D208 - Predictive Modeling (Task 2)

By: Jacob Colp

A1.

Will a patient be a long-term or short-term stay on their initial admin?

To make it more apparent how this will be answered with logistic regression - we will define longterm stay as >7 days. Then we can use feature engineering to classify the patients that we have data on. Therefore - a True value would be a long-term stay and a False value would be short-term. We have implemented something similar on my team at work. It uses a more advanced classification model, but we have implemented the model for hospice patients. We found it far more beneficial in a clinical setting and more accurate than attempting to estimate the actual number of days a patient would be in hospice. Kind of morbid, but it does help greatly in informing care strategy.

A2.

The ultimate goal of my data analysis will be to identify key features in classifying patients as long-term or short-term stay patients. This is a slight itteration (or generalization - depending on how you view it) on the linear model that I implemented in the first assessment. The ideal outcome would be to implement this model in a clinical setting in a way that can inform patient care. From attending nursing school - it is well known in the medical space that a patients perception of the environment can greatly impact recovery. If a caretaker knows that a patient is going to spend an extended time in their care - it may be beneficial to make certain changes in the way they are cared for. Can we be less aggressive in our assessments to allow them time to rest? Can we have their family bring in things from home that make them feel comfortable? Can we have better continuity of care so they have time to build relationships with their care team? Can we stagger procedures to reduce stress? All of these are critical decisions that can greatly impact patient outcomes. If a patient can be classified based off of data gathered during an initial assessment - the decisions can be made earlier and hopefully improve patient outcomes and experience.

B1.

There are 5 basic assumptions within the implementation of a logistic regression model. The assumptions are as follows:

- Response variable is binary: In simplest terms - this going to mean that our dependent vraible has one of two outcomes. One outcome is going to be affirmative and one will indicate

opposition. Some examples of that would 1 and 0, or True and False, etc.. In our research question - this will be indicated by long-term stay or short-term stay. Our affirmative response being long-term.

- Observations are independent: This is going to mean that our observations are not repeating. We are not going to have duplicate individuals. Each person gets one measurement and one measurement only.

- No Multicollinearity in Independent Variables: This assumption has a big word, so it is perhaps the most intimidating at this juncture. However, I believe it is easiest to understand with an example from our current dataset. When choosing variables to feed our model - we are looking for variables that provide unique information. Variables that provide unique variance. Take a look at initial days and total charge. Their variance is almost identical. However, when you stop to think about it - that makes perfect sense! They are effectively communicating the same and the longer you stay in the hospital - the more it is going to cost.

- No Extreme Outliers: This is going to be true for most predictive analytics. Edge cases will not benefit a model and may only serve to confuse a given model.

- Sufficient Sample Size: A logistic model is going to assume that there is an appropriate amount of observervations for different outcomes within our independent variables. The more rare a given outcome - the the fewer observations needed. The calculation to decide if we do in fact have a large enough sample size is the number of independent variables (i.e. 7) multiplied by 10 and divided by the probablity of the least frequent outcome (i.e. 30%). Written out - this would like (7*10)/.30 = 233. So, in this instance - we would need 233 observations of our rare combination.

(https://www.voxco.com/blog/logistic-regression-assumptions/)

B2.

I have utilized Python for all of my projects leading up to this point. The most base reasoning for that is its flexibility. In a real world setting - it is significantly to implement systems built in Python. In theory it would allow us to easily build applications for end users to interact with our outcomes. Python is far more prevelant in environments where scalability is a consideration (i.e. hospitals). Within the market - R tends to be more niche for academia and pure statistics. My team and I have utilized R from time to time, but that is mostly in one-time analysis problem sets where we are less concerned with the deliverable.

Outside of its flexibility - there is also an awesome community of people who dedicate a lot of time to developing packages for Python. A nice explanation of some packages that will be seen within this project can be found here: https://towardsdatascience.com/3-top-python-packages-to-learn-statistic-for-data-scientist-d753b76e6099. Leveraging these packages gives me, as an end-user, a very easy interface to perform my analysis. Using Scikit-Learn, you could very easily instantiate and train a model in just a few lines of code! Pandas and Numpy give me a great

interface to ask, and answer, questions of my data. The dataframe implementation makes data feel familiar in its tabular form. Finally, packages like Seaborn and Matplotlib give great visualizations that can be created in just a single line of code.

In short - I find Python to be flexible, yet powerful. The syntax is second nature and allows me to make data meaningful to an external stakeholder. I can give them ways to touch and tinke with data, that don't require them to have knowledge of a programming language. Throw a plotly visualization into a quick flask app - suddenly you have an interactive playground for data. And, all of it can be done in less than 50 lines of code.

B3.

My research question is simply trying to categorize patients. Is this patient going to be here for an extended period of time? Yes or no? 1 or 0? Logistic regression is utilized within these types of problem sets. It is a model that seeks to categorize observations based upon historical data. Within a clinical setting - it gives caretakers the ability to make quick decisions. Were this to be applied in a real world scenario - we could present a simple yes or no prediction within a patients chart. Do we expect this patient to be with us for a time greater than a week? Yes or no? That type of quick data point can inform everything from the first to the last interaction and it can be communicated very quickly. It isn't ambiguous. There is no middle ground.

Additionally - I have knowledge of the efficacy of a linear regression model with this dataset (from my previous project). It yielded lackluster results at best. However, perhaps by bringing it up a level, in terms of specificity - we can provide true benefit to our stakeholders.

(https://learn.g2.com/logistic-regression)

In [ ]:
```
import pandas as pd
import numpy as np

import seaborn as sns
from matplotlib import pyplot

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.utils import resample
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

In [ ]:
```
med_df = pd.read_csv(r'medical_clean.csv')

med_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   CaseOrder           10000 non-null  int64  
 1   Customer_id         10000 non-null  object 
 2   Interaction         10000 non-null  object 
 3   UID                 10000 non-null  object 
 4   City                10000 non-null  object 
 5   State               10000 non-null  object 
 6   County              10000 non-null  object 
 7   Zip                 10000 non-null  int64  
 8   Lat                 10000 non-null  float64
 9   Lng                 10000 non-null  float64
 10  Population          10000 non-null  int64  
 11  Area                10000 non-null  object 
 12  TimeZone            10000 non-null  object 
 13  Job                 10000 non-null  object 
 14  Children            10000 non-null  int64  
 15  Age                 10000 non-null  int64  
 16  Income              10000 non-null  float64
 17  Marital             10000 non-null  object 
 18  Gender              10000 non-null  object 
 19  ReAdmis             10000 non-null  object 
 20  VitD_levels         10000 non-null  float64
 21  Doc_visits          10000 non-null  int64  
 22  Full_meals_eaten    10000 non-null  int64  
 23  vitD_supp           10000 non-null  int64  
 24  Soft_drink          10000 non-null  object 
 25  Initial_admin       10000 non-null  object 
 26  HighBlood           10000 non-null  object 
 27  Stroke              10000 non-null  object 
 28  Complication_risk   10000 non-null  object 
 29  Overweight          10000 non-null  object 
 30  Arthritis           10000 non-null  object 
 31  Diabetes            10000 non-null  object 
 32  Hyperlipidemia      10000 non-null  object 
 33  BackPain            10000 non-null  object 
 34  Anxiety             10000 non-null  object 
 35  Allergic_rhinitis   10000 non-null  object 
 36  Reflux_esophagitis  10000 non-null  object 
 37  Asthma              10000 non-null  object 
 38  Services            10000 non-null  object 
 39  Initial_days        10000 non-null  float64
 40  TotalCharge         10000 non-null  float64
 41  Additional_charges  10000 non-null  float64
 42  Item1               10000 non-null  int64  
 43  Item2               10000 non-null  int64  
 44  Item3               10000 non-null  int64  
 45  Item4               10000 non-null  int64  
 46  Item5               10000 non-null  int64  
 47  Item6               10000 non-null  int64  
 48  Item7               10000 non-null  int64  
 49  Item8               10000 non-null  int64  
dtypes: float64(7), int64(16), object(27)
memory usage: 3.8+ MB
```

In [ ]:
```
med_df.describe()
```

Out[ ]:

	CaseOrder	Zip	Lat	Lng	Population	Children	
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00
mean	5000.50000	50159.323900	38.751099	-91.243080	9965.253800	2.097200	53.51
std	2886.89568	27469.588208	5.403085	15.205998	14824.758614	2.163659	20.63
min	1.00000	610.000000	17.967190	-174.209700	0.000000	0.000000	18.00
25%	2500.75000	27592.000000	35.255120	-97.352982	694.750000	0.000000	36.00
50%	5000.50000	50207.000000	39.419355	-88.397230	2769.000000	1.000000	53.00
75%	7500.25000	72411.750000	42.044175	-80.438050	13945.000000	3.000000	71.00
max	10000.00000	99929.000000	70.560990	-65.290170	122814.000000	10.000000	89.00

8 rows × 23 columns

In [ ]:
```
#Essentially boiler plate code at this point. I have just copy and pasted this from my
#This just renames our survey question columns to be less ambiguous

med_df.rename({'Item1':'Survey_Timely_Admission', 'Item2':'Survey_Timely_Treatment', '

med_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
 #   Column                                 Non-Null Count  Dtype  
---  ------                                 --------------  -----  
 0   CaseOrder                              10000 non-null  int64  
 1   Customer_id                            10000 non-null  object 
 2   Interaction                            10000 non-null  object 
 3   UID                                    10000 non-null  object 
 4   City                                   10000 non-null  object 
 5   State                                  10000 non-null  object 
 6   County                                 10000 non-null  object 
 7   Zip                                    10000 non-null  int64  
 8   Lat                                    10000 non-null  float64
 9   Lng                                    10000 non-null  float64
 10  Population                             10000 non-null  int64  
 11  Area                                   10000 non-null  object 
 12  TimeZone                               10000 non-null  object 
 13  Job                                    10000 non-null  object 
 14  Children                               10000 non-null  int64  
 15  Age                                    10000 non-null  int64  
 16  Income                                 10000 non-null  float64
 17  Marital                                10000 non-null  object 
 18  Gender                                 10000 non-null  object 
 19  ReAdmis                                10000 non-null  object 
 20  VitD_levels                            10000 non-null  float64
 21  Doc_visits                             10000 non-null  int64  
 22  Full_meals_eaten                       10000 non-null  int64  
 23  vitD_supp                              10000 non-null  int64  
 24  Soft_drink                             10000 non-null  object 
 25  Initial_admin                          10000 non-null  object 
 26  HighBlood                              10000 non-null  object 
 27  Stroke                                 10000 non-null  object 
 28  Complication_risk                      10000 non-null  object 
 29  Overweight                             10000 non-null  object 
 30  Arthritis                              10000 non-null  object 
 31  Diabetes                               10000 non-null  object 
 32  Hyperlipidemia                         10000 non-null  object 
 33  BackPain                               10000 non-null  object 
 34  Anxiety                                10000 non-null  object 
 35  Allergic_rhinitis                      10000 non-null  object 
 36  Reflux_esophagitis                     10000 non-null  object 
 37  Asthma                                 10000 non-null  object 
 38  Services                               10000 non-null  object 
 39  Initial_days                           10000 non-null  float64
 40  TotalCharge                            10000 non-null  float64
 41  Additional_charges                     10000 non-null  float64
 42  Survey_Timely_Admission                10000 non-null  int64  
 43  Survey_Timely_Treatment                10000 non-null  int64  
 44  Survey_Timely_Visits                   10000 non-null  int64  
 45  Survey_Reliability                     10000 non-null  int64  
 46  Survey_Options                         10000 non-null  int64  
 47  Survey_Hours_of_Treatment              10000 non-null  int64  
 48  Survey_Courteous_Staff                 10000 non-null  int64  
 49  Survey_Evidence_of_Active_Listening    10000 non-null  int64  
dtypes: float64(7), int64(16), object(27)
memory usage: 3.8+ MB
```

In [ ]:
```
#Essentially boiler plate code at this point. I have just copy and pasted this from my
#This encodes categorical variables into numeric values. For our binary - it replaces
```

```
med_df.Complication_risk.replace({'High':3,'Medium':2,'Low':1}, inplace=True)

for x in med_df:
    if med_df[x].dtype == 'object':
        med_df[x].replace({'Yes':1,'No':0}, inplace=True)
    else:
        continue

for x in med_df:
    print(med_df[x].value_counts())
```

```
1        1
6671     1
6664     1
6665     1
6666     1
        ..
3334     1
3335     1
3336     1
3337     1
10000    1
Name: CaseOrder, Length: 10000, dtype: int64
C412403    1
D294364    1
B203210    1
C20177     1
K216020    1
          ..
J694995    1
N704840    1
A197688    1
H115454    1
I569847    1
Name: Customer_id, Length: 10000, dtype: int64
8cd49b13-f45a-4b47-a2bd-173ffa932c2f    1
dc1799a6-61d1-44a3-9b94-b89584baddfc    1
dc6bab10-659b-4c78-ba87-87ffa3def32f    1
1cb70cc8-47b7-4192-8bed-faad0f27ab3b    1
06d0da86-5600-472d-a35d-7632775b5cd7    1
                                      ..
2d5c049d-0431-443f-a9d7-46f875998599    1
4ccc7838-5c2c-4a3c-9e3b-2a646cec157e    1
4767cd0e-626d-4c5a-834e-a948aba315c2    1
8b1e1ea3-e596-4a97-b2cb-7f9926e9fbee    1
bc482c02-f8c9-4423-99de-3db5e62a18d5    1
Name: Interaction, Length: 10000, dtype: int64
3a83ddb66e2ae73798bdf1d705dc0932    1
5d85418e862aab28ed18975446153694    1
28970195a839f74e572d2f43bd7fcb71    1
24665d1c5070b579d0c190324599ca6d    1
2d42fd6803d96fcffe64d3fcaab430dc    1
                                   ..
ce28b38948f9737918e240d8ce6b275f    1
d98f314ddd12ca99e05858219127a06e    1
43e47521f56f27d0fc8e00ba7141de1b    1
ffd9be64812b5e36ba9367abe51be705    1
95663a202338000abdf7e09311c2a8a1    1
Name: UID, Length: 10000, dtype: int64
Houston             36
San Antonio         26
Springfield         22
New York            21
Miami               21
                    ..
Coyote               1
Tiline               1
Monon                1
Sullivans Island     1
Coraopolis           1
Name: City, Length: 6072, dtype: int64
```

```
TX    553
CA    550
PA    547
NY    514
IL    442
OH    383
MO    328
FL    304
VA    287
IA    276
MI    273
MN    267
NC    254
GA    247
KS    220
WI    214
KY    210
OK    207
WV    207
IN    195
AL    194
TN    194
WA    191
AR    190
NE    185
CO    179
NJ    176
LA    173
MA    149
MS    134
MD    131
SC    128
SD    123
ME    122
OR    122
MT    112
NM    110
ID    109
ND    108
AZ    108
CT     80
NH     79
UT     72
AK     70
VT     60
NV     51
WY     51
PR     43
HI     34
DE     17
RI     14
DC     13
Name: State, dtype: int64
Jefferson      118
Washington     100
Franklin        93
Los Angeles     88
Montgomery      80
               ...
Jenkins          1
```

```
Sully           1
Panola          1
Kandiyohi       1
Sterling        1
Name: County, Length: 1607, dtype: int64
25674    4
78104    4
68355    4
62098    4
24136    4
        ..
18337    1
58442    1
71353    1
81612    1
15108    1
Name: Zip, Length: 8612, dtype: int64
36.06702    4
33.34798    4
35.25512    4
39.38610    4
37.86890    4
           ..
41.00911    1
39.20560    1
46.36035    1
34.96563    1
40.49998    1
Name: Lat, Length: 8588, dtype: int64
-90.30464     4
-94.39542     4
-80.59756     4
-84.59579     4
-95.56405     4
             ..
-121.03097    1
-74.00429     1
-76.95300     1
-111.96820    1
-80.19959     1
Name: Lng, Length: 8725, dtype: int64
0        109
195       14
115       11
178       11
285       11
        ...
8092       1
11147      1
27175      1
7371       1
41524      1
Name: Population, Length: 5951, dtype: int64
Rural       3369
Suburban    3328
Urban       3303
Name: Area, dtype: int64
America/New_York                  3889
America/Chicago                   3771
America/Los_Angeles                937
```

```
America/Denver                    612
America/Detroit                   262
America/Indiana/Indianapolis      151
America/Phoenix                   100
America/Boise                      86
America/Anchorage                  50
America/Puerto_Rico                43
Pacific/Honolulu                   34
America/Menominee                  14
America/Nome                       12
America/Indiana/Vincennes           8
America/Kentucky/Louisville         6
America/Sitka                       6
America/Toronto                     5
America/Indiana/Marengo             3
America/Indiana/Tell_City           3
America/North_Dakota/Beulah         2
America/Yakutat                     1
America/Indiana/Winamac             1
America/Indiana/Knox                1
America/North_Dakota/New_Salem      1
America/Indiana/Vevay               1
America/Adak                        1
Name: TimeZone, dtype: int64
Outdoor activities/education manager    29
Exhibition designer                     27
Theatre director                        27
Scientist, audiological                 26
Toxicologist                            25
                                        ..
Government social research officer       6
Phytotherapist                           6
Engineer, control and instrumentation    6
Public relations account executive       6
Licensed conveyancer                     6
Name: Job, Length: 639, dtype: int64
0     2548
1     2509
3     1489
2     1475
4      995
7      213
8      209
6      191
5      169
9      108
10      94
Name: Children, dtype: int64
47    161
52    159
74    159
41    157
86    156
     ...
63    123
51    122
20    120
36    118
80    116
Name: Age, Length: 72, dtype: int64
```

```
14572.40    2
20474.03    2
37132.97    2
29508.62    2
24997.02    2
           ..
41900.29    1
35093.92    1
44848.08    1
20815.96    1
62682.63    1
Name: Income, Length: 9993, dtype: int64
Widowed          2045
Married          2023
Separated        1987
Never Married    1984
Divorced         1961
Name: Marital, dtype: int64
Female       5018
Male         4768
Nonbinary     214
Name: Gender, dtype: int64
0    6331
1    3669
Name: ReAdmis, dtype: int64
18.135431    2
15.939760    2
17.821860    2
20.184170    2
18.741340    2
            ..
18.825293    1
16.849021    1
15.111106    1
20.583694    1
18.388620    1
Name: VitD_levels, Length: 9976, dtype: int64
5    3823
6    2436
4    2385
7     634
3     595
8      61
2      58
1       6
9       2
Name: Doc_visits, dtype: int64
0    3715
1    3615
2    1856
3     612
4     169
5      25
6       6
7       2
Name: Full_meals_eaten, dtype: int64
0    6702
1    2684
2     544
3      64
```

```
4       5
5       1
Name: vitD_supp, dtype: int64
0    7425
1    2575
Name: Soft_drink, dtype: int64
Emergency Admission      5060
Elective Admission       2504
Observation Admission    2436
Name: Initial_admin, dtype: int64
0    5910
1    4090
Name: HighBlood, dtype: int64
0    8007
1    1993
Name: Stroke, dtype: int64
2    4517
3    3358
1    2125
Name: Complication_risk, dtype: int64
1    7094
0    2906
Name: Overweight, dtype: int64
0    6426
1    3574
Name: Arthritis, dtype: int64
0    7262
1    2738
Name: Diabetes, dtype: int64
0    6628
1    3372
Name: Hyperlipidemia, dtype: int64
0    5886
1    4114
Name: BackPain, dtype: int64
0    6785
1    3215
Name: Anxiety, dtype: int64
0    6059
1    3941
Name: Allergic_rhinitis, dtype: int64
0    5865
1    4135
Name: Reflux_esophagitis, dtype: int64
0    7107
1    2893
Name: Asthma, dtype: int64
Blood Work     5265
Intravenous    3130
CT Scan        1225
MRI             380
Name: Services, dtype: int64
63.544320    2
67.421390    2
70.325420    2
63.334690    1
67.036510    1
            ..
5.977596     1
5.799041     1
```

```
6.415853     1
7.328631     1
70.850590    1
Name: Initial_days, Length: 9997, dtype: int64
7555.452000    2
7964.681000    2
8081.346000    2
3726.702860    1
8449.859000    1
               ..
2345.477165    1
3672.779714    1
3392.003760    1
3866.635381    1
7887.553000    1
Name: TotalCharge, Length: 9997, dtype: int64
24109.572640    3
25325.816470    3
6315.622130     3
11995.005160    3
4880.460246     3
                ..
23226.177810    1
12415.288840    1
28931.863720    1
7219.133007     1
11643.190000    1
Name: Additional_charges, Length: 9418, dtype: int64
4    3455
3    3404
5    1377
2    1315
6     225
1     213
7      10
8       1
Name: Survey_Timely_Admission, dtype: int64
3    3439
4    3351
5    1421
2    1360
1     213
6     204
7      12
Name: Survey_Timely_Treatment, dtype: int64
4    3464
3    3379
5    1358
2    1356
6     220
1     211
7      11
8       1
Name: Survey_Timely_Visits, dtype: int64
3    3422
4    3394
5    1388
2    1346
6     231
1     207
```

```
7      12
Name: Survey_Reliability, dtype: int64
4    3446
3    3423
2    1380
5    1308
6     219
1     211
7      13
Name: Survey_Options, dtype: int64
4    3464
3    3371
5    1403
2    1319
6     220
1     213
7      10
Name: Survey_Hours_of_Treatment, dtype: int64
4    3487
3    3456
2    1345
5    1274
1     215
6     212
7      11
Name: Survey_Courteous_Staff, dtype: int64
3    3401
4    3337
5    1429
2    1391
6     221
1     209
7      12
Name: Survey_Evidence_of_Active_Listening, dtype: int64
```

In [ ]:
```
#This will be the code to feature engineer our dependent variable
#We will classify our patients by their initial length of stay - either being greater

med_df['Long_Term_Stay'] = med_df['Initial_days'].apply(lambda x: 1 if x > 7 else 0)

#We can print out our value counts here and see that our apply properly mapped 1 and 0
med_df['Long_Term_Stay'].value_counts()
```

Out[ ]:
```
1    7810
0    2190
Name: Long_Term_Stay, dtype: int64
```

C1.

This will mark the midway point for my data preparation. Given pre-existing knowledge of the dataset - I am aware that it is clean and has no need for imputation (no missing values). However, I have made a few adjustments to the data to better fit my preference. I find that having numeric values is easier to work with in the totality of a given project. It also communicates an ordinal value for model to better interpret relationship. Additionally - I have converted Yes and No values to binary values. While not necessarily required for specific models - it does keep interactions more consistent. Following that step - I went ahead with our feature

engineering to generate the long-term and short-term categorization of our existing records. Post this commentary - I will implement feature scaling to limit impact of different scales amongst the different variables. Then I will split the dataset into a training and test set to allow for validation of the model. Additionally, I will run some checks to ensure that logistic regression assumptions are met (namely multicolinearity). The goals and preparation will remain similar to the previous to our previous implementation of linear regression.

(https://www.kdnuggets.com/2019/07/data-pre-processing-optimizing-regression-model-performance.html)

C2.

Summary statistics can be found in the second cell following our imports. They will be output from the describe method. That said - I believe it would be a bit excessive to try and have meaningful discussion surrounding the summary statistics for all of our variables. Therefore, I would like to point out some interesting behavior and draw attention to the inherent limitations of summary statistics. First, I would like to draw your attention to a few means. If you take a look at both population and income - there is a very broad range of values. With minimums in the hundreds and maximums in the hundreds of thousands. Income's mean feels a bit more believable than population, but both feel to poorly represented by measure of central tendency. In this instance - it may make more sense to group values into ranges and find a median to represent our general population. Ultimately, those measures are meant to represent the studied population, but a mean can be greatly impacted by outliers. And with our extreme variance, evidenced by the large standard deviation - I believe that is what is happening here. Were we to begin making decisions purely off this information - we could end up woefully ill informed.

As an interesting exercise - I am going to reuse the same variables that I selected in my first submission. I am curious if this model will perform better by keeping it more generalized. So, our list of independent variables will be as follows:

- Age
- Income
- VitD_Levels
- Doc_Visits
- Soft_Drink
- HighBlood
- Complication_Risk
- Overweight
- Stroke

To reiterate - our dependent variable will be our "Long_Term_Stay" column we engineered. This is a column filled with binary values that indicate whether a patient stayed more than 7 days in the hospital. Interestingly - we have far more long-term stays than short-term stays. I would

have thought we would see more people staying in hospitals less than a week. However, we do have a solid number of observations to start off with.

C3.

Outside of the feature engineering that we have implemented - all transformations that have taken place so far are code snippets that I have utilized in the past when working with this dataset. I initially start by renaming the survey items to map to their questions. Then, I replace the complication risks with numeric values to make an ordinal relationship easier to understand for a model. The for loop that follows searches for object data types and then replaces yes and no's with binary values. I do this so that I don't have to go through and specify the columns to do this operation on. Following these operations - I do a quick apply method to create my dependent variable. Post this commentary - I will be implementing a few steps that are more specific to building a regression model. First, I will scale the data to help with the drastically different data scales. Then, I will check for multicollinearity. Finally, I will be splitting our dataset into a training and test set. That should fully prepare us to step right into our logistic regression!

(https://www.kdnuggets.com/2019/07/data-pre-processing-optimizing-regression-model-performance.html)

In [ ]:
```
#Here I will be scaling the dataset to better account for the broad distribution of di
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScale
#This is me writing a note to myself to ensure that I inverse the transformation to ge

scaler = preprocessing.StandardScaler()

med_df_scaled = pd.DataFrame(scaler.fit_transform(med_df[['Age', 'Income', 'VitD_level

#Here I am splitting out the outcome variable into its own data frame for the eventual
Long_Term_Stay_Df = med_df.Long_Term_Stay
predictor_variables_df = med_df_scaled

predictor_variables_df
```

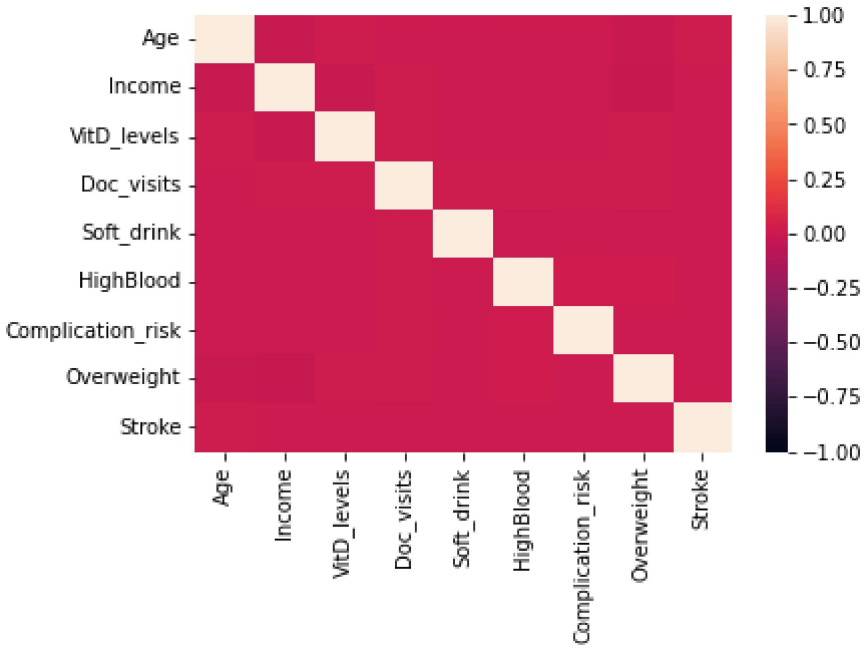
Out[ ]:

	Age	Income	VitD_levels	Doc_visits	Soft_drink	HighBlood	Complication_risk	Overwe
0	-0.024795	1.615914	0.583603	0.944647	-0.588898	1.202076	-0.168873	-1.562
1	-0.121706	0.221443	0.483901	-0.967981	-0.588898	1.202076	1.200737	0.640
2	-0.024795	-0.915870	0.046227	-0.967981	-0.588898	1.202076	-0.168873	0.640
3	1.186592	-0.026263	-0.687811	-0.967981	-0.588898	-0.831894	-0.168873	-1.562
4	-1.526914	-1.377325	-0.260366	-0.011667	1.698086	-0.831894	-1.538483	-1.562
...	...	...	...	...	...	...	...	
9995	-1.381548	0.192047	-0.487525	-0.967981	-0.588898	1.202076	-0.168873	-1.562
9996	1.622691	-0.894380	0.105476	-0.011667	-0.588898	1.202076	-0.168873	0.640
9997	-0.412438	0.891569	-0.414049	-0.967981	1.698086	1.202076	1.200737	0.640
9998	-0.509349	-0.378271	0.964820	-0.011667	-0.588898	-0.831894	-0.168873	0.640
9999	0.798948	0.778133	0.210377	-0.011667	-0.588898	-0.831894	-1.538483	0.640

10000 rows × 9 columns

In [ ]:
```
#Here I am generating a heatmap to check for any multicollineartiy. I expand the minim

sns.heatmap(predictor_variables_df.corr(), vmin=-1.0)

#We've seen this before! We're definitely not seeing any multicollinearity!
```

Out[ ]:
```
<AxesSubplot:>
```



In [ ]:
```
#Here I am splitting out the training and test sets for our predictor and outcome vari
#https://realpython.com/train-test-split-python-data/

X_Train, X_Test, Y_Train, Y_Test = train_test_split(predictor_variables_df, Long_Term_
```

```
X_Train.reset_index(drop=True)
X_Test.reset_index(drop=True)
Y_Train.reset_index(drop=True)
Y_Test.reset_index(drop=True)
```

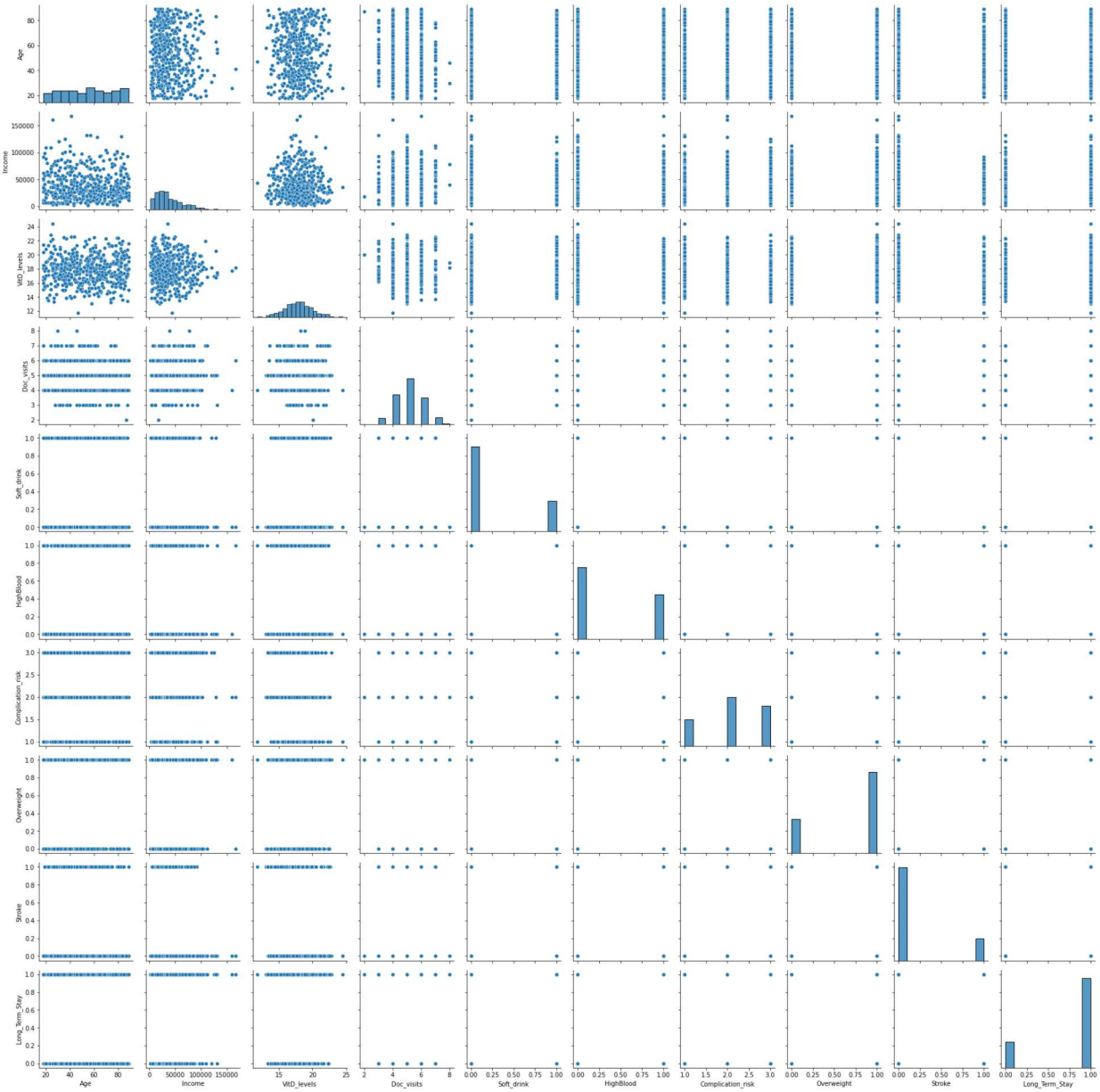
Out[ ]:
```
0       0
1       0
2       1
3       1
4       1
       ..
2495    1
2496    1
2497    1
2498    1
2499    1
Name: Long_Term_Stay, Length: 2500, dtype: int64
```

C4.

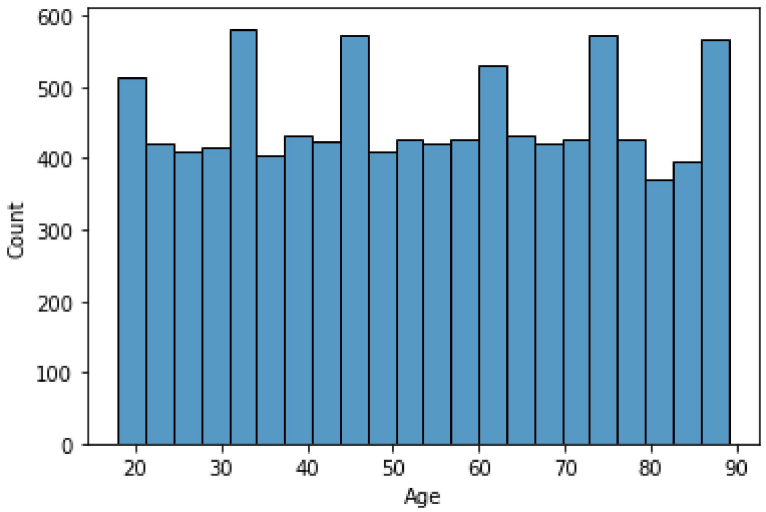
Below you will find the univariate and bivariate visualizations generated. Again, knowing the dataset - it is fairly obvious that there isn't a strong correlation that exists in the dataset, and will be hard to hard to build a model from. By looking at the generated visualizations - there is no directly apparent relationship amongst variables. At the very least - no strong relationships have broken through.

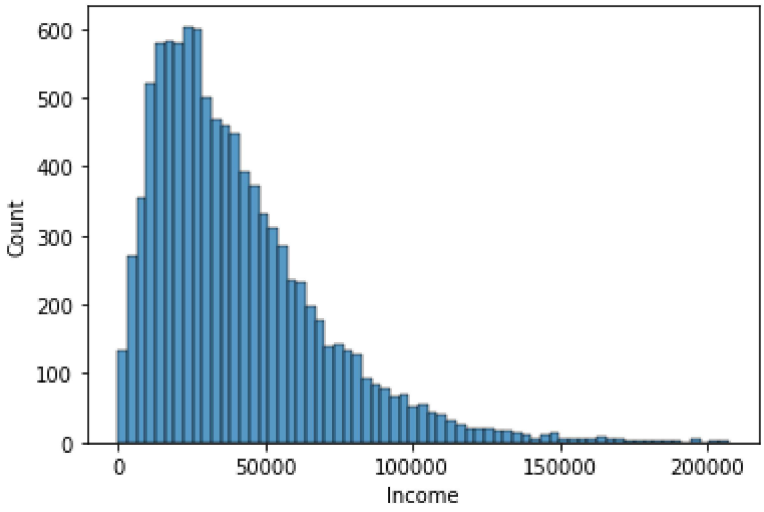
In [ ]:
```
#https://seaborn.pydata.org/tutorial/axis_grids.html
#All Bivariate plots for the cleaned and scaled dataset

g = sns.PairGrid(med_df[['Age', 'Income', 'VitD_levels', 'Doc_visits', 'Soft_drink', '
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```

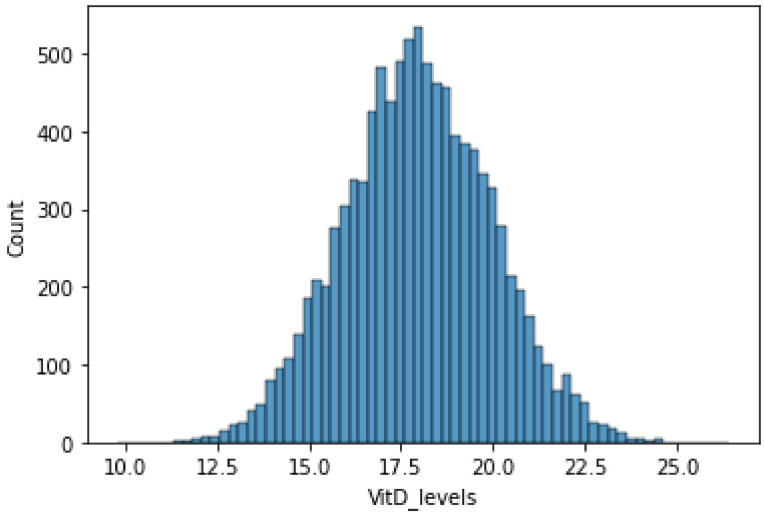
Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x1319d63b880>
```

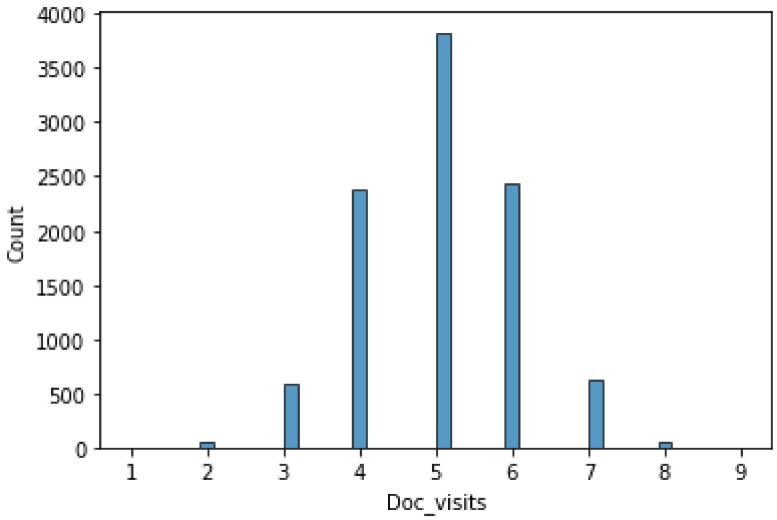


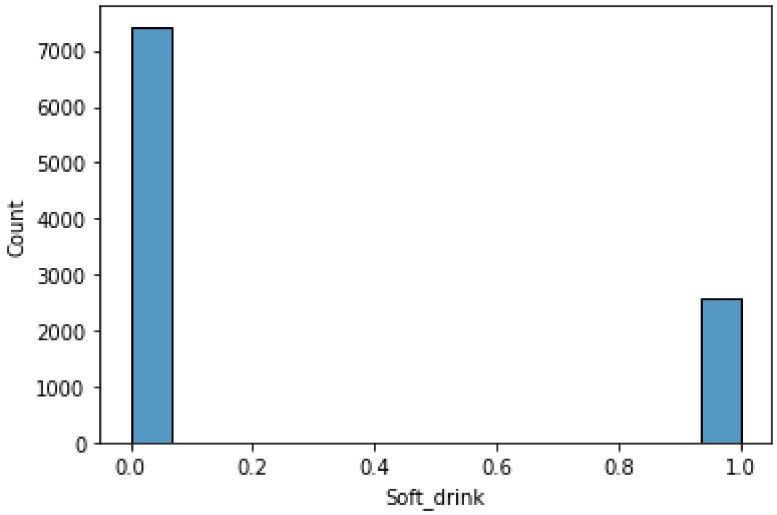
In [ ]:
```
for x in med_df[['Age', 'Income', 'VitD_levels', 'Doc_visits', 'Soft_drink', 'HighBloc
    sns.histplot(x = med_df[x])
    pyplot.show()
```

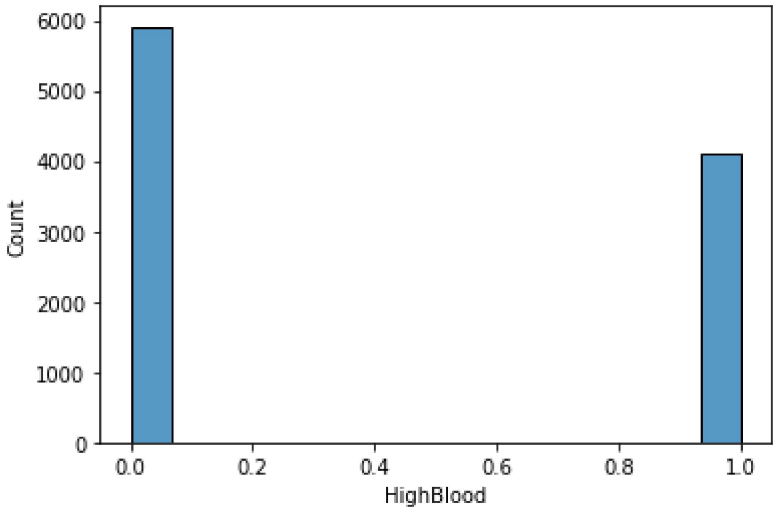


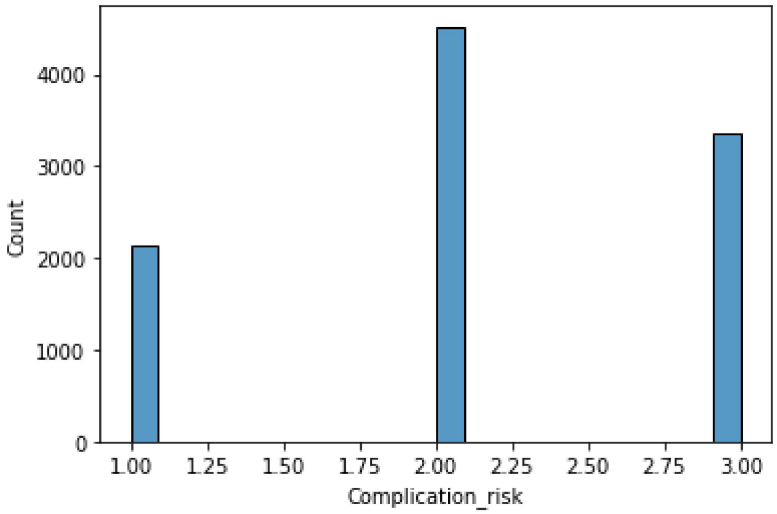

Count
Income
0
50000
100000
150000
200000
0
100
200
300
400
500
600

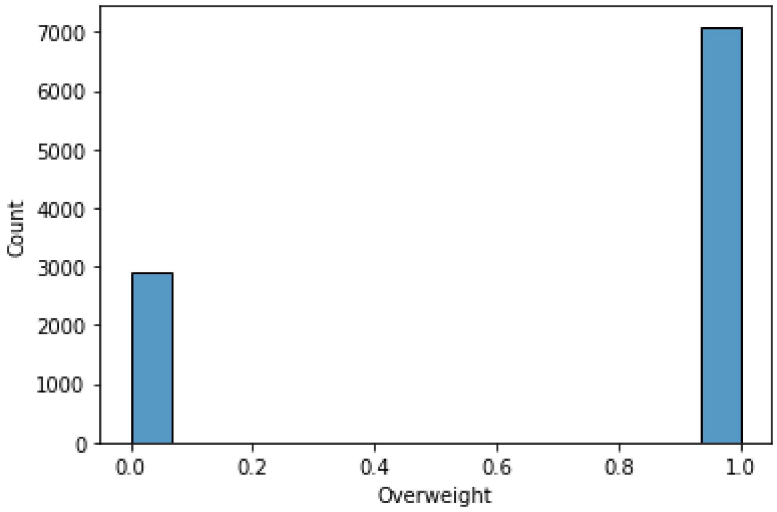

Count
VitD_levels
10.0
12.5
15.0
17.5
20.0
22.5
25.0
0
100
200
300
400
500

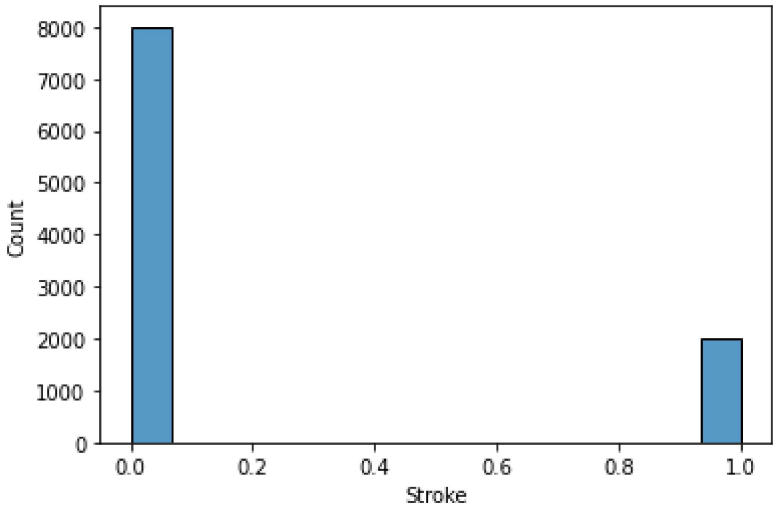

Count
Doc_visits
1
2
3
4
5
6
7
8
9
0
500
1000
1500
2000
2500
3000
3500
4000

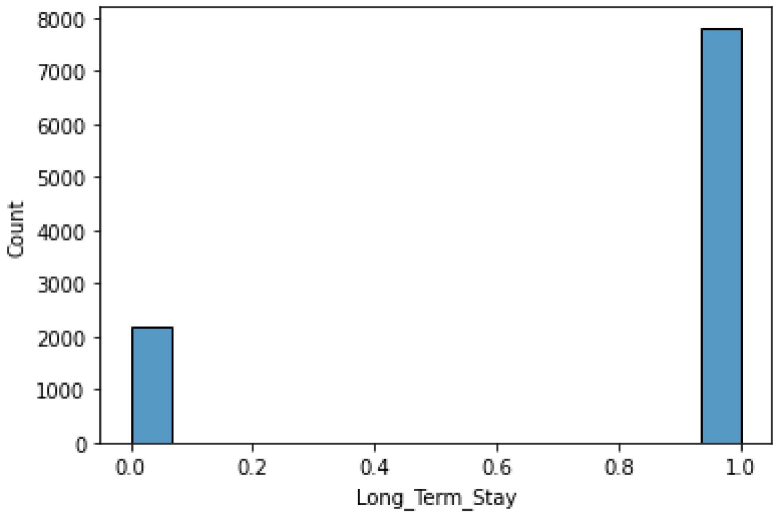

Count
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
Soft_drink


Count
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
HighBlood


Count
4000
3000
2000
1000
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Complication_risk







C5.

I will generate a copy of the prepared dataset below.

In [ ]:
```
med_df_scaled.to_csv('Jcolp_Prepared_Dataset_D208_2.csv')
```

D1.

Below this segment will be the code that implements our initial logistic regression model. There are effectively 4 steps that are going to be utilized here: The first step is to instatiate or model. This is the first variable assignment. Within that function, I have set a seed for the model. This is a parameter that will bring a consistent shuffle to the data everytime that the code is run. That way - we don't get variable performance. Setting that parameter is valuable in this discussion, so I don't end up writing commentary about a number that ends up not being repeatable. Next - I fit that logistic model to the training data and predict off our testing set.

Now, below that are two different outputs. Initially - I am just using the default scoring function attached to the logistic regression model. Take a quick peak at that score! 78% model accuracy! Woah! Genius! But, I got suspicious... that score is good... maybe TOO good. So, I threw together a little confusion matrix to see what our model is predicting. And, I was correct. The model is just throwing 1's against the wall. In fairness to the model - I provided it with an unbalanced dataset. And, it is largely imbalanced towards a positive binary outcome. So, along with a reduced model in the next step - I will also balance the dataset and evaluate performance then. But, effectively this is currently telling us that it is safe to assume that every patient is going to be a longterm patient. That isn't very helpful.

(https://elitedatascience.com/imbalanced-classes)

In [ ]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegres

LR = LogisticRegression(random_state=10)

LR.fit(X_Train, Y_Train)

y_prediction = LR.predict(X_Test)

y_prediction
```

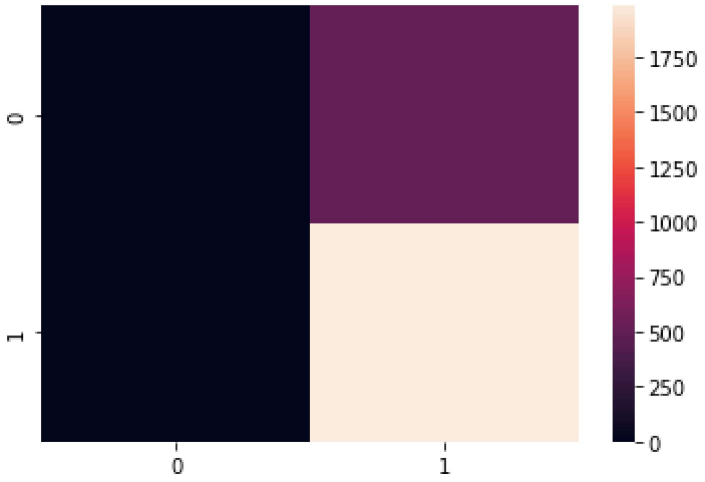
Out[ ]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [ ]:
```
#This function gives the mean accuracy on the test data and labels

LR.score(X_Test,Y_Test)
```

Out[ ]:
```
0.794
```

In [ ]:
```
#https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-h

cm = metrics.confusion_matrix(Y_Test,y_prediction)
sns.heatmap(cm)
```

Out[ ]:
```
<AxesSubplot:>
```



D2.

Below I have implemented both a balancing of data and feature reduction methodology. First, I have downsampled our majority class. We had 7810 patients that qualified as "long-term" stay patients. I briefly discussed this in section D1, but this was ultimately leading to our model only outputting positive binary outcomes. Now, I do have some hesitancy about simply ridding ourselves of valuable data, however this does seem to be an approach that others have implemented. That said - there is differing opinions on its value. But, I wanted to be able to produce a model that would output both positive and negative outcomes, and this was a way to achieve that goal. I could have tuned acceptance criteria, but in evaluating distributions in our bivariate distributions - it didn't appear that there was a strong variation in our classes anyway. This did ultimately did lead to our model successfully outputting both possibilities. But, that did come at the expense of model accuracy.

Following that - we have implemented some other feature selection. Similar to my previous project - I am using the SelectKBest module from the Sklearn package. However, I have changed the variable selection model that is utilized to score the variables. Rather than a regression model - I have utilized the f_classif model. This is a measure of the ANOVA F-Statistic. The easiest way to understand that concept is to know that the larger that F-Statistic is - the bigger difference is there is between group means. So, we're finding variables that give us the greatest degree of variance within the data.

(https://datascience.stackexchange.com/questions/96779/why-does-my-logistic-regression-predict-all-0s) (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selectio

In [ ]:
```
#Value counts to rebalance dataframe

med_df.Long_Term_Stay.value_counts()
```

Out[ ]:
```
1    7810
0    2190
Name: Long_Term_Stay, dtype: int64
```

In [ ]:
```
#I am going to downsample our positive binary outcomes to continue dealing with real v

df_majority = med_df[med_df.Long_Term_Stay==1]
df_minority = med_df[med_df.Long_Term_Stay==0]

df_majority_balanced = resample(df_majority, replace=False, n_samples=2190, random_sta

med_df_balanced = pd.concat([df_minority,df_majority_balanced])

med_df_balanced.Long_Term_Stay.value_counts()
```

Out[ ]:
```
0    2190
1    2190
Name: Long_Term_Stay, dtype: int64
```

In [ ]:
```
#Going back through scaling and splitting with our balanced dataset

scaler = preprocessing.StandardScaler()

med_df_balanced_scaled = pd.DataFrame(scaler.fit_transform(med_df_balanced[['Age', 'In

#Here I am splitting out the outcome variable into its own data frame for the eventual
Long_Term_Stay_Df = med_df_balanced.Long_Term_Stay
predictor_variables_df = med_df_balanced_scaled

predictor_variables_df
```

Out[ ]:

	Age	Income	VitD_levels	Doc_visits	Soft_drink	HighBlood	Complication_risk	Overwe
0	-0.028619	-0.899684	0.023306	-0.972035	-0.585084	1.189350	-0.172058	0.645
1	1.187433	-0.027566	-0.706419	-0.972035	-0.585084	-0.840795	-0.172058	-1.550
2	-1.536523	-1.352067	-0.281486	-0.012703	1.709157	-0.840795	-1.542263	-1.550
3	1.090148	1.425029	0.789742	0.946628	-0.585084	-0.840795	-0.172058	0.645
4	-0.271829	0.517091	0.809788	0.946628	-0.585084	-0.840795	-1.542263	0.645
...	...	...	...	...	...	...	...	
4375	-0.369113	0.679021	0.858930	-0.012703	-0.585084	1.189350	1.198148	0.645
4376	-1.196028	0.370492	1.521924	-0.972035	-0.585084	-0.840795	-0.172058	0.645
4377	-0.952818	-0.968348	0.271679	-0.012703	-0.585084	1.189350	-1.542263	-1.550
4378	-1.536523	4.336613	2.708483	-0.012703	-0.585084	-0.840795	1.198148	0.645
4379	-1.196028	-0.207248	-0.929153	-0.012703	-0.585084	1.189350	-0.172058	0.645

4380 rows × 9 columns

In [ ]:
```
#Here I am splitting out the training and test sets for our predictor and outcome vari
#https://realpython.com/train-test-split-python-data/

X_Train, X_Test, Y_Train, Y_Test = train_test_split(predictor_variables_df, Long_Term_

X_Train.reset_index(drop=True)
X_Test.reset_index(drop=True)
```

```
Y_Train.reset_index(drop=True)
Y_Test.reset_index(drop=True)
```

Out[ ]:
```
0       1
1       0
2       0
3       1
4       0
       ..
1090    0
1091    1
1092    0
1093    1
1094    0
Name: Long_Term_Stay, Length: 1095, dtype: int64
```

D3.

Below you will find the reduced logistic regression model. As previously mentioned - I have utilized the SelectKBest function with a scoring function of f_classif. From that - I am keeping 3 variables. I am keeping the 3 features that have the best scoring from that scoring function. Following that - the code is the exact same as our previous implementation. Upon my current run - I have kept the variables Overweight, Stroke, and Age.

In [ ]:
```
#Here we will do our feature selection

feature_selection = SelectKBest(score_func=f_classif, k=3)

feature_fit = feature_selection.fit(X_Train,Y_Train)

np.set_printoptions(precision=3)
scores_df = pd.DataFrame(feature_fit.pvalues_)

columns_df = pd.DataFrame(X_Train.columns)

feature_scores = pd.concat([columns_df, scores_df], axis=1)

feature_scores.columns = ['Columns', 'Scores']

print(feature_scores.nlargest(3,'Scores'))

#I am using this so in theory this code could be more dynamic. Were this in productior
top_3_columns = feature_scores.nlargest(3,'Scores')

X_Train, X_Test, Y_Train, Y_Test = train_test_split(predictor_variables_df[list(top_3_
```

```
      Columns    Scores
3  Doc_visits  0.800901
1      Income  0.731309
7  Overweight  0.726646
```

In [ ]:
```
#Now lets run back through our logistic regression

LR = LogisticRegression(random_state=10)

LR.fit(X_Train, Y_Train)

y_prediction = LR.predict(X_Test)
```

```
y_prediction
```

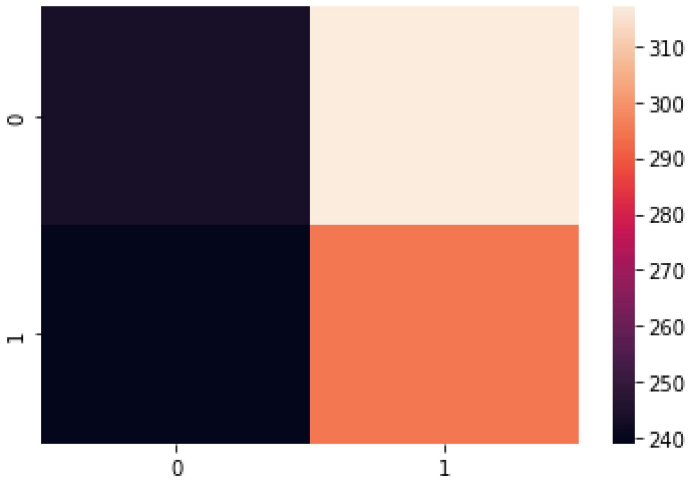
Out[ ]:
```
array([1, 0, 1, ..., 1, 0, 0], dtype=int64)
```

In [ ]:
```
LR.score(X_Test,Y_Test)
```

Out[ ]:
```
0.4922374429223744
```

In [ ]:
```
cm = metrics.confusion_matrix(Y_Test,y_prediction)
sns.heatmap(cm)
```

Out[ ]:
```
<AxesSubplot:>
```



E1.

The goal of the data analysis process was to ultimately produce an initial and reduced logistic regression model. Those models were intended to attempt to model and predict the classification of patients into long-term and short-term stay populations. Initially - I started with the same list of variables that I had utilized in my previous analysis. I believe it to be an interesting exercise in generating understanding around where different models provide value. The independent variables that I initially selected were the ones that felt connected to the question intuitively.

So, the first step in the process was to engineer a feature that grouped our patients into a binary outcome. Effectively - this grouped them into stays that exceeded 7 days and those that didn't.

Following that - we were ready to implement the initial model. Again, the first model was based off of a independent variables that were intuitively connected to the research question. But, following the initial model - I was tasked with reducing the model. So, I elected to use the module from Sklearn called SelectKBest. This is an easy feature of the package that allows us to select the best performing variables based upon different evaluation metrics. For this implementation - I eleceted to utilize the f_classif model. That model computes the ANOVA F-value from the provided data sample. The F-statistic is the ratio of variation between sample means and the variation within samples. In practice - that means that it is identifying the

columns that account for a large variance amongst groups. Since we were evaluating larger classification of data - this was a tangentially related model that seemed to provide value to our end result.

Once we reduced that model - we used the default score method that is attached to the Logistic Regression model in Sklearn. This method returns the mean accuracy of the test data and their labels. There are numerous scoring metrics that could have also been implemented, but this gives a great start for evaluating the efficacy of our model.

(https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selectio (https://www.statology.org/anova-f-value-p-value/) (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_m

E2.

Below I will again print the output from the final implementation of our model.

In [ ]:
```
#Selected features and their f_classif score

print(feature_scores.nlargest(3,'Scores'))
```

```
      Columns    Scores
3  Doc_visits  0.800901
1      Income  0.731309
7  Overweight  0.726646
```

In [ ]:
```
#Array of predicted values for our test dataset

y_prediction
```

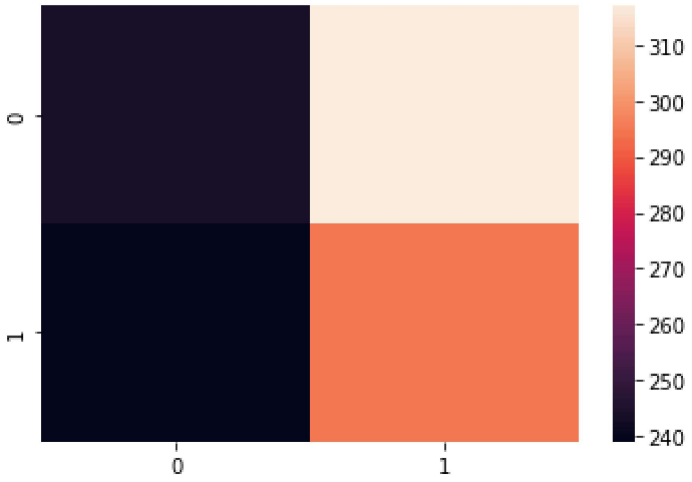
Out[ ]:
```
array([1, 0, 1, ..., 1, 0, 0], dtype=int64)
```

In [ ]:
```
#Default scoring method for logistic regression that provides the mean accuracy of pre

LR.score(X_Test,Y_Test)
```

Out[ ]:
```
0.4922374429223744
```

In [ ]:
```
#Confusion matrix for our model. The Y axis is our predicted values, and x is the test
#This will show the density of wrong/right answers

cm = metrics.confusion_matrix(Y_Test,y_prediction)
sns.heatmap(cm)
```

Out[ ]:
```
<AxesSubplot:>
```



E3.

I utilized several sources when implementing my logistic regression model:

- https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selectio
- https://www.statology.org/anova-f-value-p-value/
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_m
- https://datascience.stackexchange.com/questions/96779/why-does-my-logistic-regression-predict-all-0s
- https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a

These are the sources that are specific to the implementation of the logistic regression model. Two of the sources are specific documentation from sklearn. This is valuable in its explanation of the utilization of their model, and feature selection. Following that - I have some sources that were to answer or backup more specific needs. The first being an article on ANOVA F-statistics. I wanted to make sure that I was understanding the feature scoring model that was being implemented, and how to interpret and reiterate the scores. I also spent some time looking into the reasoning that my initial model was producing a singular output. From that research - it was suggested that my dataset was imbalanced. I had noticed this initially, but was unsure what its ultimate impact would be. The last article from towardsdatascience gave a good overview of some ways to resample our data to correct the imbalance that we noted in the first model.

In [ ]:
```
print(LR.coef_)
print(np.exp(LR.coef_))
print(LR.feature_names_in_)
print(LR.intercept_)

#(https://towardsdatascience.com/interpreting-coefficients-in-linear-and-logistic-regr
```

```
[[-0.039 -0.033 -0.046]]
[[0.962 0.968 0.955]]
['Doc_visits' 'Income' 'Overweight']
[0.016]
```

F1.

In short - our model yielded better results than our multi-linear regression model. Going with a classification model appears to better fit the size and structure of our data.

The equation for our model would look something like: 1/(1+exp(-(-0.014+(1.038(X1))+(0.964(X2))+(0.965(X3))))). This is just the logistic function with our intercept and coefficients placed appropriately. But, would ulitmately plot our model with the input of different real numbers.

The coefficients have been printed out in the above cell. To interpret the coeffcients - I have converted them to regular odds from the log odds. Now that they are converted - we can more easily understand their impact on the data. To interpret these - we would say that for each unit of our independent variable that increases, the odds that the observation in the "Long-Term Stay" class are (**above cofficient**) times larger as the odds that they are not in that class. In my most recent run of the model - Doc_Visits, Income and Overweight have the larger score from our ANOVA scoring function. So, to fully explain the model outcome when applied to new data utilizing our coefficients - for every unit increase of doctors visit (the doctor visits again), the probability of being a long term stay patient is multiplied by .962! So, a more clear way to interpret this is to understand that each unit of increase will resault in a 3.8% reduction in probability of a long-term stay encounter.

While I would like to say that our model has a large practical and statistical significance - that is just not true. Comparative to our linear regression - it is a huge improvement. However, our reduced model consistently performs worse than the flip of a coin. Simply by looking at our final score - we can easily recognize that this model would have little impact in production.

As stated in my previous projects - a large portion of the issues in our model can be attributed to our data set. However, there were some other avenues that could been explored to potentially improve our final outcome. For example, we could have attempted upsampling our minority class and see what impact that had on the model. Or, we could have decreased the threshold for decision our model was taking. Essentially, we could have given our model a bit more permission to select a negative binary outcome. Several sources referenced this as a potential solution. Additionally, were this a real world implementation - I would have seen this as a better proof of concept for a classification model. With the middling success of this implementation - it indicates to me that we are headed in a better direction than linear regression. So, we could have expanded on that classification with a more nuanced model.

F2.

Finally, we come to my recommendation. Based upon my findings here - I would start with the recommendation to reevaluate our data set. If this were a sample of our patient population - perhaps we could ask for additional short-term stay patient records to better fill out our classes. However, I do feel like this implementation was a strong indicator that a more generalized classification model is the appropriate way forward for our data set (excluding any massive increase in data volume). If we could expand the data set, and implement a better classification model - we may have a really useful outcome for our clinicians. But, based upon our findings from the initial model - I would be able to tell staff that it is safe to assume patients are going to be spending more than 7 days in the hospital. This would allow them to better tailor their care plan to that mindset. That said - I would love to expand on this and provide better accuracy to bring a high level of confidence to my recommendations.

Sources

Benton, J. (2020, July 22). Interpreting coefficients in linear and logistic regression. Medium. https://towardsdatascience.com/interpreting-coefficients-in-linear-and-logistic-regression-6ddf1295f6f1

Galarnyk, M. (2021, November 17). Logistic regression using Python (scikit-learn). Medium. https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a

Why does my logistic regression predict all 0's? (n.d.). Data Science Stack Exchange. https://datascience.stackexchange.com/questions/96779/why-does-my-logistic-regression-predict-all-0s

Sklearn.linear_model.LogisticRegression. (n.d.). scikit-learn. Retrieved May 7, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sk

How to interpret the F-value and P-value in ANOVA. (2021, December 7). Statology. https://www.statology.org/anova-f-value-p-value

Sklearn.feature_selection.f_classif. (n.d.). scikit-learn. Retrieved May 7, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif

Python, R. (2020, November 23). Split your dataset with scikit-learn's train_test_split() – Real Python. Python Tutorials – Real Python. https://realpython.com/train-test-split-python-data/

Galarnyk, M. (2021, November 17). Logistic regression using Python (scikit-learn). Medium. https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a

Sklearn.linear_model.LogisticRegression. (n.d.). scikit-learn. Retrieved May 7, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

How to handle Imbalanced classes in machine learning. (2020, May 23). EliteDataScience. https://elitedatascience.com/imbalanced-classes

Building structured multi-plot grids — Seaborn 0.11.2 documentation. (n.d.). seaborn: statistical data visualization — seaborn 0.11.2 documentation. https://seaborn.pydata.org/tutorial/axis_grids.html

Sklearn.preprocessing.StandardScaler. (n.d.). scikit-learn. Retrieved May 7, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

From data pre-processing to optimizing a regression model performance. (n.d.). KDnuggets. https://www.kdnuggets.com/2019/07/data-pre-processing-optimizing-regression-model-performance.html

Joby, A. (n.d.). What is logistic regression? Learn when to use it. Learn Hub | G2. https://learn.g2.com/logistic-regression

Wijaya, C. Y. (2021, October 12). 3 top Python packages to learn statistic for data scientist. Medium. https://towardsdatascience.com/3-top-python-packages-to-learn-statistic-for-data-scientist-d753b76e6099

Logistic regression assumptions. (2022, February 7). Voxco. https://www.voxco.com/blog/logistic-regression-assumptions/