Healthcare Insurance Analysis

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.

Install Important Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

>> DATA SCIENCE

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

```
df1=pd.read_csv('Hospitalisation details.csv')
df2=pd.read_csv('Medical Examinations.csv')
df3=pd.read_csv('Names.csv')
df=pd.merge(df1, df2,on='Customer ID')
df=pd.merge(df, df3, on='Customer ID')
df.head()
```

```
  Customer ID  year month  date  children  charges Hospital tier City 
tier  \
0      Id2335  1992   Jul     9         0   563.84       tier - 2  tier 
- 3
1      Id2334  1992   Nov    30         0   570.62       tier - 2  tier 
- 1
2      Id2333  1993   Jun    30         0   600.00       tier - 2  tier 
- 1
3      Id2332  1992   Sep    13         0   604.54       tier - 3  tier
```

```
- 3
4      Id2331  1998   Jul    27         0   637.26      tier - 3  tier
- 3

  State ID    BMI  HBA1C Heart Issues Any Transplants Cancer
history  \
0    R1013  17.58   4.51           No              No             No

1    R1013  17.60   4.39           No              No             No

2    R1013  16.47   6.35           No              No            Yes

3    R1013  17.70   6.28           No              No             No

4    R1013  22.34   5.57           No              No             No


  NumberOfMajorSurgeries smoker                              name
0                      1     No                German, Mr.  Aaron K
1                      1     No              Rosendahl, Mr.  Evan P
2                      1     No                  Albano, Ms.  Julie
3                      1     No  Riveros Gonzalez, Mr.  Juan D. Sr.
4                      1     No               Brietzke, Mr.  Jordan
```

2. Check for missing values in the dataset

```
df.info()

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

```
dtypes: float64(3), int64(2), object(12)
memory usage: 310.2+ KB
```

Upon initial inspection, it looks like there are no null values. But the dataset contains '?' at some places. For Example -

```
df.loc[df['State ID']=='?']
```

```
     Customer ID  year month  date  children   charges Hospital
tier  \
542        Id1793  1995   Dec     1         3   4827.90      tier - 1

2165        Id170  2000   Sep     5         1  37165.16      tier - 1


     City tier State ID     BMI  HBA1C Heart Issues Any Transplants  \
542   tier - 2        ?  18.905   4.91          yes              No
2165  tier - 3        ?  37.620   6.32          yes             yes

     Cancer history NumberOfMajorSurgeries smoker
name
542              No                      1     No  Capriolo, Mr.
Michael
2165             No                      2    yes      Torphy, Mr.
Bobby
```

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

```
trivial_counts=df.eq('?').sum().sum()
trivial_percentage=(trivial_counts/len(df))*100
print('Trivial Counts:',trivial_counts)
print('Trivial Percentgae: %.2f' % trivial_percentage + '%')
```

```
Trivial Counts: 11
Trivial Percentgae: 0.47%
```

Detailed columnwise Trivial counts

```
trivial_counts=df.eq('?').sum()
trivial_percentage=(trivial_counts/len(df))*100        ##
Here,len(df)==2335

# Create a table with trivial counts and percentages
table = [['Features', 'Trivial Counts', 'Trivial Percentage (%)']]
for feat, count in trivial_counts.items():
    table.append([feat, count, trivial_percentage[feat]])

# Print the table using tabulate
```

```python
from tabulate import tabulate
print(tabulate(table, headers='firstrow', tablefmt='psql'))
```

```
+-------------------------+-------------------
+--------------------------+
| Features                |   Trivial Counts |   Trivial Percentage (%)
|
|-------------------------+-------------------
+--------------------------|
| Customer ID             |                0 |                  0
|
| year                    |                2 |                  0.0856531
|
| month                   |                3 |                  0.12848
|
| date                    |                0 |                  0
|
| children                |                0 |                  0
|
| charges                 |                0 |                  0
|
| Hospital tier           |                1 |                  0.0428266
|
| City tier               |                1 |                  0.0428266
|
| State ID                |                2 |                  0.0856531
|
| BMI                     |                0 |                  0
|
| HBA1C                   |                0 |                  0
|
| Heart Issues            |                0 |                  0
|
| Any Transplants         |                0 |                  0
|
| Cancer history          |                0 |                  0
|
| NumberOfMajorSurgeries  |                0 |                  0
|
| smoker                  |                2 |                  0.0856531
|
| name                    |                0 |                  0
|
+-------------------------+-------------------
+--------------------------+
```

We can see that those trivial values contains around 0.1% data in each feature. So, we can eliminate them.

```
# Replace '?' with NaN values
df.replace('?',np.nan, inplace=True)

#Check Null Values
df.dropna(inplace=True)
df.isnull().sum()

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

#check Duplicate Values
df['Customer ID'].duplicated().sum()

np.int64(0)

df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 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
```

So, total 10 data have been dropped, which is 0.43% of the total data.

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

```
df.head()

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

  State ID    BMI  HBA1C Heart Issues Any Transplants Cancer
history  \
0    R1013  17.58   4.51           No              No             No

1    R1013  17.60   4.39           No              No             No

2    R1013  16.47   6.35           No              No            Yes

3    R1013  17.70   6.28           No              No             No

4    R1013  22.34   5.57           No              No             No


  NumberOfMajorSurgeries smoker                                name
0                      1     No                German, Mr.  Aaron K
1                      1     No              Rosendahl, Mr.  Evan P
2                      1     No                  Albano, Ms.  Julie
3                      1     No  Riveros Gonzalez, Mr.  Juan D. Sr.
4                      1     No               Brietzke, Mr.  Jordan
```

We observe that 'Hospital tier' and 'City tier' columns have alphanumeric values. We aim to convert them into numerical values, where 'tier-1' becomes '1', 'tier-2' becomes '2', etc., to enable more efficient data handling.

```
df['Hospital_tier_num'] = df['Hospital tier'].str.extract('tier - (\
d+)').astype(int)
df['City_tier_num'] = df['City tier'].str.extract('tier - (\
d+)').astype(int)
df.head()
```

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

  State ID    BMI  HBA1C Heart Issues Any Transplants Cancer
history  \
0    R1013  17.58   4.51           No              No             No

1    R1013  17.60   4.39           No              No             No

2    R1013  16.47   6.35           No              No            Yes

3    R1013  17.70   6.28           No              No             No

4    R1013  22.34   5.57           No              No             No


  NumberOfMajorSurgeries smoker                             name  \
0                      1     No              German, Mr.  Aaron K
1                      1     No           Rosendahl, Mr.  Evan P
2                      1     No               Albano, Ms.  Julie
3                      1     No  Riveros Gonzalez, Mr.  Juan D. Sr.
4                      1     No             Brietzke, Mr.  Jordan

   Hospital_tier_num  City_tier_num
0                  2              3
1                  2              1
2                  2              1
3                  3              3
4                  3              3
```

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. Design a suitable strategy to create dummy variables with these restraints.

```
df['State ID'].value_counts()
```

```
State ID
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: count, dtype: int64
```

```
# Create a new column called 'stateflag' that categorizes state IDs.

df['State_Flag'] = np.where((df['State ID'] == 'R1011') | (df['State
ID'] == 'R1012') | (df['State ID'] == 'R1013'), df['State ID'],
'other')
df['State_Flag'].replace('R1011',1,inplace=True)
df['State_Flag'].replace('R1012',2,inplace=True)
df['State_Flag'].replace('R1013',3,inplace=True)
df['State_Flag'].replace('other',0,inplace=True)
df.head()
```

```
  Customer ID  year month  date  children  charges Hospital tier City
tier  \
0      Id2335  1992   Jul     9         0   563.84      tier - 2  tier
- 3
1      Id2334  1992   Nov    30         0   570.62      tier - 2  tier
- 1
2      Id2333  1993   Jun    30         0   600.00      tier - 2  tier
- 1
3      Id2332  1992   Sep    13         0   604.54      tier - 3  tier
- 3
```

```
4       Id2331  1998   Jul    27          0   637.26      tier - 3  tier
- 3

  State ID    BMI  HBA1C Heart Issues Any Transplants Cancer
history  \
0    R1013  17.58   4.51           No              No             No

1    R1013  17.60   4.39           No              No             No

2    R1013  16.47   6.35           No              No            Yes

3    R1013  17.70   6.28           No              No             No

4    R1013  22.34   5.57           No              No             No


  NumberOfMajorSurgeries smoker                               name  \
0                      1     No                German, Mr.  Aaron K
1                      1     No              Rosendahl, Mr.  Evan P
2                      1     No                  Albano, Ms.  Julie
3                      1     No  Riveros Gonzalez, Mr.  Juan D. Sr.
4                      1     No                Brietzke, Mr.  Jordan

   Hospital_tier_num  City_tier_num  State_Flag
0                  2              3           3
1                  2              1           3
2                  2              1           3
3                  3              3           3
4                  3              3           3
```

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

```
# Replace 'No major surgery' with '0'.
df['NumberOfMajorSurgeries'].replace('No major
surgery','0',inplace=True)
df['NumberOfMajorSurgeries'] =
df['NumberOfMajorSurgeries'].astype(int)
df.head()

  Customer ID  year month  date  children  charges Hospital tier City
tier  \
0       Id2335  1992   Jul     9          0   563.84      tier - 2  tier
- 3
1       Id2334  1992   Nov    30          0   570.62      tier - 2  tier
- 1
2       Id2333  1993   Jun    30          0   600.00      tier - 2  tier
- 1
3       Id2332  1992   Sep    13          0   604.54      tier - 3  tier
- 3
```

```
4      Id2331  1998   Jul    27         0   637.26      tier - 3  tier
- 3

  State ID    BMI  HBA1C Heart Issues Any Transplants Cancer
history  \
0    R1013  17.58   4.51           No              No             No

1    R1013  17.60   4.39           No              No             No

2    R1013  16.47   6.35           No              No            Yes

3    R1013  17.70   6.28           No              No             No

4    R1013  22.34   5.57           No              No             No


   NumberOfMajorSurgeries smoker
name  \
0                       1     No               German, Mr.  Aaron K

1                       1     No             Rosendahl, Mr.  Evan P

2                       1     No                 Albano, Ms.  Julie

3                       1     No  Riveros Gonzalez, Mr.  Juan D. Sr.

4                       1     No              Brietzke, Mr.  Jordan


   Hospital_tier_num  City_tier_num  State_Flag
0                  2              3           3
1                  2              1           3
2                  2              1           3
3                  3              3           3
4                  3              3           3
```

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

```
#Covert 'year' column into int datatype.
df['year']=df['year'].astype('int')

# create a dictionary to map month names to integers
month_no = {'Jun': 6, 'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov':
11, 'Dec': 12}

# convert month column to integer using the dictionary
df['month'] = df['month'].map(month_no)

# Calculate Date of Birth
```

```
import datetime
df['D.O.B']=pd.to_datetime({'year': df['year'], 'month': df['month'],
'day': df['date']})

#Age Calculation
def age_calculation(DOB):
    today = datetime.datetime.today()
    age = today.year - DOB.year - ((today.month, today.day) <
(DOB.month, DOB.day))
    return (age)

df['Age'] = df['D.O.B'].apply(age_calculation)

df.head()

  Customer ID  year  month  date  children  charges Hospital tier City
tier  \
0      Id2335  1992      7     9         0   563.84      tier - 2
tier - 3
1      Id2334  1992     11    30         0   570.62      tier - 2
tier - 1
2      Id2333  1993      6    30         0   600.00      tier - 2
tier - 1
3      Id2332  1992      9    13         0   604.54      tier - 3
tier - 3
4      Id2331  1998      7    27         0   637.26      tier - 3
tier - 3

  State ID    BMI  ...  Any Transplants Cancer history
NumberOfMajorSurgeries  \
0    R1013  17.58  ...               No             No
1
1    R1013  17.60  ...               No             No
1
2    R1013  16.47  ...               No            Yes
1
3    R1013  17.70  ...               No             No
1
4    R1013  22.34  ...               No             No
1

  smoker                                name Hospital_tier_num
City_tier_num  \
0     No               German, Mr.  Aaron K                  2
3
1     No             Rosendahl, Mr.  Evan P                  2
1
2     No                 Albano, Ms.  Julie                  2
1
3     No  Riveros Gonzalez, Mr.  Juan D. Sr.                  3
```

```
3
4     No               Brietzke, Mr.  Jordan                3
3

   State_Flag       D.O.B  Age
0           3 1992-07-09   32
1           3 1992-11-30   31
2           3 1993-06-30   31
3           3 1992-09-13   31
4           3 1998-07-27   26

[5 rows x 22 columns]
```

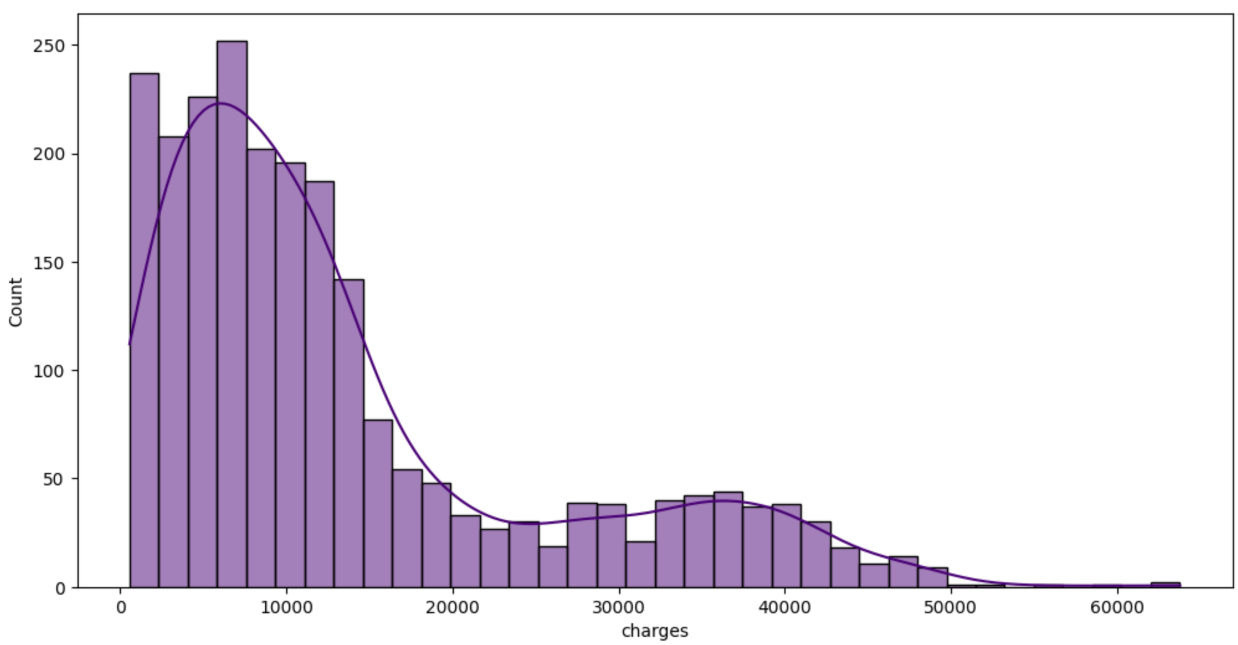
8. The gender of the patient may be an important factor in determining the cost of hospitalization. The salutations in a beneficiary's name can be used to determine their gender. Make a new field for the beneficiary's gender.

```
#Create a new column called Gender where define Male as 1 and Female
as 0.
gender = [1 if 'Mr.' in name else 0 for name in df['name']]
df['Gender']=gender
df.head()

  Customer ID  year  month  date  children  charges Hospital tier City
tier  \
0      Id2335  1992      7     9         0   563.84      tier - 2
tier - 3
1      Id2334  1992     11    30         0   570.62      tier - 2
tier - 1
2      Id2333  1993      6    30         0   600.00      tier - 2
tier - 1
3      Id2332  1992      9    13         0   604.54      tier - 3
tier - 3
4      Id2331  1998      7    27         0   637.26      tier - 3
tier - 3

  State ID    BMI  ...  Cancer history NumberOfMajorSurgeries
smoker  \
0    R1013  17.58  ...              No                      1     No

1    R1013  17.60  ...              No                      1     No

2    R1013  16.47  ...             Yes                      1     No

3    R1013  17.70  ...              No                      1     No

4    R1013  22.34  ...              No                      1     No
```

```
                               name  Hospital_tier_num City_tier_num  \
0                German, Mr.  Aaron K                  2             3  
1              Rosendahl, Mr.  Evan P                  2             1  
2                  Albano, Ms.  Julie                  2             1  
3  Riveros Gonzalez, Mr.  Juan D. Sr.                  3             3  
4               Brietzke, Mr.  Jordan                  3             3  

  State_Flag      D.O.B  Age  Gender
0          3 1992-07-09   32       1
1          3 1992-11-30   31       1
2          3 1993-06-30   31       0
3          3 1992-09-13   31       1
4          3 1998-07-27   26       1

[5 rows x 23 columns]
```

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

```
plt.figure(figsize=(12, 6))
sns.histplot(df['charges'], legend=True, kde=True,color="#490275")
plt.show()
```



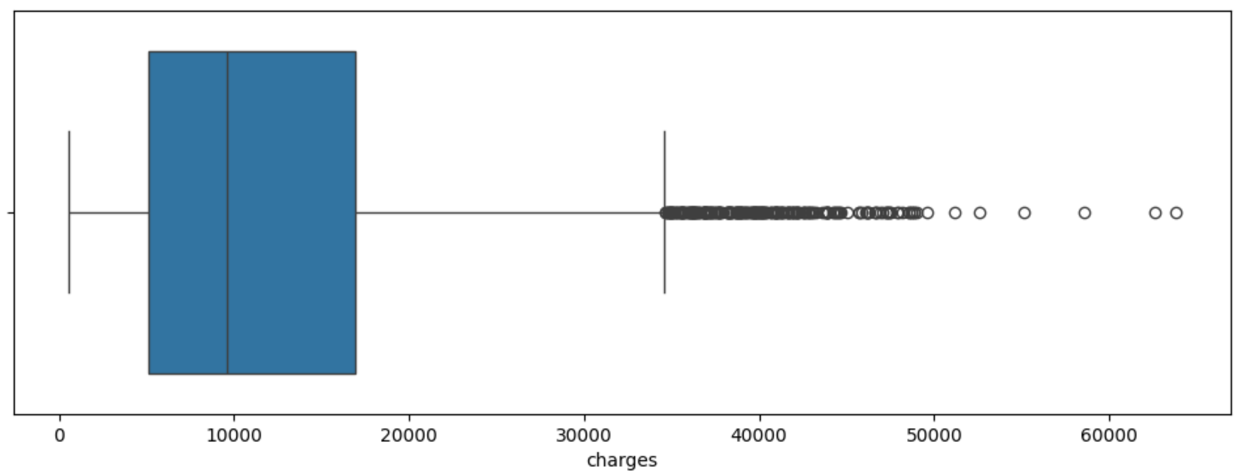
Observation:

• The distribution is right-skewed (positively skewed), with a longer tail on the right side. This indicates that while most 'charges' values are lower, there are some significantly higher values.

• The majority of 'charges' values fall below 20,000. Beyond this point, the frequency of 'charges' decreases, but there are still some occurrences up to 60,000.

• The highest peak in the histogram occurs between 5,000 and 10,000.

```
plt.figure(figsize=(12, 4))
sns.boxplot(x=df['charges'])
plt.show()
```



Observation:

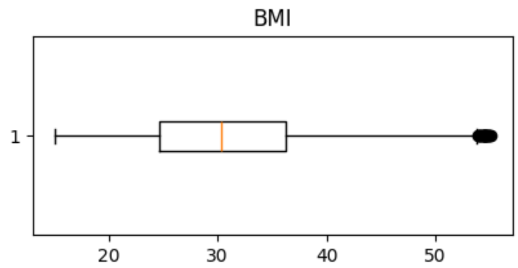
• The median is around 9,000, which indicates that half of the 'charges' values are below this point.

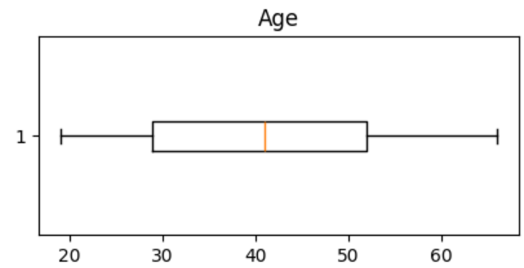
• 50% of the central data is contained in the box, ranging approximately from 5,000 to 17,000.

• The lower whisker ends near 0, while the upper whisker ends around 35,000.

• The outliers extend up to approximately 63,000.

```
#Check Outliers in other features such as Age and BMI.
cols=['BMI','Age']
fig,ax=plt.subplots(1,2,figsize=(12,2))
for i,col in enumerate(cols):
    ax[i].boxplot(x=df[col].values,vert=False)
    ax[i].set_title(col)
plt.subplots_adjust(wspace=0.5)
plt.show()
```





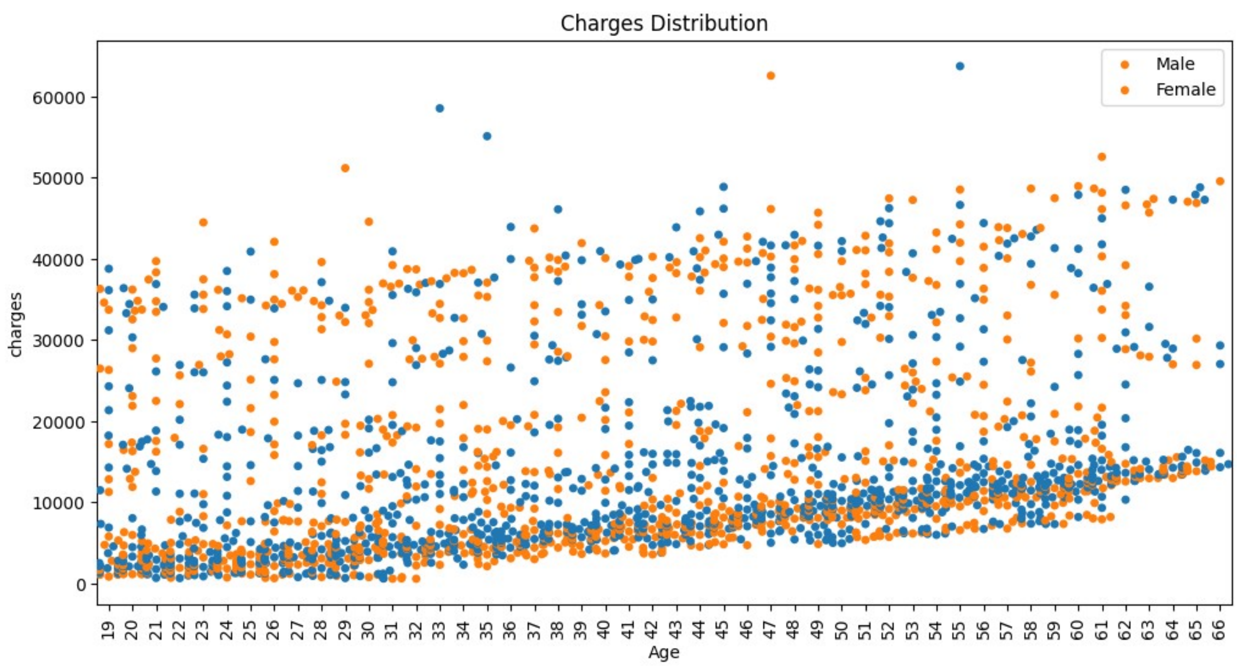
Observation:

• The median BMI is around 30 and age is around 40.

• For BMI, there are several outliers beyond the upper whisker, with values approximately exceeding 55.

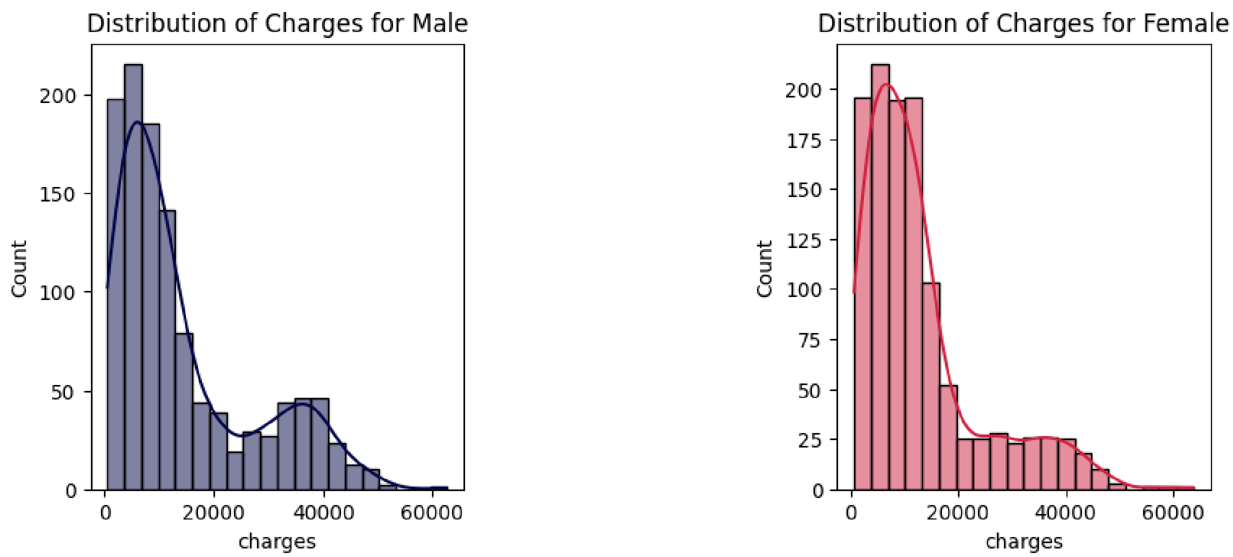
• For age, the whiskers range from around 18 to 64.

```
plt.figure(figsize=(12,6))
sns.swarmplot(data=df, y='charges', x='Age', hue='Gender')
plt.title('Charges Distribution')
plt.legend(labels=['Male','Female'])
plt.xticks(rotation=90)
plt.show()
```

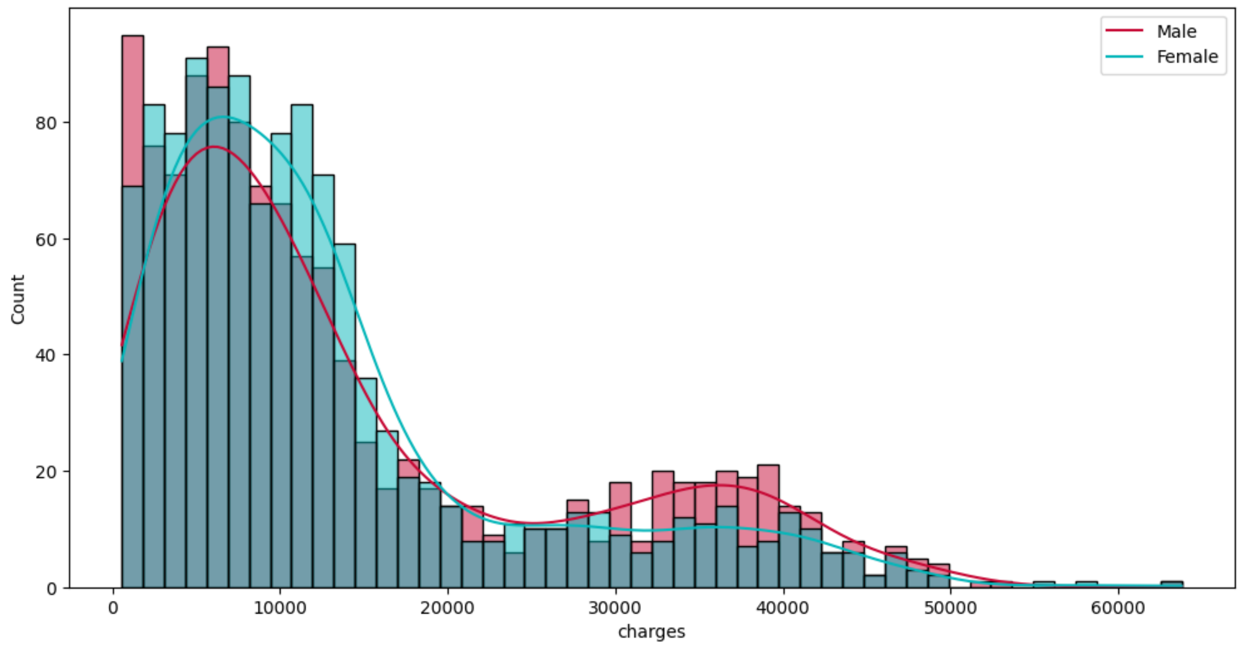


10. State how the distribution is different across gender and tiers of hospitals.

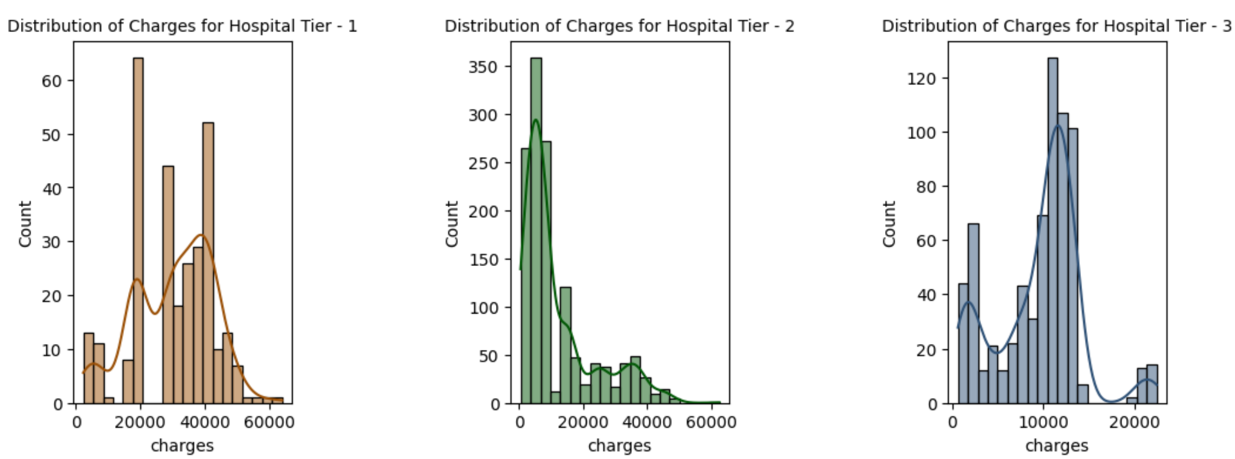
```
# Plot histogram for male and Female charges

fig, ax = plt.subplots(1, 2, figsize=(10, 4))
for i, (gender, label, color) in enumerate(zip([1, 0], ['Male',
'Female'], ['#020647', '#cf2340'])):
    sns.histplot(df[df['Gender'] == gender]['charges'], ax=ax[i],
kde=True, bins=20, color=color)
    ax[i].set_title(f'Distribution of Charges for {label}')

fig.subplots_adjust(wspace=1)
plt.show()
```



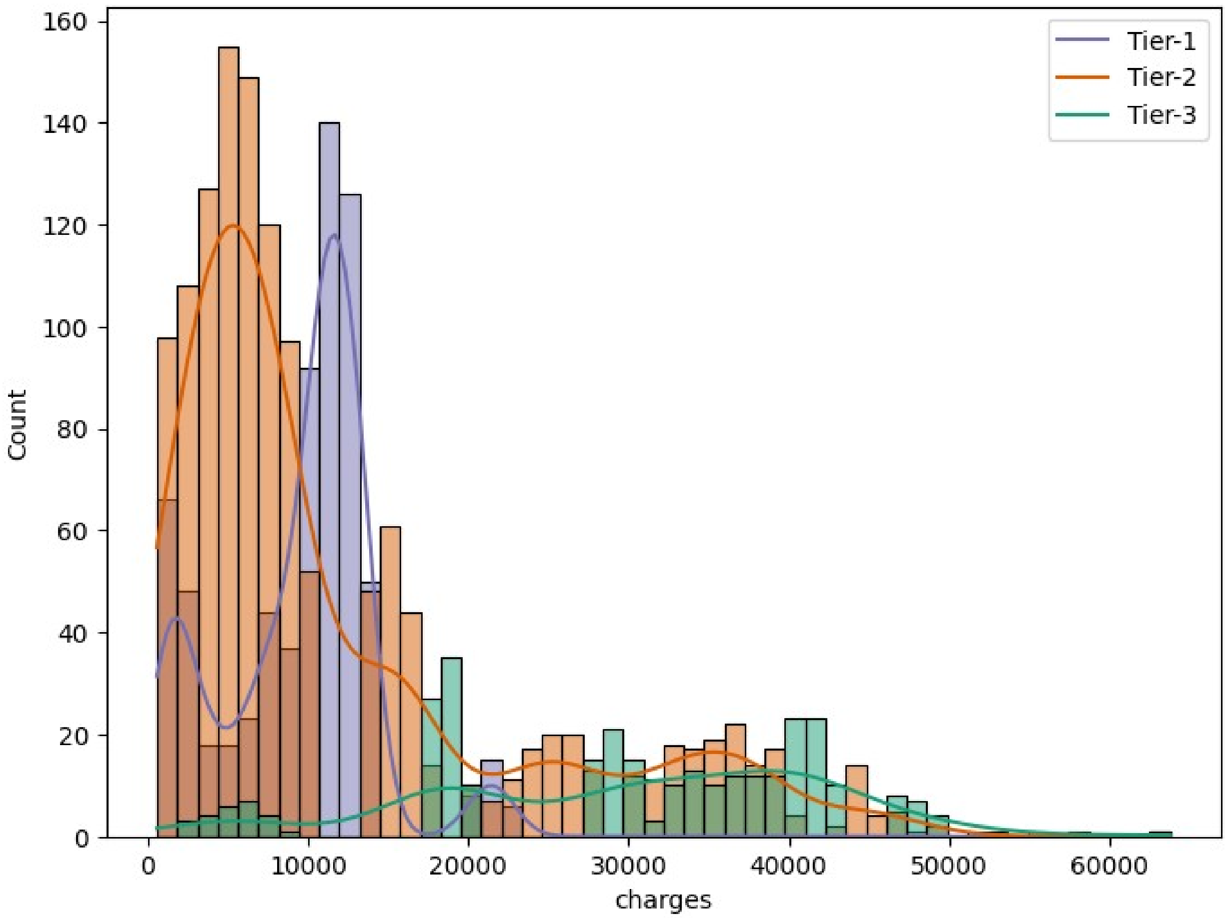

```
# Plot histogram for combined male and Female charges.
plt.figure(figsize=(12, 6))
custom_palette = ["#06b7ba", "#c70a36"]
sns.histplot(data=df, x='charges', hue='Gender', alpha=0.5, bins=50,
palette=custom_palette,kde=True)
plt.legend(labels=['Male', 'Female'])
plt.show()
```



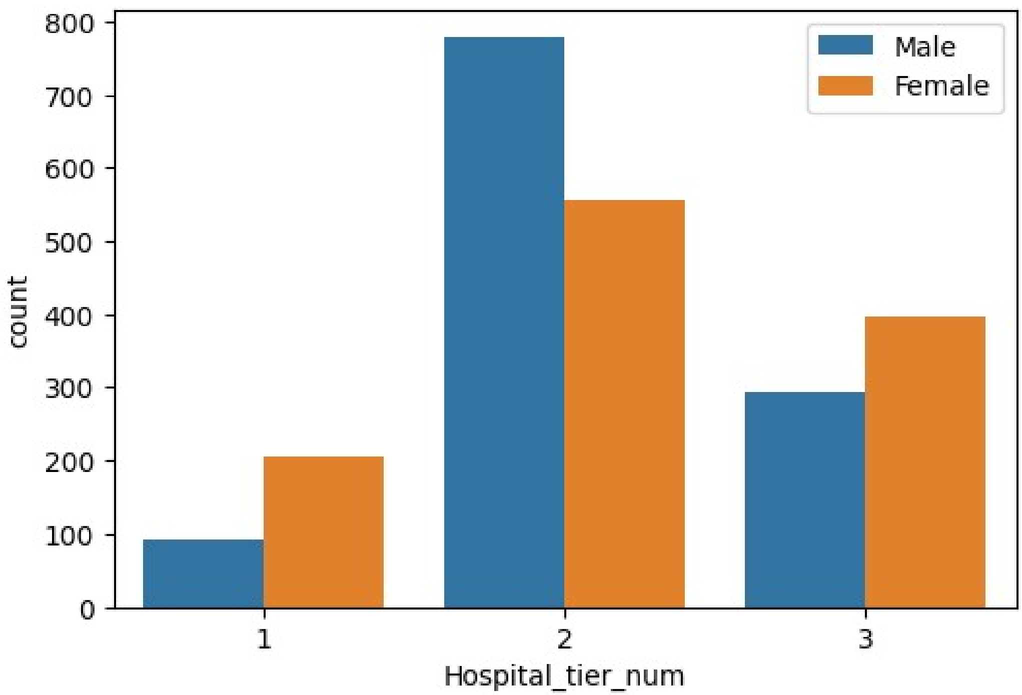

```python
# Plot histogram for Different Tier of hospital charges

fig, ax = plt.subplots(1,3,figsize=(12,4))
hospital_tier = [1,2,3]
color_pal = ['#9c5209','#045908','#315378']

for i, hospital_tier in enumerate(hospital_tier):
    sns.histplot(df[df['Hospital_tier_num'] == hospital_tier]
['charges'], ax=ax[i], kde=True, bins=20,color=color_pal[i])
    ax[i].set_title(f'Distribution of Charges for Hospital Tier -
{hospital_tier}', fontsize=10)

fig.subplots_adjust(wspace=1)
plt.show()
```



```
# Plot histogram for combined different Tier of hospital charges
plt.figure(figsize=(8, 6))
sns.histplot(data=df, x='charges', hue='Hospital_tier_num', alpha=0.5,
bins=50, palette='Dark2', kde=True)
plt.legend(labels=['Tier-1', 'Tier-2', 'Tier-3'])
plt.show()
```




```
# Plot countplot for different tiers of hospitals across gender.
plt.figure(figsize=(6, 4))
sns.countplot(data=df, x='Hospital_tier_num', hue='Gender')
plt.legend(labels=['Male', 'Female'])
plt.show()
```



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

```
# Calculate the median hospitalization cost for each tier of hospitals
tier_medians = df.groupby('Hospital_tier_num')
['charges'].median().reset_index()

#covert tier_medians into dataframe to plot radar plot
r_theta = {'r': tier_medians['charges'].tolist(), 'theta':['Tier-
1','Tier-2','Tier-3']}
radar_data = pd.DataFrame(r_theta)
radar_data

           r   theta
0  32097.435  Tier-1
1   7168.760  Tier-2
2  10676.830  Tier-3

import plotly.express as px
radar_plot=px.line_polar(radar_data,
r='r',theta='theta',line_close=True)
radar_plot.update_traces(fill='toself')
radar_plot.show()
```


```
{"config":{"plotlyServerURL":"https://plot.ly"},"data":
[{"fill":"toself","hovertemplate":"r=%{r}<br>theta=%{theta}<extra></
extra>","legendgroup":"","line":
{"color":"#636efa","dash":"solid"},"marker":
```

```
{"symbol":"circle"},"mode":"lines","name":"","r":
[32097.434999999998,7168.76,10676.83,32097.434999999998],"showlegend":
false,"subplot":"polar","theta":["Tier-1","Tier-2","Tier-3","Tier-
1"],"type":"scatterpolar"}],"layout":{"autosize":true,"legend":
{"tracegroupgap":0},"margin":{"t":60},"polar":{"angularaxis":
{"direction":"clockwise","rotation":90,"type":"category"},"domain":
{"x":[0,1],"y":[0,1]},"radialaxis":{"autorange":true,"range":
[0,32097.434999999998],"type":"linear"}},"template":{"data":{"bar":
[{"error_x":{"color":"#2a3f5f"},"error_y":
{"color":"#2a3f5f"},"marker":{"line":
{"color":"#E5ECF6","width":0.5},"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"bar"}],"barpo
lar":[{"marker":{"line":{"color":"#E5ECF6","width":0.5},"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"barpolar"}],"
carpet":[{"aaxis":
{"endlinecolor":"#2a3f5f","gridcolor":"white","linecolor":"white","min
orgridcolor":"white","startlinecolor":"#2a3f5f"},"baxis":
{"endlinecolor":"#2a3f5f","gridcolor":"white","linecolor":"white","min
orgridcolor":"white","startlinecolor":"#2a3f5f"},"type":"carpet"}],"ch
oropleth":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"choropleth"}],"contour":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"contour"}],"contourcarpet":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"contourcarpet"}],"heatmap":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"heatmap"}],"heatmapgl":[{"colorbar":
{"outlinewidth":0,"ticks":""},"colorscale":[[0,"#0d0887"],
[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],
[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],
[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],
[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"heatmapgl"}],"histogram":[{"marker":{"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"histogram"}],
"histogram2d":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
```

```
[1,"#f0f921"]],"type":"histogram2d"}],"histogram2dcontour":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"histogram2dcontour"}],"mesh3d":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"mesh3d"}],"parcoords":[{"line":
{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"parcoords"}],"pie":
[{"automargin":true,"type":"pie"}],"scatter":[{"fillpattern":
{"fillmode":"overlay","size":10,"solidity":0.2},"type":"scatter"}],"sc
atter3d":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"marker":
{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatter3d"}],"scattercarpet":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattercarpet"}],"scattergeo":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattergeo"}],"scattergl":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattergl"}],"scattermapbox":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattermapbox"}],"scatterpolar"
:[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterpolar"}],"scatterpolargl
":[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterpolargl"}],"scatterterna
ry":[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterternary"}],"surface":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"surface"}],"table":[{"cells":{"fill":
{"color":"#EBF0F8"},"line":{"color":"white"}},"header":{"fill":
{"color":"#C8D4E3"},"line":
{"color":"white"}},"type":"table"}]},"layout":{"annotationdefaults":
{"arrowcolor":"#2a3f5f","arrowhead":0,"arrowwidth":1},"autotypenumbers
":"strict","coloraxis":{"colorbar":
{"outlinewidth":0,"ticks":""}},"colorscale":{"diverging":
[[0,"#8e0152"],[0.1,"#c51b7d"],[0.2,"#de77ae"],[0.3,"#f1b6da"],
[0.4,"#fde0ef"],[0.5,"#f7f7f7"],[0.6,"#e6f5d0"],[0.7,"#b8e186"],
[0.8,"#7fbc41"],[0.9,"#4d9221"],[1,"#276419"]],"sequential":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
```

```
[0.8888888888888888,"#fdca26"],[1,"#f0f921"]],"sequentialminus":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],[1,"#f0f921"]]},"colorway":
["#636efa","#EF553B","#00cc96","#ab63fa","#FFA15A","#19d3f3","#FF6692"
,"#B6E880","#FF97FF","#FECB52"],"font":{"color":"#2a3f5f"},"geo":
{"bgcolor":"white","lakecolor":"white","landcolor":"#E5ECF6","showlake
s":true,"showland":true,"subunitcolor":"white"},"hoverlabel":
{"align":"left"},"hovermode":"closest","mapbox":
{"style":"light"},"paper_bgcolor":"white","plot_bgcolor":"#E5ECF6","po
lar":{"angularaxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"bgcolor":"#E5ECF
6","radialaxis":
{"gridcolor":"white","linecolor":"white","ticks":""}},"scene":
{"xaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
,"yaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
,"zaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
},"shapedefaults":{"line":{"color":"#2a3f5f"}},"ternary":{"aaxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"baxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"bgcolor":"#E5ECF
6","caxis":
{"gridcolor":"white","linecolor":"white","ticks":""}},"title":
{"x":5.0e-2},"xaxis":
{"automargin":true,"gridcolor":"white","linecolor":"white","ticks":"",
"title":
{"standoff":15},"zerolinecolor":"white","zerolinewidth":2},"yaxis":
{"automargin":true,"gridcolor":"white","linecolor":"white","ticks":"",
"title":{"standoff":15},"zerolinecolor":"white","zerolinewidth":2}}}}}
```

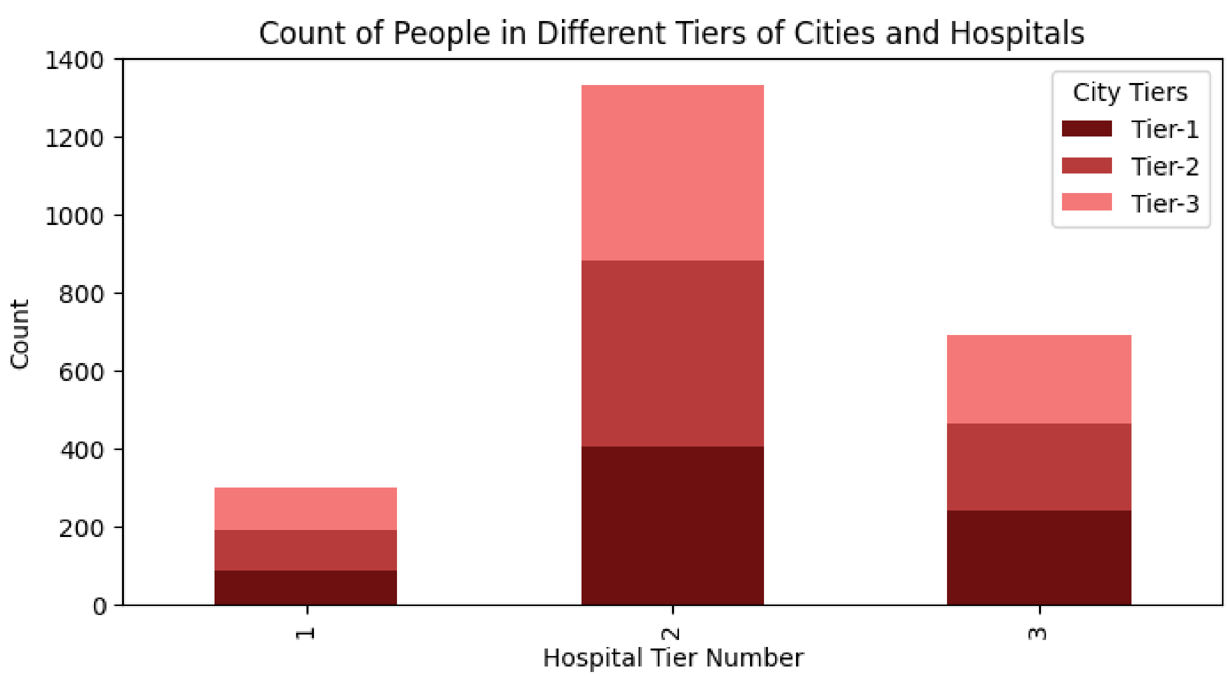

12. Create a frequency table and a stacked bar chart to visualize the count of people in the different tiers of cities and hospitals.

```
#Create crosstab frequency table
freq_table = pd.crosstab(df['Hospital_tier_num'],df['City_tier_num'])
freq_table

#Calculating hospital tier wise and City tier wise total population
frequency_table=freq_table.copy()
total_city_population = frequency_table.sum(axis=1)
total_hospital_population = frequency_table.sum(axis=0)
frequency_table['City_Total_Population'] = total_city_population
```

```
frequency_table.loc['Hosp_Total_Population'] =
total_hospital_population

#Calculating total polulation
total_population = total_hospital_population.sum()
frequency_table.loc['Hosp_Total_Population','City_Total_Population'] =
total_population
frequency_table
```

City_tier_num	1	2	3	City_Total_Population
Hospital_tier_num				
1	85.0	106.0	109.0	300.0
2	403.0	479.0	452.0	1334.0
3	241.0	222.0	228.0	691.0
Hosp_Total_Population	729.0	807.0	789.0	2325.0

```
# Create a stacked bar chart
fig, ax = plt.subplots(figsize=(8, 4))
freq_table.plot(kind='bar', stacked=True, ax=ax, color=['#6e1010',
'#b83b3b', '#f57878'])

plt.title('Count of People in Different Tiers of Cities and
Hospitals')
plt.xlabel('Hospital Tier Number')
plt.ylabel('Count')
plt.legend(title='City Tiers',labels=['Tier-1', 'Tier-2', 'Tier-3'])
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.

Question	X-Variable	Y-Variable	X-Variable Datatype	Y-Variable Datatype	Statistical Test
Q.13(a)	Hospital Tier	Avg charges	Categorical	Continuous	ANOVA
Q.13(b)	City Tier	Avg charges	Categorical	Continuous	ANOVA
Q.13(c)	Smoker	Avg charges	Categorical	Continuous	ANOVA
Q.13(d)	Smoker	Heart Issues	Categorical	Categorical	Chi Square

H0: Equal Means (or no relationship between variables)

Ha: Unequal Means (or effect or relationship between variables)

Q. 13(a)

```
# Checking the Avg. value of different Hospital Tiers
avg_tier1_charges=df[df['Hospital_tier_num'] == 1]['charges'].mean()
avg_tier2_charges=df[df['Hospital_tier_num'] == 2]['charges'].mean()
avg_tier3_charges=df[df['Hospital_tier_num'] == 3]['charges'].mean()
print('Avg. Tier-1 Hospital Charges: ',avg_tier1_charges,'\n', 'Avg.
Tier-2 Hospital Charges: ',avg_tier2_charges,'\n', 'Avg. Tier-3
Hospital Charges: ', avg_tier3_charges)

#Perform the ANOVA test for the average hospitalization costs for the
three types of hospitals.
Tier1_charges=df[df['Hospital_tier_num'] == 1]['charges']
Tier2_charges=df[df['Hospital_tier_num'] == 2]['charges']
Tier3_charges=df[df['Hospital_tier_num'] == 3]['charges']

from scipy.stats import f_oneway
f_stat, p_val = f_oneway(Tier1_charges,Tier2_charges,Tier3_charges)

print('F-statistic: %.3f' %  f_stat)
print('p-value: %.3f' %  p_val)

# Interpret the results
if p_val < 0.05:
```

```
    print('Reject the null hypothesis:','\n', 'Conclusion: The average hospitalization costs for the three types of hospitals are significantly different.')
else:
    print('Fail to reject the null hypothesis:','\n', 'Conclusion: The average hospitalization costs for the three types of hospitals are not significantly different.')
```

```
Avg. Tier-1 Hospital Charges:  30131.995899999998
 Avg. Tier-2 Hospital Charges:  11875.883860569715
 Avg. Tier-3 Hospital Charges:  9487.456222865412
F-statistic: 493.990
p-value: 0.000
Reject the null hypothesis:
 Conclusion: The average hospitalization costs for the three types of hospitals are significantly different.
```

Q. 13(b)

```
# Checking the Avg. value of different City Tiers
avg_tier1_charges=df[df['City_tier_num'] == 1]['charges'].mean()
avg_tier2_charges=df[df['City_tier_num'] == 2]['charges'].mean()
avg_tier3_charges=df[df['City_tier_num'] == 3]['charges'].mean()
print('Avg. Tier-1 Hospital Charges: ',avg_tier1_charges,'\n', 'Avg. Tier-2 Hospital Charges: ',avg_tier2_charges,'\n', 'Avg. Tier-3 Hospital Charges: ', avg_tier3_charges)

#Perform the ANOVA test for the average hospitalization costs for the three types of cities.
Tier1_charges=df[df['City_tier_num'] == 1]['charges']
Tier2_charges=df[df['City_tier_num'] == 2]['charges']
Tier3_charges=df[df['City_tier_num'] == 3]['charges']

f_stat, p_val = f_oneway(Tier1_charges,Tier2_charges,Tier3_charges)

print('F-statistic: %.3f' %  f_stat)
print('p-value: %.3f' %  p_val)

if p_val < 0.05:
    print('Reject the null hypothesis:','\n', 'Conclusion: The average hospitalization costs for the three types of cities are significantly different.')
else:
    print('Fail to reject the null hypothesis:','\n', 'Conclusion: The average hospitalization costs for the three types of cities are not significantly different.')
```

```
Avg. Tier-1 Hospital Charges:  13009.972578875171
 Avg. Tier-2 Hospital Charges:  13471.919281288725
 Avg. Tier-3 Hospital Charges:  14045.312065906212
```

```
F-statistic: 1.454
p-value: 0.234
Fail to reject the null hypothesis:
 Conclusion: The average hospitalization costs for the three types of
cities are not significantly different.
```

Q. 13(c)

```
# Checking the Avg. value of Smokers and Non-smokers
avg_Smoker_charges=df[df['smoker'] == 'yes']['charges'].mean()
avg_NonSmoker_charges=df[df['smoker'] == 'No']['charges'].mean()
print('Avg. Smoker Charges: ',avg_Smoker_charges,'\n', 'Avg. Non-
Smoker Charges: ',avg_NonSmoker_charges)

#Perform the ANOVA test for the average hospitalization costs for the
smokers.
Smoker_charges=df[df['smoker'] == 'yes']['charges']
NonSmoker_charges=df[df['smoker'] == 'No']['charges']

f_stat, p_val = f_oneway(Smoker_charges,NonSmoker_charges)

print('F-statistic: %.3f' %  f_stat)
print('p-value: %.3f' %  p_val)

if p_val < 0.05:
    print('Reject the null hypothesis:','\n', 'Conclusion: The average
hospitalization costs for the smokers and Non-smokers are
significantly different.')
else:
    print('Fail to reject the null hypothesis:','\n', 'Conclusion: The
average hospitalization costs for the smokers and Non-smokers are not
significantly different.')

Avg. Smoker Charges:  32866.96022633745
 Avg. Non-Smoker Charges:  8409.19924959217
F-statistic: 5499.054
p-value: 0.000
Reject the null hypothesis:
 Conclusion: The average hospitalization costs for the smokers and
Non-smokers are significantly different.
```

Q. 13(d)

```
# Create a contingency table
contingency_table = pd.crosstab(df['smoker'], df['Heart Issues'])
contingency_table

Heart Issues    No  yes
smoker
```

```
No              1108  731
yes              297  189

# Perform the Chi-Square test
from scipy.stats import chi2_contingency
chi2, p, dof, expected = chi2_contingency(contingency_table)

print('Chi-Square Statistic: %.3f' % chi2)
print('p-value: %.3f' % p)

# Interpret the results
if p < 0.05:
    print('Reject the null hypothesis:','\n', 'Conclusion: Smoking and
heart issues are not independent.')
else:
    print('Fail to reject the null hypothesis:','\n', 'Conclusion:
Smoking and heart issues are independent.')

Chi-Square Statistic: 0.086
p-value: 0.769
Fail to reject the null hypothesis:
 Conclusion: Smoking and heart issues are independent.
```

>> MACHINE LEARNING

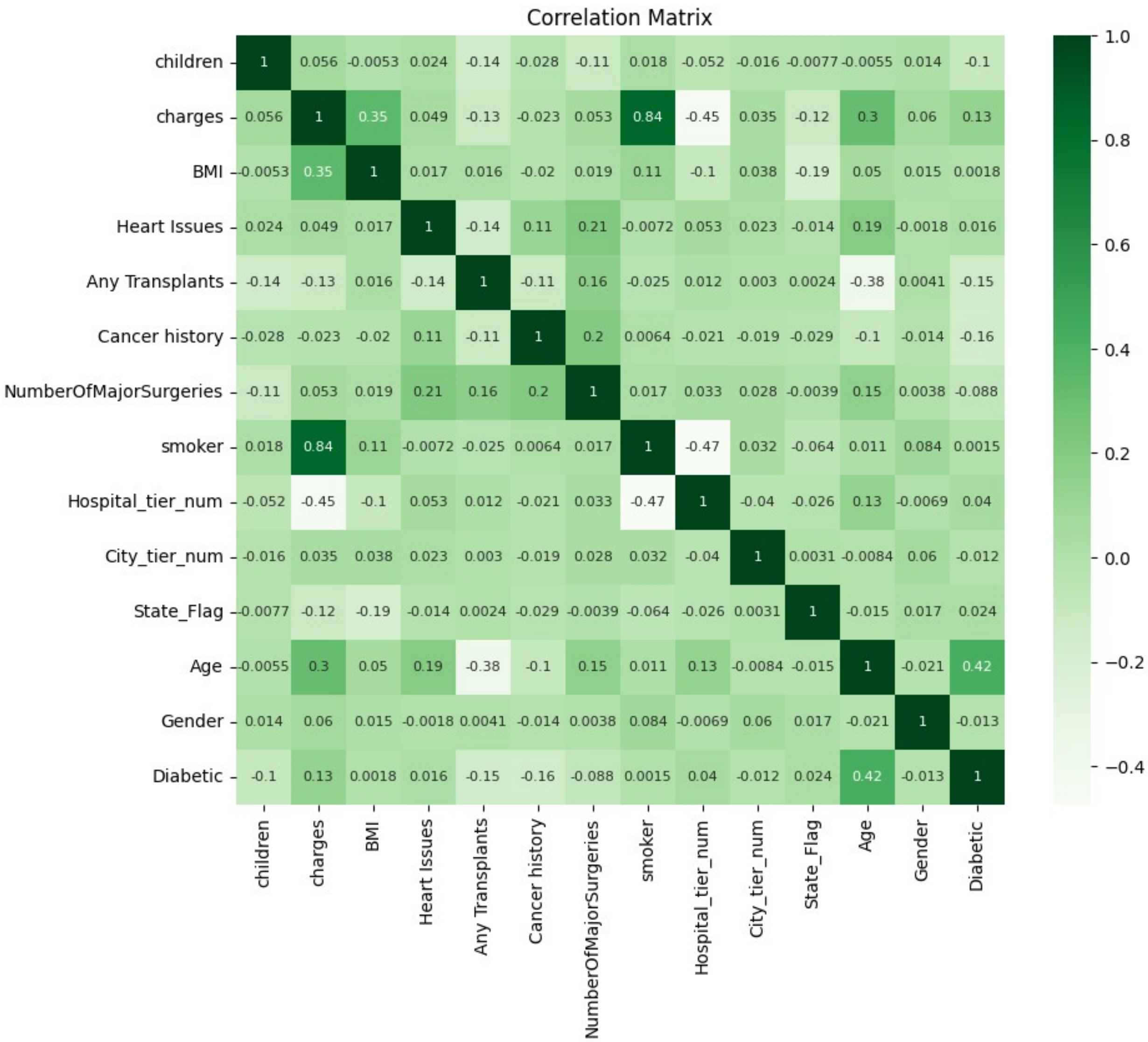
1. Examine the correlation between predictors to identify highly correlated predictors.

```
#Creting a new Dataframe with relevant features only
df1=df.copy()
df.drop(['Customer ID', 'year','month','date','Hospital tier','City
tier','State ID','name','D.O.B'], axis=1, inplace=True)
df.head()

   children  charges    BMI  HBA1C Heart Issues Any Transplants  \
0         0   563.84  17.58   4.51           No              No
1         0   570.62  17.60   4.39           No              No
2         0   600.00  16.47   6.35           No              No
3         0   604.54  17.70   6.28           No              No
4         0   637.26  22.34   5.57           No              No

  Cancer history  NumberOfMajorSurgeries smoker  Hospital_tier_num  \
0             No                       1     No                  2
1             No                       1     No                  2
2            Yes                       1     No                  2
3             No                       1     No                  3
4             No                       1     No                  3

   City_tier_num  State_Flag  Age  Gender
```

```
0               3            3   32        1
1               1            3   31        1
2               1            3   31        0
3               3            3   31        1
4               3            3   26        1

# Consider HBA1C>=8 as a Diabetic customer and assign diabetic
customer as 1 and non-diabetic as 0.
df['Diabetic'] = df['HBA1C'].apply(lambda x: '1' if x >= 8 else '0')
df.drop(['HBA1C'], axis=1, inplace=True)
df.head()

   children  charges    BMI Heart Issues Any Transplants Cancer
history  \
0         0   563.84  17.58           No              No
No
1         0   570.62  17.60           No              No
No
2         0   600.00  16.47           No              No
Yes
3         0   604.54  17.70           No              No
No
4         0   637.26  22.34           No              No
No

   NumberOfMajorSurgeries smoker  Hospital_tier_num  City_tier_num  \
0                        1     No                  2              3
1                        1     No                  2              1
2                        1     No                  2              1
3                        1     No                  3              3
4                        1     No                  3              3

   State_Flag  Age  Gender Diabetic
0           3   32       1        0
1           3   31       1        0
2           3   31       0        0
3           3   31       1        0
4           3   26       1        0

#Transform categorical columns into numerical using Label Encoding
df2=df.copy()
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
categorical_col=['Heart Issues','Any Transplants','Cancer
history','smoker']
for i in categorical_col:
    df[i] = le.fit_transform(df[i])
df3=df.copy()
df.head()
```

```
   children   charges     BMI  Heart Issues  Any Transplants  Cancer
history  \
0         0   563.84  17.58             0                0
0
1         0   570.62  17.60             0                0
0
2         0   600.00  16.47             0                0
1
3         0   604.54  17.70             0                0
0
4         0   637.26  22.34             0                0
0

   NumberOfMajorSurgeries  smoker  Hospital_tier_num  City_tier_num  \
0                       1       0                  2              3
1                       1       0                  2              1
2                       1       0                  2              1
3                       1       0                  3              3
4                       1       0                  3              3

   State_Flag  Age  Gender Diabetic
0           3   32       1        0
1           3   31       1        0
2           3   31       0        0
3           3   31       1        0
4           3   26       1        0
```

```
# Correction between predictors using Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='Greens',
square=True,annot_kws={'fontsize': 8})
plt.title('Correlation Matrix')
plt.show()
```


Correlation Matrix
children
charges
BMI
Heart Issues
Any Transplants
Cancer history
NumberOfMajorSurgeries
smoker
Hospital_tier_num
City_tier_num
State_Flag
Age
Gender
Diabetic
1 0.056 -0.0053 0.024 -0.14 -0.028 -0.11 0.018 -0.052 -0.016 -0.0077 -0.0055 0.014 -0.1
0.056 1 0.35 0.049 -0.13 -0.023 0.053 0.84 -0.45 0.035 -0.12 0.3 0.06 0.13
-0.0053 0.35 1 0.017 0.016 -0.02 0.019 0.11 -0.1 0.038 -0.19 0.05 0.015 0.0018
0.024 0.049 0.017 1 -0.14 0.11 0.21 -0.0072 0.053 0.023 -0.014 0.19 -0.0018 0.016
-0.14 -0.13 0.016 -0.14 1 -0.11 0.16 -0.025 0.012 0.003 0.0024 -0.38 0.0041 -0.15
-0.028 -0.023 -0.02 0.11 -0.11 1 0.2 0.0064 -0.021 -0.019 -0.029 -0.1 -0.014 -0.16
-0.11 0.053 0.019 0.21 0.16 0.2 1 0.017 0.033 0.028 -0.0039 0.15 0.0038 -0.088
0.018 0.84 0.11 -0.0072 -0.025 0.0064 0.017 1 -0.47 0.032 -0.064 0.011 0.084 0.0015
-0.052 -0.45 -0.1 0.053 0.012 -0.021 0.033 -0.47 1 -0.04 -0.026 0.13 -0.0069 0.04
-0.016 0.035 0.038 0.023 0.003 -0.019 0.028 0.032 -0.04 1 0.0031 -0.0084 0.06 -0.012
-0.0077 -0.12 -0.19 -0.014 0.0024 -0.029 -0.0039 -0.064 -0.026 0.0031 1 -0.015 0.017 0.024
-0.0055 0.3 0.05 0.19 -0.38 -0.1 0.15 0.011 0.13 -0.0084 -0.015 1 -0.021 0.42
0.014 0.06 0.015 -0.0018 0.0041 -0.014 0.0038 0.084 -0.0069 0.06 0.017 -0.021 1 -0.013
-0.1 0.13 0.0018 0.016 -0.15 -0.16 -0.088 0.0015 0.04 -0.012 0.024 0.42 -0.013 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

Observation:

The heatmap reveals some notable correlations:

- Charges are strongly correlated with smoking, and also show a significant correlation with BMI and age.

- Additionally, age is correlated with diabetes, indicating a possible relationship between these two factors.

2. Develop a regression model Linear or Ridge. Evaluate the model with k-fold cross validation.

Also, ensure that you apply all the following suggestions:

• Implement the stratified 5-fold cross validation technique for both model building and validation.

• Utilize effective standardization techniques and hyperparameter tuning.

• Incorporate sklearn-pipelines to streamline the workflow.

• Apply appropriate regularization techniques to address the bias-variance trade-off.

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

• Develop Gradient Boost model and determine the variable importance scores, and identify the redundant variables.

```
#Define X and Y variable for the Regression Model
y=df['charges']
df.pop('charges')
X=df
```

Regression Model with Ridge Regression using K-Fold Cross Validation

```
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error, 
explained_variance_score, r2_score
```

```python
# Define the k-fold cross validation object
ridge_kfold = KFold(n_splits=5, shuffle=True, random_state=34)

# Define the pipeline for Ridge Regression
ridge_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('ridge', Ridge())
])

# Define the hyperparameter tuning space for Ridge Regression
ridge_param_grid = {'ridge__alpha': [0.001, 0.01, 0.1, 1, 10]}

# Perform k-fold cross validation for Ridge Regression with
hyperparameter tuning
ridge_maes = []
ridge_mses = []
ridge_rmses = []
ridge_accuracies = []
ridge_r2s = []

for i, (ridge_train_idx, ridge_val_idx) in
enumerate(ridge_kfold.split(X)):
    ridge_X_train, ridge_X_val = X.iloc[ridge_train_idx],
X.iloc[ridge_val_idx]
    ridge_y_train, ridge_y_val = y.iloc[ridge_train_idx],
y.iloc[ridge_val_idx]
    print(f"Fold {i}:")
    print(f"  Train: index={ridge_train_idx}")
    print(f"  Test:  index={ridge_val_idx}")

    ridge_grid_search = GridSearchCV(ridge_pipeline, ridge_param_grid,
cv=5, scoring='neg_mean_squared_error')
    ridge_grid_search.fit(ridge_X_train, ridge_y_train)

    ridge_y_pred = ridge_grid_search.predict(ridge_X_val)

    ridge_maes.append(mean_absolute_error(ridge_y_val, ridge_y_pred))
    ridge_mses.append(mean_squared_error(ridge_y_val, ridge_y_pred))
    ridge_rmses.append(mean_squared_error(ridge_y_val,
ridge_y_pred)**(1/2.0))
    ridge_accuracies.append(explained_variance_score(ridge_y_val,
ridge_y_pred))
    ridge_r2s.append(r2_score(ridge_y_val, ridge_y_pred))

# Get the best hyperparameters and the corresponding model
ridge_best_params = ridge_grid_search.best_params_
ridge_best_model = ridge_grid_search.best_estimator_
print(f'Best Hyperparameters: {ridge_best_params}')
```

```
# Calculate the average of the metrics
avg_ridge_mae = sum(ridge_maes) / len(ridge_maes)
avg_ridge_mse = sum(ridge_mses) / len(ridge_mses)
avg_ridge_rmse = sum(ridge_rmses) / len(ridge_rmses)
avg_ridge_accuracy = sum(ridge_accuracies) / len(ridge_accuracies) *
100
avg_ridge_r2 = sum(ridge_r2s) / len(ridge_r2s)

print(f'Average MAE: {avg_ridge_mae}')
print(f'Average MSE: {avg_ridge_mse}')
print(f'Average RMSE: {avg_ridge_rmse}')
print(f'Average Accuracy: {avg_ridge_accuracy:.2f}%')
print(f'Average R-squared: {avg_ridge_r2:.2f}')
```

```
Fold 0:
  Train: index=[   0    1    2 ... 2322 2323 2324]
  Test:  index=[   5    8   28   45   48   49   54   56   66   71   72
86   90   91
   94   97   98  100  101  114  116  124  130  135  148  155  156  157
  158  160  163  164  169  177  180  183  191  192  206  210  227  228
  229  235  247  250  251  258  260  262  265  287  296  300  302  305
  307  308  310  319  325  332  336  347  348  353  360  364  371  379
  382  386  388  393  407  413  418  427  438  439  442  443  444  459
  465  474  477  482  486  497  502  506  509  516  534  542  544  552
  555  559  561  562  564  566  567  570  571  576  577  579  586  593
  600  602  612  617  624  629  631  632  635  639  649  654  655  659
  666  669  670  680  684  686  687  694  699  706  710  712  715  726
  732  734  735  743  755  765  766  767  769  784  786  787  790  792
  794  797  806  818  824  828  829  835  838  841  843  849  851  858
  869  870  874  881  888  892  897  902  909  910  912  922  930  935
  941  944  946  953  957  970  989  990 1001 1011 1017 1025 1028 1029
 1030 1031 1035 1053 1054 1063 1068 1071 1075 1077 1090 1095 1100 1107
 1110 1111 1117 1122 1133 1135 1137 1142 1145 1149 1158 1161 1162 1163
 1165 1174 1176 1177 1182 1185 1189 1199 1200 1214 1219 1226 1231 1238
 1241 1244 1252 1258 1269 1272 1273 1278 1281 1286 1289 1310 1312 1315
 1316 1327 1330 1333 1336 1340 1347 1349 1351 1353 1355 1357 1359 1361
 1370 1373 1375 1377 1386 1391 1401 1403 1407 1408 1410 1411 1425 1426
 1427 1432 1437 1445 1451 1454 1467 1474 1476 1477 1479 1480 1481 1482
 1483 1501 1503 1505 1506 1510 1512 1527 1529 1538 1539 1548 1554 1558
 1568 1579 1582 1584 1585 1591 1593 1603 1607 1609 1610 1614 1618 1619
 1626 1627 1630 1633 1638 1639 1641 1648 1649 1662 1663 1666 1670 1672
 1679 1681 1692 1693 1695 1703 1712 1713 1715 1718 1719 1720 1722 1727
 1730 1735 1738 1741 1742 1749 1750 1760 1762 1765 1771 1775 1779 1784
 1785 1805 1806 1809 1817 1820 1821 1825 1829 1831 1837 1846 1848 1855
 1857 1860 1871 1873 1875 1877 1878 1881 1885 1887 1899 1905 1915 1919
 1923 1924 1935 1960 1963 1964 1969 1981 1989 1993 2012 2026 2028 2029
 2030 2031 2047 2048 2050 2057 2058 2063 2068 2076 2088 2089 2090 2101
 2110 2113 2117 2121 2127 2128 2134 2136 2137 2141 2144 2149 2155 2156
 2158 2175 2189 2193 2194 2197 2198 2199 2204 2210 2223 2226 2233 2234
 2235 2252 2256 2265 2266 2275 2284 2288 2289 2297 2298 2307 2308 2314
```

```
 2318 2320 2321]
Fold 1:
  Train: index=[    0    1    2 ... 2322 2323 2324]
  Test:  index=[  12   22   23   24   31   32   34   35   50   58   59
61   68   75
   83   96  102  103  109  121  122  125  126  128  131  133  134  142
  144  151  154  161  172  175  179  182  195  197  199  211  212  217
  224  234  236  244  249  252  253  254  256  261  267  268  272  281
  286  292  295  299  301  309  311  318  320  321  323  329  334  335
  338  341  342  346  349  356  357  368  377  378  387  392  394  406
  416  420  421  423  424  431  434  456  457  466  473  479  483  490
  492  496  504  507  512  519  522  527  530  533  537  540  545  546
  547  553  556  560  563  565  568  575  578  580  581  583  584  589
  590  594  596  603  605  610  621  636  641  643  646  647  660  662
  664  665  668  677  678  681  683  688  690  692  709  714  718  720
  721  725  731  738  744  748  750  751  754  756  763  768  770  773
  788  804  807  809  813  817  825  826  831  833  837  842  848  855
  857  860  868  877  880  885  886  890  898  899  903  908  914  925
  934  936  947  948  955  956  962  963  968  969  971  972  976  977
  978  980  983  985  988  994 1000 1004 1010 1014 1015 1023 1036 1037
 1049 1051 1052 1057 1059 1066 1067 1072 1081 1086 1093 1098 1113 1134
 1136 1138 1139 1140 1141 1143 1147 1148 1151 1164 1168 1169 1183 1184
 1188 1191 1193 1194 1198 1202 1217 1224 1225 1229 1234 1235 1250 1255
 1256 1259 1262 1264 1270 1275 1276 1279 1283 1288 1292 1295 1296 1300
 1322 1334 1337 1338 1339 1341 1343 1344 1354 1356 1364 1379 1384 1385
 1388 1395 1398 1400 1406 1419 1421 1431 1435 1438 1439 1446 1450 1453
 1456 1458 1469 1471 1473 1475 1485 1490 1491 1500 1511 1515 1519 1532
 1533 1536 1540 1541 1544 1550 1560 1577 1581 1589 1598 1602 1605 1608
 1612 1625 1628 1632 1645 1646 1647 1661 1665 1667 1669 1686 1694 1700
 1706 1714 1723 1726 1732 1734 1746 1747 1748 1751 1764 1768 1769 1773
 1776 1780 1786 1793 1794 1803 1808 1814 1816 1835 1838 1853 1856 1859
 1865 1870 1882 1884 1889 1892 1894 1903 1906 1907 1908 1909 1910 1921
 1922 1925 1929 1930 1931 1938 1939 1944 1945 1958 1973 1975 1976 1982
 1995 2004 2009 2016 2018 2027 2034 2041 2046 2052 2053 2055 2060 2064
 2077 2078 2082 2085 2087 2099 2100 2102 2103 2114 2115 2122 2123 2125
 2129 2138 2143 2148 2161 2163 2164 2171 2176 2188 2190 2192 2195 2202
 2213 2242 2247 2253 2255 2257 2261 2263 2267 2272 2276 2277 2304 2309
 2310 2312 2315]
Fold 2:
  Train: index=[    1    2    3 ... 2320 2321 2324]
  Test:  index=[   0    4    7   14   16   17   18   20   27   38   39
40   41   46
   47   51   52   57   60   62   67   77   80   84   85   87   88   95
   99  104  105  110  111  112  123  138  167  171  173  181  185  188
  193  198  202  208  216  219  220  225  232  257  259  263  269  271
  275  276  277  280  283  285  289  290  293  294  303  313  315  316
  317  322  331  337  339  340  343  345  355  359  366  369  370  374
  375  385  389  397  402  404  405  411  414  417  422  445  447  448
  451  460  462  470  471  472  476  487  491  494  503  508  511  518
```

```
  520  521  523  525  526  529  531  532  535  539  549  550  551  569
  582  591  597  599  604  620  623  625  626  627  633  645  650  651
  653  658  674  679  693  697  698  703  707  708  717  719  722  724
  728  729  741  747  753  757  758  764  782  793  795  796  798  801
  805  810  815  822  823  834  836  844  846  847  850  853  854  856
  859  865  866  871  872  875  878  893  896  904  905  906  907  911
  915  917  921  923  927  928  933  937  943  949  954  958  964  965
  982  986  987  995 1005 1012 1019 1020 1021 1033 1040 1043 1046 1056
 1064 1069 1080 1083 1087 1088 1097 1099 1102 1103 1104 1112 1114 1115
 1121 1144 1152 1153 1154 1160 1172 1173 1175 1178 1180 1187 1197 1203
 1204 1206 1207 1211 1215 1216 1221 1223 1245 1246 1247 1249 1265 1266
 1274 1277 1287 1290 1291 1293 1298 1304 1306 1307 1309 1311 1313 1314
 1323 1325 1326 1328 1342 1345 1366 1367 1369 1378 1380 1383 1387 1392
 1404 1415 1417 1418 1420 1424 1428 1433 1442 1443 1449 1452 1455 1459
 1462 1465 1466 1489 1493 1495 1497 1498 1502 1504 1509 1513 1521 1522
 1524 1535 1543 1545 1546 1551 1553 1578 1580 1583 1590 1592 1595 1601
 1604 1616 1617 1620 1621 1622 1631 1651 1652 1655 1657 1658 1671 1674
 1676 1688 1689 1697 1698 1699 1707 1709 1710 1728 1729 1731 1736 1739
 1740 1745 1755 1787 1789 1791 1796 1797 1804 1807 1823 1830 1842 1844
 1847 1858 1861 1864 1867 1874 1876 1880 1886 1893 1902 1904 1913 1916
 1917 1918 1926 1932 1934 1940 1942 1943 1951 1952 1955 1967 1987 1990
 1994 1996 1997 1999 2000 2021 2022 2023 2025 2033 2045 2054 2061 2062
 2069 2072 2074 2081 2083 2084 2086 2093 2097 2106 2124 2132 2135 2139
 2140 2142 2146 2150 2153 2159 2162 2169 2180 2181 2182 2185 2186 2205
 2206 2208 2221 2225 2229 2232 2246 2251 2259 2260 2278 2280 2286 2299
 2303 2322 2323]
Fold 3:
  Train: index=[   0    1    3 ... 2321 2322 2323]
  Test:  index=[   2    9   13   15   29   30   42   63   64   65   69
70   73   74
   76   81   89   92   93  118  120  132  141  143  150  152  159  162
  165  168  174  176  178  184  187  189  196  200  203  204  207  214
  215  218  221  226  231  237  245  255  266  270  273  284  306  312
  327  344  361  381  383  384  391  395  396  398  400  401  403  408
  415  425  426  430  446  454  463  468  469  475  478  489  493  499
  505  510  513  515  536  538  554  557  558  572  587  588  592  595
  607  608  609  611  613  615  616  618  619  630  637  640  652  656
  663  667  671  673  675  676  682  695  702  704  713  716  736  739
  745  749  752  760  771  777  779  780  781  783  785  800  820  821
  832  839  840  845  852  862  864  873  876  879  882  884  887  891
  895  900  901  913  918  919  920  924  926  931  932  939  940  942
  951  966  973  979  991  992  996  997  998  999 1002 1003 1007 1026
 1034 1039 1045 1050 1058 1062 1065 1074 1076 1078 1084 1085 1089 1105
 1109 1118 1119 1127 1128 1132 1155 1156 1167 1181 1186 1192 1195 1201
 1205 1213 1220 1222 1228 1233 1236 1237 1240 1242 1251 1253 1257 1260
 1261 1268 1294 1297 1299 1302 1308 1318 1319 1321 1329 1331 1335 1360
 1363 1365 1374 1376 1390 1393 1394 1396 1397 1402 1409 1416 1423 1430
 1440 1444 1457 1460 1461 1464 1478 1484 1486 1487 1488 1492 1494 1496
 1517 1523 1525 1528 1530 1531 1534 1547 1552 1555 1562 1563 1564 1566
```

```
 1567 1569 1570 1572 1573 1574 1575 1576 1586 1587 1594 1596 1597 1599
 1600 1611 1615 1624 1635 1636 1637 1640 1642 1643 1644 1650 1653 1659
 1660 1664 1673 1678 1683 1684 1685 1687 1690 1702 1704 1711 1716 1717
 1721 1725 1759 1761 1766 1767 1777 1778 1781 1782 1788 1790 1792 1798
 1801 1811 1813 1815 1819 1822 1824 1826 1827 1828 1833 1836 1839 1843
 1845 1849 1850 1851 1852 1854 1879 1883 1888 1890 1895 1896 1901 1911
 1912 1920 1927 1928 1941 1946 1947 1949 1953 1962 1965 1966 1970 1974
 1977 1983 1985 2001 2003 2006 2007 2008 2011 2013 2014 2017 2019 2024
 2032 2035 2036 2037 2039 2040 2044 2049 2051 2067 2071 2073 2075 2079
 2080 2096 2098 2104 2108 2109 2111 2116 2118 2119 2120 2126 2130 2131
 2157 2160 2172 2173 2177 2178 2179 2184 2196 2201 2203 2207 2209 2211
 2214 2218 2219 2227 2228 2236 2237 2238 2240 2243 2244 2249 2250 2262
 2264 2269 2270 2271 2273 2287 2293 2294 2296 2301 2302 2305 2306 2311
 2317 2319 2324]
Fold 4:
  Train: index=[   0    2    4 ... 2322 2323 2324]
  Test:  index=[   1    3    6   10   11   19   21   25   26   33   36
37   43   44
   53   55   78   79   82  106  107  108  113  115  117  119  127  129
  136  137  139  140  145  146  147  149  153  166  170  186  190  194
  201  205  209  213  222  223  230  233  238  239  240  241  242  243
  246  248  264  274  278  279  282  288  291  297  298  304  314  324
  326  328  330  333  350  351  352  354  358  362  363  365  367  372
  373  376  380  390  399  409  410  412  419  428  429  432  433  435
  436  437  440  441  449  450  452  453  455  458  461  464  467  480
  481  484  485  488  495  498  500  501  514  517  524  528  541  543
  548  573  574  585  598  601  606  614  622  628  634  638  642  644
  648  657  661  672  685  689  691  696  700  701  705  711  723  727
  730  733  737  740  742  746  759  761  762  772  774  775  776  778
  789  791  799  802  803  808  811  812  814  816  819  827  830  861
  863  867  883  889  894  916  929  938  945  950  952  959  960  961
  967  974  975  981  984  993 1006 1008 1009 1013 1016 1018 1022 1024
 1027 1032 1038 1041 1042 1044 1047 1048 1055 1060 1061 1070 1073 1079
 1082 1091 1092 1094 1096 1101 1106 1108 1116 1120 1123 1124 1125 1126
 1129 1130 1131 1146 1150 1157 1159 1166 1170 1171 1179 1190 1196 1208
 1209 1210 1212 1218 1227 1230 1232 1239 1243 1248 1254 1263 1267 1271
 1280 1282 1284 1285 1301 1303 1305 1317 1320 1324 1332 1346 1348 1350
 1352 1358 1362 1368 1371 1372 1381 1382 1389 1399 1405 1412 1413 1414
 1422 1429 1434 1436 1441 1447 1448 1463 1468 1470 1472 1499 1507 1508
 1514 1516 1518 1520 1526 1537 1542 1549 1556 1557 1559 1561 1565 1571
 1588 1606 1613 1623 1629 1634 1654 1656 1668 1675 1677 1680 1682 1691
 1696 1701 1705 1708 1724 1733 1737 1743 1744 1752 1753 1754 1756 1757
 1758 1763 1770 1772 1774 1783 1795 1799 1800 1802 1810 1812 1818 1832
 1834 1840 1841 1862 1863 1866 1868 1869 1872 1891 1897 1898 1900 1914
 1933 1936 1937 1948 1950 1954 1956 1957 1959 1961 1968 1971 1972 1978
 1979 1980 1984 1986 1988 1991 1992 1998 2002 2005 2010 2015 2020 2038
 2042 2043 2056 2059 2065 2066 2070 2091 2092 2094 2095 2105 2107 2112
 2133 2145 2147 2151 2152 2154 2165 2166 2167 2168 2170 2174 2183 2187
 2191 2200 2212 2215 2216 2217 2220 2222 2224 2230 2231 2239 2241 2245
```

```
 2248 2254 2258 2268 2274 2279 2281 2282 2283 2285 2290 2291 2292 2295
 2300 2313 2316]
Best Hyperparameters: {'ridge__alpha': 0.001}
Average MAE: 2753.121926443369
Average MSE: 20010465.494363222
Average RMSE: 4461.915143040959
Average Accuracy: 85.67%
Average R-squared: 0.86
```

The Ridge Regression model with an alpha of 0.001 demonstrates strong predictive performance, with a high R-squared value and accuracy, along with reasonably low error metrics (MAE, MSE, RMSE). This suggests that the model is well-regularized and effectively captures the underlying patterns in the data while maintaining generalizability across different folds of the cross-validation.

Gradient Boosting Model with Variable Importance Scores

```python
from sklearn.ensemble import GradientBoostingRegressor

# Define the pipeline for Gradient Boosting
gb_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('gb', GradientBoostingRegressor())
])

# Define the hyperparameter tuning space for Gradient Boosting
gb_param_grid = {
    'gb__n_estimators': [100, 200, 300, 400, 500, 1000],
    #'gb__learning_rate': [0.3, 0.5, 1],
    #'gb__max_depth': [1,2],
    #'gb__min_samples_split': [3,4,5],
    #'gb__min_samples_leaf': [1,2]
}

# Define the k-fold cross validation object
gb_kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Perform k-fold cross validation for Gradient Boosting with
hyperparameter tuning
gb_maes = []
gb_mses = []
gb_rmses = []
gb_accuracies = []
gb_r2s = []

for i, (gb_train_idx, gb_val_idx) in enumerate(gb_kfold.split(X)):
    gb_X_train, gb_X_val = X.iloc[gb_train_idx], X.iloc[gb_val_idx]
    gb_y_train, gb_y_val = y.iloc[gb_train_idx], y.iloc[gb_val_idx]
    print(f"Fold {i+1}:")
```

```
    #print(f"  Train: index={gb_train_idx}")
    #print(f"  Test:  index={gb_val_idx}")

    gb_grid_search = GridSearchCV(gb_pipeline, gb_param_grid, cv=5,
scoring='neg_mean_squared_error',verbose=2)
    gb_grid_search.fit(gb_X_train, gb_y_train)

    gb_y_pred = gb_grid_search.predict(gb_X_val)

    gb_maes.append(mean_absolute_error(gb_y_val, gb_y_pred))
    gb_mses.append(mean_squared_error(gb_y_val, gb_y_pred))
    gb_rmses.append(mean_squared_error(gb_y_val, gb_y_pred)**(1/2.0))
    gb_accuracies.append(explained_variance_score(gb_y_val,
gb_y_pred))
    gb_r2s.append(r2_score(gb_y_val, gb_y_pred))

# Get the best hyperparameters and the corresponding model
gb_best_params = gb_grid_search.best_params_
gb_best_model = gb_grid_search.best_estimator_
print(f'Best Hyperparameters: {gb_best_params}')
```

```
Fold 1:
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
time=   0.3s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
```

```
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
time=   0.3s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
[CV] END ...............................gb__n_estimators=400; total
time=   1.0s
[CV] END ...............................gb__n_estimators=400; total
time=   0.9s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
Fold 2:
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
```

```
[CV] END ...............................gb__n_estimators=200; total
time=   0.3s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.7s
[CV] END ...............................gb__n_estimators=300; total
time=   0.7s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.8s
[CV] END ...............................gb__n_estimators=400; total
time=   1.3s
[CV] END ...............................gb__n_estimators=400; total
time=   1.0s
[CV] END ...............................gb__n_estimators=400; total
time=   0.9s
[CV] END ...............................gb__n_estimators=400; total
time=   1.3s
[CV] END ...............................gb__n_estimators=400; total
time=   1.0s
Fold 3:
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.1s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
[CV] END ...............................gb__n_estimators=200; total
```

```
time=   0.6s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
time=   0.3s
[CV] END ...............................gb__n_estimators=200; total
time=   0.3s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.5s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
[CV] END ...............................gb__n_estimators=400; total
time=   0.9s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
Fold 4:
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
```

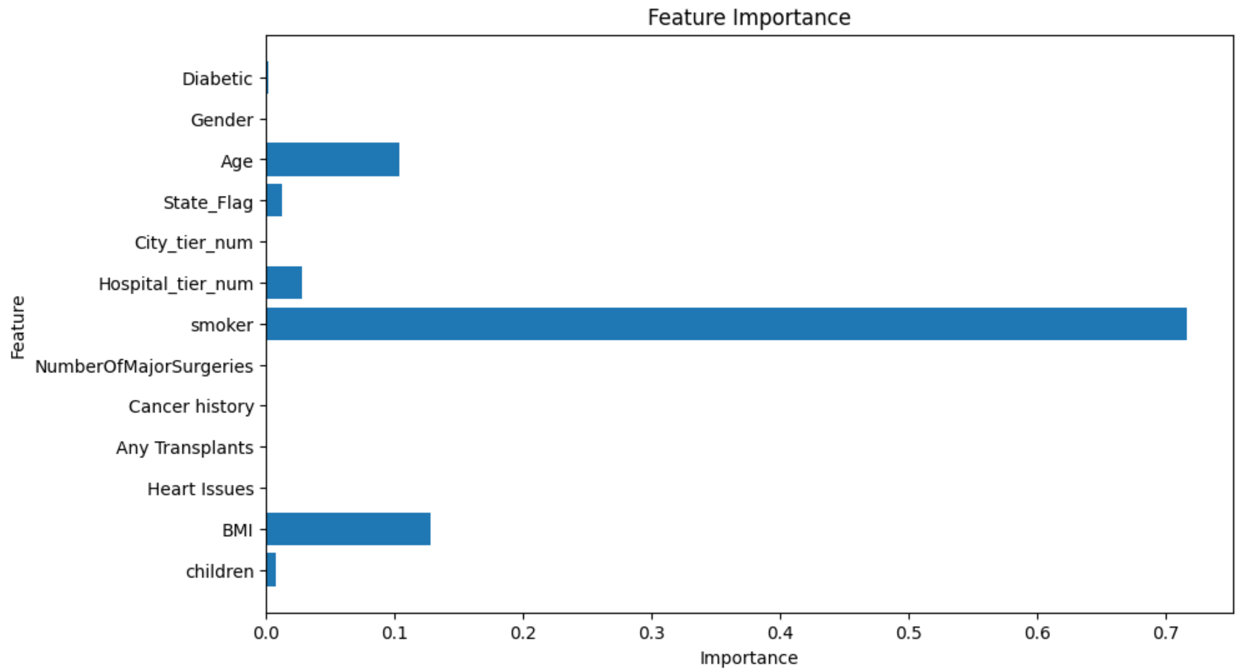
```
[CV] END ...............................gb__n_estimators=100; total
time=   0.2s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=400; total
time=   0.9s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
Fold 5:
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ................................gb__n_estimators=50; total
time=   0.0s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
```

```
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=100; total
time=   0.1s
[CV] END ...............................gb__n_estimators=200; total
time=   0.5s
[CV] END ...............................gb__n_estimators=200; total
time=   0.3s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=200; total
time=   0.3s
[CV] END ...............................gb__n_estimators=200; total
time=   0.4s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=300; total
time=   0.6s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
[CV] END ...............................gb__n_estimators=400; total
time=   0.8s
[CV] END ...............................gb__n_estimators=400; total
time=   0.9s
[CV] END ...............................gb__n_estimators=400; total
time=   1.3s
[CV] END ...............................gb__n_estimators=400; total
time=   1.3s
Best Hyperparameters: {'gb__n_estimators': 400}

# Calculate the average of the metrics
avg_gb_mae = sum(gb_maes) / len(gb_maes)
avg_gb_mse = sum(gb_mses) / len(gb_mses)
avg_gb_rmse = sum(gb_rmses) / len(gb_rmses)
avg_gb_accuracy = sum(gb_accuracies) / len(gb_accuracies) * 100
avg_gb_r2 = sum(gb_r2s) / len(gb_r2s)

print(f'Average MAE: {avg_gb_mae}')
print(f'Average MSE: {avg_gb_mse}')
print(f'Average RMSE: {avg_gb_rmse}')
print(f'Average Accuracy: {avg_gb_accuracy:.2f}%')
print(f'Average R-squared: {avg_gb_r2:.2f}')
```

```
Average MAE: 2125.803116928094
Average MSE: 13082270.633482825
Average RMSE: 3584.6096743740395
Average Accuracy: 90.80%
Average R-squared: 0.91
```

The Gradient Boosting model outperforms the Ridge Regression model in terms of error metrics (MAE, MSE, RMSE), accuracy, and R-squared value. This suggests that Gradient Boosting is better suited for this regression task, providing more accurate and reliable predictions compared to Ridge Regression.

```
# Get the feature importance scores
feature_importances =
gb_best_model.named_steps['gb'].feature_importances_

# Create a DataFrame to store the feature importance scores
df_feature_importances = pd.DataFrame({'Feature': X.columns,
'Importance': feature_importances})

# Sort the DataFrame by importance in descending order
df_feature_importances_sort =
df_feature_importances.sort_values(by='Importance', ascending=False)
print(df_feature_importances_sort)

                  Feature  Importance
6                  smoker    0.717429
1                     BMI    0.127649
10                    Age    0.103964
7       Hospital_tier_num    0.027680
9              State_Flag    0.011800
0                children    0.007196
12               Diabetic    0.001128
11                 Gender    0.001030
8           City_tier_num    0.000821
2            Heart Issues    0.000607
5   NumberOfMajorSurgeries    0.000324
3         Any Transplants    0.000237
4          Cancer history    0.000136

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(range(len(df_feature_importances)),
df_feature_importances['Importance'])
plt.yticks(range(len(df_feature_importances)),
df_feature_importances['Feature'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance')
plt.show()
```



The feature importance scores show that 'smoker' is the most significant predictor, while features like 'Cancer history' and 'Any Transplants' have minimal impact.

```
# Identify redundant variables by selecting features with importance < 0.01
redundant_variables =
df_feature_importances[df_feature_importances['Importance'] < 0.01]
['Feature'].tolist()
print(f'Redundant Variables: {redundant_variables}')
```

```
Redundant Variables: ['children', 'Heart Issues', 'Any Transplants',
'Cancer history', 'NumberOfMajorSurgeries', 'City_tier_num', 'Gender',
'Diabetic']
```

3. Case scenario:

Estimate the cost of hospitalization for Christopher, Ms. Jayna (Date of birth 12/28/1988; height 170 cm; and weight 85 kgs). She lives with her partner and two children in a tier-1 city, and her state's State ID is R1011. 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.

```
#Age Calculation
jayna_dob ='12/28/1988'
jayna_dob = pd.to_datetime(jayna_dob, format='%m/%d/%Y')
```

```
jayna_age = age_calculation(jayna_dob.date())
print(jayna_age)

35

#BMI Calculation
jayna_height = 170
jayna_weight = 85
jayna_bmi = round(jayna_weight / ((jayna_height/100) ** 2),2)     #
Convert height from cm to m
print(jayna_bmi)

29.41

#Make Dataframe
jayna_df =pd.DataFrame({'children':[2], 'BMI':[jayna_bmi],
       'Heart Issues':[0], 'Any Transplants':[0], 'Cancer history':
[1],
       'NumberOfMajorSurgeries':[0], 'smoker':[1],
'Hospital_tier_num':[1],
       'City_tier_num':[1], 'State_Flag':[1], 'Age':[jayna_age],
'Gender': [0], 'Diabetic':[0]})
jayna_df

   children    BMI  Heart Issues  Any Transplants  Cancer history  \
0         2  29.41             0                0               1

   NumberOfMajorSurgeries  smoker  Hospital_tier_num  City_tier_num  \
0                       0       1                  1              1

   State_Flag  Age  Gender  Diabetic
0           1   35       0         0

#Hospital_cost Calculation
ridge_hospitalization_cost = ridge_grid_search.predict(jayna_df)
print(f"Predicted Hospitalization Cost:
{round(ridge_hospitalization_cost[0], 3)}")

Predicted Hospitalization Cost: 31879.883
```

4. Find the predicted hospitalization cost using the best models (Gradient Boosting Model)

```
gb_hospitalization_cost= gb_grid_search.predict(jayna_df)
print(f"Predicted Hospitalization Cost:
{round(gb_hospitalization_cost[0],3)}")

Predicted Hospitalization Cost: 27831.229
```