Healthcare Insurance Analysis

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df_hos=pd.read_csv("Hospitalisation details.csv")
df_hos
```

Out[2]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID	Age
0	Id2335	1992	Jul	9	0	563.84	tier - 2	tier - 3	R1013	31
1	Id2334	1992	Nov	30	0	570.62	tier - 2	tier - 1	R1013	31
2	Id2333	1993	Jun	30	0	600.00	tier - 2	tier - 1	R1013	30
3	Id2332	1992	Sep	13	0	604.54	tier - 3	tier - 3	R1013	31
4	Id2331	1998	Jul	27	0	637.26	tier - 3	tier - 3	R1013	25
...	...	...	...	...	...	...	...	...	...	...
2338	Id5	1989	Jun	19	0	55135.40	tier - 1	tier - 2	R1012	34
2339	Id4	1991	Jun	6	1	58571.07	tier - 1	tier - 3	R1024	32
2340	Id3	1970	?	11	3	60021.40	tier - 1	tier - 1	R1012	53
2341	Id2	1977	Jun	8	0	62592.87	tier - 2	tier - 3	R1013	46
2342	Id1	1968	Oct	12	0	63770.43	tier - 1	tier - 3	R1013	55

2343 rows × 10 columns

In [3]:
```
df_med= pd.read_csv("Medical Examinations.csv")
df_med
```

Out[3]:

	Customer ID	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker
0	Id1	47.410	7.47	No	No	No	No major surgery	yes
1	Id2	30.360	5.77	No	No	No	No major surgery	yes
2	Id3	34.485	11.87	Yes	No	No	2	yes
3	Id4	38.095	6.05	No	No	No	No major surgery	yes
4	Id5	35.530	5.45	No	No	No	No major surgery	yes
...	...	...	...	...	...	...	...	...
2330	Id2331	22.340	5.57	No	No	No	1	No
2331	Id2332	17.700	6.28	No	No	No	1	No
2332	Id2333	16.470	6.35	No	No	Yes	1	No
2333	Id2334	17.600	4.39	No	No	No	1	No

2334	Id2335	17.580	4.51	No	No	No		1	No

2335 rows × 8 columns

In [4]:
```
df_nam= pd.read_excel("Names.xlsx")
df_nam
```

Out[4]:

	Customer ID	name	Gender
0	Id1	Hawks, Ms. Kelly	Female
1	Id2	Lehner, Mr. Matthew D	Male
2	Id3	Lu, Mr. Phil	Male
3	Id4	Osborne, Ms. Kelsey	Female
4	Id5	Kadala, Ms. Kristyn	Female
...	...	...	...
2330	Id2331	Brietzke, Mr. Jordan	Male
2331	Id2332	Riveros Gonzalez, Mr. Juan D. Sr.	Male
2332	Id2333	Albano, Ms. Julie	Female
2333	Id2334	Rosendahl, Mr. Evan P	Male
2334	Id2335	German, Mr. Aaron K	Male

2335 rows × 3 columns

Collate the files so that all the information is in one place

In [5]:
```
df_nam_med= df_nam.merge(df_med, on="Customer ID")
df_nam_med.head()
```

Out[5]:

	Customer ID	name	Gender	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker
0	Id1	Hawks, Ms. Kelly	Female	47.410	7.47	No	No	No	No major surgery	yes
1	Id2	Lehner, Mr. Matthew D	Male	30.360	5.77	No	No	No	No major surgery	yes
2	Id3	Lu, Mr. Phil	Male	34.485	11.87	Yes	No	No	2	yes
3	Id4	Osborne, Ms. Kelsey	Female	38.095	6.05	No	No	No	No major surgery	yes
4	Id5	Kadala, Ms. Kristyn	Female	35.530	5.45	No	No	No	No major surgery	yes

In [6]:
```
df= df_nam_med.merge(df_hos, on="Customer ID")
df.head()
```

Out[6]:

	Customer ID	name	Gender	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker

0	Id1	Hawks, Ms. Kelly	Female	47.410	7.47	No	No	No	No major surgery	yes
1	Id2	Lehner, Mr. Matthew D	Male	30.360	5.77	No	No	No	No major surgery	yes
2	Id3	Lu, Mr. Phil	Male	34.485	11.87	Yes	No	No	2	yes
3	Id4	Osborne, Ms. Kelsey	Female	38.095	6.05	No	No	No	No major surgery	yes
4	Id5	Kadala, Ms. Kristyn	Female	35.530	5.45	No	No	No	No major surgery	yes

In [7]:
```
df.shape
```

Out[7]:
```
(2335, 19)
```

Check for missing values in the dataset

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Customer ID                0
name                       0
Gender                     0
BMI                        0
HBA1C                      0
Heart Issues               0
Any Transplants            0
Cancer history             0
NumberOfMajorSurgeries     0
smoker                     0
year                       0
month                      0
date                       0
children                   0
charges                    0
Hospital tier              0
City tier                  0
State ID                   0
Age                        0
dtype: int64
```

Find the rows that have trivial value (for example, and delete such rows if they do not contain significant information

In [9]:
```
print('No. of observations with ? value= ',(df['smoker']=="?").sum())
```

```
No. of observations with ? value=  2
```

In [10]:
```
print('No. of observations with ? value= ',(df['year']=="?").sum())
```

```
No. of observations with ? value=  2
```

In [11]:
```
print('No. of observations with ? value= ',(df['month']=="?").sum())
```

```
No. of observations with ? value=  3
```

In [12]:
```
print('No. of observations with ? value= ',(df['Hospital tier']=="?").sum())
```

```
No. of observations with ? value=  1
```

In [13]:
```
print('No. of observations with ? value= ',(df['City tier']=="?").sum())
```
```
No. of observations with ? value=  1
```

In [14]:
```
print('No. of observations with ? value= ',(df['State ID']=="?").sum())
```
```
No. of observations with ? value=  2
```

In [15]:
```
print('No. of observations with ? value= ',(df['Age']=="?").sum())
```
```
No. of observations with ? value=  2
```

In [16]:
```
df=df.drop(df[df['smoker']=="?"].index).reset_index(drop=True)
```

In [17]:
```
print('No. of observations with ? value= ',(df['smoker']=="?").sum())
```
```
No. of observations with ? value=  0
```

In [18]:
```
df=df.drop(df[df['year']=="?"].index).reset_index(drop=True)
```

In [19]:
```
print('No. of observations with ? value= ',(df['year']=="?").sum())
```
```
No. of observations with ? value=  0
```

In [20]:
```
df=df.drop(df[df['month']=="?"].index).reset_index(drop=True)
```

In [21]:
```
print('No. of observations with ? value= ',(df['month']=="?").sum())
```
```
No. of observations with ? value=  0
```

In [22]:
```
df=df.drop(df[df['City tier']=="?"].index).reset_index(drop=True)
```

In [23]:
```
print('No. of observations with ? value= ',(df['City tier']=="?").sum())
```
```
No. of observations with ? value=  0
```

In [24]:
```
df=df.drop(df[df['State ID']=="?"].index).reset_index(drop=True)
```

In [25]:
```
print('No. of observations with ? value= ',(df['State ID']=="?").sum())
```
```
No. of observations with ? value=  0
```

In [26]:
```
df=df.drop(df[df['Hospital tier']=="?"].index).reset_index(drop=True)
```

In [27]:
```
print('No. of observations with ? value= ',(df['Hospital tier']=="?").sum())
```
```
No. of observations with ? value=  0
```

In [28]:
```
df=df.drop(df[df['Age']=="?"].index).reset_index(drop=True)
```

In [29]:
```
print('No. of observations with ? value= ',(df['Age']=="?").sum())
```
```
No. of observations with ? value=  0
```

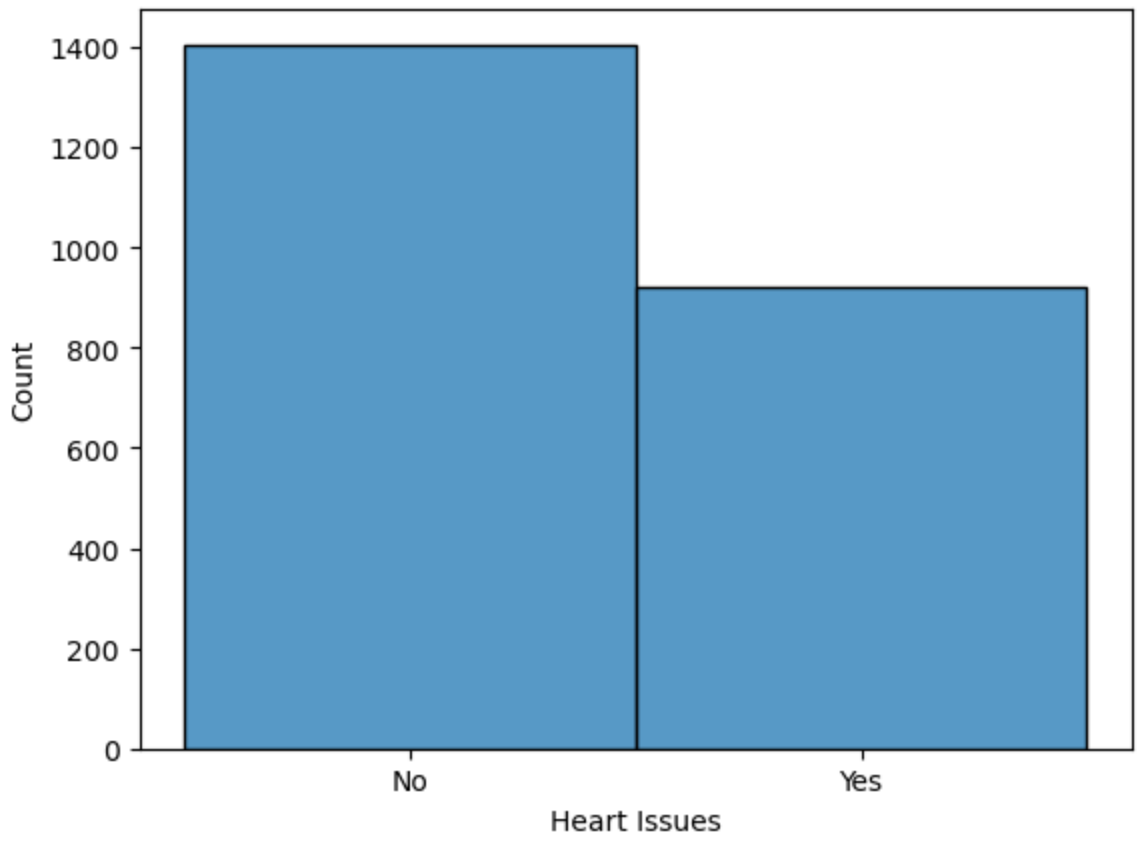
In [30]:
```
df.shape
```
Out[30]:
```
(2325, 19)
```

In [31]:
```
df['Heart Issues'].value_counts()
```
Out[31]:
```
No     1405
Yes     920
```

```
Name: Heart Issues, dtype: int64
```

In [32]:
```
sns.histplot(df['Heart Issues'])
```

Out[32]:
```
<AxesSubplot:xlabel='Heart Issues', ylabel='Count'>
```

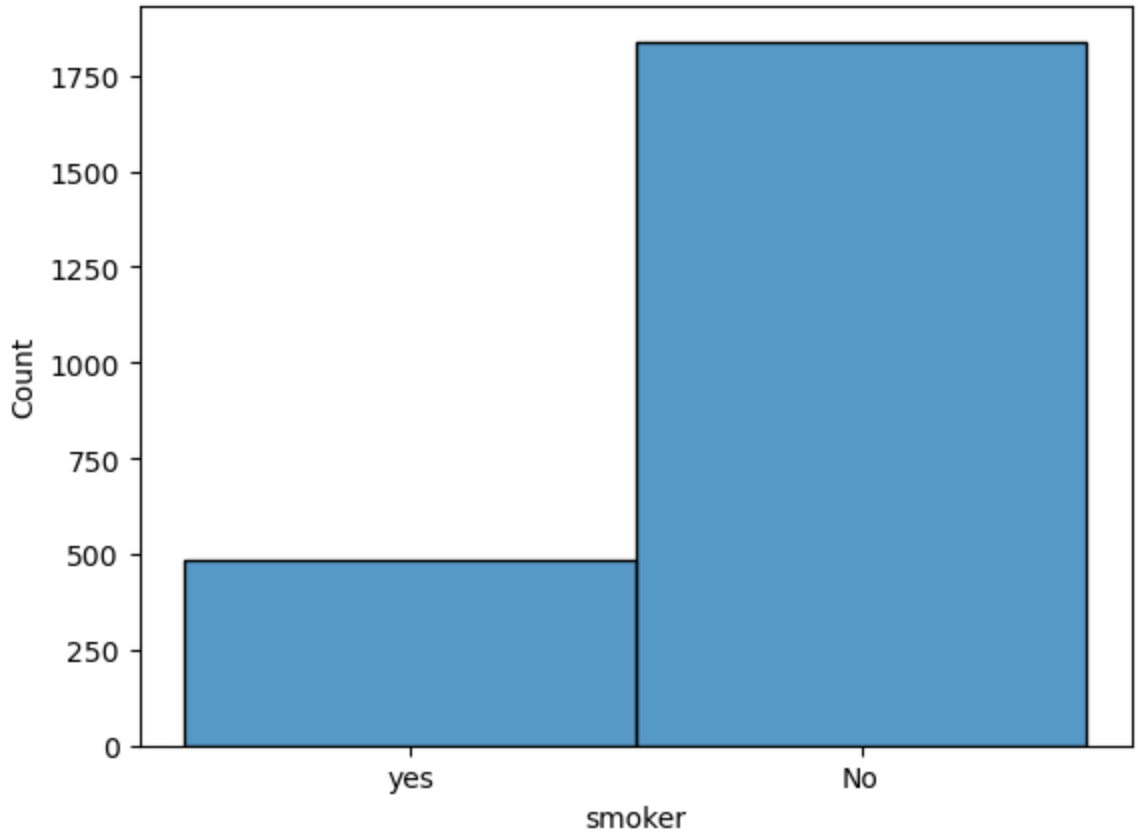


In [33]:
```
df['smoker'].value_counts()
```

Out[33]:
```
No     1839
yes     486
Name: smoker, dtype: int64
```

In [34]:
```
sns.histplot(df['smoker'])
```

Out[34]:
```
<AxesSubplot:xlabel='smoker', ylabel='Count'>
```

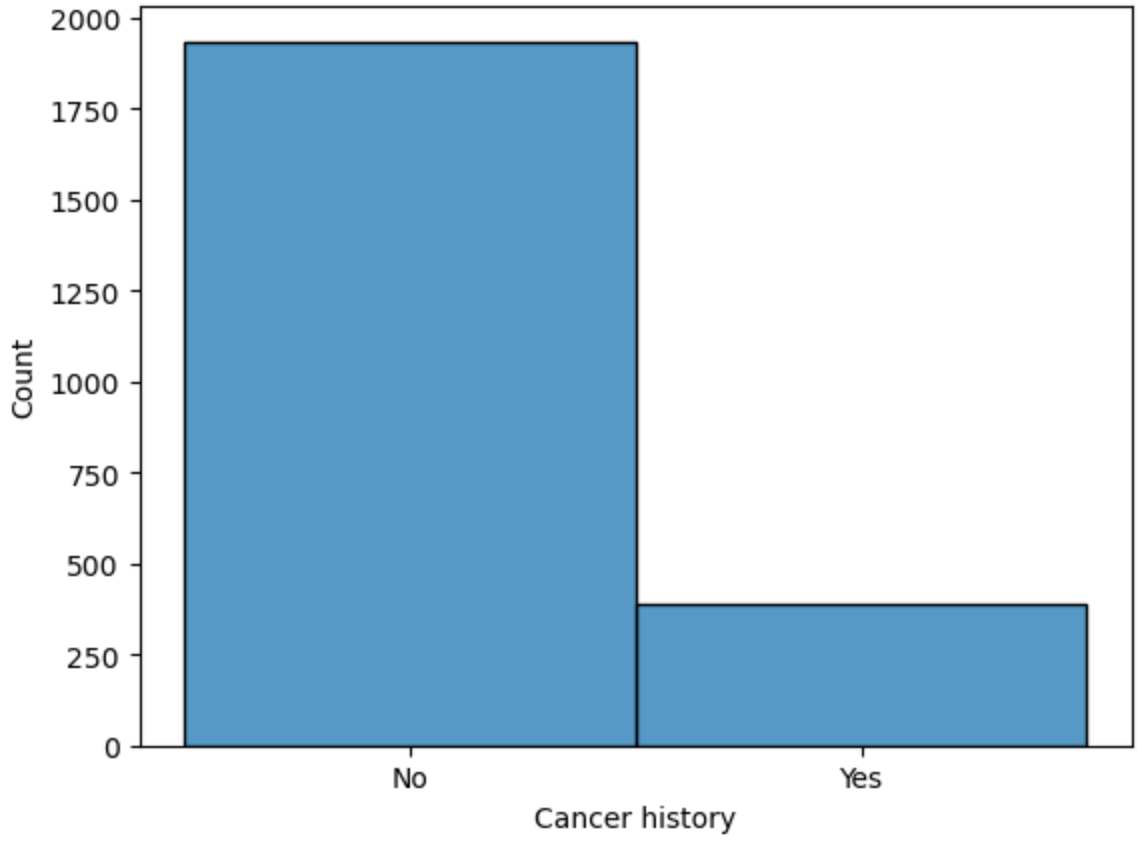


In [35]:
```
df['Cancer history'].value_counts()
```

Out[35]:
```
No     1934
Yes     391
Name: Cancer history, dtype: int64
```

In [36]:
```
sns.histplot(df['Cancer history'])
```

Out[36]:
```
<AxesSubplot:xlabel='Cancer history', ylabel='Count'>
```

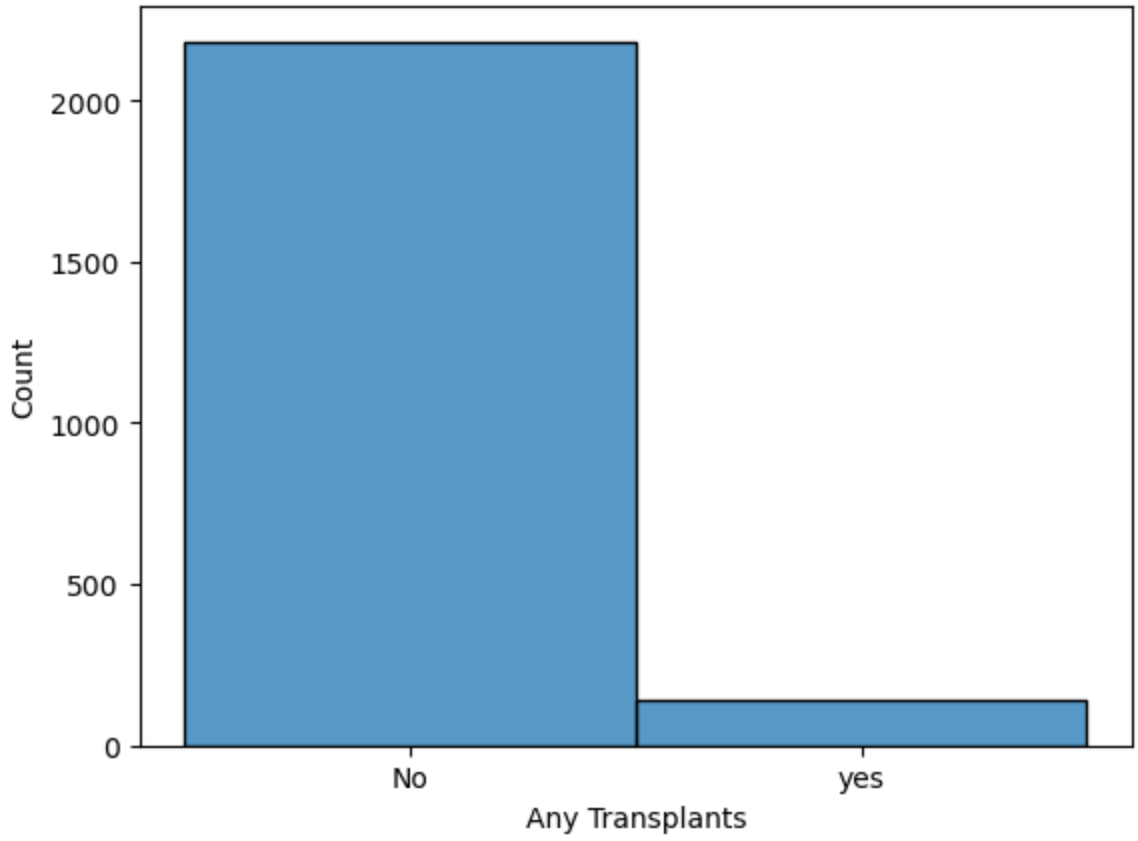


In [37]:
```
df['Any Transplants'].value_counts()
```

Out[37]:
```
No     2183
yes     142
Name: Any Transplants, dtype: int64
```

In [38]:
```
sns.histplot(df['Any Transplants'])
```

Out[38]:
```
<AxesSubplot:xlabel='Any Transplants', ylabel='Count'>
```



In [39]:
```
df.dtypes
```

Out[39]:
```
Customer ID                object
name                       object
Gender                     object
BMI                       float64
HBA1C                     float64
Heart Issues               object
Any Transplants            object
Cancer history             object
NumberOfMajorSurgeries     object
smoker                     object
year                       object
month                      object
date                        int64
children                    int64
charges                   float64
Hospital tier              object
City tier                  object
State ID                   object
Age                        object
dtype: object
```

Use the necessary transformation methods to deal with the nominal and ordinal categorical variables in the dataset

In [40]:
```
df['Gender'] = df['Gender'].apply({'Male':1, 'Female':0}.get)
df['Any Transplants'] = df['Any Transplants'].apply({'yes':1, 'No':0}.get)
```

```
df['Cancer history'] = df['Cancer history'].apply({'Yes':1, 'No':0}.get)
df['smoker'] = df['smoker'].apply({'yes':1, 'No':0}.get)
df['Hospital tier'] = df['Hospital tier'].apply({'tier - 1':1, 'tier - 2':2, 'tier - 3':
df['City tier'] = df['City tier'].apply({'tier - 1':1, 'tier - 2':2, 'tier - 3':3}.get)
df['Heart Issues'] = df['Heart Issues'].apply({'Yes':1, 'No':0}.get)
df['State ID'] = df['State ID'].apply({'R1011':1, 'R1012':2, 'R1013':3, 'R1014':4, 'R101
                                       'R1018':8, 'R1019':9, 'R1020':10, 'R1021':11, 'R1
                                      'R1024':14, 'R1025':15, 'R1026':16}.get)
df['month'] = df['month'].apply({'Jun':6, 'Jul':7, 'Aug':8, 'Sep':9, 'Oct':10, 'Nov':11,
```

In [41]:
```
df['year']=df['year'].astype('int')
```

In [42]:
```
df['Age']=df['Age'].astype('int')
```

In [43]:
```
df.dtypes
```

Out[43]:
```
Customer ID                object
name                       object
Gender                      int64
BMI                       float64
HBA1C                     float64
Heart Issues                int64
Any Transplants             int64
Cancer history              int64
NumberOfMajorSurgeries     object
smoker                      int64
year                        int32
month                       int64
date                        int64
children                    int64
charges                   float64
Hospital tier               int64
City tier                   int64
State ID                    int64
Age                         int32
dtype: object
```

The variable NumberOfMajorSurgeries also appears to have string values. Apply a suitable method to clean up this variable.

In [44]:
```
df['NumberOfMajorSurgeries'].value_counts()
```

Out[44]:
```
No major surgery    1070
1                    961
2                    272
3                     22
Name: NumberOfMajorSurgeries, dtype: int64
```

In [45]:
```
df['NumberOfMajorSurgeries']=df['NumberOfMajorSurgeries'].replace('No major surgery',0)
```

In [46]:
```
df['NumberOfMajorSurgeries'].value_counts()
```

Out[46]:
```
0    1070
1     961
2     272
3      22
Name: NumberOfMajorSurgeries, dtype: int64
```

In [47]:
```
df['NumberOfMajorSurgeries']=df['NumberOfMajorSurgeries'].astype('int')
```

In [48]:
```
df.dtypes
```

Out[48]:
```
Customer ID                object
name                       object
```

```
Gender                      int64
BMI                       float64
HBA1C                     float64
Heart Issues                int64
Any Transplants             int64
Cancer history              int64
NumberOfMajorSurgeries      int32
smoker                      int64
year                        int32
month                       int64
date                        int64
children                    int64
charges                   float64
Hospital tier               int64
City tier                   int64
State ID                    int64
Age                         int32
dtype: object
```

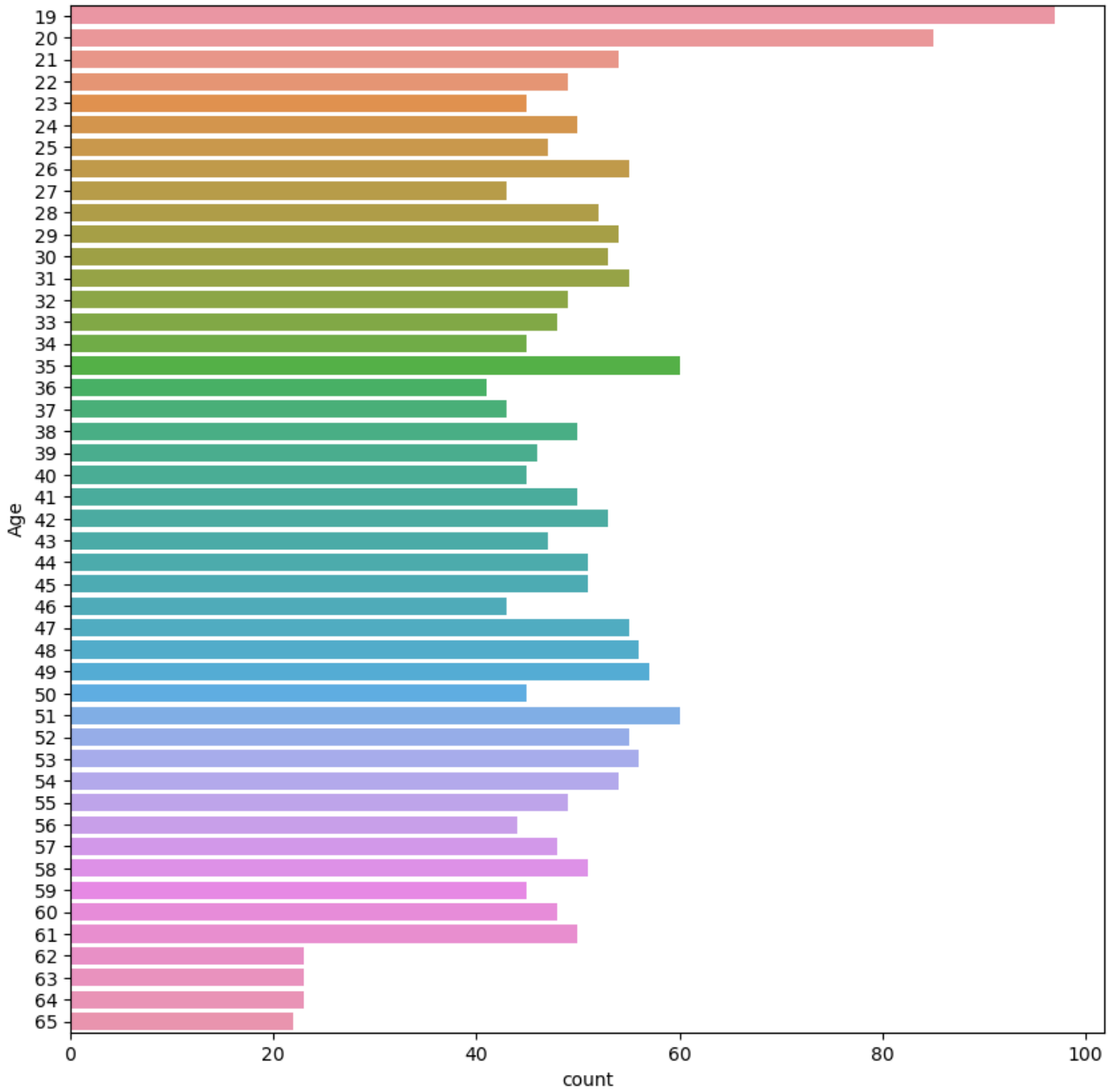
Age appears to be a significant factor in this analysis

In [49]:
```
df['Age'].value_counts().head()
```

Out[49]:
```
19    97
20    85
35    60
51    60
49    57
Name: Age, dtype: int64
```

In [50]:
```
plt.figure(figsize=(10,10))
sns.countplot(y=df['Age'],data=df)
```

Out[50]:
```
<AxesSubplot:xlabel='count', ylabel='Age'>
```



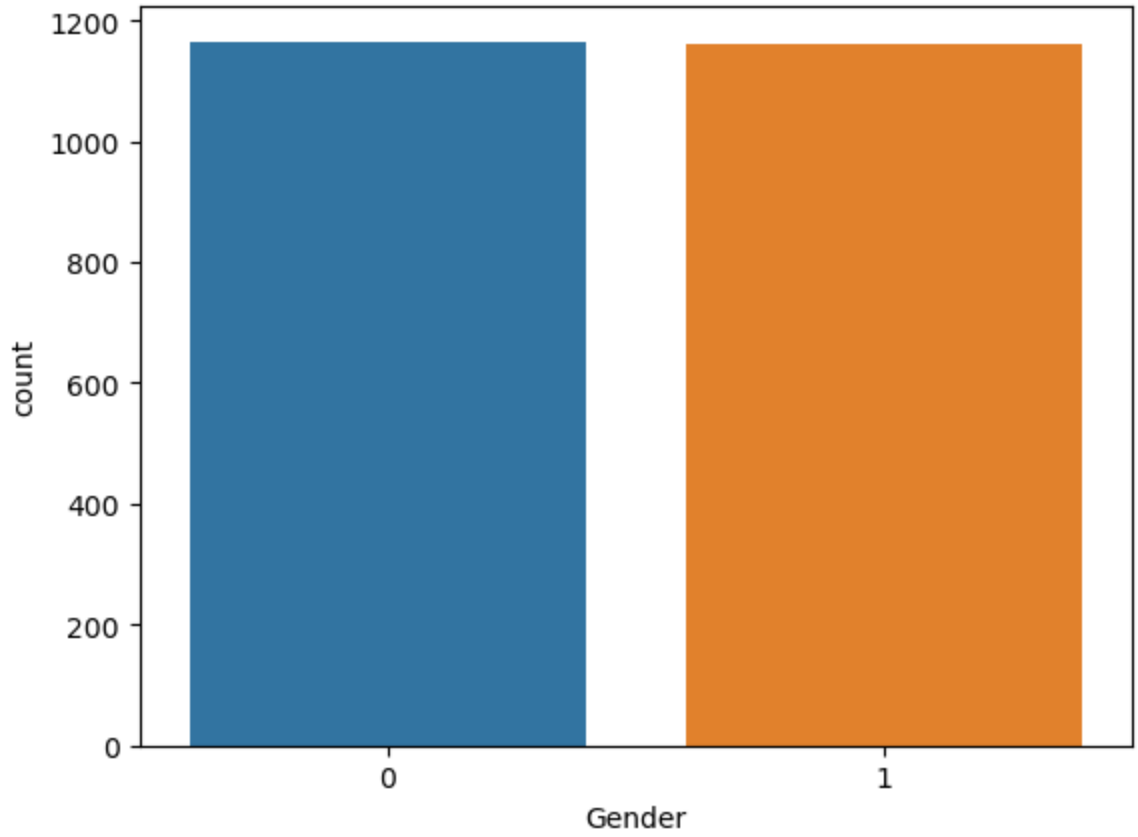
The gender of the patient may be an important factor in determining the cost of hospitalization. The salutations in a beneficiary's name can be used to determine their gender. Make a new field for the beneficiary's gender.

In [51]:
```
df['Gender'].value_counts()
```

Out[51]:
```
0    1165
1    1160
Name: Gender, dtype: int64
```

In [52]:
```
sns.countplot(x=df['Gender'],data=df)
```

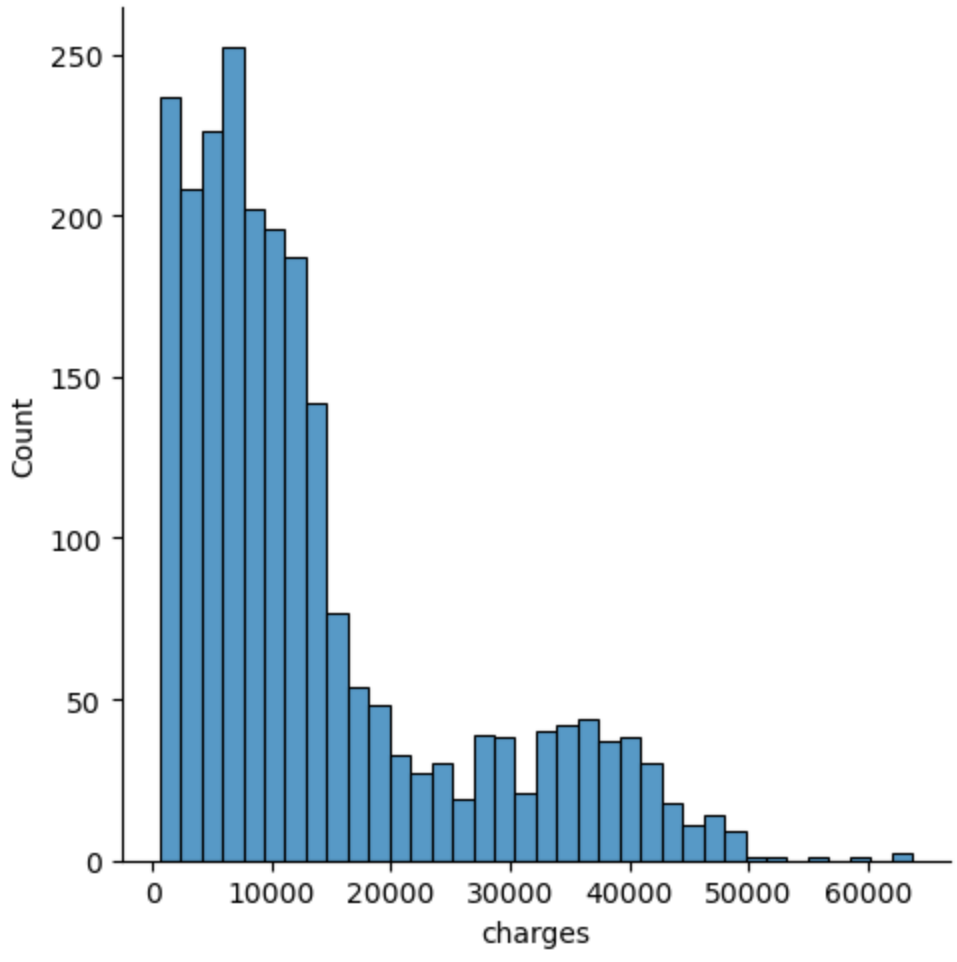
Out[52]:
```
<AxesSubplot:xlabel='Gender', ylabel='count'>
```



0= Female & 1= Male

In [53]:
```
sns.displot(df['charges'])
```

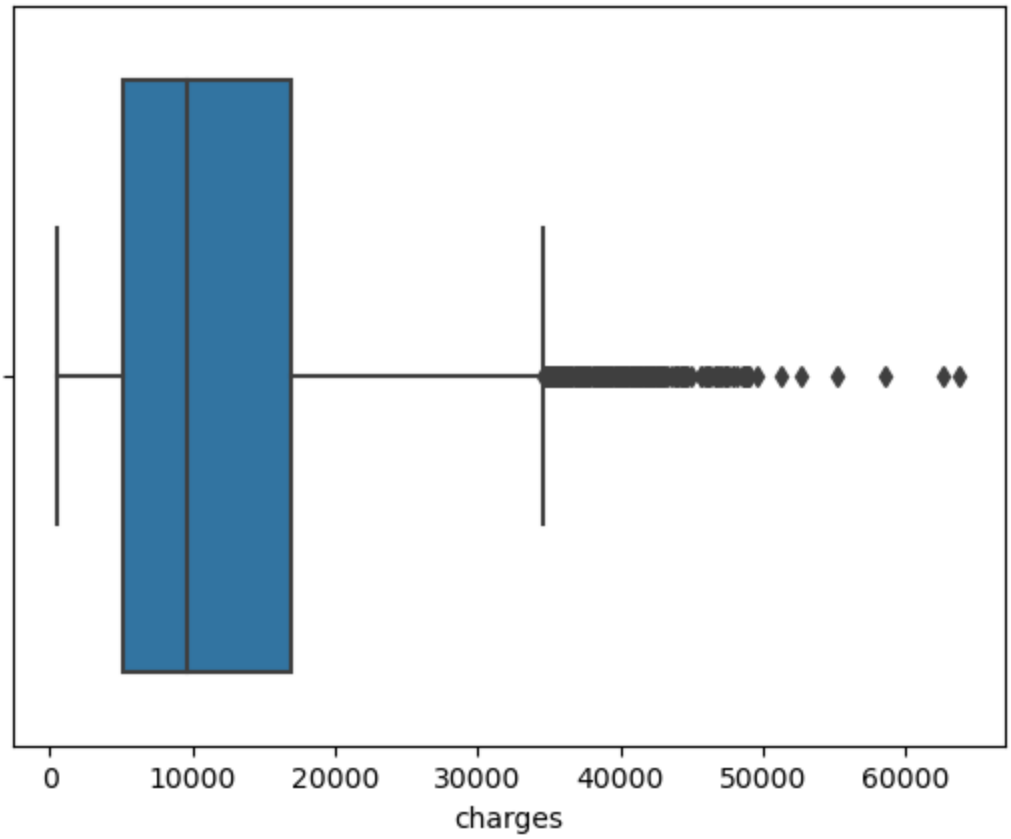
Out[53]:
```
<seaborn.axisgrid.FacetGrid at 0x29230ba8d60>
```



You should also visualize the distribution of costs using a histogram, box and whisker plot, and swarm plot.

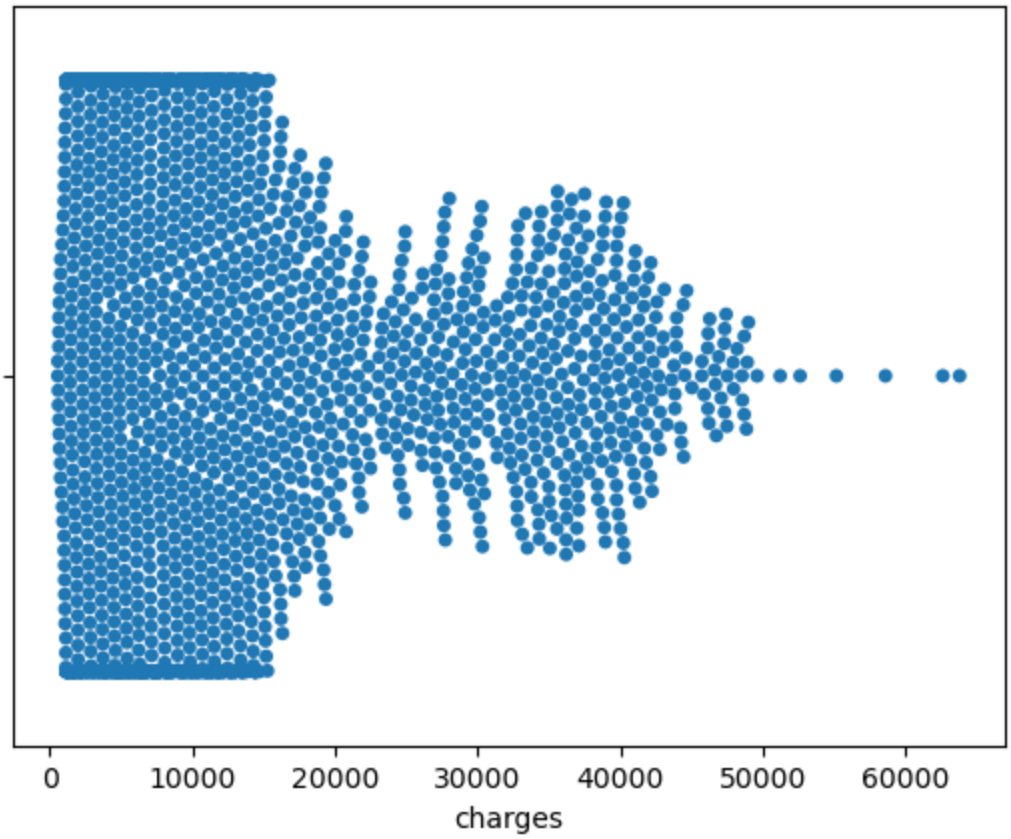
In [54]:
```
sns.boxplot(x=df['charges'],data=df)
```

Out[54]:
```
<AxesSubplot:xlabel='charges'>
```



In [55]:
```
sns.swarmplot(x=df['charges'],data=df)
```

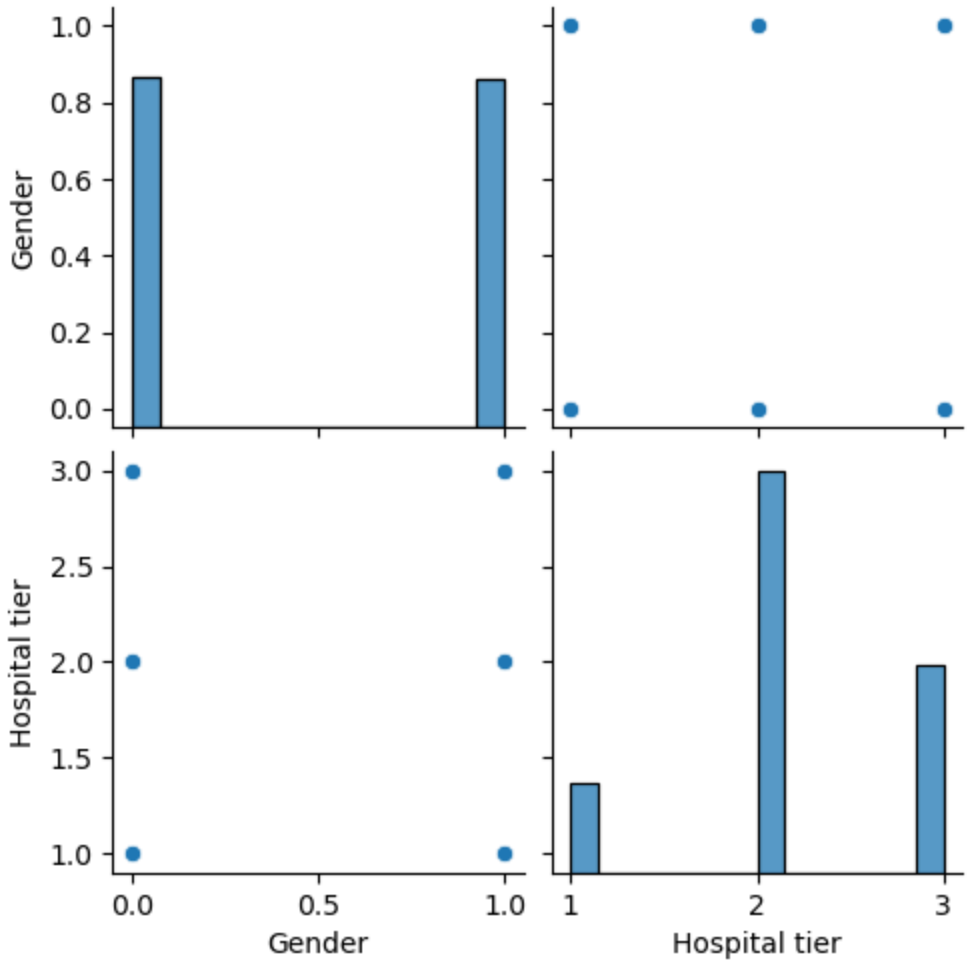
Out[55]:
```
<AxesSubplot:xlabel='charges'>
```



State how the distribution is different across gender and tiers of hospitals

In [56]:
```
sns.pairplot(df[['Gender','Hospital tier']])
```

Out[56]:
```
<seaborn.axisgrid.PairGrid at 0x29230222430>
```



Create a frequency table and a stacked bar chart to visualize the count of people in the different tiers of cities and hospitals

In [57]:
```
crosstab=pd.crosstab(index=df['Hospital tier'], columns=df['City tier'])
```

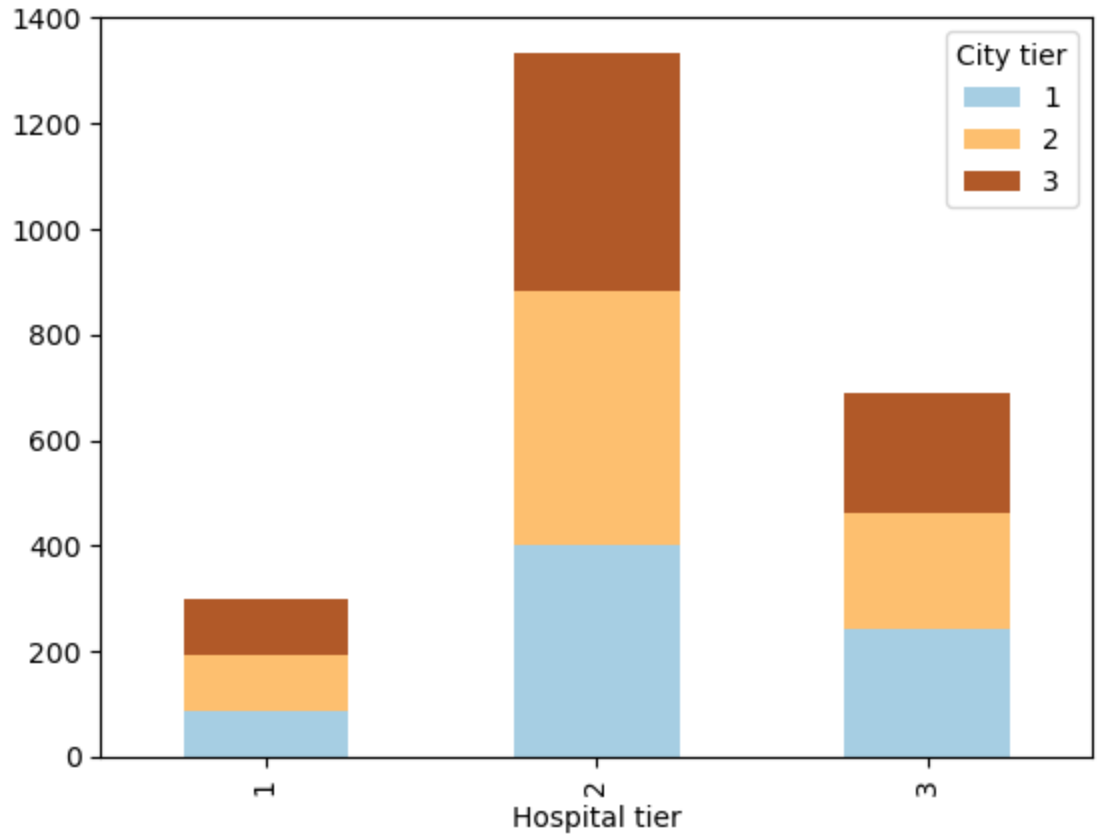
In [58]:
```
crosstab
```

Out[58]:

Hospital tier	City tier 1	2	3
1	85	106	109
2	403	479	452
3	241	222	228

In [59]:
```
crosstab.plot(kind='bar', stacked=True,colormap='Paired')
```

Out[59]:
```
<AxesSubplot:xlabel='Hospital tier'>
```



In [60]:
```
df[['charges','Hospital tier']].describe()
```

Out[60]:

	charges	Hospital tier
count	2325.000000	2325.000000
mean	13521.660254	2.168172
std	11863.492697	0.630972
min	563.840000	1.000000
25%	5116.500000	2.000000
50%	9634.540000	2.000000

75%	16903.500000	3.000000
max	63770.430000	3.000000

The average hospitalization costs for the three types of hospitals are not significantly different

In [61]:
```
from scipy import stats
from scipy.stats import chi2_contingency
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

In [62]:
```
anova_df=pd.DataFrame()
anova_df['Hospital_tier']=df['Hospital tier']
anova_df['charges']=df['charges']
anova_df.dropna(inplace=True)
```

In [63]:
```
anova_df.head()
```

Out[63]:

	Hospital_tier	charges
0	1	63770.43
1	2	62592.87
2	1	58571.07
3	1	55135.40
4	1	52590.83

In [64]:
```
lm=ols('charges~Hospital_tier',data=anova_df).fit()
table=sm.stats.anova_lm(lm)
table
```

Out[64]:

	df	sum_sq	mean_sq	F	PR(>F)
Hospital_tier	1.0	6.526309e+10	6.526309e+10	579.042037	2.013400e-114
Residual	2323.0	2.618224e+11	1.127087e+08	NaN	NaN

Since P value is less than 0.05, thus we accept alternate hypothesis

The average hospitalization costs for the three types of cities are not significantly different

In [65]:
```
df[['charges','City tier']].describe()
```

Out[65]:

	charges	City tier
count	2325.000000	2325.000000
mean	13521.660254	2.025806
std	11863.492697	0.807786
min	563.840000	1.000000
25%	5116.500000	1.000000
50%	9634.540000	2.000000
75%	16903.500000	3.000000

max	63770.430000	3.000000

In [66]:
```
anova_df1=pd.DataFrame()
anova_df1['City_tier']=df['City tier']
anova_df1['charges']=df['charges']
anova_df1.dropna(inplace=True)
```

In [67]:
```
anova_df1.head()
```

Out[67]:

	City_tier	charges
0	3	63770.43
1	3	62592.87
2	3	58571.07
3	2	55135.40
4	3	52590.83

In [68]:
```
lm=ols('charges~City_tier',data=anova_df1).fit()
table=sm.stats.anova_lm(lm)
table
```

Out[68]:

	df	sum_sq	mean_sq	F	PR(>F)
City_tier	1.0	4.075841e+08	4.075841e+08	2.898322	0.088806
Residual	2323.0	3.266779e+11	1.406276e+08	NaN	NaN

Since P value is 0.08 which is greater than 0.05, thus it fail to reject H0 i.e, we accept Null Hypothesis

Smoking and heart issues are independent

In [69]:
```
chi_sq=pd.DataFrame()
chi_sq['smoker']=df['smoker']
chi_sq['Heart_Issues']=df['Heart Issues']
chi_sq.dropna(inplace=True)
```

In [70]:
```
data_crosstab=pd.crosstab(chi_sq['smoker'],chi_sq['Heart_Issues'])
```

In [71]:
```
stat,p,dof,expected=chi2_contingency(data_crosstab)
alpha=0.05
if p<=alpha:
    print('Dependent(reject H0)')
else:
    print('Independent(H0 holds true)')
```

```
Independent(H0 holds true)
```

since p value for the chi square test is greater than 0.05, we can say that heart issue is independent of smoker

Machine Learning

In [72]:
```
df1=df.copy()
```

In [73]:
```
df1.head()
```

Out[73]:

	Customer ID	name	Gender	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker
0	Id1	Hawks, Ms. Kelly	0	47.410	7.47	0	0	0	0	1
1	Id2	Lehner, Mr. Matthew D	1	30.360	5.77	0	0	0	0	1
2	Id4	Osborne, Ms. Kelsey	0	38.095	6.05	0	0	0	0	1
3	Id5	Kadala, Ms. Kristyn	0	35.530	5.45	0	0	0	0	1
4	Id6	Baker, Mr. Russell B.	1	32.800	6.59	0	0	0	0	1

In [74]:
```
df1.drop(columns=['Customer ID', 'name'],axis=1, inplace=True)
```

In [75]:
```
df1.head()
```

Out[75]:

	Gender	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	year	month	date	c
0	0	47.410	7.47	0	0	0	0	1	1968	10	12	
1	1	30.360	5.77	0	0	0	0	1	1977	6	8	
2	0	38.095	6.05	0	0	0	0	1	1991	6	6	
3	0	35.530	5.45	0	0	0	0	1	1989	6	19	
4	1	32.800	6.59	0	0	0	0	1	1962	8	4	

Examine the correlation between predictors to identify highly correlated predictors. Use a heatmap to visualize this.

In [76]:
```
df1.corr()
```

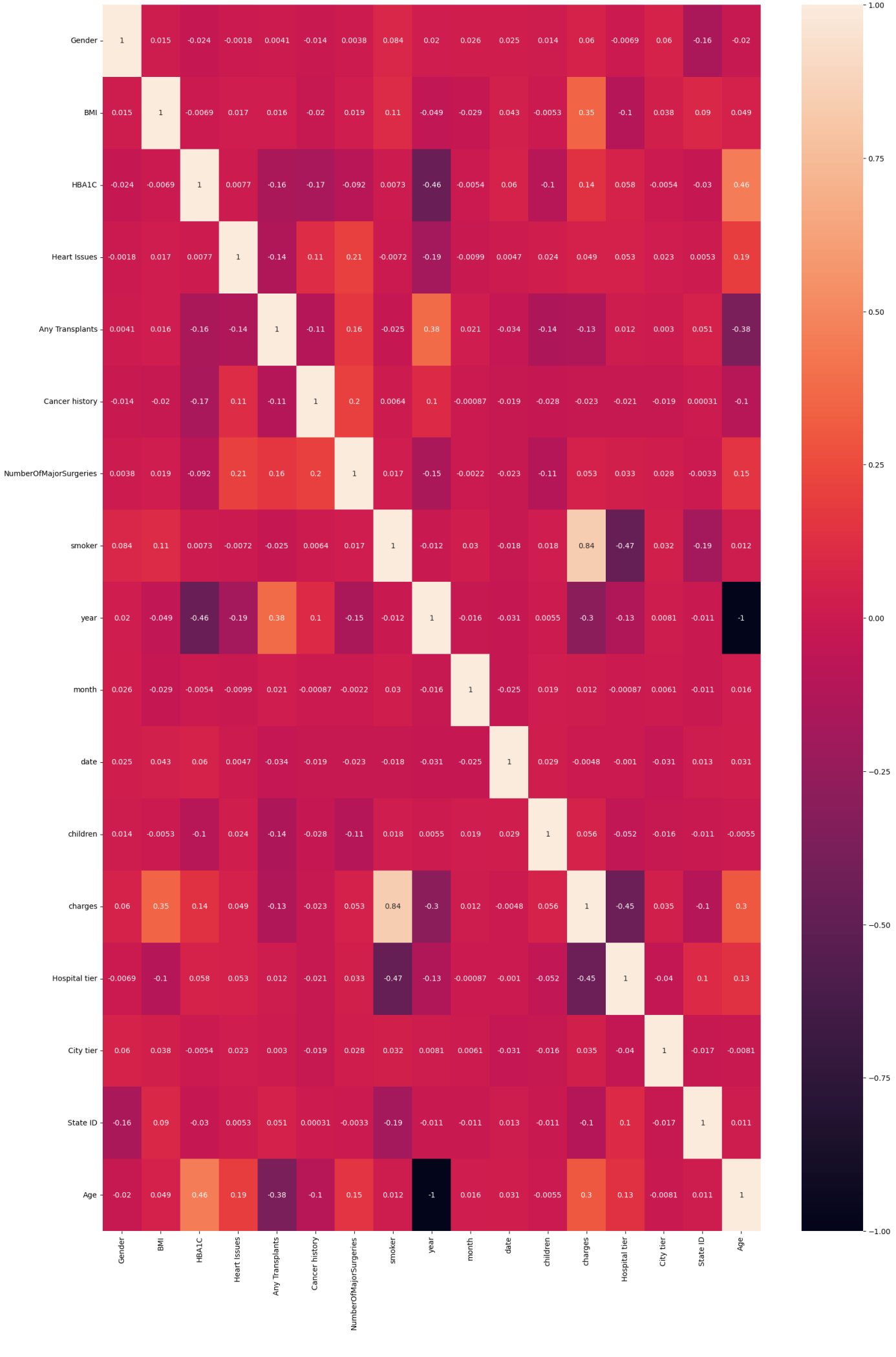
Out[76]:

	Gender	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurge
Gender	1.000000	0.015239	-0.023890	-0.001778	0.004141	-0.013983	0.003
BMI	0.015239	1.000000	-0.006920	0.017129	0.015893	-0.020235	0.018
HBA1C	-0.023890	-0.006920	1.000000	0.007699	-0.159855	-0.170921	-0.091
Heart Issues	-0.001778	0.017129	0.007699	1.000000	-0.140269	0.111190	0.206
Any Transplants	0.004141	0.015893	-0.159855	-0.140269	1.000000	-0.114677	0.158
Cancer history	-0.013983	-0.020235	-0.170921	0.111190	-0.114677	1.000000	0.204
NumberOfMajorSurgeries	0.003842	0.018851	-0.091594	0.206147	0.158593	0.204208	1.000
smoker	0.083612	0.107126	0.007257	-0.007159	-0.025101	0.006415	0.017
year	0.020197	-0.049260	-0.460558	-0.192273	0.381084	0.101073	-0.151

month	0.026380	-0.029179	-0.005368	-0.009940	0.020752	-0.000874	-0.002
date	0.025301	0.042765	0.059789	0.004734	-0.033858	-0.018599	-0.022
children	0.014332	-0.005339	-0.101379	0.023984	-0.142040	-0.027880	-0.113
charges	0.060156	0.346730	0.139697	0.049299	-0.127028	-0.022522	0.053
Hospital tier	-0.006927	-0.104771	0.057855	0.053376	0.011729	-0.021429	0.033
City tier	0.059716	0.038123	-0.005404	0.023152	0.002970	-0.018639	0.027
State ID	-0.163290	0.090484	-0.029681	0.005282	0.051225	0.000306	-0.003
Age	-0.020197	0.049260	0.460558	0.192273	-0.381084	-0.101073	0.151

In [77]:
```
plt.figure(figsize=(20,30))
sns.heatmap(df1.corr(),annot=True)
```

Out[77]:
```
<AxesSubplot:>
```


Gender 1 0.015 -0.024 -0.0018 0.0041 -0.014 0.0038 0.084 0.02 0.026 0.025 0.014 0.06 -0.0069 0.06 -0.16 -0.02
BMI 0.015 1 -0.0069 0.017 0.016 -0.02 0.019 0.11 -0.049 -0.029 0.043 -0.0053 0.35 -0.1 0.038 0.09 0.049
HBA1C -0.024 -0.0069 1 0.0077 -0.16 -0.17 -0.092 0.0073 -0.46 -0.0054 0.06 -0.1 0.14 0.058 -0.0054 -0.03 0.46
Heart Issues -0.0018 0.017 0.0077 1 -0.14 0.11 0.21 -0.0072 -0.19 -0.0099 0.0047 0.024 0.049 0.053 0.023 0.0053 0.19
Any Transplants 0.0041 0.016 -0.16 -0.14 1 -0.11 0.16 -0.025 0.38 0.021 -0.034 -0.14 -0.13 0.012 0.003 0.051 -0.38
Cancer history -0.014 -0.02 -0.17 0.11 -0.11 1 0.2 0.0064 0.1 -0.00087 -0.019 -0.028 -0.023 -0.021 -0.019 0.00031 -0.1
NumberOfMajorSurgeries 0.0038 0.019 -0.092 0.21 0.16 0.2 1 0.017 -0.15 -0.0022 -0.023 -0.11 0.053 0.033 0.028 -0.0033 0.15
smoker 0.084 0.11 0.0073 -0.0072 -0.025 0.0064 0.017 1 -0.012 0.03 -0.018 0.018 0.84 -0.47 0.032 -0.19 0.012
year 0.02 -0.049 -0.46 -0.19 0.38 0.1 -0.15 -0.012 1 -0.016 -0.031 0.0055 -0.3 -0.13 0.0081 -0.011 -1
month 0.026 -0.029 -0.0054 -0.0099 0.021 -0.00087 -0.0022 0.03 -0.016 1 -0.025 0.019 0.012 -0.00087 0.0061 -0.011 0.016
date 0.025 0.043 0.06 0.0047 -0.034 -0.019 -0.023 -0.018 -0.031 -0.025 1 0.029 -0.0048 -0.001 -0.031 0.013 0.031
children 0.014 -0.0053 -0.1 0.024 -0.14 -0.028 -0.11 0.018 0.0055 0.019 0.029 1 0.056 -0.052 -0.016 -0.011 -0.0055
charges 0.06 0.35 0.14 0.049 -0.13 -0.023 0.053 0.84 -0.3 0.012 -0.0048 0.056 1 -0.45 0.035 -0.1 0.3
Hospital tier -0.0069 -0.1 0.058 0.053 0.012 -0.021 0.033 -0.47 -0.13 -0.00087 -0.001 -0.052 -0.45 1 -0.04 0.1 0.13
City tier 0.06 0.038 -0.0054 0.023 0.003 -0.019 0.028 0.032 0.0081 0.0061 -0.031 -0.016 0.035 -0.04 1 -0.017 -0.0081
State ID -0.16 0.09 -0.03 0.0053 0.051 0.00031 -0.0033 -0.19 -0.011 -0.011 0.013 -0.011 -0.1 0.1 -0.017 1 0.011
Age -0.02 0.049 0.46 0.19 -0.38 -0.1 0.15 0.012 -1 0.016 0.031 -0.0055 0.3 0.13 -0.0081 0.011 1
Gender BMI HBA1C Heart Issues Any Transplants Cancer history NumberOfMajorSurgeries smoker year month date children charges Hospital tier City tier State ID Age
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00

Develop and evaluate the final model using regression with a stochastic gradient descent optimizer.

In [78]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_variance_
from tabulate import tabulate
```

In [79]:
```
X = df1.drop(['charges'], axis=1)
Y = df1['charges']
```

Standardizing the features

In [80]:
```
X=StandardScaler().fit_transform(X)
```

In [81]:
```
#Splitting the data to test and train
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

Let us calculate R-Squared, SGD Regressor Model Accuracy

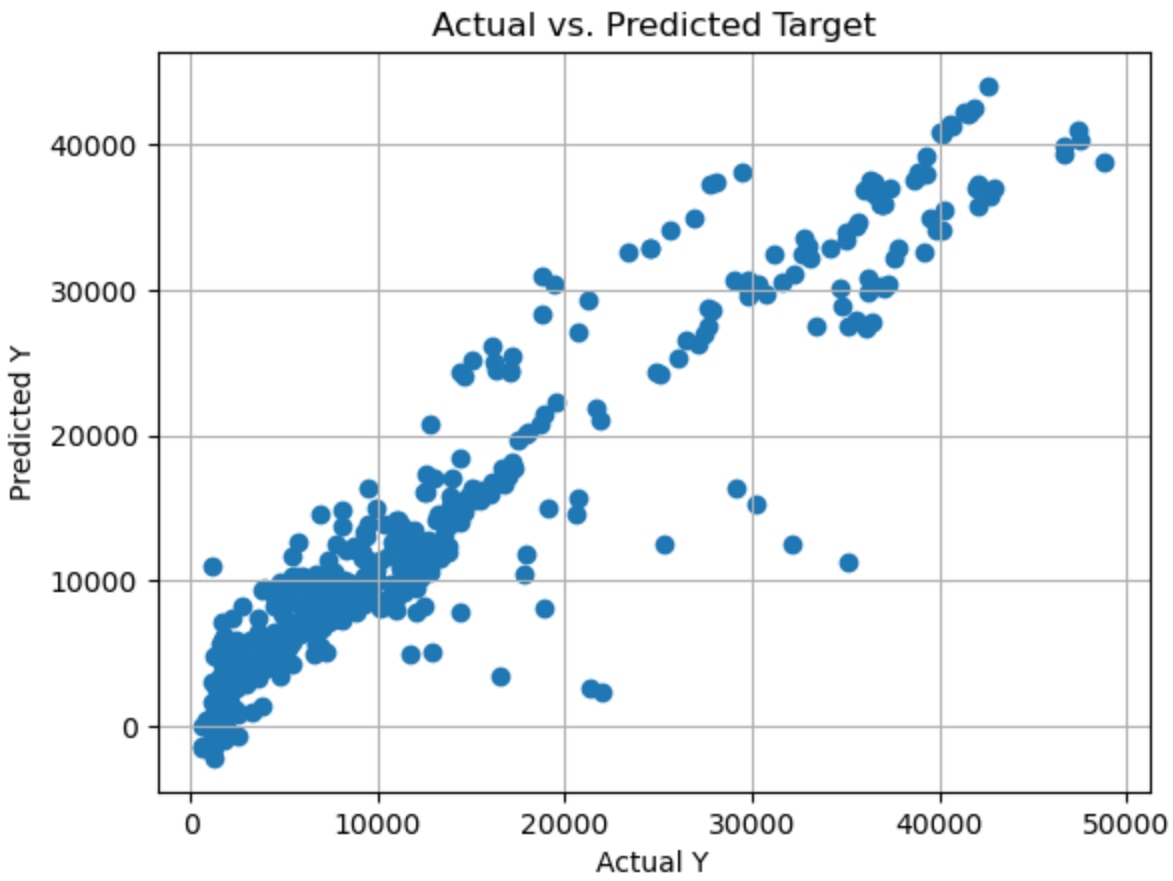
In [82]:
```
print("SGD Regression: Health Insurance Analysis ")
sgd_reg = SGDRegressor()
sgd_reg.fit(x_train, y_train)
sgd_score = sgd_reg.score(x_train, y_train)
print("R-squared:", sgd_score)

sgd_y_pred = sgd_reg.predict(x_test)
sgd_accuracy = explained_variance_score(y_test, sgd_y_pred)
sgd_accuracy = round(sgd_accuracy*100, 6)
print("SGD Regressor Model Accuracy:", sgd_accuracy, "%")
print()
```

```
SGD Regression: Health Insurance Analysis
R-squared: 0.8533187319127354
SGD Regressor Model Accuracy: 88.428217 %
```

Graph of the Actual vs. Predicted Target

In [83]:
```
plt.scatter(y_test, sgd_y_pred)
plt.grid()
plt.xlabel('Actual Y')
plt.ylabel('Predicted Y')
plt.title('Actual vs. Predicted Target')
plt.show()
```



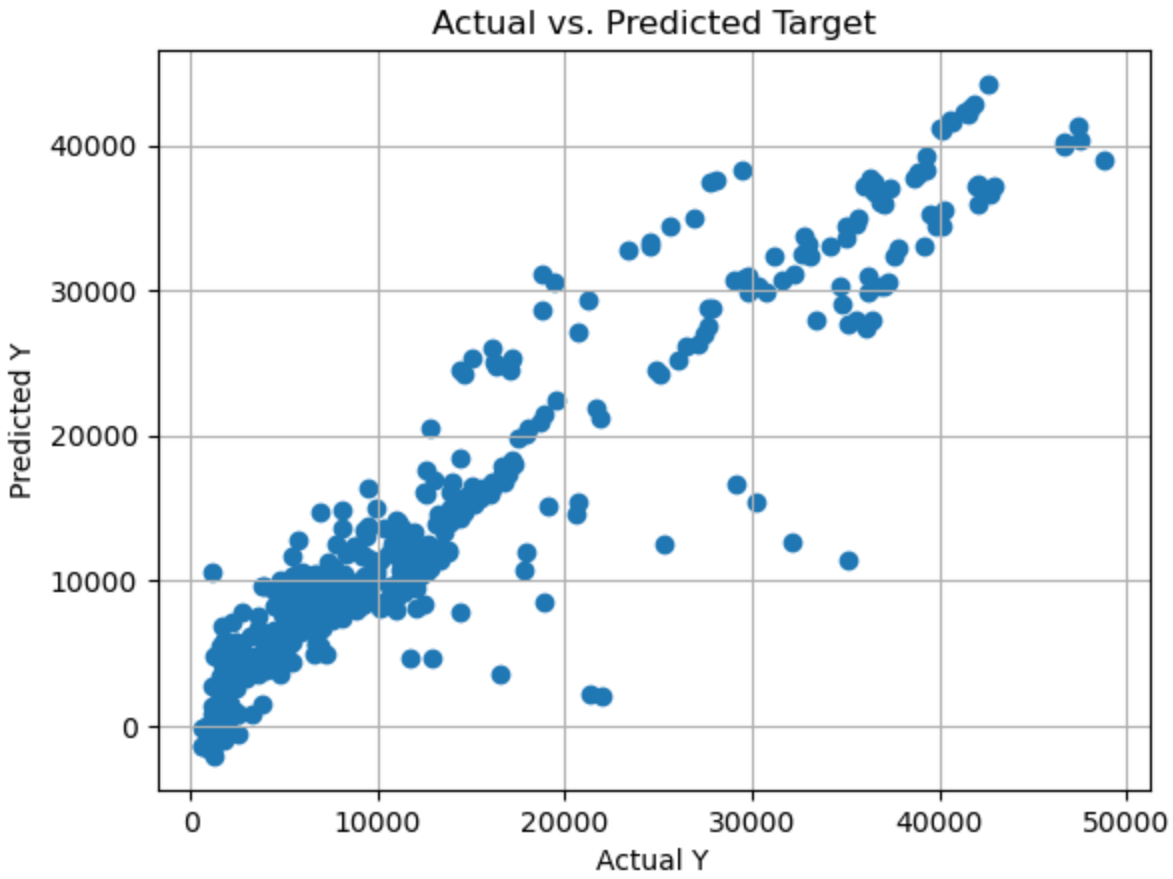
Let us calculate R-Squared, Linear Regressor Model Accuracy

In [84]:
```
print("Linear Regression: Boston Housing Prediction")
lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)
lin_score = lin_reg.score(x_train, y_train)
print("R-squared:", lin_score)

lin_y_pred = lin_reg.predict(x_test)
lin_accuracy = explained_variance_score(y_test, lin_y_pred)
lin_accuracy = round(lin_accuracy*100, 6)
print("Linear Regressor Model Accuracy:", lin_accuracy, "%")
print()
```

```
Linear Regression: Boston Housing Prediction
R-squared: 0.8535499220634002
Linear Regressor Model Accuracy: 88.425559 %
```

In [85]:
```
plt.scatter(y_test, lin_y_pred)
plt.grid()
plt.xlabel('Actual Y')
plt.ylabel('Predicted Y')
plt.title('Actual vs. Predicted Target')
plt.show()
```



Perform the stratified 5 fold cross validation

In [86]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

In [87]:
```
# create a KFold object with 5 splits
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)
scores = cross_val_score(sgd_reg, x_train, y_train, scoring='r2', cv=folds)
scores
```

Out[87]:
```
array([0.81314716, 0.83158359, 0.86930225, 0.87126011, 0.86018169])
```

Use random forest for cost prediction

In [88]:
```
# Random forest model
from sklearn.ensemble import RandomForestClassifier
```

In [89]:
```
rf_model = RandomForestRegressor()
rf_model.fit(x_train, y_train)
```

Out[89]:
```
RandomForestRegressor()
```

In [90]:
```
rf_predict = rf_model.predict(x_test)
```

In [91]:
```
rf_accuracy = explained_variance_score(y_test, rf_predict)
rf_accuracy = round(rf_accuracy*100, 6)
print("Random Forest Regressor Model Accuracy:", rf_accuracy, "%")
print()
```

```
Random Forest Regressor Model Accuracy: 92.232783 %
```

Use Extreme gradient boosting for cost prediction

In [92]:
```
from xgboost import XGBRegressor
```

In [93]:
```
model = XGBRegressor()
```

In [94]:
```
model.fit(x_train, y_train)
```

Out[94]:
```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
             grow_policy='depthwise', importance_type=None,
             interaction_constraints='', learning_rate=0.300000012, max_bin=256,
             max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=0,
             max_depth=6, max_leaves=0, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=0,
             num_parallel_tree=1, predictor='auto', random_state=0, ...)
```

In [95]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

Out[95]:
```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
             grow_policy=None, importance_type='gain',
             interaction_constraints='', learning_rate=0.300000012,
             max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=0, max_depth=6, max_leaves=None, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=4,
             num_parallel_tree=1, predictor=None, random_state=0, ...)
```

In [96]:
```
train_data_prediction = model.predict(x_train)
print(train_data_prediction)
```

```
[ 9211.719   3178.2673   798.486  ... 27897.738  13281.112  33490.72  ]
```

In [97]:
```
# Getting R squared value for training dataset
r2_train = metrics.r2_score(y_train, train_data_prediction)
print('R squared value for training dataset : ', r2_train)
```

```
R squared value for training dataset :  0.9979566915082827
```

In [98]:
```
test_data_prediction = model.predict(x_test)
```

In [99]:
```
r2_train = metrics.r2_score(y_test, test_data_prediction)
print('R squared value for testing dataset : ', r2_train)
```

```
R squared value for testing dataset :  0.9029961439264175
```

In [100…
```
xg_accuracy = explained_variance_score(y_test, test_data_prediction)
xg_accuracy = round(xg_accuracy*100, 6)
print("XGBoost Regressor Model Accuracy:", xg_accuracy, "%")
print()
```

```
XGBoost Regressor Model Accuracy: 90.374098 %
```

Estimate the cost of hospitalization for Christopher, Ms. Jayna (her date of birth is 12/28/1988, height is 170 cm, and weight is 85 kgs). She lives in a tier 1 city and her state's State ID is R1011. She lives with her partner and two children. She was found to be nondiabetic (HbA1c = 5.8). She smokes but is otherwise healthy. She has had no transplants or major surgeries. Her father died of lung cancer. Hospitalization costs will be estimated using tier 1 hospitals.

In [101…
```
data = {'Gender':0, 'month':12, 'date':28, 'year':1988, 'City tier':1, 'State ID':1, 'ch
index = [0]
cost_df = pd.DataFrame(data, index)
cost_df
```

Out[101]:

	Gender	month	date	year	City tier	State ID	children	HBA1C	smoker	Any Transplants	NumberOfMajorSurgeries	Ca his
0	0	12	28	1988	1	1	2	5.8	1	0	0	

In [102…
```
cost_pred = sgd_reg.predict(cost_df)
print("Cost of Hospitalization for Christopher, Ms. Jayna, using SGD Regression is: ", c

cost_pred1 = lin_reg.predict(cost_df)
print("Cost of Hospitalization for Christopher, Ms. Jayna, using Linear Regression is: "

cost_pred2 = rf_model.predict(cost_df)
print("Cost of Hospitalization for Christopher, Ms. Jayna using Random Forest is: ", cos

cost_pred3 = model.predict(cost_df)
print("Cost of Hospitalization for Christopher, Ms. Jayna, using XGBoost is: ", cost_pre
```

```
Cost of Hospitalization for Christopher, Ms. Jayna, using SGD Regression is:  [73358.298
93707]
Cost of Hospitalization for Christopher, Ms. Jayna, using Linear Regression is:  [66313.
52291818]
Cost of Hospitalization for Christopher, Ms. Jayna using Random Forest is:  [47433.2094]
Cost of Hospitalization for Christopher, Ms. Jayna, using XGBoost is:  [45331.117]
```

In [ ]: