In [1]:
```
# Standard Data Science Imports
import numpy as np
import pandas as pd
from pandas import DataFrame
from sklearn.linear_model import LogisticRegression
import pandas as pd, numpy as np, matplotlib.pyplot as plt, os, sys, seaborn as s

# Visualization libraries
%matplotlib inline


# Scikit-learn
import sklearn
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV


from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

from numpy.linalg import norm
# Configure libraries
import warnings
warnings.filterwarnings('ignore')
# Configure libraries
warnings.filterwarnings('ignore')
plt.rcParams['figure.figsize'] = (10, 10)
plt.style.use('seaborn')
```

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

In [3]:
```
# Examining first five records of dataset
df.head()
```

Out[3]:

	CaseOrder	Customer_id	Interaction	UID	City	State
0	1	C412403	8cd49b13-f45a-4b47-a2bd-173ffa932c2f	3a83ddb66e2ae73798bdf1d705dc0932	Eva	AL
1	2	Z919181	d2450b70-0337-4406-bdbb-bc1037f1734c	176354c5eef714957d486009feabf195	Marianna	FL
2	3	F995323	a2057123-abf5-4a2c-abad-8ffe33512562	e19a0fa00aeda885b8a436757e889bc9	Sioux Falls	SD
3	4	A879973	1dec528d-eb34-4079-adce-0d7a40e82205	cd17d7b6d152cb6f23957346d11c3f07	New Richland	MN

In [4]:
```
# Viewing DataFrame descriptive information
df.info
```

Out[4]:
```
<bound method DataFrame.info of       CaseOrder Customer_id
Interaction  \
0             1     C412403  8cd49b13-f45a-4b47-a2bd-173ffa932c2f
1             2     Z919181  d2450b70-0337-4406-bdbb-bc1037f1734c
2             3     F995323  a2057123-abf5-4a2c-abad-8ffe33512562
3             4     A879973  1dec528d-eb34-4079-adce-0d7a40e82205
4             5     C544523  5885f56b-d6da-43a3-8760-83583af94266
...         ...         ...                                   ...
9995       9996     B863060  a25b594d-0328-486f-a9b9-0567eb0f9723
9996       9997     P712040  70711574-f7b1-4a17-b15f-48c54564b70f
9997       9998     R778890  1d79569d-8e0f-4180-a207-d67ee4527d26
9998       9999     E344109  f5a68e69-2a60-409b-a92f-ac0847b27db0
9999      10000     I569847  bc482c02-f8c9-4423-99de-3db5e62a18d5

                                UID          City State        County  \
0     3a83ddb66e2ae73798bdf1d705dc0932           Eva    AL        Morgan
1     176354c5eef714957d486009feabf195      Marianna    FL       Jackson
2     e19a0fa00aeda885b8a436757e889bc9   Sioux Falls    SD     Minnehaha
3     cd17d7b6d152cb6f23957346d11c3f07  New Richland    MN        Waseca
4     d2f0425877b10ed6bb381f3e2579424a    West Point    VA  King William
...                                ...           ...   ...           ...
9995  39184dc28cc038871912ccc4500049e5       Norlina    NC        Warren
9996  3cd124ccd43147404292e883bf9ec55c        Milmay    NJ      Atlantic
9997  41b770aeee97a5b9e7f69c906a8119d7     Southside    TN    Montgomery
9998  2bb491ef5b1beb1fed758cc6885c167a         Quinn    SD    Pennington
9999  95663a202338000abdf7e09311c2a8a1    Coraopolis    PA     Allegheny

        Zip       Lat        Lng  ...  TotalCharge Additional_charges Item1  \
0     35621  34.34960  -86.72508  ...  3726.702860       17939.403420     3
1     32446  30.84513  -85.22907  ...  4193.190458       17612.998120     3
2     57110  43.54321  -96.63772  ...  2434.234222       17505.192460     2
3     56072  43.89744  -93.51479  ...  2127.830423       12993.437350     3
4     23181  37.59894  -76.88958  ...  2113.073274        3716.525786     2
...     ...       ...        ...  ...          ...                ...   ...
9995  27563  36.42886  -78.23716  ...  6850.942000        8927.642000     3
9996   8340  39.43609  -74.87302  ...  7741.690000       28507.150000     3
9997  37171  36.36655  -87.29988  ...  8276.481000       15281.210000     3
9998  57775  44.10354 -102.01590  ...  7644.483000        7781.678000     5
9999  15108  40.49998  -80.19959  ...  7887.553000       11643.190000     4

     Item2  Item3  Item4  Item5 Item6 Item7 Item8
0        3      2      2      4     3     3     4
1        4      3      4      4     4     3     3
2        4      4      4      3     4     3     3
3        5      5      3      4     5     5     5
4        1      3      3      5     3     4     3
...    ...    ...    ...    ...   ...   ...   ...
9995     2      2      3      4     3     4     2
9996     3      4      2      5     3     4     4
9997     3      3      4      4     2     3     2
9998     5      3      4      4     3     4     3
9999     3      3      2      3     6     4     3

[10000 rows x 50 columns]>
```

In [5]:
```
# Get overview of descriptive stats
df.describe()
```

Out[5]:

	CaseOrder	Zip	Lat	Lng	Population	Children	
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000
mean	5000.50000	50159.323900	38.751099	-91.243080	9965.253800	2.097200	53
std	2886.89568	27469.588208	5.403085	15.205998	14824.758614	2.163659	20
min	1.00000	610.000000	17.967190	-174.209700	0.000000	0.000000	18
25%	2500.75000	27592.000000	35.255120	-97.352982	694.750000	0.000000	36
50%	5000.50000	50207.000000	39.419355	-88.397230	2769.000000	1.000000	53
75%	7500.25000	72411.750000	42.044175	-80.438050	13945.000000	3.000000	71
max	10000.00000	99929.000000	70.560990	-65.290170	122814.000000	10.000000	89

8 rows × 23 columns

In [6]:
```
# Getting data types of features
df.dtypes
```

Out[6]:
```
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 [7]:
```
# Checking for null values
df.isnull()
```

Out[7]:

	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 [8]:
```
#change to integers
df['TotalCharge'] = df['TotalCharge'].astype(int)
df['Initial_days'] = df['Initial_days'].astype(int)
#Change object to category
df["Gender"] = df["Gender"].astype('category')
df["ReAdmis"] = df["ReAdmis"].astype('category')
df["Soft_drink"] = df["Soft_drink"].astype('category')
df["Initial_admin"] = df["Initial_admin"].astype('category')
df["HighBlood"] = df["HighBlood"].astype('category')
df["Stroke"] = df["Stroke"].astype('category')
df["Overweight"] = df["Overweight"].astype('category')
df["Arthritis"] = df["Arthritis"].astype('category')
df["Diabetes"] = df["Diabetes"].astype('category')
df["Hyperlipidemia"] = df["Hyperlipidemia"].astype('category')
df["BackPain"] = df["BackPain"].astype('category')
df["Anxiety"] = df["Anxiety"].astype('category')
df["Allergic_rhinitis"] = df["Allergic_rhinitis"].astype('category')
df["Reflux_esophagitis"] = df["Reflux_esophagitis"].astype('category')
df["Services"] = df["Services"].astype('category')
df["Asthma"] = df["Asthma"].astype('category')
df["Marital"] = df["Marital"].astype('category')
df["Complication_risk"] = df["Complication_risk"].astype('category')
```

In [9]:
```
#drop columns not being used
to_drop = ['CaseOrder', 'Customer_id', 'Marital', 'Age', 'Hyperlipidemia', 'Asthm
df.drop(to_drop, inplace=True, axis=1)
```

In [10]:
```
#check data types
df.dtypes
```

Out[10]:
```
ReAdmis             category
Doc_visits             int64
Full_meals_eaten       int64
Soft_drink          category
HighBlood           category
Stroke              category
Overweight          category
Arthritis           category
Diabetes            category
BackPain            category
Anxiety             category
dtype: object
```

In [11]:
```
#check for missing values
df.isnull().sum()
```

Out[11]:
```
ReAdmis             0
Doc_visits          0
Full_meals_eaten    0
Soft_drink          0
HighBlood           0
Stroke              0
Overweight          0
Arthritis           0
Diabetes            0
BackPain            0
Anxiety             0
dtype: int64
```

In [12]:
```
#Run get dummies on categorical
pd.get_dummies(df, columns = ['ReAdmis', 'HighBlood', 'Overweight','Soft_drink',
```

Out[12]:

	Doc_visits	Full_meals_eaten	ReAdmis_No	ReAdmis_Yes	HighBlood_No	HighBlood_Yes
0	6	0	1	0	0	1
1	4	2	1	0	0	1
2	4	1	1	0	0	1
3	4	1	1	0	1	0
4	5	0	1	0	1	0
...	...	...	...	...	...	...
9995	4	2	1	0	0	1
9996	5	0	0	1	0	1
9997	4	2	0	1	0	1
9998	5	2	0	1	1	0
9999	5	0	0	1	1	0

10000 rows × 20 columns

In [13]:
```
#create new df with the get dummies responses
df_ready = pd.get_dummies(df, columns = ['ReAdmis', 'HighBlood', 'Overweight','So
```

In [14]:
```
#check data types
df_ready.dtypes
```

Out[14]:
```
Doc_visits          int64
Full_meals_eaten    int64
ReAdmis_No          uint8
ReAdmis_Yes         uint8
HighBlood_No        uint8
HighBlood_Yes       uint8
Overweight_No       uint8
Overweight_Yes      uint8
Soft_drink_No       uint8
Soft_drink_Yes      uint8
Stroke_No           uint8
Stroke_Yes          uint8
Arthritis_No        uint8
Arthritis_Yes       uint8
Diabetes_No         uint8
Diabetes_Yes        uint8
BackPain_No         uint8
BackPain_Yes        uint8
Anxiety_No          uint8
Anxiety_Yes         uint8
dtype: object
```

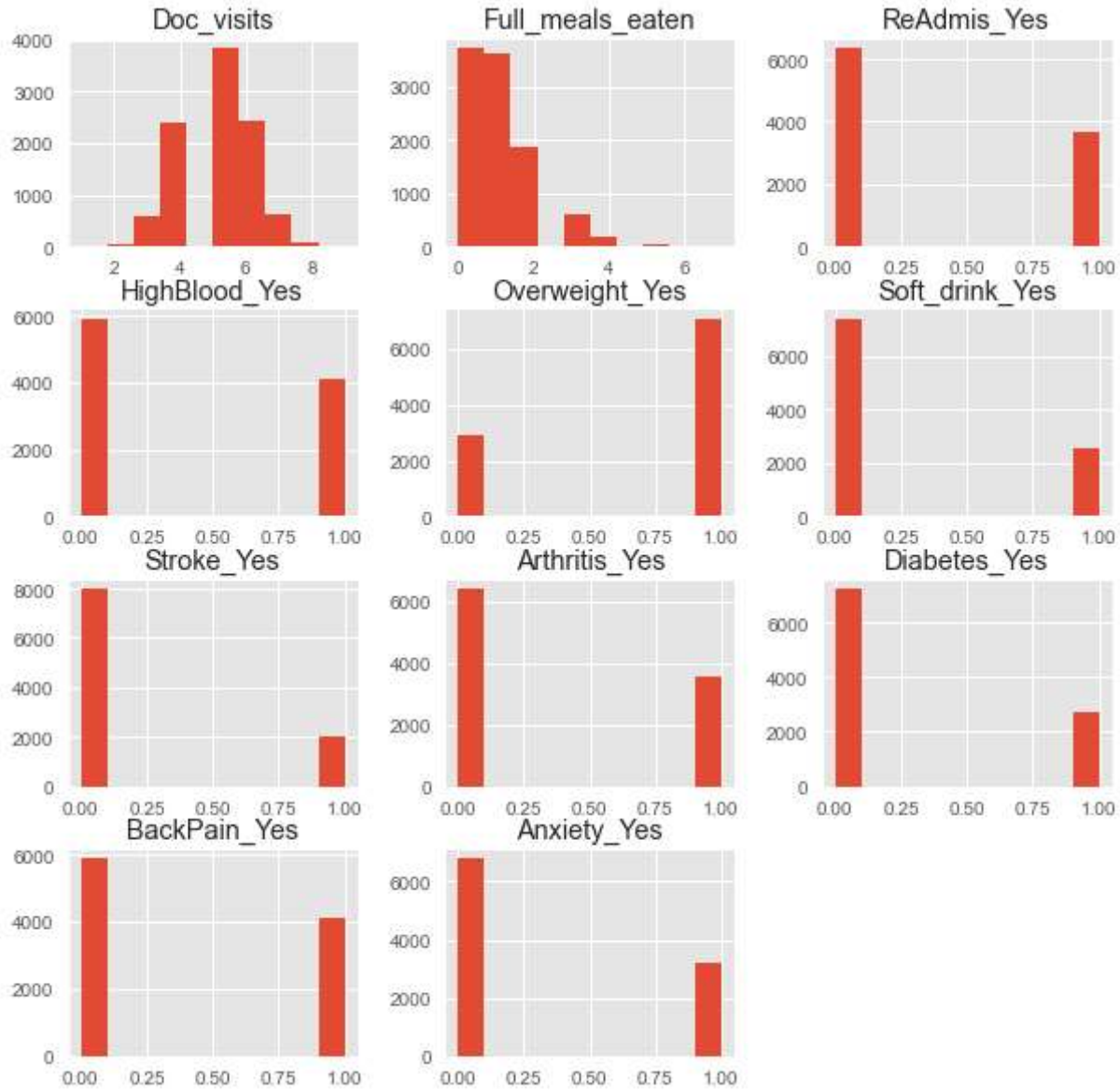
In [15]:
```
#drop multiple columns by name
df_ready.drop(['ReAdmis_No', 'HighBlood_No', 'Overweight_No', 'Soft_drink_No', 'A
```

In [16]:
```
#check data types
df_ready.dtypes
```

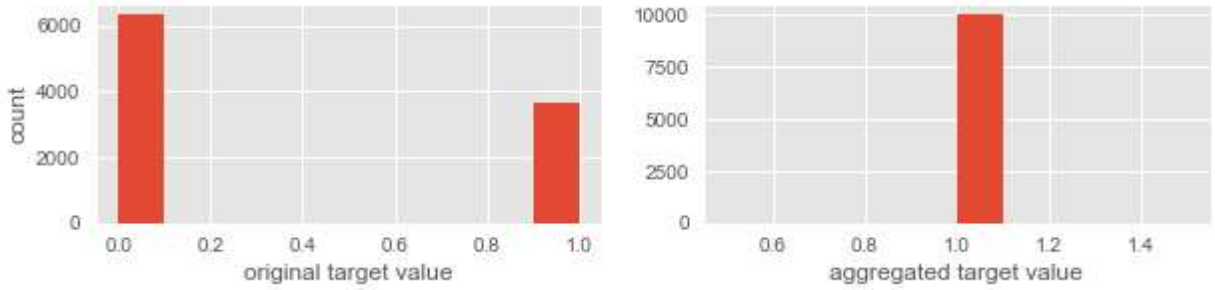
Out[16]:
```
Doc_visits          int64
Full_meals_eaten    int64
ReAdmis_Yes         uint8
HighBlood_Yes       uint8
Overweight_Yes      uint8
Soft_drink_Yes      uint8
Stroke_Yes          uint8
Arthritis_Yes       uint8
Diabetes_Yes        uint8
BackPain_Yes        uint8
Anxiety_Yes         uint8
dtype: object
```

In [17]:
```
#save prepared data
df_ready.to_csv('Documents/PreparedData D209 1.30.23.csv')
```

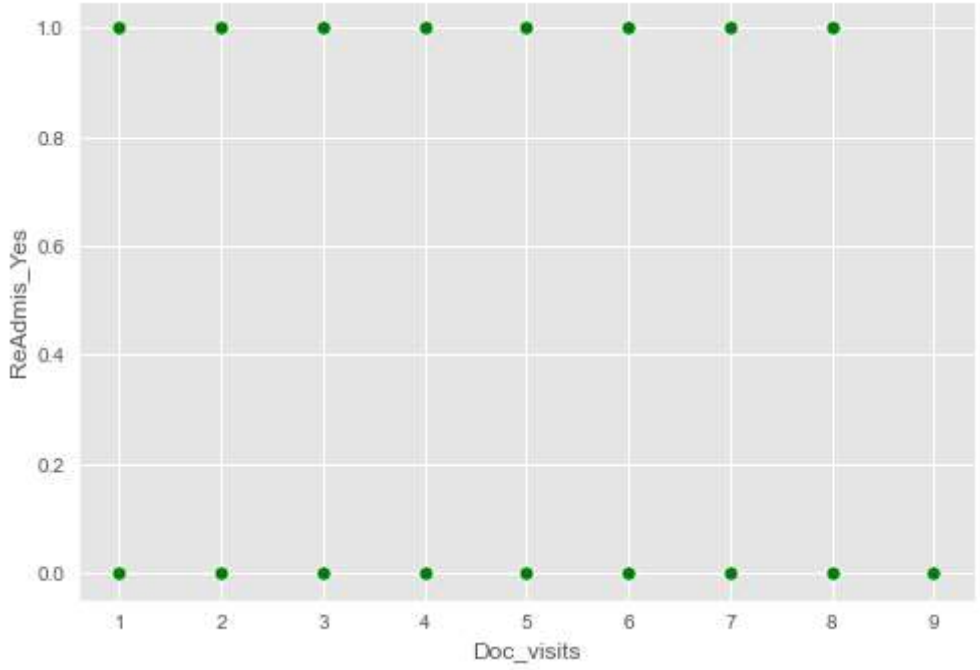
In [18]:
```
#view histograms to get a feel for the data
plt.style.use('ggplot')

X = df_ready.drop('ReAdmis_Yes', 1).values

# drop target variable
y1 = df_ready['ReAdmis_Yes'].values
pd.DataFrame.hist(df_ready, figsize = [10,10]);
plt.show()
```



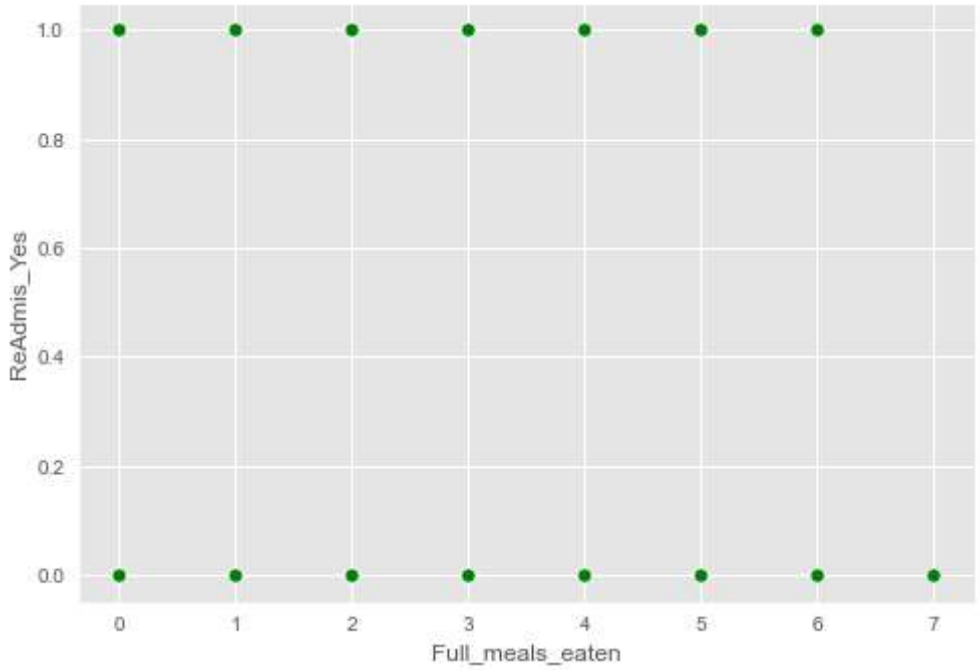
In [19]:
```
y = y1 <= 1 # is the rating <= 1?
# plot histograms of original target variable
# and aggregated target variable
plt.figure(figsize=(10,2));
plt.subplot(1, 2, 1 );
plt.hist(y1);
plt.xlabel('original target value')
plt.ylabel('count')
plt.subplot(1, 2, 2);
plt.hist(y)
plt.xlabel('aggregated target value')
plt.show()
```



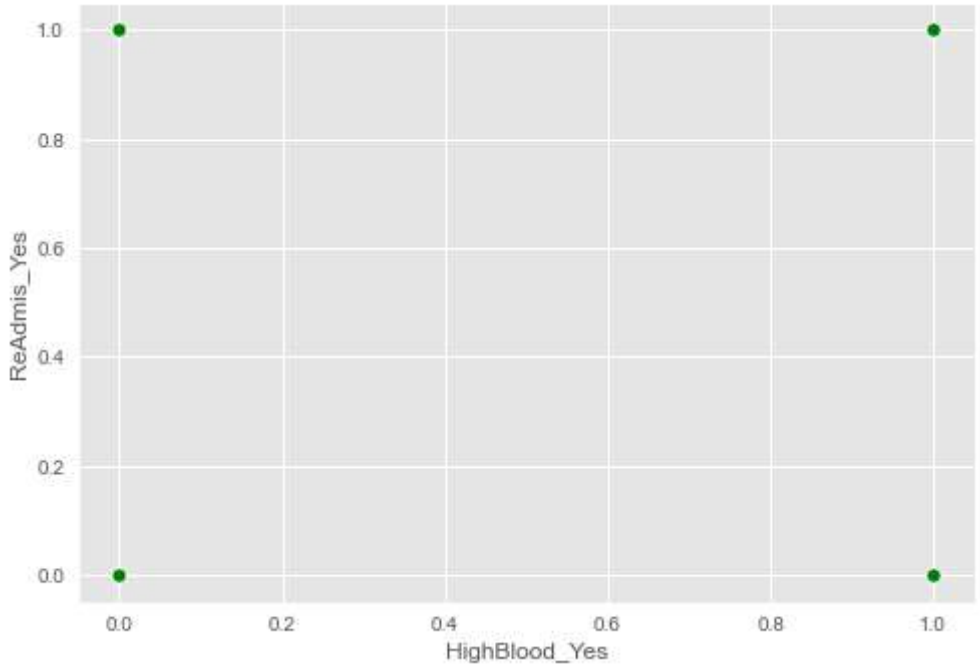
In [20]:
```
#Scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x = df_ready['Doc_visits'], y = df_ready['ReAdmis_Yes'], color='g
plt.show()
```



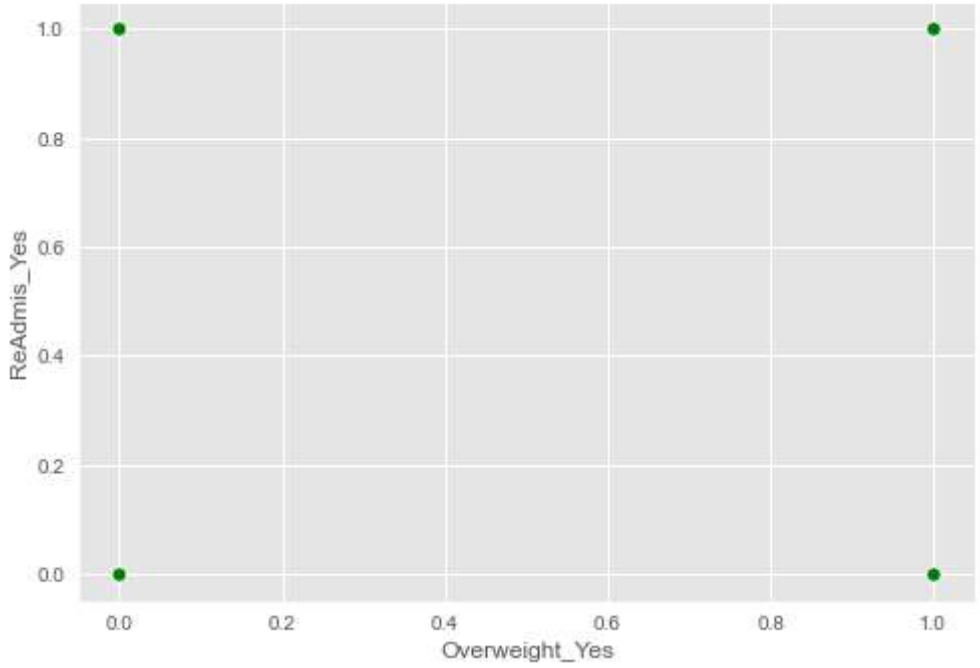
In [21]:
```
#Scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x = df_ready['Full_meals_eaten'], y = df_ready['ReAdmis_Yes'], co
plt.show()
```



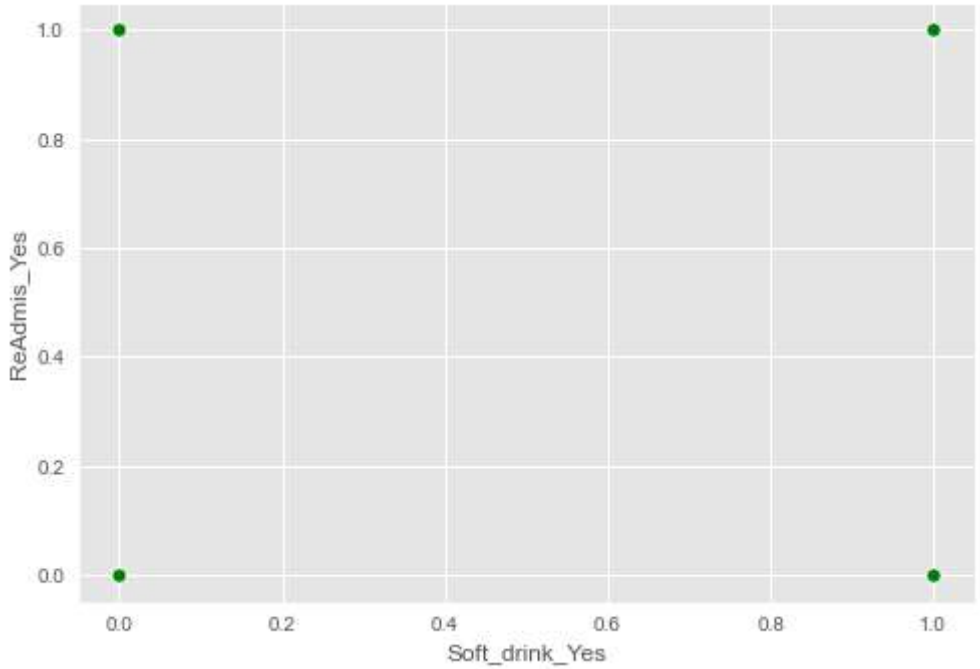
In [22]:
```
#Scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x = df_ready['HighBlood_Yes'], y = df_ready['ReAdmis_Yes'], color
plt.show()
```



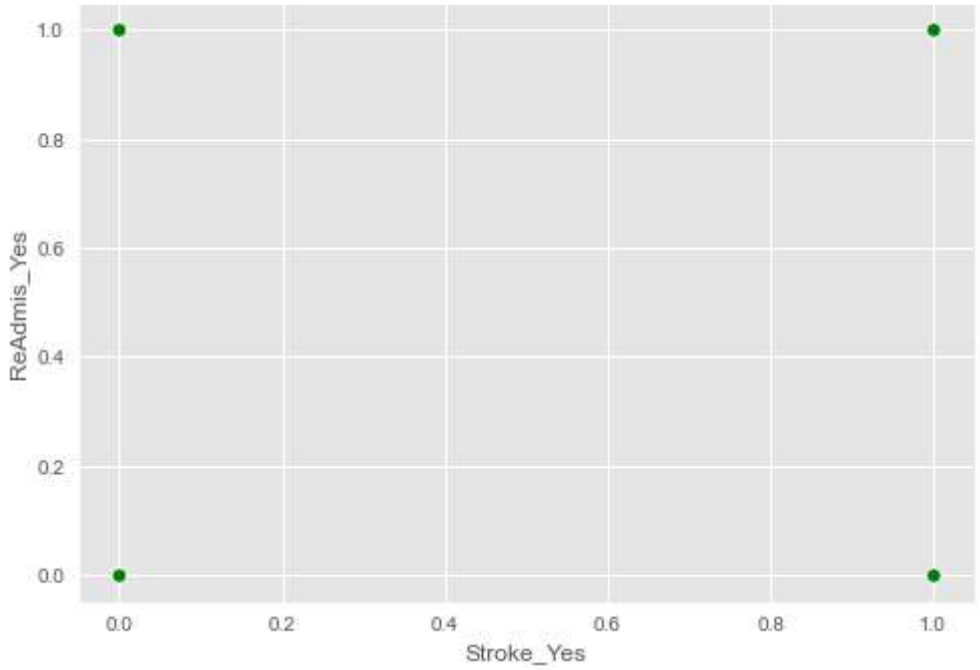
In [23]:
```
#Scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x = df_ready['Overweight_Yes'], y = df_ready['ReAdmis_Yes'], colo
plt.show()
```



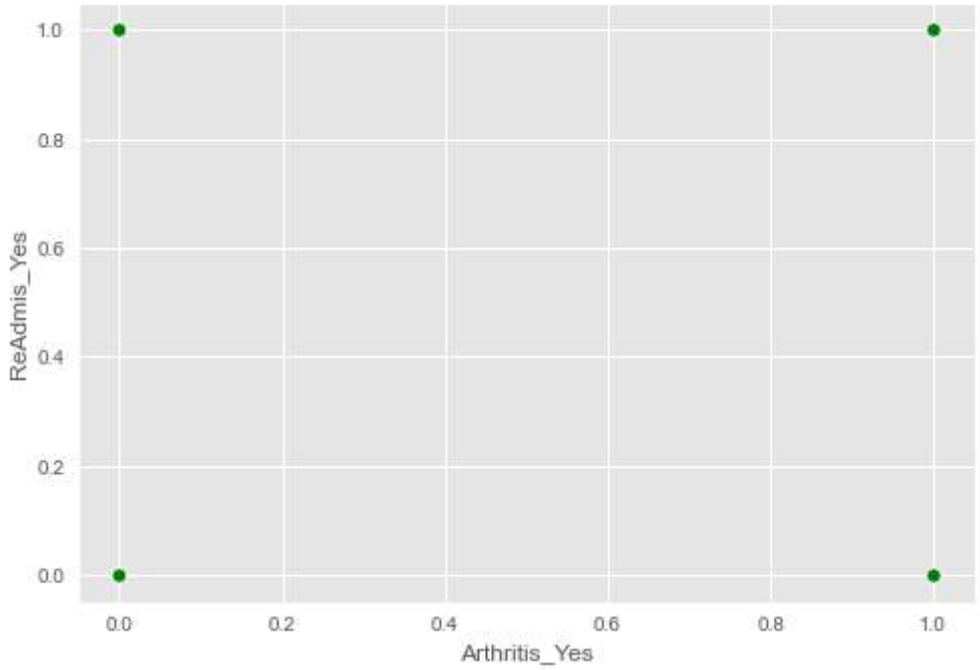
In [24]:
```
#Scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x = df_ready['Soft_drink_Yes'], y = df_ready['ReAdmis_Yes'], colo
plt.show()
```



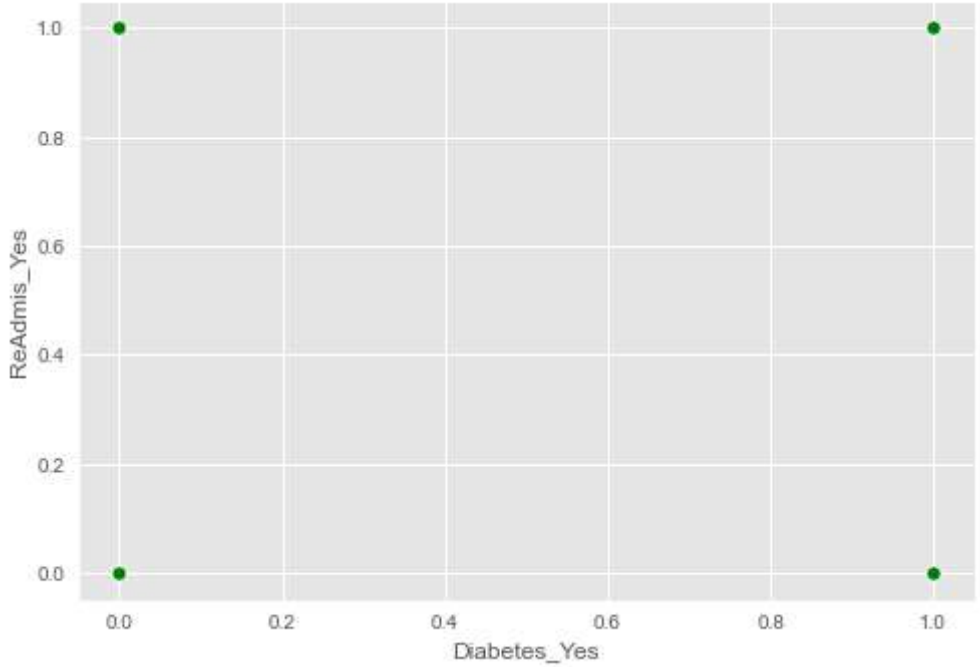
In [25]:
```
#Scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x = df_ready['Stroke_Yes'], y = df_ready['ReAdmis_Yes'], color='g
plt.show()
```



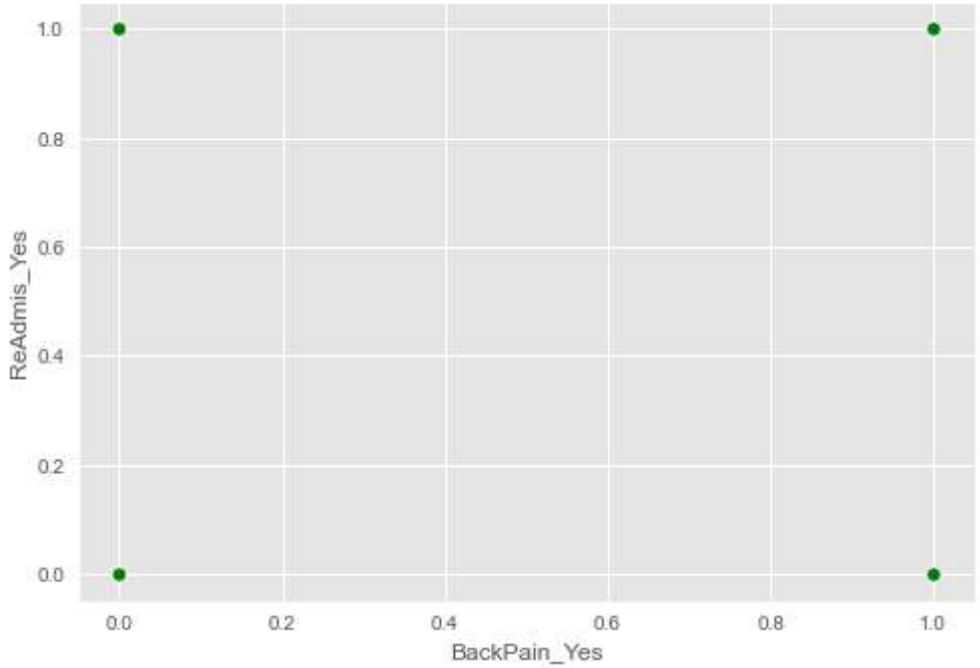
In [26]:
```
#Scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x = df_ready['Arthritis_Yes'], y = df_ready['ReAdmis_Yes'], color
plt.show()
```



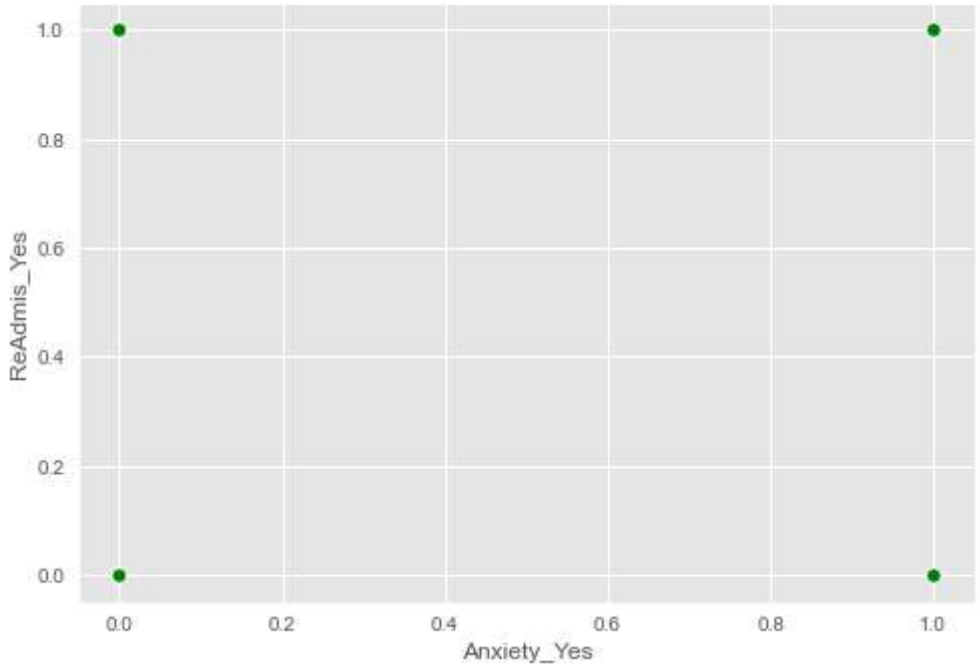
In [27]:
```
#Scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x = df_ready['Diabetes_Yes'], y = df_ready['ReAdmis_Yes'], color=
plt.show()
```



In [28]:
```
#Scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x = df_ready['BackPain_Yes'], y = df_ready['ReAdmis_Yes'], color=
plt.show()
```



In [29]:
```
#Scatterplot to get an idea of correlations between potentially related variables
sns.scatterplot(x = df_ready['Anxiety_Yes'], y = df_ready['ReAdmis_Yes'], color='
plt.show()
```



In [30]:
```
df_uniques = pd.DataFrame([[i, len(df_ready[i].unique())] for i in df_ready.colun
df_ready.nunique()
```

Out[30]:
```
Doc_visits          9
Full_meals_eaten    8
ReAdmis_Yes         2
HighBlood_Yes       2
Overweight_Yes      2
Soft_drink_Yes      2
Stroke_Yes          2
Arthritis_Yes       2
Diabetes_Yes        2
BackPain_Yes        2
Anxiety_Yes         2
dtype: int64
```

In [31]:
```
binary_variables = list(df_uniques[df_uniques['Unique Values'] == 2].index)
binary_variables
```

Out[31]:
```
['ReAdmis_Yes',
 'HighBlood_Yes',
 'Overweight_Yes',
 'Soft_drink_Yes',
 'Stroke_Yes',
 'Arthritis_Yes',
 'Diabetes_Yes',
 'BackPain_Yes',
 'Anxiety_Yes']
```

In [32]:
```
categorical_variables = list(df_uniques[(3 >= df_uniques['Unique Values']) & (df_
categorical_variables
```

Out[32]:
```
[]
```

In [33]:
```
[[i, list(df[i].unique())] for i in categorical_variables]
```

Out[33]:
```
[]
```

In [34]:
```
ordinal_variables = ['Full_meals_eaten']
```

In [35]:
```
numeric_variables = list(set(df_ready.columns) - set(ordinal_variables) - set(cat
```

In [36]:
```
from sklearn.preprocessing import LabelBinarizer, LabelEncoder
lb, le = LabelBinarizer(), LabelEncoder()
```

In [37]:
```
for column in ordinal_variables:
    df_ready[column] = le.fit_transform(df_ready[column])
```

In [38]:
```
for column in binary_variables:
    df_ready[column] = lb.fit_transform(df_ready[column])
```

In [39]:
```
categorical_variables = list(set(categorical_variables) - set(ordinal_variables))
```

In [40]:
```
## to avoid multi-collinearity
df_ready = pd.get_dummies(df_ready, columns = categorical_variables, drop_first=T
```

In [41]:
```
from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler()
for column in [ordinal_variables + numeric_variables]:
    df_ready[column] = mm.fit_transform(df_ready[column])
```

In [42]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_repo

# Set up X and y variables
y, X = df_ready['ReAdmis_Yes'], df_ready.drop(columns='ReAdmis_Yes')

# Split the data into training and test samples
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_s
```

In [43]:
```
#save the data
X_train.to_csv('Documents/PreparedData D209 X Train.csv')
X_test.to_csv('Documents/PreparedData D209 X Test.csv')
y_train.to_csv('Documents/PreparedData D209 Y Train.csv')
y_test.to_csv('Documents/PreparedData D209 Y Test.csv')
```

In [44]:
```
# Initialize KNN model
knn = KNeighborsClassifier(n_neighbors = 11)
# Fit data to KNN model
knn.fit(X_train, y_train)
```

Out[44]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=11)
```

In [45]:
```
# Predict outcomes from test set
y_pred = knn.predict(X_test)
```

In [46]:
```
# Export y_pred dataset
y_pred_df = pd.DataFrame(y_pred)
y_pred_df.to_csv('Documents/PreparedData D209 Y Pred.csv')
```

In [47]:
```
# Print initial accuracy score of KNN model
print('Initial accuracy score KNN model: ', accuracy_score(y_test, y_pred))
```

```
Initial accuracy score KNN model:  0.58325
```

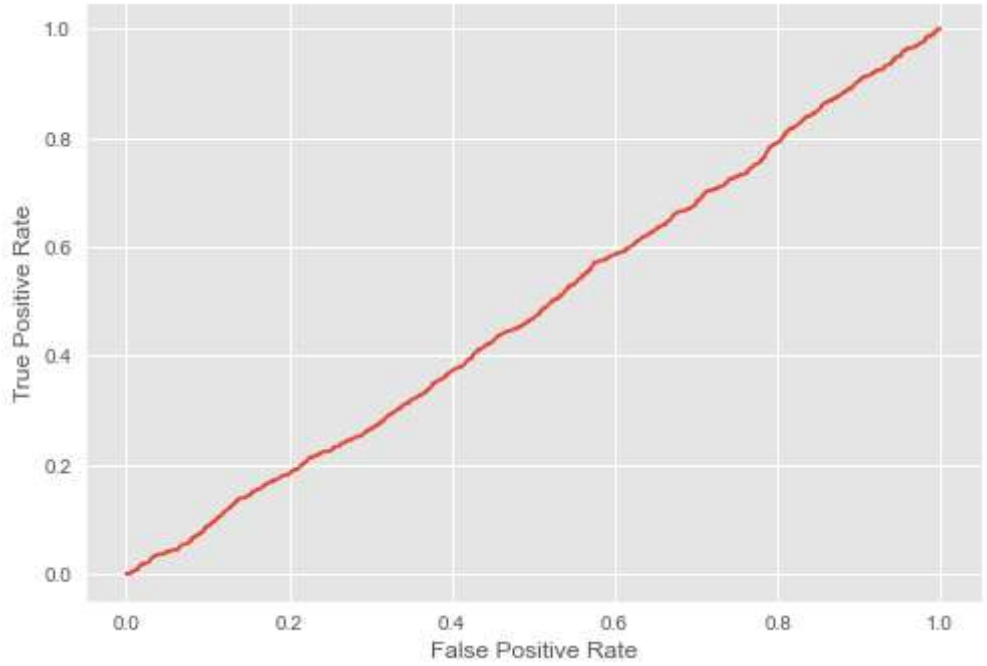
In [48]:
```
# Compute classification metrics
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.64      0.79      0.71      2568
           1       0.36      0.21      0.26      1432

    accuracy                           0.58      4000
   macro avg       0.50      0.50      0.49      4000
weighted avg       0.54      0.58      0.55      4000
```

In [49]:
```
regressor = LogisticRegression()
regressor = regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)


#instantiate the model
log_regression = LogisticRegression()

#fit the model using the training data
log_regression.fit(X_train,y_train)

#calculate AUC of model
auc = metrics.roc_auc_score(y_test, y_pred)

#print AUC score
print(auc)
```

```
0.5
```

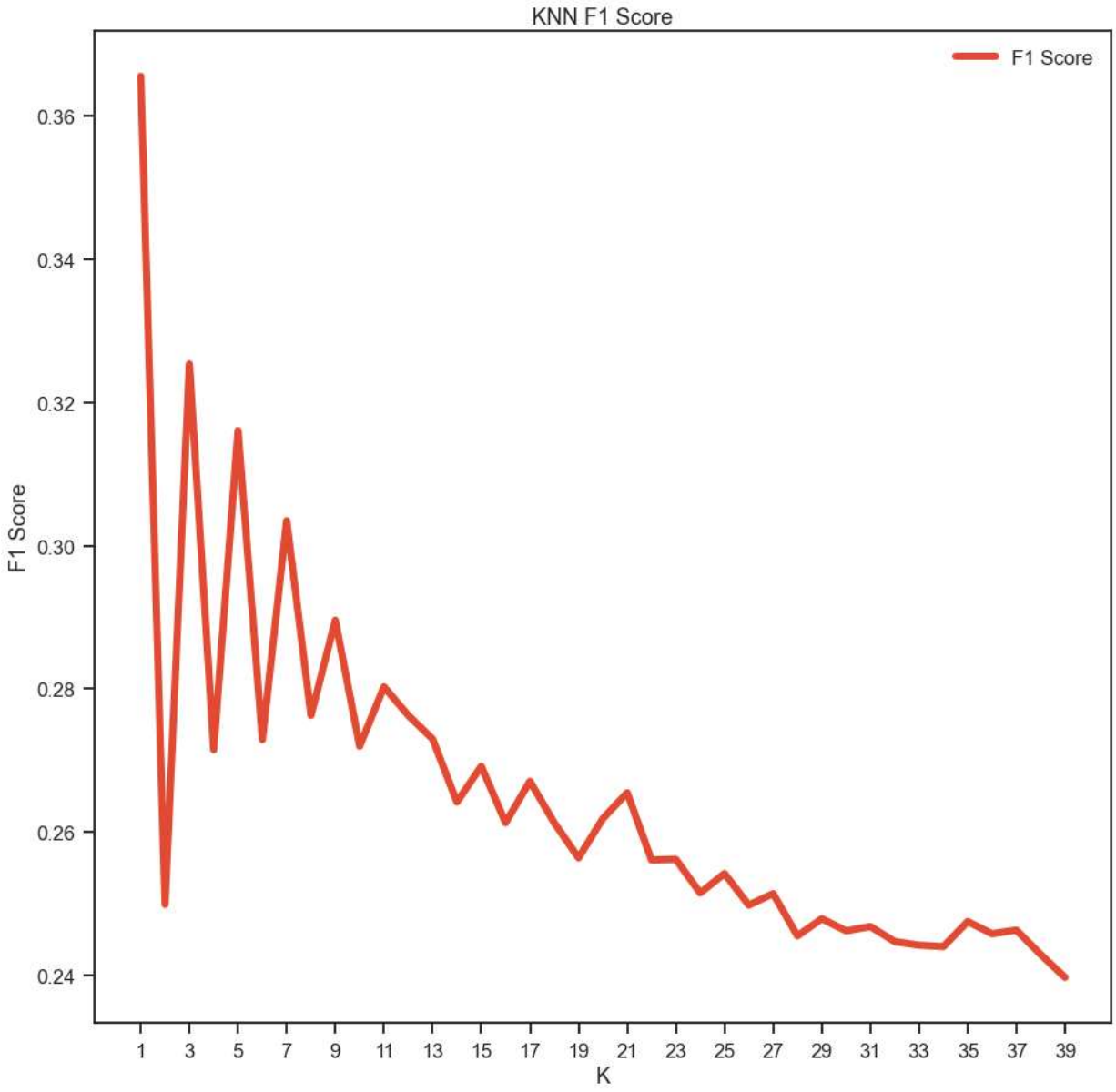
In [50]:
```
#define metrics
y_pred_proba = log_regression.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [51]:
```
# Estimate KNN model and report outcomes
knn = KNeighborsClassifier(n_neighbors=3)
knn = knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

max_k = 40
f1_scores = list()
error_rates = list() # 1-accuracy

# keep initialiting the KNN and look at the F1 score and look at the one that max
for k in range(1, max_k):

    knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
    knn = knn.fit(X_train, y_train)

    y_pred = knn.predict(X_test)
    f1 = f1_score(y_pred, y_test)
    f1_scores.append((k, round(f1_score(y_test, y_pred), 4)))
    error = 1-round(accuracy_score(y_test, y_pred), 4)
    error_rates.append((k, error))

f1_results = pd.DataFrame(f1_scores, columns=['K', 'F1 Score'])
error_results = pd.DataFrame(error_rates, columns=['K', 'Error Rate'])
```

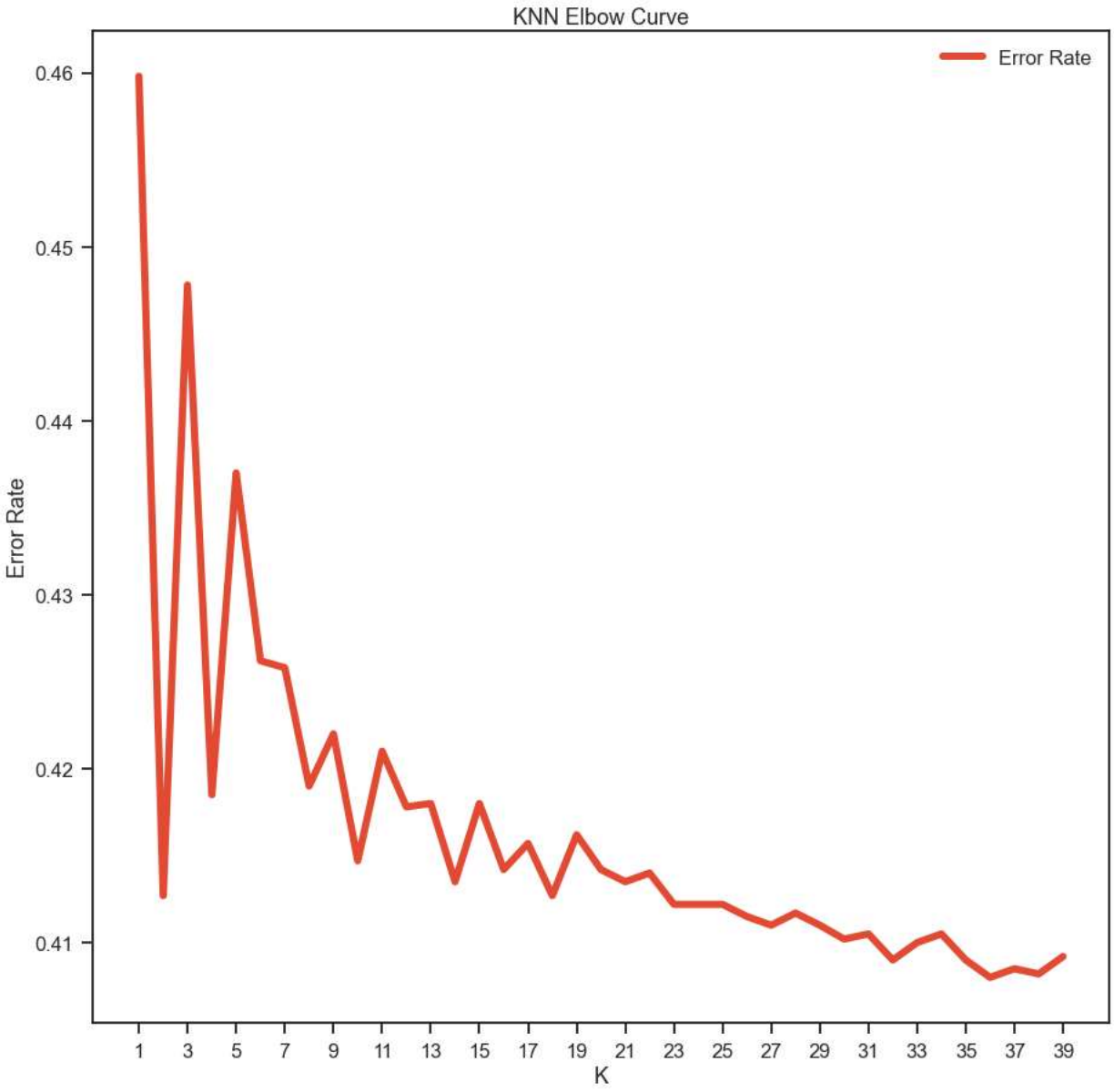
In [52]:
```
# Plot F1 results
sns.set_context('talk')
sns.set_style('ticks')

plt.figure(dpi=300)
ax = f1_results.set_index('K').plot(figsize=(15, 15), linewidth=6)
ax.set(xlabel='K', ylabel='F1 Score')
ax.set_xticks(range(1, max_k, 2));
plt.title('KNN F1 Score')
plt.show()
```

```
<Figure size 2400x1650 with 0 Axes>
```



In [53]:
```
# Plot Accuracy (Error Rate) results
sns.set_context('talk')
sns.set_style('ticks')

plt.figure(dpi=300)
ax = error_results.set_index('K').plot(figsize=(15, 15), linewidth=6)
ax.set(xlabel='K', ylabel='Error Rate')
ax.set_xticks(range(1, max_k, 2))
plt.title('KNN Elbow Curve')
plt.show()
```

```
<Figure size 2400x1650 with 0 Axes>
```



In [54]:
```
# Set steps for pipeline object
steps = [('scaler', StandardScaler()),
 ('knn', KNeighborsClassifier())]
```

In [55]:
```
# Initiate pipeline
pipeline = Pipeline(steps)
```

In [56]:
```
# Split dataframe
X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = train_test_split()
```

In [57]:
```
# Scale dateframe with pipeline object
knn_scaled = pipeline.fit(X_train_scaled, y_train_scaled)
```

In [58]:
```
# Predict from scaled dataframe
y_pred_scaled = pipeline.predict(X_test_scaled)
```

In [59]:
```
# Print new accuracy score of scaled KNN model
print('New accuracy score of scaled KNN model: {:0.3f}'.format(accuracy_score(y_t
```

```
New accuracy score of scaled KNN model: 0.560
```

In [60]:
```
# Compute classification metrics after scaling
print(classification_report(y_test_scaled, y_pred_scaled))
```

```
              precision    recall  f1-score   support

           0       0.63      0.75      0.68      1261
           1       0.36      0.24      0.28       739

    accuracy                           0.56      2000
   macro avg       0.49      0.49      0.48      2000
weighted avg       0.53      0.56      0.54      2000
```

In [61]:
```
#Confusion_matrix & generate results
cf_matrix = confusion_matrix(y_test, y_pred)
print(cf_matrix)
```

```
[[2105  463]
 [1174  258]]
```

In [62]:
```
# Visualize confusion matrix
group_names = ['True Neg', 'False Pos', 'False Neg', 'True Pos']
group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten()/np.
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names,group_counts,grou
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

Out[62]: <AxesSubplot:>

In [63]:
```
# Set up parameters grid
param_grid = {'n_neighbors': np.arange(1, 50)}
# Re-initializing KNN for cross validation
knn = KNeighborsClassifier()
# Initializing GridSearch cross validation
knn_cv = GridSearchCV(knn , param_grid, cv=5)
# Fit model to
knn_cv.fit(X_train, y_train)
# Print best parameters
print('Best parameters for this KNN model: {}'.format(knn_cv.best_params_))
```

```
Best parameters for this KNN model: {'n_neighbors': 48}
```

In [64]:
```
# Generate model best score
print('Best score for this KNN model: {:.3f}'.format(knn_cv.best_score_))
```

```
Best score for this KNN model: 0.618
```

In [65]:
```
# Fit it to the data
knn_cv.fit(X, y)
```

Out[65]:

GridSearchCV

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])})
```

estimator: KNeighborsClassifier

```
KNeighborsClassifier()
```

KNeighborsClassifier

```
KNeighborsClassifier()
```

In [66]:
```
# Compute predicted probabilities: y_pred_prob
y_pred_prob = knn_cv.predict_proba(X_test)[:,1]
```

In [67]:
```
# Compute and print AUC score
print("The Area under curve (AUC) on validation dataset is: {:.4f}".format(roc_au
```

```
The Area under curve (AUC) on validation dataset is: 0.5658
```

In [68]:
```
# Compute cross-validated AUC scores: cv_auc
cv_auc = cross_val_score(knn_cv, X, y, cv=5, scoring='roc_auc')
```

In [69]:
```
# Print list of AUC scores
print("AUC scores computed using 5-fold cross-validation: {}".format(cv_auc))
```

```
AUC scores computed using 5-fold cross-validation: [0.51136091 0.48909813 0.503
86658 0.49898089 0.50502882]
```

In [ ]: