Data Analysis Project - Velib in Paris

Authors: Yasmine BOULKAID, Edda IVELAND, Vilde OPDAL, Laura VAZQUEZ

Institut National des Sciences Appliquées de Toulouse, 2023-2024

4th year, Génie de Mathématiques et Modélisation

Part 1: An introduction to the data (descriptive statistics)

Every day, thousands of parisians use the velib bikes to get around the city. In this project, we are going to analyze the circulation of bikers and the business of the different stations in Paris. To begin with, we will present the data set we are working with. Secondly, we will perform a principal component analysis for dimension reduction. Thirdly, we will implement different clustering methods to the raw data and the projected data. Followingly, we will do a Multiple Correspondence Analysis and k-means clustering on its projected data. Finally, we will conclude with interpretations on the whole analysis.

1.1. Uploading and verification of the data

In [1]:
```
# We start by importing the nescessary packages for the analysis
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')

%matplotlib inline
import pandas as pd
import numpy as np
import random as rd
```

In [2]:
```
# Uploading the data
path    = ''  # If data in 'data' directory
loading = pd.read_csv(path + 'velibLoading.csv', sep = " ")
coord = pd.read_csv(path + 'velibCoord.csv', sep = " ")
```

In [3]:
```
# Here we show the first five lines of loading
loading.head()
```

Out[3]:

	Lun-00	Lun-01	Lun-02	Lun-03	Lun-04	Lun-05	Lun-06	Lun-07
1	0.038462	0.038462	0.076923	0.038462	0.038462	0.038462	0.038462	0.038462
2	0.478261	0.478261	0.478261	0.434783	0.434783	0.434783	0.434783	0.434783
3	0.218182	0.145455	0.127273	0.109091	0.109091	0.109091	0.090909	0.090909
4	0.952381	0.952381	0.952381	0.952381	0.952381	0.952381	0.952381	1.000000
5	0.927536	0.811594	0.739130	0.724638	0.724638	0.724638	0.724638	0.724638

5 rows × 168 columns

In [4]:
```
print('Shape of loading dataframe:', np.shape(loading))
```

```
Shape of loading dataframe: (1189, 168)
```

- Each row represents a station. There are 1189 stations. Station 1 is EURYALE DEHAYNIN for example.
- Every column represents an hour of each day. Lun-00 is Monday at midnight for example. This is why there are 168 colums because 7 days a week * 24 hours a day = 168.
- The values of the data frame is the fullness of bikes of each station at a given hour. It is $\frac{numberofbikes}{numberofplaces}$ in each station. The value goes from 0 to 1. 0 indicates that a station is empty and 1 indicates that a station is full.

In [5]:
```
# Here we show the first five lines of coord
coord.head()
```

Out[5]:

	longitude	latitude	bonus	names
1	2.377389	48.886300	0	EURYALE DEHAYNIN
2	2.317591	48.890020	0	LEMERCIER
3	2.330447	48.850297	0	MEZIERES RENNES
4	2.271396	48.833734	0	FARMAN
5	2.366897	48.845887	0	QUAI DE LA RAPEE

In [6]:
```
print('Shape of loading dataframe:', np.shape(coord))
```

```
Shape of loading dataframe: (1189, 4)
```

- Each row represents a station. There are 1189 stations as in the first table.
- The column "names" tells us the name of a given station.
- Longlitude and Latitude reveals the location of each station.
- Bonus = 1 tells us if a given station is located on a hill. If not (bonus = 0), the station is on flat ground.

Next, we check if there is any missing data in the "loading" and "coord" tables:

In [7]:
```
loading_missing_value = loading.isna().sum().sort_values(ascending=False)

print('--- Loading ---')
#print(loading_missing_value)
print('Number of missing values = ', loading_missing_value.sum())

# --- #
print('')

coord_missing_value = coord.isna().sum().sort_values(ascending=False)

print('--- Coord ---')
#print(coord_missing_value)
print('Number of missing values = ', loading_missing_value.sum())
```

```
--- Loading ---
Number of missing values =  0

--- Coord ---
Number of missing values =  0
```

This quick verification confirms that there are no missing values in the data set. We continue by checking for duplicated data:

In [8]:
```
print("Are there no duplicates in Loading? : ",loading.duplicated().sum()==0
print("Are there no duplicates in Coord? : ",coord.duplicated().sum()==0)
```

```
Are there no duplicates in Loading? :  True
Are there no duplicates in Coord? :  True
```

Now that we know there are no duplicates, we will check if any stations are present more than once

In [9]:
```
station_names = coord.names.value_counts().sort_values(ascending=False)
print("Number of different named stations = ",len(station_names))
name = station_names.index[0]
coord[coord.names == name]
```

```
Number of different named stations =  1161
```

Out[9]:

	longitude	latitude	bonus	names
362	2.404770	48.876604	1	PORTE DES LILAS
450	2.405960	48.875412	1	PORTE DES LILAS
957	2.411046	48.878099	1	PORTE DES LILAS

We now see that 1189-1161=28 stations appear at least twice with the same name. Now, are they the same station or different ones just with the same name? To check, we calculate the mean loading value of the three different Porte de Lilas stations:

In [10]:
```
print("Mean loading value at Porte des lilas 1: ", pd.Series(loading.mean(ax
print("Mean loading value at Porte des lilas 2: ",pd.Series(loading.mean(axi
print("Mean loading value at Porte des lilas 3: ",pd.Series(loading.mean(axi
```

```
Mean loading value at Porte des lilas 1:  0.24002202538840497
Mean loading value at Porte des lilas 2:  0.24616506681724065
Mean loading value at Porte des lilas 3:  0.22638888888888853
```

The values are different, so we conclude that there are three different stations named Porte de Lilas, they are just close to each other.

In [11]:
```
import numpy as np
```

In [12]:
```
print('--- Average fill rate ---')
print(np.mean(loading.mean()))

print('--- Least crowded station, on average ---')
i = np.argmin(loading.mean(axis=1))
print("Index: ", i, ", Name:", coord.names[i])
print("Average: ",np.mean(loading, axis=1)[i])

print('--- Fullest station, on average ---')
j = np.argmax(loading.mean(axis=1))
print("Index: ", j, ", Name:", coord.names[j])
print("Average: ", np.mean(loading, axis=1)[j])
```

```
--- Average fill rate ---
0.3816217759807477
--- Least crowded station, on average ---
Index:  996 , Name:  LEO FROT ROQUETTE
Average:  0.5293096597444423
--- Fullest station, on average ---
Index:  1106 , Name:  CHATEAU ROUGE
Average:  0.26073908730158746
```

Before we start plotting graphs and whatnot we wanted to know general information about our stations. We easily find the average fill rate, the least and most crowded stations.

1.2. Visualization of the data

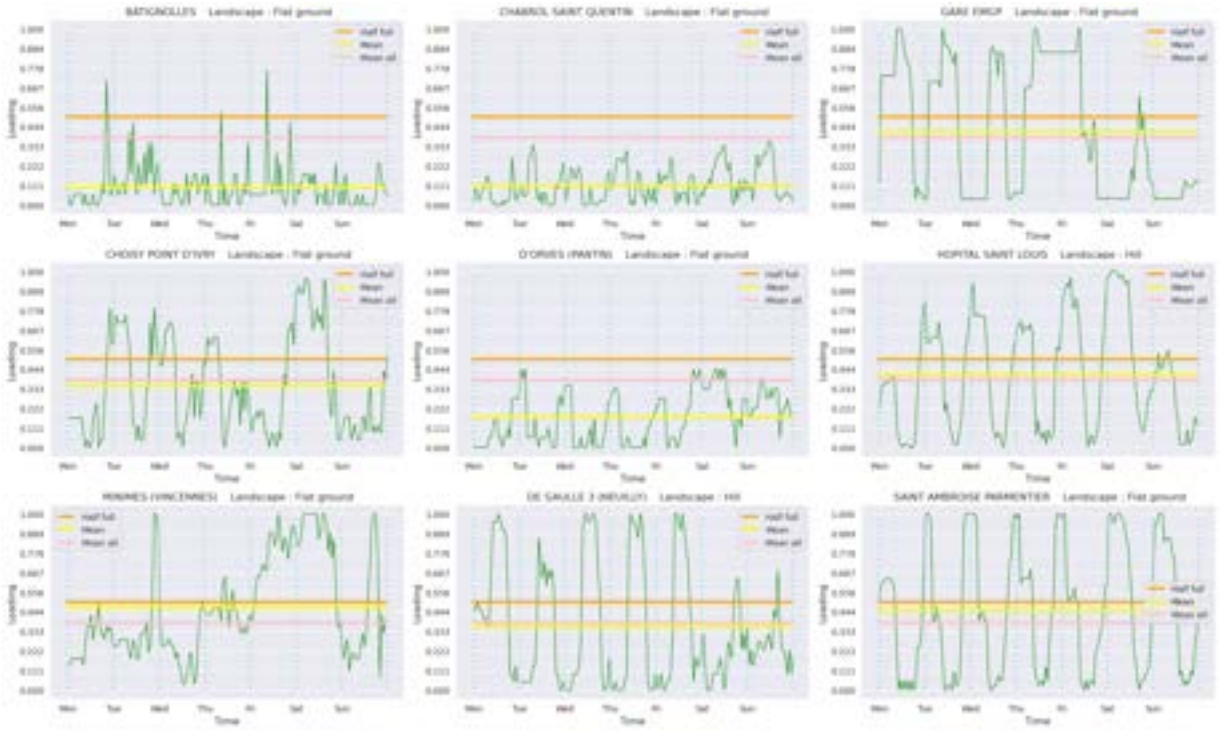
1.2.1. Loading of 9 random stations throughout the week

Firstly we wanted to visualize the loading of a station throughout the week. Since there are a lot of stations, we would have been unable to visualize all of them, so we are choosing nine stations randomly, as not to add any bias into our study.

- The vertical dotted lines separate the different days of the week.
- The orange line is positionated at 0.5 (medium fullness).
- The yellow line shows the mean fullness of the station.
- The pink line shows the mean fullness of all stations.

In [13]:
```
stations = np.arange(loading.shape[0])
#rd.shuffles randomizes the order of the stations
rd.shuffle(stations)
stations = stations[:9]
loading_data = loading.to_numpy()
def landscape(i):
    if coord.bonus[i] == 1:
        return "Hill"
    else:
        return "Flat ground"

# Define days of the week
days_of_week = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', '']
n_steps=loading.shape[1]
time_range= np.linspace(1, n_steps, n_steps)
time_tick= np.linspace(1, n_steps, 8)

fig, axs = plt.subplots(3,3,figsize=(20,12))

mean_all_satations= loading.mean().mean()
for i in range(3):
    for j in range(3):
        k_station = rd.randrange(loading.shape[0])
        mean_station= loading.iloc[k_station,:].mean()
        axs[i, j].plot(time_range, loading_data[k_station, :], linewidth = 1
        axs[i, j].set_xticks(time_tick, days_of_week, fontsize=15)
        axs[i, j].set_yticks(np.linspace(0,1,10))
        axs[i, j].set_title(coord.names[1 + k_station]+ '    Landscape : ' +
        axs[i, j].vlines(x = time_tick, ymin = 0, ymax = 1, colors = "lightb
        axs[i, j].hlines(y = 0.5, colors = "orange", xmin=0, xmax=168, linew
        axs[i, j].hlines(y = mean_station, colors = "yellow", xmin=0, xmax=1
        axs[i, j].hlines(y = mean_all_satations, colors = "pink", xmin=0, xm

for ax in axs.flat:
    ax.set_xlabel('Time', fontsize = 12)
    ax.set_ylabel('Loading',fontsize = 12)
    ax.tick_params(axis='x', labelsize=10)
    ax.tick_params(axis='y', labelsize=10)
    ax.legend()

plt.tight_layout()
```

```
plt.show()
```



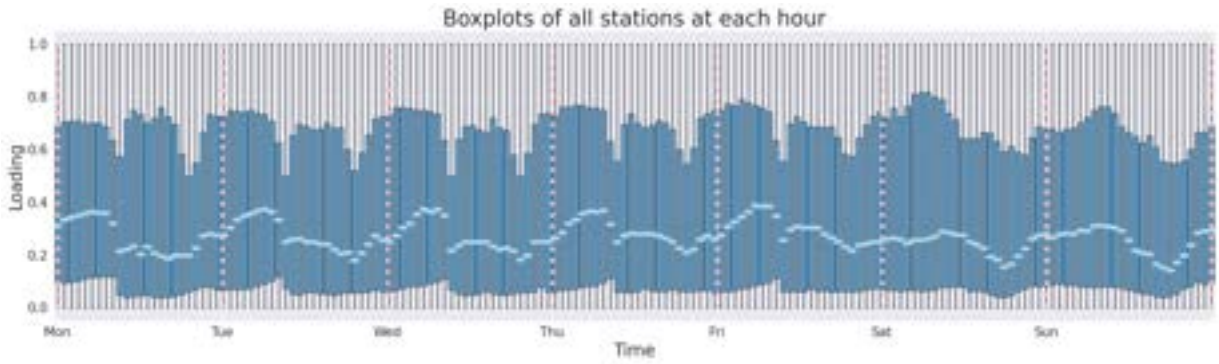
We can already see different trends. We observe that most stations have a periodic variation of fullness. Some stations are always empty during the night and some during the day. Some stations are more empty (or fuller) during the weekdays or the week-ends. It would be interesting to see later on if some of them are in a business district or leisure areas.

1.2.2. Boxplots of all stations at each hour

The following code generates a boxplot visualization to illustrate the loading data across different stations over time. The data is represented by boxes, with the width of each box indicating the range of loading values. The medians of the data are highlighted with thicker lines in light blue color. Additionally, vertical pink dotted lines are drawn every 24 hours for clarity.

In [14]:
```
plt.figure(figsize = (20,6))
bp = plt.boxplot(loading_data, widths = 0.75, patch_artist = True)
for box in bp['boxes']:
    box.set_alpha(0.7)

for median in bp['medians']:
    median.set(color = "lightblue", linewidth=5)

plt.vlines(x = time_tick, ymin = 0, ymax = 1,
           colors = "pink", linestyle = "dotted", linewidth = 5)


plt.xlabel('Time', fontsize = 20)
plt.ylabel('Loading', fontsize = 20)
```

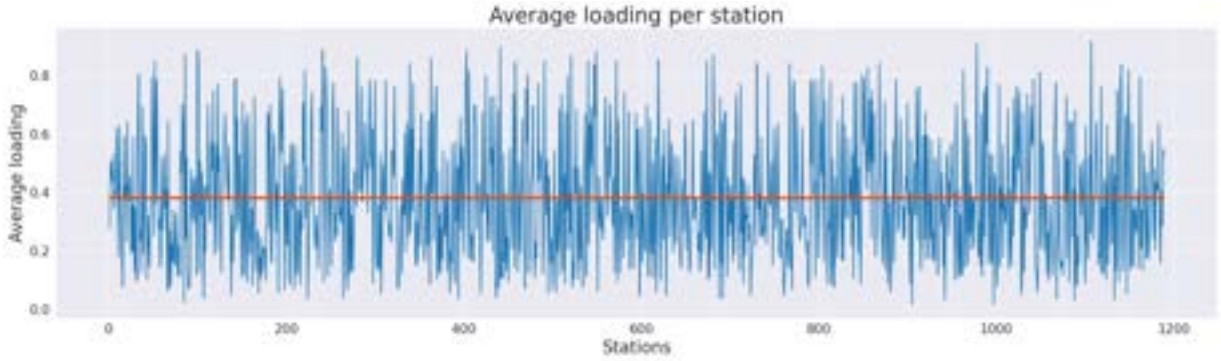
```
plt.title("Boxplots of all stations at each hour", fontsize = 25)
plt.xticks(time_tick, days_of_week, fontsize=15)
plt.yticks(fontsize = 15)

plt.tight_layout()
plt.show()
```



We can clearly see a pattern throughout the weekdays, and similarities on Sundays ans Saturdays, where bikes are more used in the afternoon/evening. We observe that from Monday to Friday, the average fill rate follows almost exactly the same trend. It goes higher during the morning until around 08:00 in the morning then it drops to a minimum around 19:00 in the evening.

1.2.3. Average station fill rate

In [15]:
```
loading_mean = pd.Series(loading.mean(axis=1))
n_stations = loading.shape[0]  # number of observed stations
stations   = np.arange(n_stations)

plt.figure(figsize = (20,6))

plt.plot(loading_mean)
plt.hlines(y = loading.mean().mean(), xmin=0, xmax=n_stations,
           colors = "OrangeRed", linewidth = 3)

plt.xlabel('Stations', fontsize = 20)
plt.ylabel('Average loading', fontsize = 20)
plt.title("Average loading per station", fontsize = 25)
plt.xticks(fontsize = 15)
plt.yticks(fontsize = 15)

plt.tight_layout()
plt.show()
```

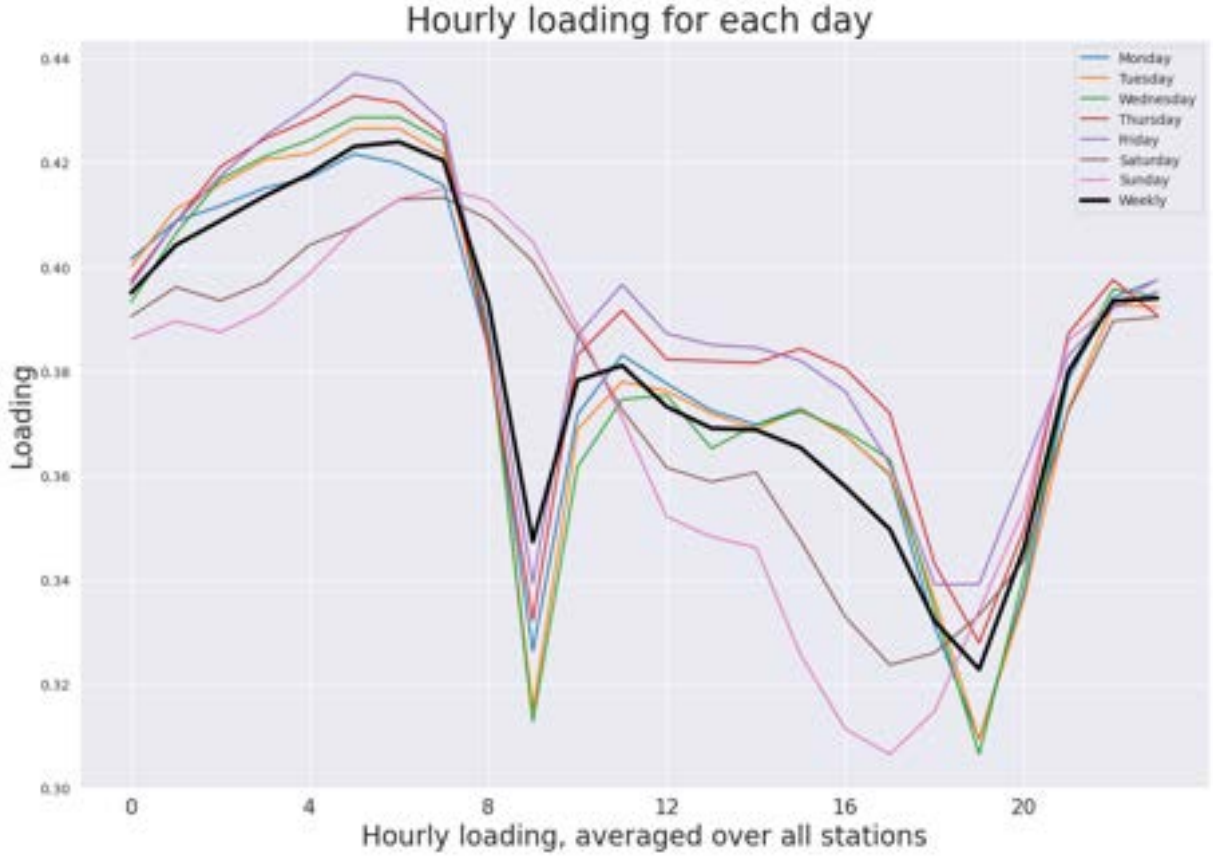


Here we plotted the evolution of the average load for each station in blue and the average loading for the entire data set in orange. We notice that the average loading varies greatly from station to station.

1.2.3. Hourly loading for each day

Next we decided to visualize the average hourly loading (AHL) for each day (in colour), as well as the average hourly rate for the week (in black).

In [16]:
```
mean_per_hour_per_day = loading.mean(axis = 0).to_numpy()
mean_per_hour_per_day = mean_per_hour_per_day.reshape((7, 24))

mean_per_hour = mean_per_hour_per_day.mean(axis=0)

days = ["Monday", "Tuesday", "Wednesday","Thursday", "Friday", "Saturday", "
plt.figure(figsize = (15,10))

plt.plot(mean_per_hour_per_day.transpose())
plt.plot(mean_per_hour, color = "black", linewidth = 3)

plt.xlabel('Hourly loading, averaged over all stations', fontsize = 20)
plt.ylabel('Loading', fontsize = 20)
plt.title("Hourly loading for each day ", fontsize = 25)
plt.legend(days + ['Weekly'])
plt.xticks(ticks = np.arange(0,24,4), labels=np.arange(0,24,4), fontsize = 1

plt.tight_layout
plt.show()
```



This plot confirms our previous observations. The loading varies periodically per day, as we saw in the boxplots.

The first thing we notice is that the Average Hourly Loading (AHL) has a similar behaviour on the week days on one hand and on the week ends on the other. Durind the week the stations are full around 6am and they start to get empty a few hours later, the emptiest hour being 10 am. This makes sense as it is the times that people would tend to go to work. The stations are also emptier on average around 8pm, which would correspond to the time that people go home from work.

On the week ends however the stations are slightly less full between 00am and 7am which would suggest that people go out on Friday and Saturday night. They are at their fullest around 10am which is completely different from their state during the week. They are emptiest between 12pm and 8pm, whith the lowest point being reached around 6pm. This would suggest that people tend to stay home in the morning and only start going out during the afternoon. Or maybe when they go out during the week end they favour other means of transportation than the velibs.

1.3. Visualization of the data on a map of Paris

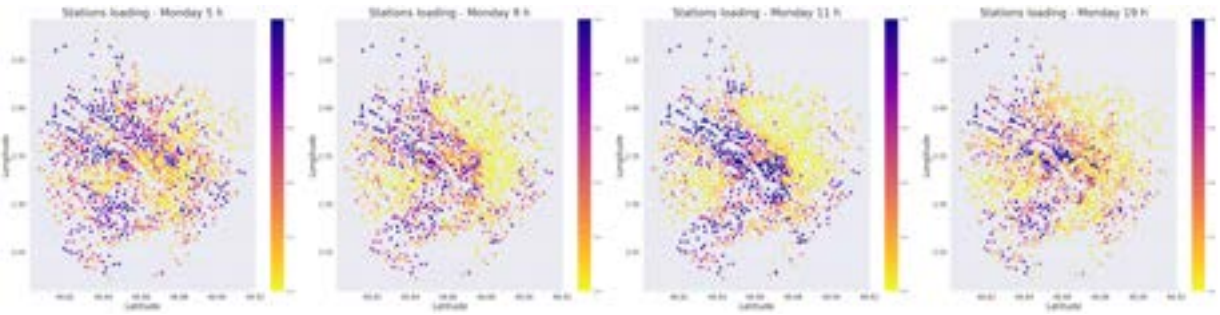
1.3.1. Stations loading on Monday

The next plot shows the velib stations placed on a map, at four different hours on a

Monday. Those colored as yellow are the empty ones and those in purple are the full ones.

In [17]:
```
import matplotlib.cm as cm
import matplotlib.patches as mpatches
import plotly.express as px
```

In [18]:
```
hours = [5, 9, 11, 19]

s, n = 10, len(hours)
fig, axs = plt.subplots(1, n, figsize = (s*n, s))

for (i,h) in enumerate(hours):
    im = axs[i].scatter(coord.latitude, coord.longitude, c = loading_data[:,
    axs[i].set_title('Stations loading - Monday {} h'.format(h), fontsize =
    plt.colorbar(im, ax=axs[i])

for ax in axs.flat:
    ax.set_xlabel('Latitude', fontsize = 20)
    ax.set_ylabel('Longitude', fontsize = 20)
    ax.tick_params(axis='x', labelsize=15)
    ax.tick_params(axis='y', labelsize=15)

plt.tight_layout()
plt.show()
```



Here we have plotted the average stations loading on Monday at four different hours : 5am, 9am, 11am and 7pm. These hours correspond to the times where the stations are fullest (5am and 11am) and emptiest (9am and 7pm) according to the previous graph we analysed.

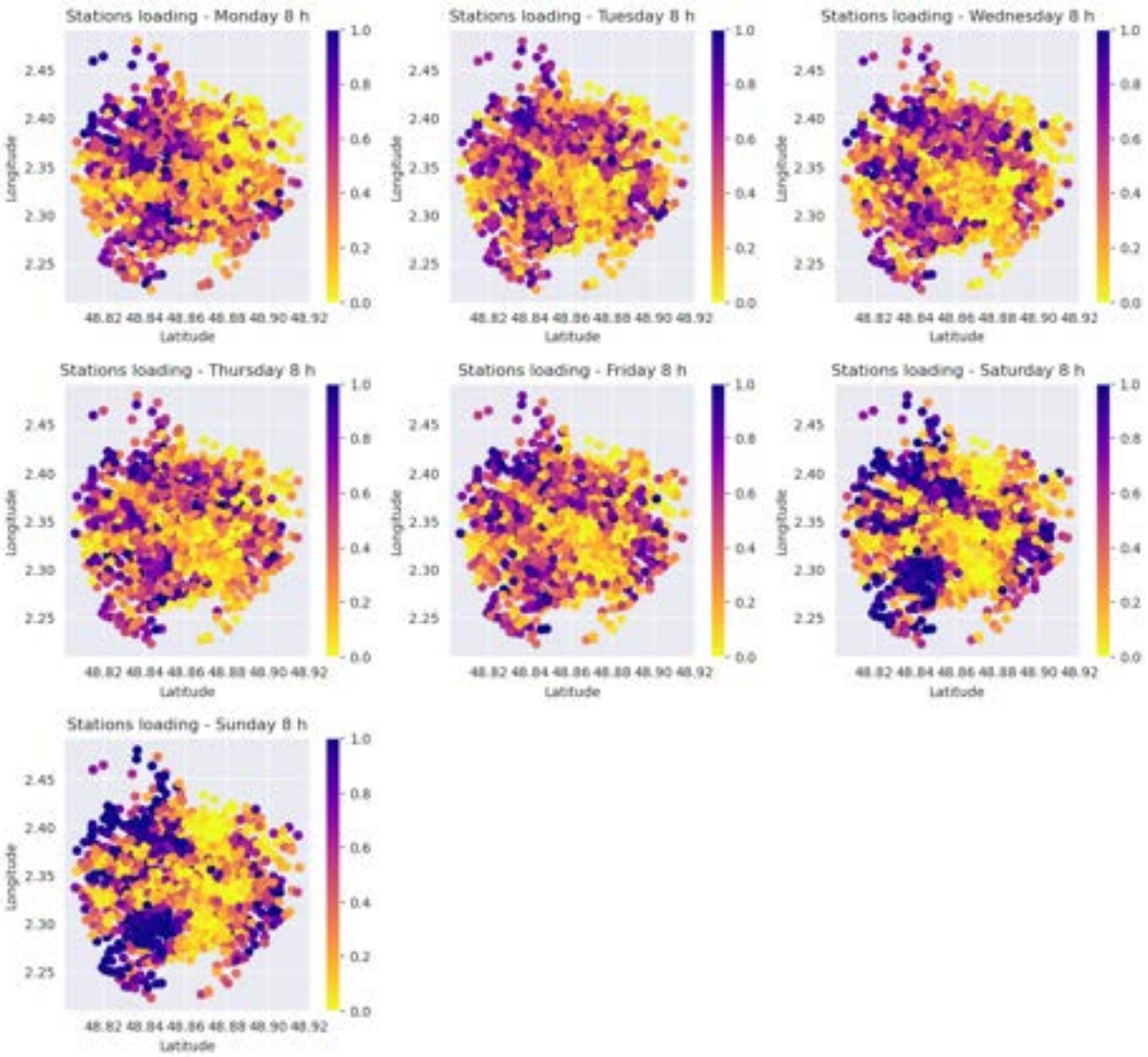
We observe that at 5am the stations in the city center are empty and those in the suburb are full. We can interpret this as people being at home before work. On the other hand, at 11h on a Monday, the stations in the suburb are mostly empty and those in the city center are full. We could think this is due to work shifts. Most of the jobs are in the city center and their shifts start before 11h, meaning that people move from the suburbs to the city center.

We can see that the fullest stations tend to always be in the city center. This could be explained by the fact that there is more movement in the city center and that there are more velibs there than in the suburbs.

1.3.2. Stations loading at 8 am

The next plot shows the velib stations placed on a map, at 8 o'clock on all different days of the week. As the last plot, those colored as yellow are the empty ones and those in purple are the full ones.

In [19]:
```
h = 8
hours = np.arange(h, 168, 24)

load_per_hour = loading_data[:, hours]

days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",

# --- #

s, m = 10, 3
k = 1 + len(days)//m

fig = plt.figure(figsize=(s+1, s))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=.3, wspace=.25)

for (i,d) in enumerate(days):
    ax = fig.add_subplot(k, m, i+1)
    im = ax.scatter(coord.latitude, coord.longitude, c = load_per_hour[:,i],
    plt.colorbar(im)

    ax.set_title('Stations loading - ' + d + ' {} h'.format(h))
    ax.set_xlabel('Latitude')
    ax.set_ylabel('Longitude')
    ax.tick_params(axis='x')
    ax.tick_params(axis='y')

#plt.tight_layout()
plt.show()
```

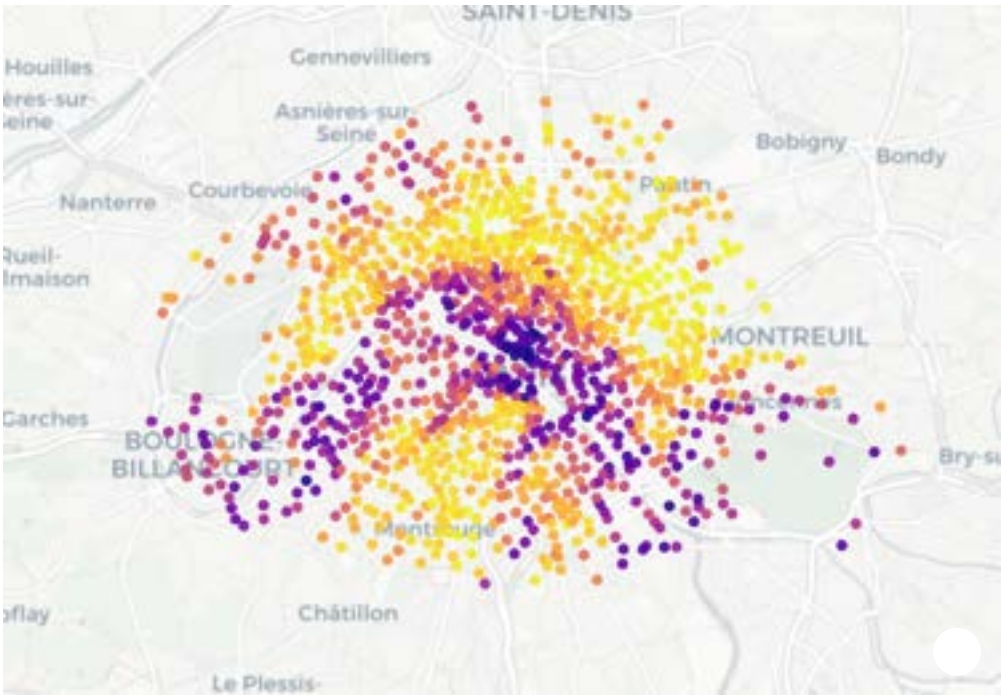


In Paris the neighbourhoods are referred to as arrondissements. We note that at 8 o'clock almost everyday the stations in IV and V arrondissements are empty.

1.3.3. Average loading of all stations at 6 pm

In [20]:
```
h = 18
hours = np.arange(h, 168, 24)
load_per_hour = loading_data[:, hours].mean(axis=1)

# --- #

fig = px.scatter_mapbox(coord, lat = 'latitude', lon = 'longitude',
                        mapbox_style = "carto-positron",
                        color = load_per_hour, color_continuous_scale = px.c
                        zoom  = 10, opacity = .9,
                        title = 'Stations loading - Weekly average at {} h'.

fig.show()
```

Stations loading - Weekly average at 18 h



We remark that in average, at 6 pm, the stations in the suburbs are empty, meaning that people are at the city center.

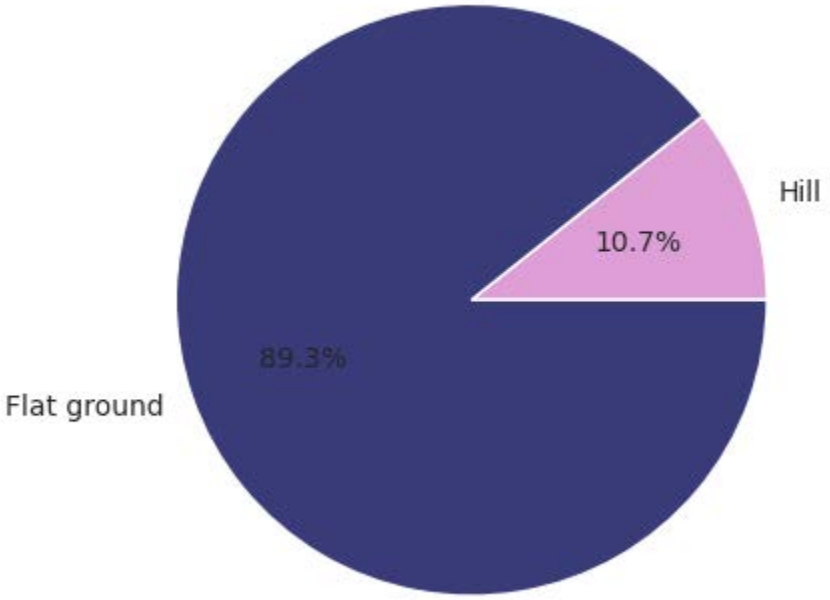
1.4 Influence of Altitude Difference on Station Loading

1.4.1. Pie chart of type of ground and map with stations colored by thir placement

The next pie chart shows the percentage of stations placed on flat ground and on a hill.

In [21]:
```
loading_hill   = loading_data[coord.bonus == 1]
loading_valley = loading_data[coord.bonus == 0]

# --- #

size = [len(loading_hill), len(loading_valley)]
labels = ['Hill', 'Flat ground']

plt.pie(size, labels = labels, autopct="%1.1f%%",
        colors = [sns.color_palette('tab20b')[-1],sns.color_palette('tab20b'
```

```
plt.show()
```

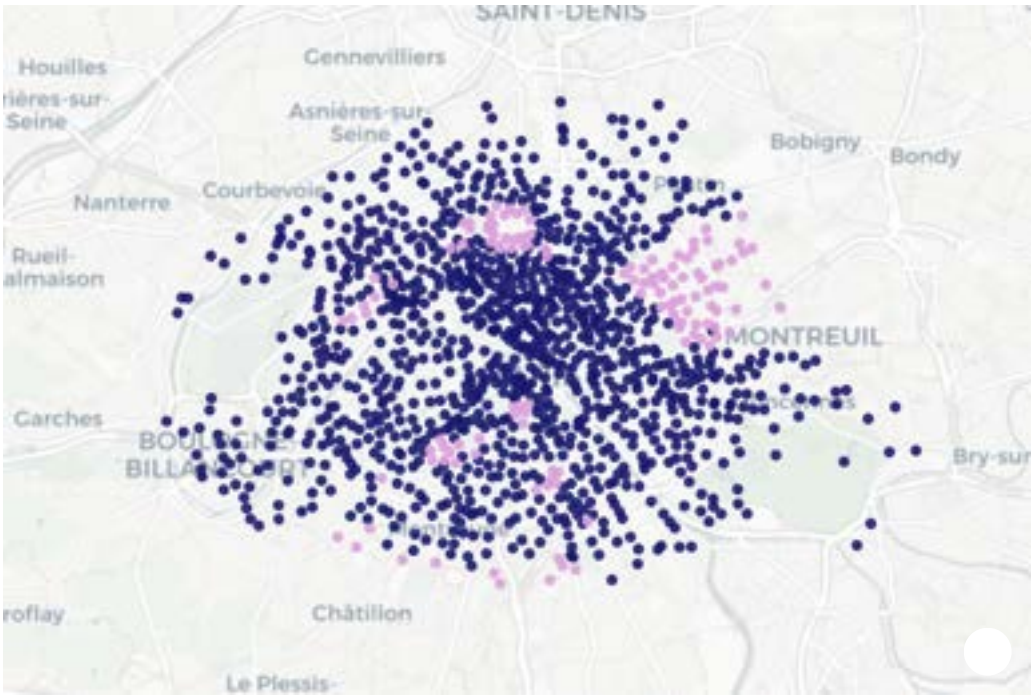


We note that the majority of the stations, 89.3%, are placed on flat ground. We can deduct that this fact is due to costumer convenience, as riding on flat ground is easier.

The next fonction shows the stations colored by thir placement. Those in pink are placed on a hill and those in purple are placed on flat ground.

In [22]:
```
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# --- #

fig = px.scatter_mapbox(coord, lat = 'latitude', lon = 'longitude',
                        mapbox_style = "carto-positron",
                        color = 'hill',
                        color_discrete_map = {0:'midnightblue', 1:'plum'},
                        labels = {0: "Flat ground", 1: "Hill"},
                        zoom = 10, opacity = .9,
                        title = 'Hilltop stations')


fig.show()
```

Hilltop stations



We note two big hill zones: Montmartre and Réservoir de Belleville.

1.4.2. Average fullnes of stations on hills and flat ground

In [23]:
```
# Combine loading and bonus columns from coord DataFrame
loading_hill = loading.copy()
loading_hill['hill'] = coord['bonus']

hilltop_average = 0
flat_average = 0
compteur_hill = 0
compteur_flat = 0

# Iterate over each row in loading_hill DataFrame
for i in range(len(loading_hill)):
    if loading_hill.iloc[i, -1] == 1:  # Check if the station is on a hill
        hilltop_average += loading.iloc[i].mean()  # Add the mean of the loa
        compteur_hill += 1  # Increment hilltop counter
    else:
        flat_average += loading.iloc[i].mean()  # Add the mean of the loadin
        compteur_flat += 1  # Increment flat ground counter
```

```
# Calculate the average fullness for hilltop and flat ground stations
hilltop_average /= compteur_hill
flat_average /= compteur_flat

print(f'The average fullness on hilltop stations is {hilltop_average}, where
```

```
The average fullness on hilltop stations is 0.16104660259524073, whereas the
average fullness of flat ground stations is 0.4079994097095229
```

The stations that are located on hill are on average emptier that the ones on flat ground. This could be explained by two factors:

1. There are less stations on hills than on flat ground so the average is smaller.
2. It is more tiresome to go up a hill with a bike than to go with another means of transportation. On the other hand it is very easy to go down a hill with a bike. This could mean that people who take velibs down a hill do not bother to take them back up and simply store them elsewhere.

Part 2 : Principal Component Analysis (PCA)

We are now going to continue the exploration of the data by doing a Principal Component Analysis.

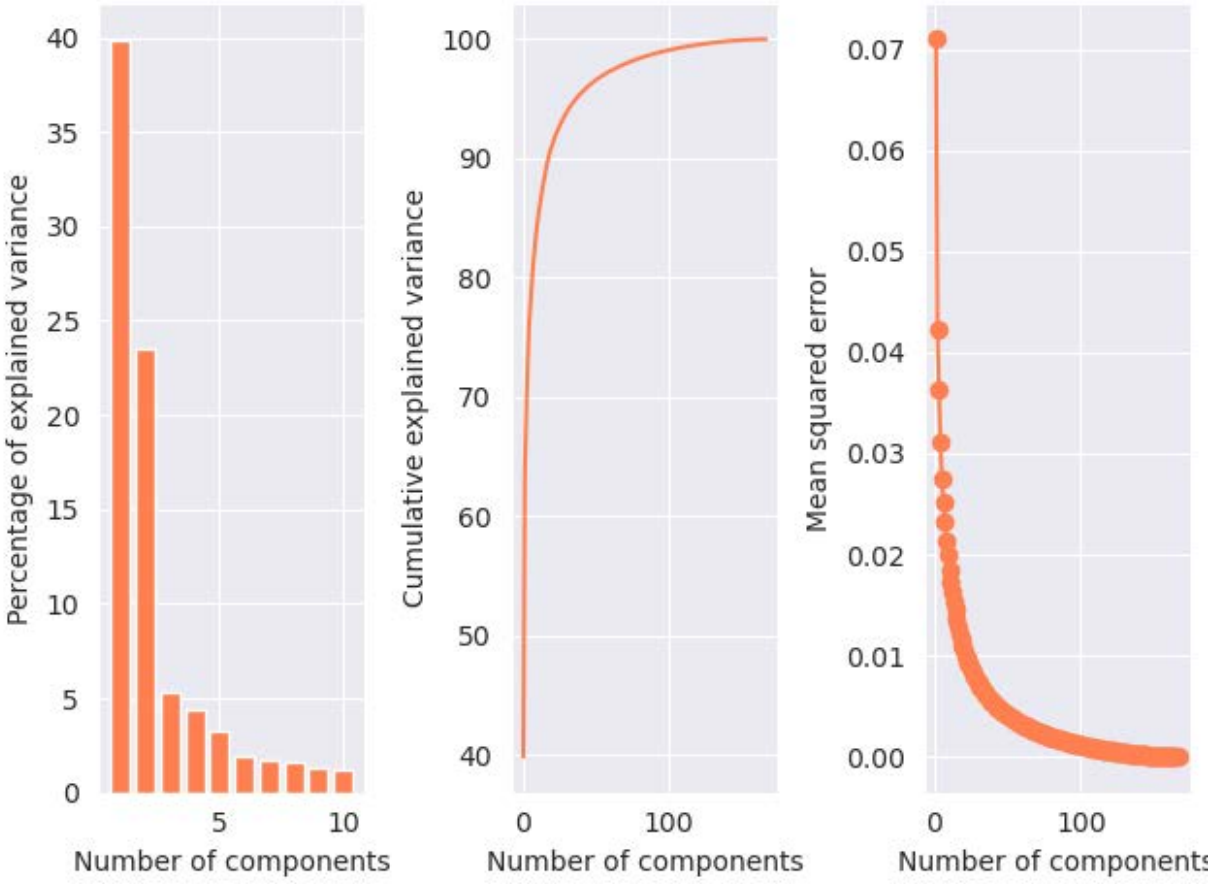
2.1. How many components on the PCA should we keep?

In [24]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

In [25]:
```
ss = StandardScaler()
loading_scaled = ss.fit_transform(loading)
pca = PCA()
loading_pca = pca.fit_transform(loading_scaled)
```

In [26]:
```
explained_variance_ratio = 100*pca.explained_variance_ratio_

loading_array = loading.values
n = loading_array.shape[1]
MSE = np.zeros(n-1)

for d in range(n-1):
    pca = PCA(d+1)  # project from 64 to 2 dimensions
    projected = pca.fit_transform(loading_array)
    reconstructed = pca.inverse_transform(projected)
    MSE[d] = np.mean((loading_array - reconstructed)** 2)


plt.subplot(1,3,1)
n_bars = 10
```

```
plt.bar(np.arange(1, n_bars+1), explained_variance_ratio[:n_bars], color='co
plt.xlabel("Number of components")
plt.ylabel("Percentage of explained variance")

plt.subplot(1,3,2)
plt.plot(np.cumsum(explained_variance_ratio), color='coral')
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance');

plt.subplot(1,3,3)
plt.plot(np.arange(1,n), MSE, marker='o', color='coral')
plt.xlabel('Number of components')
plt.ylabel('Mean squared error');

plt.tight_layout()
plt.show()
```

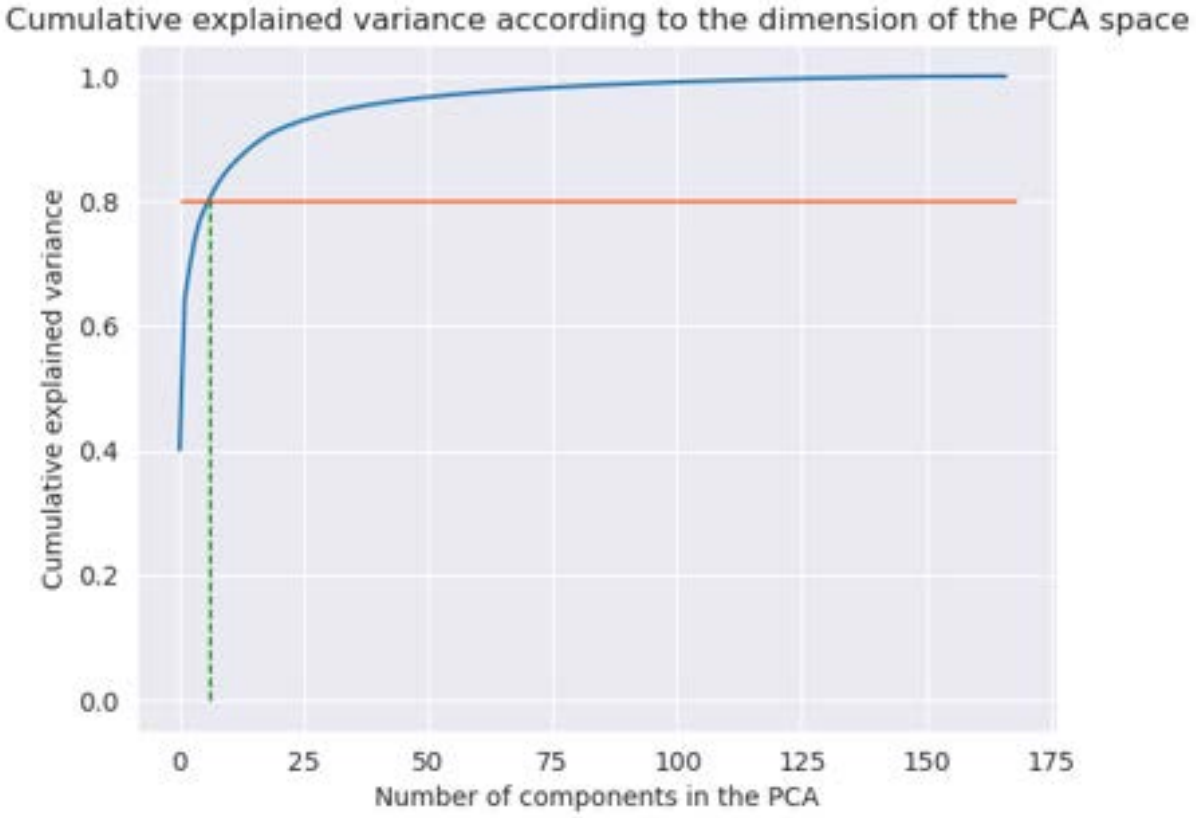


In [27]:

```
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
# Find the index where cumulative variance exceeds or reaches 0.8
index_08 = np.argmax(cumulative_variance > 0.8)


plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.hlines(y = 0.8, xmin=0, xmax=168, colors = "OrangeRed", linewidth = 1)
plt.vlines(x=index_08, ymin=0, ymax=0.8, colors="green", linestyle="--", lin

plt.title('Cumulative explained variance according to the dimension of the P
plt.xlabel('Number of components in the PCA')
```

```
plt.ylabel('Cumulative explained variance');
print("Number of components needed to explain 80% of the variance:",index_08
```

```
Number of components needed to explain 80% of the variance: 6
```



In [28]:
```
pca = PCA(0.8).fit(loading_scaled)
pca.n_components_
print("Number of components needed to explain 80% of the variance:",pca.n_co
```

```
Number of components needed to explain 80% of the variance: 8
```

In [29]:
```
print('--- Explained variance ---')
for k in range(1,8):
    percentage=round(sum([explained_variance_ratio[i] for i in range(k)]),2)
    print('Component', k, ' | eigenvalue :', round(pca.explained_variance_[k
```

```
--- Explained variance ---
Component 1  | eigenvalue : 66.95  | percentage of variance : 39.81  | cumul
ative % variance:  39.81
Component 2  | eigenvalue : 39.52  | percentage of variance : 23.5  | cumula
tive % variance:  63.32
Component 3  | eigenvalue : 8.88  | percentage of variance : 5.28  | cumulat
ive % variance:  68.6
Component 4  | eigenvalue : 7.31  | percentage of variance : 4.35  | cumulat
ive % variance:  72.95
Component 5  | eigenvalue : 5.51  | percentage of variance : 3.28  | cumulat
ive % variance:  76.22
Component 6  | eigenvalue : 3.25  | percentage of variance : 1.93  | cumulat
ive % variance:  78.15
Component 7  | eigenvalue : 2.83  | percentage of variance : 1.68  | cumulat
ive % variance:  79.84
```

We will restrict ourselves to the first 6 principal components to explain 78% of the variance, given that there is only 3.28% gain if we keep 8 components compared to 6.

In addition, we will keep in mind that the first two components have a higher percentage of variance than the rest of components, we will use them later for some plots.

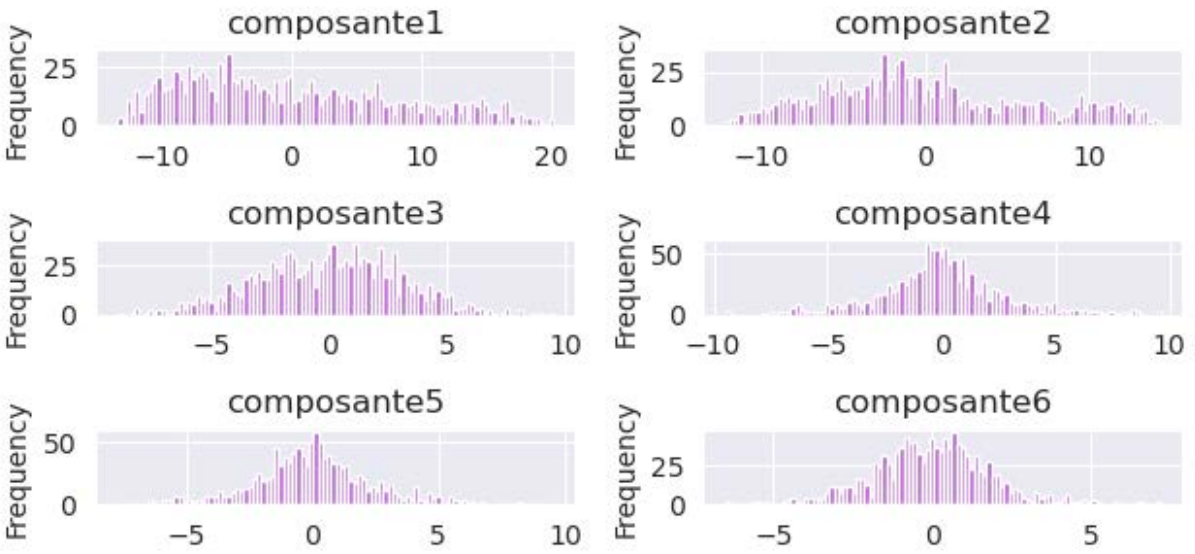
Let's call 'loading_pca6' the new dataset containing the loading of all 1189 stations and composed by the fisrt 6 dimensions of the pca.

In [30]:
```
loading_pca6 = loading_pca[:,:6]
loading_pca6.shape
```

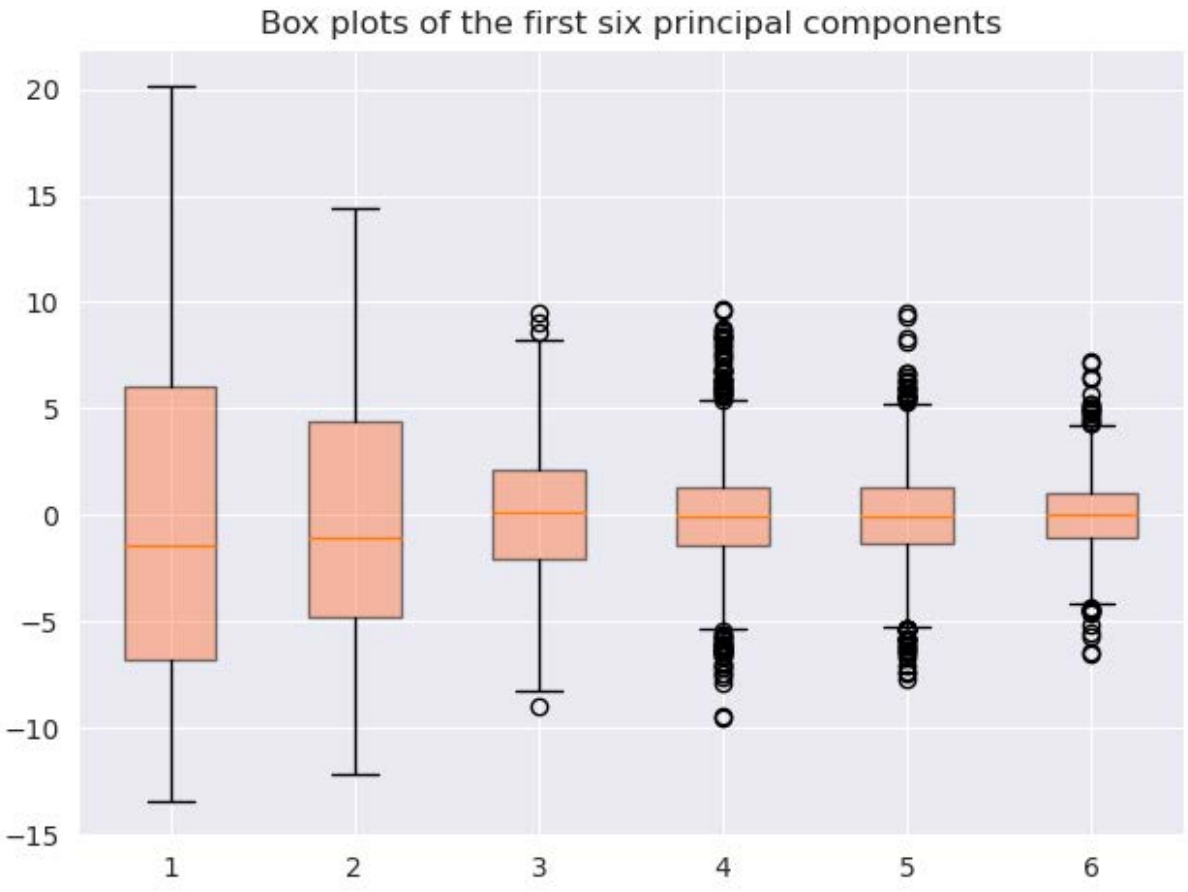
Out[30]: (1189, 6)

2.2. Visualization the dispersion of the observations on the 6-th first components of the PCA.

In [31]:
```
for i in range(6):
    plt.subplot(5, 2, i+1)
    plt.hist(loading_pca[:,i], bins = 100, color="mediumorchid")
    plt.title("composante%i" % (i+1))
    plt.ylabel("Frequency")

plt.tight_layout()
plt.show()
```



In [32]:
```
box = plt.boxplot(loading_pca[:,:6], patch_artist=True)
plt.setp(box["boxes"], facecolor="coral", alpha=.5)
plt.title("Box plots of the first six principal components")
plt.tight_layout()
plt.show()
```



We observe that the three first pca dimensions have a larger variance than the others.

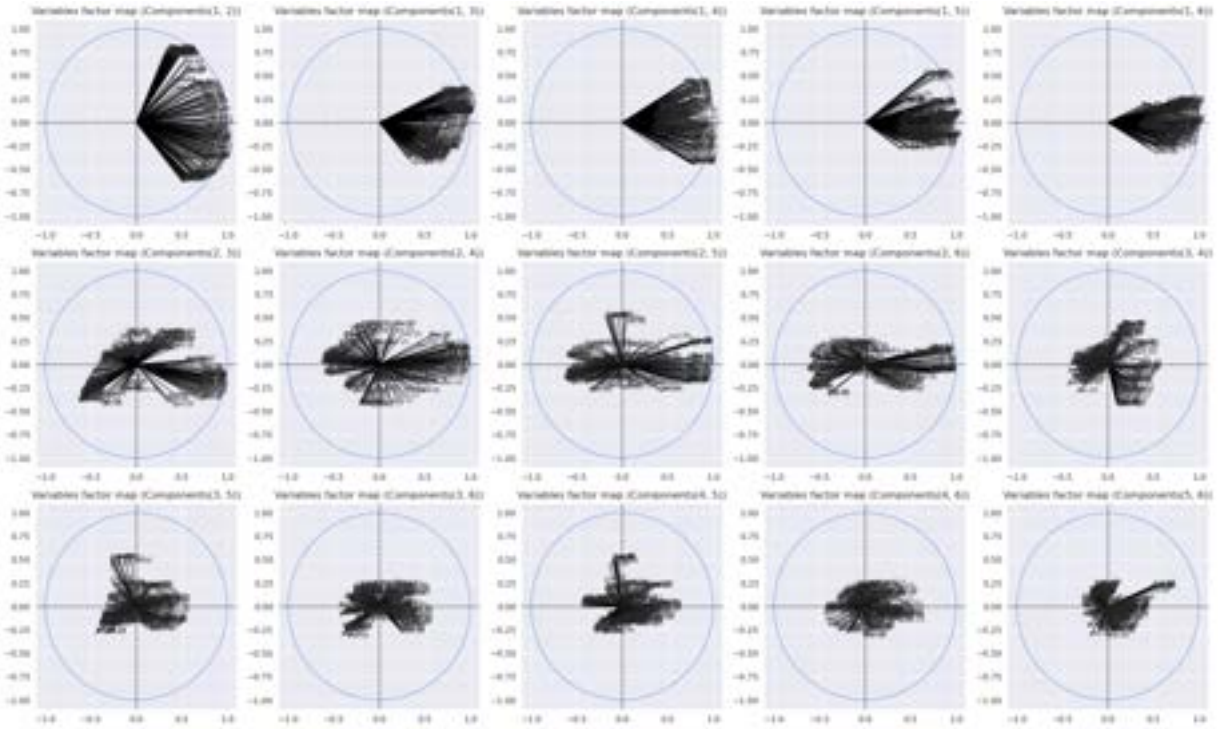
2.3. Variable factor maps

2.3.1. Variable Factor Maps for the first 6 principal components

The next plot visualizes the Variable Factor Map for each combination of the first 6 principal components obtained from Principal Component Analysis (PCA). This visualization depicts the factor map showcasing the relationship between variables in a dataset. Each point represents a variable, and its position is determined by its contributions to the principal components. The arrows indicate the direction and magnitude of these contributions.

In [33]:
```
# Calculate the coordinates of the variables on the factor map for the first
coords = []
components = []
for a in range(6):
    for b in range(a + 1, 6):
        coord1 = pca.components_[a] * np.sqrt(pca.explained_variance_[a])
        coord2 = pca.components_[b] * np.sqrt(pca.explained_variance_[b])
        coords.append((coord1, coord2))
        components.append((a+1, b+1))

# Plot Variable factor map for each combination of the first 6 principal com
```

```
fig, axs = plt.subplots(3, 5, figsize=(20, 12))
for k, (coord1, coord2) in enumerate(coords):
    row = k // 5
    col = k % 5
    ax = axs[row, col]
    for i, j, nom in zip(coord1, coord2, loading.columns):
        ax.text(i, j, nom, fontsize=7)
        ax.arrow(0, 0, i, j, color='black', alpha=0.7, width=0.0001)

    ax.axis((-1.1, 1.1, -1.1, 1.1))
    ax.hlines(y=0, xmin=-1.1, xmax=1.1, colors="black", linestyle="--", line
    ax.vlines(x=0, ymin=-1.1, ymax=1.1, colors="black", linestyle="--", line
    ax.add_artist(plt.Circle((0, 0), radius=1, color='cornflowerblue', fill=

    ax.set_title(f'Variables factor map (Components{components[k]})')
    ax.grid(True)
plt.tight_layout()
plt.show()
```



We notice that the variables are very well represented on the first few components and less so on the others. That was expected by construction of the PCA. Another thing we notice is that the variables are only represented on the positive part of the first component. To try and understand that better we decided to color the variables with different factors.

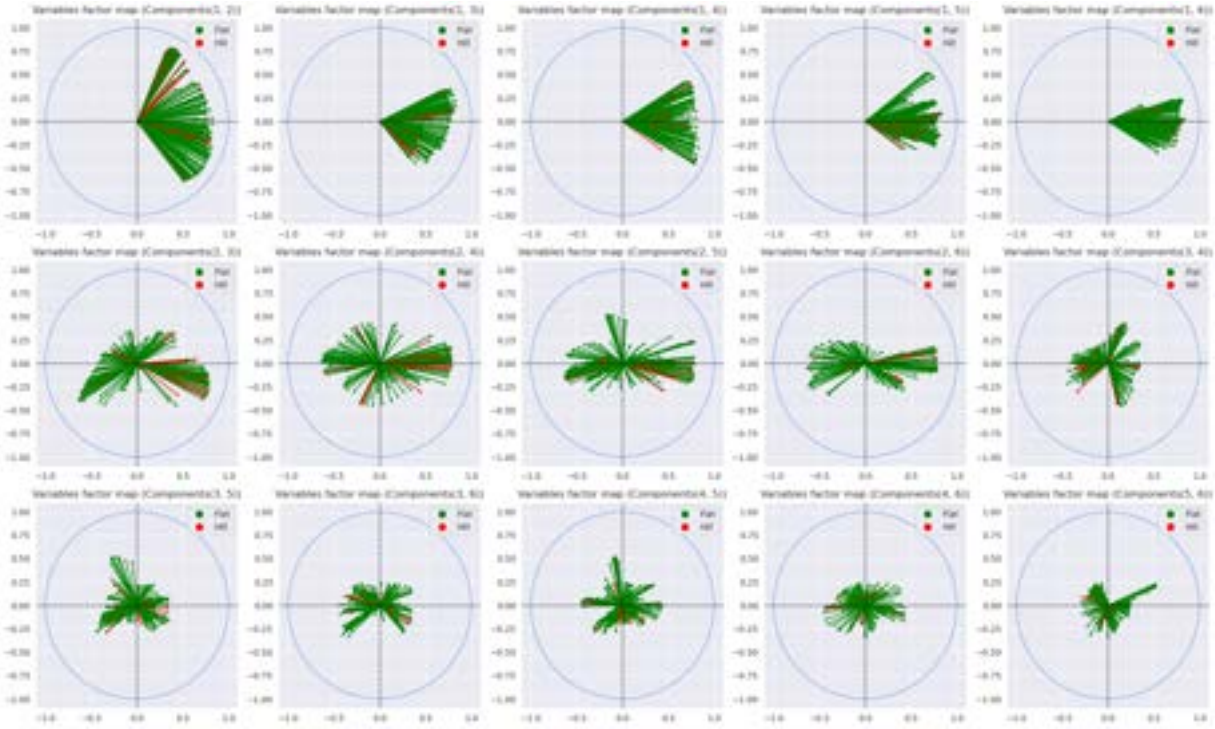
2.3.2. Variable Factor Map labeled by hill position

In this specific plot, we examine the Variable Factor Map for each combination of the first 6 principal components, with the color of each point denoting a binary feature: red for stations located on hills and green for stations situated on flat ground. This allows us

to discern any discernible patterns or clusters based on terrain elevation.

In [34]:
```
from matplotlib.lines import Line2D
from matplotlib.patches import Patch
```

In [35]:
```
coords = []
components = []
for a in range(6):
    for b in range(a + 1, 6):
        coord1 = pca.components_[a] * np.sqrt(pca.explained_variance_[a])
        coord2 = pca.components_[b] * np.sqrt(pca.explained_variance_[b])
        coords.append((coord1, coord2))
        components.append((a+1, b+1))

# Plot Variable factor map for each combination of the first 6 principal com
fig, axs = plt.subplots(3, 5, figsize=(20, 12))
for k, (coord1, coord2) in enumerate(coords):
    row = k // 5
    col = k % 5
    ax = axs[row, col]
    for i, j, nom in zip(coord1, coord2, coord["bonus"]):
        color = "red" if nom else "green"
        ax.arrow(0,0,i,j,color=color)
        ax.plot(i, j, "o", color=color, markersize=2)

    ax.axis((-1.1, 1.1, -1.1, 1.1))
    ax.hlines(y=0, xmin=-1.1, xmax=1.1, colors="black", linestyle="--", line
    ax.vlines(x=0, ymin=-1.1, ymax=1.1, colors="black", linestyle="--", line
    ax.add_artist(plt.Circle((0, 0), radius=1, color='cornflowerblue', fill=

    ax.set_title(f'Variables factor map (Components{components[k]})')
    ax.grid(True)

    # Create custom legend with colors
    labels = ['Flat','Hill']
    legend_elements = [Line2D([0], [0], marker='o', color='w', markerfacecol
                       Line2D([0], [0], marker='o', color='w', markerfacecol

    ax.legend(handles=legend_elements, labels=labels)

plt.tight_layout()
plt.show()
```



We cannot deduce anything labeling by hill position. After further reflection we realised that coloring a variable (ie. an hour of the week) depending on wheather or not it is on a hill is not pertinent.

Let's study the variable factor labeled by day and night hours.

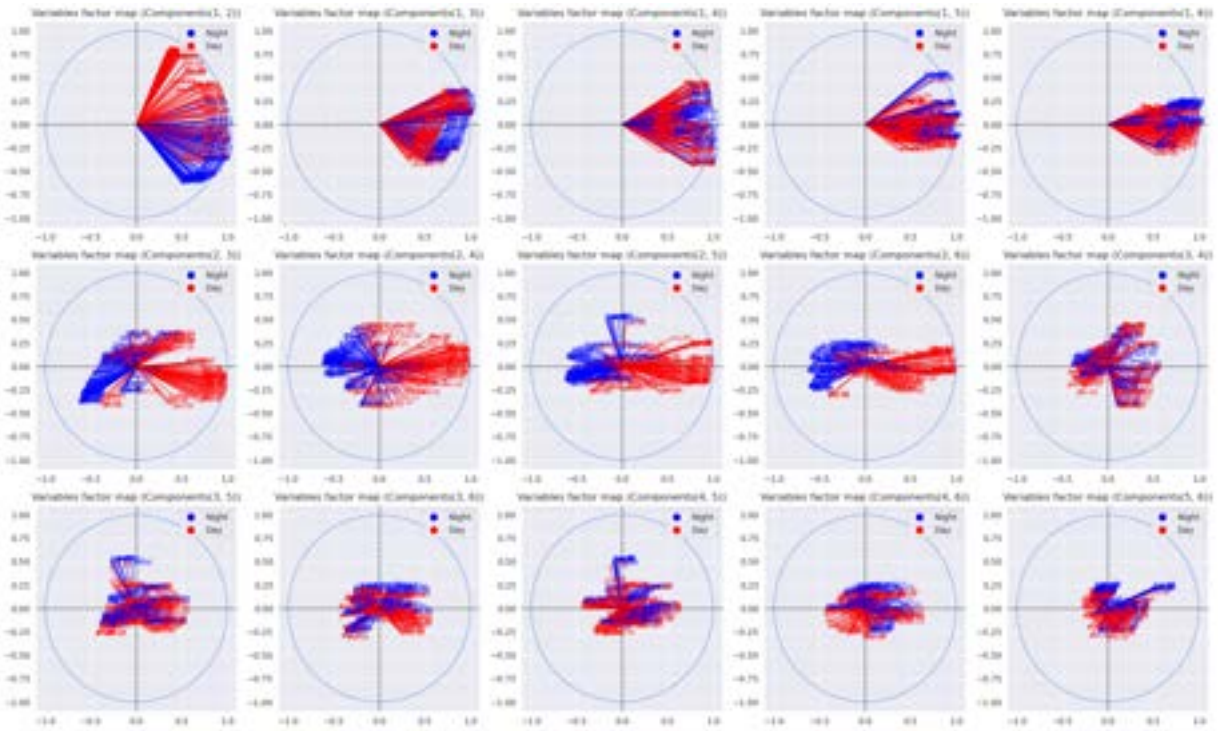
2.3.3. Variable Factor Map labeled by day and night hours

This plot illustrates the Variable factor map derived from Principal Component Analysis (PCA), showcasing the distribution of variables (stations) along the two primary components. What distinguishes this plot is the color differentiation for lines corresponding to day and night hours. We will considere from 10 pm until 8 am the night hours.

Red lines indicate stations full during the day, while blue lines represent those full during the night. This visualization offers insights into the temporal patterns of station activity across the dataset.

In [36]:
```
# Calculate whether each hour corresponds to day or night
hour_of_day = np.arange(168) % 24
is_night = ((hour_of_day >= 22) | (hour_of_day < 8)).astype(int)  # 1 for ni

# Calculate the coordinates of the variables on the factor map for the first
coords = []
components = []
for a in range(6):
    for b in range(a + 1, 6):
        coord1 = pca.components_[a] * np.sqrt(pca.explained_variance_[a])
        coord2 = pca.components_[b] * np.sqrt(pca.explained_variance_[b])
```

```
        coords.append((coord1, coord2))
        components.append((a+1, b+1))

# Plot Variable factor map for each combination of the first 6 principal com
fig, axs = plt.subplots(3, 5, figsize=(20, 12))
for k, (coord1, coord2) in enumerate(coords):
    row = k // 5
    col = k % 5
    ax = axs[row, col]
    for i, j, nom, night in zip(coord1, coord2, loading.columns, is_night):
        color = 'blue' if night else 'red'
        ax.arrow(0, 0, i, j, color=color, alpha=0.7, width=0.0001)
        ax.text(i, j, nom, fontsize=7, color=color)

    ax.axis((-1.1, 1.1, -1.1, 1.1))
    ax.hlines(y=0, xmin=-1.1, xmax=1.1, colors="black", linestyle="--", line
    ax.vlines(x=0, ymin=-1.1, ymax=1.1, colors="black", linestyle="--", line
    ax.add_artist(plt.Circle((0, 0), radius=1, color='cornflowerblue', fill=

    ax.set_title(f'Variables factor map (Components{components[k]})')
    ax.grid(True)

    labels = ['Night','Day']
    legend_elements = [Line2D([0], [0], marker='o', color='w', markerfacecol
                       Line2D([0], [0], marker='o', color='w', markerfacecol

    ax.legend(handles=legend_elements, labels=labels)

plt.tight_layout()
plt.show()
```



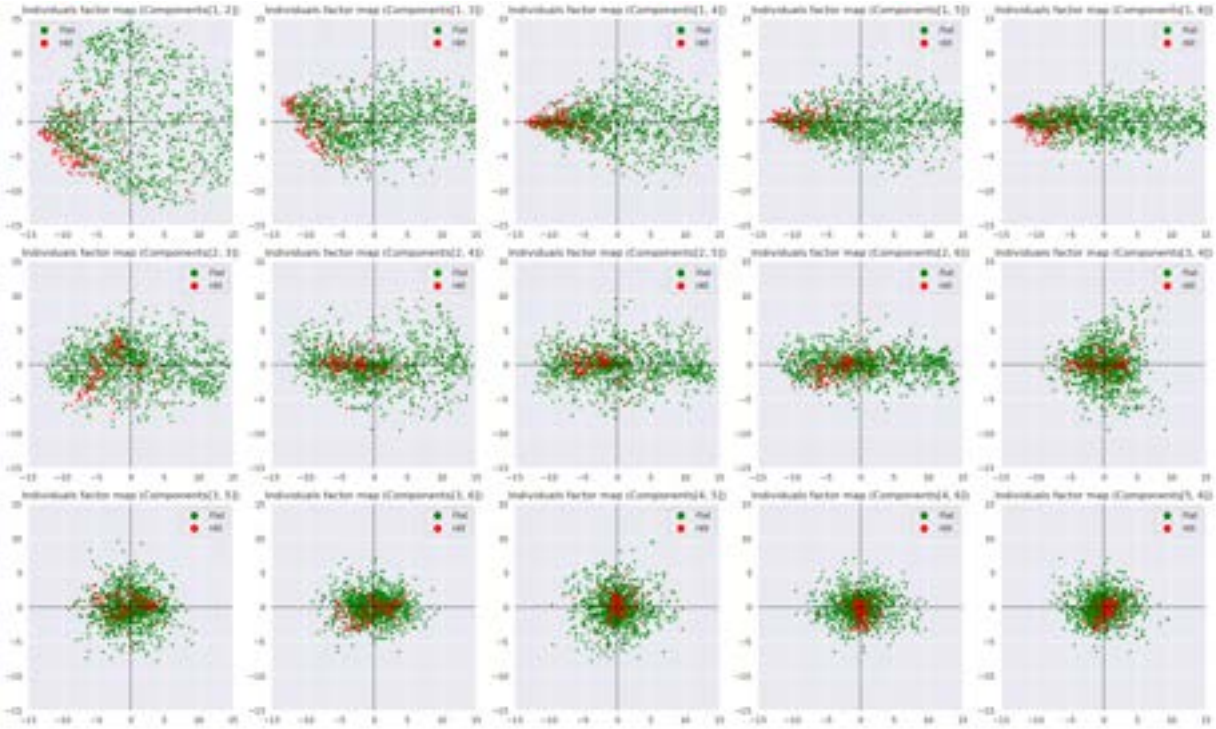
We observe that the second component effectively distinguishes the stations based on the time of day, separating them into distinct groups corresponding to day (component

2 > 0) and night hours (component 2 < 0).

2.4. Individual factor maps

2.4.1. Individual factor maps colored by hill position

In [37]:
```
from itertools import combinations

# Get all combinations of indices for the first 6 principal components
indices_combinations = list(combinations(range(6), 2))

# Plot Variable factor map for each combination of the first 6 principal com
fig, axs = plt.subplots(3, 5, figsize=(20, 12))
for k, (a, b) in enumerate(indices_combinations):
    row = k // 5
    col = k % 5
    ax = axs[row, col]
    for nom in [False, True]:
        indices = coord["bonus"] == nom  # Get indices where the condition i
        color = "red" if nom else "green"  # Determine color based on bonus
        ax.scatter(loading_pca[indices, a], loading_pca[indices, b], s=5, li

    ax.axis((-15, 15, -15, 15))
    ax.hlines(y=0, xmin=-15, xmax=15, colors="black", linestyle="--", linewi
    ax.vlines(x=0, ymin=-15, ymax=15, colors="black", linestyle="--", linewi

    ax.set_title(f'Individuals factor map (Components{[a+1, b+1]})')
    ax.grid(True)

    # Create custom legend with colors
    labels = ['Flat', 'Hill']
    legend_elements = [Line2D([0], [0], marker='o', color='w', markerfacecol
                       Line2D([0], [0], marker='o', color='w', markerfacecol

    ax.legend(handles=legend_elements)

plt.tight_layout()
plt.show()
```



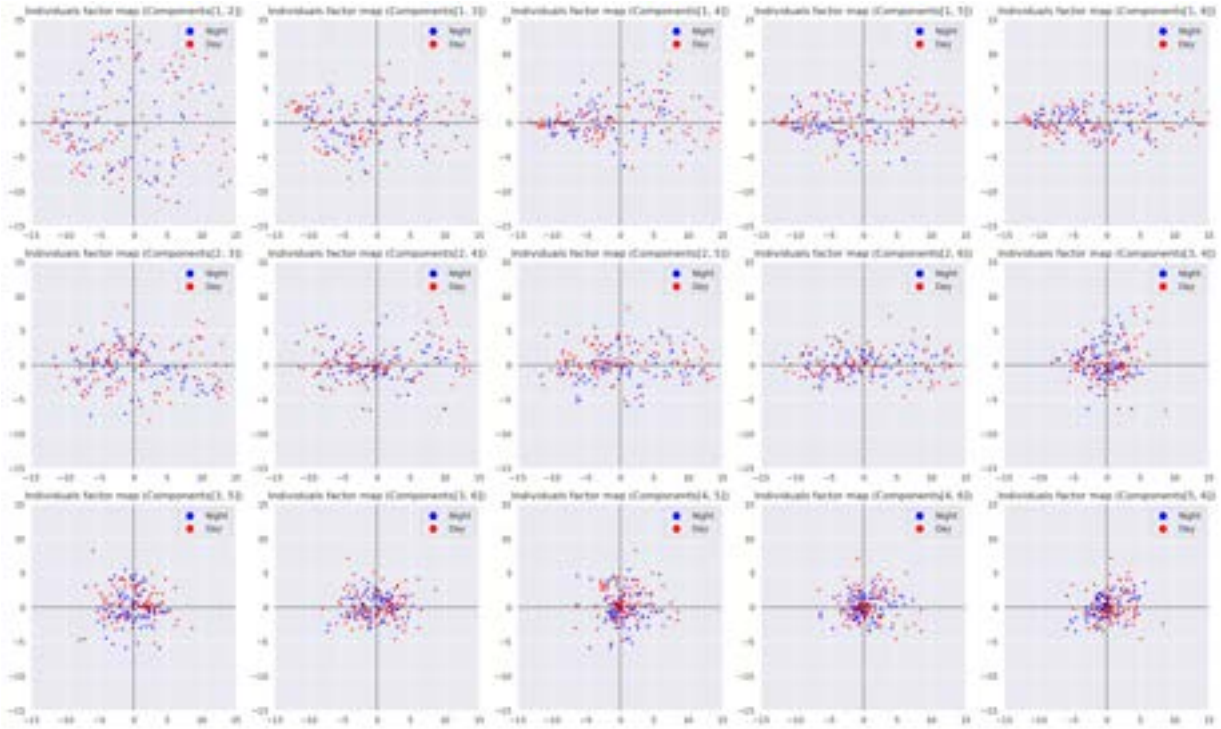
We observe most of the stations are placed on flat ground, which corresponds to what we saw in the pie chart.

We note that the stations placed on a hill are in the negative part of component 1. However, being in the negative side of component 1 does not mean that the staation is placed on a hill.

2.4.2. Individual factor maps colored by day and night

In [38]:
```
# Get all combinations of indices for the first 6 principal components
indices_combinations = list(combinations(range(6), 2))

# Plot individual factor map for each combination of the first 15 principal
fig, axs = plt.subplots(3, 5, figsize=(20, 12))
for k, (a, b) in enumerate(indices_combinations):
    if k >= len(axs.flat):  # Check if we've used all subplots
        break
    row = k // 5
    col = k % 5
    ax = axs[row, col]
    for night in [0, 1]:
        indices = np.where(is_night[:1189] == night)[0][:1189]  # Get indice
        color = 'blue' if night else 'red'  # Determine color based on night
        ax.scatter(loading_pca[indices, a], loading_pca[indices, b], s=5, li

    ax.axis((-15, 15, -15, 15))
    ax.hlines(y=0, xmin=-15, xmax=15, colors="black", linestyle="--", linewi
    ax.vlines(x=0, ymin=-15, ymax=15, colors="black", linestyle="--", linewi

    ax.set_title(f'Individuals factor map (Components{[a+1, b+1]})')
    ax.grid(True)
```

```
    # Create custom legend with colors
    labels = ['Night', 'Day']
    legend_elements = [Line2D([0], [0], marker='o', color='w', markerfacecol
                       Line2D([0], [0], marker='o', color='w', markerfacecol

    ax.legend(handles=legend_elements)

plt.tight_layout()
plt.show()
```



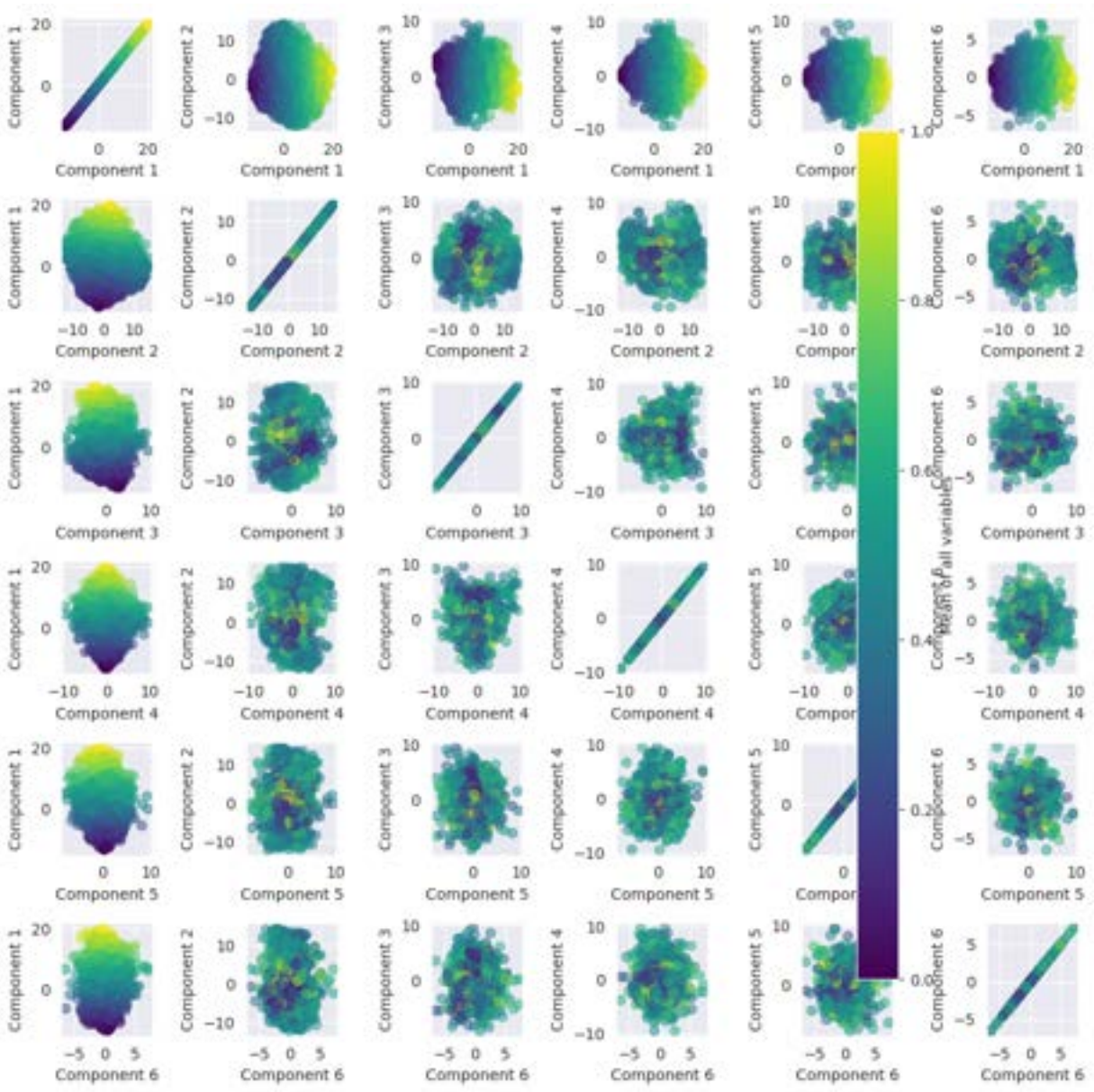
We cannot make new interpretations labeling the individual factor maps by night and day hours.

2.4.4. Individual factor maps of mean of all stations

In [39]:
```
# Calculate the mean of all variables in the loading DataFrame
mean_loading = loading.mean(axis=1)

# Define the number of components to consider
num_components = 6

# Create a grid of subplots
fig, axs = plt.subplots(num_components, num_components, figsize=(10, 10))

# Loop through each combination of components
for i in range(num_components):
    for j in range(num_components):
        # Plot the scatter plot for the current combination of components
        axs[i, j].scatter(loading_pca[:, i], loading_pca[:, j], c=mean_loadi

        # Set labels for the axes
```

```
        axs[i, j].set_xlabel(f'Component {i+1}')
        axs[i, j].set_ylabel(f'Component {j+1}')

# Add a colorbar to the last axis
fig.colorbar(axs[0, 0].scatter([], [], c=[], cmap='viridis'), ax=axs[:, :],

# Adjust layout
plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_336014/4055499583.py:24: UserWarning:

This figure includes Axes that are not compatible with tight_layout, so resu
lts might be incorrect.
```



We immediately see that the loading score is well represented by component 1. Small loading scores are in the negative side of component 1 and the higher loading scores are in the positive side of component 1.

Part 3: Clustering on original data

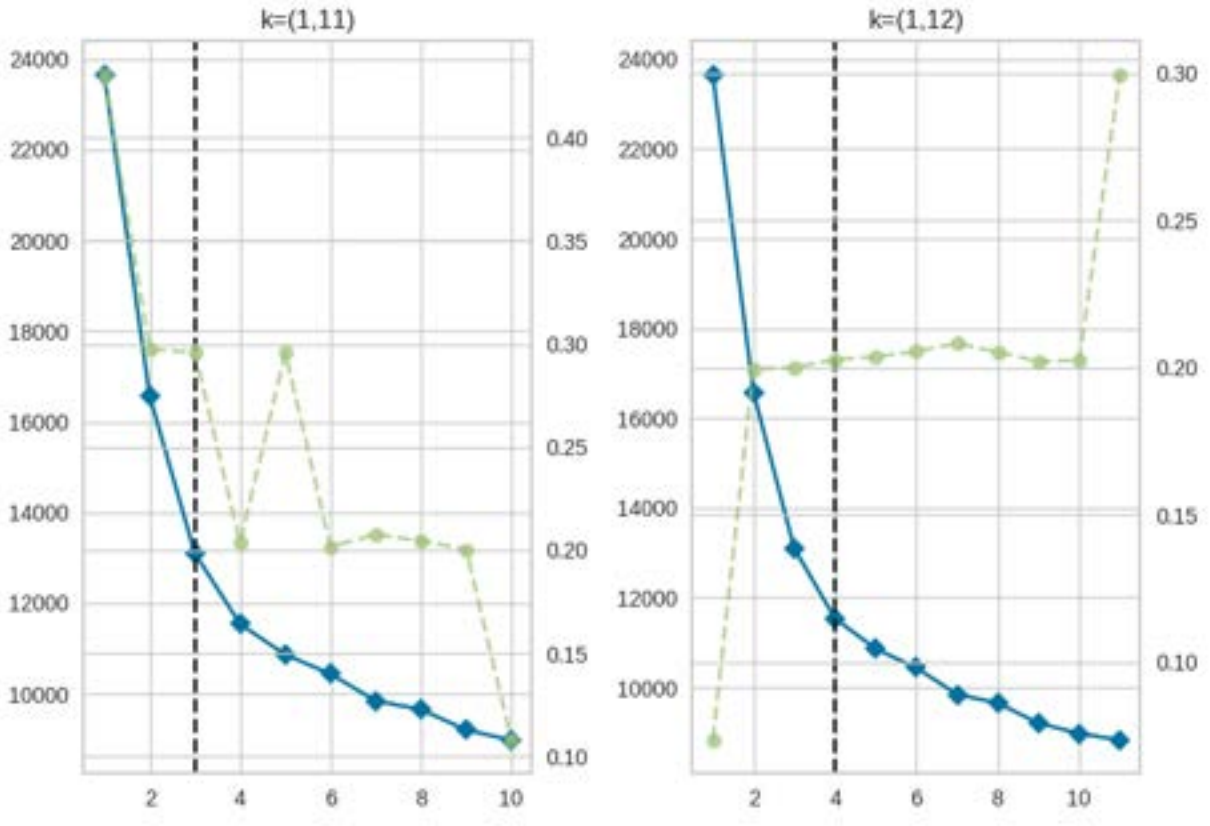
3.1. K-means clustering

3.1.1. Selection of the number of clusters

3.1.1.1. Determining the number of clusters using the total within sum of square metric

In [40]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from matplotlib import colors

from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
```

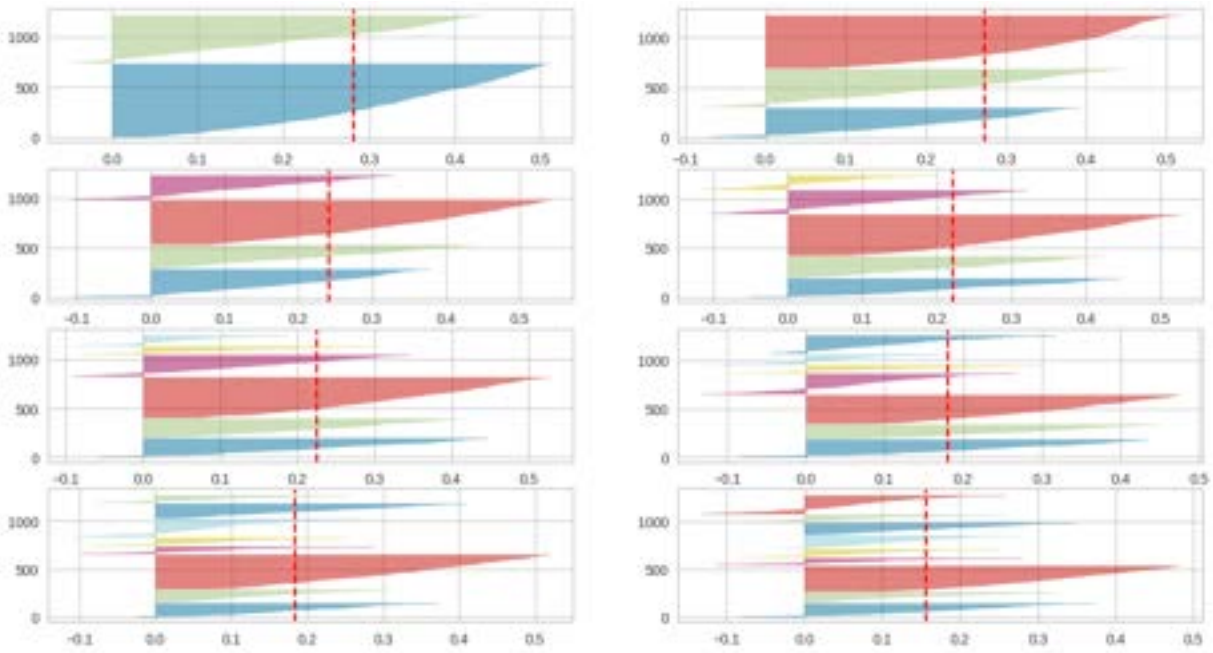
In [41]:
```
kmeans = KMeans(init='k-means++', max_iter=100, n_init = "auto", random_stat
visualizer_1 = KElbowVisualizer(kmeans, k=(1,11), metric='distortion')
visualizer_2 = KElbowVisualizer(kmeans, k=(1,12), metric='distortion')

plt.subplot(1,2,1)
visualizer_1.fit(loading)
plt.title("k=(1,11)")

plt.subplot(1,2,2)
visualizer_2.fit(loading)
plt.title("k=(1,12)")

plt.tight_layout()
plt.show()
```



Here we observe an anomaly in the function KElbowVisualizer. When we are looking for the best number of clusters in two different intervals we get different values. Since we do not know which one is better we will perform the k-means clustering with k=2 and k=3 to see which one is actually better.

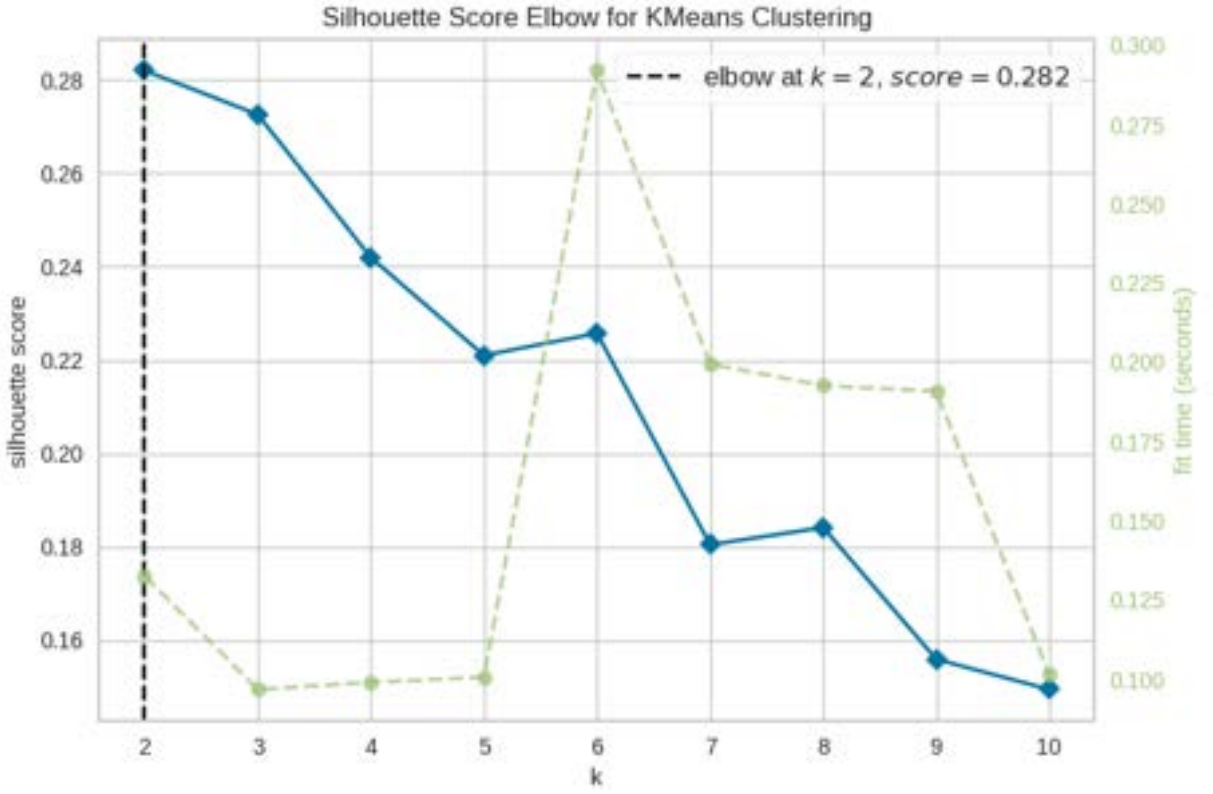
3.1.1.2. Determining the number of clusters using the silhouette scores metric

In [42]:
```
fig, ax = plt.subplots(4, 2, figsize=(15,8))

for k in range(2, 10):
    kmeans = KMeans(n_clusters=k, init='k-means++',n_init = "auto", max_iter
    q, mod = divmod(k, 2)

    # Create SilhouetteVisualizer instance with KMeans instance
    visualizer = SilhouetteVisualizer(kmeans, colors='yellowbrick', ax=ax[q-
    visualizer.fit(loading)
```



Here we visualised the silhouette score for different values of k. We can see that no matter the value of k, there is always a negative score which means the classification will never be optimal. We will keep that in mind in the rest of our study.

In [43]:
```
visualizer = KElbowVisualizer(kmeans, k=(2,11), n_init = "auto", metric='sil

visualizer.fit(loading)
visualizer.show()
```



Out[43]:
```
<Axes: title={'center': 'Silhouette Score Elbow for KMeans Clustering'}, xl
abel='k', ylabel='silhouette score'>
```

According to the distortion score elbow method, the best K values are either 3 or 4.

For the silhouette score elbow method, the optimal K value is 2.

We will further investigate the number of clusters using K values of 2, 3, and 4.

3.1.2. Visualization and interpretation of k-means clusters

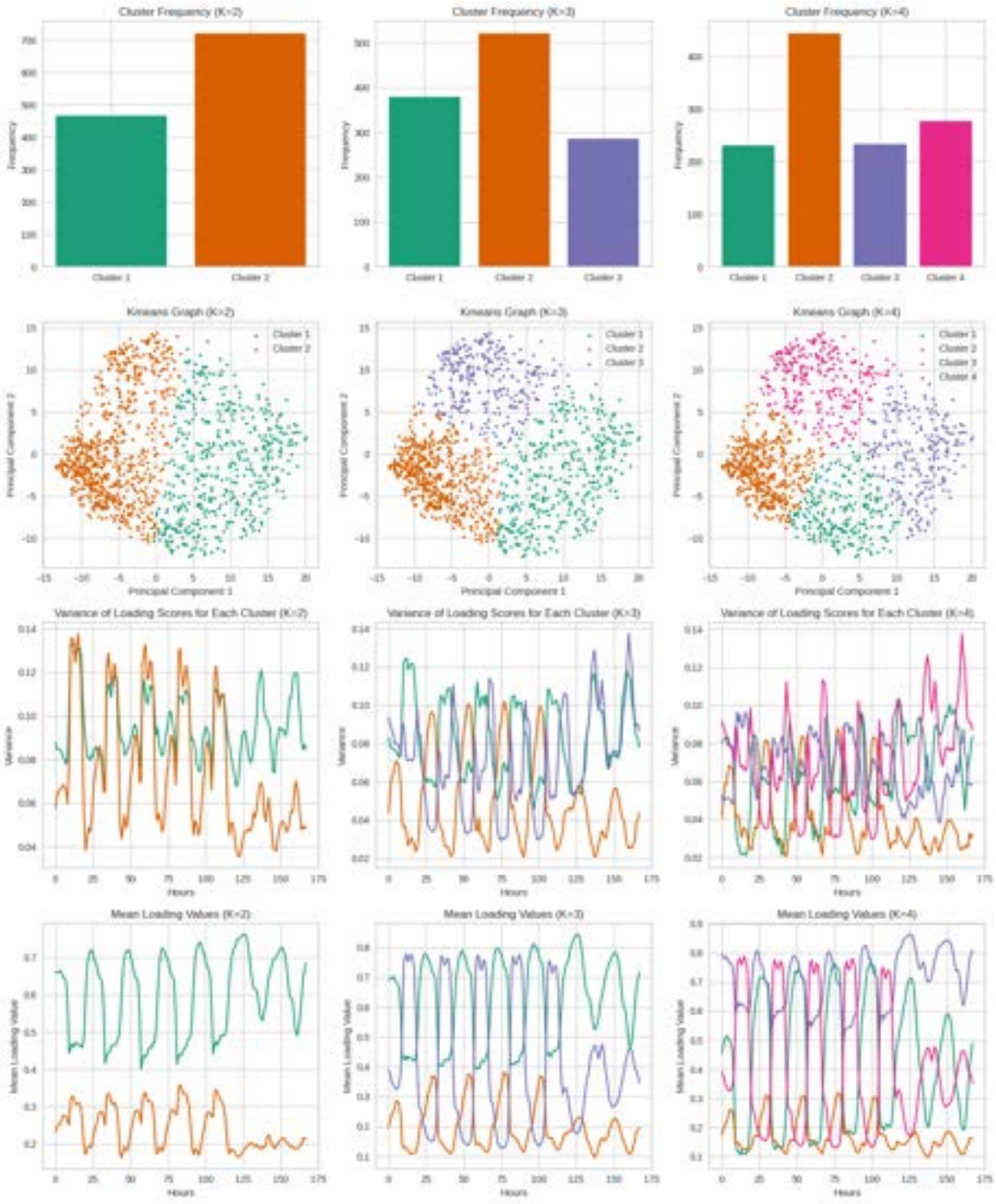
3.1.2.1. Four descriptive plots of cluster

For each value of k we have four different plots:

1. histogram of the frequency of each cluster
2. individual factor map colored by cluster
3. variance of loading scores for each cluster
4. mean loading values per hour cluster

In [44]:
```
K_values = [2, 3, 4]
cmap = plt.get_cmap('Dark2')
fig, axs = plt.subplots(4, len(K_values), figsize=(15, 18))
```

```
# Loop through each value of K
for i, K in enumerate(K_values):
    # Perform K-means clustering
    kmeans_loading = KMeans(n_clusters=K, init='k-means++', n_init = "auto",
    clusters_kmeans_loading = kmeans_loading.fit_predict(loading)
    cluster_freq = np.unique(clusters_kmeans_loading, return_counts=True)
    cluster_names = [f"Cluster {i+1}" for i in range(K)]

    # Plot Cluster Frequency
    axs[0, i].bar(cluster_names, cluster_freq[1], color=cmap.colors)
    axs[0, i].set_ylabel("Frequency")
    axs[0, i].set_title(f"Cluster Frequency (K={K})")

    # Plot Kmeans Graph
    for j in range(K):
        axs[1, i].scatter(loading_pca[clusters_kmeans_loading == j, 0], load
    axs[1, i].set_title(f"Kmeans Graph (K={K})")
    axs[1, i].legend()
    axs[1, i].set_xlabel("Principal Component 1")
    axs[1, i].set_ylabel("Principal Component 2")

    # Plot the new type of plot (replace 'plot 3' with the appropriate title
    x = np.arange(loading.shape[1])
    for j in range(K):
        axs[2, i].plot(x, np.var(loading[clusters_kmeans_loading == j], axis

    axs[2, i].set_xlabel("Hours")
    axs[2, i].set_ylabel("Variance")
    axs[2, i].set_title(f"Variance of Loading Scores for Each Cluster (K={K}

    # Plot mean loading values
    x = np.arange(loading.shape[1])
    for j in range(K):
        axs[3, i].plot(x, np.mean(loading[clusters_kmeans_loading == j], axi

    axs[3, i].set_xlabel("Hours")
    axs[3, i].set_ylabel("Mean Loading Value")
    axs[3, i].set_title(f"Mean Loading Values (K={K})")


# Adjust layout and display the plot
plt.tight_layout()
plt.show()
```



Interpretations:

First of all, we note that the variance for each cluster is smaller than 0.15 which we deem small enough to plot the mean loading scores.

- k=2 According to the fourth graph, cluster1 corresponds to the stations that are mostly full throughout the week while cluster2 corresponds to the stations that are mostly empty throughout the week. Additionnaly, on the second graph we can see that the clusters are clearly delimited by the component1 axis ie. cluster1 is exclusively represented with a positive component1 and cluster2 with a negative

component1. According to the PCA anaysis we performed earlier we excpect the stations on hills to only be present in cluster2.

- k=3 Once again, cluster1 corresponds to the stations that are mostly full during the week and cluster2 to the emptier ones. According to the graph of individuals we can see that cluster3 is represented on the positive part of the second component. According to the PCA analysis this means that the stations in cluster3 are full during the day. This seems to be confirmed by the final graph.

- k=4 This clusters distinguishes two groups well defined by component2, cluster3 and cluster4. According to our study of PCA components, the stations in cluster4 are full during day hours and those in cluster3 are full during night hours. This conclusion is coherent with the fourth plot! Cluster1 and cluster2 have the same caracteristics as they had in both previous cases.

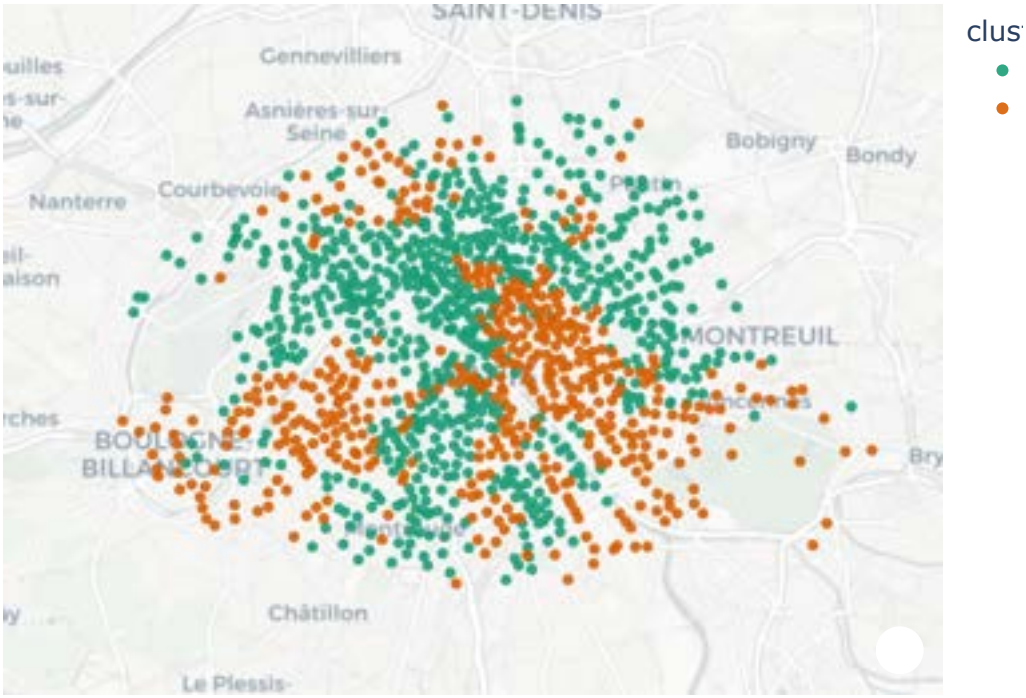
3.1.2.1. Visualization of clusters on the map

We will now compare the stations colored by clusters and the stations colored by hill position to see if they match.

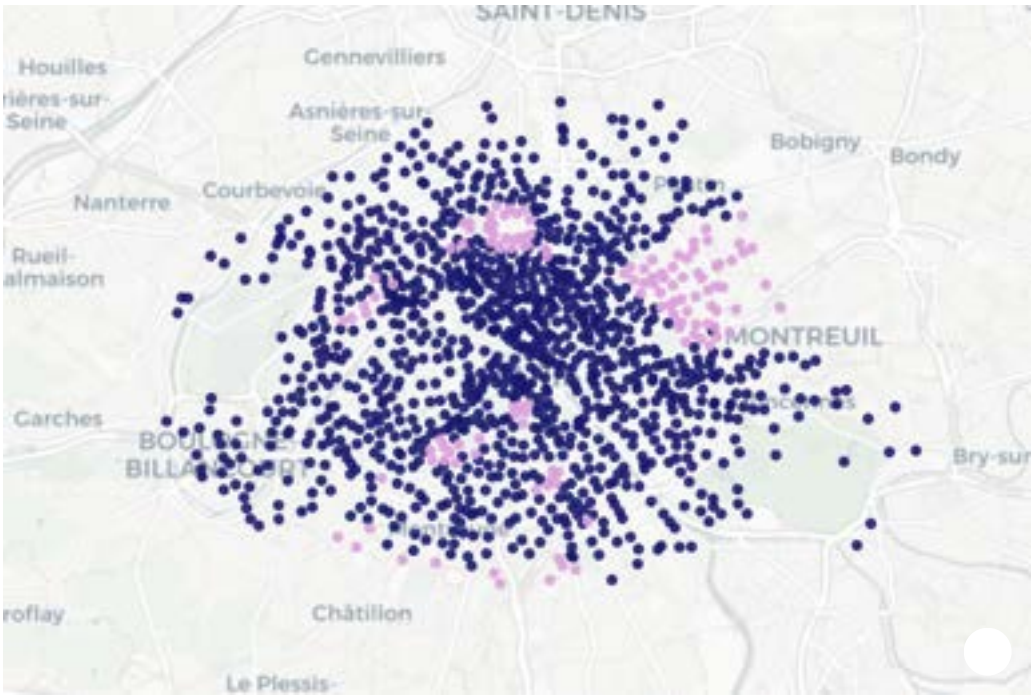
In [45]:
```
K = 2

# Perform KMeans clustering on your data
kmeans_loading = KMeans(n_clusters=K, init='k-means++',n_init = "auto",  ran
clusters_kmeans_loading = kmeans_loading.fit_predict(loading)

# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_kmeans_loading]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         #color_discrete_map={0: 'orange', 1: '#7570b3'},
                        color_discrete_sequence=px.colors.qualitative.Dark2,
                        mapbox_style="carto-positron", zoom=10, opacity=0.9,
                        title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                        color_discrete_map={0: 'midnightblue', 1: 'plum'},
                        mapbox_style="carto-positron", zoom=10, opacity=0.9,
                        title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()
```

```
# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



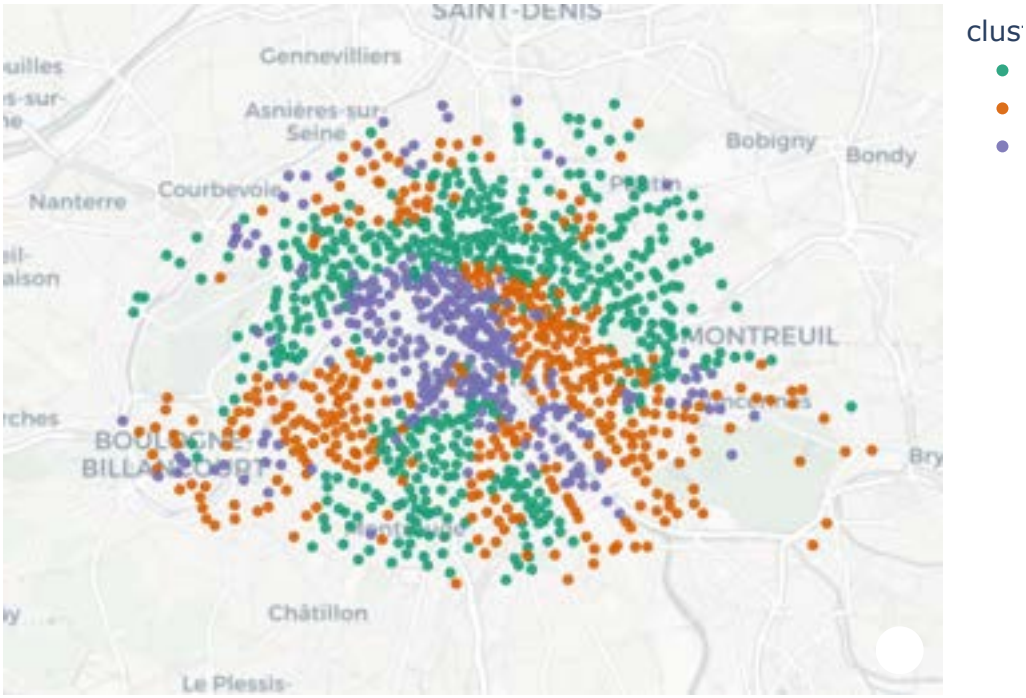
We excpected the stations on hills to only be present in cluster2 which is the case.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

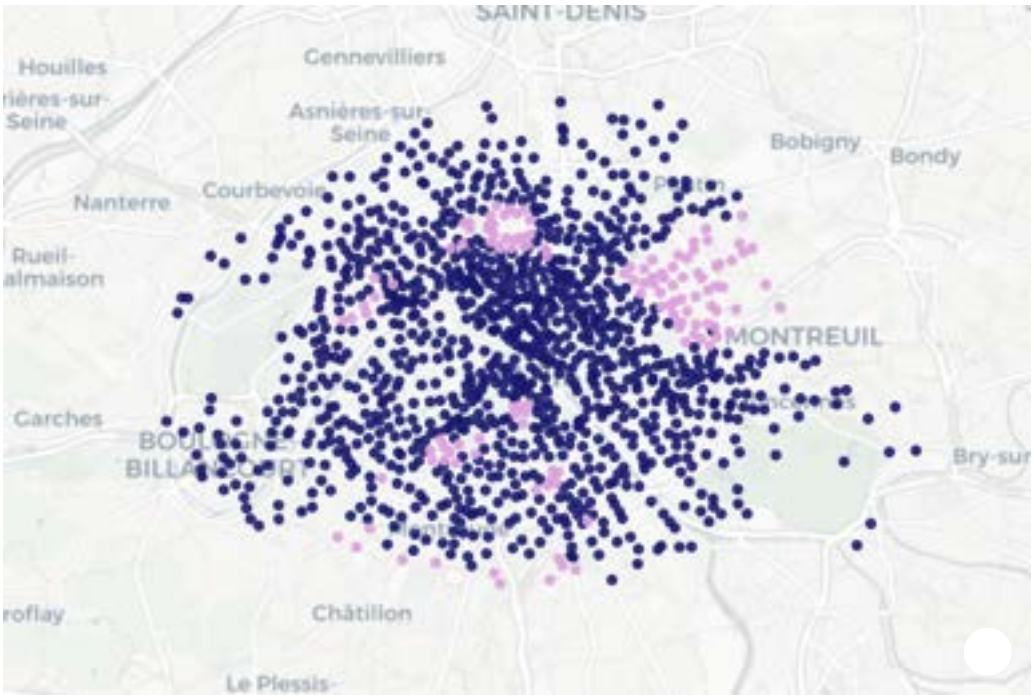
In [46]:
```
K = 3

# Perform KMeans clustering on your data
kmeans_loading = KMeans(n_clusters=K, init='k-means++',n_init = "auto", rand
clusters_kmeans_loading = kmeans_loading.fit_predict(loading)

# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_kmeans_loading]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
```

```
                        title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                        color_discrete_map={0: 'midnightblue', 1: 'plum'},
                        mapbox_style="carto-positron", zoom=10, opacity=0.9,
                        title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

The first thing we notice is that cluster3 corresponds to the stations in the city center. Additionnaly, we saw earlier that the stations in cluster3 are full during the daytime. This means that during the day there are a lot of velibs in the city center. This makes sense. People could go into the city center center during the day for a number of reasons (work, school, shopping, sightseeing, leisure and enjoying the weather, appointments, etc.). At the end of the day everyone would then go home, probably in the suburbs, with velibs, leaving the stations empty.

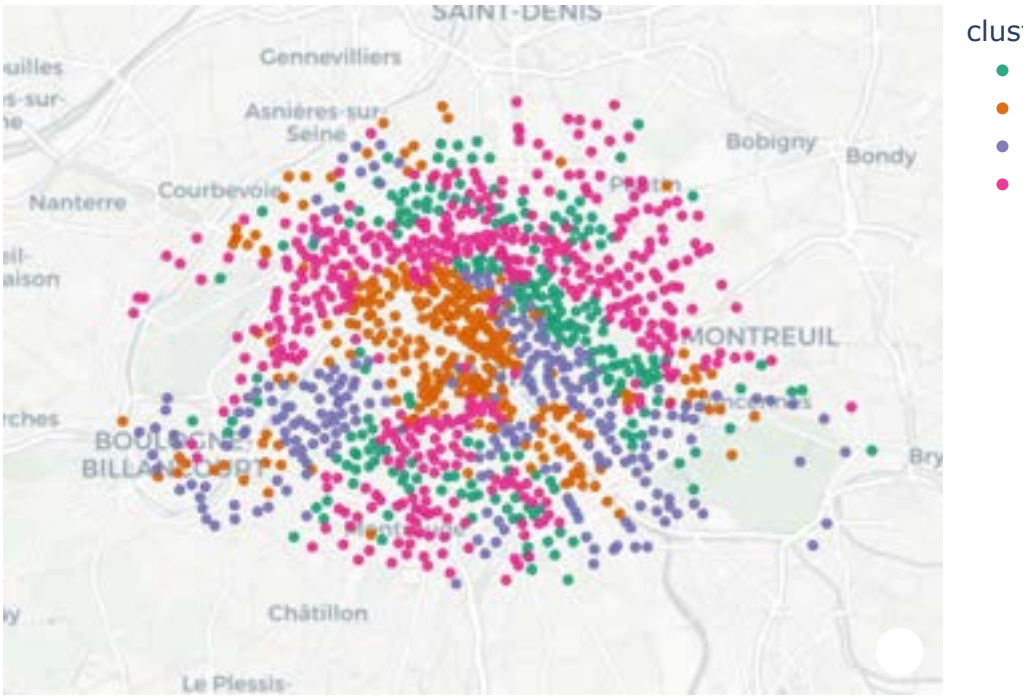
The stations on hill are in cluster2. Earlier we said that the stations in cluster2 are mostly empty during the week. This makes sense as it is very easy to go down a hill with a bike. This could mean that people who take velibs down a hill do not bother to take them back up and simply store them elsewhere, meaning that the stations on hill are empty almost all the time.

The stations in cluster1 are still relatively in the city center but in a zone where housing is more affordable. This could explain wht stations in thay cluster are on average full no

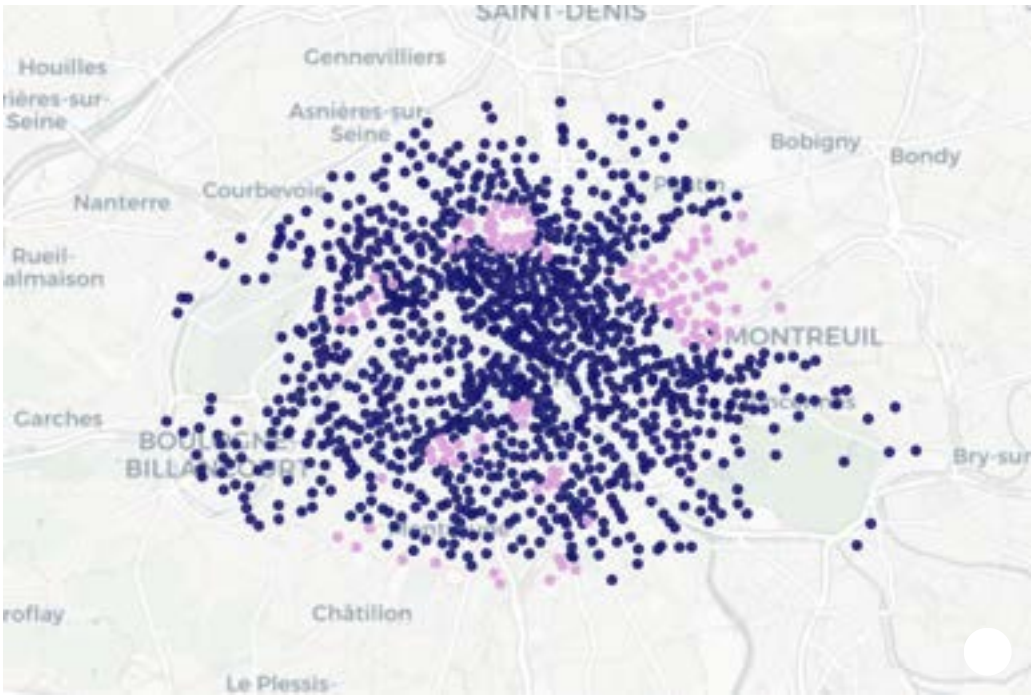
matter the time of day as opposed to the hyper-center of Paris.

In [47]:
```
K = 4

# Perform KMeans clustering on your data
kmeans_loading = KMeans(n_clusters=K, init='k-means++',n_init = "auto", rand
clusters_kmeans_loading = kmeans_loading.fit_predict(loading)

# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_kmeans_loading]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

Again, we observe that cluster2 corresponds approximately to the stations placed on a hill and earlier we said that the stations in cluster2 are mostly empty during the week. So the interpretation is the same as in the previous case.

Cluster4 is once again the city center cluster and it is fuller during the day.

According to the previous cell cluster1 is on average fuller at night. With the map we also see that it is in neighborhoods that are famous for their nightlife and bars (Répoublique, Bastille, etc.)

3.2. HCA clustering

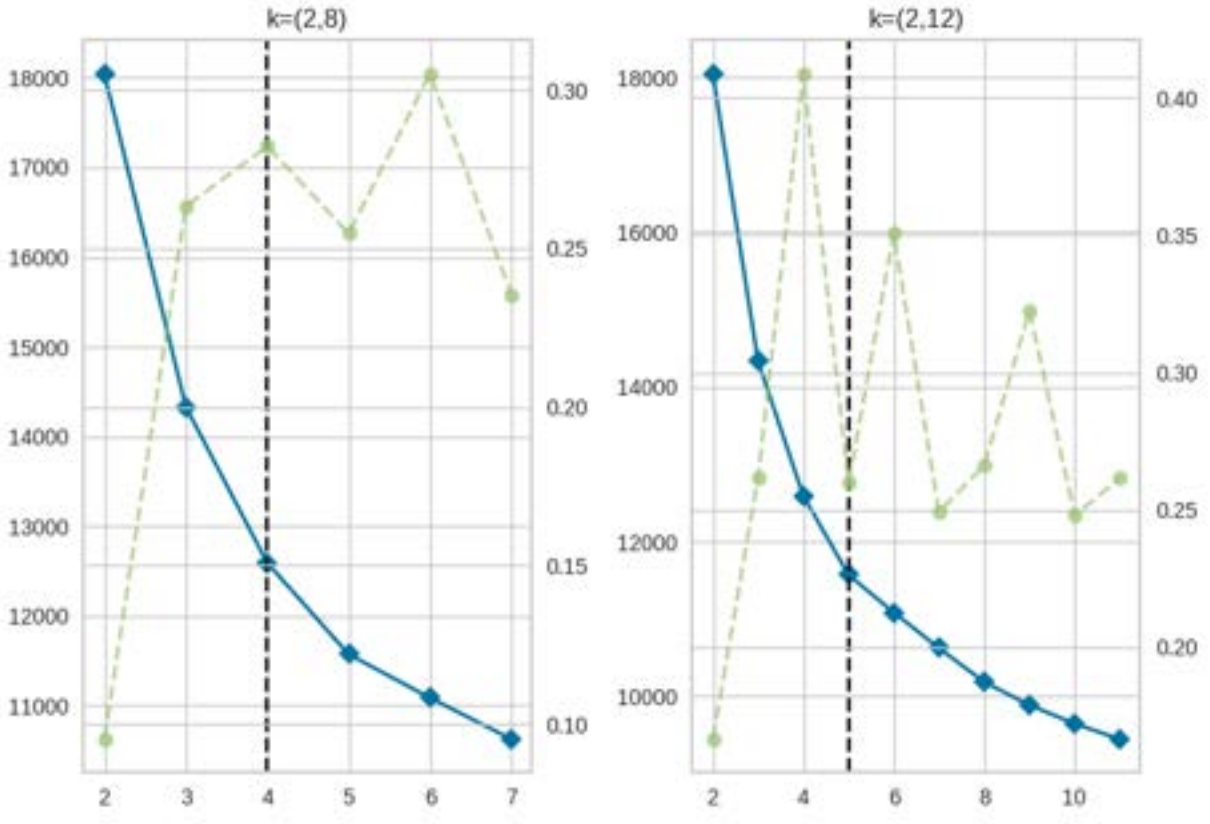
In this section, we will perform the HCA clustering method to make the same analysis as in the preview section.

3.2.1. Selection of the number of clusters

3.2.1.1. Determining the number of clusters using the total within sum of square metric

In [48]:
```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage, cut_tree
import scipy.cluster.hierarchy as sch
```

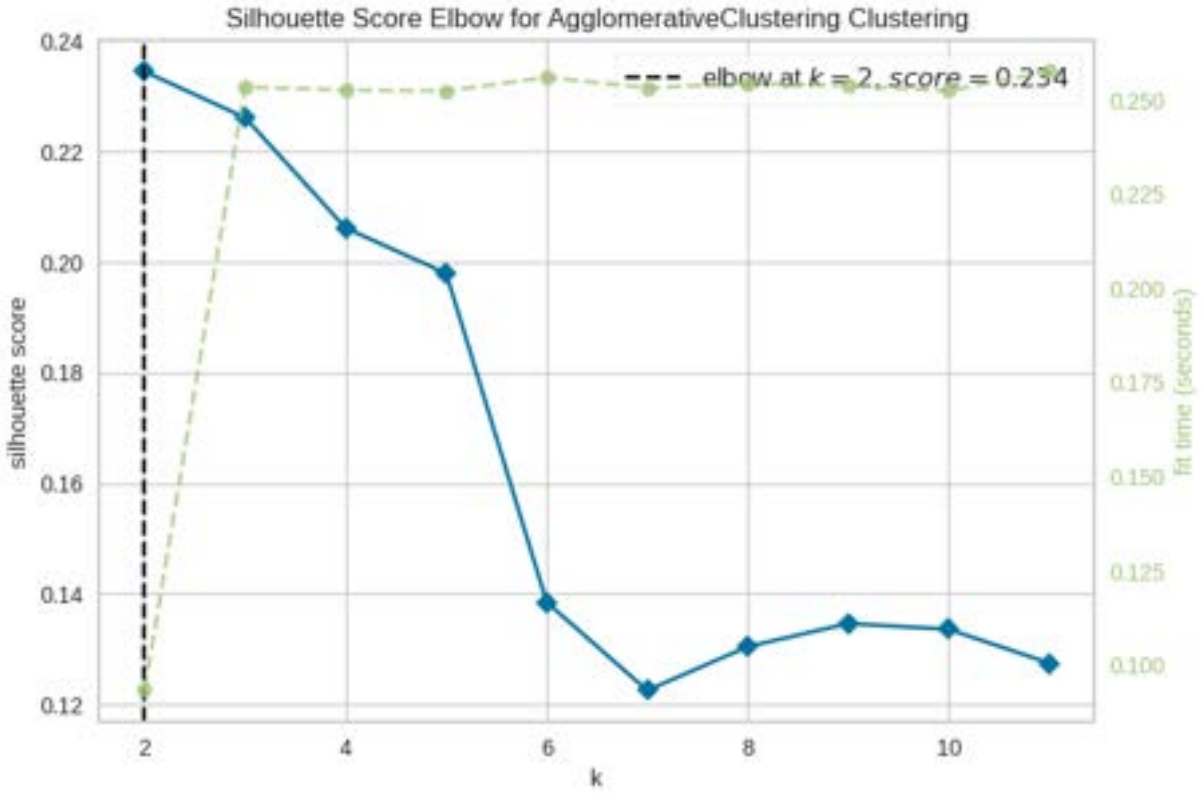
In [49]:
```
ac = AgglomerativeClustering(linkage='ward', compute_distances=True)
visualizer_1 = KElbowVisualizer(ac, k=(2,8), metric='distortion')
visualizer_2 = KElbowVisualizer(ac, k=(2,12), metric='distortion')

plt.subplot(1,2,1)
visualizer_1.fit(loading)  # Fit the data to the visualizer
plt.title("k=(2,8)")

plt.subplot(1,2,2)
visualizer_2.fit(loading)  # Fit the data to the visualizer
plt.title("k=(2,12)")

plt.tight_layout()
plt.show()
```



Here we observe an anomaly in the function KElbowVisualizer. When we are looking for the best number of clusters in two different intervals we get different values. Since we do not know which one is better we will perform the k-means clustering with k=4 and k=5 to see which one is actually better.

3.2.1.2. Determining the number of clusters using the silhouette metric

In [50]:
```
ac = AgglomerativeClustering(linkage='ward', compute_distances=True)
visualizer = KElbowVisualizer(ac, k=(2,12), metric='silhouette')

visualizer.fit(loading)  # Fit the data to the visualizer
visualizer.show()
plt.show()
```



According to the distortion score elbow method, the best K values are either 4 or 5.

For the silhouette score elbow method, the optimal K value is 2.

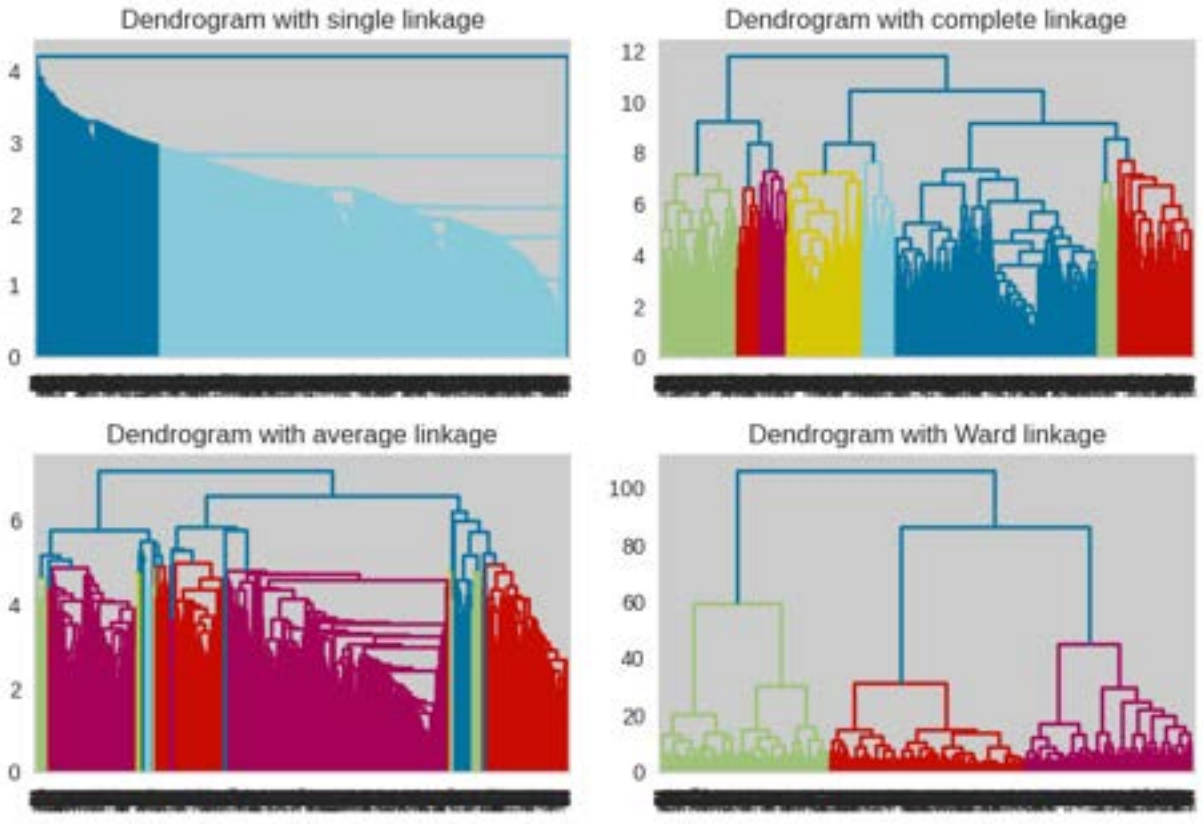
We will further investigate the number of clusters using K values of 2, 4, and 5.

3.2.2. Visualization of different dendrograms and evaluation of the effect of the choice of the linkage function

3.2.2.1. Dendrogram with different linkage methods

In [51]:
```
k=2

plt.subplot(2,2,1)
linkage_matrix_single = sch.linkage(loading, method='single')
sch.dendrogram(linkage_matrix_single)
plt.title("Dendrogram with single linkage")

plt.subplot(2,2,2)
```

```
linkage_matrix_complete = sch.linkage(loading, method='complete')
sch.dendrogram(linkage_matrix_complete)
plt.title("Dendrogram with complete linkage")

plt.subplot(2,2,3)
linkage_matrix_average = sch.linkage(loading, method='average')
sch.dendrogram(linkage_matrix_average)
plt.title("Dendrogram with average linkage")

plt.subplot(2,2,4)
linkage_matrix_ward = sch.linkage(loading, method='ward')
sch.dendrogram(linkage_matrix_ward)
plt.title("Dendrogram with Ward linkage")

plt.tight_layout()
plt.show()
```

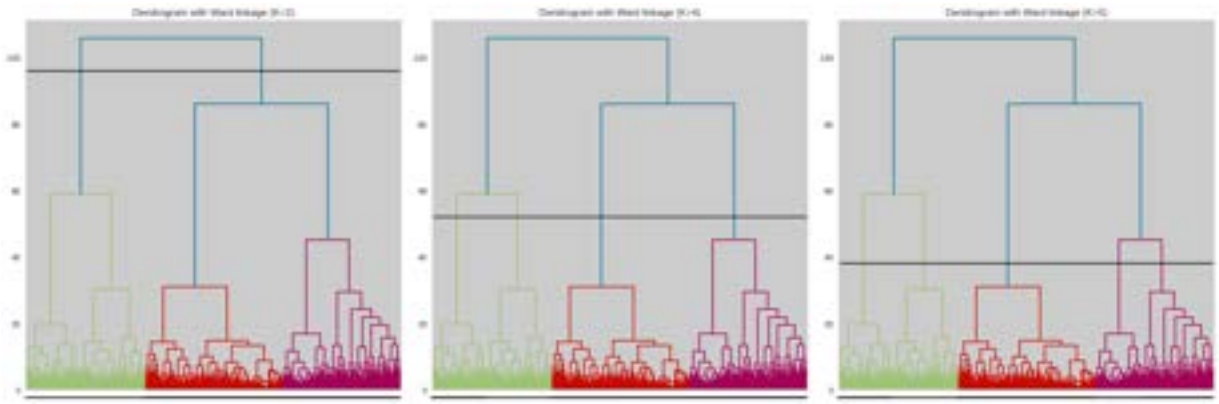


As seen in class, we will use Ward linkage to study the different values of k.

3.2.2.2. Dendrograms with Ward linkage

In [52]:
```
K_values = [2, 4, 5]
n_rows = 1
n_cols = len(K_values)
fig, axs = plt.subplots(n_rows, n_cols, figsize=(7*n_cols, 7*n_rows))

for i, K in enumerate(K_values):
    ac = AgglomerativeClustering(n_clusters=K, compute_distances=True, linka
    clusters = ac.fit_predict(loading)
    children = ac.children_
```

```
    distances = ac.distances_
    n_observations = np.arange(2, children.shape[0] + 2)
    linkage_matrix = np.c_[children, distances, n_observations]

    ax = axs[i] if n_rows == 1 else axs[i // n_cols, i % n_cols]

    sch.dendrogram(linkage_matrix, labels=ac.labels_, ax=ax)

    # Cutting the dendrogram to get K classes
    max_d = 0.5 * (ac.distances_[-K] + ac.distances_[-K + 1])
    ax.axhline(y=max_d, c='k')

    ax.set_title(f"Dendrogram with Ward linkage (K={K})")

plt.tight_layout()
plt.show()
```

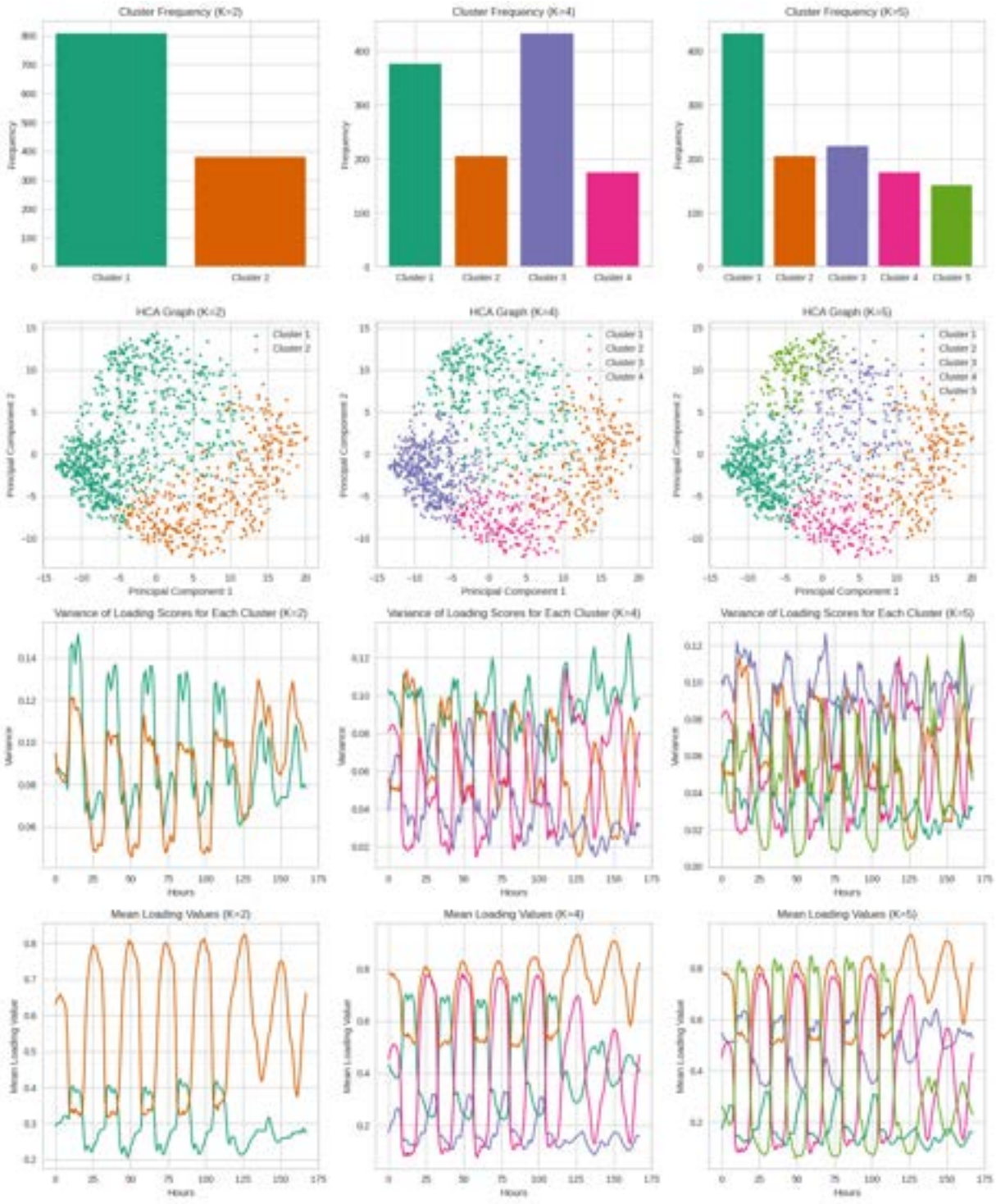


3.2.3. Visualization and interpretation of k-means clusters

3.2.3.1. Four descriptive plots of cluster

In [53]:
```
K_values = [2, 4, 5]
cmap = plt.get_cmap('Dark2')
fig, axs = plt.subplots(4, len(K_values), figsize=(15, 18))

for i, K in enumerate(K_values):

    # Agglomerative Clustering
    ac = AgglomerativeClustering(n_clusters=K, compute_distances=True, linka
    clusters_ac = ac.fit_predict(loading)
    cluster_freq = np.unique(clusters_ac, return_counts=True)
    cluster_names = [f"Cluster {i+1}" for i in range(K)]

    # Plot Cluster Frequency
    axs[0, i].bar(cluster_names, cluster_freq[1], color=cmap.colors)
    axs[0, i].set_ylabel("Frequency")
    axs[0, i].set_title(f"Cluster Frequency (K={K})")

    # Scatter plot of Agglomerative Clustering results
    for j in range(K):
        axs[1, i].scatter(loading_pca[clusters_ac == j, 0], loading_pca[clus
    axs[1, i].set_title(f"HCA Graph (K={K})")
```

```
    axs[1, i].legend()
    axs[1, i].set_xlabel("Principal Component 1")
    axs[1, i].set_ylabel("Principal Component 2")

    # Variance Plot
    x = np.arange(loading.shape[1])
    for j in range(K):
        axs[2, i].plot(x, np.var(loading[clusters_ac == j], axis=0), color=c

    axs[2, i].set_xlabel("Hours")
    axs[2, i].set_ylabel("Variance")
    axs[2, i].set_title(f"Variance of Loading Scores for Each Cluster (K={K}

    # Compute and plot mean loading values
    x = np.arange(loading.shape[1])  # Assuming loading is your data matrix
    for j in range(K):
        axs[3, i].plot(x, np.mean(loading[clusters_ac == j], axis=0), color=

    axs[3, i].set_xlabel("Hours")
    axs[3, i].set_ylabel("Mean Loading Value")
    axs[3, i].set_title(f"Mean Loading Values (K={K})")

plt.tight_layout()
plt.show()
```



Interpretations:

First of all, we note that the variance for each cluster (third line plots) is smaller than 0.15, meaning that plotting the mean loading scores (fourth line plots) is appropriate.

- k=2 Cluster1 correspond to the stations mostly empty during all week. While, cluster2 correspond to the stations mostly full during all week at night. We see a period of 24 h.
- k=4 We observe a diference during weekend for all clusters. Cluster1 is the only one

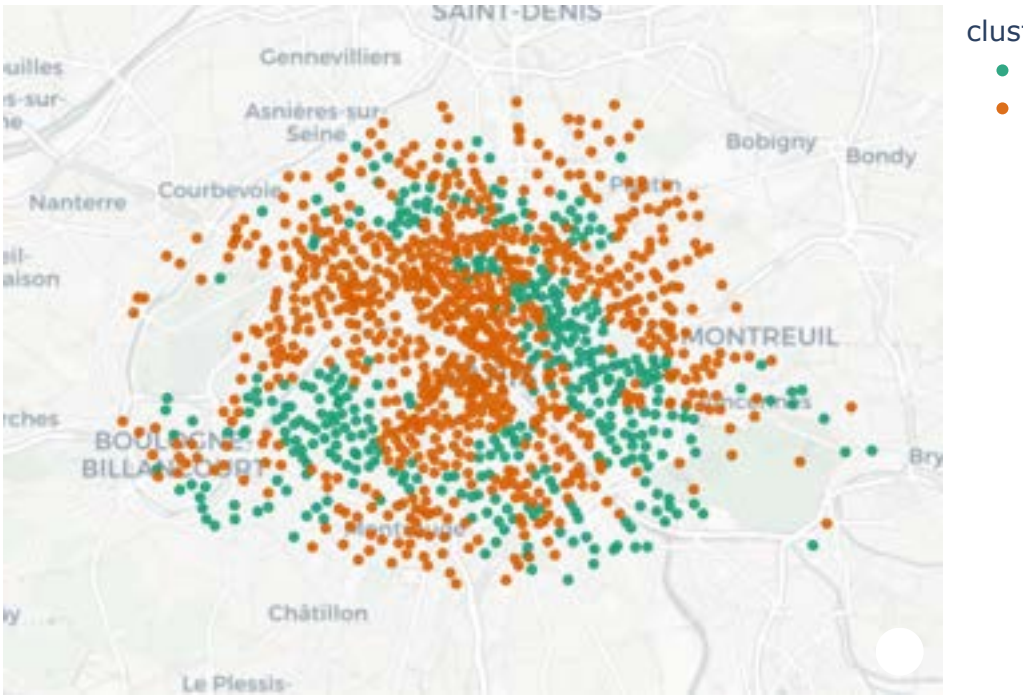
full during day and empty during nigth, it must correspon to the stations in the suburbs. Cluster2 is often full, specially on weekends, this cluster must correspond to the less popular stations. Cluster3 correspond to the stations mostly empty during all week. Cluster4 is similar to Cluster2, having the same period but, on the contrary, Cluster4 is mostly empty during weekends.

- k=5 We observe a similar clustering as in the case k=4, but this time two clusters, Cluster3 and Cluster5, are periodically full during day and empty during nigth.

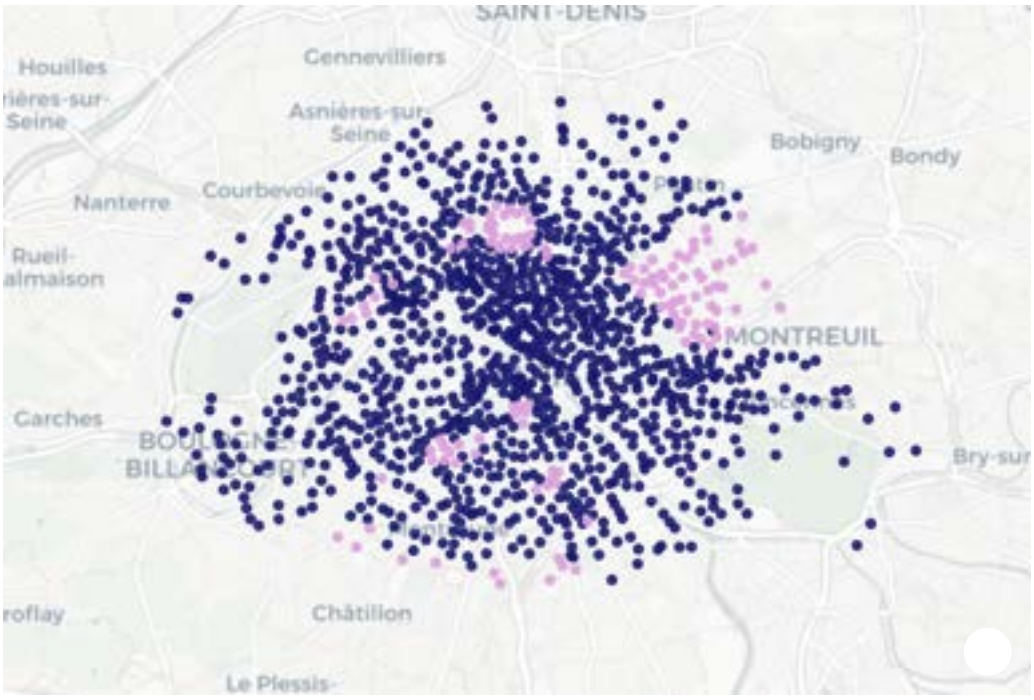
3.2.3.2. Visualization of clusters on the map

In [54]:
```
K = 2

# Perform Agglomerative Clustering
ac = AgglomerativeClustering(n_clusters=K, compute_distances=True, linkage='
clusters_ac = ac.fit_predict(loading)
# Convert cluster labels to strings for better visualization
cluster_names = ["Cluster " for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_ac]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



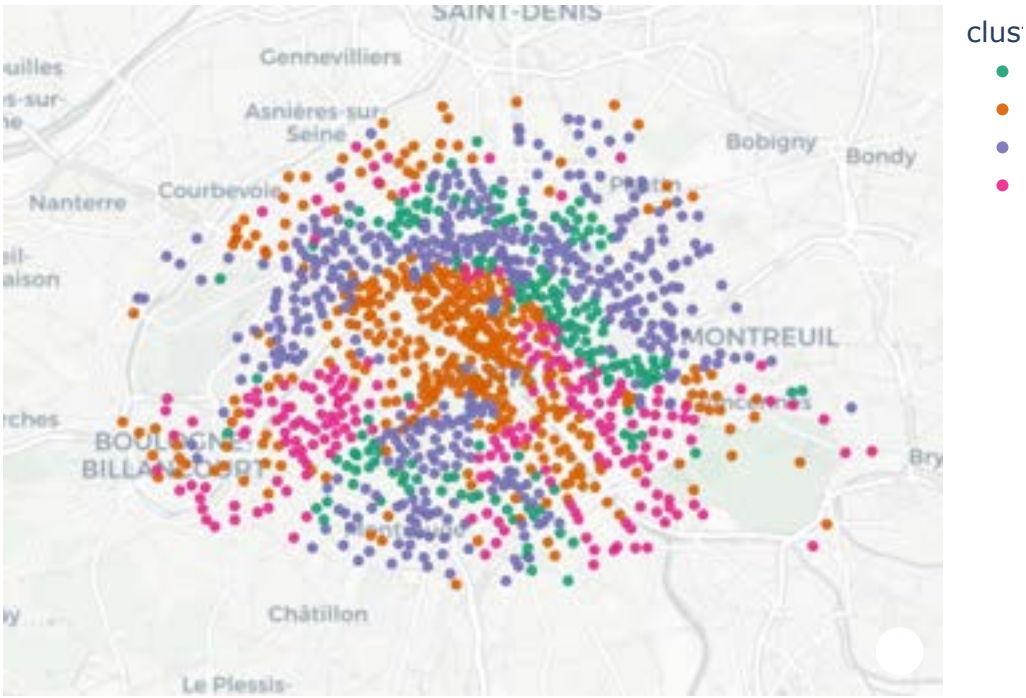
Our last interpretation was: Cluster1 correspond to the stations mostly empty during all week. While, cluster2 correspond to the stations mostly full during all week at night. We see a period of 24 h. In fact the stations in Cluster1 are placed in the city center, which explains why those stations are often empty.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

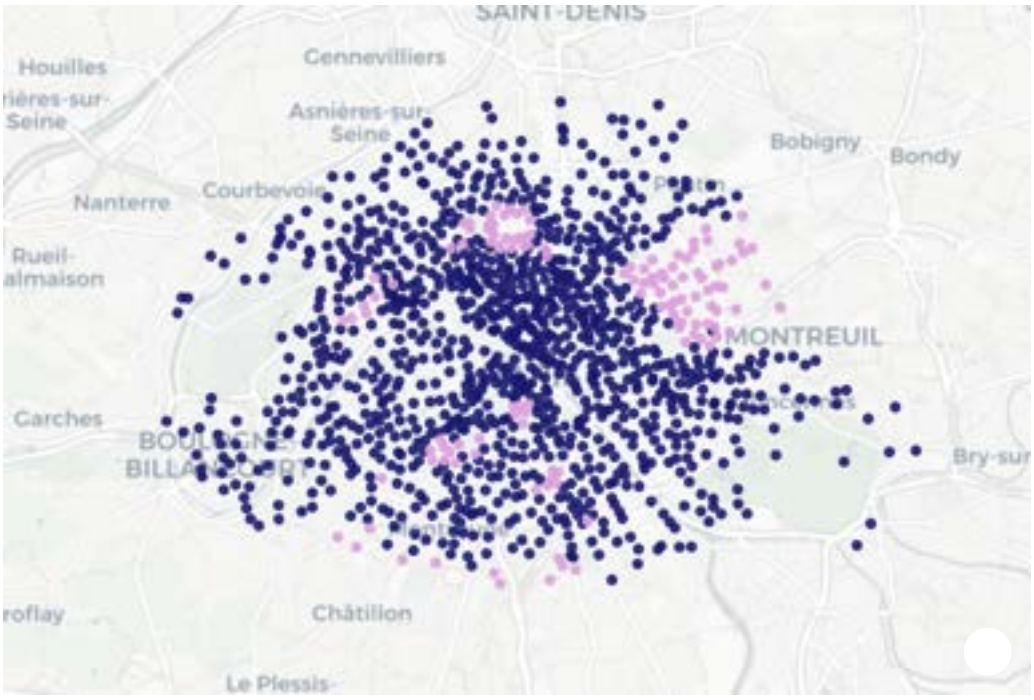
In [55]:
```
K = 4

# Perform Agglomerative Clustering
ac = AgglomerativeClustering(n_clusters=K, compute_distances=True, linkage='
clusters_ac = ac.fit_predict(loading)
# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_ac]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
```

```
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')


# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



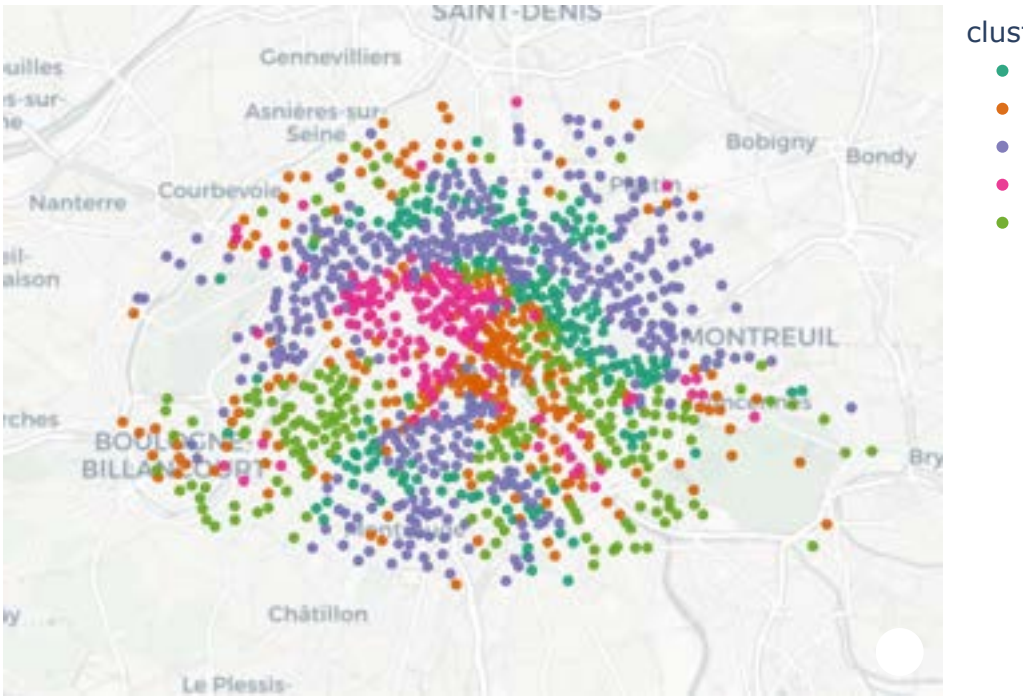
Our last interpretation was: We observe a diference during weekend for all clusters. Cluster1 is the only one full during day and empty during nigth, it must correspon to the stations in the suburbs. Cluster2 is often full, specially on weekends, this cluster must correspond to the less popular stations. Cluster3 correspond to the stations mostly empty during all week. Cluster4 is similar to Cluster2, having the same period but, on the contrary, Cluster4 is mostly empty during weekends.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

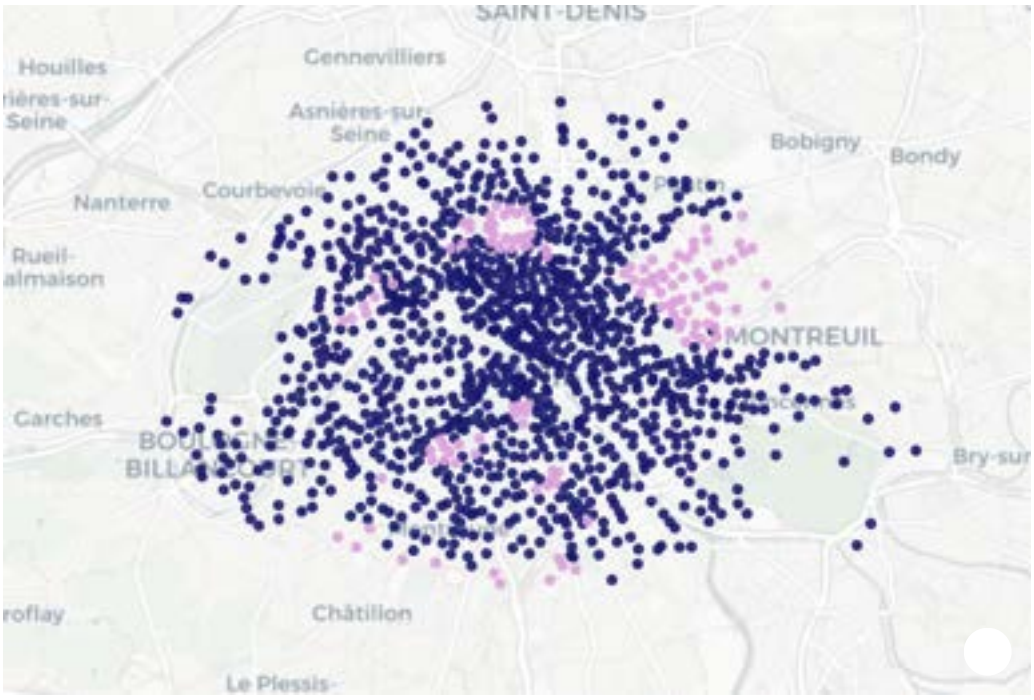
In [56]:
```
K = 5
cmap = plt.get_cmap('Dark2')
# Perform Agglomerative Clustering
ac = AgglomerativeClustering(n_clusters=K, compute_distances=True, linkage='
clusters_ac = ac.fit_predict(loading)
# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_ac]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
```

```
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')


# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



We observe that Cluster3 corresponds approximately to the stations placed on a hill.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

3.3. Gaussian Mixture clustering on original data

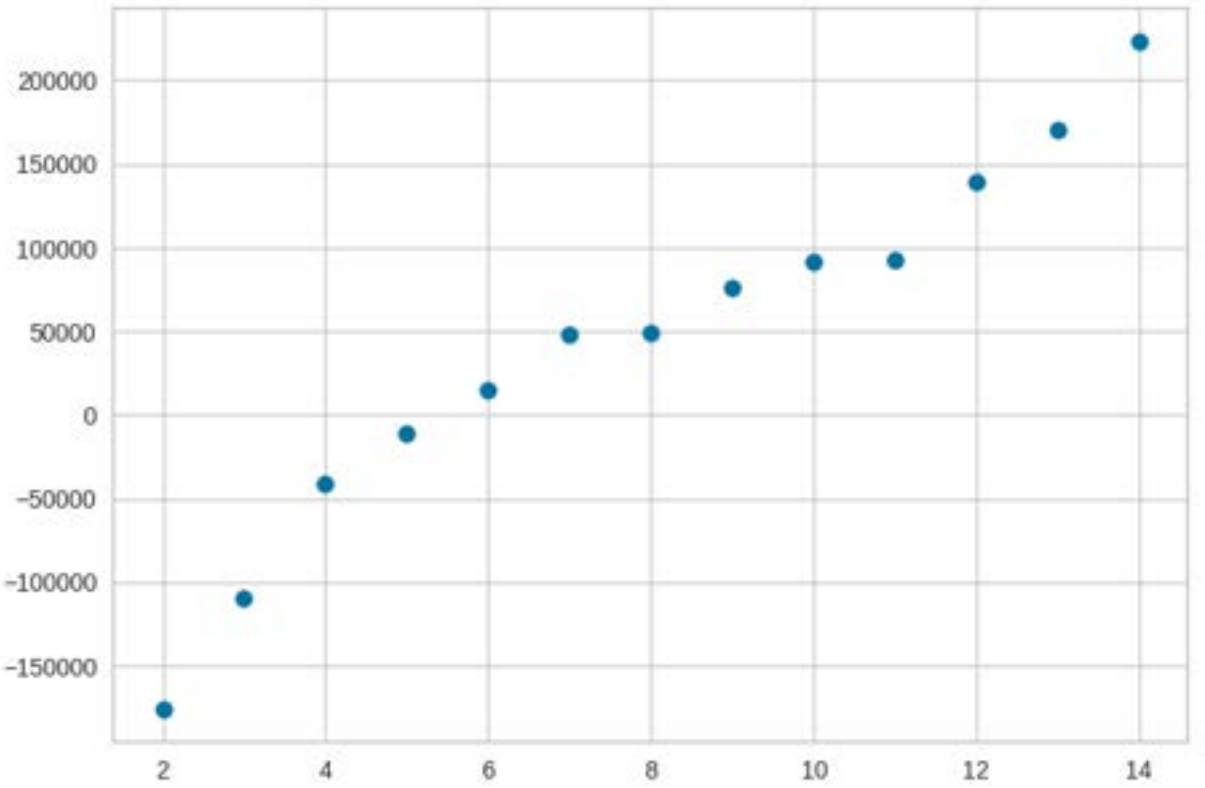
3.3.1. Selection of the number of clusters

Gaussain Mixture Modeling being based on probability and not on the distance between/whithin clusters, we are going to use others methods to detemine the best number of clusters.

3.3.1.1. Determining the number of clusters using BIC

In [57]:
```
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
import plotly.express as px
```

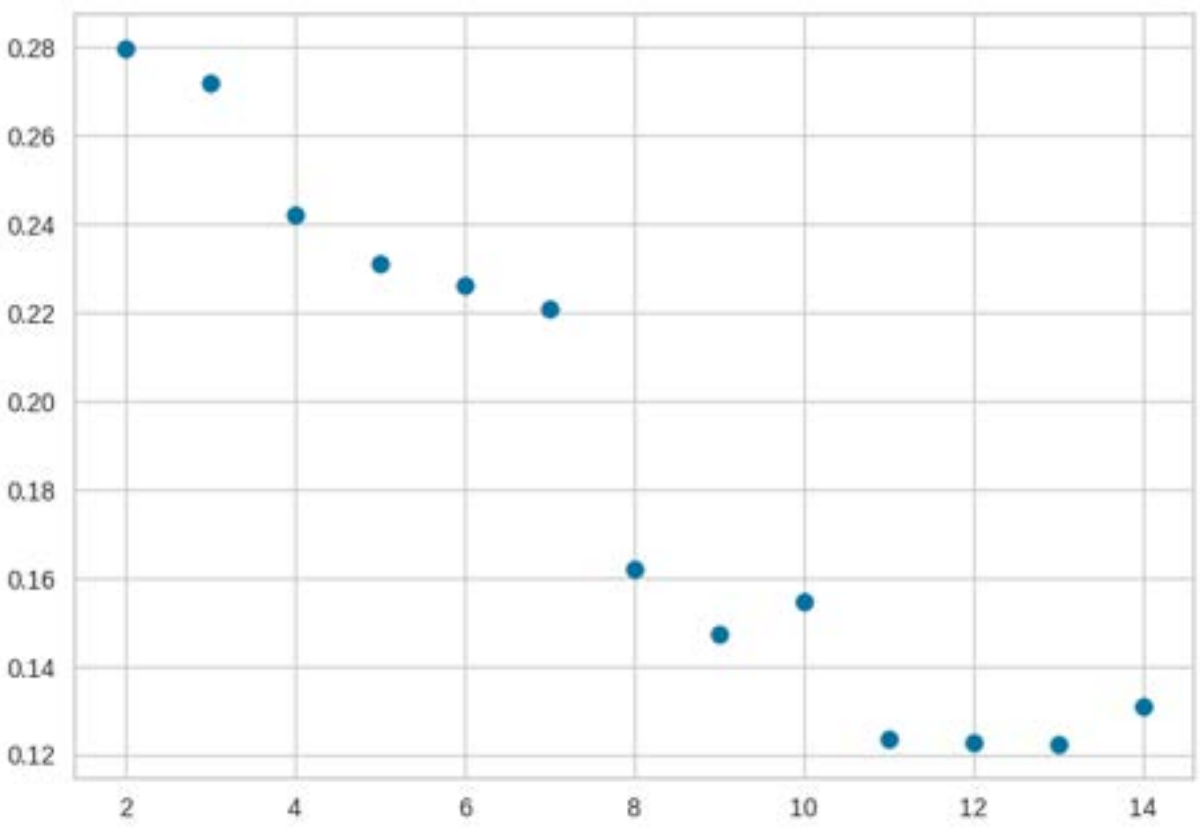
In [58]:
```
k_max = 15

bic = []
for k in range(2, k_max):
    gmm = GaussianMixture(n_components=k, init_params='kmeans', n_init=3)
    gmm.fit(loading)
    bic.append(gmm.bic(loading))
bic = np.array(bic)

plt.scatter(range(2, k_max), bic)
plt.show()
```



The number of clusters corresponding to the lowest BIC value is the optimal number of clusters for your data, so the best k= 2 .

3.3.1.2. Determining the number of clusters using silhouette score

In [59]:
```
k_max = 15

silhouette = []
for k in range(2, k_max):
    gmm = GaussianMixture(n_components=k, init_params='kmeans', n_init=3)
    clusters_gmm = gmm.fit_predict(loading)
    silhouette.append( silhouette_score(loading, clusters_gmm, metric='eucli
silhouette = np.array(silhouette)

plt.scatter(range(2, k_max), silhouette)
plt.show()
```



We look for the highest silhouette score, indicating the best clustering solution. Optimal Number of Clusters: 2

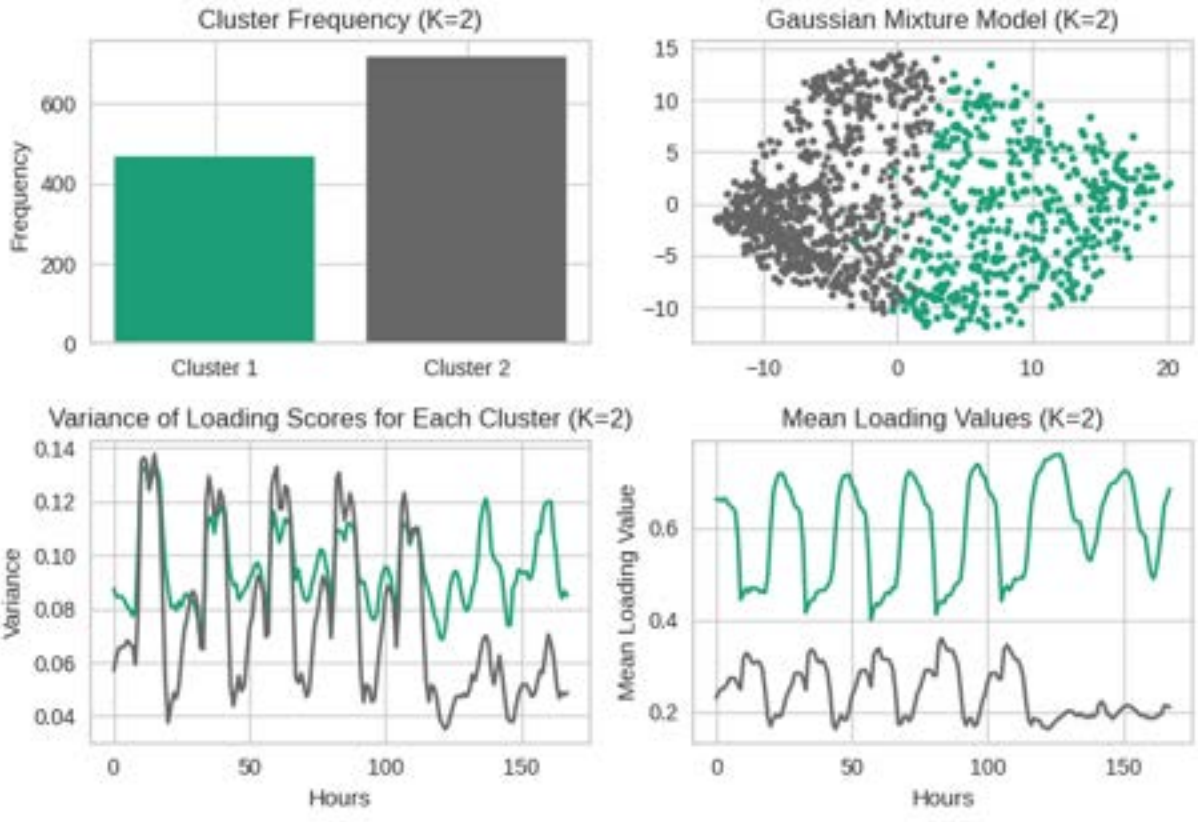
According to both methods, the optimal K value is 2.

3.3.2. Visualization and interpretation of GMM

3.3.2.1 Four descriptive plots of cluster

In [60]:
```
K = 2
cmap = plt.get_cmap('Dark2', K)
x = np.arange(0, 168)

# Gaussian Mixture Model
gmm = GaussianMixture(n_components=K, n_init=3)
clusters_gmm = gmm.fit_predict(loading)
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_freq = np.unique(clusters_gmm, return_counts=True)

# Plot Cluster Frequency
plt.subplot(2, 2, 1)
plt.bar(cluster_names, cluster_freq[1], color=cmap.colors)
plt.ylabel("Frequency")
plt.title(f"Cluster Frequency (K={K})")


# Scatter plot of GMM results
plt.subplot(2, 2, 2)
```

```
plt.scatter(loading_pca[:, 0], loading_pca[:, 1], c=clusters_gmm, s=7, linew
plt.grid(True)
plt.title(f'Gaussian Mixture Model (K={K})')

# Plot the new type of plot (replace 'plot 3' with the appropriate title)
x = np.arange(loading.shape[1])
for j in range(K):
    plt.subplot(2, 2, 3)
    plt.plot(x, np.var(loading[clusters_gmm == j], axis=0), color=cmap.color

plt.xlabel("Hours")
plt.ylabel("Variance")
plt.title(f"Variance of Loading Scores for Each Cluster (K={K})")

# Compute and plot mean loading values
x = np.arange(loading.shape[1])  # Assuming loading is your data matrix
for j in range(K):
    plt.subplot(2, 2, 4)
    plt.plot(x, np.mean(loading[clusters_gmm == j], axis=0), color=cmap.colo
plt.xlabel("Hours")
plt.ylabel("Mean Loading Value")
plt.title(f"Mean Loading Values (K={K})")


plt.tight_layout()
plt.show()
```



Interpretations:

First of all, we note that the variance for each cluster is smaller than 0.15, meaning that

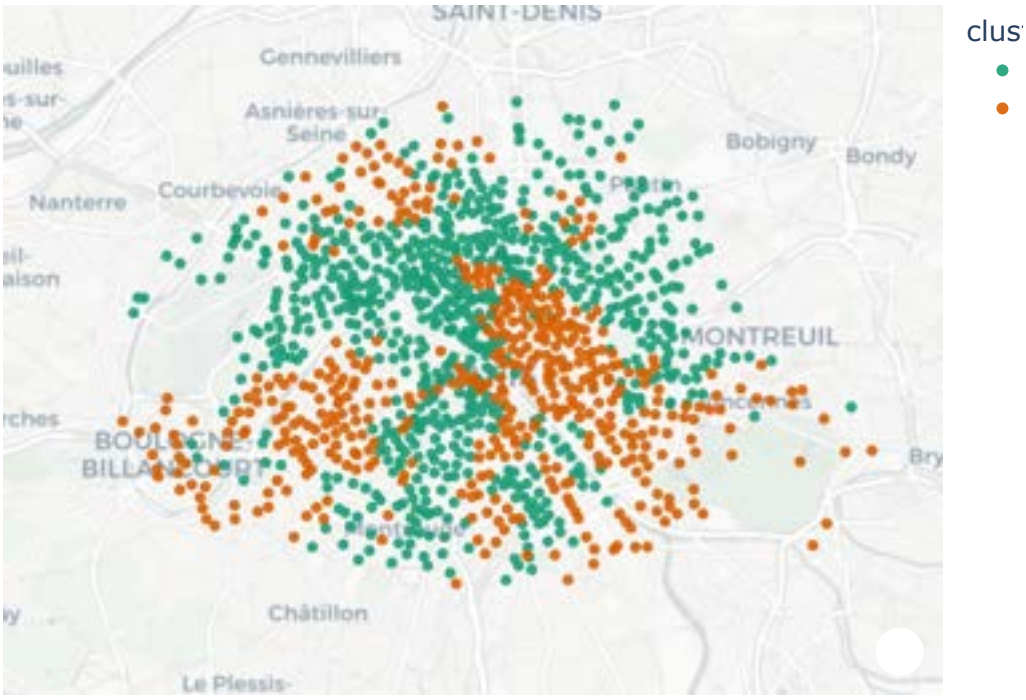
plotting the mean loading values is appropriate.

- k=2 Cluster2 correspond to the stations mostly empty during all week. While, cluster1 correspond to the stations mostly full during all week.

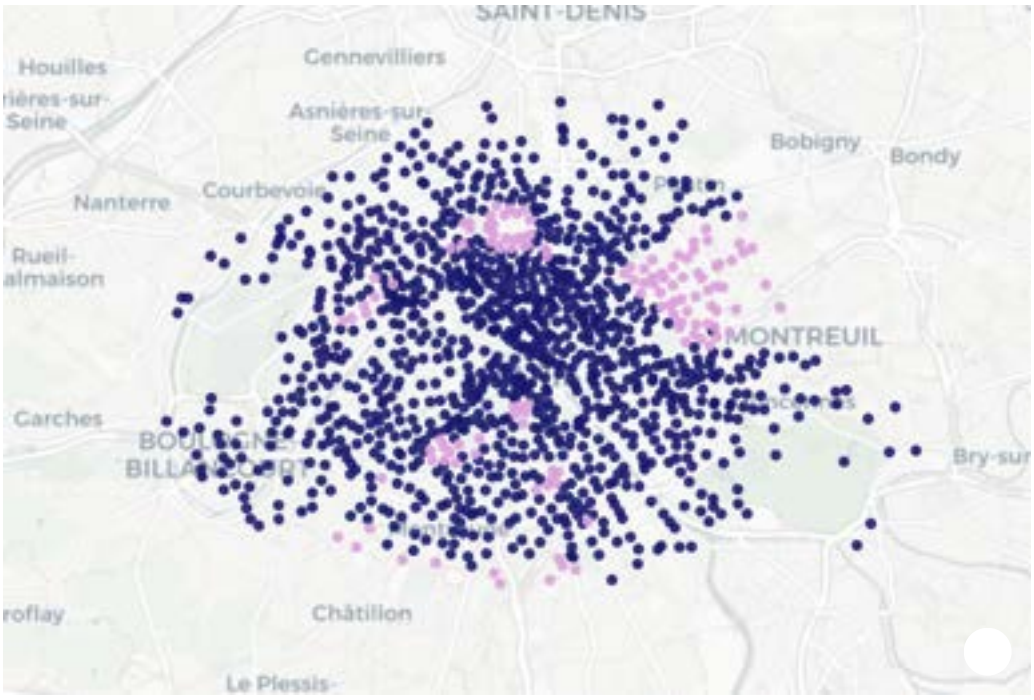
3.3.2.2. Visualization of clusters on the map

In [61]:
```
K = 2
# Gaussian Mixture Model
gmm = GaussianMixture(n_components=K, n_init=3)
clusters_gmm = gmm.fit_predict(loading)

# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_gmm]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

fig2.show()
```

Stations by Clusters



Hilltop stations



Previously we said that Cluster1 correspond to the stations mostly full during all week. Those stations correspond to the ones placed on a hill and the suburbs. This makes sense as those stations are the less visited.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

3.4. Comparison of clustering methods

The purpose of this last section is to compare the different results we obtained previously.

In [62]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

The function `matchClasses` reorders the classes found in the two classifications so that they match.This function returns the rearranged confusion matrix and the associated class indices for each point of the dataset.

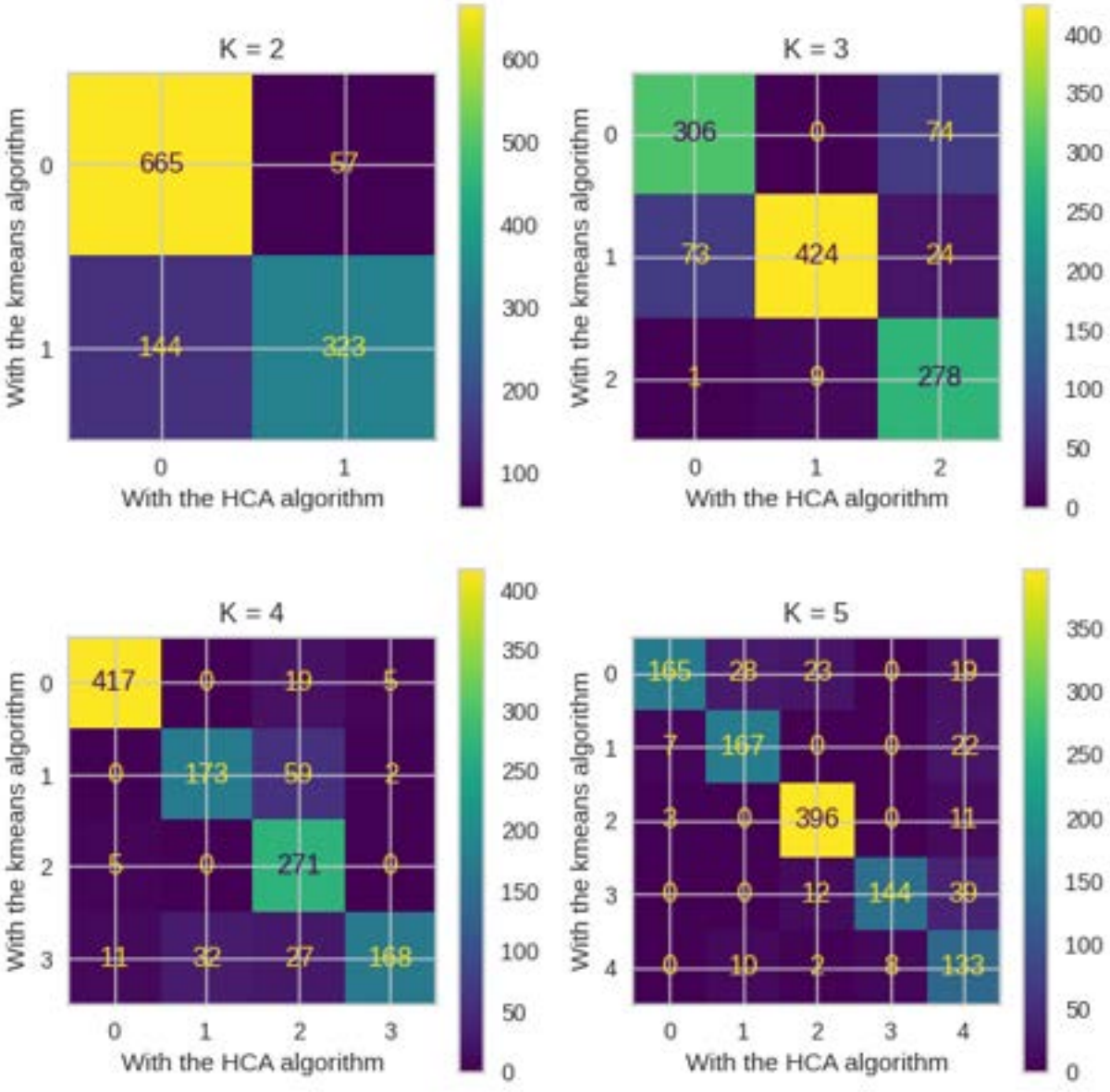
In [63]:
```
def matchClasses(classif1, classif2):
```

```
    cm = confusion_matrix(classif1, classif2)
    K = cm.shape[0]
    a, b = np.zeros(K), np.zeros(K)
    for j in range(K):
        for i in range(K):
            if (a[j] < cm[i,j]):
                a[j] = cm[i,j]
                b[j] = i
    a = a.astype(int)
    b = b.astype(int)

    print ("")
    print ("Classes size:", a)
    print ("Class (in the classif1 numbering):", b)
    print ("")

    table = cm.copy()
    for i in range(K):
        table[:,b[i]] = cm[:,i]

    clusters = classif2.copy()
    n = classif2.shape[0]
    for i in range(n):
        for j in range(K):
            if (classif2[i] == j):
                clusters[i] = b[j]

    return table, clusters
```

3.4.1. HCA vs. k-means

In [64]:
```
K_values = [2, 3, 4, 5]

fig, axs = plt.subplots(2, 2, figsize=(7, 7))

for i, K in enumerate(K_values):
    kmeans = KMeans(n_clusters=K, n_init=10)
    clusters_kmeans_loading = kmeans.fit_predict(loading)

    HCA = AgglomerativeClustering(n_clusters=K, compute_distances=True, link
    clusters_HCA = HCA.fit_predict(loading)

    cm, clusters_kmeans_loading_sorted = matchClasses(clusters_kmeans_loadin

    row = i // 2
    col = i % 2

    ConfusionMatrixDisplay(cm).plot(ax=axs[row, col])
    axs[row, col].set_xlabel('With the HCA algorithm')
    axs[row, col].set_ylabel('With the kmeans algorithm')
    axs[row, col].set_title(f'K = {K}')

plt.tight_layout()
plt.show()
```

```
Classes size: [665 323]
Class (in the classif1 numbering): [0 1]


Classes size: [306 278 424]
Class (in the classif1 numbering): [0 2 1]


Classes size: [271 173 417 168]
Class (in the classif1 numbering): [2 1 0 3]


Classes size: [396 167 133 165 144]
Class (in the classif1 numbering): [2 1 4 0 3]
```

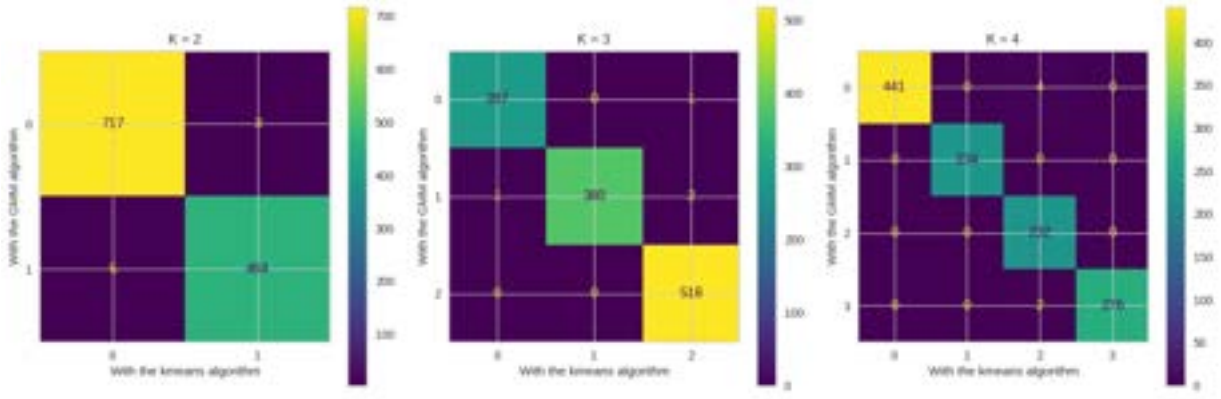


The four matrices are not diagonal, meaning that the cluster made by both methods are different.

3.4.2. GMM vs. k-means

In [65]:
```
K_values = [2, 3, 4]

fig, axs = plt.subplots(1, 3, figsize=(15, 5))

for i, K in enumerate(K_values):
    kmeans = KMeans(n_clusters=K, n_init=10)
    clusters_kmeans_loading = kmeans.fit_predict(loading)

    gmm = GaussianMixture(n_components=K, n_init=10, init_params='kmeans')
    clusters_gmm = gmm.fit_predict(loading)

    cm, clusters_kmeans_loading_sorted = matchClasses(clusters_gmm, clusters

    ConfusionMatrixDisplay(cm).plot(ax=axs[i])
    axs[i].set_xlabel('With the kmeans algorithm')
    axs[i].set_ylabel('With the GMM algorithm')
    axs[i].set_title(f'K = {K}')

plt.tight_layout()
plt.show()
```

```
Classes size: [464 717]
Class (in the classif1 numbering): [1 0]


Classes size: [518 380 287]
Class (in the classif1 numbering): [2 1 0]


Classes size: [234 441 276 232]
Class (in the classif1 numbering): [1 0 3 2]
```



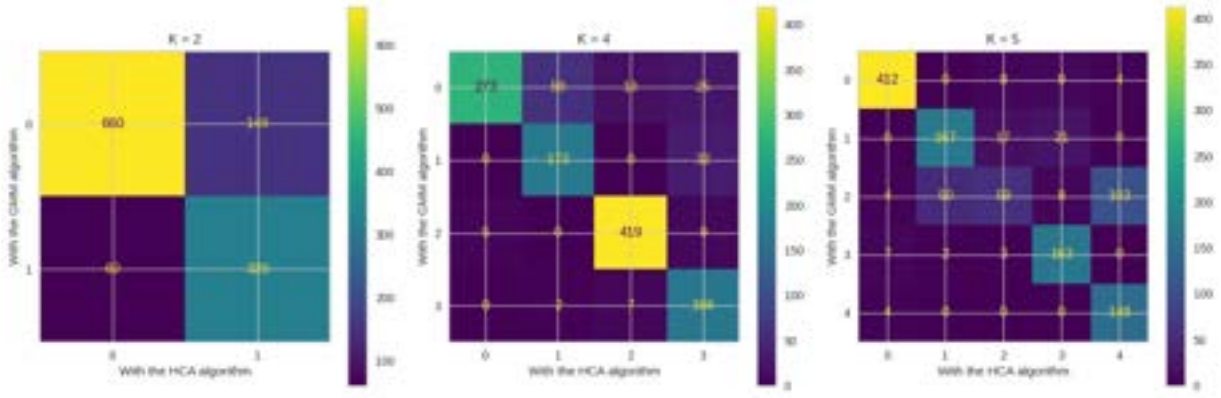
The three matrices are diagonal, meaning that the cluster made by both methods are similar.

3.4.3. HCA versus GMM

In [66]:
```
K_values = [2, 4, 5]
```

```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

for i, K in enumerate(K_values):
    HCA = AgglomerativeClustering(n_clusters=K, compute_distances=True, link
    clusters_HCA = HCA.fit_predict(loading)

    gmm = GaussianMixture(n_components=K, n_init=10, init_params='kmeans')
    clusters_gmm = gmm.fit_predict(loading)

    cm, clusters_kmeans_loading_sorted = matchClasses(clusters_HCA, clusters

    ConfusionMatrixDisplay(cm).plot(ax=axs[i])
    axs[i].set_xlabel('With the HCA algorithm')
    axs[i].set_ylabel('With the GMM algorithm')
    axs[i].set_title(f'K = {K}')

plt.tight_layout()
plt.show()
```

```
Classes size: [660 320]
Class (in the classif1 numbering): [0 1]


Classes size: [273 173 166 419]
Class (in the classif1 numbering): [0 1 3 2]


Classes size: [148 163 167  59 412]
Class (in the classif1 numbering): [4 3 1 2 0]
```



Almost the four matrices are not diagonal, meaning that the cluster made by both methods are different.

To do for further exploration: Correspondance Analysis of non diagonal matrix.

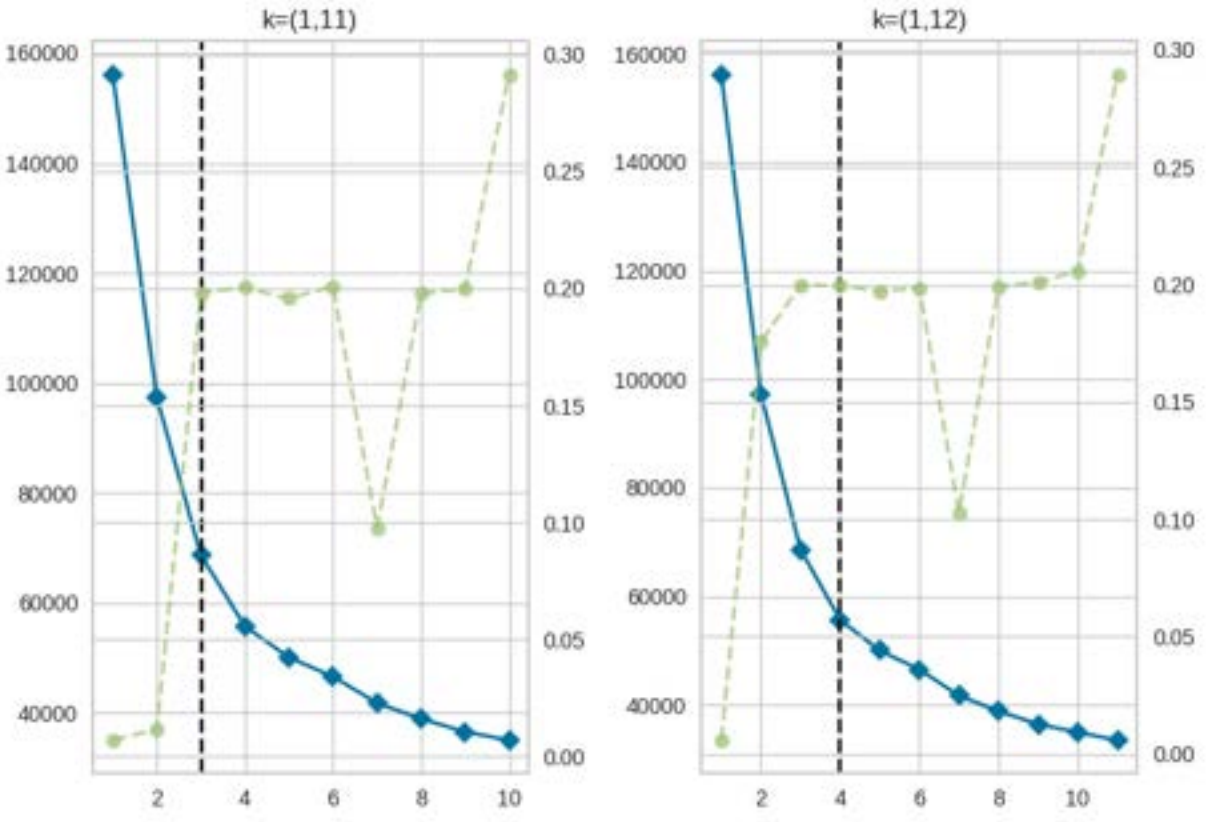
Part 4: Clustering on reduced data

4.1. K-means clustering

4.1.1. Selection of the number of clusters

4.1.1.1. Determining the number of clusters using the total within sum of square metric

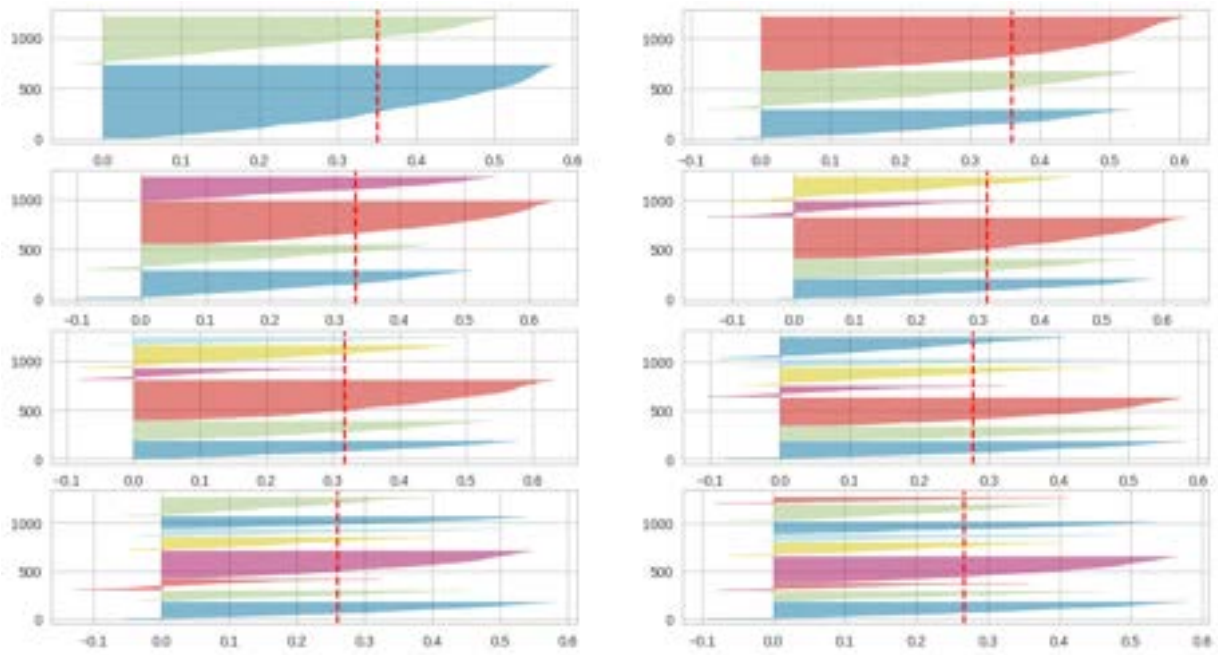
In [67]:
```
kmeans = KMeans(init='k-means++', max_iter=100,  n_init = "auto",random_stat
visualizer_1 = KElbowVisualizer(kmeans, k=(1,11), metric='distortion')
visualizer_2 = KElbowVisualizer(kmeans, k=(1,12), metric='distortion')

plt.subplot(1,2,1)
visualizer_1.fit(loading_pca6)
plt.title("k=(1,11)")

plt.subplot(1,2,2)
visualizer_2.fit(loading_pca6)
plt.title("k=(1,12)")

plt.tight_layout()
plt.show()
```



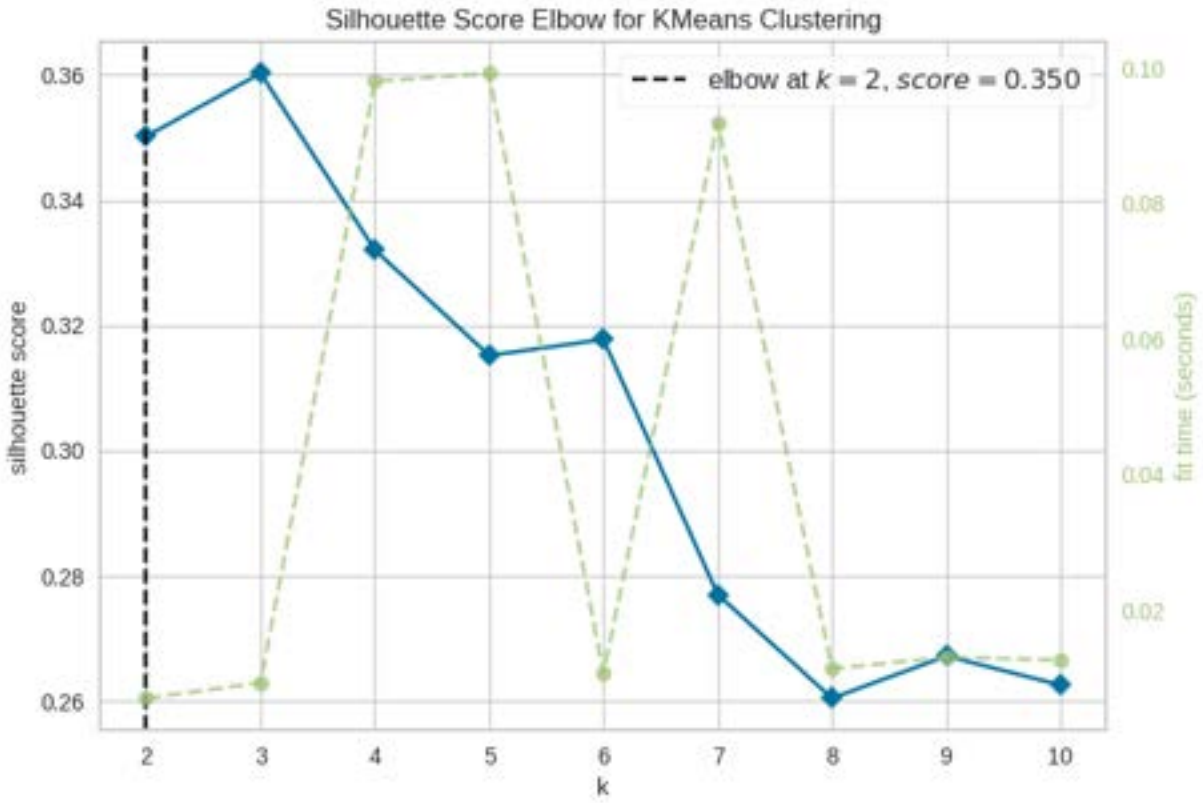
Here we observe an anomaly in the function KElbowVisualizer. When we are looking for the best number of clusters in two different intervals we get different values. Since we do not know which one is better we will perform the k-means clustering with k=3 and k=4 to see which one is actually better.

4.1.1.2. Determining the number of clusters using the silhouette scores metric

In [68]:
```
fig, ax = plt.subplots(4, 2, figsize=(15,8))

for k in range(2, 10):
    kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init = "
    q, mod = divmod(k, 2)

    # Create SilhouetteVisualizer instance with KMeans instance
    visualizer = SilhouetteVisualizer(kmeans, colors='yellowbrick', ax=ax[q-
    visualizer.fit(loading_pca6)
```



In [69]:
```
visualizer = KElbowVisualizer(kmeans, k=(2,11),metric='silhouette')

visualizer.fit(loading_pca6)
visualizer.show()
```



Out[69]:
```
<Axes: title={'center': 'Silhouette Score Elbow for KMeans Clustering'}, xl
abel='k', ylabel='silhouette score'>
```

According to the distortion score elbow method, the best K values are either 3 or 4.

For the silhouette score elbow method, the optimal K value is 2.

We will further investigate the number of clusters using K values of 2, 3, and 4.

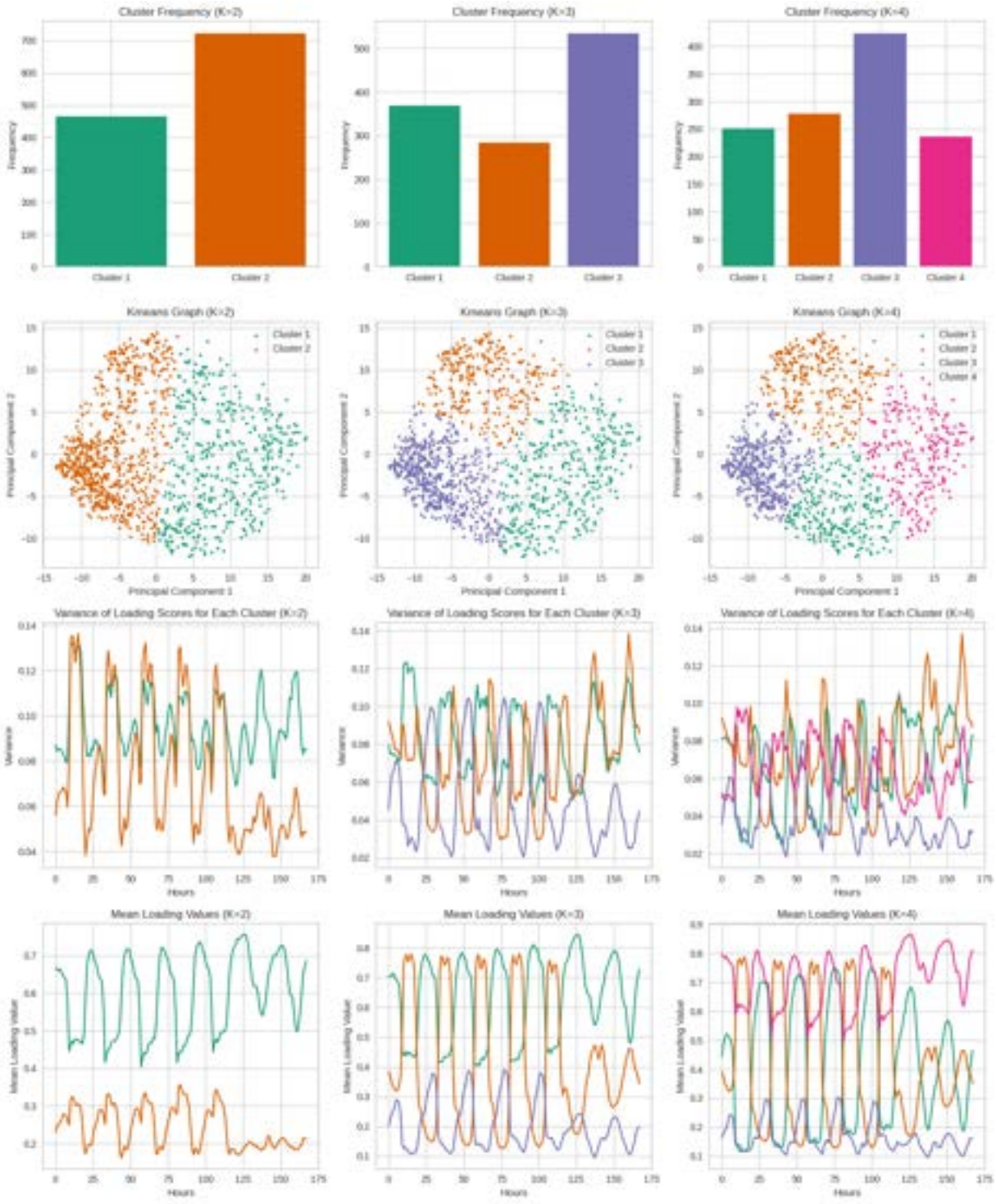
Interesting note: We found the same values of K with the raw data.

4.1.2. Visualization and interpretation of k-means clusters

4.1.2.1. Four descriptive plots of cluster

In [70]:
```
K_values = [2, 3, 4]
cmap = plt.get_cmap('Dark2')
fig, axs = plt.subplots(4, len(K_values), figsize=(15, 18))

# Loop through each value of K
for i, K in enumerate(K_values):
    # Perform K-means clustering
    kmeans_pca = KMeans(n_clusters=K, init='k-means++',  n_init = "auto", ra
    clusters_pca = kmeans_pca.fit_predict(loading_pca6)
    cluster_freq = np.unique(clusters_pca, return_counts=True)
    cluster_names = [f"Cluster {i+1}" for i in range(K)]

    # Plot Cluster Frequency
    axs[0, i].bar(cluster_names, cluster_freq[1], color=cmap.colors)
    axs[0, i].set_ylabel("Frequency")
```

```
    axs[0, i].set_title(f"Cluster Frequency (K={K})")

    # Plot Kmeans Graph
    for j in range(K):
        axs[1, i].scatter(loading_pca[clusters_pca == j, 0], loading_pca[clu
    axs[1, i].set_title(f"Kmeans Graph (K={K})")
    axs[1, i].legend()
    axs[1, i].set_xlabel("Principal Component 1")
    axs[1, i].set_ylabel("Principal Component 2")

    # Plot the new type of plot (replace 'plot 3' with the appropriate title
    x = np.arange(loading.shape[1])
    for j in range(K):
        axs[2, i].plot(x, np.var(loading[clusters_pca == j], axis=0), color=

    axs[2, i].set_xlabel("Hours")
    axs[2, i].set_ylabel("Variance")
    axs[2, i].set_title(f"Variance of Loading Scores for Each Cluster (K={K}

    # Plot mean loading values
    x = np.arange(loading.shape[1])
    for j in range(K):
        axs[3, i].plot(x, np.mean(loading[clusters_pca == j], axis=0), color

    axs[3, i].set_xlabel("Hours")
    axs[3, i].set_ylabel("Mean Loading Value")
    axs[3, i].set_title(f"Mean Loading Values (K={K})")



# Adjust layout and display the plot
plt.tight_layout()
plt.show()
```



Interpretations:

First of all, we note that the variance for each cluster is smaller than 0.15 which we deem small enough to plot the mean loading scores.

- k=2 According to the fourth graph, cluster1 corresponds to the stations that are mostly full throughout the week while cluster2 corresponds to the stations that are mostly empty throughout the week. Additionnaly, on the second graph we can see that the clusters are clearly delimited by the component1 axis ie. cluster1 is exclusively represented with a positive component1 and cluster2 with a negative

component1. According to the PCA anaysis we performed earlier we excpect the stations on hills to only be present in cluster2.

- k=3 Once again, cluster1 corresponds to the stations that are mostly full during the week and cluster3 to the emptier ones. According to the graph of individuals we can see that cluster2 is represented on the positive part of the second component. According to the PCA analysis this means that the stations in cluster2 are full during the day. This seems to be confirmed by the final graph. In addition, we expect the hills to be in cluster 3.

- k=4 This clusters distinguishes two groups well defined by component2, cluster1 and cluster2. According to our study of PCA components, the stations in cluster2 are full during day hours and those in cluster1 are full during night hours. This conclusion is coherent with the fourth plot! Cluster4 corresponds to the stations that are mostly full throughout the week while cluster3 corresponds to the stations that are mostly empty throughout the week.

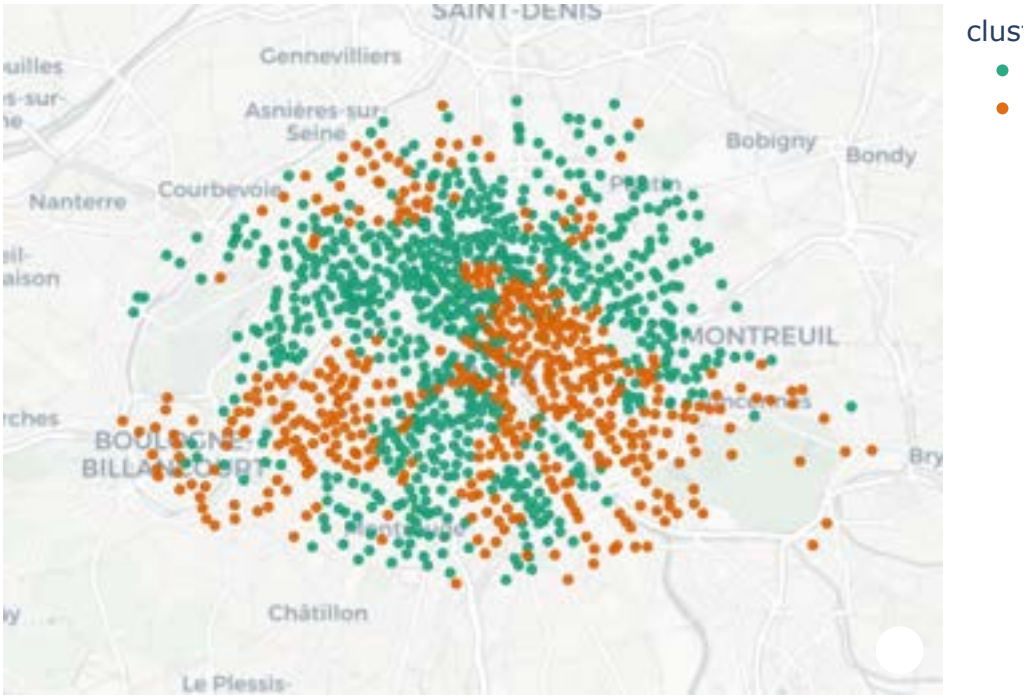
4.1.2.2. Visualization of clusters on the map

We will now compare the stations colored by clusters and the stations colored by hill position to see if they match.

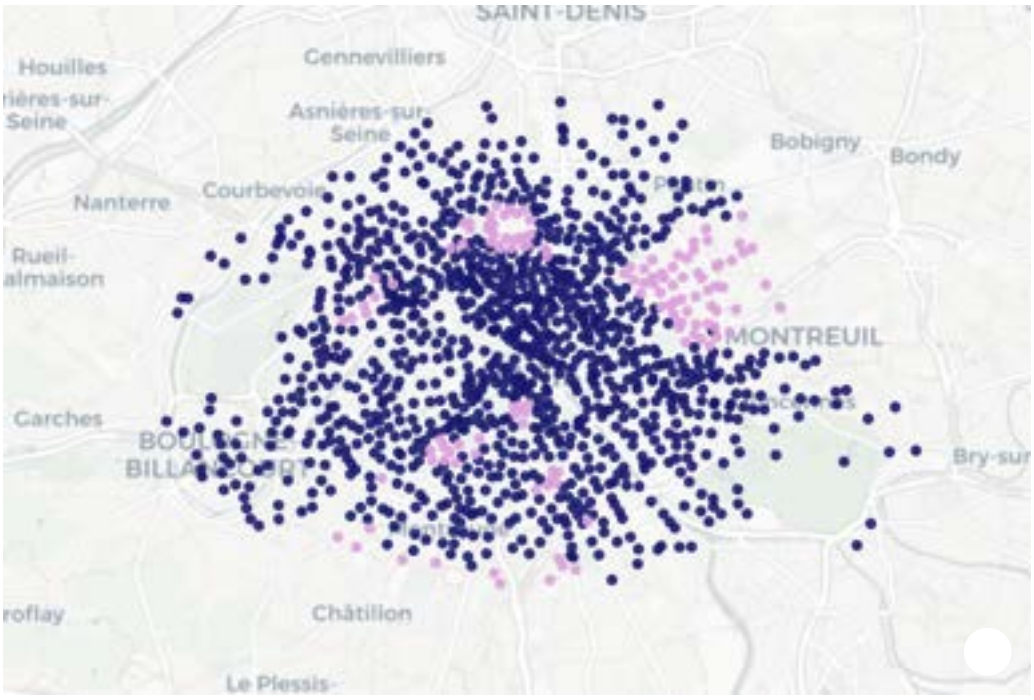
In [71]:
```
K = 2

# Perform KMeans clustering
kmeans_pca = KMeans(n_clusters=K, init='k-means++', n_init = "auto", random_
clusters_pca = kmeans_pca.fit_predict(loading_pca6)

# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_pca]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
```

```
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



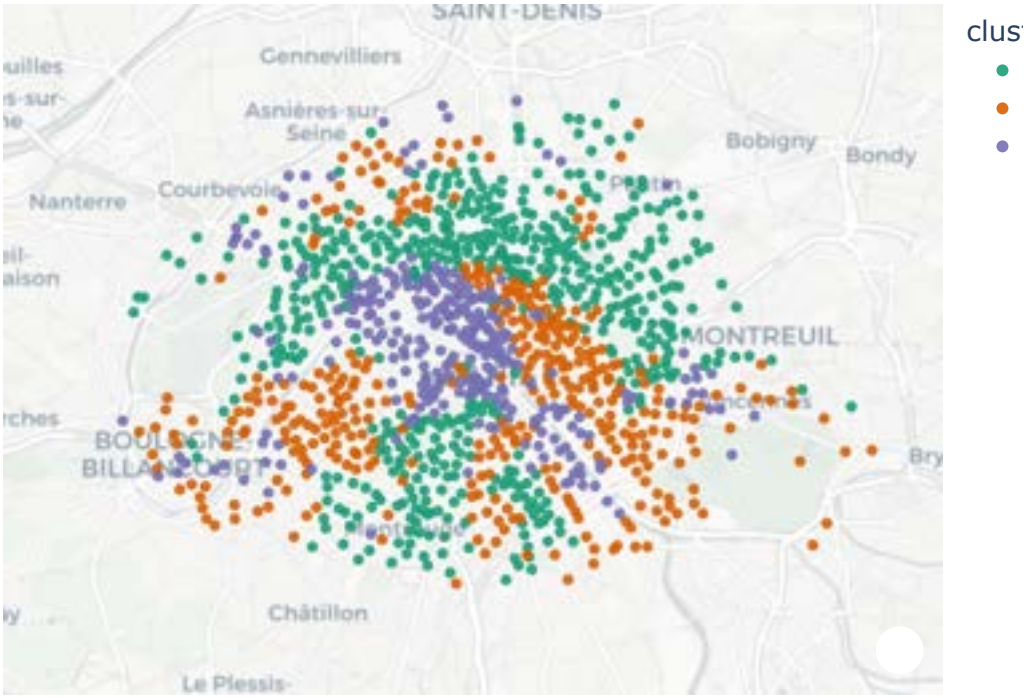
We excpected the stations on hills to only be present in cluster2 which is the case.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

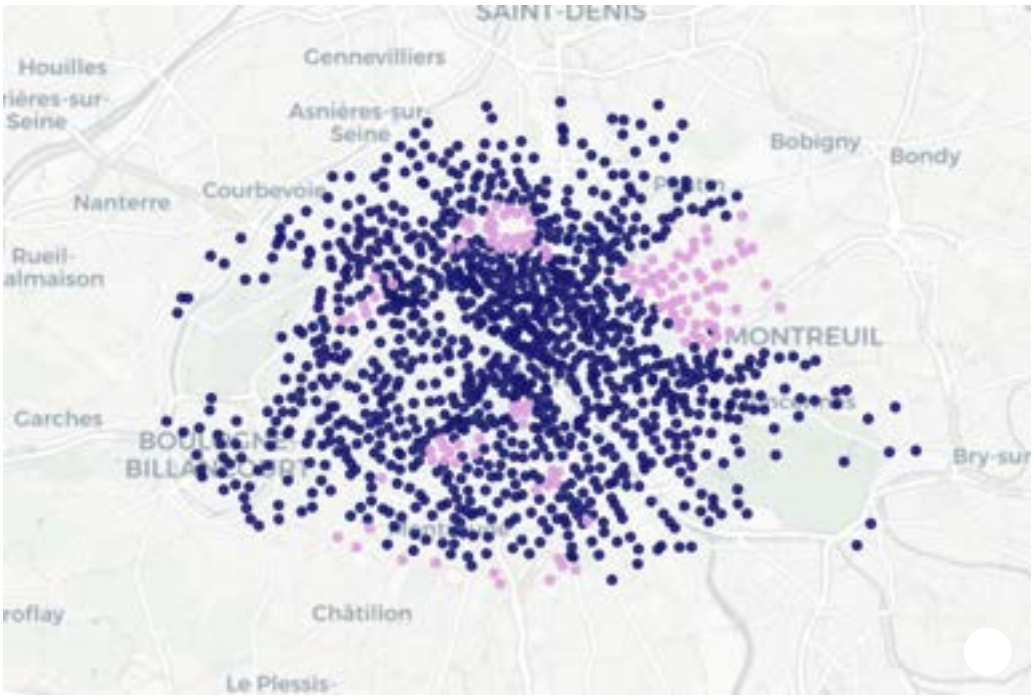
In [72]:
```
K = 3

# Perform KMeans clustering
kmeans_pca = KMeans(n_clusters=K, init='k-means++',  n_init = "auto", random
clusters_pca = kmeans_pca.fit_predict(loading_pca6)

# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_pca]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
```

```
                        title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                        color_discrete_map={0: 'midnightblue', 1: 'plum'},
                        mapbox_style="carto-positron", zoom=10, opacity=0.9,
                        title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



The first thing we notice is that cluster2 corresponds to the stations in the city center. Additionnaly, we saw earlier that the stations in cluster2 are full during the daytime. This means that during the day there are a lot of velibs in the city center. This makes sense. People could go into the city center center during the day for a number of reasons (work, school, shopping, sightseeing, leisure and enjoying the weather, appointments, etc.). At the end of the day everyone would then go home, probably in the suburbs, with velibs, leaving the stations empty.

We expected the hills to be in cluster 3 and with this plot we verify that the stations on hill are in cluster3.

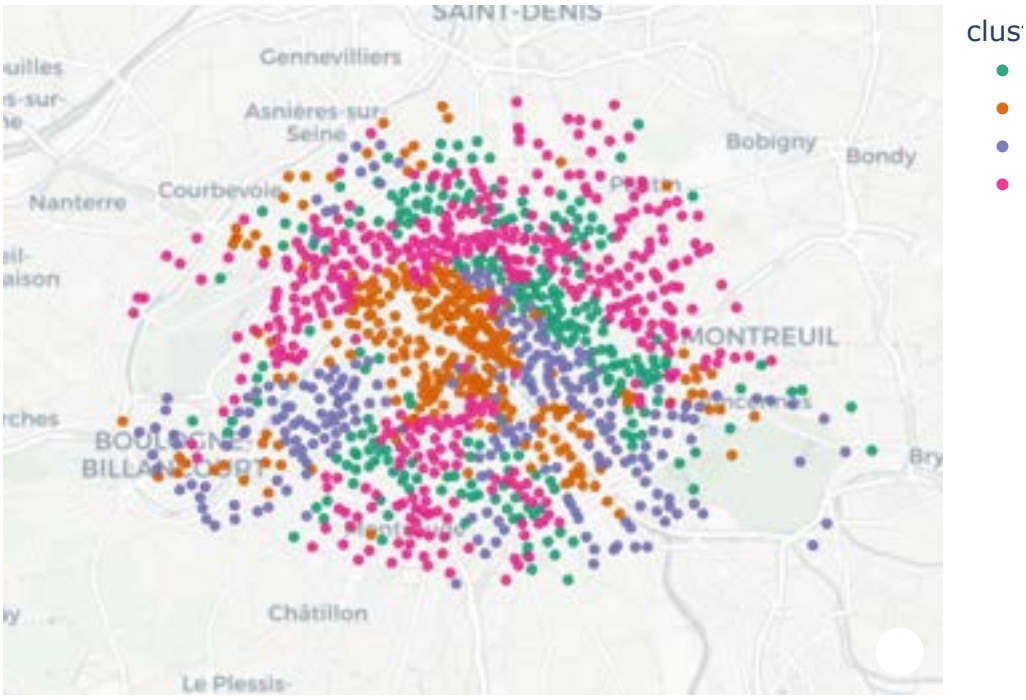
The stations in cluster1 are still relatively in the city center but in a zone where housing is more affordable. This could explain why stations in that cluster are on average full no matter the time of day as opposed to the hyper-center of Paris.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

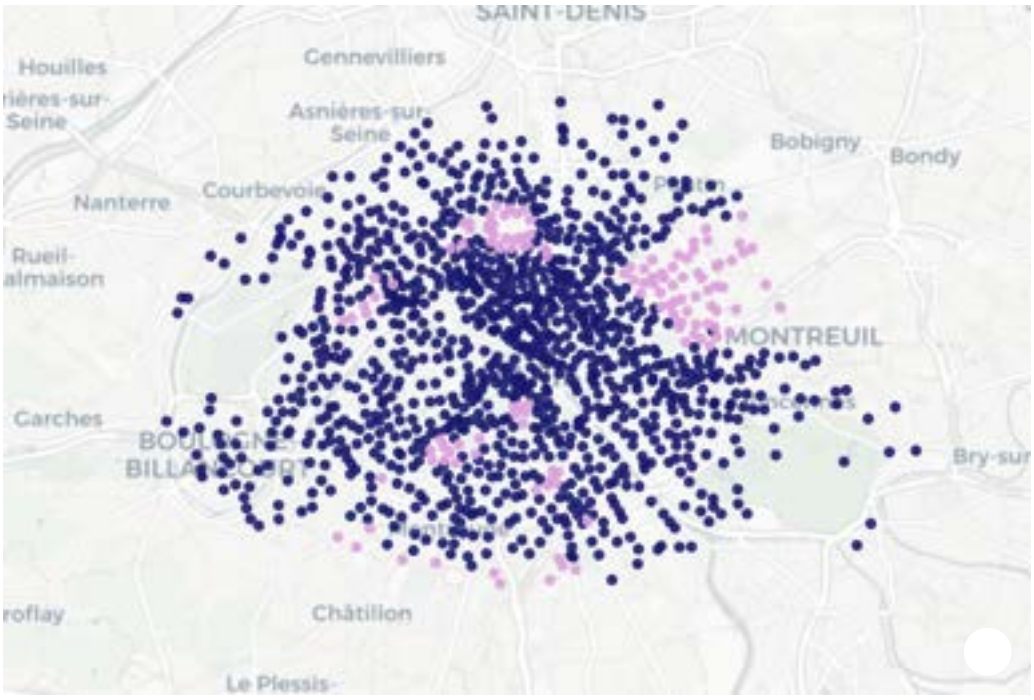
In [73]:
```
K = 4
```

```
# Perform KMeans clustering
kmeans_pca = KMeans(n_clusters=K, init='k-means++', n_init = "auto", random_
clusters_pca = kmeans_pca.fit_predict(loading_pca6)

# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_pca]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



Again, we observe that cluster3 corresponds approximately to the stations placed on a hill and that cluster2 is once again the city center cluster and it is fuller during the day.

According to the previous cell cluster1 is on average fuller at night. With the map we also see that it is in housing neighborhoods, meaning that during the night the stations are full because of all the bikes brougth after work.

The stations in cluster4 are still relatively in the city center but in a zone where housing is more affordable. This could explain why stations in that cluster are on average full no matter the time of day as opposed to the hyper-center of Paris.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

- Note: For all values of k, the interpretations using the projected data are the same as with the original data.

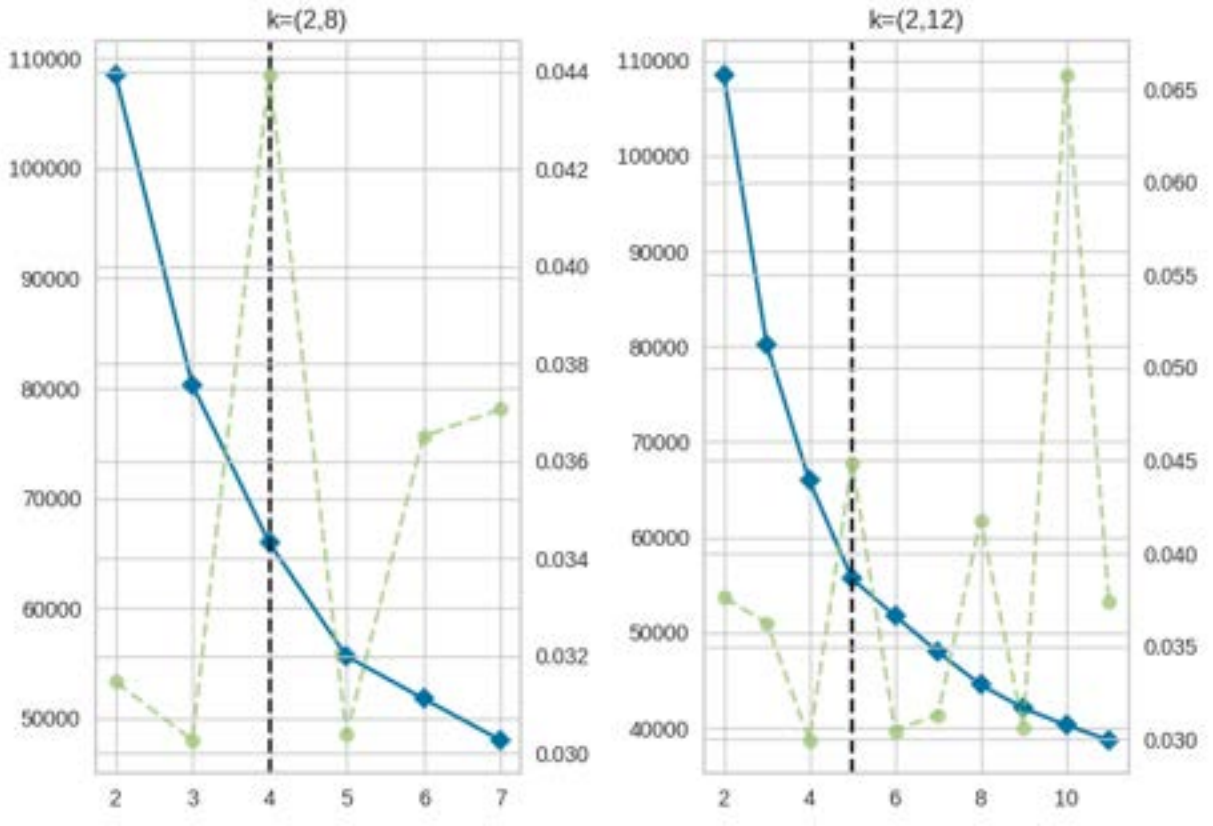
4.2. HCA clustering

In this section, we will perform the HCA to make the same analysis as in th preview section.

4.2.1. Selection of the number of clusters

4.2.1.1. Determining the number of clusters using the total within sum of square metric

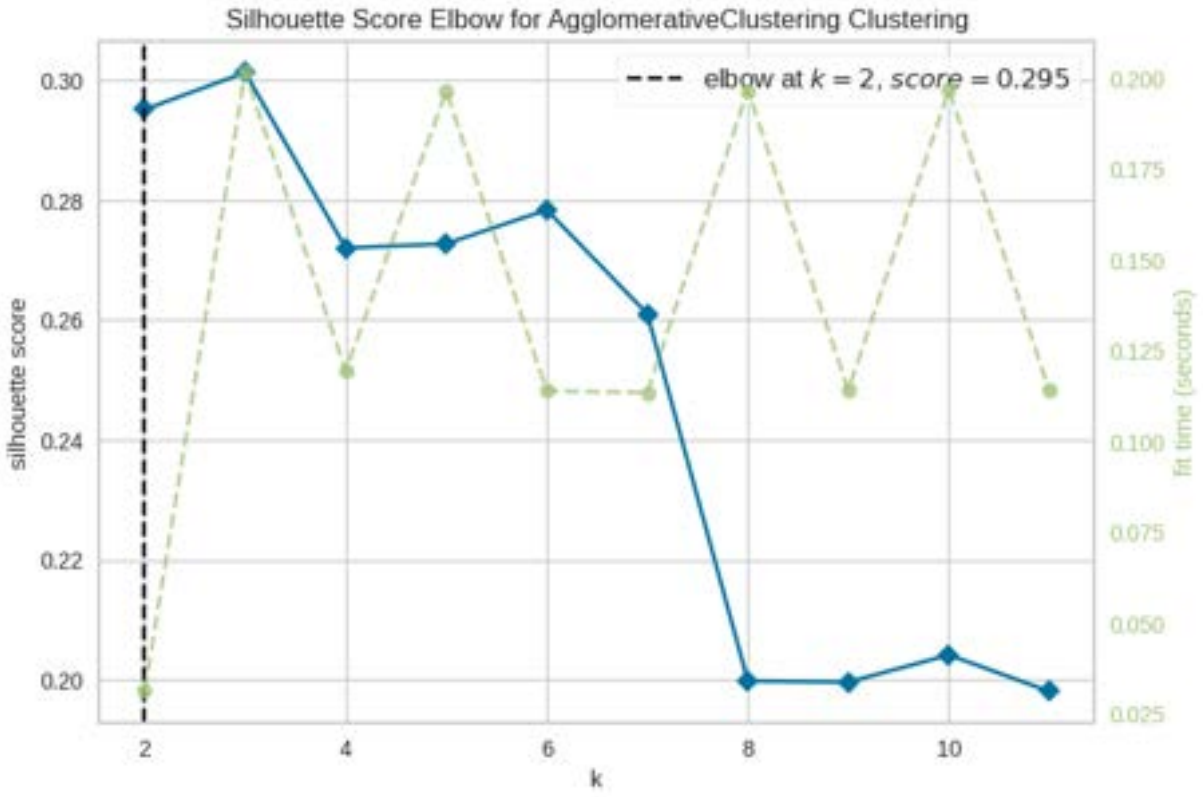
In [74]:
```
ac = AgglomerativeClustering(linkage='ward', compute_distances=True)
visualizer_1 = KElbowVisualizer(ac, k=(2,8), metric='distortion')
visualizer_2 = KElbowVisualizer(ac, k=(2,12), metric='distortion')

plt.subplot(1,2,1)
visualizer_1.fit(loading_pca6)  # Fit the data to the visualizer
plt.title("k=(2,8)")

plt.subplot(1,2,2)
visualizer_2.fit(loading_pca6)  # Fit the data to the visualizer
plt.title("k=(2,12)")

plt.tight_layout()
plt.show()
```



We observe two different best values of k depending on the range of values we give to the elbow algorithm. For k=(2,8) the best value is k_best=4, but for For k=(2,12) the best value is k_best=5. We cannot conclude of which k value is the best. So now we will use silhouette scores to determine the best number of clusters.

4.2.1.2. Determining the number of clusters using the silhouette metric

In [75]:
```
ac = AgglomerativeClustering(linkage='ward', compute_distances=True)
visualizer = KElbowVisualizer(ac, k=(2,12), metric='silhouette')

visualizer.fit(loading_pca6)  # Fit the data to the visualizer
visualizer.show()
plt.show()
```



According to the distortion score elbow method, the best K values are either 4 or 5.

For the silhouette score elbow method, the optimal K value is 2.

We will further investigate the number of clusters using K values of 2, 4, and 5.

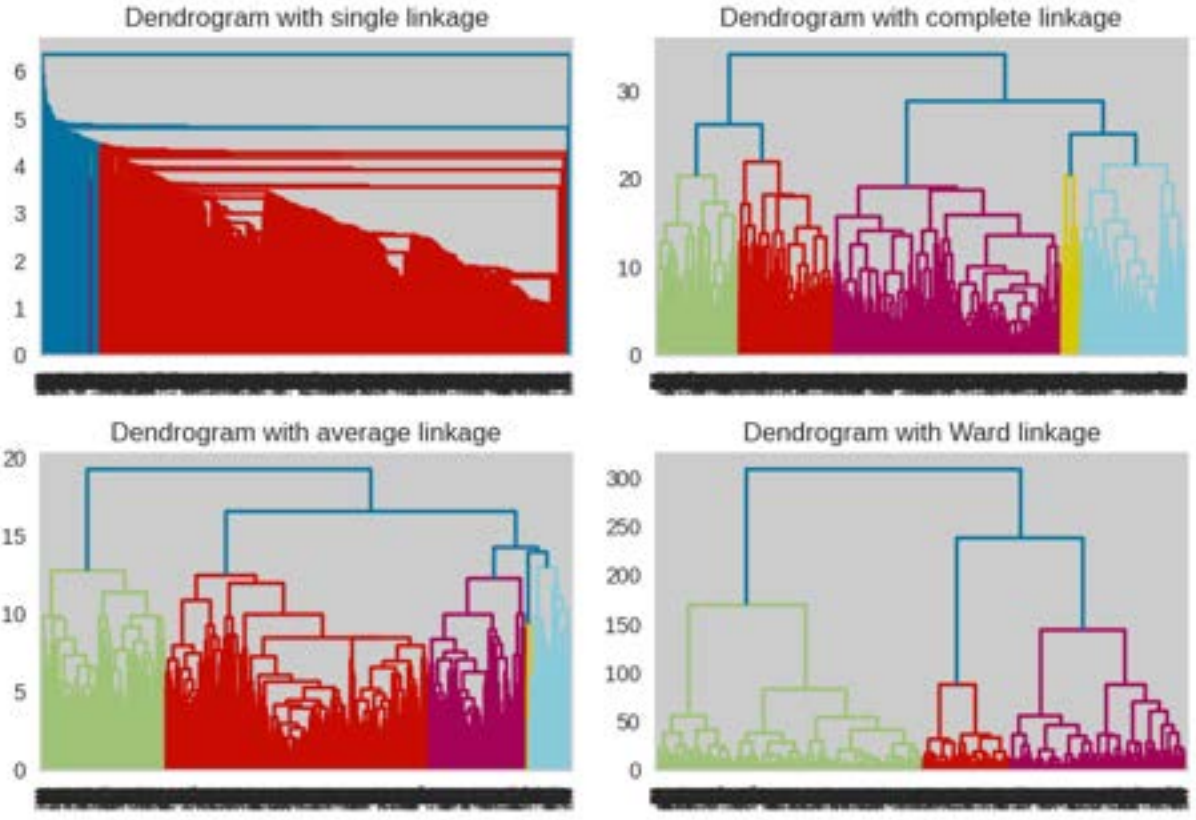
Interesting note: Same results as with original data.

4.2.2. Visualization of different dendrograms and evaluation of the effect of the choice of the linkage function

4.2.2.1. Dendrogram with different linkage methods

In [76]:
```
k=2

plt.subplot(2,2,1)
linkage_matrix_single = sch.linkage(loading_pca6, method='single')
sch.dendrogram(linkage_matrix_single)
plt.title("Dendrogram with single linkage")
```

```
plt.subplot(2,2,2)
linkage_matrix_complete = sch.linkage(loading_pca6, method='complete')
sch.dendrogram(linkage_matrix_complete)
plt.title("Dendrogram with complete linkage")

plt.subplot(2,2,3)
linkage_matrix_average = sch.linkage(loading_pca6, method='average')
sch.dendrogram(linkage_matrix_average)
plt.title("Dendrogram with average linkage")

plt.subplot(2,2,4)
linkage_matrix_ward = sch.linkage(loading_pca6, method='ward')
sch.dendrogram(linkage_matrix_ward)
plt.title("Dendrogram with Ward linkage")

plt.tight_layout()
plt.show()
```

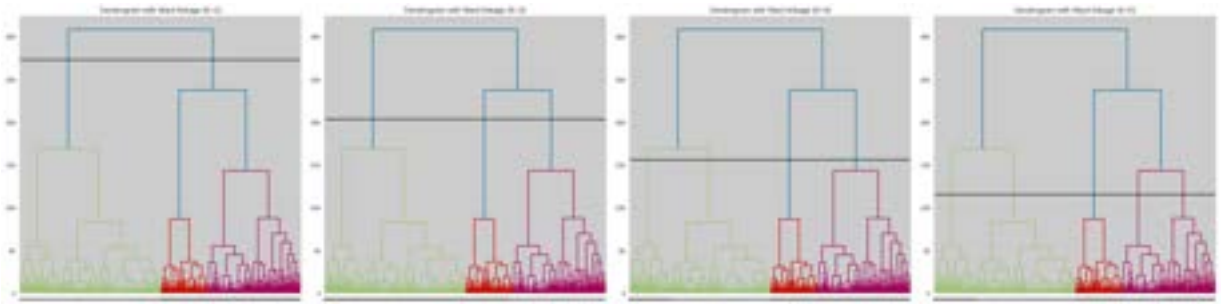


As seen in class, we will use Ward linkage to study the different values of k because this method is clearer than the others.

4.2.2.2. Dendrogram with Ward linkage

In [77]:
```
K_values = [2, 3, 4, 5]
n_rows = 1
n_cols = len(K_values)
fig, axs = plt.subplots(n_rows, n_cols, figsize=(7*n_cols, 7*n_rows))

for i, K in enumerate(K_values):
```

```
    ac = AgglomerativeClustering(n_clusters=K, compute_distances=True, linka
    clusters = ac.fit(loading_pca6)
    children = ac.children_
    distances = ac.distances_
    n_observations = np.arange(2, children.shape[0] + 2)
    linkage_matrix = np.c_[children, distances, n_observations]

    ax = axs[i] if n_rows == 1 else axs[i // n_cols, i % n_cols]

    sch.dendrogram(linkage_matrix, labels=ac.labels_, ax=ax)

    # Cutting the dendrogram to get K classes
    max_d = 0.5 * (ac.distances_[-K] + ac.distances_[-K + 1])
    ax.axhline(y=max_d, c='k')

    ax.set_title(f"Dendrogram with Ward linkage (K={K})")

plt.tight_layout()
plt.show()
```



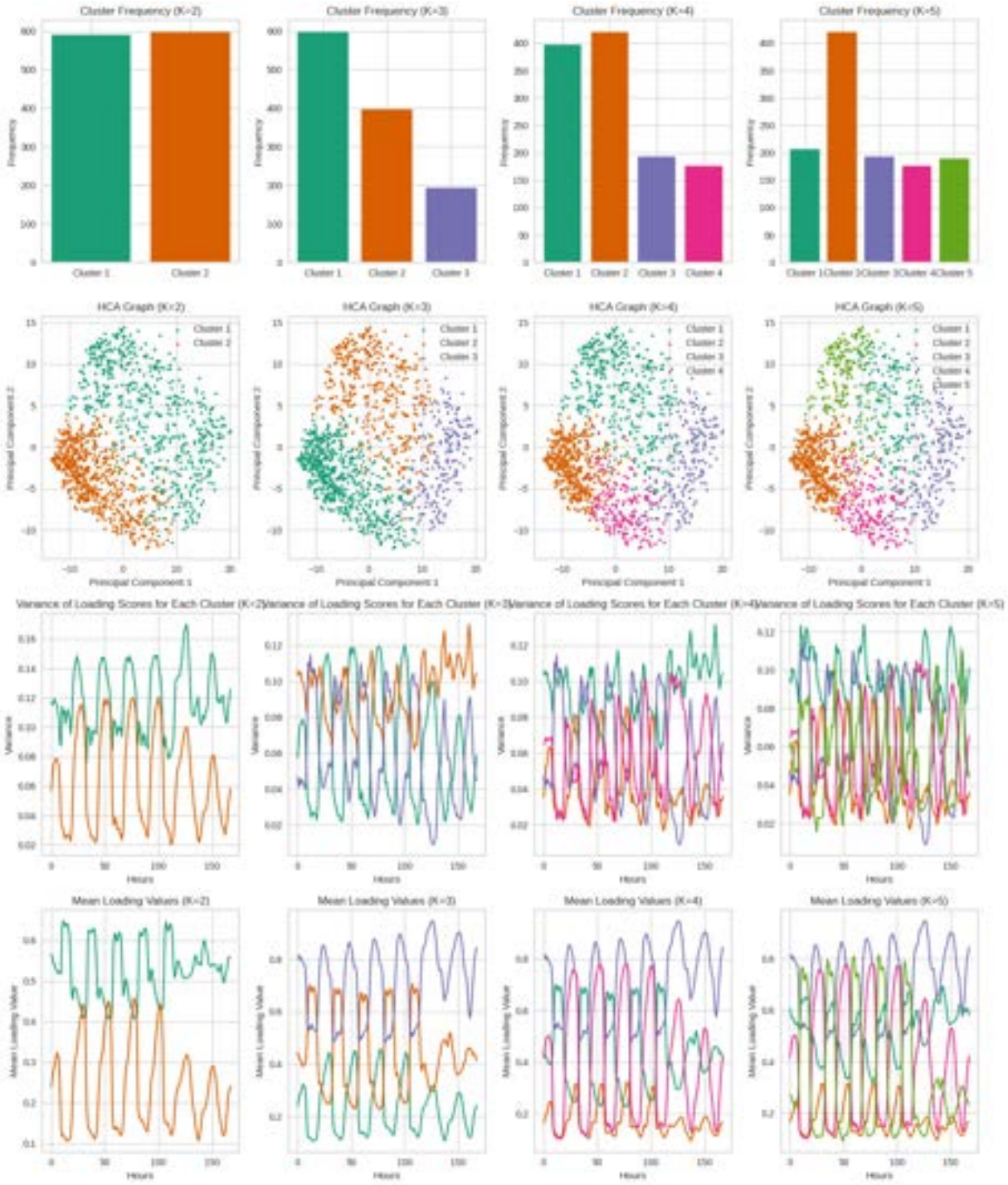
We plotted the Ward linkage graph for k=2, 4, 5. But we see that k=3 seems a reasonable number of clusters, we will the study it too.

4.2.3. Visualization and interpretation of k-means clusters

4.2.3.1. Four descriptive plots of cluster

In [78]:
```
K_values = [2, 3, 4, 5]
cmap = plt.get_cmap('Dark2')
fig, axs = plt.subplots(4, len(K_values), figsize=(15, 18))

for i, K in enumerate(K_values):

    # Agglomerative Clustering
    ac = AgglomerativeClustering(n_clusters=K, compute_distances=True, linka
    clusters_ac = ac.fit_predict(loading_pca6)
    cluster_freq = np.unique(clusters_ac, return_counts=True)
    cluster_names = [f"Cluster {i+1}" for i in range(K)]

    # Plot Cluster Frequency
    axs[0, i].bar(cluster_names, cluster_freq[1], color=cmap.colors)
    axs[0, i].set_ylabel("Frequency")
    axs[0, i].set_title(f"Cluster Frequency (K={K})")

    # Scatter plot of Agglomerative Clustering results
```

```
    for j in range(K):
        axs[1, i].scatter(loading_pca[clusters_ac == j, 0], loading_pca[clus
    axs[1, i].set_title(f"HCA Graph (K={K})")
    axs[1, i].legend()
    axs[1, i].set_xlabel("Principal Component 1")
    axs[1, i].set_ylabel("Principal Component 2")

    # Variance Plot
    x = np.arange(loading.shape[1])
    for j in range(K):
        axs[2, i].plot(x, np.var(loading[clusters_ac == j], axis=0), color=c

    axs[2, i].set_xlabel("Hours")
    axs[2, i].set_ylabel("Variance")
    axs[2, i].set_title(f"Variance of Loading Scores for Each Cluster (K={K}

    # Compute and plot mean loading values
    x = np.arange(loading.shape[1])  # Assuming loading is your data matrix
    for j in range(K):
        axs[3, i].plot(x, np.mean(loading[clusters_ac == j], axis=0), color=

    axs[3, i].set_xlabel("Hours")
    axs[3, i].set_ylabel("Mean Loading Value")
    axs[3, i].set_title(f"Mean Loading Values (K={K})")

plt.tight_layout()
plt.show()
```



Interpretations:

First of all, we note that the variance for each cluster is smaller than 0.15, except for k=2. We will then only study the mean loading scores for the other values of k.

We also notice that the average loadings fluctuate more than with the k-means clustering method. In particular, the behaviours is different on the week-ends. For example, with k=3 clusters 1 and 2 are emptier during the week-ends whereas cluster3 is fuller. This behaviour is also visible with k=4 and k=5.

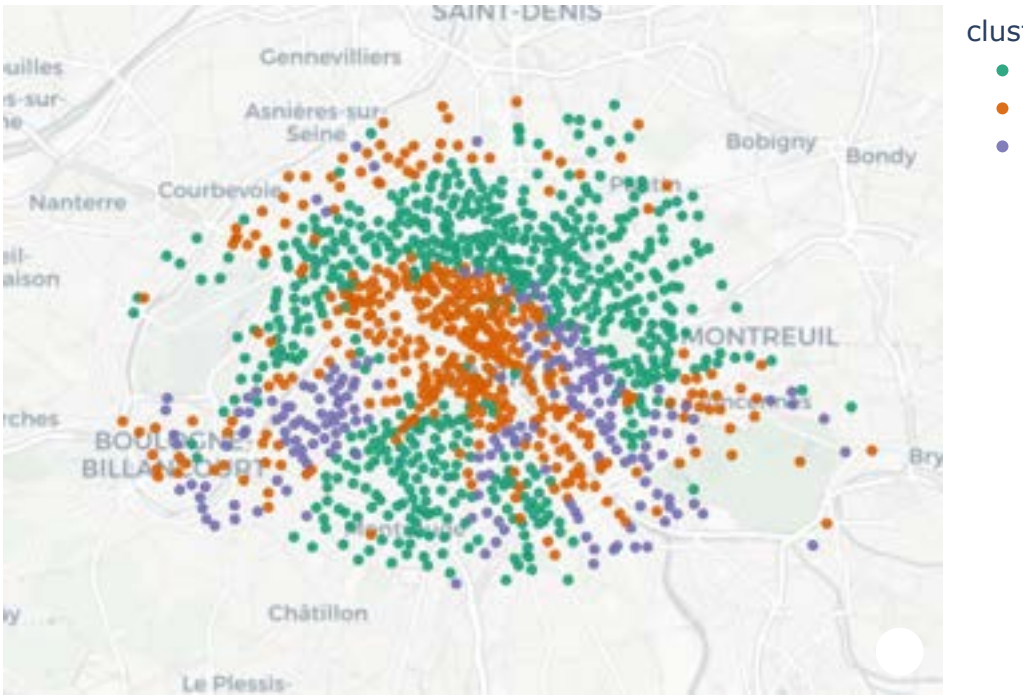
Lastly, we notice that the classes are less balanced than with the k-means algorithm.

That is especially true for k=5.

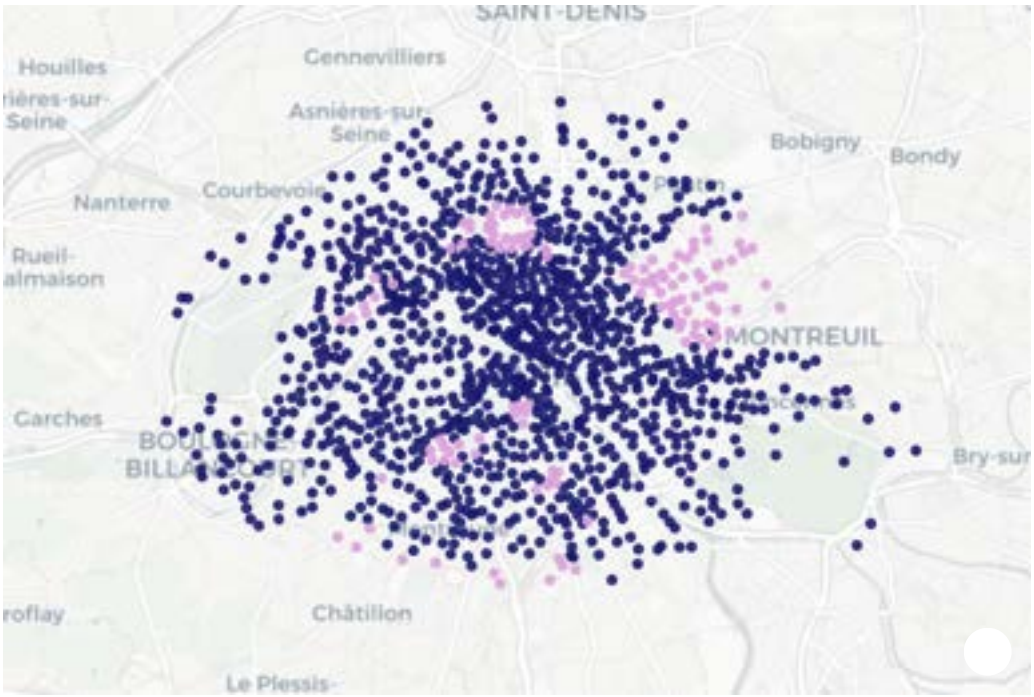
4.2.3.2. Visualization of clusters on the map

In [79]:
```
K = 3

# Perform Agglomerative Clustering
ac = AgglomerativeClustering(n_clusters=K, compute_distances=True, linkage='
clusters_ac = ac.fit_predict(loading_pca6)
# Convert cluster labels to strings for better visualization
cluster_names = ["Cluster " for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_ac]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



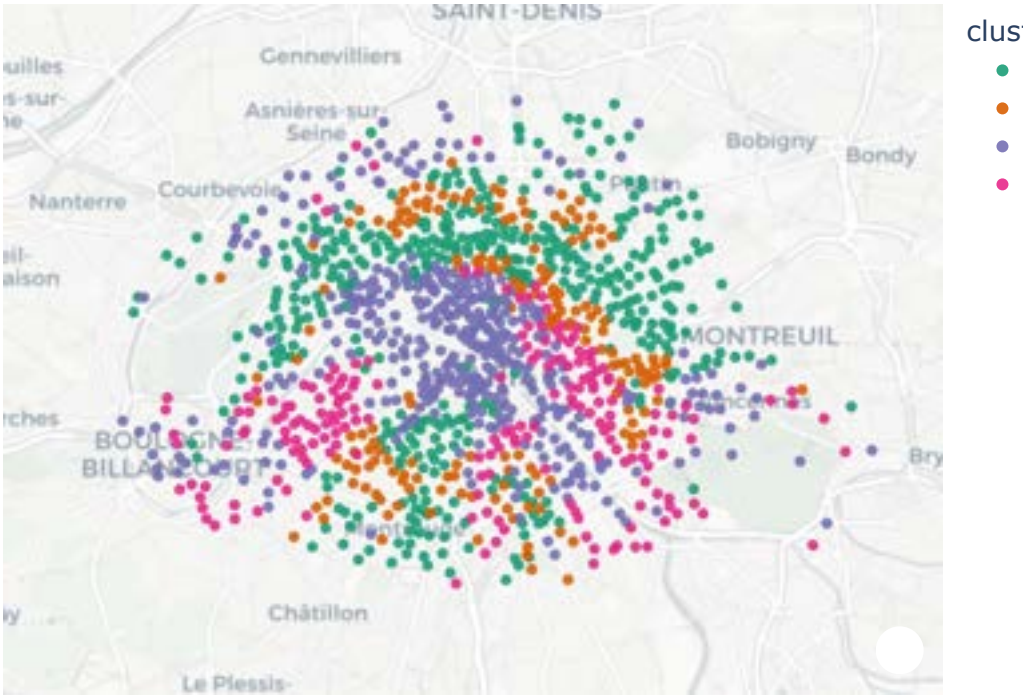
Once again, there is a cluster that is the city-center and the cluster that is emptiest is the one that contains the stations on hills.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

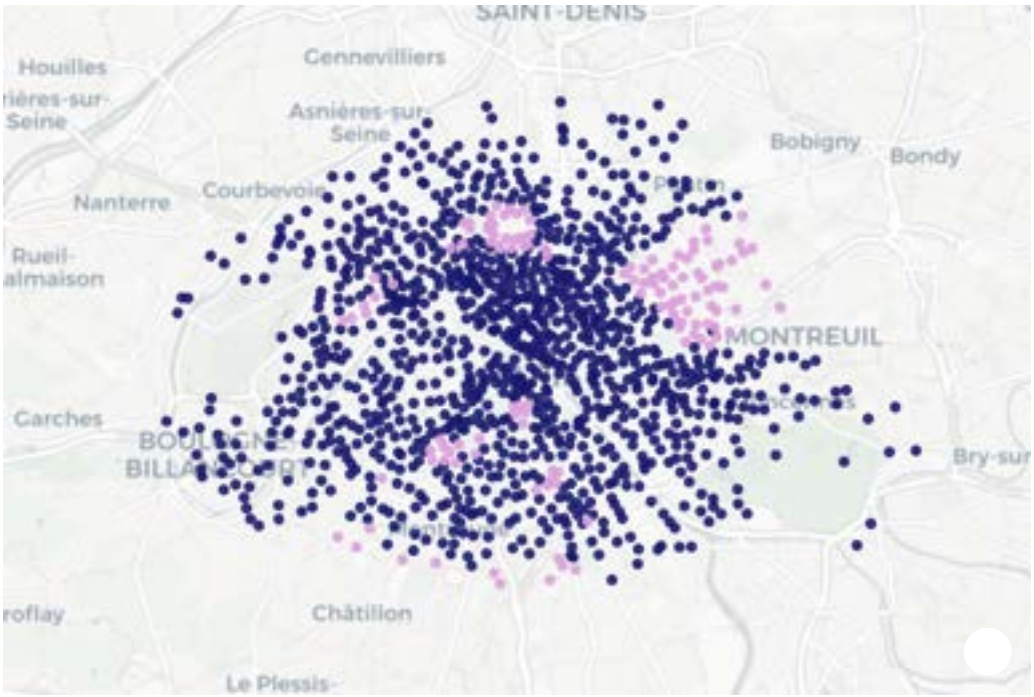
In [80]:
```
K = 4

# Perform Agglomerative Clustering
ac = AgglomerativeClustering(n_clusters=K, compute_distances=True, linkage='
clusters_ac = ac.fit_predict(loading_pca6)
# Convert cluster labels to strings for better visualization
cluster_names = ["Cluster " for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_ac]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
```

```
                        title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                        color_discrete_map={0: 'midnightblue', 1: 'plum'},
                        mapbox_style="carto-positron", zoom=10, opacity=0.9,
                        title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



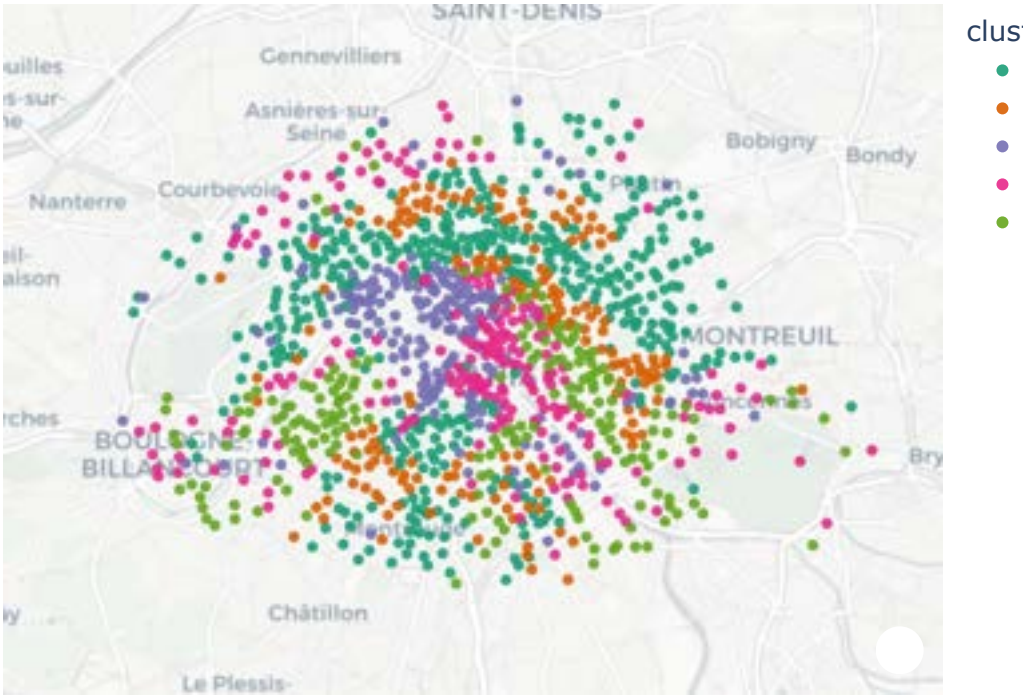
Once again we note that cluster 1 correspond to the stations placed in the city-center and that cluster2 is the emptiest containing the stations on hill.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

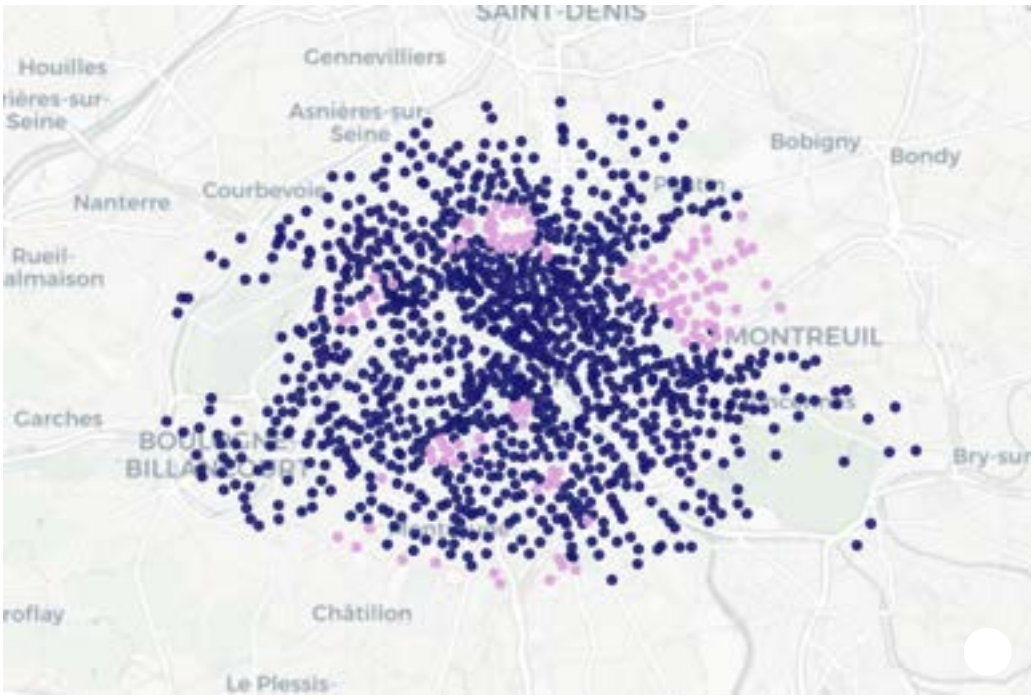
In [81]:
```
K = 5

# Perform Agglomerative Clustering
ac = AgglomerativeClustering(n_clusters=K, compute_distances=True, linkage='
clusters_ac = ac.fit_predict(loading_pca6)
# Convert cluster labels to strings for better visualization
cluster_names = ["Cluster " for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_ac]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
```

```
                        title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                        color_discrete_map={0: 'midnightblue', 1: 'plum'},
                        mapbox_style="carto-positron", zoom=10, opacity=0.9,
                        title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



It is hard to make interpretations in this case, maybe five classes is not the appropriate choice.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

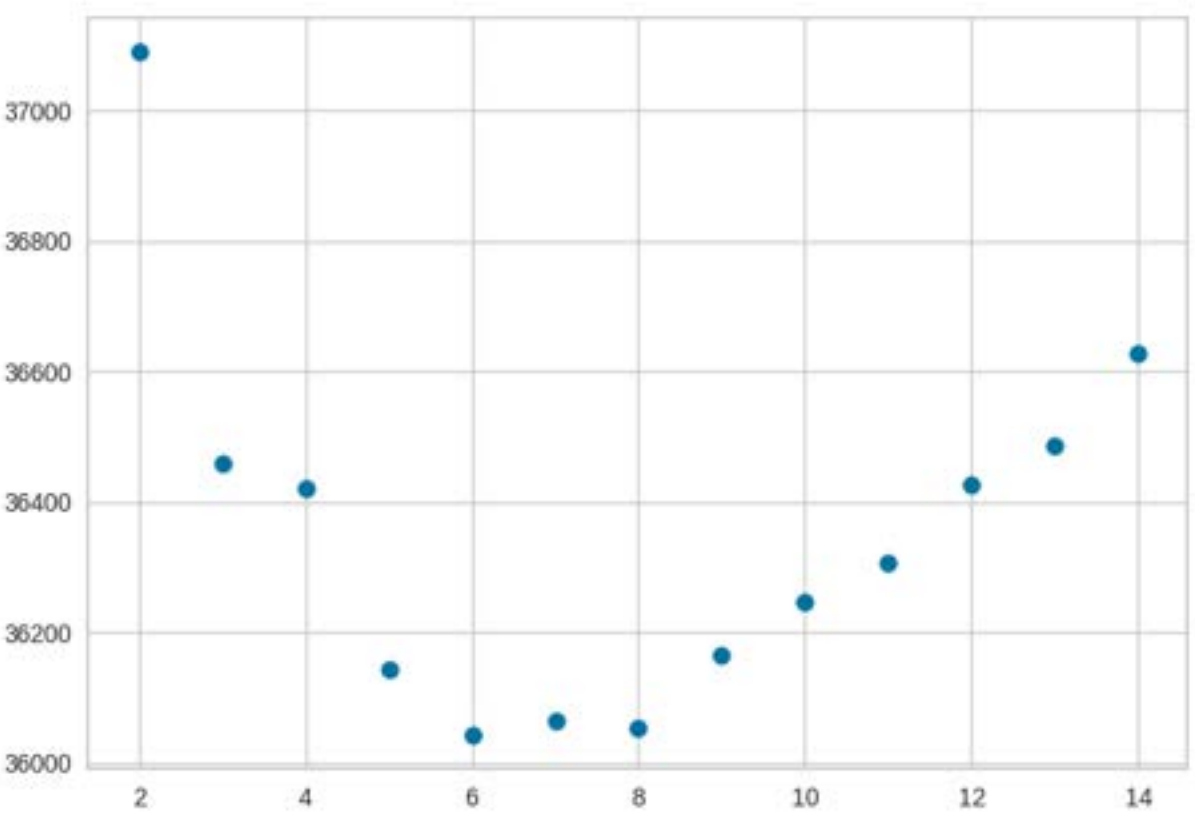
4.3. Gaussian Mixture clustering on original data

4.3.1. Selection of the number of clusters

4.3.1.1. Determining the number of clusters using BIC

In [82]:
```
k_max = 15

bic = []
for k in range(2, k_max):
    gmm = GaussianMixture(n_components=k, init_params='kmeans', n_init=3)
    gmm.fit(loading_pca6)
    bic.append(gmm.bic(loading_pca6))
bic = np.array(bic)
```

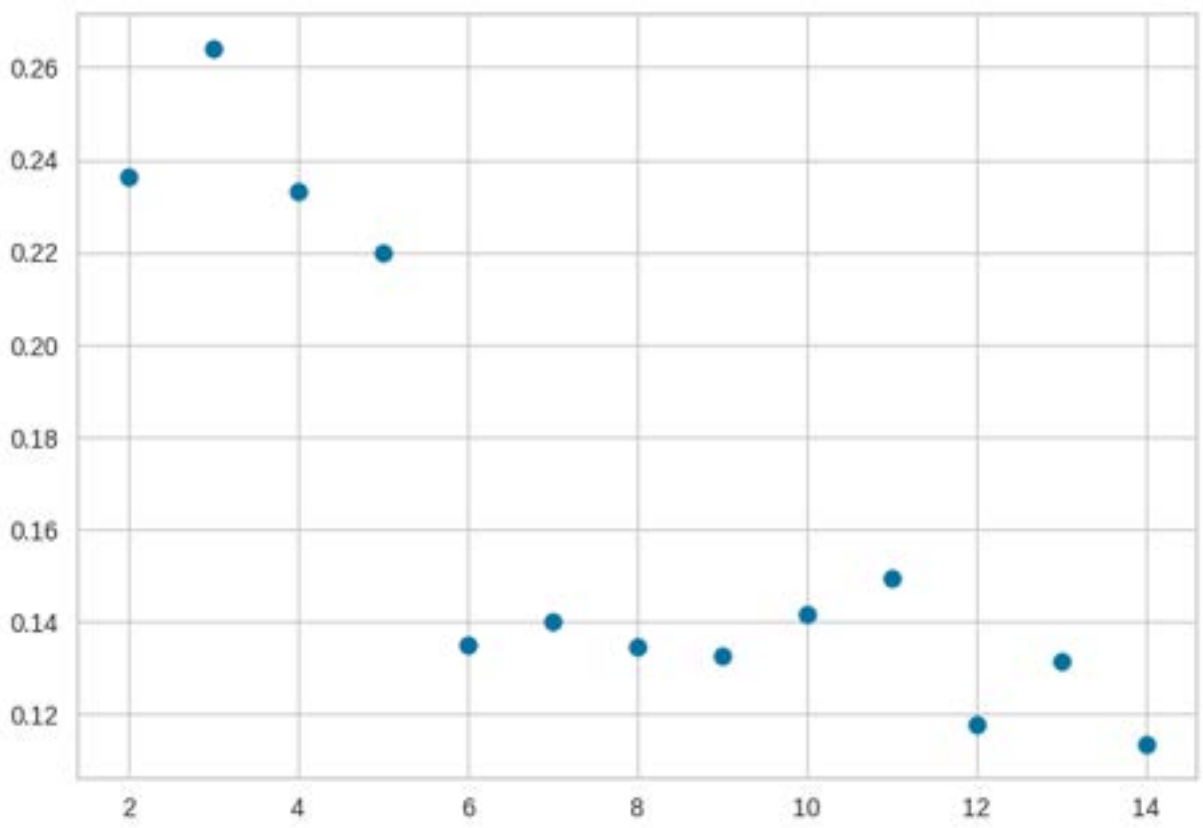
```
plt.scatter(range(2, k_max), bic)
plt.show()
```



The number of clusters corresponding to the lowest BIC value is the optimal number of clusters for your data, so the best k= 7.

4.3.1.2. Determining the number of clusters using silhouette score

In [83]:
```
k_max = 15

silhouette = []
for k in range(2, k_max):
    gmm = GaussianMixture(n_components=k, init_params='kmeans', n_init=3)
    clusters_gmm = gmm.fit_predict(loading_pca6)
    silhouette.append( silhouette_score(loading_pca6, clusters_gmm, metric='
silhouette = np.array(silhouette)

plt.scatter(range(2, k_max), silhouette)
plt.show()
```



We look for the highest silhouette score, indicating the best clustering solution. Optimal Number of Clusters: 3

According to BIC, the optimal K value is 7.

According to Silhoute, the optimal K value is 3.

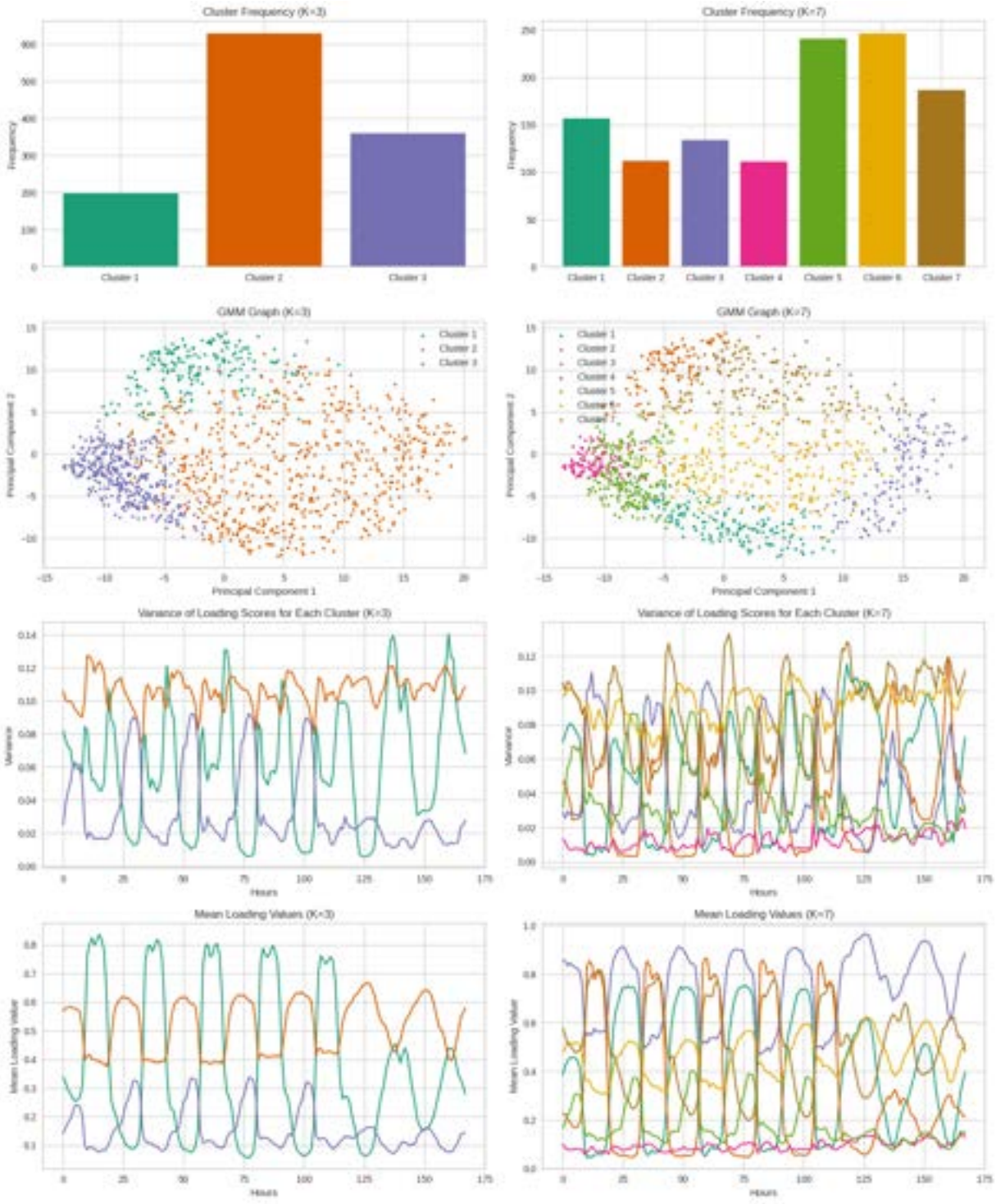
Earlier we saw that making interpretations with more than 4 clusters is hard so we will keep that in mind.

4.3.2. Visualization and interpretation of GMM

4.3.2.1. Four descriptive plots of cluster

In [84]:
```
K_values = [3, 7]
cmap = plt.get_cmap('Dark2')
fig, axs = plt.subplots(4, len(K_values), figsize=(15, 18))

for i, K in enumerate(K_values):

# Gaussian Mixture Model
    gmm = GaussianMixture(n_components=K, n_init=3)
    clusters_gmm = gmm.fit_predict(loading_pca6)
    cluster_freq = np.unique(clusters_gmm, return_counts=True)
    cluster_names = [f"Cluster {i+1}" for i in range(K)]

    # Plot Cluster Frequency
```

```
    axs[0, i].bar(cluster_names, cluster_freq[1], color=cmap.colors)
    axs[0, i].set_ylabel("Frequency")
    axs[0, i].set_title(f"Cluster Frequency (K={K})")

    # Scatter plot of Agglomerative Clustering results
    for j in range(K):
        axs[1, i].scatter(loading_pca[clusters_gmm == j, 0], loading_pca[clu
    axs[1, i].set_title(f"GMM Graph (K={K})")
    axs[1, i].legend()
    axs[1, i].set_xlabel("Principal Component 1")
    axs[1, i].set_ylabel("Principal Component 2")

    # Variance Plot
    x = np.arange(loading.shape[1])
    for j in range(K):
        axs[2, i].plot(x, np.var(loading[clusters_gmm == j], axis=0), color=

    axs[2, i].set_xlabel("Hours")
    axs[2, i].set_ylabel("Variance")
    axs[2, i].set_title(f"Variance of Loading Scores for Each Cluster (K={K}

    # Compute and plot mean loading values
    x = np.arange(loading.shape[1])  # Assuming loading is your data matrix
    for j in range(K):
        axs[3, i].plot(x, np.mean(loading[clusters_gmm == j], axis=0), color

    axs[3, i].set_xlabel("Hours")
    axs[3, i].set_ylabel("Mean Loading Value")
    axs[3, i].set_title(f"Mean Loading Values (K={K})")

plt.tight_layout()
plt.show()
```

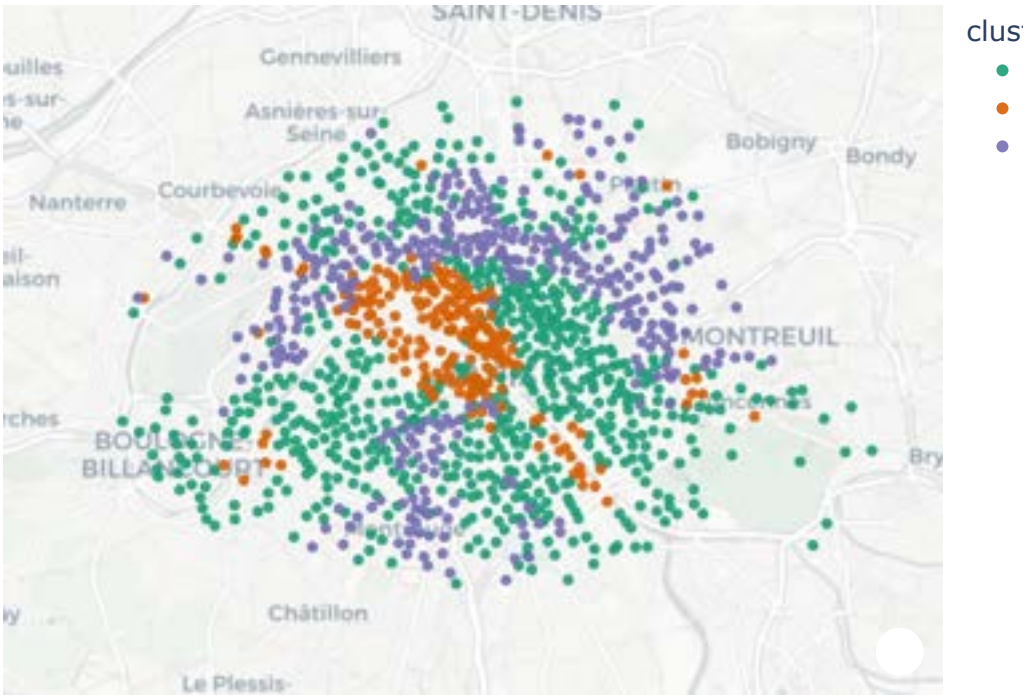


- k = 3 : Cluster2 is the densest cluster. On average it is neither full nor empty and has the same behaviour during the week and during the week-end. Clusters 1 and 2 are emptier during the week end.
- k = 7 : We can see that cluster4, which is by far the emptiest one,. Cluster1 and cluster2 are complementary. Cluster1 is full during the night and empty during the day. Vice versa for cluster2. Cluster6 is neither full nor empty. Cluster3 is always somewhat full.

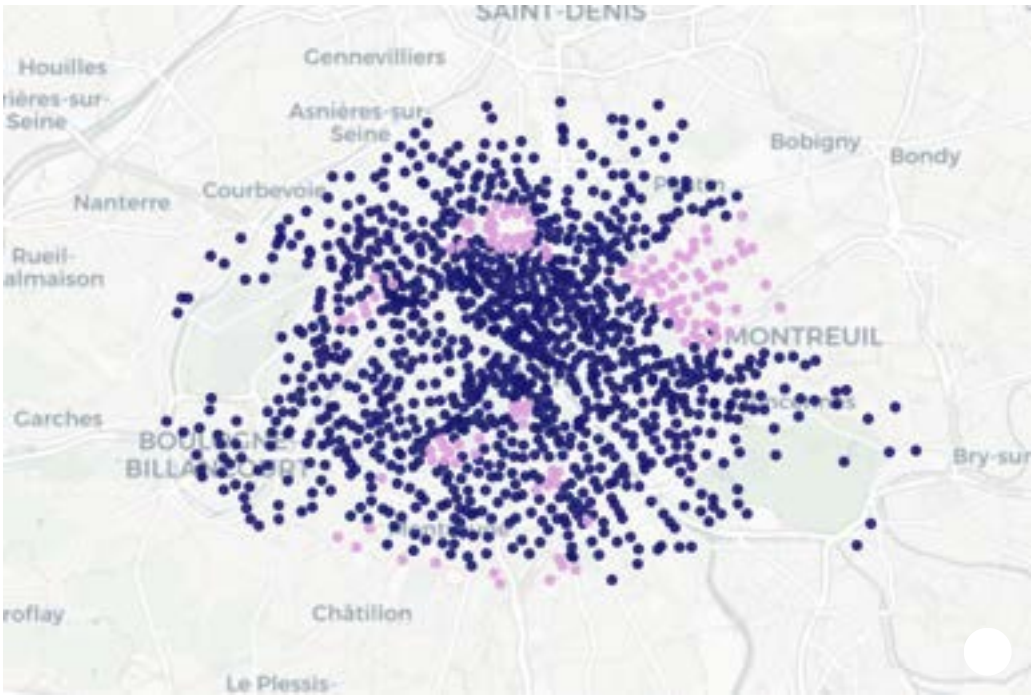
4.3.2.2. Visualization of clusters on the map

In [85]:
```
K = 3
# Gaussian Mixture Model
gmm = GaussianMixture(n_components=K, n_init=3)
clusters_gmm = gmm.fit_predict(loading_pca6)

# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_gmm]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')
# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

fig2.show()
```

Stations by Clusters



Hilltop stations



Cluster3 is the city-center and it is full during the day which doesn't contradict our previous analysis.

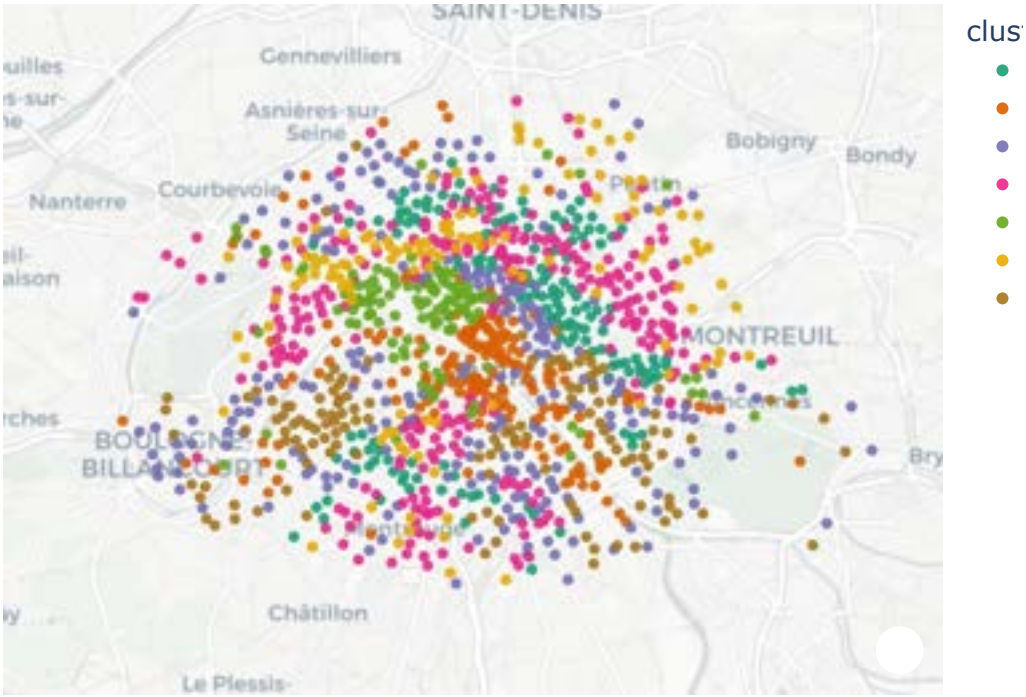
The hill stations are in cluster one which is the one where stations are on average emptier, which is also coherent.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

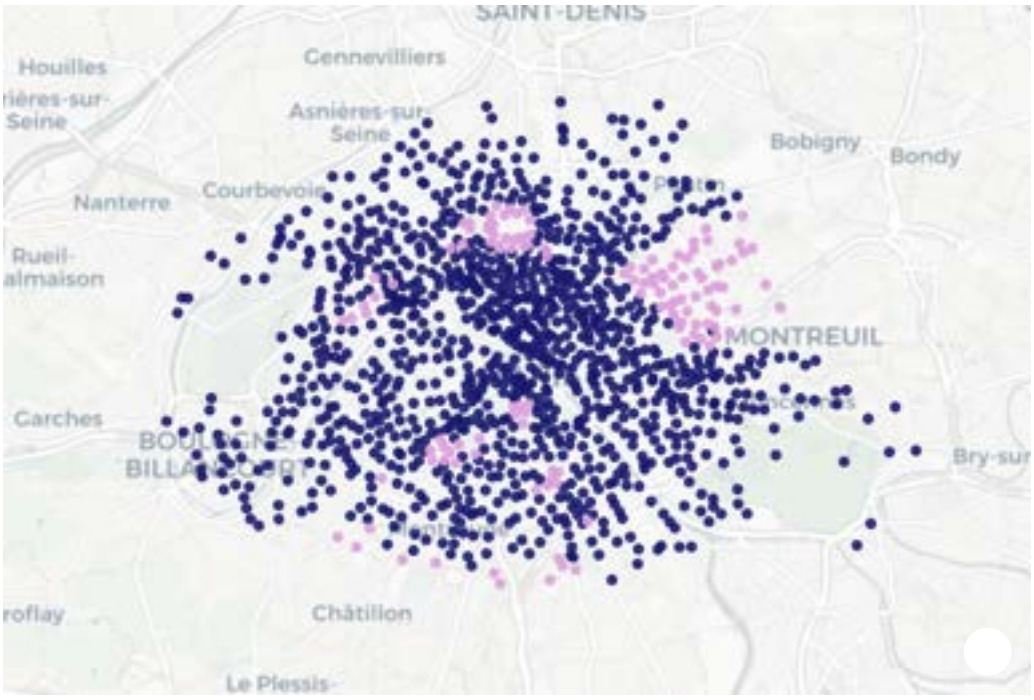
In [86]:
```
K = 7
# Gaussian Mixture Model
gmm = GaussianMixture(n_components=K, n_init=3)
clusters_gmm = gmm.fit_predict(loading_pca6)

# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_gmm]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical
```

```
# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

fig2.show()
```

Stations by Clusters



Hilltop stations



Cluster6 and cluster4 are located in the city-center. Cluster6 is always around half filled, which means there are always a number of velibs being used but still enough remain in the city-center. Cluster4 on the other hand is by far the emptiest one. It would mean that velibs are constantly used. These interpretations seem to be plausible even though they are not coherent with what we have said previously. However, this clustering with GMM and k=7 was very hard to interpret so we are more inclined to trust our previous results.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

4.4. Comparison of clustering methods

We will use the function `matchClasses` already created in the previous part. For easy lecture we will rewrite it.

the function `matchClasses` reorders the classes found in the two classifications so that they match.This function returns the rearranged confusion matrix and the associated class indices for each point of the dataset.

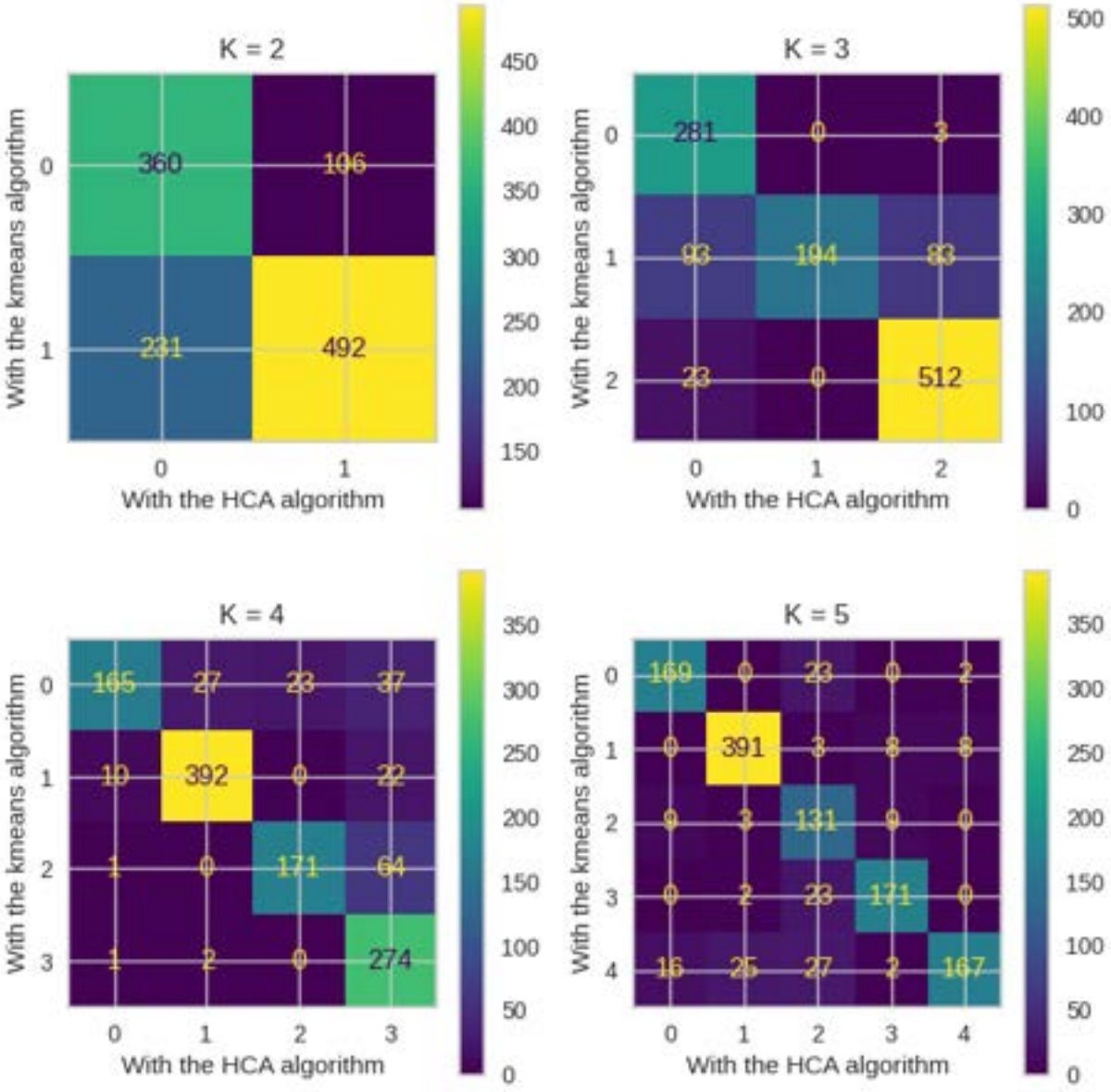
In [87]:
```
def matchClasses(classif1, classif2):
    cm = confusion_matrix(classif1, classif2)
    K = cm.shape[0]
    a, b = np.zeros(K), np.zeros(K)
    for j in range(K):
        for i in range(K):
            if (a[j] < cm[i,j]):
                a[j] = cm[i,j]
                b[j] = i
    a = a.astype(int)
    b = b.astype(int)

    print ("")
    print ("Classes size:", a)
    print ("Class (in the classif1 numbering):", b)
    print ("")

    table = cm.copy()
    for i in range(K):
        table[:,b[i]] = cm[:,i]

    clusters = classif2.copy()
    n = classif2.shape[0]
    for i in range(n):
        for j in range(K):
            if (classif2[i] == j):
                clusters[i] = b[j]

    return table, clusters
```

4.4.1. HCA vs. k-means

In [88]:
```
K_values = [2, 3, 4, 5]

fig, axs = plt.subplots(2, 2, figsize=(7, 7))

for i, K in enumerate(K_values):
    kmeans = KMeans(n_clusters=K, n_init=10)
    clusters_kmeans_loading = kmeans.fit_predict(loading_pca6)

    HCA = AgglomerativeClustering(n_clusters=K, compute_distances=True, link
    clusters_HCA = HCA.fit_predict(loading_pca6)

    cm, clusters_kmeans_loading_sorted = matchClasses(clusters_kmeans_loadin

    row = i // 2
    col = i % 2

    ConfusionMatrixDisplay(cm).plot(ax=axs[row, col])
    axs[row, col].set_xlabel('With the HCA algorithm')
    axs[row, col].set_ylabel('With the kmeans algorithm')
    axs[row, col].set_title(f'K = {K}')

plt.tight_layout()
```

```
plt.show()
```

```
Classes size: [360 492]
Class (in the classif1 numbering): [0 1]


Classes size: [512 281 194]
Class (in the classif1 numbering): [2 0 1]


Classes size: [274 392 171 165]
Class (in the classif1 numbering): [3 1 2 0]


Classes size: [131 391 169 167 171]
Class (in the classif1 numbering): [2 1 0 4 3]
```



With k=2 the clusters are significantly different. Thinking back, that is a result we could have expected. Indeed, with k-means clustering and k=2 the two clusters were

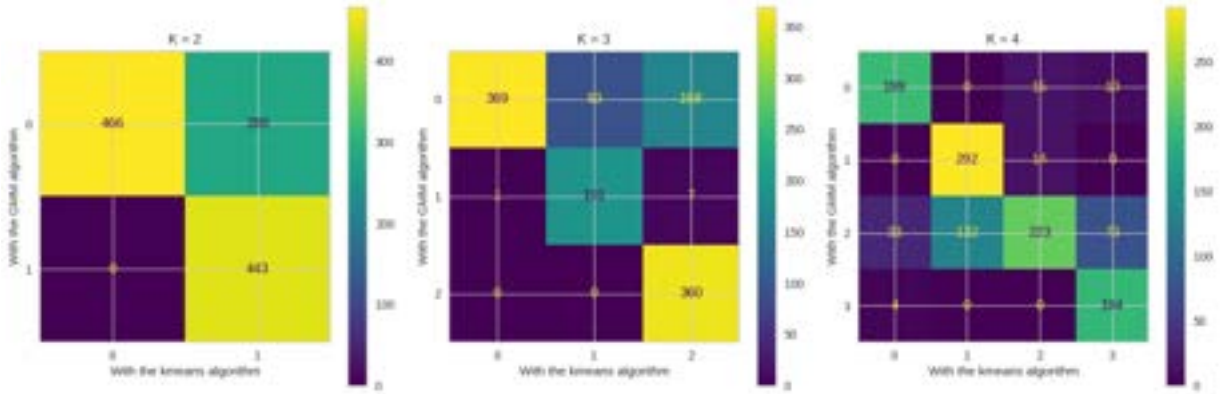
seperated by the axis component1=0. That was not the case with HCA and k=2.

With the other values, both methods create similar clusters, which is satisfactory. No need to perform a MCA between clusters in this case.

4.4.2. GMM vs. k-means

In [89]:
```
K_values = [2, 3, 4]

fig, axs = plt.subplots(1, 3, figsize=(15, 5))

for i, K in enumerate(K_values):
    kmeans = KMeans(n_clusters=K, n_init=10)
    clusters_kmeans_loading = kmeans.fit_predict(loading_pca6)

    gmm = GaussianMixture(n_components=K, n_init=10, init_params='kmeans')
    clusters_gmm = gmm.fit_predict(loading_pca6)

    cm, clusters_kmeans_loading_sorted = matchClasses(clusters_gmm, clusters

    ConfusionMatrixDisplay(cm).plot(ax=axs[i])
    axs[i].set_xlabel('With the kmeans algorithm')
    axs[i].set_ylabel('With the GMM algorithm')
    axs[i].set_title(f'K = {K}')

plt.tight_layout()
plt.show()
```

```
Classes size: [443 466]
Class (in the classif1 numbering): [1 0]


Classes size: [360 369 191]
Class (in the classif1 numbering): [2 0 1]


Classes size: [223 292 194 199]
Class (in the classif1 numbering): [2 1 3 0]
```

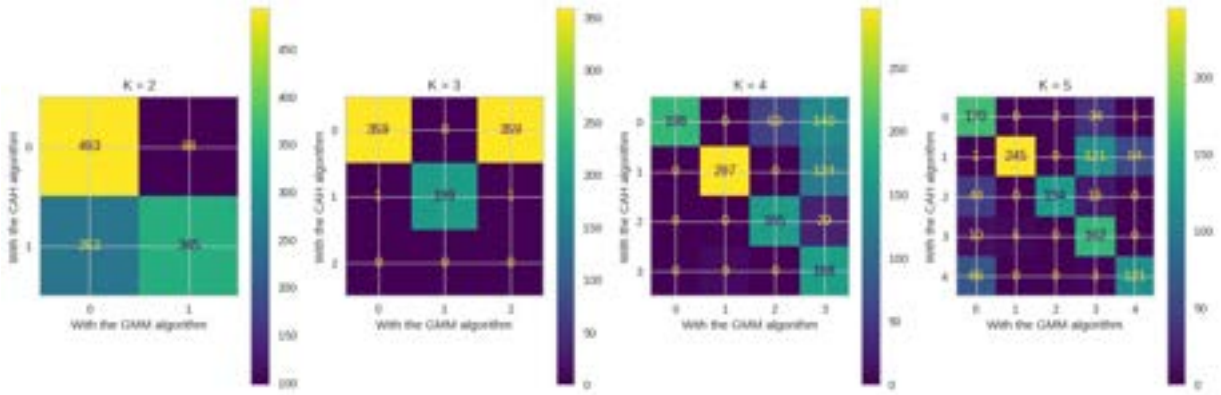


For all values of k the confusion matrices are not satisfactory. We should perform a MCA between clusters in this case.

4.4.3. HCA versus GMM

In [90]:
```
K_values = [2, 3, 4, 5]

fig, axs = plt.subplots(1, 4, figsize=(15, 5))

for i, K in enumerate(K_values):
    HCA = AgglomerativeClustering(n_clusters=K, compute_distances=True, link
    clusters_HCA = HCA.fit_predict(loading_pca6)

    gmm = GaussianMixture(n_components=K, n_init=10, init_params='kmeans')
    clusters_gmm = gmm.fit_predict(loading_pca6)

    cm, clusters_kmeans_loading_sorted = matchClasses(clusters_HCA, clusters

    ConfusionMatrixDisplay(cm).plot(ax=axs[i])
    axs[i].set_xlabel('With the GMM algorithm')
    axs[i].set_ylabel('With the CAH algorithm')
    axs[i].set_title(f'K = {K}')

plt.tight_layout()
plt.show()
```

```
Classes size: [493 345]
Class (in the classif1 numbering): [0 1]


Classes size: [239 199 359]
Class (in the classif1 numbering): [0 1 0]


Classes size: [297 165 168 198]
Class (in the classif1 numbering): [1 2 3 0]


Classes size: [245 162 134 170 121]
Class (in the classif1 numbering): [1 3 2 0 4]
```



Once again, the cconfusion matrices are unsatisfactory and we should perform a MCA between clusters in this case.

Part 5: Multiple Correspondence Analysis (MCA)

Multiple Correspondence Analysis (MCA) is a dimensionality reduction technique used primarily for categorical data analysis. It extends the principles of Principal Component Analysis (PCA) to handle categorical variables, enabling the exploration and visualization of complex relationships within multivariate categorical datasets.

5.1. Creation of new cathegorical dataframe

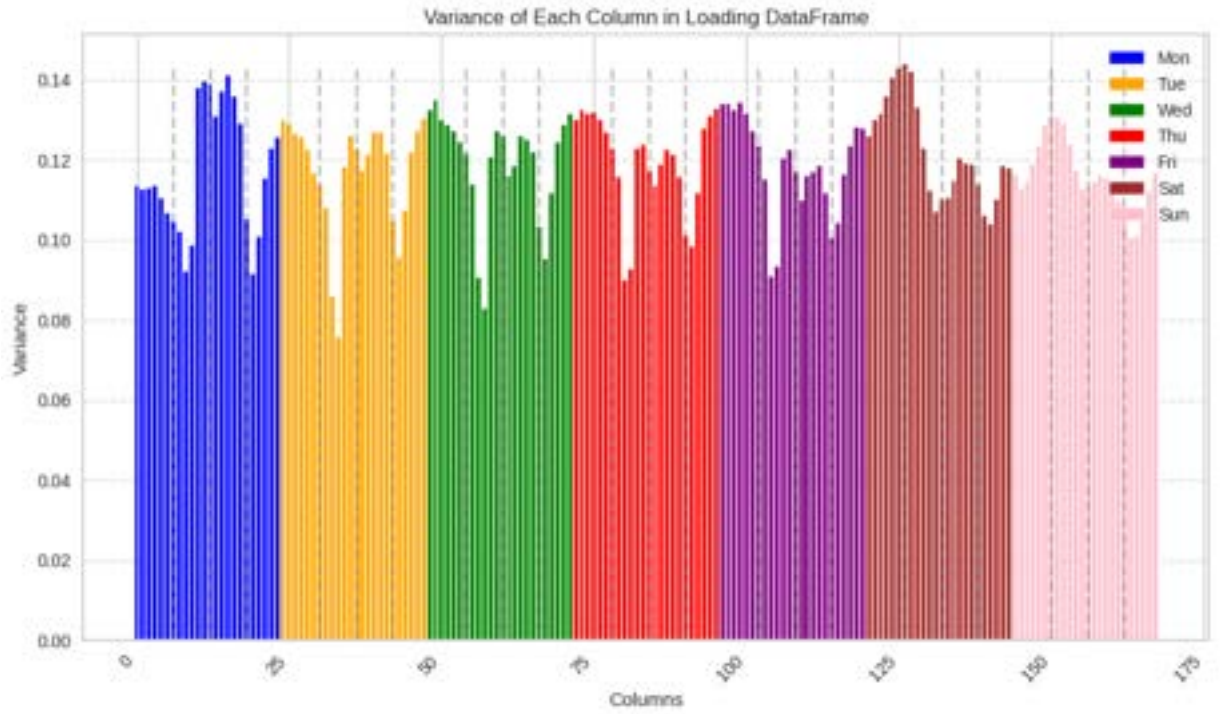
5.1.1. Variance of each column in the loading DataFrame

In data analysis, understanding the temporal patterns within the dataset can provide valuable insights. One common approach is to aggregate data based on time intervals, such as days of the week. Before performing such aggregation, it's crucial to assess the variability of the data to ensure that meaningful insights are derived from the analysis.

The next code calculates the variance of each column in the loading DataFrame. Analyzing the variance helps in understanding the dispersion of data values within each column. This step is essential as it allows us to identify columns with significant variability, which can influence subsequent aggregation and analysis.

In [91]:
```
# Define the days of the week and corresponding colors
days_of_week = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
color_palette = ['blue', 'orange', 'green', 'red', 'purple', 'brown', 'pink'

# Compute the variance of loading
loading_variance = loading.var()

# Plot the variances
plt.figure(figsize=(10, 6))

# Iterate over each day of the week
for i in range(7):
    # Compute the start and end indices for the current day
    start_index = i * 24
    end_index = (i + 1) * 24

    # Plot the variances for the current day with a different color
    plt.bar(np.arange(start_index, end_index), loading_variance[start_index:

    # Add vertical lines every 6 x-axis points
    for j in range(start_index + 6, end_index, 6):
        plt.vlines(x=j, ymin=0, ymax=loading_variance.max(), colors='gray',
```

```
plt.title('Variance of Each Column in Loading DataFrame')
plt.xlabel('Columns')
plt.ylabel('Variance')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend()
plt.tight_layout()
plt.show()
```



5.1.2. Creation of new cathegorical DataFrame

Having just verified that the variance of each day's data is sufficiently low to warrant grouping by 6 hours per day, the next step is to compute the mean for each quart of the day of the week. This involves aggregating the data by dividing it into 24-hour intervals representing different times of the day, and then calculating the mean for each interval.

In [92]:
```
# Calculate the mean for each day of the week
loading_per_day = loading.groupby(np.arange(len(loading.columns)) // 6, axis

# Rename columns to represent days of the week
loading_per_day.columns = ['Mon00-06','Mon06-12','Mon12-18','Mon18-24','Tue0

# Display the shape of the resulting DataFrame
print("Shape of the DataFrame after averaging for each day of the week:", lo

# Display the shape of the resulting DataFrame
loading_per_day.head()
```

```
Shape of the DataFrame after averaging for each day of the week: (1189, 28)
```

Out[92]:

	Mon00-06	Mon06-12	Mon12-18	Mon18-24	Tue00-06	Tue06-12	Tue12-18	Tue
1	0.044872	0.030678	0.023810	0.345238	0.535714	0.298280	0.055556	0.2
2	0.456522	0.231884	0.050725	0.630435	1.000000	0.507246	0.050725	0.7
3	0.136364	0.160606	0.392730	0.405797	0.326087	0.477602	0.893939	0.8
4	0.952381	0.984127	0.992063	0.111111	0.047619	0.555556	0.761905	0.1
5	0.775362	0.862319	0.886548	0.422208	0.372339	0.535447	0.816248	0.7

5 rows × 28 columns

In the following code snippet, we define bins and labels to convert quantitative values into qualitative classes. These classes are categorized based on predefined intervals, with each interval corresponding to a specific label:

A: 0.00%- 0.25%

B: 0.25%- 0.50%

C: 0.50%- 0.75%

D: 0.75%- 1.00%

In [93]:
```
# Define the bins and labels for the qualitative classes
bins = [0, 0.25, 0.5, 0.75, 1]
labels = ['A', 'B', 'C', 'D']

# Apply pd.cut to convert quantitative values to qualitative classes
loading_qualitative = loading_per_day.apply(lambda x: pd.cut(x, bins=bins, l

# Display the shape of the resulting DataFrame
print("Shape of the qualitative_classes DataFrame :", loading_qualitative.sh

# Display the qualitative classes DataFrame
print("Qualitative classes DataFrame:")
loading_qualitative.head()

loading_qualitative.astype('category')
```

```
Shape of the qualitative_classes DataFrame : (1189, 28)
Qualitative classes DataFrame:
```

Out[93]:

	Mon00-06	Mon06-12	Mon12-18	Mon18-24	Tue00-06	Tue06-12	Tue12-18	T
1	A	A	A	B	C	B	A	
2	B	A	A	C	D	C	A	
3	A	A	B	B	B	B	D	
4	D	D	D	A	A	C	D	
5	D	D	D	B	B	C	D	
...	...	...	...	...	...	...	...	
1185	A	A	NaN	A	A	A	A	
1186	A	B	B	A	A	B	C	
1187	C	B	A	A	B	A	A	
1188	B	B	D	D	D	D	D	
1189	D	D	D	B	A	C	D	

1189 rows × 28 columns

The next code adds the "bonus" column from coord to qualitative_classes.

In [94]:
```
# Add the "bonus" column from coord to qualitative_classes
loading_qualitative['bonus'] = coord['bonus']

#loading_qualitative = loading_qualitative.set_index('bonus')


# Display the shape of the resulting DataFrame
print("Shape of the qualitative_classes DataFrame :", loading_qualitative.sh
# Display the updated DataFrame
print("Updated DataFrame:")
loading_qualitative.head()
```

```
Shape of the qualitative_classes DataFrame : (1189, 29)
Updated DataFrame:
```

Out[94]:

	Mon00-06	Mon06-12	Mon12-18	Mon18-24	Tue00-06	Tue06-12	Tue12-18	Tue
1	A	A	A	B	C	B	A	
2	B	A	A	C	D	C	A	
3	A	A	B	B	B	B	D	
4	D	D	D	A	A	C	D	
5	D	D	D	B	B	C	D	

5 rows × 29 columns

5.2. Implementing MCA

Using the `prince` package, we will perform a [MCA]

In [95]:
```
#pip install prince
```

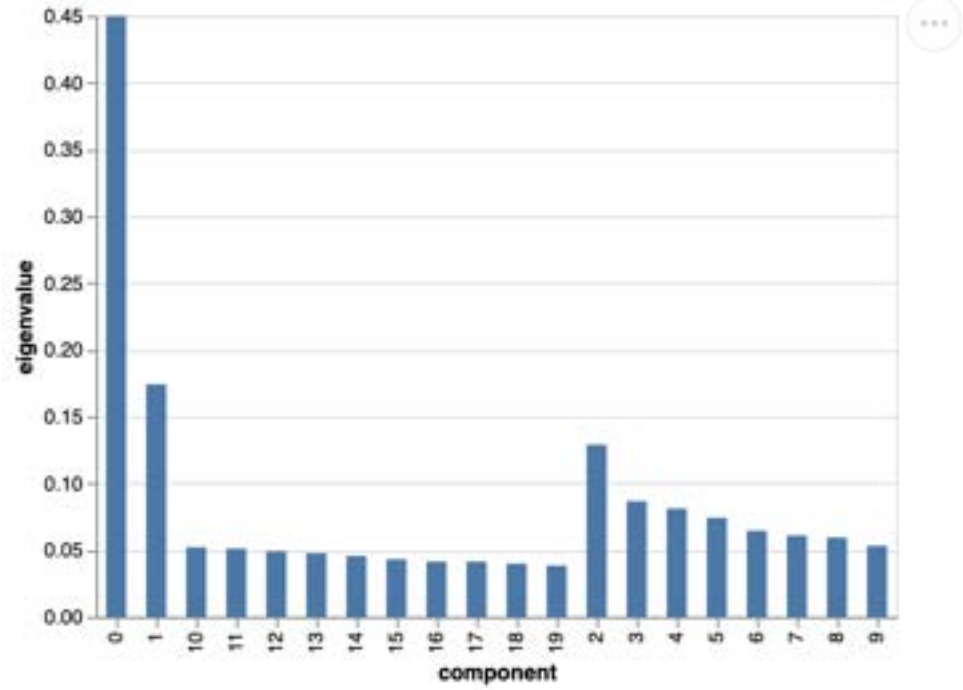
In [96]:
```
import prince
```

In [97]:
```
mca = prince.MCA(
    n_components=20,
    n_iter=10,
    copy=True,
    check_input=True,
    engine='sklearn',
    random_state=42
)
loading_mca = mca.fit_transform(loading_qualitative)
```

How many components on the MCA should we keep?

In [98]:
```
mca.scree_plot()
```

Out[98]:



In [99]:
```
display(mca.eigenvalues_summary)
```

component	eigenvalue	% of variance	% of variance (cumulative)
0	0.450	15.13%	15.13%
1	0.174	5.86%	20.99%
2	0.129	4.34%	25.33%
3	0.087	2.91%	28.25%
4	0.081	2.73%	30.97%
5	0.074	2.50%	33.47%
6	0.065	2.17%	35.64%
7	0.061	2.05%	37.69%
8	0.059	2.00%	39.69%
9	0.053	1.79%	41.48%
10	0.052	1.75%	43.23%
11	0.051	1.72%	44.95%
12	0.049	1.66%	46.61%
13	0.048	1.60%	48.21%
14	0.046	1.54%	49.75%
15	0.043	1.45%	51.20%
16	0.041	1.39%	52.59%
17	0.041	1.39%	53.98%
18	0.040	1.34%	55.33%
19	0.038	1.29%	56.62%

We will restrict ourselves to the first 16 components of the mca, corresponding to 51.20% of cummulative variance.

In [100…
```
loading_mca16 = loading_mca.iloc[:, :16]
print('Shape of loading_mca',loading_mca.shape)
print('Shape of loading_mca16',loading_mca16.shape)
```

```
Shape of loading_mca (1189, 20)
Shape of loading_mca16 (1189, 16)
```

In [101…
```
loading_mca16.head()
```

Out[101…

	0	1	2	3	4	5	6	
1	-0.296954	-0.584896	0.440117	-0.350795	0.101756	0.439306	-0.178957	0.08
2	0.430117	-0.758254	-0.107881	-0.243988	-0.655459	-0.144735	0.028227	0.26
3	0.224501	-0.037527	0.626605	0.676530	0.227393	0.252020	-0.316827	0.56
4	0.126351	0.696360	0.109043	0.301776	0.182454	-0.221914	0.132187	-0.39
5	0.370323	0.237575	0.727097	0.142240	0.058033	0.245786	0.167132	0.06

5.3. Variable factor maps

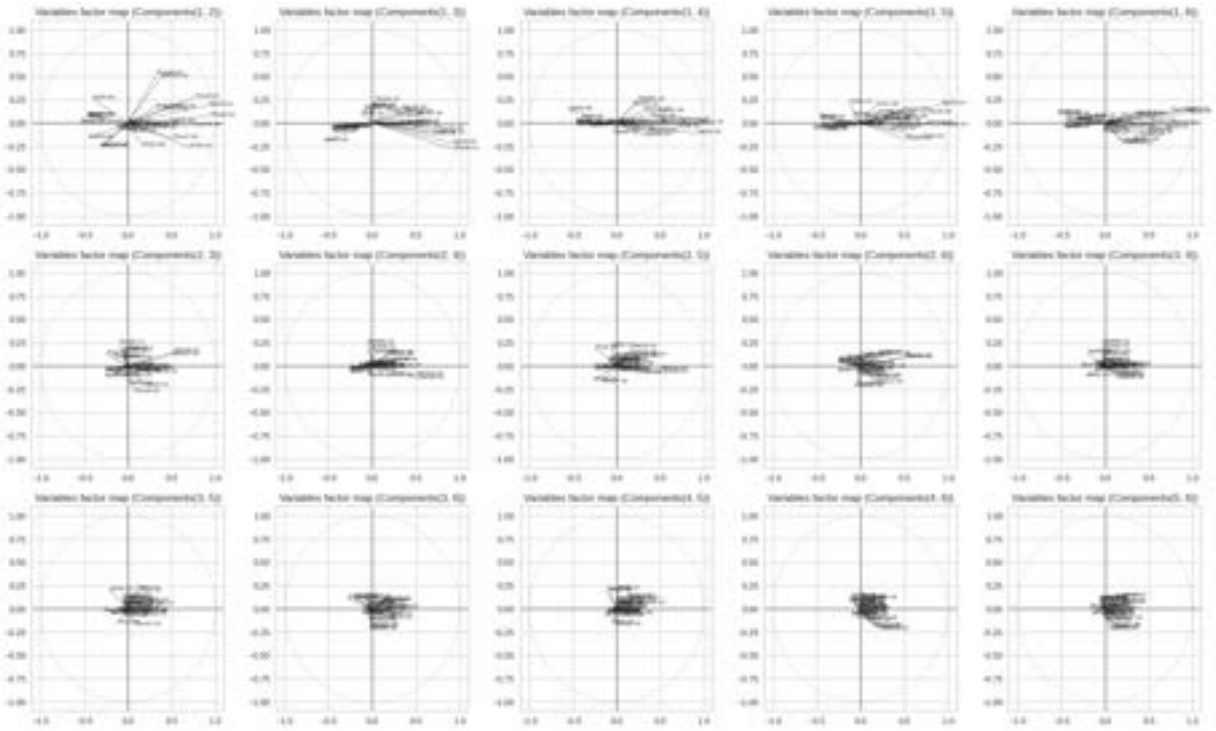
5.3.1. Variable Factor Maps for the first 6 principal components

The next plot visualizes the Variable Factor Map for each combination of the first 6 principal components obtained from MCA. This visualization depicts the factor map showcasing the relationship between variables in a dataset. Each point represents a variable, and its position is determined by its contributions to the principal components. The arrows indicate the direction and magnitude of these contributions.

In [102…
```
#We stock the coodinates for each variable per component
c_coordinates=mca.column_coordinates(loading_qualitative)
```

In [103…
```
# Calculate the coordinates of the variables on the factor map for the first
coords = []
components = []
for a in range(6):
    for b in range(a + 1, 6):
        coord1 = c_coordinates.iloc[:, a] * np.sqrt(mca.eigenvalues_[a])
        coord2 = c_coordinates.iloc[:, b] * np.sqrt(mca.eigenvalues_[b])
        coords.append((coord1, coord2))
        components.append((a+1, b+1))

# Adjusted variable labels to match the dataset
variable_labels = loading_qualitative.columns

# Plot Variable factor map for each combination of the first 6 principal com
fig, axs = plt.subplots(3, 5, figsize=(20, 12))
for k, (coord1, coord2) in enumerate(coords):
    row = k // 5
    col = k % 5
    ax = axs[row, col]
    for i, j, nom in zip(coord1, coord2, variable_labels):
        ax.text(i, j, nom, fontsize=7)
        ax.arrow(0, 0, i, j, color='black', alpha=0.7, width=0.0001)

    ax.axis((-1.1, 1.1, -1.1, 1.1))
    ax.hlines(y=0, xmin=-1.1, xmax=1.1, colors="black", linestyle="--", line
    ax.vlines(x=0, ymin=-1.1, ymax=1.1, colors="black", linestyle="--", line
    ax.add_artist(plt.Circle((0, 0), radius=1, color='cornflowerblue', fill=
```

```
    ax.set_title(f'Variables factor map (Components{components[k]})')
    ax.grid(True)
plt.tight_layout()
plt.show()
```



We note that many stations are well represented by component 1.

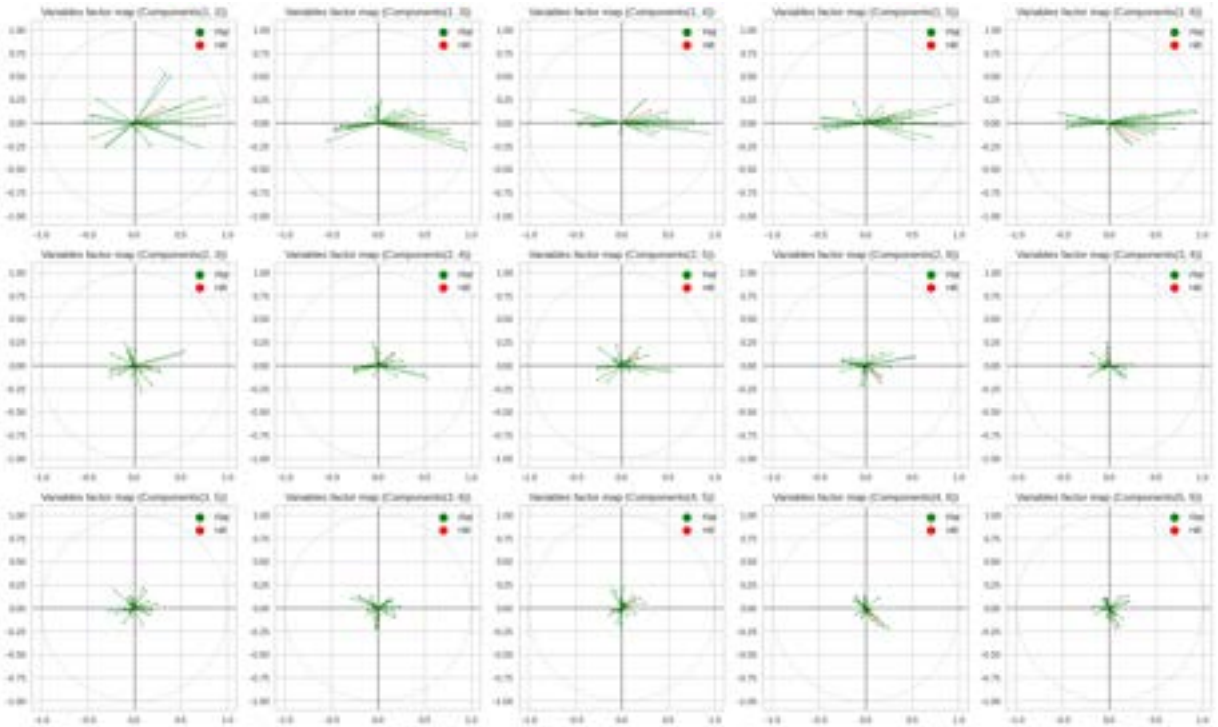
5.3.2. Variable Factor Map labeled by hill position

In this specific plot, we examine the Variable Factor Map for each combination of the first 6 principal components, with the color of each point denoting a binary feature: red for stations located on hills and green for stations situated on flat ground. This allows us to discern any discernible patterns or clusters based on terrain elevation.

In [104…

```
# Calculate the coordinates of the variables on the factor map for the first
coords = []
components = []
for a in range(6):
    for b in range(a + 1, 6):
        coord1 = c_coordinates.iloc[:, a] * np.sqrt(mca.eigenvalues_[a])
        coord2 = c_coordinates.iloc[:, b] * np.sqrt(mca.eigenvalues_[b])
        coords.append((coord1, coord2))
        components.append((a+1, b+1))

# Plot Variable factor map for each combination of the first 6 principal com
fig, axs = plt.subplots(3, 5, figsize=(20, 12))
for k, (coord1, coord2) in enumerate(coords):
    row = k // 5
    col = k % 5
    ax = axs[row, col]
```

```
    for i, j, nom, bonus in zip(coord1, coord2, loading_qualitative.columns,
        color = "red" if bonus else "green"  # Red for "Hill", green for "Fl
        ax.arrow(0, 0, i, j, color=color)  # Specify the arrow color
        ax.plot(i, j, "o", color=color, markersize=2)

    ax.axis((-1.1, 1.1, -1.1, 1.1))
    ax.hlines(y=0, xmin=-1.1, xmax=1.1, colors="black", linestyle="--", line
    ax.vlines(x=0, ymin=-1.1, ymax=1.1, colors="black", linestyle="--", line
    ax.add_artist(plt.Circle((0, 0), radius=1, color='cornflowerblue', fill=

    ax.set_title(f'Variables factor map (Components{components[k]})')
    ax.grid(True)

    # Create custom legend with colors
    labels = ['Flat', 'Hill']
    legend_elements = [Line2D([0], [0], marker='o', color='w', markerfacecol
                       Line2D([0], [0], marker='o', color='w', markerfacecol

    ax.legend(handles=legend_elements, labels=labels)

plt.tight_layout()
plt.show()
```



We cannot deduce anything labeling by hill position. Let's study the variable factor labeled by day and night hours.

5.3.3. Variable Factor Map labeled by day and night hours

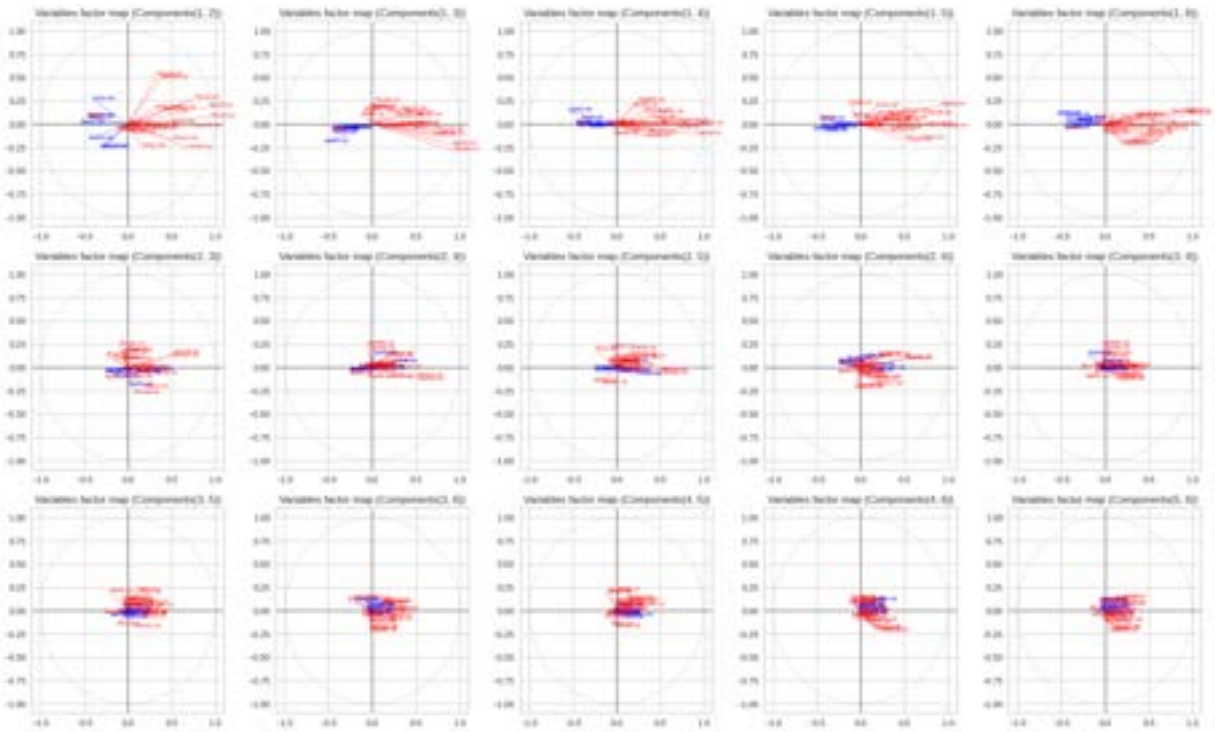
This plot illustrates the Variable factor map derived from MCA, showcasing the distribution of variables (stations) along the two primary components. What distinguishes this plot is the color differentiation for lines corresponding to day and night

hours. We will considere from 00 am until 6 am the night hours.

Red lines indicate stations active during the day, while blue lines represent those active during the night. This visualization offers insights into the temporal patterns of station activity across the dataset.

In [105…

```
# Identify night hours from midnight until 6 am
night_hours = ['Mon00-06', 'Tue00-06', 'Wed00-06', 'Thu00-06', 'Fri00-06', '

# Create a boolean array indicating whether each hour is a night hour
is_night = loading_qualitative.columns.isin(night_hours)

# Calculate the coordinates of the variables on the factor map for the first
coords = []
components = []
for a in range(6):
    for b in range(a + 1, 6):
        coord1 = c_coordinates.iloc[:, a] * np.sqrt(mca.eigenvalues_[a])
        coord2 = c_coordinates.iloc[:, b] * np.sqrt(mca.eigenvalues_[b])
        coords.append((coord1, coord2))
        components.append((a+1, b+1))

# Plot Variable factor map for each combination of the first 6 principal com
fig, axs = plt.subplots(3, 5, figsize=(20, 12))
for k, (coord1, coord2) in enumerate(coords):
    row = k // 5
    col = k % 5
    ax = axs[row, col]
    for i, j, nom, is_night_hour in zip(coord1, coord2, loading_qualitative.
        color = 'blue' if is_night_hour else 'red'
        ax.arrow(0, 0, i, j, color=color, alpha=0.7, width=0.0001)
        ax.text(i, j, nom, fontsize=7, color=color)

    ax.axis((-1.1, 1.1, -1.1, 1.1))
    ax.hlines(y=0, xmin=-1.1, xmax=1.1, colors="black", linestyle="--", line
    ax.vlines(x=0, ymin=-1.1, ymax=1.1, colors="black", linestyle="--", line
    ax.add_artist(plt.Circle((0, 0), radius=1, color='cornflowerblue', fill=

    ax.set_title(f'Variables factor map (Components{components[k]})')
    ax.grid(True)

plt.tight_layout()
plt.show()
```



We observe that component 1 effectively distinguishes the stations based on the time of day, separating them into distinct groups corresponding to day (component 1 > 0) and night (component 1 < 0) hours.

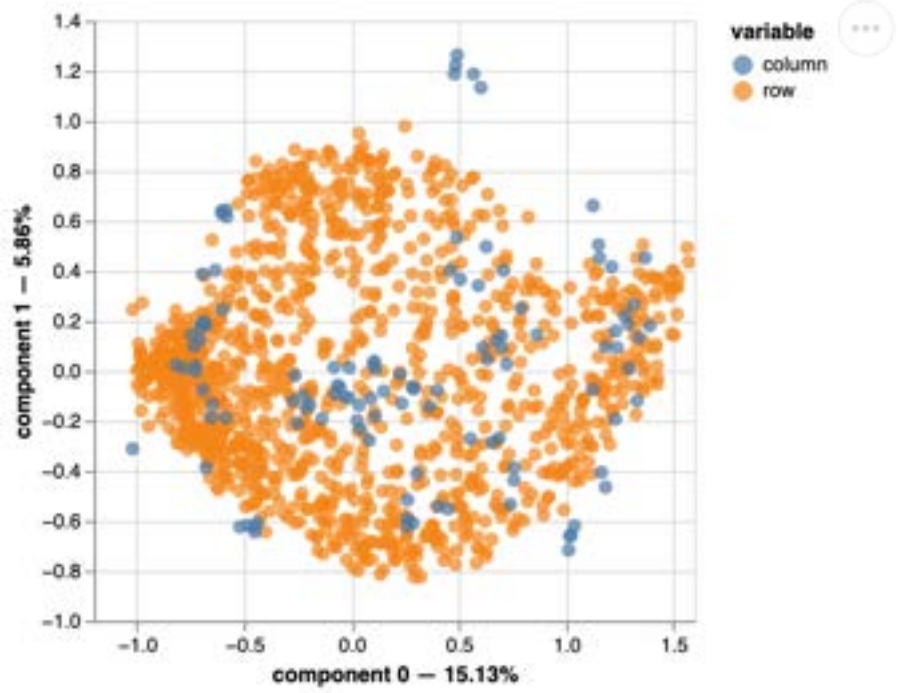
5.4. Individual factor maps

5.4.1. Individual factor maps for the first compenent with the six first components

The next plot visualizes the Individual factor maps for each combination of the first compenent with the six first components obtained from MCA. This visualization depicts the factor map showcasing the relationship between variables in a dataset. Each point represents a variable, and its position is determined by its contributions to the components.

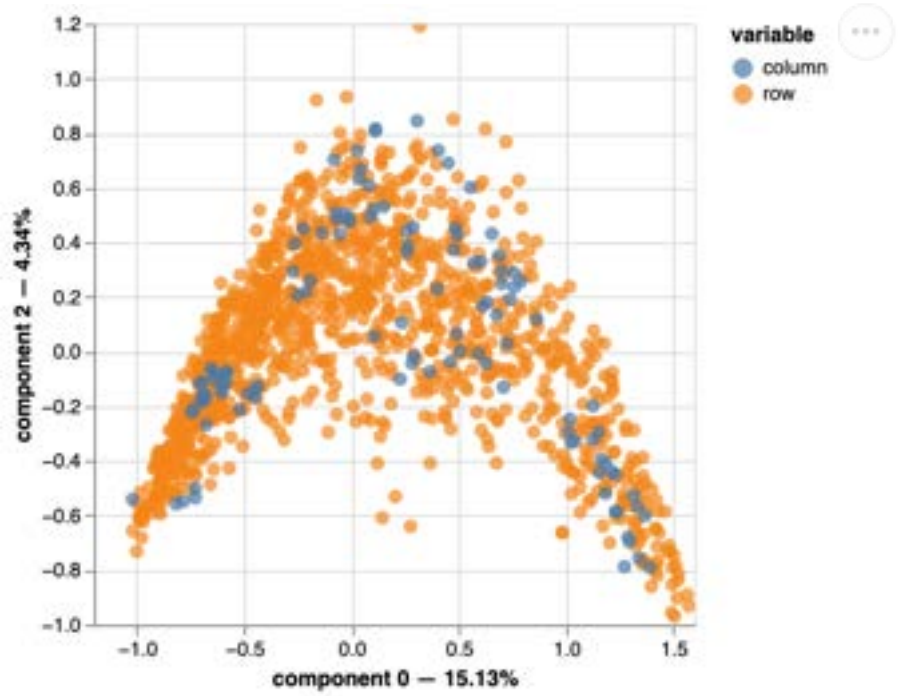
In [106…
```
mca.plot(loading_qualitative,x_component=0,y_component=1,show_column_markers
```

Out[106…



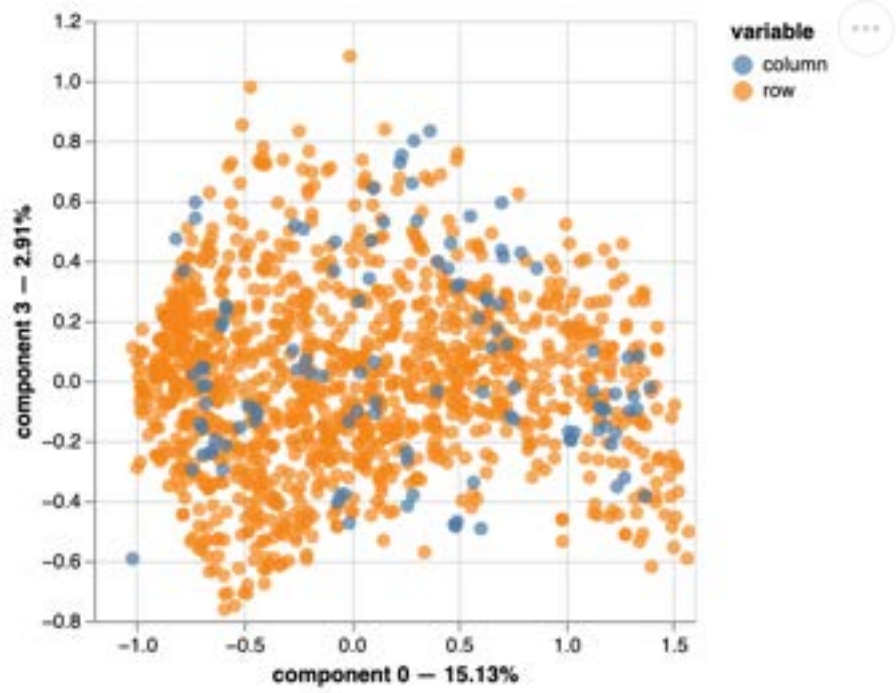
In [107…
```
mca.plot(loading_qualitative,x_component=0,y_component=2,show_column_markers
```

Out[107…



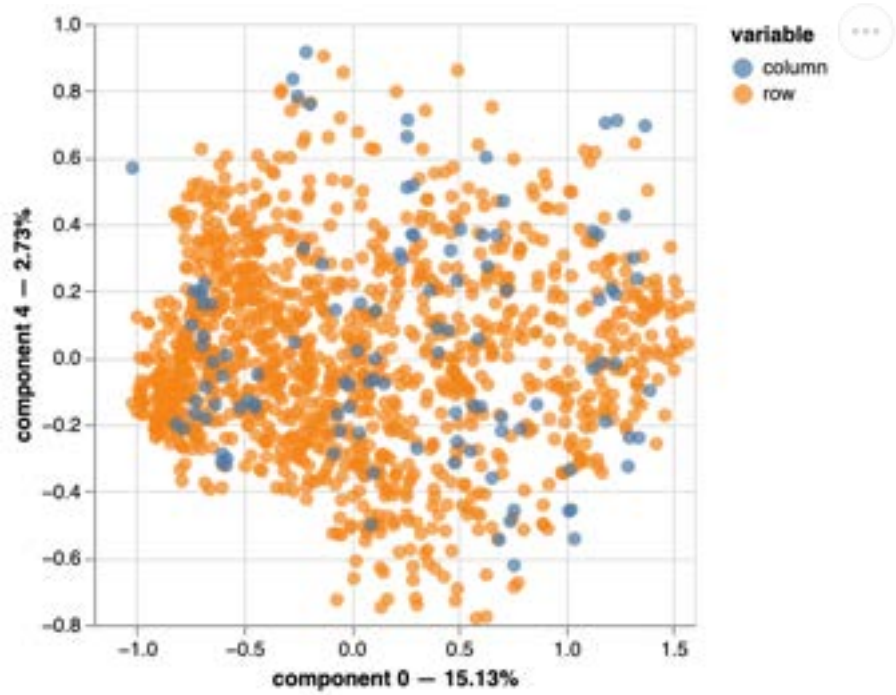
In [108…
```
mca.plot(loading_qualitative,x_component=0,y_component=3,show_column_markers
```

Out[108…



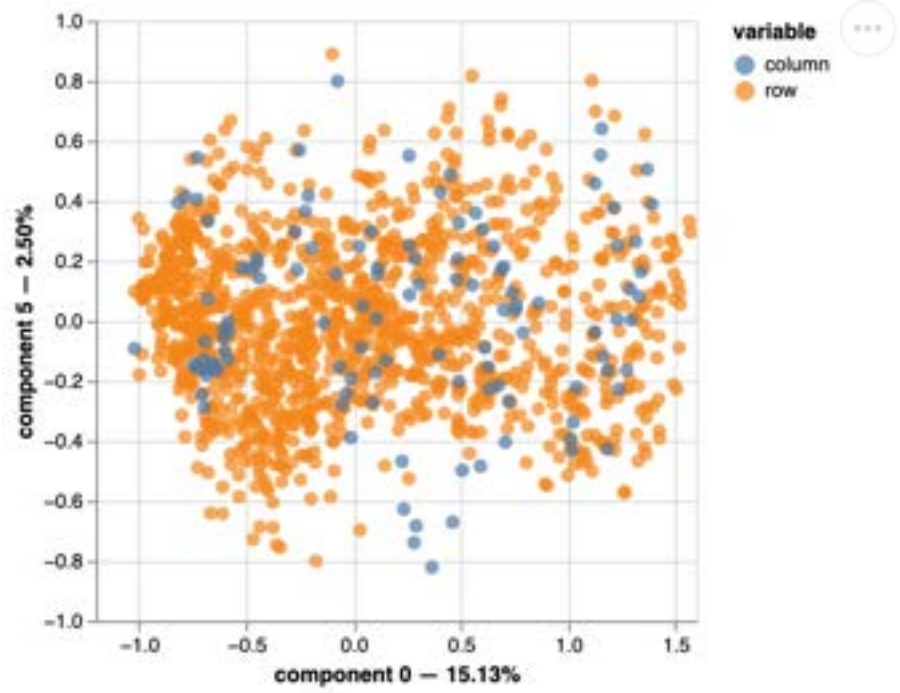
In [109…
```
mca.plot(loading_qualitative,x_component=0,y_component=4,show_column_markers
```

Out[109…



In [110…
```
mca.plot(loading_qualitative,x_component=0,y_component=5,show_column_markers
```

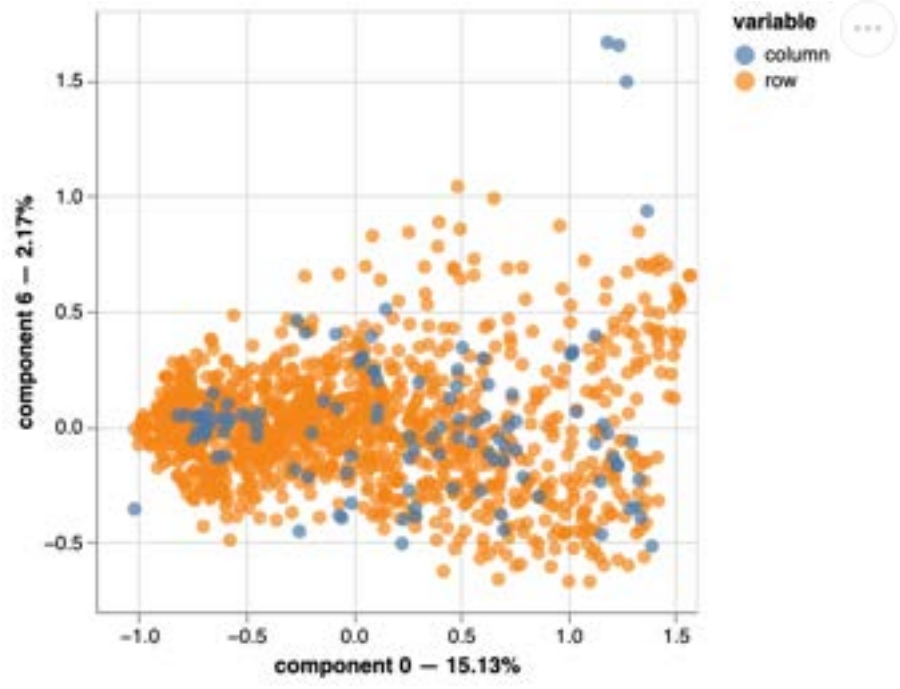
Out[110…



In [111…

```
mca.plot(loading_qualitative,x_component=0,y_component=6,show_column_markers
```

Out[111…

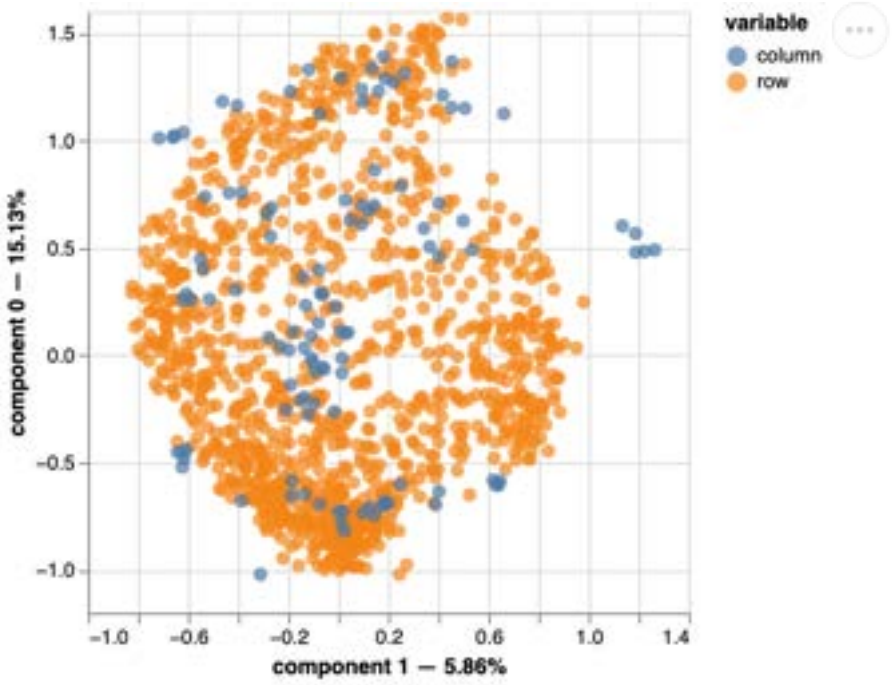


5.4.2. Individual factor maps for the second component with the six first components

The next plot visualizes the Individual factor maps for each combination of the second compenent with the six first components obtained from MCA. This visualization depicts the factor map showcasing the relationship between variables in a dataset. Each point represents a variable, and its position is determined by its contributions to the components.

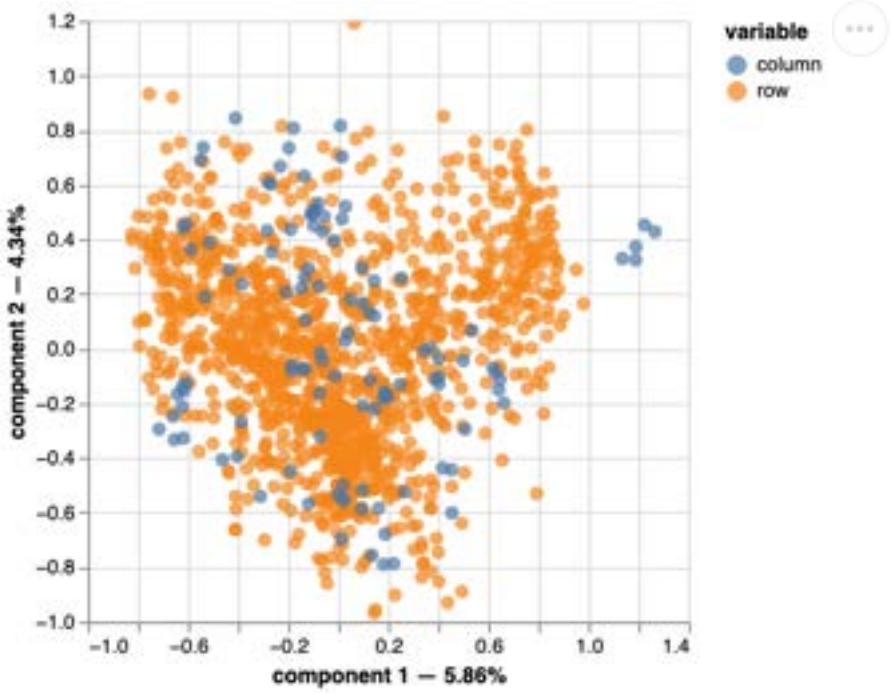
In [112…
```
mca.plot(loading_qualitative,x_component=1,y_component=0,show_column_markers
```

Out[112…



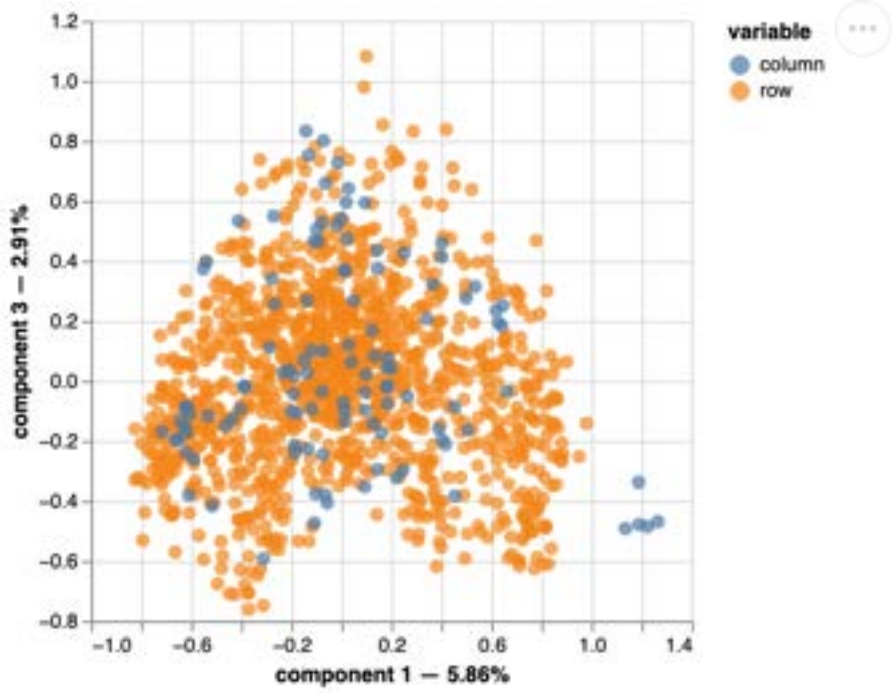
In [113…
```
mca.plot(loading_qualitative,x_component=1,y_component=2,show_column_markers
```

Out[113…



In [114…
```
mca.plot(loading_qualitative,x_component=1,y_component=3,show_column_markers
```

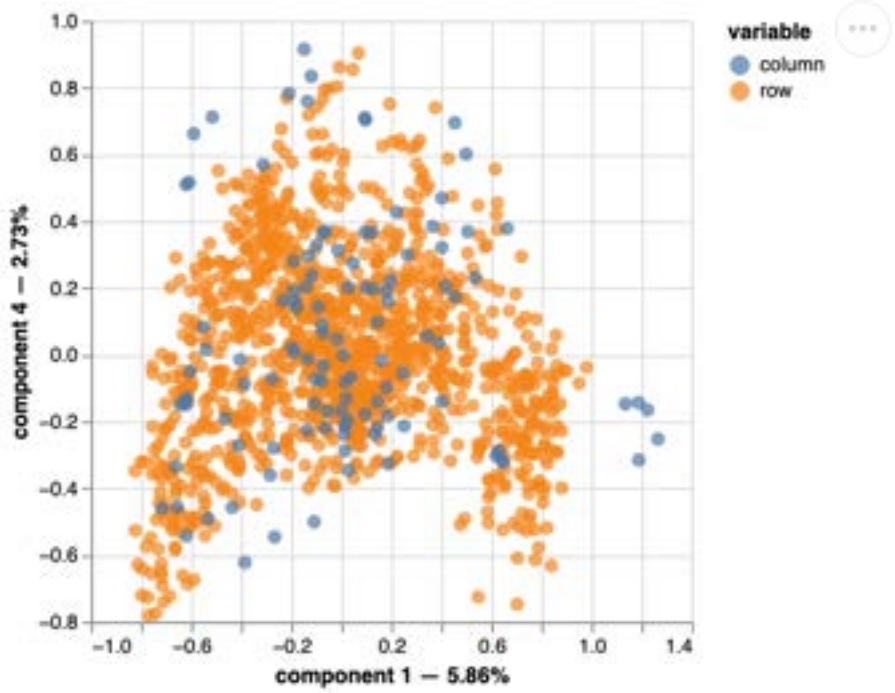
Out[114…



In [115…

```
mca.plot(loading_qualitative,x_component=1,y_component=4,show_column_markers
```

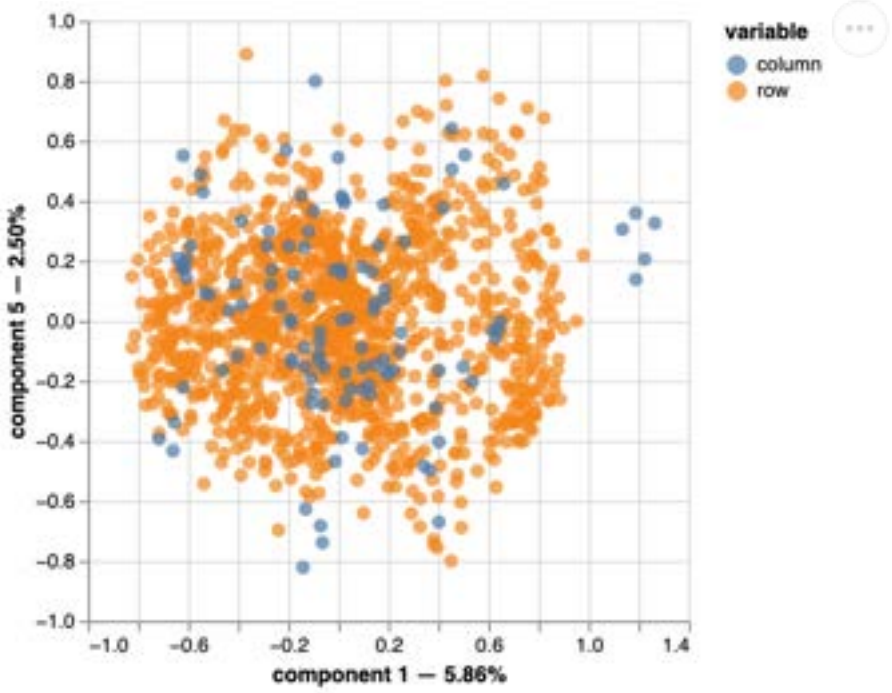
Out[115…



In [116…

```
mca.plot(loading_qualitative,x_component=1,y_component=5,show_column_markers
```

Out[116…



5.5. Which variables contribute most strongly to the axes?

In [117…

```
contrib = mca.column_contributions_.style.format('{:.1%}')
display(contrib.highlight_max(color='orange').highlight_min(color='lightblue
```

	0	1	2	3	4	5	6	7	8	9	10	
Mon00-06_A	1.4%	0.3%	0.3%	0.1%	0.5%	0.1%	0.0%	1.1%	0.0%	0.4%	0.2%	0.
Mon00-06_B	0.0%	0.2%	1.2%	0.0%	0.8%	0.0%	0.2%	0.6%	1.2%	1.0%	0.3%	5.
Mon00-06_C	0.2%	0.0%	0.2%	0.0%	0.1%	0.1%	0.1%	0.2%	0.1%	0.0%	4.0%	0.
Mon00-06_D	2.1%	0.0%	0.6%	0.1%	0.0%	0.0%	0.1%	0.1%	2.7%	0.1%	2.5%	0.
Mon06-12_A	1.4%	1.2%	0.8%	0.0%	0.1%	2.1%	0.1%	0.6%	0.0%	0.4%	0.0%	0.
Mon06-12_B	0.0%	0.0%	1.7%	0.2%	0.1%	1.9%	0.2%	0.4%	4.1%	0.2%	0.2%	0.
Mon06-12_C	0.5%	0.5%	0.0%	0.3%	0.0%	2.2%	0.8%	0.0%	1.2%	2.6%	1.4%	0.
Mon06-12_D	1.3%	1.2%	0.1%	0.0%	0.8%	1.3%	1.1%	0.1%	2.9%	2.6%	2.8%	0.
Mon12-18_A	0.8%	3.8%	0.2%	0.3%	0.1%	0.5%	0.1%	0.0%	0.4%	0.1%	0.1%	0.
Mon12-18_B	0.1%	0.0%	0.0%	3.2%	0.2%	3.6%	0.0%	0.1%	6.7%	0.0%	0.6%	0.
Mon12-18_C	0.2%	0.4%	0.0%	0.9%	0.5%	2.3%	0.4%	0.0%	0.1%	0.0%	0.4%	0.
Mon12-18_D	0.6%	6.9%	0.7%	1.1%	0.2%	1.5%	0.0%	0.0%	1.1%	0.1%	0.2%	0.
Mon18-24_A	1.8%	0.3%	0.4%	0.0%	0.5%	0.6%	0.0%	0.0%	0.7%	0.4%	0.0%	2.
Mon18-24_B	0.0%	0.2%	2.7%	0.0%	0.3%	0.0%	1.1%	0.5%	1.7%	3.3%	0.4%	1.
Mon18-24_C	0.7%	0.5%	0.2%	0.0%	2.6%	0.0%	0.1%	0.3%	0.4%	1.3%	2.9%	1.
Mon18-24_D	2.1%	0.1%	2.4%	0.0%	0.1%	1.0%	2.0%	3.0%	0.6%	0.0%	0.6%	0.
Tue00-06_A	1.1%	3.1%	0.1%	0.6%	1.4%	0.1%	0.0%	0.8%	0.0%	0.2%	0.0%	0.
Tue00-06_B	0.1%	0.1%	0.3%	0.0%	4.4%	0.5%	0.0%	0.3%	2.2%	3.3%	0.0%	0
Tue00-06_C	0.1%	1.2%	0.6%	0.5%	3.3%	0.5%	0.2%	0.0%	0.0%	2.0%	1.9%	0
Tue00-06_D	1.9%	1.8%	0.7%	0.3%	2.9%	0.5%	0.1%	0.6%	1.3%	0.1%	0.9%	0.
Tue06-12_A	1.6%	0.0%	2.5%	2.7%	0.5%	2.2%	0.0%	0.0%	0.1%	0.3%	0.1%	0
Tue06-12_B	0.0%	0.1%	2.7%	2.3%	0.1%	1.1%	0.9%	0.1%	1.6%	0.0%	0.1%	0.
Tue06-12_C	0.9%	0.0%	0.0%	0.1%	0.4%	0.8%	0.0%	0.3%	3.4%	1.8%	0.1%	0.
Tue06-12_D	1.0%	0.3%	0.6%	0.4%	1.4%	0.8%	3.1%	0.9%	0.2%	2.9%	0.3%	0.
Tue12-18_A	0.9%	3.8%	0.3%	0.1%	0.4%	0.7%	0.0%	0.0%	0.6%	0.2%	0.0%	0.
Tue12-18_B	0.1%	0.0%	0.0%	2.7%	0.9%	3.9%	0.1%	1.1%	2.7%	0.0%	0.1%	0
Tue12-18_C	0.5%	0.4%	0.1%	0.8%	1.2%	0.9%	0.0%	0.4%	0.1%	0.6%	1.2%	0.
Tue12-18_D	0.4%	7.0%	1.1%	1.9%	0.6%	1.1%	0.0%	0.1%	0.2%	0.0%	0.2%	0.
Tue18-24_A	1.8%	0.4%	0.4%	0.0%	1.0%	0.7%	0.0%	0.0%	0.8%	0.1%	0.0%	2.
Tue18-24_B	0.0%	0.2%	3.1%	0.1%	0.0%	0.6%	0.9%	1.7%	0.8%	4.9%	2.2%	0.
Tue18-24_C	0.7%	0.6%	0.4%	0.1%	1.4%	0.0%	0.0%	0.6%	0.3%	3.8%	1.3%	3.
Tue18-24_D	2.0%	0.1%	2.3%	0.0%	0.4%	0.2%	1.2%	4.5%	0.0%	0.0%	0.1%	0

	0	1	2	3	4	5	6	7	8	9	10	
Wed00-06_A	1.1%	3.1%	0.1%	0.9%	1.6%	0.0%	0.0%	0.9%	0.1%	0.0%	0.4%	0.
Wed00-06_B	0.1%	0.1%	0.2%	0.0%	6.3%	1.4%	0.4%	0.8%	0.3%	0.9%	1.1%	0
Wed00-06_C	0.1%	1.1%	0.9%	0.9%	1.8%	0.3%	1.0%	0.2%	0.0%	4.9%	0.7%	0
Wed00-06_D	1.9%	2.0%	0.7%	0.4%	2.1%	1.3%	1.4%	0.8%	0.7%	0.5%	0.5%	0.
Wed06-12_A	1.3%	0.0%	2.1%	4.5%	0.2%	2.4%	0.0%	0.3%	1.3%	0.1%	0.0%	0.
Wed06-12_B	0.0%	0.1%	2.7%	3.7%	0.4%	0.7%	2.4%	0.2%	0.3%	0.2%	0.3%	0.
Wed06-12_C	0.7%	0.1%	0.1%	0.2%	1.1%	0.4%	0.0%	1.3%	7.4%	0.6%	2.3%	0.
Wed06-12_D	1.1%	0.1%	1.4%	0.4%	0.7%	0.1%	10.2%	2.8%	0.3%	0.6%	1.6%	0.
Wed12-18_A	0.8%	3.8%	0.3%	0.2%	0.4%	0.8%	0.0%	0.0%	0.1%	0.3%	0.0%	0.
Wed12-18_B	0.1%	0.0%	0.0%	4.0%	0.9%	3.4%	1.3%	0.5%	0.1%	0.1%	0.3%	3.
Wed12-18_C	0.4%	0.6%	0.0%	0.4%	1.9%	0.1%	0.2%	0.1%	0.2%	1.4%	0.1%	2
Wed12-18_D	0.4%	6.2%	0.8%	2.0%	0.9%	0.2%	0.4%	0.8%	0.0%	0.1%	0.0%	0.
Wed18-24_A	1.9%	0.3%	0.4%	0.0%	0.8%	0.4%	0.0%	0.1%	1.5%	0.0%	0.0%	1.
Wed18-24_B	0.0%	0.1%	3.6%	0.1%	0.2%	0.2%	0.4%	2.8%	1.1%	0.4%	3.2%	0.
Wed18-24_C	0.6%	0.8%	0.1%	0.1%	1.5%	0.1%	0.2%	1.1%	0.6%	0.8%	3.9%	3.
Wed18-24_D	2.0%	0.1%	2.0%	0.0%	0.7%	0.1%	1.1%	5.7%	0.0%	0.0%	0.0%	0.
Thu00-06_A	1.1%	3.3%	0.1%	1.0%	1.8%	0.0%	0.0%	0.8%	0.0%	0.0%	0.2%	0
Thu00-06_B	0.1%	0.2%	0.2%	0.0%	4.5%	2.6%	1.9%	1.9%	0.0%	0.5%	0.6%	0.
Thu00-06_C	0.1%	0.9%	0.7%	1.1%	3.5%	0.1%	0.0%	0.9%	0.7%	3.5%	0.2%	0
Thu00-06_D	1.9%	2.4%	0.6%	0.3%	2.2%	1.7%	1.3%	0.6%	0.6%	0.8%	0.2%	0.
Thu06-12_A	1.2%	0.0%	2.3%	3.6%	0.4%	4.2%	0.0%	0.0%	1.0%	0.0%	0.0%	0
Thu06-12_B	0.0%	0.0%	2.0%	2.3%	0.8%	1.4%	3.3%	0.5%	0.4%	0.0%	0.6%	0.
Thu06-12_C	0.6%	0.0%	0.2%	0.0%	1.2%	0.1%	0.0%	3.1%	8.9%	0.6%	1.3%	0.
Thu06-12_D	1.0%	0.0%	0.6%	0.0%	1.9%	0.8%	13.4%	2.3%	2.2%	0.9%	0.2%	0.
Thu12-18_A	0.8%	4.0%	0.4%	0.3%	0.5%	1.0%	0.0%	0.0%	0.4%	0.4%	0.0%	0.
Thu12-18_B	0.1%	0.0%	0.0%	3.5%	0.7%	1.7%	2.2%	0.1%	1.4%	0.0%	0.4%	1.
Thu12-18_C	0.3%	0.4%	0.0%	0.6%	0.9%	1.6%	0.9%	0.7%	0.0%	4.9%	0.6%	0.
Thu12-18_D	0.4%	6.4%	1.2%	2.0%	0.2%	0.4%	0.7%	0.3%	0.0%	0.7%	0.0%	0
Thu18-24_A	2.0%	0.1%	0.6%	0.0%	0.8%	0.5%	0.0%	0.2%	0.8%	0.1%	0.0%	0.
Thu18-24_B	0.0%	0.0%	3.2%	0.0%	0.0%	0.3%	0.1%	1.6%	0.1%	2.6%	4.6%	0.
Thu18-24_C	0.5%	0.3%	0.8%	0.1%	0.9%	0.5%	0.2%	1.7%	1.4%	2.0%	0.1%	1.
Thu18-24_D	2.2%	0.0%	2.2%	0.1%	0.4%	0.0%	0.0%	3.0%	0.4%	0.0%	2.9%	0.

	0	1	2	3	4	5	6	7	8	9	10	
Fri00-06_A	1.1%	3.2%	0.2%	0.5%	1.7%	0.0%	0.0%	0.8%	0.1%	0.0%	0.0%	0.
Fri00-06_B	0.1%	0.1%	0.4%	0.1%	5.4%	0.8%	0.3%	2.9%	0.2%	0.4%	0.0%	0.
Fri00-06_C	0.1%	1.1%	0.8%	0.3%	1.6%	2.0%	0.6%	0.0%	0.0%	2.4%	0.1%	1
Fri00-06_D	2.0%	2.2%	0.4%	0.4%	1.2%	2.2%	1.4%	0.2%	0.0%	0.4%	0.1%	0.
Fri06-12_A	1.5%	0.0%	2.5%	1.7%	0.6%	2.5%	0.0%	0.0%	1.0%	0.0%	0.1%	1
Fri06-12_B	0.0%	0.0%	2.3%	2.4%	0.4%	0.4%	2.9%	0.7%	0.0%	0.4%	0.8%	0.
Fri06-12_C	0.7%	0.0%	0.2%	0.7%	0.8%	0.6%	0.2%	0.5%	7.4%	0.6%	0.2%	0.
Fri06-12_D	1.1%	0.0%	0.8%	0.4%	2.0%	0.2%	13.3%	0.1%	4.4%	0.1%	0.2%	1.
Fri12-18_A	1.0%	3.5%	0.6%	0.4%	0.4%	0.7%	0.1%	0.0%	0.3%	0.2%	0.0%	0.
Fri12-18_B	0.1%	0.1%	0.1%	4.0%	0.7%	3.2%	1.5%	0.0%	0.0%	0.0%	0.2%	1.
Fri12-18_C	0.3%	0.9%	0.0%	0.6%	0.4%	0.3%	0.0%	0.3%	0.0%	2.0%	0.2%	0.
Fri12-18_D	0.6%	5.2%	0.6%	2.0%	0.2%	0.9%	1.0%	0.4%	0.5%	0.6%	0.7%	0.
Fri18-24_A	1.9%	0.2%	0.2%	0.4%	0.8%	1.4%	0.1%	0.3%	1.3%	0.0%	0.0%	0.
Fri18-24_B	0.0%	0.3%	1.7%	0.8%	0.0%	0.7%	1.4%	0.4%	0.3%	1.7%	6.6%	3.
Fri18-24_C	0.6%	0.2%	0.5%	0.4%	2.0%	0.2%	1.2%	0.2%	0.1%	3.3%	0.1%	0
Fri18-24_D	2.1%	0.1%	1.7%	0.2%	0.0%	0.5%	0.3%	0.0%	2.4%	0.4%	4.1%	0.
Sat00-06_A	1.8%	1.4%	0.2%	0.5%	0.0%	1.9%	0.0%	0.0%	0.1%	0.0%	0.0%	0.
Sat00-06_B	0.0%	0.0%	1.1%	1.4%	0.1%	4.8%	0.1%	3.8%	0.2%	2.7%	0.6%	4
Sat00-06_C	0.2%	0.6%	1.3%	0.6%	0.0%	1.1%	0.1%	0.3%	0.5%	7.2%	4.1%	0
Sat00-06_D	2.7%	1.1%	1.1%	0.2%	0.4%	0.3%	0.0%	1.0%	0.3%	0.3%	0.6%	1.
Sat06-12_A	1.4%	1.5%	0.1%	0.7%	0.4%	0.6%	0.4%	0.0%	0.3%	0.0%	0.0%	0.
Sat06-12_B	0.1%	0.0%	0.9%	1.6%	0.7%	1.0%	1.4%	1.3%	0.5%	2.0%	1.4%	3
Sat06-12_C	0.1%	0.7%	1.6%	0.7%	0.0%	1.0%	0.0%	0.3%	0.8%	5.6%	5.1%	0.
Sat06-12_D	2.6%	0.8%	1.1%	0.1%	0.0%	0.2%	0.0%	2.5%	0.2%	0.7%	0.1%	2.
Sat12-18_A	1.7%	0.2%	0.1%	1.1%	0.0%	0.6%	0.0%	0.1%	0.2%	0.0%	0.1%	0.
Sat12-18_B	0.0%	0.0%	1.1%	1.6%	0.0%	0.1%	2.0%	0.0%	0.1%	0.2%	0.0%	0
Sat12-18_C	0.5%	0.0%	0.3%	2.0%	0.2%	0.2%	1.5%	0.8%	0.0%	0.1%	0.1%	1
Sat12-18_D	2.1%	0.6%	0.9%	0.3%	0.3%	1.2%	0.2%	2.1%	0.9%	0.0%	0.3%	0
Sat18-24_A	1.7%	0.4%	0.1%	1.2%	0.6%	0.5%	0.6%	0.2%	0.1%	0.1%	0.2%	1.
Sat18-24_B	0.0%	0.0%	1.1%	1.5%	1.9%	0.6%	0.6%	0.6%	0.4%	0.3%	0.5%	1.
Sat18-24_C	0.6%	0.2%	0.2%	0.9%	0.2%	0.0%	0.3%	1.6%	0.0%	0.7%	0.8%	0.
Sat18-24_D	1.9%	0.8%	1.0%	0.1%	0.2%	3.6%	2.2%	0.9%	0.8%	0.3%	0.6%	0.

	0	1	2	3	4	5	6	7	8	9	10	
Sun00-06_A	1.9%	0.2%	0.6%	1.6%	0.2%	0.4%	0.1%	0.0%	0.5%	0.0%	0.0%	2.
Sun00-06_B	0.0%	0.0%	2.6%	1.1%	0.7%	0.2%	1.7%	0.0%	1.0%	0.0%	4.7%	0.
Sun00-06_C	0.3%	0.2%	1.4%	1.7%	0.5%	0.1%	0.0%	3.1%	0.0%	0.7%	0.5%	2.
Sun00-06_D	2.8%	0.1%	1.8%	0.1%	0.5%	0.1%	0.6%	2.9%	0.0%	0.6%	1.5%	0.
Sun06-12_A	1.2%	0.5%	0.2%	1.5%	0.1%	0.2%	0.4%	0.2%	0.9%	0.2%	0.0%	5.
Sun06-12_B	0.1%	0.0%	0.7%	1.9%	0.0%	0.2%	2.0%	0.0%	1.2%	0.0%	5.2%	0.
Sun06-12_C	0.1%	0.4%	2.4%	1.5%	0.4%	0.1%	0.3%	2.0%	0.3%	0.5%	0.2%	3.
Sun06-12_D	2.8%	0.2%	1.3%	0.0%	0.4%	0.0%	0.4%	2.3%	0.0%	1.2%	2.2%	1.
Sun12-18_A	1.4%	0.4%	0.1%	1.0%	0.0%	0.4%	0.3%	0.1%	0.0%	0.0%	0.0%	2.
Sun12-18_B	0.0%	0.0%	1.1%	2.5%	0.8%	0.2%	0.5%	0.2%	0.5%	1.2%	2.4%	1.
Sun12-18_C	0.8%	0.1%	0.1%	0.8%	0.1%	0.0%	0.7%	1.5%	0.6%	1.6%	1.9%	0.
Sun12-18_D	1.6%	0.8%	0.4%	0.2%	0.9%	2.3%	0.5%	4.2%	0.0%	0.0%	0.1%	0.
Sun18-24_A	1.8%	0.1%	0.3%	1.2%	0.1%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.
Sun18-24_B	0.0%	0.1%	2.2%	0.6%	0.4%	0.1%	0.9%	0.0%	0.1%	0.1%	0.9%	1.
Sun18-24_C	0.6%	0.1%	0.3%	1.1%	0.3%	0.0%	0.2%	3.5%	0.4%	0.0%	0.0%	0.
Sun18-24_D	2.2%	0.2%	1.2%	0.0%	0.6%	0.5%	1.1%	2.7%	0.3%	0.2%	0.8%	0.
bonus_0	0.1%	0.0%	0.1%	0.1%	0.2%	0.0%	0.1%	0.0%	0.0%	0.0%	0.0%	0
bonus_1	0.9%	0.2%	0.8%	1.5%	1.5%	0.0%	0.7%	0.2%	0.0%	0.2%	0.0%	0.

We observe that Sun00-06_D is the variable that contrbutes the most for component 0 with 2.8%. Mon12-18_C is the variable that contributes the most for component 17 with the higher percentage of 14%.

5.6. Quality of the representation of individuals and variables

In [118…

```
quality_row = mca.row_cosine_similarities(loading_qualitative).head(10).styl
print("First 10 statons silarities")
display(quality_row)

quality_comun = mca.column_cosine_similarities(loading_qualitative).style.fo
print("Variables silarities")
display(quality_comun.background_gradient())
```

```
First 10 statons silarities
```

	0	1	2	3	4	5	6	7	
1	0.0343	0.133	0.0754	0.0479	0.00403	0.0751	0.0125	0.00306	0
2	0.0647	0.201	0.00407	0.0208	0.15	0.00733	0.000279	0.0249	0.0(
3	0.0118	0.00033	0.0919	0.107	0.0121	0.0149	0.0235	0.0736	0
4	0.00484	0.147	0.00361	0.0276	0.0101	0.0149	0.0053	0.0467	
5	0.0334	0.0137	0.129	0.00493	0.00082	0.0147	0.0068	0.000926	0.(
6	0.00188	0.0906	0.0396	0.0188	0.00234	0.00317	0.0158	0.0398	0
7	1.13e-05	0.112	0.0921	0.00102	0.000583	0.03	0.00315	0.0053	0
8	0.000169	0.17	0.259	0.00113	0.000347	0.0188	0.00482	0.00188	0.(
9	0.12	0.00615	0.0782	0.0774	0.0192	0.00981	0.07	0.00115	0
10	0.00498	0.00716	0.000784	0.0115	0.0235	0.00625	0.000115	0.0307	

Variables silarities

	0	1	2	3	4	5	6
Mon00-06_A	0.286	0.0211	0.0156	0.00348	0.0203	0.00337	2.9e-05
Mon00-06_B	0.0057	0.0108	0.0571	9.25e-05	0.0236	1.35e-05	0.0038
Mon00-06_C	0.0284	0.00108	0.00942	0.000226	0.00145	0.00227	0.00249
Mon00-06_D	0.35	0.00146	0.0285	0.00274	0.000284	0.00038	0.00141
Mon06-12_A	0.291	0.0951	0.0451	0.000164	0.00477	0.0705	0.00401
Mon06-12_B	5.59e-05	7.32e-05	0.0835	0.00668	0.00244	0.0557	0.00584
Mon06-12_C	0.087	0.0291	1.3e-05	0.0108	0.000781	0.0581	0.0191
Mon06-12_D	0.194	0.067	0.00601	0.000183	0.022	0.0322	0.024
Mon12-18_A	0.202	0.387	0.0163	0.0129	0.00264	0.0212	0.00327
Mon12-18_B	0.0169	0.00257	0.000692	0.0885	0.00524	0.086	0.00027
Mon12-18_C	0.0263	0.0199	0.000158	0.0261	0.0128	0.0554	0.00873
Mon12-18_D	0.106	0.46	0.0345	0.0372	0.00671	0.0423	0.000153
Mon18-24_A	0.428	0.0309	0.0254	0.00024	0.0228	0.0236	0.000782
Mon18-24_B	0.000551	0.016	0.132	0.00024	0.00784	0.000727	0.0275
Mon18-24_C	0.107	0.0277	0.0104	8.78e-05	0.0718	0.000531	0.00183
Mon18-24_D	0.319	0.00538	0.103	7.73e-05	0.00159	0.0249	0.0438
Tue00-06_A	0.232	0.252	0.0049	0.0241	0.0527	0.00216	0.00113
Tue00-06_B	0.00806	0.00392	0.0147	0.000159	0.123	0.0127	0.000143
Tue00-06_C	0.014	0.0728	0.0276	0.0144	0.0918	0.013	0.00379
Tue00-06_D	0.324	0.115	0.0321	0.00888	0.0883	0.0147	0.00134
Tue06-12_A	0.287	0.000235	0.133	0.096	0.0166	0.0665	0.00102
Tue06-12_B	0.000677	0.00699	0.168	0.094	0.00353	0.0387	0.0262
Tue06-12_C	0.156	0.000232	0.000264	0.00434	0.0122	0.0212	5.89e-07
Tue06-12_D	0.133	0.0146	0.0257	0.0105	0.0345	0.0182	0.0623
Tue12-18_A	0.226	0.371	0.023	0.00669	0.0164	0.0294	0.00175
Tue12-18_B	0.0143	0.000726	0.000319	0.0781	0.0247	0.0978	0.0017
Tue12-18_C	0.0708	0.0228	0.00238	0.0242	0.0312	0.0231	0.00104
Tue12-18_D	0.0677	0.447	0.0515	0.0614	0.0178	0.0299	0.000274
Tue18-24_A	0.421	0.0346	0.0279	0.00161	0.0438	0.029	0.000315
Tue18-24_B	0.000182	0.0105	0.144	0.00268	0.000119	0.0166	0.0211
Tue18-24_C	0.11	0.0367	0.0161	0.00314	0.0403	0.000246	0.000132
Tue18-24_D	0.314	0.00312	0.1	0.0012	0.0101	0.00476	0.0272

	0	1	2	3	4	5	6
Wed00-06_A	0.23	0.261	0.0034	0.0377	0.0621	0.000723	0.000847
Wed00-06_B	0.00943	0.00468	0.0103	0.00103	0.177	0.0367	0.00981
Wed00-06_C	0.0149	0.0671	0.0378	0.0261	0.0483	0.00772	0.0223
Wed00-06_D	0.327	0.134	0.0345	0.0123	0.0646	0.0355	0.0337
Wed06-12_A	0.241	0.000172	0.114	0.161	0.00775	0.075	0.00105
Wed06-12_B	8.88e-05	0.00809	0.167	0.155	0.015	0.0254	0.0754
Wed06-12_C	0.104	0.0034	0.00409	0.00655	0.0305	0.0104	0.000561
Wed06-12_D	0.151	0.00452	0.0575	0.00967	0.0169	0.00249	0.208
Wed12-18_A	0.193	0.369	0.0184	0.00769	0.0177	0.033	4.77e-06
Wed12-18_B	0.0155	0.000938	4.38e-05	0.118	0.0245	0.0853	0.0276
Wed12-18_C	0.0551	0.0348	0.000266	0.0107	0.0507	0.00327	0.00474
Wed12-18_D	0.0648	0.399	0.0394	0.0647	0.028	0.00543	0.00875
Wed18-24_A	0.459	0.0309	0.0307	0.00195	0.035	0.0155	0.000327
Wed18-24_B	0.00311	0.00824	0.166	0.00291	0.00507	0.00611	0.01
Wed18-24_C	0.0962	0.0501	0.00637	0.00236	0.0422	0.00157	0.00351
Wed18-24_D	0.313	0.00654	0.0868	0.00114	0.0199	0.0021	0.0234
Thu00-06_A	0.224	0.272	0.00819	0.0417	0.0681	7.51e-07	0.000173
Thu00-06_B	0.0131	0.00906	0.00879	0.000296	0.126	0.0669	0.0419
Thu00-06_C	0.013	0.0507	0.0288	0.033	0.0968	0.00144	0.00035
Thu00-06_D	0.324	0.162	0.0274	0.00895	0.0667	0.0487	0.0309
Thu06-12_A	0.224	1.87e-07	0.121	0.125	0.012	0.126	0.000194
Thu06-12_B	0.00187	0.00269	0.121	0.0943	0.0315	0.0504	0.101
Thu06-12_C	0.0998	0.00233	0.00753	0.000382	0.0361	0.00214	0.00055
Thu06-12_D	0.139	0.000924	0.0266	0.00093	0.0491	0.0181	0.275
Thu12-18_A	0.183	0.374	0.0252	0.0159	0.02	0.0395	0.00148
Thu12-18_B	0.0101	3.44e-05	0.00197	0.104	0.0193	0.043	0.0499
Thu12-18_C	0.042	0.0218	2.71e-06	0.017	0.0242	0.0405	0.0194
Thu12-18_D	0.0637	0.406	0.0559	0.0638	0.00735	0.0116	0.0165
Thu18-24_A	0.488	0.00867	0.0394	0.000411	0.0365	0.0217	0.0016
Thu18-24_B	0.00274	1.46e-05	0.146	0.0011	8.27e-07	0.00673	0.00131
Thu18-24_C	0.0841	0.0159	0.0367	0.00243	0.0254	0.0121	0.00402
Thu18-24_D	0.35	2.6e-05	0.101	0.00205	0.0119	2.61e-06	0.000856

	0	1	2	3	4	5	6
Fri00-06_A	0.236	0.264	0.0143	0.0215	0.0627	0.000454	0.000296
Fri00-06_B	0.0165	0.00318	0.019	0.00228	0.153	0.0195	0.00747
Fri00-06_C	0.0112	0.0642	0.0327	0.00932	0.0434	0.0507	0.0124
Fri00-06_D	0.358	0.151	0.0212	0.013	0.039	0.0648	0.0351
Fri06-12_A	0.266	6.22e-05	0.13	0.059	0.0198	0.0755	0.00116
Fri06-12_B	0.0025	0.00187	0.137	0.0954	0.0165	0.0136	0.0848
Fri06-12_C	0.122	0.000679	0.00998	0.022	0.0229	0.0156	0.00401
Fri06-12_D	0.154	0.000877	0.0345	0.0124	0.0507	0.00525	0.274
Fri12-18_A	0.221	0.318	0.0367	0.0195	0.0182	0.0252	0.0024
Fri12-18_B	0.0117	0.00363	0.00239	0.121	0.0187	0.0839	0.0342
Fri12-18_C	0.0442	0.0523	0.00081	0.0182	0.00985	0.00742	0.000376
Fri12-18_D	0.0933	0.33	0.028	0.062	0.00553	0.0239	0.0226
Fri18-24_A	0.468	0.0151	0.0126	0.019	0.0368	0.0569	0.00179
Fri18-24_B	0.00142	0.0161	0.0774	0.0244	0.00107	0.0187	0.0322
Fri18-24_C	0.0863	0.013	0.0227	0.0119	0.0542	0.00529	0.0263
Fri18-24_D	0.336	0.00568	0.0756	0.00675	6.66e-05	0.0141	0.00577
Sat00-06_A	0.436	0.135	0.0107	0.0226	0.00117	0.0749	0.00142
Sat00-06_B	0.00106	0.00166	0.0496	0.0412	0.00394	0.123	0.00124
Sat00-06_C	0.023	0.0346	0.0544	0.0159	0.000774	0.027	0.00173
Sat00-06_D	0.475	0.073	0.056	0.00755	0.0123	0.00927	0.000274
Sat06-12_A	0.334	0.134	0.00888	0.0316	0.016	0.0229	0.0143
Sat06-12_B	0.00948	0.00185	0.0382	0.0481	0.0203	0.025	0.0314
Sat06-12_C	0.0209	0.0374	0.0693	0.0201	3.12e-05	0.0236	2.6e-08
Sat06-12_D	0.461	0.0559	0.0531	0.00294	7.56e-05	0.00469	4.38e-05
Sat12-18_A	0.449	0.0192	0.00592	0.0552	0.000171	0.0247	0.00105
Sat12-18_B	0.00388	0.0011	0.0487	0.0482	0.00102	0.00299	0.0447
Sat12-18_C	0.081	0.00146	0.0146	0.0583	0.00522	0.00541	0.0327
Sat12-18_D	0.328	0.0386	0.0424	0.00992	0.00937	0.0318	0.00386
Sat18-24_A	0.431	0.0349	0.00384	0.0578	0.0258	0.0194	0.0209
Sat18-24_B	0.00174	0.00249	0.051	0.0461	0.0527	0.0156	0.0126
Sat18-24_C	0.091	0.00918	0.00984	0.0267	0.00665	0.000231	0.00685
Sat18-24_D	0.31	0.0475	0.0454	0.00173	0.00703	0.0953	0.0502

	0	1	2	3	4	5	6
Sun00-06_A	0.436	0.0164	0.0387	0.0684	0.00764	0.0167	0.00219
Sun00-06_B	0.00166	5.2e-05	0.121	0.0334	0.0203	0.00598	0.0396
Sun00-06_C	0.0505	0.0121	0.0596	0.0498	0.0128	0.00236	0.000678
Sun00-06_D	0.465	0.0037	0.0846	0.00229	0.0148	0.00174	0.0137
Sun06-12_A	0.273	0.0458	0.013	0.0665	0.0023	0.0076	0.0121
Sun06-12_B	0.0146	6.92e-05	0.0329	0.0565	0.000437	0.00611	0.0449
Sun06-12_C	0.0137	0.0248	0.105	0.0417	0.0108	0.00223	0.00545
Sun06-12_D	0.474	0.0116	0.0632	0.000566	0.0114	8.62e-06	0.00908
Sun12-18_A	0.383	0.039	0.00732	0.0518	6.62e-05	0.0184	0.0109
Sun12-18_B	0.00182	0.000143	0.0487	0.0738	0.0213	0.0053	0.0103
Sun12-18_C	0.123	0.00346	0.00237	0.0233	0.00325	0.000591	0.015
Sun12-18_D	0.254	0.0489	0.0167	0.00512	0.0262	0.0587	0.0107
Sun18-24_A	0.438	0.00509	0.024	0.055	0.00391	0.00433	0.00101
Sun18-24_B	0.000321	0.0047	0.101	0.0179	0.0128	0.00198	0.0218
Sun18-24_C	0.0866	0.00357	0.011	0.0336	0.00849	0.000251	0.00378
Sun18-24_D	0.344	0.0139	0.0549	0.000513	0.0178	0.0139	0.0238
bonus_0	0.101	0.0138	0.0287	0.0354	0.0389	0.000571	0.0128
bonus_1	0.115	0.0107	0.0322	0.0387	0.0357	0.000933	0.0139

Part 6: Clustering with Multiple Correspondence Analysis

6.1. K-means clustering on qualitaive data

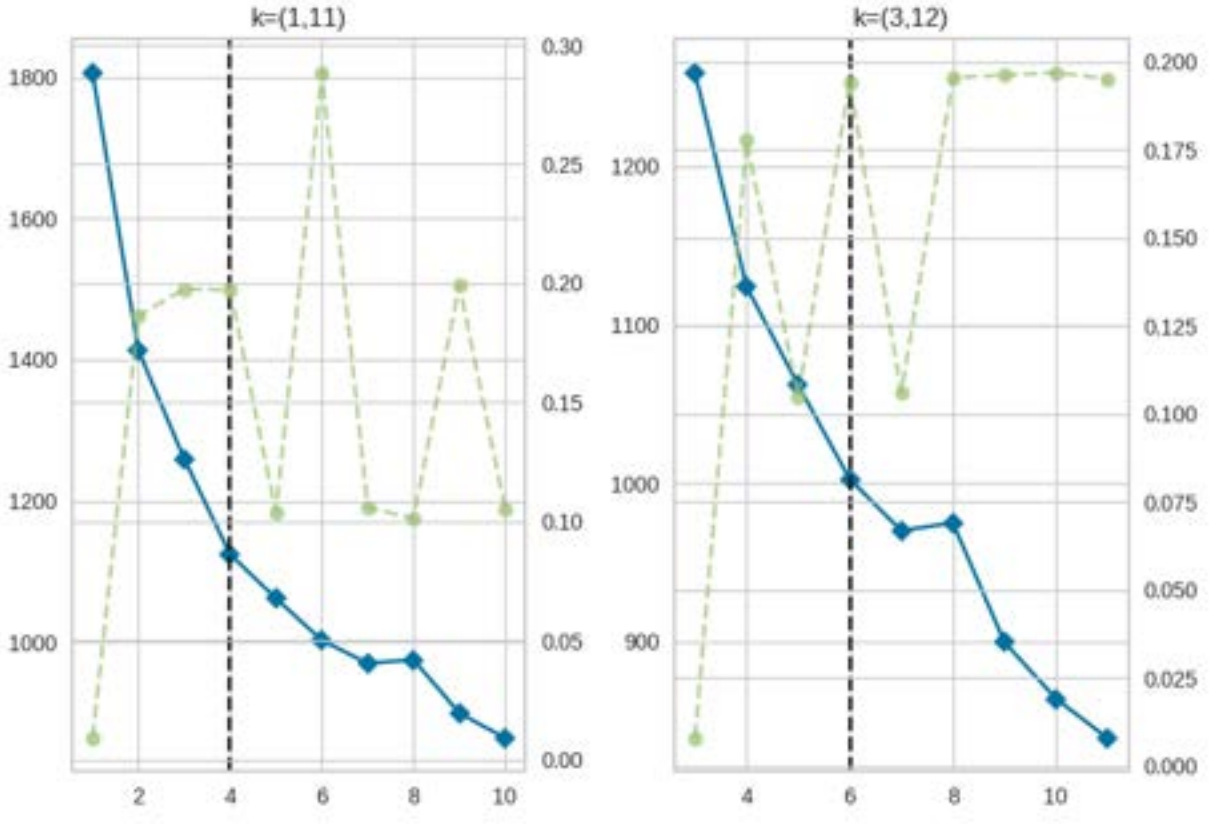
6.1.1. Selection of the number of clusters

6.1.1.1. Determining the number of clusters using the total within sum of square as metric

In [119…

```
kmeans = KMeans(init='k-means++', max_iter=100, n_init="auto", random_state=
visualizer_1 = KElbowVisualizer(kmeans, k=(1,11), metric='distortion')
visualizer_2 = KElbowVisualizer(kmeans, k=(3,12), metric='distortion')

plt.subplot(1,2,1)
visualizer_1.fit(loading_mca16)
plt.title("k=(1,11)")
```

```
plt.subplot(1,2,2)
visualizer_2.fit(loading_mca16)
plt.title("k=(3,12)")

plt.tight_layout()
plt.show()
```

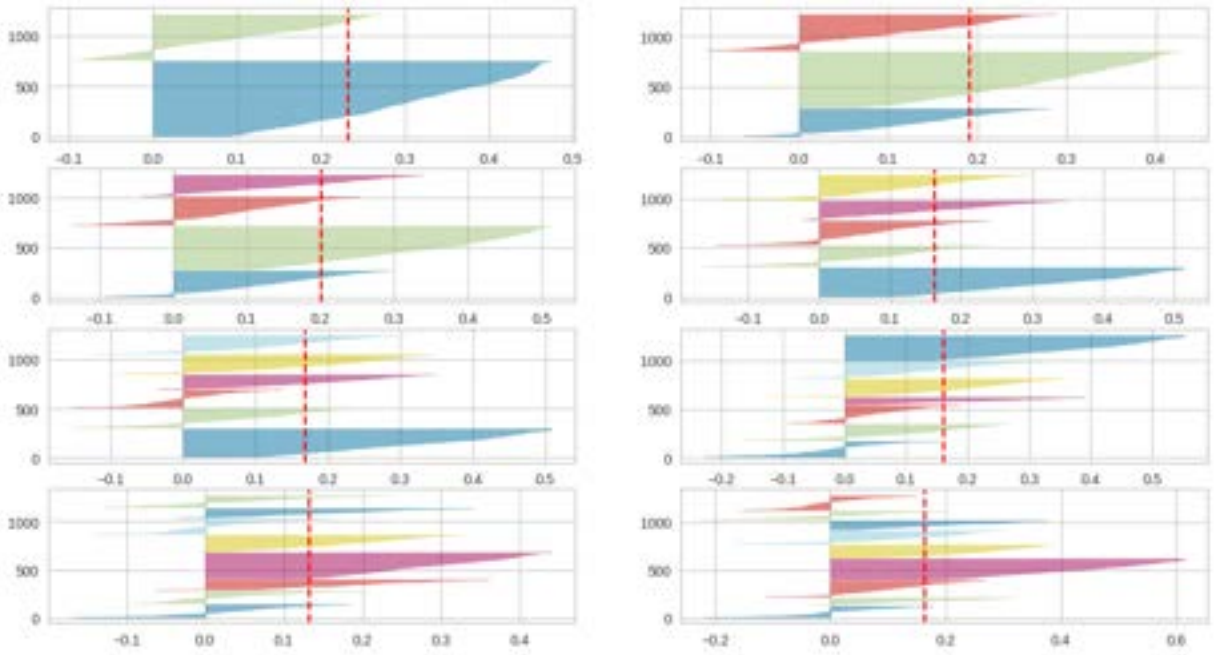


Here we observe an anomaly in the function KElbowVisualizer. When we are looking for the best number of clusters in two different intervals we get different values. Since we do not know which one is better we will perform the k-means clustering with k=4 and k=6 to see which one is actually better.

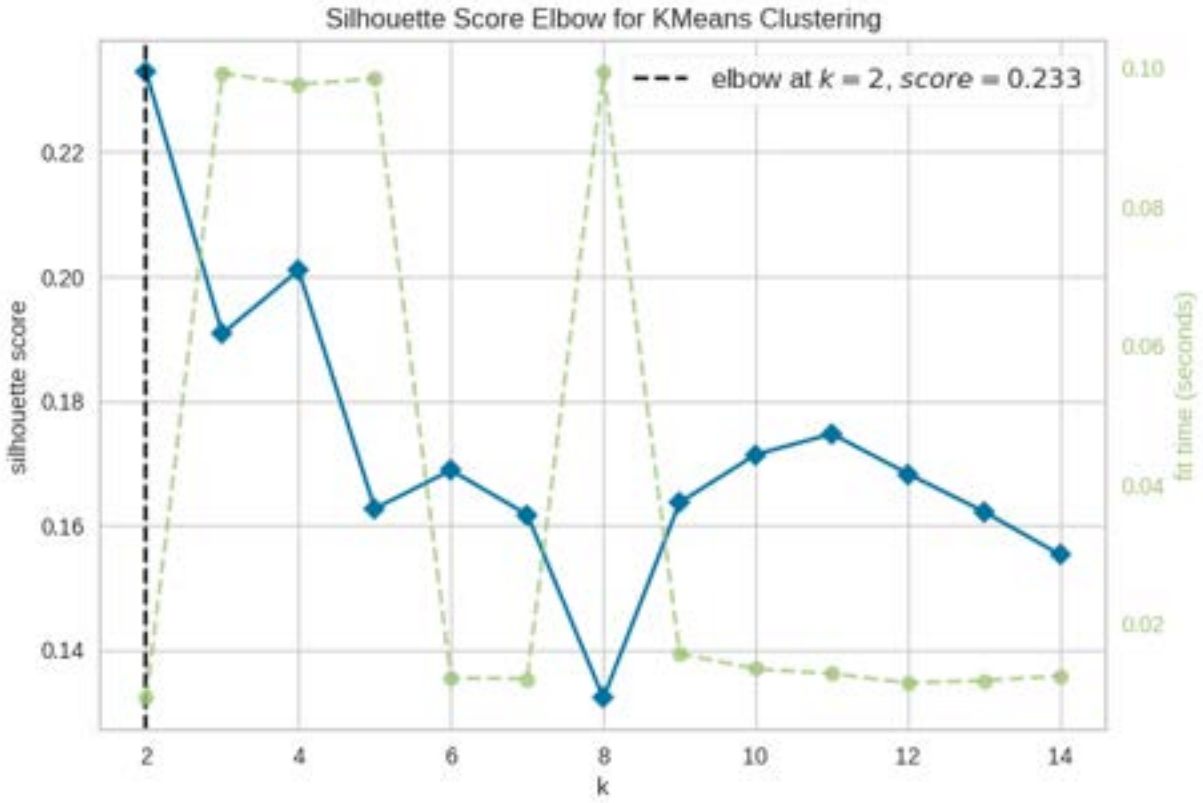
6.1.1.2. Determining the number of clusters using the silhouette scores

In [120…

```
fig, ax = plt.subplots(4, 2, figsize=(15,8))

for k in range(2, 10):
    kmeans = KMeans(n_clusters=k, init='k-means++',n_init="auto", max_iter=1
    q, mod = divmod(k, 2)

    # Create SilhouetteVisualizer instance with KMeans instance
    visualizer = SilhouetteVisualizer(kmeans, colors='yellowbrick', ax=ax[q-
    visualizer.fit(loading_mca16)
```



In [121…
```
visualizer = KElbowVisualizer(kmeans, k=(2,15), metric='silhouette')

visualizer.fit(loading_mca16)
visualizer.show()
```



Out[121…
```
<Axes: title={'center': 'Silhouette Score Elbow for KMeans Clustering'}, xl
abel='k', ylabel='silhouette score'>
```

According to the distortion score elbow method, the best K values are either 4 or 6.

For the silhouette score elbow method, the optimal K value is 2.

We will further investigate the number of clusters using K values of 2, 4, and 6.

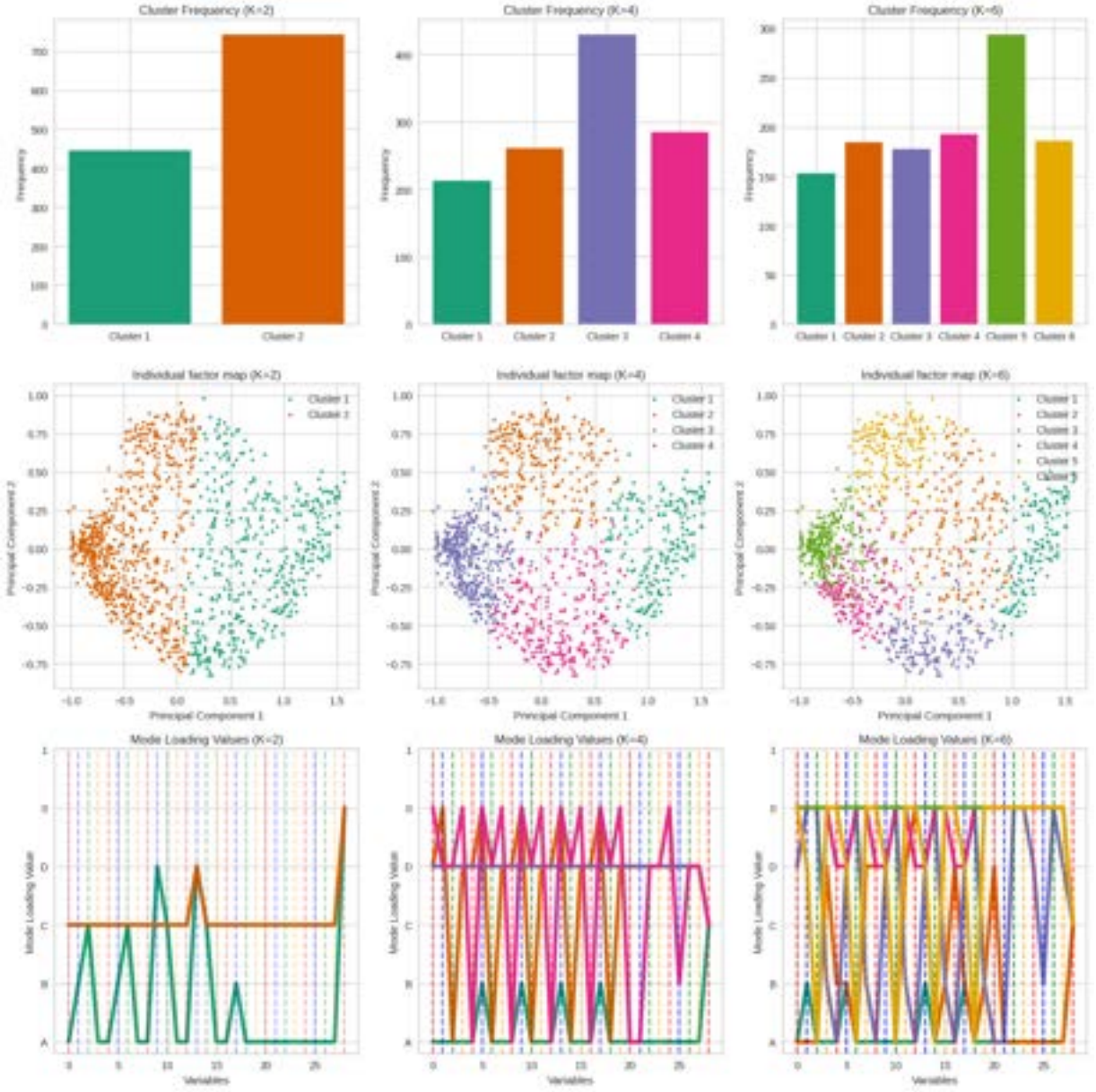
6.1.2. Visualization and interpretation of k-means clusters

6.1.2.1. Three descriptive plots of cluster

In [122…

```
K_values = [2, 4, 6]

# Define the colormap outside the loop
cmap = plt.get_cmap('Dark2')

# Define four different colors for the vertical lines
line_colors = ['red', 'blue', 'green', 'orange']

fig, axs = plt.subplots(3, len(K_values), figsize=(15, 15))

# Define the desired y-axis tick positions and labels
ytick_positions = [0, 1, 2, 3, 4, 5]
ytick_labels = ['A', 'B', 'C', 'D', '0', '1']

for i, K in enumerate(K_values):
    kmeans_loading = KMeans(n_clusters=K, init='k-means++',n_init="auto", ra
    clusters_kmeans_loading = kmeans_loading.fit_predict(loading_mca16)
    cluster_freq = np.unique(clusters_kmeans_loading, return_counts=True)
    cluster_names = [f"Cluster {i+1}" for i in range(K)]

    # Plot Cluster Frequency
    axs[0, i].bar(cluster_names, cluster_freq[1], color=cmap.colors)
    axs[0, i].set_ylabel("Frequency")
    axs[0, i].set_title(f"Cluster Frequency (K={K})")

    # Plot Kmeans Graph
    for j in range(K):
        axs[1, i].scatter(loading_mca.iloc[clusters_kmeans_loading == j, 0],
    axs[1, i].set_title(f"Individual factor map (K={K})")
    axs[1, i].legend()
    axs[1, i].set_xlabel("Principal Component 1")
    axs[1, i].set_ylabel("Principal Component 2")

    # Plot mode of each qualitative variable
    x = np.arange(loading_qualitative.shape[1])
    for j in range(K):
        mode_loading = loading_qualitative.iloc[clusters_kmeans_loading == j
        axs[2, i].plot(x, mode_loading.tolist(), color=cmap.colors[j], linew

        # Add vertical lines with custom colors
        for k in range(len(x)):
            axs[2, i].axvline(x=k, color=line_colors[k % 4], linestyle='--',

    axs[2, i].set_xlabel("Variables")
    axs[2, i].set_ylabel("Mode Loading Value")
    axs[2, i].set_title(f"Mode Loading Values (K={K})")

    # Set y-axis tick positions and labels
```

```
    axs[2, i].set_yticks(ytick_positions)
    axs[2, i].set_yticklabels(ytick_labels)

plt.tight_layout()
plt.show()
```



The Mode Loading Values plots have vertical dotted lines indicading the different intervals of time of each day.

Red vertical dotted line = 00 am to 6 am

Blue vertical dotted line = 6 am to 12 pm

Green vertical dotted line = 12 pm to 6 pm

Yellow vertical dotted line = 6 pm to 00 am

Reminder: Considerating our previous analysis of the variable factor maps, we said that

component 1 separetes the variables according night or day hours. Those being in the negative part correspond to the night hours.

First of all, we observe that for all Mode Loading Value plots, from variable 20 (corresponding to Sat_00-06), all the clusters have a different behaviour. This is expected as the activities on week-ends are different from those of week-days.

*k=2

In the individual factor map, the clusters are perfectly separeted by the axis 0 of the component 1, meaning that Cluster1 represents the stations full during the day and Cluster2 representes the stations full during the night. The Mode Loading Values plot verifies this interpretation because Cluster1 is on avarage full (classes C and D) between 6 am to 12 pm (blue vertical dotted line) and 12pm 6 pm (green vertical dotted line). Cluster2 correspond to the stations mostly full all the time.

*k=4

In the individual factor map, the clusters 1 and 3 are perfectly separeted by the axis 0 of the component 1, meaning that Cluster1 represents the stations full during the day and cluster3 representes the stations full during the night. The Mode Loading Values plot verifies this interpretation because Cluster1 is the fullest between 6 am to 12 pm (blue vertical dotted line).

Cluster4 correspond to the stations full every 18 hours: between 6 am to 00 am (blue, green and yellow vertical dotted lines). This corresponds to the stations mostly full during the day.

We observe that cluster 2 has a period of 18 hours, being the fullest between 6 pm to 12 pm (yellow, red and blue vertical dotted lines). This corresponds to the stations mostly full expect during the afternoon.

*k=6

In the individual factor map, the clusters 4 and 5 are perfectly separeted from clusters 1 and 2 by the axis 0 of the component 1, meaning that clusters 4 and 5 represents the stations full during the night and clusters 1 and 2 representes the stations full during the day. The Mode Loading Values plot verifies this interpretation because Cluster4 is the fullest between 00 am to 12 pm (red and blue vertical dotted lines). We observe on Mode Loading Value plot that cluster5 actually represents the stations that are on flat ground.

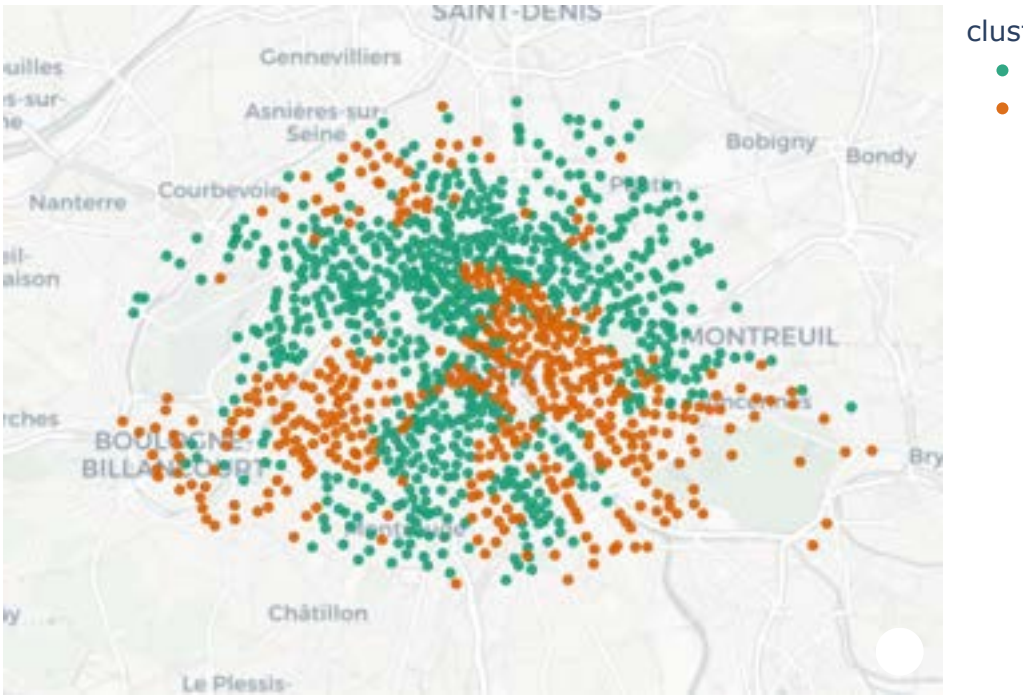
6.1.2.2. Visualization of clusters on the map

We will now compare the stations colored by clusters and the stations colored by hill position to see if they match.

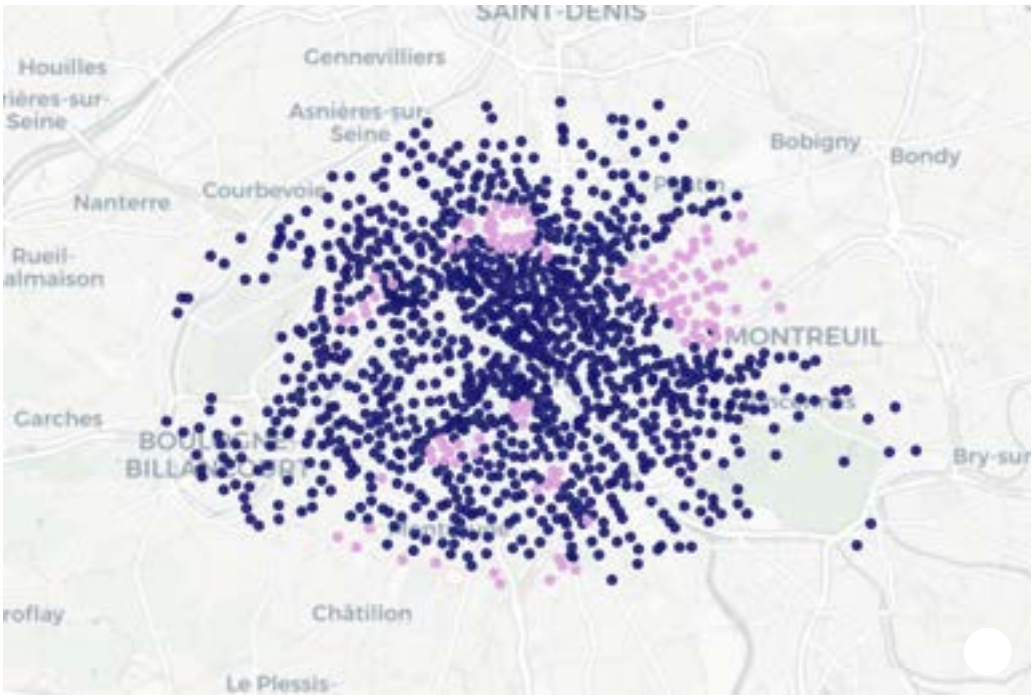
In [123…

```
K = 2

# Perform KMeans clustering on your data
kmeans_loading = KMeans(n_clusters=K, init='k-means++', n_init="auto",random
clusters_kmeans_loading = kmeans_loading.fit_predict(loading_mca16)

# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_kmeans_loading]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')
# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



Earlier we said that cluster2 correspond to the stations mostly full all the time. Hovewer the stations on hills are in cluster2, which is not consistent.

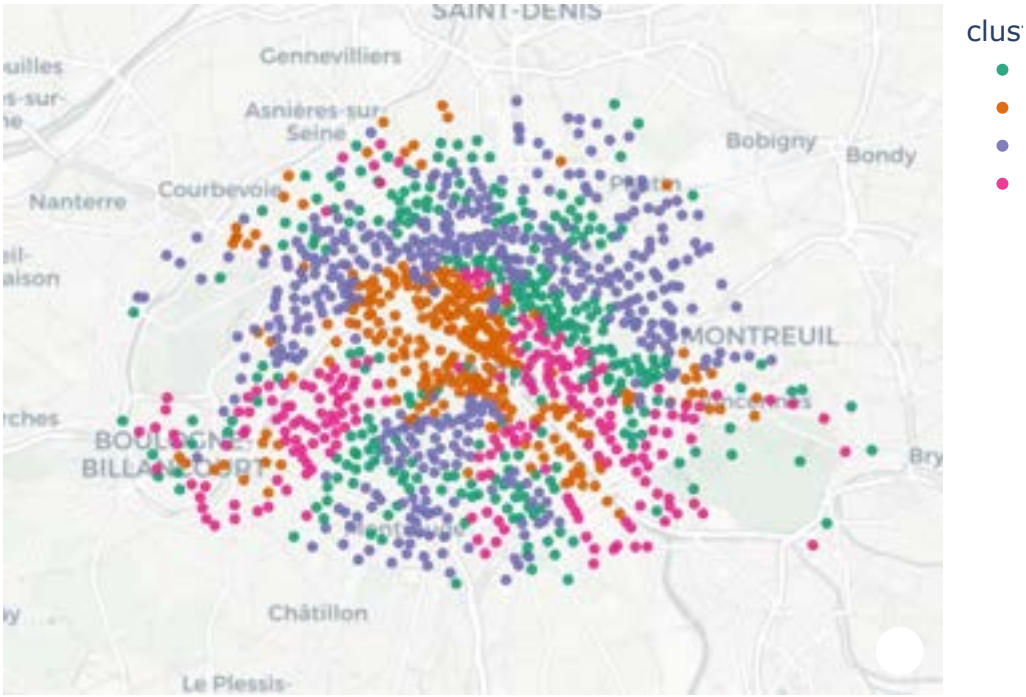
Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

In [124…

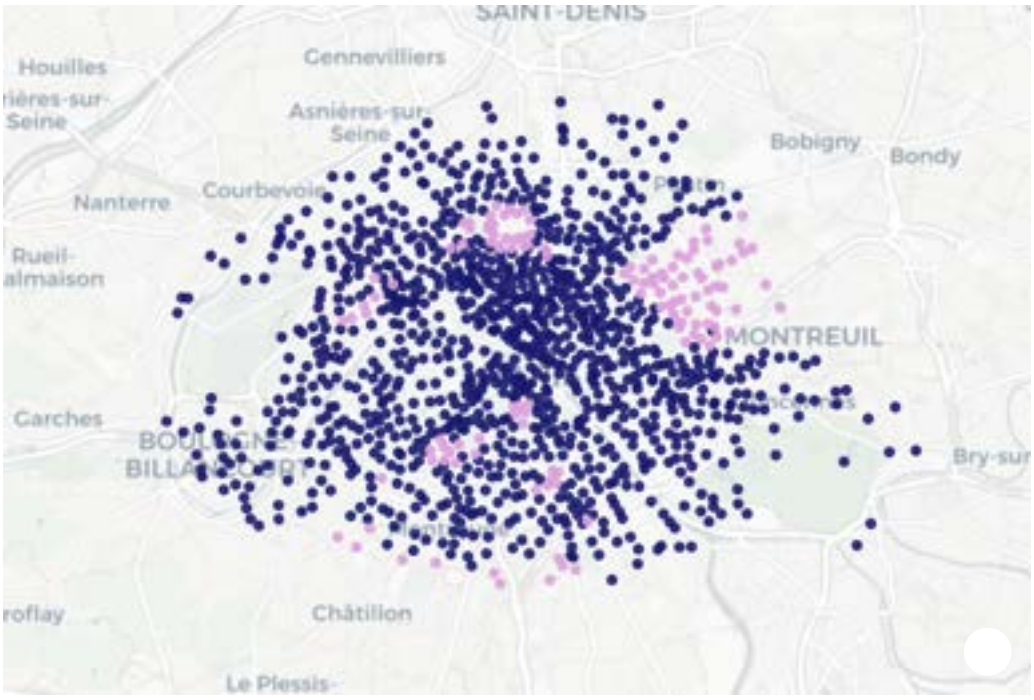
```
K = 4

# Perform KMeans clustering on your data
kmeans_loading = KMeans(n_clusters=K, init='k-means++',n_init="auto", random
clusters_kmeans_loading = kmeans_loading.fit_predict(loading_mca16)

# Convert cluster labels to strings for better visualization
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_kmeans_loading]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
```

```
                        mapbox_style="carto-positron", zoom=10, opacity=0.9,
                        title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                        color_discrete_map={0: 'midnightblue', 1: 'plum'},
                        mapbox_style="carto-positron", zoom=10, opacity=0.9,
                        title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



We noted that Cluster1 is the fullest between 6 am to 12 pm (blue vertical dotted line) which is not coherent with the location of this cluster.

We observed that cluster 2 corresponds to the stations mostly full all the time expect during the afternoon. This cluster matches the location of the city-center. This behaviour could be explained by the fact that people could be riding during the afternoon.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.

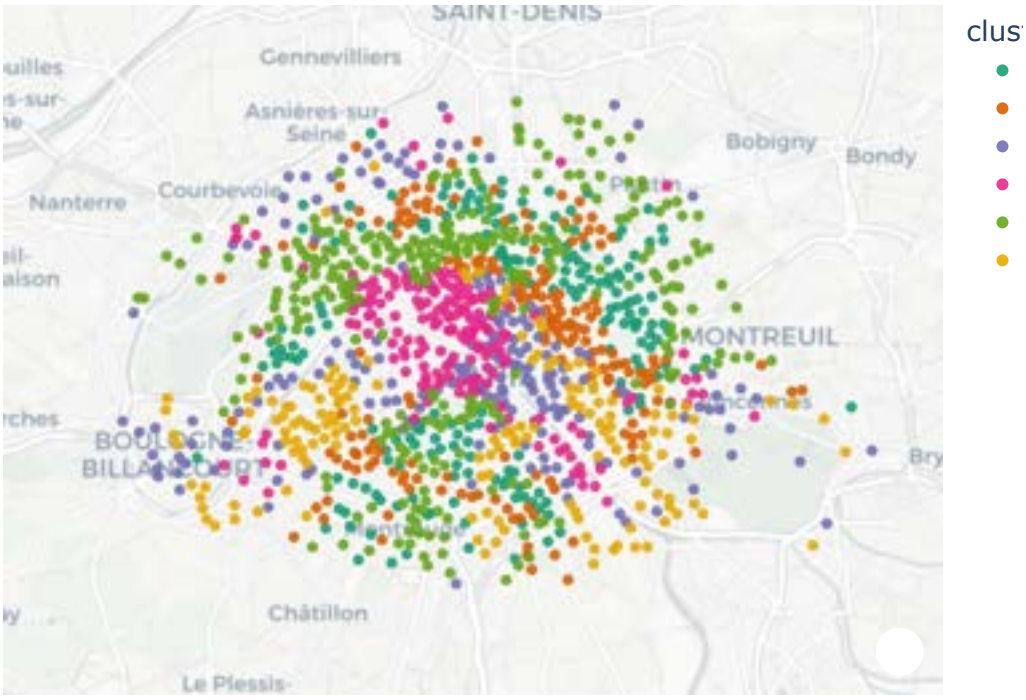
- Visualization k=6

In [125…

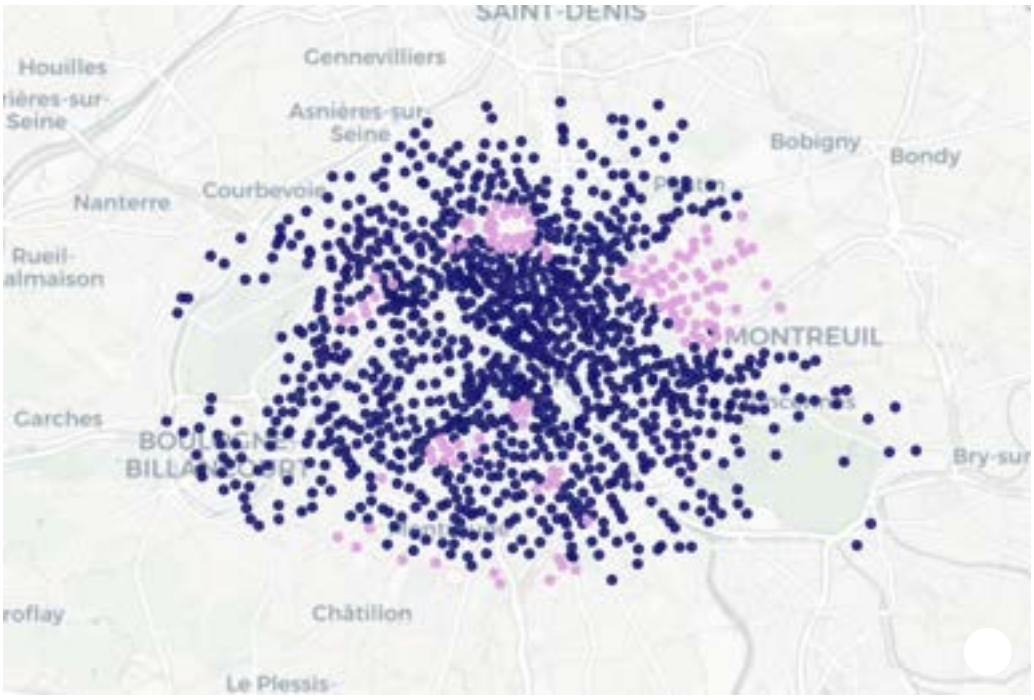
```
K = 6

# Perform KMeans clustering on your data
kmeans_loading = KMeans(n_clusters=K, init='k-means++',n_init="auto", random
clusters_kmeans_loading = kmeans_loading.fit_predict(loading_mca16)

# Convert cluster labels to strings for better visualization
```

```
cluster_names = [f"Cluster {i+1}" for i in range(K)]
cluster_labels = [f"Cluster {label+1}" for label in clusters_kmeans_loading]

# Add cluster labels to the DataFrame
coord['cluster'] = cluster_labels
coord['hill'] = coord['bonus'].astype('category') # convert to categorical

# Plot the stations on a map colored by clusters using Plotly
fig1 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='clus
                         color_discrete_sequence=px.colors.qualitative.Dark2,
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Stations by Clusters')

# Plot the stations on a map colored by hill status using Plotly
fig2 = px.scatter_mapbox(coord, lat='latitude', lon='longitude', color='hill
                         color_discrete_map={0: 'midnightblue', 1: 'plum'},
                         mapbox_style="carto-positron", zoom=10, opacity=0.9,
                         title='Hilltop stations')

# Plot the first plot on the left subplot
#fig1.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig1.show()

# Plot the second plot on the right subplot
#fig2.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
fig2.show()
```

Stations by Clusters



Hilltop stations



Unfortunately the MCA analysis we performed was not conclusive. It may be because of the way we created our qualitative variables, splitting them by day/night and judging how full they are by comparing the average loading to quantiles could have yielded better results. Additionnaly, computing the mode while considering how full the stations are and their location on a hill does not seem really pertinent.

Nb. there is an issue with the colors of the clusters, we will only use their names to distinguish them.