

correlation

May 30, 2022

```
[1]: import os

import networkx as nx
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
```

```
[2]: import plotly.io as pio
pio.renderers.default = "plotly_mimetype+notebook+vscode+pdf"
```

0.1 Modify Data

0.1.1 Create Merged .csv File with Data from All Cities

```
[3]: # setup global variables - data file directory and name
DATA_FILES_DIR = 'original_data_files'
DATA_FILE_NAME = 'data/data.csv.gz'

all_data_files = os.listdir(DATA_FILES_DIR)

def create_data_file():
    # read each data original data file and concatinate it to single df
    os.chdir(DATA_FILES_DIR)
    df = pd.concat(map(pd.read_csv, all_data_files), ignore_index=True)
    os.chdir('..')    # return to previous dir - main dir

    # remove some patterns from city column
    df['city'] = df['city'].str.replace(',Croatia', '')
    df['city'] = df['city'].str.replace(r'+', ' ', regex=False)

    # sort data by datetime and city and save it to .csv file
    df = df.sort_values(by=['date_time', 'city'])
    df.to_csv(DATA_FILE_NAME, index=False, compression='gzip')
    print('Data processed successfully')
```

```

# create data file if does not exist
if not os.path.exists(DATA_FILE_NAME):
    print('Creating data file')
    create_data_file()
else:
    print('Data has already been processed')

```

Data has already been processed

0.2 Import Data & Data Info

```
[4]: # import data
df_data = pd.read_csv(DATA_FILE_NAME, compression='gzip')
df_data.info()
```

#	Column	Non-Null Count	Dtype
0	date_time	1066968	non-null
1	sunrise	1066968	non-null
2	sunset	1066968	non-null
3	moonrise	1066968	non-null
4	moonset	1066968	non-null
5	moon_phase	1066968	non-null
6	moon_illumination	1066968	non-null
7	time	1066968	non-null
8	tempC	1066968	non-null
9	tempF	1066968	non-null
10	windspeedMiles	1066968	non-null
11	windspeedKmph	1066968	non-null
12	winddirDegree	1066968	non-null
13	winddir16Point	1066968	non-null
14	weatherCode	1066968	non-null
15	weatherIconUrl	1066968	non-null
16	weatherDesc	1066968	non-null
17	precipMM	1066968	non-null
18	precipInches	1066968	non-null
19	humidity	1066968	non-null
20	visibility	1066968	non-null
21	visibilityMiles	1066968	non-null
22	pressure	1066968	non-null
23	pressureInches	1066968	non-null
24	cloudcover	1066968	non-null
25	HeatIndexC	1066968	non-null
26	HeatIndexF	1066968	non-null

```
27 DewPointC          1066968 non-null  int64
28 DewPointF          1066968 non-null  int64
29 WindChillC         1066968 non-null  int64
30 WindChillF         1066968 non-null  int64
31 WindGustMiles      1066968 non-null  int64
32 WindGustKmph        1066968 non-null  int64
33 FeelsLikeC          1066968 non-null  int64
34 FeelsLikeF          1066968 non-null  int64
35 uvIndex             1066968 non-null  int64
36 city                1066968 non-null  object
dtypes: float64(2), int64(25), object(10)
memory usage: 301.2+ MB
```

```
[5]: # check null values -> no null values
df_data.isna().sum()
```

```
[5]: date_time          0
sunrise              0
sunset               0
moonrise              0
moonset               0
moon_phase            0
moon_illumination     0
time                  0
tempC                 0
tempF                 0
windspeedMiles        0
windspeedKmph          0
winddirDegree          0
winddir16Point         0
weatherCode            0
weatherIconUrl         0
weatherDesc             0
precipMM               0
precipInches            0
humidity               0
visibility              0
visibilityMiles         0
pressure               0
pressureInches          0
cloudcover              0
HeatIndexC              0
HeatIndexF              0
DewPointC               0
DewPointF               0
WindChillC              0
WindChillF              0
```

```
WindGustMiles      0
WindGustKmph       0
FeelsLikeC         0
FeelsLikeF         0
uvIndex            0
city               0
dtype: int64
```

```
[6]: df_data.head(5)
```

```
[6]:          date_time    sunrise    sunset   moonrise   moonset moon_phase \
0  2016-10-31 00:00:00  06:40 AM  04:50 PM  07:02 AM  05:38 PM  New Moon
1  2016-10-31 00:00:00  06:42 AM  04:54 PM  07:05 AM  05:41 PM  New Moon
2  2016-10-31 00:00:00  06:38 AM  04:50 PM  07:00 AM  05:38 PM  New Moon
3  2016-10-31 00:00:00  06:26 AM  04:44 PM  06:48 AM  05:31 PM  New Moon
4  2016-10-31 00:00:00  06:41 AM  04:52 PM  07:04 AM  05:39 PM  New Moon

      moon_illumination  time  tempC  tempF  ...  DewPointC  DewPointF \
0                      0     0     5     40  ...        2        36
1                      0     0    14     58  ...        8        47
2                      0     0     9     48  ...        5        41
3                      0     0    14     57  ...        5        41
4                      0     0     5     40  ...        2        36

      WindChillC  WindChillF  WindGustMiles  WindGustKmph  FeelsLikeC  FeelsLikeF \
0              1         34             16             25            1           34
1             12         54             22             35            12           54
2              6         43             17             28            6           43
3             12         53             20             33            12           53
4              1         34             16             25            1           34

      uvIndex  city
0        3  Bakar
1        4  Bale
2        3  Banjol
3        4  Baska
4        3  Boljun

[5 rows x 37 columns]
```

1 Correlation

```
[7]: def create_folder(folder_name):
    if not os.path.exists(folder_name):
        print(f'Creating folder {folder_name}')
        os.mkdir(folder_name)
```

```
# to always have the newest plot versions, delete file before creating new one
def remove_file_if_exists(file_path):
    if os.path.exists(file_path):
        os.remove(file_path)
```

[8]: # global variables
CORRELATION_DIR = 'correlation_plots'

create directory if does not exist
create_folder(CORRELATION_DIR)

[9]: # function to calculate correlation matrix values
def create_correlation_matrix(data, towns, field):
 towns_cnt = len(towns)
 # init zero matrix with m=n=count of cities
 # set values to -13, just to be sure it is an impossible correlation value
 ret_matrix = np.zeros((towns_cnt, towns_cnt)) - 13

 # iterate through every city combination and calculate the correlation
 # normalize the date for each town
 for i, town1 in enumerate(towns):
 town1_values = np.array(data.loc[data['city'] == town1][field])
 town1_values = (town1_values - np.mean(town1_values)) / (np.
 std(town1_values) * len(town1_values))
 # correlation 1 on diagonal
 ret_matrix[i,i] = 1.0

 # having in mind that ret_matrix[i,j] == ret_matrix[j,i]
 for j, town2 in enumerate(towns[i+1:], i+1):
 town2_values = np.array(data.loc[data['city'] == town2][field])
 town2_values = (town2_values - np.mean(town2_values)) / (np.
 std(town2_values))
 ret_matrix[i,j] = np.correlate(town1_values, town2_values)[0]
 ret_matrix[j,i] = ret_matrix[i,j]

 return ret_matrix

[10]: unique_towns = sorted(list(df_data['city'].unique()))

CORRELATION_COLUMN = 'humidity' # choose which column will be used for analysis
CORRELATION_COLUMN = 'tempC' # choose which column will be used for analysis
CORRELATION_COLUMN = 'windspeedKmph' # choose which column will be used for analysis

truncate df and leave only wanted column

```

df_data = df_data[['date_time', CORRELATION_COLUMN, 'city']]

# modify output folder
CORRELATION_DIR = f'{CORRELATION_DIR}/{CORRELATION_COLUMN}'
create_folder(CORRELATION_DIR)

CORRELATION_DATA_FILENAME = f'data/correlation/
                           {CORRELATION_COLUMN}_correlation_data.npy'

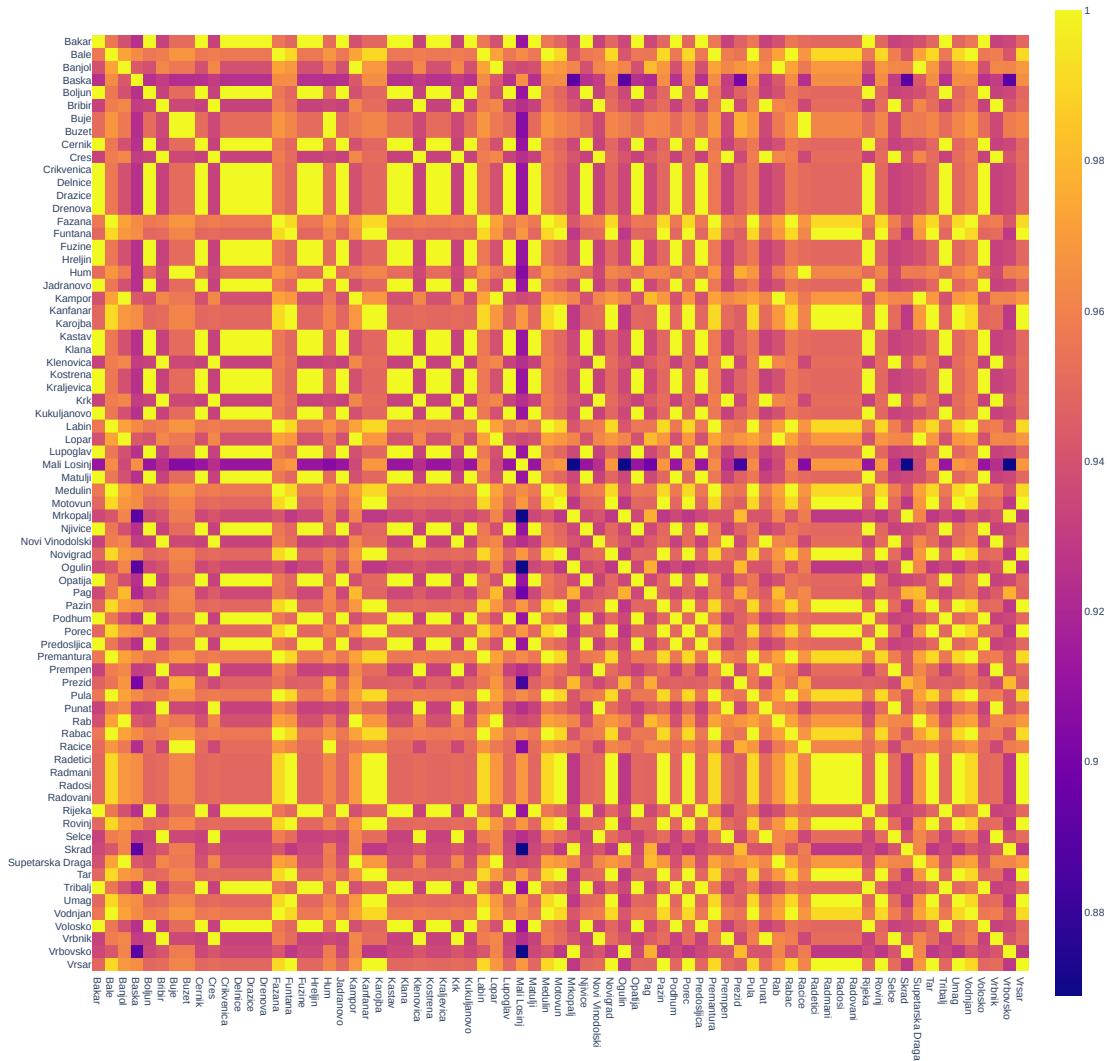
# check if we already have correlation matrix saved
if os.path.exists(CORRELATION_DATA_FILENAME):
    print('Correlation file exists!')
    corr_matrix = np.load(CORRELATION_DATA_FILENAME)
else:
    print('Correlation file does not exist.. Creating one...')
    corr_matrix = create_correlation_matrix(
        data=df_data,
        towns=unique_towns,
        field=CORRELATION_COLUMN
    )
    np.save(CORRELATION_DATA_FILENAME, corr_matrix)

```

Correlation file exists!

```
[11]: # plot correlation matrix
CORRELATION_MATRIX_FILENAME = f'{CORRELATION_DIR}/
                             {CORRELATION_COLUMN}_correlation_matrix.png'
remove_file_if_exists(CORRELATION_MATRIX_FILENAME)
fig = px.imshow(
    corr_matrix,
    x=unique_towns,
    y=unique_towns,
    width=1300,
    height=1300
)
fig.update_layout(
    title_text=f'Correlation Matrix Analysis',
    title_x=0.5
)
fig.write_image(CORRELATION_MATRIX_FILENAME)
fig.show()
```

Correlation Matrix Analysis



```
[12]: # function to plot correlation bar chart
def plot_town_bar_chart(cor, towns, field):
    for i, town in enumerate(towns):
        CORRELATION_IMAGE_FILENAME = f'{CORRELATION_DIR}/
        {field}_{town}_correlation_chart.png'
        remove_file_if_exists(CORRELATION_IMAGE_FILENAME)

        curr_towns = towns.copy()
        curr_towns.remove(town)

        curr_values = cor[i]
        curr_values = np.delete(curr_values, i)
```

```

curr_df = pd.DataFrame({'CITY': curr_towns, 'VALUES': curr_values})

fig = px.bar(
    curr_df,
    x='CITY',
    y='VALUES',
    hover_name='CITY',
    width=1000,
    height=500
)
fig.update_layout(title_text=f'Correlation - {town}', title_x=0.5)
fig.update_xaxes(
    tickangle=90,
    tickmode='linear',
    title=''
)
fig.update_yaxes(
    title='%',
    range=[0.5,1.01]
)
fig.write_image(CORRELATION_IMAGE_FILENAME)

if town == 'Rijeka':
    fig.show()

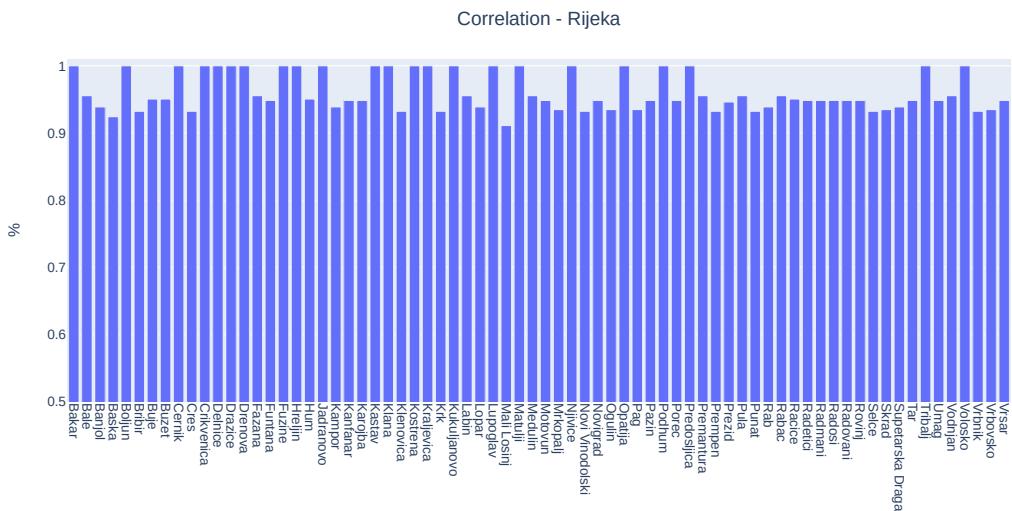
```

[13]: # call function for creating bar charts for each town

```

plot_town_bar_chart(
    cor=corr_matrix,
    towns=unique_towns,
    field=CORRELATION_COLUMN
)

```



1.1 Correlation Map

```
[14]: # plot map with values from SVD_V (towns to concept)
def plot_correlation_map(partitions, data_geo, corr_matrix, map_borders):
    named_colorscales = px.colors.DEFAULT_PLOTLY_COLORS * 10
    CORR_MAP_FILENAME = f'{CORRELATION_DIR}/
˓→{CORRELATION_COLUMN}_correlation_map_{len(partitions)}communities.png'
    remove_file_if_exists(CORR_MAP_FILENAME)

    mapbox_access_token = (open('.mapbox_token").read())
    fig = go.Figure()
    fig.update_layout(
        width=1800,
        height=800,
    )

    # create a list with all dfs to plot cities in scatter plot at the end
    list_data_geo_nodes = []

    # iterate through partitions and draw them on the map
    for i, partition in enumerate(partitions):
        # cast set to list and extract wanted cities from df
        partition = list(partition)
        # print(partition)
        # if there is a single element in the partition, print it
        if len(partition) < 2:
            ind = partition[0]
            print(f'There is a single element partition: {unique_towns[ind]}')
            continue

        data_geo_nodes = data_geo.loc[data_geo.index.isin(partition)]

        # append df to list
        list_data_geo_nodes.append(data_geo_nodes)

        # iterate through elements in partition and plot the pairs
        for j in range(len(partition)-1):
            # extract values
            corr_value = corr_matrix[partition[j], partition[j+1]]
            nodes_index = [partition[j], partition[j+1]]

            # truncate df to just two cities
            data_geo_pair = data_geo_nodes.loc[data_geo_nodes.index.
˓→isin(nodes_index)]
```

```

        city_from, city_to = data_geo_nodes.at[partition[j], "CITY"], data_geo_nodes.at[partition[j+1], "CITY"]

        # calculate scaled width and opacity
        scaled_width = scale_range(
            old_value=corr_value,
            corr_matrix=corr_matrix,
            new_min=0.5,
            new_max=3.5,
        )
        scaled_opacity = scale_range(
            old_value=corr_value,
            corr_matrix=corr_matrix,
            new_min=0.3,
            new_max=1.0,
        )

    # draw lines and group them by partitions using legendgroup
    fig.add_trace(
        go.Scattermapbox(
            mode = "lines",
            lon = data_geo_pair['LNG'],
            lat = data_geo_pair['LAT'],
            name=f'{city_from} - {city_to}: corr:{round(corr_value, 2)}',
            legendgroup=f'Partition {i+1}',
            showlegend=True,
            line=dict(color=named_colorscales[i], width=scaled_width),
            opacity=scaled_opacity
        )
    )

    # plot cities as scatters on the map with different color
    all_data_geo_nodes = pd.concat(list_data_geo_nodes)
    fig.add_trace(
        go.Scattermapbox(
            mode = "markers",
            lon = all_data_geo_nodes['LNG'],
            lat = all_data_geo_nodes['LAT'],
            text=all_data_geo_nodes['CITY'],
            showlegend=False,
            marker=dict(color=named_colorscales[i+1], size=7)
        )
    )

    # setup layout parameters
    fig.update_layout(

```

```

height=700,
margin = {
    'l':15,
    'r':35,
    't':35,
    'b':15,
},
autosize=True,
mapbox = {
    'accesstoken': mapbox_access_token,
    'center': {
        'lon': np.average(map_borders[0:2]),
        'lat': np.average(map_borders[2:4])
    },
    'style': "open-street-map",
    'zoom': 7.5
},
title_text=f'Correlation Between Cities ({len(partitions)} Partitions)',
title_x=0.5
)

fig.write_image(CORR_MAP_FILENAME)
fig.show()

```

```

[15]: from geopy.distance import geodesic as GD

# function to scale up correlation values
def scale_range(old_value, corr_matrix, new_min, new_max):
    old_min = np.min(corr_matrix)
    old_max = np.max(corr_matrix)

    old_range = old_max - old_min
    new_range = new_max - new_min

    if old_value == old_min:
        return new_min

    new_value = (((old_value - old_min) * new_range) / old_range) + new_min
    return new_value

# function to create graph from correlation matrix
def create_graph(corr_matrix, towns_index, data_geo):
    G = nx.Graph()
    distance = np.zeros((len(towns_index), len(towns_index)))
    for i in towns_index:
        town1_data = (data_geo.iloc[i]['LAT'], data_geo.iloc[i]['LNG'])
        for j in towns_index[i+1:]:
            town2_data = (data_geo.iloc[j]['LAT'], data_geo.iloc[j]['LNG'])
            distance[i][j] = GD(town1_data, town2_data).miles
    G.add_weighted_edges_from([(i, j, distance[i][j]) for i in towns_index for j in towns_index[i+1:]])
    return G

```

```

        town2_data = (data_geo.iloc[j]['LAT'], data_geo.iloc[j]['LNG'])
        distance[i][j] = GD(town1_data, town2_data).km
        distance[j][i] = GD(town1_data, town2_data).km
    # new_min = np.min(corr_matrix)
    # new_max = np.max(corr_matrix)
    for i in towns_index:
        for j in towns_index[i+1:]:
            G.add_edge(i, j, weight=corr_matrix[i,j])
            # distance[i][j] = scale_range(distance[i][j], distance, new_min,
            ↪new_max)
            # G.add_edge(i, j, weight=corr_matrix[i,j] / distance[i,j])
            # G.add_edge(i, j, weight=distance[i,j]/corr_matrix[i,j])
            # G.add_edge(i, j, weight=distance[i,j] * corr_matrix[i,j])
            # print(f'{unique_towns[i]}, {unique_towns[j]} --- w:{corr_matrix[i,j] / curr_distance} corr: {corr_matrix[i,j]} dist:{distance[i][j]} / {curr_distance}')
            ↪{corr_matrix[i,j]/curr_distance} corr: {corr_matrix[i,j]} dist:{distance[i][j]} / {curr_distance}')

    return G

```

```

[16]: # import cities with its longitude and latitude
GEO_POSITION_FILENAME = 'data/geo_position.csv'
df_geo_position = pd.read_csv(GEO_POSITION_FILENAME, )
df_geo_position.sort_values(by=['CITY'], inplace=True)
df_geo_position.reset_index(drop=True, inplace=True)

# left right up down
map_borders = (
    np.min(df_geo_position['LNG']),
    np.max(df_geo_position['LNG']),
    np.max(df_geo_position['LAT']),
    np.min(df_geo_position['LAT']),
)

# call function to create graph G
G = create_graph(
    corr_matrix=corr_matrix,
    towns_index=list(df_geo_position.index),
    data_geo=df_geo_position
)

# send G to create n partitions and plot them on data
# G_partitions = nx.algorithms.community.greedy_modularity_communities(G,
#     ↪weight='weight', cutoff=4, best_n=7, resolution=300)
for min_no_of_communities in range(4, 7, 1):
    resoultion_value = 1
    iter = 0
    while True:

```

```

G_partitions = nx.algorithms.community.louvain_communities(
    G=G,
    weight='weight',
    seed=100,
    threshold=1e-07,
    resolution=resoultion_value
)
curr_len = len(G_partitions)
if curr_len >= min_no_of_communities:
    break
resoultion_value += 0.0001
iter += 1

print(f'Min communities: {min_no_of_communities} -- Iter: {iter} --')
print(f'Resoluton value: {resoultion_value}')

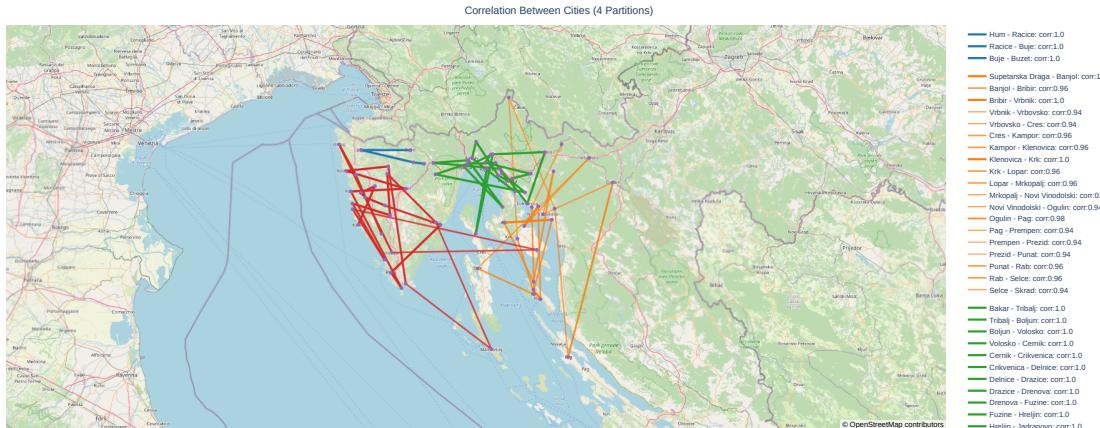
# G_partitions = nx.algorithms.community.girvan_newman(G)
# ret_list = []
# for G_now in next(G_partitions):
#     ret_list.append(list(G_now))
# print(ret_list)

# order them by number of cities in partition (just to have it better drawn)
G_partitions.sort(key=len)

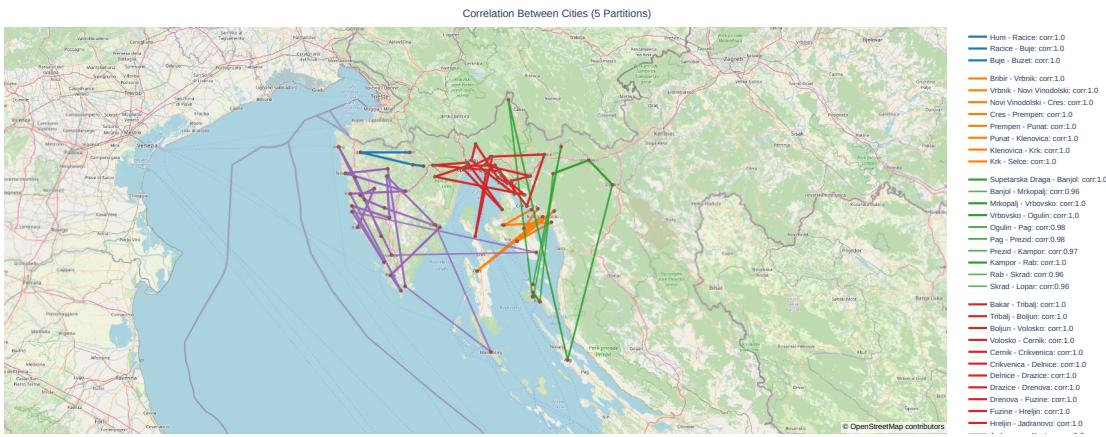
# plot map
plot_correlation_map(
    partitions=G_partitions,
    data_geo=df_geo_position,
    corr_matrix=corr_matrix,
    map_borders=map_borders,
)

```

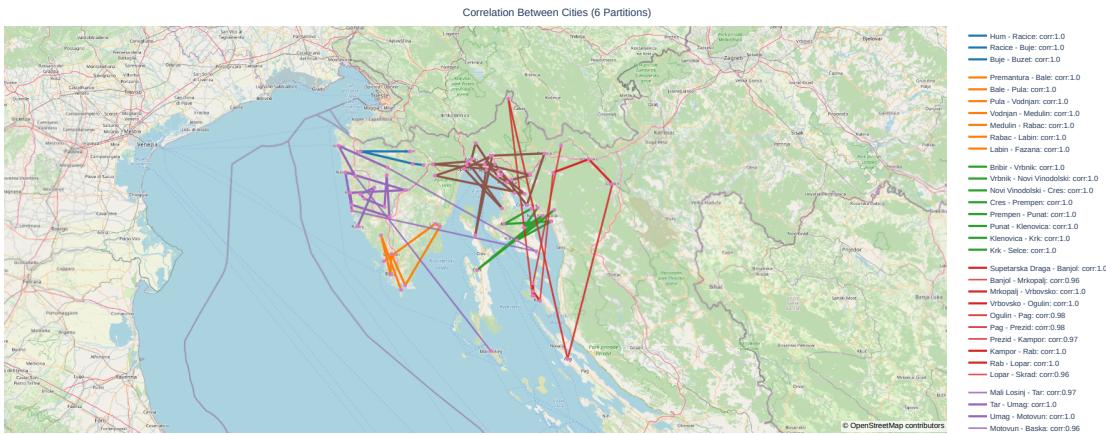
Min communities: 4 -- Iter: 167 -- Resoluton value: 1.0166999999999982



Min communities: 5 -- Iter: 272 -- Resoluton value: 1.0271999999999997



Min communities: 6 -- Iter: 318 -- Resoluton value: 1.0317999999999965



2 Export to HTML

```
[17]: # save notebook before nbconvert
import IPython
```

```
[18]: %%javascript
IPython.notebook.save_notebook()
```

```
<IPython.core.display.Javascript object>
```

```
[19]: # export notebook results to HTML and PDF
jupyter_out_filename = f'{CORRELATION_COLUMN}_correlation'
```

```
!jupyter nbconvert --output-dir 'output' --output {jupyter_out_filename}_
↳--to=HTML correlation.ipynb
!jupyter nbconvert --output-dir 'output' --output {jupyter_out_filename}_
↳--to=pdf correlation.ipynb

jupyter_out_filename_no_code = f'{CORRELATION_COLUMN}_correlation_no_code'
!jupyter nbconvert --output-dir 'output' --output_
↳{jupyter_out_filename_no_code} --no-input --to=HTML correlation.ipynb
!jupyter nbconvert --output-dir 'output' --output_
↳{jupyter_out_filename_no_code} --no-input --to=pdf correlation.ipynb
```

```
[NbConvertApp] Converting notebook correlation.ipynb to HTML
[NbConvertApp] Writing 4539489 bytes to output/tempC_correlation.html
```