Submission
✔ **Ran successfully**
Submitted by Seshadri 4 months ago

Public Score
1811953.68248

Code Output

In this kernel we try to get a basic understanding of the problem and explore a few simple solutions to see a how they differ.

In case you haven't already figured it out, this is a version of the "Traveling Salesman Problem" (https://en.wikipedia.org/wiki/Travelling_salesman_problem). May be the winning solution is going to be one of the very advanced approaches to that problem. But let us not get scared by an NP-Complete problem. We will start with some simple dumb approaches first and see where they lead us.

Let us first load the file and look at the data.

In [2]:
```
df_cities = pd.read_csv('../input/cities.csv')
df_cities.head()
```

Out[2]:

	CityId	X	Y
0	0	316.836739	2202.340707
1	1	4377.405972	336.602082
2	2	3454.158198	2820.053011
3	3	4688.099298	2935.898056
4	4	1010.696952	3236.750989

Taking a cue from the other kernels, let us first do a scatter plot of the locations of all the cities.

- There is a surpise waiting in the locations of the cities :)

Code

So it looks like the cities are arranged in a circles pattern

So it looks like the cities are arranged in a reindeer pattern.

- The red dot indicates the North Pole (CityId = 0).
- All we have to a do a is a find a path that goes from the red dot, touches all the other dots, and comes back to the red-dot, with minimum total distance travelled !

Dumbest Path: Go in the order of CityIDs: 0, 1, 2.. etc. and come back to zero when you reach the end.

Per the problem: **"submission is scored on the Euclidean distance of your submitted path, subject to the constraint that every 10th step is 10% more lengthy unless coming from a prime CityId."**

To see how good this path is, we need a couple of functions:

1. A function to tell if a number is a prime
2. Another funtion to compute the total distance given a series of numbers

In [4]:
```
# To improve the performance, instead of checking whether each membe
r is a prime,
# we first a generate a list where each element tells whether the nu
mber indicated
# by the position is a prime or not.

# using sieve of eratosthenes
def sieve_of_eratosthenes(n):
    primes = [True for i in range(n+1)] # Start assuming all number
s are primes
    primes[0] = False # 0 is not a prime
    primes[1] = False # 1 is not a prime
    for i in range(2,int(np.sqrt(n)) + 1):
        if primes[i]:
            k = 2
            while i*k <= n:
                primes[i*k] = False
                k += 1
    return(primes)
prime_cities = sieve_of_eratosthenes(max(df_cities.CityId))
```

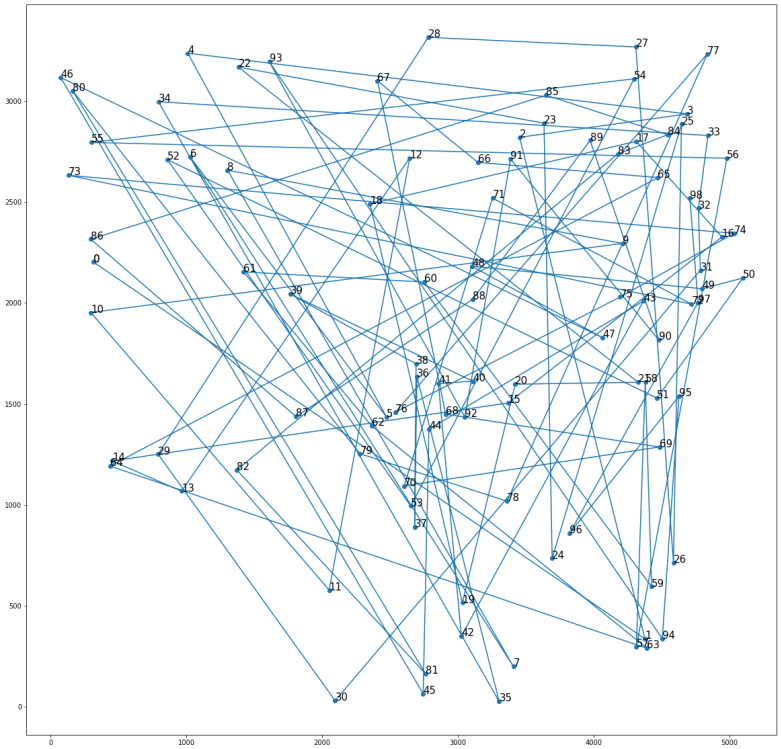
In [5]:
```
def total_distance(dfcity,path):
    prev_city = path[0]
    total_distance = 0
    step_num = 1
    for city_num in path[1:]:
        next_city = city_num
        total_distance = total_distance + \
            np.sqrt(pow((dfcity.X[city_num] - dfcity.X[prev_city
]),2) + pow((dfcity.Y[city_num] - dfcity.Y[prev_city]),2)) * \
            (1+ 0.1*((step_num % 10 == 0)*int(not(prime_cities[pre
v_city]))))
        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(to
tal_distance(df_cities,dumbest_path)))
```

```
Total distance with the dumbest path is 446,884,407.5212135
```

Let us take a look at the first 100 steps of the dumbest path

Code



As we can see, the dumbest path seems pretty bad. We are sending Santa all over the map, without any consideration for him whatsoever :)

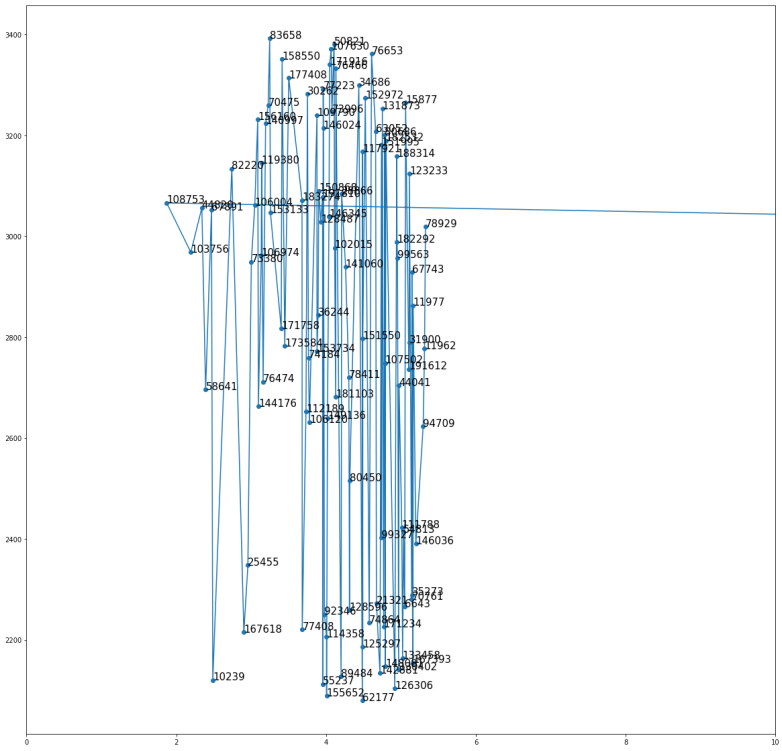
Slightly better path: sort the cities in X,Y coordinate_order

In [7]:
```
sorted_cities = list(df_cities.iloc[1:,].sort_values(['X','Y'])['C
ityId'])
sorted_cities = [0] + sorted_cities + [0]
print('Total distance with the sorted city path is '+ "{:,}".forma
t(total_distance(df_cities,sorted_cities)))
```

```
Total distance with the sorted city path is 196,478,811.25956938
```

We already reduced our total distance by more than half using a simple sorting !!
Let us look at how it looks in a map (zoomed in) .

Code



From the plot above, we can also see that now Santa is going up and down around the same X value (note X axis only ranges from 0-10), before moving to the next X. This is slightly more efficient path.

Question for more experienced matplotlib users: Can anyone tell me why I am seeing a big white space above this figure ?

Iteration 3 : Sorted cities within a grid of squares

One of the problems we can see with the previous Path is that though Santa is moving in a systematic left-to-right fashion, he is moving between the extremes of the Y-axis while doing that.

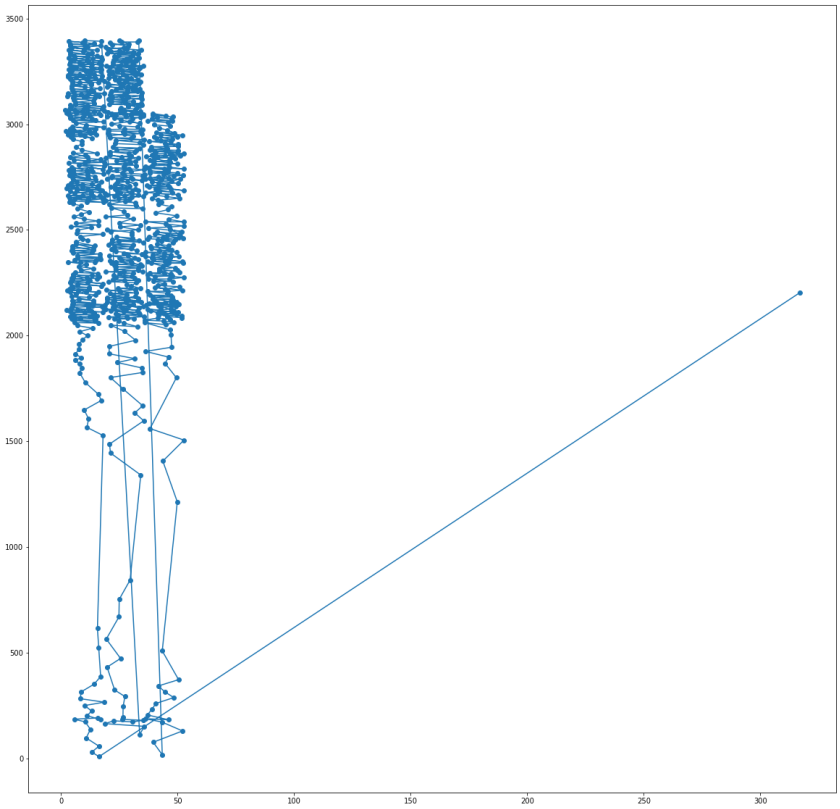
If we divide the the whloe network into a grid of X's and Y's, then Santa can cover each square in the grid, before moving on to the next square. It should be more efficient as shown below.

In [9]:
```
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)))
```

```
Total distance with the sorted cities with a grid path is 3,226,33
1.4903367283
```

Code

Out[10]:
```
[<matplotlib.lines.Line2D at 0x7f065060d470>]
```



Great improvement with sliced, sorted cities in terms of score.

We can now see that Santa is going to a point, trying to cover many of the cities around that point, before moving up. This feels like a better way to cover all the cities and gives better results as expected.

Iteration 4: Grid of Squares + Zig Zag Scanning

One of the above problems we can see in the above path is that once Santa reaches the city with the highest Y, he comes back all the way to city with lowest-Y and starts moving up, as opposed, following a zig-zag pattern which should be more efficient.

- Let us see if we an implement a zig-zag path and get some more improvement.

In [11]:
```
zigzag_sorted_cities1 = list(df_cities.iloc[1:].sort_values(['Xcut
s','Ycuts','X','Y'])['CityId'])
zigzag_sorted_cities2 = list(df_cities.iloc[1:].sort_values(['Xcut
s','Ycuts','X','Y'], ascending = [True,False,True,True])['CityId'
])
chooser_pattern = list(df_cities.iloc[1:].sort_values(['Xcuts']).g
roupby(['Xcuts']).ngroup()%2)

zigzag_cities = [zigzag_sorted_cities1[i] if chooser_pattern[i] ==
0 else zigzag_sorted_cities2[i] for i in range(len(chooser_pattern
))]
zigzag_cities =  [0] + zigzag_cities + [0]
print('Total distance with the Zig-Zag with grid city path is '+ "
{:,}".format(total_distance(df_cities,zigzag_cities)))
```

```
Total distance with the Zig-Zag with grid city path is 2,233,635.0
```

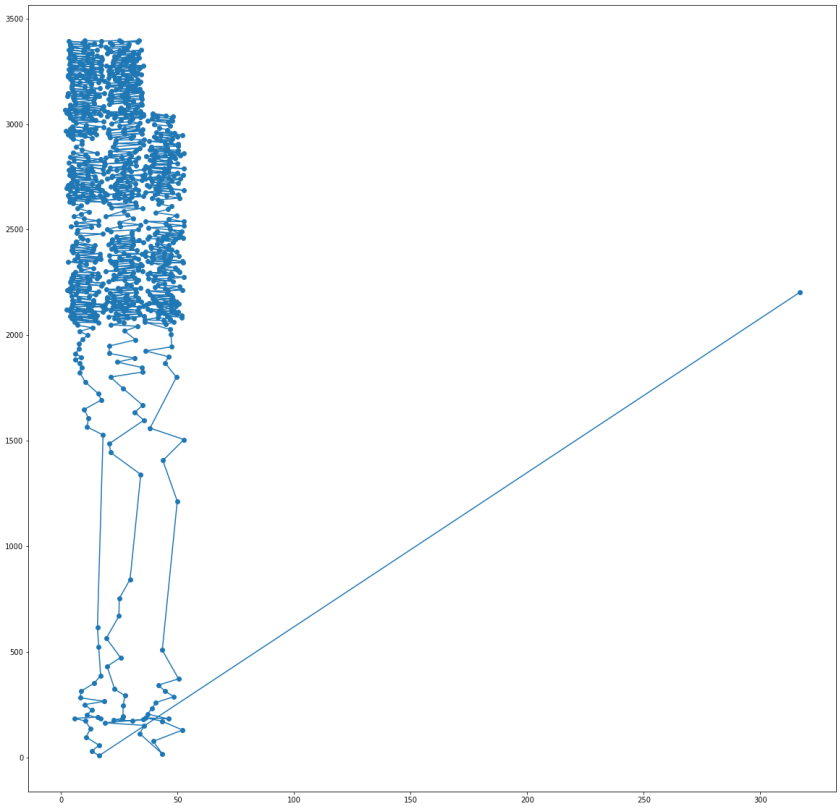
```
317220334
```

By introducing a zig-zag path at the grid-level, we eliminated those long trips for Santa from highest Y to lowest-Y and further improved the score from 3.2 million to 2.2 million.

Let us see how this path looks.

In [12]:
```
df_path = pd.DataFrame({'CityId':zigzag_cities}).merge(df_cities,h
ow = 'left')
fig, ax = plt.subplots(figsize=(20,20))
ax.plot(df_path.iloc[0:1000,]['X'], df_path.iloc[0:1000,]['Y'],mar
ker = 'o')
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x7f06479743c8>]
```



We sort of arbitrarily chose the grid size of 300x300 grid size. Next let us see if the grid size makes a difference

Iteration #5 Optimizing for grid-size:

Code

```
Total distance with the Zig-Zag with grid city path with grid
size (100,100) is 3,243,001.24552974
Total distance with the Zig-Zag with grid city path with grid
size (100,200) is 2,483,012.2590593975
Total distance with the Zig-Zag with grid city path with grid
size (100,300) is 2,458,138.678978236
Total distance with the Zig-Zag with grid city path with grid
size (100,400) is 2,600,638.502037033
Total distance with the Zig-Zag with grid city path with grid
size (100,500) is 2,779,870.5123764514
```

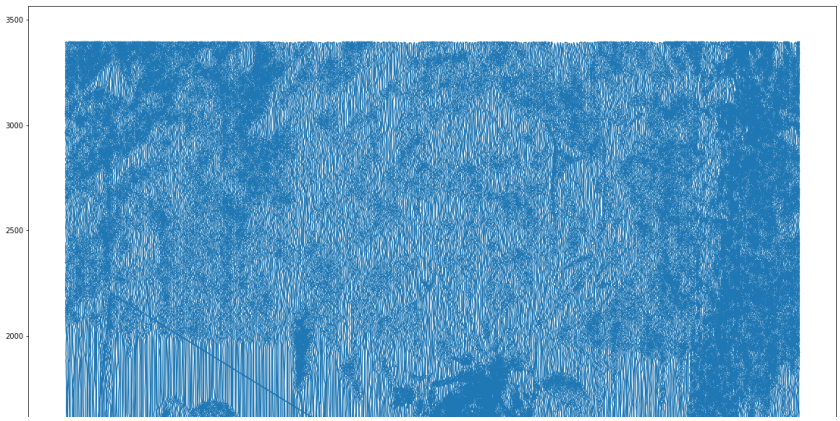
```
size (100,300) is 2,779,870.5123764514
Total distance with the Zig-Zag with grid city path with grid
size (200,100) is 3,354,740.6090092813
Total distance with the Zig-Zag with grid city path with grid
size (200,200) is 2,429,013.5317182303
Total distance with the Zig-Zag with grid city path with grid
size (200,300) is 2,247,440.072765479
Total distance with the Zig-Zag with grid city path with grid
size (200,400) is 2,242,396.9024366746
Total distance with the Zig-Zag with grid city path with grid
size (200,500) is 2,281,677.0080440654
Total distance with the Zig-Zag with grid city path with grid
size (300,100) is 3,526,422.0913899704
Total distance with the Zig-Zag with grid city path with grid
size (300,200) is 2,509,184.775228575
Total distance with the Zig-Zag with grid city path with grid
size (300,300) is 2,233,635.0317220334
Total distance with the Zig-Zag with grid city path with grid
size (300,400) is 2,150,771.3569796314
Total distance with the Zig-Zag with grid city path with grid
size (300,500) is 2,122,787.4517501732
Total distance with the Zig-Zag with grid city path with grid
size (400,100) is 3,728,141.505106538
Total distance with the Zig-Zag with grid city path with grid
size (400,200) is 2,663,878.015409067
Total distance with the Zig-Zag with grid city path with grid
size (400,300) is 2,333,905.262941517
Total distance with the Zig-Zag with grid city path with grid
size (400,400) is 2,201,697.109105353
Total distance with the Zig-Zag with grid city path with grid
size (400,500) is 2,137,198.9506304367
Total distance with the Zig-Zag with grid city path with grid
size (500,100) is 3,946,734.9490872365
Total distance with the Zig-Zag with grid city path with grid
size (500,200) is 2,853,307.7209141403
Total distance with the Zig-Zag with grid city path with grid
size (500,300) is 2,492,626.5269496064
Total distance with the Zig-Zag with grid city path with grid
size (500,400) is 2,334,451.262207789
Total distance with the Zig-Zag with grid city path with grid
size (500,500) is 2,253,162.959148464
```

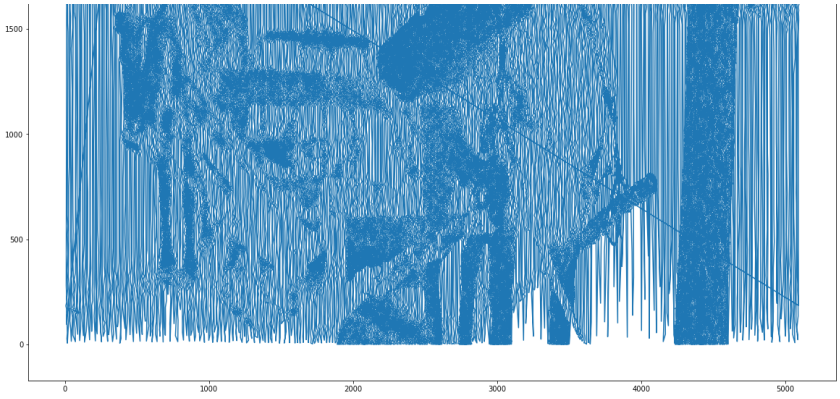
In [14]:

```
df_path = pd.DataFrame({'CityId':zigzag_cities}).merge(df_cities,h
ow = 'left')
fig, ax = plt.subplots(figsize=(20,20))
ax.plot(df_path['X'], df_path['Y'])
```

Out[14]:

```
[<matplotlib.lines.Line2D at 0x7f0647532dd8>]
```





It looks like (300,500) is better grid-size than (300,300), but not my much. Well, all experiments don't work :)¥

Let us take a look on how the final graph looks with the above algorithm.

Iteration 6: Nearest Neighbor / Greedy Algorithm:

Wikipedia link (https://en.wikipedia.org/wiki/Travelling_salesman_problem#Computing_a_solution) says Nearest Neighbor approach is one of the easier heuristic approaches to solve the traveling salesman problem. Let us see how much improvement we can get with it.

For the nearest neighbor algorithm, we will use the function from XYZT's Kernel in the same competition: (https://www.kaggle.com/thexyzt/xyzt-s-visualizations-and-various-tsp-solvers)

In [15]:
```
# Function from XYZT's Kernel on the same topic.
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)))
```

```
Total distance with the Nearest Neighbor path is 1,812,602.1861388
374
```

Nearest neighbor algorithm offered a significant improvement from a cost of about 2.2 million to 1.8 million.

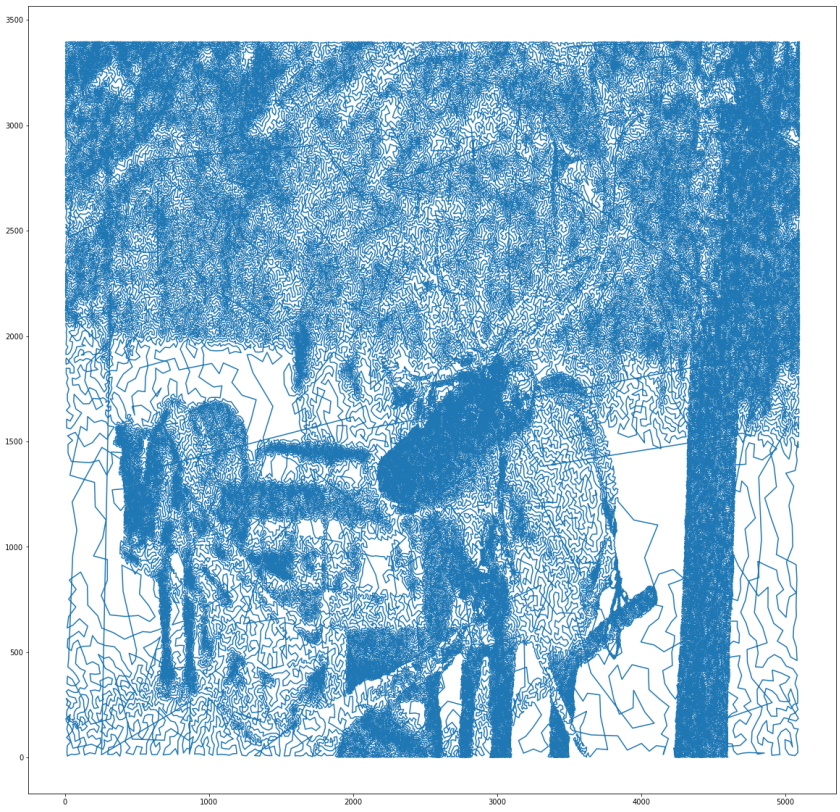
Let us see how the path looks with the neaerst neighbor alrgorithm.

In [16]:
```
df_path = pd.DataFrame({'CityId':nnpath}).merge(df_cities,how = 'l
```

```
eft')
fig, ax = plt.subplots(figsize=(20,20))
ax.plot(df_path['X'], df_path['Y'])
```

Out[16]:

```
[<matplotlib.lines.Line2D at 0x7f06470fc9b0>]
```



Iteration 7: Nearest Neighbor / Greedy Algorithm With Prime Swaps

So far we haven't used the prime-number related constraint on the path for the optimization. It says "every 10th step is 10% more lengthy unless coming from a prime CityId". So, it is in our interest to make sure that the prime numbers end up at the beginning of every 10th step.

Approach:

have an index ending in 9, or two cities that occur after the current city and have an index ending in 9.

- For example if we have a prime city in the 76th place, we will try to see if that can be swapped with any of the cities in the 59th, 69th, 79th and 89th place and result in a shorter path.

- When checking to see if the path becomes shorter by swapping the cities, we will only check the length of the sub-path that corresponds to the swap. No need to check the length of the entire path. This will make the search lot more efficient.

In [17]:

```
nnpath_with_primes = nnpath.copy()
for index in range(20,len(nnpath_with_primes)-30):
    city = nnpath_with_primes[index]
    if (prime_cities[city] &  ((index+1) % 10 != 0)):
```

```
        for i in range(-1,3):
            tmp_path = nnpath_with_primes.copy()
            swap_index = (int((index+1)/10) + i)*10 - 1
            tmp_path[swap_index],tmp_path[index] = tmp_path[index
],tmp_path[swap_index]
            if total_distance(df_cities,tmp_path[min(swap_index,in
dex) - 1 : max(swap_index,index) + 2]) < total_distance(df_cities,
nnpath_with_primes[min(swap_index,index) - 1 : max(swap_index,inde
x) + 2]):
                nnpath_with_primes = tmp_path.copy()
                break
print('Total distance with the Nearest Neighbor With Prime Swaps '
+  "is {:,}".format(total_distance(df_cities,nnpath_with_primes)))
```

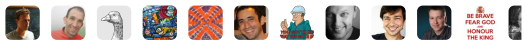
```
Total distance with the Nearest Neighbor With Prime Swaps is 1,81
1,953.6824856116
```

We reduced the total distance by another 650 units with prime swaps.

Let us submit the best path we have so far to the competition ...

In [18]:
```
pd.DataFrame({'Path':nnpath_with_primes}).to_csv('nnpath_with_prim
es.csv',index  = False)
```

** More to come .. **



Output Files

New Dataset | New Kernel | Download All

Output Files

nnpath_with_primes.csv

About this file

This file was created from a Kernel, it does not have a description.

nnpath_with_primes.csv

	Path
1	Path
2	0
3	78934
4	111804
5	52086
6	18295
7	134585
8	109090
9	37150
10	96442
11	126815
12	122542
13	142366
14	25283
15	117533
16	5333
17	69414
18	2187
19	177800
20	95592
21	124518
22	167893
23	97649
24	71921
25	131753
26	77028
27	12492
28	93526
29	11752
30	145197
31	186667
32	97580

Run Info

Succeeded	True	Run Time	1115.8 seconds
Exit Code	0	Queue Time	0 seconds
Docker Image Name	kaggle/python(Dockerfile)	Output Size	0
Timeout Exceeded	False	Used All Space	False
Failure Message			

Log

Download Log

```
Time  Line #  Log Message
3.9s        1  [NbConvertApp] Converting notebook script.ipynb to html
4.0s        2  [NbConvertApp] Executing notebook with kernel: python3
1114.9s     3  [NbConvertApp] Support files will be in __results___files/
               [NbConvertApp] Making directory __results___files
1114.9s     4  [NbConvertApp] Making directory __results___files
1114.9s     5  [NbConvertApp] Making directory __results___files
               [NbConvertApp] Making directory __results___files
1114.9s     6  [NbConvertApp] Making directory __results___files
               [NbConvertApp] Making directory __results___files
1114.9s     7  [NbConvertApp] Making directory __results___files
1114.9s     8  [NbConvertApp] Writing 325408 bytes to __results__.html
1114.9s     9
1114.9s    11  Complete. Exited with code 0.
```

Comments (24)

Sort by

All Comments | Hotness

Click here to enter a comment...

Rajnish Singh • Posted on Version 15 • 4 months ago • Options • Reply 1

Succinctly explained! Awesome part is no TSP used.

Seshadri Kernel Author • Posted on Version 15 • 4 months ago • Options • Reply 0

Thanks. I am trying to take into the prime number constraint as well, without using the TSP solvers. Let us see how it goes.

Aamir Siddiq... • Posted on Version 15 • 5 months ago • Options • Reply 0

very helpful. Thanks for uploading.

Seshadri Kernel Author • Posted on Version 15 • 5 months ago • Options • Reply 0

Thanks !

Steve Chadw... • Posted on Version 15 • 5 months ago • Options • Reply 0

Nice introduction to the problem, I really appreciate the clear explanations of the different approaches.

Seshadri Kernel Author • Posted on Version 15 • 4 months ago • Options • Reply 0

Thanks.

zyfghx • Posted on Latest Version • 4 months ago • Options • Reply 0

very useful, Thanks for sharing

Seshadri Kernel Author • Posted on Latest Version • 4 months ago • Options • Reply 0

Thanks.

fermion • Posted on Latest Version • 4 months ago • Options • Reply 0

Very nice and concise introduction to the problem. I really enjoy reading it. Thanks!

Seshadri Kernel Author • Posted on Latest Version • 4 months ago • Options • Reply 0

Thanks.

Ram sahu • Posted on Latest Version • 4 months ago • Options • Reply 0

Awesome explanatory kernel

aekapolh • Posted on Latest Version • 4 months ago • Options • Reply 0

Thanks for sharing

Ozay OKUMU... • Posted on Latest Version • 4 months ago • Options • Reply 0

def sieve*of*eratosthenes(n): not fully understand this function. especially k

Özgür Ce... • Posted on Latest Version • 4 months ago • Options • Reply 0

It's an algorithm that finds all primes up to n. I believe the complexity of this implementation is O(n*sqrt(n)). It can be modified to make the complexity O(n) but its not that important since n~200k and the algorithm will find the primes in couple of seconds.

daydreamer • Posted on Latest Version • 4 months ago • Options • Reply 0

It was helpful. Thanks.

Jerry Thomas • Posted on Latest Version • 4 months ago • Options • Reply 0

Very nice explanation and the progressive improvement in the distance.

fzhurd • Posted on Latest Version • 4 months ago • Options • Reply 0

good job, thanks

Taylor • Posted on Latest Version • 4 months ago • Options • Reply 0

Awesome explanatory kernel! appreciate the clear explanations of the different approaches!

Alan Zhang • Posted on Latest Version • 4 months ago • Options • Reply 0

Thanks for your sharing

nick • Posted on Latest Version • 3 months ago • Options • Reply 0

thanks for the nice intro!

Mykhaylo Ku... • Posted on Latest Version • 3 months ago • Options • Reply 0

Very nice explanation. I just started to learn data science and not understand all code but I got the idea how it works, and I like it. Thanks guys!

Suhail Aham... • Posted on Latest Version • 3 months ago • Options • Reply 0

You can fix the white space in the 3rd plot by not annotating the 0th city,

df*path = pd.DataFrame({'CityId':sorted*cities}).merge(df*cities,how = 'left') fig, ax = plt.subplots(figsize=(20,20)) ax.set*xlim(0,10)
ax.plot(df*path.iloc[0:100,]['X'], df*path.iloc[0:100,]['Y'],marker = 'o')
for i, txt in enumerate(df*path.iloc[0:100,]['CityId']): if i !=0: ax.annotate(txt, (df*path.iloc[0:100,]['X'][i], df_path.iloc[0:100,]['Y'][i]),size = 15)

4 months ago

This Comment was deleted.

Seshadri Kernel Author • Posted on Version 16 • 4 months ago • Options • Reply 0

Thanks.

Similar Kernels