Contents

Temporary Notes

Abstract

1 INTRODUCTION

1.1 State of the Art in Data Science

Use stuff from the presentation

1. Why is data analysis useful?
    1.1. Modern amounts of data explanation
    1.2. Data analysis explanation
2. What tools are used to deal with large amounts of data?
    2.1. Traditional - ETL and DW (microsoft, qlikview, ...)
    2.2. Explorational - spark, R, python, matlab
3. Why choose python?
    3.1. Advantages and disadvantages of using python
    3.2. Overview of chosen packages

Rewrite everything

1.2 Goals of the Research

In the modern world, big data and machine learning are becoming more and more prominent as companies such as Facebook, Google and Amazon gather and analyze all sorts of data from their users. But which tools are they using to do it?

Right now, the two main languages in data science are `Python` and `R`, while `Matlab` is also quite popular despite only being used in the academic environment.

This work's objective is to show how to use the Python 3 programming language in dealing with different kinds of data, and to help clarify any problems that might come up. It may be useful for long-term users of other languages that want to try Python out as well as users of Python 2, support for which will be stopped in 2020.

1.3 Thesis Overview

This work will be split into three parts, each working with a different dataset.

In the first part, I'll show you how to obtain, plot and predict stocks based on the last 17 years' worth of stock data from NYSE[1]. I'll also cover some common problems that might occur

when one is trying to deal with such amount of data.

In the second part, I'll cover scraping facebook's API, and plotting geolocation data of their events. I'll also discuss some problems that might occur while trying to download data, as well as how to use latest tools from Python (like the asyncio library) to speed up the data gathering part greatly. I'll also give you a brief overview of the current data visualization landscape, and show you which plotting packages are the best to use when dealing with geolocation data.

In the third part, I'll delve into the YouTube system, and will try to download and analyze their videos.

Lastly, the fourth part will contain conclusions.

[1]New York Stock Exchange

2 TABULAR DATA ANALYSIS

2.1 Project Overview

1. Goals
    1.1. Describe the project
2. Obtaining the data (pandas-datareader)
3. Traditional stock visualizations (matplotlib)
4. Stock correlation matrix (matplotlib)
5. LSTM training (keras)

2.2 Data gathering (pandas-datareader)

2.3 Traditional stock visualizations (matplotlib)

2.4 Stock correlation matrix visualization (matplotlib)

2.5 Predictive analysis with a LSTM neural network (keras)

3 GRAPH DATA ANALYSIS

3.1 Project Overview

1. Goals

   1.1. to show how to run community detection algorithms in igraph

   1.2. to show how to plot the communities using two different methods - datashader (larger data) and cairo (smaller data)

   1.3. to show how to make the communities visually separatable and how to incorporate node weights in the plot

2. Tools(Libraries) used

   2.1. Why I chose igraph

3.2 Preprocessing and igraph creation

1. Importing the data form konekt
2. Optimizing edges renaming with numpy vectorize/jit

3.3 Community detection (igraph)

3.4 Plotting large graphs (datashader)

Add a note about how datashader failed to plot the entire graph and why it's not a good idea in the first place (hard to see the points)

3.5 Plotting small graphs (igraph)

3.5.1 Simple plot

Clustering the subgraph

The size of the subgraph we're clustering is smaller than the one used in the datashader example (only nodes), so we can use the infomap clustering algorithm on it right away.

list the number of nodes

```
# Selecting high degree nodes
hdg_vertices = g.vs.select(_degree_ge=800)  # Select vertices with a high degree
hdg_subgraph = hdg_vertices.subgraph()  # Create a new subgraph
hdg_vcount = hdg_subgraph.vcount()  # Will be used later in the layout calculation
# Check the number of vertices
print(f'Number of vertices in the subgraph: {hdg_vcount}')
```

Listing 1: Using infomap to cluster the subgraph

Making communities visible

There are a couple of ways to make communities in your graph more visible on the resulting plot. You could (1) use color to distinguish between them, (2) draw vertices from one community close to each other, (3) separate communities by drawing their boundaries, or (4) label each vertice with their community label. Some of those techniques are only effective when applied to very small graphs (like labelling), while other are a better fit for a medium-sized graph like the one used in this example.

```
# Setting up the plot
hdg_style = {}  # Creating a style dictionary
hdg_style['layout'] = \
    hdg_subgraph.layout_fruchterman_reingold(maxiter=1000, area=hdg_vcount**3)
hdg_style['vertex_size'] = 0.001
hdg_style['bbox'] = (500, 500)
hdg_style['vertex_shape'] = 'circle'
hdg_style['edge_width'] = 0.1
```

Listing 2: Initializing color and weight lists

```
# Plotting
save_fname = 'plot_naive.svg'
ig.plot(hdg_subgraph, save_fname, **hdg_style)
```

Listing 3: Assigning color to vertices

```
# Selecting high degree nodes
hdg_vertices = g.vs.select(_degree_ge=800)  # Select vertices with a high degree
hdg_subgraph = hdg_vertices.subgraph()  # Create a new subgraph
hdg_vcount = hdg_subgraph.vcount()  # Will be used later in the layout calculation
# Check the number of vertices
print(f'Number of vertices in the subgraph: {hdg_vcount}')

# Setting up the plot
hdg_style = {}  # Creating a style dictionary
hdg_style['layout'] = \
    hdg_subgraph.layout_fruchterman_reingold(maxiter=1000, area=hdg_vcount**3)
hdg_style['vertex_size'] = 0.001
hdg_style['bbox'] = (500, 500)
hdg_style['vertex_shape'] = 'circle'
hdg_style['edge_width'] = 0.1

```

```
17  # Plotting
18  save_fname = 'plot_naive.svg'
19  ig.plot(hdg_subgraph, save_fname, **hdg_style)
```

Listing 4: Assigning color to edges

3.5.2 Simple community plot

```
# Clustering the high degree subgraph
hdg_imap = hdg_subgraph.community_infomap()
hdg_membership = hdg_imap.membership
```

Listing 5: Using infomap to cluster the subgraph

```
# Selecting random colors for groups using a list comprehension with an f-string
community_colors = [f'#{random.randint(0, 0xFFFFFF):06x}' for comm in hdg_imap]
# Creating a list that will be used to assign color to every vertice
vert_colors = list(range(hdg_subgraph.vcount()))
# Initializing lists that will hold the edge attributes
edge_colors = []
edge_weights = []
```

Listing 6: Initializing color and weight lists

```
# Assigning the vertice color based on their community
for comm_id, comm in enumerate(hdg_imap):
    for vert in comm:
        vert_colors[vert] = community_colors[comm_id]
```

Listing 7: Assigning color to vertices

```
# Assigning the edge color and weight based on the vertices it connects
# Adding weights will make the group separation more visible
for edge in hdg_subgraph.es:
    if hdg_membership[edge.source] == hdg_membership[edge.target]:
        edge_colors.append(vert_colors[edge.source])
        edge_weights.append(3 * hdg_vcount)
    else:
        edge_colors.append('#dbdbdb')
        edge_weights.append(0.1)
```

Listing 8: Assigning color to edges

```
# Styling the plot - graph properties
hdg_subgraph.vs['color'] = vert_colors  # Adding color as a vertice property
hdg_subgraph.es['color'] = edge_colors  # Adding color as an edge property
hdg_subgraph.es['weight'] = edge_weights  # Adding weight as an edge property
```

Listing 9: Styling using graph properties

```
# Styling the plot - style dictionary
hdg_comm_style = {}
hdg_comm_style['layout'] = \
```

```
    hdg_subgraph.layout_fruchterman_reingold(maxiter=1000,
                                             weights=hdg_subgraph.es['weight'],
                                             area=hdg_vcount**3,
                                             repulserad=hdg_vcount**3)
hdg_comm_style['bbox'] = (500, 500)
hdg_comm_style['vertex_size'] = 0.5
hdg_comm_style['vertex_shape'] = 'circle'
hdg_comm_style['edge_width'] = 0.1
hdg_comm_style['palette'] = ig.PrecalculatedPalette(community_colors)
# hdg_comm_style['mark_groups'] = True # Activate if you want to delineate the groups
```

Listing 10: Styling using a style dict

```
# Plotting
save_fname = 'plot_infomap.svg'
ig.plot(hdg_imap, save_fname, **hdg_comm_style)  # mark_groups=True
```

Listing 11: Saving the plot to a file

```
# Clustering the high degree subgraph
hdg_imap = hdg_subgraph.community_infomap()
hdg_membership = hdg_imap.membership

# Selecting random colors for groups using a list comprehension with an f-string
community_colors = [f'#{random.randint(0, 0xFFFFFF):06x}' for comm in hdg_imap]
# Creating a list that will be used to assign color to every vertice
vert_colors = list(range(hdg_subgraph.vcount()))

# Initializing lists that will hold the edge attributes
edge_colors = []
edge_weights = []

# Assigning the vertice color based on their community
for comm_id, comm in enumerate(hdg_imap):
    for vert in comm:
        vert_colors[vert] = community_colors[comm_id]

# Assigning the edge color and weight based on the vertices it connects
# Adding weights will make the group separation more visible
for edge in hdg_subgraph.es:
    if hdg_membership[edge.source] == hdg_membership[edge.target]:
        edge_colors.append(vert_colors[edge.source])
        edge_weights.append(3 * hdg_vcount)
    else:
        edge_colors.append('#dbdbdb')
        edge_weights.append(0.1)

# Styling the plot - graph properties
```

```
hdg_subgraph.vs['color'] = vert_colors  # Adding color as a vertice property
hdg_subgraph.es['color'] = edge_colors  # Adding color as an edge property
hdg_subgraph.es['weight'] = edge_weights  # Adding weight as an edge property

# Styling the plot - style dictionary
hdg_comm_style = {}
hdg_comm_style['layout'] = \
    hdg_subgraph.layout_fruchterman_reingold(maxiter=1000,
                                             weights=hdg_subgraph.es['weight'],
                                             area=hdg_vcount**3,
                                             repulserad=hdg_vcount**3)
hdg_comm_style['bbox'] = (500, 500)
hdg_comm_style['vertex_size'] = 0.5
hdg_comm_style['vertex_shape'] = 'circle'
hdg_comm_style['edge_width'] = 0.1
hdg_comm_style['palette'] = ig.PrecalculatedPalette(community_colors)

# Plotting
save_fname = 'plot_infomap.svg'
ig.plot(hdg_imap, save_fname, **hdg_comm_style)
```

Listing 12: Full version of the code

3.5.3 Weighted community plot

Pagerank application

Pagerank is an algorithm developed by google. cite

```
hdg_pgrank = hdg_subgraph.pagerank()
hdg_pgrank_arr = np.array(hdg_pgrank)
```

Listing 13: Using pagerank to assign weights to vertices

Style dictionary

Style dictionary for the weighted graph is very similar to the one that was used to create the regular community plot, with an addition of ...

```
hdg_comm_style = {}
hdg_comm_style['layout'] = \
    hdg_subgraph.layout_fruchterman_reingold(maxiter=1000,
                                             weights=hdg_subgraph.es['weight'],
                                             area=hdg_vcount**3,
                                             repulserad=hdg_vcount**3)
hdg_comm_style['bbox'] = (500, 500)
hdg_comm_style['vertex_size'] = hdg_pgrank_arr * 3000
hdg_comm_style['vertex_shape'] = 'circle'
hdg_comm_style['edge_width'] = 0.1
hdg_comm_style['palette'] = ig.PrecalculatedPalette(community_colors)
```

Listing 14: Styling using a style dict

Plotting and results

```
save_fname = 'plot_pagerank_fg.svg'
ig.plot(hdg_imap, save_fname, **hdg_comm_style)
```

Listing 15: Saving the plot to a file

```
hdg_pgrank = hdg_subgraph.pagerank()
hdg_pgrank_arr = np.array(hdg_pgrank)

# Style dict
hdg_comm_style = {}
hdg_comm_style['layout'] = \
    hdg_subgraph.layout_fruchterman_reingold(maxiter=1000,
                                             weights=hdg_subgraph.es['weight'],
                                             area=hdg_vcount**3,
                                             repulserad=hdg_vcount**3)
```

```
hdg_comm_style['bbox'] = (500, 500)
hdg_comm_style['vertex_size'] = hdg_pgrank_arr * 3000
hdg_comm_style['vertex_shape'] = 'circle'
hdg_comm_style['edge_width'] = 0.1
hdg_comm_style['palette'] = ig.PrecalculatedPalette(community_colors)

save_fname = 'plot_pagerank_fg.svg'
ig.plot(hdg_imap, save_fname, **hdg_comm_style)
```

Listing 16: Full version of the code

4 GEODATA ANALYSIS

4.1 Project Overview

1. Goals
    1.1. to show how to deal with geodata in python
2. Plots
    2.1. Animated plot - KPI per country (libnamehere)
    2.2. Plot - KPI per state in a country (libnamehere)
    2.3. Plot - KPI per region in a city (libnamehere)

4.2 Country-level plotting (libnamehere)

4.3 State-level plotting (libnamehere)

4.4 City-level plotting (libnamehere)

References

[1] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]. *Annalen der Physik*, 322(10):891–921, 1905.

[2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[3] Jean Francois Puget. The most popular language for machine learning and data science is âĂę.