Regression Modeling

June 29, 2023

[1]:
```
#importing necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder #for feature engineering
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error # for␣
↪evaluating ml models
```

[2]:
```
#loading the dataset
df=pd.read_csv("/kaggle/input/data-visualizatiion/insurance.csv")
df
```

[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
…	…	…	…	…	…	…	…
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 x 7 columns]

[3]:
```
#first 5 rows
df.head()
```

[3]:

	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
```

[4]:
```
#last 5 rows
df.tail()
```

[4]:
```
      age     sex    bmi  children smoker     region      charges
1333   50    male  30.97         3     no  northwest  10600.5483
1334   18  female  31.92         0     no  northeast   2205.9808
1335   18  female  36.85         0     no  southeast   1629.8335
1336   21  female  25.80         0     no  southwest   2007.9450
1337   61  female  29.07         0    yes  northwest  29141.3603
```

[5]:
```
# checking the data types and null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

[6]:
```
# checking unique values
df.nunique()
```

[6]:
```
age           47
sex            2
bmi          548
children       6
smoker         2
region         4
charges     1337
dtype: int64
```

[7]:
```
# checking the columns
df.columns
```

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

[8]:
```
# checking duplicate values
df.duplicated().value_counts()
```

[8]:
```
False    1337
True        1
dtype: int64
```

[9]:
```
# droping duplicate values
df.drop_duplicates(inplace=True)
```

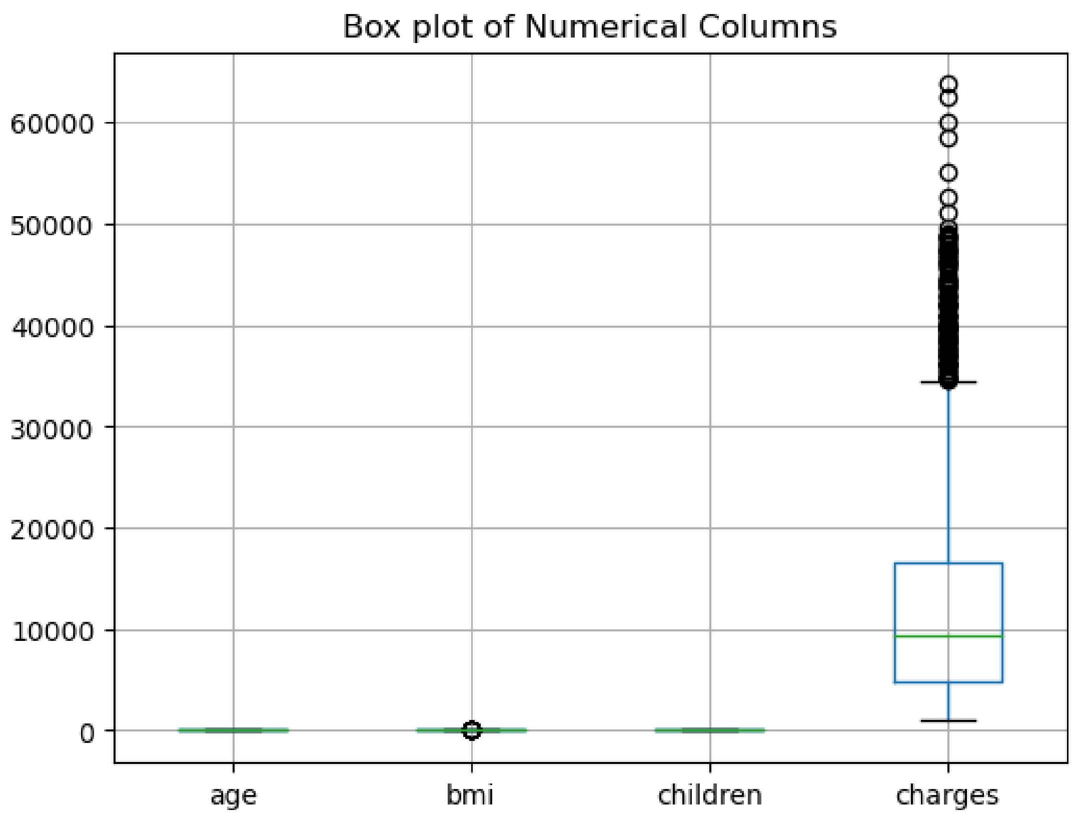
[10]:
```
# rows, columns
df.shape
```

[10]: (1337, 7)

[11]:
```
# rows * columns
df.size
```

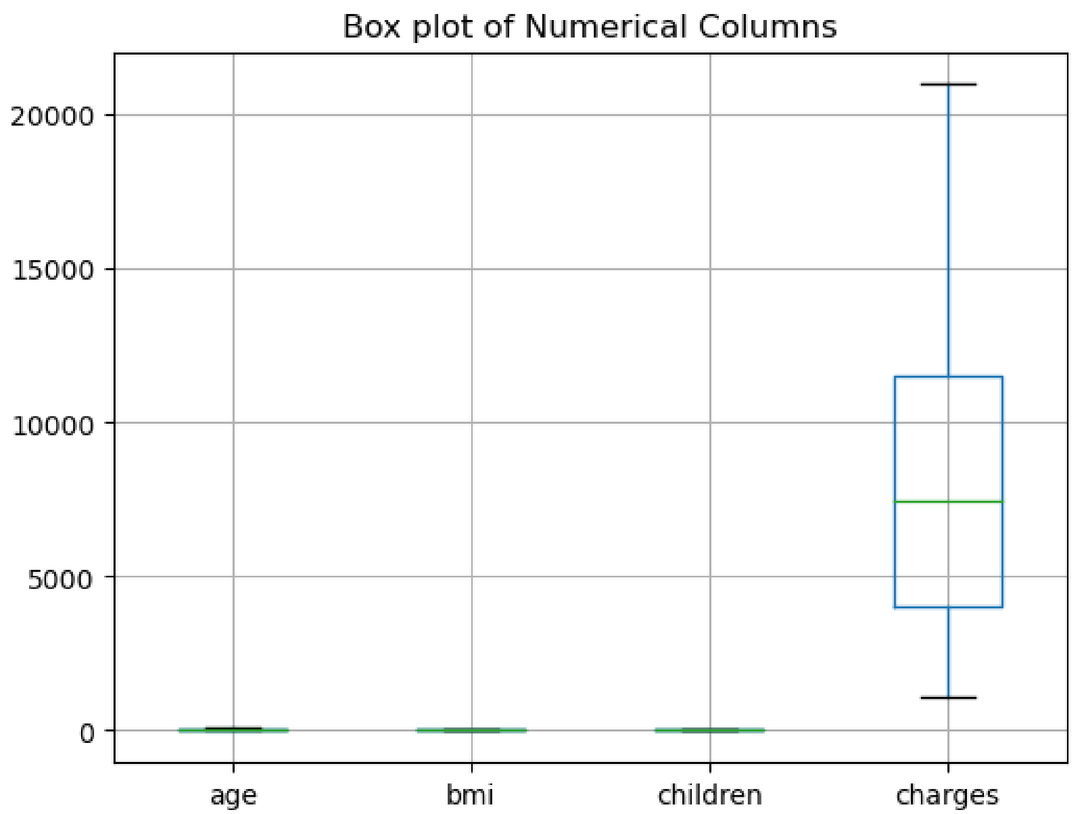
[11]: 9359

[12]:
```
# Check for outliers
# Visualize box plots for numerical columns
df.boxplot(column=['age', 'bmi', 'children', 'charges'])
plt.title('Box plot of Numerical Columns')
plt.show()
```



[13]:
```
# Handling outliers
# Remove outliers based on a specific threshold
df_cleaned = df[(df['charges'] < 21000) & (df['bmi'] < 46)]
```

[14]:
```
# visualize boxplot after removing outliers
df_cleaned.boxplot(column=['age', 'bmi', 'children', 'charges'])
plt.title('Box plot of Numerical Columns')
plt.show()
```



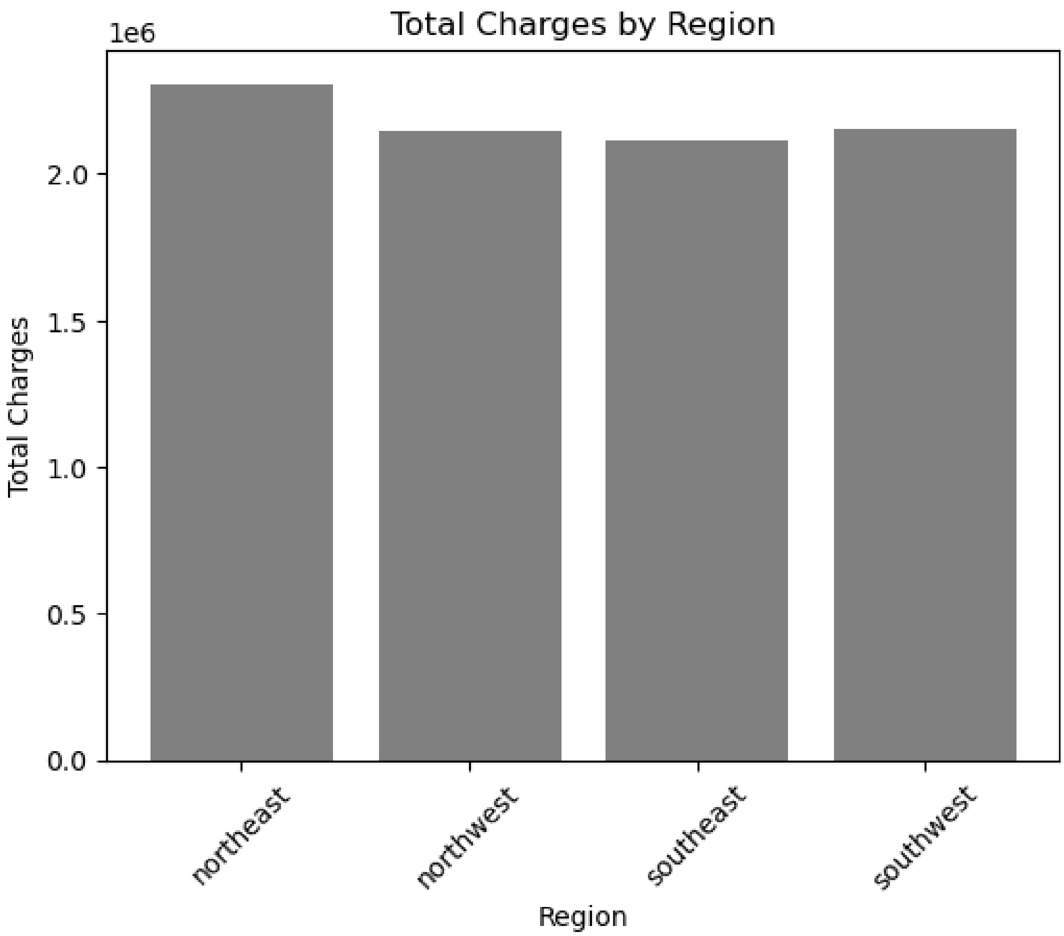
[15]:
```
# Get summary statistics
df_cleaned.describe()
```

[15]:

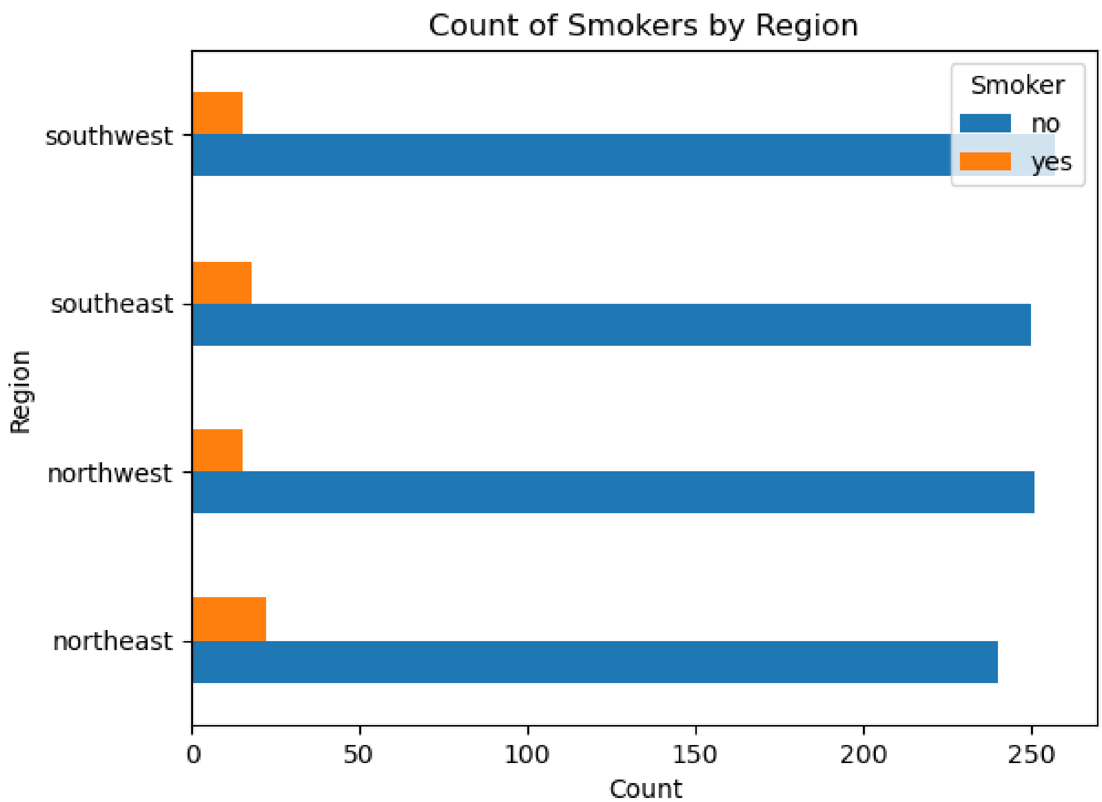
	age	bmi	children	charges
count	1068.000000	1068.000000	1068.000000	1068.000000
mean	38.371723	30.091910	1.076779	8159.150438
std	13.943164	5.856569	1.221375	4929.643963
min	18.000000	15.960000	0.000000	1121.873900
25%	26.000000	25.840000	0.000000	4038.478863
50%	38.000000	29.830000	1.000000	7441.277000
75%	50.000000	33.933750	2.000000	11539.380487
max	64.000000	45.900000	5.000000	20984.093600

[16]:
```
# Calculate the total charges by region
charges_by_region = df_cleaned.groupby('region')['charges'].sum()

# Create a bar chart
plt.bar(charges_by_region.index, charges_by_region.values, color='grey')

# Add labels and title
plt.xlabel('Region')
```

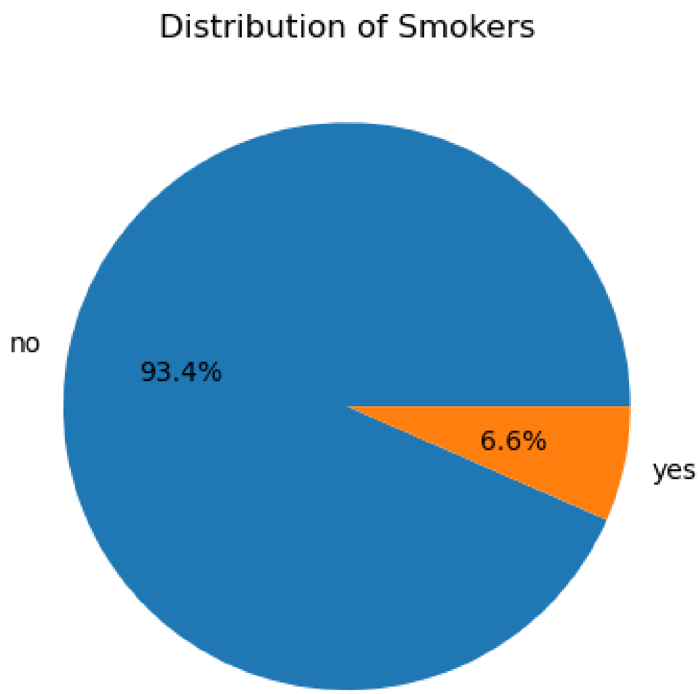
```
plt.ylabel('Total Charges')
plt.title('Total Charges by Region')

# Rotate x-axis labels for better visibility (optional)
plt.xticks(rotation=45)

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



[17]:
```
smokers_by_region = df_cleaned.groupby('region')['smoker'].value_counts().
 ↪unstack().fillna(0)
smokers_by_region.plot(kind='barh')
plt.xlabel('Count')
plt.ylabel('Region')
plt.title('Count of Smokers by Region')
plt.legend(title='Smoker', loc='upper right')
plt.show()
```

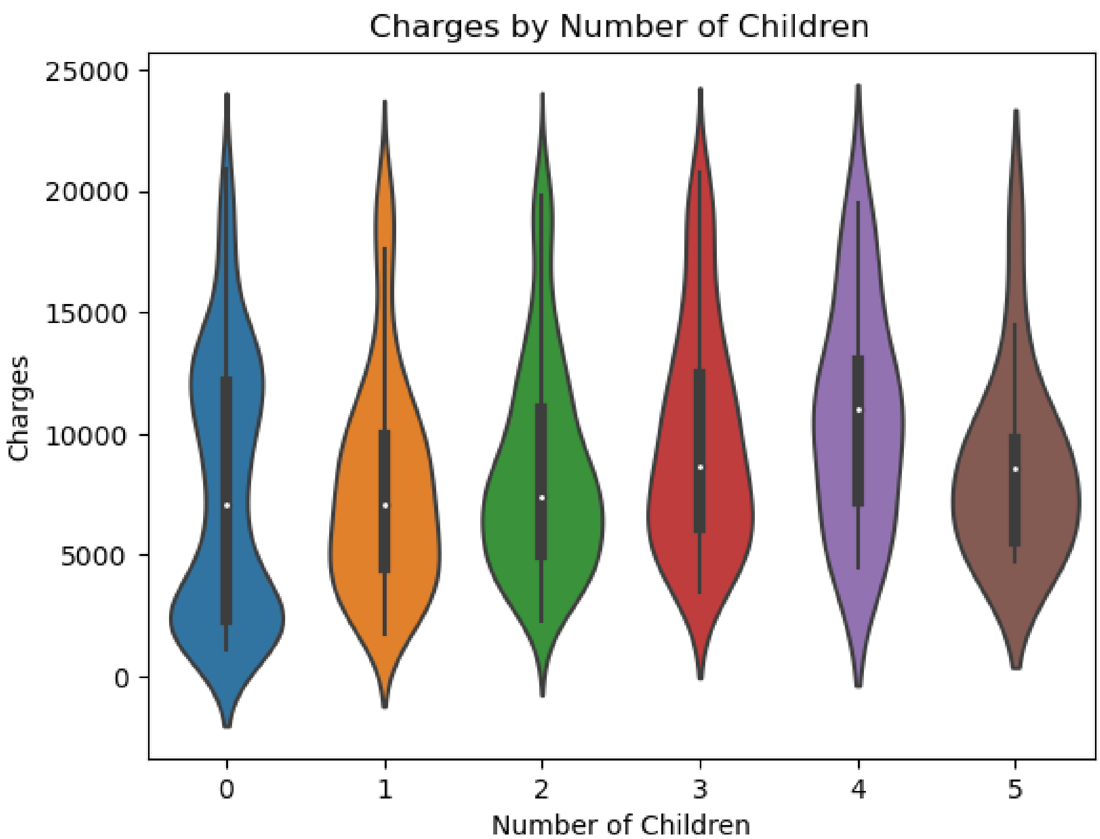


[18]:
```
smoker_counts = df_cleaned['smoker'].value_counts()
plt.pie(smoker_counts, labels=smoker_counts.index, autopct='%1.1f%%')
plt.title('Distribution of Smokers')
plt.show()
```

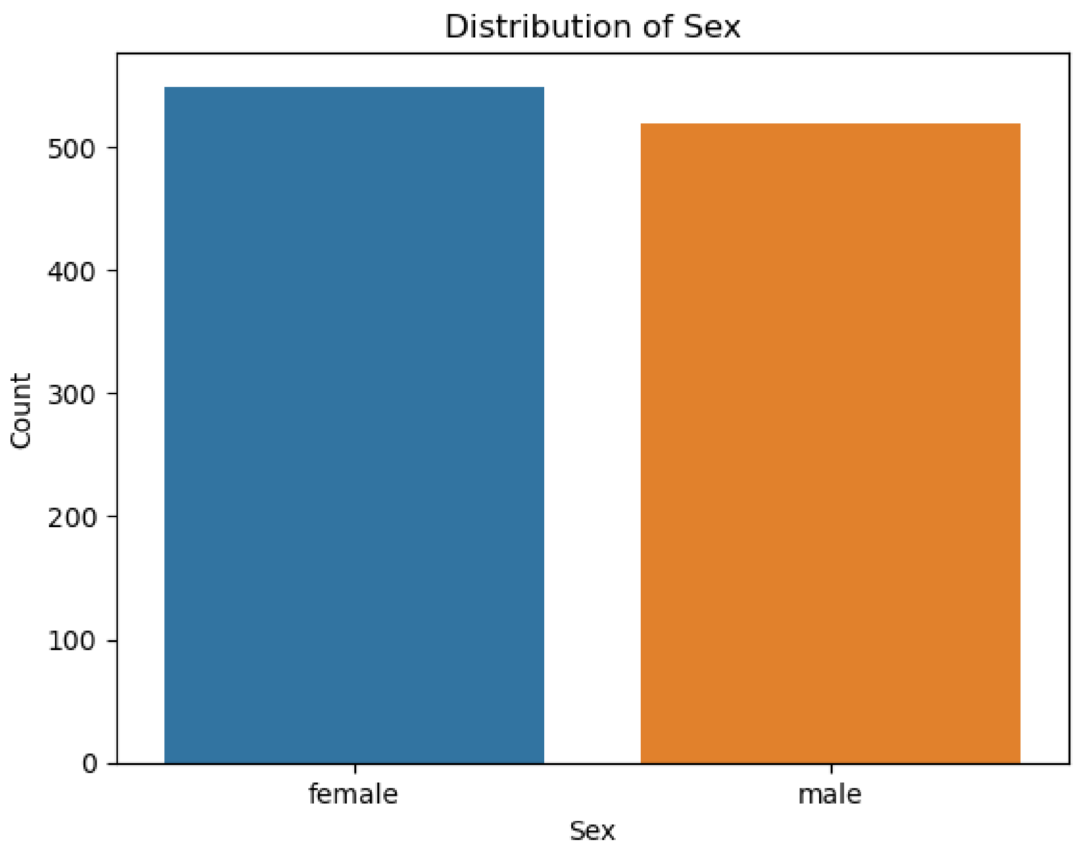


[19]:

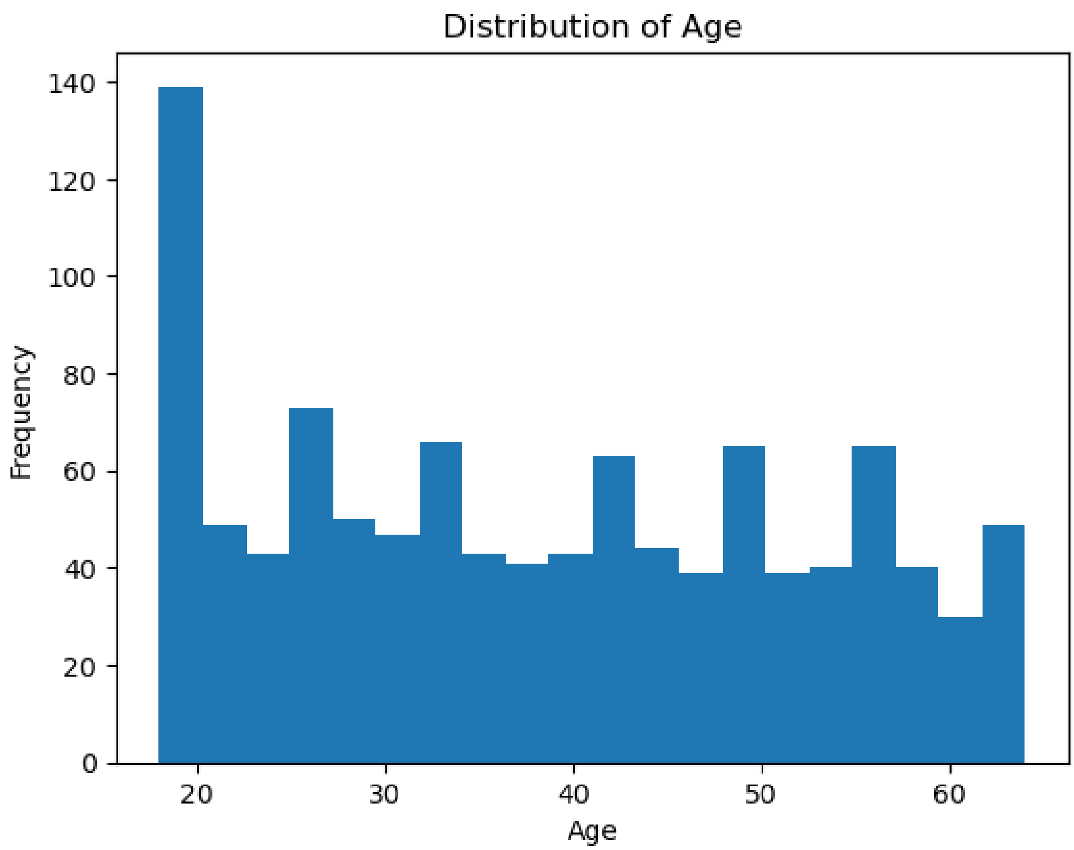
```
sns.violinplot(x=df_cleaned['children'], y=df_cleaned['charges'])
plt.xlabel('Number of Children')
plt.ylabel('Charges')
plt.title('Charges by Number of Children')
plt.show()
```



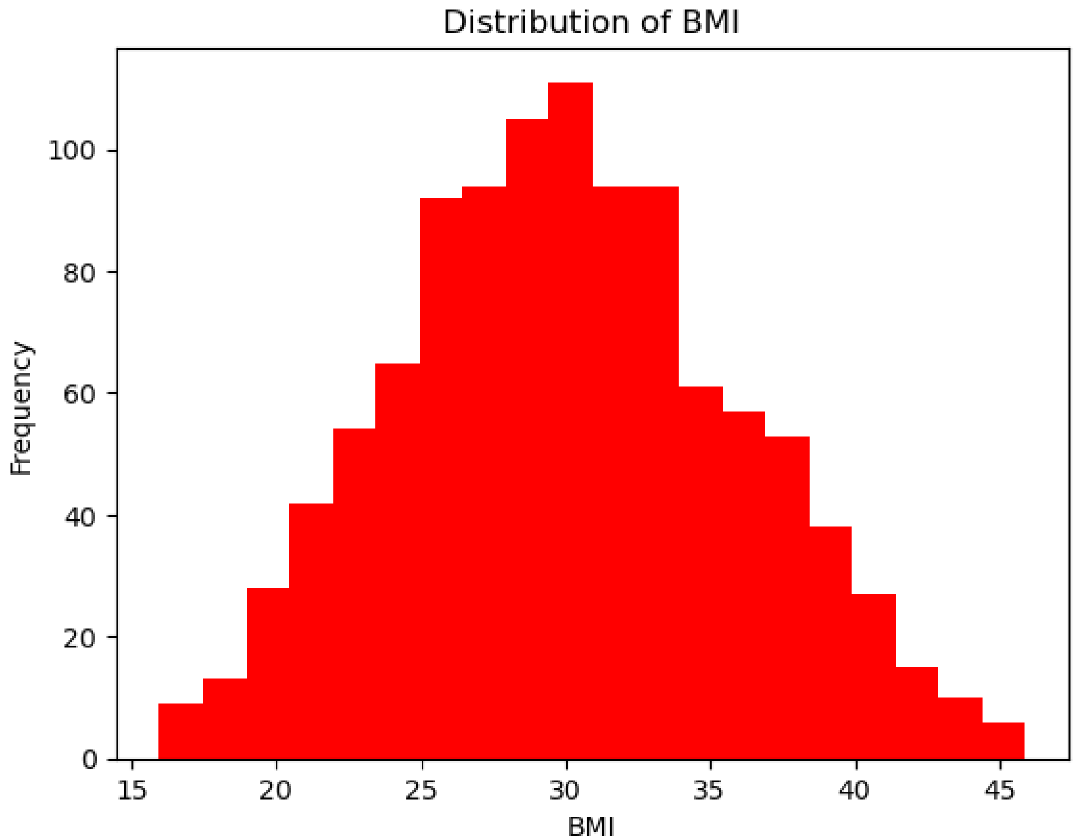
[20]:
```
sns.countplot(x='sex', data=df_cleaned)
plt.xlabel('Sex')
plt.ylabel('Count')
plt.title('Distribution of Sex')
plt.show()
```



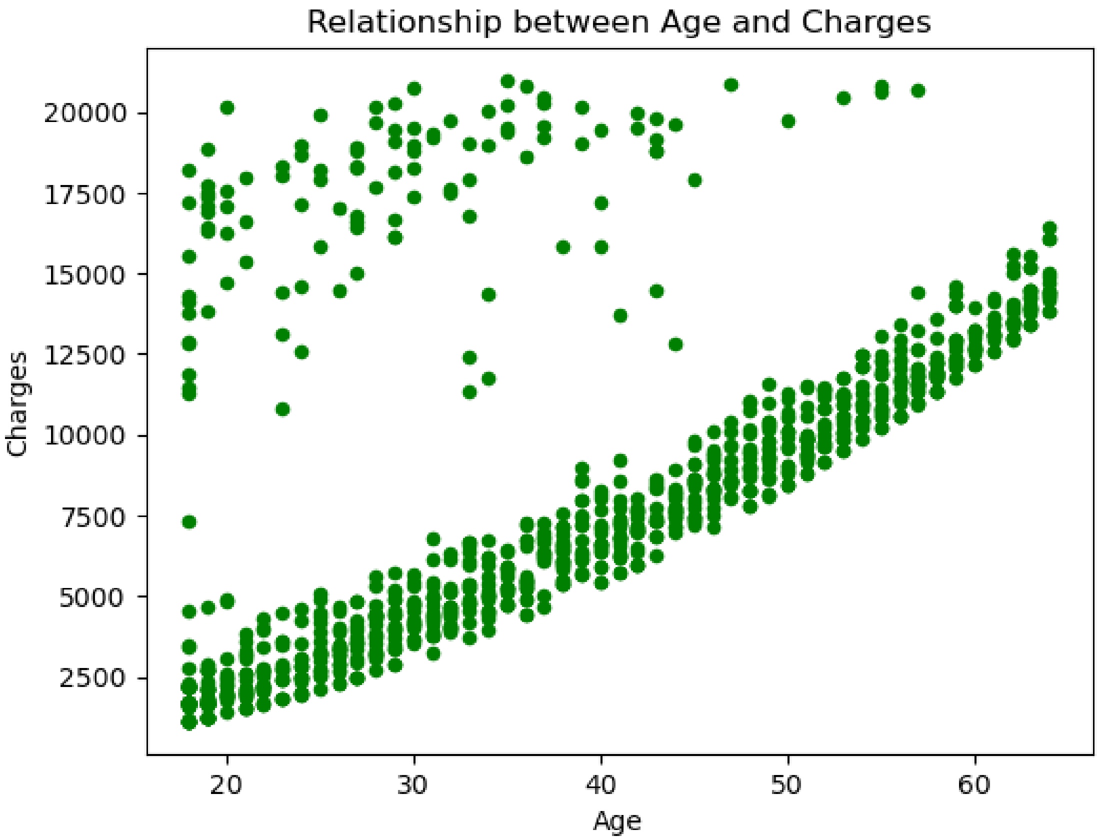
[21]:
```
# distribution of age using histogram
df_cleaned['age'].plot(kind='hist', bins=20)
plt.xlabel('Age')
plt.title('Distribution of Age')
plt.show()
```



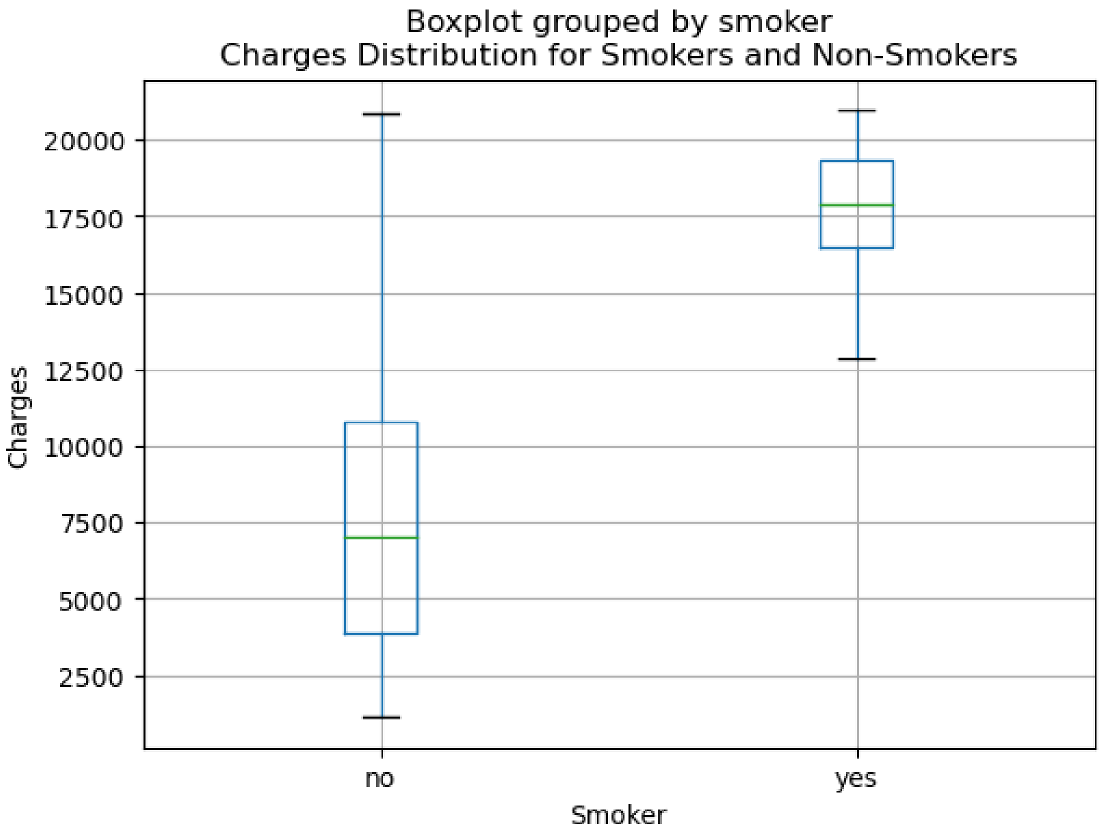
[22]:
```
# distribution of BMI using histogram
df_cleaned['bmi'].plot(kind='hist', bins=20, color='r')
plt.xlabel('BMI')
plt.title('Distribution of BMI')
plt.show()
```



[23]:
```
# relationship between age and charges using scatterplot
df_cleaned.plot(kind='scatter', x='age', y='charges', color='g')
plt.xlabel('Age')
plt.ylabel('Charges')
plt.title('Relationship between Age and Charges')
plt.show()
```



[24]:
```
# charges distribution for smokers vs non-smokers using boxplot
df_cleaned.boxplot(column='charges', by='smoker')
plt.xlabel('Smoker')
plt.ylabel('Charges')
plt.title('Charges Distribution for Smokers and Non-Smokers')
plt.show()
```



[25]:
```
# Feature engineering - Creating new feature such as Age groups
df_cleaned['age_group'] = pd.cut(df_cleaned['age'], bins=[0, 25, 40, 60,␣
↪df_cleaned['age'].max()], labels=['Young', 'Adult', 'Middle-aged', 'Senior'])
```

```
/tmp/ipykernel_479/1296161964.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['age_group'] = pd.cut(df_cleaned['age'], bins=[0, 25, 40, 60,
df_cleaned['age'].max()], labels=['Young', 'Adult', 'Middle-aged', 'Senior'])
```

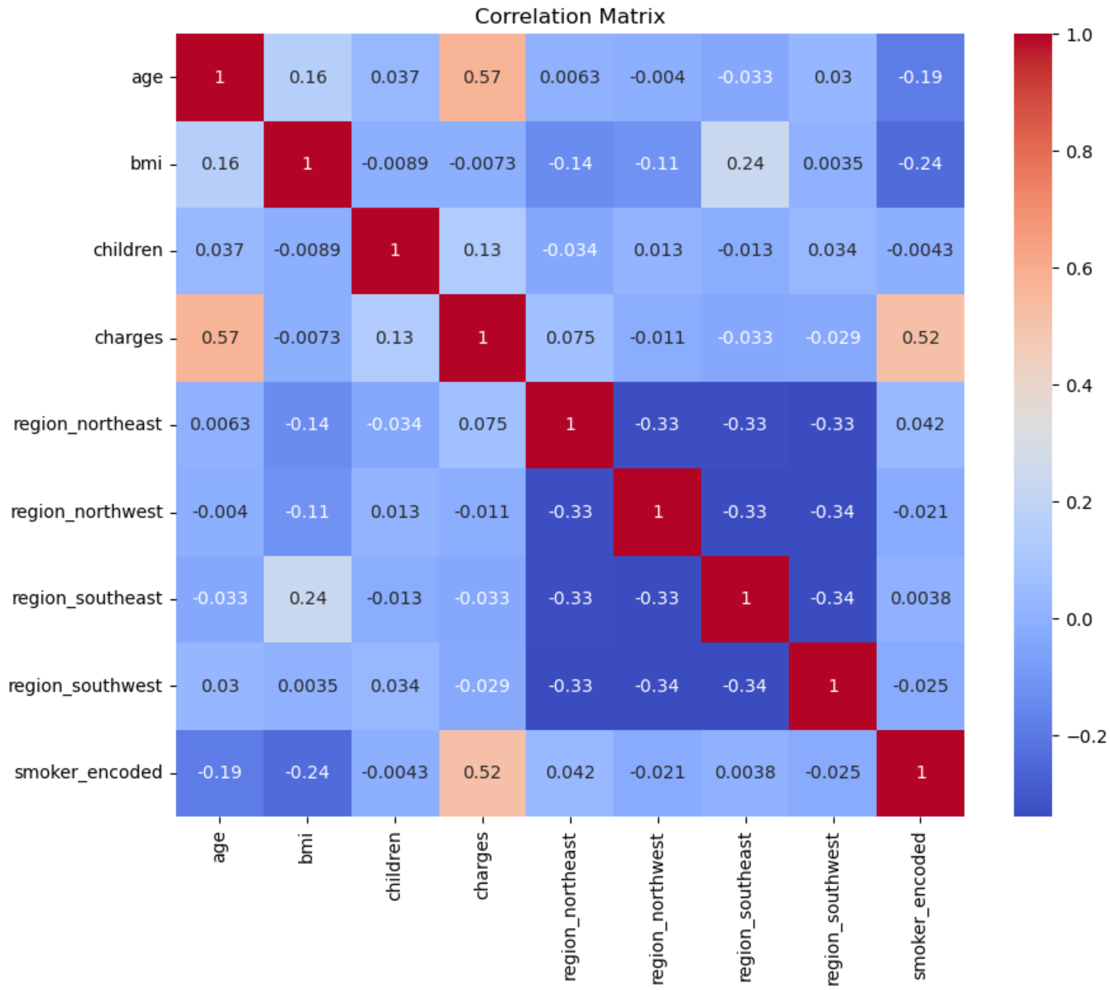
[26]:
```
# Feature encoding - Handling categorical variables such as One-hot encoding␣
↪for the 'region' column
df_encoded = pd.get_dummies(df_cleaned, columns=['region'], prefix='region')

# Label encoding for the 'smoker' column
label_encoder = LabelEncoder()
df_encoded['smoker_encoded'] = label_encoder.fit_transform(df_encoded['smoker'])
```

[27]:
```
# Confirm the changes
df_encoded.head()
```

[27]:
```
   age     sex    bmi  children smoker      charges age_group  \
0   19  female  27.90         0    yes  16884.9240     Young
1   18    male  33.77         1     no   1725.5523     Young
2   28    male  33.00         3     no   4449.4620     Adult
4   32    male  28.88         0     no   3866.8552     Adult
5   31  female  25.74         0     no   3756.6216     Adult

   region_northeast  region_northwest  region_southeast  region_southwest  \
0                 0                 0                 0                 1
1                 0                 0                 1                 0
2                 0                 0                 1                 0
4                 0                 1                 0                 0
5                 0                 0                 1                 0

   smoker_encoded
0               1
1               0
2               0
4               0
5               0
```

[28]:
```
# Perform correlation analysis
corr_matrix = df_encoded.corr()

# Visualize correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()

# Identify relevant features based on correlation
threshold = 0.3
relevant_features = corr_matrix[(corr_matrix['charges'].abs() > threshold) &
 ↪(corr_matrix.index != 'charges')].index.tolist()
print("Relevant features based on correlation:")
print(relevant_features)
```

```
/tmp/ipykernel_479/3840923490.py:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  corr_matrix = df_encoded.corr()
```



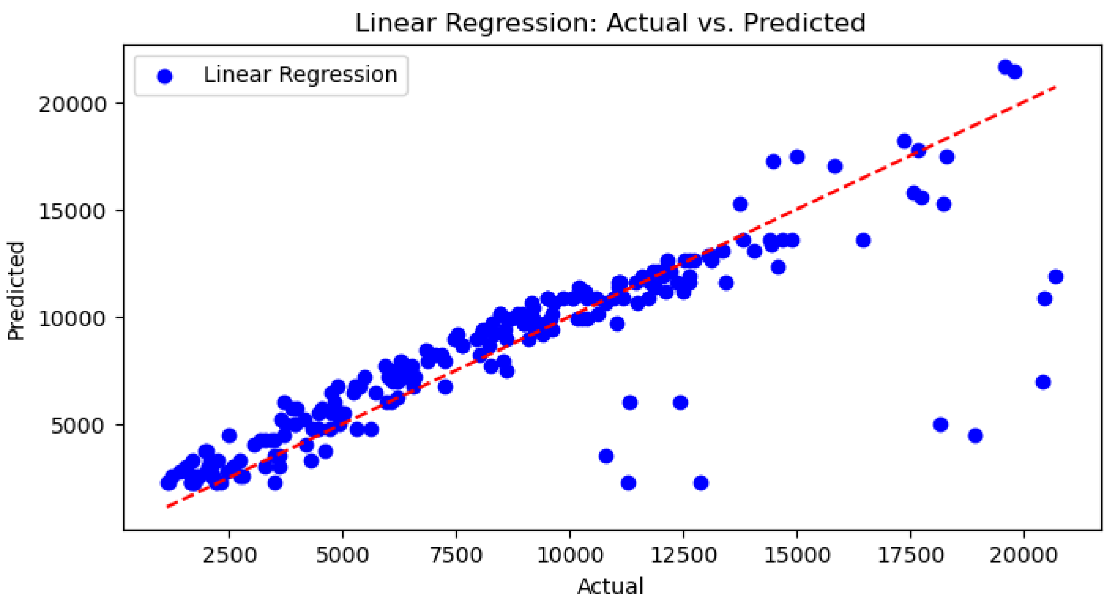

```
Relevant features based on correlation:
['age', 'smoker_encoded']
```

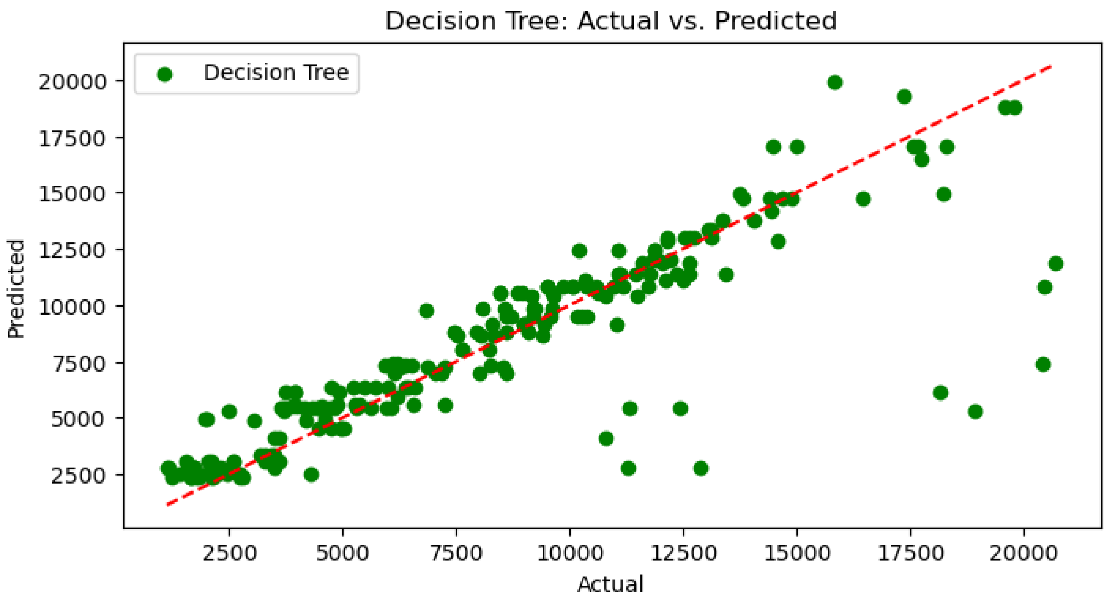
[29]:
```python
# Select the relevant features
X = df_encoded[['age', 'smoker_encoded']]
y = df_encoded['charges']

# 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_state=42)
```

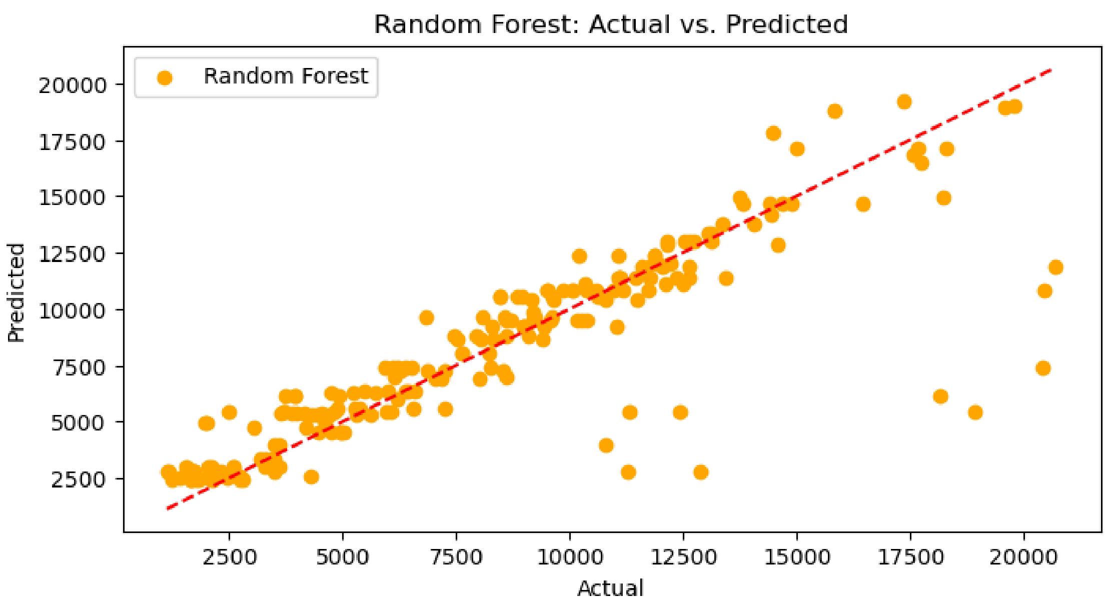
[30]:
```python
# Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_predictions = lr_model.predict(X_test)
lr_mse = mean_squared_error(y_test, lr_predictions)
lr_mae = mean_absolute_error(y_test, lr_predictions)
```

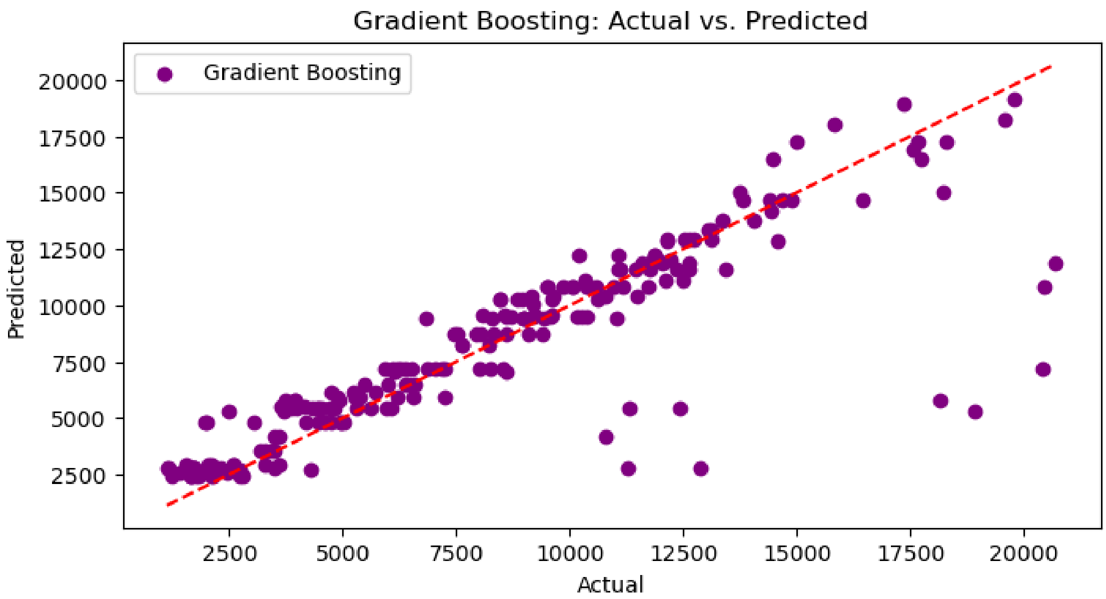
```python
# Plot actual vs. predicted values for Linear Regression
plt.figure(figsize=(8, 4))
plt.scatter(y_test, lr_predictions, color='blue', label='Linear Regression')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',␣
↪linestyle='--')
plt.title('Linear Regression: Actual vs. Predicted')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.legend()
plt.show()

# Decision Tree
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)
dt_predictions = dt_model.predict(X_test)
dt_mse = mean_squared_error(y_test, dt_predictions)
dt_mae = mean_absolute_error(y_test, dt_predictions)

# Plot actual vs. predicted values for Decision Tree
plt.figure(figsize=(8, 4))
plt.scatter(y_test, dt_predictions, color='green', label='Decision Tree')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',␣
↪linestyle='--')
plt.title('Decision Tree: Actual vs. Predicted')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.legend()
plt.show()

# Random Forest
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_mse = mean_squared_error(y_test, rf_predictions)
rf_mae = mean_absolute_error(y_test, rf_predictions)

# Plot actual vs. predicted values for Random Forest
plt.figure(figsize=(8, 4))
plt.scatter(y_test, rf_predictions, color='orange', label='Random Forest')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',␣
↪linestyle='--')
plt.title('Random Forest: Actual vs. Predicted')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.legend()
plt.show()
```

```python
# Gradient Boosting
gb_model = GradientBoostingRegressor()
gb_model.fit(X_train, y_train)
gb_predictions = gb_model.predict(X_test)
gb_mse = mean_squared_error(y_test, gb_predictions)
gb_mae = mean_absolute_error(y_test, gb_predictions)

# Plot actual vs. predicted values for Gradient Boosting
plt.figure(figsize=(8, 4))
plt.scatter(y_test, gb_predictions, color='purple', label='Gradient Boosting')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',␣
↪linestyle='--')
plt.title('Gradient Boosting: Actual vs. Predicted')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.legend()
plt.show()
```




Decision Tree: Actual vs. Predicted
Decision Tree
Predicted
Actual
20000
17500
15000
12500
10000
7500
5000
2500
2500
5000
7500
10000
12500
15000
17500
20000


Random Forest: Actual vs. Predicted
Random Forest
Predicted
Actual
20000
17500
15000
12500
10000
7500
5000
2500
2500
5000
7500
10000
12500
15000
17500
20000



[31]:

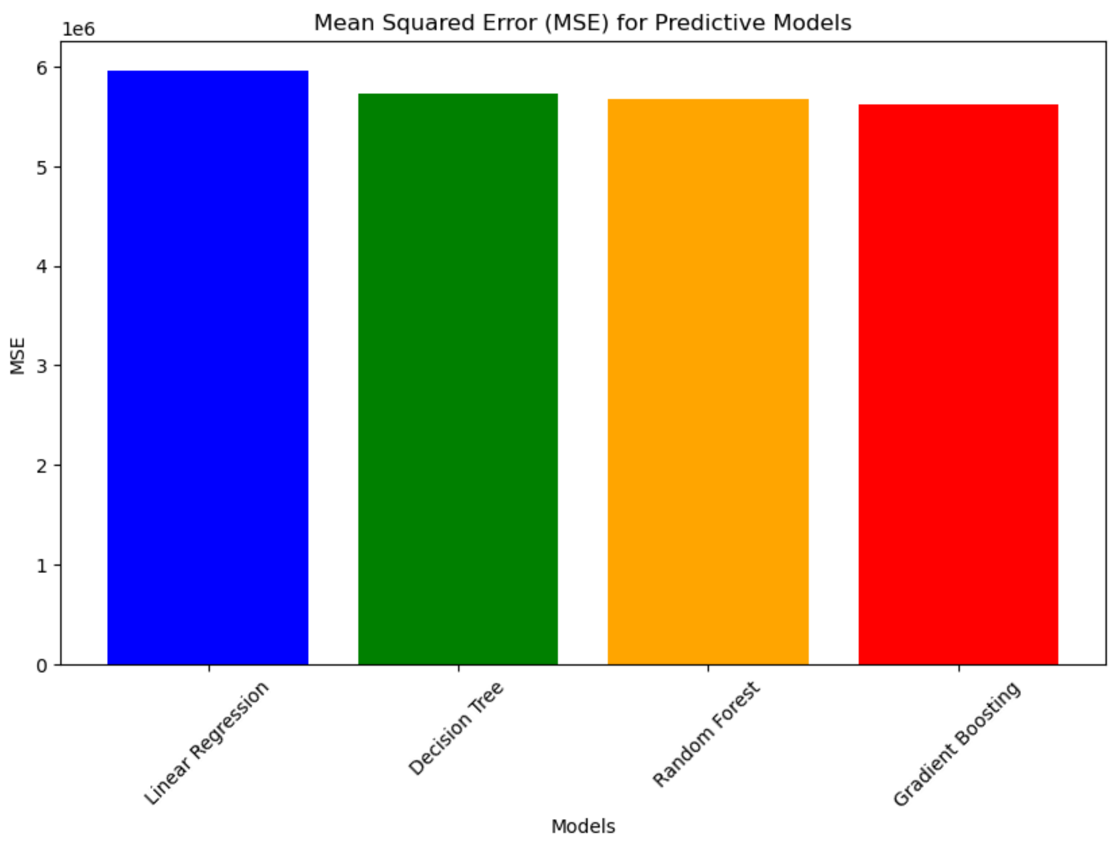
```
# Print the evaluation metrics
print("Linear Regression - MSE: ", lr_mse)
print("Linear Regression - MAE: ", lr_mae)
print("Decision Tree - MSE: ", dt_mse)
print("Decision Tree - MAE: ", dt_mae)
print("Random Forest - MSE: ", rf_mse)
print("Random Forest - MAE: ", rf_mae)
print("Gradient Boosting - MSE: ", gb_mse)
print("Gradient Boosting - MAE: ", gb_mae)
```

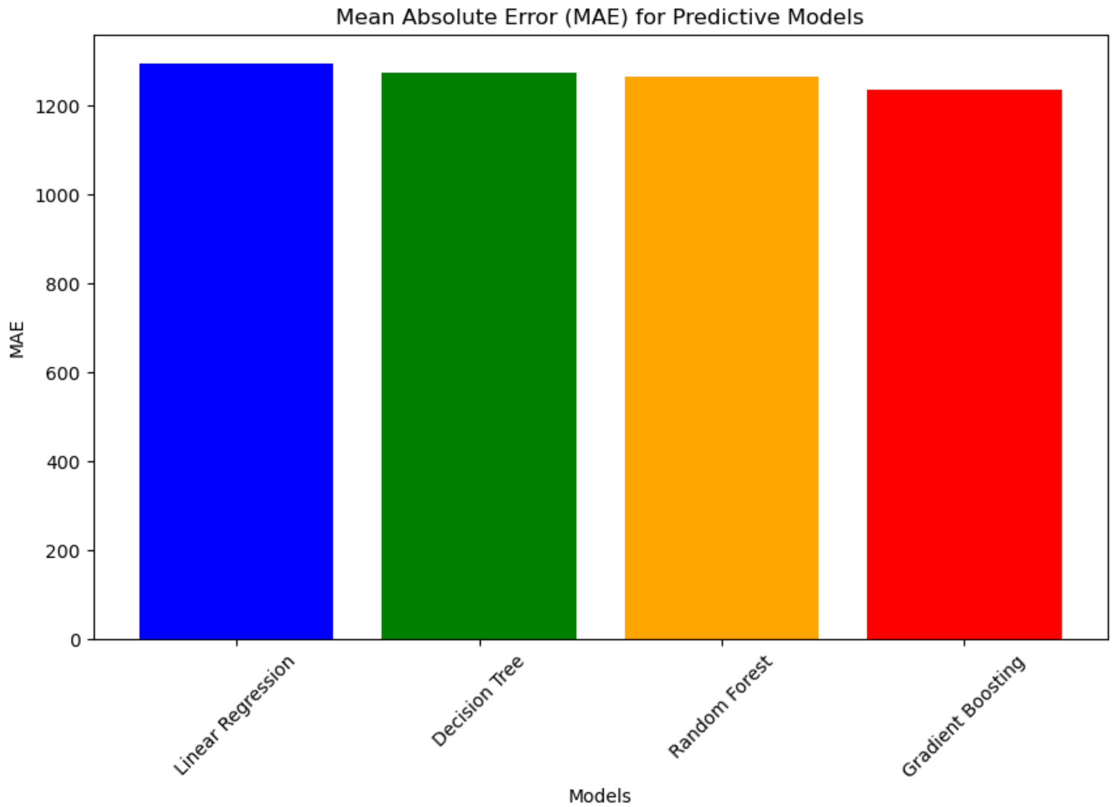
```
Linear Regression - MSE:  5958015.664548768
Linear Regression - MAE:  1294.7790805755785
Decision Tree - MSE:  5728632.311218643
Decision Tree - MAE:  1275.0243094744608
Random Forest - MSE:  5679776.920386332
Random Forest - MAE:  1258.9081506796574
Gradient Boosting - MSE:  5620075.226137549
Gradient Boosting - MAE:  1234.9739469039018
```

[32]:

```
# MSE values
mse_values = [5958015.664548768, 5728632.311218643, 5667626.334646116, 5620075.
↪226137549]
# MAE values
mae_values = [1294.7790805755785, 1275.0243094744608, 1266.1176489889306, 1234.
↪973946903902]

# Model names
```

```
models = ['Linear Regression', 'Decision Tree', 'Random Forest', 'Gradient␣
 ↪Boosting']

# Specify colors for the bars
colors = ['blue', 'green', 'orange', 'red']

# Plot MSE values
plt.figure(figsize=(10, 6))
plt.bar(models, mse_values, color=colors)
plt.title('Mean Squared Error (MSE) for Predictive Models')
plt.xlabel('Models')
plt.ylabel('MSE')
plt.xticks(rotation=45)
plt.show()

# Plot MAE values
plt.figure(figsize=(10, 6))
plt.bar(models, mae_values, color=colors)
plt.title('Mean Absolute Error (MAE) for Predictive Models')
plt.xlabel('Models')
plt.ylabel('MAE')
plt.xticks(rotation=45)
plt.show()
```





[33]:

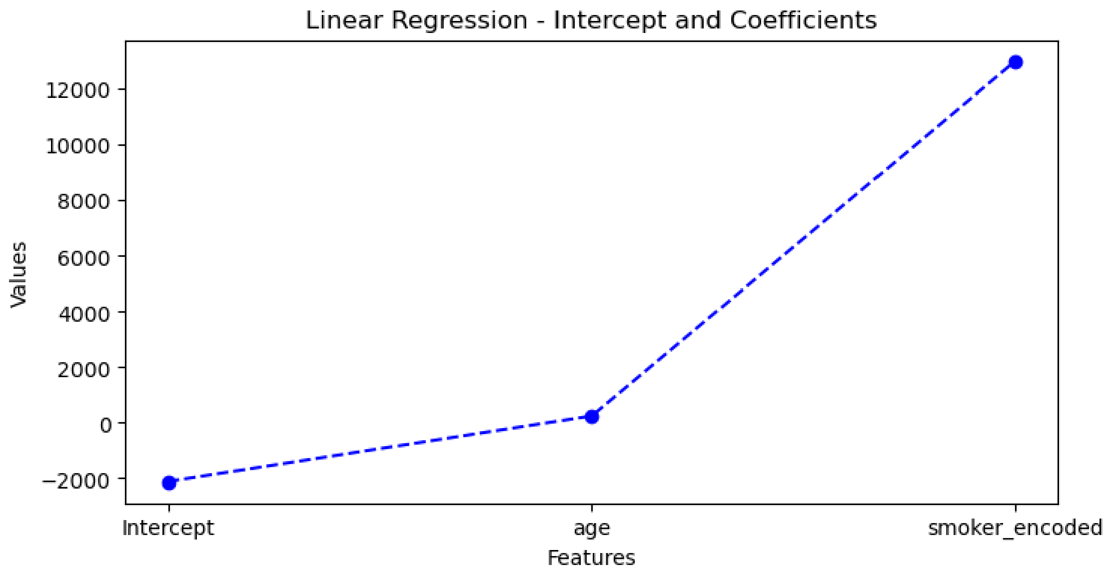
```
# Linear Regression
print("Linear Regression:")
print("Intercept:", lr_model.intercept_)
print("Coefficients:", lr_model.coef_)
print()

# Decision Tree
print("Decision Tree:")
# Feature importances
importance = dt_model.feature_importances_
for i, feature in enumerate(X.columns):
    print(f"{feature}: {importance[i]}")
print()

# Random Forest
print("Random Forest:")
# Feature importances
importance = rf_model.feature_importances_
```

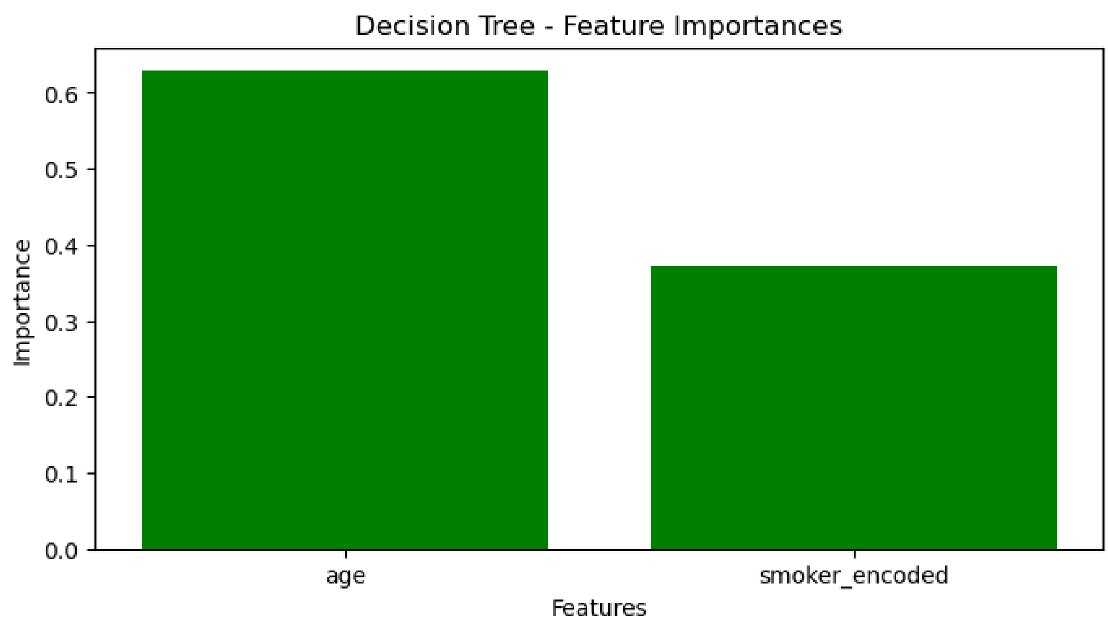
```
for i, feature in enumerate(X.columns):
    print(f"{feature}: {importance[i]}")
print()

# Gradient Boosting
print("Gradient Boosting:")
# Feature importances
importance = gb_model.feature_importances_
for i, feature in enumerate(X.columns):
    print(f"{feature}: {importance[i]}")
```

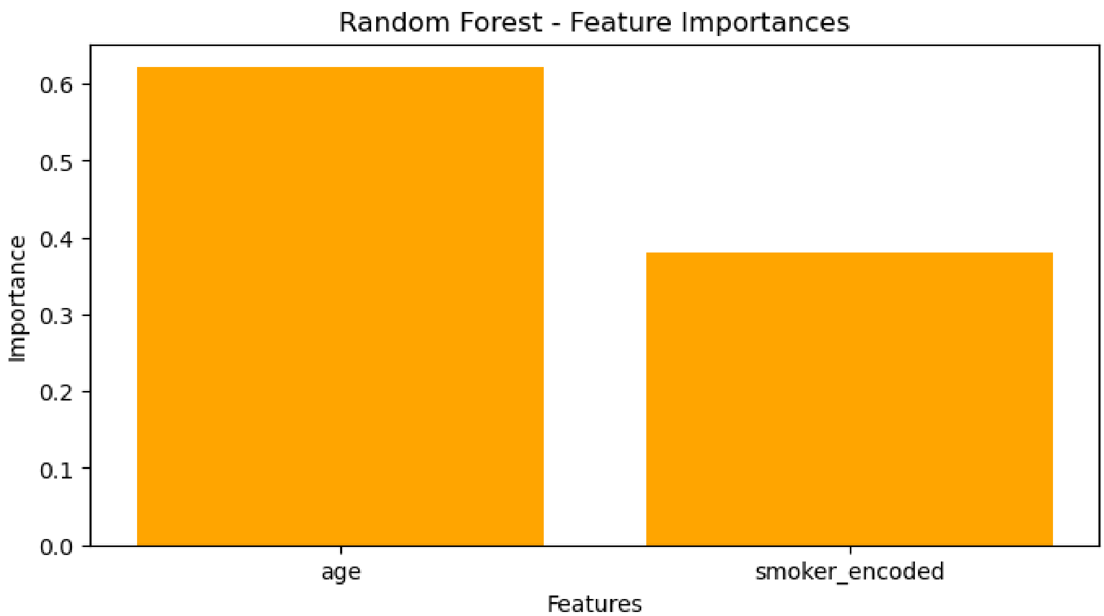
```
Linear Regression:
Intercept: -2114.0646431750074
Coefficients: [  245.45032705 13001.55159468]

Decision Tree:
age: 0.6283321204573397
smoker_encoded: 0.3716678795426604

Random Forest:
age: 0.6098794240338558
smoker_encoded: 0.39012057596614425

Gradient Boosting:
age: 0.6199252235871211
smoker_encoded: 0.3800747764128788
```

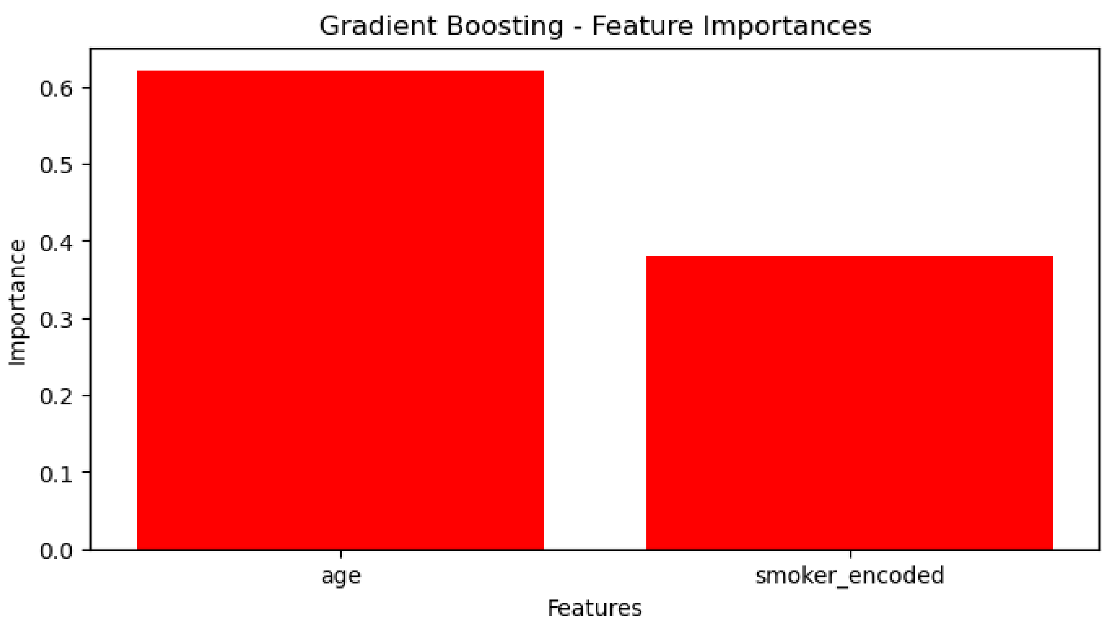
[34]:
```
# Linear Regression
linear_regression_intercept = -2114.0646431750074
linear_regression_coefficients = [245.45032705, 13001.55159468]

# Features
features = ['Intercept', 'age', 'smoker_encoded']
# Values
values = [linear_regression_intercept] + linear_regression_coefficients

# Decision Tree
decision_tree_feature_importances = [0.6283321204573397, 0.3716678795426604]

# Random Forest
random_forest_feature_importances = [0.6206192296796063, 0.3793807703203938]

# Gradient Boosting
gradient_boosting_feature_importances = [0.619925223587121, 0.380074776412879]

# Plot intercept and coefficients for Linear Regression
plt.figure(figsize=(8, 4))
```

```
plt.scatter(features, values, color='blue')
plt.plot(features, values, color='blue', linestyle='--')
plt.title('Linear Regression - Intercept and Coefficients')
plt.xlabel('Features')
plt.ylabel('Values')
plt.show()

# Plot feature importances for Decision Tree
plt.figure(figsize=(8, 4))
plt.bar(['age', 'smoker_encoded'], decision_tree_feature_importances,␣
 ↪color='green')
plt.title('Decision Tree - Feature Importances')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.show()

# Plot feature importances for Random Forest
plt.figure(figsize=(8, 4))
plt.bar(['age', 'smoker_encoded'], random_forest_feature_importances,␣
 ↪color='orange')
plt.title('Random Forest - Feature Importances')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.show()

# Plot feature importances for Gradient Boosting
plt.figure(figsize=(8, 4))
plt.bar(['age', 'smoker_encoded'], gradient_boosting_feature_importances,␣
 ↪color='red')
plt.title('Gradient Boosting - Feature Importances')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.show()
```


Linear Regression - Intercept and Coefficients
12000
10000
8000
6000
4000
2000
0
−2000
Values
Intercept
age
smoker_encoded
Features


Decision Tree - Feature Importances
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Importance
age
smoker_encoded
Features





[35]:
```
# Example input for prediction
new_data = pd.DataFrame({'age': [30], 'smoker_encoded': [1]})

# Linear Regression
lr_predictions = lr_model.predict(new_data)
```

```
print("Linear Regression Predictions:", lr_predictions)

# Decision Tree
dt_predictions = dt_model.predict(new_data)
print("Decision Tree Predictions:", dt_predictions)

# Random Forest
rf_predictions = rf_model.predict(new_data)
print("Random Forest Predictions:", rf_predictions)

# Gradient Boosting
gb_predictions = gb_model.predict(new_data)
print("Gradient Boosting Predictions:", gb_predictions)
```

```
Linear Regression Predictions: [18250.99676297]
Decision Tree Predictions: [19323.2621875]
Random Forest Predictions: [19201.23906329]
Gradient Boosting Predictions: [18919.38558574]
```

[36]:
```
mse_values = [5958015.664548768, 5728632.311218643, 5667626.334646116, 5620075.
 ↪226137549]
mae_values = [1294.7790805755785, 1275.0243094744608, 1266.1176489889306, 1234.
 ↪973946903902]
models = ['Linear Regression', 'Decision Tree', 'Random Forest', 'Gradient␣
 ↪Boosting']

# Find the index of the minimum MSE
min_mse_index = mse_values.index(min(mse_values))
# Find the index of the minimum MAE
min_mae_index = mae_values.index(min(mae_values))

# Get the best model based on MSE
best_model_mse = models[min_mse_index]
best_mse = mse_values[min_mse_index]

# Get the best model based on MAE
best_model_mae = models[min_mae_index]
best_mae = mae_values[min_mae_index]

# Print the best model based on MSE and MAE
print("Best Model based on MSE: ", best_model_mse)
print("MSE: ", best_mse)
print("Best Model based on MAE: ", best_model_mae)
print("MAE: ", best_mae)
```

```
Best Model based on MSE:  Gradient Boosting
MSE:  5620075.226137549
Best Model based on MAE:  Gradient Boosting
```

```
MAE:  1234.973946903902
```

[37]:
```python
# Example input for prediction
new_data = pd.DataFrame({'age': [35], 'smoker_encoded': [0]})
# Gradient Boosting
gb_predictions = gb_model.predict(new_data)
print("Gradient Boosting Predictions:", gb_predictions)
```

```
Gradient Boosting Predictions: [6169.09883742]
```

[38]:
```python
# Example input for prediction
new_data = pd.DataFrame({'age': [35], 'smoker_encoded': [1]})
# Gradient Boosting
gb_predictions = gb_model.predict(new_data)
print("Gradient Boosting Predictions:", gb_predictions)
```

```
Gradient Boosting Predictions: [19735.06590245]
```

[39]:
```python
# Example input for prediction
new_data = pd.DataFrame({'age': [67], 'smoker_encoded': [0]})
# Gradient Boosting
gb_predictions = gb_model.predict(new_data)
print("Gradient Boosting Predictions:", gb_predictions)
```

```
Gradient Boosting Predictions: [14685.8553561]
```

[40]:
```python
# Example input for prediction
new_data = pd.DataFrame({'age': [67], 'smoker_encoded': [1]})
# Gradient Boosting
gb_predictions = gb_model.predict(new_data)
print("Gradient Boosting Predictions:", gb_predictions)
```

```
Gradient Boosting Predictions: [19151.9711604]
```

[41]:
```python
# Example input for prediction
new_data = pd.DataFrame({'age': [55], 'smoker_encoded': [0]})
# Gradient Boosting
gb_predictions = gb_model.predict(new_data)
print("Gradient Boosting Predictions:", gb_predictions)
```

```
Gradient Boosting Predictions: [12234.15555275]
```

[42]:
```python
# Example input for prediction
new_data = pd.DataFrame({'age': [55], 'smoker_encoded': [1]})
# Gradient Boosting
gb_predictions = gb_model.predict(new_data)
print("Gradient Boosting Predictions:", gb_predictions)
```

```
Gradient Boosting Predictions: [18231.52831105]
```