Extending RAWGraphs
Project in Information Visualisation

Lukas Brunner, Jochen Flachhuber, Lukas Kurzmann, Tobias Striemitzer

Institute of Interactive Systems and Data Science (ISDS),
Graz University of Technology
A-8010 Graz, Austria

30 Jun 2019

Abstract

Since information visualization becomes more and more important, tools for representing data in form of graphs and charts becomes a key element of information visualization. One of the frameworks which can be used to generate data visualizations is RAWGraphs. This paper is going to introduce RAWGraphs, which calls itself *the missing link between spreadsheets and data visualization*. RAWGraphs consists of a lot of different possibilities to visualize data, is extendable and there is a possibility to self host it.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

In the jungle of information visualization tools it is hard to find the right one for visualizing data. RAWGraphs is one framework, which is easy to handle and the user is able to get quick results without any prerequisites. Unfortunately the amount of possible built-in graphs is limited. In total there are currently 21 different graph types than can be chosen. By investigating the available graphs, there was one very common graph missing, namely the line chart[Mauri et al. 2017b]. This paper will cover in the first section some basic information about RAWGraphs itself and how to use it, the next section explains how to extend RAWGraphs. Since RAWGraphs uses the D3 JavaScript library which got a rather large update last year from version 3 to version 5 the following section also show some basics of how to update from version 3 to version 5 on the basis of a custom implemented paired bar chart. The last section is going to give some detailed information about a similarity map extension for RAWGraphs and the algorithms behind it, like principal component analysis (PCA), force directed placement (FDP) and t-distributed stochastic neighbourhood embedding (t-SNE).

Chapter 2

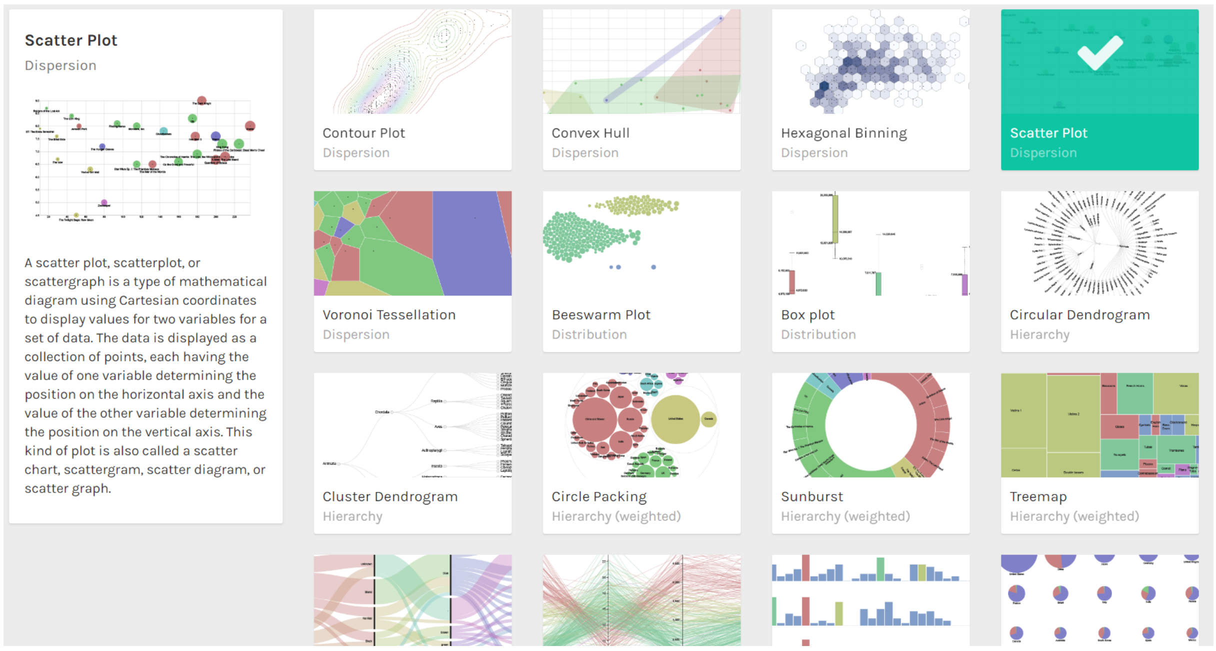
RAWGraphs General

Visualisations are a useful way to present data to an audience. So creating them is a big aspect of data science. That is why there are a lot of handy tools, which make it easier for the user to generate them, without too much prior knowledge. One of those tools is the open source web application RAWGraphs. The main intention for the developers was to make it easy for the user to create a graph by just having the raw data in form of either a file, like csv or xls, or just by pasting a it into a given textbox. The application offers a variety of static charts to use from [Mauri et al. 2017b].

RAWGraphs has 21 predefined charts which include well-known ones, like Bar or Pie Charts, as well as uncommon and exotic ones, like Streamgraphs or Alluvial Diagram. Figure 2.1 shows part of the graph selection screen. Every chart has a preview image, a title and a category shown on the overview. Clicking on a chart opens a description window which can be seen in Figure 2.2. To use the application, some knowledge about the visualisations themselve is still highly advised, because the descriptions are bare-boned and do not tell the user, which type of dimensions he needs to choose for his data. A full list of all the available Graphs is shown here (order defined by the application) [Mauri et al. 2017b]:

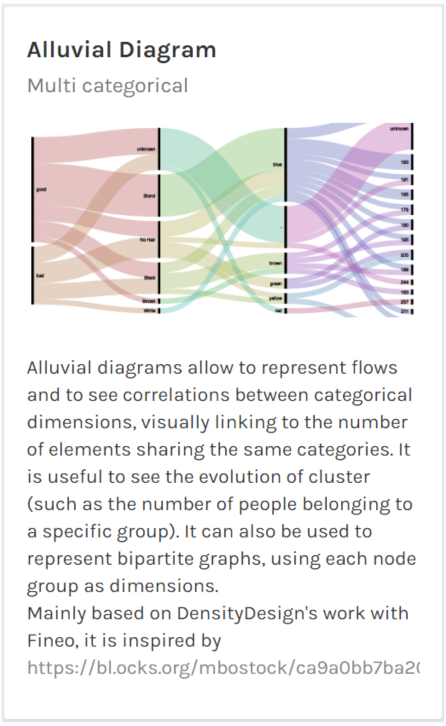
- Contour Plot
- Convex Hull
- Hexagonal Binning
- Scatter Plot
- Voronoi Tessellation
- Beeswarm Plot
- Box plot
- Circular Dendrogram
- Cluster Dendrogram
- Circle Packing
- Sunburst
- Treemap
- Alluvial Diagram
- Parallel Coordinates
- Bar chart
- Pie chart
- Gantt Chart
- Area graph
- Bump Chart
- Horizon Graph
- Streamgraph

One of the most important things about RAWGraphs for advanced users is, that everyone is able to create their own custom charts. To do that it is necessary to self-host the application and have some knowledge about the JavaScript library D3 [Mauri et al. 2017b]. The exact steps necessary to achieve a custom chart will be discussed in a later chapter.


Scatter Plot
Dispersion

A scatter plot, scatterplot, or scattergraph is a type of mathematical diagram using Cartesian coordinates to display values for two variables for a set of data. The data is displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis. This kind of plot is also called a scatter chart, scattergram, scatter diagram, or scatter graph.

Figure 2.1: Overview of some of the predefined charts available in RAWGraphs. [Screenshot taken by the author of this paper from the Web Application RAWGraphs [Mauri et al. 2017a]]


Alluvial Diagram
Multi categorical

Alluvial diagrams allow to represent flows and to see correlations between categorical dimensions, visually linking to the number of elements sharing the same categories. It is useful to see the evolution of cluster (such as the number of people belonging to a specific group). It can also be used to represent bipartite graphs, using each node group as dimensions.
Mainly based on DensityDesign's work with Fineo, it is inspired by https://bl.ocks.org/mbostock/ca9a0bb7ba2(

Figure 2.2: A description of the Alluvial Diagram. [Screenshot taken by the author of this paper from the Web Application RAWGraphs [Mauri et al. 2017a]]

Chapter 3

Extending Raw Graphs

The following chapter is going to explain some basics to create a custom chart in RAWGraphs [Mauri et al. 2017a].

3.1 Getting started

To extend RAWGraphs, clone their Git-repository and follow the instructions to install and run RAWGraphs as a local version. After cloning the repository browse to the raw folder and execute the command 'bower install' to setup the project. RAWGraphs can now run on a local web server with Python's built-in server: python -m http.server 4000. Use your web browser of choice and navigate to `localhost:4000/`.

3.2 Structure

First of all a new script file has to be created (newChart.js) and added in the charts/ folder. After creating the script file, add it to the references in the index.html. The index.html includes all script files that are needed and is the starting point of the RAWGraphs application. To add datasets that will be included in the samples, which simplifies the process of testing the newly created chart two modifications have to be done. The first one is the controller.js file which can be found in the js/ folder. In this file all sample datasets are included with a title, a type and an url. The created dataset has to be in the data/ folder saved as a common data exchange format (CSV, TSV or DSV). These few steps are neccessary for the structure of a custom chart

3.3 Create new Chart

This is a short explanation how to create a new chart in RAWGraphs. As previously mentionen first include the new script file in the index.html seen in Listing 3.1.

After adding a new script file in the chart/ folder, the implementation can begin. Starting with a self-executing function seen in Listing 3.2 which wraps the whole chart code.

3.3.1 Model

Afterwards a model has to be defined. This model should handle the incoming data. Depending on the chart some dimensions need to be defined e.g x and y axis. The available data types of these parameters are: number, string and date. Also adding a title is important too, because the user needs to know what the dimension is used for similar to Listing 3.3.

```html
<script src="charts/ newChart.js"></script>
```

Listing 3.1: This listing shows how to add a reference in index.html.

```javascript
(function(){
    // your code here...
})();
```

Listing 3.2: This listing shows the self executing function.

3.3.2 Map

To define the dimensions for the model a so called map function is needed to transform the data into viable objects. In this model.map funtion the complete data set is included and will return the values directly for the chart which can be seen in Listing 3.4.

3.3.3 Chart

The next step for creating a new chart is the chart implementation itself, starting with a simple line to create the new chart. Afterwards a title and a description which is displayed when choosing a chart on the RAWGraphs site should be implemented. There are also some variables for options of the chart e.g. width and height which can be found on the left side of the selected chart. Most of them are text boxes or checkboxes to adjust the chart as seen in Listing 3.5.

3.3.4 Draw

The last part to complete the new chart is the draw function. All the objects that should be drawn are defined in this function by using D3 version 5 as seen in Listing 3.6.

```javascript
//creating the model
var model = raw.model();

//model dimensions
var x = model.dimension()
    .title("X Axis")
    .types(Number);

var y = model.dimension()
    .title("Y Axis")
    .types(Number);
```

Listing 3.3: This listing shows how to create a model and define the dimensions for the model.

```javascript
//map function
model.map(function(data) {
    return data.map(function(d) {
        return {
            x : +x(d),
            y : +y(d)
        }
    })
})
```

Listing 3.4: This listing shows the map function which receives the whole data set.

```javascript
//creating the chart
var chart = raw.chart();

//title and description of the chart
chart.title("New chart")
    .description("Description of the new custom chart.")

//Boxes left side of the chart
var width = chart.number()
    .title('Width')
    .defaultValue(900)
var height = chart.number()
    .title('Height')
    .defaultValue(600)
```

Listing 3.5: This listing shows the creation of the chart and all information that is needed.

```
// Draw the bars
barchart.selectAll(".bar")
    .data(item.values)
    .enter().append("rect")
    .attr("transform", "translate(0," + titleSpace + ")")
    .attr("class", "bar")
    .attr("x", function(d) {
        return xScale(d.category);
    })
    .attr("width", xScale.bandwidth())
    .attr("y", function(d) {
        return yScale(d.size);
    })
    .attr("height", function(d) {
        return h - yScale(d.size);
    })
    .style("fill", function(d) {
        return colors()(d.color);
    });
```

Listing 3.6: This listing shows an example of a bar chart on how to use the draw function to create specific objects.

Chapter 4

Line Chart

The following chapter is going to explain what a Line Chart is and how to create one in RAWGraphs.

4.1 About

A line chart is a graph which displays information as a series of points connected via a line. It is often used to visualize data over a certain period of time. Figure 4.1 shows an example of a line chart.

4.2 Implementation

The main implementation for the line chart is split up into two main parts and using D3 version 5 [Lea 2019]. The first part contains the x and y axis which uses the index of the data and then appends a circle for each data point of the given data set. The second part appends a path between the data points. The source code can be seen in Listings 4.1 and 4.2.

4.3 Features

Some more features were added in this implementation. An important feature when saving the chart as SVG is the possibility to use a viewbox (Listing 4.3) instead of using the hardcoded values for width and height. If there are too many values for the x-axis which leads to overlapping labels, an options text box was created to rotate these values around an angle (Listing 4.4). This should help to see all of the given data on the x-axis. In the standard version of the line chart each line is displayed with all the data points on it as seen in Figure 4.4. A checkbox was added to give the user the choice if the datapoints should be displayed. If there is more than one line in the chart, a checkbox with the possibility to show a legend of the different datasets was implemented (Listing 4.5). Sometimes smooth lines are needed to explain the given data set better, so a checkbox which enables this feature was added (Figure 4.6). Each of these features can easily be changed by the user to give the user more opportunities to work with the given data set (Figure 4.2).

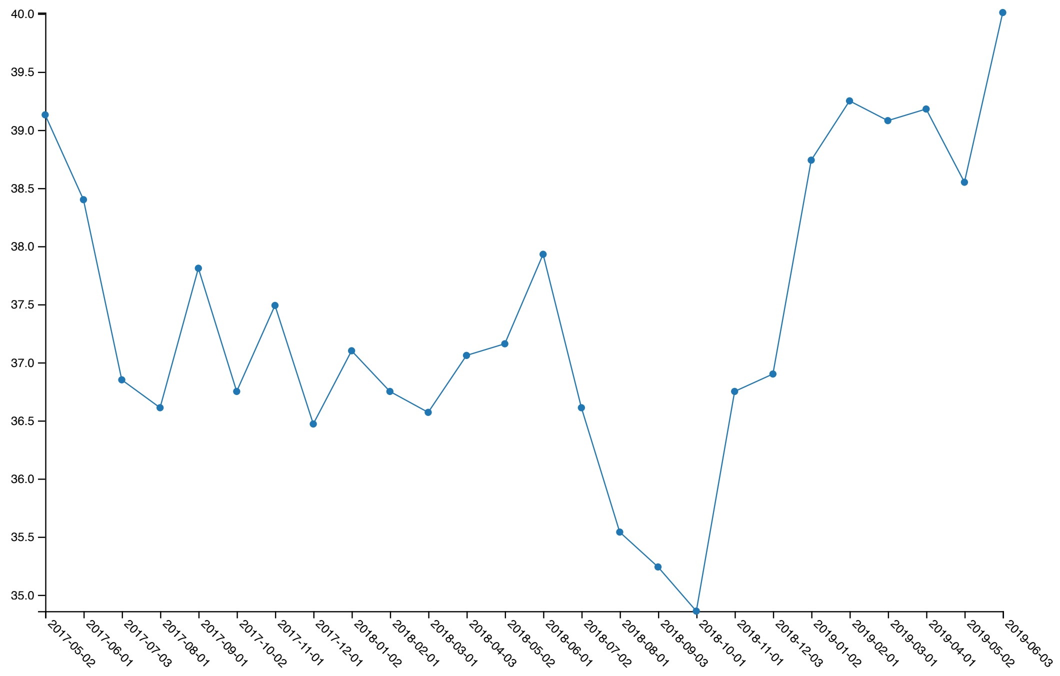


Figure 4.1: Line Chart with the Gold prices as datapoints in the period from 2017 to 2019. [Image was exported from RAWGraphs tool]

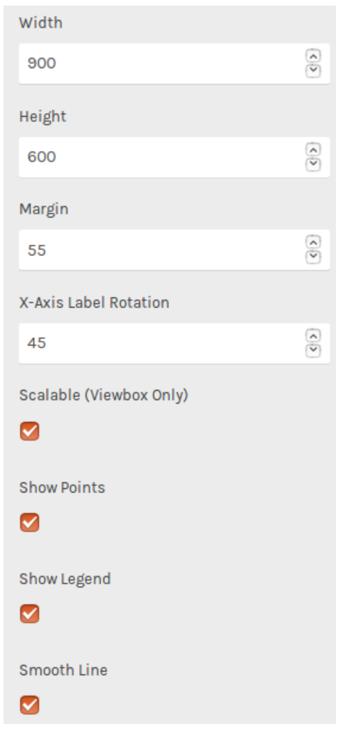
```
var xScale = d3.scaleLinear()
  .domain([0, n-1])
  .range([data_margin.left, data_width-data_margin.right])
var yScale = d3.scaleLinear()
  .domain([min, max])
  .range([data_height-data_margin.top, data_margin.bottom])

svg.selectAll(".dot-"+key)
  .data(dataset)
  .enter().append("circle") // Uses the enter().append() method
  .attr("class", "dot-"+key) // Assign a class for styling
  .attr("cx", function(d, i) { return xScale(i).toFixed(2)})
  .attr("cy", function(d) { return yScale(d.y).toFixed(2) })
  .attr("r", 3)
  .style("fill",color(key));
```

Listing 4.1: This listing shows the creation of the indices and the circles from the data set

```
svg.append("path")
  .datum(dataset)
  .attr("d", lines[key])
  .style("fill","#FFFFFF")
  .style("fill-opacity","0")
  .style("stroke",color(key));
```

Listing 4.2: This listing shows the creation of the path between the data points


Width

900

Height

600

Margin

55

X-Axis Label Rotation

45

Scalable (Viewbox Only) ✓

Show Points ✓

Show Legend ✓

Smooth Line ✓

Figure 4.2: Shows all possible features that can be selected. [Screenshot was created by the author]

```
if(scalable()) {
  selection
    .attr("viewBox", "0 0 " + width() + " " + height());
  document.getElementById('chart').style.width = "100%";
}
else {
  selection
    .attr("width", width())
    .attr("height", height())
.attr("viewBox", "0 0 " + width() + " " + height());
document.getElementById('chart').style.width = "auto";
        }
```

Listing 4.3: This listing shows the Viewbox as a checkbox

```
.attr('transform',function(d,i){
  return 'rotate('+textRotation()+')';
});
```

Listing 4.4: This listing shows the option to rotate the x-axis values

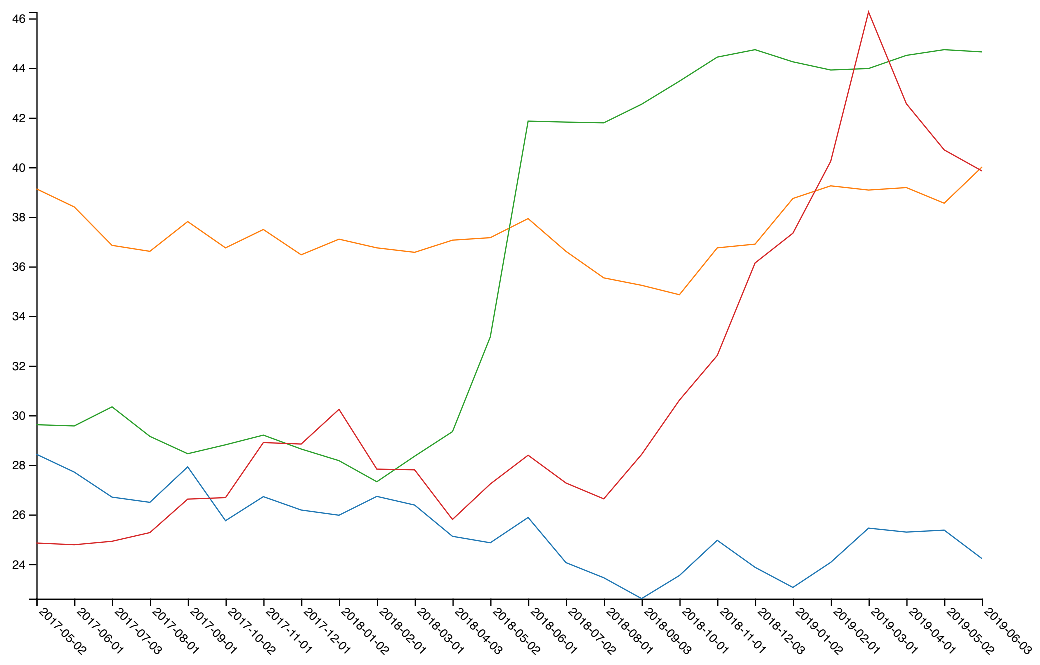


Figure 4.3: Line Chart with the rotated x-axis [Image was exported from RAWGraphs tool]

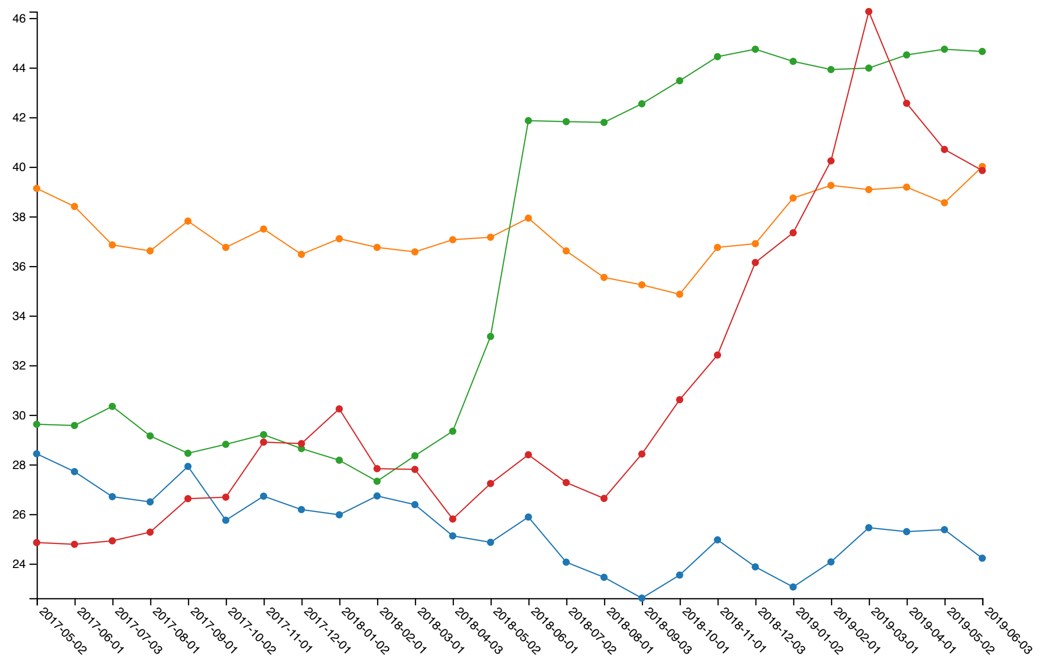


Figure 4.4: Line Chart with data points selected [Image was exported from RAWGraphs tool]

```
if(showLegend()) {
  var legend = selection.append("g")
    .attr("font-family", "sans-serif")
    .attr("font-size", 10)
    .attr("text-anchor", "end")
  var y = 10;
  Object.keys(colors).forEach(function(key) {
    y += 30;
    legend.append("rect")
      .attr("x", width() - 17)
      .attr("y", y)
      .attr("width", 15)
      .attr("height", 15)
      .attr("fill", colors[key])
      .attr("stroke", '#000')
      .attr("stroke-width",2);

    legend.append("text")
      .attr("x", width() - 24)
      .attr("y", y+7.5)
      .attr("dy", "0.32em")
      .text(key)

  });
}
```

Listing 4.5: This listing shows the possibility to have a legend of each dataset

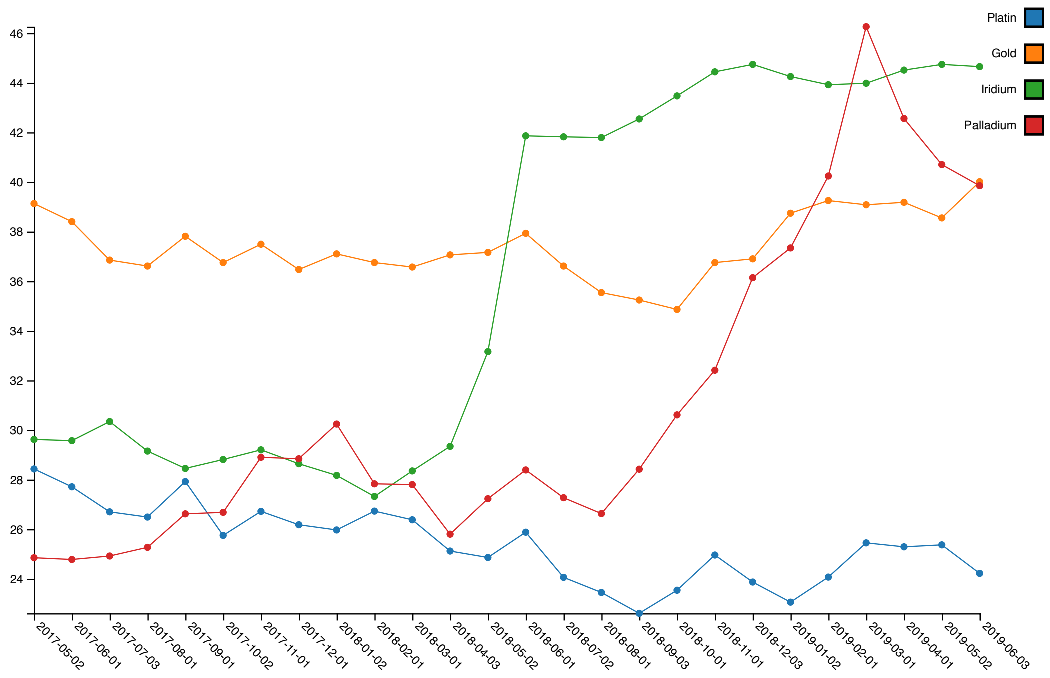


Figure 4.5: Line Chart with the selected Legend in the top right corner. [Image was exported from RAWGraphs tool]

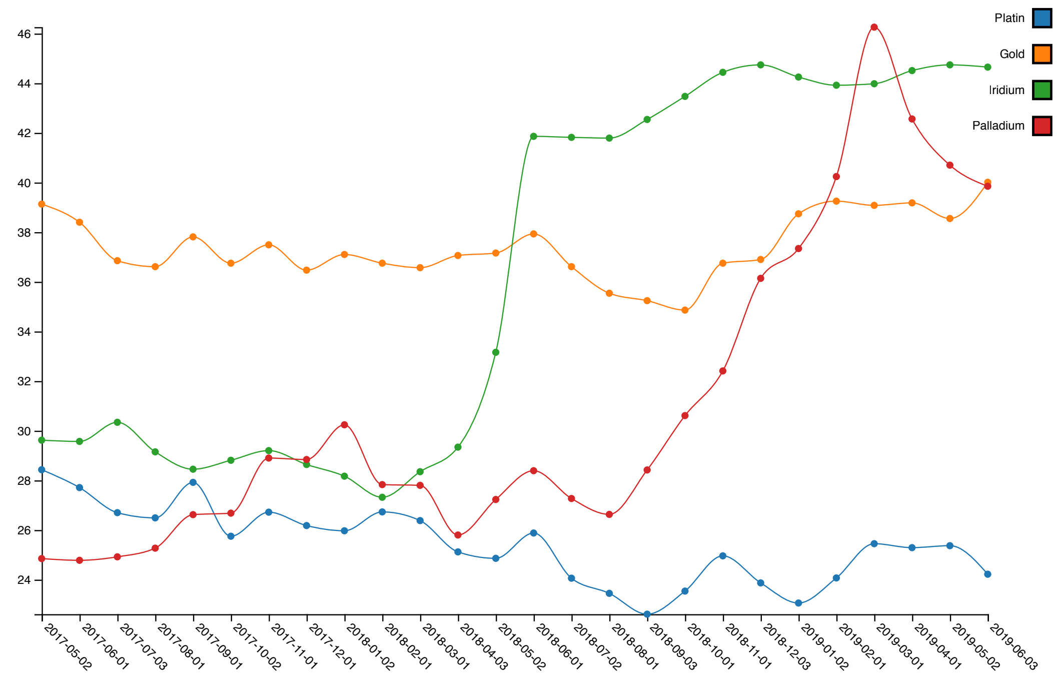


Figure 4.6: Line Chart with smooth lines selected. [Image was exported from RAWGraphs tool]

Chapter 5

D3v3 vs. D3v5

The following chapter is going to explain some basics about RAWGraphs update process of charts, and how to convert D3 version 3 to D3 version 5 on basis of a paired bar chart. First of all some terminologies have to be clarified.

5.1 Paired Bar Chart

A paired bar chart which is also called pyramid chart is a chart consisting of one common axis and 2 groups. This chart type creates the possibility to compare two categories which are sharing some common value. The most common usage of paired bar charts is for displaying information of populations based on their age. Figure 5.1 shows a sample chart of population based on age and gender.

5.2 D3js

D3 is a JavaScript library to visualize data using standard functionalities of the web. It combines SVG, Canvas and HTML to create visualizations that are static or interactive. The current version number of D3js is 5. Up to last year RAWGraph was using D3 version 3 but now the developers updated it to version 5. Therefore some changes have to be made to use the graph of previous versions of RAWGraph with the latest one [Bostock et al. 2011].

5.3 Adaption of Paired Bar Chart from D3v3 to D3v5

In the following section the process of updating the paired bar chart from D3 version 3 to D3 version 5 is described step by step.

5.3.1 D3 scaleBand Function

Since the D3 scaleband function changed over the versions they had to be changed also in the provided paired bar chart file. In Listing 5.1 the conversion from version 3 to version 5 is shown.

5.3.2 D3 scaleLinear Function

Since the D3 scaleLinear function changed over the versions they had to be changed also in the provided paired bar chart file. In Listing 5.2 the conversion from version 3 to version 5 is shown.

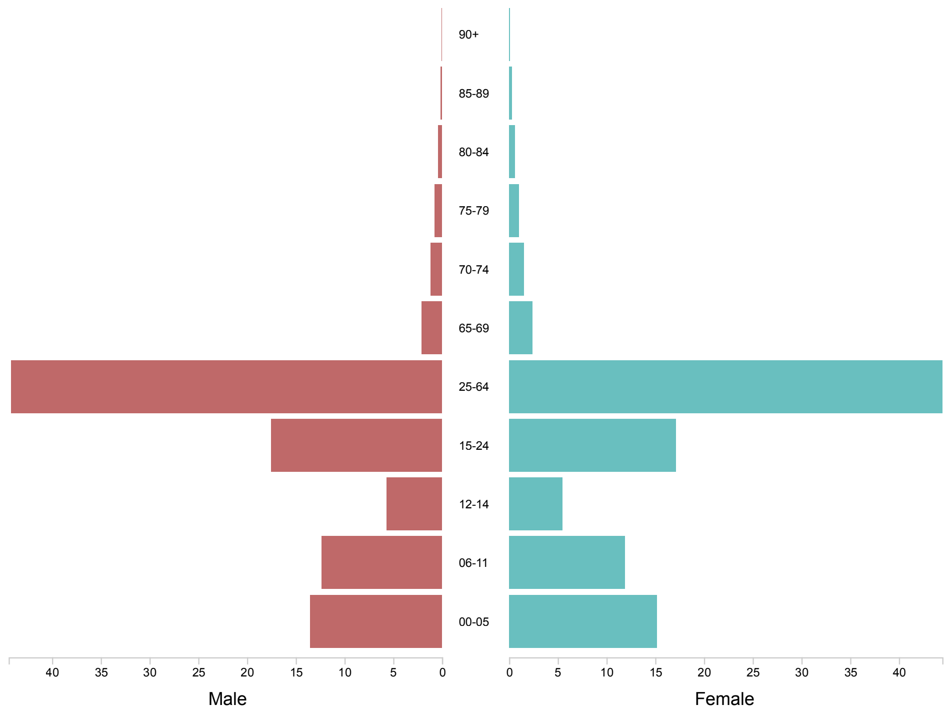


Figure 5.1: Paired bar chart showing population based on age and gender. [Image was exported from RAWGraphs tool]

```
//D3v3
var middleScale = d3.scale.ordinal()
    .rangeBands([0, h], +xPadding(), 0);

//D3v5
var middleScale = d3.scaleBand()
    .rangeRound([0,h])
    .padding(+xPadding());
```

Listing 5.1: This listing shows the conversion of the scaleband function from D3 version 3 to D3 version 5.

```
//D3v3
var rightScale = d3.scale.linear()
    .range([w / 2 + middlePadding + marginLeft, w + marginLeft]);

//D3v5
var rightScale = d3.scaleLinear().range([w / 2 + middlePadding + marginLeft, w +
    marginLeft]);
```

Listing 5.2: This listing shows the conversion of the scaleLinear function from D3 version 3 to D3 version 5.

```
//D3v3
.call(d3.svg.axis().scale(middleScale).orient(orient));

//D3v5
.call(d3.axisLeft(middleScale));
```

Listing 5.3: This listing shows the conversion of the axis function from D3 version 3 to D3 version 5. The orient parameter could be "left", "right", "top", "bottom", but is in version 5 obsolete.

5.3.3 D3 Axis Convention

Another change between the versions is the convention of how axes are generated. In previous versions, the axis function was one function with an orient (orientation) property, which had to be set explicitly. In Version 5 every position of the axis have a different function for each position (top, right, bottom and left) and the orient property is obsolete. In Listing 5.3 the difference can be observed between the versions.

5.3.4 D3 Other Changes

Since in the paired bar chart file only a view changes had to be done, there are a lot more that can occur during inspection of a JavaScript file using D3 version 3. A good starting point to get more into it is the D3 git repository, where a README file of changes exists. The README file can be found here: `https://github.com/d3/d3/blob/master/CHANGES.md`

Chapter 6

Similarity Map

A Similarity Map is a multivariate chart type that looks a lot like a Scatterplot. In contrast to a regular Scatterplot, the position on the 2d grid is not directly taken as two features from the datasets. The position is calculated via an algorithm from more than two features per data point. These algorithms can handle any number of features which could only visualized in a high-dimensional space and transforms and shrinks the large number of features to the most distinctive features without loosing information about the other features so that the dataset can be visualized in a 2d grid. In the following sections the algorithms that were used in this implementation are described.

6.1 Principal Component Analysis

The first algorithm used for the implementation of the Similarity Map is the Principle Component Analysis or PCA for short. PCA is a dimensionality reduction algorithm used in many fields of science including machine learning and data science. PCA is used to reduce a large number of features to a smaller subset that can be handled easier and even visualized. PCA falls under the category of feature extraction algorithms where no information of the original data is lost. This is achieved by combining the original features in a certain way to get new features. Of these new features the least important ones can be dropped to get a smaller number of features. PCA consists of several steps which are explained using some code snippets.

6.1.1 Step 1: Subtract the Mean

For PCA to work properly, the mean has to be subtracted from each feature of your dataset. The mean subtracted is the average across each dimension. This produces a data set whose mean is zero.

6.1.2 Step 2: Calculate the Covariance Matrix

Covariance is measured between two dimensions. The resulting matrix is a square matrix with the number of features as dimensions. Each entry is the calculated covariance between to dimensions. E.g. the entry in row 2, column 3 of the matrix is the covariance calculated between feature 2 and feature 3. The code for this calculation is listed in 6.1.

6.1.3 Step 3: Calculate the Eigenvectors and Eigenvalues of the Covariance Matrix

The next step is to calculate the eigenvalues and eigenvectors of the covariance matrix. The number of eigenvalues and eigenvectors corresponds to the number of features of the dataset. The important property of these to values is that the better an eigenvector fits the datapoints the larger the corresponding eigenvalue is. In this implementation the eigenvalues and eigenvectors are calculated with the help of a mathematics library for JavaScript.

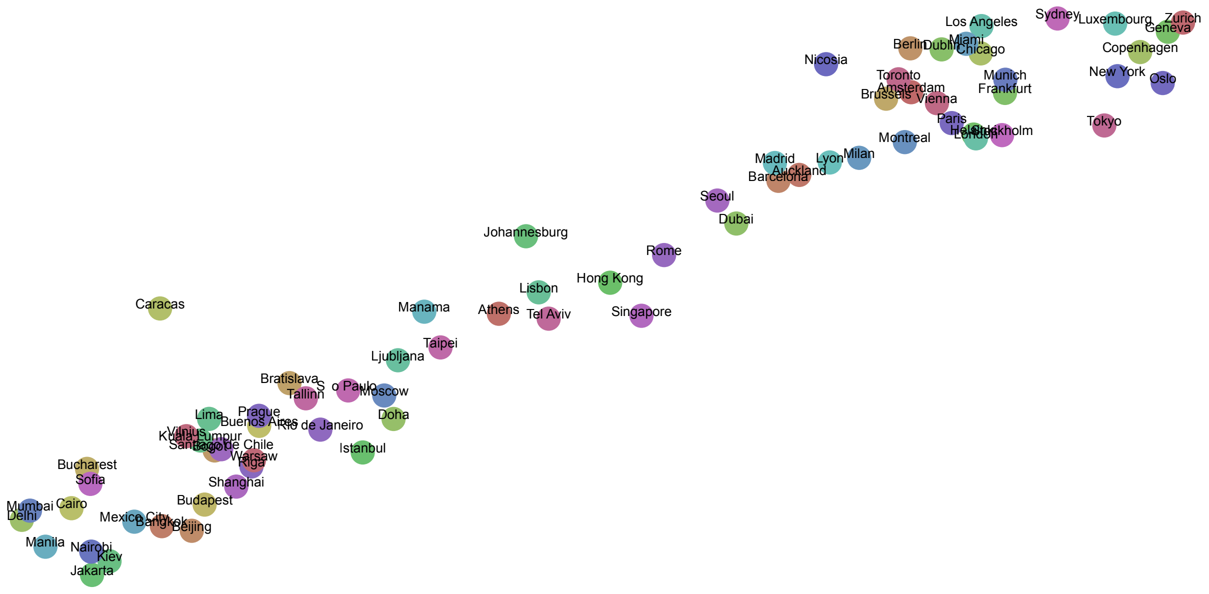


Figure 6.1: Similarity Map with cities as datapoints and various financial properties of these cities as features. [Image was exported from RAWGraphs tool]

```
function calculateCovariance(adjustedMatrix, i, j) {
    var covariance = 0;
    var n = adjustedMatrix.length;
    adjustedMatrix.forEach(function(row){
        if(Array.isArray(row)) {
           covariance += row[i] * row[j];
        }
    });
    covariance = covariance / (n - 1);
    return covariance;
}

function calculateCovarianceMatrix(adjustedMatrix){
    var feature_amount = adjustedMatrix[0].length;
    var covarianceMatrix = [];
    for(var n = 0; n < feature_amount; n++) {
        covarianceMatrix[n] = new Array(feature_amount);
    }
    for(var i = 0; i < feature_amount; i++){
        for(var j = 0; j < feature_amount; j++) {
            covarianceMatrix[i][j] = calculateCovariance(adjustedMatrix,i,j);
        }
    }
    return covarianceMatrix;
}
```

Listing 6.1: This listing shows the calculation of the covariance matrix.

```
function calculateFeatureVector(eigenvectors, eigenvalues) {
  var sorted_array = sortArray(eigenvalues);
  var feature_vector = Array(2);

  for(var i = 0; i < 2; i++) {
      feature_vector[i] = eigenvectors[eigenvalues.indexOf(sorted_array[i])]
  }
  return feature_vector
}
```

Listing 6.2: This listing shows sorting of the eigenvalues and the formation of the feature vector.

6.1.4 Step 4: Choosing Principal Components

Now that everything that is necessary is calculated the principal components can be chosen. For this the eigenvalues have to be sorted descendigly. The principal components are the features with the largest eigenvalues. In this step the features with smaller eigenvalues can be dropped until only the desired number of features remains. In this implementation the dimension is limited to 2 as seen in Listing 6.2.

6.1.5 Step 5: Calculating the New Dataset

The final step of the PCA is the calculation of the new dataset so that it can be visualized in the desired way. For this the matrix of the chosen eigenvectors is multiplied with the original mean-adjusted data. In Listing 6.3 all of the previous steps are put together.

6.1.6 Remarks

PCA is a tried a tested dimensionality reduction algorithm that is fast, simple to use, and intuitive. This makes it a good candidate for this usecase as long as the dataset allows it. PCA is just a linear projection and therefore does not work very good on non-linear dependencies.

6.2 Force-Directed Placement

The second algorithm available in this implementation is called Force-Directed Placement or FDP for short. As the name suggests this algorithm models a simple physics model where forces of attraction pull the datapoints closer together or push the further apart depending on their similarity. This is done iteratively until a stable position for each datapoint is reached.

6.2.1 Step 1: Initialization

The original FDP algorithm takes random initial positions which results in a different layout after each run. For the usecase of generating visualizations random results are not good, because even a change in the size of plot would result in a totally new visualization. To fix this problem this implementation calculates the initial position with the PCA so that every run results in the same visualization.

6.2.2 Step 2: Similarity Calculation

In this step the similarity is calculated over all features for each pairwise combination of datapoints. The results are stored in a matrix for further calculations. Listing 6.4 shows that this implementation uses the Cosine Similarity.

```
function performPCA(data) {
    var copy_data = new Array(data.length);
    copy_data = data.slice();
    var meanMatrix = calculateMeanMatrix(extractFeatureMatrix(copy_data));
    var adjustedMatrix = calculateAdjustedMatrix(meanMatrix,extractFeatureMatrix(
        copy_data));
    var covarianceMatrix = calculateCovarianceMatrix(adjustedMatrix);
    var eigenvalues = calculateEigenvalues(covarianceMatrix);
    var eigenvectors = calculateEigenvectors(covarianceMatrix);
    var featurevector = calculateFeatureVector(eigenvectors, eigenvalues);
    return featurevector;
}

function pca(data) {
    var featurevector = performPCA(data);
    data.forEach(function(d) {

        var value = calculateMatrixMultiplication(d.features, featurevector);

        d.x = Number.isNaN(value[0]) ? rand(margin(), width() - margin()) : value
            [0];
        d.y = Number.isNaN(value[1]) ? rand(margin(), height() - margin()) : value
            [1];

    });
    return data;
}
```

Listing 6.3: This listing shows the calculation of the final dataset. Some Errorhandling is done in case something went wrong.

```
function similarity(vectorA, vectorB) {
    var dotProduct = 0.0;
    var normA = 0.0;
    var normB = 0.0;
    for (var i = 0; i < vectorA.length; i++) {
        dotProduct += vectorA[i] * vectorB[i];
        normA += Math.pow(vectorA[i], 2);
        normB += Math.pow(vectorB[i], 2);
    }
    return dotProduct / (Math.sqrt(normA) * Math.sqrt(normB));
}

function calculateSimilarityMatrix(featureMatrix) {
    var similarityMatrix = [];
    for(var i = 0; i < featureMatrix.length; i++) {
        similarityMatrix[i] = new Array(featureMatrix.length);
    }
    for(var k = 0; k < featureMatrix.length; k++) {
        for(var j = 0; j < featureMatrix.length; j++) {
            similarityMatrix[k][j] = similarity(featureMatrix[k], featureMatrix[j]);
        }
    }
    return similarityMatrix;
}
```

Listing 6.4: This listing shows the calculation of the similarity of the datapoints with all features considered. The result is stored in a matrix.

```
function calculateForce(i, data, similarityMatrix) {
    var force = {x: 0, y: 0};
    for (var j = 0; j < data.length; j++) {
        if (i != j) {
            var direction = {
                x: data[i].x - data[j].x,
                y: data[i].y - data[j].y
            };
            var normalizedDirection = normalize(direction);

            force.x += normalizedDirection.x * similarityMatrix[i][j];
            force.y += normalizedDirection.y * similarityMatrix[i][j];
        }
    }
    return force;
}
```

Listing 6.5: This listing shows the calculation of the force for each datapoint.

```
for (var k = 0; k < nIter(); k++) {
    for (var i = 0; i < data.length; i++) {
        var force = calculateForce(i, data, similarityMatrix);
        data[i].x += force.x;
        data[i].y += force.y;
    }
}
}
```

Listing 6.6: This listing shows the loop where the forces are added to the current position for each datapoint.

6.2.3 Step 3: Force-Directed Placement

in the most important step the force for each datapoint is initialized according to the similarity matrix which can be seen in Listing 6.5. Then this force is added to the current position of the datapoint to get a new position. This process is iterated until the positions are stable which means that they do not change anymore or until a fixed number of iterations is reached.

6.2.4 Remarks

The FDP algorithm is another fairly easy approach to model multi-dimensional data in on a 2d grid. The underlying physical model makes this algorithm intuitive and to some extend predictable. One benfit of this algorithm the iterative nature which makes it useful for interactive visualizations so that the user can observe the change over time. Unfortunately due to the static nature of RAW this aspect is lost in this usecase. The initialization of the algorithm with deterministically calculated points via PCA makes it more useful in combination with RAW because the redrawing of the visualization due to the implementation of RAW results in the same visualization everytime.

```
var L2 = function(x1, x2) {
  var D = x1.length;
  var d = 0;
  for(var i=0;i<D;i++) {
    var x1i = x1[i];
    var x2i = x2[i];
    d += (x1i-x2i)*(x1i-x2i);
  }
  return d;
}

var xtod = function(X) {
  var N = X.length;
  var dist = zeros(N * N);
  for(var i=0;i<N;i++) {
    for(var j=i+1;j<N;j++) {
      var d = L2(X[i], X[j]);
      dist[i*N+j] = d;
      dist[j*N+i] = d;
    }
  }
  return dist;
}
```

Listing 6.7: This listing shows the calculation of the pairwise distances used in the preparation for the t-SNE algorithm. Taken from [Karpathy 2016].

6.3 t-distributed Stochastic Neighbor Embedding

The third algorithm that was used in this implementation is called t-distributed Stochastic Neighbor Embedding or t-SNE for short. t-SNE is a dimensionality reduction algorithm similar to PCA, but the t-SNE algorithm has no learning capabilities. The t-SNE algorithm is particularly well suited for the visualization of high-dimensional datasets.

6.3.1 Step 1: Initialization

The original t-SNE algorithm takes random initial positions and would be therefore nondeterministic but as with the FPD algorithm this implementation calculates the initial positions via PCA so that indepedent runs result in the same visualization.

6.3.2 Step 2: Calculate Pairwise Distances

The calculation of pairwise distances is done in the high-dimensional space. In the library used in this implementation the L2 norm is used for the distance calculation.

6.3.3 Step 3: Calculate Coniditional Probabilities

In this step the previously calculated pairwise distances are transformed into pairwise probabilities. Basically the probability is calculated that any datapoint picks any other datapoint as its neighbour - the smaller the distance, the higher the probability gets. This is done with a normal or gaussian distribution put on top of every point and then calculate the probability of every other datapoint based on their pairwise distance. In this step some user interaction takes place. The user can change the variance of the distribution, but not directly because the data is in most cases not evenly distributed but has some clusters. For these clusters a small variance works best, but for some datapoints further apart this variance

```javascript
function tsne(data) {
    //create options object
    var opt = {}
    opt.perplexity = perplexity();
    opt.dim = 2;
    opt.epsilon = learningRate();
    var model = new tsnejs.tSNE(opt);

    //run PCA on data to get intial positions
    var pca_data = pca(data);
    var raw_data = Array(pca_data.length);
    for (var f = 0; f < pca_data.length; f++) {
      raw_data[f] = pca_data[f].features;
    }
    //put the positions into the algorithm
    model.initDataRaw(raw_data);
    //run the algorithm
    for (var k = 0; k < nIter(); k++) {
      model.step();
    }
    //calculate final position on 2d grid
    var positions = model.getSolution();
    for (var i = 0; i < data.length; i++) {
        data[i].x = positions[i][0];
        data[i].y = positions[i][1];
    }
    return data;
}
```

Listing 6.8: This listing shows the usage of the t-SNE implementation of the library from [Karpathy 2016].

would be to small. So the user can set a value called perplexity which is linked to the entropy. With this parameter the algorithm itself calculates the variance for each datapoint automatically so that the entropy is the same for all datapoints the same indepedent of the closeness to other points.

6.3.4 Step 4: Iteratively Map High Dimensionality Probabilities to Lower Dimensionality Probabilities

The last step is of t-SNE is an iterative process. In this step the dimensionality reduction takes place. The algorithm tries to find a t-distribution in a lower dimensional space that is similar to the original distribution in the higher dimensional space. To compare the probability distributions the Kullback-Leibler-Divergence is used. The iterative part of the t-SNE algorithm is basically the optimization of this cost function. For this a gradient descent is used.

6.3.5 Running t-SNE

Listing 6.8 shows how to use the t-SNE implementation provided by the library from [Karpathy 2016]. Nearly all of the actual computations are still in the library only the initialization and the final calculation of the positions on the 2d grid is shown.

6.3.6 Remarks

The t-SNE algorithm is very popular even though it is relatively young compared to the other two algorithms that where used in this implementation. This is because it is very flexible due to the non-linear nature of the algorithm. Thereore it can find structure in multi-dimensional data where other dimensionality reduction algorithms like PCA cannot. On the other hand this flexibility can be confusing to users because under the hood the algorithm makes a lot of adjustments to itself to deliver clean results. The random initialization of the datapoints is something that makes this algorithm somewhat unreliable so a fixed initialization with e.g. PCA is recommended. This short overview is neither complete nor detailed enough to accurately reflect the t-SNE algorithm. There are several tutorial available online like the one from [Strayer 2018] or the original paper from [L.J.P. van der Maaten 2008].

Chapter 7

Conclusion

RAWGraphs is a powerful tool to create visualizations. A large benefit of RAWGraphs is that it is fairly straight forward in its workflow, and nearly no prior knowledge is needed by the user. Another benefit of RAWGraphs is that it is open-source, can be self-hosted and extended with custom chart types. Furthermore it offers some features to tweak the visualizations and exporting them aswell in various formats. A drawback of this tool is that it only provides static visualizations.

Someone might be surprised to find out that RAWGraphs did not have such a basic chart like the line graph already implemented. Fortunately with such a simple chart there were a lot of possibilites to add extra features to it that do not come with the original version of RAWGraphs.

Paired bar charts are charts to visually compare 2 groups belong a common axis, which is a feature they share. A good example is the age in a population dataset and the groups are male and female. In the project an already implemented paired bar chart was converted from D3v3 to D3v5, since there were some minor changes in the function calls between the versions and RAWGraph changed the used version from 3 to 5.

The similarity map was the most complex chart that was implemented in this project, which was challenging but also interesting to get to know some algorithms that are used with it. There are some fundamental differencies in the algorithms which makes them unique in the way that they handle the data. Some are rather simple like the PCA or FDP algorithms which is a good point to start and also due to some changes to the FDP algorithm deliver consistent results. The t-SNE algorithm was a lot more complex to understand but can also handle more complex data. The iterative algorithms FDP and t-SNE are somewhat limited in their capabilities due to the static nature of RAWGraphs. These algorithms have more potential than what is used in this implementation.

All in all RAWGraphs is an easy but powerful tool for creating static information visualizations with very little effort of the user.

Bibliography

Bostock, Michael, Vadim Ogievetsky, and Jeffrey Heer [2011]. *D3 Data-Driven Documents*. IEEE Transactions on Visualization and Computer Graphics 17.12 (Dec 2011), pages 2301–2309. ISSN 1077-2626. doi:10.1109/TVCG.2011.185. `http://dx.doi.org/10.1109/TVCG.2011.185` (cited on page 15).

Karpathy, Andrej [2016]. *tsnejs*. 2016. `github.com/karpathy/tsnejs` (cited on pages 24–25).

L.J.P. van der Maaten, G.E. Hinton [2008]. *Visualizing High-Dimensional Data Using t-SNE.* Journal of Machine Learning Research (2008) (cited on page 26).

Lea, Gord [2019]. *D3 v5 Line Chart*. 2019. `bl.ocks.org/gordlea/27370d1eea8464b04538e6d8ced39e89` (cited on page 9).

Mauri, Michele, Tommaso Elli, Giorgio Caviglia, Giorgio Uboldi, and Matteo Azzi [2017a]. *RAWGraphs*. [Online; accessed 30-June-2019]. 2017. `https://rawgraphs.io` (cited on pages 4–5).

Mauri, Michele, Tommaso Elli, Giorgio Caviglia, Giorgio Uboldi, and Matteo Azzi [2017b]. *RAWGraphs: A Visualisation Platform to Create Open Outputs*. Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter. CHItaly '17. Cagliari, Italy: ACM, 2017, 28:1–28:5. ISBN 978-1-4503-5237-6. doi:10.1145/3125571.3125585. `http://doi.acm.org/10.1145/3125571.3125585` (cited on pages 1, 3).

Strayer, Nick [2018]. *t-SNE explained in plain javascript*. 2018. `observablehq.com/@nstrayer/t-sne-explained-in-plain-javascript` (cited on page 26).