Data Scientist test solution, Jully P. P. Pacheco

Chosen problem: Diabetes-related hospital readmission

Knowing the dataset and formulating the scientific problem

As it is an exercise, the packages will no be imported all at once in the first cell as usual, but they will be imported according to the observed needings, and their use will be explained when necessary.

In [1]:
```
import pandas as pd
```

In [2]:
```
raw_data = pd.read_csv('./dataset_diabetes/diabetic_data.csv')

print(raw_data.shape)
print(raw_data.keys())
raw_data.head()
```

```
       'pioglitazone', 'rosiglitazone', 'acarbose', 'miglitol', 'trog
litazone',
       'tolazamide', 'examide', 'citoglipton', 'insulin',
       'glyburide-metformin', 'glipizide-metformin',
       'glimepiride-pioglitazone', 'metformin-rosiglitazone',
       'metformin-pioglitazone', 'change', 'diabetesMed', 'readmitte
d'],
      dtype='object')
```

Out[2]:

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_id	discha
0	2278392	8222157	Caucasian	Female	[0-	?	6	

Since better input data will result in a better overall performance of any model, the first step that will be developed is an overall study of the dataset. From the information on the main source in Strack et. al. 2014 (https://www.hindawi.com/journals/bmri/2014/781670/ (https://www.hindawi.com/journals/bmri/2014/781670/)), and Table 1 of the supplementary

material (https://www.hindawi.com/journals/bmri/2014/781670/tab1/ (https://www.hindawi.com/journals/bmri/2014/781670/tab1/)), we have the following preliminary insights:

- The extraction of this dataset from a bigger one was done by clinical experts who consider that all the 50 features of this specific dataset could be important to evaluate the effectiveness of the medical treatment of patients with diabetes
- In some cases, there are several entries registered for the same patient to the hospital. In the study performed in the source article, multiple entries for the same patient (identified by the "patient_nbr" column) were considered as not independent, and for the sake of using a Multivariate Logistic Regression model, such repetitive entries were removed to leave only one entry for each patient. In that manuscript the discharge motive ("discharge_disposition_id", e.j. patients who died or were sent to a hospice were not considered) was also taken into account to reduce the original dataset from 101766 data rows to around 70000
- In the source article, the features age, gender, race, admission source, discharge disposition, primary diagnosis, the specialty of physician, time in hospital, and result of the HbA1c test were correlated among them and examined to determine the importance of measuring the HbA1c value to reduce hospital readmissions in less than 30 days. The results showed that primary diagnosis is a crucial factor to determine the probability of readmission in less than 30 days, which is also related to the frequency with which HbA1c tests were taken, independently of their result.
- 97 % of the data of feature "weight" is missing, as well as 52 % and 53 % of the "payer_code" and "medical_specialty", respectively

With basis on these insights, and taking into account that there is a previous report where the study of the influence of the measurement of the HbAc1 parameter on early readmissions ("readmission" == '<30') was favored, the formulation of the scientific problem that is intended to solve here is **to perform a generalization of the conclusions in that study by taking all variables (features) into account in our first approach and evaluating which, if some of them do, represent a cause for early readmission, middle time readmission ("readmission" == '<30'), or a thus considered completely effective treatment with no further readmissions ("readmission" == 'NO')**. With this, the first treatment that will be performed on the dataset is described below.

- The features "encounter_id" and "patient_nbr" will be removed for not considering them relevant for a classification problem. In this case, considering that there is already an indicator of the frequency of readmission in each row, there are indicators of health conditions in the year previous to each admission, and the fact that we do not know how all of those evolved with time during the period of the ten years the dataset was taken, we will consider all admissions as independent entries.
- Although the "payer_code" feature could be useful for further statistics, it is not considered as relevant for the specific problem, and as it has more than 50 % missing data, this will also be dropped.
- The medical specialty did not show to be important for hospital readmission in the previous study of this data set, and it has more than 50 % of missing data; thus this feature will also be disregarded.
- In the case of the feature "weight", it will be dismissed because of the huge amount of missing data.

- **Note:** We are defining this problem by thinking about the aim of this exercise and the short time available to complete it. This means the additional literature where this dataset is studied has not been taken into account This is important because there are more than 100 publications that cite the source manuscript, more than 50 % of them in the Information and Computing Sciences area. There, it is probable that this problem has been solved already.

In [3]:
```
## First, some of the given information about the dataset will be confi

print(raw_data['weight'].value_counts())
print(raw_data['patient_nbr'].value_counts())
```

```
?            98569
[75-100)      1336
[50-75)        897
[100-125)      625
[125-150)      145
[25-50)         97
[0-25)          48
[150-175)       35
[175-200)       11
>200             3
Name: weight, dtype: int64
88785891     40
43140906     28
23199021     23
1660293      23
88227540     23
             ..
71081460      1
30060018      1
67443444      1
141344240     1
93251151      1
Name: patient_nbr, Length: 71518, dtype: int64
```

It can be seen in the output of the cell above that 98569 data in the weight variable correspond to a symbol "?". This value represents indeed approximately 97 % of the 101766 total data. It can also be seen that the same patient id appears more than once on several occasions.

In [4]:
```
## Remotion of selected columns:

raw_data.drop(['weight', 'encounter_id', 'patient_nbr', 'payer_code', 'm
raw_data
```

Out[4]:

	race	gender	age	admission_type_id	discharge_disposition_id	admission_so
0	Caucasian	Female	[0-10)	6	25	
1	Caucasian	Female	[10-20)	1	1	
2	AfricanAmerican	Female	[20-30)	1	1	
3	Caucasian	Male	[30-40)	1	1	
4	Caucasian	Male	[40-50)	1	1	
...	...	...	...	...	...	
101761	AfricanAmerican	Male	[70-80)	1	3	
101762	AfricanAmerican	Female	[80-90)	1	4	
101763	Caucasian	Male	[70-80)	1	1	
101764	Caucasian	Female	[80-90)	2	3	
101765	Caucasian	Male	[70-80)	1	1	

101766 rows × 45 columns

Cleaning data

As the features of the dataset were previously selected by experts of the medical area, we will assume that all the remaining features may have some importance for a classification model concerning the effectiveness of overall medical treatment of diabetes in reducing early readmissions, thus, all of them will be processed in order to leave them usable by most of the classification models. The first step will be looking for missing values.

From the information available from the source, it is expected that columns containing the race and primary diagnosis ("race" and "diag_3") have 2 % and 1 % of missing values, respectively. Considering that the complete dataset is large enough, the rows containing missing values in those parameters will also be dropped since that will eliminate only around 3% of the data.

In the output of the cell below it can be seen that the missing values in this dataset are represented by the symbol "?".

In [5]:
```
#raw_data.keys()
print(raw_data['race'].unique())
print(raw_data['diag_3'].unique())
```

```
['Caucasian' 'AfricanAmerican' '?' 'Other' 'Asian' 'Hispanic']
['?' '255' 'V27' '403' '250' 'V45' '38' '486' '996' '197' '250.6' '42
7'
 '627' '414' '416' '714' '428' '582' 'V43' '250.01' '263' '250.42' '27
6'
 '482' '401' '250.41' '585' '781' '278' '998' '568' '682' '618' '250.0
2'
 '305' '707' '496' '599' '715' '424' '518' '553' '794' '411' 'V42' '53
1'
 '511' '490' '562' '250.8' '250.7' '250.52' '784' '491' '581' '420'
'8'
 '724' '730' '789' '131' '250.82' '999' '41' '493' '250.03' '753' '78
6'
 '529' 'E888' '425' '595' '303' '560' '711' '492' '332' '296' '438' '3
62'
 '250.4' '654' '244' 'V70' '737' '625' '681' '250.51' '404' 'V10' '81
0'
 '280' '440' '785' '588' '569' '272' '997' '250.43' '918' '584' '54'
'788'
 '426' '722' '250.92' '196' '461' '535' '787' '891' '284' '458' '648'
 '780' '182' '285' '593' '413' '664' '564' '201' '356' 'V15' '292' '78
2'
 '473' '455' 'E932' '357' '348' '294' '250.23' '459' 'E878' '437' '73
3'
 '507' '525' '250.53' '397' '572' '805' '453' '331' '736' '402' '591'
 '576' '465' '533' '703' '349' '315' '658' '608' '578' '716' '382' '30
0'
 '282' '571' '536' '596' '287' '644' 'V11' '558' 'E885' '162' '198' '2
18'
 '412' '396' 'V14' '570' '433' 'E934' '882' '288' '577' '443' '729' '8
36'
 '295' '799' '281' '304' '153' '410' '616' '250.83' '601' '291' '75'
'512'
 '660' '250.5' '598' '337' '574' '653' 'V58' '311' '415' '386' '602'
'790'
 '112' '873' '620' '436' '70' '155' '138' '663' '530' '710' '42' '342'
 '250.91' 'E884' '515' '307' '704' '728' '731' '583' '238' '441' '293'
 '573' '532' '290' '594' '319' '250.13' '250.12' '519' '346' '380' '13
5'
 '642' '698' '924' '905' 'E933' '555' '309' 'E879' '286' '565' '752'
'580'
 '446' '444' '344' '252' '35' '813' '394' '301' '575' '258' 'V17' '80
2'
 '435' '746' 'V12' '709' '881' 'E935' '139' '250.81' '718' '365' '202'
 '334' '185' '398' 'V44' '517' 'E849' '614' '466' '626' '250.9' '368'
 '605' '883' '289' '478' '617' '429' '442' 'V25' '866' '610' '557' '95
9'
 'E942' '94' '920' '345' '313' '379' '79' '516' '586' '821' '600' '24
2'
 '373' '592' 'V64' '487' '253' '706' 'E947' '117' '340' 'E950' '656'
 'E949' '590' 'V09' '250.22' '934' '694' '203' '250.93' '995' '726' '9
23'
 '958' '275' 'E929' '211' 'V18' 'V66' '199' '665' '53' '279' '522' '79
```

```
1'
 '890' '456' 'E938' 'E816' '122' '721' 'V65' '136' '480' '423' 'E920'
 '793' '647' '537' '351' '845' '336' '274' '719' '945' '434' '494' '22
7'
 '157' '208' '174' 'V57' '812' '734' '150' 'V23' '447' '692' '228' 'V1
6'
 '756' '405' 'E928' '823' '552' '528' '389' '240' '454' '792' '366' 'E
939'
 '907' '270' '310' '266' '387' 'E931' '783' '245' '607' '355' 'E930'
'705'
 '372' '369' '611' '283' 'V46' '110' '867' 'E956' '251' '250.2' '820'
 '712' '695' '567' '343' '723' 'V08' '273' '623' '807' '451' '495' '70
1'
 '34' 'V53' '314' '472' 'E945' '11' '189' '534' '354' '333' 'V54' '27
7'
 '659' '708' '452' '655' '816' '670' '621' '246' '953' '865' 'E817' '6
46'
 '151' '378' '78' '298' '840' '641' '521' '745' '619' '912' '506' 'E90
4'
 '259' 'E870' 'E980' '383' '204' '696' '566' '727' '47' 'E943' '358'
'191'
 '965' '921' '432' '27' 'E861' '758' '477' '524' '751' '652' '556' '18
8'
 '825' '919' '732' '908' '951' '962' '685' 'E850' 'E944' '527' '341'
'693'
 '250.1' 'V49' '860' '323' 'V55' '579' '508' '969' '205' '462' 'E880'
 '680' '697' '826' '200' '457' '717' '738' '742' '735' '235' '308' '72
5'
 '241' '824' '464' '260' '917' '239' '661' '892' '261' 'E883' '943' '7
44'
 'E936' '796' '318' '967' '350' '854' 'E905' '9' '741' 'E941' '170' '6
43'
 '317' '759' '909' 'V22' '831' '713' '180' '801' '360' '359' '501' '33
5'
 '250.11' '306' '811' '690' 'V02' '271' '214' '847' '543' 'V63' '906'
 '842' '686' '445' '808' '861' 'E852' '220' 'E887' 'E858' '915' '970'
 '256' '747' '395' '243' '815' '481' '5' 'E927' '297' '299' '851' '86
4'
 '922' '384' 'E876' '225' '158' 'E937' '871' '88' '966' 'E917' 'E812'
 'V62' 'E924' '604' '233' 'E916' '377' '797' 'V72' '172' '7' '421' '85
2'
 'E819' '972' '916' '956' '3' 'E965' '173' '193' '154' '347' '862' '25
0.3'
 '987' '470' '262' 'E855' '161' '115' '179' '910' '312' '17' '460' '26
5'
 '66' '163' 'V60' '870' 'E906' '514' '944' '844' '417' '152' '183' '99
1'
 '216' '385' '164' '935' '510' '814' '485' '850' '250.21' 'E919' '872'
 '195' '431' '597' '933' '171' '884' '156' '868' '483' 'E815' '542' 'V
61'
 '853' '374' 'E881' 'E882' 'E822' '192' '754' '327' '523' '500' 'V85'
 '992' '657' '684' '603' 'E826' '550' '913' '376' '755' '361' '186' '7
20'
 '250.31' '674' '911' 'E813' '226' '365.44' 'E818' '146' '955' 'E894'
 '475' 'V13' '880' '930' 'E915' '381' '132' '353' '795' '893' 'V01' 'E
853'
 '863' '540' 'E828' '430' '800' 'E865' '148' 'E946' '822' '879' '848'
```

```
 'V86' 'V03' '338' '989' '388' 'E966' '111' 'E922' '123' '757' 'E901'
 '141' '268' 'E892' '649' '702' '948' '223' '484' 'E886' '838' '928'
'236'
 '624' '837' 'E987' 'V07' '841' '622' 'E912' 'E955' '463' 'V06' 'E864'
 '217' '877' '391' 'E825' '952' '669' '875' 'E900' '215' '538' '980'
'834'
 '448' '175' '49' '876' '230' '57' 'E854' '942' '14' '750' '370' '671'
 '971']
```

In [6]:
```
## Filtering rows with "?" in columns "race" and "diag_3":

new_frame1 = raw_data[raw_data['race'] != '?']
new_frame2 = new_frame1[new_frame1['diag_3'] != '?']
new_frame2.reset_index(drop=True, inplace = True)

print(new_frame2.shape)
#print(new_frame2.keys())
```

```
(98144, 45)
```

From the output of the cell above it is found that 3622 rows were removed. That value corresponds to 3.6 % of the original dataset, o.6% more than expected. This puts in evidence that, as the information ontained from the description of the data set in https://www.hindawi.com/journals/bmri/2014/781670/tab1/ (https://www.hindawi.com/journals/bmri/2014/781670/tab1/) only takes into account one significant digit to describe the percentage of missing values, this leaves untracked the missing values if they are present in a quantity less than 1 %, and so, it is possible that there are some missing values in other columns.

First, the columns with the second and third diagnoses will be stripped of possible missing values.

In [7]:
```
new_frame3 = new_frame2[new_frame2['diag_1'] != '?']
new_frame4 = new_frame3[new_frame3['diag_2'] != '?']
new_frame4.reset_index(drop=True, inplace = True)

new_frame4
```

Out[7]:

	race	gender	age	admission_type_id	discharge_disposition_id	admission_sou
0	Caucasian	Female	[10-20)	1	1	
1	AfricanAmerican	Female	[20-30)	1	1	
2	Caucasian	Male	[30-40)	1	1	
3	Caucasian	Male	[40-50)	1	1	
4	Caucasian	Male	[50-60)	2	1	
...	...	...	...	...	...	
98048	AfricanAmerican	Male	[70-80)	1	3	
98049	AfricanAmerican	Female	[80-90)	1	4	
98050	Caucasian	Male	[70-80)	1	1	
98051	Caucasian	Female	[80-90)	2	3	
98052	Caucasian	Male	[70-80)	1	1	

98053 rows × 45 columns

The output of the cell above shows that the total number of rows was reduced from 98144 to 98053. This indicates that in fact there were 91 rows with missing data which were not reported in the original source because they represent less than 1 % of the original dataset.

To evaluate if there are missing values in other columns, the description of the quantity and type of data that, according to the source manuscript, should be present in each column will be used again (see specifically https://www.hindawi.com/journals/bmri/2014/781670/tab1/ (https://www.hindawi.com/journals/bmri/2014/781670/tab1/)).

In [8]:
```
print(new_frame4.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98053 entries, 0 to 98052
Data columns (total 45 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   race                      98053 non-null  object
 1   gender                    98053 non-null  object
 2   age                       98053 non-null  object
 3   admission_type_id         98053 non-null  int64 
 4   discharge_disposition_id  98053 non-null  int64 
 5   admission_source_id       98053 non-null  int64 
 6   time_in_hospital          98053 non-null  int64 
 7   num_lab_procedures        98053 non-null  int64 
 8   num_procedures            98053 non-null  int64 
 9   num_medications           98053 non-null  int64 
 10  number_outpatient         98053 non-null  int64 
 11  number_emergency          98053 non-null  int64 
 12  number_inpatient          98053 non-null  int64 
 13  diag_1                    98053 non-null  object
 14  diag_2                    98053 non-null  object
 15  diag_3                    98053 non-null  object
 16  number_diagnoses          98053 non-null  int64 
 17  max_glu_serum             98053 non-null  object
 18  A1Cresult                 98053 non-null  object
 19  metformin                 98053 non-null  object
 20  repaglinide               98053 non-null  object
 21  nateglinide               98053 non-null  object
 22  chlorpropamide            98053 non-null  object
 23  glimepiride               98053 non-null  object
 24  acetohexamide             98053 non-null  object
 25  glipizide                 98053 non-null  object
 26  glyburide                 98053 non-null  object
 27  tolbutamide               98053 non-null  object
 28  pioglitazone              98053 non-null  object
 29  rosiglitazone             98053 non-null  object
 30  acarbose                  98053 non-null  object
 31  miglitol                  98053 non-null  object
 32  troglitazone              98053 non-null  object
 33  tolazamide                98053 non-null  object
 34  examide                   98053 non-null  object
 35  citoglipton               98053 non-null  object
 36  insulin                   98053 non-null  object
 37  glyburide-metformin       98053 non-null  object
 38  glipizide-metformin       98053 non-null  object
 39  glimepiride-pioglitazone  98053 non-null  object
 40  metformin-rosiglitazone   98053 non-null  object
 41  metformin-pioglitazone    98053 non-null  object
 42  change                    98053 non-null  object
 43  diabetesMed               98053 non-null  object
 44  readmitted                98053 non-null  object
dtypes: int64(11), object(34)
memory usage: 33.7+ MB
None
```

From the previous output, it can be verified that the columns are well classified according to the data type they should contain, and the numerical ones do not contain missing values. Now to check that among the columns containing categorical (object) data, there are no further missing values, it will be checked if the number of "categories" is the right one (for example that the target column has only 3 different values).

In [9]:
```
categorical_data = new_frame4.select_dtypes(include=['object'])
#categorical_data.head()

for column in categorical_data.keys():
    print(column, categorical_data[column].value_counts().count())

categorical_data
```

55	None	None	No	No	...	No	Up	No	No	No
27	None	None	No	No	...	No	No	No	No	No
03	None	None	No	No	...	No	Up	No	No	No
50	None	None	No	No	...	No	Steady	No	No	No
50	None	None	No	No	...	No	Steady	No	No	No
...	...	...	...	...	...	...	...	...	...	...
58	None	>8	Steady	No	...	No	Down	No	No	No
87	None	None	No	No	...	No	Steady	No	No	No
96	None	None	Steady	No	...	No	Down	No	No	No

By comparing the output of the cell above with the source information, it was verified that the right number of categories is present in all columns and so, that there are no missing values in any of them, except for "diag_1", "diag_2", and "diag_3"; in those cases, the number of categories is lower than the one mentioned in the source (i.e. 848, 923, and 954 categories for diag_1, diag_2, and diag_3, respectively). This indicates that in those features some information was indeed lost when the rows with no information in any of the "race" or "diag_1,2,3" columns were removed. However, the 2239 different categories counted here represent only 82 % of the previewed in the source (2725). This means that in some way some of the features that should be different are not being seen as separated ones here. This is maybe a warning at the time of using these data in the classification model.

One last verification to be performed, that could be important for a classification problem, is verifying that there is a similar number of examples of each category to be classified. This is done in the cell below.

In [10]:
```
categorical_data['readmitted'].value_counts()
```

Out[10]:
```
NO     52338
>30    34649
<30    11066
Name: readmitted, dtype: int64
```

The biggest difference between two groups of data is around five times between no-readmitted patients and patients readmitted in less than 30 days. Considering that that difference does not reach even one order of magnitude and that the number of examples of early readmitted patients continues to be large, at this point, it is not considered that treatment of filtering to obtain the same quantity of examples of each category is necessary.

Further data treatment

Transforming categorical to numerical variables, grouping categories in diagnoses columns

As most of the Machine Learning algorithms can not directly process categorical data, and that it is of interest to conserve the largest possible quantity of features, the ones with categorical data types need to be converted to numerical ones. However, considering the large number of different categories present in columns "diag_1", "diag_2", and "diag_3", when compared to the number of categories in all the other columns, another data preprocessing is needed for those cases.

To have a number of categories of at least the same order in "diag_1", "diag_2", and "diag_3" as in the other features, the information from the source regarding the grouping of diseases according to their icd9 codes in Circulatory, Respiratory, Digestive, Diabetes, Injury, Musculoskeletal, Genitourinary, Neoplasms, and Others (see https://www.hindawi.com/journals/bmri/2014/781670/tab2/ (https://www.hindawi.com/journals/bmri/2014/781670/tab2/)) will be used.

In [11]:
```
import numpy as np
```

In [12]:
```
## In the following function, the data in each column is being converte
## intervals to group the features in the 9 types of the disease.


def diagnosis(column):
    new_clas=np.chararray(categorical_data.shape[0]).astype('S15')

    for ind, i  in enumerate(column):
        try:
            i = int(i)
        except:
            try:
                i = int(float(i))
            except:
                i = i[1::]
                i = int(i)

        if ((i >= 390) and (i <= 459) or (i==785)):
            new_clas[ind] = 'Circulatory'
        elif ((i >= 460) and (i <= 519) or (i==786)):
            new_clas[ind] = 'Respiratory'
        elif ((i >= 520) and (i <= 579) or (i==787)):
            new_clas[ind] = 'Digestive'
        elif i == 250:
            new_clas[ind] = 'Diabetes'
        elif ((i >= 800) and (i <= 999)):
            new_clas[ind] = 'Injury'
        elif ((i >= 710) and (i <= 739)):
            new_clas[ind] = 'Musculoskeletal'
        elif ((i >= 580) and (i <= 629) or (i==788)):
            new_clas[ind] = 'Genitourinary'
        elif ((i >= 140) and (i <= 239)):
            new_clas[ind] = 'Neoplasm'
        else:
            new_clas[ind] ='Others'

    return new_clas
```

In [14]:
```
New_diag1 = diagnosis(categorical_data['diag_1'])
New_diag2 = diagnosis(categorical_data['diag_2'])
New_diag3 = diagnosis(categorical_data['diag_3'])
```

In [15]:
```
categorical_data['diag_1'] = New_diag1
categorical_data['diag_2'] = New_diag2
categorical_data['diag_3'] = New_diag3
```

```
<ipython-input-15-91580a722b6b>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  categorical_data['diag_1'] = New_diag1
<ipython-input-15-91580a722b6b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  categorical_data['diag_2'] = New_diag2
<ipython-input-15-91580a722b6b>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  categorical_data['diag_3'] = New_diag3
```

In [16]:
```
categorical_data
```

Out[16]:

	race	gender	age	diag_1	diag_2	diag_3	max_glu_serum
0	Caucasian	Female	[10-20)	b'Others'	b'Diabetes'	b'Others'	None
1	AfricanAmerican	Female	[20-30)	b'Others'	b'Diabetes'	b'Others'	None
2	Caucasian	Male	[30-40)	b'Others'	b'Diabetes'	b'Circulatory'	None
3	Caucasian	Male	[40-50)	b'Neoplasm'	b'Neoplasm'	b'Diabetes'	None
4	Caucasian	Male	[50-60)	b'Circulatory'	b'Circulatory'	b'Diabetes'	None
...	...	...	...	...	...	...	...
98048	AfricanAmerican	Male	[70-80)	b'Diabetes'	b'Others'	b'Circulatory'	None
98049	AfricanAmerican	Female	[80-90)	b'Digestive'	b'Others'	b'Digestive'	None
98050	Caucasian	Male	[70-80)	b'Others'	b'Genitourinary'	b'Others'	None
98051	Caucasian	Female	[80-90)	b'Injury'	b'Others'	b'Injury'	None
98052	Caucasian	Male	[70-80)	b'Digestive'	b'Digestive'	b'Digestive'	None

98053 rows × 34 columns

In [18]:
```
## Proving that the right number of categories are present in each set:

print(categorical_data['diag_1'].value_counts().count())
print(categorical_data['diag_2'].value_counts().count())
print(categorical_data['diag_3'].value_counts().count())
categorical_data.shape
```

```
9
9
9
```

Out[18]: (98053, 34)

Using OneHot encoding to transform columns with no intrinsically numerical values

Now, it is finally possible to transform all the columns with categorical data into numerical ones

that we could use in a model. To avoid obtaining a false hierarchy of data related to the number of categories in each column of the dataset, a OneHot encoder type will be used. To avoid this same problem, the columns "admission_type_id", "discharge_disposition_id", and "admission_source_id" should also be encoded in this way (even if they are of type int64), because the numbers in there represent different categories and not real numerical values with mathematical significance.

In [19]:
```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')
```

At this stage, one may want to start proving how different models behave with this dataset, and if the transformation type used is suitable. Thus initially only the first diagnoses column will be converted to numerical.

In [20]:
```
## Starting with First diagnosis column:

enc_df = pd.DataFrame(enc.fit_transform(categorical_data[['diag_1']]).to
enc_df
```

Out[20]:

	0	1	2	3	4	5	6	7	8
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
98048	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
98049	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
98050	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
98051	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
98052	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

98053 rows × 9 columns

Normalizing data columns containing intrinsically mathematical values

An additional data preprocessing procedure that may serve to improve an algorithm performance is to normalize the values in the columns containing real numerical data, i.e. where the numbers have an actual mathematical significance, so they vary between zero and one.

In [21]:
```
intrinsically_numerical = new_frame4[['time_in_hospital', 'num_lab_proce
                                      'num_medications', 'number_outpatie
                                      'number_inpatient', 'number_diagnos


def normalize(df):
    result = df.copy()
    for feature_name in df.columns:
        max_value = df[feature_name].max()
        min_value = df[feature_name].min()
        result[feature_name] = (df[feature_name] - min_value) / (max_val
    return result
```

In [22]:
```
numeric_normal = normalize(intrinsically_numerical)
numeric_normal
```

Out[22]:

	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpa
0	0.153846	0.442748	0.000000	0.2125	0.000
1	0.076923	0.076336	0.833333	0.1500	0.047
2	0.076923	0.328244	0.166667	0.1875	0.000
3	0.000000	0.381679	0.000000	0.0875	0.000
4	0.153846	0.229008	1.000000	0.1875	0.000
...	...	...	...	...	
98048	0.153846	0.381679	0.000000	0.1875	0.000
98049	0.307692	0.244275	0.500000	0.2125	0.000
98050	0.000000	0.396947	0.000000	0.1000	0.023
98051	0.692308	0.335878	0.333333	0.2500	0.000
98052	0.384615	0.091603	0.500000	0.0250	0.000

98053 rows × 8 columns

Technical testing of a trial training dataset

Below, a first dataset that will be used for the first test of algorithm performance is defined. This contains all the intrinsically numerical data and the encoded data for the primary diagnostic.

Note that this is done as an arbitrary test, and the features were not chosen by following any data or context-based criterium, and so it is not intended as the definitive training dataset.

In [23]:
```
first_input = numeric_normal.join(enc_df)
first_input
```

Out[23]:

	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpati
0	0.153846	0.442748	0.000000	0.2125	0.000
1	0.076923	0.076336	0.833333	0.1500	0.047
2	0.076923	0.328244	0.166667	0.1875	0.000
3	0.000000	0.381679	0.000000	0.0875	0.000
4	0.153846	0.229008	1.000000	0.1875	0.000
...	...	...	...	...	
98048	0.153846	0.381679	0.000000	0.1875	0.000
98049	0.307692	0.244275	0.500000	0.2125	0.000
98050	0.000000	0.396947	0.000000	0.1000	0.023
98051	0.692308	0.335878	0.333333	0.2500	0.000
98052	0.384615	0.091603	0.500000	0.0250	0.000

98053 rows × 17 columns

Encoding the target dataset

The target dataset can be encoded in a simpler way by using the LabelEncoder tool of scikit learn. This is possible because a classification algorithm assumes the fact that the target is formed by different unrelated categories and so numerical values are not taken as a sequential series of numbers.

In [24]:
```
from sklearn.preprocessing import LabelEncoder
lb_make = LabelEncoder()

encoded_target = pd.DataFrame({'readmitted': lb_make.fit_transform(categ
print(type(encoded_target))
encoded_target.head()
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[24]:

	readmitted
0	1
1	2
2	2
3	2
4	1

Joining the complete trial dataset:

In [27]:
```
complete_num_set = pd.concat([first_input, encoded_target], axis=1)

complete_num_set.info()
print(complete_num_set.shape)
complete_num_set
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98053 entries, 0 to 98052
Data columns (total 18 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   time_in_hospital    98053 non-null  float64
 1   num_lab_procedures  98053 non-null  float64
 2   num_procedures      98053 non-null  float64
 3   num_medications     98053 non-null  float64
 4   number_outpatient   98053 non-null  float64
 5   number_emergency    98053 non-null  float64
 6   number_inpatient    98053 non-null  float64
 7   number_diagnoses    98053 non-null  float64
 8   0                   98053 non-null  float64
 9   1                   98053 non-null  float64
 10  2                   98053 non-null  float64
 11  3                   98053 non-null  float64
 12  4                   98053 non-null  float64
 13  5                   98053 non-null  float64
 14  6                   98053 non-null  float64
 15  7                   98053 non-null  float64
 16  8                   98053 non-null  float64
 17  readmitted          98053 non-null  int64  
dtypes: float64(17), int64(1)
memory usage: 13.5 MB
(98053, 18)
```

Out[27]:

	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpat
0	0.153846	0.442748	0.000000	0.2125	0.000
1	0.076923	0.076336	0.833333	0.1500	0.047
2	0.076923	0.328244	0.166667	0.1875	0.000
3	0.000000	0.381679	0.000000	0.0875	0.000
4	0.153846	0.229008	1.000000	0.1875	0.000
...	...	...	...	...	
98048	0.153846	0.381679	0.000000	0.1875	0.000
98049	0.307692	0.244275	0.500000	0.2125	0.000
98050	0.000000	0.396947	0.000000	0.1000	0.023
98051	0.692308	0.335878	0.333333	0.2500	0.000
98052	0.384615	0.091603	0.500000	0.0250	0.000

98053 rows × 18 columns

Separating train and validation sets

To take the metrics of implementing a classification algorithm, the dataset is split into four sets: x_train, y_train, x_val, and y_val. In this case, 30 % of the data will be let as the validation set. The original dataset is shuffled before separating the validation set to avoid possible bias induced by intentionally or unintentionally ordered data in the original dataset.

In [25]:
```
from sklearn.model_selection import train_test_split
```

In [28]:
```
training, test = train_test_split(complete_num_set, test_size=0.3, randc
```

In [29]:
```
x_train = training.drop('readmitted', axis=1)
y_train = training['readmitted']

x_val = test.drop('readmitted', axis=1)
y_val = test['readmitted']

print(len(x_train), len(y_train))
print(len(x_val), len(y_val))
```

```
68637 68637
29416 29416
```

Choosing a trial classification algorithm

Decision Tree

Classification problems where having the importance of variables is especially relevant are commonly treated with a Random Forest type algorithm because those parameters are relatively simple to obtain from there. Here the performance of a simpler algorithm of Decision Tree algorithm is done visualizing to test a Random Forest model later.

In [30]:
```
from sklearn.tree import DecisionTreeClassifier
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
```

In [31]:
```
drugTree.fit(x_train, y_train)
```

Out[31]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In [32]:
```
predTree = drugTree.predict(x_val)
```

In [33]:
```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_val, predTr
```

```
DecisionTrees's Accuracy:  0.5618710905629589
```

Initially, it is observed that the algorithm runs fastly (less than one minute) with these parameters and data, but has an unacceptable accuracy. However the technical test can be considered successful, and now it is possible to concentrate on encoding and including the

remaining features, make a rigorous selection of them to form the input dataset, change or tuning the classification model.

Important Review on the proposed Scientific Problem

At this point, having a better understanding and familiarity with the complete dataset, both in the technical and content aspects, and having into account the poor performance of the model (which can be normal in a first trial), it is time to revisit the feasibility of solving the originally proposed problem. From the acquired knowledge of the data, it is reasonable to think that the initially proposed problem can be too general, and given the nature of the data acquired that were taken without any specific purpose, taking the complete dataset will not necessarily help to solve it. So it important to have in mind that it is possible that we have to instead, define a very specific problem, as for example:

Given the correlation found in the source manuscript, determine if there is not only a correlation but a causality between taking the HbA1c test and a probable No-readmission, i.e, given the result of the test, to infer if that induces the physician to administrate some specific medicine that could result in the main factor to determine the effectiveness of the treatment.

Until now, the dataset was let the most complete possible, and so the solution of several different specific problems can still done; but it is important to know that that problem should be defined before any further data treatment can be performed, and that will depend on what is the specific question one wants to answer. If we want to know if a first hospital entry for mental disease can provoke a diabetic crisis in less than 30 days after leaving the hospital, we should filter and separate the data containing only the information related to those specific diseases. Otherwise, other variables or data points would only increase noise thus avoiding the algorithm to learn.

So far, after proving better parameters in the model, and different input datasets, the way I would continue to deal with the given dataset is to restart by defining a very specific problem, filtering the data that is certainly related to that problem (remove the data introducing bias or noise), from the datasets here called categorical_data (the encoding form of it), numeric_normal, and encoded_target, and only then put it as the input of a model.

Testing a neural network (this section can be disregarded of the evaluation process)

Neural networks are a good option for having into account possible correlations between variables, but this was not taken as a first option because normally they take more time to run, and the importance of variables is not obtained straightforwardly. However, as a matter of curiosity of performance in the same dataset, a very simple NN was tested, but the results of the cells below can be desconsidered in the evaluation of this excersise.

In [31]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [33]:
```
def get_model():
    model = Sequential([
        Dense(128, activation = 'relu', input_shape = (x_train.shape[1],
        Dense(128, activation = 'relu'),
        Dense(128, activation = 'relu'),
        Dense(128, activation = 'relu'),
        Dense(128, activation = 'relu'),
        Dense(128, activation = 'relu'),
        Dense(1)
    ])
    return model

model = get_model()
```

In [34]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               2304
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_3 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_5 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 129
=================================================================
Total params: 84,993
Trainable params: 84,993
Non-trainable params: 0
_________________________________________________________________
```

In [35]:
```
model.compile(optimizer='adam', loss = 'mse', metrics=['mae'])
```

In [37]:
```
history = model.fit(x_train, y_train, epochs = 100, validation_split=0.1
```

In [38]:
```
print (history.history['loss'])
```

```
[0.4674466550350189, 0.4416429400444031, 0.4394891858100891, 0.4385744
333267212, 0.4380425810813904, 0.43836236000061035, 0.436796456575393
7, 0.43656378984451294, 0.4360140562057495, 0.43569841980934143, 0.435
76231598854065, 0.4350576102733612, 0.4353143274784088, 0.434435844421
3867, 0.4338119626045227, 0.43391644954681396, 0.4330251216888428, 0.4
330905079841614, 0.43255844712257385, 0.43280985951423645, 0.431755721
5690613, 0.43146899342536926, 0.4310721755027771, 0.4308125078678131,
0.430289089679718, 0.42957913875579834, 0.429609090089798, 0.428672969
3412781, 0.4284752607345581, 0.42801612615585327, 0.4273175895214081,
0.42738768458366394, 0.4263532757759094, 0.42575931549072266, 0.425899
56521987915, 0.42495423555374146, 0.4238167703151703, 0.42331567406654
36, 0.4228644371032715, 0.4217927157878876, 0.4218646287918091, 0.4207
2710394859314, 0.420585960149765, 0.41920140385627747, 0.4195270836353
302, 0.4183397591114044, 0.4175196588039398, 0.4168720245361328, 0.415
7051742076874, 0.41539400815963745, 0.41424039006233215, 0.41367465257
644653, 0.41282904148101807, 0.4119727909564972, 0.4105384647846222,
0.41012293100357056, 0.40932419896125793, 0.4085913896560669, 0.407853
5735607147, 0.4066597819328308, 0.4060578942298889, 0.404712527990341
2, 0.40426650643348694, 0.4031219482421875, 0.4017867147922516, 0.4011
984169483185, 0.40032878518104553, 0.39980781078338623, 0.398494511842
72766, 0.39715611934661865, 0.39668041467666626, 0.3955700993537903,
0.394748330116272, 0.3927306234836578 4, 0.3924490809440613, 0.39201068
87817383, 0.39084330201148987, 0.3893015682697296, 0.3886492848396301,
0.38808706402778625, 0.38681891560554504, 0.38626986742019653, 0.38479
73942756653, 0.38341793417930603, 0.38338956236839294, 0.3819813132286
072, 0.3807101249694824, 0.3800809979438782, 0.37904495000839233, 0.37
726056575775146, 0.37723276019096375, 0.37623125314712524, 0.375523418
1880951, 0.37453681230545044, 0.3737413287162781, 0.3736761808395386,
0.37145036458969116, 0.3724038004875183, 0.371381014585495, 0.37007069
58770752]
```

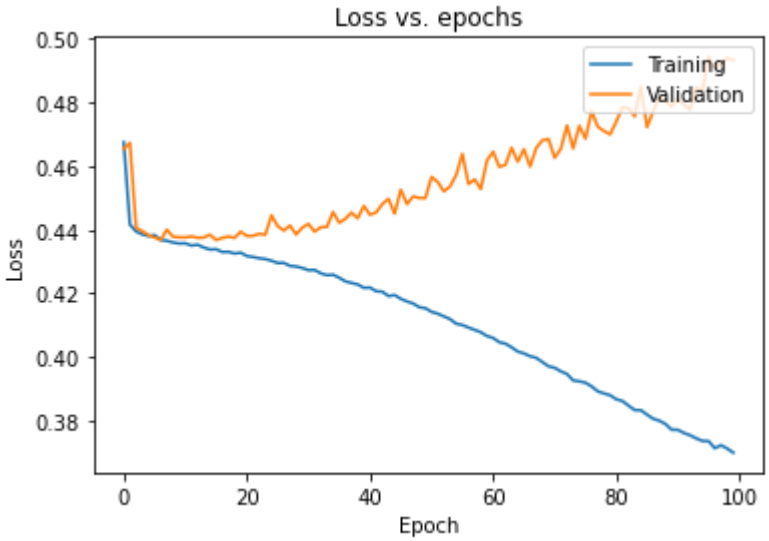
In [40]:
```
model.evaluate(x_val, y_val, verbose=2)
```

```
920/920 - 1s - loss: 0.4849 - mae: 0.5822
```

Out[40]:
```
[0.48485907912254333, 0.5821600556373596]
```

In [41]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [42]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



The current form of this model is suffering from overfitting, and so one should introduce some regularization parameters if it is intended to pursue better results.

In [ ]:
```
```