D603 – Machine Learning

Performance Assessment #1 – Classification Data Mining Models

Bernard Connelly

Master of Science, Data Analytics, Western Governors University

Dr. Sherin Aly

March 12, 2025

Performance Assessment #1 – Classification Data Mining Models

Part 1: Purpose & Justification

B1. Business Proposal

Does a patient’s health metrics and services received post-admission impact re-admission rates to the hospital?

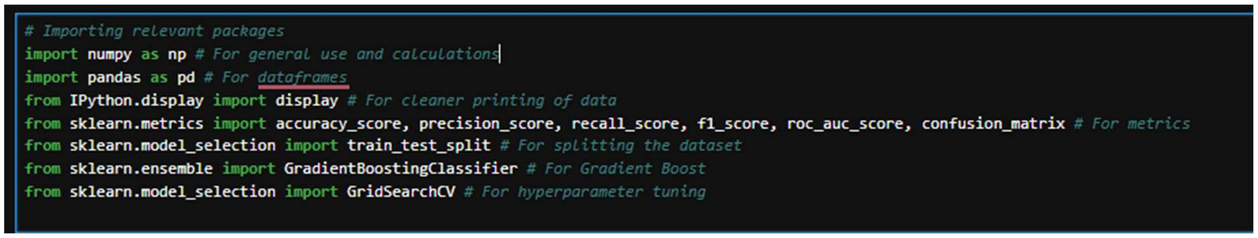
B2. Goal of Analysis

The goal will be to determine if the health conditions, choices, and care received upon admission to the hospital impact whether or not a patient is re-admitted within a month to the hospital. The goal will be to remove demographic and financial factors to focus on the effects a patient's overall health, treatment received, personal choices, and specific details surrounding their initial admission have on re-admission rates.

C1. Explanation of Classification Method

Gradient boost is an effective classification method for this project as it fits various potential needs that align with this dataset. First, it works well with datasets of moderate size, and 10,000 observations fit categorization. Gradient Boost also works well with mixed datatypes, of which there is a variety in the columns maintained for the dataset. Medical data is also prone to missing values or inconsistencies, which Gradient Boost can handle well. Finally, Gradient Boost’s methodology helps identify the strongest predictors of a dependent variable, which, in this case, is re-admission – a primary concern of the hospital industry.

C2. Packages & Libraries Used



```
# Importing relevant packages
import numpy as np # For general use and calculations
import pandas as pd # For dataframes
from IPython.display import display # For cleaner printing of data
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix # For metrics
from sklearn.model_selection import train_test_split # For splitting the dataset
from sklearn.ensemble import GradientBoostingClassifier # For Gradient Boost
from sklearn.model_selection import GridSearchCV # For hyperparameter tuning
```

Numpy and Pandas were imported for general calculations in Python and dataframes respectively. IPython.display was imported as an alternative to using the print command to export data or multiple lines of code. The sklearn.metrics package imported the various metrics required for the task, including accuracy, precision, f1 score, roc-auc score, and the confusion matrix. Sklearn.model_selection test_train_split was utilized to split the dataset for the model. Sklearn.ensemble GradientBoostClassifer was imported to perform Gradient Boost. GridSearchCV was imported from sklearn.model_selection for hyperparameter tuning.

Part 2: Data Preparation

D1. Preprocessing Goal

All relevant categorical variables were encoded via one-hot encoding, and the ordinal variables were changed via ordinal encoding so they would be reflected with numerical inputs. The encoding was used to ensure the data was adequately prepared for Gradient Boosting. This is important to avoid errors and improve model interpretability.

D2. Identification & Classification of Variables for Analysis

The breakdown of the datatypes remaining for analysis is as seen below.

Categorical Variables:

'readmis', 'soft_drink', 'highblood', 'stroke', 'complication_risk', 'overweight', 'arthritis', 'diabetes', 'hyperlipidemia', 'backpain', 'anxiety',

'allergic_rhinitis', 'reflux_esophagitis', 'asthma',

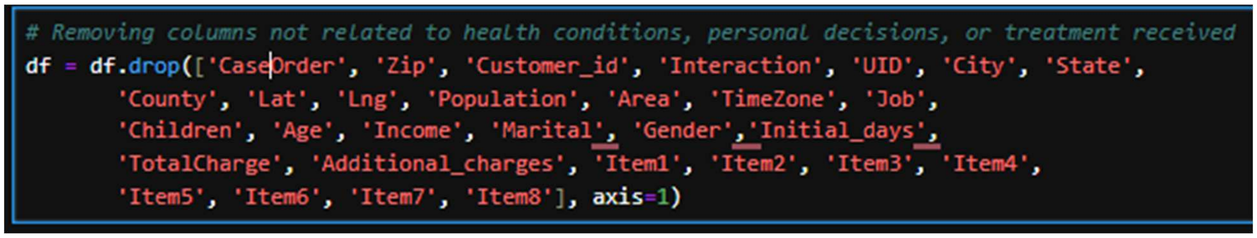
'initial_admin, 'services'

Numeric Variables:

'vitd_levels', 'doc_visits', 'full_meals_eaten', 'vitd_supp'

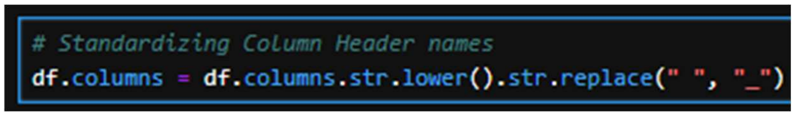
D3 + D4. Explanation of Cleaning & Data File

After importing the medical_clean.csv data file, the first step I took was to drop the columns that were not directly related to the business proposal. All columns related to individual demographics and finances were removed to ensure the analysis targeted the patient's health, the reasons for admission, the services received, and lifestyle choices. In addition, the indexes and survey responses were also dropped as they were irrelevant to the question.



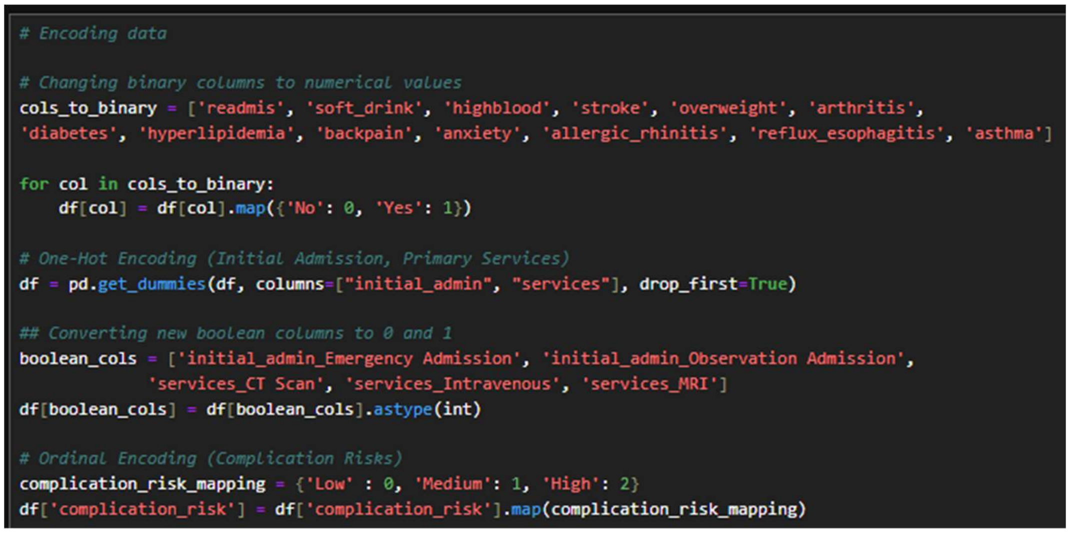
```
# Removing columns not related to health conditions, personal decisions, or treatment received
df = df.drop(['CaseOrder', 'Zip', 'Customer_id', 'Interaction', 'UID', 'City', 'State',
        'County', 'Lat', 'Lng', 'Population', 'Area', 'TimeZone', 'Job',
        'Children', 'Age', 'Income', 'Marital', 'Gender','Initial_days',
        'TotalCharge', 'Additional_charges', 'Item1', 'Item2', 'Item3', 'Item4',
        'Item5', 'Item6', 'Item7', 'Item8'], axis=1)
```

Subsequently, the column headers were standardized per Python convention by changing spaces to underscores and removing capital letters.



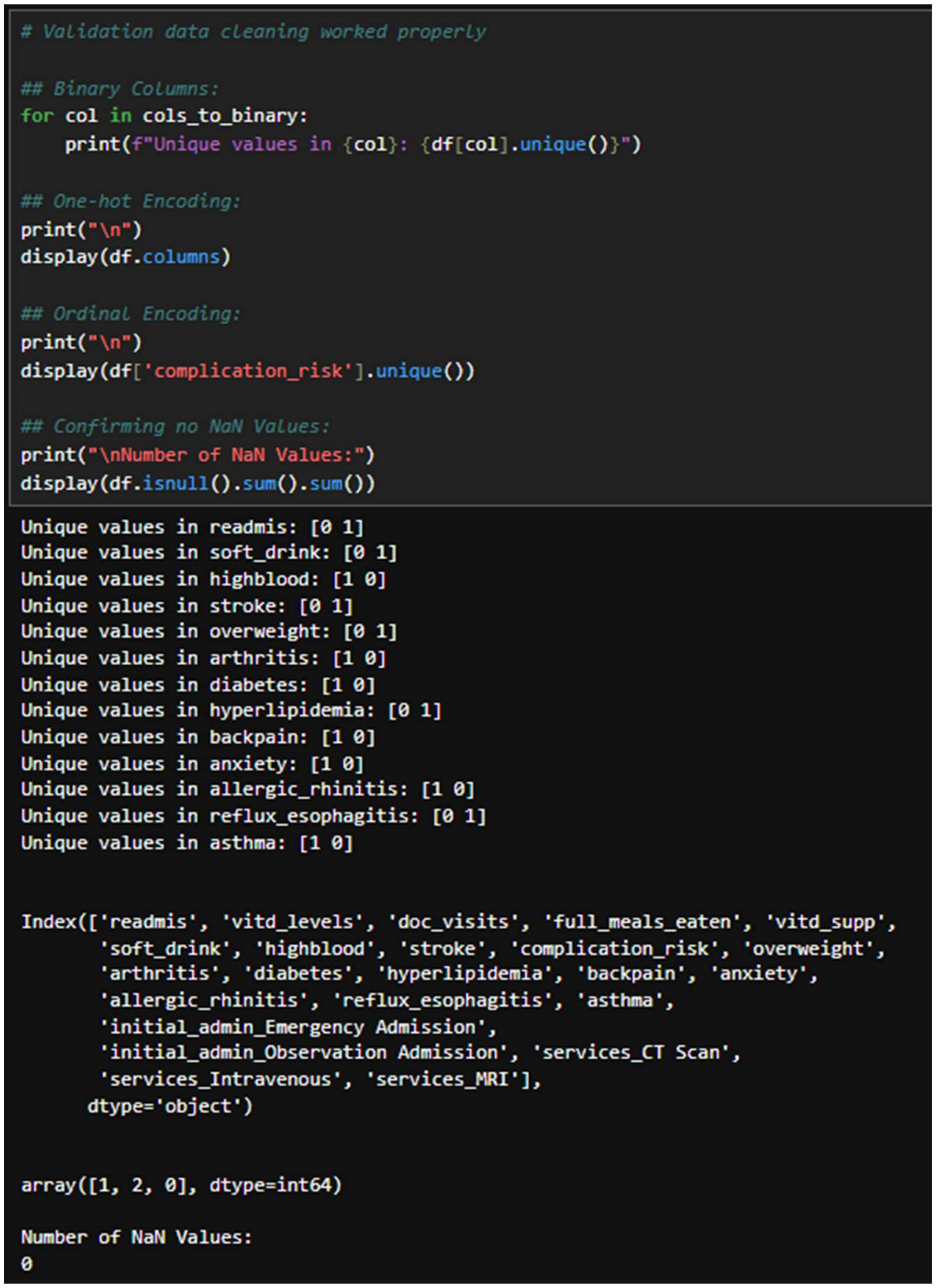
```
# Standardizing Column Header names
df.columns = df.columns.str.lower().str.replace(" ", "_")
```

After confirming that no null values remained in the dataset using an isnull().sum() statement, I moved on to profiling the data to verify the best methods to properly encode the remaining variables for Gradient Boost. After review, I identified the binary columns to change from Yes/No to 1/0 and the categorical variables to either one-hot or ordinally encode, depending on their outputs. As a part of the one-hot encoding, I also converted all the new Boolean columns to 0/1.



```
# Encoding data

# Changing binary columns to numerical values
cols_to_binary = ['readmis', 'soft_drink', 'highblood', 'stroke', 'overweight', 'arthritis',
'diabetes', 'hyperlipidemia', 'backpain', 'anxiety', 'allergic_rhinitis', 'reflux_esophagitis', 'asthma']

for col in cols_to_binary:
    df[col] = df[col].map({'No': 0, 'Yes': 1})

# One-Hot Encoding (Initial Admission, Primary Services)
df = pd.get_dummies(df, columns=["initial_admin", "services"], drop_first=True)

## Converting new boolean columns to 0 and 1
boolean_cols = ['initial_admin_Emergency Admission', 'initial_admin_Observation Admission',
             'services_CT Scan', 'services_Intravenous', 'services_MRI']
df[boolean_cols] = df[boolean_cols].astype(int)

# Ordinal Encoding (Complication Risks)
complication_risk_mapping = {'Low' : 0, 'Medium': 1, 'High': 2}
df['complication_risk'] = df['complication_risk'].map(complication_risk_mapping)
```

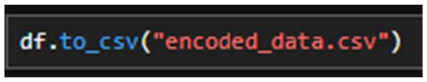
My final step for cleaning before exporting the file was to validate the changes made and prepare the dataset for further analysis. In this step, I also ensured none of the replaced values came back as NaN values, as those would also affect the analysis further down the line.



```
# Validation data cleaning worked properly

## Binary Columns:
for col in cols_to_binary:
    print(f"Unique values in {col}: {df[col].unique()}")

## One-hot Encoding:
print("\n")
display(df.columns)

## Ordinal Encoding:
print("\n")
display(df['complication_risk'].unique())

## Confirming no NaN Values:
print("\nNumber of NaN Values:")
display(df.isnull().sum().sum())
```

```
Unique values in readmis: [0 1]
Unique values in soft_drink: [0 1]
Unique values in highblood: [1 0]
Unique values in stroke: [0 1]
Unique values in overweight: [0 1]
Unique values in arthritis: [1 0]
Unique values in diabetes: [1 0]
Unique values in hyperlipidemia: [0 1]
Unique values in backpain: [1 0]
Unique values in anxiety: [1 0]
Unique values in allergic_rhinitis: [1 0]
Unique values in reflux_esophagitis: [0 1]
Unique values in asthma: [1 0]


Index(['readmis', 'vitd_levels', 'doc_visits', 'full_meals_eaten', 'vitd_supp',
       'soft_drink', 'highblood', 'stroke', 'complication_risk', 'overweight',
       'arthritis', 'diabetes', 'hyperlipidemia', 'backpain', 'anxiety',
       'allergic_rhinitis', 'reflux_esophagitis', 'asthma',
       'initial_admin_Emergency Admission',
       'initial_admin_Observation Admission', 'services_CT Scan',
       'services_Intravenous', 'services_MRI'],
      dtype='object')


array([1, 2, 0], dtype=int64)

Number of NaN Values:
0
```

Finally, the dataset was exported.

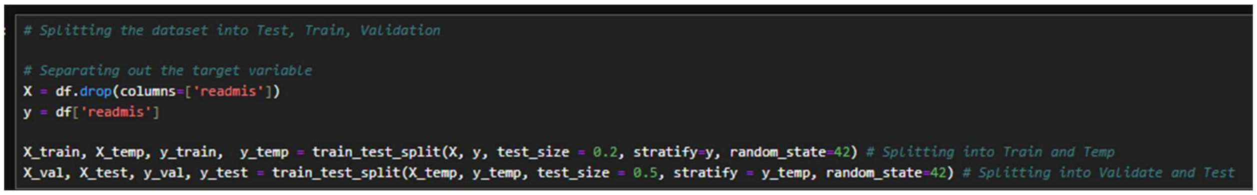


```
df.to_csv("encoded_data.csv")
```

Part 3: Data Training & Analysis

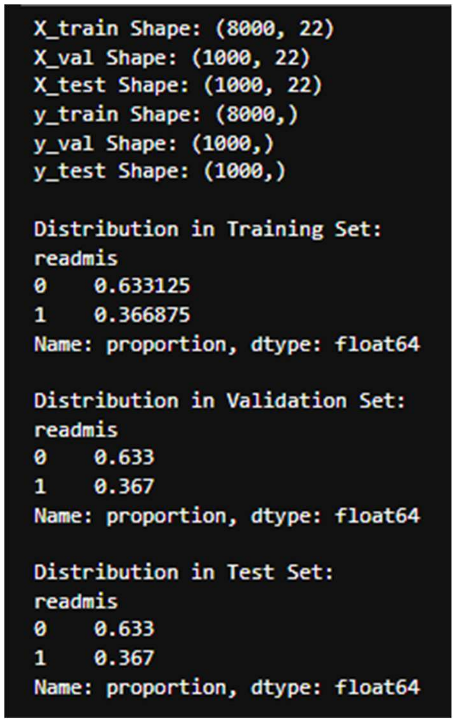
E1. Splitting the Dataset

The dataset needed to be split along three lines – training, testing, and validation. To properly accomplish this, two separate splits were made. The first split was an 80/20 split similar to a regression model train/test split. The two sets were labeled train (80%) and temp (20%). The second split came when taking the temp dataset and splitting it 50/50 to create the validation and test datasets. Also, I utilized a stratified sampling to ensure class balance in my splits.



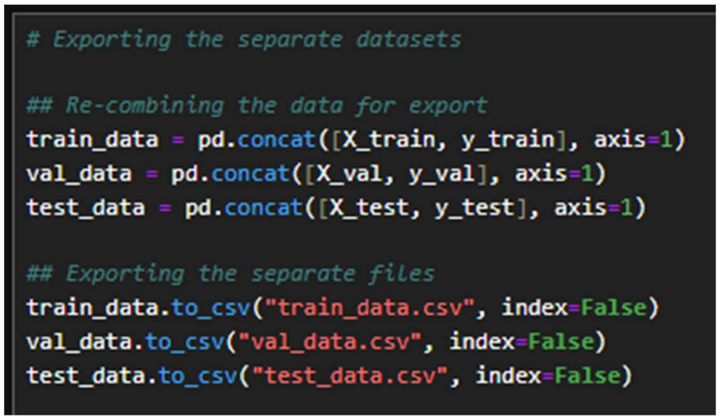
```python
# Splitting the dataset into Test, Train, Validation

# Separating out the target variable
X = df.drop(columns=['readmis'])
y = df['readmis']

X_train, X_temp, y_train,  y_temp = train_test_split(X, y, test_size = 0.2, stratify=y, random_state=42) # Splitting into Train and Temp
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size = 0.5, stratify = y_temp, random_state=42) # Splitting into Validate and Test
```

Afterward, I validated the effectiveness of the splits. I identified that all 22 features were maintained for the X splits, and the appropriate number of observations was created in each split. Additionally, the stratification was adequate, with a 63% no 37% yes average across all datasets.



```
X_train Shape: (8000, 22)
X_val Shape: (1000, 22)
X_test Shape: (1000, 22)
y_train Shape: (8000,)
y_val Shape: (1000,)
y_test Shape: (1000,)

Distribution in Training Set:
readmis
0    0.633125
1    0.366875
Name: proportion, dtype: float64

Distribution in Validation Set:
readmis
0    0.633
1    0.367
Name: proportion, dtype: float64

Distribution in Test Set:
readmis
0    0.633
1    0.367
Name: proportion, dtype: float64
```

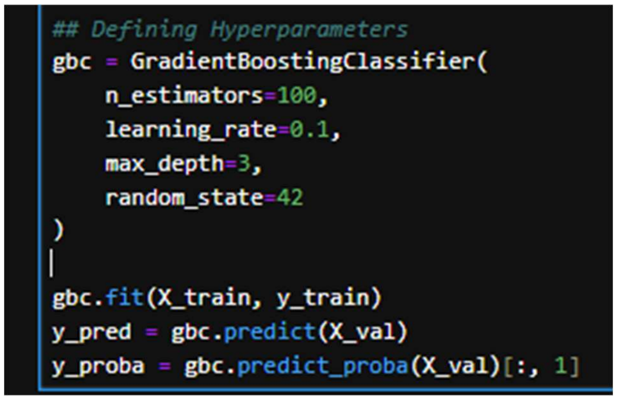
Finally, I re-combined and exported the datasets.



```
# Exporting the separate datasets

## Re-combining the data for export
train_data = pd.concat([X_train, y_train], axis=1)
val_data = pd.concat([X_val, y_val], axis=1)
test_data = pd.concat([X_test, y_test], axis=1)

## Exporting the separate files
train_data.to_csv("train_data.csv", index=False)
val_data.to_csv("val_data.csv", index=False)
test_data.to_csv("test_data.csv", index=False)
```

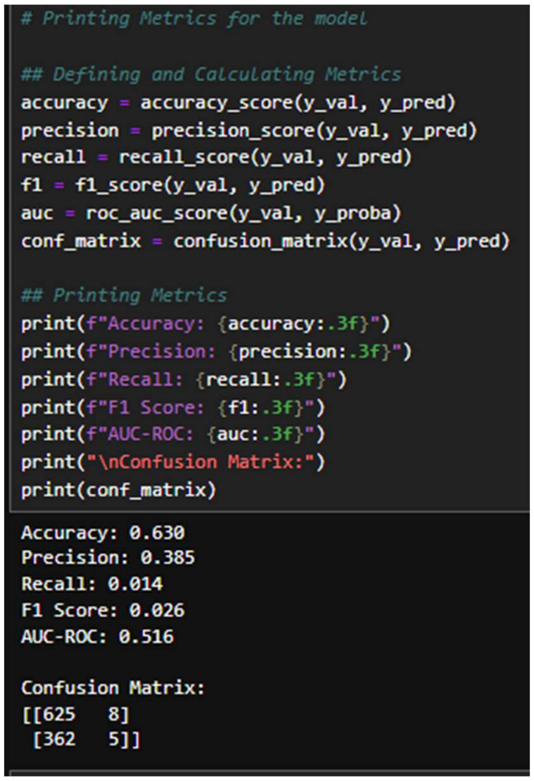
E2. Initial Model & Metrics

After the data was split correctly and exported, the next step was to create the initial Gradient Boost model and export the relevant metrics to review its effectiveness. This was completed using the GradientBoostClassifier class from the sklearn.ensemble module, and the hyperparameters were selected at their standard starting values of 0.1 for the learning rate, 3 for depth, and 100 for number of iterations.



```
## Defining Hyperparameters
gbc = GradientBoostingClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    random_state=42
)

gbc.fit(X_train, y_train)
y_pred = gbc.predict(X_val)
y_proba = gbc.predict_proba(X_val)[:, 1]
```

The metrics were then exported to evaluate the efficacy of the initial model on the dataset. The results indicate the model is not very effective in its current state, with some metrics well outside an acceptable range, including capturing almost no readmissions properly with a 0.014 recall and an AUC-ROC value of .516, indicating the model is barely outperforming a coin toss on accurate predictions.

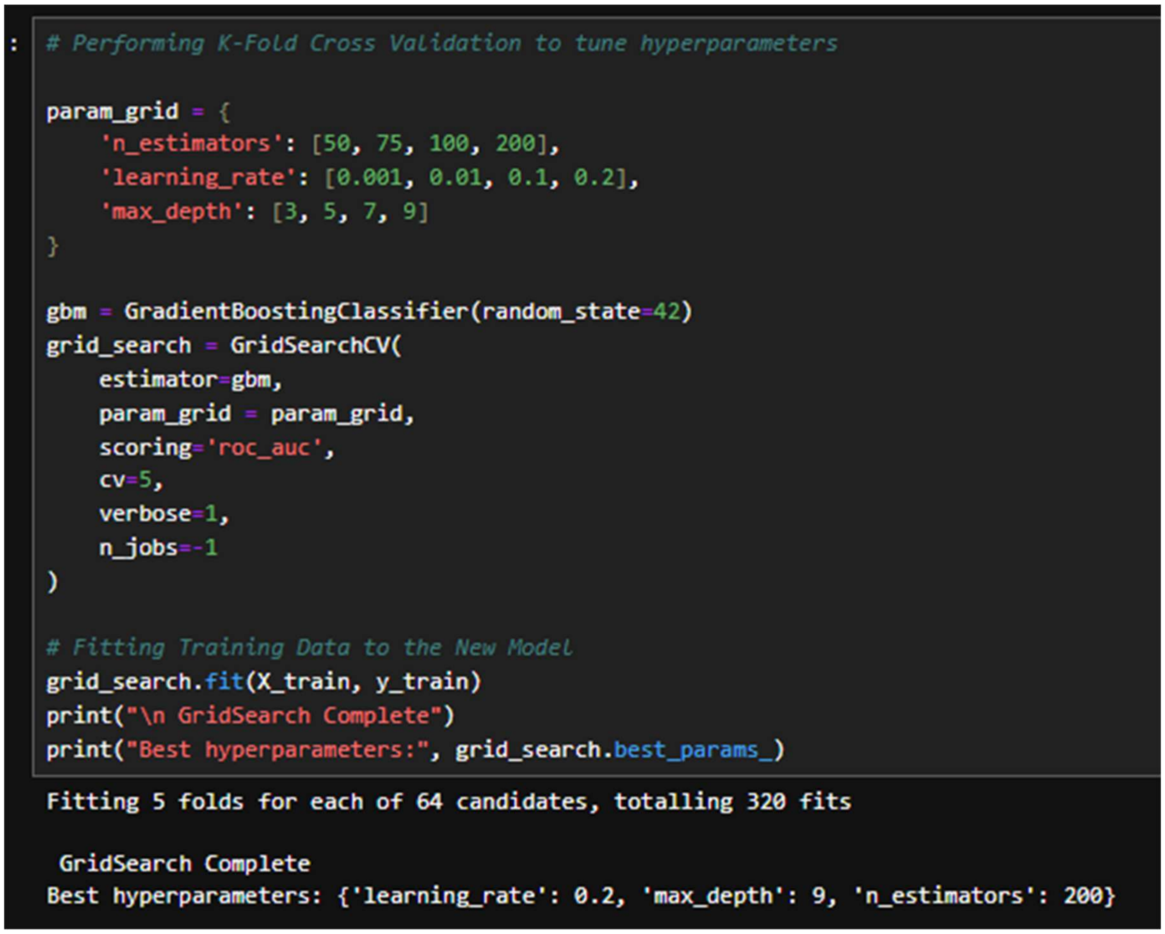


```
# Printing Metrics for the model

## Defining and Calculating Metrics
accuracy = accuracy_score(y_val, y_pred)
precision = precision_score(y_val, y_pred)
recall = recall_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
auc = roc_auc_score(y_val, y_proba)
conf_matrix = confusion_matrix(y_val, y_pred)

## Printing Metrics
print(f"Accuracy: {accuracy:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")
print(f"F1 Score: {f1:.3f}")
print(f"AUC-ROC: {auc:.3f}")
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.630
Precision: 0.385
Recall: 0.014
F1 Score: 0.026
AUC-ROC: 0.516

Confusion Matrix:
[[625   8]
 [362   5]]
```

E3. Hyperparameter Tuning

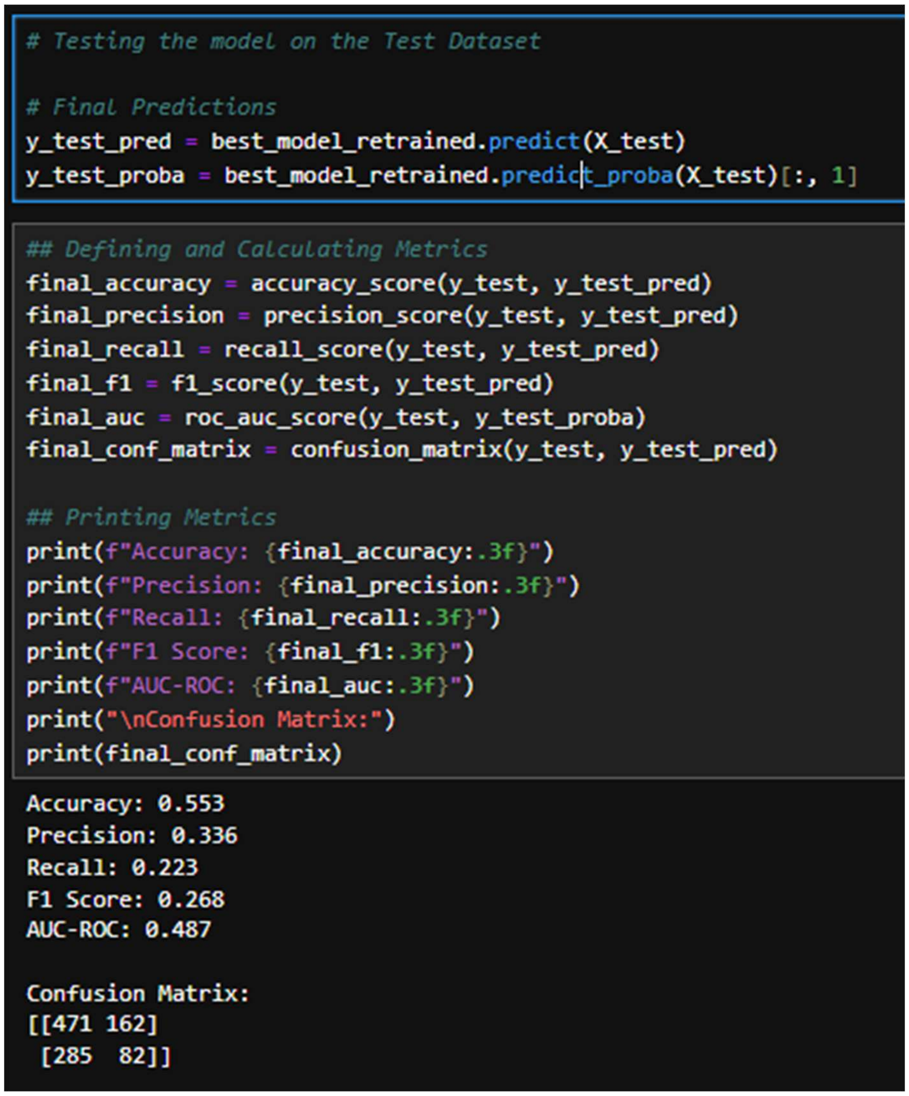
After running the initial model, hyperparameter tuning was utilized to improve the overall predictive power of the Gradient Boost. The most impactful hyperparameters for Gradient Boost are the learning rate, the depth of the trees, and the number of estimators. Each of these is useful to tune together as they often interact. A lower learning rate may require more estimators to maintain performance. A deeper depth can increase the overall complexity, which may require reducing the number of estimators to prevent overfitting. To tune these values, a 4x4x4 grid was created for hyperparameter ranges, and they were compared via a K-Fold Cross-Validation.



```
# Performing K-Fold Cross Validation to tune hyperparameters

param_grid = {
    'n_estimators': [50, 75, 100, 200],
    'learning_rate': [0.001, 0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7, 9]
}

gbm = GradientBoostingClassifier(random_state=42)
grid_search = GridSearchCV(
    estimator=gbm,
    param_grid = param_grid,
    scoring='roc_auc',
    cv=5,
    verbose=1,
    n_jobs=-1
)

# Fitting Training Data to the New Model
grid_search.fit(X_train, y_train)
print("\n GridSearch Complete")
print("Best hyperparameters:", grid_search.best_params_)
```

```
Fitting 5 folds for each of 64 candidates, totalling 320 fits

 GridSearch Complete
Best hyperparameters: {'learning_rate': 0.2, 'max_depth': 9, 'n_estimators': 200}
```

The outcome was a learning rate of 0.2, a max depth of 9, and 200 estimators – all metrics were increased in complexity compared to the initial model.



```
# Testing the model on the Test Dataset

# Final Predictions
y_test_pred = best_model_retrained.predict(X_test)
y_test_proba = best_model_retrained.predict_proba(X_test)[:, 1]
```

```
## Defining and Calculating Metrics
final_accuracy = accuracy_score(y_test, y_test_pred)
final_precision = precision_score(y_test, y_test_pred)
final_recall = recall_score(y_test, y_test_pred)
final_f1 = f1_score(y_test, y_test_pred)
final_auc = roc_auc_score(y_test, y_test_proba)
final_conf_matrix = confusion_matrix(y_test, y_test_pred)

## Printing Metrics
print(f"Accuracy: {final_accuracy:.3f}")
print(f"Precision: {final_precision:.3f}")
print(f"Recall: {final_recall:.3f}")
print(f"F1 Score: {final_f1:.3f}")
print(f"AUC-ROC: {final_auc:.3f}")
print("\nConfusion Matrix:")
print(final_conf_matrix)
```

```
Accuracy: 0.553
Precision: 0.336
Recall: 0.223
F1 Score: 0.268
AUC-ROC: 0.487

Confusion Matrix:
[[471 162]
 [285  82]]
```

E4. Prediction & Metrics

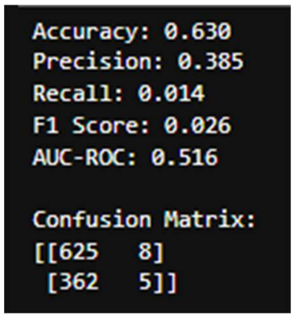
The updated hyperparameters were utilized to re-run the training data values and confirm the metrics' increased accuracy in the validation training set. Once confirmed, the final step was running the optimized model on the test dataset to introduce data the model had not yet touched. The final results were a model that improved in several places compared to the initial iteration.

Part 4: Summary and Results

F1. Comparison of Models

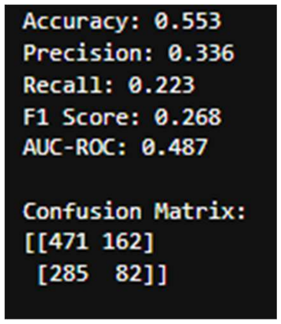
The hyperparameter tuning had a largely positive effect on the model's efficacy. The model itself demonstrated that the selected variables do not predict readmission to the hospital very well on their own. The tuning demonstrated the following changes to the dataset:

Initial Model



```
Accuracy: 0.630
Precision: 0.385
Recall: 0.014
F1 Score: 0.026
AUC-ROC: 0.516

Confusion Matrix:
[[625   8]
 [362   5]]
```

Final model



```
Accuracy: 0.553
Precision: 0.336
Recall: 0.223
F1 Score: 0.268
AUC-ROC: 0.487

Confusion Matrix:
[[471 162]
 [285  82]]
```

Accuracy, or overall correctness of the model, decreased slightly from 63% to 55%.
Precision, or how many positive predictions were correct, decreased slightly from 39% to 34%.
Recall, or how many actual positives were caught, increased significantly from 1.4% to 22.3%.
F1 Score, or the balance between false positives and false negatives, also increased significantly, changing from 2.6% to 27%.
The AUC-ROC score, or overall ability to distinguish the different classes, also decreased from 51.6% to 48.7%.

F2. Results and Implications

After creating and assessing the model, the first detail gleaned from the metrics is that the model itself is not a very effective predictor of patient readmission rates. This is true before and after tuning the hyperparameters to make the model function more effectively. The AUC-ROC score demonstrates that the predictive power never gets better than a coinflip chance at predicting readmission, and after tuning the model, it gets worse than a coin flip. Additionally, accuracy and precision rates never reach acceptable levels for a passable model. After improving the model, the recall and F1 scores improved compared to the initial model but are still under 30% for both metrics, leading to poor predictive power overall. The main implication and takeaway is that the selected variables do not explain why a patient is re-admitted to the hospital. Therefore, additional factors or variables should be folded into the model, and the research question should be rejected as a valid predictor.

F3. Limitations

One major limitation of the analysis is that it removes all demographic factors from the dataset to predict the dependent variable. The model metrics suggest that this choice resulted in poor predictive power. Without knowing where the patients came from, their gender, familial status, and other essential details, the analysis lacked the granularity needed to break patients into different categories, which may glean compelling insights to reduce re-admission and decrease costs.

F4. Business Recommendations

The primary recommendation to the business is to take one of two approaches to the business question. The first approach would be to collect additional data related to the customer's health history, choices, and details of the care they received while in the hospital and run a broader hyper-tuning of the relevant parameters to improve the model's overall accuracy further. With each step towards a more complicated model, the recall and F1 scores increased in the previous model. If additional computing power was acquired or RamdonizedSearchCV was utilized, the parameter grid could be expanded, and better metrics to tune the model may result in improved performance. The second and more prevalent suggestion would be to scrap the initial business question overall and instead pose a new inquiry that folds in patients' demographic metrics to better understand what affects re-admission to the hospital.

References

No other sources were used outside of the WGU course materials provided