In [1]:
```
#Import the libraries needed
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sl
from statsmodels.formula.api import ols
```

In [2]:
```
#Load the dataset
df = pd.read_csv("medical clean 1.1.23.csv")
```

In [3]:
```
#Check total rows/columns
rows = df.shape[0]
cols = df.shape[1]
# display the number of rows and columns
print("Number of Rows: " + str(rows))
print("Number of Columns: " + str(cols))
```

```
Number of Rows: 10000
Number of Columns: 50
```

In [4]:
```
#check for nan and null values
df.isna().any()
```

Out[4]:
```
CaseOrder             False
Customer_id           False
Interaction           False
UID                   False
City                  False
State                 False
County                False
Zip                   False
Lat                   False
Lng                   False
Population            False
Area                  False
TimeZone              False
Job                   False
Children              False
Age                   False
Income                False
Marital               False
Gender                False
ReAdmis               False
VitD_levels           False
Doc_visits            False
Full_meals_eaten      False
vitD_supp             False
Soft_drink            False
Initial_admin         False
HighBlood             False
Stroke                False
Complication_risk     False
Overweight            False
Arthritis             False
Diabetes              False
Hyperlipidemia        False
BackPain              False
Anxiety               False
Allergic_rhinitis     False
Reflux_esophagitis    False
Asthma                False
Services              False
Initial_days          False
TotalCharge           False
Additional_charges    False
Item1                 False
Item2                 False
Item3                 False
Item4                 False
Item5                 False
Item6                 False
Item7                 False
Item8                 False
dtype: bool
```

In [5]:
```
df.isnull().any()
```

Out[5]:
```
CaseOrder             False
Customer_id           False
Interaction           False
UID                   False
City                  False
State                 False
County                False
Zip                   False
Lat                   False
Lng                   False
Population            False
Area                  False
TimeZone              False
Job                   False
Children              False
Age                   False
Income                False
Marital               False
Gender                False
ReAdmis               False
VitD_levels           False
Doc_visits            False
Full_meals_eaten      False
vitD_supp             False
Soft_drink            False
Initial_admin         False
HighBlood             False
Stroke                False
Complication_risk     False
Overweight            False
Arthritis             False
Diabetes              False
Hyperlipidemia        False
BackPain              False
Anxiety               False
Allergic_rhinitis     False
Reflux_esophagitis    False
Asthma                False
Services              False
Initial_days          False
TotalCharge           False
Additional_charges    False
Item1                 False
Item2                 False
Item3                 False
Item4                 False
Item5                 False
Item6                 False
Item7                 False
Item8                 False
dtype: bool
```

In [6]:
```
# using isnull() function
df.isnull()
```

Out[6]:

	CaseOrder	Customer_id	Interaction	UID	City	State	County	Zip	Lat	Lng	...	Tc
0	False	False	False	False	False	False	False	False	False	False	...	
1	False	False	False	False	False	False	False	False	False	False	...	
2	False	False	False	False	False	False	False	False	False	False	...	
3	False	False	False	False	False	False	False	False	False	False	...	
4	False	False	False	False	False	False	False	False	False	False	...	
...	...	...	...	...	...	...	...	...	...	...	...	
9995	False	False	False	False	False	False	False	False	False	False	...	
9996	False	False	False	False	False	False	False	False	False	False	...	
9997	False	False	False	False	False	False	False	False	False	False	...	
9998	False	False	False	False	False	False	False	False	False	False	...	
9999	False	False	False	False	False	False	False	False	False	False	...	

10000 rows × 50 columns

In [7]:
```
# using isnull() function
df.notnull()
```

Out[7]:

	CaseOrder	Customer_id	Interaction	UID	City	State	County	Zip	Lat	Lng	...	TotalC
0	True	True	True	True	True	True	True	True	True	True	...	
1	True	True	True	True	True	True	True	True	True	True	...	
2	True	True	True	True	True	True	True	True	True	True	...	
3	True	True	True	True	True	True	True	True	True	True	...	
4	True	True	True	True	True	True	True	True	True	True	...	
...	...	...	...	...	...	...	...	...	...	...	...	
9995	True	True	True	True	True	True	True	True	True	True	...	
9996	True	True	True	True	True	True	True	True	True	True	...	
9997	True	True	True	True	True	True	True	True	True	True	...	
9998	True	True	True	True	True	True	True	True	True	True	...	
9999	True	True	True	True	True	True	True	True	True	True	...	

10000 rows × 50 columns

In [8]:
```
# count of unique values in each column
print(df.nunique())
```

```
CaseOrder             10000
Customer_id           10000
Interaction           10000
UID                   10000
City                   6072
State                    52
County                 1607
Zip                    8612
Lat                    8588
Lng                    8725
Population             5951
Area                      3
TimeZone                 26
Job                     639
Children                 11
Age                      72
Income                 9993
Marital                   5
Gender                    3
ReAdmis                   2
VitD_levels            9976
Doc_visits                9
Full_meals_eaten          8
vitD_supp                 6
Soft_drink                2
Initial_admin             3
HighBlood                 2
Stroke                    2
Complication_risk         3
Overweight                2
Arthritis                 2
Diabetes                  2
Hyperlipidemia            2
BackPain                  2
Anxiety                   2
Allergic_rhinitis         2
Reflux_esophagitis        2
Asthma                    2
Services                  4
Initial_days           9997
TotalCharge            9997
Additional_charges     9418
Item1                     8
Item2                     7
Item3                     8
Item4                     7
Item5                     7
Item6                     7
Item7                     7
Item8                     7
dtype: int64
```

In [9]:
```
#check data types
df.dtypes
```

Out[9]:
```
CaseOrder              int64
Customer_id           object
Interaction           object
UID                   object
City                  object
State                 object
County                object
Zip                    int64
Lat                  float64
Lng                  float64
Population             int64
Area                  object
TimeZone              object
Job                   object
Children               int64
Age                    int64
Income               float64
Marital               object
Gender                object
ReAdmis               object
VitD_levels          float64
Doc_visits             int64
Full_meals_eaten       int64
vitD_supp              int64
Soft_drink            object
Initial_admin         object
HighBlood             object
Stroke                object
Complication_risk     object
Overweight            object
Arthritis             object
Diabetes              object
Hyperlipidemia        object
BackPain              object
Anxiety               object
Allergic_rhinitis     object
Reflux_esophagitis    object
Asthma                object
Services              object
Initial_days         float64
TotalCharge          float64
Additional_charges   float64
Item1                  int64
Item2                  int64
Item3                  int64
Item4                  int64
Item5                  int64
Item6                  int64
Item7                  int64
Item8                  int64
dtype: object
```

In [10]:
```
#Change object to category for ReAdmis
df["ReAdmis"] = df["ReAdmis"].astype('category')
#Change object to category for Anxiety
df["Anxiety"] = df["Anxiety"].astype('category')
#Change object to category for Services
df["Services"] = df["Services"].astype('category')
#Change object to category for Complication Risk
df["Complication_risk"] = df["Complication_risk"].astype('category')
#Change object to category for Overweight
df["Overweight"] = df["Overweight"].astype('category')
#Change object to category for Initial_admin
df["Initial_admin"] = df["Initial_admin"].astype('category')
#Change object to category for Arthritis
df["Arthritis"] = df["Arthritis"].astype('category')
#Change object to category for HighBlood
df["HighBlood"] = df["HighBlood"].astype('category')
#Change object to category for Soft_drink
df["Soft_drink"] = df["Soft_drink"].astype('category')
#Change object to category for Stroke
df["Stroke"] = df["Stroke"].astype('category')
#Change object to category for Gender
df["Gender"] = df["Gender"].astype('category')
#Change object to category for Diabetes
df["Diabetes"] = df["Diabetes"].astype('category')
df.dtypes
```

Out[10]:
```
CaseOrder              int64
Customer_id           object
Interaction           object
UID                   object
City                  object
State                 object
County                object
Zip                    int64
Lat                  float64
Lng                  float64
Population             int64
Area                  object
TimeZone              object
Job                   object
Children               int64
Age                    int64
Income               float64
Marital               object
Gender              category
ReAdmis             category
VitD_levels          float64
Doc_visits             int64
Full_meals_eaten       int64
vitD_supp              int64
Soft_drink          category
Initial_admin       category
HighBlood           category
Stroke              category
Complication_risk   category
Overweight          category
```

```
Arthritis             category
Diabetes              category
Hyperlipidemia          object
BackPain                object
Anxiety               category
Allergic_rhinitis       object
Reflux_esophagitis      object
Asthma                  object
Services              category
Initial_days           float64
TotalCharge            float64
Additional_charges     float64
Item1                    int64
Item2                    int64
Item3                    int64
Item4                    int64
Item5                    int64
Item6                    int64
Item7                    int64
Item8                    int64
dtype: object
```

In [11]:
```
#change floats to integers
df['Initial_days'] = df['Initial_days'].apply(np.int64)
df.dtypes
```

Out[11]:
```
CaseOrder              int64
Customer_id           object
Interaction           object
UID                   object
City                  object
State                 object
County                object
Zip                    int64
Lat                  float64
Lng                  float64
Population             int64
Area                  object
TimeZone              object
Job                   object
Children               int64
Age                    int64
Income               float64
Marital               object
Gender              category
ReAdmis             category
VitD_levels          float64
Doc_visits             int64
Full_meals_eaten       int64
vitD_supp              int64
Soft_drink          category
Initial_admin       category
HighBlood           category
Stroke              category
Complication_risk   category
Overweight          category
Arthritis           category
Diabetes            category
Hyperlipidemia        object
BackPain              object
Anxiety             category
Allergic_rhinitis     object
Reflux_esophagitis    object
Asthma                object
Services            category
Initial_days           int64
TotalCharge          float64
Additional_charges   float64
Item1                  int64
Item2                  int64
Item3                  int64
Item4                  int64
Item5                  int64
Item6                  int64
Item7                  int64
Item8                  int64
dtype: object
```

In [12]:
```
#drop all columns and rows not being used
to_drop = ['CaseOrder', 'Customer_id','Interaction', 'UID', 'City', 'State', 'Cou
df.drop(to_drop, inplace=True, axis=1)
```

In [13]:
```
#Run get dummies on categorical
pd.get_dummies(df, columns = ['ReAdmis', 'Anxiety', 'Services', 'Complication_ris
```

Out[13]:

	Age	Doc_visits	Initial_days	ReAdmis_No	ReAdmis_Yes	Anxiety_No	Anxiety_Yes	Services
0	53	6	10	1	0	0	1	
1	51	4	15	1	0	1	0	
2	53	4	4	1	0	1	0	
3	78	4	1	1	0	1	0	
4	22	5	1	1	0	1	0	
...	...	...	...	...	...	...	...	
9995	25	4	51	1	0	0	1	
9996	87	5	68	0	1	1	0	
9997	45	4	70	0	1	0	1	
9998	43	5	63	0	1	1	0	
9999	70	5	70	0	1	1	0	

10000 rows × 32 columns

In [14]:
```
#create new df with the get dummies responses
dfupdated = pd.get_dummies(df, columns = ['ReAdmis', 'Anxiety', 'Services', 'Over
```

In [15]:
```
#Drop columns to avoid dummy variable trap
to_drop = ['ReAdmis_No', 'Anxiety_No', 'Services_CT Scan', 'Services_Blood Work',
dfupdated.drop(to_drop, inplace=True, axis=1)
```

In [16]:
```
#RUN VIF
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [17]:
```
#find design matrix for linear regression model using 'Initial_days' as response
y, X = dmatrices('Initial_days ~ Age+Doc_visits', data=dfupdated, return_type='da
```

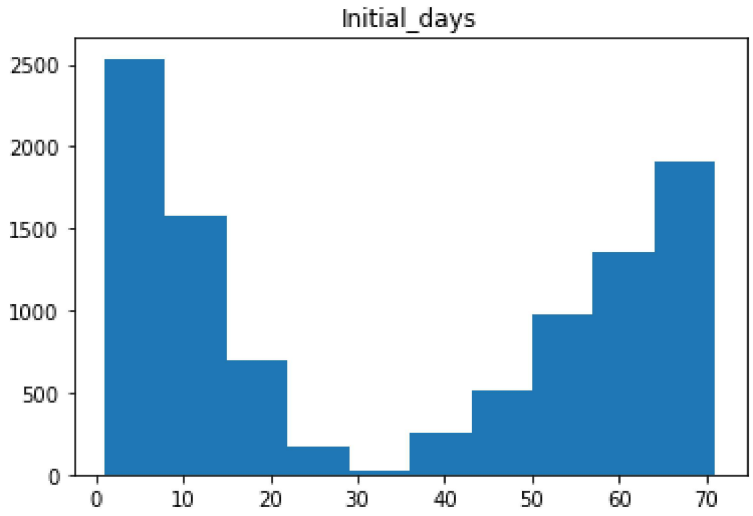
In [18]:
```
#calculate VIF for each explanatory variable
vif = pd.DataFrame()
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['variable'] = X.columns

#view VIF for each explanatory variable
vif
```

Out[18]:

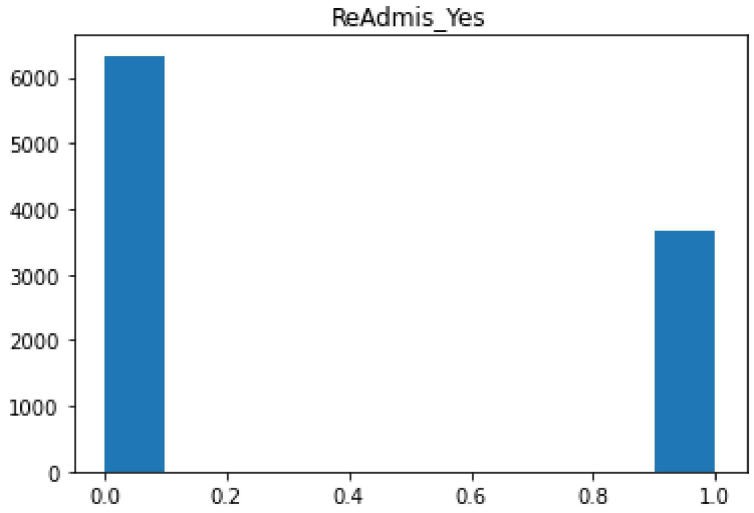
	VIF	variable
0	30.528353	Intercept
1	1.000048	Age
2	1.000048	Doc_visits

In [19]:
```
#export data and save file
dfupdated.to_csv('Documents/PreparedDataSubmit.csv')
```

In [20]:
```
#Create Histograms
plt.hist(dfupdated["Initial_days"])
plt.title("Initial_days");
```

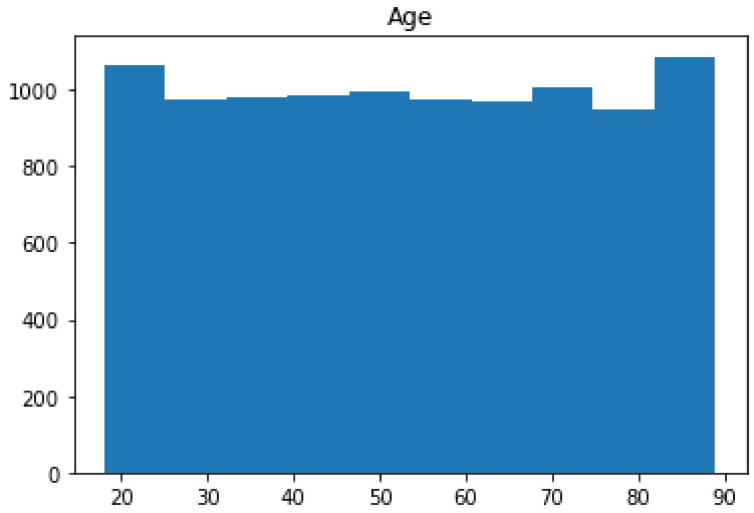


In [21]:
```
plt.hist(dfupdated["ReAdmis_Yes"])
plt.title("ReAdmis_Yes");
```



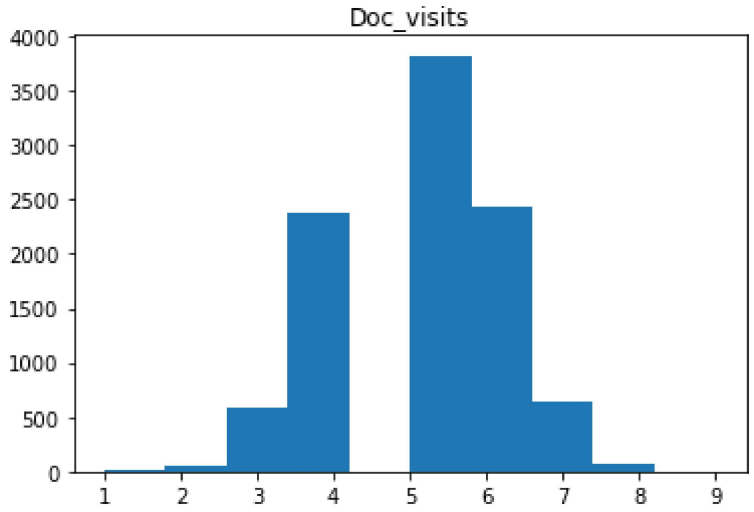
In [22]:
```
plt.hist(dfupdated["Age"])
plt.title("Age")
```

Out[22]: Text(0.5, 1.0, 'Age')



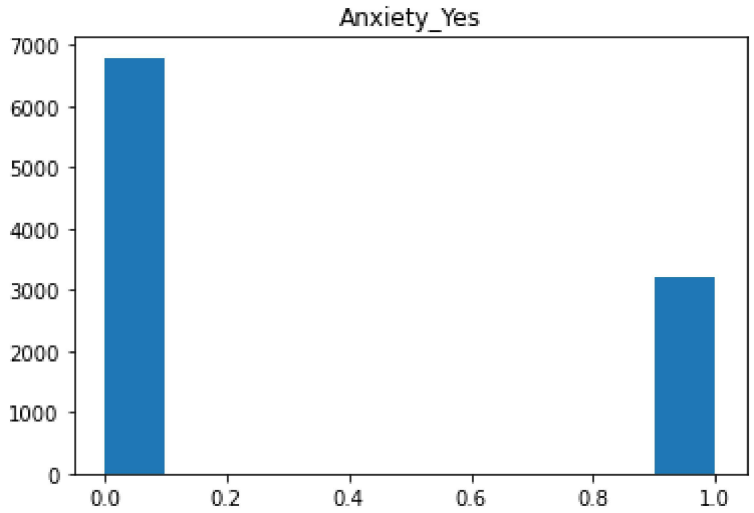
In [23]:
```
plt.hist(dfupdated["Doc_visits"])
plt.title("Doc_visits")
```

Out[23]:
```
Text(0.5, 1.0, 'Doc_visits')
```



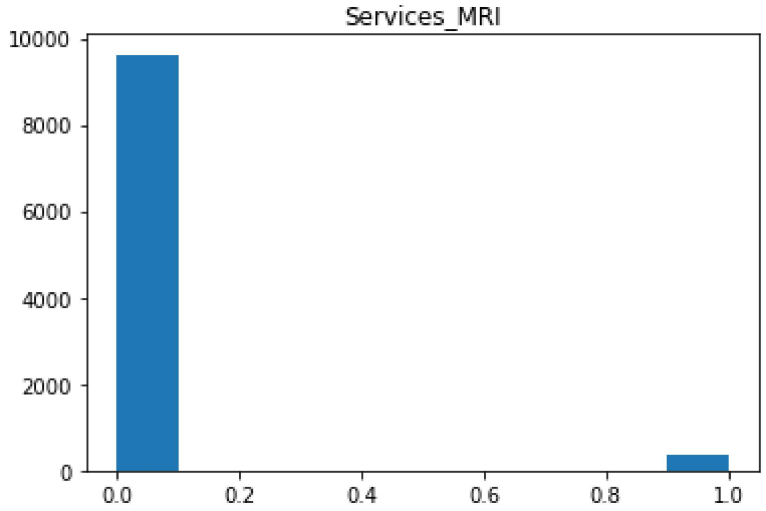
In [24]:
```
plt.hist(dfupdated["Anxiety_Yes"])
plt.title("Anxiety_Yes")
```

Out[24]:
```
Text(0.5, 1.0, 'Anxiety_Yes')
```



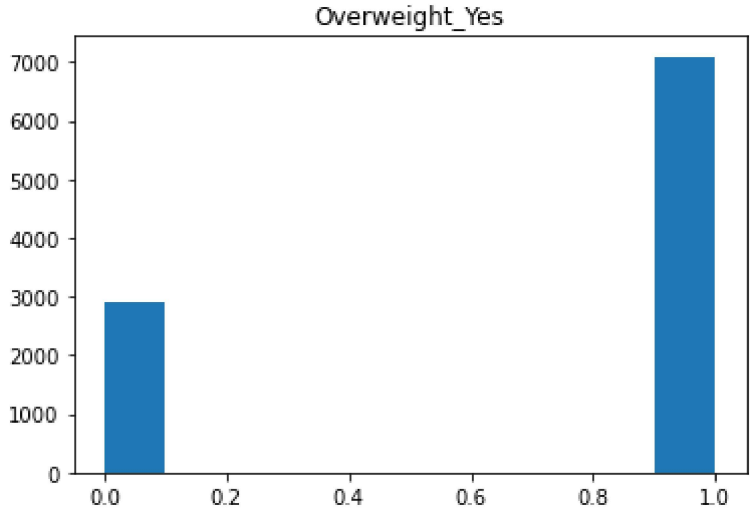
In [25]:
```
plt.hist(dfupdated["Services_MRI"])
plt.title("Services_MRI")
```

Out[25]:
```
Text(0.5, 1.0, 'Services_MRI')
```



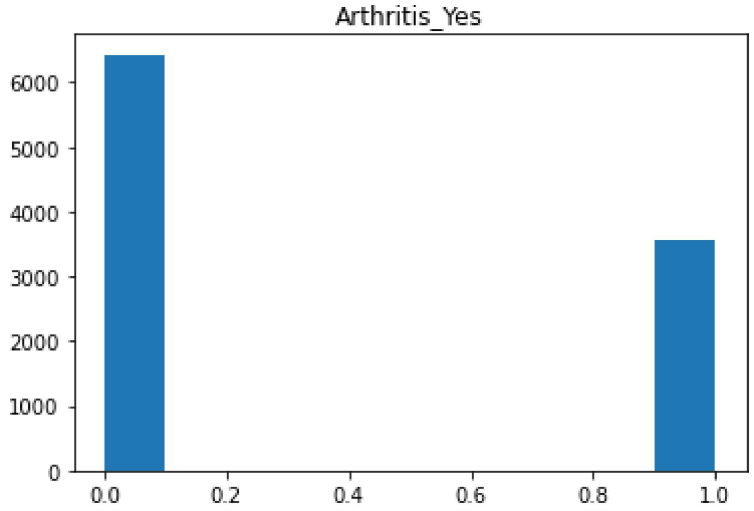
In [26]:
```
plt.hist(dfupdated["Overweight_Yes"])
plt.title("Overweight_Yes")
```

Out[26]:
```
Text(0.5, 1.0, 'Overweight_Yes')
```



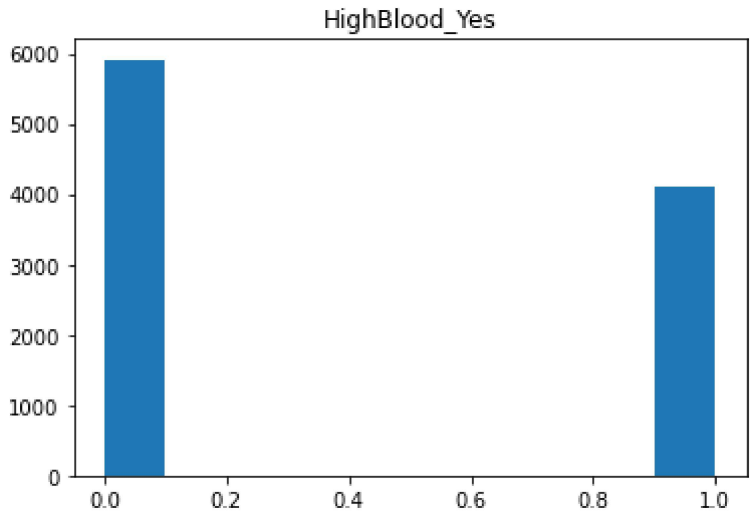
In [27]:
```
plt.hist(dfupdated["Arthritis_Yes"])
plt.title("Arthritis_Yes")
```

Out[27]:
```
Text(0.5, 1.0, 'Arthritis_Yes')
```



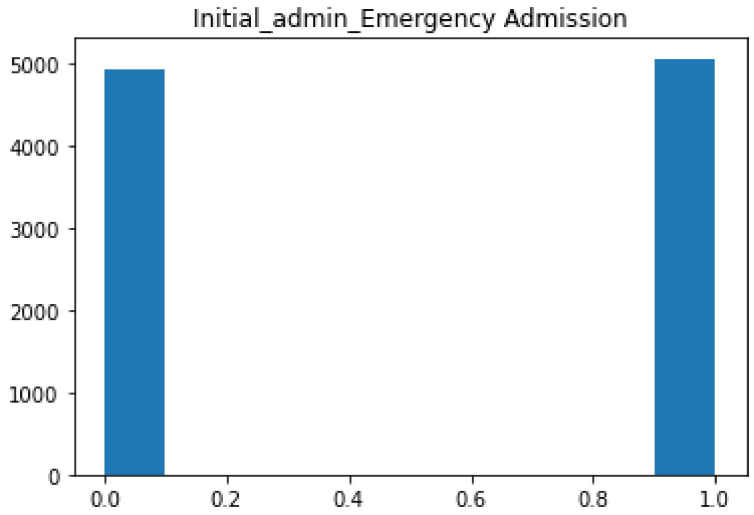
In [28]:
```
plt.hist(dfupdated["HighBlood_Yes"])
plt.title("HighBlood_Yes")
```

Out[28]:
```
Text(0.5, 1.0, 'HighBlood_Yes')
```



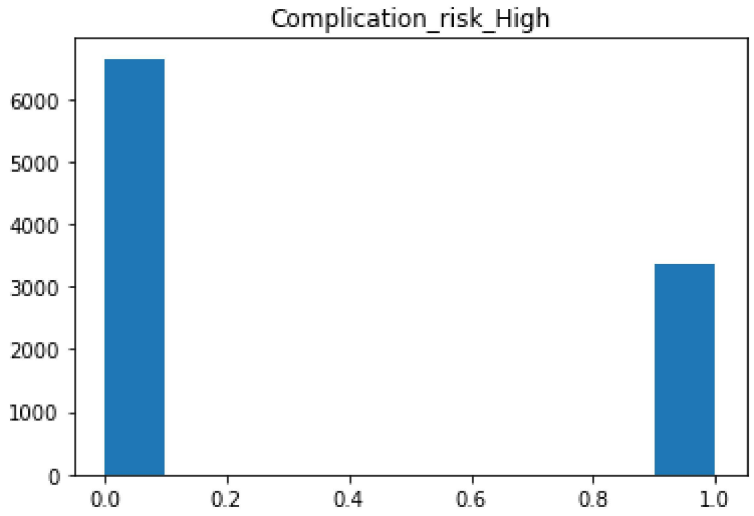
In [29]:
```
plt.hist(dfupdated["Initial_admin_Emergency Admission"])
plt.title("Initial_admin_Emergency Admission")
```

Out[29]:
```
Text(0.5, 1.0, 'Initial_admin_Emergency Admission')
```



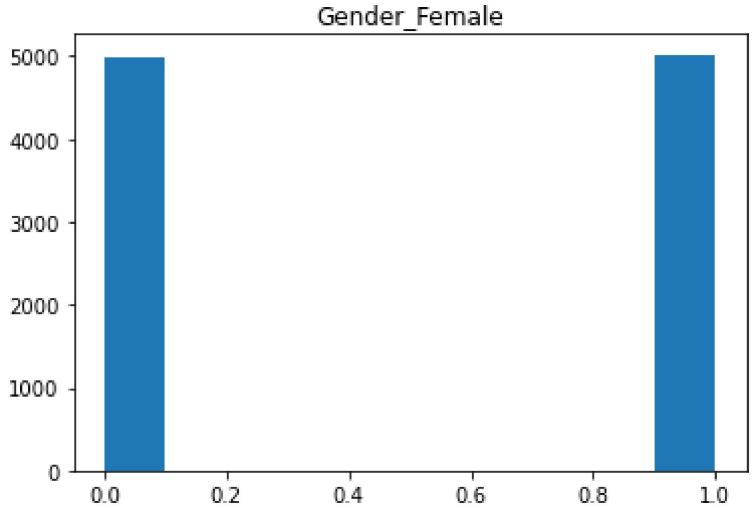
In [30]:
```
plt.hist(dfupdated["Complication_risk_High"])
plt.title("Complication_risk_High")
```

Out[30]:
```
Text(0.5, 1.0, 'Complication_risk_High')
```



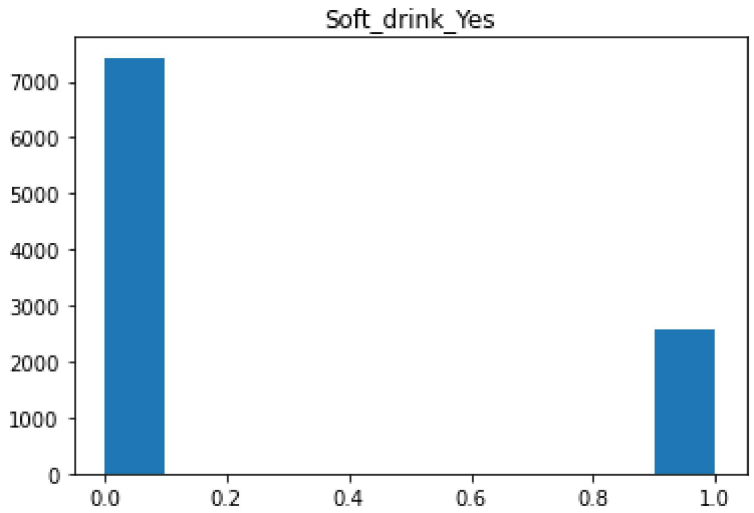
In [31]:
```
plt.hist(dfupdated["Gender_Female"])
plt.title("Gender_Female")
```

Out[31]:
```
Text(0.5, 1.0, 'Gender_Female')
```



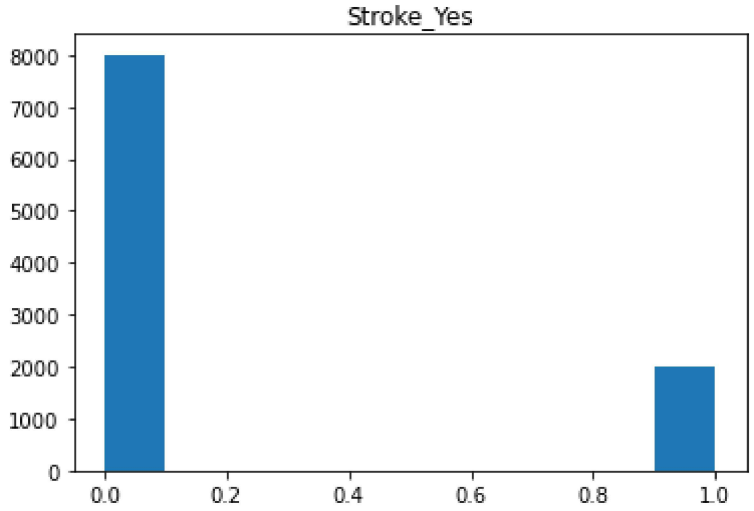
In [32]:
```
plt.hist(dfupdated["Soft_drink_Yes"])
plt.title("Soft_drink_Yes")
```

Out[32]:
```
Text(0.5, 1.0, 'Soft_drink_Yes')
```



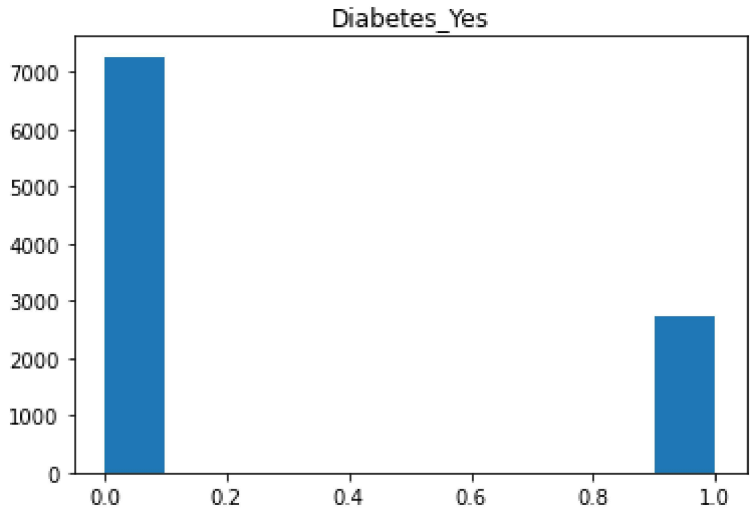
In [33]:
```
plt.hist(dfupdated["Stroke_Yes"])
plt.title("Stroke_Yes")
```

Out[33]: Text(0.5, 1.0, 'Stroke_Yes')

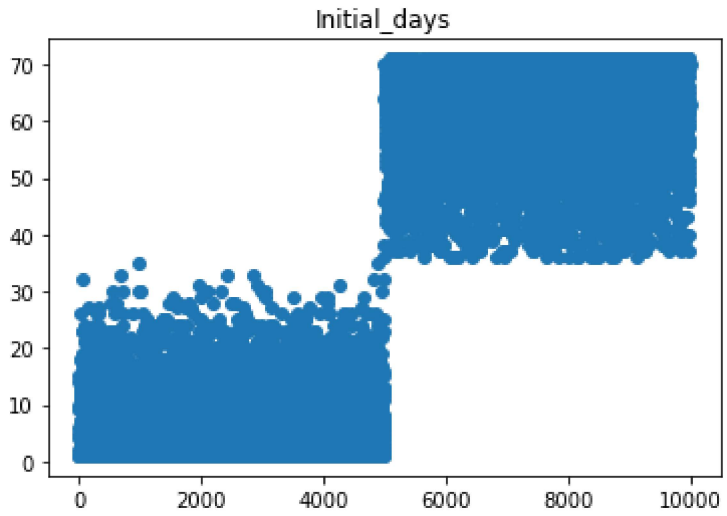


In [34]:
```
plt.hist(dfupdated["Diabetes_Yes"])
plt.title("Diabetes_Yes")
```

Out[34]: Text(0.5, 1.0, 'Diabetes_Yes')

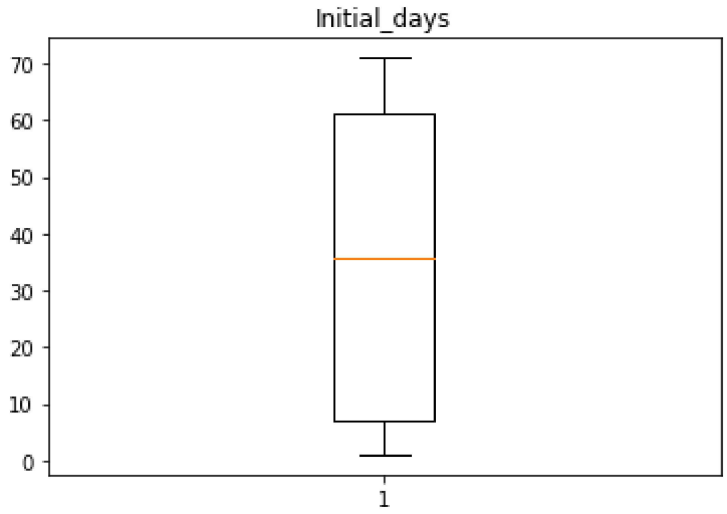


In [35]:
```
#Scatterplot for Initial Days
plt.scatter(dfupdated.index,dfupdated["Initial_days"])
plt.title("Initial_days")
plt.show()
```

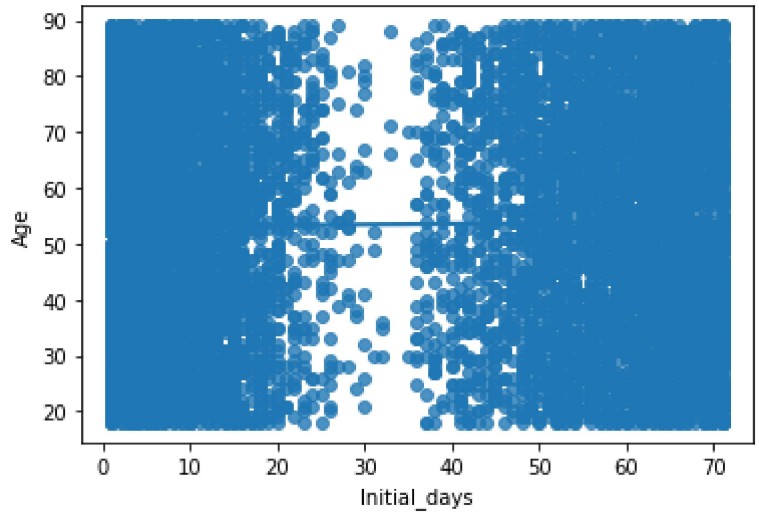


In [36]:
```
#Boxplot for initial days
plt.boxplot (dfupdated["Initial_days"])
plt.title("Initial_days")
```

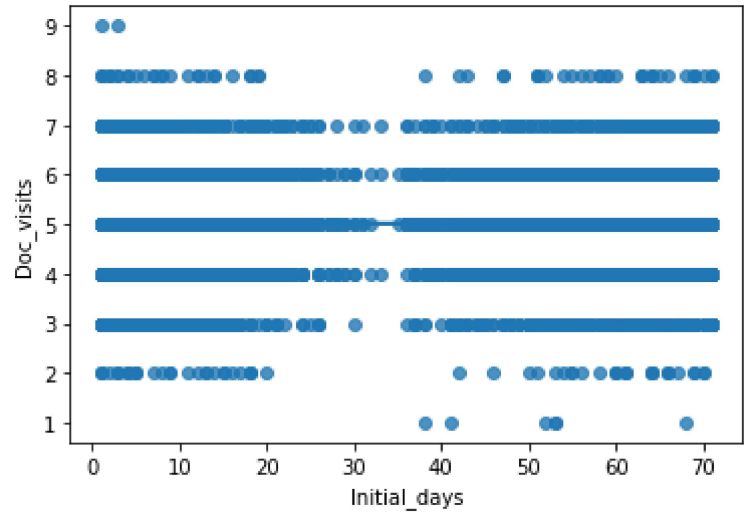
Out[36]: Text(0.5, 1.0, 'Initial_days')



In [37]:
```
#scatterplot for initial days and age with trend line
sns.regplot(x="Initial_days", y="Age", data=dfupdated, ci=None)
plt.show()
```

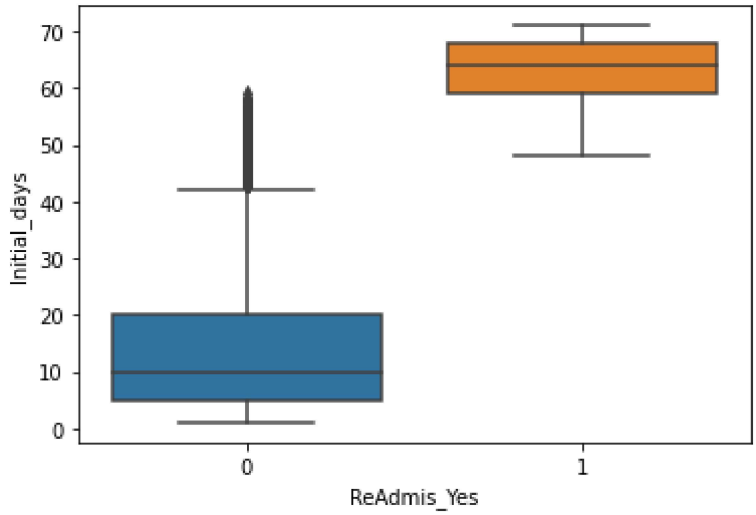


In [38]:
```
#scatterplot for initial days and doctor visits with trend line
sns.regplot(x="Initial_days", y="Doc_visits", data=dfupdated, ci=None)
plt.show()
```



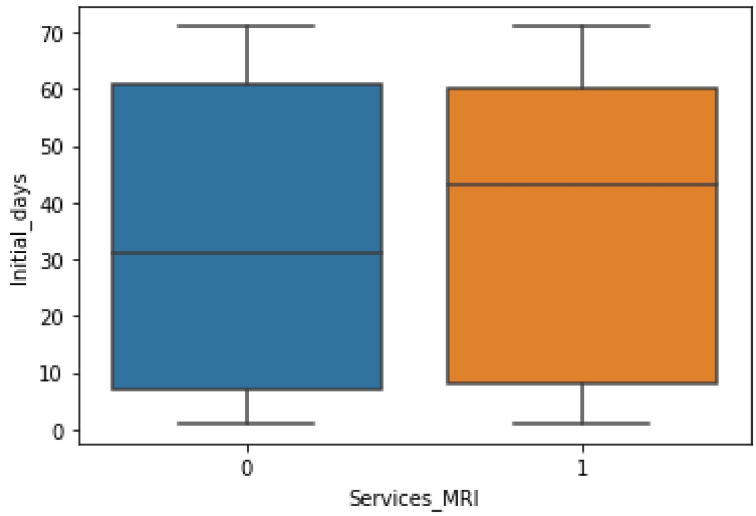
In [39]:
```
#visualizing numerical and categorical data with a box plot
sns.boxplot(data=dfupdated, x="ReAdmis_Yes", y="Initial_days")
```

Out[39]: <AxesSubplot:xlabel='ReAdmis_Yes', ylabel='Initial_days'>



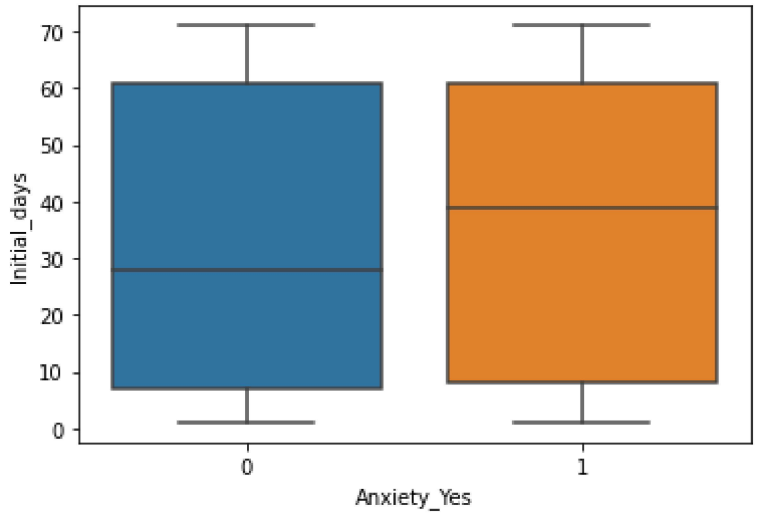
In [40]:
```
sns.boxplot(data=dfupdated, x="Services_MRI", y="Initial_days")
```

Out[40]: <AxesSubplot:xlabel='Services_MRI', ylabel='Initial_days'>



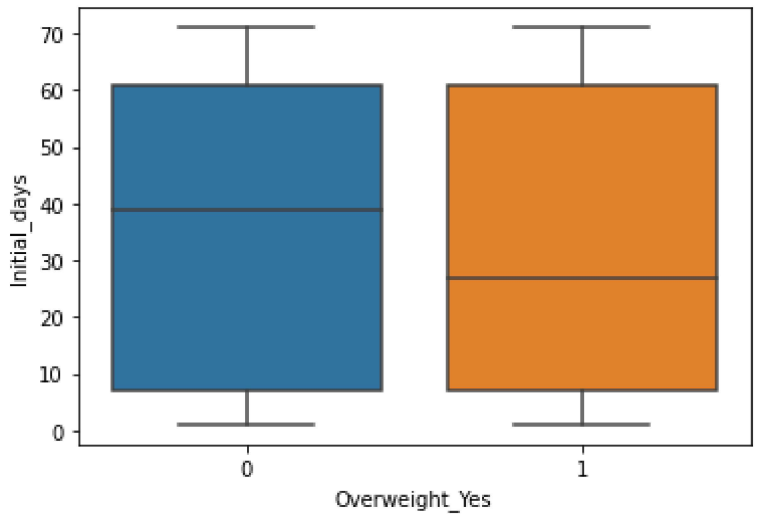
In [41]:
```
sns.boxplot(data=dfupdated, x="Anxiety_Yes", y="Initial_days")
```

Out[41]:
```
<AxesSubplot:xlabel='Anxiety_Yes', ylabel='Initial_days'>
```



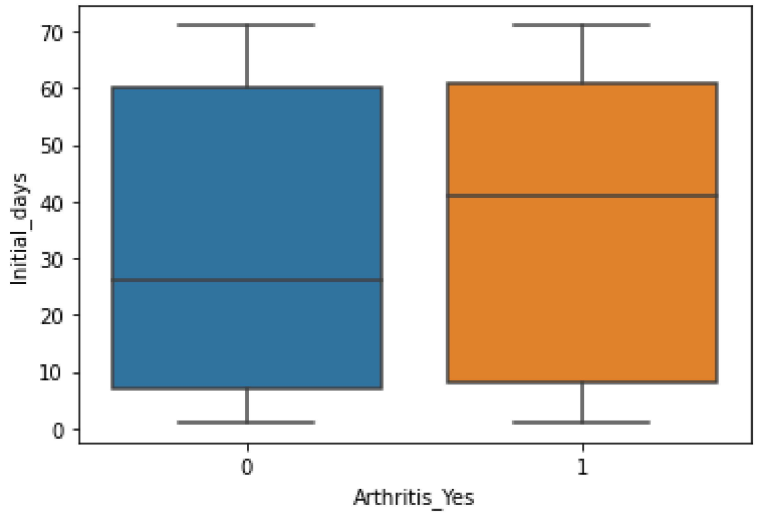
In [42]:
```
sns.boxplot(data=dfupdated, x="Overweight_Yes", y="Initial_days")
```

Out[42]:
```
<AxesSubplot:xlabel='Overweight_Yes', ylabel='Initial_days'>
```



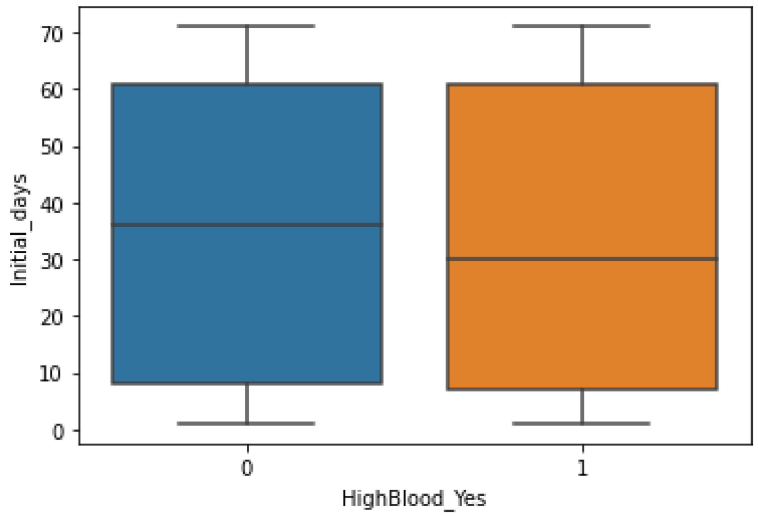
In [43]:
```
sns.boxplot(data=dfupdated, x="Arthritis_Yes", y="Initial_days")
```

Out[43]:
```
<AxesSubplot:xlabel='Arthritis_Yes', ylabel='Initial_days'>
```



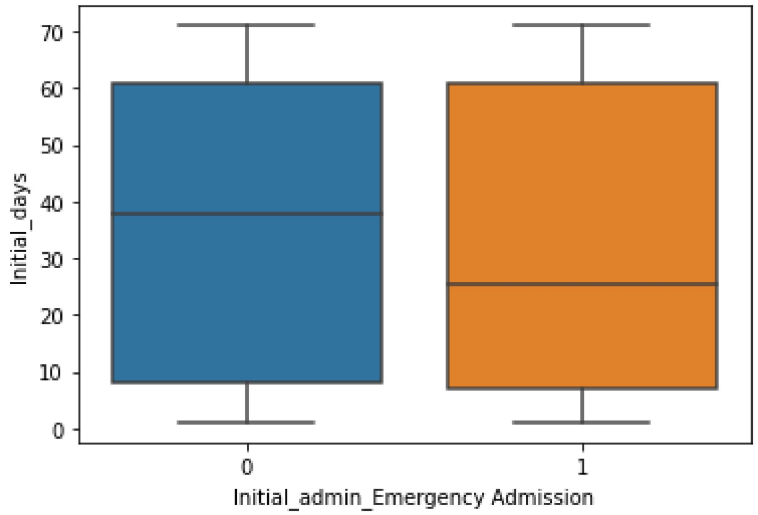
In [44]:
```
sns.boxplot(data=dfupdated, x="HighBlood_Yes", y="Initial_days")
```

Out[44]:
```
<AxesSubplot:xlabel='HighBlood_Yes', ylabel='Initial_days'>
```



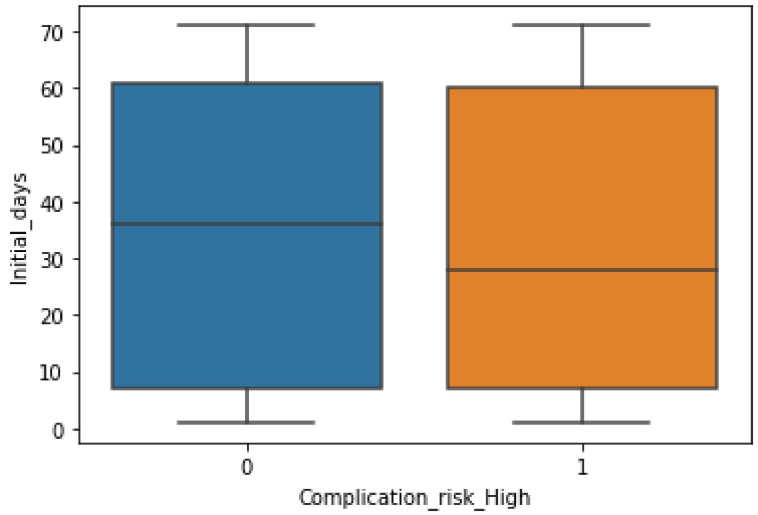
In [45]:
```
sns.boxplot(data=dfupdated, x="Initial_admin_Emergency Admission", y="Initial_da
```

Out[45]:
```
<AxesSubplot:xlabel='Initial_admin_Emergency Admission', ylabel='Initial_days'>
```



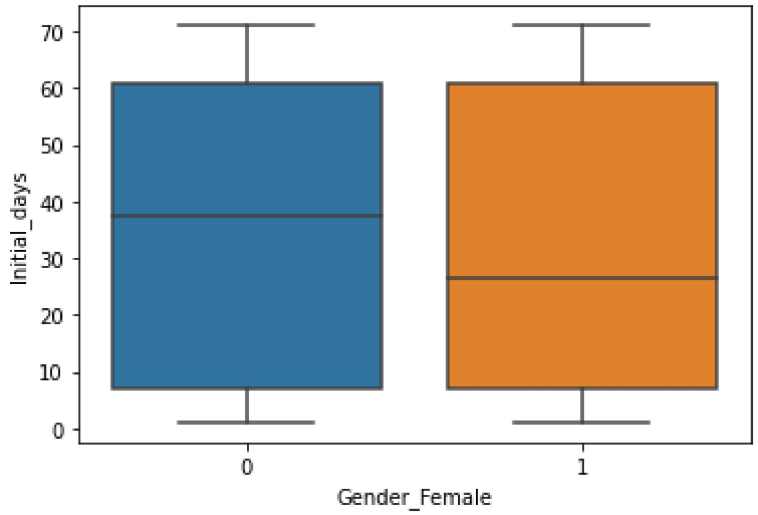
In [46]:
```
sns.boxplot(data=dfupdated, x="Complication_risk_High", y="Initial_days")
```

Out[46]:
```
<AxesSubplot:xlabel='Complication_risk_High', ylabel='Initial_days'>
```



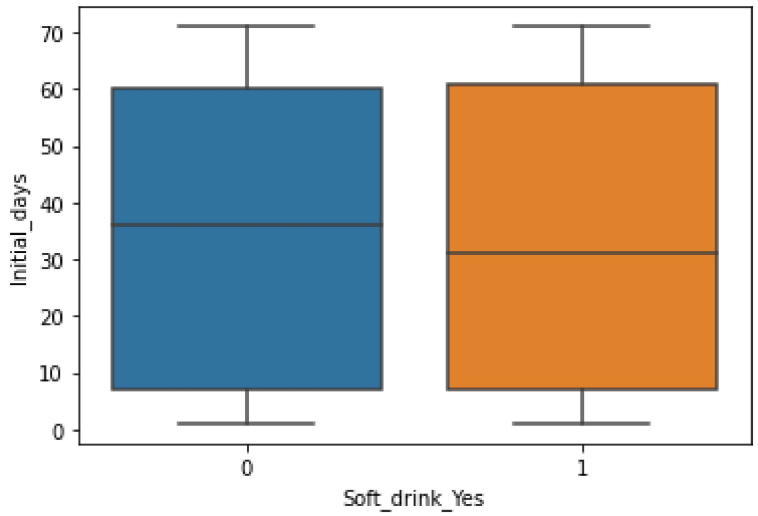
In [47]:
```
sns.boxplot(data=dfupdated, x="Gender_Female", y="Initial_days")
```

Out[47]:
```
<AxesSubplot:xlabel='Gender_Female', ylabel='Initial_days'>
```



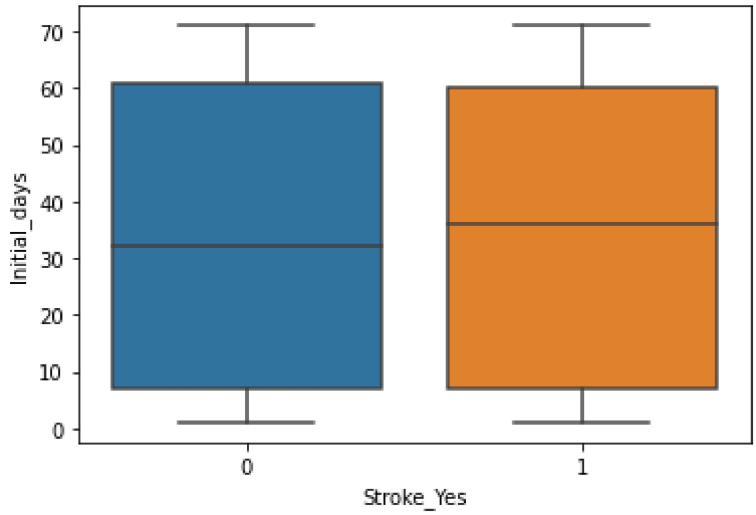
In [48]:
```
sns.boxplot(data=dfupdated, x="Soft_drink_Yes", y="Initial_days")
```

Out[48]:
```
<AxesSubplot:xlabel='Soft_drink_Yes', ylabel='Initial_days'>
```



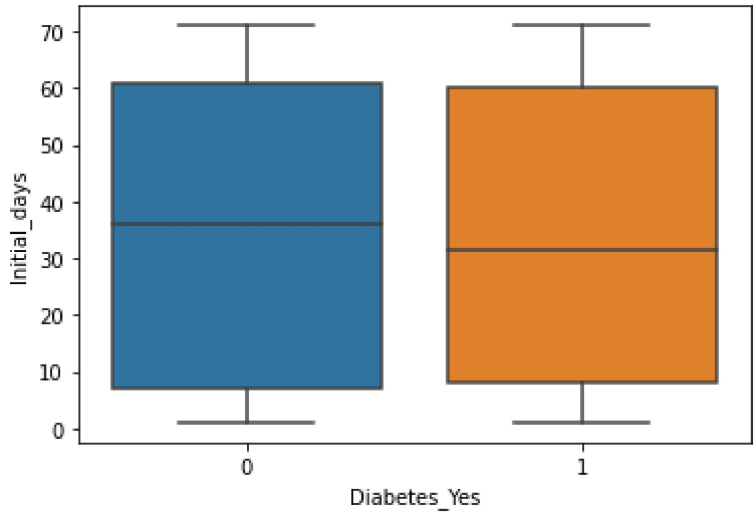
In [49]:
```
sns.boxplot(data=dfupdated, x="Stroke_Yes", y="Initial_days")
```

Out[49]:
```
<AxesSubplot:xlabel='Stroke_Yes', ylabel='Initial_days'>
```



In [50]:
```
sns.boxplot(data=dfupdated, x="Diabetes_Yes", y="Initial_days")
```

Out[50]:
```
<AxesSubplot:xlabel='Diabetes_Yes', ylabel='Initial_days'>
```



In [51]:
```
#Pearson’s correlation coefficient test on doc visits
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['Doc_visits']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: -0.007
```

In [52]:
```
#Pearson’s correlation coefficient test on soft drink
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['Soft_drink_Yes']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: 0.002
```

In [54]:
```
#Pearson’s correlation coefficient test on age
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['Age']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: 0.016
```

In [55]:
```
#Pearson’s correlation coefficient test on ReAdmis_Yes
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['ReAdmis_Yes']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: 0.851
```

In [58]:
```
#Pearson's correlation coefficient test on Anxiety_Yes
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['Anxiety_Yes']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: 0.012
```

In [59]:
```
#Pearson's correlation coefficient test on Services_MRI
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['Services_MRI']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: 0.008
```

In [60]:
```
#Pearson's correlation coefficient test on Overweight_Yes
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['Overweight_Yes']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: -0.011
```

In [61]:
```
#Pearson's correlation coefficient test on Arthritis_Yes
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['Arthritis_Yes']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: 0.019
```

In [62]:
```
#Pearson's correlation coefficient test on HighBlood_Yes
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['HighBlood_Yes']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: -0.006
```

In [63]:
```
dfupdated.rename(columns = {'Initial_admin_Emergency Admission':'EmergencyAdmissi
```

In [64]:
```
#Pearson's correlation coefficient test on Initial_admin_Emergency Admission
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['EmergencyAdmission']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: -0.011
```

In [65]:
```
#Pearson's correlation coefficient test on Complication_risk_High
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['Complication_risk_High']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: -0.006
```

In [66]:
```
#Pearson's correlation coefficient test on Gender_Female
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['Gender_Female']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: -0.008
```

In [67]:
```
#Pearson's correlation coefficient test on Stroke Yes
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['Stroke_Yes']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: -0.002
```

In [68]:
```
#Pearson's correlation coefficient test on Diabetes Yes
from scipy.stats import pearsonr
# Convert dataframe into series
list1 = dfupdated['Initial_days']
list2 = dfupdated['Diabetes_Yes']
 # Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: -0.002
```

In [69]:
```
#Multiple Regression Model
# (First regression model. Split the data into two different data sets with a 7:3
from sklearn.model_selection import train_test_split
```

In [70]:
```
# We specify random seed so that the train and test data set always have the same
np.random.seed(0)
dfupdated_train, dfupdated_test = train_test_split(dfupdated, train_size = 0.7, t
```

In [71]:
```
#Next - rescale the features. This does not include the dummy variables.  Rescali
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [72]:
```
# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Age', 'Doc_visits', 'Initial_days']
dfupdated_train[num_vars] = scaler.fit_transform(dfupdated_train[num_vars])
dfupdated_train
```

Out[72]:

	Age	Doc_visits	Initial_days	ReAdmis_Yes	Anxiety_Yes	Services_MRI	Overweight_Yes
1191	0.633803	1.000000	0.085714	0	0	0	1
4458	0.450704	0.285714	0.100000	0	1	0	0
1131	0.690141	0.428571	0.071429	0	0	0	0
9562	0.380282	0.571429	0.885714	1	0	0	1
6579	0.380282	0.714286	0.985714	1	1	0	1
...	...	...	...	...	...	...	...
350	0.915493	0.571429	0.085714	0	0	0	0
79	0.225352	0.714286	0.128571	0	1	0	0
8039	0.830986	0.571429	0.814286	1	1	0	0
6936	0.816901	0.428571	0.900000	1	1	0	0
5640	0.239437	0.428571	0.814286	1	0	0	1

7000 rows × 15 columns

In [73]:
```
#Divide the data into X and Y
y_train = dfupdated_train.pop('Initial_days')
X_train = dfupdated_train
```

In [74]:
```
#Build a linear model and add all variables
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[74]:

OLS Regression Results

Dep. Variable:	Initial_days	**R-squared:**	0.727
Model:	OLS	**Adj. R-squared:**	0.727
Method:	Least Squares	**F-statistic:**	1332.
Date:	Wed, 04 Jan 2023	**Prob (F-statistic):**	0.00
Time:	16:48:36	**Log-Likelihood:**	1482.0
No. Observations:	7000	**AIC:**	-2934.
Df Residuals:	6985	**BIC:**	-2831.
Df Model:	14		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2424	0.012	20.196	0.000	0.219	0.266
Age	0.0015	0.008	0.182	0.856	-0.014	0.017
Doc_visits	-0.0067	0.016	-0.425	0.671	-0.038	0.024
ReAdmis_Yes	0.6628	0.005	136.359	0.000	0.653	0.672
Anxiety_Yes	0.0131	0.005	2.603	0.009	0.003	0.023
Services_MRI	0.0043	0.012	0.348	0.728	-0.020	0.028
Overweight_Yes	-0.0053	0.005	-1.036	0.300	-0.015	0.005
Arthritis_Yes	0.0055	0.005	1.134	0.257	-0.004	0.015
HighBlood_Yes	-0.0102	0.005	-2.141	0.032	-0.020	-0.001
EmergencyAdmission	-0.0245	0.005	-5.215	0.000	-0.034	-0.015
Complication_risk_High	0.0002	0.005	0.046	0.963	-0.009	0.010
Gender_Female	0.0057	0.005	1.221	0.222	-0.003	0.015
Soft_drink_Yes	-0.0044	0.005	-0.817	0.414	-0.015	0.006
Stroke_Yes	0.0012	0.006	0.208	0.835	-0.010	0.013
Diabetes_Yes	-0.0009	0.005	-0.164	0.870	-0.011	0.009

Omnibus:	1404.389	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2413.115
Skew:	1.338	**Prob(JB):**	0.00
Kurtosis:	4.056	**Cond. No.**	15.0

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [75]:
```
#create a new dataframe with only the variables kept
dfupdated2 = dfupdated[['Initial_days', 'Age', 'Doc_visits', 'ReAdmis_Yes', 'Anxi
```

In [76]:
```
# (Second regression model. Split the data into two different data sets with a 7:
from sklearn.model_selection import train_test_split
```

In [77]:
```
# We specify random seed so that the train and test data set always have the same
np.random.seed(0)
dfupdated2_train, df_test = train_test_split(dfupdated2, train_size = 0.7, test_s
```

In [78]:
```
#Next - rescale the features. This does not include the dummy variables.  Rescali
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [79]:
```
# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Age', 'Doc_visits', 'Initial_days']
dfupdated2_train[num_vars] = scaler.fit_transform(dfupdated2_train[num_vars])
dfupdated2_train
```

Out[79]:

	Initial_days	Age	Doc_visits	ReAdmis_Yes	Anxiety_Yes	Services_MRI	Overweight_Yes
1191	0.085714	0.633803	1.000000	0	0	0	1
4458	0.100000	0.450704	0.285714	0	1	0	0
1131	0.071429	0.690141	0.428571	0	0	0	0
9562	0.885714	0.380282	0.571429	1	0	0	1
6579	0.985714	0.380282	0.714286	1	1	0	1
...	...	...	...	...	...	...	...
350	0.085714	0.915493	0.571429	0	0	0	0
79	0.128571	0.225352	0.714286	0	1	0	0
8039	0.814286	0.830986	0.571429	1	1	0	0
6936	0.900000	0.816901	0.428571	1	1	0	0
5640	0.814286	0.239437	0.428571	1	0	0	1

7000 rows × 14 columns

In [80]:
```
#Divide the data into X and Y
y_train = dfupdated2_train.pop('Initial_days')
X_train = dfupdated2_train
```

In [81]:
```
#Build a linear model and add all variables
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[81]:

OLS Regression Results

Dep. Variable:	Initial_days	**R-squared:**	0.727
Model:	OLS	**Adj. R-squared:**	0.727
Method:	Least Squares	**F-statistic:**	1434.
Date:	Wed, 04 Jan 2023	**Prob (F-statistic):**	0.00
Time:	16:50:38	**Log-Likelihood:**	1482.0
No. Observations:	7000	**AIC:**	-2936.
Df Residuals:	6986	**BIC:**	-2840.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2425	0.012	20.362	0.000	0.219	0.266
Age	0.0015	0.008	0.182	0.855	-0.014	0.017
Doc_visits	-0.0067	0.016	-0.424	0.671	-0.038	0.024
ReAdmis_Yes	0.6628	0.005	136.375	0.000	0.653	0.672
Anxiety_Yes	0.0131	0.005	2.602	0.009	0.003	0.023
Services_MRI	0.0043	0.012	0.347	0.729	-0.020	0.028
Overweight_Yes	-0.0053	0.005	-1.036	0.300	-0.015	0.005
Arthritis_Yes	0.0055	0.005	1.134	0.257	-0.004	0.015
HighBlood_Yes	-0.0102	0.005	-2.141	0.032	-0.020	-0.001
EmergencyAdmission	-0.0245	0.005	-5.216	0.000	-0.034	-0.015
Gender_Female	0.0057	0.005	1.221	0.222	-0.003	0.015
Soft_drink_Yes	-0.0044	0.005	-0.817	0.414	-0.015	0.006
Diabetes_Yes	-0.0009	0.005	-0.164	0.869	-0.011	0.009
Stroke_Yes	0.0012	0.006	0.209	0.835	-0.010	0.013

Omnibus:	1404.366	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2413.051
Skew:	1.338	**Prob(JB):**	0.00
Kurtosis:	4.056	**Cond. No.**	14.7

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [82]:
```
#create a new dataframe with only the variables kept
Dfupdated3 = dfupdated2[['Initial_days', 'Age', 'Doc_visits', 'ReAdmis_Yes', 'An
```

In [83]:
```
# Third regression model. Split the data into two different data sets with a 7:3
from sklearn.model_selection import train_test_split
```

In [84]:
```
# We specify random seed so that the train and test data set always have the same
np.random.seed(0)
Dfupdated3_train, df_test = train_test_split(Dfupdated3, train_size = 0.7, test_s
```

In [85]:
```
#Next - rescale the features. This does not include the dummy variables.  Rescali
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [86]:
```
# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Age','Doc_visits', 'Initial_days']
Dfupdated3_train[num_vars] = scaler.fit_transform(Dfupdated3_train[num_vars])
Dfupdated3_train
```

Out[86]:

	Initial_days	Age	Doc_visits	ReAdmis_Yes	Anxiety_Yes	Services_MRI	Overweight_Yes
1191	0.085714	0.633803	1.000000	0	0	0	1
4458	0.100000	0.450704	0.285714	0	1	0	0
1131	0.071429	0.690141	0.428571	0	0	0	0
9562	0.885714	0.380282	0.571429	1	0	0	1
6579	0.985714	0.380282	0.714286	1	1	0	1
...	...	...	...	...	...	...	...
350	0.085714	0.915493	0.571429	0	0	0	0
79	0.128571	0.225352	0.714286	0	1	0	0
8039	0.814286	0.830986	0.571429	1	1	0	0
6936	0.900000	0.816901	0.428571	1	1	0	0
5640	0.814286	0.239437	0.428571	1	0	0	1

7000 rows × 13 columns

In [87]:
```
#Divide the data into X and Y
y_train = Dfupdated3_train.pop('Initial_days')
X_train = Dfupdated3_train
```

In [88]:
```
#Build a linear model and add all variables
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[88]:

OLS Regression Results

Dep. Variable:	Initial_days	**R-squared:**	0.727
Model:	OLS	**Adj. R-squared:**	0.727
Method:	Least Squares	**F-statistic:**	1554.
Date:	Wed, 04 Jan 2023	**Prob (F-statistic):**	0.00
Time:	16:52:14	**Log-Likelihood:**	1482.0
No. Observations:	7000	**AIC:**	-2938.
Df Residuals:	6987	**BIC:**	-2849.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2422	0.012	20.478	0.000	0.219	0.265
Age	0.0015	0.008	0.182	0.855	-0.014	0.017
Doc_visits	-0.0067	0.016	-0.426	0.670	-0.038	0.024
ReAdmis_Yes	0.6628	0.005	136.385	0.000	0.653	0.672
Anxiety_Yes	0.0131	0.005	2.604	0.009	0.003	0.023
Services_MRI	0.0042	0.012	0.344	0.731	-0.020	0.028
Overweight_Yes	-0.0053	0.005	-1.035	0.301	-0.015	0.005
Arthritis_Yes	0.0055	0.005	1.132	0.258	-0.004	0.015
HighBlood_Yes	-0.0102	0.005	-2.140	0.032	-0.020	-0.001
EmergencyAdmission	-0.0245	0.005	-5.216	0.000	-0.034	-0.015
Gender_Female	0.0057	0.005	1.220	0.222	-0.003	0.015
Soft_drink_Yes	-0.0044	0.005	-0.821	0.412	-0.015	0.006
Stroke_Yes	0.0012	0.006	0.207	0.836	-0.010	0.013

Omnibus:	1404.342	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2412.981
Skew:	1.338	**Prob(JB):**	0.00
Kurtosis:	4.056	**Cond. No.**	14.5

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [89]:
```
#create a new dataframe with only the variables kept
Dfupdated4 = Dfupdated3[['Initial_days', 'Doc_visits', 'ReAdmis_Yes', 'Anxiety_Ye
```

In [90]:
```
# 4th regression model. Split the data into two different data sets with a 7:3 ra
from sklearn.model_selection import train_test_split
```

In [91]:
```
# We specify random seed so that the train and test data set always have the same
np.random.seed(0)
Dfupdated4_train, df_test = train_test_split(Dfupdated4, train_size = 0.7, test_s
```

In [92]:
```
#Next - rescale the features. This does not include the dummy variables.  Rescali
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [93]:
```
# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Doc_visits', 'Initial_days']
Dfupdated4_train[num_vars] = scaler.fit_transform(Dfupdated4_train[num_vars])
Dfupdated4_train
```

Out[93]:

	Initial_days	Doc_visits	ReAdmis_Yes	Anxiety_Yes	Services_MRI	Overweight_Yes	Arthritis_
1191	0.085714	1.000000	0	0	0	1	
4458	0.100000	0.285714	0	1	0	0	
1131	0.071429	0.428571	0	0	0	0	
9562	0.885714	0.571429	1	0	0	1	
6579	0.985714	0.714286	1	1	0	1	
...	...	...	...	...	...	...	
350	0.085714	0.571429	0	0	0	0	
79	0.128571	0.714286	0	1	0	0	
8039	0.814286	0.571429	1	1	0	0	
6936	0.900000	0.428571	1	1	0	0	
5640	0.814286	0.428571	1	0	0	1	

7000 rows × 12 columns

In [94]:
```
#Divide the data into X and Y
y_train = Dfupdated4_train.pop('Initial_days')
X_train = Dfupdated4_train
```

In [95]:
```
#Build a linear model and add all variables
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[95]:

OLS Regression Results

Dep. Variable:	Initial_days	**R-squared:**	0.727
Model:	OLS	**Adj. R-squared:**	0.727
Method:	Least Squares	**F-statistic:**	1696.
Date:	Wed, 04 Jan 2023	**Prob (F-statistic):**	0.00
Time:	16:53:54	**Log-Likelihood:**	1482.0
No. Observations:	7000	**AIC:**	-2940.
Df Residuals:	6988	**BIC:**	-2858.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2430	0.011	21.768	0.000	0.221	0.265
Doc_visits	-0.0067	0.016	-0.426	0.670	-0.038	0.024
ReAdmis_Yes	0.6628	0.005	136.412	0.000	0.653	0.672
Anxiety_Yes	0.0131	0.005	2.607	0.009	0.003	0.023
Services_MRI	0.0042	0.012	0.346	0.729	-0.020	0.028
Overweight_Yes	-0.0053	0.005	-1.036	0.300	-0.015	0.005
Arthritis_Yes	0.0055	0.005	1.134	0.257	-0.004	0.015
HighBlood_Yes	-0.0102	0.005	-2.139	0.032	-0.020	-0.001
EmergencyAdmission	-0.0245	0.005	-5.218	0.000	-0.034	-0.015
Gender_Female	0.0057	0.005	1.223	0.221	-0.003	0.015
Soft_drink_Yes	-0.0044	0.005	-0.821	0.412	-0.015	0.006
Stroke_Yes	0.0012	0.006	0.211	0.833	-0.010	0.013

Omnibus:	1404.341	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2412.985
Skew:	1.338	**Prob(JB):**	0.00
Kurtosis:	4.056	**Cond. No.**	13.9

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [96]:
```
#create a new dataframe with only the variables kept
Dfupdated5 = Dfupdated4[['Initial_days', 'Doc_visits', 'ReAdmis_Yes', 'Anxiety_Ye
```

In [97]:
```
# Fifth regression model. Split the data into two different data sets with a 7:3
from sklearn.model_selection import train_test_split
```

In [98]:
```
# We specify random seed so that the train and test data set always have the same
np.random.seed(0)
Dfupdated5_train, Dfupdated5_test = train_test_split(Dfupdated5, train_size = 0.7
```

In [99]:
```
#Next - rescale the features. This does not include the dummy variables.  Rescali
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [100]:
```
# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Doc_visits', 'Initial_days']
Dfupdated5_train[num_vars] = scaler.fit_transform(Dfupdated5_train[num_vars])
Dfupdated5_train
```

Out[100]:

	Initial_days	Doc_visits	ReAdmis_Yes	Anxiety_Yes	Services_MRI	Overweight_Yes	Arthritis_
1191	0.085714	1.000000	0	0	0	1	
4458	0.100000	0.285714	0	1	0	0	
1131	0.071429	0.428571	0	0	0	0	
9562	0.885714	0.571429	1	0	0	1	
6579	0.985714	0.714286	1	1	0	1	
...	...	...	...	...	...	...	
350	0.085714	0.571429	0	0	0	0	
79	0.128571	0.714286	0	1	0	0	
8039	0.814286	0.571429	1	1	0	0	
6936	0.900000	0.428571	1	1	0	0	
5640	0.814286	0.428571	1	0	0	1	

7000 rows × 11 columns

In [101]:
```
#Divide the data into X and Y
y_train = Dfupdated5_train.pop('Initial_days')
X_train = Dfupdated5_train
```

In [102]:
```
#Build a linear model and add all variables
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[102]:

OLS Regression Results

Dep. Variable:	Initial_days	**R-squared:**	0.727
Model:	OLS	**Adj. R-squared:**	0.727
Method:	Least Squares	**F-statistic:**	1866.
Date:	Wed, 04 Jan 2023	**Prob (F-statistic):**	0.00
Time:	16:56:32	**Log-Likelihood:**	1481.9
No. Observations:	7000	**AIC:**	-2942.
Df Residuals:	6989	**BIC:**	-2866.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2432	0.011	21.921	0.000	0.221	0.265

In [103]:
```
iety_Yes', 'Overweight_Yes', 'Arthritis_Yes', 'HighBlood_Yes', 'EmergencyAdmissior
```

In [104]:
```
# 6th regression model. Split the data into two different data sets with a 7:3 ra
from sklearn.model_selection import train_test_split
```

In [105]:
```
# We specify random seed so that the train and test data set always have the same
np.random.seed(0)
```

In [106]:
```
Dfupdated6_train, Dfupdated6_test = train_test_split(Dfupdated6, train_size = 0.7
```

In [107]:
```
#Next - rescale the features. This does not include the dummy variables.  Rescali
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [108]:
```
# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Doc_visits', 'Initial_days']
Dfupdated6_train[num_vars] = scaler.fit_transform(Dfupdated6_train[num_vars])
Dfupdated6_train
```

Out[108]:

	Initial_days	Doc_visits	ReAdmis_Yes	Anxiety_Yes	Overweight_Yes	Arthritis_Yes	HighBlooc
1191	0.085714	1.000000	0	0	1	0	
4458	0.100000	0.285714	0	1	0	1	
1131	0.071429	0.428571	0	0	0	0	
9562	0.885714	0.571429	1	0	1	1	
6579	0.985714	0.714286	1	1	1	1	
...	...	...	...	...	...	...	
350	0.085714	0.571429	0	0	0	1	
79	0.128571	0.714286	0	1	0	0	
8039	0.814286	0.571429	1	1	0	0	
6936	0.900000	0.428571	1	1	0	0	
5640	0.814286	0.428571	1	0	1	1	

7000 rows × 10 columns

In [109]:
```
#Divide the data into X and Y
y_train = Dfupdated6_train.pop('Initial_days')
X_train = Dfupdated6_train
```

In [110]:
```
#Build a linear model and add all variables
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[110]:

OLS Regression Results

Dep. Variable:	Initial_days	**R-squared:**	0.727
Model:	OLS	**Adj. R-squared:**	0.727
Method:	Least Squares	**F-statistic:**	2073.
Date:	Wed, 04 Jan 2023	**Prob (F-statistic):**	0.00
Time:	16:58:02	**Log-Likelihood:**	1481.9
No. Observations:	7000	**AIC:**	-2944.
Df Residuals:	6990	**BIC:**	-2875.
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2434	0.011	21.963	0.000	0.222	0.265
Doc_visits	-0.0068	0.016	-0.431	0.667	-0.038	0.024
ReAdmis_Yes	0.6628	0.005	136.441	0.000	0.653	0.672
Anxiety_Yes	0.0130	0.005	2.598	0.009	0.003	0.023
Overweight_Yes	-0.0053	0.005	-1.037	0.300	-0.015	0.005
Arthritis_Yes	0.0055	0.005	1.126	0.260	-0.004	0.015
HighBlood_Yes	-0.0102	0.005	-2.135	0.033	-0.020	-0.001
EmergencyAdmission	-0.0245	0.005	-5.218	0.000	-0.034	-0.015
Gender_Female	0.0058	0.005	1.232	0.218	-0.003	0.015
Soft_drink_Yes	-0.0044	0.005	-0.814	0.416	-0.015	0.006

Omnibus:	1403.932	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2411.855
Skew:	1.338	**Prob(JB):**	0.00
Kurtosis:	4.055	**Cond. No.**	13.8

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [111]:
```
#create a new dataframe with only the variables kept
Dfupdated7 = Dfupdated6[['Initial_days', 'ReAdmis_Yes', 'Anxiety_Yes', 'Overweigh
```

In [112]:
```
# 7th regression model. Split the data into two different data sets with a 7:3 ra
from sklearn.model_selection import train_test_split
```

In [113]:
```
# We specify random seed so that the train and test data set always have the same
np.random.seed(0)
Dfupdated7_train, Dfupdated7_test = train_test_split(Dfupdated7, train_size = 0.7
```

In [114]:
```
#Next - rescale the features. This does not include the dummy variables.  Rescali
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [116]:
```
# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Initial_days']
Dfupdated7_train[num_vars] = scaler.fit_transform(Dfupdated7_train[num_vars])
Dfupdated7_train
```

Out[116]:

	Initial_days	ReAdmis_Yes	Anxiety_Yes	Overweight_Yes	Arthritis_Yes	HighBlood_Yes	Emer
1191	0.085714	0	0	1	0	1	
4458	0.100000	0	1	0	1	0	
1131	0.071429	0	0	0	0	1	
9562	0.885714	1	0	1	1	0	
6579	0.985714	1	1	1	1	1	
...	...	...	...	...	...	...	
350	0.085714	0	0	0	1	0	
79	0.128571	0	1	0	0	0	
8039	0.814286	1	1	0	0	0	
6936	0.900000	1	1	0	0	1	
5640	0.814286	1	0	1	1	1	

7000 rows × 9 columns

In [117]:
```
#Divide the data into X and Y
y_train = Dfupdated7_train.pop('Initial_days')
X_train = Dfupdated7_train
```

In [118]:
```
#Build a linear model and add all variables
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[118]:

OLS Regression Results

Dep. Variable:	Initial_days	**R-squared:**	0.727
Model:	OLS	**Adj. R-squared:**	0.727
Method:	Least Squares	**F-statistic:**	2333.
Date:	Wed, 04 Jan 2023	**Prob (F-statistic):**	0.00
Time:	17:05:02	**Log-Likelihood:**	1481.8
No. Observations:	7000	**AIC:**	-2946.
Df Residuals:	6991	**BIC:**	-2884.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2396	0.007	36.118	0.000	0.227	0.253
ReAdmis_Yes	0.6628	0.005	136.472	0.000	0.653	0.672
Anxiety_Yes	0.0130	0.005	2.597	0.009	0.003	0.023
Overweight_Yes	-0.0054	0.005	-1.045	0.296	-0.015	0.005
Arthritis_Yes	0.0055	0.005	1.127	0.260	-0.004	0.015
HighBlood_Yes	-0.0102	0.005	-2.140	0.032	-0.020	-0.001
EmergencyAdmission	-0.0245	0.005	-5.225	0.000	-0.034	-0.015
Gender_Female	0.0057	0.005	1.225	0.221	-0.003	0.015
Soft_drink_Yes	-0.0044	0.005	-0.815	0.415	-0.015	0.006

Omnibus:	1404.452	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2413.223
Skew:	1.338	**Prob(JB):**	0.00
Kurtosis:	4.055	**Cond. No.**	5.77

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [119]:
```
variables kept
', 'ReAdmis_Yes', 'Anxiety_Yes', 'Overweight_Yes', 'Arthritis_Yes', 'HighBlood_Yes
```

In [120]:
```
# 8th regression model. Split the data into two different data sets with a 7:3 ra
from sklearn.model_selection import train_test_split
```

In [121]:
```
# We specify random seed so that the train and test data set always have the same
np.random.seed(0)
Dfupdated8_train, Dfupdated8_test = train_test_split(Dfupdated8, train_size = 0.7
```

In [122]:
```
#Next - rescale the features. This does not include the dummy variables.  Rescali
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [123]:
```
# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Initial_days']
Dfupdated8_train[num_vars] = scaler.fit_transform(Dfupdated8_train[num_vars])
Dfupdated8_train
```

Out[123]:

	Initial_days	ReAdmis_Yes	Anxiety_Yes	Overweight_Yes	Arthritis_Yes	HighBlood_Yes	Emer
1191	0.085714	0	0	1	0	1	
4458	0.100000	0	1	0	1	0	
1131	0.071429	0	0	0	0	1	
9562	0.885714	1	0	1	1	0	
6579	0.985714	1	1	1	1	1	
...	...	...	...	...	...	...	
350	0.085714	0	0	0	1	0	
79	0.128571	0	1	0	0	0	
8039	0.814286	1	1	0	0	0	
6936	0.900000	1	1	0	0	1	
5640	0.814286	1	0	1	1	1	

7000 rows × 8 columns

In [124]:
```
#Divide the data into X and Y
y_train = Dfupdated8_train.pop('Initial_days')
X_train = Dfupdated8_train
```

In [125]:
```
#Build a linear model and add all variables
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[125]:

OLS Regression Results

Dep. Variable:	Initial_days	**R-squared:**	0.727
Model:	OLS	**Adj. R-squared:**	0.727
Method:	Least Squares	**F-statistic:**	2666.
Date:	Wed, 04 Jan 2023	**Prob (F-statistic):**	0.00
Time:	17:09:04	**Log-Likelihood:**	1481.4
No. Observations:	7000	**AIC:**	-2947.
Df Residuals:	6992	**BIC:**	-2892.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2385	0.007	36.663	0.000	0.226	0.251
ReAdmis_Yes	0.6628	0.005	136.478	0.000	0.653	0.672
Anxiety_Yes	0.0130	0.005	2.585	0.010	0.003	0.023
Overweight_Yes	-0.0053	0.005	-1.038	0.299	-0.015	0.005
Arthritis_Yes	0.0055	0.005	1.130	0.258	-0.004	0.015
HighBlood_Yes	-0.0102	0.005	-2.140	0.032	-0.020	-0.001
EmergencyAdmission	-0.0246	0.005	-5.244	0.000	-0.034	-0.015
Gender_Female	0.0057	0.005	1.218	0.223	-0.003	0.015

Omnibus:	1405.094	**Durbin-Watson:**	2.001
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2414.966
Skew:	1.338	**Prob(JB):**	0.00
Kurtosis:	4.057	**Cond. No.**	5.62

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [126]:
```
frame with only the variables kept
ated8[['Initial_days', 'ReAdmis_Yes', 'Anxiety_Yes', 'Arthritis_Yes', 'HighBlood_
```

In [127]:
```
# 9th regression model. Split the data into two different data sets with a 7:3 ra
from sklearn.model_selection import train_test_split
```

In [128]:
```
# We specify random seed so that the train and test data set always have the same
np.random.seed(0)
Dfupdated9_train, Dfupdated9_test = train_test_split(Dfupdated9, train_size = 0.7
```

In [129]:
```
#Next - rescale the features. This does not include the dummy variables.  Rescali
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [130]:
```
# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Initial_days']
Dfupdated9_train[num_vars] = scaler.fit_transform(Dfupdated9_train[num_vars])
Dfupdated9_train
```

Out[130]:

	Initial_days	ReAdmis_Yes	Anxiety_Yes	Arthritis_Yes	HighBlood_Yes	EmergencyAdmission
1191	0.085714	0	0	0	1	1
4458	0.100000	0	1	1	0	1
1131	0.071429	0	0	0	1	1
9562	0.885714	1	0	1	0	1
6579	0.985714	1	1	1	1	1
...	...	...	...	...	...	...
350	0.085714	0	0	1	0	0
79	0.128571	0	1	0	0	1
8039	0.814286	1	1	0	0	1
6936	0.900000	1	1	0	1	0
5640	0.814286	1	0	1	1	1

7000 rows × 7 columns

In [131]:
```
#Divide the data into X and Y
y_train = Dfupdated9_train.pop('Initial_days')
X_train = Dfupdated9_train
```

In [132]:
```
#Build a linear model and add all variables
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[132]:

OLS Regression Results

Dep. Variable:	Initial_days	**R-squared:**	0.727
Model:	OLS	**Adj. R-squared:**	0.727
Method:	Least Squares	**F-statistic:**	3110.
Date:	Wed, 04 Jan 2023	**Prob (F-statistic):**	0.00
Time:	17:12:35	**Log-Likelihood:**	1480.9
No. Observations:	7000	**AIC:**	-2948.
Df Residuals:	6993	**BIC:**	-2900.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2347	0.005	43.503	0.000	0.224	0.245
ReAdmis_Yes	0.6628	0.005	136.490	0.000	0.653	0.672
Anxiety_Yes	0.0130	0.005	2.588	0.010	0.003	0.023
Arthritis_Yes	0.0055	0.005	1.122	0.262	-0.004	0.015
HighBlood_Yes	-0.0103	0.005	-2.167	0.030	-0.020	-0.001
EmergencyAdmission	-0.0245	0.005	-5.230	0.000	-0.034	-0.015
Gender_Female	0.0057	0.005	1.220	0.223	-0.003	0.015

Omnibus:	1405.511	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2416.007
Skew:	1.339	**Prob(JB):**	0.00
Kurtosis:	4.057	**Cond. No.**	4.42

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [133]:
```
#create a new dataframe with only the variables kept
Dfupdated10 = Dfupdated9[['Initial_days', 'ReAdmis_Yes', 'Anxiety_Yes', 'HighBloo
```

In [134]:
```
# 10th regression model. Split the data into two different data sets with a 7:3 r
from sklearn.model_selection import train_test_split
```

In [135]:
```
# We specify random seed so that the train and test data set always have the same
np.random.seed(0)
Dfupdated10_train, Dfupdated10_test = train_test_split(Dfupdated10, train_size =
```

In [136]:
```
#Next – rescale the features. This does not include the dummy variables.  Rescali
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [137]:
```
# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Initial_days']
Dfupdated10_train[num_vars] = scaler.fit_transform(Dfupdated10_train[num_vars])
Dfupdated10_train
```

Out[137]:

	Initial_days	ReAdmis_Yes	Anxiety_Yes	HighBlood_Yes	EmergencyAdmission	Gender_Fema
1191	0.085714	0	0	1	1	
4458	0.100000	0	1	0	1	
1131	0.071429	0	0	1	1	
9562	0.885714	1	0	0	1	
6579	0.985714	1	1	1	1	
...	...	...	...	...	...	
350	0.085714	0	0	0	0	
79	0.128571	0	1	0	1	
8039	0.814286	1	1	0	1	
6936	0.900000	1	1	1	0	
5640	0.814286	1	0	1	1	

7000 rows × 6 columns

In [138]:
```
#Divide the data into X and Y
y_train = Dfupdated10_train.pop('Initial_days')
X_train = Dfupdated10_train
```

In [139]:
```
#Build a linear model and add all variables
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[139]:

OLS Regression Results

Dep. Variable:	Initial_days	**R-squared:**	0.727
Model:	OLS	**Adj. R-squared:**	0.727
Method:	Least Squares	**F-statistic:**	3731.
Date:	Wed, 04 Jan 2023	**Prob (F-statistic):**	0.00
Time:	17:15:35	**Log-Likelihood:**	1480.3
No. Observations:	7000	**AIC:**	-2949.
Df Residuals:	6994	**BIC:**	-2907.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2367	0.005	46.406	0.000	0.227	0.247
ReAdmis_Yes	0.6628	0.005	136.495	0.000	0.653	0.672
Anxiety_Yes	0.0131	0.005	2.599	0.009	0.003	0.023
HighBlood_Yes	-0.0103	0.005	-2.170	0.030	-0.020	-0.001
EmergencyAdmission	-0.0245	0.005	-5.234	0.000	-0.034	-0.015
Gender_Female	0.0057	0.005	1.212	0.226	-0.004	0.015

Omnibus:	1406.000	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2417.346
Skew:	1.339	**Prob(JB):**	0.00
Kurtosis:	4.058	**Cond. No.**	4.13

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [140]:
```
#Create new dataframe removing the highest p-value
Dfupdated11 = Dfupdated10[['Initial_days', 'ReAdmis_Yes', 'Anxiety_Yes', 'HighBlo
```

In [141]:
```
# 11th regression model. Split the data into two different data sets with a 7:3 r
from sklearn.model_selection import train_test_split
```

In [142]:
```
# We specify random seed so that the train and test data set always have the same
np.random.seed(0)
Dfupdated11_train, Dfupdated11_test = train_test_split(Dfupdated11, train_size =
```

In [143]:
```
#Next - rescale the features. This does not include the dummy variables.  Rescali
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [144]:
```
# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Initial_days']
Dfupdated11_train[num_vars] = scaler.fit_transform(Dfupdated11_train[num_vars])
Dfupdated11_train
```

Out[144]:

	Initial_days	ReAdmis_Yes	Anxiety_Yes	HighBlood_Yes	EmergencyAdmission
1191	0.085714	0	0	1	1
4458	0.100000	0	1	0	1
1131	0.071429	0	0	1	1
9562	0.885714	1	0	0	1
6579	0.985714	1	1	1	1
...	...	...	...	...	...
350	0.085714	0	0	0	0
79	0.128571	0	1	0	1
8039	0.814286	1	1	0	1
6936	0.900000	1	1	1	0
5640	0.814286	1	0	1	1

7000 rows × 5 columns

In [145]:
```
#Divide the data into X and Y
y_train = Dfupdated11_train.pop('Initial_days')
X_train = Dfupdated11_train
```

In [146]:
```
#Build a linear model and add all variables
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[146]:

OLS Regression Results

Dep. Variable:	Initial_days	**R-squared:**	0.727
Model:	OLS	**Adj. R-squared:**	0.727
Method:	Least Squares	**F-statistic:**	4664.
Date:	Wed, 04 Jan 2023	**Prob (F-statistic):**	0.00
Time:	17:19:43	**Log-Likelihood:**	1479.5
No. Observations:	7000	**AIC:**	-2949.
Df Residuals:	6995	**BIC:**	-2915.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2396	0.005	52.989	0.000	0.231	0.248
ReAdmis_Yes	0.6627	0.005	136.488	0.000	0.653	0.672
Anxiety_Yes	0.0131	0.005	2.600	0.009	0.003	0.023
HighBlood_Yes	-0.0104	0.005	-2.191	0.029	-0.020	-0.001
EmergencyAdmission	-0.0244	0.005	-5.212	0.000	-0.034	-0.015

Omnibus:	1405.284	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2415.341
Skew:	1.338	**Prob(JB):**	0.00
Kurtosis:	4.056	**Cond. No.**	3.59

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [155]:
```
#Review the means of the remaining variables
print(Dfupdated11.describe())
```

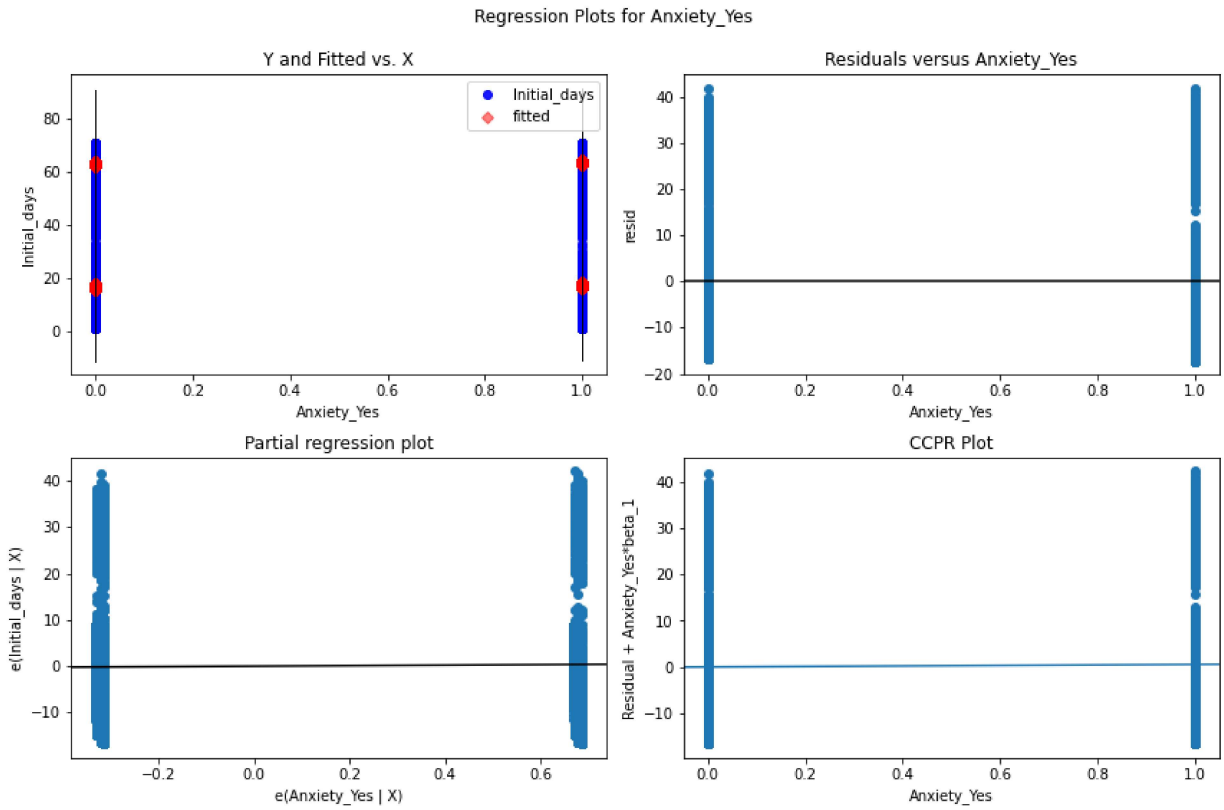
```
       Initial_days   ReAdmis_Yes   Anxiety_Yes  HighBlood_Yes  \
count  10000.000000  10000.000000  10000.000000   10000.000000
mean      33.956000      0.366900      0.321500       0.409000
std       26.301628      0.481983      0.467076       0.491674
min        1.000000      0.000000      0.000000       0.000000
25%        7.000000      0.000000      0.000000       0.000000
50%       35.500000      0.000000      0.000000       0.000000
75%       61.000000      1.000000      1.000000       1.000000
max       71.000000      1.000000      1.000000       1.000000

       EmergencyAdmission
count        10000.000000
mean             0.506000
std              0.499989
min              0.000000
25%              0.000000
50%              1.000000
75%              1.000000
max              1.000000
```

In [160]:
```
#Create residual plot
#fit multiple linear regression model
model = ols('Initial_days ~ Anxiety_Yes + HighBlood_Yes + ReAdmis_Yes + Emergency
#view model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Initial_days   R-squared:                       0.725
Model:                            OLS   Adj. R-squared:                  0.725
Method:                 Least Squares   F-statistic:                     6577.
Date:                Wed, 04 Jan 2023   Prob (F-statistic):               0.00
Time:                        17:40:57   Log-Likelihood:                -40436.
No. Observations:               10000   AIC:                          8.088e+04
Df Residuals:                    9995   BIC:                          8.092e+04
Df Model:                           4
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025
0.975]
---------------------------------------------------------------------------------------
Intercept              17.6652      0.266     66.389      0.000      17.144
18.187
Anxiety_Yes             0.5677      0.296      1.921      0.055      -0.012
1.147
HighBlood_Yes          -0.4501      0.281     -1.603      0.109      -1.000
0.100
ReAdmis_Yes            46.4535      0.286    162.166      0.000      45.892
47.015
EmergencyAdmission     -1.4851      0.276     -5.378      0.000      -2.026
-0.944
==============================================================================
Omnibus:                     1973.273   Durbin-Watson:                   1.269
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3361.943
Skew:                           1.329   Prob(JB):                         0.00
Kurtosis:                       4.001   Cond. No.                         3.59
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
```

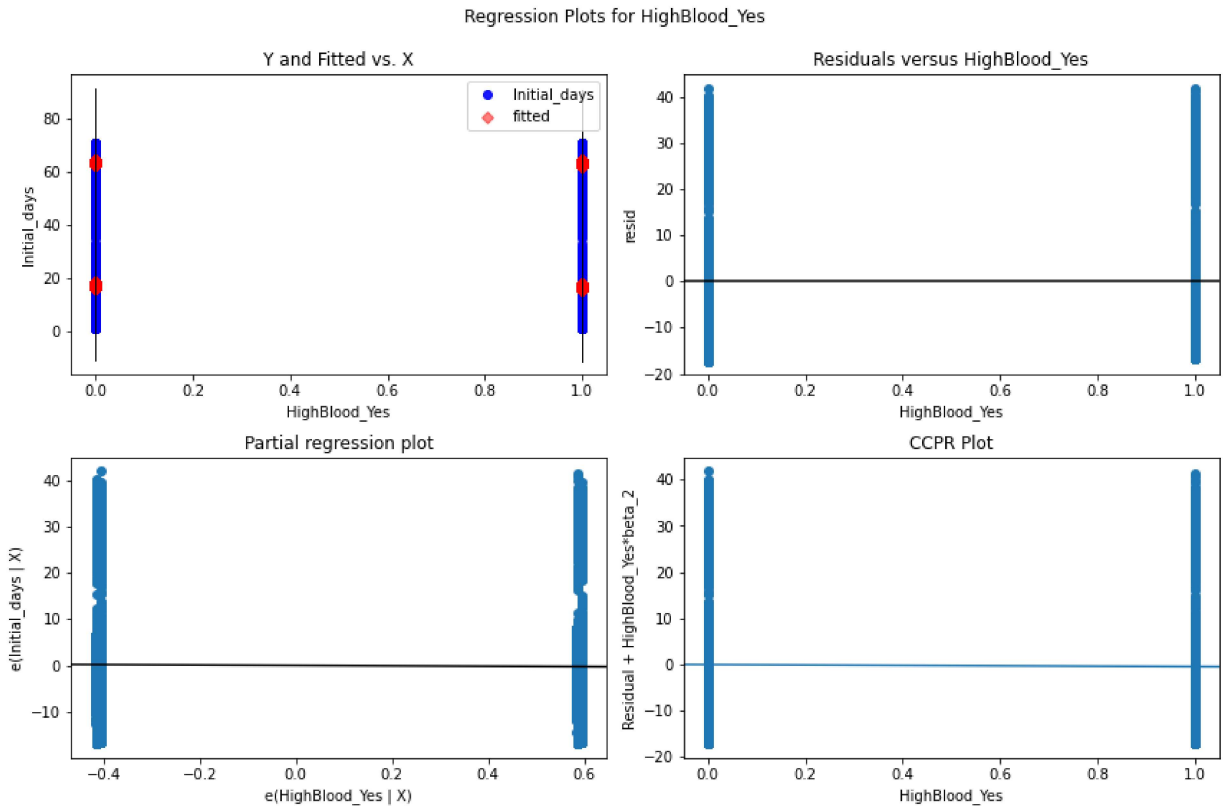
In [162]:
```
#create residual vs. predictor plot for 'Anxiety_Yes'
fig = plt.figure(figsize=(12,8))
fig = sm.graphics.plot_regress_exog(model, 'Anxiety_Yes', fig=fig)
```

```
eval_env: 1
```



In [163]:
```
#create residual vs. predictor plot for 'HighBlood Yes'
fig = plt.figure(figsize=(12,8))
fig = sm.graphics.plot_regress_exog(model, 'HighBlood_Yes', fig=fig)
```

```
eval_env: 1
```



In [165]:
```
import scipy.stats as st
# create 95% confidence interval
st.t.interval(alpha=0.95,
              df=len(Dfupdated11)-1,
              loc=np.mean(Dfupdated11),
              scale=st.sem(Dfupdated11))
```

```
C:\Users\Brittany\anaconda3\lib\site-packages\numpy\core\fromnumeric.py:3438: F
utureWarning: In a future version, DataFrame.mean(axis=None) will return a scal
ar mean over the entire DataFrame. To retain the old behavior, use 'frame.mean
(axis=0)' or just 'frame.mean()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

Out[165]:
```
(array([33.44043515,  0.35745216,  0.31234438,  0.3993622 ,  0.49619921]),
 array([34.47156485,  0.37634784,  0.33065562,  0.4186378 ,  0.51580079]))
```

In [ ]: