***Comprehensive Analysis and Visualization of the Diabetes Dataset***

* **Youssef Ahraf Abdou: 243345**
* **Abdou Osama Ahmed: 249357**

**Importing libraries and reading dataset**



This code imports 4 different libraries:

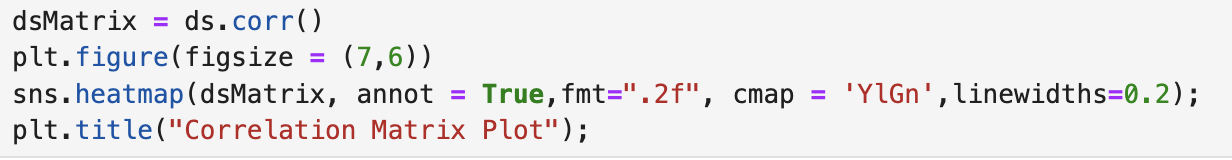
1. Numpy: which used for array and mathematics calculations and matrix manipulation.
2. Pandas: which used for reading data files such as .csv (comma separated values) and as DataFram datatype and used for some math calculations for this DataFrame such as calculation the mode and mean etc.
3. Matplotlib: which used for data analysis and visualization and it includes chats such as bie charts or histograms.
4. Seaborn: which is built on top matplotlib and used for better customization.

Every used library has an alias. For example, numpy. We use the word as and type whatever name we want to use for this library.

The second part of the codes, uses the pandas.read\_csv () method. Which read .csv files and saves it as a DataFrame. In this code, I saved in ds which stands for dataset and it going to be used throughout this project.

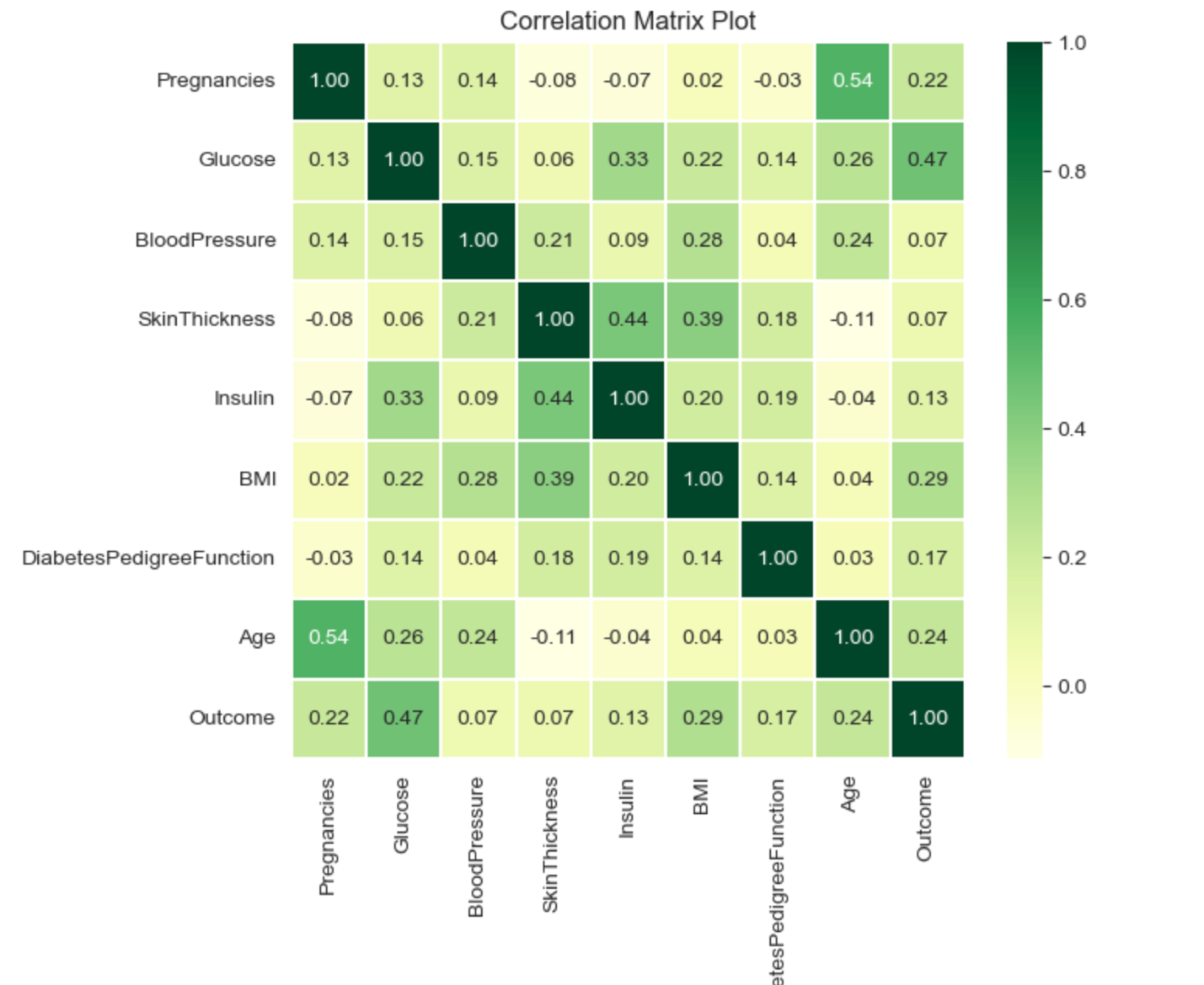
**Visualization**

**Correlation matrix**



In the first line, we used .corr() method, which constructs a correlation matrix of the DataFrame, and then saved it in a variable which dsMatrix.

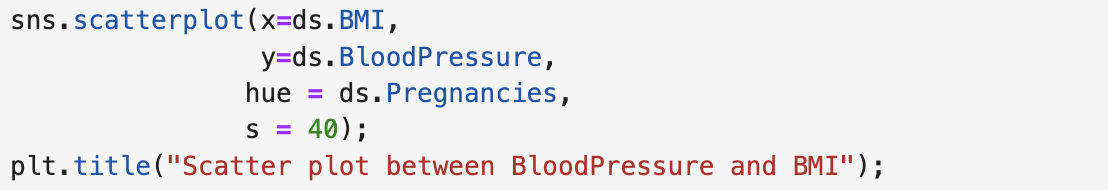
For the visualization of the correlation matrix. We used matplot. The .figure is used to edit the figure or the image that will be generated, it has an argument of figsize which is a tuple that contains both the desired width and height. Then, we used seaborn with heatmap() method. Which shows the relation between each future with itself, and other features but in terms of colors. In this case, we called the method passing 5 arguments. dsMatrix, which is the DataFrame that we want to visualize. Then, annot, which is set to true. This argument makes numbers shown instead of just colors. Fmt which is used to set the format of this number, in this case we said let the format be float, and not only this but also only show two numbers after the decimal point. Then, we have cmap, which sets the color schema for the generated visual. In this example, we used “YlGn”, which uses green and it’s shades. Note that each color map has different and unique colors with different shadings. For example, some maps uses dark colors to state a heavy relationship, while other maps uses light colors to state this relationship. In this example, YlGn uses heavy colors to states heavy correlation. the linewidth is used to leave gaps between each block in the diagram. Finally, we have the .title method, which set the title of the diagram to “Correlation Matrix Plot”.



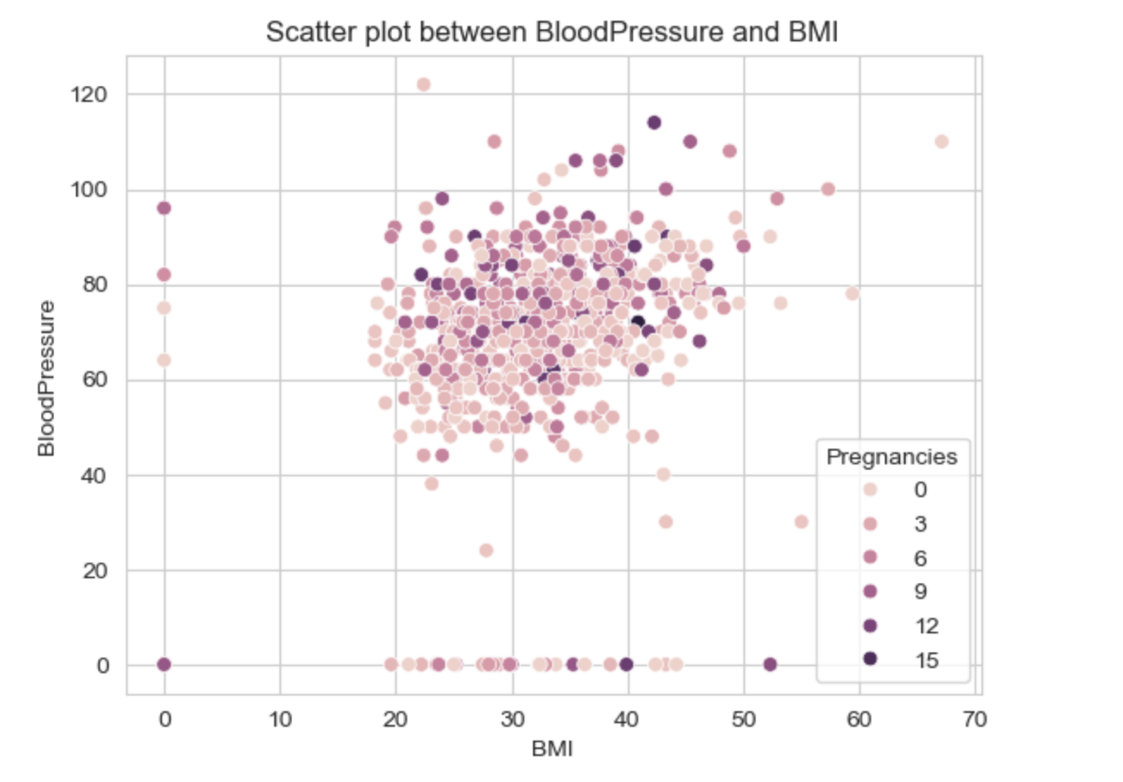
***Observations:***

This output shows the correlation between each feature, and every other feature. For example, Pregnancies have a relationship of 1 with Pregnancies. Which makes sense as the relationship with itself will be strong, meanwhile, it has a correlation of 0.13 with Glucose level. which means that Pregnancies affect Glucose levels with 13%, which makes sense why it has a lighter color than the relationship of Pregnancies with itself. And this process can be considered for each block in the diagram.

**Scatter plot between blood pressure and BMI (Body Mass Index)**

****

Firstly, we used the method scatterplot in the seaborn library, which constructs a scatter plot. The argument x is used to set the x axis, in this example we want the x axis to be the BMI, to do so we need to send ds but note that we only want the BMI not the whole other feature. Therefore, we used the **.**BMI which means that we only want the BMI feature. And the same applies for y. the hue argument is used to group each dot by a specific feature, in this example we want to group them by the number of pregnancies. Also, the s argument is a shorthand for size, which is set to 40 to make the figure larger in size. Finally, we used the title method to set the title of the plot.



***Observations:***

This scatter plot shows that the average BMI is between 20 – 50 with a blood pressure of 40 – 100. However, its noticeable that there is outliers in some areas, which can affect the average BMI and resulting in false data. Moreover, each number of pregnancies is grouped by a single color, which makes it easier to visualize data and state relation between blood pressure and BMI and also visualize which group of individuals have this BMI rate.

**Distribution plot**

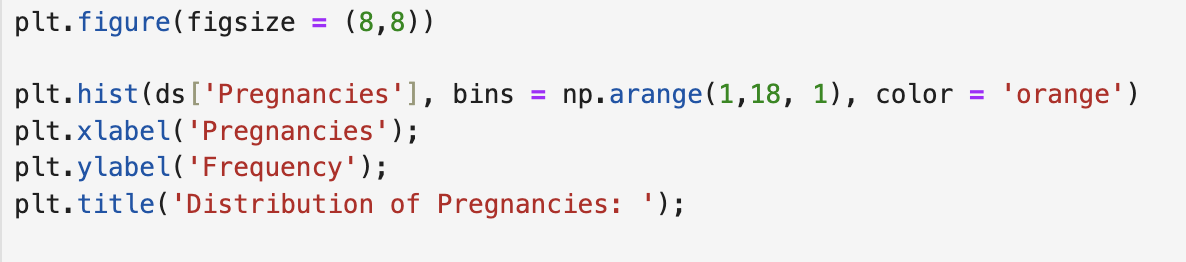
For the following:

* Pregnancies
* Skin thickness
* Age

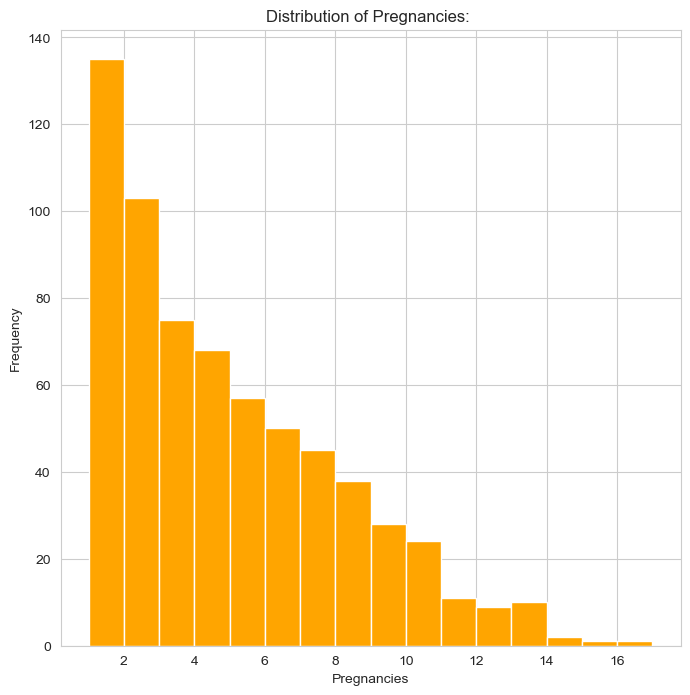


Firstly, this line uses the set\_style method to change the style of the figure. In our case, it’s used to add grid lines with white background.

***Pregnancies:***



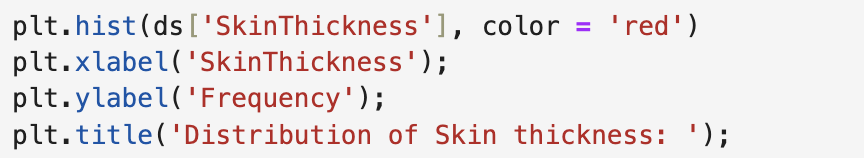
Firstly, we used the figure method to change the size of the figure with figsize argument set to (8,8) which makes the figure 8 wide and 8 long. Then we used the hist method which is a shorthand for histogram. First argument is used to send the data that we want to visualize, which in this case it’s the pregnancies. Note that this is just another way of saying that we want to use only the Pregnancies feature in ds DataFrame. Then we used the bins argument with np.arange(1, 18, 1) which generates an array of numbers starting from 1 (inclusive) to 18 (exclusive), with a step of 1. Then the xlabel method and ylabel method are used to put labels for the x and y axis. Then we used the color argument to set the color to orange. Finally, we used the title method to set the desired title.



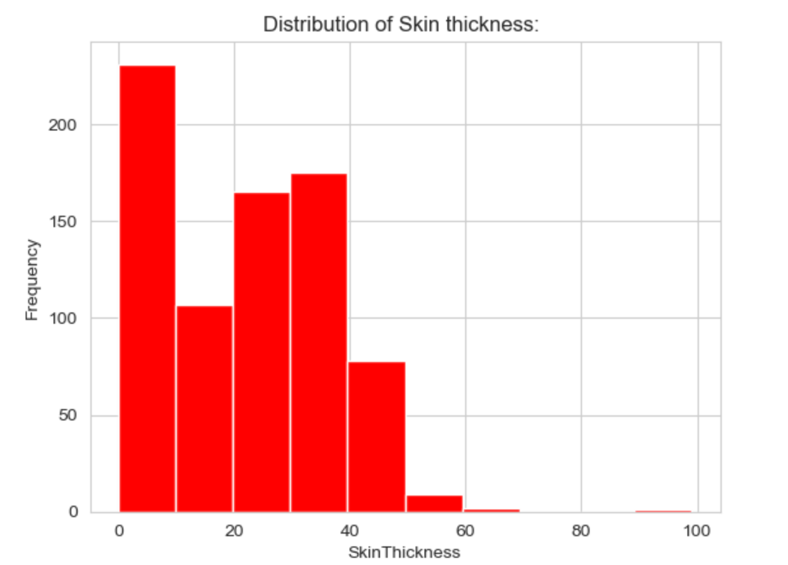
***Observations:***

The previous diagram shows that most individuals have number of two pregnancies. Moreover, the diagram can be considered right-skewed which means that most of individuals will mostly have a number of 1 to 8 pregnancies. However, it can be indicated that there are outliers laying In 14 to 16 which can influence the average number of pregnancies.

***Skin Thickness:***



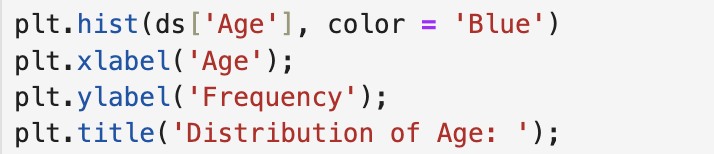
Firstly, hist method to construct a histogram. We set the first argument to ds[‘SkinThickness’] because we want to use only the SkinThickness feature in ds DataFrame. Then we used the color argument to set the color to red. Then the xlabel method and ylabel method are used to put labels for the x and y axis. Finally, we used the title method to set the desired title.



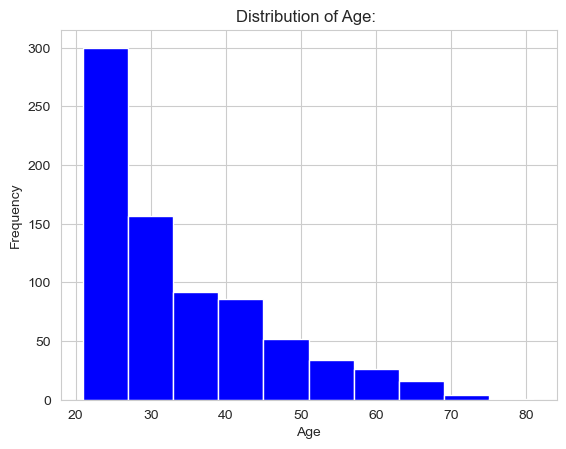
***Observations:***

The previous diagram shows that most individuals have number of 0 to 70 skin thickness. It’s also noticeable that the number of people who have from 0 to 10 skin thickness are above 200 people and with the number of skin thickness the frequency decreases suggesting lower number of people. Moreover, the diagram can be considered right-skewed. However, there are outliers laying In 60 to 100 which can influence the average number of pregnancies.

***Age:***



Firstly, hist method to construct a histogram. We set the first argument to ds[‘Age] because we want to use only the age feature in ds DataFrame. Then we used the color argument to set the color to blue. Then the xlabel method and ylabel method are used to put labels for the x and y axis. Finally, we used the title method to set the desired title.



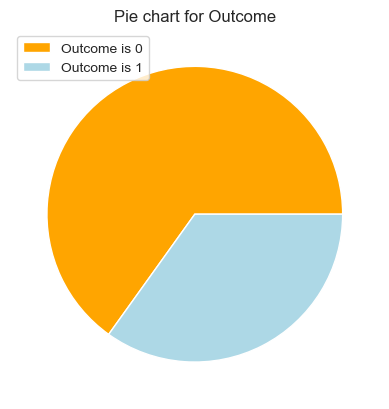
***Observations:***

The previous diagram shows that most individuals have number of 21 to 50 years old. Moreover, the figure shows that people for approximately 21 years old to approximately 27 years old has occurred 300 time in the dataset with the age increases the frequency decreases suggesting lower number of people. Moreover, the diagram can be considered right-skewed. Moreover, there are outliers laying In between 69 to 75 which can influence the average age.

**pie chart for outcome**

****

The first line is used to group people who have an outcome of 0 and an outcome of 1. It uses type casting to use the count method for each individual with 0 and 1 and then saves it inside a list indicating index 0 with the count of the 0 outcome individuals, and index 1 with the count of the 1 outcome individuals. Then we used the pie method to construct a pie chart. In the first argument, we used type casting to make the list an array with the array () method in numpy library, then we set the colors to be orange and light blue for 0 and 1 in order. Moreover, we used the legend method to put a label indicating the colors of each group of outcomes. Finally, we used the title method to put a title for the chart.

****

***Observations:***

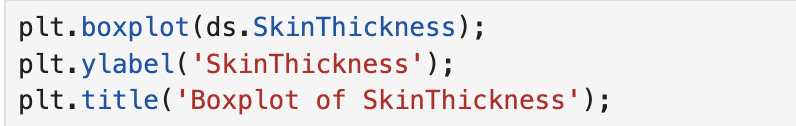
The previous diagram shows that majority of individuals have and outcome of 0, indicating false diabetes test result.

**Boxplot**

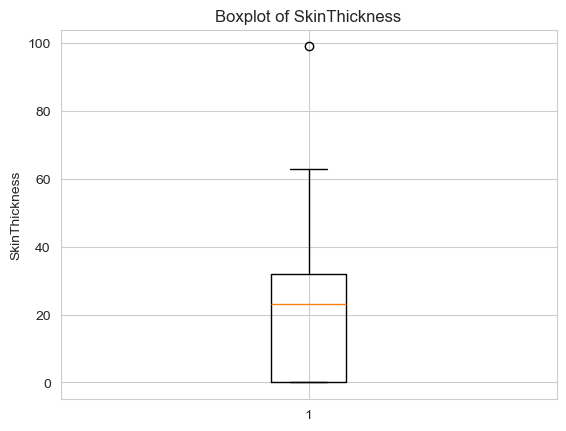
For the following:

* Skin thickness
* DiabetesPedigreeFunction
* BloodPressure

***SkinThickness:***



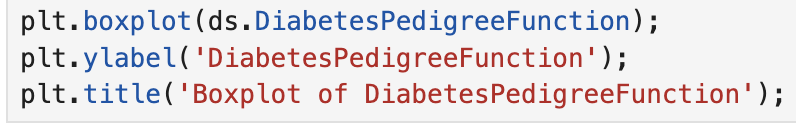
Firstly, we used the boxplot method to use a boxplot diagram, with a single argument to send only the skin thickness feature in the ds DataFrame. Then setting a ylabel and a title.



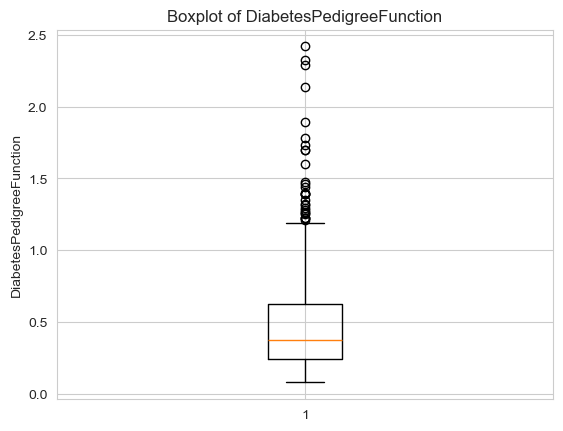
***Observations:***

It can be that there is only one outlier in the skinThickness data. Moreover, it can be indicated that the data has a right-skewedness because that 50% lays between 24 to 22 (approximately) meanwhile the other 50% lays in between 22 and 0 and this only makes sense if there are a lot of people in between 24 to 22 indicating a number of large frequencies. However, we can in fact say that this is true as we visualized the frequency and distribution of the skinThickness feature.

***DiabitesPedigreeFunction:***



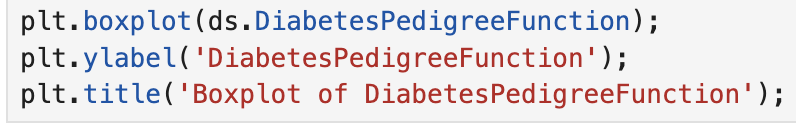
Firstly, we used the boxplot method to use a boxplot diagram, with a single argument to send only the diabetes pedigree function feature in the ds DataFrame. Then setting a ylabel and a title.



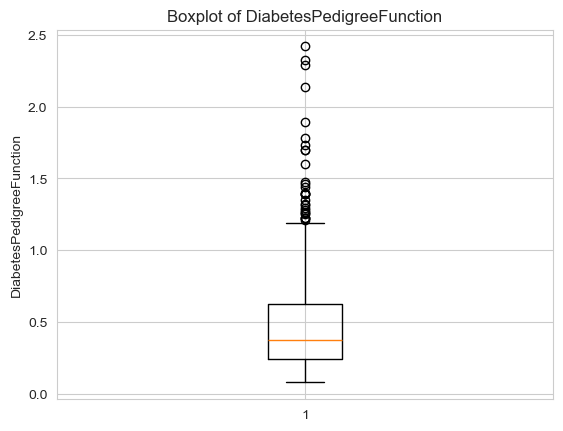
***Observations:***

It can be that there is a large group of outliers in data. Moreover, it can be indicated that the data has a left-skewedness because that 50% lays between 0.4 to 0.3 (approximately) meanwhile the other 50% lays in between 0.7 and 0.4. Moreover, we can notice that the data need to have a maximum value of 1.4 (approximately) and a minimum value of 0.1 (approximately) to not be considered an outlier.

***DiabitesPedigreeFunction:***



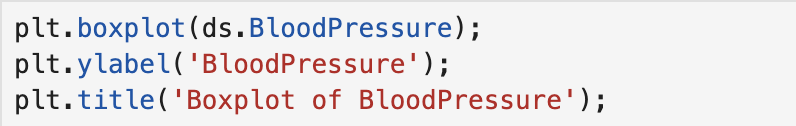
Firstly, we used the boxplot method to use a boxplot diagram, with a single argument to send only the diabetes pedigree function feature in the ds DataFrame. Then setting a ylabel and a title.



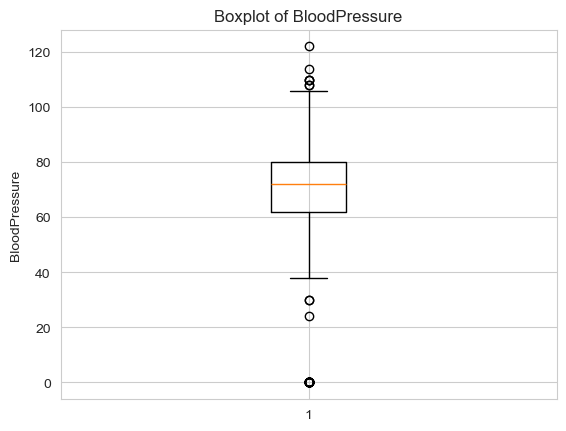
***Observations:***

It can be that there is a large group of outliers in data. Moreover, it can be indicated that the data has a left-skewedness because that 50% lays between 0.4 to 0.3 (approximately) meanwhile the other 50% lays in between 0.7 and 0.4. Moreover, we can notice that the data need to have a maximum value of 1.4 (approximately) and a minimum value of 0.1 (approximately) to not be considered an outlier.

***BloodPressure:***



Firstly, we used the boxplot method to use a boxplot diagram, with a single argument to send only the BloodPressure feature in the ds DataFrame. Then setting a ylabel and a title.



***Observations:***

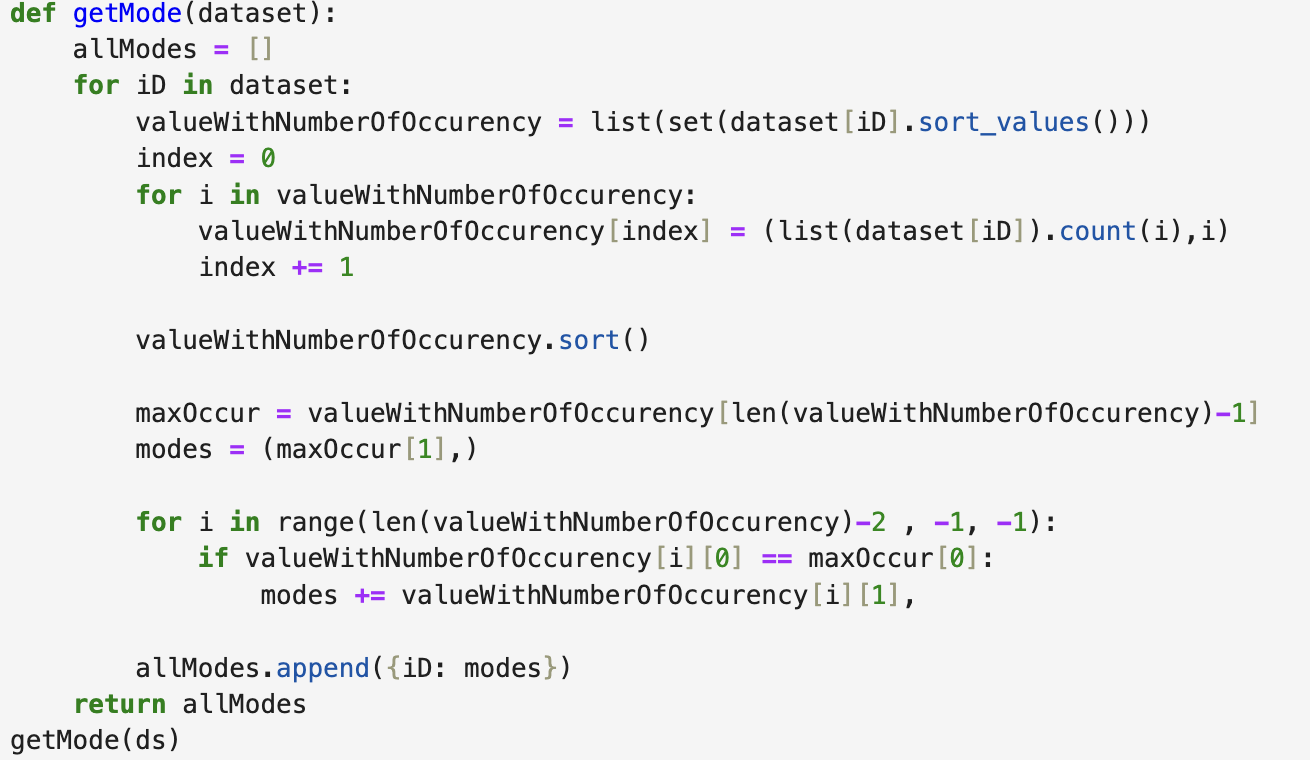
It can be that there is a large group of outliers in data both large and small. Moreover, it can be indicated that the data has a right-skewedness because that 50% lays between 80 to 74 (approximately) meanwhile the other 50% lays in between 74 and 60 Moreover, we can notice that the data need to have a maximum value of 110 (approximately) and a minimum value of 40 (approximately) to not be considered an outlier.

**Data Statistics**

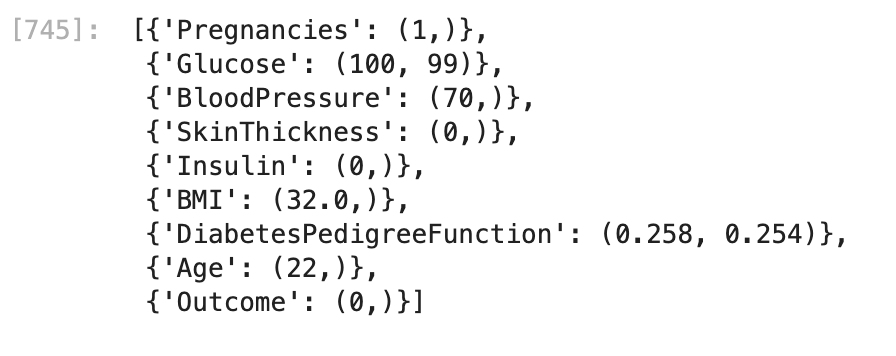
**Central tendency**

* *Mode calculation*
* *Mean calculation*
* *Median calculation*

**Mode**

****

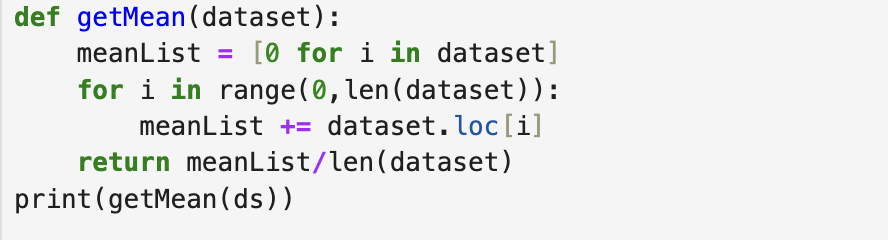
Firstly, we defined a function that receives a dataset as a parameter. Then, constructed an empty list to append the mode of each single feature in data. In next lines, we used a for loop for each key in the dataset (to calculate each single mode values), then we defined a list called valueWithNumberOfOccurency to store the values showed in this single feature in every iteration. For example, if we say ds[‘Pregnancies’] we would get all the values of the Pregnancies feature. What we want to is to get values (without duplication) and calculate how many times this value has appeared in our data. Therefore, we made a list out of the set (note that sets remove duplication which means that we would have all values appeared in this feature), and then sorting these values with the sort method. Then, we want to iterate on valueWithNumberOfOccurency to calculate every occurrence of this value in a tuple, and save it in valueWithNumberOfOccurency[index] which means that each time it will append a new tuple containing how many times this value appeared, and the value itself. Then we used the sort method to arrange this list in an ascending order, meaning that the tuple with higher number of occurrences will be in the end of the list. That’s why we saved the maximum number of occurrence in a variable called maxOccur and gave it a value of the last value in the list. Eventually, we created a tuple to make sure that we send back all modes of this single feature not only a single one. Therefore, we made a for loop to iterate from the end of the valueWithNumberOfOccurency (excluding the last element because we already have it) to the first element in the valueWithNumberOfOccurency. Then, we want to check if the element of Ith index in valueWithNumberOfOccurency the value of the 0 index in the tuple (which is the number of occurrences) has the same maximum number of occurrences, if so, we want to append this new value that has the same maximum occurrences to be a new mode in the tuple modes. Then finally, we will add the iD (key aka feature name) and include a tuple including the number of occurrences. Then returning the list of all these dictionaries.



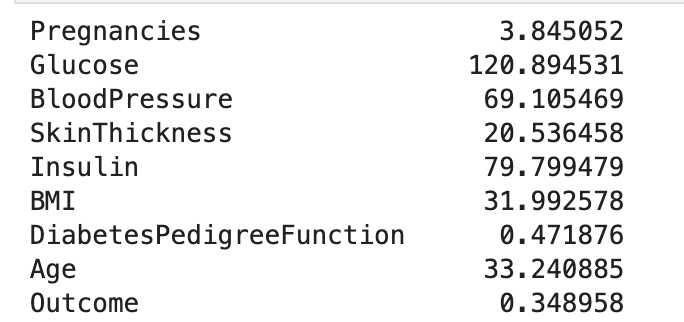
***Observations:***

1. **Pregnancies (Mode: 1)**
   * The most common value for the number of pregnancies is 1, indicating that a majority of the dataset contains individuals who have had either no pregnancies or only one.
2. **Glucose (Modes: 100, 99)**
   * There are two modes here, 100 and 99, showing that these glucose levels are the most frequently observed. These values could indicate common glucose levels in the population under study.
3. **BloodPressure (Mode: 70)**
   * A blood pressure value of 70 is the most frequent, which might represent a normal or slightly low blood pressure level in this dataset.
4. **SkinThickness (Mode: 0)**
   * The mode being 0 suggests that many individuals in the dataset have a recorded skin thickness of zero. This could indicate missing or unrecorded data.
5. **Insulin (Mode: 0)**
   * Similar to SkinThickness, an insulin value of 0 might represent missing or unrecorded data, as it's unlikely for insulin levels to be genuinely zero.
6. **BMI (Mode: 32.0)**
   * The most common BMI is 32.0, which falls into the overweight or obese range. This might suggest a prevalence of obesity in this population.
7. **DiabetesPedigreeFunction (Modes: 0.258, 0.254)**
   * The diabetes pedigree function has two close modes, 0.258 and 0.254, indicating these values are most frequent. This metric represents the likelihood of diabetes based on family history, and these values might suggest moderate risk is common.
8. **Age (Mode: 22)**
   * The most common age is 22, indicating a younger demographic within the dataset.
9. **Outcome (Mode: 0)**
   * An outcome mode of 0 implies that most individuals in the dataset do not have diabetes, which is consistent with many population datasets where diabetes is less common.

**Mean**



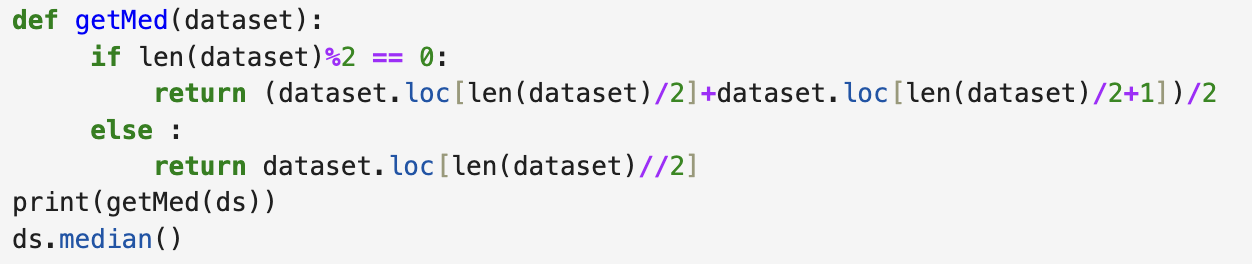
Firstly, we defined a function called getMean which receives a DateFrame called dataset as a parameter. Then, we defined a list called meanList using list comprehension to make a list that has a zero instead of containing the name. Then, we iterate over every row in the dataset will we reach the last row (len(dataset)) and we did not use len(dataset)-1 as the range method does not include the end of the range. Then we want to add this list to the meanList (addition not appending), and as we do this we add every column of the current row to every column of the meanList. Also, note that when accessing a row in dataframe we need to use the loc[i] as we can not access the row by saying dataset[i] directly. Then we return the sum of every row (meanList) but divided by the size of the dataset. Then, we called the function.



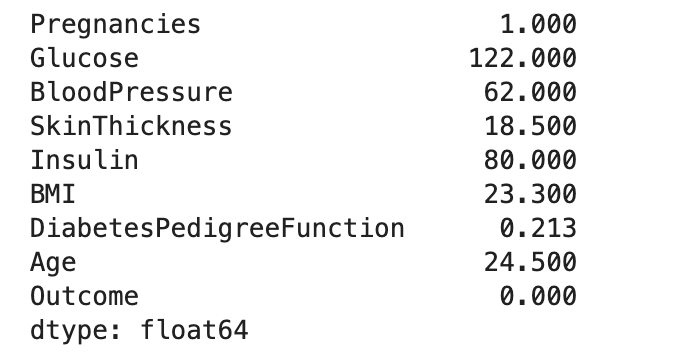
***Observations:***

1. **Pregnancies**:
   * The average number of pregnancies is **3.85**, indicating that most individuals in the dataset have had 3–4 pregnancies.
2. **Glucose**:
   * The average glucose level is **120.89**, which falls within the range that might indicate a risk of diabetes. This highlights the dataset's potential focus on diabetes-related health metrics.
3. **Blood Pressure**:
   * The average blood pressure is **69.11**, which is slightly below the normal range of 80 mmHg (diastolic pressure). This might suggest some individuals in the dataset may have low blood pressure.
4. **Skin Thickness**:
   * The mean skinfold thickness is **20.54 mm**, which can indicate the level of subcutaneous fat. This value may hint at a moderately healthy population.
5. **Insulin**:
   * The average insulin level is **79.80**, which could suggest normal fasting insulin levels in most individuals. However, values significantly above this may indicate insulin resistance.
6. **BMI (Body Mass Index)**:
   * The mean BMI is **31.99**, which falls into the **obese** category (BMI ≥ 30). This suggests a tendency towards overweight or obesity among individuals in the dataset.
7. **Diabetes Pedigree Function**:
   * The average value is **0.47**, representing a moderate genetic predisposition to diabetes. A higher value would suggest a stronger hereditary link.
8. **Age**:
   * The average age is **33.24 years**, indicating a relatively young population. This could be important in understanding the onset of diabetes or other health conditions in this age group.
9. **Outcome**:
   * The average outcome is **0.35**, which indicates that about **35%** of individuals in the dataset have a positive diabetes diagnosis (assuming a binary outcome where 1 = diabetic, 0 = non-diabetic).

**Median**



Firstly, we defined a function called getMed to calculate the median. Then if the size of the data set is even, then we want to return the two middle values (by accessing the middle row (which is the size of the dataset divided by two) using the loc method) but added and divided by two. If the size is odd, then we only want to return the exact middle value. Note that we used the // operator as the loc method only receives a +ve integer, which makes sense as this is an index. Then, we called the function.



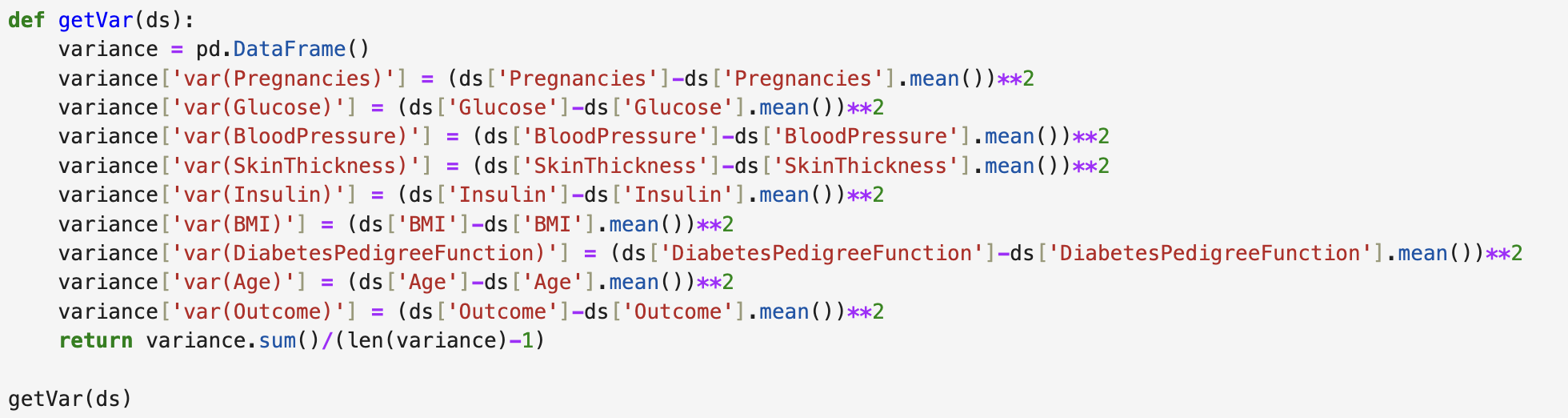
***Observations:***

1. **Pregnancies**: 1.000
   1. This suggests the individual had 1 pregnancy.
2. **Glucose**: 122.000
   1. Glucose levels are slightly above the typical fasting range, potentially indicating mild hyperglycemia.
3. **Blood Pressure**: 62.000
   1. This is relatively low and might indicate hypotension.
4. **Skin Thickness**: 18.500
   1. The subcutaneous fat measurement is on the lower side.
5. **Insulin**: 80.000
   1. Insulin levels are within a normal range.
6. **BMI**: 23.300
   1. The BMI is in the normal range, suggesting a healthy body weight.
7. **Diabetes Pedigree Function**: 0.213
   1. This indicates a low genetic predisposition to diabetes.
8. **Age**: 24.500
   1. A young individual.
9. **Outcome**: 0.000
   1. This person does not have diabetes.

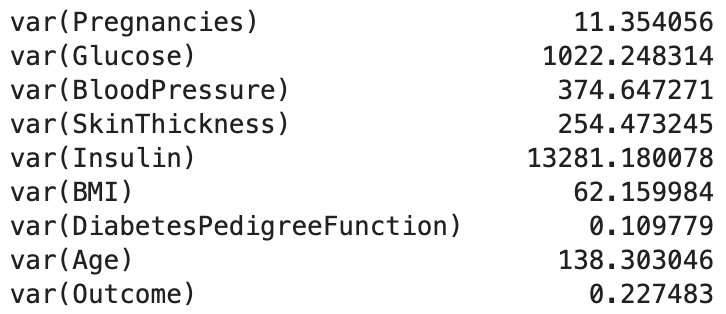
**Dispersion**

* *Variance*
* *Standard deviation*

**Variance**



Firstly, we want to define the variance, which an empty dataframe that will continue the variance values of each feature. Then for each feature, we will add the values in a column called *“var(FeatureName)”* and if does not exist, it will create a new key in the dataframe containing this name. To calculate the variance, we want to subtract all values in for this feature by the mean and then give it a power of two. Firstly, we will access the desired feature by saying ds[‘Feature’] , this will give us a sequence containing all values for this feature, then by using the mean function in pandas we will automatically subtract the mean from each row in the sequence. Eventually, to give it a power of two we will use the power operator and repeat this for every feature. Then, we called the function.



***Observations:***

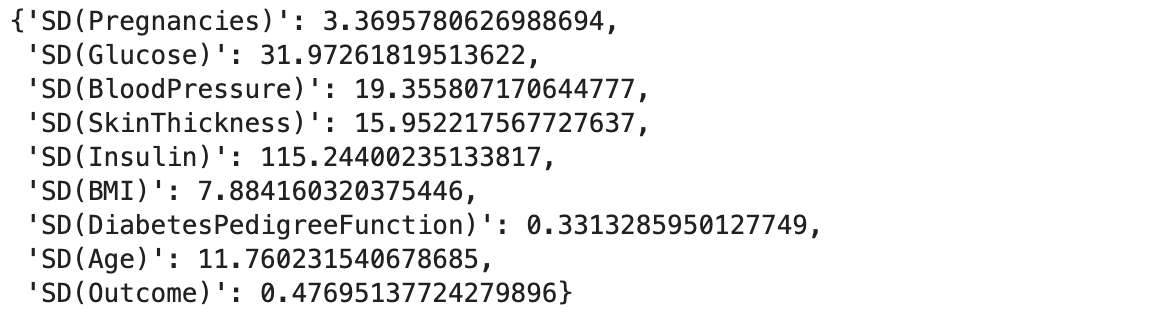
* **Glucose, Insulin, and Blood Pressure**
  + exhibit the highest variances, which is typical in medical data, as these features tend to have significant variation across individuals, reflecting different health conditions.
* **variance in BMI**
  + indicates that there is a mix of people within both normal weight and overweight categories.
* **Skin Thickness,Age**
  + also show considerable variation, which could help in identifying the diverse body compositions and age groups in the dataset.
* **Diabetes Pedigree Function**
  + has low variance, implying that the population in this dataset is relatively similar in terms of genetic risk for diabetes.
  + **Outcome variable** shows moderate variance, indicating that the dataset includes both diabetic and non-diabetic individuals, though it appears that non-diabetic individuals are more common.

**Standard deviation**

**Variance**



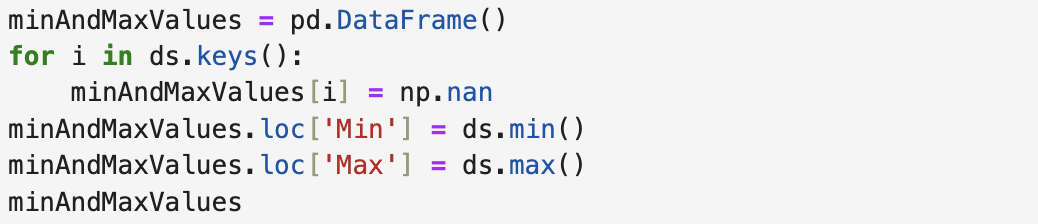
Firstly, we defined a function called getSd to calculate the standard deviation. To do so, we called the variance function we created earlier to store the variance of each feature. Then, created an empty dictionary called SD to store the variance in this variable. Then we will get the square root (using the sqrt method in numpy) and passing the desired feature by saying variance and then passing the desired feature variance name which is “var(feature)”. Then we will store this result in the *“SD(FeatureName)”key* and if does not exist, it will create a new key in the dectionary containing this name.



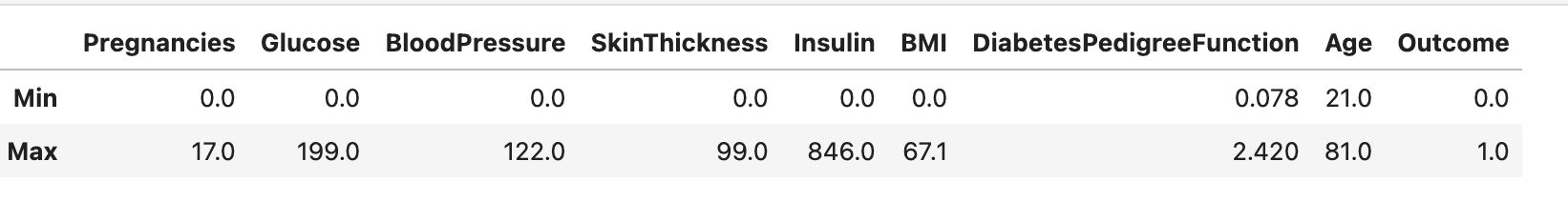
***Observations:***

* **High SD**:
  + Features like **Insulin** and **Glucose** show high variability, indicating large differences between individuals in these measurements.
* **Low SD**:
  + Features like **DiabetesPedigreeFunction** and **Outcome** show lower variability, suggesting less spread in their values.
* **Moderate SD**:
  + Features like **Pregnancies**, **BloodPressure**, and **BMI** have moderate variability, which is typical for continuous features that have a wide range but aren't as spread out as others.

**Maximum and minimum**



Firstly, we created a dataframe called minAndMaxValues, this will store the min and maximum values. Then, we will iterate over every key in the dataset (ds) and create the same key in minAndMaxValues but we will make an initial value of nan by using the numpy nan (Not A Number), then we will create a new row called “Min” that will have the minimum value, and “max” that will have the maximum value. To calculate the min and max, we will us the min() and max() method in pandas library by saying ds.min(). This will give is the min and max but as a dataframe.



***Observations:***

* **Pregnancies**:
  + **Min**: 0.0
  + **Max**: 17.0
  + The number of pregnancies in the dataset ranges from 0 to 17, showing a wide range of values. Some individuals have no pregnancies, while others have up to 17 pregnancies.
* **Glucose**:
  + **Min**: 0.0
  + **Max**: 199.0
  + The glucose levels vary from 0 to 199. This indicates a wide variation in glucose levels, with 0 potentially representing missing or abnormal data points, and 199 being a high glucose level.
* **BloodPressure**:
  + **Min**: 0.0
  + **Max**: 122.0
  + The blood pressure readings range from 0 to 122, with 0 indicating a potential data issue or missing value, while the maximum value is within a typical range for human blood pressure levels.
* **SkinThickness**:
  + **Min**: 0.0
  + **Max**: 99.0
  + Skin thickness varies from 0 to 99, suggesting some values might be unreasonably low or missing. The maximum value (99) is within a reasonable range for human skin thickness.
* **Insulin**:
  + **Min**: 0.0
  + **Max**: 846.0
  + Insulin levels vary from 0 to 846. The minimum value of 0 could suggest missing or abnormal data, while the maximum insulin level of 846 is quite high, indicating some extreme cases.
* **BMI (Body Mass Index)**:
  + **Min**: 0.0
  + **Max**: 67.1
  + The BMI values range from 0 to 67.1, with 0 likely representing missing or invalid data, and the maximum value (67.1) being well above typical BMI values, possibly indicating obesity or data errors.
* **DiabetesPedigreeFunction**:
  + **Min**: 0.078
  + **Max**: 2.420
  + The diabetes pedigree function ranges from 0.078 to 2.420, indicating variability in the genetic likelihood of diabetes. The low minimum value suggests a relatively low genetic risk in some individuals.
* **Age**:
  + **Min**: 21.0
  + **Max**: 81.0
  + The ages range from 21 to 81, indicating a wide range of ages within the dataset, but excluding very young individuals (below 21).
* **Outcome**:
  + **Min**: 0.0
  + **Max**: 1.0
  + The **Outcome** variable is binary, with values 0 and 1. The range shows that there are both negative and positive outcomes for diabetes (0 for non-diabetic, 1 for diabetic).