A Predictive Analysis of Diabetes in the United States Based on Data Mining Classification Methods

Michael Nguyen, Oscar Gil, and Anusia Edward

Shiley-Marcos School of Engineering, University of San Diego

Abstract

The purpose of this study is to predict whether an individual is suffering from diabetes within the United States. For this study, it was hypothesized that at least one of the four classification techniques of K-Nearest Neighbor (KNN), Naive Bayes, Random Forest, and Logistic Regression will result in a model that predicts an individual's health status of having diabetes or not. More specifically, a secondary hypothesis is that the best predictor will be the Naive Bayes, based on the accuracy being greater than seventy-five percent. The dataset was obtained from Kaggle. The predictor variables observed in this study include: the number of adults within a household, general health, physical health, mental health, height, weight, sex, smoker, asthmatic, orange vegetable intake, geen vegetable intake, fruit intake, fruit juice intake. Additionally, health insurance, blood pressure medication, heart disease, high cholesterol, or high blood pressure was studied. The study determined that the best model was logistic regression, which had an accuracy of 89%. Further research regarding the precision and sensitivity of the model could be evaluated using additional predictors such as genetic history, presence of ketones in urine, or blurred vision.

Keywords: Keywords: Diabetes, K-Nearest Neighbor (KNN), Naive Bayes, Random Forest, and Logistic Regression

Table of Contents

Introduction

Background and Practical Implications

Diabetes was declared a major public health problem by the Centers for Disease Control and Prevention (CDC) in 1994 (World Health Organization, 2021). Since then, the prevalence of diabetes has steadily increased around the world. Diabetes is a chronic disease that is caused by the body's inability to utilize or produce insulin. Insulin is a hormone that works by regulating the body's blood sugar levels by helping the cells absorb sugar and glucose from the bloodstream (Forouhi et al., 2010). Without insulin, the body enters a state of hyperglycaemia, which if left untreated can lead to many health risks and complications. Such health complications may include glaucoma, cataracts, kidney disease, hypertension, diabetic ketoacidosis, skin complications, and damaged nerves and blood vessels (Forouhi et al., 2010). If an individual's diabetes is left untreated, it can also lead to heart attacks, strokes, or limb amputations in extreme cases (Forouhi et al., 2010). The International Diabetes Federation (IDF) cites diabetes as the cause of approxmately 6.7 million deaths in 2021 alone (Tonnies et al., 2021). Furthermore, the IDF expects the prevalence of diabetes to increase to 783 million people by the year 2045 (Tonnies et al., 2021).

Purpose, Objective, and Hypothesis of Present Study

Due to the expected increase in the prevalence of diabetes, the purpose of this study is to address this public health concern by creating a model to precisely diagnose individuals with diabetes. This is important as it will allow for more individuals to seek help if diagnosed with diabetes. The machine learning objective is to create and tune four classification models to predict an individual's diagnosis for diabetes. The following four classification models were employed to carry out the objective for this study: K-Nearest Neighbor (KNN), Naive Bayes, Random Forest, and Logistic Regression. The general hypothesis of this study is that at least one of the four classification techniques utilized will result in a model that predicts an individual's diagnosis of diabetes based on a subset of risk factors. More specifically, a secondary hypothesis is that the best model will be the Naive Bayes with an accuracy score above 0.75.

Methods

Data Collection, Wrangling, Preprocessing, and Splitting

The data for this study was sourced from Kaggle as a raw csv file. It did not have prior preprocessing, and it was initially collected through a health-related telephone survey conducted by the Behavioral Risk Factor Surveillance System for U.S. residents. The initial dataset had 330 features and 441,456 records regarding health risk factors of comorbidities. The data consisted of ordinal, binary, discrete, and continuous variables. No text-based data were present within the dataset. There were many missing values, and the data was skewed and contained outliers. Based on these initial assessments of the data, the first step in preprocessing the data was dimensionality reduction. Features that had a threshold value of 85% of missing values were removed, resulting in a total of 216 features.

Next, features containing redundant or irrelevant information were removed. For example, the ‘SEQNO’ and ‘_PSU’ columns both indicated the patient’s ID, or ‘CTELENUM’ and ‘PVTRESD1’ both indicated whether or not the survey was taken using a landline. These columns that contained the same information were removed. Additionally, features that were irrelevant to diabetes were removed such as the ‘CELLFON3’ feature, which indicated whether or not the study was conducted via a cell phone. After the removal of the irrelevant and redundant features, nineteen independent variables and the dependent variable were selected for the predictive analysis. Previous research was cross-referenced to ensure that the following independent variables are important risk factors for diabetes: 'HHADULT', 'GENHLTH', 'PHYSHLTH', 'MENTHLTH','HLTHPLN1', 'BPMEDS','RFCHOL', 'HTM4', 'WTKG3', 'ORNGDAY','GRENDAY_', 'FRUTDA1_', 'FTJUDA1_', '_SMOKER3','ASTHMA3', '_MICHD', 'SEX', and 'BPHIGH4'. The dependent variable for this study is ‘DIABETE3’, which is a binary variable indicating whether or not a person has diabetes. Each of the binary variables were recorded as 0 and 1, as initially they were 1 and 2. After, all of the categorical and numerical variable’s unknown values were recorded as “99999”. This was carried out to ensure the missing and unknown values could be easily handled. The “99999” values were then recorded as NaN. These missing values were then handled using sklearn.impute’s ‘SimpleImputer’ function.

Then, the numeric variables of 'HHADULT', 'GENHLTH', 'PHYSHLTH', 'MENTHLTH', 'HTM4', 'WTKG3', 'ORNGDAY_','GRENDAY_', 'FRUTDA1_', 'FTJUDA1_' were analyzed for outliers using boxplots and skewness using histograms (Appendix A). The outliers were handled by creating an InterQuartile Range (IQR) equation and removing the specified values. Based on the histograms produced, the following variables had a right-skewed distribution: ‘HHADULT’, ‘WTKG3’, ‘ORNGDAY_’, ‘GRENDAY_’, ‘FRUTDA1’, and ‘FTJUDA1’. The variables ‘PHYSHLTH’, and ‘MENTHLTH’ had a left-skewed distribution, while ‘HTM4’ had a normal distribution. The skewness of the numeric variables were handled through the use of sklearn.preprocessing’s ‘StandardScalar’ function. Finally, the data was split using a random stratified split of 80% training set and 20% testing set using sk.learn’s ‘train_test_split’ and specifying ‘stratify = y’. The random stratified split was completed to ensure that the proportions of the target variable, diabetes, was proportional within the sets. The final analytics base table contained a total of 252,171 instances.

Sample Characteristics

The population addressed in the hypothesis of this study is in reference to all individuals within various countries around the world. For the purpose of this study however, the sample population being observed is individuals within the United States. The sample features for this statistical study include the following: ‘HHADULT’, ‘GENHLTH’, ‘PHYSHLTH’, ‘MENTHLTH’, ‘HLTHPLN1’, ‘BPMEDS’, ‘_RFCHOL’, ‘HTM4’, ‘WTKG3’, ‘ORNGDAY_’, ‘GRENDAY_’, ‘FRUTDA1_’, ‘FTJUDA1_’, ‘_SMOKER3’, ‘ASTHMA3’, ‘_MICHD’, ‘SEX’, ‘BPHIGH4’, and ‘DIABETE3’. ‘HHADULT’ indicates the number of adults within the household and is a discrete numerical variable. ‘GENHLTH’ is a binary variable in which an individual indicates if they feel good about their health (0) or poorly (1). ‘PHYSHLTH’ indicates the number of days a person felt their poor health led to the inhibition of daily activities. This variable was recorded as a discrete numerical variable. ‘MENTHLTH’ is the number of days a person felt stressed or depressed. This variable was recorded as a discrete numerical variable. ‘HLTHPLN1’ is a binary variable which indicates whether a person has health care coverage (0), or not (1). ‘BPMEDS’ is a binary variable that indicates whether a person is taking medicine for high blood pressure (0), or is not (1). ‘_RFCHOL’ is a binary variable that indicates whether an adult has high cholesterol (1) or normal cholesterol (0). ‘HTM4’ is a continuous numerical variable which indicates height (cm). ‘WTKG3’ is a continuous numerical variable that indicates weight (kg). ‘ORNGDAY_’ is a discrete numerical variable which indicates orange vegetable intake per day. ‘GRENDAY_’ is a discrete numerical variable which indicates green vegetable intake per day. ‘FRUTDA1_’ is a discrete numerical variable which indicates fruit intake per day. ‘FTJUDA1_’ is a discrete numerical variable which indicates fruit juice intake per day. The ‘_SMOKER3’ variable is a binary variable which indicates whether a person is a smoker (1) or not (0). ‘ASTHMA3’ is a binary variable which indicates whether a person has asthma (1) or not

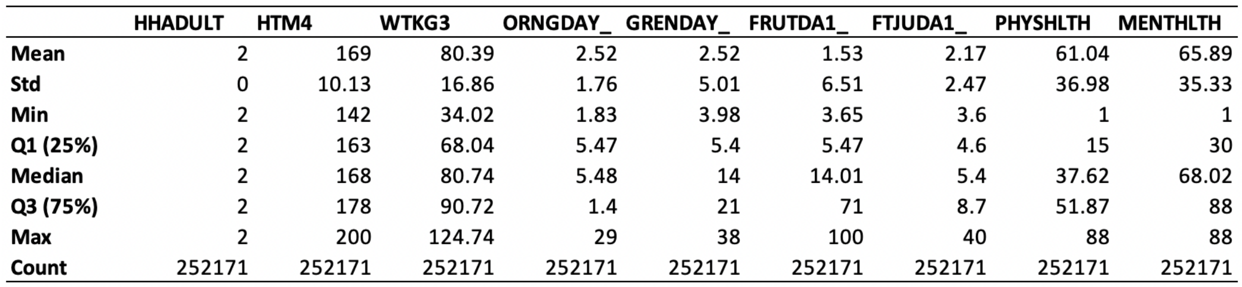
(0). The '_MICHD' variable is a binary variable which indicates whether a person has coronary heart disease and myocardial infarction (1) or not (0). The 'SEX' variable is a binary variable which indicates male (1) or female (0). 'BPHIGH4' is a binary variable which indicates high blood pressure (1) or regular blood pressure (0). Finally, the 'DIABETE3' variable is a binary variable which indicates whether a person has diabetes (1) or not (0).

Exploratory Data Analysis

In addition to preprocessing and wrangling the data, exploratory data analysis was also performed on the dataset. Exploratory data analysis was conducted in order to gain insight of patterns and possible relationships within the dataset. From the nineteen independent variables, nine were identified as quantitative variables. The nine variables are as follows: 'HHADULT', 'HTM4', 'WTKG3', 'ORNGDAY_', 'GRENDAY_', 'FRUTDA1_', 'FTJUDA1_', 'PHYSHLTH', and 'MENTHLTH'. The descriptive statistics for each of the quantitative variables can be seen in Table 1. In Table 1, the mean, standard deviation, minimum, first

Table 1

Descriptive Statistics of the Quantitative Variables



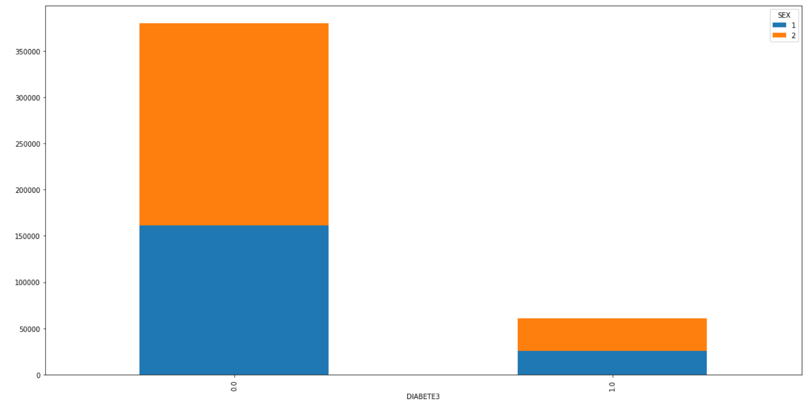
	HHADULT	HTM4	WTKG3	ORNGDAY_	GRENDAY_	FRUTDA1_	FTJUDA1_	PHYSHLTH	MENTHLTH
Mean	2	169	80.39	2.52	2.52	1.53	2.17	61.04	65.89
Std	0	10.13	16.86	1.76	5.01	6.51	2.47	36.98	35.33
Min	2	142	34.02	1.83	3.98	3.65	3.6	1	1
Q1 (25%)	2	163	68.04	5.47	5.4	5.47	4.6	15	30
Median	2	168	80.74	5.48	14	14.01	5.4	37.62	68.02
Q3 (75%)	2	178	90.72	1.4	21	71	8.7	51.87	88
Max	2	200	124.74	29	38	100	40	88	88
Count	252171	252171	252171	252171	252171	252171	252171	252171	252171

quartile value, median, third quartile value, maximum, and total count were included. From the table, it can be seen that the average number of adults was two people. The average height and weight of the participants was 169 cm and 80.39 kg respectively. Additionally, on average, an individual consumed orange and green vegetables approximately three times a day while eating fruits on average once a day. It should also be noted that on average, participants' physical and mental health were poor, as out of a six month period it was reported on average that their health was poor for 61 and 66 days respectively.

Furthermore, when exploring the data, a bar chart of the frequency of diabetic versus non-diabetic patients with sex as an overaly was visualized (Figure 1). When looking at Figure 1,

Figure 1

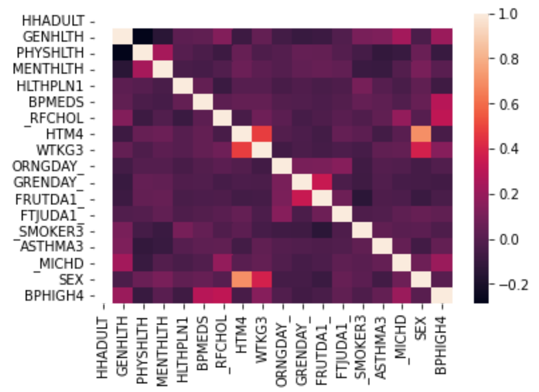
Bar Chart of Diabetes with Sex as an Overlay



it is evident that for diabetic patients (1), the majority of diabetic patients are females (2). This is an interesting observation as diabetes is more common in males than females. In fact, current research suggests that men are twice as likely to develop type 2 diabetes than women (Meissner, 2021). This discrepancy, however, may be the case as more women within the U.S. are likely to suffer from gestational diabetes, which was labeled patients in this dataset as diabetic even if patients recovered from it post pregnancy, possibly leading to the greater number of females classified as diabetic (Deputy et al., 2018). Correspondingly, the relationship between each of the variables was visualized through a correlation heat map (Figure 2). From this heat map it should be noted that the variables ‘HTM4’ and ‘SEX’ are highly correlated with a correlation coefficeint of approximately 0.8.

Figure 2.

Correlation Heat Map of all the Features Within the Dataset.



Modeling

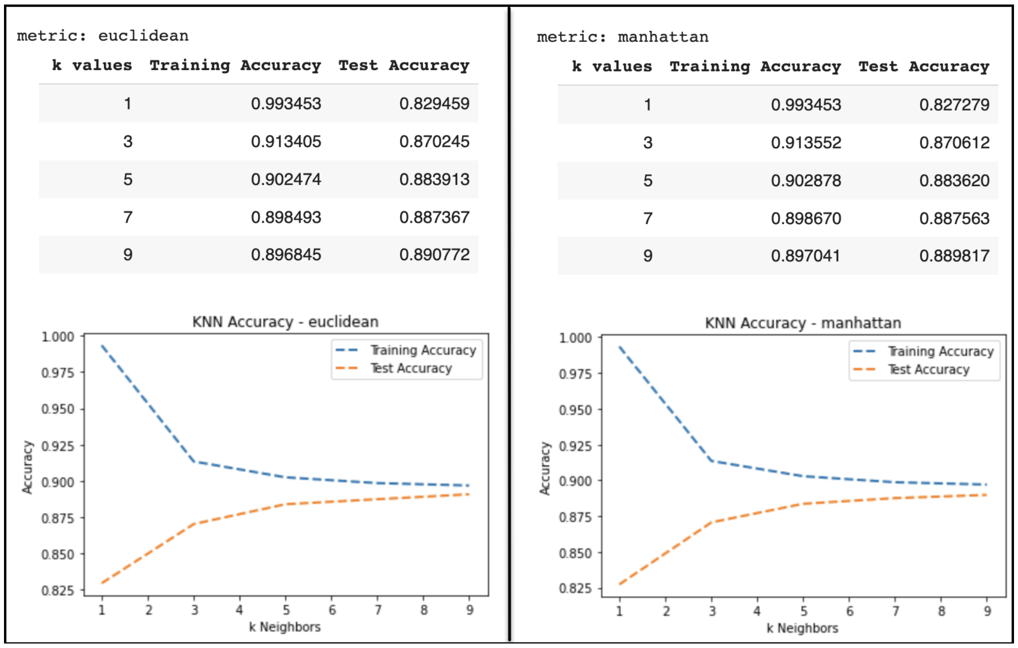
Model 1: KNN

KNN was the first model processed. A "for loop" for k values of 1, 3, 5, 7, and 9 were run for Euclidean and Manhattan metrics to identify the best performance. Odd k values were selected to prevent ties when interpreting cluster results. "The odd value of K should be preferred over even values in order to ensure that there are no ties in the voting" (Goyal, 2021).

Figure 3 compares Euclidean and Manhattan accuracy and plots for training and test data set results. The metrics produced identical results in the table and the plots, which can be challenging to identify as differences are in the ten thousandth place. The k value of 3 with a test accuracy of 0.870612, using a Manhattan metric, performed the best. The k value of 5 had a better accuracy but was not selected because the accuracy range between its training and test result was too close; the k value of 3 results had the highest range, which is best suited to prevent overfitting. The confusion matrix results had a false negative count of 1,411 out of 40,823 predictions, meaning that 3% of the total predictions in this model are false negative, which are people with diabetes not identified.

Figure 3

For loop results and plot comparing Euclidean and Manhattan accuracies



```
metric: euclidean
   k values   Training Accuracy   Test Accuracy
          1            0.993453        0.829459
          3            0.913405        0.870245
          5            0.902474        0.883913
          7            0.898493        0.887367
          9            0.896845        0.890772
```

```
metric: manhattan
   k values   Training Accuracy   Test Accuracy
          1            0.993453        0.827279
          3            0.913552        0.870612
          5            0.902878        0.883620
          7            0.898670        0.887563
          9            0.897041        0.889817
```

Model 2: Naive Bayes

Naive Bayes was the second model processed as we tried to identify the model with the best accuracy. “Naive Bayes has higher accuracy and speed when we have large data points” (Sharma, 2021). Therefore, Naive Bayes was a perfect fit for the extensive data set in our project.

The Naive Bayes model used was Bernoulli since our target variable is binary, 1s and 0s. Gaussian and Multinomial Naive Bayes model types were inappropriate for our target data type. An attempt using Multinomial caused an error since our target variable is 1s and 0s, and although Gaussian processed, its results were not trustworthy as the target is not a continuous data type. The Bernoulli model had an accuracy of 0.88834. The confusion matrix results had a false negative count of 513 out of 40,823 predictions, meaning that 1% of the total predictions in this model are false negative. This is a 2% improvement in this area over the KNN model.

Model 3: Random Forest

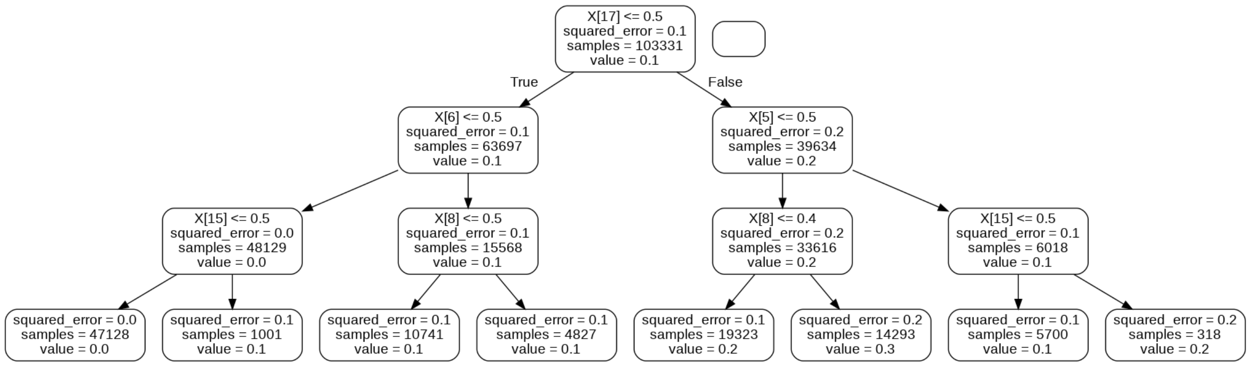
The third model that was used is the random forest model. The random forest model consists of using decision trees, a large number of individual decision trees that are intertwined together to make a “forest”.

The first step to running the random forest model is assigning the randomforestregressor to a variable which is next used as a fit with x_train and y_train to the randomforestregressor. Then prediction is attained through rf.predict on x_test. The prediction is used to get the absolute error. The result is a mean absolute error of 0.19 degrees. The mean absolute error shows the absolute difference between model prediction and target value but doesn’t consider the direction. A low mean absolute error value is good and shows that on average, the forecast’s distance from the true value is 0.19 degrees. The mean absolute percentage error is then attained using the absolute error which is then in turn used to get the accuracy. The accuracy that came from this was “-inf %”. Since the model was not able to produce a coherent accuracy and that it is negative as well, it shows that this model, the random forest, does not work well for this data set but continued processing of this model is still done in order to get the trees.

The first decision tree that was created was too big so it was too hard to visualize. The original tree size is hundreds in layers deep while thousands of layers in width which makes the decision tree seem like an actual “forest” and making it difficult to map out as well. In order to make it easier to visualize, a max depth of 3 was applied and it resulted in Figure 4.

Figure 4

Max Depth 3 Decision Tree



Each of the bottom of the tree is a leaf node. The first number on each of the nodes is the variable and value to split the node on. The second number is the mean squared error of the node. The third number is the number of data points in that node.The last number is the prediction in degrees for all data points in that node. The numbers that are in the prediction section are all 0.1, 0.2 and 0 which is very misleading. It shows how this model does not work well with this data set at all.

Model 4: Logistic Regression

The last model that was used for the dataset was the logistic regression. Logistic regression is a machine learning algorithm that is used with classification issues and is used to predict probabilities.

The first step that was taken to running the logistic regression model is to assign the logisticregression function to a variable. Then the x_train and y_train is fit to the logistic regression. The x_test is also used in logreg.predict in order to get the y prediction. From using the y_test and the y prediction, the confusion matrix of the classification matrix and the classification report can be made. The classification report contains the precision, recall, f1-score, support, accuracy, macro average and weighted average as seen below in table 2.

Table 2

Logistic Regression Classification Report

	precision	recall	f1-score	support
0	0.5	1	0.94	36504
1	0.54	0.02	0.03	4319
accuracy			0.89	40823
macro avg	0.72	0.51	0.49	40823
weighted avg	0.86	0.89	0.85	40823

The model has an accuracy of .89 or 89% which is fairly high showing that this model has good performance. The model's confusion matrix reports a false negative count of 57 out of 40823. This is very low and indicates a very good model since false negative is the worst result that can be obtained. Afterwards, the ROC plot is made as shown below in Figure 5.

Figure 5

ROC Plot

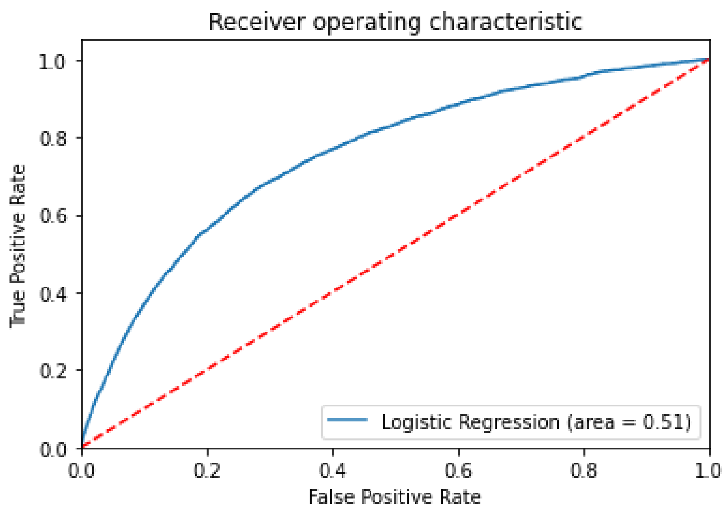


Figure 5 shows the dotted red line as the standard ROC curve of a purely random classifier. The classifier that is used is the blue curve which is the logistic regression. The plot shows that the classifier curve is a good distance away from the ROC curve which means that the logistic regression classifier is a good classifier in this case.

Results

KNN, Naive Bayes, and Logistic Regression models produced results with very similar accuracy. The accuracy was so close that each model's false negative count and percentage helped determine the winner. The best model was Logistic Regression, which had the highest accuracy at 89% and the lowest false negative count at 57, as seen in Table 3.

Table 3

Model accuracy and false negative comparisons

Model Name	Accuracy %	False Negative Count	False Negative Percent
KNN	87.06%	1411	0.03
Naïve Bayes	88.83%	513	0.01
Logistic Regression	89.00%	57	0.0014

Discussion

The predictor variables consider an individual's household composition regarding how many adults reside there, general physical and mental health, height and weight, sex, smoker, asthmatic, and particular fruit and vegetable intake. Based on the models' results, these predictor variables can help explain with 87 to 89 percent accuracy whether a person has diabetes. In addition, the Logistic Regression model produced results at 89 percent accuracy with the lowest false negative prediction totals of all candidate models at 57. The class imbalance ratio for the target variable in the complete data set is present through stratification in the train and test data sets. Given this knowledge, it is vital to consider a potential bias in the false negative results. Ongoing monitoring of false negatives would be ethical to prevent misdiagnosing a person who truly has diabetes. Running all three models in production until concerns for bias are no longer a problem can be a solution.

References

Deputy, N. P., Kim, S. Y., Conrey, E. J., & Bullard, K. M. (2018). *Prevalence and changes in preexisting diabetes and gestational diabetes among women who had a live birth—United States*, 2012–2016. Morbidity and Mortality Weekly Report. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6319799/

Forouhi, N. G., & Wareham, N. J. (2010). *Epidemiology of diabetes.* Medicine, 38(11), 602-606.

Meissner, M., PhD. (2021, March 2). *How diabetes affects men vs. women.* MedicalNewsToday. https://www.medicalnewstoday.com/articles/diabetes-affects-men-women#prevalence

Tönnies, T., Rathmann, W., Hoyer, A., Brinks, R., & Kuss, O. (2021). *Quantifying the underestimation of projected global diabetes prevalence by the International Diabetes Federation (IDF) Diabetes Atlas.* BMJ Open Diabetes Research and Care, 9(1), e002122.

World Health Organization. (2021, November 10). *Diabetes*. https://www.who.int/news-room/fact-sheets/detail/diabetes#:%7E:text=The%20number%20of%20people%20with,stroke%20and%20lower%20limb%20amputation.

Goyal, C. (2021, June 9). *20 Questions to Test your Skills on KNN Algorithm*. https://www.analyticsvidhya.com/blog/2021/05/20-questions-to-test-your-skills-on-k-nearest-neighbour/

Sharma, A. (2021). *Naive Bayes with Hyperpameter Tuning*. https://www.kaggle.com/code/akshaysharma001/naive-bayes-with-hyperpameter-tuning

504_FinalProject_Group4

August 10, 2022

1 Appendix A

Predictive Analysis of Diabetes

```
[ ]: # Necessary Imports
     import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split
     from sklearn.impute import SimpleImputer
     import seaborn as sns
     import matplotlib.pyplot as plt
     from sklearn.preprocessing import StandardScaler
     from sklearn.ensemble import RandomForestRegressor
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.naive_bayes import BernoulliNB, GaussianNB
     from sklearn.metrics import confusion_matrix, accuracy_score,␣
      ↪classification_report
```

Preprocessing & EDA:

```
[ ]: from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ]: # Loading in the dataset
     #diabetes_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/2015.csv')
```

```
[ ]: # Loading in the dataset from shared drive
     diabetes_df = pd.read_csv('/content/drive/Shared drives/ADS504/2015.csv',
                               engine='python')
```

```
[ ]: # Visualizing the data
     diabetes_df.head(2)
```

```
[ ]:    _STATE  FMONTH        IDATE IMONTH   IDAY    IYEAR  DISPCODE       SEQNO  \
     0       1       1  b'01292015'  b'01'  b'29'  b'2015'      1200  2015000001
     1       1       1  b'01202015'  b'01'  b'20'  b'2015'      1100  2015000002
```

```
         _PSU  CTELENUM  …  _PAREC1  _PASTAE1  _LMTACT1  _LMTWRK1  _LMTSCL1  \
0  2015000001       1.0  …        4         2       1.0       1.0       1.0
1  2015000002       1.0  …        2         2       3.0       3.0       4.0

   _RFSEAT2  _RFSEAT3  _FLSHOT6  _PNEUMO2  _AIDTST3
0         1         1       NaN       NaN       1.0
1         2         2       NaN       NaN       2.0

[2 rows x 330 columns]
```

[ ]:
```python
# checking size of the data
diabetes_df.shape
```

[ ]:
```
(441456, 330)
```

Total of 330 columns and 441,456 rows of data. There are too many columns to work with. Columns to remove: empty columns, redundant columns, and irrelvant columns.

[ ]:
```python
# Dimensionality Reduction

# Looking for any empty columns (85% or higher was empty)
nan_cols85 = [i for i in diabetes_df.columns if diabetes_df[i].isnull().sum()\
              > 0.85*len(diabetes_df)]

# nan_cols85 <- was too long so it's included below in a condensed format
```

[ ]:
```python
# Removal of Empty Columns
diabetes_df85 = diabetes_df.copy(deep=True)
diabetes_df85 = diabetes_df85.drop(['COLGHOUS','LADULT','CCLGHOUS','ASTHNOW',
'DIABAGE2','NUMPHON2','PREGNANT','STOPSMK2','INSULIN','BLDSUGAR','FEETCHK2',
'DOCTDIAB','CHKHEMO3','FEETCHK','EYEEXAM','DIABEYE', 'DIABEDU', 'PAINACT2',
'QLMENTL2', 'QLSTRES2', 'QLHLTH2', 'CRGVREL1', 'CRGVLNG1', 'CRGVHRS1',
'CRGVPRB1', 'CRGVPERS', 'CRGVHOUS', 'CRGVMST2', 'VIDFCLT2', 'VIREDIF3',
'VIPRFVS2', 'VINOCRE2', 'VIEYEXM2', 'VIINSUR2', 'VICTRCT4', 'VIGLUMA2',
'VIMACDG2', 'CDHOUSE', 'CDASSIST', 'CDHELP', 'CDSOCIAL', 'CDDISCUS', 'WTCHSALT',
'LONGWTCH', 'DRADVISE', 'ASTHMAGE', 'ASATTACK', 'ASERVIST', 'ASDRVIST',
'ASRCHKUP', 'ASACTLIM', 'ASYMPTOM', 'ASNOSLEP', 'ASTHMED3', 'ASINHALR',
'HAREHAB1', 'STREHAB1', 'CVDASPRN', 'ASPUNSAF', 'RLIVPAIN', 'RDUCHART',
'RDUCSTRK', 'ARTTODAY', 'ARTHWGT', 'ARTHEXER', 'ARTHEDU', 'TETANUS', 'HPVADVC2',
'HPVADSHT', 'SHINGLE2', 'HADMAM', 'HOWLONG', 'HADPAP2', 'LASTPAP2', 'HPVTEST',
'HPLSTTST', 'HADHYST2', 'PROFEXAM', 'LENGEXAM', 'BLDSTOOL', 'LSTBLDS3',
'HADSIGM3', 'HADSGCO1', 'LASTSIG3', 'PCPSAAD2', 'PCPSADI1', 'PCPSARE1',
'PSATEST1', 'PSATIME', 'PCPSARS1', 'PCPSADE1', 'SCNTPAID', 'SCNTWRK1',
'SCNTLPAD', 'SCNTLWK1', 'RCSGENDR', 'RCSRLTN2', 'CASTHDX2', 'CASTHNO2',
'EMTSUPRT', 'LSATISFY', 'ADPLEASR', 'ADDOWN', 'ADSLEEP', 'ADENERGY', 'ADEAT1',
'ADFAIL', 'ADTHINK', 'ADMOVE', 'MISTMNT', 'ADANXEV', '_CRACE1', '_CPRACE',
'_CLLCPWT'], axis=1)
```

```
[ ]: diabetes_df85.shape
```

```
[ ]: (441456, 216)
```

List of empty columns that were removed (filtered out columns having more nan values than a threshold of value of 85%):

'COLGHOUS', 'LADULT', 'CCLGHOUS', 'ASTHNOW', 'DIABAGE2', 'NUMPHON2', 'PREGNANT', 'STOPSMK2', 'INSULIN', 'BLDSUGAR', 'FEETCHK2', 'DOCTDIAB', 'CHKHEMO3', 'FEETCHK',' EYEEXAM', 'DIABEYE', 'DIABEDU', 'PAINACT2', 'QLMENTL2', 'QLSTRES2', 'QLHLTH2', 'CRGVREL1', 'CRGVLNG1', 'CRGVHRS1', 'CRGVPRB1', 'CRGVPERS', 'CRGVHOUS', 'CRGVMST2', 'VIDFCLT2', 'VIREDIF3', 'VIPRFVS2', 'VINOCRE2', 'VIEYEXM2', 'VIINSUR2', 'VICTRCT4', 'VIGLUMA2', 'VIMACDG2', 'CDHOUSE', 'CDASSIST', 'CDHELP', 'CDSOCIAL', 'CDDISCUS', 'WTCHSALT', 'LONGWTCH', 'DRADVISE', 'ASTHMAGE', 'ASATTACK', 'ASERVIST', 'ASDRVIST', 'ASRCHKUP', 'ASACTLIM', 'ASYMPTOM', 'ASNOSLEP', 'ASTHMED3', 'ASINHALR', 'HAREHAB1', 'STREHAB1', 'CVDASPRN', 'ASPUNSAF', 'RLIVPAIN', 'RDUCHART', 'RDUCSTRK', 'ARTTODAY', 'ARTHWGT', 'ARTHEXER', 'ARTHEDU', 'TETANUS', 'HPVADVC2', 'HPVADSHT', 'SHINGLE2', 'HADMAM', 'HOWLONG', 'HADPAP2', 'LASTPAP2', 'HPVTEST', 'HPLSTTST', 'HADHYST2', 'PROFEXAM', 'LENGEXAM', 'BLDSTOOL', 'LSTBLDS3', 'HADSIGM3', 'HADSGCO1', 'LASTSIG3', 'PCPSAAD2', 'PCPSADI1', 'PCPSARE1', 'PSATEST1', 'PSATIME', 'PCPSARS1', 'PCPSADE1', 'SCNTPAID', 'SCNTWRK1', 'SCNTLPAD', 'SCNTLWK1', 'RCSGENDR', 'RCSRLTN2', 'CASTHDX2', 'CASTHNO2', 'EMTSUPRT', 'LSATISFY', 'ADPLEASR', 'ADDOWN', 'ADSLEEP', 'ADENERGY', 'ADEAT1', 'ADFAIL', 'ADTHINK', 'ADMOVE', 'MISTMNT', 'ADANXEV', '_CRACE1', '_CPRACE', '_CLLCPWT'

```
[ ]: diabetes_df85.head(3)
     # list(diabetes_df85) <- included below as 'List of variables that are left:'
```

```
[ ]:    _STATE  FMONTH        IDATE IMONTH   IDAY    IYEAR  DISPCODE       SEQNO  \
     0       1       1  b'01292015'  b'01'  b'29'  b'2015'      1200  2015000001
     1       1       1  b'01202015'  b'01'  b'20'  b'2015'      1100  2015000002
     2       1       1  b'02012015'  b'02'  b'01'  b'2015'      1200  2015000003

              _PSU  CTELENUM  …  _PAREC1  _PASTAE1  _LMTACT1  _LMTWRK1  _LMTSCL1  \
     0  2015000001       1.0  …        4         2       1.0       1.0       1.0
     1  2015000002       1.0  …        2         2       3.0       3.0       4.0
     2  2015000003       1.0  …        9         9       9.0       9.0       9.0

        _RFSEAT2  _RFSEAT3  _FLSHOT6  _PNEUMO2  _AIDTST3
     0         1         1       NaN       NaN       1.0
     1         2         2       NaN       NaN       2.0
     2         9         9       9.0       9.0       NaN

     [3 rows x 216 columns]
```

List of the variables that are left:

['STATE', 'FMONTH', 'IDATE', 'IMONTH', 'IDAY', 'IYEAR', 'DISPCODE', 'SEQNO', '_PSU', 'CTELENUM', 'PVTRESD1', 'STATERES', 'CELLFON3', 'NUMADULT', 'NUMMEN', 'NUMWOMEN', 'CTELNUM1', 'CELLFON2', 'CADULT', 'PVTRESD2', 'CSTATE', 'LANDLINE', 'HHADULT', 'GENHLTH', 'PHYSHLTH', 'MENTHLTH', 'POORHLTH', 'HLTHPLN1', 'PERSDOC2', 'MEDCOST', 'CHECKUP1', 'BPHIGH4', 'BPMEDS', 'BLOODCHO', 'CHOLCHK', 'TOLDHI2', 'CVDINFR4', 'CVDCRHD4', 'CVDSTRK3', 'ASTHMA3', 'CHCSCNCR', 'CHCOCNCR', 'CHCCOPD1', 'HAVARTH3', 'ADDEPEV2', 'CHCKIDNY', 'DIABETE3', 'SEX', 'MARITAL', 'EDUCA', 'RENTHOM1', 'NUMHHOL2', 'CPDEMO1', 'VETERAN3', 'EMPLOY1', 'CHILDREN', 'INCOME2', 'INTERNET', 'WEIGHT2', 'HEIGHT3', 'QLACTLM2', 'USEEQUIP', 'BLIND', 'DECIDE', 'DIFFWALK', 'DIFFDRES', 'DIFFALON', 'SMOKE100', 'SMOKDAY2', 'LASTSMK2', 'USENOW3', 'ALCDAY5', 'AVEDRNK2', 'DRNK3GE5', 'MAXDRNKS', 'FRUITJU1', 'FRUIT1', 'FVBEANS', 'FVGREEN', 'FVORANG', 'VEGETAB1', 'EXERANY2', 'EXRACT11', 'EXEROFT1', 'EXERHMM1', 'EXRACT21', 'EXEROFT2', 'EXERHMM2', 'STRENGTH', 'LMTJOIN3', 'ARTHDIS2', 'ARTHSOCL', 'JOINPAIN', 'SEATBELT', 'FLUSHOT6', 'FLSHTMY2', 'IMFVPLAC', 'PNEUVAC3', 'HIVTST6', 'HIVTSTD3', 'WHRTST10', 'PDIABTST', 'PREDIAB1', 'CAREGIV1', 'CRGVEXPT', 'CIMEMLOS', 'PCDMDECN', 'SCNTMNY1', 'SCNTMEL1', 'SXORIENT', 'TRNSGNDR', 'QSTVER', 'QSTLANG', 'EXACTOT1', 'EXACTOT2', 'MSCODE', '_STSTR', '_STRWT', '_RAWRAKE', '_WT2RAKE', '_CHISPNC', '_DUALUSE', '_DUALCOR', '_LLCPWT', '_RFHLTH', '_HCVU651', '_RFHYPE5', '_CHOLCHK', '_RFCHOL', '_MICHD', '_LTASTH1', '_CASTHM1', '_ASTHMS1', '_DRDXAR1', '_PRACE1', '_MRACE1', '_HISPANC', '_RACE', '_RACEG21', '_RACEGR3', '_RACE_G1', '_AGEG5YR', '_AGE65YR', '_AGE80', '_AGE_G', 'HTIN4', 'HTM4', 'WTKG3', '_BMI5', '_BMI5CAT', '*RFBMI5*', 'CHLDCNT', '*EDUCAG*', 'INCOMG', '*SMOKER3*', 'RFSMOK3', 'DRNKANY5', 'DROCDY3', 'RFBING5', '*DRNKWEK*', 'RFDRHV5', 'FTJUDA1', 'FRUTDA1*', 'BEANDAY*', 'GRENDAY*', 'ORNGDAY*', 'VEGEDA1*', 'MISFRTN*', 'MISVEGN', '*FRTRESP*', 'VEGRESP', '*FRUTSUM*', 'VEGESUM', '*FRTLT1*', 'VEGLT1', '*FRT16*', 'VEG23', '*FRUITEX*', 'VEGETEX', '*TOTINDA*', '*METVL11*', '*METVL21*', 'MAXVO2*', 'FC60*', 'ACTIN11*', 'ACTIN21*', 'PADUR1*', 'PADUR2*', 'PAFREQ1*', 'PAFREQ2*', 'MINAC11', '*MINAC21*', '*STRFREQ*', '*PAMISS1*', 'PAMIN11*', 'PAMIN21*', 'PA1MIN*', 'PAVIG11*', 'PAVIG21*', 'PA1VIGM*', '_PACAT1', '_PAINDX1', '_PA150R2', '_PA300R2', '_PA30021', '_PASTRNG', '_PAREC1', '_PASTAE1', '_LMTACT1', '_LMTWRK1', '_LMTSCL1', '_RFSEAT2', '_RFSEAT3', '_FLSHOT6', '_PNEUMO2', '_AIDTST3']

```
[ ]: # Subsetting Data (removal of redundant/irrelevant variables)
     diabetes_df85RI = diabetes_df85[['HHADULT', 'GENHLTH', 'PHYSHLTH', 'MENTHLTH',
                                     'HLTHPLN1', 'BPMEDS','_RFCHOL', 'HTM4',␣
      ↪'WTKG3',
                                     'ORNGDAY_','GRENDAY_', 'FRUTDA1_', 'FTJUDA1_',
                                     '_SMOKER3','ASTHMA3',  '_MICHD', 'SEX',
                                     'BPHIGH4','DIABETE3']]
```

A set of 19 independent variables alongside the dependent variable was chosen for the predictive analysis based on previous research, which indicated these variables as being an important risk factor for diabetes. Variables that were redundant or irrelevant were removed through this process. Examples of redundant data would include 'SEQNO' and '_PSU' which both indicate the patient's

ID or ’CTELENUM’and ’PVTRESD1’which indicated whether or not the survey was taken using a landline. Additionally, there were the variables ’NUMADULT’ and ’HHADULT’ which both indicated the number of adults in a household. An example of irrelevant data that was removed would include the variable ’CELLFON3’ which indicated whether or not the study was conducted via a cell phone.

- Dependent Variable: ’DIABETE3’
- Independent Variables: ’HHADULT’, ’GENHLTH’, ’PHYSHLTH’, ’MENTHLTH’,’HLTHPLN1’, ’BPMEDS’,’RFCHOL’, ’HTM4’, ’WTKG3’, ’ORNGDAY’,’GRENDAY_’, ’FRUTDA1_’, ’FTJUDA1_’, ’_SMOKER3’,’ASTHMA3’, ’_MICHD’, ’SEX’, ’BPHIGH4’

```
[ ]: import warnings
     warnings.filterwarnings("ignore")
```

Preprocessing Categorical Variables:

```
[ ]: # GENHLTH
     diabetes_df85RI['GENHLTH']=\
      diabetes_df85RI['GENHLTH'].replace({1:0, 2:0, 3:0, 4:1, 5:1, 7:99999, 9:99999})
     diabetes_df85RI.GENHLTH.unique()
```

```
[ ]: array([    0., 99999.,     nan])
```

0 = good health, 1 = bad health

```
[ ]: # HLTHPLN1
     diabetes_df85RI['HLTHPLN1']\
     = diabetes_df85RI['HLTHPLN1'].replace({1:0, 2:1, 7:99999, 9:99999})

     diabetes_df85RI.HLTHPLN1.unique()
```

```
[ ]: array([    0,     1, 99999])
```

0 = have healthcare, 1 = no health care

```
[ ]: # BPMEDS
     diabetes_df85RI['BPMEDS']\
     = diabetes_df85RI['BPMEDS'].replace({1:0, 2:1, 7:99999, 9:99999})

     diabetes_df85RI.BPMEDS.unique()
```

```
[ ]: array([0.0000e+00,        nan, 1.0000e+00, 9.9999e+04])
```

0 = taking BP meds, 1 = not taking BP meds

```
[ ]: # _SMOKER3
     diabetes_df85RI['_SMOKER3']\
     = diabetes_df85RI['_SMOKER3'].replace({3:0, 4:0, 1:1, 2:1,9:99999})
```

```
diabetes_df85RI._SMOKER3.unique()
```

```
[ ]: array([    0,     1, 99999])
```

0 = non smoker, 1 = smoker

```
[ ]: # ASTHMA3
diabetes_df85RI['ASTHMA3']\
= diabetes_df85RI['ASTHMA3'].replace({2:0, 1:1, 7:99999, 9:99999})

diabetes_df85RI.ASTHMA3.unique()
```

```
[ ]: array([    1,     0, 99999])
```

0 = no asthma, 1 = asthma

```
[ ]: # MICHD
diabetes_df85RI['_MICHD']\
= diabetes_df85RI['_MICHD'].replace({2:0, 1:1, 7:99999, 9:99999})

diabetes_df85RI._MICHD.unique()
```

```
[ ]: array([ 0., nan,  1.])
```

0 = no coronary heart disease or myocardial infarction, 1 = heart disease

```
[ ]: # _RFCHOL
diabetes_df85RI['_RFCHOL']\
= diabetes_df85RI['_RFCHOL'].replace({1:0, 2:1, 9:99999})

diabetes_df85RI._RFCHOL.unique()
```

```
[ ]: array([1.0000e+00, 0.0000e+00,        nan, 9.9999e+04])
```

0 = okay LDL, 1 = high LDL

```
[ ]: # BPHIGH4
diabetes_df85RI['BPHIGH4']\
= diabetes_df85RI['BPHIGH4'].replace({2:0, 3:0, 4:0, 7:99999, 9:99999})

diabetes_df85RI.BPHIGH4.unique()
```

```
[ ]: array([1.0000e+00, 0.0000e+00, 9.9999e+04,        nan])
```

0 = normal BP, 1 = high bp

```
[ ]: # SEX
diabetes_df85RI['SEX']\
= diabetes_df85RI['SEX'].replace({2:0})

diabetes_df85RI.SEX.unique()
```

```
[ ]: array([0, 1])
```

0 = female, 1 = male

```
[ ]: # DIABETE3 -> Binary outcome with unknown
diabetes_df85RI['DIABETE3']\
= diabetes_df85RI['DIABETE3'].replace({2:1, 3:0, 4:0, 7:99999, 9:99999})

diabetes_df85RI.DIABETE3.unique()
```

```
[ ]: array([0.0000e+00, 1.0000e+00, 9.9999e+04,        nan])
```

1 = diabetic, 0 = not diabetic, 99999 = unknown (<- this will be imputed after splitting)

Recoding the unknown responses as 99999 to be imputed later for the numeric data:

```
[ ]: # HHADULT
diabetes_df85RI['HHADULT']\
= diabetes_df85RI['HHADULT'].replace({77:99999, 99:99999})
```

```
[ ]: # PHYSHLTH
diabetes_df85RI['PHYSHLTH']\
= diabetes_df85RI['PHYSHLTH'].replace({77:99999, 99:99999})
```

```
[ ]: # DIABETES3
diabetes_df85RI = diabetes_df85RI[diabetes_df85RI.DIABETE3 != 99999]
diabetes_df85RI = diabetes_df85RI.dropna(subset=['DIABETE3'])
```

```
[ ]: # MENTHLTH
diabetes_df85RI['MENTHLTH']\
= diabetes_df85RI['MENTHLTH'].replace({77:99999, 99:99999})
```

```
[ ]: diabetes_df85RI.BPHIGH4.unique()
```

```
[ ]: array([1.0000e+00, 0.0000e+00, 9.9999e+04,        nan])
```

```
[ ]: diabetes_df85RI.replace(99999, np.nan)
diabetes_df85RI.BPHIGH4.unique() # checking that the 99999 was replaced w/NA
```

```
[ ]: array([1.0000e+00, 0.0000e+00, 9.9999e+04,        nan])
```

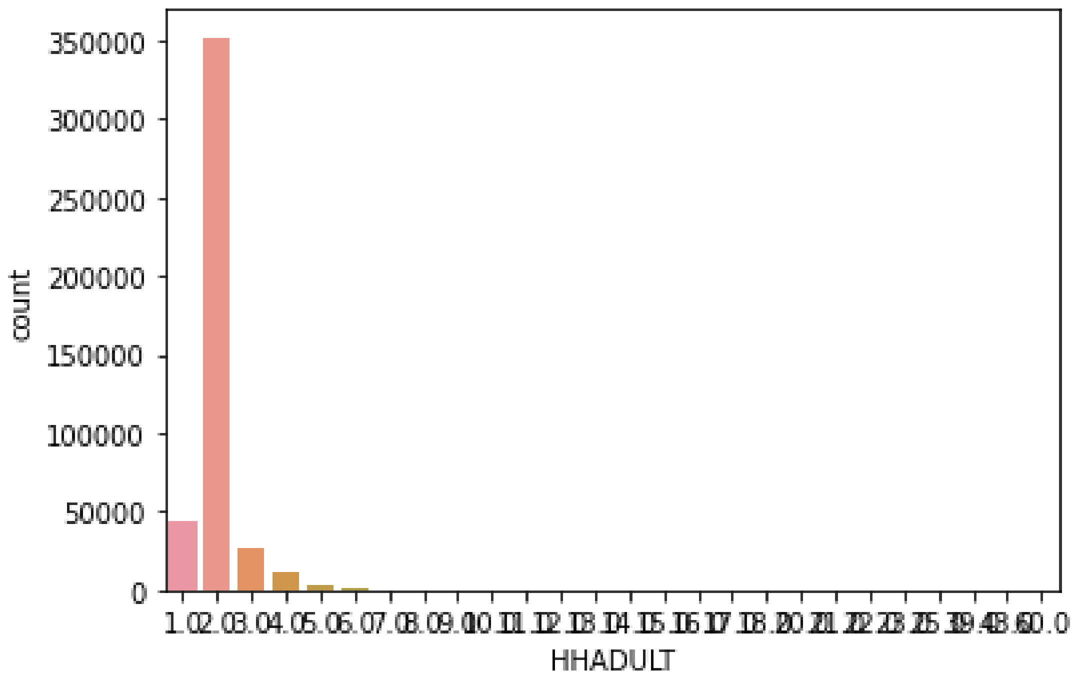
```
[ ]: # imputing missing values
diabetes_df85RI[diabetes_df85RI==99999] = np.nan
imp_mean = SimpleImputer( strategy='most_frequent')
```

```
imp_mean.fit(diabetes_df85RI)
diabetes_df85RIi = imp_mean.transform(diabetes_df85RI)
diabetes_df85RIi = pd.DataFrame(diabetes_df85RIi, columns = ['HHADULT',␣
↪'GENHLTH',
                                                              'PHYSHLTH',
          'MENTHLTH','HLTHPLN1', 'BPMEDS','_RFCHOL', 'HTM4', 'WTKG3',
           'ORNGDAY_','GRENDAY_', 'FRUTDA1_', 'FTJUDA1_','_SMOKER3','ASTHMA3',
           '_MICHD', 'SEX',
          'BPHIGH4','DIABETE3'])
diabetes_df85RIi.head()
diabetes_df85RIi.isnull().sum()
```

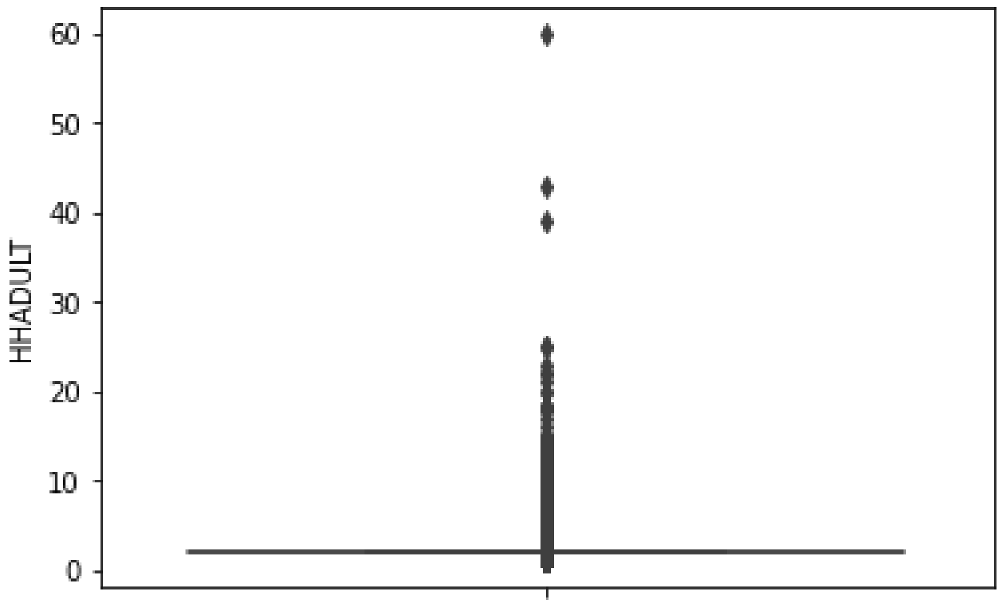
[ ]:
```
HHADULT     0
GENHLTH     0
PHYSHLTH    0
MENTHLTH    0
HLTHPLN1    0
BPMEDS      0
_RFCHOL     0
HTM4        0
WTKG3       0
ORNGDAY_    0
GRENDAY_    0
FRUTDA1_    0
FTJUDA1_    0
_SMOKER3    0
ASTHMA3     0
_MICHD      0
SEX         0
BPHIGH4     0
DIABETE3    0
dtype: int64
```

Preprocessing Numeric Variables:

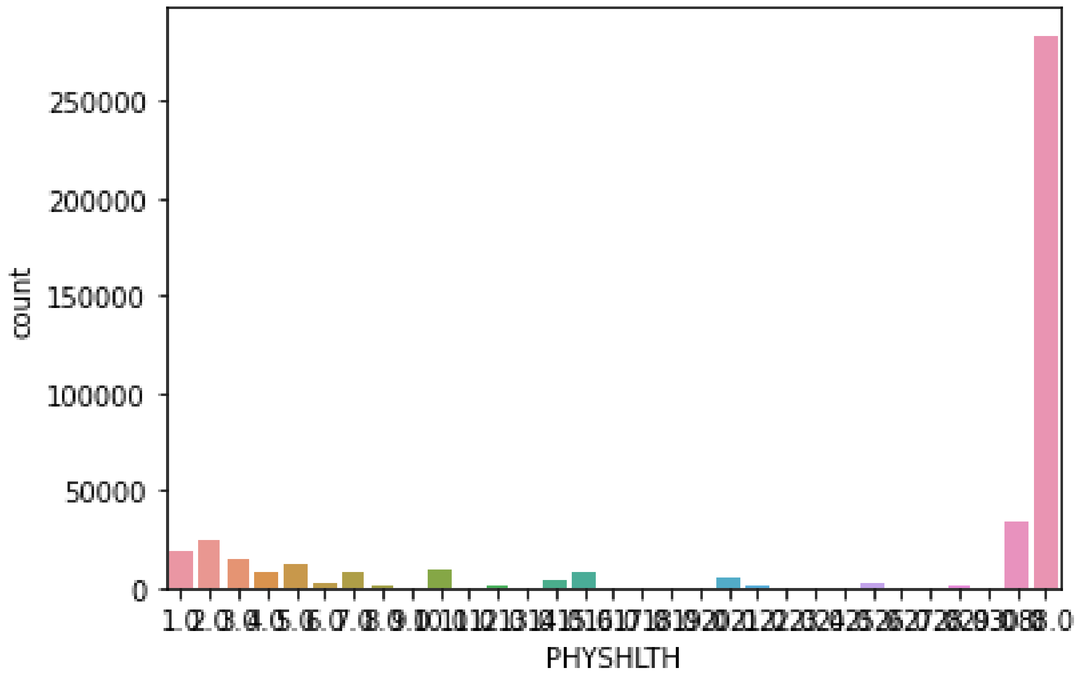
[ ]:
```
# HHADULT
ax = sns.countplot(x="HHADULT",data=diabetes_df85RIi)
```



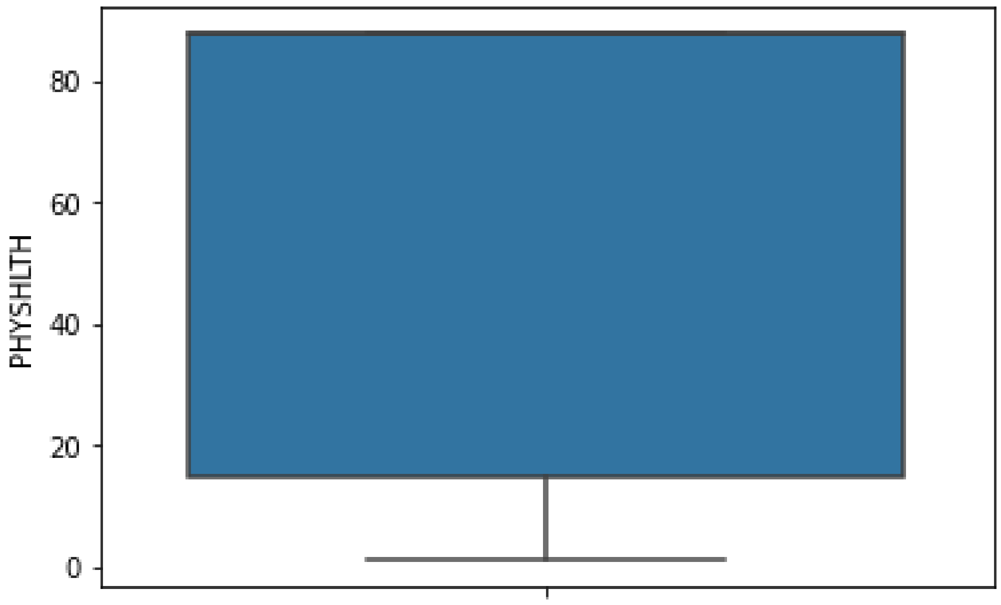
[ ]:
```
sns.boxplot( y=diabetes_df85RIi["HHADULT"]);
plt.show()
```



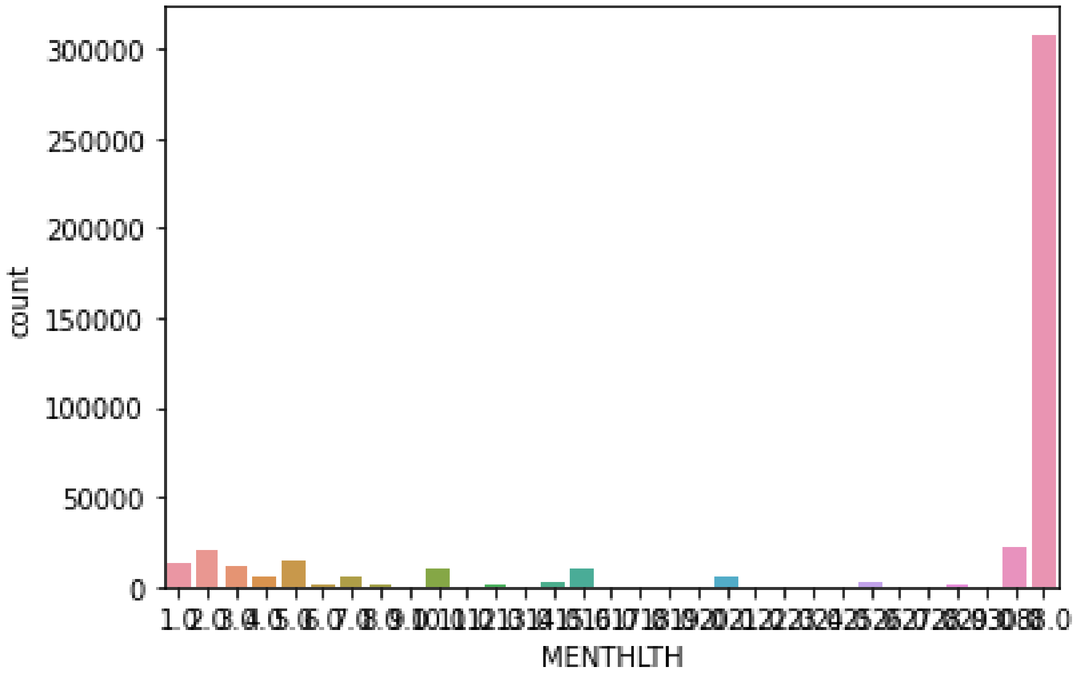
```
[ ]: # PHYSHLTH
     ax = sns.countplot(x="PHYSHLTH",data=diabetes_df85RIi)
```



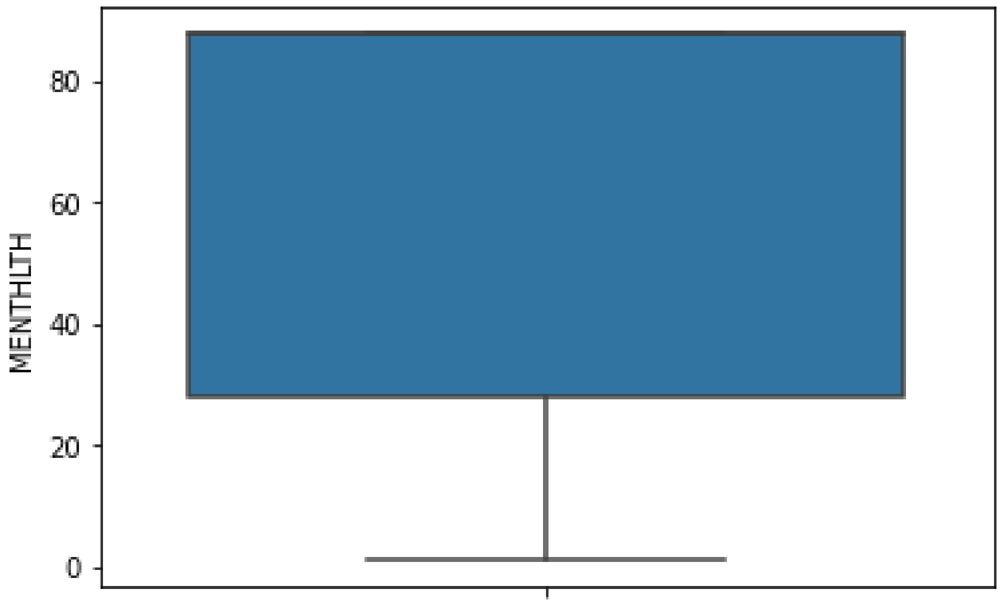
```
[ ]: sns.boxplot( y=diabetes_df85RIi["PHYSHLTH"]);
     plt.show()
```



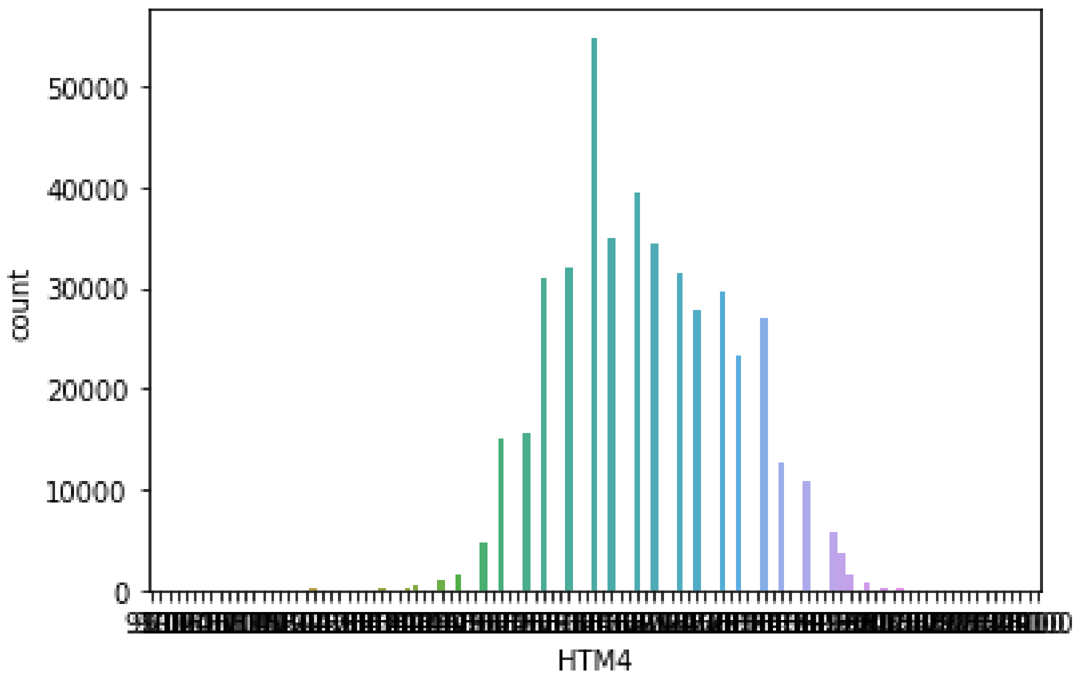
```
[ ]: # MENTHLTH
     ax = sns.countplot(x="MENTHLTH",data=diabetes_df85RIi)
```



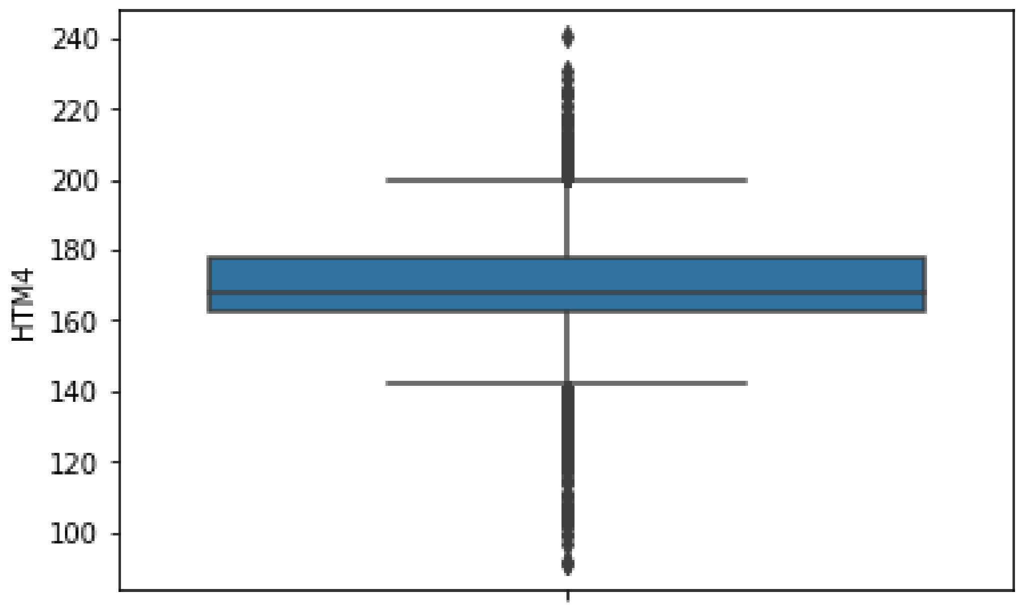
```
[ ]: sns.boxplot( y=diabetes_df85RIi["MENTHLTH"]);
     plt.show()
```



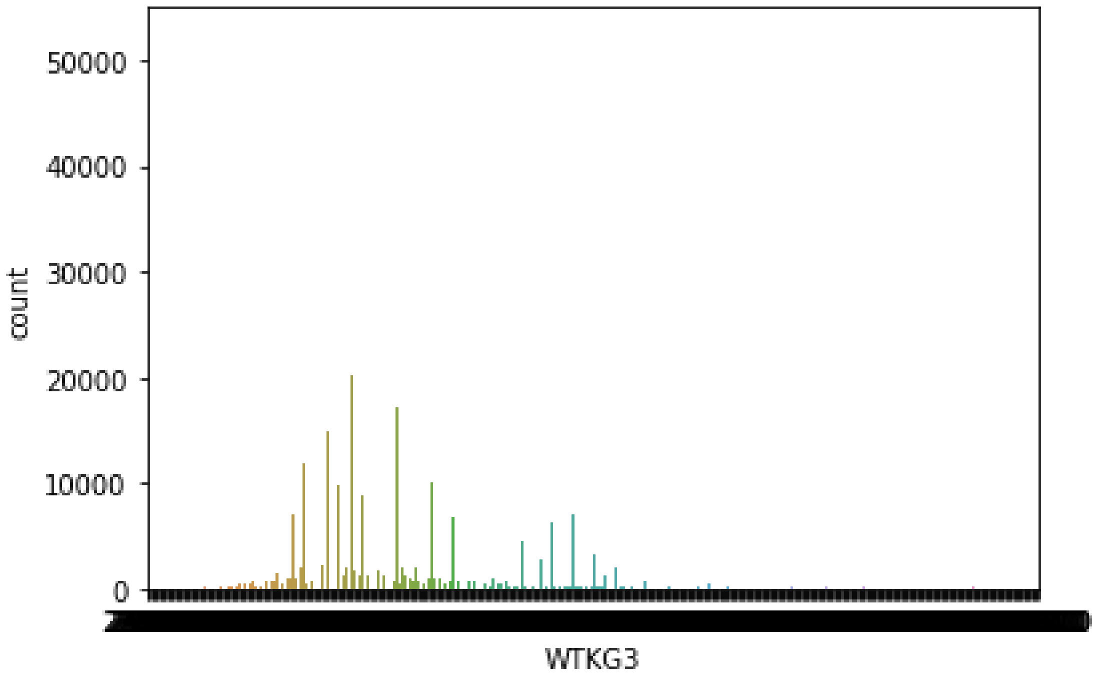
```
[ ]: # HTM4
     ax = sns.countplot(x="HTM4",data=diabetes_df85RIi)
```



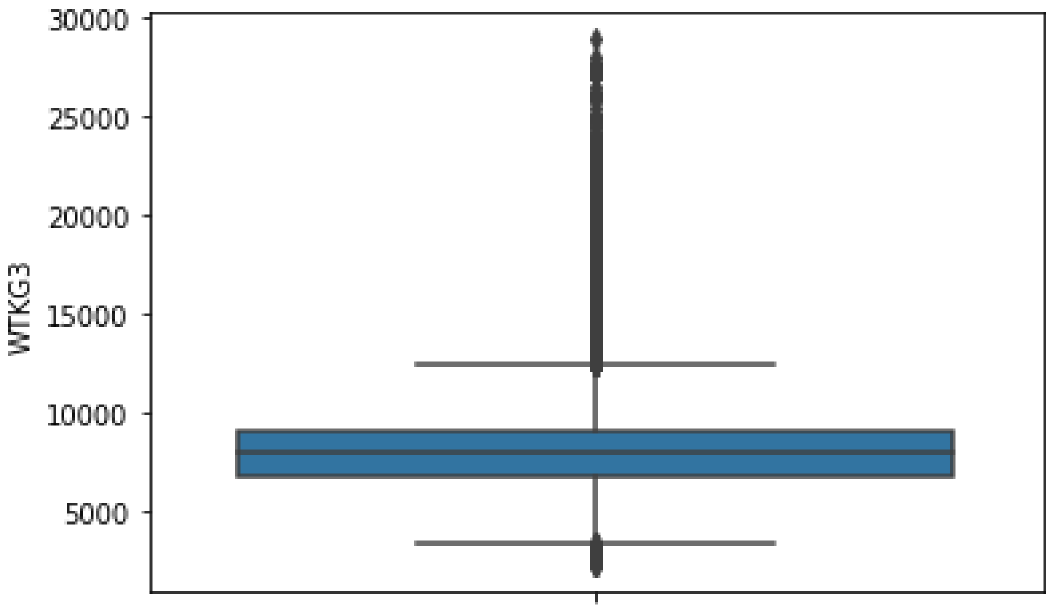
```
[ ]: sns.boxplot( y=diabetes_df85RIi["HTM4"]);
     plt.show()
```



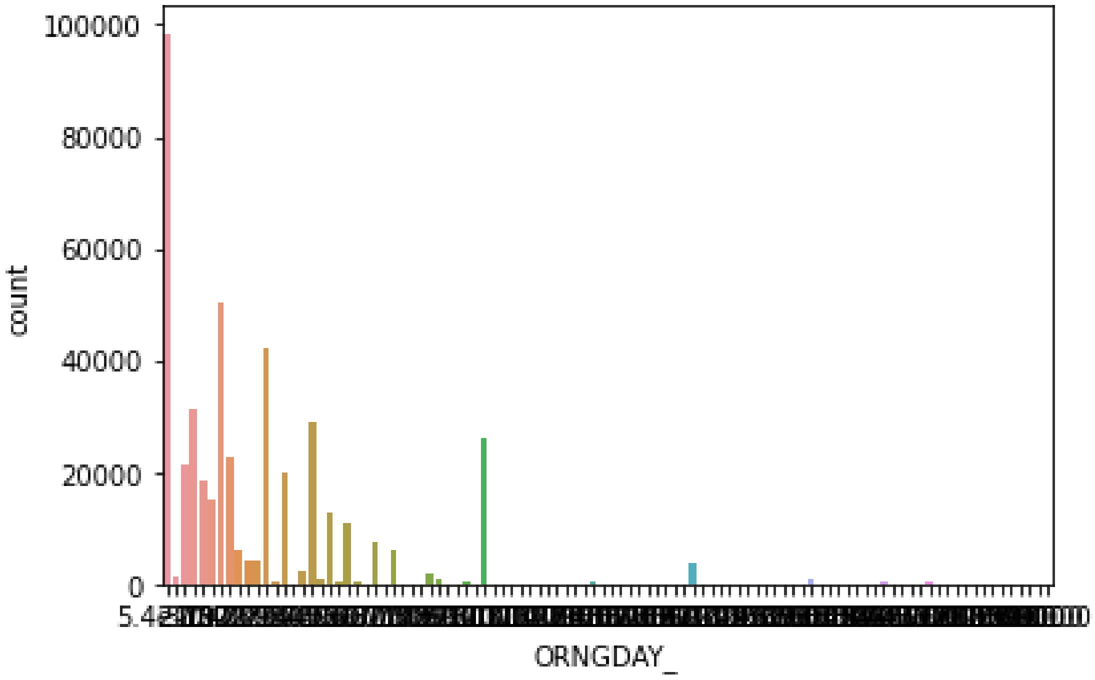
[ ]:
```
# WTKG3
ax = sns.countplot(x="WTKG3",data=diabetes_df85RIi)
```



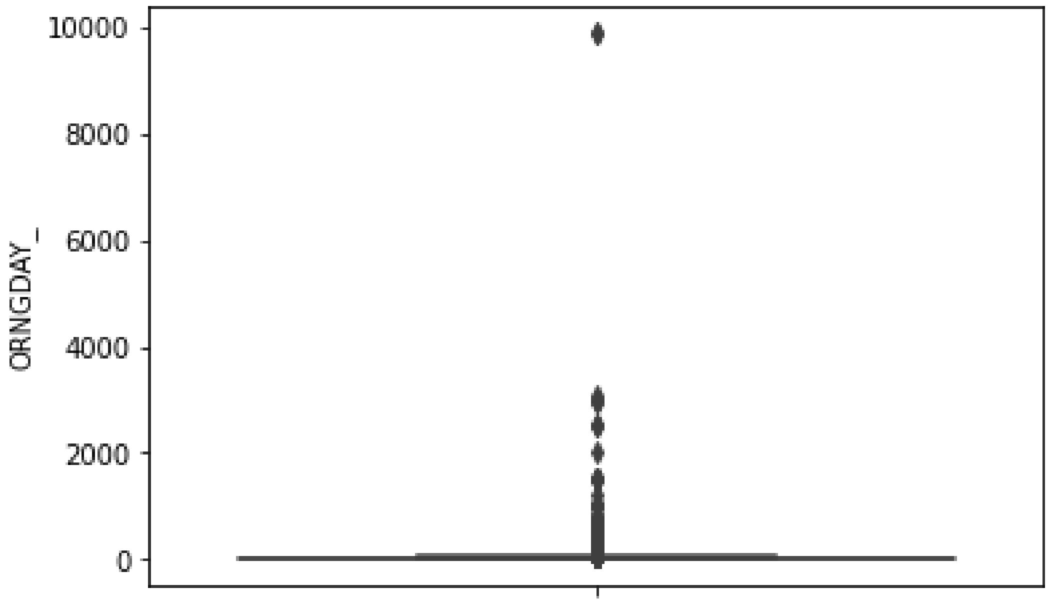
[ ]:
```
sns.boxplot( y=diabetes_df85RIi["WTKG3"]);
plt.show()
```



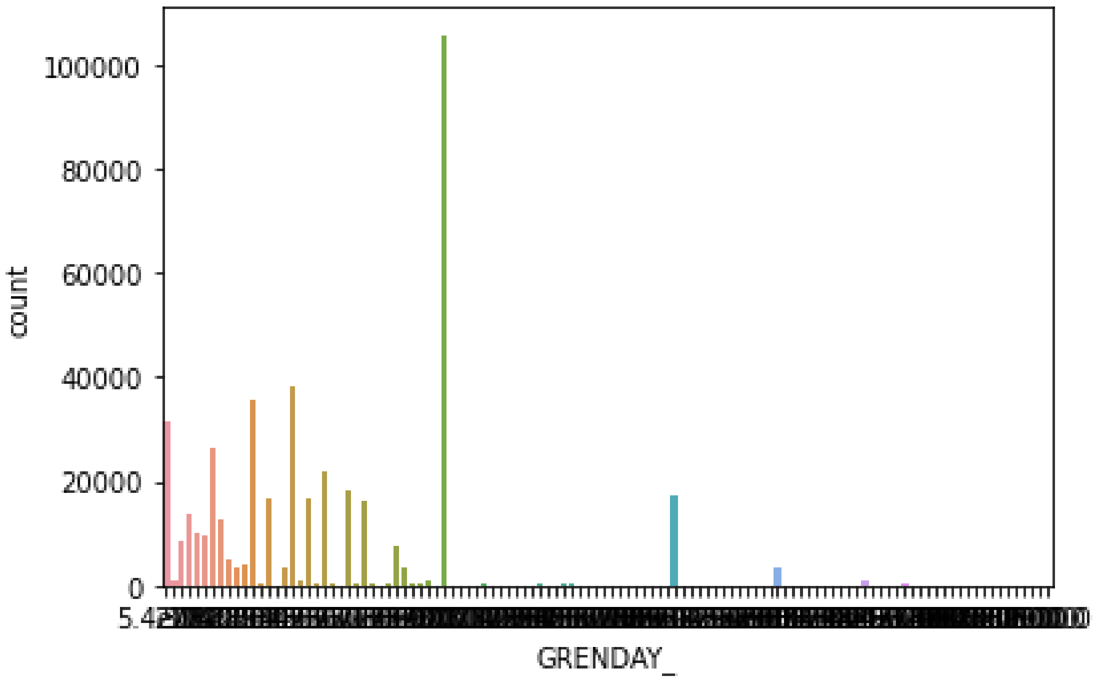
[ ]:
```
# ORNGDAY_
ax = sns.countplot(x="ORNGDAY_",data=diabetes_df85RIi)
```



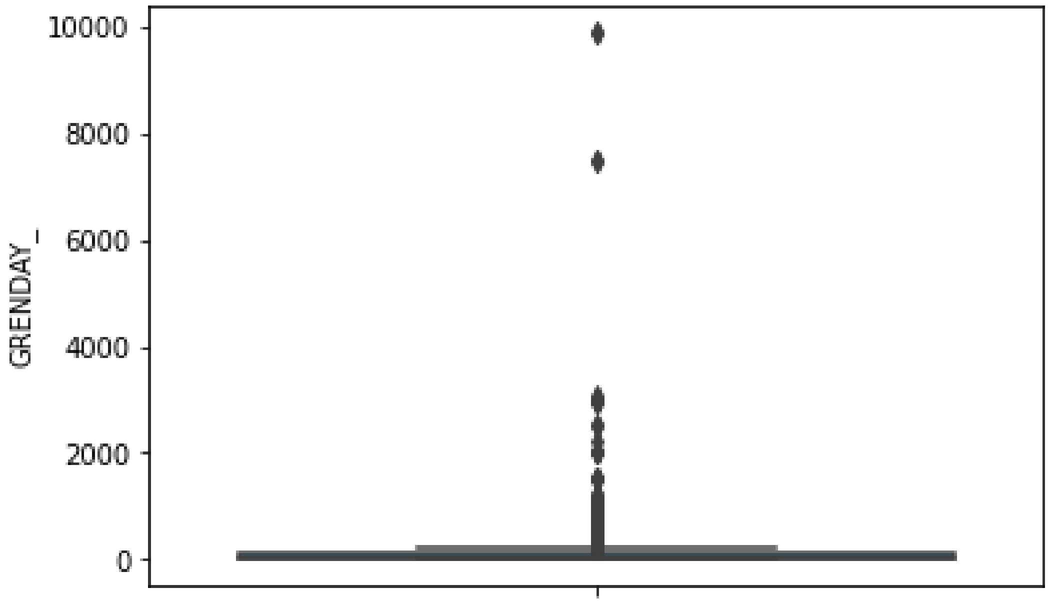
```
[ ]: sns.boxplot( y=diabetes_df85RIi["ORNGDAY_"]);
     plt.show()
```



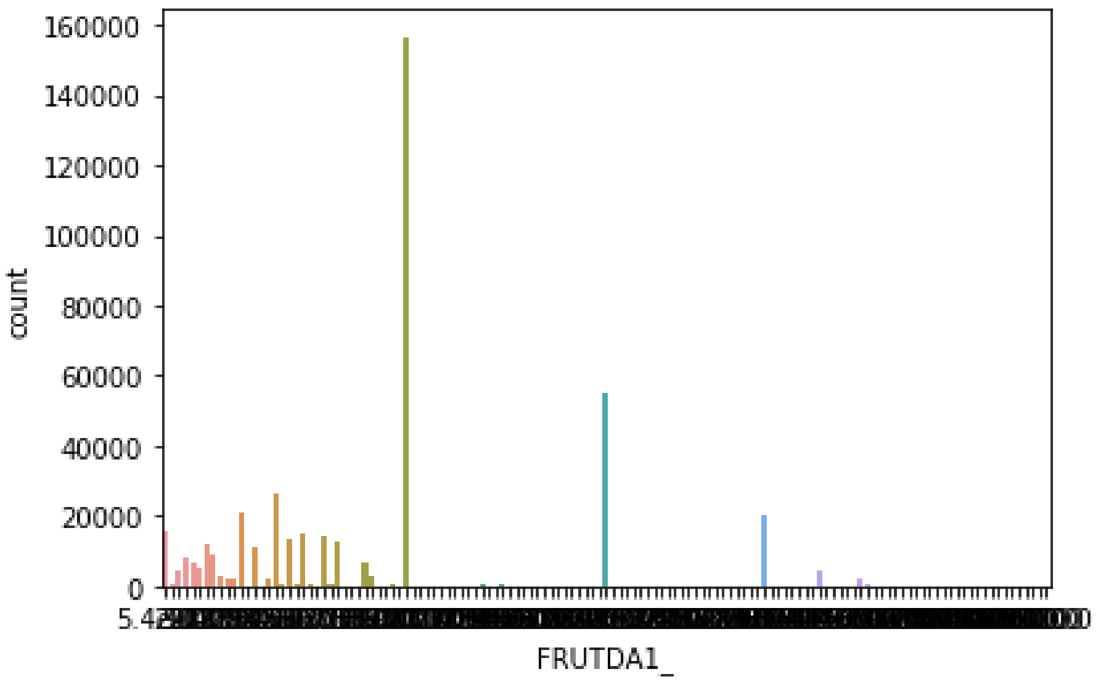
```
[ ]: # GRENDAY_
     ax = sns.countplot(x="GRENDAY_",data=diabetes_df85RIi)
```



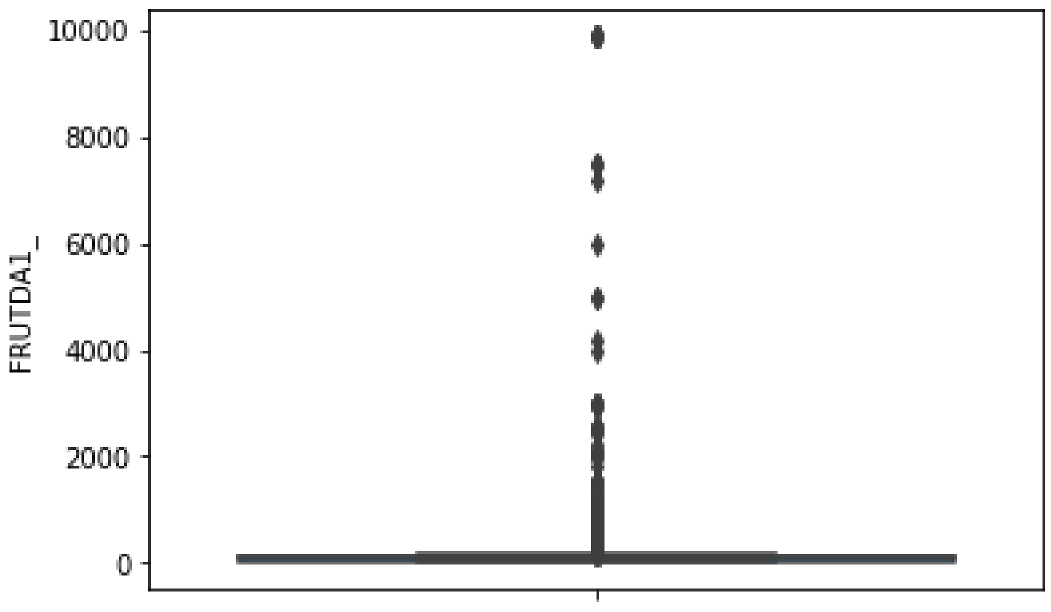
[ ]:
```
sns.boxplot( y=diabetes_df85RIi["GRENDAY_"]);
plt.show()
```



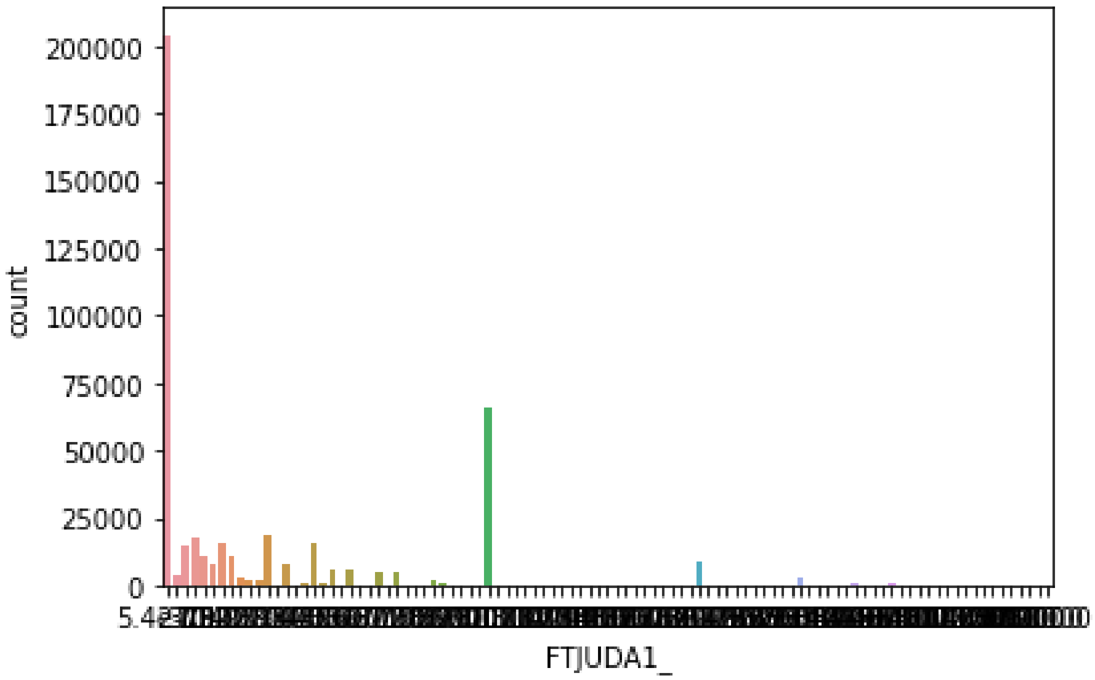
[ ]:
```
# FRUTDA1
ax = sns.countplot(x="FRUTDA1_",data=diabetes_df85RIi)
```



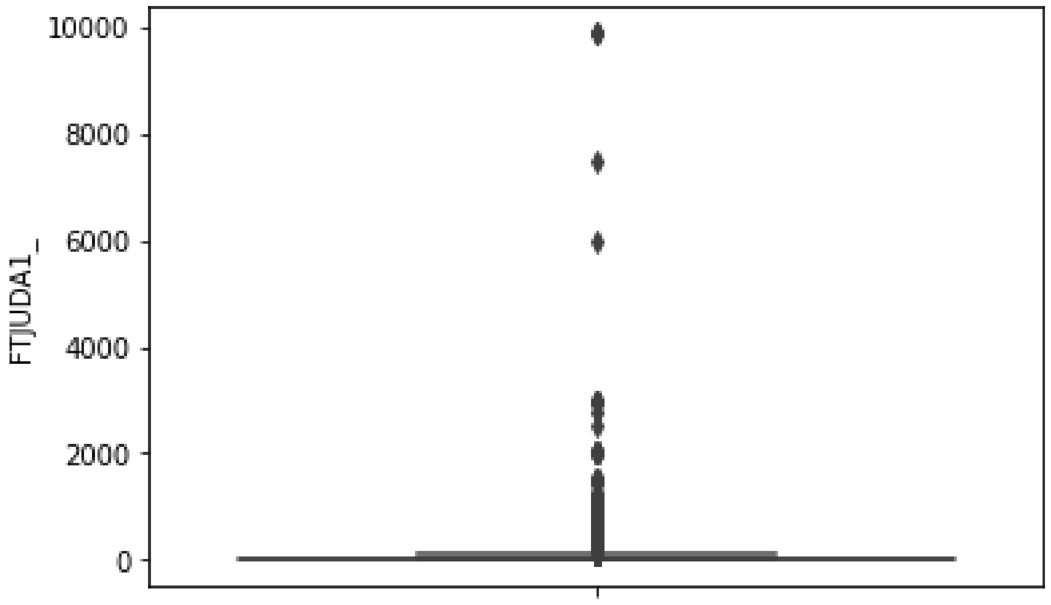
```
[ ]: sns.boxplot( y=diabetes_df85RIi["FRUTDA1_"]);
     plt.show()
```



```
[ ]: # FTJUDA1
     ax = sns.countplot(x="FTJUDA1_",data=diabetes_df85RIi)
```



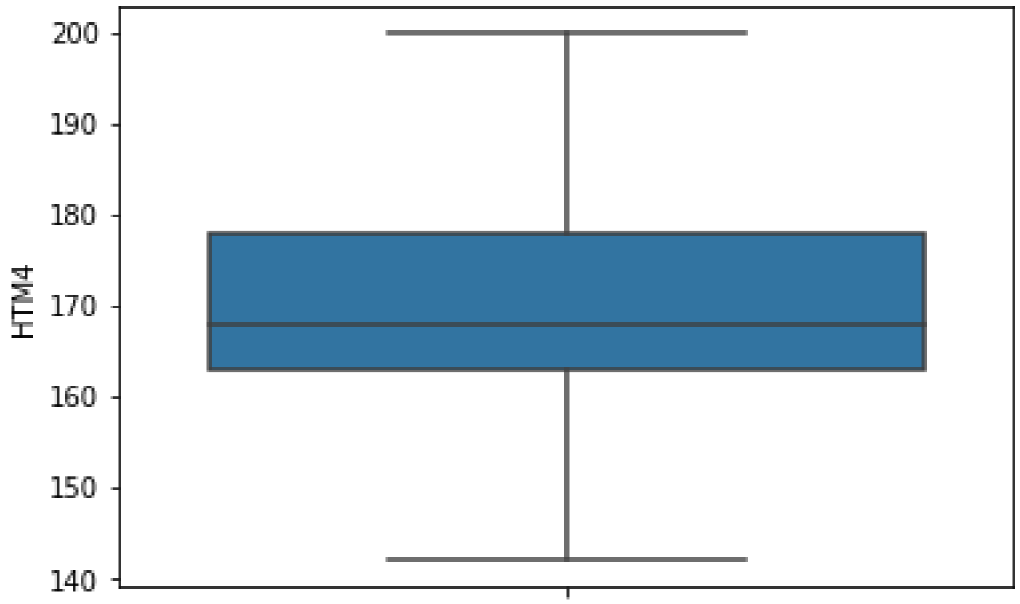
```
[ ]: sns.boxplot( y=diabetes_df85RIi["FTJUDA1_"]);
     plt.show()
```



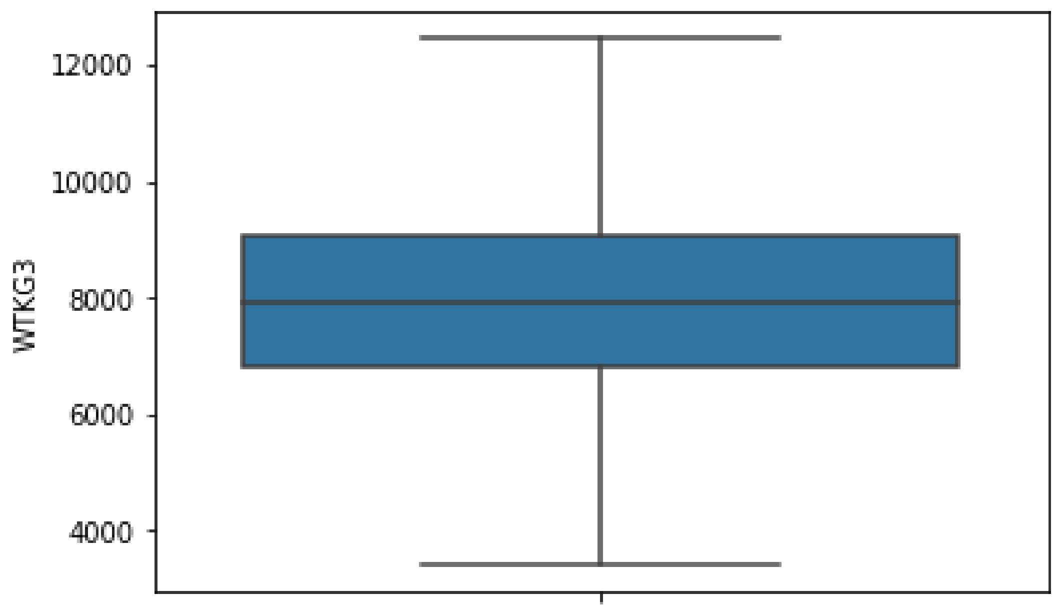
[ ]:
```
# handling outliers for numeric attributes
# (HHADULT, TRM4, WTKG3, ORNGDAY_, GRENDAY_, FRUTDA1_,
# FTJUDA1_), 'GENHLTH', 'PHYSHLTH', 'MENTHLTH'

cols = ['HHADULT', 'HTM4', 'WTKG3', 'ORNGDAY_', 'GRENDAY_', 'FRUTDA1_',
        'FTJUDA1_', 'GENHLTH', 'PHYSHLTH', 'MENTHLTH']

Q1 = diabetes_df85RIi[cols].quantile(0.25)
Q3 = diabetes_df85RIi[cols].quantile(0.75)
IQR = Q3 - Q1

diabetes_df85RIio = diabetes_df85RIi[~((diabetes_df85RIi[cols]\
                                        < (Q1 - 1.5 * IQR))\
                                        |(diabetes_df85RIi[cols] > \
                                          (Q3 + 1.5 * IQR))).any(axis=1)]
```

Outliers were handled using the IQR found for each of the variables.

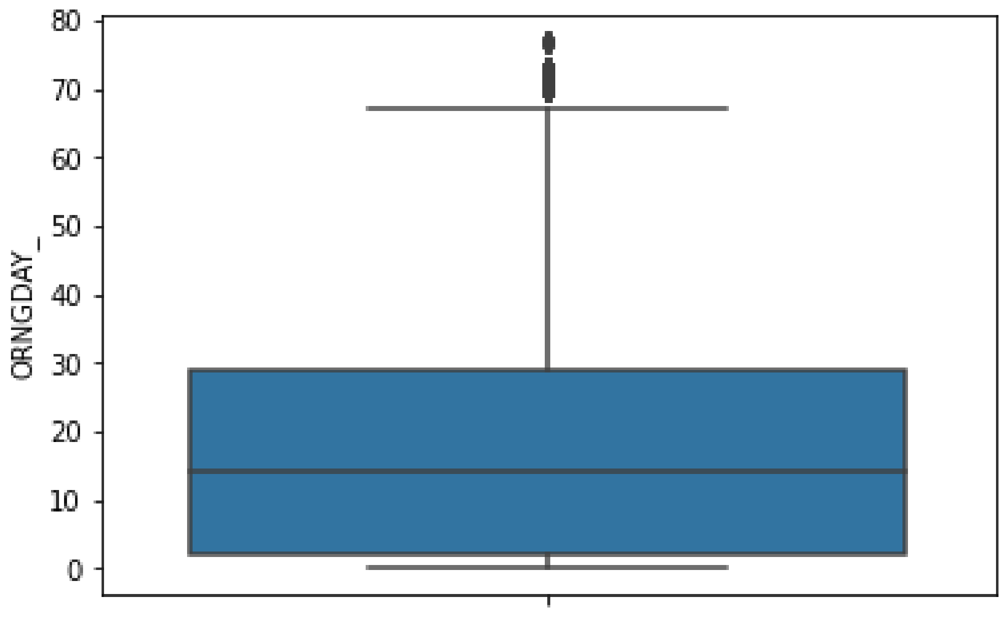
[ ]:
```
# HTM4 check
sns.boxplot( y=diabetes_df85RIio["HTM4"]);
plt.show()
```



[ ]:
```
# WTKG3 check
sns.boxplot( y=diabetes_df85RIio["WTKG3"]);
plt.show()
```

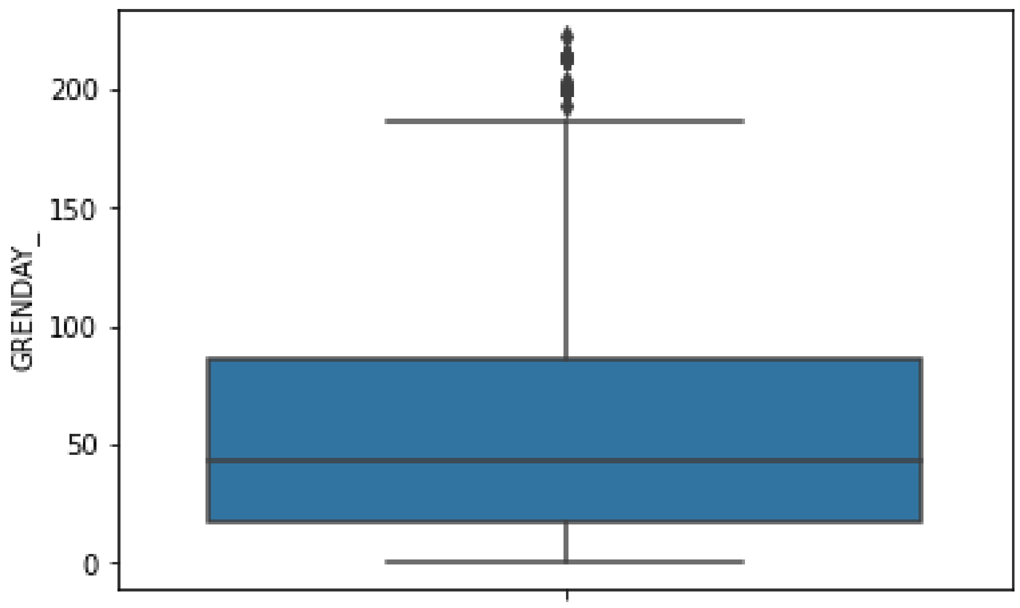


```
[ ]: # ORNGDAY_ check
     sns.boxplot( y=diabetes_df85RIio["ORNGDAY_"]);
     plt.show()
```

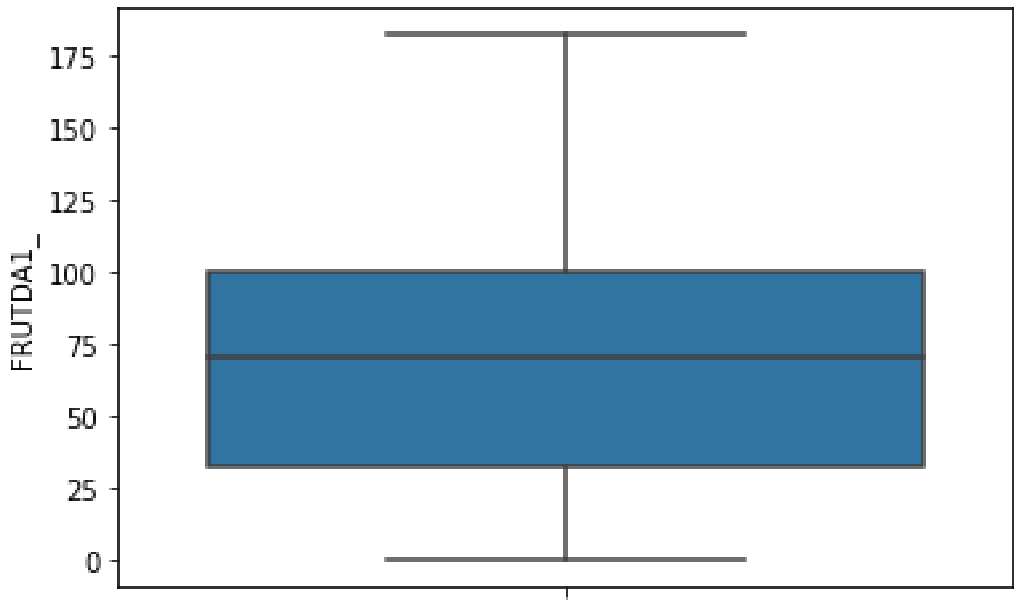


```
[ ]: # GRENDAY_ check
     sns.boxplot( y=diabetes_df85RIio["GRENDAY_"]);
```

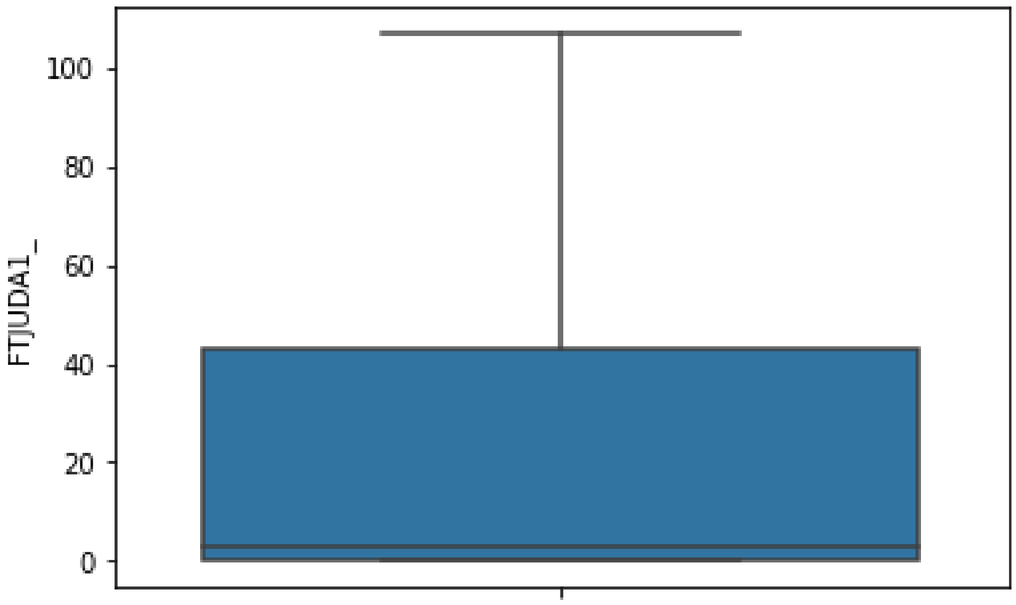
```
plt.show()
```



```
[ ]: # FRUTDA1_ check
     sns.boxplot( y=diabetes_df85RIio["FRUTDA1_"]);
     plt.show()
```



```
[ ]: # FTJUDA1_ check
     sns.boxplot( y=diabetes_df85RIio["FTJUDA1_"]);
     plt.show()
```

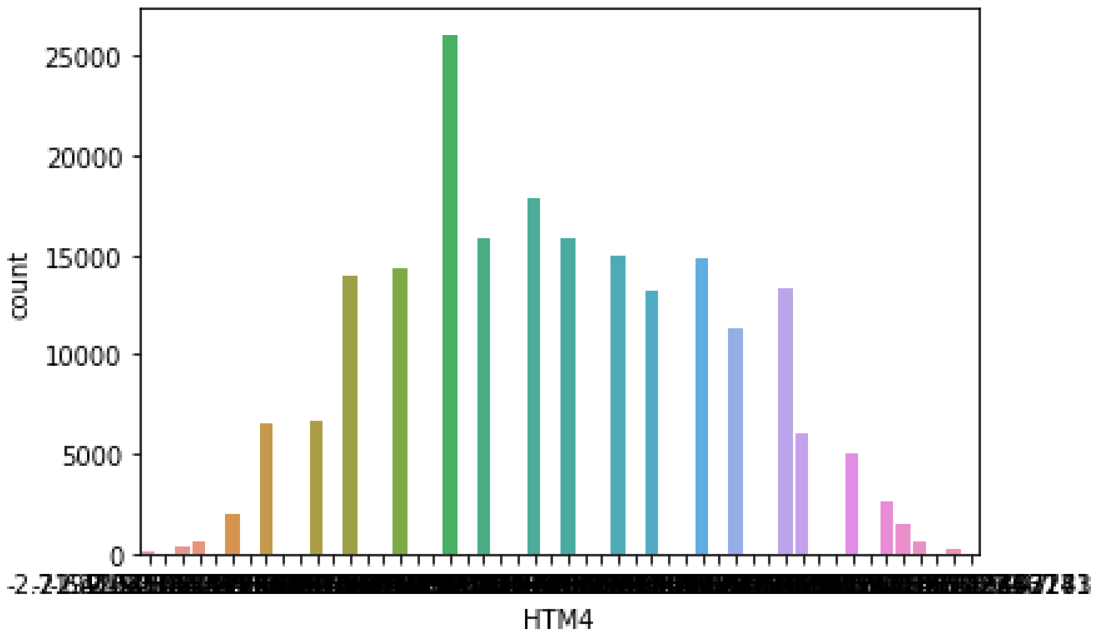


```
[ ]: diabetes_df85RIio.describe()
```

```
[ ]:          HHADULT   GENHLTH        PHYSHLTH        MENTHLTH        HLTHPLN1  \
     count  204112.0  204112.0   204112.000000   204112.000000   204112.000000
     mean        2.0       0.0       66.205069       68.374324        0.059428
     std         0.0       0.0       36.066599       34.554820        0.236425
     min         2.0       0.0        1.000000        1.000000        0.000000
     25%         2.0       0.0       30.000000       88.000000        0.000000
     50%         2.0       0.0       88.000000       88.000000        0.000000
     75%         2.0       0.0       88.000000       88.000000        0.000000
     max         2.0       0.0       88.000000       88.000000        1.000000

                  BPMEDS         _RFCHOL            HTM4           WTKG3  \
     count  204112.000000   204112.000000   204112.000000   204112.000000
     mean        0.057758        0.357779      169.520205     8005.650119
     std         0.233285        0.479348       10.118063     1656.616988
     min         0.000000        0.000000      142.000000     3402.000000
     25%         0.000000        0.000000      163.000000     6804.000000
     50%         0.000000        0.000000      168.000000     7938.000000
     75%         0.000000        1.000000      178.000000     9072.000000
     max         1.000000        1.000000      200.000000    12474.000000

                 ORNGDAY_       GRENDAY_       FRUTDA1_       FTJUDA1_        _SMOKER3  \
     count  2.041120e+05  2.041120e+05  2.041120e+05  2.041120e+05  204112.000000
     mean   1.809971e+01  5.196872e+01  6.657320e+01  2.492826e+01       0.125176
     std    1.815293e+01  3.972859e+01  3.591539e+01  3.605960e+01       0.330920
     min    5.400000e-79  5.400000e-79  5.400000e-79  5.400000e-79       0.000000
     25%    2.000000e+00  1.700000e+01  3.300000e+01  5.400000e-79       0.000000
     50%    1.400000e+01  4.300000e+01  7.100000e+01  3.000000e+00       0.000000
     75%    2.900000e+01  8.600000e+01  1.000000e+02  4.300000e+01       0.000000
     max    7.700000e+01  2.230000e+02  1.830000e+02  1.070000e+02       1.000000

                  ASTHMA3         _MICHD            SEX         BPHIGH4  \
     count  204112.000000  204112.000000  204112.000000  204112.000000
     mean        0.106912       0.063387       0.439494       0.381781
     std         0.309002       0.243658       0.496327       0.485824
     min         0.000000       0.000000       0.000000       0.000000
     25%         0.000000       0.000000       0.000000       0.000000
     50%         0.000000       0.000000       0.000000       0.000000
     75%         0.000000       0.000000       1.000000       1.000000
     max         1.000000       1.000000       1.000000       1.000000

                 DIABETE3
     count  204112.000000
     mean        0.105795
     std         0.307576
     min         0.000000
     25%         0.000000
     50%         0.000000
     75%         0.000000
     max         1.000000
```

[ ]:
```
diabetes_df85RIion = diabetes_df85RIio.copy()

# Normalizing the numeric data
# (HHADULT, HTM4, WTKG3, ORNGDAY_, GRENDAY_, FRUTDA1_,
# FTJUDA1_, 'GENHLTH', 'PHYSHLTH', 'MENTHLTH')
diabetes_df85RIion = diabetes_df85RIio.copy()

scaler = StandardScaler()

diabetes_df85RIion.iloc[:,7:13] = scaler.fit_transform(diabetes_df85RIion.iloc\
                                                        [:,7:13].to_numpy())

diabetes_df85RIion.iloc[:,1:2] = scaler.fit_transform(diabetes_df85RIion.iloc\
                                                        [:,1:2].to_numpy())
```

[ ]:
```
# ORNGDAY_ check
ax = sns.countplot(x="HTM4",data=diabetes_df85RIion)
```



```
[ ]: # Splitting of the data 80% Train/20% Test
     x = diabetes_df85RIion.loc[:, diabetes_df85RIion.columns !="DIABETE3"]
     y = diabetes_df85RIion['DIABETE3']
     x_train, x_test, y_train, y_test = train_test_split(x, y, stratify = y,
                                                          test_size=0.20,
                                                          random_state=42)
```

```
[ ]: x_train.shape
```

```
[ ]: (163289, 18)
```

```
[ ]: y_train.shape
```

```
[ ]: (163289,)
```

```
[ ]: x_test.shape
```

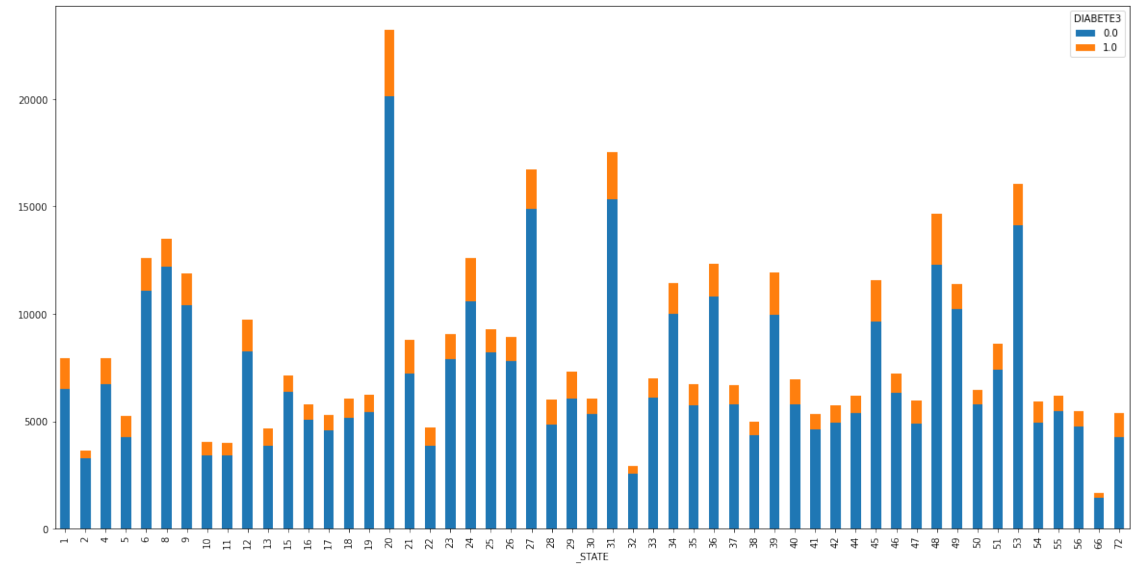
```
[ ]: (40823, 18)
```

```
[ ]: y_test.shape
```

```
[ ]: (40823,)
```

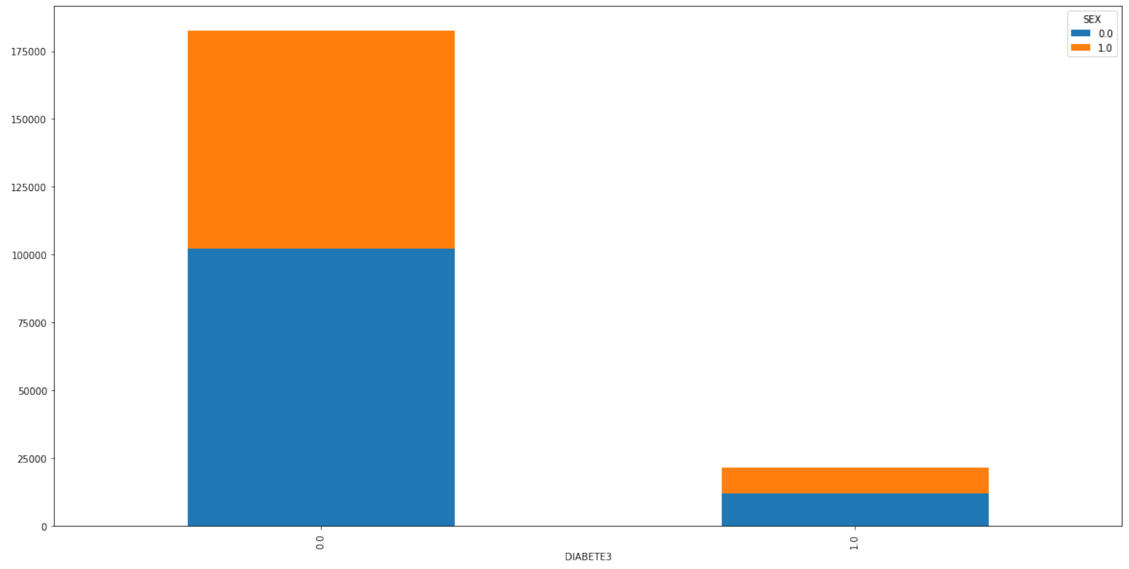
```
[ ]: # Number of people per state with diabetes as an overlay
     crosstab_1 = pd.crosstab(diabetes_df85['_STATE'], diabetes_df85RI['DIABETE3'])
     crosstab_1.plot(kind='bar', stacked=True, figsize = (20,10))
```

```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbe3539a910>
```



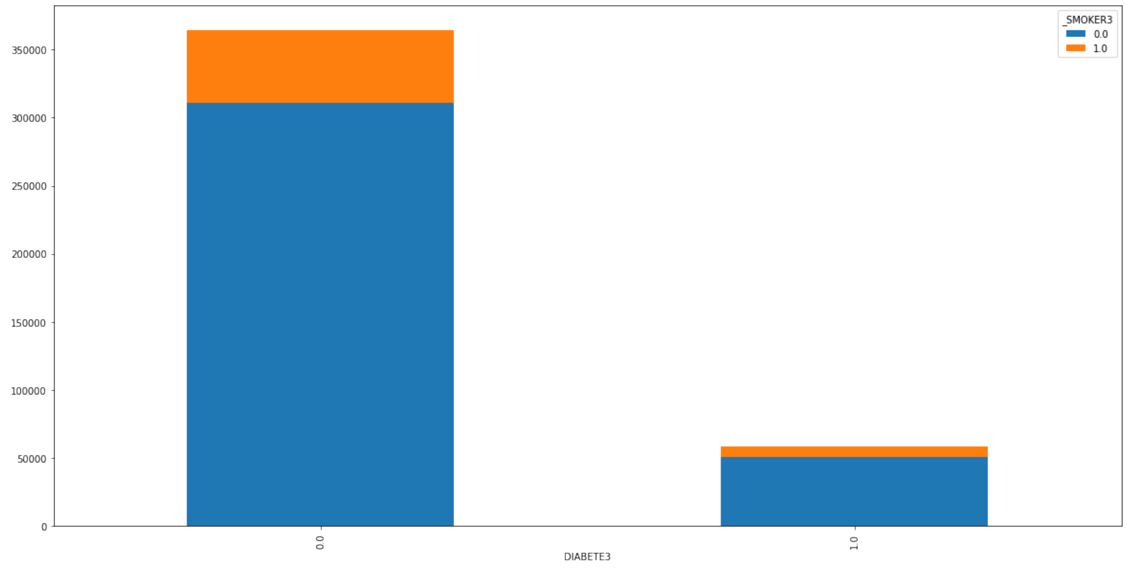
```
[ ]: # EDA: Diabetic vs. Non Diabetic for males vs. females as overlay
     crosstab_1 = pd.crosstab(diabetes_df85RIion['DIABETE3'],␣
      ↪diabetes_df85RIion['SEX'])
     crosstab_1.plot(kind='bar', stacked=True, figsize = (20,10))
```

```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbe34f51a10>
```



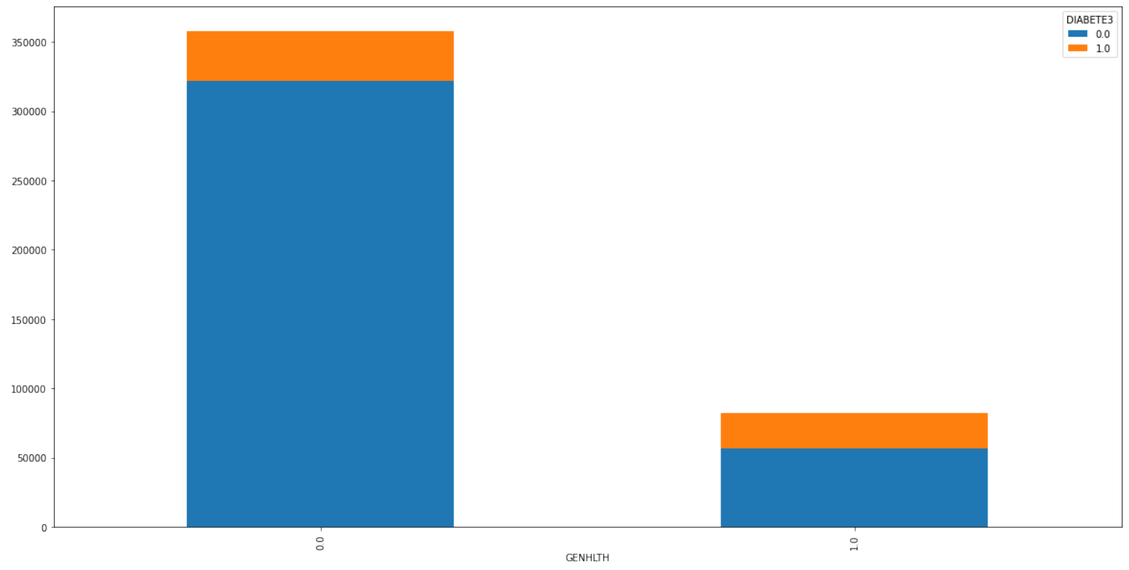
```
[ ]: # EDA: Diabetes vs. Non Diabetic for SMOKERS as overlay
     crosstab_1 = pd.crosstab(diabetes_df85RI['DIABETE3'],␣
      ↪diabetes_df85RI['_SMOKER3'])
     crosstab_1.plot(kind='bar', stacked=True, figsize = (20,10))
```

```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbe360ffcd0>
```



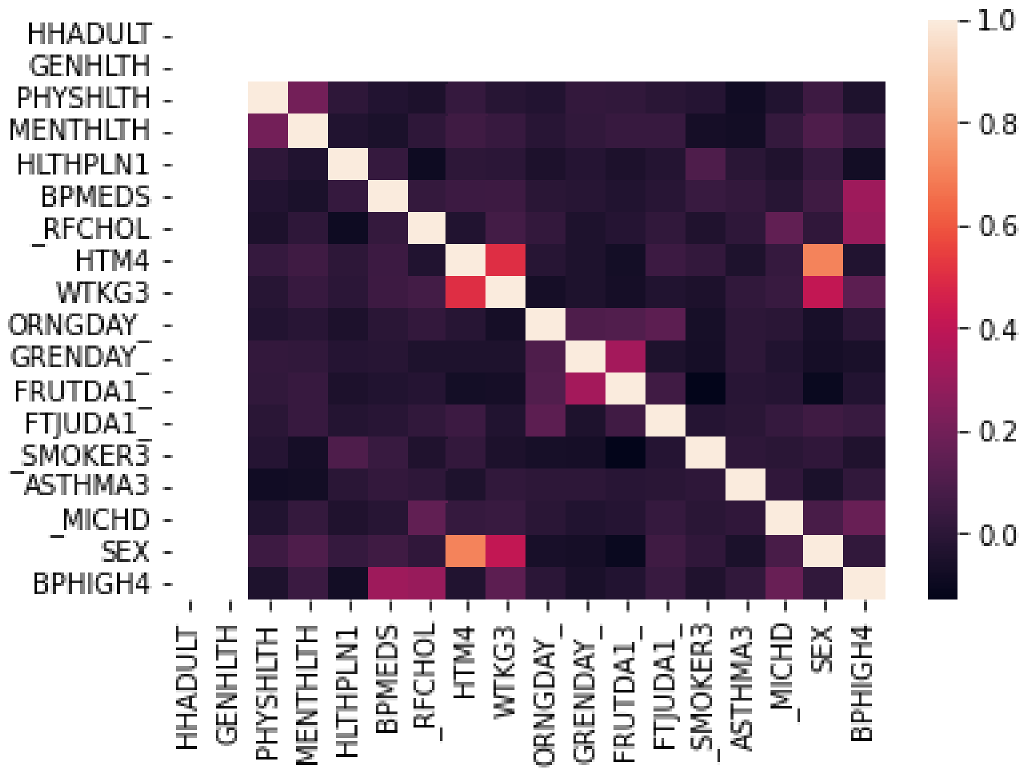
```
[ ]: # GEN HEALTH VS DIABETES
     crosstab_1 = pd.crosstab(diabetes_df85RI['GENHLTH'],
                              diabetes_df85RI['DIABETE3'])
     crosstab_1.plot(kind='bar', stacked=True, figsize = (20,10))
```

```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbe36538bd0>
```



```
[ ]: # Correlation Heatmap
     sns.heatmap(x_train.corr())
```

```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbe369b3210>
```



x_train, x_test, y_train, and y_test

Model 1: KNN

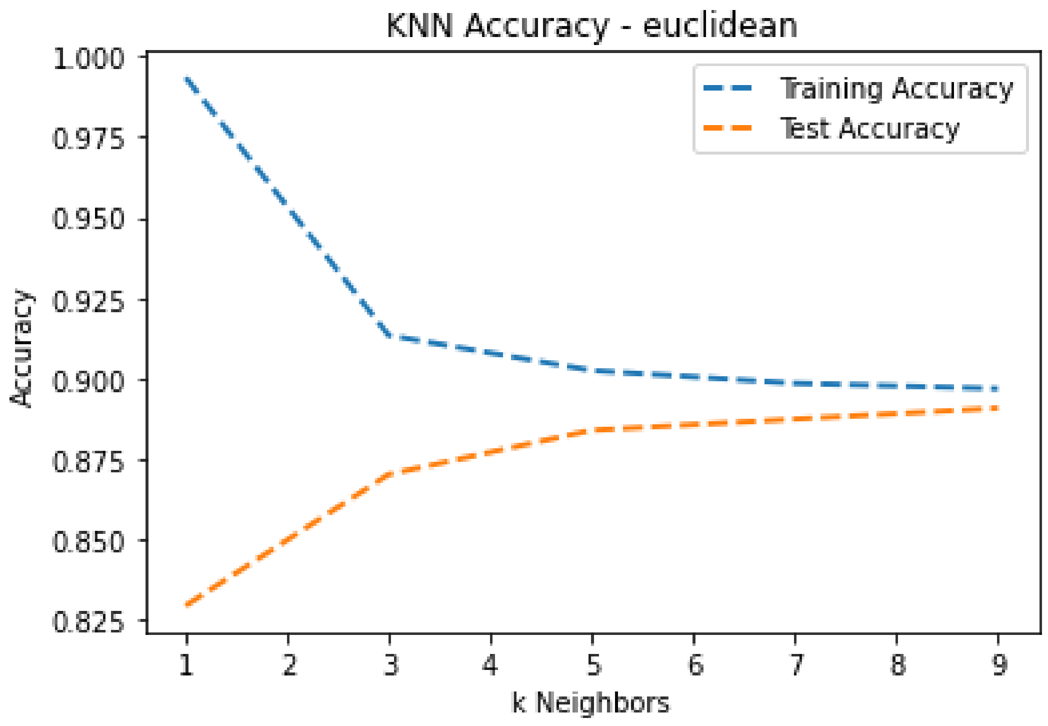
Evaluate KNN Euclidean and Manhattan metric accuracy, to determine which has best results.

```
[ ]: metric='euclidean'
     knn_accuracy = []

     for i in range(1,10,2):
       clsf = KNeighborsClassifier(metric=metric, n_neighbors=i).fit(x_train,␣
      ↪y_train)

       clsf_train_pred = clsf.predict(x_train)
       clsf_test_pred = clsf.predict(x_test)

       knn_accuracy.append({'k values': i,
                            'Training Accuracy': accuracy_score(clsf_train_pred,␣
      ↪y_train),
                            'Test Accuracy': accuracy_score(clsf_test_pred, y_test)})

     # Results
     print("metric:", metric)
     pd.DataFrame(knn_accuracy)
```

```
metric: euclidean
```

```
[ ]:    k values  Training Accuracy  Test Accuracy
     0         1           0.993453       0.829459
     1         3           0.913405       0.870245
     2         5           0.902474       0.883913
     3         7           0.898493       0.887367
     4         9           0.896845       0.890772
```

```
[ ]: # Plot
     print("\n")

     plt.plot(pd.DataFrame(knn_accuracy)['k values'], pd.
      ↪DataFrame(knn_accuracy)['Training Accuracy'], '--', linewidth=2,␣
      ↪label='Training Accuracy')
     plt.plot(pd.DataFrame(knn_accuracy)['k values'], pd.
      ↪DataFrame(knn_accuracy)['Test Accuracy'], '--', linewidth=2, label='Test␣
      ↪Accuracy')
     plt.xlabel('k Neighbors')
     plt.ylabel('Accuracy')
     plt.title('KNN Accuracy - ' + metric)
     plt.legend()
     plt.show()
```



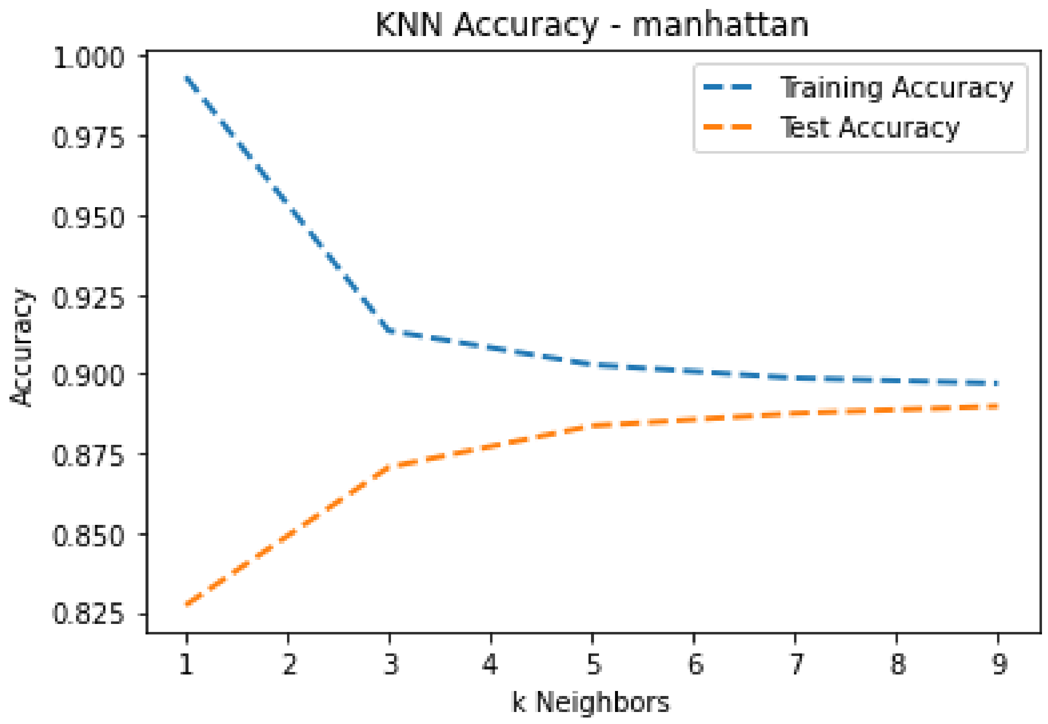
[ ]:
```
metric='manhattan'
knn_accuracy = []

for i in range(1,10,2):
  clsf = KNeighborsClassifier(metric=metric, n_neighbors=i).fit(x_train,␣
 ↪y_train)

  clsf_train_pred = clsf.predict(x_train)
  clsf_test_pred = clsf.predict(x_test)

  knn_accuracy.append({'k values': i,
                       'Training Accuracy': accuracy_score(clsf_train_pred,␣
 ↪y_train),
                       'Test Accuracy': accuracy_score(clsf_test_pred, y_test)})

# Results
```

```
print("metric:", metric)
pd.DataFrame(knn_accuracy)
```

```
metric: manhattan
```

```
[ ]:    k values  Training Accuracy  Test Accuracy
     0         1           0.993453       0.827279
     1         3           0.913552       0.870612
     2         5           0.902878       0.883620
     3         7           0.898670       0.887563
     4         9           0.897041       0.889817
```

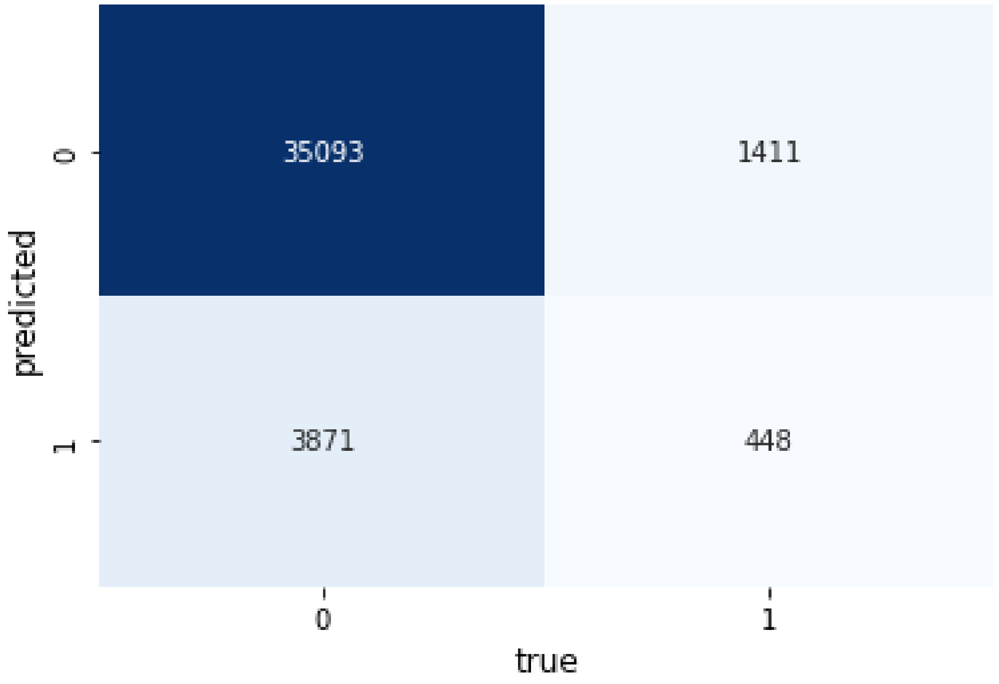
```
[ ]: # Plot
     print("\n")

     plt.plot(pd.DataFrame(knn_accuracy)['k values'], pd.
      ↪DataFrame(knn_accuracy)['Training Accuracy'], '--', linewidth=2,␣
      ↪label='Training Accuracy')
     plt.plot(pd.DataFrame(knn_accuracy)['k values'], pd.
      ↪DataFrame(knn_accuracy)['Test Accuracy'], '--', linewidth=2, label='Test␣
      ↪Accuracy')
     plt.xlabel('k Neighbors')
     plt.ylabel('Accuracy')
     plt.title('KNN Accuracy - ' + metric)
     plt.legend()
     plt.show()
```



```
[ ]: # Manhattan, k = 3
clsf = KNeighborsClassifier(metric='manhattan', n_neighbors=3).fit(x_train,␣
 ↪y_train)

# Predict
knn_pred = clsf.predict(x_test)

# Score
clsf_score = clsf.score(x_test, y_test)

# Accuracy
print('Accuracy: {}'.format(clsf_score))

# Confusion matrix
print("\n")
print("Confusion Matrix:")
nb_matrix = confusion_matrix(y_test, knn_pred)
sns.heatmap(nb_matrix, annot=True, fmt='d', cbar=False, cmap="Blues")
plt.xlabel('true', fontsize=12)
plt.ylabel('predicted', fontsize=12)
plt.show()

# Classification Report
```

```
print("\n")
print("KNN - Manhattan, k = 3")
print("\n")
print(classification_report(y_test, knn_pred))
```

```
Accuracy: 0.8706121549126717


Confusion Matrix:
```

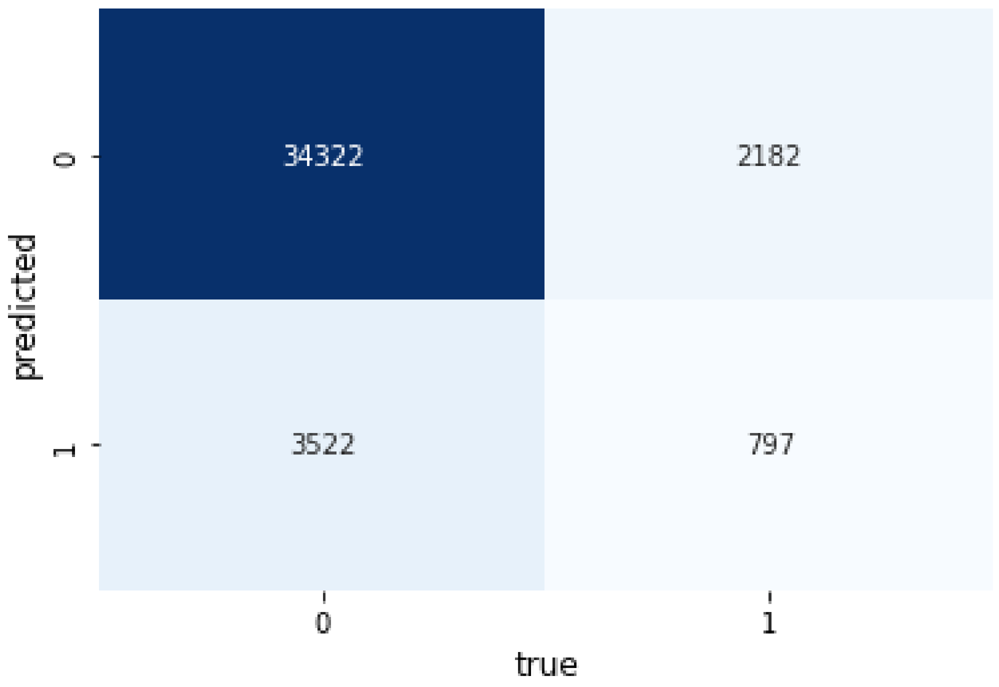


```
KNN - Manhattan, k = 3


              precision    recall  f1-score   support

         0.0       0.90      0.96      0.93     36504
         1.0       0.24      0.10      0.15      4319

    accuracy                           0.87     40823
   macro avg       0.57      0.53      0.54     40823
weighted avg       0.83      0.87      0.85     40823
```

Model 2: Naive Bayes

```
[ ]: # GaussianNB
     clf_nb = GaussianNB()
     clf_nb.fit(x_train, y_train)

     # predict
     clf_nb_pred = clf_nb.predict(x_test)

     # Score
     nb_score = clf_nb.score(x_test, y_test)

     # Accuracy
     print('Accuracy: {}'.format(nb_score))

     # Confusion matrix
     print("\n")
     print("Confusion Matrix:")
     nb_matrix = confusion_matrix(y_test, clf_nb_pred)
     sns.heatmap(nb_matrix, annot=True, fmt='d', cbar=False, cmap="Blues")
     plt.xlabel('true', fontsize=12)
     plt.ylabel('predicted', fontsize=12)
     plt.show()

     # Classification Report
     print("\n")
     print("Naive Bayes - Gaussian")
     print("\n")
     print(classification_report(y_test, clf_nb_pred))
```

```
Accuracy: 0.8602748450628322


Confusion Matrix:
```

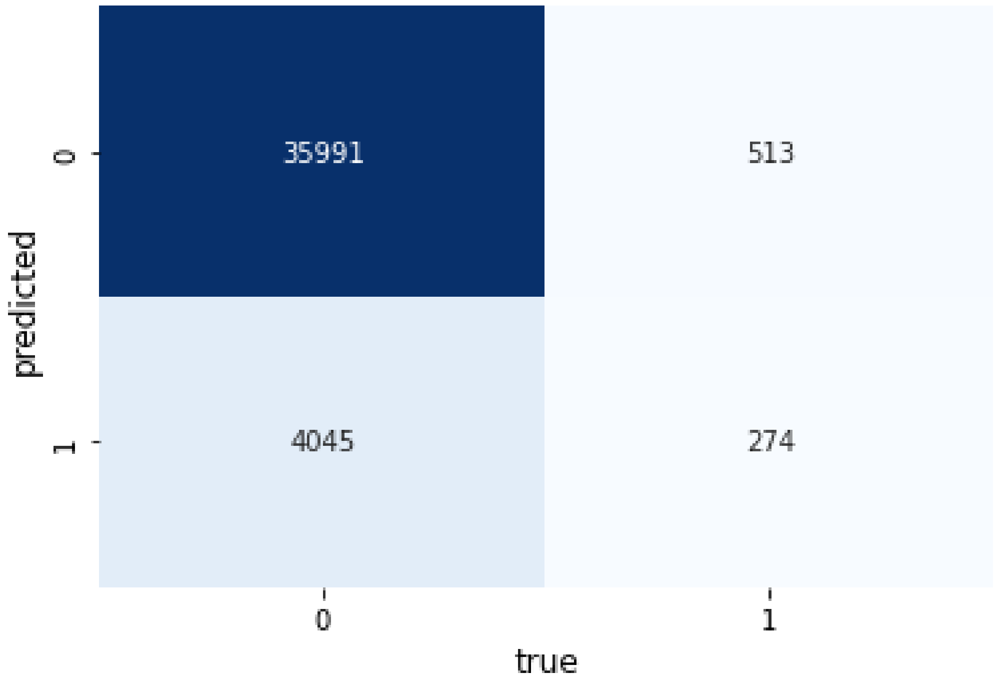


```
Naive Bayes - Gaussian

              precision    recall  f1-score   support

         0.0       0.91      0.94      0.92     36504
         1.0       0.27      0.18      0.22      4319

    accuracy                           0.86     40823
   macro avg       0.59      0.56      0.57     40823
weighted avg       0.84      0.86      0.85     40823
```

[ ]:
```
# BernoulliNB
clf_nb = BernoulliNB()
clf_nb.fit(x_train, y_train)

# predict
clf_nb_pred = clf_nb.predict(x_test)

# Score
nb_score = clf_nb.score(x_test, y_test)
```

```
# Accuracy
print('Accuracy: {}'.format(nb_score))

# Confusion matrix
print("\n")
print("Confusion Matrix:")
nb_matrix = confusion_matrix(y_test, clf_nb_pred)
sns.heatmap(nb_matrix, annot=True, fmt='d', cbar=False, cmap="Blues")
plt.xlabel('true', fontsize=12)
plt.ylabel('predicted', fontsize=12)
plt.show()

# Classification Report
print("\n")
print("Naive Bayes - Bernoulli")
print("\n")
print(classification_report(y_test, clf_nb_pred))
```

```
Accuracy: 0.8883472552237709


Confusion Matrix:
```



```
Naive Bayes - Bernoulli


              precision    recall  f1-score   support

         0.0       0.90      0.99      0.94     36504
         1.0       0.35      0.06      0.11      4319

    accuracy                           0.89     40823
   macro avg       0.62      0.52      0.52     40823
weighted avg       0.84      0.89      0.85     40823
```

Model 3: Random Forest

[ ]:
```
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
```

[ ]:
```
rf.fit(x_train, y_train)
```

[ ]:
```
RandomForestRegressor(n_estimators=1000, random_state=42)
```

[ ]:
```
predictions = rf.predict(x_test)
```

[ ]:
```
# Calculate the absolute errors
errors = abs(predictions - y_test)
```

[ ]:
```
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 0.19 degrees.
```

[ ]:
```
# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / y_test)
```

[ ]:
```
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy,2), '%')
```

```
Accuracy: -inf %
```

[ ]:
```
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
# Pull out one tree from the forest
tree = rf.estimators_[5]
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
```

```
# Pull out one tree from the forest
tree = rf.estimators_[5]
# Export the image to a dot file
export_graphviz(tree, out_file = 'tree.dot', rounded = True, precision = 1)
# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('tree.dot')
# Write graph to a png file
graph.write_png('tree.png')
```

[ ]:
```
# Limit depth of tree to 3 levels
rf_small = RandomForestRegressor(n_estimators=10, max_depth = 3)
rf_small.fit(x_train, y_train)
# Extract the small tree
tree_small = rf_small.estimators_[5]
# Save the tree as a png image
export_graphviz(tree_small, out_file = 'small_tree.dot', rounded = True,␣
 ↪precision = 1)
(graph, ) = pydot.graph_from_dot_file('small_tree.dot')
graph.write_png('small_tree.png')
```

Model 4: Logistic Regression

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

[ ]:
```
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
```

[ ]:
```
LogisticRegression()
```

[ ]:
```
y_pred = logreg.predict(x_test)
print('Accuracy of logistic regression classifier on test set: {:.2f}'.
 ↪format(logreg.score(x_test, y_test)))
```

```
Accuracy of logistic regression classifier on test set: 0.89
```

[ ]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

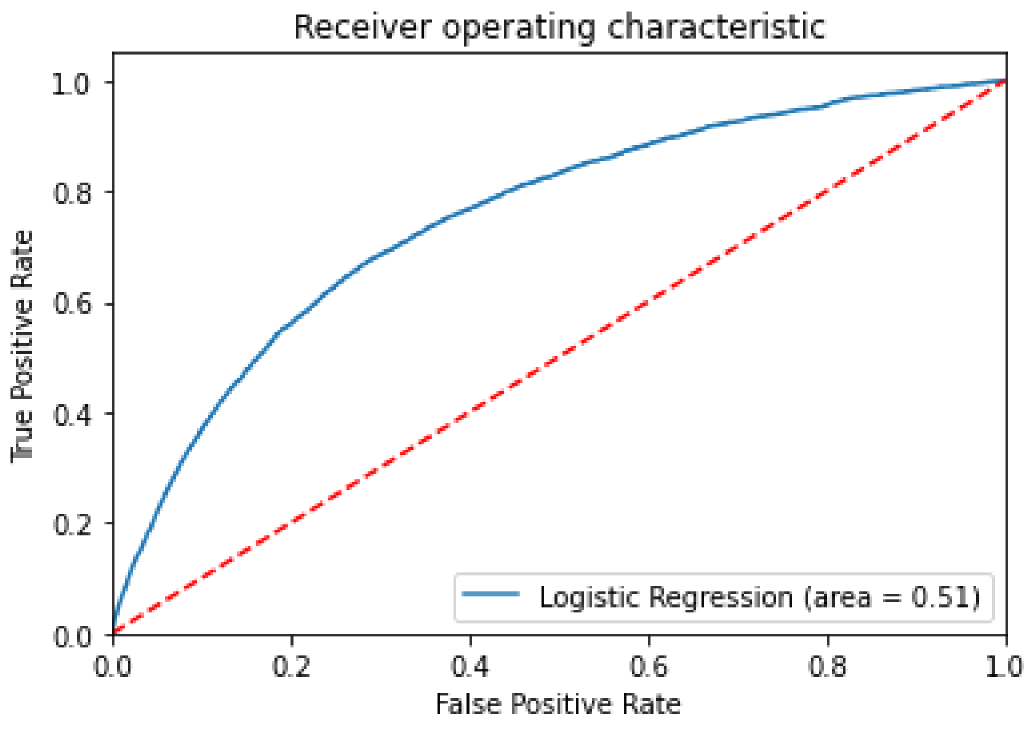
```
[[36447    57]
 [ 4252    67]]
```

[ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support
```

```
         0.0       0.90      1.00      0.94     36504
         1.0       0.54      0.02      0.03      4319

    accuracy                           0.89     40823
   macro avg       0.72      0.51      0.49     40823
weighted avg       0.86      0.89      0.85     40823
```

[ ]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, logreg.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



```
[ ]: %%capture
     !wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
     from colab_pdf import colab_pdf
     colab_pdf('504_FinalProject_Group4.ipynb')
```