In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:

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

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

In [4]:

```
data.head
```

Out[4]:

```
<bound method NDFrame.head of       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 x 7 columns]>
```

In [5]:

```
data.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 [6]:

```
data.describe()
```

Out[6]:

	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 [7]:

```
data['age'].unique()
data['children'].unique()
data['bmi'].unique()
data['sex'].unique()
data['smoker'].unique()
data['charges'].unique()
```

Out[7]:

```
array([16884.924 ,  1725.5523,  4449.462 , ...,  1629.8335,  2007.945 ,
       29141.3603])
```

In [8]:

```
data.isnull().sum()
```

Out[8]:

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

In [9]:

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

Out[9]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	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	0	31.920	0	no	northeast	2205.98080
1335	18	0	36.850	0	no	southeast	1629.83350
1336	21	0	25.800	0	no	southwest	2007.94500
1337	61	0	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

In [10]:

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

Out[10]:

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

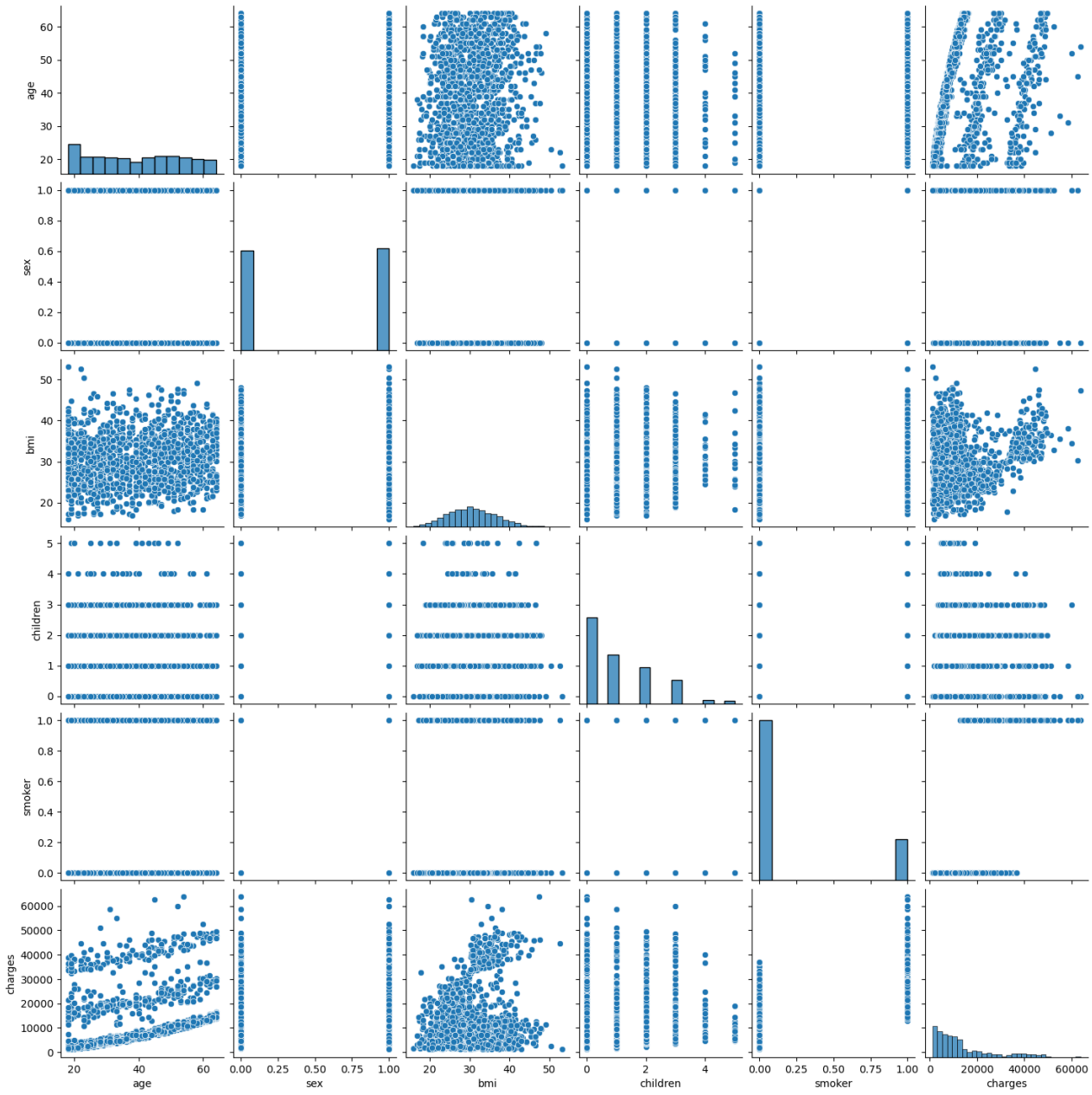
1338 rows × 7 columns

In [11]:

```
sns.pairplot(data)
```

Out[11]:

```
<seaborn.axisgrid.PairGrid at 0x231f1316080>
```

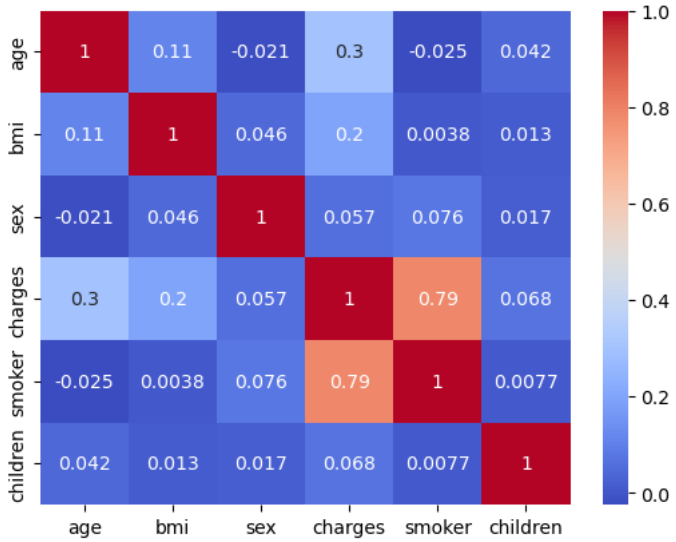


In [12]:

```
columns=data[['age','bmi','sex','charges','smoker','children']]
subset=columns.corr()
sns.heatmap(subset,annot=True,cmap='coolwarm')
```

Out[12]:

<Axes: >



In [13]:

```
from sklearn.model_selection import train_test_split
X=data[['age','bmi','sex','charges','children']]
y=data['smoker']
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=2)
```

In [14]:

```
#Now we are calculating our data good fit for linear regression model
```

In [15]:

```
from sklearn.linear_model import LinearRegression
```

In [16]:

```
lr=LinearRegression()
```

In [17]:

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

Out[17]:

▾ LinearRegression

LinearRegression()

In [18]:

```
print(lr.intercept_)
coeff_data=pd.DataFrame(lr.coef_,X.columns,columns=['coefficient'])
coeff_data
```

0.42484839188702184

Out[18]:

	coefficient
age	-0.007772
bmi	-0.010035
sex	0.019906
charges	0.000030
children	-0.017475

In [19]:

```
lr.score(x_test,y_test)
```

Out[19]:

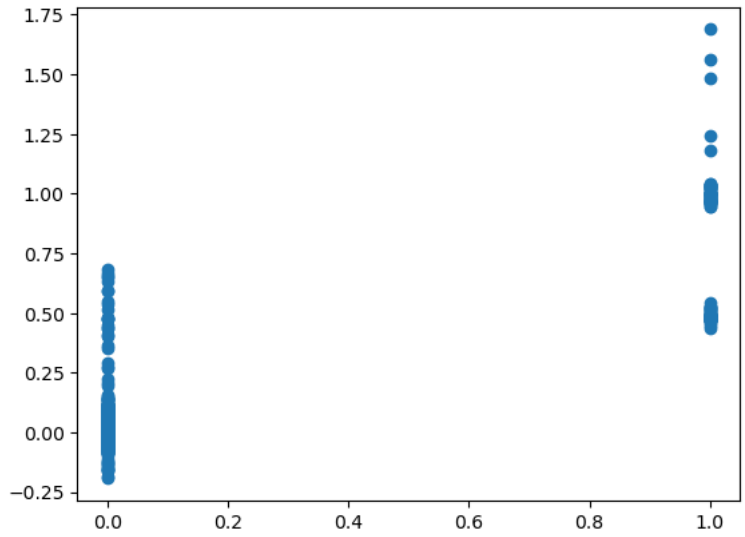
```
0.728894897343781
```

In [20]:

```
predictions=lr.predict(x_test)
plt.scatter(y_test,predictions)
```

Out[20]:

```
<matplotlib.collections.PathCollection at 0x231fba9bfa0>
```



In [21]:

```
'''In above linear regression model our insurance data is not fitted accuratly.
 so now we are on logistic regression model'''
```

Out[21]:

```
'In above linear regression model our insurance data is not fitted accuratly.\n so now we are on logistic regression model'
```

In [22]:

```
#importing libraries& dropping null values
```

In [23]:

```
x=np.array(data['charges']).reshape(-1,1)
y=np.array(data['smoker']).reshape(-1,1)
data.dropna(inplace=True)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
from sklearn.linear_model import LogisticRegression
lg=LogisticRegression()
```

In [24]:

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

```
C:\Users\jas_m\anaconda3\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was p
assed 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)
```

Out[24]:

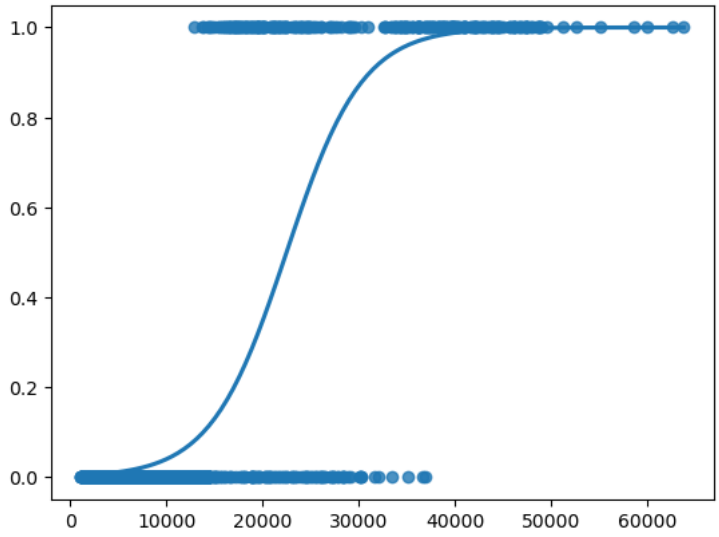
```
LogisticRegression
LogisticRegression()
```

In [25]:

```
#plotting our model
sns.regplot(x=x,y=y,data=data,logistic=True,ci=None)
```

Out[25]:

```
<Axes: >
```



In [26]:

```
lg.score(x_test,y_test)
```

Out[26]:

```
0.8930348258706468
```

In [27]:

```
"""Now we calculated the logistic regression ,
 it gives better pridiction and accuracy with compared to linear regression
 and also we are looking with DecisionTree &randomForest
 for getting more accuracy"""
```

Out[27]:

```
'Now we calculated the logistic regression ,\n it gives better pridiction and accuracy with compared to linear regression\n
and also we are looking with DecisionTree &randomForest\n for getting more accuracy'
```

In [28]:

```
#Decision tree
#importing libraries&fitting our data
from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier(random_state=0)
clf.fit(x_train,y_train)
```

Out[28]:

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

In [29]:

```
#accuracy score for desicion tree
score=clf.score(x_test,y_test)
print(score)
```

```
0.8880597014925373
```

In [30]:

```
#importing libraries& fittingdata
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

```
C:\Users\jas_m\AppData\Local\Temp\ipykernel_37128\310474025.py:4: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  rfc.fit(x_train,y_train)
```

Out[30]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [31]:

```
x_depth':[3,5,7,10,20],'min_samples_leaf':[5,10,20,50,100,200],'n_estimators':[10,25,30,50,100,200],'n_estimators':[10,25,30,50,100,200]}
```

In [32]:

```
#for finding optimal parameter values we are importing GridSearchCv
from sklearn.model_selection import GridSearchCV
grid_search=GridSearchCV(estimator=rfc,param_grid=params,cv=2,scoring="accuracy")
```

In [33]:

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

```
ector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel
().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\jas_m\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A column-v
ector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel
().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\jas_m\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A column-v
ector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel
().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\jas_m\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A column-v
ector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel
().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\jas_m\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:686: DataConversionWarning: A column-v
ector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel
().
  estimator.fit(X_train, y_train, **fit_params)
```

In [34]:

```
grid_search.best_score_
```

Out[34]:

```
0.9241452991452992
```

In [35]:

```
rf_best=grid_search.best_estimator_
rf_best
```

Out[35]:

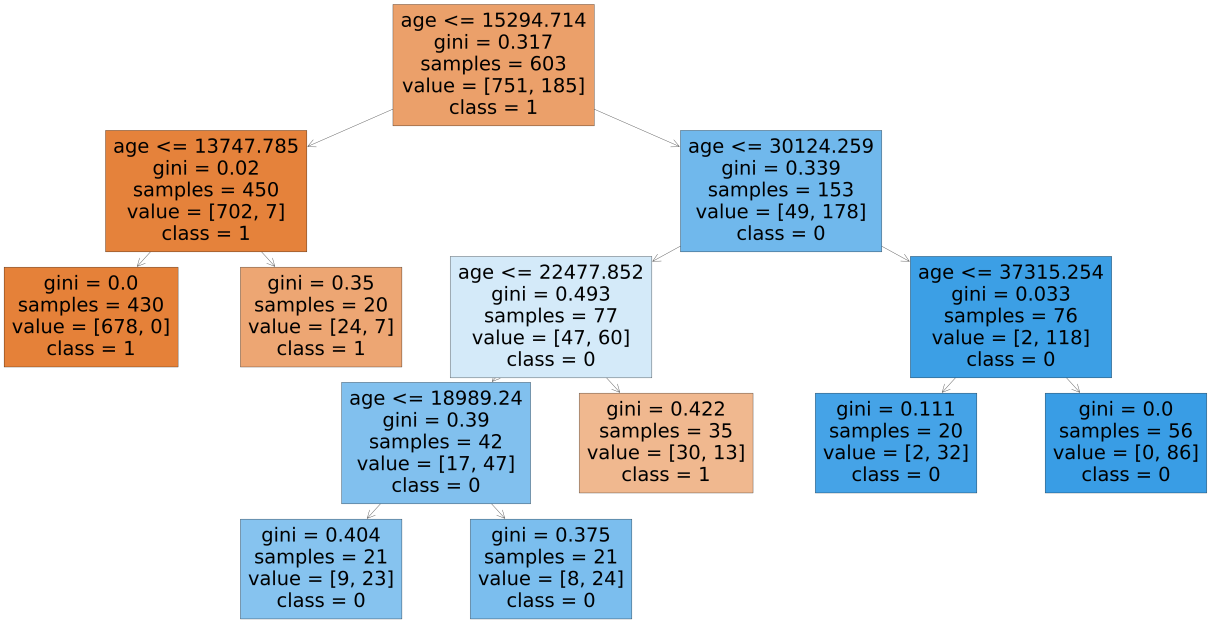
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=7, min_samples_leaf=20, n_estimators=10)
```

In [36]:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[4],feature_names=X.columns,class_names=['1','0'],filled=True)
```

Out[36]:

```
[Text(0.4090909090909091, 0.9, 'age <= 15294.714\ngini = 0.317\nsamples = 603\nvalue = [751, 185]\nclass = 1'),
 Text(0.18181818181818182, 0.7, 'age <= 13747.785\ngini = 0.02\nsamples = 450\nvalue = [702, 7]\nclass = 1'),
 Text(0.09090909090909091, 0.5, 'gini = 0.0\nsamples = 430\nvalue = [678, 0]\nclass = 1'),
 Text(0.2727272727272727, 0.5, 'gini = 0.35\nsamples = 20\nvalue = [24, 7]\nclass = 1'),
 Text(0.6363636363636364, 0.7, 'age <= 30124.259\ngini = 0.339\nsamples = 153\nvalue = [49, 178]\nclass = 0'),
 Text(0.45454545454545453, 0.5, 'age <= 22477.852\ngini = 0.493\nsamples = 77\nvalue = [47, 60]\nclass = 0'),
 Text(0.36363636363636365, 0.3, 'age <= 18989.24\ngini = 0.39\nsamples = 42\nvalue = [17, 47]\nclass = 0'),
 Text(0.2727272727272727, 0.1, 'gini = 0.404\nsamples = 21\nvalue = [9, 23]\nclass = 0'),
 Text(0.45454545454545453, 0.1, 'gini = 0.375\nsamples = 21\nvalue = [8, 24]\nclass = 0'),
 Text(0.5454545454545454, 0.3, 'gini = 0.422\nsamples = 35\nvalue = [30, 13]\nclass = 1'),
 Text(0.8181818181818182, 0.5, 'age <= 37315.254\ngini = 0.033\nsamples = 76\nvalue = [2, 118]\nclass = 0'),
 Text(0.7272727272727273, 0.3, 'gini = 0.111\nsamples = 20\nvalue = [2, 32]\nclass = 0'),
 Text(0.9090909090909091, 0.3, 'gini = 0.0\nsamples = 56\nvalue = [0, 86]\nclass = 0')]
```



In [37]:

```
#accurate score for random forest
score=rfc.score(x_test,y_test)
print(score)
```

```
0.8880597014925373
```

In [38]:

```
"""In above all three models we more accuracy in LINEAR REGRESSION
 with respcet to other two models"""
```

Out[38]:

```
'In above all three models we more accuracy in LINEAR REGRESSION\n with respcet to other two models'
```

In [39]:

```
#calculating r2 error
from sklearn.metrics import r2_score
```

In [40]:

```
prediction=lg.predict(x_test)
```

In [41]:

```
prediction
```

Out[41]:

```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 1], dtype=int64)
```

In [42]:

```
r2=r2_score(y_test,prediction)
r2
```

Out[42]:

```
0.33517941617630076
```

In [43]:

```
#mean absolute errror
from sklearn.metrics import mean_absolute_error
from sklearn import metrics
metrics.mean_absolute_error(y_test,prediction)
```

Out[43]:

```
0.10696517412935323
```

In [44]:

```
#root mean square error
from sklearn.metrics import mean_squared_error
np.sqrt(metrics.mean_squared_error(y_test,prediction))
```

Out[44]:

```
0.3270553074471552
```

In [45]:

```
"""With this insurance Dataset we are concluded that LOGISTIC REGRESSION
 is best-fit for predicting the values with respect to other regression models"""
```

Out[45]:

```
'With this insurance Dataset we are concluded that LOGISTIC REGRESSION\n is best-fit for predicting the values with respect
to other regression models'
```

In [ ]: