**Question 1:**

import pandas as pd

def changeLetterToGender(x):

if x == 'F':

x = 'female'

elif x == 'M':

x = 'male'

return x

#import file

raw = pd.read\_csv('ECA.csv', index\_col="PersonID")

#find number of rows with null values for age

raw[raw['age'].isnull()].count()

#drop all null values in the dataframe and change age from float to int

cleaned = raw.dropna()

cleaned = cleaned.astype({'age':'int32'})

#round charges to 2dp

cleaned['charges'] = cleaned['charges'].apply(lambda x:round(x,2))

#change all letters to actual sex name

cleaned['sex'] = cleaned['sex'].apply(lambda x:changeLetterToGender(x))

I must first import the Pandas library in order to use Python to read the data set. Next, a Pandas Dataframe called "df" will be created using the function "pd.read\_csv.". This dataframe will contain the data from the csv indicated in the code. The dataset I am using is the ECA.csv file.

Next, to handle missing values I used “raw[raw['age'].isnull()].count()” code to find the number of rows with null (missing) values in the 'age' column. Rows with missing values are dropped using “cleaned = raw.dropna()” because it may affect the quality of my analysis if left untouched.

Following that, I converted the 'age' column from a floating-point data type to an integer data type using cleaned = cleaned.astype({'age':'int32'}). Making the variable "age" an integer can improve the data's interpretability and suitability for specific analysis techniques. I also rounded the charges column to 2 decimal places using the function “cleaned['charges'] = cleaned['charges'].apply(lambda x:round(x,2))”. Doing so makes the dataset more readable and improves the presentation so that it is more appropriate for performing analysis.

Finally, I defined a custom function changeLetterToGender(x) to change gender labels from single letters ('M' and 'F') to their full names ('male' and 'female'). I applied this function to the sex column using “cleaned['sex'] = cleaned['sex'].apply(lambda x:changeLetterToGender(x))”. This makes the data more descriptive and human-readable, improving the clarity of the dataset.

All the above steps were part of my data preprocessing tasks to ensure that the dataset’s quality is enhanced. This also ensures that the data will be reliable and suitable when running analysis.

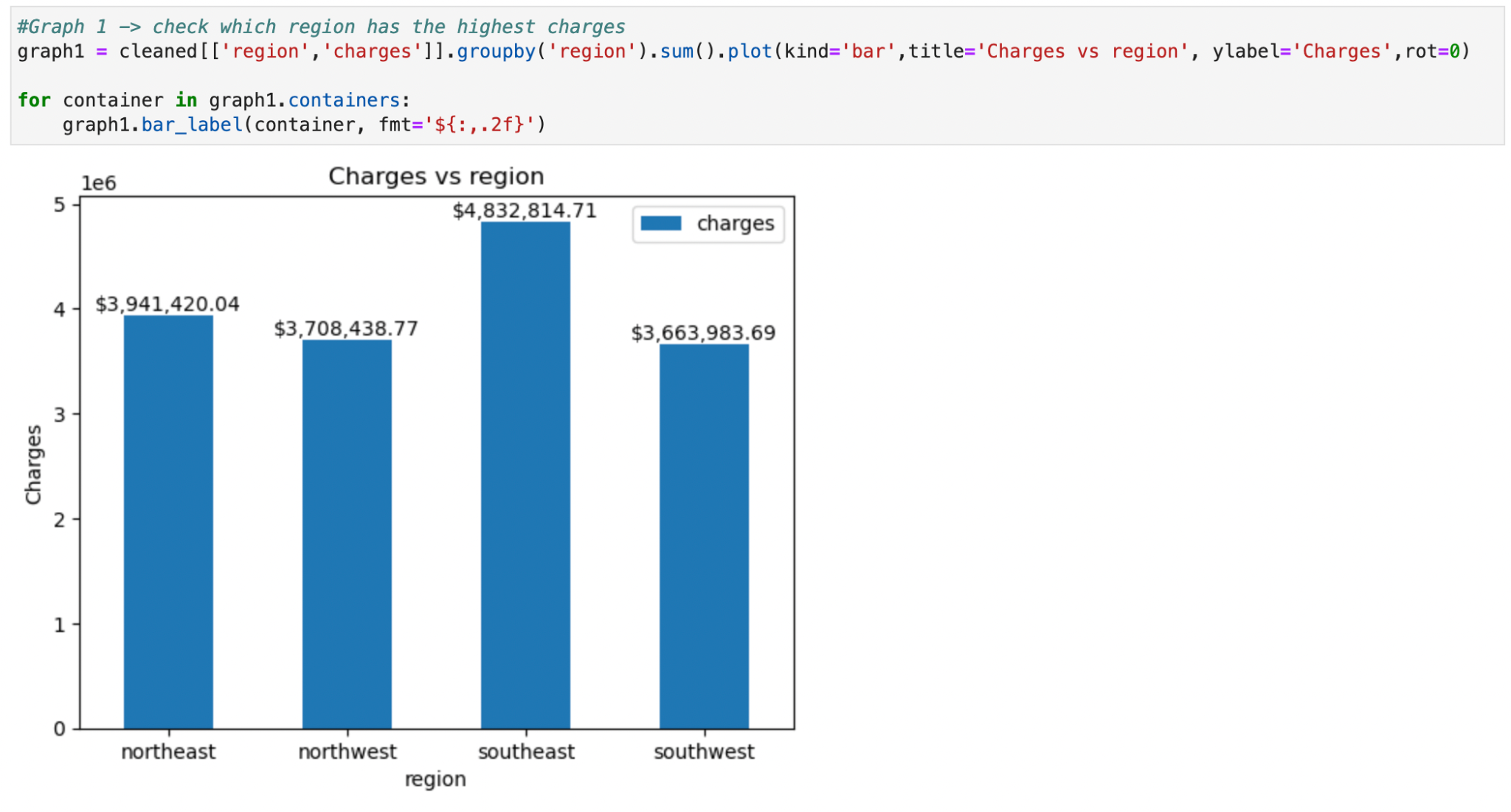
**Question 2:**

#Graph 1 -> check which region has the highest charges

graph1 = cleaned[['region','charges']].groupby('region').sum().plot(kind='bar',title='Charges vs region', ylabel='Charges',rot=0)

for container in graph1.containers:

graph1.bar\_label(container, fmt='${:,.2f}')



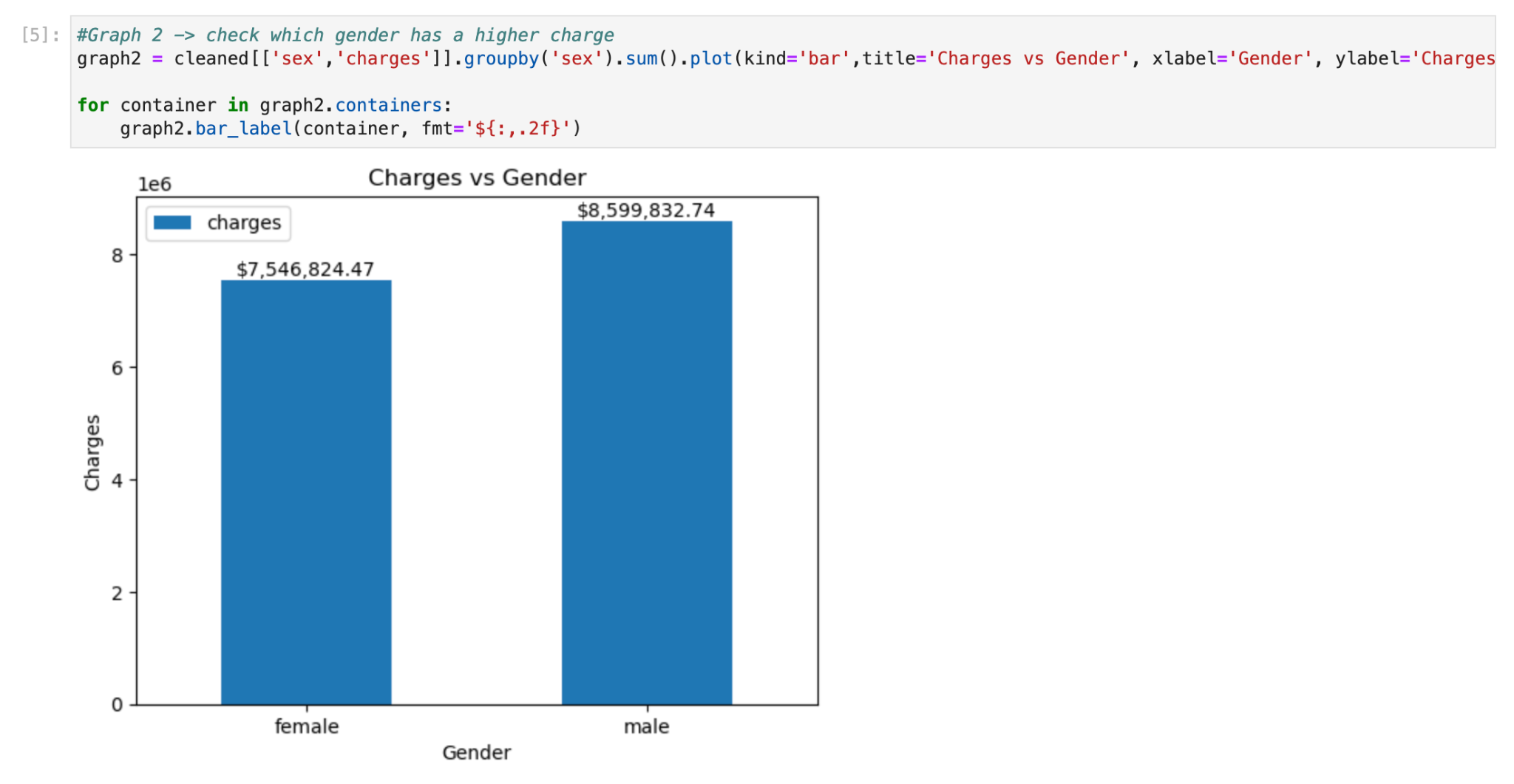
I wanted to find out which of the four regions provided in the dataset has the highest charges. This graph shows that the southeast region has the highest charges for medical costs for both males and females regardless of other factors.

#Graph 2 -> check which gender has a higher charge

graph2 = cleaned[['sex','charges']].groupby('sex').sum().plot(kind='bar',title='Charges vs Gender', xlabel='Gender', ylabel='Charges',rot=0)

for container in graph2.containers:

graph2.bar\_label(container, fmt='${:,.2f}')



This graph studies the difference in charges between females and males. As shown above, males have higher charges.

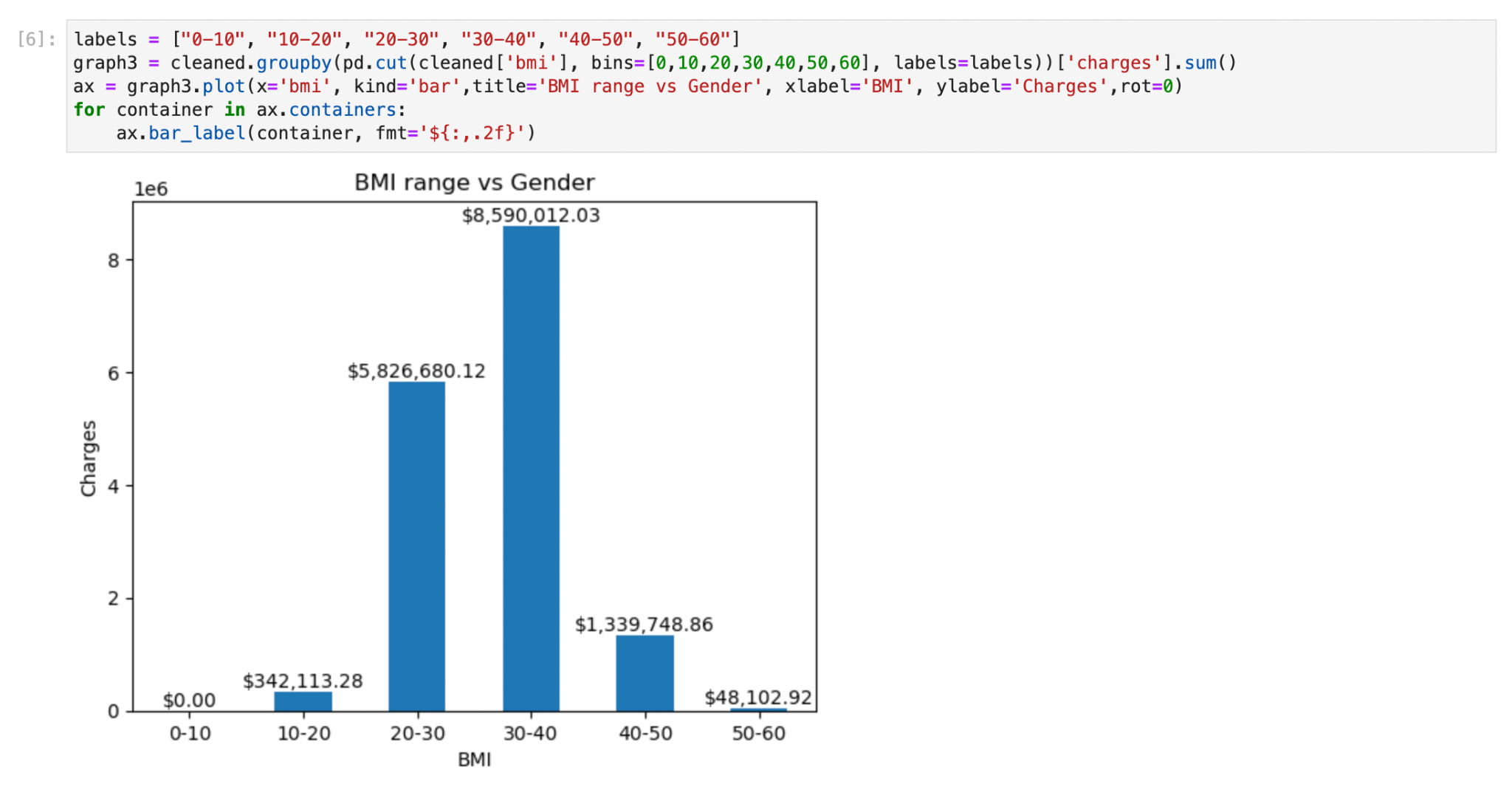
labels = ["0-10", "10-20", "20-30", "30-40", "40-50", "50-60"]

graph3 = cleaned.groupby(pd.cut(cleaned['bmi'], bins=[0,10,20,30,40,50,60], labels=labels))['charges'].sum()

ax = graph3.plot(x='bmi', kind='bar',title='BMI range vs Charges', xlabel='BMI', ylabel='Charges',rot=0)

for container in ax.containers:

ax.bar\_label(container, fmt='${:,.2f}')



I wanted to find out if the different ranges of BMI will incur higher medical costs. As shown in the graph, the BMI range of 30-40 has the highest charges out of all the other ranges.

**Question 3:**

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

# Encode categorical variables

encoder = LabelEncoder()

raw['sex'] = encoder.fit\_transform(raw['sex'])

raw['region'] = encoder.fit\_transform(raw['region'])

# Split the dataset into features and target variable

X = raw[['age', 'sex', 'bmi', 'children', 'region', 'charges']]

y = raw['smoker']

# Split the data into a training set and a testing set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

In order to separate the dataset into features and the target variable, as well as a training set and a testing set for validation, I transformed categorical variables—like "sex" and "region"—into a numerical format.

from sklearn.tree import DecisionTreeClassifier

# Create a Decision Tree classifier

dt\_classifier = DecisionTreeClassifier()

# Fit the model to the training data

dt\_classifier.fit(X\_train, y\_train)

After cleaning up the data, I created the decision tree model from Scikit-Learn.

import matplotlib.pyplot as plt

from sklearn.tree import plot\_tree

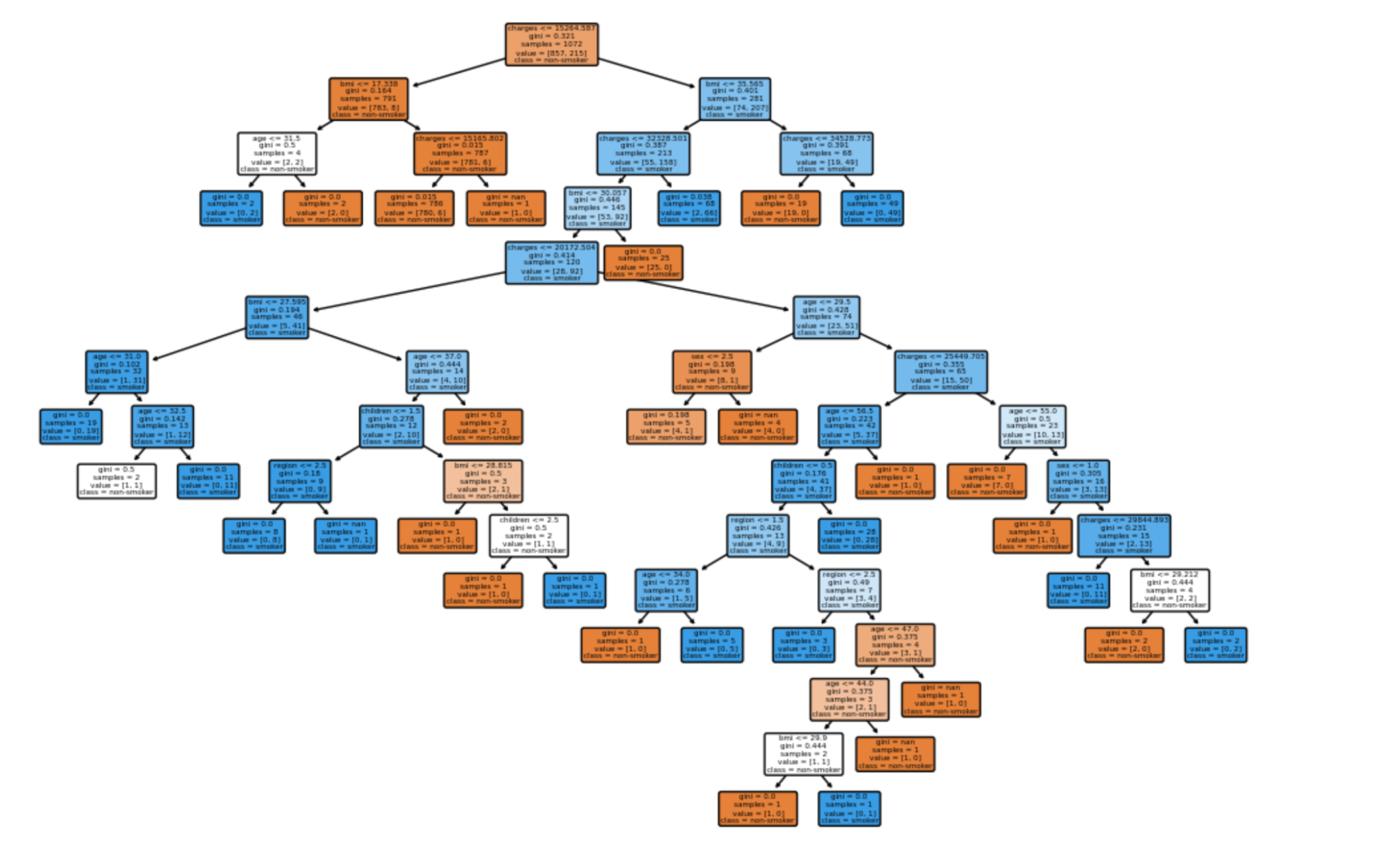
plt.figure(figsize=(12, 8))

plot\_tree(dt\_classifier, feature\_names=X.columns, class\_names=['non-smoker', 'smoker'], filled=True, rounded=True)

plt.show()

The decision-making process of my model is made easier to interpret and comprehend by using the code mentioned above, which plots the decision tree with labeled nodes and class distributions at each node.

**Question 4:**

****

Based on the code I created above, this is the decision tree that has been plotted.

**Question 5:**

It is true that decision trees have applications outside of prediction, such as exploratory data analysis (EDA). To comprehend a dataset more thoroughly, exploratory data analysis (EDA) entails locating patterns, relationships, and insights within the data. (Analyticsvidhya, 2023) In this case, decision trees provide a number of advantages:

1. Feature Importance: Decision trees can show how crucial certain features are for explaining variation in the target variable. The splits and nodes in the tree allow analysts to pinpoint the important factors influencing the behavior of the data, which aids in determining which investigations should be prioritised. (Analyticsvidhya, 2023)

2. Segmentation: Based on feature values, decision trees divide the data into subsets. By highlighting specific subpopulations within the data, this segmentation can help uncover hidden trends or patterns that traditional summary statistics might miss.

3. Outlier Detection: By separating data points that do not match the general patterns shown in the tree structure, decision trees can find outliers, which may point to questions about the quality of the data or anomalous observations.

4. Variable Relationships: Decision tree structures help analysts visualise the relationships between various variables and the target variable. (Analyticsvidhya, 2023) This information can help guide future exploratory data analysis and the development of hypotheses.

5. Data Preprocessing: Decision trees can provide guidance on the need for feature engineering, suggest imputation techniques, or highlight missing values in data preprocessing steps.

Even though decision trees are an effective tool for EDA, not all datasets will benefit from their use. They may fail to detect nuances in nonlinear relationships and overfit noisy data. (Analyticsvidhya, 2023) A more thorough understanding of the data can be obtained by combining decision tree analysis with other EDA techniques, such as statistical testing and data visualisation.

**Reference List**

A. (2023, March 2). *Tree Based Algorithms: A Complete Tutorial from Scratch (in R &#038; Python)*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/