Problem statement:

A significant public health concern is the rising cost of healthcare. Therefore, it's crucial to be able to predict future costs and gain a solid understanding of their causes.

Objective:

The objective of this project is to predict patients' healthcare costs and to identify factors contributing to this prediction.

In [244]:
```
# Data manipulation
import numpy as np
import pandas as pd

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# warnings
import warnings
warnings.simplefilter('ignore')
```

-- Reading data from all the datasets

In [245]:
```
df1=pd.read_csv('Downloads/Datasets (1)/Capstone_1/Hospitalisation details.csv')
df1.head(2)
```

Out[245]:

	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

In [246]:
```
df2=pd.read_csv('Downloads/Datasets (1)/Capstone_1/Medical Examinations.csv')
df2.head(2)
```

Out[246]:

	Customer ID	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker
0	Id1	47.41	7.47	No	No	No	No major surgery	yes
1	Id2	30.36	5.77	No	No	No	No major surgery	yes

In [247]:
```
df3=pd.read_excel('Downloads/Datasets (1)/Capstone_1/Names.xlsx')
df3.head(2)
```

Out[247]:

	Customer ID	name
0	Id1	Hawks, Ms. Kelly
1	Id2	Lehner, Mr. Matthew D

In [248]:
```
print(df1.shape)
print(df2.shape)
print(df3.shape)
```

```
(2343, 9)
(2335, 8)
(2335, 2)
```

In [249]:
```
df1.duplicated().sum()
```
Out[249]:
```
0
```

--Combining all the files using merge, so that all the information is in one place. key column is to be join is Customer ID.

In [250]:
```
# Merge Dataframes on Customer ID coloumn
merged_df=pd.merge(df1,df2,on='Customer ID',how='outer')
merged_df=pd.merge(merged_df,df3,on='Customer ID',how='outer')
merged_df.head(2)
```
Out[250]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	
0	Id2335	1992	Jul	9	0	563.84	tier - 2	tier - 3	R1013	17.58	4.51	No	No	No	1	No	Ge Mr.
1	Id2334	1992	Nov	30	0	570.62	tier - 2	tier - 1	R1013	17.60	4.39	No	No	No	1	No	Rose Mr. E

In [251]:
```
merged_df.shape
```
Out[251]:
```
(2343, 17)
```

In [252]:
```
merged_df.nunique()
```
Out[252]:
```
Customer ID               2338
year                        48
month                        8
date                        30
children                     6
charges                   2333
Hospital tier                4
```

```
City tier                    4
State ID                    17
BMI                       1335
HBA1C                      667
Heart Issues                 2
Any Transplants              2
Cancer history               2
NumberOfMajorSurgeries       4
smoker                       3
name                      2335
dtype: int64
```

In [253]:
```
merged_df.drop_duplicates(inplace=True)
```

In [254]:
```
merged_df.shape
```

Out[254]: (2343, 17)

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

In [255]:
```
# Replace '?' with NaN
merged_df.replace('?',pd.NA,inplace=True)

# Create a boolean mask for trivial values
trivial_mask=merged_df.applymap(lambda x:x=='?')
```

In [256]:
```
trivial_mask.head(2)
```

Out[256]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	nar
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fal
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fal

In [257]:
```
# Count the number of rows for trivial value
trivial_rows_count=trivial_mask.any(axis=1).sum()
trivial_rows_count
```

Out[257]:
```
0
```

In [258]:
```
percentage_trivial_rows=(trivial_rows_count/len(merged_df))*100
print(f'percentage of rows with trivial values: {percentage_trivial_rows:.2f}%')
```

```
percentage of rows with trivial values: 0.00%
```

**** Above we replaced '?' with pd.NaN and created boolean mask for trivial values, and lastly found percentage of the rows with trivial values**

-- Checking for missing values in the dataset

In [259]:
```
merged_df.isnull().sum()
```

Out[259]:
```
Customer ID                6
year                       2
month                      3
date                       0
children                   0
charges                    0
Hospital tier              1
City tier                  1
State ID                   2
BMI                        8
HBA1C                      8
Heart Issues               8
Any Transplants            8
Cancer history             8
NumberOfMajorSurgeries     8
smoker                    10
name                       8
```

```
dtype: int64
```

There are missing values in data we have to treat this missing values

In [260]:
```
merged_df.dropna(subset=['Customer ID','year','month','Hospital tier','City tier','State ID','BMI','HBA1C','Heart Issues',
                         'Any Transplants','Cancer history','NumberOfMajorSurgeries','smoker','name'],inplace=True)
```

In [261]:
```
merged_df.isnull().sum()
```

Out[261]:
```
Customer ID               0
year                      0
month                     0
date                      0
children                  0
charges                   0
Hospital tier             0
City tier                 0
State ID                  0
BMI                       0
HBA1C                     0
Heart Issues              0
Any Transplants           0
Cancer history            0
NumberOfMajorSurgeries    0
smoker                    0
name                      0
dtype: int64
```

In [262]:
```
merged_df.shape
```

Out[262]:
```
(2325, 17)
```

**Now no null values in dataset we treat it by deleting rows which have null values....The other way to teat null values is by replacing with median and mode, but here we don't have huge amount of null values in each column so we dropped the rows which contains null value.

-- 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. Creating a suitable strategy to create dummy variables with these restraints.

In [263]:
```
merged_df['State ID'].value_counts()
```

Out[263]:
```
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 [264]:
```
merged_df['State ID'].nunique()
```

Out[264]:
```
16
```

In [265]:
```
merged_df.replace(['R1024','R1026','R1021','R1016','R1025','R1023','R1017','R1019','R1022','R1014','R1015','R1018','R1020'],
                  'others',inplace=True)
```

In [266]:
```
merged_df['State ID'].value_counts()
```

Out[266]:
```
R1013     609
R1011     574
R1012     572
others    570
Name: State ID, dtype: int64
```

In [267]:
```
merged_df.replace('R1011',0,inplace=True)
merged_df.replace('R1012',1,inplace=True)
merged_df.replace('R1013',2,inplace=True)
merged_df.replace('others',3,inplace=True)
```

****Except R1013,R1012,R1011 state ID's we combined all other state ID's and put them in one ID named as 'other'.**

****filtered all the state id a part from 3 having maximum occurences and put in under new state_id name as 'other', now there are 4 unique state id's.**

****and replaced R1011 with 0, R1012 with 1, R1013 with 2 and others with 3.**

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

In [268]:
```
merged_df['NumberOfMajorSurgeries'].unique()
```

Out[268]:
```
array(['1', 'No major surgery', '2', '3'], dtype=object)
```

In [269]:
```
merged_df.replace(['No major surgery'],0,inplace=True)
```

In [270]:
```
merged_df['NumberOfMajorSurgeries']=merged_df['NumberOfMajorSurgeries'].astype('int64')
```

In [271]:
```
merged_df['NumberOfMajorSurgeries'].unique()
```

Out[271]:
```
array([1, 0, 2, 3], dtype=int64)
```

**Above we replaced the value 'No major surgery' with 0.

**And the column 'NumberOfMajorSurgeries' is object datatype, so converted this column into int64 by using astype(int64).

-- Shows the HBA1C report (HBA1C measures the amount of sugar in the blood (glucose), where HBA1C greater than 6.5 is considered diabetic assuming as 1 and less than 6.5 is considered as non-diabetic assuming as 0

In [272]:
```
merged_df["HBA1C"] = np.where(merged_df["HBA1C"] > 6.5, 1.0, 0)
merged_df["HBA1C"]=merged_df["HBA1C"].astype('int64')
merged_df.rename(columns = {'HBA1C':'Diabetic'}, inplace = True)
merged_df.head(2)
```

Out[272]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	Diabetic	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	
0	Id2335	1992	Jul	9	0	563.84	tier - 2	tier - 3	2	17.58	0	No	No	No	1	No	Ge Mr.
1	Id2334	1992	Nov	30	0	570.62	tier - 2	tier - 1	2	17.60	0	No	No	No	1	No	Rose Mr. E

**Above HBA1C column values changed with 1(diabetic) if HBA1C is greater than 6.5 and with 0(non-diabetic) if HBA1C is less than 6.5. ¶

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

** year,month and date columns are object datatype converting them into datetime datatype

In [273]:
```
merged_df['year']=pd.to_datetime(merged_df['year'])
merged_df['month']=pd.to_datetime(merged_df['month'],format='%b')
merged_df['date']=pd.to_datetime(merged_df['date'],format='%d',errors='coerce')
```

In [274]:
```
merged_df.dtypes
```

Out[274]:
```
Customer ID                         object
year                        datetime64[ns]
month                       datetime64[ns]
date                        datetime64[ns]
children                             int64
charges                            float64
Hospital tier                       object
City tier                           object
State ID                             int64
BMI                                float64
Diabetic                             int64
Heart Issues                        object
Any Transplants                     object
Cancer history                      object
NumberOfMajorSurgeries               int64
smoker                              object
name                                object
dtype: object
```

In [275]:
```
merged_df.head(2)
```

Out[275]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	Diabetic	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	
0	Id2335	1992-01-01	1900-07-01	1900-01-09	0	563.84	tier - 2	tier - 3	2	17.58	0	No	No	No	1	No	M
1	Id2334	1992-01-01	1900-11-01	1900-01-30	0	570.62	tier - 2	tier - 1	2	17.60	0	No	No	No	1	No	Ro Mr

**ofter converting, extract particular year,month and day from those columns and assign them into new columns respectively

In [276]:
```
from datetime import date
```

In [277]:
```
merged_df['Year']=merged_df['year'].apply(lambda x:int(x.year))
```

In [278]:
```
merged_df['Month']=merged_df['month'].apply(lambda x:int(x.month))
```

In [279]:
```
merged_df['Date']=merged_df['date'].apply(lambda x:int(x.day))
```

In [280]:
```
merged_df.head(2)
```

Out[280]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	Diabetic	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	
0	Id2335	1992-01-01	1900-07-01	1900-01-09	0	563.84	tier - 2	tier - 3	2	17.58	0	No	No	No	1	No	M
1	Id2334	1992-01-01	1900-11-01	1900-01-30	0	570.62	tier - 2	tier - 1	2	17.60	0	No	No	No	1	No	Ro Mr

In [281]:
```
merged_df.dtypes
```

Out[281]:
```
Customer ID                object
year               datetime64[ns]
month              datetime64[ns]
date               datetime64[ns]
children                    int64
charges                   float64
Hospital tier              object
City tier                  object
State ID                    int64
BMI                       float64
Diabetic                    int64
```

```
Heart Issues                 object
Any Transplants              object
Cancer history               object
NumberOfMajorSurgeries        int64
smoker                       object
name                         object
Year                          int64
Month                         int64
Date                          int64
dtype: object
```

**dropping old year, month and date columns

In [282]:
```
merged_df=merged_df.drop(columns=['year','month','date'])
```

In [283]:
```
merged_df.head(2)
```

Out[283]:

	Customer ID	children	charges	Hospital tier	City tier	State ID	BMI	Diabetic	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	name	Year	Month
0	Id2335	0	563.84	tier - 2	tier - 3	2	17.58	0	No	No	No	1	No	German, Mr. Aaron K	1992	7
1	Id2334	0	570.62	tier - 2	tier - 1	2	17.60	0	No	No	No	1	No	Rosendahl, Mr. Evan P	1992	11

**Now combining all three columns(year,month and date) into one column as Date_of_ Birth

In [284]:
```
merged_df['Date_of_Birth']=merged_df.apply(lambda row:pd.to_datetime(f"{row['Year']}-{row['Month']}-{row['Date']}",
                                                                   errors='coerce'),axis=1)
```

In [285]:
```
merged_df.head(2)
```

Out[285]:

	Customer ID	children	charges	Hospital tier	City tier	State ID	BMI	Diabetic	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	name	Year	Month
0	Id2335	0	563.84	tier - 2	tier - 3	2	17.58	0	No	No	No	1	No	German, Mr. Aaron K	1992	7
1	Id2334	0	570.62	tier - 2	tier - 1	2	17.60	0	No	No	No	1	No	Rosendahl, Mr. Evan P	1992	11

**caluculating age by substracting Date_of_Birth from current_datetime and assigning age to the new column as age

In [286]:

```
from datetime import datetime
current_datetime=datetime.now()
current_datetime
```

Out[286]: datetime.datetime(2024, 2, 7, 16, 57, 20, 683935)

In [287]:

```
merged_df['age']=(current_datetime - merged_df['Date_of_Birth']).dt.days//365
merged_df.head(2)
```

Out[287]:

	Customer ID	children	charges	Hospital tier	City tier	State ID	BMI	Diabetic	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	name	Year	Month
0	Id2335	0	563.84	tier - 2	tier - 3	2	17.58	0	No	No	No	1	No	German, Mr. Aaron K	1992	7
1	Id2334	0	570.62	tier - 2	tier - 1	2	17.60	0	No	No	No	1	No	Rosendahl, Mr. Evan P	1992	11

**df['date_of_birth']).dt.days calculates the difference in days, and then // 365 converts it to years. This approach handles missing or invalid values gracefully.

--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 [288]:
```
merged_df['gender']=merged_df['name'].apply(lambda x: 'male' if 'Mr' in x else ('female' if 'Ms' in x else None))
```

In [289]:
```
merged_df.head(2)
```

Out[289]:

	Customer ID	children	charges	Hospital tier	City tier	State ID	BMI	Diabetic	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	name	Year	Month
0	Id2335	0	563.84	tier - 2	tier - 3	2	17.58	0	No	No	No	1	No	German, Mr. Aaron K	1992	7
1	Id2334	0	570.62	tier - 2	tier - 1	2	17.60	0	No	No	No	1	No	Rosendahl, Mr. Evan P	1992	11

In [290]:
```
merged_df['gender'].unique()
```

Out[290]:
```
array(['male', 'female'], dtype=object)
```

****In the name column, names have their salutations like Mr and Ms based on these keywords separated names where male for the names which have Mr and female for the names which have Ms then assigned them to the new column as gender**

In [291]:
```
merged_df.isnull().sum()
```

Out[291]:
```
Customer ID          0
children             0
charges              0
Hospital tier        0
City tier            0
State ID             0
BMI                  0
```

```
Diabetic                  0
Heart Issues              0
Any Transplants           0
Cancer history            0
NumberOfMajorSurgeries    0
smoker                    0
name                      0
Year                      0
Month                     0
Date                      0
Date_of_Birth             0
age                       0
gender                    0
dtype: int64
```

In [293]:
```
#merged_df1=merged_df.copy()
```

In [294]:
```
#merged_df1.to_csv('Downloads/Datasets (1)/Capstone_1/Hospitalisation details and Medical Examinations.csv')
```

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

In [295]:
```
merged_df['gender'].value_counts()
```

Out[295]:
```
male      1302
female    1023
Name: gender, dtype: int64
```

In [296]:
```
distrubution=merged_df.groupby(['gender','Hospital tier']).size()
distrubution
```

Out[296]:
```
gender  Hospital tier
female  tier - 1          88
        tier - 2         705
        tier - 3         230
male    tier - 1         212
```

```
        tier - 2        705
        tier - 3        230
male    tier - 1        212
        tier - 2        629
        tier - 3        461
dtype: int64
```

**Total females are 1023, out of 1023 88 belongs to tier - 1, 705 belongs to tier - 2 and 230 belongs to tier - 3.

**Total males are 1302, out of 1302 212 belongs to tier - 1, 629 belongs to tier - 2 and 461 belongs to tier - 3.

**Replacing Hospital tier and City tier column values with 1,2 and 3.

In [299]:
```
merged_df['Hospital tier'].unique()
```

Out[299]:
```
array(['tier - 2', 'tier - 3', 'tier - 1'], dtype=object)
```

In [300]:
```
merged_df['City tier'].unique()
```

Out[300]:
```
array(['tier - 3', 'tier - 1', 'tier - 2'], dtype=object)
```

In [301]:
```
merged_df['Hospital tier'].unique()
```

Out[301]:
```
array(['tier - 2', 'tier - 3', 'tier - 1'], dtype=object)
```

In [302]:
```
merged_df.replace('tier - 1',1,inplace=True)
merged_df.replace('tier - 2',2,inplace=True)
merged_df.replace('tier - 3',3,inplace=True)
```

In [303]:
```
merged_df.dtypes
```

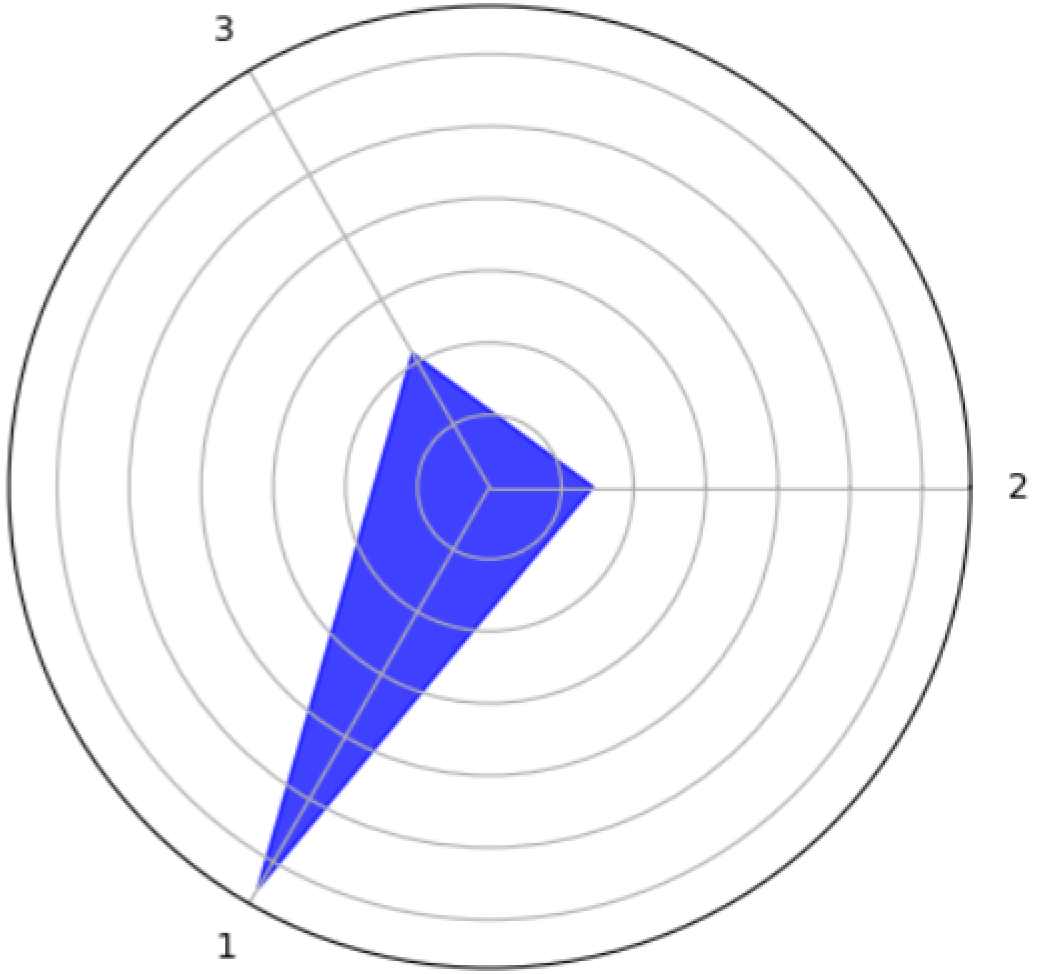
```
Out[303]: Customer ID                          object
          children                              int64
          charges                             float64
          Hospital tier                         int64
          City tier                             int64
          State ID                              int64
          BMI                                 float64
          Diabetic                              int64
          Heart Issues                         object
          Any Transplants                      object
          Cancer history                       object
          NumberOfMajorSurgeries                int64
          smoker                               object
          name                                 object
          Year                                  int64
          Month                                 int64
          Date                                  int64
          Date_of_Birth                datetime64[ns]
          age                                   int64
          gender                               object
          dtype: object
```

--Create a radar chart to showcase the median hospitalization cost for each tier of hospitals

In [309]:

```
# Select relevant columns
df = merged_df[['Hospital tier', 'charges']]

# Calculate median hospitalization cost for each tier
median_costs = df.groupby('Hospital tier')['charges'].median().tolist()

# Prepare data for radar chart
labels = df['Hospital tier'].unique()
values = [median_costs[labels.tolist().index(label)] for label in labels]

# Ensure values are non-negative
values = np.maximum(values, 0)

# Create a polar bar plot (radar chart)
fig, ax = plt.subplots(figsize=(5, 5), subplot_kw=dict(polar=True))
ax.fill(np.deg2rad(labels * 360 / len(labels)), values, color='blue', alpha=0.75)

ax.set_yticklabels([])
ax.set_xticks(np.linspace(0, 2 * np.pi, len(labels), endpoint=False))
ax.set_xticklabels(df['Hospital tier'].unique())
plt.title('Median Hospitalization Cost by Tier', size=16, color='blue', y=1.1)
plt.show()
```

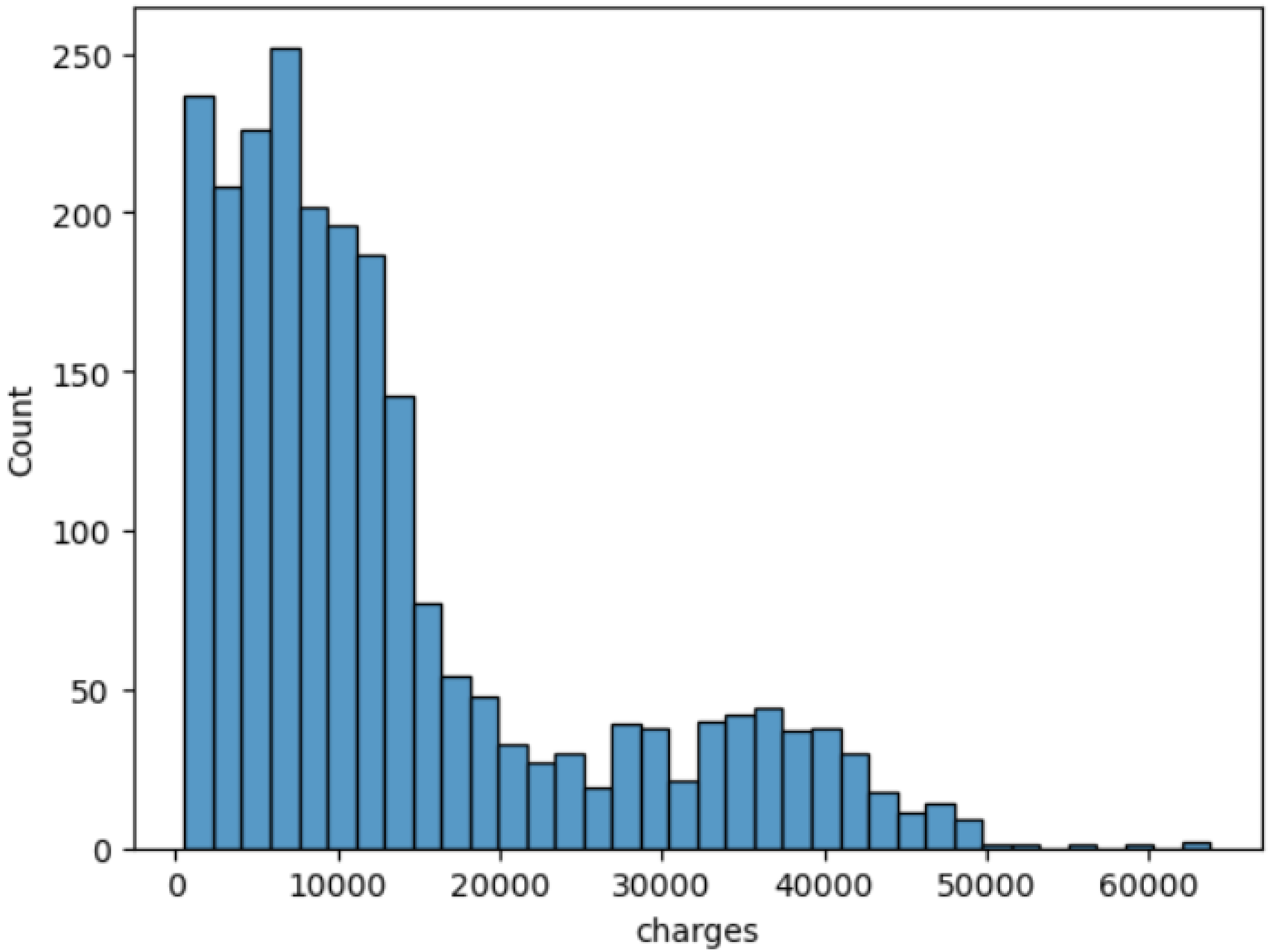
Median Hospitalization Cost by Tier



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

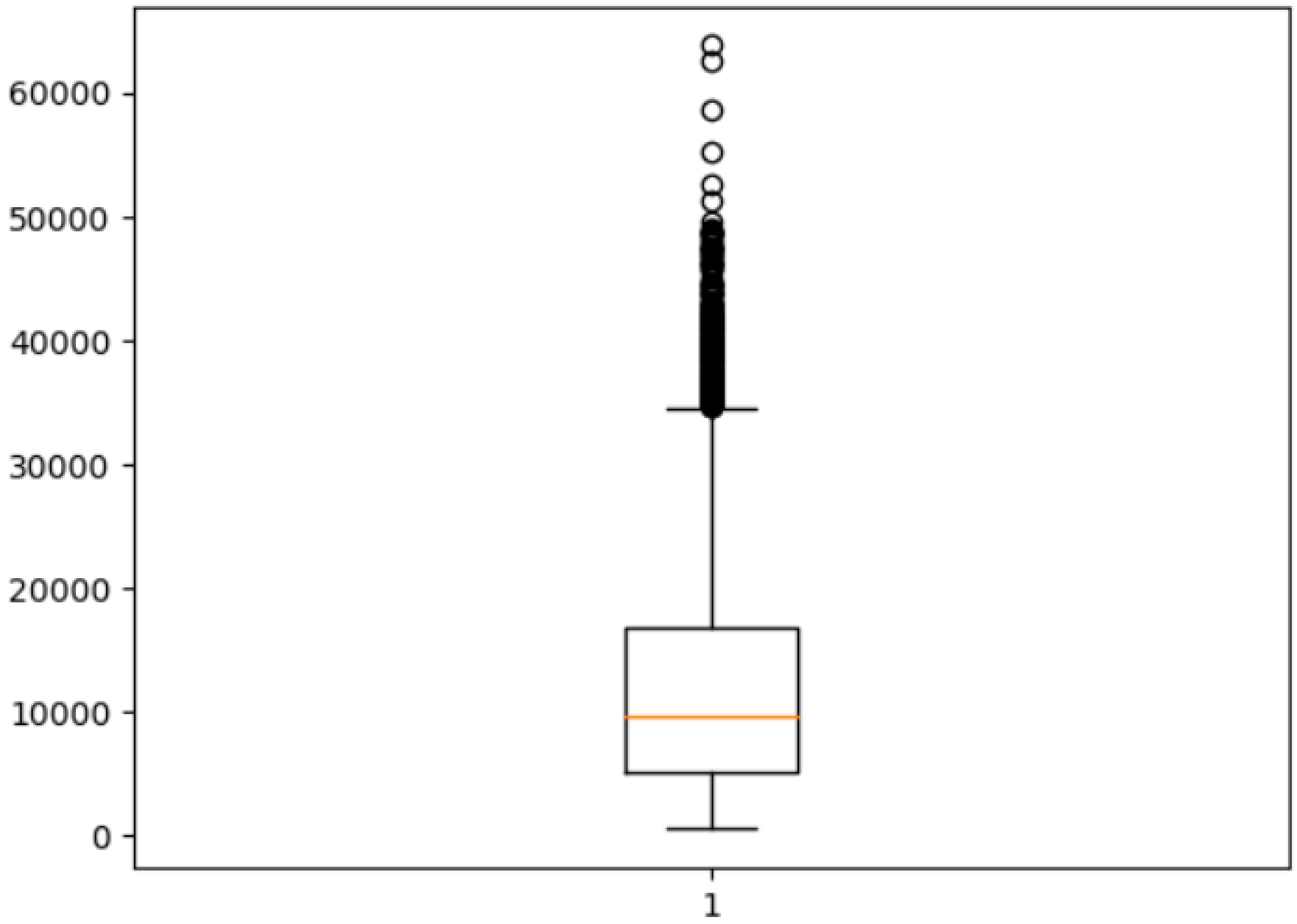
histogram using seaborn :

In [209]:
```
sns.histplot(merged_df['charges']);
```



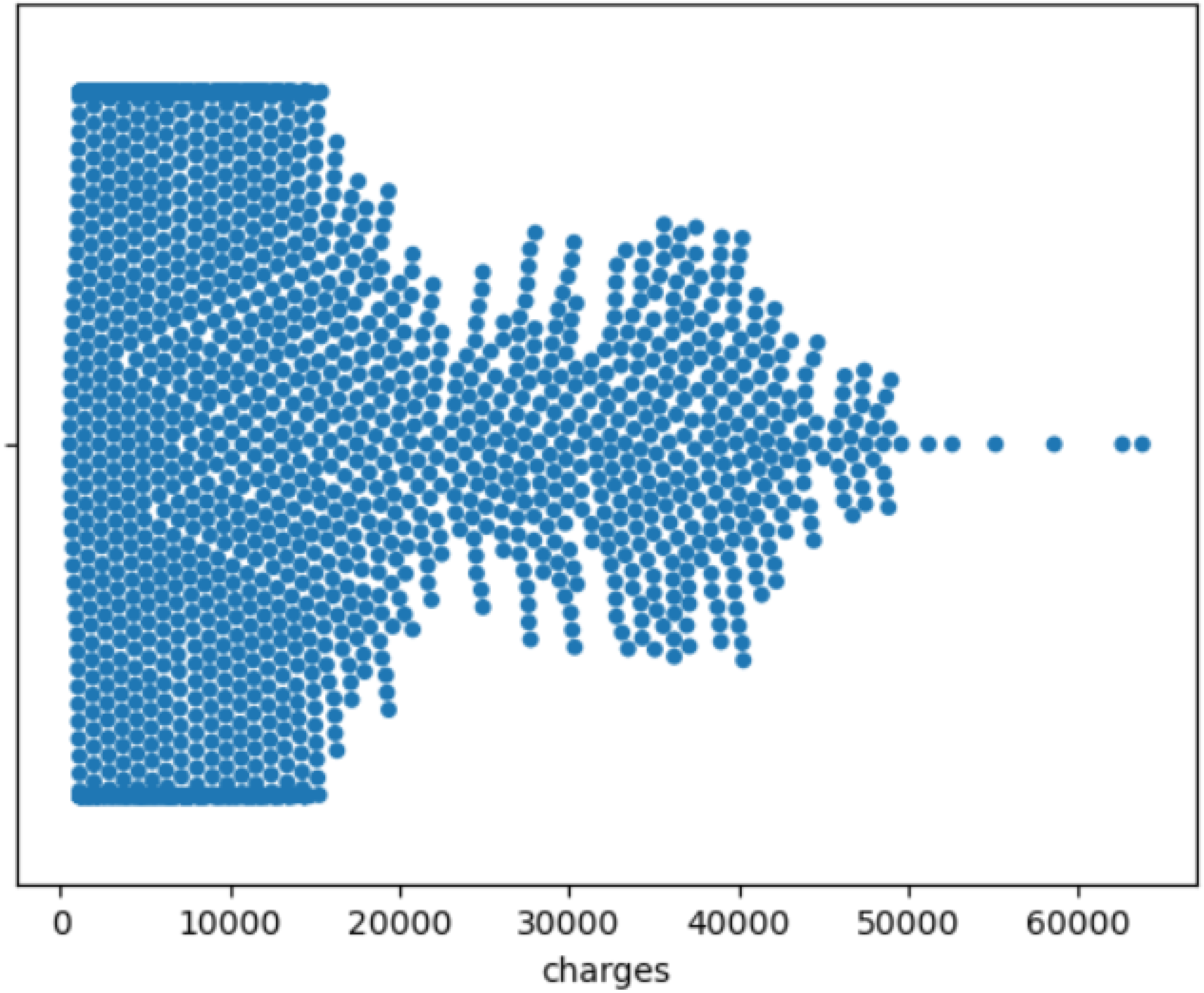
box and whisker plot using matplotlib.pyplot :

In [210]:
```
plt.boxplot(merged_df['charges']);
```



swarm plot using seaborn :

In [211]:
```
sns.swarmplot(merged_df['charges']);
```



In [310]:
```
merged_df.head(3)
```

Out[310]:

	Customer ID	children	charges	Hospital tier	City tier	State ID	BMI	Diabetic	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	name	Year	Month
0	Id2335	0	563.84	2	3	2	17.58	0	No	No	No	1	No	German, Mr. Aaron K	1992	7
1	Id2334	0	570.62	2	1	2	17.60	0	No	No	No	1	No	Rosendahl, Mr. Evan P	1992	11
2	Id2333	0	600.00	2	1	2	16.47	0	No	No	Yes	1	No	Albano, Ms. Julie	1993	6

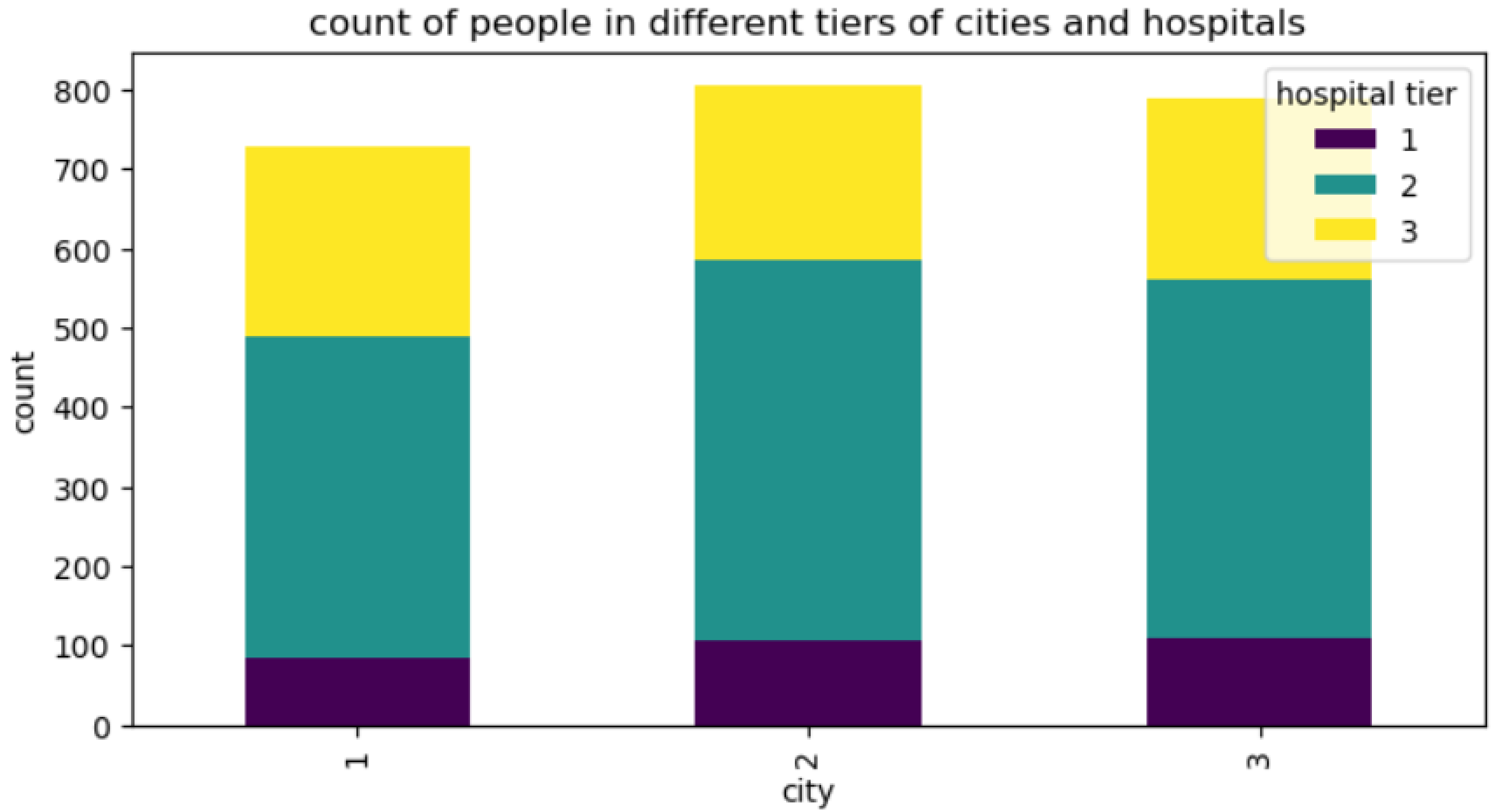
--Create a frequency table and a stacked bar chart to visualize the count of people in the different tiers of cities and hospitals

In [311]:
```
frequency_table=pd.crosstab(merged_df['City tier'],merged_df['Hospital tier'])
frequency_table
```

Out[311]:

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

In [312]:
```
ax=frequency_table.plot(kind='bar',stacked=True,colormap='viridis',figsize=(8,4))
plt.xlabel('city')
plt.ylabel('count')
plt.title('count of people in different tiers of cities and hospitals')
plt.legend(title='hospital tier');
```



** In city 2 their are more people and In city 1 their are less people

--Test the following null hypotheses for given conditions

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

In [314]:
```
merged_df.groupby('Hospital tier')[['charges']].median()
```

Out[314]:

Hospital tier	charges
1	32097.435
2	7168.760
3	10676.830

**Above, we did group by to test null hypothesis for the columns 'Hospital tier' along with 'charges' . In statement given that The average hospitalization costs for the three types of hospitals are significantly not different but we proved that The average hospitalization costs for the three types of cities are significantly different , So null hypothesis(H0) is rejected and Alternative hypothesis(H1) is accepted.

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

In [315]:
```
merged_df.groupby('City tier')[['charges']].median()
```

Out[315]:

City tier	charges
1	10027.15
2	8968.33
3	9880.07

**Above, we did group by to test null hypothesis for the columns 'City tier' along with 'charges' columns. In statement given that The average hospitalization costs for the three types of cities are significantly not different as given like that The average hospitalization costs for the three types of cities are significantly not different, So null hypothesis(H0) is accepted and Alternative hypothesis(H1) is rejected.

c. The average hospitalization cost for smokers is not significantly different from the average cost for nonsmokers

In [316]:
```
merged_df.groupby('smoker')[['charges']].median()
```

Out[316]:

smoker	charges
No	7537.160
yes	34125.475

**Above, we did group by to test null hypothesis for the columns 'smoker_yes' along with 'charges' columns. In statement given that The average hospitalization costs for smokers are significantly not different from the average cost for nonsmokers but we proved that The average hospitalization costs for thesmokers are significantly different from the average cost for nonsmokers, So null hypothesis(H0) is rejected and Alternative hypothesis(H1) is accepted.

d. Smoking and heart issues are independent

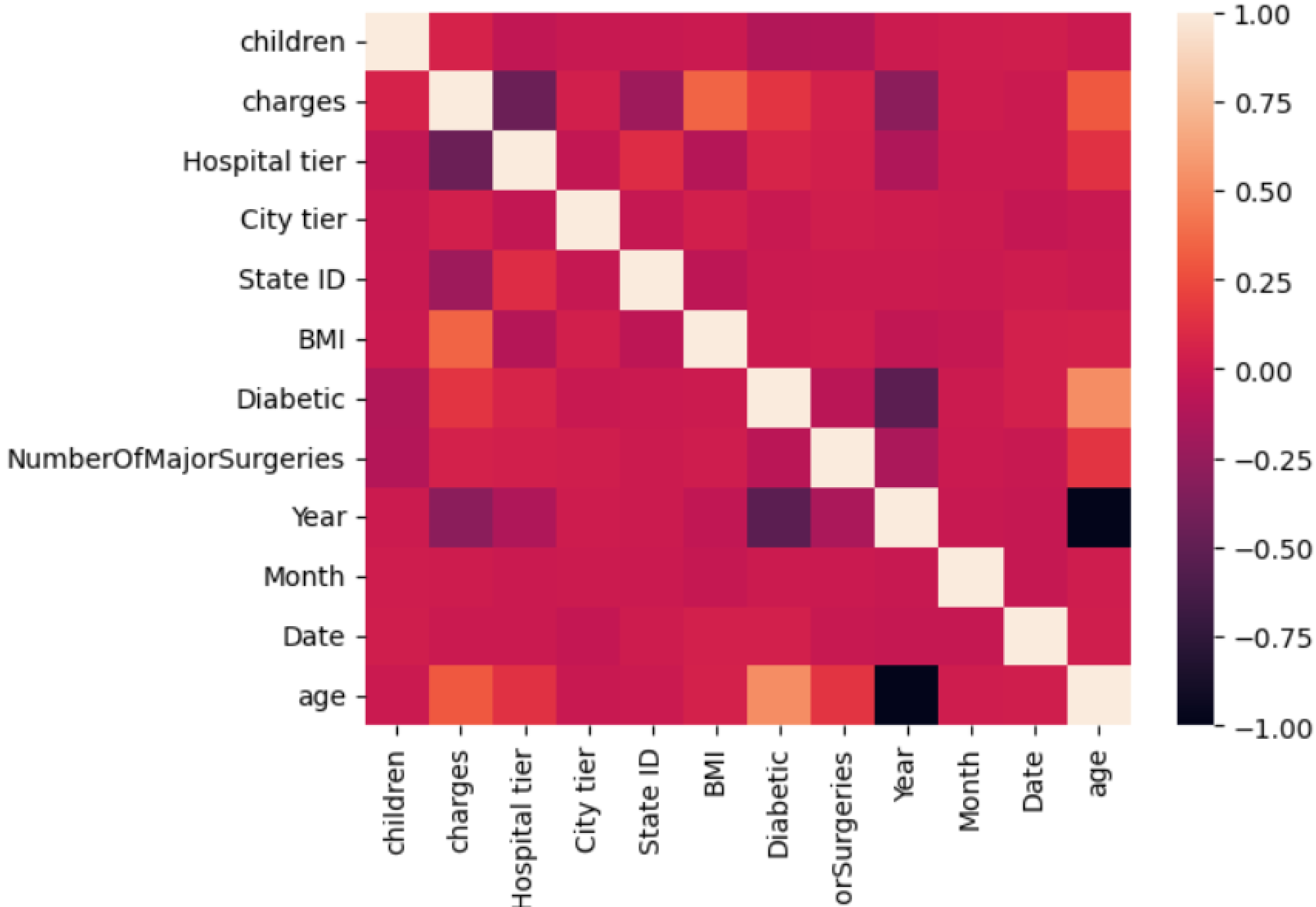
In [317]:
```
merged_df.groupby('smoker')['Heart Issues'].value_counts()
```

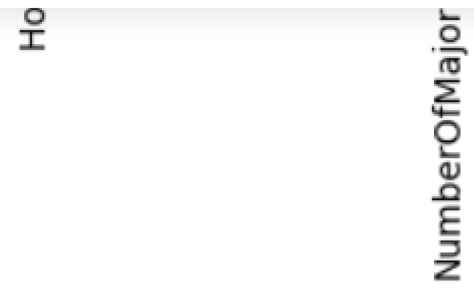
Out[317]:
```
smoker  Heart Issues
No      No              1108
        yes              731
yes     No               297
        yes              189
Name: Heart Issues, dtype: int64
```

** given that Smoking and heart issues are independent but we proved Smoking and heart issues are dependent So null hypothesis(H0) is rejected and Alternative hypothesis(H1) is accepted.

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

In [318]:
```
sns.heatmap(merged_df.corr());
```





**Correlation is high between charges and age, charges and bmi, Diabetic and age.

In [319]:
```
merged_df.head(2)
```

Out[319]:

	Customer ID	children	charges	Hospital tier	City tier	State ID	BMI	Diabetic	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	name	Year	Month
0	Id2335	0	563.84	2	3	2	17.58	0	No	No	No	1	No	German, Mr. Aaron K	1992	7
1	Id2334	0	570.62	2	1	2	17.60	0	No	No	No	1	No	Rosendahl, Mr. Evan P	1992	11

**Deleting unwanted columns

In [320]:
```
merged_df=merged_df.drop(columns=['Customer ID','name','Year','Month','Date','Date_of_Birth'])
```

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

In [321]:
```
merged_df=pd.get_dummies(merged_df,columns=['Heart Issues','Any Transplants','Cancer history','smoker','gender'])
```

In [322]:
```
merged_df.head()
```

	children	charges	Hospital tier	City tier	State ID	BMI	Diabetic	NumberOfMajorSurgeries	age	Heart Issues_No	Heart Issues_yes	Any Transplants_No	Any Transplants_yes	Cancer history_No
0	0	563.84	2	3	2	17.58	0	1	31	1	0	1	0	1
1	0	570.62	2	1	2	17.60	0	1	31	1	0	1	0	1
2	0	600.00	2	1	2	16.47	0	1	30	1	0	1	0	0
3	0	604.54	3	3	2	17.70	0	1	31	1	0	1	0	1
4	0	637.26	3	3	2	22.34	0	1	25	1	0	1	0	1

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

In [342]:
```
from sklearn.model_selection import train_test_split,KFold,StratifiedKFold,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error
```

--Use standardization

****Using StandardScaler for the columns which contain values in much different compare with other elememts. so to bring values in a similar range StandardScaler is being used.**

In [343]:
```
cols_to_scale=['BMI','age','charges']
```

In [344]:
```
scaler=StandardScaler()
```

In [345]:
```
merged_df[cols_to_scale]=scaler.fit_transform(merged_df[cols_to_scale])
```

In [346]:
```
merged_df.head(5)
```

Out[346]:

	children	charges	Hospital tier	City tier	State ID	BMI	Diabetic	NumberOfMajorSurgeries	age	Heart Issues_No	Heart Issues_yes	Any Transplants_No	Any Transplants_yes
0	0	-1.092478	2	3	2	-1.534432	0	1	-0.691386	1	0	1	0
1	0	-1.091907	2	1	2	-1.532145	0	1	-0.691386	1	0	1	0
2	0	-1.089430	2	1	2	-1.661390	0	1	-0.766187	1	0	1	0
3	0	-1.089047	3	3	2	-1.520707	0	1	-0.691386	1	0	1	0
4	0	-1.086288	3	3	2	-0.990000	0	1	-1.140192	1	0	1	0

****Now the values are in same range.**

****splitting data in x and y variables to train and test**

In [347]:
```
x=merged_df.drop(['charges'],axis=1)
y=merged_df.charges
```

****Initializing stochastic gradient descent regression algorithm for model building.**

In [348]:
```
model=SGDRegressor()
```

--Use hyperparameter tuning effectively

--Use appropriate regularization techniques to address the bias-variance trade-off

In [349]:
```
# Define hyperparameters and regularization techniques
param_grid = {
    'alpha': [1.0,2.0,3.0,4.0],
    'penalty': ['l1', 'l2'],
    'learning_rate': ['constant', 'optimal', 'invscaling', 'adaptive']
}
```

--Perform the stratified 5-fold cross-validation technique for model building and validation

In [350]:
```
kfold=KFold(n_splits=5,shuffle=True,random_state=42)
```

In [351]:
```
# Using GridSearchCV for hyperparameter tuning
grid_search=GridSearchCV(model,param_grid,cv=kfold,scoring='neg_root_mean_squared_error')
grid_search.fit(x,y)

# get the best parameters
best_params=grid_search.best_params_
best_params
```

Out[351]:
```
{'alpha': 1.0, 'learning_rate': 'adaptive', 'penalty': 'l2'}
```

In [352]:
```
# use the best hyperparameters obtained from the tuning step
best_model=SGDRegressor(**best_params)
best_model
```

Out[352]:
```
SGDRegressor(alpha=1.0, learning_rate='adaptive')
```

--Create five folds in the data, and introduce a variable to identify the folds.

--For each fold, run a for loop and ensure that 80 percent of the data is used to train the model and the remaining 20 percent is used to validate it in each iteration.

--Develop five distinct models and five distinct validation scores (root mean squared error values).

In [353]:
```
# create an SGDRegressor
best_model=SGDRegressor(alpha=1.0, learning_rate='adaptive', penalty='l2')

# Intializing the performance metricslist
fold_rmse=[]

# iterate through each fold
for fold_num,(train_indices,test_indices) in enumerate(kfold.split(x,y)):

    # Add new column 'fold' to identify the folds in dataframe
    merged_df['current_fold']=-1

    # assign the fold number to 'current_fold' for the rows corresponding to the set of the current fold
    valid_indices=test_indices[test_indices<len(merged_df)]   # ensure indices are valid
    #merged_df.iloc[valid_indices,merged_df.columns.get_loc('fold')]=fold_num

    # extract data for the currentfold using train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x.iloc[test_indices],y.iloc[test_indices],test_size=0.2,random_state=42)

    # train the model on 80% of the data
    best_model.fit(x_train,y_train)

    # validate the model on remaining 20% and caluculate performance metric
    best_model.score(x_test,y_test)
    y_pred=best_model.predict(x_test)
    rmse=np.sqrt(mean_squared_error(y_test,y_pred))

    # store the Root Mean Squared Error for this fold
    fold_rmse.append(rmse)

    print(f'model {fold_num+1}: Root Mean Squared Error - {rmse}')

# display average peformance across all folds
average_performance=sum(fold_rmse)/len(fold_rmse)
```

```
average_performance=sum(fold_rmse)/len(fold_rmse)
print(f'\n Average Root Mean Squared Error across folds: {average_performance}')
```

```
model 1: Root Mean Squared Error - 0.8081283683329464
model 2: Root Mean Squared Error - 0.7653262262955396
model 3: Root Mean Squared Error - 0.7328291957595342
model 4: Root Mean Squared Error - 0.7665826657415752
model 5: Root Mean Squared Error - 0.8052117936985456

 Average Root Mean Squared Error across folds: 0.7756156499656282
```

**** we got acccuracy 0.7760572291155676 for stochastic gradient descent optimizer**

-- Determine the variable importance scores, and identify the redundant variables

In [354]:

```
feature_importances=best_model.coef_

# craete a dataframe to dispaly feature importances
importance_df=pd.DataFrame({'Feature':x.columns,'Importance':feature_importances})
importance_df=importance_df.sort_values(by='Importance',ascending=False)

print('Feature Importances:')
importance_df
```

```
Feature Importances:
```

Out[354]:

	Feature	Importance
15	smoker_yes	0.239176
4	BMI	0.166311
7	age	0.126185
6	NumberOfMajorSurgeries	0.019144
5	Diabetic	0.018950

2	City tier	0.010148
0	children	0.009509
13	Cancer history_Yes	0.007901
10	Any Transplants_No	0.007857
16	gender_female	0.004053
9	Heart Issues_yes	0.001980
8	Heart Issues_No	-0.000503
18	current_fold	-0.001477
17	gender_male	-0.002576
11	Any Transplants_yes	-0.006380
12	Cancer history_No	-0.006424
3	State ID	-0.091361
1	Hospital tier	-0.156375
14	smoker_No	-0.237699

first variable importance score is smoker_yes with value 0.238756 and least one is smoker_No with value -0.238033.

Identify the redundant variables

In [355]:
```
# identify potentially redundant variables based on low importance scores
redundant_variables=importance_df[importance_df['Importance']<0.002816].loc[:,'Feature'].tolist()

print('\n potentially redundant variables')
redundant_variables
```

```
 potentially redundant variables
```

Out[355]:
```
['Heart Issues_yes',
 'Heart Issues_No',
 'current_fold',
 'gender_male',
 'Any Transplants_yes',
 'Cancer history_No',
 'State ID',
 'Hospital tier',
 'smoker_No']
```

--Use random forest and extreme gradient boosting for cost prediction, share your cross validation results, and calculate variable importance scores

random forest:

In [356]:
```
# random forest model
rf_model=RandomForestRegressor(n_estimators=100,max_depth=5,random_state=42)

# cross validation results for random forest
scores=cross_val_score(rf_model,x,y,cv=5,scoring='neg_root_mean_squared_error')

mean_rmse=np.sqrt(-scores)

print('Random Forest cross validation results:')
print('RMSE Scores:',scores)
print('Average RMSE:',np.mean(mean_rmse))


# fit random forest model on the entire dataset for variable impotance
rf_model.fit(x,y)

rf_feature_importance=rf_model.feature_importances_
rf_importance_df=pd.DataFrame({'Feature':x.columns,'Importance':rf_feature_importance})
rf_importance_df=rf_importance_df.sort_values(by='Importance',ascending=False)
```

```
rf_importance_df
```

```
Random Forest cross validation results:
RMSE Scores: [-0.52636761 -0.37172947 -0.40340656 -0.61888638 -1.59051095]
Average RMSE: 0.8036398089379512
```

Out[356]:

	Feature	Importance
15	smoker_yes	0.483518
14	smoker_No	0.285806
4	BMI	0.110902
7	age	0.087149
1	Hospital tier	0.013550
3	State ID	0.009283
0	children	0.009272
12	Cancer history_No	0.000126
13	Cancer history_Yes	0.000107
2	City tier	0.000106
6	NumberOfMajorSurgeries	0.000061
16	gender_female	0.000033
5	Diabetic	0.000026
9	Heart Issues_yes	0.000022
11	Any Transplants_yes	0.000017
8	Heart Issues_No	0.000015
17	gender_male	0.000007
10	Any Transplants_No	0.000000
18	current_fold	0.000000

extreme gradient boosting:

In [357]:
```
# random forest model
xgb_model=XGBRegressor(n_estimators=100,max_depth=5,random_state=42)

# cross validation results for random forest
scores=cross_val_score(xgb_model,x,y,cv=5,scoring='neg_root_mean_squared_error')

mean_rmse=np.sqrt(-scores)

print('xgboost cross validation results:')
print('RMSE Scores:',scores)
print('Average RMSE:',np.mean(mean_rmse))


# fit xgboost model on the entire dataset for variable impotance
xgb_model.fit(x,y)

xgb_feature_importance=xgb_model.feature_importances_
xgb_importance_df=pd.DataFrame({'Feature':x.columns,'Importance':xgb_feature_importance})
xgb_importance_df=xgb_importance_df.sort_values(by='Importance',ascending=False)
xgb_importance_df
```

```
xgboost cross validation results:
RMSE Scores: [-0.95820253 -0.3716196  -0.4357029  -0.70415023 -1.52681228]
Average RMSE: 0.8646682607452852
```

Out[357]:

	Feature	Importance
14	smoker_No	0.925316
7	age	0.019799
4	BMI	0.015307
1	Hospital tier	0.009637
3	State ID	0.006281

0	children	0.005610
16	gender_female	0.004999
12	Cancer history_No	0.003594
6	NumberOfMajorSurgeries	0.002641
2	City tier	0.002153
8	Heart Issues_No	0.001850
5	Diabetic	0.001541
10	Any Transplants_No	0.001273
15	smoker_yes	0.000000
17	gender_male	0.000000
9	Heart Issues_yes	0.000000
13	Cancer history_Yes	0.000000
11	Any Transplants_yes	0.000000
18	current_fold	0.000000

Average RMSE of random forest model is 0.8041491006399605 and Average RMSE of xgboost model is 0.8645686797212473 So we got Average RMSE of xgboost model is greater than Average RMSE of random forest model.

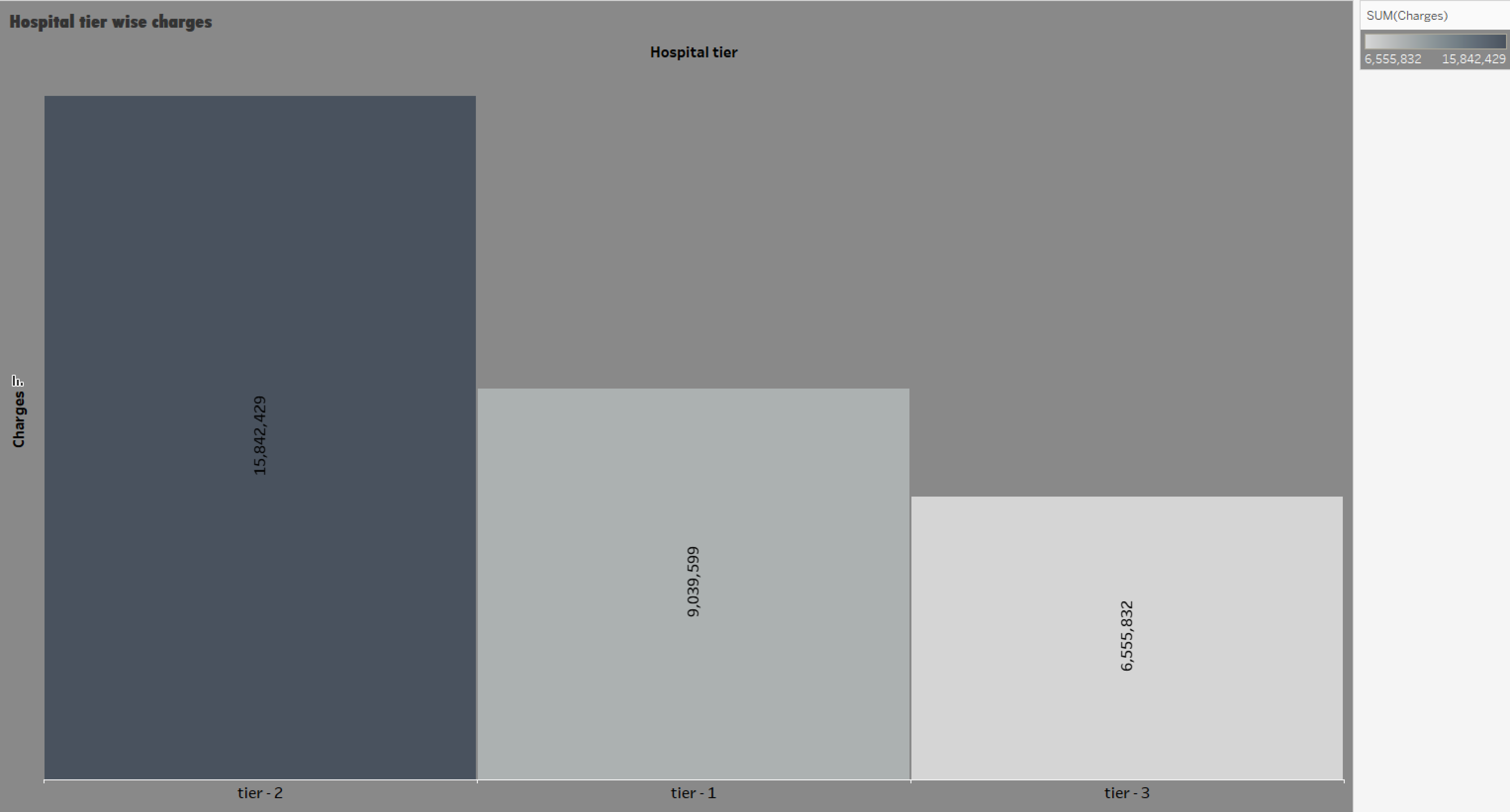
** Random forest model is predicting better than xgboost model and stochastic gradient descent optimizer.**

charges according to state ID and city tier

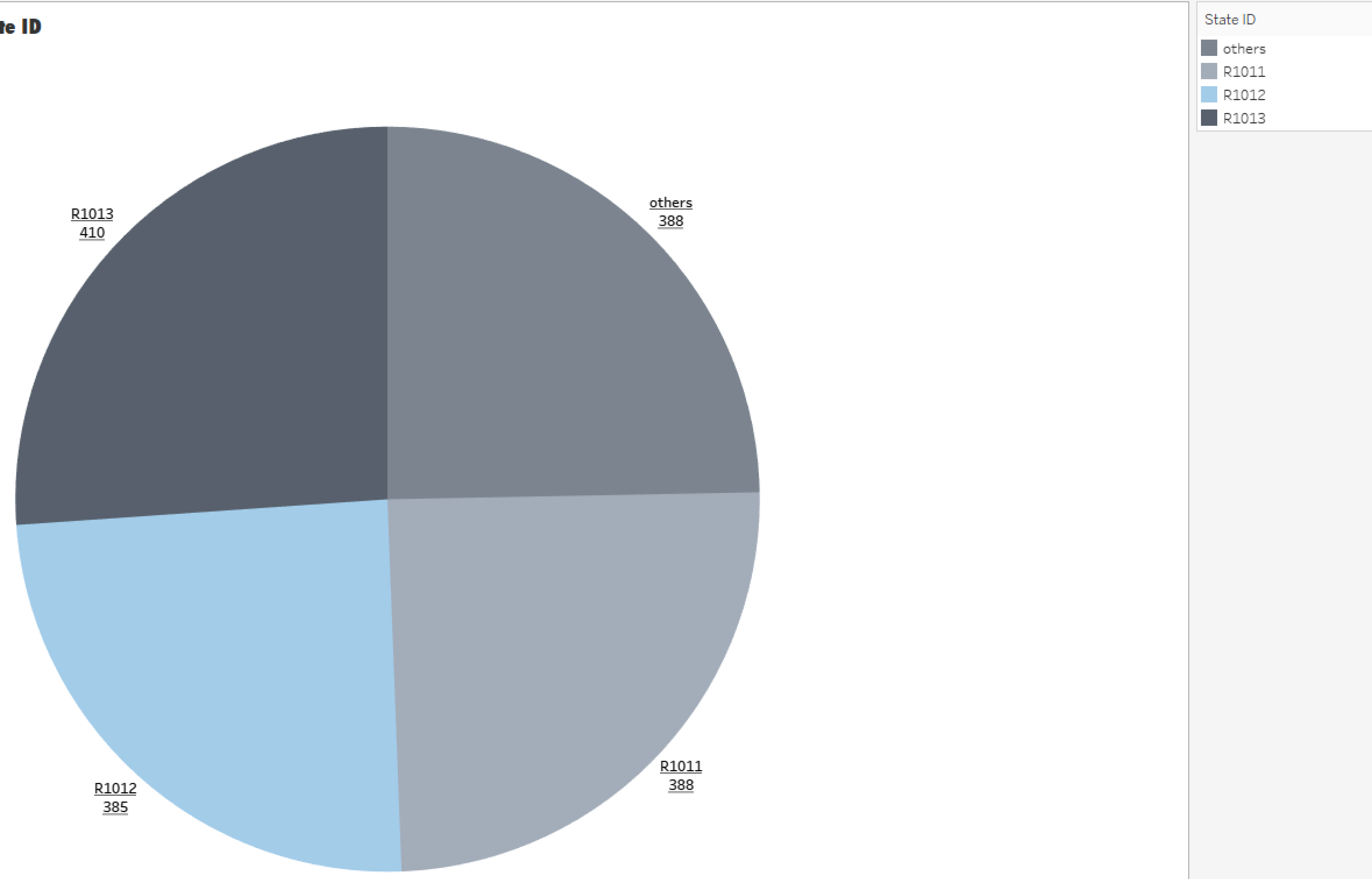
City tier	State ID: others	R1011	R1012	R1013
tier - 1	2,342,341	3,043,887	2,359,849	1,738,193
tier - 2	2,209,076	3,955,583	2,160,554	2,546,627
tier - 3	2,456,656	4,174,156	2,327,528	2,123,412

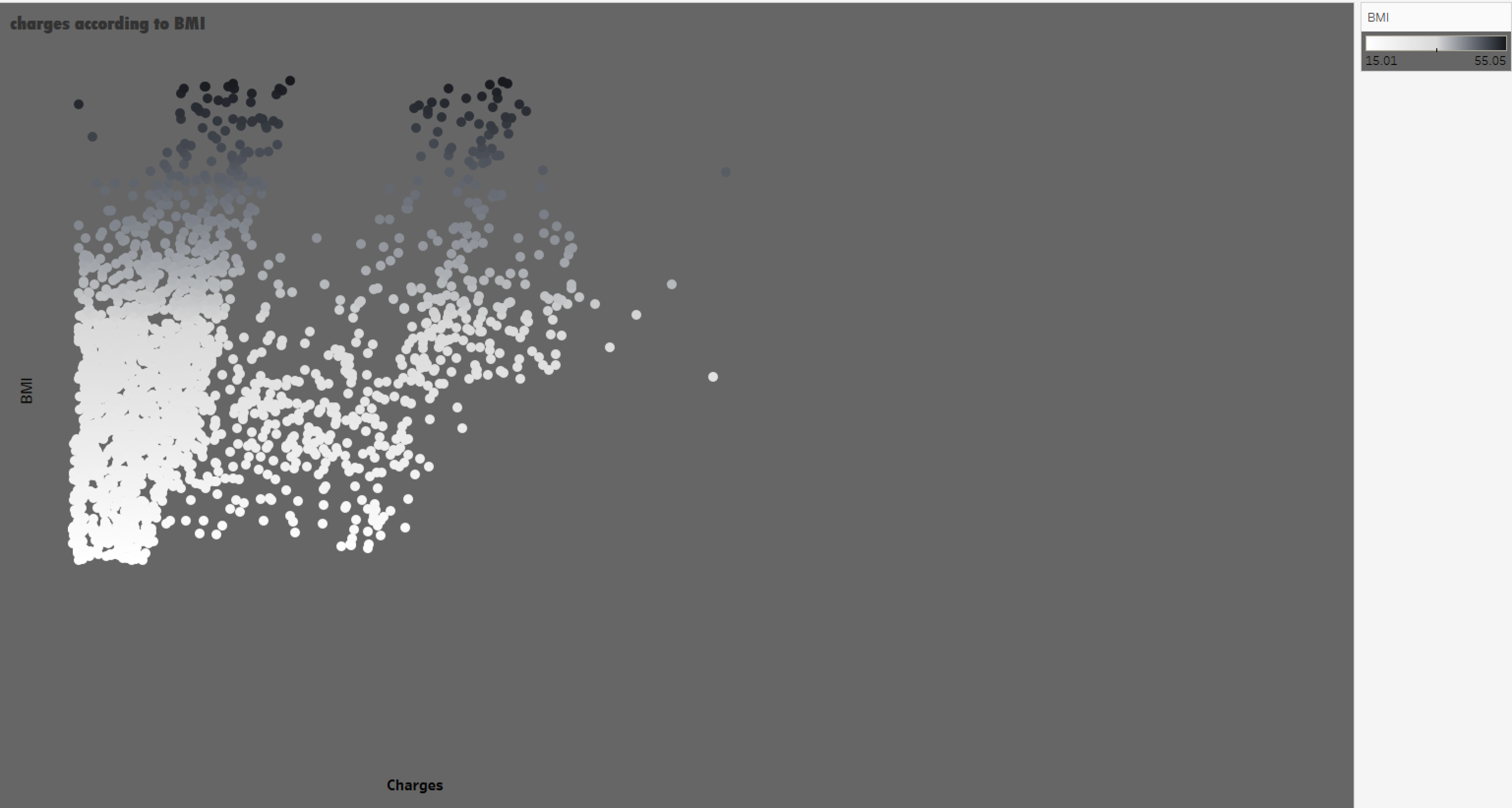
SUM(Charges)

1,738,193 4,174,156

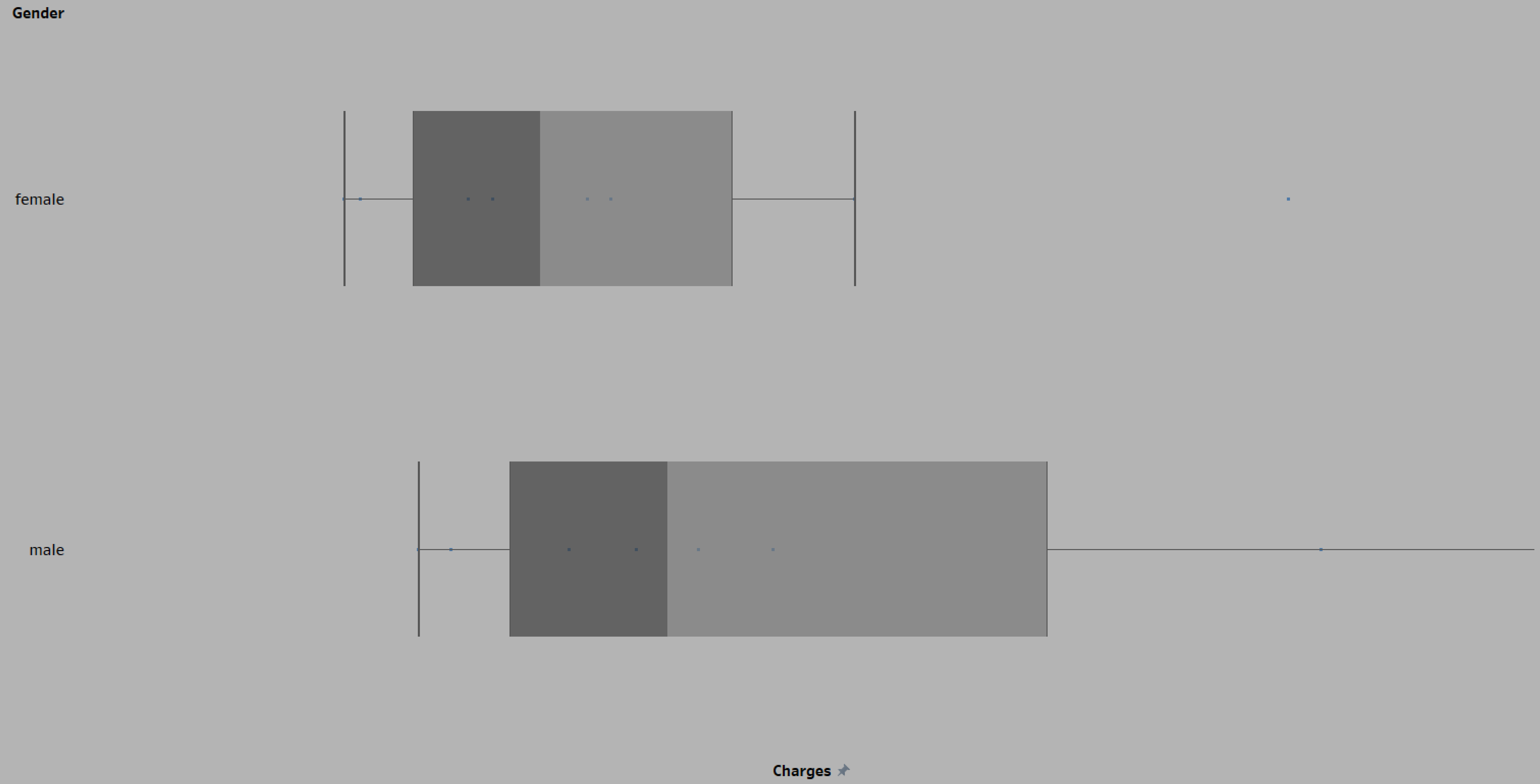

Hospital tier wise charges
Hospital tier
Charges
15,842,429
9,039,599
6,555,832
tier - 2
tier - 1
tier - 3
SUM(Charges)
6,555,832
15,842,429

Number of major surgeries according to state ID

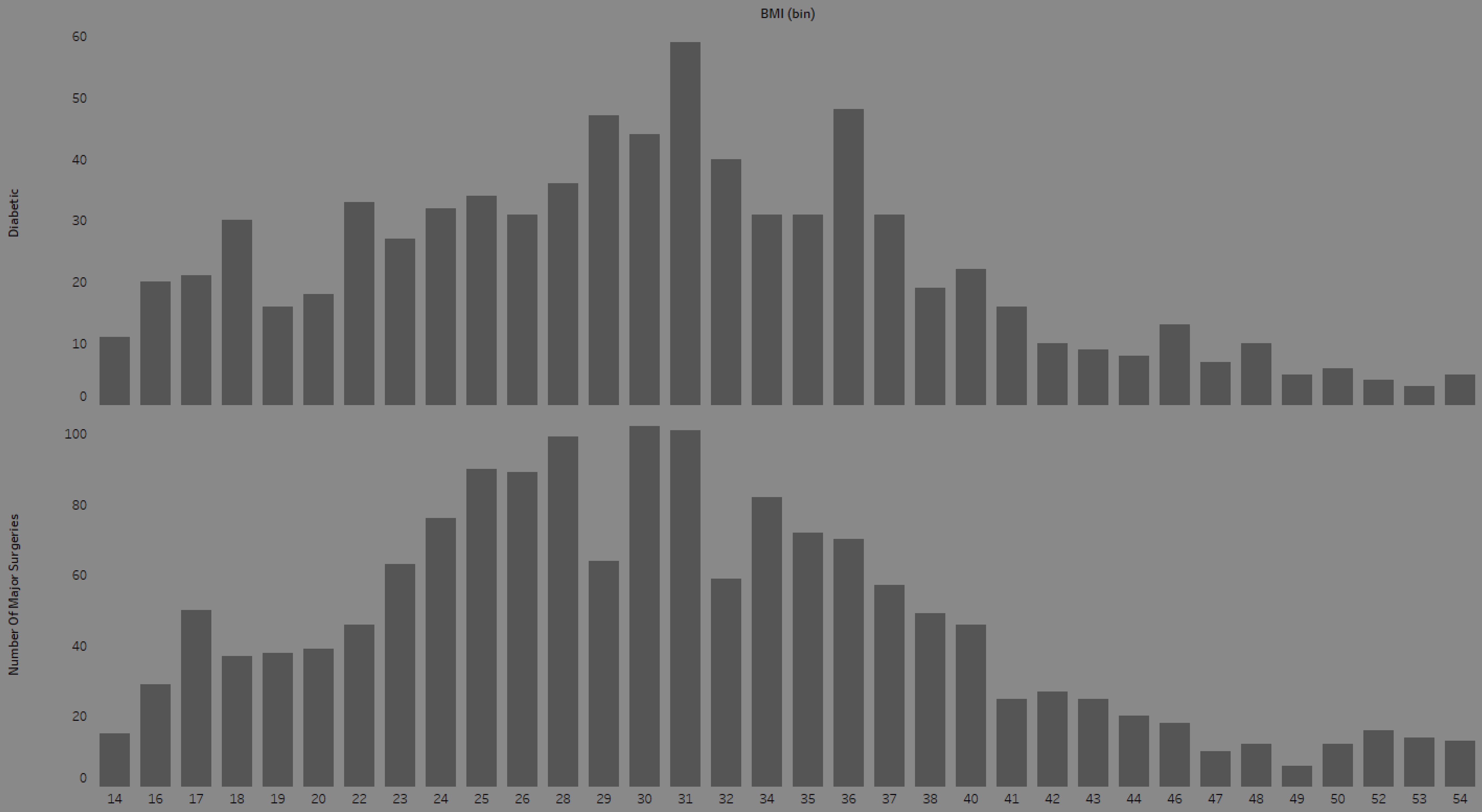



charges according to BMI
BMI
Charges
BMI
15.01
55.05

charges according to gender



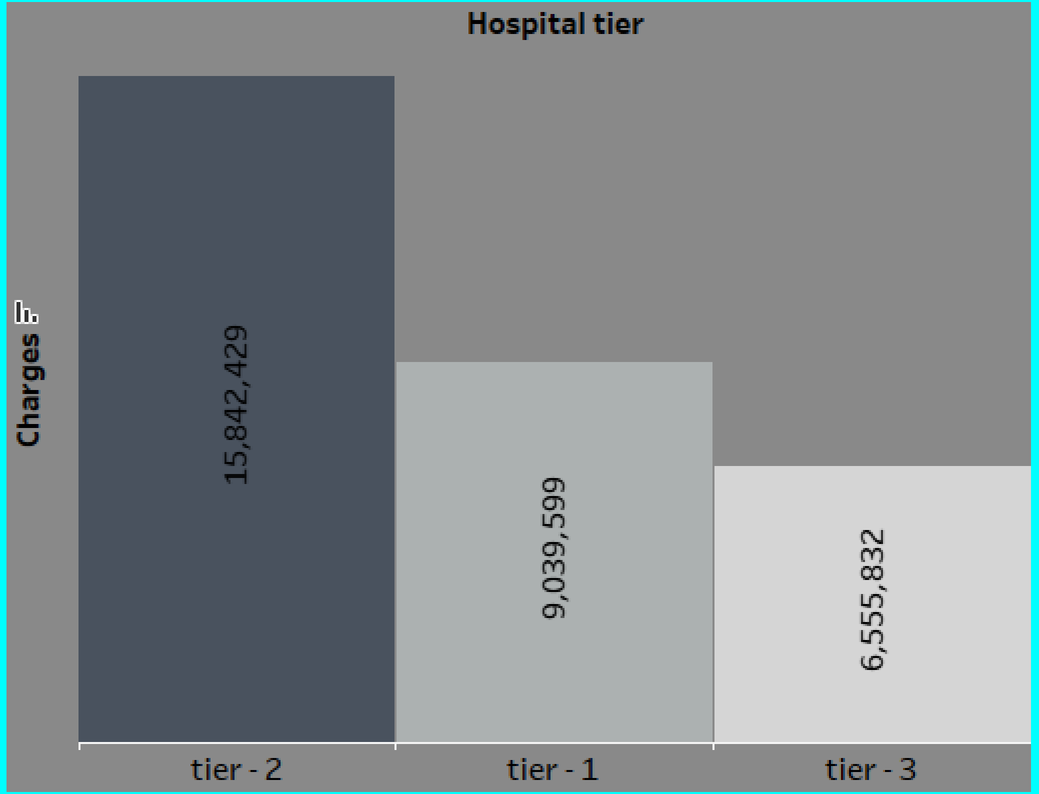
BMI with Diabetic and Major surgeries



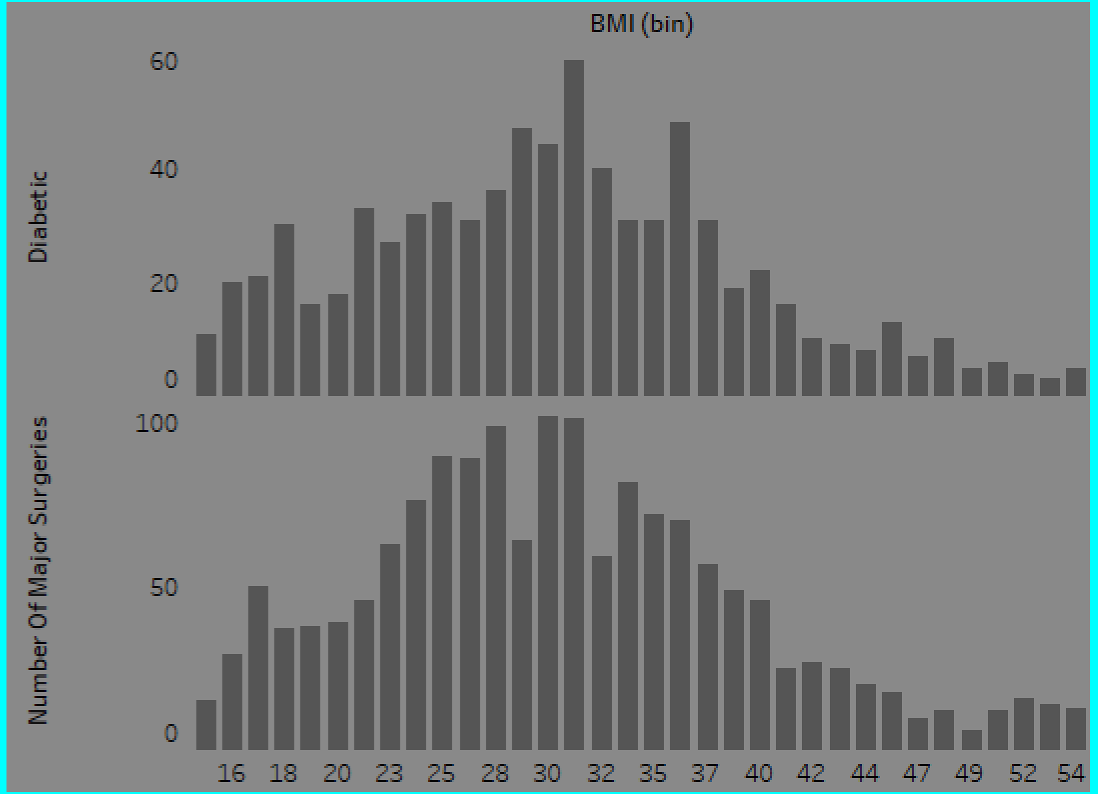
charges according to state ID and city tier

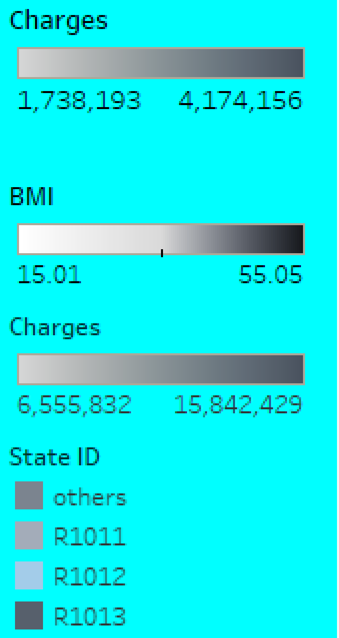
	State ID			
City tier	others	R1011	R1012	R1013
tier - 1	2,342,341	3,043,887	2,359,849	1,738,193
tier - 2	2,209,076	3,955,583	2,160,554	2,546,627
tier - 3	2,456,656	4,174,156	2,327,528	2,123,412

Hospital tier wise charges

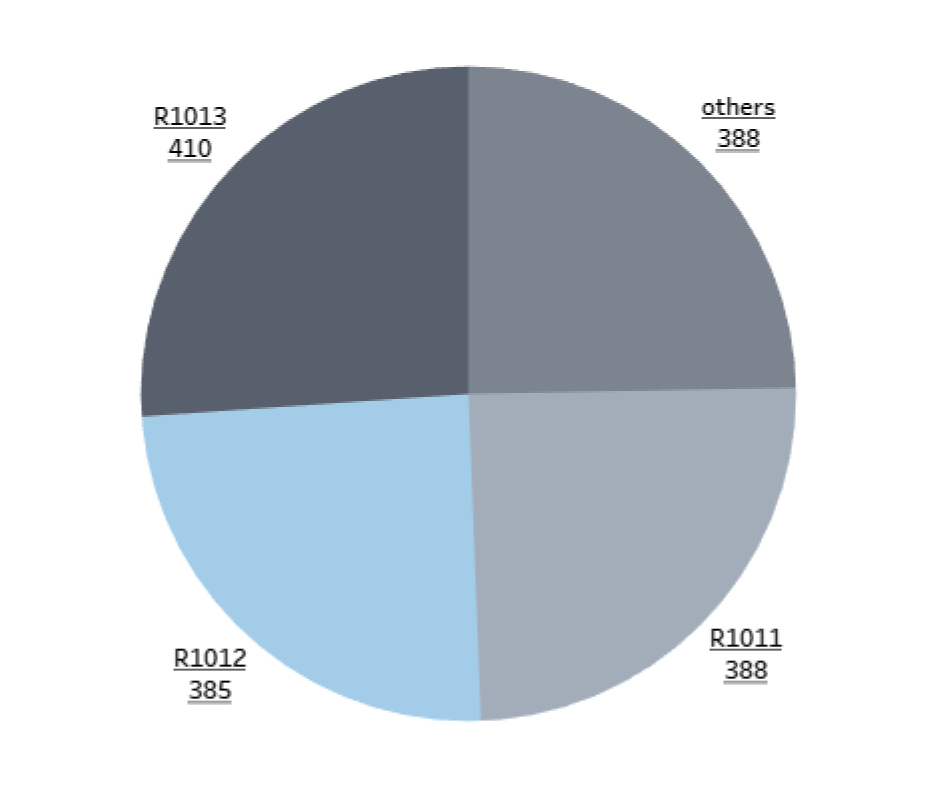


BMI with Diabetic and Major surgeries

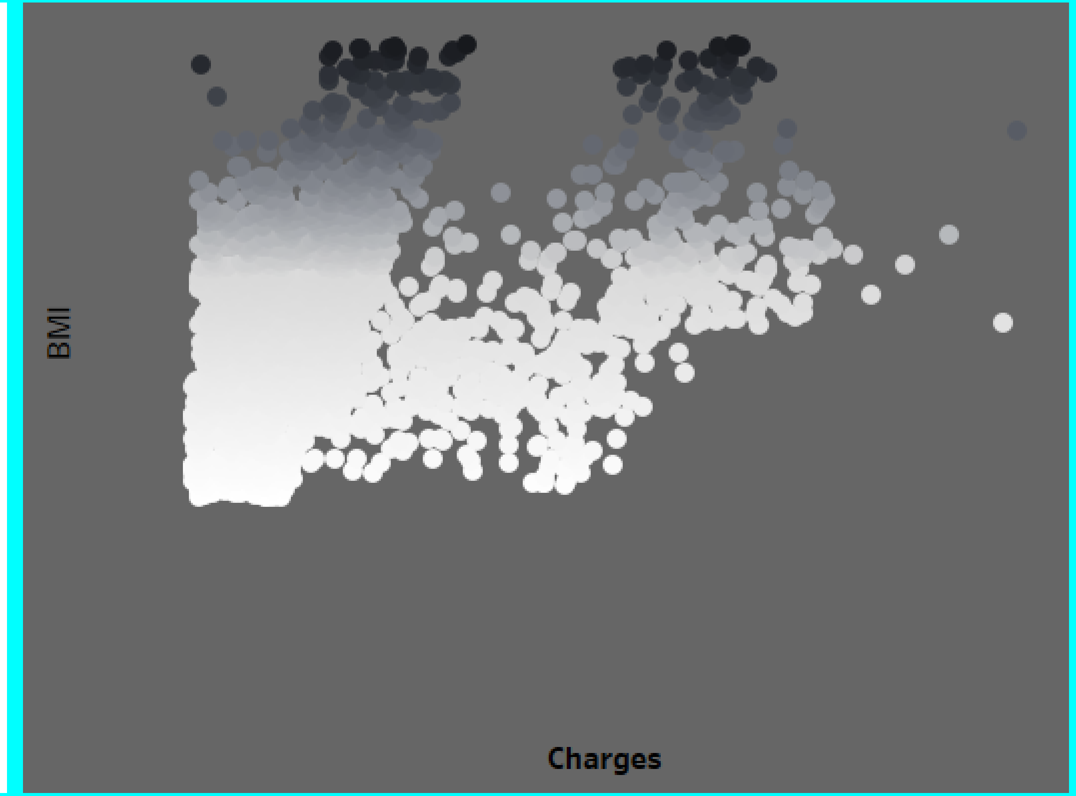




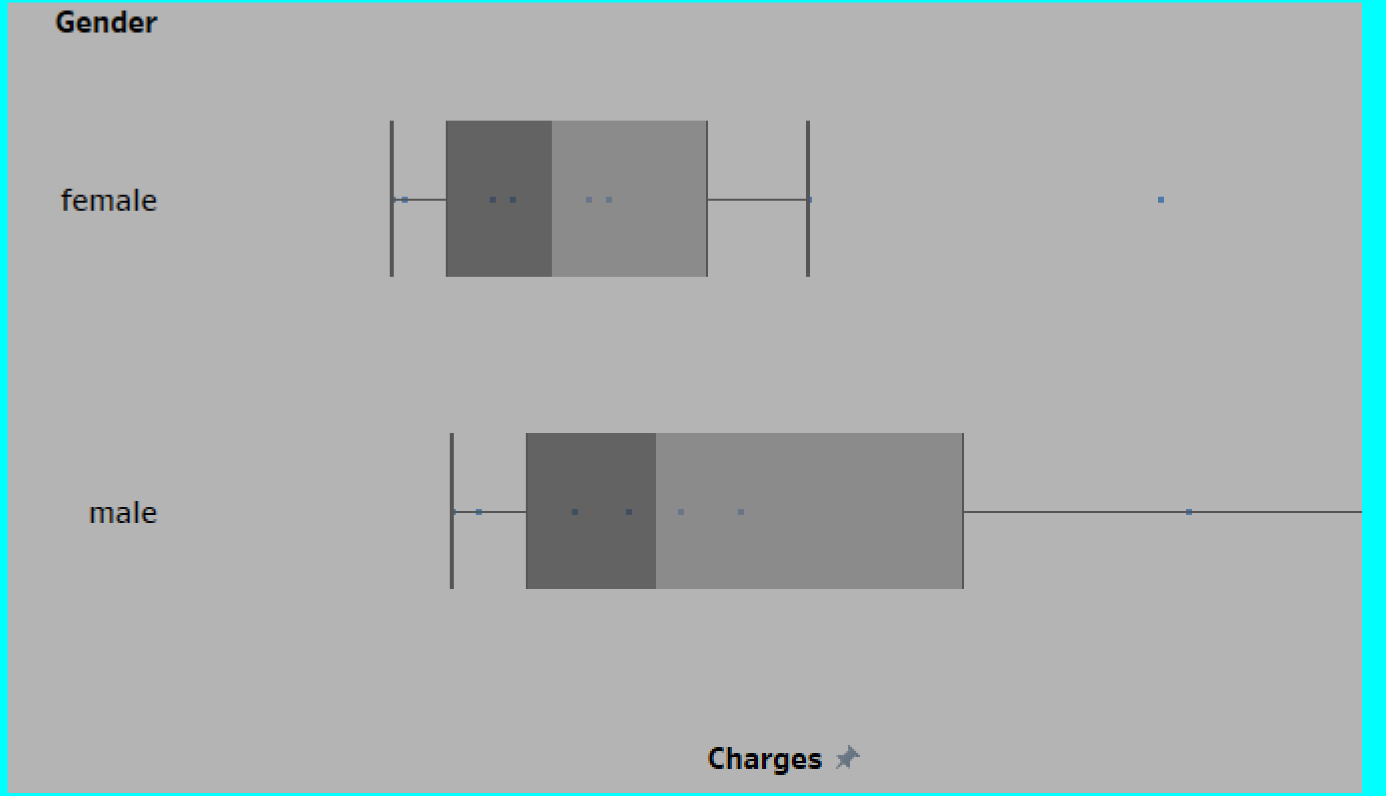
Number of major surgeries according to state ID



charges according to BMI



charges according to gender



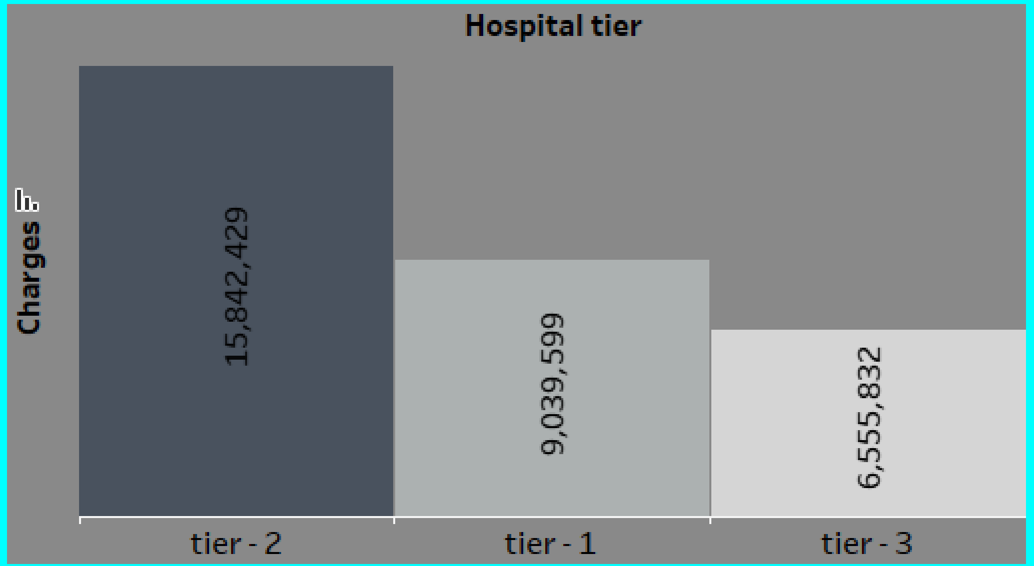
Story telling for Health care Analysis data

I have created multiple visulizations using 'Hospitalization and Medicalexamination details.csv' file. Firstly created heatmap using state ID,city tier and charges variabls then Bar graph with charges and city tier, ofter this histogram with BMI,diabetic,and Number of major surgeries,fourth visualization would be Pie chart with state ID and Number of major surgeries, fifth one is Scatter plot with BMI and Charges variables and the last visualization is Box and whisker plot using gender and charges variables . Ofter creating these visualizations created Dashboard to view all the sheets at one place and to get better understanding.

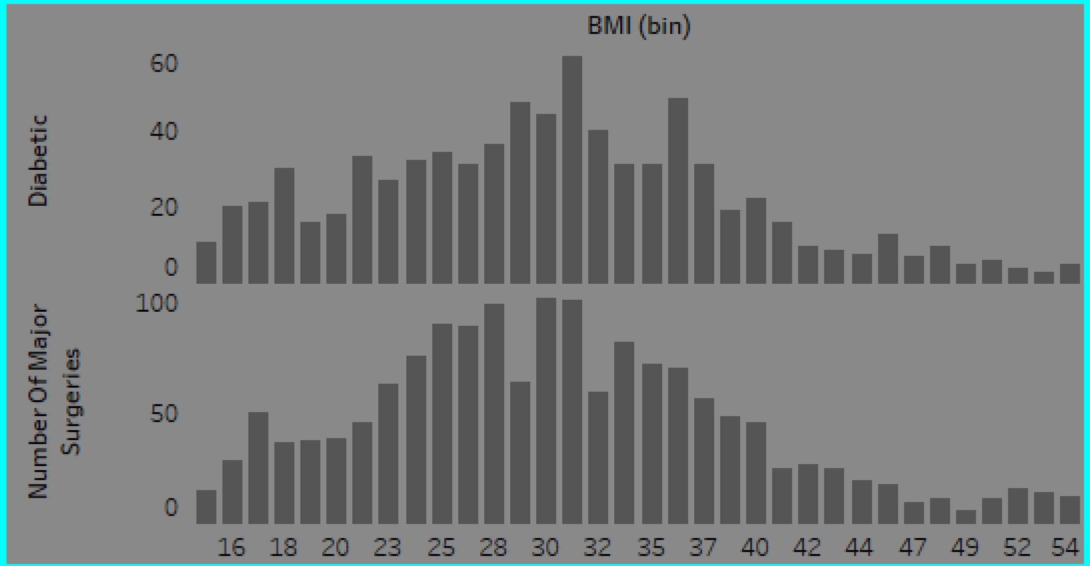
charges according to state ID and city tier

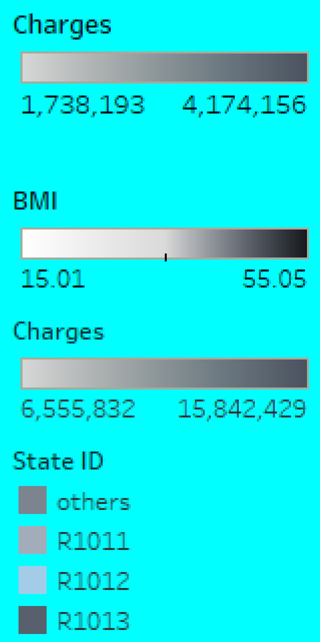
	State ID			
City tier	others	R1011	R1012	R1013
tier - 1	2,342,341	3,043,887	2,359,849	1,738,193
tier - 2	2,209,076	3,955,583	2,160,554	2,546,627
tier - 3	2,456,656	4,174,156	2,327,528	2,123,412

Hospital tier wise charges

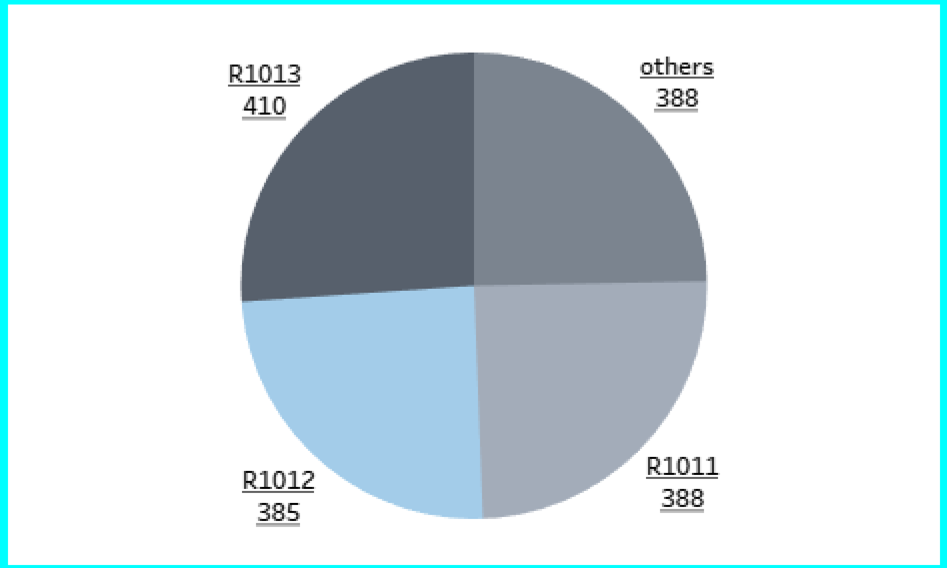


BMI with Diabetic and Major surgeries

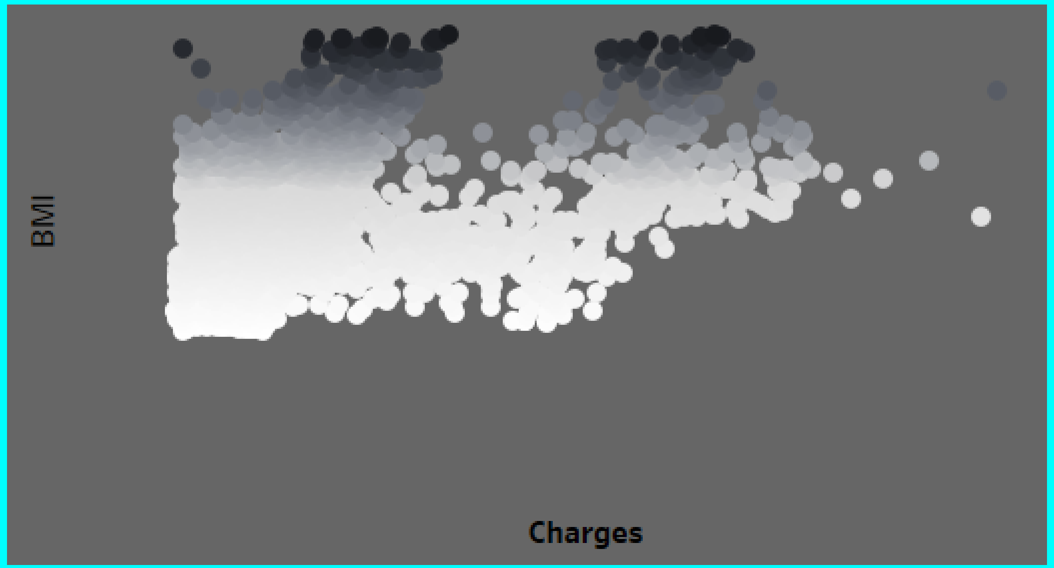




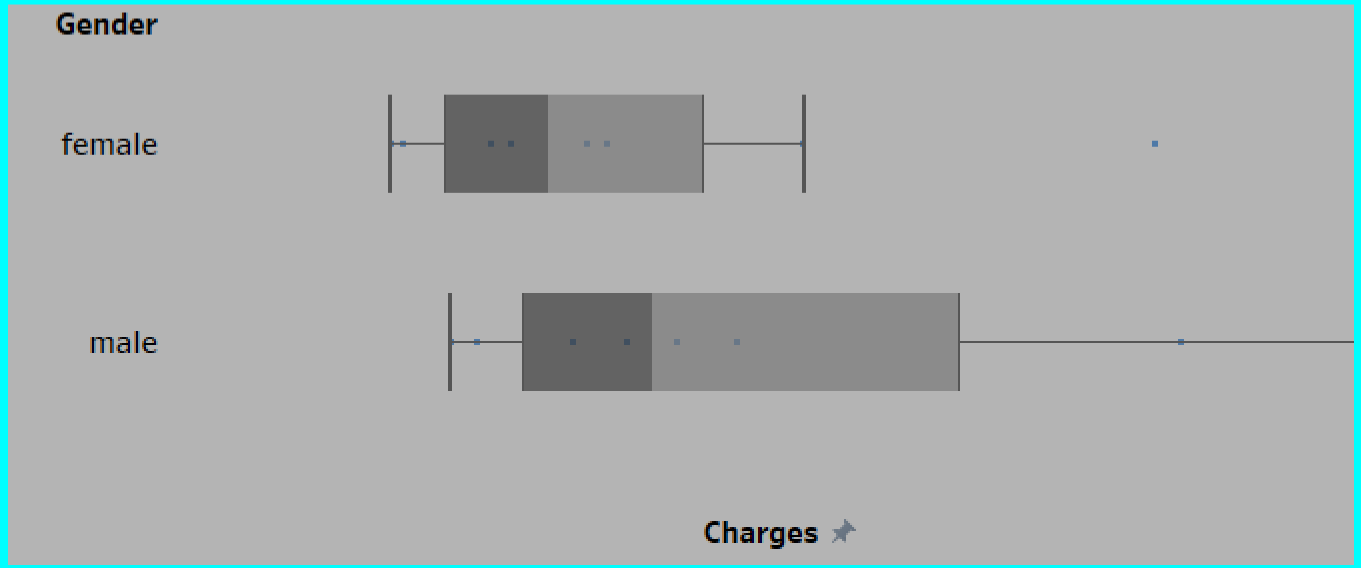
Number of major surgeries according to state ID



charges according to BMI



charges according to gender



Limit to 1000 rows

```sql
/* Merge the two tables by first identifying the columns in the data tables */
CREATE TABLE hopitalization_details_and_medical_examinations AS
SELECT
    hopitalization_details.Customer_ID,
    hopitalization_details.year,
    hopitalization_details.month,
    hopitalization_details.date,
    hopitalization_details.children,
    hopitalization_details.charges,
    hopitalization_details.Hospital_tier,
    hopitalization_details.City_tier,
    hopitalization_details.State_ID,
    medical_examinations.BMI,
    medical_examinations.HBA1C,
    medical_examinations.Heart_Issues,
    medical_examinations.Any_Transplants,
    medical_examinations.Cancer_history,
    medical_examinations.NumberOfMajorSurgeries,
    medical_examinations.smoker
FROM hopitalization_details
INNER JOIN medical_examinations  ON hopitalization_details.Customer_ID = medical_examinations.Customer_ID;
select * from hopitalization_details_and_medical_examinations
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Customer_ID	year	month	date	children	charges	Hospital_tier	City_tier	State_ID	BMI	HBA1C	Heart_Issues	Any_Transplants	Cancer_history	NumberOfMajorSurgeries	smoker
Id1	1968	Oct	12	0	63770	tier - 1	tier - 3	R1013	47	7	No	No	No	No major surgery	yes
Id10	1978	Dec	29	0	48885	tier - 1	tier - 2	R1013	38	11	No	No	No	No major surgery	yes
Id100	1977	Jun	27	2	40284	tier - 1	tier - 3	R1012	48	5	No	No	No	No major surgery	yes
Id1000	1989	Dec	17	3	11250	tier - 3	tier - 2	R1026	39	4	No	No	No	No major surgery	No
Id1001	1969	Dec	30	2	11244	tier - 3	tier - 1	R1016	26	6	yes	No	Yes	1	No
Id1002	1976	Jun	28	2	11217	tier - 3	tier - 2	R1025	31	6	yes	No	No	No major surgery	No
Id1003	1970	Jun	14	2	11188	tier - 3	tier - 2	R1012	32	7	yes	No	No	2	No
Id1004	1972	Sep	3	0	11186	tier - 3	tier - 2	R1021	31	5	No	No	No	2	No
Id1005	1966	Aug	6	0	11165	tier - 3	tier - 1	R1016	26	6	yes	No	No	2	No
Id1006	1969	Jun	25	2	11164	tier - 3	tier - 2	R1011	36	5	yes	No	Yes	1	No
Id1007	1969	Nov	30	2	11151	tier - 3	tier - 2	R1011	27	5	yes	No	Yes	1	No
Id1008	1980	Aug	20	2	11103	tier - 3	tier - 1	R1021	34	5	No	No	No	No major surgery	No
Id1009	1966	Jul	5	0	11094	tier - 3	tier - 1	R1013	42	5	yes	No	No	2	No
Id101	1981	Oct	4	1	40274	tier - 1	tier - 3	R1013	36	8	yes	No	No	No major surgery	yes
Id1010	1966	Sep	9	0	11091	tier - 3	tier - 1	R1013	40	6	yes	No	No	2	No
Id1011	1972	Oct	7	3	11086	tier - 3	tier - 2	R1012	28	5	No	No	No	2	No
Id1012	1967	Sep	4	0	11083	tier - 3	tier - 2	R1012	27	10	yes	No	No	No major surgery	No

Result Grid
Form Editor
Field Types
Query Stats
Execution Plan

Limit to 1000 rows

```sql
/*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
     AVG(children) AS average_children,
     AVG(Bmi) AS average_bmi,
     AVG(charges) AS average_charges
FROM hopitalization_details_and_medical_examinations
WHERE hba1c < 6.5 AND heart_issues = 'yes';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

average_children	average_bmi	average_charges
1.0867	31.0483	13115.8983

Result Grid

```sql
/* Find the average hospitalization cost for each hospital tier and each city level */
SELECT
    hospital_tier,
    city_tier,
    AVG(charges) AS average_charges
FROM hopitalization_details_and_medical_examinations
GROUP BY hospital_tier, city_tier;
```

hospital_tier	city_tier	average_charges
tier - 1	tier - 3	31893.8919
tier - 1	tier - 2	28788.4579
tier - 3	tier - 2	9283.4550
tier - 3	tier - 1	9775.3802
tier - 3	tier - 3	9342.1930
tier - 1	tier - 1	29519.6512
tier - 2	tier - 3	12101.2031
tier - 2	tier - 1	11515.4218
tier - 2	tier - 2	11973.6597
tier - 3	?	770.0000
?	tier - 3	700.0000

Limit to 1000 rows

```sql
/* Determine the number of people who have had major surgery with a history of cancer */
SELECT COUNT(NumberOfMajorSurgeries > 0) AS number_of_people
FROM hopitalization_details_and_medical_examinations
WHERE  cancer_history = 'yes';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

number_of_people
391

Result Grid

Limit to 1000 rows

```sql
/* Determine the number of tier-1 hospitals in each state */
SELECT
	state_id,
	COUNT(Hospital_tier='tier-1') AS number_of_tier1_hospitals
FROM hopitalization_details_and_medical_examinations
GROUP BY state_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

state_id	number_of_tier1_hospitals
R1013	611
R1012	575
R1026	84
R1016	64
R1025	40
R1021	70
R1011	574
R1024	159
R1023	38
R1019	26
R1022	14
R1017	36
R1015	12
R1020	6
R1018	9
R1014	13
?	2

Result Grid

Form Editor

Field Types

Query Stats

Execution Plan