Harsha Joshi
45561623

Practical 3 Report

It is important to understand that data from the restaurant.csv and restaurant_pair.csv that was provided, can be loaded into an Integrated Development Environment (IDE) to perform operations. Standard Python language operations, such as if-else statements, loops, etc. directly correlate to SQL or traditional DBMS methods of extracting, manipulating, or analysing data. For the following tasks the Python IDE – Visual Studio Code (VSCode) will be utilised below, with relevant screenshots of the terminal presented as the output. It is also to be noted that python commands will also be utilised in the explanations to explain the code.

Task 1

Before counting the number of records and the respective distinct values, it is important to understand the set up of the file nested_loop_by_name.py, as stated on the Task Sheet. Briefly, this file utilises the function csv_loader() from the csv_loader.py to read any .csv file. It is important that the appropriate file, with its relevant file path is specified while initialising:

```
with open(r'C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 
3\DataLinkage\DataLinkage_py\data\restaurant.csv', 'r') as f:
            lines = f.read().splitlines()
```

The fact the file path is highlighted in red demonstrates that the file path is being successfully read. csv_loader.py, also utilises a for loop such to appropriately assign values to the relevant setter functions in restaurant.py. Setter functions are flexible functions which "set or mutate the value of an attribute (such as id, name, address and city regarding the restaurant dataset) in a class (which is the restaurant table itself)" (Ramos, L.P., 2022).

Going back to nested_loop_by_name.py, a for-range loop and relevant indexing is utilised to iterate over only the specified row and column lengths' to add (.append) these values in string format into an empty result list. It is important that this will only occur on the condition that restaurant name's are matching. Finally, the time, – merely calculating the time taken – benchmark and results are calculated via the functions' specified in measurement.py. The specificities of measurement.py will be further explained in upcoming tasks. It is important to note that benchmark and result value can be flexible and easily changeable based on what file is loaded into Python via the csv_loader() function. Hence, at times, if not changed in the appropriate location the terminal may output older iterations of the data.

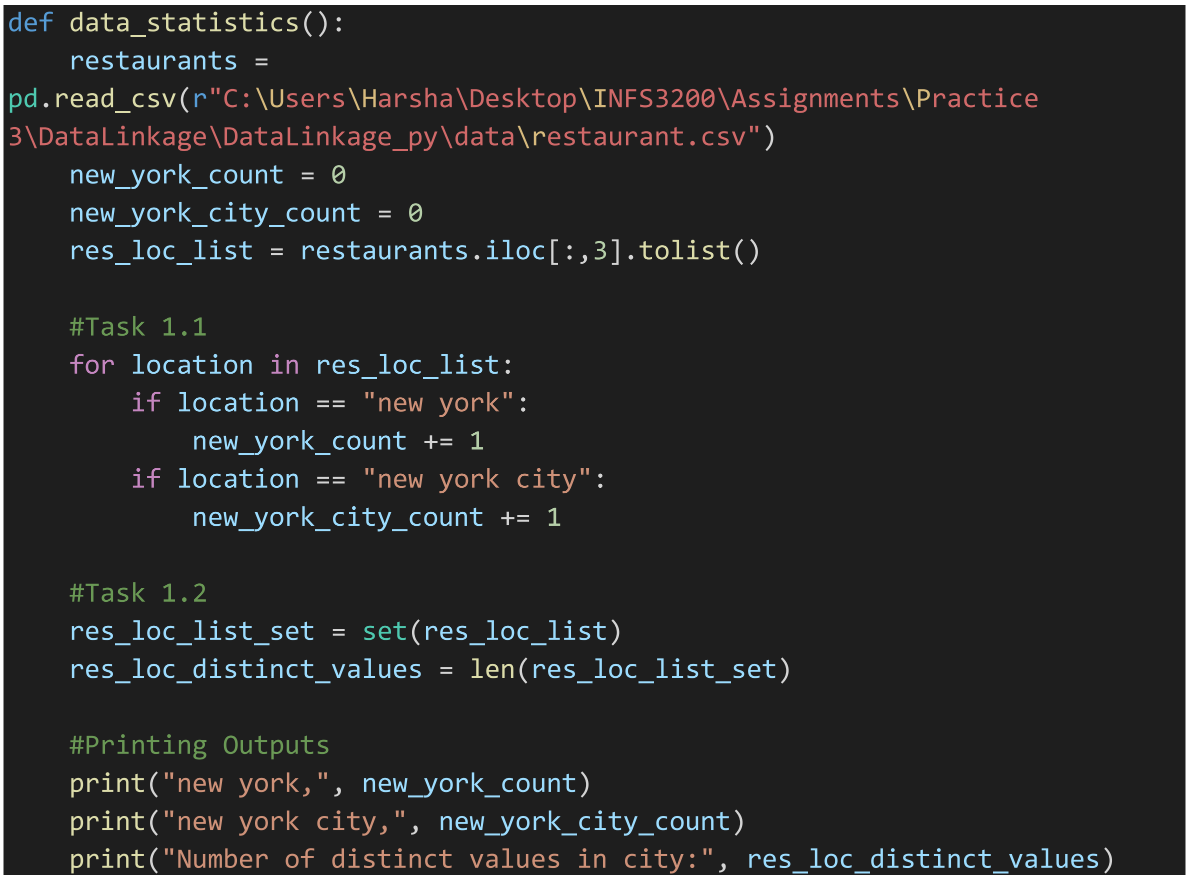
Task 1.1 & Task 1.2

For this task, Python's pandas package was imported (as pd) to simplify the implementation and file loading operations for this task, which the csv_loader.py/nested_loop_by_name.py does at length. Essentially, pandas is an open-source python package catered towards data structures and analysis, hence can be implemented here to read and extract information from restaurant.csv via the pd.read_csv() command.

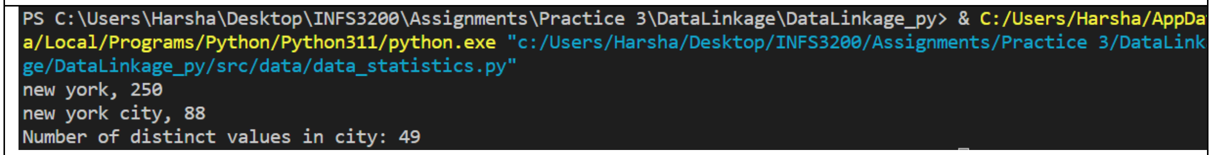
Following this, for Task 1.1, a for loop was used to iterate over the restaurant dataframe (pandas treats csv files as dataframes). However, to iterate successfully, the dataframe must be converted into a list such that it becomes a list of strings. This is done so that each string element (which is the variable location) of the newly formed list of strings (res_loc_list) can be checked against the string inputs "new york" and "new york city" respectively. It is to be noted that the pandas command .iloc[:, 3] was utilised in order to only convert the fourth row of the restaurant.csv file via indexing and further converted into a list via the inbuilt .tolist() python command. Similarly, two different counters (new_york_count and new_york_city_count) were set up to find the number of restaurant location list elements matching with provided string values of “new york” and “new york city” respectively. The output is as below and counts conditionally given that the if statement is true.

For Task 1.2, the number of distinct values was found by simply removing duplicate values in the list of strings directly developed from the restaurant dataframe (res_loc_list). This was done via the set command which essentially orders the list and removing any duplicates in the list and returns a new list variable (res_loc_list_set). Finding the length of this new list will give the number of distinct elements (res_loc_distinct_values), because it has already been filtered to remove duplicates above via the set command.

Finally, all relevant outputs are printed as in the screenshot below.



```
def data_statistics():
    restaurants = 
pd.read_csv(r"C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 
3\DataLinkage\DataLinkage_py\data\restaurant.csv")
    new_york_count = 0
    new_york_city_count = 0
    res_loc_list = restaurants.iloc[:,3].tolist()

    #Task 1.1
    for location in res_loc_list:
        if location == "new york":
            new_york_count += 1
        if location == "new york city":
            new_york_city_count += 1

    #Task 1.2
    res_loc_list_set = set(res_loc_list)
    res_loc_distinct_values = len(res_loc_list_set)

    #Printing Outputs
    print("new york,", new_york_count)
    print("new york city,", new_york_city_count)
    print("Number of distinct values in city:", res_loc_distinct_values)
```



```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDa
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLink
ge/DataLinkage_py/src/data/data_statistics.py"
new york, 250
new york city, 88
Number of distinct values in city: 49
```

Task 2

The task sheet provides two extremely important equations for precision and recall, which can be utilised to examine and explain the meanings of true positive, false positive, false negative and true negative. The given file measurement.py is the mathematical backbone of nested_loop_by_name_jaccard.py. Firstly, it loads the given file restaurant_pair.csv, in the same way as the csv_loader.py mentioned in Task 1. This is defined under the load_benchmark() function and returns the benchmark itself and is the standard by which results must adhere to. It is to be noted that the restaurant_pair.csv file which mimics the data linkage pair mentioned in the description of Task 2 – i.e. id1_id2 = 1_2, which are ID's of the records themselves. Hence, in this example, 1 and 2 refer to the very first and second restaurant listed in the restaurant.csv file. This is further applicable to the rest of the restaurant_pair.csv file, with this file only containing one column of specified and data linked restaurant pairs.

Further, the calc_measure() function takes one argument called results. It is to be noted that this variable is fairly flexible when implemented (and is typically reassigned according to the different) and appears in multiple different areas in the given code. In this case, it refers to the data linked ID's specified in restaurant_pair.csv file. In the below tables, the derivation of true positive, false positive, false negative, true negative, precision, recall and f-measure are specified.

Task 2.1

```python
def calc_measure(results):

    benchmark = load_benchmark()
    if len(results) == 0:
        print('Precision = 0, Recall = 0, Fmeasure = 0')
        return
    count = 0

    for pair in results:
        if pair in benchmark:
            count = count + 1
    if count == 0:
        print('Precision=0, Recall=0, Fmeasure=0')
        return
    precision = count / len(results)
    recall = count / len(benchmark)
    f_measure = 2 * precision * recall / (precision + recall)

    print("Precision=", precision, ", Recall=", recall, ", Fmeasure=",
f_measure)
    return
```

According to the task sheet the values in the given equations are as follows, where:

- tp = true positive
- fp = false positive
- fn = false negative

$$Precision = \frac{tp}{tp + fp} \qquad Recall = \frac{tp}{tp + fn}$$

In referring to the code above it is evident that:

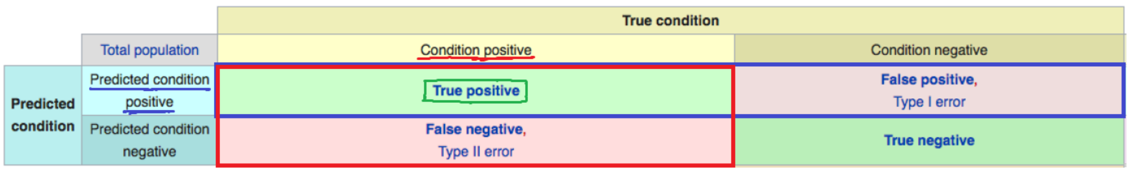
- `precision = count / len(results)`
- `recall = count / len(benchmark)`

To continue deriving, count, len(results) and len(benchmark) values must be known. Looking at the given code above and the explanation provided at the start of Task 2, it is evident that the

benchmark is the given and original restaurant_pair.csv file. Because this is the standard file of comparison, it could potentially directly link to the f-measure specified in Task 2.2. Further, results refers to the filtered results of the restaurant_pair.csv from the benchmark, given certain parameters – these will be explored more in Task 3. `len` is simply a python command which takes length of a list to return an integer value. This is command useful in finding out how many outcomes the filtered results string list has as compared to the outcomes the standard unfiltered benchmark string list has.	
$Precision = \frac{\text{count}}{\text{len(results)}}$	$Precision = \frac{\text{count}}{\text{len(benchmark)}}$
$\frac{tp}{tp + fp} = \frac{\text{count}}{\text{len(results)}}$	$\frac{tp}{tp + fn} = \frac{\text{count}}{\text{len(benchmark)}}$
The numerator and denominator of the two equations above are split up and calculated as follows. Also, it is to be noted that the true positive = count was substituted in when relevant to simplify the final equations.	
$tp = \text{count}$	$tp = \text{count}$
$tp + fp = \text{len(results)}$ $fp = \text{len(results)} - tp$ $fp = \text{len(results)} - \text{count}$	$tp + fn = \text{len(benchmark)}$ $fn = \text{len(benchmark)} - tp$ $fn = \text{len(benchmark)} - \text{count}$
Hence, • $\boldsymbol{tp = \text{count}}$ • $\boldsymbol{fp = \text{len(results)} - \text{count}}$ • $\boldsymbol{fn = \text{len(benchmark)} - \text{count}}$	
It is to be noted that true negative (tn) is not included in these equations because if the linked pair is a true negative – "an outcome where the model correctly predicts the negative class" (GoogleforDevelopers, 2023) – it would not appear in further filtered down results, though the benchmark (which the unfiltered standard dataset) would ideally contain all possible true and false value. A true negative is a correctly identified negative outcome and it would not appear in the filtered down version (results) of a dataset due to not being relevant to the conditions specified.	

Task 2.2

To continue to find the precision and recall, it is important to continue expanding on the idea of the `benchmark` and `results` variables that were explained above. It is important to specify the fact that f-measure specifically refers to benchmark, which are the gold-standard linking results, as specified by the original version of the restaurant_pair.csv file – this is stated in the task sheet. Similarly, the results significantly impact the precision or recall values as shown via the annotated table below.



		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	True positive	False positive, Type I error
	Predicted condition negative	False negative, Type II error	True negative

The annotated table above, includes two colours – red and blue directly highlight the values used in the recall and precision equations respectively. It is evident that there is a fairly close relationship between recall and precision, the former being the 'condition positive' while the latter is 'predicted condition positive'. Condition positive refers to the sensitivity and is the "probability of a positive test result, conditioned on the individual truly being positive" (Wikipedia, 2023), while predicted condition positive refers to the positive predictive value and is the "true positive values to all positive test results (including false positives)" (Safari et al., 2015). It is evident that these two values have highly significant meanings – as they refer to the how many positive results are viable based on truly positive outcomes and truly positive results based on all positive results.

We can substitute the known values of true positive, false positive and false negative from Task 2.1 to further derive and simplify the equations precision and recall equations as below.

Precision = Predicted Condition Positive (Positive Predictive Value)	**Recall = Condition Positive (Sensitivity)**
$\frac{tp}{tp+fp} = \frac{\text{count}}{\text{count} + (\text{len(results)} - \text{count})}$ $= \frac{\text{count}}{\text{count} + \text{len(results)} - \text{count}}$ $= \frac{\text{count}}{\text{count} - \text{count} + \text{len(results)}}$ $= \frac{\text{count}}{0 + \text{len(results)}} = \frac{\text{count}}{\text{len(results)}}$	$\frac{tp}{tp+fn} = \frac{\text{count}}{\text{count} + (\text{len(benchmark)} - \text{count})}$ $= \frac{\text{count}}{\text{count} + \text{len(benchmark)} - \text{count}}$ $= \frac{\text{count}}{\text{count} - \text{count} + \text{len(benchmark)}}$ $= \frac{\text{count}}{0 + \text{len(benchmark)}} = \frac{\text{count}}{\text{len(benchmark)}}$
Hence, • $\boldsymbol{Precision} = \frac{\textbf{count}}{\textbf{len(results)}}$ • $\boldsymbol{Recall} = \frac{\textbf{count}}{\textbf{len(benchmark)}}$	

Task 3

To effectively demonstrate this task, it is best to first define what q and threshold value mean and change their values to see the effect on precision, recall and f-measure. In accordance with the task sheet, it is evident that q or q-gram more specifically, is a continuous sequence of q items from a given string, whereby the items can be as written in the original string (inclusive of special characters, characters, punctuation, etc.). Hence, it is evident that q can hypothetically be any positive integer value to effectively create a q-gram. An example of this is as follows from the task sheet, where a q-gram is each segment of three characters.

- "**University_of_Queensland**": {"##U", "#Un", Uni", "niv", "ive", "ver", "ers", "rsi", "sit", "ity", "ty_", "y_o", "_of", "of_", "f_Q", "_Qu", "Que", "uee", "een", "ens", "nsl", "sla", "lan", "and", "nd#", "d##"}, where using character "#" is optional to pad the beginning and ending of the string for the completeness of 3-gram.

In accordance with the default code given in nested_loop_by_jaccard.py and the example listed, q = 3 will be used to test the effect the threshold has on precision, recall and f-measure.

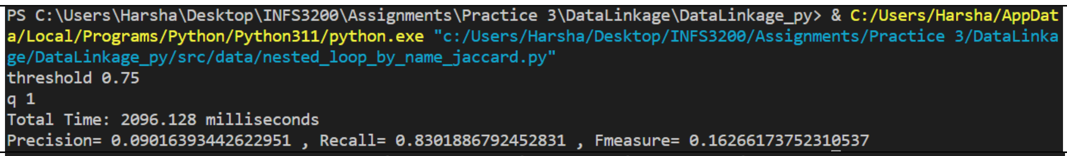
On the other hand, threshold is limited to between 0-1 (0%-100%). This specifically provides a similarity value to aspire to – a benchmark of sorts – for the Jaccard coefficient. Directly from the task sheet, the Jaccard coefficient compares members for two sets to see which members are shared and which are distinct. It measures the similarity between the two sets. Therefore, it is likely that the threshold changing will more likely demonstrate more extreme changes on precision, recall and f-measure in comparison with the q value changing. This is because the q-gram merely refers to the split of the relevant input strings, while the threshold is more closely related with the Jaccard coefficient.

As mathematically demonstrated in Task 2, precision and recall are the predicted positive value and sensitivity of the dataset respectively. Thereby, this mirrors the Jaccard coefficient which specifically mentions the shared member to distinct member ratio – very like the fact that precision is a function of true positive values to all recorded positive values and recall is a function of recorded positive values to true positive values.

To note the changes to precision, recall and f-measure, the q and threshold values are changed as specified below.

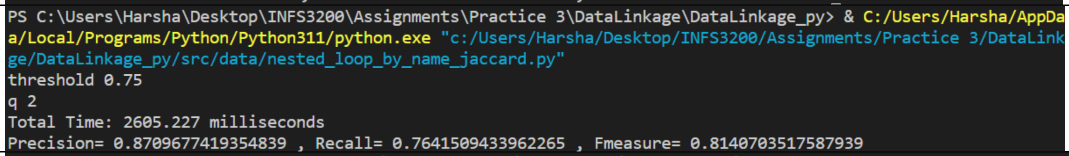
Precision, Recall, F-Measure: Q-values Changed & Threshold = 0.75

1)



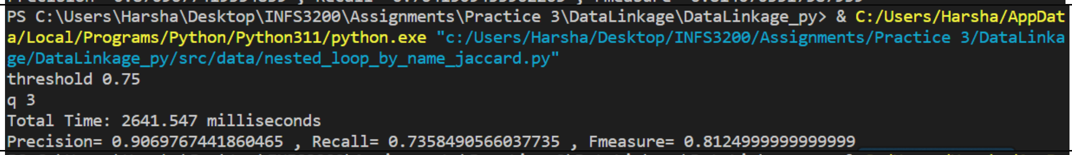
```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
threshold 0.75
q 1
Total Time: 2096.128 milliseconds
Precision= 0.09016393442622951 , Recall= 0.8301886792452831 , Fmeasure= 0.16266173752310537
```

2)



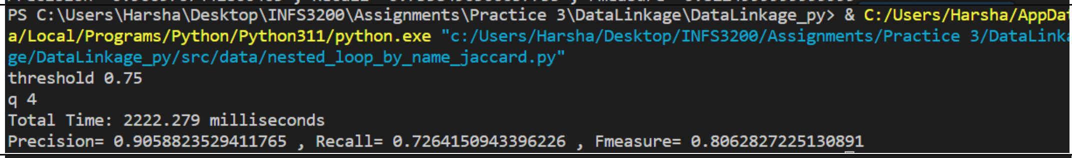
```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDa
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLink
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
threshold 0.75
q 2
Total Time: 2605.227 milliseconds
Precision= 0.8709677419354839 , Recall= 0.7641509433962265 , Fmeasure= 0.8140703517587939
```

3)



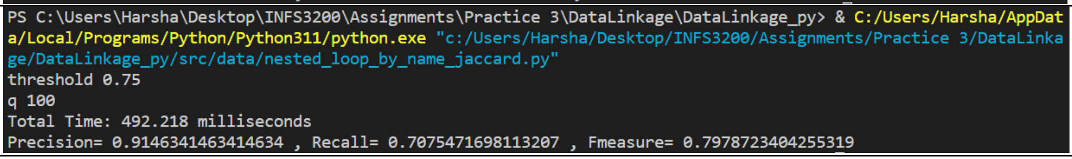
```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
threshold 0.75
q 3
Total Time: 2641.547 milliseconds
Precision= 0.9069767441860465 , Recall= 0.7358490566037735 , Fmeasure= 0.8124999999999999
```

4)



```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLink
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
threshold 0.75
q 4
Total Time: 2222.279 milliseconds
Precision= 0.9058823529411765 , Recall= 0.7264150943396226 , Fmeasure= 0.8062827225130891
```

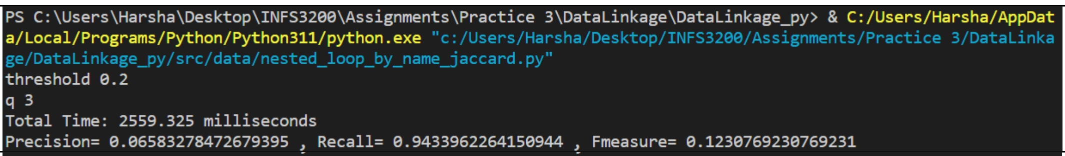
5)



```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
threshold 0.75
q 100
Total Time: 492.218 milliseconds
Precision= 0.9146341463414634 , Recall= 0.7075471698113207 , Fmeasure= 0.7978723404255319
```

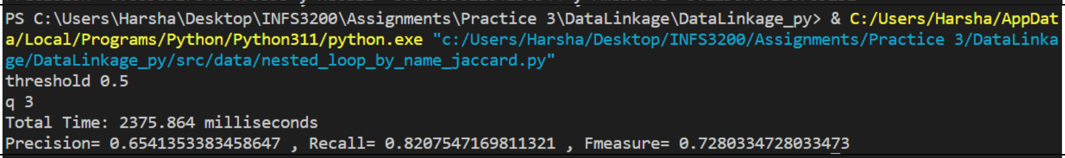
Precision, Recall, F-Measure: Threshold Changed & Q = 3

1)



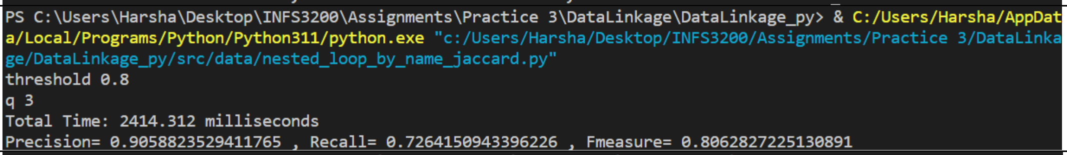
```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
threshold 0.2
q 3
Total Time: 2559.325 milliseconds
Precision= 0.06583278472679395 , Recall= 0.9433962264150944 , Fmeasure= 0.1230769230769231
```

2)



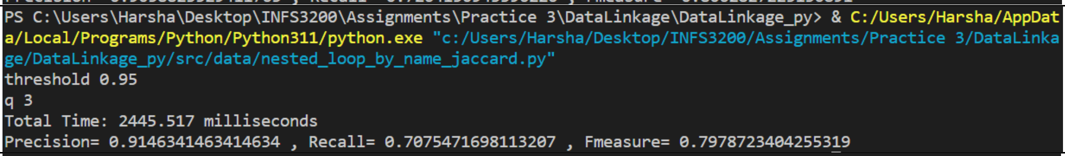
```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
threshold 0.5
q 3
Total Time: 2375.864 milliseconds
Precision= 0.6541353383458647 , Recall= 0.8207547169811321 , Fmeasure= 0.7280334728033473
```

3)



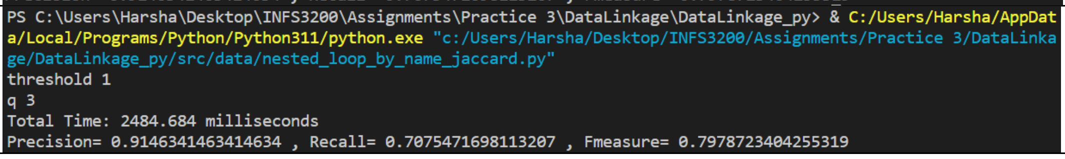
```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
threshold 0.8
q 3
Total Time: 2414.312 milliseconds
Precision= 0.9058823529411765 , Recall= 0.7264150943396226 , Fmeasure= 0.8062827225130891
```

4)



```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
threshold 0.95
q 3
Total Time: 2445.517 milliseconds
Precision= 0.9146341463414634 , Recall= 0.7075471698113207 , Fmeasure= 0.7978723404255319
```

5)



```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
threshold 1
q 3
Total Time: 2484.684 milliseconds
Precision= 0.9146341463414634 , Recall= 0.7075471698113207 , Fmeasure= 0.7978723404255319
```

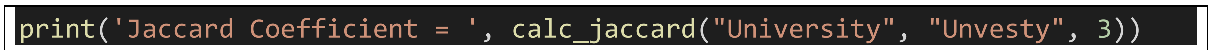
Looking at the above it is evident that the q-value changing between 1 and 2-4 significantly impacts the precision. The precision changes from about 0.09 to 0.87 when the q-gram length goes from 1 to 2 and this is most likely due to comparing string by string (with q = 2) rather than character by character (q = 1). Further, as the q value increases it is evident that the precision (positive predictive value) continues to increase in slight increments – going from 0.90 to 0.91 (between q = 3 to q = 100), as the q-gram tokenises in many more iterations. Similarly, recall follows the opposite trend whereby it is at its highest outcome while q = 1, at 0.80 and steadily digressing to 0.72 by the time q = 4. The outlier value here where q = 100 only ends up changing the recall value to 0.70 despite being 96 times greater than q = 4. This is most likely to do with the sample size, as recall measures sensitivity of the dataset, a smaller dataset is mathematically more viable to lead a better recall as more values are more probable to be truly positive.

In referencing the second table it is evident that while the threshold changes, precision and recall can stabilise significantly due to the greater impact the threshold has on both the precision and recall, as explained previously. It is evident that when the threshold changes between 0.2-0.8. Both the precision and recall experience significant changes going from 0.065 to 0.90 and 0.94 down to 0.72 respectively. While threshold = 0.95 and 1, significantly similar values were demonstrated whereby both precision and recall remained around 0.91 and 0.7 respectively. Thus, it is evident that both these values experience relatively good accuracy while the threshold is greater than 0.95.

Ideally, the best possible values for q and threshold are q = 3 and threshold = 0.95 in order to not alienate either precision or recall's impact on the dataset, considering the above.

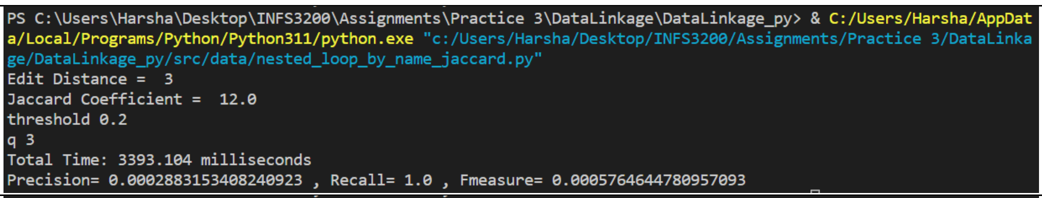
Task 4

```
def calc_ed(str1, str2):

    '''
         * Please implement the calculation of edit distance between two
strings
         * Dynamic programming should be used
         */
    '''
    if len(str1) > len(str2):
        str1, str2 = str2, str1

    short_string_distance = range(len(str1) + 1)
    for i2, c2 in enumerate(str2): #longer string
        long_string_distance = [i2+1]
        for i1, c1 in enumerate(str1): #shorter string
            if c1 == c2:
                long_string_distance.append(short_string_distance[i1])
            else:
                long_string_distance.append(1 +
min((short_string_distance[i1], short_string_distance[i1 + 1],
long_string_distance[-1])))
        short_string_distance = long_string_distance
    return short_string_distance[-1]

print('Edit Distance = ', calc_ed("University", "Unvesty"))
```



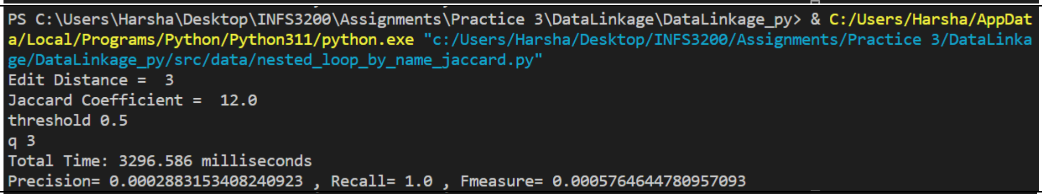
```
print('Jaccard Coefficient = ', calc_jaccard("University", "Unvesty", 3))
```

1)



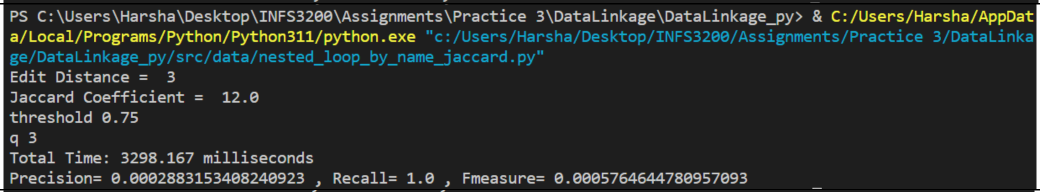
```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
Edit Distance =  3
Jaccard Coefficient =  12.0
threshold 0.2
q 3
Total Time: 3393.104 milliseconds
Precision= 0.0002883153408240923 , Recall= 1.0 , Fmeasure= 0.0005764644780957093
```

2)



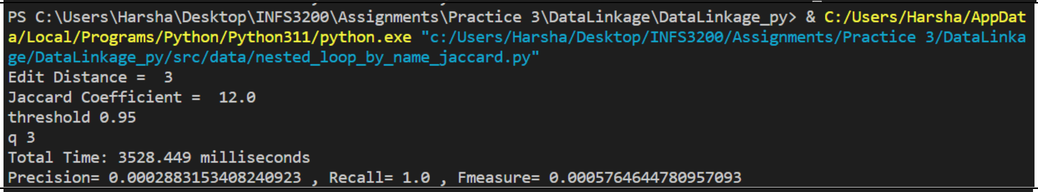
```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
Edit Distance =  3
Jaccard Coefficient =  12.0
threshold 0.5
q 3
Total Time: 3296.586 milliseconds
Precision= 0.0002883153408240923 , Recall= 1.0 , Fmeasure= 0.0005764644780957093
```

3)



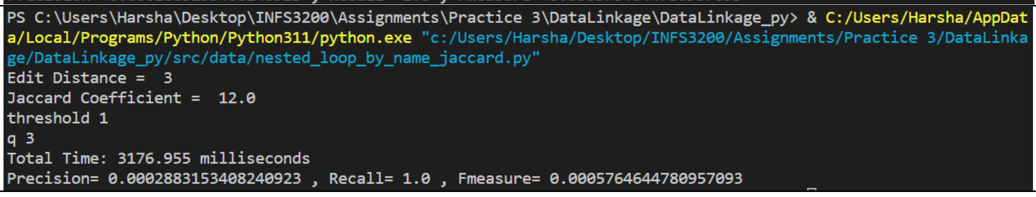
```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
Edit Distance =  3
Jaccard Coefficient =  12.0
threshold 0.75
q 3
Total Time: 3298.167 milliseconds
Precision= 0.0002883153408240923 , Recall= 1.0 , Fmeasure= 0.0005764644780957093
```

4)



```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
Edit Distance =  3
Jaccard Coefficient =  12.0
threshold 0.95
q 3
Total Time: 3528.449 milliseconds
Precision= 0.0002883153408240923 , Recall= 1.0 , Fmeasure= 0.0005764644780957093
```

5)



```
PS C:\Users\Harsha\Desktop\INFS3200\Assignments\Practice 3\DataLinkage\DataLinkage_py> & C:/Users/Harsha/AppDat
a/Local/Programs/Python/Python311/python.exe "c:/Users/Harsha/Desktop/INFS3200/Assignments/Practice 3/DataLinka
ge/DataLinkage_py/src/data/nested_loop_by_name_jaccard.py"
Edit Distance =  3
Jaccard Coefficient =  12.0
threshold 1
q 3
Total Time: 3176.955 milliseconds
Precision= 0.0002883153408240923 , Recall= 1.0 , Fmeasure= 0.0005764644780957093
```

From the above, it is evident that the precision, recall and f-measure remain the same given varying threshold values with the q value remaining consistent at the value of 3. The strings that were tested on where "University" and "Unvesty" as specified in the task sheet to achieve the outcome of these values. The major changing factor here is the time taken, which can be linked to the fact that the threshold values change the benchmark for the two string values to aspire to. It is evident that the higher the threshold, the less time is taken to process the string query.

References

GoogleforDevelopers (2022). Retrieved from https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative#:~:text=True%20Negative%20(TN)%3A&amp;text=A%20true%20positive%20is%20an,incorrectly%20predicts%20the%20positive%20class

Ramos, L. P. (2022). Getters and setters: Manage attributes in Python. Retrieved from https://realpython.com/python-getter-setter/

Safari, S., Baratloo, A., Elfil, M., & Negida, A. (2015). Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4608333/#:~:text=Positive%20predictive%20value%3A&text=It%20is%20the%20ratio%20of,of%20a%20positive%20test%20result

Wikipedia (2023). Retrieved from https://en.wikipedia.org/wiki/Sensitivity_and_specificity