Medical Insurance Price Prediction

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as pt
import warnings
warnings.filterwarnings("ignore")
```

```
df=pd.read_csv("/content/insurance.csv")
df
```

	age	sex	bmi	children	smoker	region	expenses
0	19	female	27.9	0	yes	southwest	16884.92
1	18	male	33.8	1	no	southeast	1725.55
2	28	male	33.0	3	no	southeast	4449.46
3	33	male	22.7	0	no	northwest	21984.47
4	32	male	28.9	0	no	northwest	3866.86
...	...	...	...	...	...	...	...
1333	50	male	31.0	3	no	northwest	10600.55
1334	18	female	31.9	0	no	northeast	2205.98
1335	18	female	36.9	0	no	southeast	1629.83
1336	21	female	25.8	0	no	southwest	2007.95
1337	61	female	29.1	0	yes	northwest	29141.36

1338 rows × 7 columns

```
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   expenses  1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

```
df.describe()
```

	age	bmi	children	expenses
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.665471	1.094918	13270.422414
std	14.049960	6.098382	1.205493	12110.011240
min	18.000000	16.000000	0.000000	1121.870000
25%	27.000000	26.300000	0.000000	4740.287500
50%	39.000000	30.400000	1.000000	9382.030000
75%	51.000000	34.700000	2.000000	16639.915000
max	64.000000	53.100000	5.000000	63770.430000

```
df.isnull().sum()
```

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

Pie chart for the sex, smoker, and region column

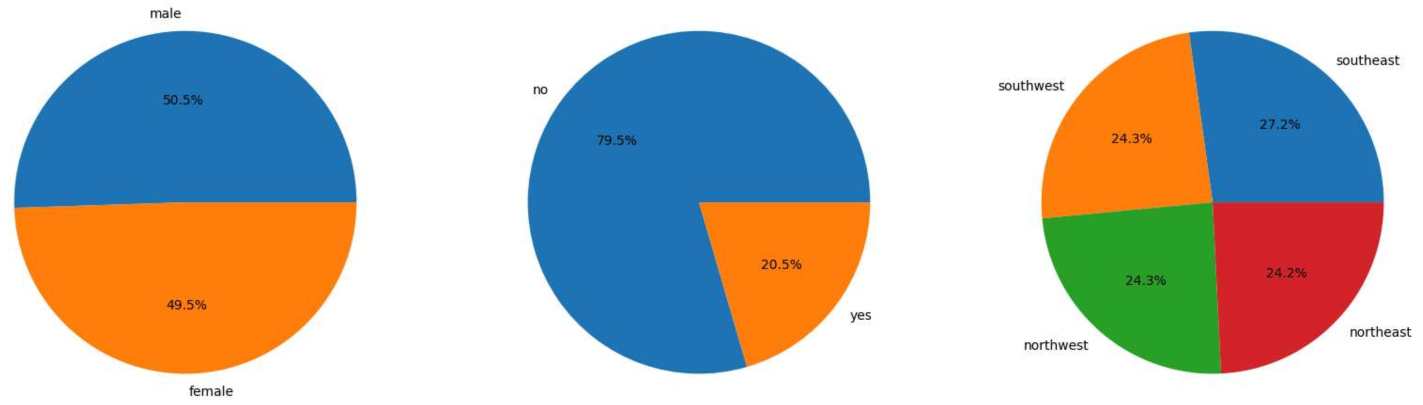
```
features = ['sex', 'smoker', 'region']

plt.subplots(figsize=(20, 10))
for i, col in enumerate(features = ['age', 'bmi']

plt.subplots(figsize=(17, 7))
for i, col in enumerate(features):
    plt.subplot(1, 2, i + 1)
    sb.scatterplot(data=df, x=col,
                y='charges',
                hue='smoker')
plt.show()
features):
    plt.subplot(1, 3, i + 1)

    x = df[col].value_counts()
    plt.pie(x.values,
            labels=x.index,
            autopct='%1.1f%%')

plt.show()
```



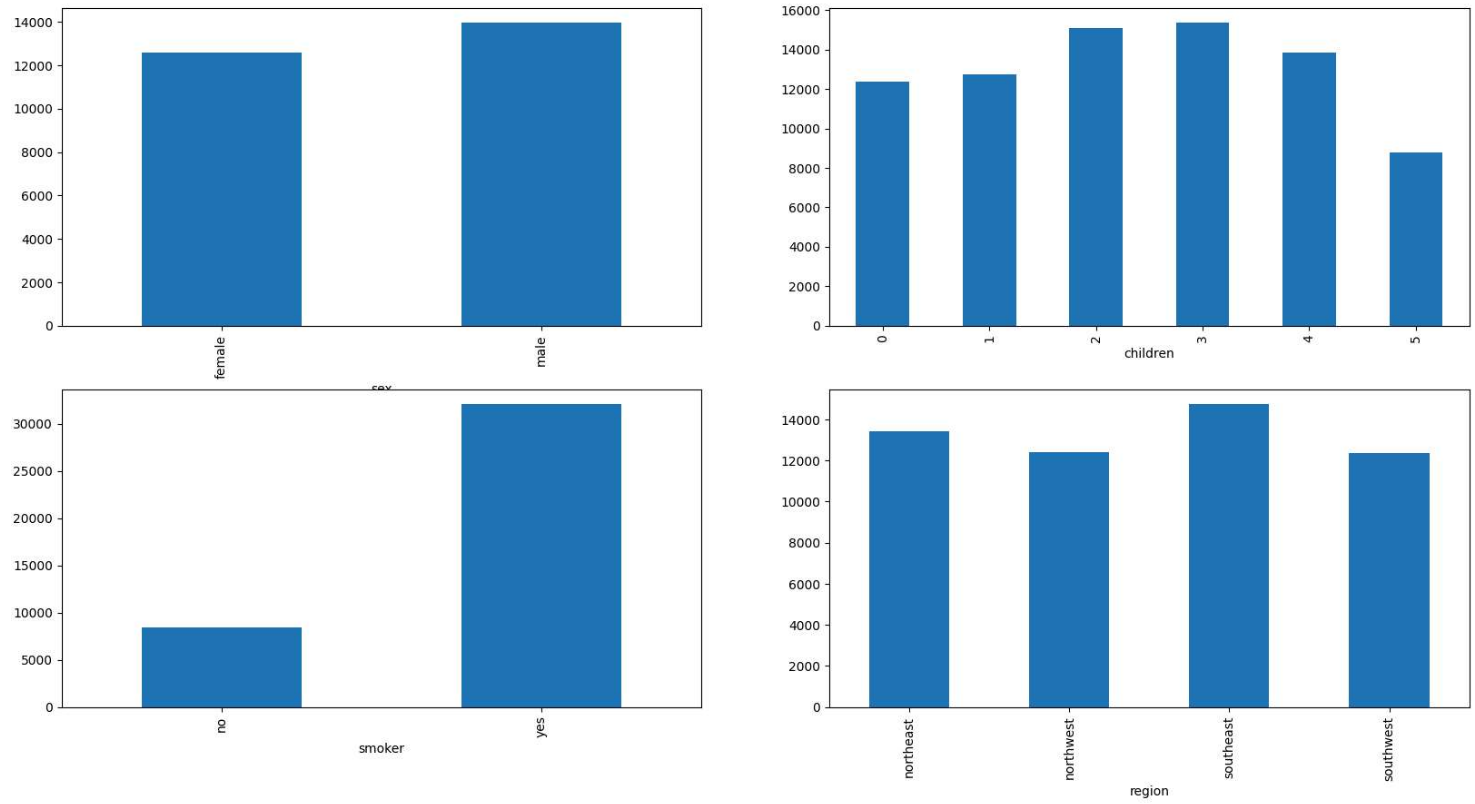
Comparison between expenses paid between different groups

```
import pandas as pd
import matplotlib.pyplot as plt

# Convert 'charges' column to numeric, handling errors
# Check if 'expenses' column exists before conversion
if 'expenses' in df.columns:
    df['expenses'] = pd.to_numeric(df['expenses'], errors='coerce')

features = ['sex', 'children', 'smoker', 'region']

plt.subplots(figsize=(20, 10))
for i, col in enumerate(features):
    plt.subplot(2, 2, i + 1)
    # Handle potential missing values in 'expenses'
    df.groupby(col)['expenses'].mean().plot.bar()
plt.show()
```



Scatter plot of the charges paid v/s age and BMI respectively

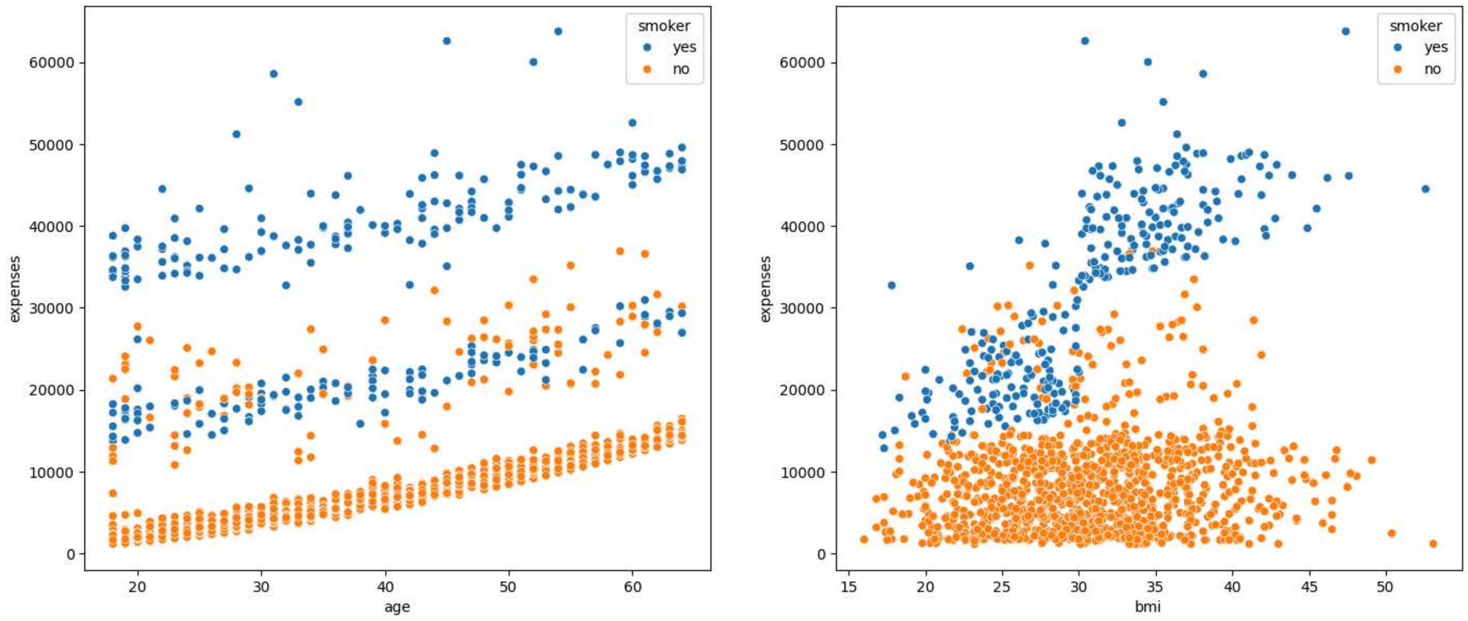
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb # Import seaborn


df = pd.read_csv('/content/insurance.csv')

features = ['age', 'bmi']

plt.subplots(figsize=(17, 7))
for i, col in enumerate(features):
    plt.subplot(1, 2, i + 1)
    sb.scatterplot(data=df, x=col,
                   y='expenses',
                   hue='smoker')
plt.show()
```

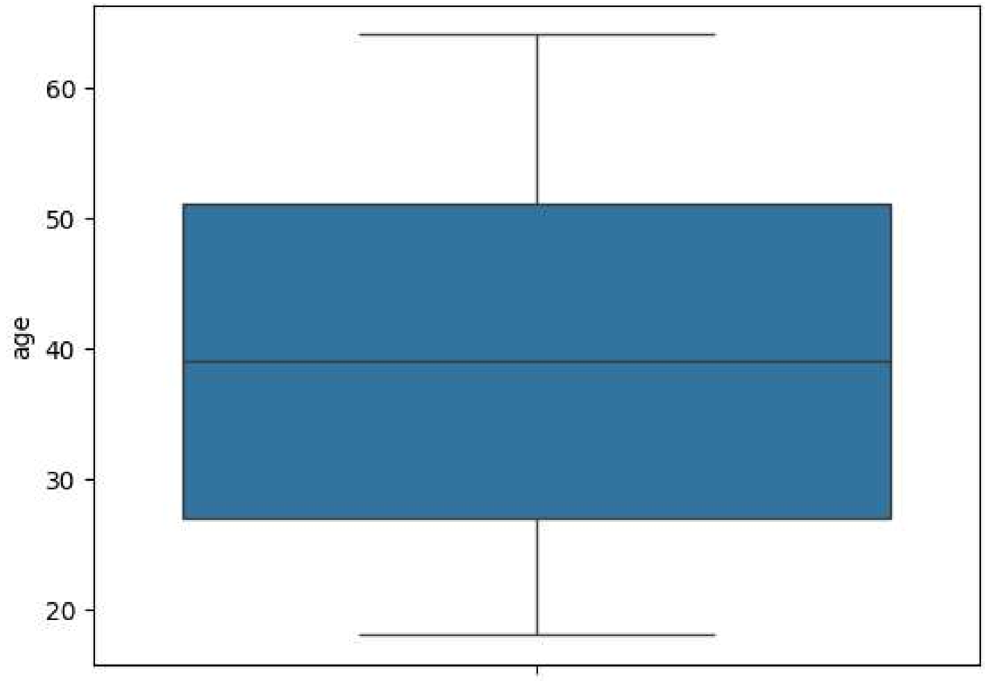
```
<ipython-input-10-a85cdb1d55bd>:12: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is depre
  plt.subplot(1, 2, i + 1)
```



Boxplot of age

```
df.drop_duplicates(inplace=True)
sb.boxplot(df['age'])
```

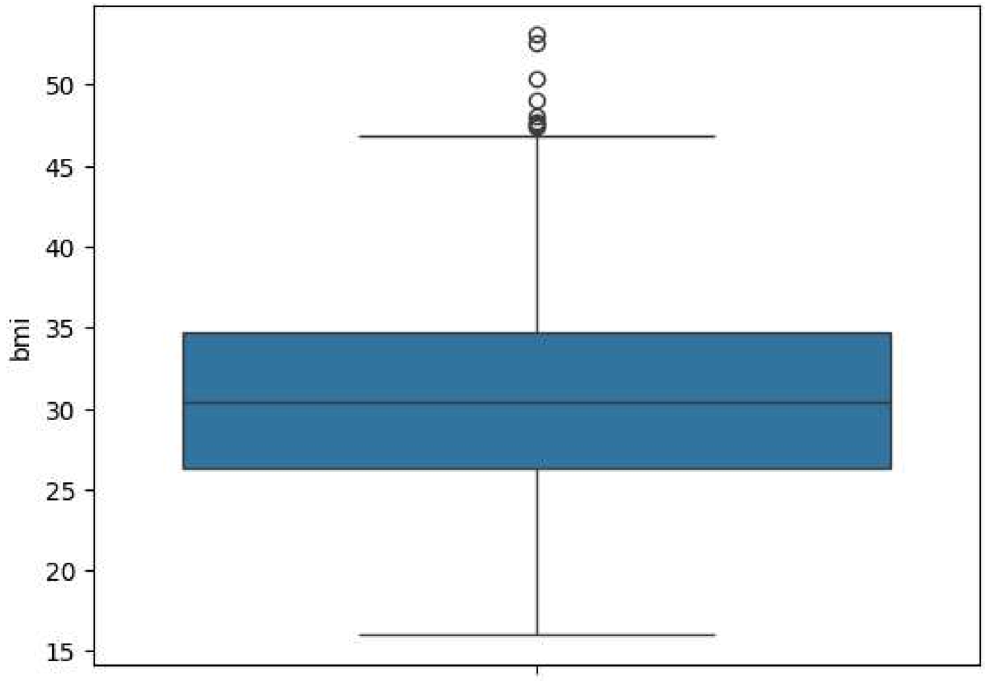
```
<Axes: ylabel='age'>
```



Box plot of bmi

```
import seaborn as sns
sns.boxplot(df['bmi'])
```

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



```
Q1=df['bmi'].quantile(0.25)
Q2=df['bmi'].quantile(0.5)
Q3=df['bmi'].quantile(0.75)
iqr=Q3-Q1
lowlim=Q1-1.5*iqr
upplim=Q3+1.5*iqr
print(lowlim)
print(upplim)
```

```
13.699999999999998
47.300000000000004
```

```
df['age'].skew()
```

```
0.054780773126998195
```

correlation matrix

```
!pip install --upgrade pandas

import pandas as pd
import matplotlib.pyplot as plt



df['sex'] = df['sex'].map({'male': 0, 'female': 1})
df['smoker'] = df['smoker'].map({'yes': 1, 'no': 0})
df['region'] = df['region'].map({'northwest': 0, 'northeast': 1, 'southeast': 2, 'southwest': 3})

# Try using to_string() to display the correlation matrix
print(df.corr().to_string())
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.25.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.4)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas)
               age  sex       bmi  children  smoker  region  expenses
age       1.000000  NaN  0.112069  0.041536     NaN     NaN  0.298308
sex            NaN  NaN       NaN       NaN     NaN     NaN       NaN
bmi       0.112069  NaN  1.000000  0.013574     NaN     NaN  0.199298
children  0.041536  NaN  0.013574  1.000000     NaN     NaN  0.067389
smoker         NaN  NaN       NaN       NaN     NaN     NaN       NaN
region         NaN  NaN       NaN       NaN     NaN     NaN       NaN
expenses  0.298308  NaN  0.199298  0.067389     NaN     NaN  1.000000
```

```
xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.2,random_state=42)
lrmodel=LinearRegression()
lrmodel.fit(xtrain,ytrain)
print(lrmodel.score(xtrain,ytrain))
print(lrmodel.score(xtest,ytest))
print(cross_val_score(lrmodel,X,Y,cv=5,).mean())
```

```
0.1096025209475574
0.13771209251047456
0.1167572302298773
```

```
from sklearn.metrics import r2_score
svrmodel=SVR()
svrmodel.fit(xtrain,ytrain)
ypredtrain1=svrmodel.predict(xtrain)
ypredtest1=svrmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain1))
print(r2_score(ytest,ypredtest1))
print(cross_val_score(svrmodel,X,Y,cv=5,).mean())
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1310: DataConversionWarning: A column-vector y was pas
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1310: DataConversionWarning: A column-vector y was pas
  y = column_or_1d(y, warn=True)
-0.1006002667676189
-0.13368071493013267
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1310: DataConversionWarning: A column-vector y was pas
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1310: DataConversionWarning: A column-vector y was pas
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1310: DataConversionWarning: A column-vector y was pas
  y = column_or_1d(y, warn=True)
-0.10361331629076327
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1310: DataConversionWarning: A column-vector y was pas
  y = column_or_1d(y, warn=True)
```

```
rfmodel=RandomForestRegressor(random_state=42)
rfmodel.fit(xtrain,ytrain)
ypredtrain2=rfmodel.predict(xtrain)
ypredtest2=rfmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain2))
print(r2_score(ytest,ypredtest2))
print(cross_val_score(rfmodel,X,Y,cv=5,).mean())
from sklearn.model_selection import GridSearchCV
estimator=RandomForestRegressor(random_state=42)
param_grid={'n_estimators':[10,40,50,98,100,120,150]}
grid=GridSearchCV(estimator,param_grid,scoring="r2",cv=5)
grid.fit(xtrain,ytrain)
print(grid.best_params_)
rfmodel=RandomForestRegressor(random_state=42,n_estimators=120)
rfmodel.fit(xtrain,ytrain)
ypredtrain2=rfmodel.predict(xtrain)
ypredtest2=rfmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain2))
print(r2_score(ytest,ypredtest2))
print(cross_val_score(rfmodel,X,Y,cv=5,).mean())
```

RandomForestRegressor:

0.9738163260247533

0.8819423353068565

0.8363637309718952

Hyperparametertuning:

{'n_estimators': 120}

0.9746383984429655

0.8822009842175969

0.8367438097052858

```
gbmodel=GradientBoostingRegressor()
gbmodel.fit(xtrain,ytrain)
ypredtrain3=gbmodel.predict(xtrain)
ypredtest3=gbmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain3))
print(r2_score(ytest,ypredtest3))
print(cross_val_score(gbmodel,X,Y,cv=5,).mean())
from sklearn.model_selection import GridSearchCV
estimator=GradientBoostingRegressor()
param_grid={'n_estimators':[10,15,19,20,21,50],'learning_rate':[0.1,0.19,0.2,0.21,0.8,1]}
grid=GridSearchCV(estimator,param_grid,scoring="r2",cv=5)
grid.fit(xtrain,ytrain)
print(grid.best_params_)
gbmodel=GradientBoostingRegressor(n_estimators=19,learning_rate=0.2)
gbmodel.fit(xtrain,ytrain)
ypredtrain3=gbmodel.predict(xtrain)
ypredtest3=gbmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain3))
print(r2_score(ytest,ypredtest3))
print(cross_val_score(gbmodel,X,Y,cv=5,).mean())
```

GradientBoostingRegressor:

0.8931345821166041

0.904261922040551

0.8549940291799407

Hyperparametertuning

{'learning_rate': 0.2, 'n_estimators': 21} 0.8682397447116927

0.9017109716082661

0.8606041910125791

```
xgmodel=XGBRegressor()
xgmodel.fit(xtrain,ytrain)
ypredtrain4=xgmodel.predict(xtrain)
ypredtest4=xgmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain4))
print(r2_score(ytest,ypredtest4))
print(cross_val_score(xgmodel,X,Y,cv=5,).mean())
from sklearn.model_selection import GridSearchCV
estimator=XGBRegressor()
param_grid={'n_estimators':[10,15,20,40,50],'max_depth':[3,4,5],'gamma':[0,0.15,0.3,0.5,1]}
grid=GridSearchCV(estimator,param_grid,scoring="r2",cv=5)
grid.fit(xtrain,ytrain)
print(grid.best_params_)
xgmodel=XGBRegressor(n_estimators=15,max_depth=3,gamma=0)
xgmodel.fit(xtrain,ytrain)
ypredtrain4=xgmodel.predict(xtrain)
ypredtest4=xgmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain4))
print(r2_score(ytest,ypredtest4))
print(cross_val_score(xgmodel,X,Y,cv=5,).mean())
```

```
0.9118184447288513
-0.2653837203979492
-0.2806396007537842
{'gamma': 0, 'max_depth': 3, 'n_estimators': 10}
0.24651789665222168
0.07248115539550781
0.053965306282043456
```

```
import pandas as pd # Import pandas

# Check if 'feature_importances_' exists
if hasattr(grid.best_estimator_, 'feature_importances_'):
    # Handle cases where X might not have the 'columns' attribute
    if hasattr(X, 'columns'):
        feats = pd.DataFrame(data=grid.best_estimator_.feature_importances_,
                             index=X.columns,
                             columns=['Importance']) # Set column name here
    else:
        feats = pd.DataFrame(data=grid.best_estimator_.feature_importances_,
                             index=range(len(grid.best_estimator_.feature_importances_)),
                             columns=['Importance']) # Set column name here
    print(feats)  # Display the feature importances
else:
    print("The 'best_estimator_' does not have the 'feature_importances_' attribute.")
```

```
   Importance
0    0.525635
1    0.261984
2    0.212381
```

```
df.drop(df[['sex','region']],axis=1,inplace=True)
Xf=df.drop(df[['charges']],axis=1)
X=df.drop(df[['charges']],axis=1)
xtrain,xtest,ytrain,ytest=train_test_split(Xf,Y,test_size=0.2,random_state=42)
finalmodel=XGBRegressor(n_estimators=15,max_depth=3,gamma=0)
finalmodel.fit(xtrain,ytrain)
ypredtrain4=finalmodel.predict(xtrain)
ypredtest4=finalmodel.predict(xtest)
print(r2_score(ytrain,ypredtrain4))
print(r2_score(ytest,ypredtest4))
print(cross_val_score(finalmodel,X,Y,cv=5,).mean())
```

Final Model:

Train accuracy : 0.870691899927822

Test accuracy : 0.904151903449132

CV Score : 0.8600710679082143