Problem Statement: To predict how best the data set

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
df=pd.read_csv(r"C:\Users\HP\Downloads\insurance (1).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

Data cleaning & Data 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.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.shape
```

Out[6]:
```
(1338, 7)
```

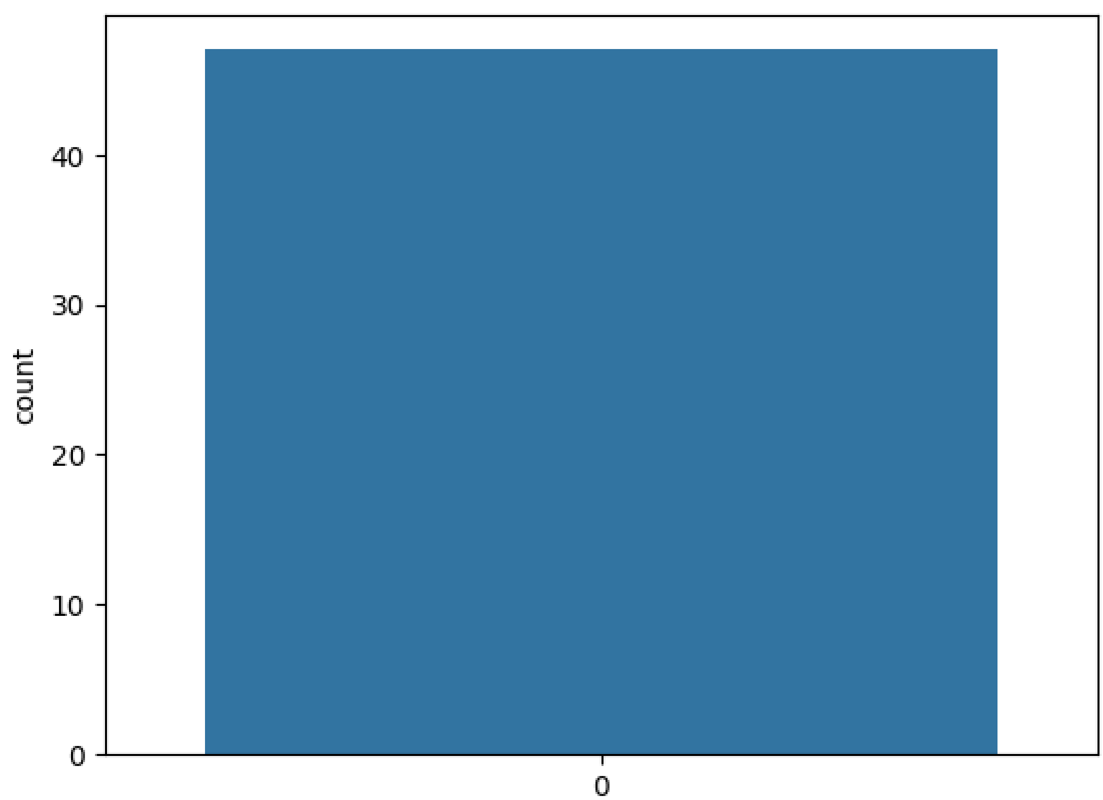
In [7]:
```
df.describe()
```

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]:
```
sns.countplot(df['age'].unique())
```

Out[8]: <Axes: ylabel='count'>



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

Out[9]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

In [10]:
```
df.columns
```

Out[10]:
```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtyp
e='object')
```

In [11]:
```
smoker={"smoker":{"yes":1,"no":0}}
df=df.replace(smoker)
df
```

Out[11]:

	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
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	0	northwest	10600.54830
1334	18	female	31.920	0	0	northeast	2205.98080
1335	18	female	36.850	0	0	southeast	1629.83350
1336	21	female	25.800	0	0	southwest	2007.94500
1337	61	female	29.070	0	1	northwest	29141.36030

1338 rows × 7 columns

In [12]:
```
sex={"sex":{"male":1,"female":0}}
df=df.replace(sex)
df
```

Out[12]:

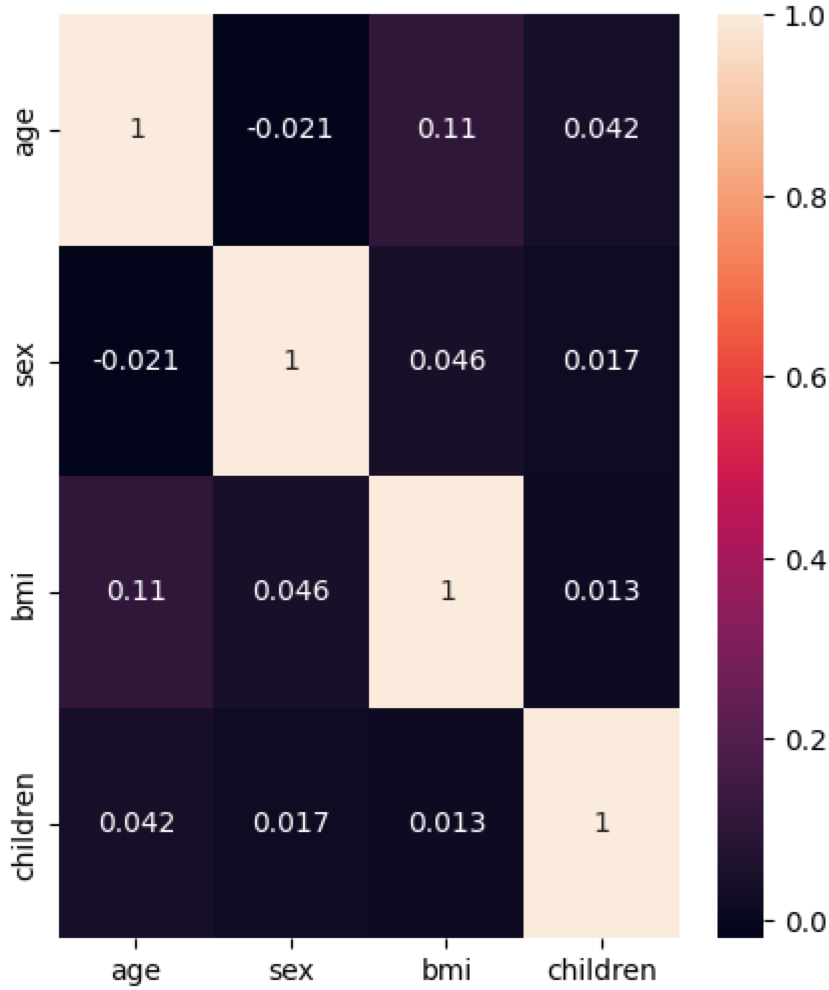
	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	southwest	16884.92400
1	18	1	33.770	1	0	southeast	1725.55230
2	28	1	33.000	3	0	southeast	4449.46200
3	33	1	22.705	0	0	northwest	21984.47061
4	32	1	28.880	0	0	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	1	30.970	3	0	northwest	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

Data Visualisation

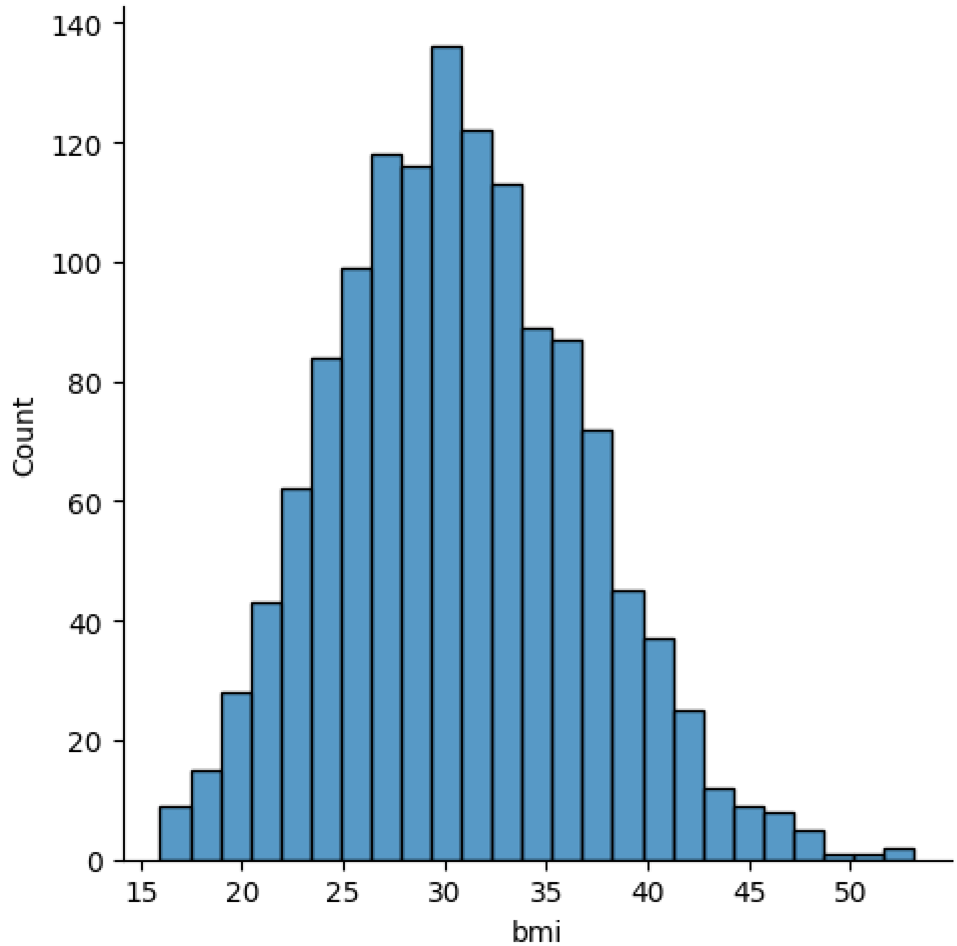
In [13]:
```python
idf=df[['age', 'sex', 'bmi', 'children']]
plt.figure(figsize=(5,6))
sns.heatmap(idf.corr(),annot=True)
```

Out[13]: <Axes: >

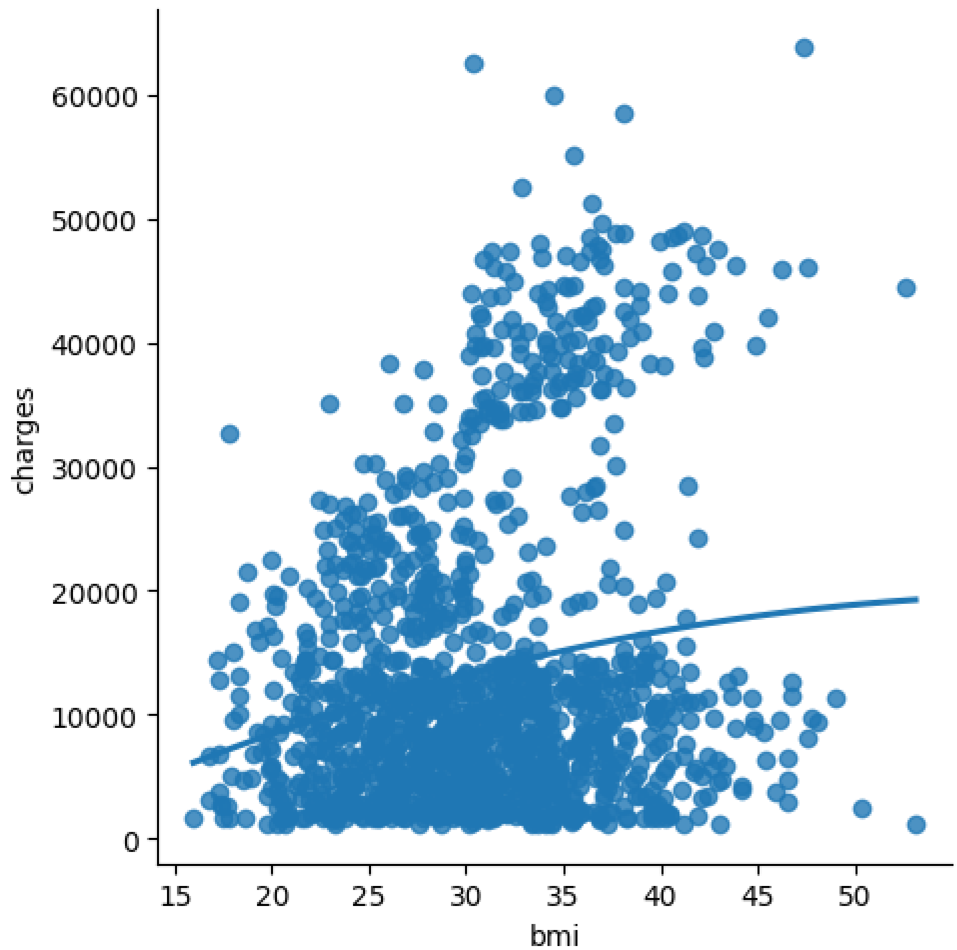


In [14]:
```
sns.displot(df['bmi'])
```

Out[14]: <seaborn.axisgrid.FacetGrid at 0x1b061c77fd0>



In [15]:
```
sns.lmplot(x='bmi',y='charges',order=2,data=df,ci=None)
plt.show()
```



Linear Rgression

In [16]:
```
feature=df.columns[0:3]
target=df.columns[-1]
x=df[feature].values
y=df[target].values
```

In [17]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25)
regr=LinearRegression()
regr.fit(x_train,y_train)
print(regr.score(x_test,y_test))
```

```
0.07228473092692334
```

Logistic Regression

In [23]:
```
lg = LogisticRegression()
lg.fit(x_train,y_train)
print(lg.score(x_test,y_test))
print(lg.score(x_train,y_train))
```

```
0.9492537313432836
0.9491525423728814
```

Decision Tree

In [19]:
```
x=["age","sex","bmi","children","charges"]
y=["yes","No"]
all_inputs=df[x]
all_classes=df["smoker"]
```

In [20]:
```
x_train,x_test,y_train,y_test=train_test_split(all_inputs,all_classes,test_siz
```

In [21]:
```
clt=DecisionTreeClassifier(random_state=0)
clt.fit(x_train,y_train)
```

Out[21]:
```
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 [22]:
```
score=clt.score(x_test,y_test)
print(score)
```

```
0.9492537313432836
```

Random Forest

In [24]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
score=rfc.score(x_test,y_test)
score1=rfc.score(x_train,y_train)
print(score,score1)
```

```
0.9582089552238806 1.0
```

In [25]:
```
rf=RandomForestClassifier()
```

In [26]:
```python
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 [27]:
```python
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[27]:
```
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.

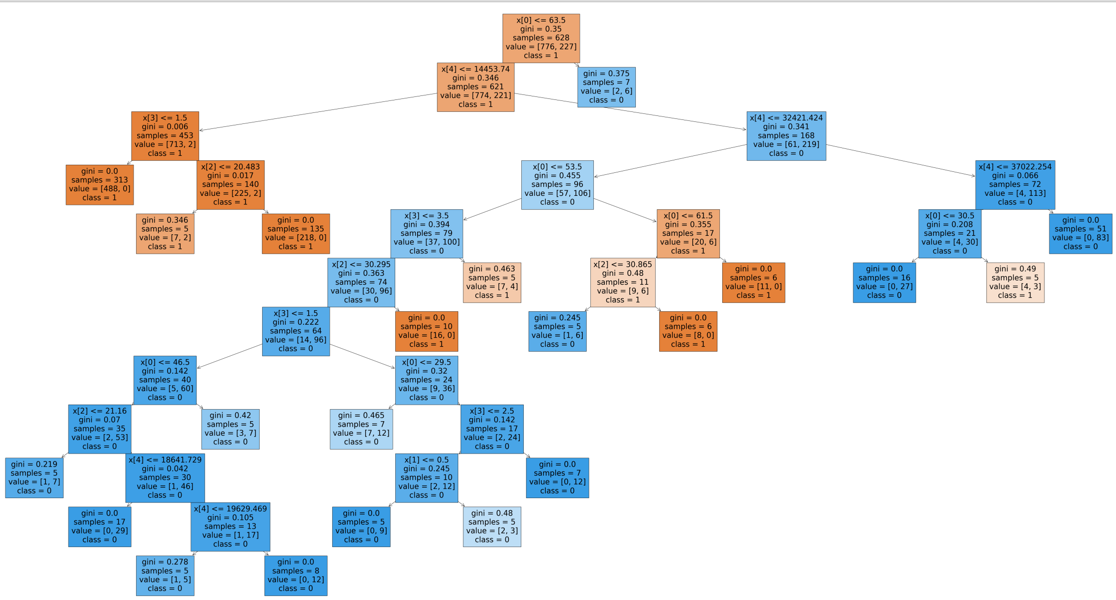
In [28]:
```python
grid_search.best_score_
```

Out[28]:
```
0.9661016612193939
```

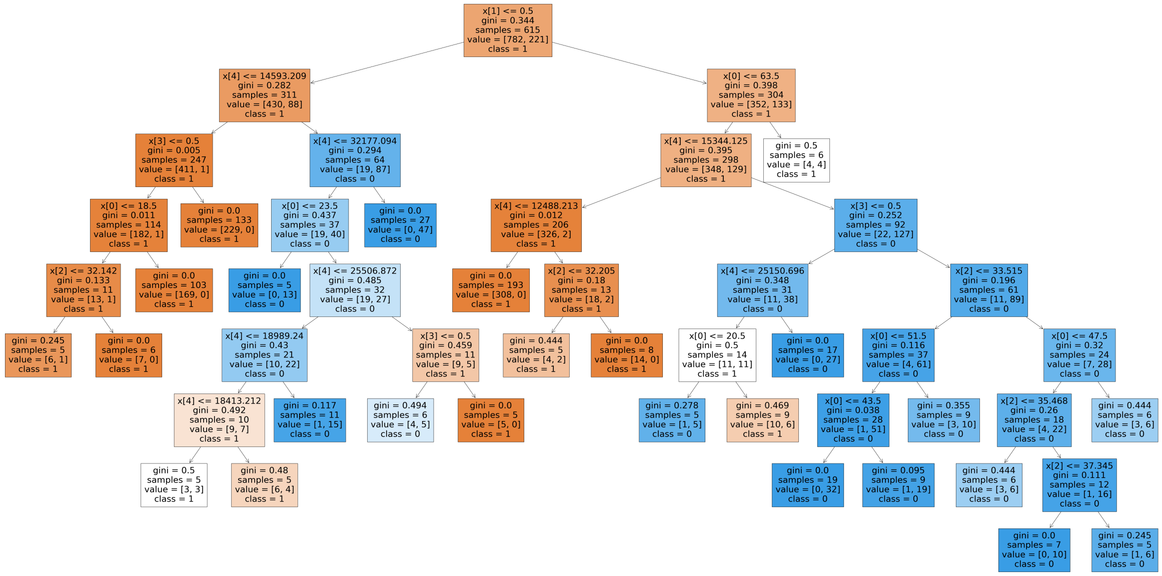
In [29]:
```python
rf_best=grid_search.best_estimator_
print(rf_best)
```

```
RandomForestClassifier(max_depth=20, min_samples_leaf=5, n_estimators=30)
```

In [31]:
```python
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[5],class_names=["1","0"],filled=True);
```



In [34]:
```python
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[7],class_names=["1","0"],filled=True);
```



In [35]:
```python
rf_best.feature_importances_
```

Out[35]:
```
array([0.03276102, 0.00815141, 0.06358871, 0.01448967, 0.88100918])
```

In [36]:
```python
imp_df=pd.DataFrame({"Varname":x_train.columns,"Imp":rf_best.feature_importanc
imp_df.sort_values(by="Imp",ascending=False)
```

Out[36]:

	Varname	Imp
4	charges	0.881009
2	bmi	0.063589
0	age	0.032761
3	children	0.014490
1	sex	0.008151

Conclusion:Based on accuracy scores of all models that were implimented we can conclude that "Logistic Regression" is best model for the given data set.

In [ ]:
```python
```