In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing,svm
from sklearn import metrics
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

Data collection

Read the data

In [2]:
```
df=pd.read_csv(r"C:\Users\Sudheer\AppData\Local\Microsoft\Windows\INetCache\IE
df
```

Out[2]:

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

1338 rows × 7 columns

2.Data cleaning and Preprocessing

In [3]:
```
df.info()
```

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

In [4]:
```
df.columns
```

Out[4]:
```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype
='object')
```

In [5]:
```
df.head()
```

Out[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

In [6]:
```
df.tail()
```

Out[6]:

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

In [7]:
```
df.shape
```

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

In [8]:
```
df.describe()
```

Out[8]:

	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

In [9]:
```
df.duplicated().sum()
```

Out[9]: 1

To find unique values

In [10]:
```
df['age'].unique()
df['children'].unique()
df['bmi'].unique()
```

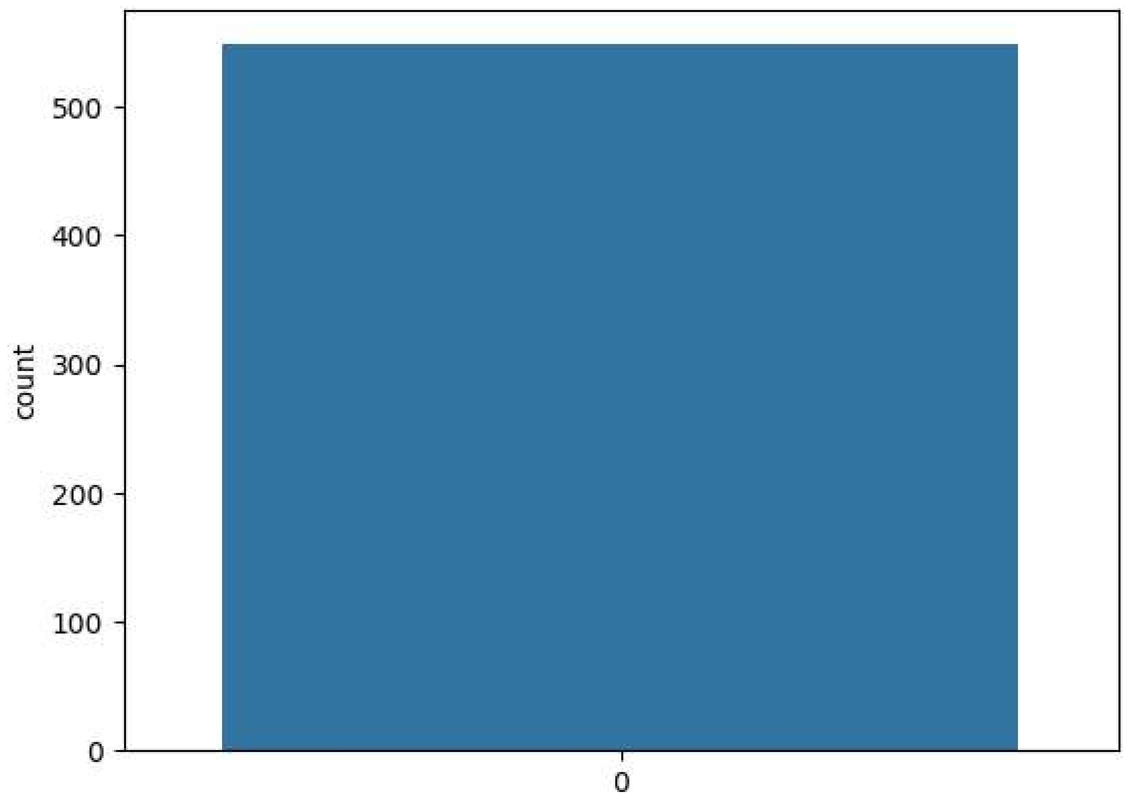
Out[10]:
```
array([27.9  , 33.77 , 33.   , 22.705, 28.88 , 25.74 , 33.44 , 27.74 ,
       29.83 , 25.84 , 26.22 , 26.29 , 34.4  , 39.82 , 42.13 , 24.6  ,
       30.78 , 23.845, 40.3  , 35.3  , 36.005, 32.4  , 34.1  , 31.92 ,
       28.025, 27.72 , 23.085, 32.775, 17.385, 36.3  , 35.6  , 26.315,
       28.6  , 28.31 , 36.4  , 20.425, 32.965, 20.8  , 36.67 , 39.9  ,
       26.6  , 36.63 , 21.78 , 30.8  , 37.05 , 37.3  , 38.665, 34.77 ,
       24.53 , 35.2  , 35.625, 33.63 , 28.   , 34.43 , 28.69 , 36.955,
       31.825, 31.68 , 22.88 , 37.335, 27.36 , 33.66 , 24.7  , 25.935,
       22.42 , 28.9  , 39.1  , 36.19 , 23.98 , 24.75 , 28.5  , 28.1  ,
       32.01 , 27.4  , 34.01 , 29.59 , 35.53 , 39.805, 26.885, 38.285,
       37.62 , 41.23 , 34.8  , 22.895, 31.16 , 27.2  , 26.98 , 39.49 ,
       24.795, 31.3  , 38.28 , 19.95 , 19.3  , 31.6  , 25.46 , 30.115,
       29.92 , 27.5  , 28.4  , 30.875, 27.94 , 35.09 , 29.7  , 35.72 ,
       32.205, 28.595, 49.06 , 27.17 , 23.37 , 37.1  , 23.75 , 28.975,
       31.35 , 33.915, 28.785, 28.3  , 37.4  , 17.765, 34.7  , 26.505,
       22.04 , 35.9  , 25.555, 28.05 , 25.175, 31.9  , 36.   , 32.49 ,
       25.3  , 29.735, 38.83 , 30.495, 37.73 , 37.43 , 24.13 , 37.145,
       39.52 , 24.42 , 27.83 , 36.85 , 39.6  , 29.8  , 29.64 , 28.215,
       37.   , 33.155, 18.905, 41.47 , 30.3  , 15.96 , 33.345, 37.7  ,
       27.835, 29.2  , 26.41 , 30.69 , 41.895, 30.9  , 32.2  , 32.11 ,
       31.57 , 26.2  , 30.59 , 32.8  , 18.05 , 39.33 , 32.23 , 24.035,
       36.08 , 22.3  , 26.4  , 31.8  , 26.73 , 23.1  , 23.21 , 33.7  ,
       33.25 , 24.64 , 33.88 , 38.06 , 41.91 , 31.635, 36.195, 17.8  ,
       24.51 , 22.22 , 38.39 , 29.07 , 22.135, 26.8  , 30.02 , 35.86 ,
       20.9  , 17.29 , 34.21 , 25.365, 40.15 , 24.415, 25.2  , 26.84 ,
       24.32 , 42.35 , 19.8  , 32.395, 30.2  , 29.37 , 34.2  , 27.455,
       27.55 , 20.615, 24.3  , 31.79 , 21.56 , 28.12 , 40.565, 27.645,
       31.2  , 26.62 , 48.07 , 36.765, 33.4  , 45.54 , 28.82 , 22.99 ,
       27.7  , 25.41 , 34.39 , 22.61 , 37.51 , 38.   , 33.33 , 34.865,
       33.06 , 35.97 , 31.4  , 25.27 , 40.945, 34.105, 36.48 , 33.8  ,
       36.7  , 36.385, 34.5  , 32.3  , 27.6  , 29.26 , 35.75 , 23.18 ,
       25.6  , 35.245, 43.89 , 20.79 , 30.5  , 21.7  , 21.89 , 24.985,
       32.015, 30.4  , 21.09 , 22.23 , 32.9  , 24.89 , 31.46 , 17.955,
       30.685, 43.34 , 39.05 , 30.21 , 31.445, 19.855, 31.02 , 38.17 ,
       20.6  , 47.52 , 20.4  , 38.38 , 24.31 , 23.6  , 21.12 , 30.03 ,
       17.48 , 20.235, 17.195, 23.9  , 35.15 , 35.64 , 22.6  , 39.16 ,
       27.265, 29.165, 16.815, 33.1  , 26.9  , 33.11 , 31.73 , 46.75 ,
       29.45 , 32.68 , 33.5  , 43.01 , 36.52 , 26.695, 25.65 , 29.6  ,
       38.6  , 23.4  , 46.53 , 30.14 , 30.   , 38.095, 28.38 , 28.7  ,
       33.82 , 24.09 , 32.67 , 25.1  , 32.56 , 41.325, 39.5  , 34.3  ,
       31.065, 21.47 , 25.08 , 43.4  , 25.7  , 27.93 , 39.2  , 26.03 ,
       30.25 , 28.93 , 35.7  , 35.31 , 31.   , 44.22 , 26.07 , 25.8  ,
       39.425, 40.48 , 38.9  , 47.41 , 35.435, 46.7  , 46.2  , 21.4  ,
       23.8  , 44.77 , 32.12 , 29.1  , 37.29 , 43.12 , 36.86 , 34.295,
       23.465, 45.43 , 23.65 , 20.7  , 28.27 , 35.91 , 29.   , 19.57 ,
       31.13 , 21.85 , 40.26 , 33.725, 29.48 , 32.6  , 37.525, 23.655,
       37.8  , 19.   , 21.3  , 33.535, 42.46 , 38.95 , 36.1  , 29.3  ,
       39.7  , 38.19 , 42.4  , 34.96 , 42.68 , 31.54 , 29.81 , 21.375,
       40.81 , 17.4  , 20.3  , 18.5  , 26.125, 41.69 , 24.1  , 36.2  ,
       40.185, 39.27 , 34.87 , 44.745, 29.545, 23.54 , 40.47 , 40.66 ,
       36.6  , 35.4  , 27.075, 28.405, 21.755, 40.28 , 30.1  , 32.1  ,
       23.7  , 35.5  , 29.15 , 27.   , 37.905, 22.77 , 22.8  , 34.58 ,
       27.1  , 19.475, 26.7  , 34.32 , 24.4  , 41.14 , 22.515, 41.8  ,
       26.18 , 42.24 , 26.51 , 35.815, 41.42 , 36.575, 42.94 , 21.01 ,
       24.225, 17.67 , 31.5  , 31.1  , 32.78 , 32.45 , 50.38 , 47.6  ,
       25.4  , 29.9  , 43.7  , 24.86 , 28.8  , 29.5  , 29.04 , 38.94 ,
       44.   , 20.045, 40.92 , 35.1  , 29.355, 32.585, 32.34 , 39.8  ,
```

```
       24.605, 33.99 , 28.2  , 25.   , 33.2  , 23.2  , 20.1  , 32.5  ,
       37.18 , 46.09 , 39.93 , 35.8  , 31.255, 18.335, 42.9  , 26.79 ,
       39.615, 25.9  , 25.745, 28.16 , 23.56 , 40.5  , 35.42 , 39.995,
       34.675, 20.52 , 23.275, 36.29 , 32.7  , 19.19 , 20.13 , 23.32 ,
       45.32 , 34.6  , 18.715, 21.565, 23.   , 37.07 , 52.58 , 42.655,
       21.66 , 32.   , 18.3  , 47.74 , 22.1  , 19.095, 31.24 , 29.925,
       20.35 , 25.85 , 42.75 , 18.6  , 23.87 , 45.9  , 21.5  , 30.305,
       44.88 , 41.1  , 40.37 , 28.49 , 33.55 , 40.375, 27.28 , 17.86 ,
       33.3  , 39.14 , 21.945, 24.97 , 23.94 , 34.485, 21.8  , 23.3  ,
       36.96 , 21.28 , 29.4  , 27.3  , 37.9  , 37.715, 23.76 , 25.52 ,
       27.61 , 27.06 , 39.4  , 34.9  , 22.   , 30.36 , 27.8  , 53.13 ,
       39.71 , 32.87 , 44.7  , 30.97 ])
```

3.Data Visualization:Visualize the unique counts

In [11]:
```
sns.countplot(df['bmi'].unique())
```

Out[11]: <Axes: ylabel='count'>

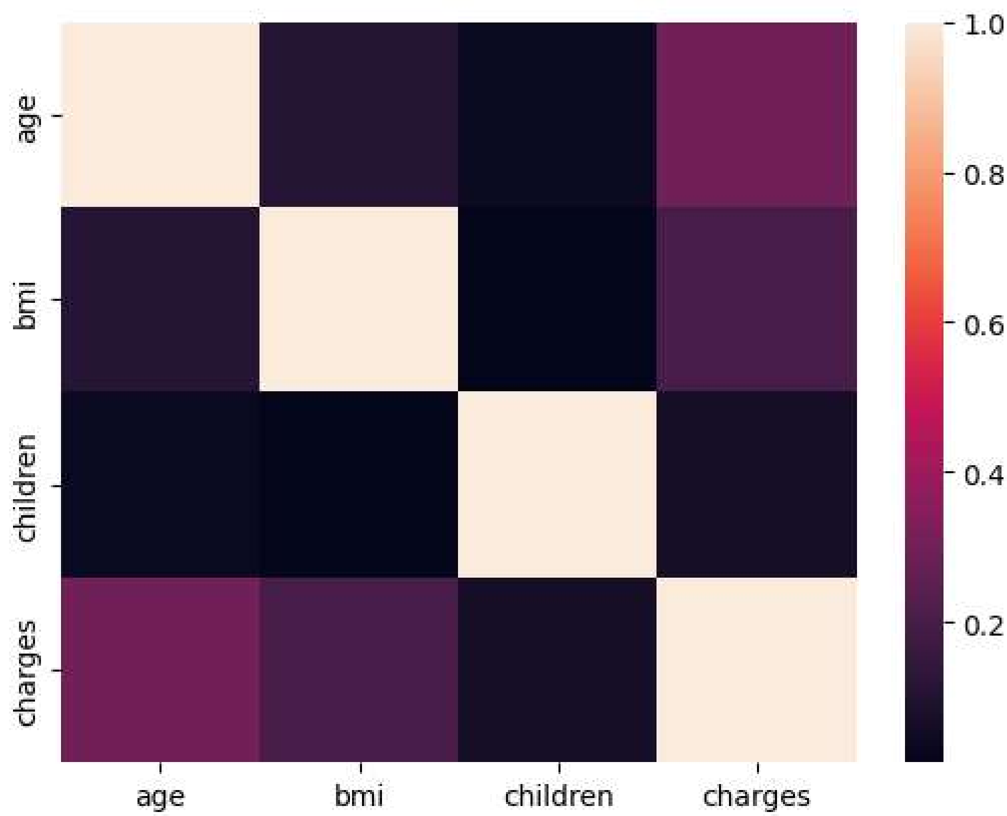


In [12]:
```
df.isnull().sum()
```

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

In [13]:
```
Insurancedf=df[['age','bmi', 'children','charges']]
sns.heatmap(Insurancedf.corr())
```

Out[13]: <Axes: >



to check the null values

In [14]:
```
df.replace(np.nan,'0',inplace = True)
```

In [15]:
```
df.isnull().sum()
```

Out[15]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

Feature Scaling:To split the data into train and test

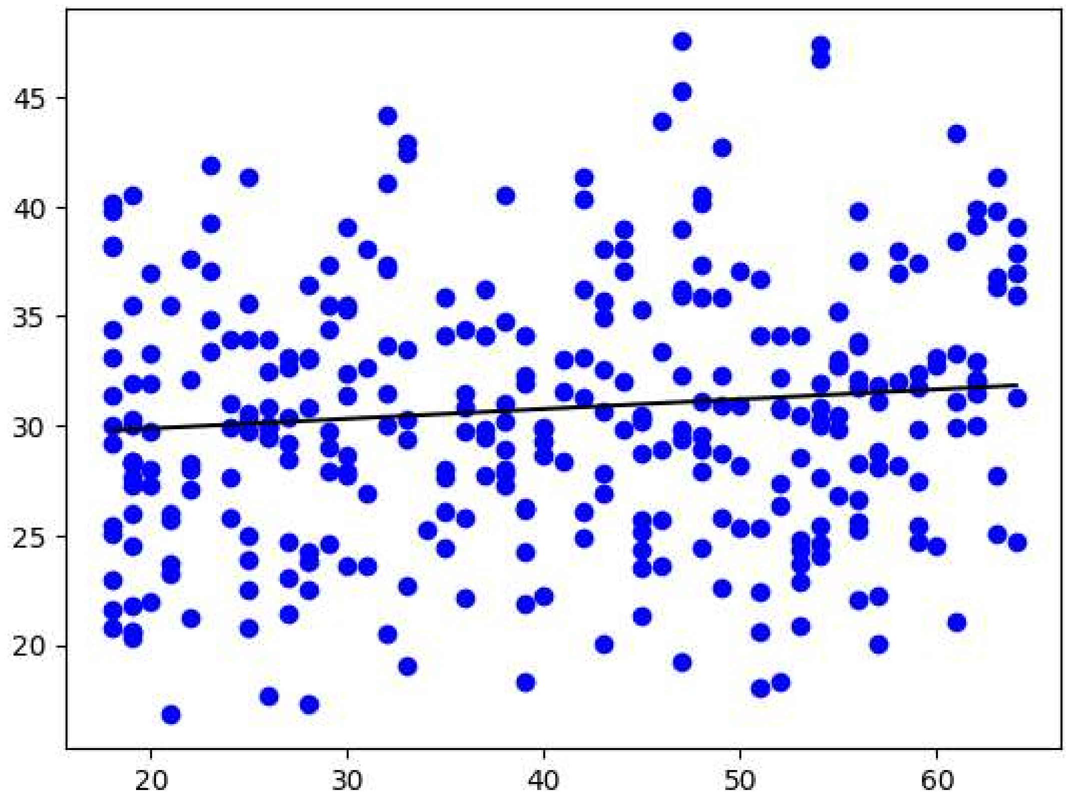
data

In [16]:
```
x=np.array(df['age']).reshape(-1,1)
y=np.array(df['bmi']).reshape(-1,1)
```

In [17]:
```
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.25)
reg=LinearRegression()
reg.fit(X_train,y_train)
print(reg.score(X_test,y_test))
```

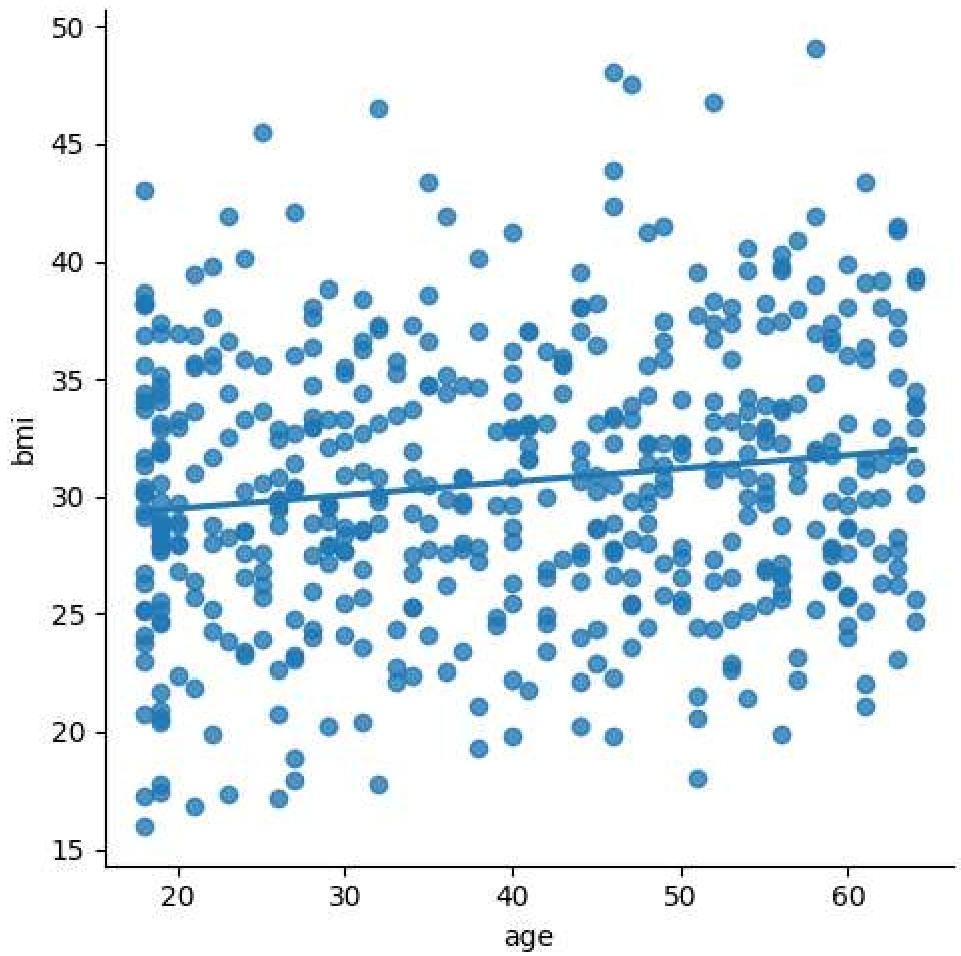
```
0.015220865325818234
```

In [18]:
```
y_pred=reg.predict(X_test)
plt.scatter(X_test,y_test,color='b')
plt.plot(X_test,y_pred,color='k')
plt.show()
```



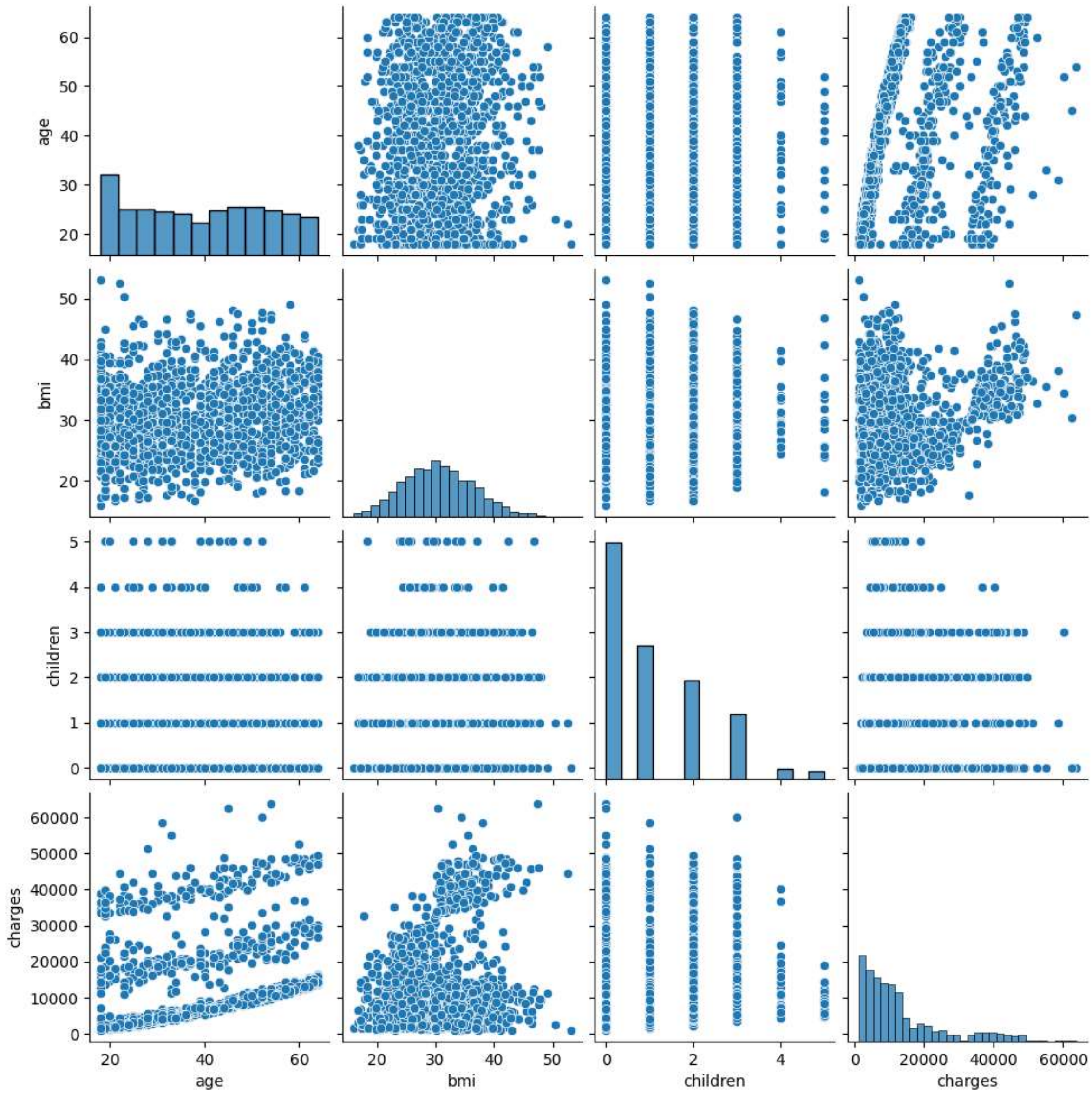
In [19]:
```
df500=df[:][:500]
sns.lmplot(x="age",y="bmi",data=df500,order=1,ci=None)
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x24e7eaf44f0>



In [20]:
```
sns.pairplot(df)
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x24e7eb493f0>



4.Data Modelling:Using Linear,Ridge and Lasso

In [21]:
```
from sklearn.linear_model import Ridge,RidgeCV,Lasso
from sklearn.preprocessing import StandardScaler
```

In [22]:
```
T={"sex":{'male':1,'female':2}}
df=df.replace(T)
df
```

Out[22]:

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

1338 rows × 7 columns

In [24]:
```
features = df.columns[0:2]
target = df.columns[-1]
#X and y values
X = df[features].values
y = df[target].values
#splot
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, randc
print("The dimension of X_train is {}".format(X_train.shape))
print("The dimension of X_test is {}".format(X_test.shape))
#Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
The dimension of X_train is (936, 2)
The dimension of X_test is (402, 2)
```

In [25]:
```
#Model
lr = LinearRegression()
#Fit model
lr.fit(X_train, y_train)
#predict
#prediction = lr.predict(X_test)
#actual
actual = y_test
train_score_lr = lr.score(X_train, y_train)
test_score_lr = lr.score(X_test, y_test)
print("\nRidg Model:\n")
print("The train score for lr model is {}".format(train_score_lr))
print("The test score for lr model is {}".format(test_score_lr))
```

```
Ridg Model:

The train score for lr model is 0.08144731818197626
The test score for lr model is 0.1022033122179885
```

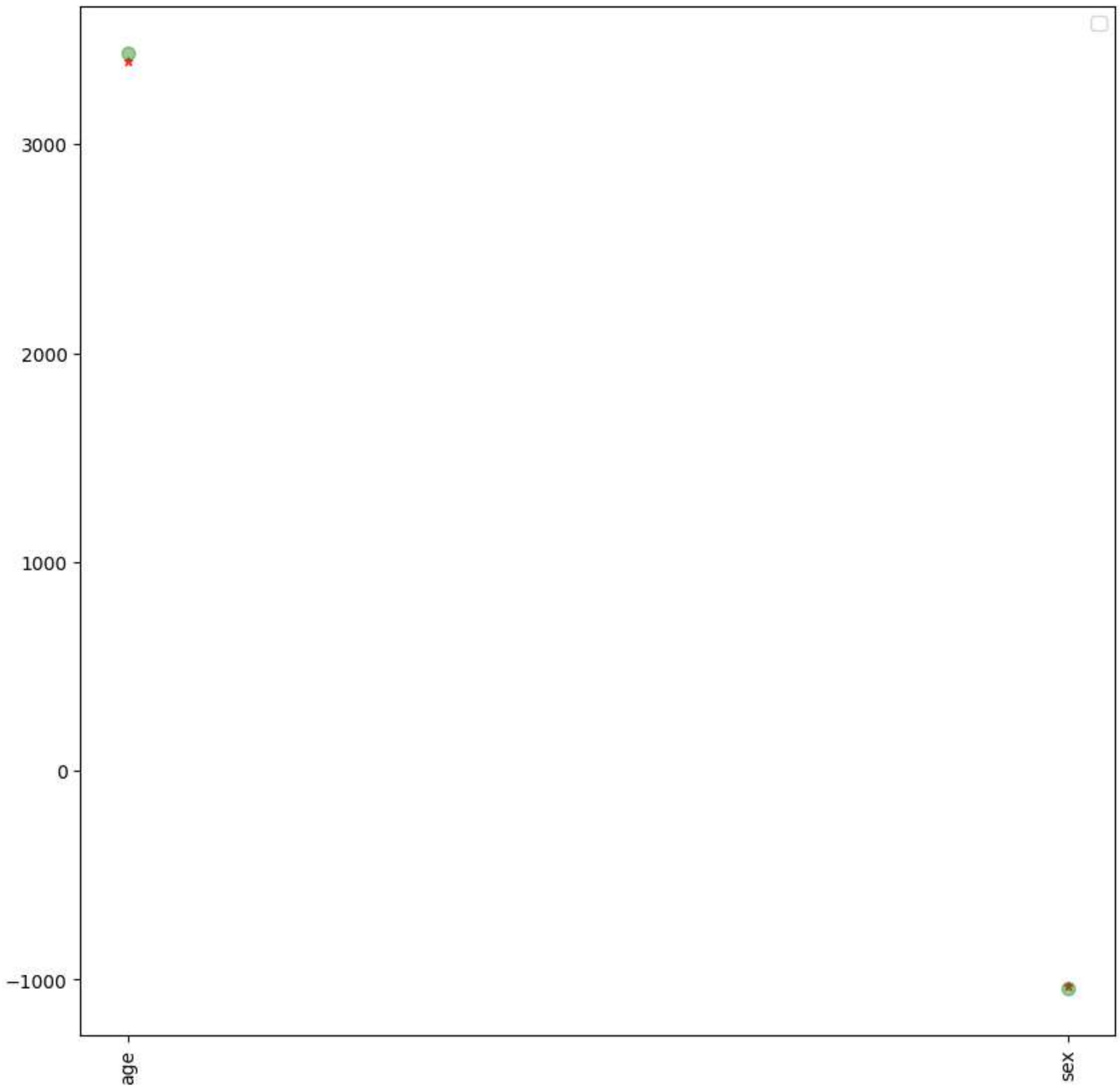
In [26]:
```
#Ridge Regression Model
ridgeReg = Ridge(alpha=10)
ridgeReg.fit(X_train,y_train)
#train and test scorefor ridge regression
train_score_ridge = ridgeReg.score(X_train, y_train)
test_score_ridge = ridgeReg.score(X_test, y_test)
print("\nRidge Model:\n")
print("The train score for ridge model is {}".format(train_score_ridge))
print("The test score for ridge model is {}".format(test_score_ridge))
```

```
Ridge Model:

The train score for ridge model is 0.08143796463046804
The test score for ridge model is 0.10202509697425621
```

In [29]:
```
plt.figure(figsize = (10, 10))
plt.plot(features,ridgeReg.coef_,alpha=0.7,linestyle='none',marker='*',markers
#lot(rr100.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blu
plt.plot(features,lr.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,
plt.xticks(rotation = 90)
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argumen
t.
```



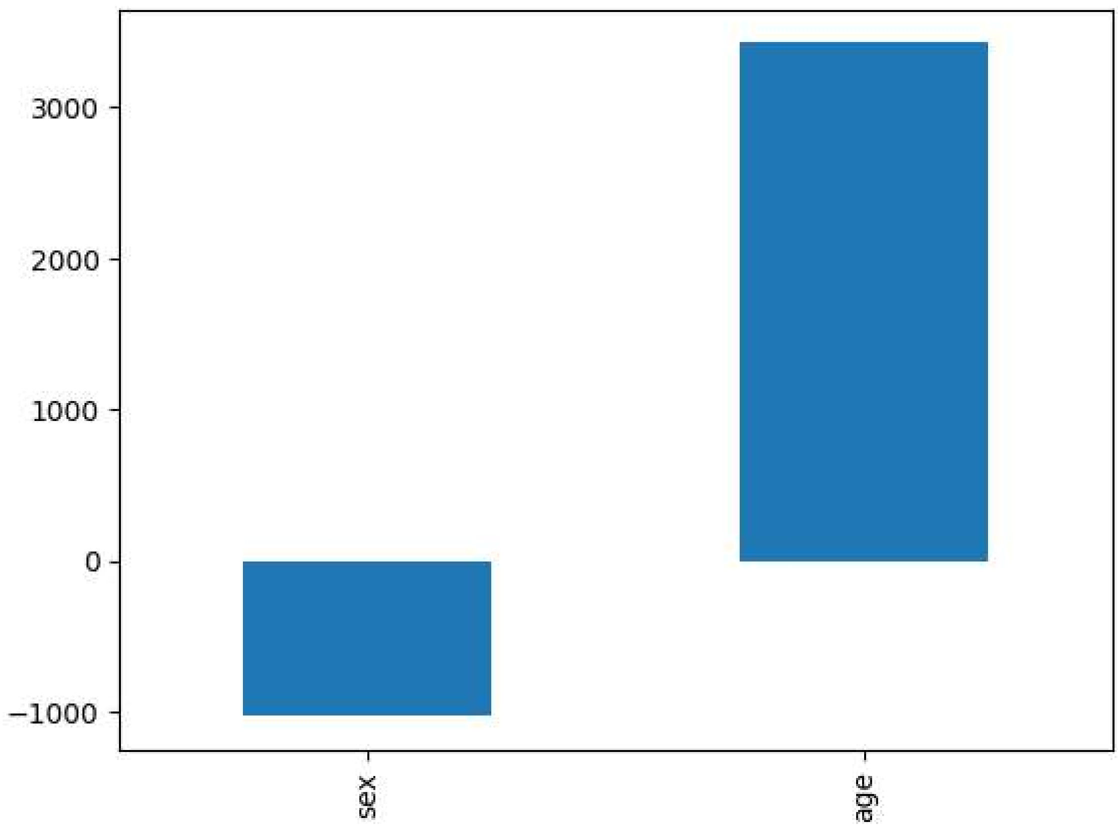
In [30]:
```
#Lasso regression model
print("\nLasso Model: \n")
lasso = Lasso(alpha = 10)
lasso.fit(X_train,y_train)
train_score_ls =lasso.score(X_train,y_train)
test_score_ls =lasso.score(X_test,y_test)
print("The train score for ls model is {}".format(train_score_ls))
print("The test score for ls model is {}".format(test_score_ls))
```

```
Lasso Model:

The train score for ls model is 0.08144600503720834
The test score for ls model is 0.10226046691368151
```

In [31]:
```
pd.Series(lasso.coef_, features).sort_values(ascending = True).plot(kind = "ba
```

Out[31]: <Axes: >



In [34]:
```
#Using the linear CV model
from sklearn.linear_model import LassoCV
#Lasso Cross validation
lasso_cv = LassoCV(alphas = [0.0001, 0.001,0.01, 0.1, 1, 10], random_state=0).
#score
print(lasso_cv.score(X_train, y_train))
print(lasso_cv.score(X_test, y_test))
```

```
0.08144600503720834
0.10226046691368151
```

In [35]:
```
from sklearn.linear_model import ElasticNet
regr=ElasticNet()
regr.fit(X,y)
print(regr.coef_)
print(regr.intercept_)
regr.score(X,y)
```

```
[ 257.44760657 -511.96287073]
3941.933287820535
```

Out[35]:
```
0.09164498902013407
```

In [37]:
```
y_pred_elastic=regr.predict(X_train)
```

In [38]:
```
mean_squared_error=np.mean((y_pred_elastic-y_train)**2)
print("Mean Squared Error on test set",mean_squared_error)
```

```
Mean Squared Error on test set 251337238.06352755
```

5.Data Prediction&Evaluation

In [39]:
```
prediction=lr.predict(X_test)
```

model=LinearRegression() model.fit(X_train,y_train) y_pred=model.predict(X_test) r2=r2_score(y_test,y_pred) print("R2_score:",r2)

To find Error

In [41]:
```
print('MAE:',metrics.mean_absolute_error(y_test,prediction))
print('MSE:',metrics.mean_squared_error(y_test,prediction))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
MAE: 8564.630066831065
MSE: 109448853.74996018
RMSE: 10461.780620427871
```

Cross Validation

In [42]:
```
ridge_cv=RidgeCV(alphas=[1,10,100]).fit(X_train,y_train)
print("THE TRAIN SCORE FOR RIDGE MODEL IS : {}".format(ridge_cv.score(X_train,
print("THE TEST SCORE FOR RIDGE MODEL IS :{}".format(ridge_cv.score(X_test,y_t
```

```
THE TRAIN SCORE FOR RIDGE MODEL IS : 0.08143796463046815
THE TEST SCORE FOR RIDGE MODEL IS :0.10202509697425932
```

In [43]:
```
lasso_cv=LassoCV(alphas=[1,10,100]).fit(X_train,y_train)
print("THE TRAIN SCORE FOR RIDGE MODEL IS : {}".format(lasso_cv.score(X_train,
print("THE TEST SCORE FOR RIDGE MODEL IS :{}".format(lasso_cv.score(X_test,y_t
```

```
THE TRAIN SCORE FOR RIDGE MODEL IS : 0.08144600503720834
THE TEST SCORE FOR RIDGE MODEL IS :0.10226046691368151
```

In [44]:
```
import pickle
```

In [45]:
```
filename="prediction"
pickle.dump(lr,open(filename,'wb'))
```

Conclusion

In the above project I got same value for four models :Linear for 0.08,0.102;Ridge for 0.08,0.102;Lasso for 0.08,0.102;Elastic for 0.08,0.102.So here I concluded there is no best fit model for the Health Insurance data set. I am going on Logistic Regression

Logistic Regression

In [46]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

In [47]:
```
df=pd.read_csv(r"C:\Users\Sudheer\AppData\Local\Microsoft\Windows\INetCache\IE
df
```

Out[47]:

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

1338 rows × 7 columns

In [48]:
```
pd.set_option('display.max_row',10000000000)
pd.set_option('display.max_columns',10000000000)
pd.set_option('display.width',95)
print('This dataFrame has %d rows and %d columns'%(df.shape))
```

```
This dataFrame has 1338 rows and 7 columns
```

In [49]:
```
T={"smoker":{'yes':1,'no':2}}
df=df.replace(T)
df
```

Out[49]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	1	southwest	16884.924000
1	18	male	33.770	1	2	southeast	1725.552300
2	28	male	33.000	3	2	southeast	4449.462000
3	33	male	22.705	0	2	northwest	21984.470610
4	32	male	28.880	0	2	northwest	3866.855200
5	31	female	25.740	0	2	southeast	3756.621600
6	46	female	33.440	1	2	southeast	8240.589600
7	37	female	27.740	3	2	northwest	7281.505600
8	37	male	29.830	2	2	northeast	6406.410700
9	60	female	25.840	0	2	northwest	28923.136920
10	25	male	26.220	0	2	northeast	2721.320800
11	62	female	26.290	0	1	southeast	27808.725100

In [50]:
```
T={"sex":{'male':1,'female':2}}
df=df.replace(T)
df
```

Out[50]:

	age	sex	bmi	children	smoker	region	charges
0	19	2	27.900	0	1	southwest	16884.924000
1	18	1	33.770	1	2	southeast	1725.552300
2	28	1	33.000	3	2	southeast	4449.462000
3	33	1	22.705	0	2	northwest	21984.470610
4	32	1	28.880	0	2	northwest	3866.855200
5	31	2	25.740	0	2	southeast	3756.621600
6	46	2	33.440	1	2	southeast	8240.589600
7	37	2	27.740	3	2	northwest	7281.505600
8	37	1	29.830	2	2	northeast	6406.410700
9	60	2	25.840	0	2	northwest	28923.136920
10	25	1	26.220	0	2	northeast	2721.320800
11	62	2	26.290	0	1	southeast	27808.725100

In [86]:
```
features_matrix=df.iloc[:,0:4]
target_vector=df.iloc[:,-3]
print('The Features Matrix Has %d Rows And %d Columns(s)'%(features_matrix.sha
print('The Features Matrix Has %d Rows And %d Columns(s)'%(np.array(target_vec
```

```
The Features Matrix Has 1338 Rows And 4 Columns(s)
The Features Matrix Has 1338 Rows And 1 Columns(s)
```

In [54]:
```
features_matrix_standardized=StandardScaler().fit_transform(features_matrix)
```

In [55]:
```
algorithm=LogisticRegression(max_iter=100000)
```

In [56]:
```
Logistic_Regression_Model=algorithm.fit(features_matrix_standardized,target_ve
```

In [57]:
```
observation=[[0,0.01,0.0003,0.0004]]
```

In [58]:
```
predictions=Logistic_Regression_Model.predict(observation)
```

In [59]:
```
print('The Model Predicted The Obsevation To Belong To Class %s'%(predictions)
```

```
The Model Predicted The Obsevation To Belong To Class [2]
```

In [80]:
```
print(""" The Model Says The Probability Of The Observation We Passed Belongin
print()
print(""" The Model Says The Probability Of The Observation We Passed Belongin
print()
```

```
 The Model Says The Probability Of The Observation We Passed Belonging To Cla
ss [2]

 The Model Says The Probability Of The Observation We Passed Belonging To Cla
ss [2]
```

In [62]:
```
features = df.columns[0:2]
target = df.columns[-2]
#X and y values
X = df[features].values
y = df[target].values
#splot
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
print("The dimension of X_train is {}".format(X_train.shape))
print("The dimension of X_test is {}".format(X_test.shape))
#Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
The dimension of X_train is (936, 2)
The dimension of X_test is (402, 2)
```

legr = LogisticRegression() #Fit model legr.fit(X_train, y_train) #predict #prediction = lr.predict(X_test) #actual actual = y_test train_score_legr = legr.score(X_train, y_train) test_score_legr = legr.score(X_test, y_test) print("\nRidg Model:\n") print("The train score for legr model is {}".format(train_score_legr)) print("The test score for legr model is {}".format(test_score_legr))

In [64]:
```
x=np.array(df['age']).reshape(-1,1)
y=np.array(df['smoker']).reshape(-1,1)
```

In [65]:
```
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.30)
lerg=LogisticRegression()
lerg.fit(X_train,y_train)
print(lerg.score(X_test,y_test))
```

```
0.8059701492537313

C:\Users\Sudheer\AppData\Local\Programs\Python\Python310\lib\site-packages\sk
learn\utils\validation.py:1143: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_sampl
es, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Decision Tree Regression

In [66]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

In [67]:
```
df=pd.read_csv(r"C:\Users\Sudheer\AppData\Local\Microsoft\Windows\INetCache\IE
df
```

Out[67]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.924000
1	18	male	33.770	1	no	southeast	1725.552300
2	28	male	33.000	3	no	southeast	4449.462000
3	33	male	22.705	0	no	northwest	21984.470610
4	32	male	28.880	0	no	northwest	3866.855200
5	31	female	25.740	0	no	southeast	3756.621600
6	46	female	33.440	1	no	southeast	8240.589600
7	37	female	27.740	3	no	northwest	7281.505600
8	37	male	29.830	2	no	northeast	6406.410700
9	60	female	25.840	0	no	northwest	28923.136920
10	25	male	26.220	0	no	northeast	2721.320800
11	62	female	26.290	0	yes	southeast	27808.725100

In [68]:
```
df.info()
```

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

In [69]:
```
df['sex'].value_counts()
```

Out[69]:
```
sex
male      676
female    662
Name: count, dtype: int64
```

In [70]:
```
df['bmi'].value_counts()
```

Out[70]:
```
bmi
32.300    13
28.310     9
30.495     8
30.875     8
31.350     8
30.800     8
34.100     8
28.880     8
33.330     7
35.200     7
25.800     7
32.775     7
27.645     7
32.110     7
38.060     7
25.460     7
30.590     7
27.360     7
```

In [71]:
```
converter={"sex":{"male":1,"female":2}}
df=df.replace(converter)
df
```

Out[71]:

	age	sex	bmi	children	smoker	region	charges
0	19	2	27.900	0	yes	southwest	16884.924000
1	18	1	33.770	1	no	southeast	1725.552300
2	28	1	33.000	3	no	southeast	4449.462000
3	33	1	22.705	0	no	northwest	21984.470610
4	32	1	28.880	0	no	northwest	3866.855200
5	31	2	25.740	0	no	southeast	3756.621600
6	46	2	33.440	1	no	southeast	8240.589600
7	37	2	27.740	3	no	northwest	7281.505600
8	37	1	29.830	2	no	northeast	6406.410700
9	60	2	25.840	0	no	northwest	28923.136920
10	25	1	26.220	0	no	northeast	2721.320800
11	62	2	26.290	0	yes	southeast	27808.725100

In [72]:
```
x=["sex","age","bmi"]
y=["1","2"]
all_inputs=df[x]
all_classes=df["region"]
```

In [73]:
```
(x_train,x_test,y_train,y_test)=train_test_split(all_inputs,all_classes,test_s
```

In [74]:
```
clf=DecisionTreeClassifier(random_state=0)
```

In [75]:
```
clf.fit(x_train,y_train)
```

Out[75]:
```
DecisionTreeClassifier(random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [76]:
```
score=clf.score(x_test,y_test)
print(score)
```

```
0.30044843049327335
```

Conclusion:

n the above project I have done four models,I got same value for four models :Linear for 0.08,0.102;Ridge for 0.08,0.102;Lasso for 0.08,0.102;Elastic for 0.08,0.102.So here I concluded there is no best fit model for the Health Insurance data set. I am going on Logistic Regression.In the Logistic Regression I got accuracy 81,In Decision Tree Regression I got accuracy 31.So,I Concluded That LogisticRegression is the best fit model for the Health Insurance

In [ ]:
```

```