Problem statement:

    To check which model is best suitable for the dataset insurance

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing,svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

Data cleaning and Preprocessing

In [2]:

```
df=pd.read_csv(r"C:\Users\DELL\Downloads\insurance.csv")
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

In [3]:

```
df.head()
```

Out[3]:

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

In [4]:

```
df.tail()
```

Out[4]:

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

In [5]:

```
df.shape
```

Out[5]:

```
(1338, 7)
```

In [6]:

```
df.size
```

Out[6]:

```
9366
```

In [7]:

```
df.columns
```

Out[7]:

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

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.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 [10]:

```
convert={"sex":{"male":0,"female":1}}
df=df.replace(convert)
df
```

Out[10]:

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

1338 rows × 7 columns

In [11]:

```
convert={"smoker":{"yes":1,"no":0}}
df=df.replace(convert)
df
```

Out[11]:

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

1338 rows × 7 columns

In [12]:

```
from sklearn.preprocessing import StandardScaler
```

In [13]:

```
convert={"region":{"southwest":0,"southeast":1,"northwest":3,"northeast":4}}
df=df.replace(convert)
df
```

Out[13]:

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

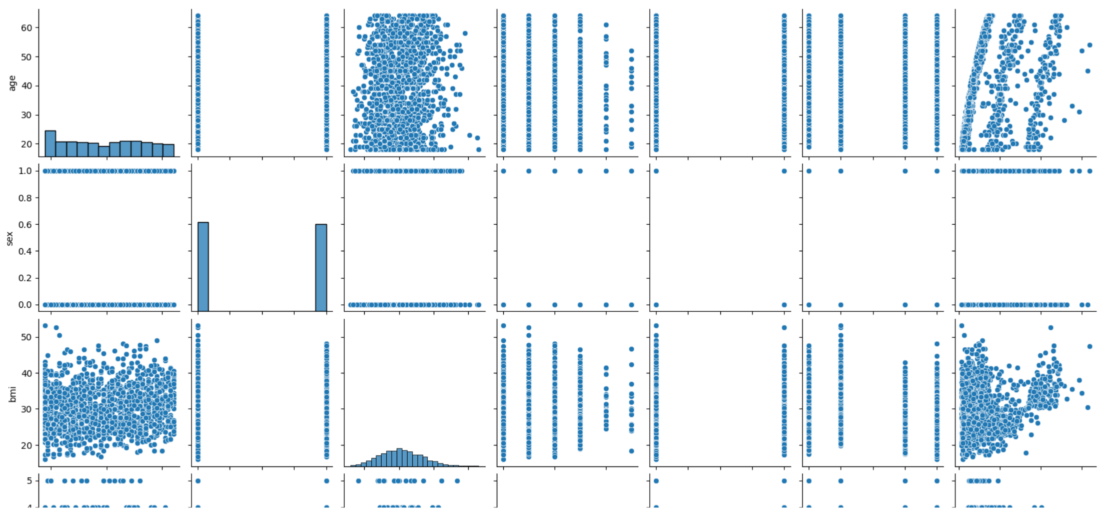
1338 rows × 7 columns

In [14]:

```
sns.pairplot(df)
```

Out[14]:

<seaborn.axisgrid.PairGrid at 0x1d5104a98d0>

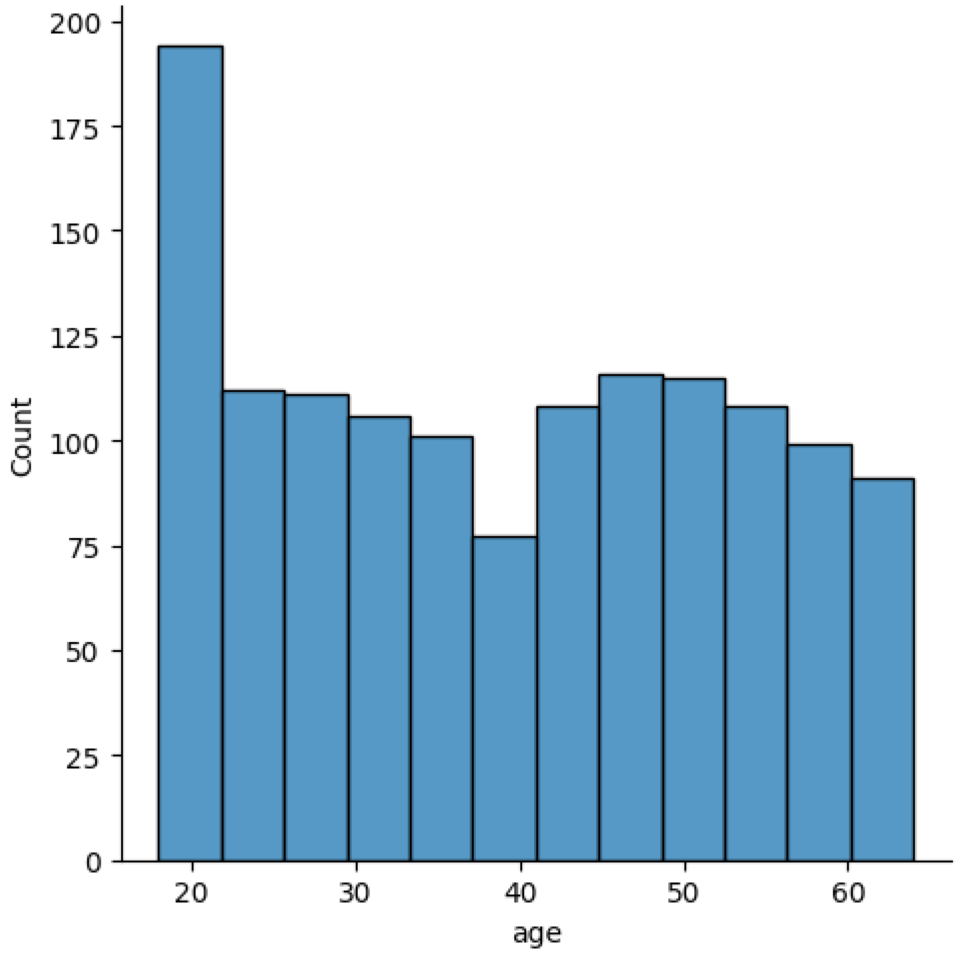


In [15]:

```
sns.displot(df['age'])
```

Out[15]:

<seaborn.axisgrid.FacetGrid at 0x1d515ad0b80>

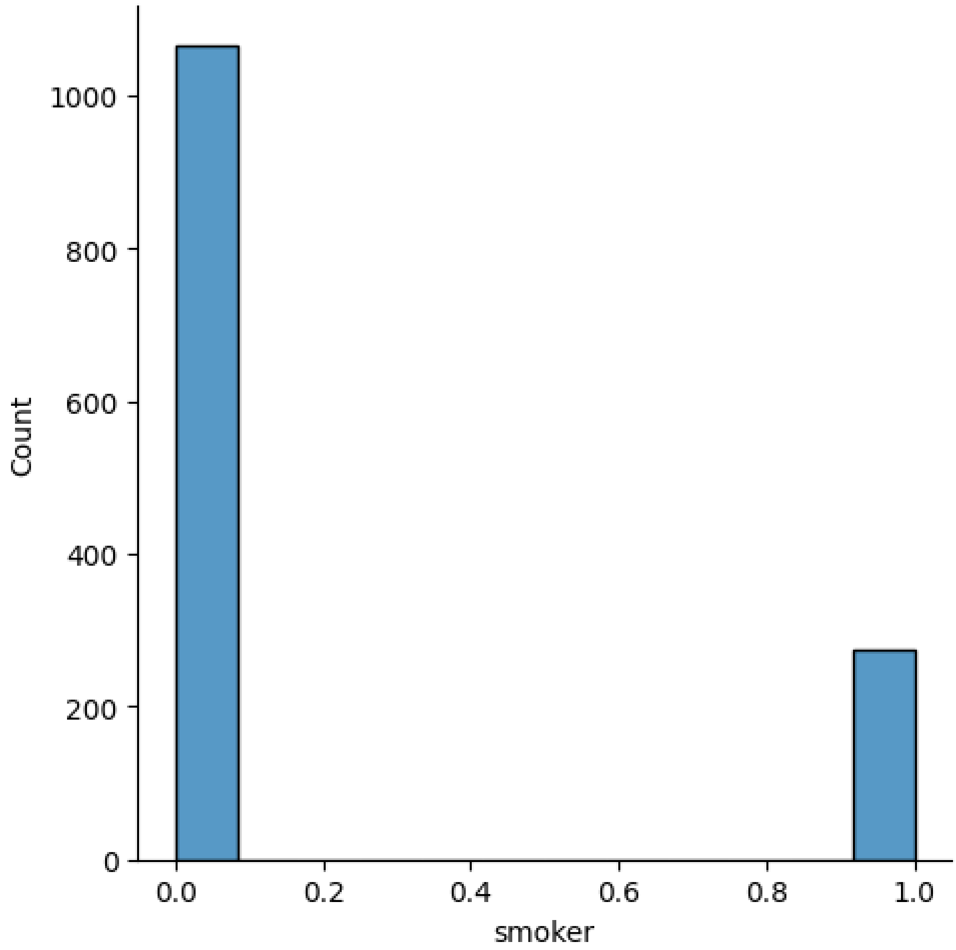


In [16]:

```
sns.displot(df['smoker'])
```

Out[16]:

```
<seaborn.axisgrid.FacetGrid at 0x1d57edaa860>
```

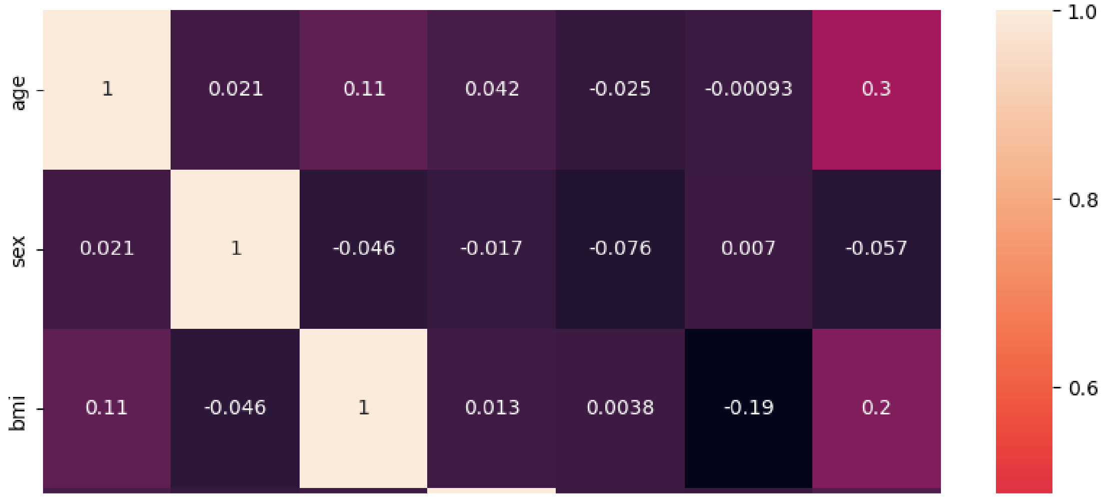


In [17]:

```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot=True)
```

Out[17]:

```
<Axes: >
```



In [18]:

```
df
```

Out[18]:

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

1338 rows × 7 columns

In [19]:

```
features=df.columns[0:4]
target=df.columns[-1]
x=df[features].values
y=df[target].values
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=17)
print("The dimension of X_train is {}".format(x_train.shape))
print("The dimension of X_test is {}".format(x_test.shape))
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
The dimension of X_train is (936, 4)
The dimension of X_test is (402, 4)
```

In [20]:

```
#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("\nLinear Regression 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))
```

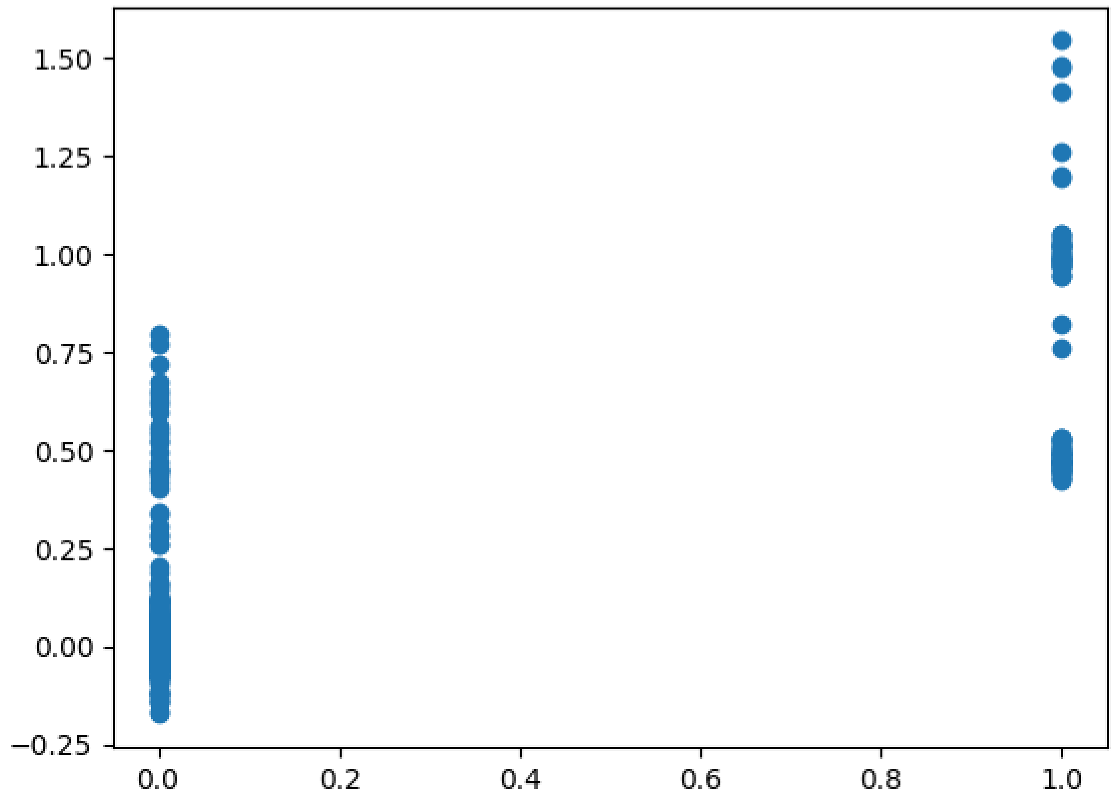
```

Linear Regression Model:

The train score for lr model is 0.1203468852993338
The test score for lr model is 0.09987297545748941
```

In [130]:

```
lm=LinearRegression()
lm.fit(x_train,y_train)
predictions=lm.predict(x_test)
plt.scatter(y_test,predictions)
```

Out[130]:

```
<matplotlib.collections.PathCollection at 0x1d5209899f0>
```



Ridge Regression

In [21]:

```
from sklearn.linear_model import Ridge,RidgeCV,Lasso
```

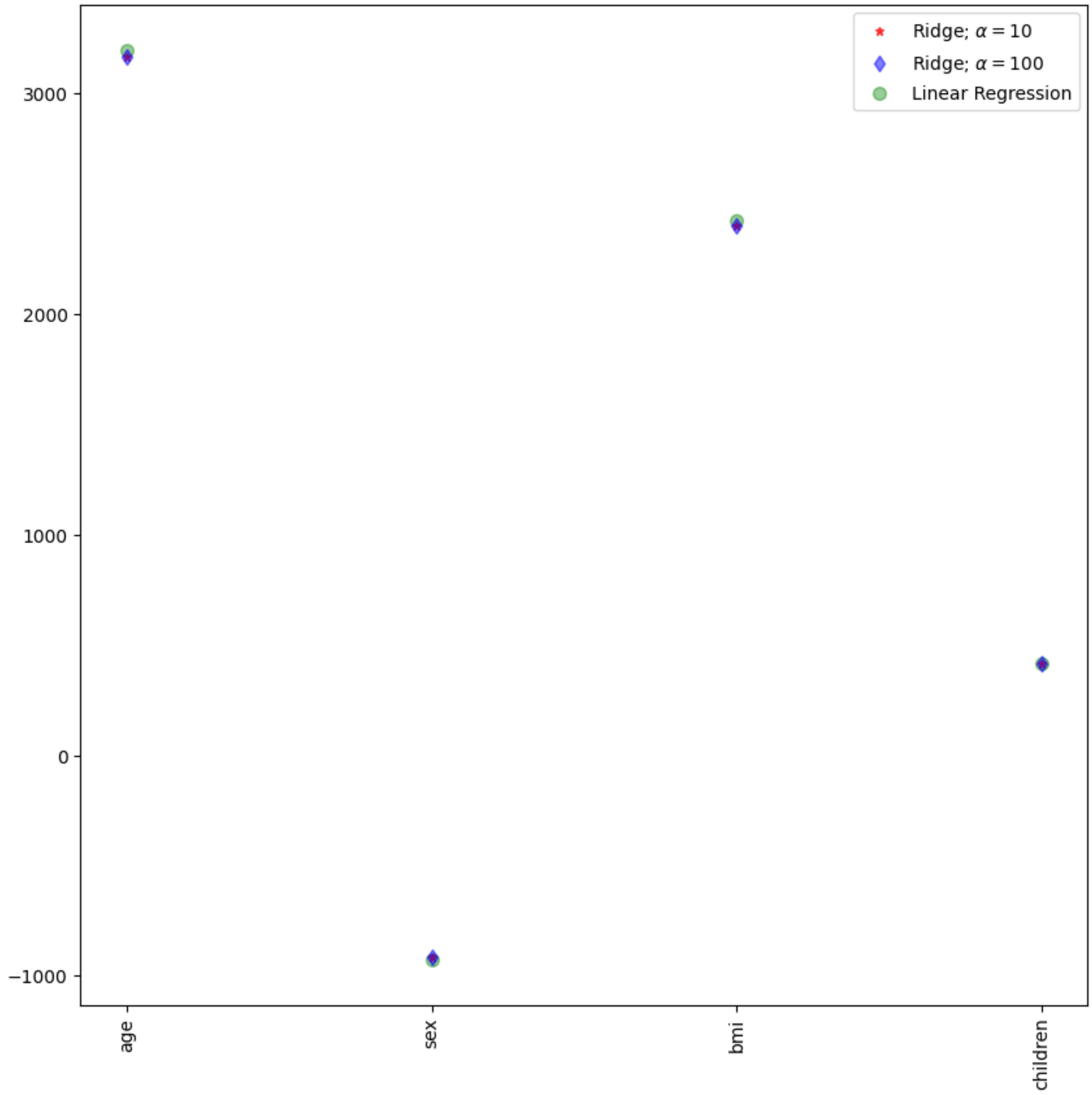
In [22]:

```
#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 0.12033561350484978
The test score for ridge model is 0.10024774091471034
```

In [23]:

```
plt.figure(figsize = (10, 10))
plt.plot(features,ridgeReg.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color
plt.plot(features,ridgeReg.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color
plt.plot(features,lr.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color='gree
plt.xticks(rotation = 90)
plt.legend()
plt.show()
```



Lasso Regression

In [24]:

```
#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))
```

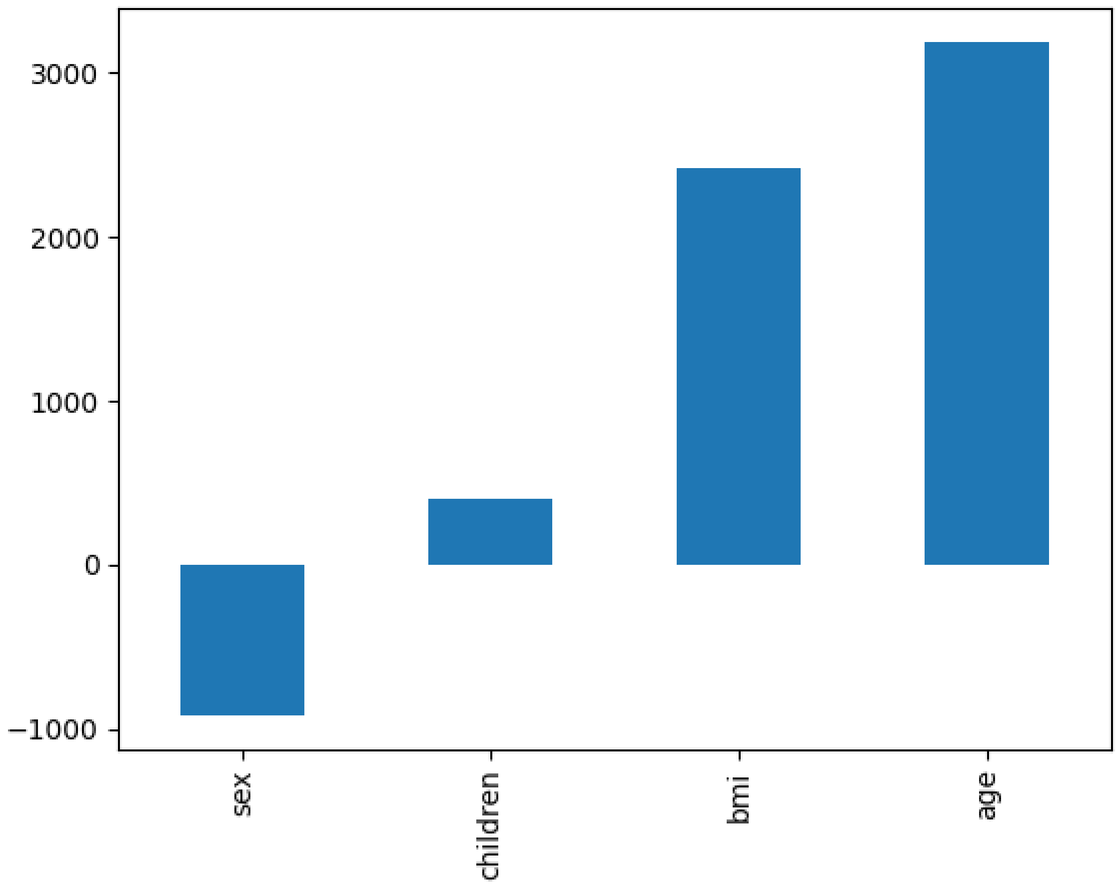
```
Lasso Model:

The train score for ls model is 0.12034453842851467
The test score for ls model is 0.10002810543470508
```

In [25]:

```
pd.Series(lasso.coef_, features).sort_values(ascending = True).plot(kind = "bar")
```

Out[25]:

<Axes: >



Elastic Net

In [27]:

```
from sklearn.linear_model import ElasticNet
regr = ElasticNet()
regr.fit(x,y)
print(regr.coef_)
print(regr.intercept_)
regr.score(x,y)
```

```
[ 240.50365352 -441.64432419  326.22952678  401.3182634 ]
-6383.214926157012
```

Out[27]:

```
0.1215573874611382
```

In [28]:

```
y_pred_elastic=regr.predict(x_train)
```

In [30]:

```
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 560157769.1020994
```

Logistic Regression

In [31]:

```
from sklearn.linear_model import LogisticRegression
```

In [32]:

```
df=pd.read_csv(r"C:\Users\DELL\Downloads\insurance.csv")
df
```

Out[32]:

	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 [33]:

```
convert={"sex":{"male":0,"female":1}}
df=df.replace(convert)
df
```

Out[33]:

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

1338 rows × 7 columns

In [34]:

```
convert={"smoker":{"yes":1,"no":0}}
df=df.replace(convert)
df
```

Out[34]:

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

1338 rows × 7 columns

In [35]:

```
convert={"region":{"southwest":0,"southeast":1,"northwest":3,"northeast":4}}
df=df.replace(convert)
df
```

Out[35]:

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

1338 rows × 7 columns

In [69]:

```
features=df[['age','sex','bmi','region']]
features.columns=['age','sex','bmi','region']
target=df[['smoker']]
target.columns=['smoker']
```

In [70]:

```
print('The Features Matrix Has %d Rows And %d Columns(s)'%(features.shape))
```

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

In [71]:

```
features_standardized=StandardScaler().fit_transform(features)
```

In [72]:

```
algorithm=LogisticRegression(max_iter=1000)
```

In [73]:

```
from sklearn.preprocessing import StandardScaler
```

In [74]:

```
Logistic_Regression_Model=algorithm.fit(features_standardized,target)
```

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

In [75]:

```
observation=[[28,1,28.880,2]]
```

In [76]:

```
predictions=Logistic_Regression_Model.predict(observation)
print('The model predicted the observation to belong to class %s'%(predictions))
```

```
The model predicted the observation to belong to class [0]
```

In [77]:

```
print('The algorithm was trained to predict one of the two classes:%s'%(algorithm.classes
```

```
The algorithm was trained to predict one of the two classes:[0 1]
```

In [134]:

```
print("""The model says the probability of the observation we passed belonging to class['
print()
print("""The model says the probability of the observation we passed belonging to class['
print()
```

```
The model says the probability of the observation we passed belonging to c
lass['0']is 0.9580033379487249

The model says the probability of the observation we passed belonging to c
lass['1']is 0.041996662051275016
```

In [135]:

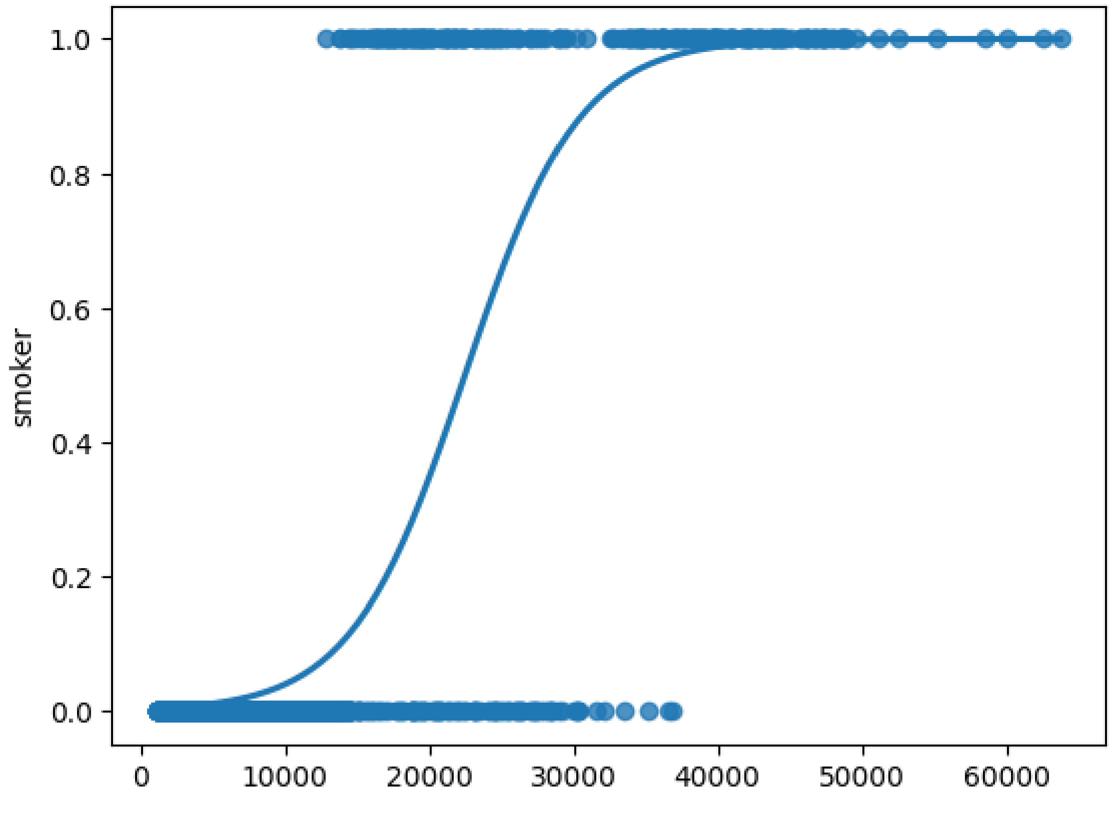
```
x1=np.array(df["charges"]).reshape(-1,1)
```

In [82]:

```
sns.regplot(x=x1,y=target,data=df,logistic=True,ci=None)
```

Out[82]:

```
<Axes: ylabel='smoker'>
```



Decision Tree

In [83]:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

In [84]:

```
x=["age","sex","bmi","children","region"]
y=["Yes","No"]
all_inputs=df[x]
all_classes=df["smoker"]
```

In [86]:

```
(x_train,x_test,y_train,y_test)=train_test_split(all_inputs,all_classes,test_size=0.30)
```

In [87]:

```
clf=DecisionTreeClassifier(random_state=0)
```

In [88]:

```
clf.fit(x_train,y_train)
```

Out[88]:

```
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 [89]:

```
score=clf.score(x_test,y_test)
print(score)
```

```
0.6865671641791045
```

Random Forest

In [91]:

```
import matplotlib.pyplot as plt,seaborn as sns
```

In [92]:

```
x=df.drop('smoker',axis=1)
y=df['smoker']
```

In [93]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.7)
x_train.shape,x_test.shape
```

Out[93]:

```
((936, 6), (402, 6))
```

In [94]:

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[94]:

```
RandomForestClassifier()
```

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 [95]:

```
rf=RandomForestClassifier()
```

In [97]:

```
params={'max_depth':[2,3,5,10,20],
 'min_samples_leaf':[5,10,20,50,100,200],
 'n_estimators':[10,25,30,50,100,200]}
```

In [99]:

```
from sklearn.model_selection import GridSearchCV
grid_search=GridSearchCV(estimator=rf,param_grid=params,cv=2,scoring='accuracy')
grid_search.fit(x_train,y_train)
```

Out[99]:

```
GridSearchCV(cv=2, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [2, 3, 5, 10, 20],
                         'min_samples_leaf': [5, 10, 20, 50, 100, 200],
                         'n_estimators': [10, 25, 30, 50, 100, 200]},
             scoring='accuracy')
```

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 [100]:

```
grid_search.best_score_
```

Out[100]:

```
0.9476495726495726
```

In [101]:

```
rf_best=grid_search.best_estimator_
print(rf_best)
```

```
RandomForestClassifier(max_depth=20, min_samples_leaf=5, n_estimators=200)
```

In [102]:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[5],feature_names=x.columns,class_names=["1","0"],filled=Tru
```

Out[102]:

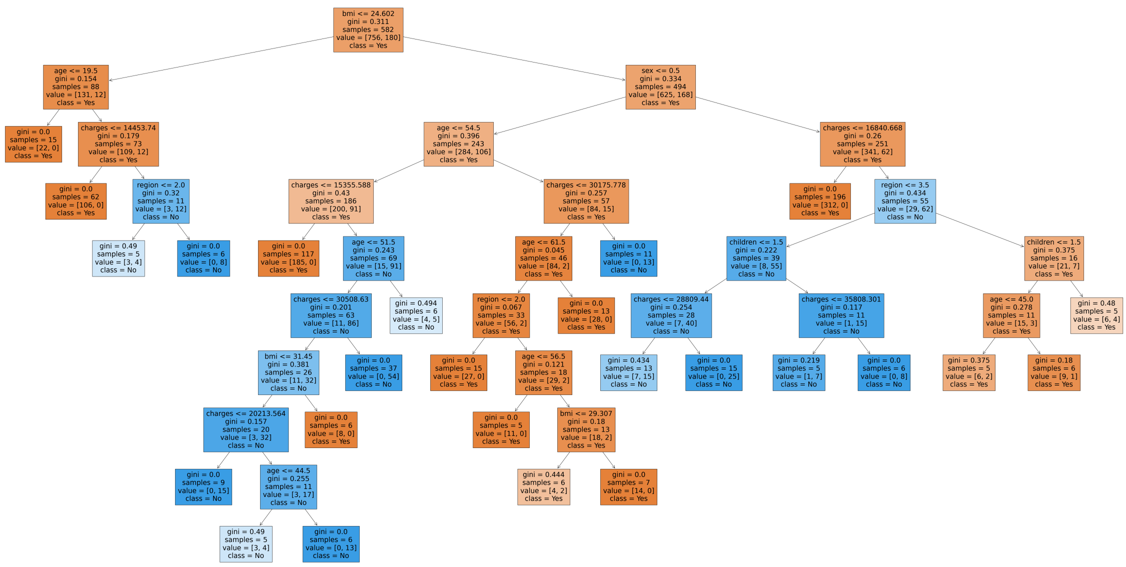
```
[Text(0.65, 0.95, 'bmi <= 34.885\ngini = 0.303\nsamples = 589\nvalue =
[762, 174]\nclass = 1'),
 Text(0.4888888888888889, 0.85, 'age <= 60.5\ngini = 0.281\nsamples = 43
7\nvalue = [570, 116]\nclass = 1'),
 Text(0.3333333333333333, 0.75, 'region <= 3.5\ngini = 0.266\nsamples =
418\nvalue = [553, 104]\nclass = 1'),
 Text(0.17777777777777778, 0.65, 'children <= 2.5\ngini = 0.235\nsamples
= 307\nvalue = [412, 65]\nclass = 1'),
 Text(0.08888888888888889, 0.55, 'region <= 0.5\ngini = 0.216\nsamples =
258\nvalue = [348, 49]\nclass = 1'),
 Text(0.044444444444444446, 0.45, 'charges <= 12995.738\ngini = 0.264\ns
amples = 85\nvalue = [108, 20]\nclass = 1'),
 Text(0.022222222222222223, 0.35, 'gini = 0.0\nsamples = 67\nvalue = [10
0, 0]\nclass = 1'),
 Text(0.06666666666666667, 0.35, 'bmi <= 30.35\ngini = 0.408\nsamples =
18\nvalue = [8, 20]\nclass = 0'),
 Text(0.044444444444444446, 0.25, 'bmi <= 26.9\ngini = 0.488\nsamples =
```

In [103]:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[7],feature_names=x.columns,class_names=["Yes","No"],filled=
```

Out[103]:

```
[Text(0.3287037037037037, 0.95, 'bmi <= 24.602\ngini = 0.311\nsamples = 58
2\nvalue = [756, 180]\nclass = Yes'),
 Text(0.07407407407407407, 0.85, 'age <= 19.5\ngini = 0.154\nsamples = 88
\nvalue = [131, 12]\nclass = Yes'),
 Text(0.037037037037037035, 0.75, 'gini = 0.0\nsamples = 15\nvalue = [22,
0]\nclass = Yes'),
 Text(0.1111111111111111, 0.75, 'charges <= 14453.74\ngini = 0.179\nsample
s = 73\nvalue = [109, 12]\nclass = Yes'),
 Text(0.07407407407407407, 0.65, 'gini = 0.0\nsamples = 62\nvalue = [106,
0]\nclass = Yes'),
 Text(0.14814814814814814, 0.65, 'region <= 2.0\ngini = 0.32\nsamples = 11
\nvalue = [3, 12]\nclass = No'),
 Text(0.1111111111111111, 0.55, 'gini = 0.49\nsamples = 5\nvalue = [3, 4]
\nclass = No'),
 Text(0.18518518518518517, 0.55, 'gini = 0.0\nsamples = 6\nvalue = [0, 8]
\nclass = No'),
 Text(0.5833333333333334, 0.85, 'sex <= 0.5\ngini = 0.334\nsamples = 494\n
value = [625, 168]\nclass = Yes'),
 Text(0.4074074074074074, 0.75, 'age <= 54.5\ngini = 0.396\nsamples = 243
\nvalue = [284, 106]\nclass = Yes'),
 Text(0.2962962962962963, 0.65, 'charges <= 15355.588\ngini = 0.43\nsample
s = 186\nvalue = [200, 91]\nclass = Yes'),
 Text(0.25925925925925924, 0.55, 'gini = 0.0\nsamples = 117\nvalue = [185,
0]\nclass = Yes'),
 Text(0.3333333333333333, 0.55, 'age <= 51.5\ngini = 0.243\nsamples = 69\n
value = [15, 91]\nclass = No'),
 Text(0.2962962962962963, 0.45, 'charges <= 30508.63\ngini = 0.201\nsample
s = 63\nvalue = [11, 86]\nclass = No'),
 Text(0.25925925925925924, 0.35, 'bmi <= 31.45\ngini = 0.381\nsamples = 26
\nvalue = [11, 32]\nclass = No'),
 Text(0.2222222222222222, 0.25, 'charges <= 20213.564\ngini = 0.157\nsampl
es = 20\nvalue = [3, 32]\nclass = No'),
 Text(0.18518518518518517, 0.15, 'gini = 0.0\nsamples = 9\nvalue = [0, 15]
\nclass = No'),
 Text(0.25925925925925924, 0.15, 'age <= 44.5\ngini = 0.255\nsamples = 11
\nvalue = [3, 17]\nclass = No'),
 Text(0.2222222222222222, 0.05, 'gini = 0.49\nsamples = 5\nvalue = [3, 4]
\nclass = No'),
 Text(0.2962962962962963, 0.05, 'gini = 0.0\nsamples = 6\nvalue = [0, 13]
\nclass = No'),
```




```
\nclass = Yes'),
 Text(0.5185185185185185, 0.25, 'bmi <= 29.307\ngini = 0.18\nsamples = 13
\nvalue = [18, 2]\nclass = Yes'),
 Text(0.4814814814814815, 0.15, 'gini = 0.444\nsamples = 6\nvalue = [4,
```

```
2]\nclass = Yes'),
 Text(0.5555555555555556, 0.15, 'gini = 0.0\nsamples = 7\nvalue = [14, 0]
\nclass = Yes'),
 Text(0.5185185185185185, 0.45, 'gini = 0.0\nsamples = 13\nvalue = [28, 0]
\nclass = Yes'),
 Text(0.5555555555555556, 0.55, 'gini = 0.0\nsamples = 11\nvalue = [0, 13]
\nclass = No'),
 Text(0.7592592592592593, 0.75, 'charges <= 16840.668\ngini = 0.26\nsample
s = 251\nvalue = [341, 62]\nclass = Yes'),
 Text(0.7222222222222222, 0.65, 'gini = 0.0\nsamples = 196\nvalue = [312,
0]\nclass = Yes'),
 Text(0.7962962962962963, 0.65, 'region <= 3.5\ngini = 0.434\nsamples = 55
\nvalue = [29, 62]\nclass = No'),
 Text(0.6666666666666666, 0.55, 'children <= 1.5\ngini = 0.222\nsamples =
39\nvalue = [8, 55]\nclass = No'),
 Text(0.5925925925925926, 0.45, 'charges <= 28809.44\ngini = 0.254\nsample
s = 28\nvalue = [7, 40]\nclass = No'),
 Text(0.5555555555555556, 0.35, 'gini = 0.434\nsamples = 13\nvalue = [7, 1
5]\nclass = No'),
 Text(0.6296296296296297, 0.35, 'gini = 0.0\nsamples = 15\nvalue = [0, 25]
\nclass = No'),
 Text(0.7407407407407407, 0.45, 'charges <= 35808.301\ngini = 0.117\nsampl
es = 11\nvalue = [1, 15]\nclass = No'),
 Text(0.7037037037037037, 0.35, 'gini = 0.219\nsamples = 5\nvalue = [1, 7]
\nclass = No'),
 Text(0.7777777777777778, 0.35, 'gini = 0.0\nsamples = 6\nvalue = [0, 8]\n
class = No'),
 Text(0.9259259259259259, 0.55, 'children <= 1.5\ngini = 0.375\nsamples =
16\nvalue = [21, 7]\nclass = Yes'),
 Text(0.8888888888888888, 0.45, 'age <= 45.0\ngini = 0.278\nsamples = 11\n
value = [15, 3]\nclass = Yes'),
 Text(0.8518518518518519, 0.35, 'gini = 0.375\nsamples = 5\nvalue = [6, 2]
\nclass = Yes'),
 Text(0.9259259259259259, 0.35, 'gini = 0.18\nsamples = 6\nvalue = [9, 1]
\nclass = Yes'),
 Text(0.9629629629629629, 0.45, 'gini = 0.48\nsamples = 5\nvalue = [6, 4]
\nclass = Yes')]
```

In [104]:

```
rf_best.feature_importances_
```

Out[104]:

```
array([0.0333835 , 0.01005727, 0.06982971, 0.01202703, 0.01196482,
       0.86273766])
```

In [105]:

```
imp_df=pd.DataFrame({"Varname":x_train.columns,"Imp":rf_best.feature_importances_})
imp_df.sort_values(by="Imp",ascending=False)
```

Out[105]:

	Varname	Imp
5	charges	0.862738
2	bmi	0.069830
0	age	0.033384
3	children	0.012027
4	region	0.011965
1	sex	0.010057

In [110]:

```
print(rfc.score(x_train,y_train))
```

1.0

In [111]:

```
print(rfc.score(x_test,y_test))
```

0.9552238805970149

Model Saving for all the models

For Linear Regression

In [119]:

```
import pickle
fname="y_pred"
pickle.dump=lr,open(fname,'wb')
```

In [120]:

```
import pickle
fname="y_pred"
pickle.dump=algorithm,open(fname,'wb')
```

Conclusion:

```
By observing all the models the logistic regression has highest accuracy.Therefo
re we can say that logistic regression is the best fit model for the insurance d
ata set
```