**Description of the Problem**

The goal of this report will be to analyse the diabetes dataset with the outcome being to create an algorithm to process the data and predict whether a patient being admitted into hospital for a diabetic related issue is going to be re-admitted again.

The dataset contains 50 attributes and 101766 instances. It contains a class attribute that contains the information of whether a patient was re-admitted to the hospital before or after 30 days, using the syntax “>30”, “<30” or “NO”. Of this data, 54864 were not re-admitted and 46902 were. This was discovered with the howManyReadmitted function in the script illustrated in figure 1 with the labelled results.

An important issue with the data is the missing values illustrated with a “?”, which is a string data type. This is a problem for the Pandas library to recognise as missing data as it is not in the format of NaN, which is what pandas will recognise as missing data. This will need to be converted. This was implemented with the setUpMissingData function and is shown in figure 1.

The data contains 192849 missing values, which is a small percentage and means that removing them will not make much difference to the analysis. This was displayed with the howManyNaN function in figure 1. The handling of the missing values will be discussed in the following section under the heading “Data Pre-Proccessing”.

Another important fact is the missing values are exclusively from seven columns which will be investigated further in the following section for which to remove and keep based on the accuracy of the machine learning algorithm. This was implemented with the displayMissingValues and is also in figure 1.

The dataset also contains 2 attributes (examide and Citoglipton) that have a single unique data value. These should be removed from the dataset to improve performance because a single value does not contribute information to the machine learning data model. This was observed with the isUnique function illustrated in figure 2, which shows the total the different types of value in an attribute.

The last aspect of the initial format of the data is an additional information table of number coded values for the attribute’s admission\_type\_id, discharge\_disposition\_id, and admission\_souce\_id. An interesting code from the discharge\_displostion\_id is the “Expired” option which means that the person died, so this should be removed from the data as the person will be contributing unfairly to the not re-admitted class.

|  |  |  |
| --- | --- | --- |
| Diabetes.csv | Count | % |
| No of Attributes | 50 |  |
| No of Instances | 101766 |  |
| Re-Admitted | 46902 |  |
| Not Re-admitted | 54864 |  |
| Missing Values | 192849 |  |
| Single Value Attributes | 2 |  |

The dataset shows that there is a fair balance between Re-admitted and not readmitted after adding the different categories of less than and more than 30 days for the re-admitted value which indicates a cause for exploring the problem.

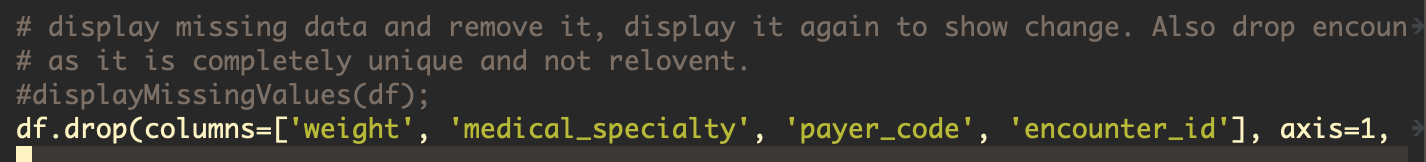
**Data Pre-Processing**

Before applying the data pre-processing to the project, the initial script for gathering information of the data, “data\_analysis.py”, was rewritten as a module to be used in the final script that the machine learning algorithm is implemented in. This is to make the code more readable and to keep the print logs to a minimum.

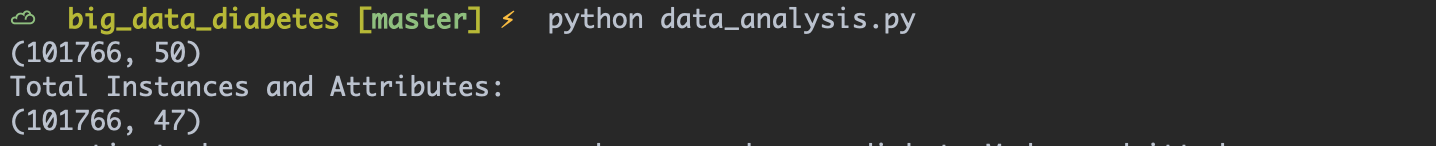
Handle Missing Values

The very first aspect of this process was handled in the first phase with the setupMissingData function. This uses the Pandas replace function to find all instances values in the data that are equal to a “?” and replaces them with the value NaN. The reason this was done in the first phase was the Pandas library functions will only recognise missing values as the data format NaN, so in order to count missing values accurately this had to be implemented right away. Using the knowledge gained from the previous section, the output shows the weight attribute having 98569/101766 missing values, which is …%. This is almost all missing data so this attribute can be completely removed. The payer code and medical\_specialty attributes are 40256 and 49949 which is a little under 50%, but after some analysis the payer\_code and medical\_specialty are not contributing much to the analysis so it can be removed. The encounter\_id is a unique id for each incident which is not relevant, so this is removed. Details are listed below.

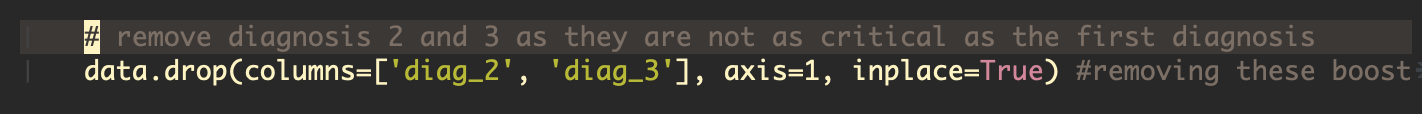
Code



Console Output



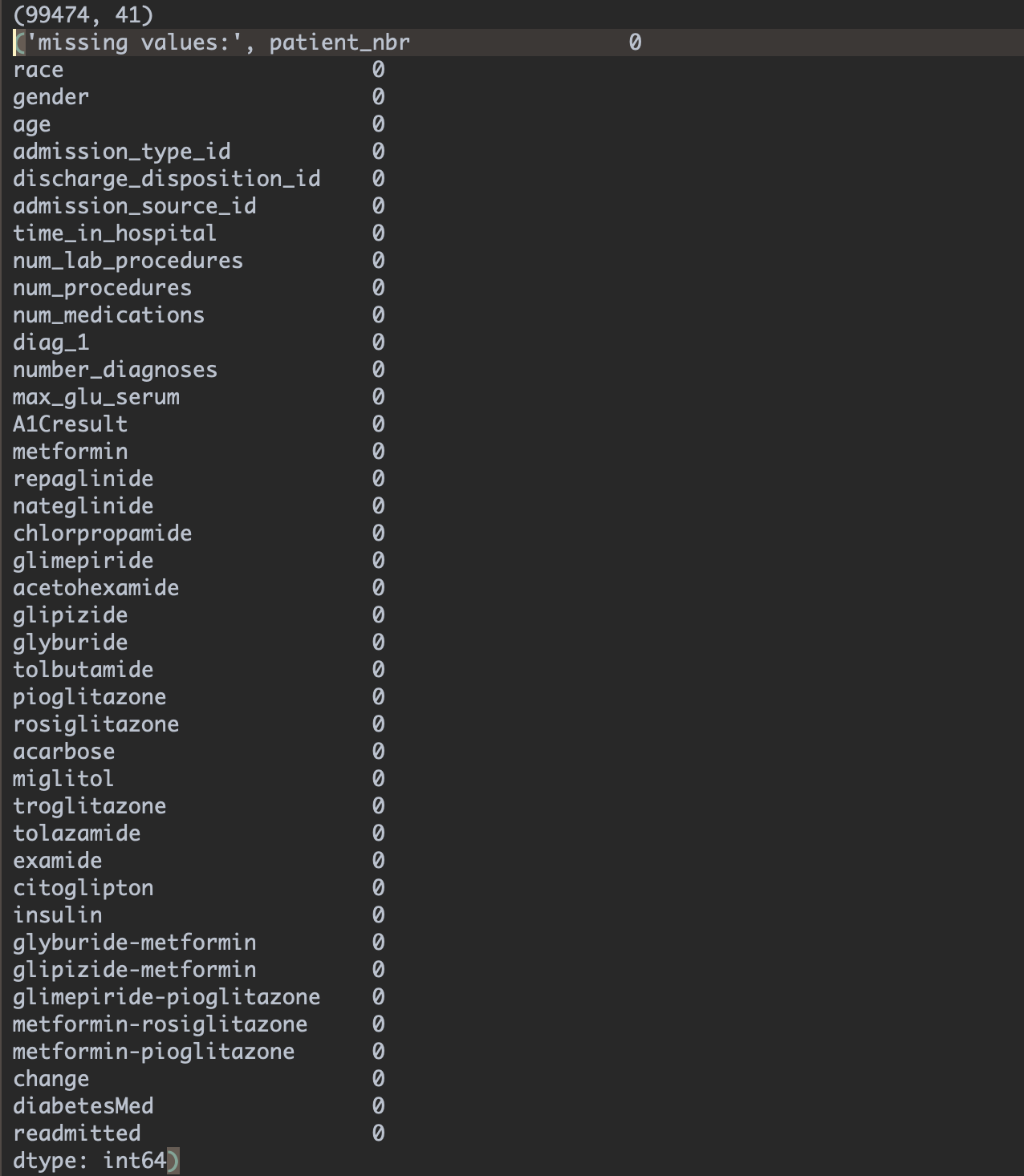
Another two attributes that are to be removed are the diag\_2 and diag\_3 which are the second and third diagnosis. Although the missing values in these columns are small, they are providing mostly the same information and are affecting the performance exponentially as each attribute contains a large number of categories, so it is best to remove these.



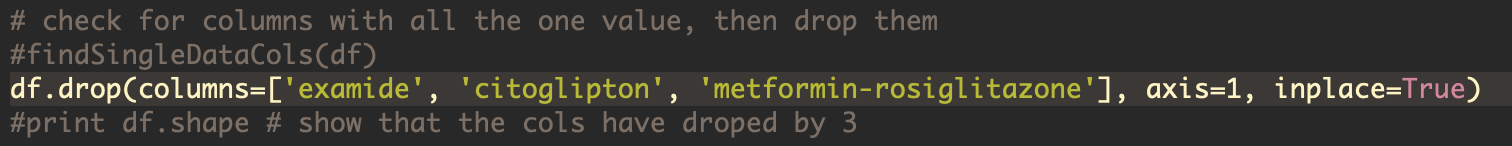
The next step is to remove the missing entries in the row that are in the smallest percentile, which are the race and diag\_1. As mentioned in the previous section and in figure 1, these attributes have 2273 (race) and 21 (diag\_1) which is so small that removing them will be the easiest way to handle the missing values. There are many other suitable ways to handle missing values in pre-processing such as imputation, where maths formulas such as standard deviation can be used to replace missing values. However in this case the data is categorical so the most sensible approach is to remove. This can be done simply with the dropna function from the Pandas library which is illustrated below.



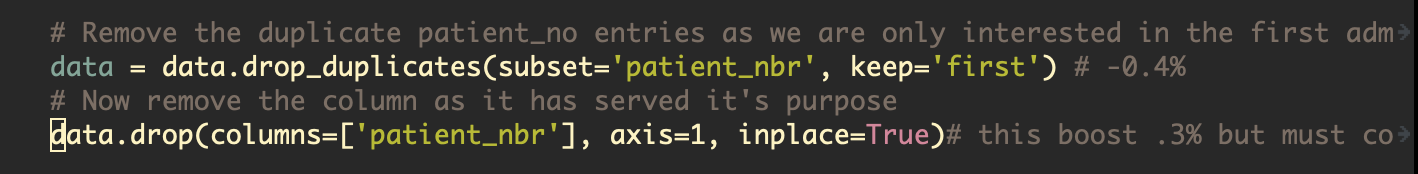
The output from the console has now been changed from (101766, 50) to (99474, 41) and there is now no missing values. Details are listed below.



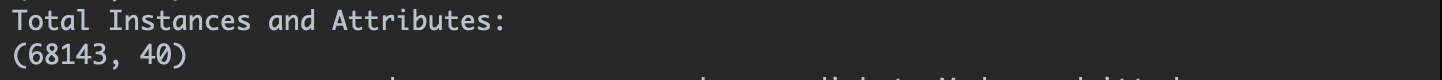
The next step is to remove the columns discovered with al one single value. This is done with the same drop command as before.



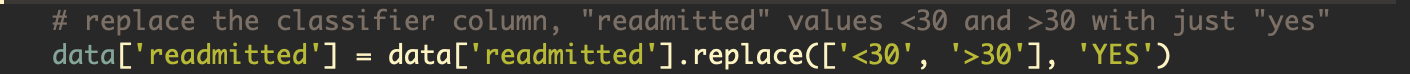
Next, as discovered in the data from figure 1, the patient\_nbr attribute has a large number of unique values but it is not 100%, which means that the data has return patients in it and it is not fair to use them in a machine learning algorithm as the numbers could potentially be used to “overfit” the data model, (more on this in the next section). This is handled by first looking up the patient\_nbr and removing the duplicates and then removing the column itself as it has served it’s purpose at this point, which drops the instances substantially to 68143.



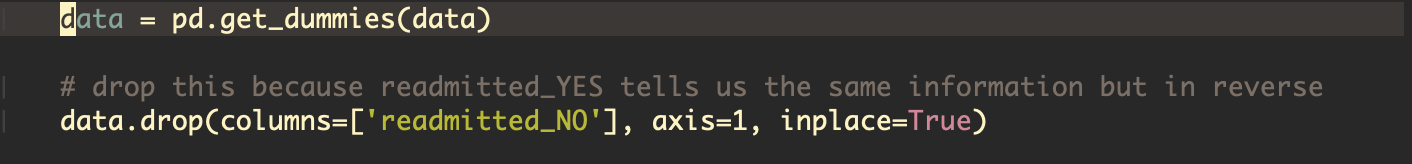
Console output



Now that the main data has been fully processed, the classifier column need to be filtered to have 2 simple values of ‘YES’ and ‘NO’. This is done by changing the more than 30 days value and less than 30 days values to just being ‘YES’ so that the algorithm can classify the data much easier. This is illustrated below.



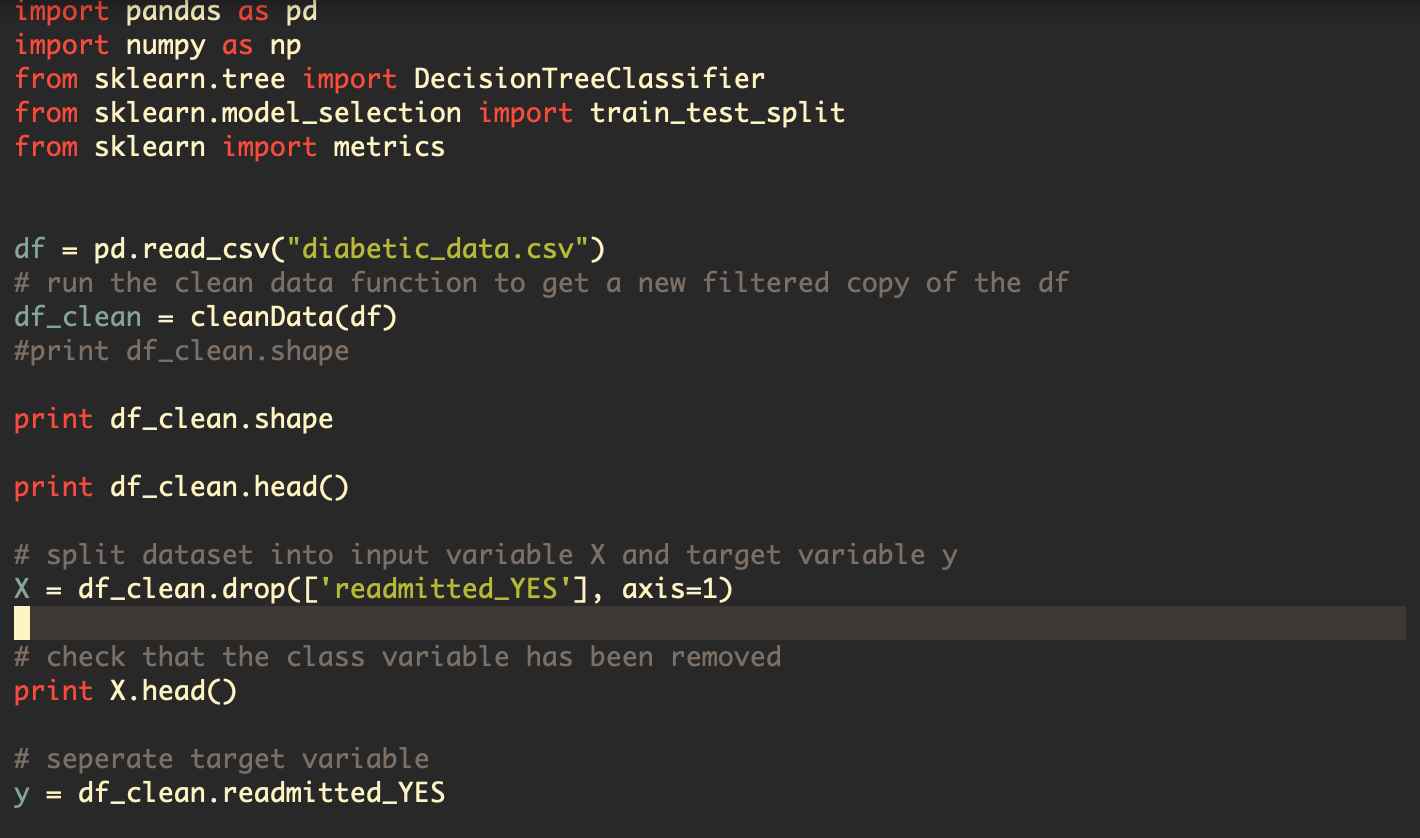
When data is category based in a dataset, the data must be converted either a numerical or binary format so that the machine learning library can understand the data. There are two common methods to achieve this which are the label encoder from sklearn library and the get dummies from the Pandas library. The problem with label encoder is that it uses integers to replace the values which can be misinterpreted and compared as values more or less than each other which can cause errors. The latter is more efficient as it creates new categories from each value in the attribute in the data and uses a binary data format of 1 or 0 to illustrate having that value. The pros and cons of this will be discussed later. This is shown below where the bottom line is removing the newly created readmitted\_NO attribute that will not be needed as we now have readmitted\_YES providing the full information.



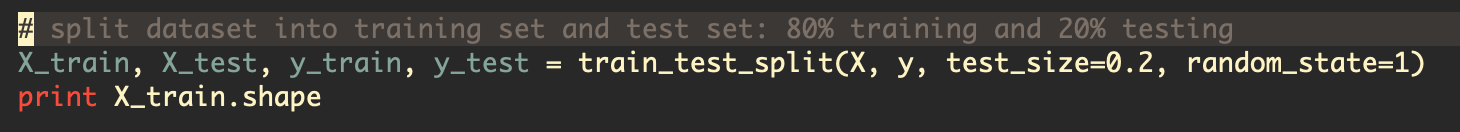
**Model Construction**

The chosen machine learning algorithm is a decision tree as it is a known and effective algorithm for processing data that is category based. Compared to other algorithms, a decision tree is easier to set up and requires less pre-processing to implement. The main reason for this choice is that decision trees are known to perform well on large datasets. The disadvantages of this choice is that they are prone to “overfitting” and require careful tuning and data pre-processing, more on this in the next section.

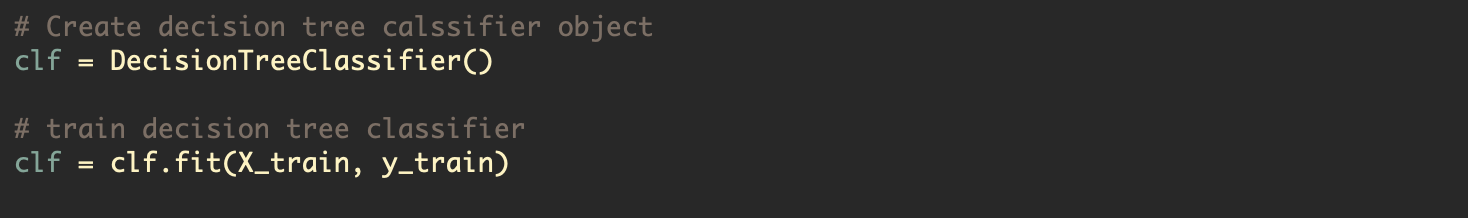
For constructing the data model, an iterative process will be used to achieve the best results possible. To start with, the pre-processed data is imported into the script and the decision tree module is loaded in from the sklearn library. The dataset is then split by removing the classifier attribute and assigning the data to variable X, then setting variable y to the readmitted\_YES attribute which is the target variable.



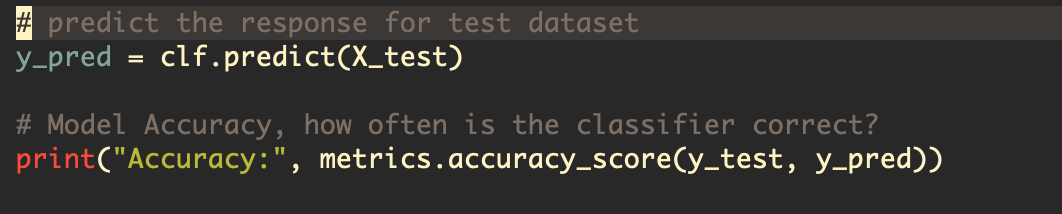
Now the data set can be split into training and testing data using 80% train, 20% split. The training data is what is used to build the algorithm and the test data is what is used for testing the algorithm’s ability to predict the classification. This was achieved using the sklearn library’s train\_test\_split function and setting the parameters to the appropriate data and setting the last parameter to random\_state to ensure the testing is split into random groups for better accuracy.



Now that the train and test data is setup, the decision tree model can be built using the decisionTreeClassifier function which creates the decision tree object from sklearn. Then the decision tree can be built using the training data which is passed in through the fit function. The following split shows the following numbers



Lastly, the model is trained and the predict function can be used on our model to make predictions on our test data. Then the accuracy can be tested with the accuracy\_score function which tests how often the classifier is correct.



Results pre tuning



**Tuning the Data Model**

For the tuning of the model of a decision tree the “impurity” of each split is measured using a technique called information gain. This technique uses a mathematical formula called entropy which measures the amount of information gained from a split. By default, the decision tree will create a tree that continues to expand until all the leaves are pure which results in a large tree. This is bad because it means the tree is at risk of being “overfitted”, which is when the model fits the data too well and will not perform well in the real world. Another risk is “underfitting”, which is when the depth of the tree is too short and not enough data has been captured.

In order to avoid these issues, a trial and error approach was implemented using entropy to find the best splits to build the tree from starting with a max depth of three and building up to 7, where after 7 the accuracy of the model began to decline.

**Testing Results**

Max depth 3. Max depth 4



Max depth 5 Max depth 6 - Best



Max depth 7 - Declining



The results show that with careful tuning of the model the accuracy can be increased from 56.6% to 62.5% (rounded to 1 decimal place).

**Discussion**

The significance of pre-processing the data has made a large impact on the model. The first point of data pre-processing being vital was that the Pandas library can only do so much with non-processed data, i.e number data types written as “?” can’t be processed as missing data. Another important aspect was the sklearn library was only compatible with number format data, so for any form of functionality the data pre-processing of strings to numbers was essential. Pre-processing also can have a large effect on performance, as discussed in this report previously. The reason was a factor was the dataset had a large amount of missing values and irrelevant data that hurt the accuracy of the algorithm as well as slowing down performance by including extra work of processing this extra data.

Tuning a model after creation is another vital aspect of machine learning. Without this step, a model will never perform at its best for speed and accuracy and this was a factor with this project as the size of the tree was very large and inaccurate before the tuning process. The balance of training data vs testing is also a factor as this had a large effect on the accuracy of the model. For example a smaller split, like 70%-30% would likely have been less accurate with smaller training data, but might perform better when used with real world data to actually predict/classify a dataset. Overall, after strong pre-processing and careful tuning, the accuracy and speed of the model was greatly improved.

Figure 1.

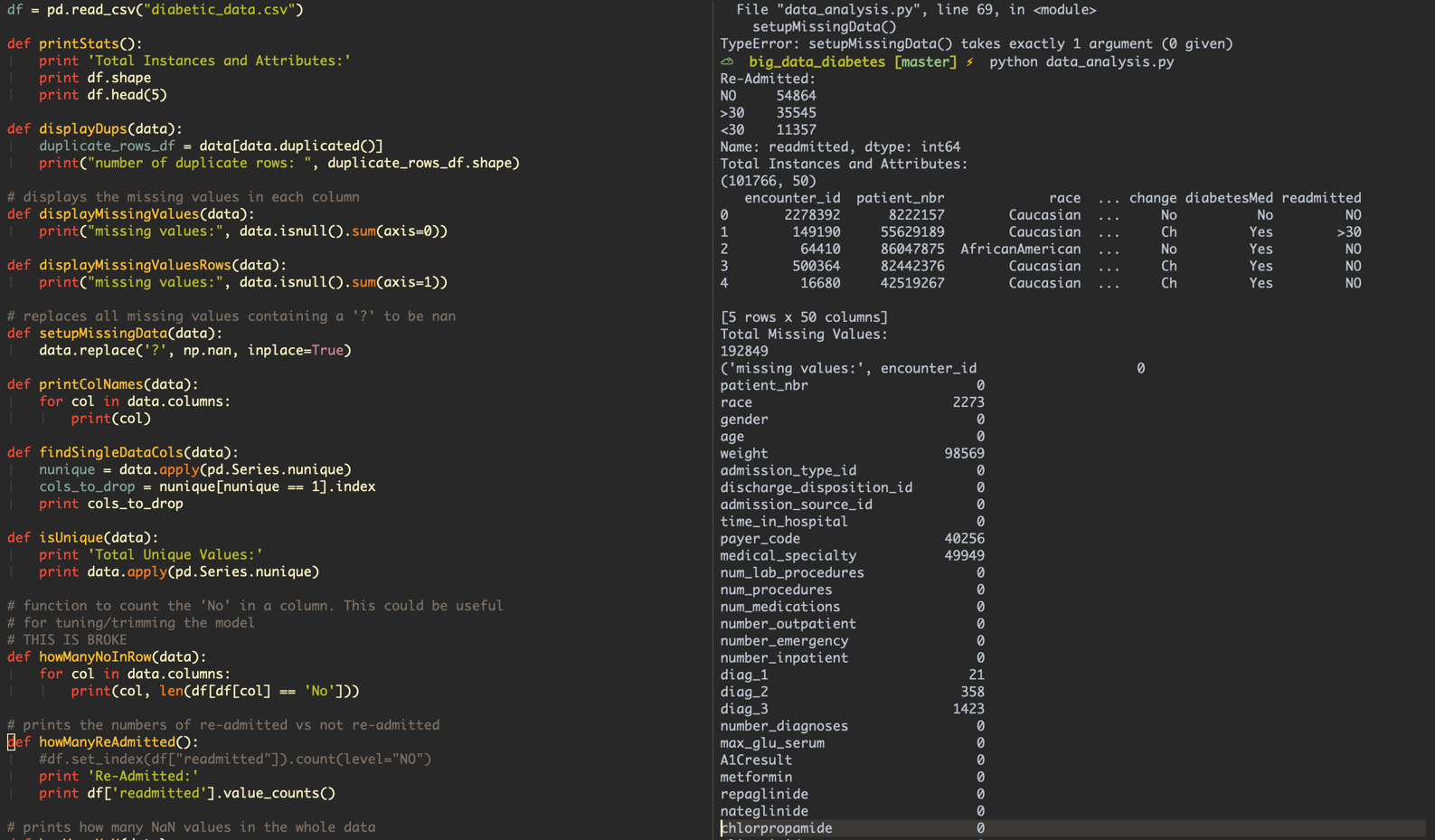


Figure 2.

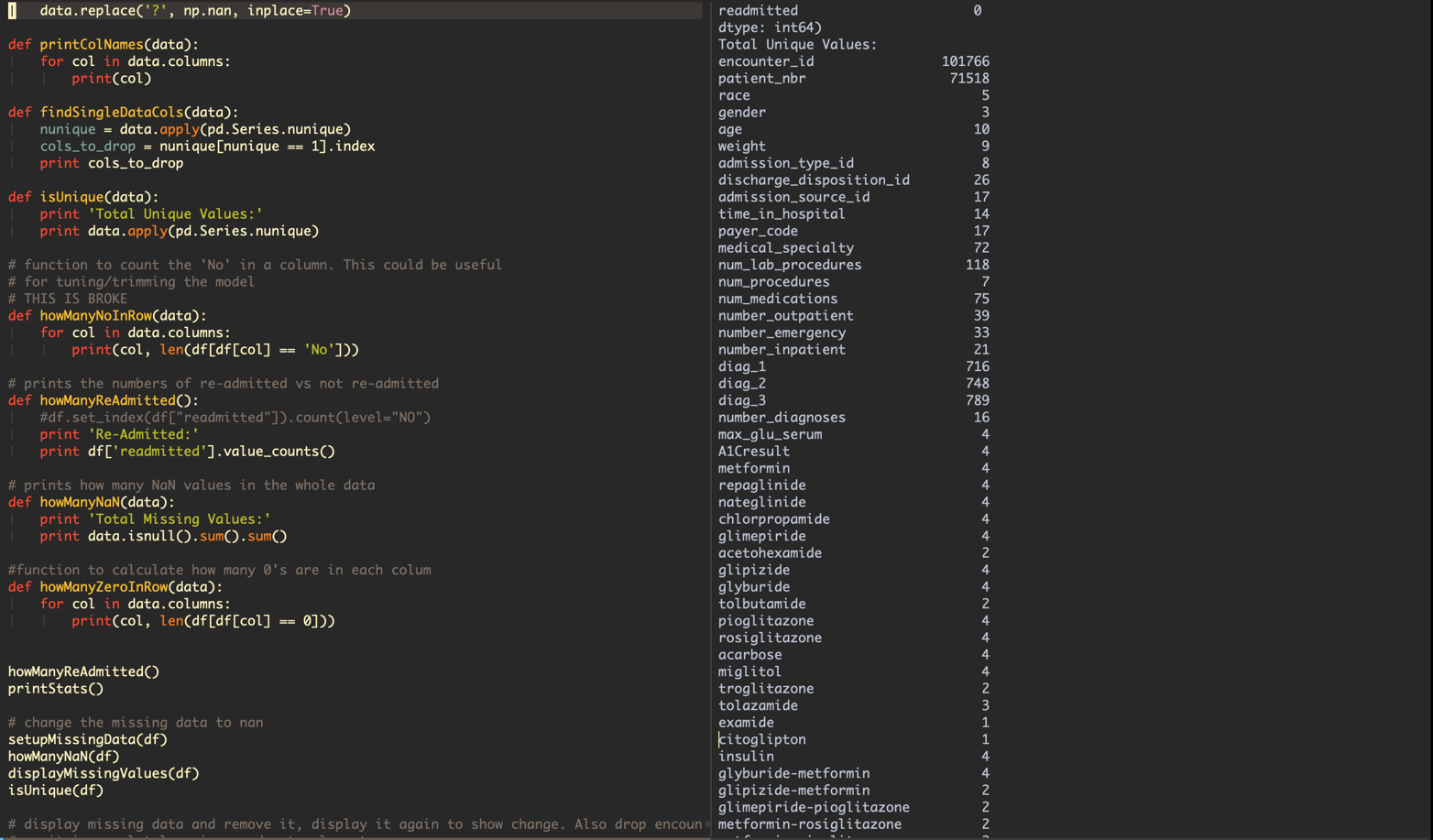


Figure 3.

