project-healthcareinsurance-1

April 8, 2023

Healthcare Insurance Analysis

Week 1: Data Science or Data Analysis

[1]:
```
# Importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[2]:
```
hosp_details = pd.read_csv("Hospitalisation details.csv")
medical_exams = pd.read_csv("Medical Examinations.csv")
customer_names = pd.read_excel("Names.xlsx")
```

[3]:
```
hosp_details.head(2)
```

[3]:
```
  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

  State ID
0    R1013
1    R1013
```

[4]:
```
hosp_details.shape
```

[4]: (2343, 9)

[5]:
```
medical_exams.head(2)
```

[5]:
```
  Customer ID    BMI  HBA1C Heart Issues Any Transplants Cancer history  \
0         Id1  47.41   7.47           No              No             No
1         Id2  30.36   5.77           No              No             No

  NumberOfMajorSurgeries smoker
0       No major surgery    yes
1       No major surgery    yes
```

[6]:
```
medical_exams.shape
```

[6]:
```
(2335, 8)
```

[7]:
```
customer_names.head(2)
```

[7]:
```
  Customer ID                    name
0         Id1        Hawks, Ms.  Kelly
1         Id2  Lehner, Mr.  Matthew D
```

[8]:
```
customer_names.shape
```

[8]:
```
(2335, 2)
```

Week 1

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

[9]:
```
combined_df = pd.merge(customer_names, hosp_details, on = "Customer ID")
combined_df.head(2)
```

[9]:
```
  Customer ID                    name  year month  date  children   charges  \
0         Id1        Hawks, Ms.  Kelly  1968   Oct    12         0  63770.43
1         Id2  Lehner, Mr.  Matthew D  1977   Jun     8         0  62592.87

  Hospital tier City tier State ID
0      tier - 1  tier - 3    R1013
1      tier - 2  tier - 3    R1013
```

[10]:
```
final_data = pd.merge(combined_df, medical_exams, on = "Customer ID")
final_data.head(2)
```

[10]:
```
  Customer ID                    name  year month  date  children   charges  \
0         Id1        Hawks, Ms.  Kelly  1968   Oct    12         0  63770.43
1         Id2  Lehner, Mr.  Matthew D  1977   Jun     8         0  62592.87

  Hospital tier City tier State ID    BMI  HBA1C Heart Issues Any Transplants  \
0      tier - 1  tier - 3    R1013  47.41   7.47           No              No
1      tier - 2  tier - 3    R1013  30.36   5.77           No              No

  Cancer history NumberOfMajorSurgeries smoker
0             No      No major surgery    yes
1             No      No major surgery    yes
```

[11]:
```
final_data.shape
```

[11]:
```
(2335, 17)
```

2. Check for missing values in the dataset

[12]:
```
# Let us check for missing values in the data set
final_data.isnull().sum()
```

[12]:
```
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
```

[13]:
```
# We can see that there are no null values in the dataset
```

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

[14]:
```
trivial_value = final_data[final_data.eq("?").any(1)]
trivial_value.head(2)
```

[14]:
```
    Customer ID             name  year month  date  children   charges  \
2           Id3       Lu, Mr.  Phil  1970     ?    11         3  60021.40
169       Id170  Torphy, Mr.  Bobby  2000   Sep     5         1  37165.16

    Hospital tier City tier State ID     BMI  HBA1C Heart Issues  \
2        tier - 1  tier - 1    R1012  34.485  11.87          yes
169      tier - 1  tier - 3        ?  37.620   6.32          yes

    Any Transplants Cancer history NumberOfMajorSurgeries smoker
2                No             No                      2    yes
169             yes             No                      2    yes
```

[15]:
```
round(trivial_value.shape[0]/final_data.shape[0]*100, 2)
```

[15]: 0.43

[16]:
```
# 43% of rows have trivial values
```

[17]:
```
# Drop the rows containing the trivial values in the data set.
final_data.drop(final_data[final_data.eq("?").any(1)].index, axis=0,␣
↪inplace=True)
```

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

[18]:
```
# First we will deal with the nominal & categorical variable.
```

[19]:
```
final_data["Heart Issues"].value_counts()
final_data["Any Transplants"].value_counts()
final_data["Cancer history"].value_counts()
final_data["smoker"].value_counts()
```

[19]:
```
No     1839
yes     486
Name: smoker, dtype: int64
```

[20]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

[21]:
```
final_data["Heart Issues"] = le.fit_transform(final_data["Heart Issues"])
final_data["Any Transplants"] = le.fit_transform(final_data["Any Transplants"])
final_data["Cancer history"] = le.fit_transform(final_data["Cancer history"])
final_data["smoker"] = le.fit_transform(final_data["smoker"])
```

[22]:
```
final_data.head(2)
```

[22]:
```
  Customer ID                    name  year month  date  children   charges  \
0         Id1       Hawks, Ms.  Kelly  1968   Oct    12         0  63770.43
1         Id2  Lehner, Mr.  Matthew D  1977   Jun     8         0  62592.87

  Hospital tier City tier State ID    BMI  HBA1C  Heart Issues  \
0      tier - 1  tier - 3    R1013  47.41   7.47             0
1      tier - 2  tier - 3    R1013  30.36   5.77             0

   Any Transplants  Cancer history NumberOfMajorSurgeries  smoker
0                0               0       No major surgery       1
1                0               0       No major surgery       1
```

[23]:
```
# Handling ordinal features
```

[24]:
```
# We have two ordinal variables: Hospital Tier and City Tier
# We will define a function to remove the word "tier" and "-" from them
```

[25]:
```python
def ordinal(val):
    return int(val.replace("tier", "").replace(" ", "").replace("-", ""))
```

[26]:
```python
final_data["Hospital tier"] = final_data["Hospital tier"].map(ordinal)
final_data["City tier"] = final_data["City tier"].map(ordinal)
```

[27]:
```python
final_data.head(2)
```

[27]:
```
  Customer ID                   name  year month  date  children   charges  \
0         Id1       Hawks, Ms.  Kelly  1968   Oct    12         0  63770.43
1         Id2  Lehner, Mr.  Matthew D  1977   Jun     8         0  62592.87

   Hospital tier  City tier State ID    BMI  HBA1C  Heart Issues  \
0              1          3    R1013  47.41   7.47             0
1              2          3    R1013  30.36   5.77             0

   Any Transplants  Cancer history NumberOfMajorSurgeries  smoker
0                0               0       No major surgery       1
1                0               0       No major surgery       1
```

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.

[28]:
```python
final_data["State ID"].value_counts()
```

[28]:
```
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
```

[29]:
```python
# We can see that only R1011, R1012, and R1013 need to be considered from their
↪counts
```

[30]:
```
# Let us create dummies for State Id
```

[31]:
```
Dummies = pd.get_dummies(final_data["State ID"], prefix= "State_ID")
```

[32]:
```
Dummies.head(2)
```

[32]:
```
   State_ID_R1011  State_ID_R1012  State_ID_R1013  State_ID_R1014  \
0               0               0               1               0
1               0               0               1               0

   State_ID_R1015  State_ID_R1016  State_ID_R1017  State_ID_R1018  \
0               0               0               0               0
1               0               0               0               0

   State_ID_R1019  State_ID_R1020  State_ID_R1021  State_ID_R1022  \
0               0               0               0               0
1               0               0               0               0

   State_ID_R1023  State_ID_R1024  State_ID_R1025  State_ID_R1026
0               0               0               0               0
1               0               0               0               0
```

[33]:
```
# However we need only R1011, R1012, and R1013
```

[34]:
```
Dummy = Dummies[['State_ID_R1011','State_ID_R1012', 'State_ID_R1013']]
```

[35]:
```
Dummy.head(2)
```

[35]:
```
   State_ID_R1011  State_ID_R1012  State_ID_R1013
0               0               0               1
1               0               0               1
```

[36]:
```
# Let us now merge dummy to our data frame
```

[37]:
```
data = pd.concat([final_data, Dummy], axis=1)
```

[38]:
```
data.head()
```

[38]:
```
  Customer ID                    name  year month  date  children   charges  \
0         Id1        Hawks, Ms.  Kelly  1968   Oct    12         0  63770.43
1         Id2  Lehner, Mr.  Matthew D  1977   Jun     8         0  62592.87
3         Id4      Osborne, Ms.  Kelsey  1991   Jun     6         1  58571.07
4         Id5     Kadala, Ms.  Kristyn  1989   Jun    19         0  55135.40
5         Id6   Baker, Mr.  Russell B.  1962   Aug     4         0  52590.83

   Hospital tier  City tier State ID     BMI  HBA1C  Heart Issues  \
0              1          3    R1013  47.410   7.47             0
```

```
1               2           3     R1013   30.360    5.77              0
3               1           3     R1024   38.095    6.05              0
4               1           2     R1012   35.530    5.45              0
5               1           3     R1011   32.800    6.59              0

   Any Transplants  Cancer history NumberOfMajorSurgeries  smoker  \
0                0               0       No major surgery       1
1                0               0       No major surgery       1
3                0               0       No major surgery       1
4                0               0       No major surgery       1
5                0               0       No major surgery       1

   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
```

[39]:
```
# We are now in a position to drop the feature "State ID"
```

[40]:
```
data.drop(['State ID'], inplace=True, axis=1)
```

[41]:
```
data.head(2)
```

[41]:
```
  Customer ID                   name  year month  date  children   charges  \
0         Id1        Hawks, Ms.  Kelly  1968   Oct    12         0  63770.43
1         Id2  Lehner, Mr.  Matthew D  1977   Jun     8         0  62592.87

   Hospital tier  City tier    BMI  HBA1C  Heart Issues  Any Transplants  \
0              1          3  47.41   7.47             0                0
1              2          3  30.36   5.77             0                0

   Cancer history NumberOfMajorSurgeries  smoker  State_ID_R1011  \
0               0       No major surgery       1               0
1               0       No major surgery       1               0

   State_ID_R1012  State_ID_R1013
0               0               1
1               0               1
```

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

[42]:
```
data['NumberOfMajorSurgeries'].unique()
```

[42]:
```
array(['No major surgery', '3', '1', '2'], dtype=object)
```

[43]:
```
# "No major surgery" is a string and needs to be replaced
```

[44]:
```
data['NumberOfMajorSurgeries'] = data['NumberOfMajorSurgeries'].replace('No␣
↪major surgery', 0)
```

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

[46]:
```
data.head(2)
```

[46]:
```
  Customer ID                    name  year month  date  children   charges  \
0         Id1        Hawks, Ms.  Kelly  1968   Oct    12         0  63770.43
1         Id2  Lehner, Mr.  Matthew D  1977   Jun     8         0  62592.87

   Hospital tier  City tier    BMI  HBA1C  Heart Issues  Any Transplants  \
0              1          3  47.41   7.47             0                0
1              2          3  30.36   5.77             0                0

   Cancer history  NumberOfMajorSurgeries  smoker  State_ID_R1011  \
0               0                       0       1               0
1               0                       0       1               0

   State_ID_R1012  State_ID_R1013
0               0               1
1               0               1
```

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

[47]:
```
# Create a mapping of the month names to their corresponding numerical␣
↪representation
month_mapping = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,␣
↪'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}

# Convert the year, month, and date columns into a standard datetime format
data['dateofbirth'] = pd.to_datetime(data['year'].astype(str) + '-' +␣
↪data['month'].map(month_mapping).astype(str).str.zfill(2) + '-' +␣
↪data['date'].astype(str).str.zfill(2), errors='coerce')

# Drop the original year, month, and date columns
data = data.drop(columns=['year', 'month', 'date'])
```

[48]:
```
from datetime import datetime

def calculate_age(dateofbirth):
    today = datetime.today()
    age = today.year - dateofbirth.dt.year
    return age
```

```
data['Age'] = calculate_age(data['dateofbirth'])
```

[49]:
```
data.head(2)
```

[49]:
```
  Customer ID                    name  children   charges  Hospital tier  \
0         Id1        Hawks, Ms.  Kelly         0  63770.43              1
1         Id2  Lehner, Mr.  Matthew D         0  62592.87              2

   City tier    BMI  HBA1C  Heart Issues  Any Transplants  Cancer history  \
0          3  47.41   7.47             0                0               0
1          3  30.36   5.77             0                0               0

   NumberOfMajorSurgeries  smoker  State_ID_R1011  State_ID_R1012  \
0                       0       1               0               0
1                       0       1               0               0

   State_ID_R1013 dateofbirth  Age
0               1  1968-10-12   55
1               1  1977-06-08   46
```

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.

[50]:
```
# Let us check which all are the salutations used
```

[51]:
```
data['name'].str.extract(r'(\w+\.)').squeeze().unique()
```

[51]:
```
array(['Ms.', 'Mr.', 'Mrs.'], dtype=object)
```

[52]:
```
# Let us create a mapping for the saluations to create a distinction for gender
```

[53]:
```
salutation = {
    'Mr.': 'Male',
    'Mrs.': 'Female',
    'Ms.': 'Female'}
```

[54]:
```
# Let us now apply the map to our data frame to create a new feature called␣
↪gender
```

[55]:
```
data['gender'] = data['name'].str.extract(r'(\w+\.)').squeeze().map(salutation)
```

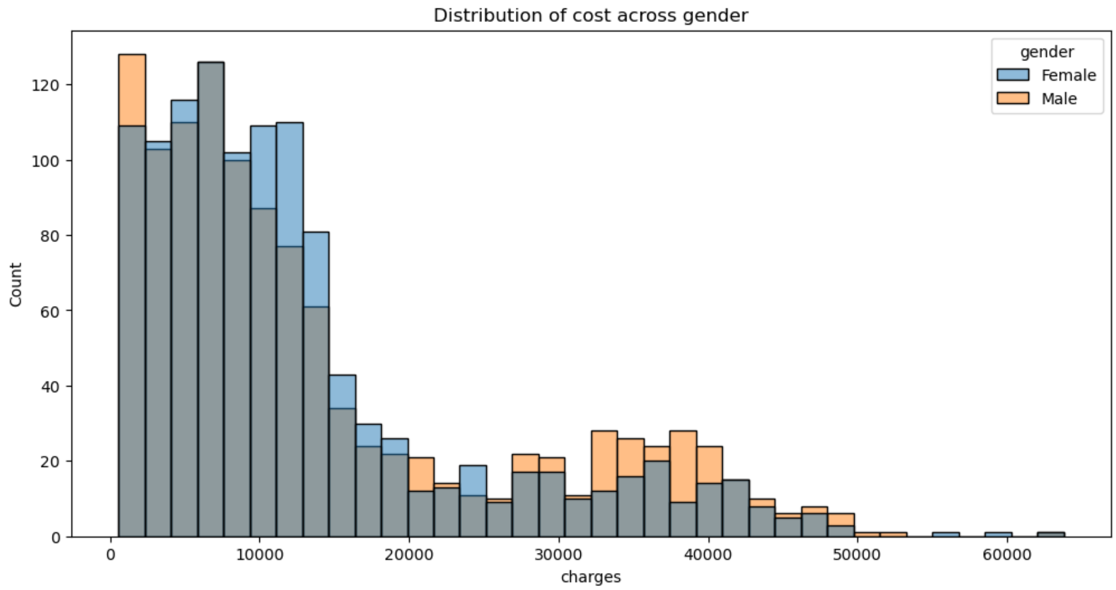
[56]:
```
data['gender'].unique()
```

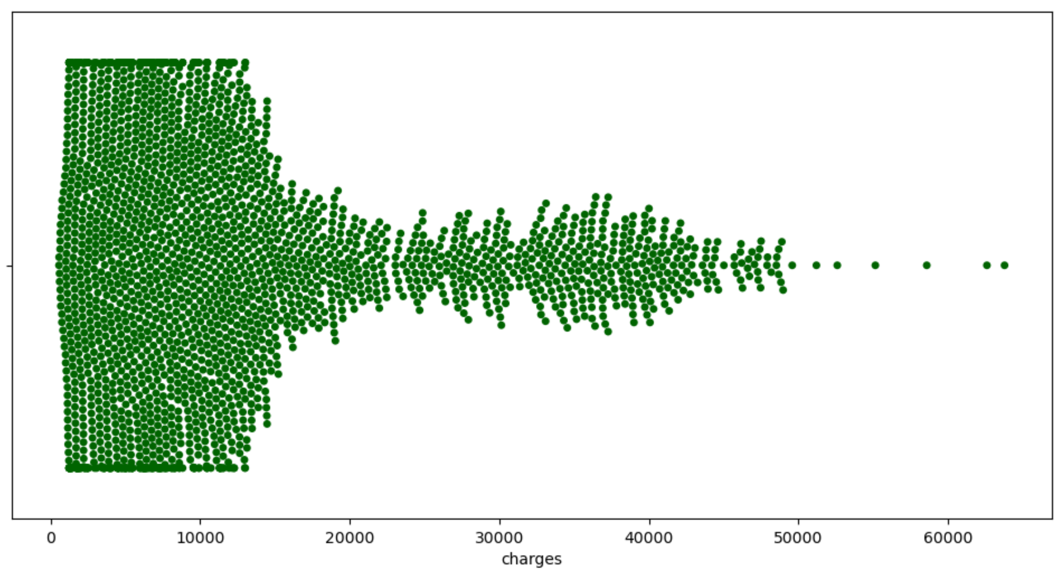
[56]:
```
array(['Female', 'Male'], dtype=object)
```

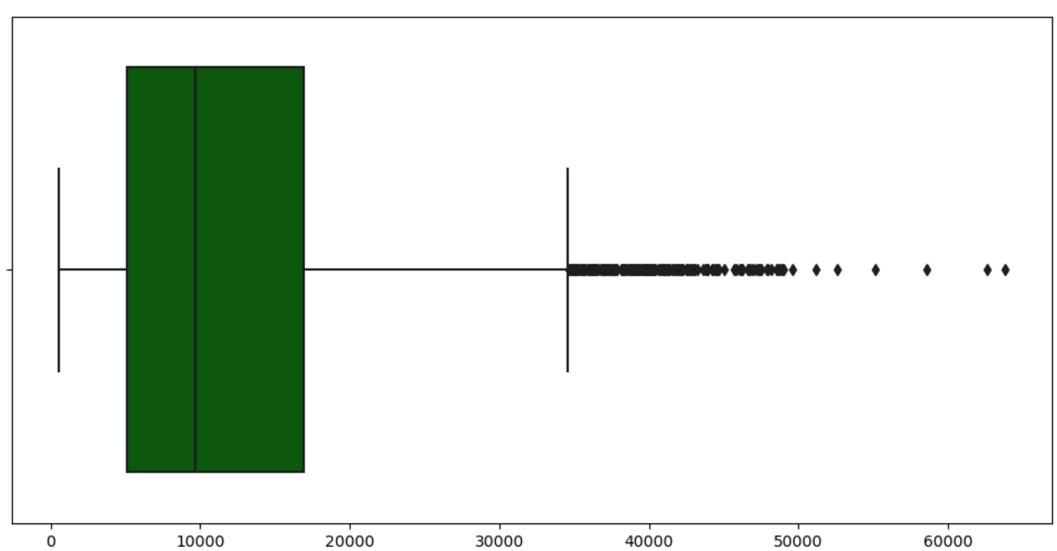
9. You should also visualize the distribution of costs using a histogram, box and whisker plot, and swarm plot.

[57]:
```
fig, ax = plt.subplots(ncols=1,nrows=3,figsize=(12,20))
ax[0] = sns.histplot(data =data,x='charges',ax=␣
 ↪ax[0],color='darkgreen',hue='gender')
ax[0].set_title("Distribution of cost across gender")
ax[1] = sns.swarmplot(data =data,x='charges',ax=␣
 ↪ax[1],color='darkgreen',hue='gender')
ax[2] = sns.boxplot(data=data,x= 'charges',ax=␣
 ↪ax[2],color='darkgreen',hue='gender')
```

```
C:\Users\akhil\anaconda3\lib\site-packages\seaborn\categorical.py:1296:
UserWarning: 13.2% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

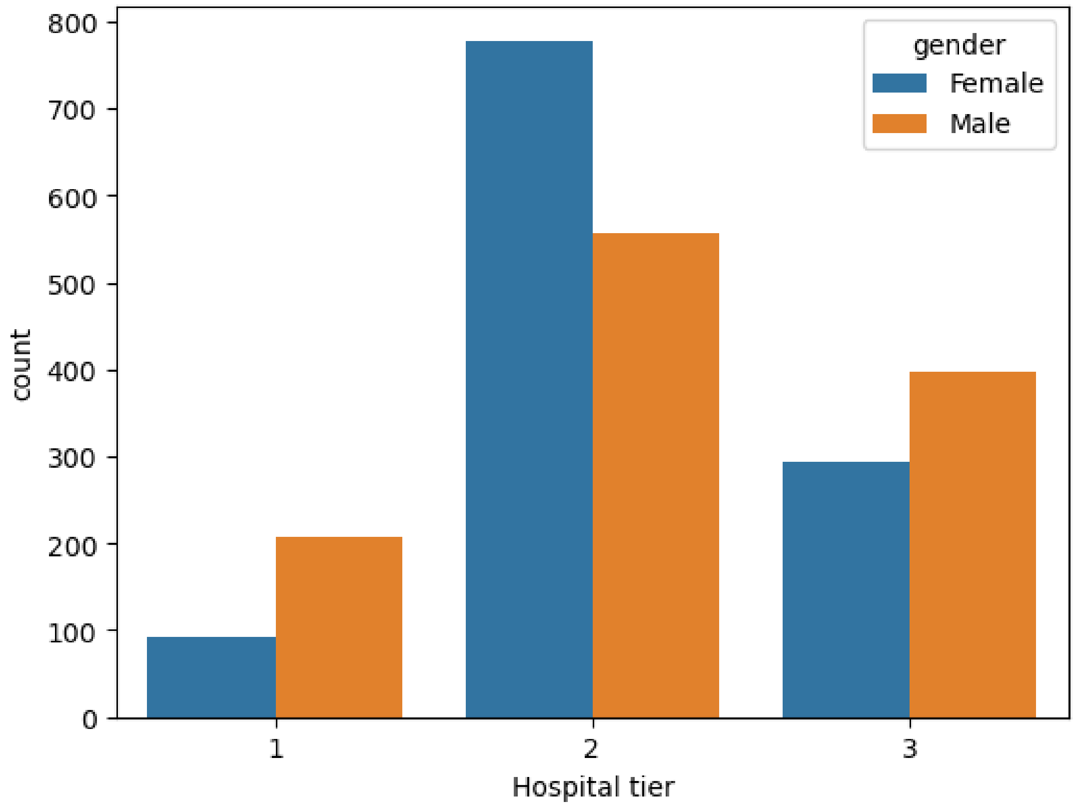

Distribution of cost across gender
gender
Female
Male
Count
0
20
40
60
80
100
120
0
10000
20000
30000
40000
50000
60000
charges


0
10000
20000
30000
40000
50000
60000
charges


0
10000
20000
30000
40000
50000
60000
charges

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

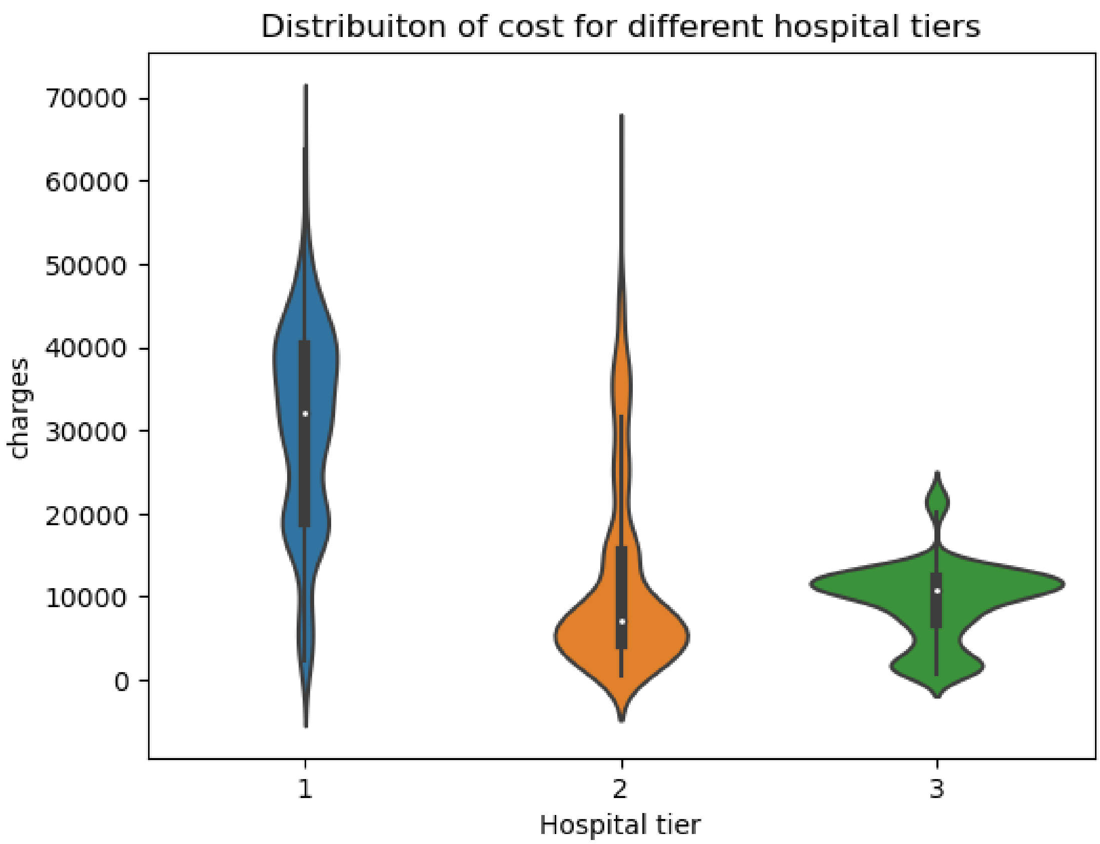
[58]:
```
sns.countplot(data = data, x='Hospital tier', hue= 'gender')
plt.show()
```



[59]:
```
# From the above plot it is clear that the number of female in the tier 1 and 3␣
↪is half of the male
# In tier 2 hospitals, the female count is slightly more compared to male
```

[60]:
```
sns.violinplot(data=data,x='Hospital tier',y='charges')
plt.title('Distribuiton of cost for different hospital tiers')
```

[60]:
```
Text(0.5, 1.0, 'Distribuiton of cost for different hospital tiers')
```



[61]:
```
# Tier 1 hospitals charge more on patients compared to hospitals in tier 2 and␣
↪tier 3
```

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

[62]:
```
Median_Hospitalization_Cost = data.groupby('Hospital tier')['charges'].median()
Tier = ['tier - 1','tier - 2','tier - 3']
```

[63]:
```
df = pd.DataFrame({
    'Hospital Tier':Tier,
    'Median Hospitalization Cost':Median_Hospitalization_Cost
})
```

[64]:
```
import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(go.Scatterpolar(
      r=df['Median Hospitalization Cost'],
      theta=df['Hospital Tier'],
      fill='toself',
      name='Median Hospitalization Cost'
```

```
))
# Add text annotations for the median values
fig.add_trace(go.Scatterpolar(
      r=df['Median Hospitalization Cost'],
      theta=df['Hospital Tier'].unique(),
      text=round(df['Median Hospitalization Cost'],1),
      mode='text',
      hoverinfo='none'
))

# Set the title of the chart and axis properties
fig.update_layout(
    title='Median Hospitalization Cost by Tier of Hospital',
    polar=dict(
        radialaxis=dict(
            visible=True,
            range=[0, max(df['Median Hospitalization Cost'])],
            tickfont=dict(size=14),
            ticksuffix='$'
        ),
        angularaxis=dict(
            visible=True,
            tickfont=dict(size=14),
            rotation=270
        )
    ),template='plotly_dark'
)

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

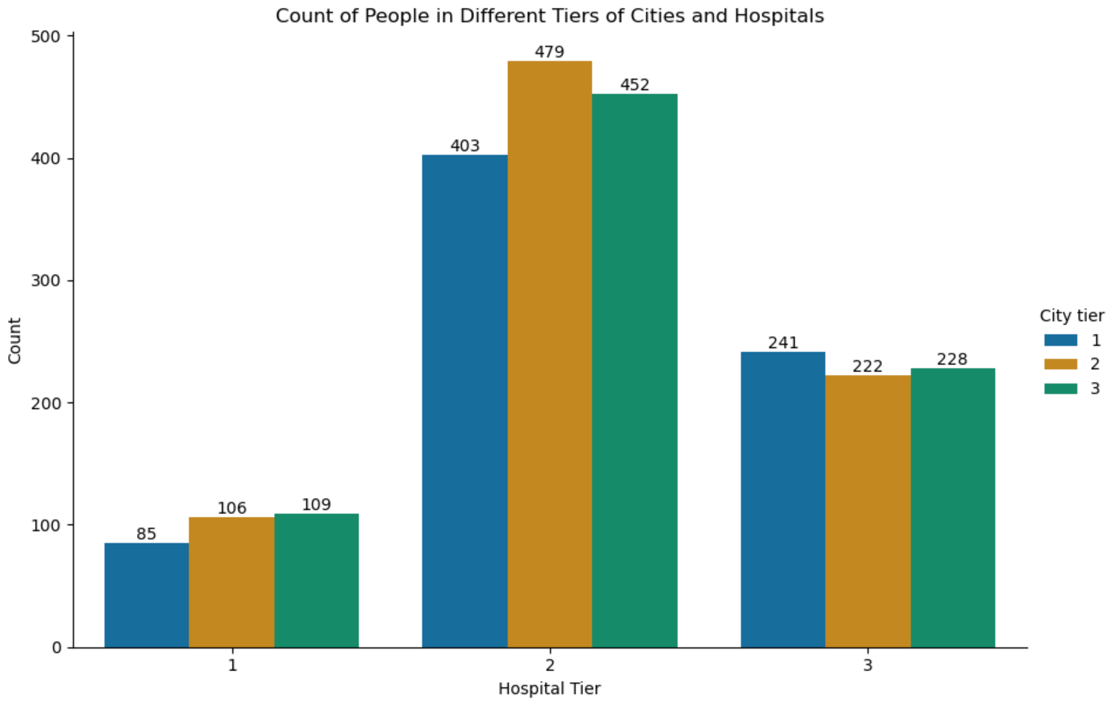
12. Create a frequency table and a stacked bar chart to visualize the count of people in the different tiers of cities and hospitals

[65]:
```
frequency_table = pd.crosstab(data['Hospital tier'], [data['City tier']],␣
↪rownames=['Hospital Tier'], colnames=['City Tier'])
frequency_table
```

[65]:
```
City Tier        1    2    3
Hospital Tier
1               85  106  109
2              403  479  452
3              241  222  228
```

[66]:
```
ax = sns.catplot(x='Hospital tier', hue='City tier', kind='count', data=data,␣
↪height=6, aspect=1.5,palette='colorblind')
plt.xlabel('Hospital Tier')
```

```
plt.ylabel('Count')
plt.title('Count of People in Different Tiers of Cities and Hospitals')
for container in ax.ax.containers:
    ax.ax.bar_label(container)
```



13. Test the following null hypotheses:

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

[67]:
```
from scipy.stats import kruskal
```

[68]:
```
data1 = data[data["Hospital tier"]==1].charges.mean()
data2 = data[data["Hospital tier"]==2].charges.mean()
data3 = data[data["Hospital tier"]==3].charges.mean()
```

[69]:
```
stat, p = kruskal([data1], [data2], [data3])
print('stat=%.3f, p=%.3f' % (stat, p))
if p < 0.05:
    print('Hospitalization costs are different for different tiers of␣
 ↪hospitals')
else:
    print('Hospitalization costs are same for different tiers of hospitals')
```

```
stat=2.000, p=0.368
Hospitalization costs are same for different tiers of hospitals
```

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

[70]:
```python
data1 = data[data["City tier"]==1].charges.mean()
data2 = data[data["City tier"]==2].charges.mean()
data3 = data[data["City tier"]==3].charges.mean()
```

[71]:
```python
stat, p = kruskal([data1], [data2], [data3])
print('stat=%.3f, p=%.3f' % (stat, p))
if p < 0.05:
    print('Hospitalization costs are different for different tiers of cities')
else:
    print('Hospitalization costs are same for different tiers of cities')
```

```
stat=2.000, p=0.368
Hospitalization costs are same for different tiers of cities
```

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

[72]:
```python
data1 = data[data["smoker"]==1].charges.mean()
data2 = data[data["smoker"]==0].charges.mean()
```

[73]:
```python
stat, p = kruskal([data1], [data2])
print('stat=%.3f, p=%.3f' % (stat, p))
if p < 0.05:
    print('Hospitalization costs are different for smokers and non-smokers')
else:
    print('Hospitalization costs are same for smokers and non-smokers')
```

```
stat=1.000, p=0.317
Hospitalization costs are same for smokers and non-smokers
```

d. Smoking and heart issues are independent

[74]:
```python
from scipy.stats import chi2_contingency
```

[75]:
```python
# Chi-squared test of independence for smoking and heart issues
smoking_heart = pd.crosstab(data['smoker'], data['Heart Issues'])
chi2_stat, p_value, dof, expected = chi2_contingency(smoking_heart)
print(f"p-value for smoking and heart issues chi-squared test: {p_value:.4f}")
if p < 0.05:
    print('Smoking and heart issues are related')
else:
    print('Smoking and heart issues are independent')
```

```
p-value for smoking and heart issues chi-squared test: 0.7695
Smoking and heart issues are independent
```

Week 2

Machine Learning

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

[76]:
```
data.columns
```

[76]:
```
Index(['Customer ID', 'name', 'children', 'charges', 'Hospital tier',
       'City tier', 'BMI', 'HBA1C', 'Heart Issues', 'Any Transplants',
       'Cancer history', 'NumberOfMajorSurgeries', 'smoker', 'State_ID_R1011',
       'State_ID_R1012', 'State_ID_R1013', 'dateofbirth', 'Age', 'gender'],
      dtype='object')
```

[77]:
```
# Let us drop the columns that are not useful to model building.
data.drop(["Customer ID","dateofbirth","name"],inplace=True, axis=1)
```

[78]:
```
data['gender'] = le.fit_transform(data['gender'])
```

[79]:
```
data.columns
```

[79]:
```
Index(['children', 'charges', 'Hospital tier', 'City tier', 'BMI', 'HBA1C',
       'Heart Issues', 'Any Transplants', 'Cancer history',
       'NumberOfMajorSurgeries', 'smoker', 'State_ID_R1011', 'State_ID_R1012',
       'State_ID_R1013', 'Age', 'gender'],
      dtype='object')
```

[80]:
```
correlation = data.corr()
correlation
```

[80]:
```
                        children   charges  Hospital tier  City tier  \
children                1.000000  0.055901      -0.052438  -0.015760
charges                 0.055901  1.000000      -0.446687   0.035300
Hospital tier          -0.052438 -0.446687       1.000000  -0.039755
City tier              -0.015760  0.035300      -0.039755   1.000000
BMI                    -0.005339  0.346730      -0.104771   0.038123
HBA1C                  -0.101379  0.139697       0.057855  -0.005404
Heart Issues            0.023984  0.049299       0.053376   0.023152
Any Transplants        -0.142040 -0.127028       0.011729   0.002970
Cancer history         -0.027880 -0.022522      -0.021429  -0.018639
NumberOfMajorSurgeries -0.113161  0.053308       0.033230   0.027937
smoker                  0.017713  0.838462      -0.474077   0.032034
State_ID_R1011          0.011666  0.286956      -0.114685   0.036049
State_ID_R1012          0.005247 -0.074636       0.020272  -0.018253
State_ID_R1013         -0.013834 -0.150634       0.002455   0.002766
Age                    -0.005457  0.304395       0.133771  -0.008070
gender                  0.014332  0.060156      -0.006927   0.059716

                             BMI     HBA1C  Heart Issues  Any Transplants  \
children               -0.005339 -0.101379      0.023984        -0.142040
```

charges	0.346730	0.139697	0.049299	-0.127028
Hospital tier	-0.104771	0.057855	0.053376	0.011729
City tier	0.038123	-0.005404	0.023152	0.002970
BMI	1.000000	-0.006920	0.017129	0.015893
HBA1C	-0.006920	1.000000	0.007699	-0.159855
Heart Issues	0.017129	0.007699	1.000000	-0.140269
Any Transplants	0.015893	-0.159855	-0.140269	1.000000
Cancer history	-0.020235	-0.170921	0.111190	-0.114677
NumberOfMajorSurgeries	0.018851	-0.091594	0.206147	0.158593
smoker	0.107126	0.007257	-0.007159	-0.025101
State_ID_R1011	0.115671	0.015525	0.005852	-0.058553
State_ID_R1012	0.017939	-0.019513	0.021770	-0.066453
State_ID_R1013	-0.208744	0.033453	-0.027967	0.064563
Age	0.049260	0.460558	0.192273	-0.381084
gender	0.015239	-0.023890	-0.001778	0.004141

	Cancer history	NumberOfMajorSurgeries	smoker \
children	-0.027880	-0.113161	0.017713
charges	-0.022522	0.053308	0.838462
Hospital tier	-0.021429	0.033230	-0.474077
City tier	-0.018639	0.027937	0.032034
BMI	-0.020235	0.018851	0.107126
HBA1C	-0.170921	-0.091594	0.007257
Heart Issues	0.111190	0.206147	-0.007159
Any Transplants	-0.114677	0.158593	-0.025101
Cancer history	1.000000	0.204208	0.006415
NumberOfMajorSurgeries	0.204208	1.000000	0.017199
smoker	0.006415	0.017199	1.000000
State_ID_R1011	0.011919	0.000208	0.336112
State_ID_R1012	0.058222	-0.002098	-0.106998
State_ID_R1013	-0.066475	-0.002056	-0.094547
Age	-0.101073	0.151442	0.011939
gender	-0.013983	0.003842	0.083612

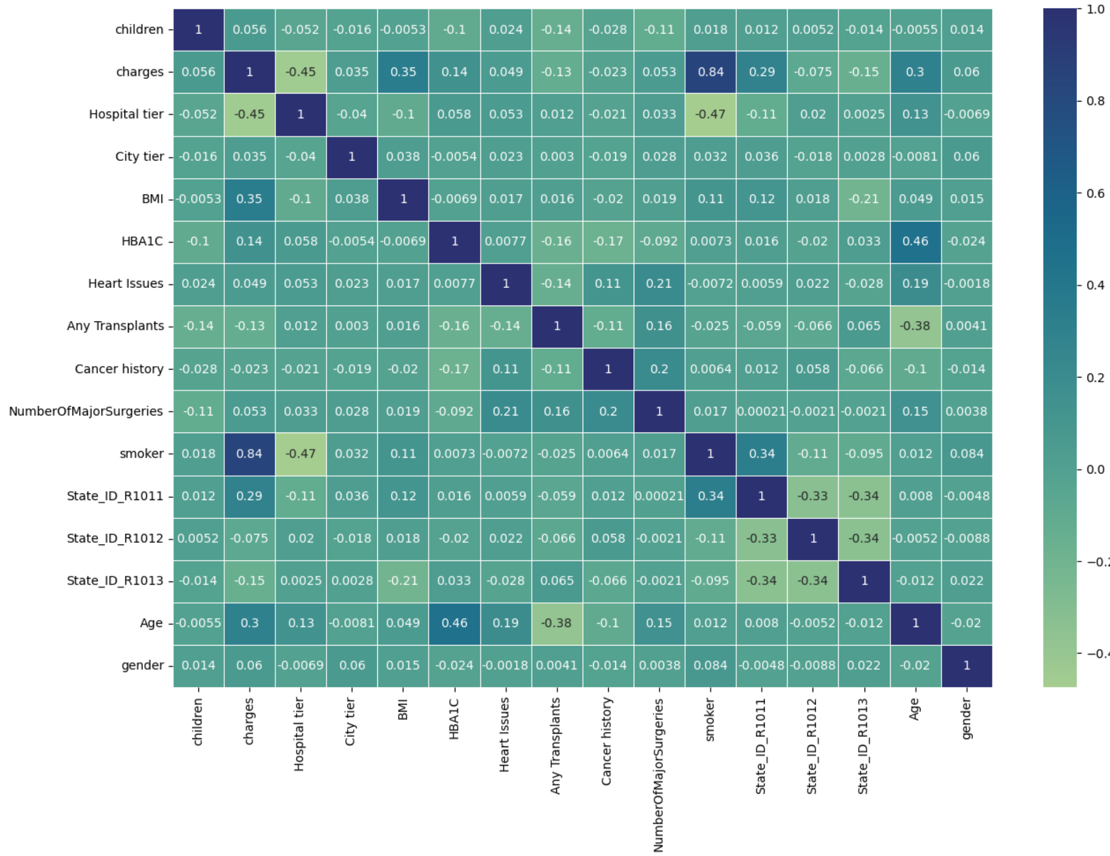
	State_ID_R1011	State_ID_R1012	State_ID_R1013 \
children	0.011666	0.005247	-0.013834
charges	0.286956	-0.074636	-0.150634
Hospital tier	-0.114685	0.020272	0.002455
City tier	0.036049	-0.018253	0.002766
BMI	0.115671	0.017939	-0.208744
HBA1C	0.015525	-0.019513	0.033453
Heart Issues	0.005852	0.021770	-0.027967
Any Transplants	-0.058553	-0.066453	0.064563
Cancer history	0.011919	0.058222	-0.066475
NumberOfMajorSurgeries	0.000208	-0.002098	-0.002056
smoker	0.336112	-0.106998	-0.094547
State_ID_R1011	1.000000	-0.327054	-0.341085

```
State_ID_R1012                -0.327054        1.000000        -0.340296
State_ID_R1013                -0.341085       -0.340296         1.000000
Age                            0.008022       -0.005229        -0.011926
gender                        -0.004754       -0.008758         0.021824

                             Age    gender
children               -0.005457  0.014332
charges                 0.304395  0.060156
Hospital tier           0.133771 -0.006927
City tier              -0.008070  0.059716
BMI                     0.049260  0.015239
HBA1C                   0.460558 -0.023890
Heart Issues            0.192273 -0.001778
Any Transplants        -0.381084  0.004141
Cancer history         -0.101073 -0.013983
NumberOfMajorSurgeries  0.151442  0.003842
smoker                  0.011939  0.083612
State_ID_R1011          0.008022 -0.004754
State_ID_R1012         -0.005229 -0.008758
State_ID_R1013         -0.011926  0.021824
Age                     1.000000 -0.020197
gender                 -0.020197  1.000000
```

[81]:

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



[82]:
```
# We can see that smoker and charges have the highest correlation of 0.84
```

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

[83]:
```
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import RandomForestRegressor
```

[84]:
```
x = data.drop(['charges'], axis=1)
y = data['charges']
```

[85]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=42)
```

[86]:
```
f1 = ColumnTransformer([
    ('data_transform', StandardScaler(), slice(0,15))
])
```

[87]:
```
f2 = SGDRegressor()
```

[88]:
```
pipe = Pipeline([
    ('f1', f1),
    ('f2', f2)
])
```

[89]:
```
pipe.fit(x_train, y_train)
```

[89]:
```
Pipeline(steps=[('f1',
                 ColumnTransformer(transformers=[('data_transform',
                                                  StandardScaler(),
                                                  slice(0, 15, None))])),
                ('f2', SGDRegressor())])
```

[90]:
```
y_pred = pipe.predict(x_test)
```

[91]:
```
pipe.named_steps['f2'].coef_
```

[91]:
```
array([  496.39059785, -1087.85935338,   151.70791984,  2825.38944847,
         130.89565771,   -90.37060878,    84.49003429,    55.54809943,
          16.93428511,  9030.35589716,  -377.65979651,  -161.34898718,
        -447.56653864,  3453.43962836,   -38.68989757])
```

[92]:
```
from sklearn.metrics import r2_score
r2_score(y_pred, y_test)
```

[92]:
```
0.8181469329379886
```

[93]:
```
from sklearn.metrics import mean_squared_error
print(np.sqrt(mean_squared_error(y_pred, y_test)))
```

```
4601.210844223612
```

[94]:
```
# Parameter tuning using gridsearch
```

[95]:
```
param_grid = {
   'loss': ['squared_loss', 'huber', 'epsilon_insensitive','r2_score'],
    'penalty': ['l1', 'l2', 'elasticnet'],
    'alpha': [0.00001, 0.0001, 0.001, 0.01, 0.05, 0.1],
    'learning_rate': ['constant', 'optimal', 'invscaling', 'adaptive']
}
```

[96]:
```
folds = 5
grid_search = GridSearchCV(f2, param_grid, cv=folds, n_jobs=-1)
grid_search.fit(x_train, y_train)
```

```
C:\Users\akhil\anaconda3\lib\site-
packages\sklearn\model_selection\_validation.py:372: FitFailedWarning:


360 fits failed out of a total of 1440.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
360 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\akhil\anaconda3\lib\site-
packages\sklearn\model_selection\_validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\akhil\anaconda3\lib\site-
packages\sklearn\linear_model\_stochastic_gradient.py", line 1537, in fit
    return self._fit(
  File "C:\Users\akhil\anaconda3\lib\site-
packages\sklearn\linear_model\_stochastic_gradient.py", line 1472, in _fit
    self._validate_params()
  File "C:\Users\akhil\anaconda3\lib\site-
packages\sklearn\linear_model\_stochastic_gradient.py", line 162, in
_validate_params
    raise ValueError("The loss %s is not supported. " % self.loss)
ValueError: The loss r2_score is not supported.


C:\Users\akhil\anaconda3\lib\site-
packages\sklearn\model_selection\_search.py:969: UserWarning:

One or more of the test scores are non-finite: [-2.52095254e+18 -1.78996503e+18
-8.03914315e+17  5.48517311e-02
  5.05072989e-02  4.71520342e-02  1.53939632e-01  1.65509281e-01
  1.58460231e-01             nan             nan             nan
 -6.55198858e+20 -1.85513745e+20 -3.49209739e+20  5.90877490e-01
  2.54158887e-01  2.66035652e-01  8.39926185e-01  7.28694754e-01
  7.41459217e-01             nan             nan             nan
 -1.01984309e+13 -3.77959835e+11 -9.26352486e+11 -6.58599331e-03
 -6.64466429e-03 -6.62737046e-03  1.64733379e-02  1.63290167e-02
  1.66539224e-02             nan             nan             nan
```

```
-4.70764111e+11 -3.72996248e+11 -1.32529177e+11  5.25447697e-02
 4.76963439e-02  5.02236290e-02  1.92139221e-01  1.54724026e-01
 1.46818885e-01             nan             nan             nan
-2.61482119e+18 -1.80278264e+18 -2.58195320e+18  4.56376783e-02
 1.45236549e-02  1.24890032e-02  1.51725188e-01  1.21563020e-01
 1.34602332e-01             nan             nan             nan
-9.62679975e+18 -1.10456034e+19 -3.37717953e+18  1.74219773e-01
 2.62966064e-02  4.95142408e-02  6.41147398e-01  2.97040069e-01
 2.91605071e-01             nan             nan             nan
-8.18897108e+10 -9.52711615e+12 -6.17828648e+12 -6.57097798e-03
-7.70689790e-03 -7.54164353e-03  1.64040510e-02  1.58559552e-02
 1.56958286e-02             nan             nan             nan
-1.88519609e+11 -2.31443386e+10 -5.21970340e+10  4.67504022e-02
 1.68686828e-02  1.79168864e-02  1.68684613e-01  1.29723836e-01
 1.26017203e-01             nan             nan             nan
-1.24281056e+18 -1.21858375e+18 -2.09427304e+18  5.18328989e-02
-1.72372695e-02 -1.59098264e-02  1.72604102e-01  2.54383447e-02
 2.21570756e-02             nan             nan             nan
-5.37070254e+16 -5.83902078e+16 -4.34340554e+16  2.83503832e-02
-1.35876809e-02 -1.11402161e-02  2.02718779e-01  2.99200952e-02
 4.57174591e-02             nan             nan             nan
-4.98219412e+13 -2.61174206e+11 -9.04973431e+09 -6.83551043e-03
-1.93729732e-02 -1.80540572e-02  1.64707105e-02  8.73462018e-03
 9.47710412e-03             nan             nan             nan
-4.44248110e+10 -1.40948307e+12 -1.65559482e+11  4.92746741e-02
-1.75252690e-02 -1.51868147e-02  1.95849663e-01  2.53274676e-02
 2.81936015e-02             nan             nan             nan
-1.38792723e+18 -3.27083330e+18 -1.84537508e+18  1.68665818e-02
-1.37081742e-01 -1.17523905e-01  1.41635523e-01 -1.73258007e-02
-1.31913968e-02             nan             nan             nan
-2.40821456e+12 -1.98531132e+12 -2.00089602e+13 -4.95818656e-03
-1.37328954e-01 -1.16932217e-01  3.43511569e-02 -1.68293384e-02
-1.57351827e-02             nan             nan             nan
-2.24404751e+12 -2.75993084e+11 -4.63261216e+11 -8.71283610e-03
-1.37564544e-01 -1.17585854e-01  1.55226953e-02 -1.82422138e-02
-1.59073042e-02             nan             nan             nan
-1.64914693e+11 -1.60848950e+11 -4.91786752e+11  1.59928775e-02
-1.37157705e-01 -1.16517624e-01  1.57614183e-01 -1.52956808e-02
-1.30373048e-02             nan             nan             nan
-3.03824609e+18 -2.00291963e+18 -1.07222511e+18 -1.41372649e-02
-5.41075722e-01 -4.75645539e-01  1.20362439e-01 -7.98650757e-02
-6.24120824e-02             nan             nan             nan
-6.64008661e+12 -3.13278862e+13 -2.04188528e+13 -1.42496983e-02
-5.41984662e-01 -4.76179806e-01 -6.57254012e-04 -6.60379062e-02
-5.75756846e-02             nan             nan             nan
-8.13985776e+11 -2.29420340e+11 -8.19206868e+11 -1.48571819e-02
-5.42604847e-01 -4.76694603e-01  1.08574966e-02 -6.63828090e-02
-5.66733026e-02             nan             nan             nan
```

```
-5.81296577e+10 -6.25369880e+09 -2.66206434e+10 -1.45171866e-02
-5.41121298e-01 -4.75604745e-01  1.13074180e-01 -6.52720995e-02
-5.69027630e-02             nan             nan             nan
-2.23220787e+18 -3.65661354e+18 -2.47993498e+18 -1.93911433e-02
-8.28155351e-01 -7.68244274e-01  4.34230976e-02 -1.21430194e-01
-1.22186843e-01             nan             nan             nan
-1.25948158e+13 -9.20751680e+12 -2.06734457e+13 -2.19818076e-02
-8.28982422e-01 -7.70553370e-01 -5.33645391e-03 -1.37264078e-01
-1.17511900e-01             nan             nan             nan
-3.36992350e+11 -3.81625073e+11 -1.46610673e+11 -1.92208934e-02
-8.29080407e-01 -7.70723038e-01  5.65867288e-03 -1.36456744e-01
-1.16962825e-01             nan             nan             nan
-5.29190607e+10 -5.36048411e+10 -3.05259513e+10 -1.98817906e-02
-8.27144508e-01 -7.68615732e-01  6.85371023e-02 -1.36900133e-01
-1.16449706e-01             nan             nan             nan]
```

[96]:
```
GridSearchCV(cv=5, estimator=SGDRegressor(), n_jobs=-1,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.05, 0.1],
                         'learning_rate': ['constant', 'optimal', 'invscaling',
                                           'adaptive'],
                         'loss': ['squared_loss', 'huber',
                                  'epsilon_insensitive', 'r2_score'],
                         'penalty': ['l1', 'l2', 'elasticnet']})
```

[97]:
```
print(grid_search.best_params_)
```

```
{'alpha': 1e-05, 'learning_rate': 'optimal', 'loss': 'epsilon_insensitive',
'penalty': 'l1'}
```

[98]:
```
sgd = SGDRegressor(alpha = 0.00001, learning_rate='optimal', loss=␣
 ↪'epsilon_insensitive', penalty='l1')
sgd.fit(x_train, y_train)
```

[98]:
```
SGDRegressor(alpha=1e-05, learning_rate='optimal', loss='epsilon_insensitive',
             penalty='l1')
```

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

[100]:
```
r2_score(y_test, y_sgd)
```

[100]:
```
0.8445741936484602
```

[101]:
```
print(np.sqrt(mean_squared_error(y_test, y_sgd)))
```

```
4657.940360838209
```

[102]:
```
# We can see that by using grid search, r2 score has reduced and also mean
↪square error also has reduced
```

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

Random Forest Regressor

[103]:
```
rf = RandomForestRegressor()
rf.fit(x_train, y_train)
rf_pred = rf.predict(x_test)
```

[104]:
```
r2_score(rf_pred, y_test)
```

[104]:
```
0.8895953748084797
```

[105]:
```
print(np.sqrt(mean_squared_error(rf_pred, y_test)))
```

```
3766.227505963301
```

[106]:
```
from sklearn.model_selection import cross_val_score
cv = KFold(n_splits=5, shuffle=True, random_state=42)
```

[107]:
```
score = cross_val_score(rf, x, y, cv=cv, scoring='r2')
```

[108]:
```
mean_score = np.mean(score)
mean_score
```

[108]:
```
0.9126379298989933
```

[109]:
```
param_grid = {
    'n_estimators':[100, 200, 300],
    'max_depth' : [5, 10, 15],
    'min_samples_split' : [2, 5, 10],
    'min_samples_leaf' : [1, 2, 4]
}
```

[110]:
```
grid_search = GridSearchCV(rf, param_grid=param_grid, cv=5)
```

[111]:
```
grid_search.fit(x_train, y_train)
```

[111]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid={'max_depth': [5, 10, 15],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [100, 200, 300]})
```

[112]:
```
print(grid_search.best_params_)
```

```
{'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
300}
```

[113]:
```
rf_best = RandomForestRegressor(max_depth= 15, min_samples_leaf= 2,␣
 ↪min_samples_split= 5, n_estimators= 100)
```

[114]:
```
rf_best.fit(x_train, y_train)
```

[114]:
```
RandomForestRegressor(max_depth=15, min_samples_leaf=2, min_samples_split=5)
```

[115]:
```
rf_best_pred = rf_best.predict(x_test)
```

[116]:
```
print(r2_score(y_test, rf_best_pred))
```

```
0.9022369302459861
```

[117]:
```
print(np.sqrt(mean_squared_error(y_test, rf_best_pred)))
```

```
3694.1946413887204
```

[118]:
```
# Determine the variable importance scores, and identify the redundant variables
```

[119]:
```
rf_imp = pd.Series(rf_best.feature_importances_, index=x_train.columns)
rf_imp
```

[119]:
```
children                 0.013292
Hospital tier            0.020013
City tier                0.001680
BMI                      0.122517
HBA1C                    0.010545
Heart Issues             0.000859
Any Transplants          0.000270
Cancer history           0.000671
NumberOfMajorSurgeries   0.001300
smoker                   0.723324
State_ID_R1011           0.005889
State_ID_R1012           0.001508
State_ID_R1013           0.006373
Age                      0.089485
gender                   0.002274
dtype: float64
```

XGD Regressor

[120]:
```
conda install -c anaconda py-xgboost
```

```
Collecting package metadata (current_repodata.json): …working… done
Note: you may need to restart the kernel to use updated packages.
```

```
Solving environment: …working… done

# All requested packages already installed.
```

[121]:
```python
from xgboost import XGBRegressor
xg = XGBRegressor()
xg.fit(x_train, y_train)
```

[121]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, …)
```

[122]:
```python
y_xg = xg.predict(x_test)
```

[123]:
```python
r2_score(y_test, y_xg)
```

[123]:
```
0.8859356894082085
```

[124]:
```python
print(np.sqrt(mean_squared_error(y_test, y_xg)))
```

```
3990.3156272834885
```

[125]:
```python
cross_val_score(xg, x_train, y_train, cv=5, scoring='r2').mean()
```

[125]:
```
0.9053438668324592
```

[126]:
```python
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.5]
}
```

[127]:
```python
grid = GridSearchCV(xg, param_grid = param_grid, cv=5)
grid.fit(x_train, y_train)
```

[127]:
```
GridSearchCV(cv=5,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    callbacks=None, colsample_bylevel=None,
```

```
                                      colsample_bynode=None,
                                      colsample_bytree=None,
                                      early_stopping_rounds=None,
                                      enable_categorical=False, eval_metric=None,
                                      feature_types=None, gamma=None, gpu_id=None,
                                      grow_policy=None, importance_type=None,
                                      interaction_constraints=None,
                                      learning_rate=None, max_bin=None,
                                      max_cat_threshold=None,
                                      max_cat_to_onehot=None, max_delta_step=None,
                                      max_depth=None, max_leaves=None,
                                      min_child_weight=None, missing=nan,
                                      monotone_constraints=None, n_estimators=100,
                                      n_jobs=None, num_parallel_tree=None,
                                      predictor=None, random_state=None, …),
             param_grid={'learning_rate': [0.01, 0.1, 0.5],
                         'max_depth': [3, 4, 5],
                         'n_estimators': [50, 100, 150]})
```

[128]:
```python
print(grid.best_params_)
```

```
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 50}
```

[129]:
```python
xg_best = XGBRegressor(learning_rate= 0.1, max_depth= 5, n_estimators=50)
```

[130]:
```python
xg_best.fit(x_train, y_train)
```

[130]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=5, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=50, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, …)
```

[131]:
```python
xg_predict = xg_best.predict(x_test)
```

[132]:
```python
r2_score(y_test, xg_predict)
```

[132]:
```
0.9010200107510296
```

[133]:
```python
print(np.sqrt(mean_squared_error(y_test, xg_predict)))
```

```
3717.1155366635785
```

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

[134]:
```
import datetime as dt
```

[135]:
```
# First we need to calculate the age of the person.
date = str(19881228)
date1 = pd.to_datetime(date, format = "%Y%m%d")
```

[136]:
```
current_date = dt.datetime.now()
current_date
```

[136]:
```
datetime.datetime(2023, 4, 8, 14, 18, 41, 787848)
```

[137]:
```
age = (current_date - date1)
age
```

[137]:
```
Timedelta('12519 days 14:18:41.787848')
```

[138]:
```
age = int(12421/365)
age
```

[138]:
```
34
```

[139]:
```
# 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)
```

[139]:
```
29.41
```

[140]:
```
# Training data
```

[141]:
```
list = [[2, 1, 1, 29.41, 5.8, 0, 0, 1, 0, 1, 1, 0, 0, 34, 0]]
```

[142]:
```
training_list = x_train.columns.tolist()
```

[143]:
```
df = pd.DataFrame(list, columns=training_list)
df
```

[143]:
```
   children  Hospital tier  City tier    BMI  HBA1C  Heart Issues  \
0         2              1          1  29.41    5.8             0
```

```
   Any Transplants  Cancer history  NumberOfMajorSurgeries  smoker  \
0                0               1                       0       1

   State_ID_R1011  State_ID_R1012  State_ID_R1013  Age  gender
0               1               0               0   34       0
```

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

[144]:
```
Hospitalization_cost = []
```

[145]:
```
#Predicting hospitalization cost through SGDRegressor
Cost1 = sgd.predict(df)
Hospitalization_cost.append(Cost1)
```

[146]:
```
# Predicting hospitalization cost through Random Forest
Cost2 = rf_best.predict(df)
Hospitalization_cost.append(Cost2)
```

[147]:
```
# Predicting hospitalization cost through XGBoost
Cost3 = xg_best.predict(df)
Hospitalization_cost.append(Cost3)
```

[148]:
```
Hospitalization_cost
```

[148]:
```
[array([31739.84793002]),
 array([24660.41141508]),
 array([23995.322], dtype=float32)]
```

[149]:
```
avg_cost = np.mean(Hospitalization_cost)
avg_cost
```

[149]:
```
26798.52720357569
```

[ ]:
```
```