Data Mining
Task 2- Predictive Analysis

D209 WGU

TRESA (TESSIE) AUSTIN

Part I: Research Question

A. Define the research question and describe the purpose of this data analysis:

1. Readmission rates in hospitals are a significant concern as the impact to patient outcomes, costs, and overall quality of care are impacted. A critical objective for any hospital is to reduce readmission rates since it could be tied to regulatory and reimbursement policies. Based on that it's important to research the following question:

   Using a decision tree model, can patients who will be readmitted within 30 days of initial admission be identified?

2. Define the goals of the data analysis.
   Building on the analysis done around the kNN analysis previously, it was found that the model performed well in identifying readmissions but there was still room for improvement. The precision of the kNN analysis was 80% for readmission, so 20% of the patients predicted to be readmitted were not. Taking that a step further by using decision tree, the goal here to see if we can provide better context and more useful information for hospitals to manage patient care.

Part II: Method Justification

B. Explain the reasons for your chosen classification method from part A1 by doing the following:

1. The choice to use a decision tree for this type of analysis has several advantages as found on Simplilearn.com(2023):
   a. Interpretability: They are easy to understand and visualize since they are based on what they are named after, a tree. This type of transparency is crucial in healthcare settings to help in understanding reasoning behind factors related to patients readmission rates.
   b. Handling Data Types: The ability to manage both numerical and categorical data is huge factor. Since the data we are working with has variables that are not linear, using a decision tree will be helpful here.
   c. Scenario Analysis: Determining the best, worst, and expected outcomes are part of the analysis.
   d. Minimal Data Preparation: This requires little preprocessing and normalization.
   e. Robustness: Decision trees are flexible and can be applied to both categorical and continuous data, this is the type of data we have in our dataset.
2. One assumption around decision trees is that they are easy to understand and interpret by humans. Explorium (2023) states that a decision tree algorithm is one of the easiest machine learning algorithms since there is not much math involved and it's one of the most widely used methods in machine learning. Visually it represent a structure that humans are familiar with as well.

3. Below are the packages and libraries being used in Python that supports the analysis:

   - Pandas
     - Provides data structures such as DataFrame which provides ease in data loading. Pandas also provides essential processing steps around evaluating data for missing values, encoding categorical variables, ad splitting dataset (GeeksforGeeks, 2023).
   - Matplotlib.pyplot
     - Matplotlib is effectively used to visualize data, model results and performance metrics to help understand visualizations (Tutorialspoint, n.d.).
   - Sklearn.metrics import confusion_matrix, accuracy_score, classification_report, mean_squared_error

    - According the scikit_learn.org (2024) the following were found:
        - The confusion_matrix evaluates the accuracy of the classification.
        - The accuracy_score is a multilabel classification that computes subset accuracy.
        - The classification_report provides a detailed summary of the classification performance of a model and includes several key metrics for evaluating the quality of the predictions.
        - The mean_squared_error assesses the average squared difference between the observed and predicted values. It is usually greater than zero since is a squared value and usually positive. Frost (2021) advises that a model with less error produces more precise predictions.
- Sklearn.model_selection import train_test_split, GridSearchCV
    - The key role of "train_test_split" is to create a division of data into two subsets which is crucial for evaluating the performance of machine learning models (Stojiljković, 2020**).**
    - The GridSearchCV implements a fit and score method that applies to the parameters of the estimator and then is optimized by a cross-validated grid search.
- Sklearn.tree import DecisionTreeClassifier, plot_tree
    - DecisionTreeClassifier, according to scikit_learn.org (2024) is a class capable of performing multi-class classification on a dataset by taking two arrays and holding the training samples in one shape and the class labels for training samples in another shape. If there are classes with the same and highest probability, the classifier will predict the class with the lowest index.
    - Plot_tree is a decision tree visualizer added to allow easily reproducable figures of the tree (Płoński, 2024)
- Sklearn.preprocessing import LabelEncoder
    - Per an article on geeksforgeeks.org (2023) the label encoder is used to convert categorical columns into numerical ones so they can be bitted by machine learning models which only take numerical data. This is a pre-processing step needed to address the variables that will be used.

Part III: Data Preparation

C. Perform data preparation for the chosen data set by doing the following:

1. One data preprocessing goal for using a decision tree is to address variables that are categorical using one hot encoding. Using a binary structure to replace categorical values will allow the analysis to be completed.
2. The initial data set variables that will be used to perform the analysis

ReAdmis	Categorical (Target)
Population	Numeric
Children	Numeric
Age	Numeric
Income	Numeric
Gender	Categorical
VitD_levels	Numeric

Doc_visits	Numeric
Full_meals_eaten	Numeric
vitD_supp	Numeric
Initial_admin	Categorical
HighBlood	Categorical
Stroke	Categorical
Overweight	Categorical
Diabetes	Categorical
Asthma	Categorical
Services	Categorical
Initial_days	Numeric
TotalCharge	Numeric
Additional_charges	Numeric

3. The steps used to prepare the data for the analysis with code
    1. Data will be loaded

```
# Set the correct file path for the medical data
medical_file_path = "medical_clean.csv"
# Read the medical data file with keep_default_na
df = pd.read_csv(medical_file_path, keep_default_na=False, index_col=0)
```

    2. A check of duplicates will be performed

```
# Code to check for duplicates
has_duplicates = df.duplicated().any()
print("Duplicates present:", has_duplicates)
```

Duplicates present: False

    3. A check for missing values will be performed.

```
# Check for missing data
missing_data = df.isnull().sum()

# Display the missing data counts
print("Missing data counts:")
print(missing_data)
```

Missing data counts:
Customer_id 0
Interaction 0
UID 0
City 0
State 0
County 0
Zip 0

```
Lat                 0
Lng                 0
Population          0
Area                0
TimeZone            0
Job                 0
Children            0
Age                 0
Income              0
Marital             0
Gender              0
ReAdmis             0
VitD_levels         0
Doc_visits          0
Full_meals_eaten    0
vitD_supp           0
Soft_drink          0
Initial_admin       0
HighBlood           0
Stroke              0
Complication_risk   0
Overweight          0
Arthritis           0
Diabetes            0
Hyperlipidemia      0
BackPain            0
Anxiety             0
Allergic_rhinitis   0
Reflux_esophagitis  0
Asthma              0
Services            0
Initial_days        0
TotalCharge         0
Additional_charges  0
Item1               0
Item2               0
Item3               0
Item4               0
Item5               0
Item6               0
Item7               0
Item8               0
dtype: int64
```

4. Then after those checks, a look at data types and an inspection of the DataFrame needs to be completed.

```
# Display data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

Index: 10000 entries, 1 to 10000
Data columns (total 49 columns):

#	Column	Non-Null Count	Dtype
0	Customer_id	10000 non-null	object
1	Interaction	10000 non-null	object
2	UID	10000 non-null	object
3	City	10000 non-null	object
4	State	10000 non-null	object
5	County	10000 non-null	object
6	Zip	10000 non-null	int64
7	Lat	10000 non-null	float64
8	Lng	10000 non-null	float64
9	Population	10000 non-null	int64
10	Area	10000 non-null	object
11	TimeZone	10000 non-null	object
12	Job	10000 non-null	object
13	Children	10000 non-null	int64
14	Age	10000 non-null	int64
15	Income	10000 non-null	float64
16	Marital	10000 non-null	object
17	Gender	10000 non-null	object
18	ReAdmis	10000 non-null	object
19	VitD_levels	10000 non-null	float64
20	Doc_visits	10000 non-null	int64
21	Full_meals_eaten	10000 non-null	int64
22	vitD_supp	10000 non-null	int64
23	Soft_drink	10000 non-null	object
24	Initial_admin	10000 non-null	object
25	HighBlood	10000 non-null	object
26	Stroke	10000 non-null	object
27	Complication_risk	10000 non-null	object
28	Overweight	10000 non-null	object
29	Arthritis	10000 non-null	object
30	Diabetes	10000 non-null	object
31	Hyperlipidemia	10000 non-null	object
32	BackPain	10000 non-null	object
33	Anxiety	10000 non-null	object
34	Allergic_rhinitis	10000 non-null	object
35	Reflux_esophagitis	10000 non-null	object
36	Asthma	10000 non-null	object
37	Services	10000 non-null	object
38	Initial_days	10000 non-null	float64
39	TotalCharge	10000 non-null	float64
40	Additional_charges	10000 non-null	float64
41	Item1	10000 non-null	int64
42	Item2	10000 non-null	int64
43	Item3	10000 non-null	int64
44	Item4	10000 non-null	int64
45	Item5	10000 non-null	int64

```
 46  Item6           10000 non-null  int64
 47  Item7           10000 non-null  int64
 48  Item8           10000 non-null  int64
dtypes: float64(7), int64(15), object(27)
memory usage: 3.8+ MB
```

```
# Visually inspect df
pd.set_option("display.max_columns", None)
df.head(5)
```

CaseOrder	Customer_id	Interaction	UID	City	State	County	Zip	Lat	Lng	Population	Area	TimeZone	Job	Children	Age	Income	
1	C412403	8cd49b13-f45a-4b47-a2bd-173ffa932c2f	3a83ddb66e2ae73798bdf1d705dc0932	Eva	AL	Morgan	35621	34.34960	-86.72508	2951	Suburban	America/Chicago	Psychologist, sport and exercise	1	53	86575.93	I
2	Z919181	d2450b70-0337-4406-bdbb-bc1037f1734c	176354c5eef714957d486009feabf195	Marianna	FL	Jackson	32446	30.84513	-85.22907	11303	Urban	America/Chicago	Community development worker	3	51	46805.99	
3	F995323	a2057123-abf5-4a2c-abad-8ffe33512562	e19a0fa00aeda885b8a436757e889bc9	Sioux Falls	SD	Minnehaha	57110	43.54321	-96.63772	17125	Suburban	America/Chicago	Chief Executive Officer	3	53	14370.14	V
4	A879973	1dec528d-eb34-4079-adce-0d7a40e82205	cd17d7b6d152cb6f23957346d11c3f07	New Richland	MN	Waseca	56072	43.89744	-93.51479	2162	Suburban	America/Chicago	Early years teacher	0	78	39741.49	
5	C544523	5885f56b-d6da-43a3-8760-83583af94266	d2f0425877b10ed6bb381f3e2579424a	West Point	VA	King William	23181	37.59894	-76.88958	5287	Rural	America/New_York	Health promotion specialist	1	22	1209.56	V

5. Convert needed variables for decision tree analysis. For some variables scikit-learn.org (2024) advises that LabelEncoder is a way to transform non-numerical labels to numeral labels. Variables such as Gender will be converted with One hot encoding to ensure needed values for our decision tree. Each of the needed variables will be addressed below. A check for missing values after this step was performed to ensure all data aligns for the needed analysis.

```
# Label encode binary categorical columns
label_encoder = LabelEncoder()
binary_columns = ['ReAdmis', 'HighBlood', 'Stroke', 'Overweight', 'Diabetes', 'Asthma']
for col in binary_columns:
    df[f'{col}_encoded'] = label_encoder.fit_transform(df[col])

# Check for missing values after label encoding
print("Missing values after label encoding:")
print(df[[f'{col}_encoded' for col in binary_columns]].isnull().sum())
```

6. One hot encoding must be used for converting multi class categorical variables such as Initial_admin and Services. A reset of the indices before concatenation was done to ensure data aligns for the needed analysis.

```
# Initialize OneHotEncoder

one_hot_encoder = OneHotEncoder(sparse_output=False)


# Encode categorical columns
```

```
initial_admin_encoded = one_hot_encoder.fit_transform(df[['Initial_admin']])

initial_admin_encoded_df = pd.DataFrame(initial_admin_encoded,
columns=one_hot_encoder.get_feature_names_out(['Initial_admin']))


services_encoded = one_hot_encoder.fit_transform(df[['Services']])

services_encoded_df = pd.DataFrame(services_encoded,
columns=one_hot_encoder.get_feature_names_out(['Services']))


gender_encoded = one_hot_encoder.fit_transform(df[['Gender']])

gender_encoded_df = pd.DataFrame(gender_encoded,
columns=one_hot_encoder.get_feature_names_out(['Gender']))
```

7. Combine the original data frame with the one hot encoded columns using concatenation. Another check for any missing values was done here as well. If any missing values are found, they will be handled based on the data type and a subsequent check for missing values is completed after handling.

```
# Concatenate the encoded DataFrames
df_combined = pd.concat([df, initial_admin_encoded_df, services_encoded_df,
gender_encoded_df], axis=1)

# Check for missing values after concatenation
print("Missing values after concatenation:")
print(df_combined.isnull().sum())

# Handle missing values (if any)
# Fill missing values for numeric columns
numeric_columns = df_combined.select_dtypes(include=['number']).columns
df_combined[numeric_columns] =
df_combined[numeric_columns].fillna(df_combined[numeric_columns].median())

# Fill missing values for non-numeric columns
non_numeric_columns = df_combined.select_dtypes(exclude=['number']).columns
df_combined[non_numeric_columns] = df_combined[non_numeric_columns].fillna('Unknown')

# Check for missing values after handling
print("Missing values after handling:")
print(df_combined.isnull().sum())
```

8. Create a new dataframe, tree_df, to hold the converted variables and do a final check for any missing values.

```
# Drop original categorical columns
df_combined.drop(['ReAdmis', 'Gender', 'Initial_admin', 'HighBlood', 'Stroke', 'Overweight',
'Diabetes', 'Asthma', 'Services'], axis=1, inplace=True, errors='ignore')
```

```
# Select columns for the decision tree model
selected_columns = [
   'Population', 'Children', 'Age', 'Income', 'VitD_levels', 'Doc_visits', 'Full_meals_eaten',
'vitD_supp',
   'Initial_days', 'TotalCharge', 'Additional_charges'
] + list(initial_admin_encoded_df.columns) + list(services_encoded_df.columns) +
list(gender_encoded_df.columns) + [f'{col}_encoded' for col in binary_columns if col !=
'ReAdmis']

# Create new DataFrame for the decision tree
tree_df = df_combined[selected_columns].assign(ReAdmis=df['ReAdmis_encoded'])

# Check for missing values in the final DataFrame
print("Missing values in the final DataFrame:")
print(tree_df.isnull().sum())
```

9. Inspect new dataframe and save a copy of the data set.

```
# Display the final DataFrame
print(tree_df.head())
# Save tree_df to a CSV file
tree_df.to_csv("tree_df.csv", index=False)
print("tree_df has been saved to tree_df.csv'.")
```

4. Provide a copy of the cleaned data set.

Cleaned data set saved as tree_df.csv.

Part IV: Analysis

D. Perform the data analysis and report on the results by doing the following:

1. DataCamp (2024) advises that classification is a two-step process, a learning, and a prediction step. In the learning step, the model is created based on given training data. Whereas in the prediction step, the model is used to predict the response to given data. The following code is used to split **tree_df.csv** into training/test sets

```
# Ensure there are no missing values in the target column
tree_df = tree_df.dropna(subset=['ReAdmis'])

# Separate features and target variable
X = tree_df.drop('ReAdmis', axis=1)
y = tree_df['ReAdmis']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the decision tree model
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
```

```
# Combine the training features and target into a single DataFrame
train_tree_df = pd.concat([X_train, y_train], axis=1)
test_tree_df = pd.concat([X_test, y_test], axis=1)

# Save the train and test DataFrames to CSV files
train_tree_df.to_csv("train_tree_df.csv", index=False)
test_tree_df.to_csv("test_tree_df.csv", index=False)

print("train_tree_df has been saved to 'train_tree_df.csv'.")
print("test_tree_df has been saved to 'test_tree_df.csv'.")
```

The files for the training and test sets are saved as train_tree_df.csv and test_tree_df.csv.

2. The Decision Tree Classification technique is used to appropriately analyze the data and employs a supervised learning algorithm that is used for both classification and regression tasks(Vishnani, 2021). In this instance it's being used for classification by using a top-down approach where the root node is at the top of the structure and the outcomes are represented by the leaves. Using the DecisionTreeClassifier with tree depth, number of leaves, and feature importance gives a true picture of the analysis and using plot_tree we can get a picture of the tree.

   First we need to determine the number of nodes in the tree, the depth of the tree, leaves on the tree. Nodes are the building blocks of the tree which are broken down into root (the topmost node), internal or children which split the data based on different features. Leaves or terminal nodes define the end point of the tree where no further splitting occurs and represent the predicted output(Explorium, 2023). Depth of a node refers to the level it is in the tree, the root has a depth of zero. Depth represents the number of splits or decisions required to reach a leaf node. Ronaghan (2018) states that feature importance is calculated as the decrease node impurity weighted by the probability of reaching that node. The higher the value the more important the feature is.

```
# Determine the number of nodes in the tree
num_nodes = decision_tree.tree_.node_count
print(f"Number of nodes in the tree: {num_nodes}")

# Determine the depth of the tree
tree_depth = decision_tree.get_depth()
print(f"Depth of the tree: {tree_depth}")

# Determine the number of leaves on the tree
num_leaves = decision_tree.get_n_leaves()
print(f"Number of leaves in the tree: {num_leaves}")
```

```
Number of nodes in the tree: 255
Depth of the tree: 15
Number of leaves in the tree: 128
```

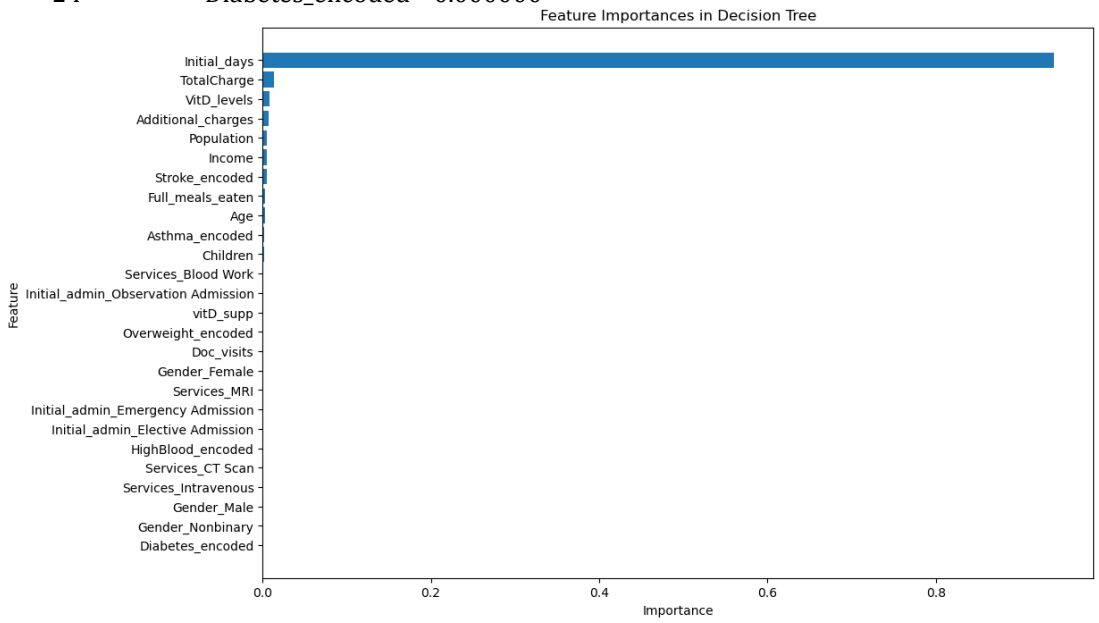
To better visualize the decision tree, we need to determine feature importances and create a plot of the tree. Feature importance is a technique that assigns a score to each feature based on it's usefulness in predicting the target variable. This is calculated by looking at how much each feature decreases the impurity in the nodes where it is used to make splits(GeeksforGeeks, 2023)

```
# Determine feature importances
feature_importances = decision_tree.feature_importances_
for feature, importance in zip(X.columns, feature_importances):
    print(f"Feature: {feature}, Importance: {importance}")

# Create a DataFrame for better visualization
importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

# Sort the DataFrame by importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)
print(importance_df)

# Plot the feature importances
plt.figure(figsize=(12, 8))
plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importances in Decision Tree')
plt.gca().invert_yaxis()
plt.show()
```

```
Feature: Population, Importance: 0.005543140205777189
Feature: Children, Importance: 0.001619391500343881
Feature: Age, Importance: 0.002908588869456267
Feature: Income, Importance: 0.004827185649213546
Feature: VitD_levels, Importance: 0.00801481488313941
Feature: Doc_visits, Importance: 0.0008540031380562946
Feature: Full_meals_eaten, Importance: 0.0029661632955967927
Feature: vitD_supp, Importance: 0.0010735629547510724
Feature: Initial_days, Importance: 0.9404234805333056
Feature: TotalCharge, Importance: 0.01311014283908253
Feature: Additional_charges, Importance: 0.006857759995673881
Feature: Initial_admin_Elective Admission, Importance: 0.0002681252681252681
Feature: Initial_admin_Emergency Admission, Importance: 0.00035750035750035746
Feature: Initial_admin_Observation Admission, Importance: 0.001147677618265853
Feature: Services_Blood Work, Importance: 0.0012065637065637063
Feature: Services_CT Scan, Importance: 0.0
Feature: Services_Intravenous, Importance: 0.0
Feature: Services_MRI, Importance: 0.00036660523551909293
Feature: Gender_Female, Importance: 0.0004989549162481496
Feature: Gender_Male, Importance: 0.0
Feature: Gender_Nonbinary, Importance: 0.0
Feature: HighBlood_encoded, Importance: 0.0002681252681252681
Feature: Stroke_encoded, Importance: 0.004723204316402864
```

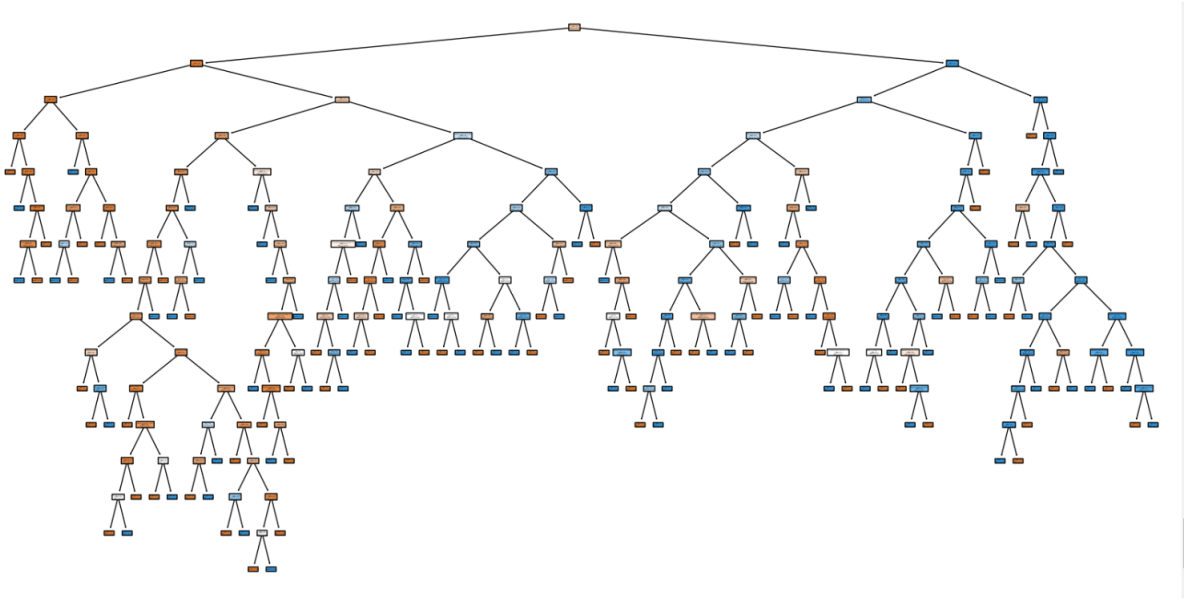
Feature: Overweight_encoded, Importance: 0.0010266676933343595
Feature: Diabetes_encoded, Importance: 0.0
Feature: Asthma_encoded, Importance: 0.0019383417555186265

	Feature	Importance
8	Initial_days	0.940423
9	TotalCharge	0.013110
4	VitD_levels	0.008015
10	Additional_charges	0.006858
0	Population	0.005543
3	Income	0.004827
22	Stroke_encoded	0.004723
6	Full_meals_eaten	0.002966
2	Age	0.002909
25	Asthma_encoded	0.001938
1	Children	0.001619
14	Services_Blood Work	0.001207
13	Initial_admin_Observation Admission	0.001148
7	vitD_supp	0.001074
23	Overweight_encoded	0.001027
5	Doc_visits	0.000854
18	Gender_Female	0.000499
17	Services_MRI	0.000367
12	Initial_admin_Emergency Admission	0.000358
11	Initial_admin_Elective Admission	0.000268
21	HighBlood_encoded	0.000268
15	Services_CT Scan	0.000000
16	Services_Intravenous	0.000000
19	Gender_Male	0.000000
20	Gender_Nonbinary	0.000000
24	Diabetes_encoded	0.000000



 # Visualize the decision tree

```
plt.figure(figsize=(20,10))
plot_tree(
    decision_tree,
    feature_names=list(X.columns),
    class_names=["Class 0", "Class 1"],
    filled=True,
    rounded=True,
    proportion=True,
    precision=2
)
plt.show()
```



Based on the above plot of the decision tree, the tree is quire deep which means that it's capturing a lot of detail from the training data and could be overfilling. Using a grid search approach to tuning hyperparameter will help build a model and evaluate for every combination of algorithms per grid (Saini, 2020). The grid search takes the model that is being trained and different values of the hyperparameters to perform calculations of errors for various values. Using a random set of values for the grid we can perform our search with cross validation to fine the best options and train a "pruned" decision tree for analysis.

```
# Define the parameter grid
param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=decision_tree, param_grid=param_grid, cv=5, n_jobs=-1,
scoring='accuracy')
grid_search.fit(X_train, y_train)
```

```
# Get the best parameters and the best model
best_params = grid_search.best_params_
best_decision_tree = grid_search.best_estimator_

# Train the pruned decision tree on the entire training set
best_decision_tree.fit(X_train, y_train)

# Make predictions
y_pred = best_decision_tree.predict(X_test)
```

```
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 10}
```

We can then evaluate the model for accuracy. Finding the mean squared error as well as calculating the confusion matrix and classification report give us more information than just a visual of the tree and what it shows us.

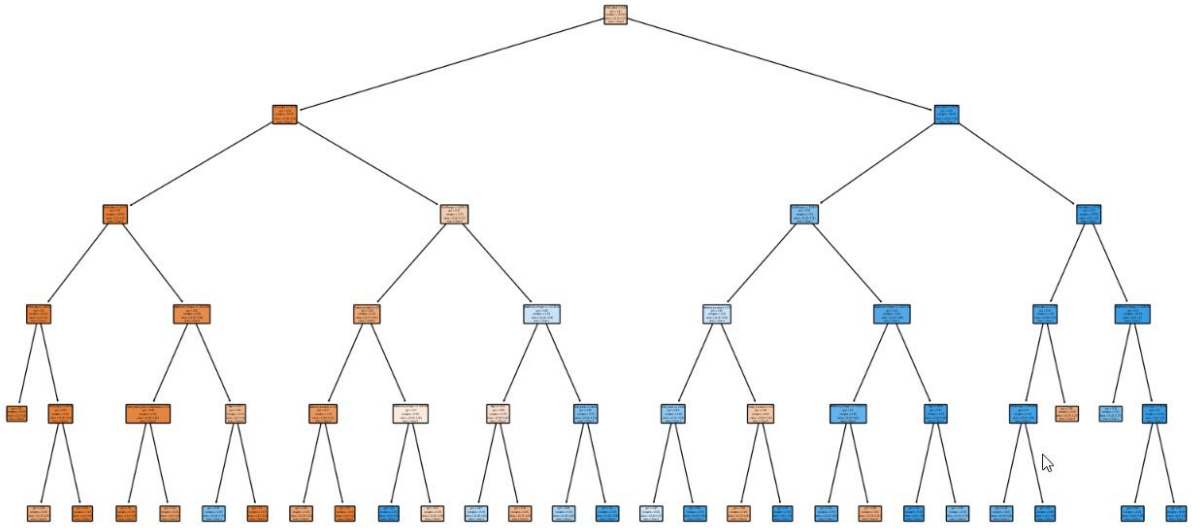
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
print("Mean Squared Error:", mse)
```

```
Accuracy: 0.977
Confusion Matrix:
 [[1273   18]
 [  28  681]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.98      1291
         1.0       0.97      0.96      0.97       709

    accuracy                           0.98      2000
   macro avg       0.98      0.97      0.97      2000
weighted avg       0.98      0.98      0.98      2000

Mean Squared Error: 0.023
```

Another visual of the decision tree can also be ran to “see” what we have found during our analysis.

```
# Visualize the pruned decision tree
plt.figure(figsize=(20,10))
plot_tree(
```

```
    best_decision_tree,
    feature_names=list(X.columns),
    class_names=["Class 0", "Class 1"],
    filled=True,
    rounded=True,
    proportion=True,
    precision=2
)
plt.show()
```



3. The code used to perform the above analysis is listed as D209_Austin_T_Task2_02.ipynb.

Part V: Data Summary and Implications

E. Summary of data analysis:

1. In the pruned analysis, accuracy was found to be 0.977 and the mean squared error (MSE) was found to be 0.023. The definition of accuracy is the ratio of correctly predicted observations to the total observations according to DataCamp(2024). 0.977 or 97.7% indicates the model is making a large portion of the correct predictions. It's able to distinguish between the classes (readmitted versus no readmitted). The mean squared error (MSE) is a low value at 0.023 which indicates that the difference between the predicted and actual values is small so the model as good accuracy.

2. Results and implications of these two measures show that we can use this analysis to help the hospital with knowing who will be readmitted. The accuracy measure of 0.977 or 97.7% indicates the model is making a sizable portion of the correct predictions. It's able to distinguish between the classes (readmitted versus no readmitted). The mean squared error (MSE) is a low value at 0.023 which indicates that the difference between the predicted and actual values is small so the model as good accuracy. Using these two together indicates that the model is reliable and precise.

3. One limitation found with the confusion matrix is that there is a small number of false positives and false negatives. This is a small limitation of the model and shows that it's fallible.

- **False Positives (FP): 18**: The model incorrectly predicted the positive class (1.0) 18 times.
- **False Negatives (FN): 28**: The model incorrectly predicted the negative class (0.0) 28 times.

4. Recommended course for the research question "Using a decision tree model, can patients who will be readmitted within 30 days of initial admission be identified?" we have shown that the model above is reliable and precise. Using the variables outlined here and moving down the decision tree can help the hospital administration make better decisions around patient care and how to use their dollars. Ensuring the information around feature importance found will also be important to ensure that we aren't barking up the wrong tree.

   The hospital can use this information to determine what factors "decide" if a patient is readmitted. Based on the feature importance, the initial admission length of stay is the most crucial factor followed by a patients total charge, their vitamin D levels and the cost of additional charges incurred. The relationship of length of initial stay with total charges and additional charges are all in line, as typically the longer you are in the hospital the more your bill is. A patients vitamin D level could be used to address readmission if more research is done to see what level of vitamin D deficiency triggers a readmission if there is such a correlation.

Part VI: Demonstration

F. Panopto video link

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=64cf10eb-7539-4474-9a16-b1a8000481cd

G. Web Sources

Płoński, P. (2020, June 22). Visualize decision tree. MLJAR. Retrieved June 26, 2024, from https://mljar.com/blog/visualize-decision-tree/

Scikit-learn developers. (2024). *sklearn.preprocessing.LabelEncoder*. scikit-learn. https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.LabelEncoder.html

Simplilearn. (2023). The best guide on how to implement decision tree in Python. Simplilearn. Retrieved June 26, 2024, from https://www.simplilearn.com/tutorials/machine-learning-tutorial/decision-tree-in-python

H. Sources

DataCamp. (2024). *Decision tree classification in Python*. DataCamp. https://www.datacamp.com/tutorial/decision-tree-classification-python

Explorium. (2023, August 6). The complete guide to decision trees. Explorium. Retrieved July 8, 2024, from https://www.explorium.ai/blog/machine-learning/the-complete-guide-to-decision-

trees/#:~:text=The%20main%20benefits%20of%20decision,can%20automatically%20handle%20missing%20values

Frost, J. (2021, November 12). *Mean squared error (MSE) - Statistics by Jim*. https://statisticsbyjim.com/regression/mean-squared-error-mse/

GeeksforGeeks. (2023, February 24). Gini impurity and entropy in decision tree ML. GeeksforGeeks. Retrieved July 8, 2024, from https://www.geeksforgeeks.org/gini-impurity-and-entropy-in-decision-tree-ml/

GeeksforGeeks. (n.d.). ML | Label Encoding of datasets in Python. GeeksforGeeks. Retrieved June 26, 2024, from https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

Repala, S. (2023, May 10). *A deep dive into decision tree algorithms: Classification, regression, and beyond*. Medium. https://medium.com/@satyarepala/a-deep-dive-into-decision-tree-algorithms-classification-regression-and-beyond-d2cf67d6f814

Ronaghan, S. (2018, May 11). The mathematics of decision trees, random forest, and feature importance in scikit-learn and Spark. Towards Data Science. Retrieved July 8, 2024, from https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e

Vishnani, A. (2021, March 17). An exhaustive guide to classification using decision trees. Towards Data Science. https://towardsdatascience.com/an-exhaustive-guide-to-classification-using-decision-trees-8d472e77223f