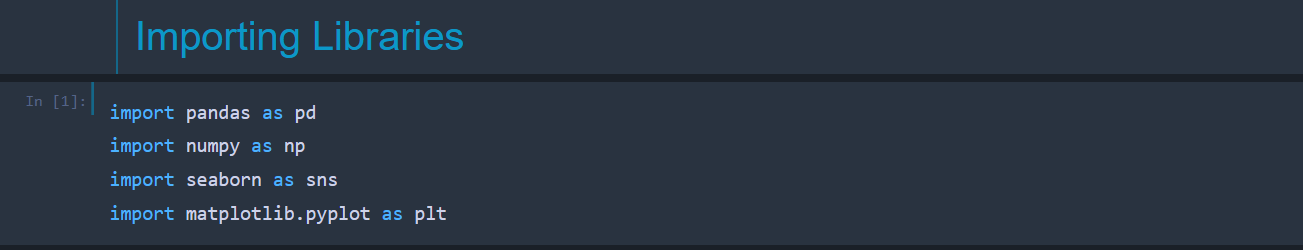
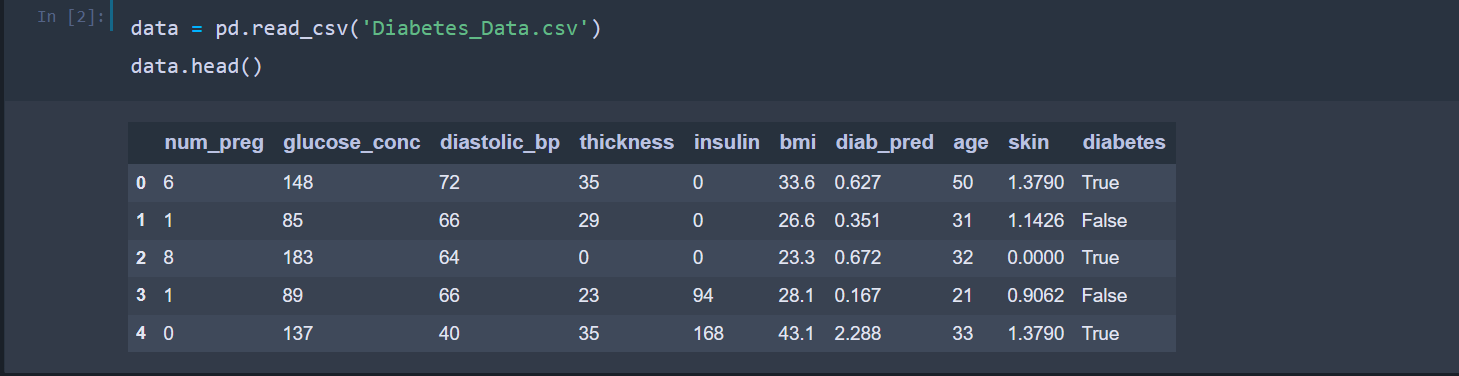
**DIABETES PREDICTION**

We all would have met a **Diabetic** person at one point or the other in our lives. So, what exactly does **Diabetes** mean. In simple words it is a health condition which affects how our body turns food into energy. The sugar contained in our food gets converted to energy by the help of a **hormone** called **Insulin.** When a person is affected with **Diabetes,** either the person’s body does not make enough **Insulin** or it does not make use of the available **Insulin** in an effective way leading to high blood sugar levels.

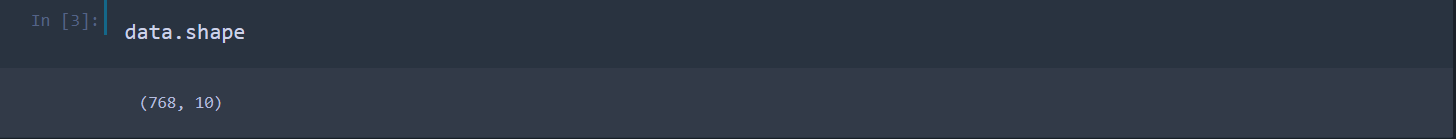
In this section we will be dealing with the data related to **Diabetes.** Using the available data, we will predict whether a patient will be affected with **Diabetes** or not. We go through the complete process of Data Analysis, starting from defining our problem statement, to the last step i.e., getting the predictions for patients.

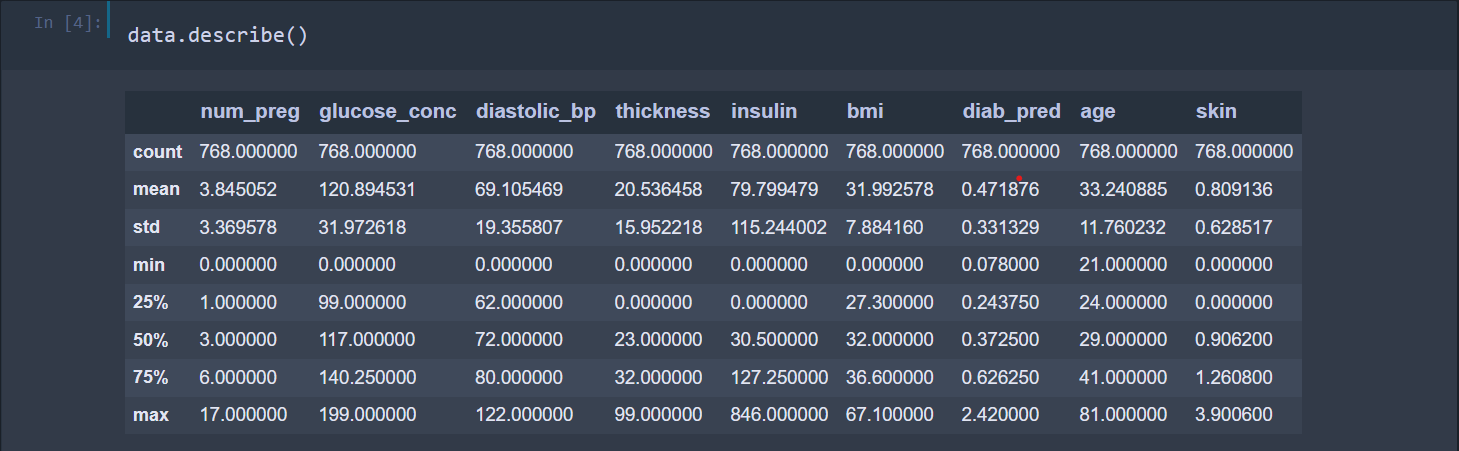
We can start off the usual way, i.e., by importing all the necessary libraries like pandas, NumPy, matplotlib, seaborn etc in python.

Next we load the dataset into our **Jupyter Notebook** using the **read\_csv()** method similar to the previous problem statements. We can use the **head()** method to have a look at the dataset and a few rows of data as well.

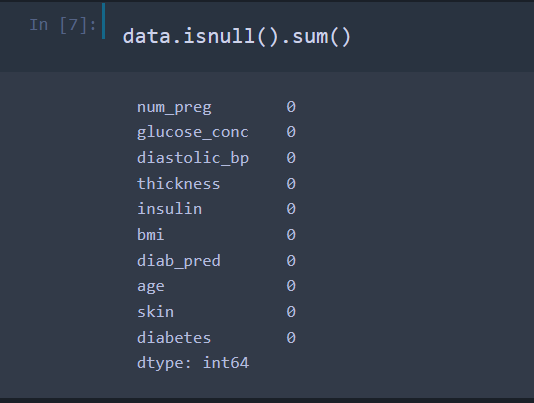


It is observed that we have 10 columns as shown in the above image. The first **9** columns contain information regarding the patient’s health. Details like **age, insulin content in blood, glucose concentration** are available in the dataset. The last column contains information regarding whether a patient is **diabetic** or not. The dataset contains **768** rows in total.



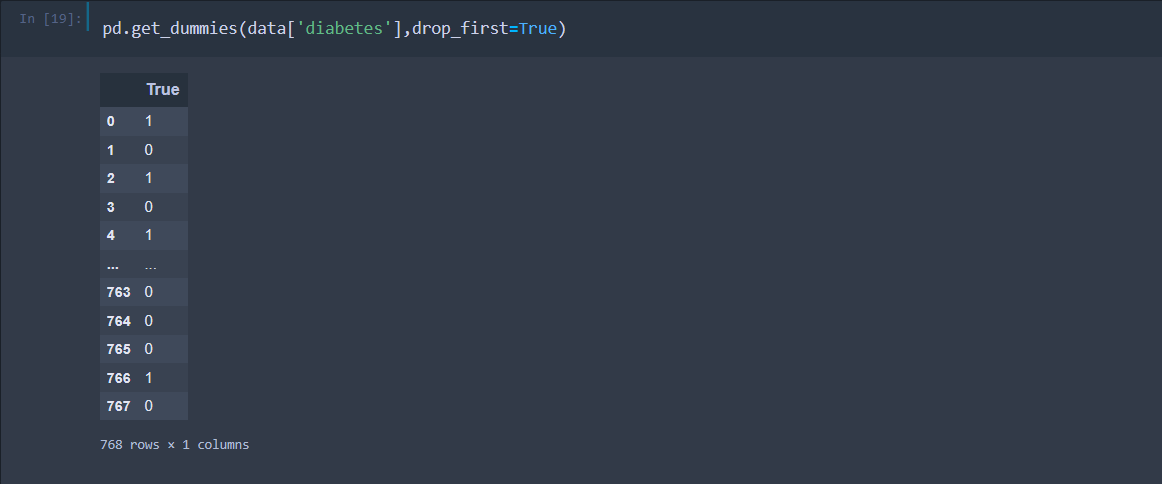
The **describe()** method can be used to derive some **Statistical** insights (**mean, min, max** etc) about various features of the dataset.

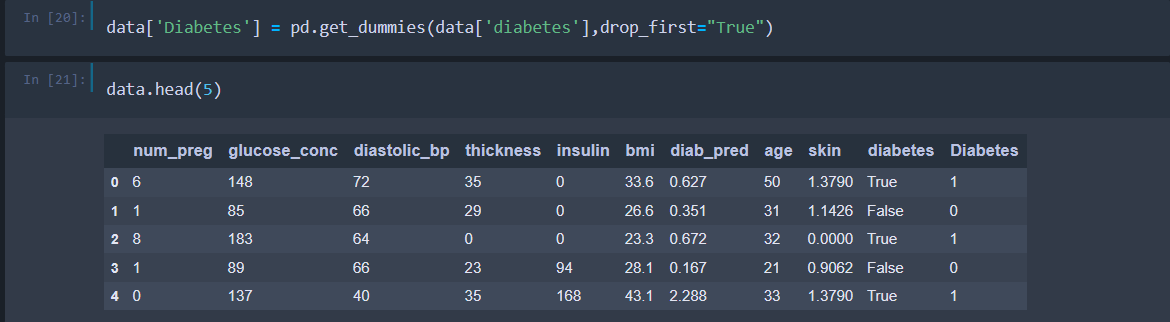
Let’s go ahead and find out the **data type** of the values present in each of the column. We use the **dtypes** property for this purpose. The last column contains Boolean values (**True, False)** and all other columns have either or **float values.**



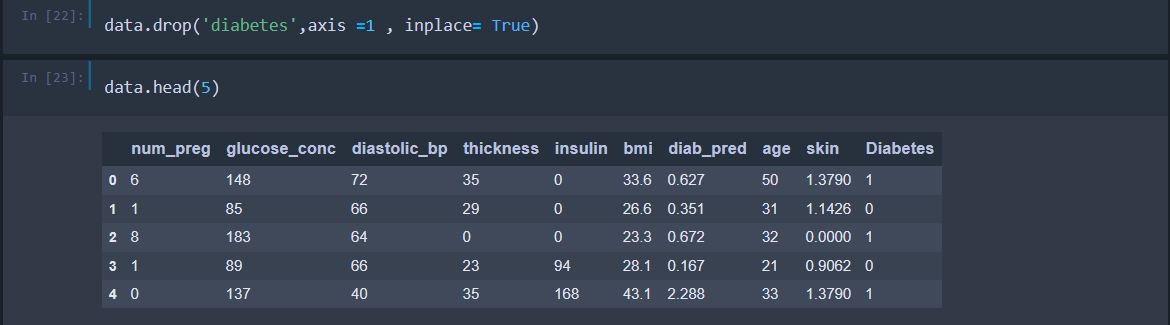
Now let’s identify the **errors** in our dataset, if any and use methods/functions to handle those errors affectively.

Now that we have handled the errors and we have an understanding about the dataset, lets move ahead and explore how the features of our dataset are related to each other. In other words, lets plot the **correlation matrix** to get the **correlation coefficients** between various features of the dataset. Similar to the earlier problem statements, we use the **Seaborn** library for this purpose. The **target variable** is found to have a decent correlation with features like **glucose concentration, bmi (**body mass index) and **age.**

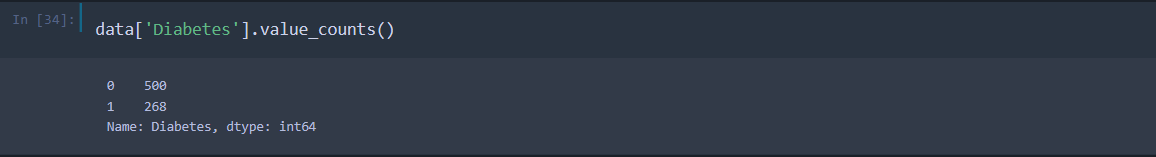
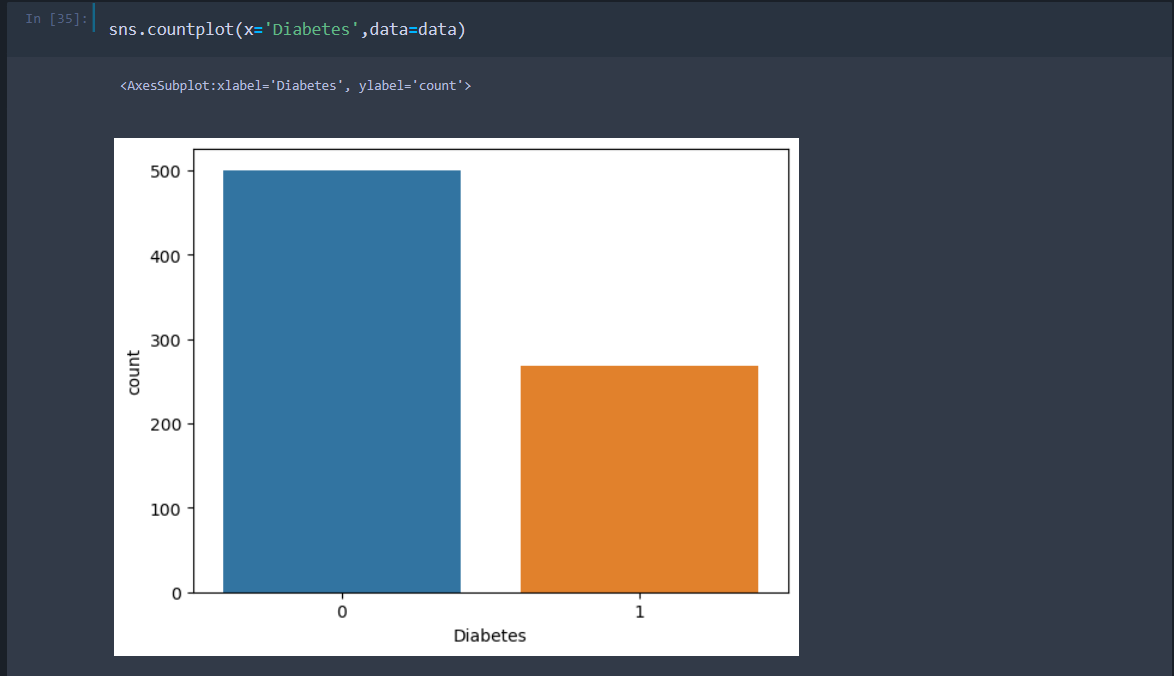
As seen earlier, the last column contains **Boolean** values and in order for us to use this column in our **Machine Learning** models, we will have to convert these **Boolean** values into **numeric values.** We can use the **get\_dummies()** method provided by pandas for this purpose.



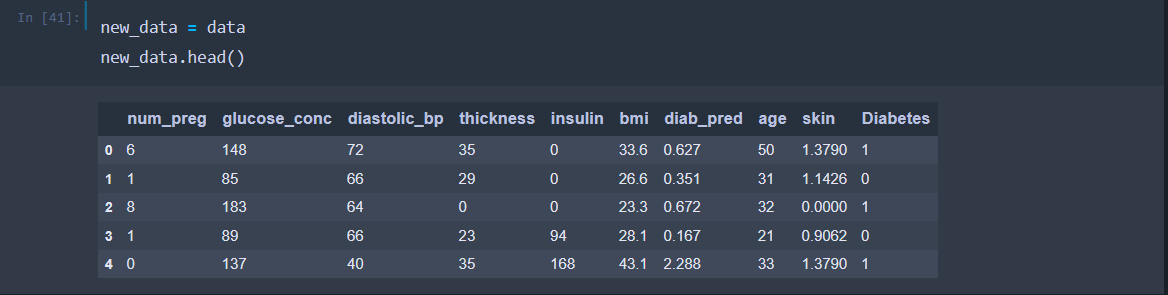
We create a new column named **Diabetes** and assign the values obtained above to that column. The dataset has 2 columns for **diabetes.** One column contains **Boolean** values and the other contains **Numeric** values. We will have to **drop** the column having **Boolean** values.



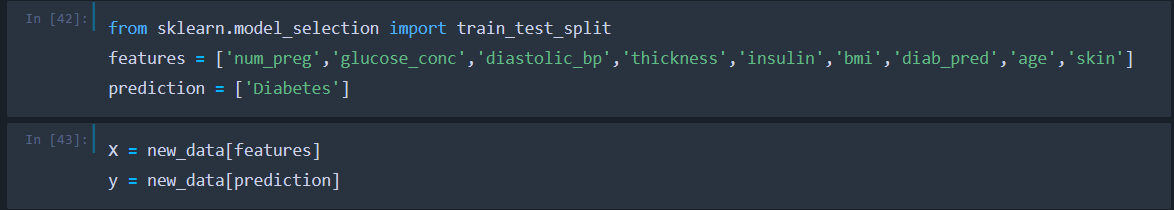
Next, let’s have a look at the ratio of **diabetic** people to **non-diabetic** people. We can use the **value\_counts()** method on the **Diabetes** column to get the **count** of unique values of that column. We can use the **countplot()** method provided by **Seaborn** library to visualize the same.



0 indicates that the person is **not diabetic** and 1 indicates that the person is **diabetic.** We create a copy of the current dataset, to train different **Machine Learning** models. This step is optional and we can even use the original dataset for this purpose.

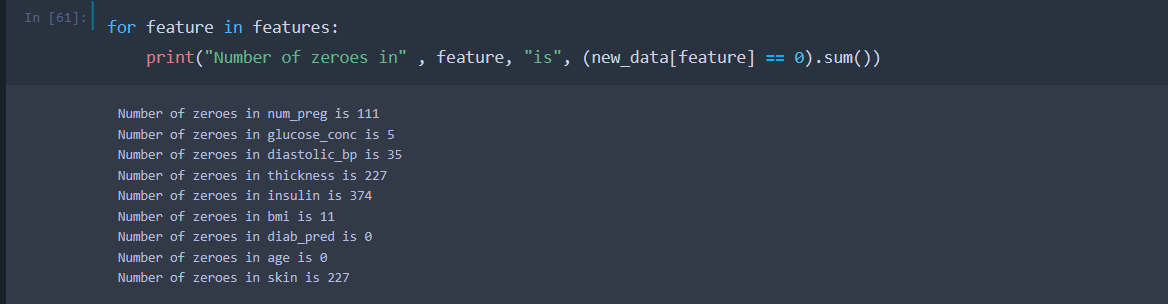


After this we will split the data into **training data** and **testing data.** The first 9 columns are stored in a list called **features** and the last column is stored in a list called **predictions.**

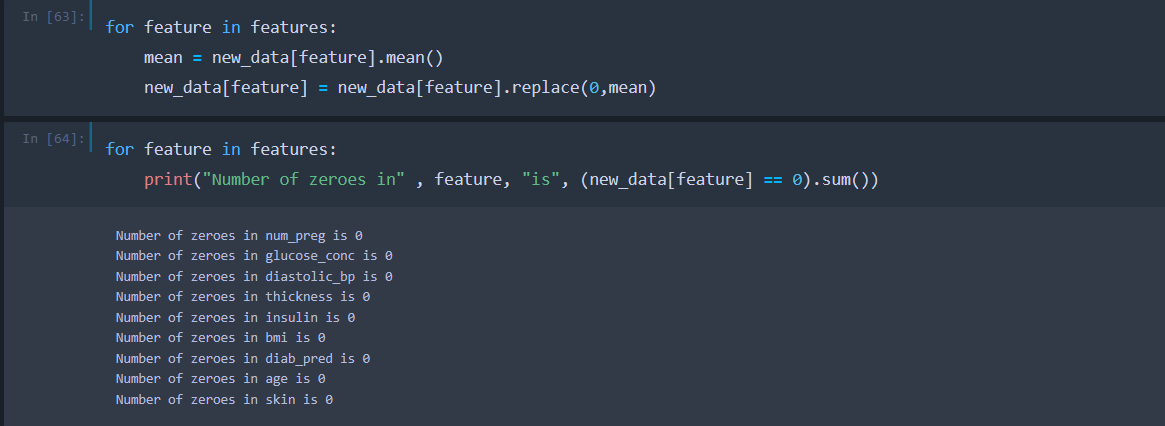
****

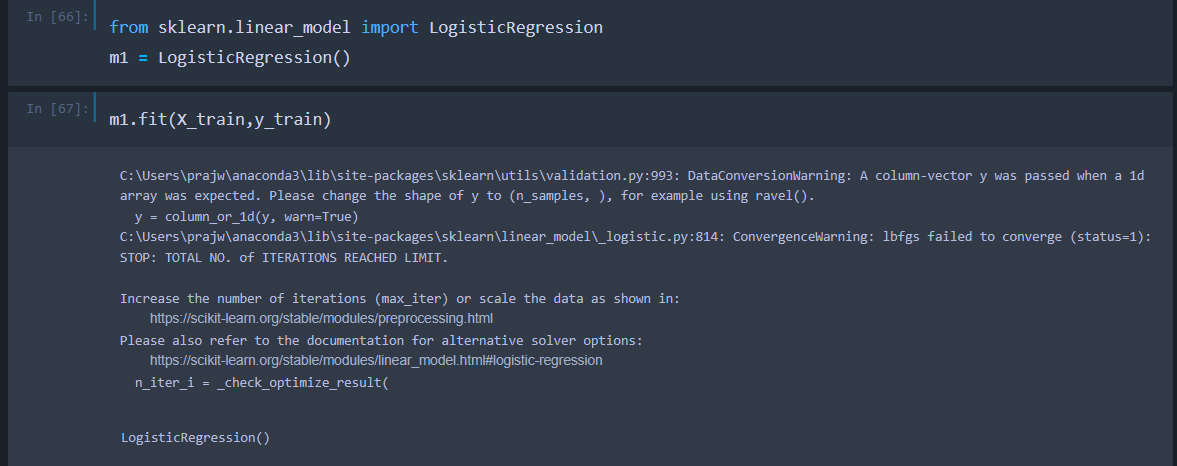


When we initially printed the dataset, it was seen that there were few zeroes in some columns. The zeroes in few columns are errors, since values like value of **insulin** cannot be 0 in a person’s body etc. We will have to develop methods to handle or get rid of these zeroes in our dataset.

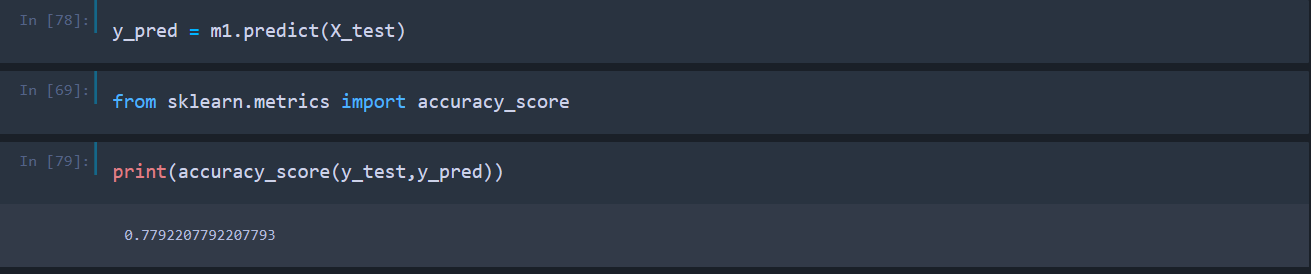


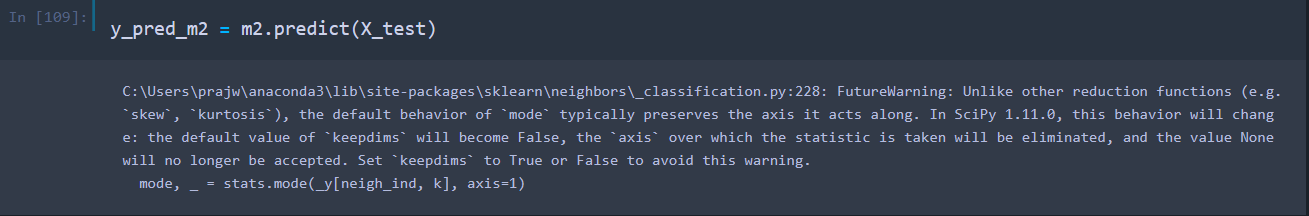
The above code gives us the **count of zeroes** in each column of the dataset. Columns like **insulin, thickness** have the highest number of zeroes and we cannot remove the rows containing zeroes, since it would lead to absurd results. The most common method to handle errors like **NaN** values, **missing** values, **zeroes** is to replace these errors with the mean value of that particular column.

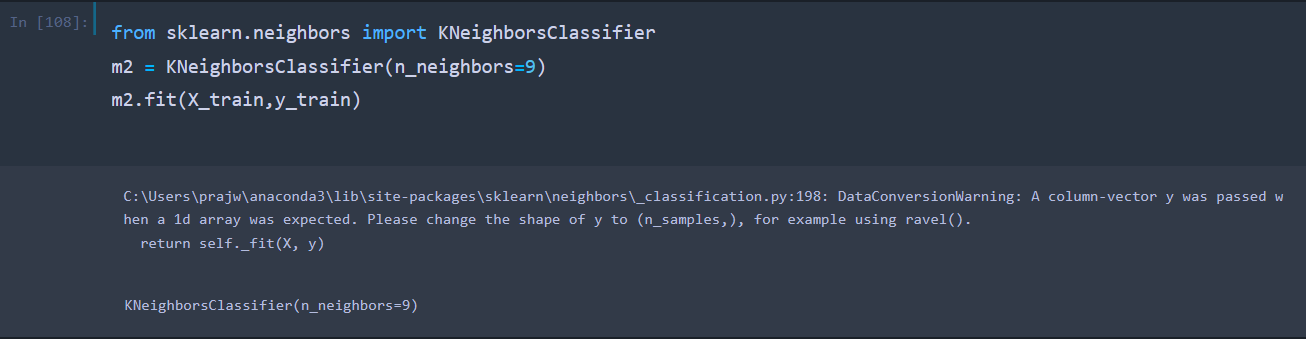
****The below code replaces the zeroes with the mean of the column to which the zero belongs to. If we look at the **number of zeroes** in each column now, it can be seen that there are no zeroes and we have handled all the zeroes.

Now we have the data in the right format to train different **Machine Learning** algorithms using our data and to get the predictions. To start off with we will use **Logistic Regression.** We import the model, create a **Logistic Regression** object and fit the **training data** into the model.

We can print the **training score** and **testing score** to get an idea about how well our model has performed when we fit the **training data** to train our model. We store the predicted values in a variable named **y\_pred.** We can get the accuracy of the model using the **accuracy\_mode()** method. It is observed that we get an **accuracy** of **78%** using the **Logistic Regression** model.



Next up, we can use the **KNN Classifier** algorithm for the same purpose. We import the **KNeighborsClassifier,** create an object of KNeighborsClassifier and use the training data to train the algorithm. Similar to the previous algorithm, we store the predictions in a variable called **y\_pred\_m2** and print the **accuracy score** to print the **accuracy.** We get an accuracy of **77.2%** in this case. The complete **Jupyter Notebook** can be obtained using this [link](https://github.com/Prajwalk09/Diabetes-Data-Analysis).

****