**Implementations**

The code base consists of 3 python files and 1 jupyter file. The main file is the jupyter file, this file connects the other python files together I.e using them as modules.

Those modules include the load\_dataset.py(this is used for the reading of the dataset and returning the required datas(we shall talk more on this), the similarity\_module.py(this is where the similarity calculations are done and values returned,.

**load\_dataset.py**

I imported the csv library to help convert the dataset file which is a csv file to dictionary type for easy manipulation.Then the file is opened using the default python open keyword.

**Why dictionary**

**Dictionary is the best data type to use for the storing the values as I took advantage of the key:value pair, this helps to assign all attributes of a music or artists to a particular key(artists or musid id)**

The load\_dataset contains a two functions I.e(Artist\_features and Music\_features)

**The Artist\_features:** a variable *artist\_features* containing a empty **dictionary data type** is being initialized, in the next 3 lines the *csv\_reader(*a variable containing our dataset in a dictionary form) is being looped through and our empty *artist\_features* is being filled with each artists as keys with their respective features unique to them in a **list datatype** as values. And we return the *artist\_features* which is now a **dictionary** holding artists with their respective features.

**The Music\_features:** a variable *music\_features* containing a empty **dictionary data type** is being initialized, in the next 3 lines the *csv\_reader(*a variable containing our dataset in a dictionary form) is being looped through and our empty *music\_features* is being filled with each music id as keys with their respective features unique to them in a **list datatype** as values. And we return the *music\_features* which is now a **dictionary** holding music ids with their respective features.

**similarity\_module.py**

The similarity\_module.py contains the *contains six functions*, which are the *similarity*, *euclidean\_similarity*, *cosine\_similarity, pearson\_similarity, jaccard\_similarity, mahattan\_similarity this all takes three parameters* ***a, id\_1, id\_2*** (where a represents the variable to hold either the *music\_features or artist\_features dictionary, depending on the similarity we interested in) and* ***id\_1 and id\_2*** are variables that hold either artist name or music id, depending on the similarity we interested in.

We have an important function which is the *similarity function*

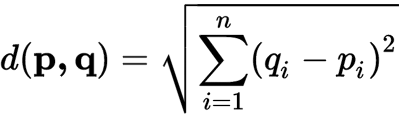
The **similiarity function** takes in 4 parameters **a** which takes the dictionary *music\_features or artist\_features depending on what we interested in finding,* ***id1*** *and* ***id2***which represents the two respective id’s, ***function*** which represents the metric to be used (either manhattan, euclidean, cosine,pearson or jaccard). In this function we check for the metric with our condtional statement and select the appropriate metric function for the function given, gets the returned score from the function and returns it to our main\_file.ipynb.

Concerning our metric functions, each of them collects the same parameters (a, id1, id2), and In all of them we retreive the features of the two id’s and we use the mathematical formula specific to those metrics to calculate the distances.

**Formulas used**

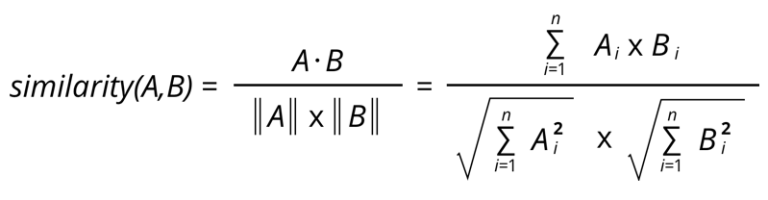
**Euclidean**

**Euclidean distance** between two points in [Euclidean space](https://en.wikipedia.org/wiki/Euclidean_space) is the length of a [line segment](https://en.wikipedia.org/wiki/Line_segment) between the two points. It can be calculated from the [Cartesian coordinates](https://en.wikipedia.org/wiki/Cartesian_coordinate) of the points using the [Pythagorean theorem](https://en.wikipedia.org/wiki/Pythagorean_theorem), therefore occasionally being called the

**Pythagorean distance**

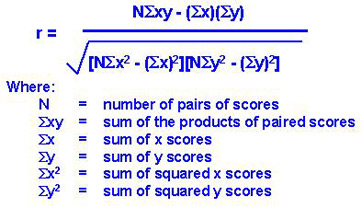
**Cosine**

Cosine similarity metric finds the normalized dot product of the two attributes. By determining the cosine similarity

****

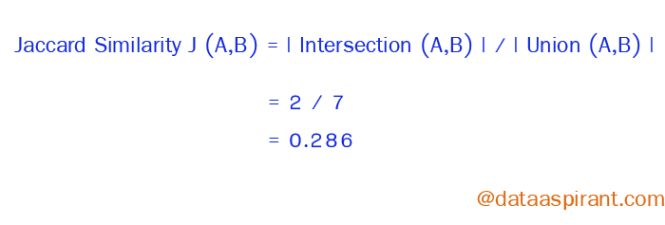
**Pearson**

Pearson correlation coefficient or Pearson’s correlation coefficient or Pearson’s r is defined in statistics as the measurement of the strength of the relationship between two variables and their association with each other.



**Jaccard**

The Jaccard similarity measures similarity between finite sample sets, and is defined as the cardinality of the intersection of sets divided by the cardinality of the union of the sample sets

****

**Manhattan**

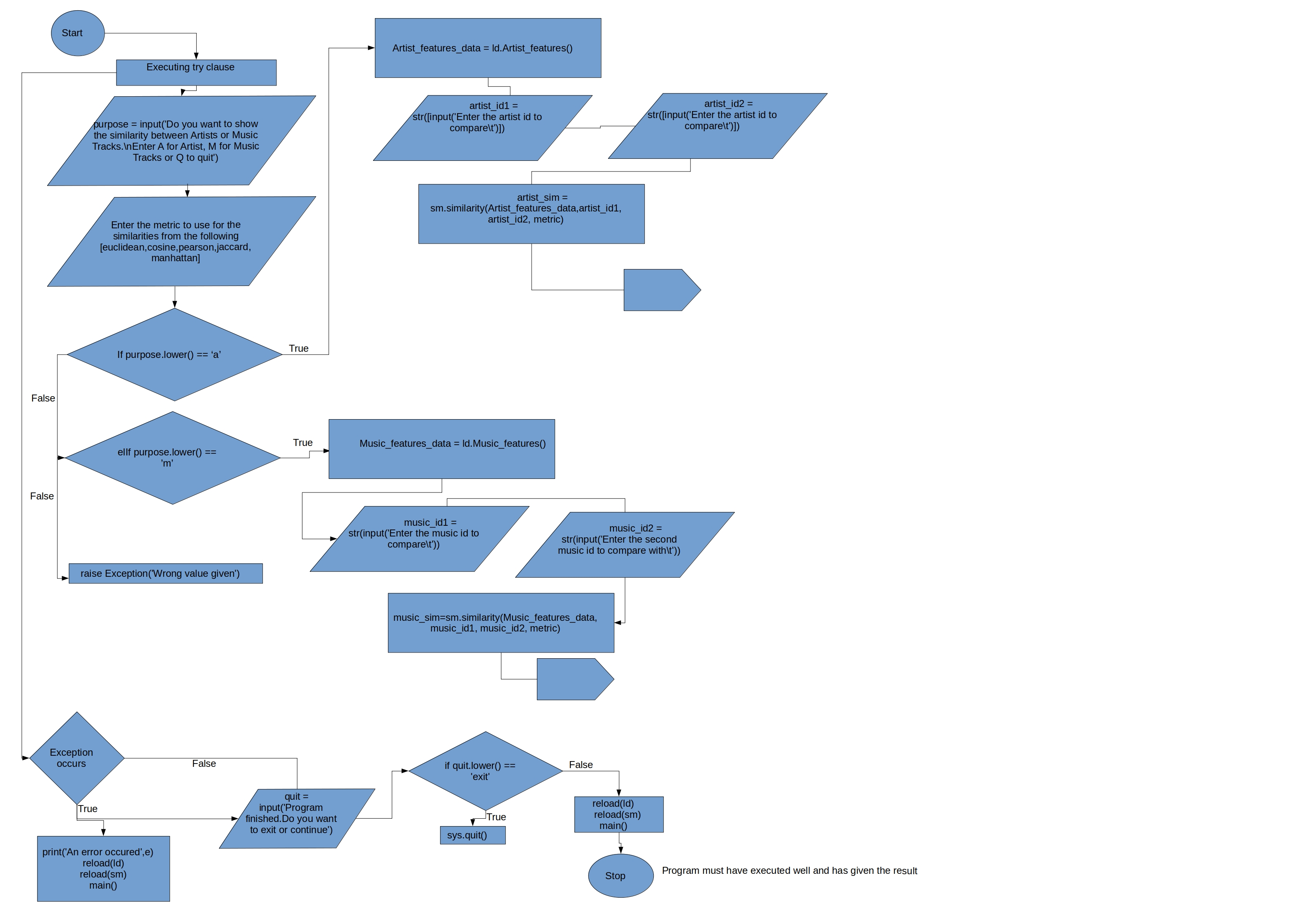
Manhattan distance is a metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates

**Manhattan distance = |x1 – x2| + |y1 – y2|**

**main\_file.py**

The mother ship that brings all the pieces together, first by importing all the modules A flowchart diagram shall be used to explain this file and the whole workflow the code base.

**Flow Chart Diagram**



**Problem Encountered**

1. I realized that once the program is been executed once, and the user is trying to execute it again, it throws error on second execution. So I found a way around this by reloading the module using the python reload library to reload the imported modules, this was the way round that bug.

**References**

1. **https://www.dabblingbadger.com/blog/2020/2/27/implementing-euclidean-distance-matrix-calculations-from-scratch-in-python**
2. **https://clay-atlas.com/us/blog/2020/03/27/cosine-similarity-text-calculate-python/**
3. **https://bigdata-madesimple.com/implementing-the-five-most-popular-similarity-measures-in-python/**