HPDA Project

Elia Vaglietti **023279125A**

November 20, 2024

1 Data inspection and management

The first thing to do when dealing with a dataset big enough, is to put it in a dataframe from the `pandas` library and get some fast insights (`df.info()`).

```
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Date                 8760 non-null   object
 1   RentedBikeCount      8760 non-null   int64
 2   Hour                 8760 non-null   int64
 3   Temperature          8760 non-null   float64
 4   Humidity             8760 non-null   int64
 5   WindSpeed            8760 non-null   float64
 6   Visibility           8760 non-null   int64
 7   DewPointTemperature  8760 non-null   float64
 8   SolarRadiation       8760 non-null   float64
 9   Rainfall             8760 non-null   float64
 10  Snowfall             8760 non-null   float64
 11  Seasons              8760 non-null   object
 12  Holiday              8760 non-null   object
 13  FunctioningDay       8760 non-null   object
dtypes: float64(6), int64(4), object(4)
memory usage: 958.3+ KB
```

In the *SeoulBikeData* data set there are 9 quantitative classes, 1 ordinal and 3 nominal ones.
The `Data` entry is a string which has to be carefully managed. Not only because it's the only ordinal variable, also because if leaved as found in the data set, it would have been converted to date and would have been represented as amount of seconds from a specific date in the '70s. To handle it, during the loading phase, I create a new *Date* object that takes the date string appropriately parsed. Doing so, it will be easier to visualize the real data as explained in the section 2. Apart of this, no other modifications have been made to the data set in the loading phase.

As expected, the `Seasons` categorical column can be either *Spring*, *Summer*, *Autumn* or *Winter*.
`Holiday` can have the value *Holiday* or *No Holiday*.
`FunctioningDay` can be *Yes* or *No*
All the other columns were made of simple `int` or `float` numbers.

2 Design visualization

To design a visualization for a dataset with so many attributes I had to make a choice about which ones I wanted to see first and then, in case, choose other attributes to display. Since one of the visualization was already set (Scatter-plot), I only could choose about the second one. I chose the parallel coordinates because

it allows to get quickly an idea about the distribution of a data category against another. This visualization type, together with the Scatter-plot can give a good insight about the data distributions and correlations among the data set. Since all these categories are quantitative (except `date` which is ordinal), therefore they are represented with the position.
The nominal categories are encoded in the hue of the dots (scatter-plot) or the lines (parallel coordinates), as it is one of the best variable visualization for this kind of data.

In conclusion of this section, what I wanted is to have a visualization of four quantitative/ordinal variables, respectively on both of the two graphics axes and one nominal variable encoded in the color.

3 Component implementation

Every component has the D3 class that implements what is done in the graphic and the react component that is called in the `index.js` file and manages the communication/interaction phase.

The color of the nominal variables is chosen by using the Color Brewer website, that allows to choose wisely the hue for a color-map. This type of data doesn't have a legend on screen, so it follows now the legend to know the colors.

Functioning day	
Yes	
No	

Holiday	
Holiday	
No Holiday	

Seasons	
Spring	
Summer	
Autumn	
Winter	

Scatter-plot Quantitative data is easy to understand when encoded with the position. For this reason this plot fits very well when dealing with a data set composed for its major part by quantitative and ordinal data. In this case the scatter-plot is 2D, so only 2 of this kind of attributes can be seen at a time. Moreover, a 3rd encoding is done using the color, as explained before.

To select a portion of the data-set I've used a Brush 2D, that draws a rectangular region. This can be resized and moved. Every time a brush is used, the selected part of the elements is highlighted (opacity increased and stroke-width increased), while the color of the non-selected part is set to gray, in both graphics. This is done by storing the data in the `SelectedItemSlice.js` *redux* slice, through a dispatch of the function that sets the slice. When updating the plot, the data that are in this selected items set are modified accordingly. The visualization is updated every time that the set is updated. Therefore, the render is updated also when the Parallel Coordinates brushing is used.

Parallel coordinates As in the scatter-plot, the parallel coordinates drawing allows visualization of two quantitative or ordinal attributes and one nominal attribute. The nominal attribute is the same as in the scatter-plot. This consistency helps in identifying clusters of data based on color in both visualizations at a glance. The difference in this plot is that the lines allow to understand in a easier way the distribution of an attribute against one other.

The selection in this case is done using two 1D brushes on the Y-axis. When both are active, the application computes the intersection of the 2 sets. This is forwarded to the `SelectedItemSlice.js` slice as before. The selected lines are highlighted in a similar way as for the scatter-plot.

Pros and Cons Follows a list of pros and cons of my design:

Pros:

- **Interactive brushing:** The brushing allows to have interactive feedback over the data-set. This is the best way to visualize high-dimensional data, specially combined with the Parallel coordinates graphics. Moreover, for the Parallel Coordinates, the two independent brushes allows to select only those data that fits some specific attributes range, which helps in selecting only the data that we're looking for.
- **Best data encoding for these attributes:** The best way to understand quantitative/ordinal data and nominal data without interaction is to use respectively positions and colors, as done here.
- **Distribution over time:** Parallel coordinates drawings are particularly efficient when dealing with **date** attributes. By plotting the latter on one axis and another one on the remaining axis, it's easy to grasp a sketch of the distribution of that variable over time.
- **Easy implementation strategy:** D3 + React + Redux is a good strategy to implement this kind of interactive visualization in which more components (React) can "talk" to each other (Redux) and visualizing fancy graphics (D3). Moreover, this MVC structure is easy to maintain and expand in the future.

Cons:

- **Visual clutter:** Many data means many lines and many dots. When the render plot is not big enough or the data are too many, it becomes difficult to understand the details of the distributions.
- **Limited amount of variables shown:** The scatter-plot 2D alone can show 3 out of 13 variables in the same time, same for the Parallel Coordinates. In total, you can look at 5 variables out of 13, which is not optimal when dealing with a high dimensional data-set.
- **Performance issues:** Many high-dimensional data are difficult to manage on a single machine. That's why an optimized rendering must be done, without doing unnecessary work.
- **Coloring issues:** When not selected, the items of the scatter-plot have opacity different from 1. This implies that, using different colors, there are dots with a mixture of basic hues where they overlap. This can cause problems in understanding the nominal attribute.

4 Conclusions

In conclusion, in this project we implemented a scatter-plot and a parallel coordinates visualizations to analyze a high-dimensional dataset. Using D3.js, React, and Redux, we created an interactive and insightful experience.

The scatter-plot visualizes two quantitative or ordinal attributes, with a third nominal attribute encoded by color. The 2D brush interaction allows users to select and highlight specific data subsets, aiding in pattern and correlation identification.

Parallel coordinates complement the scatter-plot by visualizing multiple attributes simultaneously, effectively showing data distribution and clusters. On the parallel coordinates drawings, the 1D brushes on the Y-axis enable precise data range selection.

Despite some limitations like visual clutter and performance issues with large datasets, the combination of these techniques provides a comprehensive data visualization approach. Integration with Redux ensures consistent state management and accessibility across components. Overall, these visualizations offer a robust, maintainable, and scalable solution for exploring complex datasets.