In [1]:
```
#Visualization
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(color_codes=True)
```

In [2]:
```
import os
working_directory = os.getcwd()
print(working_directory)
```

```
/Users/m.adamu/Downloads
```

In [8]:
```
# Load CSV file into a DataFrame
df = pd.read_csv('Medical_insurance.csv')
```

In [9]:
```
#To display the top five rows
df.head()
```

Out[9]:

	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

In [10]:
```
#to give a summary statistics of the Data Set
df.describe()
```

Out[10]:

	age	bmi	children	charges
count	2772.000000	2772.000000	2772.000000	2772.000000
mean	39.109668	30.701349	1.101732	13261.369959
std	14.081459	6.129449	1.214806	12151.768945
min	18.000000	15.960000	0.000000	1121.873900
25%	26.000000	26.220000	0.000000	4687.797000
50%	39.000000	30.447500	1.000000	9333.014350
75%	51.000000	34.770000	2.000000	16577.779500
max	64.000000	53.130000	5.000000	63770.428010

In [11]:
```
#To display the botton 5 rows
df.tail()
```

Out[11]:

	age	sex	bmi	children	smoker	region	charges
2767	47	female	45.320	1	no	southeast	8569.86180
2768	21	female	34.600	0	no	southwest	2020.17700
2769	19	male	26.030	1	yes	northwest	16450.89470
2770	23	male	18.715	0	no	northwest	21595.38229
2771	54	male	31.600	0	no	southwest	9850.43200

In [12]:
```
df.dtypes
```

Out[12]:
```
age           int64
sex          object
bmi         float64
children      int64
smoker       object
region       object
charges     float64
dtype: object
```

In [13]:
```
# Check for missing values
```

```
missing_values = df.isnull().sum()
```

In [14]:
```
# Drop rows with missing values
df_cleaned = df.dropna()
```

In [15]:
```
# View missing values by column
missing_values = df.isnull().sum()
print(missing_values)
```

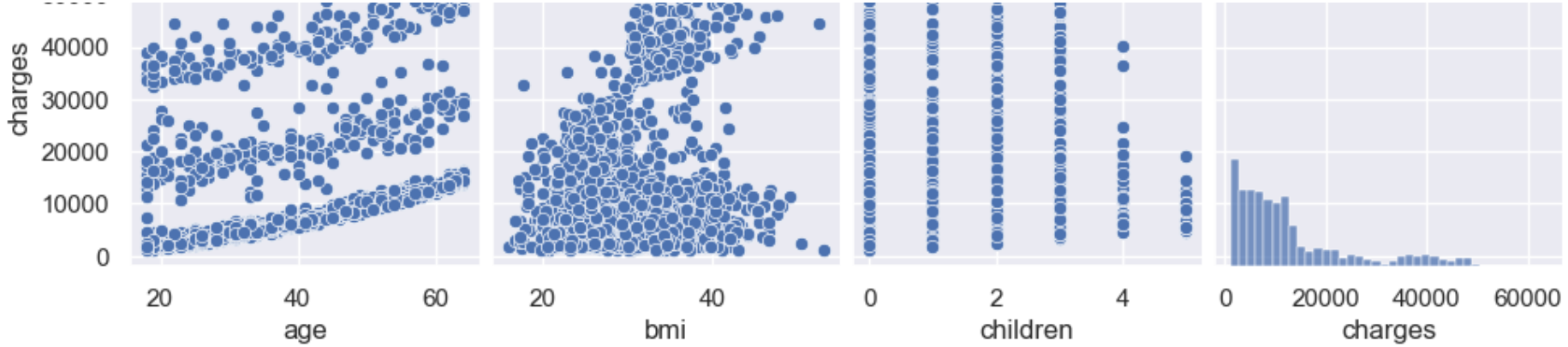
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

In [18]:
```
# List of columns with missing values
columns_with_missing_values = ['age','bmi', 'children',,'charges']
# Fill missing values with mean for each column
for column in columns_with_missing_values:
    df[column] = df[column].fillna(df[column].mean())
```

In [19]:
```
# Pairplot to visualize relationships between numerical variables
sns.pairplot(df)
plt.show()
```

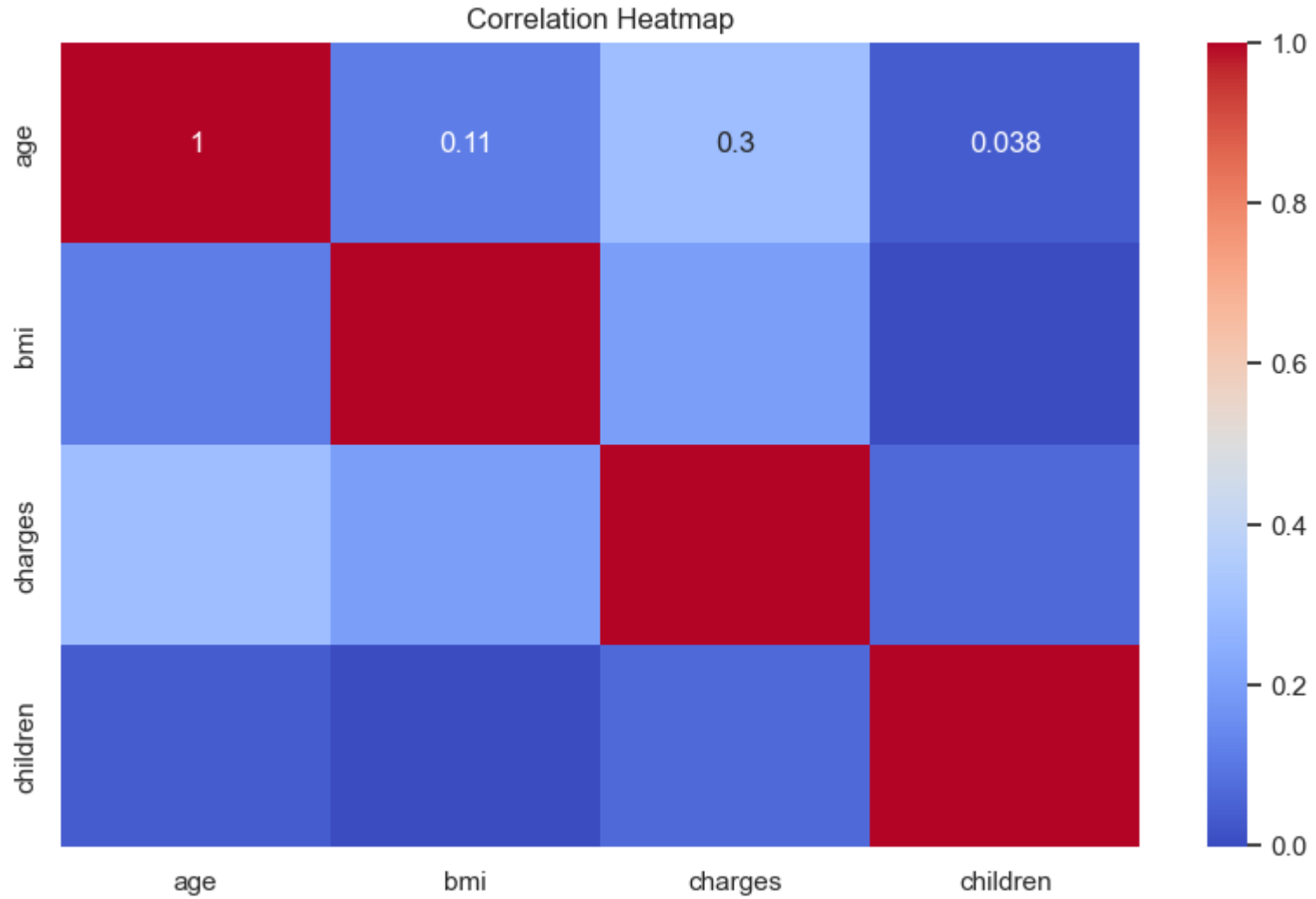
```
/Users/m.adamu/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option i
s deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/Users/m.adamu/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option i
s deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/Users/m.adamu/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option i
s deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/Users/m.adamu/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option i
s deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```


age
bmi
children
60
50
40
30
20
50
40
30
20
5
4
3
2
1
0
60000
50000



In [24]:
```
# Specify columns for correlation heatmap
columns_of_interest = ['age', 'bmi', 'charges','children']

# Compute the correlation matrix for Age,BMI,Charges and Children
correlation_matrix = df[columns_of_interest].corr()
# Plot correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



In [30]:
```
# Select multiple columns of interest
selected_columns = ['age', 'bmi', 'children']

# Compute correlation matrix between selected columns and 'charges'
correlation_with_charges = df[selected_columns + ['charges']].corr()['charges'].sort_values(ascending=False)

print(correlation_with_charges)
```

```
charges     1.000000
age         0.298624
bmi         0.199846
children    0.066442
Name: charges, dtype: float64
```

In [35]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import matplotlib.pyplot as plt


# Select predictor variable(s) (features) for single and multi-variable linear regression
# Single variable
X_single = df[['age']]  # Example: using 'age' as predictor variable
# Multi-variable
X_multi = df[['age', 'bmi', 'children']]  # Example: using 'age', 'bmi', and 'children' as predictor variables
y = df['charges']  # Target variable

# Split the data into training and testing sets (80% train, 20% test)
X_single_train, X_single_test, y_train, y_test = train_test_split(X_single, y, test_size=0.2, random_state=42)
X_multi_train, X_multi_test, _, _ = train_test_split(X_multi, y, test_size=0.2, random_state=42)

# Create and train the single variable linear regression model
model_single = LinearRegression()
model_single.fit(X_single_train, y_train)

# Create and train the multi-variable linear regression model
model_multi = LinearRegression()
model_multi.fit(X_multi_train, y_train)

# Make predictions
y_predict_single = model_single.predict(X_single_test)
y_predict_multi = model_multi.predict(X_multi_test)

# Evaluate the models
r2_single = r2_score(y_test, y_predict_single)
mse_single = mean_squared_error(y_test, y_predict_single)
r2_multi = r2_score(y_test, y_predict_multi)
mse_multi = mean_squared_error(y_test, y_predict_multi)

print("Single Variable Linear Regression:")
print("R-squared:", r2_single)
print("Mean Squared Error:", mse_single)
```

```
print("\nMulti-variable Linear Regression:")
print("R-squared:", r2_multi)
print("Mean Squared Error:", mse_multi)

# Visualize the model's performance (scatter plot of actual vs. predicted charges)
plt.scatter(y_test, y_pred_single, label='Single Variable Linear Regression')
plt.scatter(y_test, y_pred_multi, label='Multi-variable Linear Regression')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.title('Model Performance')
plt.legend()
plt.show()
```

```
Single Variable Linear Regression:
R-squared: 0.120560254669691
Mean Squared Error: 134977253.89197862

Multi-variable Linear Regression:
R-squared: 0.15987237337389792
Mean Squared Error: 128943592.28463751
```

In [36]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import r2_score, mean_squared_error


# Select predictor variables and target variable
X = df[['age', 'bmi', 'children']]  # Example: selecting 'age', 'bmi', and 'children' as predictors
y = df['charges']  # Target variable

# 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 train the Linear Regression model
model_lr = LinearRegression()
```

```
model_lr.fit(X_train, y_train)

# Evaluate the Linear Regression model
y_pred_lr = model_lr.predict(X_test)
r2_lr = r2_score(y_test, y_pred_lr)
mse_lr = mean_squared_error(y_test, y_pred_lr)

print("Linear Regression:")
print("R-squared:", r2_lr)
print("Mean Squared Error:", mse_lr)

# Create and train the Ridge Regression model
alpha = 1.0  # Regularization strength, you can tune this hyperparameter
model_ridge = Ridge(alpha=alpha)
model_ridge.fit(X_train, y_train)

# Evaluate the Ridge Regression model
y_pred_ridge = model_ridge.predict(X_test)
r2_ridge = r2_score(y_test, y_pred_ridge)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)

print("\nRidge Regression:")
print("R-squared:", r2_ridge)
print("Mean Squared Error:", mse_ridge)
```

```
Linear Regression:
R-squared: 0.15987237337389792
Mean Squared Error: 128943592.28463751

Ridge Regression:
R-squared: 0.15987222203586549
Mean Squared Error: 128943615.51214358
```

In [ ]: