D208 - Predictive Modeling (Task 1)

By: Jacob Colp

A1.

What features best predict initial admissions day?

A2.

The goal of the data analysis is to identify important features in distinguishing the number of days that a patient will spend in the hospital. Being able to identify this will allow the hospital to better strategize staffing, admission and overall patient care. Rather than leaving patients on more acute floors - we may be able to transfer them to long term units in an effort to enhance care for all parties.

B1.

Multiple Linear Regression follows the same assumptions as simple linear regression. Those assumptions are as follows (https://www.scribbr.com/statistics/multiple-linear-regression/):

- Homogeneity of Variance (Homoscedasticity): This essentially means that the variance of observations is similar among variables. So, if we look at them on a scatter plot - their spread should look pretty similar. There are specific ways to test this, but that is the most simple explanation! (https://blog.minitab.com/en/statistics-and-quality-data-analysis/dont-be-a-victim-of-statistical-hippopotomonstrosesquipedaliophobia)
- Independence of Observations (No multicolinearity): So, I think the easiest way to understand this is with an example. And, for ease - lets use the current dataset. One instance that I have observed of multicolinearity exists between initial days and total charge. If you were to blindly evaluate the relationships between all of our variables - you would see their relationship and think "Woah!" But, if we pause for just a second and think, you would also recognize that this makes perfect sense. And, these are really going to explain the same variance within the data. You would expect the price to increase with the number of days
- Normality: Data needs to follow a normal distribution
- Linearity: The line of best fit through the data points is a straight line, and not curved.

B2.

There are several advantages to utilizing the Python programming language in any form of analysis. In this instance - I believe that the largest advantage is the fact that there are several packages available to complete this particular problem. The Python community has done a

great job of creating packages that address data problems at different levels. A great explanation of a few popular packages can be found here: https://towardsdatascience.com/3-top-python-packages-to-learn-statistic-for-data-scientist-d753b76e6099. With these packages - a lot of heavy lifting has been done by the developers and it allows you to focus more on the analysis than the algorithm itself. Outside of that - I prefer Python to R for its overall flexibility and support. Python makes the ingestion and exploration of data really easy with Pandas and Numpy. Visualization packages like Seaborn and Matplotlib allow you to visualize different parts of your analysis to gain visual insight. Finally, my preference lies in the Python syntax. I have utilized R a few times, but I have never found the syntax to be as intuitive. I will concede that I learned Python first, and I am sure that is the main cause, but it holds a special place in my heart.

B3.

My research question seeks to understand variables that are going to best explain initial admission days. For a hospital - I believe that being able to understand those factor will serve both the patient and hospital better. Doctors will be able to appropriately set expectations with patients and better be able to triage them throughout their stay. I believe that multiple linear regression is the appropriate solution for this question, because it allows us to understand the interplay of multiple variables. It would be naive to assume that any accurate estimation could be made for a patients stay based upon only a single factor. Being able to play around with different variable combinations for solid correlation will allow us to identify those key variables. It will also help to alleviate any impact from confounding variables (Spiegelhalter, 2019). We will better be able to fully evaluate the impact of all variables on our dependent variable.

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 LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
```

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

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
```

C1.

My main data preparation goals will differ somewhat from a real-world project. I benefit from the pre-existing knowledge of a clean and prepared data set (validated above with the info function). This removes the need for any kind of imputation. However, I prefer to convert ordinal sting values to a more clear ordinal numeric value. I will also convert Yes/No columns to binary values. Outside of these steps - I will also split the dataset into a training/test set and scale numeric features. Splitting data into a training and test set will allow us to validate the efficacy of any model. Scaling features will normalize values around a 0 mean. Scaling will help us to minimize the impact of the different scales of measurement that may be found in our data set. I will also be checking for multicollinearity in this step as well. That will help to narrow the feature selection and ensure that we are not overfitting by having variability explained by multiple features. (https://www.kdnuggets.com/2019/07/data-pre-processing-optimizing-regression-model-performance.html)

C2.

Specific summary statistics can be found above as output by the describe function. There are 50 columns with 10,000 non-null rows a piece. At initial glance - the most outstanding thing is drastically different scales of data. Populations and Incomes reach into the hundreds or thousands where other features, such as children, will max out around 10. Other interesting behavior can be found in those same feature and offer a sobering reminder of where summary statistics can fail. The max of population is just over 122,000 and the minimum is 0. That places the mean value around 9,000. Without looking at the distribution of data - I can say, with a high level of confidence, that this mean is does not best represent this particular feature. Perhaps a median would be a better singular value to represent central tendency, or maybe this feature's behavior is too complex to be understood by a single metric. I would also like to draw attention to our dependent feature (Initial_Days), because it would appear to have similar behavior. The mean may be representative of our data set, but I would struggle to believe that the average hospital stay is over a month. That in and of itself may be cause for concern in our data collection process. The sample may not be representative of the underlying population. My list of predictor variables will be:

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

These predictor variables are subject to change based upone further exploration. These are variables that I found initially intriguing for their usability in a clinical setting. Knowing this data set - I may end up utilizing PCA. Over this project and previous - I have not noticed a strong

correlation between variables. PCA may help in expanding their usability by combining variables to add a higher level of context to individual observations.

C3.

Some of this code is effectively boiler plate code that I have taken from previous projects working with this dataset. The first step that I take is converting the complication risk text column to a column of ordinal values, so a model can imply their relationship. Within that same code block - I convert other Yes/No object columns to binary values. For scaling and utilization within our regression - it is better to be dealing with numeric values. I then scale the dataset utilizing standard scaler. Following that - I break out our dependent variable into its own DF for our eventual split of the training and test sets. I do a brief check to verify that there is not any multicollinearity between our predictor variables and then split out our training and test sets of data. Ultimately - these steps prepare the data for the model. (https://www.kdnuggets.com/2019/07/data-pre-processing-optimizing-regression-model-performance.html)

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 [ ]:

```
#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', 'Initial_da

#Here I am splitting out the outcome variable into its own data frame for the eventual
Initial_days_df = med_df_scaled.Initial_days
predictor_variables_df = med_df_scaled.drop('Initial_days',axis=1)

predictor_variables_df
```

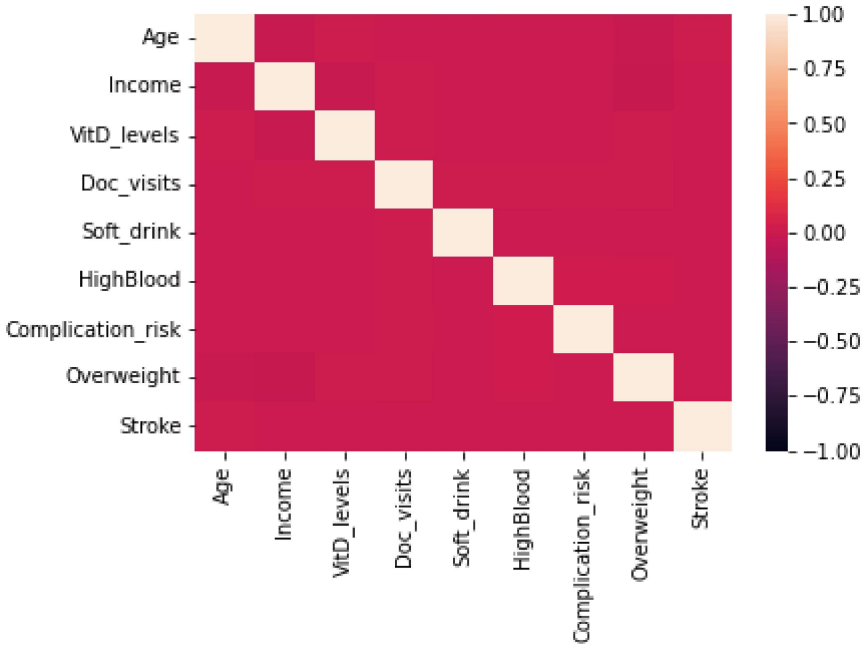
Out[ ]:

	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)
```

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, Initial_da

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

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

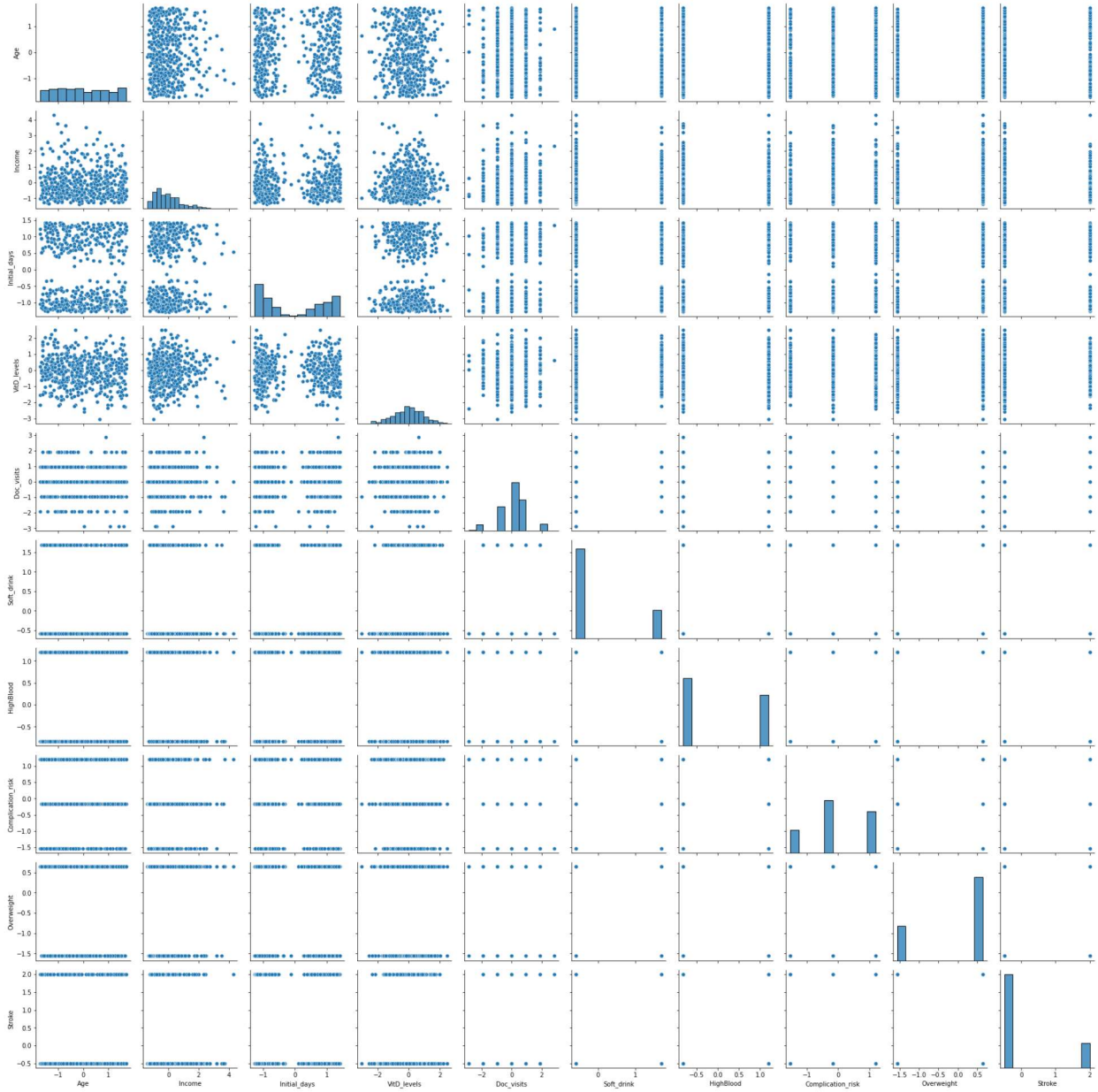
Out[ ]:
```
0      -0.981920
1       0.885049
2      -0.535196
3      -1.230516
4       1.256231
          ...
2495   -1.136224
2496    0.876970
2497    0.917067
2498   -1.232423
2499    0.942555
Name: Initial_days, Length: 2500, dtype: float64
```

C4.

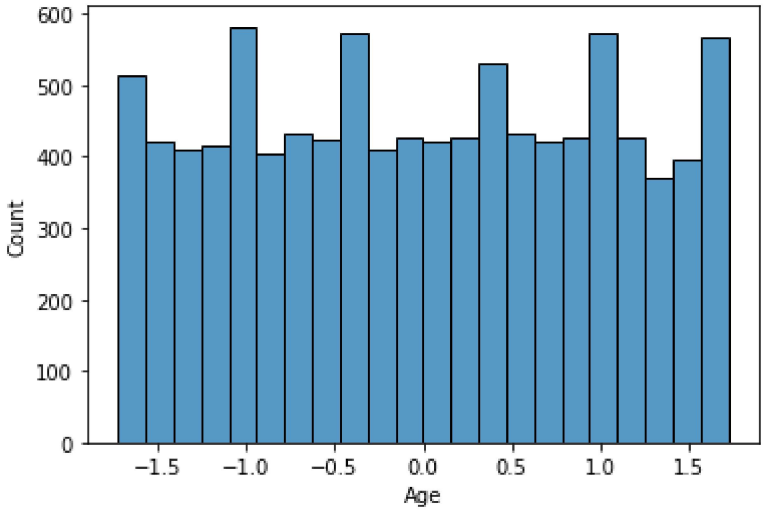
Below - you will find the univariate and bivariate visualizations for the variables that are currently within scope of the model. At an initial glance with the bivariate viz's I am not seeing any strong correlation with out dependent variable. All of these visualizations are derived from a sample from the scaled data set. Again, having used this dataset before - I know that there really isn't a superb relationship between any of our variables (that aren't immediately related i.e. cost and length of stay). (https://seaborn.pydata.org/tutorial/axis_grids.html)

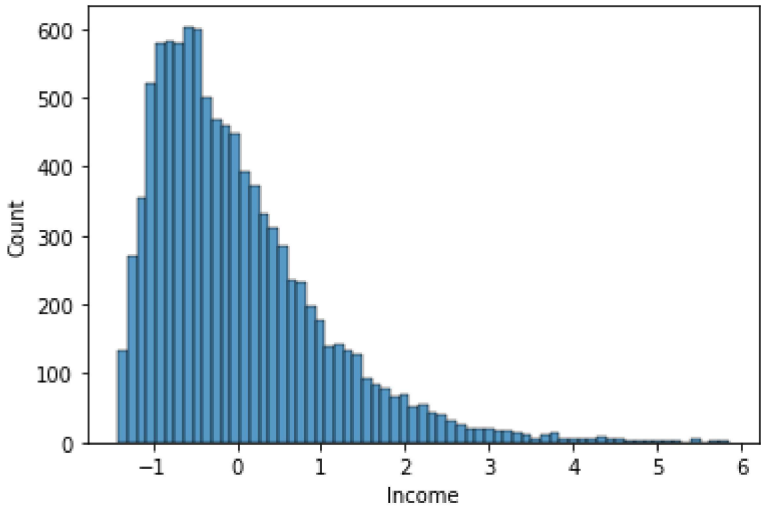
In [ ]:
```
#https://seaborn.pydata.org/tutorial/axis_grids.html
#All Bivariate plots for the cleaned and scaled dataset

g = sns.PairGrid(med_df_scaled.sample(500))
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```

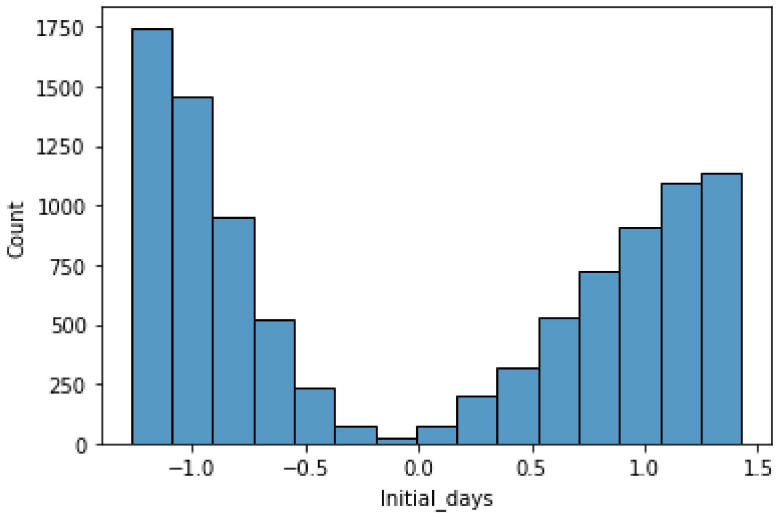
Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x23b3588fb20>
```

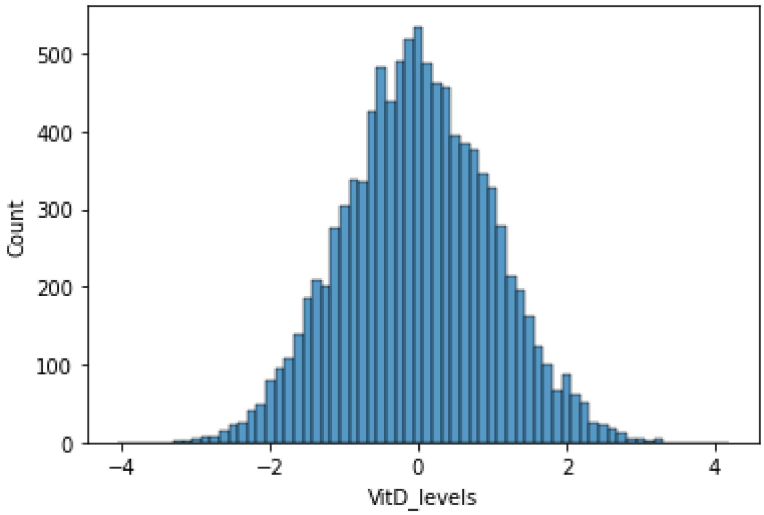


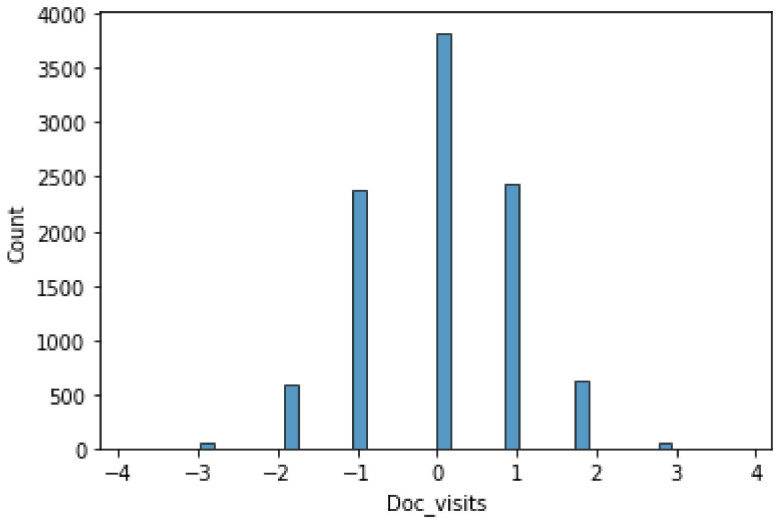
In [ ]:
```
#All univariate plots for cleaned and sclaed dataset

for x in med_df_scaled.columns:
    sns.histplot(x = med_df_scaled[x])
    pyplot.show()
```

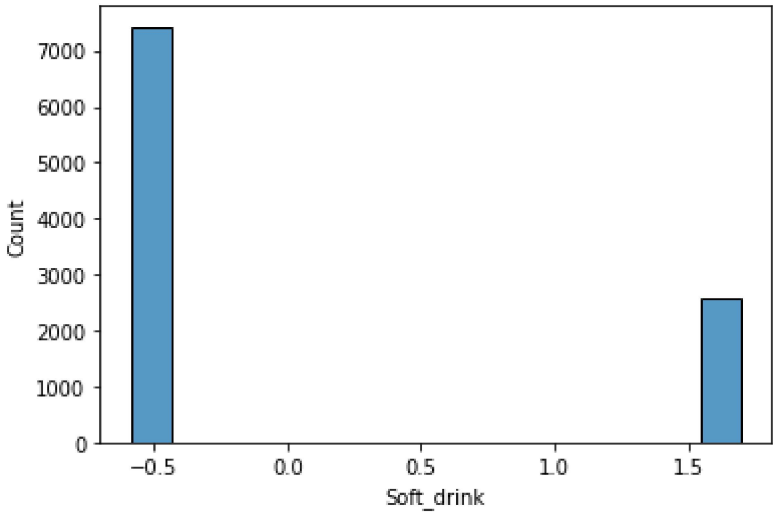

Count
Age

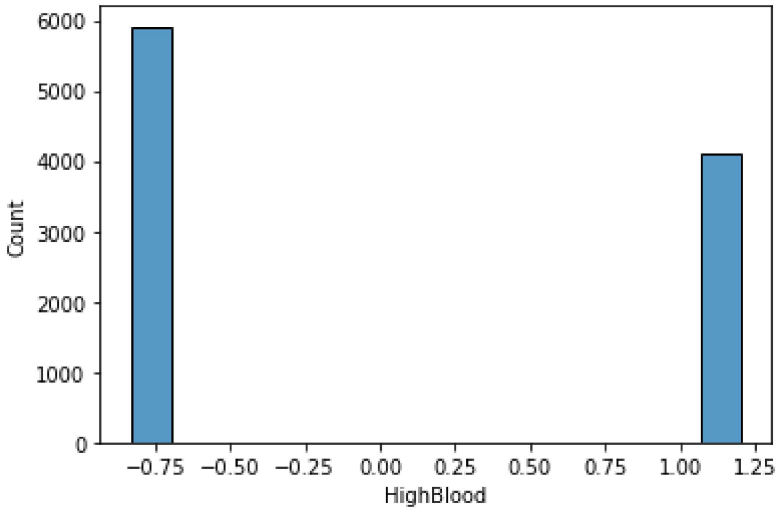

Count
Income

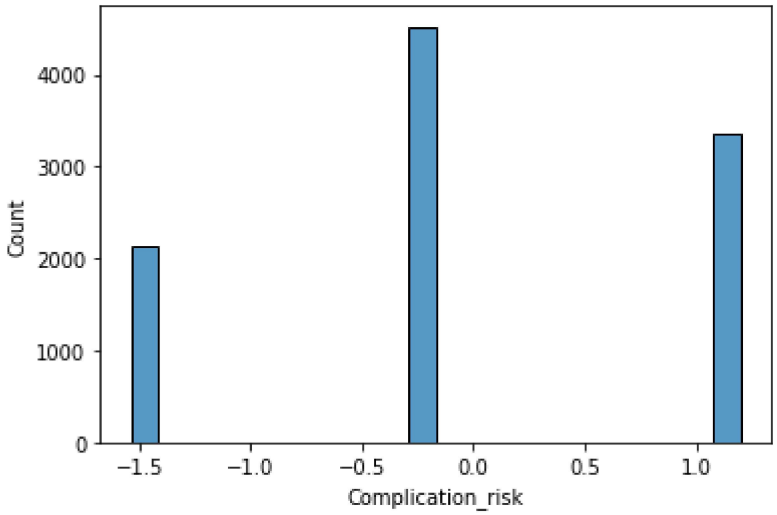

Count
Initial_days

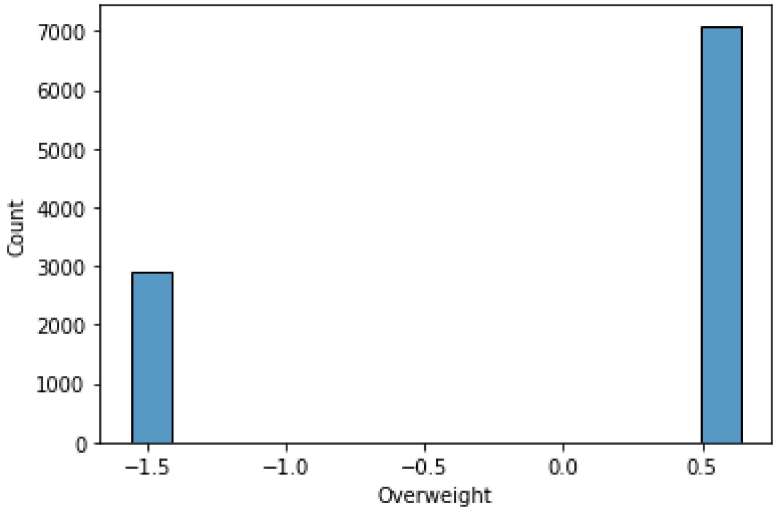

Count
VitD_levels

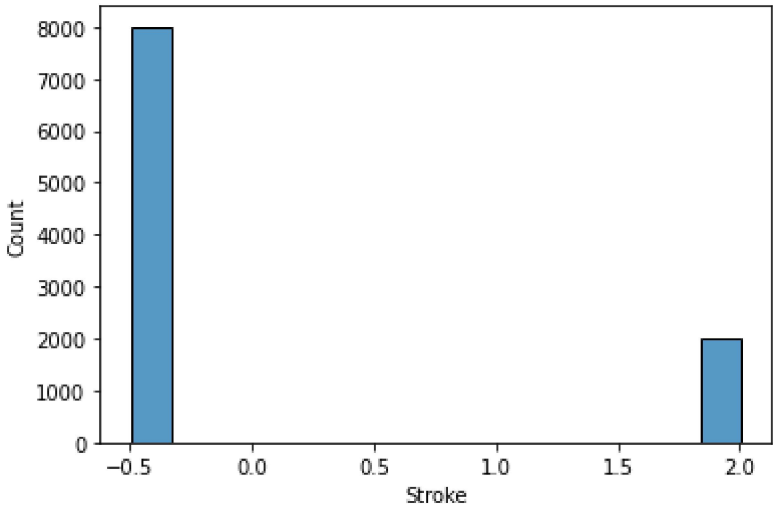

Count
Doc_visits


Count
Soft_drink


6000
5000
4000
3000
2000
1000
0
Count
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
HighBlood


4000
3000
2000
1000
0
Count
−1.5
−1.0
−0.5
0.0
0.5
1.0
Complication_risk


7000
6000
5000
4000
3000
2000
1000
0
Count
−1.5
−1.0
−0.5
0.0
0.5
Overweight



C5.

Below is the output code for the cleaned and prepared dataset with the initial variable set.

In [ ]:
```
#Here is the copy of the cleaned and scaled dataset

med_df_scaled.to_csv('D208 - JacobColp - Cleaned Dataset.csv')
```

D1.

Below you will find the initial multiple regression model. There is 4 steps: the first stepis to instantiate our model. Basically - we are just creating the object that our parameters are going to be placed into. That model object is then fit to our training data. Third - we have our model that we are going to make a prediction against. This is going to use the trained model to actually attempt to predict the outcomes (Initial_Days). The final step is to score our predictions. I have output 3 different measures of the efficacy of our model. The first output is an r^2 score. That is going to explain the level of variance in the dependent variable that can be predicted by the independent variables. The closer to 1 that number is - the better. The mean squared error is going to tell us how far off from a set of points that our regression line is. The lower that number - the better. The root mean squared errors then tells us the spread of the mean squared error. Again, the closer the number is to 0 - the better. (https://www.analyticsvidhya.com/blog/2021/05/multiple-linear-regression-using-python-and-scikit-learn/) (https://akhilendra.com/evaluation-metrics-regression-mae-mse-rmse-rmsle/)

In [ ]:
```
#https://www.analyticsvidhya.com/blog/2021/05/multiple-linear-regression-using-python-

#Instantiate the model
LR = LinearRegression()

#Fit the model
LR.fit(X_Train, Y_Train)

#Make a prediction on test data
y_prediction = LR.predict(X_Test)
```

```
y_prediction
```

Out[ ]:
```
array([-0.023, -0.035,  0.006, ..., -0.043, -0.027,  0.016])
```

In [ ]:
```
#Scoring our model
score= r2_score(Y_Test, y_prediction)

print('r2 score: ',score)

print('mean_sqrd_error: ', mean_squared_error(Y_Test, y_prediction))

print('root_mean_squared_error: ', np.sqrt(mean_squared_error(Y_Test, y_prediction)))
```

```
r2 score:  -0.0018022632362411084
mean_sqrd_error:  1.0077278348125542
root_mean_squared_error:  1.0038564811827209
```

D2.

To start evaluating our first model - lets look through the model evaluation metrics that we utilized above. To start - the r2 score was negative. Not a great start, but also, not the end of the world. The r2 score tells the user how well a combination of variables will explain the variance in a dataset. In this case - our r2 score was negative, which means that the line that our model attempted to fit, accounted for the variance worse than a horizontal line. The other evaluation metrics that we utilized were the mean squared error and root mean squared error. The mean squared error will tell us how wrong we were, and the root mean squared error will tell us how spread out our errors were. Knowing that we were utilizing a scaled dataframe that would max out at no more than 2 or -2, the above errors tell us that we weren't just wrong. We were very wrong. (https://akhilendra.com/evaluation-metrics-regression-mae-mse-rmse-rmsle/) (https://stats.stackexchange.com/questions/12900/when-is-r-squared-negative)

So, lets revisit our variable selection. Perhaps the number of variables actually made our model weaker. To test this we will utilize an sklearn module for feature selection. Within that we will be utilizing the "f_regression" module to score our variables for regression. From there we will select the best 2 or 3 variables (potentially throwing in one if we don't have categorical or continuous), based upon the results from that test. Then we will regenerate our regression model and verify our evaluation metrics - hopefully performing better. By decreasing the number of variables, we are hoping to limit the amount of noise in our model and that will hopefully yield better results. (https://www.datacamp.com/community/tutorials/feature-selection-python)

As a brief note - I have worked with this dataset before and recognize that it's randomly generated. So correlations don't really make sense and are very few and far between. I don't really expect any model to perform well on this data.

In [ ]:
```
#Here we are going to do the feature selection. We will use a simple function from the
#https://www.datacamp.com/community/tutorials/feature-selection-python
#https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-
#'Age', 'Income', 'Initial_days', 'VitD_levels', 'Doc_visits', 'Soft_drink', 'HighBloo
```

```
feature_selection = SelectKBest(score_func=f_regression, k=3)

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

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']

#Here are the three variables that we are going to utilize for our reduced model
print(feature_scores.nlargest(3,'Scores'))

#With these results I would not utilize any of these features. Because none of them ho
#the best performing features out of our selected columns. It is important to note tho


#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_

#Instantiate the model
LR = LinearRegression()

#Fit the model
LR.fit(X_Train, Y_Train)

#Make a prediction on test data
y_prediction = LR.predict(X_Test)

y_prediction

#Scoring our model
score= r2_score(Y_Test, y_prediction)

print('r2 score: ',score)

print('mean_sqrd_error: ', mean_squared_error(Y_Test, y_prediction))

print('root_mean_squared_error: ', np.sqrt(mean_squared_error(Y_Test, y_prediction)))
```

```
       Columns    Scores
2  VitD_levels  0.868013
3   Doc_visits  0.791916
7   Overweight  0.724608
r2 score:  -0.002314959508105119
mean_sqrd_error:  0.9993405938785985
root_mean_squared_error:  0.9996702425693177
```

D3.

Above is the code for a reduced model. You can see that this model was almost equally horrible. It remains worse than a horizontal line at explaining the variance in our data. And, our error values remain abismally high. While there was marignal, and I mean marginal, improvement - even the best 3 features out of our initial list seem to do a poor job in our model. We have utilized VitD_levels (continuous), Doc_visits (continuous), and Overweight (categorical) as the three best features based upon our feature selection (which will be discussed more in the next section).

E1.

The initial intent of the data analysis process as outlined by the project description includes an initial model and reduced model. So, several independent variables were chosen as a start for a multi-linear regression model. The initial selection was based upon their ease of use and prevelance within a clinical setting. As an additional factor - they intuitively seemed connected to the length of a patient's hospital stay. The first model was implemented and yielded less than ideal results. Utilimately created a linear model that fit the data worse than a horizontal line.

The next step in the process was to run through feature selection in an effort to reduce the number of variables and hopefully yield a model that better fit the data. While it did technically improve - the reduced model yielded only slightly better results and ultimately fell short of surpassing that horizontal line.

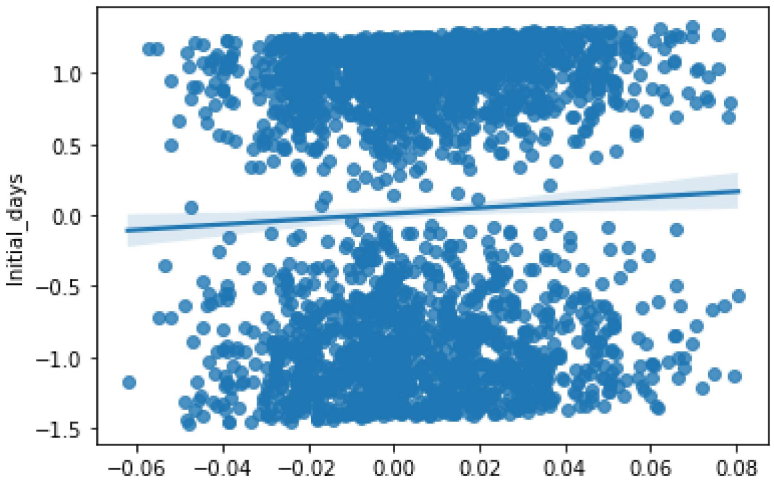
To implement feature selection - I elected to utilize a function of sklearn's feature selection module, called SelectKBest. Effectively, that module will allow you to run various statistical tests, get scoring metrics for several variables, and maintain your best performers. Within that module, I utilized a regression model to score the independent variables. We did not end up identifying any variables within our test set that were able to exceed an alpha value less than .05. In traditional practice, that would eliminate these features from being utilized within a model. Because, the relationship could be happening purely by chance and the odds of that are not exceedingly low. However, in knowing that correlation is almost nonexisitant within the dataset (and certainly not in our chosen variables), we proceeded anyway. I chose to utilize a regression model for scoring since that was the ultimate goal of our output, and we were utilizing a mix of continuous and categorical variables. The regression model was more robust for this versus something like a Chi2 test of independence. (https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e)

We utilized the same scoring metrics for both the initial model and our reduced model. To restate from previous sections, the r2 score was used to identify how well our model fit the dataset. The squared error metrics were utilized to tell us how far off we were. To summarize, our line fit poorly. The predictions were very far off (for our scaled data - 1 is a large difference) and our predictions were very spread out. (https://akhilendra.com/evaluation-metrics-regression-mae-mse-rmse-rmsle/)

Below you will find the residual plot. Ideally, you would see a lot of values at zero - meaning that there was no difference between the prediction and the actual value. However, that is not the case. Barring a few examples that get close - most of the values are very off in the positive or negative direction. (https://www.qualtrics.com/support/stats-iq/analyses/regression-guides/interpreting-residual-plots-improve-regression/)

In [ ]:
```
#Here is the residual plot

residuals = y_prediction - Y_Test

sns.regplot(y= residuals, x= y_prediction)
```

Out[ ]:
```
<AxesSubplot:ylabel='Initial_days'>
```



E2.

Below I will reiterate and print out results from data analysis process.

In [ ]:
```
#Residual Error
print(residuals)
```

```
1646    1.264682
6391   -1.202094
5547   -0.635154
2751    1.162001
7586   -0.498930
          ...
5968   -1.082361
8149   -0.714601
5506   -0.435770
8863   -0.839848
4431    0.982942
Name: Initial_days, Length: 2500, dtype: float64
```

In [ ]:
```
#Predicted Values
print(y_prediction)
```

```
[ 0.008  0.    -0.049 ... -0.01   0.01   0.02 ]
```

In [ ]:
```
#r2 value, mean squared error and root mean squared error for our reduced model
print('r2 score: ',score)
```

```
print('mean_sqrd_error: ', mean_squared_error(Y_Test, y_prediction))

print('root_mean_squared_error: ', np.sqrt(mean_squared_error(Y_Test, y_prediction)))
```

```
r2 score:  -0.002314959508105119
mean_sqrd_error:  0.9993405938785985
root_mean_squared_error:  0.9996702425693177
```

E3.

I utilized a couple resources to implement my regression model. Below will be a list:

- https://www.analyticsvidhya.com/blog/2021/05/multiple-linear-regression-using-python-and-scikit-learn/
- https://akhilendra.com/evaluation-metrics-regression-mae-mse-rmse-rmsle/
- https://stats.stackexchange.com/questions/12900/when-is-r-squared-negative
- https://www.datacamp.com/community/tutorials/feature-selection-python
- https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e

All of these have code snippets within them and the sections in which they are utilized have been annotated with their links in comments. However, I will briefly touch on what was gleaned from each. The first resource from Analytics Vidhya provided me with a general implementation of a multi-linear regression model in Python. It was utilized for the initial model creation. The next resource from Akhilendra was utilized for an overview of different evaluation metrics that can be utilized for regression models. There is no code within this article, but it helped me to understand why certain metrics were utilized in other resources. The stackexchange link simply helped me to understand the meaning of a negative r2 value. I had never run across that before and needed a simple explanation. Datacamp was utilized to give me a high level overview of different feature selection techniques that could be utilized. It specifically informed my utilization of sklearn's selectkbest. I wanted to reaffirm information that briefly touched on from Datacamp and so I utilized the Towards Data Science article to reiterate and clarify information.

F1.

In summary, the results of my data analysis were lackluster at best. Both models that were generated yielded very inaccurate predictions.

Based upon the coefficients and intercept from the below output - the regression equation would be: y = -0.013x + 0.0032 + -0.008x + 0.0032 + -0.008x + 0.0032

The coefficients are implying that all of our variables have a very small, and negative, impact on our depedendent variable. The coefficients as well as the intercept have been printed out below. (https://dss.princeton.edu/online_help/analysis/interpreting_regression.htm)

To reiterate what has been stated multiple times throughout this process, our models are inaccurate from a statistical perspective and not usable from a practical perspective. Simply referencing the r2 score is enough to indicate that this model should not be utilized. Put that in

conjunction with the residuals and it becomes quite evident just how poorly this model has performed.

The limitations of the data analysis really start and stop with the data set. It is extremely difficult to implement a model on a data set where correlation is almost nonexistant and isn't rational otherwise. That said - there are some general limitations that exist with linear regression, multiple or otherwise. First, it does present as a blunt tool to solve, what can be, complex issues. In field specific terminology, it is prone to underfitting - effectively meaning that is does not account for enough variance. Additionally, since it is making broad assumptions, it can be sensitive to outliers. (https://iq.opengenus.org/advantages-and-disadvantages-of-linear-regression/)

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

```
[-0.005 -0.019 -0.014]
0.005933312774572002
```

F2.

My recommendation based upon my evaluation of the data would be to regenerate the data set. Making the assumption that we can trust the integrity of the data - it was extremely apparent early on that this data set was not going to yield good results for our request of a regression model. The relationship amongst data appears random at best and in a nonrational way. If we could generate a more reliable data set - we could retry the regression model to attempt to find better results. Outside of regenerating the data set - I would recommend that we try different models. Perhaps a more generalized classification model would work better. Within the same vein as the original research question - could we identify short term and long term patients? We wouldn't be trying to make much broader assumptions that may be easier with our data set. I would also strongly recommend that this model not be used in any way. It would not deliver clinical value and may hinder patient experience more than anything.

Citations

"Advantages and Disadvantages of Linear Regression." OpenGenus IQ: Computing Expertise & Legacy, 27 July 2020, iq.opengenus.org/advantages-and-disadvantages-of-linear-regression/.

"Interpreting Regression Output." Data and Statistical Services, dss.princeton.edu/online_help/analysis/interpreting_regression.htm.

Shaikh, Rahil. "Feature Selection Techniques in Machine Learning with Python." Medium, 28 Oct. 2018, towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e.

Paul, Sayak. "Python Feature Selection Tutorial : A Beginner's Guide." Datacamp, www.datacamp.com/community/tutorials/feature-selection-python.

"When is R Squared Negative?" Cross Validated, stats.stackexchange.com/questions/12900/when-is-r-squared-negative.

"Evaluation Metrics for Regression Models- MAE Vs MSE Vs RMSE Vs RMSLE." Machine Learning — Make Better Career Decisions, akhilendra.com/evaluation-metrics-regression-mae-mse-rmse-rmsle/.

"Multiple Linear Regression Using Python and Scikit-learn." Analytics Vidhya, 1 May 2021, www.analyticsvidhya.com/blog/2021/05/multiple-linear-regression-using-python-and-scikit-learn/. Accessed 12 Apr. 2022.

"Interpreting Residual Plots to Improve Your Regression." Qualtrics XM // The Leading Experience Management Software, 12 Apr. 2021, www.qualtrics.com/support/stats-iq/analyses/regression-guides/interpreting-residual-plots-improve-regression/.

"Building Structured Multi-plot Grids — Seaborn 0.11.2 Documentation." Seaborn: Statistical Data Visualization — Seaborn 0.11.2 Documentation, seaborn.pydata.org/tutorial/axis_grids.html.

Python, Real. "Split Your Dataset With Scikit-learn's Train_test_split() – Real Python." Python Tutorials – Real Python, 23 Nov. 2020, realpython.com/train-test-split-python-data/. Accessed 12 Apr. 2022.

"Sklearn.preprocessing.StandardScaler." Scikit-learn, scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. Accessed 12 Apr. 2022.

"From Data Pre-processing to Optimizing a Regression Model Performance." KDnuggets, www.kdnuggets.com/2019/07/data-pre-processing-optimizing-regression-model-performance.html.

Wijaya, Cornellius Y. "3 Top Python Packages to Learn Statistic for Data Scientist." Medium, 12 Oct. 2021, towardsdatascience.com/3-top-python-packages-to-learn-statistic-for-data-scientist-d753b76e6099.

"Multiple Linear Regression | A Quick and Simple Guide." Scribbr, 26 Oct. 2020, www.scribbr.com/statistics/multiple-linear-regression/.

"Homoscedasticity? Don't Be a Victim of Statistical Hippopotomonstrosesquipedaliophobia." blog.minitab.com/en/statistics-and-quality-data-analysis/dont-be-a-victim-of-statistical-hippopotomonstrosesquipedaliophobia.