PATH OPTIMIZATION FOR DELIVERING PACKAGES

TEAM ID: 8

VINEETH DUDIPALLI
VENKATA SAI TEJA YERRANAGULA
ABILASH PANDIT

MOTIVATION:

- Generally delivering companies have to deliver goods in a lot of cities.
- Every customer expects their package to arrive as soon as possible.
- So we need a path which is optimized for the company to deliver goods in every place while travelling the shortest distance.

PROJECT IDEA:

- The dataset contains a list of cities and their coordinates in a csv file.
- We have to find the shortest path possible which visits each city exactly once.
- The output is a ordered list in which we visit each city.

TECHNOLOGIES USED:

- Pycharm
- Anaconda
- Pandas Library
- Matplotlib
- Concorde TSP Solver
- Turtle Graphics

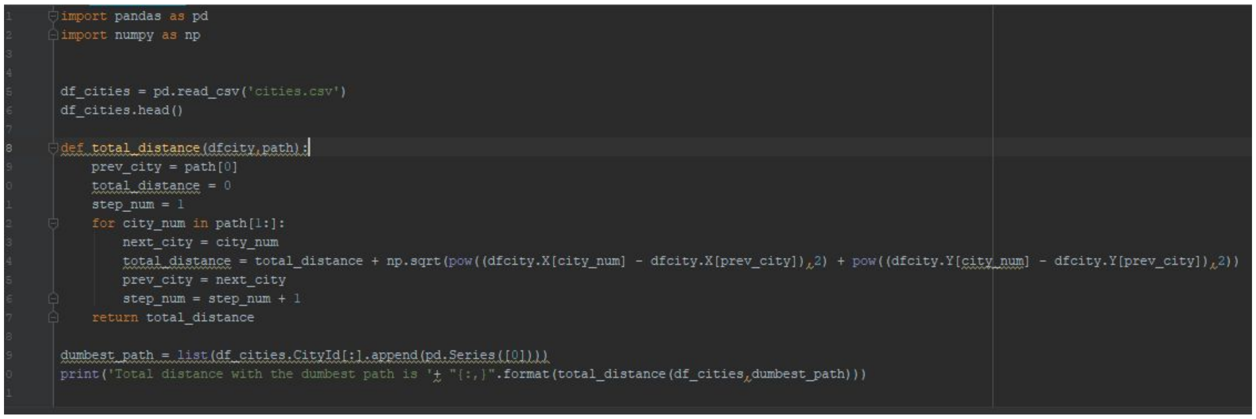
APPROACH:

- We will be testing the data with different types of approaches(around 5-10).
- We will start with a basic or random approach by taking the order of city Id and we assume it to be the worst case.
- The next approaches will be the optimized version of the path which will have a shorter distance when compared to the previous approach.

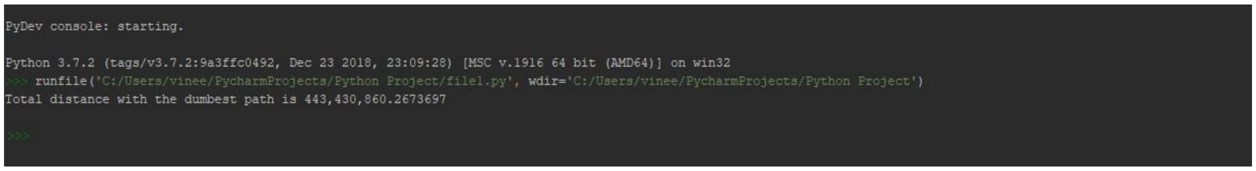
SOLUTION & RESULTS:

- We have considered around six different approaches to find the optimized path.

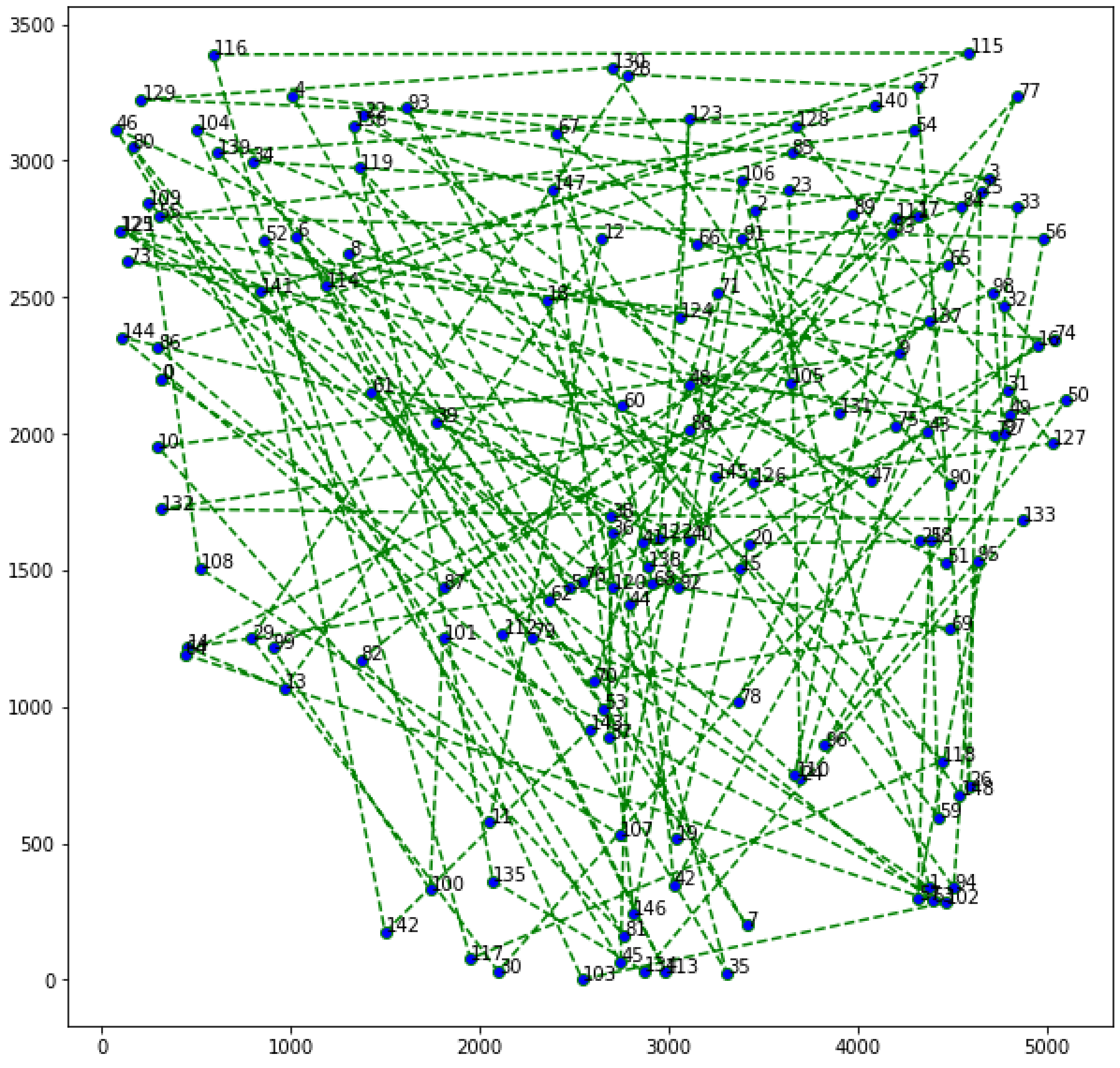
1) **DUMBEST PATH:** This is the worst case scenario where we find the path in the increasing order of city ID.



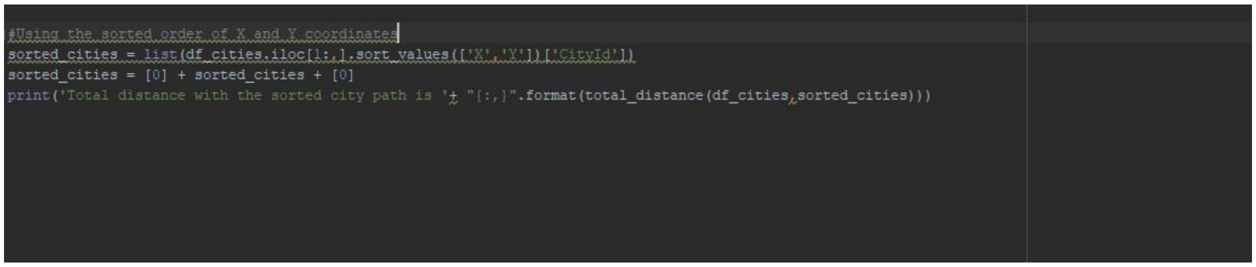
```python
import pandas as pd
import numpy as np


df_cities = pd.read_csv('cities.csv')
df_cities.head()

def total_distance(dfcity,path):
    prev_city = path[0]
    total_distance = 0
    step_num = 1
    for city_num in path[1:]:
        next_city = city_num
        total_distance = total_distance + np.sqrt(pow((dfcity.X[city_num] - dfcity.X[prev_city]),2) + pow((dfcity.Y[city_num] - dfcity.Y[prev_city]),2))
        prev_city = next_city
        step_num = step_num + 1
    return total_distance

dumbest_path = list(df_cities.CityId[:].append(pd.Series([0])))
print('Total distance with the dumbest path is '+ "{:,}".format(total_distance(df_cities,dumbest_path)))
```



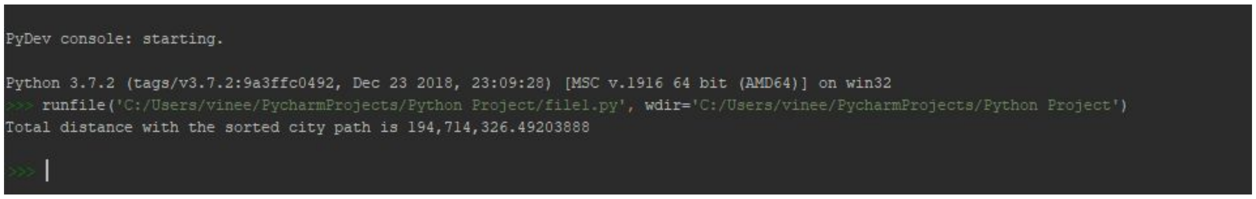
```
PyDev console: starting.

Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
>>> runfile('C:/Users/vinee/PycharmProjects/Python Project/file1.py', wdir='C:/Users/vinee/PycharmProjects/Python Project')
Total distance with the dumbest path is 443,430,860.2673697

>>>
```



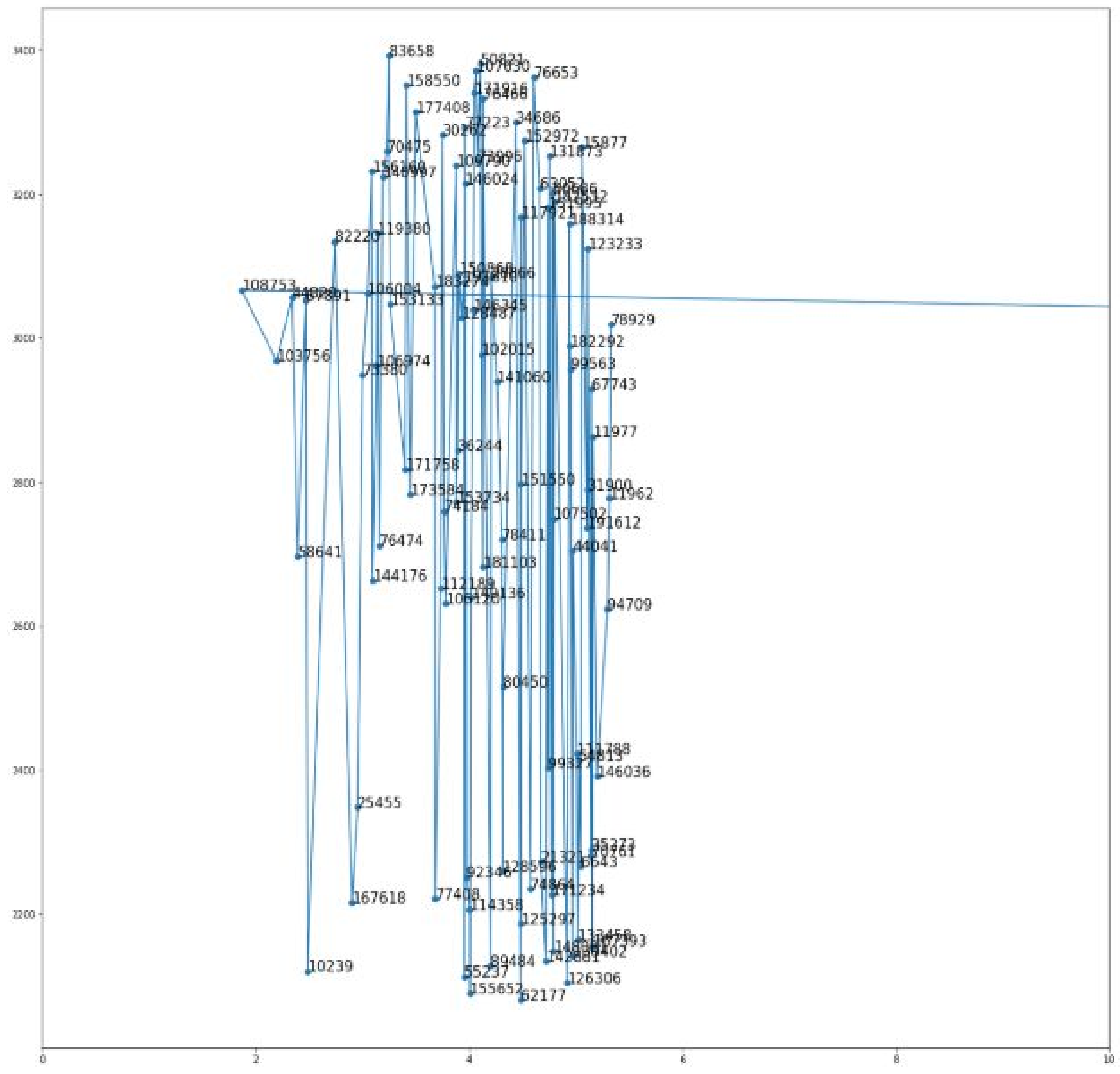
2) **<u>SORTING WITH X,Y:</u>** In this Scenario, we sort the data using X and Y coordinates of the cities.



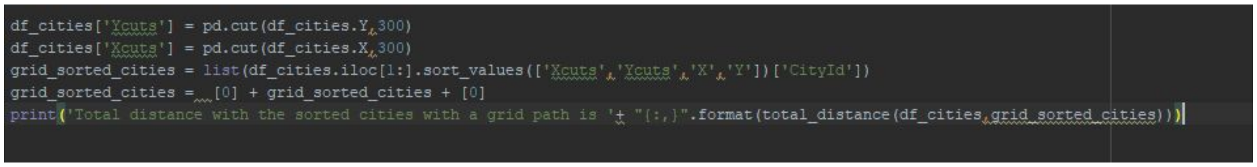
```
#Using the sorted order of X and Y coordinates
sorted_cities = list(df_cities.iloc[1:,].sort_values(['X','Y'])['CityId'])
sorted_cities = [0] + sorted_cities + [0]
print('Total distance with the sorted city path is '+ "{:,}".format(total_distance(df_cities,sorted_cities)))
```



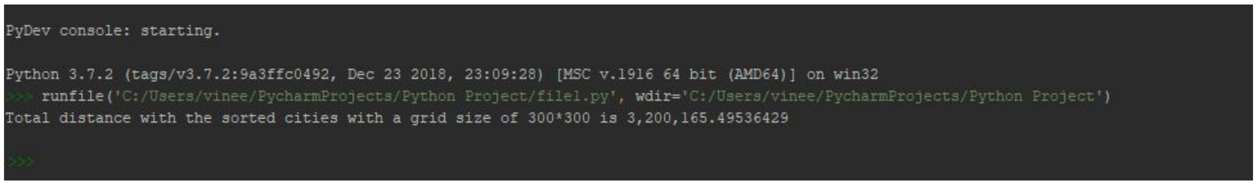
```
PyDev console: starting.

Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
>>> runfile('C:/Users/vinee/PycharmProjects/Python Project/file1.py', wdir='C:/Users/vinee/PycharmProjects/Python Project')
Total distance with the sorted city path is 194,714,326.49203888

>>> 
```



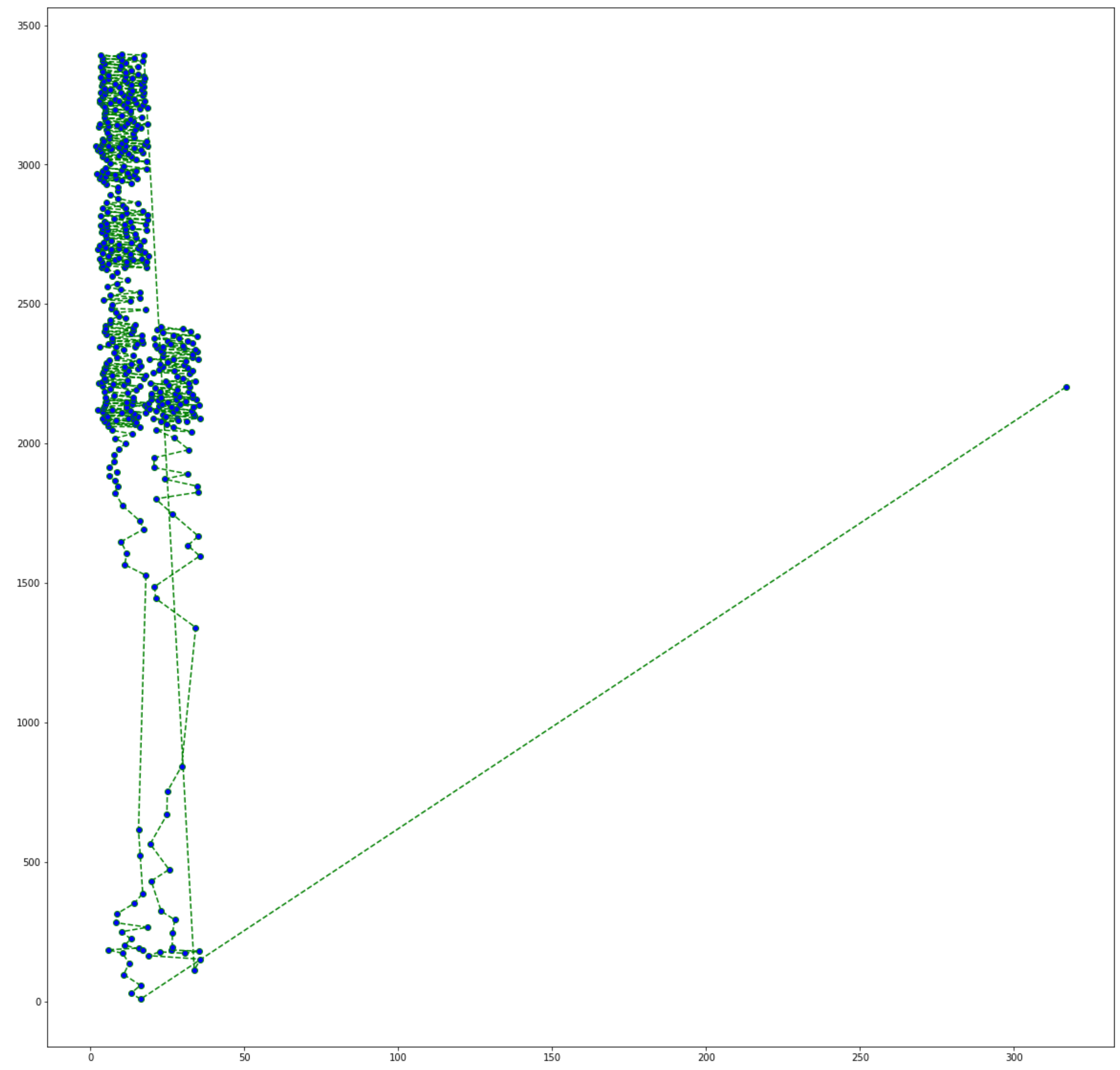
3) **<u>SORTING USING GRIDS:</u>** In this scenario we divide the whole data into grids of certain size and then sort the data in a particular grid. Now we can find the path by considering each grid.



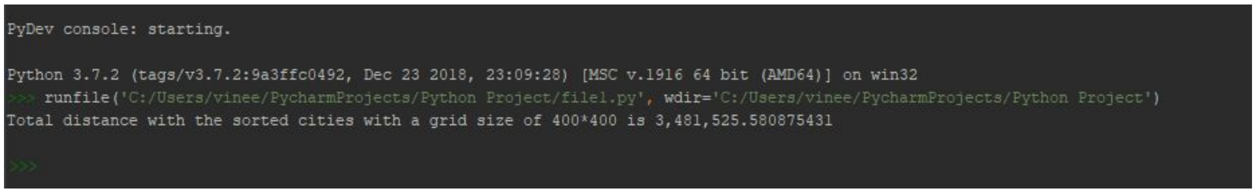
```
df_cities['Ycuts'] = pd.cut(df_cities.Y,300)
df_cities['Xcuts'] = pd.cut(df_cities.X,300)
grid_sorted_cities = list(df_cities.iloc[1:].sort_values(['Xcuts','Ycuts','X','Y'])['CityId'])
grid_sorted_cities =  [0] + grid_sorted_cities + [0]
print('Total distance with the sorted cities with a grid path is '+ "{:,}".format(total_distance(df_cities,grid_sorted_cities)))
```



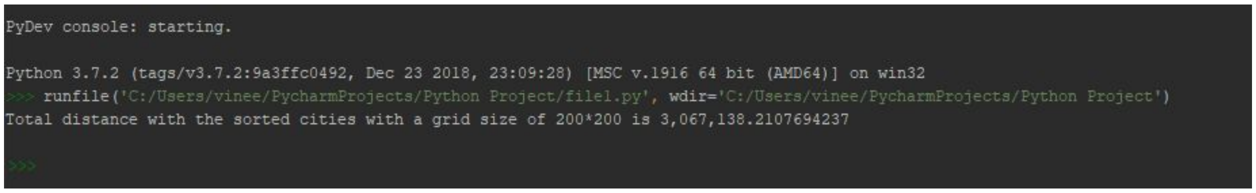
```
PyDev console: starting.

Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
>>> runfile('C:/Users/vinee/PycharmProjects/Python Project/file1.py', wdir='C:/Users/vinee/PycharmProjects/Python Project')
Total distance with the sorted cities with a grid size of 300*300 is 3,200,165.49536429

>>>
```


3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300

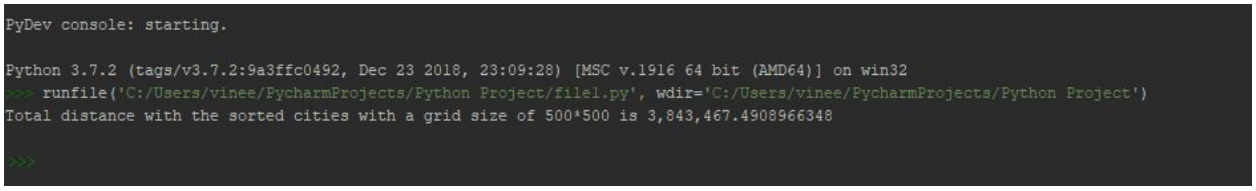
4) **<u>GRID OPTIMIZATION:</u>** Here we improve the above mentioned method by taking grids of different sizes.



```
PyDev console: starting.

Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
>>> runfile('C:/Users/vinee/PycharmProjects/Python Project/file1.py', wdir='C:/Users/vinee/PycharmProjects/Python Project')
Total distance with the sorted cities with a grid size of 400*400 is 3,481,525.580875431

>>>
```

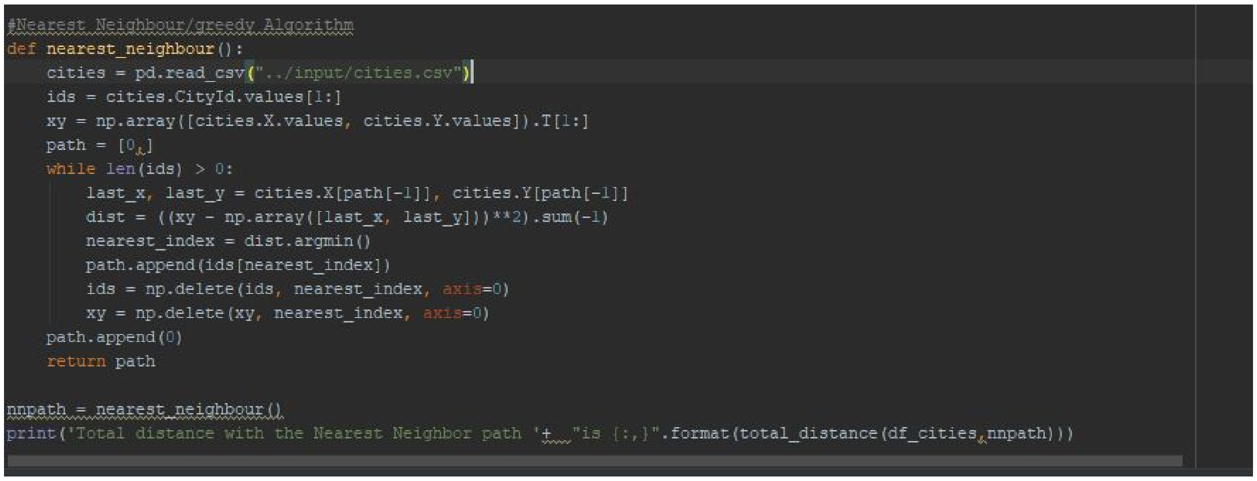


```
PyDev console: starting.

Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
>>> runfile('C:/Users/vinee/PycharmProjects/Python Project/file1.py', wdir='C:/Users/vinee/PycharmProjects/Python Project')
Total distance with the sorted cities with a grid size of 200*200 is 3,067,138.2107694237

>>>
```

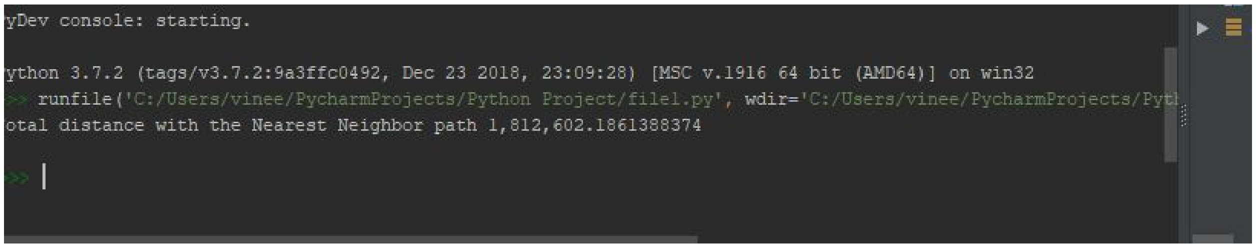


```
PyDev console: starting.

Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
>>> runfile('C:/Users/vinee/PycharmProjects/Python Project/file1.py', wdir='C:/Users/vinee/PycharmProjects/Python Project')
Total distance with the sorted cities with a grid size of 500*500 is 3,843,467.4908966348

>>>
```

5) **<u>NEAREST NEIGHBOUR:</u>** In this scenario we use the method of nearest neighbour or greedy algorithm. In this algorithm, the next city is chosen as the city which is closest to the current city in terms of euclidean distance.



```
#Nearest Neighbour/greedy Algorithm
def nearest_neighbour():
    cities = pd.read_csv("../input/cities.csv")
    ids = cities.CityId.values[1:]
    xy = np.array([cities.X.values, cities.Y.values]).T[1:]
    path = [0,]
    while len(ids) > 0:
        last_x, last_y = cities.X[path[-1]], cities.Y[path[-1]]
        dist = ((xy - np.array([last_x, last_y]))**2).sum(-1)
        nearest_index = dist.argmin()
        path.append(ids[nearest_index])
        ids = np.delete(ids, nearest_index, axis=0)
        xy = np.delete(xy, nearest_index, axis=0)
    path.append(0)
    return path

nnpath = nearest_neighbour()
print('Total distance with the Nearest Neighbor path '+ "is {:,}".format(total_distance(df_cities,nnpath)))
```



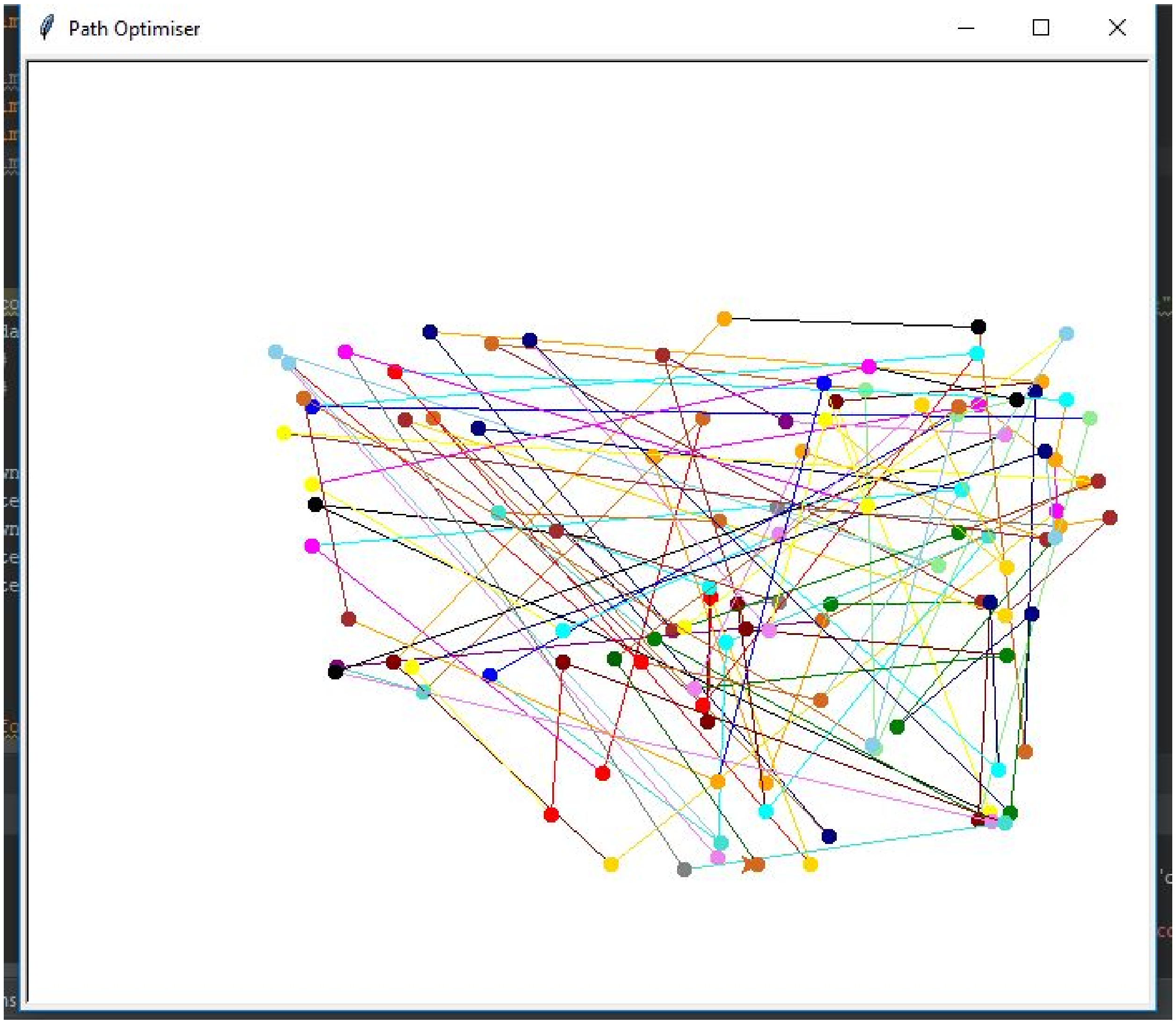
```
yDev console: starting.

ython 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
>> runfile('C:/Users/vinee/PycharmProjects/Python Project/file1.py', wdir='C:/Users/vinee/PycharmProjects/Pyt
otal distance with the Nearest Neighbor path 1,812,602.1861388374

>> |
```

<u>**CONCLUSIONS:**</u>

- We can see from the results that the dumbest path is indeed the worst case scenario.
- There is a huge drop in total distance when we choose other methods over the initial dumbest path method.
- Among all the techniques used, Nearest Neighbour gives the best result.

<u>**MOVING TURTLE GRAPHICS:**</u>

- We added one more visualization method to the project called moving turtle graphics which shows a turtle that moves from one city to another city on the graph upon execution.



CHALLENGES FACED:

- The data set is huge so the program takes a lot of time to execute.
- Due to the time constraint we were unable to add some more features to the project like weather factors etc.

FUTURE WORK:

- We would like to add other factors which effect the path like weather conditions in cities.
- Apart from that we can also add multiple deliveries who cover some amount of cities respectively.