

svd

August 17, 2022

```
[26]: import os

import numpy as np
import pandas as pd

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

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

[28]: 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/svd')

[29]: # keep only needed data
def truncate_df(df, columns_to_keep, years=None, months=None, hours=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)
    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

```

0.1 Prepare Data for SVD Analysis

```
[30]: # function to create csv file used for analysis
def create_svd_df(df, column_value, outstring):
    OUTPUT_FILENAME = f'data/svd/data_svd_{outstring}.csv'

    unique_towns = sorted(list(df['city'].unique()))

    new_index = pd.to_datetime(df['date_time'], format='%Y-%m-%d %H:%M:%S').
    ↪unique()
    data = np.array(df[column_value])
    data = data.reshape(len(new_index), len(unique_towns))

    ret_df = pd.DataFrame(
        data=data,
        columns=unique_towns,
        index=new_index,
        dtype=float
    )
    ret_df.to_csv(OUTPUT_FILENAME)
```

```
[31]: # svd global variables and creating dirs
SVD_FOLDER_PATH = 'svd_plots'
create_folder(SVD_FOLDER_PATH)

FULL_DATA_FILENAME = 'data/data.csv.gz'

COLUMNS_TO_KEEP = ['tempC']
df_data = pd.read_csv(FULL_DATA_FILENAME, compression='gzip')

df_data, PREFIX_STR = truncate_df(
    df=df_data.copy(),
    columns_to_keep=COLUMNS_TO_KEEP,
```

```

    months=[11,12,1,2],
)

SVD_FOLDER_PATH = f'{SVD_FOLDER_PATH}/{PREFIX_STR}'
create_folder(SVD_FOLDER_PATH)

# create reconstruction dir
# SVD_RECONSTRUCTION_FOLDER_PATH = f'{SVD_FOLDER_PATH}/reconstruction'
# create_folder(SVD_RECONSTRUCTION_FOLDER_PATH)

```

Creating folder svd_plots/columns-tempC-months-11_12_1_2

```
[32]: GEOLOCATION_FILENAME = 'data/geo_position.csv'
df_geolocation = pd.read_csv(GEOLOCATION_FILENAME).sort_values(by='CITY')
unique_towns = sorted(list(df_geolocation['CITY'].unique())) # get unique ↴
names of towns ordered by name

SVD_DATA_FILENAME = f'data/svd/data_svd_{PREFIX_STR}.csv'

if not os.path.exists(SVD_DATA_FILENAME):
    print(f'Creating svd file for {PREFIX_STR}')
    create_svd_df(df=df_data, column_value=COLUMNS_TO_KEEP, ↴
outstring=PREFIX_STR)
```

Creating svd file for columns-tempC-months-11_12_1_2

0.2 SVD

```
[33]: print(f'Loading svd file for {PREFIX_STR}')
df_svd = pd.read_csv(SVD_DATA_FILENAME, index_col=0)
df_svd.index = pd.to_datetime(df_svd.index)

# df_svd = trunc_df(df_svd)
svd_A = np.array(df_svd)

# build matrix U, S, V
svd_U, svd_S, svd_V = np.linalg.svd(svd_A, full_matrices=False)
```

Loading svd file for columns-tempC-months-11_12_1_2

0.2.1 Precision of SVD Reconstruction

```
[34]: # function to calculate and plot precision of svd reconstruction
def svd_precision(svd_S):
    SVD_PRECISION_FILENAME = f'{SVD_FOLDER_PATH}/{PREFIX_STR}_svd_precision.png'
    remove_file_if_exists(SVD_PRECISION_FILENAME)

    fig = go.Figure(
```

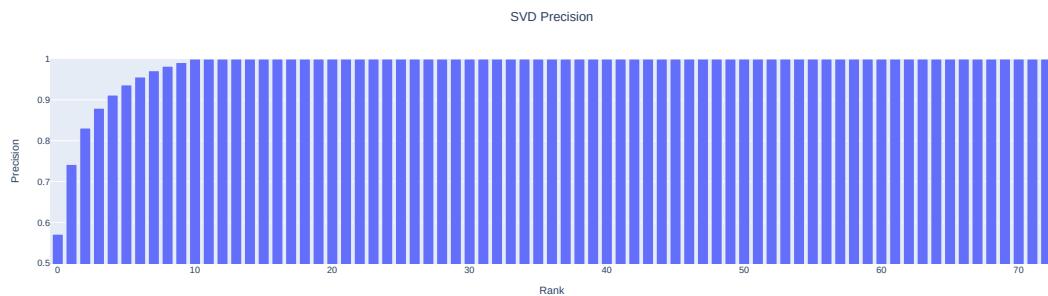
```

        data=[go.Bar(
            x=np.arange(np.size(svd_S)),
            y=np.cumsum(svd_S / np.sum(svd_S))
        )
    ]
)

fig.update_layout(
    title_text='SVD Precision',
    title_x=0.5,
    xaxis_title='Rank',
    yaxis_title='Precision',
    width=1485,
    height=450,
)
fig.update_yaxes(range=[0.5, 1])
fig.write_image(SVD_PRECISION_FILENAME)
fig.show()

```

[35]: # Calculating svd reconstruction precision
`svd_precision(svd_S)`



0.2.2 Full Reconstruction & Lower Rank Reconstruction

[36]: # full reconstruction - matrix svd_Ar
`svd_Ar = np.dot(svd_U * svd_S, svd_V)
print(f'Diff: {np.mean(np.abs(svd_A - svd_Ar))}')`

lower rank reconstruction - matrix svd_Ar
`k = 5
svd_Ar = np.dot(svd_U[:, :k] * svd_S[:k], svd_V[:k, :])`

`print(f'Diff reconstruction: {np.mean(np.abs(svd_A - svd_Ar))}')`

Diff: 1.7170415483460063e-14

```
Diff reconstruction: 0.26469466692042387
```

0.2.3 Average SVD Error

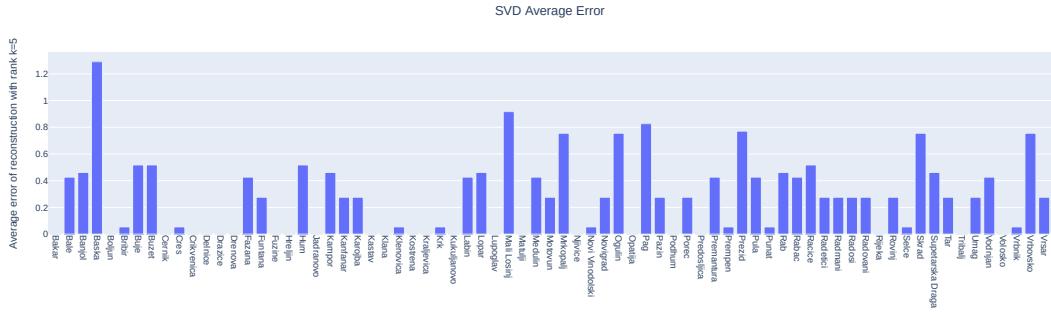
```
[37]: # function to calculate and plot average error of svd for k=n
def svd_average_error(svd_A, svd_Ar, k, unique_towns):
    SVD_AVG_ERR_FILENAME = f'{SVD_FOLDER_PATH}/{PREFIX_STR}_svd_avg_err.png'
    remove_file_if_exists(SVD_AVG_ERR_FILENAME)

    svd_err = np.average(np.abs(svd_A - svd_Ar), axis=0)
    asix_range = np.arange(0, len(unique_towns))

    fig = go.Figure(
        data=[
            go.Bar(
                x=asix_range,
                y=svd_err
            )
        ]
    )

    fig.update_layout(
        title_text='SVD Average Error', title_x=0.5,
        yaxis_title=f'Average error of reconstruction with rank k={k}',
        xaxis=dict(tickmode='array',
                   tickvals=asix_range,
                   ticktext=unique_towns
        ),
        width=1485,
        height=450,
    )
    fig.update_xaxes(tickangle=90)
    fig.write_image(SVD_AVG_ERR_FILENAME)
    fig.show()
```

```
[38]: # Calculating average error
svd_average_error(
    svd_A=svd_A,
    svd_Ar=svd_Ar,
    k=k,
    unique_towns=unique_towns
)
```



0.2.4 Dates to Concept - SVD_U

```
[39]: # function to plot dates to concept for k=n
def svd_dates_to_concept(k, index, svd_U, trim=False):
    SVD_DTC_FILENAME = f'{SVD_FOLDER_PATH}/{PREFIX_STR}_svd_dates_to_concept.
    ↪png'
    remove_file_if_exists(SVD_DTC_FILENAME)

    fig = go.Figure()
    # if we have limited date range
    if trim == True:
        x_tmp = np.arange(0, len(index))
        for i in range(k):
            fig.add_trace(go.Scatter(
                x=x_tmp,
                y=svd_U[:, i], name=f'k={i}')
        )
        fig.update_layout(
            xaxis=dict(
                showticklabels=False,
            ),
            title=dict(
                text='Dates to Concept - Ploted as Continous Function',
                x=0.5,
            )
        )
    # if we have full date range
    else:
        for i in range(k):
            fig.add_trace(go.Scatter(
                x=index,
                y=svd_U[:, i], name=f'k={i}')
        )
    fig.update_layout(
```

```

        title=dict(
            text='Dates to Concept',
            x=0.5,
        )
    )

fig.write_image(SVD_DTC_FILENAME)

fig.update_layout(
    xaxis=dict(
        rangeslider=dict(visible=True),
    ),
    width=1485,
    height=450,
)
fig.show()

```

[40]: # dates to concept

```

svd_dates_to_concept(
    k=k,
    index=df_svd.index,
    svd_U=svd_U,
)

```



[41]: if 'months-' in PREFIX_STR or 'years-' in PREFIX_STR:

```

svd_dates_to_concept(
    k=k,
    index=df_svd.index,
    svd_U=svd_U,
    trim=True
)

```



0.2.5 Towns to Concept - SVD_V

```
[42]: # function to plot towns to concept for k=n
def svd_towns_to_concept(k, svd_V, unique_towns, type='bar'):
    SVD_TTC_FILENAME = f'{SVD_FOLDER_PATH}/{PREFIX_STR}_svd_towns_to_concept.
    ↪png'
    remove_file_if_exists(SVD_TTC_FILENAME)

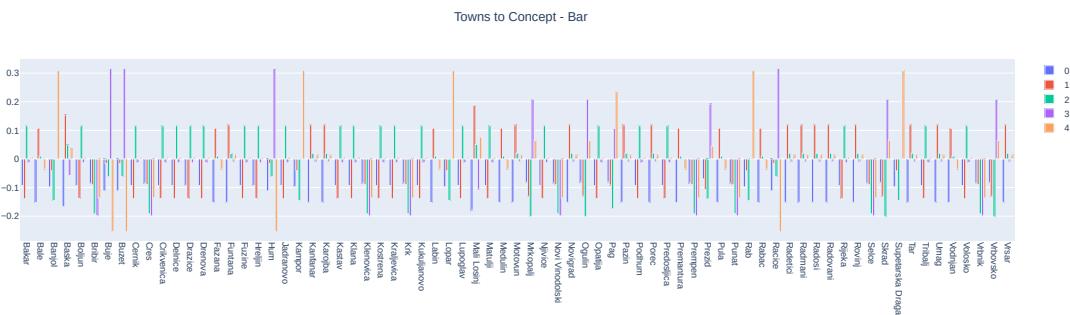
    asix_range = np.arange(0, len(unique_towns))
    all_plots = []
    for i in range(k):
        if type == 'bar':
            all_plots.append(go.Bar(x=asix_range, y=svd_V[i, :], name=f'{i}'))
        elif type == 'lines':
            all_plots.append(go.Scatter(mode='lines', x=asix_range, y=svd_V[i, :],
    ↪], name=f'{i}'))

    fig = go.Figure(data=all_plots)

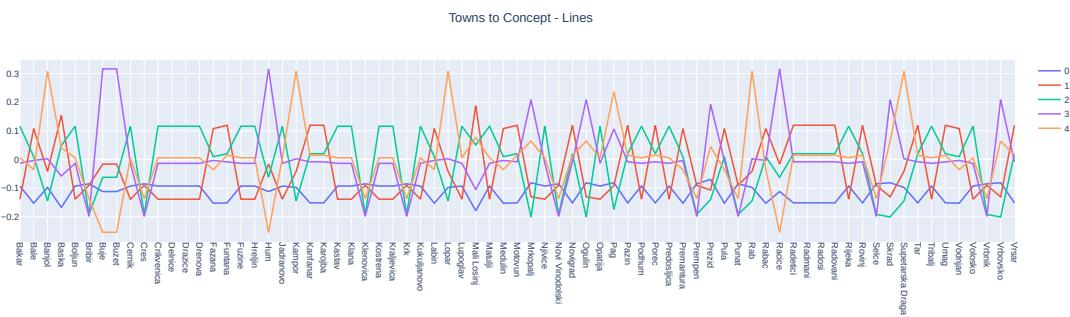
    fig.update_layout(
        title_text=f'Towns to Concept - {type.capitalize()}',
        title_x=0.5,
        xaxis=dict(
            tickmode='array',
            tickvals=asix_range,
            ticktext=unique_towns
        ),
        width=1485,
        height=450,
    )

    fig.update_xaxes(tickangle=90)
    fig.write_image(SVD_TTC_FILENAME)
    fig.show()
```

```
[43]: # towns to concept
        svd_towns_to_concept(
            k=k,
            svd_V=svd_V,
            unique_towns=unique_towns
        )
```



```
[44]: # towns to concept
        svd_towns_to_concept(
            k=k,
            svd_V=svd_V,
            unique_towns=unique_towns,
            type='lines'
        )
```



0.2.6 Singular Vector - SVD_S

```
[45]: ';' .join(map(str, svd_S))
```

[45]: '4403.428259171285; 1317.1458083507682; 687.3012277028485; 374.21577789497553;
248.11449701802832; 191.58990962630992; 149.7830421513899; 118.21206477793639;
85.12620685213918; 69.38056856521939; 65.96496331085869; 3.965747360498728e-13;

0.2.7 SVD Maps

```
[46]: # plot map with values from SVD_V (towns to concept)
def plot_svd_map(vector, k, data_geo):
    SVD_MAP_FILENAME = f'{SVD_FOLDER_PATH}/{PREFIX_STR}_svd_map_k{k}.png'
    remove_file_if_exists(SVD_MAP_FILENAME)

    data_geo['VALUES'] = vector
    px.set_mapbox_access_token(open(".mapbox_token").read())

    fig = px.scatter_mapbox(
        data_geo,
        size = [2] * len(data_geo.index),
        lat="LAT",
        lon="LNG",
        color="VALUES",
        hover_name="CITY",
        color_continuous_scale=px.colors.cyclical.Phase,
    )

    fig.update_layout(
        width=1485,
        height = 700,
        margin = {
            'l':5,
```

```

        'r':5,
        't':5,
        'b':5,
    },
    autosize=True,
    mapbox = {
        'style': "open-street-map",
        'zoom': 7.5
    }
)
fig.write_image(SVD_MAP_FILENAME)
fig.show()

```

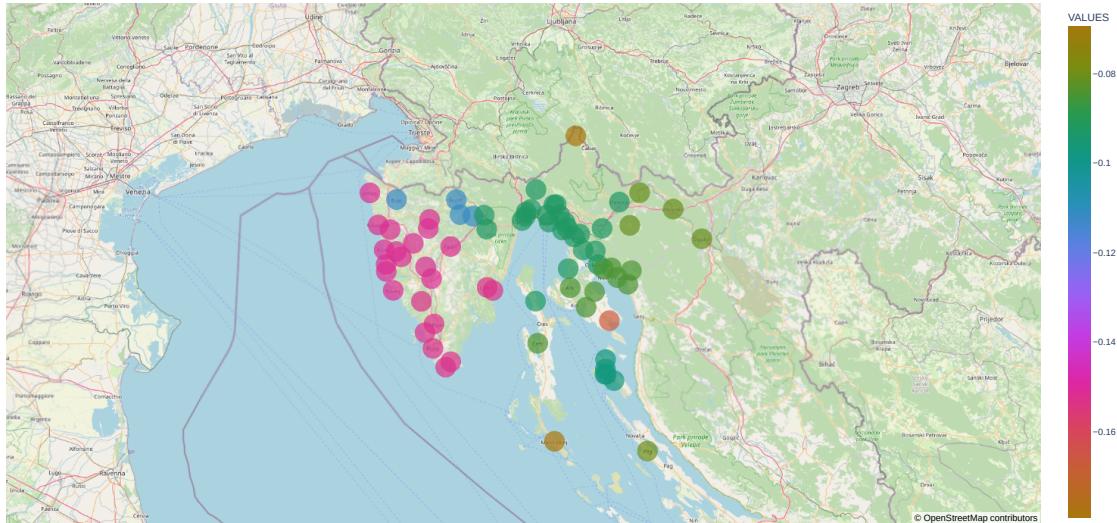
[47]: # plot maps

```

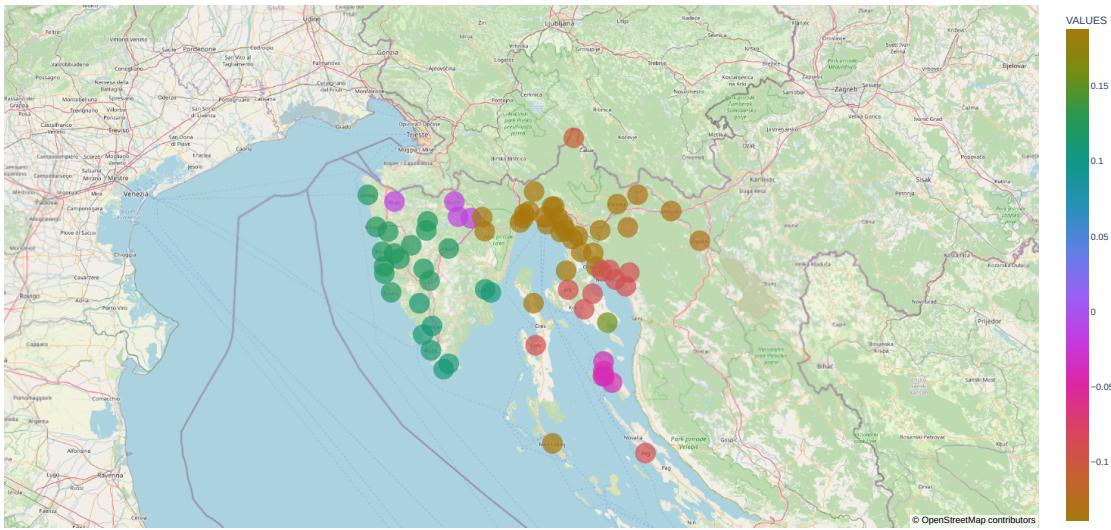
for i in range(k):
    print(f'Plotting map for k = {i}')
    plot_svd_map(
        vector=svd_V[i, :],
        k=i,
        data_geo=df_geolocation.copy()
    )

```

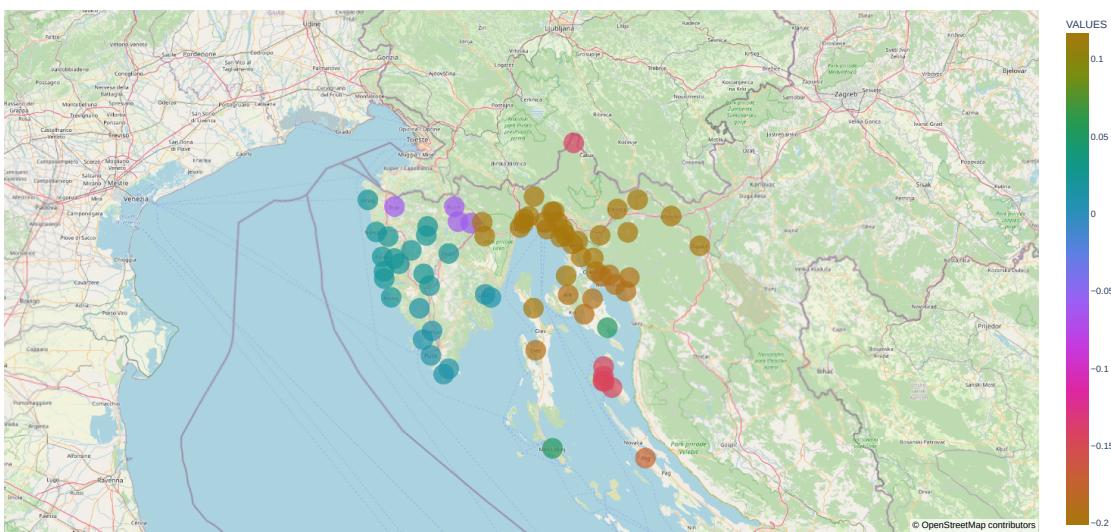
Plotting map for k = 0



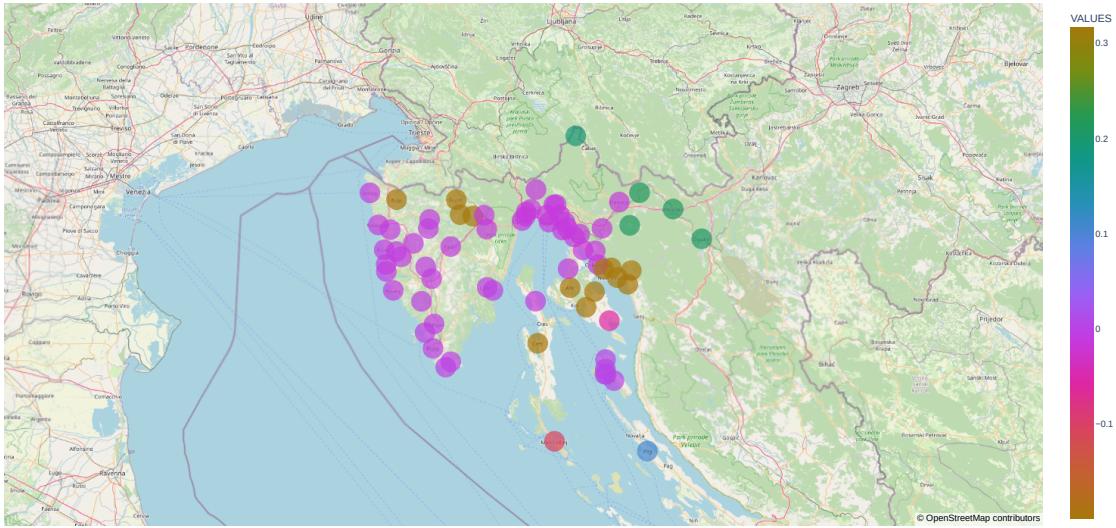
Plotting map for k = 1



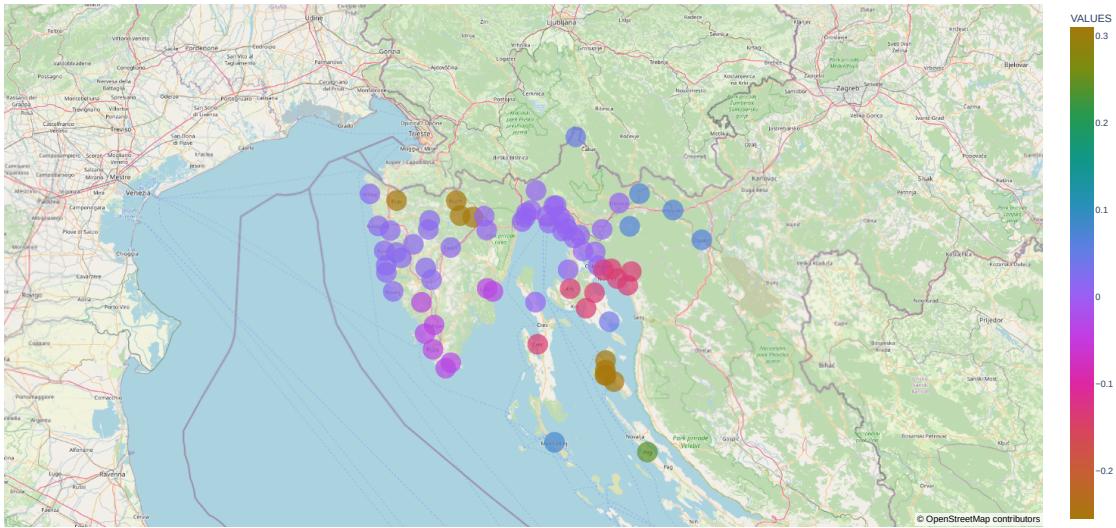
Plotting map for $k = 2$



Plotting map for $k = 3$



Plotting map for $k = 4$



1 Export to HTML

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

# wait few sec for plot above to finish
time.sleep(3)
```

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

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

```
[50]: # export notebook results to HTML  
jupyter_out_filename = f'{PREFIX_STR}_svd'  
!jupyter nbconvert --output-dir 'output' --output {jupyter_out_filename}  
    --to=HTML svd.ipynb  
!jupyter nbconvert --output-dir 'output' --output {jupyter_out_filename}  
    --to=pdf svd.ipynb  
  
jupyter_out_filename_no_code = f'{PREFIX_STR}_svd_no_code'  
!jupyter nbconvert --output-dir 'output' --output  
    {jupyter_out_filename_no_code} --no-input --to=HTML svd.ipynb  
!jupyter nbconvert --output-dir 'output' --output  
    {jupyter_out_filename_no_code} --no-input --to=pdf svd.ipynb
```

```
[NbConvertApp] Converting notebook svd.ipynb to HTML  
Traceback (most recent call last):  
  File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-  
  packages/nbformat/reader.py", line 18, in parse_json  
    nb_dict = json.loads(s, **kwargs)  
  File "/usr/lib/python3.9/json/_init_.py", line 346, in loads  
    return _default_decoder.decode(s)  
  File "/usr/lib/python3.9/json/decoder.py", line 337, in decode  
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())  
  File "/usr/lib/python3.9/json/decoder.py", line 353, in raw_decode  
    obj, end = self.scan_once(s, idx)  
json.decoder.JSONDecodeError: Unterminated string starting at: line 108588  
column 26 (char 14528210)
```

The above exception was the direct cause of the following exception:

```
Traceback (most recent call last):  
  File "/home/HDD/00-Development/016-Diplomski/venv/bin/jupyter-nbconvert", line  
8, in <module>  
    sys.exit(main())  
  File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-  
  packages/jupyter_core/application.py", line 269, in launch_instance  
    return super().launch_instance(argv=argv, **kwargs)  
  File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-  
  packages/traitlets/config/application.py", line 846, in launch_instance  
    app.start()  
  File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-  
  packages/nbconvert/nbconvertapp.py", line 414, in start  
    self.convert_notebooks()
```

```
File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-
packages/nbconvert/nbconvertapp.py", line 588, in convert_notebooks
    self.convert_single_notebook(notebook_filename)
File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-
packages/nbconvert/nbconvertapp.py", line 551, in convert_single_notebook
    output, resources = self.export_single_notebook(
File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-
packages/nbconvert/nbconvertapp.py", line 479, in export_single_notebook
    output, resources = self.exporter.from_filename(
File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-
packages/nbconvert/exporters/exporter.py", line 189, in from_filename
    return self.from_file(f, resources=resources, **kw)
File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-
packages/nbconvert/exporters/exporter.py", line 207, in from_file
    nbformat.read(file_stream, as_version=4), resources=resources, **kw
File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-
packages/nbformat/__init__.py", line 170, in read
    return reads(buf, as_version, capture_validation_error, **kwargs)
File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-
packages/nbformat/__init__.py", line 88, in reads
    nb = reader.reads(s, **kwargs)
File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-
packages/nbformat/reader.py", line 72, in reads
    nb_dict = parse_json(s, **kwargs)
File "/home/HDD/00-Development/016-Diplomski/venv/lib/python3.9/site-
packages/nbformat/reader.py", line 21, in parse_json
    raise NotJSONError(("Notebook does not appear to be JSON: %r" % s)[:77] +
"") from e
nbformat.reader.NotJSONError: Notebook does not appear to be JSON: '{\n "cells": [
\n   {\n     "cell_type": "c...
```