F23 Assignment 2 (10%)

Importing the libraries

In [1]:
```
import pandas as pd #To read the csv file and manage dataframes
import numpy as np #To use aggregations
import warnings #To hide the warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)

import seaborn as sns #To create visualizations
import matplotlib.pyplot as plt #To create visualizations

from sklearn.model_selection import train_test_split #To split the dataset into tra
from sklearn.linear_model import LinearRegression #To fit the data into a Linear Re
from sklearn.metrics import mean_squared_error, r2_score #To check the mean squarec
from sklearn.preprocessing import StandardScaler #To scale the dataset
```

Importing the dataset

In [2]:
```
HealthCareData = pd.read_csv("HealthCareDataSet - Sheet1 (1).csv") # Importing the
```

Data Cleaning and Pre-processing

Shows the shape of the dataset

In [3]:
```
HealthCareData.shape
```

Out[3]:
```
(10999, 13)
```

Shows the schema of the dataset

In [4]:
```
HealthCareData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10999 entries, 0 to 10998
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Id                     10999 non-null  int64  
 1   LengthOfStay           10999 non-null  int64  
 2   ReadmissionCount       5429 non-null   float64
 3   Gender                 10999 non-null  object 
 4   FacilityId             10999 non-null  int64  
 5   KidneyAilments         0 non-null      float64
 6   HeartAilments          0 non-null      float64
 7   PyschologicalAilments  10999 non-null  bool   
 8   SubstanceAbuseHistory  10607 non-null  object 
 9   BMI                    10997 non-null  float64
 10  ABG                    10999 non-null  float64
 11  Pulse                  10997 non-null  float64
 12  SecondaryDiagnosis     10760 non-null  float64
dtypes: bool(1), float64(7), int64(3), object(2)
memory usage: 1.0+ MB
```

To calculate the total null values in each column

```
In [5]: HealthCareData.isnull().sum()
```

```
Out[5]: Id                           0
        LengthOfStay                 0
        ReadmissionCount          5570
        Gender                       0
        FacilityId                   0
        KidneyAilments           10999
        HeartAilments            10999
        PyschologicalAilments        0
        SubstanceAbuseHistory      392
        BMI                          2
        ABG                          0
        Pulse                        2
        SecondaryDiagnosis         239
        dtype: int64
```

Since Kidney Ailments and HeartAilments are completely null columns, we drop them

```
In [6]: HealthCareData.drop(['KidneyAilments','HeartAilments'],axis = 1,inplace = True)
```

Id and FacilityId can be considered as primary keys here so we drop them

```
In [7]: HealthCareData.drop(['Id','FacilityId'],axis = 1,inplace = True)
```

Data Cleaning and Pre-processing

.isnull() is used to find the null values in the dataset and .sum() is used to sum the null values

```
In [8]: HealthCareData.isnull().sum()
```

```
Out[8]: LengthOfStay                0
        ReadmissionCount         5570
        Gender                      0
        PyschologicalAilments       0
        SubstanceAbuseHistory     392
        BMI                         2
        ABG                         0
        Pulse                       2
        SecondaryDiagnosis        239
        dtype: int64
```

.describe() shows the aggregated summary of the column selected

```
In [9]: HealthCareData.ReadmissionCount.describe()
```

```
Out[9]: count    5429.000000
        mean        1.883036
        std         1.102079
        min         1.000000
        25%         1.000000
        50%         2.000000
        75%         2.000000
        max         5.000000
        Name: ReadmissionCount, dtype: float64
```

.value_counts() shows the distinct value in the selected column

```
In [10]: HealthCareData.ReadmissionCount.value_counts()
```

Out[10]:
```
1.0    2701
2.0    1422
3.0     736
4.0     380
5.0     190
Name: ReadmissionCount, dtype: int64
```

Considering ReadmissionCount as a Categorical Variable since it is discrete and ranges between 1 and 5.
if the data is categorical then we replace the null values with mode.

In [11]:
```
HealthCareData['ReadmissionCount'].fillna(HealthCareData['ReadmissionCount'].mode()
HealthCareData.ReadmissionCount = HealthCareData.ReadmissionCount.astype('int').ast
```

In [12]:
```
HealthCareData.SubstanceAbuseHistory.describe()
```

Out[12]:
```
count     10607
unique        6
top        None
freq       9080
Name: SubstanceAbuseHistory, dtype: object
```

In [13]:
```
HealthCareData.SubstanceAbuseHistory.value_counts()
```

Out[13]:
```
None       9080
Yes        1517
Some          3
Slight        3
Few           3
Unknown       1
Name: SubstanceAbuseHistory, dtype: int64
```

Continuing Cleaning and Pre-processing column
The null values can be added in the Unknown column since it indicates that the values are unavailable
The columns 'Slight', 'Few' and 'Some' can be merged together since they mean the same thing

In [14]:
```
HealthCareData['SubstanceAbuseHistory'].fillna('Unknown', inplace=True)

HealthCareData['SubstanceAbuseHistory'] = HealthCareData['SubstanceAbuseHistory'].r
```

In [15]:
```
HealthCareData.SecondaryDiagnosis.describe()
```

Out[15]:
```
count    10760.000000
mean         2.446933
std          1.779553
min          0.000000
25%          1.000000
50%          2.000000
75%          3.000000
max         10.000000
Name: SecondaryDiagnosis, dtype: float64
```

In [16]:
```
HealthCareData.SecondaryDiagnosis.value_counts()
```

```
Out[16]: 1.0     4196
         2.0     2610
         3.0     1568
         4.0      946
         5.0      545
         6.0      363
         7.0      221
         8.0      133
         0.0       81
         9.0       59
         10.0      38
         Name: SecondaryDiagnosis, dtype: int64
```

Considering SecondaryDiagnosis as a Categorical Variable since it is discrete and ranges between 1 and 10
Since the contribution of null data is very less, we decided to remove it.

```
In [17]: (HealthCareData['SecondaryDiagnosis'].isnull().sum()/HealthCareData['SecondaryDiagr
```

```
Out[17]: 0.9077443123551977
```

```
In [18]: HealthCareData = HealthCareData[HealthCareData['SecondaryDiagnosis'].isnull() == Fa
         HealthCareData.SecondaryDiagnosis = HealthCareData.SecondaryDiagnosis.astype('int')
```

```
In [19]: HealthCareData.isnull().sum()
```

```
Out[19]: LengthOfStay             0
         ReadmissionCount         0
         Gender                   0
         PyschologicalAilments    0
         SubstanceAbuseHistory    0
         BMI                      1
         ABG                      0
         Pulse                    1
         SecondaryDiagnosis       0
         dtype: int64
```

BMI is a numerical value, so we first check if it is skewed or not. We check for the skewness so that we can accordingly replace the null values with either mean or median or if the missing is data is small then we remove it entirely

```
In [20]: HealthCareData.BMI.describe()
```

```
Out[20]: count    10759.000000
         mean        29.061298
         std          2.004916
         min         19.800000
         25%         27.700000
         50%         29.000000
         75%         30.400000
         max         36.700000
         Name: BMI, dtype: float64
```

Percentage of null BMI values compared to all BMI values:

```
In [21]: (HealthCareData.BMI.isnull().sum()/HealthCareData.BMI.sum())*100
```

```
Out[21]: 0.0003198255032054511
```

Therefore we can decide to remove the null values entirely

```
In [22]: HealthCareData = HealthCareData[HealthCareData['BMI'].isnull() == False]
         HealthCareData.isnull().sum()
```

Out[22]:

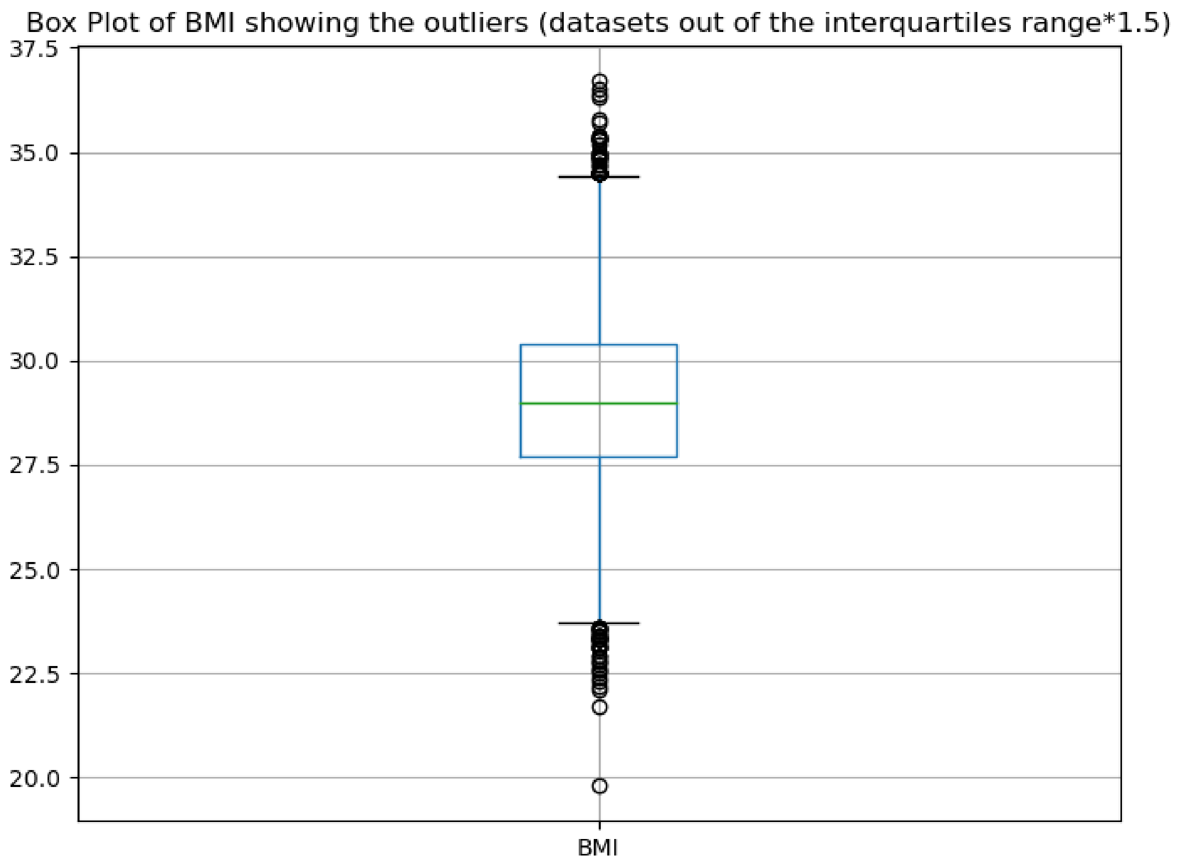
```
LengthOfStay            0
ReadmissionCount        0
Gender                  0
PyschologicalAilments   0
SubstanceAbuseHistory   0
BMI                     0
ABG                     0
Pulse                   0
SecondaryDiagnosis      0
dtype: int64
```

Handling the Outliers

Handling the Outliers in BMI. We first plot a boxplot to visualize the outliers.

In [23]:

```
BMI = 'BMI showing the outliers (datasets out of the interquartiles range*1.5)'
plt.figure(figsize=(8, 6))
HealthCareData.boxplot(column='BMI')
plt.title(f'Box Plot of {BMI}')
plt.show()
```



We then create Quartile1_BMI (First Quartile) and Quartile3_BMI (Third Quartile). Then we calculate the Inter Quartile Range

In [24]:

```
Quartile1_BMI = HealthCareData['BMI'].quantile(0.25)
Quartile3_BMI = HealthCareData['BMI'].quantile(0.75)
IQR_BMI = Quartile3_BMI - Quartile1_BMI
```

Further, we calculate the lower bound and upper bound for the same

In [25]:

```
lower_bound_BMI = Quartile1_BMI - 1.5 * IQR_BMI
upper_bound_BMI = Quartile3_BMI + 1.5 * IQR_BMI
```

After calculating the lower and upper bounds, we find the outliers which are the datapoints outside the bounds.

```
In [26]: BMI_Outliers = HealthCareData[(HealthCareData['BMI'] < lower_bound_BMI) | (HealthCa
```

To deal with outliers we can use the methods:

1) Cutting: The extreme numbers that you identify as outliers should be eliminated or truncated. Take care not to erase too much information.

2) Elaboration: Substitute more typical values for outliers. To achieve this, you can swap them out for the mean, median, or another quantitatively determined value.

3) Winsorization: The closest non-outlier values should be used in place of the extreme ones. For instance,

We unanimously decided to use Imputation to deal with the outliers. We used median to replace all the outliers in each numerical data.

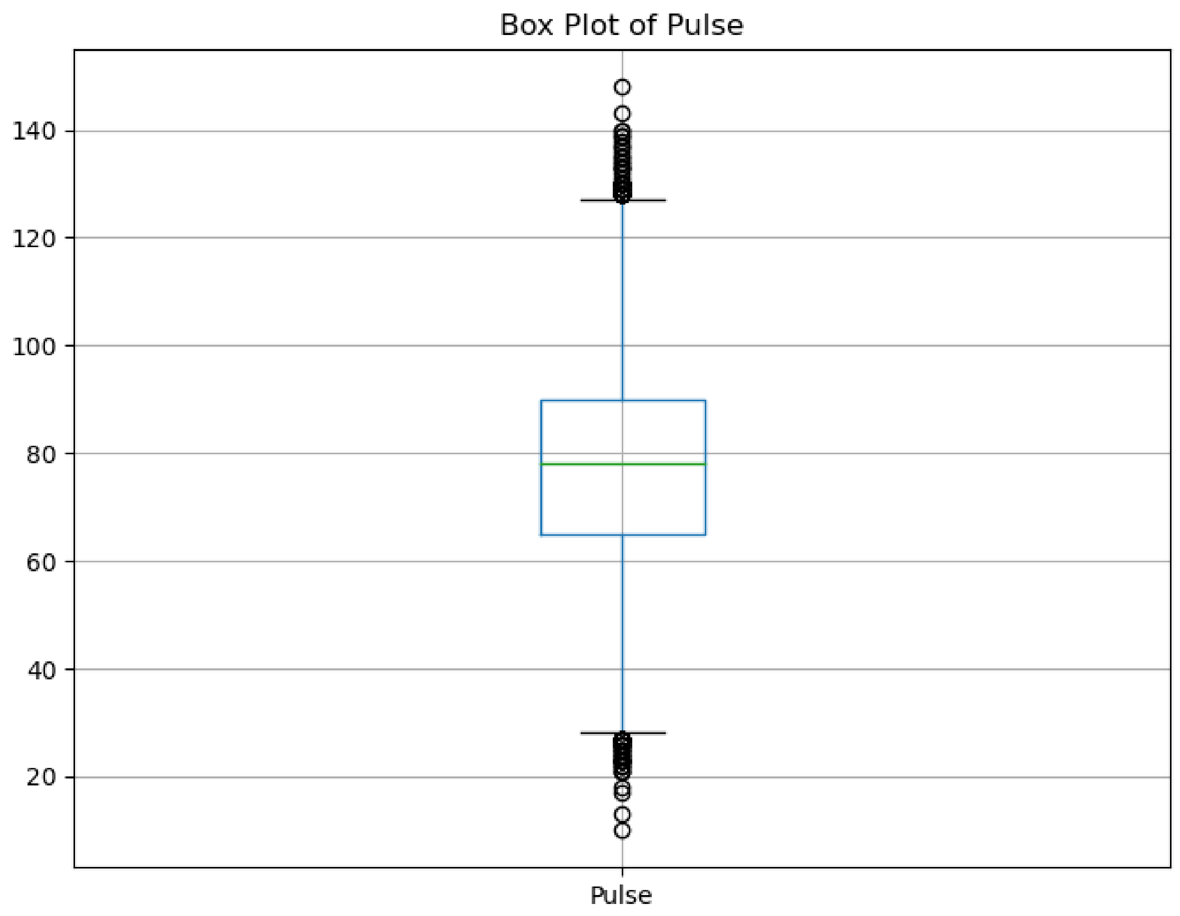
```
In [27]: # Remove BMI Outliers
         median_BMI = HealthCareData['BMI'].median()
         HealthCareData['BMI'][(HealthCareData['BMI'] < lower_bound_BMI) | (HealthCareData['
```

Calculating outliers for Pulse

```
In [28]: HealthCareData.Pulse.describe()

Out[28]: count    10759.000000
         mean        77.749419
         std         18.159736
         min         10.000000
         25%         65.000000
         50%         78.000000
         75%         90.000000
         max        148.000000
         Name: Pulse, dtype: float64
```

Creating boxplot for Pulse and calculating and replacing the outliers for the same.

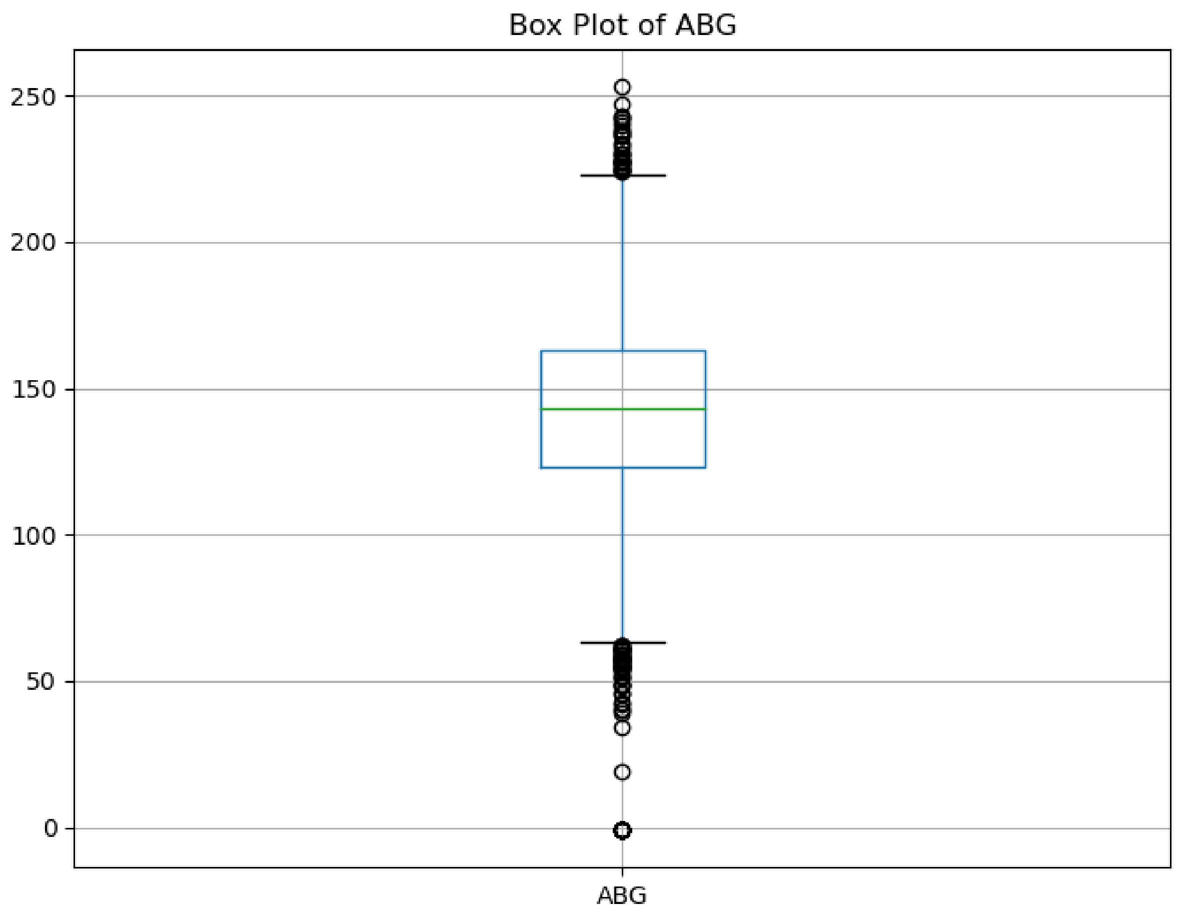
```
In [29]: Pulse = 'Pulse'
         plt.figure(figsize=(8, 6))
         HealthCareData.boxplot(column='Pulse')
         plt.title(f'Box Plot of {Pulse}')
         plt.show()


         Q1_P = HealthCareData['Pulse'].quantile(0.25)
         Q3_P = HealthCareData['Pulse'].quantile(0.75)
         IQR_P = Q3_P - Q1_P
         lower_bound_P = Q1_P - 1.5 * IQR_P
         upper_bound_P = Q3_P + 1.5 * IQR_P
         outliers_P = HealthCareData[(HealthCareData['Pulse'] < lower_bound_P) | (HealthCare
```



```
In [30]: # Remove Pulse Outliers
         median_P = HealthCareData['Pulse'].median()
         HealthCareData['Pulse'][(HealthCareData['Pulse'] < lower_bound_P) | (HealthCareData
```

Calculating outliers for Pulse

```
In [31]: ABG ='ABG'

         plt.figure(figsize=(8, 6))
         HealthCareData.boxplot(column='ABG')
         plt.title(f'Box Plot of {ABG}')
         plt.show()
```



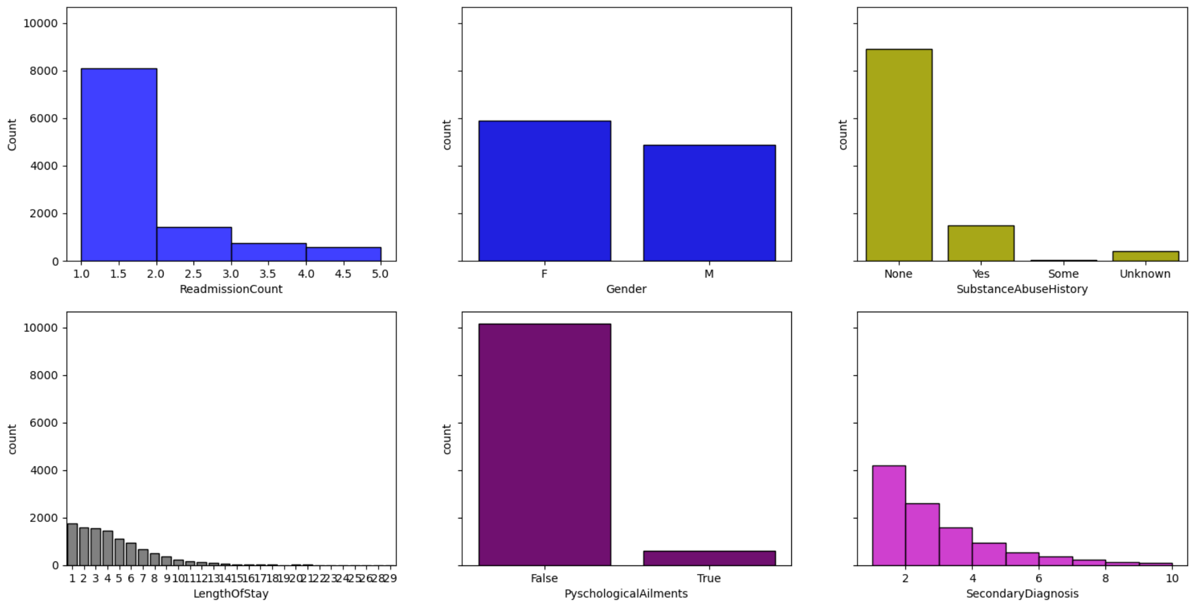
```
In [32]: Q1_A = HealthCareData['ABG'].quantile(0.25)
         Q3_A = HealthCareData['ABG'].quantile(0.75)
         IQR_A = Q3_A - Q1_A
         lower_bound_A = Q1_A - 1.5 * IQR_A
         upper_bound_A = Q3_A + 1.5 * IQR_A
         outliers_A = HealthCareData[(HealthCareData['ABG'] < lower_bound_A) | (HealthCareDa
```

```
In [33]: # Remove ABG Outliers
         median_ABG = HealthCareData['ABG'].median()
         HealthCareData['ABG'][(HealthCareData['ABG'] < lower_bound_A) | (HealthCareData['AE
```

Exploratory Data Analysis and Visualization

Analysis of univariate and bivariate data

```
In [34]: figure, ax = plt.subplots(2,3,figsize=(18,9),sharey = True)

         edge = [1, 2, 3, 4, 5]
         ax[0,0].set_title('',fontsize = 10)
         sns.histplot(x = 'ReadmissionCount', data = HealthCareData,ax = ax[0,0],color = 'b'

         ax[1,1].set_title('',fontsize = 10)
         sns.countplot(x = 'Gender', data = HealthCareData,ax = ax[0,1],color = 'b',edgecolo

         ax[1,0].set_title('',fontsize = 10)
         sns.countplot(x = 'SubstanceAbuseHistory', data = HealthCareData,ax = ax[0,2],color

         ax[1,0].set_title('',fontsize = 10)
         sns.countplot(x = 'LengthOfStay', data = HealthCareData,ax = ax[1,0],color='grey',e
ax[1,1].set_title('',fontsize = 10)
sns.countplot(x = 'PyschologicalAilments', data = HealthCareData,ax = ax[1,1],color

ax[1,2].set_title('',fontsize = 10)
edges_1 = [1,2,3,4,5,6,7,8,9,10]
sns.histplot(x = 'SecondaryDiagnosis', data = HealthCareData,ax = ax[1,2],color='m'

plt.show()
```



ReadmissionCount:

1. Hightest frquency is shown for ReadmissionCount more than 8000 (>75% of total population), which indicates that a majority of people were re-admitted.
2. Reasons of readmission once could be due to:
   (1) Unresolved medical conditions or complications: A patient's initial health issue might not always be totally resolved during their initial hospital stay. This may require a readmission for additional care and supervision.
   (2) Poor post-discharge care: Poor follow-up care or disregard for recommended drugs and treatment regimens can cause a patient's condition to worsen, necessitating readmission.

Gender:

1. There are two categories of Gender: Male and Female.When the two genders are plotted, we can see that the female count is greater than the male count by a range of more than 50%
2. The contribution of Male Gender is 4880 out of 10759 in total.
3. The inbalance in the genders could be due to Pregnancy and Maternity Care, Cultural and Societal Factors, etc.

SubstanceAbuseHistory: SubstanceAbuseHistory population data is divided into 4 samples

1. None: None data population have highest frequency
2. Yes: Yes data population have greater frequency than unknown
3. Some: Some data population have negligible frequency

4. Unkown: Unkown data population have less frequency than yes

LengthOfStay:A patient's duration of stay in the hospital may be prolonged for a variety of reasons. Some of these causes are as follows:

1. Lengthofstay is approx 2000 for patience who length of stay is 1 day
2. Count frequency reduce slowly from 2 lengthofstay
3. Illness Severity: Patients with more severe or complicated medical issues sometimes require lengthier hospital stays.
4. Complications: Complications that arise during therapy might result in a longer stay. Complications might include infections, severe drug responses.

PyschologicalAilments: These could be defined as Mental Health disorders. Some psychological ailments include these observations:

1. More than 90% of the population stated False for Psychological Ailments.
2. About 5% state yes for the same. Reasons could be due to depression, anxiety disorders, bipolar disorder, etc.

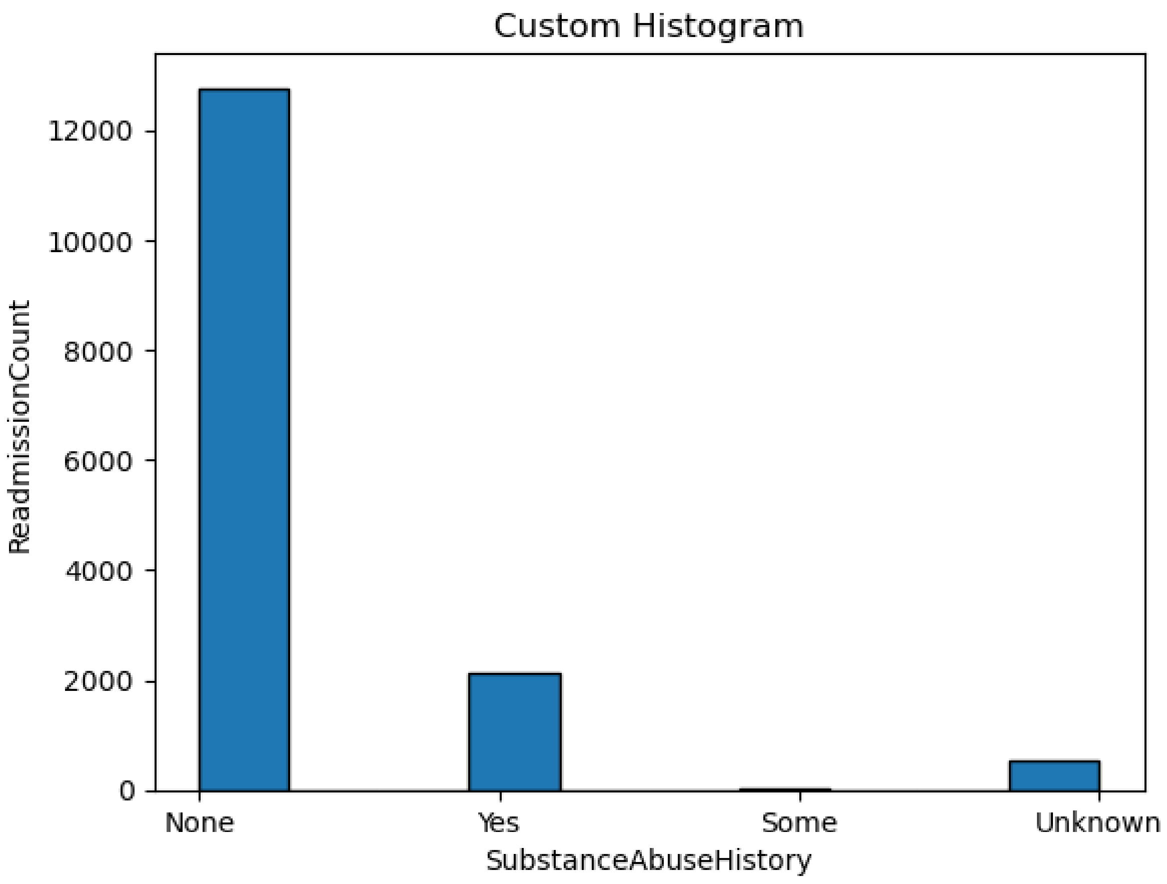
SecondaryDiagnosis:

1. The highest frequency can be observed in the first bin with the value of 4196.
2. There can be seen 11 unique values in total.
3. Some secondary diagnoses may signify a higher degree of medical complexity, necessitating a longer hospital stay for thorough care. Some secondary diseases could call for prolonged recuperation times, lengthening the hospital stay.

In [35]:
```
HealthCareData.PyschologicalAilments.describe()
```

Out[35]:
```
count     10759
unique        2
top       False
freq      10163
Name: PyschologicalAilments, dtype: object
```

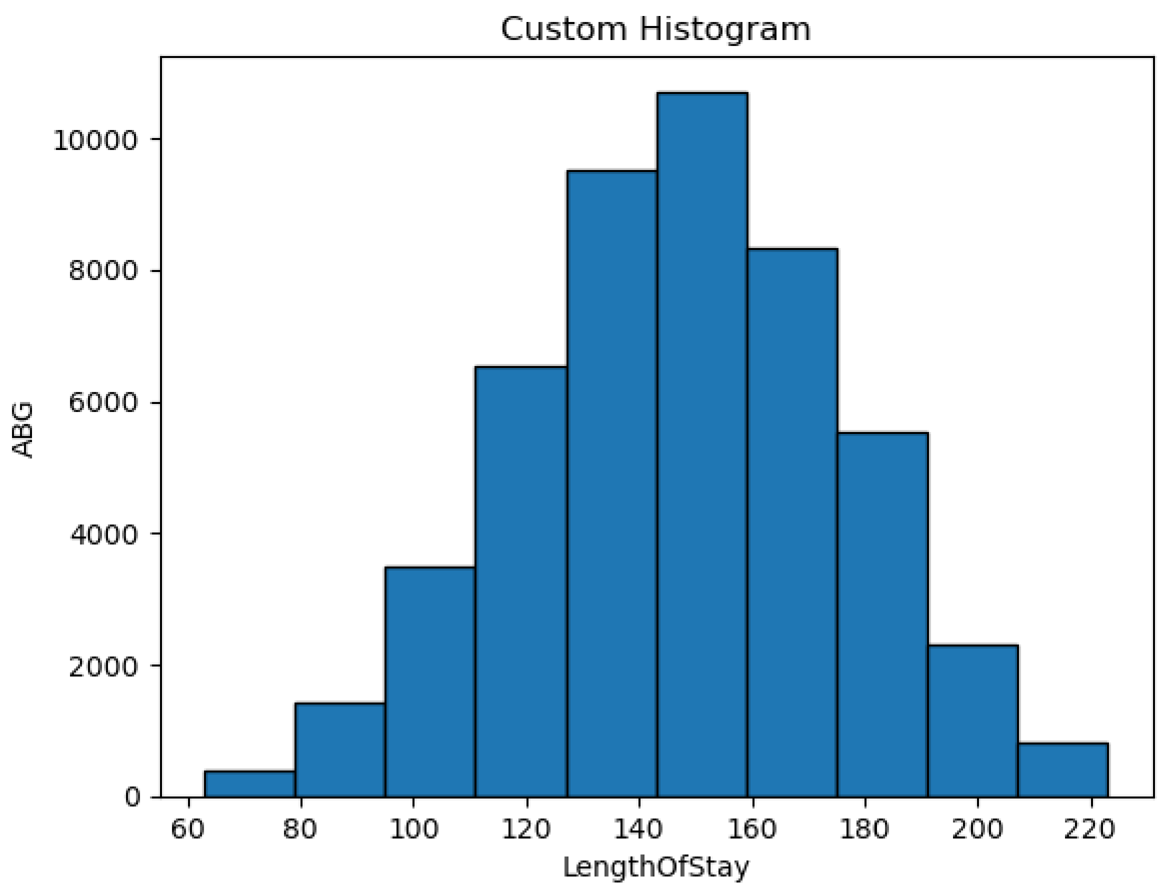
In [36]:
```
plt.hist(HealthCareData['SubstanceAbuseHistory'],weights = HealthCareData['Readmiss
plt.title("Custom Histogram")
plt.xlabel("SubstanceAbuseHistory")
plt.ylabel("ReadmissionCount")
plt.show()
```



ReadmissionCount vr SubstanceAbuseHistory: If you categories or partition data on substanceAbuseHistory

1. For yes we have 12000 readmissioncount
2. For other three readmissioncount drastical reduce
3. For yes its reduce to 2000
4. For some its approx to zero
5. For unkown its just beyond zero

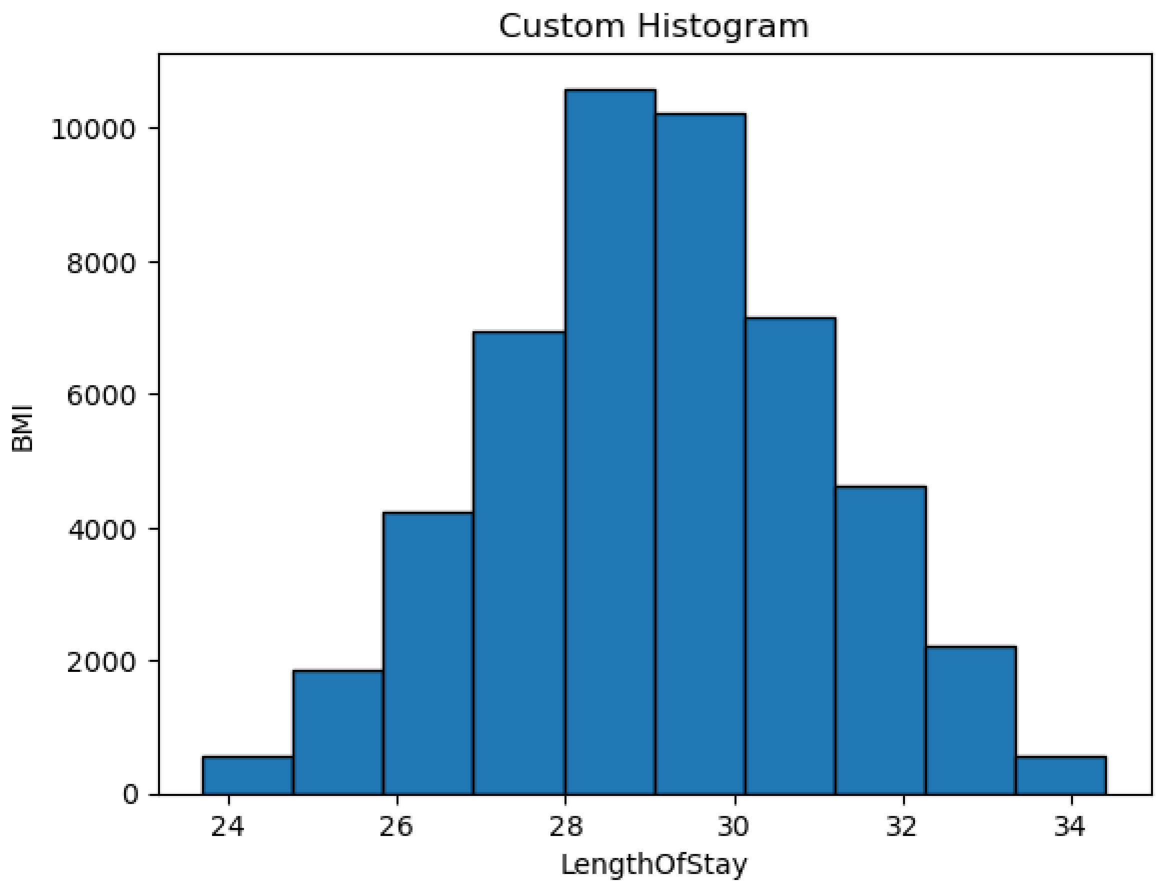
In [37]:
```
plt.hist(
         HealthCareData['ABG'],\
         weights = HealthCareData['LengthOfStay'],\
         edgecolor = 'black'
        )

plt.title("Custom Histogram")
plt.xlabel("LengthOfStay")
plt.ylabel("ABG")
plt.show()
```



When we plot the ABG (test for severe breathing and lung difficulties) against the length of stay, we can deduce:

1. If there is a high number of ABG values, the person's duration of stay is likewise high. However, that is not always the case.
2. The maximum number of patients admitted to the hospital is in the 140-160 range.

In [38]:
```
plt.hist(
         HealthCareData['BMI'],\
         weights = HealthCareData['LengthOfStay'],\
         edgecolor = 'black'
        )

plt.title("Custom Histogram")
plt.xlabel("LengthOfStay")
plt.ylabel("BMI")
plt.show()
```

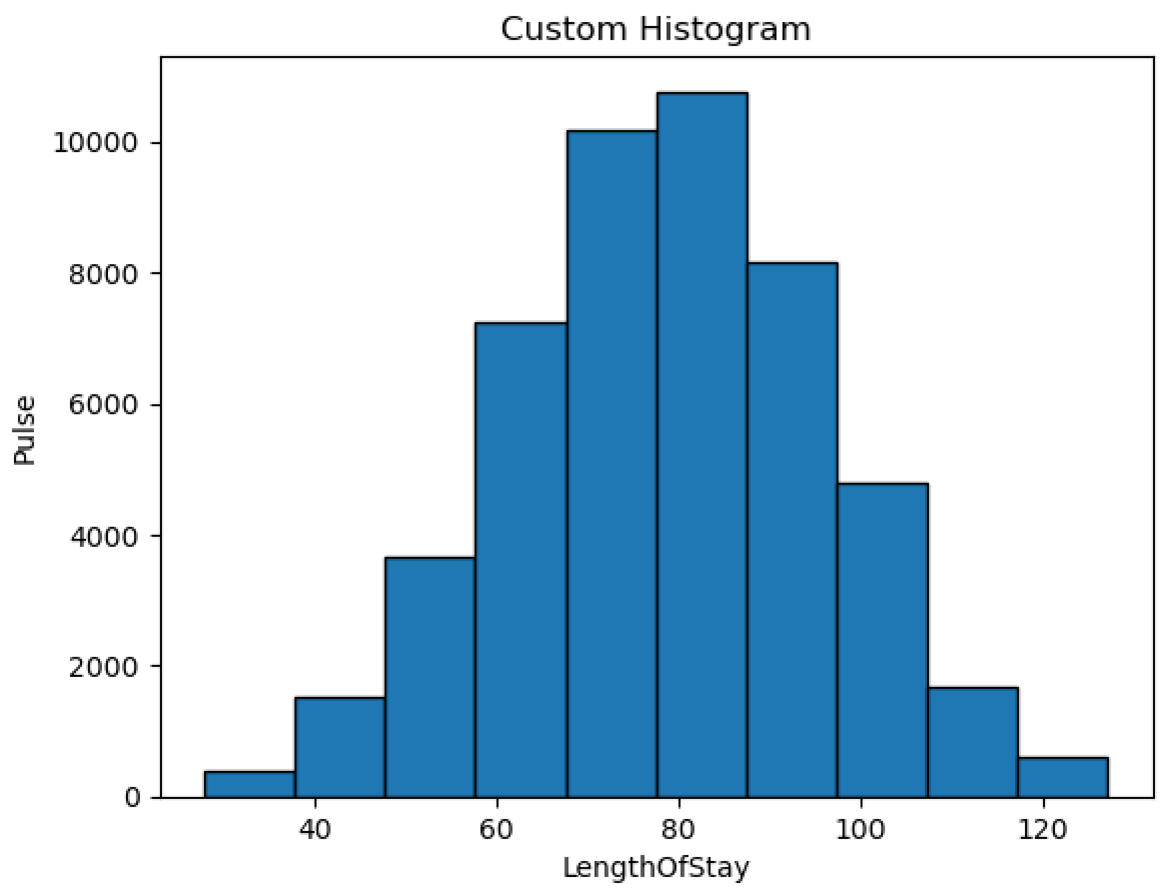


If user divide data as per lengthofstay

1. We have bins range with feature Lengthofstay, which is of range 24 to 34
2. Where we have highest BMI from 24 to 34 range
3. From 24 to 28 range of lengthofstay we have BMI range of 20 to 6500
4. From 30 to 34 range of lengthofstay we have BMI range of 6200 to 20

5. It seems that Lengthofstay vr BMI is uniformally distributed

In [39]:
```
plt.hist(
         HealthCareData['Pulse'],\
         weights = HealthCareData['LengthOfStay'],\
         edgecolor = 'black'
        )

plt.title("Custom Histogram")
plt.xlabel("LengthOfStay")
plt.ylabel("Pulse")
plt.show()
```



In many clinical conditions, the duration of stay in a hospital and a patient's pulse rate might be associated. The rate of your pulse in beats per minute (bpm).

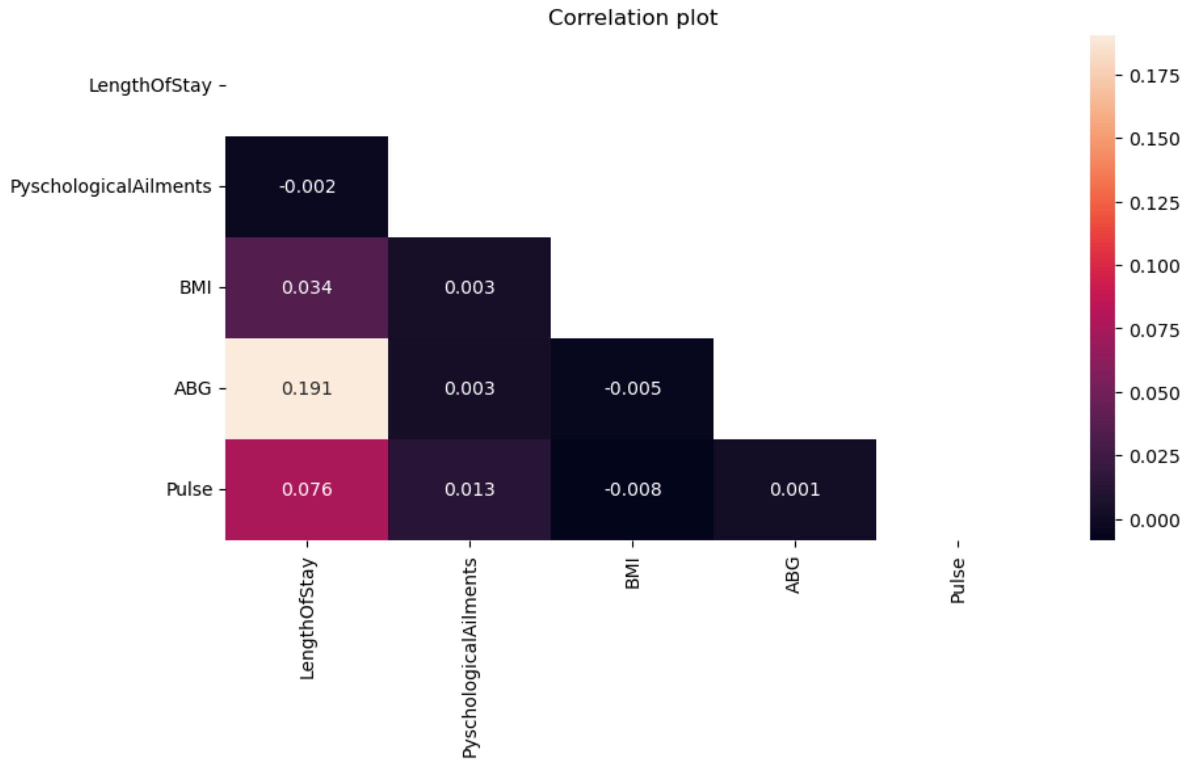
Several factors can alter the association between duration of stay and pulse rate:

1. Patients hospitalized with acute cardiovascular disorders, such as heart attacks, may have an increased or erratic pulse rate.
2. Post-Surgical Care: Following some procedures, particularly those involving the heart or major blood arteries, patients are continuously watched for symptoms of problems, such as irregular pulse rates.

As a result of the graph, we can deduce that in many circumstances, the length of stay is closely connected to pulse

Correlation plot to check for Multicollinearity and Identify the ideal features required to create the model on

In [40]:
```
plt.figure(figsize=(10,5))
corrvalue = HealthCareData.select_dtypes(exclude = 'object').corr()
matrix = np.triu(corrvalue)

sns.heatmap(corrvalue, annot=True, fmt=".3f",mask = matrix)
plt.title('Correlation plot')
plt.show()
```



Feature Engineering using hot-encoding

We have considered the categorical variables:

'Gender','SecondaryDiagnosis','PyschologicalAilments',' SubstanceAbuseHistory','ReadmissionCount'

```
In [41]: df_encoded = pd.get_dummies(HealthCareData, columns = ['Gender','SecondaryDiagnosis
                                                    'PyschologicalAilments','Subs


         df_encoded.columns
```

```
Out[41]: Index(['LengthOfStay', 'BMI', 'ABG', 'Pulse', 'Gender_F', 'Gender_M',
                'SecondaryDiagnosis_0', 'SecondaryDiagnosis_1', 'SecondaryDiagnosis_2',
                'SecondaryDiagnosis_3', 'SecondaryDiagnosis_4', 'SecondaryDiagnosis_5',
                'SecondaryDiagnosis_6', 'SecondaryDiagnosis_7', 'SecondaryDiagnosis_8',
                'SecondaryDiagnosis_9', 'SecondaryDiagnosis_10',
                'PyschologicalAilments_False', 'PyschologicalAilments_True',
                'SubstanceAbuseHistory_None', 'SubstanceAbuseHistory_Some',
                'SubstanceAbuseHistory_Unknown', 'SubstanceAbuseHistory_Yes',
                'ReadmissionCount_1', 'ReadmissionCount_2', 'ReadmissionCount_3',
                'ReadmissionCount_4', 'ReadmissionCount_5'],
               dtype='object')
```

Scaling and Fitting the Model

```
In [42]: target_column = 'LengthOfStay'
         X = df_encoded.drop(['LengthOfStay'],axis = 1)
         y = df_encoded['LengthOfStay']

         X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, test_siz

         # Scaling is done after train test split to prevent data leakage
         scaler = StandardScaler()
         X_train_sc = scaler.fit_transform(X_train)
         X_test_sc = scaler.transform(X_test)

         model = LinearRegression()
         model.fit(X_train_sc, y_train)
```

```
# Make predictions on the scaled test data
ypred_test = model.predict(X_test_sc)
ypred_train = model.predict(X_train_sc)
```

In [44]:
```
mse = mean_squared_error(y_test, ypred_test)
r2 = r2_score(y_test, ypred_test)

# Print evaluation metrics
print(f"Mean Squared Error : {mse}")
print(f"R-squared : {r2}")
```

```
Mean Squared Error : 1.4693039155776864
R-squared : 0.8718324358071616
```

We have received an accuracy of 87.18%

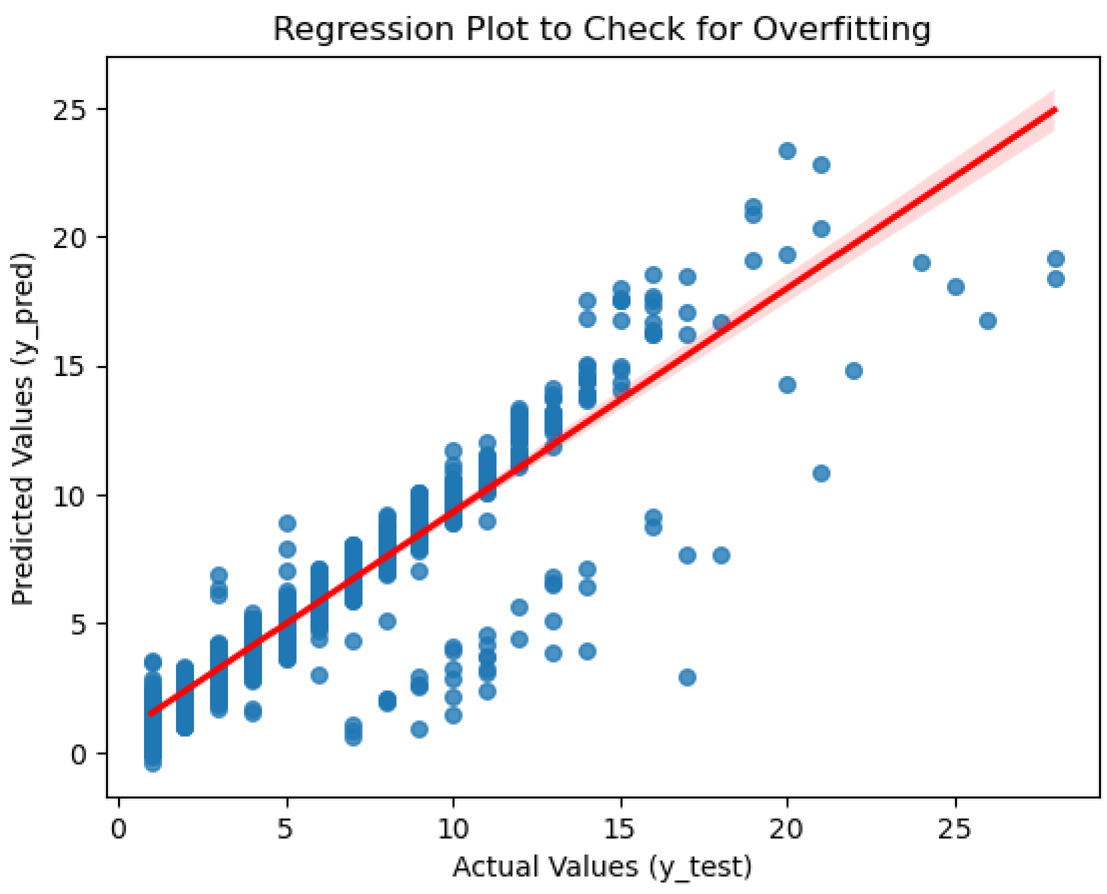
In [45]:
```
mse_train = mean_squared_error(y_train, ypred_train)
r2_train = r2_score(y_train, ypred_train)

print(f"Mean Squared Error : {mse_train}")
print(f"R-squared : {r2_train}")
```

```
Mean Squared Error : 1.5915167108363966
R-squared : 0.850890656649953
```

After fitting the model on the training data as well, we can see that the accuracy is 85.1%. The training model accuracy is slightly less than the testing model accuracy which indicates that the data is not overfitting. This is generally a desirable outcome because it suggests that the model is not simply memorizing the training data but is capable of generalizing to unseen data.

Plotting a Regression Plot to Re-check if the data is Underfitting or Overfitting or neither

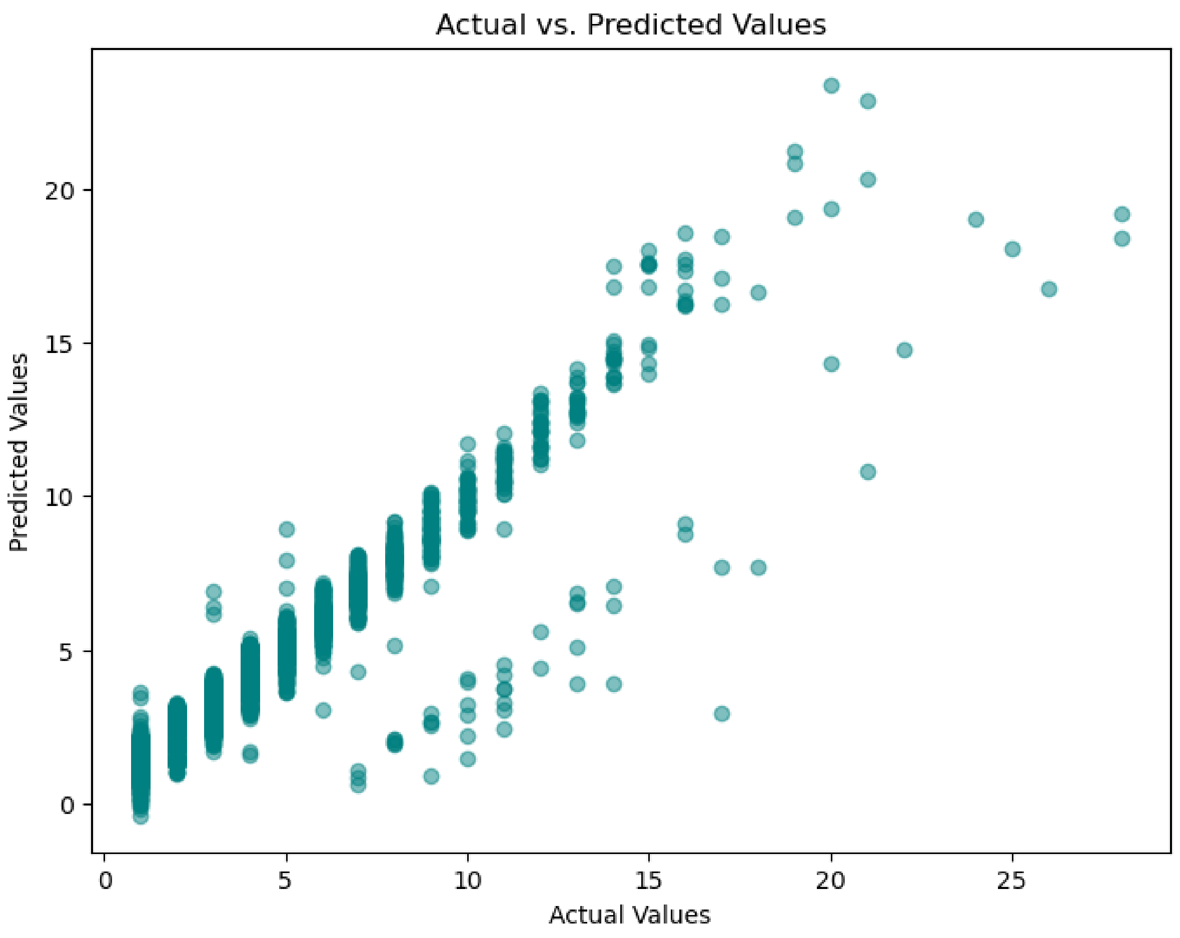
In [50]:
```
sns.regplot(y = ypred_test, x = y_test, scatter_kws={'s': 30}, line_kws={'color': '
plt.title("Regression Plot to Check for Overfitting")
plt.xlabel("Actual Values (y_test)")
plt.ylabel("Predicted Values (y_pred)")
plt.show()
```



Creating a Scatter plot of Actual vs Predicted Values

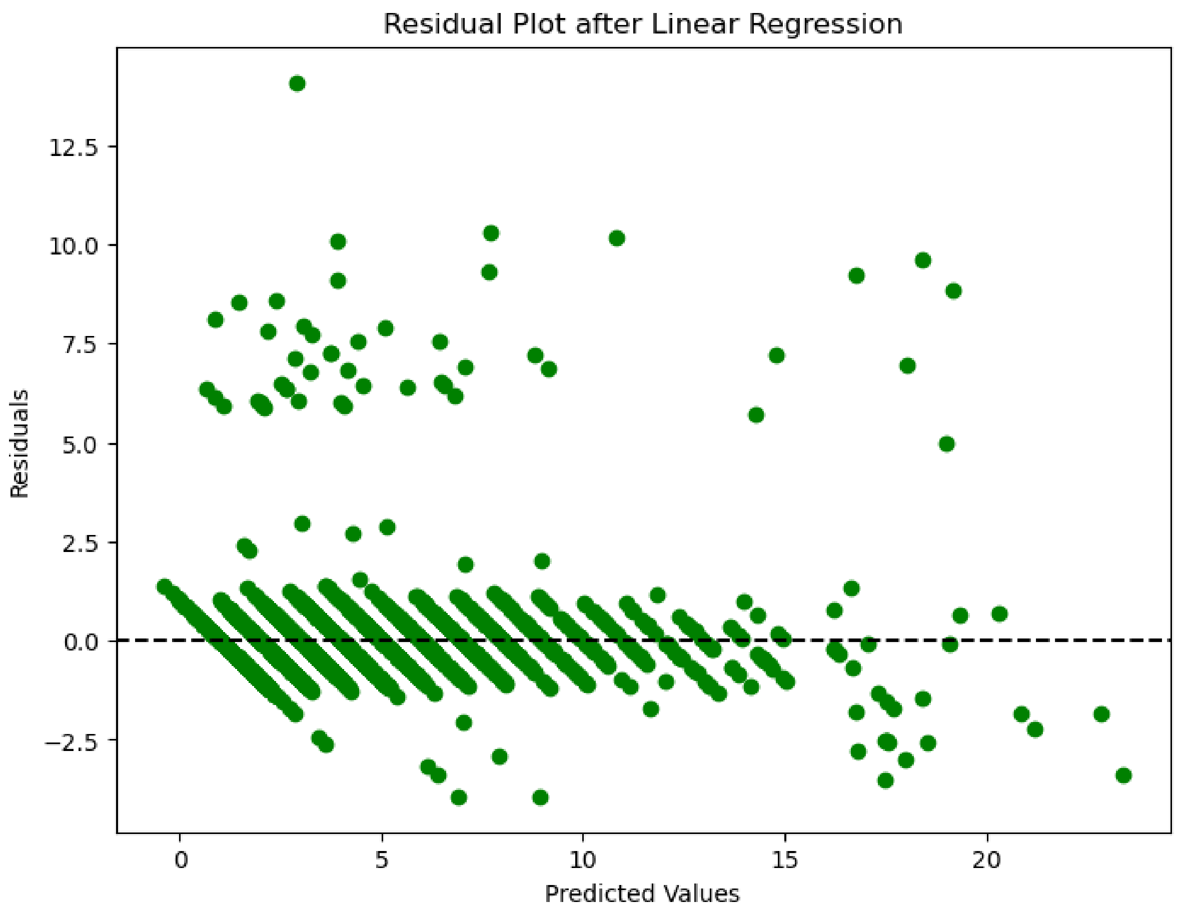
```
In [51]: plt.figure(figsize=(8, 6))
         plt.scatter(y_test, ypred_test, alpha=0.5,color='teal')
         plt.xlabel('Actual Values')
         plt.ylabel('Predicted Values')
         plt.title('Actual vs. Predicted Values')
```

```
Out[51]: Text(0.5, 1.0, 'Actual vs. Predicted Values')
```



Calculatting and creating a scatter plot of residuals

```
In [54]: residuals = y_test - ypred_test

         # Plotting residuals
         plt.figure(figsize=(8, 6))
         plt.scatter(ypred_test, residuals, color='green')
         plt.axhline(y=0, color='k', linestyle='--')
         plt.xlabel('Predicted Values')
         plt.ylabel('Residuals')
         plt.title('Residual Plot after Linear Regression')
         plt.show()
```



```
In [ ]: HealthCareData.to_csv('HealthCareData.csv', index=False)
```

```
In [ ]:
```