D209 – Data Mining (Task 1)

Morrell J. Parrish

Western Governors University

Table of Contents

D209 – Data Mining (Task 1)

A. Research Question

During this course of research, we will determine which customers are at a higher risk of churn and which features (variables) can be an indicator for churn.

A2. Analysis Goal

The objective of this analysis is to use data classification to determine which customers, new or existing, are at a higher risk for churn based on the similarities to other customers with similar features that may have or have not churned in the past. “The churn rate, also known as the rate of attrition or customer churn; is the frequency in which consumers discontinue doing business with a company. It is commonly represented as the percentage of service subscribers who cancel their memberships within a specified time frame” (Frankenfield, 2022).

B. Chosen Technique

The ***k*-nearest neighbors** classification method will be used for this analysis. ***K*-nearest neighbors** works on the principle that every data point that is close to another is in the same class. In other words, it uses similarity to classify a new data point (Dwivedi, *nd*)
The expected outcome of this classification analysis should show the targeted variable (churn) and the relationship between the nearing ***k*-neighbors**.

B2. Assumption

The primary assumption of a **KNN** model is that data points/instances that are close to each other are highly similar, whereas data points that are far away from another group are dissimilar to those data points. The distance between two points on a graph is used by a **KNN** model to calculate similarity. The farther apart the points are, the less similar they are. There are several methods for calculating distance between points, but the most commonly used distance metric is Euclidean distance (the distance between two points in a straight line) (Nelson, 2020).

B3. Packages/Libraries Use Justification

The following packages/libraries will be used for this analysis:

- Pandas
    - used to read and manipulate data via series (one-dimensional structure) or dataframes (multi-dimensional data structure)
- NumPy
    - used to perform mathematical computations
- Matplotlib
    - used to create visualization (plotting and graphing)
- Seaborn
    - used to create visualization (plotting and graphing)
- Scikit-learn
    - used to perform scientific computations
    - used to split our data into training and test sets
    - used for predicting and classification analysis

C. Data Preparation Description

To use the churn dataset in our analysis we will first need to prepare the data.

The following steps were taken to prepare the dataset for analysis:

- download the churn dataset
- determine which variables will be used in the analysis
- import the dataset into *PyCharm*

- remove independent variables, demographics, and personal identification variables not being used in the analysis
  - caseorder, customer_id, interaction, UID, city, state, county, zip, lat, lng, population, timezone, job, email, contacts
- determine if any outliners exist and remove them

C2. Variable Identification and Classification

The **continuous variables** (16) that will be used in this analysis will include age, children, income**,** outage_sec_perweek, yearly_equip_failure, tenure, monthlycharge, **b**andwidth_GB_Year, item1 (timelyresponse), item2 (fixes), item3 (replacements), item4 (reliability), item5 (options), item6 (respectfulness), item7 (courteous), and item8 (listening).

The **categorical variables** (19) that will be used in this analysis will include area, marital, gender, churn, techie, contract, portmodem, tablet, internetservice, phone, multiple, onlinesecurity, onlinebackup, deviceprotection, techsupport, streamingtv, streamingmovies, paperlessbilling, paymentmethod.

C3. Data Preparation Steps

To use the churn dataset in our analysis we will first need to prepare the data:

- import the dataset into *Python (PyCharm)*
- view the dataframe's description, structure, and data types
- view summary statistics
- evaluate the dataset, remove null or missing values
- remove any outliners
- remove demographics, and personal identification

- caseorder, customer_id, interaction, UID, city, state, county, zip, lat, lng, population, area, timezone, job, email, contacts

▪ convert binomial variables (yes/no to 1 and 0) to numerical variables

The below code was used to prepare our data:

```
# Standard data science imports
import NumPy as np
import pandas as pd
from pandas import Series, DataFrame

# Visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Scikit-learn
import sklearn
from sklearn import datasets
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report

# Ignore Warning Code
import warnings
warnings.filterwarnings('ignore')

# Load data set into Pandas dataframe
df = pd.read_csv('churn_clean.csv')

# Remove less meaningful demographic variables
df = df.drop(columns=['CaseOrder', 'Customer_id', 'Interaction', 'UID',
'City', 'State', 'County', 'Zip', 'Lat', 'Lng', 'Population', 'TimeZone',
'Email', 'Contacts', 'Job'])

# Display Churn dataframe
print(df)

# Rename last 8 columns
df.rename(columns={'Item1': 'TimelyResponse', 'Item2': 'Fixes', 'Item3':
'Replacements', 'Item4': 'Reliability', 'Item5': 'Options', 'Item6':
'Respectfulness', 'Item7': 'Courteous', 'Item8': 'Listening'}, inplace=True)

# Get column info
print(df.info())

# Describe Churn dataset
print(df.describe())

# Convert binary variables (yes/no, female/male) to 0 or 1
df['DmyGender'] = [1 if v == 'Male' else 0 for v in df['Gender']]
df['DmyChurn'] = [1 if v == 'Yes' else 0 for v in df['Churn']]
df['DmyTechie'] = [1 if v == 'Yes' else 0 for v in df['Techie']]
```

```
df['DmyContract'] = [1 if v == 'Two Year' else 0 for v in df['Contract']]
df['DmyPort_modem'] = [1 if v == 'Yes' else 0 for v in df['Port_modem']]
df['DmyTablet'] = [1 if v == 'Yes' else 0 for v in df['Tablet']]
df['DmyInternetService'] = [1 if v == 'Fiber Optic' else 0 for v in df['InternetService']]
df['DmyPhone'] = [1 if v == 'Yes' else 0 for v in df['Phone']]
df['DmyMultiple'] = [1 if v == 'Yes' else 0 for v in df['Multiple']]
df['DmyOnlineSecurity'] = [1 if v == 'Yes' else 0 for v in df['OnlineSecurity']]
df['DmyOnlineBackup'] = [1 if v == 'Yes' else 0 for v in df['OnlineBackup']]
df['DmyDeviceProtection'] = [1 if v == 'Yes' else 0 for v in df['DeviceProtection']]
df['DmyTechSupport'] = [1 if v == 'Yes' else 0 for v in df['TechSupport']]
df['DmyStreamingTV'] = [1 if v == 'Yes' else 0 for v in df['StreamingTV']]
df['DmyStreamingMovies'] = [1 if v == 'Yes' else 0 for v in df['StreamingMovies']]
df['DmyPaperlessBilling'] = [1 if v == 'Yes' else 0 for v in df['PaperlessBilling']]

# Drop original categories
df2 = df.drop(columns=['Gender', 'Churn', 'Techie', 'Contract', 'Port_modem',
'Tablet', 'InternetService', 'Phone','Multiple', 'OnlineSecurity',
'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
'StreamingMovies', 'PaperlessBilling'])

print(df2.describe())
```

C4. Cleaned Data Set

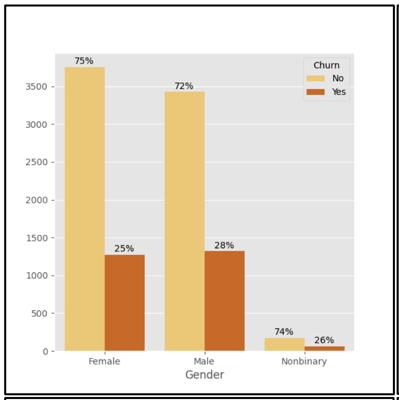
The prepared dataset used for this analysis has been uploaded with the assessment file.

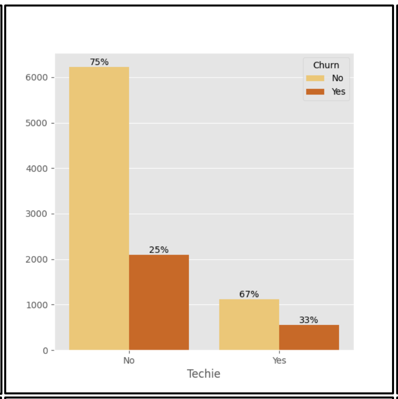
D. Analysis

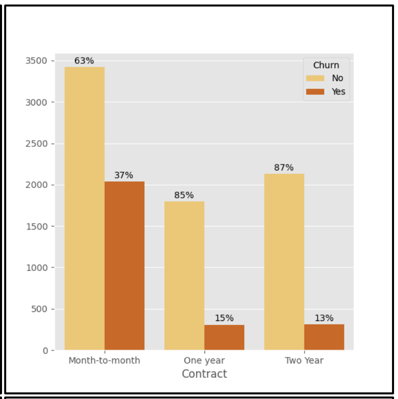
The training and test datasets used for this analysis have been uploaded with the assessment file.

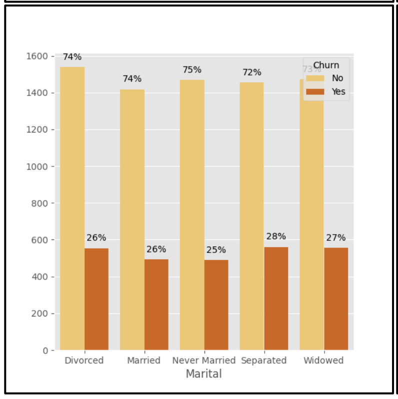
D2. Analytical Technique Description

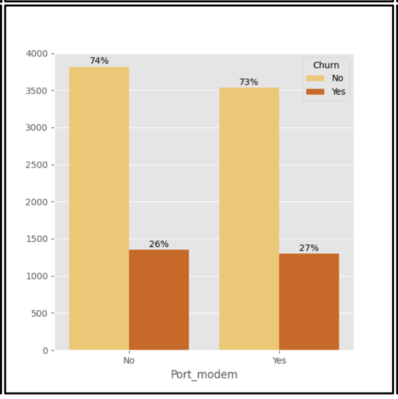
Our analytical technique includes the following steps: (1) read in or load the data using Pandas' ***read( )*** function – in this case it will be our cleaned churn data set, (2) check to make sure the data was read in properly using Pandas' ***head( )*** function, (3) verify the shape of the data using Pandas' '***shape'*** function, (4) perform exploratory analysis by creating a series of visual (boxplots, , scatterplots, correlation matrix, and heatmaps etc.…), (5) identify and remove any outliners, (6) split up the dataset into inputs (X) and our target variable (y) using Pandas' ***drop( )*** function – this allows you to drop the target variable from the dataframe and store it in the variable 'X', (7) verify the data again using Pandas' ***head ( )*** function, (8) split the dataset into training and test sets using Scikit-learn's function '***train_test_split',*** (9) create/build the initial model using Scikit-learn's function ***'KNeighborsClassifier'***, (10) train model - this can be done using the KNN's ***fit*** ( ), (11) once trained we can now test the model using KNN's ***predict ( )*** function, (12) verify model's accuracy on the test data using KNN's ***score*** function, (13) perform ***k-Fold Cross Validation*** – splits the data set into 'k', one group is used as the test set and the others are used as training sets – recording the accuracy score in an array for each test, (14) find the average of k-Fold Validation tests by using Numpy's ***mean ( )*** function, passing in the 'cv_score' from the validation test, (15) perform hypertuning parameters using Scikit-learn's function ***GridSearchCV ( )***, (16) check/verify after training which value for 'n_neighbor' did the best by calling '***best_params_***' on the model, (17) after finding the optimal value, I used the '***best_score_'*** function to check the accuracy of the model.

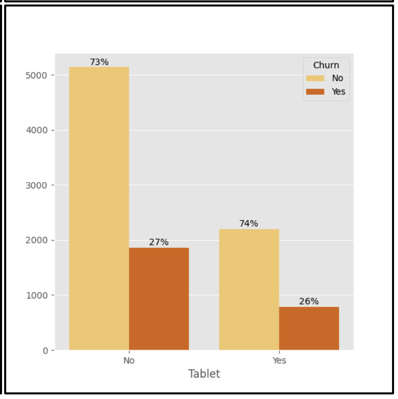












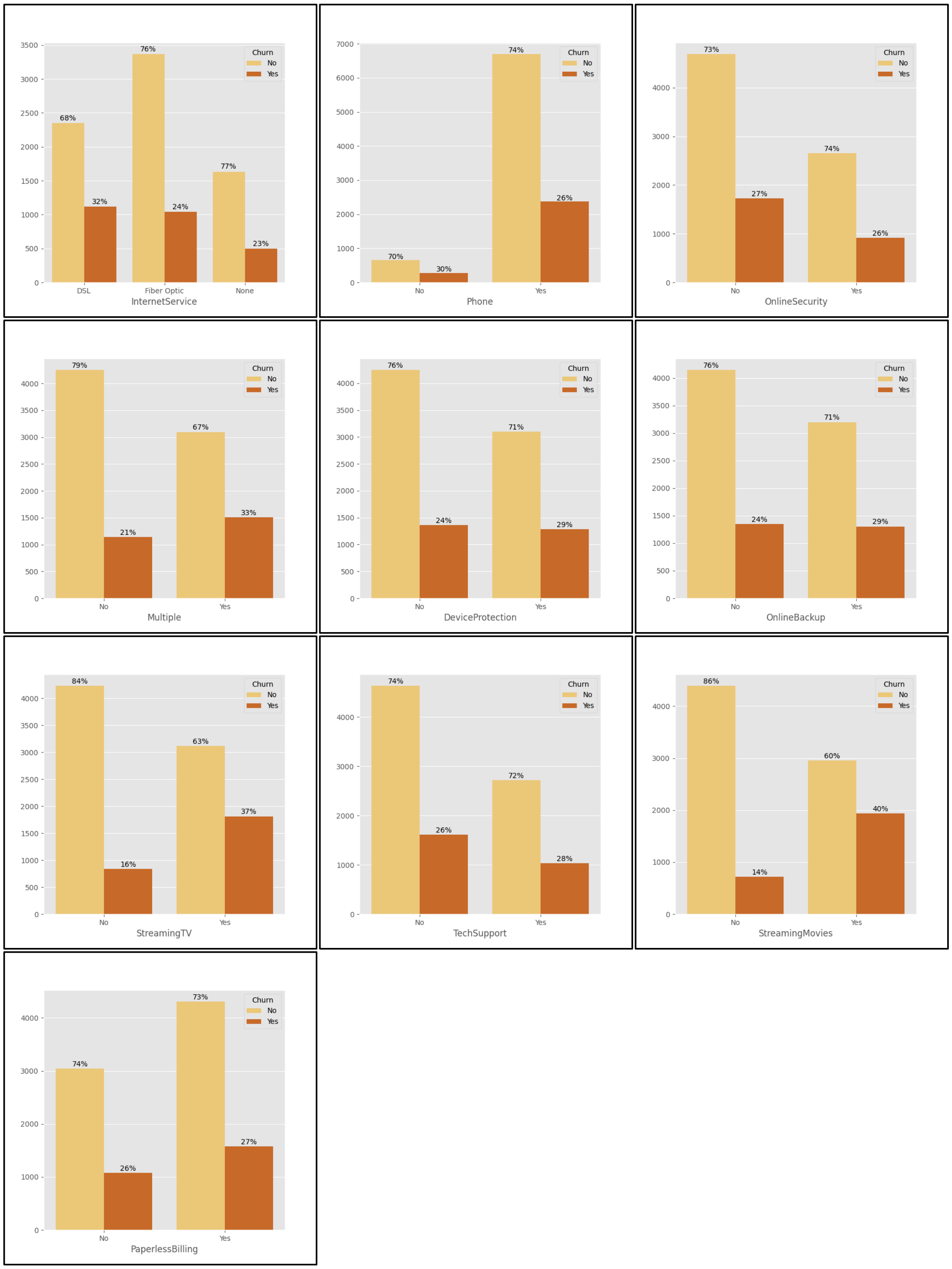
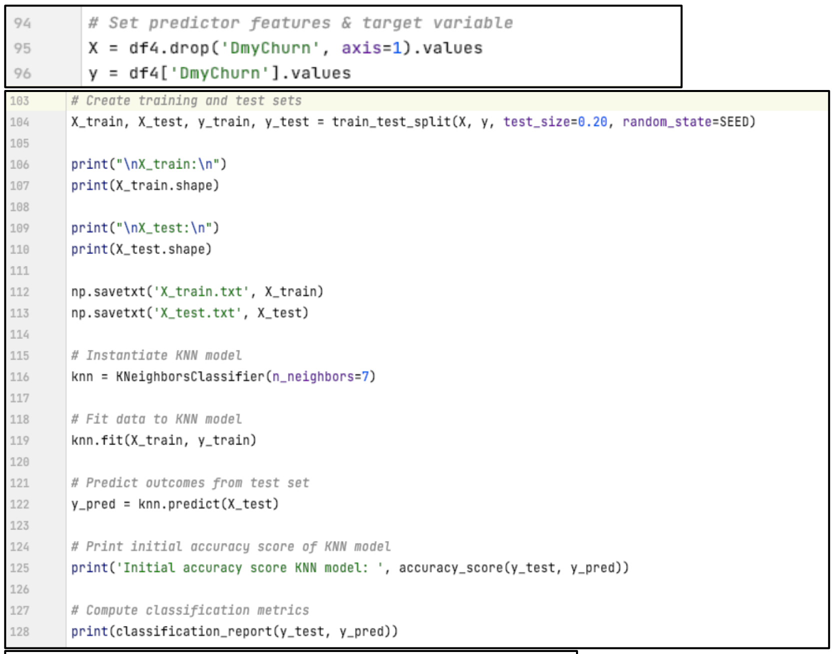
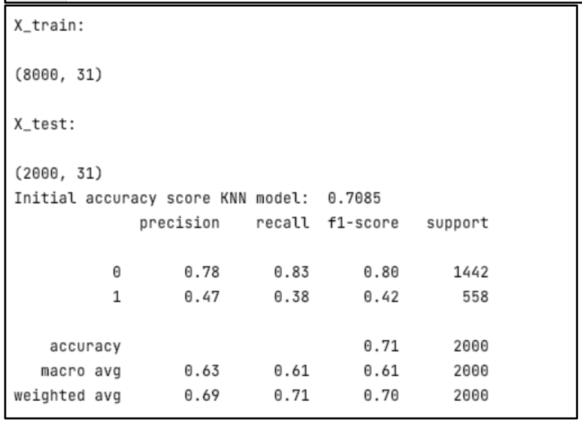


***Figure 1**: Churn and Categorical Variables*



```
# Set predictor features & target variable
X = df4.drop('DmyChurn', axis=1).values
y = df4['DmyChurn'].values
```

```
# Create training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=SEED)

print("\nX_train:\n")
print(X_train.shape)

print("\nX_test:\n")
print(X_test.shape)

np.savetxt('X_train.txt', X_train)
np.savetxt('X_test.txt', X_test)

# Instantiate KNN model
knn = KNeighborsClassifier(n_neighbors=7)

# Fit data to KNN model
knn.fit(X_train, y_train)

# Predict outcomes from test set
y_pred = knn.predict(X_test)

# Print initial accuracy score of KNN model
print('Initial accuracy score KNN model: ', accuracy_score(y_test, y_pred))

# Compute classification metrics
print(classification_report(y_test, y_pred))
```



```
X_train:

(8000, 31)

X_test:

(2000, 31)
Initial accuracy score KNN model:  0.7085
              precision    recall  f1-score   support

           0       0.78      0.83      0.80      1442
           1       0.47      0.38      0.42       558

    accuracy                           0.71      2000
   macro avg       0.63      0.61      0.61      2000
weighted avg       0.69      0.71      0.70      2000
```

Figure 2*: Output and Intermediate Calculations*

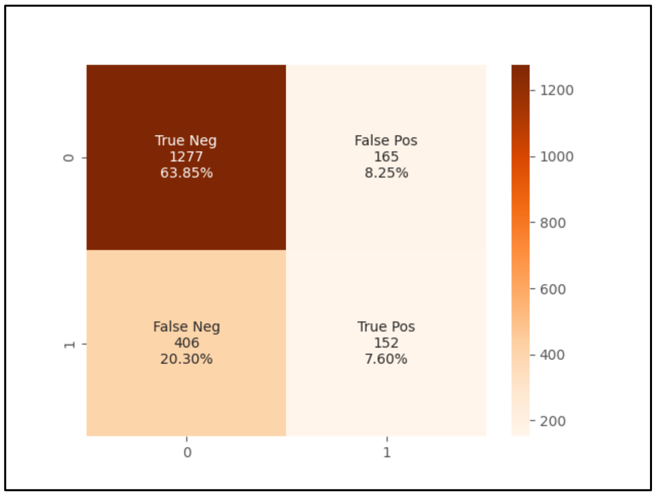
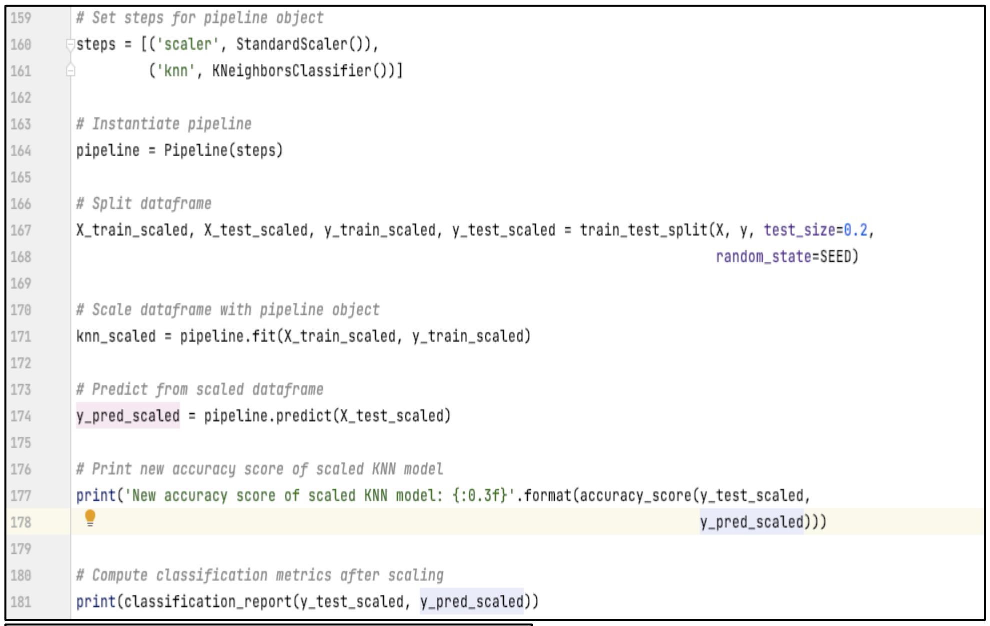
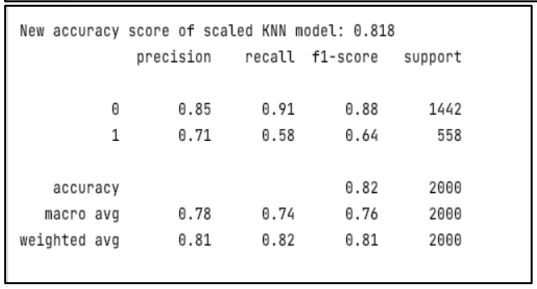


Figure 3*: Confusion Matrix*

According to the classification report our intermediate calculations yielded that 47% of the customers predicted to churn did so. The model also only correctly predicted this outcome for 38% of those customers; the model's accuracy score is about 71%.



```
# Set steps for pipeline object
steps = [('scaler', StandardScaler()),
         ('knn', KNeighborsClassifier())]

# Instantiate pipeline
pipeline = Pipeline(steps)

# Split dataframe
X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = train_test_split(X, y, test_size=0.2,
                                                                                random_state=SEED)

# Scale dataframe with pipeline object
knn_scaled = pipeline.fit(X_train_scaled, y_train_scaled)

# Predict from scaled dataframe
y_pred_scaled = pipeline.predict(X_test_scaled)

# Print new accuracy score of scaled KNN model
print('New accuracy score of scaled KNN model: {:0.3f}'.format(accuracy_score(y_test_scaled,
                                                                              y_pred_scaled)))

# Compute classification metrics after scaling
print(classification_report(y_test_scaled, y_pred_scaled))
```



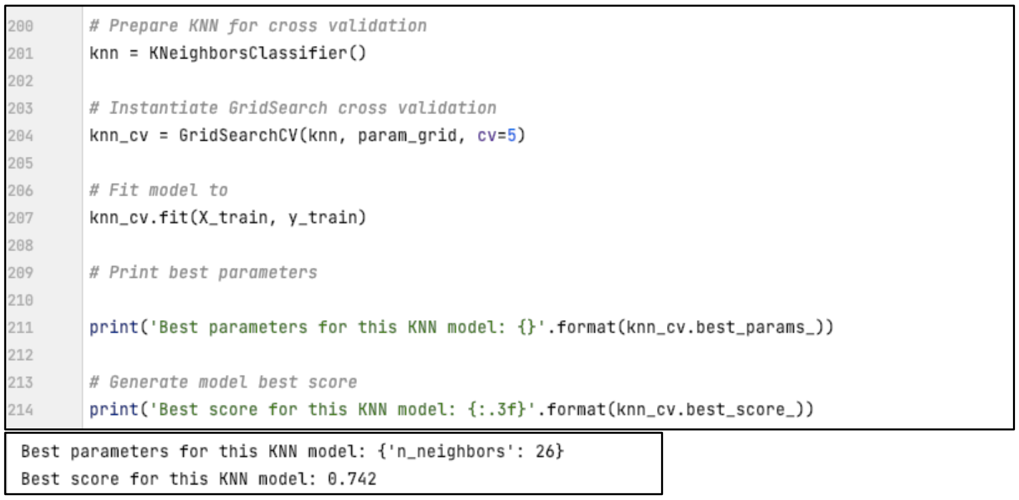
```
New accuracy score of scaled KNN model: 0.818
              precision    recall  f1-score   support

           0       0.85      0.91      0.88      1442
           1       0.71      0.58      0.64       558

    accuracy                           0.82      2000
   macro avg       0.78      0.74      0.76      2000
weighted avg       0.81      0.82      0.81      2000
```

I performed some hypertuning parameters on my initial model using Skicit-learn's ***GridSearchCV( )*** function; this allowed our model to be trained (multiple times) on a range of specified parameters; after hypertuning our model our retrained scaled model calculations yielded that 71% of the customers predicted to churn did so. The model also only correctly predicted this outcome for 58% of those customers; the model's accuracy score is about 82%; the models accuracy increased by 11%.

D3. Classification Analysis Code

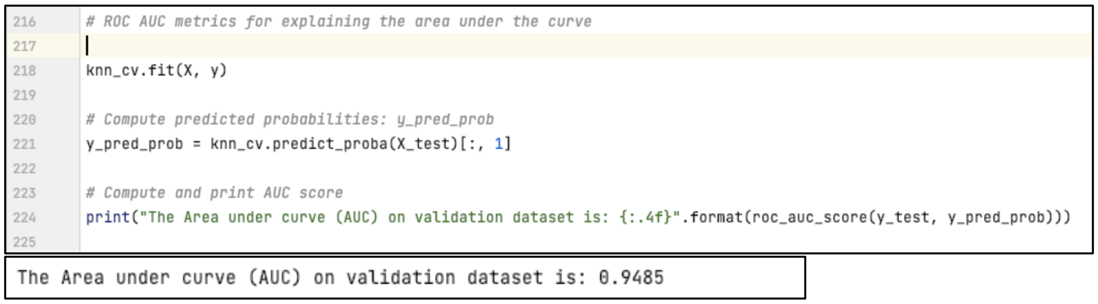
```
# Set steps for pipeline object
steps = [('scaler', StandardScaler()),
         ('knn', KNeighborsClassifier())]

# Instantiate pipeline
pipeline = Pipeline(steps)

# Split dataframe
X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled =
train_test_split(X, y, test_size=0.2, random_state=SEED)

# Scale dataframe with pipeline object
knn_scaled = pipeline.fit(X_train_scaled, y_train_scaled)

# Predict from scaled dataframe
y_pred_scaled = pipeline.predict(X_test_scaled)

# Print new accuracy score of scaled KNN model
print('New accuracy score of scaled KNN model:
{:0.3f}'.format(accuracy_score(y_test_scaled, y_pred_scaled)))

# Compute classification metrics after scaling
print(classification_report(y_test_scaled, y_pred_scaled))

# Create confusion_matrix & generate results
cf_matrix = confusion_matrix(y_test, y_pred)
print(cf_matrix)

# Generate confusion matrix visual
group_names = ['True Neg', 'False Pos', 'False Neg', 'True Pos']
group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten()
/ np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names,
group_counts, group_percentages)]
labels = np.asarray(labels).reshape(2, 2)

sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Oranges')
plt.show()

# Set up parameters grid
param_grid = {'n_neighbors': np.arange(1, 50)}

# Prepare KNN for cross validation
knn = KNeighborsClassifier()

# Instantiate GridSearch cross validation
knn_cv = GridSearchCV(knn, param_grid, cv=5)

# Fit model to
knn_cv.fit(X_train, y_train)

# Print best parameters
print('Best parameters for this KNN model: {}'.format(knn_cv.best_params_))

# Generate model best score
print('Best score for this KNN model: {:.3f}'.format(knn_cv.best_score_))

# ROC AUC metrics for explaining the area under the curve
# Fit it to the data
knn_cv.fit(X, y)

# Compute predicted probabilities: y_pred_prob
y_pred_prob = knn_cv.predict_proba(X_test)[:, 1]

# Compute and print AUC score
print("The Area under curve (AUC) on validation dataset is:
{:.4f}".format(roc_auc_score(y_test, y_pred_prob)))

# Compute cross-validated AUC scores: cv_auc
cv_auc = cross_val_score(knn_cv, X, y, cv=5, scoring='roc_auc')

# Print list of AUC scores
print("AUC scores computed using 5-fold cross-validation: {}".format(cv_auc))
```

E. Summary



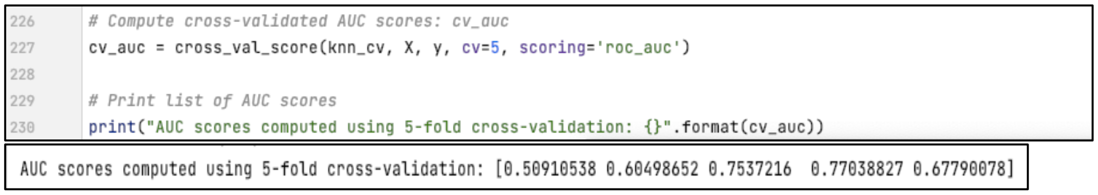
```
# Prepare KNN for cross validation
knn = KNeighborsClassifier()

# Instantiate GridSearch cross validation
knn_cv = GridSearchCV(knn, param_grid, cv=5)

# Fit model to
knn_cv.fit(X_train, y_train)

# Print best parameters

print('Best parameters for this KNN model: {}'.format(knn_cv.best_params_))

# Generate model best score
print('Best score for this KNN model: {:.3f}'.format(knn_cv.best_score_))
```

```
Best parameters for this KNN model: {'n_neighbors': 26}
Best score for this KNN model: 0.742
```



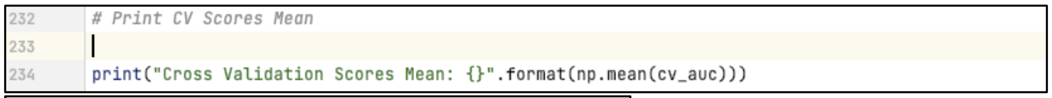
```
# ROC AUC metrics for explaining the area under the curve

knn_cv.fit(X, y)

# Compute predicted probabilities: y_pred_prob
y_pred_prob = knn_cv.predict_proba(X_test)[:, 1]

# Compute and print AUC score
print("The Area under curve (AUC) on validation dataset is: {:.4f}".format(roc_auc_score(y_test, y_pred_prob)))
```

```
The Area under curve (AUC) on validation dataset is: 0.9485
```

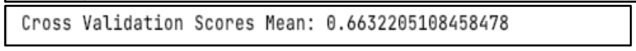


```
# Compute cross-validated AUC scores: cv_auc
cv_auc = cross_val_score(knn_cv, X, y, cv=5, scoring='roc_auc')

# Print list of AUC scores
print("AUC scores computed using 5-fold cross-validation: {}".format(cv_auc))
```

```
AUC scores computed using 5-fold cross-validation: [0.50910538 0.60498652 0.7537216  0.77038827 0.67790078]
```



```
# Print CV Scores Mean

print("Cross Validation Scores Mean: {}".format(np.mean(cv_auc)))
```



```
Cross Validation Scores Mean: 0.6632205108458478
```

AUC values range from 0 to 1; the AUC for our trained model is 0.9485 or 95%. A model with 100% incorrect predictions has an AUC of 0.0; one with 100% correct predictions has an AUC of 1.0; the greater the AUC, the better the model's performance in distinguishing between positive and negative classes.

E2. Classification Analysis Results and Implications

The below chart summarizes/compares our intermediate and scaled models. The scaled modeled improved the accuracy of predictions by 11%; the AUC of the scaled model signifies that it performs significantly greater at distinguishing between positive and negative classes.

	Intermediate Model	Scaled Model	-/+
Precision	47%	71%	+ 24%
Recall	38%	58%	+ 20%
Accuracy	71%	82%	+ 11%
CV mean		66%	
AUC		95%	

E3. Analysis Limitation

Although the scaled model's accuracy increased by 11%, it still does not indicate what commonalities exist between the churn customers; what services they shared; what their customer satisfaction ratings were; and how long they were customers before churning.

E4. Recommended course of Action

A recommended course of action would be to conduct additional analysis into each of the variables for those churn customers to find commonalities and trends; and it could be a combination of variables (multiple services) that will indicate whether a customer (new or existing) will churn; this further in-dept analysis will provide management with more hindsight and direction when making decisions on what services to offer, contracts (tenure), and pricing.

G. Panopto video recording

VideoLink

References

Dwivedi, R. (n.d.). How does K-nearest neighbor works in Machine Learning: KNN algorithm. How does K-nearest Neighbor Works in Machine Learning | KNN algorithm. Retrieved July 21, 2022, from https://www.analyticssteps.com/blogs/how-does-k-nearest-neighbor-works-machine-learning-classification-problem

Google. (n.d.). Classification: Roc curve and AUC | Machine Learning | Google Developers. Google. Retrieved July 23, 2022, from https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

Nelson, D. (2020, August 23). What is a KNN (K-nearest neighbors)? Unite.AI. Retrieved July 21, 2022, from https://www.unite.ai/what-is-k-nearest-neighbors/#:~:text=The%20primary%20assumption%20that%20a,two%20points%20on%20a%20graph.

WGU. (n.d.). NVM2 TASK 1: Classification Analysis. WGU Performance Assessment. Retrieved July 23, 2022, from https://tasks.wgu.edu/student/000194226/course/20900018/task/2807/overview