D209 Task 2 (Predictive Analysis) Performance Assessment

Hillary Osei (Student ID #011039266)

Western Governors University, College of Information Technology

Program Mentor: Dan Estes

February 19, 2025

Table of Contents

Part I: Research Question

Section A1) Research Question

The research question is “What are the major predictor variables that can predict patient readmissions?” The prediction method used to analyze this is the decision tree.

Section A2) Defined Goal

The data analysis aims to determine the essential factors that could determine how likely patients are readmitted to the hospital within one month of their initial release by analyzing patient demographic information, medical conditions, hospital admissions information, initial hospital stay length, and medical services received.

Part II: Method Justification

Section B1) Explanation of Prediction

The decision tree is a supervised non-parametric machine learning algorithm to make predictions. It graphically resembles a tree or a flowchart and includes a root node, branches, internal nodes, and leaf nodes. The decision tree can handle categorical and continuous variables (IBM, n.d.) The root node, which is at the very top of the tree, represents the best predictor, and the terminal leaf node, which is at the end, represents the algorithm's final outcome (Elleh, n.d.).

The expected outcome of the decision tree is to identify the most significant variables that predict and contribute most to patient readmissions. The decision tree will categorize predictors based on importance, therefore valuable insights into which factors should be targeted by healthcare providers to help reduce hospital readmissions.

Section B2) Summary of Method Assumption

An assumption for the decision tree is that each decision can be represented by a binary choice. This is because the trees usually make binary splits, meaning that each node is divided into two based on a feature or condition (Anshul, 2025).

Section B3) Packages or Libraries List

Below is a list of the Python packages and libraries used for the data analysis, and justifications for using them:

1. **pandas**: Used for data manipulation and analysis, creating and managing dataframes
2. **matplotlib.pyplot**: Creates graphs and plots for visualizing data (e.g. histograms)

3. **numpy**: Performs mathematical operations on arrays and numerical data
4. **math**: Computes basic math functions like floor and ceiling
5. **seaborn**: Creates heatmaps and histograms
6. **statsmodels.stats.outliers_influence.variance_inflation_factor:** Calculates VIF values for predictor variables, checking for multicollinearity
7. **sklearn.model_selection.train_test_split**: Splits data into training and testing sets
8. **sklearn.feature_selection.SelectKBest, f_classif**: Selects features by ranking based on statistical significance in predicting target variable
9. **sklearn.tree.DecisionTreeClassifier**: Builds decision tree model for predicting patient readmissions
10. **sklearn.tree.plot_tree**: Visualizes decision tree
11. **sklearn.model_selection.GridSearchCV**: Performs hyperparameter tuning to find optimal decision tree configuration
12. **sklearn.metrics.confusion_matrix**: Creates a confusion matrix evaluating the performance of decision tree by comparing actual vs predicted data
13. **sklearn.metrics.roc_auc_score**: Calculates area under ROC curve
14. **sklearn.metrics.roc_curve**: Plots the ROC curve
15. **sklearn.metrics.classification_report**: Provides summary of decision tree's performance metrics, including precision, recall, F1-score, and accuracy
16. **sklearn.metrics.accuracy_score**: Calculates accuracy of decision tree
17. **sklearn.metrics.mean_squared_error**: Measures prediction error for decision tree

Part III: Data Preparation

Section C1) Data Preprocessing

Data preprocessing involves preparing raw data for analysis by cleaning, for example, handling duplicates, missing values, and outliers and transforming it into a usable format (Jain, 2025). An important data preprocessing goal for decision trees is re-expressing categorical variables using one-hot encoding to convert them into dummy variables. Because most machine learning models are incompatible with categorical variables, numeric variables are needed in order for the model to effectively process and split the data into decision nodes.

Section C2) Data Set Variables

Below is a table of the predictor variables used in the data types and their respective data types:

Predictor Variable	Data Type
Children	Numeric
Age	Numeric
Income	Numeric
Marital	Categorical
Gender	Categorical
Doc_visits	Numeric
Initial_admin	Categorical
HighBlood	Categorical
Stroke	Categorical
Complication_risk	Categorical
Overweight	Categorical
Arthritis	Categorical
Diabetes	Categorical
Hyperlipidemia	Categorical
BackPain	Categorical
Anxiety	Categorical
Allergic_rhinitis	Categorical
Reflux_esophagitis	Categorical
Asthma	Categorical
Services	Categorical
Initial_days	Numeric

Section C3) Steps for Analysis

The data preparation process begins by loading the clean medical data into the dataframe using pd.read_csv(), and setting the dataframe to the variable “df.”

```
df = pd.read_csv('medical_clean.csv')
```

Next is to get an overview of the dataframe using df.info() to inspect non-null counts and the data types of each variable. Duplicate rows are checked using df.duplicated(), which confirms that no duplicates are present in the dataset. Missing values are detected using df.isnull.sum(), with results showing that no missing values were found in the variables.

Irrelevant columns, including "CaseOrder," "Customer_id," "Interaction," "UID," "City," "State," and others, are removed using df.drop() in order to focus the analysis process on variables relevant to the research question:

```
# Drop irrelevant columns
df = df.drop(columns=['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State', 'County', 'Zip', 'Lat', 'Lng', 'Population', 'Area',
                      'VitD_levels', 'Full_meals_eaten', 'vitD_supp', 'Soft_drink','TimeZone', 'Job', 'Additional_charges', 'TotalCharge',
                      'Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6', 'Item7', 'Item8'])
```

Outliers are checked in the numeric variables “Children,” “Age,” “Income,” “Doc_visits,” and “Initial_days,” with outliers found in the variables “Children” and “Income.” These outliers are retained to preserve potentially valuable insights for the data analysis:

```
# Check for outliers in 'Children'
plt.figure(figsize=(6, 4))  # Adjust figure size
sns.boxplot(x=df["Children"])

# Add title and labels
plt.title("Boxplot for Children", fontsize=16)
plt.xlabel("Children", fontsize=12)

# Show the plot
plt.show()


# Check for outliers in 'Age'
plt.figure(figsize=(6, 4))  # Adjust figure size
sns.boxplot(x=df["Age"])
```

```
# Add title and labels
plt.title("Boxplot for Age", fontsize=16)
plt.xlabel("Age", fontsize=12)

# Show the plot
plt.show()

# Check for outliers in 'Income'
plt.figure(figsize=(6, 4))  # Adjust figure size
sns.boxplot(x=df["Income"])

# Add title and labels
plt.title("Boxplot for Income", fontsize=16)
plt.xlabel("Income", fontsize=12)

# Show the plot
plt.show()

# Check for outliers in 'Doc_visits'
plt.figure(figsize=(6, 4))  # Adjust figure size
sns.boxplot(x=df["Doc_visits"])

# Add title and labels
plt.title("Boxplot for Doc_visits", fontsize=16)
plt.xlabel("Doc_visits", fontsize=12)

# Show the plot
plt.show()

# Check for outliers in 'Initial_days'
plt.figure(figsize=(6, 4))  # Adjust figure size
sns.boxplot(x=df["Initial_days"])

# Add title and labels
plt.title("Boxplot for Initial_days", fontsize=16)
plt.xlabel("Initial_days", fontsize=12)

# Show the plot
plt.show()
```

Summary statistics for the numeric predictor variables “Children,” “Age,” “Income,” “Doc_visits,” and “Initial_days” are examined to get information of distribution patterns

```
# Get summary statistics info for continuous variables
df.describe()
```

Value counts for categorical variables “Marital,” “Gender,” “ReAdmis,” “Initial_admin,” “HighBlood,” Stroke,” “Complication_risk,” “Overweight,” “Arthritis,” “Diabetes,” “Hyperlipidemia,” “BackPain,” “Asthma,” “Anxiety,” “Allergic_rhinitis,” “Reflux_esophagitis,” and “Services.”

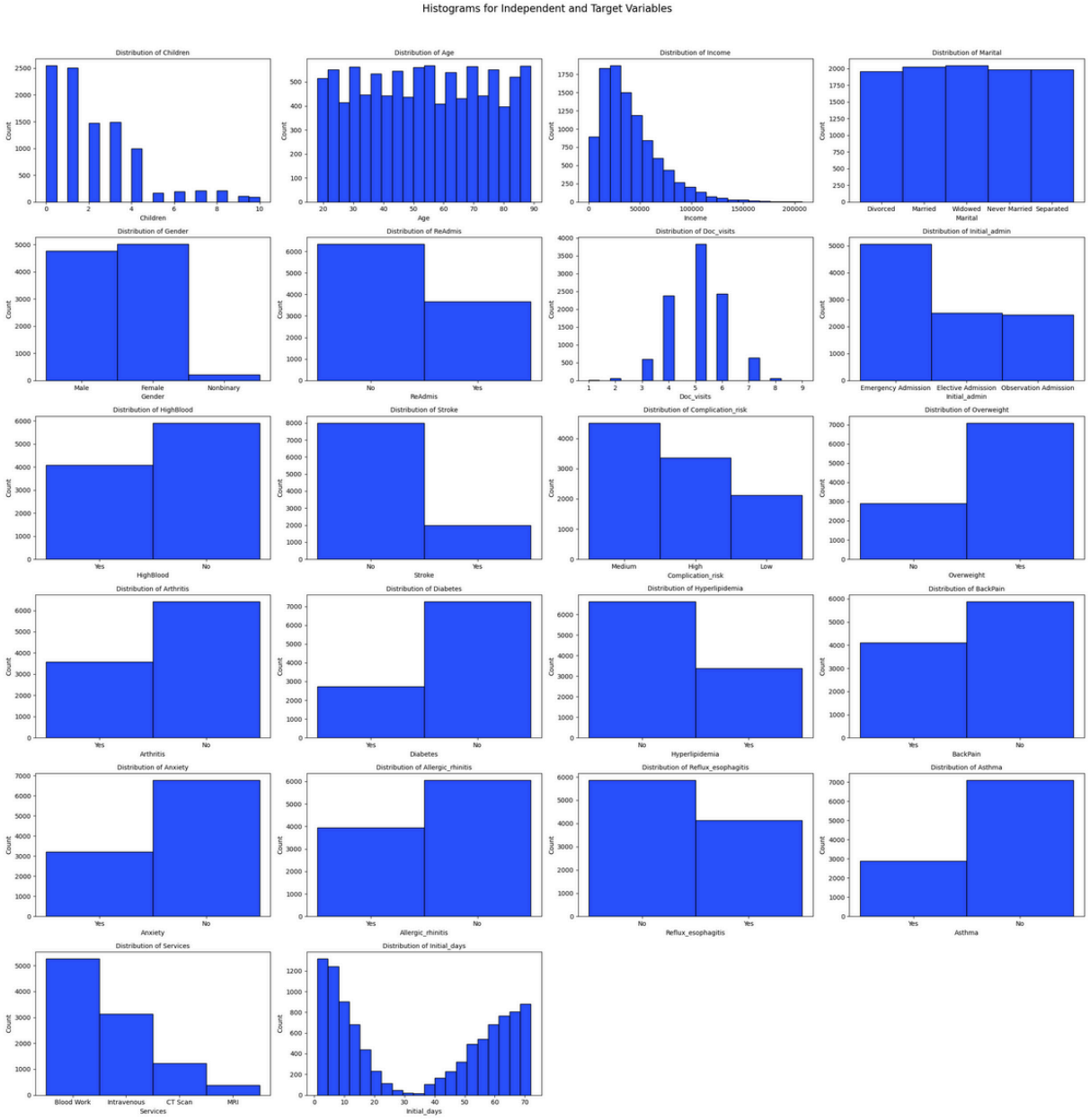
```
# Get value counts for 'Marital'
df['Marital'].value_counts()
# Get value counts for 'Gender'
df['Gender'].value_counts()
# Get value counts for 'ReAdmis'
df['ReAdmis'].value_counts()
# Get value counts for 'Initial_admin'
df['Initial_admin'].value_counts()
# Get value counts for 'HighBlood'
df['HighBlood'].value_counts()
# Get value counts for 'Stroke'
df['Stroke'].value_counts()
# Get value counts for 'Complication_risk'
df['Complication_risk'].value_counts()
# Get value counts for 'Overweight'
df['Overweight'].value_counts()
# Get value counts for 'Arthritis'
df['Arthritis'].value_counts()
# Get value counts for 'Diabetes'
df['Diabetes'].value_counts()
# Get value counts for 'Hyperlipidemia'
df['Hyperlipidemia'].value_counts()
# Get value counts for 'BackPain'
df['BackPain'].value_counts()
# Get value counts for 'Asthma'
df['Asthma'].value_counts()
# Get value counts for 'Anxiety'
df['Anxiety'].value_counts()
# Get value counts for 'Allergic_rhinitis'
df['Allergic_rhinitis'].value_counts()
# Get value counts for 'Reflux_esophagitis'
```

```
df['Reflux_esophagitis'].value_counts()
# Get value counts for 'Services'
df['Services'].value_counts()
```

Histograms are created to visualize the distributions of the independent and target variables. Based on the results, "Age" shows a uniform distribution, "Income" shows a right-skewed distribution, and "Initial_days" shows a bimodal distribution.

```
# Create histogram grid

# Define the number of columns in the grid
num_cols = 4

# Calculate the number of rows required
num_rows = math.ceil(len(df.columns) / num_cols)

# Create the subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(24, 24))
fig.subplots_adjust(hspace=0.4, wspace=0.4)  # Add spacing between plots

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Loop through each predictor variable and the target variable to plot the histograms
for i, col in enumerate(df.columns):
        sns.histplot(df[col], bins=20, kde=False, ax=axes[i], color='blue')  # Plot histogram
        axes[i].set_title(f"Distribution of {col}", fontsize=10)  # Add title
        axes[i].set_xlabel(col)  # Label x-axis
        axes[i].set_ylabel("Count")  # Label y-axis

# Remove unused subplots
for j in range(len(df.columns), len(axes)):
        fig.delaxes(axes[j])

# Add overall title for the grid of plots
plt.suptitle("Histograms for Independent and Target Variables", fontsize=16, y=1.02)

# Adjust layout to avoid overlap
plt.tight_layout()

# Show the plot
```

plt.show()



Categorical variables with yes/no values, including “HighBlood,” “Stroke,” “Overweight,” “Arthritis,” “Diabetes,” “Hyperlipidemia,” “BackPain,” “Asthma,” “Anxiety,” “Allergic_rhinitis,” “Reflux_esophagitis,” and “Readmis,” are re-expressed to binary values (0 or 1) to make it compatible with machine learning models.

```
# Re-express categorical variables w/ Yes/No values
yes_no = {"Yes": 1, "No": 0}
df["HighBlood"] = df["HighBlood"].replace(yes_no).infer_objects(copy=False)
```

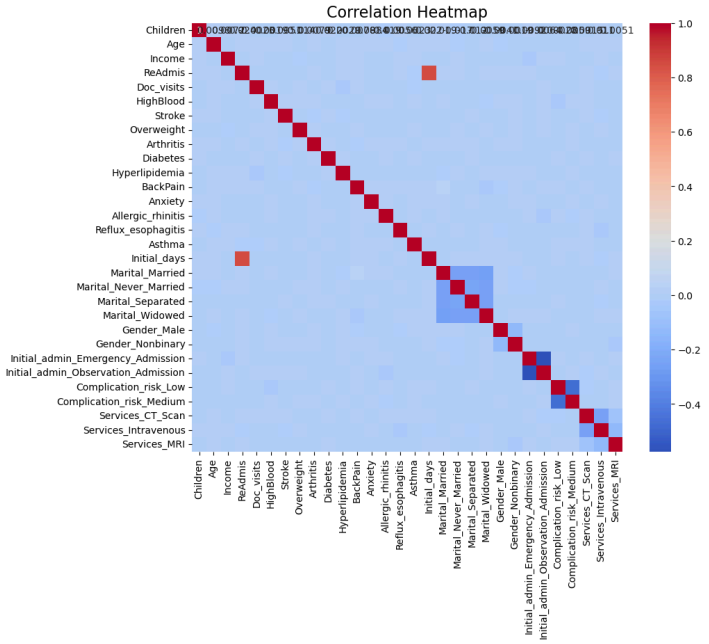
```
df["Stroke"] = df["Stroke"].replace(yes_no).infer_objects(copy=False)
df["Overweight"] = df["Overweight"].replace(yes_no).infer_objects(copy=False)
df["Arthritis"] = df["Arthritis"].replace(yes_no).infer_objects(copy=False)
df["Diabetes"] = df["Diabetes"].replace(yes_no).infer_objects(copy=False)
df["Hyperlipidemia"] = df["Hyperlipidemia"].replace(yes_no).infer_objects(copy=False)
df["BackPain"] = df["BackPain"].replace(yes_no).infer_objects(copy=False)
df["Asthma"] = df["Asthma"].replace(yes_no).infer_objects(copy=False)
df["Anxiety"] = df["Anxiety"].replace(yes_no).infer_objects(copy=False)
df["Allergic_rhinitis"] = df["Allergic_rhinitis"].replace(yes_no).infer_objects(copy=False)
df["Reflux_esophagitis"] = df["Reflux_esophagitis"].replace(yes_no).infer_objects(copy=False)
df["ReAdmis"] = df["ReAdmis"].replace(yes_no).infer_objects(copy=False)
```

Dummy variables are generated for categorical variables with more than two categories, including “Marital,” “Gender,” “Initial_admin,” “Complication_risk,” and “Services,” using pd.get_dummies(). The original categorical variables are dropped using df.drop(). Then, spaces present in the new encoded variables are replaced with underscores to standardize variable naming.

```
# Create dummy variables for other categorical variables and concatenate them with the original dataframe
df = pd.concat([df, pd.get_dummies(df["Marital"], prefix="Marital", drop_first=True, dtype='int')], axis=1)
df = pd.concat([df, pd.get_dummies(df["Gender"], prefix="Gender", drop_first=True, dtype='int')], axis=1)
df = pd.concat([df, pd.get_dummies(df["Initial_admin"], prefix="Initial_admin", drop_first=True, dtype='int')], axis=1)
df = pd.concat([df, pd.get_dummies(df["Complication_risk"], prefix="Complication_risk", drop_first=True, dtype='int')], axis=1)
df = pd.concat([df, pd.get_dummies(df["Services"], prefix="Services", drop_first=True, dtype='int')], axis=1)


# Drop original categorical variables used for dummies
df = df.drop(columns=["Marital"])
df = df.drop(columns=["Gender"])
df = df.drop(columns=["Initial_admin"])
df = df.drop(columns=["Complication_risk"])
df = df.drop(columns=["Services"])

# Replace spaces with '_'
df.columns = df.columns.str.replace(' ', '_')
```

A correlation map is made to visualize the relationships between the predictor and target variables and check for multicollinearity. This is important because high correlation can cause redundancy and overfitting, thus affecting the model's ability to make accurate predictions (Adison, 2024). The heatmap shows that most variables have low correlation values, meaning there is no strong correlation between "ReAdmis" and the predictors. On the other hand, "Initial_days" and "Income" have a positive correlation, meaning that higher income is associated with longer initial stay durations and vice versa.

```
# Check for correlation using heatmap

plt.figure(figsize=(10, 8))  # Set figure size
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

# Add a title
plt.title("Correlation Heatmap", fontsize=16)

# Show the plot
plt.show()
```



Section C4) Cleaned Data Set

Cleaned data set is attached, titled “task2_cleaned.csv”

Part IV: Analysis

Section D1) Splitting the Data

The data split into training and testing datasets, using the following code below, to evaluate the model's performance on unseen data (Noble Desktop, 2024):

```
# Separate predictor and target variables
X = df[['Children', 'Age', 'Income', 'Doc_visits', 'HighBlood', 'Stroke','Overweight', 'Arthritis', 'Diabetes', 'Hyperlipidemia', 'BackPain', 'Anxiety', 'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Marital_Married', 'Marital_Never_Married', 'Marital_Separated', 'Marital_Widowed', 'Gender_Male', 'Gender_Nonbinary', 'Initial_admin_Emergency_Admission', 'Initial_admin_Observation_Admission', 'Complication_risk_Low','Complication_risk_Medium', 'Services_CT_Scan', 'Services_Intravenous','Services_MRI']]
y = df['ReAdmis']

# Code adapted from:
# Elleh, F. (n.d.). Welcome to D209 Data Mining 1 [Lecture slides]. Western Governors University. Slide 35.
# Split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_size=0.2, random_state=42)
```

The predictor variables and target variables are separated into "X_train," "X_test," "y_train," and "y_test." "X_train" is the training data for the predictor variables and "X_test" is the testing data for the predictor variables. On the other hand, "y_train" is training data for the target variable, and "y_test" is testing data for the target variable. 80% is used for training, and 20% is used for testing. Random_state is set to a random number 42 to ensure the data split is reproducible, meaning splits will produce the same results even if the code is rerun several times (Saturn Cloud, 2023).

Next, the training and testing sets are converted into dataframes and saved as csv files, as seen below:

```
# Save training and testing datasets as csv
pd.DataFrame(X_train).to_csv('X_train2.csv')
pd.DataFrame(X_test).to_csv('X_test2.csv')
pd.DataFrame(y_train).to_csv('y_train2.csv')
pd.DataFrame(y_test).to_csv('y_test2.csv')
```

Section D2) Output and Intermediate Calculations

The optimal hyperparameters for the decision tree model are determined using grid search with cross-validation. The param_grid dictionary specifies the range of values to be tested. The max_depth parameter is set to [2, 4, 6, 8, 10, 12], controlling the maximum depth of the tree. The min_samples_leaf parameter is set to [1, 2, 5, 10, 20], representing the minimum number of samples for a leaf node.

The DecisionTreeClassifier object is instantiated and passed into GridSearchCV, which tests all combinations of the max_depth and min_samples_leaf parameters using 5-fold cross-validation. The model is then trained on the training dataset using grid_search.fit(X_train, y_train) to find the configurations for the best-performing model. Next, the best model creates predictions on the test dataset using grid_search.predict(X_test).

Finally, the optimal hyperparameters are found through the grid search, using grid_search.best_params_. The best hyperparameters for the decision tree are determined to be at max_depth of 2 and min_sample_leaf of 1.

```
# Define parameters for grid search
param_grid = {
    'max_depth': [2, 4, 6, 8, 10, 12],
    'min_samples_leaf': [1, 2, 5, 10, 20]
}

# Code adapted from:
# Navlani, A. (2024). Decision Tree Classification in Python Tutorial. DataCamp.
# https://www.datacamp.com/tutorial/decision-tree-classification-python

# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Perform grid search
grid_search = GridSearchCV(clf, param_grid, cv=5)

# Train the model on the training data using grid search
grid_search.fit(X_train, y_train)

# Predict based on best model
y_pred = grid_search.predict(X_test)

# Print the best hyperparameters found by grid search
print("Best hyperparameters:", grid_search.best_params_)
```

The decision tree model is trained using the optimal hyperparameters found using the grid search. DecisionTreeClassifier is initialized with max_depth = 2, meaning the tree is limited to two levels, and min_sample_leaf = 1, meaning the leaf nodes have a minimum of one sample. The model is trained using fit(X_train, y_train). After training, dt.predict(X_test) makes predictions based on the test dataset.

```
# Fit the decision tree model with the found parameters
dt = DecisionTreeClassifier(max_depth = 2, min_samples_leaf = 1)
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)
```

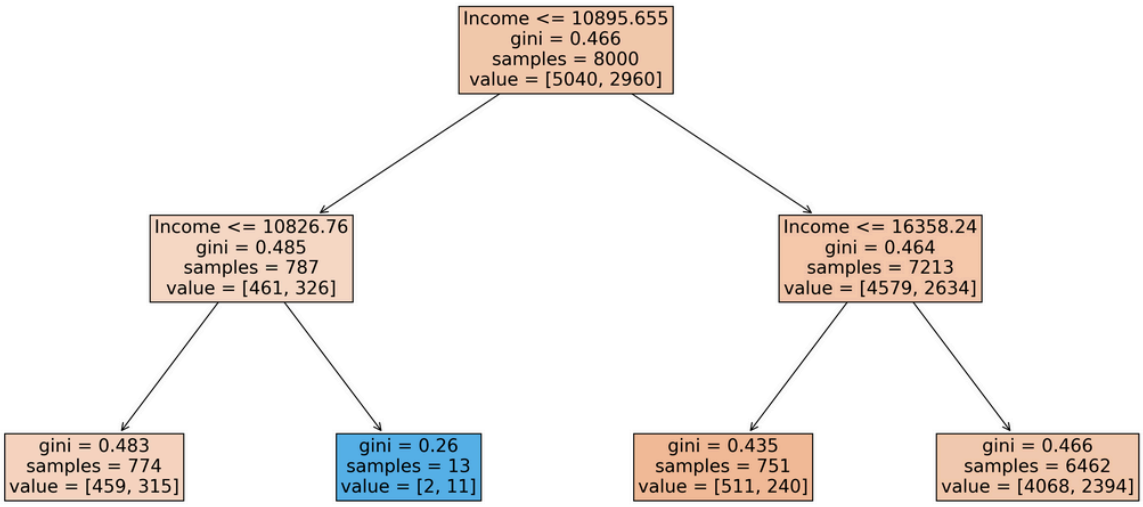
The model's performance is evaluated using accuracy and AUC. The accuracy_score function calculates the correctly predicted outcomes for both the test and training datasets. The test accuracy evaluates how well the model predicts unseen data, and accuracy_score(y_train, grid.search.predict(X_train)) evaluates how well the model fits training data. The model's accuracy on the test set is 64.6%, while the training accuracy is at 63.1%. This means that there is minimal overfitting in the model since the percentages are close to each other. However, the AUC is at 0.5007052186177715 meaning that the model has difficulties differentiating between the classes.

```
# Calculate the accuracy for best model
accuracy = accuracy_score(y_test, y_pred)
# Calculate the accuracy for training dataset
training_accuracy = accuracy_score(y_train, grid_search.predict(X_train))

print("Accuracy for best model:", accuracy)
print("Training Accuracy:", training_accuracy)
```

The decision tree model is visualized using plot_tree(), plotting the best-performing model using grid_search.best_estimator, using the following code:

```
# Plot the decision tree
plt.figure(figsize=(20,10), dpi=200)
plot_tree(grid_search.best_estimator_, filled=True, fontsize=16, feature_names=X.columns)
plt.show()
```

Here is a visualization of the decision tree below:



The decision tree shows that income is the most important factor in predicting patient readmission. The root node splits at Income ≤ $10,895.655. The internal nodes split at Income ≤ $10,826.76 on the left and Income ≤ $16,358.24. The Gini Index measures how mixed or impute a dataset is and ranges from 0 to 1. 0 represents a pure dataset, and 1 represents an impure

dataset (Dorfer, 2023). Samples represent how many patients fall into the specific node, and values show the number of patients in each class. The leaf nodes at the bottom (terminal nodes) do not split further, representing the final classifications. The blue node has a low Gini Index at 0.26, meaning it is mostly from class 1, with 1 out of 13 patients being readmitted. This suggests that some lower-income patients are at a higher risk of being readmitted.

A confusion matrix is made to evaluate the performance of the decision tree model by comparing predicted and actual outcomes, as seen below:

```
# Create confusion matrix
print('The confusion matrix for decision tree model')
print(confusion_matrix(y_test, y_pred))
```

The confusion matrix outputs the following:
[[1291 0]
 [708 1]]

The model correctly identifies 1,291 cases as not readmitted (true negatives). There are 708 cases where the model incorrectly predicted non-readmission when the patient was actually readmitted (false negatives). There are 0 cases of non-readmitted patients being misclassified as readmitted (false positives). There is 1 case where the model correctly identifies a readmitted patient.

The Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) are metrics used to evaluate how well a model performs on a dataset. MSE determines the average squared difference between predicted and actual values while RMSE determines the square root of the average squared difference between predicted and actual values. The lower the MSE and RMSE are, the better the model performs (Bobbitt, 2021). Because the target variable for the decision tree is categorical (readmitted or not readmitted), MSE and RMSE are not an appropriate evaluation metric (Straw, n.d.), instead, other metrics such as confusion matrix, precision, recall, F-1 score, and ROC-AUC score are suitable for the analysis. However, here is the code for calculating MSE and RMSE, as required:

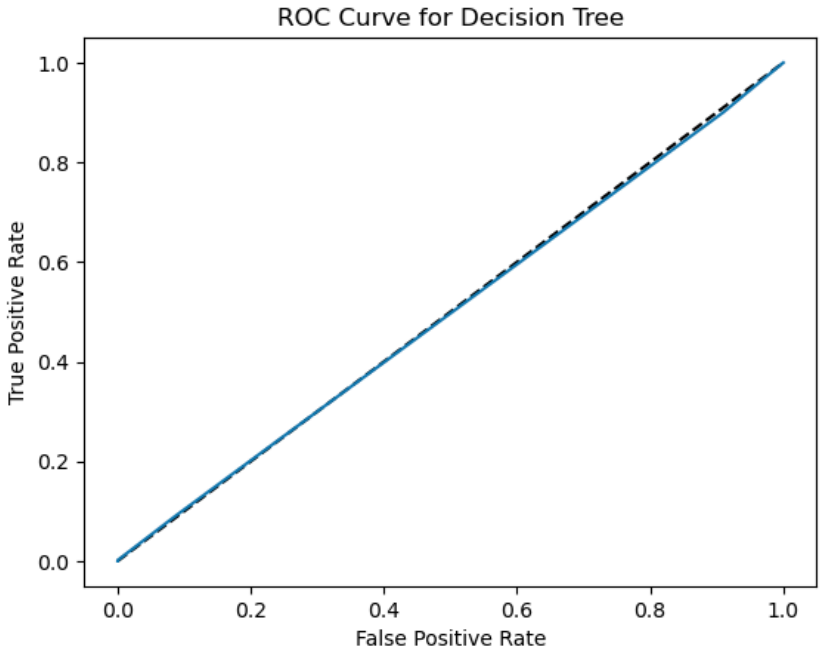
```
# Calculate MSE & RMSE

mse = mean_squared_error(y_test, y_pred)
rmse = mse**(1/2)

print("Mean squared error: ", mse)
print("Root mean squared error: ", rmse)
```

The output for calculating MSE & RMSE is as follows:
Mean squared error: 0.354
Root mean squared error: 0.5949789912257406

Next, the ROC curve is plotted to evaluate the model's performance by plotting the true positive rate on the y-axis against the false positive rate at numerous binary thresholds on the x-axis. This is shown in the code below:

```
# Generate ROC Curve
y_pred_prob = grid_search.best_estimator_.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Decision Tree')
plt.show()
```

The closer the ROC curve is to the top left corner of the plot, the better the model is at classifying the data. The AUC is also calculated, assessing how well the model distinguishes between non-readmissions and readmissions classes. An AUC value close to 0.50 means that the model performs similarly to a model with random classifications and an AUC close to 1 means that the model performs well (Bobbitt, 2021). Below is ROC curve and AUC score for the decision tree:




```
AUC: 0.5007052186177715
```

The ROC curve is almost diagonal and has an AUC score of 0.5007052186177715, meaning that the model has difficulties differentiating between the readmitted and non-admitted classes and essentially makes random guesses.

Next, a classification report with an assessment of the decision tree model is generated, which includes the metrics precision, recall, F1-score, and support. Below is the code and its output:

```
# Print classification Report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.65      1.00      0.78      1291
           1       1.00      0.00      0.00       709

    accuracy                           0.65      2000
   macro avg       0.82      0.50      0.39      2000
weighted avg       0.77      0.65      0.51      2000
```

The classification report shows that the model performs well for class 0 (not readmitted), but struggles to identify class 1 (readmitted). The precision for class 0 is 0.65, meaning that 65% of patients predicted as not readmitted are correct. The recall is 1.00, meaning all actual non-readmitted patients are correctly identified. The F1-score is 0.78, meaning that it performs relatively well since it is close to 1 (Bobbitt, 2022). For class 1 (readmitted), precision is at 1.00, meaning all patients predicted as readmitted are correct. However, recall is 0.00, meaning that the model fails to identify actual readmitted patients, and the F1-score is also 0.00, meaning that the model performs very poorly.

Section D3) Code Execution

Find optimal decision tree parameters:

```
# Define parameters for grid search
param_grid = {
        'max_depth': [2, 4, 6, 8, 10, 12],
        'min_samples_leaf': [1, 2, 5, 10, 20]
}

# Code adapted from:
# Navlani, A. (2024). Decision Tree Classification in Python Tutorial. DataCamp.
# https://www.datacamp.com/tutorial/decision-tree-classification-python

# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Perform grid search
grid_search = GridSearchCV(clf, param_grid, cv=5)

# Train the model on the training data using grid search
grid_search.fit(X_train, y_train)
```

```
# Predict based on best model
y_pred = grid_search.predict(X_test)

# Print the best hyperparameters found by grid search
print("Best hyperparameters:", grid_search.best_params_)
```

Fit decision tree model with hyperparameters:

```
# Fit the decision tree model with the found parameters
dt = DecisionTreeClassifier(max_depth = 2, min_samples_leaf = 1)
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)
```

Calculate model accuracy:

```
# Calculate the accuracy for best model
accuracy = accuracy_score(y_test, y_pred)
# Calculate the accuracy for training dataset
training_accuracy = accuracy_score(y_train, grid_search.predict(X_train))

print("Accuracy for best model:", accuracy)
print("Training Accuracy:", training_accuracy)
```

Plot decision tree:

```
# Plot the decision tree
plt.figure(figsize=(20,10), dpi=200)
plot_tree(grid_search.best_estimator_, filled=True, fontsize=16, feature_names=X.columns)
plt.show()
```

Create confusion matrix:

```
# Create confusion matrix
print('The confusion matrix for decision tree model')
print(confusion_matrix(y_test, y_pred))
```

Calculate MSE & RMSE:

```
# Calculate MSE & RMSE
mse = mean_squared_error(y_test, y_pred)
rmse = mse**(1/2)
print("Mean squared error: ", mse)
print("Root mean squared error: ", rmse)
```

Create ROC curve & calculate AUC:

```
# Generate ROC Curve
y_pred_prob = grid_search.best_estimator_.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Decision Tree')
plt.show()

# Calculate and print AUC
auc = roc_auc_score(y_test, y_pred)
print("AUC:", auc)
```

Generate classification report:

```
# Print classification report
print(classification_report(y_test, y_predicted))
```

Part V: Data Summary and Implications

Section E1) Accuracy and MSE

The accuracy score for the decision tree model is 64.6%, meaning that the model classified 64.6% of the instances in the test dataset. The accuracy score is likely influenced by the model's ability to identify non-readmissions effectively as opposed to readmissions. However, no accuracy measurement for the prediction model is needed for this analysis.

MSE is a metric used to evaluate models with continuous target variables. Since the target variable in the analysis is categorical (patient readmissions: yes or no), MSE is not a suitable evaluation metric (Straw, n.d.). As required, the MSE was still calculated and determined to be at 0.354. Since the MSE is relatively low, the model's predictions are closer to the actual values.

Instead, the precision, recall, and F1-score are more suitable for evaluating the model's ability to predict and distinguish between readmitted and readmitted patients. Based on the results from the classification report, the model correctly identifies non-admitted patients but struggles with identifying readmitted patients. The precision for non-readmissions is at 0.65, meaning that 65% of predicted non-readmissions are correct. The recall is 1.00, meaning that the model accurately

identifies all actual non-readmissions, and has an F1-score of 0.78, meaning that the class performs strongly. For the readmitted class, the precision is 1.00 meaning that all predicted readmissions are correct.

Section E2) Results and Implications

The prediction analysis reveals that the decision tree model does not identify readmitted patients. The accuracy score is 64.6%, meaning the model has moderate performance. However, the AUC score of 0.50 means that the model does poorly at differentiating between readmitted and non-readmitted patients and performs no better than a model with random guessing. The confusion matrix further displays this since the model correctly classifies most non-readmitted patients with 1,291 true negatives but fails to identify almost all readmitted patients with 708 false negatives and only one true positive.

The classification report shows this as well. The model performs well for class 0 (not readmitted) with a recall of 1.00, meaning it correctly identifies all non-admitted patients. However, it fails to do so for class 1 (readmitted), with a recall of 0.00, meaning it does not identify any readmitted cases. The results of the AUC, confusion matrix, and classification report show that the model is biased toward predicting non-readmission.

The decision tree visual shows that income is the most influential factor in predicting readmissions because it splits on a certain income threshold. This suggests that some lower-income patients are more likely to be readmitted to the hospital within one month of initial release. However, the depth of the tree is limited at 2, so the model could be oversimplified.

Section E3) Limitation

One limitation of the data analysis is class imbalance, where the model is better at predicting non-readmissions (the majority class) than readmissions. This is evident in the classification report and the confusion matrix. Therefore, the decision tree may not be able to accurately predict patients who are at risk of being readmitted. This limitation can be addressed by using another model, like a random forest.

Section E4) Course of Action

The decision tree model should not be used to evaluate patient readmissions due to its bias toward predicting non-readmissions. Instead, hospitals can explore other models, such as random forest, to improve accuracy in predicting readmissions. Hospitals should implement post-discharge monitoring through follow-up calls or home visits to reduce readmissions. Since the model suggests that income influences readmissions, financial assistance, and medication

support programs should be provided to lower-income patients to help reduce their risk of returning to the hospital.

Hospitals can develop a program to reduce readmissions by creating personalized discharge plans, ensuring that patients understand what is required to fully recover (consistent medication regime, adhering to post-care instructions, follow-up appointments). In addition, making communication between patients and doctors seamless and fast can help prevent avoidable readmissions. Until a more reliable model is developed, hospitals and their staff must rely on clinical judgment to assess patients' readmission risks.

Part VI: Demonstration

Section F) Panopto Demonstration

Panopto recording is found at this link:
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=950bbea9-9be8-458f-98fd-b28a017e845d

Section G) Sources of Third-Party Code

Elleh, K. (n.d.). *Welcome to D209 Data Mining 1* [Lecture slides]. Western Governors University. Slide 35.

Navlani, A. (2024). *Decision Tree Classification in Python Tutorial*. DataCamp. https://www.datacamp.com/tutorial/decision-tree-classification-python

Section H) Sources

Adison, J. J. (2024, June 13). *High Correlation in Decision Tree Models: Why It Matters*. Medium. Retrieved February 19, 2025, from https://medium.com/@yot181/high-correlation-in-decision-tree-models-why-it-matters-abstract-9f4415fba73a

Anshul. (2025, February 17). *Decision Tree*. Analytics Vidhya. Retrieved February 19, 2025, from

https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/#h-decision-tree-assumptions

Bobbitt, Z. (2021, August 9). *How to Interpret a ROC Curve (With Examples)*. Statology. Retrieved February 20, 2025, from https://www.statology.org/interpret-roc-curve/

Bobbitt, Z. (2021, September 30). *MSE vs. RMSE: Which Metric Should You Use?* Statology. Retrieved February 20, 2025, from https://www.statology.org/mse-vs-rmse/

Bobbitt, Z. (2022, May 9). *How to Interpret the Classification Report in sklearn (With Example)*. Statology. Retrieved February 20, 2025, from https://www.statology.org/sklearn-classification-report/

Dorfer, T. A. (2023, January 29). *Decision Trees. Part 4: Gini Index | by om pramod*. Medium. Retrieved February 20, 2025, from https://medium.com/@ompramod9921/decision-trees-91530198a5a5

Elleh, F. (n.d.). *Welcome to D209 Data Mining 1* [Lecture Slides]. Western Governors University. Slide 83.

IBM. (n.d.). *What is a Decision Tree?* IBM. Retrieved February 19, 2025, from https://www.ibm.com/think/topics/decision-trees

Jain, S. (2025, January 28). *Data Preprocessing in Data Mining*. GeeksforGeeks. Retrieved February 19, 2025, from https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/

Noble Desktop. (2024, August 22). *Training vs. Testing Data: Understanding the Importance*. Noble Desktop. https://www.nobledesktop.com/learn/python/training-vs--testing-data-understanding-the-importance

Saturn Cloud. (2023, July 26). *What Is 'random_state' in sklearn.model_selection.train_test_split Example?* Saturn Cloud. https://saturncloud.io/blog/what-is-randomstate-in-sklearnmodelselectiontraintestsplit-example/

Straw, E. (n.d.). Dr. Straw's D209 supplemental material. Western Governors University. https://srm--c.vf.force.com/apex/CourseArticle?id=kA0S60000000mELKAY

Western Governors University. (n.d.). *Medical Data Considerations and Dictionary*.