Final Project

April 17, 2024

[1]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in ./anaconda3/lib/python3.11/site-
packages (2.0.3)
Requirement already satisfied: numpy in ./anaconda3/lib/python3.11/site-packages
(from xgboost) (1.24.3)
Requirement already satisfied: scipy in ./anaconda3/lib/python3.11/site-packages
(from xgboost) (1.11.1)
```

[2]:
```
pip install --upgrade pip
```

```
Requirement already satisfied: pip in ./anaconda3/lib/python3.11/site-packages
(24.0)
Note: you may need to restart the kernel to use updated packages.
```

[3]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression, BayesianRidge, Ridge, Lasso
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.compose import make_column_transformer,␣
↪TransformedTargetRegressor, ColumnTransformer, make_column_selector
from sklearn.preprocessing import OneHotEncoder, PolynomialFeatures,␣
↪StandardScaler, LabelEncoder
from sklearn.dummy import DummyRegressor
from sklearn.metrics import mean_squared_error, r2_score, median_absolute_error
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from sklearn.tree import plot_tree, DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor, BaggingRegressor,␣
↪RandomForestRegressor
from xgboost import XGBRegressor
from xgboost.plotting import plot_importance
from sklearn.cluster import KMeans
import plotly.graph_objects as go
from sklearn.neural_network import MLPRegressor
```

```python
from sklearn import preprocessing
import warnings
warnings.filterwarnings('ignore')
```

[4]:
```python
insurance=pd.read_csv('insurance.csv')
insurance1= pd.read_csv('insurance.csv')
insurance2=pd.read_csv('insurance.csv')
df=pd.read_csv('insurance.csv')
data=pd.read_csv('insurance.csv')
```

[5]:
```python
insurance.head()
```

[5]:

	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

[6]:
```python
insurance.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
```

[7]:
```python
insurance.describe()
```

[7]:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

[8]:
```
insurance.describe(include = 'all')
```

[8]:
```
                age   sex          bmi     children smoker     region  \
count   1338.000000  1338  1338.000000  1338.000000   1338       1338
unique          NaN     2          NaN          NaN      2          4
top             NaN  male          NaN          NaN     no  southeast
freq            NaN   676          NaN          NaN   1064        364
mean      39.207025   NaN    30.663397     1.094918    NaN        NaN
std       14.049960   NaN     6.098187     1.205493    NaN        NaN
min       18.000000   NaN    15.960000     0.000000    NaN        NaN
25%       27.000000   NaN    26.296250     0.000000    NaN        NaN
50%       39.000000   NaN    30.400000     1.000000    NaN        NaN
75%       51.000000   NaN    34.693750     2.000000    NaN        NaN
max       64.000000   NaN    53.130000     5.000000    NaN        NaN

             charges
count    1338.000000
unique           NaN
top              NaN
freq             NaN
mean    13270.422265
std     12110.011237
min      1121.873900
25%      4740.287150
50%      9382.033000
75%     16639.912515
max     63770.428010
```

[9]:
```
insurance.shape
```

[9]:
```
(1338, 7)
```

[10]:
```
cat_cols = insurance.select_dtypes('object').columns.tolist()
print('Categorical Features:')
print(', '.join(cat_cols))
```

```
Categorical Features:
sex, smoker, region
```

[11]:
```
insurance.select_dtypes('object').nunique()
```

[11]:
```
sex       2
smoker    2
region    4
dtype: int64
```

[12]:
```
insurance.isna().sum()
```

[12]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

[13]:
```
insurance['age'].value_counts()[:30]
```

[13]:
```
age
18    69
19    68
50    29
51    29
47    29
46    29
45    29
20    29
48    29
52    29
22    28
49    28
54    28
53    28
21    28
26    28
24    28
25    28
28    28
27    28
23    28
43    27
29    27
30    27
41    27
42    27
44    27
31    27
40    27
32    26
Name: count, dtype: int64
```

[14]:
```
insurance['sex'].value_counts()
```

[14]:
```
sex
male      676
female    662
Name: count, dtype: int64
```

[15]:
```
insurance['sex'].value_counts(normalize = True)
```

[15]:
```
sex
male      0.505232
female    0.494768
Name: proportion, dtype: float64
```

Male are coded with “1” and Female are coded with “0”

[16]:
```
insurance['children'].value_counts()
```

[16]:
```
children
0    574
1    324
2    240
3    157
4     25
5     18
Name: count, dtype: int64
```

[17]:
```
child_m = data['children'].describe()
child_m['mean']
```

[17]:
```
1.0949177877429
```

[18]:
```
insurance['smoker'].value_counts()
```

[18]:
```
smoker
no     1064
yes     274
Name: count, dtype: int64
```

[19]:
```
insurance['smoker'].value_counts(normalize= True)
```

[19]:
```
smoker
no     0.795217
yes    0.204783
Name: proportion, dtype: float64
```

[20]:
```
smoke_habits_gender = insurance.groupby(['sex'])['smoker'].value_counts()
smoke_habits_gender
```

[20]:
```
sex     smoker
female  no        547
        yes       115
male    no        517
        yes       159
Name: count, dtype: int64
```

[21]:
```
insurance['region'].value_counts()
```

[21]:
```
region
southeast    364
southwest    325
northwest    325
northeast    324
Name: count, dtype: int64
```

[22]:
```
insurance['region'].value_counts(normalize= True)
```

[22]:
```
region
southeast    0.272048
southwest    0.242900
northwest    0.242900
northeast    0.242152
Name: proportion, dtype: float64
```

[23]:
```
charges_m = insurance['charges'].describe()
charges_m['mean']
```

[23]:
```
13270.422265141257
```

Converting objects labels into categorical

[24]:
```
insurance[['sex', 'smoker', 'region']] = insurance[['sex', 'smoker', 'region']].
  ↪astype('category')
insurance.dtypes
```

[24]:
```
age            int64
sex         category
bmi          float64
children       int64
smoker      category
region      category
charges      float64
dtype: object
```

[25]:
```
df[['sex', 'smoker', 'region']] = df[['sex', 'smoker', 'region']].
  ↪astype('category')
df.dtypes
```

[25]:
```
age            int64
sex         category
bmi          float64
children       int64
smoker      category
region      category
charges      float64
dtype: object
```

Converting category labels into numerical using LabelEncoder

[26]:
```
label = LabelEncoder()
label.fit(insurance.sex.drop_duplicates())
insurance.sex = label.transform(insurance.sex)
label.fit(insurance.smoker.drop_duplicates())
insurance.smoker = label.transform(insurance.smoker)
label.fit(insurance.region.drop_duplicates())
insurance.region = label.transform(insurance.region)
insurance.dtypes
```

[26]:
```
age           int64
sex           int64
bmi         float64
children      int64
smoker        int64
region        int64
charges     float64
dtype: object
```

[27]:
```
label = LabelEncoder()
label.fit(df.sex.drop_duplicates())
df.sex = label.transform(df.sex)
label.fit(df.smoker.drop_duplicates())
df.smoker = label.transform(df.smoker)
label.fit(df.region.drop_duplicates())
df.region = label.transform(df.region)
df.dtypes
```

[27]:
```
age           int64
sex           int64
bmi         float64
children      int64
smoker        int64
region        int64
charges     float64
dtype: object
```

[28]:
```
insurance.corr()
```

[28]:

	age	sex	bmi	children	smoker	region	charges
age	1.000000	-0.020856	0.109272	0.042469	-0.025019	0.002127	0.299008
sex	-0.020856	1.000000	0.046371	0.017163	0.076185	0.004588	0.057292
bmi	0.109272	0.046371	1.000000	0.012759	0.003750	0.157566	0.198341
children	0.042469	0.017163	0.012759	1.000000	0.007673	0.016569	0.067998
smoker	-0.025019	0.076185	0.003750	0.007673	1.000000	-0.002181	0.787251
region	0.002127	0.004588	0.157566	0.016569	-0.002181	1.000000	-0.006208
charges	0.299008	0.057292	0.198341	0.067998	0.787251	-0.006208	1.000000

[29]:

```
df.corr()
```

[29]:

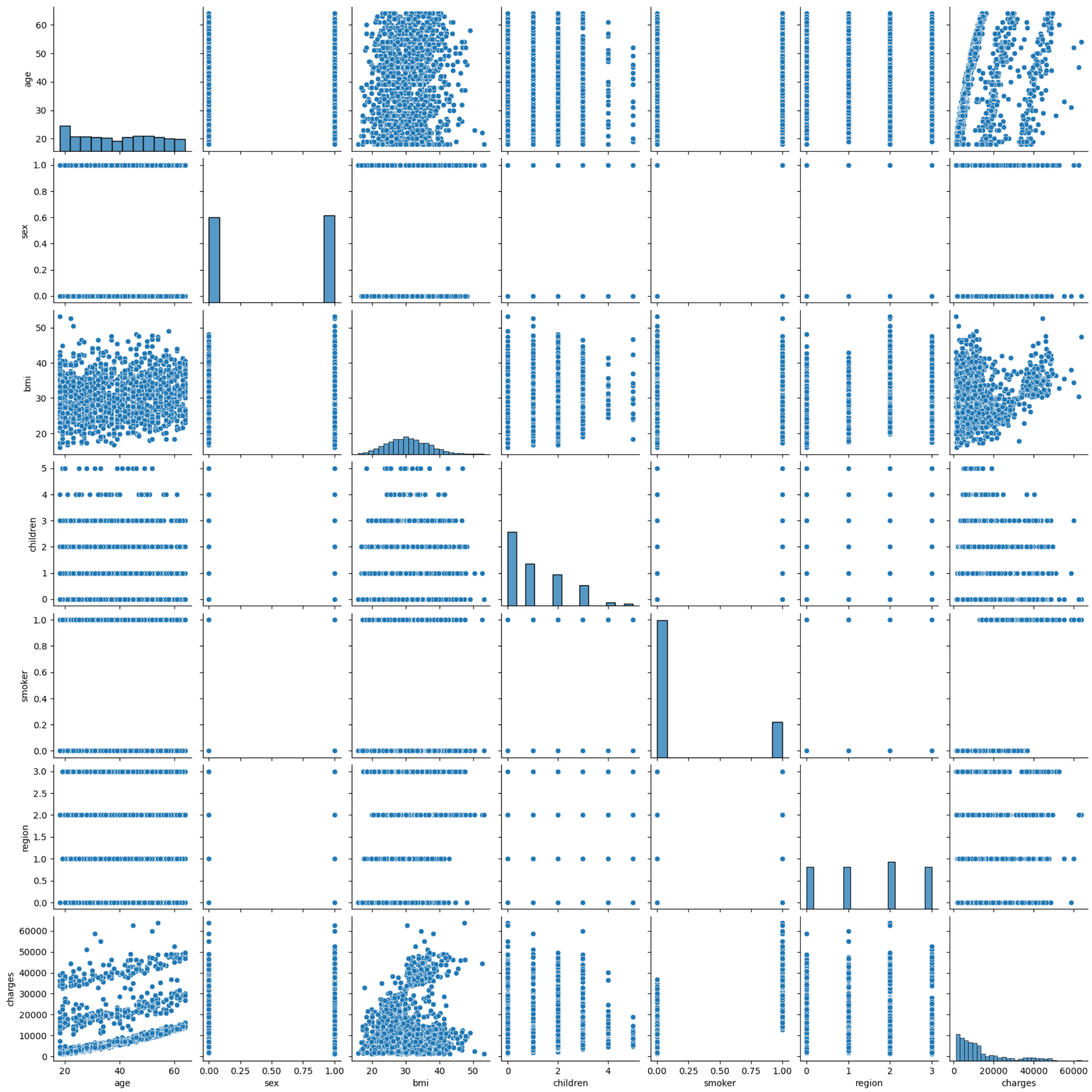
	age	sex	bmi	children	smoker	region	charges
age	1.000000	-0.020856	0.109272	0.042469	-0.025019	0.002127	0.299008
sex	-0.020856	1.000000	0.046371	0.017163	0.076185	0.004588	0.057292
bmi	0.109272	0.046371	1.000000	0.012759	0.003750	0.157566	0.198341
children	0.042469	0.017163	0.012759	1.000000	0.007673	0.016569	0.067998
smoker	-0.025019	0.076185	0.003750	0.007673	1.000000	-0.002181	0.787251
region	0.002127	0.004588	0.157566	0.016569	-0.002181	1.000000	-0.006208
charges	0.299008	0.057292	0.198341	0.067998	0.787251	-0.006208	1.000000

Male are coded with "1" and Female are coded with "0"

[30]:

```
sns.pairplot(insurance)
```

[30]: <seaborn.axisgrid.PairGrid at 0x136a2cc90>



[31]:
```
insurance.corr()['charges'].sort_values()
```

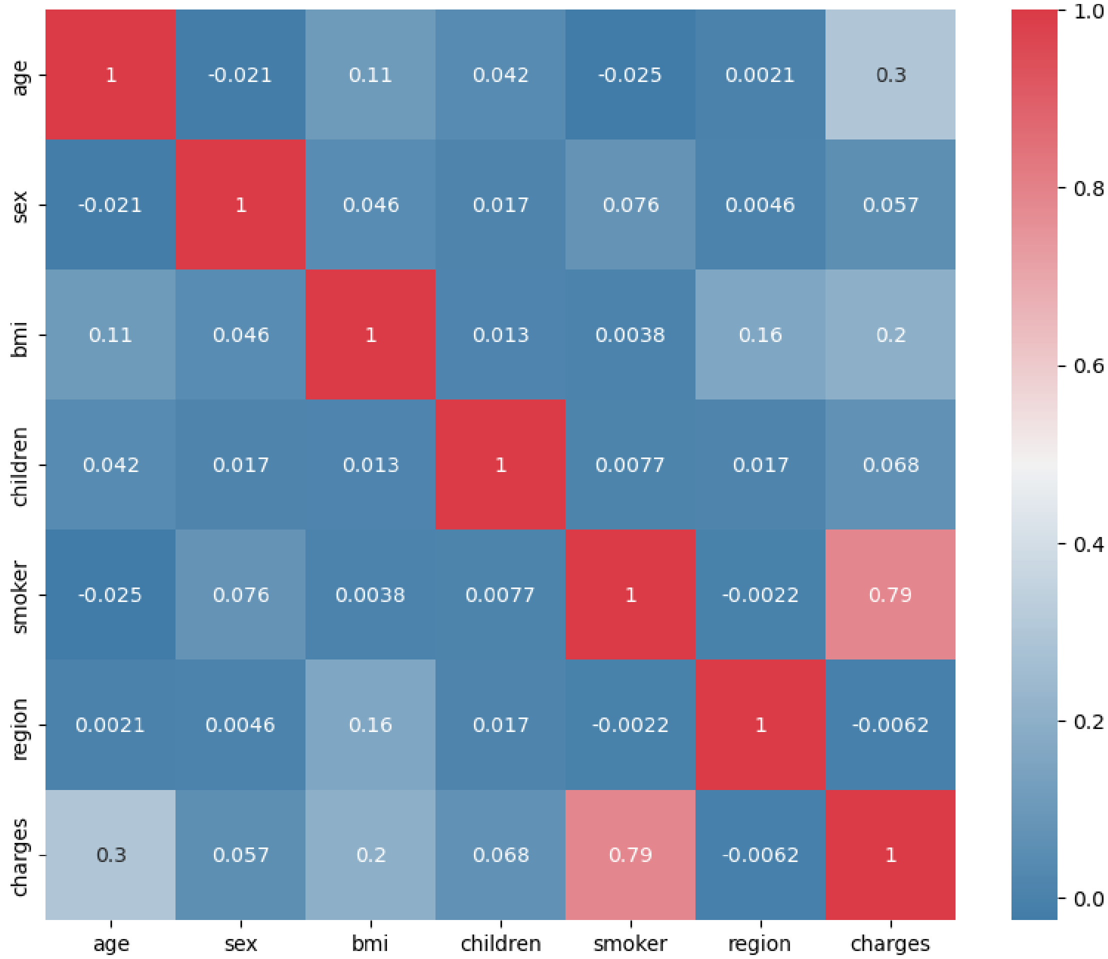
[31]:
```
region      -0.006208
sex          0.057292
children     0.067998
bmi          0.198341
age          0.299008
smoker       0.787251
charges      1.000000
Name: charges, dtype: float64
```

[32]:
```
plt.figure(figsize = (15,5))
f, ax = plt.subplots(figsize=(10, 8))
```

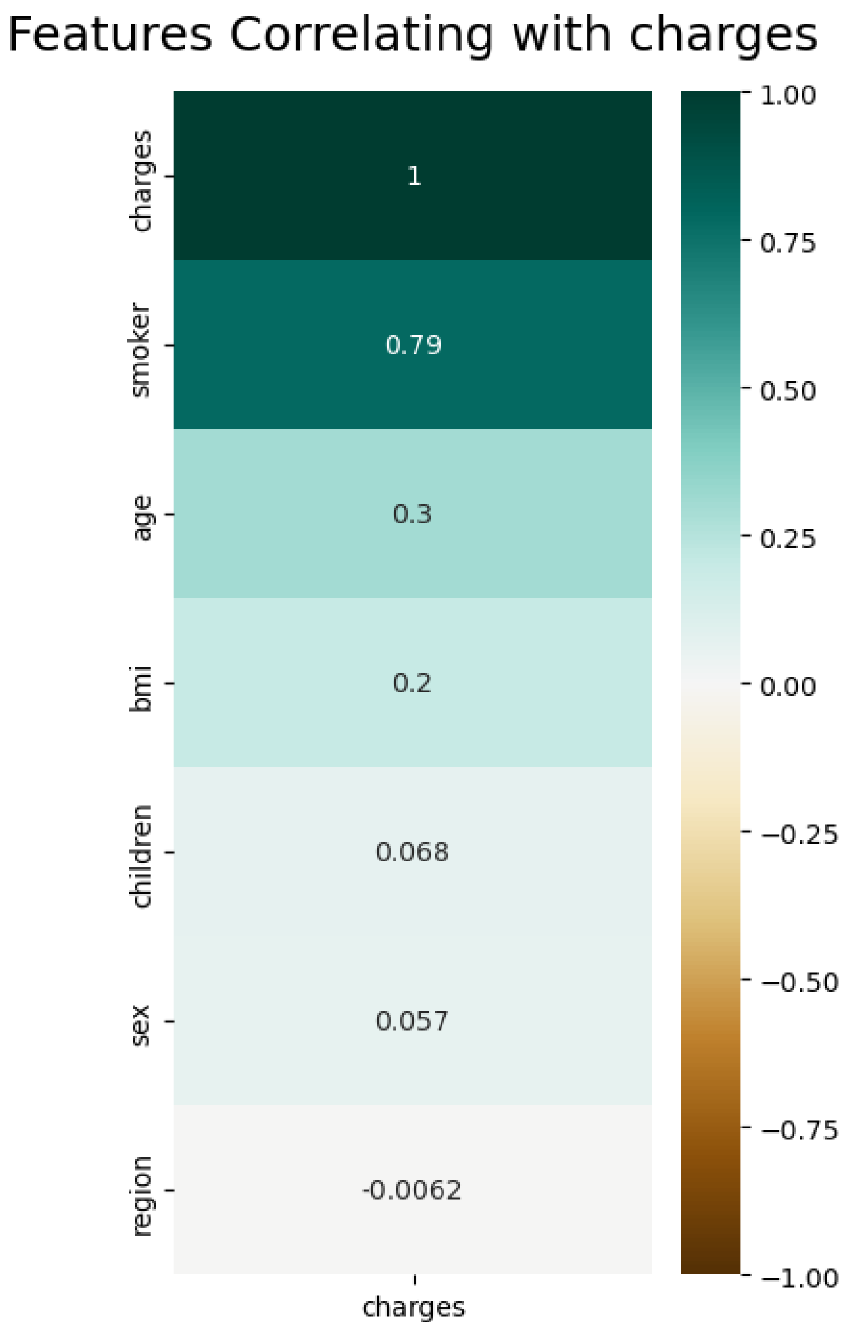
```
corr = insurance.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool_), cmap=sns.
  ↪diverging_palette(240,10,as_cmap=True),
            square=True, ax=ax, annot=True)
```

[32]: <Axes: >

<Figure size 1500x500 with 0 Axes>



[33]:
```
plt.figure(figsize=(4, 8))
heatmap = sns.heatmap(insurance.corr(numeric_only=True)[['charges']].
  ↪sort_values(by='charges', ascending=False), vmin=-1, vmax=1, annot=True,␣
  ↪cmap='BrBG')
heatmap.set_title('Features Correlating with charges', fontdict={'fontsize':
  ↪18}, pad=16);
```



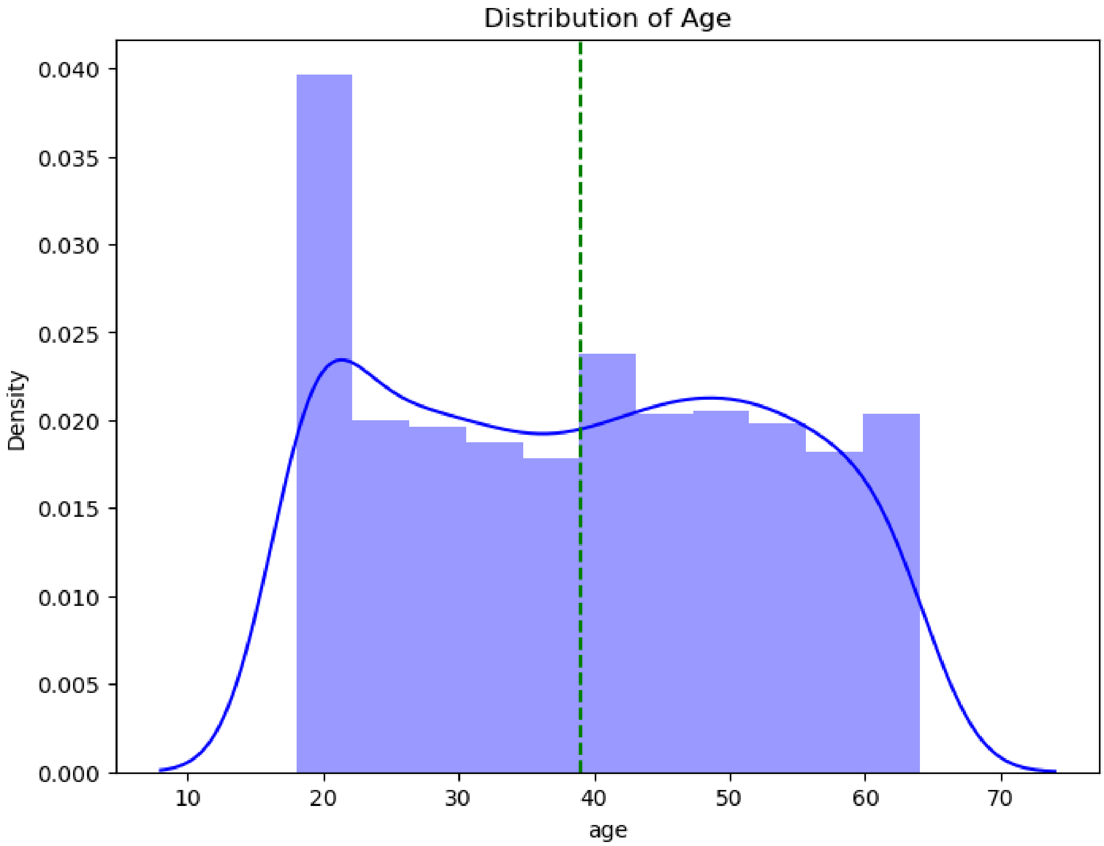
Smoking is the most important feature in deciding insurance charges of an individual followed by age and bmi. Now I'll explore each feature in detail:

let's pay attention to the age of the patients and explore it in a bit more detail.

[34]:
```
plt.figure(figsize=(8,6))
plt.title("Distribution of Age")
```

```
ax = sns.distplot(insurance["age"], color = 'b')
plt.axvline(39, linestyle = '--', color = 'green', label = 'mean Age')
```

[34]: <matplotlib.lines.Line2D at 0x143e32bd0>

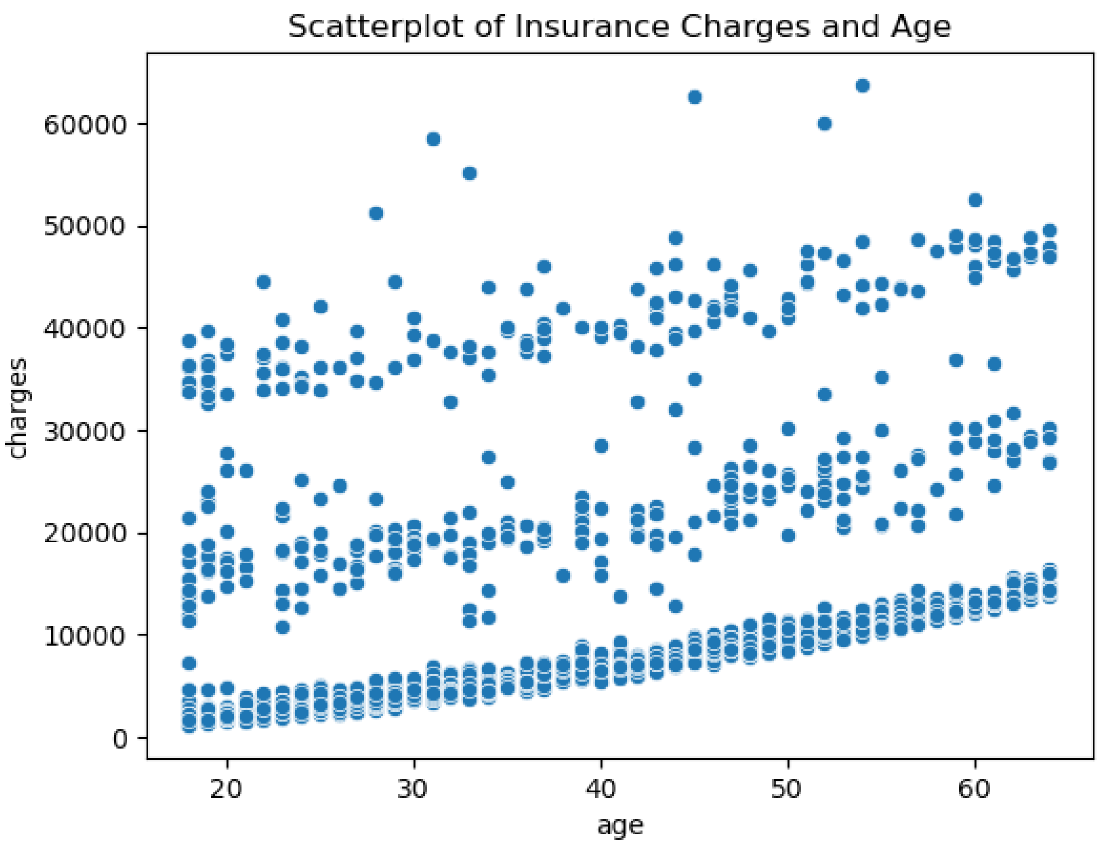


[35]:
```
labels = data['age'].value_counts().keys().to_list()
values = data['age'].value_counts().to_list()

fig = go.Figure(go.Pie(labels=labels,
                       values=values,
                       hole=0.5))
fig.show()
```

We have patients under 20 in our data set. This is the minimum age of patients in our set. The maximum age is 64 years.

[36]:
```
sns.scatterplot(x=data.age,y=data.charges,palette='Set1').
 ↪set(title='Scatterplot of Insurance Charges and Age')
```

[36]: [Text(0.5, 1.0, 'Scatterplot of Insurance Charges and Age')]

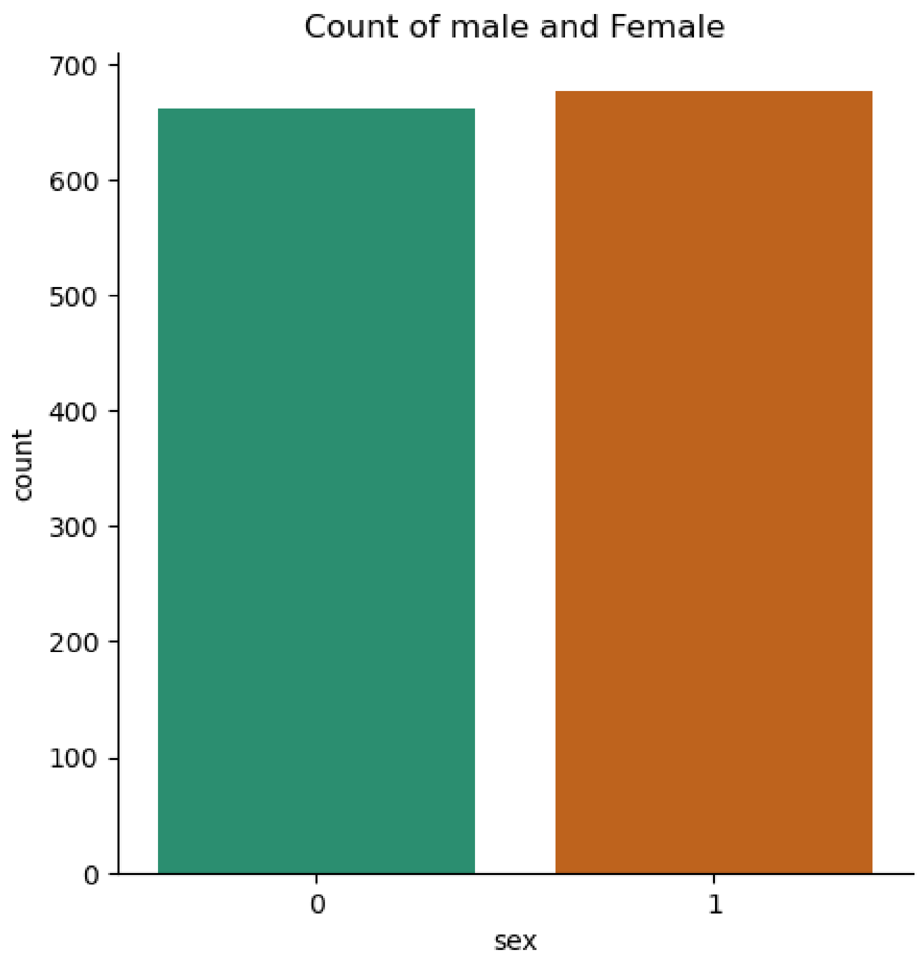


The scatterplot shows, Insurance Charges generally increase with increasing Age other than a few exceptions. That probably has to do with smoking habits of individuals. May be that person is a non-smoker!!!

Now to study "Sex" in detail.

[37]:
```
#Sex
sns.catplot(x="sex", kind="count", palette="Dark2", data=insurance)

plt.title("Count of male and Female")
```

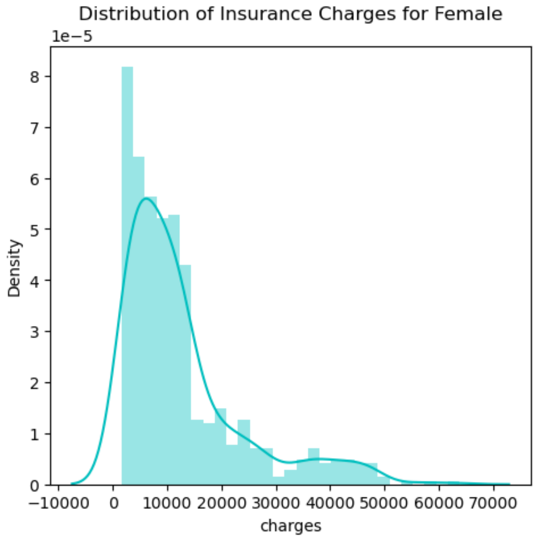
[37]:
```
Text(0.5, 1.0, 'Count of male and Female')
```

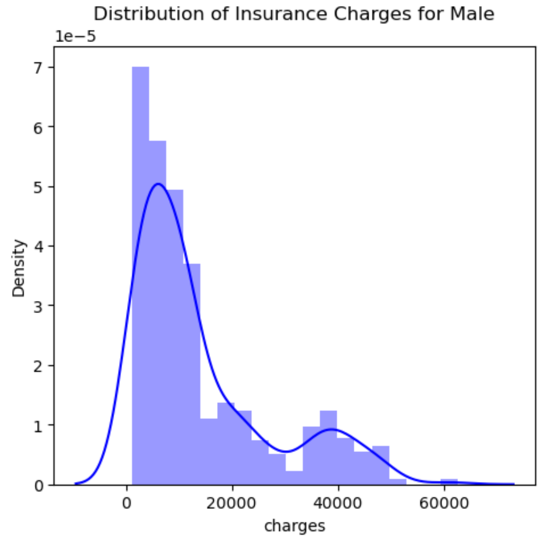


That means there isn't much difference between number of male and female patients.

[38]:
```
f=plt.figure(figsize=(12,5))

ax=f.add_subplot(121)
sns.distplot(data[(data.sex == 'female')]["charges"],color='c',ax=ax)
ax.set_title('Distribution of Insurance Charges for Female')

ax=f.add_subplot(122)
sns.distplot(data[(data.sex == 'male')]['charges'],color='b',ax=ax)
ax.set_title('Distribution of Insurance Charges for Male')
```

[38]:
```
Text(0.5, 1.0, 'Distribution of Insurance Charges for Male')
```

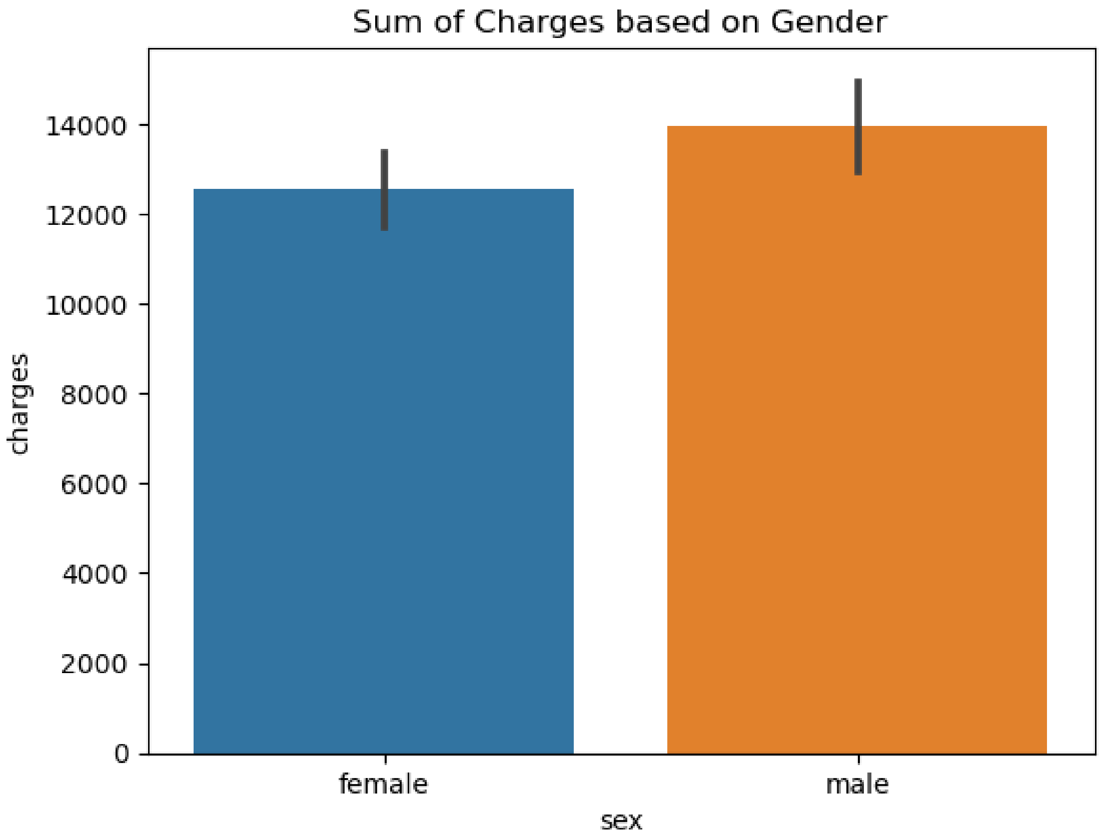




Comparison between Insurance charges of female and male. There isn't a marked difference. That means gender doesn't play an important role in deciding Insurance Charges of an individual. Exact value is 0.057, value taken from collinearity table.

[39]:
```
sns.barplot(data=data, x='sex', y='charges')
plt.title("Sum of Charges based on Gender")
```

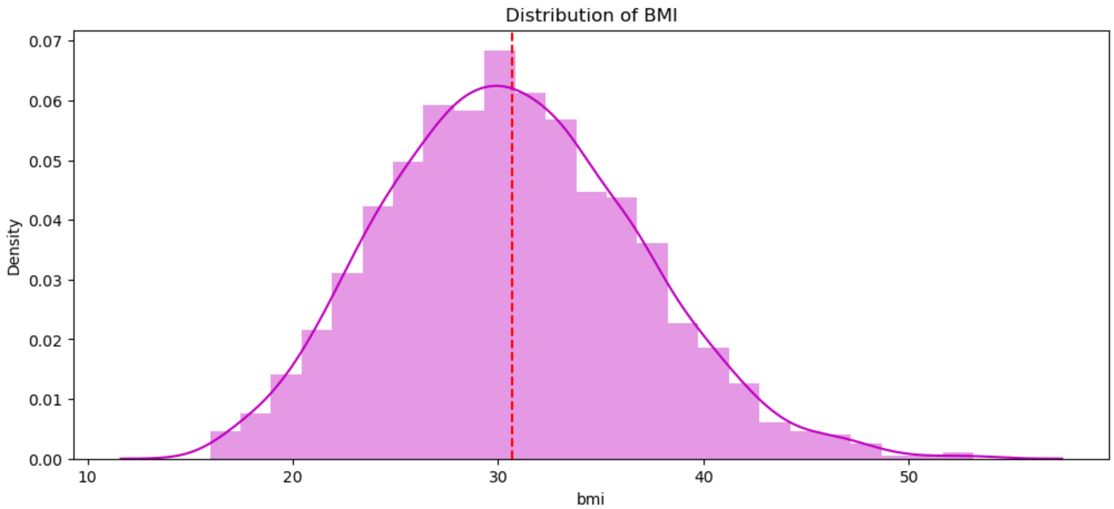
[39]:
```
Text(0.5, 1.0, 'Sum of Charges based on Gender')
```



Now let's pay attention to bmi.

[40]:
```
plt.figure(figsize=(8,6))
bmi_des = data['bmi'].describe()
plt.figure(figsize=(12,5))
plt.title("Distribution of BMI")
plt.axvline(bmi_des['mean'], linestyle = "--", color = "red")
ax = sns.distplot(insurance["bmi"], color = 'm')
```

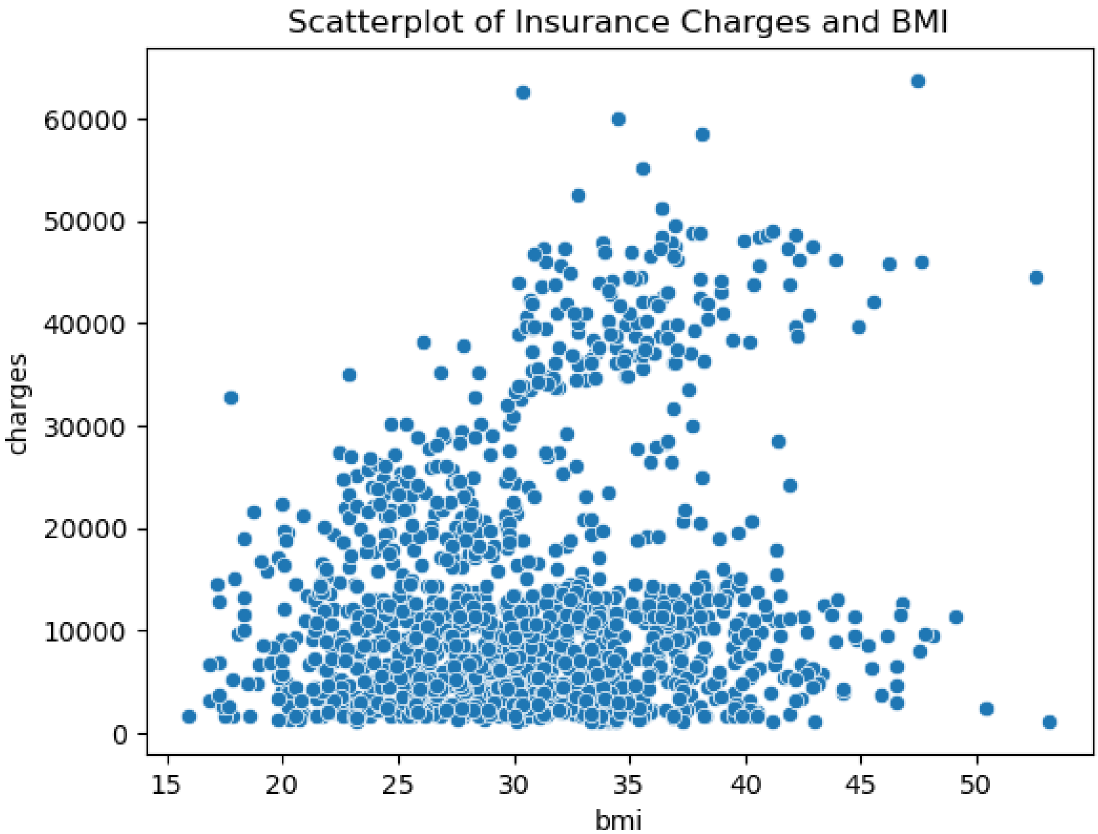
```
<Figure size 800x600 with 0 Axes>
```



The average BMI in patients is 30.66 represented by the red line in the graph which is a fairly uniformly distributed graph.

[41]:
```
sns.scatterplot(x=data.bmi,y=data.charges,palette='Set1').
 ↪set(title='Scatterplot of Insurance Charges and BMI')
```

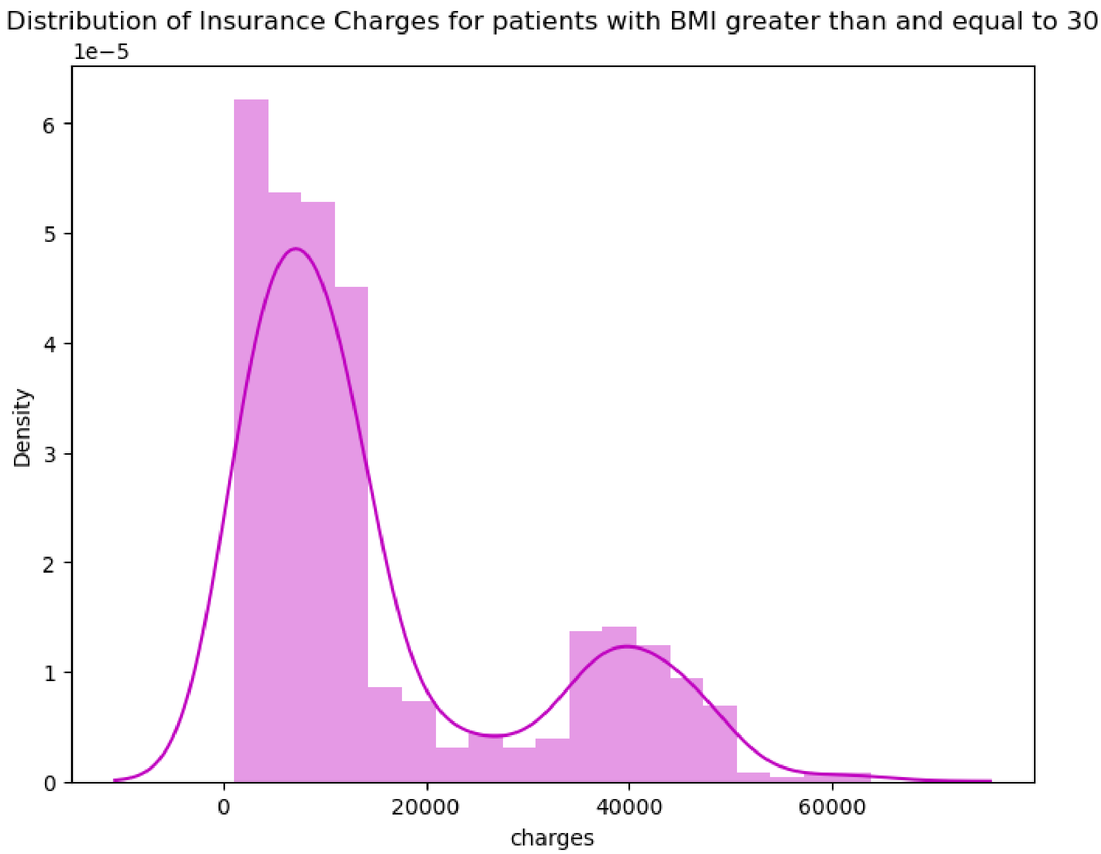
[41]:
```
[Text(0.5, 1.0, 'Scatterplot of Insurance Charges and BMI')]
```



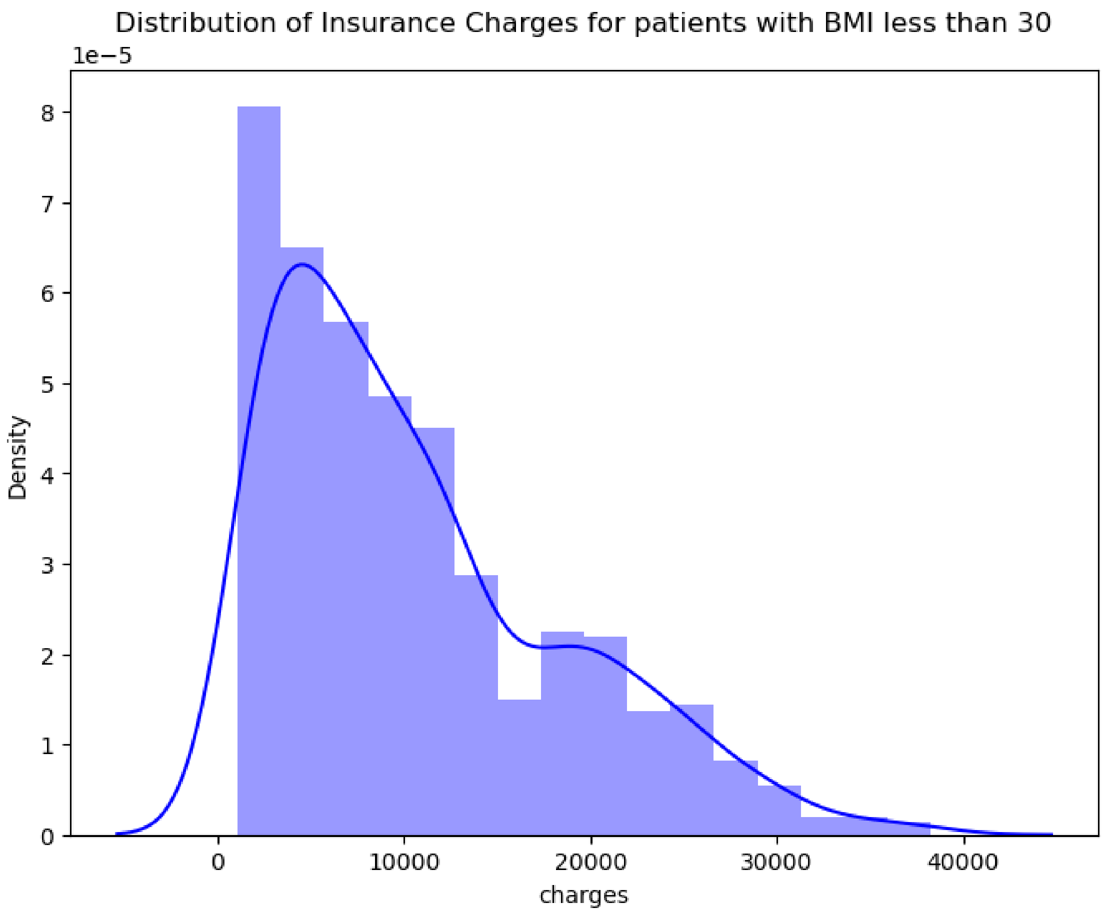
The scatterplot shows a general trend of increase in Insurance Charges with increase in BMI.

Let's explore bmi further. let's look at the distribution of costs in patients with BMI greater than 30 and less than 30.

[42]:
```
plt.figure(figsize=(8,6))
plt.title("Distribution of Insurance Charges for patients with BMI greater than␣
↪and equal to 30")
ax = sns.distplot(insurance[(insurance.bmi >= 30)]['charges'], color = 'm')
```



[43]:
```
plt.figure(figsize=(8,6))
plt.title("Distribution of Insurance Charges for patients with BMI less than␣
↪30")
ax = sns.distplot(insurance[(insurance.bmi < 30)]['charges'], color = 'b')
```

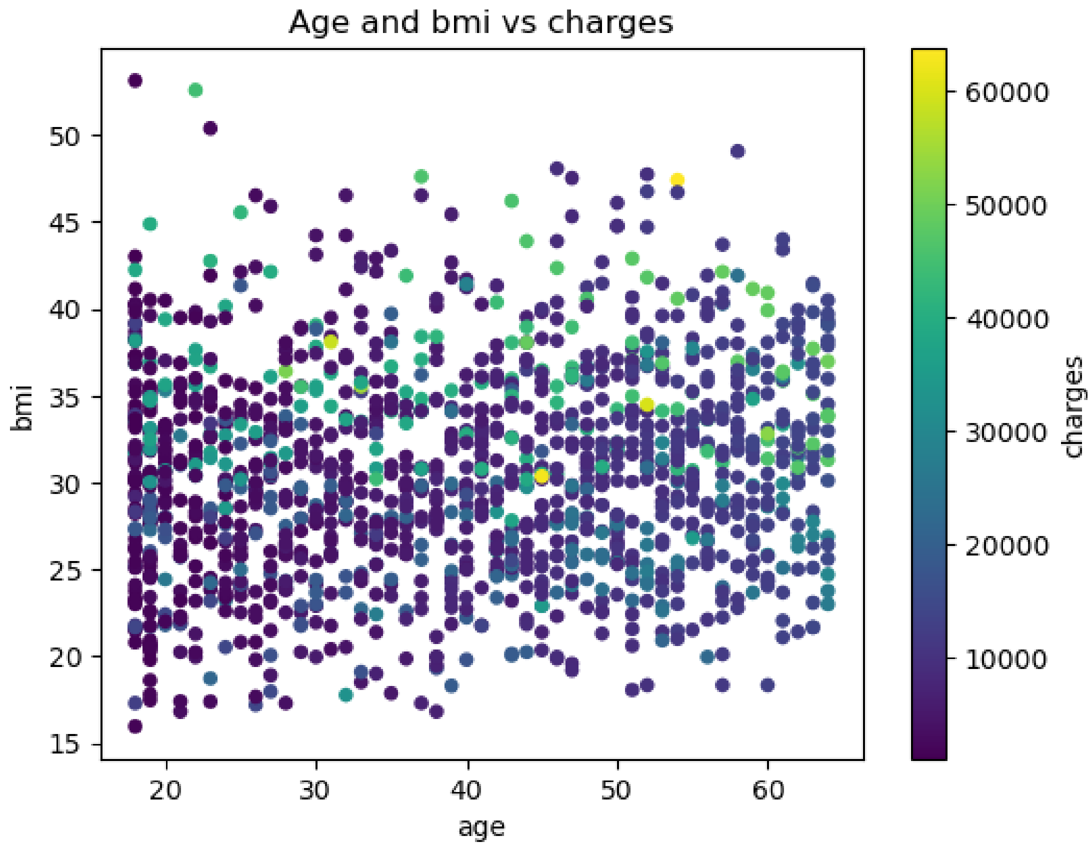


The comparison clearly shows Insurance Charges for individuals with BMI greater than or equal 30 are more than for individuals with BMI less than 30.

Plot graph to show relationship between Age, BMI and Insurance Charges.

[44]:
```
insurance.plot(kind='scatter', x='age', y='bmi',c= 'charges')
plt.title("Age and bmi vs charges")

#It shows that age and bmi dont have a collective effect on insurance charges.
```

[44]:
```
Text(0.5, 1.0, 'Age and bmi vs charges')
```



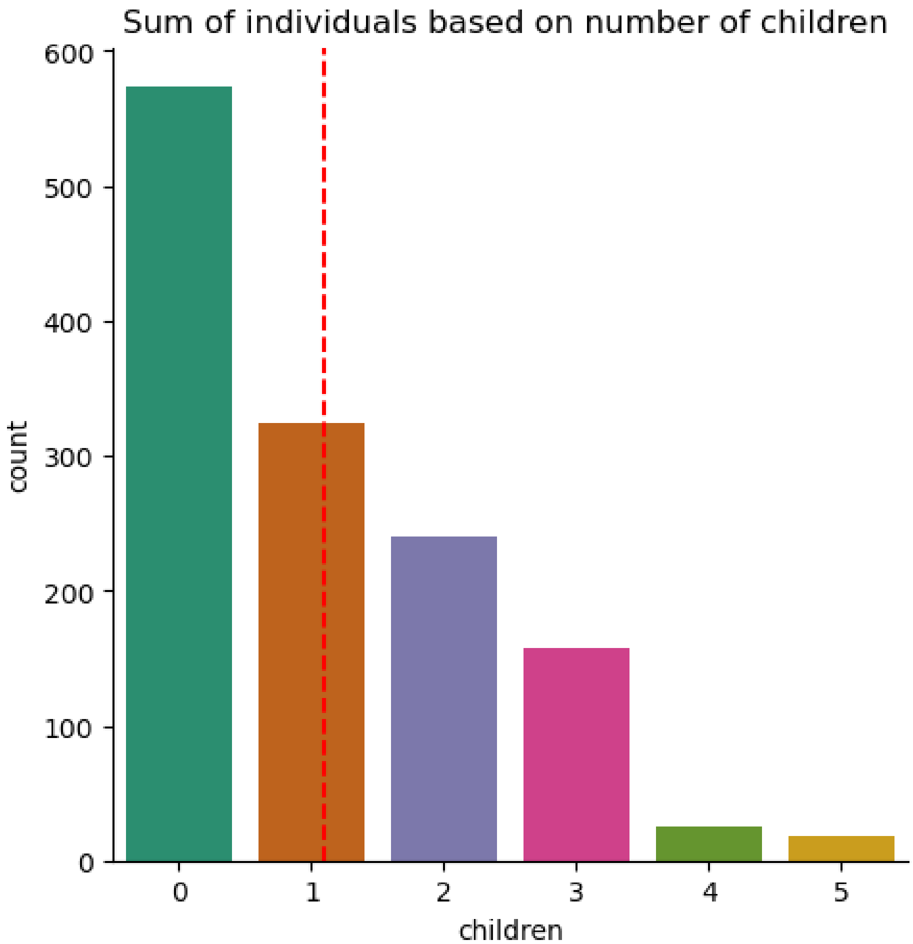
The scatterplot shows no collinearity between Age and BMI when deciding Insurance charges. Points are scattered randomly all over the plot. The exact value is: 0.1092, value taken from the collinearity table and the heatmap.

Let's pay attention to children. First, let's see how many children our patients have.

[45]:
```
print('Mean value of children: {}'.format(child_m['mean']))
sns.catplot(x="children", kind="count", palette="Dark2", data=insurance)
plt.axvline(child_m['mean'], linestyle = "--", color = "red", )
plt.title("Sum of individuals based on number of children ")
```

```
Mean value of children: 1.0949177877429
```

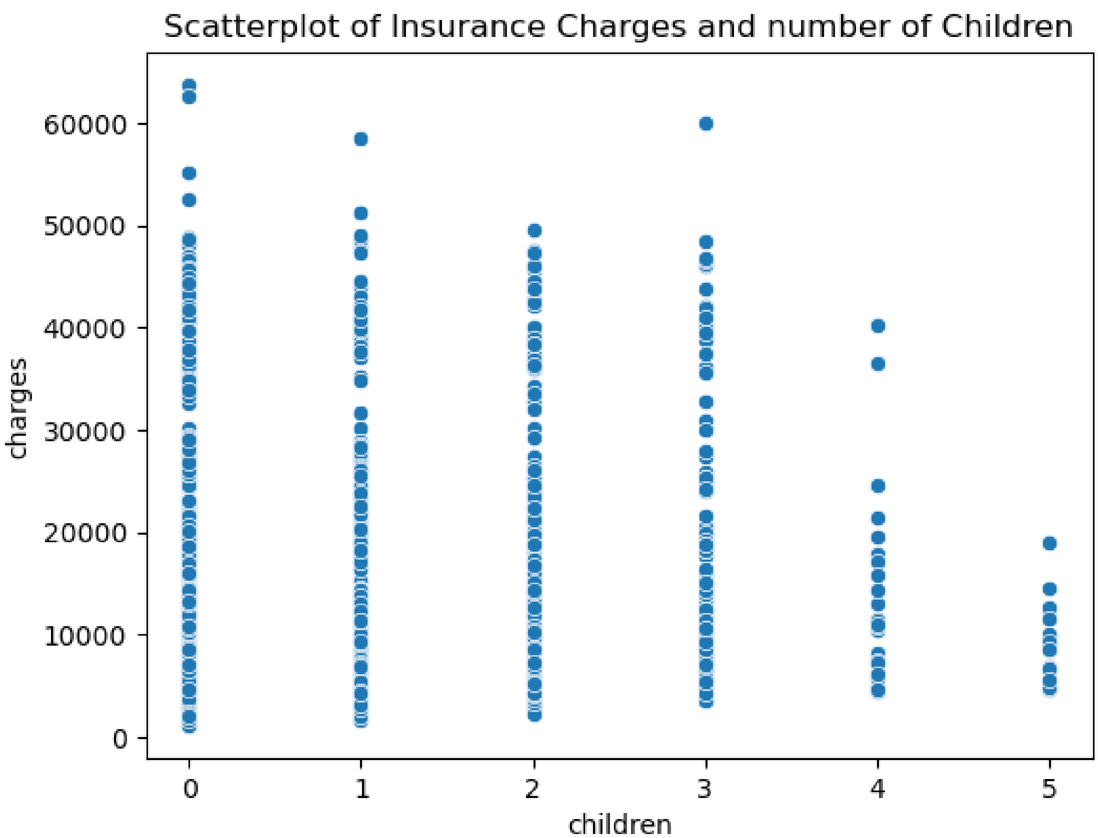
[45]:
```
Text(0.5, 1.0, 'Sum of individuals based on number of children ')
```



Most individuals have no child and a very few have 5 children. Mean number of children: 1.09, value taken from describe() method.

[46]:
```
sns.scatterplot(x=data.children,y=data.charges,palette='Set1').
 ↪set(title='Scatterplot of Insurance Charges and number of Children')
```

[46]: [Text(0.5, 1.0, 'Scatterplot of Insurance Charges and number of Children')]

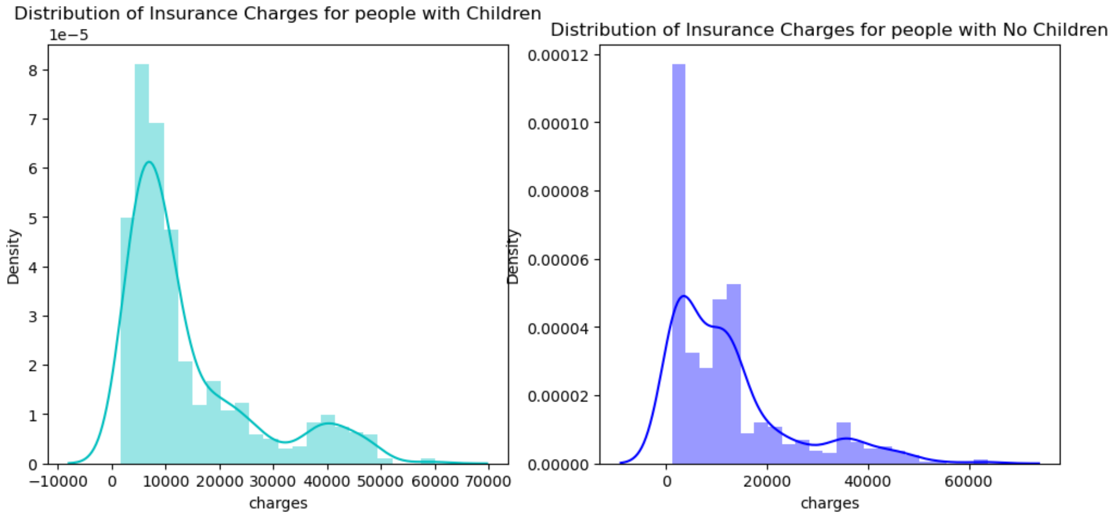


Comparison of Distribution of Insurance Charges for people with children or no children.

Number of children doesn't play much role in deciding Insurance charges of an individual. Exact value is: 0.068 (Value taken from the heatmap). This can be further explained by the following subplots:

[47]:
```
f= plt.figure(figsize=(12,5))

ax=f.add_subplot(121)
sns.distplot(insurance[(insurance.children >0 )]["charges"],color='c',ax=ax)
ax.set_title('Distribution of Insurance Charges for people with Children')

ax=f.add_subplot(122)
sns.distplot(insurance[(insurance.children == 0)]['charges'],color='b',ax=ax)
ax.set_title('Distribution of Insurance Charges for people with No Children')
```

[47]:
```
Text(0.5, 1.0, 'Distribution of Insurance Charges for people with No Children')
```

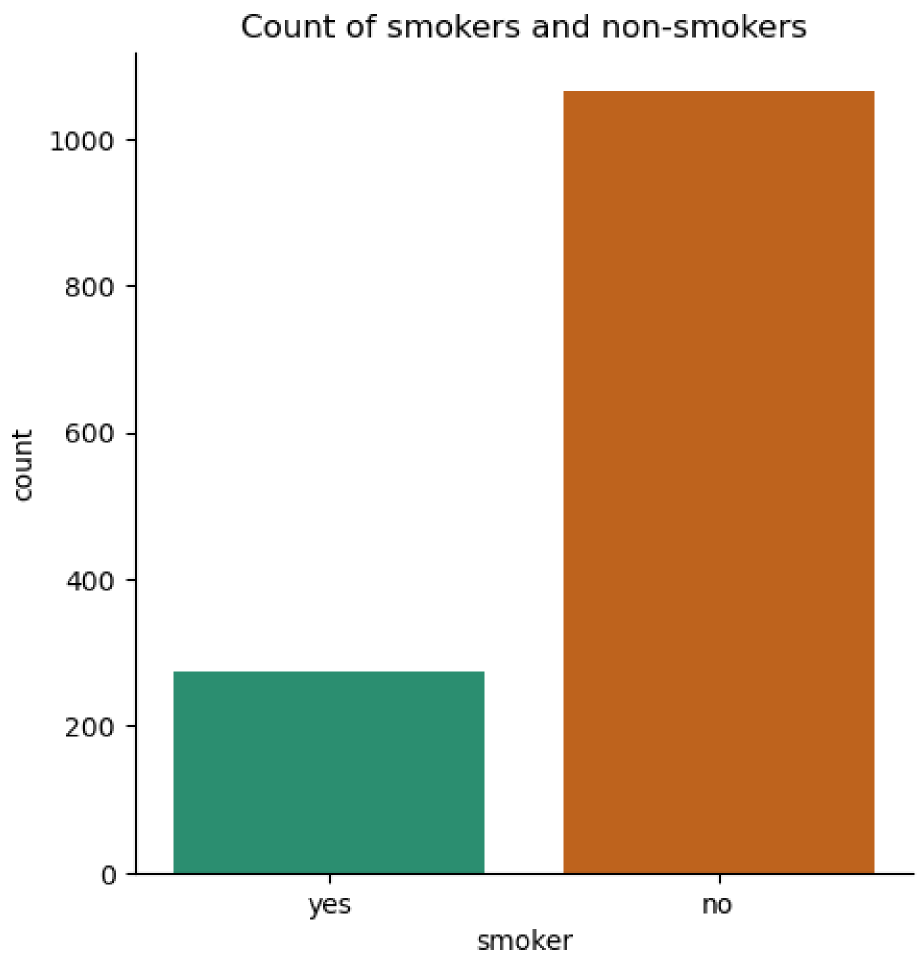


Comparison of Insurance charges of individuals with no child to individuals with children. Not much difference

Investigating smoker feature now

[48]:
```
sns.catplot(x="smoker", kind="count", palette="Dark2",data=data)
plt.title("Count of smokers and non-smokers")
```

[48]:
```
Text(0.5, 1.0, 'Count of smokers and non-smokers')
```



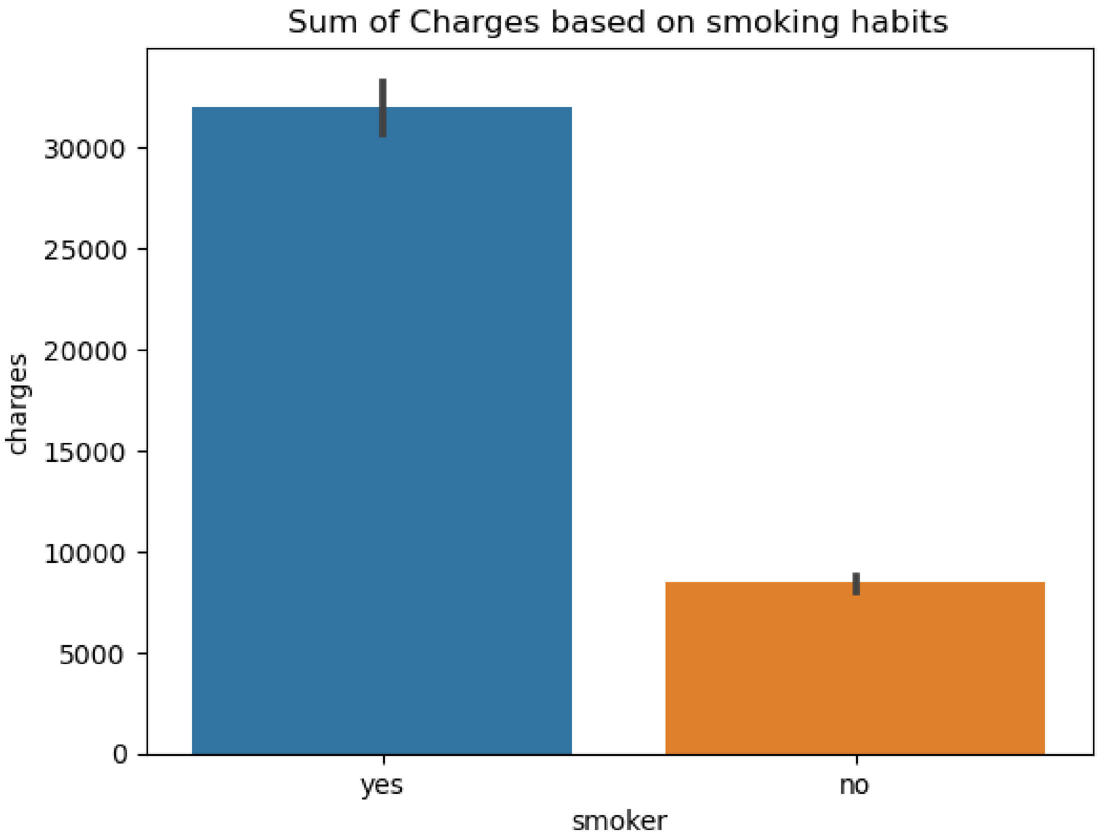
The dataset has more non-smoking than smoking individuals.

[49]:
```
keys = data['smoker'].value_counts().keys().to_list()
values = data['smoker'].value_counts().to_list()
fig = go.Figure(go.Pie(labels=keys,
                       values= values,
                       hole = 0.5))
fig.show()
```

There are more non smokers than smokers in our data set.

[50]:
```
sns.barplot(data=data, x='smoker', y='charges')
plt.title("Sum of Charges based on smoking habits")
```

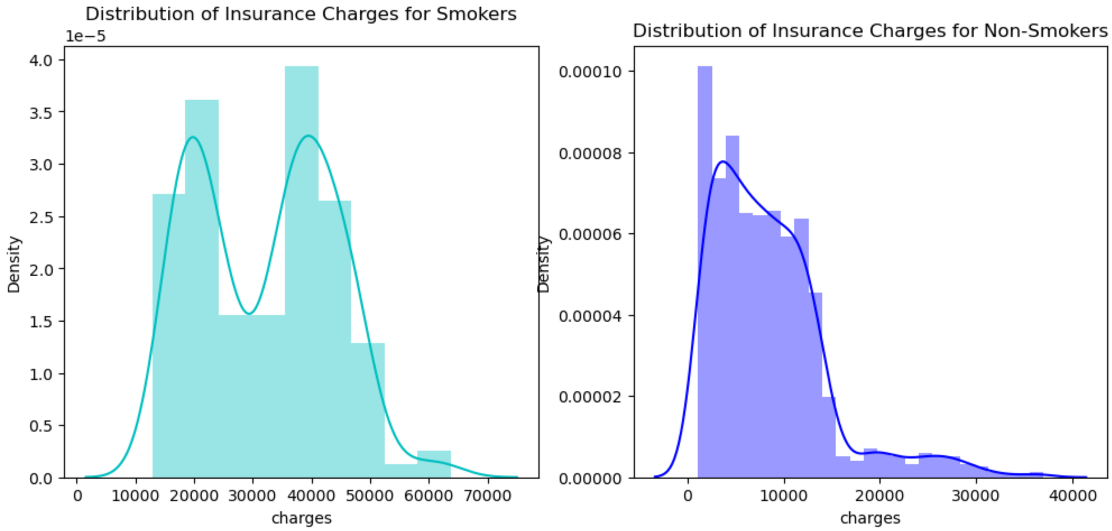
[50]:
```
Text(0.5, 1.0, 'Sum of Charges based on smoking habits')
```



Smokers pay much more Insurance charges than non-smokers. Exact value is: 0.79 which shows a strong relationship. Value taken from the heatmap. This can be explained further with the help of following subplots.

[51]:
```
f= plt.figure(figsize=(12,5))

ax=f.add_subplot(121)
sns.distplot(insurance[(insurance.smoker == 1)]["charges"],color='c',ax=ax)
ax.set_title('Distribution of Insurance Charges for Smokers')

ax=f.add_subplot(122)
sns.distplot(insurance[(insurance.smoker == 0)]['charges'],color='b',ax=ax)
ax.set_title('Distribution of Insurance Charges for Non-Smokers')
```

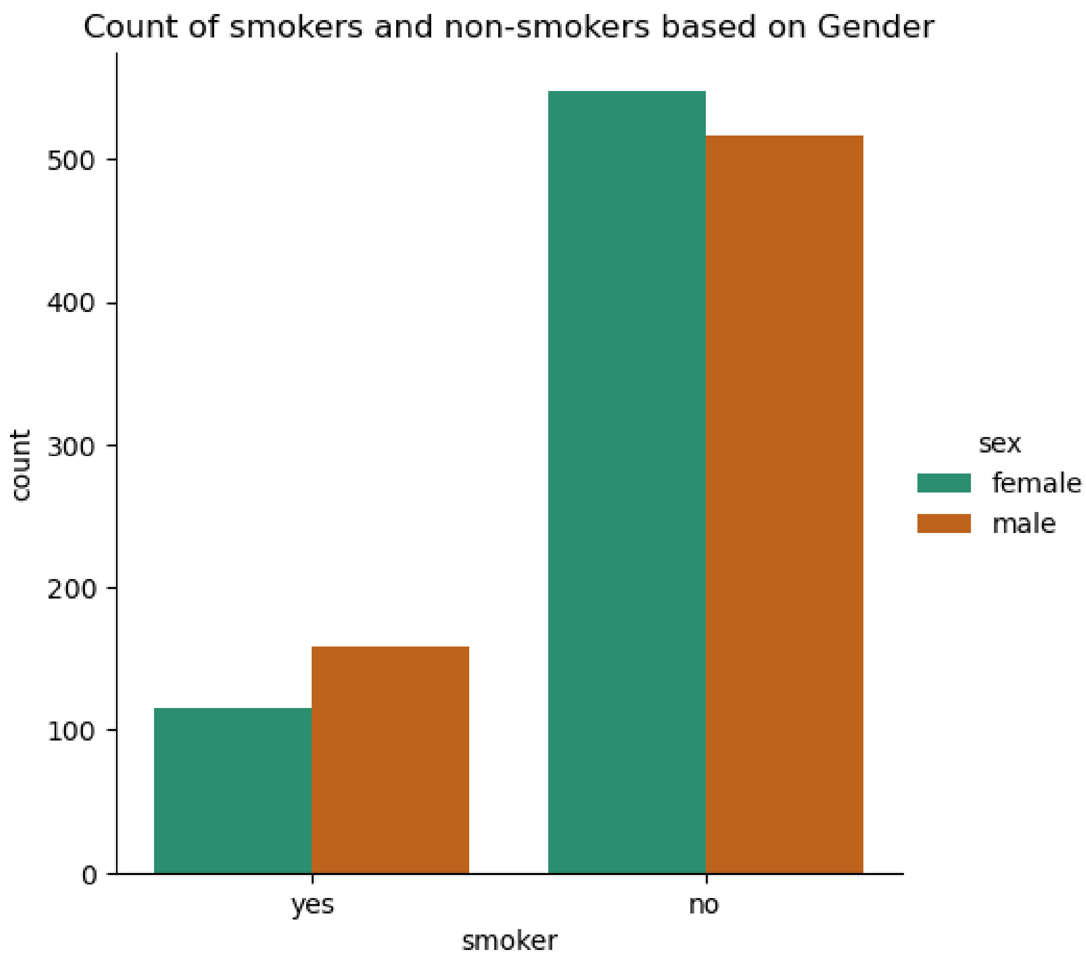
[51]:
```
Text(0.5, 1.0, 'Distribution of Insurance Charges for Non-Smokers')
```



Plot explaining how many males and females are smokers and how many are non smokers.

[52]:
```
sns.catplot(x="smoker", kind="count", hue = 'sex', palette="Dark2", data=data)
plt.title("Count of smokers and non-smokers based on Gender")
```

[52]:
```
Text(0.5, 1.0, 'Count of smokers and non-smokers based on Gender')
```



We can notice that the dataset has more male smokers than women smokers and more female non-smokers than male non-smokers. And that can be explained below:

[53]:
```
smoke_habits_gender
```

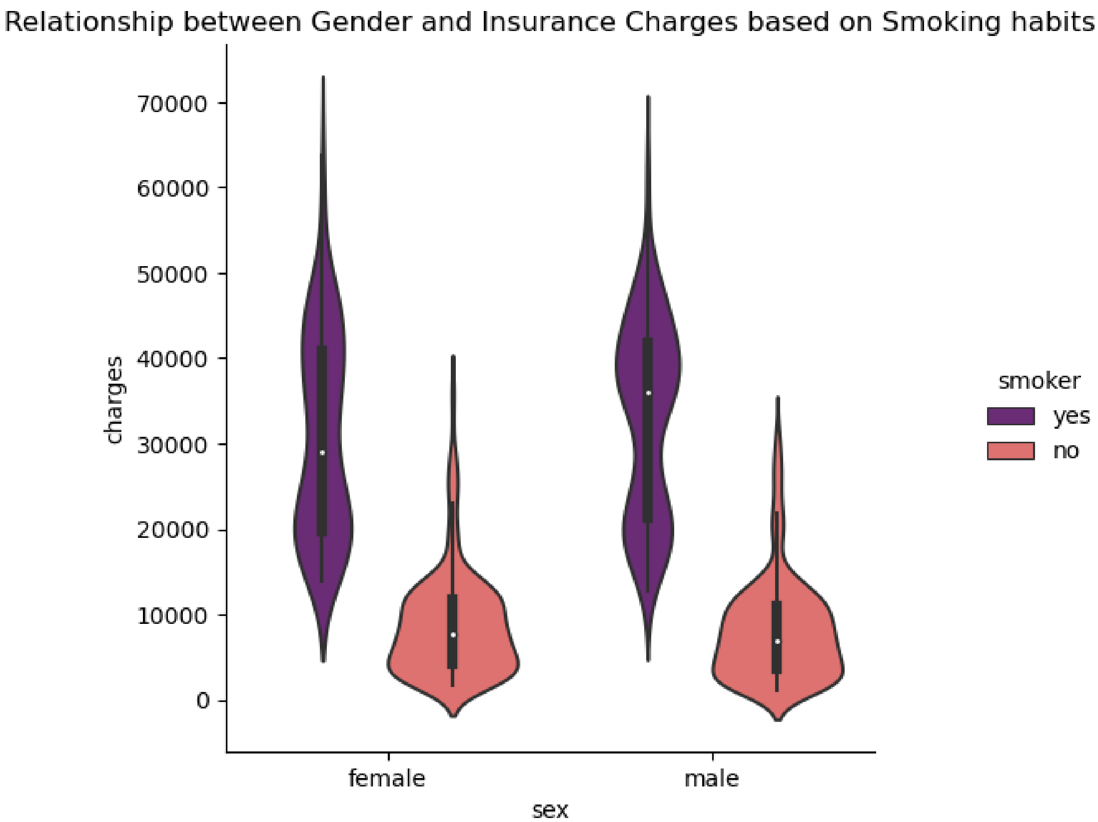
[53]:
```
sex     smoker
female  no        547
        yes       115
male    no        517
        yes       159
Name: count, dtype: int64
```

Violin graph explaining relationship between Gender and Insurance Charges based on whether they are Smoker or Non Smoker.

[54]:
```
sns.catplot(x="sex", y="charges", hue="smoker",
            kind="violin", data=data, palette = 'magma')
```

```
plt.title("Relationship between Gender and Insurance Charges based on Smoking␣
↪habits")
```

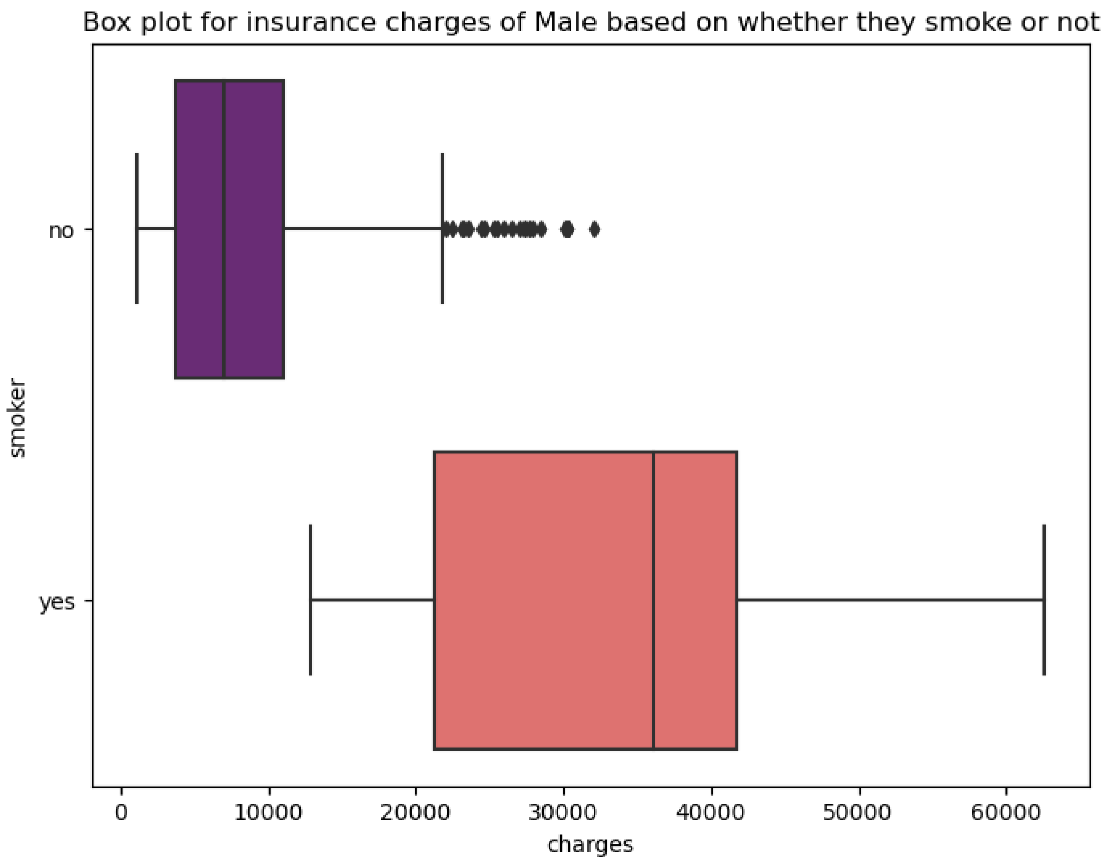
[54]:
```
Text(0.5, 1.0, 'Relationship between Gender and Insurance Charges based on
Smoking habits')
```



Female are coded with "0" and Male with "1"

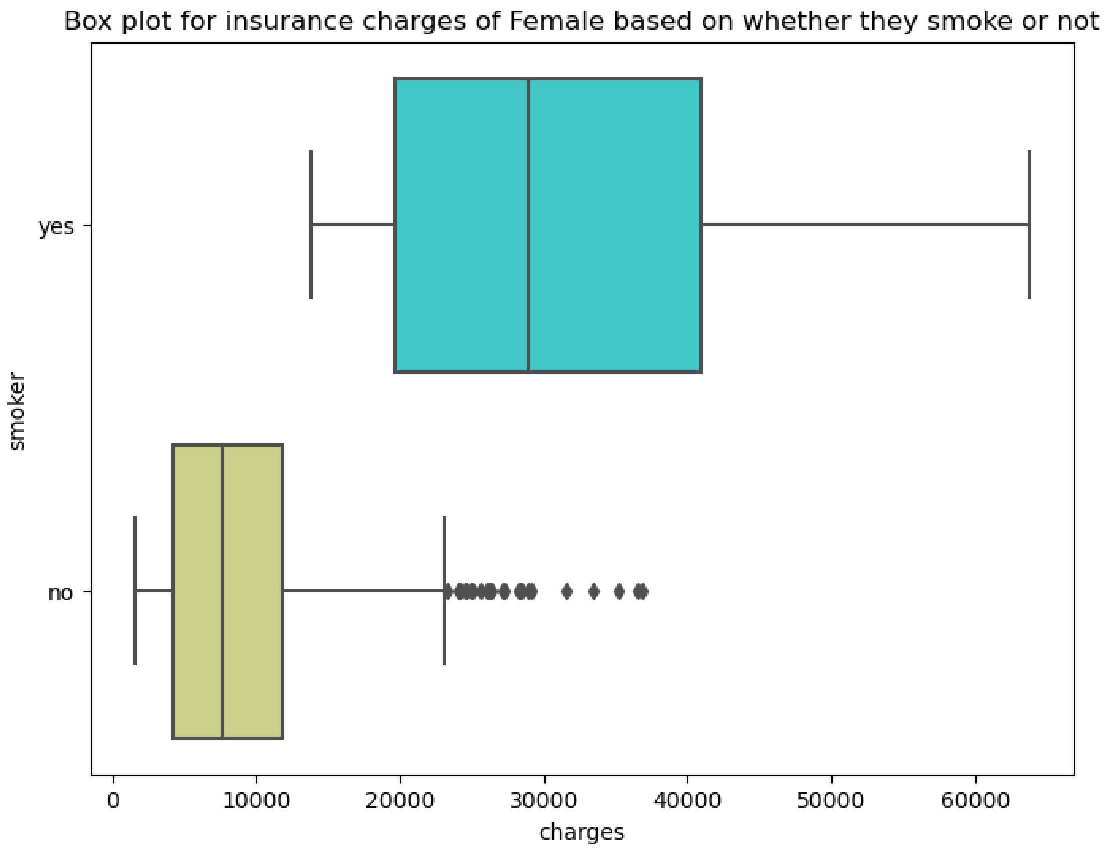
[55]:
```
plt.figure(figsize=(8,6))
plt.title("Box plot for insurance charges of Male based on whether they smoke␣
↪or not")
sns.boxplot(y="smoker", x="charges", data =  data[(data.sex == 'male')] ,␣
↪orient="h", palette = 'magma')
```

[55]:
```
<Axes: title={'center': 'Box plot for insurance charges of Male based on whether
they smoke or not'}, xlabel='charges', ylabel='smoker'>
```



[56]:
```
plt.figure(figsize=(8,6))
plt.title("Box plot for insurance charges of Female based on whether they smoke␣
 ↪or not")
sns.boxplot(y="smoker", x="charges", data =  data[(data.sex == 'female')] ,␣
 ↪orient="h", palette = 'rainbow')
```

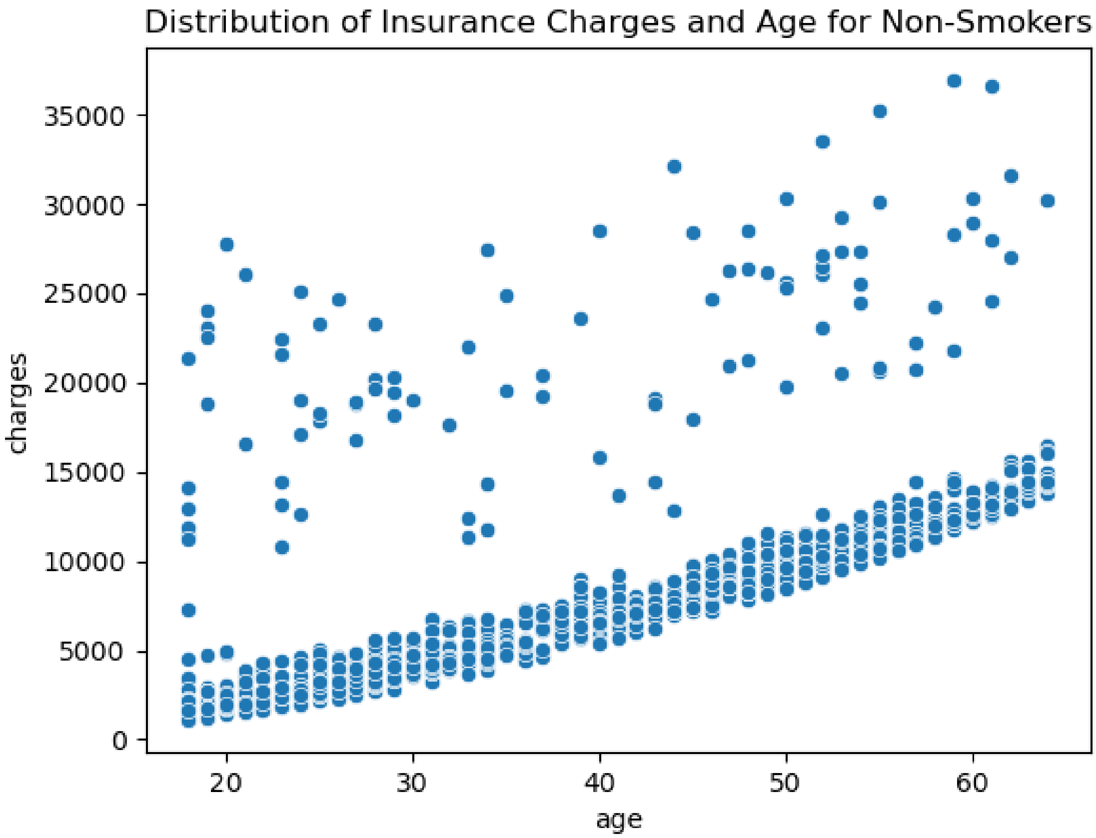
[56]:
```
<Axes: title={'center': 'Box plot for insurance charges of Female based on
whether they smoke or not'}, xlabel='charges', ylabel='smoker'>
```



Now let's see how the cost of treatment depends on the age of smokers and non-smokers patients.

[57]:
```
#Non smokers
sns.scatterplot(x=insurance[(insurance.smoker == 0)].age,y=insurance[(insurance.
 ↪smoker == 0)].charges,palette='Set1').set(title='Distribution of Insurance␣
 ↪Charges and Age for Non-Smokers')
```

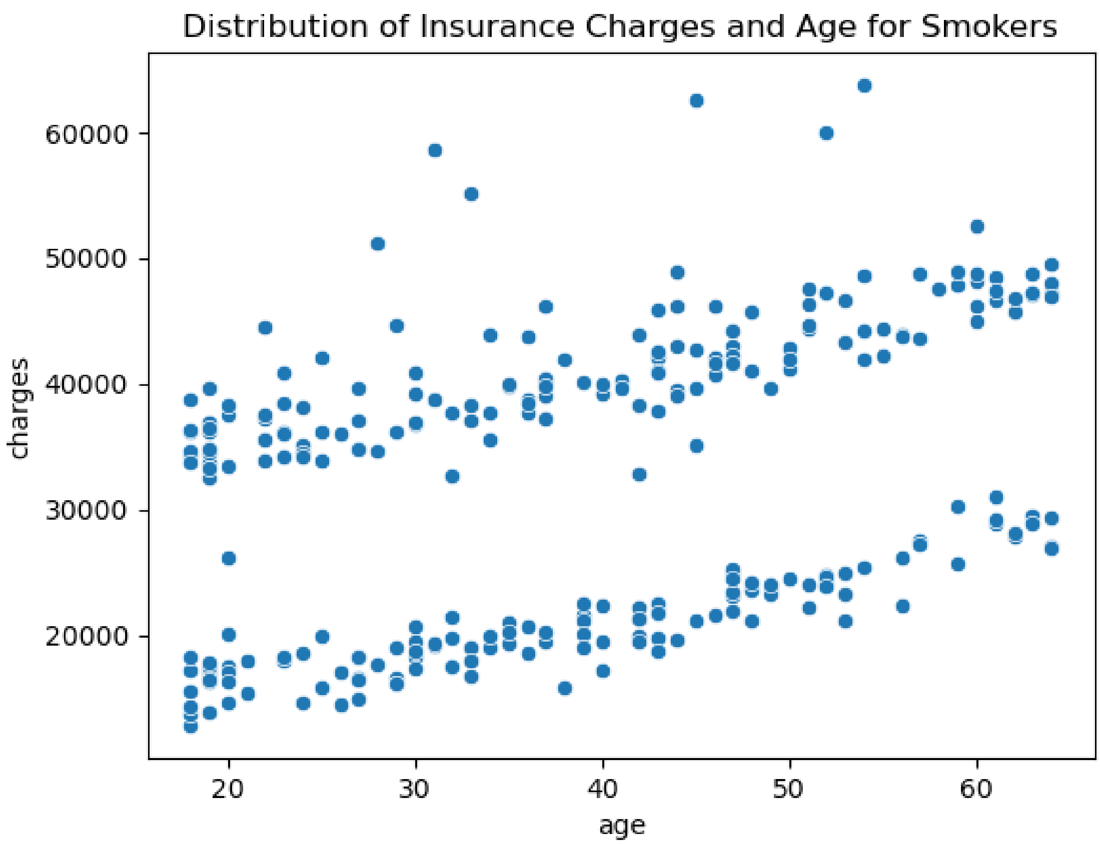
[57]: [Text(0.5, 1.0, 'Distribution of Insurance Charges and Age for Non-Smokers')]



Distribution of insurance charges and age for non-smokers

[58]:
```
#Smokers
sns.scatterplot(x=insurance[(insurance.smoker == 1)].age,y=insurance[(insurance.
    ↪smoker == 1)].charges,palette='Set1').set(title='Distribution of Insurance␣
    ↪Charges and Age for Smokers')
```

[58]: [Text(0.5, 1.0, 'Distribution of Insurance Charges and Age for Smokers')]



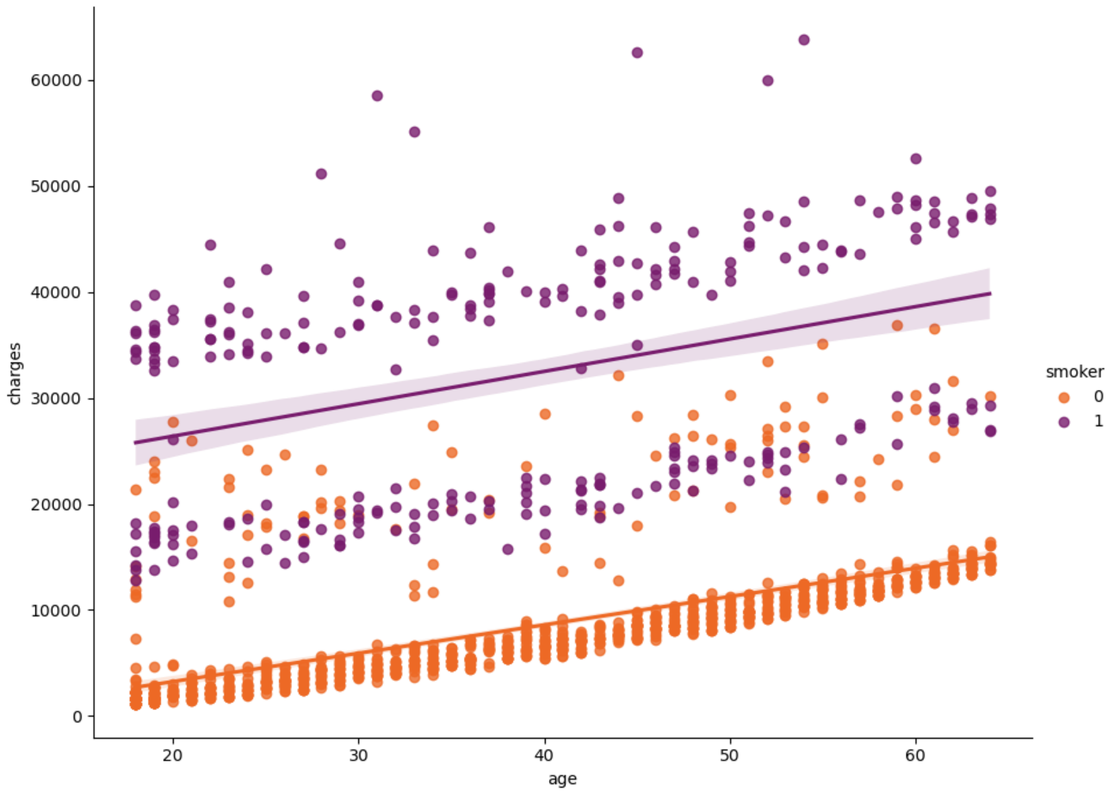
Distribution of insuarnce charges and age for smokers. Again it can be seen non-smokers, no matter how old they are, pay more charges

The Linear regression plot between Age and Insurance Charges based on Smoking Habits with linear fit line.

[59]:
```
plt.figure(figsize=(8,6))
sns.lmplot(x="age", y="charges", hue="smoker", data=insurance, palette =␣
  ↪'inferno_r', height=7, aspect=1.3)
ax.set_title('Smokers and non-smokers')
```

[59]:
```
Text(0.5, 1.0, 'Smokers and non-smokers')
```

```
<Figure size 800x600 with 0 Axes>
```

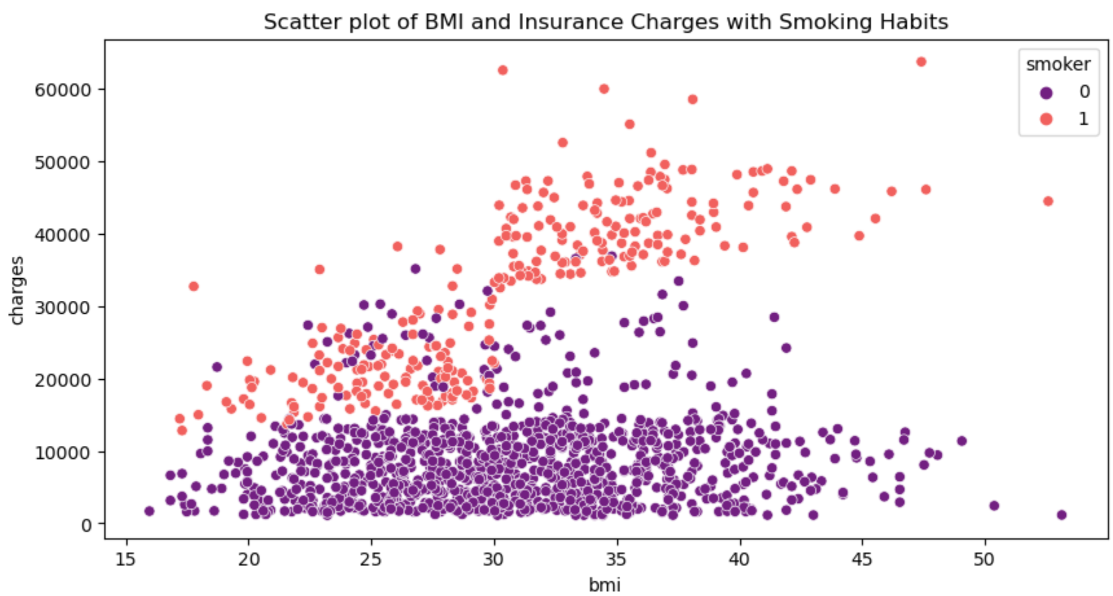


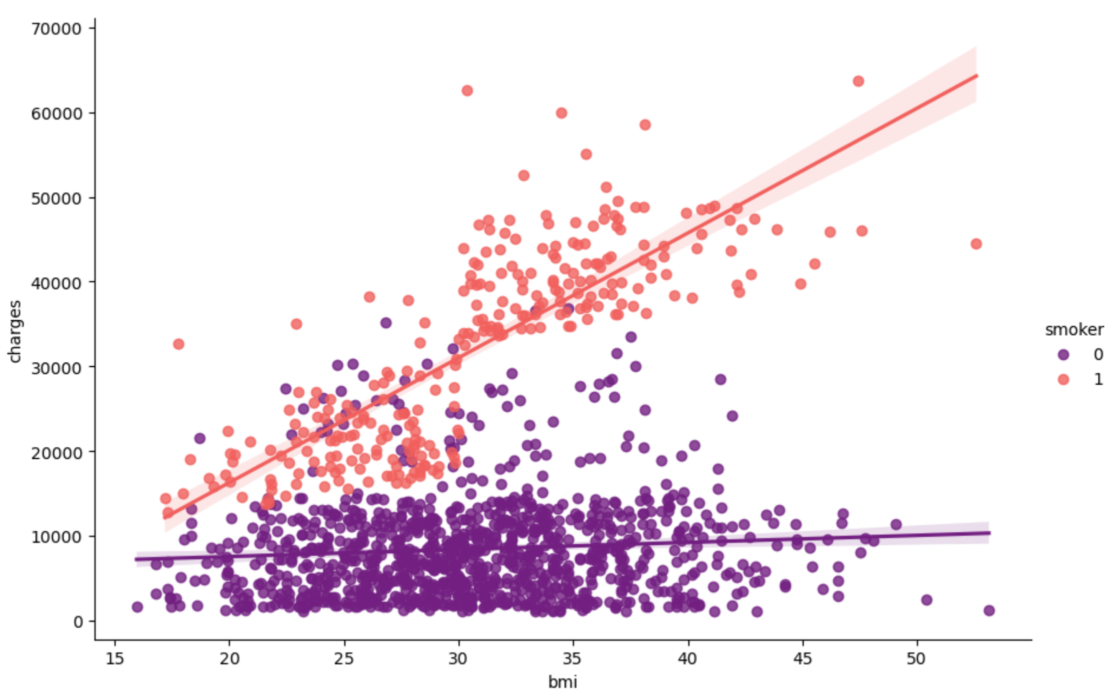
Here non-smokers are coded as "0" and smokers are coded as "1".The Linear regression plot with linear fit line clearly shows smokers are paying way more insurance charges than non smokers. The relationship between Charges and Age isn't a very strong one. It has some effect but that's not massive.

Scatter plot between BMI and Insurance Charges based on Smoking Habits. The Linear regression plot between BMI and Insurance Charges based on Smoking Habits with linear fit line.

[60]:
```
plt.figure(figsize=(10,5))
ax = sns.scatterplot(x='bmi',y='charges', data=insurance, palette='magma',␣
 ↪hue='smoker')
ax.set_title('Scatter plot of BMI and Insurance Charges with Smoking Habits')

sns.lmplot(x="bmi", y="charges", hue="smoker", data=insurance, palette =␣
 ↪'magma', height=6, aspect=1.5)
```

[60]:
```
<seaborn.axisgrid.FacetGrid at 0x14743f610>
```



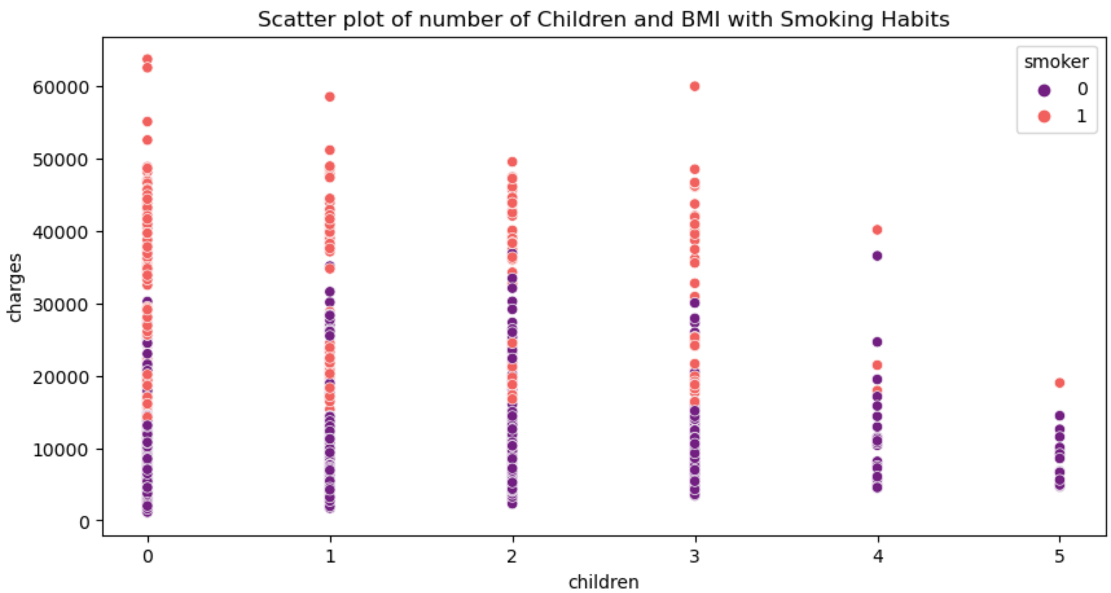


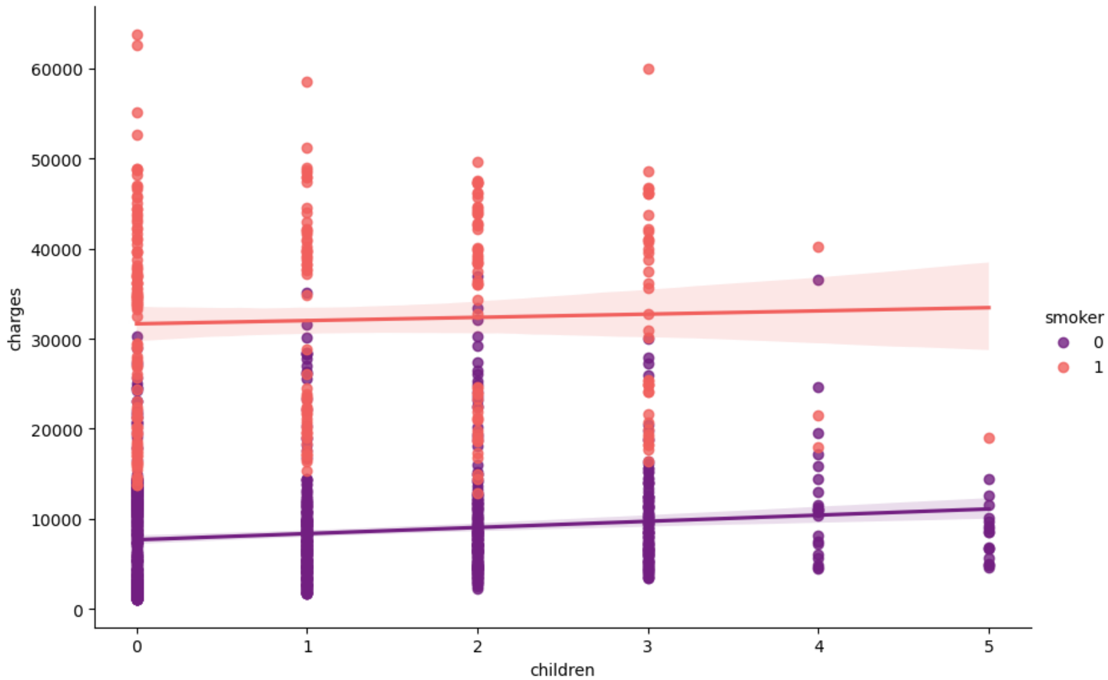
For smokers, increase in BMI, results in increase in Insurance charges but for non-smokers, there isn't much relationship between BMI and charges.

Scatter plot between number of Children and Insurance Charges based on Smoking Habits. The Linear regression plot between number of Children and Insurance Charges based on Smoking Habits with linear fit line.

[61]:
```
plt.figure(figsize=(10,5))
ax = sns.scatterplot(x='children',y='charges', data=insurance, palette='magma',␣
 ↪hue='smoker')
ax.set_title('Scatter plot of number of Children and BMI with Smoking Habits')

sns.lmplot(x="children", y="charges", hue="smoker", data=insurance, palette =␣
 ↪'magma', height=6, aspect=1.5)
```

[61]: <seaborn.axisgrid.FacetGrid at 0x147460fd0>



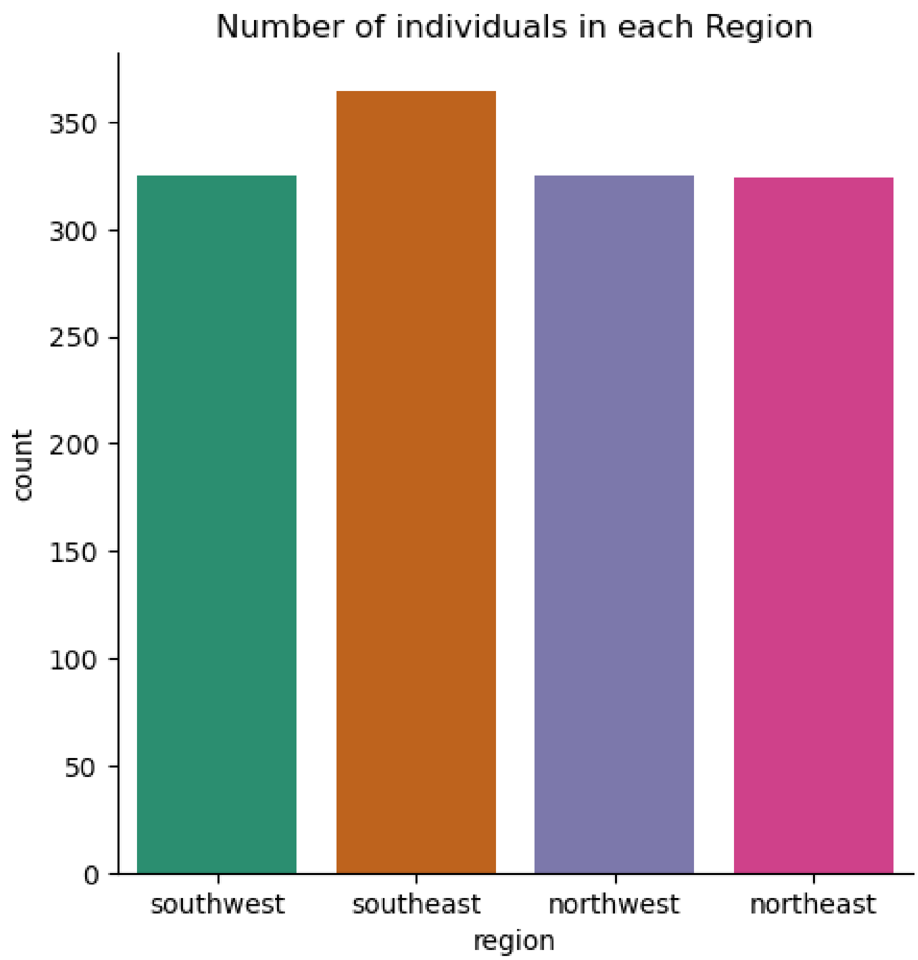


For smokers, we can see number of children doesn't play any role on deciding Charges. Its high anyways. But for non-smokers, charges are lower than smokers but then number of children has a slight effect on deciding Charges.

Now to investigate Region Feature:

[62]:
```
#Region
sns.catplot(x="region", kind="count", palette="Dark2",data=data)
plt.title("Number of individuals in each Region")
```

[62]: Text(0.5, 1.0, 'Number of individuals in each Region')

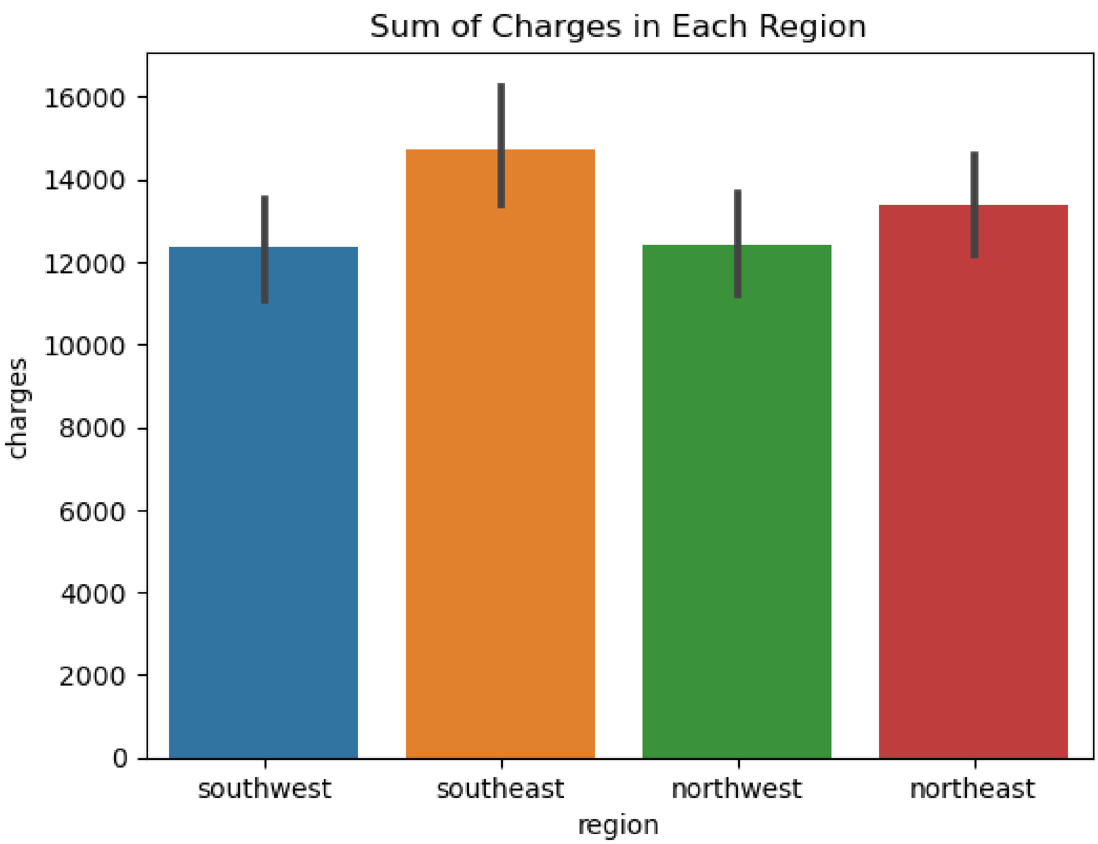


Count of individuals (in the dataset) in each region. Maximum number of individuals are from southeast region.

Barplot between Insurance charges and Regions.

[63]:
```
sns.barplot(data=data, x='region', y='charges')
plt.title("Sum of Charges in Each Region")
```

[63]: Text(0.5, 1.0, 'Sum of Charges in Each Region')

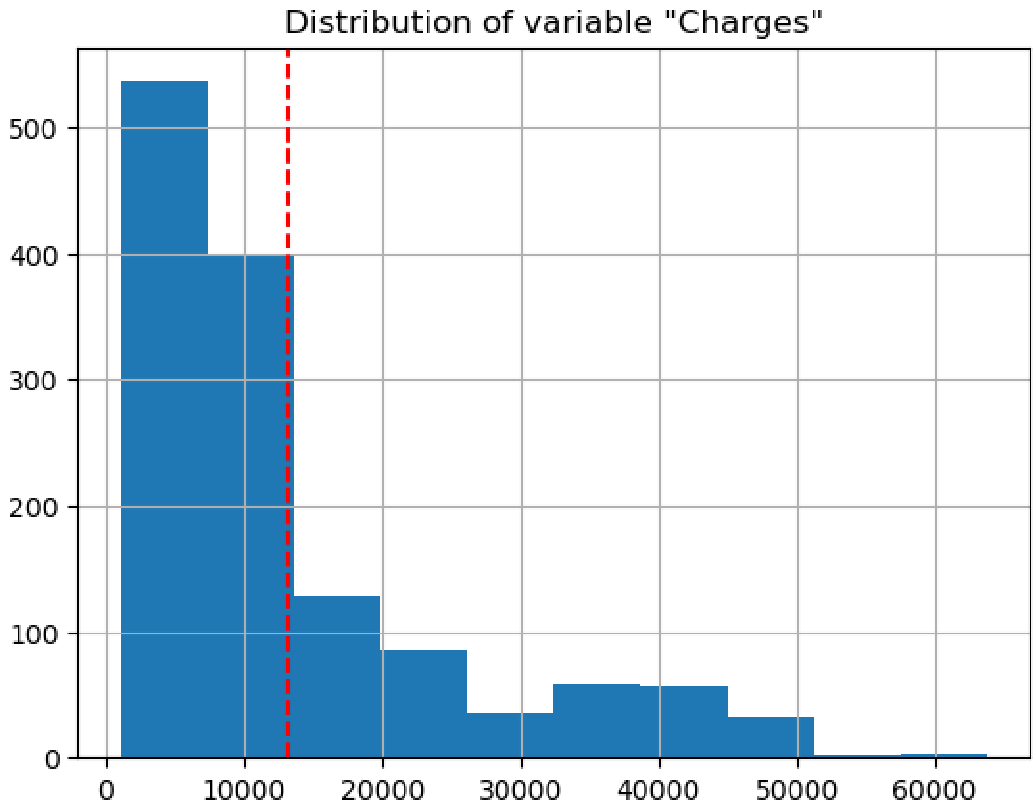


Sum of Insurance charges individuals are paying in each region. Again, people in Southeast region are paying the most.

Now to examine target variable, Charges:

[64]:
```
data['charges'].hist();
plt.title('Distribution of variable "Charges"')
plt.axvline(charges_m['mean'], linestyle = '--', color = "red")
```

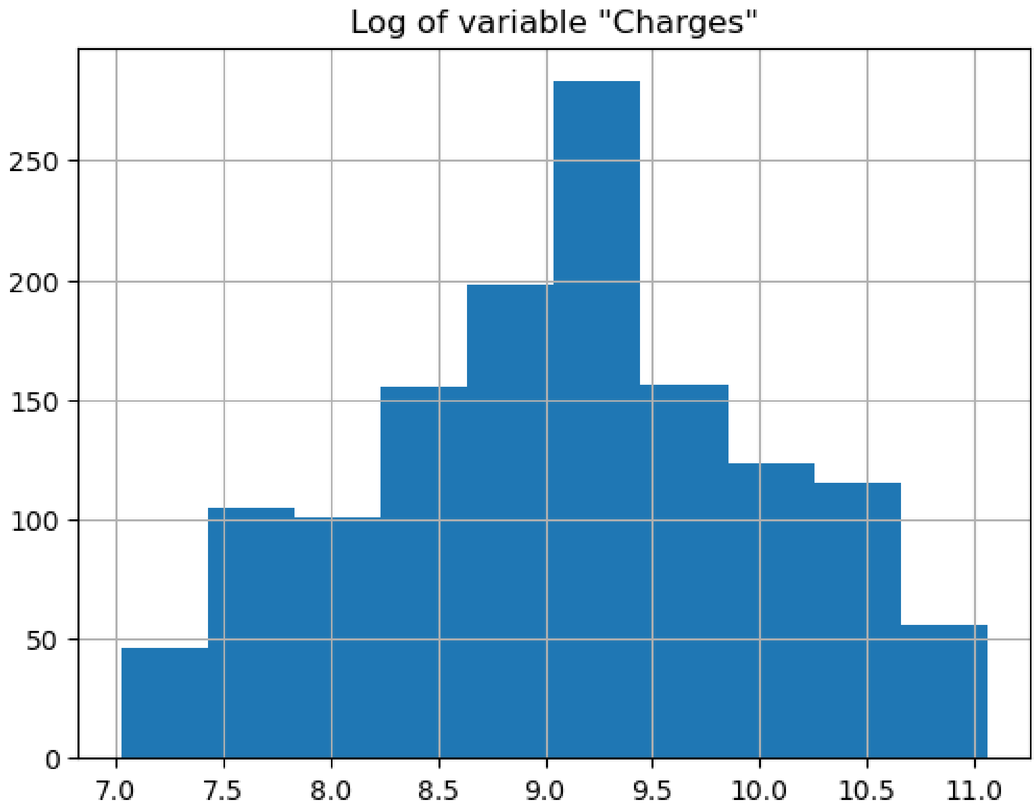
[64]:
```
<matplotlib.lines.Line2D at 0x14742a6d0>
```



This distribution is right-skewed. To make it closer to normal we can apply natural log.

[65]:
```
(np.log(insurance['charges'])).hist();
plt.title('Log of variable "Charges"')
```

[65]: Text(0.5, 1.0, 'Log of variable "Charges"')



It's better now. As distribution of target variable matters a lot in regression models.

[66]:
```
numeric_features = ['bmi', 'age', 'children']
transformer = make_column_transformer((StandardScaler(), numeric_features),
                                      remainder = 'passthrough')
```

Since smoker has the highest score, want to start with basic model with only smoker as input feature. split the data in X and y.

[67]:
```
def make_score(model, Xtrain, Xtest, ytrain, ytest):

    train_MSE = mean_squared_error(ytrain, model.predict(Xtrain), squared=False)
    test_MSE = mean_squared_error(ytest, model.predict(Xtest), squared=False)
    train_R = model.score(Xtrain, ytrain)
    test_R = model.score(Xtest, ytest)

    output=pd.DataFrame(["%.3f" %train_MSE, "%.3f" %test_MSE, "%.3f" %train_R,␣
 ↪"%.3f" %test_R], index=['Training data MSE:', 'Testing data MSE:', 'Training␣
 ↪data R^2 score:', 'Testing data R^2 score:'], columns=['Score'])
    return output
```

[68]:
```python
def cv_make_score(model,Xtrain, Xtest, ytrain, ytest):
    MSE_train=np.sqrt(np.abs(cross_val_score(model, Xtrain, ytrain, cv=5, scoring='neg_mean_squared_error')))
    print(f"MSE on train data with CV=5:",MSE_train)
    MSE_test=np.sqrt(np.abs(cross_val_score(model, Xtest, ytest, cv=5, scoring='neg_mean_squared_error')))
    print(f"MSE on test data with CV=5:",MSE_test)
    model_train_score= cross_val_score(model, Xtrain, ytrain, cv=5)
    model_test_score= cross_val_score(model, Xtest, ytest, cv=5)
    print(f"R^2 on training data is:", model_train_score)
    print(f"R^2 on test data is:", model_test_score)

    output=pd.DataFrame(["%.3f" %MSE_train.mean(), "%.3f" %MSE_test.mean(), "%.3f" %model_train_score.mean(), "%.3f" %model_test_score.mean()], index=['Training data MSE (mean)', 'Testing data MSE (mean)', 'Training data R^2 score (mean)', 'Testing data R^2 score(mean)'], columns=['Score'])
    return output
```

[69]:
```python
X1= insurance[['smoker']]
y1= insurance['charges']

X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2,random_state=22)

lm=LinearRegression().fit(X1_train, y1_train)
output=make_score(lm, X1_train, X1_test, y1_train, y1_test)
output
```

[69]:

	Score
Training data MSE:	7539.347
Testing data MSE:	7170.328
Training data R^2 score:	0.612
Testing data R^2 score:	0.651

Now with three most important input features without label encoding: Smoker, age, BMI

[70]:
```python
X2= insurance[['smoker', 'age', 'bmi']]
y2= insurance['charges']

X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2,random_state=22)

lm.fit(X2_train, y2_train)
output=make_score(lm, X2_train, X2_test, y2_train, y2_test)
output
```

[70]:
```
                          Score
Training data MSE:      6100.435
Testing data MSE:       6027.434
Training data R^2 score:   0.746
Testing data R^2 score:    0.753
```

[71]:
```
X = insurance.drop(['charges'], axis = 1)
y = insurance['charges']
```

Baseline Model

[72]:
```
baseline_preds=np.ones(len(y))*y.mean()
baseline_preds
```

[72]:
```
array([13270.42226514, 13270.42226514, 13270.42226514, …,
       13270.42226514, 13270.42226514, 13270.42226514])
```

[73]:
```
print("MSE of Baseline model is:", "%.3f" %mean_squared_error(y,␣
 ↪baseline_preds, squared=False))
```

```
MSE of Baseline model is: 12105.485
```

Mean of target variable

[74]:
```
print("Mean of target variable is:", "%.3f" %y.mean())
```

```
Mean of target variable is: 13270.422
```

Split the data into training and test data

[75]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,random_state=22)
```

Now to determine the baseline performance for a Regression model using the DummyRegressor on the training data. and finding the baseline accuracy? Assigning the baseline training accuracy as a float to baseline. First we will consider R^2 since thats the default value. Then mean and median.

Linear Regression model:

[76]:
```
print("Statistics of Linear Regression model: ")
lm.fit(X_train, y_train)
output=make_score(lm, X_train, X_test, y_train, y_test)
output
```

```
Statistics of Linear Regression model:
```

[76]:
```
                          Score
Training data MSE:      6066.321
Testing data MSE:       5970.445
Training data R^2 score:   0.749
Testing data R^2 score:    0.758
```

Linear Regression with Cross validation:

```
[77]: print("Statistics of Linear Regression model with cv=5: ")
output=cv_make_score(lm, X_train, X_test, y_train, y_test)
output
```

```
Statistics of Linear Regression model with cv=5:
MSE on train data with CV=5: [6736.18319773 6327.56648077 5156.93047458
5831.85349125 6476.04603912]
MSE on test data with CV=5: [4519.886668   7182.28299889 5984.55629623
6110.8285228  6219.77518775]
R^2 on training data is: [0.75408355 0.7172788  0.77803097 0.71799588
0.73245679]
R^2 on test data is: [0.83348179 0.74312312 0.67324177 0.70655995 0.77226494]
```

[77]:

	Score
Training data MSE (mean)	6105.716
Testing data MSE (mean)	6003.466
Training data R^2 score (mean)	0.740
Testing data R^2 score(mean)	0.746

Ridge Regression model:

```
[78]: print("Statistics of Ridge model: ")
ridge = Ridge(alpha = 0.5)
ridge.fit(X_train, y_train)
output=make_score(ridge, X_train, X_test, y_train, y_test)
output
```

```
Statistics of Ridge model:
```

[78]:

	Score
Training data MSE:	6066.382
Testing data MSE:	5973.011
Training data R^2 score:	0.749
Testing data R^2 score:	0.758

Ridge Model with CV=5

```
[79]: print("Statistics of Ridge model with cv=5: ")
output=cv_make_score(ridge, X_train, X_test, y_train, y_test)
output
```

```
Statistics of Ridge model with cv=5:
MSE on train data with CV=5: [6744.07870001 6326.52802376 5157.71014495
5825.7691689  6473.70575712]
MSE on test data with CV=5: [4513.98378746 7213.70210452 5979.98577267
6113.39434921 6208.60273622]
R^2 on training data is: [0.75350673 0.71737159 0.77796384 0.718584
0.73265012]
```

```
R^2 on test data is: [0.83391644 0.74087078 0.67374068 0.70631348 0.77308236]
```

[79]:

	Score
Training data MSE (mean)	6105.558
Testing data MSE (mean)	6005.934
Training data R^2 score (mean)	0.740
Testing data R^2 score(mean)	0.746

Polynomial Features

[80]:
```
X = insurance.drop(['charges'], axis = 1)
y = insurance['charges']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪2,random_state=22)
```

[81]:
```
pipe = Pipeline(steps=[('poly', PolynomialFeatures()),
                        ('classifier', lm)])
pipe.fit(X_train, y_train)
```

[81]:
```
Pipeline(steps=[('poly', PolynomialFeatures()),
                ('classifier', LinearRegression())])
```

[82]:
```
print("Statistics of Polynomial Features model: ")
output=make_score(pipe, X_train, X_test, y_train, y_test)
output
```

```
Statistics of Polynomial Features model:
```

[82]:

	Score
Training data MSE:	4706.201
Testing data MSE:	4967.868
Training data R^2 score:	0.849
Testing data R^2 score:	0.832

[83]:
```
print("Statistics of Polynomial Features model with cv=5: ")
output=cv_make_score(pipe, X_train, X_test, y_train, y_test)
output
```

```
Statistics of Polynomial Features model with cv=5:
MSE on train data with CV=5: [5409.22917587 5070.66210432 3724.63214149
4520.8181841  5113.975651  ]
MSE on test data with CV=5: [3370.8674516  6260.84937976 5500.9112084
5348.6900154  5580.75922157]
R^2 on training data is: [0.84142652 0.81844246 0.88420852 0.83053644
0.83316338]
R^2 on test data is: [0.90738315 0.80480599 0.72392194 0.77519082 0.81665584]
```

[83]:

	Score
Training data MSE (mean)	4767.863

```
Testing data MSE (mean)         5212.415
Training data R^2 score (mean)     0.842
Testing data R^2 score(mean)       0.806
```

Grid Search with Polynomial Features

[84]:
```python
params = {'poly__degree': [1,2,3,4,5,6,7,10]}
grid=GridSearchCV(pipe, param_grid=params, cv=5,␣
↪scoring='neg_mean_squared_error')
grid.fit(X_train, y_train)
grid.best_estimator_
print("Best Estimator:", grid.best_params_)
print("Best score:", np.sqrt(np.abs(grid.best_score_)))
pd.DataFrame(grid.cv_results_)
```

```
Best Estimator: {'poly__degree': 2}
Best score: 4804.889240253629
```

[84]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	\
0	0.002049	0.000885	0.000721	0.000304	
1	0.003262	0.000925	0.001015	0.000407	
2	0.005221	0.001166	0.001121	0.000212	
3	0.013764	0.002730	0.000926	0.000540	
4	0.051013	0.001500	0.001283	0.000645	
5	0.310532	0.234941	0.003083	0.001838	
6	0.286364	0.018280	0.002321	0.000912	
7	0.932578	0.087304	0.004889	0.001034	

	param_poly__degree	params	split0_test_score	\
0	1	{'poly__degree': 1}	-4.537616e+07	
1	2	{'poly__degree': 2}	-2.925976e+07	
2	3	{'poly__degree': 3}	-3.183980e+07	
3	4	{'poly__degree': 4}	-3.335502e+07	
4	5	{'poly__degree': 5}	-7.430026e+07	
5	6	{'poly__degree': 6}	-8.748958e+11	
6	7	{'poly__degree': 7}	-1.245347e+14	
7	10	{'poly__degree': 10}	-2.691355e+14	

	split1_test_score	split2_test_score	split3_test_score	split4_test_score	\
0	-4.003810e+07	-2.659393e+07	-3.401052e+07	-4.193917e+07	
1	-2.571161e+07	-1.387288e+07	-2.043780e+07	-2.615275e+07	
2	-3.043526e+07	-1.616803e+07	-2.220704e+07	-2.738485e+07	
3	-2.949647e+07	-2.464087e+07	-2.850669e+07	-3.296450e+07	
4	-1.048492e+08	-1.181055e+08	-3.872253e+08	-7.280967e+07	
5	-4.951641e+11	-2.597717e+10	-7.461261e+13	-7.790638e+09	
6	-3.696724e+12	-2.327206e+13	-3.632950e+12	-4.935321e+12	
7	-6.515160e+14	-8.668807e+13	-4.279663e+14	-1.498668e+14	

```
   mean_test_score  std_test_score  rank_test_score
0    -3.759158e+07    6.622070e+06                4
1    -2.308696e+07    5.409410e+06                1
2    -2.560700e+07    5.761501e+06                2
3    -2.979271e+07    3.194789e+06                3
4    -1.514580e+08    1.191720e+08                5
5    -1.520329e+13    2.970641e+13                6
6    -3.201435e+13    4.685532e+13                7
7    -3.170345e+14    2.037069e+14                8
```

Now let's take that grid model and create some predictions using the test set and create regression matrices for them. Let‘s inspect the differences in a DataFrame. First, we concatenate the true and predicted y:

[85]:
```
grid_pred = grid.predict(X_test)
y_valid = y_test.copy()
gg = []
for i in grid_pred:
 gg.append(i)
gf = pd.DataFrame(data=gg)
gf = gf.set_index(y_valid.index)
gf.rename(columns={0: "predicted"}, inplace=True)
df321 = pd.concat([y_valid, gf], axis=1)
df321.columns = ["Y_true", "Y_pred"]
df321.head()
```

[85]:
```
           Y_true         Y_pred
1231  20167.33603  13896.259766
768   14319.03100  14937.904053
847    2438.05520   1779.103271
510   11763.00090  13720.580811
363    2597.77900   3775.863770
```

And compute the difference score, which will show us if more values are positive or negative:

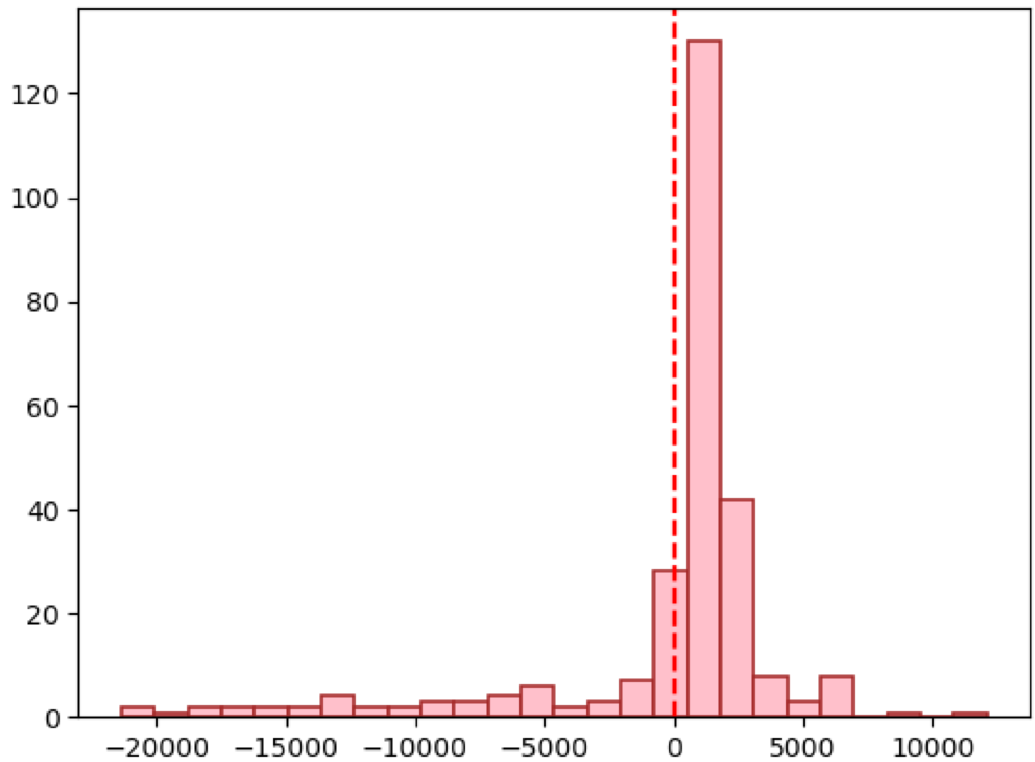
[86]:
```
df321["diff"] = df321["Y_pred"]- df321["Y_true"]
df321.head()
```

[86]:
```
           Y_true         Y_pred         diff
1231  20167.33603  13896.259766 -6271.076264
768   14319.03100  14937.904053   618.873053
847    2438.05520   1779.103271  -658.951929
510   11763.00090  13720.580811  1957.579911
363    2597.77900   3775.863770  1178.084770
```

There is a tendency for the y_true values being overestimated, as we can see in the histogram:

[87]:
```
plt.hist(df321["diff"], bins=26, color="pink", edgecolor="brown", linewidth=1.2)
plt.axvline(0, color="red", linestyle="dashed", linewidth=1.6)
```

```
plt.show()
```



[88]:
```
pd.DataFrame({"Count": [(df321["diff"]<0).sum(),
                        (df321["diff"]==0).sum(),
                        (df321["diff"]>0).sum()]},
columns=["Count"],index=["Underestimation", "Exact Estimation",␣
 ↪"Overestimation"])
```

[88]:

	Count
Underestimation	61
Exact Estimation	0
Overestimation	207

[89]:
```
pipe = Pipeline(steps=[('poly', PolynomialFeatures(degree=2)),
                        ('classifier', lm)])
print("Statistics of Polynomial Features model with cv=5 and degree=2: ")
output=cv_make_score(pipe, X_train,X_test, y_train, y_test)
output
```

```
Statistics of Polynomial Features model with cv=5 and degree=2:
MSE on train data with CV=5: [5409.22917587 5070.66210432 3724.63214149
4520.8181841  5113.975651  ]
MSE on test data with CV=5: [3370.8674516  6260.84937976 5500.9112084
```

```
5348.6900154  5580.75922157]
R^2 on training data is: [0.84142652 0.81844246 0.88420852 0.83053644
0.83316338]
R^2 on test data is: [0.90738315 0.80480599 0.72392194 0.77519082 0.81665584]
```

[89]:

	Score
Training data MSE (mean)	4767.863
Testing data MSE (mean)	5212.415
Training data R^2 score (mean)	0.842
Testing data R^2 score(mean)	0.806

Now get rid of unwanted features

[90]:
```
XR = insurance.drop(['charges', 'sex', 'region', 'children'], axis = 1)
yr = insurance.charges
```

Split the data

[91]:
```
XR_train, XR_test, yr_train, yr_test = train_test_split(XR, yr, test_size=0.2,␣
↪random_state=0)
```

[92]:
```
pipe = Pipeline(steps=[('poly', PolynomialFeatures(degree=2)),
                       ('classifier', lm)])

pipe.fit(XR_train, yr_train)
print("Statistics of Polynomial Features model with cv=5 and reduced features:␣
↪")
MSE_train=np.sqrt(np.abs(cross_val_score(pipe, XR_train, yr_train, cv=5,␣
↪scoring='neg_mean_squared_error')))
print(f"MSE on train data with CV=5:",MSE_train)
MSE_test=np.sqrt(np.abs(cross_val_score(pipe, XR_test, yr_test, cv=5,␣
↪scoring='neg_mean_squared_error')))
print(f"MSE on test data with CV=5:",MSE_test)
pipe_train_score= cross_val_score(pipe, XR_train, yr_train, cv=5)
pipe_test_score= cross_val_score(pipe, XR_test, yr_test, cv=5)
print(f"R^2 on training data is:", pipe_train_score)
print(f"R^2 on test data is:", pipe_test_score)

output=pd.DataFrame(["%.3f" %MSE_train.mean(), "%.3f" %MSE_test.mean(), "%.3f"␣
↪%pipe_train_score.mean(), "%.3f" %pipe_test_score.mean()], index=['Training␣
↪data MSE (mean):', 'Testing data MSE (mean):', 'Training data R^2 score␣
↪(mean):', 'Testing data R^2 score(mean):'], columns=['Score'])
output
```

```
Statistics of Polynomial Features model with cv=5 and reduced features:
MSE on train data with CV=5: [4723.56420002 4476.67379871 4817.00776247
4730.95826359 6108.28377529]
MSE on test data with CV=5: [5410.98344297 4213.36288184 4185.53104258
5224.21128805 2950.59728691]
```

```
R^2 on training data is: [0.85812025 0.82669669 0.82369919 0.84713261
0.76900463]
R^2 on test data is: [0.84212914 0.90315876 0.89675216 0.77079484 0.9264583 ]
```

[92]:

	Score
Training data MSE (mean):	4971.298
Testing data MSE (mean):	4396.937
Training data R^2 score (mean):	0.825
Testing data R^2 score(mean):	0.868

PCA

[93]:
```
#No need for this
poly= PolynomialFeatures(degree=7)
XP= poly.fit_transform(X)
print(XP.shape)
```

```
(1338, 1716)
```

[94]:
```
X = insurance.drop(['charges'], axis = 1)
y = insurance['charges']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,random_state=22)
pca = PCA(n_components=3)
pipe2 = Pipeline([('poly', PolynomialFeatures()),
                        ('pca', PCA()),
                        ('model', lm)])
params2= {'pca__n_components': [1,3,5,6,7,10,100]}
grid2=GridSearchCV(pipe2, param_grid= params2, cv=5,␣
 ↪scoring='neg_mean_squared_error')
grid2.fit(X_train,y_train)
print(grid2.best_estimator_)
print("Best Estimator:", grid2.best_params_)
print("Best score:", "%.3f" %np.sqrt(np.abs(grid2.best_score_)))
pd.DataFrame(grid2.cv_results_)
```

```
Pipeline(steps=[('poly', PolynomialFeatures()), ('pca', PCA(n_components=10)),
                ('model', LinearRegression())])
Best Estimator: {'pca__n_components': 10}
Best score: 5959.234
```

[94]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	\
0	0.012165	0.006288	0.001293	0.000575	
1	0.004204	0.002306	0.000743	0.000387	
2	0.003811	0.001149	0.000673	0.000310	
3	0.004763	0.002128	0.000830	0.000406	
4	0.004912	0.001786	0.000492	0.000008	
5	0.006412	0.002353	0.000950	0.000543	
6	0.000936	0.000042	0.000000	0.000000	

```
  param_pca__n_components                        params  split0_test_score  \
0                       1    {'pca__n_components': 1}      -1.709222e+08
1                       3    {'pca__n_components': 3}      -1.661895e+08
2                       5    {'pca__n_components': 5}      -1.674235e+08
3                       6    {'pca__n_components': 6}      -1.640407e+08
4                       7    {'pca__n_components': 7}      -4.181227e+07
5                      10   {'pca__n_components': 10}      -4.066285e+07
6                     100  {'pca__n_components': 100}                NaN

   split1_test_score  split2_test_score  split3_test_score  split4_test_score  \
0      -1.253502e+08      -1.138632e+08      -1.082974e+08      -1.397581e+08
1      -1.222363e+08      -1.132787e+08      -1.115188e+08      -1.363509e+08
2      -1.228862e+08      -1.112412e+08      -1.088183e+08      -1.363182e+08
3      -1.198299e+08      -1.092529e+08      -1.081125e+08      -1.335505e+08
4      -3.815612e+07      -2.753389e+07      -3.274483e+07      -3.993559e+07
5      -3.753670e+07      -2.795201e+07      -3.213371e+07      -3.927707e+07
6                NaN                NaN                NaN                NaN

   mean_test_score  std_test_score  rank_test_score
0    -1.316382e+08    2.240289e+07                6
1    -1.299149e+08    2.015642e+07                5
2    -1.293375e+08    2.139816e+07                4
3    -1.269573e+08    2.067858e+07                3
4    -3.603654e+07    5.218900e+06                2
5    -3.551247e+07    4.761781e+06                1
6              NaN             NaN                7
```

Since best PCA n_components comes out to be 10 from GridSearchCV, we can apply that on pipe2.

[95]:
```
pipe2 = Pipeline([('poly', PolynomialFeatures()),
                        ('pca', PCA(10)),
                        ('model', lm)])
pipe2_mse=np.sqrt(np.abs(cross_val_score(pipe2, X_train, y_train, cv=5,␣
 ↪scoring='neg_mean_squared_error')))
print("MSE of Polynomial Features model with PCA is:", pipe2_mse)
print("Average MSE of Polynomial Features model with PCA is:", "%.3f"␣
 ↪%pipe2_mse.mean())
```

```
MSE of Polynomial Features model with PCA is: [6376.74328173 6126.72021555
5286.96567859 5668.6605823  6267.14183332]
Average MSE of Polynomial Features model with PCA is: 5945.246
```

DecisionTree Regressor:

[96]:
```
XD = insurance.drop(['charges'], axis = 1)
yD = insurance['charges']
```

```
XD_train, XD_test, yD_train, yD_test = train_test_split(XD, yD, test_size=0.
 ↪2,random_state=22)
tree = DecisionTreeRegressor(max_depth=3, random_state=0)
tree.fit(XD_train, yD_train)
print("Statistics of Tree Regressor model with max_depth=3")
output=make_score(tree, XD_train, XD_test, yD_train, yD_test)
output

#y_pred = tree.predict(data_test)
```

```
Statistics of Tree Regressor model with max_depth=3
```

[96]:

	Score
Training data MSE:	4471.388
Testing data MSE:	5222.981
Training data R^2 score:	0.863
Testing data R^2 score:	0.815

[97]:
```
print("Statistics of Tree Regressor model with CV=5 and max_depth=3")
output=cv_make_score(tree, XD_train, XD_test, yD_train, yD_test )
output
```

```
Statistics of Tree Regressor model with CV=5 and max_depth=3
MSE on train data with CV=5: [4887.430102   5169.47611306 3508.05459993
4399.49619476 5076.54677497]
MSE on test data with CV=5: [3892.86162273 6429.27661473 4722.68756104
6077.6505657  4689.00227868]
R^2 on training data is: [0.87054438 0.81129735 0.89728295 0.83950994
0.83559657]
R^2 on test data is: [0.87647793 0.79416264 0.79651095 0.70973769 0.87056808]
```

[97]:

	Score
Training data MSE (mean)	4608.201
Testing data MSE (mean)	5162.296
Training data R^2 score (mean)	0.851
Testing data R^2 score(mean)	0.809

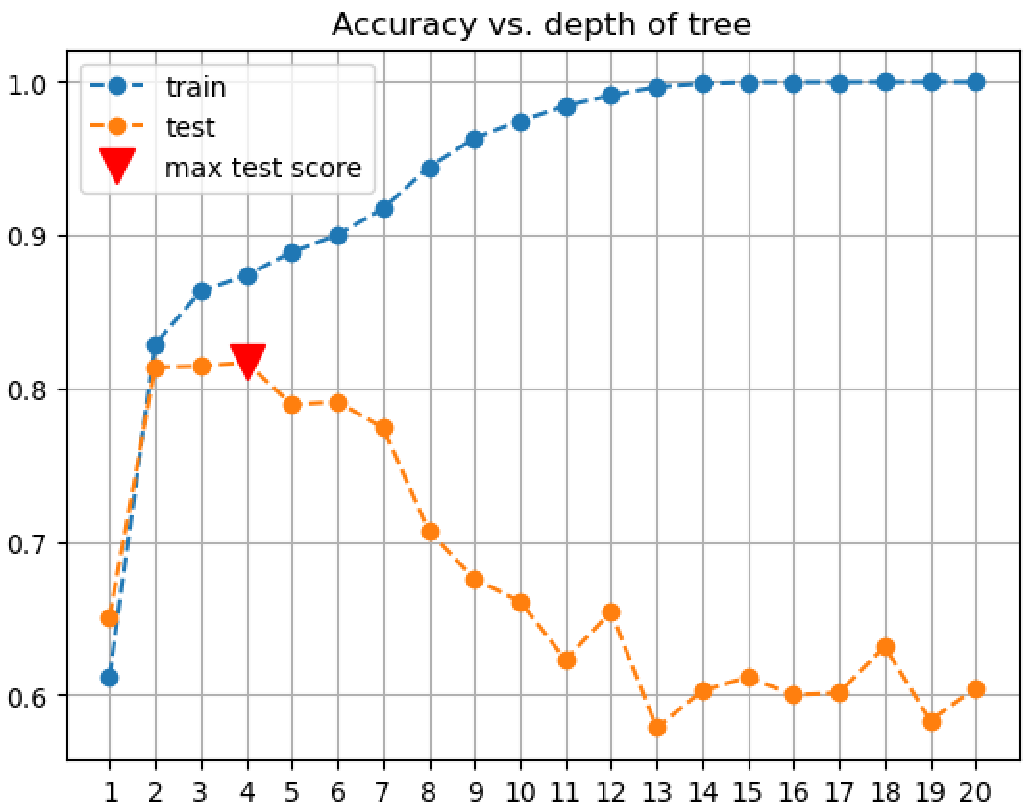
[98]:
```
depths = list(range(1, 21))
print(depths)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Fine tuning of Depth of tree:

[99]:
```
tree = DecisionTreeRegressor(random_state=0)
train_scores = []
test_scores = []
for depth in depths:
    tree=DecisionTreeRegressor(max_depth=depth).fit(XD_train, yD_train)
```

```
    train_scores.append(tree.score(XD_train, yD_train))
    test_scores.append(tree.score(XD_test, yD_test))

plt.plot(depths, train_scores, '--o', label = 'train')
plt.plot(depths, test_scores, '--o', label = 'test')
plt.title('Accuracy vs. depth of tree')
plt.xticks(depths)
plt.plot(np.argmax(test_scores)+1, max(test_scores),
         'v', markersize = 13, color = 'red',
        label = 'max test score')
plt.legend()
plt.grid();
```

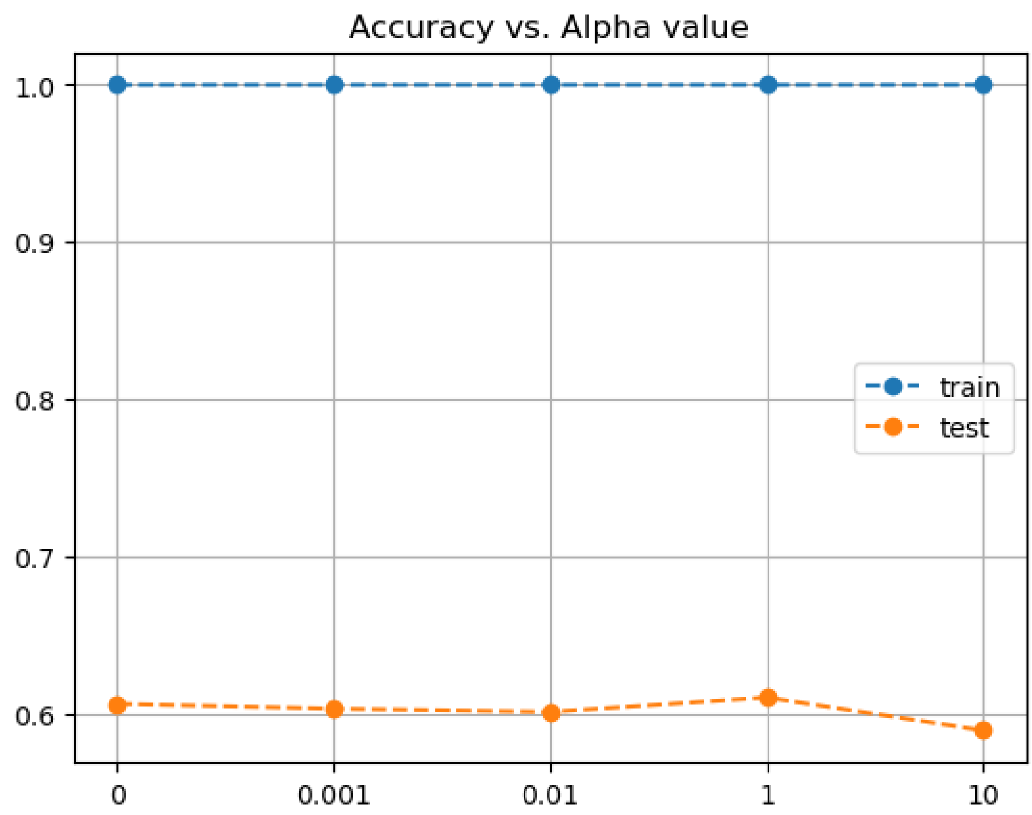


Fine tuning of alpha

[100]:
```
alphas = [0, 0.001, 0.01, 1, 10]
```

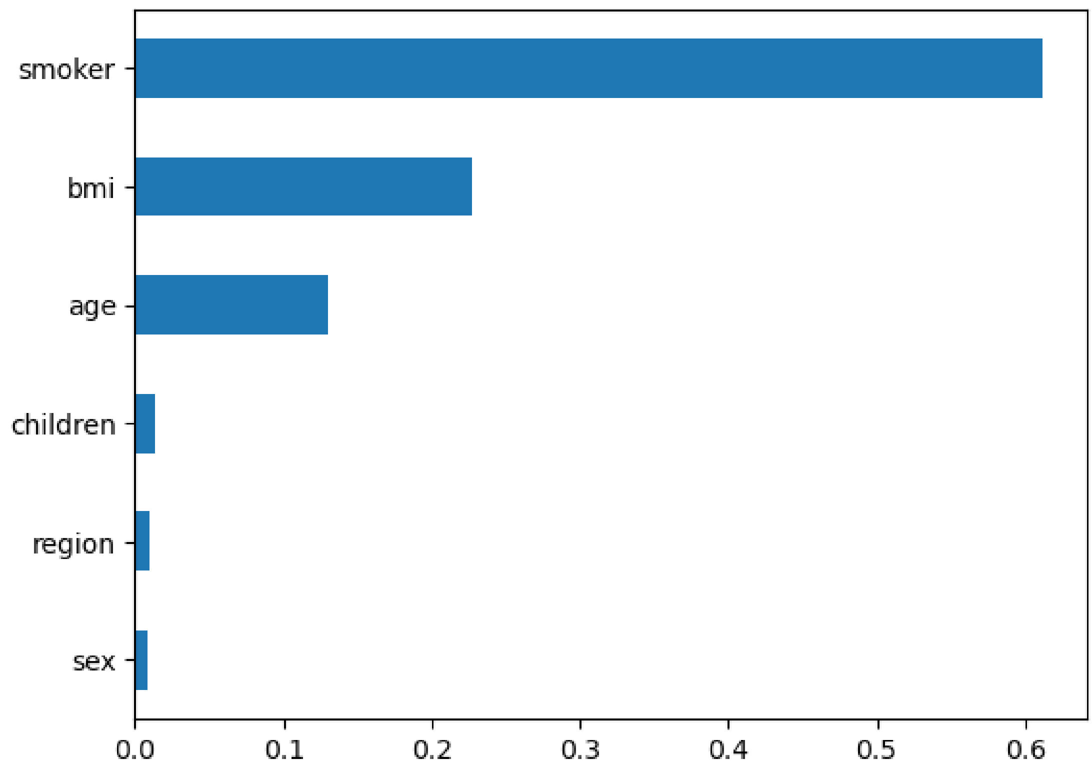
[101]:
```
ccp_train_scores = []
ccp_test_scores = []
for alpha in alphas:
    tree = DecisionTreeRegressor(ccp_alpha=alpha)
```

```
    tree.fit(XD_train, yD_train)
    ccp_train_scores.append(tree.score(XD_train, yD_train))
    ccp_test_scores.append(tree.score(XD_test, yD_test))

#Answer test
plt.plot(range(len(alphas)), ccp_train_scores, '--o', label = 'train')
plt.plot(range(len(alphas)), ccp_test_scores, '--o', label = 'test')
plt.title('Accuracy vs. Alpha value') #it should say accuracy vs ccp_alpha value
plt.xticks(range(len(alphas)), alphas)
plt.legend()
plt.grid();
```



Plotting Feature Importance

[102]:
```
tree = DecisionTreeRegressor(ccp_alpha=.001).fit(XD_train, yD_train)
importances = {k:v for k,v in zip(XD_train.columns, tree.feature_importances_)}
pd.Series(importances).sort_values().plot(kind = 'barh');
```



Now fitting Tree model again with max_depth=4

[103]:
```
tree1 = DecisionTreeRegressor(max_depth=4, random_state=0)
tree1.fit(XD_train, yD_train)
print("Statistics of DecisionTreeRegressor model with max_depth=4")
output=make_score(tree1, XD_train, XD_test, yD_train, yD_test)
output
```

```
Statistics of DecisionTreeRegressor model with max_depth=4
```

[103]:

	Score
Training data MSE:	4295.688
Testing data MSE:	5189.529
Training data R^2 score:	0.874
Testing data R^2 score:	0.817

[104]:
```
print("Statistics of DecisionTreeRegressor model with CV=5 and max_depth=4")
output=cv_make_score(tree1, XD_train, XD_test, yD_train, yD_test)
output
```

```
Statistics of DecisionTreeRegressor model with CV=5 and max_depth=4
MSE on train data with CV=5: [4892.90840631 5218.73855306 3452.56222579
4720.15400608 4993.86916687]
MSE on test data with CV=5: [4699.7970272  6548.12483861 5412.54886546
```

```
6607.11499429 4983.16004285]
R^2 on training data is: [0.870254   0.80768373 0.90050692 0.8152627
0.84090798]
R^2 on test data is: [0.81996172 0.7864823  0.73272011 0.65696145 0.85381925]
```

[104]:

	Score
Training data MSE (mean)	4655.646
Testing data MSE (mean)	5650.149
Training data R^2 score (mean)	0.847
Testing data R^2 score(mean)	0.770

Gradient Boosting, Bagging, Random Forest Models

[105]:
```
Xf = insurance.drop(['charges'], axis = 1)
yf = insurance['charges']
Xf_train, Xf_test, yf_train, yf_test = train_test_split(Xf, yf, test_size=0.2,␣
↪random_state=0)
```

[106]:
```
#Gradient Boosting Model
gboost = GradientBoostingRegressor(random_state=22).fit(Xf_train, yf_train)
gboost1= GradientBoostingRegressor(max_depth=3, min_samples_leaf=9,␣
↪min_samples_split=2, n_estimators=50,random_state=22).fit(Xf_train, yf_train)
#Bagging Model
bagging = BaggingRegressor(DecisionTreeRegressor(max_depth=4,random_state=22)).
↪fit(Xf_train, yf_train)

#post pruned trees with ccp_alpha
bagging_postprune = BaggingRegressor(DecisionTreeRegressor(ccp_alpha=0.001),
                                     n_estimators=100,
                                     random_state = 22).fit(Xf_train, yf_train)
#Random forest model
forest = RandomForestRegressor(random_state=22, n_jobs=-1).fit(Xf_train,␣
↪yf_train)
forest1 = RandomForestRegressor(max_depth=4, min_samples_leaf=7,␣
↪min_samples_split=2, n_estimators=200,random_state=22, n_jobs=-1).
↪fit(Xf_train, yf_train).fit(Xf_train, yf_train)
```

[107]:
```
X2= df[['smoker', 'age', 'bmi']]
y2= df['charges']

X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.
↪2,random_state=22)
```

[108]:
```
lim_gboost = GradientBoostingRegressor(random_state=22).fit(X2_train, y2_train)
lim_gboost1= GradientBoostingRegressor(max_depth=3, min_samples_leaf=10,␣
↪min_samples_split=2, n_estimators=50,random_state=22).fit(X2_train, y2_train)
```

Calculate Predicted Insurance Charges with Gradient Boosting model.

[109]:
```
y_pred = gboost1.predict(Xf_test)
y_pred_train_gboost=gboost1.predict(Xf_train)
```

Table of Actual and Predicted Insurance charges along with their Difference with Gradient Boosting Regressor.

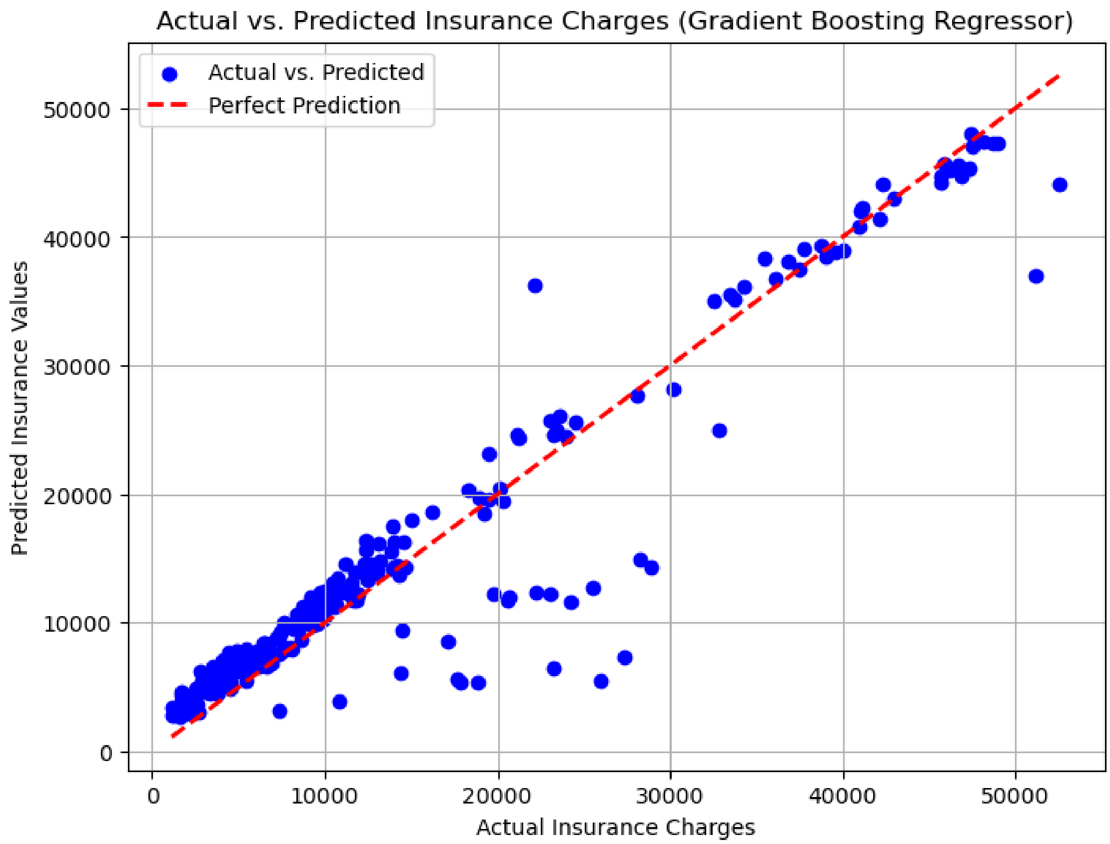
[110]:
```
df_scores_comp = pd.DataFrame({'Actual':yf_test, 'Predicted':y_pred,␣
↪'Difference': yf_test-y_pred})
df_scores_comp
```

[110]:

```
            Actual     Predicted   Difference
578     9724.53000  12361.630351 -2637.100351
610     8547.69130   9945.731053 -1398.039753
569    45702.02235  44667.364954  1034.657396
1034   12950.07120  14325.949720 -1375.878520
198     9644.25250  11383.398532 -1739.146032
…              …             …            …
1084   15019.76005  18010.239798 -2990.479748
726     6664.68595   7702.004651 -1037.318701
1132   20709.02034  12061.528818  8647.491522
725    40932.42950  40839.184985    93.244515
963     9500.57305   9964.799572  -464.226522

[268 rows x 3 columns]
```

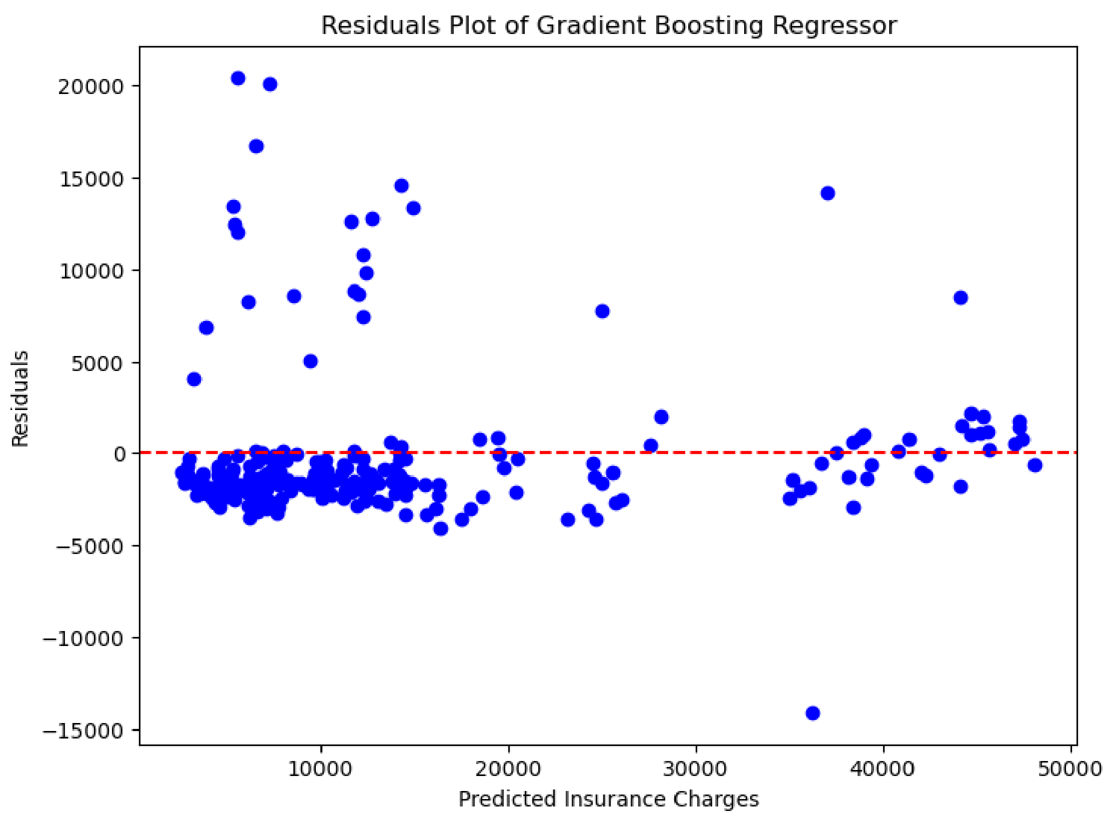
Plot of Actual and Predicted insurance Charges.

[111]:
```
plt.figure(figsize=(8, 6))
plt.scatter(yf_test, y_pred, color='blue', marker='o', label='Actual vs.␣
↪Predicted')
plt.plot([min(yf_test), max(yf_test)], [min(yf_test), max(yf_test)],␣
↪color='red', linestyle='--', lw=2, label='Perfect Prediction')
plt.xlabel('Actual Insurance Charges')
plt.ylabel('Predicted Insurance Values')
plt.title('Actual vs. Predicted Insurance Charges (Gradient Boosting␣
↪Regressor)')
plt.legend()
plt.grid(True)
plt.show()
```

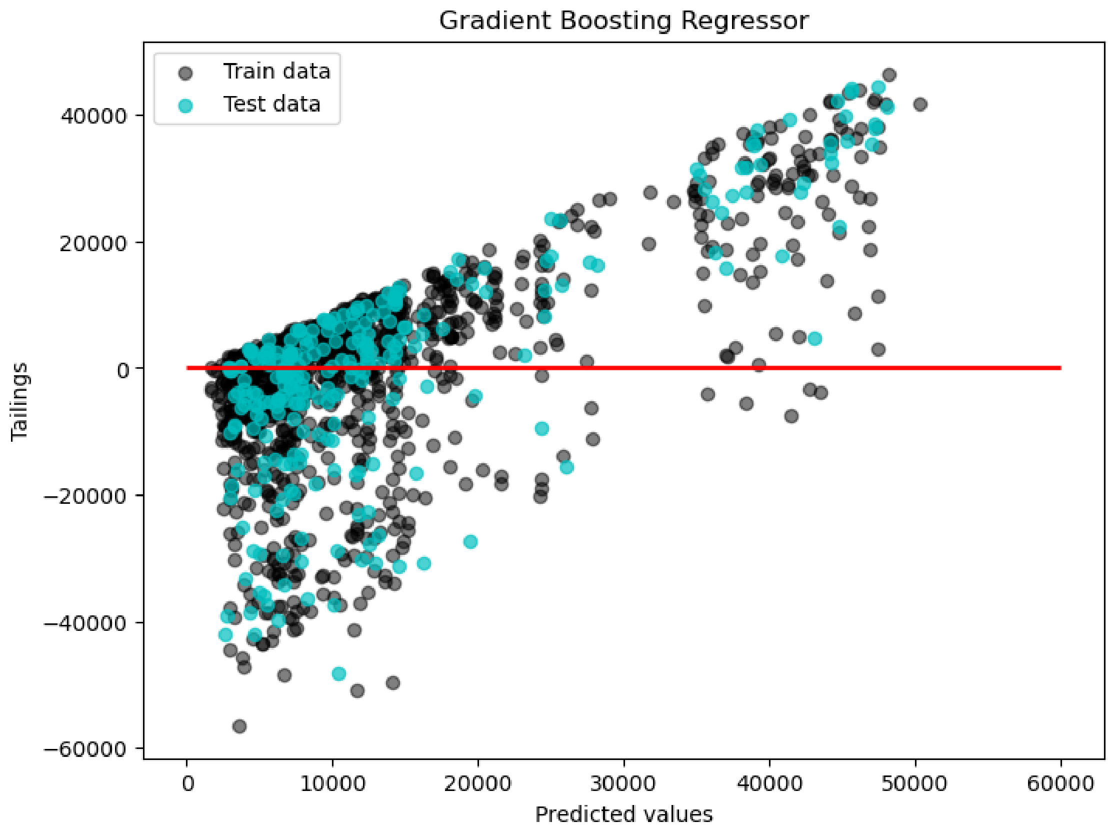


Plot of residuals of Gradient boosting regressor

[112]:
```
plt.figure(figsize=(8, 6))
residuals = yf_test - y_pred
plt.scatter(y_pred, residuals, color='blue', marker='o')
plt.xlabel('Predicted Insurance Charges')
plt.ylabel('Residuals')
plt.title('Residuals Plot of Gradient Boosting Regressor')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



[113]:
```
plt.figure(figsize=(8,6))

plt.scatter(y_pred_train_gboost,y_pred_train_gboost - y_train,

          c = 'black', marker = 'o', s = 35, alpha = 0.5,

          label = 'Train data')
plt.scatter(y_pred,y_pred - y_test,
          c = 'c', marker = 'o', s = 35, alpha = 0.7,
          label = 'Test data')
plt.xlabel('Predicted values')
plt.ylabel('Tailings')
plt.title('Gradient Boosting Regressor')
plt.legend(loc = 'upper left')
plt.hlines(y = 0, xmin = 0, xmax = 60000, lw = 2, color = 'red')
plt.show()
```



[114]:
```
lim_forest = RandomForestRegressor(random_state=22, n_jobs=-1).fit(X2_train,␣
 ↪y2_train)
lim_forest1 = RandomForestRegressor(max_depth=5, min_samples_leaf=7,␣
 ↪min_samples_split=2, n_estimators=600,random_state=22, n_jobs=-1).
 ↪fit(X2_train, y2_train)
```

Calculate Predicted Insurance Charges with Random forest Regressor

[115]:
```
Xf = insurance.drop(['charges'], axis = 1)
yf = insurance['charges']
Xf_train, Xf_test, yf_train, yf_test = train_test_split(Xf, yf, test_size=0.2,␣
 ↪random_state=0)
```

[116]:
```
y_pred1 = forest1.predict(Xf_test)
y_pred_train_forest=forest1.predict(Xf_train)
```

Table of Actual and Predicted Insurance charges along with their Difference with Random Forest Regressor.

[117]:
```
df_scores_comp1 = pd.DataFrame({'Actual':yf_test, 'Predicted':y_pred1,␣
 ↪'Difference': yf_test-y_pred1})
```

```
df_scores_comp1
```

[117]:
```
            Actual      Predicted   Difference
578    9724.53000  12790.451985 -3065.921985
610    8547.69130  10279.040273 -1731.348973
569   45702.02235  44959.122176   742.900174
1034  12950.07120  13870.043275  -919.972075
198    9644.25250  11079.262089 -1435.009589
…             …             …            …
1084  15019.76005  17023.598473 -2003.838423
726    6664.68595   7136.744908  -472.058958
1132  20709.02034  12543.849642  8165.170698
725   40932.42950  40315.286253   617.143247
963    9500.57305  10695.596504 -1195.023454

[268 rows x 3 columns]
```

Calculate Predicted Insurance Charges with Bagging Regressor

[118]:
```
y_pred2 = bagging.predict(Xf_test)
y_pred_train_bagging=bagging.predict(Xf_train)
```

[119]:
```
df_scores_comp2 = pd.DataFrame({'Actual':yf_test, 'Predicted':y_pred2,␣
 ↪'Difference': yf_test-y_pred2})
df_scores_comp2
```

[119]:
```
            Actual      Predicted   Difference
578    9724.53000  12810.905122 -3086.375122
610    8547.69130  10665.628599 -2117.937299
569   45702.02235  45467.190539   234.831811
1034  12950.07120  14273.369469 -1323.298269
198    9644.25250  11313.689404 -1669.436904
…             …             …            …
1084  15019.76005  16059.588716 -1039.828666
726    6664.68595   7567.162927  -902.476977
1132  20709.02034  12794.065786  7914.954554
725   40932.42950  39308.078446  1624.351054
963    9500.57305  10695.462936 -1194.889886

[268 rows x 3 columns]
```

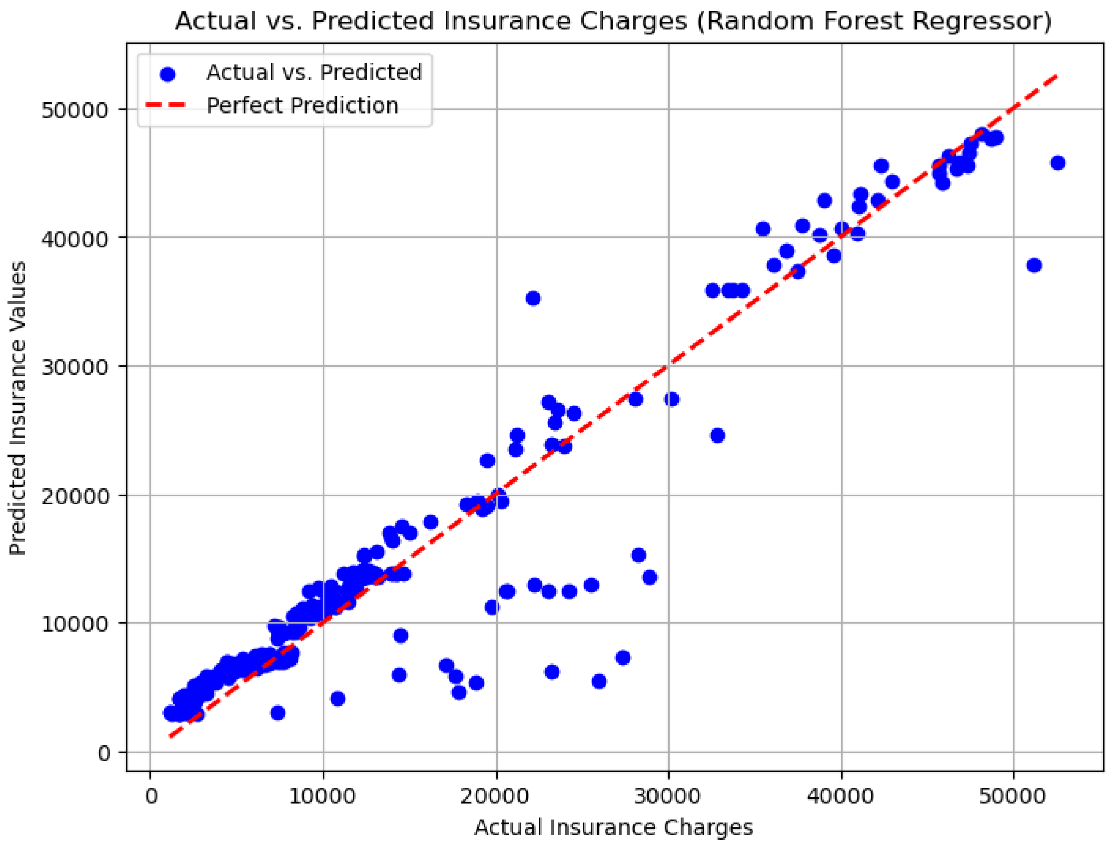
[120]:
```
df_scores_comp3 = pd.DataFrame({'Actual':yf_test, 'Predicted GBR':y_pred,␣
 ↪'Predicted RFR':y_pred1, 'Predicted BR':y_pred2,'Diff GBR':␣
 ↪yf_test-y_pred,'Diff RFR': yf_test-y_pred1, 'Diff BR': yf_test-y_pred2})
df_scores_comp3
```

[120]:
```
            Actual  Predicted GBR  Predicted RFR  Predicted BR     Diff GBR  \
578    9724.53000   12361.630351   12790.451985  12810.905122 -2637.100351
```

```
610    8547.69130    9945.731053   10279.040273  10665.628599 -1398.039753
569   45702.02235   44667.364954   44959.122176  45467.190539  1034.657396
1034  12950.07120   14325.949720   13870.043275  14273.369469 -1375.878520
198    9644.25250   11383.398532   11079.262089  11313.689404 -1739.146032
…             …              …              …             …            …
1084  15019.76005   18010.239798   17023.598473  16059.588716 -2990.479748
726    6664.68595    7702.004651    7136.744908   7567.162927 -1037.318701
1132  20709.02034   12061.528818   12543.849642  12794.065786  8647.491522
725   40932.42950   40839.184985   40315.286253  39308.078446    93.244515
963    9500.57305    9964.799572   10695.596504  10695.462936  -464.226522

         Diff RFR      Diff BR
578  -3065.921985 -3086.375122
610  -1731.348973 -2117.937299
569    742.900174   234.831811
1034  -919.972075 -1323.298269
198  -1435.009589 -1669.436904
…             …            …
1084 -2003.838423 -1039.828666
726   -472.058958  -902.476977
1132  8165.170698  7914.954554
725    617.143247  1624.351054
963  -1195.023454 -1194.889886

[268 rows x 7 columns]
```

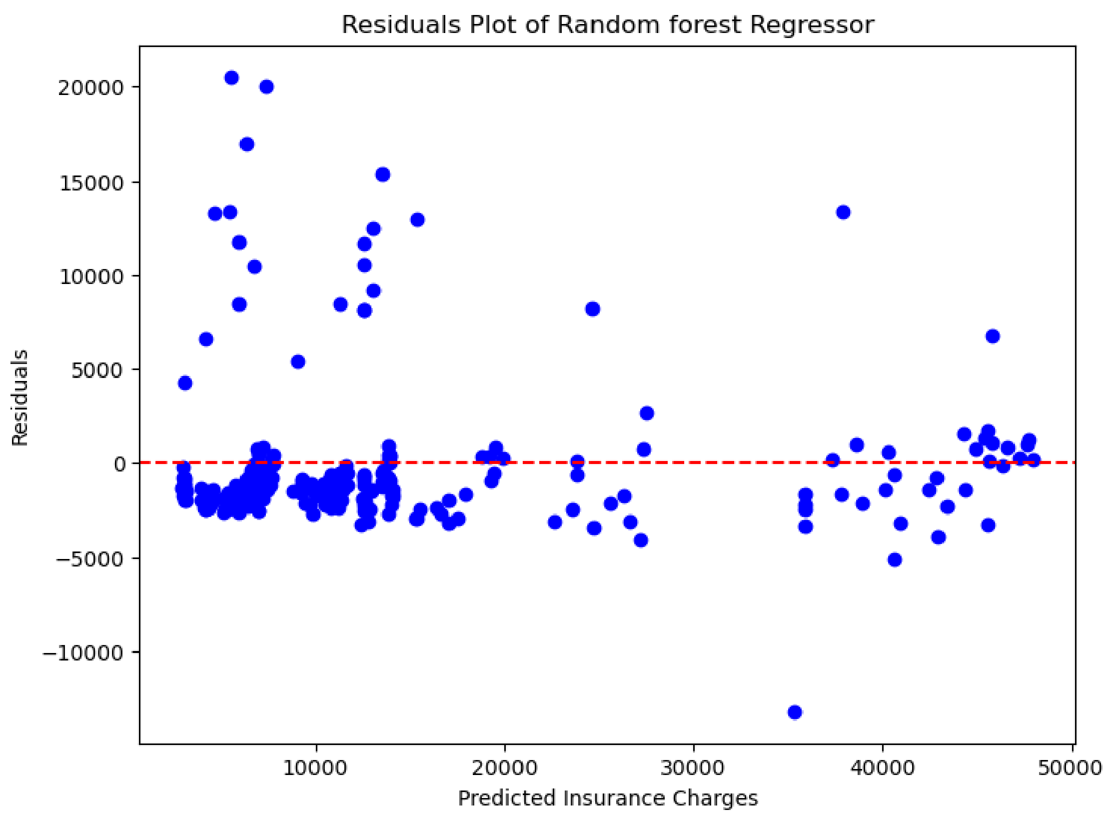
Plot of Actual and Predicted insurance Charges with Random Forest

[121]:
```python
plt.figure(figsize=(8, 6))
plt.scatter(yf_test, y_pred1, color='blue', marker='o', label='Actual vs.
 ↪Predicted')
plt.plot([min(yf_test), max(yf_test)], [min(yf_test), max(yf_test)],
 ↪color='red', linestyle='--', lw=2, label='Perfect Prediction')
plt.xlabel('Actual Insurance Charges')
plt.ylabel('Predicted Insurance Values')
plt.title('Actual vs. Predicted Insurance Charges (Random Forest Regressor)')
plt.legend()
plt.grid(True)
plt.show()
```



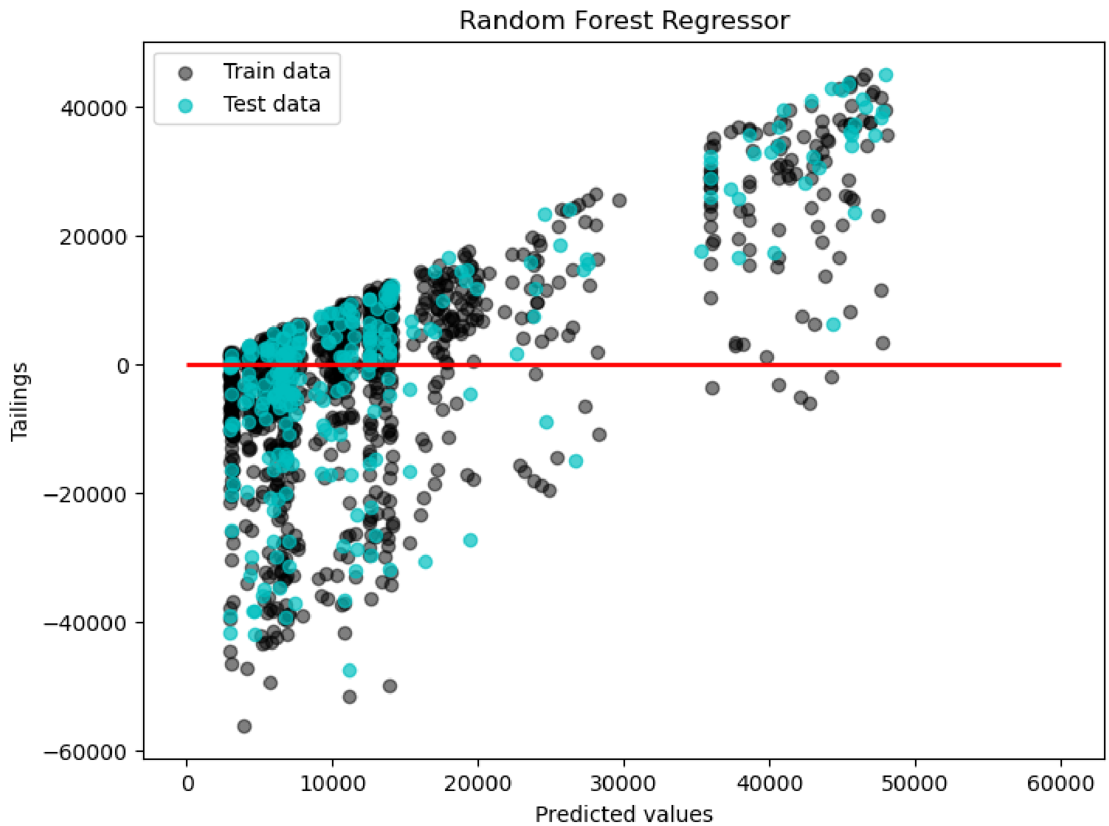
Plot of residuals of randon forest regressor

[122]:
```
plt.figure(figsize=(8, 6))
residuals = yf_test - y_pred1
plt.scatter(y_pred1, residuals, color='blue', marker='o')
plt.xlabel('Predicted Insurance Charges')
plt.ylabel('Residuals')
plt.title('Residuals Plot of Random forest Regressor')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



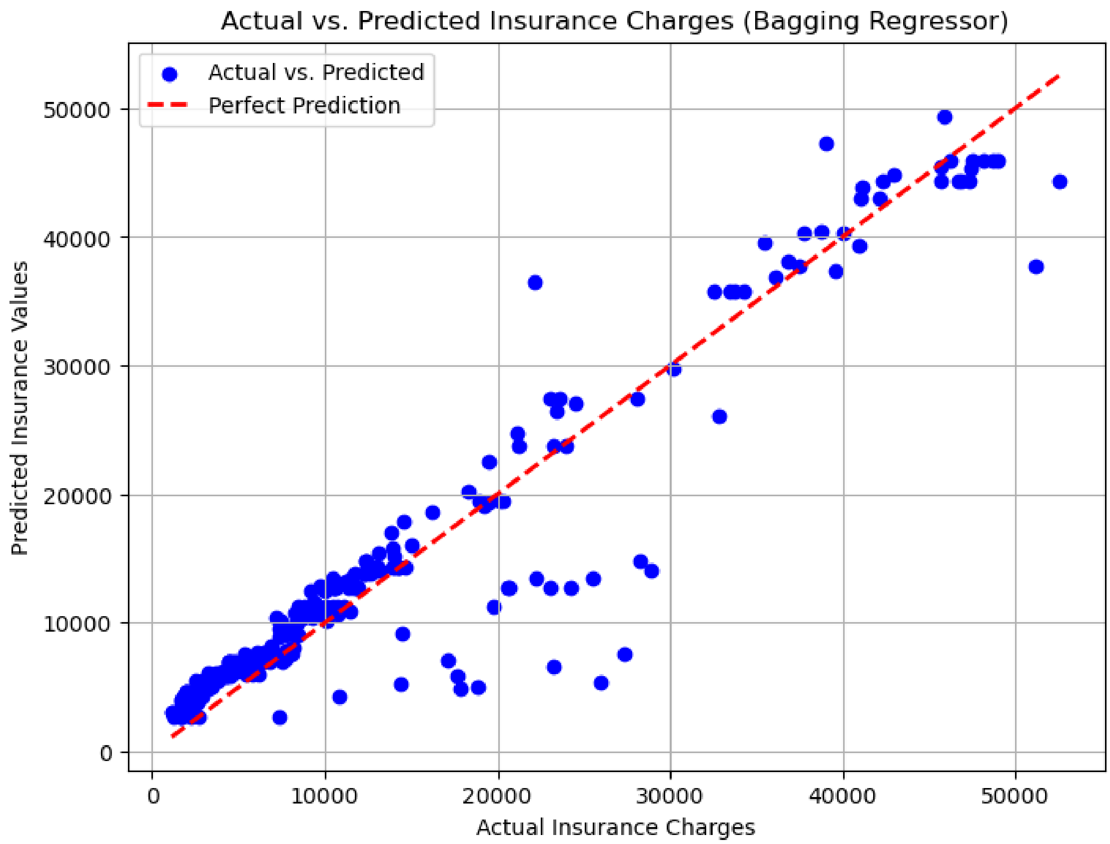
[123]:

```
plt.figure(figsize=(8,6))

plt.scatter(y_pred_train_forest,y_pred_train_forest - y_train,

          c = 'black', marker = 'o', s = 35, alpha = 0.5,

          label = 'Train data')
plt.scatter(y_pred1,y_pred1 - y_test,
          c = 'c', marker = 'o', s = 35, alpha = 0.7,
          label = 'Test data')
plt.xlabel('Predicted values')
plt.ylabel('Tailings')
plt.title('Random Forest Regressor')
plt.legend(loc = 'upper left')
plt.hlines(y = 0, xmin = 0, xmax = 60000, lw = 2, color = 'red')
plt.show()
```



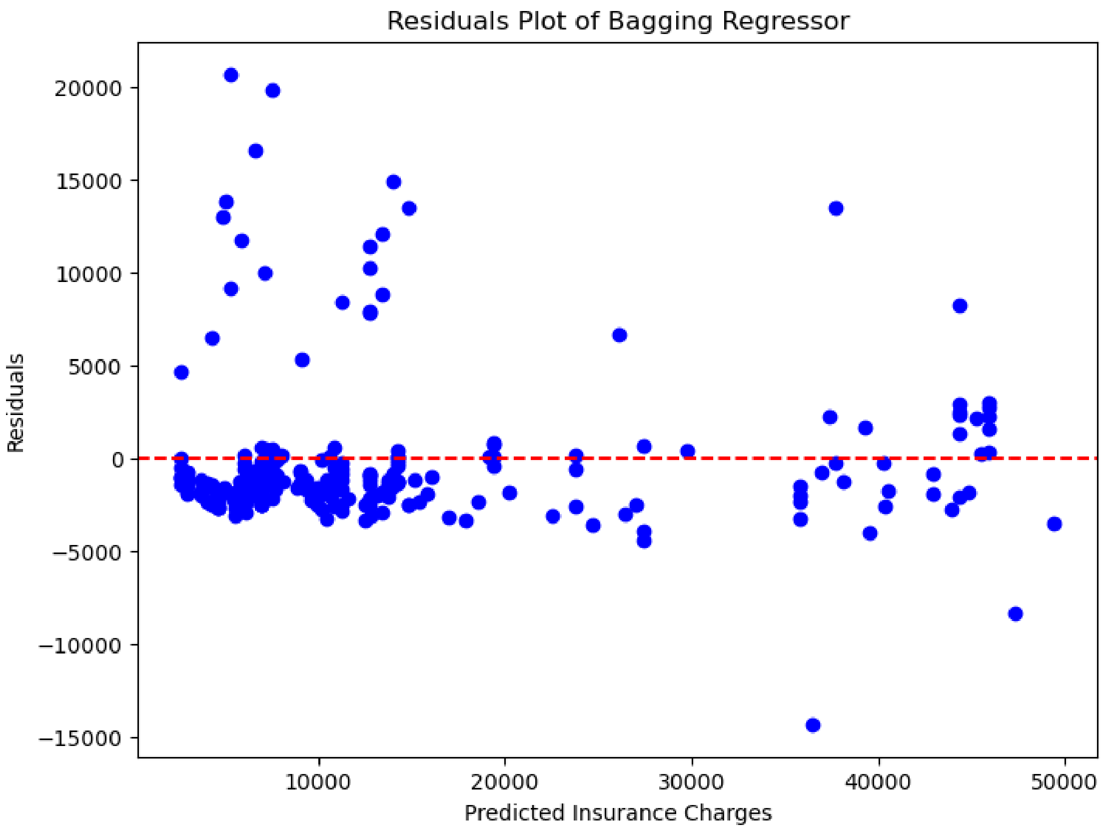
Plot of Actual and Predicted insurance Charges with Bagging Regressor

[124]:
```
plt.figure(figsize=(8, 6))
plt.scatter(yf_test, y_pred2, color='blue', marker='o', label='Actual vs.␣
↪Predicted')
plt.plot([min(yf_test), max(yf_test)], [min(yf_test), max(yf_test)],␣
↪color='red', linestyle='--', lw=2, label='Perfect Prediction')
plt.xlabel('Actual Insurance Charges')
plt.ylabel('Predicted Insurance Values')
plt.title('Actual vs. Predicted Insurance Charges (Bagging Regressor)')
plt.legend()
plt.grid(True)
plt.show()
```



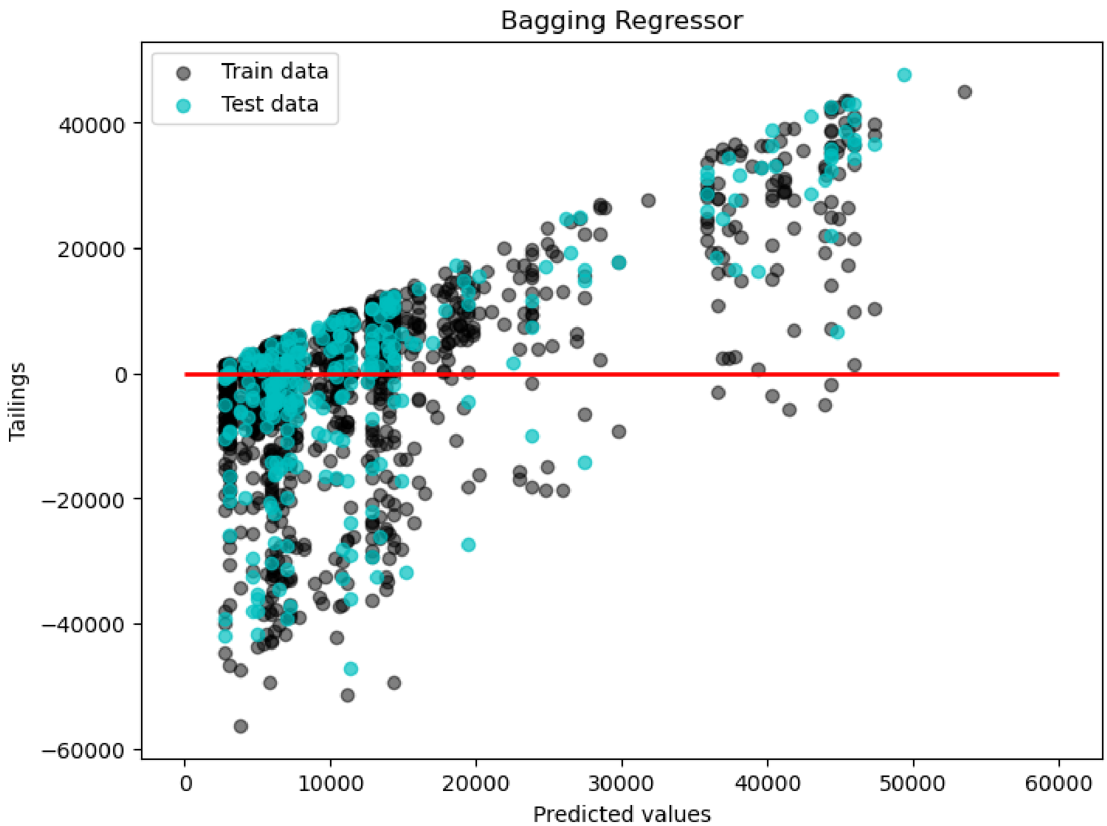
Residuals plot of bagging regressor

[125]:
```
plt.figure(figsize=(8, 6))
residuals = yf_test - y_pred2
plt.scatter(y_pred2, residuals, color='blue', marker='o')
plt.xlabel('Predicted Insurance Charges')
plt.ylabel('Residuals')
plt.title('Residuals Plot of Bagging Regressor')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



[126]:

```
plt.figure(figsize=(8, 6))

plt.scatter(y_pred_train_bagging,y_pred_train_bagging - y_train,

          c = 'black', marker = 'o', s = 35, alpha = 0.5,

          label = 'Train data')
plt.scatter(y_pred2,y_pred2 - y_test,
          c = 'c', marker = 'o', s = 35, alpha = 0.7,
          label = 'Test data')
plt.xlabel('Predicted values')
plt.ylabel('Tailings')
plt.title('Bagging Regressor')
plt.legend(loc = 'upper left')
plt.hlines(y = 0, xmin = 0, xmax = 60000, lw = 2, color = 'red')
plt.show()
```



[127]:
```
print("Statistics of gboost model:")
output=make_score(gboost, Xf_train, Xf_test, yf_train, yf_test)
output
```

Statistics of gboost model:

[127]:

	Score
Training data MSE:	3836.164
Testing data MSE:	4024.666
Training data R^2 score:	0.897
Testing data R^2 score:	0.898

[128]:
```
print("Statistics of gboost model with 3 Features:")
output=make_score(lim_gboost, X2_train, X2_test, y2_train, y2_test)
output
```

Statistics of gboost model with 3 Features:

[128]:

	Score
Training data MSE:	3834.743
Testing data MSE:	5012.398
Training data R^2 score:	0.900

```
Testing data R^2 score:      0.829
```

[129]:
```python
print("Statistics of gboost model after GridSearch:")
output=make_score(gboost1, Xf_train, Xf_test, yf_train, yf_test)
output
```

```
Statistics of gboost model after GridSearch:
```

[129]:

	Score
Training data MSE:	4233.361
Testing data MSE:	3972.907
Training data R^2 score:	0.875
Testing data R^2 score:	0.901

[130]:
```python
print("Statistics of gboost model with 3 Features and after GridSearch:")
output=make_score(lim_gboost1, X2_train, X2_test, y2_train, y2_test)
output
```

```
Statistics of gboost model with 3 Features and after GridSearch:
```

[130]:

	Score
Training data MSE:	4188.445
Testing data MSE:	4910.606
Training data R^2 score:	0.880
Testing data R^2 score:	0.836

[131]:
```python
print("Statistics of BaggingRegressor model: ")
output=make_score(bagging, Xf_train, Xf_test, yf_train, yf_test)
output
```

```
Statistics of BaggingRegressor model:
```

[131]:

	Score
Training data MSE:	4388.486
Testing data MSE:	4003.088
Training data R^2 score:	0.866
Testing data R^2 score:	0.899

[132]:
```python
print("Statistics of Postpruned Bagging Regressor model:")
output=make_score(bagging_postprune, Xf_train, Xf_test, yf_train, yf_test)
output
```

```
Statistics of Postpruned Bagging Regressor model:
```

[132]:

	Score
Training data MSE:	1932.486
Testing data MSE:	4409.670
Training data R^2 score:	0.974
Testing data R^2 score:	0.878

[133]:
```
print("Statistics of Random Forest model: ")
output=make_score(forest, Xf_train, Xf_test, yf_train, yf_test)
output
```

```
Statistics of Random Forest model:
```

[133]:

	Score
Training data MSE:	1930.205
Testing data MSE:	4425.672
Training data R^2 score:	0.974
Testing data R^2 score:	0.877

[134]:
```
print("Statistics of Random Forest model with 3 Features: ")
output=make_score(lim_forest, X2_train, X2_test, y2_train, y2_test)
output
```

```
Statistics of Random Forest model with 3 Features:
```

[134]:

	Score
Training data MSE:	1912.864
Testing data MSE:	5757.353
Training data R^2 score:	0.975
Testing data R^2 score:	0.775

[135]:
```
print("Statistics of Random Forest model after GridSearch: ")
output=make_score(forest1, Xf_train, Xf_test, yf_train, yf_test)
output
```

```
Statistics of Random Forest model after GridSearch:
```

[135]:

	Score
Training data MSE:	4395.499
Testing data MSE:	3947.109
Training data R^2 score:	0.865
Testing data R^2 score:	0.902

[136]:
```
print("Statistics of Random Forest model with 3 Features and after GridSearch:")
output=make_score(lim_forest1, X2_train, X2_test, y2_train, y2_test)
output
```

```
Statistics of Random Forest model with 3 Features and after GridSearch:
```

[136]:

	Score
Training data MSE:	4150.845
Testing data MSE:	4956.903
Training data R^2 score:	0.882
Testing data R^2 score:	0.833

[137]:
```
print("Statistics of gboost model with CV=5: ")
output=cv_make_score(gboost, Xf_train, Xf_test, yf_train, yf_test)
output
```

```
Statistics of gboost model with CV=5:
MSE on train data with CV=5: [4553.60201305 4251.98806251 4753.3670256
4432.81490312 5793.20251145]
MSE on test data with CV=5: [5129.70372305 4767.93819726 4084.47448543
5188.62722593 3863.06052878]
R^2 on training data is: [0.86814673 0.84365643 0.82832688 0.8657928  0.7922207
]
R^2 on test data is: [0.85811577 0.87598796 0.90167766 0.77390661 0.87394014]
```

[137]:

	Score
Training data MSE (mean)	4756.995
Testing data MSE (mean)	4606.761
Training data R^2 score (mean)	0.840
Testing data R^2 score(mean)	0.857

[138]:
```
print("Statistics of gboost model with 3 Features and CV=5: ")
output=cv_make_score(lim_gboost, X2_train, X2_test, y2_train, y2_test)
output
```

```
Statistics of gboost model with 3 Features and CV=5:
MSE on train data with CV=5: [4941.42742761 5349.16981005 3496.63382296
4549.65354217 5011.82812075]
MSE on test data with CV=5: [4141.71680837 6542.33467759 6117.71502427
6660.66098907 4870.49897251]
R^2 on training data is: [0.86766807 0.79795053 0.89795067 0.82836775
0.83976167]
R^2 on test data is: [0.8601806  0.78685974 0.658539   0.65137875 0.86035435]
```

[138]:

	Score
Training data MSE (mean)	4669.743
Testing data MSE (mean)	5666.585
Training data R^2 score (mean)	0.846
Testing data R^2 score(mean)	0.763

[139]:
```
print("Statistics of gboost model with CV=5 after GridSearch: ")
output=cv_make_score(gboost1, Xf_train, Xf_test, yf_train, yf_test)
output
```

```
Statistics of gboost model with CV=5 after GridSearch:
MSE on train data with CV=5: [4355.40802489 4199.36587749 4585.1802421
4430.60643235 5574.22572487]
MSE on test data with CV=5: [5132.25642927 4492.54899545 3554.33773417
4946.25995962 3284.02813878]
R^2 on training data is: [0.87937468 0.84750227 0.84026046 0.86592649 0.8076315
]
```

```
R^2 on test data is: [0.85797452 0.88989976 0.92554443 0.79453549 0.90889808]
```

[139]:

	Score
Training data MSE (mean)	4628.957
Testing data MSE (mean)	4281.886
Training data R^2 score (mean)	0.848
Testing data R^2 score(mean)	0.875

[140]:
```
print("Statistics of gboost model with 3 Features and CV=5 after GridSearch: ")
output=cv_make_score(lim_gboost1, X2_train, X2_test, y2_train, y2_test)
output
```

```
Statistics of gboost model with 3 Features and CV=5 after GridSearch:
MSE on train data with CV=5: [4806.23131328 5015.23582751 3280.66837351
4230.2955168  4924.15614547]
MSE on test data with CV=5: [3623.91129033 6292.58880961 5217.51803006
5653.9567995  4639.99579706]
R^2 on training data is: [0.87481014 0.8223899  0.91016729 0.85161717
0.84531874]
R^2 on test data is: [0.89295614 0.80282189 0.75163492 0.74879738 0.87325942]
```

[140]:

	Score
Training data MSE (mean)	4451.317
Testing data MSE (mean)	5085.594
Training data R^2 score (mean)	0.861
Testing data R^2 score(mean)	0.814

[141]:
```
print("Statistics of BaggingRegressor model with CV=5: ")
output=cv_make_score(bagging, Xf_train, Xf_test, yf_train, yf_test)
output
```

```
Statistics of BaggingRegressor model with CV=5:
MSE on train data with CV=5: [4389.34180629 4170.13953598 4737.74602256
4731.03489464 5646.42869408]
MSE on test data with CV=5: [5148.54989606 4376.50620001 3878.65486853
5128.02841704 3450.27933304]
R^2 on training data is: [0.87859434 0.84847108 0.83174607 0.8561793
0.80375924]
R^2 on test data is: [0.85657334 0.88008789 0.90464755 0.79993506 0.89001148]
```

[141]:

	Score
Training data MSE (mean)	4734.938
Testing data MSE (mean)	4396.404
Training data R^2 score (mean)	0.844
Testing data R^2 score(mean)	0.866

[142]:
```
print("Statistics of Postpruned Bagging Regressor model with CV=5: ")
output=cv_make_score(bagging_postprune, Xf_train, Xf_test, yf_train, yf_test)
output
```

```
Statistics of Postpruned Bagging Regressor model with CV=5:
MSE on train data with CV=5: [4942.07468396 4472.56579912 5163.22042196
5014.13779343 6031.722775  ]
MSE on test data with CV=5: [5231.92199252 4865.04724315 4152.4323241
5109.56509271 4420.98946642]
R^2 on training data is: [0.84469001 0.82701461 0.79744595 0.82828464
0.77475892]
R^2 on test data is: [0.85240485 0.87088499 0.89837865 0.78074434 0.83489783]
```

[142]:

	Score
Training data MSE (mean)	5124.744
Testing data MSE (mean)	4755.991
Training data R^2 score (mean)	0.814
Testing data R^2 score(mean)	0.847

[143]:

```
print("Statistics of Random Forest model with CV=5: ")
output=cv_make_score(forest, Xf_train, Xf_test, yf_train, yf_test)
output
```

```
Statistics of Random Forest model with CV=5:
MSE on train data with CV=5: [4948.48956336 4459.95571579 5162.74467623
5021.39911586 6044.59765175]
MSE on test data with CV=5: [5210.51435627 4919.4757084  4279.66338753
5086.15554291 4408.39688821]
R^2 on training data is: [0.84428656 0.82798868 0.79748327 0.82778693
0.77379633]
R^2 on test data is: [0.85361022 0.86797984 0.89205587 0.78274879 0.83583704]
```

[143]:

	Score
Training data MSE (mean)	5127.437
Testing data MSE (mean)	4780.841
Training data R^2 score (mean)	0.814
Testing data R^2 score(mean)	0.846

[144]:

```
print("Statistics of Random Forest model with 3 Features and CV=5: ")
output=cv_make_score(lim_forest, X2_train, X2_test, y2_train, y2_test)
output
```

```
Statistics of Random Forest model with 3 Features and CV=5:
MSE on train data with CV=5: [5122.8966154  5462.43968698 4141.07697024
4759.89278354 5464.28637715]
MSE on test data with CV=5: [4025.7771279  6326.7235711  5691.1051919
6382.57093393 4833.27553704]
R^2 on training data is: [0.85777007 0.78930305 0.8568681  0.81213902
0.80952368]
R^2 on test data is: [0.867899   0.80067687 0.70450112 0.67988171 0.86248071]
```

[144]:

	Score
Training data MSE (mean)	4990.118

```
Testing data MSE (mean)          5451.890
Training data R^2 score (mean)      0.825
Testing data R^2 score(mean)        0.783
```

[145]:
```python
print("Statistics of Random Forest model after GridSearch with CV=5: ")
output=cv_make_score(forest1, Xf_train, Xf_test, yf_train, yf_test)
output
```

```
Statistics of Random Forest model after GridSearch with CV=5:
MSE on train data with CV=5: [4345.09178226 4171.95171747 4634.48383145
4430.47142761 5606.81698728]
MSE on test data with CV=5: [5365.17155173 4438.87868594 3742.64650736
4977.04046768 3028.83239595]
R^2 on training data is: [0.87994543 0.84948683 0.83680669 0.86593466
0.80537545]
R^2 on test data is: [0.84479104 0.89251468 0.91744613 0.79197033 0.92250668]
```

[145]:
```
                                    Score
Training data MSE (mean)         4637.763
Testing data MSE (mean)          4310.514
Training data R^2 score (mean)      0.848
Testing data R^2 score(mean)        0.874
```

[146]:
```python
print("Statistics of Random Forest model with 3 Features and after GridSearch␣
 ↪with CV=5: ")
output=cv_make_score(lim_forest1, X2_train, X2_test, y2_train, y2_test)
output
```

```
Statistics of Random Forest model with 3 Features and after GridSearch with
CV=5:
MSE on train data with CV=5: [4832.43241755 5039.01059133 3251.86728116
4247.25898263 4949.3149406 ]
MSE on test data with CV=5: [3183.58630853 6380.99396419 5032.19854826
5953.57379366 4753.56472127]
R^2 on training data is: [0.87344148 0.82070198 0.91173765 0.85042475
0.84373408]
R^2 on test data is: [0.91738862 0.79724263 0.7689648  0.72146827 0.86697927]
```

[146]:
```
                                    Score
Training data MSE (mean)         4463.977
Testing data MSE (mean)          5060.783
Training data R^2 score (mean)      0.860
Testing data R^2 score(mean)        0.814
```

Hyperparamter tuning of GradientBoosting Model

[147]:
```python
# param_grid ={
#       'n_estimators': [30, 50, 100],
#       'min_samples_split': [1,2,3],
```

```
#      'max_depth': [2,3,4],
#      'min_samples_leaf' : [7,8,9,10]}

# grid_search_gboost = GridSearchCV(lim_gboost, param_grid, cv=5,␣
↪scoring='neg_mean_squared_error')
# grid_search_gboost.fit(X2_train, y2_train)

# best_params = grid_search_gboost.best_params_
# best_score = "%.3f" %np.sqrt(np.abs(grid_search_gboost.best_score_))
# print("Best parameters:", best_params)
# print("Best Score:", best_score)
```

Hyperparamter tuning of Bagging Regresssor

[148]:
```
# param_grid ={
#      'n_estimators': [10, 30, 50, 100],
#       'max_samples': [200,300,400],
#       'max_features': [5,6,7]}

# grid_search_bagging = GridSearchCV(bagging, param_grid,␣
↪scoring='neg_mean_squared_error')
# grid_search_bagging.fit(Xf_train, yf_train)

# best_params = grid_search_bagging.best_params_
# best_score = "%.3f" %np.sqrt(np.abs(grid_search_bagging.best_score_))
# print("Best parameters:", best_params)
# print("Best Score:", best_score)
```

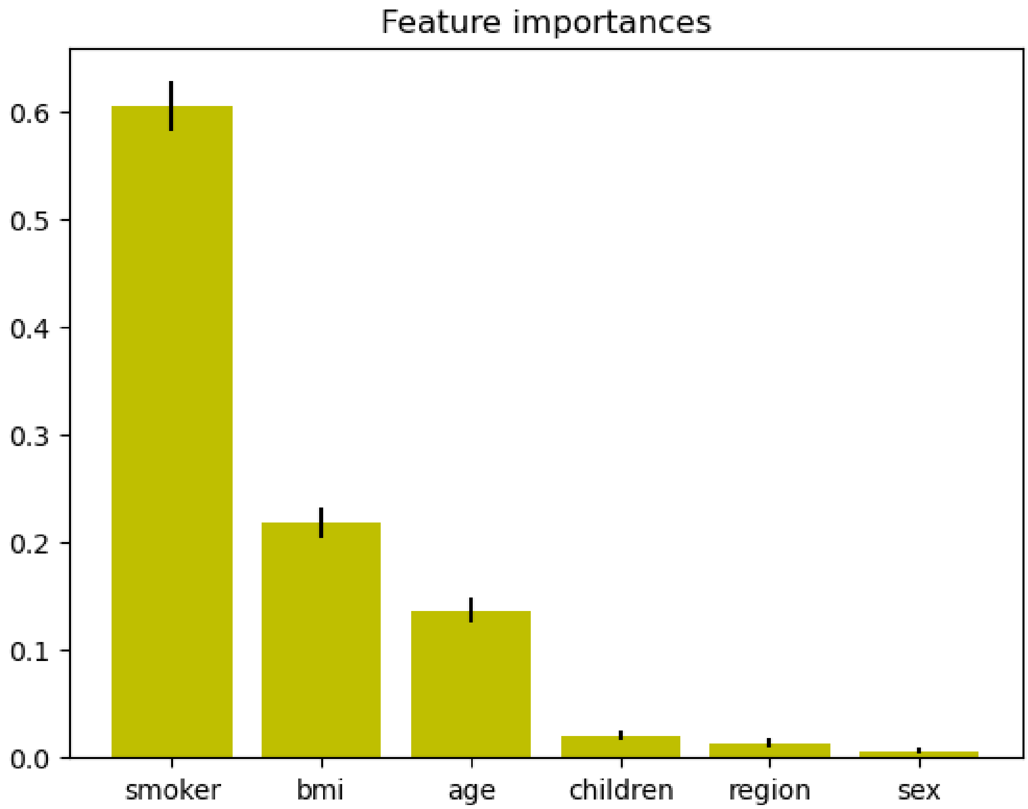
Hyperparameter tuning of Random Forest Regressor:

[149]:
```
# param_grid = {
#      'n_estimators': [200, 400,600, 800],
#      'min_samples_leaf': [5,7,9],
#      'max_depth': [4, 5, 6],
#      'min_samples_split': [1, 2, 3]}

# grid_search = GridSearchCV(lim_forest, param_grid, cv=5,␣
↪scoring='neg_mean_squared_error')
# grid_search.fit(X2_train, y2_train)

# best_params = grid_search.best_params_
# best_score = "%.3f" %np.sqrt(np.abs(grid_search.best_score_))
# print("Best parameters:", best_params)
# print("Best score:", best_score)

#pd.DataFrame(grid_search.cv_results_)
```

[150]:
```
print('Feature importance ranking\n\n')
importances = forest.feature_importances_
std = np.std([forest.feature_importances_ for forest in forest.
 ↪estimators_],axis=0)
indices = np.argsort(importances)[::-1]
variables = ['age', 'sex', 'bmi', 'children','smoker', 'region']
importance_list = []
for f in range(Xf.shape[1]):
    variable = variables[indices[f]]
    importance_list.append(variable)
    print("%d.%s(%f)" % (f + 1, variable, importances[indices[f]]))

# Plot feature importance of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(importance_list, importances[indices],
       color="y", yerr=std[indices], align="center")
```

```
Feature importance ranking


1.smoker(0.604253)
2.bmi(0.218423)
3.age(0.136862)
4.children(0.020854)
5.region(0.013593)
6.sex(0.006015)
```

[150]: <BarContainer object of 6 artists>



XGBoost Regressor

[151]:
```
Xx = df.drop(['charges'], axis = 1)
yx = df['charges']
Xx_train, Xx_test, yx_train, yx_test = train_test_split(Xx, yx, test_size=0.
↪2,random_state=22)
```

[152]:
```
xgboost= XGBRegressor(random_state=0)
xgboost.fit(Xx_train, yx_train)
xgboost_lim= XGBRegressor(random_state=0)
xgboost_lim.fit(X2_train, y2_train)
```

[152]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
```

```
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=0, …)
```

[153]:
```python
print("Statistics of xgboost model: ")
output=make_score(xgboost, Xx_train, Xx_test, yx_train, yx_test)
output
```

```
Statistics of xgboost model:
```

[153]:

	Score
Training data MSE:	724.217
Testing data MSE:	5711.582
Training data R^2 score:	0.996
Testing data R^2 score:	0.778

[154]:
```python
print("Statistics of xgboost model with 3 Features: ")
output=make_score(xgboost_lim, X2_train, X2_test, y2_train, y2_test)
output
```

```
Statistics of xgboost model with 3 Features:
```

[154]:

	Score
Training data MSE:	1305.936
Testing data MSE:	6002.028
Training data R^2 score:	0.988
Testing data R^2 score:	0.755

[155]:
```python
print("Statistics of xgboost model with CV=5: ")
output=cv_make_score(xgboost,Xx_train, Xx_test, yx_train, yx_test)
output
```

```
Statistics of xgboost model with CV=5:
MSE on train data with CV=5: [5456.49777626 5537.4625251  4048.98527029
5172.92016827 5455.03027062]
MSE on test data with CV=5: [4686.99725563 6500.20773446 5895.72380073
6138.48491473 5946.94192322]
R^2 on training data is: [0.83864302 0.78347575 0.86316341 0.77812223
0.81016844]
R^2 on test data is: [0.82094105 0.78959577 0.68287033 0.70389784 0.79180614]
```

[155]:

	Score
Training data MSE (mean)	5134.179
Testing data MSE (mean)	5833.671
Training data R^2 score (mean)	0.815
Testing data R^2 score(mean)	0.758

[156]:
```python
print("Statistics of xgboost model with 3 Features and CV=5: ")
output=cv_make_score(xgboost_lim,X2_train, X2_test, y2_train, y2_test)
output
```

```
Statistics of xgboost model with 3 Features and CV=5:
MSE on train data with CV=5: [5313.8151982  5693.39214762 4390.90727359
5449.48039005 5691.58529303]
MSE on test data with CV=5: [5089.24110467 7156.0211985  6388.22490777
6574.22140974 5025.08279755]
R^2 on training data is: [0.84697137 0.77110983 0.83907691 0.7537635
0.79334754]
R^2 on test data is: [0.78888811 0.74499821 0.6276743  0.66036859 0.8513493 ]
```

[156]:

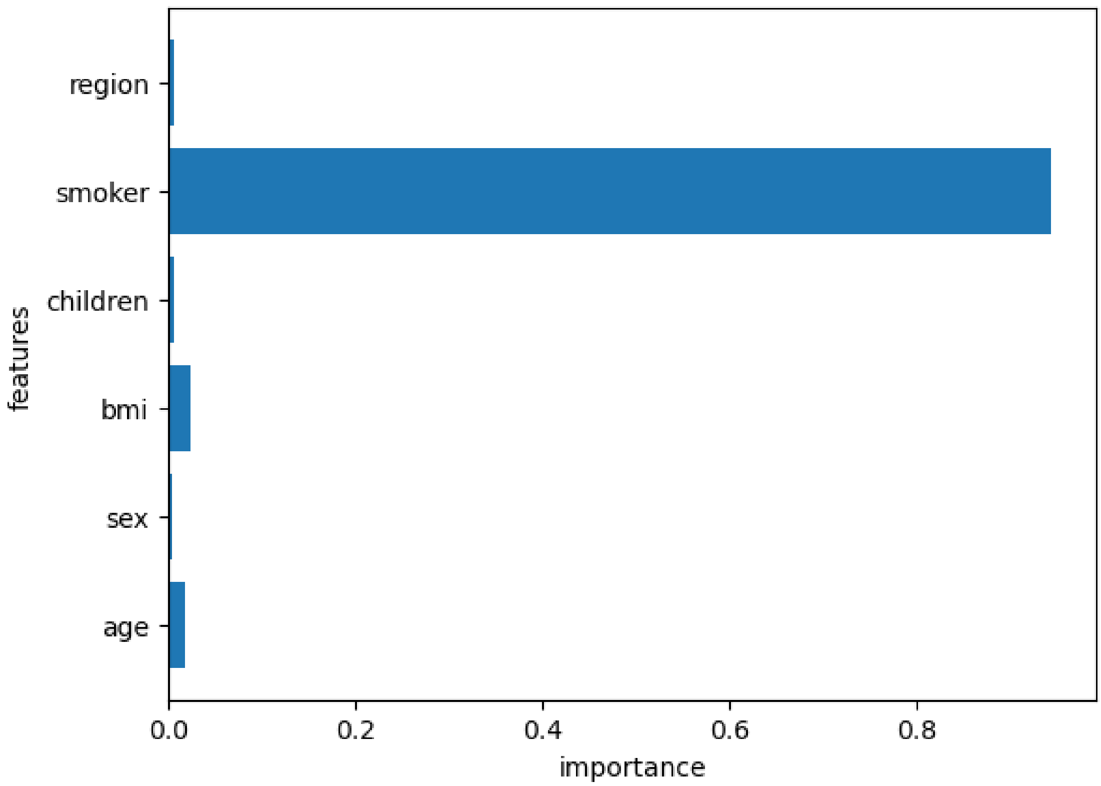
	Score
Training data MSE (mean)	5307.836
Testing data MSE (mean)	6046.558
Training data R^2 score (mean)	0.801
Testing data R^2 score(mean)	0.735

Plotting Feature importance

[157]:
```
xgboost.feature_importances_
```

[157]:
```
array([0.01731146, 0.00391589, 0.02305918, 0.00670818, 0.9436723 ,
       0.005333  ], dtype=float32)
```

[158]:
```
def plot_feature_importances(model):
    n_features = Xx_train.shape[1]
    plt.barh(range(n_features),model.feature_importances_,align="center")
    plt.yticks(np.arange(n_features),Xx_train)
    plt.xlabel("importance")
    plt.ylabel("features")
    plt.show
plot_feature_importances(xgboost)
plt.savefig("Features Importances")
```



Hyperparameters tuning of XGBoost model

[159]:
```
# set parameters for the optimization using GridSearch
param_grid = {
    'max_depth': [1, 2, 3, 5],
    'learning_rate': [0.1, 0.01, 0.001],
    'subsample': [0.5, 0.7, 1],
    'n_estimators': [300, 500, 1000, 1100]
}
grid_xgb = GridSearchCV(xgboost_lim, param_grid, scoring =␣
 ↪'neg_mean_squared_error', n_jobs= -1)
grid_xgb.fit(X2_train, y2_train)  # train model
best_params = grid_xgb.best_params_
best_score = "%.3f" %np.sqrt(np.abs(grid_xgb.best_score_))
print("Best parameters:", best_params)
print("Best score:", best_score)
```

```
Best parameters: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500,
'subsample': 0.7}
Best score: 4513.531
```

[160]:
```
xgboost1= XGBRegressor(learning_rate=0.01, max_depth=3, n_estimators=500,␣
↪subsample=0.7, random_state=0, eval_metric = 'error')
xgboost1.fit(Xx_train, yx_train)
```

[160]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric='error', feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.01, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=500, n_jobs=None,
             num_parallel_tree=None, random_state=0, …)
```

[161]:
```
xgboost1_lim= XGBRegressor(learning_rate=0.01, max_depth=3, n_estimators=500,␣
↪subsample=0.7, random_state=0, eval_metric = 'error')
xgboost1_lim.fit(X2_train, y2_train)
```

[161]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric='error', feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.01, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=500, n_jobs=None,
             num_parallel_tree=None, random_state=0, …)
```

[162]:
```
print("Statistics of xgboost model after GridSearch: ")
output=make_score(xgboost1, Xx_train, Xx_test, yx_train, yx_test)
output
```

```
Statistics of xgboost model after GridSearch:
```

[162]:

	Score
Training data MSE:	4029.055
Testing data MSE:	4744.108
Training data R^2 score:	0.889
Testing data R^2 score:	0.847

[163]:
```
print("Statistics of xgboost model with 3 Features and after GridSearch: ")
output=make_score(xgboost1_lim, X2_train, X2_test, y2_train, y2_test)
output
```

```
Statistics of xgboost model with 3 Features and after GridSearch:
```

[163]:

	Score
Training data MSE:	4147.019
Testing data MSE:	4864.881
Training data R^2 score:	0.883
Testing data R^2 score:	0.839

[164]:
```
print("Statistics of xgboost model with CV=5 after GridSearch: ")
output=cv_make_score(xgboost1,Xx_train, Xx_test, yx_train, yx_test)
output
```

```
Statistics of xgboost model with CV=5 after GridSearch:
MSE on train data with CV=5: [4815.41824845 4928.95695366 3172.90305097
4207.61541681 4823.49968457]
MSE on test data with CV=5: [3254.17223568 6343.61512925 4994.37448026
5665.73720746 4725.52135404]
R^2 on training data is: [0.87433109 0.82844831 0.91597211 0.85320397
0.85157789]
R^2 on test data is: [0.91368471 0.79961111 0.77242486 0.74774949 0.86854413]
```

[164]:

	Score
Training data MSE (mean)	4389.679
Testing data MSE (mean)	4996.684
Training data R^2 score (mean)	0.865
Testing data R^2 score(mean)	0.820

[165]:
```
print("Statistics of xgboost model with 3 Features and CV=5 after GridSearch: ")
output=cv_make_score(xgboost1_lim,X2_train, X2_test, y2_train, y2_test)
output
```

```
Statistics of xgboost model with 3 Features and CV=5 after GridSearch:
MSE on train data with CV=5: [4873.32975243 5023.59742925 3277.31956092
4251.10124173 4905.22626234]
MSE on test data with CV=5: [3459.49249175 6319.37477394 5110.30416465
5707.75552096 4631.6810326 ]
R^2 on training data is: [0.87129026 0.82179717 0.91035059 0.850154
0.84650573]
R^2 on test data is: [0.90244907 0.80113965 0.76173727 0.74399413 0.87371325]
```

[165]:

	Score
Training data MSE (mean)	4466.115
Testing data MSE (mean)	5045.722
Training data R^2 score (mean)	0.860
Testing data R^2 score(mean)	0.817

Lasso Regression Model

[166]:
```
XT = insurance.drop(['charges'], axis = 1)
yT = insurance['charges']
```

```
XT_train, XT_test, yT_train, yT_test = train_test_split(XT, yT, test_size=0.2,␣
 ↪random_state=0)
lasso = Lasso(alpha=0.2, fit_intercept=True, precompute=False, max_iter=1000,
              tol=0.0001, warm_start=False, positive=False, random_state=None,␣
 ↪selection='cyclic')
lasso.fit(XT_train, yT_train)
lasso.score(XT_test, yT_test)
```

[166]: 0.7998690236224705

[167]:
```
print("Statistics of Lasso model: ")
output=make_score(lasso, XT_train, XT_test, yT_train, yT_test)
output
```

```
Statistics of Lasso model:
```

[167]:

	Score
Training data MSE:	6142.441
Testing data MSE:	5643.300
Training data R^2 score:	0.737
Testing data R^2 score:	0.800

[168]:
```
print("Statistics of Lasso model with CV=5: ")
output=cv_make_score(lasso, XT_train, XT_test, yT_train, yT_test)
output
```

```
Statistics of Lasso model with CV=5:
MSE on train data with CV=5: [6089.54099378 5732.57557519 5877.89465409
6199.69379975 6934.88784015]
MSE on test data with CV=5: [7079.04575141 4605.41126558 5832.99557911
6464.1125689  4043.44156681]
R^2 on training data is: [0.7641967  0.71581868 0.7374916  0.7374832
0.70225551]
R^2 on test data is: [0.72979151 0.88429838 0.7994775  0.64908595 0.86189285]
```

[168]:

	Score
Training data MSE (mean)	6166.919
Testing data MSE (mean)	5605.001
Training data R^2 score (mean)	0.731
Testing data R^2 score(mean)	0.785

[169]:
```
XX = insurance1.drop(['charges'], axis = 1)
yy = insurance1['charges']
```

[170]:
```
insurance1.select_dtypes('object').columns.tolist()
```

[170]: ['sex', 'smoker', 'region']

[171]:
```
#make transformer
transformer1 = make_column_transformer((OneHotEncoder(), insurance1.
 ↪select_dtypes('object').columns.tolist()),
                                      remainder = StandardScaler(),
                                     verbose_feature_names_out=False)

#pipeline to transform and model
pipe2 = Pipeline([('transform', transformer1), ('model', Lasso(alpha = 0.5))])
#fit the train data
pipe2.fit(XX, yy)
#dataframe of coefficients
coef_df = pd.DataFrame({'features': pipe2.named_steps['transform'].
 ↪get_feature_names_out(),
          'coef': pipe2.named_steps['model'].coef_}).sort_values(by = 'coef')
print(coef_df.shape)

coef_df.head(10)
```

(11, 2)

[171]:

	features	coef
2	smoker_no	-2.384504e+04
6	region_southeast	-1.429208e+02
7	region_southwest	-6.836411e+01
1	sex_male	-4.751916e-13
3	smoker_yes	1.891499e-11
0	sex_female	1.290663e+02
5	region_northwest	5.343800e+02
10	children	5.725270e+02
4	region_northeast	8.873688e+02
9	bmi	2.066632e+03

[172]:
```
non_zero_coefs = coef_df[coef_df['coef'] != 0].shape[0]
print(f'Using Lasso with alpha = 0.2 resulted in {non_zero_coefs} non-zero␣
 ↪coefficients.')
```

Using Lasso with alpha = 0.2 resulted in 11 non-zero coefficients.

[173]:
```
coef_df[coef_df['coef'] != 0].shape[0]
positive_coefs = coef_df[coef_df['coef'] > 0]['features'].tolist()
print(len(positive_coefs))
print(positive_coefs)

negative_coefs = coef_df[coef_df['coef'] < 0]['features'].tolist()
print(len(negative_coefs))
print(negative_coefs)
insurance1.dtypes
```

7

```
['smoker_yes', 'sex_female', 'region_northwest', 'children', 'region_northeast',
'bmi', 'age']
4
['smoker_no', 'region_southeast', 'region_southwest', 'sex_male']
```

[173]:
```
age            int64
sex           object
bmi          float64
children       int64
smoker        object
region        object
charges      float64
dtype: object
```

Neural Networks

[174]:
```
nnet= MLPRegressor(alpha=0.0001)
XT = insurance.drop(['charges'], axis = 1)
yT = insurance['charges']
XT_train, XT_test, yT_train, yT_test = train_test_split(XT, yT, test_size=0.2,␣
↪random_state=0)
```

[175]:
```
nnet.fit(XT_train, yT_train)

output=make_score(nnet, XT_train, XT_test, yT_train, yT_test)
output
```

[175]:

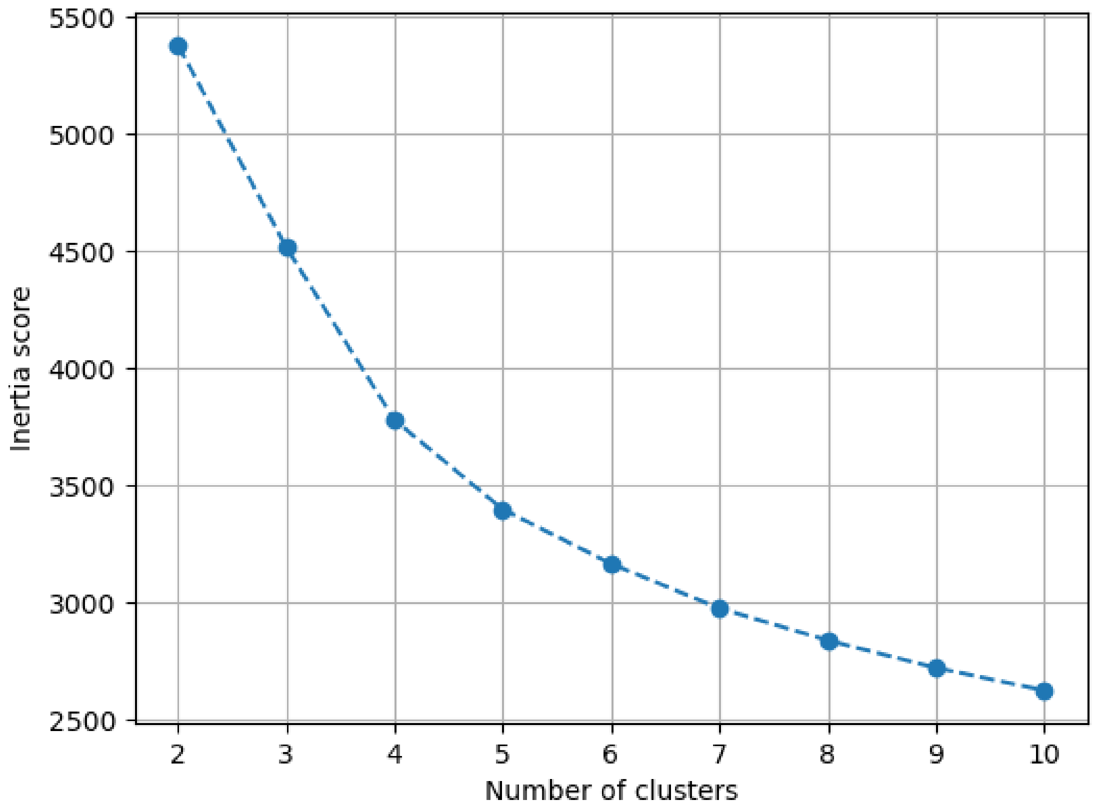
	Score
Training data MSE:	11887.378
Testing data MSE:	12356.773
Training data R^2 score:	0.014
Testing data R^2 score:	0.040

KMeans Clustering

[176]:
```
#there is no need for this but just doing it to check score.
X1 = insurance1.drop(['charges'], axis = 1)
y1 = insurance1['charges']

#make transformer
transformer1 = make_column_transformer((OneHotEncoder(drop = 'first'),␣
↪insurance1.select_dtypes('object').columns.tolist()),
                                      remainder = StandardScaler(),
                                     verbose_feature_names_out=False)
```

[177]:
```
data_scaled = transformer1.fit_transform(insurance1)

inertia_scores = []
for i in range(2, 11):
```

```
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data_scaled)
    labels = kmeans.labels_
    inertia_scores.append(kmeans.inertia_)
plt.plot(range(2, 11), inertia_scores, '--o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia score')
plt.grid();
```



[178]:
```
kmeans_5 = KMeans(n_clusters=5, random_state=22)
kmeans_5.fit(data_scaled)
print("KMeans Clustering inertia is:",kmeans_5.inertia_)
#print("KMeans cluster centres:", kmeans_5.cluster_centers_)
print("The number of iterations required to converge:", kmeans_5.n_iter_)
print("First five predicted labels:", kmeans_5.labels_[:5])
```

```
KMeans Clustering inertia is: 3398.2495711048987
The number of iterations required to converge: 17
First five predicted labels: [0 0 3 4 0]
```

[179]:
```
pred = kmeans_5.predict(data_scaled)
print("KMeans predicted values:", pred)
frame = pd.DataFrame(data_scaled)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

```
KMeans predicted values: [0 0 3 … 0 0 4]
```

[179]:
```
cluster
0    377
3    297
4    285
1    223
2    156
Name: count, dtype: int64
```

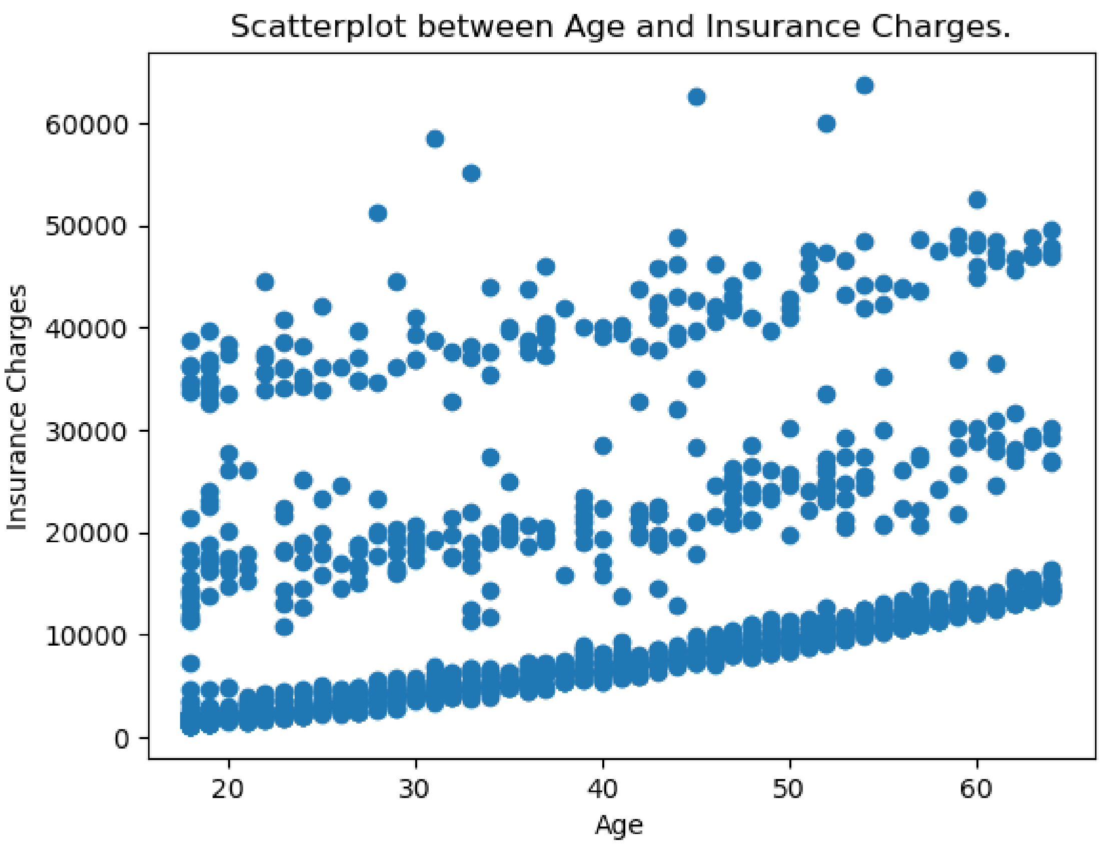
[180]:
```
insurance2['label'] =  kmeans_5.labels_
insurance2.head()
```

[180]:

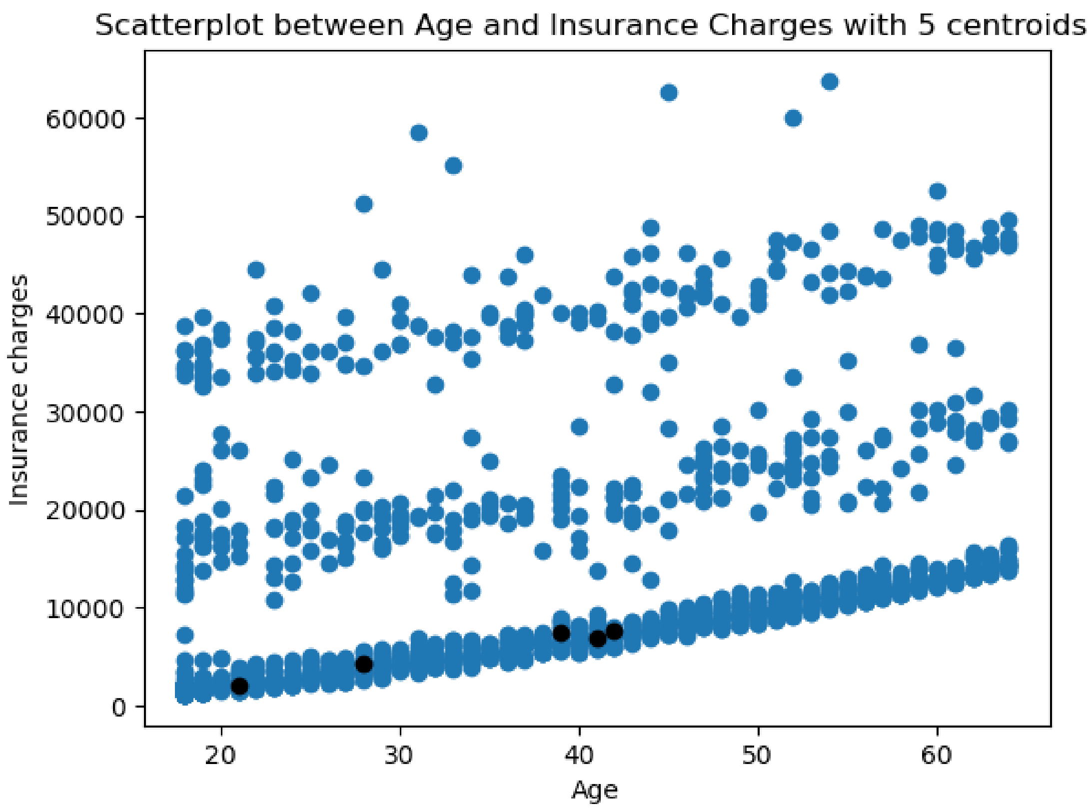
	age	sex	bmi	children	smoker	region	charges	label
0	19	female	27.900	0	yes	southwest	16884.92400	0
1	18	male	33.770	1	no	southeast	1725.55230	0
2	28	male	33.000	3	no	southeast	4449.46200	3
3	33	male	22.705	0	no	northwest	21984.47061	4
4	32	male	28.880	0	no	northwest	3866.85520	0

KMeans Clustering study relationship between Age and Insurance charges.

[181]:
```
plt.scatter(insurance['age'],insurance['charges'])
plt.xlabel('Age')
plt.ylabel('Insurance Charges')
plt.title("Scatterplot between Age and Insurance Charges.")
plt.show()
```



[182]:
```
K=5
centroids = insurance.sample(n=K)
plt.scatter(insurance['age'],insurance['charges'])
plt.scatter(centroids['age'],centroids['charges'],c='black')
plt.xlabel('Age')
plt.ylabel('Insurance charges')
plt.title("Scatterplot between Age and Insurance Charges with 5 centroids.")
plt.show()
```



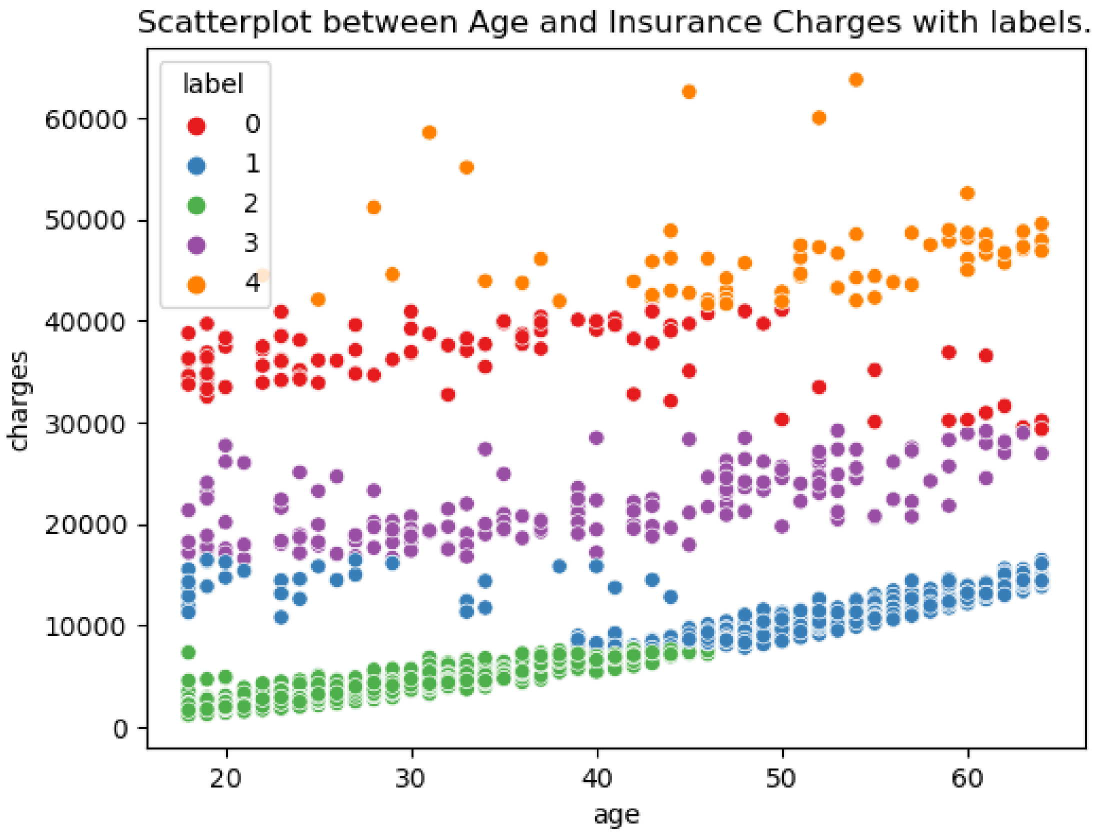
KMeans Clustering, plot between Age and Insurance Charges with 5 centroids.

[183]:
```
km_sample = KMeans(n_clusters=5)
km_sample.fit(insurance[['age','charges']])
```

[183]: KMeans(n_clusters=5)

[184]:
```
labels_sample = km_sample.labels_
insurance['label'] = labels_sample
sns.
↪scatterplot(x=insurance['age'],y=insurance['charges'],hue=insurance['label'],palette='Set1'
plt.title("Scatterplot between Age and Insurance Charges with labels.")
```

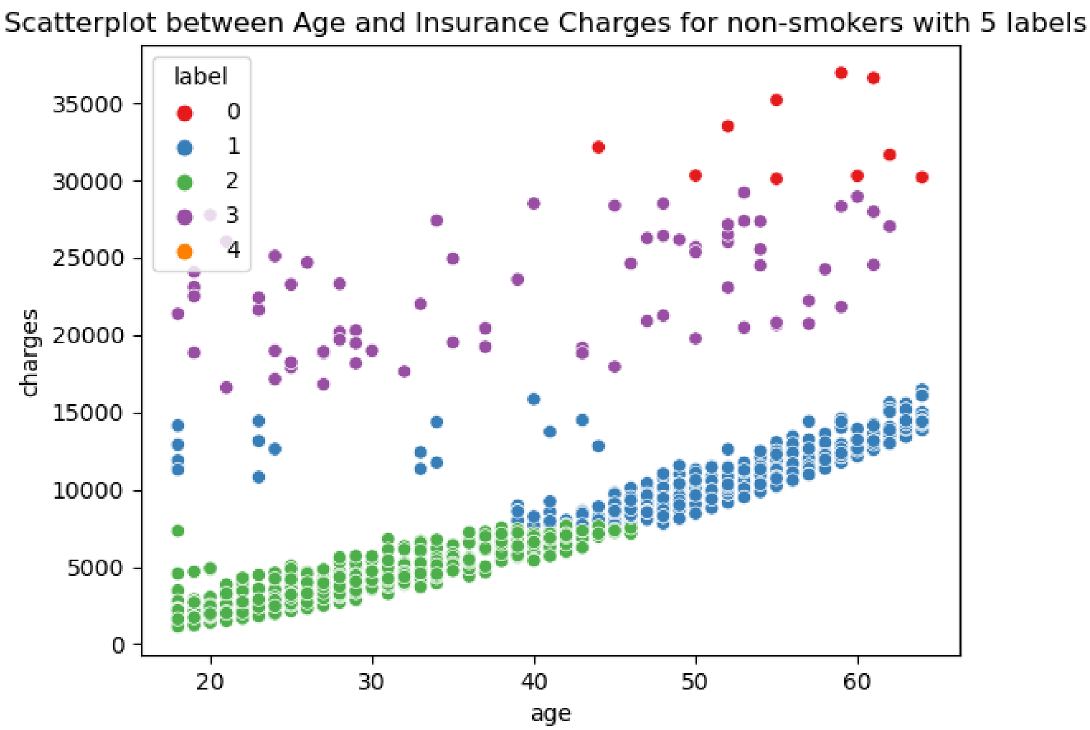
[184]: Text(0.5, 1.0, 'Scatterplot between Age and Insurance Charges with labels.')



KMeans clustering study relationship between Age and Insurance charges based on Smoking Habits.

[185]:
```
sns.scatterplot(x=insurance[(insurance.smoker == 0)].age,y=insurance[(insurance.
 ↪smoker == 0)].charges, hue=insurance['label'],palette='Set1')
plt.title("Scatterplot between Age and Insurance Charges for non-smokers with 5␣
 ↪labels.")
```

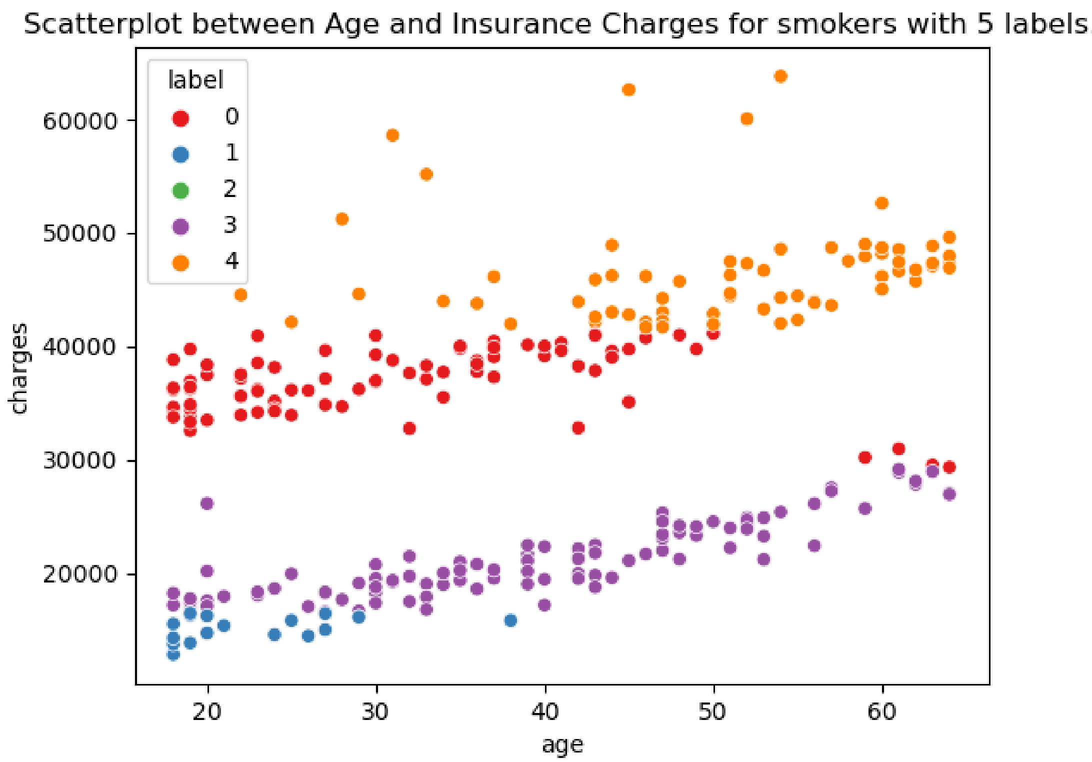
[185]:
```
Text(0.5, 1.0, 'Scatterplot between Age and Insurance Charges for non-smokers
with 5 labels.')
```



Distribution of insurance charges and age for non-smokers

[186]:
```
sns.scatterplot(x=insurance[(insurance.smoker == 1)].age,y=insurance[(insurance.
↪smoker == 1)].charges, hue=insurance['label'],palette='Set1')
plt.title("Scatterplot between Age and Insurance Charges for smokers with 5␣
↪labels.")
```

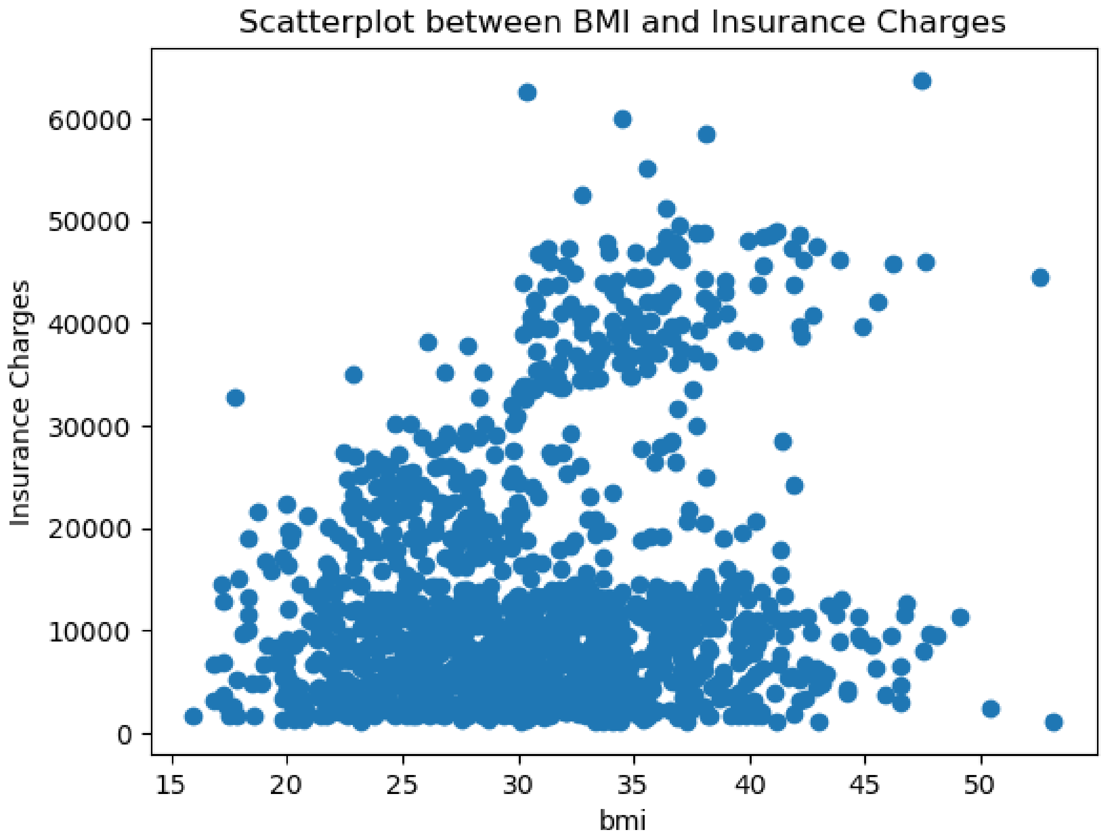
[186]:
```
Text(0.5, 1.0, 'Scatterplot between Age and Insurance Charges for smokers with 5
labels.')
```



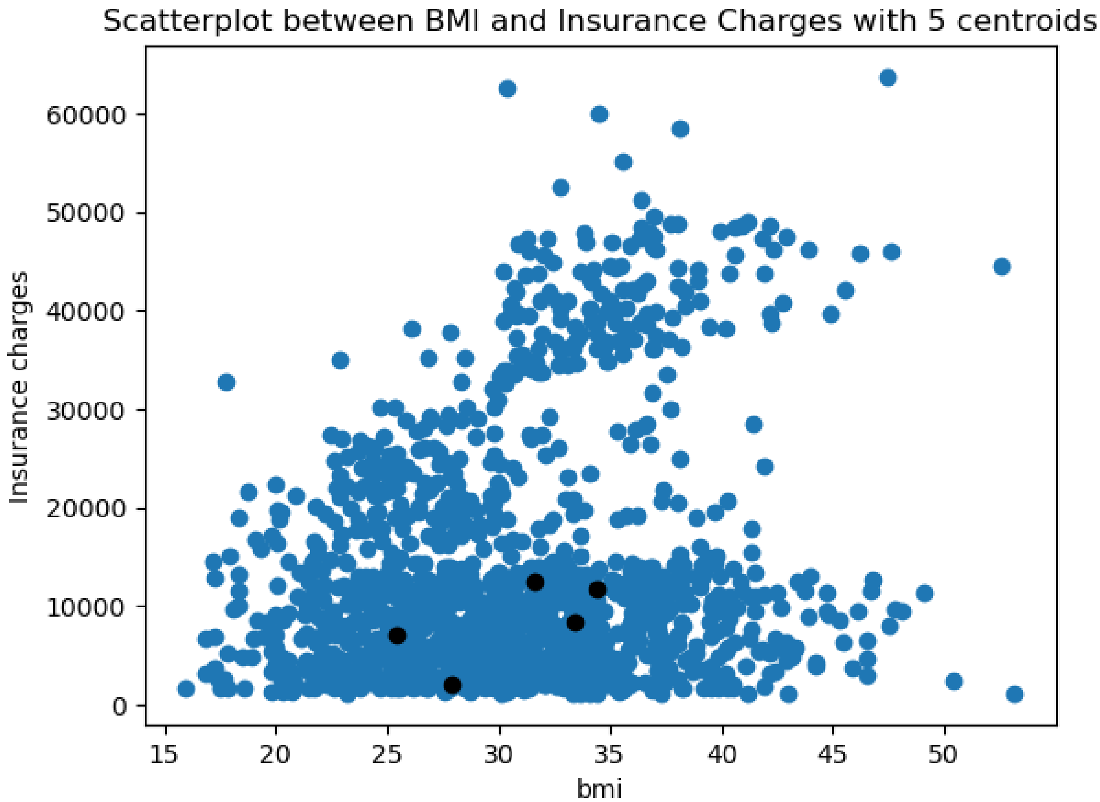
Distribution of insuarnce charges and age for smokers

KMeans Clustering study relationship between BMI and Insurance Charges.

[187]:
```
plt.scatter(insurance['bmi'],insurance['charges'])
plt.xlabel('bmi')
plt.ylabel('Insurance Charges')
plt.title("Scatterplot between BMI and Insurance Charges")
plt.show()
```



[188]:
```
K=5
centroids = insurance.sample(n=K)
plt.scatter(insurance['bmi'],insurance['charges'])
plt.scatter(centroids['bmi'],centroids['charges'],c='black')
plt.xlabel('bmi')
plt.ylabel('Insurance charges')
plt.title("Scatterplot between BMI and Insurance Charges with 5 centroids")
plt.show()
```



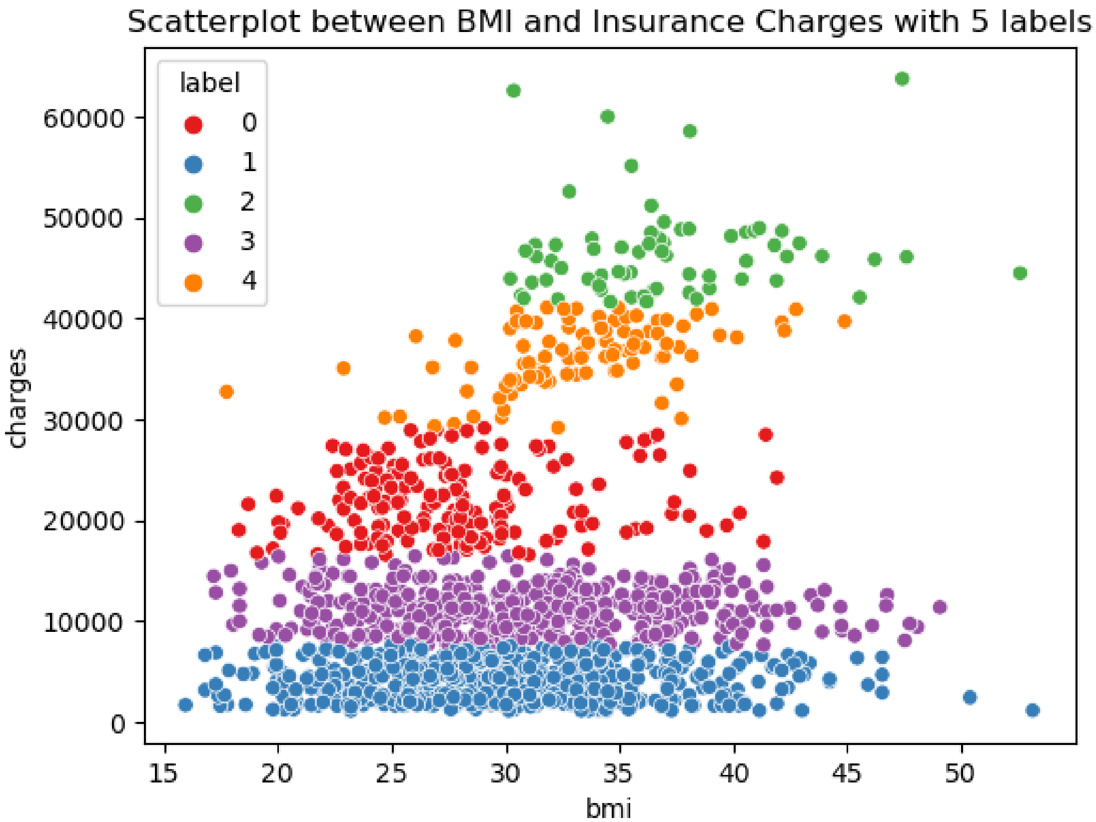
KMeans Clustering, plot between BMI and Insurance Charges with 5 centroids.

[189]:
```
km_sample = KMeans(n_clusters=5)
km_sample.fit(insurance[['bmi','charges']])
```

[189]: KMeans(n_clusters=5)

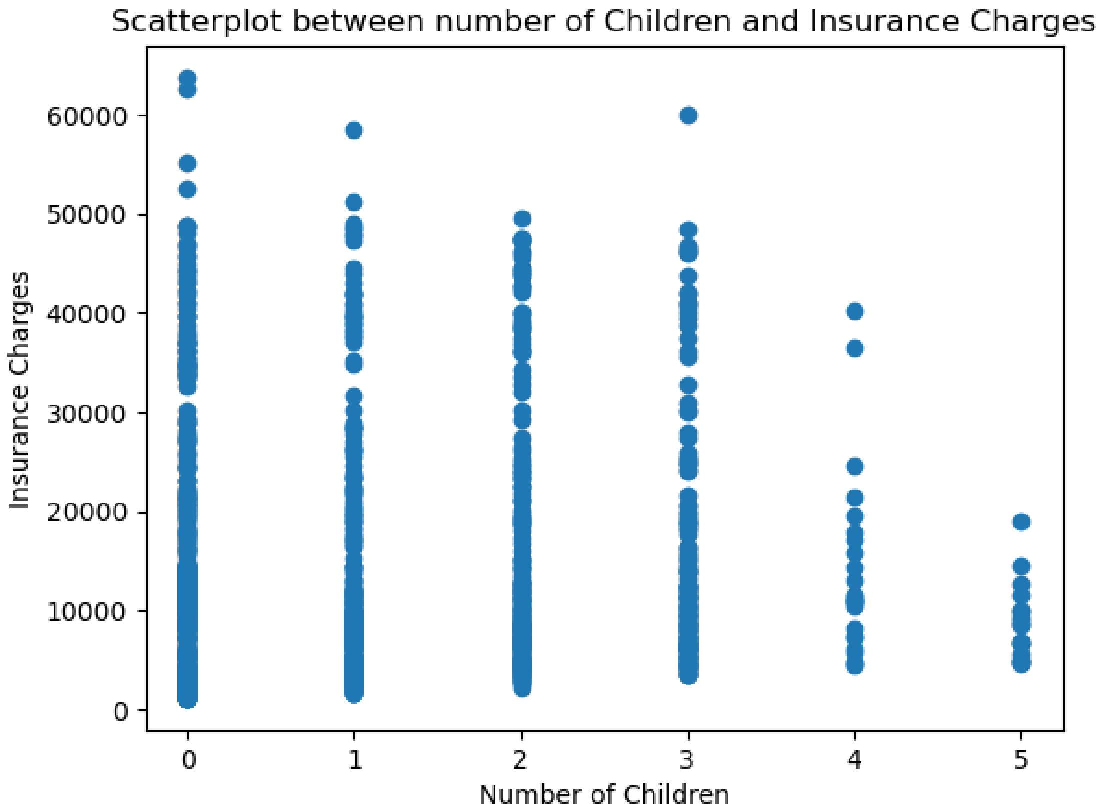
[190]:
```
labels_sample = km_sample.labels_
insurance['label'] = labels_sample
sns.
 ↪scatterplot(x=insurance['bmi'],y=insurance['charges'],hue=insurance['label'],palette='Set1'
plt.title("Scatterplot between BMI and Insurance Charges with 5 labels")
```

[190]: Text(0.5, 1.0, 'Scatterplot between BMI and Insurance Charges with 5 labels')



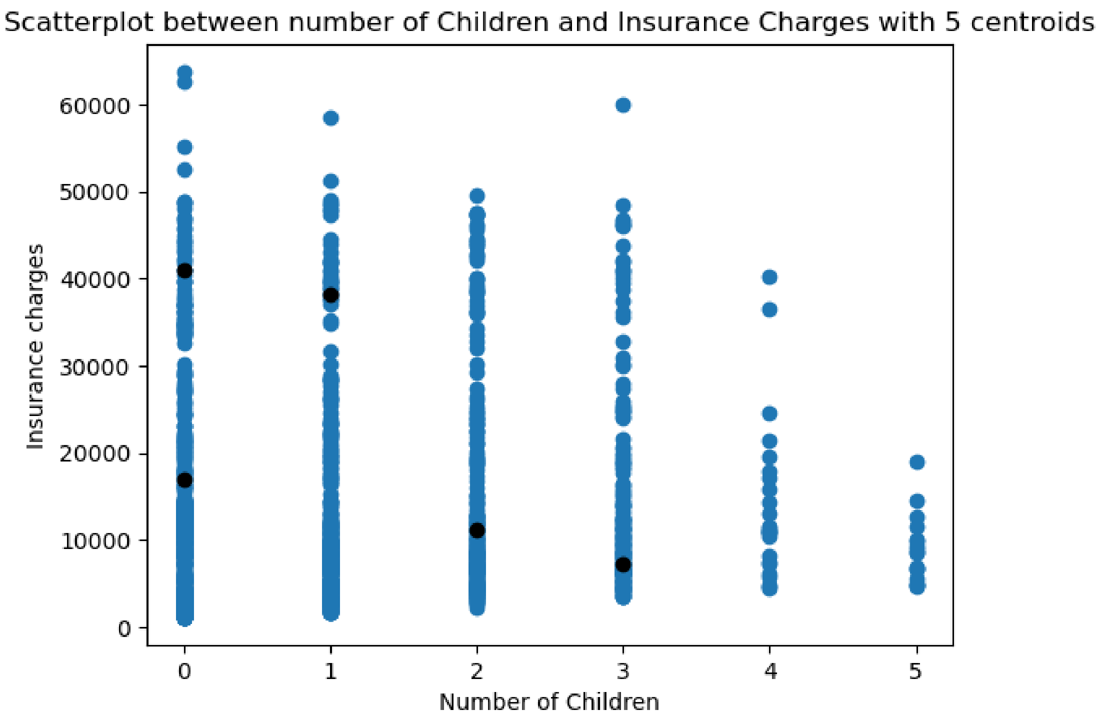
KMeans Clustering study relationship between Children and Insurance Charges.

[191]:
```
plt.scatter(insurance['children'],insurance['charges'])
plt.xlabel('Number of Children')
plt.ylabel('Insurance Charges')
plt.title("Scatterplot between number of Children and Insurance Charges")
plt.show()
```



[192]:

```
K=5
centroids = insurance.sample(n=K)
plt.scatter(insurance['children'],insurance['charges'])
plt.scatter(centroids['children'],centroids['charges'],c='black')
plt.xlabel('Number of Children')
plt.ylabel('Insurance charges')
plt.title("Scatterplot between number of Children and Insurance Charges with 5␣
↪centroids")
plt.show()
```



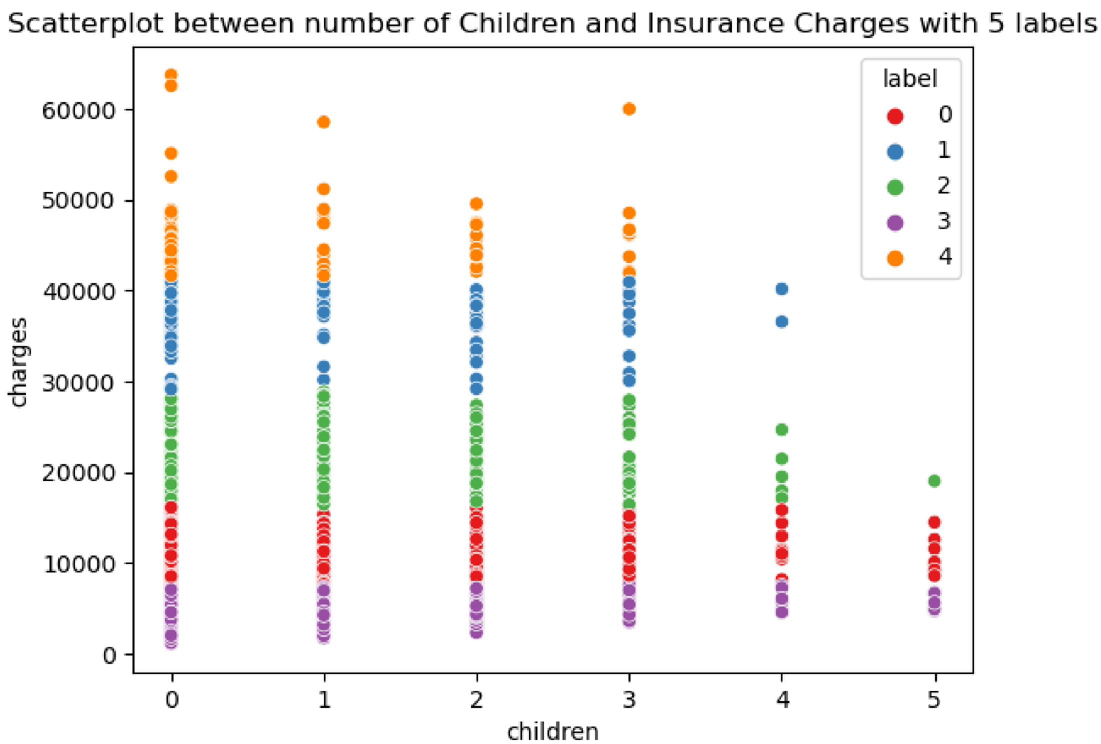
KMeans Clustering, plot between Children and Insurance Charges with 5 centroids.

[193]:
```
km_sample = KMeans(n_clusters=5)
km_sample.fit(insurance[['children','charges']])
```

[193]:
```
KMeans(n_clusters=5)
```

[194]:
```
labels_sample = km_sample.labels_
insurance['label'] = labels_sample
sns.
  ↪scatterplot(x=insurance['children'],y=insurance['charges'],hue=insurance['label'],palette='
plt.title("Scatterplot between number of Children and Insurance Charges with 5␣
  ↪labels")
```

[194]:
```
Text(0.5, 1.0, 'Scatterplot between number of Children and Insurance Charges
with 5 labels')
```

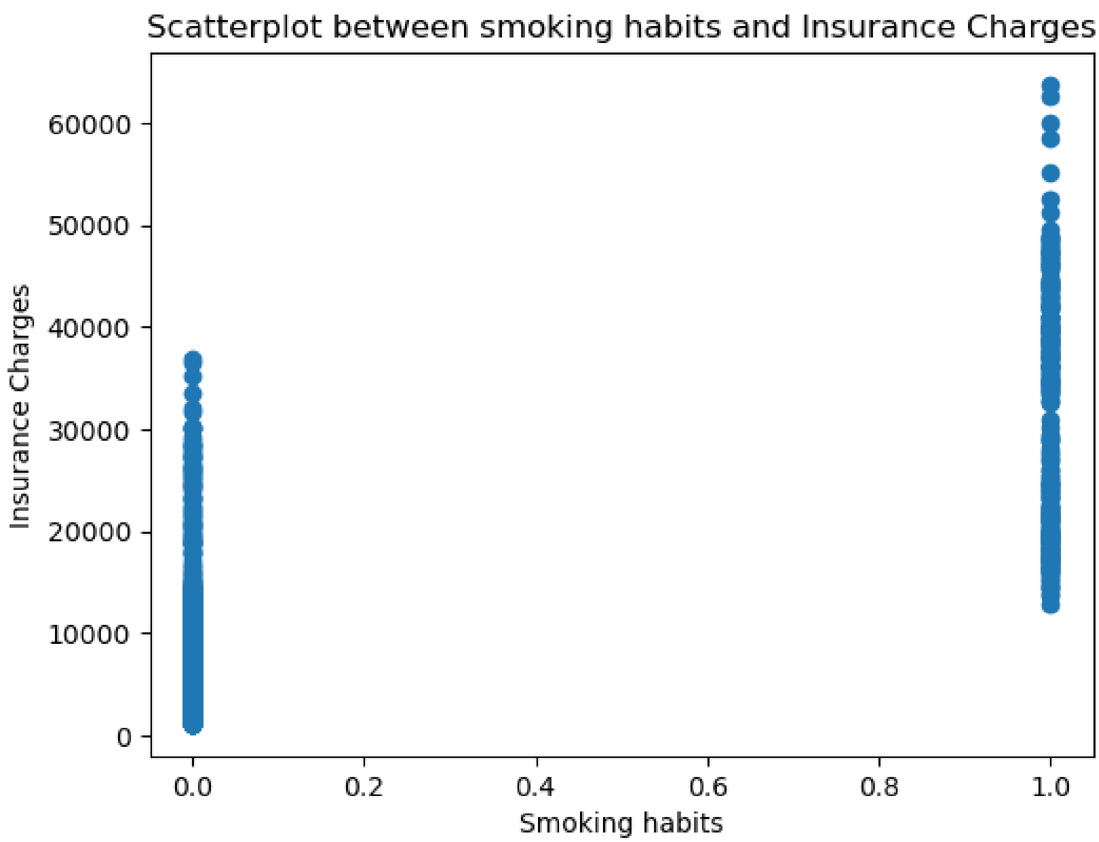


KMeans Clustering study relationship between Smoking and Insurance Charges.

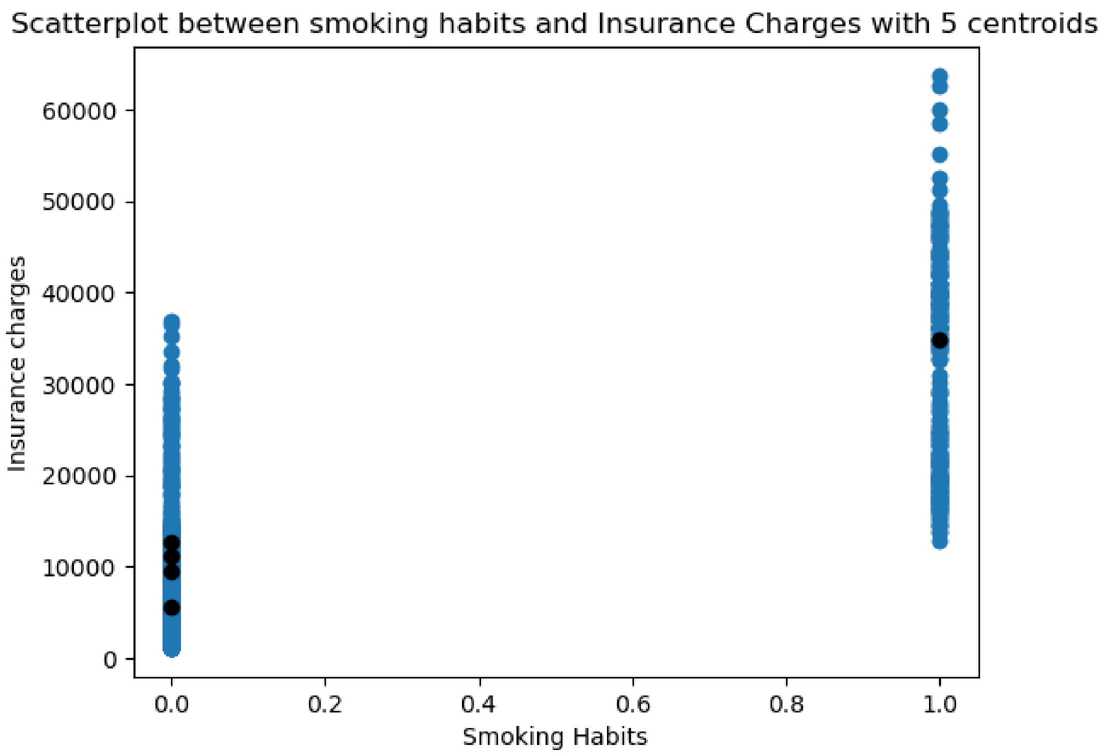
[195]:
```
#label_encoder object knows how to understand object features.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'smoking'.
insurance1['smoker']= label_encoder.fit_transform(insurance['smoker'])
print(insurance1.head())
```

```
   age     sex     bmi  children  smoker     region      charges
0   19  female  27.900         0       1  southwest  16884.92400
1   18    male  33.770         1       0  southeast   1725.55230
2   28    male  33.000         3       0  southeast   4449.46200
3   33    male  22.705         0       0  northwest  21984.47061
4   32    male  28.880         0       0  northwest   3866.85520
```

[196]:
```
plt.scatter(insurance1['smoker'],insurance1['charges'])
plt.xlabel('Smoking habits')
plt.ylabel('Insurance Charges')
plt.title("Scatterplot between smoking habits and Insurance Charges")
plt.show()
```



[197]:
```
K=5
centroids = insurance1.sample(n=K)
plt.scatter(insurance1['smoker'],insurance1['charges'])
plt.scatter(centroids['smoker'],centroids['charges'],c='black')
plt.xlabel('Smoking Habits')
plt.ylabel('Insurance charges')
plt.title("Scatterplot between smoking habits and Insurance Charges with 5␣
 ↪centroids")
plt.show()
```



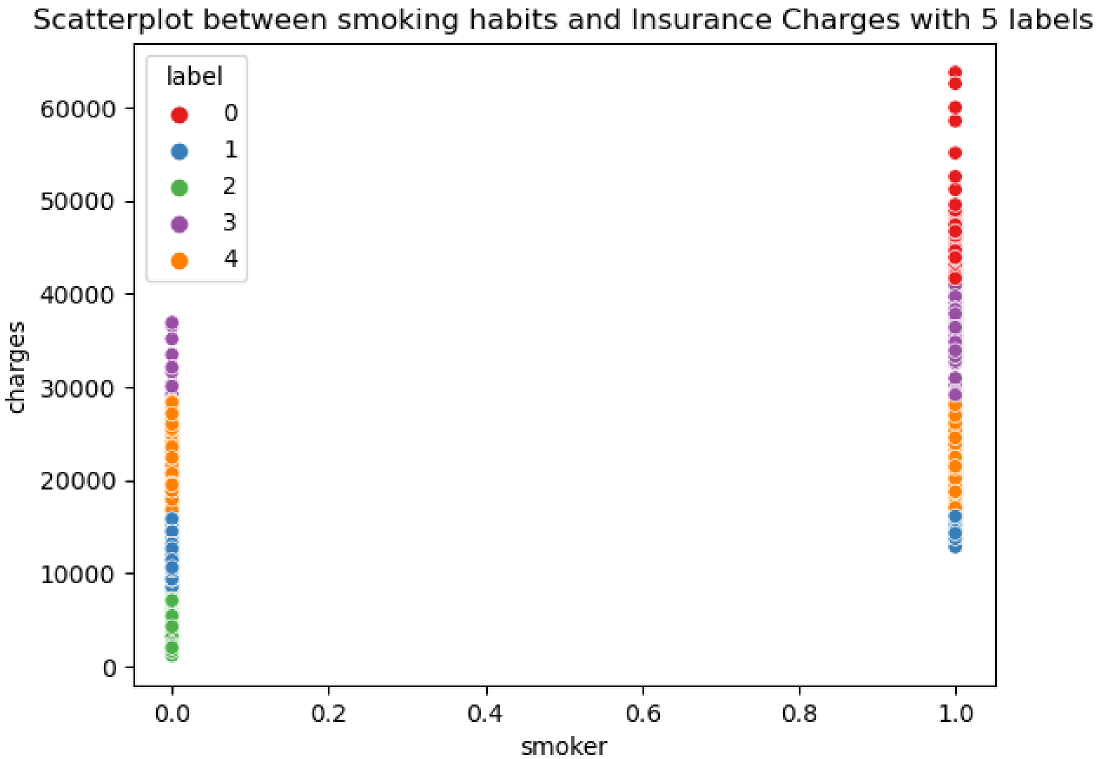
KMeans Clustering, plot between Smoking Habits and Insurance Charges with 5 centroids.

[198]:
```
km_sample = KMeans(n_clusters=5)
km_sample.fit(insurance1[['smoker','charges']])
```

[198]:
```
KMeans(n_clusters=5)
```

[199]:
```
labels_sample = km_sample.labels_
insurance['label'] = labels_sample
sns.
 ↪scatterplot(x=insurance['smoker'],y=insurance['charges'],hue=insurance['label'],palette='Se
plt.title("Scatterplot between smoking habits and Insurance Charges with 5␣
 ↪labels")
```

[199]:
```
Text(0.5, 1.0, 'Scatterplot between smoking habits and Insurance Charges with 5
labels')
```


Scatterplot between smoking habits and Insurance Charges with 5 labels
label
0
1
2
3
4
60000
50000
40000
30000
20000
10000
0
charges
0.0
0.2
0.4
0.6
0.8
1.0
smoker

```
[ ]:
```