Table of Contents

INTRODUCTION

A health care insurance company is interested in determining the average cost of each new customer (patient). They want to examine existing health conditions to determine potential risks and/or new conditions in order to analyze the potential associated costs. They will utilize hierarchical condition category (HCC) coding, which is a risk-adjustment model referring to the degree of illness originally designed with this specific purpose in mind.

HCC relies on ICD-10 (International Classification of Diseases, Tenth Revision) coding to assign risk scores to patients. ICD-10 codes specifically refer to the patient's illness. Each HCC is mapped to an ICD-10 code. Insurance companies use HCC coding, along with demographic factors (such as age and gender), to assign patients a risk adjustment factor (RAF) score. Insurances can use a patient's RAF score to predict possible costs using algorithms. For example, a patient with minor health conditions could be expected to have average medical costs for a given time. However, a patient with multiple conditions ranging in severity would be more likely to have higher health care utilization and costs.

HCC coding helps convey patient complexity and paint a picture of the patient (past, present, and potential future). In addition to helping predict the utilization of health care resources, RAF scores are used to influence risk adjust quality and cost metrics. By accounting for variations in patient complexity, quality and cost performance can be more accurately measured.

PROBLEM STATEMENT

How can a health insurance company predict if their customers will have Essential (Primary) Hypertension disease (I10) and/or Peripheral vascular disease (I739) in the coming year (2020) with a minimum accuracy rate of 85% based on the 2018-2019 ICD patient history?

DATASETS

We will be using several datasets from 2018-2019 for our analysis. (The data was sourced from Kaggle.) The attributes of each data are detailed below.

ADMISSIONS DATA

- **member_id** - patients' ID
- **admission_date** - date when hospital admission occurred
- **discharge_date** - date when member was discharge from the hospital
- **admission_type** - hospital admission type
- **icd_code** - ICD code (diagnosis) used for hospital admission
- **is_readmission** - 1 if member was in the hospital in last 30 days, otherwise 0

- **er_to_inp_admission_transfer** - 1 if member had inpatient (INP) and ER (emergency) hospital admission same day, otherwise 0
- **days_to_prev_admission** - number of days since last admission

Disease Data

- **year_of_service** - year when ICD/HCC was detected by doctors
- **icd_code** - ICD code (disease)
- **icd_chronic_or_acute** - chronic/acute disease type
- **hcc_code** - HCC coding for ICD
- **hcc_chronic_or_acute** - chronic/acute HCC type

Labs Data

- **date_of_service** - date when some service was provided to patients
- **loinc_code** - laboratory test code
- **performed_test_name** - laboratory test name
- **abnormal_code** - flag that indicates low (L) or high (H) lab results
- **result_value** - result value
- **normal_low_value_numeric** - the lower bound for reference lab result
- **normal_high_value_numeric** - the upper bound for reference lab result

Patients Data

- **date_of_service** - date when some service was provided to patients
- **patient_age** - patient's age
- **patient_gender** - patient's gender
- **patient_age** - patient's age
- **dual_status** - flag that refers to Medicaid/Medicate insurance programs
- **patient_age** - patient's age
- **insurance_company** - company providing medical insurance
- **insurance_type** - insurance type
- **pbp_type** - member's risk group
- **county** - member's county
- **city** - member's city

Prescriptions Data

- **date_filled** - date when prescription was filled for the member
- **ndc_number** - unique drug ID
- **days_supply** - number of days for which the prescription was filled
- **drug_name** - drug name
- **metric_quantity** - drug dosage
- **new_or_refill** - 1 if a new drug was prescribed, otherwise 0

- **hcc_9** - flag that identifies that member has HCC 9. Similar logic should be used for other **hcc_N** columns

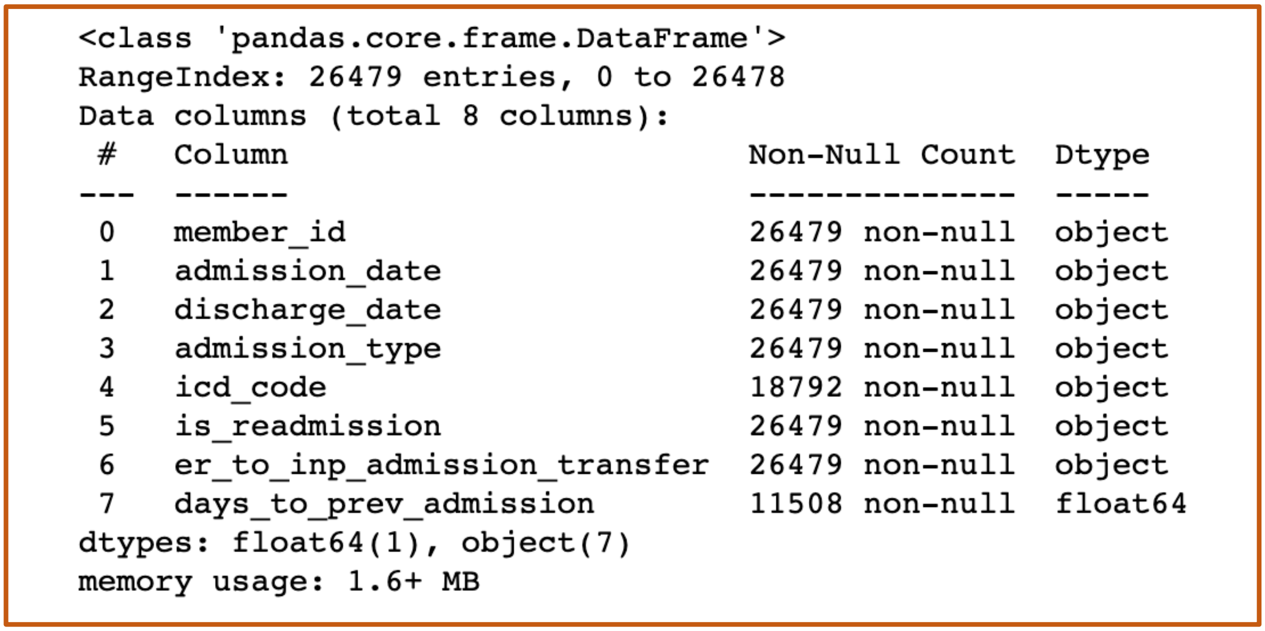
DATA WRANGLING

We transformed data from one "raw" data form into a different format to make it more appropriate and valuable for our analysis. In this step we did the following:

- Fix the abnormalities
- Impute the missing values
- Drop the records/columns that are irrelevant for the analysis

Admissions Data

We print information about the admissions data including the index dtype and columns, non-null values, and memory usage.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26479 entries, 0 to 26478
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   member_id                    26479 non-null  object
 1   admission_date               26479 non-null  object
 2   discharge_date               26479 non-null  object
 3   admission_type               26479 non-null  object
 4   icd_code                     18792 non-null  object
 5   is_readmission               26479 non-null  object
 6   er_to_inp_admission_transfer 26479 non-null  object
 7   days_to_prev_admission       11508 non-null  float64
dtypes: float64(1), object(7)
memory usage: 1.6+ MB
```

Figure 1

We visualize data sparsity in our dataset.

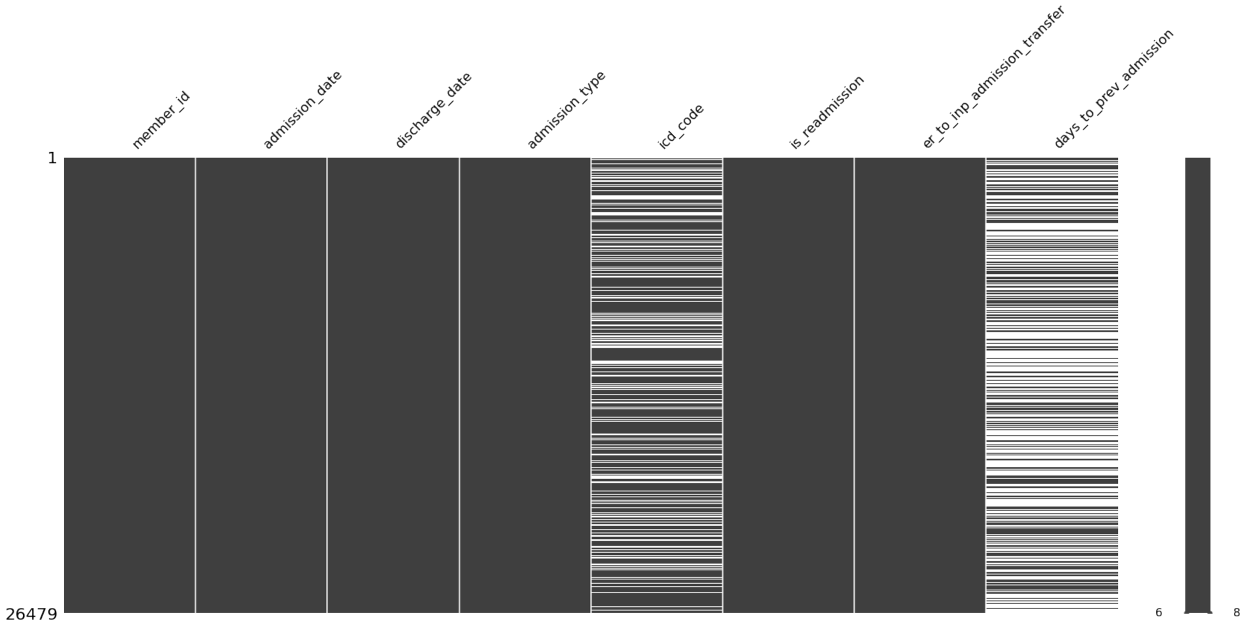


Figure 2

We observe that **days_to_prev_admission** has the most null values. We have no way of guessing these values. We don't impute these values until further analysis. Second most null values are in the **icd_code** column. These icd codes were entered at the admissions. We saved the codes entered for each member id in a dictionary for future analysis. We drop the **icd_code** column all together since we have the records of icd codes in the disease data.

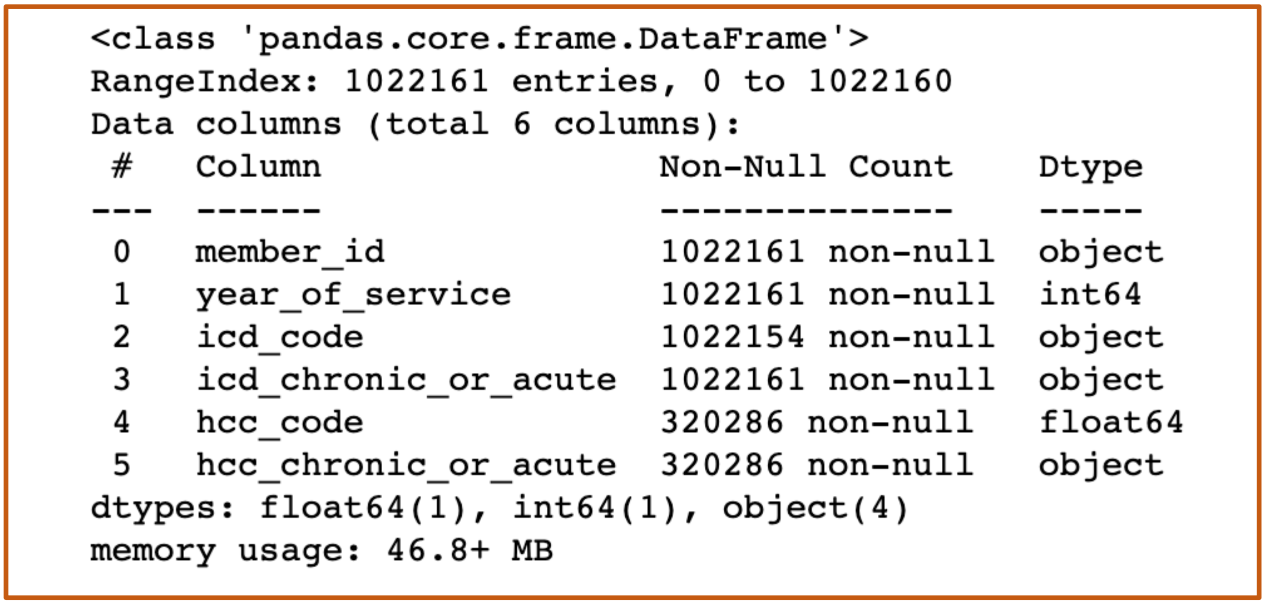
To use the date time specific functions in Pandas, we convert **admission_date** and **discharge_date** into an appropriate format. We notice that there are records from year 2020 aside from year 2018 and 2019 in our dataframe. However, we only consider records from 2018 and 2019 in our analysis. We eliminate any records that do not meet the criteria. In some records values entered for **admission_date** and **discharge_date** were abnormal in the sense that admission's date were later than the discharge date of the visit. We interchange these values for each abnormal record.

We create a new column called **nth_visit** which stores the rank of the visits per patient. If a patient's record is from the third visit over the period, then we have entered 3 in this column.

The column **visit_duration** is created to keep the records of the duration of each visit. This column is entered 0 if admission and discharge dates are made on the same day. To distinguish these patients from the patients who didn't make the visit, we created another column called **visit_duration_updated** which equals **visit_duration** plus 1.

Disease Data

We print information about disease data including the index dtype and columns, non-null values, and memory usage.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1022161 entries, 0 to 1022160
Data columns (total 6 columns):
 #   Column                Non-Null Count    Dtype
---  ------                --------------    -----
 0   member_id             1022161 non-null  object
 1   year_of_service       1022161 non-null  int64
 2   icd_code              1022154 non-null  object
 3   icd_chronic_or_acute  1022161 non-null  object
 4   hcc_code              320286 non-null   float64
 5   hcc_chronic_or_acute  320286 non-null   object
dtypes: float64(1), int64(1), object(4)
memory usage: 46.8+ MB
```

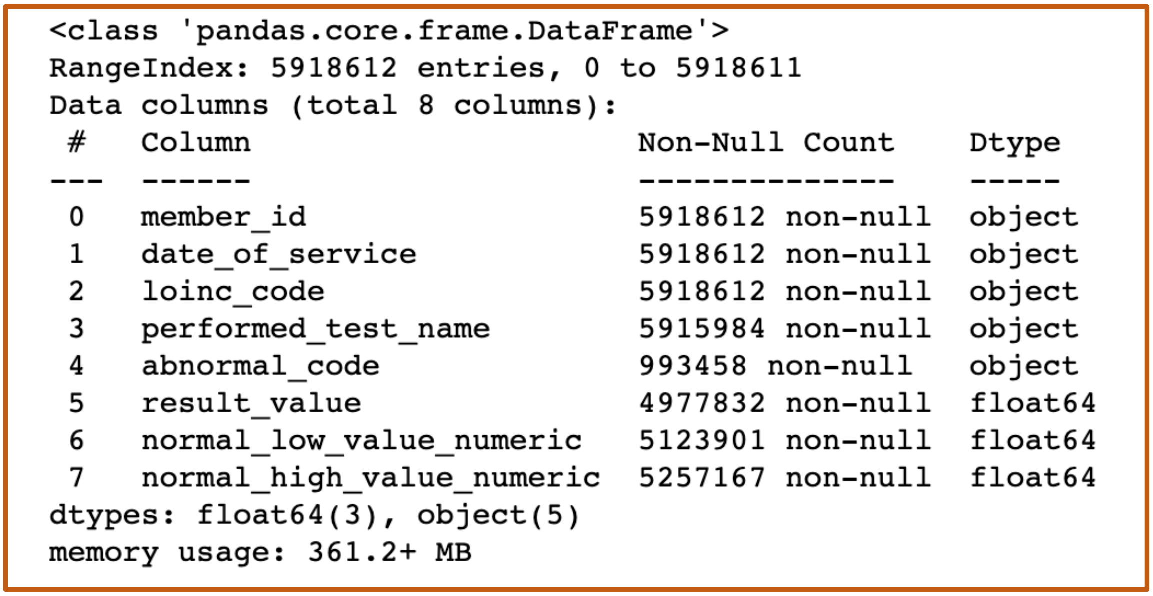
Figure 3

We have a significant number of missing values in **hcc_code***.* However, we drop this column. The hcc codes are determined based on the icd codes. Including both **icd_code** and **hcc_code** can cause issues related to correlation. Thereby it is sufficient for us to use icd codes only. We also have many missing values in **hcc_chronic_or_acute**. This column is dropped since it is associated to **hcc_code**.

Using the method unique() we spot some records from 2021 apart from 2018 and 2019. These records were eliminated.

LABS DATA

We print information about labs data including the index dtype and columns, non-null values, and memory usage.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5918612 entries, 0 to 5918611
Data columns (total 8 columns):
 #   Column                      Non-Null Count    Dtype  
---  ------                      --------------    -----  
 0   member_id                   5918612 non-null  object 
 1   date_of_service             5918612 non-null  object 
 2   loinc_code                  5918612 non-null  object 
 3   performed_test_name         5915984 non-null  object 
 4   abnormal_code               993458 non-null   object 
 5   result_value                4977832 non-null  float64
 6   normal_low_value_numeric    5123901 non-null  float64
 7   normal_high_value_numeric   5257167 non-null  float64
dtypes: float64(3), object(5)
memory usage: 361.2+ MB
```

Figure 4

We have over 5 million records in this dataset. It is challenging to compare all these quantities in our table. We visualize the missing values in order to better analyze.

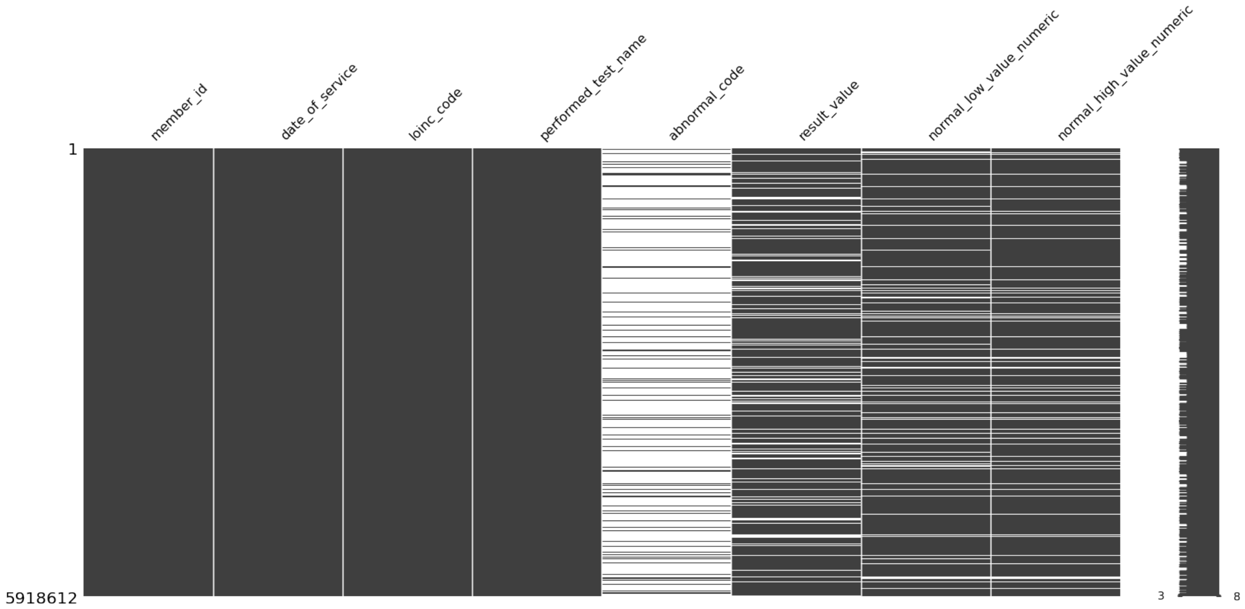


Figure 5

We find the most missing values in **abnormal_code**. We impute this column based on **result_value**, **normal_low_value_numeric,** and **normal_high_value_numeric**.

Furthermore, the values entered in this column were not consistent; there were multiple distinct values for the same conclusion. For instance, “H”, “HIGH”, “ABOVE”, “HH”, “+” and “>” were entered to encode the results that are higher than the normal high values of the lab tests. We replace all these values with ‘H’. All the abnormalities of the same type are fixed.

Additionally**, normal_low_value_numeric** and **normal_high_value_numeric** contain many null values. We fill the missing values corresponding to some well-known lab tests in these columns. Apart from that, we don’t impute the missing values for these columns since we have no way of knowing these values and we don’t want to make assumptions that might be misleading. We don’t impute the missing values in **result_value** for the same reason.

The result values are significantly bigger than both normal low and normal high values. The average of **result_value** is 1,312,4710. On the other hand, the average of **normal_low_value_numeric** and **normal_high_value_numeric** are ~54.5 and ~213.7, respectively. After some careful examination we saw that each result value multiplied by 1000 before entered, which caused the issue. We divided each result value by 1000 to correct this abnormality.

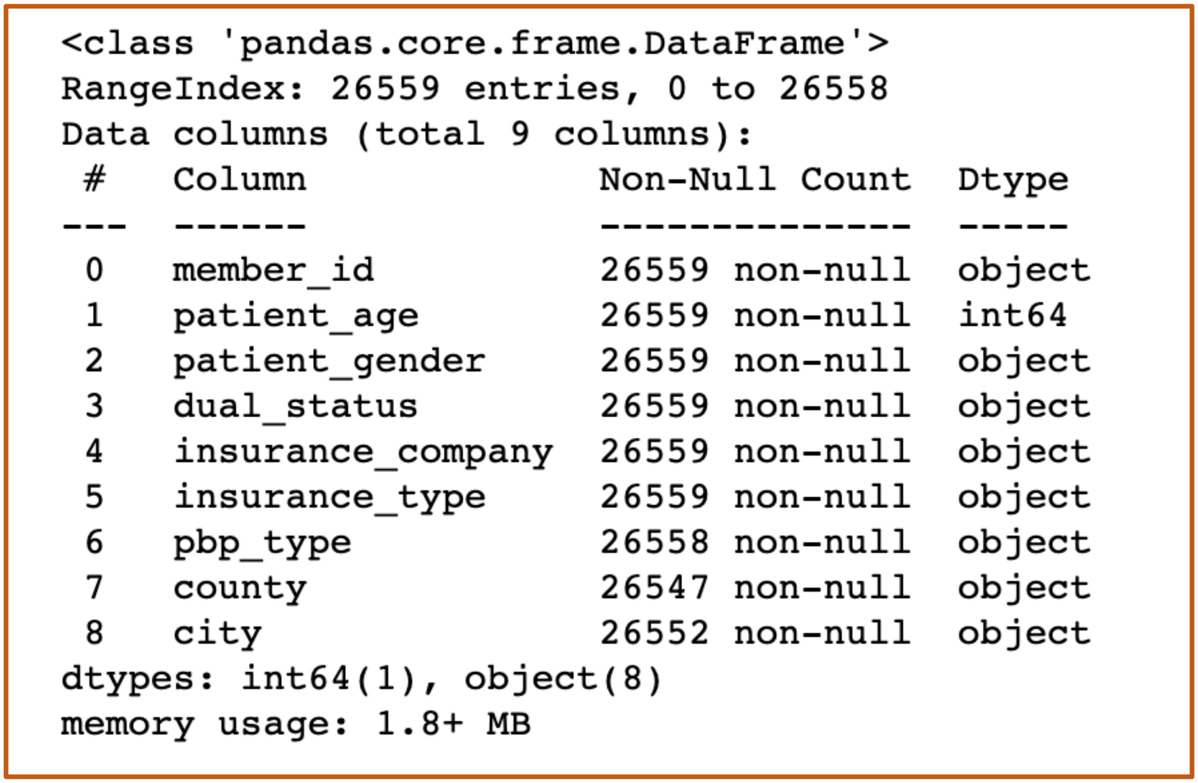
The names of the administrated lab tests are recorded in column **performed_test_name**. We will drop this column since column **loinc_code** contains codes for the performed laboratory tests and this column is sufficient to encode the lab tests.

We drop **normal_low_value_numeric**, **normal_high_value_numeric**, and **date_of_service** as these columns are not indicative for a patient to have a disease.

We add a new column called **num_of_tests** to contain the total number of lab tests administered for each patient.

Patients Data

We print information about patients data including the index dtype and columns, non-null values, and memory usage.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26559 entries, 0 to 26558
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   member_id          26559 non-null  object
 1   patient_age        26559 non-null  int64 
 2   patient_gender     26559 non-null  object
 3   dual_status        26559 non-null  object
 4   insurance_company  26559 non-null  object
 5   insurance_type     26559 non-null  object
 6   pbp_type           26558 non-null  object
 7   county             26547 non-null  object
 8   city               26552 non-null  object
dtypes: int64(1), object(8)
memory usage: 1.8+ MB
```

Figure 6

As it is shown in the info table this dataset has a few missing values. None are filled.

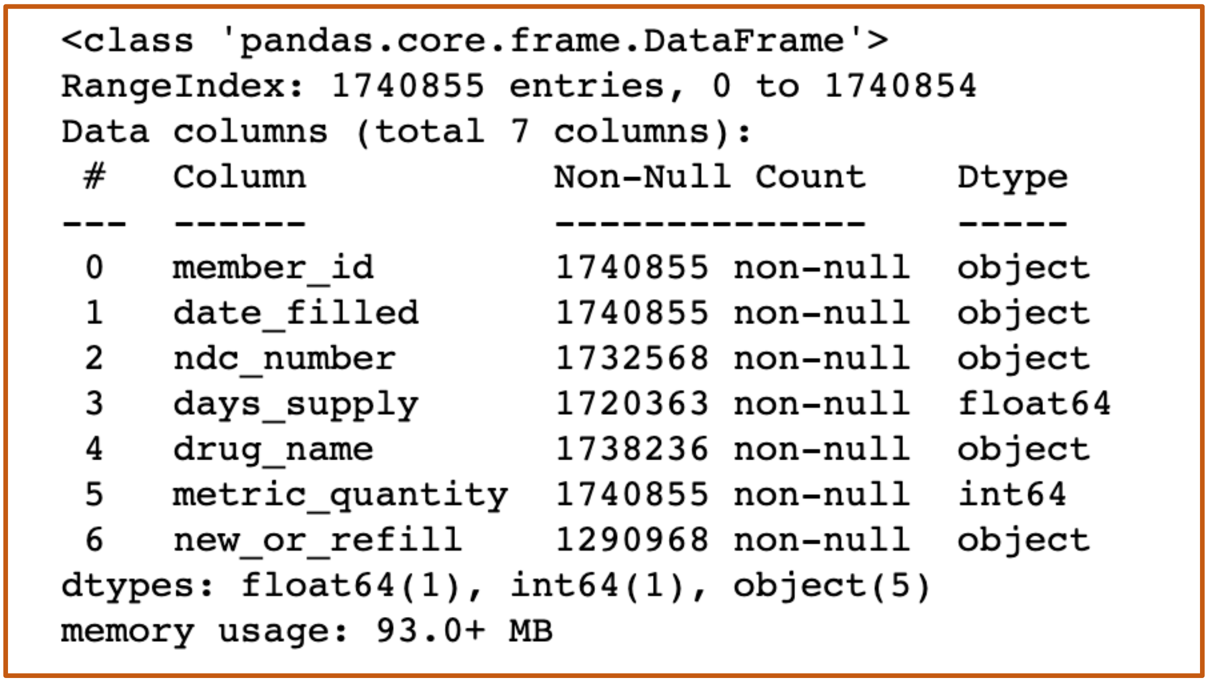
No abnormalities are found.

For the sake of patients' privacy, most values in this dataset were encoded with strings of many random characters. We change these values for readability purposes.

All columns except **member_id** and **patient_age** are converted into categorical dtypes. We subsequently define dummy variables for these columns.

PRESCRIPTIONS DATA

We print information about prescriptions data including the index dtype and columns, non-null values, and memory usage.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1740855 entries, 0 to 1740854
Data columns (total 7 columns):
 #   Column           Non-Null Count    Dtype
---  ------           --------------    -----
 0   member_id        1740855 non-null  object
 1   date_filled      1740855 non-null  object
 2   ndc_number       1732568 non-null  object
 3   days_supply      1720363 non-null  float64
 4   drug_name        1738236 non-null  object
 5   metric_quantity  1740855 non-null  int64
 6   new_or_refill    1290968 non-null  object
dtypes: float64(1), int64(1), object(5)
memory usage: 93.0+ MB
```

Figure 7

We visualize data sparsity in our dataset.

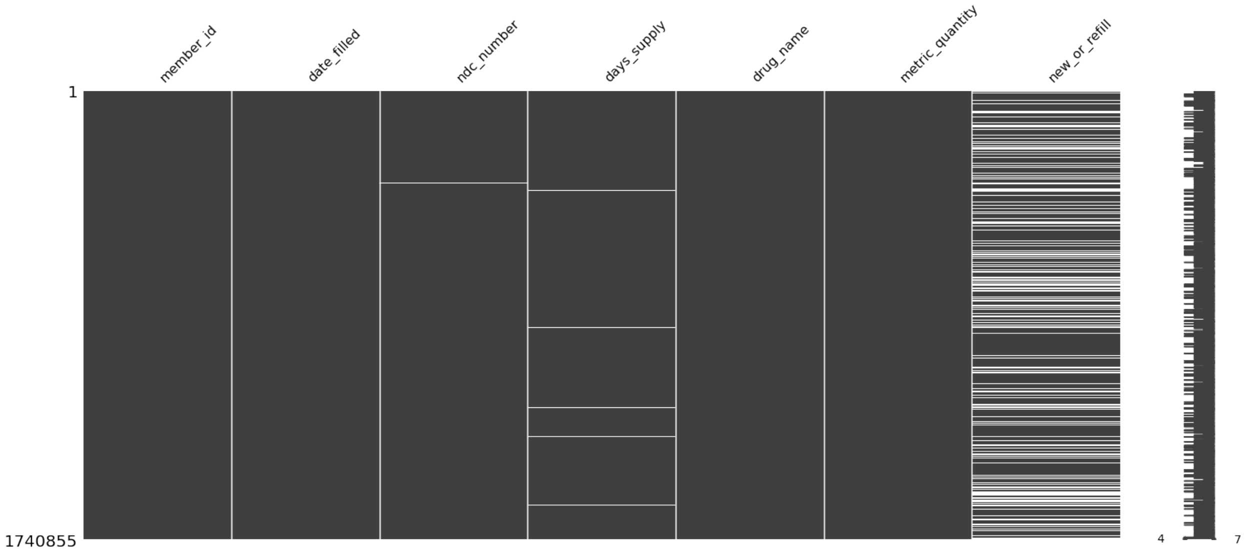


Figure 8

We find the most missing values in **new_or_refill**. We don’t impute the missing values until further analysis. We convert this column into categorical dtype since it contains only 2 unique values: “N” and “R”, to indicate new and refill, respectively. We define dummy variables for it since we eventually have a dataframe where we have only one row for each member id.

Some values in column **metric_quantity** were entered as 0 which is obscure given the fact that metric quantity represents drug dosage and therefore it must be strictly positive. We replace these values with null since we don't have adequate info.

We add a new column called **drug_quantity_rate** to record **days_supply** divided by **metric_quantity**. This will indicate how much medicine was given to the patient for each medicine prescribed.

We drop **days_supply** and **metric_quantity** once we extract the useful info from these columns.

The column **ndc_number** contains the unique drug id which is sufficient to identify prescribed medicine. We drop **drug_name** since it is no longer needed. We also drop **date_filled** column since we will not consider the dates in our model.

SUBSET ALL DATASETS

The size of our data is very large. This can cause some issues with the implementations of our algorithms given the fact that we use a single laptop which limits our computational power. We subset all the dataframes to make sure that we will not run into an issue.

We filter the disease dataset with respect to the patients who have illnesses 'I10' and/or 'I739'. Then we pick 2000 records from these records randomly. We filter all other dataframes based on the member ids that appear in these 2000 records.

ONE RECORD PER MEMBER ID

We convert each dataframe into a new dataframe where we have exactly one record per patient/member id. This process is challenging since we would like to maintain the original info.

ADMISSIONS DATA

We consider each visit without any aggregation. The maximum number of visits among all patients is 24. We create 24 columns called v1, v2, … v24 where we keep the records whether the patient had a visit or not. For instance, a patient with only 5 visits will have 1 for all columns v1 through v5 and 0 in all other columns v6 through v24.

We also create the columns **vi_re_adm**, **vi_er_to_inp**, **vi_days_to_prev**, **vi_len**, **vi_len_pos**, **vi_ER**, **vi_INP**, …, **vi_INP REHAB** for each i=1, 2, …, 24 to include all the information for the ith visit recorded in the admissions dataset.

We accomplished the listed tasks above using two functions that we write. The new dataframe has 282 columns.

Disease Data

We have 4 columns in the resulting dataframe for the disease data: **icd_I10**, **icd_I739**, **icd_I10_chronic**, **icd_I739_chronic**. If a patient has disease "I10", we enter 1 for **icd_I10**, we enter 0 otherwise. If the disease is chronic, we enter 1 for **icd_I10_chronic**, we enter 0 otherwise. Columns **icd_I739** and **icd_I739_chronic** are filled similarly.

Labs Data

We consider each lab test performed without any type of aggregation. We create one column for each lab test called **lab_#** where # is the loinc code of the lab test. This column is filled with 1 if a patient has done the test, and the column is filled with 0 otherwise. Another column called **lab_#_res** is created, which is associated to this column that stores the test's result. The resulting dataframe produces 3003 columns in total.

Patients Data

We already have one record per patient in the dataframe for Patients data. All we need to do is get dummy variables for the columns whose dtypes are categorical. We get 266 columns in total.

Prescriptions Data

We consider each medicine prescribed without any type of aggregation. We create one column for each medicine called **ndc#_number**, where number is the ndc number of the medicine. This column is filled with 1 if a patient was prescribed the drug and filled with 0 otherwise. Two columns called **ndc#_number_refill** (the medicine's status in terms of whether it is new or refill) and **ndc#_number_rate** (the medicine's quantity rate) are created. We have 21824 columns in total in the resulting dataframe.

DATAFRAMES MERGED

In continuation of the previous step, we merge all the dataframes that contain one record per member id. This provides us with a dataframe that has 30,057 attributes.

EXPLORATORY DATA ANALYSIS (EDA)

This step is a critical process of performing initial examinations on our data as we discover patterns, point out abnormalities and test hypotheses. We use summary statistics and visual representations.

DATA BY ADMISSION TYPE

The horizontal bar plot below shows that ER and INP are the admission types that appear the most. Both types show around 800 times. OTH, which shows around 300 times, is the admission type that follows next.

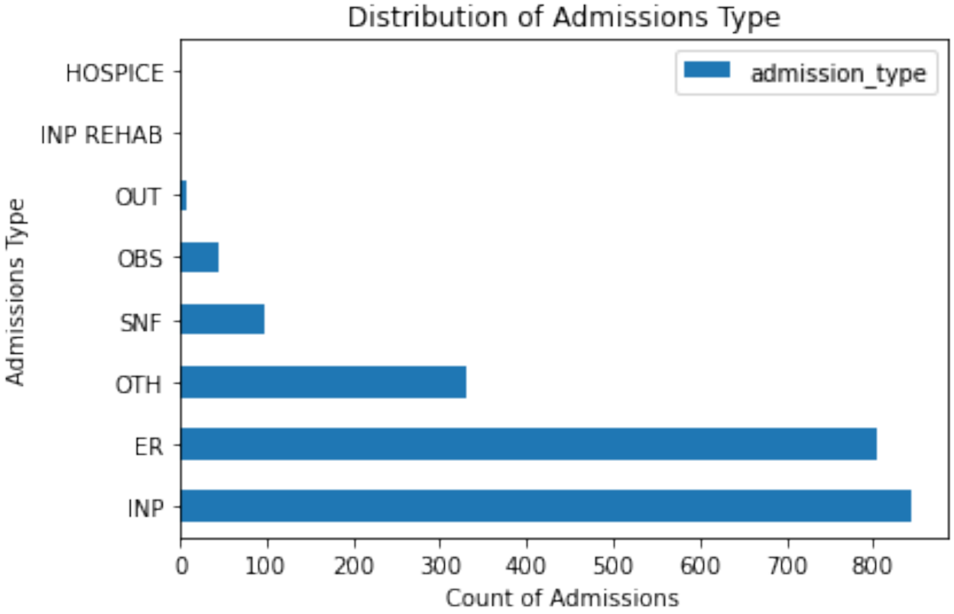
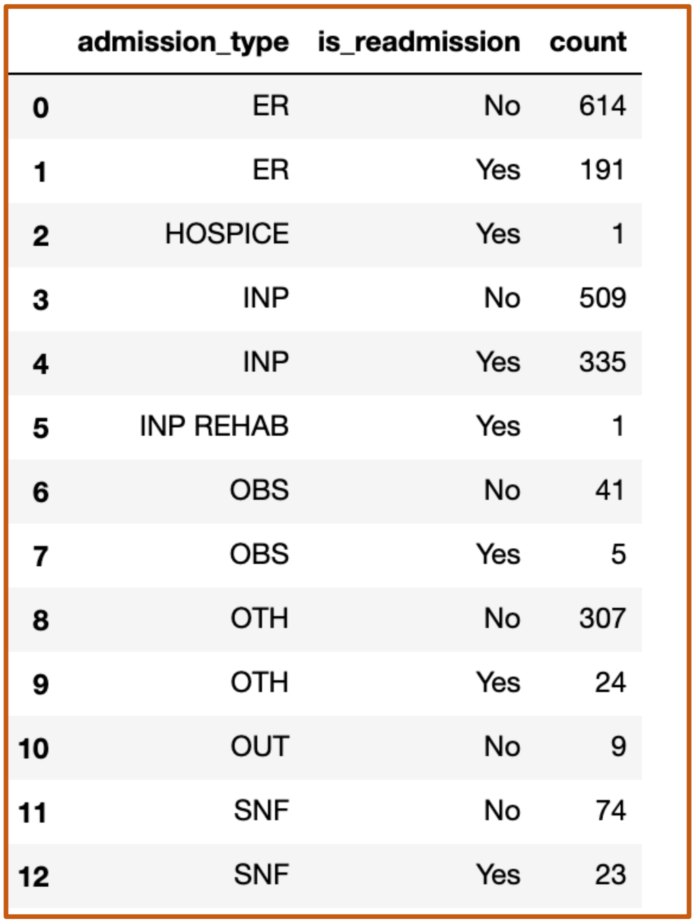


Figure 9

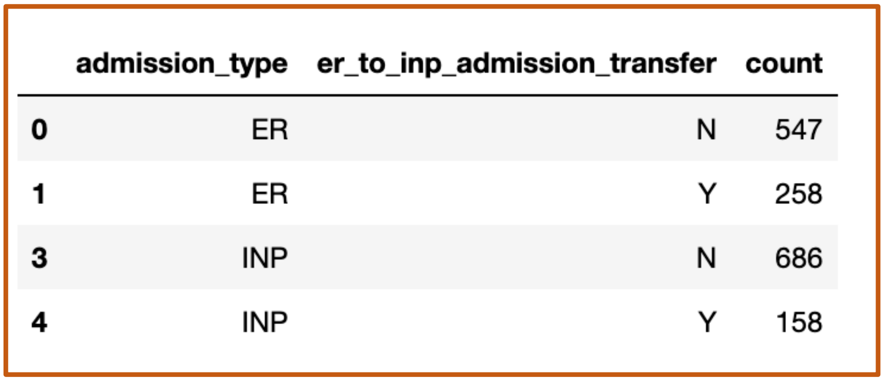
The next figure implies that approximately 25% of ER visits are readmissions. On the other hand, around 40% of INP admissions are readmissions. These two percentages indicate that the portion of hospital visits which are made by the same group of patients is significant.



	admission_type	is_readmission	count
0	ER	No	614
1	ER	Yes	191
2	HOSPICE	Yes	1
3	INP	No	509
4	INP	Yes	335
5	INP REHAB	Yes	1
6	OBS	No	41
7	OBS	Yes	5
8	OTH	No	307
9	OTH	Yes	24
10	OUT	No	9
11	SNF	No	74
12	SNF	Yes	23

Figure 10

258 out of 805 of emergency admissions (ER type) transferred to inpatient service (INP type). Additionally, 158 out of 844 inpatient admissions were made right after an emergency admission. See below for details.



	admission_type	er_to_inp_admission_transfer	count
0	ER	N	547
1	ER	Y	258
3	INP	N	686
4	INP	Y	158

Figure 11

One can see the comparison of columns **visit_duration_updated** and **prev_days_to_admission** below. Admissions of type SNF have the greatest average number of days in visit durations. This is expected; generally, patients who are admitted to SNF (skilled nursing facilities) are recovering from surgery, injury, or acute illness which take days. Also, worth noting that OBS (observational stay) admissions type has the greatest average number of days in days since the previous admission. We don’t have an educated guess on this fact.

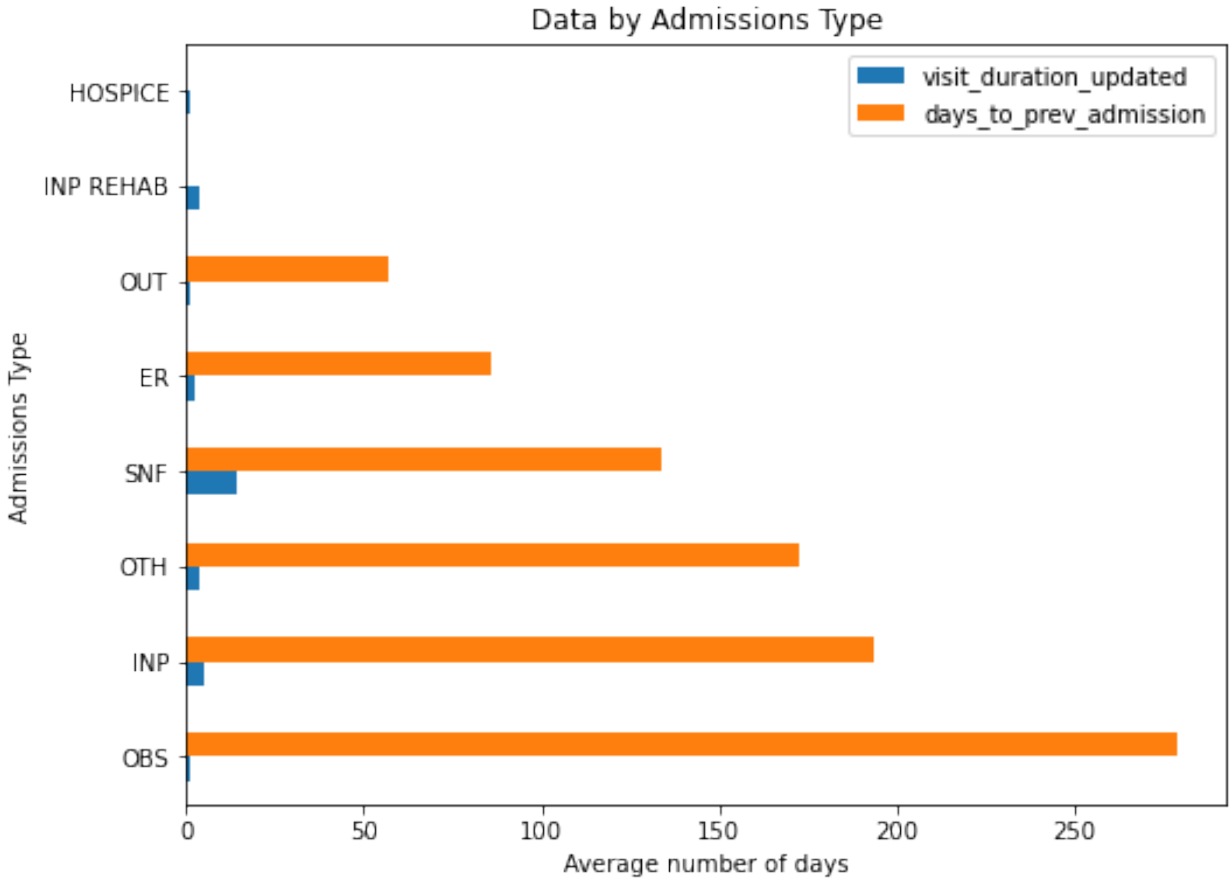


Figure 12

Age Distribution By Gender

Age distribution by gender is below. The distributions for both genders are alike and similar to a Gaussian distribution. We also observe that there are more patients under 30 in women than in men.

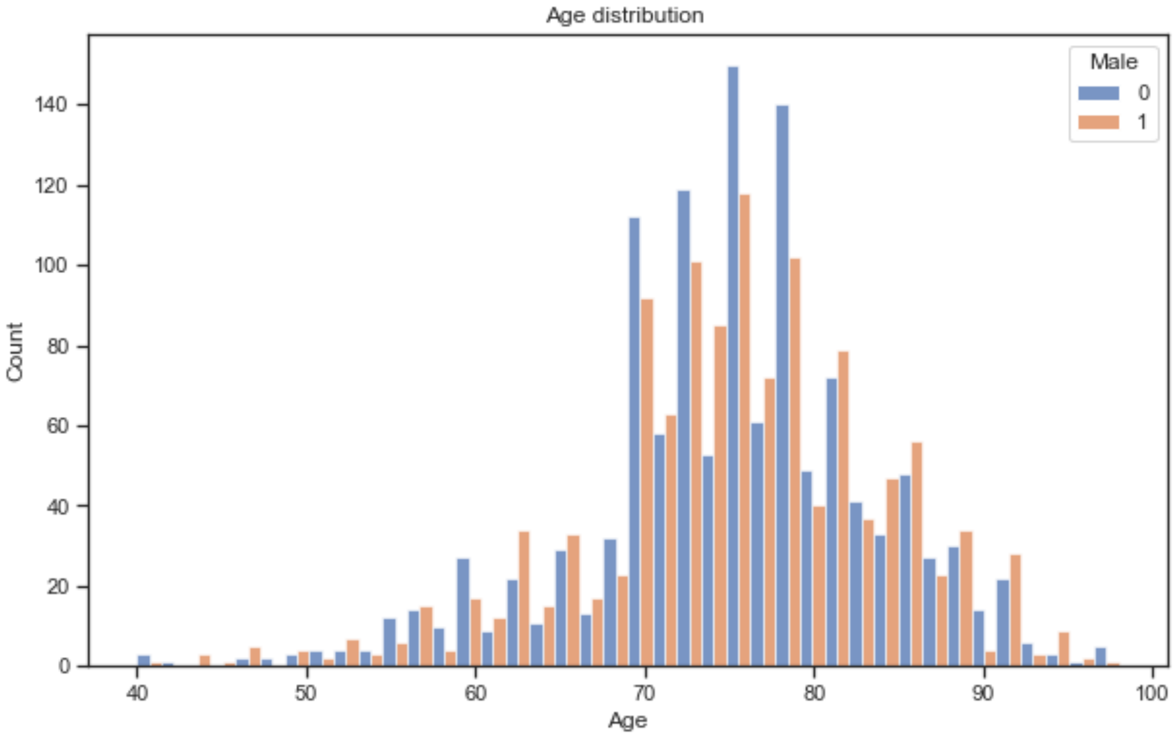


Figure 13

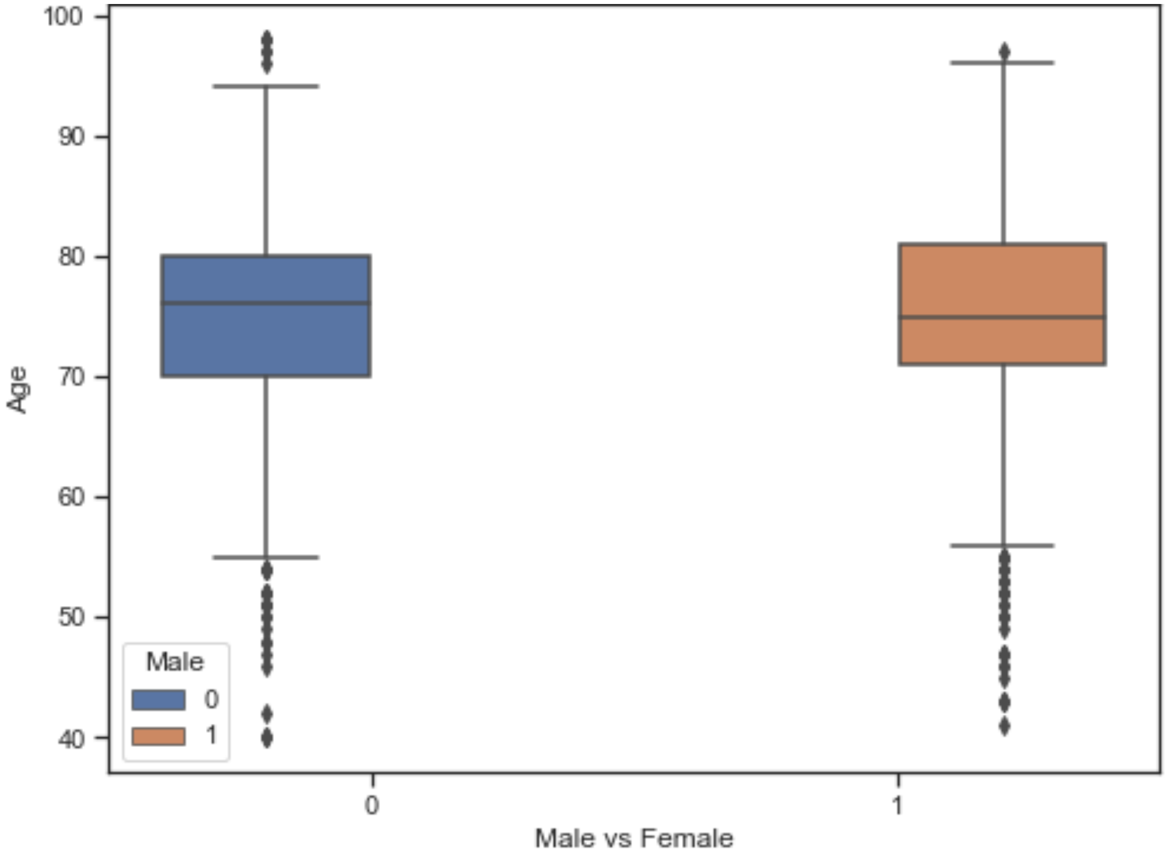
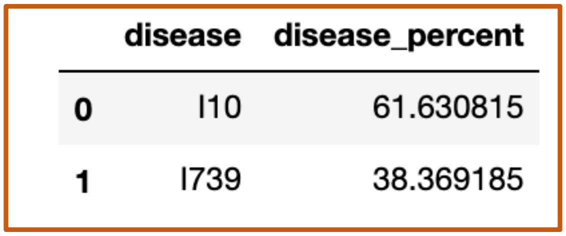


Figure 14

The box and whisker plot above (Figure 14) shows that the interquartile range of age distribution is between 70 and 80 for both men and women. However, the median of men's age is about 75 and it is smaller than the median of women's age. Also, women have more outliers above 90 than men.

Comparison of the Diseases

We have the percentages of the diseases in the following table which is also illustrated in Figure 15.



	disease	disease_percent
0	I10	61.630815
1	I739	38.369185

Figure 15

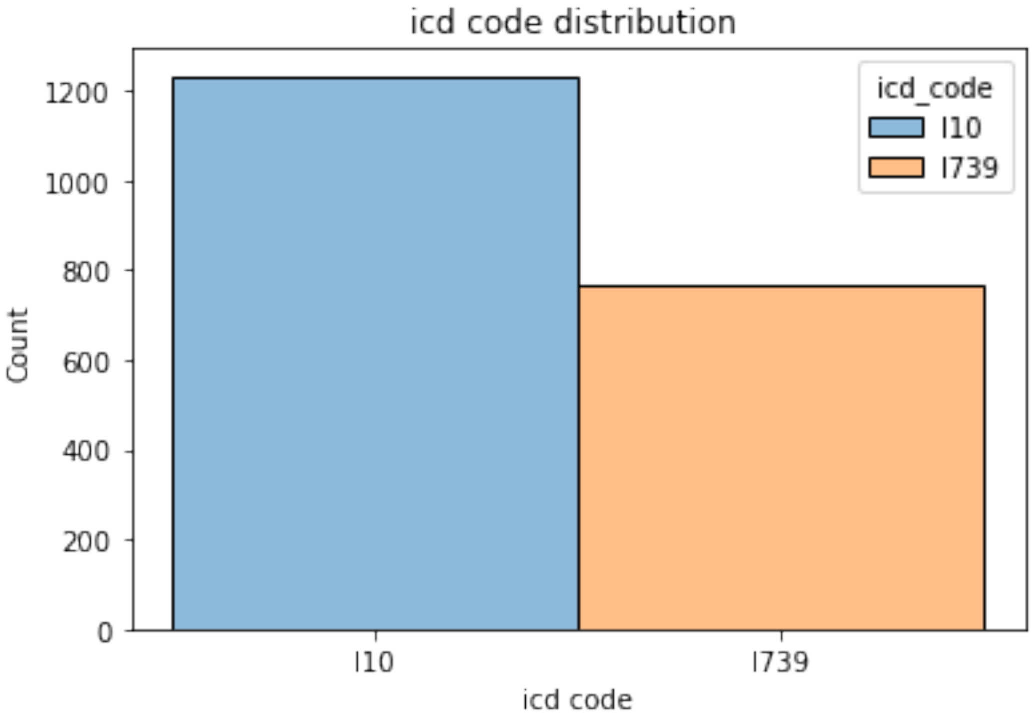
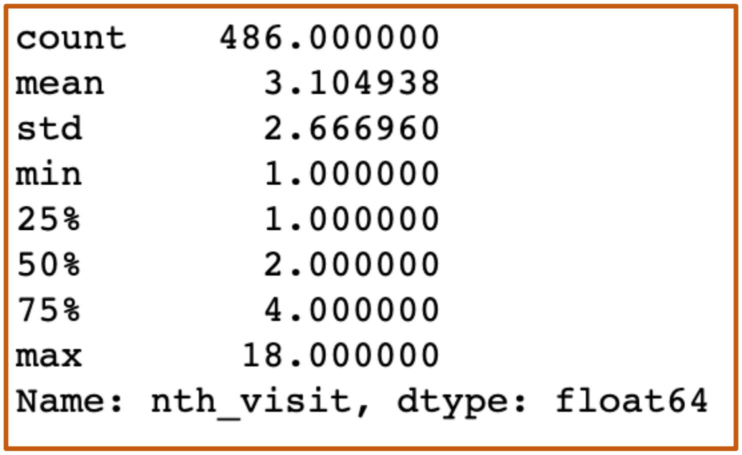


Figure 16

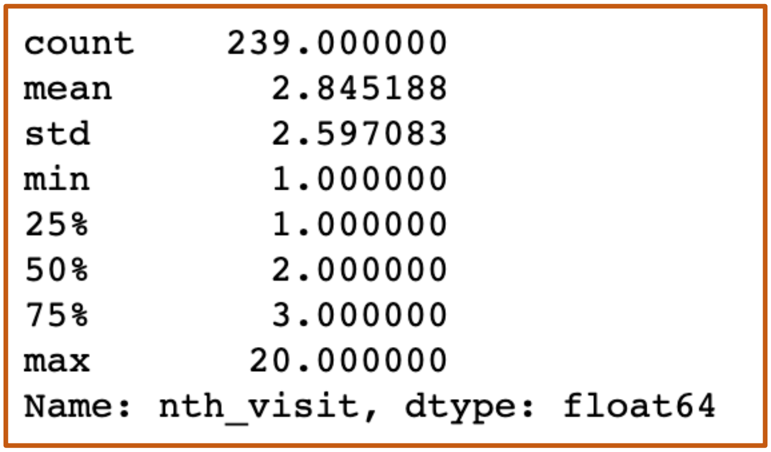
Statistical summary of number of hospital visits shows that patients with disease I10 visited the hospital, about 3.1 days on average (Figure 17), more than the patients with disease I739, 2.8 days on average (Figure 18). On the other hand, statistical summary of **days_to_prev_admission** shows that patients with I10 have about 195 as the average number of days since previous admission (Figure 19). This is 7 days higher than the average number of days since previous admission for patients with I739

(Figure 20). This is unexpected since patients with I10 visits hospital more, the average number of days since previous admission for these patients must be smaller. This might be because we have so many nulls in **days_to_prev_admission.**



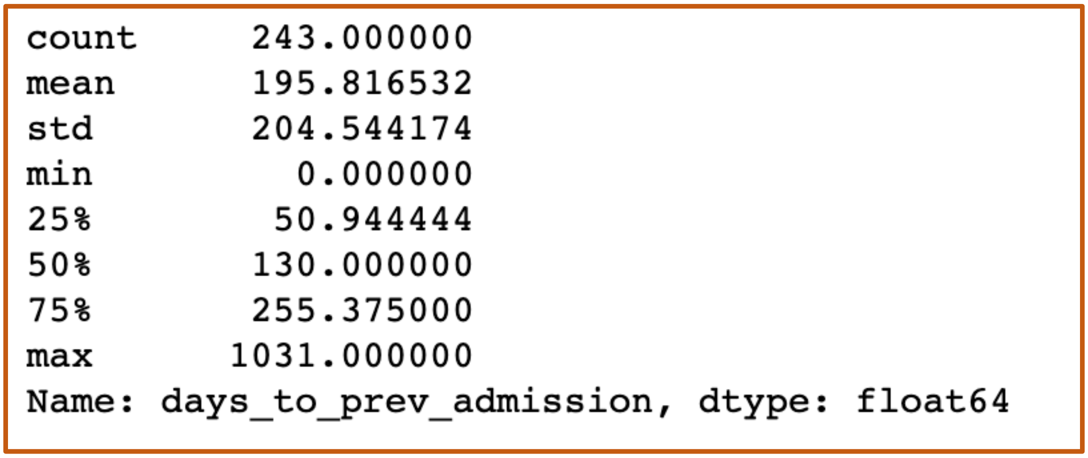
```
count    486.000000
mean       3.104938
std        2.666960
min        1.000000
25%        1.000000
50%        2.000000
75%        4.000000
max       18.000000
Name: nth_visit, dtype: float64
```

Figure 17 - Statistical summary of number of hospital visits for patients with I10



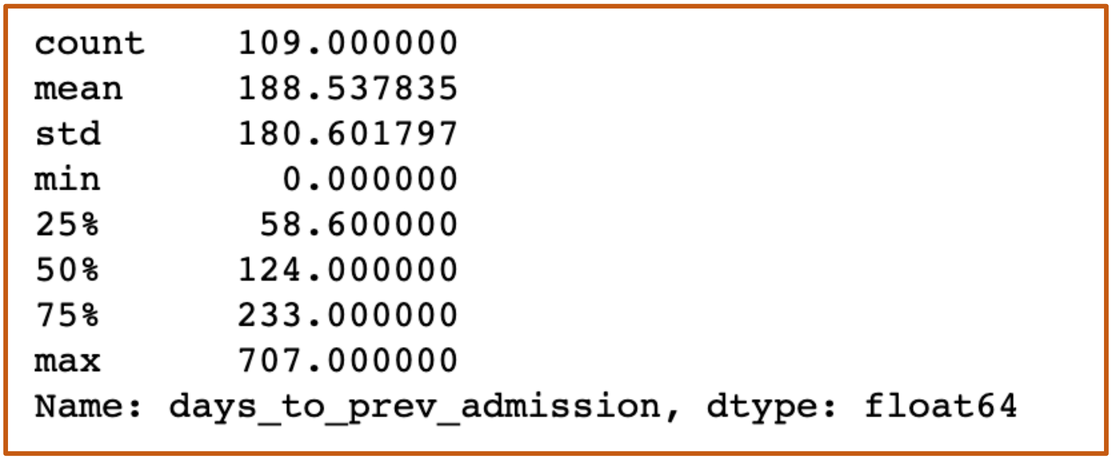
```
count    239.000000
mean       2.845188
std        2.597083
min        1.000000
25%        1.000000
50%        2.000000
75%        3.000000
max       20.000000
Name: nth_visit, dtype: float64
```

Figure 18 - Statistical summary of number of hospital visits for patients with I739



```
count      243.000000
mean       195.816532
std        204.544174
min          0.000000
25%         50.944444
50%        130.000000
75%        255.375000
max       1031.000000
Name: days_to_prev_admission, dtype: float64
```

Figure 19 - Statistical summary of days since previous admission for patients with I10

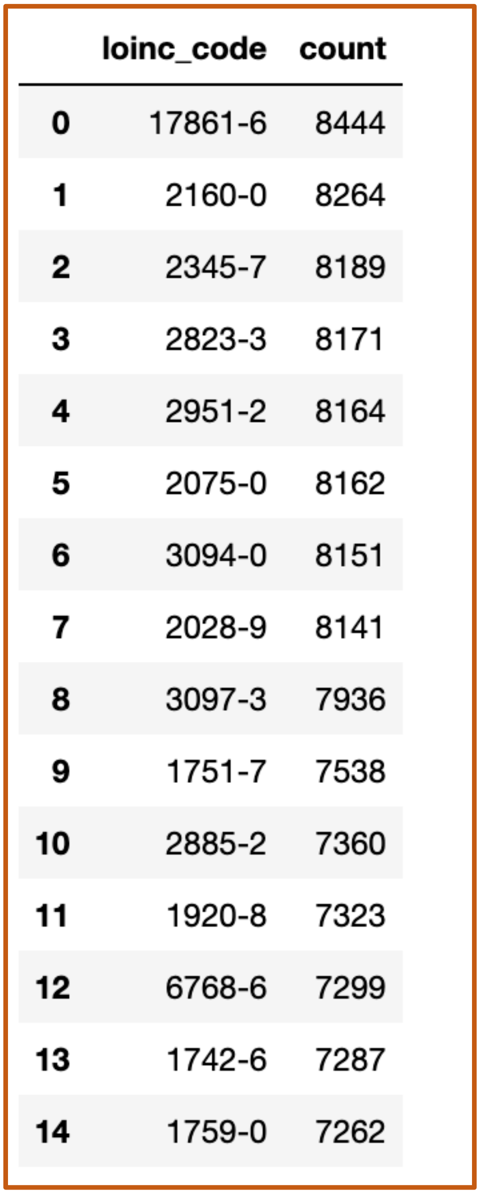


```
count     109.000000
mean      188.537835
std       180.601797
min         0.000000
25%        58.600000
50%       124.000000
75%       233.000000
max       707.000000
Name: days_to_prev_admission, dtype: float64
```

Figure 20 - Statistical summary of days since previous admission for patients with I739

EXPLORE THE LAB TESTS

The value counts in Figure 21 shows that tests can be divided into groups. Each group contained tests that were administered approximately the same number of times. This might be that some tests are typically interpreted along with results from other tests done at the same time. For instance, lab tests '2075-0' examined chloride levels in the blood and it is almost always done with test '2951-2', which measures sodium levels in the blood. This may explain why we have almost the same number for each of these two tests.



	loinc_code	count
0	17861-6	8444
1	2160-0	8264
2	2345-7	8189
3	2823-3	8171
4	2951-2	8164
5	2075-0	8162
6	3094-0	8151
7	2028-9	8141
8	3097-3	7936
9	1751-7	7538
10	2885-2	7360
11	1920-8	7323
12	6768-6	7299
13	1742-6	7287
14	1759-0	7262

Figure 21

We examined the top ten lab tests administered (Figure 22). It is worth noting that the top ten lab tests done have no results labelled as 'ABNORM'. However, they have some results labelled as 'L' and 'H'. Also, the lab test '2345-7' has a significant number of 'H' results. This test indicates the level of glucose in blood. A high level of blood glucose may imply that the patient is a Type 1 diabetic.

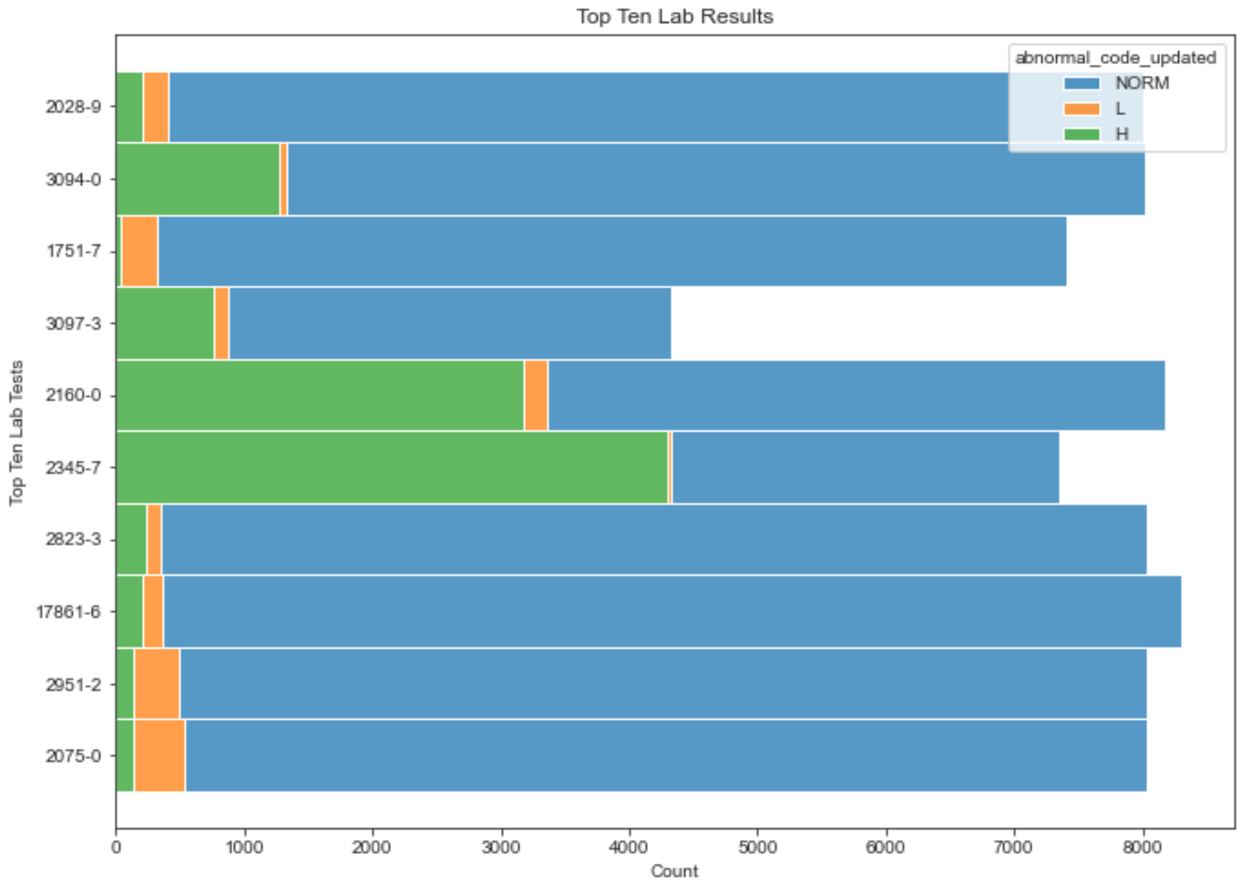


Figure 22

PREPROCESSING AND TRAINING

Using the method describe(), we discover that our resulting dataframe has values such as “inf” and “-inf” that might be problematic for our model. These values are imputed.

We constructed numerous potentially useful features, derived from the original data and summary statistics. We explored many of these features and found various trends. However, there is a challenge: we have too many features, many of which are uncorrelated with one an another. We will use feature selection algorithms to reduce the number of features. Doing so will enable us to visualize the data better and gain more insights. We use “SelectKBest()” function to determine the best 100 features among 30,056 features. The most useful 100 features can be summarized in the following manner:

- Created from **days_to_prev_admission**
- Duration of the hospital visits
- Drug quantity rates
- Total number of hospital visits
- Total number of lab tests administered
- Total number of medicines prescribed
- Type(s) of lab tests administered
- Types(s) of medicines prescribed
- Chronic verses acute

We filter the dataframe to include only the best 100 features. We split the data 30% to 70%. We use 30% for testing and 70% for training.

RANDOM FOREST CLASSIFICATION MODEL

We try the random forest classification model as our base model. We will set the hyperparameter **n_estimators** to 10, which is the square root of the number of features included in the data.

According to the classification report, the base model has an average performance of approximately 97% ranging from precision, recall, f1-score, and support. The performance of the models is given in Figure 24.

Next, we try cross-validation. We define a pipeline to assess performance. That way we don't have to check performance on the test split repeatedly. Also, **cross_validate** performs the fitting as part of the process. This uses the default settings for the random forest so we can then proceed to investigate different hyperparameters. We explore the hyperparameter **randomforestclassifier__n_estimators** which represents the number of trees in our random forest model. For the sake of simplicity, we try only 8 values ranging between 5 and 40 for the parameter.

The best result is obtained when the number of trees (**n_estimators**) is 30 with approx. This is consistent with the following picture which shows the relationship between 'n_estimators' and testing accuracy.

The average of the scores in grid search is 0.98. This result is better than our first result which was approx. 0.97 where we only have 10 trees in the model.

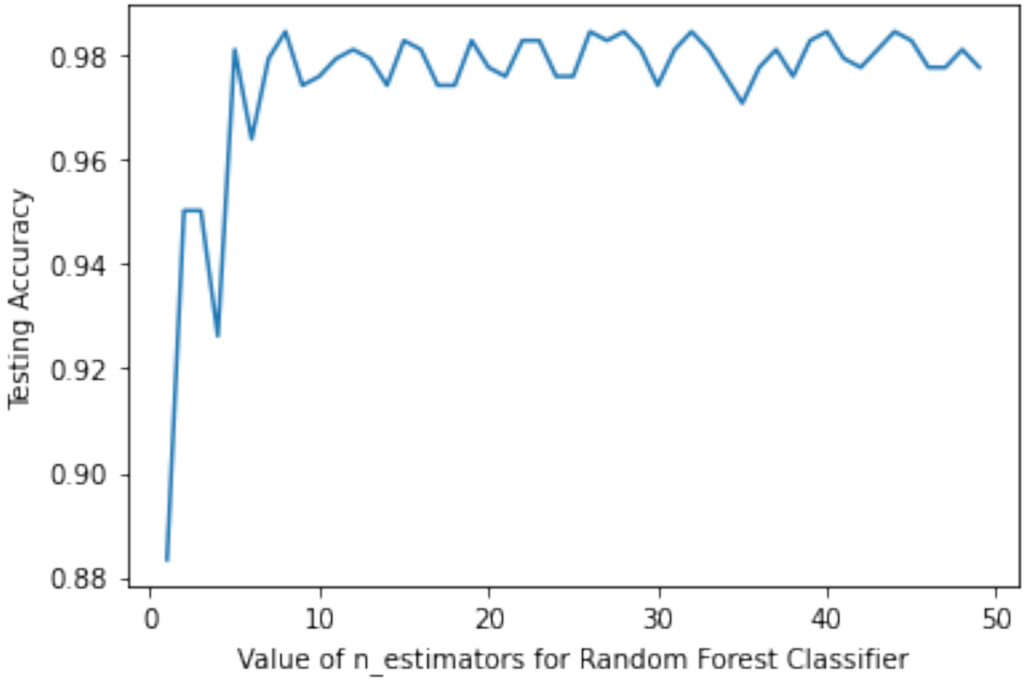
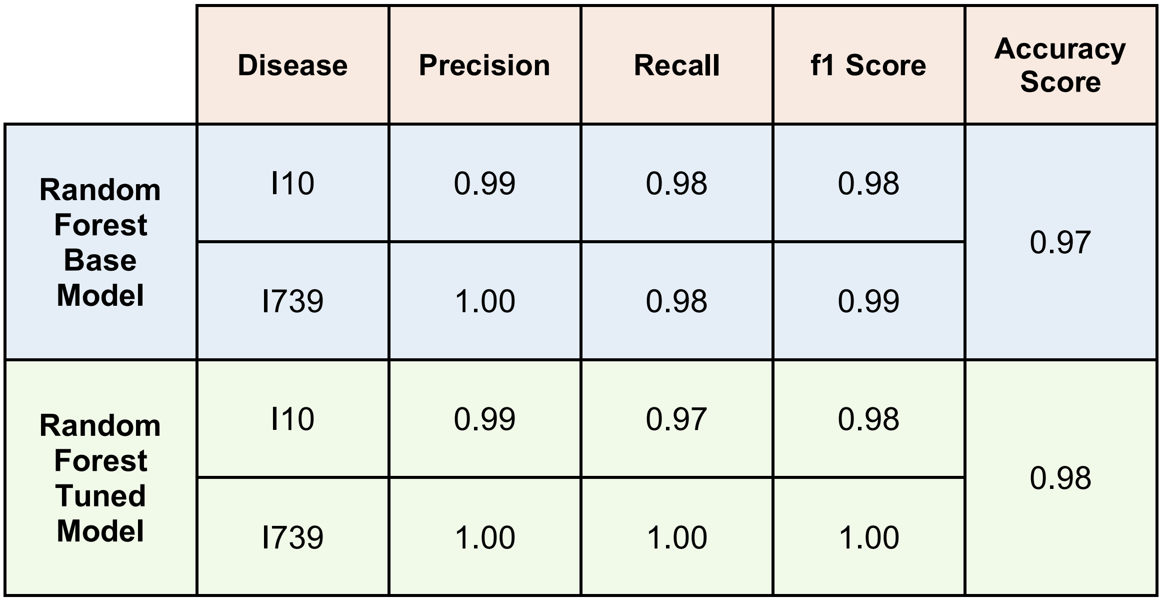


Figure 23



	Disease	Precision	Recall	f1 Score	Accuracy Score
Random Forest Base Model	I10	0.99	0.98	0.98	0.97
	I739	1.00	0.98	0.99	
Random Forest Tuned Model	I10	0.99	0.97	0.98	0.98
	I739	1.00	1.00	1.00	

Figure 24

KNN Model

We explore the KNN algorithm next. We set the number of neighbors to 10 for our base model. We get approx. 0.95 accuracy score in our first KNN model with 10 neighbors. This result is worse than any random forest model we have. To improve the performance, we do hyperparameter search using GridSearchCV. We get 0.96 as the best accuracy score in our grid search (see Figure 26). This result is better than our first KNN model accuracy score. We also observe that the number of neighbors must be 45 among the values we tried. This is consistent with the relationship between the error and the number of neighbors shown below.

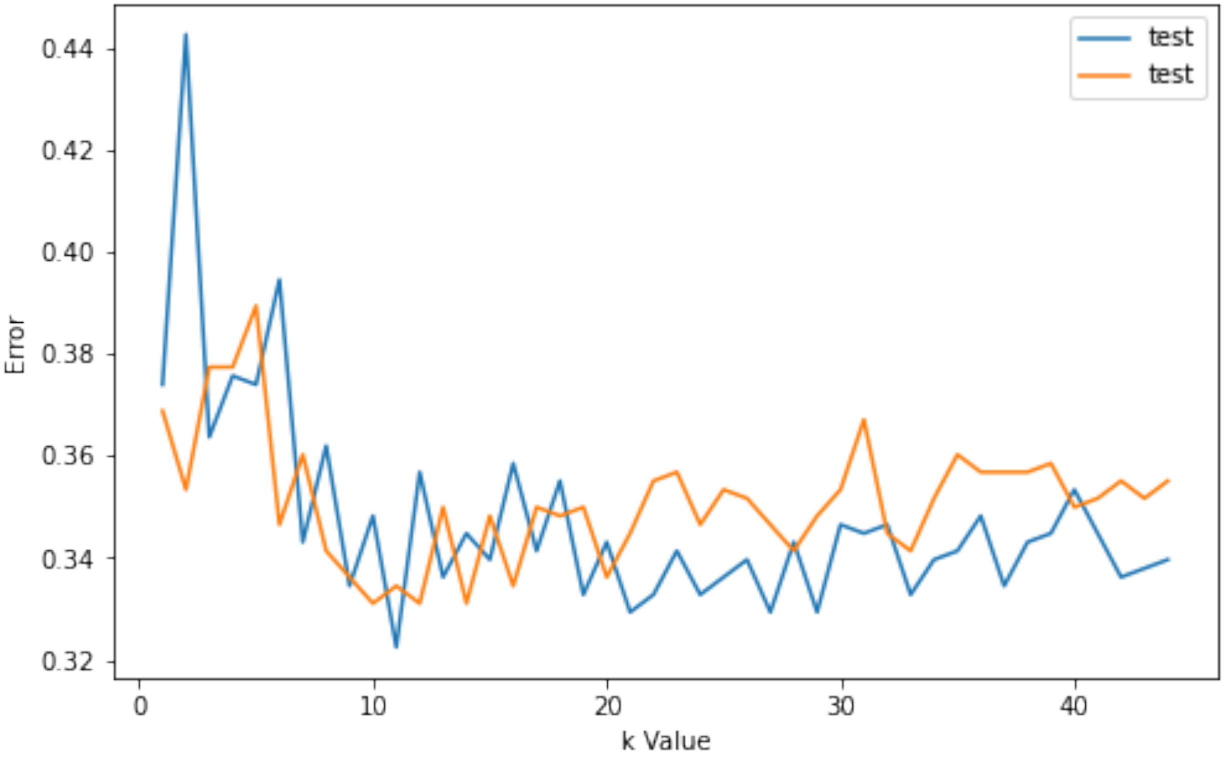
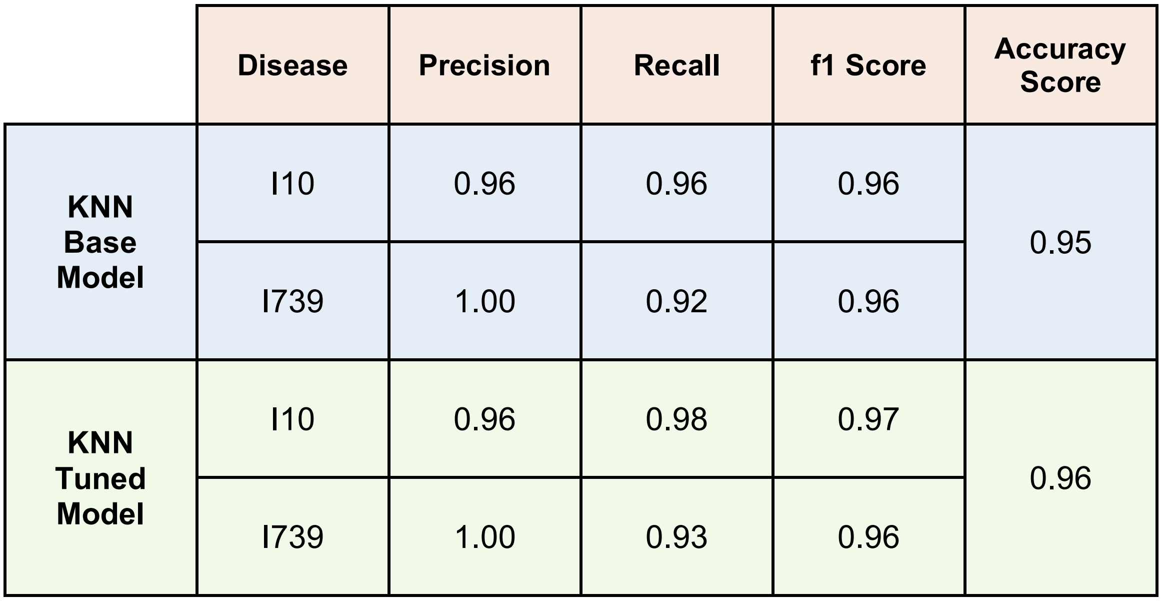


Figure 25



	Disease	Precision	Recall	f1 Score	Accuracy Score
KNN Base Model	I10	0.96	0.96	0.96	0.95
	I739	1.00	0.92	0.96	
KNN Tuned Model	I10	0.96	0.98	0.97	0.96
	I739	1.00	0.93	0.96	

Figure 26

Model Selection

The random forest classification model has a higher cross-validation test score compare to KNN model by almost 2%. It also exhibits less variability. We select the random forest classification model for these reasons.

MODEL

We use the best model that we built in the preprocessing and training phase. We try out the model with 3 different numbers of features. In general, one should prefer a model with a smaller number of features for the sake of simplicity and the computational efficiency. In addition to that, the performance of the model is also crucial for decision making. We determine the optimal number of features.

Refit Model – 30056 Features

First, we refit the best model using all 30056 features. We observe the following:

- Using the cross-validation we see that the average of the accuracy scores with 5-folds is 81% approximately. The standard deviation of the scores is 2.9% approximately. This indicates that the variance of the scores is significant.
- The classification report shows that the patients with 'I10' were classified correctly at a rate of 97%. On the other hand, the patients with 'I739' were classified correctly at a lower rate of approximately 66%.

Refit Model – 100 Features

Next, we refit the best model using the best 100 features. Doing so produces the following results:

- The average of the accuracy scores in the cross-validation with 5-folds is about 97%. This is much higher than the previous model. The standard deviation of the scores is 0.7% approximately, which implies that the variance of the scores is fairly small.
- The classification report tells us that the patients with 'I10' were classified correctly at a rate of 97%. The patients with 'I739' were classified correctly at a higher rate of approximately 100%.

Refit Model – 50 Features

Lastly, we refit the best model using the best 50 features. The following are worth noting:

- The average of the accuracy scores in the cross-validation with 5-folds is about 98%. This is about 1% higher than the previous model's score. The standard deviation of the scores is approximately 0.3%, which is also less than the counterpart of the previous model.
- The classification report shows that the patients with 'I10' were classified correctly at a rate of 98%. The patients with 'I739' were classified correctly at a higher rate of approximately 100%.
- Overall, the performance of this model is the best compared to the previous models.

Conclusion

The model with only 50 features has a higher accuracy score in comparison to the first two models. Not only that, it has a lower variance of the scores which implies that the model depends less on the data points that have been chosen. Furthermore, this model has the highest f1 score as well as the highest recall for both classes. In general, one should prefer a model with the least number of features for the sake of simplicity and the computational efficiency. Given the fact that our model is doing better with a lesser number of features, we choose our last model which has only 50 features.

CRITIQUES OF THE MODEL

WHAT IS GOOD ABOUT THE MODEL?

The accuracy score is very high in general for classification models. The model is also doing well with respect to other metrics such as precision, recall and f1 score (see previous section on Models). Additionally, the original data is mostly preserved. There have been only minor alterations made on the data along the process. Another advantage of the model is using a modest quantity of 50 features, which is helpful for computational effectiveness.

WHAT IS BAD ABOUT THE MODEL?

The model considers 2 diseases, which could cause potential issues given its binary nature. For instance, some features can favor only one of the diseases.

If diseases are mutually exclusive with respect to an attribute such as being chronic or acute, then the performance of the model may decrease when it is applied to a pair of other diseases. Likewise, some medicines are given only to the patients with one of the two diseases. These medicines are highly conclusive as a result and can interfere with the model. We don't want a model that relies on only a few attributes because in future models we may not have data regarding to the attributes for all patients.

Having 2000 patients is another downside to the model. This is only 7.5% of the total number of patients and therefore may indicate that the model is not representative of the population.

FUTURE PROJECTS

We would like to increase the number of records gradually; we want to double, triple, and quadruple the number of records and compare the performance of the model with our project at each case. This will allow us to see how much our model depends on the number of records and whether the model is viable on a larger scale.

We would like to classify patients with respect to more diseases. We would like to increase the number of diseases one at a time to test and review how our model is doing in each case. We would repeat this process, steadily increasing the number of diseases by one each time until we are confident that the model can handle more complex classifications. In addition, we may consider each case with multiple number of records to assess the performances better.

Finally, we would like to classify all 26,559 patients in accordance with all 12,959 diseases using our model.