Healthcare Insurance Analysis

In [1]:
```
# Let's import the necessary library.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
# Let's remove the unnecessary warnings.
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
# 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 [4]:
```
cust_details.head()
```

Out[4]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID
0	Id1	1968	Oct	12	0	63770.43	tier - 1	tier - 3	R1013
1	Id2	1977	Jun	8	0	62592.87	tier - 2	tier - 3	R1013
2	Id3	1970	?	11	3	60021.40	tier - 1	tier - 1	R1012
3	Id4	1991	Jun	6	1	58571.07	tier - 1	tier - 3	R1024
4	Id5	1989	Jun	19	0	55135.40	tier - 1	tier - 2	R1012

In [5]:
```
cust_details.shape
```

Out[5]:
```
(2335, 9)
```

In [6]:
```
medical_details.head()
```

Out[6]:

	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 [7]:
```
medical_details.shape
```

Out[7]:
```
(2335, 8)
```

In [8]:
```
cust_name.head()
```

Out[8]:

	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 [9]:
```
cust_name.shape
```
Out[9]: (2335, 2)

Project Task: Week 1

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

In [10]:
```
# Now combining the data so that all information could be examine in once go through.
cust_df1 = pd.merge(cust_name, cust_details, on = "Customer ID")
cust_df1.head()
```
Out[10]:

	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 [11]:
```
# 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[11]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	HBA1C	Heart Issues	Trans
0	Id1	Hawks, Ms. Kelly	1968	Oct	12	0	63770.43	tier - 1	tier - 3	R1013	47.410	7.47	No	
1	Id2	Lehner, Mr. Matthew D	1977	Jun	8	0	62592.87	tier - 2	tier - 3	R1013	30.360	5.77	No	
2	Id3	Lu, Mr. Phil	1970	?	11	3	60021.40	tier - 1	tier - 1	R1012	34.485	11.87	yes	
3	Id4	Osborne, Ms. Kelsey	1991	Jun	6	1	58571.07	tier - 1	tier - 3	R1024	38.095	6.05	No	
4	Id5	Kadala, Ms. Kristyn	1989	Jun	19	0	55135.40	tier - 1	tier - 2	R1012	35.530	5.45	No	

In [12]:
```
final_df.shape
```
Out[12]: (2335, 17)

2. Check for missing values in the dataset

In [13]:
```
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 [14]:
```
final_df.dtypes.value_counts()
```

Out[14]:
```
object     12
float64     3
int64       2
dtype: int64
```

In [15]:
```
# Lets check the missing values in the data set.
final_df.isnull().sum()
```

Out[15]:
```
Customer ID               0
name                      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
dtype: int64
```

There is no missing value in the dataset it is clear fromt he above code, 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 [16]:
```
trivial_value = final_df[final_df.eq("?").any(1)]
trivial_value
```

Out[16]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	HBA1C	Heart Issues	Ti
2	Id3	Lu, Mr. Phil	1970	?	11	3	60021.40	tier - 1	tier - 1	R1012	34.485	11.87	yes	
169	Id170	Torphy, Mr. Bobby	2000	Sep	5	1	37165.16	tier - 1	tier - 3	?	37.620	6.32	yes	
559	Id560	Pearlman, Mr. Oz	1994	Jul	1	3	17663.14	tier - 1	tier - 3	R1013	23.980	4.90	No	
634	Id635	Bruns, Mr. Zachary T	2004	Jul	17	0	15518.18	tier - 2	tier - 3	R1015	25.175	4.96	No	
1285	Id1286	Ainsley, Ms. Katie M.	?	Dec	12	1	8547.69	tier - 2	tier - 1	R1013	29.370	8.01	yes	
1288	Id1289	Levine, Ms. Annie J.	?	Jul	24	0	8534.67	tier - 2	tier - 3	R1024	24.320	11.56	yes	
1792	Id1793	Capriolo, Mr. Michael	1995	Dec	1	3	4827.90	tier - 1	tier - 2	?	18.905	4.91	yes	
2317	Id2318	Gagnon, Ms. Candice M	1996	?	18	0	770.38	tier - 3	?	R1012	18.820	5.51	yes	
2321	Id2322	Street, Ms. Holly	2002	?	19	0	750.00	tier - 3	tier - 1	R1012	21.380	8.01	No	
2323	Id2324	Duffy, Ms. Meghan K	1999	Dec	26	0	700.00	?	tier - 3	R1013	22.240	5.04	No	

In [17]:
```
trivial_value.shape
```

Out[17]: (10, 17)

In [18]:
```
# Percentage of row that have the trivial values
round(trivial_value.shape[0]/final_df.shape[0]*100, 2)
```

Out[18]: 0.43

There is total 0.43% of rows contain the trivial values.

In [19]:
```
# Now lets drop the all row that contain the trivial values in the data set.
final_df.drop(final_df[final_df.eq("?").any(1)].index, axis=0, inplace=True)
```

In [20]:
```
final_df.shape
```

Out[20]: (2325, 17)

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

In [21]:
```
# First we will deal with the nominal categorical variable.
```

In [22]:
```
final_df["Heart Issues"].value_counts()
```

Out[22]:
```
No      1405
yes      920
Name: Heart Issues, dtype: int64
```

In [23]:
```
final_df["Any Transplants"].value_counts()
```

Out[23]:
```
No     2183
yes     142
Name: Any Transplants, dtype: int64
```

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

Out[24]:
```
No     1934
Yes     391
Name: Cancer history, dtype: int64
```

In [25]:
```
final_df["smoker"].value_counts()
```

Out[25]:
```
No     1839
yes     486
Name: smoker, dtype: int64
```

In [26]:
```
# We have some categorical values so first of all we have to transform then by using the label e
from sklearn.preprocessing import LabelEncoder
```

In [27]:
```
le = LabelEncoder()
```

In [28]:
```
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 [29]:
```
final_df["Heart Issues"].value_counts()
```

Out[29]:
```
0    1405
1     920
Name: Heart Issues, dtype: int64
```

In [30]:
```
final_df.head()
```

Out[30]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	HBA1C	Heart Issues	Trans
0	Id1	Hawks, Ms. Kelly	1968	Oct	12	0	63770.43	tier - 1	tier - 3	R1013	47.410	7.47	0	
1	Id2	Lehner, Mr. Matthew D	1977	Jun	8	0	62592.87	tier - 2	tier - 3	R1013	30.360	5.77	0	
3	Id4	Osborne, Ms. Kelsey	1991	Jun	6	1	58571.07	tier - 1	tier - 3	R1024	38.095	6.05	0	
4	Id5	Kadala, Ms. Kristyn	1989	Jun	19	0	55135.40	tier - 1	tier - 2	R1012	35.530	5.45	0	
5	Id6	Baker, Mr. Russell B.	1962	Aug	4	0	52590.83	tier - 1	tier - 3	R1011	32.800	6.59	0	

In [31]:
```
# Now we will deal with the ordinal categorical variable.
```

In [32]:
```
def clean_ordinal_variable(val):
    return int(val.replace("tier", "").replace(" ", "").replace("-", ""))
```

In [33]:
```
final_df["Hospital tier"] = final_df["Hospital tier"].map(clean_ordinal_variable)
final_df["City tier"] = final_df["City tier"].map(clean_ordinal_variable)
```

In [34]:
```
final_df["City tier"].value_counts()
```

Out[34]:
```
2    807
3    789
1    729
Name: City tier, dtype: int64
```

In [35]:
```
final_df.head()
```

Out[35]:

	Customer ID	name	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	HBA1C	Heart Issues	Trans
0	Id1	Hawks, Ms. Kelly	1968	Oct	12	0	63770.43	1	3	R1013	47.410	7.47	0	
1	Id2	Lehner, Mr. Matthew D	1977	Jun	8	0	62592.87	2	3	R1013	30.360	5.77	0	
3	Id4	Osborne, Ms. Kelsey	1991	Jun	6	1	58571.07	1	3	R1024	38.095	6.05	0	
4	Id5	Kadala, Ms. Kristyn	1989	Jun	19	0	55135.40	1	2	R1012	35.530	5.45	0	
5	Id6	Baker, Mr. Russell B.	1962	Aug	4	0	52590.83	1	3	R1011	32.800	6.59	0	

5. 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 [36]:
```
final_df["State ID"].value_counts()
```

Out[36]:
```
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
```

It is clear from the above code some of the state is worth investigator like R1013, R1012, R1011 and R1024.

In [37]:
```
Dummies = pd.get_dummies(final_df["State ID"], prefix= "State_ID")
```

In [38]:
```
Dummies
```

Out[38]:

	State_ID_R1011	State_ID_R1012	State_ID_R1013	State_ID_R1014	State_ID_R1015	State_ID_R1016	State_ID_R1
0	0	0	1	0	0	0	
1	0	0	1	0	0	0	
3	0	0	0	0	0	0	
4	0	1	0	0	0	0	
5	1	0	0	0	0	0	
...	...	...	...	...	...	...	
2330	0	0	1	0	0	0	
2331	0	0	1	0	0	0	
2332	0	0	1	0	0	0	
2333	0	0	1	0	0	0	
2334	0	0	1	0	0	0	

2325 rows × 16 columns

In [39]:
```
# 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[39]:

	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 [40]:
```
final_df = pd.concat([final_df, Dummy], axis=1)
```

In [41]:
```
final_df.drop(['State ID'], inplace=True, axis=1)
```

In [42]:
```
final_df.head()
```

Out[42]:

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

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

In [43]:
```
final_df['NumberOfMajorSurgeries'].value_counts()
```

Out[43]:
```
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 [44]:
```
final_df['NumberOfMajorSurgeries'] = final_df['NumberOfMajorSurgeries'].replace('No major surge
```

In [45]:
```
final_df['NumberOfMajorSurgeries'] = final_df["NumberOfMajorSurgeries"].astype(int)
```

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

In [46]:
```
final_df["year"] = pd.to_datetime(final_df["year"], format='%Y').dt.year
final_df["year"]
```

Out[46]:
```
0       1968
1       1977
3       1991
4       1989
5       1962
        ...
2330    1998
2331    1992
2332    1993
2333    1992
2334    1992
Name: year, Length: 2325, dtype: int64
```

In [47]:
```
final_df["month"] = pd.to_datetime(final_df["month"], format='%b').dt.month
final_df["month"]
```

Out[47]:
```
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 [48]:
```
final_df['DateInt'] = final_df["year"].astype(str) + final_df["month"].astype(str).str.zfill(2)
```

In [49]:
```
final_df['DOB'] = pd.to_datetime(final_df.DateInt, format = "%Y%m%d")
```

In [50]:
```
final_df.drop(["DateInt"], inplace = True, axis=1)
```

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

Out[51]:

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

In [175]:
```
import datetime as dt
current_date = dt.datetime.now()
```

In [53]:
```
final_df['age'] = (((current_date - final_df.DOB).dt.days)/365).astype(int)
```

In [54]:
```
final_df.head()
```

Out[54]:

hildren	charges	Hospital tier	City tier	BMI	...	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	State_ID_R
0	63770.43	1	3	47.410	...	0	0	0	0	1	
0	62592.87	2	3	30.360	...	0	0	0	0	1	
1	58571.07	1	3	38.095	...	0	0	0	0	1	
0	55135.40	1	2	35.530	...	0	0	0	0	1	
0	52590.83	1	3	32.800	...	0	0	0	0	1	

8. 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 [55]:
```
def gender(val):
    if "Ms." in val:
        return 0
    else:
        return 1
```

the salutation (Ms.) denote the female and (Mr.) denote the male.

> The gender will play the inportant role to predict the hospitalization cost so for model building we directly denote the gender by int.

> > Male = 1 & Female = 0

In [56]:
```
final_df["gender"] = final_df["name"].map(gender)
```

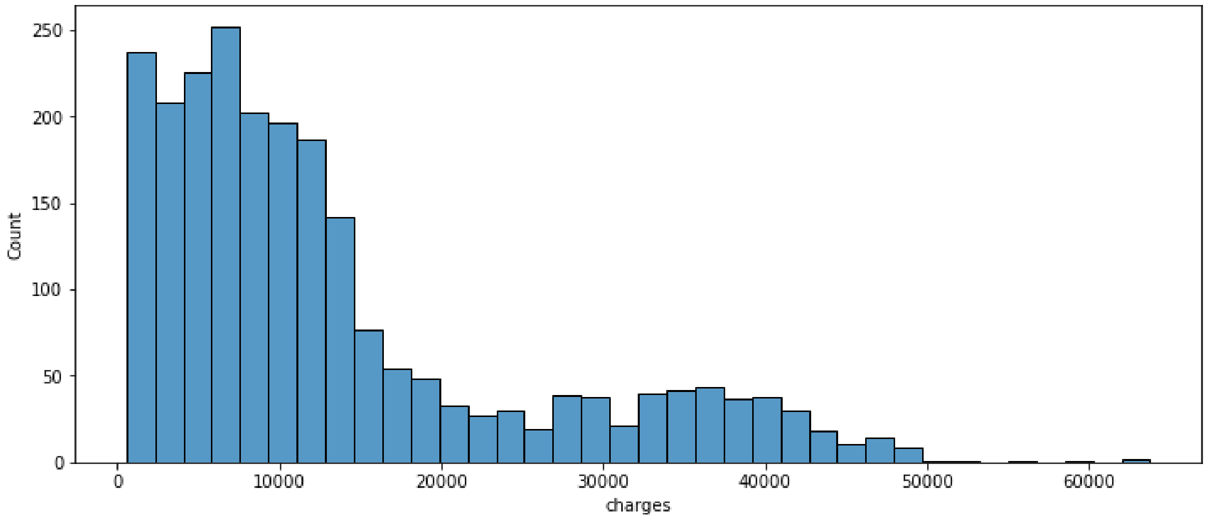
In [57]:
```
final_df.head()
```

Out[57]:

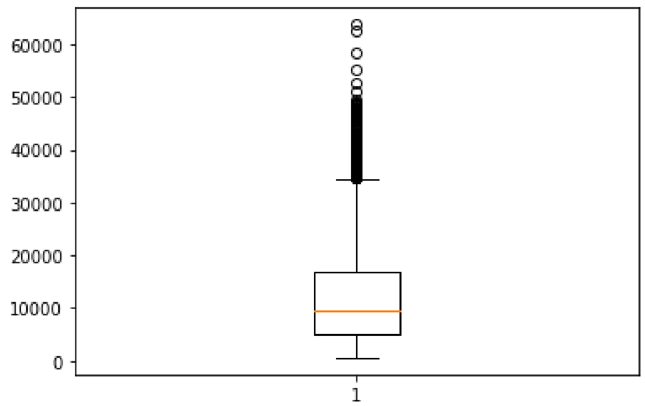
dren	charges	Hospital tier	City tier	BMI	...	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	State_ID_R1011	Stat
0	63770.43	1	3	47.410	...	0	0	0	1	0	
0	62592.87	2	3	30.360	...	0	0	0	1	0	
1	58571.07	1	3	38.095	...	0	0	0	1	0	
0	55135.40	1	2	35.530	...	0	0	0	1	0	
0	52590.83	1	3	32.800	...	0	0	0	1	1	

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

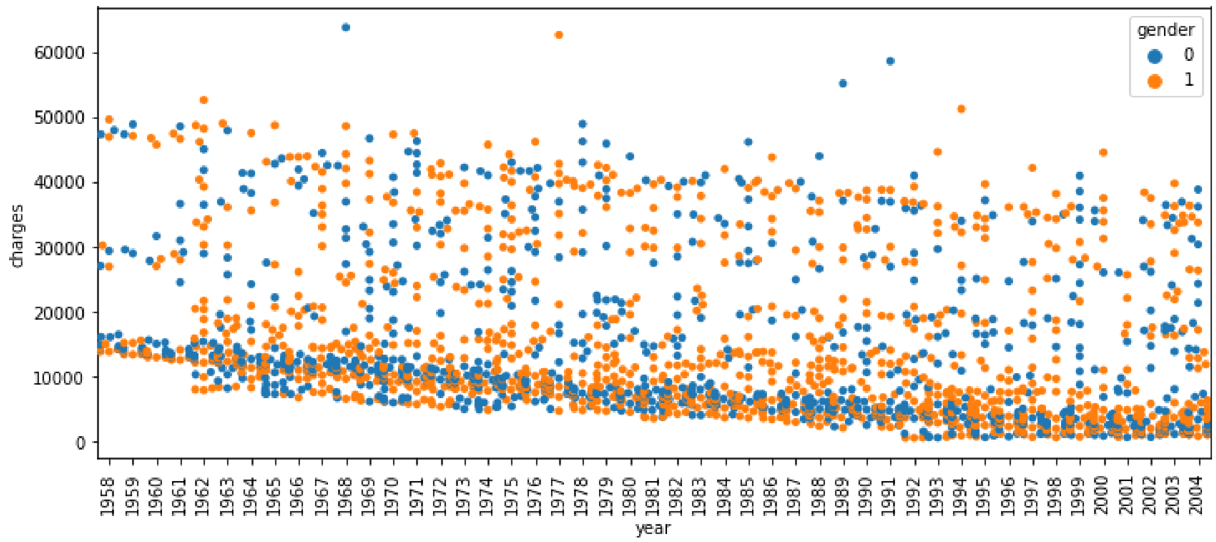
In [58]:
```
# Lets make the histogram for the cost distribution.
plt.figure(figsize=(12,5))
sns.histplot(final_df['charges'])
plt.show()
```



In [59]:
```
# Now visualize the cost distribution of the hospitals by box or whisker plot.
plt.boxplot(final_df['charges'])
plt.show()
```

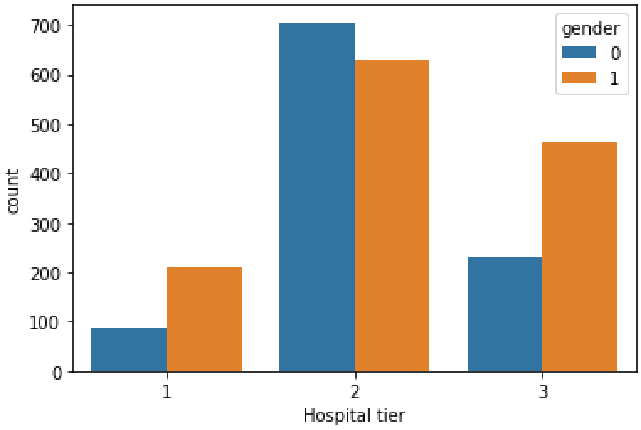


In [60]:
```
# Now visualize the cost distribution of the hospitals by swarm plot.
plt.figure(figsize=(12,5))
sns.swarmplot(x='year', y='charges', hue="gender", data=final_df)
plt.xticks(rotation=90)
plt.show()
```



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

In [61]:
```
sns.countplot(data = final_df, x='Hospital tier', hue= 'gender')
plt.show()
```



From the above representation it is clear that the number of female in the tier 1 and 3 is half of the male just in tier 2 hospital female is little bit more as compare to male.

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

In [62]:
```
print("median cost of tier 1 hospitals:", final_df[final_df["Hospital tier"]==1].charges.median
print("median cost of tier 2 hospitals:", final_df[final_df["Hospital tier"]==2].charges.median
print("median cost of tier 3 hospitals:", final_df[final_df["Hospital tier"]==3].charges.median
```

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

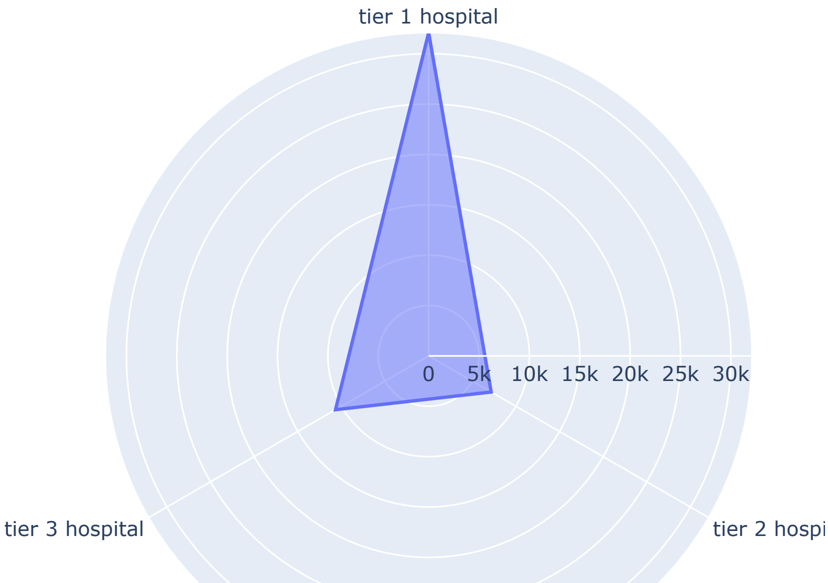
In [63]:
```
df = pd.DataFrame(dict(r=[32097.43, 7168.76, 10676.83],theta=['tier 1 hospital','tier 2 hospita
```

In [64]:
```
df
```

Out[64]:

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

In [65]:
```
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. Create a frequency table and a stacked bar chart to visualize the count of people in the different tiers of cities and hospitals

In [66]:
```
# Frequency table for count of the people according to the tier of city and hospitals.
final_df["Hospital tier"].value_counts()
```

Out[66]:
```
2    1334
3     691
1     300
Name: Hospital tier, dtype: int64
```

In [67]:
```
city_freq = final_df["City tier"].value_counts().rename_axis('City&hospital_tier').reset_index(
```

In [68]:
```
hospital_freq = final_df["Hospital tier"].value_counts().rename_axis('City&hospital_tier').rese
```

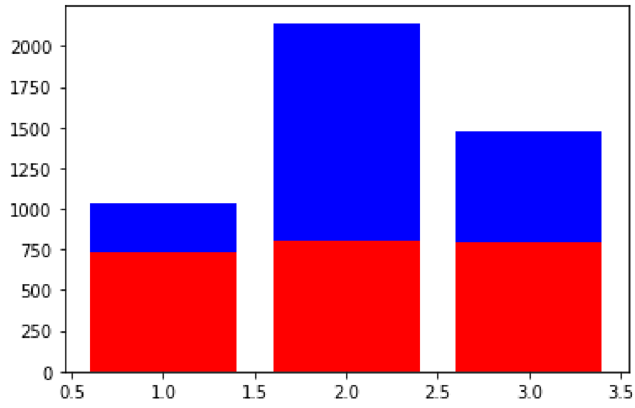
In [69]:
```
df = pd.merge(city_freq, hospital_freq, on = 'City&hospital_tier')
```

In [70]:
```
df
```

Out[70]:

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

In [71]:
```
plt.bar(df["City&hospital_tier"], df["city_counts"], color='r')
plt.bar(df["City&hospital_tier"], df["hospital_counts"], bottom=df["city_counts"], color='b')
plt.show()
```



13. Test the following null hypotheses:

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

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

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

d. Smoking and heart issues are independent

In [72]:
```
from scipy.stats import ttest_1samp
```

In [73]:
```
# a. The average hospitalization costs for the three types of hospitals are not significantly d
print("median cost of tier 1 hospitals:", final_df[final_df["Hospital tier"]==1].charges.median
print("median cost of tier 2 hospitals:", final_df[final_df["Hospital tier"]==2].charges.median
print("median cost of tier 3 hospitals:", final_df[final_df["Hospital tier"]==3].charges.median
```

```
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 [74]:
```
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 [75]:
```
# b. The average hospitalization costs for the three types of cities are not significantly diffe
print("median cost of tier 1 city:", final_df[final_df["City tier"]==1].charges.median())
print("median cost of tier 2 city:", final_df[final_df["City tier"]==2].charges.median())
print("median cost of tier 3 city:", final_df[final_df["City tier"]==3].charges.median())
```

```
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 [76]:
```
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 [77]:
```
# c. The average hospitalization cost for smokers is not significantly different from the averag
print("median cost of smoker:", final_df[final_df["smoker"]==1].charges.median())
print("median cost of non smoker:", final_df[final_df["smoker"]==0].charges.median())
```

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

In [78]:
```
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 [79]:
```
# d. Smoking and heart issues are independent
from scipy.stats import chi2_contingency
table = [[final_df["Heart Issues"].value_counts()],[final_df["smoker"].value_counts()]]
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
```

Project Task: Week 2

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

In [80]:
```
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 [81]:
```
# In the data frame same of the column are not usable to model building so lets first drop all.
#then indentify the highly corelated predictor.
final_df.drop(["Customer ID",'name', 'year', 'month', 'date', 'DOB'], inplace=True, axis=1)
```

In [82]:
```
final_df.shape
```

Out[82]: (2325, 16)

In [83]:
```
final_df.head()
```

Out[83]:

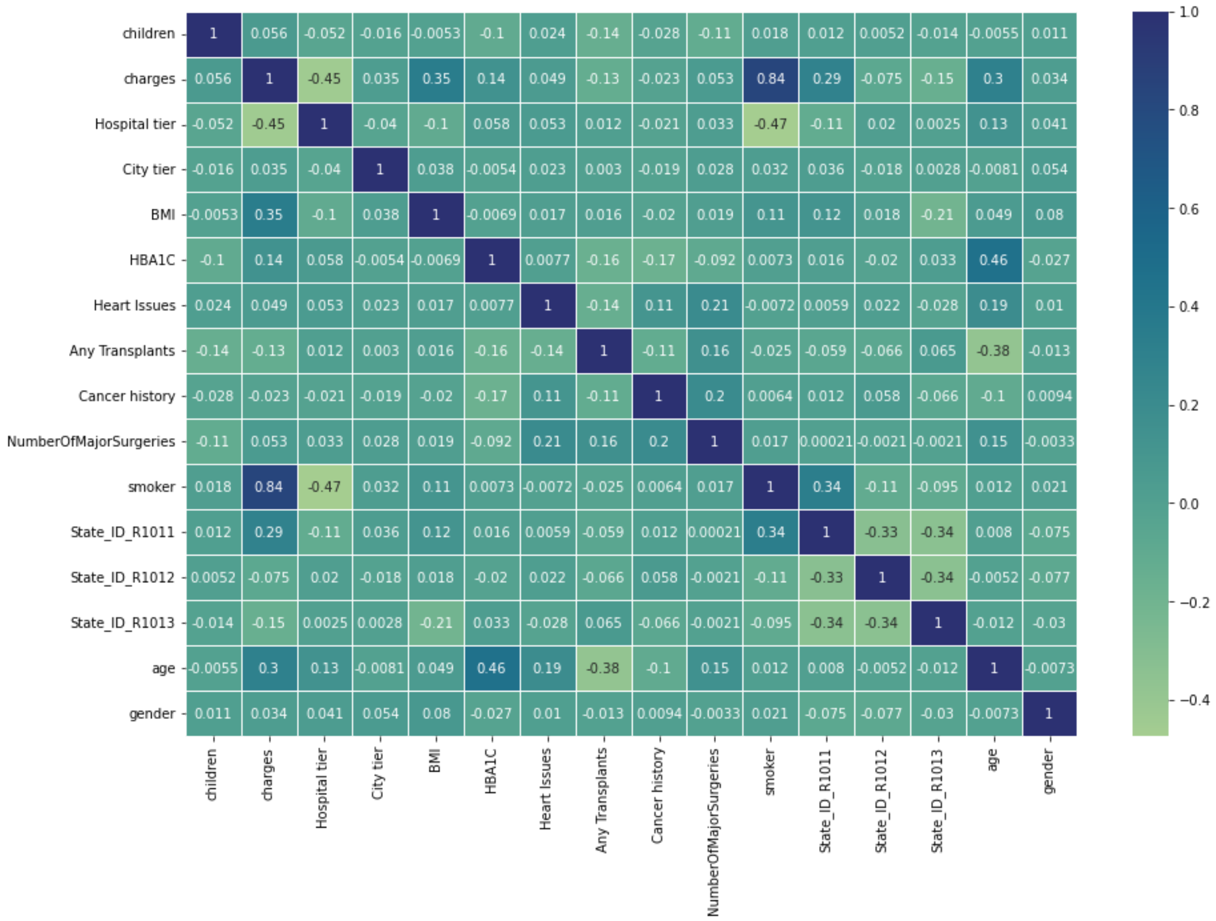
	children	charges	Hospital tier	City tier	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker
0	0	63770.43	1	3	47.410	7.47	0	0	0	0	1
1	0	62592.87	2	3	30.360	5.77	0	0	0	0	1
3	1	58571.07	1	3	38.095	6.05	0	0	0	0	1
4	0	55135.40	1	2	35.530	5.45	0	0	0	0	1
5	0	52590.83	1	3	32.800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

In [84]:
```
corr = final_df.corr()
corr
```

Out[84]:

	children	charges	Hospital tier	City tier	BMI	HBA1C	Heart Issues	Any Transplants	C h
children	1.000000	0.055901	-0.052438	-0.015760	-0.005339	-0.101379	0.023984	-0.142040	-0.0
charges	0.055901	1.000000	-0.446687	0.035300	0.346730	0.139697	0.049299	-0.127028	-0.0
Hospital tier	-0.052438	-0.446687	1.000000	-0.039755	-0.104771	0.057855	0.053376	0.011729	-0.0
City tier	-0.015760	0.035300	-0.039755	1.000000	0.038123	-0.005404	0.023152	0.002970	-0.0
BMI	-0.005339	0.346730	-0.104771	0.038123	1.000000	-0.006920	0.017129	0.015893	-0.0
HBA1C	-0.101379	0.139697	0.057855	-0.005404	-0.006920	1.000000	0.007699	-0.159855	-0.1
Heart Issues	0.023984	0.049299	0.053376	0.023152	0.017129	0.007699	1.000000	-0.140269	0.
Any Transplants	-0.142040	-0.127028	0.011729	0.002970	0.015893	-0.159855	-0.140269	1.000000	-0.1
Cancer history	-0.027880	-0.022522	-0.021429	-0.018639	-0.020235	-0.170921	0.111190	-0.114677	1.0
NumberOfMajorSurgeries	-0.113161	0.053308	0.033230	0.027937	0.018851	-0.091594	0.206147	0.158593	0.2
smoker	0.017713	0.838462	-0.474077	0.032034	0.107126	0.007257	-0.007159	-0.025101	0.0
State_ID_R1011	0.011666	0.286956	-0.114685	0.036049	0.115671	0.015525	0.005852	-0.058553	0.0
State_ID_R1012	0.005247	-0.074636	0.020272	-0.018253	0.017939	-0.019513	0.021770	-0.066453	0.0
State_ID_R1013	-0.013834	-0.150634	0.002455	0.002766	-0.208744	0.033453	-0.027967	0.064563	-0.0
age	-0.005457	0.304395	0.133771	-0.008070	0.049260	0.460558	0.192273	-0.381084	-0.1
gender	0.011205	0.034069	0.041261	0.054073	0.079930	-0.027339	0.010277	-0.012737	0.0

In [85]:
```
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.

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

Note:

• Perform the stratified 5-fold cross-validation technique for model building and validation • Use standardization and hyperparameter tuning effectively • Use sklearn-pipelines • Use appropriate regularization techniques to address the bias-variance trade-off

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

b. 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

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

d. Determine the variable importance scores, and identify the redundant variables

In [86]:
```
# lets first seperate the input and output data.
x = final_df.drop(["charges"], axis=1)
y = final_df[['charges']]
```

In [87]:
```
# Lets split the data set into the training and testing data.
from sklearn.model_selection import train_test_split
```

In [88]:
```
x_train, x_test, y_train, y_test  = train_test_split(x,y, test_size=.20, random_state=10)
```

In [89]:
```
# Now standardize the data.
from sklearn.preprocessing import StandardScaler
```

In [90]:
```
sc = StandardScaler()
```

In [91]:
```
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

In [92]:
```
from sklearn.linear_model import SGDRegressor
```

In [93]:
```
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[93]:
```
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, 50,
                                   100, 500, 1000],
                         'penalty': ['l2', 'l1', 'elasticnet']},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [94]:
```
model_cv.best_params_
```

Out[94]:
```
{'alpha': 100, 'penalty': 'l1'}
```

In [95]:
```
sgd = SGDRegressor(alpha= 100, penalty= 'l1')
```

In [127]:
```
sgd.fit(x_train, y_train)
```

Out[127]:
```
SGDRegressor(alpha=100, penalty='l1')
```

In [108]:
```
sgd.score(x_test, y_test)
```

Out[108]:
```
0.8578904042087353
```

In [109]:
```
y_pred = sgd.predict(x_test)
```

In [110]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [111]:
```
sgd_mae = mean_absolute_error(y_test, y_pred)
sgd_mse = mean_squared_error(y_test, y_pred)
sgd_rmse = lin_mse*(1/2.0)
```

In [112]:
```
print("MAE:", sgd_mae)
print("MSE:", sgd_mse)
print("RMSE:", sgd_rmse)
```

```
MAE: 3152.2079261574295
MSE: 23903675.025537692
RMSE: 11951837.512768846
```

In [114]:
```
# d. Determine the variable importance scores, and identify the redundant variables
importance = sgd.coef_
```

In [121]:
```
pd.DataFrame(importance, index = x.columns, columns=['Feature_imp'])
```

Out[121]:

	Feature_imp
children	405.463359
Hospital tier	-1122.354960
City tier	0.000000
BMI	2710.955036
HBA1C	100.924926
Heart Issues	0.000000
Any Transplants	0.000000
Cancer history	0.000000
NumberOfMajorSurgeries	0.000000
smoker	8707.309379
State_ID_R1011	-159.907496
State_ID_R1012	0.000000
State_ID_R1013	-364.309134
age	3460.249400
gender	0.000000

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

random forest

In [129]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [132]:
```
# 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[132]:
```
RandomForestRegressor(n_estimators=1000, random_state=42)
```

In [133]:
```
score = rf.score(x_test,y_test)
score
```

Out[133]:
```
0.9222696338245824
```

In [135]:
```
y_pred = rf.predict(x_test)
```

In [139]:
```
rf_mae = mean_absolute_error(y_test, y_pred)
```

In [140]:
```
rf_mae
```

Out[140]:
```
1870.3529629462323
```

extreme gradient boosting

In [142]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

In [143]:
```
# 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[143]:
```
GradientBoostingRegressor(n_estimators=1000, random_state=42)
```

In [144]:
```
score = gbr.score(x_test,y_test)
score
```

Out[144]:
```
0.9042734212625119
```

In [145]:
```
y_pred = gbr.predict(x_test)
```

In [146]:
```
gbr_mae = mean_absolute_error(y_test, y_pred)
gbr_mae
```

Out[146]:
```
2375.8700944163274
```

4. Case scenario:

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 [183]:
```
# First we need to calculate the age of the person.
date = str(19881228)
date1 = pd.to_datetime(date, format = "%Y%m%d")
```

In [184]:
```
current_date = dt.datetime.now()
current_date
```

Out[184]:
```
datetime.datetime(2022, 12, 31, 17, 5, 42, 991258)
```

In [191]:
```
age = (current_date - date)
age
```

Out[191]:
```
Timedelta('12421 days 17:05:42.991258')
```

In [194]:
```
age = int(12421/365)
age
```

Out[194]:
```
34
```

In [197]:
```
# 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[197]:
```
29.41
```

In [208]:
```
# Now lets gen
list = [[2,1,1,24.41,5.8,0,0,0,0,1,1,0,0,34,0]]
```

In [207]:
```
df = pd.DataFrame(list, columns = ['children', 'Hospital tier', 'City tier', 'BMI', 'HBA1C','He
                               'Cancer history','NumberOfMajorSurgeries', 'smoker', 'State_ID_R1
                               'State_ID_R1013', 'age', 'gender'] )
df
```

Out[207]:

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

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

In [228]:
```
Hospital_cost = []
```

In [229]:
```
# Now lets predict the hospitalization cost through SGDRegressor
Cost1 = sgd.predict(df)
Hospital_cost.append(Cost1)
```

In [230]:
```
# Now lets predict the hospitalization cost through Random Forest
Cost2 = rf.predict(df)
Hospital_cost.append(Cost2)
```

In [232]:
```
# Now lets predict the hospitalization cost throug Extreme gradient Booster
Cost3 = gbr.predict(df)
Hospital_cost.append(Cost3)
```

In [234]:
```
avg_cost = np.mean(Hospital_cost)
avg_cost
```

Out[234]: 104922.5970678889

So in the new case the avg predicted hospitalization cost is 104922.59

In [ ]:

```
/* 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
```

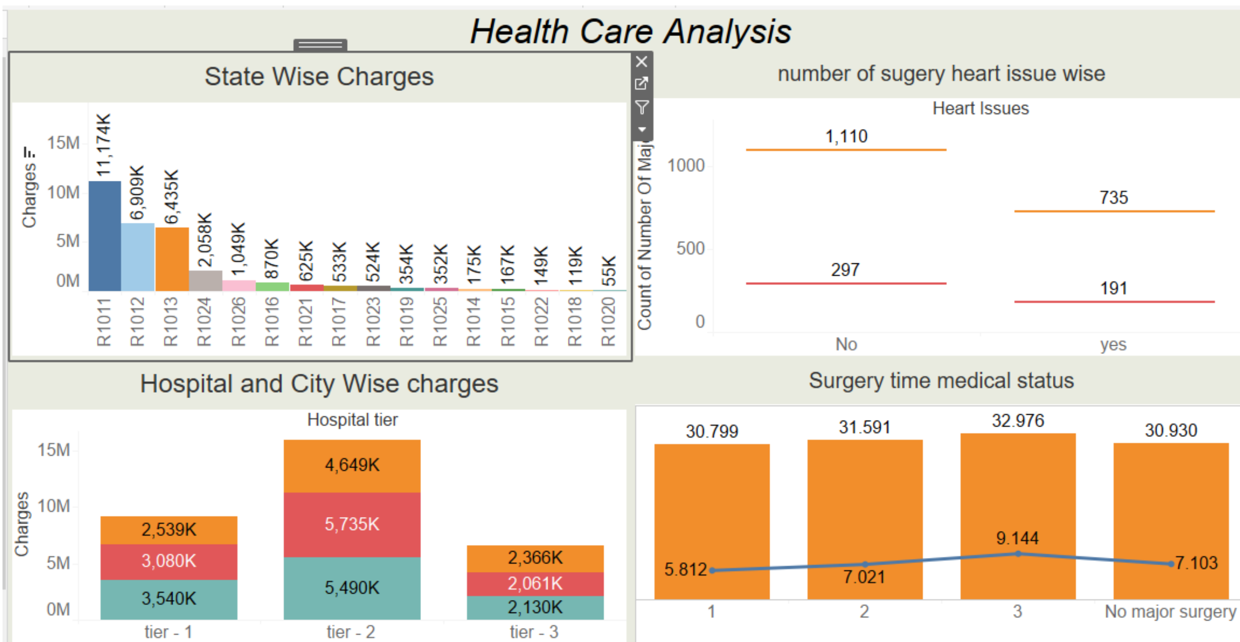
```
each state. */

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

Project Task: Week 2

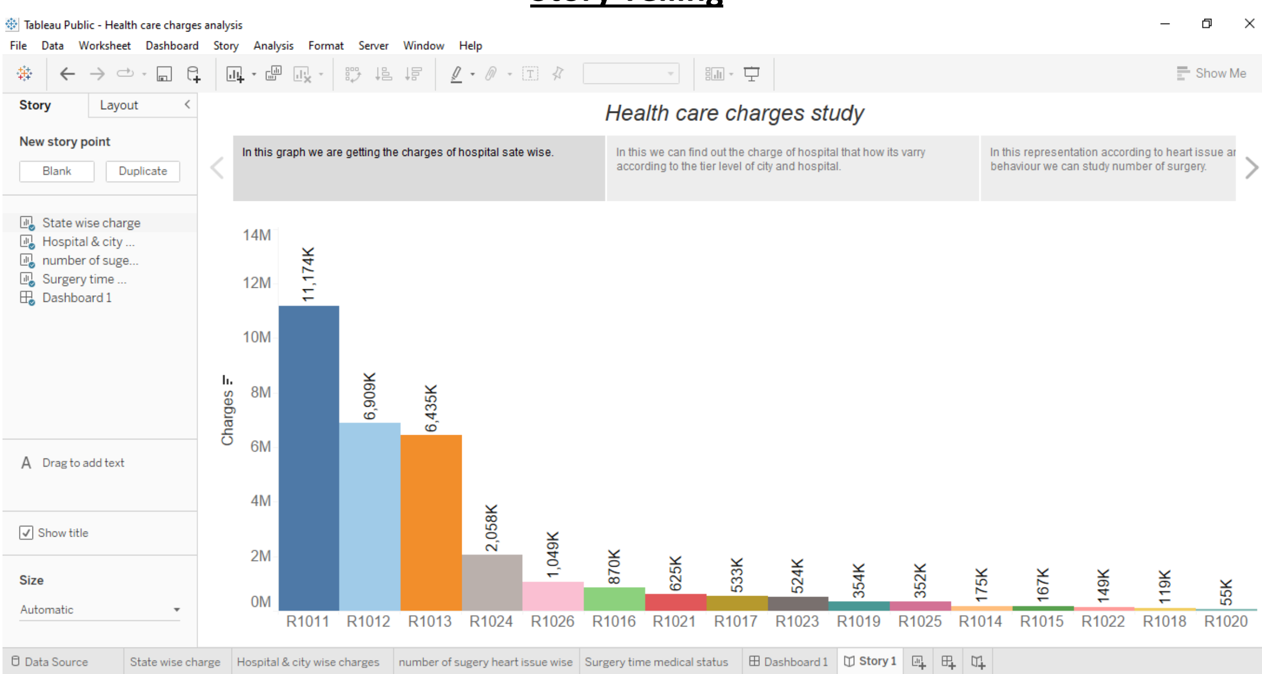
1. Create a dashboard in Tableau by selecting the appropriate chart types and business metrics

Note: Put more emphasis on data storytelling

<u>Dashboard</u>



<u>Story Telling</u>



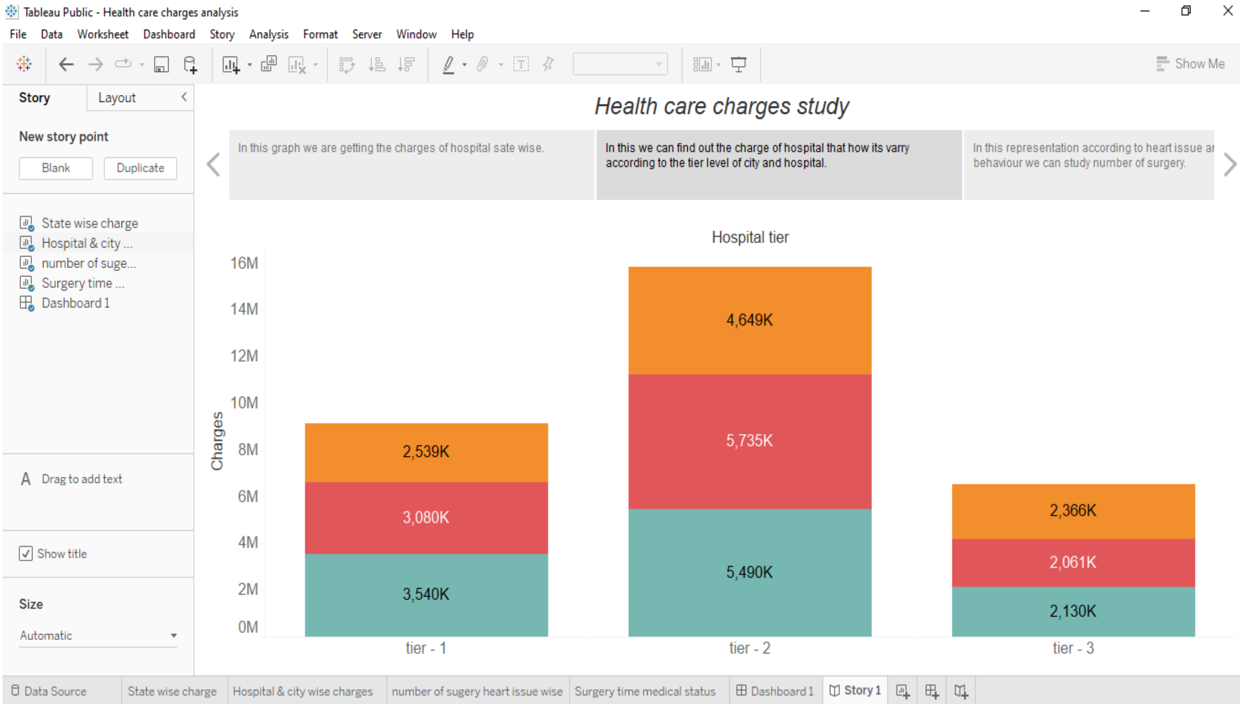


Tableau Public - Health care charges analysis
File Data Worksheet Dashboard Story Analysis Format Server Window Help
Show Me
Story Layout
New story point
Blank Duplicate
State wise charge
Hospital & city ...
number of suge...
Surgery time ...
Dashboard 1
Drag to add text
Show title
Size
Automatic
Health care charges study
In this graph we are getting the charges of hospital sate wise.
In this we can find out the charge of hospital that how its varry according to the tier level of city and hospital.
In this representation according to heart issue ar behaviour we can study number of surgery.
Hospital tier
Charges
16M
14M
12M
10M
8M
6M
4M
2M
0M
2,539K
3,080K
3,540K
4,649K
5,735K
5,490K
2,366K
2,061K
2,130K
tier - 1
tier - 2
tier - 3
Data Source
State wise charge
Hospital & city wise charges
number of sugery heart issue wise
Surgery time medical status
Dashboard 1
Story 1

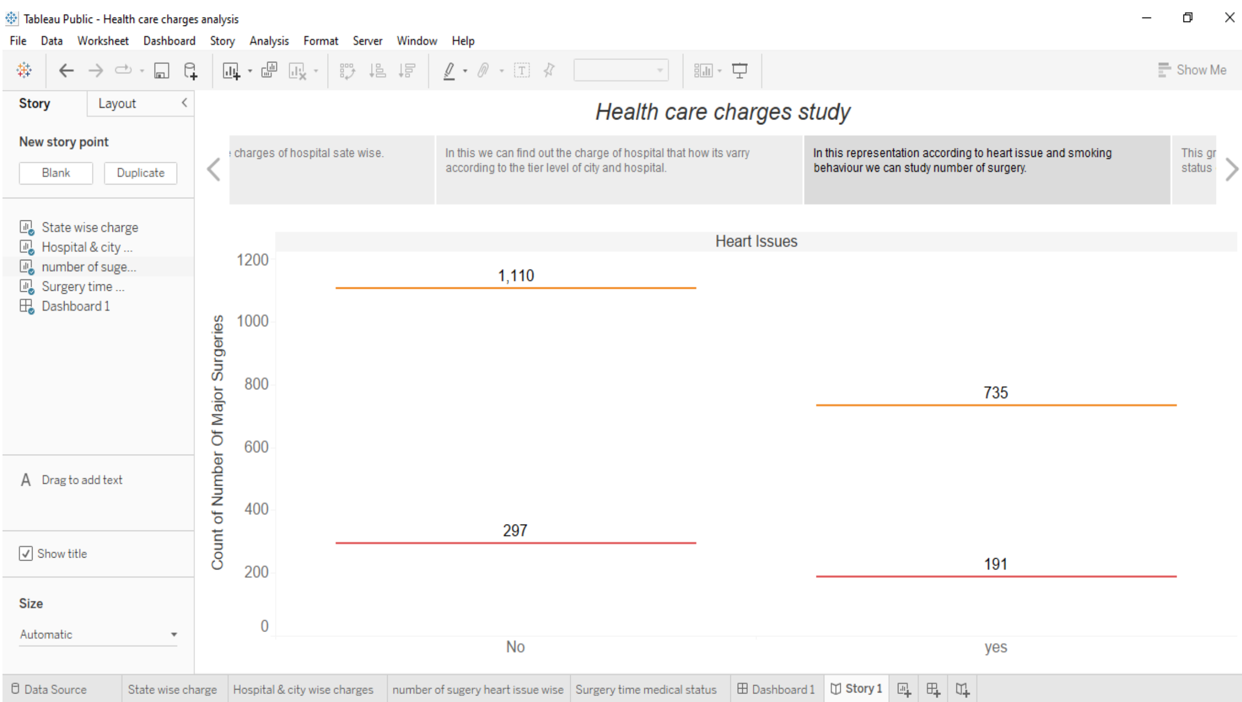


Tableau Public - Health care charges analysis
File Data Worksheet Dashboard Story Analysis Format Server Window Help
Show Me
Story Layout
New story point
Blank Duplicate
State wise charge
Hospital & city ...
number of suge...
Surgery time ...
Dashboard 1
Drag to add text
Show title
Size
Automatic
Health care charges study
charges of hospital sate wise.
In this we can find out the charge of hospital that how its varry according to the tier level of city and hospital.
In this representation according to heart issue and smoking behaviour we can study number of surgery.
This gr status
Heart Issues
Count of Number Of Major Surgeries
1200
1000
800
600
400
200
0
1,110
297
735
191
No
yes
Data Source
State wise charge
Hospital & city wise charges
number of sugery heart issue wise
Surgery time medical status
Dashboard 1
Story 1

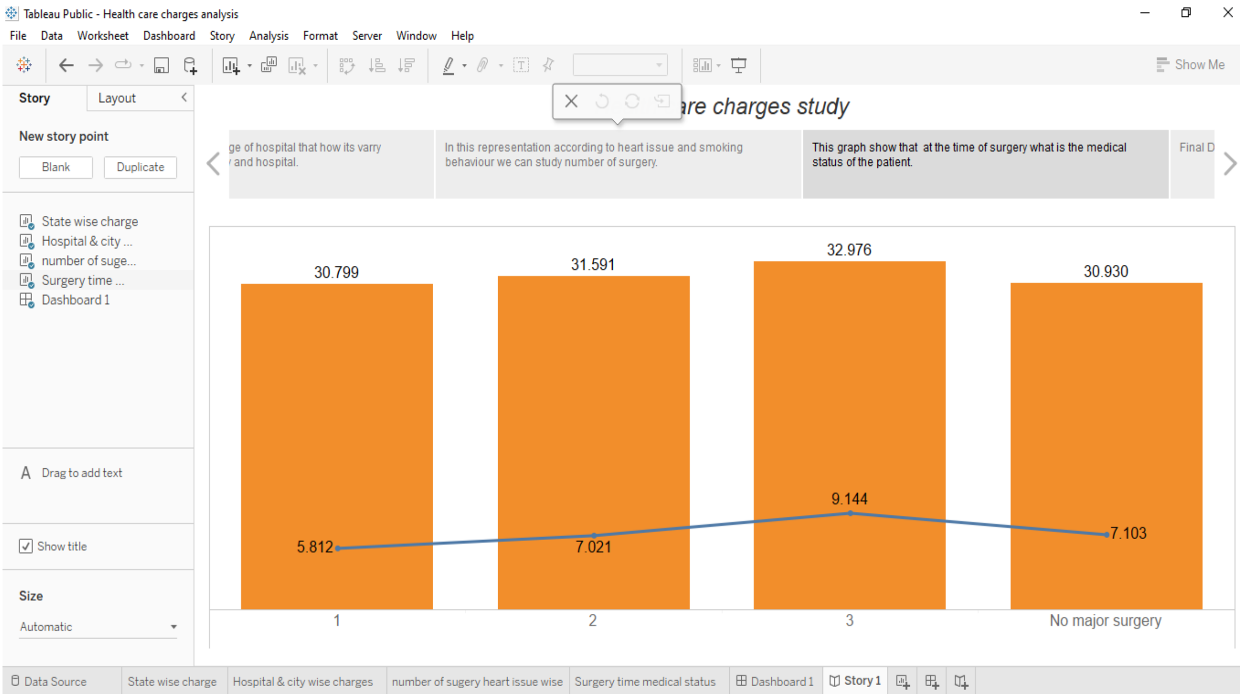


Tableau Public - Health care charges analysis
File Data Worksheet Dashboard Story Analysis Format Server Window Help
Show Me
Story
Layout
New story point
Blank
Duplicate
State wise charge
Hospital & city ...
number of suge...
Surgery time ...
Dashboard 1
Drag to add text
Show title
Size
Automatic
are charges study
ge of hospital that how its varry
and hospital.
In this representation according to heart issue and smoking behaviour we can study number of surgery.
This graph show that at the time of surgery what is the medical status of the patient.
Final D
30.799
31.591
32.976
30.930
9.144
5.812
7.021
7.103
1
2
3
No major surgery
Data Source
State wise charge
Hospital & city wise charges
number of sugery heart issue wise
Surgery time medical status
Dashboard 1
Story 1

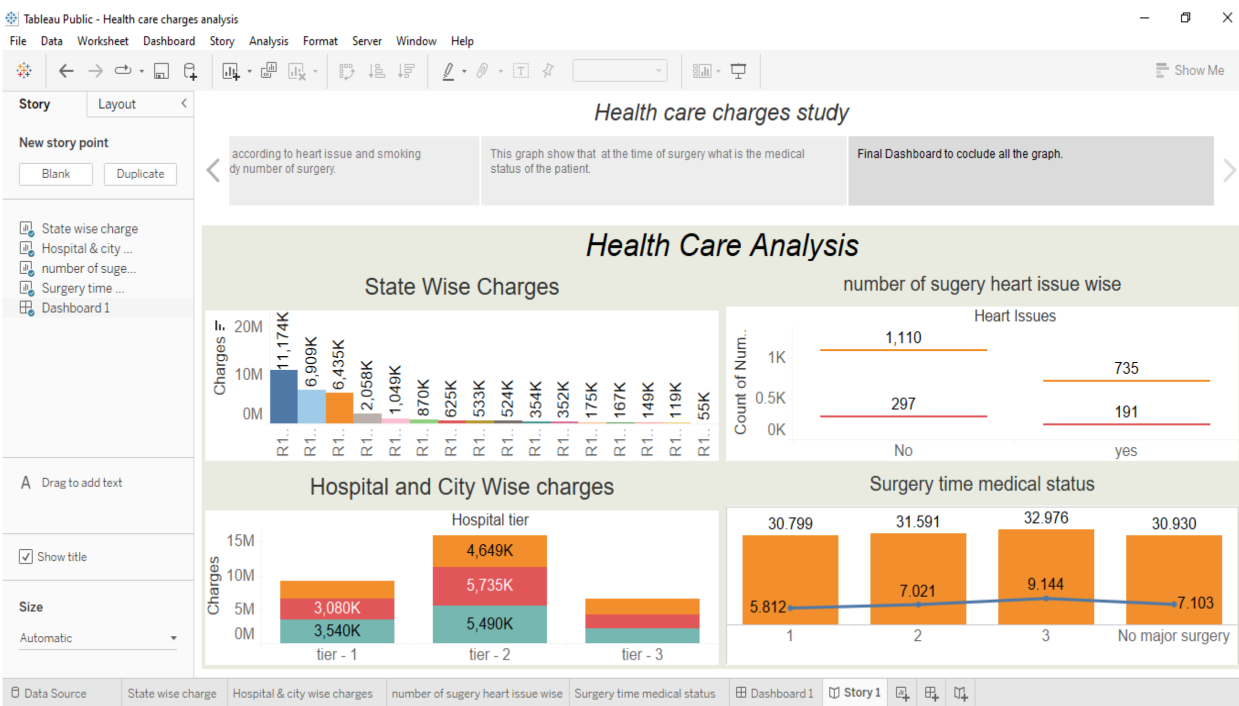


Tableau Public - Health care charges analysis
File Data Worksheet Dashboard Story Analysis Format Server Window Help
Show Me
Story
Layout
New story point
Blank
Duplicate
State wise charge
Hospital & city ...
number of suge...
Surgery time ...
Dashboard 1
Drag to add text
Show title
Size
Automatic
Health care charges study
according to heart issue and smoking
dy number of surgery.
This graph show that at the time of surgery what is the medical status of the patient.
Final Dashboard to coclude all the graph.
Health Care Analysis
State Wise Charges
Charges
20M
10M
0M
11,174K
6,909K
6,435K
2,058K
1,049K
870K
625K
533K
524K
354K
352K
175K
167K
149K
119K
55K
number of sugery heart issue wise
Heart Issues
Count of Num..
1K
0.5K
0K
1,110
297
735
191
No
yes
Hospital and City Wise charges
Hospital tier
15M
10M
5M
0M
4,649K
5,735K
5,490K
3,080K
3,540K
tier - 1
tier - 2
tier - 3
Surgery time medical status
30.799
31.591
32.976
30.930
5.812
7.021
9.144
7.103
1
2
3
No major surgery
Data Source
State wise charge
Hospital & city wise charges
number of sugery heart issue wise
Surgery time medical status
Dashboard 1
Story 1