Final Report

Project :
Large-scale models and simulation methods

Completed by:

Aderssa Dhia (dhia.aderssa@edu.univ-eiffel.fr),
M'rad Omar (omar.mrad@edu.univ-eiffel.fr),
Bouzid Mayssa (maissa.bouzid@edu.univ-eiffel.fr) ,
Dridi Feryel (feryel.dridi@edu.univ-eiffel.fr)

Supervised by

Mr.Sebastian Hörl

2024/2025

Dedication

We would like to dedicate this report to our esteemed professor Hörl, whose engaging teaching and invaluable guidance have guided our work throughout this project. Your commitment to our academic success and your expertise have been a source of inspiration for us.

Thank you for your unwavering support and encouragement.

Data preparation :

```
df_age_persons = df_census.groupby("age")["weight"].sum().reset_index()
df_age = pd.merge(df_age, df_age_persons, on = "age")

px.bar(df_age, x = "age", y = ["count", "weight"], barmode = "group")
```

```
df_age = df_census.groupby(["age", "csp"])["weight"].sum().reset_index()
df_age = df_age[df_age["csp"].isin([1, 3, 5])]
df_age["csp"] = df_age["csp"].astype(str)

px.bar(df_age, x = "age", y = "weight", color = "csp", barmode = "group")
```

```
df_census["department_id"] = df_census["iris_id"].str[:2]
df_census["municipality_id"] = df_census["iris_id"].str[:5]
```

```
df_census
```

	age	csp	weight	iris_id	department_id	municipality_id
0	68	7	3.324481	010040101	01	01004
1	33	6	2.957422	010040202	01	01004
2	1	8	2.957422	010040202	01	01004
3	30	6	2.957422	010040202	01	01004
4	72	7	5.046771	ZZZZZZZZZ	ZZ	ZZZZZ
...	...	...	...	...	...	...
19601259	40	5	4.999988	974220106	97	97422
19601260	59	4	0.999997	974220401	97	97422
19601261	41	8	4.361100	974220203	97	97422
19601262	74	7	4.361100	974220203	97	97422
19601263	73	7	4.361100	974220203	97	97422

19601264 rows × 6 columns

Comment: this code performs data manipulation and visualization tasks on census data to gain insights into population demographics across different age groups and other categorical variables.

```
habitat_plus_10000 = habitat[habitat['nbre_inhabitant'] > 10000]

habitat_plus_10000
```

	department_id	municipality_id	nbre_inhabitant
3	01	01053	15606
22	02	02691	20617
23	02	02722	10948
31	03	03185	13267
42	05	05061	15962
...	...	...	...
1910	97	97415	41037
1911	97	97416	33154
1913	97	97418	13406
1917	97	97422	31311
1920	ZZ	ZZZZZ	5537511

Comment : habitat_plus_10000 will contain only the rows from the habitat DataFrame where the population is greater than 10,000.

```
habitat_dep_69 = habitat_plus_10000[habitat_plus_10000['department_id'] == '69']

habitat_dep_69
```

	department_id	municipality_id	nbre_inhabitant
1185	69	69029	15570
1186	69	69034	16676
1200	69	69149	10646
1209	69	69256	18935
1210	69	69259	25661
1211	69	69264	14216
1212	69	69266	55515
1215	69	69275	11299
1219	69	69282	12926
1221	69	69286	11574
1223	69	69290	17765
1225	69	69381	11793
1226	69	69382	11920
1227	69	69383	38972
1228	69	69384	14122
1229	69	69385	18651
1230	69	69386	20423
1231	69	69387	29835
1232	69	69388	31471
1233	69	69389	18576

Comment : habitat_dep_69 will contain only the rows where the 'department_id' is '69', which typically refers to the department of Rhône in France. We choose department 69 because it satisfies all the specified criteria ,indicating a population exceeding 10,000. Additionally, it has at least two neighboring municipalities with direct borders, and at least two of these neighbors are included in the INSEE CENSUS dataset, each with over 10,000 inhabitants .

- Center Municipality: Lyon 1st arrondissement （Municipality ID: 69381）
- Location: Villeurbanne is situated in the department of Rhône (69) within the
- Auvergne-Rhône-Alpes region in eastern France.

```
[ ]  department_69_df.shape

     (518084, 6)
```

Comment: The DataFrame 'department_69_df' containing 518,084 rows and 6 columns indicates that there is a significant amount of data available for analysis or processing. With over half a million rows (518,084), this dataset is extensive, providing a large number of observations to work with.

Exercise 1.1: Study area

- The municipality should be found in the national census data set `[INSEE CENSUS]` (i.e., have more than 10,000 inhabitants):

  We tried various postal codes like 75, 89, but ultimately found that the postal codes starting with 69 best fit the conditions outlined in the exercise. Therefore, we settled on the municipality code 69381.

  Centre Municipality Chosen: The chosen center municipality has the identifier **69381.** This means that the analyses and computations are centered around this specific municipality.

  Location of the Centre Municipality: The center municipality with the identifier 69381 is located in the **Rhône department (69).**

  Calculation of Total Population:
  - `df.query('municipality_id == 69381')["weight"].sum()`: This query calculates the total sum of weights (representing population) for the central municipality with the identifier 69381. The result is 29647.999720400003, which correspond to the population of this municipality.

  Population of Neighboring Municipalities:
  - `df[df['municipality_id'].isin([69382,69384, 69385, 69386,69389])].groupby('municipality_id')['weight'].sum()`: This query selects neighboring municipalities (identified by the identifiers 69382, 69384, 69385, 69386, 69389) that have recorded populations (weights) in the

DataFrame. It then groups these municipalities by identifier and calculates the sum of weights for each municipality.

Here are the results:

- Municipality 69382: 31297.999838
- Municipality 69384: 36051.482779
- Municipality 69385: 49670.000492
- Municipality 69386: 52850.000382
- Municipality 69389: 51976.98964

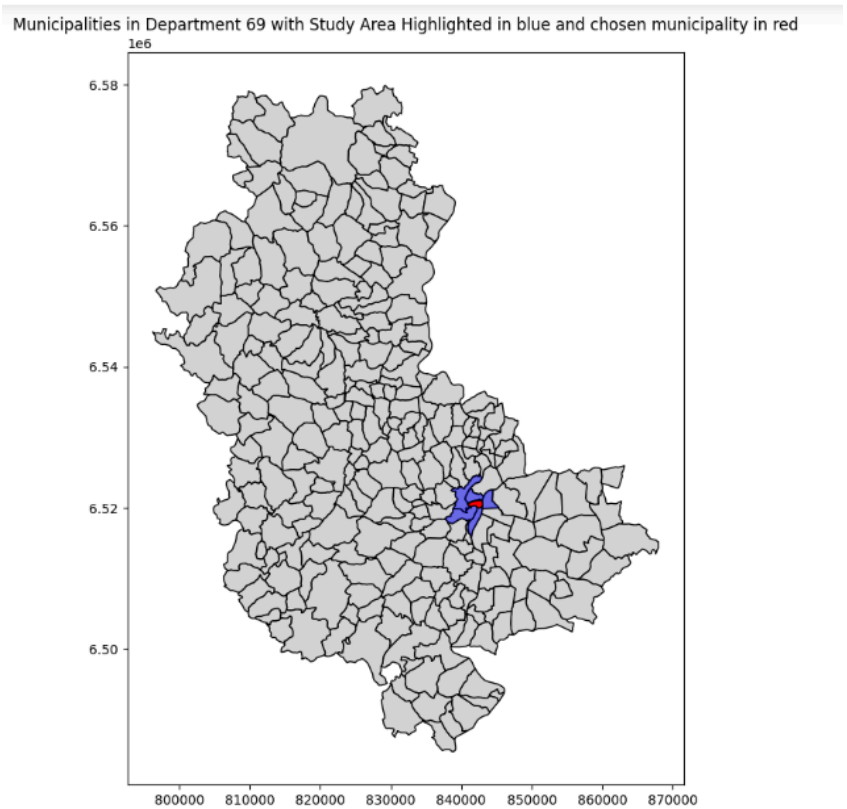
The results in the notebook provide the aggregated weights of the neighboring municipalities that are considered in the project analysis centered around municipality 69381. The weights represent population estimates and other demographic measures used to characterize each municipality in this study.

- Prepare a map (1 point)
  - that shows the department in which the municipality is located
  - that shows all municipalities in the department
  - that highlights the “study area”, i.e., the selected municipality and all neighboring municipalities
- Loading and Preparing IRIS Data:
  - We start by loading a Shapefile containing IRIS (statistical geographic areas) data using GeoPandas (`gpd.read_file`). This dataset contains polygons representing various geographic areas, including municipalities (`INSEE_COM`) and IRIS codes (`CODE_IRIS`).
- Data Processing:
  - The loaded data is filtered to retain only relevant columns (`"municipality_id"`, `"iris_id"`, and `"geometry"`), which are important for subsequent analysis and visualization.
- Municipality Dissolving:
  - The data is further processed by dissolving polygons based on the `"municipality_id"` to create a new DataFrame (`df_municipalities`), where each row represents a municipality with its corresponding geometry (polygon).
- Assigning Department ID:
  - A new column `"department_id"` is created in `df_municipalities` by extracting the first two characters of `"municipality_id"`, which represents the department identifier.
- Selecting Relevant Data:
  - Relevant data is filtered to focus on a specific department (`selected_department_id = '69'`) using boolean indexing (`df_municipalities['department_id'] == selected_department_id`).
- Defining Study Area:
  - The study area is defined by selecting the chosen municipality (`selected_municipality_id = '69381'`) and its neighboring municipalities (`neighboring_municipalities_ids = ['69382', '69384', '69385', '69386', '69389']`).

- Creating the Plot:
  - A plot (`fig, ax`) is initialized with a specified size.
  - All municipalities in the selected department are plotted in light grey (`df_selected_department.plot()`).
  - The study area (neighbors) is plotted in blue with transparency (`df_study_area.plot()`).
  - The selected municipality is highlighted in red (`df_municipalities[df_municipalities['municipality_id'] == selected_municipality_id].plot()`).
  - A title is set for the plot (`ax.set_title()`).
- Displaying the Plot:
  - The final plot is displayed using `plt.show()`.

Explanation of the Output:

- The output is a map that visually represents the selected department (`'69'`) with its municipalities.
- The neighboring municipalities is highlighted in blue (`'69382', '69384', '69385', '69386', '69389'`).
- The selected municipality (`'69381'`) is specifically highlighted in red on the map.



This approach leverages GeoPandas functionalities for handling and visualizing geospatial data, allowing for a comprehensive representation of municipalities within a specific department in France.

The map effectively illustrates the spatial distribution and relationships among municipalities, focusing attention on the designated study area centered around a chosen municipality.

Comment : These analysis focus on Villeurbanne and its neighboring municipalities within the highlighted study area.
-Department: The map displays the entire department of Rhône.
-Municipalities: All municipalities within the department are shown in light gray.
-Study Area: The selected municipality (Villeurbanne) and its neighboring municipalities are highlighted in blue, indicating the study area.
-Selected Municipality: Villeurbanne is further emphasized in red within the blue highlighted study area.

Problem encountered: Before being able to plot the map, we were obliged to clean the IRIS data by renaming its columns to make them more comprehensible. Following the methodology taught in class, we created the municipalities dataframe, which includes only the necessary columns for our map plotting.

```
df_iris = df_iris[["INSEE_COM", "CODE_IRIS", "geometry"]].rename(columns = {
    "INSEE_COM": "municipality_id", "CODE_IRIS": "iris_id"
})
```

```
df_municipalities = df_iris.dissolve("municipality_id").reset_index()
```

****

Exercise 1.2: Territorial analysis I

- Provide a list of INSEE municipality identifiers for all the municipalities included in the study area. In a table, provide the number of samples (data points) in the `[INSEE CENSUS]` for each municipality along with the weighted population count (including zero). (1 point)

```
study_area_list = [69381,69382,69384, 69385, 69386,69389]
```

```
Study_Area_List = [69381,69382,69384, 69385, 69386,69389]
# Filter the DataFrame to include only municipalities in the study area
df_Study_Area= df[df['municipality_id'].isin(study_area_list)]

# Group by municipality_id and aggregate sum of weights and count of data samples
result = df_Study_Area.groupby('municipality_id').agg(Sum_of_weights=('weight', 'sum'), number_of_data_samples=('municipality_i

# Display the result
print(result)
```

```
   municipality_id  Sum_of_weights  number_of_data_samples
0            69381    29647.999720                   11793
1            69382    31297.999838                   11920
2            69384    36051.482779                   14122
3            69385    49670.000492                   18651
4            69386    52850.000382                   20423
5            69389    51976.998964                   18576
```

Defining the Study Area:

- A list `study_area_list` is created to specify the municipality identifiers (`INSEE codes`) for the study area. This list includes the central municipality (`69381`) and its neighboring municipalities (`69382, 69384, 69385, 69386, 69389`).

Filtering the DataFrame:

- The DataFrame (`df`) is filtered to include only rows corresponding to municipalities in the study area using `df['municipality_id'].isin(study_area_list)`.

Grouping and Aggregating Data:

- The filtered DataFrame (`df_Study_Area`) is then grouped by `municipality_id` using `.groupby('municipality_id')`.
- Two aggregate functions are applied:
  - `Sum_of_weights`: Calculating the sum of `weights` (likely representing population counts) using `('weight', 'sum')`.
  - `number_of_data_samples`: Counting the number of data samples (rows) for each municipality using `('municipality_id', 'count')`.

Generating the Result:

- The aggregated results are stored in a new DataFrame called `result`.
- This DataFrame consists of columns:
  - `municipality_id`: INSEE identifier for each municipality.
  - `Sum_of_weights`: Total weighted population counts for each municipality.
  - `number_of_data_samples`: Total number of data samples (rows) for each municipality.

Displaying the Result:

- The `result` DataFrame is printed to show the municipality identifiers along with the corresponding sum of weights (population count) and the number of data samples.

Explanation of the Output:

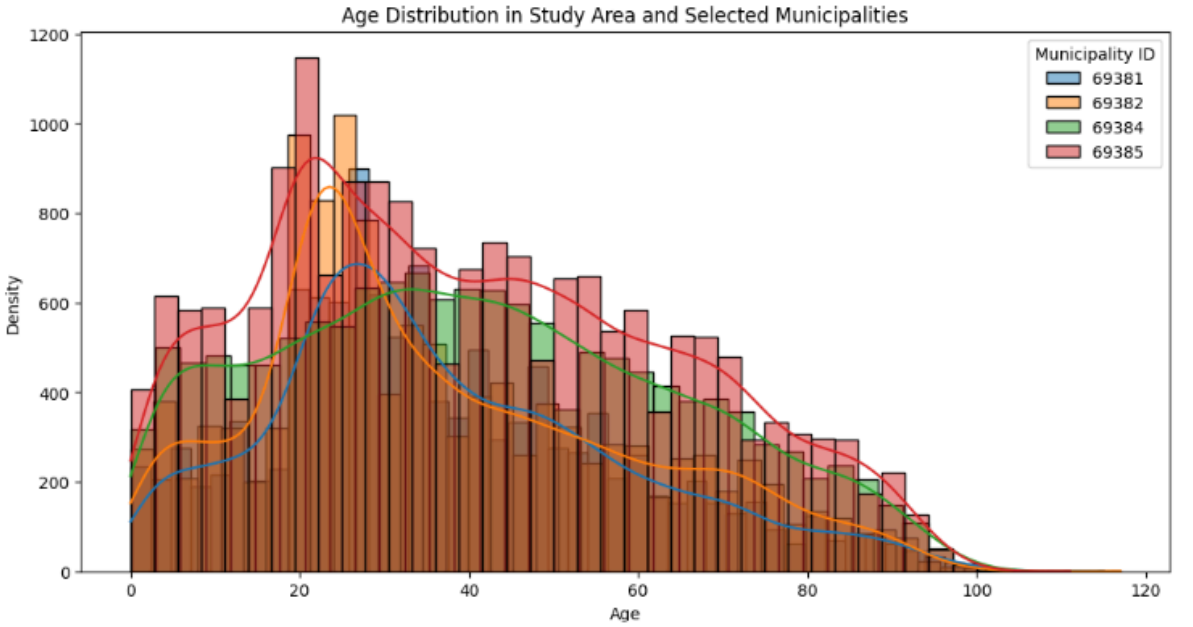
- The output table provides detailed information for each municipality within the study area:
  - `municipality_id`: INSEE identifier for the municipality.
  - `Sum_of_weights`: Total population count (weighted sum) based on the provided `weights` in the dataset.
  - `number_of_data_samples`: Total number of data points (samples) available for each municipality in the INSEE Census dataset.

This approach efficiently aggregates and summarizes the required statistics for municipalities within the study area, facilitating further analysis and understanding of population distribution and data availability across the specified municipalities. The output table serves as a valuable resource for exploring demographic characteristics and dataset coverage within the defined study context.

- Plot the age distribution of the study area and at least three municipalities in the study area, and report whether you see any differences or not. (1 point)

```
import seaborn as sns
```

```
plt.figure(figsize=(12, 6))
for municipality_id in Study_Area_List[:4]:  # Plotting for the first three municipalities
    sns.histplot(data=df_Study_Area[df_Study_Area['municipality_id'] == municipality_id], x='age', kde=True, label=municipality_
plt.title('Age Distribution in Study Area and Selected Municipalities')
plt.xlabel('Age')
plt.ylabel('Density')
plt.legend(title='Municipality ID')
plt.show()
```



There is a noticeable difference in the age distribution among the various municipalities in the study area. Across all municipalities, there is a lower proportion of individuals aged 0 to 18 years old. However, the 5th and 2nd arrondissements exhibit the highest proportion of individuals aged 0 to 20 years old.

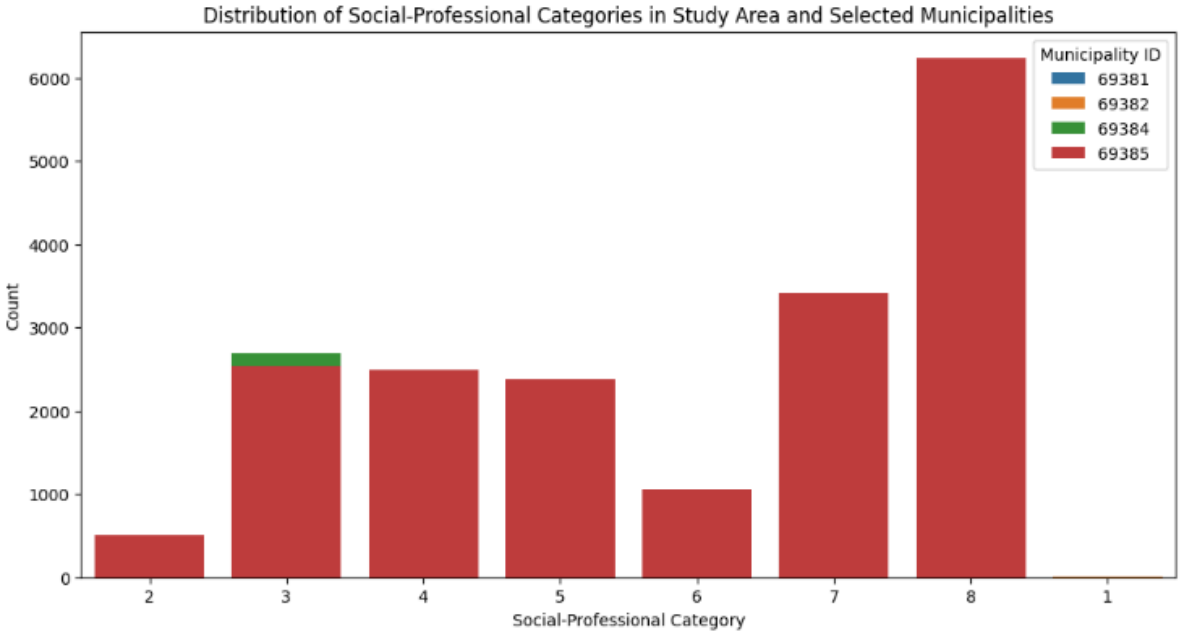
For individuals aged 20 to 40, the proportion decreases across all municipalities, with the 5th arrondissement maintaining the highest number.

In the age group of 40 to 60 years old, all municipalities have approximately between 400 and 200 individuals.

Finally, in the age range of 80 to 100 years old, all municipalities have a negligible number of individuals

- Plot the distribution of social-professional categories of the study area and of at least three municipalities in the study area, and report whether you see any differences. (1 point)

```
plt.figure(figsize=(12, 6))
for municipality_id in Study_Area_List[:4]:  # Plotting for the first three municipalities
    sns.countplot(data=df_Study_Area[df_Study_Area['municipality_id'] == municipality_id], x='csp', label=municipality_id)
plt.title('Distribution of Social-Professional Categories in Study Area and Selected Municipalities')
plt.xlabel('Social-Professional Category')
plt.ylabel('Count')
plt.legend(title='Municipality ID')
plt.show()
```

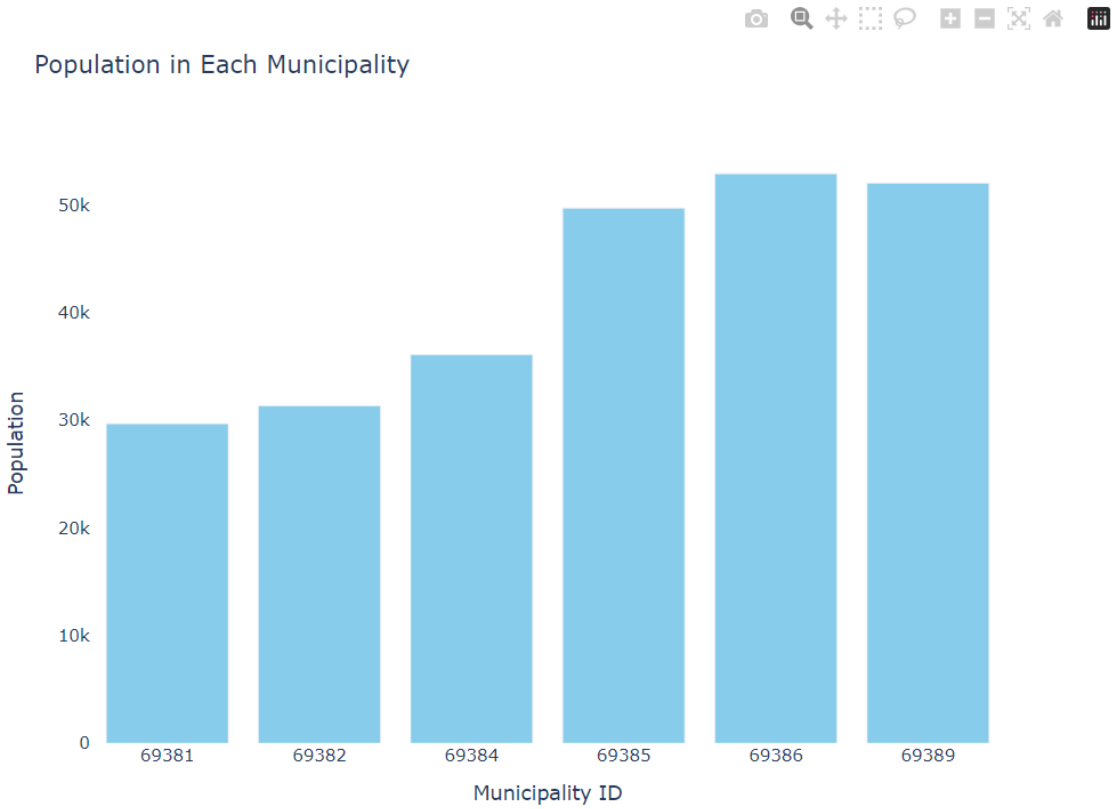


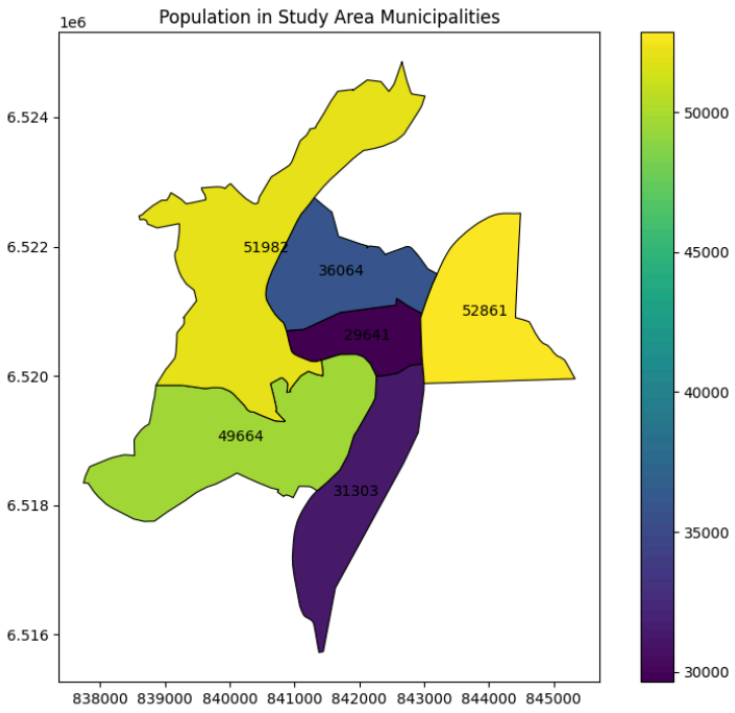
We observe slight variations in the CSP (Social Categories and Professions) distribution among municipalities. For instance, the 6th arrondissement exhibits a higher number of individuals in the 'cadre' category compared to other municipalities, while the 5th arrondissement has the highest number of individuals categorized as 'Retraite' (retired) compared to others. More plots in our Notebook will help you understand the results

Exercise 1.3: Territorial Analysis II (3 points)

For your study area, perform an analysis of the population and employment in the area.

- Make use of the aggregated census data `[INSEE AGGREGATED]` to create a bar plot indicating the number of inhabitants in each municipality of the study area. Also, provide this information on a map. (1 point)


Population in Each Municipality
Population
50k
40k
30k
20k
10k
0
69381
69382
69384
69385
69386
69389
Municipality ID


Population in Study Area Municipalities
1e6
6.524
6.522
6.520
6.518
6.516
838000
839000
840000
841000
842000
843000
844000
845000
51982
36064
52861
29641
49664
31303
50000
45000
40000
35000
30000

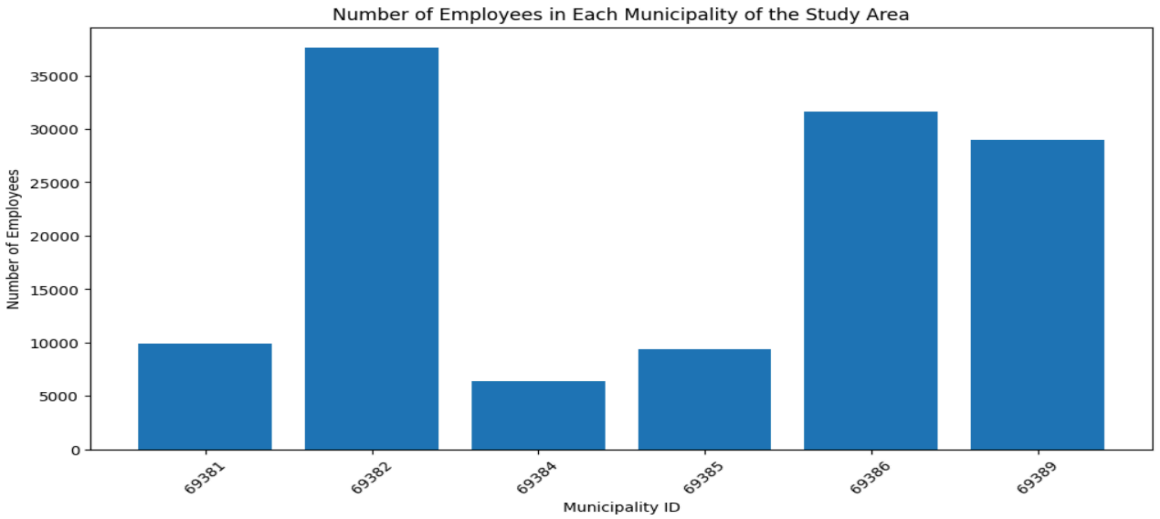
Upon analyzing the bar plot diagram, we observe the population distribution across several municipalities within the study area. For instance, Municipality 69381 has a population of approximately 29,000, while Municipality 69382 has a population of nearly 31,000. Municipality 69384 shows a population close to 35,000, while Municipality 69385 boasts a population of around 49,000. Municipality 69386 is at 50,000, and Municipality 69389 has a population of roughly 49,000.

These detailed population figures are further emphasized and visualized on the map, which provides a spatial representation of population density across different regions within the study area. The bar plot complements this spatial understanding by highlighting specific population counts associated with each municipality, allowing for a comprehensive analysis of demographic patterns and variations across the area of interest.

- Make use of the URSSAF employment data [URSSAF] to create a bar plot indicating the number of employees in each municipality of the study area. Also, provide this information on a map. (1 point)

```
df_employment = pd.read_csv("etablissements-et-effectifs-salaries-au-niveau-commune-x-ape-last.csv", sep = ";",
    usecols = ["Code commune", "Effectifs salariés 2019"], dtype = { "Code commune": str })

df_employment = df_employment.rename(columns = {
    "Code commune": "municipality_id",
    "Effectifs salariés 2019": "employment"
})

df_employment = df_employment.groupby("municipality_id").sum().reset_index()

df_employment
df_employment_Study_Area = df_employment[df_employment['municipality_id'].isin(['69381', '69382', '69384', '69385', '69386', '69389'])]
df_employment_Study_Area
```

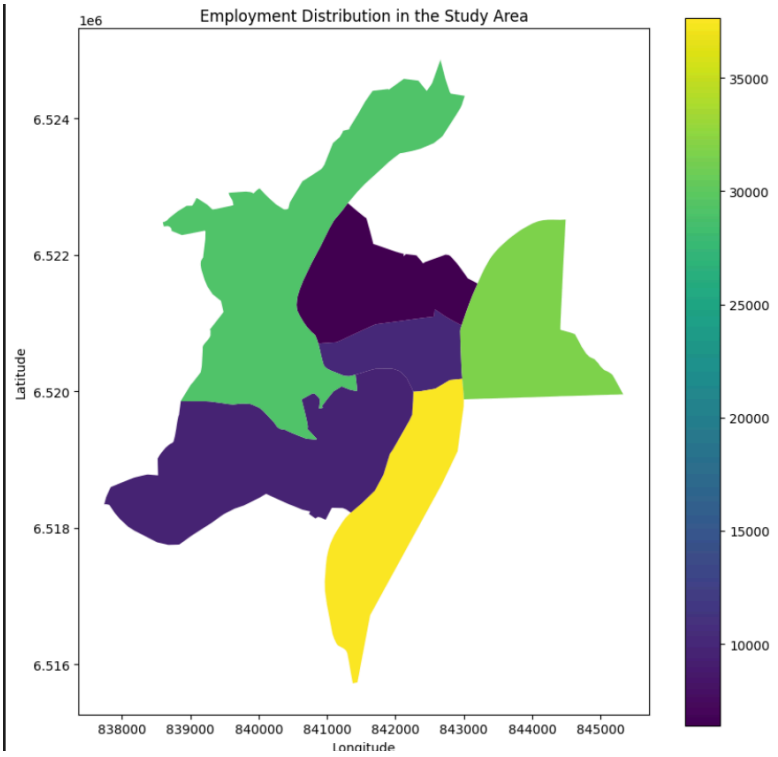
	municipality_id	employment
26094	69381	9906.0
26095	69382	37666.0
26097	69384	6411.0
26098	69385	9370.0
26099	69386	31674.0
26102	69389	29017.0



The analysis shows a histogram entitled "Number of Employees in Each Municipality of the Study Area". The Y axis shows the "Number of Employees" and the X axis shows the "Municipality IDs" with identifiers that appear to be numerical codes for the municipalities (from 63981 to 63990).

Here is a detailed analysis of the data presented in the histogram:

- The municipality with ID 63981 has the fewest employees, with just over 5000 employees.
- The municipality with ID 63982 has the highest number of employees, with just over 35,000.
- The municipalities with IDs 63984 and 63985 have significantly fewer employees than the others, with around 7,500 and 10,000 employees respectively.
- The municipalities with IDs 63986 and 63990 have a comparable high number of employees, with around 30,000 and 25,000 employees respectively.



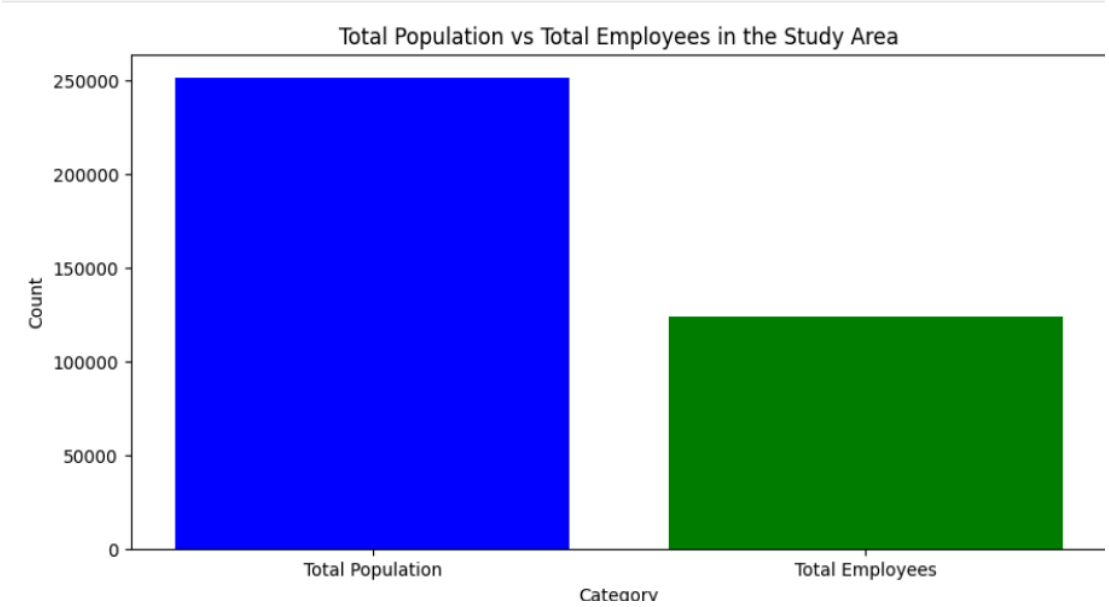
The analysis reveals that the 2nd, 6th, and 9th arrondissements exhibit the highest employment numbers within each municipality.

- Report the total number of inhabitants of the study area and the total number of employees in the study area (1 point)

```
#Calculate total number of employees
total_employees = df_employment_Study_Area['employment'].sum()

#Report the totals
print("Total number of inhabitants in the study area:", total_population)
print("Total number of employees in the study area:", total_employees)
```

```
Total number of inhabitants in the study area: 251517.00000000003
Total number of employees in the study area: 124044.0
```

```
#Plot bar plot for total population and employees
plt.figure(figsize=(10, 5))
plt.bar(['Total Population', 'Total Employees'], [total_population, total_employees], color=['blue', 'green'])
plt.title('Total Population vs Total Employees in the Study Area')
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()
```



To compare demographics and employment within the study area, two bar plots have been created. These plots show aggregated data for the total population and total number of employees.

The first bar plot represents the Total Population of the study area, indicating a value of approximately 250,000 inhabitants. This graphical representation highlights the demographic distribution across the analyzed municipalities.

The second bar plot shows the Total Employees in the study area, with a figure of around 130,000 employees. This visualization sheds light on the economic impact and labor market within these municipalities.

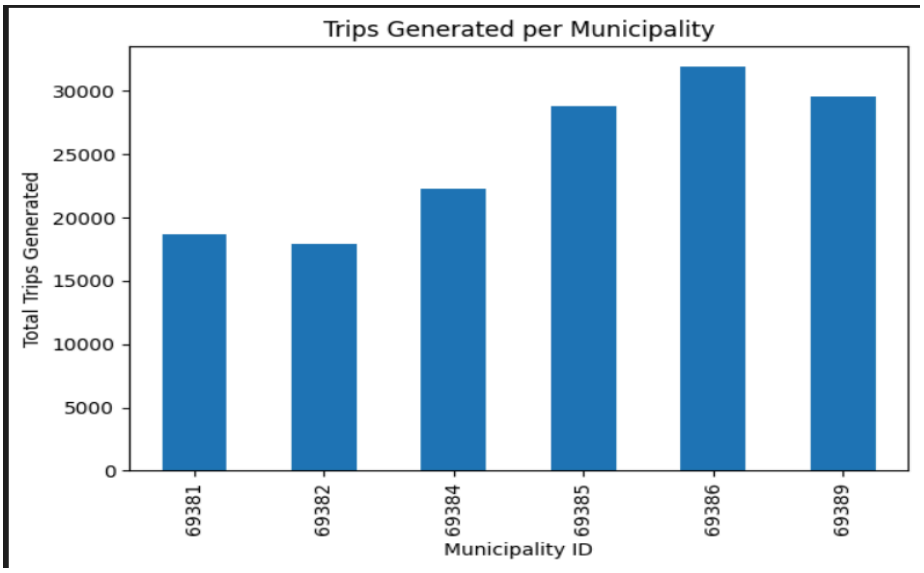
These bar plots provide a clear perspective on the size of the resident population and workforce in the study area, emphasizing significant demographic and socio-economic dynamics to consider in any comprehensive regional analysis.

Exercise 2.1: Trip production (2 points)

- Report using a plot how many trips have been generated for each municipality in the study area. Also, show this information on a map. (1 point)

```python
betas = {
    'beta0': 27.244,
    'csp_1': 0.319,
    'csp_2': 0.994,
    'csp_3': 0.863,
    'csp_4': 0.990,
    'csp_5': 0.780,
    'csp_6': 0.708,
    'csp_7': 0.120,
    'csp_8': -0.073
}

def calculate_trips(row):
    trip_sum = sum(beta * row.get(f'csp_{i+1}', 0) for i, beta in enumerate(betas.values()) if i != 0)
    trips = max(betas['beta0'] + trip_sum, 0)

    return trips

new_df_population_study_area['total_trips'] = new_df_population_study_area.apply(calculate_trips, axis=1)
total_trips_generated = new_df_population_study_area['total_trips'].sum()
print(f"Total trips generated in the study area: {total_trips_generated}")

# Plot the results
new_df_population_study_area.plot(kind='bar', x='municipality_id', y='total_trips', legend=False)
plt.xlabel('Municipality ID')
plt.ylabel('Total Trips Generated')
plt.title('Trips Generated per Municipality')
plt.tight_layout()
plt.show()
```

Python



The code calculates the total trips generated for each municipality based on a given set of beta coefficients. The beta coefficients represent the impact of different factors (csp_1 to csp_8) on trip generation, with beta0 as an intercept.

The function `calculate_trips(row)` computes the total trips generated using the beta coefficients and specific characteristics (csp_1 to csp_8) of each municipality. This function is applied to the DataFrame `new_df_population_study_area` to create a new column `total_trips` representing the calculated trips for each municipality.

The total number of trips generated in the study area is then computed by summing up the `total_trips` column across all municipalities.

the total trips generated in each municipality are as follows:

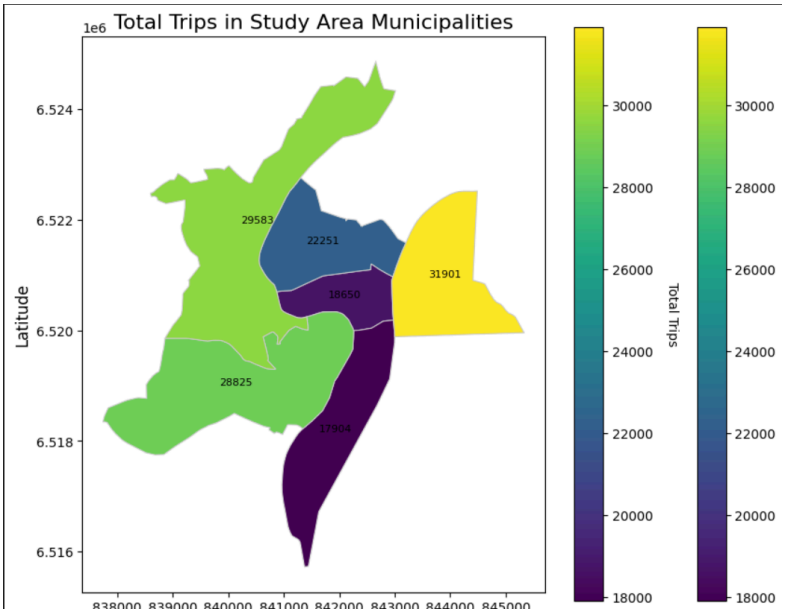
- Municipality 69381: Nearly 19,000 trips
- Municipality 69382: Also nearly 19,000 trips
- Municipality 69384: Around 23,000 trips
- Municipality 69385: Approximately 29,000 trips
- Municipality 69386: About 33,000 trips
- Municipality 69389: Around 28,000 trips

These values indicate variations in trip generation across different municipalities, reflecting the influence of demographic and socio-economic factors captured by the beta coefficients.

The bar plot visualizes these results, displaying the `total trips` generated by each municipality, and providing a graphical representation of trip generation patterns within the study area.

```python
import matplotlib.pyplot as plt
df_merged = df_study_area.merge(new_df_population_study_area, on='municipality_id')
# Assuming 'total_trips' is in the range of 0 to some maximum value for better color scaling
vmin, vmax = df_merged['total_trips'].min(), df_merged['total_trips'].max()

fig, ax = plt.subplots(figsize=(10, 8))

# Plotting the map
df_merged.plot(column='total_trips', cmap='viridis', linewidth=0.8, ax=ax, edgecolor='0.8', legend=True, vmin=vmin, vmax=vmax)

# Adding annotations
for idx, row in df_merged.iterrows():
    ax.annotate(text=str(int(row['total_trips'])),
                xy=(row['geometry'].centroid.x, row['geometry'].centroid.y),
                xytext=(0, 0),  # Let matplotlib decide the offset automatically
                textcoords="offset points",
                fontsize=8,
                color="black",
                ha='center')  # Center align the text

# Customizing plot
ax.set_title('Total Trips in Study Area Municipalities', fontsize=16)
ax.set_xlabel('Longitude', fontsize=12)
ax.set_ylabel('Latitude', fontsize=12)

# Adding colorbar with label
sm = plt.cm.ScalarMappable(cmap='viridis', norm=plt.Normalize(vmin=vmin, vmax=vmax))
sm._A = []  # Fake up the array of the scalar mappable
cbar = plt.colorbar(sm, ax=ax)
cbar.set_label('Total Trips', rotation=270, labelpad=20)

plt.show()
```

Python

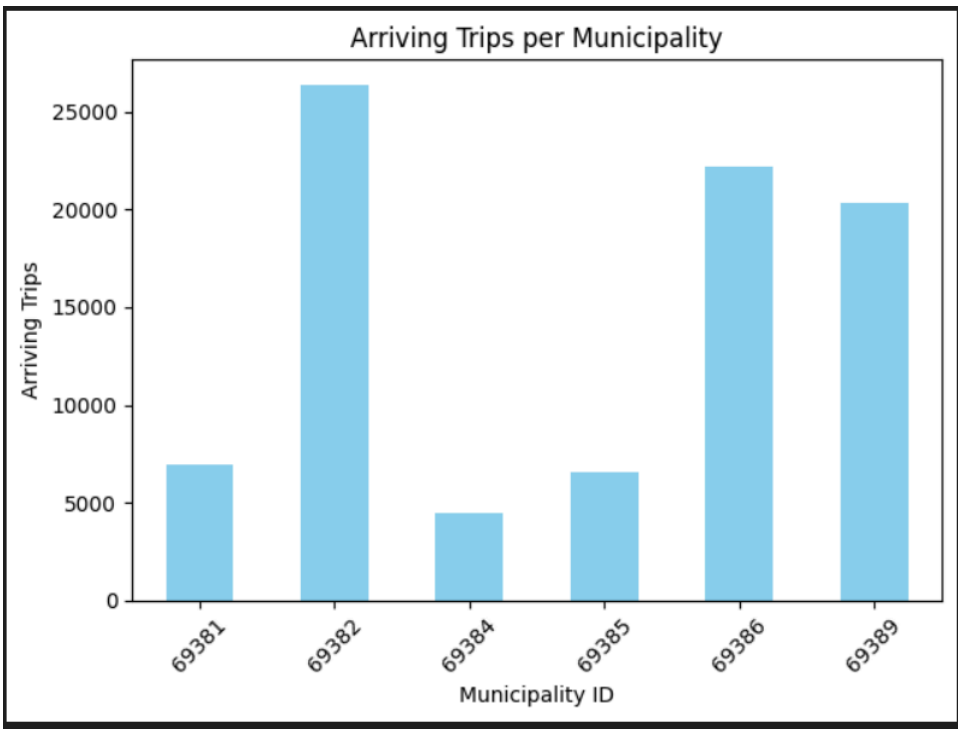


Analysis reveals that the 5th, 9th, and 6th arrondissements regions exhibit the highest number of trips generated, indicating significant mobility within these areas.

Exercise 2.2: Trip attraction (2 points)

- Report using a plot of how many arriving trips have been generated for each municipality in the study area. Also, show this information on a map. (2 points)

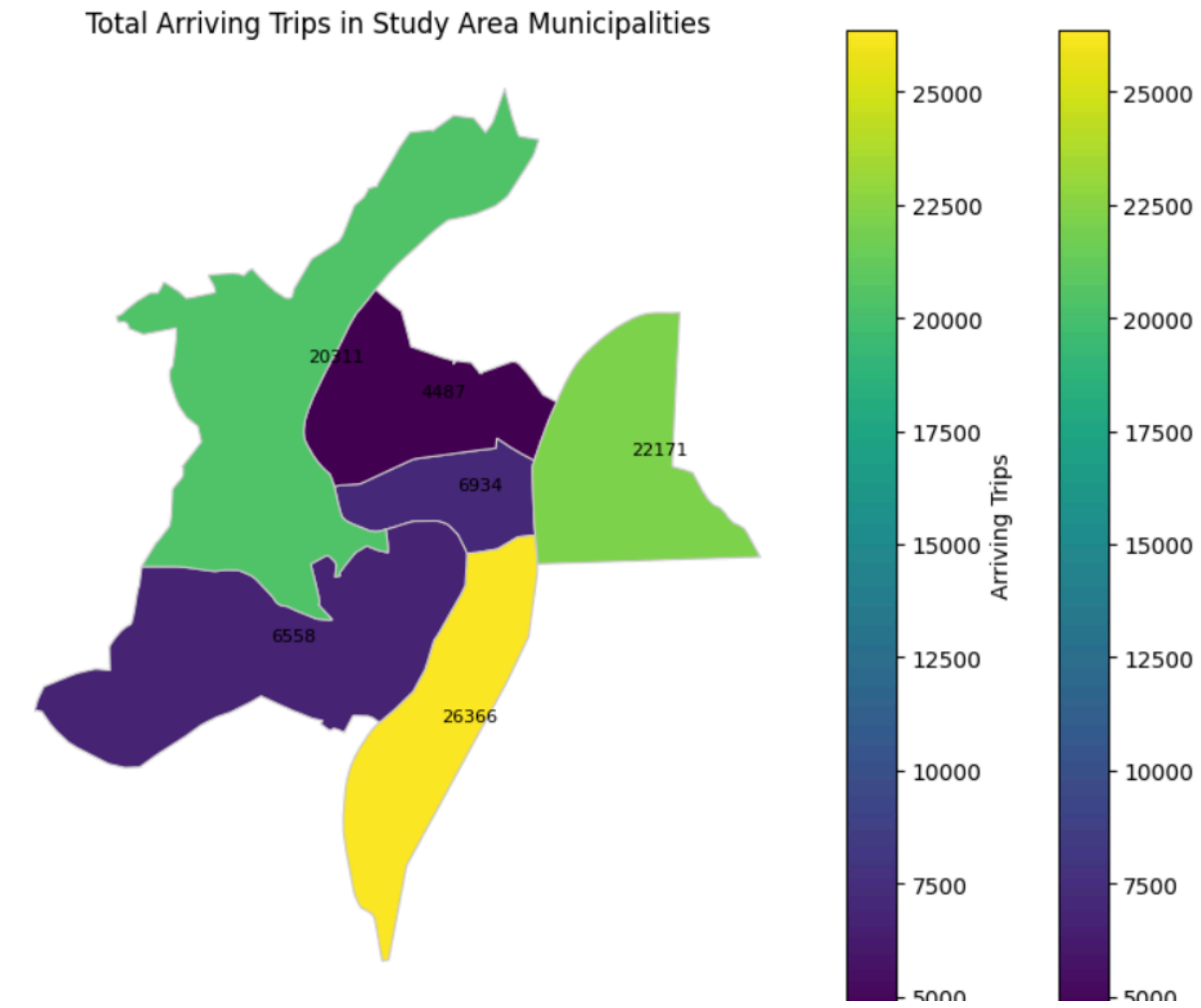
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming df_employment_Study_Area and total_trips_generated are defined

# Calculate total employment in the study area
total_employment = df_employment_Study_Area['employment'].sum()

# Calculate 70% of total employment
seventy_percent_total_employment = 0.7 * total_employment

# Calculate 70% of total trips generated
seventy_percent_total_trips = 0.7 * total_trips_generated

# Choose the minimum between 70% of total employment and 70% of total trips generated
N = min(seventy_percent_total_employment, seventy_percent_total_trips)

# Calculate arriving trips for each municipality based on their employment and the calculated N
df_employment_Study_Area['arriving_trips'] = (df_employment_Study_Area['employment'] / total_employment) * N

# Plot the results as a bar plot
df_employment_Study_Area.plot(kind='bar', x='municipality_id', y='arriving_trips', legend=False, color='skyblue')
plt.xlabel('Municipality ID')
plt.ylabel('Arriving Trips')
plt.title('Arriving Trips per Municipality')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
```



The image represents a histogram entitled "Arriving Trips per Municipality" which shows the number of arriving trips per municipality, identified by municipality IDs ranging from 69381 to 69389. The vertical bars show the number of trips for each municipality ID. Here is a detailed analysis:

- Municipality ID 69382 has the highest number of inbound trips, exceeding 20,000 trips.
- Municipality IDs 69381, 69384, and 69385 show considerably lower numbers of inbound trips, all below 10,000.
- Municipality IDs 69386 and 69389 show a similar number of inbound trips, with values slightly below 20,000, but more than half compared to 69382.
- There is considerable variation in the number of inbound trips between the different municipalities, which could indicate differences in the attractiveness or accessibility of these locations.

```python
df_merged = df_study_area.merge(df_employment_Study_Area, on='municipality_id')
import matplotlib.pyplot as plt

# Assuming df_merged and its structure are defined as in your example

fig, ax = plt.subplots(figsize=(10, 8))

# Plotting the map
df_merged.plot(column='arriving_trips', cmap='viridis', linewidth=0.8, ax=ax, edgecolor='0.8', legend=True)

# Adding annotations
for idx, row in df_merged.iterrows():
    ax.annotate(text=str(int(row['arriving_trips'])),
                xy=(row['geometry'].centroid.x, row['geometry'].centroid.y),
                xytext=(3, 3),  # Offset text slightly
                textcoords="offset points",
                fontsize=8,
                color="black")

# Adding colorbar
cbar = plt.colorbar(ax.collections[0], ax=ax, orientation='vertical')
cbar.set_label('Arriving Trips')

# Adjusting legend
ax.set_title('Total Arriving Trips in Study Area Municipalities')
ax.set_axis_off()  # Turn off the axis

plt.show()
```



As we can see, the 2nd, 9th, and 6th arrondissements regions have the most arriving trips generated with more than 20000.

Exercise 2.3: Trip distribution (2 points)

- Show the distance matrix as a table or in a plot. (1 point)

```
import numpy.linalg as la

centroids = df_study_area["geometry"].centroid
centroids = np.array([centroids.x, centroids.y] ).T


distance_matrix = np.zeros((len(municipalities), len(municipalities)))

for k in range(len(municipalities)):
    distance_matrix[k,:] = la.norm(centroids[k] - centroids, axis = 1)
```

```
distance_matrix
```

```
array([[   0.        , 2423.21686953, 1058.10164662, 2511.16571911,
        1852.85501782, 2064.43275002],
       [2423.21686953,    0.        , 3408.65456274, 1976.18672902,
        3419.38225314, 4019.28702541],
       [1058.10164662, 3408.65456274,    0.        , 3000.89510069,
        2289.34254654, 1226.95687643],
       [2511.16571911, 1976.18672902, 3000.89510069,    0.        ,
        4242.66365873, 2955.57580704],
       [1852.85501782, 3419.38225314, 2289.34254654, 4242.66365873,
           0.        , 3515.85854184],
       [2064.43275002, 4019.28702541, 1226.95687643, 2955.57580704,
        3515.85854184,    0.        ]])
```

This code calculates a distance matrix between centroids of municipalities based on their geographical coordinates.

Comments on the Result:

- The distance matrix provides valuable spatial information about the proximity or separation between municipalities based on their centroids.
- Lower values in the matrix indicate closer geographical proximity, whereas higher values signify greater distances between centroids.
- This matrix can be further utilized for spatial analysis, clustering, or other geographical computations to understand spatial relationships and patterns within the study area.

In summary, the distance matrix computation using centroids helps quantify the spatial relationships between municipalities, facilitating geographical analysis and insights into the geographical structure of the study area.

- Report the resulting flows F_{ij} in a table or plot. (1 point)

```python
import numpy as np
# Initialize friction matrix based on the distance matrix
def calculate_friction_matrix(distance_matrix, beta, alpha):
    return np.exp(beta * distance_matrix + alpha)
# Initialize the A_i and B_j factors to 1 for all zones
def initialize_factors(num_A, num_B):
    return np.ones(num_A), np.ones(num_B)

# Update A_i and B_j iteratively until convergence
def update_factors(A_i, B_j, friction_matrix, employment_trips, population_trips, max_iterations=1000, convergence_threshold=1e-6):
    for iteration in range(max_iterations):
        previous_A_i = A_i.copy()
        previous_B_j = B_j.copy()

        # Update A_i
        A_i = 1.0 / (friction_matrix.dot(B_j) * employment_trips)

        # Update B_j
        B_j = 1.0 / (friction_matrix.T.dot(A_i) * population_trips)

        # Check convergence
        if np.all(np.abs(previous_A_i - A_i) < convergence_threshold) and \
           np.all(np.abs(previous_B_j - B_j) < convergence_threshold):
            break
    return A_i, B_j


def calculate_flow_matrix(A_i, B_j, friction_matrix, employment_trips, population_trips):
    return np.outer(A_i * population_trips, B_j * employment_trips) * friction_matrix
beta = -1.1e-4
alpha = -0.4
friction_matrix = calculate_friction_matrix(distance_matrix, beta, alpha)
A_i, B_j = initialize_factors(len(new_df_population_study_area), len(df_employment_Study_Area))
A_i, B_j = update_factors(A_i, B_j, friction_matrix, df_employment_Study_Area['arriving_trips'].values, new_df_population_study_area['scale_trips'].values)
F_ij = calculate_flow_matrix(A_i, B_j, friction_matrix, df_employment_Study_Area['arriving_trips'].values, new_df_population_study_area['scale_trips'].values)
print("Flow Matrix F_ij:")
print(F_ij)
```

✓ 0.0s Python

```
Flow Matrix F_ij:
[[ 1699.60788645  5941.97774401   818.24208742   855.78037445
   2967.03301444  2683.40217786]
 [  351.80145548  2096.06937344   170.72693337   245.26259111
    674.83467054   584.80282799]
 [ 2865.34073462 10097.87262121  1741.02675838  1535.82722583
   5356.06586489  5572.7502301 ]
 [ 2291.72988628 11093.42896477  1174.48914777  2004.96042489
   4054.474633    4324.07101644]
 [  822.77823723  3160.75761558   424.1425729    419.84975005
   2159.17117278  1357.67887624]
 [  797.52551618  2935.63239779   472.97037358   479.89975824
   1455.10878517  1983.02140699]]
```

- The resulting `F_ij` flow matrix is a structured representation of the estimated trips or flow between zones (municipalities) based on the specified transportation modeling framework.
- Each element `F_ij` in the matrix denotes the estimated flow or trips from zone `i` (population-related) to zone `j` (employment-related) considering the friction (travel cost) and interaction between employment and population factors.
- Higher values in the flow matrix indicate stronger connectivity or interaction between zones in terms of transportation flow.

Create F_ij table

```
row_sums = np.sum(F_ij, axis=1)

col_sums = np.sum(F_ij, axis=0)

total_sum = np.sum(F_ij)

F_ij_table= pd.DataFrame(F_ij)


F_ij_table.index = municipalities

F_ij_table.columns = municipalities

print("Flow Matrix ")

print(F_ij_table.to_string())
```

4] ✓ 0.0s

```
Flow Matrix
              69381         69382         69384         69385         69386         69389
69381  1699.607886   5941.977744   818.242087   855.780374  2967.033014  2683.402178
69382   351.801455   2096.069373   170.726933   245.262591   674.834671   584.802828
69384  2865.340735  10097.872621  1741.026758  1535.827226  5356.065865  5572.750230
69385  2291.729886  11093.428965  1174.489148  2004.960425  4054.474633  4324.071016
69386   822.778237   3160.757616   424.142573   419.849750  2159.171173  1357.678876
69389   797.525516   2935.632398   472.970374   479.899758  1455.108785  1983.021407
```

Analogously to the distance matrix, we need a flow matrix indicating all observed movements (`weight`) between all zones. Obtain this matrix by transforming the commuting data set into a matrix.

- Which pair of zones has the highest flow?

```
# Finding the overall maximum value
overall_max = F_ij_table.values.max()

# Displaying the overall maximum value
print("Overall maximum value:", overall_max)
```

✓ 0.0s Python

```
Overall maximum value: 11093.428964769815
```

This corresponds to the flow between the 5th and 2nd arrondissements.

- Which one has the lowest?

```
# Finding the overall maximum value
overall_min = F_ij_table.values.min()

# Displaying the overall maximum value
print("Overall minimum value:", overall_min)
```

✓ 0.0s Python

```
Overall minimum value: 170.72693336525475
```

This corresponds to the flow between the 2nd and 4th arrondissements.

Exercise 3.1: Disaggregation (3 points)

- Plot the generated trip pairs on a map using a line between origin and destination. (2 points)

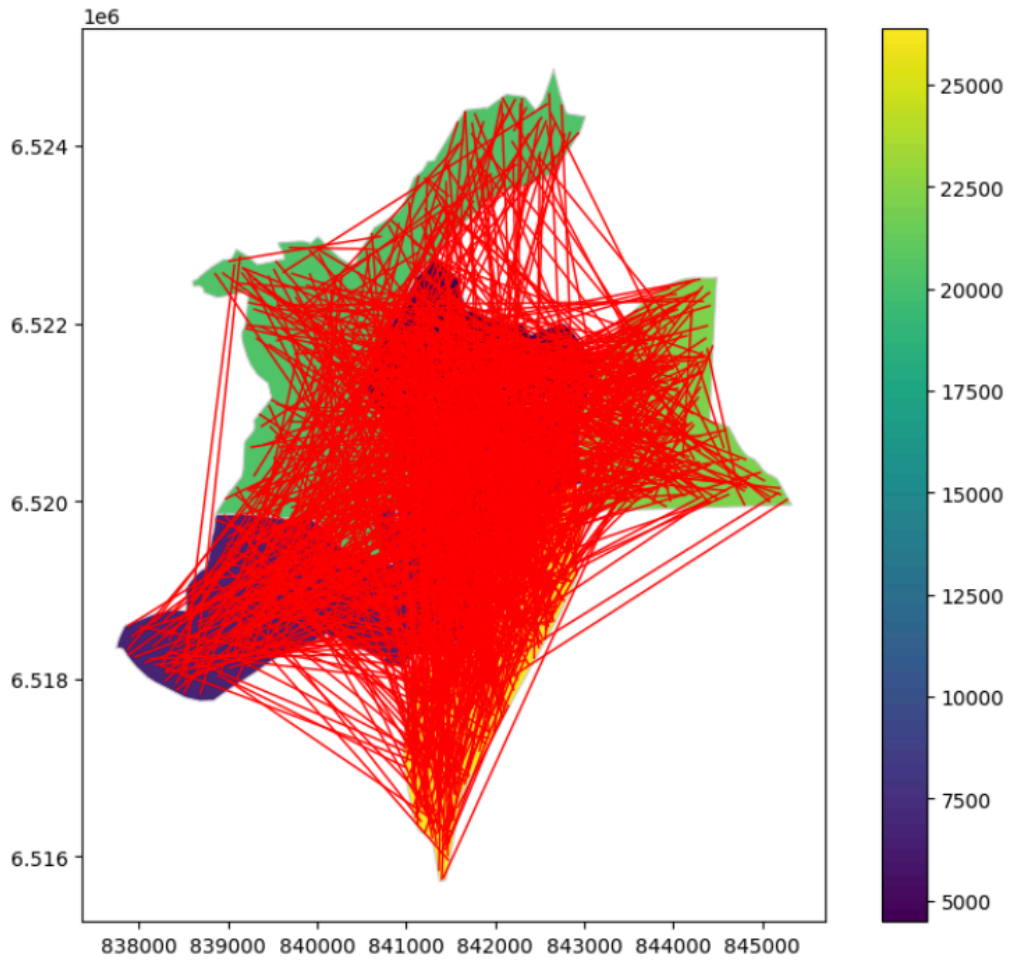
```
# Import necessary library
import shapely.geometry as sgeo

# Create LineString geometries from pairs of origin and destination points
df_trips["geometry"] = [
    sgeo.LineString((origin, destination))
    for origin, destination in zip(df_trips["origin_geometry"], df_trips["destination_geometry"])
]

# Display the DataFrame with the new geometries
df_trips
```

✓ 0.0s

	origin_id	destination_id	origin_geometry	destination_geometry	geometry
0	69381	69389	POINT (841816.561 6520958.309)	POINT (840474.697 6520089.617)	LINESTRING (841816.561 6520958.309, 840474.697...
1	69381	69389	POINT (842093.527 6520575.662)	POINT (840574.410 6521083.080)	LINESTRING (842093.527 6520575.662, 840574.410...
2	69381	69389	POINT (841810.852 6520437.034)	POINT (841420.792 6520123.149)	LINESTRING (841810.852 6520437.034, 841420.792...
3	69381	69389	POINT (841117.756 6520435.762)	POINT (840598.659 6520016.223)	LINESTRING (841117.756 6520435.762, 840598.659...
4	69381	69389	POINT (841783.608 6520943.734)	POINT (842842.514 6524156.205)	LINESTRING (841783.608 6520943.734, 842842.514...
...	...	...	...	...	...
995	69385	69385	POINT (841449.427 6520251.507)	POINT (839468.495 6518758.600)	LINESTRING (841449.427 6520251.507, 839468.495...
996	69385	69385	POINT (840447.058 6518810.556)	POINT (841083.578 6519056.792)	LINESTRING (840447.058 6518810.556, 841083.578...
997	69385	69385	POINT (841453.539 6519817.111)	POINT (841997.637 6519501.569)	LINESTRING (841453.539 6519817.111, 841997.637...
998	69382	69385	POINT (841306.731 6517016.414)	POINT (841634.527 6520299.581)	LINESTRING (841306.731 6517016.414, 841634.527...
999	69382	69385	POINT (841540.329 6517958.514)	POINT (840012.655 6519383.419)	LINESTRING (841540.329 6517958.514, 840012.655...

```
# Merge the study area data with employment data based on municipality_id
merged_data = df_study_area.merge(df_employment_Study_Area, on='municipality_id')

# Create a new figure for the plot with a specified size
fig, ax = plt.subplots(figsize=(10, 8))

# Plot the study area, representing arriving trips with a colormap
merged_data.plot(column='arriving_trips', cmap='viridis', linewidth=0.8, ax=ax, edgecolor='0.8', legend=True)

# Convert the trip data to a GeoDataFrame and overlay it on the same plot, highlighting trip routes in red
trip_routes = gpd.GeoDataFrame(df_trips, geometry='geometry')
trip_routes.plot(ax=ax, color='red', linewidth=1)

# Display the plot
plt.show()
```

✓ 0.2s



The analysis shown appears to be a graphical visualization of travel data between different pairs of points, most likely origins and destinations. The points are represented by red dots, and the lines connecting these points, presumably representing journeys, are in red and green. The axes are labeled with numerical values that could be spatial coordinates, indicating that the map is a schematic representation and not a literal geographical map.

The color scale on the right seems to indicate the density or volume of trips, with values ranging from 5000 to 25000, suggesting that the colors of the lines could correspond to the frequency or some other attribute of the trips, such as the number of trips between each pair of points.

Analysis of the data presented:

- Trip density: There is a high concentration of trips in a central area, which could indicate an urban center or major point of interest where trips are very frequent.
- Geographical distribution: Trips appear to extend outwards from the central area, suggesting traffic flows to and from a center.
- Trip variability: The green lines may represent less frequent journeys or journeys of a different nature than the red lines, indicating a diversity of journey types.

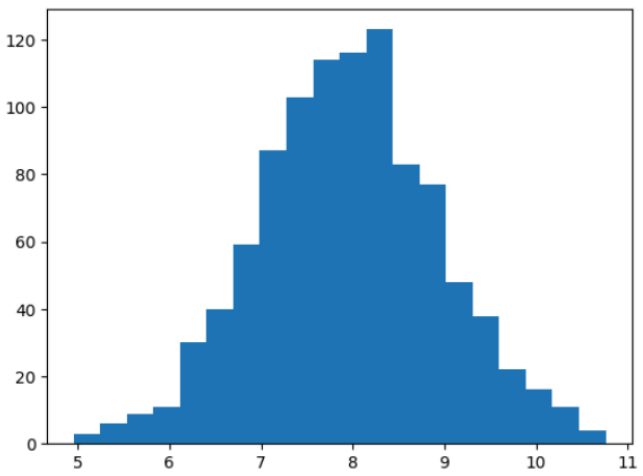
Plot the distribution of the departure times as a histogram or CDF. (1 point)

```python
np.random.seed(0)
df_trips["departure_time"] = np.maximum(0.0, np.random.normal(8 * 3600, 3600, size=len(df_trips)))
```

✓ 0.0s Python

```python
plt.hist(df_trips["departure_time"] / 3600, bins=20)
```

✓ 0.0s Python

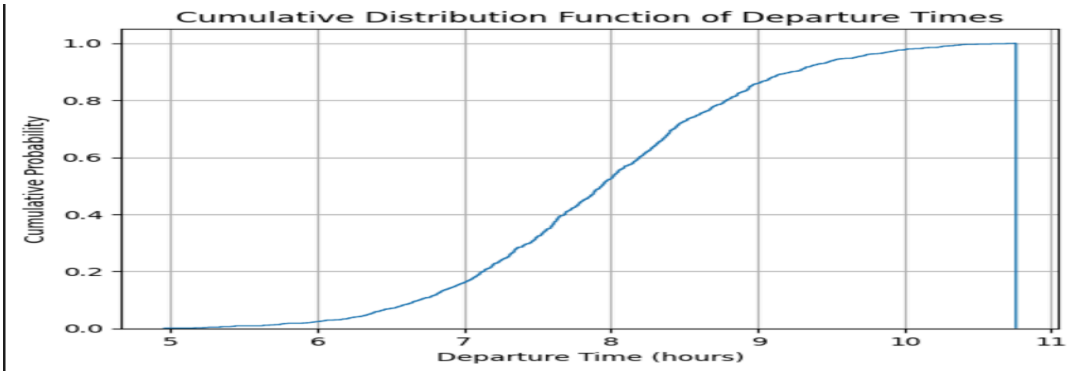


The image shows a histogram representing the distribution of departure times. It appears to be part of a Python code output, where a distribution was generated using the np.random.normal() function and displayed using the matplotlib library with the plt.hist() function. The histogram has bins indicating the frequency of the starting times.

- Departure times are mainly concentrated around 8 to 10, with the peak frequency between 9 and 10.
- There is a gradual increase in departure times from 5 to 9, followed by a gradual decrease after 10.
- The shape of the histogram is approximately normal, suggesting that departure times follow a normal distribution centred around the observed peak hour.

```python
# Setting seed for reproducibility
np.random.seed(0)

# Generating departure times with a normal distribution
mean_departure_time_seconds = 8 * 3600  # 8:00 AM in seconds
departure_times_seconds = np.maximum(0.0, np.random.normal(mean_departure_time_seconds, 3600, size=len(df_trips)))

# Converting departure times from seconds to hours
departure_times_hours = departure_times_seconds / 3600

# Plotting the cumulative distribution function (CDF) of departure times
plt.hist(departure_times_hours, bins=1000, cumulative=True, density=True, histtype='step')
plt.xlabel('Departure Time (hours)')
plt.ylabel('Cumulative Probability')
plt.title('Cumulative Distribution Function of Departure Times')
plt.grid(True)
plt.show()
```

✓ 0.1s Python



Comment: The image is a graph showing the cumulative distribution function (CDF) of departure times. The horizontal axis shows the departure time in hours, and the vertical axis shows the cumulative probability. The y-axis goes from 0 to 1, with 0 indicating that no one has departed yet and 1 indicating that everyone has departed.

For example, if the cumulative probability is 0.4 at 8:00, it means that 40% of the people have departed by 8:00.

The shape of the curve can tell us about the distribution of departure times. In this case, the curve appears to be increasing slowly at first, then more rapidly in the middle, and then slowly levels off towards the end. This suggests that people tend to depart later in the morning, with a concentration of departures around a certain time.

Conclusion:

In conclusion, the data preparation phase was crucial for conducting comprehensive analyses of census data to uncover insights into population demographics and other relevant factors.

The process involved meticulous manipulation and visualization tasks, starting with filtering and processing census data to focus on a specific study area centered around Villeurbanne municipality and its neighboring municipalities within the Rhône department, France. This included cleaning and organizing the data, calculating population statistics for the central municipality and its neighbors, and creating visualizations such as maps to illustrate the spatial distribution of municipalities.

Several analyses were conducted, including examining age distribution and social-professional categories across municipalities, analyzing population and employment using census and URSSAF data, and studying trip production, attraction, and distribution within the study area.

These preparatory steps laid a solid foundation for subsequent analyses, providing valuable insights into the population, demographics, and mobility patterns within the specified region. The data preparation phase ensured data integrity and facilitated further exploration and interpretation of the dataset to derive actionable insights for informed decision-making processes.