

INTRODUCTION

Introduction: The majority of the countries finalize health insurance costs based on many factors such as age, number of people in families, etc. What should be the actual health insurance price for an individual or a family is an issue for many companies. Many factors have been shown association with higher insurance price such as age, smoking status, pre-existing disease etc,. Researchers are still exploring more features to help insurance company to predict insurance price with a much higher accuracy.


AIM

Aim of the current project is in depth exploration to identify important features relevant to health insurance price as well as best ML model to help company increase the accuracy of health insurance prediction.

```
1 # Importing libraries
2 import pandas as pd
3 import numpy as np
```


```
1 # Importing libraries
2 import pandas as pd
3 import numpy as np
4
5 # Use pd.read_excel to read excel files
6 df = pd.read_excel("/content/Health_insurance_cost.xlsx")
```

```
1 df.head()
```




	age	gender	BMI	Children	smoking_status	location	health_insurance_price
0	19.0	female	NaN	0	yes	southwest	16884.92400
1	18.0	male	33.770	1	no	southeast	1725.55230
2	28.0	male	33.000	3	no	southeast	4449.46200
3	33.0	male	22.705	0	no	northwest	21984.47061
4	32.0	male	28.880	0	no	northwest	3866.85520

```
1 # No of rows and columns.
2 df.shape
```



```
(1338, 7)
```


```
1 # Datatype and variables
2 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                   1310 non-null  float64
1   gender                 1338 non-null  object
2   BMI                   1315 non-null  float64
3   Children              1338 non-null  int64
4   smoking_status        1338 non-null  object
5   location              1338 non-null  object
6   health_insurance_price 1336 non-null  float64
dtypes: float64(3), int64(1), object(3)
memory usage: 73.3+ KB
```

```
1 # statistics of the project
2 df.describe(include="all")
```



	age	gender	BMI	Children	smoking_status	location	health_insurance_price
count	1310.000000	1338	1315.000000	1338.000000	1338	1338	1336.000000
unique	NaN	2	NaN	NaN	2	4	NaN
top	NaN	male	NaN	NaN	no	southeast	NaN
freq	NaN	676	NaN	NaN	1064	364	NaN
mean	39.166412	NaN	30.638217	1.094918	NaN	NaN	13268.527719
std	14.055378	NaN	6.110302	1.205493	NaN	NaN	12112.797724
min	18.000000	NaN	15.960000	0.000000	NaN	NaN	1121.873900
25%	26.000000	NaN	26.210000	0.000000	NaN	NaN	4744.325050
50%	39.000000	NaN	30.305000	1.000000	NaN	NaN	9382.033000
75%	51.000000	NaN	34.580000	2.000000	NaN	NaN	16604.302645
max	64.000000	NaN	53.130000	5.000000	NaN	NaN	63770.428010

```
1 # to check the number of missing values in each of the variables
2 df.isnull().sum()
```



	0
age	28
gender	0
BMI	23
Children	0
smoking_status	0
location	0
health_insurance_price	2

dtype: int64

```
1 # i am using data1 just to calculate percentage of missing values in each of the variables
2 data1 = df.isnull().sum()/len(df)*100
3 data1
```



	0
age	2.092676
gender	0.000000
BMI	1.718984
Children	0.000000
smoking_status	0.000000
location	0.000000
health_insurance_price	0.149477

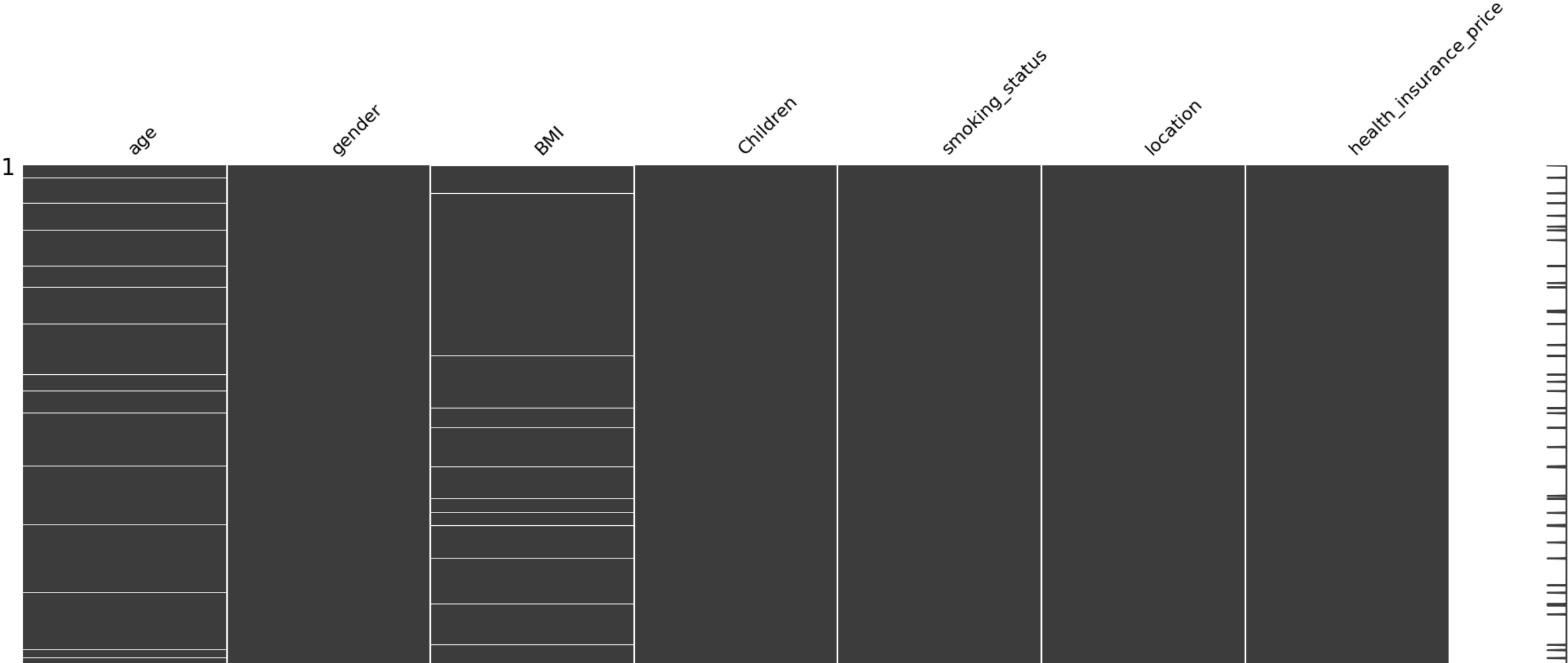
dtype: float64

```
1 Start coding or generate with AI.
```

```
1 # library to examine missing values
2 import missingno as msno
```

```
1 # creating matrix plot to see the pattern in missing values
2 msno.matrix(df)
```

↔ <Axes: >



```
1 # to check if there is any corelation between mising values
2 msno.heatmap(df)
```

↔ <Axes: >

age

BMI

health_insurance_price



From the above figure we can observe that there is no corelation between missing values.

```
1 #find the unique values of the secified column.  
2 df.smoking_status.unique()
```

↔ array(['yes', 'no'], dtype=object)

```
1 df.location.unique()
```

```
array(['southwest', 'southeast', 'northwest', 'northeast'], dtype=object)
```

```
1 df.gender.unique()
```

```
array(['female', 'male'], dtype=object)
```

```
1 df.head(5)
```

	age	gender	BMI	Children	smoking_status	location	health_insurance_price
0	19.0	female	NaN	0	yes	southwest	16884.92400
1	18.0	male	33.770	1	no	southeast	1725.55230
2	28.0	male	33.000	3	no	southeast	4449.46200
3	33.0	male	22.705	0	no	northwest	21984.47061
4	32.0	male	28.880	0	no	northwest	3866.85520

✓ Apply encoding before imputing missing values.


```
1 # Applying one hot encoding
2 df_en= pd.get_dummies(df, columns = ['gender', 'smoking_status',"location"])
```

```
1 df_en.head(5) # after encoding this is the look of the dataset
```


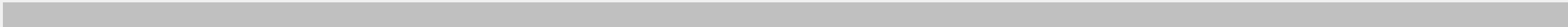

	age	BMI	Children	health_insurance_price	gender_female	gender_male	smoking_status_no	smoking_status_yes	location_northeast	location_northwest	location_southeast	location_southwest
0	19.0	NaN	0	16884.92400	True	False	False	True	False	False	False	True
1	18.0	33.770	1	1725.55230	False	True	True	False	False	False	True	False
2	28.0	33.000	3	4449.46200	False	True	True	False	False	False	True	False
3	33.0	22.705	0	21984.47061	False	True	True	False	False	True	False	False
4	32.0	28.880	0	3866.85520	False	True	True	False	False	True	False	False

```
1 # Using dataframe df_en: convert all column in 0 and 1
2
3 # Select all boolean columns
4 bool_cols = df_en.select_dtypes(include=['bool']).columns
5
6 # Convert boolean columns to 0 and 1
7 df_en[bool_cols] = df_en[bool_cols].astype(int)
8
```

```
1 df_en.head(5)
```



	age	BMI	Children	health_insurance_price	gender_female	gender_male	smoking_status_no	smoking_status_yes	location_northeast	location_northwest	location_southeast	location_sou
0	19.0	NaN	0	16884.92400	1	0	0	1	0	0	0	
1	18.0	33.770	1	1725.55230	0	1	1	0	0	0	1	
2	28.0	33.000	3	4449.46200	0	1	1	0	0	0	1	
3	33.0	22.705	0	21984.47061	0	1	1	0	0	1	0	
4	32.0	28.880	0	3866.85520	0	1	1	0	0	1	0	



✓ Dealing with missing values

```
1 # Mean imputation
2 from sklearn.impute import SimpleImputer
3 df_mean = df_en.copy(deep=True) # creating a copy for further evaluation
4 mean_imputer = SimpleImputer(strategy='mean')
5 df_mean.iloc[:, :] = mean_imputer.fit_transform(df_mean)
```

```
1 # Median imputation
2 df_median = df_en.copy(deep=True) # creating a copy for further evaluation
3 median_imputer = SimpleImputer(strategy='median')
4 df_median.iloc[:, :] = median_imputer.fit_transform(df_median)
```

```
1 # Mode imputation
2 df_mode = df_en.copy(deep=True) # creating a copy for further evaluation
3 mode_imputer = SimpleImputer(strategy='most_frequent')
4 df_mode.iloc[:, :] = mode_imputer.fit_transform(df_mode)
```


```
1 # Constant imputation
2 df_constant = df_en.copy(deep=True) # creating a copy for further evaluation
3 constant_imputer = SimpleImputer(strategy='constant', fill_value = 0)
4 df_constant.iloc[:, :] = constant_imputer.fit_transform(df_constant)
```

Fancyimput

fancyimpute is a library for missing data imputation algorithms. Fancyimpute use machine learning algorithm to impute missing values. Fancyimpute uses all the column to impute the missing values. There are two ways missing data can be imputed using Fancyimpute

KNN or K-Nearest Neighbor MICE or Multiple Imputation by Chained Equation

```
1 pip install fancyimpute
```



Collecting fancyimpute

Downloading fancyimpute-0.7.0.tar.gz (25 kB)

Preparing metadata (setup.py) ... done

Collecting knnimpute>=0.1.0 (from fancyimpute)

```

Downloading knnimpute-0.1.0.tar.gz (8.3 kB)
Preparing metadata (setup.py) ... done
Requirement already satisfied: scikit-learn>=0.24.2 in /usr/local/lib/python3.10/dist-packages (from fancyimpute) (1.3.2)
Requirement already satisfied: cvxpy in /usr/local/lib/python3.10/dist-packages (from fancyimpute) (1.5.3)
Requirement already satisfied: cvxopt in /usr/local/lib/python3.10/dist-packages (from fancyimpute) (1.3.2)
Requirement already satisfied: pytest in /usr/local/lib/python3.10/dist-packages (from fancyimpute) (7.4.4)
Collecting nose (from fancyimpute)
  Downloading nose-1.3.7-py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from knnimpute>=0.1.0->fancyimpute) (1.16.0)
Requirement already satisfied: numpy>=1.10 in /usr/local/lib/python3.10/dist-packages (from knnimpute>=0.1.0->fancyimpute) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.24.2->fancyimpute) (1.13.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.24.2->fancyimpute) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.24.2->fancyimpute) (3.5.0)
Requirement already satisfied: osqp>=0.6.2 in /usr/local/lib/python3.10/dist-packages (from cvxpy->fancyimpute) (0.6.7.post0)
Requirement already satisfied: ecos>=2 in /usr/local/lib/python3.10/dist-packages (from cvxpy->fancyimpute) (2.0.14)
Requirement already satisfied: clarabel>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from cvxpy->fancyimpute) (0.9.0)
Requirement already satisfied: scs>=3.2.4.post1 in /usr/local/lib/python3.10/dist-packages (from cvxpy->fancyimpute) (3.2.7)
Requirement already satisfied: iniconfig in /usr/local/lib/python3.10/dist-packages (from pytest->fancyimpute) (2.0.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from pytest->fancyimpute) (24.1)
Requirement already satisfied: pluggy<2.0,>=0.12 in /usr/local/lib/python3.10/dist-packages (from pytest->fancyimpute) (1.5.0)
Requirement already satisfied: exceptiongroup>=1.0.0rc8 in /usr/local/lib/python3.10/dist-packages (from pytest->fancyimpute) (1.2.2)
Requirement already satisfied: tomli>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from pytest->fancyimpute) (2.0.1)
Requirement already satisfied: qdldl in /usr/local/lib/python3.10/dist-packages (from osqp>=0.6.2->cvxpy->fancyimpute) (0.1.7.post4)
Downloading nose-1.3.7-py3-none-any.whl (154 kB)
----- 154.7/154.7 kB 4.8 MB/s eta 0:00:00
Building wheels for collected packages: fancyimpute, knnimpute
  Building wheel for fancyimpute (setup.py) ... done
  Created wheel for fancyimpute: filename=fancyimpute-0.7.0-py3-none-any.whl size=29880 sha256=404ce333e6db665467a865e8075eed71d473f6ac339d32d4cfcd8b40bdb24d9
  Stored in directory: /root/.cache/pip/wheels/7b/0c/d3/ee82d1fbdcc0858d96434af108608d01703505d453720c84ed
  Building wheel for knnimpute (setup.py) ... done
  Created wheel for knnimpute: filename=knnimpute-0.1.0-py3-none-any.whl size=11330 sha256=794dc58280331c3bd42f882ac8bf6e0b80f159c859e7b0ce3c7327953b3dc8cf
  Stored in directory: /root/.cache/pip/wheels/46/06/a5/45a724630562413c374e29c08732411d496092408b3a7bf754
Successfully built fancyimpute knnimpute
Installing collected packages: nose, knnimpute, fancyimpute
Successfully installed fancyimpute-0.7.0 knnimpute-0.1.0 nose-1.3.7

```

```

1 # KNN imputation for missing values
2 from fancyimpute import KNN
3 knn_imputer = KNN()
4 df_knn = df_en.copy(deep=True)
5 df_knn.iloc[:, :] = knn_imputer.fit_transform(df_knn)

```

```

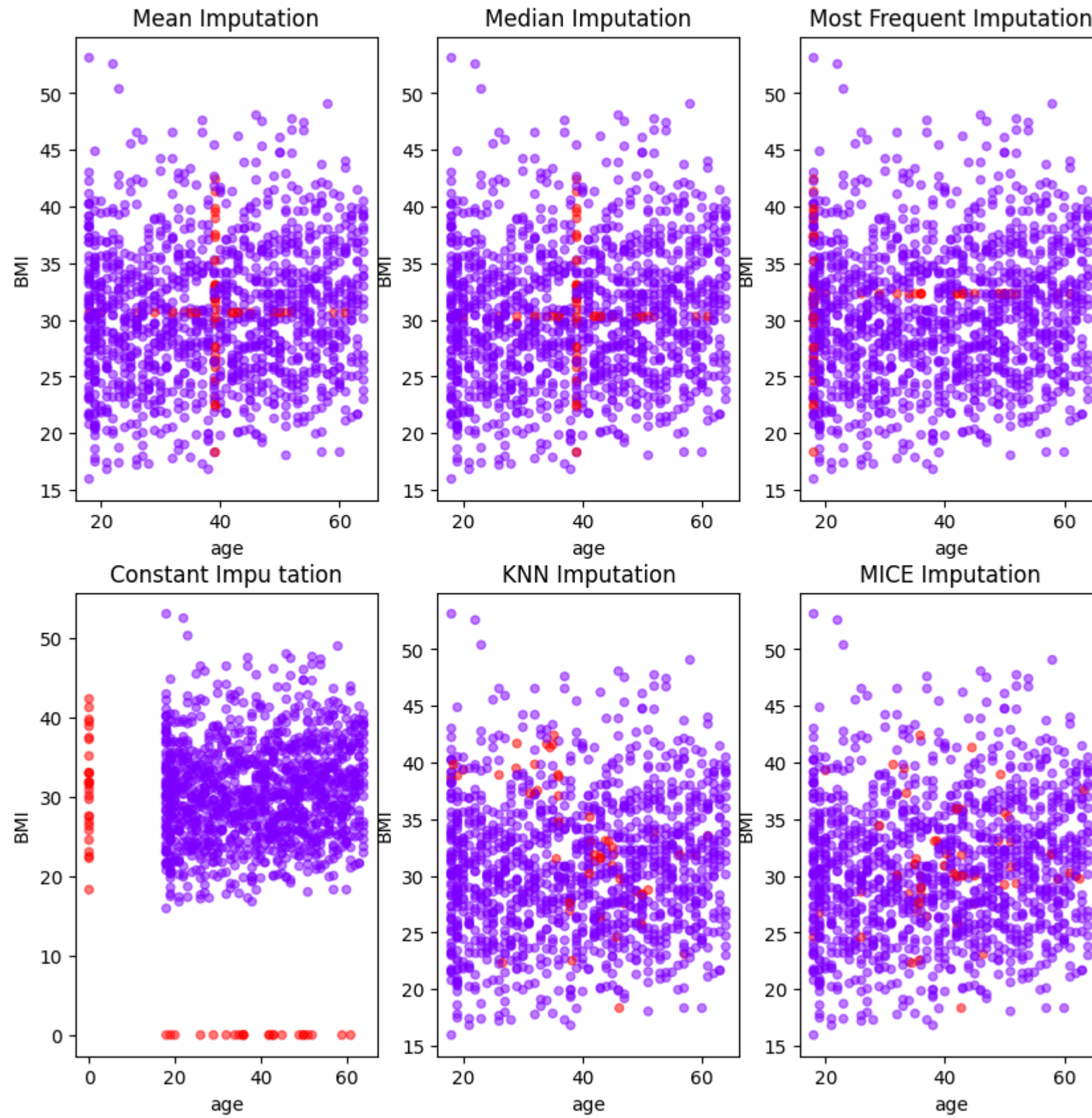
↔ Imputing row 1/1338 with 1 missing, elapsed time: 0.297
Imputing row 101/1338 with 0 missing, elapsed time: 0.298
Imputing row 201/1338 with 0 missing, elapsed time: 0.300
Imputing row 301/1338 with 0 missing, elapsed time: 0.301
Imputing row 401/1338 with 0 missing, elapsed time: 0.302
Imputing row 501/1338 with 0 missing, elapsed time: 0.302
Imputing row 601/1338 with 0 missing, elapsed time: 0.303
Imputing row 701/1338 with 0 missing, elapsed time: 0.304
Imputing row 801/1338 with 0 missing, elapsed time: 0.306
Imputing row 901/1338 with 0 missing, elapsed time: 0.307
Imputing row 1001/1338 with 0 missing, elapsed time: 0.308
Imputing row 1101/1338 with 0 missing, elapsed time: 0.309
Imputing row 1201/1338 with 0 missing, elapsed time: 0.309
Imputing row 1301/1338 with 0 missing, elapsed time: 0.311

```



```
1 #Mice imputation for missing values
2 from fancyimpute import IterativeImputer
3 MICE_imputer = IterativeImputer()
4 df_MICE = df_en.copy(deep=True)
5 df_MICE.iloc[:, :] = MICE_imputer.fit_transform(df_MICE)

1 # as we have learnt in EDA sessions plotting each of the imputation results and picking the best among all
2 from matplotlib import pyplot as plt
3 fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(10, 10))
4 nullity = df_en['age'].isnull() + df_en['BMI'].isnull() # we want to specify the color values for the data points based on the nullity values
5 imputations = {'Mean Imputation': df_mean,
6               'Median Imputation': df_median,
7               'Most Frequent Imputation': df_mode,
8               'Constant Imputation': df_constant,
9               'KNN Imputation': df_knn,
10              'MICE Imputation': df_MICE}
11 #This loop iterates through the subplot axes and the keys of the imputations dictionary simultaneously using the zip() function.
12 #axes.flatten() is used to flatten the 2D array of subplot axes into a 1D array.
13 for ax, df_key in zip(axes.flatten(), imputations):
14
15     #For each iteration of the loop, it accesses the DataFrame associated with the current df_key from the imputations dictionary.
16     imputations[df_key].plot(x='age', y='BMI', kind='scatter',
17                             alpha=0.5, c=nullity, cmap='rainbow', ax=ax,
18                             colorbar=False, title=df_key)
19     #alpha=0.5 controls the transparency of the data points in the scatter plot.
20     #c=nullity specifies the color values for the data points based on the nullity values. This could be a measure of missingness in the dataset.
21     #cmap='rainbow' specifies the colormap to be used for coloring the data points.
22     #ax=ax assigns the current subplot axis to which the scatter plot will be added.
23     #colorbar=False suppresses the display of the colorbar.
24
```



```
1 #checking the statistics of the project after performing knn imputation
2 df_knn.describe(include="all")
```



	age	BMI	Children	health_insurance_price	gender_female	gender_male	smoking_status_no	smoking_status_yes	location_northeast	location_northwest	location_sout
count	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.0
mean	39.190753	30.711828	1.094918	13265.955854	0.494768	0.505232	0.795217	0.204783	0.242152	0.242900	0.2
std	13.960328	6.111971	1.205493	12105.320304	0.500160	0.500160	0.403694	0.403694	0.428546	0.428995	0.4
min	18.000000	15.960000	0.000000	1121.873900	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	27.000000	26.315000	0.000000	4746.521225	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.0
50%	39.000000	30.495000	1.000000	9382.033000	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.0
75%	51.000000	34.700000	2.000000	16584.318157	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	1.0
max	64.000000	53.130000	5.000000	63770.428010	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0



```
1 #checking the statistics after performing Mice imputation
2 df_MICE.describe()
```



	age	BMI	Children	health_insurance_price	gender_female	gender_male	smoking_status_no	smoking_status_yes	location_northeast	location_northwest	location_sout
count	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.000000	1338.0
mean	39.221552	30.642142	1.094918	13282.167197	0.494768	0.505232	0.795217	0.204783	0.242152	0.242900	0.2
std	13.971996	6.073035	1.205493	12117.188785	0.500160	0.500160	0.403694	0.403694	0.428546	0.428995	0.4
min	18.000000	15.960000	0.000000	1121.873900	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	27.000000	26.315000	0.000000	4746.521225	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.0
50%	39.000000	30.300000	1.000000	9388.753650	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.0
75%	51.000000	34.496250	2.000000	16639.912515	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	1.0
max	64.000000	53.130000	5.000000	63770.428010	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0



Knn and Mice seems to have performed well. you can choose each of them and observed the metircs of each model. I am proceeding with KNN here.

```
1 df_knn.head(5) #after choosing KNN imputed dataset, again checking the top 5 rows
```

	age	BMI	Children	health_insurance_price	gender_female	gender_male	smoking_status_no	smoking_status_yes	location_northeast	location_northwest	location_southeast	location_s
0	19.0	38.84134	0	16884.92400	1	0	0	1	0	0	0	
1	18.0	33.77000	1	1725.55230	0	1	1	0	0	0	1	
2	28.0	33.00000	3	4449.46200	0	1	1	0	0	0	1	
3	33.0	22.70500	0	21984.47061	0	1	1	0	0	1	0	
4	32.0	28.88000	0	3866.85520	0	1	1	0	0	1	0	

data visualization

Univariate analysis

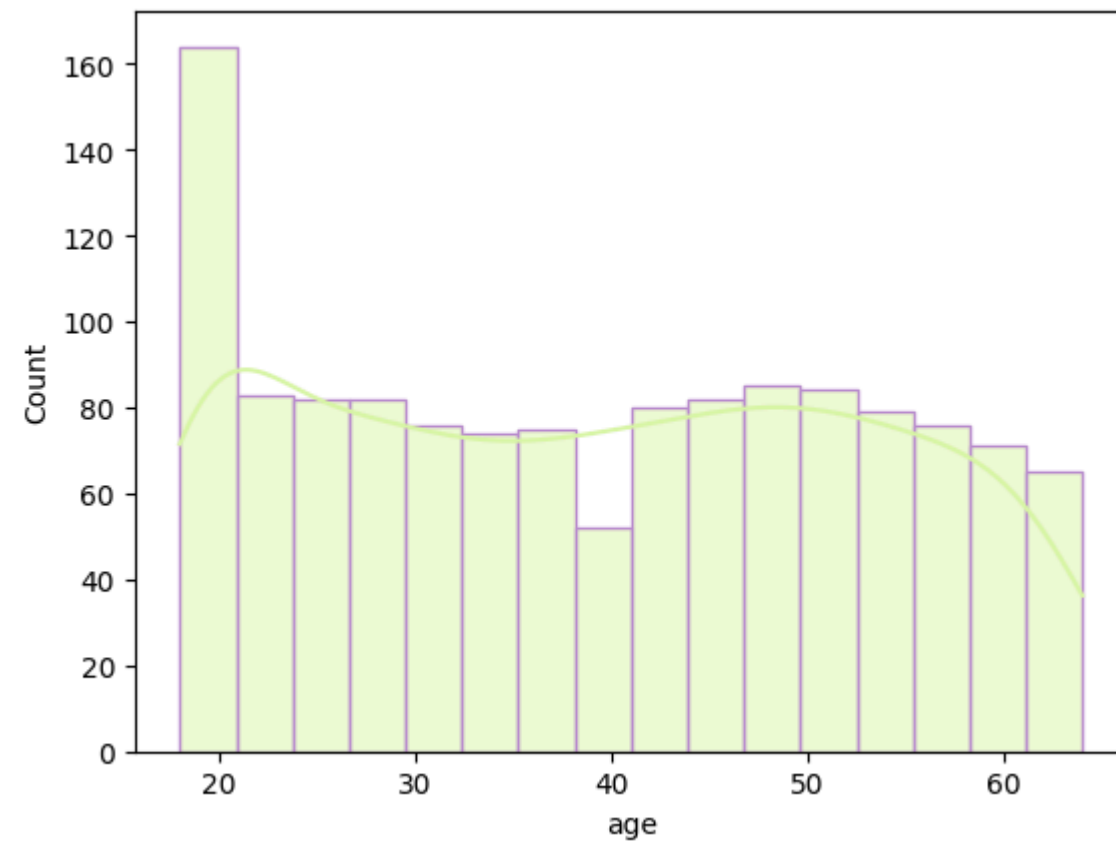
Here the aim is to check the distribution of the data. If it is normally distributed or skewed for numerical value. If it is a categorical value then we have to check if categories are balanced on unblanced.

```
1 # Please note I am not using knn imputed dataset for analysis instead raw data.
2 df.head(5)
3
```

	age	gender	BMI	Children	smoking_status	location	health_insurance_price
0	19.0	female	NaN	0	yes	southwest	16884.92400
1	18.0	male	33.770	1	no	southeast	1725.55230
2	28.0	male	33.000	3	no	southeast	4449.46200
3	33.0	male	22.705	0	no	northwest	21984.47061
4	32.0	male	28.880	0	no	northwest	3866.85520

```
1 # importing data visualisation package
2 #before we plot histogram there are a few points to consider
3 # 1. Bins should be of same size
4 # 2. include all of the data
5 # 3. use whole number
6 # 4. choose to 5 to 20 bins
7 # min=18 max=64
8 # binwidth= range/16 I am choosing 16 as number of bins here and it gives value almost equal to 3
9 # from min value i.e 18 to max value 64 you can create bins of width 3
10 import seaborn as sns
11 sns.histplot(data = df,x="age",bins=16,color="#DAF7A6",edgecolor='#BB8FCE', kde = True)
```


<Axes: xlabel='age', ylabel='Count'>



Looks like age of most of the customers fall in the range of 18-21 and there are a lesser number of customers in the range of 39-41 comparatively.

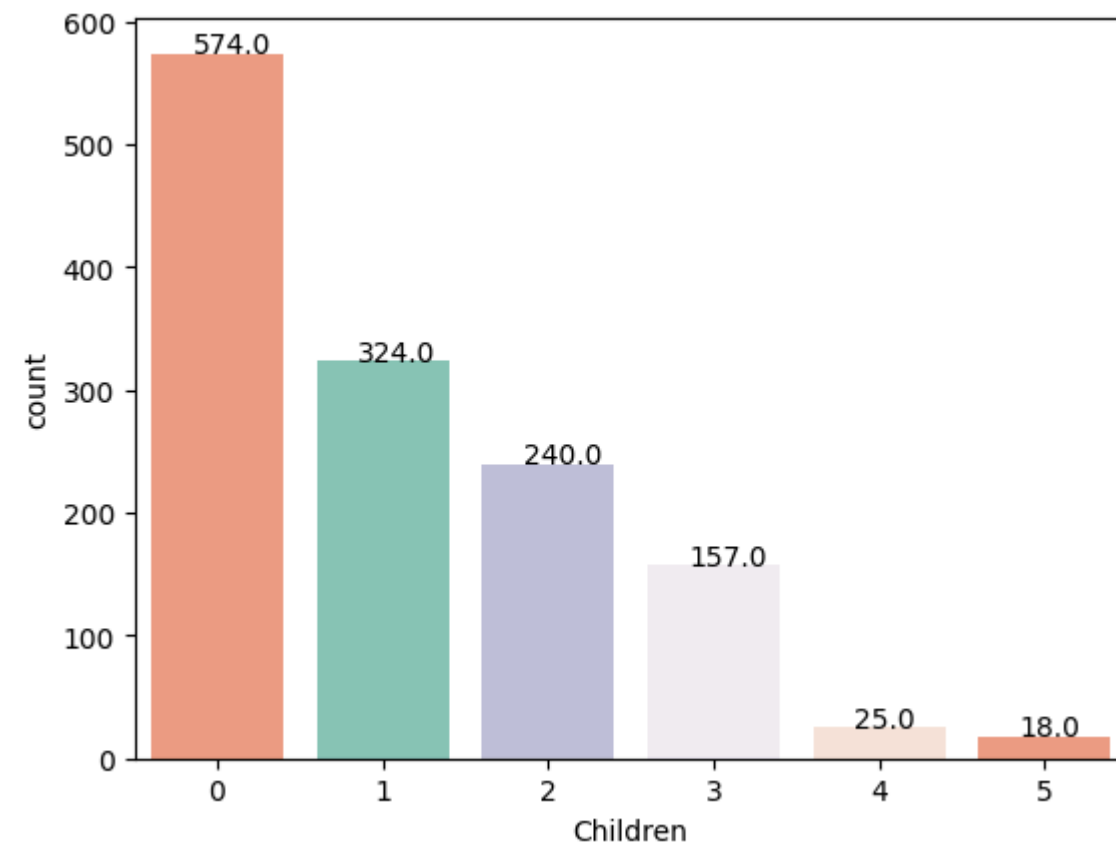
Note: width of each bin is 3. starting from 18 which is a minimum value

```
1 ax=sns.countplot(x=df.Children, palette=["#fc9272","#7fcdbb","#bcbddc","#efedf5","#fee0d2"])
2 for p in ax.patches:
3     ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+0.01))
4
5 plt.show()
```

 <ipython-input-33-67e1daed57ad>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.


```
ax=sns.countplot(x=df.Children, palette=["#fc9272", "#7fcdbb", "#bcbddc", "#efedf5", "#fee0d2"])
<ipython-input-33-67e1daed57ad>:1: UserWarning:
The palette list has fewer values (5) than needed (6) and will cycle, which may produce an uninterpretable plot.
ax=sns.countplot(x=df.Children, palette=["#fc9272", "#7fcdbb", "#bcbddc", "#efedf5", "#fee0d2"])
```



Seemingly, majority of the customers in the project have no children. That said, customers with 1 children are more followed by 2 and so on.

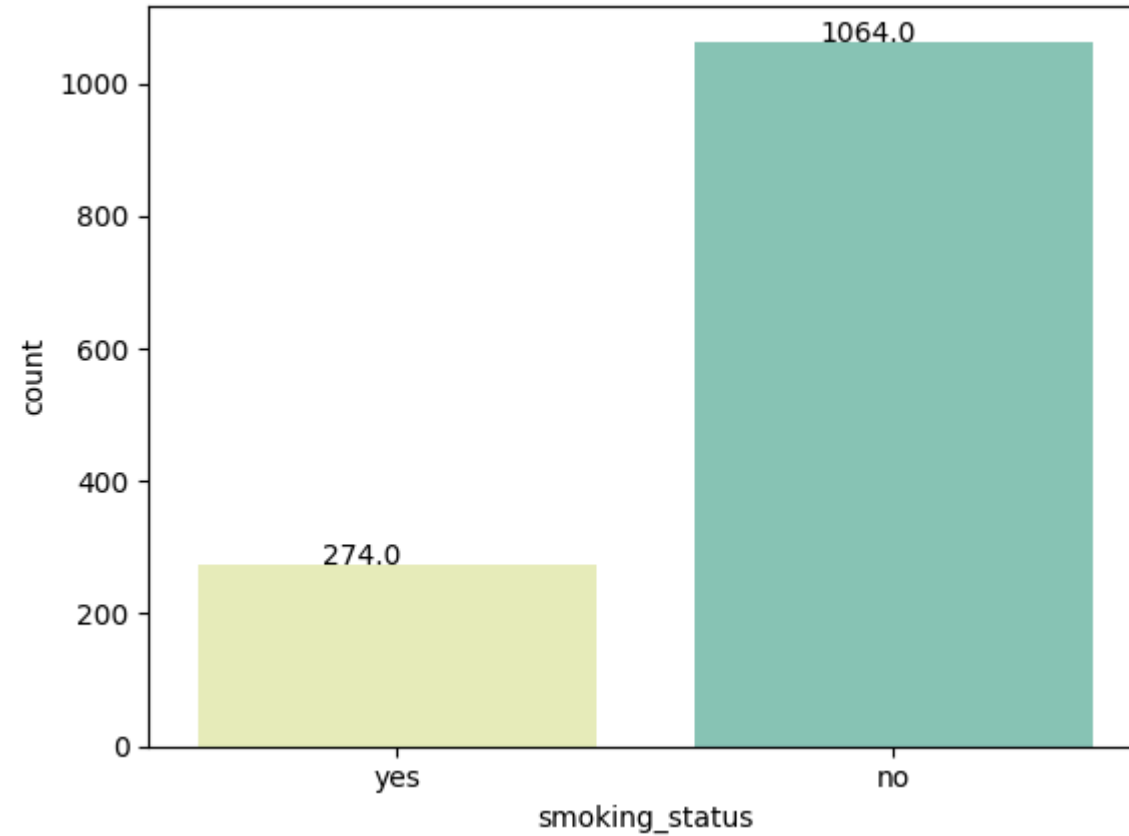
Moreover, There are very few customers with children 4 and 5.

```
1 ax =sns.countplot(x=df.smoking_status, palette=["#edf8b1", "#7fcdbb"])
2 for p in ax.patches:
3     ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+0.01))
4
5 plt.show()
```

 <ipython-input-34-cb0b7f774a67>:1: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
ax = sns.countplot(x=df.smoking_status, palette=["#edf8b1", "#7fcdbb"])
```



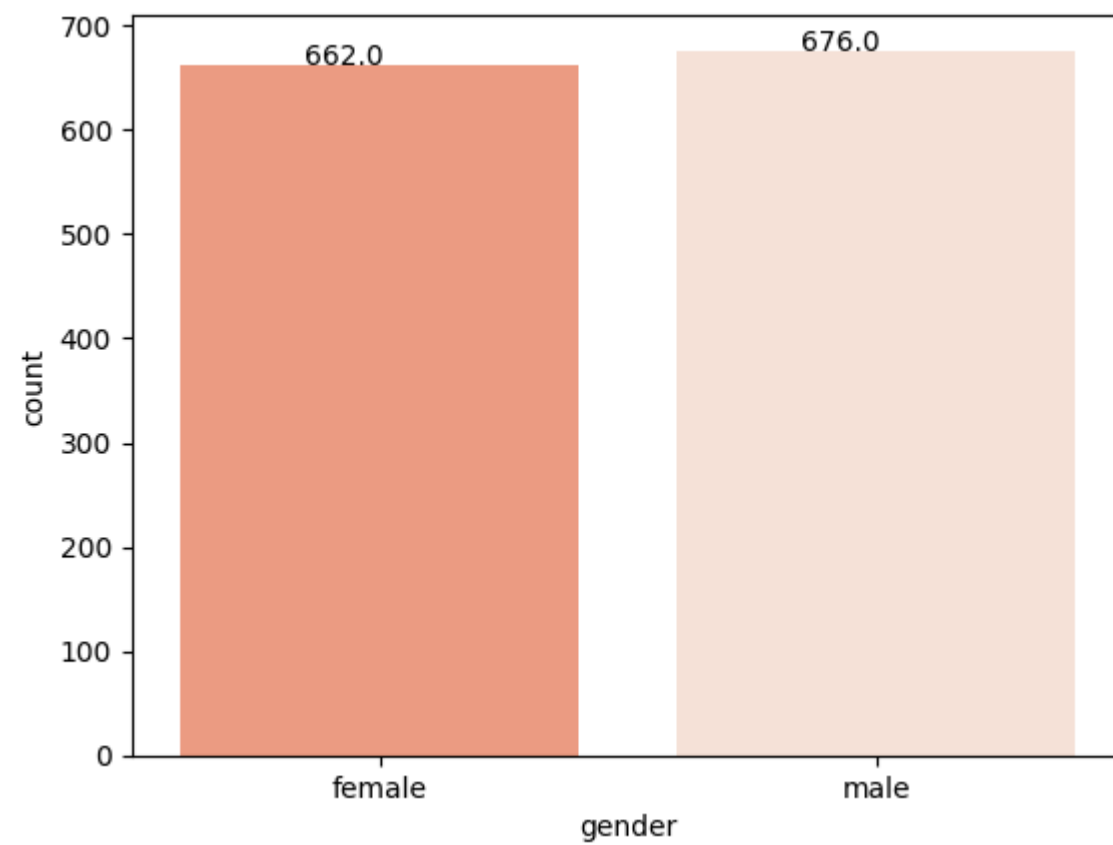
In the dataset, vast majority of customers are non smokers.

```
1 ax = sns.countplot(x=df.gender, palette=["#fc9272", "#fee0d2"])
2 for p in ax.patches:
3     ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+0.01))
4
5 plt.show()
```

 <ipython-input-35-c80edbf4c753>:1: FutureWarning:

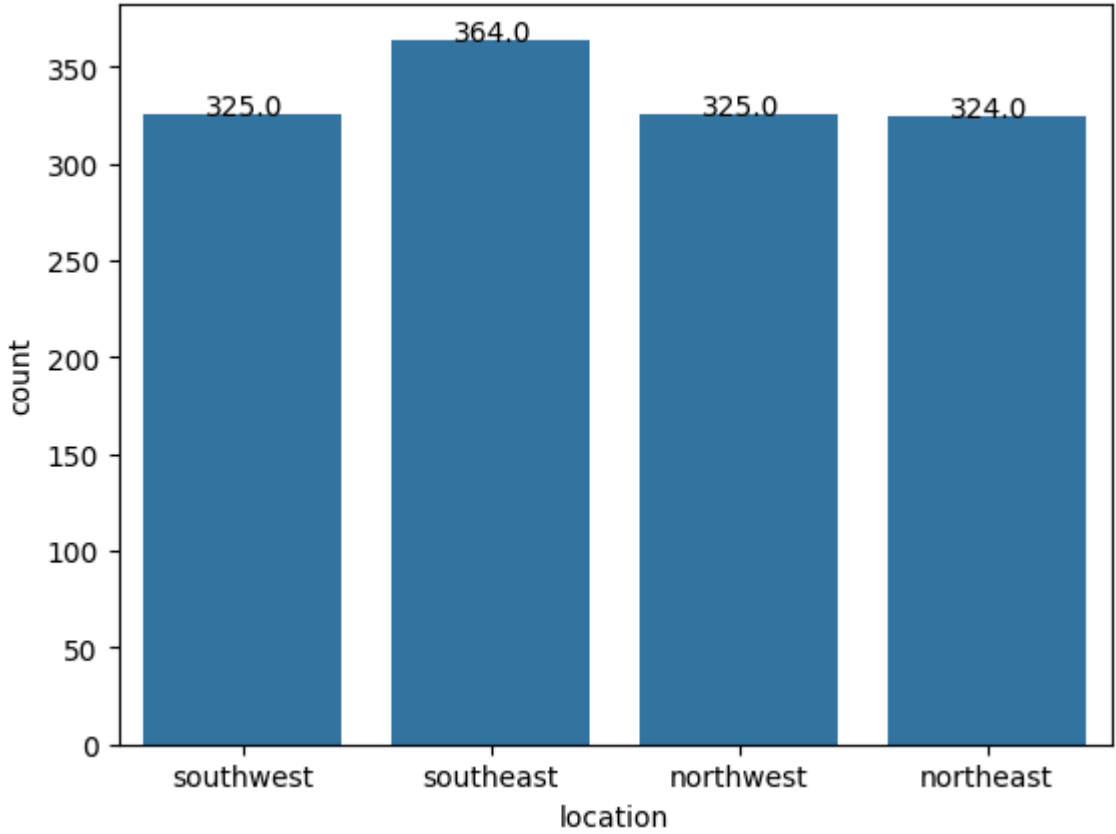
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
ax = sns.countplot(x=df.gender, palette=["#fc9272", "#fee0d2"])
```



The proportion of male and female looks almost equal

```
1 ax = sns.countplot(x=df.location)
2 for p in ax.patches:
3     ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+0.01))
4
5 plt.show()
```

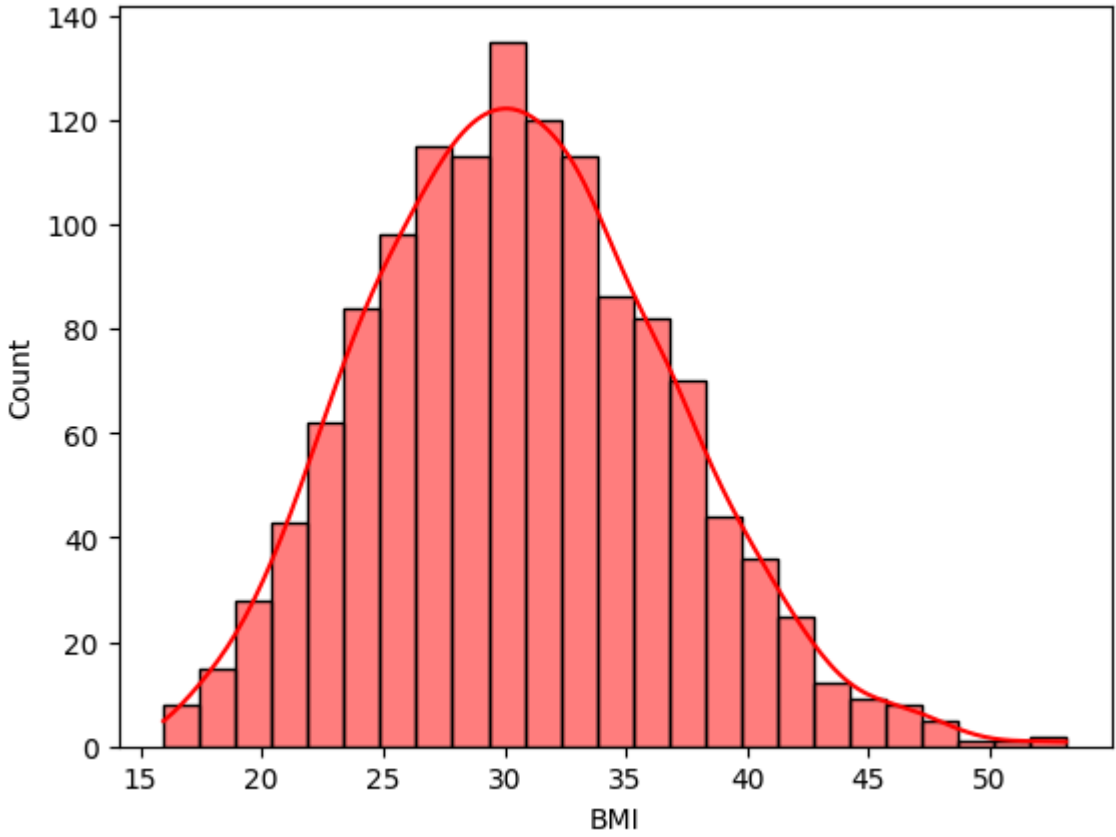



As far as location is concerned, southeast people are the highest whereas customers from other locations are almost equal in number.

```
1 sns.histplot(data = df['BMI'],color='red', kde = True) # checking the distribution of numerical variable
```



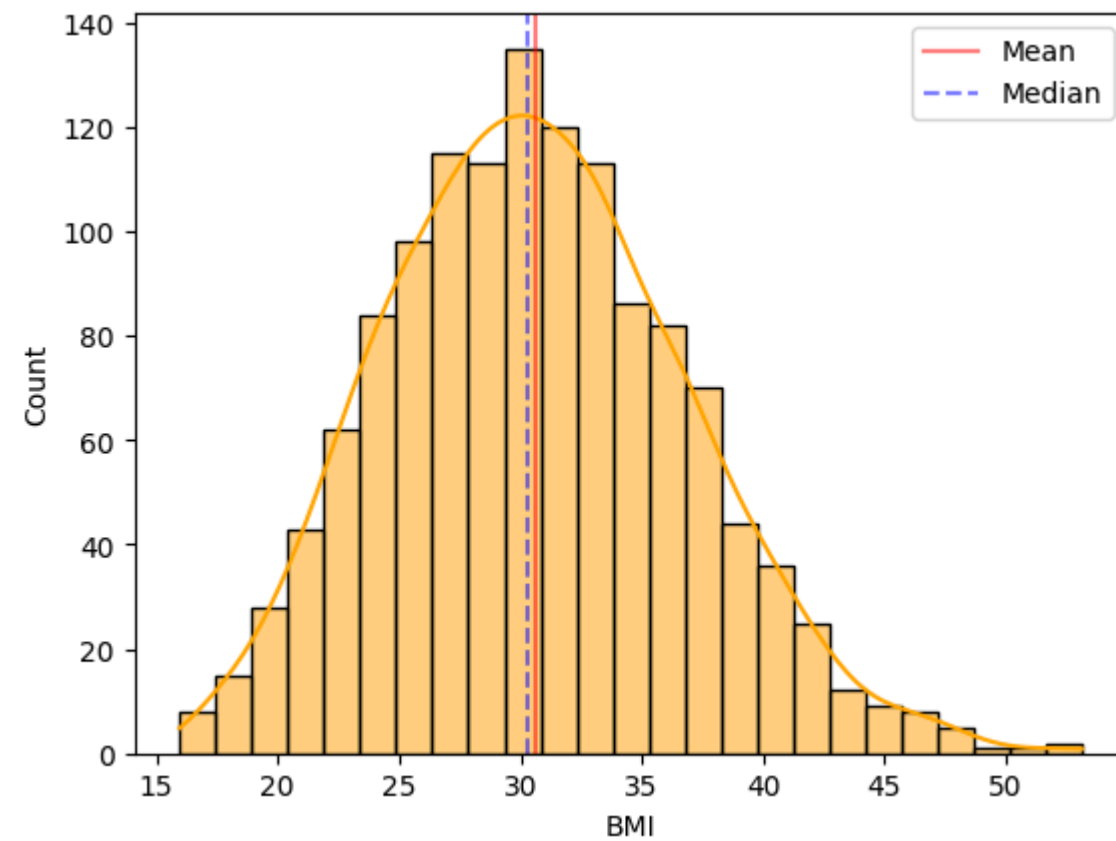
<Axes: xlabel='BMI', ylabel='Count'>



From the histogram it looks like BMI of customers follows a normal distribution. However, let's verify


```
1 sns.histplot(data = df['BMI'],color='orange', kde = True)
2 plt.axvline(x=df.BMI.mean(),color='red',alpha=0.5,label='Mean') # function used to add a vertical line
3 plt.axvline(x=df.BMI.median(),c='blue',ls='--',alpha=0.5,label='Median')
4 plt.legend()
```

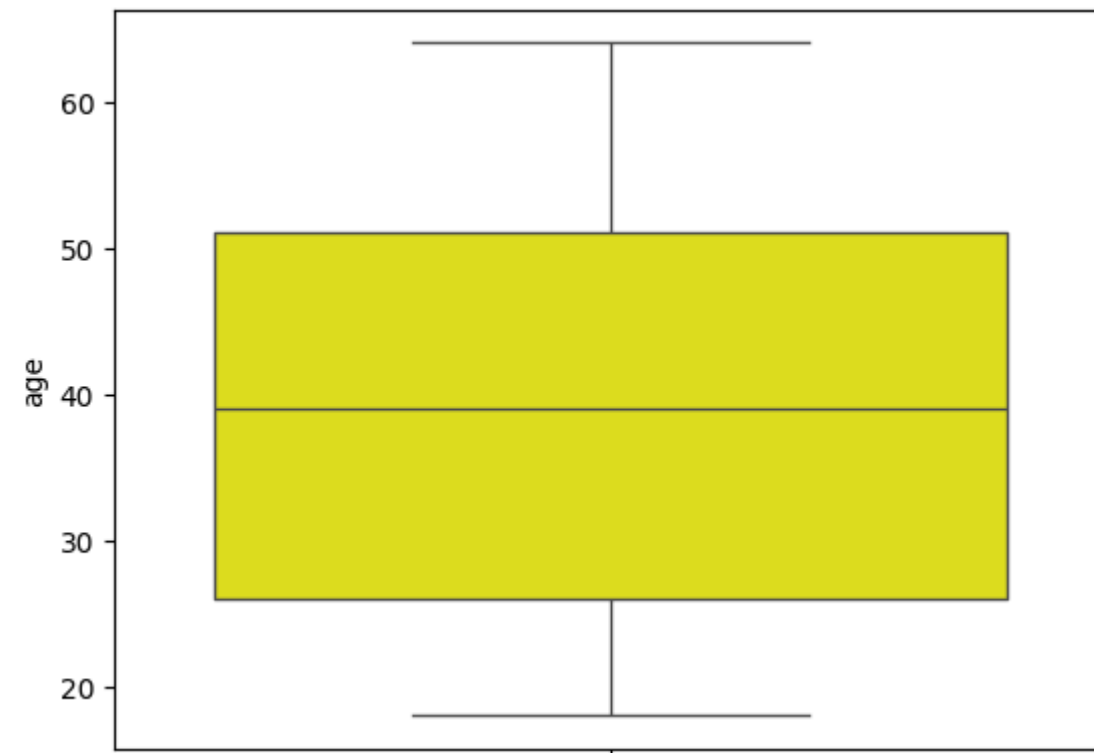
 <matplotlib.legend.Legend at 0x7a0925b462c0>



Mean, Median and mode should be equal for Normal Distribution (ND) but it is not the same here. Having said that most of the BMI values fall between 20 to 40


```
1 #dealing with outliers
2 sns.boxplot(df["age"],color="yellow")
3
```

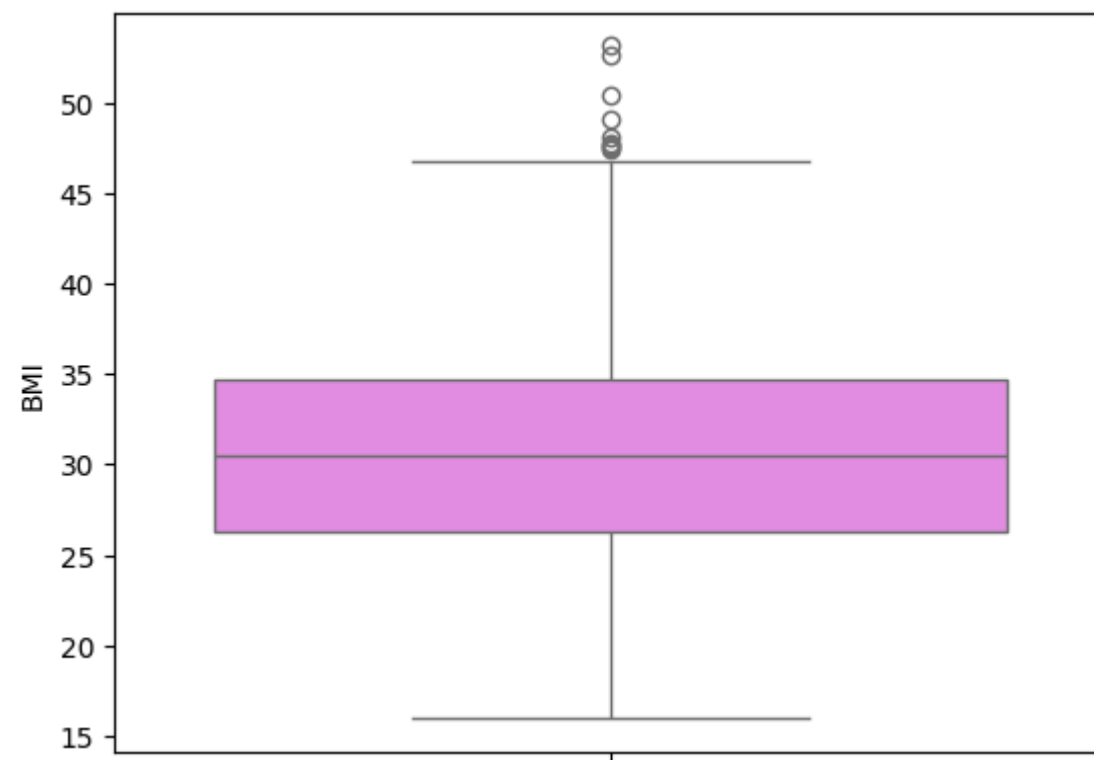
 <Axes: ylabel='age'>



I don't see any outliers here for age feature

```
1 sns.boxplot(df_knn["BMI"], color="violet")
```

 <Axes: ylabel='BMI'>



It appears that we have outliers in BMI feature. The presence of outliers can dramatically change the magnitude of regression coefficients and even the direction of coefficient signs (i.e., from positive to negative or vice versa).

```
1 #I have created a copy of the data_knn to perform zscore before performing IQR. creating a copy of data_knn after performing IQR does not make sense as we would have already dealt with outl
2 df_knn_c=df_knn.copy()
```

```
1 #Dealing with outliers
2
3 # Here I am using IQR to remove outliers.
4 Q1 = df_knn['BMI'].quantile(0.25)
5 Q3 = df_knn['BMI'].quantile(0.75)
6 print(Q3, Q1)
7 IQR = Q3 - Q1
8 print(IQR)
9 upper_bound = Q3 + 1.5 * IQR
10 lower_bound = Q1 - 1.5 * IQR
11 print(upper_bound)
12 print(lower_bound)
```

```
➦ 34.7 26.315
   8.385000000000002
   47.2775
   13.737499999999999
```

```
1 df_knn.shape
```

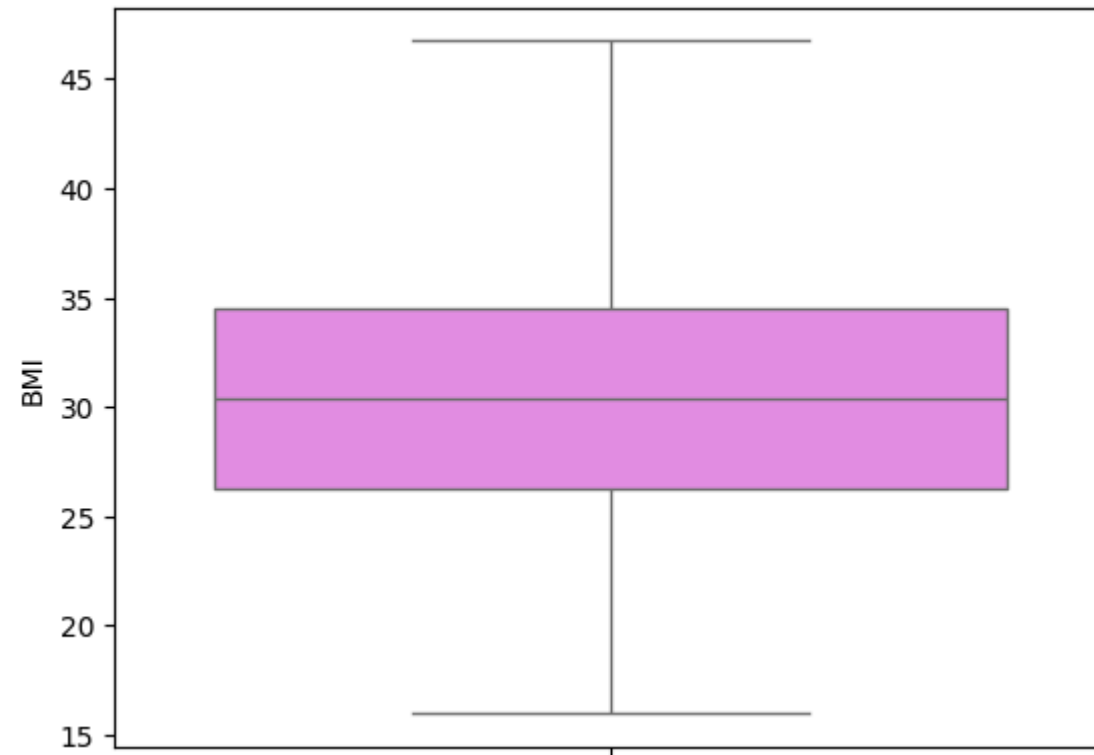
```
➦ (1338, 12)
```

```
1 df_knn = df_knn[df_knn.BMI < upper_bound]
2 df_knn = df_knn[df_knn.BMI > lower_bound]
3 df_knn.shape
```

```
➦ (1329, 12)
```

```
1 sns.boxplot(df_knn["BMI"], color="violet")
```

↗ <Axes: ylabel='BMI'>



we can see that we have handled the outliers pretty well. Having said that, let's use z-score and check how it works

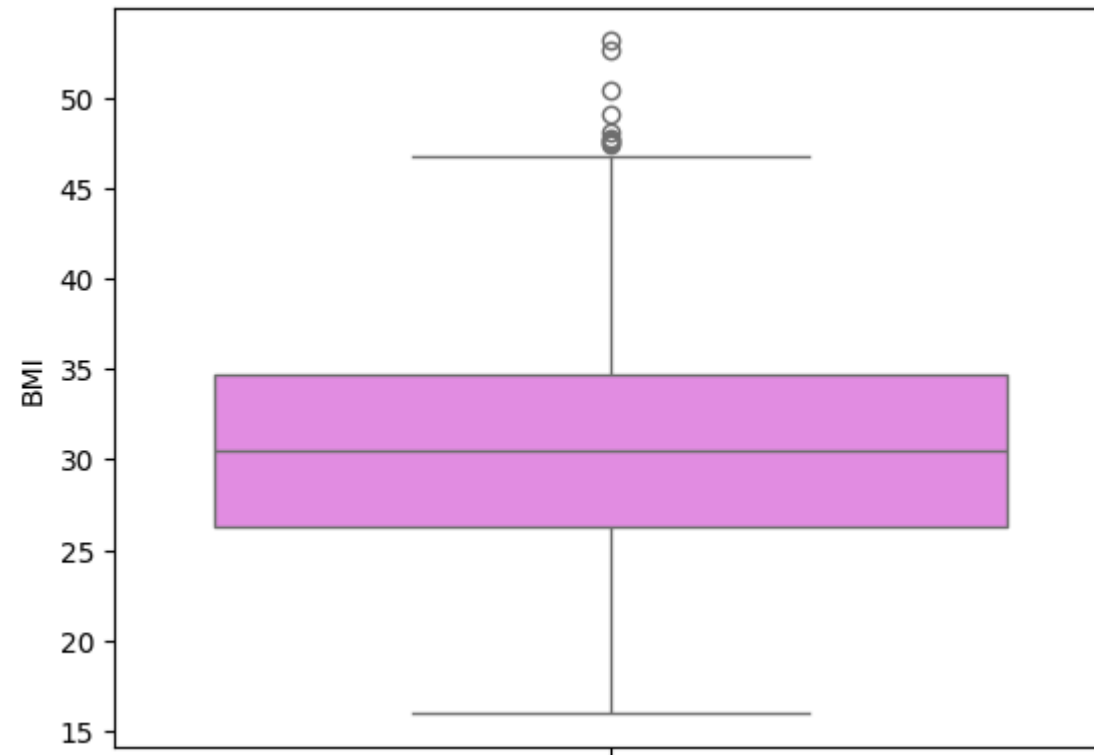
```
1 #I have created a copy of the data_knn to perform zscore before performing IQR. creating a copy of data_knn after performing IQR does not make sense as we would have already dealt with outl
2 #data_knn_c=data_knn.copy()
```

```
1 df_knn_c.shape
```

↗ (1338, 12)

```
1 sns.boxplot(df_knn_c["BMI"], color="violet") # checking the outliers
```

↔ <Axes: ylabel='BMI'>



```
1 df_knn_c['zscore'] = (df_knn_c.BMI - df_knn_c.BMI.mean())/df_knn_c.BMI.std(ddof=0)
2 df_knn_c['zscore'].unique()
```

↔ array([1.33059391e+00, 5.00544853e-01, 3.74515482e-01, -1.31051358e+00,
-2.99823494e-01, -8.13761451e-01, 4.46532265e-01, -4.86412434e-01,
-1.44332711e-01, -7.97394000e-01, -7.35197687e-01, -7.23740471e-01,
 6.03659794e-01, 1.49077563e+00, 1.86886375e+00, -1.00035039e+00,
 1.11580725e-02, -1.12392464e+00, 1.56933939e+00, 7.50966851e-01,
 8.66357380e-01, 2.76310776e-01, 5.54557441e-01, 1.97747012e-01,
-4.39765199e-01, -4.89685924e-01, -1.24831727e+00, 3.37688717e-01,
-2.18126197e+00, 9.14641360e-01, 8.00069204e-01, -7.19648608e-01,
-3.45652356e-01, -3.93117964e-01, 9.31008811e-01, -1.68369146e+00,
 3.68786874e-01, -1.62231352e+00, 9.75200928e-01, 1.50386959e+00,
-6.73001373e-01, 9.68653948e-01, -1.46191250e+00, 1.44315627e-02,
 1.03739724e+00, 1.07831587e+00, 1.30173157e+00, 6.64219362e-01,
-1.01180761e+00, 7.34599401e-01, 8.04161067e-01, 4.77630422e-01,
-4.43857061e-01, 6.08570029e-01, -3.30921650e-01, 1.02184816e+00,
 1.82197934e-01, 1.58465130e-01, -1.28187055e+00, 1.78686217e+00,
-5.48608747e-01, 4.82540657e-01, -9.83982940e-01, -7.81844922e-01,
-1.35716082e+00, -2.96550004e-01, 1.37292998e+00, 8.96637164e-01,
-1.10182859e+00, -9.75799214e-01, -3.62019807e-01, -4.27489611e-01,
 2.12477718e-01, -5.42061767e-01, 5.39826735e-01, -1.83614593e-01,
 7.88611988e-01, 1.48832051e+00, -6.26354139e-01, 1.23953526e+00,
 1.13069171e+00, 1.72155669e+00, 6.69129597e-01, -1.27941543e+00,
 7.33543858e-02, -5.74796668e-01, -6.10805060e-01, 1.43676304e+00,
-9.68433861e-01, 9.62688170e-02, 1.23871689e+00, -1.76143686e+00,
-1.86782529e+00, 1.45371170e-01, -8.59590313e-01, -9.76854757e-02,
-1.29602005e-01, -5.25694316e-01, -3.78387258e-01, 1.34086611e+00,
-4.53677532e-01, 7.16595205e-01, -1.65610397e-01, 8.19710145e-01,
 2.44394247e-01, -3.46470729e-01, 3.00312809e+00, -5.79706904e-01,
-1.20167004e+00, 1.04558097e+00, -1.13947372e+00, -2.84274416e-01,
 1.04452542e-01, 5.24277657e-01, -3.15372572e-01, -3.94754709e-01,
 1.09468332e+00, -2.11906566e+00, 6.52762146e-01, 1.81403001e-01,
-1.41935713e+00, 8.49171557e-01, -8.44041235e-01, -4.35673336e-01,

```
-9.06237548e-01, 1.94473522e-01, 8.65539007e-01, 2.91041482e-01,
-8.85778235e-01, -1.59881789e-01, 1.32873787e+00, -3.54891625e-02,
1.14869591e+00, 1.09959355e+00, -1.07727741e+00, 1.05294632e+00,
1.44167328e+00, -1.02981180e+00, -3.81304277e-01, 1.00466234e+00,
1.45476724e+00, -1.49242946e-01, -1.75430867e-01, -4.08667042e-01,
1.02921352e+00, 3.99885030e-01, -1.93247672e+00, 1.76083857e+00,
-6.74056916e-02, -2.41449815e+00, 4.30983187e-01, 1.14378567e+00,
-4.70863355e-01, -2.47447651e-01, -7.04099530e-01, -3.57263328e-03,
1.83040024e+00, 3.07990135e-02, 2.43575875e-01, 2.28845169e-01,
1.40460934e-01, -7.38471177e-01, -1.99400841e-02, 3.41780580e-01,
-2.07241842e+00, 1.41057512e+00, 2.48486110e-01, -1.09282649e+00,
8.78632968e-01, -1.37680176e+00, -7.05736275e-01, 1.78106071e-01,
2.67071508e-02, -6.51723687e-01, -1.24586215e+00, -1.22785796e+00,
4.89087638e-01, 4.15434109e-01, -9.93803410e-01, 5.18549049e-01,
1.20270850e+00, 1.83285535e+00, 1.51099777e-01, 8.97455537e-01,
-4.71681728e-01, -2.11333705e+00, -1.01508110e+00, -1.38989572e+00,
1.25672108e+00, -2.68725337e-01, -1.40380805e+00, -6.40266472e-01,
-1.13234554e-01, 8.42624576e-01, -1.60594607e+00, 1.30947352e+00,
5.72561637e-01, -8.75139392e-01, 1.54478822e+00, -1.03063017e+00,
-9.02145686e-01, -6.33719491e-01, -1.04617925e+00, 1.90487214e+00,
-1.78598803e+00, 2.75492404e-01, -8.37731425e-02, -2.19622985e-01,
5.70924892e-01, -5.33059669e-01, -5.17510590e-01, -1.65259331e+00,
-1.04945274e+00, 1.76469326e-01, -1.49792090e+00, -4.24216120e-01,
1.61271314e+00, -5.01961512e-01, 7.99013661e-02, -6.69727883e-01,
2.84109033e+00, 9.90750006e-01, 4.39985285e-01, 2.42699382e+00,
-3.09643964e-01, -1.26386635e+00, -4.92959414e-01, -8.67774039e-01,
6.02023049e-01, -1.32606266e+00, 1.11268752e+00, 1.19288802e+00,
```

```
1 #usually we will consider anything more than 3SD as an outlier.
2 df_knn_c[(df_knn_c['zscore']>3)]
```

	age	BMI	Children	health_insurance_price	gender_female	gender_male	smoking_status_no	smoking_status_yes	location_northeast	location_northwest	location_southeast	location_s
116	58.0	49.06	0	11381.3254	0	1	1	0	0	0	1	
847	23.0	50.38	1	2438.0552	0	1	1	0	0	0	1	
1047	22.0	52.58	1	44501.3982	0	1	0	1	0	0	1	
1317	18.0	53.13	0	1163.4627	0	1	1	0	0	0	1	

we can see we have only 4 rows with zscore more than 3.therefore, let's try removing them and observe the boxplot.

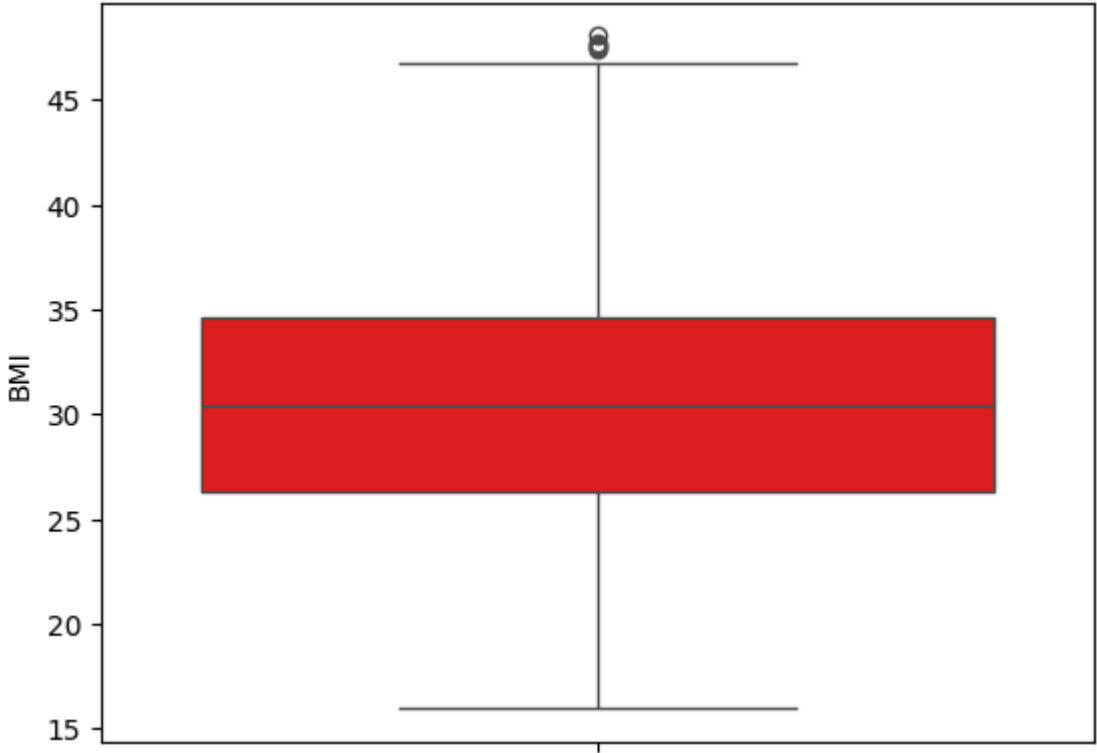
```
1 #removing those rows
2 # delete a few specified rows at index values 116, 847, 1047,1317.
3 # Note that the index values do not always align to row numbers.
4 df_knn_c = df_knn_c.drop(labels=[116,847,1047,1317], axis=0)
```

```
1 df_knn_c.shape
```

(1334, 13)

```
1 sns.boxplot(df_knn_c["BMI"], color="red") # after dealing with outliers using z-score
```

<Axes: ylabel='BMI'>



Clearly, z-score did not perform well, so, let's use IQR.

Note: For outliers, statistically it is good to treat extreme outliers ($+3 \times \text{IQR}$ and $-3 \times \text{IQR}$). Minor outliers are OK. As at the end we are using DT, RF ML models etc.

✓ Bivaritae Analysis

```
1 df.head(5) # again using the raw dataset
```

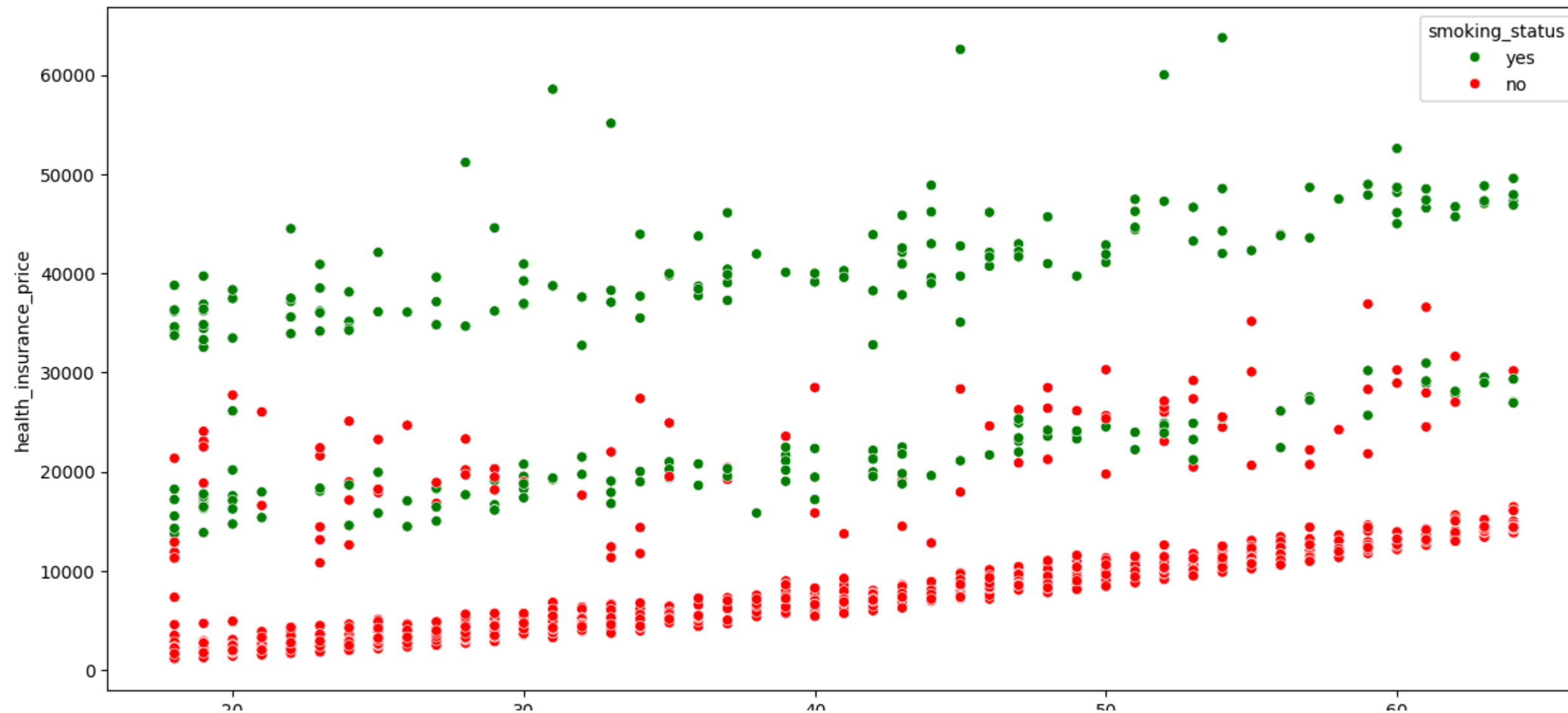


	age	gender	BMI	Children	smoking_status	location	health_insurance_price
0	19.0	female	NaN	0	yes	southwest	16884.92400
1	18.0	male	33.770	1	no	southeast	1725.55230
2	28.0	male	33.000	3	no	southeast	4449.46200
3	33.0	male	22.705	0	no	northwest	21984.47061
4	32.0	male	28.880	0	no	northwest	3866.85520

✓ Numerical-Numerical variables


```
1 plt.figure(figsize=(15, 7))
2 sns.scatterplot(x = df['age'], y = df['health_insurance_price'], hue = df['smoking_status'], palette=['green','red'])
```

<Axes: xlabel='age', ylabel='health_insurance_price'>

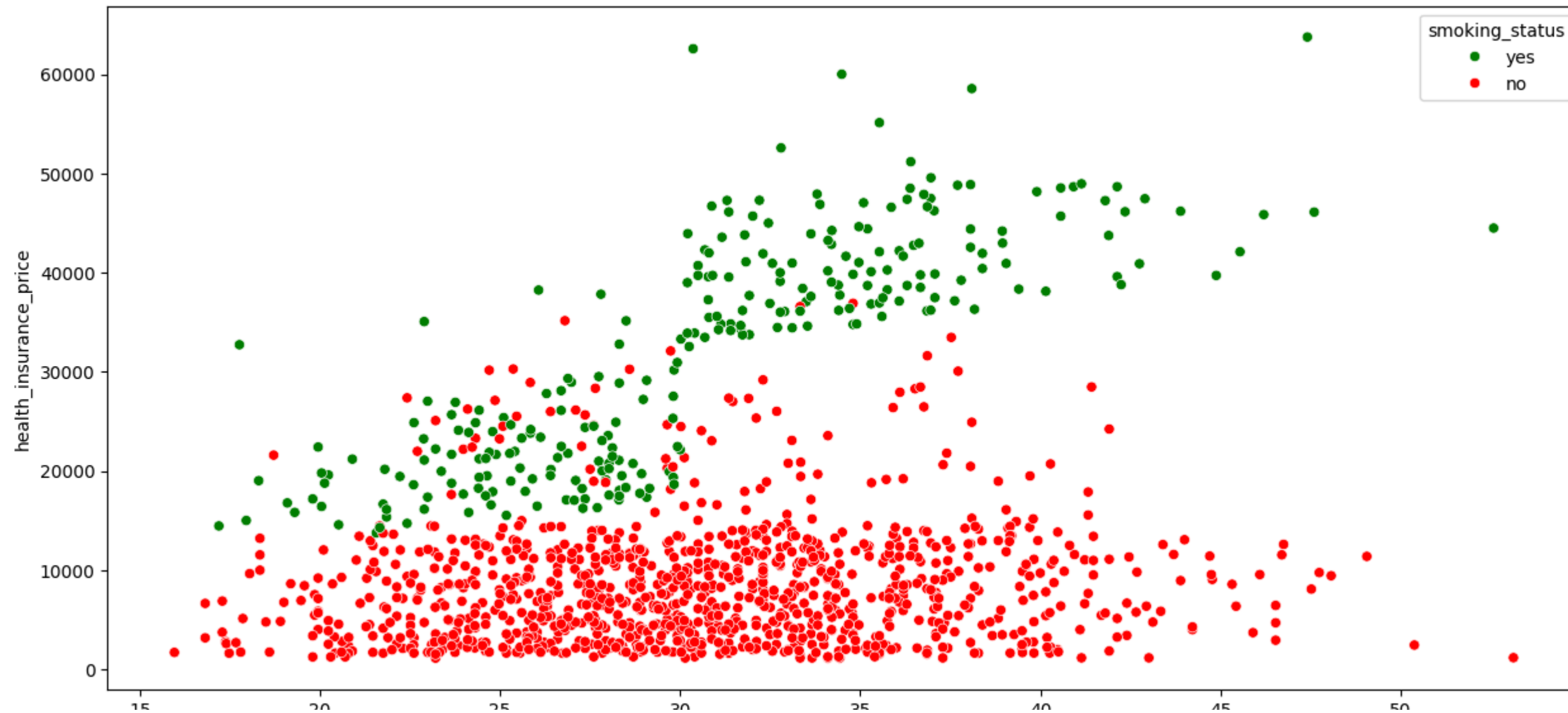


Observation:

It appears to us that is Relation between age and insurance_price as we see that increasing in age causing increase in price in case of non-smokers Having said that, Smokers insurance prices are higher in Comparison to non smokers.

```
1 plt.figure(figsize=(15, 7))
2 sns.scatterplot(x = df['BMI'], y = df['health_insurance_price'], hue = df['smoking_status'], palette=['green','red'])
```

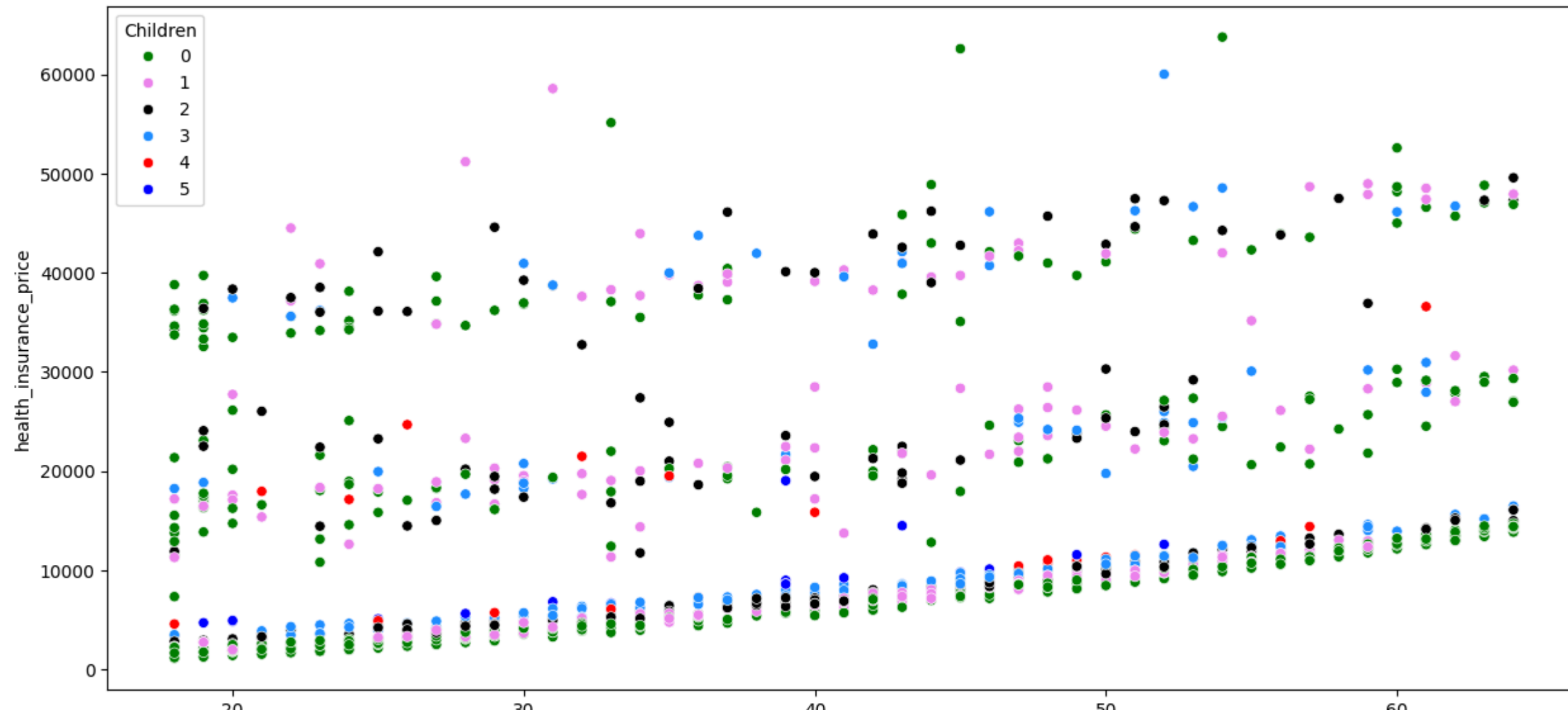
<Axes: xlabel='BMI', ylabel='health_insurance_price'>



It appears to us as the BMI of smokers increases their insurance price escalates.

```
1 plt.figure(figsize=(15, 7))
2 sns.scatterplot(x = df['age'], y = df['health_insurance_price'], hue = df['Children'], palette=['green','violet','black','dodgerblue','red',"blue"])
```

<Axes: xlabel='age', ylabel='health_insurance_price'>



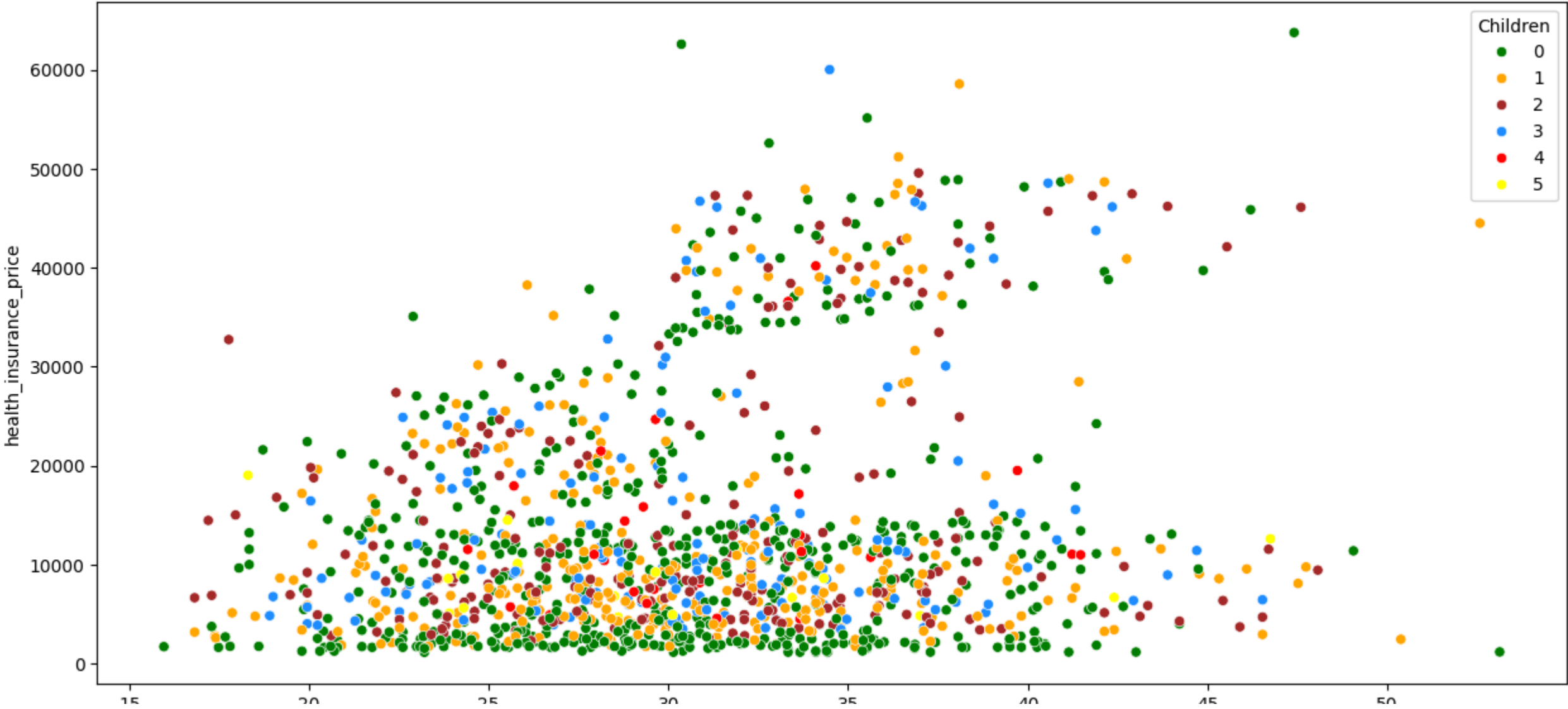
It is tough to say there is an impact of number of children of people with any age on insurance price.

ex: we can't say elder people with more children paid more insurance price nor we can say younger people with more children paid more insurance price.

Similarly, we can't say younger people with less children paid more insurance price nor we can say elder people with less children paid more insurance price.

```
1 plt.figure(figsize=(15, 7))
2 sns.scatterplot(x = df['BMI'], y = df['health_insurance_price'], hue = df['Children'], palette=['green','orange','brown','dodgerblue','red',"yellow"])
```

<Axes: xlabel='BMI', ylabel='health_insurance_price'>



It seems BMI is not having much impact on health insurance. Smoking status is having much higher impact on health insurance price as per above figures.

Category- Numerical Variables

1 df.head(5)

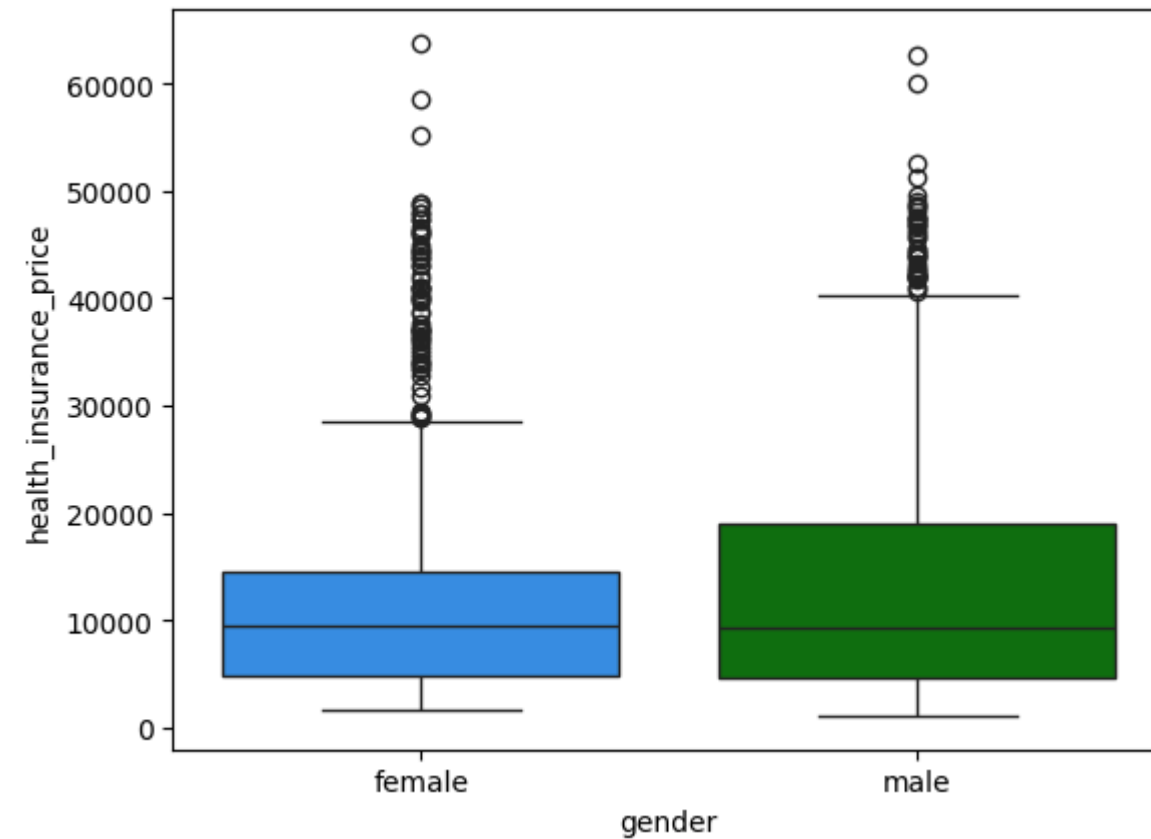
	age	gender	BMI	Children	smoking_status	location	health_insurance_price
0	19.0	female	NaN	0	yes	southwest	16884.92400
1	18.0	male	33.770	1	no	southeast	1725.55230
2	28.0	male	33.000	3	no	southeast	4449.46200
3	33.0	male	22.705	0	no	northwest	21984.47061
4	32.0	male	28.880	0	no	northwest	3866.85520

```
1 sns.boxplot(data=df, x="gender", y="health_insurance_price",palette=['dodgerblue','green'])
```


↗ <ipython-input-60-dfc510a69ca8>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.boxplot(data=df, x="gender", y="health_insurance_price",palette=['dodgerblue','green'])  
<Axes: xlabel='gender', ylabel='health_insurance_price'>
```

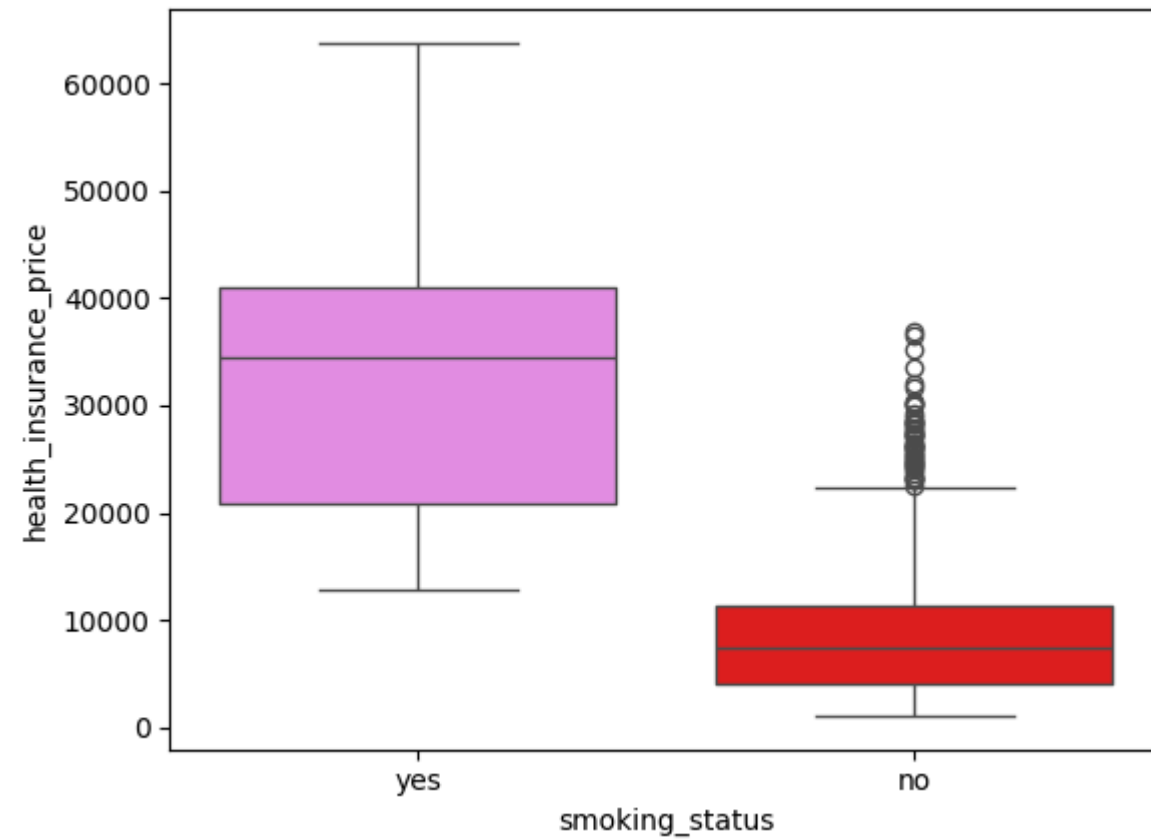


```
1 #box plot for smoking status vs health_insurance_price  
2 sns.boxplot(data=df, x="smoking_status", y="health_insurance_price",palette=['violet','red'])
```

 <ipython-input-61-f152576d01bb>:2: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.boxplot(data=df, x="smoking_status", y="health_insurance_price",palette=['violet','red'])  
<Axes: xlabel='smoking_status', ylabel='health_insurance_price'>
```



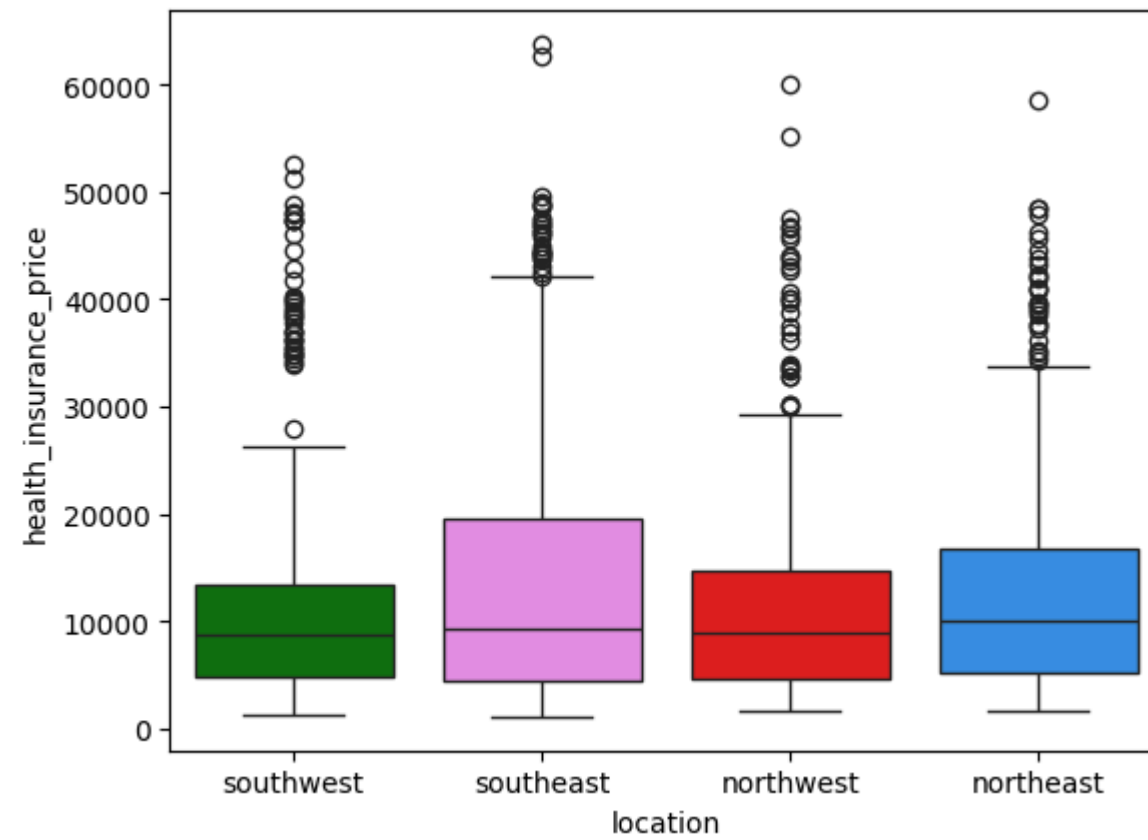
There is an impact of smoking status on health insurance price. if the status is yes then health insurance price range is high for them.

```
1 # using box plot for location vs health_insurance_price  
2 # Boxplots allow us a simple way to compare groups and view dispersion and spread in data.  
3  
4 sns.boxplot(data=df, x="location", y="health_insurance_price",palette=['green','violet','red','dodgerblue'])
```

 <ipython-input-62-583795055659>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.boxplot(data=df, x="location", y="health_insurance_price",palette=['green','violet','red','dodgerblue'])  
<Axes: xlabel='location', ylabel='health_insurance_price'>
```



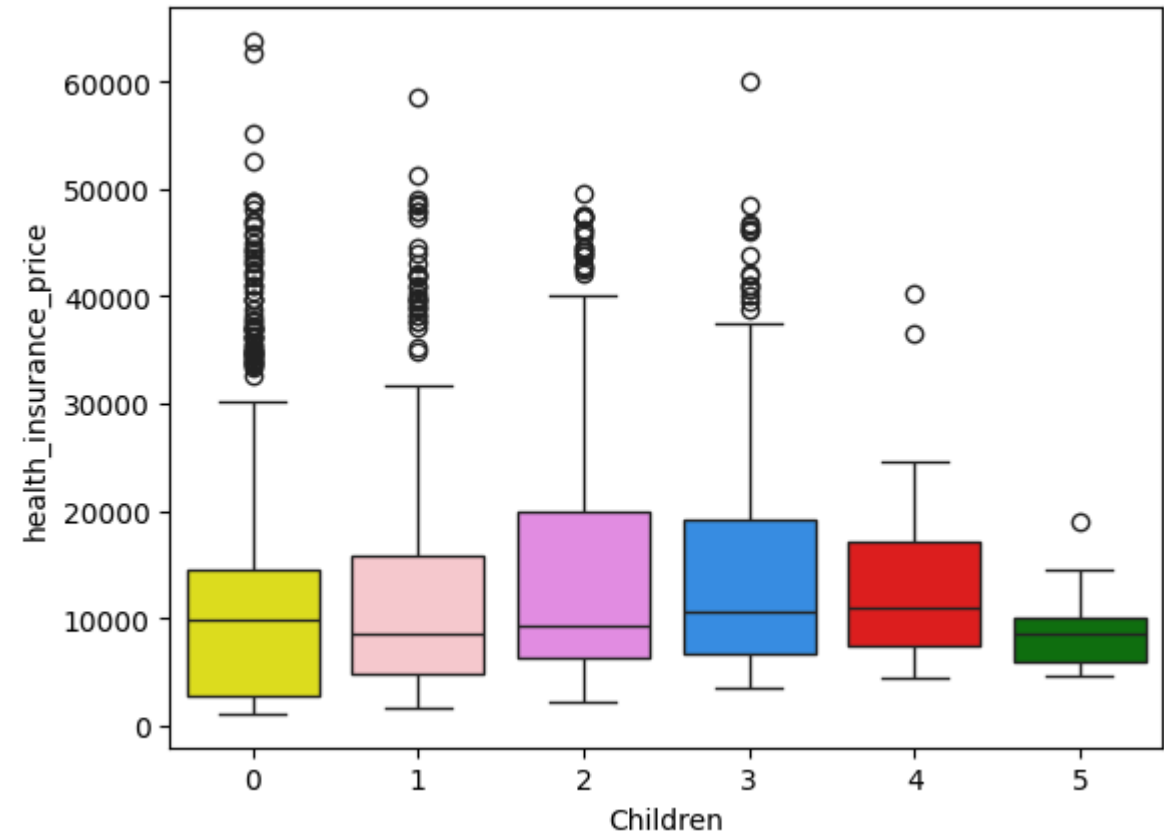
The median value of all the groups looks almost equal.so, we can't see much of an impact of location on health insurance price

```
1 #box plot between children and insurance price  
2  
3 sns.boxplot(data=df, x="Children", y="health_insurance_price",palette=['yellow','pink','violet','dodgerblue',"red","green"])
```

```
<ipython-input-63-f6d25157ae73>:3: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.boxplot(data=df, x="Children", y="health_insurance_price",palette=['yellow','pink','violet','dodgerblue',"red","green"])
<Axes: xlabel='Children', ylabel='health_insurance_price'>
```

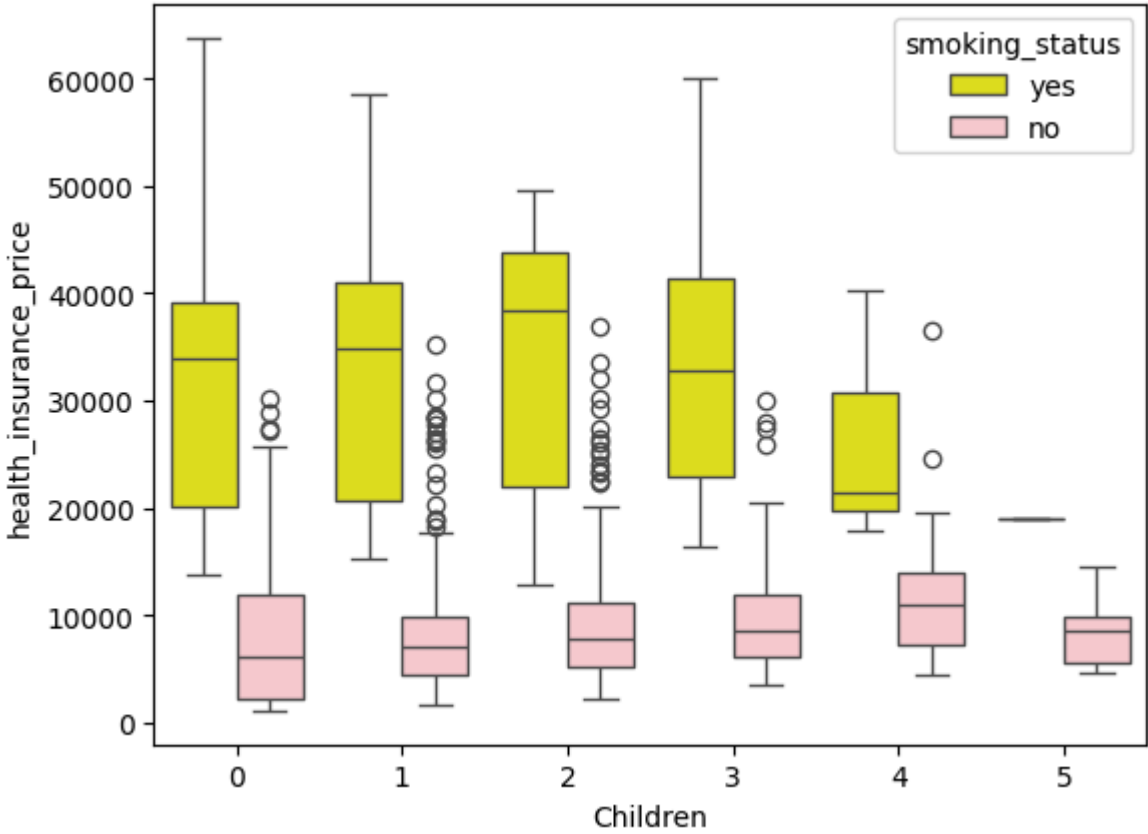


Looking at each of the groups median value, we can say that there is not much of an impact of number of children on health insurance price


```
1 sns.boxplot(data=df, x="Children", y="health_insurance_price",hue="smoking_status", palette=['yellow','pink','violet','dodgerblue',"red","green"])
```



```
<ipython-input-64-390450a9adf0>:1: UserWarning: The palette list has more values (6) than needed (2), which may not be intended.  
sns.boxplot(data=df, x="Children", y="health_insurance_price",hue="smoking_status", palette=['yellow','pink','violet','dodgerblue',"red","green"])  
<Axes: xlabel='Children', ylabel='health_insurance_price'>
```

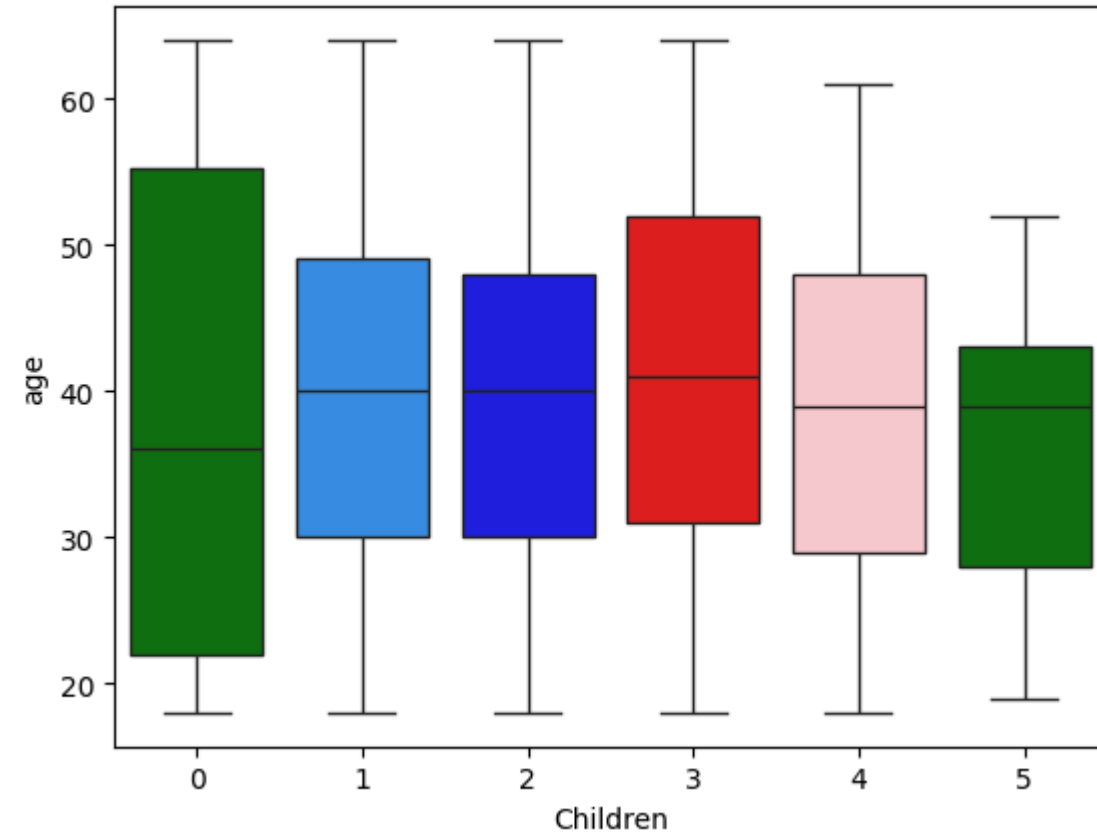


```
1 #box plot between children and age  
2  
3 sns.boxplot(data=df, x="Children", y="age",palette=['green','dodgerblue',"blue","red","pink"])
```

 <ipython-input-65-b5add6bf4f9e>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.boxplot(data=df, x="Children", y="age",palette=['green','dodgerblue',"blue","red","pink"])
<ipython-input-65-b5add6bf4f9e>:3: UserWarning:
The palette list has fewer values (5) than needed (6) and will cycle, which may produce an uninterpretable plot.
sns.boxplot(data=df, x="Children", y="age",palette=['green','dodgerblue',"blue","red","pink"])
<Axes: xlabel='Children', ylabel='age'>
```



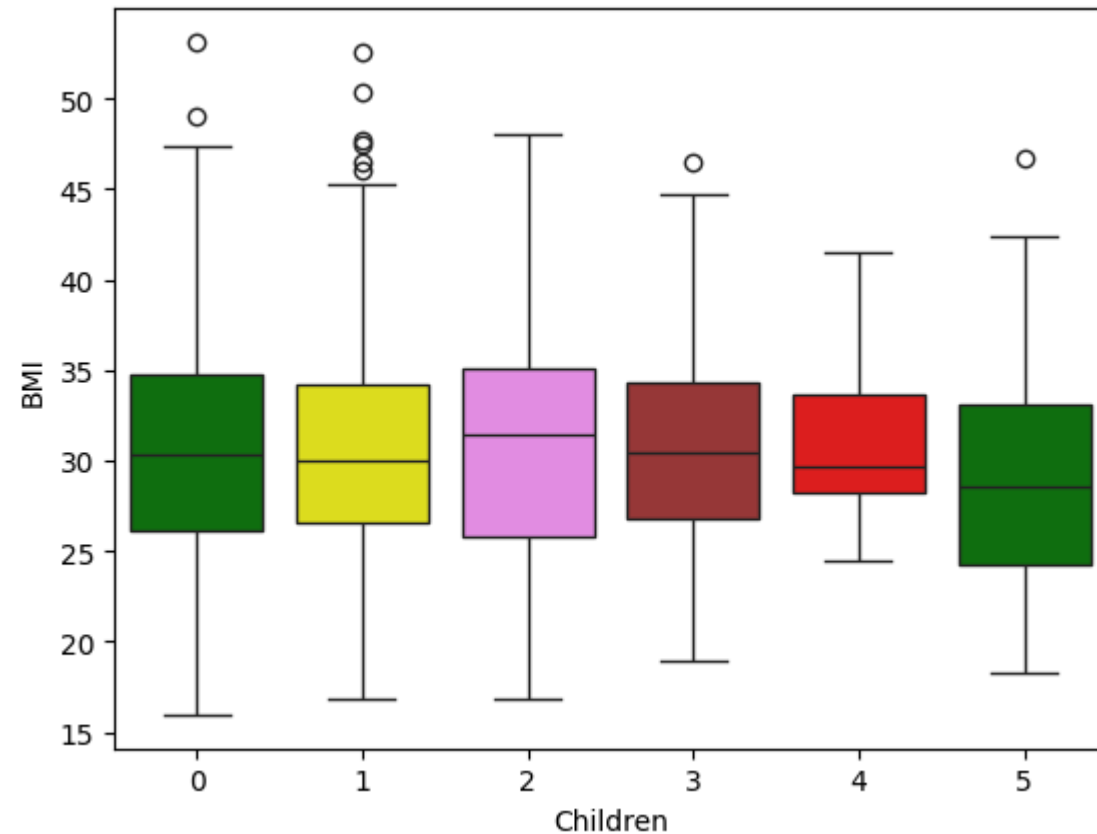
From this figure we can't say elder people have more children or have less children. Similarly, we can't say that younger people have more children or have less children. Basically, number of children is not determined by age.

```
1 # box plot between children and BMI
2
3 sns.boxplot(data=df, x="Children", y="BMI",palette=['Green','yellow',"violet","brown","red"])
```

<ipython-input-66-839fe01d6df1>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.boxplot(data=df, x="Children", y="BMI",palette=['Green','yellow',"violet","brown","red"])
<ipython-input-66-839fe01d6df1>:3: UserWarning:
The palette list has fewer values (5) than needed (6) and will cycle, which may produce an uninterpretable plot.
sns.boxplot(data=df, x="Children", y="BMI",palette=['Green','yellow',"violet","brown","red"])
<Axes: xlabel='Children', ylabel='BMI'>
```



Seemingly, there is no effect of number of children on people's BMI

```
1 # checking the correlation
2
3 df_corr = pd.DataFrame(df_knn)
4 print(df_corr.corr(method = 'spearman'))
```

```

age      BMI  Children  health_insurance_price \
age      1.000000  0.092537  0.053706      0.533113
BMI      0.092537  1.000000  0.017608      0.117599
Children 0.053706  0.017608  1.000000      0.133351
health_insurance_price 0.533113  0.117599  0.133351      1.000000
gender_female 0.020233 -0.043856 -0.017342     -0.013824
gender_male -0.020233  0.043856  0.017342      0.013824
smoking_status_no 0.020570 -0.003057 -0.016300     -0.661591
smoking_status_yes -0.020570  0.003057  0.016300      0.661591
location_northeast 0.000304 -0.126150 -0.027658      0.047263
location_northwest 0.001184 -0.117972  0.034324     -0.021529
location_southeast -0.019989  0.230326 -0.015881      0.017735
location_southwest 0.019145  0.006345  0.009662     -0.043970
```

	gender_female	gender_male	smoking_status_no \
age	0.020233	-0.020233	0.020570
BMI	-0.043856	0.043856	-0.003057
Children	-0.017342	0.017342	-0.016300
health_insurance_price	-0.013824	0.013824	-0.661591
gender_female	1.000000	-1.000000	0.079854
gender_male	-1.000000	1.000000	-0.079854
smoking_status_no	0.079854	-0.079854	1.000000
smoking_status_yes	-0.079854	0.079854	-1.000000
location_northeast	-0.000573	0.000573	-0.004947
location_northwest	0.009961	-0.009961	0.035940
location_southeast	-0.010263	0.010263	-0.068270
location_southwest	0.001195	-0.001195	0.039440

	smoking_status_yes	location_northeast \
age	-0.020570	0.000304
BMI	0.003057	-0.126150
Children	0.016300	-0.027658
health_insurance_price	0.661591	0.047263
gender_female	-0.079854	-0.000573
gender_male	0.079854	0.000573
smoking_status_no	-1.000000	-0.004947
smoking_status_yes	1.000000	0.004947
location_northeast	0.004947	1.000000
location_northwest	-0.035940	-0.322387
location_southeast	0.068270	-0.343402
location_southwest	-0.039440	-0.321730

	location_northwest	location_southeast \
age	0.001184	-0.019989
BMI	-0.117972	0.230326
Children	0.034324	-0.015881
health_insurance_price	-0.021529	0.017735
gender_female	0.009961	-0.010263
gender_male	-0.009961	0.010263
smoking_status_no	0.035940	-0.068270
smoking_status_yes	-0.035940	0.068270
location_northeast	-0.322387	-0.343402
location_northwest	1.000000	-0.344807
location_southeast	-0.344807	1.000000
location_southwest	-0.323046	-0.344105

	location_southwest
age	0.019145

Barring Age which has moderate co-relation, we don't see much of the corelation between the target variable and I variables.

```
1 # Examine multicollinearity using VIF
2 from statsmodels.stats.outliers_influence import variance_inflation_factor
3
4 # the independent variables set
5 X_vif = df_knn.drop(['health_insurance_price'], axis=1)
6
7 # VIF dataframe
8 vif_df = pd.DataFrame()
9 vif_df["feature"] = X_vif.columns
10
11 # calculating VIF for each feature
```

```

12 vif_df["VIF"] = [variance_inflation_factor(X_vif.values, i)
13                  for i in range(len(X_vif.columns))]
14
15 print(vif_df)

```

```

↗
   feature      VIF
0      age  1.013592
1      BMI  1.088373
2  Children  1.003823
3  gender_female      inf
4  gender_male      inf
5  smoking_status_no      inf
6  smoking_status_yes      inf
7  location_northeast      inf
8  location_northwest      inf
9  location_southeast      inf
10 location_southwest      inf
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.py:197: RuntimeWarning: divide by zero encountered in scalar divide
  vif = 1. / (1. - r_squared_i)

```

You can observe VIF is infinite for many columns that means there exists perfect correlation between them. so, let's drop 1 column from each of the categories as below

```

1 # Examine multicollinearity using VIF
2 from statsmodels.stats.outliers_influence import variance_inflation_factor
3
4 # the independent variables set
5 X_vif = df_knn.drop(['health_insurance_price', "gender_female", "smoking_status_no", "location_southwest"], axis=1)
6
7 # VIF dataframe
8 vif_df = pd.DataFrame()
9 vif_df["feature"] = X_vif.columns
10
11 # calculating VIF for each feature
12 vif_df["VIF"] = [variance_inflation_factor(X_vif.values, i)
13                 for i in range(len(X_vif.columns))]
14
15 print(vif_df)


```

```

↗
   feature      VIF
0      age  7.806095
1      BMI 10.751487
2  Children  1.807568
3  gender_male  2.003648
4  smoking_status_yes  1.264723
5  location_northeast  1.833465
6  location_northwest  1.839118
7  location_southeast  2.132683

```


```
1 X_vif.head(5)
```



	age	BMI	Children	gender_male	smoking_status_yes	location_northeast	location_northwest	location_southeast
0	19.0	38.84134	0	0	1	0	0	0
1	18.0	33.77000	1	1	0	0	0	1
2	28.0	33.00000	3	1	0	0	0	1
3	33.0	22.70500	0	1	0	0	1	0
4	32.0	28.88000	0	1	0	0	1	0

VIF>5 is not accepted. hence drop those columns


```
1 X_vif.head(5)
```



	age	BMI	Children	gender_male	smoking_status_yes	location_northeast	location_northwest	location_southeast
0	19.0	38.84134	0	0	1	0	0	0
1	18.0	33.77000	1	1	0	0	0	1
2	28.0	33.00000	3	1	0	0	0	1
3	33.0	22.70500	0	1	0	0	1	0
4	32.0	28.88000	0	1	0	0	1	0

```
1 X=X_vif.drop(["age","BMI"], axis=1)
```

```
1 # VIF dataframe
2 vif_df = pd.DataFrame()
3 vif_df["feature"] = X.columns
4
5 # calculating VIF for each feature
6 vif_df["VIF"] = [variance_inflation_factor(X.values, i)
7                 for i in range(len(X.columns))]
8
9 print(vif_df)
```



	feature	VIF
0	Children	1.573202
1	gender_male	1.725221
2	smoking_status_yes	1.245483
3	location_northeast	1.314837
4	location_northwest	1.325890
5	location_southeast	1.377983

After dropping the columns with VIF>5 we are left with minimum no. of features.


In the interest of avoiding multicollinearity we lost many of the features.

Therefore,I am not using Linear regression as it did not perform well when I checked it because of which I have chosen non -parametric algorithms. like rf, DT and SVM (Support Vector Machines with Gaussian Kernels)

```
1 # Separating I variables and the dependent variable
2 X=df_knn.drop(("health_insurance_price"), axis=1)
```


```
1 y=df_knn["health_insurance_price"]
```

```
1 X.head(5)
```



	age	BMI	Children	gender_female	gender_male	smoking_status_no	smoking_status_yes	location_northeast	location_northwest	location_southeast	location_southwest
0	19.0	38.84134	0	1	0	0	1	0	0	0	1
1	18.0	33.77000	1	0	1	1	0	0	0	1	0
2	28.0	33.00000	3	0	1	1	0	0	0	1	0
3	33.0	22.70500	0	0	1	1	0	0	1	0	0
4	32.0	28.88000	0	0	1	1	0	0	1	0	0


```
1 y.head(5)
```



	health_insurance_price
0	16884.92400
1	1725.55230
2	4449.46200
3	21984.47061
4	3866.85520

dtype: float64

```
1 X.shape
```

 (1329, 11)

Feature selection

```
1 # Backward feature elimination technique.
2 # note: you can choose a different one proceed
3 from sklearn.feature_selection import RFE
4 from sklearn.tree import DecisionTreeRegressor
5 rfe = RFE(estimator=DecisionTreeRegressor(random_state=0), n_features_to_select=8) # taking only 8 features
```

```
6 rfe = rfe.fit(X, y)
7 print(X.columns)
8 print(rfe.support_)
9 rfe.ranking_
```

```
➦ Index(['age', 'BMI', 'Children', 'gender_female', 'gender_male',
        'smoking_status_no', 'smoking_status_yes', 'location_northeast',
        'location_northwest', 'location_southeast', 'location_southwest'],
        dtype='object')
[ True  True  True  True False False  True  True  True  True False]
array([1, 1, 1, 1, 2, 4, 1, 1, 1, 1, 3])
```

Let's use columns with rank 1

```
1 # using ranked 1 columns to create X dataframe
2 X=X[["age", "BMI", "Children","gender_female","smoking_status_yes","location_northeast","location_northwest","location_southeast"]]
```

```
1 X.shape
```

```
➦ (1329, 8)
```

```
1 y.shape
```

```
➦ (1329,)
```

```
1 # performing cross validation technique for that using train_test_split to split X and y
2 from sklearn.model_selection import train_test_split
```

```
1 ## this is called hold out cross validation technique
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

✓ Feature Scaling

```
1 # performing transformation of data
2 from sklearn.preprocessing import StandardScaler
3 sc = StandardScaler()
4 X_train=sc.fit_transform(X_train)
5 X_train
```

```
➦ array([[ -0.13379329,  0.59031245, -0.09862461, ..., -0.54582442,
          1.7484999 , -0.62558873],
        [-1.49419206,  1.12253645, -0.09862461, ..., -0.54582442,
          -0.57191882,  1.59849427],
        [ 0.00940658, -0.10817933, -0.09862461, ...,  1.83209098,
          -0.57191882, -0.62558873],
        ...,
        [ 0.15260645, -1.10829257, -0.09862461, ...,  1.83209098,
          -0.57191882, -0.62558873],
        [ 0.79700586,  2.37246912, -0.09862461, ..., -0.54582442,
```



```
-0.57191882, 1.59849427],
[ 0.58220606, -1.08155604, -0.09862461, ..., -0.54582442,
 -0.57191882, -0.62558873]])
```

```
1 X_test=sc.transform(X_test)
2 X_test
```

```
→ array([[ -1.35099219, -1.43247296, -0.09862461, ..., -0.54582442,
          -0.57191882, -0.62558873],
         [ 0.08100651, -0.1624879 , -0.91767116, ..., -0.54582442,
          -0.57191882, -0.62558873],
         [ 0.22420638, -0.09564658, -0.91767116, ..., -0.54582442,
          -0.57191882, -0.62558873],
         ...,
         [ 0.72540593, -0.83090109,  0.72042194, ..., -0.54582442,
          -0.57191882, -0.62558873],
         [-1.42259212,  0.00294436, -0.91767116, ..., -0.54582442,
           1.7484999 , -0.62558873],
         [-0.42019303, -1.40991402, -0.09862461, ...,  1.83209098,
          -0.57191882, -0.62558873]])
```

✓ Random Forest Regressor

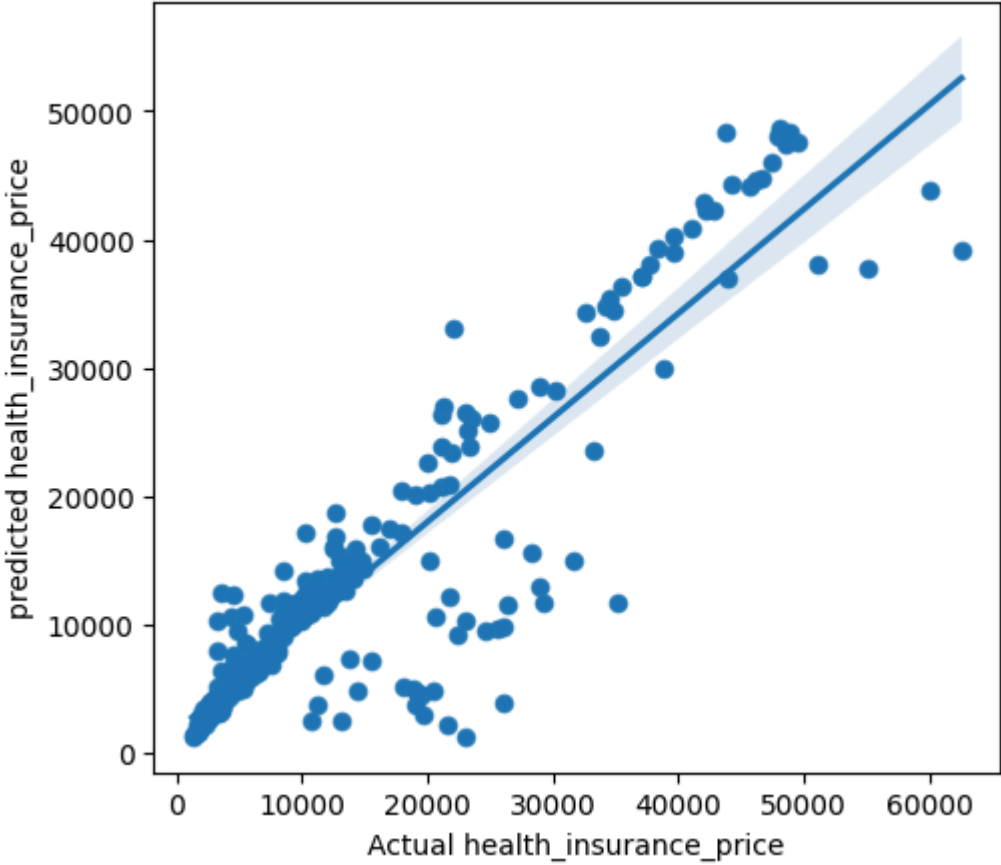
```
1 from sklearn.ensemble import RandomForestRegressor
2 import numpy as np
```

```
1 rf = RandomForestRegressor(n_estimators = 100)
2 rf.fit(X_train,y_train)
3 y_pred = rf.predict(X_test)
```


```
1 # plotting actual and predicted values using scatterplot below
2 import matplotlib.gridspec as gridspec
3
4 fig = plt.figure(figsize=(12,5))
5 grid = gridspec.GridSpec(ncols=2, nrows=1, figure=fig)
6
7 ax1 = fig.add_subplot(grid[0, 0])
8
9 sns.scatterplot(x = y_test, y = y_pred, ax=ax1)
10 sns.regplot(x = y_test, y=y_pred, ax=ax1)
11
12 ax1.set_title("Actual health_insurance_price vs. predicted health_insurance_price")
13 ax1.set_xlabel('Actual health_insurance_price')
14 ax1.set_ylabel('predicted health_insurance_price')
```

 Text(0, 0.5, 'predicted health_insurance_price')

Actual health_insurance_price vs. predicted health_insurance_price


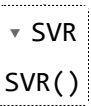


```
1 # checking the metrics for RFR
2 from sklearn import metrics
3 print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
4 print('MSE:', metrics.mean_squared_error(y_test, y_pred))
5 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
6 print("Adjusted R^2:",(1 - (1-rf.score(X_train, y_train))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1)))
```

 MAE: 3113.5441384182
MSE: 33978065.94502492
RMSE: 5829.070761710216
Adjusted R^2: 0.9761104122980105

Support Vector Machines

```
1 # Importing SVR library and creating model
2 from sklearn.svm import SVR
3 regressor = SVR(kernel = 'rbf')
4 regressor.fit(X_train, y_train)
```

  SVR
SVR()

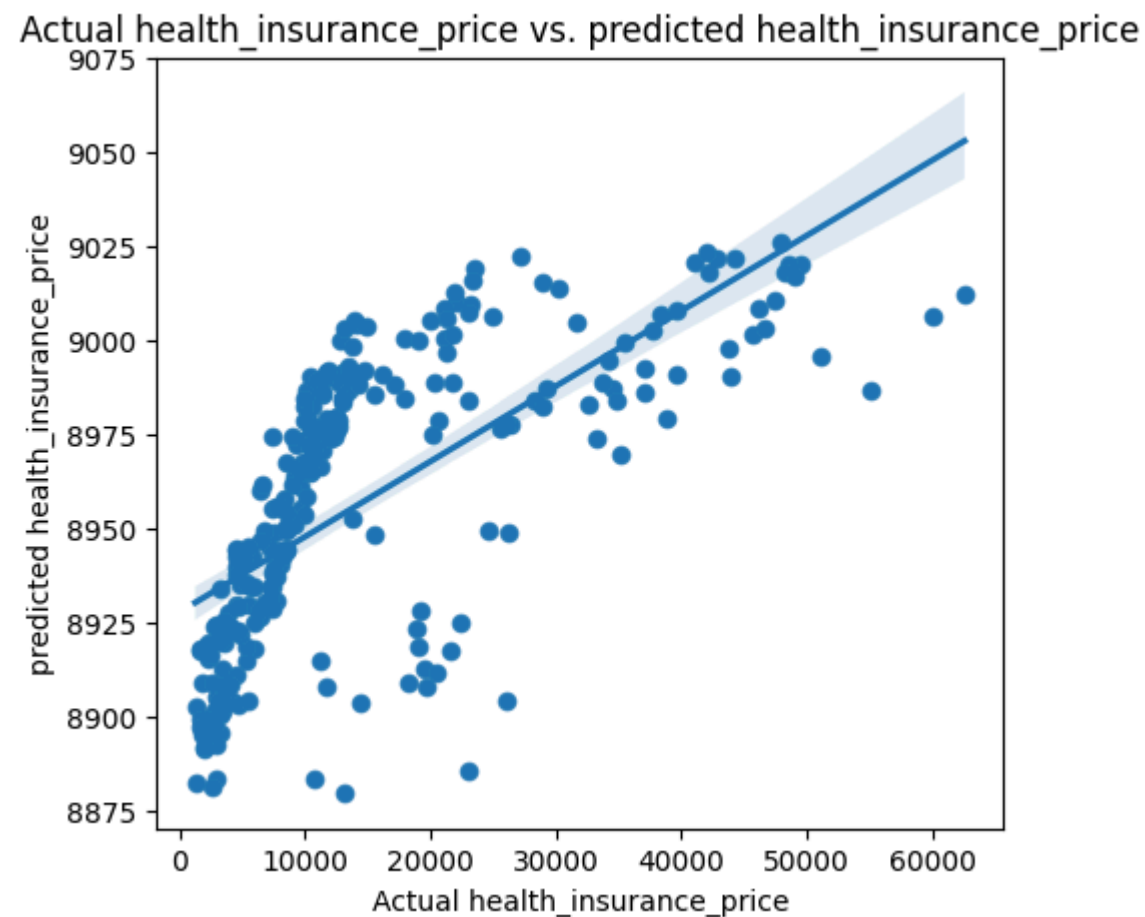
```
1 y_pred1 = regressor.predict(X_test) # prediciting y
```

```
1 # Calculating error
2 print('MAE:', metrics.mean_absolute_error(y_test, y_pred1))
3 print('MSE:', metrics.mean_squared_error(y_test, y_pred1))
4 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred1)))
```

```
MAE: 9309.005171151472
MSE: 204845616.32290965
RMSE: 14312.428735994099
```

```
1 # plotting actual and predicted values using scatterplot below
2 import matplotlib.gridspec as gridspec
3
4 fig = plt.figure(figsize=(12,5))
5 grid = gridspec.GridSpec(ncols=2, nrows=1, figure=fig)
6
7 ax1 = fig.add_subplot(grid[0, 0])
8
9 sns.scatterplot(x = y_test, y = y_pred1, ax=ax1)
10 sns.regplot(x = y_test, y=y_pred1, ax=ax1)
11
12 ax1.set_title("Actual health_insurance_price vs. predicted health_insurance_price")
13 ax1.set_xlabel('Actual health_insurance_price')
14 ax1.set_ylabel('predicted health_insurance_price')
```

```
Text(0, 0.5, 'predicted health_insurance_price')
```



This was done without scaling let's see if we can observe any difference if we scale the data.

```
1 #after scaling
2 # Importing SVR library and creating model
3 from sklearn.svm import SVR
4 regressor = SVR(kernel = 'rbf')
5 regressor.fit(X_train, y_train)
```



SVR
SVR()

```
1 y_pred1 = regressor.predict(X_test) # prediciting y
```

```
1 # Calculating error
2 print('MAE:', metrics.mean_absolute_error(y_test, y_pred1))
3 print('MSE:', metrics.mean_squared_error(y_test, y_pred1))
4 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred1)))
5 print("Adjusted R^2:", (1 - (1-regressor.score(X_train, y_train))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1)))
```

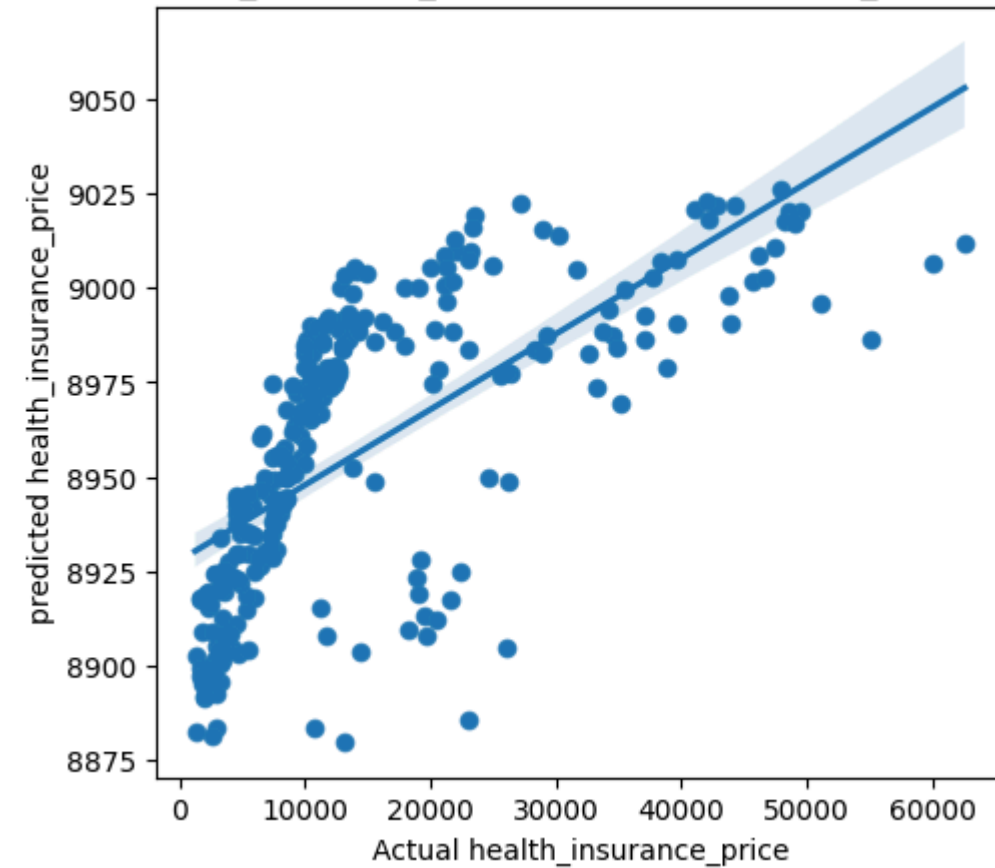


```
MAE: 9309.005171151472
MSE: 204845616.32290965
RMSE: 14312.428735994099
Adjusted R^2: -0.11171180784847201
```

```
1 import matplotlib.gridspec as gridspec
2
3 fig = plt.figure(figsize=(12,5))
4 grid = gridspec.GridSpec(ncols=2, nrows=1, figure=fig)
5
6 ax1 = fig.add_subplot(grid[0, 0])
7
8 sns.scatterplot(x = y_test, y = y_pred1, ax=ax1)
9 sns.regplot(x = y_test, y=y_pred1, ax=ax1)
10
11 ax1.set_title("Actual health_insurance_price vs. predicted health_insurance_price")
12 ax1.set_xlabel('Actual health_insurance_price')
13 ax1.set_ylabel('predicted health_insurance_price')
```

```
Text(0, 0.5, 'predicted health_insurance_price')
```

Actual health_insurance_price vs. predicted health_insurance_price



Seems a lot of improvement after scaling the data.

That said, RF worked the same with or without scaling. I have checked it and you can also do it.

✓ XGBOOST

```
1 !pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost) (2.23.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)
```

```
1 import xgboost as xgb
2
```

```
1 #We have to instantiate an XGBoost regressor object by calling the XGBRegressor() class from the XGBoost library with the hyper-parameters passed as arguments.
2 xg_reg = xgb.XGBRegressor(objective='reg:linear', colsample_bytree = 0.3, learning_rate = 0.1,
3 max_depth = 5, alpha = 10, n_estimators = 10)
```

```
1 #Fit the regressor to the training set and make predictions on the test set using the familiar .fit() and .predict() methods
2 xg_reg.fit(X_train,y_train)
```

```
3 preds = xg_reg.predict(X_test)
```

➦ /usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [09:20:48] WARNING: /workspace/src/objective/regression_obj.cu:227: reg:linear is now deprecated in favor of reg:s
warnings.warn(msg, UserWarning)

```
1 # we have to #Compute the rmse by invoking the mean_squared_error function from sklearn's
2 #metrics module.
3 rmse = np.sqrt(metrics.mean_squared_error(y_test, preds))
4 print("RMSE: %f" % (rmse))
```

➦ RMSE: 11278.576329

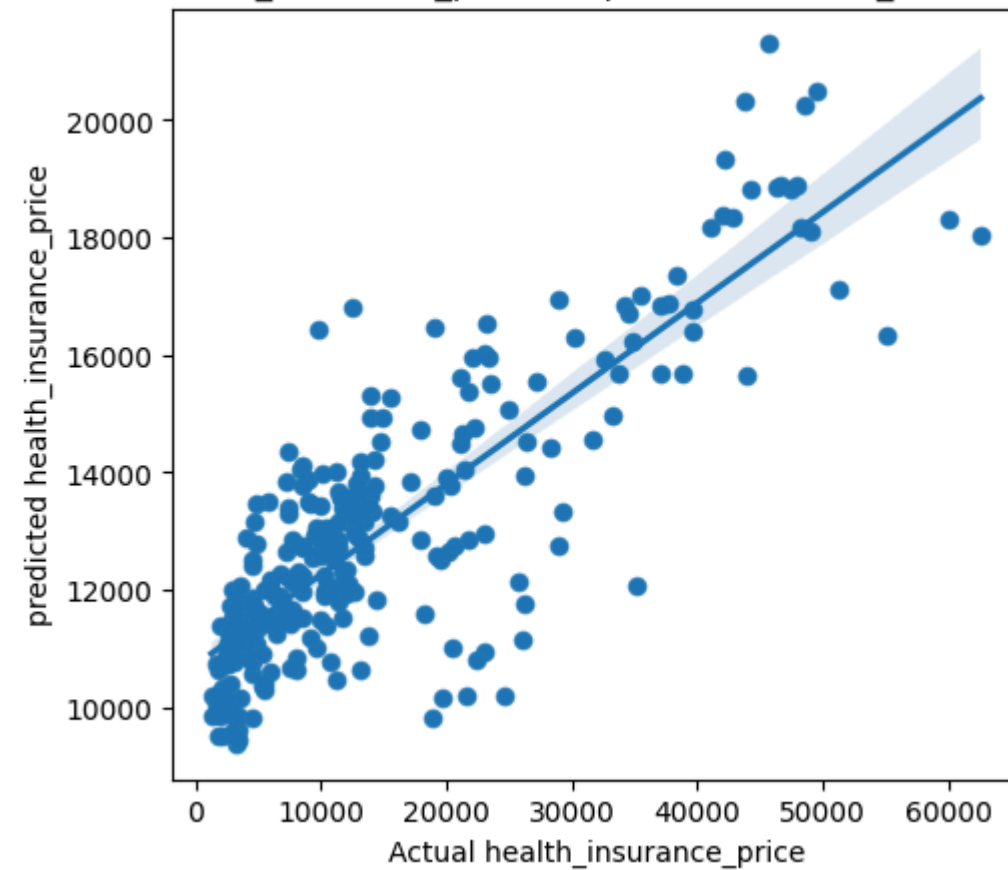
```
1 #Adjusted R^2
2 print("Adjusted R^2:",(1 - (1-xg_reg.score(X_train, y_train))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1)))
```

➦ Adjusted R^2: 0.32622343762623995

```
1 import matplotlib.gridspec as gridspec
2
3 fig = plt.figure(figsize=(12,5))
4 grid = gridspec.GridSpec(ncols=2, nrows=1, figure=fig)
5
6 ax1 = fig.add_subplot(grid[0, 0])
7
8 sns.scatterplot(x = y_test, y = preds, ax=ax1)
9 sns.regplot(x = y_test, y=preds, ax=ax1)
10
11 ax1.set_title("Actual health_insurance_price vs. predicted health_insurance_price")
12 ax1.set_xlabel('Actual health_insurance_price')
13 ax1.set_ylabel('predicted health_insurance_price')
```

```
Text(0, 0.5, 'predicted health_insurance_price')
```

Actual health_insurance_price vs. predicted health_insurance_price



Decision Tree Regressor

```
1 # importing DecisionTreeRegressor from sklearn library
2 from sklearn.tree import DecisionTreeRegressor
3
```

```
1 #creating an object for our model
2 model = DecisionTreeRegressor()
3 model.fit(X_train, y_train)
4 predictions = model.predict(X_test)
```

```
1 # we have to #Compute the rmse by invoking the mean_squared_error function from sklearn's
2 #metrics module.
3 rmse = np.sqrt(metrics.mean_squared_error(y_test, predictions))
4 print("RMSE: %f" % (rmse))
```

```
RMSE: 6221.332700
```

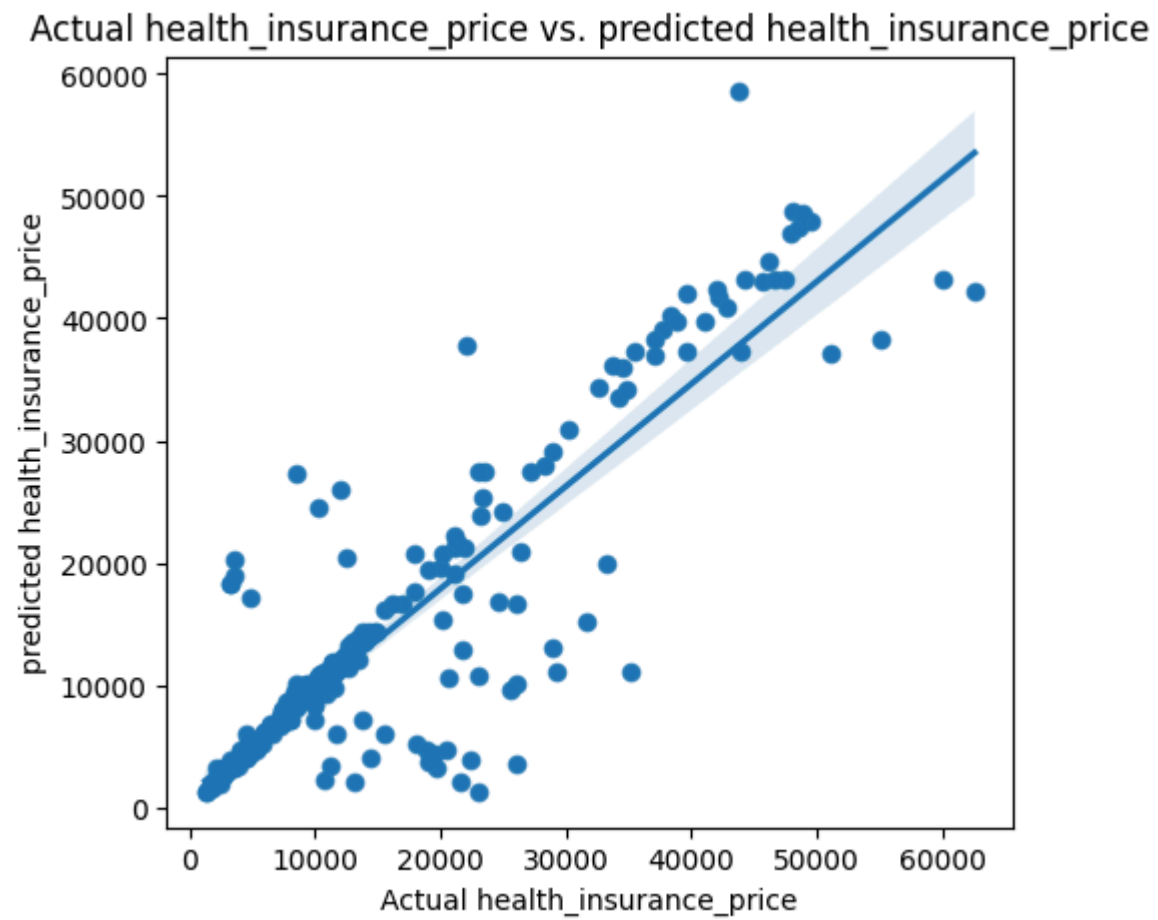
```
1 import matplotlib.gridspec as gridspec
2
3 fig = plt.figure(figsize=(12,5))
4 grid = gridspec.GridSpec(ncols=2, nrows=1, figure=fig)
5
```

```

6 ax1 = fig.add_subplot(grid[0, 0])
7
8 sns.scatterplot(x = y_test, y = predictions, ax=ax1)
9 sns.regplot(x = y_test, y=predictions, ax=ax1)
10
11 ax1.set_title("Actual health_insurance_price vs. predicted health_insurance_price")
12 ax1.set_xlabel('Actual health_insurance_price')
13 ax1.set_ylabel('predicted health_insurance_price')

```

↗ Text(0, 0.5, 'predicted health_insurance_price')



```

1 from sklearn import tree
2 #model = tree.DecisionTreeRegressor(random_state=44)
3 #model.fit(X_train, y_train)
4 #tree.plot_tree(model)

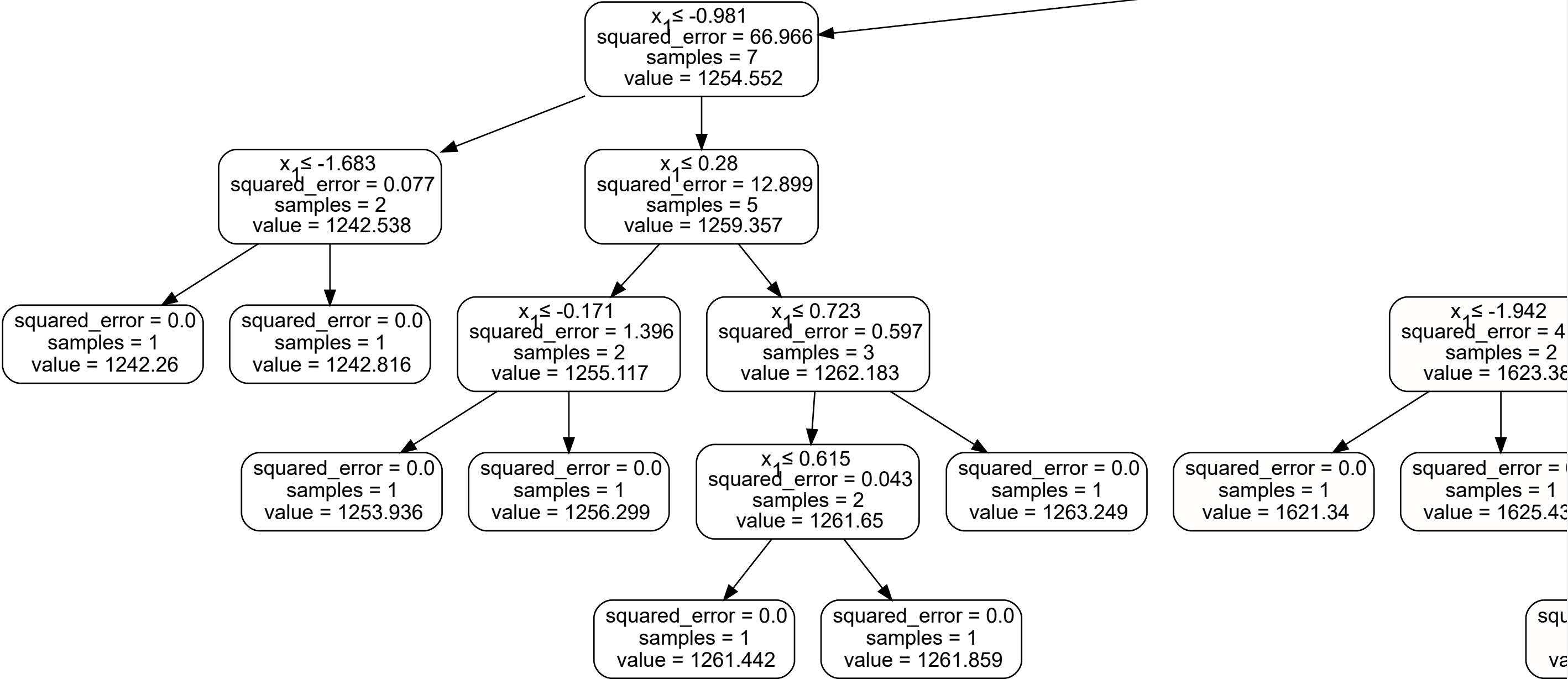
```

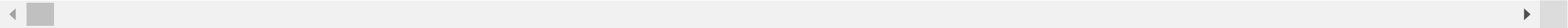
```

1 #adding graphics to the tree
2 import graphviz
3 dot_data = tree.export_graphviz(model, out_file=None, filled=True, rounded=True, special_characters=True)
4 graph = graphviz.Source(dot_data)
5 graph

```





It is splitting till the leaf node hence a big tree but we can tune it later.

```
1 model.score(X_train,y_train) #R^2 close to 1
```

→ 0.9981942449980736

```
1 model.score(X_test,y_test)
```

```
2
```

→ 0.7729067994499237

Apparently, our model is overfitting the training data. therefore, we need to initialize hyperparameters to our model to avoid letting it split till the leafnode.

We got almost 100% score on training data.

On test data we got 77% score.

That's why we are getting high score on our training data and less score on test data.

Let's perform hyperparameter tuning to address this issue.

There are ample tuning methods but we are going to use GridSearch or RandomizedSearch for hyper parameters tuning.

```
1 # Hyper parameters range intialization for tuning
2
3 parameters={"splitter":["best","random"],
4             "max_depth" : [1,3,5,7,9,11,12],
5             "min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
6             "min_weight_fraction_leaf":[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
7             "max_features":["auto","log2","sqrt",None],
8             "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90] }
```

```
1 # importing gridsearchcv
2
3 #from sklearn.model_selection import GridSearchCV
4
```

```
1 #tuning_model=GridSearchCV(model,param_grid=parameters,scoring='neg_mean_squared_error',cv=5,verbose=3)
```

```
1 #tuning_model.fit(X_train,y_train) # it takes a lot of time to train hence I am avoiding this and using randomised searchcv
```

```
1 # best hyperparameters
2 #tuning_model.best_params_
```

```
1 #tuned_hyper_model= DecisionTreeRegressor(max_depth=5,max_features='auto',max_leaf_nodes=50,min_samples_leaf=2,min_weight_fraction_leaf=0.1,splitter='random')
```

```
1 # I am choosing RandomizedSearch CV because of it's perks which we have gone through during your classes.  
2 from sklearn.model_selection import RandomizedSearchCV
```

```
1 RS = RandomizedSearchCV(estimator=model,param_distributions=parameters,cv=5,n_iter=300,n_jobs=-1, verbose=True, scoring='neg_mean_squared_error')
```

```
1 RS.fit(X_train, y_train)
```

```

➡ Fitting 5 folds for each of 300 candidates, totalling 1500 fits
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:425: FitFailedWarning:
850 fits failed out of a total of 1500.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

```

Below are more details about the failures:

171 fits failed with the following error:

Traceback (most recent call last):

```

File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1145, in wrapper
    estimator._validate_params()
File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 638, in _validate_params
    validate_parameter_constraints(
File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 96, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of DecisionTreeRegressor must be an int in the range [1, inf), a float in the range (0.0, 1.0], a str

```

150 fits failed with the following error:

Traceback (most recent call last):

```

File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1145, in wrapper
    estimator._validate_params()
File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 638, in _validate_params
    validate_parameter_constraints(
File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 96, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'min_weight_fraction_leaf' parameter of DecisionTreeRegressor must be a float in the range [0.0, 0.5]. Got 0.8 instead.

```

110 fits failed with the following error:

Traceback (most recent call last):

```

File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1145, in wrapper
    estimator._validate_params()
File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 638, in _validate_params
    validate_parameter_constraints(
File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 96, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'min_weight_fraction_leaf' parameter of DecisionTreeRegressor must be a float in the range [0.0, 0.5]. Got 0.7 instead.

```

65 fits failed with the following error:

Traceback (most recent call last):

```

File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1145, in wrapper
    estimator._validate_params()
File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 638, in _validate_params
    validate_parameter_constraints(
File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 96, in validate_parameter_constraints
    raise InvalidParameterError(

```

```

1 #let's check the best parameters and the best score
2 print('Best Parameters:', RS.best_params_, end='\n\n')

```

```
3 print('Best Score:',RS.best_score_)
```

```
Best Parameters: {'splitter': 'best', 'min_weight_fraction_leaf': 0.1, 'min_samples_leaf': 4, 'max_leaf_nodes': 70, 'max_features': None, 'max_depth': 9}
```

```
Best Score: -30896961.35437845
```

```
1 tuned_model= DecisionTreeRegressor(max_depth=9,max_features=None,max_leaf_nodes=20,
2                                     min_samples_leaf=4,min_weight_fraction_leaf=0.1,splitter='best')
```

```
1 # fitting model
2
3 tuned_model.fit(X_train,y_train)
4
```

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=9, max_leaf_nodes=20, min_samples_leaf=4,
min_weight_fraction_leaf=0.1)
```

```
File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 96, in validate_parameter_constraints
```

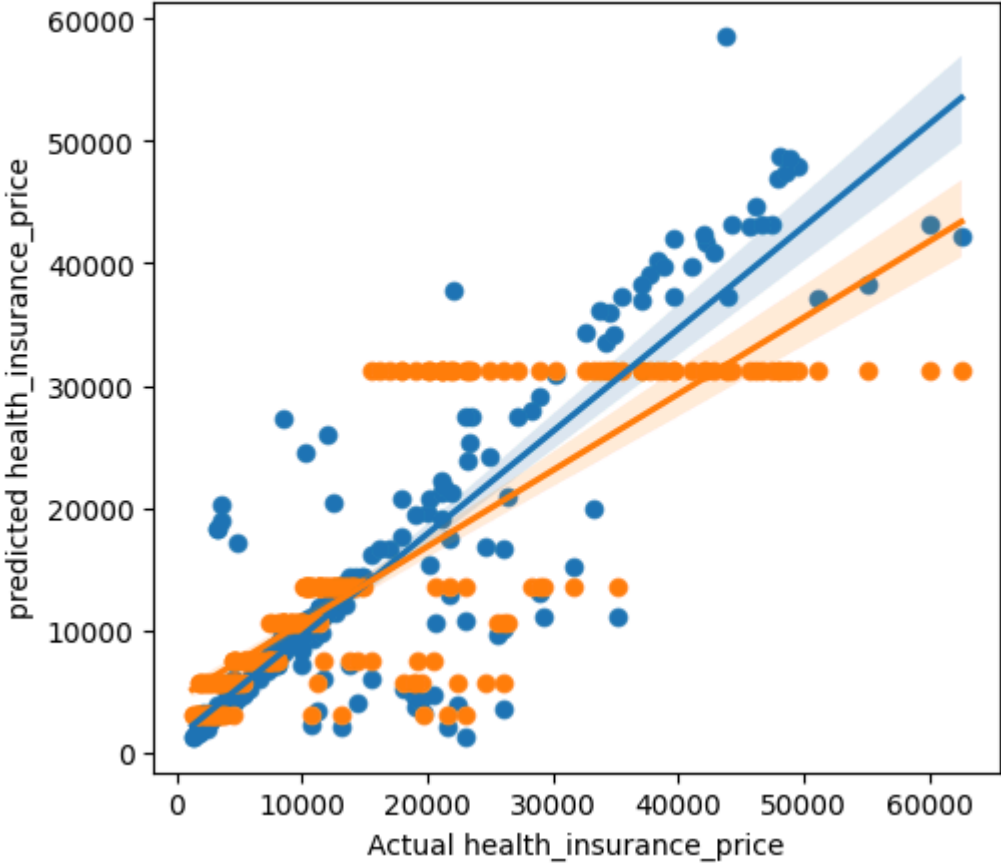
```
1 # prediction
2
3 tuned_pred=tuned_model.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:979: UserWarning: One or more of the test scores are non-finite: [-1.36212386e+08 nan]
```

```
1 #before and after tuning
2
3 fig = plt.figure(figsize=(12,5))
4 grid = gridspec.GridSpec(ncols=2, nrows=1, figure=fig)
5
6 ax1 = fig.add_subplot(grid[0, 0])
7
8
9 sns.scatterplot(x = y_test, y = predictions, ax=ax1)
10 sns.regplot(x = y_test, y=predictions, ax=ax1)
11
12 sns.scatterplot(x = y_test, y = tuned_pred, ax=ax1)
13 sns.regplot(x = y_test, y=tuned_pred, ax=ax1)
14
15 ax1.set_title("Actual health_insurance_price vs. predicted health_insurance_price")
16 ax1.set_xlabel('Actual health_insurance_price')
17 ax1.set_ylabel('predicted health_insurance_price')
```

```
Text(0, 0.5, 'predicted health_insurance_price')
```

Actual health_insurance_price vs. predicted health_insurance_price



-5.23355810e+07 nan nan nan

There is a significant improvement post tuning and looks the model is performing well comparitively

0.704135977046821 0.6556649337242233

```
1 #model's score on training data
2 tuned_model.score(X_train,y_train)
```

```
0.704135977046821
```

0.6556649337242233

```
1 #model's score on test data
2 tuned_model.score(X_test,y_test)
3
```

```
0.6556649337242233
```

DecisionTreeRegressor

You can see the difference between the R^2 values with and without hyperparameter tuning. Earlier, it was overfitting on the training data but it did not overfit post hyperparameter tuning.

Error rate of our model with hyperparameter tuning to our original model which is without the tuning of parameters.


```
1 # With hyperparameter tuning
2 print('MAE:', metrics.mean_absolute_error(y_test,tuned_pred))
3 print('MSE:', metrics.mean_squared_error(y_test, tuned_pred))
4 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, tuned_pred)))
5 print("Adjusted R^2:",(1 - (1-tuned_model.score(X_train, y_train))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1)))
```

↗ MAE: 4860.800670693239
MSE: 58687279.13171234
RMSE: 7660.762307480394
Adjusted R^2: 0.7018903298137797

```
1 # without hyperparameter tuning
2 print('MAE:', metrics.mean_absolute_error(y_test,predictions))
3 print('MSE:', metrics.mean_squared_error(y_test, predictions))
4 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
5 print("Adjusted R^2:",(1 - (1-model.score(X_train, y_train))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1)))
```

↗ MAE: 2971.1217125939847
MSE: 38704980.56948497
RMSE: 6221.332700433643
Adjusted R^2: 0.9981805390777554

If we observe the results, performing hyperparameter tuning did not help us. It means hyperparameter tuning may not fruitful everytime.

```
1 #Visualising the decision tree post tuning
2 import graphviz
3 dot_data = tree.export_graphviz(tuned_model, out_file=None, filled=True, rounded=True, special_characters=True)
4 graph = graphviz.Source(dot_data)
5 graph
```



$x_4 \leq 0.752$
squared error = 136146877.567