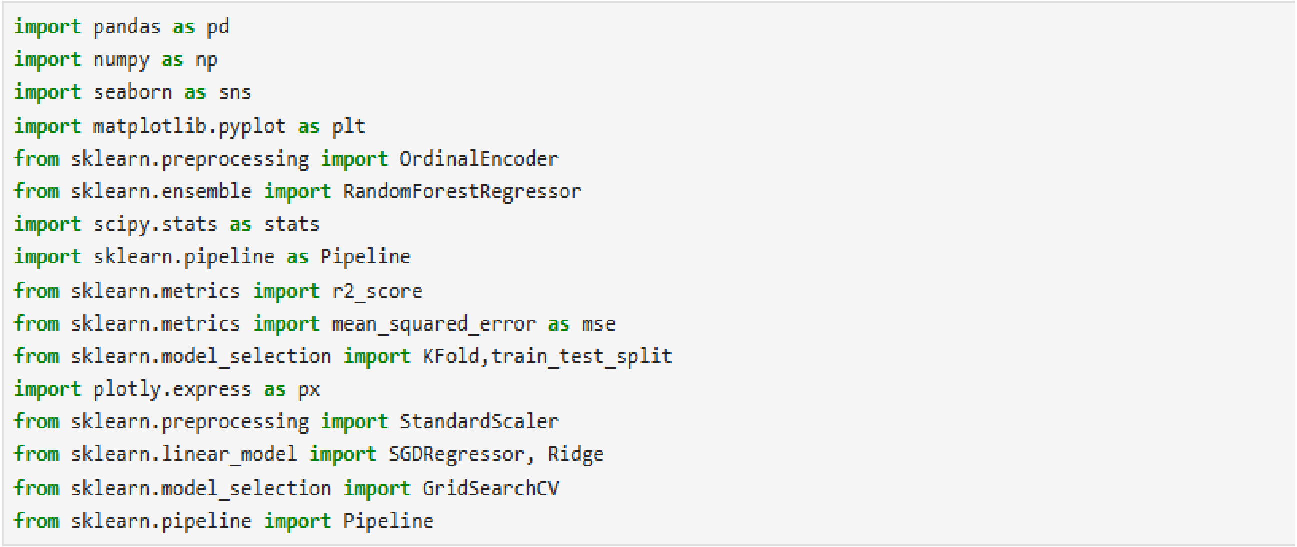
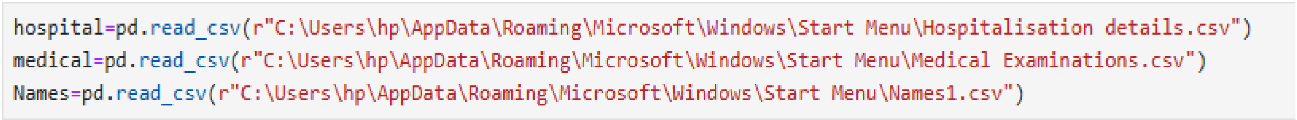
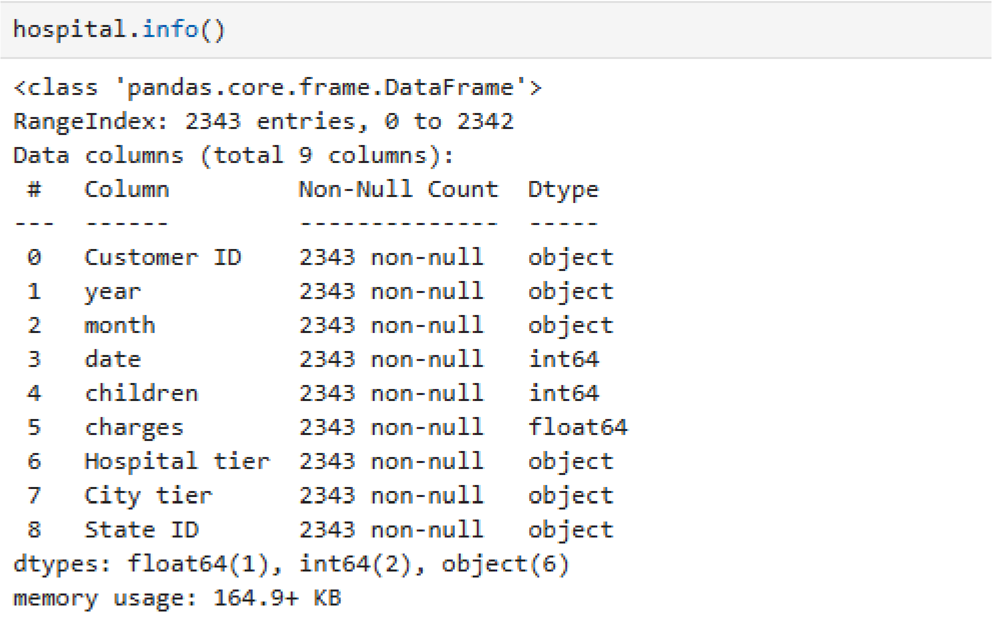
HEALTHCARE PROJECT



```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import RandomForestRegressor
import scipy.stats as stats
import sklearn.pipeline as Pipeline
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error as mse
from sklearn.model_selection import KFold,train_test_split
import plotly.express as px
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor, Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
```

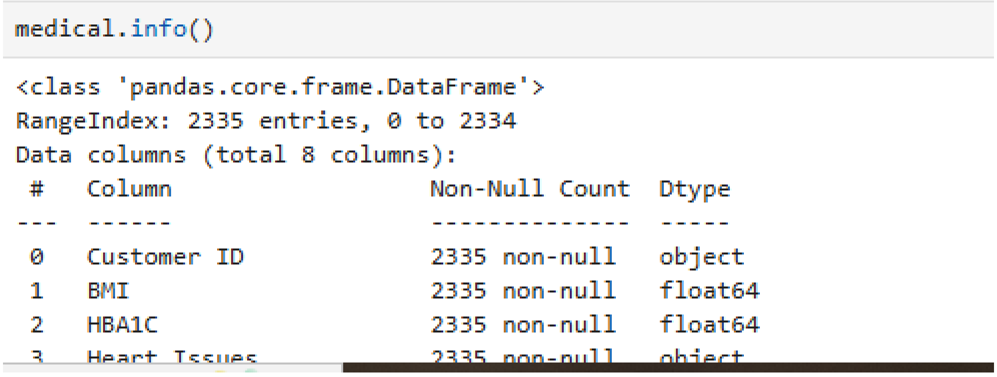


```
hospital=pd.read_csv(r"C:\Users\hp\AppData\Roaming\Microsoft\Windows\Start Menu\Hospitalisation details.csv")
medical=pd.read_csv(r"C:\Users\hp\AppData\Roaming\Microsoft\Windows\Start Menu\Medical Examinations.csv")
Names=pd.read_csv(r"C:\Users\hp\AppData\Roaming\Microsoft\Windows\Start Menu\Names1.csv")
```



```
hospital.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2343 entries, 0 to 2342
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Customer ID    2343 non-null   object 
 1   year           2343 non-null   object 
 2   month          2343 non-null   object 
 3   date           2343 non-null   int64  
 4   children       2343 non-null   int64  
 5   charges        2343 non-null   float64
 6   Hospital tier  2343 non-null   object 
 7   City tier      2343 non-null   object 
 8   State ID       2343 non-null   object 
dtypes: float64(1), int64(2), object(6)
memory usage: 164.9+ KB
```



```
medical.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2335 entries, 0 to 2334
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Customer ID             2335 non-null   object 
 1   BMI                     2335 non-null   float64
 2   HBA1C                   2335 non-null   float64
 3   Heart Issues            2335 non-null   object 
```

```
Names.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2335 entries, 0 to 2334
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Customer ID  2335 non-null   object
 1   name         2335 non-null   object
dtypes: object(2)
memory usage: 36.6+ KB
```

```
hospital.isnull().sum()
```

```
Customer ID      0
year             0
month            0
date             0
children         0
charges          0
Hospital tier    0
City tier        0
State ID         0
dtype: int64
```

```
medical.isnull().sum()
```

```
Customer ID                0
BMI                        0
HBA1C                      0
Heart Issues               0
Any Transplants            0
Cancer history             0
NumberOfMajorSurgeries     0
smoker                     0
dtype: int64
```

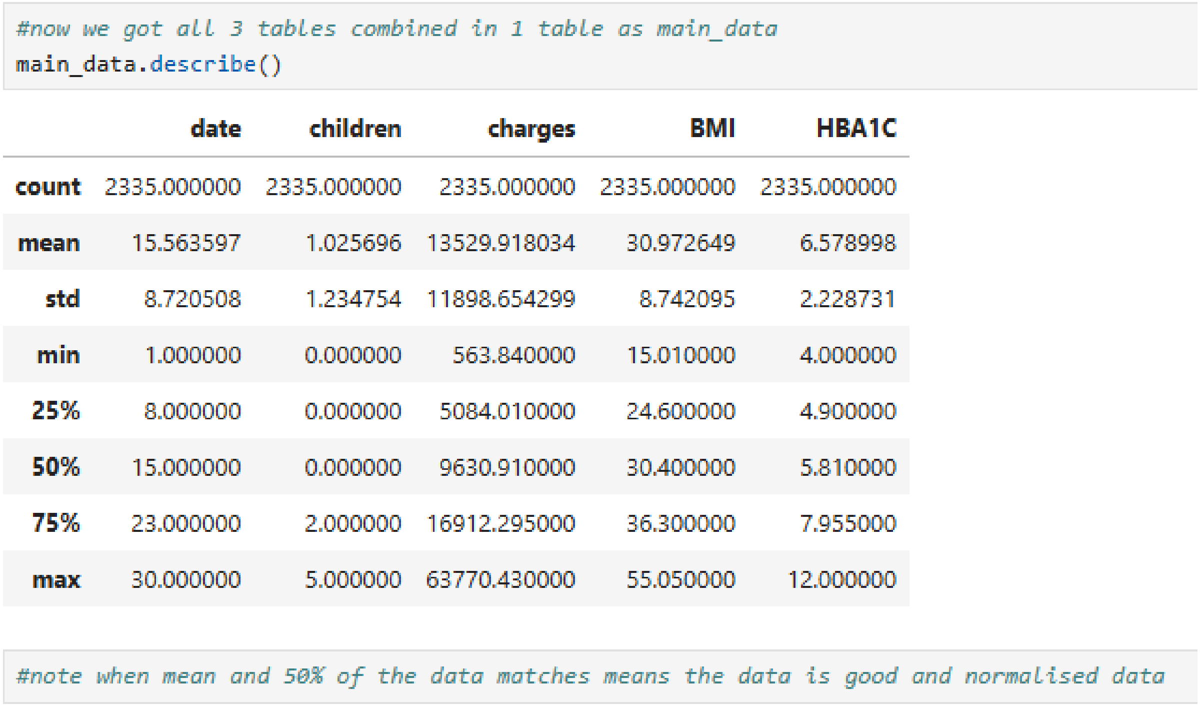
```
Names.isnull().sum()
```

```
Customer ID    0
name           0
dtype: int64
```

```
#there are no missing values in the data
#we can conclude hospitalhas some more entries than medical and names table
```

```
#now we will merge the data(note we can merge 2 table at once)
main_data=pd.merge(hospital,medical,how='inner',on='Customer ID')
```

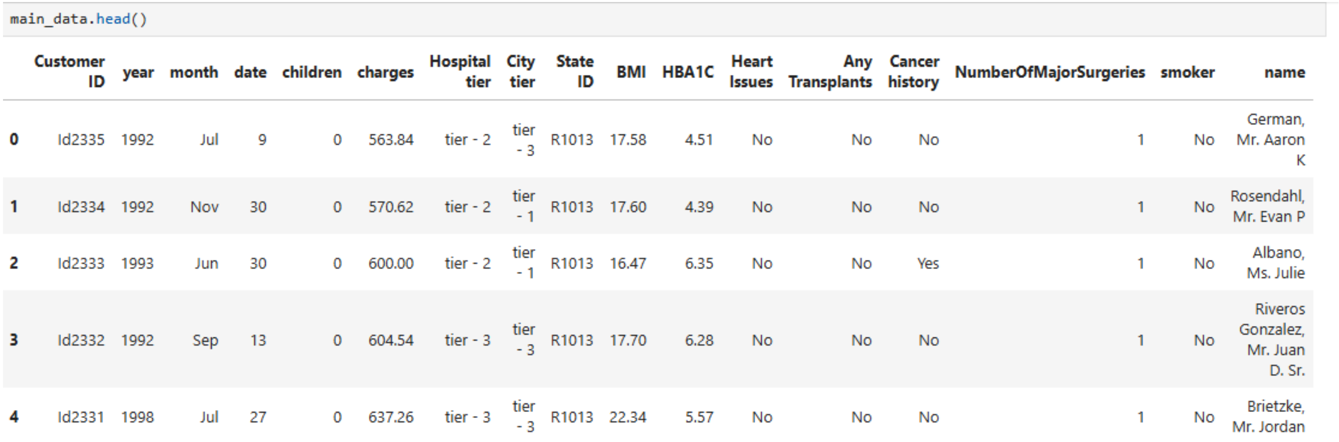
```
main_data=main_data.merge(Names, how='inner',on='Customer ID')
```



```
#now we got all 3 tables combined in 1 table as main_data
main_data.describe()
```

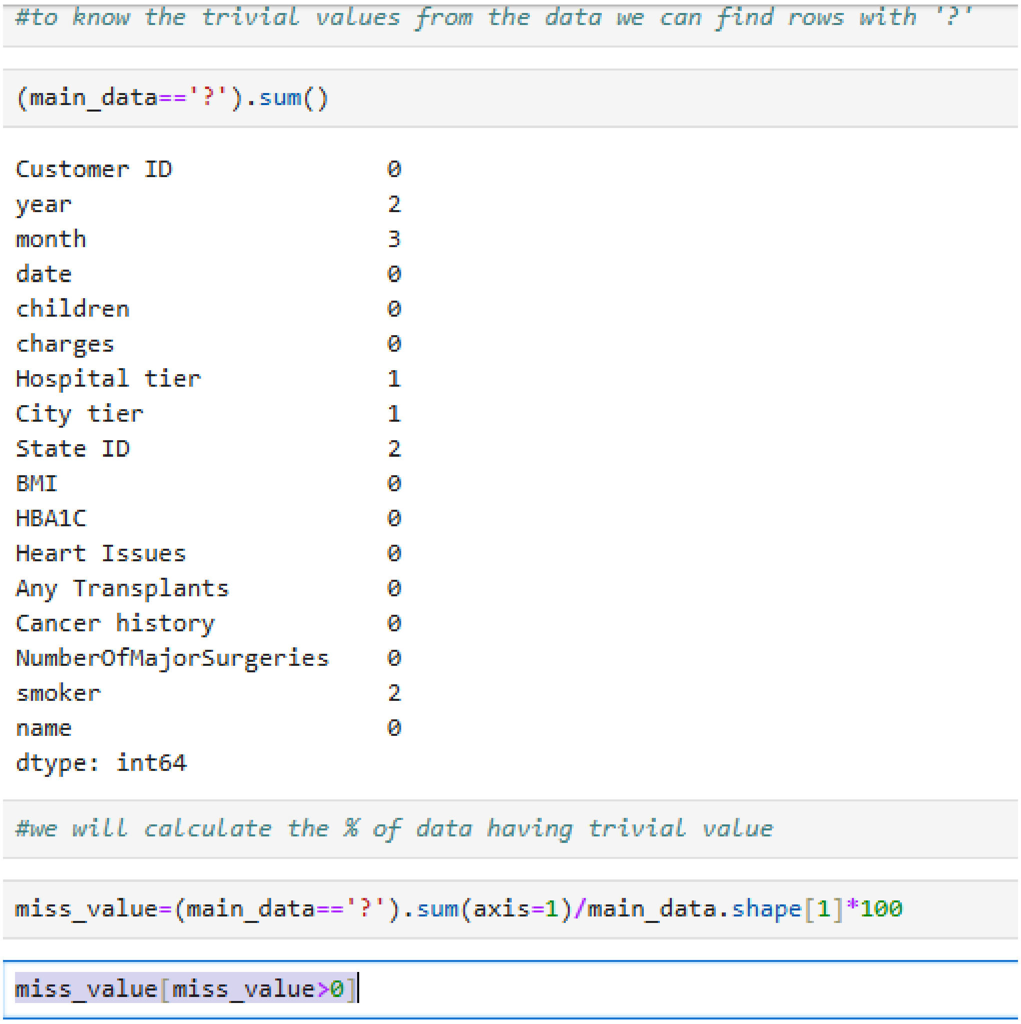
	date	children	charges	BMI	HBA1C
count	2335.000000	2335.000000	2335.000000	2335.000000	2335.000000
mean	15.563597	1.025696	13529.918034	30.972649	6.578998
std	8.720508	1.234754	11898.654299	8.742095	2.228731
min	1.000000	0.000000	563.840000	15.010000	4.000000
25%	8.000000	0.000000	5084.010000	24.600000	4.900000
50%	15.000000	0.000000	9630.910000	30.400000	5.810000
75%	23.000000	2.000000	16912.295000	36.300000	7.955000
max	30.000000	5.000000	63770.430000	55.050000	12.000000

```
#note when mean and 50% of the data matches means the data is good and normalised data
```



```
main_data.head()
```

	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	name
0	Id2335	1992	Jul	9	0	563.84	tier - 2	tier - 3	R1013	17.58	4.51	No	No	No	1	No	German, Mr. Aaron K
1	Id2334	1992	Nov	30	0	570.62	tier - 2	tier - 1	R1013	17.60	4.39	No	No	No	1	No	Rosendahl, Mr. Evan P
2	Id2333	1993	Jun	30	0	600.00	tier - 2	tier - 1	R1013	16.47	6.35	No	No	Yes	1	No	Albano, Ms. Julie
3	Id2332	1992	Sep	13	0	604.54	tier - 3	tier - 3	R1013	17.70	6.28	No	No	No	1	No	Riveros Gonzalez, Mr. Juan D. Sr.
4	Id2331	1998	Jul	27	0	637.26	tier - 3	tier - 3	R1013	22.34	5.57	No	No	No	1	No	Brietzke, Mr. Jordan



```
#to know the trivial values from the data we can find rows with '?'
```

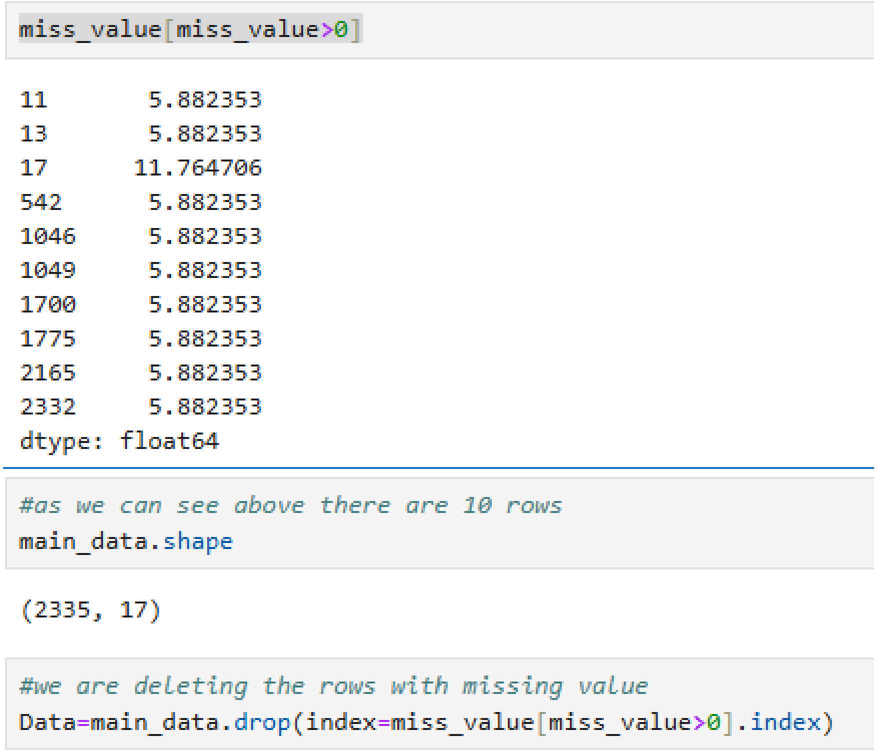
```
(main_data=='?').sum()
```

```
Customer ID                0
year                       2
month                      3
date                       0
children                   0
charges                    0
Hospital tier              1
City tier                  1
State ID                   2
BMI                        0
HBA1C                      0
Heart Issues               0
Any Transplants            0
Cancer history             0
NumberOfMajorSurgeries     0
smoker                     2
name                       0
dtype: int64
```

```
#we will calculate the % of data having trivial value
```

```
miss_value=(main_data=='?').sum(axis=1)/main_data.shape[1]*100
```

```
miss_value[miss_value>0]
```



```
miss_value[miss_value>0]
```

```
11       5.882353
13       5.882353
17      11.764706
542      5.882353
1046     5.882353
1049     5.882353
1700     5.882353
1775     5.882353
2165     5.882353
2332     5.882353
dtype: float64
```

```
#as we can see above there are 10 rows
main_data.shape
```

```
(2335, 17)
```

```
#we are deleting the rows with missing value
Data=main_data.drop(index=miss_value[miss_value>0].index)
```

```
#we are deleting the rows with missing value
Data=main_data.drop(index=miss_value[miss_value>0].index)
```

```
Data.shape
```

```
(2325, 17)
```

```
#rows got deleted successfully
```

```
Data.columns
```

```
Index(['Customer ID', 'year', 'month', 'date', 'children', 'charges',
       'Hospital tier', 'City tier', 'State ID', 'BMI', 'HBA1C',
       'Heart Issues', 'Any Transplants', 'Cancer history',
       'NumberOfMajorSurgeries', 'smoker', 'name'],
      dtype='object')
```

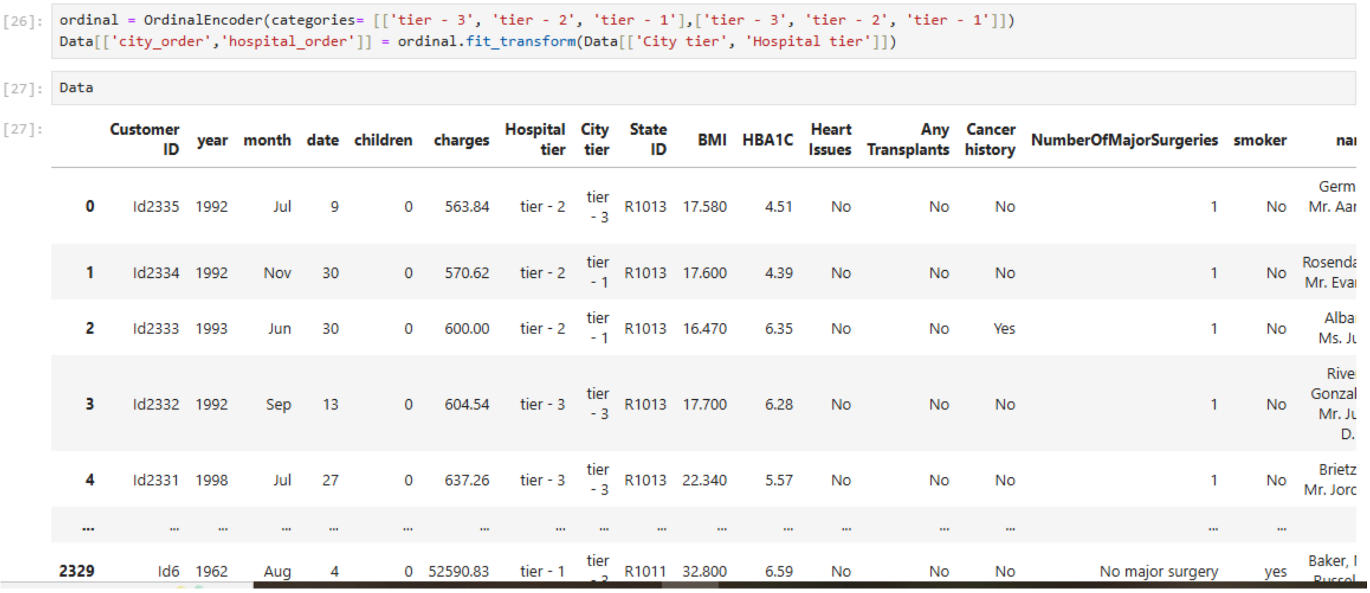
```
#Nominal value- values that are not numeric like color,
          #(heart issues,any transplant, cancer history, smoker,state id)

#Ordinal value- values that are clearly shows order and rank
        #(Hospital tier, city tier)
Group=Data[["Hospital tier","City tier"]]
```

```
Group
```

[25]:

	Hospital tier	City tier
0	tier - 2	tier - 3
1	tier - 2	tier - 1
2	tier - 2	tier - 1
3	tier - 3	tier - 3
4	tier - 3	tier - 3
...	...	...
2329	tier - 1	tier - 3
2330	tier - 1	tier - 2
2331	tier - 1	tier - 3
2333	tier - 2	tier - 3
2334	tier - 1	tier - 3

2325 rows × 2 columns

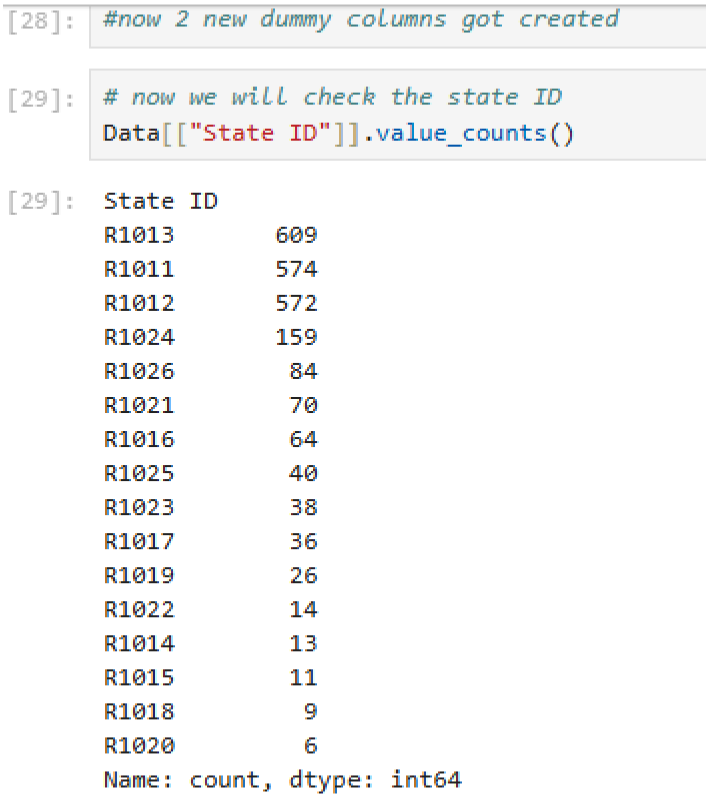


```
[26]: ordinal = OrdinalEncoder(categories= [['tier - 3', 'tier - 2', 'tier - 1'],['tier - 3', 'tier - 2', 'tier - 1']])
      Data[['city_order','hospital_order']] = ordinal.fit_transform(Data[['City tier', 'Hospital tier']])
```

```
[27]: Data
```

[27]:

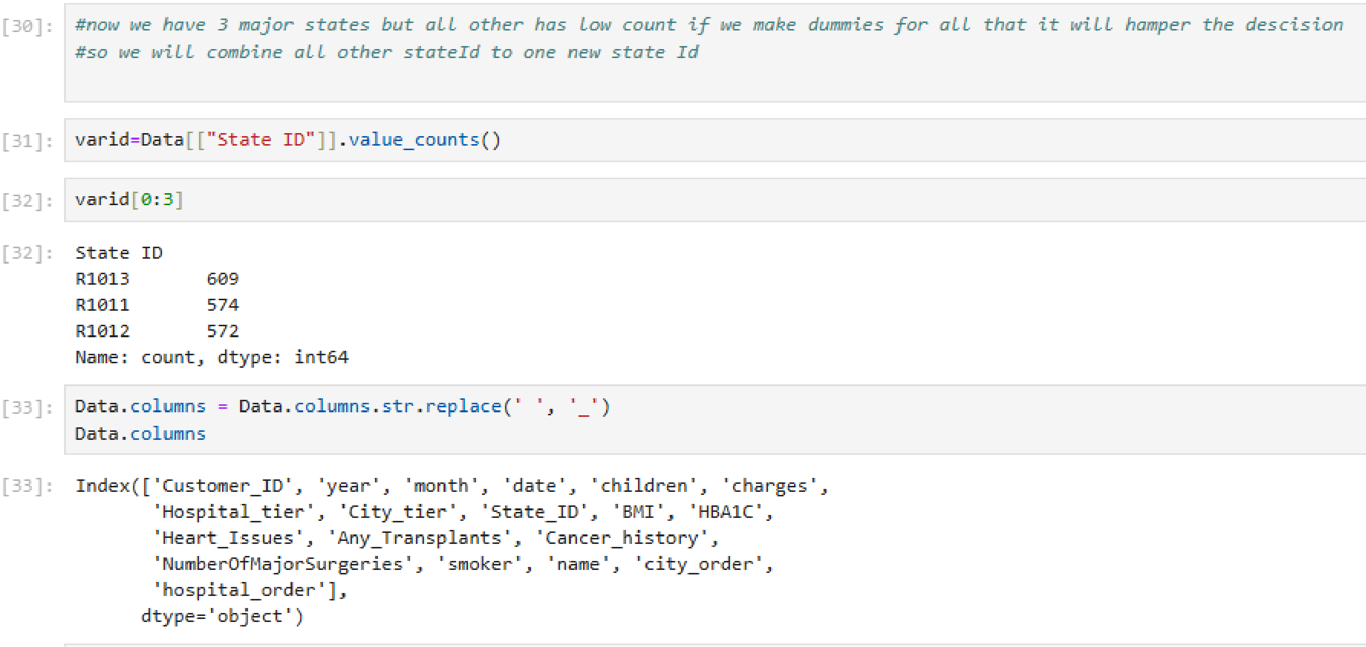
	Customer ID	year	month	date	children	charges	Hospital tier	City tier	State ID	BMI	HBA1C	Heart Issues	Any Transplants	Cancer history	NumberOfMajorSurgeries	smoker	nai
0	Id2335	1992	Jul	9	0	563.84	tier - 2	tier - 3	R1013	17.580	4.51	No	No	No	1	No	Germ Mr. Aar
1	Id2334	1992	Nov	30	0	570.62	tier - 2	tier - 1	R1013	17.600	4.39	No	No	No	1	No	Rosenda Mr. Eva
2	Id2333	1993	Jun	30	0	600.00	tier - 2	tier - 1	R1013	16.470	6.35	No	No	Yes	1	No	Alba Ms. Ju
3	Id2332	1992	Sep	13	0	604.54	tier - 3	tier - 3	R1013	17.700	6.28	No	No	No	1	No	Rive Gonzal Mr. Ju D.
4	Id2331	1998	Jul	27	0	637.26	tier - 3	tier - 3	R1013	22.340	5.57	No	No	No	1	No	Brietz Mr. Jorc
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2329	Id6	1962	Aug	4	0	52590.83	tier - 1	tier - 3	R1011	32.800	6.59	No	No	No	No major surgery	yes	Baker, I Russel



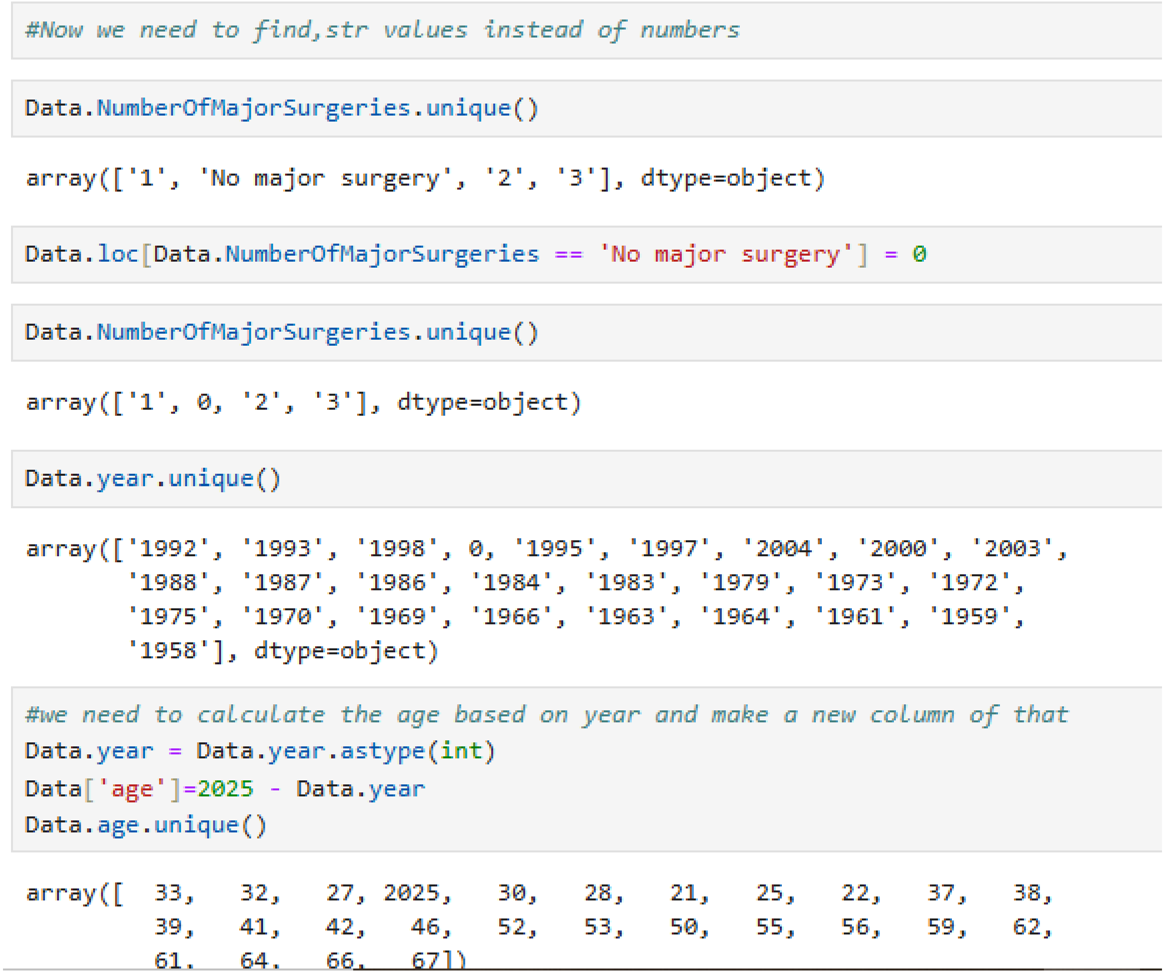
```
[28]: #now 2 new dummy columns got created
```

```
[29]: # now we will check the state ID
      Data[["State ID"]].value_counts()
```

```
[29]: 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
```



```
[30]: #now we have 3 major states but all other has low count if we make dummies for all that it will hamper the descision
      #so we will combine all other stateId to one new state Id

[31]: varid=Data[["State ID"]].value_counts()

[32]: varid[0:3]

[32]: State ID
      R1013       609
      R1011       574
      R1012       572
      Name: count, dtype: int64

[33]: Data.columns = Data.columns.str.replace(' ', '_')
      Data.columns

[33]: Index(['Customer_ID', 'year', 'month', 'date', 'children', 'charges',
             'Hospital_tier', 'City_tier', 'State_ID', 'BMI', 'HBA1C',
             'Heart_Issues', 'Any_Transplants', 'Cancer_history',
             'NumberOfMajorSurgeries', 'smoker', 'name', 'city_order',
             'hospital_order'],
            dtype='object')
```



```
#Now we need to find,str values instead of numbers
```

```
Data.NumberOfMajorSurgeries.unique()
```

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

```
Data.loc[Data.NumberOfMajorSurgeries == 'No major surgery'] = 0
```

```
Data.NumberOfMajorSurgeries.unique()
```

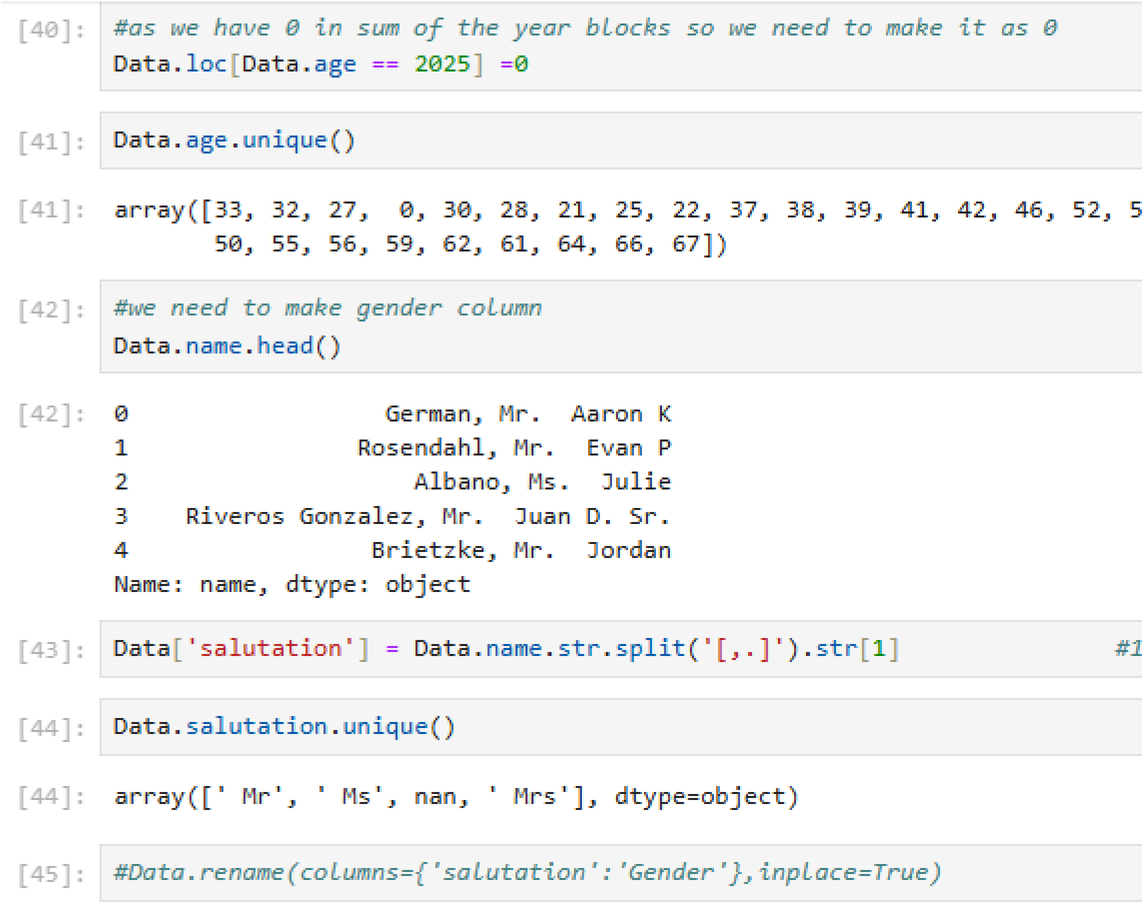
```
array(['1', 0, '2', '3'], dtype=object)
```

```
Data.year.unique()
```

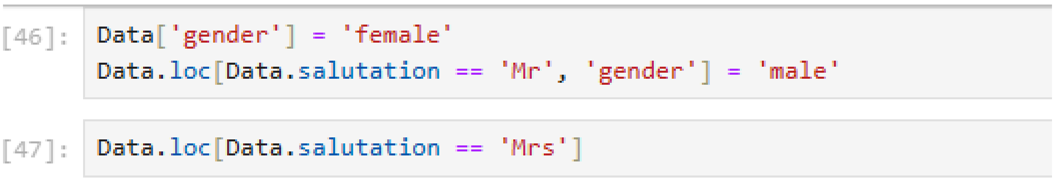
```
array(['1992', '1993', '1998', 0, '1995', '1997', '2004', '2000', '2003',
       '1988', '1987', '1986', '1984', '1983', '1979', '1973', '1972',
       '1975', '1970', '1969', '1966', '1963', '1964', '1961', '1959',
       '1958'], dtype=object)
```

```
#we need to calculate the age based on year and make a new column of that
Data.year = Data.year.astype(int)
Data['age']=2025 - Data.year
Data.age.unique()
```

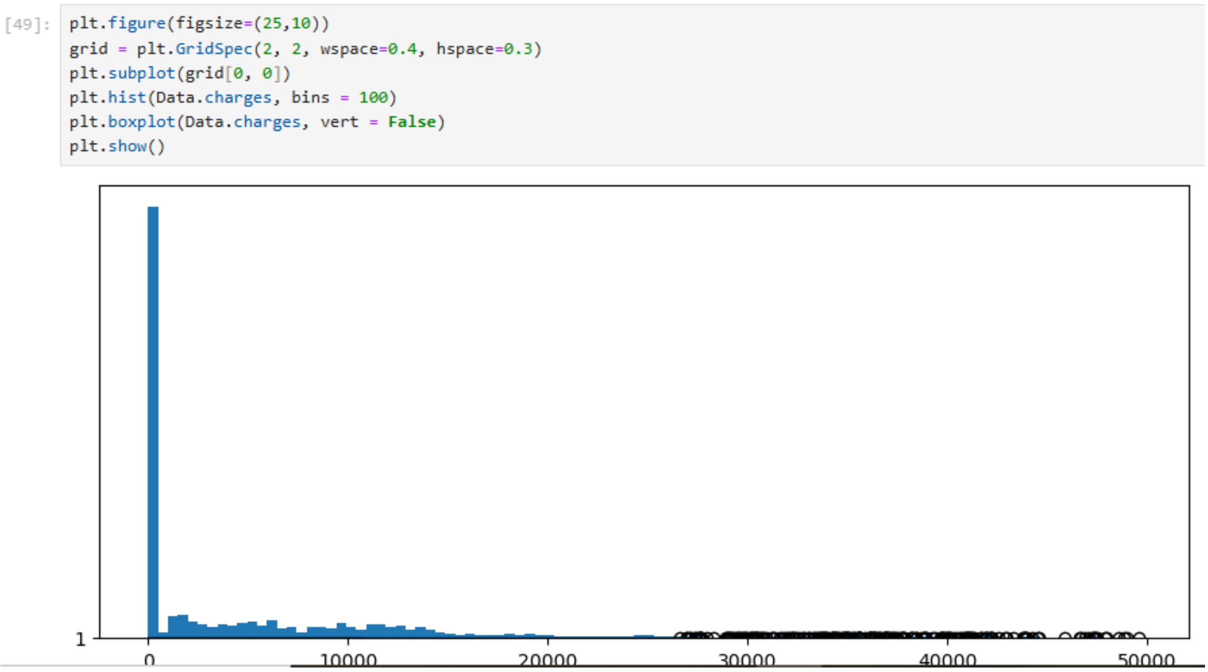
```
array([  33,   32,   27, 2025,   30,   28,   21,   25,   22,   37,   38,
         39,   41,   42,   46,   52,   53,   50,   55,   56,   59,   62,
         61,   64,   66,   67])
```



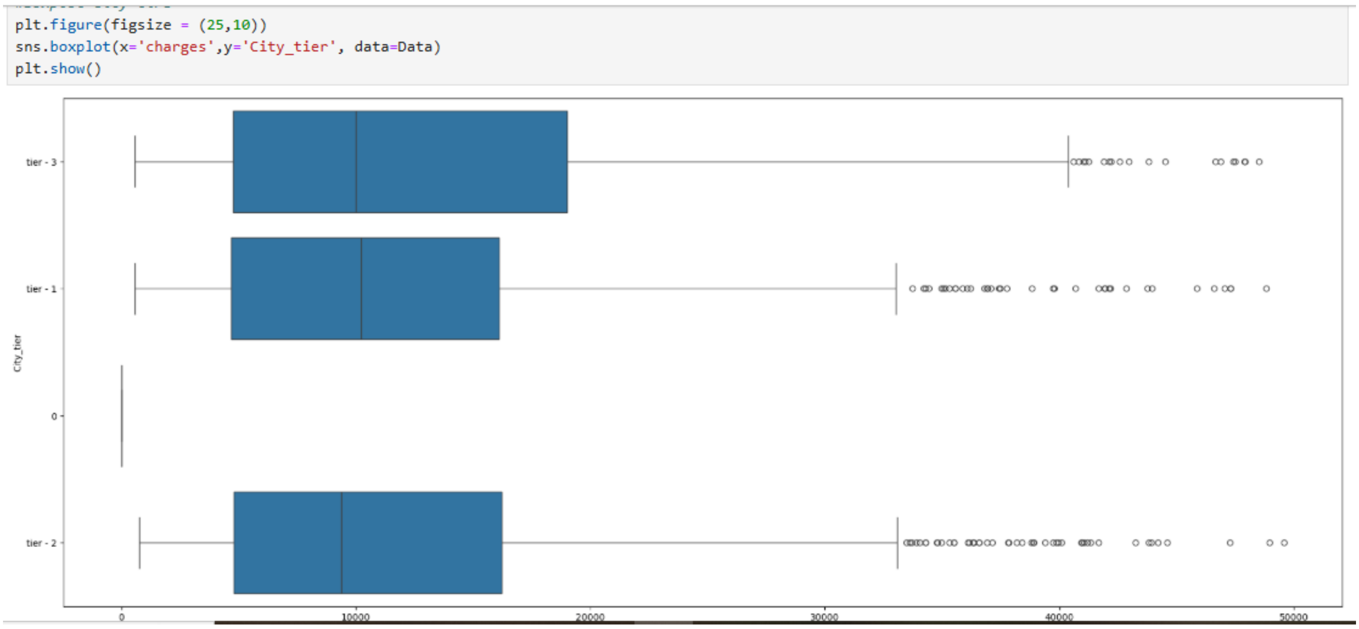
```
[40]: #as we have 0 in sum of the year blocks so we need to make it as 0
      Data.loc[Data.age == 2025] =0

[41]: Data.age.unique()

[41]: array([33, 32, 27,  0, 30, 28, 21, 25, 22, 37, 38, 39, 41, 42, 46, 52, 5
              50, 55, 56, 59, 62, 61, 64, 66, 67])

[42]: #we need to make gender column
      Data.name.head()

[42]: 0                   German, Mr.  Aaron K
      1                 Rosendahl, Mr.  Evan P
      2                     Albano, Ms.  Julie
      3     Riveros Gonzalez, Mr.  Juan D. Sr.
      4                  Brietzke, Mr.  Jordan
      Name: name, dtype: object

[43]: Data['salutation'] = Data.name.str.split('[,.]').str[1]              #1

[44]: Data.salutation.unique()

[44]: array([' Mr', ' Ms', nan, ' Mrs'], dtype=object)

[45]: #Data.rename(columns={'salutation':'Gender'},inplace=True)
```



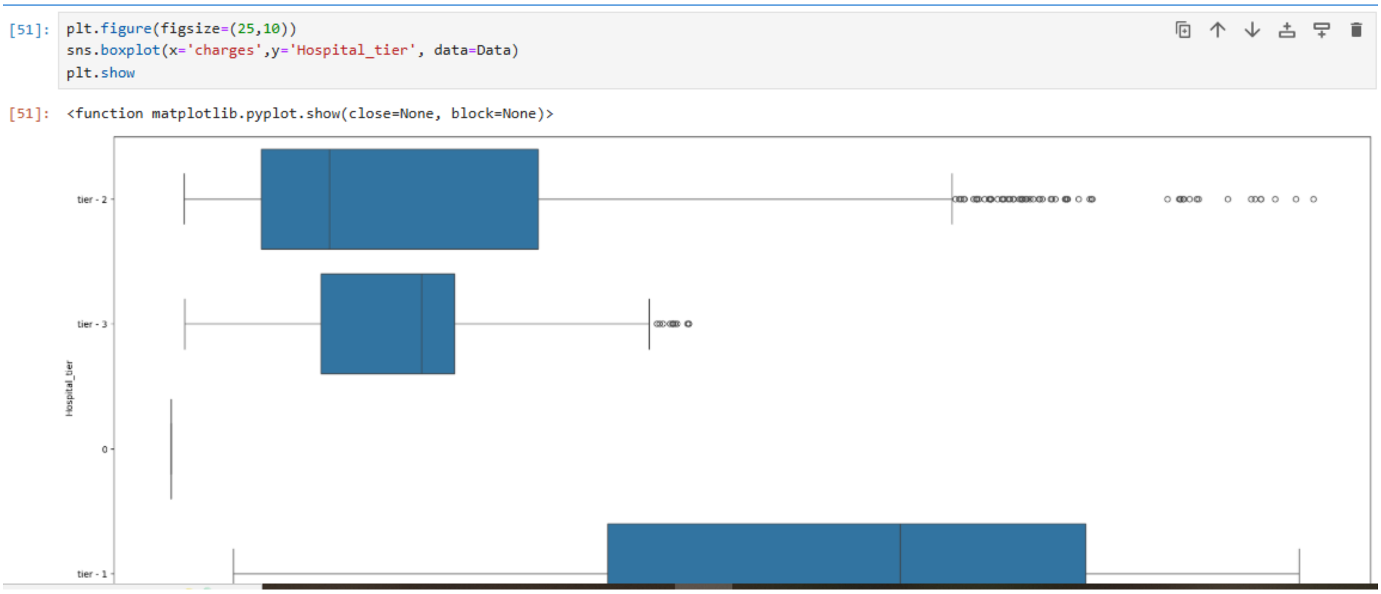
```
[46]: Data['gender'] = 'female'
      Data.loc[Data.salutation == 'Mr', 'gender'] = 'male'

[47]: Data.loc[Data.salutation == 'Mrs']
```



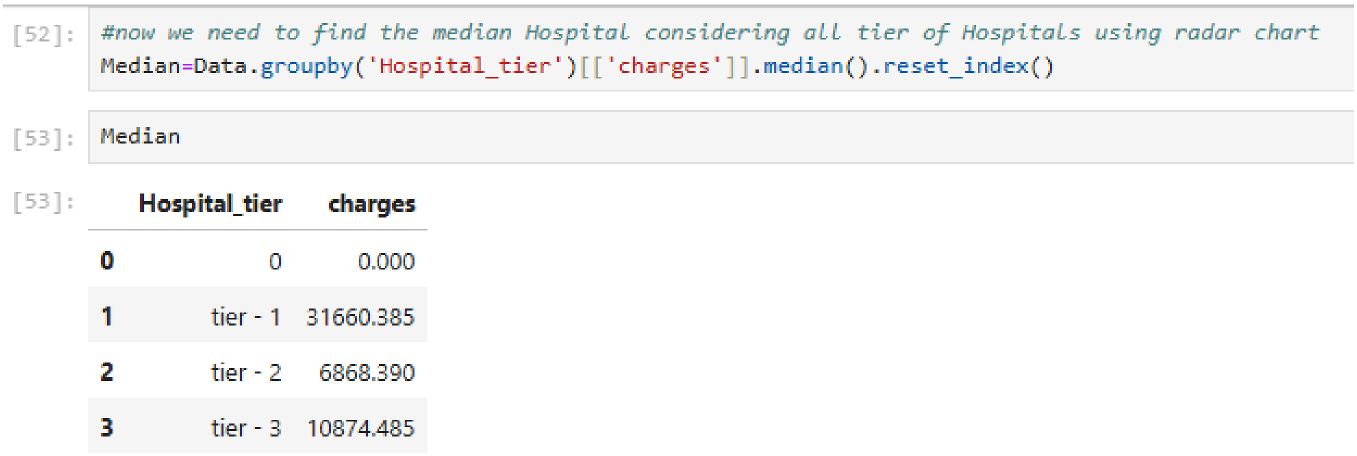
```
[49]: plt.figure(figsize=(25,10))
      grid = plt.GridSpec(2, 2, wspace=0.4, hspace=0.3)
      plt.subplot(grid[0, 0])
      plt.hist(Data.charges, bins = 100)
      plt.boxplot(Data.charges, vert = False)
      plt.show()
```



```
plt.figure(figsize = (25,10))
sns.boxplot(x='charges',y='City_tier', data=Data)
plt.show()
```

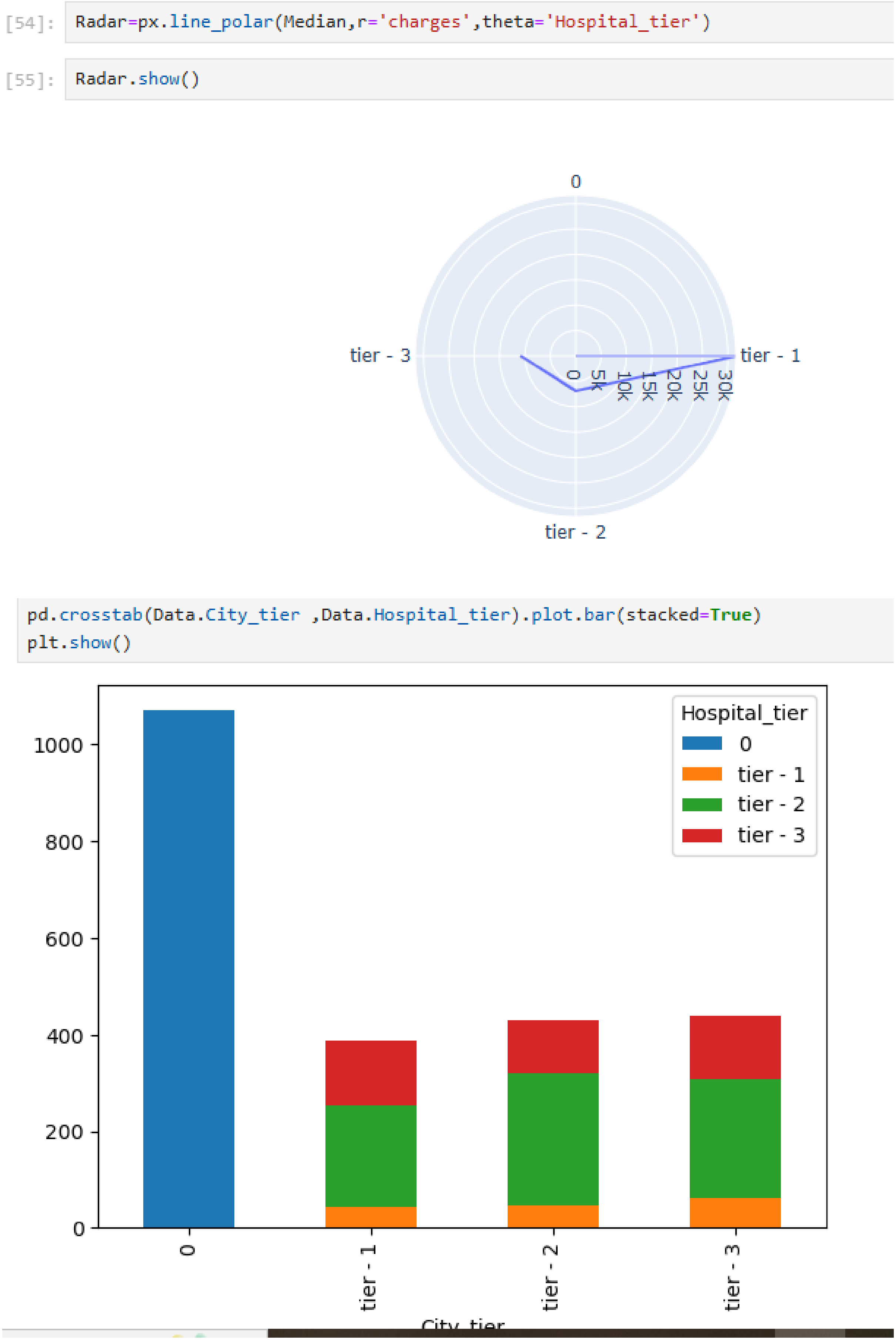


```
[51]: plt.figure(figsize=(25,10))
      sns.boxplot(x='charges',y='Hospital_tier', data=Data)
      plt.show

[51]: <function matplotlib.pyplot.show(close=None, block=None)>
```



```
[52]: #now we need to find the median Hospital considering all tier of Hospitals using radar chart
      Median=Data.groupby('Hospital_tier')[['charges']].median().reset_index()

[53]: Median

[53]:
```

	Hospital_tier	charges
0	0	0.000
1	tier - 1	31660.385
2	tier - 2	6868.390
3	tier - 3	10874.485



```
[54]: Radar=px.line_polar(Median,r='charges',theta='Hospital_tier')
```

```
[55]: Radar.show()
```

```
pd.crosstab(Data.City_tier ,Data.Hospital_tier).plot.bar(stacked=True)
plt.show()
```

57]:

```
pd.crosstab(Data.City_tier, Data.Hospital_tier)
```

57]:

Hospital_tier City_tier	0	tier - 1	tier - 2	tier - 3
0	1070	0	0	0
tier - 1	0	44	209	133
tier - 2	0	47	273	111
tier - 3	0	63	245	130

58]:

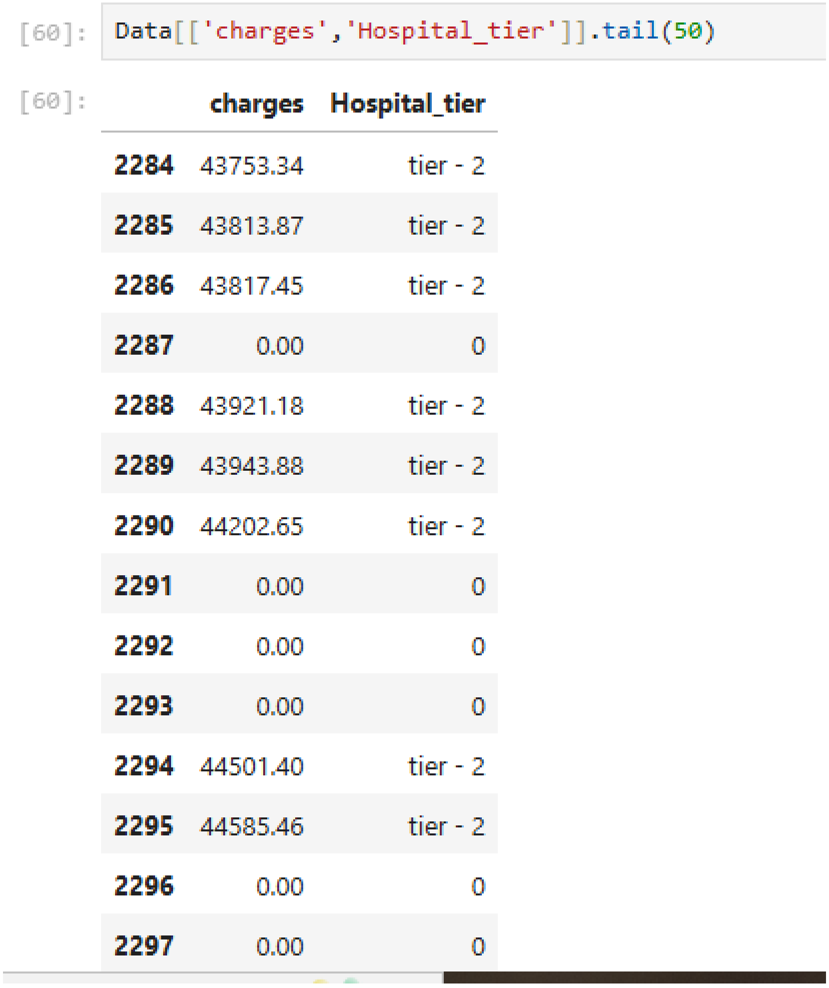
```
#Now we will check for the null hypothesis
#Average hospitalization cost across the 3 types of hospitals is not si
```

59]:

```
Data.columns
```

59]:

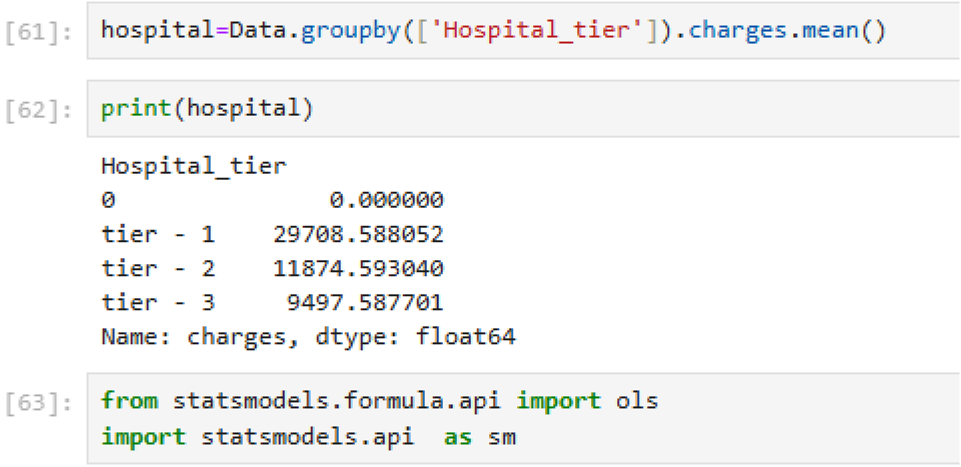
```
Index(['Customer_ID', 'year', 'month', 'date', 'children', 'charges',
       'Hospital_tier', 'City_tier', 'State_ID', 'BMI', 'HBA1C',
       'Heart_Issues', 'Any_Transplants', 'Cancer_history',
       'NumberOfMajorSurgeries', 'smoker', 'name', 'city_order',
       'hospital_order', 'age', 'salutation', 'gender'],
      dtype='object')
```



```
[60]: Data[['charges','Hospital_tier']].tail(50)
```

[60]:

	charges	Hospital_tier
2284	43753.34	tier - 2
2285	43813.87	tier - 2
2286	43817.45	tier - 2
2287	0.00	0
2288	43921.18	tier - 2
2289	43943.88	tier - 2
2290	44202.65	tier - 2
2291	0.00	0
2292	0.00	0
2293	0.00	0
2294	44501.40	tier - 2
2295	44585.46	tier - 2
2296	0.00	0
2297	0.00	0



```
[61]: hospital=Data.groupby(['Hospital_tier']).charges.mean()
```

```
[62]: print(hospital)
```

```
Hospital_tier
0                0.000000
tier - 1     29708.588052
tier - 2     11874.593040
tier - 3      9497.587701
Name: charges, dtype: float64
```

```
[63]: from statsmodels.formula.api import ols
      import statsmodels.api  as sm
```

[64]:

```
mod = ols('charges ~ Hospital_tier', data = Data).fit()
res = sm.stats.anova_lm(mod)
res
```

[64]:

	df	sum_sq	mean_sq	F	PR(>F)
Hospital_tier	3.0	1.513512e+11	5.045041e+10	938.988595	0.0
Residual	2321.0	1.247038e+11	5.372846e+07	NaN	NaN

[65]:

```
#so as we can see above P-value the average charges taken by 3 different categories of hospital
#has significant difference therefore we reject the null hypothesis
```

[66]:

```
#Average hospitalization cost across the 3 types of cities is not significantly different
city=Data.groupby(['City_tier']).charges.mean()
```

[67]:

```
print(city)
```

```
City_tier
0              0.000000
tier - 1    12998.467668
tier - 2    13096.768910
tier - 3    13922.222785
Name: charges, dtype: float64
```

[68]:

```
mod = ols('charges ~ City_tier', data = Data).fit()
res = sm.stats.anova_lm(mod)
res
```

[68]:

	df	sum_sq	mean_sq	F	PR(>F)
City_tier	3.0	1.032260e+11	3.440865e+10	462.089465	2.240926e-235
Residual	2321.0	1.728290e+11	7.446318e+07	NaN	NaN

[69]:

```
#as we can see average hospitalization cost is not significantly different
#so we fail to reject the null hypothesis
```

[70]:

```
# Average hospitalization cost for smokers is not significantly different than non-smokers
```

[71]:

```
smoker = Data.loc[Data.smoker == 'yes', 'charges']
no_smoker=Data.loc[Data.smoker !='yes', 'charges']
```

[72]:

```
stats.ttest_ind(smoker, no_smoker)
```

[72]:

```
TtestResult(statistic=69.96251699445597, pvalue=0.0, df=2323.0)
```

[73]:

```
#from p-value smoker charges to non-smoker charges are different so we reject the null hypothesis
```

[74]:

```
#Smoking and heart issues are independant
```

[75]:

```
table = pd.crosstab(Data.smoker, Data.Heart_Issues)
table
```

[75]:

Heart_Issues	0	No	yes
smoker			
0	1070	0	0
No	0	493	493
yes	0	136	133

[76]:

```
chi, p, df, expected = stats.chi2_contingency(table)
```

[77]:

```
chi, p, df, expected
```

[77]:

```
(2325.0486973279203,
 0.0,
 4,
 array([[492.43010753, 289.47526882, 288.09462366],
        [453.77204301, 266.75010753, 265.47784946],
        [123.79784946,  72.77462366,  72.42752688]]))
```

[78]:

```
#Looking at the p_value, we fail to reject the null hypothesis
```

[79]:

```
#Check the correlation between predictors to identify highly correlated predictors. Visualize using a heatmap.
#important predicting columns are-
Data.columns
```

[79]:

```
Index(['Customer_ID', 'year', 'month', 'date', 'children', 'charges',
       'Hospital_tier', 'City_tier', 'State_ID', 'BMI', 'HBA1C',
       'Heart_Issues', 'Any_Transplants', 'Cancer_history',
       'NumberOfMajorSurgeries', 'smoker', 'name', 'city_order',
       'hospital_order', 'age', 'salutation', 'gender'],
      dtype='object')
```

[80]:

```
new_data=Data.drop(columns=['year','month','date','name','city_order','State_ID','Customer_ID','gender','hospital_order',
                            'Hospital_tier','City_tier','salutation'])
```

[81]:

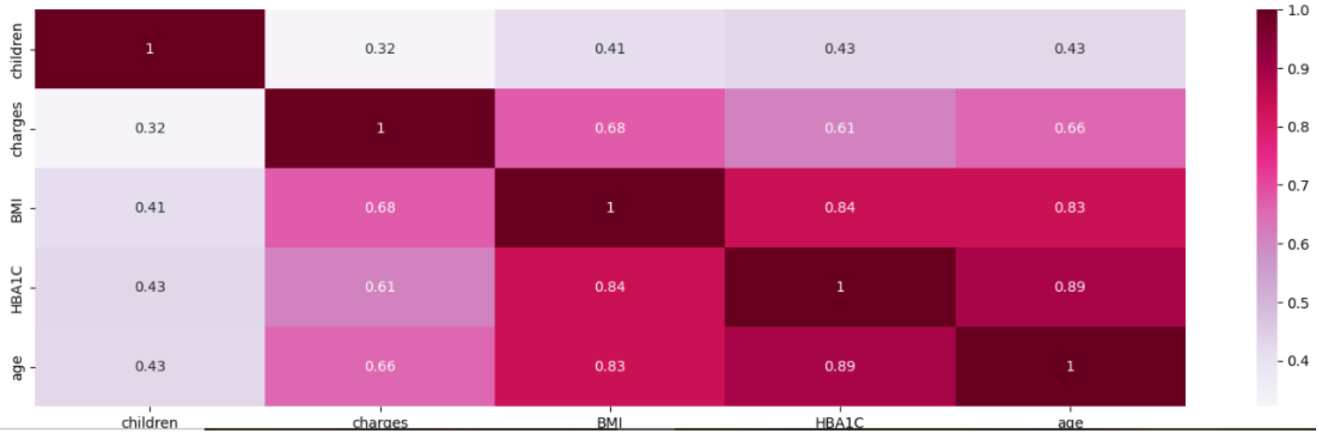
```
new_data
```

[81]:

	children	charges	BMI	HBA1C	Heart_Issues	Any_Transplants	Cancer_history	NumberOfMajorSurgeries	smoker	age
0	0	563.84	17.58	4.51	No	No	No	1	No	33
1	0	570.62	17.60	4.39	No	No	No	1	No	33
2	0	600.00	16.47	6.35	No	No	Yes	1	No	32
3	0	604.54	17.70	6.28	No	No	No	1	No	33
4	0	637.26	22.34	5.57	No	No	No	1	No	27
...	...	...	...	...	...	...	...	...	...	...

[82]:
```
numeric_data = new_data.select_dtypes(exclude='object').corr()
ma = np.ones_like(numeric_data)
```

[83]:
```
plt.figure(figsize = (18,5))
sns.heatmap(numeric_data, annot= True , cmap='PuRd')
plt.show()
```



[84]:
```
#5fold cross validation
data_2 = pd.get_dummies(new_data, drop_first=True)
data_2.reset_index(drop=True, inplace = True)
```

[85]:
```
data_2.head()
```

[85]:

	children	charges	BMI	HBA1C	age	Heart_Issues_No	Heart_Issues_yes	Any_Transplants_No	Any_Transplants_yes	Cancer_history_No	Cancer_history_Yes	NumberOf
0	0	563.84	17.58	4.51	33	True	False	True	False	True	False	
1	0	570.62	17.60	4.39	33	True	False	True	False	True	False	
2	0	600.00	16.47	6.35	32	True	False	True	False	False	True	
3	0	604.54	17.70	6.28	33	True	False	True	False	True	False	
4	0	637.26	22.34	5.57	27	True	False	True	False	True	False	

[86]:
```
model_data = data_2.drop(columns = 'charges')
model_data.head()
model_data['charges'] = data_2.charges
model_data.head()
```

[86]:

	children	BMI	HBA1C	age	Heart_Issues_No	Heart_Issues_yes	Any_Transplants_No	Any_Transplants_yes	Cancer_history_No	Cancer_history_Yes	NumberOfMajorSurg
0	0	17.58	4.51	33	True	False	True	False	True	False	
1	0	17.60	4.39	33	True	False	True	False	True	False	
2	0	16.47	6.35	32	True	False	True	False	False	True	
3	0	17.70	6.28	33	True	False	True	False	True	False	
4	0	22.34	5.57	27	True	False	True	False	True	False	

[87]:
```
X=data_2.drop(columns ='charges')
y=data_2['charges']
```

[88]:
```
pipeline = Pipeline(steps=[('scaler', StandardScaler()), ('regressor', Ridge())])
```

[89]:
```
parameters = {'regressor__alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
```

[90]:

```
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

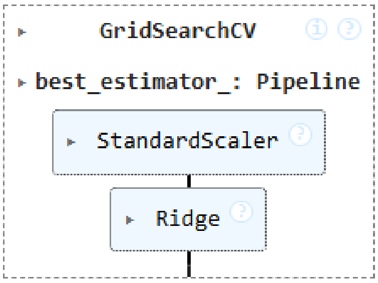
[91]:

```
model_ridge = GridSearchCV(pipeline, parameters, cv=kfold, scoring='neg_mean_squared_error')
```

[92]:

```
model_ridge.fit(X, y)
```

[92]:



▸ GridSearchCV

▸ best_estimator_: Pipeline

▸ StandardScaler

▸ Ridge

[93]:

```
model_ridge.best_params_
```

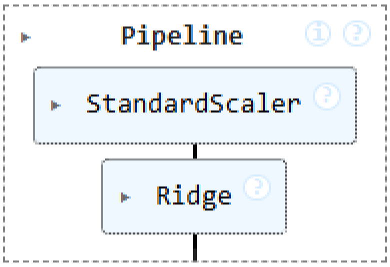
[93]:

```
{'regressor__alpha': 1}
```

[94]:

```
model_ridge.best_estimator_
```

[94]:



▸ Pipeline

▸ StandardScaler

▸ Ridge

[97]:

```
from sklearn.ensemble import GradientBoostingRegressor
```

[98]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y)
```

[99]:

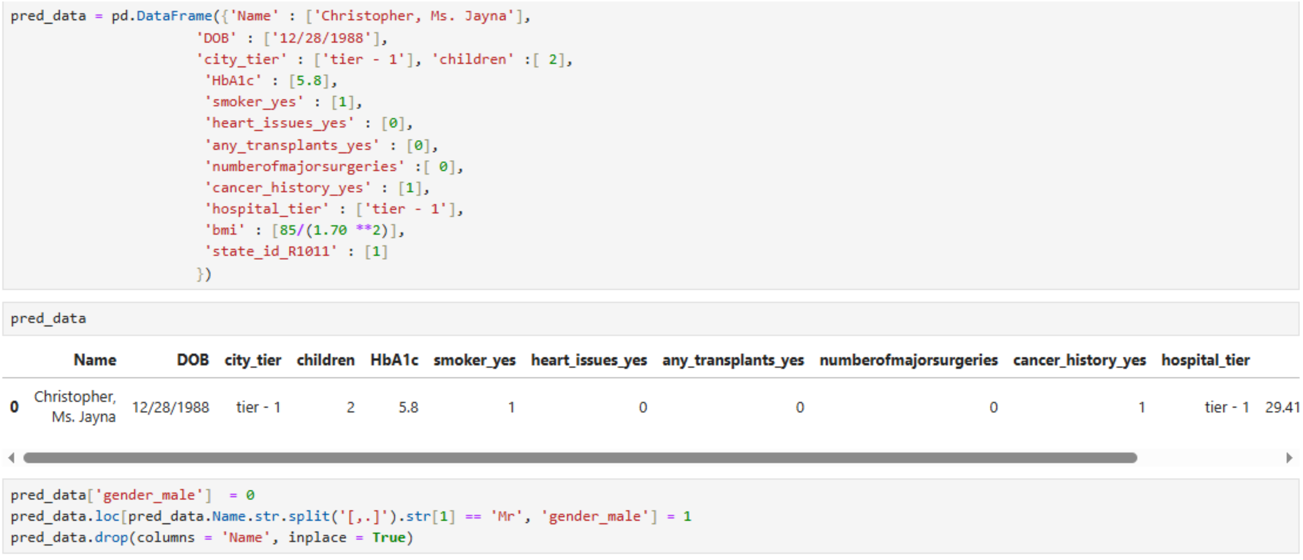
```
model = GradientBoostingRegressor()
model.fit(X_train, y_train)
```

[99]:

▾ GradientBoostingRegressor

```
GradientBoostingRegressor()
```

```
[100]: print(model.feature_importances_)

       [9.26529561e-03 1.18484675e-01 1.17478367e-02 1.69330114e-01
        1.86651393e-04 0.00000000e+00 7.93831112e-05 1.98567593e-04
        6.47220664e-07 1.64507916e-04 2.38864174e-05 4.69668547e-05
        7.23706261e-06 5.85556072e-04 6.89878675e-01]

[101]: model.score(X_train,y_train)

[101]: 0.9528373650472055

[102]: model.score(X_test,y_test)

[102]: 0.9516113339047438

[103]: model_data.columns

[103]: Index(['children', 'BMI', 'HBA1C', 'age', 'Heart_Issues_No',
              'Heart_Issues_yes', 'Any_Transplants_No', 'Any_Transplants_yes',
              'Cancer_history_No', 'Cancer_history_Yes', 'NumberOfMajorSurgeries_1',
              'NumberOfMajorSurgeries_2', 'NumberOfMajorSurgeries_3', 'smoker_No',
              'smoker_yes', 'charges'],
             dtype='object')
```

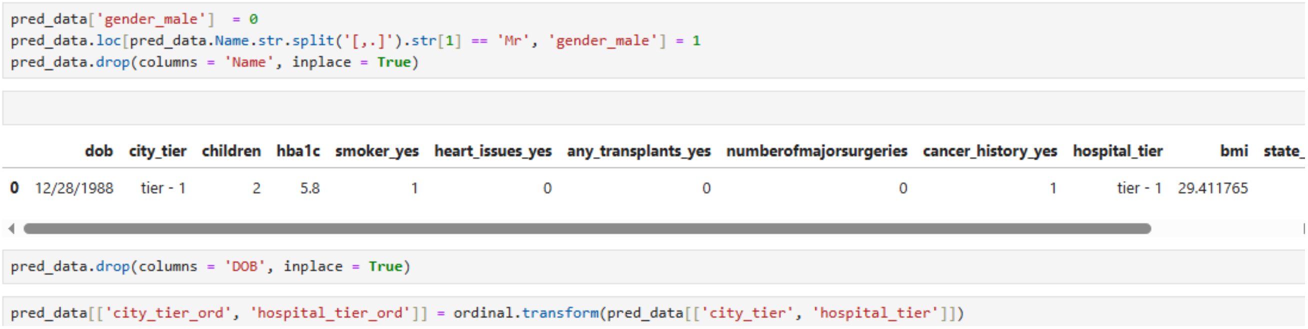


```
[118]: pred_data = pd.DataFrame({'Name' : ['Christopher, Ms. Jayna'],
                        'DOB' : ['12/28/1988'],
                        'city_tier' : ['tier - 1'], 'children' :[ 2],
                         'HbA1c' : [5.8],
                         'smoker_yes' : [1],
                         'heart_issues_yes' : [0],
                         'any_transplants_yes' : [0],
                         'numberofmajorsurgeries' :[ 0],
                         'cancer_history_yes' : [1],
                         'hospital_tier' : ['tier - 1'],
                         'bmi' : [85/(1.70 **2)],
                         'state_id_R1011' : [1]
                        })

[119]: pred_data
```

[119]:

	Name	DOB	city_tier	children	HbA1c	smoker_yes	heart_issues_yes	any_transplants_yes	numberofmajorsurgeries	cancer_history_yes	hospital_tier	
0	Christopher, Ms. Jayna	12/28/1988	tier - 1	2	5.8	1	0	0	0	1	tier - 1	29.41

```
[120]: pred_data['gender_male']  = 0
       pred_data.loc[pred_data.Name.str.split('[,.]').str[1] == 'Mr', 'gender_male'] = 1
       pred_data.drop(columns = 'Name', inplace = True)
```



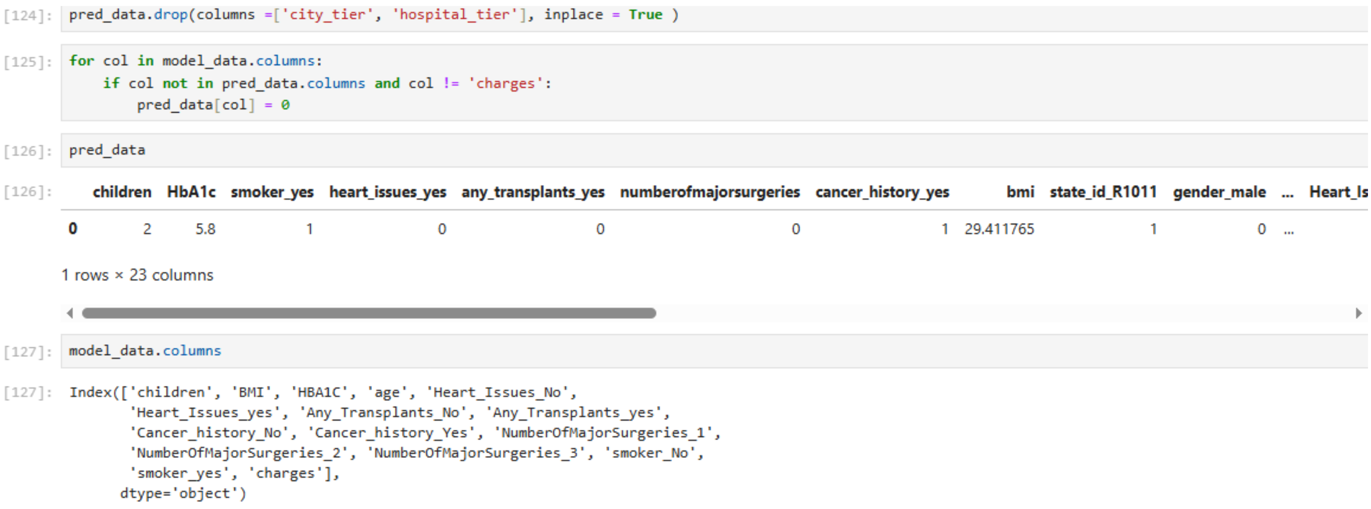
```
[120]: pred_data['gender_male']  = 0
       pred_data.loc[pred_data.Name.str.split('[,.]').str[1] == 'Mr', 'gender_male'] = 1
       pred_data.drop(columns = 'Name', inplace = True)
```

```
[109]:
```

[109]:

	dob	city_tier	children	hba1c	smoker_yes	heart_issues_yes	any_transplants_yes	numberofmajorsurgeries	cancer_history_yes	hospital_tier	bmi	state_
0	12/28/1988	tier - 1	2	5.8	1	0	0	0	1	tier - 1	29.411765	

```
[122]: pred_data.drop(columns = 'DOB', inplace = True)

[123]: pred_data[['city_tier_ord', 'hospital_tier_ord']] = ordinal.transform(pred_data[['city_tier', 'hospital_tier']])
```



```
[124]: pred_data.drop(columns =['city_tier', 'hospital_tier'], inplace = True )
```

```
[125]: for col in model_data.columns:
           if col not in pred_data.columns and col != 'charges':
               pred_data[col] = 0
```

```
[126]: pred_data
```

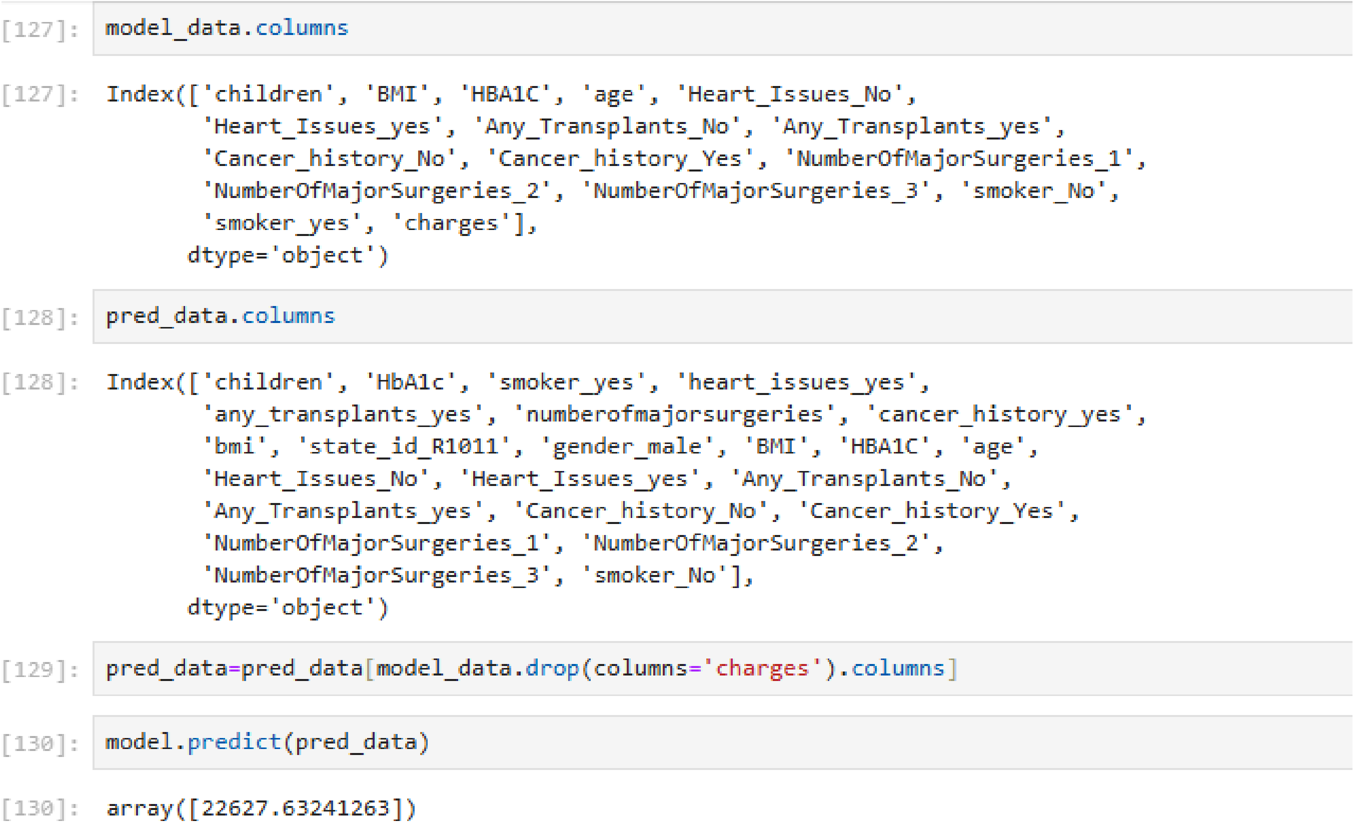
[126]:

	children	HbA1c	smoker_yes	heart_issues_yes	any_transplants_yes	numberofmajorsurgeries	cancer_history_yes	bmi	state_id_R1011	gender_male	...	Heart_Is
0	2	5.8	1	0	0	0	1	29.411765	1	0	...	

1 rows × 23 columns

```
[127]: model_data.columns
```

```
[127]: Index(['children', 'BMI', 'HBA1C', 'age', 'Heart_Issues_No',
              'Heart_Issues_yes', 'Any_Transplants_No', 'Any_Transplants_yes',
              'Cancer_history_No', 'Cancer_history_Yes', 'NumberOfMajorSurgeries_1',
              'NumberOfMajorSurgeries_2', 'NumberOfMajorSurgeries_3', 'smoker_No',
              'smoker_yes', 'charges'],
             dtype='object')
```



```
[127]: model_data.columns
```

```
[127]: Index(['children', 'BMI', 'HBA1C', 'age', 'Heart_Issues_No',
              'Heart_Issues_yes', 'Any_Transplants_No', 'Any_Transplants_yes',
              'Cancer_history_No', 'Cancer_history_Yes', 'NumberOfMajorSurgeries_1',
              'NumberOfMajorSurgeries_2', 'NumberOfMajorSurgeries_3', 'smoker_No',
              'smoker_yes', 'charges'],
             dtype='object')
```

```
[128]: pred_data.columns
```

```
[128]: Index(['children', 'HbA1c', 'smoker_yes', 'heart_issues_yes',
              'any_transplants_yes', 'numberofmajorsurgeries', 'cancer_history_yes',
              'bmi', 'state_id_R1011', 'gender_male', 'BMI', 'HBA1C', 'age',
              'Heart_Issues_No', 'Heart_Issues_yes', 'Any_Transplants_No',
              'Any_Transplants_yes', 'Cancer_history_No', 'Cancer_history_Yes',
              'NumberOfMajorSurgeries_1', 'NumberOfMajorSurgeries_2',
              'NumberOfMajorSurgeries_3', 'smoker_No'],
             dtype='object')
```

```
[129]: pred_data=pred_data[model_data.drop(columns='charges').columns]
```

```
[130]: model.predict(pred_data)
```

```
[130]: array([22627.63241263])
```

END