MINI PROJECT

PROBLEM STATEMENT : Which model is suitable for Insurance Dataset

Importing Packages

Read the Data

In [2]:

```
#importing packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv(r"C:\Users\G S R  KARTHIK\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

Data Collection and Preprocessing

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.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]:

```
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 [8]:

```
#to check null values
df.isnull().sum()
```

Out[8]:

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

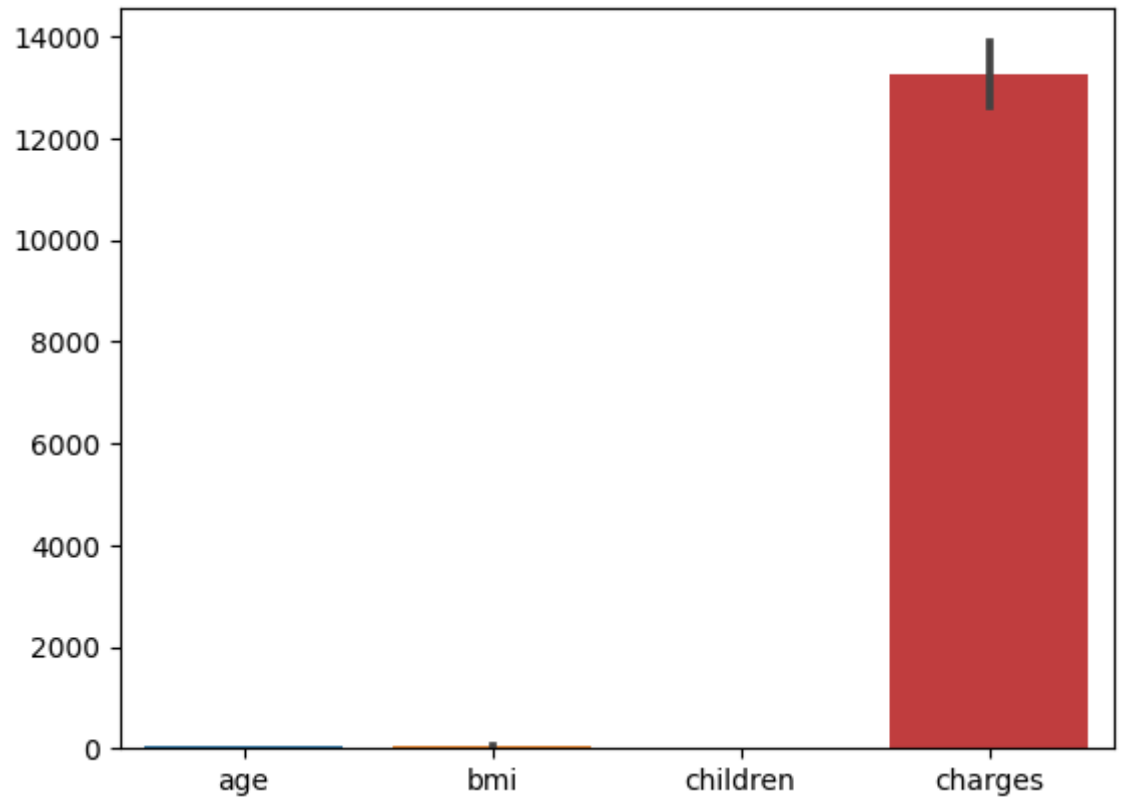
Data Visualization

In [9]:

```
#Exploratory Data Analysis
sns.barplot(df)
```

Out[9]:

```
<Axes: >
```



In [10]:

```
df.columns
```

Out[10]:

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

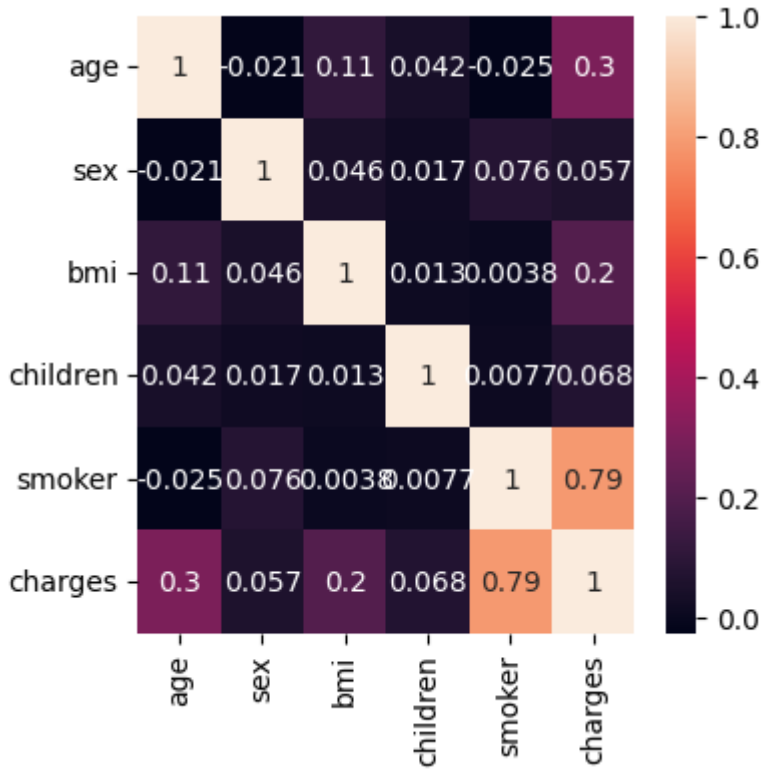
1338 rows × 7 columns

In [13]:

```
idf=df[['age', 'sex', 'bmi', 'children', 'smoker', 'charges']]
plt.figure(figsize=(4,4))
sns.heatmap(idf.corr(),annot=True)
```

Out[13]:

```
<Axes: >
```



Feature Scaling : To Split the data into training data and test data

In [14]:

```
#Training the model
X=df[['age', 'sex', 'bmi', 'children', 'smoker']]
y=df['charges']
```

Applying Linear Regression

In [15]:

```
#Linear Regression
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=100)
```

In [16]:

```
from sklearn.linear_model import LinearRegression
regr=LinearRegression()
regr.fit(X_train,y_train)
print(regr.intercept_)
coeff_df=pd.DataFrame(regr.coef_,X.columns,columns=['coefficient'])
coeff_df
```

```
-10719.483493479494
```

Out[16]:

	coefficient
age	259.757578
sex	18.216925
bmi	277.903898
children	461.169867
smoker	23981.741027

In [17]:

```
score=regr.score(X_test,y_test)
print(score)
```

```
0.780095696440481
```

In [18]:

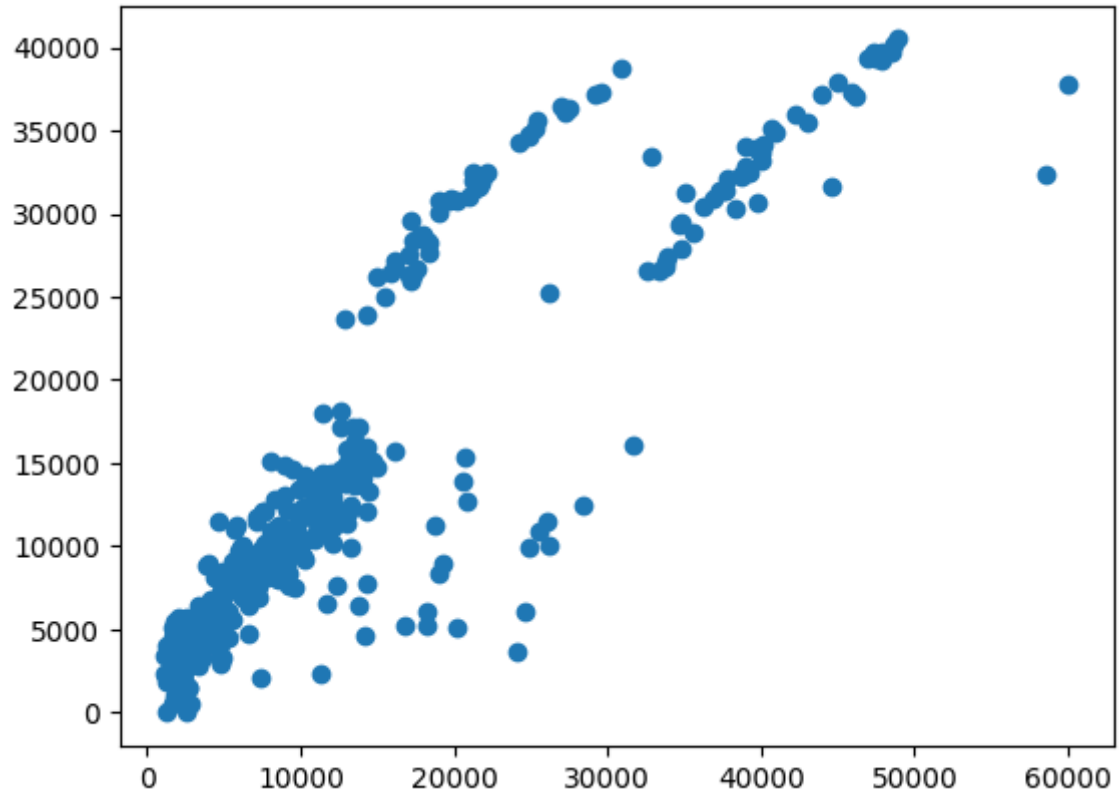
```
predictions=regr.predict(X_test)
```

In [19]:

```
plt.scatter(y_test,predictions)
```

Out[19]:

```
<matplotlib.collections.PathCollection at 0x1d48f374a30>
```



In [20]:

```
x=np.array(df['smoker']).reshape(-1,1)
y=np.array(df['charges']).reshape(-1,1)
df.dropna(inplace=True)
```

In [21]:

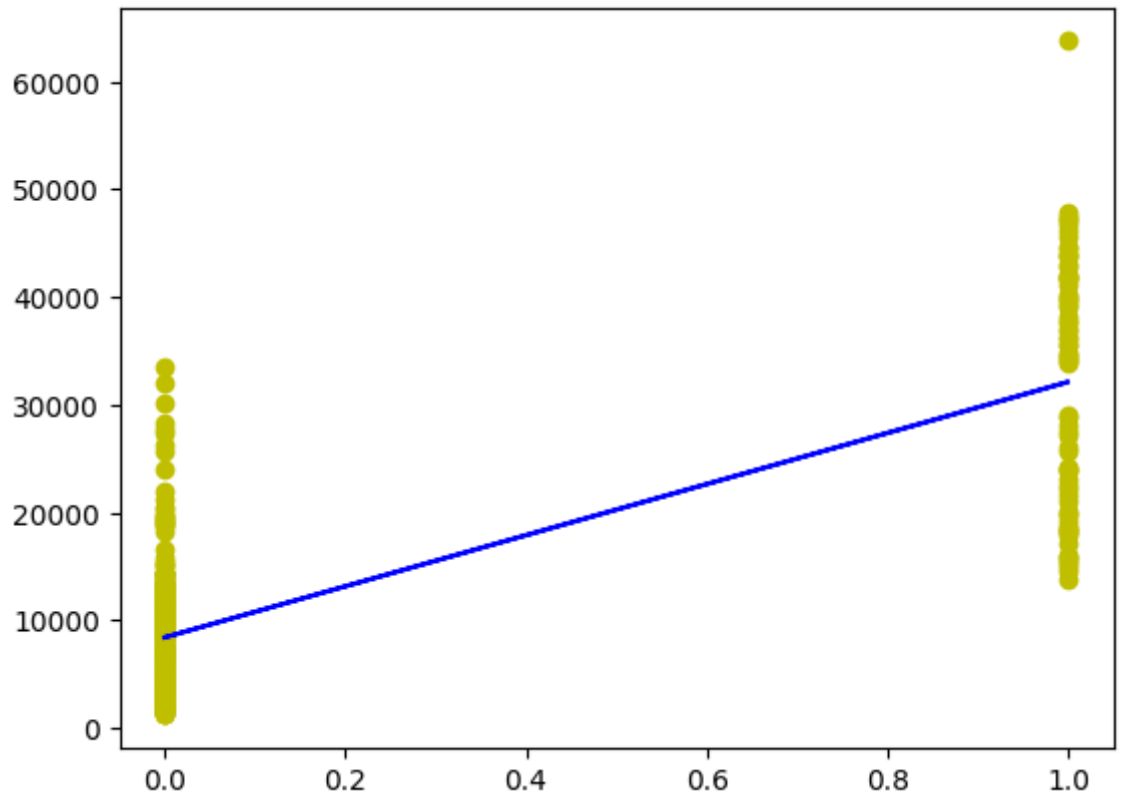
```
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
regr.fit(X_train,y_train)
regr.fit(X_train,y_train)
```

Out[21]:

```
LinearRegression
LinearRegression()
```

In [22]:

```
y_pred=regr.predict(X_test)
plt.scatter(X_test,y_test,color='y')
plt.plot(X_test,y_pred,color='b')
plt.show()
```



Since we did not get the accuracy for Linear Regression we are going to implement Logistic Regression

Logistic Regression

In [23]:

```
#Logistic Regression
x=np.array(df['charges']).reshape(-1,1)
y=np.array(df['smoker']).reshape(-1,1)
df.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
lr=LogisticRegression(max_iter=10000)
```

In [24]:

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

```
C:\Users\G S R  KARTHIK\AppData\Local\Programs\Python\Python310\lib\site-p
ackages\sklearn\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_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[24]:

```
LogisticRegression
LogisticRegression(max_iter=10000)
```

In [25]:

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

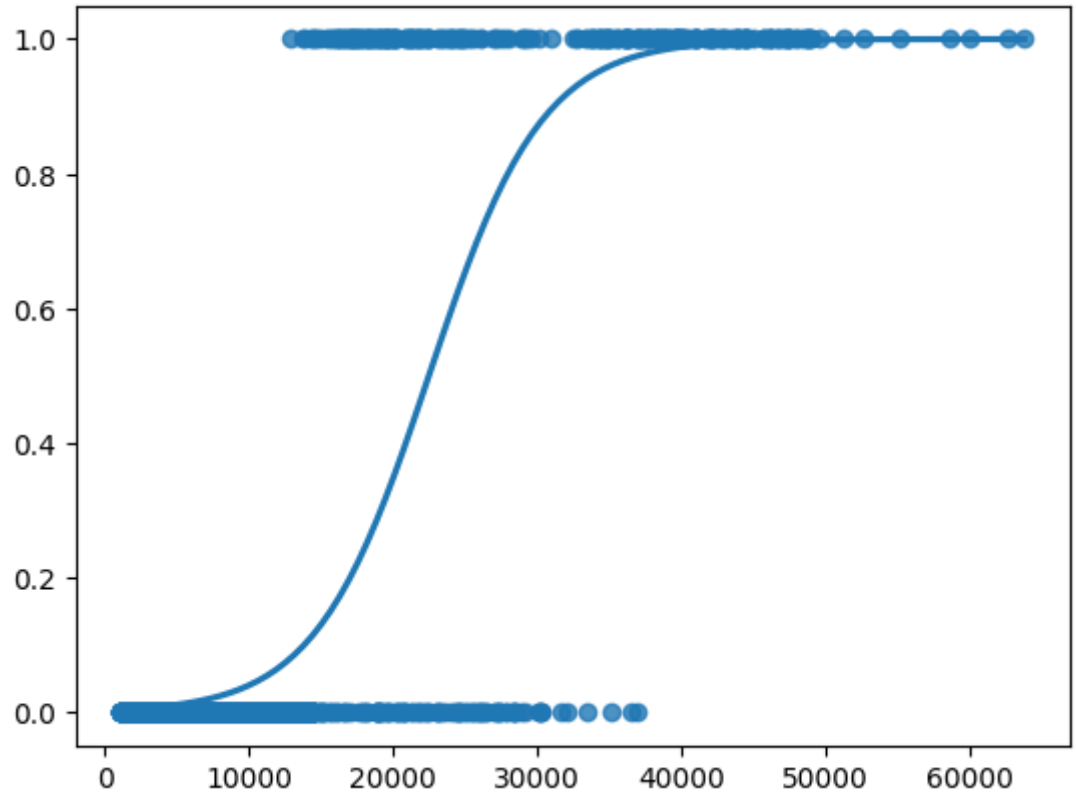
```
0.8930348258706468
```

In [26]:

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

Out[26]:

```
<Axes: >
```



We got the best fit curve for Logistic Regression .Now we are going to check that if we may get better accuracy by implementing Decision Tree and Random Forest

Decision Tree

In [27]:

```
#Decision tree
from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier(random_state=0)
clf.fit(x_train,y_train)
```

Out[27]:

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

In [28]:

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

```
0.8880597014925373
```

Random Forest

In [29]:

```
#Random forest classifier
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(X_train,y_train)
```

```
C:\Users\G S R  KARTHIK\AppData\Local\Temp\ipykernel_19992\1232785509.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[29]:

```
RandomForestClassifier
RandomForestClassifier()
```

In [30]:

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

In [31]:

```
from sklearn.model_selection import GridSearchCV
grid_search=GridSearchCV(estimator=rfc,param_grid=params,cv=2,scoring="accuracy")
```

In [32]:

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

```
C:\Users\G S R  KARTHIK\AppData\Local\Programs\Python\Python310\lib\sit
e-packages\sklearn\model_selection\_validation.py:686: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Pleas
e change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\G S R  KARTHIK\AppData\Local\Programs\Python\Python310\lib\sit
e-packages\sklearn\model_selection\_validation.py:686: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Pleas
e change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\G S R  KARTHIK\AppData\Local\Programs\Python\Python310\lib\sit
e-packages\sklearn\model_selection\_validation.py:686: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Pleas
e change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
C:\Users\G S R  KARTHIK\AppData\Local\Programs\Python\Python310\lib\sit
e-packages\sklearn\model_selection\_validation.py:686: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Pleas
e change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
```

In [33]:

```
grid_search.best_score_
```

Out[33]:

0.7938034188034188

In [34]:

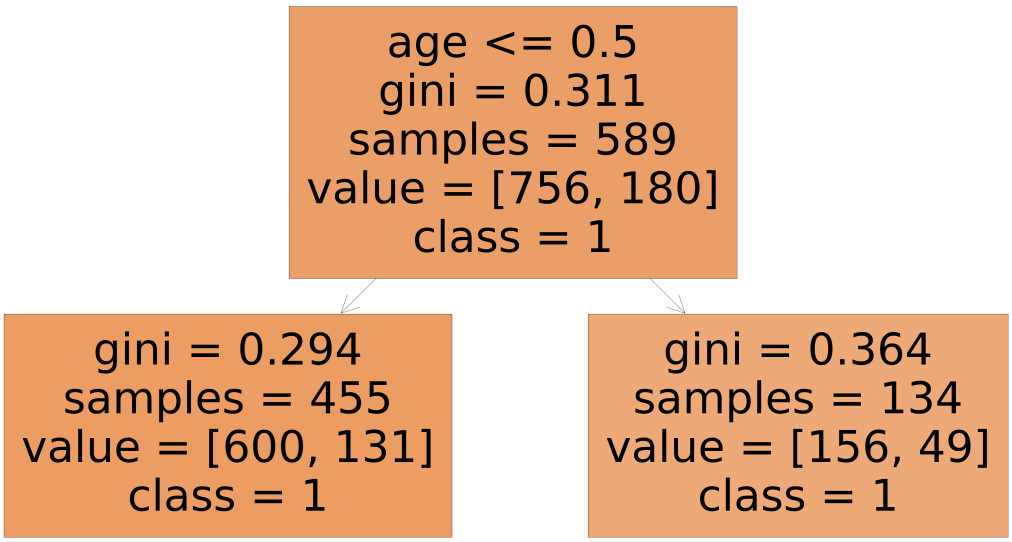
```
rf_best=grid_search.best_estimator_
rf_best
```

Out[34]:

```
RandomForestClassifier
RandomForestClassifier(max_depth=2, min_samples_leaf=5, n_estimators=10)
```

In [35]:

```
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=Tr
```



In [36]:

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

```
0.7985074626865671
```

CONCLUSION : Based on accuracy scores of all models that were implemented we can conclude that "Logistic Regression" is the best model for the given dataset

In [ ]: