Travel assistant:

Complex itinerary generation

Galan Radu

I. Implementation

First of all, I have used Python for the data processing, data mining and artificial intelligence algorithms. Libraries like numpy and pandas always simplify the process of collecting and preprocessing the data.

I have implemented the project around a few main technologies and then developed them by respecting the structural architectural principles:

-Domain:

```python
class Location:
    def __init__(self, name, latitude, longitude, country=None, city=None, street=None, schedule=None):
        """
        Creates a location
        :param name: Desired name for location
        :param latitude: Latitude of location
        :param longitude: Longitude of location
        :param country: Country of location
        :param city: City of location
        :param street: Street of location
        :param schedule: A dictionary with keys being days of the week (like "Wednesday") and values being a list of hours
        """
        self.__name = name
        self.__longitude = longitude
        self.__latitude = latitude
        self.__country = country or ""
        self.__city = city or ""
        self.__street = street or ""
        self.__schedule = schedule
```

```python
class ObjectiveVisit:
    def __init__(self, location: Location, staying_time, priority, no_of_reviews, tripadvisor_link, descriptio
        self.description = description
        self.tripadvisor_link = tripadvisor_link
        self.no_of_reviews = no_of_reviews
        self.location = location
        self.staying_time = staying_time
        self.priority = priority
        self.dayNo=0
```

The Location and the Objective Visit contains all the important data that our project will use. The data will have been already prepared at the first run time.

There were several methods used in order to obtain the desired data (essential : location, name, coordinates and optional: reviews, schedule, etc.). Initially I have used a WebCrawler for python scrapping on TripAdvisor, but due to the fact that the only coordinates on an Objective webpage were dynamically generated, a static WebCrawler was not enough. I have experimented with the popular UiPath, but the run time was too big, so I have found a relatively optimal solution (because using the

API wasn't one of the options) using another library that takes longer to load, but also loads all the dynamically generated elements.

With all the above-seen elements at hand the project is free to continue with the most important (and apparently the hardest) phase finished.

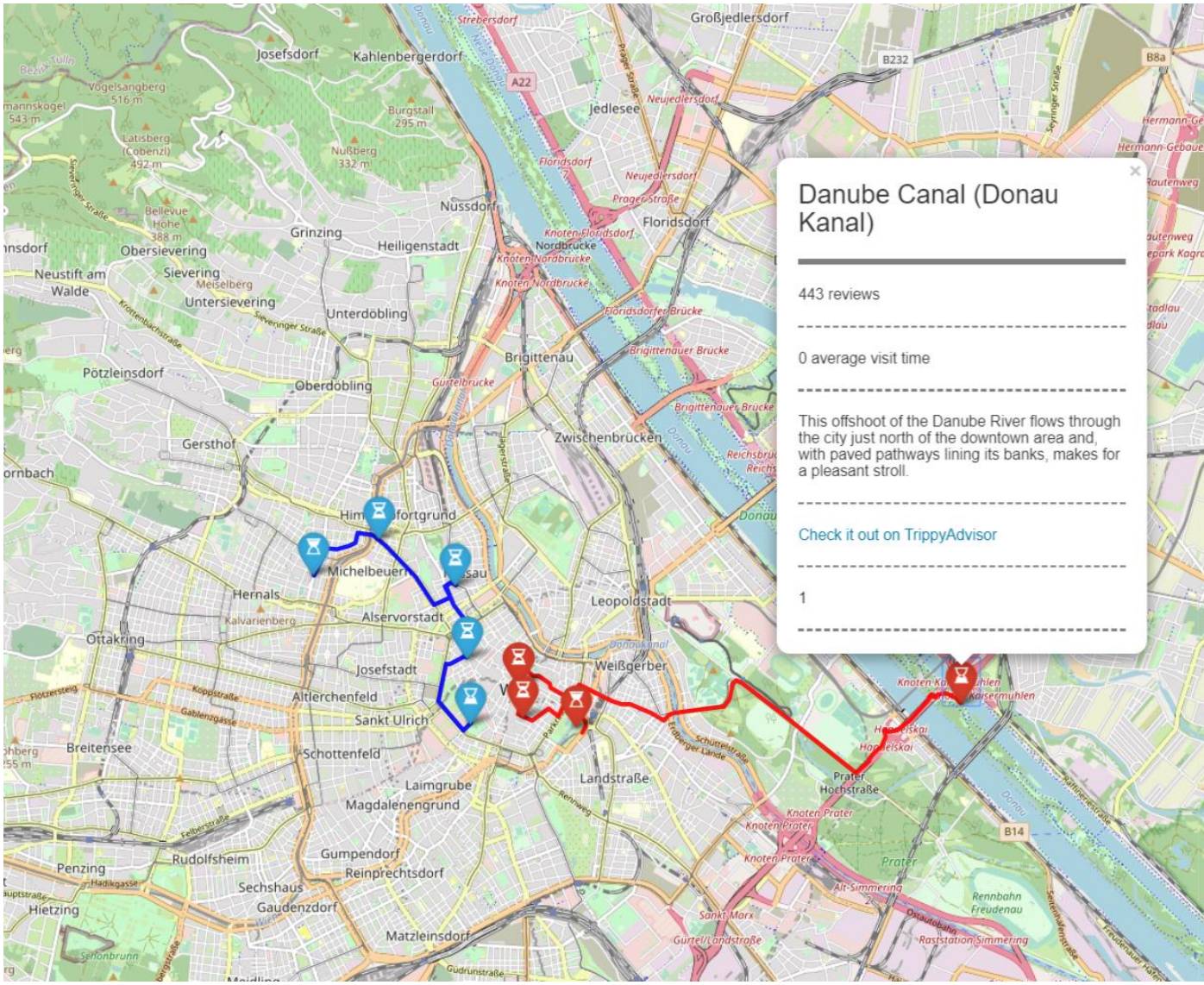
As I previously mentioned that I have structured my project, and the architecture is a classical one:
-service: does all the Business methods
-repository: does all the reading and saving in the file (in order to spare some api calls or simply retrieve information)
-domain: where I am keeping the fore-mentioned data structures
-data base: where I am keeping the actual collected data as txt and csv files
-Kmeans: is one of the libraries that I had to manually install in order to get the functions to work
-output: where I am saving all the exported data, for example the itinerary and the geosjon data about the current trip (saved individually for each day of the trip) and the html file generate by .folium to be able to view the data.

Technologies Used:
-OpenRouteService : has been used in order to get the answer to the tsp problem taking into account the given parameters. I have taken advantage of two API calls: one of them provides exact directions to take while traveling between the given location during one of the certain days, and another call is dealing with returning a GeoJson file that points the coordinates of the exact route between the locations.

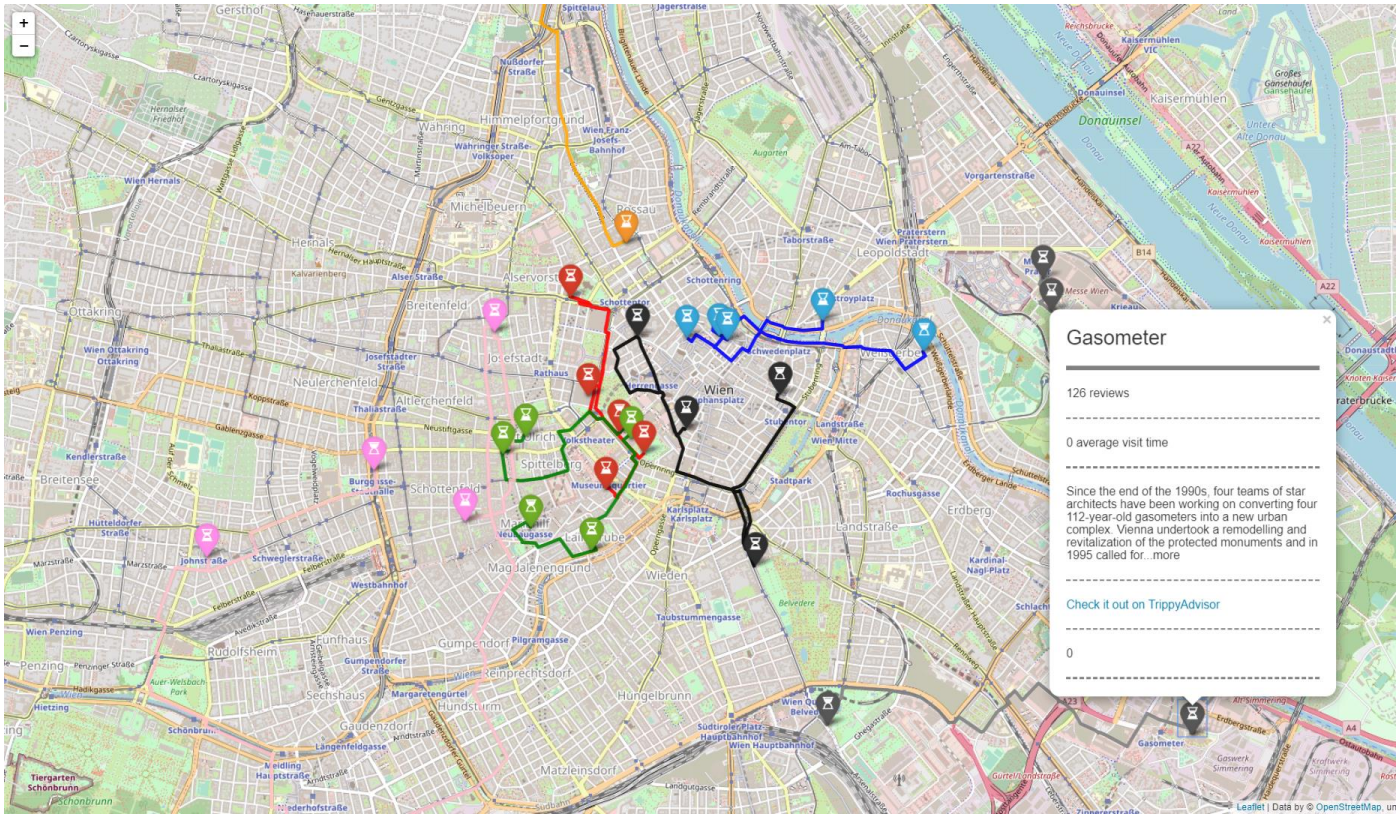
-Folium : python library used to present the graphical results. I have used it to mark out the touristic objectives individually and add them an html code that provided certain details about the location. I have also used it in order to print out all the GeoJson data generated by the OpenRouteServcie.
-Same-Size-K-Means-master: a library created by a fellow programmer, fan of the scikit-learn library. This library offers a wrapper for the well-known K-means algorithm and I have used it to generate to best daily routes. The difference between the classical algorithm is that with a couple more computations it manages to balance out the clusters into almost equal-sized clusters.

II. Experimental results

From only a 2-day trip with 10 objectives:



To a full week trip with 30 objectives:




```
[ : Kilengi Galerie of African Art : 48.237331 : 16.337669 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': [
[ : Cafe Drechsler : 48.198513 : 16.362549 : {'Monday': ['closed'], 'Tuesday': ['closed'], 'Wednesday': ['closed'], 'Thursday': ['8:00:00', ' 12:00:00'], 'Friday': [
[ : Standesamt Schlesingerplatz : 48.212662 : 16.3466 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} ||
[ : Mekhitarist Monastery of Vienna : 48.205952 : 16.354191 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday':
[ : Standesamt Schlesingerplatz : 48.212662 : 16.3466 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} ||
[ : Schonbrunner Brucke : 48.1861 : 16.31988 : {'Monday': ['closed'], 'Tuesday': ['closed'], 'Wednesday': ['closed'], 'Thursday': ['closed'], 'Friday': ['closed'], '
_______DAY4__________
[ : Denkmal Ludwig Zamenhof : 48.201969 : 16.37705 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 4
[ : Krieau Trabrennpark : 48.20985 : 16.41296 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} ||  || 0 |
[ : Donaucity-Kirche Christus, Hoffnung der Welt : 48.233021 : 16.415251 : {'Monday': ['8:00:00', '18:00:00'], 'Tuesday': ['8:00:00', '18:00:00'], 'Wednesday': ['8:0
[ : Skulptur "Scherzo Gruppe" : 48.198841 : 16.38278 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} ||
[ : Windobona Indoor Skydiving : 48.215721 : 16.40029 : {'Monday': ['closed'], 'Tuesday': ['closed'], 'Wednesday': ['closed'], 'Thursday': ['closed'], 'Friday': ['cl
[ : Hochstrahlbrunnen : 48.19923 : 16.375389 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 74 revie
[ : Backerstrasse No. 2 : 48.208862 : 16.372383 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 2 rev
[ : Kaewmany 2 : 48.19397 : 16.3883 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} ||  || 0 || Local We
[ : Funeral Museum (Bestattungsmuseum) : 48.155041 : 16.442049 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday
_______DAY5__________
[ : Mozart Haus : 48.209862 : 16.369909 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 37 reviews ||
[ : Memorial for the Victims of Nazi Military Justice : 48.20813 : 16.36372 : {'Monday': ['closed'], 'Tuesday': ['closed'], 'Wednesday': ['closed'], 'Thursday': ['cl
[ : Josephinum : 48.219669 : 16.356829 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 36 reviews ||
[ : Wiener Fischereimuseum : 48.282349 : 16.379431 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 2
[ : The Islamic Cultural Center of Bad Voslau : 48.227791 : 16.35914 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [],
[ : WUK : 48.223282 : 16.35141 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 29 reviews || 0 || Loc
[ : Minoritenkirche : 48.209332 : 16.363729 : {'Monday': ['closed'], 'Tuesday': ['closed'], 'Wednesday': ['closed'], 'Thursday': ['closed'], 'Friday': ['closed'], 'S
[ : Villa Wertheimstein : 48.24292 : 16.355659 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 4 revi
[ : Palais Ephrussi : 48.213554 : 16.362446 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 15 review
_______DAY6__________
[ : Pappala Pub : 48.208862 : 16.372383 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 8 reviews ||
[ : Ma Pitom : 48.2117 : 16.374821 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 1 review || 0 || L
[ : Donauweibchenbrunnen : 48.20282 : 16.38158 : {'Monday': ['closed'], 'Tuesday': ['closed'], 'Wednesday': ['closed'], 'Thursday': ['closed'], 'Friday': ['closed'],
[ : Skulptur "Die Treue der osterreichischen Nation" : 48.199429 : 16.37295 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday'
[ : Herkules Brunnen : 48.20504 : 16.36685 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 5 reviews
[ : Freyung Passage : 48.210621 : 16.365271 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 138 revie
[ : Josefsbrunnen : 48.208618 : 16.36973 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 31 reviews |
[ : Palais Schwarzenberg : 48.196869 : 16.37602 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 3 rev
[ : Denkmal Aron Menczer : 48.212219 : 16.37311 : {'Monday': [], 'Tuesday': [], 'Wednesday': [], 'Thursday': [], 'Friday': [], 'Saturday': [], 'Sunday': []} || 4 rev
{'type': 'FeatureCollection', 'features': [{'bbox': [16.342081, 48.200615, 16.370868, 48.218761], 'type': 'Feature', 'properties': {'segments': [{'distance': 1057.6,
{'type': 'FeatureCollection', 'features': [{'bbox': [16.202267, 48.132737, 16.341798, 48.245518], 'type': 'Feature', 'properties': {'segments': [{'distance': 7077.7,
{'type': 'FeatureCollection', 'features': [{'bbox': [16.319896, 48.185909, 16.362591, 48.237331], 'type': 'Feature', 'properties': {'segments': [{'distance': 841.5,
{'type': 'FeatureCollection', 'features': [{'bbox': [16.372369, 48.154759, 16.442107, 48.233011], 'type': 'Feature', 'properties': {'segments': [{'distance': 481.9,
{'type': 'FeatureCollection', 'features': [{'bbox': [16.349501, 48.208111, 16.385965, 48.282581], 'type': 'Feature', 'properties': {'segments': [{'distance': 736.9,
{'type': 'FeatureCollection', 'features': [{'bbox': [16.365099, 48.196812, 16.383392, 48.2123451, 'type': 'Feature', 'properties': {'segments': [{'distance': 297.0,
```

```
[{'distance': 349.3, 'duration': 69.9, 'type': 11, 'instruction': 'Head northeast', 'name': '-', 'way_points': [0, 22]}, {'distance': 183.4, 'duration': 36.7,
[{'distance': 349.3, 'duration': 69.9, 'type': 11, 'instruction': 'Head northeast', 'name': '-', 'way_points': [0, 22]}, {'distance': 183.4, 'duration': 36.7,
[{'distance': 332.1, 'duration': 66.4, 'type': 11, 'instruction': 'Head north on Martinstraße', 'name': 'Martinstraße', 'way_points': [0, 11]}, {'distance': 3
[{'distance': 332.1, 'duration': 66.4, 'type': 11, 'instruction': 'Head north on Martinstraße', 'name': 'Martinstraße', 'way_points': [0, 11]}, {'distance': 3
```

III. Analysis

I have experienced with 2 different API services for the mapping/routing systems and the simplest (and actually the only free one available) was OperRouteService.

One API for the routing system was used (it can receive more than 2 coordinates and create a route), it can generate an itinerary with explicit directions, but it can also calculate and optimize based on the TSP problem really well and fast.

Another API was used for the map representation with basic mapping functionalities like: GPS map generating and positioning, route plotting based on a GeoJson file (as generated by the OpenRouteService) and Marker plotting. The alternative was either a paid subscription to Google, either another library that was able read GeoJson, but the latter was far more difficult to implement using python.

For the clustering of the location I have implemented 4 particular methods:

-random generation (which was obviously for testing the environment)

-DBSCAN (although it is supposed to give much better results, for our case no matter the potential, we needed a fixed number of clusters, results were either completely scattered, either in one single class)

-KMEANS (the algorithm managed to create the good number of clusters, but usually due to the fact that in certain areas there is a greater concentration of objectives, the algorithm ended up generating one predominant cluster that contained the majority of the data)

-KMEANS-same-size(that seemed to be the perfect fit due to the fact that we needed homogeneity for every day of the schedule planning, but also optimization inside every day)

Refferences :

https://readthedocs.org/projects/openrouteservice-py/downloads/pdf/latest/

https://giscience.github.io/openrouteservice-r/articles/openrouteservice.html

https://pynative.com/python-random-choice/

https://giscience.github.io/openrouteservice-r/reference/ors_isochrones.html

https://openrouteservice.org

https://towardsdatascience.com/choropleth-maps-with-folium-1a5b8bcdd392

https://python-visualization.github.io/folium/quickstart.html

https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

https://github.com/ndanielsen/Same-Size-K-Means