correlation

April 3, 2022

[ ]:
```python
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
```

[ ]:
```python
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

[ ]:
```python
# setup global variables - data file directory and name
DATA_FILES_DIR = 'data'
DATA_FILE_NAME = '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)

    # 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')
```

```python
# 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

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1066968 entries, 0 to 1066967
Data columns (total 37 columns):
 #   Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   date_time          1066968 non-null  object
 1   sunrise            1066968 non-null  object
 2   sunset             1066968 non-null  object
 3   moonrise           1066968 non-null  object
 4   moonset            1066968 non-null  object
 5   moon_phase         1066968 non-null  object
 6   moon_illumination  1066968 non-null  int64
 7   time               1066968 non-null  int64
 8   tempC              1066968 non-null  int64
 9   tempF              1066968 non-null  int64
 10  windspeedMiles     1066968 non-null  int64
 11  windspeedKmph      1066968 non-null  int64
 12  winddirDegree      1066968 non-null  int64
 13  winddir16Point     1066968 non-null  object
 14  weatherCode        1066968 non-null  int64
 15  weatherIconUrl     1066968 non-null  object
 16  weatherDesc        1066968 non-null  object
 17  precipMM           1066968 non-null  float64
 18  precipInches       1066968 non-null  float64
 19  humidity           1066968 non-null  int64
 20  visibility         1066968 non-null  int64
 21  visibilityMiles    1066968 non-null  int64
 22  pressure           1066968 non-null  int64
 23  pressureInches     1066968 non-null  int64
 24  cloudcover         1066968 non-null  int64
 25  HeatIndexC         1066968 non-null  int64
 26  HeatIndexF         1066968 non-null  int64
```

```
 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
```

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

[ ]:
```
             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

[ ]:
```python
# global variables
CORRELATION_DIR = 'correlation_plots'

# recreate directory if does not exist
if not os.path.exists(CORRELATION_DIR):
    print(f'Creating folder {CORRELATION_DIR}')
    os.mkdir(CORRELATION_DIR)

# 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)
```

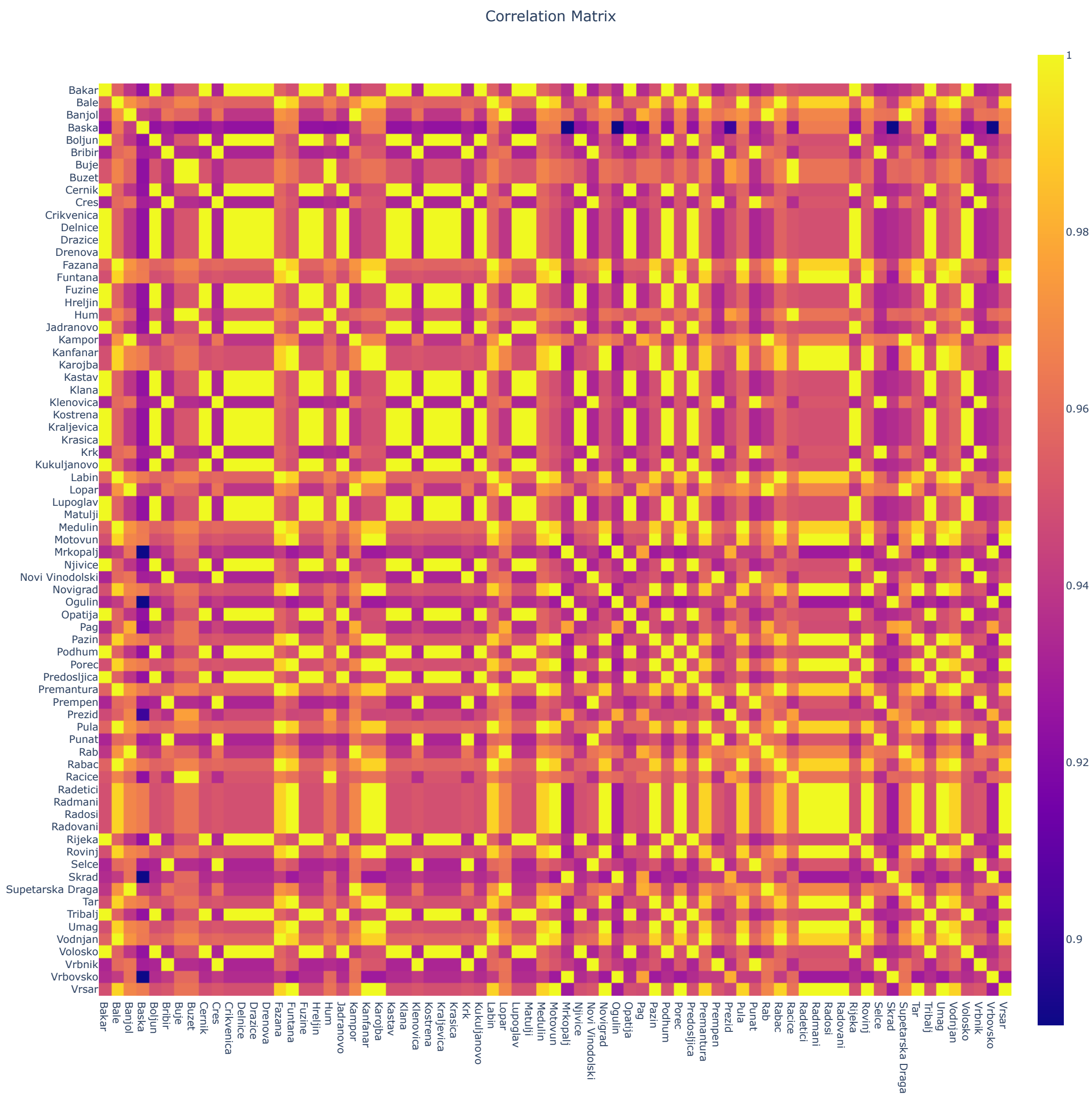
[ ]:
```python
# function to calculate correlation matrix values
def create_correlation_matrix(data, towns, field):
    towns_cnt = len(towns)
    # init zero matrix with m=n=count of cities
    # set values to -13, just to be sure it is an imposible correlation value
    ret_matrix = np.zeros((towns_cnt,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

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

CORRELATION_COLUMN = 'tempC'  # choose which column will be used for analysis
CORRELATION_DATA_FILENAME = f'{CORRELATION_COLUMN}_correlation_data.npy'

# check if we already have correlation matrix saved
```

```python
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!
```

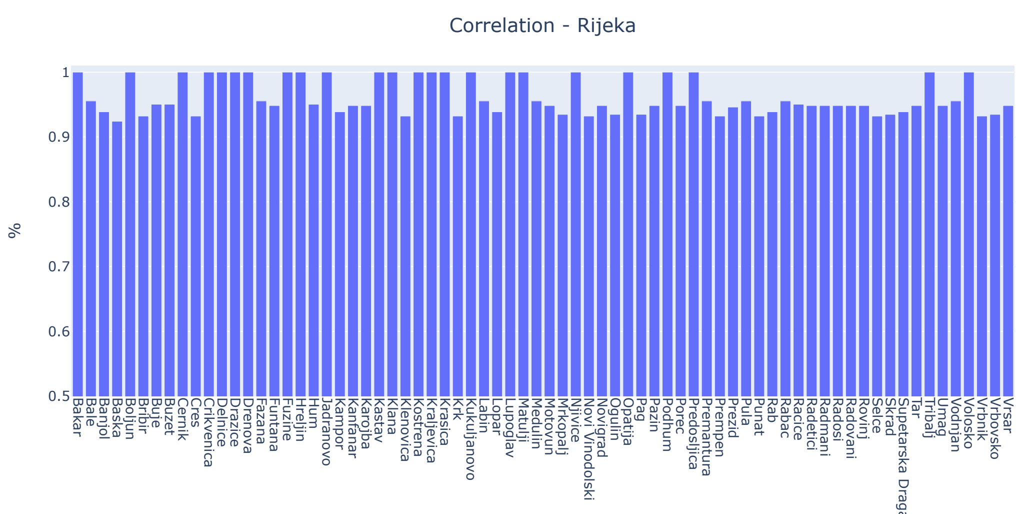
[ ]:
```python
# plot correlation matrix
CORRELATION_MATRIX_FILENAME = f'{CORRELATION_DIR}/
 ↪{CORRELATION_COLUMN}_correlation_matrix.png'
remove_file_if_exists(CORRELATION_MATRIX_FILENAME)
fig = px.imshow(
    corr_matrix,
    x=unique_towns,
    y=unique_towns,
    width=1300,
    height=1300
)
fig.update_layout(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




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

[ ]:
```
# 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

[ ]:
```
#  function to create graph from correlation matrix
def create_graph(corr_matrix, towns_index):
    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

# 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

# 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
    CORR_MAP_FILENAME = f'{CORRELATION_DIR}/
↪{CORRELATION_COLUMN}_correlation_map.png'
    remove_file_if_exists(CORR_MAP_FILENAME)

    mapbox_access_token = (open(".mapbox_token").read())
    fig = go.Figure()

    # 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):
        # if there is a single element in the partition, print it
        if len(partition) < 2:
            print(f'There is a single element partition: {partition}')
            continue
```

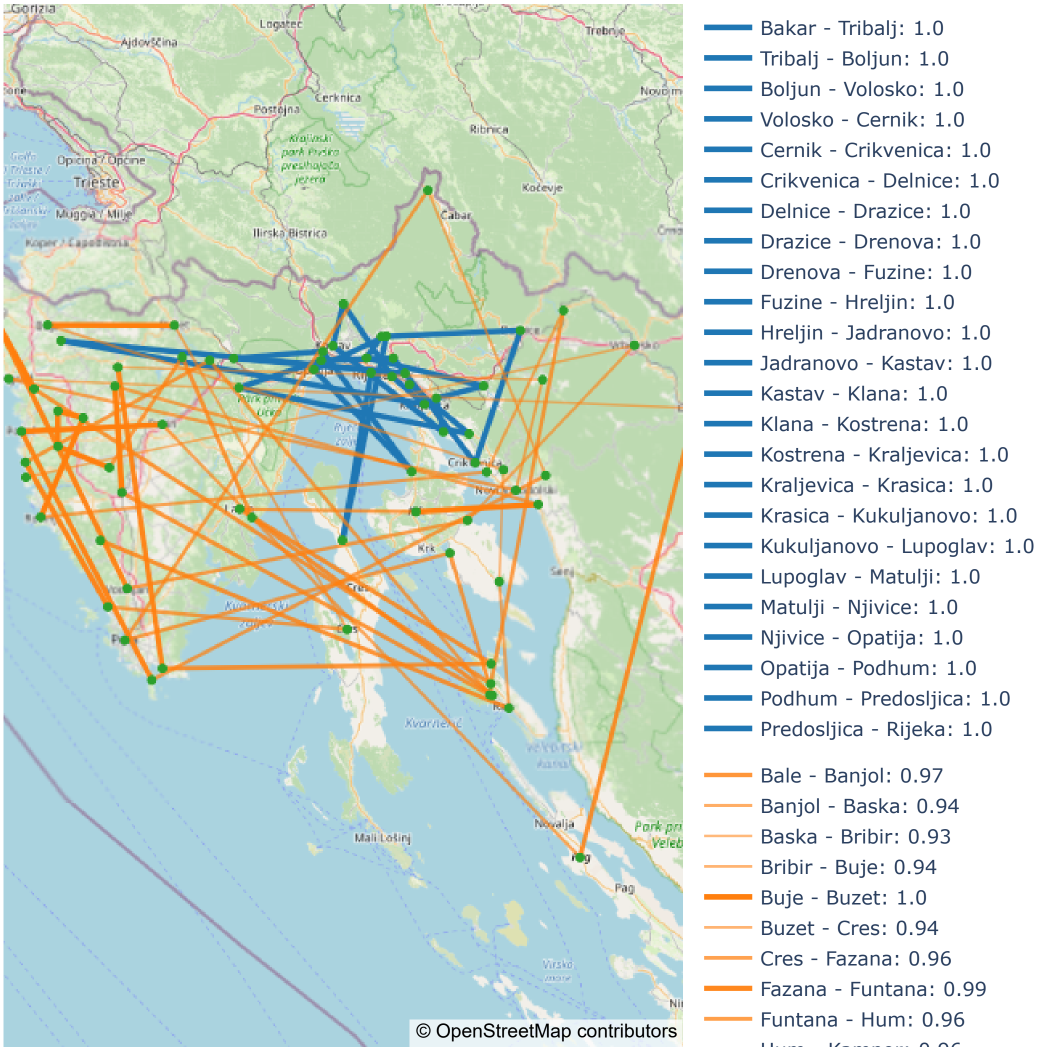
```python
    # cast set to list and extract wanted cities from df
    partition = list(partition)
    data_geo_nodes = data_geo.loc[data_geo.index.isin(partition)]

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

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

        # 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}: {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()
```

[ ]:
```
# import cities with its logitude and latitude
GEO_POSITION_FILENAME = 'geo_position.csv'
df_geo_position = pd.read_csv(GEO_POSITION_FILENAME)

# 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)
)

# send G to create n partitions and plot them on data
G_partitions = nx.algorithms.community.louvain_communities(G)
# 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
)
```

Correlation Between Cities (2 Partitions)



2 Export to HTML

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

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

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

[ ]:
```
# export notebook results to HTML
!jupyter nbconvert --to=HTML correlation.ipynb
```

```
[NbConvertApp] Converting notebook correlation.ipynb to HTML
[NbConvertApp] Writing 4435433 bytes to correlation.html
```