Healthcare Project - Patient Readmission

We aim to build predictive models, a Logistic Regression and a Decision Tree, to predict the likelihood of a patient being readmitted within 30 days of discharge. The prediction is based on three key features: age, insurance provider, and procedure cost. Let's explore these features and their potential roles:

- Age: Represents the patient's age, which may correlate with readmission rates. Younger or older patients might have a higher likelihood of readmission due to factors related to treatment and discharge processes.
- Insurance Provider: Indicates the company providing insurance coverage. This feature could influence readmission patterns due to variations in healthcare access or coverage policies.
- Procedure Cost: The cost associated with each procedure, which might reflect the complexity or intensity of the treatment, potentially affecting readmission likelihood.

1. Load Data

```
import pandas as pd

path = "C:/Users/rvrei/Documents/Healthcare_df.csv"
healthcare_df = pd.read_csv(path)
healthcare_df.head()
```

	claim_id	patient_id	procedure_id	claim_date	claim_amount	claim_status	insurance_provider	procedure_type	pr
0	CLM0001	PAT0001	41	2024-03-29	1997.79	approved	Blue Shield	CT Scan	
1	CLM0001	PAT0001	41	2024-03-29	1997.79	approved	Blue Shield	CT Scan	
2	CLM0016	PAT0016	41	2023-09-16	1080.34	approved	Aetna	MRI	
3	CLM0016	PAT0016	41	2023-09-16	1080.34	approved	Aetna	MRI	
4	CLM0004	PAT0004	14	2023-02-03	3073.56	approved	Blue Shield	Lab Test	

5 rows × 21 columns

2. Predictive Logistic Regression Model: Patient Readmission

Build a Logistic Regression model to predict the readmission of patients based on features likes ages, insurance provider, and procedure cost.

```
# Modules
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report


# Prepare features and target
X = healthcare_df[['age', 'insurance_provider','procedure_cost']]
y = healthcare_df['readmission'] # Yes/No

# Convert y to numeric values (i.e., 0 and 1, where 'No' = 0 and 'Yes' = 1)
y = healthcare_df['readmission'].apply(lambda x: 1 if x=='Yes' else 0)

# Encoding categorical variables: 'insurance_provider'
X = pd.get_dummies(X, drop_first=True)

# Splitting the dataset 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)

# Build a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Making predictions and evaluating the model
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         3
           1       1.00      0.57      0.73         7

    accuracy                           0.70        10
   macro avg       0.75      0.79      0.70        10
weighted avg       0.85      0.70      0.71        10
```

2.1 Logistic Regression Results

The model attempts to predict whether a patient will be readmitted within 30 days ("Yes") or not ("No") based on the features age, insurance provider, and procedure cost. Here's a breakdown of the results.

Class "No" (Not Readmitted)

> Precision: 0.50. Out of all predictions labeled as "No," only 50% were correct. Low precision indicates a significant number of false positives (cases predicted as "No" but were actually "Yes").
> Recall: 1.00. The model correctly identified all actual "No" cases. No false negatives for this class.
> F1-Score: 0.67. A balanced measure of precision and recall. While recall is perfect, precision is poor, which lowers the F1-score.

Class "Yes" (Readmitted)

> Precision: 1.00. Out of all predictions labeled as "Yes," the model was correct 100% of the time.
> Recall: 0.57. The model identified 57% of the actual "Yes" cases. 43% were missed (false negatives).
> F1-Score: 0.73. Reasonable performance for this class, driven by perfect precision but moderate recall.

```
Accuracy: 70%
    The model correctly predicted 7 out of 10 cases.
```

```
Macro Average:
    Precision (0.75): Average precision for both classes, treating them equally.
    Recall (0.79): Average recall for both classes, treating them equally.
    F1-Score (0.70): Average F1-score across both classes.

Weighted Average:
    Metrics weighted by the number of actual instances of each class.
    These metrics are more useful when dealing with imbalanced datasets, where one class dominates.
```

2.2 Interpretation of Results

- Interpretation of the Model
  - Class-Level Insights: The model performs better for class "Yes" (readmitted) in terms of precision, meaning it rarely falsely predicts readmission when there isn’t one. However, the recall for "Yes" is only 57%, meaning the model fails to identify nearly half of the actual readmissions.
  - Class Imbalance: Class "Yes" has more instances (support = 7) compared to class "No" (support = 3). This imbalance could contribute to the model being biased towards the majority class.
- Business Interpretation
  - High Precision for "Yes" (Readmitted): The model can be trusted to predict readmissions without many false positives. This is important when focusing resources on preventing readmission.
  - Low Recall for "Yes": The model misses 43% of patients who will actually be readmitted. If readmission prevention is a priority, this is a significant issue. To improve recall, consider adjusting the decision threshold or using other techniques (e.g., oversampling minority class).
  - Class "No": The model identifies all patients who won’t be readmitted but often predicts "No" incorrectly when the actual label is "Yes" (low precision).

- Steps to Improve the Model
    - Address Class Imbalance: Use techniques like SMOTE (Synthetic Minority Oversampling Technique) or class weighting in logistic regression to balance the classes.
    - Feature Engineering: Include additional features such as length of stay, comorbidities, or previous readmission history, which may have predictive power.
    - Threshold Adjustment: Lower the threshold for predicting "Yes" to improve recall at the expense of precision.
    - Try Alternative Models: Decision Trees or ensemble methods like Random Forests or Gradient Boosting might capture non-linear relationships better than logistic regression.
- Conclusion: The model has high precision but low recall for predicting readmissions. It is good at avoiding false positives but misses a significant number of true positives. Focus on improving recall for class "Yes" if the goal is to identify as many readmissions as possible for targeted intervention.

3. Model Visualization: Logistic Regression

To visualize the Logistic Regression model and its predictions, we can create plots to better understand how the model performs and how it relates to the input features.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, accuracy_score
from sklearn.linear_model import LogisticRegression
```

3.1 Confusion Matrix

A confusion matrix provides a detailed breakdown of the model's predictions compared to the actual values. It helps identify where the model's errors occur. Ideally, the matrix should show high counts for true positives and true negatives, and low counts for false positives and false negatives.

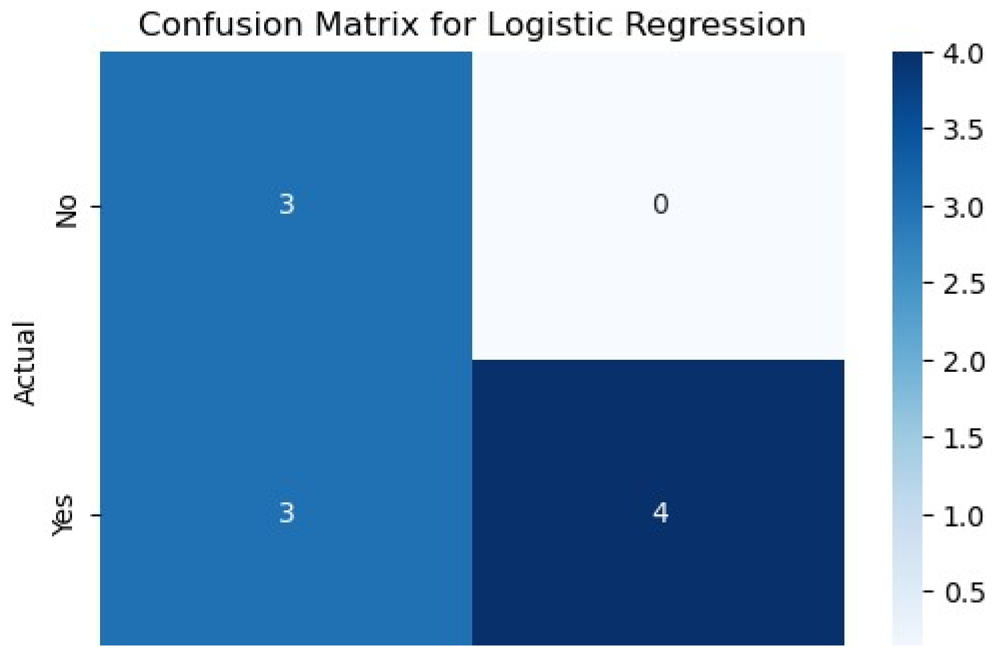
A heatmap of the confusion matrix visually represents the model's performance, highlighting the counts for true positives, false positives, true negatives, and false negatives.

```
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Logistic Regression')

# Calculate and print accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

plt.show()
```

```
Accuracy: 0.7
```



This heatmap visualizes the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). It helps in evaluating the model's accuracy.

- Diagonal values: Correct classifications. A high number on the diagonal (True Positives and True Negatives) means the model is classifying readmission well.
- Off-diagonal values: Incorrect classifications. A high number off-diagonal (False Positives or False Negatives) means the model is making more mistakes.

3.2 Feature Importance (Coefficients)

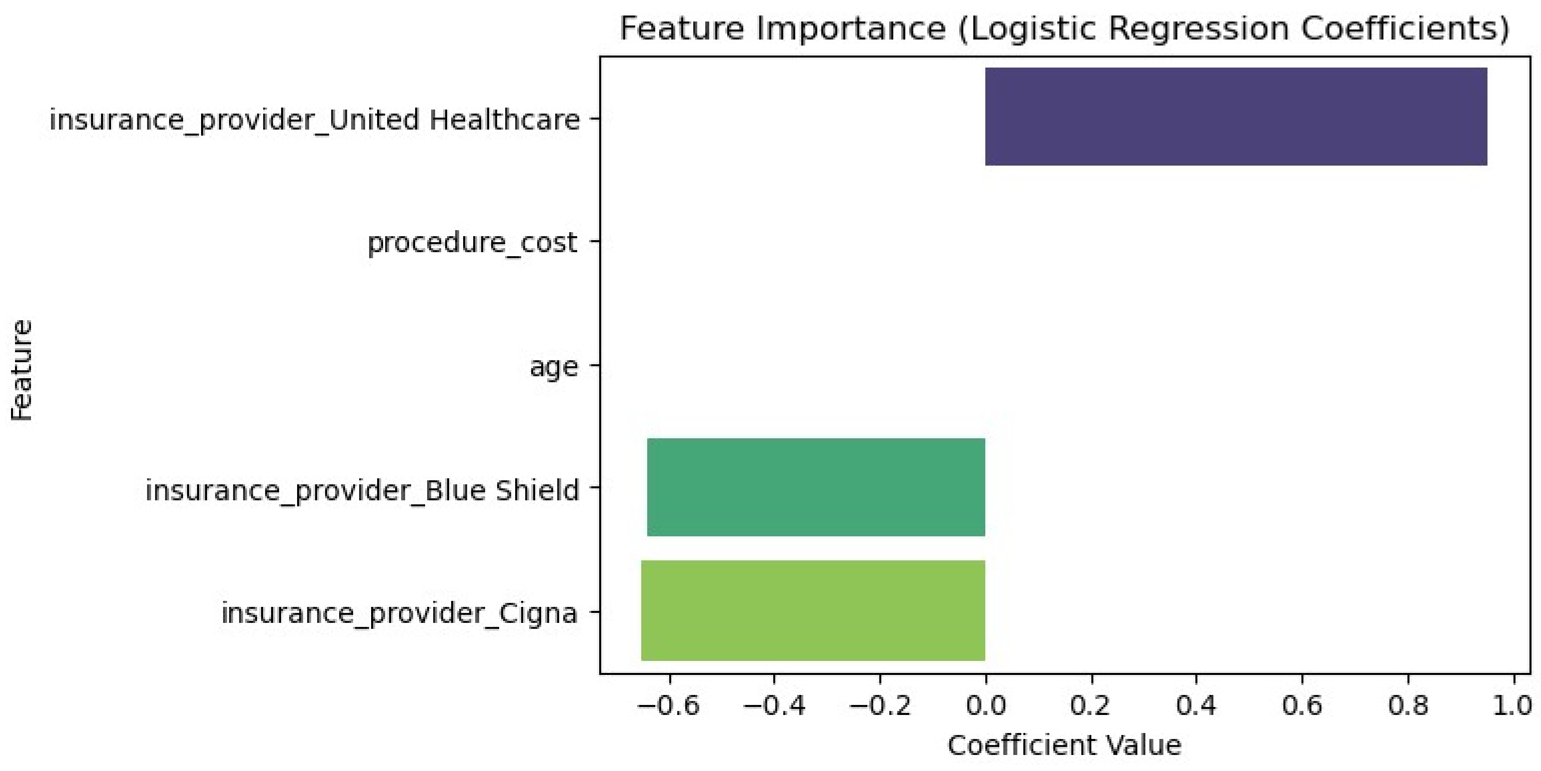
Feature Importance generates a bar plot showing the coefficients of the logistic regression model, which indicates the importance of each feature in predicting patient readmission.

Larger positive coefficients indicate a stronger predictive influence on a "Yes" readmission, while negative coefficients suggest an inverse relationship, reducing the likelihood of readmission.

Feature Importance plot to identify the most influential features for predictions.

```
coefficients = pd.DataFrame({'Feature': X_train.columns, 'Coefficient': model.coef_[0]})
coefficients = coefficients.sort_values(by='Coefficient', ascending=False)

plt.figure(figsize=(6, 4))
sns.barplot(x='Coefficient', y='Feature', data=coefficients, palette='viridis')
plt.title('Feature Importance (Logistic Regression Coefficients)')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.show()
```



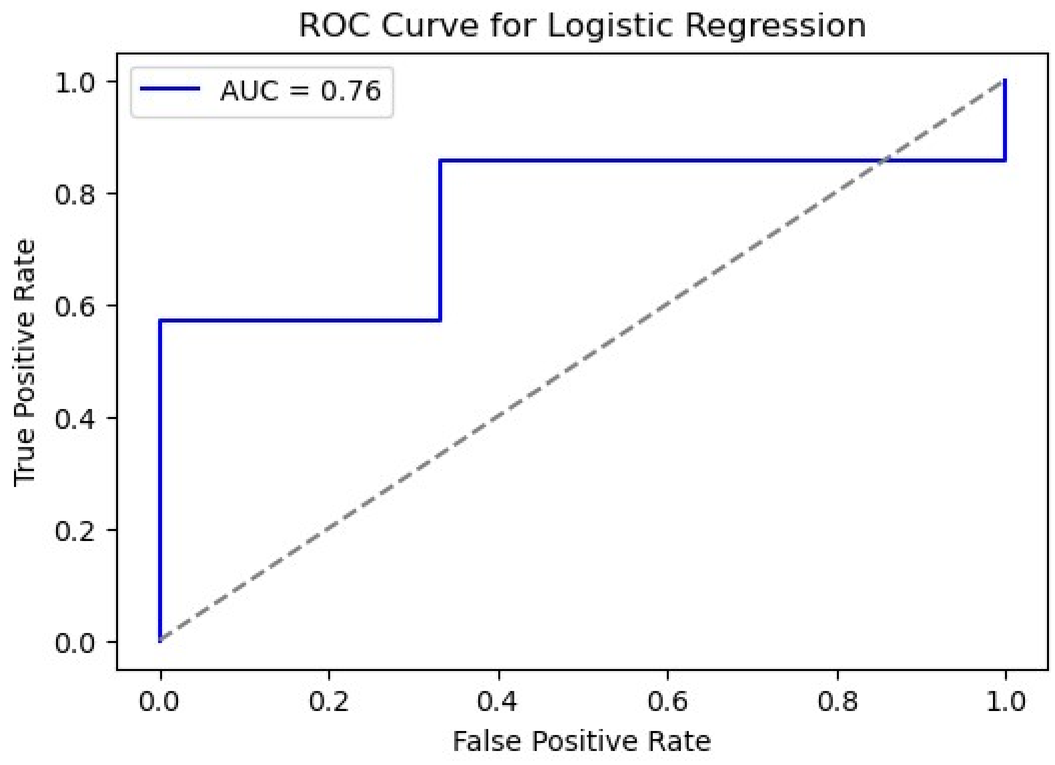
3.3 ROC Curve

The Receiver Operating Characteristic (ROC) curve helps visualize the trade-off between true positive rate (TPR) and false positive rate (FPR) across different thresholds.

```
# Get predicted probabilities for the 'Yes' class
y_prob = model.predict_proba(X_test)[:, 1]  # probability of 'Yes'

fpr, tpr, thresholds = roc_curve(y_test, y_prob)
auc = roc_auc_score(y_test, y_prob)
```

```
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.2f}')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line for random classifier
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression')
plt.legend()
plt.show()
```



- The ROC curve plots the True Positive Rate (recall) against the False Positive Rate. The closer the curve is to the top-left corner, the better the model.
- AUC (Area Under Curve): Measures the overall ability of the model to distinguish between classes. A value closer to 1 indicates a good model. A model with a high AUC is better at distinguishing between the classes (perfect model would have

an AUC of 1.0).

4. Predictive Decision Tree Model: Patient Readmission

To compare the Logistic Regression model with a Decision Tree model, we will create a similar pipeline using the DecisionTreeClassifier from sklearn. Decision Trees can capture non-linear relationships in the data and are often more interpretable than Logistic Regression.

```
# Modules
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split


# Prepare features and target
X_tree = healthcare_df[['age', 'insurance_provider', 'procedure_cost']]
y_tree = healthcare_df['readmission']  # Target: "Yes" or "No"

# Encode categorical variables
X_tree = pd.get_dummies(X_tree, drop_first=True)

# Split the dataset into training and testing sets
X_train_tree, X_test_tree, y_train_tree, y_test_tree = train_test_split(X_tree, y_tree, test_size=0.2, random_state=42)

# Build the Decision Tree model
tree_model = DecisionTreeClassifier(random_state=42)
tree_model.fit(X_train_tree, y_train_tree)

# Make predictions
y_pred_tree = tree_model.predict(X_test_tree)

# Evaluate the Decision Tree model
print("Decision Tree Classification Report:")
print(classification_report(y_test_tree, y_pred_tree))
```

```
# Compare accuracy
tree_accuracy = accuracy_score(y_test_tree, y_pred_tree)
print(f"Decision Tree Accuracy: {tree_accuracy:.2f}")
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

          No       0.60      1.00      0.75         3
         Yes       1.00      0.71      0.83         7

    accuracy                           0.80        10
   macro avg       0.80      0.86      0.79        10
weighted avg       0.88      0.80      0.81        10

Decision Tree Accuracy: 0.80
```

4.1 Decision Tree Results

- Model Training: DecisionTreeClassifier builds a tree structure where data is split at each node based on the best feature split to maximize prediction accuracy.
- Hyperparameters: Hyperparameters can be tuned like max_depth, min_samples_split, and min_samples_leaf to control tree complexity and prevent overfitting.
- Evaluation: The classification_report provides precision, recall, F1-score, and support for both classes ("Yes" and "No").
- Accuracy: The overall accuracy score is printed to compare with Logistic Regression.

After running the Decision Tree model, we compare it with the Logistic Regression model on the following criteria:

Precision and Recall:

- Does the Decision Tree improve recall for the "Yes" class (readmissions)? The results indicate that the Decision Tree model improves recall for the "Yes" class (readmissions) compared to the Logistic Regression model.

> - Logistic Regression Recall: 0.57. The Logistic Regression model correctly identifies 57% of the actual "Yes" cases (readmissions). However, 43% of the actual "Yes" cases are missed, resulting in false negatives.
> - Decision Tree Recall: 0.71. The Decision Tree model improves upon this by correctly identifying 71% of the actual "Yes" cases. It reduces the percentage of missed cases (false negatives) to 29%.

Recall for the "Yes" class (readmissions) is particularly important if the goal is to ensure that most readmitted patients are identified. Missing fewer readmissions (lower false negatives) could be critical in healthcare scenarios where timely intervention is needed. The improvement in recall suggests that the Decision Tree model is better at detecting readmissions, likely because it can handle non-linear relationships and interactions between features that Logistic Regression may miss.

- How does the precision for "No" compare? The Decision Tree model improves precision for the "No" class (non-readmissions) compared to the Logistic Regression model.

> - Logistic Regression: Precision for "No" = 0.50. Out of all the predictions labeled as "No" (non-readmissions), only 50% were actually correct. A precision of 0.50 indicates a significant number of false positives — cases predicted as "No" but that were actually "Yes" (readmissions). This suggests the model struggles to confidently predict non-readmissions and mislabels many actual readmissions as "No."
> - Decision Tree: Precision for "No" = 0.60. The Decision Tree model improves upon this, with 60% of the predictions labeled as "No" being correct. Fewer false positives compared to Logistic Regression, suggesting the Decision Tree is slightly better at distinguishing between "No" and "Yes" classes.

The Decision Tree model's higher precision for "No" means it is better at minimizing false positives (predicting "No" when it should be "Yes"). While this is an improvement, precision for "No" is still relatively low in both models, indicating that neither model is particularly strong in confidently identifying non-readmissions without error.

F1-Score:

- Is the Decision Tree better at balancing precision and recall? Higher F1-scores indicate a better balance between precision and recall.

> - Logistic regression: Class "No": F1-Score = 0.67. Class "Yes": F1-Score = 0.73.
> - Decision Tree: Class "No": F1-Score = 0.75. Class "Yes": F1-Score = 0.83.

The Decision Tree outperforms Logistic Regression for both classes ('No' and 'Yes') in terms of F1-score. This suggests that the Decision Tree achieves a better trade-off between precision and recall across both classes. However, it's worth noting that the Decision Tree may also be more prone to overfitting, depending on the data and parameters used. Further evaluation, such as using cross-validation or analyzing precision and recall individually, might provide deeper insights.

Accuracy:

- Check if the overall accuracy improves with the Decision Tree.

> - Logistic Regression accuracy: 0.70. This indicates that the model correctly predicts 70% of the total instances.
> - Decision Tree accuracy: 0.80. This indicates an improvement, with 80% of the total predictions being correct.

The Decision Tree achieves a 10 percentage point improvement in accuracy over Logistic Regression, indicating stronger overall performance in making correct predictions for this dataset. However, it is important to consider the potential risk of overfitting, as Decision Trees, particularly when not pruned or regularized, can overly adapt to the training data. This may result in poor generalization to new data. To ensure reliable performance, it is advisable to validate the model using cross-validation or by testing it on a separate dataset.

5. Model Visualization: Decision Tree

To visualize the Decision Tree model and its predictions, we can create plots to better understand how the model performs and how it relates to the input features.

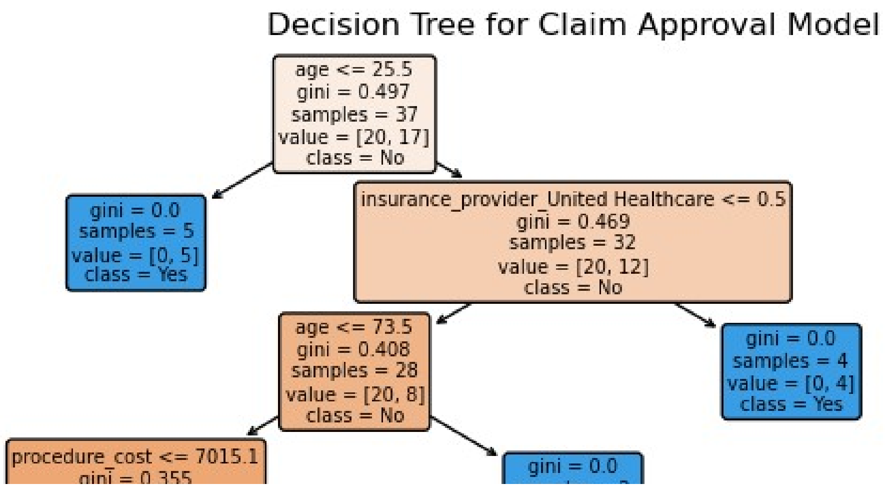
```
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, accuracy_score
import seaborn as sns
```

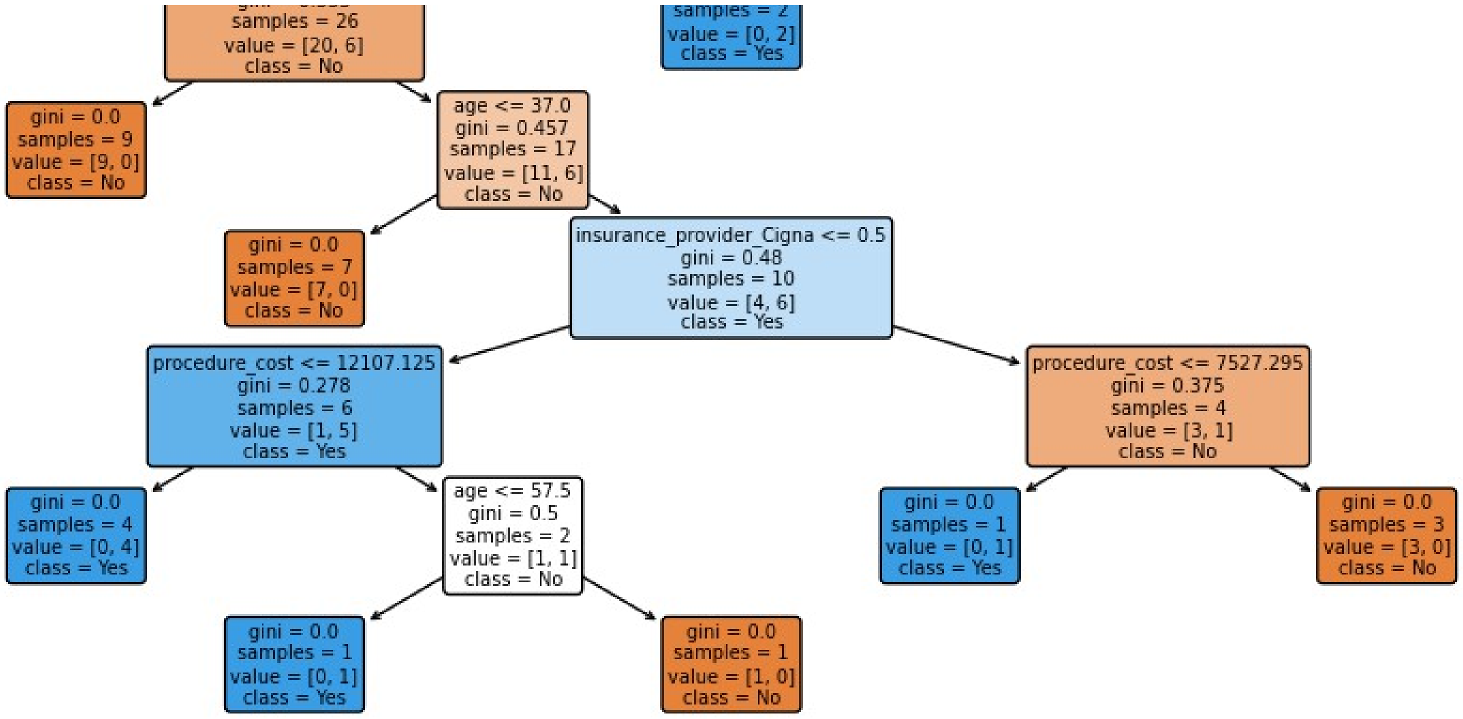
5.1 Decision Trees Plot

The Decision Tree plot provides a visual representation of the tree's structure, making it easier to understand how the model makes predictions and splits the data at each node.

Each node in the plot displays key information, including the feature used for the split, the threshold value, and metrics such as Gini impurity or entropy. The plot also shows the outcomes at each leaf. By tracing through the tree, you can follow the logic behind the model's predictions step by step.

```
plt.figure(figsize=(12, 8))
plot_tree(tree_model, filled=True, feature_names=X_train_tree.columns, class_names=['No', 'Yes'], rounded=True)
# plot_tree(tree_model, feature_names=X.columns, class_names=tree_model.classes_, filled=True)
plt.title('Decision Tree for Claim Approval Model')
plt.show()
```





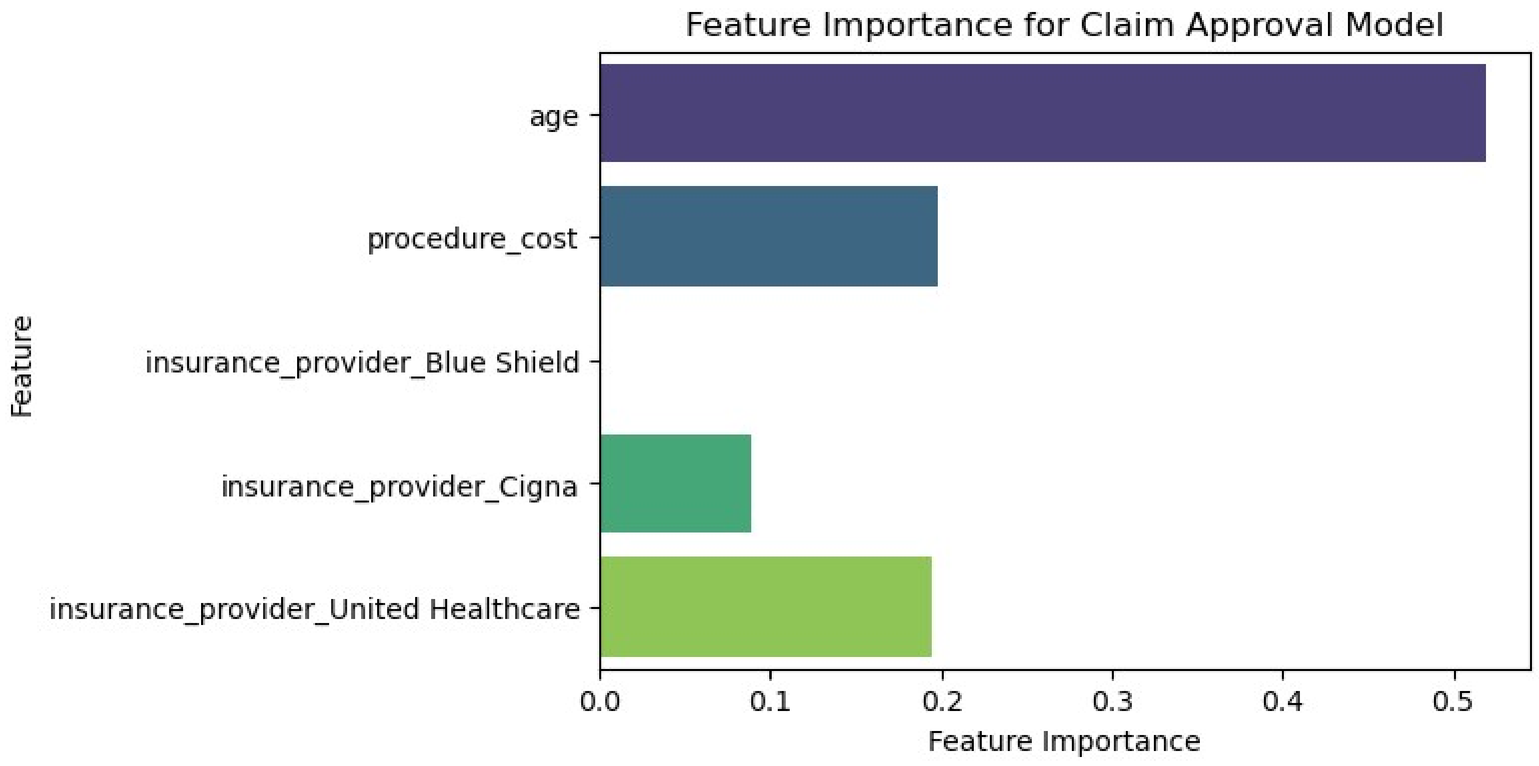
5.2 Feature Importance

Feature importance highlights the relative contribution of each feature to the model's predictions, providing insight into which features the Decision Tree relies on the most.

A bar chart is commonly used to visualize feature importance, where larger bars represent features that have a greater impact on predicting the target variable. This helps identify the most influential factors in the model's decision-making process.

```
feature_importance = tree_model.feature_importances_
feature_names = X_train_tree.columns
```

```
plt.figure(figsize=(6, 4))
sns.barplot(x=feature_importance, y=feature_names, palette='viridis')
plt.title('Feature Importance for Claim Approval Model')
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.show()
```



5.3 Confusion Matrix

The Confusion Matrix provides a detailed breakdown of the model's performance by comparing the actual values with the predicted values. It offers valuable insights into how well the model classifies patient readmissions ("Yes") versus non-readmissions ("No").

A heatmap is often used to visualize the Confusion Matrix, making it easier to interpret the results. Ideally, most values should appear along the diagonal (representing True Positives and True Negatives), indicating correct classifications, while off-diagonal values (False Positives and False Negatives) should be minimized.

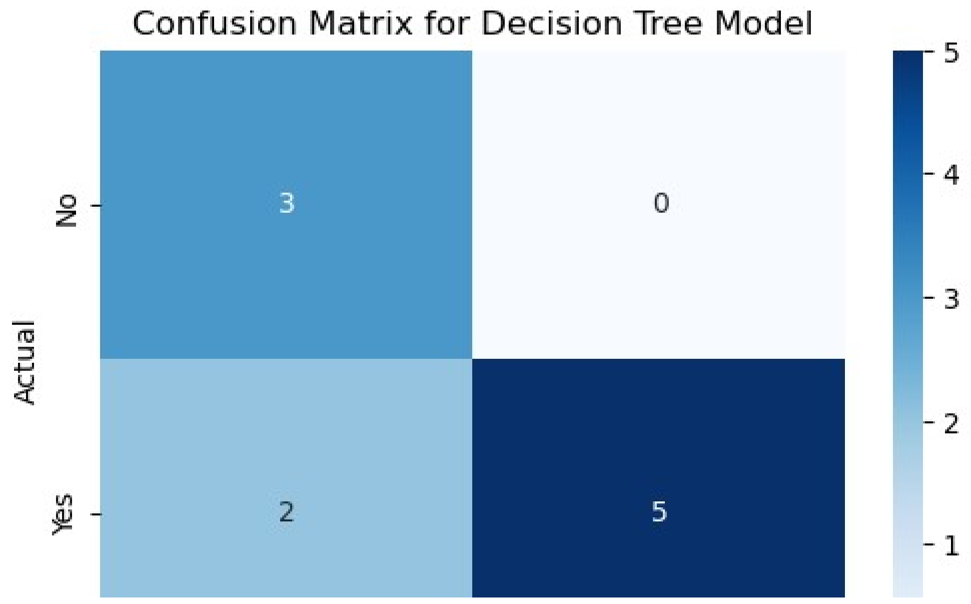
```
y_pred_tree = tree_model.predict(X_test_tree)
conf_matrix = confusion_matrix(y_test_tree, y_pred_tree)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Decision Tree Model')

# Calculate and print accuracy
accuracy = accuracy_score(y_test_tree, y_pred_tree)
print('Accuracy:', accuracy)

plt.show()
```

```
    Accuracy: 0.8
```



5.4 ROC Curve

The ROC Curve is used to evaluate how well the Decision Tree model distinguishes between the two classes (Yes and No). It plots the True Positive Rate (Recall) against the False Positive Rate, illustrating the model's performance across different classification thresholds.

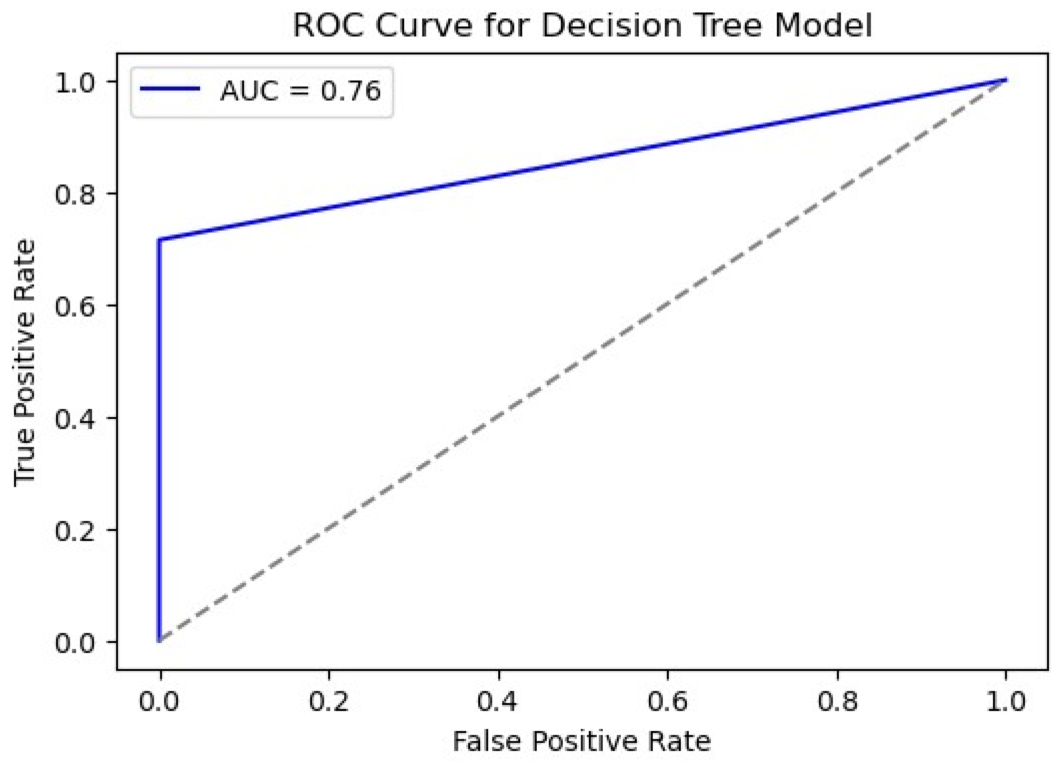
A curve that is closer to the top-left corner indicates better performance, as it shows the model is more effective at distinguishing between the classes. The AUC (Area Under the Curve) score further quantifies this ability, with a value closer to 1 indicating stronger discrimination between the classes.

```
# Convert 'Yes'/'No' to 1/0
y_test_tree_numeric = (y_test_tree == 'Yes').astype(int)

# Get predicted probabilities for the 'Yes' class
y_prob_tree = tree_model.predict_proba(X_test_tree)[:, 1]  # Probability of 'Yes' (1)

fpr_tree, tpr_tree, thresholds_tree = roc_curve(y_test_tree_numeric, y_prob_tree)
auc_tree = roc_auc_score(y_test_tree_numeric, y_prob_tree)

plt.figure(figsize=(6, 4))
plt.plot(fpr_tree, tpr_tree, color='blue', label=f'AUC = {auc:.2f}')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line for random classifier
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Decision Tree Model')
plt.legend()
plt.show()
```



- The ROC curve plots the True Positive Rate (recall) against the False Positive Rate. The closer the curve is to the top-left corner, the better the model.
- AUC (Area Under Curve): Measures the overall ability of the model to distinguish between classes. A value closer to 1 indicates a good model. A model with a high AUC is better at distinguishing between the classes (perfect model would have an AUC of 1.0).