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 [4]:
```
# Now importing the dataset for the further operation.
customer_details = pd.read_csv("Hospitalisation details.csv")
medical_details = pd.read_csv("Medical Examinations.csv")
customer_name = pd.read_excel("Names.xlsx")
```

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

Out[5]:
```
(2343, 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 [9]:
```
customer_name.head()
```

Out[9]:

	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 [10]:
```
customer_name.shape
```

Out[10]:
```
(2335, 2)
```

Project Task: Week 1

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

In [11]:
```python
# Now combining the data so that all information could be examine in once go throu
customer_df1 = pd.merge(customer_name, customer_details, on = "Customer ID")
customer_df1.head()
```

Out[11]:

	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 [12]:
```python
# Now lets combine the last data set and Complete the all information.
final_df = pd.merge(customer_df1, medical_details, on = "Customer ID")
final_df.head()
```

Out[12]:

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

In [13]:
```python
final_df.shape
```

Out[13]:
```
(2335, 17)
```

2. Check for missing values in the dataset

In [14]:
```python
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 [15]:
```python
final_df.dtypes.value_counts()
```

Out[15]:
```
object     12
float64     3
int64       2
dtype: int64
```

In [16]:
```python
# Missing values in the data set.
final_df.isnull().sum()
```

Out[16]:
```
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
```

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

In [17]:
```
trivial_value = final_df[final_df.eq("?").any(1)]
trivial_value
```

Out[17]:

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

In [18]:
```
trivial_value.shape
```

Out[18]: (10, 17)

In [19]:
```
# Percentage of row that have the trivial values
round(trivial_value.shape[0]/final_df.shape[0]*100, 2)
```

Out[19]: 0.43

In [20]:
```
# 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 [21]:
```
final_df.shape
```

Out[21]: (2325, 17)

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

In [22]:
```
# Handling with nominal categorical variable.
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 usi
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	HB
0	Id1	Hawks, Ms. Kelly	1968	Oct	12	0	63770.43	tier - 1	tier - 3	R1013	47.410	
1	Id2	Lehner, Mr. Matthew D	1977	Jun	8	0	62592.87	tier - 2	tier - 3	R1013	30.360	
3	Id4	Osborne, Ms. Kelsey	1991	Jun	6	1	58571.07	tier - 1	tier - 3	R1024	38.095	
4	Id5	Kadala, Ms. Kristyn	1989	Jun	19	0	55135.40	tier - 1	tier - 2	R1012	35.530	
5	Id6	Baker, Mr. Russell B.	1962	Aug	4	0	52590.83	tier - 1	tier - 3	R1011	32.800	

In [32]:

```
# Handling ordinal categorical variable.
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	HB
0	Id1	Hawks, Ms. Kelly	1968	Oct	12	0	63770.43	1	3	R1013	47.410	
1	Id2	Lehner, Mr. Matthew D	1977	Jun	8	0	62592.87	2	3	R1013	30.360	
3	Id4	Osborne, Ms. Kelsey	1991	Jun	6	1	58571.07	1	3	R1024	38.095	
4	Id5	Kadala, Ms. Kristyn	1989	Jun	19	0	55135.40	1	2	R1012	35.530	
5	Id6	Baker, Mr. Russell B.	1962	Aug	4	0	52590.83	1	3	R1011	32.800	

5. 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
```

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_F
0	0	0	1	0	0	
1	0	0	1	0	0	
3	0	0	0	0	0	
4	0	1	0	0	0	
5	1	0	0	0	0	
...	...	...	...	...	...	
2330	0	0	1	0	0	
2331	0	0	1	0	0	
2332	0	0	1	0	0	
2333	0	0	1	0	0	
2334	0	0	1	0	0	

2325 rows × 16 columns

In [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	H Is
0	Id1	Hawks, Ms. Kelly	1968	Oct	12	0	63770.43	1	3	47.410	7.47	
1	Id2	Lehner, Mr. Matthew D	1977	Jun	8	0	62592.87	2	3	30.360	5.77	
3	Id4	Osborne, Ms. Kelsey	1991	Jun	6	1	58571.07	1	3	38.095	6.05	
4	Id5	Kadala, Ms. Kristyn	1989	Jun	19	0	55135.40	1	2	35.530	5.45	
5	Id6	Baker, Mr. Russell B.	1962	Aug	4	0	52590.83	1	3	32.800	6.59	

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

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

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

In [45]:
```
final_df['NumberOfMajorSurgeries'] = final_df['NumberOfMajorSurgeries'].replace('N
```

In [46]:
```
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 [47]:
```
final_df["year"] = pd.to_datetime(final_df["year"], format='%Y').dt.year
final_df["year"]
```

Out[47]:
```
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 [48]:
```
final_df["month"] = pd.to_datetime(final_df["month"], format='%b').dt.month
final_df["month"]
```

Out[48]:
```
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 [49]:
```
final_df['DateInt'] = final_df["year"].astype(str) + final_df["month"].astype(str)
```

In [51]:
```
final_df['DOB'] = pd.to_datetime(final_df.DateInt, format = "%Y%m%d")
```

In [52]:
```
final_df.drop(["DateInt"], inplace = True, axis=1)
```

In [53]:
```
final_df.head()
```

Out[53]:

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

In [54]:
```python
import datetime as dt
current_date = dt.datetime.now()
```

In [55]:
```python
final_df['age'] = (((current_date - final_df.DOB).dt.days)/365).astype(int)
```

In [56]:
```python
final_df.head()
```

Out[56]:

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

5 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 [57]:
```
def gender(val):
    if "Ms." in val:
        return 0
    else:
        return 1
```

In [58]:
```
final_df["gender"] = final_df["name"].map(gender)
```

In [59]:
```
final_df.head()
```

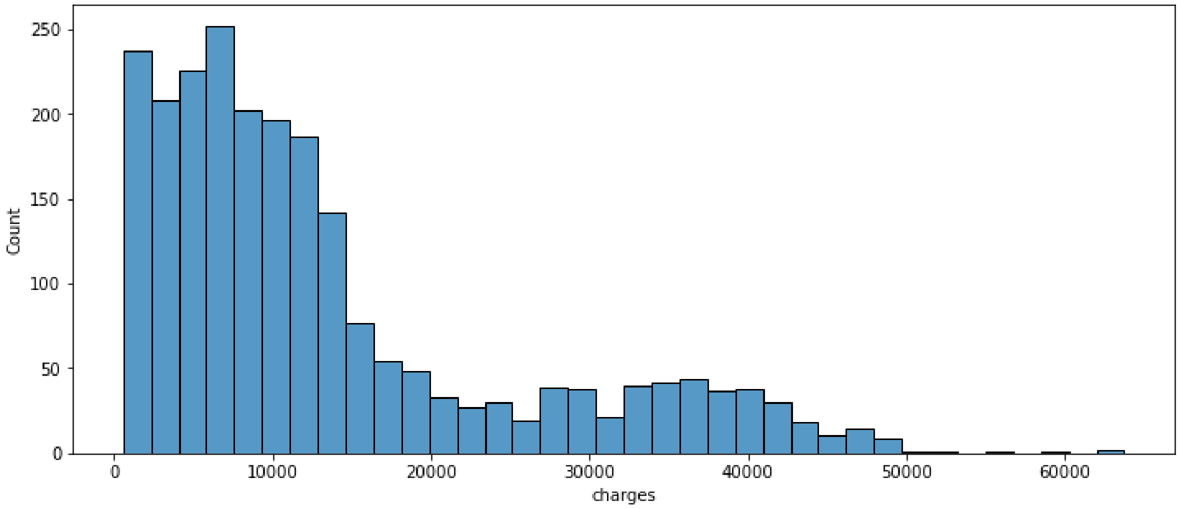
Out[59]:

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

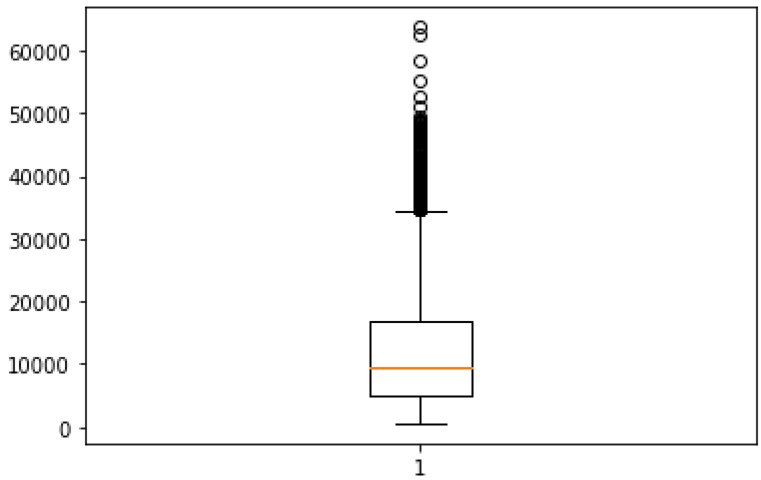
5 rows × 22 columns

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

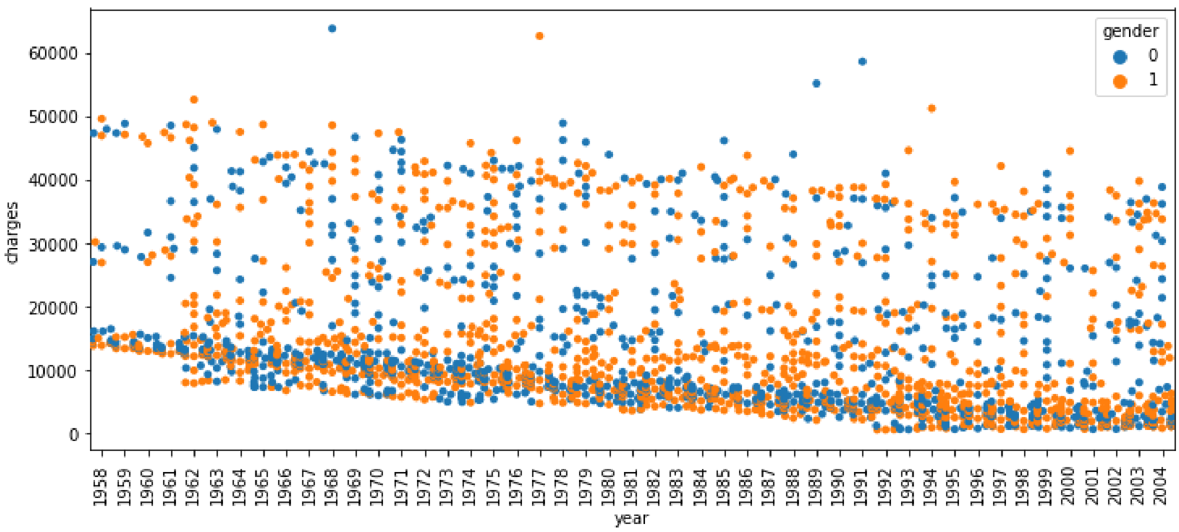
In [61]:
```
# Histogram for the cost distribution.
plt.figure(figsize=(12,5))
sns.histplot(final_df['charges'])
plt.show()
```



In [62]:
```
# Visualize the cost distribution of the hospitals by box or whisker plot.
plt.boxplot(final_df['charges'])
plt.show()
```

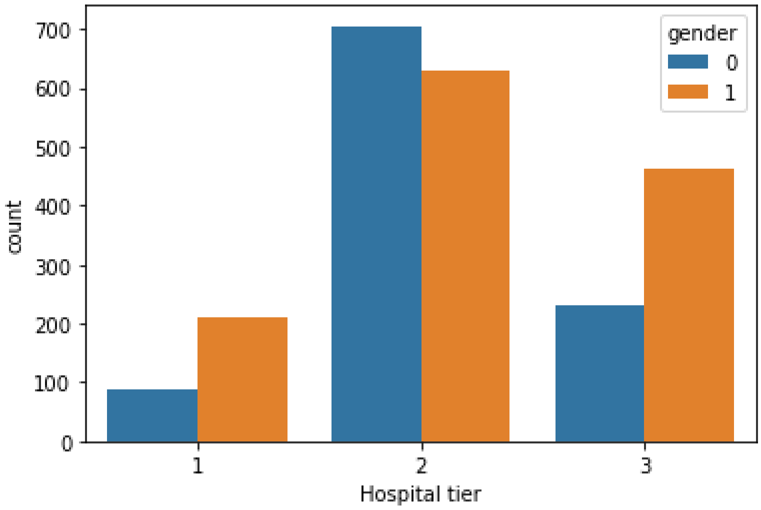


In [63]:
```
# Visualize the cost distribution of the hospitals by swarm plot.
plt.figure(figsize=(12,5))
sns.swarmplot(x='year', y='charges', hue="gender", data=final_df)
plt.xticks(rotation=90)
plt.show()
```



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

In [64]:
```
sns.countplot(data = final_df, x='Hospital tier', hue= 'gender')
plt.show()
```



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

In [65]:
```
print("median cost of tier 1 hospitals:", final_df[final_df["Hospital tier"]==1].cl
print("median cost of tier 2 hospitals:", final_df[final_df["Hospital tier"]==2].cl
print("median cost of tier 3 hospitals:", final_df[final_df["Hospital tier"]==3].cl
```

```
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 [66]:
```
df = pd.DataFrame(dict(r=[32097.43, 7168.76, 10676.83],theta=['tier 1 hospital','t
```

In [67]:
```
df
```

Out[67]:

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

In [68]:
```
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 [69]:
```
# Frequency table for count of the people according to the tier of city and hospit
final_df["Hospital tier"].value_counts()
```

Out[69]:
```
2    1334
3     691
1     300
Name: Hospital tier, dtype: int64
```

In [70]:
```
city_freq = final_df["City tier"].value_counts().rename_axis('City&hospital_tier')
```

In [71]:
```
hospital_freq = final_df["Hospital tier"].value_counts().rename_axis('City&hospita
```

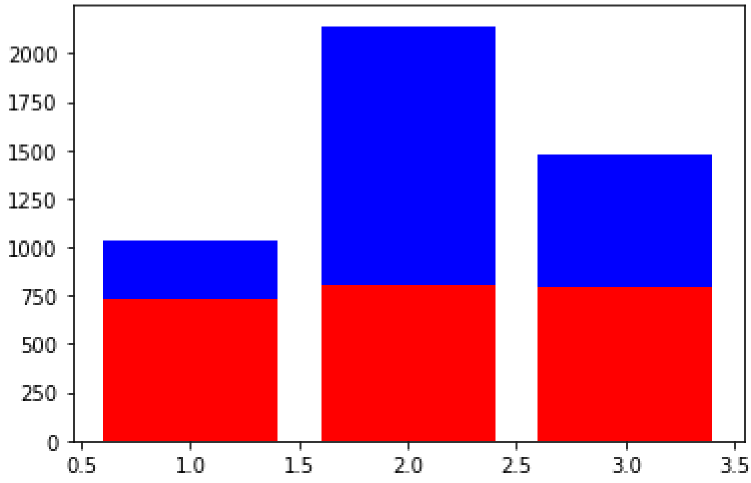
In [72]:
```
df = pd.merge(city_freq, hospital_freq, on = 'City&hospital_tier')
```

In [73]:
```
df
```

Out[73]:

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

In [74]:
```
plt.bar(df["City&hospital_tier"], df["city_counts"], color='r')
plt.bar(df["City&hospital_tier"], df["hospital_counts"], bottom=df["city_counts"],
plt.show()
```



In [75]:
```
from scipy.stats import ttest_1samp
```

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

In [76]:
```
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 [77]:
```
# b. The average hospitalization costs for the three types of cities are not signi
print("median cost of tier 1 city:", final_df[final_df["City tier"]==1].charges.me
print("median cost of tier 2 city:", final_df[final_df["City tier"]==2].charges.me
print("median cost of tier 3 city:", final_df[final_df["City tier"]==3].charges.me
```

```
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 [78]:
```
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 [79]:
```
# c. The average hospitalization cost for smokers is not significantly different fr
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 [80]:
```
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
```

In [81]:
```
# d. Smoking and heart issues are independent
from scipy.stats import chi2_contingency
table = [[final_df["Heart Issues"].value_counts()],[final_df["smoker"].value_count
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 [82]:
```
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 [83]:
```
# In the data frame same of the column are not usable to model building so lets fi
#then indentify the highly corelated predictor.
final_df.drop(["Customer ID",'name', 'year', 'month', 'date', 'DOB'], inplace=True
final_df.shape
```

Out[83]:
```
(2325, 16)
```

In [84]:
```
final_df.head()
```

Out[84]:

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

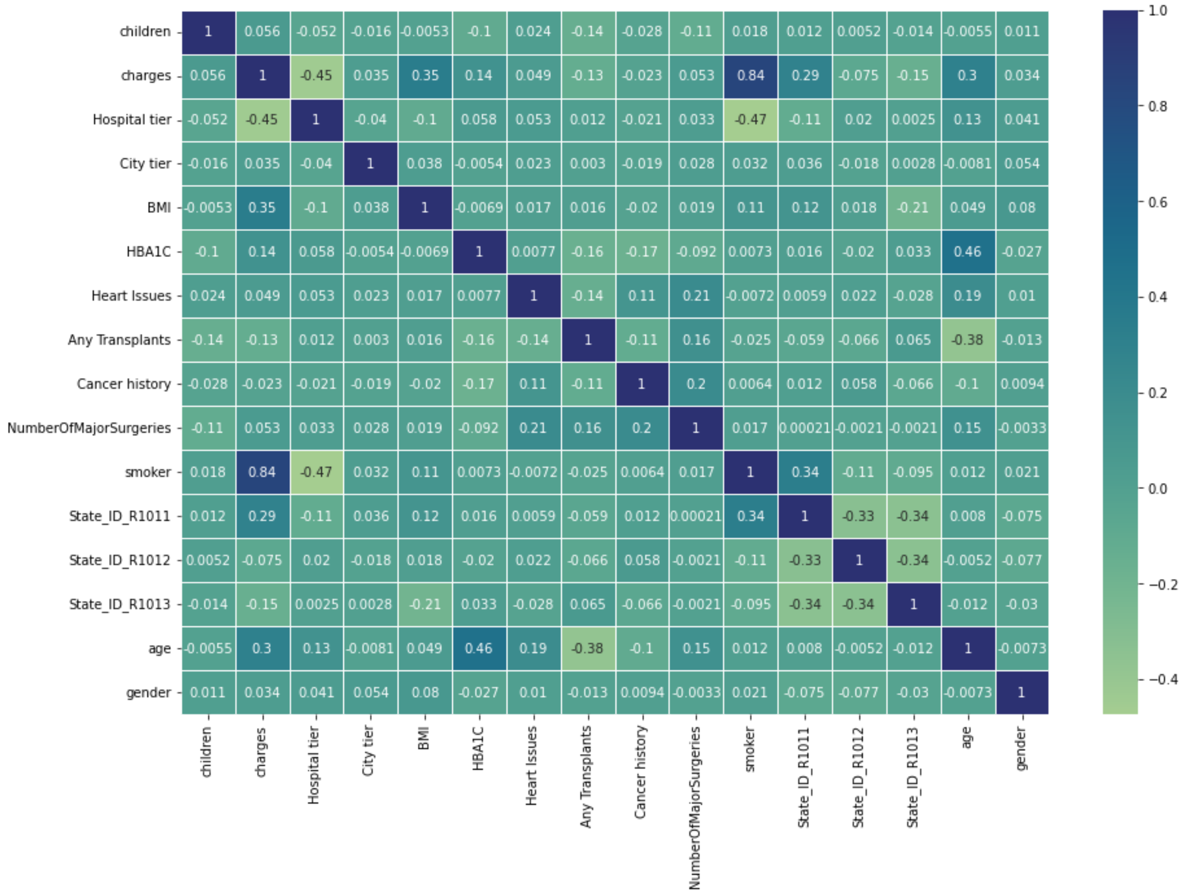
In [85]:
```
corr = final_df.corr()
corr
```

Out[85]:

	children	charges	Hospital tier	City tier	BMI	HBA1C	Hea Issu
children	1.000000	0.055901	-0.052438	-0.015760	-0.005339	-0.101379	0.02398
charges	0.055901	1.000000	-0.446687	0.035300	0.346730	0.139697	0.04929
Hospital tier	-0.052438	-0.446687	1.000000	-0.039755	-0.104771	0.057855	0.05337
City tier	-0.015760	0.035300	-0.039755	1.000000	0.038123	-0.005404	0.02315
BMI	-0.005339	0.346730	-0.104771	0.038123	1.000000	-0.006920	0.01712
HBA1C	-0.101379	0.139697	0.057855	-0.005404	-0.006920	1.000000	0.00769
Heart Issues	0.023984	0.049299	0.053376	0.023152	0.017129	0.007699	1.00000
Any Transplants	-0.142040	-0.127028	0.011729	0.002970	0.015893	-0.159855	-0.14026
Cancer history	-0.027880	-0.022522	-0.021429	-0.018639	-0.020235	-0.170921	0.11119
NumberOfMajorSurgeries	-0.113161	0.053308	0.033230	0.027937	0.018851	-0.091594	0.20614
smoker	0.017713	0.838462	-0.474077	0.032034	0.107126	0.007257	-0.00715
State_ID_R1011	0.011666	0.286956	-0.114685	0.036049	0.115671	0.015525	0.00585
State_ID_R1012	0.005247	-0.074636	0.020272	-0.018253	0.017939	-0.019513	0.02177
State_ID_R1013	-0.013834	-0.150634	0.002455	0.002766	-0.208744	0.033453	-0.02796
age	-0.005457	0.304395	0.133771	-0.008070	0.049260	0.460558	0.19227
gender	0.011205	0.034069	0.041261	0.054073	0.079930	-0.027339	0.01027

In [86]:

```python
plt.figure(figsize=(15,10))
sns.heatmap(corr, annot=True, linewidth=.5, cmap="crest")
plt.show()
```



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 [87]:
```
# Lets first seperate the input and output data.
x = final_df.drop(["charges"], axis=1)
y = final_df[['charges']]
```

In [88]:
```
# Lets split the data set into the training and testing data.
from sklearn.model_selection import train_test_split
```

In [89]:
```
x_train, x_test, y_train, y_test  = train_test_split(x,y, test_size=.20, random_st
```

In [90]:
```
# Now standardize the data.
from sklearn.preprocessing import StandardScaler
```

In [91]:
```
sc = StandardScaler()
```

In [92]:
```
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

In [93]:
```
from sklearn.linear_model import SGDRegressor
```

In [94]:
```
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)
```

Out[94]:
```
Fitting 5 folds for each of 84 candidates, totalling 420 fits
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 [95]:
```
model_cv.best_params_
```

Out[95]:
```
{'alpha': 50, 'penalty': 'l1'}
```

In [96]:
```
sgd = SGDRegressor(alpha= 100, penalty= 'l1')
```

In [97]:
```
sgd.fit(x_train, y_train)
```

Out[97]:
```
SGDRegressor(alpha=100, penalty='l1')
```

In [98]:
```
sgd.score(x_test, y_test)
```

Out[98]:
```
0.8602495677726669
```

In [99]:
```
y_pred = sgd.predict(x_test)
```

In [100…
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [102…
```
sgd_mae = mean_absolute_error(y_test, y_pred)
sgd_mse = mean_squared_error(y_test, y_pred)
```

```
sgd_rmse = sgd_mse*(1/2.0)
```

In [103…

```
print("MAE:", sgd_mae)
print("MSE:", sgd_mse)
print("RMSE:", sgd_rmse)
```

```
MAE: 3134.524710884731
MSE: 23506849.75240736
RMSE: 11753424.87620368
```

In [104…

```
# d. Determine the variable importance scores, and identify the redundant variable:
importance = sgd.coef_
```

In [105…

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

Out[105]:

	Feature_imp
children	342.564040
Hospital tier	-1179.487983
City tier	0.000000
BMI	2718.546547
HBA1C	60.773832
Heart Issues	0.000000
Any Transplants	0.000000
Cancer history	19.557174
NumberOfMajorSurgeries	0.000000
smoker	8793.400707
State_ID_R1011	-189.577264
State_ID_R1012	0.000000
State_ID_R1013	-350.073301
age	3353.637397
gender	0.000000

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

random forest

In [106…

```
from sklearn.ensemble import RandomForestRegressor
```

In [107…

```
# 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[107]:
```
RandomForestRegressor(n_estimators=1000, random_state=42)
```

In [108…
```
score = rf.score(x_test,y_test)
score
```

Out[108]:
```
0.9222696338245824
```

In [109…
```
y_pred = rf.predict(x_test)
```

In [110…
```
rf_mae = mean_absolute_error(y_test, y_pred)
```

In [111…
```
from sklearn.ensemble import GradientBoostingRegressor
```

In [112…
```
# 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[112]:
```
GradientBoostingRegressor(n_estimators=1000, random_state=42)
```

In [113…
```
score = gbr.score(x_test,y_test)
score
```

Out[113]:
```
0.9042734212625119
```

In [114…
```
y_pred = gbr.predict(x_test)
```

In [115…
```
gbr_mae = mean_absolute_error(y_test, y_pred)
gbr_mae
```

Out[115]:
```
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 [116…
```
# Calculate the age of the person.
date = str(19881228)
date1 = pd.to_datetime(date, format = "%Y%m%d")
```

In [117…
```
current_date = dt.datetime.now()
current_date
```

Out[117]:
```
datetime.datetime(2023, 5, 7, 23, 29, 41, 979602)
```

In [119…
```
age = (current_date - date)
age
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [119], in <cell line: 1>()
----> 1 age = (current_date - date)
      2 age

TypeError: unsupported operand type(s) for -: 'datetime.datetime' and 'str'
```

In [120…
```
age = int(12421/365)
age
```
Out[120]:
```
34
```

In [121…
```
# 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[121]:
```
29.41
```

In [122…
```
# Now lets gen
list = [[2,1,1,24.41,5.8,0,0,0,0,1,1,0,0,34,0]]
```

In [123…
```
df = pd.DataFrame(list, columns = ['children', 'Hospital tier', 'City tier', 'BMI'
                                   'Cancer history','NumberOfMajorSurgeries', 'smoker',
                                   'State_ID_R1013', 'age', 'gender'] )
df
```
Out[123]:

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

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

In [124…
```
Hospital_cost = []
# Now lets predict the hospitalization cost through SGDRegressor
Cost1 = sgd.predict(df)
Hospital_cost.append(Cost1)
# Now lets predict the hospitalization cost through Random Forest
Cost2 = rf.predict(df)
Hospital_cost.append(Cost2)
# Now lets predict the hospitalization cost throug Extreme gradient Booster
Cost3 = gbr.predict(df)
Hospital_cost.append(Cost3)
avg_cost = np.mean(Hospital_cost)
avg_cost
```
Out[124]:
```
103999.74746407126
```

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

```
In [ ]:
```