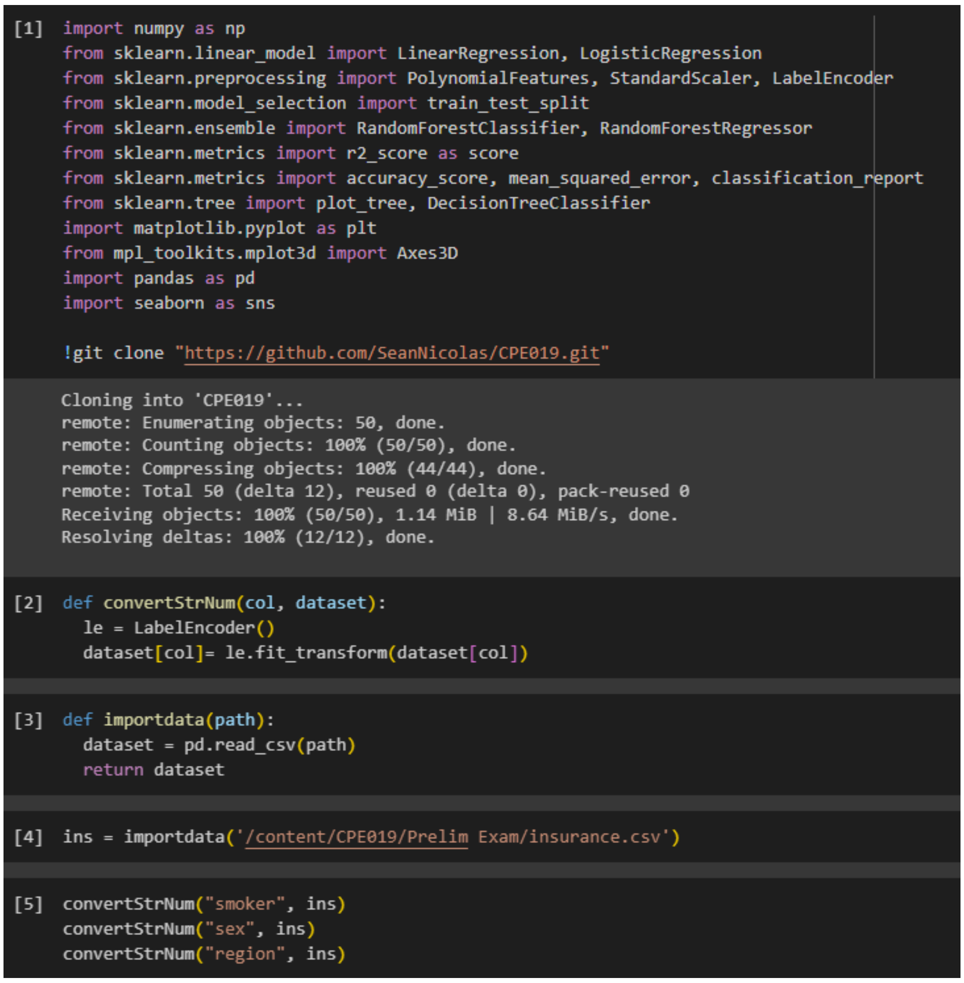
Prelim Examination	
Classification and Prediction Analysis Report	
Course Code: CPE019	**Program:** BSCpE
Course Title: Emerging Technologies 2	**Date Performed:** Feb 28, 2024
Section: CpE32S3	**Date Submitted:** Mar 6, 2024
Name: Maglalang, Charles Lester Nicolas, Sean Julian	**Instructor:** Engr. Roman Richard

Import Libraries and Creating Functions

In this stage we imported all of the libraries in a single cell block so that we can import all of the needed libraries in one execution. Creating functions including the lines of code that we often use makes it convenient and fast. Two functions included in the picture below are convertStrNum() that convert words like “yes” and “no” into numbers, the other function is importdata() which is used to easily import the dataset. In the code below, we also cloned the github path in order to easily access the dataset file.

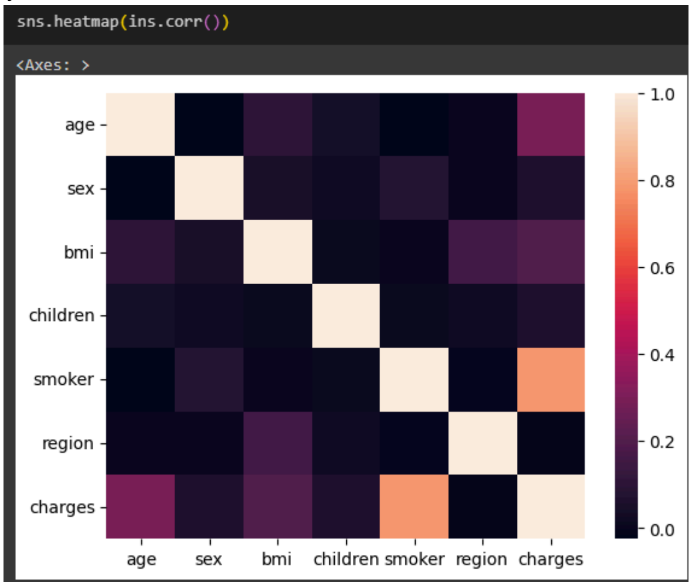


```
[1] import numpy as np
    from sklearn.linear_model import LinearRegression, LogisticRegression
    from sklearn.preprocessing import PolynomialFeatures, StandardScaler, LabelEncoder
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
    from sklearn.metrics import r2_score as score
    from sklearn.metrics import accuracy_score, mean_squared_error, classification_report
    from sklearn.tree import plot_tree, DecisionTreeClassifier
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import Axes3D
    import pandas as pd
    import seaborn as sns

    !git clone "https://github.com/SeanNicolas/CPE019.git"

    Cloning into 'CPE019'...
    remote: Enumerating objects: 50, done.
    remote: Counting objects: 100% (50/50), done.
    remote: Compressing objects: 100% (44/44), done.
    remote: Total 50 (delta 12), reused 0 (delta 0), pack-reused 0
    Receiving objects: 100% (50/50), 1.14 MiB | 8.64 MiB/s, done.
    Resolving deltas: 100% (12/12), done.

[2] def convertStrNum(col, dataset):
      le = LabelEncoder()
      dataset[col]= le.fit_transform(dataset[col])

[3] def importdata(path):
      dataset = pd.read_csv(path)
      return dataset

[4] ins = importdata('/content/CPE019/Prelim Exam/insurance.csv')

[5] convertStrNum("smoker", ins)
    convertStrNum("sex", ins)
    convertStrNum("region", ins)
```

Importing Libraries and Creating Function

Identifying Correlation

Using heatmap from the seaborn library, we can check the correlation between the different variables inside the database. This is helpful to identify how each variable affects one another and what variable are they most correlated with.



```
sns.heatmap(ins.corr())
```

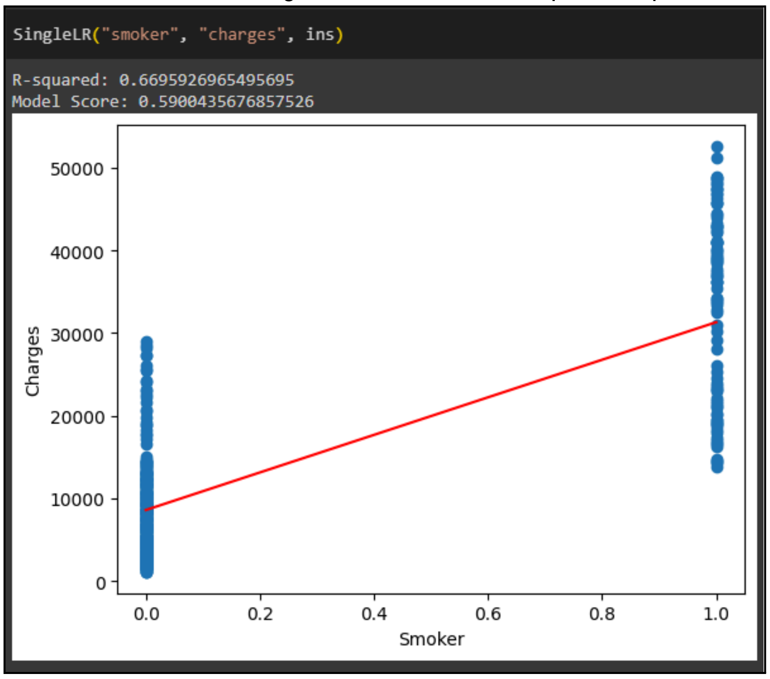
```
<Axes: >
```

Using Heatmap to Check Correlations

Singular Linear Regression

In Singular Linear Regression, the libraries that we used are Pandas and Numpy for data manipulation and reshaping arrays. Matplotlib for plotting and visualization, with sklearn.linear_model it is used for linear regression modeling. The dataset that we chose for this model is Insurance.csv, for X we used 'Smoker' and 'Charges' for our Y since they have a high relationship and fitted in this model.

It is shown here the visualization of the regression line in scatter plot and predicted values.



```
SingleLR("smoker", "charges", ins)
```

```
R-squared: 0.6695926965495695
Model Score: 0.5900435676857526
```

Linear Regression

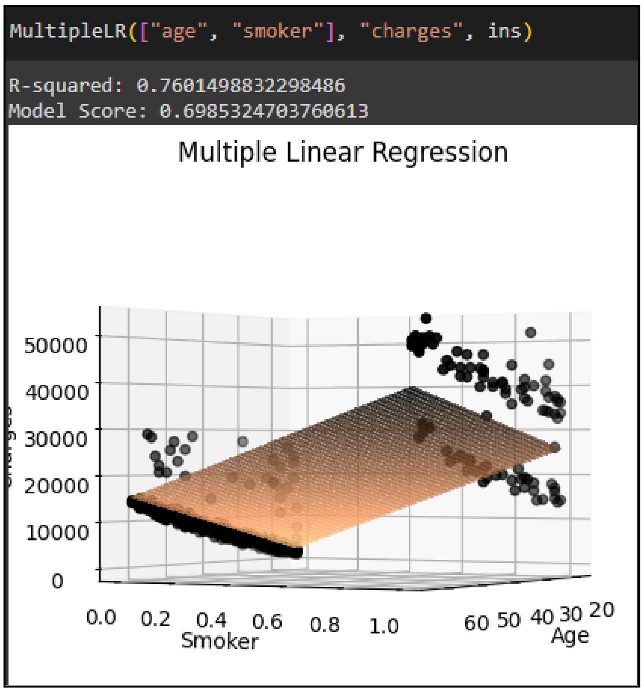
Looking at the figure above, we are able to see a linear regression between the smoker and charges variables. The connection between smoker and charges offered the highest accuracy after trying different combinations of variables

Multiple Linear Regression

Just like the Singular Linear Regression, we used the same libraries but we added Seaborn, it is the same with matplotlib but it has a high-level interface. We split the dataset into training and testing sets, incorporating multiple features into a data frame with Age, Smoker, Charges columns.

It shows here the visualization of Multiple Linear Regression using seaborn for 3d representation. Also, it shows here the actual and predicted values from the variables that we used, model score is included to show the accuracy of the model.

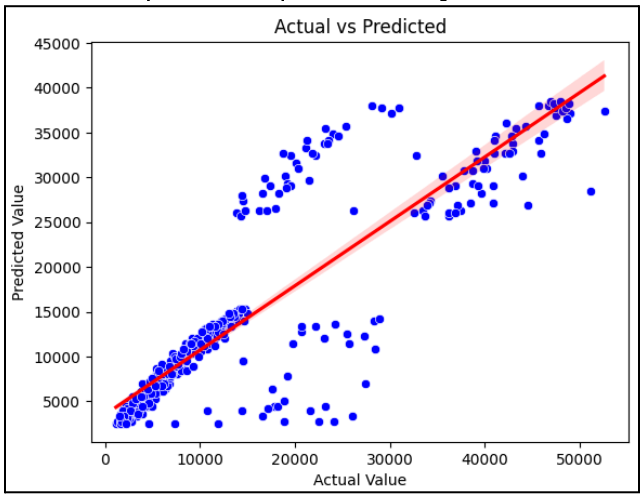
Upon checking the different combinations of variables, using age and smoker to predict charges had the highest accuracy



```
MultipleLR(["age", "smoker"], "charges", ins)
```

```
R-squared: 0.7601498832298486
Model Score: 0.6985324703760613
```

3D plot of Multiple Linear Regression

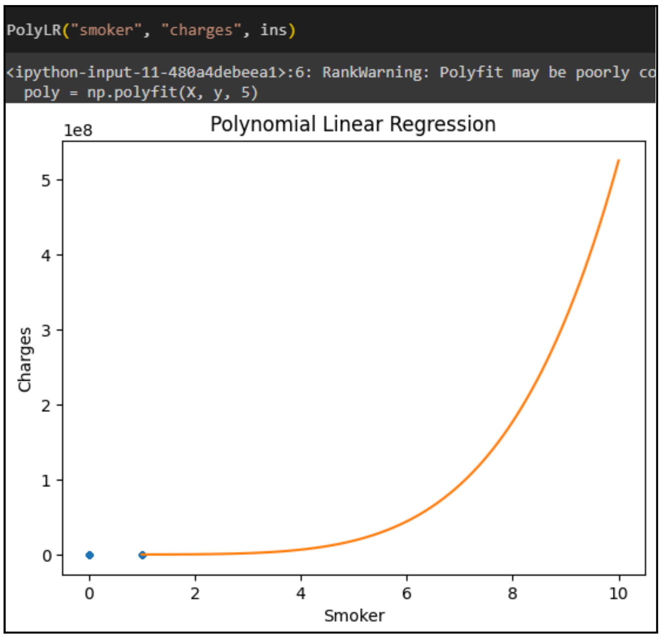


Prediction vs Actual

Polynomial Linear Regression

Compared to the other regression models, this regression needs to view the scatter plot to identify what polynomial feature will be used. In this model, we used the 'charges' and 'smoker' columns; these values represent the prediction of the polynomial curve for a given range of feature values.

It shows here the polynomial linear regression visualization, the dots represent the original data and the predicted values.

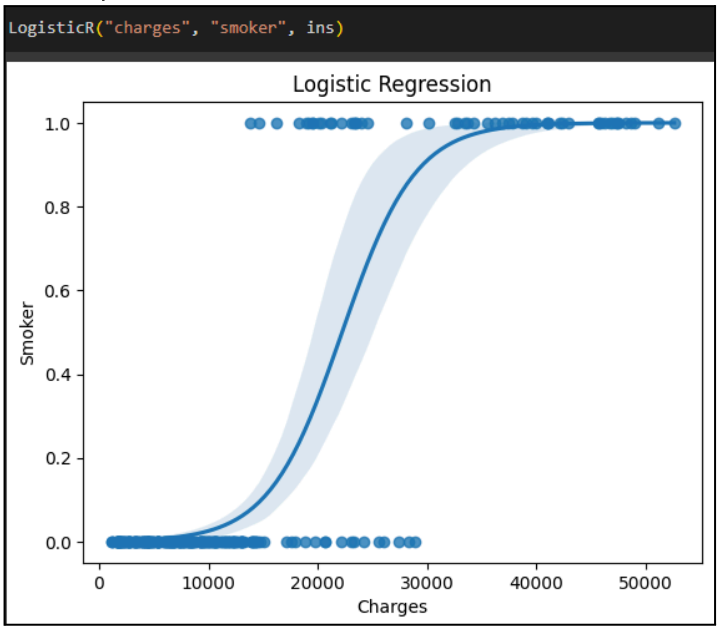


```
PolyLR("smoker", "charges", ins)
<ipython-input-11-480a4debeea1>:6: RankWarning: Polyfit may be poorly co
  poly = np.polyfit(X, y, 5)
```

Polynomial Linear Regression

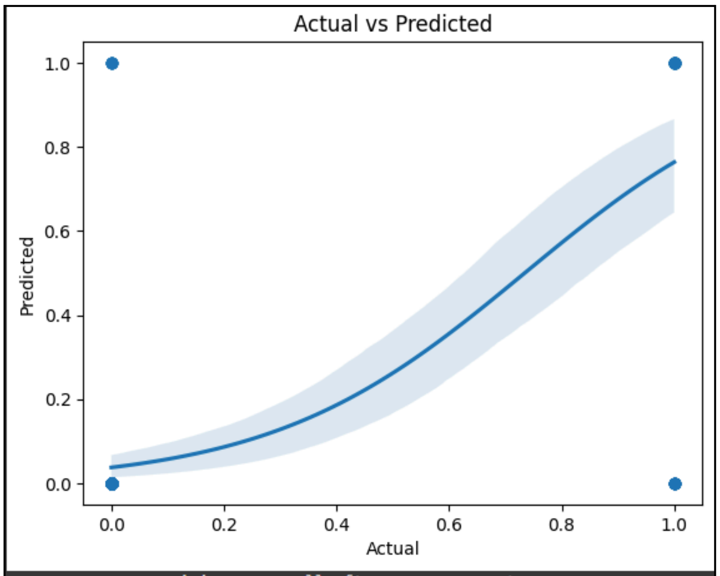
Logistic Regression Model

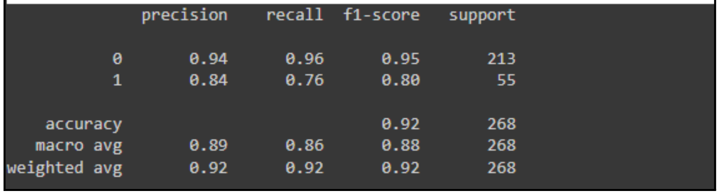
In a logistic regression model, libraries that need to be used are Pandas for data manipulation, Matplotlib for creating plots and visualization, and sklearn.linear_model.LogisticsRegression to build logistic regression models. Insurance.csv is the chosen data for this model since it fits in the regression line. 'charges' and 'smoker' columns will be the variables for our x and y.
It shows here the visualization of the logistic regression model, and the actual and predicted values are included, as well as the precision of the model.



```
LogisticR("charges", "smoker", ins)
```

Logistic Regression





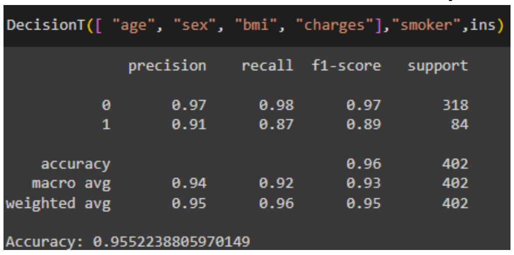
```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       213
           1       0.84      0.76      0.80        55

    accuracy                           0.92       268
   macro avg       0.89      0.86      0.88       268
weighted avg       0.92      0.92      0.92       268
```

Predicted vs Actual and Evaluation of Model

Decision Tree

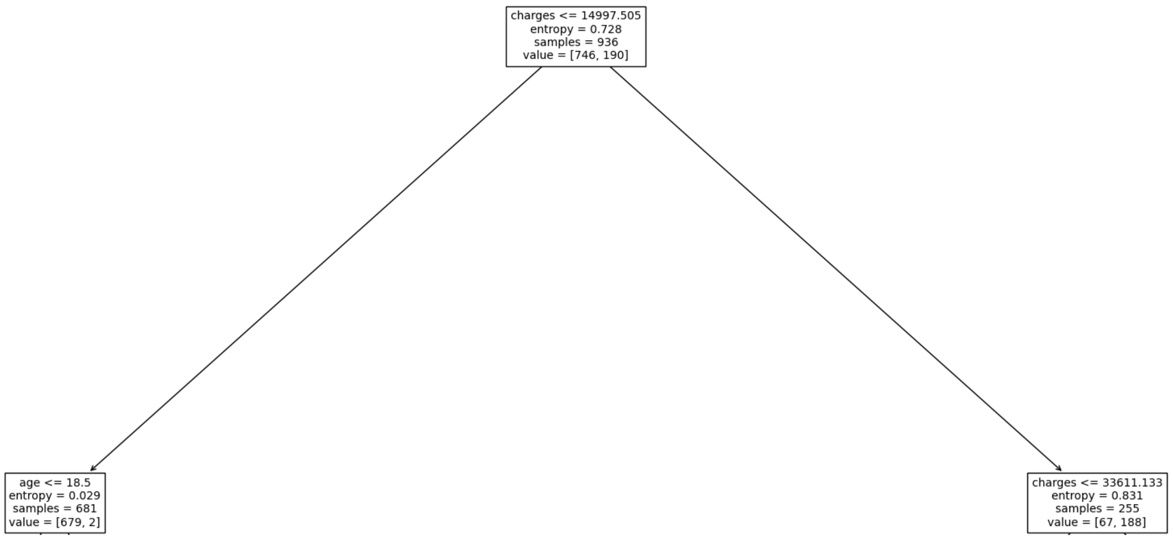
In the decision tree model, the same libraries are used, just like in the previous models. The data set was organized and separated into target values using train_test_split. A decision tree classifier was created using the entropy criterion to determine the best split at each node. The classifier trained on the training data and made predictions based on entropy.

It shows here the visualization of the decision tree and the accuracy of the model.

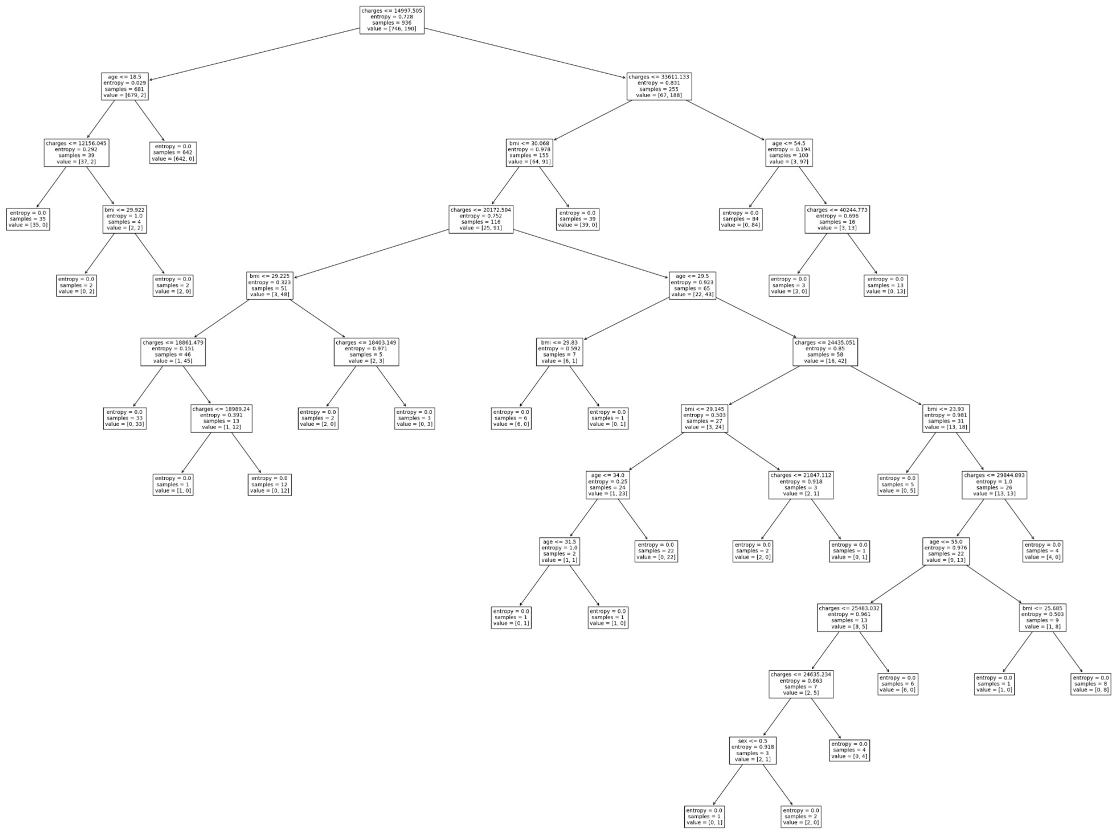


```
DecisionT([ "age", "sex", "bmi", "charges"],"smoker",ins)

              precision    recall  f1-score   support

           0       0.97      0.98      0.97       318
           1       0.91      0.87      0.89        84

    accuracy                           0.96       402
   macro avg       0.94      0.92      0.93       402
weighted avg       0.95      0.96      0.95       402

Accuracy: 0.9552238805970149
```

Evaluation of Model



Preview of Decision Tree

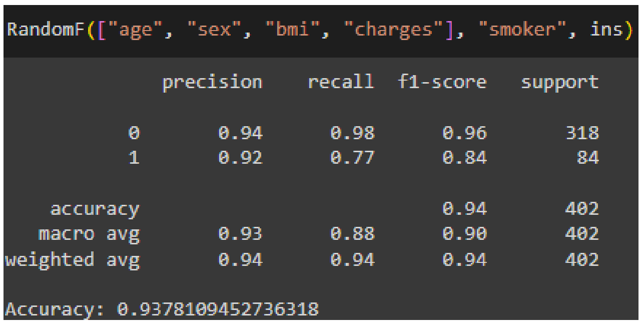


Full Decision Tree

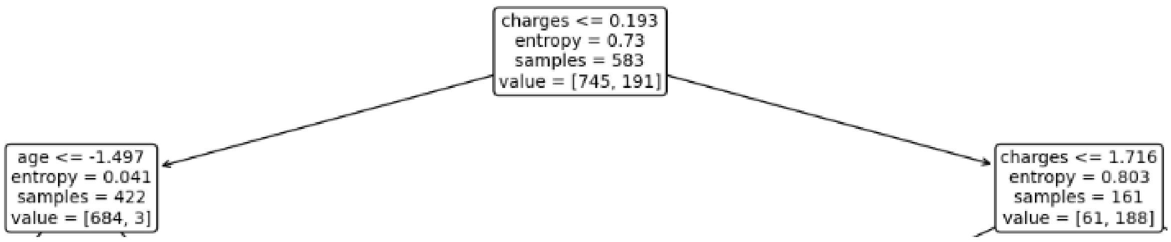
Random Forest

In creating a random forest, we found out that we can identify how many smokers there are in a region based on age, sex, BMI, and charge columns. The classifier used a default of 100 trees in the forest to predict test data. We created two variables, X and Y, where X represents the original data frame and Y represents the new data frame.

It shows here the visualization of random forests and the accuracy of the model. The predicted values for smokers in every region are also shown here.

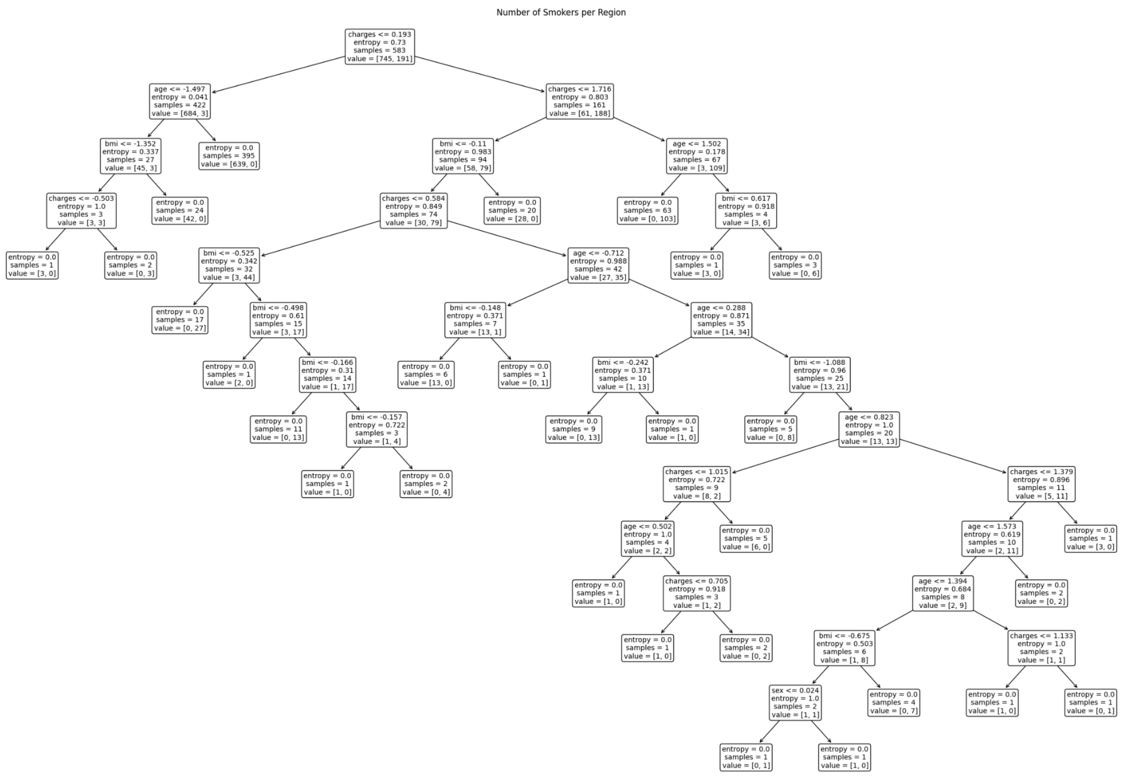


```
RandomF(["age", "sex", "bmi", "charges"], "smoker", ins)

              precision    recall  f1-score   support

           0       0.94      0.98      0.96       318
           1       0.92      0.77      0.84        84

    accuracy                           0.94       402
   macro avg       0.93      0.88      0.90       402
weighted avg       0.94      0.94      0.94       402

Accuracy: 0.9378109452736318
```



charges <= 0.193 entropy = 0.73 samples = 583 value = [745, 191]
age <= -1.497 entropy = 0.041 samples = 422 value = [684, 3]
charges <= 1.716 entropy = 0.803 samples = 161 value = [61, 188]

Preview



Number of Smokers per Region
charges <= 0.193 entropy = 0.73 samples = 583 value = [745, 191]
age <= -1.497 entropy = 0.041 samples = 422 value = [684, 3]
charges <= 1.716 entropy = 0.803 samples = 161 value = [61, 188]
bmi <= -1.352 entropy = 0.337 samples = 27 value = [45, 3]
entropy = 0.0 samples = 395 value = [639, 0]
bmi <= -0.11 entropy = 0.983 samples = 94 value = [58, 79]
age <= 1.502 entropy = 0.178 samples = 67 value = [3, 109]
charges <= -0.503 entropy = 1.0 samples = 3 value = [3, 3]
entropy = 0.0 samples = 24 value = [42, 0]
charges <= 0.584 entropy = 0.849 samples = 74 value = [30, 79]
entropy = 0.0 samples = 20 value = [28, 0]
entropy = 0.0 samples = 63 value = [0, 103]
bmi <= 0.617 entropy = 0.918 samples = 4 value = [3, 6]
entropy = 0.0 samples = 1 value = [3, 0]
entropy = 0.0 samples = 2 value = [0, 3]
bmi <= -0.525 entropy = 0.342 samples = 32 value = [3, 44]
age <= -0.712 entropy = 0.988 samples = 42 value = [27, 35]
entropy = 0.0 samples = 1 value = [3, 0]
entropy = 0.0 samples = 3 value = [0, 6]
entropy = 0.0 samples = 17 value = [0, 27]
bmi <= -0.498 entropy = 0.61 samples = 15 value = [3, 17]
bmi <= -0.148 entropy = 0.371 samples = 7 value = [13, 1]
age <= 0.288 entropy = 0.871 samples = 35 value = [14, 34]
entropy = 0.0 samples = 1 value = [2, 0]
bmi <= -0.166 entropy = 0.31 samples = 14 value = [1, 17]
entropy = 0.0 samples = 6 value = [13, 0]
entropy = 0.0 samples = 1 value = [0, 1]
bmi <= -0.242 entropy = 0.371 samples = 10 value = [1, 13]
bmi <= -1.088 entropy = 0.96 samples = 25 value = [13, 21]
entropy = 0.0 samples = 11 value = [0, 13]
bmi <= -0.157 entropy = 0.722 samples = 3 value = [1, 4]
entropy = 0.0 samples = 9 value = [0, 13]
entropy = 0.0 samples = 1 value = [1, 0]
entropy = 0.0 samples = 5 value = [0, 8]
age <= 0.823 entropy = 1.0 samples = 20 value = [13, 13]
entropy = 0.0 samples = 1 value = [1, 0]
entropy = 0.0 samples = 2 value = [0, 4]
charges <= 1.015 entropy = 0.722 samples = 9 value = [8, 2]
charges <= 1.379 entropy = 0.896 samples = 11 value = [5, 11]
age <= 0.502 entropy = 1.0 samples = 4 value = [2, 2]
entropy = 0.0 samples = 5 value = [6, 0]
age <= 1.573 entropy = 0.619 samples = 10 value = [2, 11]
entropy = 0.0 samples = 1 value = [3, 0]
entropy = 0.0 samples = 1 value = [1, 0]
charges <= 0.705 entropy = 0.918 samples = 3 value = [1, 2]
age <= 1.394 entropy = 0.684 samples = 8 value = [2, 9]
entropy = 0.0 samples = 2 value = [0, 2]
entropy = 0.0 samples = 1 value = [1, 0]
entropy = 0.0 samples = 2 value = [0, 2]
bmi <= -0.675 entropy = 0.503 samples = 6 value = [1, 8]
charges <= 1.133 entropy = 1.0 samples = 2 value = [1, 1]
sex <= 0.024 entropy = 1.0 samples = 2 value = [1, 1]
entropy = 0.0 samples = 4 value = [0, 7]
entropy = 0.0 samples = 1 value = [1, 0]
entropy = 0.0 samples = 1 value = [0, 1]
entropy = 0.0 samples = 1 value = [0, 1]
entropy = 0.0 samples = 1 value = [1, 0]

Full View