goal of data preprocessing for the Random Forest is to prepare the dataset for the algorithm by ensuring that the data is in the appropriate format and is suitable for use in the algorithm.

Data preprocessing is an important step in preparing the data for the Random Forest algorithm. By handling missing values, encoding categorical variables, selecting relevant features, and splitting the data into training and testing sets, we can improve the performance of the algorithm and obtain more accurate predictions.

C2: Dataset variables

Initially, the churn dataset has many variables which are not relevant to predict customer tenure and hence get deleted while importing the data.

Target variable –

Tenure: This is a continuous variable representing the number of years that the customer has been a client with the service provider.

Independent variable –

Variable Name	**Type**
Income	Continuous
Age	Continuous
Children	Continuous
Churn	Categorical
Multiple	Categorical
Outage_sec_perweek	Continuous
Contacts	Continuous

MonthlyCharge	Continuous
Bandwidth_GB_Year	Continuous
InternetService	Categorical
PaperlessBilling	Categorical
Contract	Categorical
Gender	Categorical
Yearly_equip_failure	Continuous

C3: Steps for Analysis

(Kumar D. , 2018) Here is the task that I am following for data preprocessing-

Handling missing and Null values: Checking the missing and null values using the .isna() and .isnull() functions. No missing and null values were found in the provided dataset.

```
# check for null
print("Check for Nulls")
print("-"*100)
print(df_churn.isnull().any())

# check for missing values
print("Check for Missing Values")
print("-"*100)
print(df_churn.isna().any())
```

Handling categorical variables: I am using categorical variables such as Contract, Gender, InternetService, Multiple, PaperlessBilling, and Churn, which need to be converted into a numerical format using Label encoding and one-hot encoding.

```
# Label encoding for Churn, multiple and PaperlessBilling variables

cust_map = {"No": 0, "Yes": 1}         # Defining the mapping

df_churn['Churn'] = df_churn['Churn'].map(cust_map)
df_churn['Multiple'] = df_churn['Multiple'].map(cust_map)
df_churn['PaperlessBilling'] = df_churn['PaperlessBilling'].map(cust_map)

# Perform one-hot encoding

df_churn = pd.get_dummies(df_churn, columns=['InternetService', 'Contract',
'Gender'])

# convert uint8 columns to int
df_churn = df_churn.applymap(lambda x: int(x) if isinstance(x, int) else x)
```

Remove features not required for the analysis - deleting all the features which are not significant in predicting Tenure from my previous linear regression analysis(D208) (Kumar D. , 2018).

The following features are deleted initially during import –

```
df_churn.drop(columns=['CaseOrder', 'Lat', 'Lng', 'Interaction', 'UID',
'City',    'State', 'Zip', 'Population', 'Job', 'Techie', 'Port_modem',
'Tablet',
'Marital', 'Phone', 'OnlineSecurity', 'OnlineBackup',
'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
'PaymentMethod','Email','Item1', 'Item2', 'Item3', 'Item4',
'Item5', 'Item6', 'Item7', 'Item8', 'County', 'Area',
'TimeZone'],inplace=True)
```

(Carolin Strobl, 2007) Standardizing the variables is generally not required for Random Forest models because Random Forests is a tree-based method and are not sensitive to the scale of the input features.

Multicollinearity is also less of a concern with Random Forest models compared to other linear models because Random Forests are non-linear and non-parametric, and do not make assumptions about the relationship between the predictor variables (Carolin Strobl, 2007).

All this code is present in the attached .ipynb file provided in the attachment.

C4: Prepared Dataset

The prepared clean data is provided in a CSV file.

Part IV - Analysis

D1: Splitting the Data

Defined the dependent variable (y) and independent variables (X) from df_churn.

(Pedregosa) The train_test_split function from scikit-learn is used to randomly split the data into training and testing sets, where the test_size parameter specifies the proportion of the data to be used for testing (in this case, 20%). The random_state parameter is set to a fixed value (0 in this case) to ensure that the same split is obtained every time the code is run. The function returns four arrays: X_train and

y_train represent the training data (features and target), while X_test and y_test represent the testing data (features and target).

```
From sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,
random_state = 0)
```

All the training and testing data sets are exported to a CSV file and will be provided with the submission.

D2: Output and Intermediate Calculations

After splitting the data, I created a Random Forest regressor with 100 trees and fit it to the training data using the fit() method. The initial model is created with random_state=42 so that the random numbers generated during the training process will be the same every time the code is run, making the results reproducible. The default value for n_estimators in scikit-learn is 100.

Then the trained model was used to make predictions on the test and training sets and evaluate its performance using metrics such as mean squared error (MSE), Root Mean Squared Error (RMSE) or R-squared.

The MSE, RMSE, and R-squared for the initial model are calculated below -

```
# Compute Initial model performance measures with training

mse_train_init = mean_squared_error(y_train, y_train_pred)
rmse_train_init = mse_train_init ** (1/2)
r2_train_init = r2_score(y_train, y_train_pred)

print("Initial Model : Training Mean Squared Error (MSE):",  round(mse_train_init,3))
print("Initial Model : Training - Root Mean Squared Error (RMSE): ", round(rmse_train_init,3))
print("Initial Model : Training - R-squared (R2 ) :",  r2_train_init)
```

```
Initial Model : Training Mean Squared Error (MSE): 0.177
Initial Model : Training - Root Mean Squared Error (RMSE):  0.421
Initial Model : Training - R-squared (R2 ) : 0.999746017414182
```

```
# Make predictions on the testing data
y_pred_init = rf_model.predict(X_test)

# Calculate mean squared error (MSE) and R-squared (R2) score
mse_init = mean_squared_error(y_test, y_pred_init)
rmse_init = mse_init ** (1/2)
r2_init = r2_score(y_test, y_pred_init)

print("Initial Model : Testing Mean Squared Error (MSE):",  round(mse_init,3))
print("Initial Model : Testing - Root Mean Squared Error (RMSE): ", round(rmse_init,3))
print("Initial Model : Testing - R-squared (R2 ) :",  r2_init)
```

```
Initial Model : Testing Mean Squared Error (MSE): 1.292
Initial Model : Testing - Root Mean Squared Error (RMSE):  1.137
Initial Model : Testing - R-squared (R2 ) : 0.9981693628262789
```

(Kumar A. , 2020) Random Forest models can provide information about the relative importance of each feature in the dataset. The importance of each feature is calculated based on the reduction in the impurity of the split achieved by the feature.

```
# Get feature importance scores
importances = rf_model.feature_importances_

# view the feature scores
feature_scores =
pd.Series(importances,index=X_train.columns).sort_values(ascending=False)
```

Here is the output –

```
Bandwidth_GB_Year              0.988684
InternetService_DSL            0.004721
MonthlyCharge                  0.004174
Age                            0.000859
Children                       0.000612
Income                         0.000222
Outage_sec_perweek             0.000214
Contacts                       0.000075
Gender_Male                    0.000060
Gender_Female                  0.000051
InternetService_Fiber Optic    0.000050
InternetService_None           0.000045
Multiple                       0.000042
Yearly_equip_failure           0.000040
Churn                          0.000040
PaperlessBilling               0.000032
Contract_Month-to-month        0.000028
Contract_Two Year              0.000025
Contract_One year              0.000022
Gender_Nonbinary               0.000004
dtype: float64
```

By examining the feature ranking, the Bandwidth_GB_Year is the most important feature in the dataset and the least important feature is Gender_Nonbinary.

To improve the performance of the model using hyperparameter tuning using GridSearchCV, I am setting up a hyperparameter tuning grid with a range of values for each hyperparameter. then set up a new Random Forest model and a GridSearchCV object for hyperparameter tuning and fit the GridSearchCV object to

the transformed training data and print the best hyperparameters and best score. The scoring parameter neg_mean_squared_error was used to negate the MSE. This means that a higher score (i.e., less negative) is better, and the function will try to find the hyperparameters that minimize the negative mean squared error.

```
# Set up hyperparameter tuning grid
param_grid = {'n_estimators': [50, 100, 200]}

# Set up Random Forest model for hyperparameter tuning
rf_model_tuned = RandomForestRegressor(random_state=42)

# Set up GridSearchCV object for hyperparameter tuning
grid_search = GridSearchCV(rf_model_tuned, param_grid, cv=5,
scoring='neg_mean_squared_error')

# Fit GridSearchCV object to training data
grid_search.fit(X_train, y_train)
```

The Best parameter and best score are shown below –

```
# Print best hyperparameters and best score
print("Best parameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Best parameters:  {'n_estimators': 200}
Best score:  -1.3314652938742175
```

The Best score of -1.3314 obtained using the scoring metric 'neg_mean_squared_error' means that the model has a mean squared error of 1.3314 on the validation set. Since this is a negative score, it indicates that the lower the value, the better the performance of the model.

D3: Code Execution

The initial model was built using below code –

```
## Create a Random Forest regressor with default hyperparameters
rf_model = RandomForestRegressor(random_state=42)

# Fit the training data
rf_model.fit(X_train, y_train)

# Make Training predictions
y_train_pred = rf_model.predict(X_train)
y_train_pred
```

```
# Compute Initial model performance measures with training

mse_train_init = mean_squared_error(y_train, y_train_pred)
rmse_train_init = mse_train_init ** (1/2)
r2_train_init = r2_score(y_train, y_train_pred)

print("Initial Model : Training Mean Squared Error (MSE):",
round(mse_train_init,3))
print("Initial Model : Training - Root Mean Squared Error (RMSE): ",
round(rmse_train_init,3))
print("Initial Model : Training - R-squared (R2 ) :",  r2_train_init)

# Make predictions on the testing data
y_pred_init = rf_model.predict(X_test)

# Calculate the mean squared error (MSE) and R-squared (R2) score
mse_init = mean_squared_error(y_test, y_pred_init)
rmse_init = mse_init ** (1/2)
r2_init = r2_score(y_test, y_pred_init)

print("Initial Model : Testing Mean Squared Error (MSE):",
round(mse_init,3))
print("Initial Model : Testing - Root Mean Squared Error (RMSE): ",
round(rmse_init,3))
print("Initial Model : Testing - R-squared (R2 ) :",  r2_init)
```

The code for feature selection and model performance improvement is shown in above section D2 and the performance and output of this new model are described in the next section.

All the code is provided in the .ipynb file provided with the submission.

Part V- Data Summary and Implications

E1: Accuracy and MSE

(Srivastava, 2019) MSE is the difference between the predicted values and the actual values.

Root Mean Squared Error (RMSE) is the square root of the MSE. This is defined as the square root of the average squared distance between the actual score and the predicted score.

Both RMSE and MSE can show if the degree of error is significant in the model.

R Squared (R2) - metric measures the relationship between the residual sum of squares (RSS) and the total sum of squares (TSS). The higher the R-squared, the better the model fits the data (Srivastava, 2019).

```
# Compute model performance measures with training

mse_train = mean_squared_error(y_train, y_train_pred)
rmse_train = mse_train ** (1/2)
r2_train = r2_score(y_train, y_train_pred)

print("Final Model : Training Mean Squared Error (MSE):",  round(mse_train,3))
print("Final Model : Training - Root Mean Squared Error (RMSE): ", round(rmse_train,3))
print("Final Model : Training - R-squared (R2 ) :",  r2_train)
```

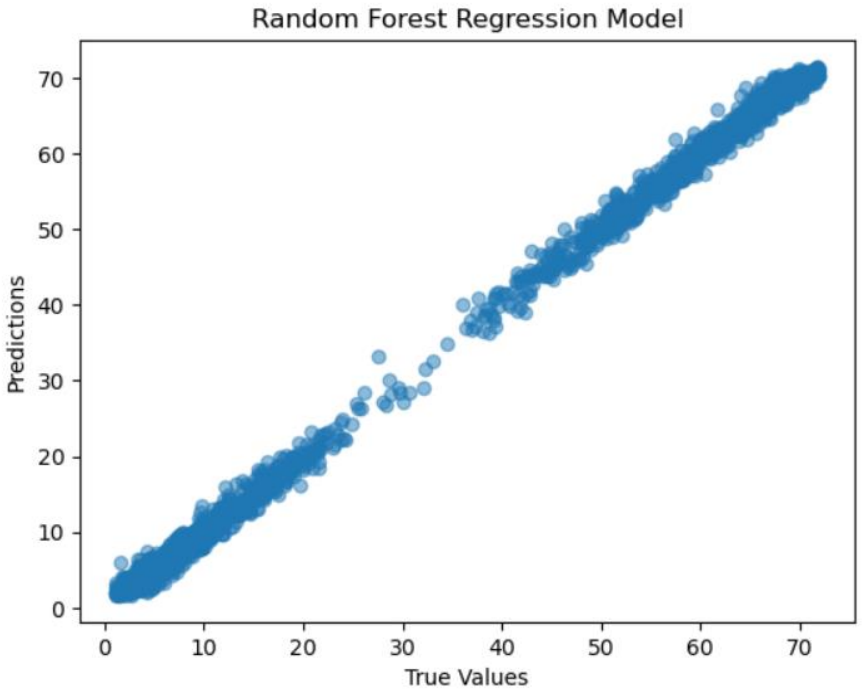
```
Final Model : Training Mean Squared Error (MSE): 0.172
Final Model : Training - Root Mean Squared Error (RMSE):  0.415
Final Model : Training - R-squared (R2 ) : 0.999753286214838
```

```
# Compute model performance measures - Testing
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** (1/2)
r2 = r2_score(y_test, y_pred)
```

```
# Print model performance measures
print("Final Model : Testing - Mean Squared Error (MSE): ", round(mse,3))
print("Final Model : Testing - Root Mean Squared Error (RMSE): ", round(rmse,3))
print("Final Model : Testing - R-squared (R2 ): ", r2)
```

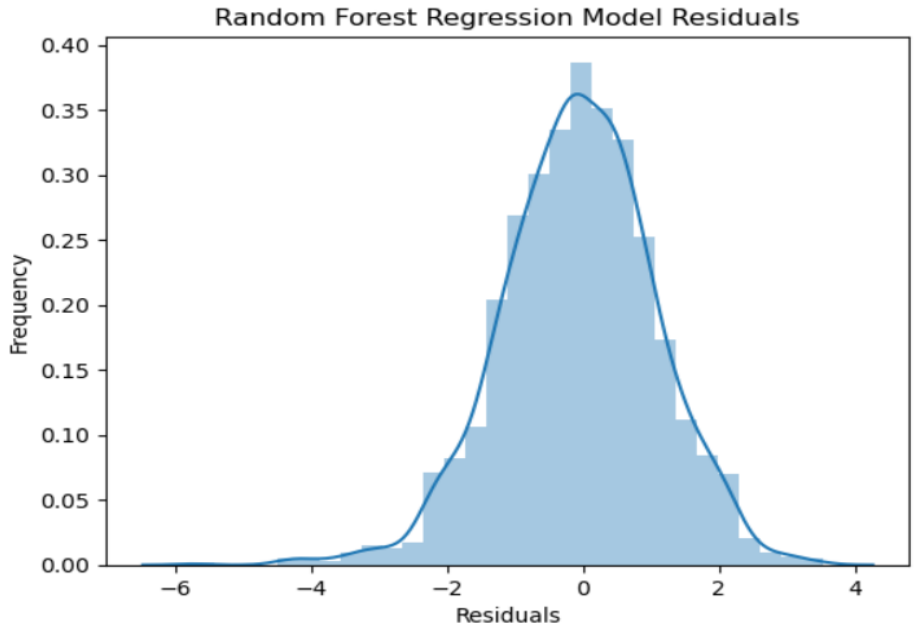
```
Final Model : Testing - Mean Squared Error (MSE):  1.274
Final Model : Testing - Root Mean Squared Error (RMSE):  1.129
Final Model : Testing - R-squared (R2 ):  0.9981940234016548
```

Scatter plot of predicted vs true values - The Scatter plot shows a tight cluster of points around the straight line which indicates that the model is accurately predicting the values.

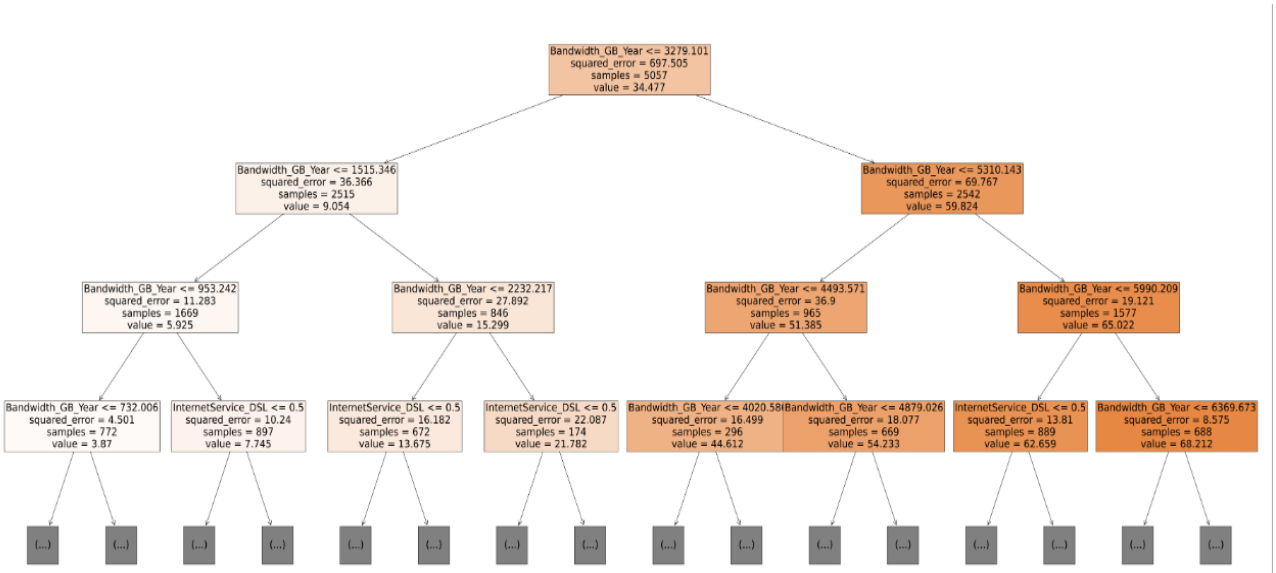


Random Forest Regression Model Residuals :

The residuals are randomly distributed around zero with no discernible pattern which indicates that the model is accurately capturing the information in the data and making unbiased predictions.



Below is shown the random forest decision tree with fewer branches by setting the max_depth parameter of the plot_tree function to a smaller value.



E2: Results and Implications

(Brownlee, 2016) Based on the above performance matrix for the final model -

- Training Mean Squared Error (MSE) of 0.172 means that on average, the model's predictions on the training set are off by the square root of 0.172 from the actual values.
- Training - Root Mean Squared Error (RMSE) of 0.415 means that on average, the model's predictions on the training set are off by 0.415 units from the actual values.
- The training R-squared (R2) of 0.9997 means that the model explains 99.97% of the variance in the target variable on the training data.
- A testing mean squared error (MSE) of 1.274 means that on average, the squared difference between the predicted values and the actual values of the target variable is 1.274. In this case, the testing MSE is significantly higher than the training MSE which suggests that the model may be slightly overfitting to the training data but the difference is not much.
- The testing RMSE of 1.129 indicates that the average error in predicting tenure for the test data is around 1.129 months.
- The testing R-squared (R2) is 0.9981 which is very similar to the training data R-squared (R2). (Brownlee, 2016).

A lower MSE and RMSE indicate better performance of the model, meaning that the predicted values are closer to the actual values. A higher R2 value indicates a better fit of the model to the data, meaning that the model explains a larger proportion of the variance in the target variable.

Overall, the final model has a slightly better performance compared to the initial model, with slightly lower MSE and RMSE values and slightly higher R2 values for both training and testing sets.

E3: Limitation

Here are some of the limitations –

Although Random Forest is designed to reduce overfitting, it is still possible to ove rfit the model if the number of trees is too high or if the data is noisy or biased.

Random forests can be computationally expensive and time-consuming for large datasets or complex models with many trees and features. This can limit its scalability and make it impractical for some applications. The initial model has a very high R-squared value for both training and testing data, indicating that the model fits the data very well. However, the testing means squared error and root mean squared error are relatively high, which suggests that the model may be overfitting the training data and not generalizing well to new data.

Additionally, the feature selection process may have some limitations as well, as it relies on statistical tests to identify important features and may not capture complex nonlinear relationships between variables. Also, the hyperparameter tuning process used in this analysis may not have identified the optimal set of parameters for the model, and further experimentation may be needed to improve performance.

E4: Course of Action

The model suggests that tenure is one of the most important factors in predicting customer churn. Therefore, the organization can focus on implementing a customer retention strategy to improve customer loyalty and reduce churn. The top 10 features identified by the model can be used by the organization to prioritize their efforts and resources towards improving these features.

Based on the above matrix, the organization can conclude that the random forest model has performed well on the training and testing data with low mean squared errors and high R-squared values, indicating that the model explains the variation in the target variable well. However, there is a slight increase in the testing mean squared error and a decrease in the R-squared value in the final model, indicating that the model may be slightly overfitting to the training data.

To address this, the organization can consider using regularization techniques or collecting more data to improve the model's generalization ability. Additionally, they can explore other algorithms and compare their performance to the random forest model to determine if a better-performing model exists.

Part VI - Demonstration

G. Provide a Panopto recording

The Panopto recording link is provided with the submission.

H. Third-Party Code References

Moffitt, C. (2017, February 06). Guide to Encoding Categorical Values in Python. pbpython.com: https://pbpython.com/categorical-encoding.html

pandas.pydata.org. (n.d.). pandas.get_dummies. https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

Pedregosa. (n.d.). Scikit-learn: Machine Learning in Python. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

R, S. E. (2021, June 17). Understand Random Forest Algorithms With Examples. Retrieved from https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Koehrsen, W. (2027, December 27). https://towardsdatascience.com/random-forest-in-python-24d0893d51c0

GeeksforGeeks. (n.d.). Working with CSV files in Python. https://www.geeksforgeeks.org/working-csv-files-python/

I. References

Brownlee, J. (2016, May 25). Metrics To Evaluate Machine Learning Algorithms in Python. https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python

Kumar, D. (2018, December 25). Introduction to Data Preprocessing in Machine Learning. https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d

Vishalmendekarhere. (2021, Jan 17). ML Assumptions, Pros & Cons. https://medium.com/swlh/its-all-about-assumptions-pros-cons-497783cfed2d

Srivastava, T. (2019, August 6). 12 Important Model Evaluation Metrics for Machine Learning Everyone Should Know. https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/