Import libraries

In [1]:
```
import numpy as np
import pandas as pd
data = pd.read_csv(r"C:\Users\Asus\Downloads\Health-Insurance-Premium-by-Kamlesh
from sklearn.feature_selection import VarianceThreshold
```

About Dataset

In [2]:
```
data.head(10)
```

Out[2]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
5	31	female	25.740	0	no	southeast	3756.62160
6	46	female	33.440	1	no	southeast	8240.58960
7	37	female	27.740	3	no	northwest	7281.50560
8	37	male	29.830	2	no	northeast	6406.41070
9	60	female	25.840	0	no	northwest	28923.13692

In [3]:
```
data.tail(20)
```

Out[3]:

	age	sex	bmi	children	smoker	region	charges
1318	35	male	39.710	4	no	northeast	19496.71917
1319	39	female	26.315	2	no	northwest	7201.70085
1320	31	male	31.065	3	no	northwest	5425.02335
1321	62	male	26.695	0	yes	northeast	28101.33305
1322	62	male	38.830	0	no	southeast	12981.34570
1323	42	female	40.370	2	yes	southeast	43896.37630
1324	31	male	25.935	1	no	northwest	4239.89265
1325	61	male	33.535	0	no	northeast	13143.33665
1326	42	female	32.870	0	no	northeast	7050.02130
1327	51	male	30.030	1	no	southeast	9377.90470
1328	23	female	24.225	2	no	northeast	22395.74424
1329	52	male	38.600	2	no	southwest	10325.20600
1330	57	female	25.740	2	no	southeast	12629.16560
1331	23	female	33.400	0	no	southwest	10795.93733
1332	52	female	44.700	3	no	southwest	11411.68500
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

TO Check null values

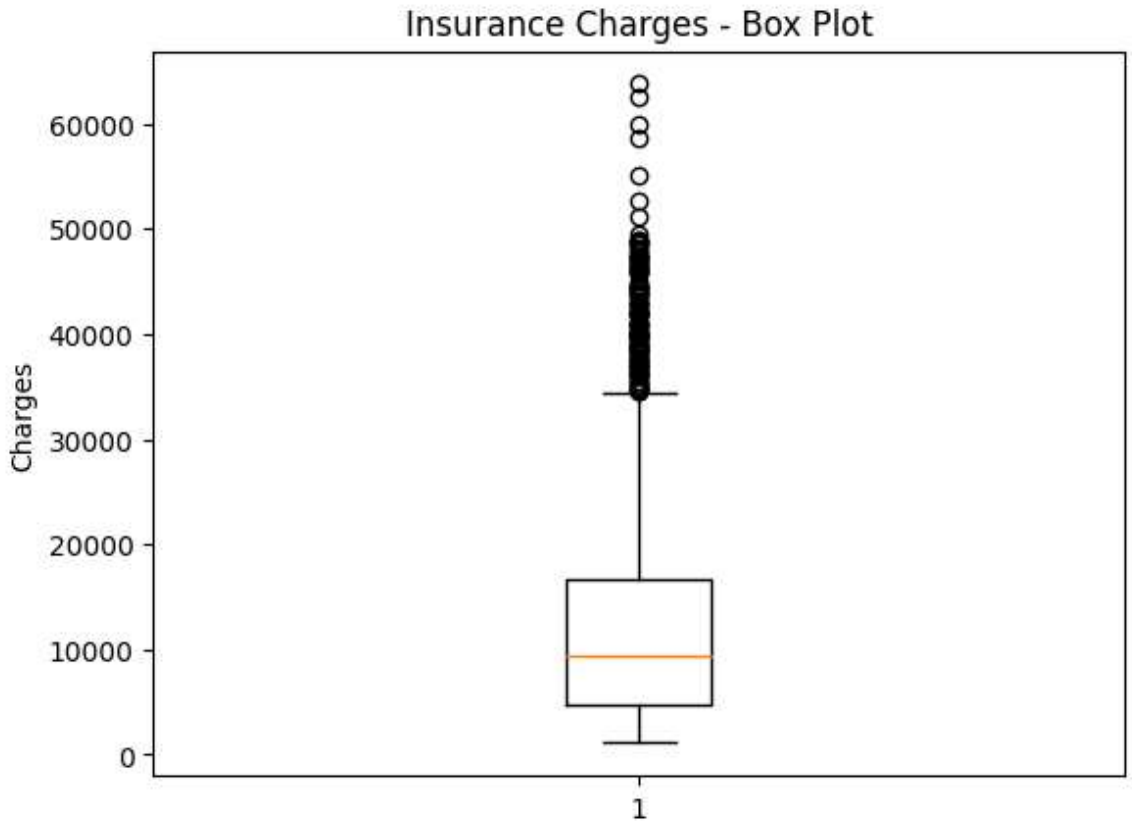
In [4]:
```
data.isnull().sum()
```

Out[4]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

Box Plot

In [5]:
```
import matplotlib.pyplot as plt

# Assuming you have stored your dataset in a pandas Data called 'data'
charges = data['charges']
```

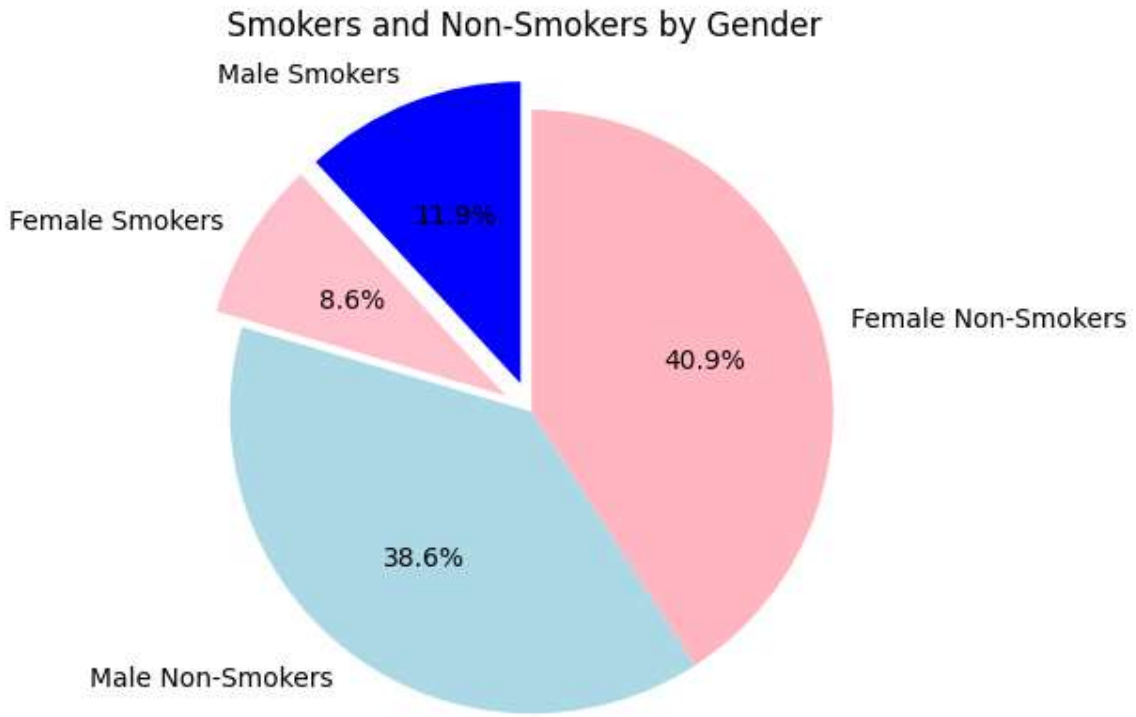
```
# Create the box plot
plt.boxplot(charges)

# Set the title and labels for the axes
plt.title("Insurance Charges - Box Plot")
plt.ylabel("Charges")

# Display the plot
plt.show()
```



Pie chart show dataset

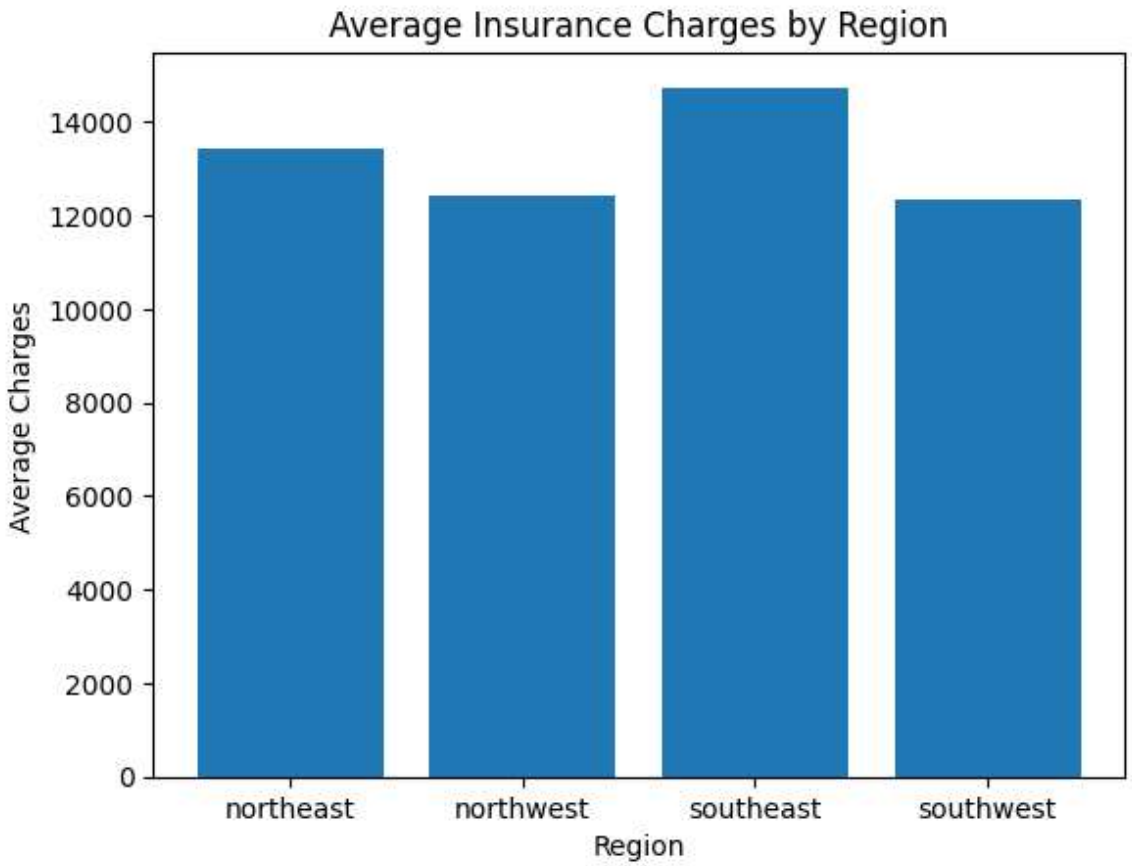
In [6]:
```
import matplotlib.pyplot as plt
import pandas as pd



# Count the number of smokers and non-smokers
smokers = data[data['smoker'] == 'yes']
non_smokers = data[data['smoker'] == 'no']

# Count the number of male and female smokers
male_smokers = smokers[smokers['sex'] == 'male'].shape[0]
female_smokers = smokers[smokers['sex'] == 'female'].shape[0]

# Count the number of male and female non-smokers
male_non_smokers = non_smokers[non_smokers['sex'] == 'male'].shape[0]
female_non_smokers = non_smokers[non_smokers['sex'] == 'female'].shape[0]

# Create the pie chart
labels = ['Male Smokers', 'Female Smokers', 'Male Non-Smokers', 'Female Non-Smok
sizes = [male_smokers, female_smokers, male_non_smokers, female_non_smokers]
```

```
colors = ['blue', 'pink', 'lightblue', 'lightpink']
explode = (0.1, 0.1, 0, 0)  # Explode the 'Male Smokers' and 'Female Smokers' sl

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle

plt.title('Smokers and Non-Smokers by Gender')
plt.show()
```



Bar graph

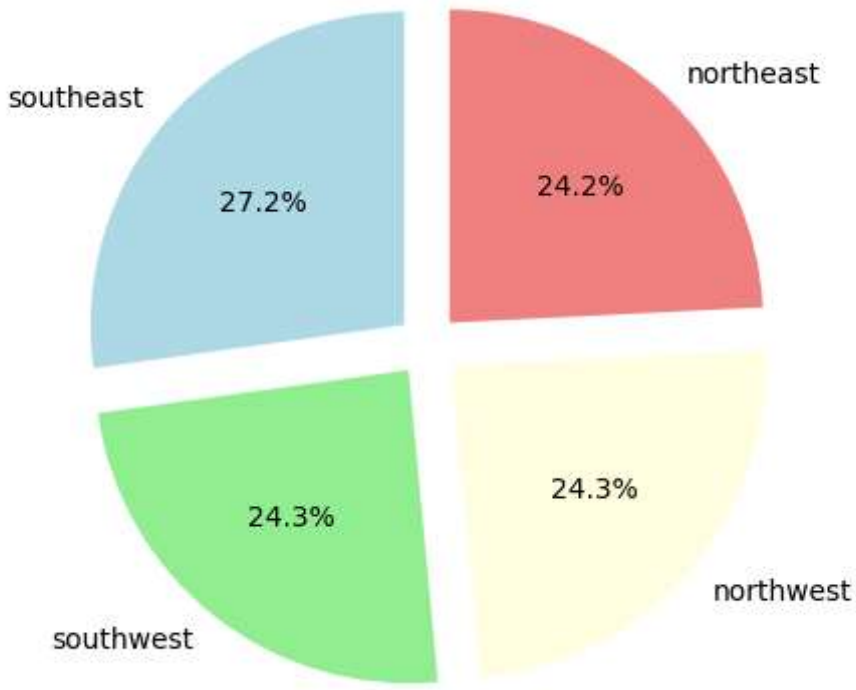
In [7]:
```
import matplotlib.pyplot as plt

# Assuming you have stored your dataset in a pandas DataFrame called 'df'
region_charges = data.groupby('region')['charges'].mean()

# Create the bar plot
plt.bar(region_charges.index, region_charges)

# Set the title and labels for the axes
plt.title("Average Insurance Charges by Region")
plt.xlabel("Region")
plt.ylabel("Average Charges")

# Display the plot
plt.show()
```



Pie chart show region percentage

In [8]:
```
import matplotlib.pyplot as plt
import pandas as pd



# Count the number of individuals in each region
region_counts = data['region'].value_counts()

# Calculate the percentage of individuals in each region
region_percentages = (region_counts / len(data)) * 100

# Create the pie chart
labels = region_percentages.index
sizes = region_percentages.values
colors = ['lightblue', 'lightgreen', 'lightyellow', 'lightcoral']
explode = (0.1, 0.1, 0.1, 0.1)  # Explode all the slices

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
```

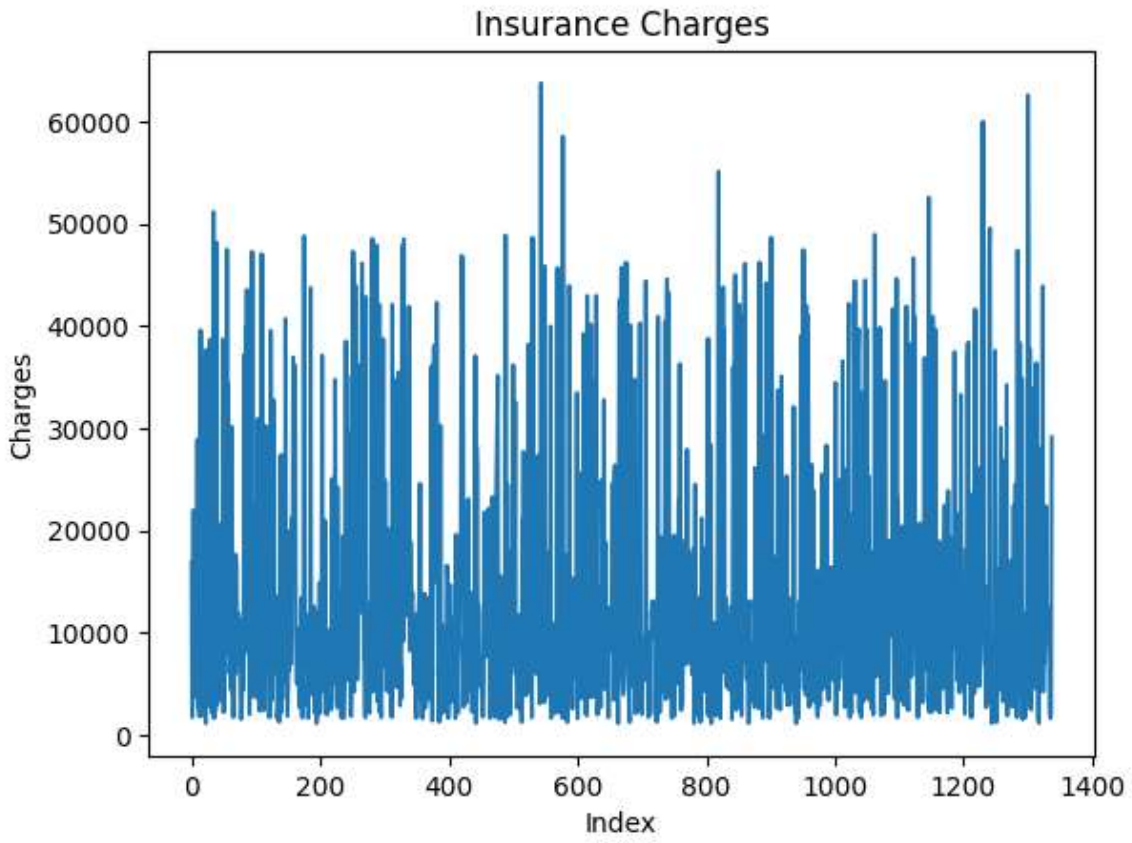
Out[8]:
```
(-1.1829856527049527,
 1.1831397445534941,
 -1.1859747271556071,
 1.1799754816745238)
```



line plot

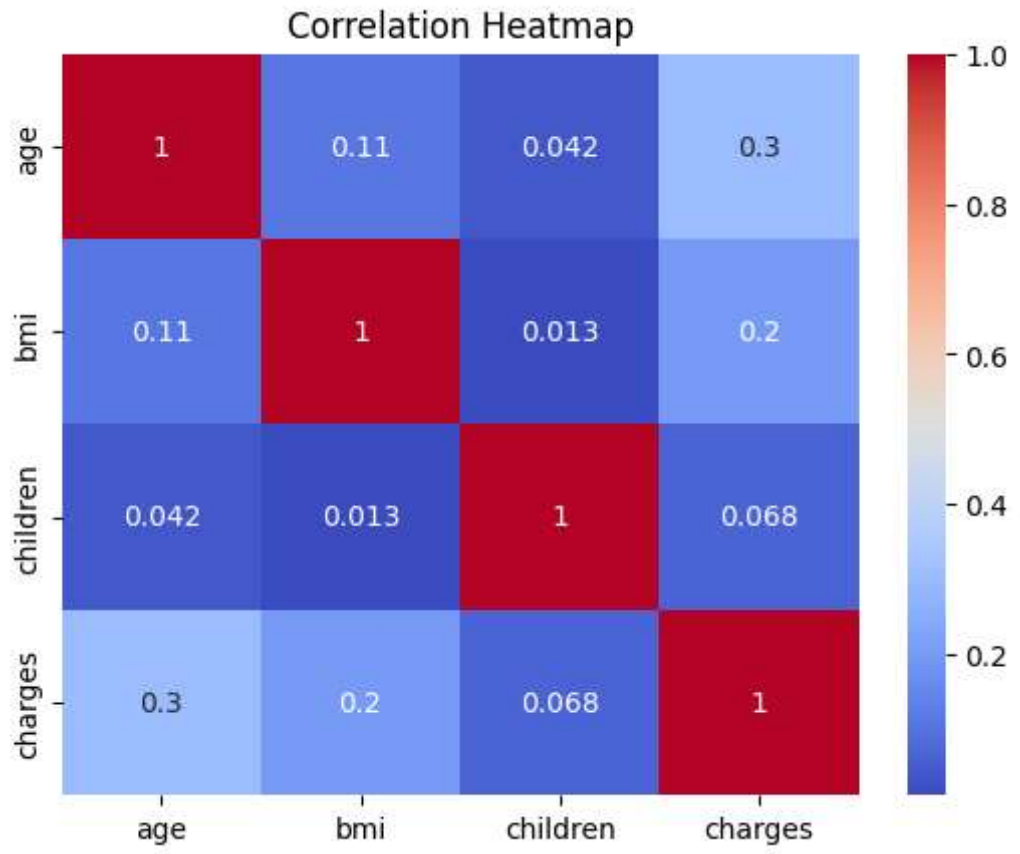
In [9]:

```python
import matplotlib.pyplot as plt

# Assuming you have stored your dataset in a pandas Data called 'data'
charges = data['charges']

# Create the line plot
plt.plot(charges)

# Set the title and labels for the axes
plt.title("Insurance Charges")
plt.xlabel("Index")
plt.ylabel("Charges")

# Display the plot
plt.show()
```



Heat Map

In [10]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have stored your dataset in a pandas Data called 'data'
# Select the columns you want to create the heatmap for
columns = ['age', 'bmi', 'children', 'charges']

# Create a correlation matrix for the selected columns
correlation_matrix = data[columns].corr()

# Create the heatmap plot
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')

# Set the title
plt.title("Correlation Heatmap")

# Display the plot
plt.show()
```



In [11]:
```
data
```

Out[11]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

Label Encoder

In [12]:
```
from sklearn.preprocessing import LabelEncoder

# Select the categorical columns to be encoded
categorical_columns = ['sex', 'smoker', 'region']
```

```
# Create an instance of LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding to the categorical columns
for column in categorical_columns:
    data[column] = label_encoder.fit_transform(data[column])

# Print the updated dataset
print(data)
```

```
      age  sex     bmi  children  smoker  region      charges
0      19    0  27.900         0       1       3  16884.92400
1      18    1  33.770         1       0       2   1725.55230
2      28    1  33.000         3       0       2   4449.46200
3      33    1  22.705         0       0       1  21984.47061
4      32    1  28.880         0       0       1   3866.85520
...   ...  ...     ...       ...     ...     ...          ...
1333   50    1  30.970         3       0       1  10600.54830
1334   18    0  31.920         0       0       0   2205.98080
1335   18    0  36.850         0       0       2   1629.83350
1336   21    0  25.800         0       0       3   2007.94500
1337   61    0  29.070         0       1       1  29141.36030

[1338 rows x 7 columns]
```

Threshold

In [13]:
```
X = data.drop('charges',axis=1)
Y = data['charges']
```

In [14]:
```
threshold = 0.1

selector = VarianceThreshold(threshold)
```

In [15]:
```
X_selected = selector.fit_transform(X)
```

In [16]:
```
selected_indices = selector.get_support(indices=True)
```

In [17]:
```
selected_features = X.columns[selected_indices]
```

In [18]:
```
print(selected_features)
```

```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region'], dtype='object')
```

In [19]:
```
num_selected_features = X_selected.shape[1]
```

In [20]:
```
print("Number of selected features:", num_selected_features)
```

```
Number of selected features: 6
```

Feature extraction

In [21]:
```
import pandas as pd
from sklearn.decomposition import PCA
```

```
from sklearn.preprocessing import StandardScaler



X = data.drop(columns=['children'])


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)


pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])


print("Explained Variance Ratio:")
for i, explained_variance in enumerate(pca.explained_variance_ratio_):
    print(f"PC{i+1}: {explained_variance:.4f}")


print("Transformed Data with PCA:")
print(pca_df.head())
```

```
Explained Variance Ratio:
PC1: 0.3131
PC2: 0.1965
Transformed Data with PCA:
        PC1       PC2
0  0.945238  0.632477
1 -1.134008 -0.269325
2 -0.828877 -0.423260
3 -0.097407  1.180830
4 -0.957931  0.450384
```

Linear Regression

In [22]:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


X = data.drop(columns=['children'])
y = data['sex']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_


model = LinearRegression()


model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)


mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 7.360811459320852e-24
```

In [23]:
```
print("Accrurancy= ", model.score(X_test, y_test))
```

```
Accrurancy=  1.0
```

In [24]:
```
print("Accrurancy= ", model.score(X_test, y_pred))
```

```
Accrurancy=  1.0
```

Random forest

In [25]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

In [26]:
```
# Convert categorical variables to numerical using one-hot encoding
df_encoded = pd.get_dummies(data)

# Split the dataset into input features (X) and target variable (y)
X = df_encoded.drop('charges', axis=1)
y = df_encoded['charges']

# 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_
```

In [27]:
```
# Initialize the random forest regressor
rf_regressor = RandomForestRegressor()

# Fit the model to the training data
rf_regressor.fit(X_train, y_train)
```

Out[27]:
```
▾ RandomForestRegressor

RandomForestRegressor()
```

In [28]:
```
# Generate predictions on the test set
y_pred = rf_regressor.predict(X_test)
```

In [29]:
```
# Calculate metrics such as mean squared error, mean absolute error, and R-squar
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("R-squared:", r2)
```

```
Mean Squared Error: 21237093.91323657
Mean Absolute Error: 2490.1732987646774
R-squared: 0.8632060068435283
```

Decision tree regression

In [30]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

In [31]:
```
# Preprocess the data
# Convert categorical variables to numerical using one-hot encoding
data = pd.get_dummies(data, drop_first=True)
```

In [32]:
```
# Split the data into training and testing sets
X = data.drop("charges", axis=1)
y = data["charges"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [33]:
```
# Create and train the decision tree regressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)
```

Out[33]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

In [34]:
```
# Predict on the test set
y_pred = regressor.predict(X_test)
```

In [35]:
```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

In [36]:
```
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 49003243.60682007
R-squared: 0.6843565603663775
```

In [37]:
```
data
```

Out[37]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	3	16884.92400
1	18	1	33.770	1	0	2	1725.55230
2	28	1	33.000	3	0	2	4449.46200
3	33	1	22.705	0	0	1	21984.47061
4	32	1	28.880	0	0	1	3866.85520
...	...	...	...	...	...	...	...
1333	50	1	30.970	3	0	1	10600.54830
1334	18	0	31.920	0	0	0	2205.98080
1335	18	0	36.850	0	0	2	1629.83350
1336	21	0	25.800	0	0	3	2007.94500
1337	61	0	29.070	0	1	1	29141.36030

1338 rows × 7 columns

Support Vector Machine

In [38]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
```

In [39]:
```
# Preprocess the data
# Convert categorical variables to numerical using one-hot encoding
data = pd.get_dummies(data, drop_first=True)
```

In [40]:
```
# Split the data into training and testing sets
X = data.drop("charges", axis=1)
y = data["charges"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [41]:
```
# Create and train the Support Vector Machine regressor
regressor = SVR(kernel='linear')
regressor.fit(X_train, y_train)
```

Out[41]:

▾ SVR

```
SVR(kernel='linear')
```

In [42]:
```
# Predict on the test set
y_pred = regressor.predict(X_test)
```

In [43]:
```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

In [44]:
```
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 165910580.55836943
R-squared: -0.068675917440014
```

In [45]:
```
data
```

Out[45]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	3	16884.92400
1	18	1	33.770	1	0	2	1725.55230
2	28	1	33.000	3	0	2	4449.46200
3	33	1	22.705	0	0	1	21984.47061
4	32	1	28.880	0	0	1	3866.85520
...	...	...	...	...	...	...	...
1333	50	1	30.970	3	0	1	10600.54830
1334	18	0	31.920	0	0	0	2205.98080
1335	18	0	36.850	0	0	2	1629.83350
1336	21	0	25.800	0	0	3	2007.94500
1337	61	0	29.070	0	1	1	29141.36030

1338 rows × 7 columns

In [46]:
```
data
```

Out[46]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	3	16884.92400
1	18	1	33.770	1	0	2	1725.55230
2	28	1	33.000	3	0	2	4449.46200
3	33	1	22.705	0	0	1	21984.47061
4	32	1	28.880	0	0	1	3866.85520
...	...	...	...	...	...	...	...
1333	50	1	30.970	3	0	1	10600.54830
1334	18	0	31.920	0	0	0	2205.98080
1335	18	0	36.850	0	0	2	1629.83350
1336	21	0	25.800	0	0	3	2007.94500
1337	61	0	29.070	0	1	1	29141.36030

1338 rows × 7 columns

Normalisation

In [47]:
```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
```

In [48]:
```
# Create a copy of the original dataset
normalized_dataset = data.copy()
```

In [49]:
```
# Select the numerical columns to be normalized
numerical_columns = ['age', 'bmi', 'children', 'charges']
```

In [50]:
```
# Initialize the MinMaxScaler
scaler = MinMaxScaler()
```

In [51]:
```
# Normalize the selected columns
normalized_dataset[numerical_columns] = scaler.fit_transform(normalized_dataset[
```

In [52]:
```
data = pd.DataFrame(normalized_dataset, columns=numerical_columns)
```

In [53]:
```
# Display the normalized dataset
print(normalized_dataset)
```

```
           age  sex       bmi  children  smoker  region   charges
0     0.021739    0  0.321227       0.0       1       3  0.251611
1     0.000000    1  0.479150       0.2       0       2  0.009636
2     0.217391    1  0.458434       0.6       0       2  0.053115
3     0.326087    1  0.181464       0.0       0       1  0.333010
4     0.304348    1  0.347592       0.0       0       1  0.043816
...        ...  ...       ...       ...     ...     ...       ...
1333  0.695652    1  0.403820       0.6       0       1  0.151299
1334  0.000000    0  0.429379       0.0       0       0  0.017305
1335  0.000000    0  0.562012       0.0       0       2  0.008108
1336  0.065217    0  0.264730       0.0       0       3  0.014144
1337  0.934783    0  0.352704       0.0       1       1  0.447249

[1338 rows x 7 columns]
```

PCA (Principal Component Analysis)

In [54]:
```
from sklearn.decomposition import PCA
import pandas as pd

# Create a copy of the original dataset
pca_dataset = data.copy()

# Select the columns to be used for PCA
feature_columns = ['age', 'bmi', 'children', 'charges']

# Extract the features
features = pca_dataset[feature_columns]

# Initialize PCA with the desired number of components
n_components = 2  # Number of components to keep
pca = PCA(n_components=n_components)

# Apply PCA on the features
pca_result = pca.fit_transform(features)
```

```
# Create a new DataFrame to store the PCA results
pca_columns = ['PC{}'.format(i+1) for i in range(n_components)]
pca_df = pd.DataFrame(data=pca_result, columns=pca_columns)

# Concatenate the PCA results with the original dataset
pca_dataset = pd.concat([pca_dataset, pca_df], axis=1)

# Display the dataset with the PCA results
print(pca_dataset)
```

```
           age       bmi  children   charges       PC1       PC2
0     0.021739  0.321227       0.0  0.251611 -0.428454 -0.163410
1     0.000000  0.479150       0.2  0.009636 -0.483649  0.025202
2     0.217391  0.458434       0.6  0.053115 -0.229048  0.398845
3     0.326087  0.181464       0.0  0.333010 -0.130012 -0.195263
4     0.304348  0.347592       0.0  0.043816 -0.216810 -0.207529
...        ...       ...       ...       ...       ...       ...
1333  0.695652  0.403820       0.6  0.151299  0.247476  0.348271
1334  0.000000  0.429379       0.0  0.017305 -0.505258 -0.173047
1335  0.000000  0.562012       0.0  0.008108 -0.495013 -0.172655
1336  0.065217  0.264730       0.0  0.014144 -0.460230 -0.181918
1337  0.934783  0.352704       0.0  0.447249  0.496767 -0.258591

[1338 rows x 6 columns]
```

Root mean square error

In [55]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
```

In [56]:
```
# Split the dataset into features (X) and target variable (y)
X = pca_dataset[['PC1', 'PC2']]
y = pca_dataset['charges']
```

In [57]:
```
# 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_
```

In [58]:
```
# Linear Regression
linear_regression = LinearRegression()
linear_regression.fit(X_train, y_train)
linear_regression_predictions = linear_regression.predict(X_test)
linear_regression_rmse = mean_squared_error(y_test, linear_regression_predictior
print("Linear Regression RMSE:", linear_regression_rmse)
```

```
Linear Regression RMSE: 0.16945580371555413
```

In [59]:
```
# Random Forest Regression
random_forest = RandomForestRegressor()
random_forest.fit(X_train, y_train)
random_forest_predictions = random_forest.predict(X_test)
random_forest_rmse = mean_squared_error(y_test, random_forest_predictions, squar
print("Random Forest RMSE:", random_forest_rmse)
```

```
Random Forest RMSE: 0.04686987191810539
```

In [60]:
```
# Decision Tree Regression
decision_tree = DecisionTreeRegressor()
decision_tree.fit(X_train, y_train)
decision_tree_predictions = decision_tree.predict(X_test)
decision_tree_rmse = mean_squared_error(y_test, decision_tree_predictions, squar
print("Decision Tree RMSE:", decision_tree_rmse)
```

```
Decision Tree RMSE: 0.06373651245671731
```

In [61]:
```
# Support Vector Machine (SVM)
svm = SVR()
svm.fit(X_train, y_train)
svm_predictions = svm.predict(X_test)
svm_rmse = mean_squared_error(y_test, svm_predictions, squared=False)
print("SVM RMSE:", svm_rmse)
```

```
SVM RMSE: 0.16451226970176322
```

Mean Squared Error

In [62]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
```

In [63]:
```
# Split the dataset into features (X) and target variable (y)
X = pca_dataset[['PC1', 'PC2']]
y = pca_dataset['charges']
```

In [64]:
```
# 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_
```

In [65]:
```
# Linear Regression
linear_regression = LinearRegression()
linear_regression.fit(X_train, y_train)
linear_regression_predictions = linear_regression.predict(X_test)
linear_regression_mse = mean_squared_error(y_test, linear_regression_predictions
print("Linear Regression MSE:", linear_regression_mse)
```

```
Linear Regression MSE: 0.02871526941288441
```

In [66]:
```
# Random Forest Regression
random_forest = RandomForestRegressor()
random_forest.fit(X_train, y_train)
random_forest_predictions = random_forest.predict(X_test)
random_forest_mse = mean_squared_error(y_test, random_forest_predictions)
print("Random Forest MSE:", random_forest_mse)
```

```
Random Forest MSE: 0.0020526468538579344
```

In [67]:
```
# Decision Tree Regression
decision_tree = DecisionTreeRegressor()
decision_tree.fit(X_train, y_train)
decision_tree_predictions = decision_tree.predict(X_test)
decision_tree_mse = mean_squared_error(y_test, decision_tree_predictions)
print("Decision Tree MSE:", decision_tree_mse)
```

```
Decision Tree MSE: 0.004242275994029003
```

In [68]:
```python
# Support Vector Machine (SVM)
svm = SVR()
svm.fit(X_train, y_train)
svm_predictions = svm.predict(X_test)
svm_mse = mean_squared_error(y_test, svm_predictions)
print("SVM MSE:", svm_mse)
```

```
SVM MSE: 0.027064286882425678
```