correlation

August 14, 2022

[18]:
```
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
```

[19]:
```
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

[20]:
```
# 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 concatanate 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)

    # fix json
    df['weatherIconUrl'] = df['weatherIconUrl'].str.replace("\[{'value': '", "")
    df['weatherIconUrl'] = df['weatherIconUrl'].str.replace("'}]", "")

    df['weatherDesc'] = df['weatherDesc'].str.replace("\[{'value': '", "")
```

```
    df['weatherDesc'] = df['weatherDesc'].str.replace("'}]", "")

    # 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')

# import data
df_full_data = pd.read_csv(DATA_FILE_NAME, compression='gzip')
```

```
Data has already been processed
```

1 Correlation

[21]:
```
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)

create_folder('data/')
create_folder('data/correlation')
```

[22]:
```
# keep only needed data
def truncate_df(df, columns_to_keep, years=None, months=None, hours=None,␣
 ↪additional_conditions=None):
    # modify timestamp column
    df['date_time'] = pd.to_datetime(df['date_time'],format='%Y-%m-%d %H:%M:%S')

    # build prefix str
    prefix_str = 'columns-' + '_'.join(columns_to_keep)

    if additional_conditions:
        for additional_condition in additional_conditions:
            column, sign, value = additional_condition
```

```
            if sign == '<':
                df = df.loc[df[column] < value]
            elif sign == '>':
                df = df.loc[df[column] > value]
            elif sign == '=':
                df = df.loc[df[column] == value]
            elif sign == 'in':
                df = df.loc[df[column].isin(value)]
            prefix_str += f'-condition-{column}-{sign}-' + '_'.join(value)

    columns_to_keep = ['date_time'] + columns_to_keep + ['city']
    df = df[columns_to_keep]

    if years:
        df = df[df['date_time'].dt.year.isin(years)]
        prefix_str += '-years-' + '_'.join(map(str, years))

    if months:
        df = df[df['date_time'].dt.month.isin(months)]
        prefix_str += '-months-' + '_'.join(map(str, months))

    if hours:
        df = df[df['date_time'].dt.hour.isin(hours)]
        prefix_str += '-hours-' + '_'.join(map(str, hours))

    prefix_str = prefix_str.replace(' ', '')
    return df, prefix_str
```

[23]:
```
# function to calculate correlation matrix values
def create_correlation_matrix(data, towns, field):
    if len(field) == 1:
        field = field[0]

    towns_cnt = len(towns)
    # init zero matrix with m=n=count of cities
    # set values to -13, just to be sure it is an imposible correlation value
    ret_matrix = np.zeros((towns_cnt, towns_cnt)) - 13

    # iterate through every city combination and calculate the correlation
    for i, town1 in enumerate(towns):
        town1_values = np.array(data.loc[data['city'] == town1][field])

        # 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):
```

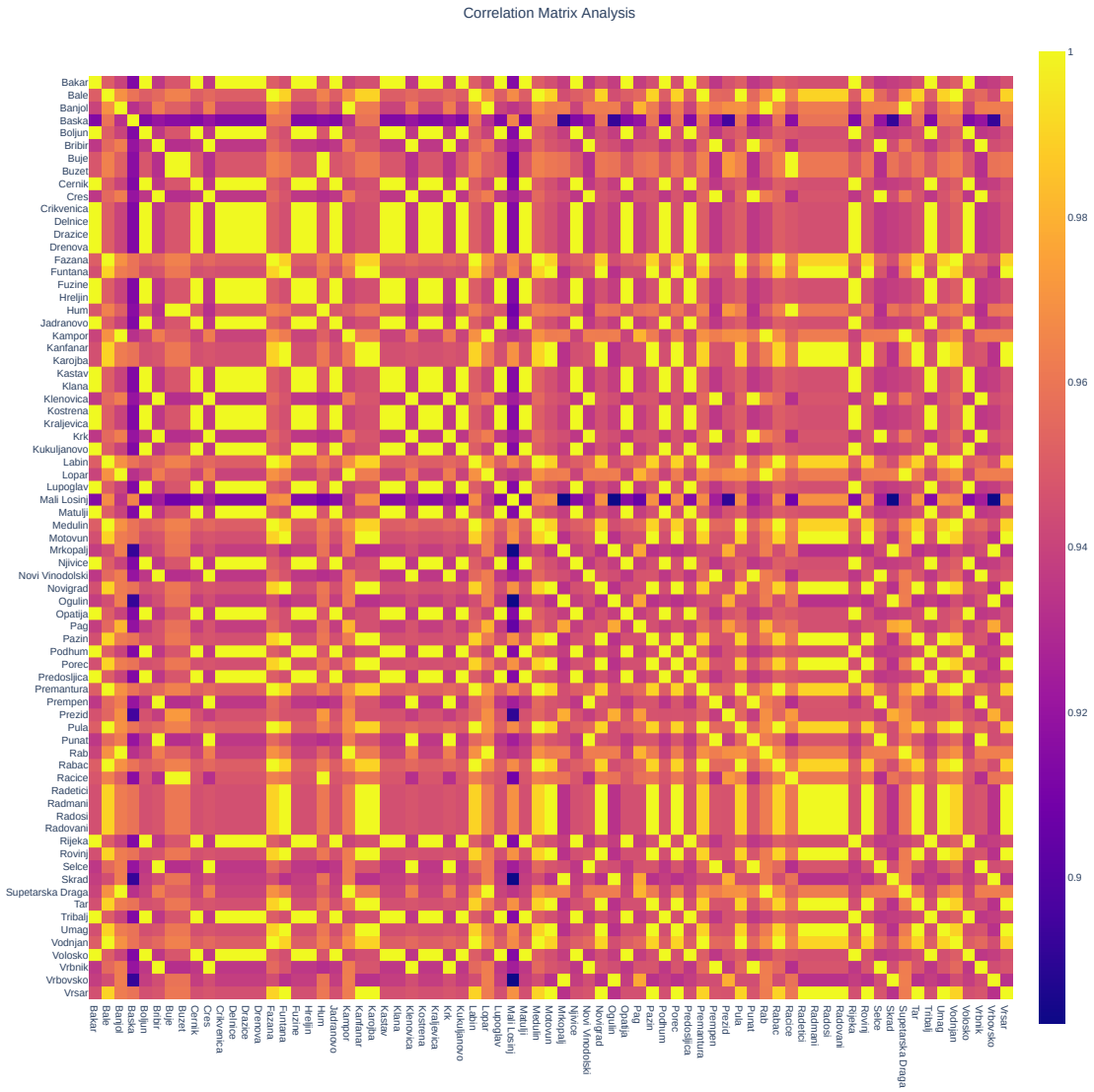
```
            town2_values = np.array(data.loc[data['city'] == town2][field])

            ret_matrix[i,j] = np.corrcoef(town1_values, town2_values)[0,1]
            ret_matrix[j,i] = ret_matrix[i,j]

    return ret_matrix
```

[24]:
```
# global variables
CORRELATION_DIR = 'correlation_plots'
# select which columns to keep
COLUMNS_TO_KEEP = ['FeelsLikeC']
# COLUMNS_TO_KEEP = ['humidity']

# create directory if does not exist
create_folder(CORRELATION_DIR)

# list of all towns
unique_towns = sorted(list(df_full_data['city'].unique()))

# create dataframe and string needed to create out files
df_data, PREFIX_STR = truncate_df(
    df=df_full_data,
    columns_to_keep=COLUMNS_TO_KEEP,
    # months=[6,7,8,9]
    # additional_conditions=[('weatherDesc', 'in', ['Clear', 'Sunny', ], )]
    # additional_conditions=[('weatherDesc', 'in', ['Partly cloudy', 'Cloudy',␣
 ↪], )]
)

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

CORRELATION_DATA_FILENAME = f'data/correlation/{PREFIX_STR}_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=COLUMNS_TO_KEEP
    )
```

```
    np.save(CORRELATION_DATA_FILENAME, corr_matrix)
```

```
Correlation file exists!
```

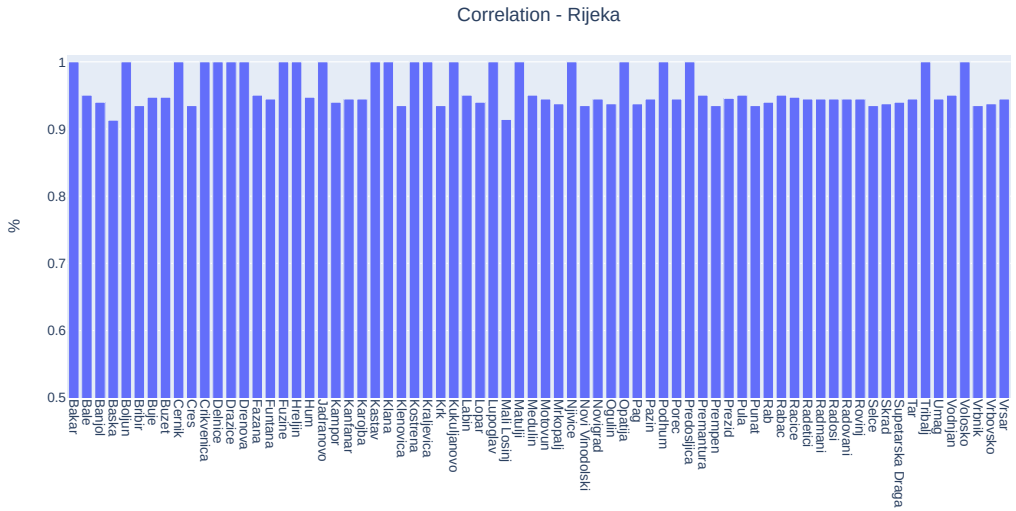
[25]:
```
# plot correlation matrix
CORRELATION_MATRIX_FILENAME = f'{CORRELATION_DIR}/
 ↪{PREFIX_STR}_correlation_matrix.png'
remove_file_if_exists(CORRELATION_MATRIX_FILENAME)
fig = px.imshow(
    corr_matrix,
    x=unique_towns,
    y=unique_towns,
    width=1300,
    height=1300
)
fig.update_layout(
    title_text=f'Correlation Matrix Analysis',
    title_x=0.5
)
fig.write_image(CORRELATION_MATRIX_FILENAME)
fig.show()
```



[26]:
```
# 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()
```

[27]:
```
# call function for creating bar charts for each town
plot_town_bar_chart(
    cor=corr_matrix,
    towns=unique_towns,
    field=PREFIX_STR
)
```



1.1 Correlation Map

[28]:
```
# 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}/
 ↪{PREFIX_STR}_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 = []

    # itterate 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)

        # if there is a single element in the partition, print it
        data_geo_nodes = data_geo.loc[data_geo.index.isin(partition)]
        if len(partition) < 2:
            ind = partition[0]
            print(f'There is a single element partition: {unique_towns[ind]}')
            fig.add_trace(
                go.Scattermapbox(
                    mode = "markers",
                    lon=data_geo_nodes['LNG'],
                    lat=data_geo_nodes['LAT'],
                    name=data_geo_nodes['CITY'].values[0],
                    legendgroup=f'Partition {i+1}',
                    showlegend=True,
                    marker=dict(color=named_colorscales[i], size=14)
                )
            )
            continue

        # append df to list
        list_data_geo_nodes.append(data_geo_nodes)
```

```
        # itterate 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(
        width=1485,
        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()
```

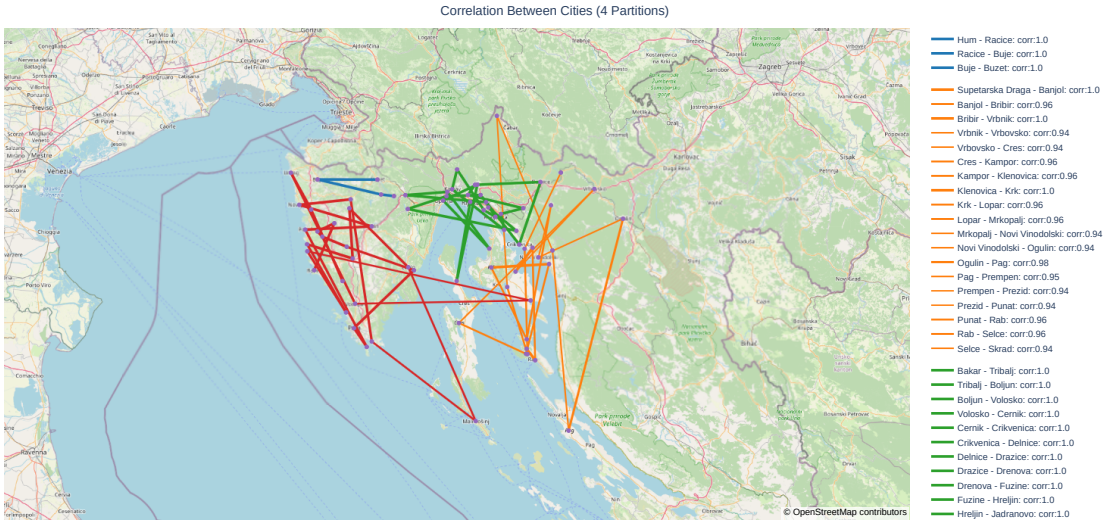
[29]:
```
from geopy.distance import geodesic as GD

# 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()
    for i in towns_index:
        for j in towns_index[i+1:]:
            G.add_edge(i, j, weight=corr_matrix[i,j])

    return G
```

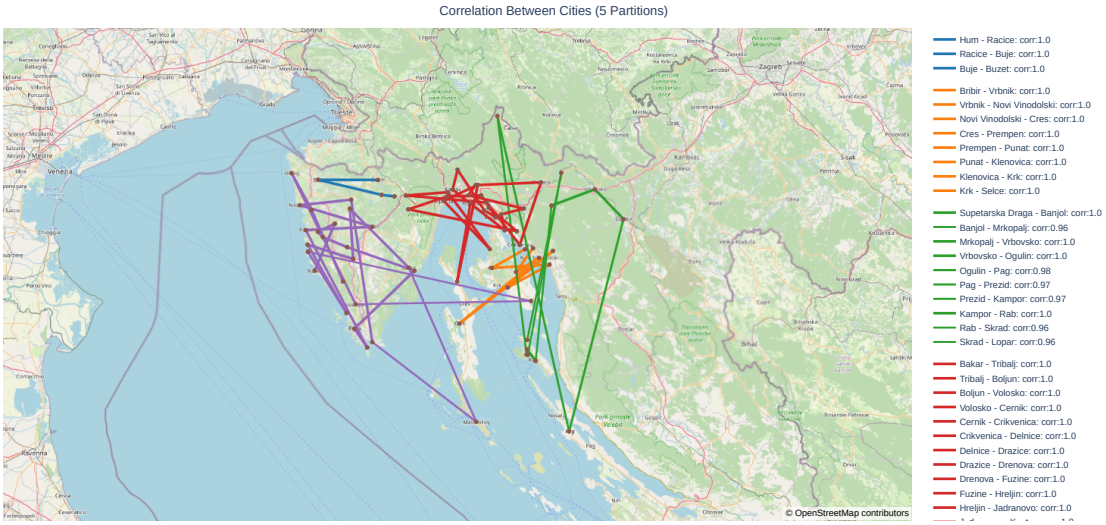
[30]:
```
# import cities with its logitude and latitude
GEO_POSITION_FILENAME = 'data/geo_position.csv'
df_geo_position = pd.read_csv(GEO_POSITION_FILENAME, )
df_geo_position.sort_values(by=['CITY'], inplace=True)
df_geo_position.reset_index(drop=True, inplace=True)

# 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
resoultion_value = 1
min_no_of_communities = 4
while True:
    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 += 0.001

    # break if there are 3 more communities than minimum set
    if len(G_partitions) >= min_no_of_communities + 3 or min_no_of_communities␣
↪== 8:
        break

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

    print(f'Min comumunities: {min_no_of_communities} -- Resoluton value:␣
↪{resoultion_value}')

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

    # increase min no of communities
    min_no_of_communities += 1
```

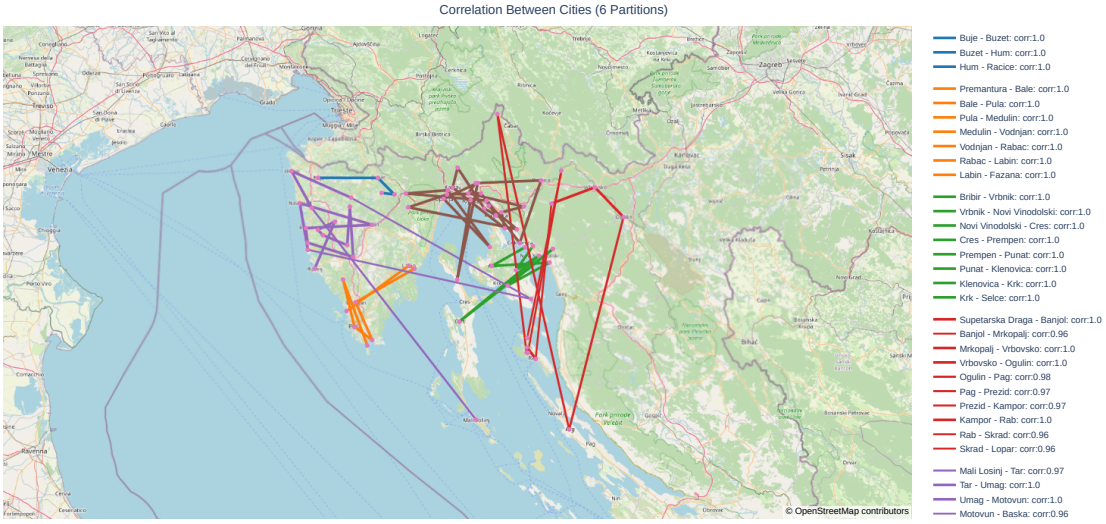
```
Min comumunities: 4 -- Resoluton value: 1.017999999999998
```



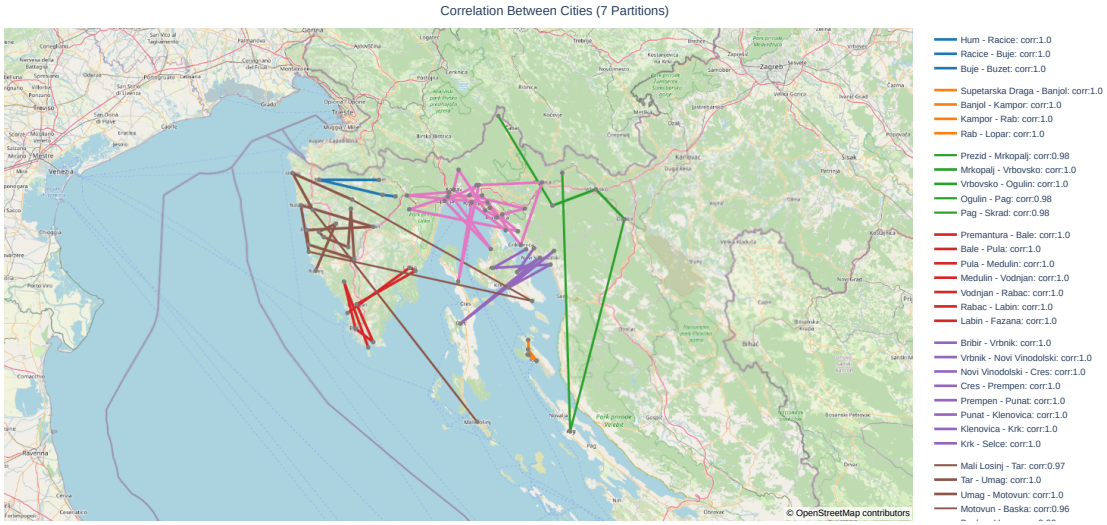

```
Min comumunities: 5 -- Resoluton value: 1.0259999999999971
```



Min comumunities: 6 -- Resoluton value: 1.0349999999999961



Min comumunities: 7 -- Resoluton value: 1.0379999999999958



2 Export to HTML

[31]:
```
# save notebook before nbconvert
import IPython
```

[32]:
```
%%javascript
IPython.notebook.save_notebook()
```

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

[33]:
```
# export notebook results to HTML and PDF
jupyter_out_filename = f'{PREFIX_STR}_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'{PREFIX_STR}_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 4548401 bytes to output/columns-
FeelsLikeC_correlation.html
```