MASTER - Notebook 2

Matteo Grazioso 884055

In [ ]:

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.dates as mdates
from datetime import datetime
import json
import warnings
warnings.filterwarnings('ignore')

import myfunctions as mf # Custom functions
```

In [ ]:

```
# Disply all columns and all rows
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

In [ ]:

```
# The file contains the data of the validation of tickets in the city of public transport of Venice.
# The file has been created by the Notebook 1.ipynb

# Import the data into a dataframe of a txt file
path = 'data/processed/dataset_cleaned_validazioni.txt'
# path = 'data/processed/dataset_cleaned_esportazioneCompleta.txt'

df = pd.read_csv(path, header=0, sep='\t')

# Save the name of the file in a variable for future use extracting the name of the file from the path
file_name = path.split('_')[-1].split('.')[0]

# Display the first 5 rows of the dataframe
df.head()

# Convert the column 'DATA' to datetime format
df['DATA'] = pd.to_datetime(df['DATA'], format='%Y-%m-%d')
```

In [ ]:

```
# First 10% of the dataframe
# df = df.iloc[:int(len(df)*0.1)]
```

In [ ]:

```
df.head()
```

Out[ ]:

	DATA	ORA	DATA_VALIDAZIONE	SERIALE	FERMATA	DESCRIZIONE	TITOLO	TICKET_CODE	DESCRIZIONE_TITOLO
0	2022-05-13	00:00:00	2022-05-13 00:00:00	65676291870913797	5089	FERROVIA	11149	4	7GG-TPL 43,60-COMVE16,40
1	2022-05-13	00:00:00	2022-05-13 00:00:00	36141384536591364	5032	FERROVIA	11107	2	48H-TPL 24,90-COMVE5,10
2	2022-05-13	00:00:00	2022-05-13 00:00:00	36144856606063108	5031	P.LE ROMA	11108	3	72H-TPL 33,40-COMVE6,60
3	2022-05-13	00:00:00	2022-05-13 00:00:00	36144856474364932	506	VENEZIA	11261	1	DAILY PASS VENEZIA - AVM
4	2022-05-13	00:00:00	2022-05-13 00:00:00	36144856606062852	5031	P.LE ROMA	11108	3	72H-TPL 33,40-COMVE6,60

In [ ]:

```python
df.tail()
```

Out[ ]:

	DATA	ORA	DATA_VALIDAZIONE	SERIALE	FERMATA	DESCRIZIONE	TITOLO	TICKET_CODE	DESCRIZIONE_TITOLO
4427556	2022-07-15	02:27:00	2022-07-15 02:27:00	37271982183271940	4525	SANTA MARIA	11261	1	DAILY PASS VENEZIA - AVM
4427557	2022-07-15	02:27:00	2022-07-15 02:27:00	37271982183274756	4525	SANTA MARIA	11261	1	DAILY PASS VENEZIA - AVM
4427558	2022-07-15	04:33:00	2022-07-15 04:33:00	36088514819663876	5030	P.LE ROMA	5	7	75'-TPL 6,64-COMVE0,86
4427559	2022-07-15	05:06:00	2022-07-15 05:06:00	40832955551087108	509	VENEZIA	12101	7	BIGL.AUT.75'MESTRE/LIDO-TSC
4427560	2022-07-15	05:13:00	2022-07-15 05:13:00	40832947760207876	509	VENEZIA	12101	7	BIGL.AUT.75'MESTRE/LIDO-TSC

Trajectories

We are interested in analyzing the trajectories of the users that use the public transport in the city of Venice.

Note that the stops are identified by the *DESCRIZIONE* column that contains the name of the stop, so each trajectory is composed by the name of the stops visited by the user, so a trajectory is composed by a sequence of strings.

In [ ]:

```python
def create_dictionary_with_trajectories(df: pd.DataFrame) -> dict[str, datetime]:
    """
        This function creates a dictionary with the trajectories of the users.
        :param df: the dataframe
        :return: the dictionary with the trajectories of the users with the key (serial, day).
            Notice: if the serial is the same for all the days the key is (serial, None)
    """
    # Create a dictionary with the trajectories of the users
    dict_trajectories = {}
    # For each user
    for serial in df['SERIALE'].unique():
        # NOTICE: the serial of ticket code 5, 5-STUD, 5-RET, 5-WKRS and the same with 6, change every day,
        # so the user is not the same, but the user is the same for the same day

        # Select only the rows of the specified user
        df_sup = df[df['SERIALE'] == serial].sort_values(by=['DATA', 'ORA'])

        # Create a list with the trajectories of the user
        list_trajectories = []

        # If the ticket code is 5, 5-STUD, 5-RET, 5-WKRS or 6, 6-STUD, 6-RET, 6-WKRS create a list with the trajectories
        # for each day and for each serial
        if df_sup['TICKET_CODE'].unique()[0] in ['5', '5-STUD', '5-RET', '5-WKRS', '6', '6-STUD', '6-RET', '6-WKRS']:
            # print('Ticket code: {}'.format(df_sup['TICKET_CODE'].unique()[0]))
            # For each day
            for day in df_sup['DATA']:
                # Select only the rows of the specified day
                df_sup_sup = df_sup[df_sup['DATA'] == day]
                # For each serial
                for serial in df_sup_sup['SERIALE'].unique():
                    # Reset list_trajectories
                    list_trajectories = []
                    # Select only the rows of the specified serial
                    df_sup_sup_sup = df_sup_sup[df_sup_sup['SERIALE'] == serial]
                    # Create a list with the trajectories of the user
                    list_stop = df_sup_sup_sup['DESCRIZIONE'].tolist()
                    # Append the list to the list of the trajectories
                    list_trajectories.append(list_stop)
                    # Insert the list of the trajectories in the dictionary with the key (serial, day)
                    dict_trajectories[serial, day] = list_trajectories
        else:
            # Create a list with the trajectories of the user
            list_stop = df_sup['DESCRIZIONE'].tolist()
            list_trajectories.append(list_stop)
            # If the serial is already in the dictionary print a warning
            if (serial, None) in dict_trajectories:
                print('Warning: the serial {} is already in the dictionary'.format(serial))
                # launch an exception
                raise Exception('The serial {} is already in the dictionary'.format(serial))
            # Insert the list of the trajectories in the dictionary with the key (serial, None)
            # None means that the serial is the same for all the days
            dict_trajectories[serial, None] = list_trajectories

    return dict_trajectories
```

In [ ]:

```
def average_length_of_trajectories(dict_trajectories: dict, is_focus_on_ticket_code: bool = False, ticket_code: str = '') -> float:
    """
        This function computes the average length of the trajectories.
        This function can receive a dictionary with the trajectories related to a specific ticket code: in this case the function prints this information.
        :param dict_trajectories: the dictionary with the trajectories of the users with the key (serial, day) or (serial, None)
        :return: the average length of the trajectories
    """
    # Notice that the key of the dictionary is a tuple (serial, day)
    # Compute the average length of the trajectories
    average_length = 0
    for chiave in dict_trajectories.keys():
        if chiave[1] is not None:
            average_length += len(dict_trajectories[chiave][0])
        else:
            average_length += len(dict_trajectories[chiave][0])
    if len(dict_trajectories.keys()) != 0:
        average_length /= len(dict_trajectories.keys())
        if is_focus_on_ticket_code:
            # Round the average length to 4 decimal places
            average_length = round(average_length, 4)
            print('The average length of the trajectories with the ticket code {} is: {}'.format(ticket_code, average_length))
        else:
            print('The average length of the trajectories is: {}'.format(average_length))
    else:
        print('WARNING: There are no trajectories to analyze')
    # Print also the number of users and the number of trajectories
    print('The number of trajectories is: {}'.format(len(dict_trajectories.keys())))

    return average_length
```

In [ ]:

```
def average_lenght_of_trajectories_per_ticket_code_stmp(df: pd.DataFrame):
    """
        This function computes the average length of the trajectories for each ticket code in the dataset.
        The order of the ticket codes is mantained because the ticket codes are stored in the dictionary "dict_ticket_codes.json".
        :param df: the dataframe
        :return: None
    """
    dict_trajectories = {}
    dict_ticket_codes = mf.open_dict_ticket_codes()

    # For each ticket code in the dictionary
    for ticket_code in dict_ticket_codes.keys():
        # Select only the rows of the specified ticket code
        df_sup = mf.focus_on_ticket_code(df, ticket_code)

        # If the dataframe is not empty, focus on the ticket code
        if df_sup.shape[0] != 0:
            # Create a dictionary with the trajectories of the users
            dict_trajectories[ticket_code] = create_dictionary_with_trajectories(df_sup)
            average_length_of_trajectories(dict_trajectories[ticket_code], is_focus_on_ticket_code=True, ticket_code=ticket_code)
        else:
            # If the dataframe is empty, skip the ticket code but launch a warning
            print('WARNING: There are no validations of the ticket code "{}"'.format(ticket_code))
```

In [ ]:

```
def average_length_of_trajectories_by_ticket_code_plot(dict_trajectories: dict, df: pd.DataFrame) -> None:
    """
        This function computes the average length of the trajectories by ticket code.
        :param dict_trajectories: the dictionary with the trajectories of the users
        :param df: the dataframe
        :return: None
    """
    # Compute the average length of the trajectories:
            # There are no colums with the coordinates of the stations, and there are no columns with the distance between the stations
            # So, I compute the average length of the trajectories by the number of stations visited

    # Open the dictionary
    dict_ticket_code = mf.open_dict_ticket_codes()

    # Create a dictionary with the number of stations visited for each ticket code
    dict_number_of_stations = {}
    for ticket_code in dict_ticket_code.keys():
        # Select only the rows of the specified ticket code
        df_sup = mf.focus_on_ticket_code(df, ticket_code)
        # If the dataframe is not empty, focus on the ticket code
        if df_sup.shape[0] != 0:
            # Create a dictionary with the trajectories of the users
            dict_trajectories[ticket_code] = create_dictionary_with_trajectories(df_sup)
            # Count the number of stations visited
            number_of_stations = 0
            for chiave in dict_trajectories[ticket_code].keys():
                if chiave[1] is not None:
                    number_of_stations += len(dict_trajectories[ticket_code][chiave][0])
                else:
                    number_of_stations += len(dict_trajectories[ticket_code][chiave][0])
            # Add the average length of the trajectories to the dictionary
            dict_number_of_stations[ticket_code] = number_of_stations / len(dict_trajectories[ticket_code].keys())

    # Plot the average length of the trajectories by ticket code
    plt.figure(figsize=(20, 10))
    plt.bar(dict_number_of_stations.keys(), dict_number_of_stations.values())
    plt.title('Average length of the trajectories by ticket code')
    plt.xlabel('Ticket code')
    plt.ylabel('Average length of the trajectories')

    # Manage the x-axis adding the description of the ticket code; note that it is possible that some ticket codes are not present in the plot
    plt.xticks(ticks=range(0, len(dict_number_of_stations.keys())), labels=[dict_ticket_code[ticket_code] for ticket_code in dict_number_of_stations.keys()], rotation=90)

    # Manage the y-axis: note that the y-axis are float numbers: do not convert them to integers and do not use the range function
    if max(dict_number_of_stations.values()) < 0.1:
        plt.yticks(ticks=np.arange(0, max(dict_number_of_stations.values()) + 1, 0.01))
    else:
        plt.yticks(ticks=np.arange(0, max(dict_number_of_stations.values()) + 1, 0.5))

    # Add the value of each bar
    for index, value in enumerate(dict_number_of_stations.values()):
        plt.text(index, value, str(round(value, 4)), ha='center', va='bottom', fontsize=10)

    plt.show()

    # Average number of stations visited by trajectory by ticket code
    # for ticket_code in dict_number_of_stations.keys(
        # print('The average number of stations visited by trajectory with ticket code "{}" is: {}'.format(ticket_code, dict_number_of_stations[ticket_code]))
```

```
In [ ]:
def most_frequent_trajectories(dict_trajectories: dict, is_focus_on_ticket_code: bool = False, ticket_code: str = '', summary: bool = True) -> None:
    """
        This function finds the most frequent trajectories.
        :param dict_trajectories: the dictionary with the trajectories of the users
        :param is_focus_on_ticket_code: True if the analysis is focused on a specific ticket code, False otherwise
        :param ticket_code: the ticket code
        :param summary: True if the summary of the most frequent trajectories is printed, False otherwise
        :return: None
    """
    # Find the most frequent trajectories
    # Create a dictionary with the number of times that a trajectory is present
    dict_trajectories_number = {}

    # For each user (identified by the serial number)
    for serial in dict_trajectories.keys():
        # Convert the list of tuples in a tuple of tuples
        trajectory = tuple(dict_trajectories[serial])

        if trajectory in dict_trajectories_number.keys():
            # If the trajectory is already present in the dictionary,
            # increase the number of times that the trajectory is present
            dict_trajectories_number[trajectory] += 1
        else:
            # Otherwise, add the trajectory to the dictionary
            dict_trajectories_number[trajectory] = 1

    # Sort the dictionary by the number of times that a trajectory is present
    dict_trajectories_number = {k: v for k, v in sorted(dict_trajectories_number.items(), key=lambda item: item[1], reverse=True)}

    if (summary):
        # Print the most frequent trajectories
        if is_focus_on_ticket_code:
            print('The most frequent trajectories with the ticket code {} are:'.format(ticket_code))
        else:
            print('The most frequent trajectories are:')
        for trajectory in list(dict_trajectories_number.keys())[:10]:
            print('The trajectory {} is present {} times'.format(trajectory, dict_trajectories_number[trajectory]))
    else:
        # Return the most frequent trajectories and the number of times that they are present,
        # ordered by the number of times that they are present
        return dict_trajectories_number
```

In [ ]:

```
def trajectories_with_at_least_k_stations(dict_trajectories: dict, k: int, summary: bool = True):
    """
        This function finds the trajectories with at least k stations visited.
        :param dict_trajectories: the dictionary with the trajectories of the users
        :param k: the number of stations
        :return: None if summary is True, the dictionary with the trajectories with at least k stations visited otherwise
    """
    # Find the trajectories with at least k stations visited
    # Create a dictionary with the number of times that a trajectory is present
    # Notice that the keys of the dictionary are list of [serial, day]. Day can be None if the serial does not change over the days
    dict_trajectories_number = {}
    for serial in dict_trajectories.keys():
        # Convert the list of tuples in a tuple of tuples
        trajectory = tuple(dict_trajectories[serial])

        if trajectory in dict_trajectories_number.keys():
            # If the trajectory is already present in the dictionary,
            # increase the number of times that the trajectory is present
            dict_trajectories_number[trajectory] += 1
        else:
            # Otherwise, add the trajectory to the dictionary
            dict_trajectories_number[trajectory] = 1

    # Sort the dictionary by the number of times that a trajectory is present
    dict_trajectories_number = {k: v for k, v in sorted(dict_trajectories_number.items(), key=lambda item: item[1], reverse=True)}

    # If summary is True, print the trajectories with at least k stations visited, return the len of the dictionary otherwise
    if summary:
        # Print the trajectories with at least k stations visited
        print('The trajectories with at least {} stations visited are:'.format(k))
        for trajectory in dict_trajectories_number.keys():
            if len(trajectory) >= k:
                print('The trajectory {} is present {} times'.format(trajectory, dict_trajectories_number[trajectory]))
    else:
        # Return the dictionary
        return dict_trajectories_number
```

In [ ]:

```
def longest_common_subsequence(trajectory_1: list, trajectory_2: list) -> list:
    """
        This function finds the Longest Common Subsequence (LCS) between two trajectories.
        :param trajectory_1: the first trajectory
        :param trajectory_2: the second trajectory
        :return: the LCS
    """
    # Find the LCS
    # Create a matrix with the length of the LCS between two trajectories
    matrix = [[0 for x in range(len(trajectory_2) + 1)] for y in range(len(trajectory_1) + 1)]
    for i in range(1, len(trajectory_1) + 1):
        for j in range(1, len(trajectory_2) + 1):
            if trajectory_1[i - 1] == trajectory_2[j - 1]:
                matrix[i][j] = matrix[i - 1][j - 1] + 1
            else:
                matrix[i][j] = max(matrix[i - 1][j], matrix[i][j - 1])

    # Find the LCS
    i = len(trajectory_1)
    j = len(trajectory_2)
    lcs = []
    while i > 0 and j > 0:
        if trajectory_1[i - 1] == trajectory_2[j - 1]:
            lcs.append(trajectory_1[i - 1])
            i -= 1
            j -= 1
        elif matrix[i - 1][j] > matrix[i][j - 1]:
            i -= 1
        else:
            j -= 1

    # Reverse the LCS
    lcs.reverse()

    return lcs
```

Compute the average length of the trajectories with the entire dataset

In [ ]:

```
# Create a dictionary with the trajectories of the users
dict_trajectories = create_dictionary_with_trajectories(df)
```

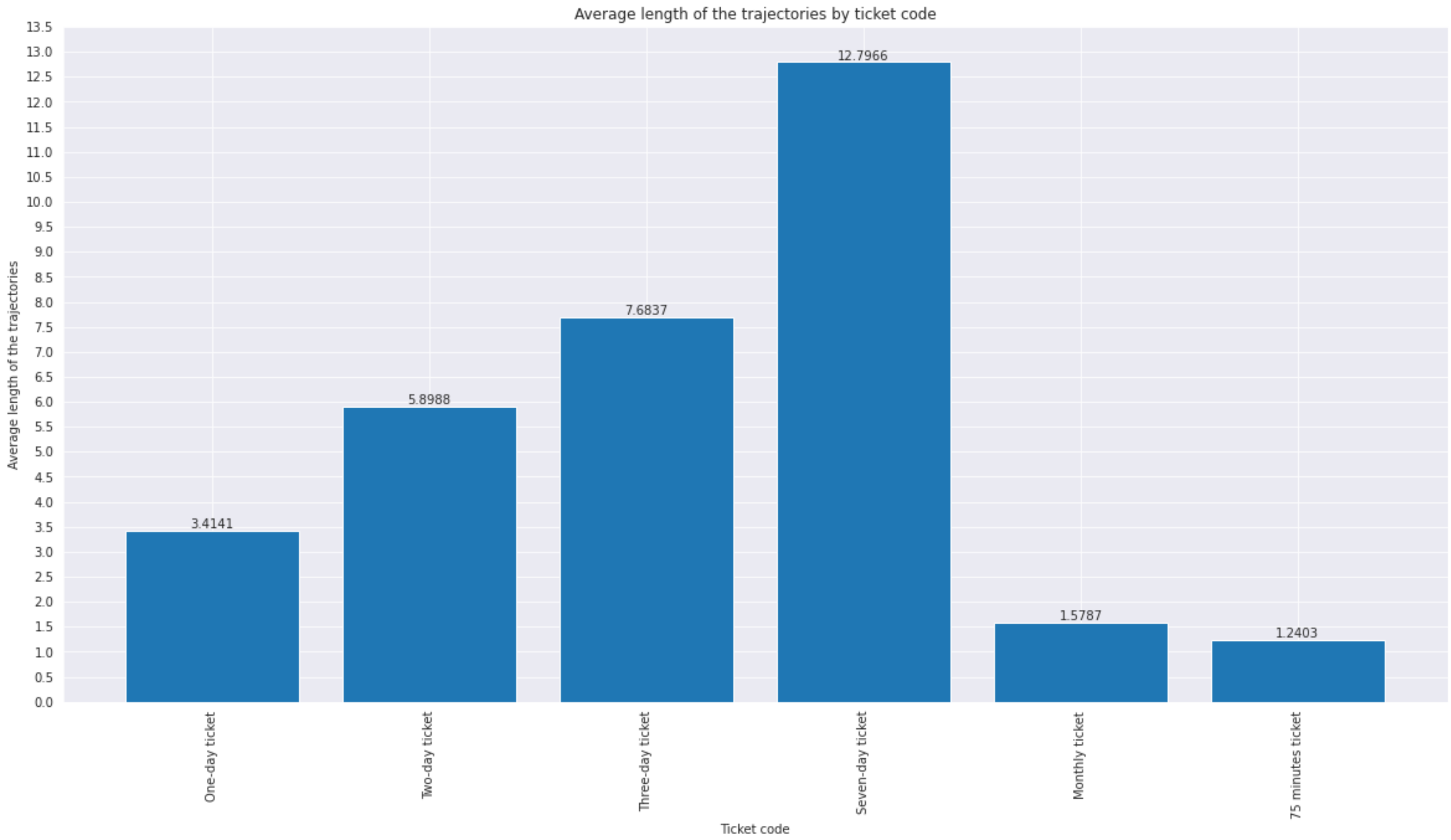
In [ ]:

```
# Print all the trajectories with the second field of the key equal to None
# for trajectory in dict_trajectories.keys():
    # if trajectory[1] != None:
        # print(trajectory)
```

In [ ]:

```
# Compute the average length of the trajectories
print('The average length of the trajectories considering all the dataframe:')
average_lenght = average_length_of_trajectories(dict_trajectories)
# Compute the average length of the trajectories by ticket code
# average_lenght_of_trajectories_per_ticket_code_stmp(df)
# Plot the average length of the trajectories by ticket code
# average_length_of_trajectories_by_ticket_code_plot(dict_trajectories, df, 'data/dictionaries/dict_ticket_codes.json')
average_length_of_trajectories_by_ticket_code_plot(dict_trajectories, df)
```

```
The average length of the trajectories considering all the dataframe:
The average length of the trajectories is: 2.5301490753011135
The number of trajectories is: 1749921
```



Compute the average length of the trajectories with the dataset with the ticket code 5

```
In [ ]: # Create a dictionary with the trajectories of the users with the ticket code '7'
        df_sup = mf.focus_on_ticket_code(df,'7')
        # Create a dictionary with the trajectories of the users
        dict_trajectories = create_dictionary_with_trajectories(df_sup)
        # Compute the average length of the trajectories
        average_lenght = average_length_of_trajectories(dict_trajectories, is_focus_on_ticket_code=True, ticket_code='7')
        # average_length_of_trajectories_by_ticket_code_plot(dict_trajectories, df_sup, 'data/dictionaries/dict_ticket_codes.json')
        # trajectories_with_at_least_k_stations(dict_trajectories, 3)
```

```
The average length of the trajectories with the ticket code 7 is: 1.2403
The number of trajectories is: 1200462
```

```
In [ ]: # For each ticket code (dict_ticket_codes.keys()):
        # a. Create a dictionary with the trajectories of the users with the ticket code
```

```python
# b. Compute the average length of the trajectories (number of stations visited)
# d. Compute the number of users
# e. Compute the number of trajectories
# c. Number of trajectories with at least k stations visited
# d. Find the most frequent trajectories
# e. Find the trajectories with at least k stations visited

ticket_code = mf.open_dict_ticket_codes()

# dict_trajectories = create_dictionary_with_trajectories(df)
# average_length_of_trajectories_by_ticket_code_plot(dict_trajectories, df, 'data/dictionaries/dict_ticket_codes.json')

for ticket_code in ticket_code.keys():
    # Exclude the ticket code '8'
    if ticket_code == '8':
        continue

    print('Ticket code: {} - "{}"'.format(ticket_code, mf.get_ticket_code_description(ticket_code)))

    # a. Create a dictionary with the trajectories of the users with the ticket code
    df_sup = mf.focus_on_ticket_code(df, ticket_code)
    dict_trajectories = create_dictionary_with_trajectories(df_sup)
    if len(dict_trajectories.keys()) == 0:
        print('\t No trajectories to analyze')
        print('')
        continue

    # b. Compute the average length of the trajectories (number of stations visited)
    average_length_of_trajectories(dict_trajectories, is_focus_on_ticket_code=True, ticket_code=ticket_code)

    # c. Compute the number of users (Seriale)
    number_of_users = len(dict_trajectories.keys()) # TODO: r u sure?

    # d. Compute the number of trajectories
    number_of_trajectories = 0
    for trajectory in dict_trajectories.values():
        number_of_trajectories += len(trajectory)
    print('The number of trajectories is: {}'.format(number_of_trajectories))

    # e. Number of trajectories with at least k stations visited
    # number_of_trj_k_stations =  trajectories_with_at_least_k_stations(dict_trajectories, k=3, summary=False)
    # # Count the number of trajectories with at least k stations visited
    # count = 0
    # for trajectory in number_of_trj_k_stations.keys():
    #     if len(trajectory) >= 3:
    #         count += 1
    # print('The number of trajectories with at least 3 stations visited is: {}'.format(count))

    # f. Find the most frequent trajectories
    # most_frequent_trajectories(dict_trajectories, is_focus_on_ticket_code=True, ticket_code=ticket_code)
    print('')
```

```
Ticket code: 1 - "One-day ticket"
The average length of the trajectories with the ticket code 1 is: 3.4141
The number of trajectories is: 297767
The number of trajectories is: 297767

Ticket code: 2 - "Two-day ticket"
The average length of the trajectories with the ticket code 2 is: 5.8988
The number of trajectories is: 107261
The number of trajectories is: 107261

Ticket code: 3 - "Three-day ticket"
The average length of the trajectories with the ticket code 3 is: 7.6837
The number of trajectories is: 109037
The number of trajectories is: 109037

Ticket code: 4 - "Seven-day ticket"
The average length of the trajectories with the ticket code 4 is: 12.7966
The number of trajectories is: 35170
The number of trajectories is: 35170

Ticket code: 5 - "Monthly ticket"
The average length of the trajectories with the ticket code 5 is: 1.5787
The number of trajectories is: 921
The number of trajectories is: 921

Ticket code: 5-STUD - "Monthly ticket for students"
         No trajectories to analyze

Ticket code: 5-RET - "Monthly ticket for retired"
         No trajectories to analyze

Ticket code: 5-WKRS - "Monthly ticket for workers"
         No trajectories to analyze

Ticket code: 6 - "Annual ticket"
         No trajectories to analyze

Ticket code: 6-STUD - "Annual ticket for students"
         No trajectories to analyze

Ticket code: 6-RET - "Annual ticket for retired"
         No trajectories to analyze

Ticket code: 6-WKRS - "Annual ticket for workers"
         No trajectories to analyze

Ticket code: 7 - "75 minutes ticket"
The average length of the trajectories with the ticket code 7 is: 1.2403
The number of trajectories is: 1200462
The number of trajectories is: 1200462
```

In [ ]:

```python
def print_trajectories_of_user(dict_trajectories: dict, user: str):
    """
        This function prints the trajectories of a user.
        :param dict_trajectories: the dictionary with the trajectories
        :param user: the user
    """
    # print the dataframes of the trajectories of the user
    df_aux = df[df['SERIALE'] == user]
    print(df_aux)
```

In [ ]:

```
# Given the serial of a user, print the trajectories of the user
# Serial of the user of the first trajectory
serial = list(dict_trajectories.keys())[0][0]
# Print the trajectories of the user
print_trajectories_of_user(dict_trajectories, serial)
```

```
         DATA       ORA      DATA_VALIDAZIONE            SERIALE  FERMATA  \
15 2022-05-13  00:01:00  2022-05-13 00:01:00  37270756644007428     5132

     DESCRIZIONE  TITOLO TICKET_CODE      DESCRIZIONE_TITOLO
15  S. MARCUOLA-   11101           7  75'-TPL 6,64-COMVE0,86
```

Find the most frequent trajectories with the entire dataset

In [ ]:

```
# # Create a dictionary with the trajectories of the users with the entire dataset
# dict_trajectories = create_dictionary_with_trajectories(df)
# # Find the most frequent trajectories and print the summary
# most_frequent_trajectories(dict_trajectories, summary=True)
```

Find the most frequent trajectories with the dataset with a specified ticket code

In [ ]:

```
# # Create a dictionary with the trajectories of the users with the ticket code '1'
# df_sup = mf.focus_on_ticket_code(df,'1')
# # Create a dictionary with the trajectories of the users with the ticket code '1'
# dict_trajectories = create_dictionary_with_trajectories(df_sup)
# # Find the most frequent trajectories with the ticket code '1' and print the summary
# most_frequent_trajectories(dict_trajectories, is_focus_on_ticket_code=True, ticket_code='1', summary=True)
```

Finds the trajectories with at least k stations visited with the entire dataset

In [ ]:

```
# # Create a dictionary with the trajectories of the users with the entire dataset
# dict_trajectories = create_dictionary_with_trajectories(df)
# # Find the trajectories with at least 6 stations visited
# trajectories_with_at_least_k_stations(dict_trajectories, 6)
```

Find the most frequent trajectories with the dataset with a specified ticket code

In [ ]:

```
# # Create a dictionary with the trajectories of the users with the ticket code '1'
# df_sup = mf.focus_on_ticket_code(df,'1')
# # Create a dictionary with the trajectories of the users with the ticket code '1'
# dict_trajectories = create_dictionary_with_trajectories(df_sup)
# # Find the trajectories with at least 20 stations visited with the ticket code '1'
# trajectories_with_at_least_k_stations(dict_trajectories, 4)
```

Longest Common Subsequence (LCS)

In [ ]:

```
# # Find the Longest Common Subsequence (LCS) between two trajectories
# trajectory_1 = ['SAN MARCO', 'P.LE ROMA', 'RIALTO', 'PUNTA SABBIO', 'BURANO']
# trajectory_2 = ['SAN MARCO', 'P.LE ROMA', 'PUNTA SABBIO', 'BURANO']

# lcs = longest_common_subsequence(trajectory_1, trajectory_2)
# print('The Longest Common Subsequence (LCS) between the trajectories {} and {} is {}'.format(trajectory_1, trajectory_2, lcs))
```

In [ ]:

```
# # Find the Longest Common Subsequence (LCS) between two trajectories
# # Trajectory 1 is the trajectory of the dataset with the ticket code 1
# # Trajectory 2 is the most frequent trajectory of the dataset with the ticket code 2
# df_sup = mf.focus_on_ticket_code(df,'1')
# dict_trajectories = create_dictionary_with_trajectories(df_sup)
# dict_trajectories_number = most_frequent_trajectories(dict_trajectories, is_focus_on_ticket_code=True, ticket_code='1', summary=False)
# trajectory_1 = list(dict_trajectories_number.keys())[8]

# df_sup = mf.focus_on_ticket_code(df,'2')
# dict_trajectories = create_dictionary_with_trajectories(df_sup)
# dict_trajectories_number = most_frequent_trajectories(dict_trajectories, is_focus_on_ticket_code=True, ticket_code='2', summary=False)
# trajectory_2 = list(dict_trajectories_number.keys())[0]

# lcs = longest_common_subsequence(trajectory_1, trajectory_2)
# print('The Longest Common Subsequence (LCS) between the trajectory {} and the trajectory {} is {}'.format(trajectory_1, trajectory_2, lcs))
```

In [ ]:

```
def most_frequent_stations_visited_by_the_users (dict_trajectories: dict, is_focus_on_ticket_code: bool = False, ticket_code: str = "", summary: bool = False) -> dict:
    """
        This function finds the most frequent stations visited by the users.
        :param dict_trajectories: the dictionary with the trajectories of the users
        :param is_focus_on_ticket_code: if True, the function is focused on the ticket code
        :param ticket_code: the ticket code
        :param summary: if True, the function prints the summary
        :return: the dictionary with the most frequent stations visited by the users
    """
    # Create a dictionary with the most frequent stations visited by the users
    dict_stations = {}

    # For each trajectory
    for trajectory in dict_trajectories.keys():
        # For each station in the trajectory
        for station in dict_trajectories[trajectory]:
            # If the station is not in the dictionary, add it
            if station not in dict_stations.keys():
                dict_stations[station] = 1
            # If the station is in the dictionary, increment its value
            else:
                dict_stations[station] += 1

    # Sort the dictionary with the most frequent stations visited by the users
    dict_stations = {k: v for k, v in sorted(dict_stations.items(), key=lambda item: item[1], reverse=True)}

    # Print the summary
    if summary and is_focus_on_ticket_code:
        print('The most frequent stations visited by the users with the ticket code {} are:'.format(ticket_code))
        for station in dict_stations.keys():
            print('The station {} is visited {} times'.format(station, dict_stations[station]))
    elif summary:
        print('The most frequent stations visited by the users are:')
        for station in dict_stations.keys():
            print('The station {} is visited {} times'.format(station, dict_stations[station]))
    return dict_stations
```

In [ ]:

```
def rearrange_dictionary_with_trajectories (dict_trajectories: dict) -> dict:
    """
        This function rearranges the dictionary with the trajectories of the users.
        :param dict_trajectories: the dictionary with the trajectories of the users
        :return: the rearranged dictionary with the trajectories of the users
    """
    # Create the rearranged dictionary with the trajectories of the users
    dict_trajectories_rearranged = {}

    # For each trajectory
    for trajectory in dict_trajectories.keys():
        # For each station in the trajectory
        for station in dict_trajectories[trajectory]:
            # If the station is not in the dictionary, add it
            if station not in dict_trajectories_rearranged.keys():
                dict_trajectories_rearranged[station] = [trajectory]
            # If the station is in the dictionary, append the trajectory
            else:
                dict_trajectories_rearranged[station].append(trajectory)

    return dict_trajectories_rearranged
```

In [ ]:

```
# # Are there typical patterns in the movements of the users?
# # Using different clustring techniques, find the most frequent trajectories and the most frequent stations visited by the users.
# # Create a dictionary with the trajectories of the users with the entire dataset
# dict_trajectories = create_dictionary_with_trajectories(df)
# # Find the most frequent trajectories and print the summary
# most_frequent_trajectories(dict_trajectories, summary=False)
# # Find the most frequent stations visited by the users and print the summary
# most_frequent_stations_visited_by_the_users(dict_trajectories, summary=False)

# # Cluster the users using the K-Means algorithm
# # Create a dictionary with the trajectories of the users with the entire dataset
# # dict_trajectories = create_dictionary_with_trajectories(df)
# # Cluster the users using the K-Means algorithm
# from sklearn.cluster import KMeans

# merged_list = sum([sublist for sublist in list(dict_trajectories.values())], [])

# # Cluster the users using the K-Means algorithm
# kmeans = KMeans(n_clusters=3, random_state=0).fit(np.array(merged_list))
# # Create a dictionary with the clusters
# dict_clusters = {}
# for i in range(len(kmeans.labels_)):
#     if kmeans.labels_[i] not in dict_clusters.keys():
#         dict_clusters[kmeans.labels_[i]] = []
#     dict_clusters[kmeans.labels_[i]].append(list(dict_trajectories.keys())[i])
# # Print the summary
# print('The clusters are:')
# for cluster in dict_clusters.keys():
#     print('The cluster {} contains the following trajectories:'.format(cluster))
#     for trajectory in dict_clusters[cluster]:
#         print(trajectory)
```