Problem Statement:

```
    for Insurance dataset implementing all models based on the accuracy,which
means which model is highest accuracy.That model is bestmodel for this dataset.
to finding bestmodel
```

In [2]:

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

Data collection:

In [3]:

```
df=pd.read_csv(r"C:\Users\raja\Downloads\insurance (1).csv")
df
```

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
...	...	...	...	...	...	...	...
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

Data cleaning and Preprocessing

In [4]:

```
df.head()
```

Out[4]:

	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 [5]:

```
df.tail()
```

Out[5]:

	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 [6]:

```
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 [7]:

```
df.describe()
```

Out[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

In [8]:

```
df.size
```

Out[8]:

```
9366
```

In [9]:

```
df.shape
```

Out[9]:

```
(1338, 7)
```

In [10]:

```
df.isna().any()
```

Out[10]:

```
age         False
sex         False
bmi         False
children    False
smoker      False
region      False
charges     False
dtype: bool
```

In [11]:

```
df['sex'].value_counts()
```

Out[11]:

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

In [12]:

```
df['smoker'].value_counts()
```

Out[12]:

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

In [13]:

```
convert={"region":{"southeast":1,"southwest":2,"northwest":3,"northeast":4}}
df=df.replace(convert)
df
```

Out[13]:

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

1338 rows × 7 columns

In [14]:

```
df['region'].value_counts()
```

Out[14]:

```
region
1    364
2    325
3    325
4    324
Name: count, dtype: int64
```

In [15]:

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

Out[15]:

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

1338 rows × 7 columns

In [16]:

```
convert={"smoker":{"no":1,"yes":2}}
df=df.replace(convert)
df
```

Out[16]:

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

1338 rows × 7 columns

In [17]:

```
features=df.columns[0:5]
target=df.columns[-1]
```

Exploratory Data Analysis

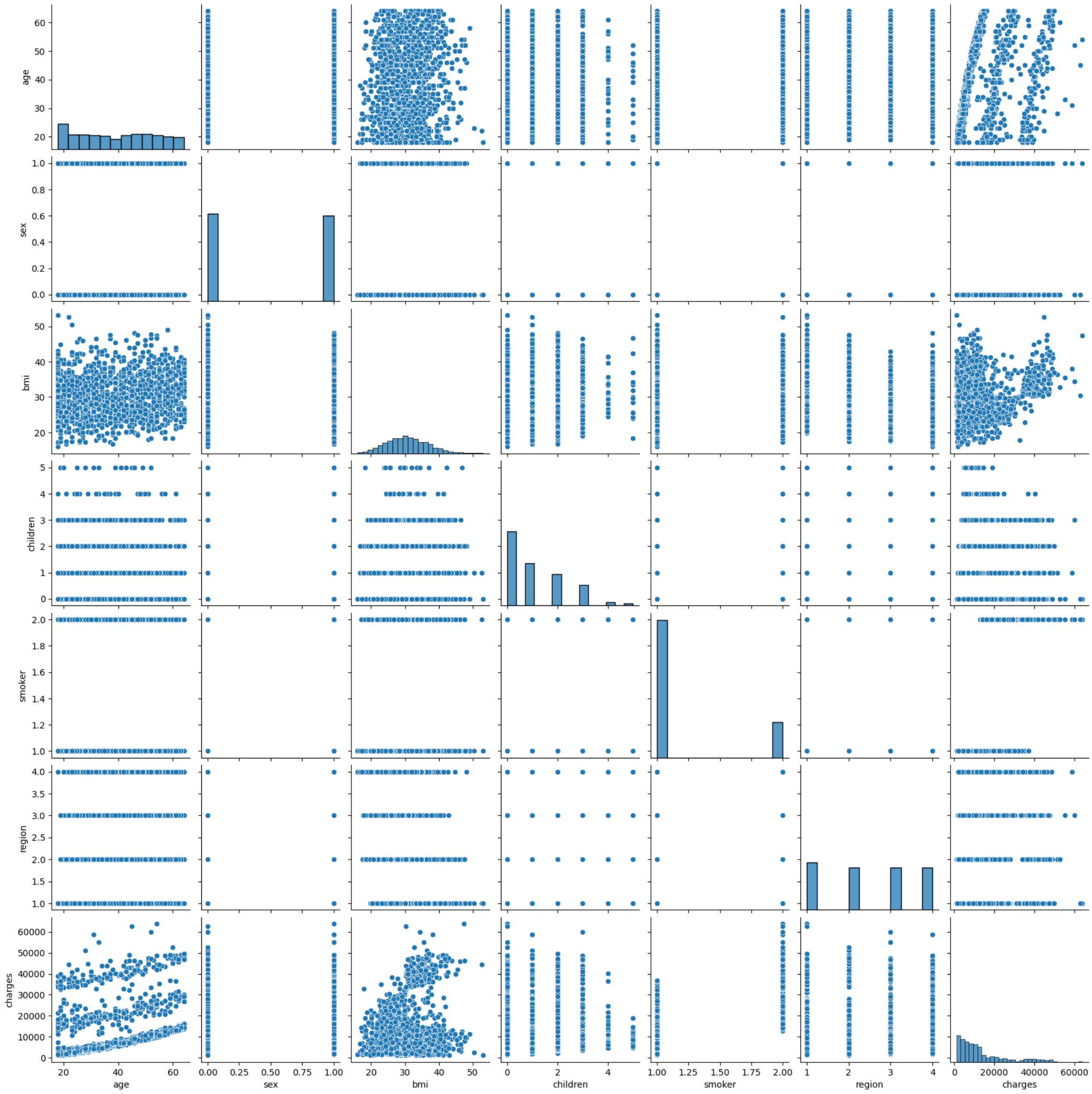
Data visualization

In [18]:

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

Out[18]:

```
<seaborn.axisgrid.PairGrid at 0x1af0e8a4160>
```

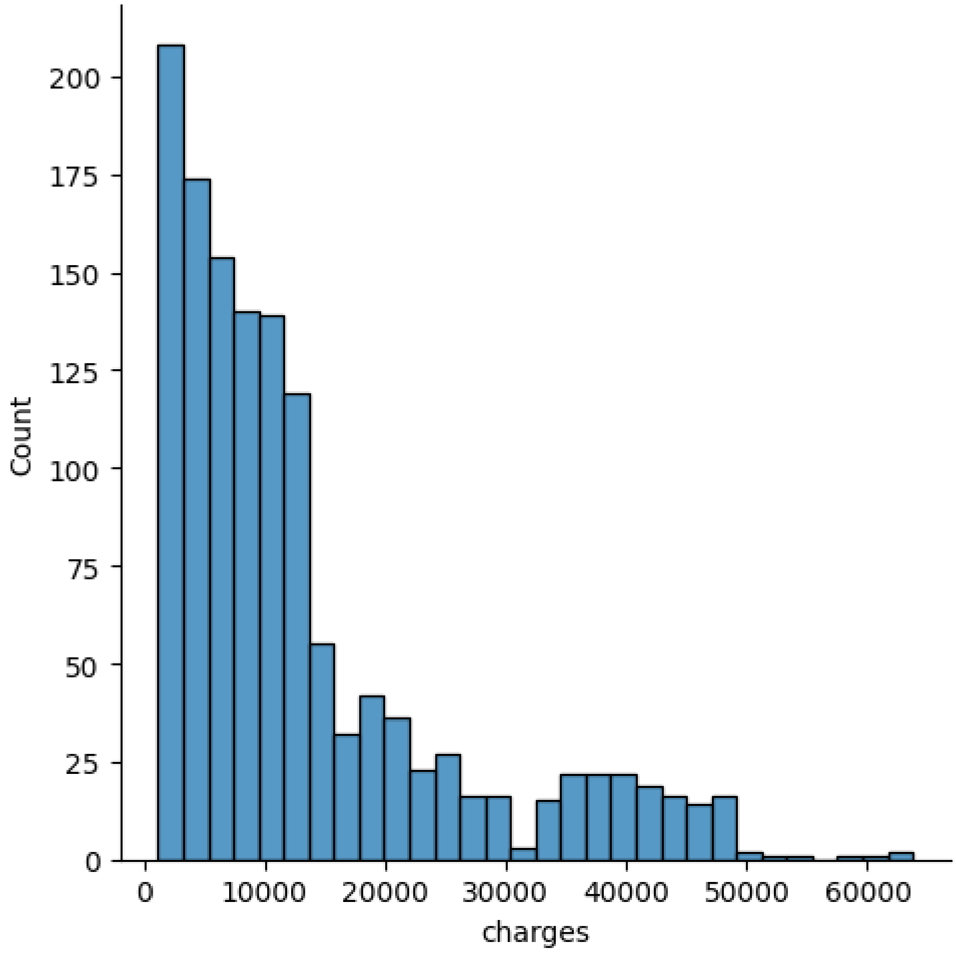


In [19]:

```
sns.displot(df["charges"])
```

Out[19]:

```
<seaborn.axisgrid.FacetGrid at 0x1af445eec50>
```



In [20]:

```
features=df.columns[0:6]
features
```

Out[20]:

```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region'], dtype='objec
t')
```

In [21]:

```
target=df.columns[-1]
target
```

Out[21]:

```
'charges'
```

In [22]:

```
#training our model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
x=df[features].values
y=df[target].values
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.50,random_state=17)
x_train.shape
x_test.shape
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.fit_transform(x_test)
```

Data Modelling

Linear Regression

In [23]:

```
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
```

In [24]:

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

Out[24]:

```
LinearRegression()
```

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.

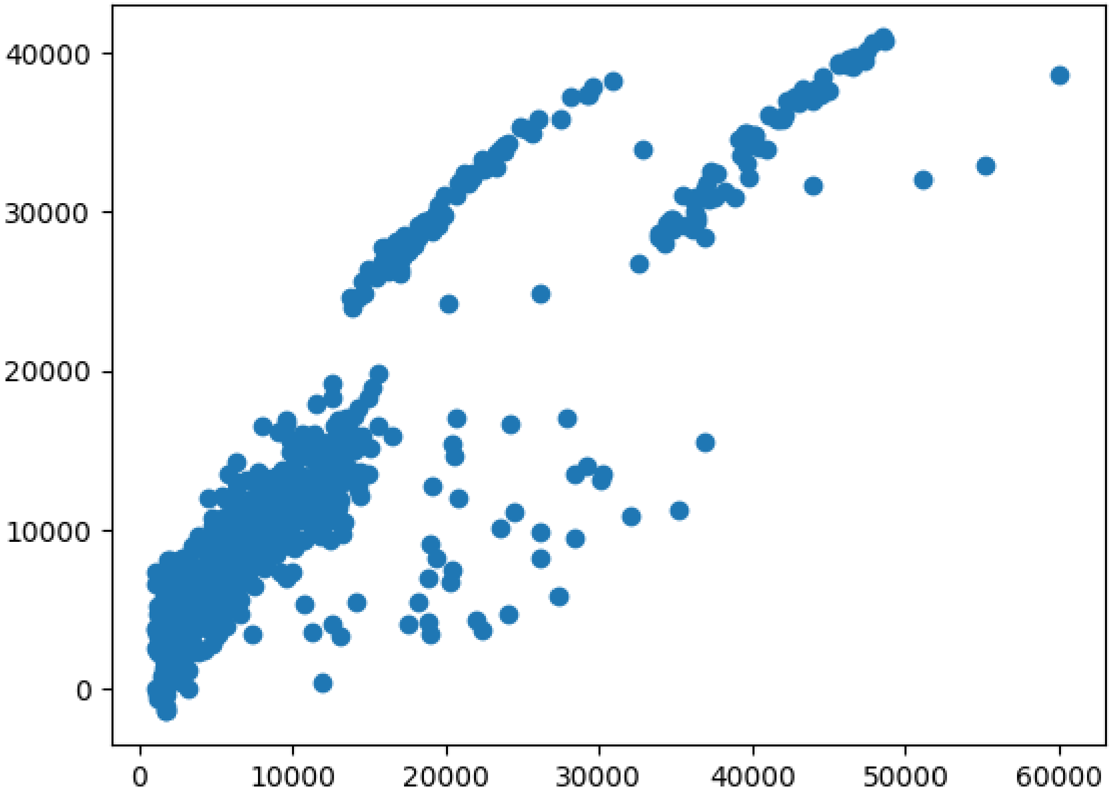
In [25]:

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

```
0.7553050021744551
```

In [26]:

```
y_pred=reg.predict(x_test)
plt.scatter(y_test,y_pred)
plt.show()
```



In [27]:

```
#evaluation of model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

In [28]:

```
model=LinearRegression()
model.fit(x_train,y_train)
```

Out[28]:

```
LinearRegression()
```

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 [29]:

```
y_pred=model.predict(x_test)
r2=r2_score(y_test,y_pred)
print("R2 score:",r2)
```

```
R2 score: 0.7553050021744551
```

RIDGE

In [30]:

```
from sklearn.linear_model import Ridge,Lasso,ElasticNet
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
```

In [31]:

```
#ridge regression model
ridgeReg=Ridge(alpha=10)
ridgeReg.fit(x_train,y_train)
train_score_ridge=ridgeReg.score(x_train,y_train)
test_score_ridge=ridgeReg.score(x_test,y_test)
print("\nRidge Model\n")
print("Train Score for ridge model is",(train_score_ridge))
print("Test Score for ridge model is",(test_score_ridge))
```

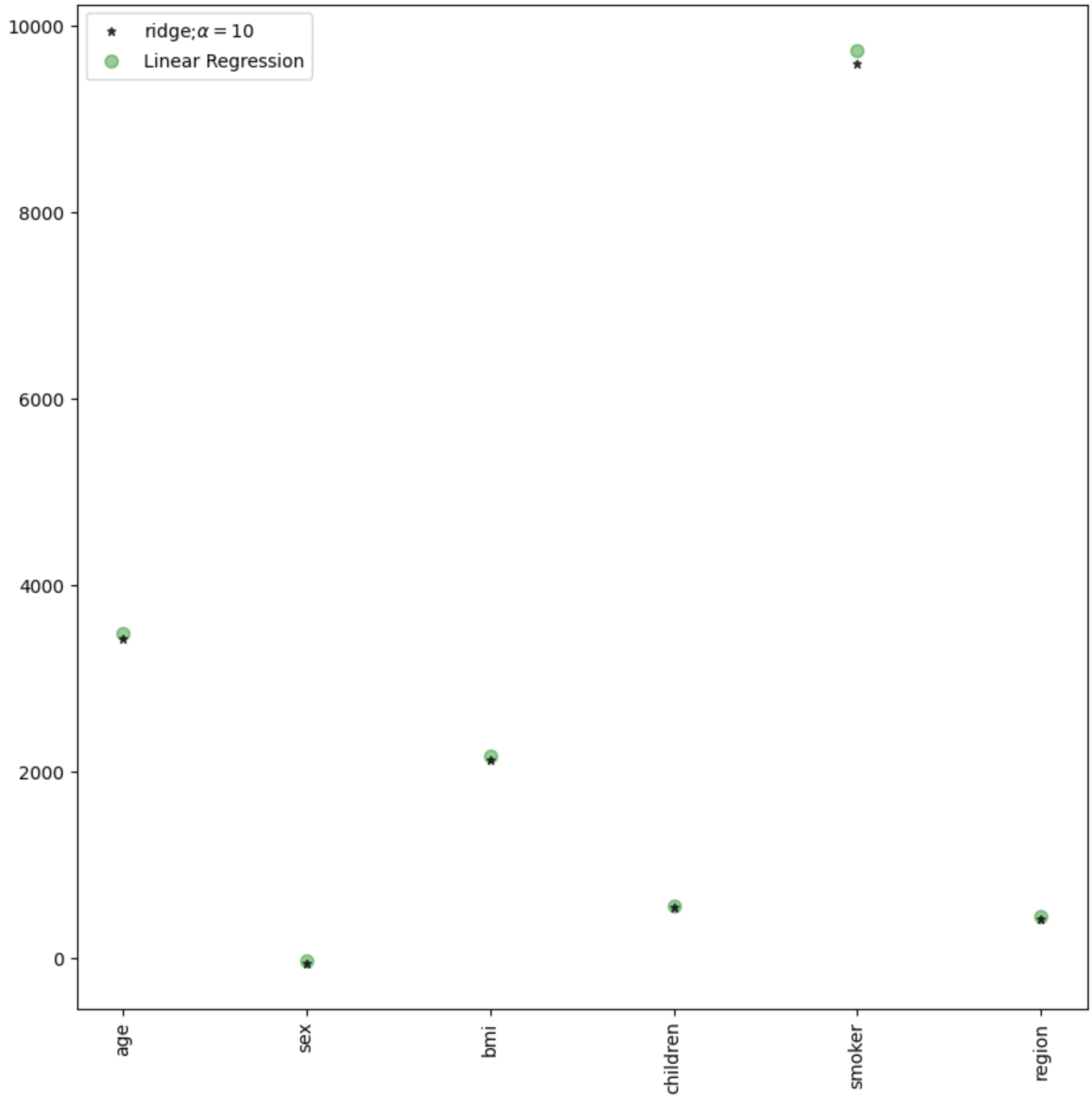
```

Ridge Model

Train Score for ridge model is 0.7408189083612313
Test Score for ridge model is 0.7555036308713557
```

In [32]:

```
plt.figure(figsize=(10,10))
plt.plot(features,ridgeReg.coef_,alpha=0.7,linestyle='none',marker="*",markersize=5,colo
plt.plot(features,reg.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color="gr
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



LASSO Regression

In [33]:

```
#Elastic Net
re=ElasticNet()
re.fit(x,y)
print(re.coef_)
#print(re.intercept_)
```

```
[ 244.53200843 -323.29321608  327.88975459  389.15389489 5841.24528723
   75.2866119 ]
```

In [34]:

```
#lasso regression
lassoReg=Lasso(alpha=10)
lassoReg.fit(x_train,y_train)
train_score_lasso=lassoReg.score(x_train,y_train)
test_score_lasso=lassoReg.score(x_test,y_test)
print("\nlasso Model\n")
print("Train Score for lasso model is",(train_score_lasso))
print("Test Score for lasso model is",(test_score_lasso))
```

```
lasso Model

Train Score for lasso model is 0.7409768055935853
Test Score for lasso model is 0.7553619179215416
```

ElasticNet

In [35]:

```
re.score(x,y)
```

Out[35]:

```
0.3919002752780354
```

In [36]:

```
y_pred_elastic=re.predict(x_train)
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 830429140.4997891
```

Logistic Regression

In [37]:

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

In [38]:

```
pd.set_option("display.max_rows",10000000000)
pd.set_option('display.max_columns',10000000000)
pd.set_option('display.width',95)
```

In [39]:

```
print("this dataframe has %d rows and %d columns"%(df.shape))
```

```
this dataframe has 1338 rows and 7 columns
```

In [40]:

```
df.head()
```

Out[40]:

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

In [41]:

```
features_matrix=df[["age","sex","bmi","region"]]
features_matrix.columns=["age","sex","bmi","region"]
target_matrix=df.iloc[:,-3]
```

In [42]:

```

print('The features matrix has %d Rows and %d columns'%(features_matrix.shape))
print('The features matrix has %d Rows and %d columns'%(np.array(target_matrix).reshape(
```

```
The features matrix has 1338 Rows and 4 columns
The features matrix has 1338 Rows and 1 columns
```

In [43]:

```
features_matrix_Standardized=StandardScaler().fit_transform(features_matrix)
```

In [44]:

```
algorithm=LogisticRegression(max_iter=100)
```

In [45]:

```
Logistic_Regression_model=algorithm.fit(features_matrix_Standardized,target_matrix)
```

In [46]:

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

In [47]:

```
predictions=Logistic_Regression_model.predict(observation)
print('The model predicted the observtaion to belong to class %s'%(predictions))
print('The algorithm was trained to predict one of the two calsses : %s'%(algorithm.clas
```

```
The model predicted the observtaion to belong to class [1]
The algorithm was trained to predict one of the two calsses : [1 2]
```

In [48]:

```
print("""The model says the prbability of the observation we passed belonging to calss [
print()
print("""The model says the prbability of the observation we passed belonging to calss [
```

```
The model says the prbability of the observation we passed belonging to ca
lss ['0'] Is 0.9800486899442347

The model says the prbability of the observation we passed belonging to ca
lss ['1'] Is 0.019951310055765316
```

In [49]:

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

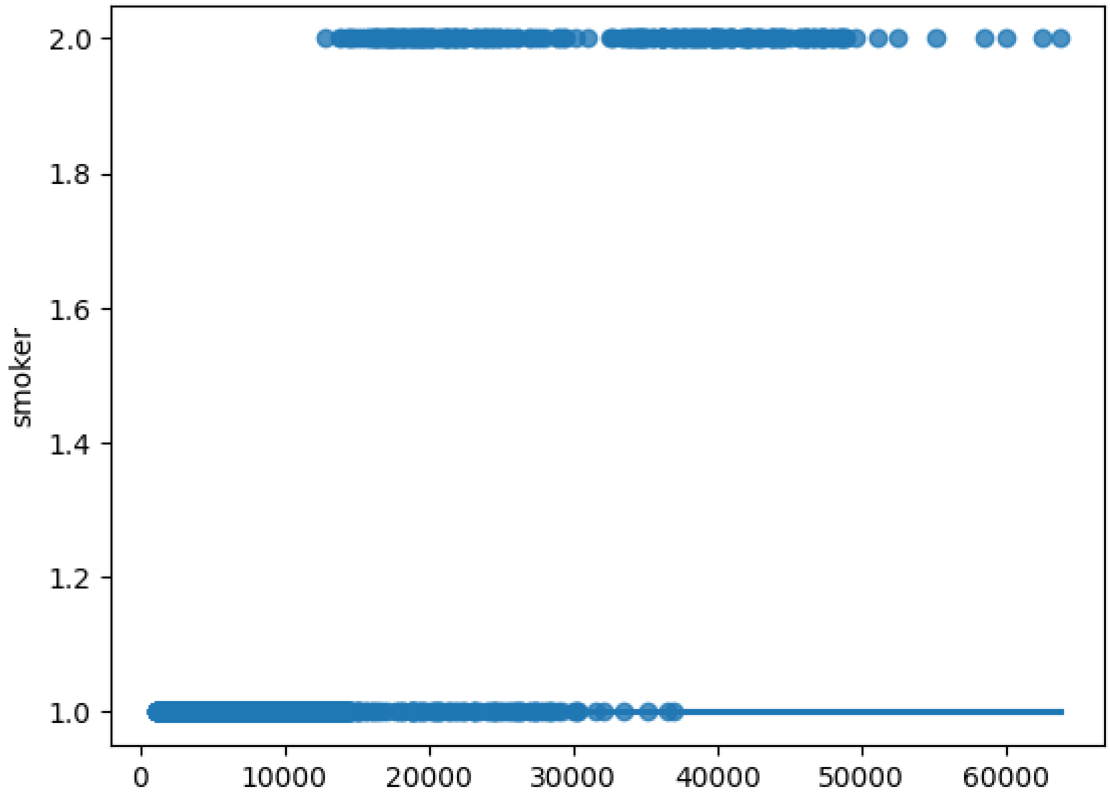
In [50]:

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

```
C:\Users\raja\AppData\Local\Programs\Python\Python310\lib\site-packages\st
atsmodels\genmod\families\links.py:198: RuntimeWarning: overflow encounter
ed in exp
  t = np.exp(-z)
```

Out[50]:

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



Decision trees

In [51]:

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

In [52]:

```
df
```

Out[52]:

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

In [53]:

```
df["sex"].value_counts()
```

Out[53]:

```
sex
0    676
1    662
Name: count, dtype: int64
```

In [54]:

```
df["smoker"].value_counts()
```

Out[54]:

```
smoker
1    1064
2     274
Name: count, dtype: int64
```

In [55]:

```
x=["age","sex","children","bmi"]
y=["0","1"]
all_inputs=df[x]
all_classes=df["smoker"]
```

In [56]:

```
x_train,x_test,y_train,y_test=train_test_split(all_inputs,all_classes,test_size=0.25)
x_train.shape,x_test.shape
```

Out[56]:

```
((1003, 4), (335, 4))
```

In [57]:

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

In [58]:

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

Out[58]:

```
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 [59]:

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

In [60]:

```
print(score)
```

```
0.6895522388059702
```

In [61]:

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

Out[61]:

```
0.9990029910269193
```

Random Forest

In [62]:

```
df1=df.drop("charges",axis=1)
df1
```

Out[62]:

	age	sex	bmi	children	smoker	region
0	19	1	27.900	0	2	2
1	18	0	33.770	1	1	1
2	28	0	33.000	3	1	1
3	33	0	22.705	0	1	3
4	32	0	28.880	0	1	3
5	31	1	25.740	0	1	1
6	46	1	33.440	1	1	1
7	37	1	27.740	3	1	3
8	37	0	29.830	2	1	4
9	60	1	25.840	0	1	3
10	25	0	26.220	0	1	4

In [63]:

```
df2=df1.drop("children",axis=1)
df2
```

Out[63]:

	age	sex	bmi	smoker	region
0	19	1	27.900	2	2
1	18	0	33.770	1	1
2	28	0	33.000	1	1
3	33	0	22.705	1	3
4	32	0	28.880	1	3
5	31	1	25.740	1	1
6	46	1	33.440	1	1
7	37	1	27.740	1	3
8	37	0	29.830	1	4
9	60	1	25.840	1	3
10	25	0	26.220	1	4

In [64]:

```
x=df2.drop("smoker",axis=1)
y=df2["smoker"]
```

In [65]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
x_train.shape,x_test.shape
```

Out[65]:

```
((1003, 4), (335, 4))
```

In [66]:

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

Out[66]:

```
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 [67]:

```
rf=RandomForestClassifier()
```

In [68]:

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

In [69]:

```
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[69]:

```
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 [70]:

```
grid_search.best_score_
```

Out[70]:

```
0.7996039792924112
```

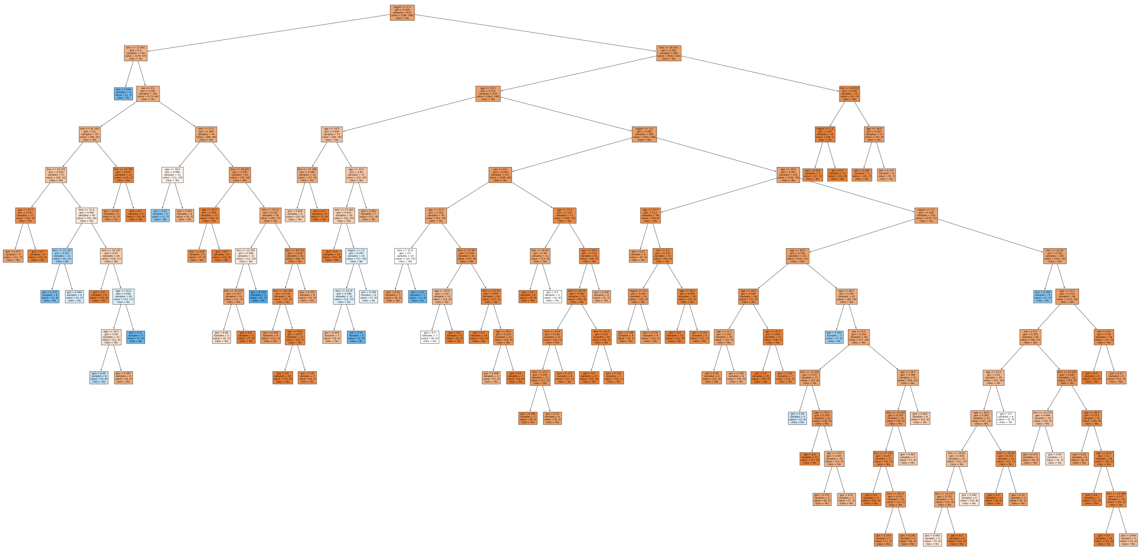
In [71]:

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

```
RandomForestClassifier(max_depth=20, min_samples_leaf=5, n_estimators=25)
```

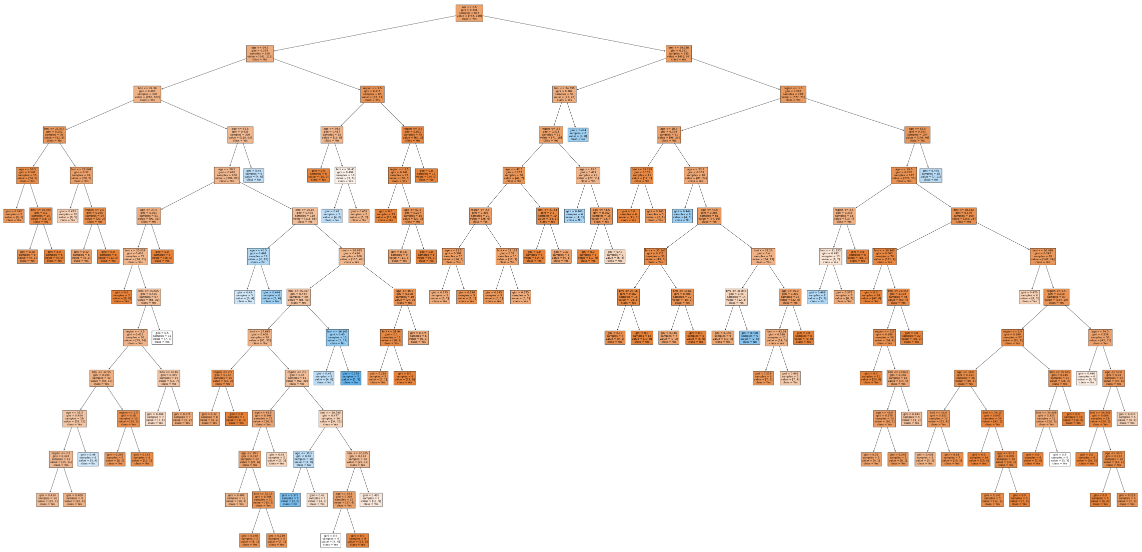
In [72]:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[5],feature_names=x.columns,class_names=['Yes','No'],filled
```



In [73]:

```
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
```



In [74]:

```
rf_best.feature_importances_
```

Out[74]:

```
array([0.39397304, 0.0469756 , 0.46710501, 0.09194635])
```

In [75]:

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

Out[75]:

	Varname	Imp
2	bmi	0.467105
0	age	0.393973
3	region	0.091946
1	sex	0.046976

In [83]:

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

```
0.9990029910269193
```

In [84]:

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

```
0.7044776119402985
```

Conclusion:

After implementing all models in the dataset RandomForest got highest accuracy. so in this dataset RandomForest is the bestmodel