PART I: RESEARCH QUESTION

A1. PROPOSAL OF QUESTION

What factors contribute significantly to the variability in additional charges incurred by the patients using the random forest method based on the available data?

A2. DEFINE GOAL

The research aims to uncover the variables that have a significant impact on the variability in additional charges by analyzing various factors such as demographics, medical history, and medical conditions. Identifying factors contributing to additional charges can help healthcare providers develop strategies to optimize healthcare costs effectively.

PART II: METHOD JUSTIFICATION

B1. EXPLANATION OF PREDICTION METHOD

Random Forest regression is an “ensemble learning method” that operates by constructing multiple decision trees during training and mean regression of the individual trees (Shafi, 2023). Each decision tree works independently and does not talk to the others. They all look at different parts of the problem and come up with their own solution. Once every decision tree has made its decision, they are gathered to see what the majority of them think. For regression, the average is taken of all their decisions.

Expected outcomes of Random Forest are accurate predictions and feature importance. Random Forest typically produces accurate predictions because it combines the predictions of multiple decision trees. Random Forest also provides a measure of feature importance. This indicates the contribution of each feature to the model’s prediction. The result is further insight into the underlying patterns in the data and identifying important features.

B2. SUMMARY OF METHOD ASSUMPTION

Random Forest regression assumes that there is some meaningful relationship between the feature variables and the target variable that can be captured by constructing decision trees. This assumption implies that Random Forest performs best when there is sufficient variation in the input features that allows the algorithm to make accurate predictions. Therefore, it is important to carefully select and preprocess the input features to ensure they contain relevant information for predicting the target variable.

B3. PACKAGES OR LIBRARIES LIST

Packages & libraries	**Usage**
csv	For reading and writing CSV files
pandas	For data manipulation, provides data structure like DataFrame
warnings	For controlling warning messages by suppressing certain warning messages that may not be critical
platform	Provides information about the platform and/or version of the Python running

numpy	For numerical computing, also for mathematical functions that enables arrays to efficiently operate
matplotlib.pylot	A plotting library, used to create histograms, bar charts
seaborn	A statistical data visualization based on matplotlib, used to create heatmaps and correlation matrices
sklearn(scikit-learn)	A machine learning library that provides tools for data mining and data analysis
RandomForestRegressor()	Used to create a Random Forest regression model
SelectKBest()	Used for feature selection based on the k highest scores
GridSearchCV()	Used for hyperparameter tuning via grid search with cross-validation
train_test_split()	Used to split arrays or matrices into random train and test subsets
mean_squared_error()	Used to calculate the mean squared error between predicted and true value
r2_score()	Used to compute the R-squared score
mean_absolute_error()	Used to calculate the mean absolute error between predicted and true values

PART III: DATA PREPARATION

C1. DATA PREPROCESSING

Feature selection involves selecting the most relevant features that contribute significantly to predicting the target variable. This helps reduce the overfitting, reduce the dimensionality of the dataset, and improve model performance. Random Forest performs feature selection by itself during the training process by evaluating the importance of each feature based on how much they contribute to reducing impurity in the decision trees.

C2. DATA SET VARIABLES

After feature selection using SelectKBest

Numeric	Categorical
Children	HighBlood
vitD_supp	Diabetes
	Gender
	Complication_risk
	Stroke
	ReAdmis
	Initial_admin

C3. STEPS FOR ANALYSIS

The preprocessing steps for this analysis:

1. Initialize all libraries and packages to be used

```python
# Data Handling
import csv
import pandas as pd

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")

# System Information
import platform

# Numeric Computation and Array Handling
import numpy as np

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

# Machine Learning
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest, f_classif

# Statistical Analysis
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy import stats

# Tree Visualisation
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz
```

2. Print Python version

```python
# Print Python version
print("Python Version:", platform.python_version())
```

```
Python Version: 3.11.5
```

3. Load the dataset into Pandas dataframe

```python
# Specifiy CSV file path
file_path = r'C:\Users\kolgi\OneDrive - Western Governors University\d209\medical_clean.csv'

# Open the CSV file and read it using DictReader
with open(file_path, 'r') as csvfile:
    csvreader = csv.DictReader(csvfile)

# Read the CSV file into a pandas DataFrame then open head
df = pd.read_csv(file_path)
```

4. View the first 5 rows of the DataFrame
   Df.head()
5. View the index, column names, non-null count and data types
   Df.info()
6. Check for null/missing values in the dataset, then count the null values for each column
   Df.isnull().sum()
7. Check for duplicates in the data

Df.duplicated()

8. Drop columns irrelevant to the analysis

```
# Drop columns irrelevant to the analysis
columns_to_drop = ['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State',
                   'County', 'Zip', 'Lat', 'Lng', 'Population', 'TimeZone', 'Job',
                  'Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item5', 'Item6', 'Item7', 'Item8']

# Drop the specified columns
df_new = df.drop(columns=columns_to_drop)

# Display the updated DataFrame
print(df_new.info)
```

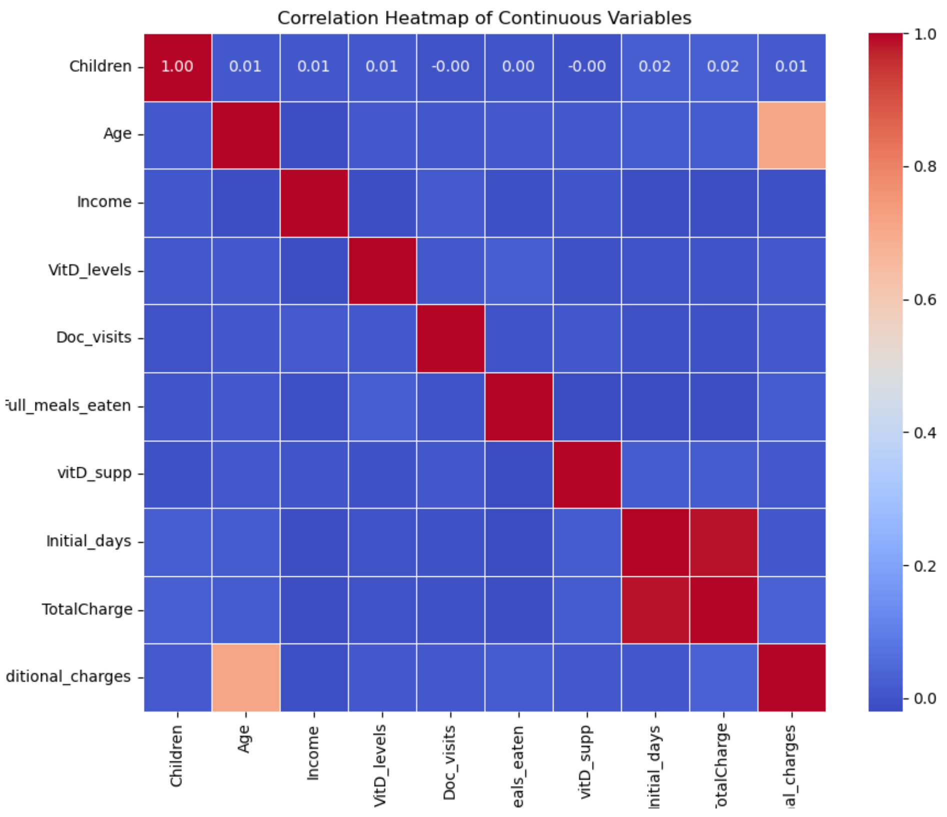
9. Calculate the correlation between all continuous variables. Create a correlation matrix heatmap for visualization

```
# Select the variables for the correlation heatmap
cont_var = ['Children', 'Age', 'Income', 'VitD_levels', 'Doc_visits', 'Full_meals_eaten',
            'vitD_supp', 'Initial_days', 'TotalCharge', 'Additional_charges']

# Subset the dataframe with the selected variables
df_subset = df[cont_var]

# Calculate the correlation matrix
correlation_matrix = df_subset.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap of Continuous Variables')
plt.savefig("corr_con.png")
plt.show()
```



10. Convert categorical variables to numerical using LabelEncoder()

```
# List of categorical columns to be encoded
categorical_columns = [
    'Area', 'Marital', 'Gender', 'ReAdmis',
    'Soft_drink', 'Initial_admin', 'HighBlood', 'Stroke', 'Complication_risk',
    'Overweight', 'Arthritis', 'Diabetes', 'Hyperlipidemia', 'BackPain',
    'Anxiety', 'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Services'
]

# Initialize LabelEncoder
encoder = LabelEncoder()

# Encode categorical columns
for column in categorical_columns:
    df_new[column] = encoder.fit_transform(df[column])

# Display the updated DataFrame
pd.set_option('display.max_columns', None)
print(df_new)
```

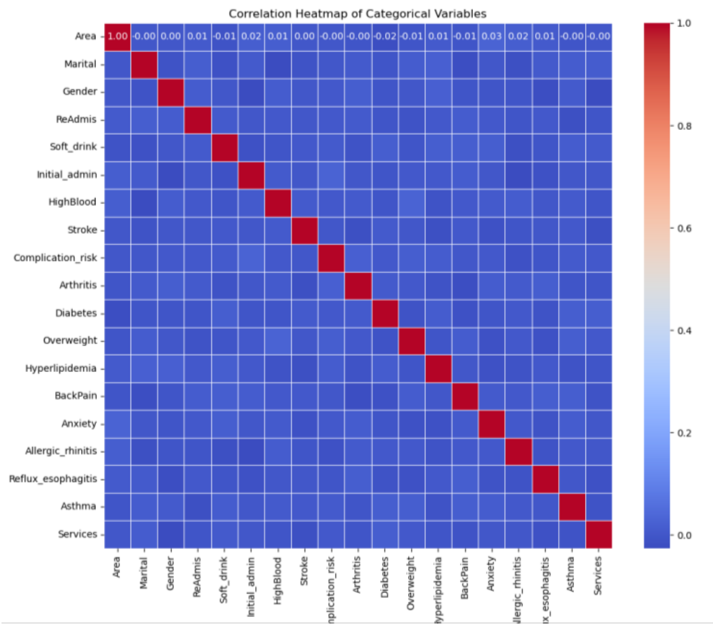
11. Calculate the correlation between all categorical variables. Create a correlation matrix heatmap for visualization

```
# Select the variables for the correlation heatmap
cat = ['Area', 'Marital', 'Gender', 'ReAdmis', 'Soft_drink', 'Initial_admin',
             'HighBlood', 'Stroke', 'Complication_risk', 'Arthritis', 'Diabetes',
             'Overweight', 'Hyperlipidemia', 'BackPain', 'Anxiety',
             'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Services']

# Subset the dataframe with the selected variables
df_subcat = df_new[cat]

# Calculate the correlation matrix
correlation_matrix = df_subcat.corr()

# Plot the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap of Categorical Variables')
plt.savefig("corr_categorical.png")
plt.show()
```



12. Drop age, initial days and total charge because of the high correlation

```
# Drop age, initial days and total charge because of the high correlation
columns_to_drop = ['Age', 'TotalCharge', 'Initial_days']

# Drop the specified columns
df_new = df_new.drop(columns=columns_to_drop)

# Display the updated DataFrame
df_new.info()
```

13. Visualize categorical variables using a bar graph

```
# Visualization for categorical variables
bar_graphs = ['Area', 'Marital', 'Gender', 'ReAdmis', 'Soft_drink', 'Initial_admin',
             'HighBlood', 'Stroke', 'Complication_risk', 'Arthritis', 'Diabetes',
             'Overweight', 'Hyperlipidemia', 'BackPain', 'Anxiety',
             'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Services']

# Calculate the number of rows and columns for the subplots
num_rows = len(bar_graphs) // 3 + (1 if len(bar_graphs) % 3 != 0 else 0)
num_cols = min(3, len(bar_graphs))

fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(15, 5*num_rows))

for i, col in enumerate(bar_graphs):
    row_index = i // num_cols
    col_index = i % num_cols
    df[col].value_counts().plot(kind='bar', ax=axes[row_index, col_index])
    axes[row_index, col_index].set_title(col)

# Remove any unused subplots
for i in range(len(bar_graphs), num_rows * num_cols):
    fig.delaxes(axes.flatten()[i])

plt.tight_layout()
plt.show()
```

14. Display the update DataFrame

```
# Display the updated DataFrame
pd.set_option('display.max_columns', None)
print(df_new)
```

15. Save the preprocessed DataFrame into a .csv file

```
# Save the filtered DataFrame to a CSV file
df_new.to_csv('filtered_data.csv', index=False)
```

16. Define the features (X) and target (y) for feature selection and print their shape

```
# From Elleh (2023)

# Assign to X all the predictor features
X = df_new.drop(["Additional_charges"], axis=1)
print(X.shape)

# Assign to y to the target variable
y = df["Additional_charges"]
print(y.shape)
```

```
(10000, 25)
(10000,)
```

17. Initialize SelectKBest with f_classif scoring function and print shape of the transformed feature matrix.

```
# From Elleh (2023)

# Initialize SKBest
skbest = SelectKBest(score_func=f_classif, k='all')
X_new = skbest.fit_transform(X, y)
print(X_new.shape)
```

```
(10000, 25)
```

18. Calculate p-values for each feature. Then, select only significant features with p-values of <0.05. Finally, print the significant features and their corresponding values, and the p-values of all features.

```
# From Elleh (2023)

# Calculate P-values for X
p_values = pd.DataFrame({'Feature': X.columns, 'p-value': skbest.pvalues_}).sort_values('p-value')
significant_features = p_values[p_values['p-value'] < 0.05]

# Print Features to keep and all P-values
features_to_keep = significant_features['Feature']
print("Features to keep:")
print(features_to_keep)
print("\nP-values:")
print(p_values)
```

```
Features to keep:
12            HighBlood
1              Children
17             Diabetes
4                Gender
14    Complication_risk
13               Stroke
9              vitD_supp
5               ReAdmis
11        Initial_admin
Name: Feature, dtype: object

P-values:
               Feature       p-value
12           HighBlood  0.000000e+00
1             Children  0.000000e+00
17            Diabetes  0.000000e+00
4               Gender  0.000000e+00
14   Complication_risk  0.000000e+00
```

19. Check VIF for multicollinearity issues amongst these features

```
# Check VIF for multicollinearity issues amongst these features

# Create a new DataFrame with the selected features
X_new = X[features_to_keep]
X_new.info()

# Calculate the VIF for each feature
vif = pd.DataFrame()
vif["Feature"] = X_new.columns
vif["VIF"] = [variance_inflation_factor(X_new.values, i) for i in range(X_new.shape[1])]

# Print the VIFs
print(vif)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   HighBlood          10000 non-null  int32
 1   Children           10000 non-null  int64
 2   Diabetes           10000 non-null  int32
 3   Gender             10000 non-null  int32
 4   Complication_risk  10000 non-null  int32
 5   Stroke             10000 non-null  int32
 6   vitD_supp          10000 non-null  int64
 7   ReAdmis            10000 non-null  int32
 8   Initial_admin      10000 non-null  int32
dtypes: int32(7), int64(2)
memory usage: 429.8 KB
             Feature       VIF
0          HighBlood  1.568058
1           Children  1.739607
2           Diabetes  1.318243
3             Gender  1.717506
4  Complication_risk  2.150924
5             Stroke  1.215652
6          vitD_supp  1.331071
7            ReAdmis  1.487972
8      Initial_admin  2.305965
```

20. Save preprocessed dataset into a .csv file

```
# Save the filtered DataFrame to a CSV file
X_new.to_csv('filtered_med.csv', index=False)
```

C4. CLEANED DATASET

.csv attached as filtered_med.csv

PART IV: ANALYSIS

D1. SPLITTING THE DATA

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_new, y, train_size = 0.8, test_size=0.2, random_state=15)
```

Train_test_split() splits the input features (X_new) and the target variable (y) into training and testing set. 80% of the data will be used for training while the remaining 20% is used for testing. The parameter random_state sets the random seed for reproducibility to a specific value of 15. The data splitting will be deterministic which ensures the same split is obtained each time the code is executed.

```
# Save the training and testing sets as csv files
pd.DataFrame(X_train).to_csv('X_train.csv')
pd.DataFrame(X_test).to_csv('X_test.csv')
pd.DataFrame(y_train).to_csv('y_train.csv')
pd.DataFrame(y_test).to_csv('y_test.csv')
```

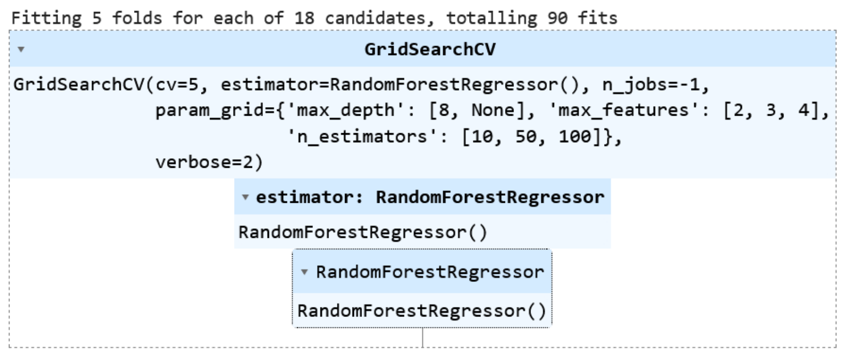
Export the training and testing files to .csv.

X_train.csv, X_test.csv, y_train.csv, and y_test.csv files attached

D2. OUTPUT & INTERMEDIATE CALCULATIONS

1.
```
[12329.42809274 10300.85893078 17365.04773973 ...  6788.19810494
 15079.23534136  8541.37062343]
```
First, initialize RandomForestRegressor(). Then, train the Random Forest model on the training data using rf_model.fit(). These two steps enable the model to be ready to make predictions on

new data. This leads to using rf_model.predict() to predict the target variable for the testing set using the trained Random Forest model. The variable y_pred (and the result shown above) contains the predicted values for the target variable corresponding to the testing set.



```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
▾                            GridSearchCV
GridSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'max_depth': [8, None], 'max_features': [2, 3, 4],
                         'n_estimators': [10, 50, 100]},
             verbose=2)
                   ▾ estimator: RandomForestRegressor
                   RandomForestRegressor()
                       ▾ RandomForestRegressor
                       RandomForestRegressor()
```

2.

Param_grid is used to define the grid of hyperparameters to search. N_estimators is the number of trees in the forest. Max_features is the number of features to consider when looking for the best split. Max_depth is the maximum depth of the tree.
Then, instantiate GridSearchCV() to perform an exhaustive search over the specified parameter values and use cross validation. Finally, use grid_search.fit() to fit the GridSearchCV object to the training data, performing the grid search and training the model on each parameter combination.

```
Mean Squared Error (MSE): 29529893.991297066
Root Mean Squared Error (RMSE): 5434.141513734903
R-squared Score: 0.3225176326399726
Accuracy: 0.3225176326399726
```

3.

- Mean Squared Error (MSE) measures the average squared difference between the estimated values and the actual value.
- Root Mean Squared Error (RMSE) is simply the square root of MSE. It gives an idea of how much error the system typically makes in its predictions, with a higher weight for large errors.
- R-squared Score measures the proportion of the variance in the dependent variable that is predictable from the independent variables.
- Accuracy for regression is the R-squared score, which is why both numbers are the same.

```
Best Parameters: {'max_depth': 8, 'max_features': 3, 'n_estimators': 100}
Best Score: 0.42240533903249877
```

4.

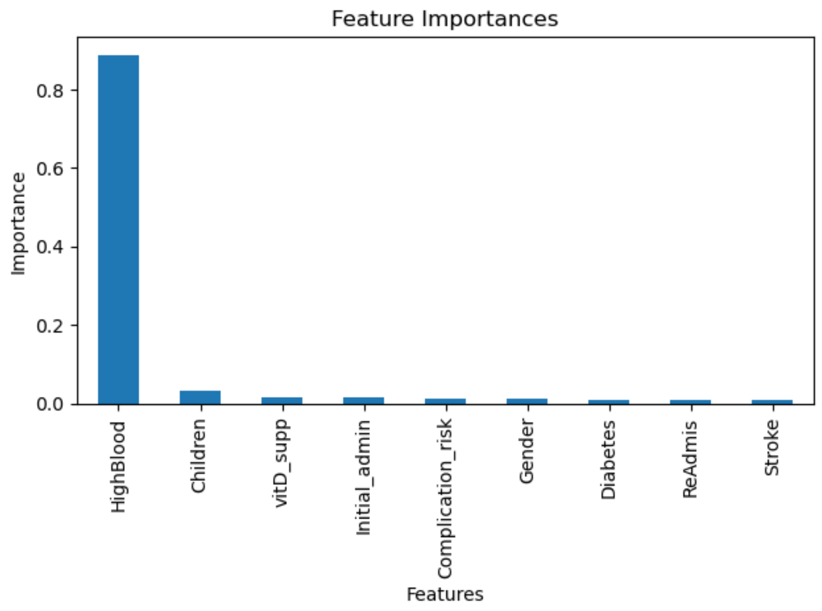
Calculate and print the best parameters during the GridSearchCV process. These parameters represent the combinations that yielded the highest score. Then, retrieve the best score achieved during the grid search. This score represents the performance of the model using the best parameters. Lastly, retrieve the best model found during the grid search. This model is trained on the entire training dataset using the best parameters.

```
Top 5 Most Predictive Features:
1. Feature 'HighBlood' - Importance: 0.8874654826955288
2. Feature 'Children' - Importance: 0.03322190617939644
3. Feature 'vitD_supp' - Importance: 0.015560686837301291
4. Feature 'Initial_admin' - Importance: 0.014411434425013579
5. Feature 'Complication_risk' - Importance: 0.013088438597880366
```

5.

Firstly, retrieve the feature importances assigned to each feature by the trained Random Forest model. Then, sort the indices of the features based on their importances in descending order.

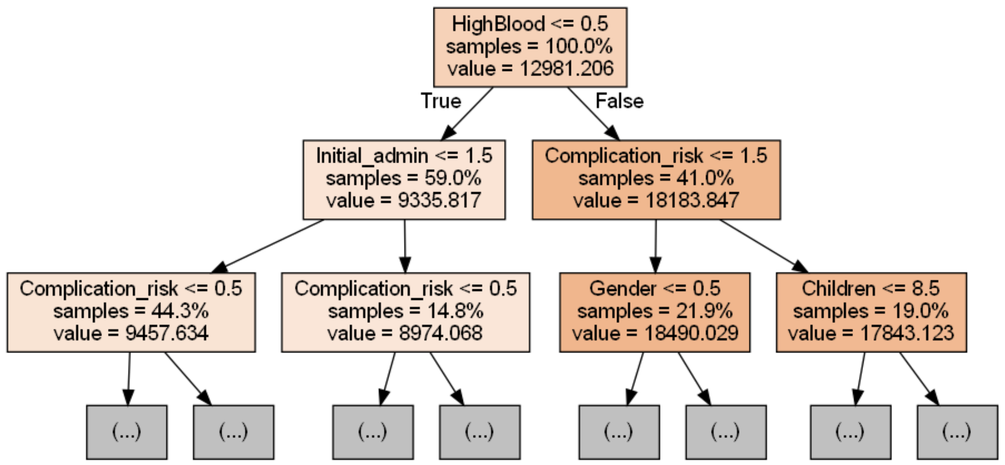
This helps identify the features that have the most influence on the predictions made by the Random Forest model. By, analyzing these top predictive features, you can gain insights into which variables are most relevant for predicting the variability in additional charges incurred by patients.



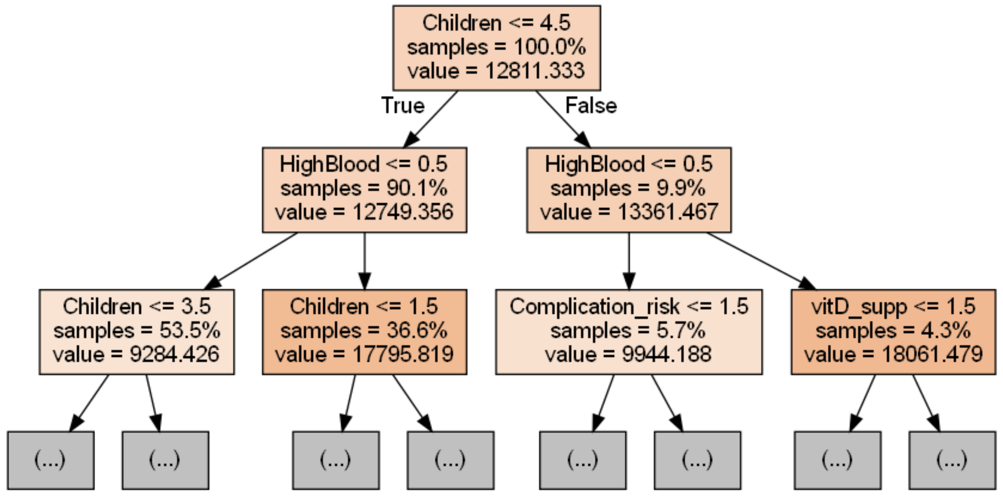
6. Create a bar chart to visualize the feature importance obtained from the best Random Forest model. First, create a Pandas series containing feature importances as values and feature names as index. Then, use the Pandas plot to generate a bar chart where each bar represents the importance of feature. Finally, add labels to the x and y axis. This bar chart provides a visual representation of feature importance which it makes it easier to interpret and identify the most influential features in the model.

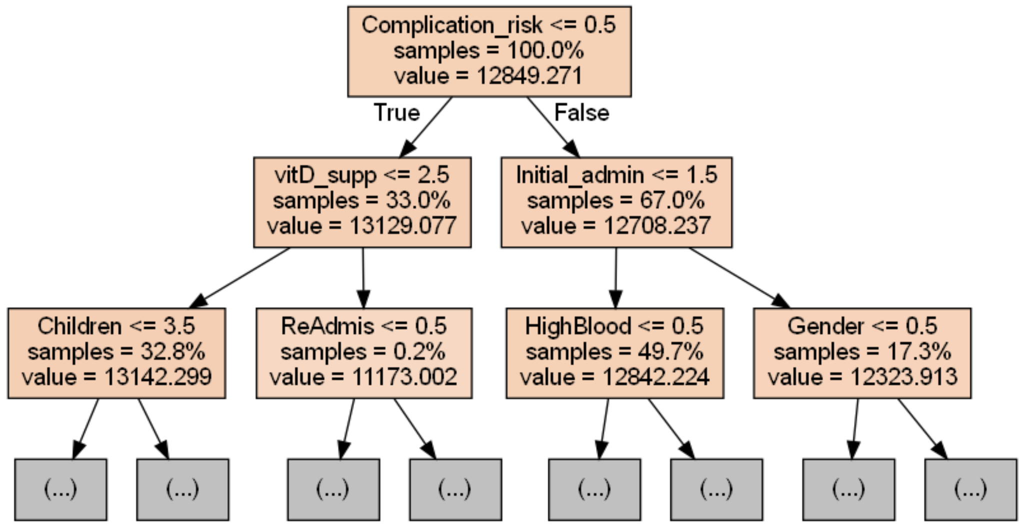
```
Evaluation Metrics for Best Model
Mean Squared Error: 24852901.37467099
Root Mean Squared Error: 4985.268435568038
R-squared Score: 0.4298183913548882
Accuracy Score 0.4298183913548882
```

7. Evaluate the performance of the Random Forest model using the best parameters obtained from the hyperparameter tuning process. First, initialize a new Random Forest model with the best parameters obtained from the hyperparameter tuning process. Also, set the random_state to 42 as to ensure reproducibility. Then, train the Random Forest model on the training data (X_train, y_train). After that, make predictions on the test data using the trained model using predict(). Finally, re-calculate and print the evaluation metrics.



8.





The top box is the variable name and value for splitting. The middle boxes are the percentage of the total samples in each split. The bottom boxes are the percentage of the split between classes in each split.

Visualizing random forests with decision trees provides a transparent and intuitive way to explore the inner workings of the model and gain insights into feature importance.

D3. CODE EXECUTION

1. Initialize and train the Random Forest model. Then, use the trained model to make predictions on the testing data.

```
# Initialize the Random Forest model
rf_model = RandomForestRegressor()

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = rf_model.predict(X_test)
print(y_pred)
```

```
[12441.54621744 10049.82739317 15563.7645073  ...  6803.42450075
 15678.66325848  8372.71451857]
```

2. Perform hyperparameter tuning using GridSearchCV to identify the optimal values for the parameters of the Random Forest model

```
# Identify optional values for parameters using hyperparameter tuning
param_grid = {
    'n_estimators': [10, 50, 100],
    'max_features': [2, 3, 4],
    'max_depth': [8, None]
}

# Instantiate GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

# Perform GridSearchCV and training the modelgrid_search.fit(X_train, y_train)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

```
GridSearchCV
GridSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'max_depth': [8, None], 'max_features': [2, 3, 4],
                         'n_estimators': [10, 50, 100]},
             verbose=2)
estimator: RandomForestRegressor
RandomForestRegressor()
RandomForestRegressor
RandomForestRegressor()
```

3. Calculate evaluation metrics of the Random Forest model

```python
# Find the best parameters
best_rf_model = RandomForestRegressor(**best_params, random_state=42)
best_rf_model.fit(X_train, y_train)
y_pred_best = best_rf_model.predict(X_test)
accuracy = best_rf_model.score(X_test, y_test)

# Re-calculate evaluation metrics
mae_best = mean_absolute_error(y_test, y_pred_best)
mse_best = mean_squared_error(y_test, y_pred_best)
rmse_best = mean_squared_error(y_test, y_pred_best, squared=False)
r2_best = r2_score(y_test, y_pred_best)

# Print evaluation metrics
print("Evaluation Metrics for Best Model:")
print("Mean Squared Error:", mse_best)
print("Root Mean Squared Error:", rmse_best)
print("R-squared Score:", r2_best)
print("Accuracy Score", accuracy)
```

```
Evaluation Metrics for Best Model:
Mean Squared Error: 24852901.37467099
Root Mean Squared Error: 4985.268435568038
R-squared Score: 0.4298183913548882
Accuracy Score 0.4298183913548882
```

4. Retrieve and print the best parameters and the corresponding score obtained from the GridSearchCV process, as well as the best model

```python
# Check and print the best params
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Check the best score for the top performing model
best_score = grid_search.best_score_
print("Best Score:", best_score)

# Get the best model from GridSearchCV
best_rf_model = grid_search.best_estimator_
```

```
Best Parameters: {'max_depth': 8, 'max_features': 3, 'n_estimators': 100}
Best Score: 0.4209787455373829
```

5. Determine the top predictive features according to the best Random Forest model

```python
# Determine which features were the most predictive according to the Random Forest Regressor
feature_importances = best_rf_model.feature_importances_
sorted_indices = feature_importances.argsort()[::-1]

print("Top 5 Most Predictive Features:")
for i in range(5):
    feature_index = sorted_indices[i]
    feature_name = X_train.columns[feature_index]
    importance_score = feature_importances[feature_index]
    print(f"{i+1}. Feature '{feature_name}' - Importance: {importance_score}")
```

```
Top 5 Most Predictive Features:
1. Feature 'HighBlood' - Importance: 0.8874654826955288
2. Feature 'Children' - Importance: 0.03322190617939644
3. Feature 'vitD_supp' - Importance: 0.015560686837301291
4. Feature 'Initial_admin' - Importance: 0.014411434425013579
5. Feature 'Complication_risk' - Importance: 0.013088438597880366
```

6. Create a bar chart to visualize the feature importances

```
# From Shafi(2023)

# Create a series containing feature importances from the model and feature names from the training data
feature_importances = pd.Series(best_rf_model.feature_importances_, index=X_train.columns).sort_values(ascending=False)

# Plot a simple bar chart
feature_importances.plot.bar()

# Set plot title and labels
plt.title("Feature Importances")
plt.xlabel("Features")
plt.ylabel("Importance")
plt.tight_layout()
plt.savefig("feature_importances.png")
plt.show()
```

7. Evaluate the performance of the Random Forest model using the best parameters obtained from the hyperparameter tuning process

```
# Find the best parameters
best_rf_model = RandomForestRegressor(**best_params, random_state=42)
best_rf_model.fit(X_train, y_train)
y_pred_best = best_rf_model.predict(X_test)
accuracy = best_rf_model.score(X_test, y_test)

# Re-calculate evaluation metrics
mae_best = mean_absolute_error(y_test, y_pred_best)
mse_best = mean_squared_error(y_test, y_pred_best)
rmse_best = mean_squared_error(y_test, y_pred_best, squared=False)
r2_best = r2_score(y_test, y_pred_best)

# Print evaluation metrics
print("Evaluation Metrics for Best Model:")
print("Mean Squared Error:", mse_best)
print("Root Mean Squared Error:", rmse_best)
print("R-squared Score:", r2_best)
print("Accuracy Score", accuracy)
```

```
Evaluation Metrics for Best Model:
Mean Squared Error: 24852901.37467099
Root Mean Squared Error: 4985.268435568038
R-squared Score: 0.4298183913548882
Accuracy Score 0.4298183913548882
```

8. Export the first 3 decision trees from the best Random Forest model

```
# From Shafi(2023)

# Define the directory to save the graphs
save_directory = r"C:\Users\kolgi\OneDrive - Western Governors University\d209\tree"

# Export the first three decision trees from the forest
for i in range(3):
    tree = best_rf_model.estimators_[i]
    dot_data = export_graphviz(tree,
                               feature_names=X_new.columns,
                               filled=True,
                               max_depth=2,
                               impurity=False,
                               proportion=True)
    graph = graphviz.Source(dot_data)
    # Save the graph as an image file in the specified directory
    graph.render(filename=f"{save_directory}\\decision_tree_{i}", format='png')
    # Display the graph
    display(graph)
```

PART V: DATA SUMMARY & IMPLICATIONS

E1. ACCURACY AND MSE

Metric	Score
Mean Squared Error (MSE)	`24852901.37`
Root Mean Squared Error (RMSE)	`4985.27`
R-Squared ($\boldsymbol{R^2}$)	`42.98%`

Mean Squared Error (MSE) is a measure of the average squared difference between the actual values and the predicted values. In my analysis, the MSE is `24852901.37` which means that on average, the squared difference between the actual and predicted additional charges incurred by patients is approximately `24852901.37.`

Root Mean Squared Error (RMSE) is the square root of the MSE. It provides a measure of the average magnitude of the errors in the predicted values. In my analysis, the RMSE is `4985.27` which means that on average, the difference between the actual and predicted additional charges incurred by patients is approximately `4985.27`.

R-Squared ($\boldsymbol{R^2}$) is a measure of how well the independent variables explain the variability of the dependent variable. It indicates the proportion of the variance in the dependent variable that is predictable from the independent variable. In my analysis, this means that an $\boldsymbol{R^2}$ of approximately 42.98% of the variability in additional charges incurred by patients is explained by the independent variables in the model. The low $\boldsymbol{R^2}$ indicates that this model is not an ideal fit for the data.

E2. RESULTS & IMPLICATIONS

Metric	Score before tuning	Best Model Score
Mean Squared Error (MSE)	`29443198.68`	`24852901.37`
Root Mean Squared Error (RMSE)	`5426.16`	`4985.27`
R-Squared ($\boldsymbol{R^2}$)	`32.05%`	`42.98%`

Above are the scores before and after hyperparameter tuning. Firstly, the MSE decreased by 4590297.31 on the best model. A lower MSE indicates that the best model's predictions are close to the actual values on average. This implies an improvement in prediction accuracy. The RMSE decreased by 440.89 on the best model. A lower RMSE indicates that the best model's predictions have less variability around the actual values. This implies an improved prediction precision. The R^2 increased by 10.5% on the best model. A higher score is ideal since it indicates that the best model has a large proportion of variability in the target variable that is explained by the feature variables.

The best parameters obtained through hyperparameter tuning are max_depth=8, max_features=4, and n_estimators=100. This indicates the optimal configuration for the Random Forest model.

The top predictive features are HighBlood, Children, vitD_supp, Initial_admin, and Gender. HighbBlood dominates the predictive power of the model with an importance of 0.8919. This suggests that patients