MST.py main.py cities.py M

cities.py > get_cities

```python
import pandas

def get_cities():
    # df = pandas.read_csv('kz.csv')
    df = pandas.read_csv('main_cities.csv')

    columns = [
        'city',
        'lat',
        'lng'
    ]

    city_id = 0
    cities = []

    while True:
        city = {
            'name': '',
            'lat': None,
            'lng': None,
        }
        city['name'] = df[columns[0]][city_id]
        city['lat'] = df[columns[1]][city_id]
        city['lng'] = df[columns[2]][city_id]

        city_id += 1
        if( city_id >= len (df[columns[0]] ) ):
            break

        cities.append(city)

    return cities
```

⊗ 0 ⚠ 0

MST.py | main.py | cities.py M

main.py > ...

```python
import distances
import cities
import MST

distances = distances.get_distances()
cities = cities.get_cities()

vertices = len(distances)

graph = MST.Graph(vertices)

for city, endpoints in distances.items():
    for city2 in endpoints:
        graph.add_edge(city2['start_id'], city2['end_id'], city2['distance'])

result, cost = graph.MST()

print ("Edges of MST")
for u, v, weight in result:
    print("%s <-> %s = ~%d km" % (cities[u]['name'], cities[v]['name'], weight))

print("Total Cost => " , cost)
```

Help

MST.py M | main.py | cities.py M

MST.py > Graph

```
1   class Graph:
2
3       def __init__(self, vertices):
4           self.Vertices = vertices
5           self.graph = []
6
7       def add_edge(self, a, b, w):
8           self.graph.append([a, b, w])
9
10  >    def MST(self):⋯
42
```

```python
def MST(self):

    result = []

    n = self.Vertices
    m = len(self.graph)

    # sort edges in ascending order by weight
    self.graph = sorted(self.graph,
                        key=lambda data: data[2])

    tree_id = []
    for i in range(n):
        tree_id.append(i)

    cost = 0          m: int
    for i in range(m):

        a, b, w = self.graph[i]

        if (tree_id[a] != tree_id[b]):
            cost += w
            result.append ( [a, b, w] )

            old_id = tree_id[b]
            new_id = tree_id[a]

            for j in range(n):
                if (tree_id[j] == old_id):
                    tree_id[j] = new_id

    return [result, cost]
```

MST.py M | main.py | cities.py M | distances.py M

distances.py > get_distances

```python
from geopy.distance import geodesic
import cities
def get_distances():
    Cities = cities.get_cities()
    distances = {}
    city_id = 0
    for city in Cities:
        city_name = city['name']
        distances[city_name] = []
        city2_id = 0
        for city2 in Cities:
            coords_1 = (city['lat'], city['lng'])
            coords_2 = (city2['lat'], city2['lng'])
            distance = geodesic(coords_1, coords_2).km

            if( distance == 0 ):
                city2_id += 1
                continue

            distances[city_name].append(
                {
                    'start_id': city_id,
                    'end_id': city2_id,
                    'end_name' : city2['name'],
                    'distance' : distance
                }
            )
            city2_id += 1
        city_id += 1
    return distances
```