INSURANCE Dataset

PROBLEM STATEMENT:Which model is suitable for insurance dataset

Import Libraries

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

In [2]:
```
df=pd.read_csv(r"C:\Users\DELL\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 & 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 [36]:
```
df.tail()
```

Out[36]:

	age	sex	bmi	children	smoker	region	charges
1333	50	1	30.97	3	no	2	10600.5483
1334	18	2	31.92	0	no	3	2205.9808
1335	18	2	36.85	0	no	1	1629.8335
1336	21	2	25.80	0	no	0	2007.9450
1337	61	2	29.07	0	yes	2	29141.3603

In [4]:
```
df.shape
```

Out[4]:
```
(1338, 7)
```

In [5]:
```
df.describe()
```

Out[5]:

	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 [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]:
```
convert={"sex":{"male":1,"female":2}}
df=df.replace(convert)
print(df)
```

```
      age  sex     bmi  children smoker     region      charges
0      19    2  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    2  31.920         0     no  northeast   2205.98080
1335   18    2  36.850         0     no  southeast   1629.83350
1336   21    2  25.800         0     no  southwest   2007.94500
1337   61    2  29.070         0    yes  northwest  29141.36030

[1338 rows x 7 columns]
```

In [8]:
```
convert={"region":{"southwest":0,"southeast":1,"northwest":2,"northeast":3}}
df=df.replace(convert)
print(df)
```

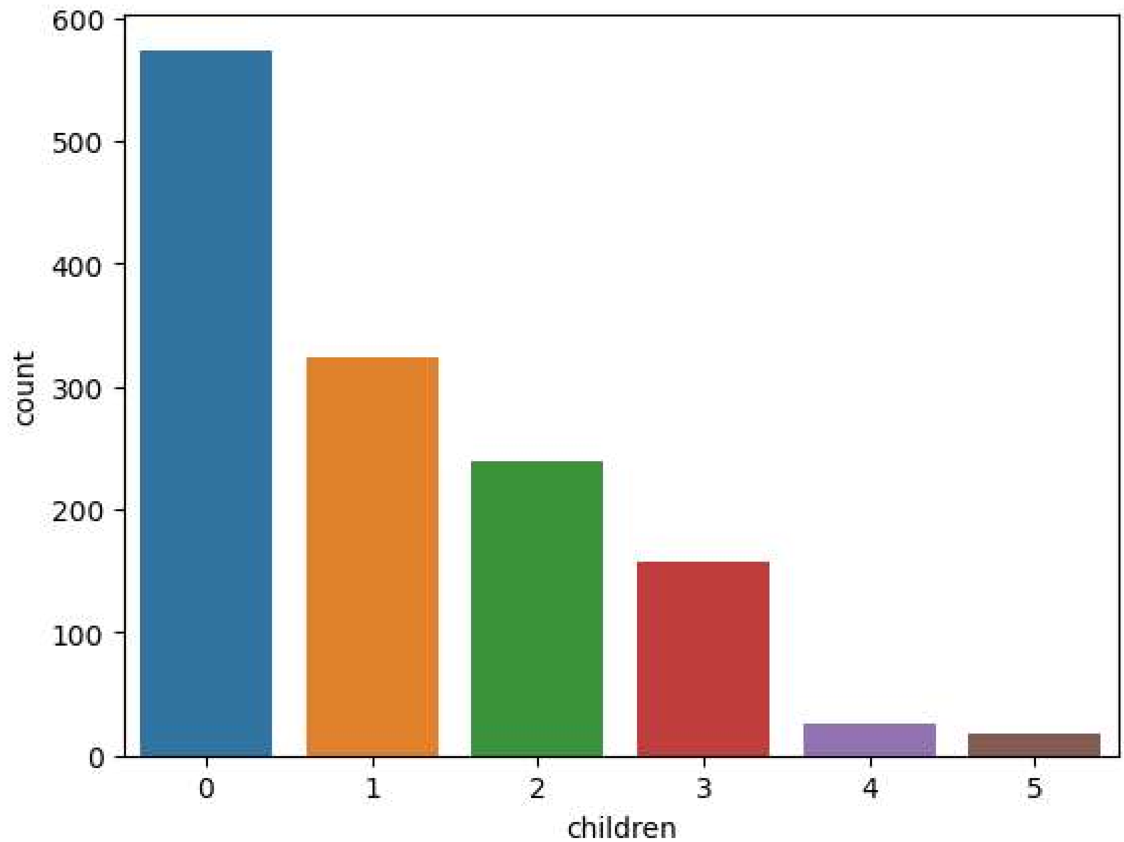
```
      age  sex     bmi  children smoker  region      charges
0      19    2  27.900         0    yes       0  16884.92400
1      18    1  33.770         1     no       1   1725.55230
2      28    1  33.000         3     no       1   4449.46200
3      33    1  22.705         0     no       2  21984.47061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

[1338 rows x 7 columns]
```

In [9]:
```
x=["age","sex","bmi","children","region","charges"]
y=["yes","No"]
all_inputs=df[x]
all_classes=df["smoker"]
```

Data Visualisation

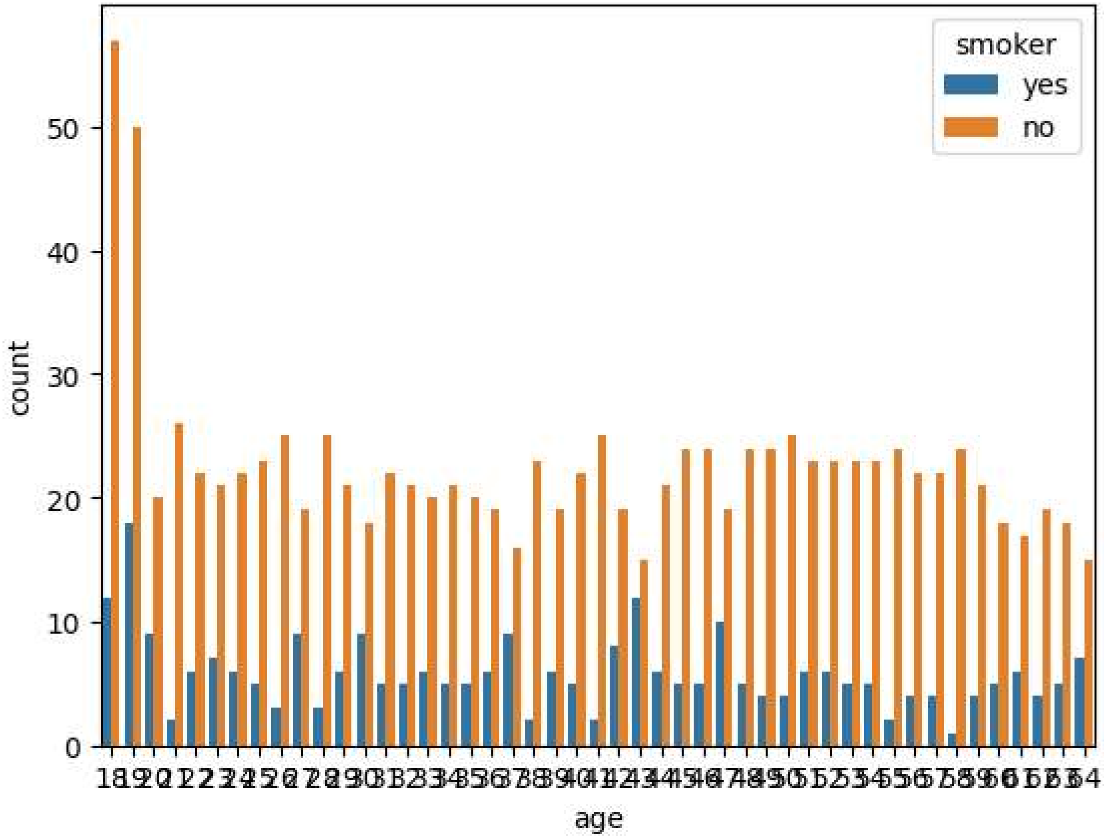
In [28]:
```
sns.countplot(x="children", data=df)
```

Out[28]: <Axes: xlabel='children', ylabel='count'>



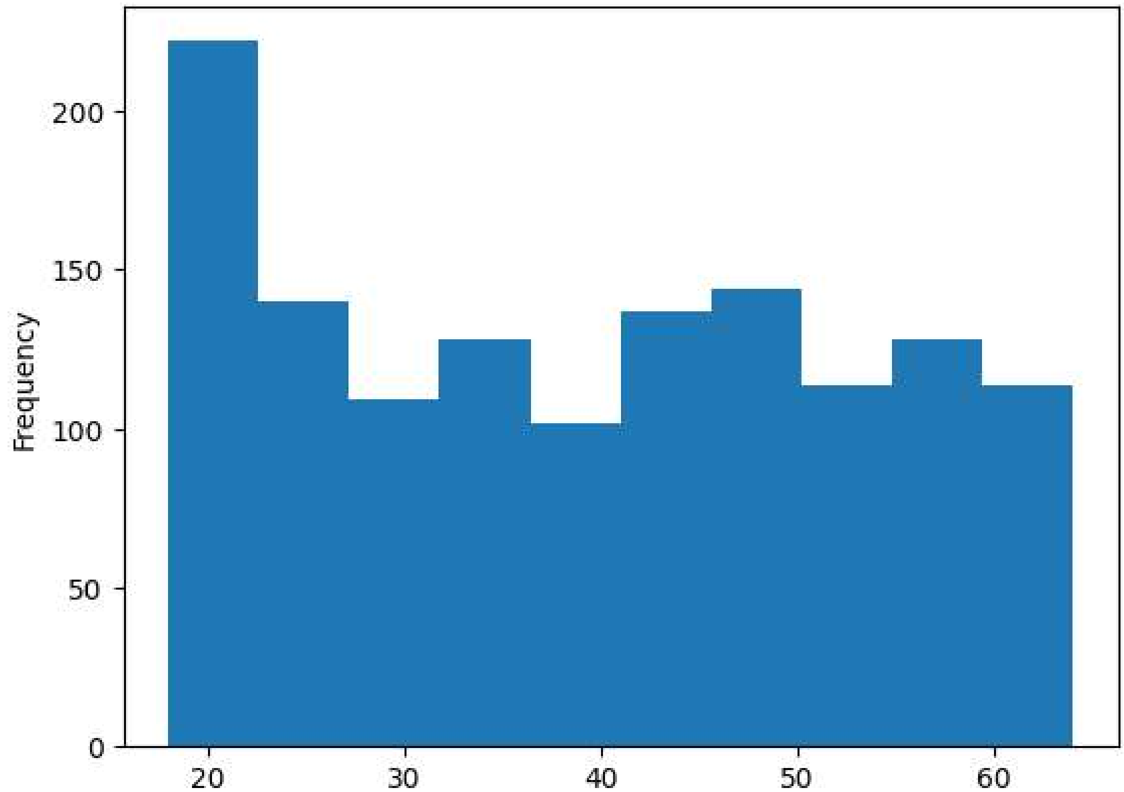
In [29]:
```
sns.countplot(x="age",hue="smoker",data=df)
```

Out[29]: <Axes: xlabel='age', ylabel='count'>



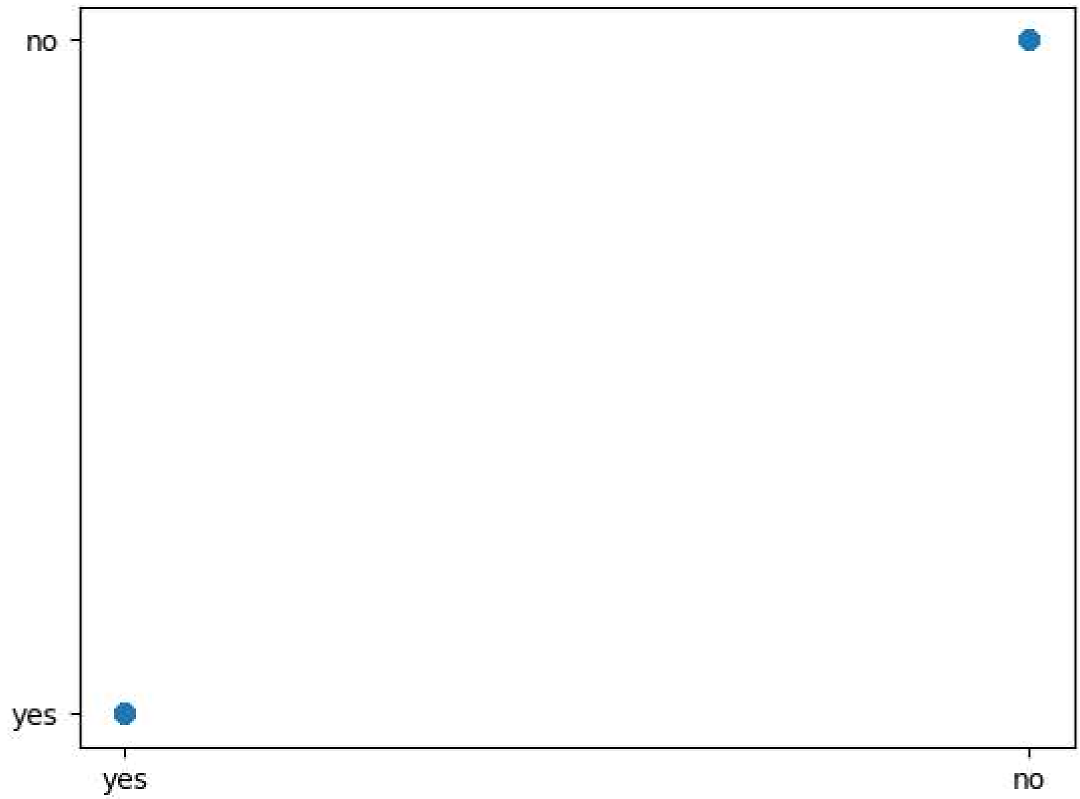
In [30]:
```
df["age"].plot.hist()
```

Out[30]: <Axes: ylabel='Frequency'>



In [42]:
```
predictions=clt.predict(x_train)
plt.scatter(y_train,predictions)
```

Out[42]: <matplotlib.collections.PathCollection at 0x2de1f4359d0>



Linear Regression

In [11]:
```
feature=df.columns[0:3]
target=df.columns[-1]
x=df[feature].values
y=df[target].values
```

In [12]:
```
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))
```

```
0.02843532430351725
```

Logistic Regression

In [31]:
```
lg = LogisticRegression()
lg.fit(x_train,y_train)
print(lg.score(x_test,y_test))
print(lg.score(x_train,y_train))
```

```
0.9552238805970149
0.9202392821535393
```

Decision Tree

In [15]:
```
x_train,x_test,y_train,y_test=train_test_split(all_inputs,all_classes,test_size
```

In [16]:
```
clt=DecisionTreeClassifier(random_state=0)
clt.fit(x_train,y_train)
```

Out[16]:
```
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 [17]:
```
score=clt.score(x_test,y_test)
print(score)
```

```
0.9492537313432836
```

RANDOM FOREST

In [22]:
```
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)
```

```
0.9701492537313433 1.0
```

In [23]:
```
rf=RandomForestClassifier()
```

In [24]:
```
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 [25]:
```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[25]:
```
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.

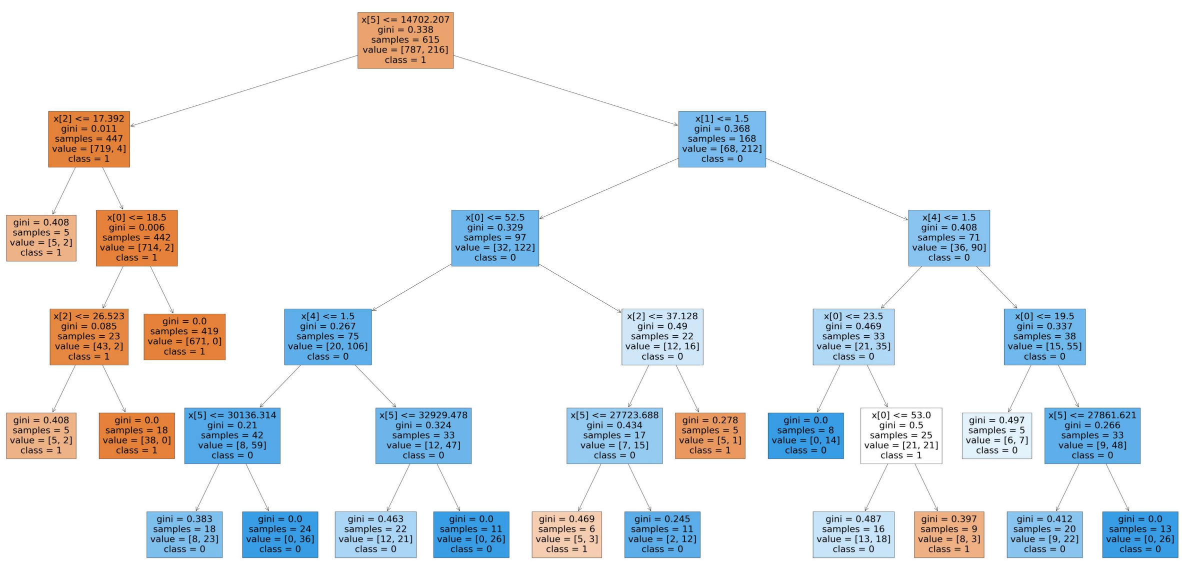
In [26]:
```python
grid_search.best_score_
```

Out[26]:
```
0.9531435137692742
```

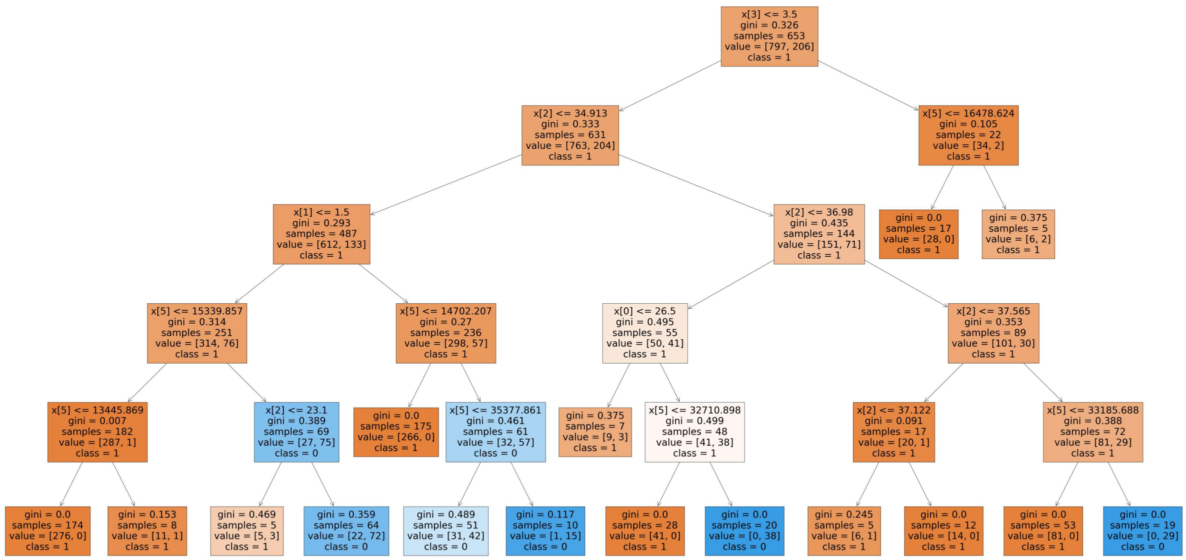
In [27]:
```python
rf_best=grid_search.best_estimator_
print(rf_best)
```

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

In [32]:
```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 [37]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[7],class_names=["1","0"],filled=True);
```



In [34]:
```
rf_best.feature_importances_
```

Out[34]:
```
array([0.03427488, 0.0059298 , 0.04189636, 0.01141585, 0.008755  ,
       0.89772811])
```

In [35]:
```
imp_df=pd.DataFrame({"Varname":x_train.columns,"Imp":rf_best.feature_importance
imp_df.sort_values(by="Imp",ascending=False)
```

Out[35]:

	Varname	Imp
5	charges	0.897728
2	bmi	0.041896
0	age	0.034275
3	children	0.011416
4	region	0.008755
1	sex	0.005930

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