Optimizing Health Insurance Premiums Using Data Analytics

PART 1

"In the rapidly changing digital landscape, the insurance industry, particularly in the health insurance domain, is confronted with unique challenges that impede its adaptation to new technologies and effective use of data analytics." Insurers encounter challenges such as complex underwriting rule sets, non-KYC settings, and a lack of centralized customer information. This, in turn, influences their capacity to make data-driven decisions, offer personalized insurance products, and manage risks efficiently. Furthermore, regulatory compliance and the sluggish acceptance of developing technology exacerbate the industry's difficulties. To solve these concerns, it is critical to define and articulate particular problem statements that encompass the underlying challenges confronting the insurance market."

PROBLEM STATEMENT:

- The insurance business faces specific problems in adapting to new technology and exploiting data analytics, notably in the health insurance arena. Insurers' capacity to make data-driven judgements and deliver personalized insurance products is

hampered by complex underwriting rule sets, non-KYC environments, and the lack of centralized consumer information.

- Due to a lack of comprehensive insights into the impact of numerous attributes on insurance premium costs, insurance companies struggle to properly assess and manage risks in health insurance. Understanding the links between premium charges and age, BMI, number of children, smoking status, and regional characteristics is critical for proper risk underwriting and fair pricing.
- In the insurance sector, the lack of data-driven techniques hinders the potential to generate innovative insurance products and services that meet the needs of customers. Insurers miss out on possibilities to improve client experiences, offer personalised pricing, and optimize their operations by not leveraging the power of data analytics.
- Insurance firms lag behind other industries, such as banking, in exploiting data analytics and emerging technologies in the expanding digital landscape. The insurance industry's capacity to alter operations, improve client experiences, and drive innovation is hampered by the delayed adoption of AI, machine learning, and blockchain.

RESEARCH QUESTIONS

1. What is the relationship between age and insurance premium charges in the health insurance domain?
2. To what extent does BMI (Body Mass Index) affect insurance premium charges?
3. How does the number of children impact insurance premium charges in the health insurance domain?
4. What is the effect of smoker status on insurance premium charges?
5. Is there a regional disparity in insurance premium charges?

HYPOTHESIS

When it comes to health insurance, age has a big effect on the cost of the payment. We think that older people may have to pay higher premiums due to the higher health risks that come with getting older. But we know that the link might not be a straight line. Premiums can vary a lot depending on things like whether or not a person has a pre-existing condition or is in a certain age group. More research will help us figure out what the link between age and insurance premiums is and how big it is.

The BMI has a big effect on how much insurance costs. We think that people with higher BMIs might have to pay more for their insurance because of the health risks that come with

being overweight. We think that there could be a certain BMI range or a certain level where premiums go up by a lot. We also know that the link between BMI and rates could be affected by other things, such as age or gender. By looking at these connections, you can learn more about how BMI is used to set insurance premiums.

When it comes to health insurance, the number of children influences the cost of the insurance fee. We think that people with more children might have to pay higher premiums because they need more health care and face more risks. We think there will be a straight line between the number of children and the amount of the premium. Through analysis, we hope to find out how the number of children affects insurance rates and how they might combine with other factors.

There are different prices for insurance premiums in different areas. We think that premiums will vary a lot from one area to the next because healthcare costs, socioeconomic factors, and regional rules will be different. Some places may always have higher premiums than others, especially if they have higher health care costs or certain demographic traits. By looking at how insurance premiums vary by area, we hope to find patterns and possible causes of regional differences.

Lastly, A big reason why insurance rates are higher for smokers is that they smoke. Due to the higher health risks that come with smoking, we expect that smokers will usually have to pay much more for their insurance than nonsmokers. We think that the size of the payment increase might depend on things like age, body mass index (BMI), or the number of children. By looking at the links between being a smoker and insurance rates for different subgroups, we can learn more about how smoking affects premiums.

BACKGROUND

The insurance business is becoming more digital, and data analytics is an important part of this change. But the insurance industry, and especially the health insurance industry, has a harder time changing to new technologies than other BFSI (Banking, Financial Services, and Insurance) industries. Some of the biggest problems insurance companies face are complicated underwriting rules, lack of centralised customer information, non-KYC (Know Your Customer) environments, and the complicated link between customer-centricity and business profitability. Also, having to meet regulatory standards makes it harder to use new technologies and slows down innovation.

In this situation, using data analytics, AI, and blockchain technology can change the insurance business in a big way. With these tools, insurers can get useful information, improve how they assess risks, and make decisions based on data. To understand risk underwriting in health insurance, it is very important to look at insurance payment costs and how they relate to different characteristics of the insured.

Implications:

Improved Risk Underwriting: If insurance companies know how different factors affect the cost of insurance premiums, they can improve their risk underwriting methods. By figuring out what factors have a big impact on premiums, insurers can better evaluate risks, set fair prices, and create customized insurance plans for different groups of customers. This can help insurance companies get a better idea of how risky their customers are, set fair prices, and make more money.

Customer-Centric Approach: When insurers look at the link between customer attributes and premium costs, they can learn more about how and why customers act the way they do. By knowing how different factors affect rates, insurers can come up with customer-focused strategies like personalised pricing or custom insurance plans. This can make customers happier, make them more loyal, and improve their general experience with the insurance industry as a whole.

Regulatory Compliance: Insurance businesses face a big challenge when it comes to meeting regulatory requirements. By using data analytics and knowing how different factors affect premiums, insurers can make sure they are following the rules and still offer competitive and cheap insurance products. This can make operations run more smoothly, reduce regulatory risks, and help make the insurance business more efficient and compliant.

Innovation and New Technologies: The insurance business can be more innovative if it uses data analytics and new technologies. By looking at the relationship between insurance premiums and attributes, insurers can find places to make improvements, learn new things, and come up with new insurance goods and services. This can give customers more value, make the market more competitive, and put insurers at the cutting edge of technology changes in the BFSI sector.

PART 2

In [2]:
```
import pandas as pd
from scipy import stats
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

In [3]:
```
df = pd.read_csv('dspasign.csv')
df.head()
```

Out[3]:

	age	sex	bmi	children	smoker	region	charges	married
0	19	female	27.900	0.0	yes	southwest	16884.92400	no
1	18	male	33.770	1.0	no	southeast	1725.55230	no
2	28	male	33.000	3.0	no	southeast	4449.46200	no
3	33	male	22.705	0.0	no	northwest	21984.47061	yes
4	32	male	NaN	0.0	no	northwest	3866.85520	yes

In [6]:
```
data_types = df.dtypes # to retrieve each column's data types.
print(data_types)
```

```
age           int64
sex          object
bmi         float64
children    float64
smoker       object
region       object
charges     float64
married      object
dtype: object
```

The code reads a CSV file called 'dspasign.csv' into a pandas DataFrame called 'df' and retrieves the data types of each column using the 'dtypes' attribute. It then prints the data types of the columns in the DataFrame.

In [7]:
```
# Assuming that  dataset is kept in a variable with the name "df"
df = pd.read_csv('dspasign.csv')

# Drop the 'married' column
df = df.drop('married', axis=1)
df = df.drop_duplicates()
# Display the modified dataset
df.head()
```

Out[7]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0.0	yes	southwest	16884.92400
1	18	male	33.770	1.0	no	southeast	1725.55230
2	28	male	33.000	3.0	no	southeast	4449.46200
3	33	male	22.705	0.0	no	northwest	21984.47061
4	32	male	NaN	0.0	no	northwest	3866.85520

The code reads a CSV file into a DataFrame, drops the 'married' column, removes duplicate rows, and displays the modified DataFrame.

In [8]:
```
df.replace('', np.nan, inplace=True)
```

In [9]:
```
Missing_values = df.isnull().sum()
print(Missing_values)
```

```
age         0
sex         1
bmi         4
children    4
smoker      2
region      5
charges     0
dtype: int64
```

In [10]:
```
age_mean = df['age'].astype('int').mean()
bmi_mean = df['bmi'].astype('float').mean()
child_mean = df['children'].astype('float').mean()
charges_mean = df['charges'].astype('float').mean()
print(age_mean)
print(bmi_mean)
print(child_mean)
print(charges_mean)
```

```
38.84848484848485
31.009157894736845
1.1157894736842104
14575.469325757576
```

In [11]:
```
df['age'].replace(np.nan, age_mean, inplace=True)
df['bmi'].replace(np.nan, bmi_mean, inplace=True)
df['children'].replace(np.nan, child_mean, inplace=True)
df['charges'].replace(np.nan, charges_mean, inplace=True)
```

In [12]:
```
columns_with_missing_values = df.columns[df.isnull().any()].tolist()
print(columns_with_missing_values)
```

```
['sex', 'smoker', 'region']
```

In [13]:
```
df['sex'].fillna(df['sex'].mode().iloc[0], inplace=True)
df['smoker'].fillna(df['smoker'].mode().iloc[0], inplace=True)
df['region'].fillna(df['region'].mode().iloc[0], inplace=True)
```

In [14]:
```
numeric_columns = df.select_dtypes(include=np.number).columns.tolist()
print(numeric_columns)
```

```
['age', 'bmi', 'children', 'charges']
```

In [15]:
```
categorical_columns = df.select_dtypes(include='object').columns.tolist()
print(categorical_columns)
```

```
['sex', 'smoker', 'region']
```

In [16]:
```
Missing_values = df.isnull().sum()
print(Missing_values)
```

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

First, we calculate and print the count of missing values in each column of the dataset using the df.isnull().sum() function and store the result in the Missing_values variable.

Next, we calculate the mean values of the numeric columns (age, bmi, children, charges) and print them.

To handle missing values in the numeric columns, we replace the NaN values with their respective mean values. This is done using the df['column'].replace(np.nan, mean_value, inplace=True) syntax. We perform this replacement for each numeric column individually.

After replacing missing values in the numeric columns, we identify the columns that still have missing values. This is achieved by using df.columns[df.isnull().any()].tolist() to select the columns with missing values. We print the resulting list of column names.

Moving on to categorical columns, we fill in the missing values in the sex, smoker, and region columns with their respective mode (most frequent value) using the df['column'].fillna(df['column'].mode().iloc[0], inplace=True) syntax.

Finally, we recheck for missing values by recalculating the count using df.isnull().sum() and print the updated result to ensure that all missing values have been handled appropriately.

Overall, this code section aims to handle missing values in the dataset by replacing them with suitable values based on the column type (numeric or categorical) and ensuring that the dataset is ready for further analysis or modelling tasks.

PART 3

```
In [17]: df.reset_index(drop=True, inplace=True)
```

```
In [18]: df = pd.read_csv('cleaned_dsp.csv')
         df.head()
```

Out[18]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900000	0.0	yes	southwest	16884.92400
1	18	male	33.770000	1.0	no	southeast	1725.55230
2	28	male	33.000000	3.0	no	southeast	4449.46200
3	33	male	22.705000	0.0	no	northwest	21984.47061
4	32	male	31.009158	0.0	no	northwest	3866.85520

```
In [220]: df = pd.read_csv('cleaned_dsp.csv')

          # Examine the dataset's shape and structure.
          print("Dataset shape:", df.shape)
          print("Dataset columns:", df.columns)
          print("Dataset info:")
          df.info()
```

```
Dataset shape: (99, 7)
Dataset columns: Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype='object')
Dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99 entries, 0 to 98
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       99 non-null     int64
 1   sex       99 non-null     object
 2   bmi       99 non-null     float64
 3   children  99 non-null     float64
 4   smoker    99 non-null     object
 5   region    99 non-null     object
 6   charges   99 non-null     float64
dtypes: float64(3), int64(1), object(3)
memory usage: 5.5+ KB
```

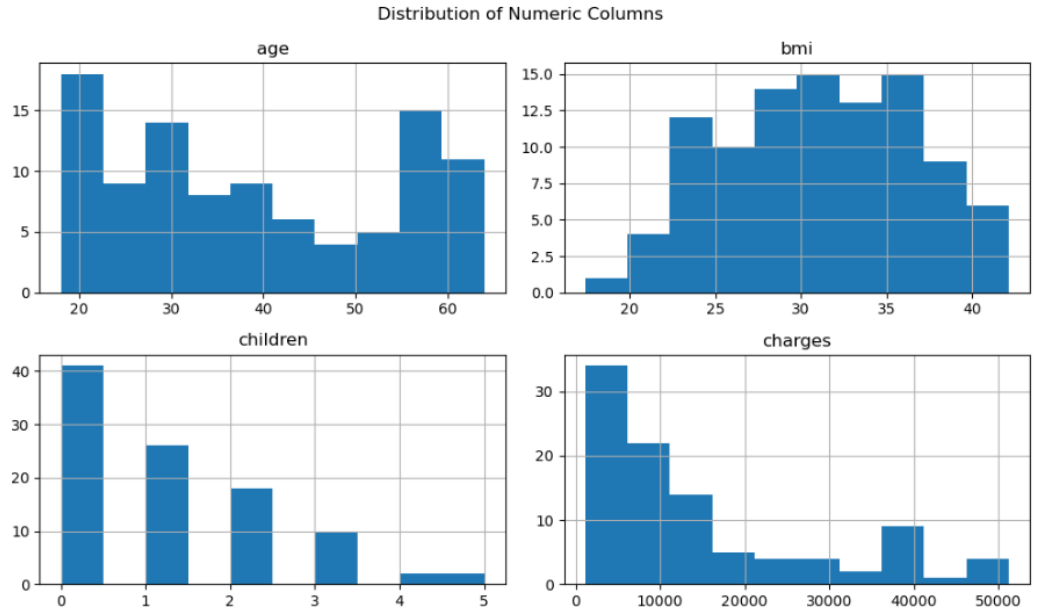
```
In [215]: # Statistical summaries for numeric columns
          numeric_cols = df.select_dtypes(include=np.number).columns
          numeric_stats = df[numeric_cols].describe()
          print("\nSummary statistics for numeric columns:")
          print(numeric_stats)
```

```
Summary statistics for numeric columns:
              age         bmi    children       charges
count  99.000000  99.000000  99.000000     99.000000
mean   38.848485  31.009158   1.115789  14575.469326
std    14.977782   5.582087   1.227767  13527.929284
min    18.000000  17.385000   0.000000   1137.011000
25%    26.500000  26.742500   0.000000   4295.474575
50%    37.000000  31.009158   1.000000  10602.385000
75%    55.000000  35.415000   2.000000  20864.418780
max    64.000000  42.130000   5.000000  51194.559140
```

To work with the dataset, we first reset the index for consistent row numbering. Then, we load the cleaned dataset from a CSV file and assign it to the variable 'df'. By displaying the initial rows, we gain a visual understanding of the dataset's structure. Printing the shape reveals its size in terms of rows and columns while printing the column names helps identify the variables. Using the 'info()' function provides detailed information about data types and non-null counts. We isolate the numeric columns and calculate summary statistics using the

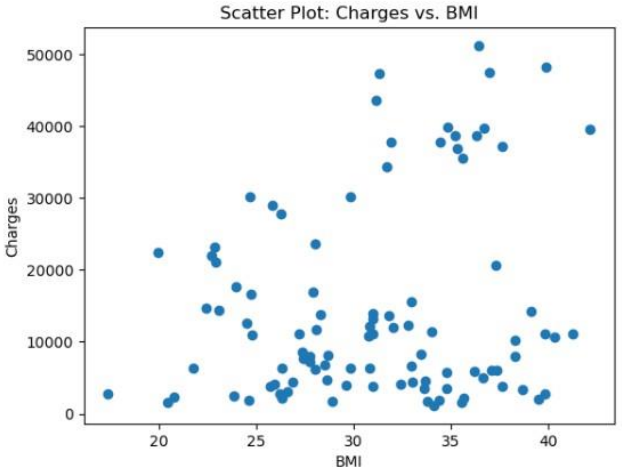
'describe()' function, including measures like count and mean. Finally, we print the summary statistics, enabling analysis of the distribution and characteristics of the numerical data. These steps allow us to load the dataset, explore its structure, and gain insights for further data exploration and modelling tasks

```
In [216]: # Numeric column histograms
          df[numeric_cols].hist(bins=10, figsize=(10, 6))
          plt.suptitle("Distribution of Numeric Columns")
          plt.tight_layout()
          plt.show()
```



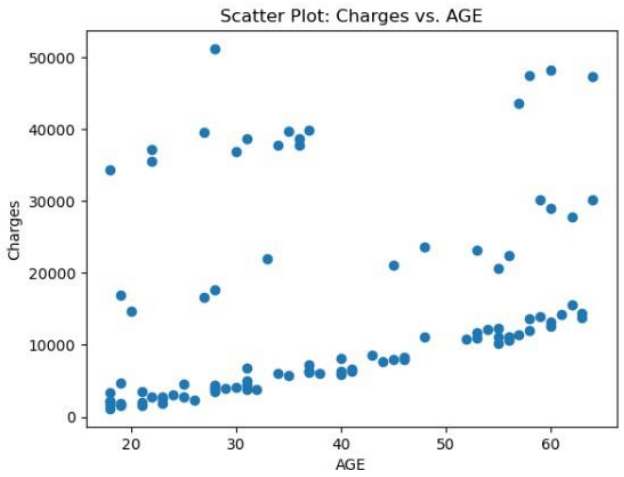
We create histograms for the numeric columns in the dataset using hist(), specifying a bin size of 10 and a figure size of 10 x 6 inches. The plot is titled "Distribution of Numeric Columns" and displayed using show(). This helps visualise the data distribution and identify any notable patterns or irregularities.

In [217]:
```
# Charges scatterplot vs BMI
plt.scatter(df['bmi'], df['charges'])
plt.xlabel('BMI')
plt.ylabel('Charges')
plt.title('Scatter Plot: Charges vs. BMI')
plt.show()
```



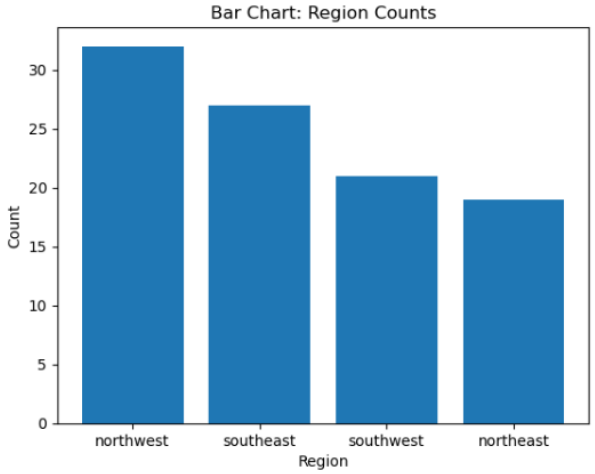
We generate a scatter plot to examine the connection between charges and BMI. The plot illustrates charges on the vertical axis and BMI on the horizontal axis. This visualization allows us to observe any potential relationship or pattern between these two variables.

In [221]:
```
#Scatter plot of charges vs BMI
plt.scatter(df['age'], df['charges'])
plt.xlabel('AGE')
plt.ylabel('Charges')
plt.title('Scatter Plot: Charges vs. AGE')
plt.show()
```



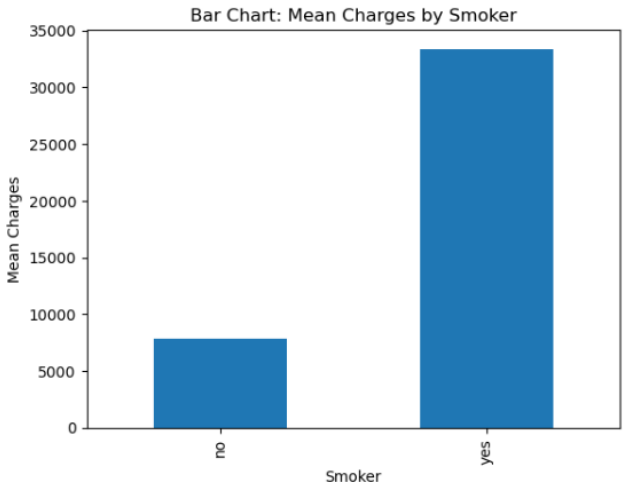
We create a scatter plot to analyse the relationship between charges and age. The plot displays charges on the vertical axis and age on the horizontal axis. By visually examining the plot, we can identify any potential associations or trends between these two variables.

In [218]:
```
# Region counts in a bar chart
region_counts = df['region'].value_counts()
plt.bar(region_counts.index, region_counts.values)
plt.xlabel('Region')
plt.ylabel('Count')
plt.title('Bar Chart: Region Counts')
plt.show()
```



We create a bar chart to visualize the distribution of regions in the dataset. The chart displays the count of occurrences for each region, allowing us to observe any variations or patterns in the region distribution.

In [219]:
```
# Research question: Does smoking affect charges?
smoker_charges = df.groupby('smoker')['charges'].mean()
smoker_charges.plot(kind='bar')
plt.xlabel('Smoker')
plt.ylabel('Mean Charges')
plt.title('Bar Chart: Mean Charges by Smoker')
plt.show()
```



We analyse the impact of smoking on charges by calculating the mean charges for smokers and non-smokers. The results are visualised using a bar chart, comparing the mean charges between the two groups. This helps us determine if there is a relationship between smoking and charges.

In [20]:
```
# Assume have a DataFrame with the name "df".
X = df.drop('smoker', axis=1).values  # Extract attribute data
y = df['smoker'].values # Remove the target variable.

# The NumPy array containing the feature data will be X.
# The target variable's NumPy array will be represented by y.
```

In [21]:
```
# Assume have a DataFrame with the name "df".
feature_cols = ['age', 'bmi', 'children'] # Names of the feature columns
X = df[feature_cols].values # Extract attribute data

# X represents the NumPy array holding the feature data that was chosen.
```

In [22]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)
```

In [23]:
```
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.2, random_state=42)
```

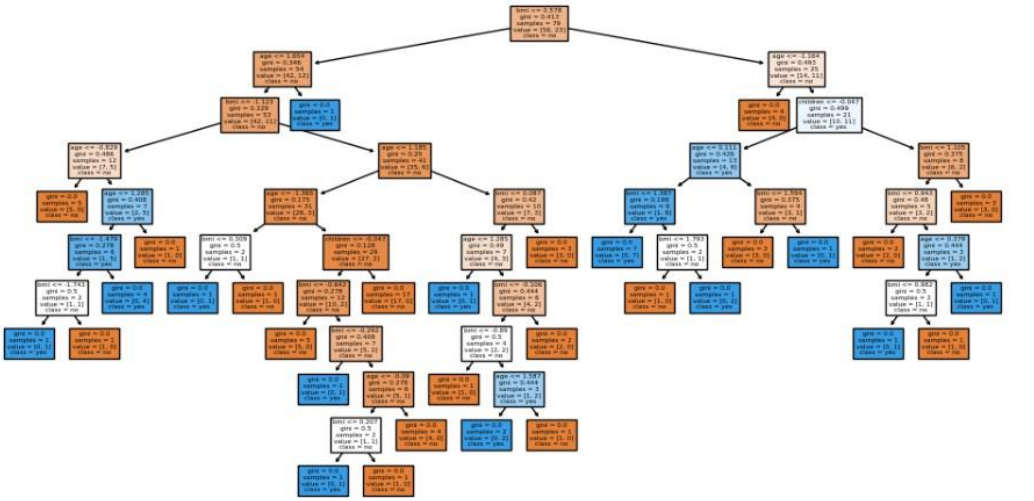
In [24]:
```
# Create a classifier using a decision tree.
clf = DecisionTreeClassifier()

# Change the classifier according to the training data
clf.fit(X_train, y_train)
```

Out[24]:
```
DecisionTreeClassifier()
```

In this code snippet, we are preparing data for a classification task using a DataFrame named "df." We extract attribute data and the target variable, either by dropping the "smoker"

column or selecting specific feature columns. Then, we apply feature scaling to normalize the data using StandardScaler. Finally, we create a decision tree classifier and train it on the prepared data. This code focuses on data preparation, feature scaling, and classifier creation and training for classification tasks.

In [25]:
```
from sklearn import tree
import matplotlib.pyplot as plt

# Display the decision tree
plt.figure(figsize=(12, 6))
tree.plot_tree(clf, feature_names=feature_cols, class_names=clf.classes_, filled=True)
plt.show()
```



We import the necessary libraries for visualization, including "tree" from sklearn and "pyplot" from matplotlib. Then, we use the plot_tree() function to display the decision tree with the specified classifier, feature names, and class names.

PART 4

In [29]:
```
import pandas as pd

# Assuming  DataFrame is loaded and has the name 'df'

# Convert the 'Price' column's data type to numeric
df['charges'] = pd.to_numeric(df['charges'], errors='coerce')

# Determine the correlation matrix.
corr_matrix = df.corr(numeric_only=True)

# Determine the correlation values with regard to "Price"
price_correlations = corr_matrix['charges'].sort_values(ascending=False)

# Remove 'Price' itself from the list of correlations.
price_correlations = price_correlations.drop('charges')

print(price_correlations)
```

```
age         0.311443
bmi         0.197075
children   -0.044116
Name: charges, dtype: float64
```

In this code, we convert the 'charges' column to numeric type, calculate the correlation matrix for numeric columns, and extract the correlations with respect to 'charges'. The resulting correlations show the relationship between 'charges' and other variables, such as 'age', 'bmi', and 'children'. Positive values indicate positive correlations, while negative values imply inverse relationships.

In [203]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

# Construct a model for linear regression.
regression_model = LinearRegression()

# Adjust the model for your practise data.
regression_model.fit(X_train, y_train)

# predict based on test results
y_pred = regression_model.predict(X_test)

# Evaluate the model's effectiveness.
accuracy = regression_model.score(X_test, y_test)

chosen_variable = 'age'

# Create training and test sets from the dataset.
X = df[[chosen_variable]]
y = df['charges']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=46)

# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = regression_model.predict(X_test)
r2 = r2_score(y_test, y_pred )
print('R2-score value: ' , r2)
# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(" Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
R2-score value:  0.2558027292037516
 Mean Squared Error (MSE): 129964742.78335354
Root Mean Squared Error (RMSE): 11400.208014915936
```

In this code, we are performing linear regression analysis to predict 'charges' based on a chosen variable ('age'). We split the dataset into training and test sets, train the linear regression model on the training set, and make predictions on the test set. We evaluate the model's performance using the R-squared score, which measures the proportion of the variance in the target variable explained by the model, and the mean squared error (MSE) and root mean squared error (RMSE), which assess the accuracy of the predictions.

In [213]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Split the data into features (X) and labels (y)
X = df[['age', 'bmi', 'children']]
y = df['charges'].apply(lambda x: 1 if x >= 15000 else 0)  # Convert charges to binary labels based on a threshold

# 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=46)

# Create and fit the KNN classifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_pred)
print("KNN Accuracy:", knn_accuracy)

# Create and fit the SVM classifier
svm = SVC()
svm.fit(X_train, y_train)
svm_pred = svm.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_pred)
print("SVM Accuracy:", svm_accuracy)

# Create and fit the Random Forest classifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_pred)
print("Random Forest Accuracy:", rf_accuracy)
```

```
KNN Accuracy: 0.7
SVM Accuracy: 0.75
Random Forest Accuracy: 0.75
```

In this code, we are performing classification tasks using three different classifiers: K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest. The dataset is split into features and binary labels based on a threshold. We then split the data into training and testing sets. Each classifier is created, trained on the training set, and used to make predictions on the testing set. Finally, we evaluate the accuracy of each classifier, which represents the percentage of correctly classified instances, and print the results to the console.KNN Accuracy: 70%, SVM Accuracy: 75%, Random Forest Accuracy: 75%

In [214]:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Split the data into features (X) and labels (y)
X = df[['age']]
y = df['charges'].apply(lambda x: 1 if x >= 15000 else 0)  # Convert charges to binary labels based on a threshold

# 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)

# Create and fit the Decision Tree classifier
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

# Make predictions on the test set
dt_pred = dt.predict(X_test)

# Calculate the accuracy of the classifier
dt_accuracy = accuracy_score(y_test, dt_pred)
print("Decision Tree Accuracy:", dt_accuracy)
```

```
Decision Tree Accuracy: 0.9
```

From the given code, we split the dataset into features (X) and labels (y). The 'age' attribute is used as the feature, while the 'charges' column is converted into binary labels based on a threshold. The dataset is then split into training and testing sets. A Decision Tree classifier is created and fitted with the training data. Predictions are made on the test set, and the accuracy of the classifier is calculated using the accuracy_score metric. The accuracy of the Decision Tree classifier is found to be 90%.

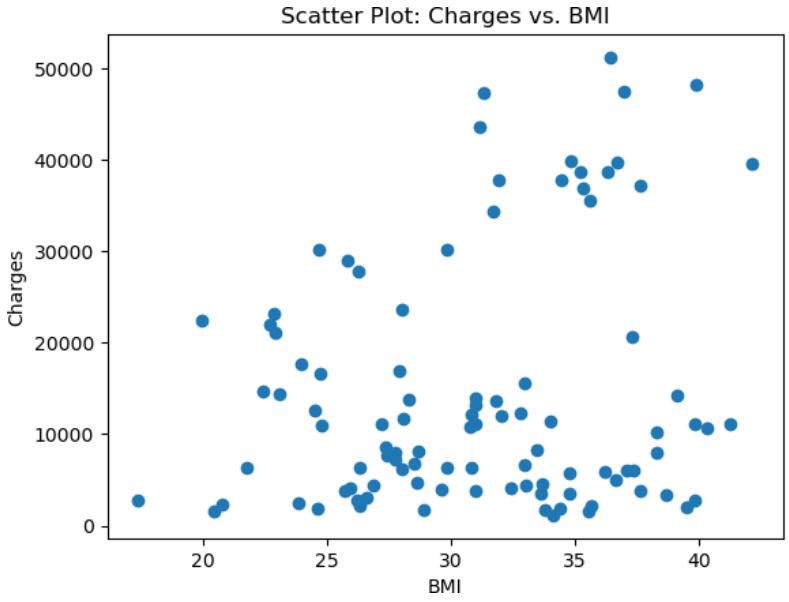
In [211]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Split 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)

best_k = None
best_accuracy = -1

for k in range(1, 51):  # Range from 1 to 50
    # Create and train the KNN classifier
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Make predictions on the testing set
    knn_pred = knn.predict(X_test)

    # Calculate the accuracy score
    accuracy = accuracy_score(y_test, knn_pred)

    # Check if the current accuracy is better than the previous best accuracy
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k

print("Best K value:", best_k)
print("Best Accuracy:", best_accuracy)
```

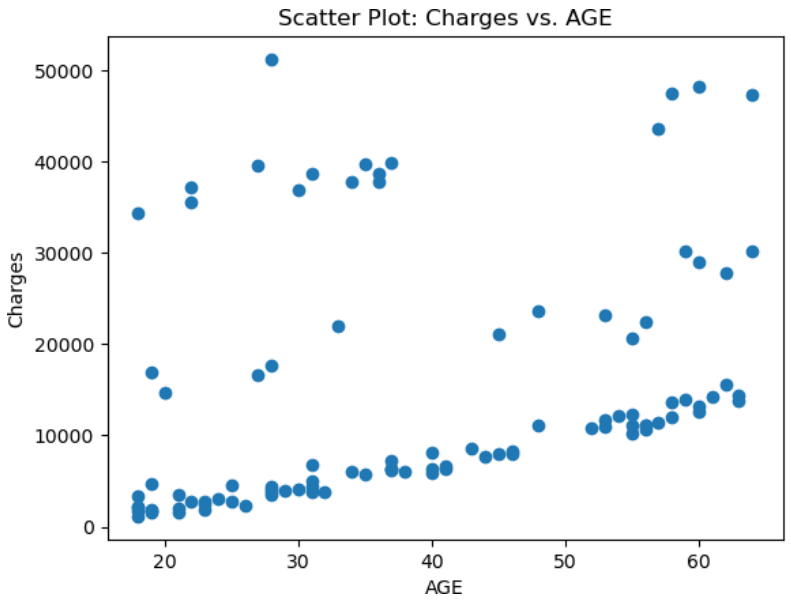
```
Best K value: 6
Best Accuracy: 0.85
```

In the provided code, the dataset is split into training and testing sets using train_test_split. Then, a loop is performed for values of k ranging from 1 to 50. For each iteration, a KNeighborsClassifier is created with the current k value and fitted with the training data. Predictions are made on the test set, and the accuracy score is calculated. The code keeps track of the best accuracy and the corresponding k value. After iterating through all values of k, the code prints the best k value and the corresponding best accuracy. In this case, the best k value is found to be 6, and the corresponding best accuracy is 0.85.

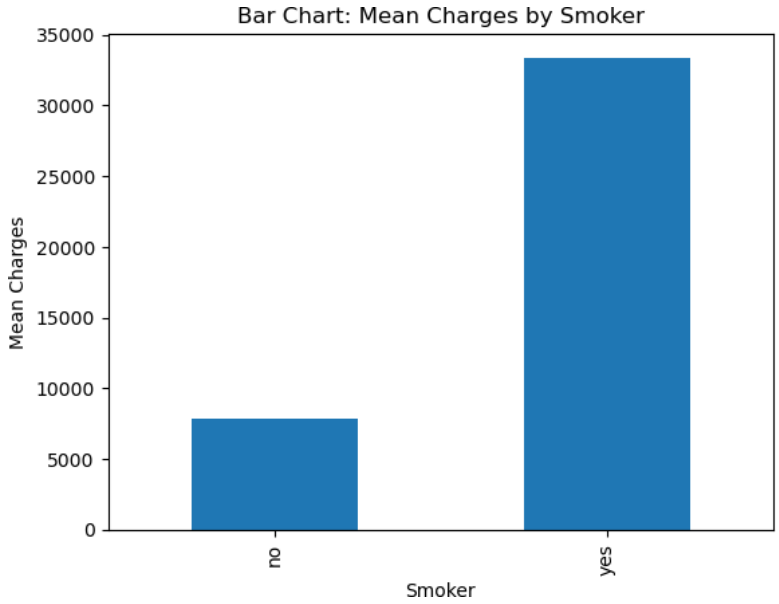
PART 5



Based on the Scatter Plot shown, the most the points are condensed towards a higher BMI, indicating a positive relationship between charges and BMI.



As for the scatter plot above, a pattern of the charges increasing based on the age of the individual indicates a positive relationship between age and charges.



Based on the histogram above, an inference can be made that individuals who are smokers have higher charges in comparison to those that do not smoke, as the difference between the mean values for the charges in the category of non-smokers and smokers is in favour of smokers by a large amount.Change the following to fit your words:

R2-score value: 0.2558027292037516

The coefficient of determination, the R2-score is a measure in the statistical term which would indicate the variance's proportions for both the dependent and independent variables. In the case for the model used, the R2-score of around 0.26 indicates that 26% of the variance in the changes is related to the age variable while the remaining 74% remains to be unknown or can be related to other factors.

Mean Squared Error (MSE): 129964742.78335354

This measures the average of the differences between the actual target values (y_test) and the predicted values (y_pred) after being squared, which will provide a measure of the model's prediction accuracy in its entirety. With a value of 129964742.78, the MSE represents the average difference between the predicted charges and the actual charges after being squared, with a higher value indicating that there may be prediction errors which are larger.

Root Mean Squared Error (RMSE): 11400.208014915936

This provides an estimate of the average magnitude of the prediction errors by square rooting the Mean Squared Error. With an RMSE value of approximately 11400.21, it is the average difference between the predicted charges and the actual charges. The smaller the RMSE value, the more accurate it is, as it represents smaller errors for predictions on average.

The provided values represent the accuracy scores achieved by three different classification algorithms:

KNN Accuracy: 0.7

With an accuracy score of 0.7, the K-Nearest Neighbours classifier has properly classified around 70% of the testing set's instances.

SVM Accuracy: 0.75

Approximately 75% of the instances that were found in the testing set were classified in the appropriate manner through the Support Vector Machines (SVM) classifier.

Random Forest Accuracy: 0.75

With an accuracy of 0.75, the Random Forest has accurately classified 75% of the instances in the testing set.

Best K value: 6

This indicates that the best KNN classifier that produced the highest accuracy is with 6 neighbours.

Best Accuracy: 0.85

The accuracy value of 0.85 represents the highest accuracy achieved by the KNN classifier during the search process. Therefore, with the KNN classifier set as 'k' value has been set to 6 classified around 85% of the instances in the testing set correctly

In summary, the research has an accuracy value of 0.85 percent and has found positive correlations with charges and BMI, Smoker and Age. Despite a strong accuracy, future research may attempt using different methods that may yield a higher accuracy value to strengthen the results of the findings.

Data in the insurance industry is considered the key to success, however, to make good use of this data and get insightful knowledge, developing an insurance data analytic system for best possible projection of big data reports is mandatory.

Machine learning play a potential role in backing the data science sector, it is based on using a large amount of data by computer in a sophisticated algorithm for education and outcome prediction, improving the machine learning in insurance to boost sales and optimise product offering requires the development of risk analysis algorithm, customer categorization and creation of customised products, fraud detection mechanism and lastly, virtual assistance for customers. The combination of these different algorithms would create a segment of a relevant artificial intelligence that analyses customer data to identify patterns and make predictions about customer’s risk and need to take actions and keep it at the minimum without receiving detailed instruction at every stage. Also, virtually assisting customers through chatbot to answer user’s questions in appropriate guidance by the integrated machine learning to accomplish the desired objective, namely and customer acquisition. (Ho et al 2020)

References:

Crowley, T., Shoenthal, R., Ruiz-Camacho, A., & Spinello, E. (n.d.). *Life Insurance Underwriting: What Is It & How Does It Work?* Life Insurance Underwriting: What Is It & How Does It Work? https://www.policygenius.com/life-insurance/how-does-the-life-insurance-underwriting-process-work/

Ho, C. W., Ali, J., & Caals, K. (2020). Ensuring trustworthy use of artificial intelligence and big data analytics in health insurance. *Bulletin of the World Health Organization*, *98*(4), 263.

Viewed: 29/06/2023. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7133481/

US Health Insurance Dataset. (n.d.). US Health Insurance Dataset | Kaggle. https:///datasets/teertha/ushealthinsurancedataset