# PERFORMANCE ASSESSMENT

**Task 1 | Classification 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 k-nearest neighbors (kNN) classification method, can we predict if a customer will churn?”

# A2

One goal of the data analysis is to build a supervised machine learning  model using k-nearest neighbors (kNN) classification method to identify customers who are at risk of churning so that the company can a) know what key features lead to customer churn and b) take measures to retain these customers.

# B1

The classification method I chose is k-nearest neighbors, kNN, a machine learning method that I used for a classification problem, which works in the following way. Given an unlabeled data point, kNN predicts its label by looking at the k closest labeled data points and taking a majority vote (Muller, nd). The entire data set is split into training and testing sets. “During the training phase, the kNN algorithm stores the entire training dataset as a reference. When making predictions, it calculates the distance between the input data point and all the training examples, using a chosen distance metric such as Euclidean distance.” (Srivastava, 2023). The k closest neighbors are identified and the data point that we’re seeking a label for is assigned the most common label of its k neighbors.

An expected outcome of the analysis is to create an algorithm that will predict whether a customer will churn based on selected key features. Other expected outcomes include being able to calculate how accurate the model is and how well it does at distinguishing between the classes. Also the test data will be classified according to their k closest neighbors, and k will be optimally calculated (Elleh, nd).

# B2

One assumption of the kNN classification method is that similar things exist in proximity to each other. Conversely, dissimilar things are far apart from each other. Because of this, kNN is sensitive to outliers. (Elleh, nd).

# 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 to use in kNN model |
| 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.neighbors import KneighborsClassifer | kNN classification |
| 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.metrics import classification report | Classification report metrics |
| From sklearn.metrics import confusion_matrix | Confusion matrix |
| From sklearn.metrics import roc_curve | ROC curve plotting |
| From sklearn.metrics import roc_auc_score | AUC score calculations |
| From sklearn.metrics import accuracy_score | Accuracy score calculations |
| From sklearn.pipeline import Pipeline | Using a pipeline to scale the data and fit the model |

# C1

One data preprocessing goal relevant to the classification method from part A1 is as follows. Prepare the data to be ready for kNN classification by: imputing missing data, check for outliers, encoding categorical variables, looking for correlation in the data, scaling the data 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 kNN classification question for part A1, as well as their types. “Churn” is the target variable. All others are feature variables.

| Categorical Variables | Numeric Variables |
|---|---|
| Churn | Monthly_Charge |
| Bandwidth_GB_Year | Tenure |
| StreamingMovies | |
| Contract_Month-to-Month | |
| StreamingTV | |
| Contract_OneYear | |
| Multiple | |
| InternetService_DSL | |
| Techie | |
| InternetService_FiberOptic | |
| DeviceProtection | |
| OnlineBackup | |
| PaymentMethod_ElectronicCheck | |
| Gender_Male | |
| Gender_Female | |
| Phone | |

# 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 reexpressed “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(["Churn"], axis=1)
y = df['Churn']
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
```

```
(9992, 35)
```

```
### 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
```

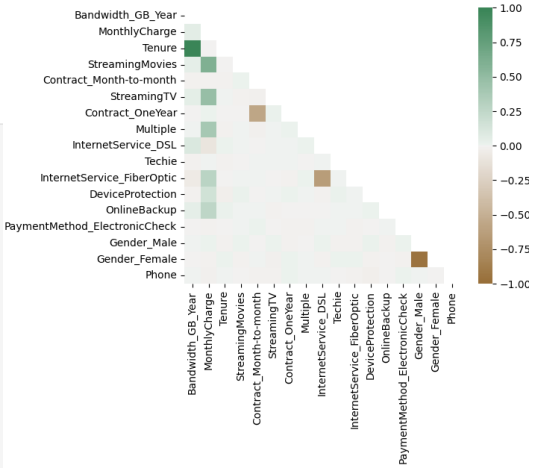
```
21               Bandwidth_GB_Year
20                   MonthlyCharge
19                          Tenure
17                  StreamingMovies
30         Contract_Month-to-month
16                      StreamingTV
31                Contract_OneYear
11                        Multiple
22              InternetService_DSL
7                           Techie
23       InternetService_FiberOptic
14                DeviceProtection
13                    OnlineBackup
34    PaymentMethod_ElectronicCheck
29                     Gender_Male
28                   Gender_Female
10                           Phone
```

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_2.corr()


mask = np.triu(np.ones_like(df_2.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_knn_classification_D209 PA1.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_knn_classification.csv' and 'testing_dataset_knn_classification.csv'.

```
# train test split

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

```
# 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_knn_classification.csv', index=False)
test_df.to_csv('testing_dataset_knn_classification.csv', index=False)
```

# D2

After preprocessing the data, I analyzed the data using kNN classification. kNN, a machine learning method used for a classification problem, works in the following way. Given an unlabeled data point, kNN predicts its label by looking at the k closest labeled data points and taking a majority vote (Muller, nd). The entire data set is split into training and testing sets. "During the training phase, the kNN algorithm stores the entire training dataset as a reference. When making predictions, it calculates the distance between the input data point and all the training examples, using a chosen distance metric such as Euclidean distance." (Srivastava, 2023). The k closest neighbors are identified and the data point that we're seeking a label for is assigned the most common label of its k neighbors. When performing a GridSearchCV to find the optimal number of k, I obtained $k = 16$.

The data analysis technique I used, along with intermediate calculations, and described here.

Define the feature variables as $X$ and the target variable as $y$. The selected features from SelecKBest are the feature variables, defined as $X$ and the target variable, "Churn", is defined as $y$. These sets were then split into training and testing sets.

```
# Defining X = features, y=target

X = df_2      #df_2 does not contain Churn data
y = df["Churn"]   #df contains churn data
print(X.shape)
print(y.shape)
```

```
(9992, 17)
(9992,)
```

```
# train test split

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

Use a pipeline to build the model.

First, since the dataset contains both numeric and categorical data, we only scale the numeric data types. So, a ColumnTransformer was used to scale only specific columns in the pipeline, namely “Bandwidth_GB_Year”, “Tenure”, and “MonthlyCharge”.

```
#select numeric and categorical data types. only scale numeric types

numeric_columns = list(X.select_dtypes('float64').columns)
categorical_columns = list(X.select_dtypes('int64').columns)
```

```
preprocessor = ColumnTransformer(
    transformers=[('num', StandardScaler(), numeric_columns)],
    remainder='passthrough')                #keep non-numeric columns unchanged

# pipeline with preprocessor and Knn classifier
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('knn', KNeighborsClassifier())
])
```

Next, hyperparameters were defined and GridSearchCV was instantiated to find the best $k$ nearest neighbors ($k = 16$).

```
# define hyperparameters for GridSearchCV
parameters = {'knn__n_neighbors': np.arange(1, 25)}

# GridSearchCV object
knncv = GridSearchCV(estimator=pipeline,
                     param_grid=parameters,
                     n_jobs=-1,
                     cv=5,
                     scoring='accuracy')
```

Now, we fit the model on the training set and obtain the training accuracy score.

```
# Fit the model
knncv.fit(X_train, y_train)

#Print best params and best score
print("Best Parameters: ", knncv.best_params_)
print("Best CV Score: ", knncv.best_score_)
```

Lastly, predict on the testing set and obtain the accuracy score.

```
# Predict on test set
y_pred = knncv.predict(X_test)
print("Accuracy score for test set: ", accuracy_score(y_test, y_pred))

# Evaluate the model on test set
test_accuracy = knncv.score(X_test, y_test)
print("Test Set Accuracy: ", test_accuracy)
```

Calculate metrics on test set.

The accuracy score for the training and testing sets are high, 89% and 88% respectively.

```
# accuracy on Train
print('Training Accuracy: ', knncv.best_score_)

# accuracy on Test
print('Testing Accuracy: ', knncv.score(X_test, y_test))
```

```
Training Accuracy:  0.8860264667920579
Testing Accuracy:  0.8769384692346173
```

I calculated the Area Under the Curve (AUC) value, obtained the confusion matrix, and a classification report including precision, recall, and f1-score.

```
#AUC (area under the curve)

print("AUC")
y_pred_probability = knncv.predict_proba(X_test)[:,1]
roc_auc_score(y_test, y_pred_probability)
```

```
AUC
0.9316811590479983
```

```
#confusion matrix

print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[1381   88]
 [ 158  372]]
```

```
print(classification_report(y_test, y_pred))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.90 | 0.94 | 0.92 | 1469 |
| 1 | 0.81 | 0.70 | 0.75 | 530 |
| accuracy | | | 0.88 | 1999 |
| macro avg | 0.85 | 0.82 | 0.83 | 1999 |
| weighted avg | 0.87 | 0.88 | 0.87 | 1999 |

# D3

The code used to perform the classification analysis explained in part D2 is included here.

- Define the feature variables as $X$ and the target variable as $y$.

```
from sklearn.preprocessing import scale      #scale data values
from sklearn.preprocessing import StandardScaler     #scale data values

from sklearn.model_selection import train_test_split     #split data into training set and testing set
from sklearn.neighbors import KNeighborsClassifier    #KNN classifier

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV    #GridSearchCV tind optimal k

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
```

```
# Defining X = features, y=target

X = df_2      #df_2 does not contain Churn data
y = df["Churn"]   #df contains churn data
print(X.shape)
print(y.shape)
```

```
(9992, 17)
(9992,)
```

- Use a pipeline to build the model. Predict on the testing set.

```
numeric_columns = list(X.select_dtypes('float64').columns)
categorical_columns = list(X.select_dtypes('int64').columns)
```

```
preprocessor = ColumnTransformer(
    transformers=[('num', StandardScaler(), numeric_columns)],
    remainder='passthrough')                #keep non-numeric columns unchanged

# pipeline with preprocessor and Knn classifier
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('knn', KNeighborsClassifier())
])

# define hyperparameters for GridSearchCV
parameters = {'knn__n_neighbors': np.arange(1, 25)}

# GridSearchCV object
knncv = GridSearchCV(estimator=pipeline,
                     param_grid=parameters,
                     n_jobs=-1,
                     cv=5,
                     scoring='accuracy')

# Fit the model
knncv.fit(X_train, y_train)

#Print best params and best score
print("Best Parameters: ", knncv.best_params_)
print("Best CV Score: ", knncv.best_score_)


# Predict on test set
y_pred = knncv.predict(X_test)
print("Accuracy score for test set: ", accuracy_score(y_test, y_pred))

# Evaluate the model on test set
test_accuracy = knncv.score(X_test, y_test)
print("Test Set Accuracy: ", test_accuracy)
```

```
Best Parameters:  {'knn__n_neighbors': 16}
Best CV Score:  0.8860264667920579
Accuracy score for test set:  0.8769384692346173
```

- Metrics. I calculated accuracy scores for training and testing data, the Area Under the Curve (AUC) value, obtained the confusion matrix, and a classification report including precision, recall, and f1-score.

```
# accuracy on Train
print('Training Accuracy: ', knncv.best_score_)

# accuracy on Test
print('Testing Accuracy: ', knncv.score(X_test, y_test))
```

```
Training Accuracy:  0.8860264667920579
Testing Accuracy:  0.8769384692346173
```

```
#AUC (area under the curve)

print("AUC")
y_pred_probability = knncv.predict_proba(X_test)[:,1]
roc_auc_score(y_test, y_pred_probability)
```

```
AUC
0.9316811590479983
```

```
#confusion matrix

print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[1381   88]
 [ 158  372]]
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.94      0.92      1469
           1       0.81      0.70      0.75       530

    accuracy                           0.88      1999
   macro avg       0.85      0.82      0.83      1999
weighted avg       0.87      0.88      0.87      1999
```

# E1

Accuracy is a metric we can use for kNN classification. It is the percent of correct classifications out of the total samples (Muller, nd). The accuracy score for the training set is about 89% and 88% for the testing. This model has a high accuracy, correctly classifiing customer churn on 88% of the testing set.

Area Under the Curve, AUC, represents the model's ability to discriminate between positive and negative classes. It is the area under the Receiver Operator Characteristic, ROC, curve. The ROC curve is an evaluation metric for binary classification problems. It plots the true positive rate against the false positive rate at various threshold values. AUC ranges in value from 0 to 1. AUC of 1 means the model made all predictions correctly. An area of 0.5 means the model is as good as random guessing (Elleh, nd). The AUC for this analysis is about 93%, indicating the model is effective at distinguishing between a "churn" and a "did not churn" label.

# E2

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

The best features to use in the kNN classifcation analysis were selected using SelectKBest. They are:

> *"Bandwidth_GB_Year", "MonthlyCharge", "Tenure", "StremingMovies", "Contract_Month-to-month", "StreamingTV", "Contract_OneYear", "Multiple", "InternetService_DSL", "Techie", "InternetService_FiberOptic", "DeviceProtection", "OnlineBackup", "PaymentMethod_ElectronicCheck", "Gender_Male", "Gender_Female", and "Phone"*.

The implication is that these features are influential factors in predicting customer churn and the company should use these factors to retain customers.

The kNN classification model has a high accuracy score on the testing data, 88% (89% on the training data). Also, the high AUC score of 93% indicates that the model is effective at distinguishing between "churn" and "did not churn" labels. We can be fairly accurate in our predictions of customer churn. GridSearchCV to find the optimal k value of 16, but the model could be improved with further hyperparameter tuning and with more observations gathered.

# E3

One limitation of this is analysis is data imbalance. Out of the 10,000 observations in the dataset, 7,350 of these are for customers who "did not churn" and thus 2,650 are for customers who did "churn". Even though I stratified the proportions of churn and did not churn when fitting the model, the data imbalance could lead to misclassifying the less common class of "did not churn" (Elleh, nd).

## E4

Based on the analysis, a recommend course of action is that the company can use this model to predict customers who are at risk of churn.

The most important features influencing customer churn are *"Bandwidth_GB_Year", "MonthlyCharge", "Tenure", "StremingMovies", "Contract_Month-to-month", "StreamingTV", "Contract_OneYear", "Multiple", "InternetService_DSL", "Techie", "InternetService_FiberOptic", "DeviceProtection", "OnlineBackup", "PaymentMethod_ElectronicCheck", "Gender_Male", "Gender_Female", and "Phone".* The company should use these factors to retain customers. For example, contract type. Customers who are on a month-to-month contract have higher churn rates than customers who do not (Elleh, nd). The company should work to secure one year or two year contracts with customers.

## 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. "D209 Webinar: Task 1 Expectations and Data Preprocessing-Python." Data Mining I – D209, nd, wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=7329050b-3de4-412a-9ff4-b0790009d02b

Elleh, Festus. "D209 Task 1 Splitting the Data and Creating the Model-Python." Data Mining I – D209, nd, https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=77e354f7-ed96-4886-9a44-b07a00d44a31

Muller, Andreas, et. al. "Machine Learning with scikit-learn." Datacamp, app.datacamp.com/learn/courses/machine-learning-with-scikit-learn

# H

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

Elleh, Festus. “D209 Webinar: Task 1 Expectations and Data Preprocessing-Python.” Data Mining I – D209, nd, wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=7329050b-3de4-412a-9ff4-b0790009d02b

Elleh, Festus. “D209 Task 1 Splitting the Data and Creating the Model-Python.” Data Mining I – D209, nd, https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=77e354f7-ed96-4886-9a44-b07a00d44a31

Muller, Andreas, et. al. “Machine Learning with scikit-learn.” Datacamp, app.datacamp.com/learn/courses/machine-learning-with-scikit-learn

Srivastava, Tavish. “A Complete Guide to K-Nearest Neighbors (Updated 2023).” Analytics Vidhya, October 20, 2023, analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

# I

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