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. The insurance industry must also take this analysis seriously. This analysis may be used by healthcare insurance providers to make a variety of strategic and tactical decisions.

Objective:

The objective of this project is to predict patients' healthcare costs and to identify factors contributing to this prediction. It will also be useful to learn the interdependencies of different factors and comprehend the significance of various tools at various stages of the healthcare cost prediction process.

Project Task: Week 1

In [1]:
```
#Importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder
import bokeh as bk
import plotly.express as px
from datetime import datetime
from sklearn.metrics import r2_score
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score, KFold
```

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

In [2]:
```
#Importing dataset
hosp_df=pd.read_csv('Hospitalisation details.csv')
```

In [3]:
```
hosp_df.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]:
```
med_df=pd.read_csv('Medical Examinations.csv')
```

In [5]:
```
med_df.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]:
```
name_df=pd.read_excel('Names.xlsx')
```

In [7]:
```
name_df.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]:
```
semi_df=pd.merge(hosp_df,med_df,on='Customer ID')
```

In [9]:
```
semi_df
```

Out[9]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurge
0	Id2335	1992	Jul	9	0	563.84	tier - 2	tier - 3	R1013	17.580	4.51	No	No	No	
1	Id2334	1992	Nov	30	0	570.62	tier - 2	tier - 1	R1013	17.600	4.39	No	No	No	
2	Id2333	1993	Jun	30	0	600.00	tier - 2	tier - 1	R1013	16.470	6.35	No	No	Yes	
3	Id2332	1992	Sep	13	0	604.54	tier - 3	tier - 3	R1013	17.700	6.28	No	No	No	
4	Id2331	1998	Jul	27	0	637.26	tier - 3	tier - 3	R1013	22.340	5.57	No	No	No	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2330	Id5	1989	Jun	19	0	55135.40	tier - 1	tier - 2	R1012	35.530	5.45	No	No	No	No major sur
2331	Id4	1991	Jun	6	1	58571.07	tier - 1	tier - 3	R1024	38.095	6.05	No	No	No	No major sur
2332	Id3	1970	?	11	3	60021.40	tier - 1	tier - 1	R1012	34.485	11.87	yes	No	No	
2333	Id2	1977	Jun	8	0	62592.87	tier - 2	tier - 3	R1013	30.360	5.77	No	No	No	No major sur
2334	Id1	1968	Oct	12	0	63770.43	tier - 1	tier - 3	R1013	47.410	7.47	No	No	No	No major sur

2335 rows × 16 columns

In [10]:
```
df=pd.merge(semi_df,name_df,on='Customer ID')
```

In [11]:
```
df.head()
```

Out[11]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries
0	Id2335	1992	Jul	9	0	563.84	tier - 2	tier - 3	R1013	17.58	4.51	No	No	No	1
1	Id2334	1992	Nov	30	0	570.62	tier - 2	tier - 1	R1013	17.60	4.39	No	No	No	1
2	Id2333	1993	Jun	30	0	600.00	tier - 2	tier - 1	R1013	16.47	6.35	No	No	Yes	1
3	Id2332	1992	Sep	13	0	604.54	tier - 3	tier - 3	R1013	17.70	6.28	No	No	No	1
4	Id2331	1998	Jul	27	0	637.26	tier - 3	tier - 3	R1013	22.34	5.57	No	No	No	1

2. Check for missing values in the dataset

In [12]:
```
df.isna().sum().any()
```

Out[12]: False

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

In [13]:
```
trivial_rows=(df=="?").sum(axis=0)
```

In [14]:
```
trival_col= (df=="?").sum(axis=1)
```

In [15]:
```
trivial_rows[trivial_rows>0].count()
```

Out[15]:
```
6
```

In [16]:
```
trival_col[trival_col>0].count()
```

Out[16]:
```
10
```

In [17]:
```
#percentage of trival rows
per=trivial_rows[trivial_rows>0].sum()/df.shape[0]*100
```

In [18]:
```
per
```

Out[18]:
```
0.47109207708779444
```

In [19]:
```
#Deleting trival value from rows
#I have replace ? from the nan
df=df.replace('?',pd.NaT)
```

In [20]:
```
df.shape  #before deleting the missing values
```

Out[20]:
```
(2335, 17)
```

In [21]:
```
df.dropna(inplace=True)
```

In [22]:
```
df.shape #final shape with no missing values anymore
```

Out[22]:
```
(2325, 17)
```

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

In [23]:
```
df.info() #Mostly our data is categorical in nature
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2325 entries, 0 to 2334
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Customer ID             2325 non-null   object 
 1   year                    2325 non-null   object 
 2   month                   2325 non-null   object 
 3   date                    2325 non-null   int64  
 4   children                2325 non-null   int64  
 5   charges                 2325 non-null   float64
 6   Hospital tier           2325 non-null   object 
 7   City tier               2325 non-null   object 
 8   State ID                2325 non-null   object 
 9   BMI                     2325 non-null   float64
 10  HBA1C                   2325 non-null   float64
 11  Heart Issues            2325 non-null   object 
 12  Any Transplants         2325 non-null   object 
 13  Cancer history          2325 non-null   object 
 14  NumberOfMajorSurgeries  2325 non-null   object 
 15  smoker                  2325 non-null   object 
 16  name                    2325 non-null   object 
dtypes: float64(3), int64(2), object(12)
memory usage: 327.0+ KB
```

In [24]:
```
df
```

Out[24]:

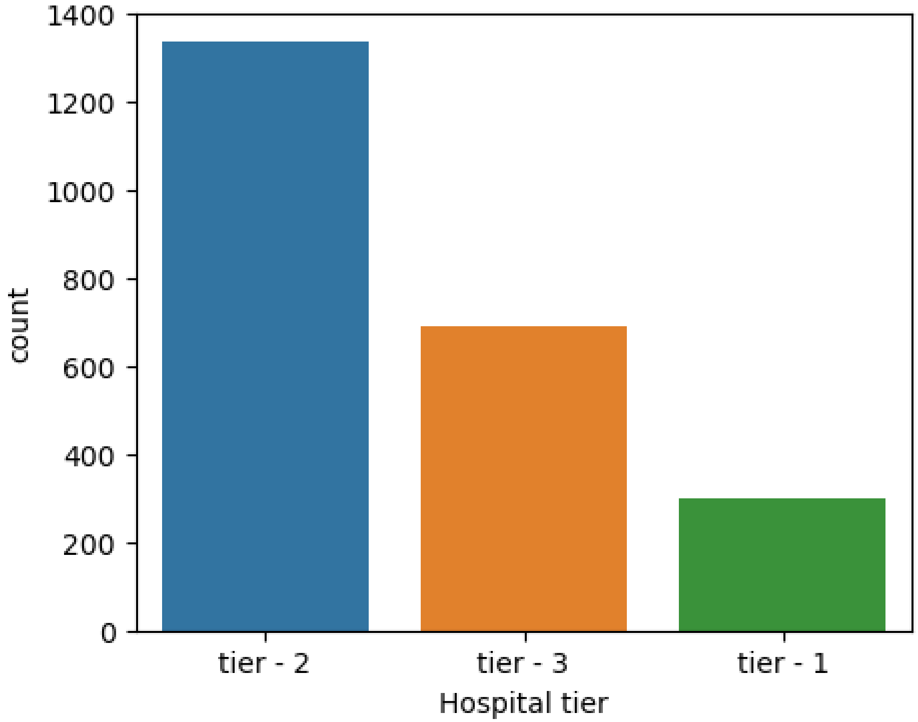
	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurge
0	Id2335	1992	Jul	9	0	563.84	tier - 2	tier - 3	R1013	17.580	4.51	No	No	No	
1	Id2334	1992	Nov	30	0	570.62	tier - 2	tier - 1	R1013	17.600	4.39	No	No	No	
2	Id2333	1993	Jun	30	0	600.00	tier - 2	tier - 1	R1013	16.470	6.35	No	No	Yes	
3	Id2332	1992	Sep	13	0	604.54	tier - 3	tier - 3	R1013	17.700	6.28	No	No	No	
4	Id2331	1998	Jul	27	0	637.26	tier - 3	tier - 3	R1013	22.340	5.57	No	No	No	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2329	Id6	1962	Aug	4	0	52590.83	tier - 1	tier - 3	R1011	32.800	6.59	No	No	No	No major sur
2330	Id5	1989	Jun	19	0	55135.40	tier - 1	tier - 2	R1012	35.530	5.45	No	No	No	No major sur
2331	Id4	1991	Jun	6	1	58571.07	tier - 1	tier - 3	R1024	38.095	6.05	No	No	No	No major sur
2333	Id2	1977	Jun	8	0	62592.87	tier - 2	tier - 3	R1013	30.360	5.77	No	No	No	No major sur
2334	Id1	1968	Oct	12	0	63770.43	tier - 1	tier - 3	R1013	47.410	7.47	No	No	No	No major sur

2325 rows × 17 columns

In [25]:
```
df['Hospital tier'].unique()
```

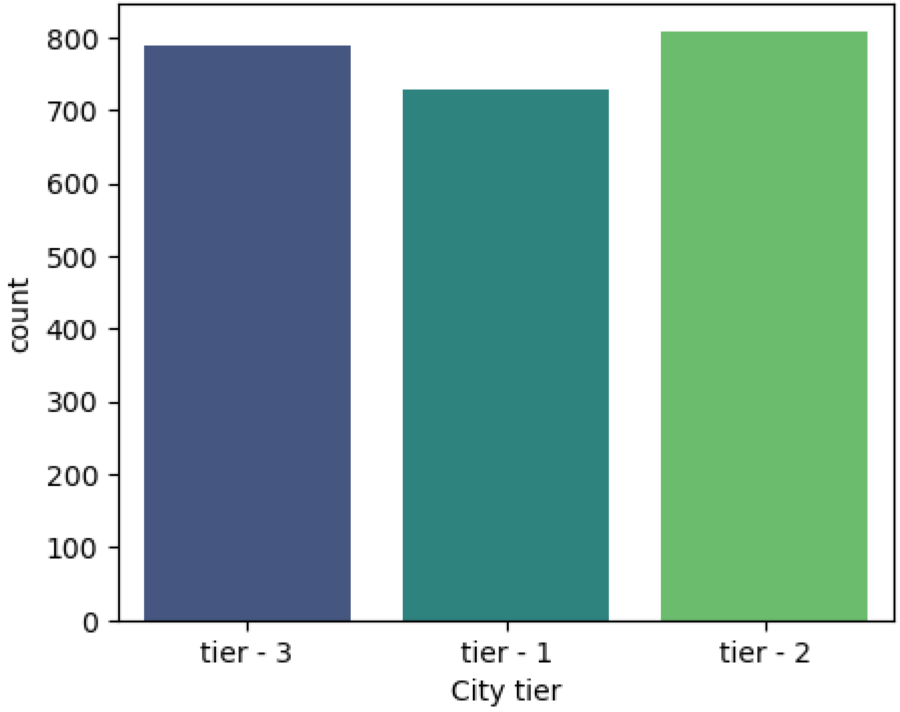
Out[25]:
```
array(['tier - 2', 'tier - 3', 'tier - 1'], dtype=object)
```

In [26]:
```
plt.figure(figsize=(5,4))
sns.countplot(df['Hospital tier'])
plt.show()
```



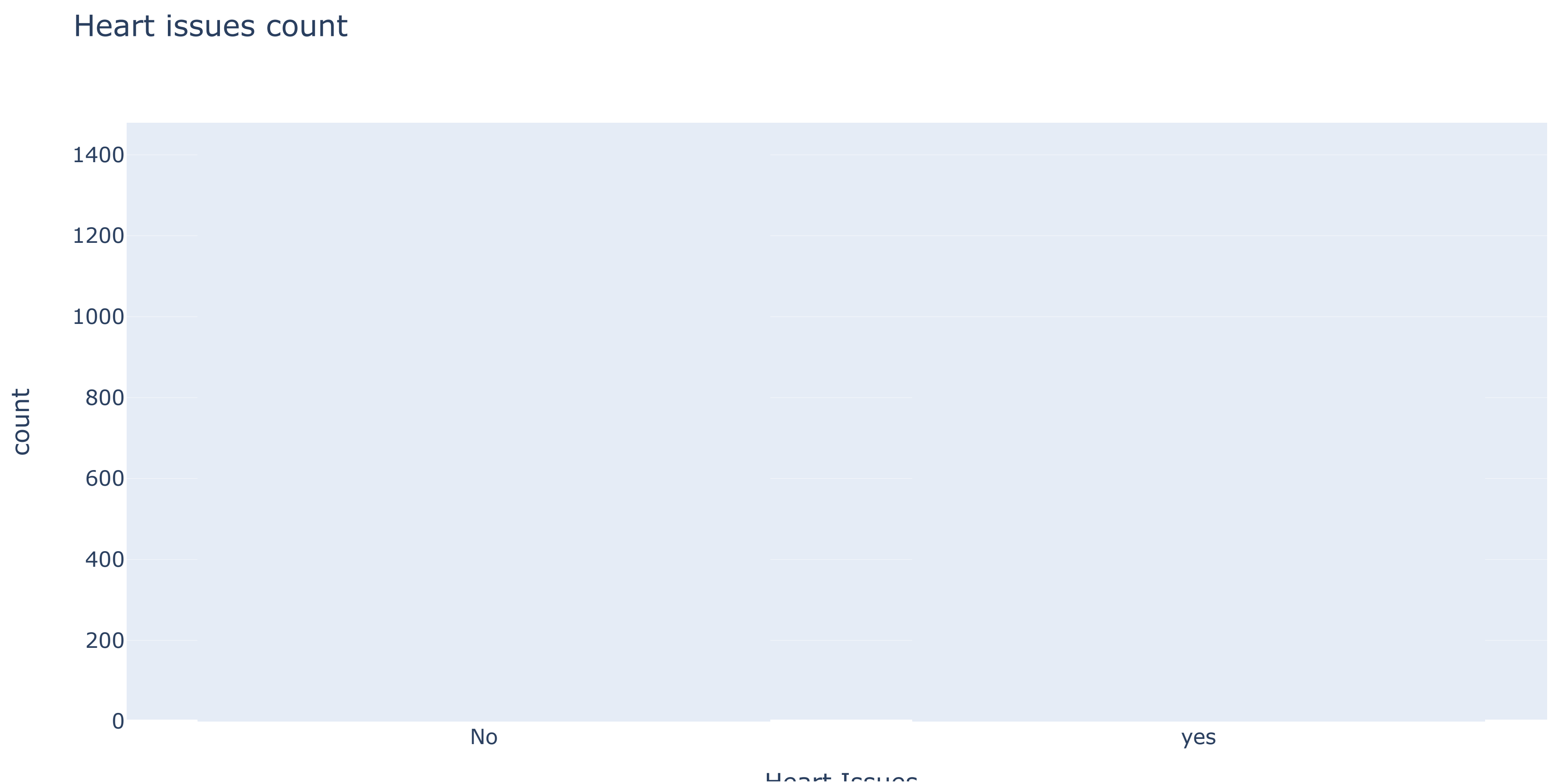
Count of Tier 2 hospitals are higher followed by tier 3 and tier 1.

In [27]:
```
plt.figure(figsize=(5,4))
df['City tier'].unique()
sns.countplot(df['City tier'],palette='viridis')
plt.show()
```



In [28]:

```
values=df['Heart Issues'].unique()
plt.figure(figsize=(5,4))
# Create a bar chart using plotly
fig = px.bar(df, x='Heart Issues',title='Heart issues count')

# Show the chart
fig.show()
```



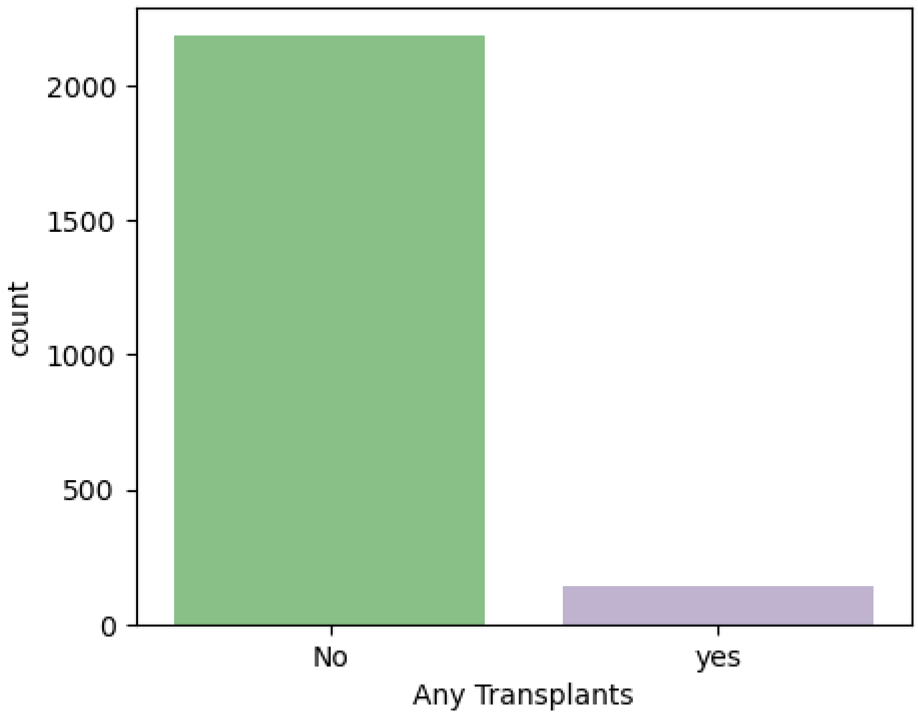
<Figure size 500x400 with 0 Axes>

In [29]:
```
le=LabelEncoder()
df['Heart Issues']=le.fit_transform(df['Heart Issues'])
```

In [30]:
```
df['Any Transplants'].unique()
```

Out[30]: array(['No', 'yes'], dtype=object)

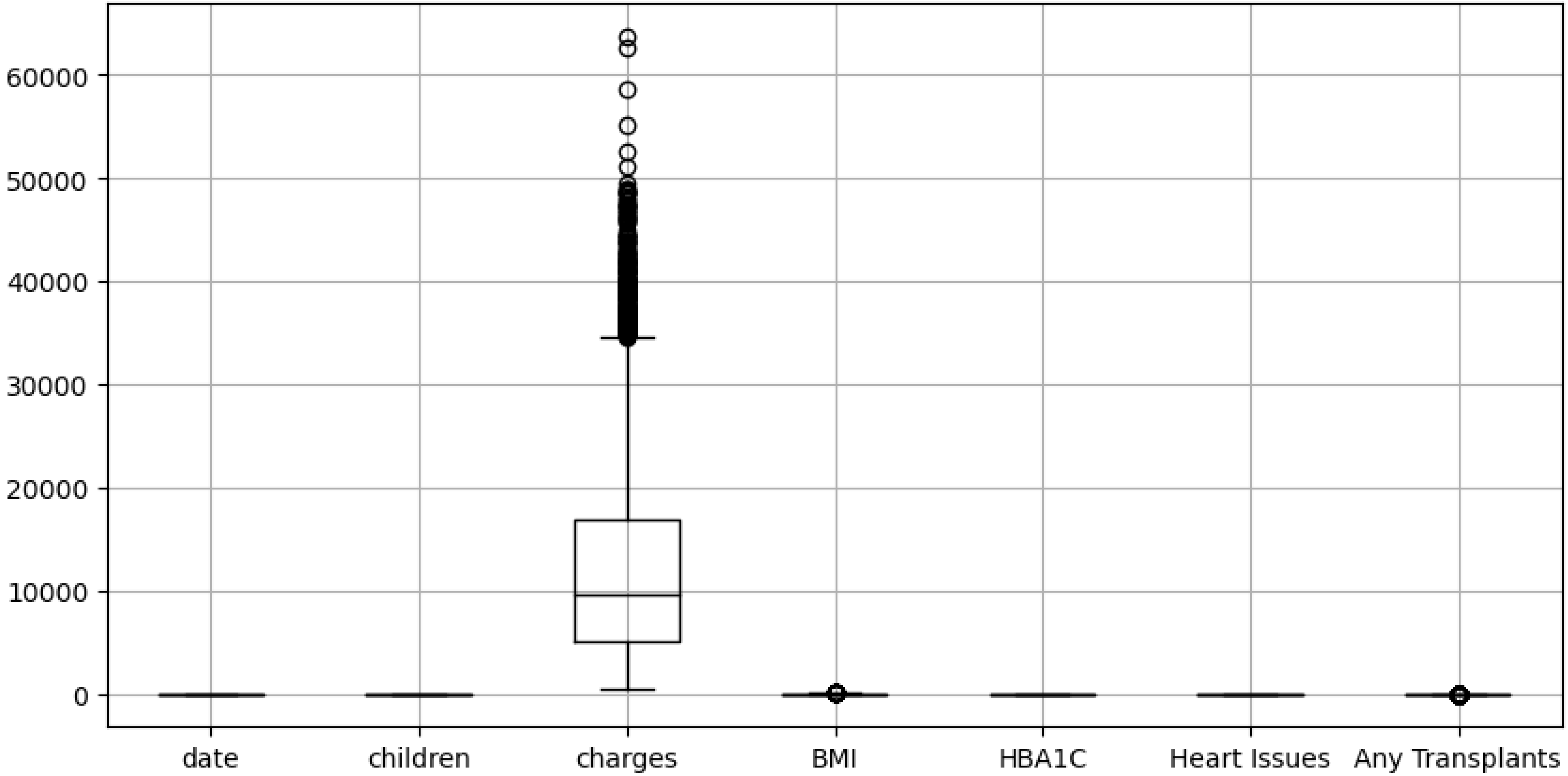
In [31]:
```
plt.figure(figsize=(5,4))
sns.countplot(df['Any Transplants'],palette='Accent')
plt.show()
```



In [32]:
```
#LabelEncoder for City tier
le=LabelEncoder()
df['Any Transplants']=le.fit_transform(df['Any Transplants'])
print(df['Any Transplants'])
```

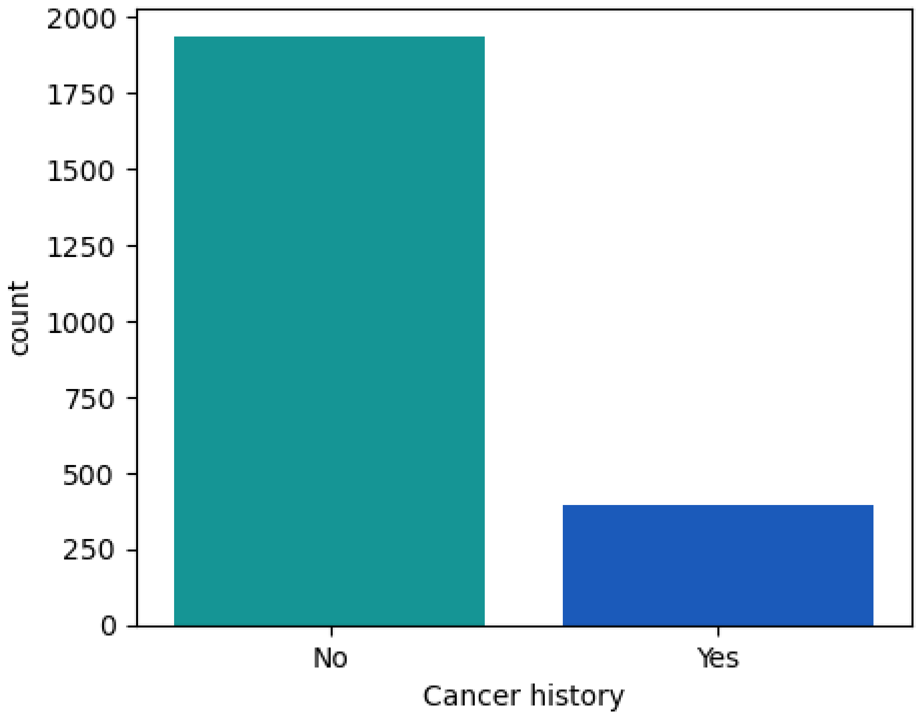
```
0       0
1       0
2       0
3       0
4       0
       ..
2329    0
2330    0
2331    0
2333    0
2334    0
Name: Any Transplants, Length: 2325, dtype: int32
```

In [33]:
```
df.boxplot(figsize=(10,5),color='k')
plt.show()
```



As we can see that there are major outliers in the charges and no outliers in other attributes

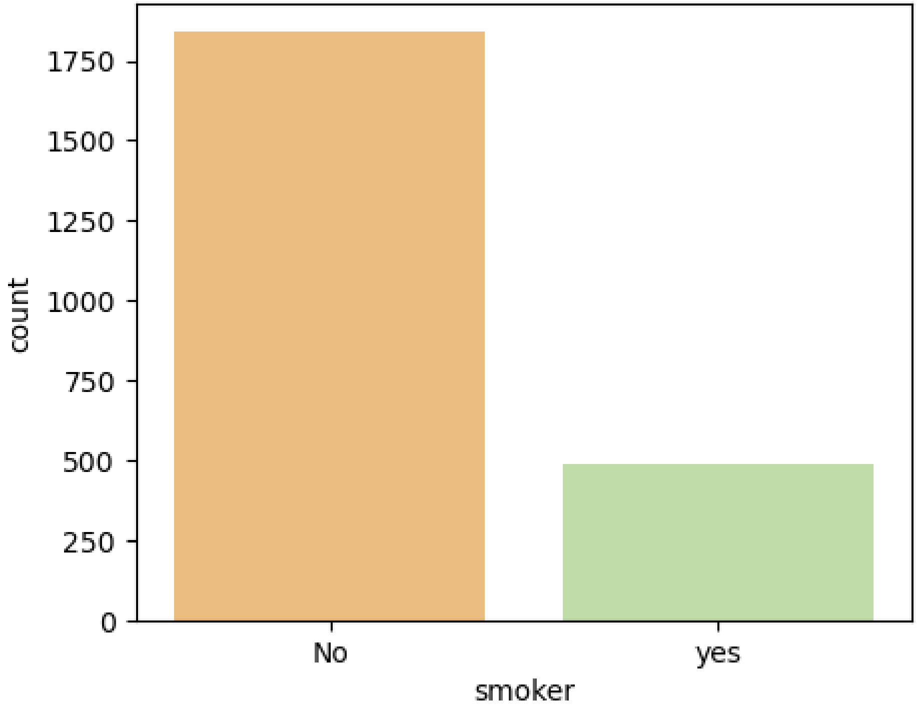
In [34]:
```
plt.figure(figsize=(5,4))
sns.countplot(df['Cancer history'],palette='winter_r')
plt.show()
```



As we can see that in our dataset there is no such Cancer History history patient.

In [35]:
```
#let's encode it
le=LabelEncoder()
df['Cancer history']=le.fit_transform(df['Cancer history'])
```

In [36]:
```
#Dealing with smoker attribute
plt.figure(figsize=(5,4))
sns.countplot(df['smoker'],palette='Spectral')
plt.show()
```



Majorly there are non-smokers.

In [37]:
```
#let's encode it
le=LabelEncoder()
df['smoker']=le.fit_transform(df['smoker'])
```

5. The dataset has State ID, which has around 16 states. All states are not represented inequal 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.

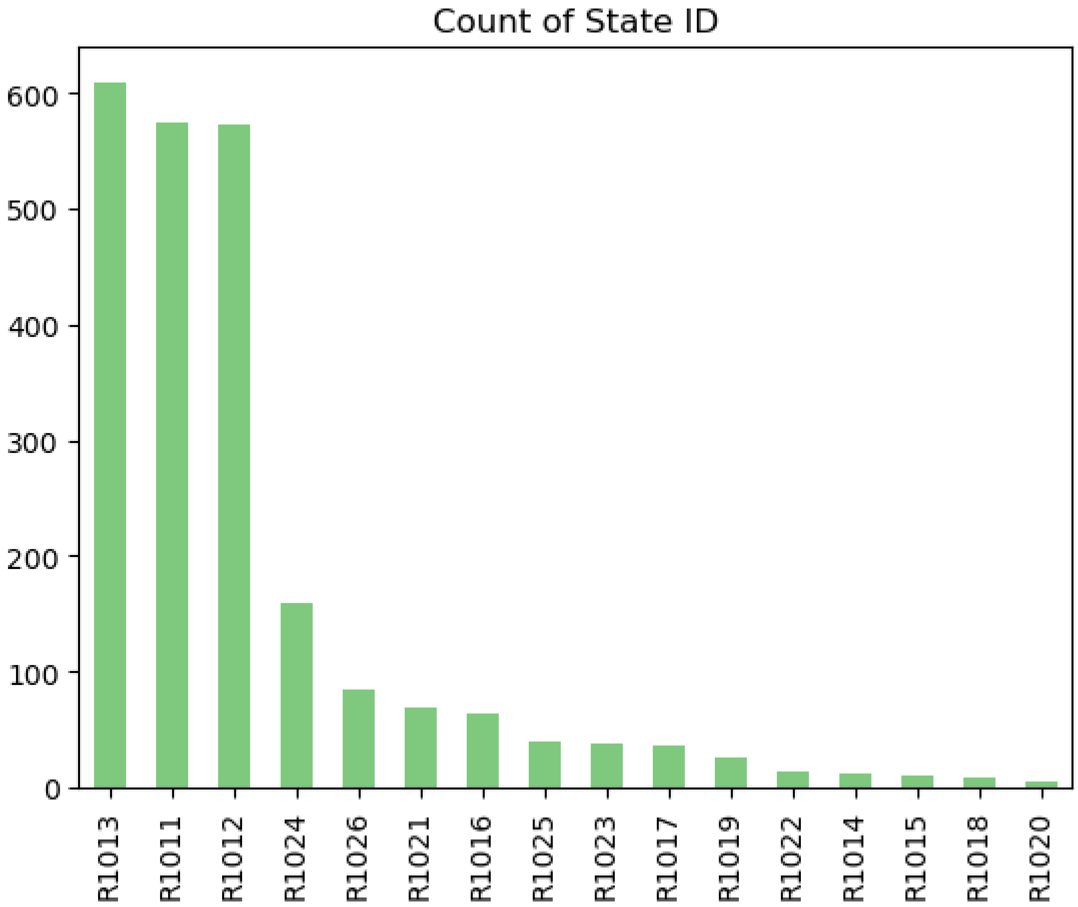
In [38]:

```python
df['State ID'].unique()
```

Out[38]:

```
array(['R1013', 'R1012', 'R1011', 'R1015', 'R1019', 'R1016', 'R1018',
       'R1025', 'R1024', 'R1023', 'R1014', 'R1021', 'R1017', 'R1020',
       'R1026', 'R1022'], dtype=object)
```

In [39]:
```
df['State ID'].value_counts().plot(kind='bar',cmap='Accent')
plt.title('Count of State ID')
plt.show()
```



In [40]:
```python
# Define the regions of interest
regions = ['R1011', 'R1012', 'R1013']

# Create dummy variables for the regions of interest
dummies = pd.get_dummies(df['State ID'][df['State ID'].isin(regions)], prefix='Region')

# Join the dummy variables to the original dataset
data = pd.concat([df, dummies], axis=1)

# Drop the original State ID column
data = data.drop('State ID', axis=1)
```

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

In [41]:
```python
data['NumberOfMajorSurgeries'].unique()
```

Out[41]:
```
array(['1', 'No major surgery', '2', '3'], dtype=object)
```

In [42]:
```python
#replacing no major surgery into 0
data['NumberOfMajorSurgeries']=data['NumberOfMajorSurgeries'].replace('No major surgery',0)
```

In [43]:
```python
data['NumberOfMajorSurgeries'].unique()
```

Out[43]:
```
array(['1', 0, '2', '3'], dtype=object)
```

In [44]:
```
#converting the datatype into int
data['NumberOfMajorSurgeries'].astype('int')
```

Out[44]:
```
0       1
1       1
2       1
3       1
4       1
       ..
2329    0
2330    0
2331    0
2333    0
2334    0
Name: NumberOfMajorSurgeries, Length: 2325, dtype: int32
```

In [45]:
```
data
```

Out[45]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	sı
0	Id2335	1992	Jul	9	0	563.84	tier - 2	tier - 3	17.580	4.51	0	0	0	1	
1	Id2334	1992	Nov	30	0	570.62	tier - 2	tier - 1	17.600	4.39	0	0	0	1	
2	Id2333	1993	Jun	30	0	600.00	tier - 2	tier - 1	16.470	6.35	0	0	1	1	
3	Id2332	1992	Sep	13	0	604.54	tier - 3	tier - 3	17.700	6.28	0	0	0	1	
4	Id2331	1998	Jul	27	0	637.26	tier - 3	tier - 3	22.340	5.57	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2329	Id6	1962	Aug	4	0	52590.83	tier - 1	tier - 3	32.800	6.59	0	0	0	0	
2330	Id5	1989	Jun	19	0	55135.40	tier - 1	tier - 2	35.530	5.45	0	0	0	0	
2331	Id4	1991	Jun	6	1	58571.07	tier - 1	tier - 3	38.095	6.05	0	0	0	0	
2333	Id2	1977	Jun	8	0	62592.87	tier - 2	tier - 3	30.360	5.77	0	0	0	0	
2334	Id1	1968	Oct	12	0	63770.43	tier - 1	tier - 3	47.410	7.47	0	0	0	0	

2325 rows × 19 columns

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

In [46]:
```
# Combine the year, month, and date columns into a single datetime column
data['birthdate'] = pd.to_datetime(data[['year', 'month', 'date']].astype(str).agg('-'.join, axis=1))

# Calculate the current age of each patient
data['age'] = (datetime.now() - data['birthdate']) // pd.Timedelta(days=365.25)
```

In [47]:
```
data
```

Out[47]:

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	BMI	HBA1C	...	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoke
0	Id2335	1992	Jul	9	0	563.84	tier - 2	tier - 3	17.580	4.51	...	0	0	1	(
1	Id2334	1992	Nov	30	0	570.62	tier - 2	tier - 1	17.600	4.39	...	0	0	1	(
2	Id2333	1993	Jun	30	0	600.00	tier - 2	tier - 1	16.470	6.35	...	0	1	1	(
3	Id2332	1992	Sep	13	0	604.54	tier - 3	tier - 3	17.700	6.28	...	0	0	1	(
4	Id2331	1998	Jul	27	0	637.26	tier - 3	tier - 3	22.340	5.57	...	0	0	1	(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
2329	Id6	1962	Aug	4	0	52590.83	tier - 1	tier - 3	32.800	6.59	...	0	0	0	'
2330	Id5	1989	Jun	19	0	55135.40	tier - 1	tier - 2	35.530	5.45	...	0	0	0	'
2331	Id4	1991	Jun	6	1	58571.07	tier - 1	tier - 3	38.095	6.05	...	0	0	0	'
2333	Id2	1977	Jun	8	0	62592.87	tier - 2	tier - 3	30.360	5.77	...	0	0	0	'
2334	Id1	1968	Oct	12	0	63770.43	tier - 1	tier - 3	47.410	7.47	...	0	0	0	'

2325 rows × 21 columns

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 [48]:
```
def extract_gender(name):
    # Define a dictionary to map salutations to gender
    salutations = {'Mr.': 'Male', 'Ms.': 'Female', 'Mrs.': 'Female', 'Miss.': 'Female'}

    # Check if any salutation is present in the name
    for salutation in salutations:
        if salutation in name:
            return salutations[salutation]

    # If no salutation is found, return None
    return None
```

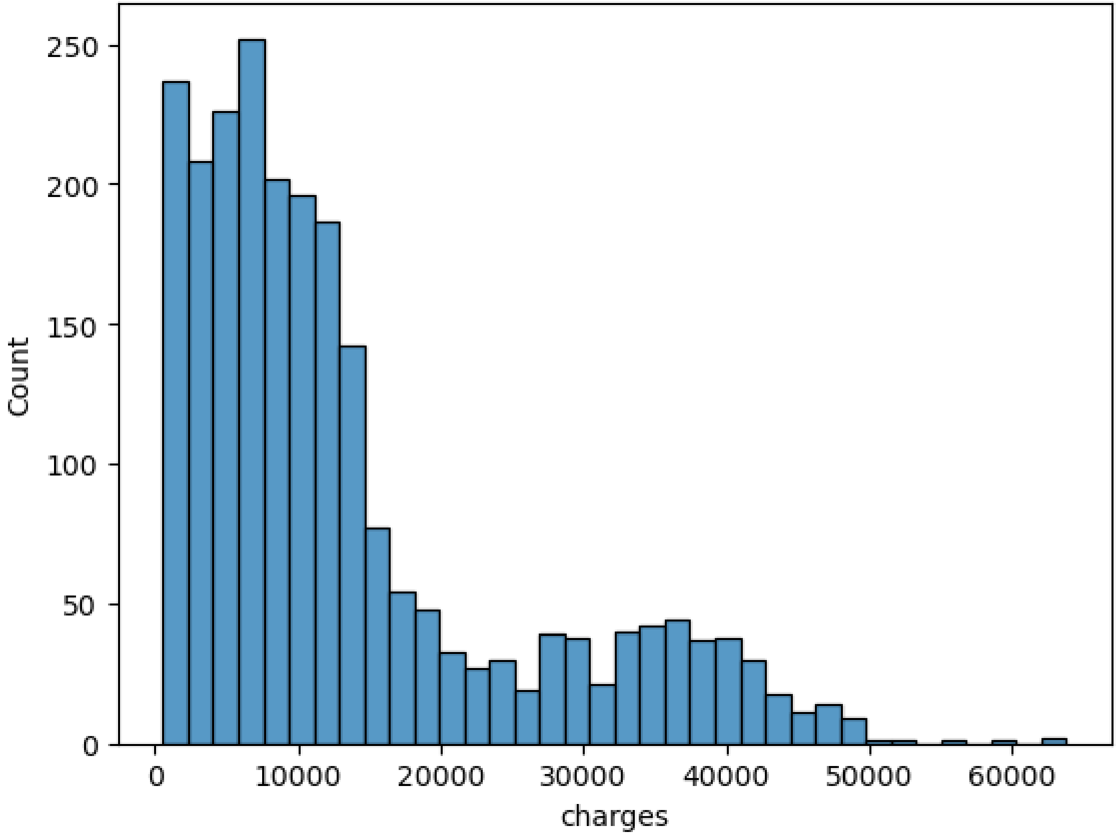
In [49]:
```
data['Gender'] = data['name'].apply(lambda x: extract_gender(x))
```

In [50]:
```
data['Gender'].value_counts()
```

Out[50]:
```
Female    1165
Male      1160
Name: Gender, dtype: int64
```

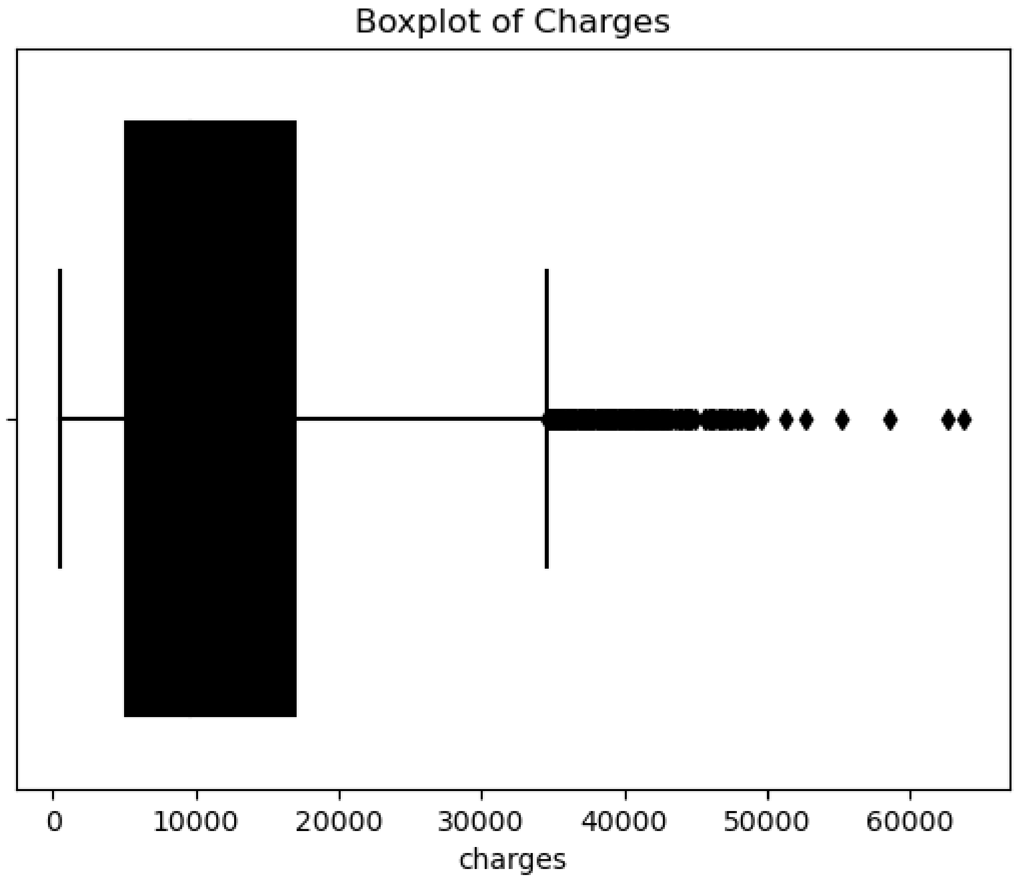
9. You should also visualize the distribution of costs using a histogram, box and whisker plot, and swarm plot.

In [51]:
```
sns.histplot(data=data,x='charges')
plt.show()
```



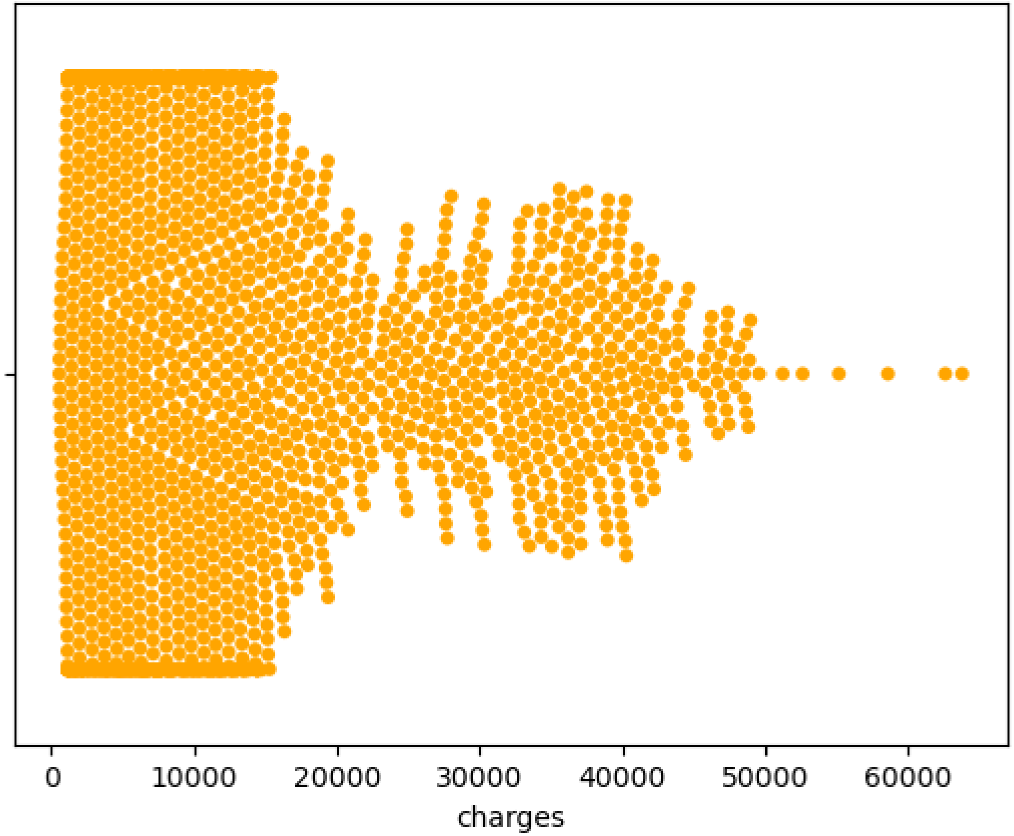
We can say that charges are positively skewed

In [52]:
```
sns.boxplot(data=data,x='charges',color='k')
plt.title('Boxplot of Charges')
plt.show()
```



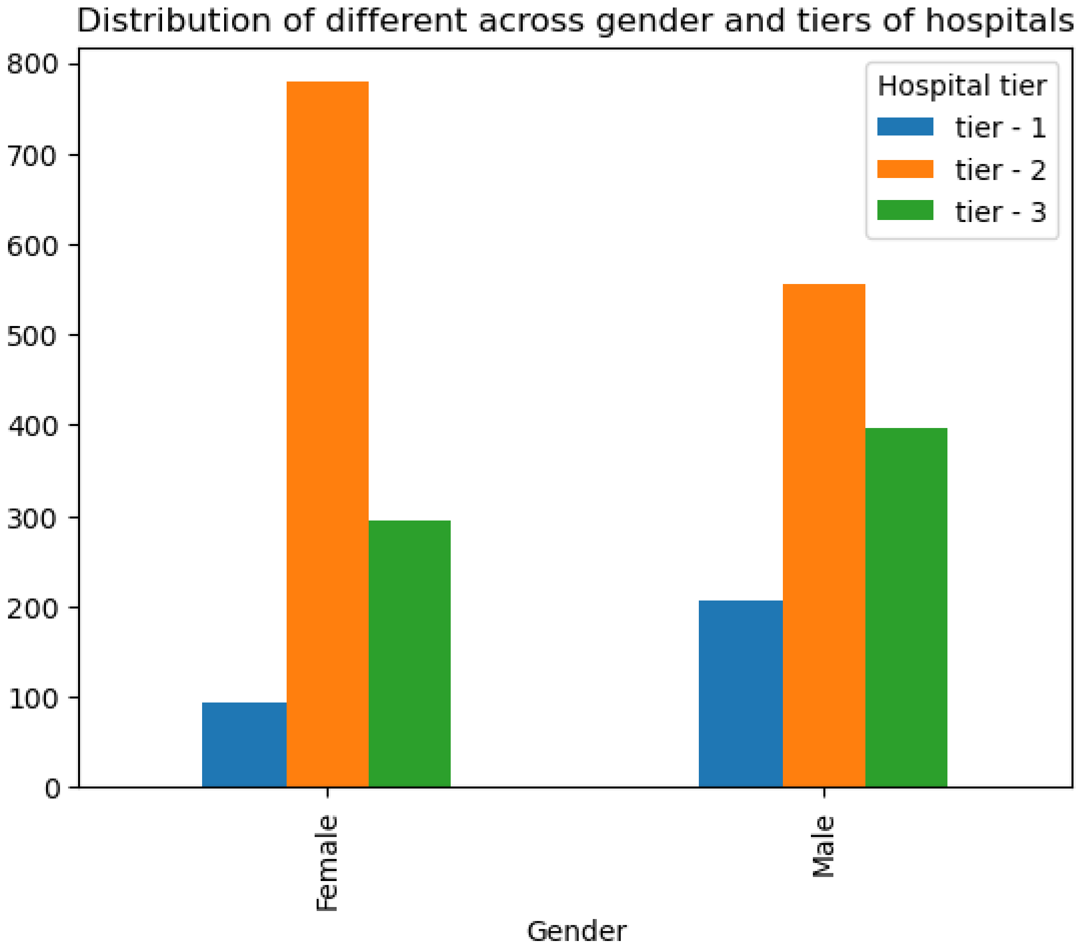
There are outliers present in the charges attribute

In [53]:
```
sns.swarmplot(data=df,x='charges',color='orange')
plt.show()
```



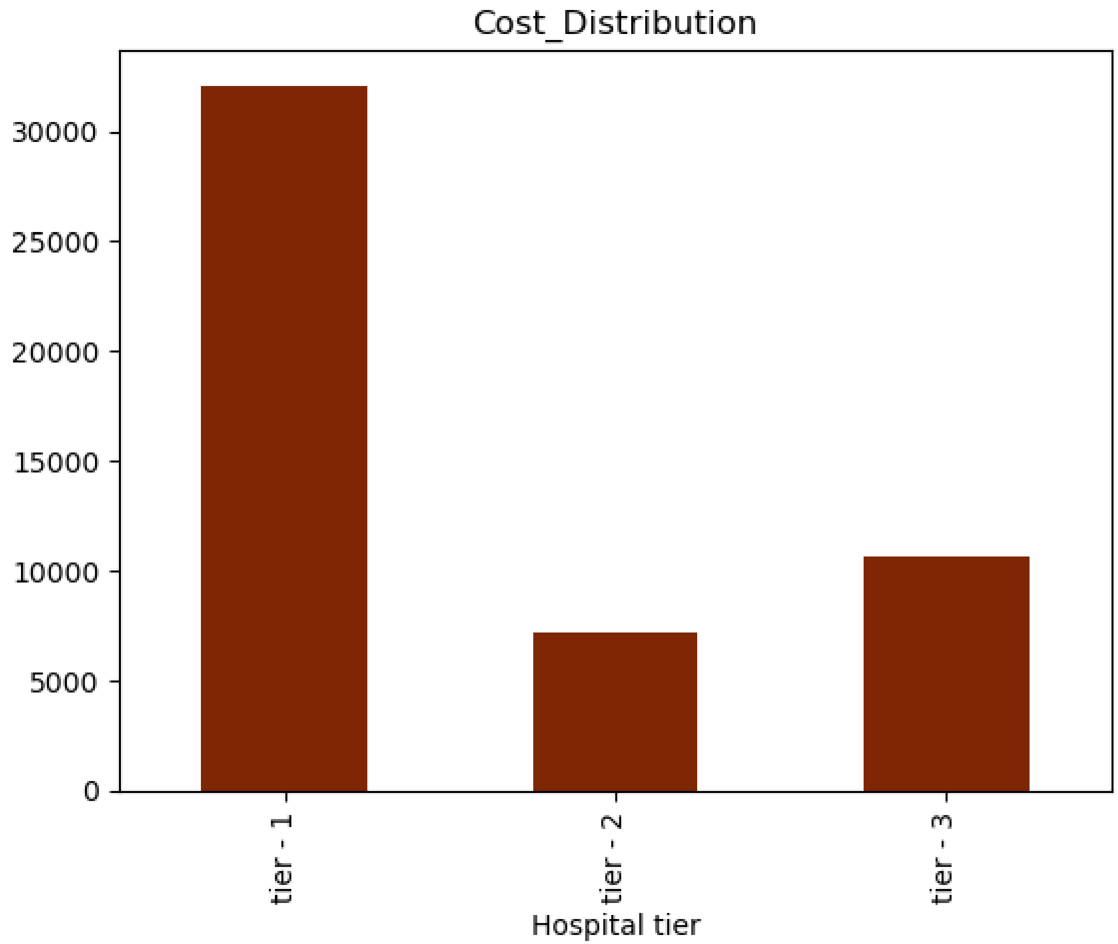
10. State how the distribution is different across gender and tiers of hospitals

In [54]:
```
data.groupby('Gender')['Hospital tier'].value_counts().unstack().plot(kind='bar')
plt.title("Distribution of different across gender and tiers of hospitals")
plt.show()
```



As we can say that both in female and male tier-2 hospitals are leading one and followed by tier-3 but tier-1 which is one of the most developed hospital have lessor count of patients the reason might the cost of treatment might be high in such hospital.

In [55]:
```
data.groupby('Hospital tier')['charges'].median().plot(kind='bar',cmap='Oranges_r')
plt.title("Cost_Distribution")
plt.show()
```



In tier-1 hospital the median of charges is higher in tier-1 hospital compare to tier-2 and tier-3 and we can come to the conclusion that as we have seen the most of the female and male has huge ratio of treatment in tier-2 hospital the reason might be cost which i have clarified by above chart but i can say there will be multiple factors affecting the same but the charges might be one of the dominant factor .

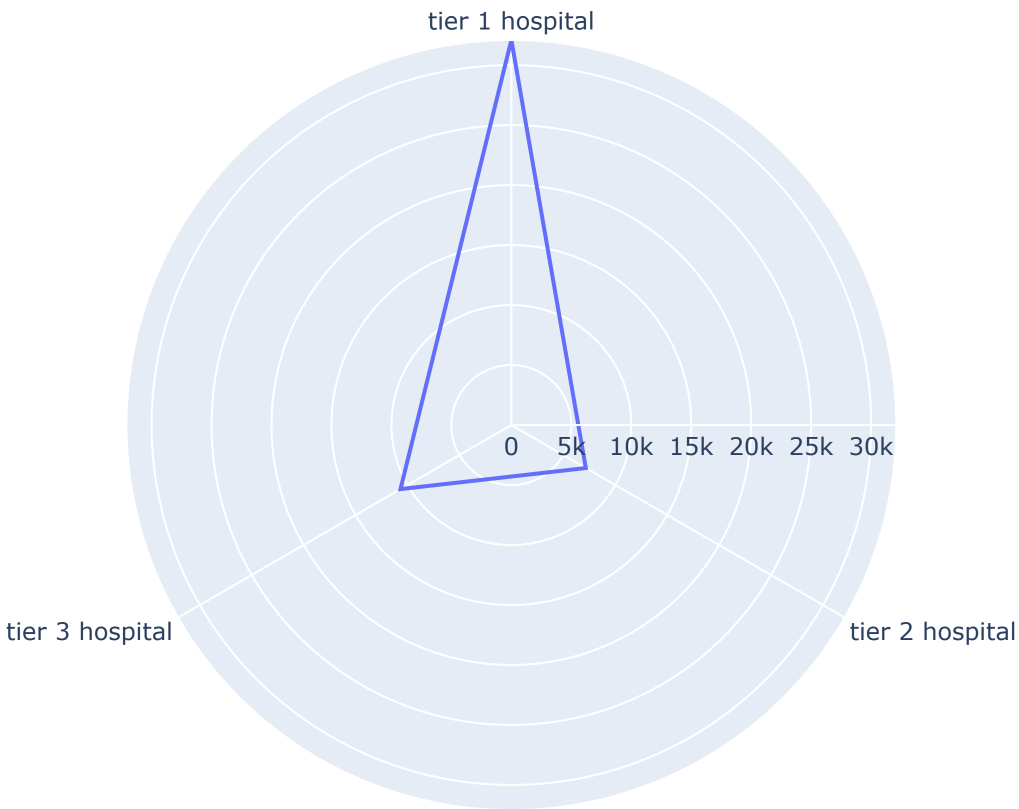
11. Create a radar chart to showcase the median hospitalization cost for each tier of hospitals

In [56]:
```
data.groupby('Hospital tier')['charges'].median()
```

Out[56]:
```
Hospital tier
tier - 1    32097.435
tier - 2     7168.760
tier - 3    10676.830
Name: charges, dtype: float64
```

In [57]:
```
radar= pd.DataFrame(dict(r=[32097.43, 7168.76, 10676.83],theta=['tier 1 hospital','tier 2 hospital','tier 3 hospital']
```

In [58]:
```
fig = px.line_polar(radar, r='r', theta='theta', line_close=True)
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

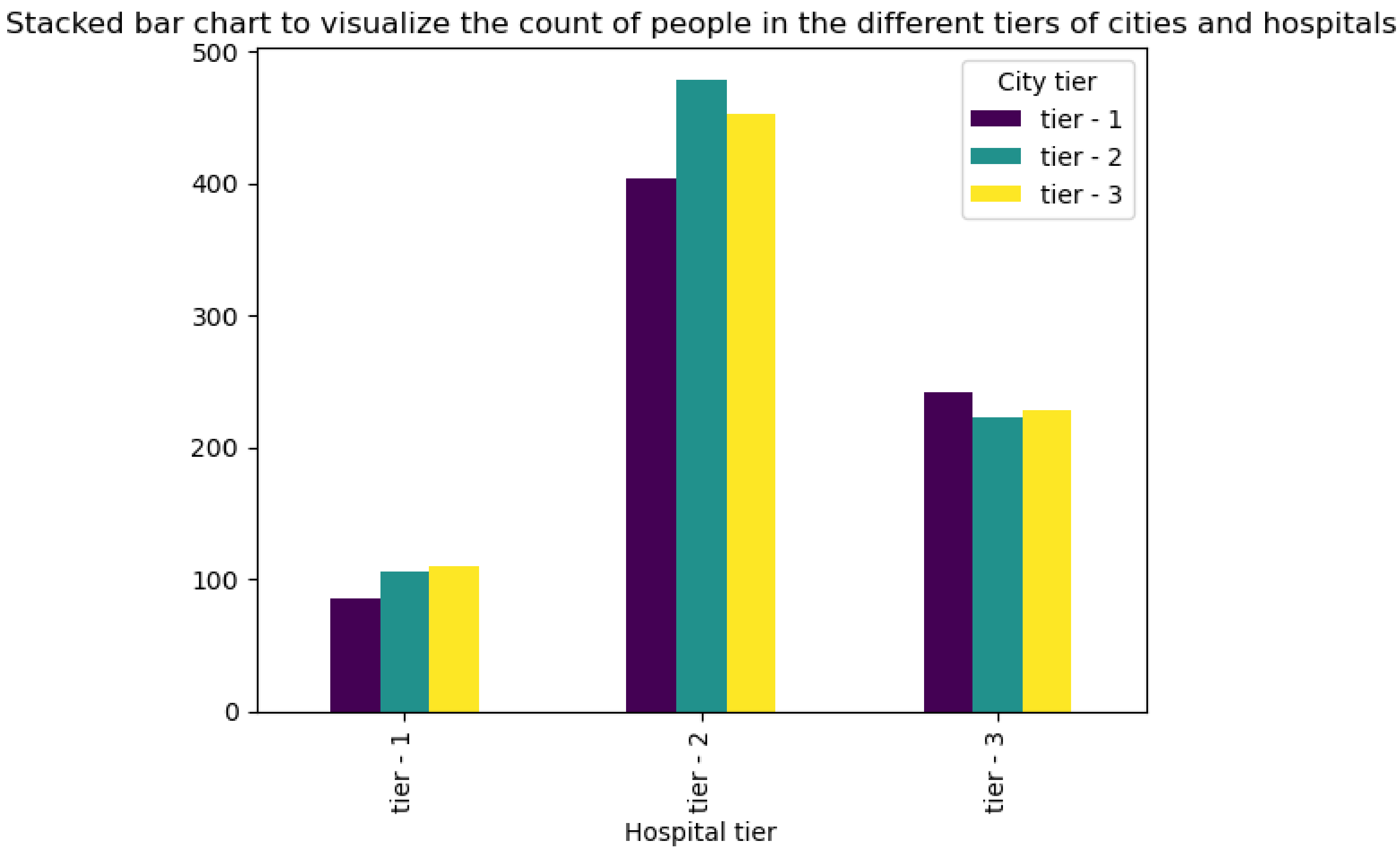
In [59]:

```
data.groupby(['Hospital tier','City tier'])['Customer ID'].count().to_frame()
```

Out[59]:

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

In [60]:
```
data.groupby(['Hospital tier','City tier'])['Customer ID'].count().unstack().plot(kind='bar',cmap='viridis')
plt.title('Stacked bar chart to visualize the count of people in the different tiers of cities and hospitals')
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

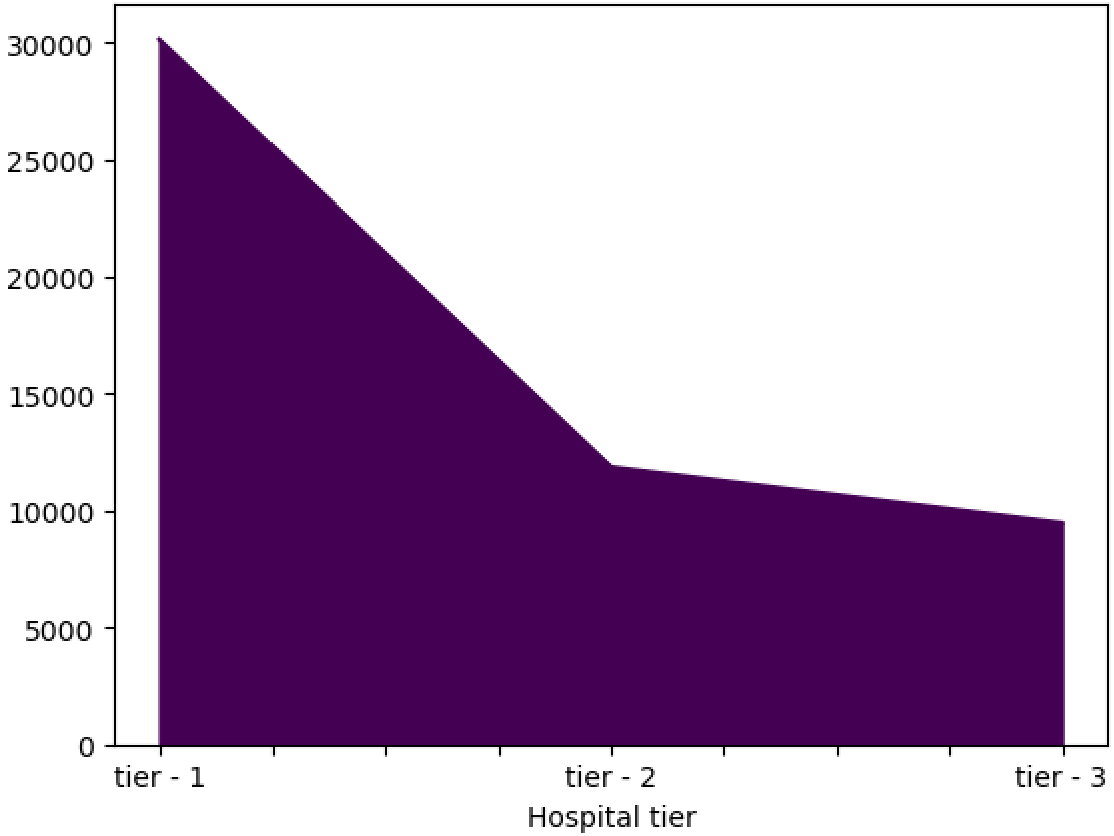
a. The average hospitalization costs for the three types of hospitals are not significantly different

In [61]:
```
data.groupby(['Hospital tier'])['charges'].mean().to_frame()
```

Out[61]:

Hospital tier	charges
tier - 1	30131.995900
tier - 2	11875.883861
tier - 3	9487.456223

In [62]:
```
#a. The average hospitalization costs for the three types of hospitals are not significantly different.
data.groupby(['Hospital tier'])['charges'].mean().plot(kind='area',cmap='viridis')
plt.show()
```



In [63]:
```
from scipy.stats import kruskal #This test is a non-parametric alternative to one-way ANOVA, and can be used to compar

data1= [30131.995900]
data2= [11875.883861]
data3= [9487.456223]

stat, p = kruskal(data1,data2,data3)

if p > 0.05:
    print('The average hospitalization costs for the three types of hospitals are not significantly different')
    print("p_value:",round(p,2))
else:
    print('The average hospitalization costs for the three types of hospitals are significantly different')
```

```
The average hospitalization costs for the three types of hospitals are not significantly different
p_value: 0.37
```

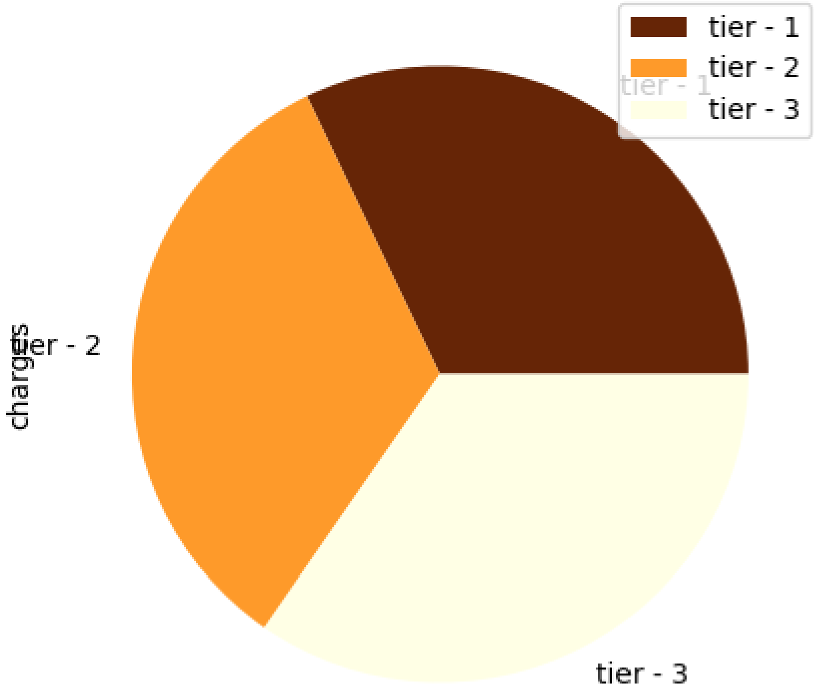
b. The average hospitalization costs for the three types of cities are not significantly different

In [64]:
```
data.groupby(['City tier'])['charges'].mean().to_frame()
```

Out[64]:

	charges
City tier	
tier - 1	13009.972579
tier - 2	13471.919281
tier - 3	14045.312066

In [65]:
```
data.groupby(['City tier'])['charges'].mean().plot(kind='pie',cmap='YlOrBr_r',figsize=(10, 5))
plt.legend()
plt.show()
```



In [66]:
```
#Let's do a hypothesis testing
from scipy.stats import kruskal #This test is a non-parametric alternative to one-way ANOVA, and can be used to compar
data1 =[13009.972579]
data2 =[13471.91928]
data3 =[14045.312066]

stat, p = kruskal(data1,data2,data3)

if p > 0.05:
    print('The average hospitalization costs for the three types of cities are not significantly different')
    print("p-value",round(p,1))
else:
    print('The average hospitalization costs for the three types of cities are significantly different')
```

```
The average hospitalization costs for the three types of cities are not significantly different
p-value 0.4
```

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

In [67]:
```
print("Hospitalization charges for non smokers",data[data['smoker']==0].groupby('smoker')['charges'].mean()[0])
print("Hospitalization charges for smokers",data[data['smoker']==1].groupby('smoker')['charges'].mean()[1])
```

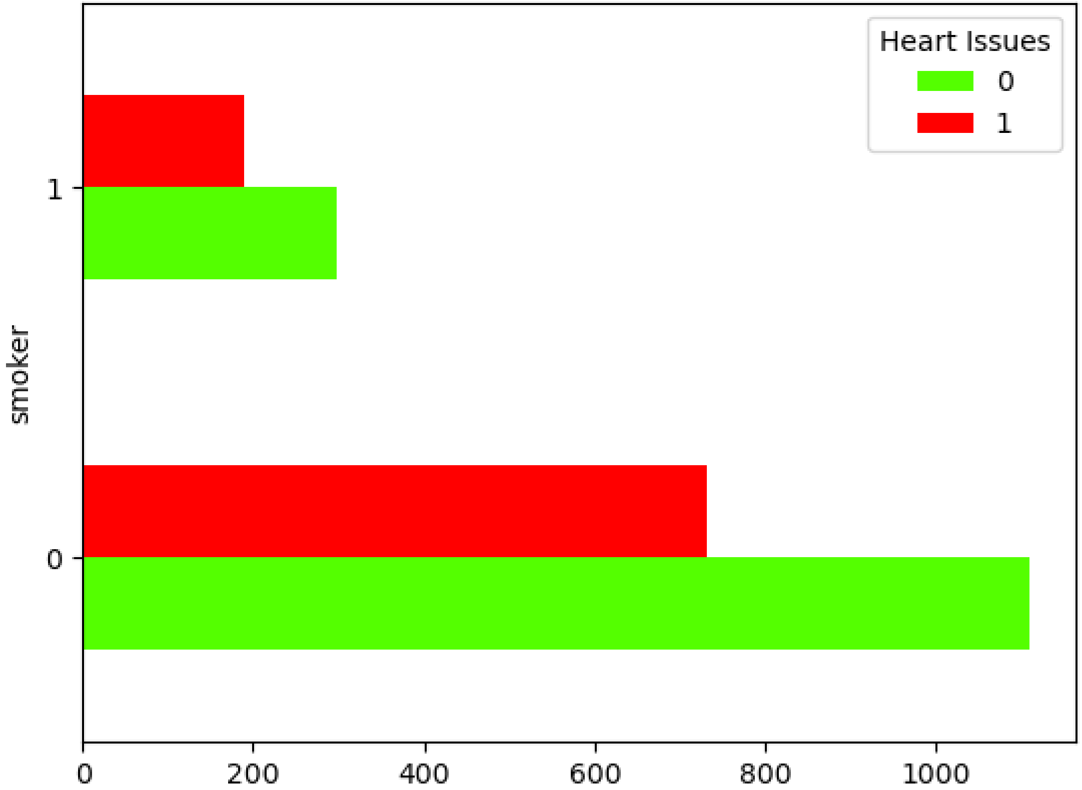
```
Hospitalization charges for non smokers 8409.19924959217
Hospitalization charges for smokers 32866.96022633745
```

In [68]:
```
import scipy.stats as stats

# Extract hospitalization costs for smokers and nonsmokers
cost_smokers = data[data['smoker'] == 1]['charges']
cost_nonsmokers = data[data['smoker'] == 0]['charges']

# Calculate the mean difference between the two groups
mean_diff = cost_smokers.mean() - cost_nonsmokers.mean()

# Perform two-sample t-test
t_stat, p_val = stats.ttest_ind(cost_smokers, cost_nonsmokers, equal_var=False)

# Check if the p-value is less than the significance level (alpha)
alpha = 0.05
if p_val > alpha:
    print('The average hospitalization cost for smokers is not significantly different from the average cost for nonsm
else:
    print('The average hospitalization cost for smokers is significantly different from the average cost for nonsmoker
```

```
The average hospitalization cost for smokers is significantly different from the average cost for nonsmokers.
```

d. Smoking and heart issues are independent or not

In [69]:
```
data.groupby('smoker')['Heart Issues'].value_counts().unstack().plot(kind='barh',cmap='prism_r')
plt.show()
```



In [70]:
```
import scipy.stats as stats

# Create a contingency table of observed frequencies
cont_table = pd.crosstab(data['smoker'], data['Heart Issues'])

# Perform chi-squared test
chi2_stat, p_val, dof, expected_freq = stats.chi2_contingency(cont_table)

# Check if the p-value is less than the significance level (alpha)
alpha = 0.05
if p_val > alpha:
    print('Smoking and heart issues are probably independent.')
else:
    print('Smoking and heart issues are probably dependent.')
```

```
Smoking and heart issues are probably independent.
```

Project Task: Week 2

Machine Learning

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

In [71]:
```
data['NumberOfMajorSurgeries']=data['NumberOfMajorSurgeries'].astype(int)
#LabelEncoder for hospital tier
le=LabelEncoder()
data['Hospital tier']=le.fit_transform(data['Hospital tier'])
data['City tier']=le.fit_transform(data['City tier'])
data['Gender']=le.fit_transform(data['Gender'])
```

In [72]:
```
data.dtypes
```

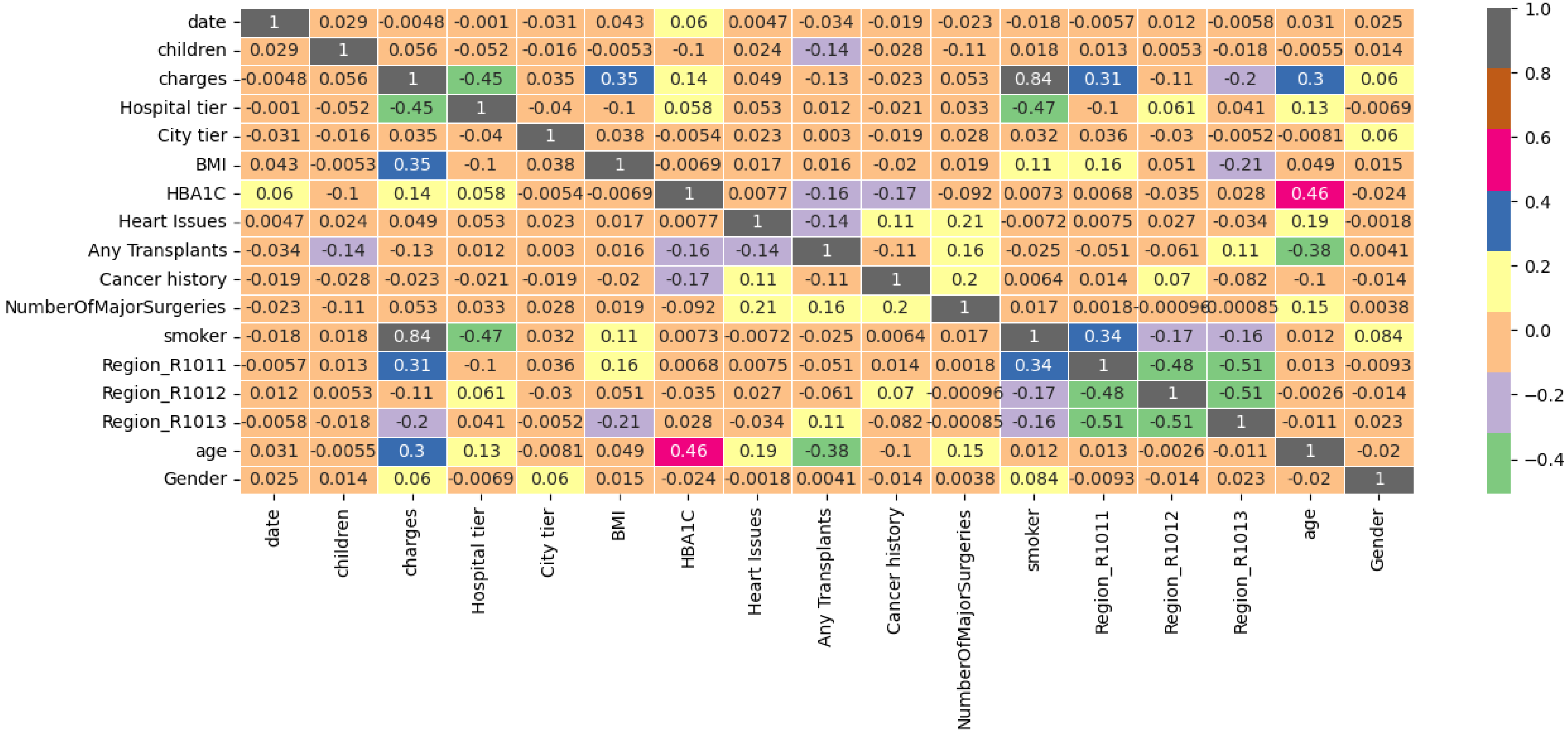
Out[72]:
```
Customer ID                      object
year                             object
month                            object
date                              int64
children                          int64
charges                         float64
Hospital tier                     int32
City tier                         int32
BMI                             float64
HBA1C                           float64
Heart Issues                      int32
Any Transplants                   int32
Cancer history                    int32
NumberOfMajorSurgeries            int32
smoker                            int32
name                             object
Region_R1011                    float64
Region_R1012                    float64
Region_R1013                    float64
birthdate                datetime64[ns]
age                               int64
Gender                            int32
dtype: object
```

In [73]:
```
data.corr()
```

Out[73]:

	date	children	charges	Hospital tier	City tier	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfM
date	1.000000	0.028668	-0.004844	-0.001017	-0.031421	0.042765	0.059789	0.004734	-0.033858	-0.018599	
children	0.028668	1.000000	0.055901	-0.052438	-0.015760	-0.005339	-0.101379	0.023984	-0.142040	-0.027880	
charges	-0.004844	0.055901	1.000000	-0.446687	0.035300	0.346730	0.139697	0.049299	-0.127028	-0.022522	
Hospital tier	-0.001017	-0.052438	-0.446687	1.000000	-0.039755	-0.104771	0.057855	0.053376	0.011729	-0.021429	
City tier	-0.031421	-0.015760	0.035300	-0.039755	1.000000	0.038123	-0.005404	0.023152	0.002970	-0.018639	
BMI	0.042765	-0.005339	0.346730	-0.104771	0.038123	1.000000	-0.006920	0.017129	0.015893	-0.020235	
HBA1C	0.059789	-0.101379	0.139697	0.057855	-0.005404	-0.006920	1.000000	0.007699	-0.159855	-0.170921	
Heart Issues	0.004734	0.023984	0.049299	0.053376	0.023152	0.017129	0.007699	1.000000	-0.140269	0.111190	
Any Transplants	-0.033858	-0.142040	-0.127028	0.011729	0.002970	0.015893	-0.159855	-0.140269	1.000000	-0.114677	
Cancer history	-0.018599	-0.027880	-0.022522	-0.021429	-0.018639	-0.020235	-0.170921	0.111190	-0.114677	1.000000	
NumberOfMajorSurgeries	-0.022525	-0.113161	0.053308	0.033230	0.027937	0.018851	-0.091594	0.206147	0.158593	0.204208	
smoker	-0.017523	0.017713	0.838462	-0.474077	0.032034	0.107126	0.007257	-0.007159	-0.025101	0.006415	
Region_R1011	-0.005670	0.013091	0.305872	-0.103434	0.035720	0.164390	0.006785	0.007493	-0.051057	0.013586	
Region_R1012	0.011583	0.005276	-0.107166	0.061433	-0.030455	0.050842	-0.035140	0.026870	-0.061381	0.069884	
Region_R1013	-0.005817	-0.018096	-0.195930	0.041449	-0.005216	-0.212078	0.027915	-0.033843	0.110759	-0.082202	
age	0.031032	-0.005457	0.304395	0.133771	-0.008070	0.049260	0.460558	0.192273	-0.381084	-0.101073	
Gender	0.025301	0.014332	0.060156	-0.006927	0.059716	0.015239	-0.023890	-0.001778	0.004141	-0.013983	

In [74]:
```
plt.figure(figsize=(15,5))
sns.heatmap(data.corr(),cmap='Accent',annot=True,linewidth=.5)
plt.show()
```



In [75]:
```python
from scipy.stats import spearmanr
# Extract features and target variable
features = data.drop(['charges','year','month','date','Customer ID','name','birthdate'], axis=1)
target = data['charges']

# Calculate Spearman rank correlation coefficients
corr_coeffs, p_values = [], []
for col in features.columns:
    corr, p = spearmanr(features[col], target)
    corr_coeffs.append(corr)
    p_values.append(p)

# Create a dictionary to store the correlation coefficients for each feature
corr_dict = dict(zip(features.columns, corr_coeffs))

# Sort the dictionary in descending order by correlation coefficient
corr_dict_sorted = {k: v for k, v in sorted(corr_dict.items(), key=lambda item: item[1], reverse=True)}

# Print the top 5 features with highest correlation coefficients
top_features = list(corr_dict_sorted.keys())[:5]
print('Top 5 features with highest correlation coefficients:')
print(top_features)
```

```
Top 5 features with highest correlation coefficients:
['smoker', 'BMI', 'HBA1C', 'children', 'Heart Issues']
```

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 [76]:
```
data.dropna(inplace=True)
```

In [77]:
```
features = data.drop(['charges','year','month','date','Customer ID','name','birthdate'], axis=1)
target = data['charges']
```

In [78]:
```
features.shape
```

Out[78]:
```
(1755, 15)
```

In [79]:
```
target.shape
```

Out[79]:
```
(1755,)
```

stochastic gradient descent optimizer

In [90]:

```python
import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import KFold,GridSearchCV
from sklearn.metrics import mean_squared_error

# Split features and target
features = data.drop(['charges','year','month','date','Customer ID','name','birthdate'], axis=1)
target =   data['charges']

# Define pipeline
pipeline = make_pipeline(
    StandardScaler(),
    SGDRegressor()
)

# Define hyperparameters for grid search
hyperparameters = {
    'sgdregressor__alpha': [0.0001, 0.001, 0.01, 0.1],
    'sgdregressor__penalty': ['l1', 'l2', 'elasticnet'],
    'sgdregressor__max_iter': [1000, 5000],
    'sgdregressor__eta0': [0.01, 0.1],
}

# Perform 5-fold cross-validation with hyperparameter tuning
cv = KFold(n_splits=5, shuffle=True, random_state=42)
grid = GridSearchCV(pipeline, hyperparameters, cv=cv, scoring='neg_mean_squared_error')
grid.fit(features, target)
val_scores = []

for fold_idx, (train_idx, val_idx) in enumerate(cv.split(features, target)):
    # Split data into train and validation sets
    X_train, y_train = features.iloc[train_idx], target.iloc[train_idx]
    X_val, y_val = features.iloc[val_idx], target.iloc[val_idx]

    # Fit pipeline on train data
    pipeline.fit(X_train, y_train)

    # Evaluate pipeline on validation data
    y_pred = pipeline.predict(X_val)
```

```
    val_score = np.sqrt(mean_squared_error(y_val, y_pred))
    val_scores.append(val_score)

    # Get feature importance scores
    coef_abs = np.abs(pipeline.named_steps['sgdregressor'].coef_)
    feature_importance = coef_abs / np.sum(coef_abs)
    feature_importance = pd.DataFrame({'feature': X_train.columns, 'importance': feature_importance})
    feature_importance = feature_importance.sort_values(by='importance', ascending=False)

    # Print results for this fold
    print(f"Fold {fold_idx+1}: Validation score = {val_score:.4f}")
    print("Top 5 features by importance:")
    print(feature_importance.head(5))
    print('-' * 50)

# Print average validation score across all folds
print(f"Average validation score: {np.mean(val_scores):.4f}")

# Print best hyperparameters and score
print('Best hyperparameters:', grid.best_params_)
print('Best score:', -grid.best_score_)
print(val_scores)
```

```
Fold 1: Validation score = 4673.4996
Top 5 features by importance:
          feature  importance
9          smoker    0.513969
13            age    0.185138
3             BMI    0.149015
1   Hospital tier    0.059570
0        children    0.023825
--------------------------------------------------
Fold 2: Validation score = 4330.5874
Top 5 features by importance:
          feature  importance
9          smoker    0.526268
13            age    0.186170
3             BMI    0.155212
1   Hospital tier    0.060127
0        children    0.019116
--------------------------------------------------
Fold 3: Validation score = 4612.0240
Top 5 features by importance:
          feature  importance
9          smoker    0.505903
13            age    0.191556
3             BMI    0.148109
1   Hospital tier    0.060343
0        children    0.024271
--------------------------------------------------
Fold 4: Validation score = 4321.1729
Top 5 features by importance:
          feature  importance
9          smoker    0.509363
13            age    0.187279
3             BMI    0.146493
1   Hospital tier    0.066156
0        children    0.022413
--------------------------------------------------
Fold 5: Validation score = 4342.3336
Top 5 features by importance:
          feature  importance
9          smoker    0.515887
13            age    0.191138
```

```
3             BMI    0.148833
1   Hospital tier    0.064143
11   Region_R1012    0.014493
--------------------------------------------------
Average validation score: 4455.9235
Best hyperparameters: {'sgdregressor__alpha': 0.1, 'sgdregressor__eta0': 0.01, 'sgdregressor__max_iter': 1000, 'sgdr
egressor__penalty': 'l1'}
Best score: 19793264.344503574
[4673.499618413974, 4330.587422843664, 4612.023999135048, 4321.172947742307, 4342.333629218638]
```

In [91]:

```
#Creating SGDREGRESSOR on the basis of best hyperparameter
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
x_train,x_test,y_train,y_test=train_test_split(features,target,test_size=0.2)
pipeline_sgd = make_pipeline(
    StandardScaler(),
    PCA(0.98),
    SGDRegressor(penalty='l2',alpha=0.0001,eta0=0.01, max_iter=1000)
)
pipeline_sgd.fit(x_train,y_train)
pred_sgdr=pipeline_sgd.predict(x_test)
```

In [92]:

```
#evaluation
print("r2_score",r2_score(y_test,pred_sgdr))
print("mean_squared_error",np.sqrt(mean_squared_error(y_test, pred_sgdr)))
```

```
r2_score 0.8797698093432349
mean_squared_error 4208.1664875324295
```

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 [93]:
```
from sklearn.ensemble import RandomForestRegressor

x_train,x_test,y_train,y_test=train_test_split(features,target,test_size=0.2) #splitting the data

# Define models
rf = RandomForestRegressor()

# Perform cross-validation and print results
cv = KFold(n_splits=5, shuffle=True, random_state=42)
scores_rf = cross_val_score(rf, features, target, cv=cv, scoring='r2').mean()

print('Random Forest:')
print('CV scores:', scores_rf)

#trying identify best hyperparameter

param_grid = {
    'n_estimators':[100, 200, 300],
    'max_depth' : [5, 10, 15],
    'min_samples_split' : [2, 5, 10],
    'min_samples_leaf' : [1, 2, 4]
}
grid = GridSearchCV(rf, param_grid = param_grid, cv=5)
grid.fit(X_train, y_train)
print(grid.best_params_)
```

```
Random Forest:
CV scores: 0.929479617578934
{'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100}
```

In [94]:
```
#building the pipeline by using best hyperparameter
pipeline_rf = make_pipeline(
    StandardScaler(),
    RandomForestRegressor(max_depth= 5, min_samples_leaf=2, min_samples_split= 2, n_estimators=200))

pipeline_rf.fit(x_train, y_train)
pred_rf=pipeline_rf.predict(x_test)

#evaluation
print("r2_score",r2_score(y_test,pred_rf))
print("mean_squared_error",np.sqrt(mean_squared_error(y_test, pred_rf)))

rf_model = pipeline_rf.named_steps['randomforestregressor']
importance_rf = rf_model.feature_importances_ #identifying important features

#Create a DataFrame to store the feature importance scores
df = pd.DataFrame({'Feature': features.columns,
                   'RandomForest Importance': importance_rf})

print('RandomForest variable importance scores:')
print(df.sort_values(by='RandomForest Importance', ascending=False))
```

```
r2_score 0.9251827115546893
mean_squared_error 3418.374669239081
RandomForest variable importance scores:
                   Feature  RandomForest Importance
9                   smoker                 0.792806
3                      BMI                 0.094435
13                     age                 0.082801
1            Hospital tier                 0.016180
12             Region_R1013                 0.003956
0                 children                 0.003803
10             Region_R1011                 0.003504
4                    HBA1C                 0.001127
2                City tier                 0.000437
11             Region_R1012                 0.000290
7           Cancer history                 0.000249
8   NumberOfMajorSurgeries                 0.000148
14                  Gender                 0.000147
6          Any Transplants                 0.000061
5             Heart Issues                 0.000057
```

Extreme gradient boosting

In [95]:
```
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score, KFold

x_train,x_test,y_train,y_test=train_test_split(features,target,test_size=0.2)
xgb = XGBRegressor() #defining model
cv = KFold(n_splits=5, shuffle=True, random_state=42)
scores_xgb = cross_val_score(xgb, features, target, cv=cv, scoring='r2').mean()

print('XGBoost:')
print('CV scores:', scores_xgb)

#finding the best hyper parameter
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.5]
}

grid = GridSearchCV(xgb, param_grid = param_grid, cv=6)
grid.fit(X_train, y_train)
print(grid.best_params_)
```

```
XGBoost:
CV scores: 0.9207379510548351
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
```

In [96]:
```python
#initiating pipeline
pipeline_xgb = make_pipeline(
    StandardScaler(),
    XGBRegressor(learning_rate=0.1, max_depth=3, n_estimators= 50))

pipeline_xgb.fit(x_train, y_train)
pred_xgb=pipeline_xgb.predict(x_test)

#evaluation
print("r2_score",r2_score(y_test,pred_xgb))
print("mean_squared_error",np.sqrt(mean_squared_error(y_test, pred_xgb)))

importance_xgb = pipeline_xgb['xgbregressor'].feature_importances_ #identifying important features

#Create a DataFrame to store the feature importance scores

df = pd.DataFrame({'Feature': features.columns,
                    'XGBoost Importance': importance_xgb})

print('XGBoost variable importance scores:')
print(df.sort_values(by='XGBoost Importance', ascending=False))
```

```
r2_score 0.928439115771055
mean_squared_error 3464.3783980938915
XGBoost variable importance scores:
                   Feature  XGBoost Importance
9                   smoker            0.875061
3                      BMI            0.040137
13                     age            0.031186
1            Hospital tier            0.019082
10             Region_R1011            0.008358
12             Region_R1013            0.006350
11             Region_R1012            0.005735
0                 children            0.004963
14                  Gender            0.003513
4                    HBA1C            0.003220
5             Heart Issues            0.002186
2                City tier            0.000209
6          Any Transplants            0.000000
7           Cancer history            0.000000
8   NumberOfMajorSurgeries            0.000000
```

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.

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

In [97]:

```
#calculating bmi and age
import datetime

current_year = datetime.date.today().year
birth_year = 1988
age = current_year - birth_year
print("Age:", age)

height = 170 / 100 # Convert cm to m
weight = 85
bmi = weight / height ** 2
print("BMI:", bmi)
```

```
Age: 35
BMI: 29.411764705882355
```

In [98]:

```python
# Define input data for Ms. Jayna
data = pd.DataFrame({
    'children': [2],
    'Hospital tier':[1],
    'City tier': [1],
    'BMI': [29.4],
    'HBA1C': [5.8],
    'Heart Issues':[0],
    'Any Transplants': [0],
    'Cancer history': [1],
    'NumberOfMajorSurgeries': [0],
    'smoker': [1],
    'Region_R1011':[1],
    'Region_R1012':[0],
    'Region_R1013':[0],
    'age':[35],
    'Gender': [1]
})

mean_cost=[]
# Predict hospital charges using the three models
rf_pred = pipeline_rf.predict(data)
xgb_pred =pipeline_xgb.predict(data)
SGD = pipeline.predict(data)

mean_cost.append(rf_pred[0])
mean_cost.append(xgb_pred[0])
mean_cost.append(SGD[0])


# Print the predicted charges for each model
print('Random Forest predicted charges:', rf_pred[0])
print('XGBoost predicted charges:', xgb_pred[0])
print('Sgd predicted charges',SGD[0])
print('Avearge cost for Ms.Jayna hospitalization cost will be',mean_cost[0]+mean_cost[1]+mean_cost[2]/3)
```

```
Random Forest predicted charges: 28649.759709730715
XGBoost predicted charges: 25975.48
Sgd predicted charges 30458.13747319075
Avearge cost for Ms.Jayna hospitalization cost will be 64777.9526695443
```

Project Task: Week 2

SQL

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.
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
3. Find the average hospitalization cost for each hospital tier and each city level
4. Determine the number of people who have had major surgery with a history of cancer
5. Determine the number of tier-1 hospitals in each state

```
/* 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`;
```

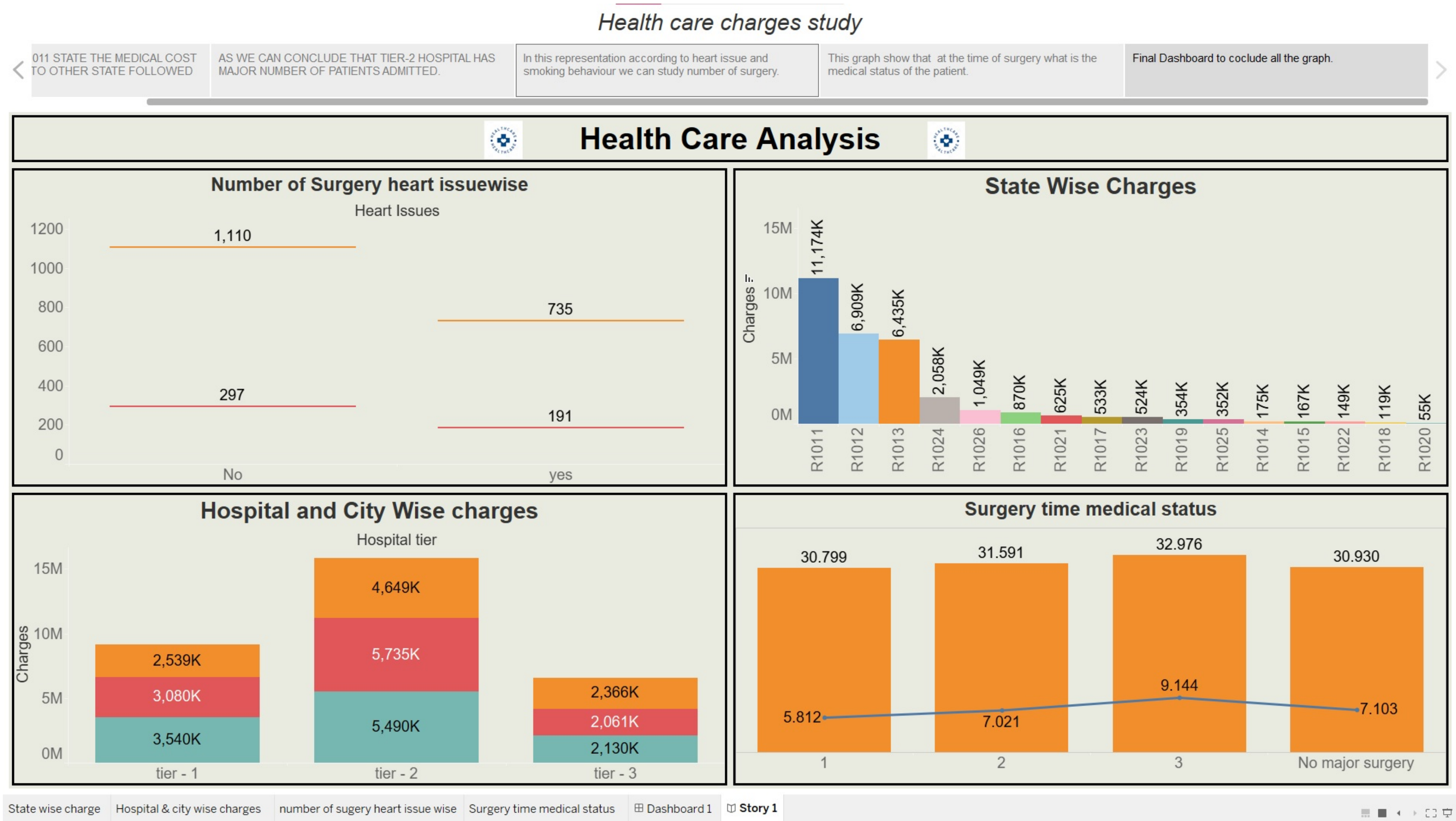
Project Task: Week 2

Tableau

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

In [101]:
```
from IPython import display
display.Image("C:\Programming\Data Science Job Readiness\Capstone Project 1\Dashboard.jpg")
```

Out[101]:



In [ ]:
```
```