Healthcare Insurance Analysis

In [1]:
```
# Let's import the necessary dependencies.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Now importing the dataset for the further operation.
cust_details = pd.read_csv("Hospitalisation details.csv")
medical_details = pd.read_csv("Medical Examinations.csv")
cust_name = pd.read_excel("Names.xlsx")
```

In [3]:
```
cust_details.head()
```

Out[3]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID
0	Id2335	1992	Jul	9	0	563.84	tier - 2	tier - 3	R1013
1	Id2334	1992	Nov	30	0	570.62	tier - 2	tier - 1	R1013
2	Id2333	1993	Jun	30	0	600.00	tier - 2	tier - 1	R1013
3	Id2332	1992	Sep	13	0	604.54	tier - 3	tier - 3	R1013
4	Id2331	1998	Jul	27	0	637.26	tier - 3	tier - 3	R1013

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

Out[4]: (2343, 9)

In [5]:
```
medical_details.head()
```

Out[5]:

	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

In [6]:
```
medical_details.shape
```

Out[6]: (2335, 8)

In [7]:
```
cust_name.head()
```

Out[7]:

	Customer ID	name
0	Id1	Hawks, Ms. Kelly
1	Id2	Lehner, Mr. Matthew D
2	Id3	Lu, Mr. Phil
3	Id4	Osborne, Ms. Kelsey
4	Id5	Kadala, Ms. Kristyn

In [8]:
```
cust_name.shape
```

Out[8]:
```
(2335, 2)
```

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

In [9]:
```
# Now combining the data so that all information could be examine in once go t
cust_df1 = pd.merge(cust_name, cust_details, on = "Customer ID")
cust_df1.head()
```

Out[9]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	State ID
0	Id1	Hawks, Ms. Kelly	1968	Oct	12	0	63770.43	tier - 1	tier - 3	R1013
1	Id2	Lehner, Mr. Matthew D	1977	Jun	8	0	62592.87	tier - 2	tier - 3	R1013
2	Id3	Lu, Mr. Phil	1970	?	11	3	60021.40	tier - 1	tier - 1	R1012
3	Id4	Osborne, Ms. Kelsey	1991	Jun	6	1	58571.07	tier - 1	tier - 3	R1024
4	Id5	Kadala, Ms. Kristyn	1989	Jun	19	0	55135.40	tier - 1	tier - 2	R1012

In [10]:
```
# Now lets combine the last data set and Complete the all information.
final_df = pd.merge(cust_df1, medical_details, on = "Customer ID")
final_df.head()
```

Out[10]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	H
0	Id1	Hawks, Ms. Kelly	1968	Oct	12	0	63770.43	tier - 1	tier - 3	R1013	47.410	
1	Id2	Lehner, Mr. Matthew D	1977	Jun	8	0	62592.87	tier - 2	tier - 3	R1013	30.360	
2	Id3	Lu, Mr. Phil	1970	?	11	3	60021.40	tier - 1	tier - 1	R1012	34.485	
3	Id4	Osborne, Ms. Kelsey	1991	Jun	6	1	58571.07	tier - 1	tier - 3	R1024	38.095	
4	Id5	Kadala, Ms. Kristyn	1989	Jun	19	0	55135.40	tier - 1	tier - 2	R1012	35.530	

In [11]:
```
final_df.shape
```

Out[11]: (2335, 17)

In [12]:
```
# Lets see the info to final data
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2335 entries, 0 to 2334
Data columns (total 17 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Customer ID            2335 non-null   object 
 1   name                   2335 non-null   object 
 2   year                   2335 non-null   object 
 3   month                  2335 non-null   object 
 4   date                   2335 non-null   int64  
 5   children               2335 non-null   int64  
 6   charges                2335 non-null   float64
 7   Hospital tier          2335 non-null   object 
 8   City tier              2335 non-null   object 
 9   State ID               2335 non-null   object 
 10  BMI                    2335 non-null   float64
 11  HBA1C                  2335 non-null   float64
 12  Heart Issues           2335 non-null   object 
 13  Any Transplants        2335 non-null   object 
 14  Cancer history         2335 non-null   object 
 15  NumberOfMajorSurgeries 2335 non-null   object 
 16  smoker                 2335 non-null   object 
dtypes: float64(3), int64(2), object(12)
memory usage: 328.4+ KB
```

In [13]:
```
final_df.dtypes.value_counts()
```

Out[13]:
```
object     12
float64     3
int64       2
dtype: int64
```

In [14]:
```
# Lets check the missing values
final_df.isnull().sum().sum()
```

Out[14]:
```
0
```

- No null or missing value but there is some unusual value that we have to deal.

3. Find the percentage of rows that have trivial value (for example, ?), and delete such rows if they do not contain significant information

In [15]:
```
trivial_value = final_df[final_df.eq("?").any(1)]
trivial_value
```

Out[15]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	State ID	BM
2	Id3	Lu, Mr. Phil	1970	?	11	3	60021.40	tier - 1	tier - 1	R1012	34.485
169	Id170	Torphy, Mr. Bobby	2000	Sep	5	1	37165.16	tier - 1	tier - 3	?	37.620
559	Id560	Pearlman, Mr. Oz	1994	Jul	1	3	17663.14	tier - 1	tier - 3	R1013	23.980
634	Id635	Bruns, Mr. Zachary T	2004	Jul	17	0	15518.18	tier - 2	tier - 3	R1015	25.175
1285	Id1286	Ainsley, Ms. Katie M.	?	Dec	12	1	8547.69	tier - 2	tier - 1	R1013	29.370
1288	Id1289	Levine, Ms. Annie J.	?	Jul	24	0	8534.67	tier - 2	tier - 3	R1024	24.320
1792	Id1793	Capriolo, Mr. Michael	1995	Dec	1	3	4827.90	tier - 1	tier - 2	?	18.905
2317	Id2318	Gagnon, Ms. Candice M	1996	?	18	0	770.38	tier - 3	?	R1012	18.820
2321	Id2322	Street, Ms. Holly	2002	?	19	0	750.00	tier - 3	tier - 1	R1012	21.380
2323	Id2324	Duffy, Ms. Meghan K	1999	Dec	26	0	700.00	?	tier - 3	R1013	22.240

In [16]:
```
trivial_value.shape
```

Out[16]: (10, 17)

In [17]:
```
# Percentage of row that have the trivial values
round(trivial_value.shape[0]/final_df.shape[0]*100, 2)
```

Out[17]: 0.43

In [18]:
```
# As percentage is too small so lets drop the all row that contain the trivial
final_df.drop(final_df[final_df.eq("?").any(1)].index, axis=0, inplace=True)
```

In [19]:
```
final_df.shape
```

Out[19]:
```
(2325, 17)
```

Nominal and Ordinal categorical variables

In [20]:
```
# First we will deal with the nominal categorical variable.
```

In [21]:
```
final_df["Heart Issues"].value_counts()
```

Out[21]:
```
No     1405
yes     920
Name: Heart Issues, dtype: int64
```

In [22]:
```
final_df["Any Transplants"].value_counts()
```

Out[22]:
```
No     2183
yes     142
Name: Any Transplants, dtype: int64
```

In [23]:
```
final_df["Cancer history"].value_counts()
```

Out[23]:
```
No     1934
Yes     391
Name: Cancer history, dtype: int64
```

In [24]:
```
final_df["smoker"].value_counts()
```

Out[24]:
```
No     1839
yes     486
Name: smoker, dtype: int64
```

In [25]:
```
# We have some categorical values so first of all we have to transform then by
from sklearn.preprocessing import LabelEncoder
```

In [26]:
```
le = LabelEncoder()
```

In [27]:
```
final_df["Heart Issues"] = le.fit_transform(final_df["Heart Issues"])
final_df["Any Transplants"] = le.fit_transform(final_df["Any Transplants"])
final_df["Cancer history"] = le.fit_transform(final_df["Cancer history"])
final_df["smoker"] = le.fit_transform(final_df["smoker"])
```

In [28]:
```
final_df.head()
```

Out[28]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	H
0	Id1	Hawks, Ms. Kelly	1968	Oct	12	0	63770.43	tier - 1	tier - 3	R1013	47.410	
1	Id2	Lehner, Mr. Matthew D	1977	Jun	8	0	62592.87	tier - 2	tier - 3	R1013	30.360	
3	Id4	Osborne, Ms. Kelsey	1991	Jun	6	1	58571.07	tier - 1	tier - 3	R1024	38.095	
4	Id5	Kadala, Ms. Kristyn	1989	Jun	19	0	55135.40	tier - 1	tier - 2	R1012	35.530	
5	Id6	Baker, Mr. Russell B.	1962	Aug	4	0	52590.83	tier - 1	tier - 3	R1011	32.800	

In [29]:
```
# Now we will deal with the ordinal categorical variable.
```

In [30]:
```
def clean_ordinal_variable(val):
    return int(val.replace("tier", "").replace(" ", "").replace("-", ""))
```

In [31]:
```
final_df["Hospital tier"] = final_df["Hospital tier"].map(clean_ordinal_variab
final_df["City tier"] = final_df["City tier"].map(clean_ordinal_variable)
```

In [32]:
```
final_df["City tier"].value_counts()
```

Out[32]:
```
2    807
3    789
1    729
Name: City tier, dtype: int64
```

In [33]:
```
final_df.head()
```

Out[33]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	H
0	Id1	Hawks, Ms. Kelly	1968	Oct	12	0	63770.43	1	3	R1013	47.410	
1	Id2	Lehner, Mr. Matthew D	1977	Jun	8	0	62592.87	2	3	R1013	30.360	
3	Id4	Osborne, Ms. Kelsey	1991	Jun	6	1	58571.07	1	3	R1024	38.095	
4	Id5	Kadala, Ms. Kristyn	1989	Jun	19	0	55135.40	1	2	R1012	35.530	
5	Id6	Baker, Mr. Russell B.	1962	Aug	4	0	52590.83	1	3	R1011	32.800	

5. Creating dummy variables

- The dataset has State ID, which has around 16 states. All states are not represented in equal proportions in the data. Creating dummy variables for all regions may also result in too many insignificant predictors. Nevertheless, only R1011, R1012, and R1013 are worth investigating further. Create a suitable strategy to create dummy variables with these restraints.

In [34]:
```
final_df["State ID"].value_counts()
```

Out[34]:
```
R1013    609
R1011    574
R1012    572
R1024    159
R1026     84
R1021     70
R1016     64
R1025     40
R1023     38
R1017     36
R1019     26
R1022     14
R1014     13
R1015     11
R1018      9
R1020      6
Name: State ID, dtype: int64
```

In [35]:
```
Dummies = pd.get_dummies(final_df["State ID"], prefix= "State_ID")
```

In [36]:
```
Dummies
```

Out[36]:

	State_ID_R1011	State_ID_R1012	State_ID_R1013	State_ID_R1014	State_ID_R1015	State_ID
0	0	0	1	0	0	
1	0	0	1	0	0	
3	0	0	0	0	0	
4	0	1	0	0	0	
5	1	0	0	0	0	
...	...	...	...	...	...	
2330	0	0	1	0	0	
2331	0	0	1	0	0	
2332	0	0	1	0	0	
2333	0	0	1	0	0	
2334	0	0	1	0	0	

2325 rows × 16 columns

In [37]:
```
# lets take only those state id which play significant role in the data set.
Dummy = Dummies[['State_ID_R1011','State_ID_R1012', 'State_ID_R1013', ]]
Dummy
```

Out[37]:

	State_ID_R1011	State_ID_R1012	State_ID_R1013
0	0	0	1
1	0	0	1
3	0	0	0
4	0	1	0
5	1	0	0
...	...	...	...
2330	0	0	1
2331	0	0	1
2332	0	0	1
2333	0	0	1
2334	0	0	1

2325 rows × 3 columns

In [38]:
```
final_df = pd.concat([final_df, Dummy], axis=1)
```

In [39]:
```
final_df.drop(['State ID'], inplace=True, axis=1)
final_df.head(10)
```

Out[39]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	BMI	HBA1
0	Id1	Hawks, Ms. Kelly	1968	Oct	12	0	63770.43	1	3	47.410	7.4
1	Id2	Lehner, Mr. Matthew D	1977	Jun	8	0	62592.87	2	3	30.360	5.7
3	Id4	Osborne, Ms. Kelsey	1991	Jun	6	1	58571.07	1	3	38.095	6.0
4	Id5	Kadala, Ms. Kristyn	1989	Jun	19	0	55135.40	1	2	35.530	5.4
5	Id6	Baker, Mr. Russell B.	1962	Aug	4	0	52590.83	1	3	32.800	6.5
6	Id7	Macpherson, Mr. Scott	1994	Oct	27	1	51194.56	1	3	36.400	6.0
7	Id8	Hallman, Mr. Stephen	1958	Jun	27	2	49577.66	2	2	36.960	7.9
8	Id9	Moran, Mr. Patrick R.	1963	Sep	4	1	48970.25	1	2	41.140	9.5
9	Id10	Benner, Ms. Brooke N.	1978	Dec	29	0	48885.14	1	2	38.060	10.7
10	Id11	Fierro Vargas, Ms. Paola Andrea	1959	Jul	22	0	48824.45	2	1	37.700	5.9

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

In [40]:
```
final_df['NumberOfMajorSurgeries'].value_counts()
```

Out[40]:
```
No major surgery    1070
1                    961
2                    272
3                     22
Name: NumberOfMajorSurgeries, dtype: int64
```

- The NumberOfMajorSurgeries variable contain string value no major Surgery that mean simpli is 0 surgery so we will replace this value into int value equal to zero.

In [41]:
```
final_df['NumberOfMajorSurgeries'] = final_df['NumberOfMajorSurgeries'].replac
```

In [42]:
```
final_df['NumberOfMajorSurgeries'] = final_df["NumberOfMajorSurgeries"].astype
```

7. Age appears to be a significant factor in this analysis. Calculate the patients' ages based on their dates of birth.

In [43]:
```
final_df["year"] = pd.to_datetime(final_df["year"], format='%Y').dt.year
final_df[["year"]]
```

Out[43]:

	year
0	1968
1	1977
3	1991
4	1989
5	1962
...	...
2330	1998
2331	1992
2332	1993
2333	1992
2334	1992

2325 rows × 1 columns

In [44]:
```
final_df["month"] = pd.to_datetime(final_df["month"], format='%b').dt.month
final_df["month"]
```

Out[44]:
```
0       10
1        6
3        6
4        6
5        8
        ..
2330     7
2331     9
2332     6
2333    11
2334     7
Name: month, Length: 2325, dtype: int64
```

In [45]:
```
final_df['DateInt'] = final_df["year"].astype(str) + final_df["month"].astype(
```

In [46]:
```
final_df['DOB'] = pd.to_datetime(final_df.DateInt, format = "%Y%m%d")
```

In [47]:
```
final_df.drop(["DateInt"], inplace = True, axis=1)
```

In [48]:
```
final_df.head()
```

Out[48]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	BMI	HBA1C	I
0	Id1	Hawks, Ms. Kelly	1968	10	12	0	63770.43	1	3	47.410	7.47	
1	Id2	Lehner, Mr. Matthew D	1977	6	8	0	62592.87	2	3	30.360	5.77	
3	Id4	Osborne, Ms. Kelsey	1991	6	6	1	58571.07	1	3	38.095	6.05	
4	Id5	Kadala, Ms. Kristyn	1989	6	19	0	55135.40	1	2	35.530	5.45	
5	Id6	Baker, Mr. Russell B.	1962	8	4	0	52590.83	1	3	32.800	6.59	

In [49]:
```
import datetime as dt
current_date = dt.datetime.now()
```

In [50]:
```
final_df['age'] = (((current_date - final_df.DOB).dt.days)/365).astype(int)
```

In [51]:
```
final_df.head()
```

Out[51]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	BMI	...	Heart Issues
0	Id1	Hawks, Ms. Kelly	1968	10	12	0	63770.43	1	3	47.410	...	C
1	Id2	Lehner, Mr. Matthew D	1977	6	8	0	62592.87	2	3	30.360	...	C
3	Id4	Osborne, Ms. Kelsey	1991	6	6	1	58571.07	1	3	38.095	...	C
4	Id5	Kadala, Ms. Kristyn	1989	6	19	0	55135.40	1	2	35.530	...	C
5	Id6	Baker, Mr. Russell B.	1962	8	4	0	52590.83	1	3	32.800	...	C

5 rows × 21 columns

8. The gender of the patient

- 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 [52]:
```
def gender(val):
    if "Ms." in val:
        return 0
    else:
        return 1
```

- Male = 1 & Female = 0

In [53]:
```
final_df["gender"] = final_df["name"].map(gender)
final_df.head(10)
```

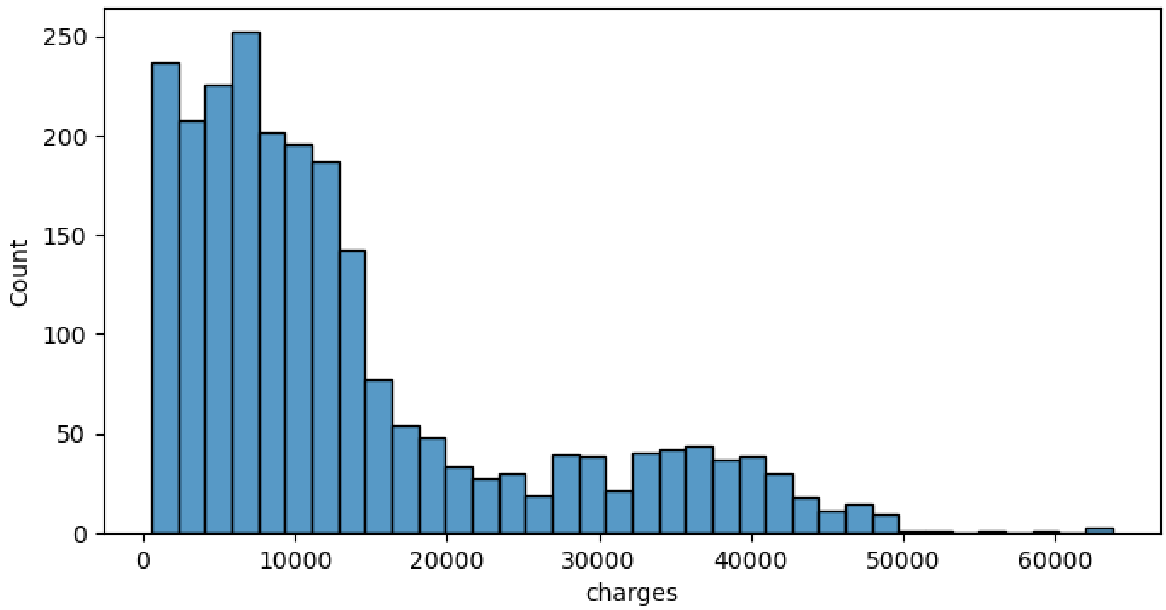
Out[53]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	BMI	...	Ti
0	Id1	Hawks, Ms. Kelly	1968	10	12	0	63770.43	1	3	47.410	...	
1	Id2	Lehner, Mr. Matthew D	1977	6	8	0	62592.87	2	3	30.360	...	
3	Id4	Osborne, Ms. Kelsey	1991	6	6	1	58571.07	1	3	38.095	...	
4	Id5	Kadala, Ms. Kristyn	1989	6	19	0	55135.40	1	2	35.530	...	
5	Id6	Baker, Mr. Russell B.	1962	8	4	0	52590.83	1	3	32.800	...	
6	Id7	Macpherson, Mr. Scott	1994	10	27	1	51194.56	1	3	36.400	...	
7	Id8	Hallman, Mr. Stephen	1958	6	27	2	49577.66	2	2	36.960	...	
8	Id9	Moran, Mr. Patrick R.	1963	9	4	1	48970.25	1	2	41.140	...	
9	Id10	Benner, Ms. Brooke N.	1978	12	29	0	48885.14	1	2	38.060	...	
10	Id11	Fierro Vargas, Ms. Paola Andrea	1959	7	22	0	48824.45	2	1	37.700	...	

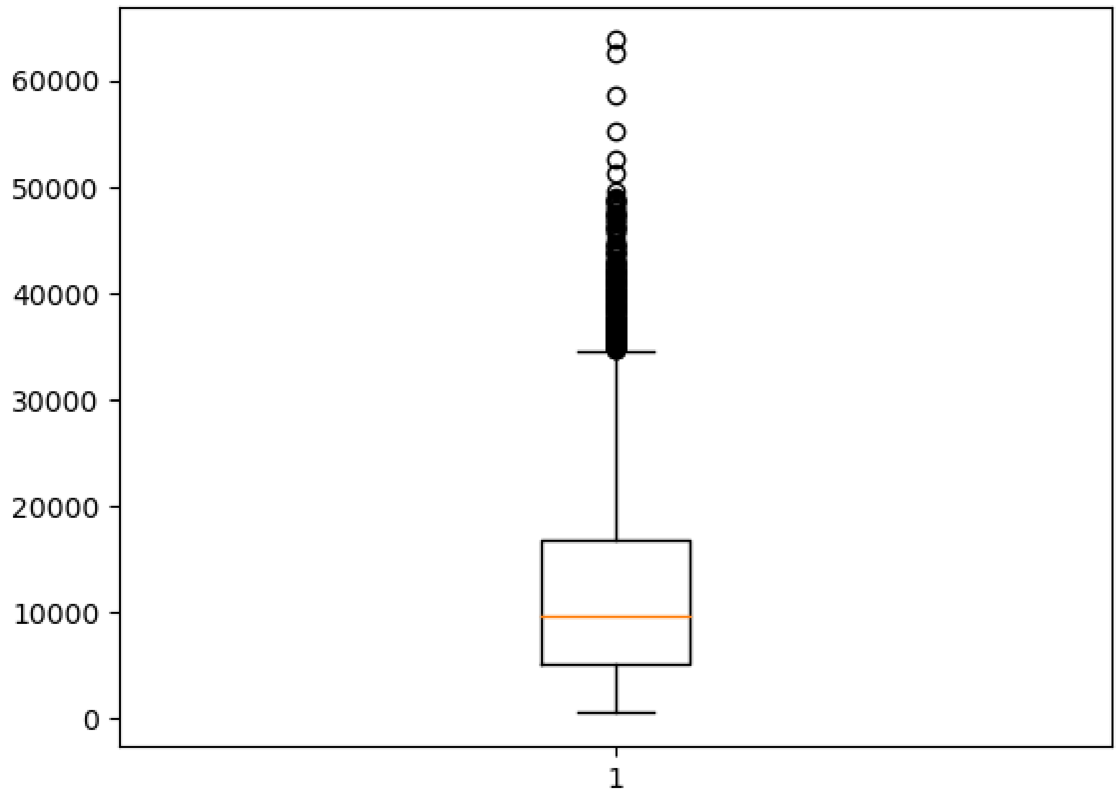
10 rows × 22 columns

9. visualizing the distribution of costs using a histogram, box and whisker plot, and swarm plot.

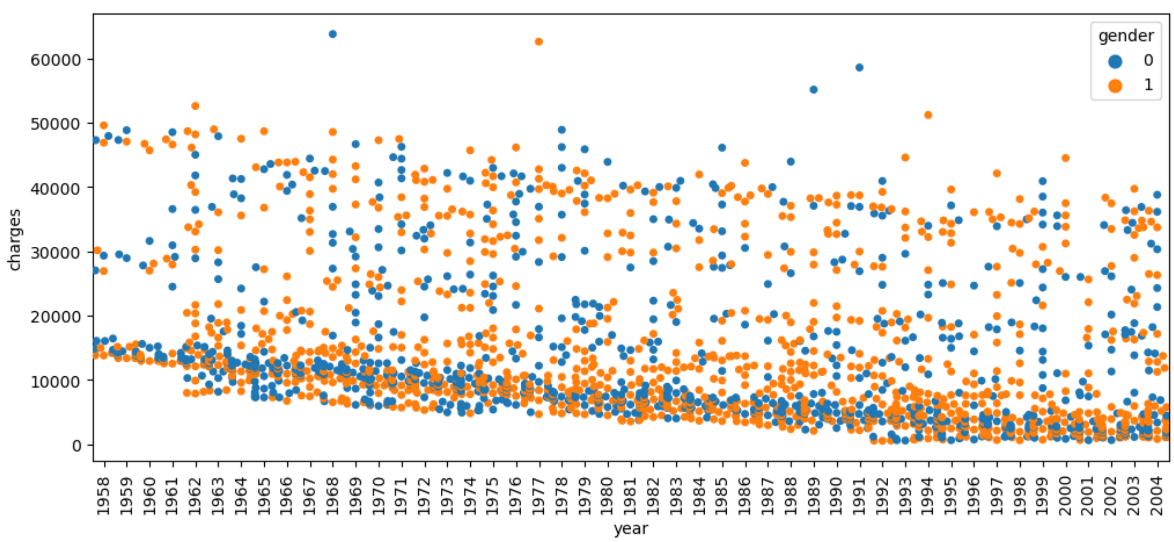
In [54]:
```
# Lets make the histogram for the cost distribution.
plt.figure(figsize=(8,4))
sns.histplot(final_df['charges'])
plt.show()
```



In [55]:
```
# Now visualize the cost distribution of the hospitals by box or whisker plot.
plt.boxplot(final_df['charges'])
plt.show()
```

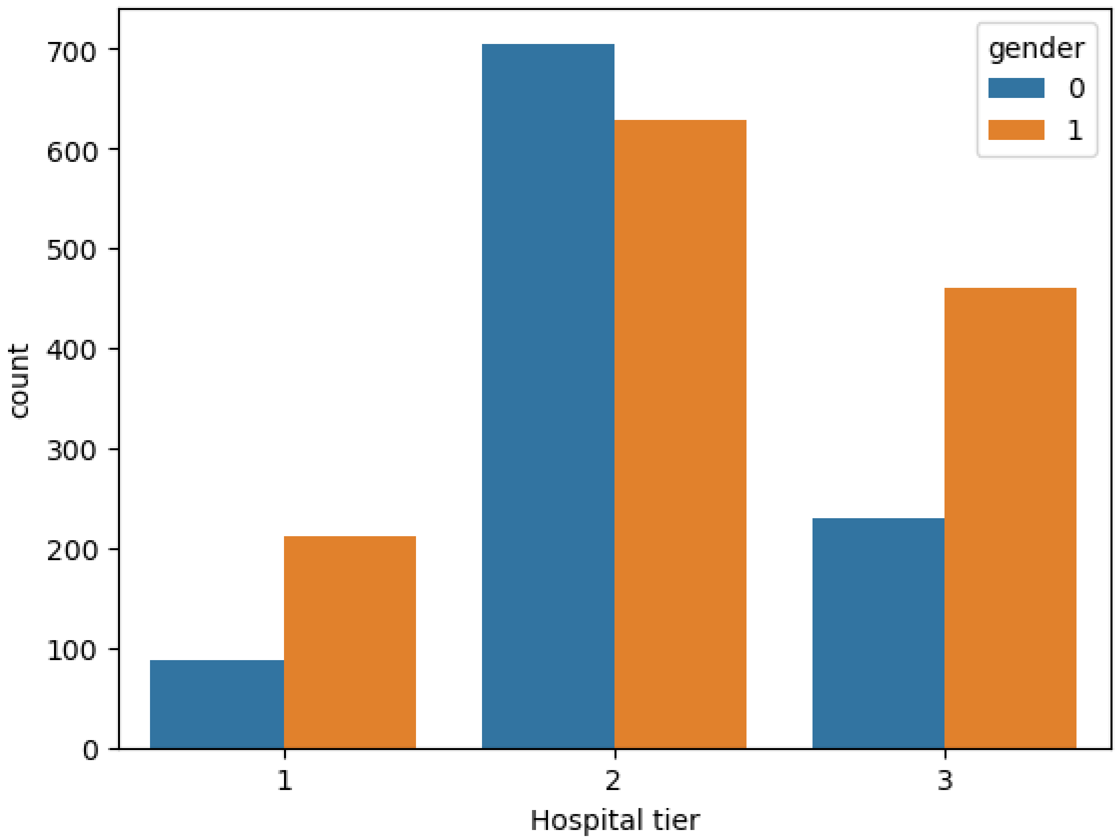


In [56]:
```
# Now visualize the cost distribution of the hospitals by swarm plot.
plt.figure(figsize=(12,5))
sns.swarmplot(x='year', y='charges', hue="gender", data=final_df)
plt.xticks(rotation=90)
plt.show()
```



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

In [57]:
```
sns.countplot(data = final_df, x='Hospital tier', hue= 'gender')
plt.show()
```



11. Creating a radar chart to showcase the median hospitalization cost for each tier of hospitals

In [58]:
```
print("median cost of tier 1 hospitals:", final_df[final_df["Hospital tier"]==
print("median cost of tier 2 hospitals:", final_df[final_df["Hospital tier"]==
print("median cost of tier 3 hospitals:", final_df[final_df["Hospital tier"]==
```

```
median cost of tier 1 hospitals: 32097.434999999998
median cost of tier 2 hospitals: 7168.76
median cost of tier 3 hospitals: 10676.83
```

In [59]:
```
df = pd.DataFrame(dict(r=[32097.43, 7168.76, 10676.83],theta=['tier 1 hospital
df
```

Out[59]:

	r	theta
0	32097.43	tier 1 hospital
1	7168.76	tier 2 hospital
2	10676.83	tier 3 hospital

In [60]:
```
import plotly.express as px
fig = px.line_polar(df, r='r', theta='theta', line_close=True)
fig.update_traces(fill='toself')
fig.show()
```

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

In [61]:
```
# Frequency table for count of the people according to the tier of city and ho
final_df["Hospital tier"].value_counts()
```

Out[61]:
```
2    1334
3     691
1     300
Name: Hospital tier, dtype: int64
```

In [62]:
```
# Frequency table for count of the people according to the tier of city and ho
final_df["Hospital tier"].value_counts()
```

Out[62]:
```
2    1334
3     691
1     300
Name: Hospital tier, dtype: int64
```

In [63]:
```
city_freq = final_df["City tier"].value_counts().rename_axis('City&hospital_ti
```

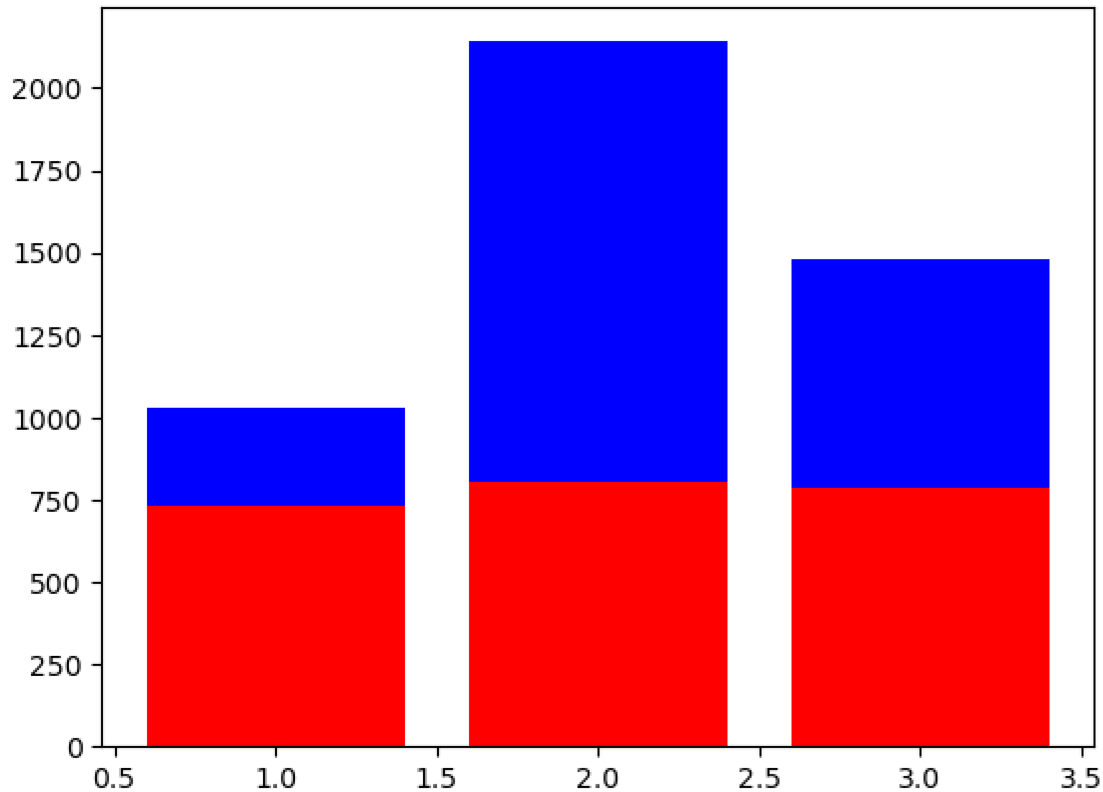
In [64]:
```
hospital_freq = final_df["Hospital tier"].value_counts().rename_axis('City&hos
```

In [65]:
```
df = pd.merge(city_freq, hospital_freq, on = 'City&hospital_tier')
df.head()
```

Out[65]:

	City&hospital_tier	city_counts	hospital_counts
0	2	807	1334
1	3	789	691
2	1	729	300

In [66]:
```
plt.bar(df["City&hospital_tier"], df["city_counts"], color='r')
plt.bar(df["City&hospital_tier"], df["hospital_counts"], bottom=df["city_count
plt.show()
```



13. Testing the following null hypotheses:-

In [67]:
```
from scipy.stats import ttest_1samp
```

In [68]:
```
# a. The average hospitalization costs for the three types of hospitals are no
print("median cost of tier 1 hospitals:", final_df[final_df["Hospital tier"]==
print("median cost of tier 2 hospitals:", final_df[final_df["Hospital tier"]==
print("median cost of tier 3 hospitals:", final_df[final_df["Hospital tier"]==
```

```
median cost of tier 1 hospitals: 32097.434999999998
median cost of tier 2 hospitals: 7168.76
median cost of tier 3 hospitals: 10676.83
```

- Interpretation H0: the distributions of all samples are equal. || H1: the distributions of one or more samples are not equal

In [69]:
```
from scipy.stats import friedmanchisquare
data1 = [32097.43]
data2 = [7168.76]
data3 = [10676.83]
stat, p = friedmanchisquare(data1, data2, data3)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
    print('Probably the same distribution')
else:
    print('Probably different distributions')
```

```
stat=2.000, p=0.368
Probably the same distribution
```

In [70]:
```
# b. The average hospitalization costs for the three types of cities are not s
print("median cost of tier 1 city:", final_df[final_df["City tier"]==1].charge
print("median cost of tier 2 city:", final_df[final_df["City tier"]==2].charge
print("median cost of tier 3 city:", final_df[final_df["City tier"]==3].charge
```

```
median cost of tier 1 city: 10027.15
median cost of tier 2 city: 8968.33
median cost of tier 3 city: 9880.07
```

In [71]:
```
data1 = [10027.15]
data2 = [8968.33]
data3 = [9880.07]
stat, p = friedmanchisquare(data1, data2, data3)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
    print('Probably the same distribution')
else:
    print('Probably different distributions')
```

```
stat=2.000, p=0.368
Probably the same distribution
```

In [72]:
```
# c. The average hospitalization cost for smokers is not significantly differe
print("median cost of smoker:", final_df[final_df["smoker"]==1].charges.mediar
print("median cost of non smoker:", final_df[final_df["smoker"]==0].charges.me
```

```
median cost of smoker: 34125.475
median cost of non smoker: 7537.16
```

In [73]:
```
from scipy.stats import kruskal
data1 = [34125.475]
data2 = [7537.16]
stat, p = kruskal(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
    print('Probably the same distribution')
else:
    print('Probably different distributions')
```

```
stat=1.000, p=0.317
Probably the same distribution
```

- Interpretation:- H0 the two samples are independent. H1: there is a dependency between the samples.

In [74]:
```
# d. Smoking and heart issues are independent
from scipy.stats import chi2_contingency
table = [[final_df["Heart Issues"].value_counts()],[final_df["smoker"].value_c
stat, p, dof, expected = chi2_contingency(table)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
    print('Probably independent')
else:
    print('Probably dependent')
```

```
stat=191.145, p=0.000
Probably dependent
```

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

In [75]:
```
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2325 entries, 0 to 2334
Data columns (total 22 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Customer ID             2325 non-null   object
 1   name                    2325 non-null   object
 2   year                    2325 non-null   int64
 3   month                   2325 non-null   int64
 4   date                    2325 non-null   int64
 5   children                2325 non-null   int64
 6   charges                 2325 non-null   float64
 7   Hospital tier           2325 non-null   int64
 8   City tier               2325 non-null   int64
 9   BMI                     2325 non-null   float64
 10  HBA1C                   2325 non-null   float64
 11  Heart Issues            2325 non-null   int32
 12  Any Transplants         2325 non-null   int32
 13  Cancer history          2325 non-null   int32
 14  NumberOfMajorSurgeries  2325 non-null   int32
 15  smoker                  2325 non-null   int32
 16  State_ID_R1011          2325 non-null   uint8
 17  State_ID_R1012          2325 non-null   uint8
 18  State_ID_R1013          2325 non-null   uint8
 19  DOB                     2325 non-null   datetime64[ns]
 20  age                     2325 non-null   int32
 21  gender                  2325 non-null   int64
dtypes: datetime64[ns](1), float64(3), int32(6), int64(7), object(2), uint8
(3)
memory usage: 315.6+ KB
```

In [76]:
```
# In the data frame same of the column are not usable to model building so let
#then indentify the highly corelated predictor.
final_df.drop(["Customer ID",'name', 'year', 'month', 'date', 'DOB'], inplace=
final_df.shape
```

Out[76]:
```
(2325, 16)
```

In [77]:
```
final_df.head()
```

Out[77]:

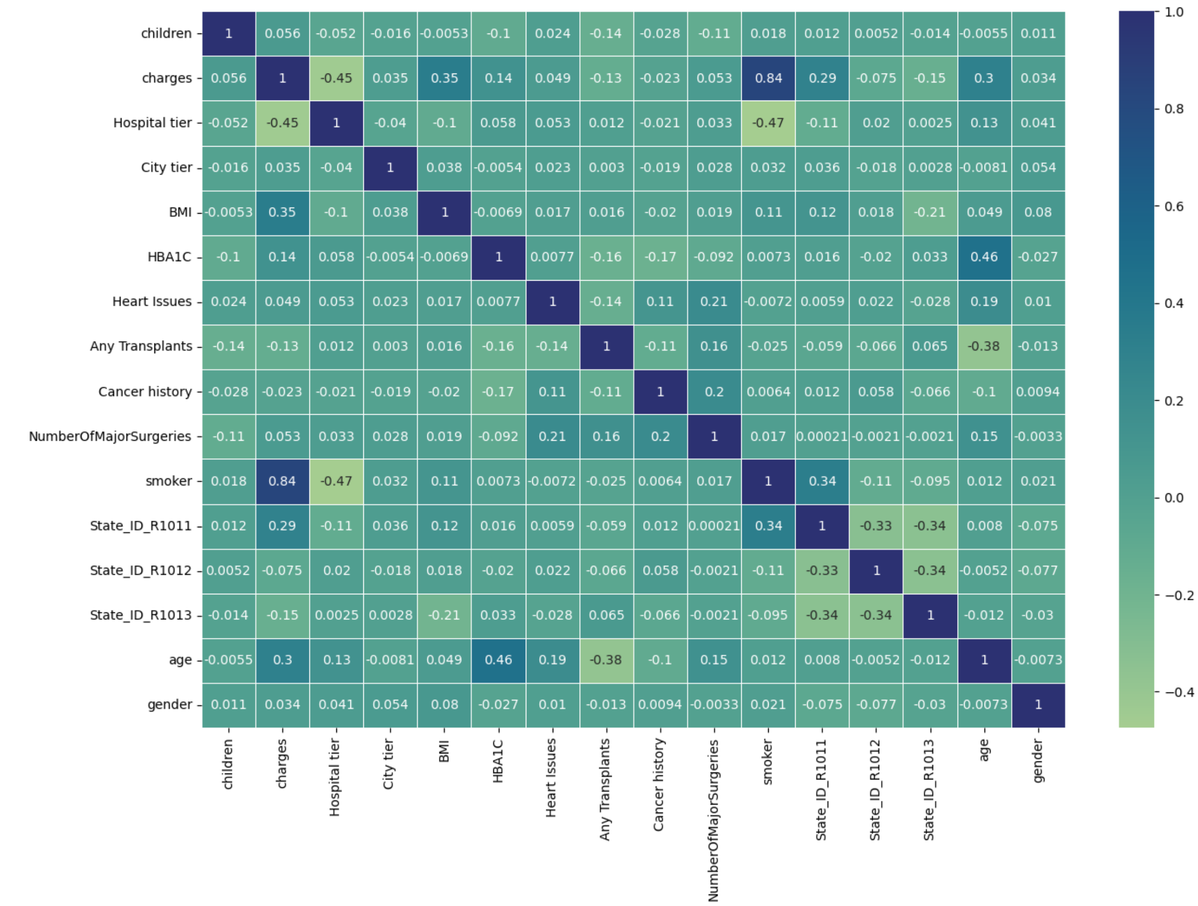
	children	charges	Hospital tier	City tier	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMa
0	0	63770.43	1	3	47.410	7.47	0	0	0	
1	0	62592.87	2	3	30.360	5.77	0	0	0	
3	1	58571.07	1	3	38.095	6.05	0	0	0	
4	0	55135.40	1	2	35.530	5.45	0	0	0	
5	0	52590.83	1	3	32.800	6.59	0	0	0	

In [78]:
```
corr = final_df.corr()
corr
```

Out[78]:

	children	charges	Hospital tier	City tier	BMI	HBA1C	He Issu
children	1.000000	0.055901	-0.052438	-0.015760	-0.005339	-0.101379	0.0239
charges	0.055901	1.000000	-0.446687	0.035300	0.346730	0.139697	0.0492
Hospital tier	-0.052438	-0.446687	1.000000	-0.039755	-0.104771	0.057855	0.0533
City tier	-0.015760	0.035300	-0.039755	1.000000	0.038123	-0.005404	0.0231
BMI	-0.005339	0.346730	-0.104771	0.038123	1.000000	-0.006920	0.0171
HBA1C	-0.101379	0.139697	0.057855	-0.005404	-0.006920	1.000000	0.0076
Heart Issues	0.023984	0.049299	0.053376	0.023152	0.017129	0.007699	1.0000
Any Transplants	-0.142040	-0.127028	0.011729	0.002970	0.015893	-0.159855	-0.1402
Cancer history	-0.027880	-0.022522	-0.021429	-0.018639	-0.020235	-0.170921	0.1111
NumberOfMajorSurgeries	-0.113161	0.053308	0.033230	0.027937	0.018851	-0.091594	0.2061
smoker	0.017713	0.838462	-0.474077	0.032034	0.107126	0.007257	-0.0071
State_ID_R1011	0.011666	0.286956	-0.114685	0.036049	0.115671	0.015525	0.0058
State_ID_R1012	0.005247	-0.074636	0.020272	-0.018253	0.017939	-0.019513	0.0217
State_ID_R1013	-0.013834	-0.150634	0.002455	0.002766	-0.208744	0.033453	-0.0279
age	-0.005457	0.304395	0.133771	-0.008070	0.049260	0.460558	0.1922
gender	0.011205	0.034069	0.041261	0.054073	0.079930	-0.027339	0.0102

In [79]:
```
plt.figure(figsize=(15,10))
sns.heatmap(corr, annot=True, linewidth=.5, cmap="crest")
plt.show()
```



- From the above corelation its clear that somker variable is highly corealted to the output variable.

Develop and evaluate the final model using regression with a stochastic gradient descent optimizer. Also, ensure that you apply all the following suggestions:

In [80]:
```
# lets first seperate the input and output data.
x = final_df.drop(["charges"], axis=1)
y = final_df[['charges']]
```

In [81]:
```
# Lets split the data set into the training and testing data.
from sklearn.model_selection import train_test_split
```

In [82]:
```
x_train, x_test, y_train, y_test  = train_test_split(x,y, test_size=.20, randc
```

In [83]:
```
# Now standardize the data.
from sklearn.preprocessing import StandardScaler
```

In [84]:
```
sc = StandardScaler()
```

In [85]:
```
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

In [86]:
```
from sklearn.linear_model import SGDRegressor
```

In [87]:
```
from sklearn.model_selection import GridSearchCV

params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2,0.3,0.4,0.5,
                    0.6,0.7,0.8,0.9,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,
                    9.0,10.0,20,50,100,500,1000],
         'penalty': ['l2', 'l1', 'elasticnet']}

sgd = SGDRegressor()

# Cross Validation
folds = 5
model_cv = GridSearchCV(estimator = sgd,
                        param_grid = params,
                        scoring = 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score = True,
                        verbose = 1)
model_cv.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 84 candidates, totalling 420 fits
```

Out[87]:
```
GridSearchCV(cv=5, estimator=SGDRegressor(),
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.
0,
                                   4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 5
0,
                                   100, 500, 1000],
                         'penalty': ['l2', 'l1', 'elasticnet']},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [88]:
```
model_cv.best_params_
```

Out[88]:
```
{'alpha': 50, 'penalty': 'l1'}
```

In [89]:
```
sgd = SGDRegressor(alpha= 100, penalty= 'l1')
```

In [90]:
```
sgd.fit(x_train, y_train)
```

Out[90]:
```
SGDRegressor(alpha=100, penalty='l1')
```

In [91]:
```
sgd.score(x_test, y_test)
```

Out[91]:
```
0.8591764409416783
```

In [92]:
```
y_pred = sgd.predict(x_test)
```

In [93]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [94]:
```
sgd_mae = mean_absolute_error(y_test, y_pred)
sgd_mse = mean_squared_error(y_test, y_pred)
sgd_rmse = sgd_mse*(1/2.0)
```

In [95]:
```
print("MAE:", sgd_mae)
print("MSE:", sgd_mse)
print("RMSE:", sgd_rmse)
```

```
MAE: 3155.204143185761
MSE: 23687356.036211107
RMSE: 11843678.018105553
```

In [96]:
```
# d. Determine the variable importance scores, and identify the redundant vari
importance = sgd.coef_
```

In [97]:
```
pd.DataFrame(importance, index = x.columns, columns=['Feature_imp'])
```

Out[97]:

	Feature_imp
children	370.159561
Hospital tier	-1135.336909
City tier	0.000000
BMI	2663.157603
HBA1C	68.327127
Heart Issues	0.000000
Any Transplants	0.000000
Cancer history	0.000000
NumberOfMajorSurgeries	0.000000
smoker	8761.364019
State_ID_R1011	-272.495381
State_ID_R1012	0.000000
State_ID_R1013	-305.968062
age	3401.025002
gender	0.000000

3. Use random forest and extreme gradient boosting for cost prediction, share your crossvalidation results, and calculate the variable importance scores

random forest

In [98]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [99]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)

# Train the model on training data
rf.fit(x_train, y_train)
```

Out[99]:
```
RandomForestRegressor(n_estimators=1000, random_state=42)
```

In [100]:
```
score = rf.score(x_test,y_test)
score
```

Out[100]:
```
0.9222696338245824
```

In [101]:
```
y_pred = rf.predict(x_test)
```

In [102]:
```
rf_mae = mean_absolute_error(y_test, y_pred)
```

In [103]:
```
rf_mae
```

Out[103]:
```
1870.3529629462323
```

Extreme gradient boosting

In [104]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

In [105]:
```
# Instantiate model with 1000 decision trees
gbr = GradientBoostingRegressor(n_estimators = 1000, random_state = 42)

# Train the model on training data
gbr.fit(x_train, y_train)
```

Out[105]:
```
GradientBoostingRegressor(n_estimators=1000, random_state=42)
```

In [106]:
```
score = gbr.score(x_test,y_test)
score
```

Out[106]:
```
0.9042734212625119
```

In [107]:
```
y_pred = gbr.predict(x_test)
gbr_mae = mean_absolute_error(y_test, y_pred)
gbr_mae
```

Out[107]:
```
2375.8700944163274
```

4. Case scenario

In [117]:
```
# First we need to calculate the age of the person.
date = str(19881228)
date1 = pd.to_datetime(date, format = "%Y%m%d")
```

In [118]:
```
current_date = (dt.datetime.now())
current_date
```

Out[118]:
```
datetime.datetime(2023, 2, 13, 16, 54, 31, 143768)
```

In [120]:
```
age = (current_date - date1)
age
```

Out[120]:
```
Timedelta('12465 days 16:54:31.143768')
```

In [121]:
```
age = int(12421/365)
age
```

Out[121]: 34

In [122]:
```
# now with the help of height and weight we will calculate the BMI.
height_m = 170/100
height_sq = height_m*height_m
BMI = 85/height_sq
np.round(BMI,2)
```

Out[122]: 29.41

In [123]:
```
# Now lets gen
list = [[2,1,1,24.41,5.8,0,0,0,0,1,1,0,0,34,0]]
```

In [124]:
```
df = pd.DataFrame(list, columns = ['children', 'Hospital tier', 'City tier', '
                              'Cancer history','NumberOfMajorSurgeries', 'smok
                              'State_ID_R1013', 'age', 'gender'] )
df
```

Out[124]:

	children	Hospital tier	City tier	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries
0	2	1	1	24.41	5.8	0	0	0	0

5. Find the predicted hospitalization cost using all models. The predicted value should be the mean of the five models' predicted values.

In [125]:
```
Hospital_cost = []
```

In [126]:
```
# Now lets predict the hospitalization cost through SGDRegressor
Cost1 = sgd.predict(df)
Hospital_cost.append(Cost1)
```

In [127]:
```
# Now lets predict the hospitalization cost through Random Forest
Cost2 = rf.predict(df)
Hospital_cost.append(Cost2)
```

In [128]:
```
# Now lets predict the hospitalization cost throug Extreme gradient Booster
Cost3 = gbr.predict(df)
Hospital_cost.append(Cost3)
```

In [129]:
```
avg_cost = np.mean(Hospital_cost)
avg_cost
```

Out[129]:
```
104104.91171519303
```

Tableau Dashboard Link :-

- [https://public.tableau.com/app/profile/akshaydeep.chauhan/viz/Healthcarechargesanalysis_1 publish=yes (https://public.tableau.com/app/profile/akshaydeep.chauhan/viz/Healthcarechargesanalysis_ publish=yes)](https://public.tableau.com/app/profile/akshaydeep.chauhan/viz/Healthcarechargesanalysis_1)

SQL :-

```
/* Question No:-1. To gain a comprehensive understanding of the factors
influencing hospitalization costs, it is
necessary to combine the tables provided. Merge the two tables by first
identifying the columns in the data tables
that will help you in merging.
a. In both tables, add a Primary Key constraint for these columns */

/* Hint: You can remove duplicates and null values from the column and then
use ALTER TABLE to add a Primary Key
constraint. */

create database job_readiness;
use job_readiness;
select * from hospital_detail;
select * from medical_detail;

-- Lets Deal with the null value.
SET SQL_SAFE_UPDATES = 0;
delete from hospital_detail where `State ID`='?';
delete from hospital_detail where `City tier`='?';

-- Now lets assign the primary key to the column in the table.
ALTER TABLE `job_readiness`.`hospital_detail`
CHANGE COLUMN `Customer ID` `Customer ID` varchar(20),
ADD PRIMARY KEY (`Customer ID`);

ALTER TABLE `job_readiness`.`medical_detail`
```

```
CHANGE COLUMN `Customer ID` `Customer ID` varchar(20),
ADD PRIMARY KEY (`Customer ID`);

-- Now lets merge the both table for better understanding of hospitalisation
cost.
select * from hospital_detail as h inner join medical_detail as m
on h.`Customer ID` = m.`Customer ID`;

/* Question No:-2. Retrieve information about people who are diabetic and
have heart problems with their average age,
 the average number of dependent children, average BMI, and average
hospitalization costs */

select m.HBA1C, m.`Heart Issues`, avg(h.children), avg(m.BMI),
avg(h.charges)
from medical_detail as m
inner join hospital_detail as h
on h.`Customer ID` = m.`Customer ID`
where m.HBA1C>6.5 and m.`Heart Issues`= 'yes';

/* Question NO.3:- Find the average hospitalization cost for each hospital
tier and each city level.*/

select `Hospital tier`, avg(charges) as avg_cost from hospital_detail group
by `Hospital tier`;
select `City tier`, avg(charges) as avg_cost from hospital_detail group by
`City tier`;

/* Question No4:- Determine the number of people who have had major surgery
with a history of cancer. */

select count(`Customer ID`) from medical_detail where `Cancer history`='Yes'
and NumberOfMajorSurgeries>0;

/* Question No5:- Determine the number of tier-1 hospitals in each state. */

select `State ID`, count(`Hospital tier`) from hospital_detail where
`Hospital tier`='tier - 1' group by `State ID`;
```