IFN 509: Data Exploration and Mining

Assessment 1

Team Name: Dream Team

Group No. 24

Student Name	Student Id
Jackson Calvert-Lane	n10787003
Bella Qian	n10661450
Darren Ross	n10788557
Brendan Wallace-Nash	n9993304

	Jackson	Bella	Darren	Brendan
Jackson	<100 %>	<100 %>	< 100%>	<100 %>
Bella	<100 %>	<100 %>	<100 %>	<100 %>

Darren	<100 %>	<100 %>	<100 %>	<100 %>
Brendan	<100 %>	<100 %>	<100 %>	<100 %>

Task1:

When processing the data through pandas, it was observed that some of the variables did not match the data dictionary provided. These variables are as follows:

Variable	From	To	Reason
admission_type_id	Integer	String (Categorical)	These values represent category ids, thus should be displayed as string
discharge_disposition_id	Integer	String (Categorical)	
admission_source_id	Integer	String (Categorical)	
diabetesMed	Object	Binary(Categorical)	These values where mapped to binary values as they were categorical data with only two categories
change	Object	Binary(Categorical)	

```
df['admission_type_id'] = df['admission_type_id'].astype(str)
df['discharge_disposition_id'] = df['discharge_disposition_id'].astype(str)
df['admission_source_id'] = df['admission_source_id'].astype(str)
# these are categorical data so I changed them to strings
```

```
df['admission_type_id'].unique()
#done this to demonstrate its now a string
```

```
array(['2', '3', '1', '5', '6', '8', '7', '4'], dtype=object)
```

```
# mapping yes and no to a binary format as there are only two catagorical values
diabetesMed_map = {'Yes':0, 'No':1}
df['diabetesMed'] = df['diabetesMed'].map(diabetesMed_map)
```

```
df['diabetesMed'].value_counts()
```

```
0    40966
1    10777
Name: diabetesMed, dtype: int64
```

```
# mapping change and no change to a binary format as there are only two catagorical values
change_map = {'No':0, 'Ch':1}
df['change'] = df['change'].map(change_map)
```

```
df['change'].value_counts()
```

```
1    26649
0    25094
Name: change, dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 51743 entries, 0 to 51765
Data columns (total 27 columns):
race                      51743 non-null int64
gender                    51743 non-null int64
age                       51743 non-null int64
admission_type_id         51743 non-null object
discharge_disposition_id  51743 non-null object
admission_source_id       51743 non-null object
time_in_hospital          51743 non-null int64
num_lab_procedures        51743 non-null int64
num_procedures            51743 non-null int64
num_medications           51743 non-null int64
number_outpatient         51743 non-null int64
number_emergency          51743 non-null int64
number_inpatient          51743 non-null int64
number_diagnoses          51743 non-null int64
max_glu_serum             51743 non-null object
A1Cresult                 51743 non-null int64
metformin                 51743 non-null object
repaglinide               51743 non-null object
nateglinide               51743 non-null object
chlorpropamide            51743 non-null object
glimepiride               51743 non-null object
glipizide                 51743 non-null object
glyburide                 51743 non-null object
insulin                   51743 non-null int64
change                    51743 non-null int64
diabetesMed               51743 non-null int64
readmitted                51743 non-null int64
dtypes: int64(16), object(11)
```

The above screenshot is the end result of datatypes for the data once we have changed the integer data to strings to identify categories. You can see that admission_type_id, admission_source_id and discharge_disposition_id are now objects, and admission_type_id is made up of unique strings representing each value within the category.

Task2:

1) To identify and report the skewness of the variables we split the data up into a numeric data frame and an object data frame. A skewness function was used to measure skewness on all numeric variables and the mode function was used to measure the skewness of all objects.

```
df_numbers.skew()
```

```
time_in_hospital       1.197547
num_lab_procedures    -0.342470
num_procedures         1.385648
num_medications        1.190215
number_outpatient      7.794059
number_emergency      21.275732
number_inpatient       3.494265
number_diagnoses      -1.303619
change                -0.060134
diabetesMed            1.436814
dtype: float64
```

```
df_objects.mode()[0:1]
```

	race	gender	age	diag_1	diag_2	diag_3	max_glu_serum	A1Cresult	metformin	repaglinide	nateglinide	chlorpropamide	glimepiride	glipizide	glyburide	insulin	readmitted
0	Caucasian	Female	[70-80)	428	276	250	None	None	No	No	No	No	No	No	No	No	NO

In terms of the numeric data the variables *change of medication* and *number of lab procedures* were moderately negatively skewed. All other variables were highly negatively skewed, except for *number of diagnosis* which was highly positively skewed. Notable variables were *number of emergency visits* (21.28), *number of inpatients* (3.49) and *number of outpatients* (7.49) which were very highly positively skewed.

For the categorical data variables such as *race, gender* and *age* were skewed towards *caucasian, Female* and *70-80* respectively. Another key insight derived from the mode of the categorical data is that most people were not prescribed medicine with all medication variables being skewed towards *No* (did not receive medicine).

2) To identify any inconsistencies in the data a for loop was created to display all the unique values for all the variables in the data. Through this we could see which variables had errors or missing data and then use the *values counts* function to determine the amount of missing data in the variable. Variables such as *weight, payer code* and *medical speciality,* had more missing data then they did valid data and as a result were dropped from the dataset. *Acetohexamide* and *tolbutamide* were not used by any patients in the

data set so they were also dropped. In the case of *patient nbr* and *encounter id* it was decided that they were not important to our exploration of the data so they two were dropped. The variables *Diag 1, Diag 2* and *Diag 3* were also dropped as they were stated to be a 3 digit code but many of the values contained strings and doubles

```
[39]: for col in df:
          print(df[col].unique())
      #for loop to display all unique vaues so we can see what colomns have missing data or errors in there data.
```

```
[ Female   Male   Unknown/Invalid ]
['[60-70)' '[80-90)' '[70-80)' '[40-50)' '[50-60)' '[90-100)' '[30-40)'
 '[20-30)' '[10-20)' '[0-10)' '?']
['?' '[100-125)' '[50-75)' '[75-100)' '[0-25)' '[125-150)' '[25-50)'
 '[150-175)' '[175-200)' '>200']
['2' '3' '1' '5' '6' '8' '7' '4']
['1' '2' '6' '3' '22' '11' '7' '23' '13' '4' '15' '9' '28' '5' '19' '8'
 '25' '14' '24' '27' '18']
['1' '7' '5' '17' '4' '2' '3' '6' '9' '10' '22' '20' '8' '14' '11' '25'
 '13']
[ 4  8  2  7  3  6  5  1 13 12 11  9 10 14]
['?' 'MD' 'BC' 'MC' 'HM' 'CP' 'SP' 'OG' 'UN' 'DM' 'CM' 'PO' 'SI' 'WC' 'CH'
```

```
[40]: df['medical_specialty'].value_counts()
```

```
[40]: ?                                32203
      Emergency/Trauma                  5616
      InternalMedicine                  3854
      Family/GeneralPractice            1949
      Cardiology                        1667
      Surgery-General                   1326
      Radiologist                       1070
      Orthopedics                        681
      Nephrology                         468
      Surgery-Vascular                   332
      Gastroenterology                   297
      Orthopedics-Reconstructive         286
```

```
df = df.drop(columns=['weight','patient_nbr', 'encounter_id','medical_specialty','acetohexamide', 'tolbutamide', 'diag_1', 'diag_2', 'diag_3' ])
```

For variables such as *age, gender* and *chlorpropamide* the missing values were less than 1% of the total values in their variables, because of this we decided it would not affect the data if we just dropped these missing values. In the case of *race* the missing values made up a larger share of the values in the variable. We decided to replace the missing values with the mode of *race*, in this case *caucasian.*

```
df['chlorpropamide'].value_counts()
#check the amount of missing values in chlorpropamide. Determined they were small enough to remove
```

```
No        51738
Steady       18
?             9
Up            1
Name: chlorpropamide, dtype: int64
```

```
indexNames = df[ df['chlorpropamide'] == '?' ].index
df.drop(indexNames , inplace=True)
```

```
df['chlorpropamide'].value_counts()
```

```
No        51738
Steady       18
Up            1
Name: chlorpropamide, dtype: int64
```

```
df['race'].value_counts()
#check the missing values of race. The missing values were large so chose to replace them with the mode (Caucasian) as it is categorical data
```

```
Caucasian          40382
AfricanAmerican     8068
Hispanic            1018
?                   1015
Other                896
Asian                378
Name: race, dtype: int64
```

```
df['race'].mode()
```

```
0    Caucasian
dtype: object
```

```
df['race'] = df['race'].replace(['?'],'Caucasian')
# replacing "?" with caucasian
```

```
df['race'].value_counts()
#showing that its done
```

```
Caucasian          41397
AfricanAmerican     8068
Hispanic            1018
Other                896
Asian                378
Name: race, dtype: int64
```

3)

a) To calculate the average *time stay* in hospital for *females* who were *readmitted* in *less than 30 days* we first made a dataframe that only had female patients.

```
df_female = df[df['gender']=='Female']
#created a data frame gender is just female
```

From here *Readmitted* values *No* and *>30* were removed from the dataframe so only *Female* patients *readmitted* in *<30* days were in the data frame.

```
indexNames = df_female[ df_female['readmitted'] == 'NO' ].index
df_female.drop(indexNames , inplace=True)
#dropped all rows with no readmission
```

```
indexNames = df_female[ df_female['readmitted'] == '>30' ].index
df_female.drop(indexNames , inplace=True)
#removed all female patients with over 30 days spent in hospital
```

Once these values were removed we were able to calculate the mean *time in hospital* for *Female* patients *readmitted* in *<30* days, which was 4 and a half days (4.53).

```
df_female['time_in_hospital'].mean()
#the average time spent in hospital by a female patient  who has been readmitted in under 30 days (<30)
#is 4 days and a half das (4.52 days)
```

```
4.524752475247524
```

b) To Calculate which *age* group has the highest risk of being *readmitted* within 30 days we first had to map numeric values to *age*.

```
age_map = {'[0-10)':0,'[10-20)':1, '[20-30)':2, '[30-40)':3,'[40-50)':4, '[50-60)':5, '[60-70)':6, '[70-80)':7,'[80-90)':8,'[90-100)':9}
df['age'] = df['age'].map(age_map)

#mapped age to for further testing that needs numerical values.
```

From here we then created a dataframe with only patients that were *readmitted* in *<30*.

```
df_readmitted =df[df['readmitted']=="<30"]
```

To calculate the risk for each *age* group of being *readmitted* in *<30* days we had to divide the number of patients in each *age* group that were *readmitted* in *<30* by the total number of patients in that *age*. To do this we created keys for the value count of *age,* in both the original dataframe and also the dataframe with only patients who were *readmitted* in *<30* days. Once we had the keys we made a for loop that divided the value counts of an *age* group from the *readmitted* in *<30* dataframe by the value count of the age group from the original dataframe. The results of this loop produced a percentage that represented the risk of patients from that *age* group to be *readmitted* in *<30*.

```
# We can use the age groups from the readmitted vs the age grounds for
# admitted people to get % of people within an age group who were readmitted
readmitted_dict = df_readmitted['age'].value_counts().to_dict()
age_dict = df['age'].value_counts().to_dict()
```

```
for key in readmitted_dict:
    print(f"{key}: {readmitted_dict[key]/age_dict[key]}")

# this for loop calculates the percentage of patients that ended up being
#admitted
```

```
7: 0.11064846942709539
6: 0.10971408887357906
8: 0.11685625646328852
5: 0.09293548767543343
4: 0.10197012340333406
9: 0.11943069306930693
3: 0.10891089108910891
2: 0.14846625766871166
1: 0.06726457399103139
```

The statement demonstrates that people in the *20-30 age* group had a 14.8% chance of being readmitted, which was a higher risk than any other *age* group.

c) To calculate how many *age* groups had more than 3000 cases of being *readmitted* we first made a dataframe that did not include any patients that were not readmitted.

```
df_drop_no = df_readmitted =df[df['readmitted']!="NO"]

#created a dataframe that removed patients that were not admitted.
```

As we have isolated only the patients that have been *readmitted* we are able to perform a values counts function on *age* to display the number of patients that have been readmitted.

```
df_drop_no = df_readmitted =df[df['readmitted']!="NO"]

#created a dataframe that removed patients that were not admitted.
```

```
df_drop_no['age'].value_counts()

#value counts show the 4 age groups (50-60, 60-70, 70-80, 80-90)
#that have had over 3000 people readmitted
```

```
7    6202
6    5345
8    4741
5    3691
4    2008
3     709
9     676
2     351
1      91
0       6
Name: age, dtype: int64
```

To precisely identify the *age* groups that have had over 3000 patients readmitted we again assigned the value counts of *age* to a dictionary and performed a for loop that dictated if each *age* group had more than 3000 inpatient readmitted.

```
readmitted_3000_dict = df_drop_no['age'].value_counts().to_dict()
```

```
for key in readmitted_dict:
    print(f"{key}: {readmitted_3000_dict[key]>3000}")

#used the dictionary-for loop method from the previous question
#to produce binary outputs for if the age category has had
#more then 3000 patients readmitted
```

```
7: True
6: True
8: True
5: True
4: False
9: False
3: False
2: False
1: False
```

From this we can clearly see that *age* groups 50-60, 60-70, 70-80 and 80-90 were the only age groups to have more than 3000 patients readmitted.

d) To find out which are the top three *races* for to be readmitted we again used the data frame we previously used to look at only patients that had been readmited. From here we could just perform a values counts function on *race* to establish the top three *races* to be readmitted.

```
df_drop_no['race'].value_counts()
```

```
Caucasian          19480
AfricanAmerican     3530
Other                346
Hispanic             344
Asian                120
Name: race, dtype: int64
```

From this statement we can see that the top three races for readmission are *Cuacasians* with 19,480 readmissions, *African Americans* with 3,530 readmissions, and *Other* with 344 readmissions.

Although this outcome provides an answer to the presented question it is not necessarily a useful insight as the amount of readmission for each *race* did not take into consideration the disproportionate representation of *races* like *cuacasiansins*. To account for this we used a similar method to how we found the risk of readmission for each *age* group.

```
readmitted_race_dict = df_drop_no['race'].value_counts().to_dict()
race_dict = df['race'].value_counts().to_dict()
```

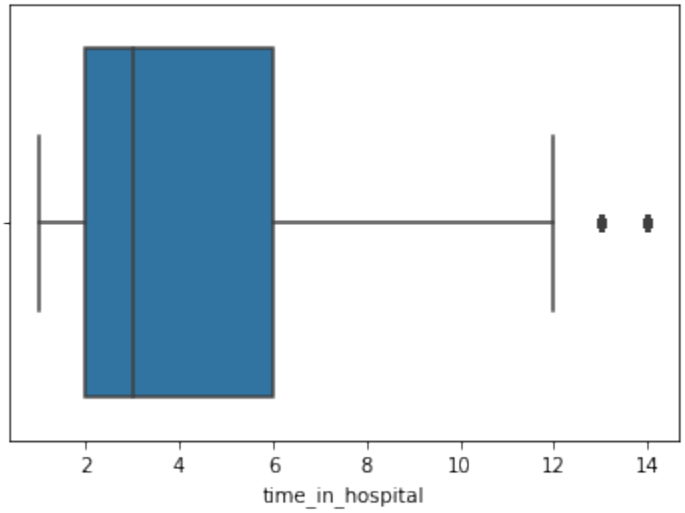
```
for key in readmitted_race_dict:
    print(f"{key}: {readmitted_race_dict[key]/race_dict[key]}")
```

```
Caucasian: 0.47065645461354466
AfricanAmerican: 0.43753098661378287
Other: 0.38789237668161436
Hispanic: 0.33858267716535434
Asian: 0.31746031746031744
```

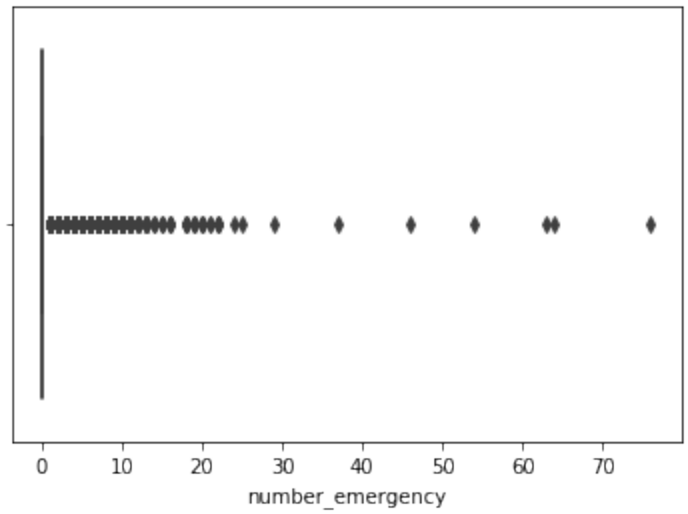
While this method still produces the same top three *races* for readmission, it better represents the remission rate for each *race* as it takes into consideration the overrepresentation of races such as *Cuacasian*.

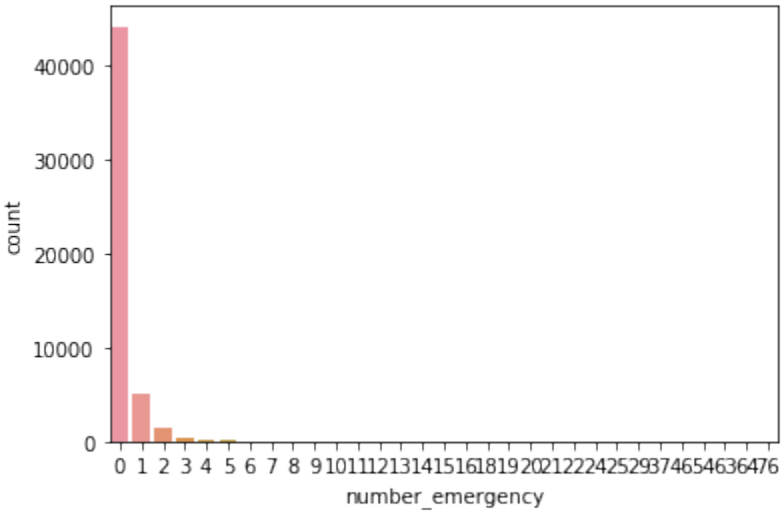
Task3:

1) In order to understand the distribution of variables, we used boxplot, distribution plot and histogram to identify data quality problems, such as skewness.

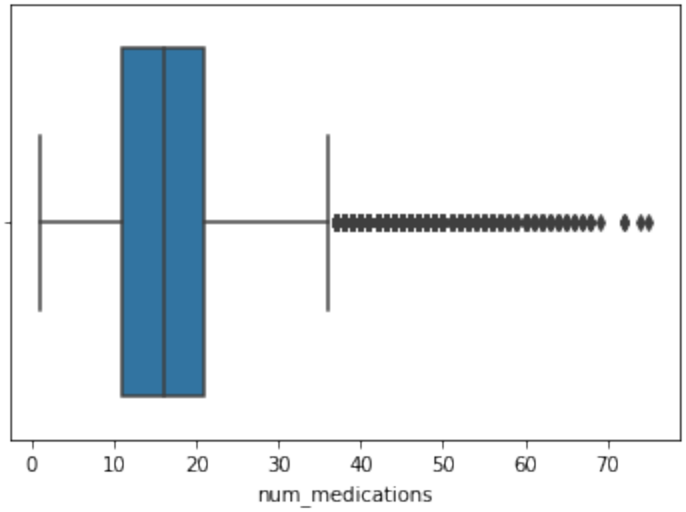


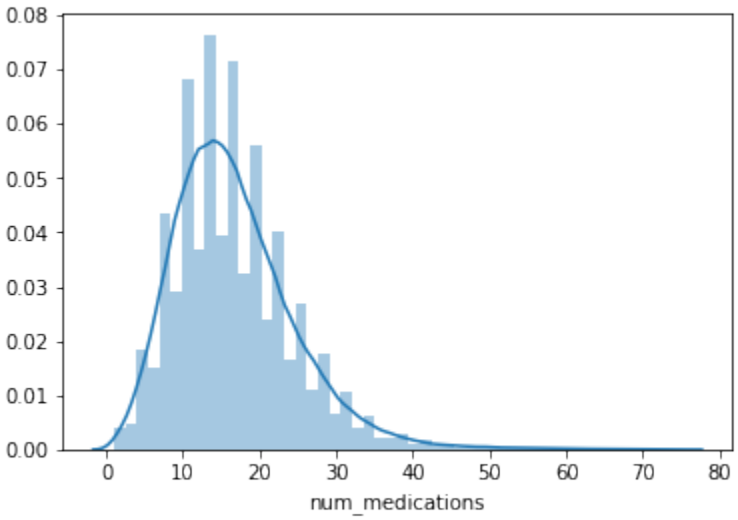
Boxplot of variable ‘time_in_hospital’ indicates that there are two potential outliers. By checking the value counts of ‘time_in_hospital’, we found that although 13(days) and 14(days) are two reasonable values rather than errors of the data set. Hence we decide to keep the values as they are.



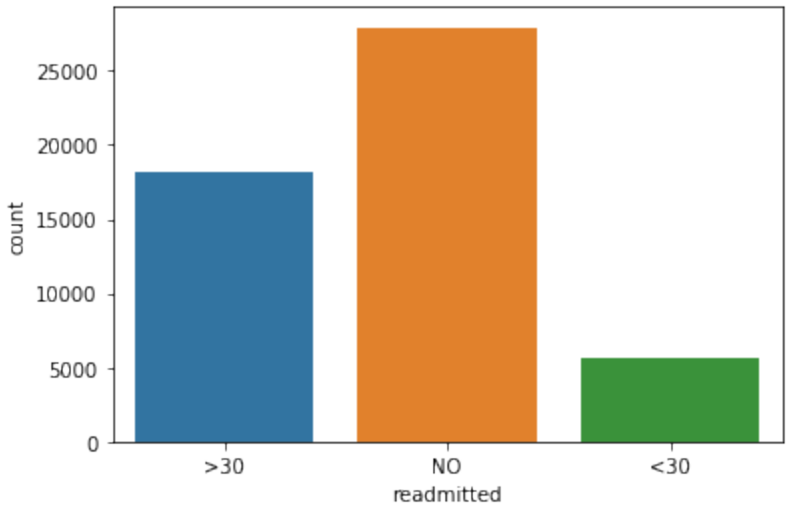


According to the boxplot and histogram of variable ‘number_emergency’, the values are extremely skewed with a large amount of potential outliers. Most patients have 0 emergency visits. However, there was a patient who had 76 emergency visits. This is significantly outside of the norm, which could be an error in the data set or a special medical condition. Further investigation is required.





Boxplot and distribution plots of variable ‘num_medicatins’both indicate that the values of this variable is heavily skewed. Since mean > median > mode , this is a positively skewed distribution.



For categorical variable ‘readmitted’, a histogram is created to help us understand the distribution of the values. There is no missing value or unusual value, this distribution also looks reasonable.

By far, we have checked and made sure there is no significant data quality problem with ‘num_medications’ and ‘readmitted’, we can proceed to the next step in our analysis to identify correlations between these two.

2) Since ‘num_medications’ is a numerical variable and ‘readmitted’ is a categorical one, we need to change ‘readmitted’ into a numerical variable.

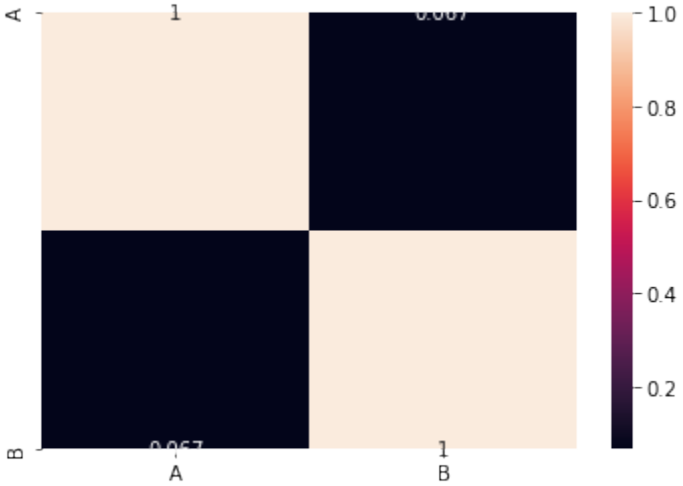
```
# change readmitted into numerical 0/1/2 variable
readmitted_map = {'NO':0, '<30':1, '>30':2}
df['readmitted'] = df['readmitted'].map(readmitted_map)
```

```
# showing that readmitted has been transformed into numerical variable
df['readmitted'].value_counts()
```

```
0    27923
2    18206
1     5614
Name: readmitted, dtype: int64
```

By using a correlation matrix between the two values, ‘num_meducations’ and ‘readmitted’, we are able to determine if there is a relationship between the two values. To do this, the categorical readmitted data should be mapped to integers. We can then make a separate dataframe with columns A and B with the data.

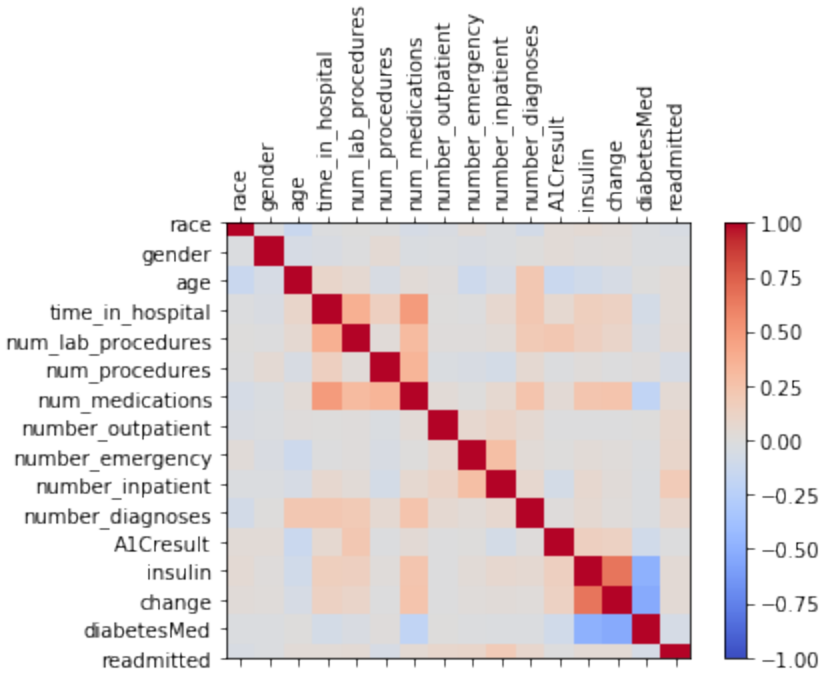
Doing this leads to the following correlation matrix;



As we can see from the above diagram, there is no correlation (0.067) between the variables ‘A’ and ‘B’ leading to no correlation between ‘readmitted’ and ‘num_medications’.

If there were to be a strong relationship between the two variables to the point where they are strongly correlated, it would be wise to remove one of them from a model. If both were present in the model and they were highly correlated, then the model would have difficulty with discovering patterns that are less obvious.

3) Within this dataset, we were able to add a correlation matrix for the dataframe on numerical values. Doing this gave the following correlation matrix:



From here we can see that there were no highly correlated variables within the dataset. Although the graph indicates that 'change' and ' insulin' are highly correlated, by going through the description of variables, we can see 'change' indicated if there was a change in diabetic medications and 'insulin' is a diabetic medication, this pair is a false predictor hence only one should be kept.

Apart from the false predators pair, the largest value for correlation was between 'num_medications' and 'time_in_hospital', with a Pearson correlation score of 0.49. This is under the threshold for 'highly correlated' thus both can remain within the dataset.

Task 4:

1) The main purpose of this text was to analyze a large volume of healthdata using various statistical and computational techniques. After first cleaning the data, task 2 and task 3 gave us a number of interesting insights. First measuring the skewness in task 2 showed many of the variables were highly skewed, notable variables emergency visits, number of impatiens and number of outpatients being very highly positively skewed. Further in this

section we analysed different groups' likelihood of being readmitted to hospital within 30 days. First reviewed the average stay time of females who were readmitted in under 30 days showing 4.5 days. Secondly reviewed the highest age group being readmitted, somewhat surprisingly the younger age group of 20-30 had a 14.8% chance of readmission making them the highest risk of all age groups. Whilst this group had the highest chance of being readmitted we found in terms of the number of the age group being readmitted age groups 50-60, 60-70, 70-80 and 80-90 had over 3000 patients readmitted.

Using a number of visualization tools we assessed the quality of the data and the distribution of a number of variables including emergency visits and number of mediations. An interesting find was a single patient had 76 emergency visits over the 9 year period (most patients having 0). Speculated that it could be a special medical condition however further review would be required. We did not find any variables that were highly correlated (number of medications & time in hospital was highest with correlation score 0.49). If further analysis was to be done (machine learning modeling etc), change and insulin are highly correlated however the change variable contains insulin in it meaning one of these would need to be removed from the model.

2) Exploring the data showed a number of quality errors. Weight had a large number of values missing (50,431/51,766 or 97.42%). Further without another variable alongside it such as BMI, height, waist measurement etc. nothing valuable could be inferred from weight, this data should be recorded as obesity is a key indicator of type 2 diabetes. Encounter ID and Patient number it is not necessary to log the patients this data can be anonymised. Acetohexamide and Tolbutamide were not prescribed to any patients. Payer code is relating to insurance data, all of these variables were dropped as they would not affect the analysis we were performing.

Variables age, Chlorpropamide and gender all had missing data (under 1%) that was small enough to be dropped without impacting our dataset in a substantial way. The choice to drop invalid/unknown because biological gender is a predicting factor in diabetes making unknown/invalid an uncontrolled variable in the data

Binary value Diabetes was mapped to a numerical value (0 for no, 1 for yes) in order to easily perform analysis on.

Numerical identifiers Admission ID, Discharge Disposition ID and Admission Source ID are for categorical data, therefore saved as string data types. For these variables to be easily displayed as a dict key and so numerical analysis cannot be done on them.

3) Dropping unnecessary columns;

```
df = df.drop(columns=['weight','patient_nbr', 'encounter_id',
                      'medical_specialty','acetohexamide',
                      'tolbutamide', 'diag_1', 'diag_2', 'diag_3' ])
```

Dropping specific '?' values (missing data)

```
df['race'].value_counts()
#check the missing values of race. The missing values were la
```

```
Caucasian          40382
AfricanAmerican     8068
Hispanic            1018
?                   1015
Other                896
Asian                378
Name: race, dtype: int64
```

```
df['race'].mode()
```

```
0    Caucasian
dtype: object
```

```
df['race'] = df['race'].replace(['?'],'Caucasian')
# replacing "?" with caucasian
```

```
df['gender'].value_counts()
#checking missing values in gender
```

```
Female             27714
Male               24039
Unknown/Invalid        4
Name: gender, dtype: int64
```

```
indexNames = df[ df['gender'] == 'Unknown/Invalid' ].index
df.drop(indexNames , inplace=True)
#the choice to drop invalid/unknown gender was made as for one t
```

```
df['age'].value_counts()
#showing the unknown varaibles for age, small enoug
```

```
[70-80)     12969
[60-70)     11612
[80-90)      9670
[50-60)      8479
[40-50)      4619
[30-40)      1717
[90-100)     1616
[20-30)       815
[10-20)       223
[0-10)         23
?              10
Name: age, dtype: int64
```

```
indexNames = df[ df['age'] == '?' ].index
df.drop(indexNames , inplace=True)
#removing "?"
```

Mapping diabestest to numerical

```
# mapping yes and no to a binary format as there are only two catagorical values
diabetesMed_map = {'Yes':0, 'No':1}
df2 = df.copy()
df2['diabetesMed'] = df2['diabetesMed'].map(diabetesMed_map)
```

Task 5:

1) Association mining would be the most suitable tool to improve our analysis as described in Task 2 and Task 3. Described in a simple way, association mining will use all the data held (Age, Race, Gender etc) and using machine learning techniques calculate the desired value.

Similar to how correlation will show how two variables are related, association mining attempts to show all of our predictor variables impact on our target variable. This makes

it perfect for medical data as we have numerous predictive variables that all may have a small impact on a patient's health.

2) The best use for a model like this would be prior to the patient getting diabetes, we could attempt to use various data points to predict chance of getting the disease then could impact healthcare outcomes (IE tell them to lose weight or take less medication then potentially less chance of getting diabetes).

For our model however input variables used would be race, age, gender and number of medications to attempt to predict the patient's likelihood of being readmitted. Using the Apriori algorithm we will attempt to find the support (probability that they have both) and the lift. Testing like this until we find most likely scenarios (IE Age & Gender highest likelihood with being readmitted).