Data Cleaning & Interactive Bokeh Charts

September 10, 2020

[1]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import minmax_scale
from sklearn.decomposition import PCA
from matplotlib.cm import get_cmap
from matplotlib.colors import rgb2hex
from bokeh.models import ColumnDataSource, LabelSet, Arrow, NormalHead
from bokeh.plotting import figure
from bokeh.io import output_notebook, show, output_file
```

1 Data Loading

First we load the data with pandas read_csv() method and provide necessery parameters so pandas is able to use the German number formatting.

[2]:
```
data_path = 'data/NRW19.csv'
df = pd.read_csv(data_path, sep = ';', thousands = '.', decimal = ',')
df.shape
```

[2]:
```
(2447, 32)
```

2 Data Preparation

Second, we filter the data to remove all “Wahlkarten” and all aggregated results to get only the data by counties. This is done by only selecting rows where the GKZ does not end with “99” which identifys “Wahlkarten” and where the GKZ does not end with “00” which represents aggregated results.

[3]:
```
df = df[~df.GKZ.str.endswith('99') & # ~ for NEGATION
        ~df.GKZ.str.endswith('00')]
df.shape # have a look at the shape
```

[3]:
```
(2118, 32)
```

Now lets have a first peak at our data.

[4]:
```
df.head(5)
```

[4]:

```
        GKZ                      Gebietsname  Wahlbe-rechtigte  Stimmen  \
8   G10101                       Eisenstadt             10798     7192
11  G10201                             Rust              1594     1055
14  G10301  Breitenbrunn am Neusiedler See              1582     1137
15  G10302                  Donnerskirchen              1535     1076
16  G10303                     Großhöflein              1676     1164

    Ungültige  Gültige   ÖVP      %   SPÖ    %.1  …  BZÖ  %.8  BIER  %.9  \
8          78     7114  2966  41.69  1450  20.38  …  NaN  0.0   NaN  0.0
11         18     1037   355  34.23   315  30.38  …  NaN  0.0   NaN  0.0
14         27     1110   376  33.87   331  29.82  …  NaN  0.0   NaN  0.0
15         17     1059   470  44.38   242  22.85  …  NaN  0.0   NaN  0.0
16         25     1139   487  42.76   271  23.79  …  NaN  0.0   NaN  0.0

     CPÖ  %.10  GILT  %.11  SLP  %.12
8   21.0  0.30   NaN   0.0  NaN   0.0
11   3.0  0.29   NaN   0.0  NaN   0.0
14   0.0  0.00   NaN   0.0  NaN   0.0
15   0.0  0.00   NaN   0.0  NaN   0.0
16   0.0  0.00   NaN   0.0  NaN   0.0

[5 rows x 32 columns]
```

We see that we have some partys with none or only very few votes. Normally these partys are represented in one column “OTHERS”.

[5]:

```python
# select the other partys and ther associated percentages
other_partys = [df.columns[i] for i in range(18, len(df.columns), 2)]
other_partys_perc = [df.columns[i] for i in range(19, len(df.columns), 2)]
print(other_partys)
print(other_partys_perc)
```

```
['KPÖ', 'WANDL', 'BZÖ', 'BIER', 'CPÖ', 'GILT', 'SLP']
['%.6', '%.7', '%.8', '%.9', '%.10', '%.11', '%.12']
```

[6]:

```python
# create new columns
df['OTHERS'] = df[other_partys].sum(axis=1)
df['%.OTHERS'] = df[other_partys_perc].sum(axis=1)
```

[7]:

```python
# drop old columns
df = df.drop(columns=other_partys)
df = df.drop(columns=other_partys_perc)
```

Because we only need the perecntages for our representation we can drop all columns that contain the absolute values and rename the percentage columns to the party names. Furter, we can drop some unneeded columns.

[8]:

```
# columns to delete
delete_cols = ['Wahlbe-rechtigte', 'Ungültige', 'Gültige']
# columns to rename
rename_cols = {}
party_name = ''
for col_name in df.columns:
    if '%' in col_name:
        delete_cols.append(party_name)
        rename_cols[col_name] = party_name
    party_name = col_name
# drop unneeded columns
df = df.drop(columns=delete_cols)
# rename columns with percentages to party names
df = df.rename(columns = rename_cols)
```

[9]:

```
# reset the index and drop the index column
df = df.reset_index(drop = True)
```

Our final preprocessed dataset looks as follows:

[10]:

```
df.tail(5)
```

[10]:

```
         GKZ  Gebietsname  Stimmen    ÖVP    SPÖ    FPÖ   NEOS  JETZT  GRÜNE  \
2113  G91901      Döbling    25563  32.26  21.64  10.86  13.34   3.22  17.17
2114  G92001  Brigittenau    22535  20.07  36.48  14.62   5.83   2.92  17.83
2115  G92101  Floridsdorf    56568  25.99  30.49  20.13   6.65   2.84  11.94
2116  G92201   Donaustadt    69291  25.83  29.42  18.26   7.78   2.96  13.79
2117  G92301      Liesing    41240  28.19  27.35  15.93   9.67   2.69  14.50

      OTHERS
2113    1.50
2114    2.25
2115    1.95
2116    1.97
2117    1.68
```

We have 2118 Rows and 10 Columns as expected.

[11]:

```
df.shape
```

[11]:

```
(2118, 10)
```

3 Preperation For Visualization

3.1 State names

Because we want to show the the name of the state for the hover information we need to add a column with the respective name of the state. We can do this by using the GKZ.

[12]:

```
# map iso Number to State
stat_iso_map = {
    1: 'Burgenland',
    2: 'Kärnten',
    3: 'Niederösterreich',
    4: 'Oberösterreich',
    5: 'Salzburg',
    6: 'Steiermark',
    7: 'Tirol',
    8: 'Vorarlberg',
    9: 'Wien',
}
```

[13]:

```
# add the state_id from each row (first number in the GKZ)
df['state_id'] = df.apply(lambda row: int(row.GKZ[1]), axis = 1)
```

[14]:

```
# add the State for each row
df['state'] = df.apply(lambda row: stat_iso_map[row.state_id], axis = 1)
```

3.2 Color

As we will color according to the state, we need to add a column with the desired color. We use one of Matplotlibs colormaps for this.

[15]:

```
# selct color map
color_map = get_cmap('Set1', 9)
# function to convert rgb colors from colormap to hex colors
def get_color(i):
    rgb = color_map(i)[:3]
    return rgb2hex(rgb)
```

[16]:

```
# we scale the state_id to a range from 0 to 1 because
# the color map is accessed in that range
df['color_index'] = minmax_scale(df['state_id'])
```

[17]:

```
# add the color by State to each row in the DataFrame
df['color'] = df.apply(lambda row: get_color(row.color_index), axis = 1)
```

3.3 Size

To diplay the circles in our visualizaten we create a new column for size and use sklearn's min-max_scale function to create the desired size range.

[18]:

```
# range for size of circles from mimimum to maximum value
size_range = (5,50)
```

[19]:

```
# add the size value for each row in the DataFrame
df['size'] = minmax_scale(df['Stimmen'], feature_range = size_range)
```

3.4 PCA

Now we use sklearn's PCA to reduce the data to 2 dimensions.

[20]:

```
# select only the columns with party results
df_parties = df.iloc[:,3:10].copy()
```

[21]:

```
# create PCA for two dimensions
pca = PCA(n_components=2)
```

[22]:

```
# fit and transform the data
pca_np = pca.fit_transform(df_parties)
```

[23]:

```
# save results to dataframe with new column names
df_pca = pd.DataFrame(pca_np, columns = ['pca_x', 'pca_y'])
```

[24]:

```
# select columns which are needed for the visualization
df_keep = df[['state', 'Gebietsname', 'Stimmen', 'color', 'size']].copy()
```

[25]:

```
# create final dataset to be visualized by adding the PCA data
vis_df = pd.concat([df_keep, df_pca], axis = 1)
```

3.5 Arrows

In the PCAs components_ attribute we find the:

"Principal axes in feature space, representing the directions of maximum variance in the data."

This gives us the direction of the original features and their "importance" for the PCA. Thus, we use it to represent the partys as "projected axes" by adding arrows to the Biplot.

[26]:

```
length_mulipy = 30 # mulitpy to make arrows better visible
# dataframe for arrows
arrow_df = pd.concat([pd.DataFrame(pca.components_.T * length_mulipy,
                                   columns = ['x', 'y']),
                      pd.DataFrame(df_parties.columns, columns = ['party'])],
                     axis = 1)
```

This is how the arrow data looks.

[27]:

```
arrow_df
```

[27]:

```
           x          y   party
0  23.896269 -11.467790     ÖVP
1 -17.742324 -17.463959     SPÖ
2  -2.926348  -2.552000     FPÖ
```

```
3  -0.018427  10.655857    NEOS
4  -0.551061   1.517379   JETZT
5  -2.271993  18.449942   GRÜNE
6  -0.385742   0.858567  OTHERS
```

4 Visualization

[28]:

```
TITLE = 'Biplot of PCA analysis for Austrias \
National Council Election 2019 (colord by State)'

TOOLS = 'hover,tap,pan,crosshair,box_zoom,\
wheel_zoom,zoom_in,zoom_out,lasso_select,save,reset'

# create figure
p = figure(tools=TOOLS,
           toolbar_location='right',
           plot_width=850,
           title=TITLE,
           aspect_ratio = 1.2)

# set tooltips
p.hover.tooltips = [
    ('Gemeinde', '@Gebietsname'),
    ('Bundesland:', '@state'),
    ('Stimmen', '@Stimmen')
]

# draw circles
source_circle = ColumnDataSource(vis_df)
p.circle('pca_x', 'pca_y',
         source=source_circle,
         size='size',
         color='color',
         fill_alpha=0.8,
         legend_group='state')

# add arrows
for index, row in arrow_df.iterrows():
    p.add_layout(Arrow(end=NormalHead(size = 5),
                       line_width = 1,
                       x_start=0,
                       y_start=0,
                       x_end=row['x'],
                       y_end=row['y']))

# add text for arrows
```

```
source_text = ColumnDataSource(arrow_df)
labels = LabelSet(source=source_text,
                  x='x',
                  y='y',
                  text='party',
                  x_offset=1,
                  y_offset=1,
                  level='overlay',
                  text_font_size = '0.8em',
                  text_color = 'black')
p.add_layout(labels)

# further seetings for figure
p.toolbar.logo = 'grey'
p.background_fill_color = 'white'
p.grid.grid_line_color = '#e6e6e6' # light grey
p.legend.label_text_font_size = '0.8em'

# show plot in notebook
output_notebook()
# save output
output_file("voting_biplot.html", title="Biplot Voting")

# show plot in notebook
show(p)
```