

iBankEx: An Interactive Visualisation Tool for the Exploration of Bank Transaction Data

by Wentao Mao, MSc

Submitted to The University of Nottingham
in September 2023

in partial fulfilment of the conditions for the award of the degree of
Master of Science in Computer Science

I declare that this dissertation is all my own work, except as indicated in the text:

Wentao Mao – 22/09/2023

Supervisor: Dr Robert Laramee

Abstract

With the growth of data, finance visualisation has recently gained tremendous popularity. However, the accessibility of financial data is poor, limiting the research in this area. Firat et al. (2023) recently proposed the first open dataset of retail bank transaction data, potentially leading to many future studies. Although previous financial visual analytic tools have worked on transaction data, they did not use open retail data. This thesis proposes an interactive web-based visualisation tool to ease the exploration of this publicly accessible transaction dataset. Multiple coordinated views help show the link between views effectively consisting of a transaction amount view with an overview of the temporal patterns of transactions, a cluster view providing cluster information about transactions, a superpositioned calendar view with bar glyphs, pie glyphs, polar area glyphs, and star glyphs demonstrating the monthly pattern of the transactions, and table views with details of the selected transaction. The novelty of the project is to apply clustering algorithms to the first bank transaction dataset with the superpositioned calendar view consisting of different types of glyphs. Case study and performance test validated iBankEx.

Keywords: Finance Visualisation, Information Visualisation, Multiple Coordinated Views, Glyphs, Interactive Visualisation

Acknowledgements

I want to say thanks to my supervisor, professor Robert Laramée. He has been patiently helping with me, providing suggestions the project, and giving me learning materials.

I want to thank my family for their supports.

Thanks for David Bowen, Ming Zhang and Ameya Pimpley as they proofread some part of my dissertation and demo video.

Contents

1 Motivation	7
2 Related Work	8
2.1 Literature Scope	8
2.2 Search Methodology.....	8
2.3 Literature Classification	9
2.4 Finance Visualisation.....	11
2.4.1 Company Information	11
2.4.2 Stock.....	12
2.4.3 Fund.....	16
2.4.4 Economic Indicators.....	17
2.4.5 Transaction	19
2.5 Visualisation Surveys	27
2.6 Visualisation Techniques	29
3 Data Characteristics.....	33
4 Project Specification.....	36
4.1 Feature Specification.....	36
4.1.1 Must-have Features	36
4.1.2 Optional Features.....	36
4.2 Technology Choices	37
4.2.1 Programming Language	37
4.2.2 Libraries.....	38
4.2.3 Rendering Mechanism.....	44
4.2.4 Tools and the Others	45
5 Project Plan and Timetable.....	47
6 Project Design	48
6.1 Visualisation Design.....	48
6.1.1 Transaction Amount View	49
6.1.2 Cluster View	50
6.1.3 Calendar View	51
6.1.4 Table View	51
6.2 System Overview	52
6.2.1 API Server Design.....	53
6.2.2 Web-based UI Design.....	54
7 Implementation.....	55
7.1 Basic Features	56
7.1.1 Web-based UI	56

7.1.2 Calendar View with Year Selection	58
7.1.3 Basic Bar and Pie Glyph.....	59
7.1.4 Calendar View Glyph Option.....	59
7.1.5 Cluster View with Logarithmic Scale and Sliders.....	60
7.1.6 Transaction Amount View with Logarithmic Scale and Sliders	61
7.1.7 Table View for Detailed Data.....	61
7.1.8 Brush and linking	62
7.1.9 Consistent Axis Style and Colour.....	63
7.1.10 Calendar View Detail-on-Demand	63
7.1.11 Transaction Clustering.....	64
7.1.12 Transaction Description Grouping	65
7.1.13 Colour Legends and Highlighting	67
7.1.14 Context and Focus	68
7.1.15 Transaction Frequency	68
7.2 Optional Features	69
7.2.1 Bar Glyph Reordering	69
7.2.2 Bar Glyph Bandwidth and Height Options.....	70
7.2.3 Pie Glyph Radius Option.....	71
7.2.4 Polar Area Glyph.....	72
7.2.5 Polar Area Glyph Radius Options	72
7.2.6 Star Glyphs	73
7.2.7 Star Glyph Radius Options	73
7.2.8 “Superpositioned” Calendar View.....	73
7.2.9 Calendar View Expanding.....	74
7.2.10 Transaction Amount View Axis Swapping.....	75
7.2.11 Transaction Amount View Expanding	77
7.2.12 Cluster View Expanding.....	78
7.2.13 Cluster View Colour Option.....	78
7.2.14 Cluster View Axis Options and Swapping.....	79
7.2.15 Table View Page Option	80
7.2.16 Table View Clear-All Button	81
7.2.17 Table View Sorting.....	82
7.2.18 Table View Synchronised Row Colour	82
7.2.19 Colour Legend Sorting	84
7.2.20 Border Colour for Brusher and Calendar View.....	87
7.2.21 Table View Consistent Radio Button Colour	88
7.2.22 Transaction Description Grouping Advanced Mode.....	89

8 Testing and Evaluation	90
8.1 Results	91
8.1.1 Case Study: Data Exploration 1 – paycheck	91
8.1.2 Data Idiosyncrasies.....	94
8.2 Performance Test	95
9 Conclusion.....	97
10 Future Work.....	97
Reference List.....	98
Appendix 1. Meeting Minutes.....	106
Appendix 2. Performance Test	106
Appendix 3. Python Documentation	106
Appendix 4. Source Code.....	107

1 Motivation

In recent years, transaction data has been growing rapidly, and analysing it can help with tasks like loan decision-making, criminal analysis, etc. (Firat et al., 2023). One of the challenges of financial data visualisation is data accessibility (Ko et al., 2016). Firat et al. (2023) recently tackled this problem by proposing the MoneyVis dataset, which is the first open retail bank transaction dataset of a single account. This may lead to much future research. Thus, visualisation may be needed to explore the dataset. However, the authors only provided basic visualisation like a stacked bar chart. Many previous visual analytic tools help explore financial data, and more specifically, many studies (e.g., Leite et al. (2020) and Chang et al. (2007)) visualised finance transaction data in a way that supports interactivity, using multiple coordinated views (i.e., views that update automatically when the user interact with another view), tables, glyphs, and clustering. However, the transaction data used in their systems is mostly from multiple accounts and are not publicly available.

The aim of the present work is to provide a web-based interactive visual analysis tool, iBankEx, to facilitate the exploration of MoneyVis. This project can help future researchers and analysts understand and work on this dataset or similar dataset and give a possible starting point for financial visualisation app developers. The contribution is 1) an interactive visualisation tool with Web-Based UI (<http://moneyvis.wentaom.com/dashboard>), 2) a description of the implementation of iBankEx and the source code (see section 7), 3) a novel superpositioned calendar view with bar glyphs, pie glyphs, polar area glyphs and star glyphs, 4) clustered transactions and clustered transaction descriptions that provide more insight than the initial dataset, and 5) Case study and performance test that evaluate iBankEx. 6) New finding of the idiosyncrasies of the MoneyVis.

The key challenges were to 1) visualise the large amount of data records, and 2) manage the large amount of code of the software. To tackle the first challenge, multiple coordinated views were utilised. Scatter plots and calendar view provide an overview of transactions while tooltips in the calendar view and tables are used for showing details. Sliders are provided to zoom into the scatter plots and views expansion is also supported to show data clearly. The second challenge was solved by carefully choosing programming languages and libraries. For example, TypeScript was used for checking type errors, and React.js was used for making reusable interactive component. Codes for loading and clustering data were separated from the browser and put into the Python API server to make them easier to change.

The thesis is structured based on the suggestion from Bob's Project Guideline (Laramee, 2021b). Section 2 provides an overview of previous research, presenting the literature scope, search methodology, classification of finance visualisation literature, and summarising the finance visualisation literature, the visualisation survey literature, and the literature on involved visualisation techniques. Section 3 describes the characteristics of the data. Section 4 provides a list of features of iBankEx and explains the technology choices. Section 5 explains the timeline of the project. Section 6 explains iBankEx's visual design and software design, providing a system overview diagram to understand the logical model of the system, a class hierarchy diagram for the API server, and a components hierarchy diagram for the user interface. Section 7 explains the implementation of the features to help the reader understand how to use the system and how to re-implement the system. Section 8 presents the new finding, and case study and performance test that evaluated iBankEx. Section 9 and 10 provides the conclusion and future works.

2 Related Work

This section introduces the visualisation surveys, and the related literature on financial visualisation and visualisation techniques. Section 2.1 introduces the scope of the literature review. Section 2.2 explains how the literature was found. Section 2.3 provides a classification for the financial visualisation research. Section 2.4, 2.5, and 2.6 summarises the financial visualisation research, visualisation surveys and related visualisation techniques respectively.

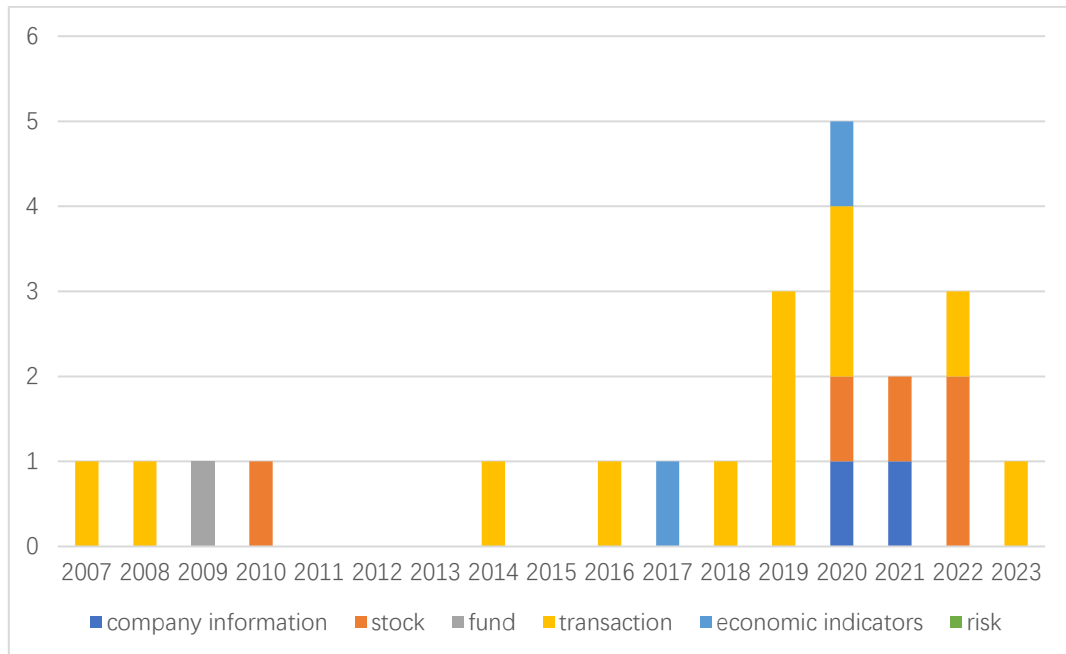


Figure 2.1 Metadata for the literatures.

2.1 Literature Scope

The area of the literature to review were chosen with the advice of the project supervisor (a.k.a. Professor Robert Laramée), a visualisation researcher. This is appropriate due to the subjectivity of selecting the scope (Laramée, 2010, 2021b) and the supervisor's experience.

The first area is “financial visualisation” as it shares a similar concept (i.e., financial visualisation) with the iBankEx and has similar data characteristics (i.e., temporal data with money value) with the transaction data used in iBankEx. The second area is visualisation survey papers as they “can help understand the current landscape of information visualisation” (McNabb & Laramée, 2017, p. 589) and help find related software and literature. The third area is visualisation technique research, including glyphs, multiple coordinated views, and clustering, which help design the iBankEx more appropriately.

2.2 Search Methodology

The search methodology for financial visualisation literature in section 2.4 was primarily based on McNabb & Laramée (2017), including “linear search” (i.e., flicking through the paper titles in the journals and conferences and read their abstracts) and “relation-search” (i.e., searching the reference list and “cited by” list), and on McNabb & Laramée (2019a). The methodologies for the survey in section 2.5 and the visualisation techniques literature in section 2.6 were supported by supervisor suggestions.

The starting points for searching financial visualisation are the financial visualisation survey (Ko et al., 2016) and MoneyVis, which provided the dataset for the present project and the

knowledge of previous transaction data visualisation research (Firat et al., 2023). Additionally, Laramee (2010) and McNabb & Laramee (2019a) suggested some journals and conferences for searching visualisation literature.

The linear search is conducted by learning the financial visualisation papers that appear in the “previous ten years of (1) the IEEE Visualization conference proceedings, (2) the IEEE Transactions on Visualization and Computer Graphics (IEEE TVCG) journal papers and (3) the EuroVis conference proceedings (called VisSym until 2005)”(Laramee, 2010, p. 2365). To choose the relevant literature, paper titles were checked first, and the abstract of the papers whose titles might relate to financial visualisation were read. As a result, six financial visualisation papers were found, see Table 2.1.1.

The relation search consists of two parts. The first is learning the papers cited by MoneyVis (Firat et al., 2023), leading to nine related financial visualisation studies, see Table 2.1.1. The second part (Mao, 2023) of the relation search is learning the financial visualisation papers, which cited Ko et al. (2016) or Lei & Zhang (2010). The following journals and conferences are included in the second part: Computer Graphics Forum, IEEE Transactions on Visualization and Computer Graphics, Information Visualization, International Symposium on Visual Information Communication, Visual Informatics, International Conference Information Visualisation, International Journal of Accounting Information Systems. This leads to ten pieces of literature, of which three were also cited by MoneyVis, see Table 2.1.1.

2.3 Literature Classification

To organise the studies well, this section categorises the financial visualisation literature by 1-m mapping (McNabb & Laramee, 2019a), giving each literature more than one dimensions (i.e., data source and visualisation techniques). These are the appropriate dimensions as the papers can be easily classified (McNabb & Laramee, 2019a) based on the output of visualisation paper reading techniques which extract the data characteristics (Laramee, 2011) and can provide an overview their visualisation techniques.

The data source dimension includes “stocks, funds, economic indicators, transactions, risk, and company information” (Ko et al., 2016, p. 603).

Stocks and fund data are the data for stocks and funds. **Economic indicators** are the statistical information which impacts financial markets, such as political and company profit news and national financial data. **Company information** category covers all the information related to a business (e.g., profit, financial statements, sales, marketing data). “**Transaction** category considers all the data generated by transactions among different subjects (e.g., bank customers, companies, and countries).” (Ko et al., 2016, p. 604). **Risk** are the systems helping analyse investment risk or systemic risk in financial institutions.

The visual technique dimension is also used by Ko et al. (2016). However, their didn’t cover all the techniques in the searched literature. Thus, the visual technique category of the present literature review is derived from the searched literature.

As Table 2.1.1 illustrates, most searched literature worked on transaction data, followed by those using stock data. Only a little searched literature used economic indicators data and fund data, and fewer performed on company information data. There is no searched literature working on risk data.

Table 2.1.1. Literature Classification. Blue means the linear search found the paper. Orange or “*” indicates MoneyVis cited them. Green means they are related to Mao (2023), Ko et al. (2016), or Lei & Zhang (2010). “!” means the research is not a visualisation system but is an improvement related to financial visualisation. The classifications were proposed by Ko et al. (2016).

Search source		Leite et al. (2016)	Ruppert et al. (2017)	Arleo et al. (2019)	Q. Q. Liu et al. (2020)	Tsang et al. (2020)	Q. Q. Liu et al. (2021)	Chang et al. (2007)	Chang et al. (2008)	Rudolph et al. (2009)	Didimo et al. (2014)	Leite et al. (2018)	Leite et al. (2020)	Lei & Zhang (2010)	Singh & Best (2019)	Yue et al. (2019)	Maças et al. (2020)	Leite, Arleo, et al. (2020)	Yue et al. (2021)	He & Li (2022)	Guo et al. (2022)	Maças et al. (2022)	Arleo et al. (2023)	Total	
Data Type	Company Information				✓		✓																	2	
	Stock					✓							✓						✓	✓	✓				5
	Fund									✓															1
	Transaction	✓		✓				✓	✓		✓	✓	✓		✓	✓	✓					✓	✓	12	
	Econ Indicators		✓																✓						2
	Risk																								0
	Visualisation Techniques	Bar charts	✓	✓	✓		✓	✓	✓				✓				✓	✓		✓		✓	✓		12
Graph				✓				✓	✓		✓		✓	✓	✓	✓	✓	✓						9	
Table					✓		✓	✓	✓		✓	✓						✓	✓				✓	8	
PCP		✓			✓	✓	✓					✓				✓		✓	✓					7	
Pie, doughnut or ring						✓					✓			✓				✓	✓			✓		6	
Clustering				✓	✓									✓				✓					✓	✓	6
Timeline					✓								✓					✓					✓	✓	5
Line chart						✓					✓			✓					✓			✓	✓		5
Glyph						✓							✓					✓	✓			✓			5
Map				✓	✓													✓						✓	4
Small multiples			✓					✓	✓								✓								4
Scatter plot		✓			✓	✓	✓					✓													4
Customised																				✓			✓		3
Area chart											✓		✓								✓		✓		3
Plot Matrix		✓				✓																			2
Candlestick															✓								✓		2
T-SNE					✓		✓																		2
Chord diagram					✓																				1
Icicle plot			✓																						1
Range chart			✓																						1
Radar chart			✓																						1
Box Plots							✓																		1
SOM																		✓							1
String and Beads*									✓																1
Heatmap									✓																1
Tree map																						✓			1
Massive sequence view																✓									1

2.4 Finance Visualisation

Although MoneyVis is the first publicly accessible bank transaction dataset, much previous research focuses on visualising the data with similar characteristics (i.e., temporal and having a column for money). Many of them used multiple coordinated views and interactive visualisation techniques. They provide a space of possibility for designing iBankEx. Section 2.4.1 to section 2.4.4 will present the essential (i.e., “concept, implementation, key related work, data characteristics, and evaluation” (Laramee, 2011, 79)) of the financial visualisation literature.

2.4.1 Company Information

Q. Q. Liu et al. (2020) proposed a visual analytic system helping compensation managers understand the company’s traffic reimbursement situation. Their work benefits from the “A* algorithm” (Yao et al., 2010, p. 1154) and a “density-based clustering algorithm called Mean Shift” (Comaniciu & Meer, 2002, p. 604). They utilised these two algorithms and multiple interactive charts (Figure 2.4.1.1) to show the overview traffic information and assist in detailed analysis to achieve their idea. Their data is temporal, geospatial, and multi-variate, spanning four months. They evaluated their work through case studies and domain experts’ feedback.

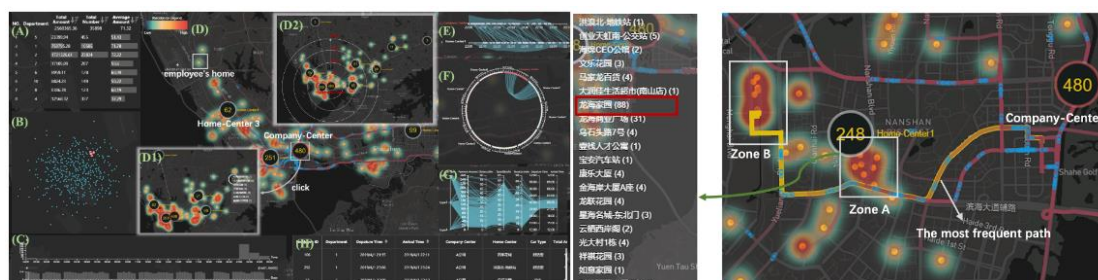


Figure 2.4.1.1. A screenshot of a Visual analytic tool for traffic reimbursement data from Q. Q. Liu et al. (2020). (A) A table showing department name, total amount, total number, and average amount of the reimbursements. (B) A Scatter plot showing reimbursement records in 2-D space obtained by t-SNE. (C) 2-layer timelines that can control the data shown in (H). (D) The map with yellow circles showing the employee’s traffic destination and the heatmap showing the density of the residential locations. (E) Timelines showing the reimbursement record. One timeline represents the home centre, and the other for the company centre. (F) Chord diagram whose angle represents time and segment represents destination cluster. (G) PCP showing detailed information for the selected cluster in (F). (H) Table for detailed reimbursement records chosen in (C).

Q. Q. Liu et al. (2021) developed a visual analytic tool (see Figure 2.4.1.2) to ease the exploration and comparison of different bank credit rating schemes. Their work strongly relies on Joachims (2002). They utilised a ranking tabular view to help interactively explore various ranking schemes, a project view based on “T-SNE” (Maaten & Hinton, 2008, p. 2579), which is for visualising high dimensional data, to show the similarities between banks, and a ranking comparison view to help understand the factors for the ranking result. The data used in their work is multivariate and static, which contains different banks’ information. A case study and domain experts’ feedback validated their work.

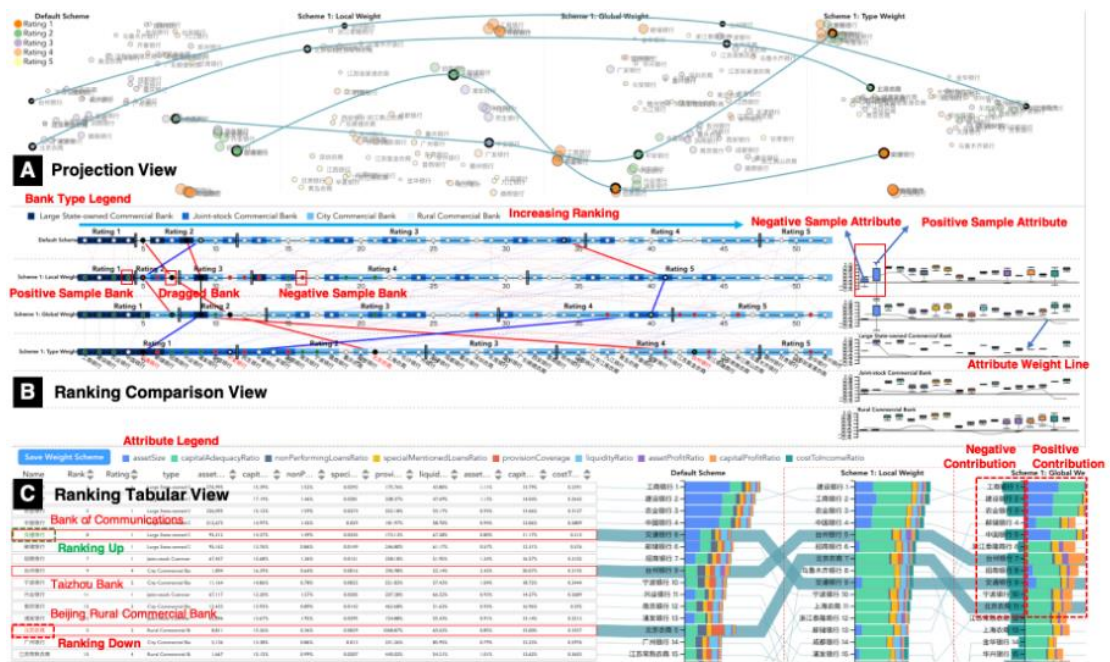


Figure 2.4.1.2. A screenshot of “RatingVis” from Q. Q. Liu et al. (2021). (A) Four scatter plots showing the projected bank data of different ranking schemes. (B) Ranking comparison views explain the individual banks' ranking schemes. (C) Table showing the current ranking schemes and allows the ranking change.

2.4.2 Stock

Lei & Zhang (2010) facilitate the analysis of stock market data through a visual analytic system, see Figure 2.4.2.1. Inspired by Dorogovtsev et al. (2006) and Zhao et al. (2008), they clustered prices by the k-core algorithm and developed a “Market View”, an “Underperforming Stock View”, an “Asset View”, a “Pattern View”, and a “Pattern Learning View” (Lei & Zhang, 2010, p. 5-6). Their work used time-series stock data and was evaluated by a user study.

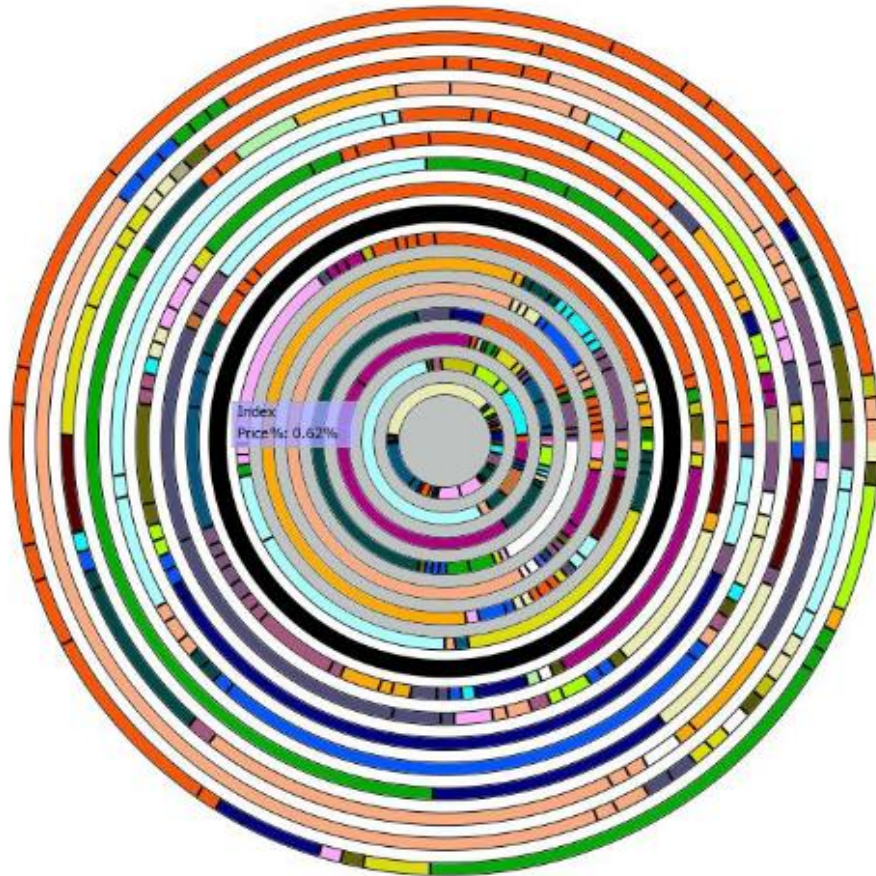


Figure 2.4.2.1. A screenshot of the ring chart from Lei & Zhang (2010). The arc length represents the capital size. The black ring is for the Heng Seng Index. The colour represents stock sectors. Grey rings represent the stocks whose prices declined, and the outside orange ones represent the stocks with better performance than the market index.

Tsang et al. (2020) proposed “TradeAO” (Tsang et al. 2020, p. 61) (Figure 2.4.2.2), a visual analytics tool that helps traders explore and assess the trading algorithm optimisation process. Their work is based on Pairs Trading (Deshpande & Barmish, 2016), Moving Average (Nakano et al., 2017) and the Multiple Linear Regression Model (Park et al., 2010). Linked views consist of an algorithm evolution view, a parameter correlation view, a trading residual view, and a cash usage view. A trading history view was also developed to facilitate the strategy overview tasks, algorithm instance inspection tasks and algorithm instance assessment tasks. They used temporal multi-variate stock data and assessed their work by expert interviews.



Figure 2.4.2.2. A screenshot of TradeAO (Tsang et al., 2020). (A) Trading evolution views show the evolution of trading algorithms with different parameters. The relative mode can help compare the performance of two algorithms with PCP. (B) Parameter correlation views indicate the overfitting condition of the algorithm. (C) Trading residual views showing the performance of the algorithm. (D) The cash usage view shows the use condition of cash usage within different trading periods. (E) Trading history view with the information of the trading orders of the entire period.

Yue et al. (2021) provided a visual analytic tool for factor investment, see Figure 2.4.2.3. Their work has a significant relation to Dingen et al. (2019). Their system chooses the financial features through sparse regression models. Meanwhile, their system provided a control panel, a factor view, a stock view, a factor list, a stock return view, and a back-testing view. They used multivariate temporal data for evaluating the system. A user study was conducted to validate the effectiveness and usability of their system.

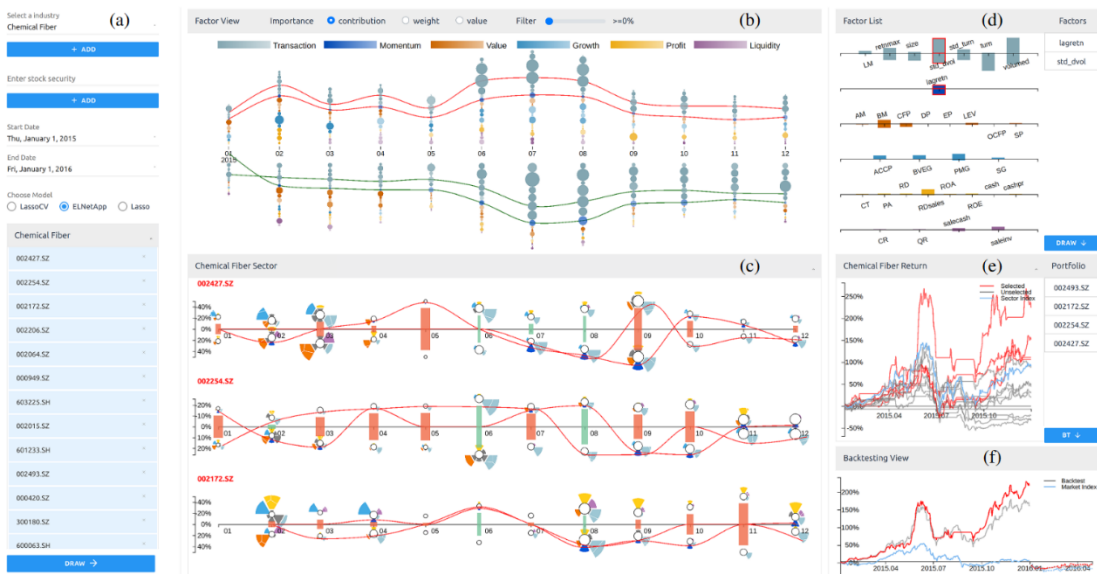


Figure 2.4.2.3. A screenshot of “iQUANT” (Yue et al., 2021, p. 189). (a) is the control panel. (b) is the factor view. (c) is the stock view showing each sector. (d) is the factor list allowing selection. (e) is the stock return view showing portfolio construction. (f) is the backtesting view

showing the trend of revenue.

Inspired by iQUANT, Guo et al. (2022) proposed a visual analysis tool (Figure 2.4.2.4) that aids stock market investors in deciding factor investing. They collaborated with domain experts and used the regression model and AR+GARCH model. Additionally, they provided a novel visualisation design to show the performance of the factors and help construct a portfolio. The nature of the data is a multivariate time-series and has 3000 stocks data spanning 7 thousand days. Case studies and user studies evaluated their work.

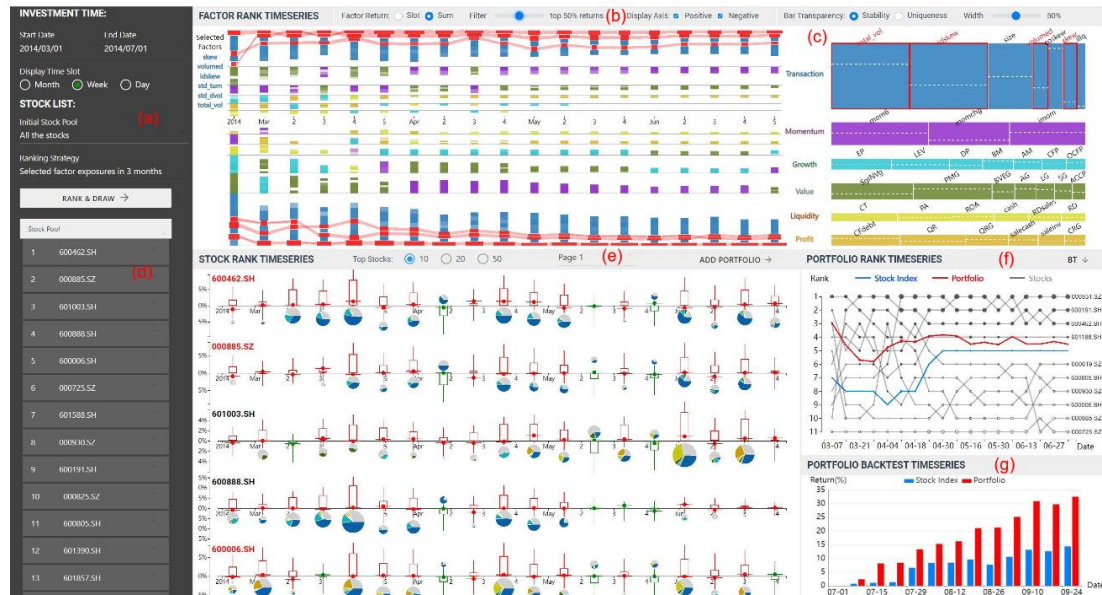


Figure 2.4.2.4. A screenshot of “RankFIRST” (Guo et al., 2022, p.2). (a) Control panel. (b) Factor view helps explore the selections’ factor return. (c) Factor tree map showing the returns of aggregated factors. (d) List of ranked stock. (e) Stock timeseries view allows the constructed portfolio. (f) Portfolio rank timeseries show the change in the rank of the stocks over time. (g) portfolio backtest timeseries showing the returns of the stock index versus the portfolio.

He & Li (2022) improved the appearance and readability of stacked charts (Figure 2.4.2.5). Their work was inspired by Byron & Wattenberg (2008). To achieve their goal, He & Li (2022) provided an optimised layer ordering algorithm based on the “minimum cumulative variance rule” (He & Li, 2022, p. 66) and used “width priority principles” (He & Li, 2022, p. 66) to improve the placement of labels. They also conducted a survey consisting of 268 questionnaires to support the choice of colour. Their data is temporal of 11 trading days. The layer ordering mechanism was evaluated by the Cumulative Variance index, measuring the sum of the volatility of each layer.



Figure 2.4.2.5. Comparison between different stack graphs generation approach: “(a) input data, (b) input order, (c) MinWig8, (d) TwoOpt21, (e) the flipped order of” (He & Li, 2022, p. 72) theirs, and (f) generated by their approach.

2.4.3 Fund

Rudolph et al. (2009) proposed a visualisation analytic system (see Figure 2.4.3.1) that helps casual users make personal finance decisions. Based on ThemeRiver (Havre et al., 2000) and investment planning (Lusardi & Mitchell, 2005), they built their financial model using the Standard portfolio theory method and used table and Themeriver to show the investment, risk, remaining money, and the baseline. The data used in their system is temporal fund data. They evaluated their work through a laboratory experiment.

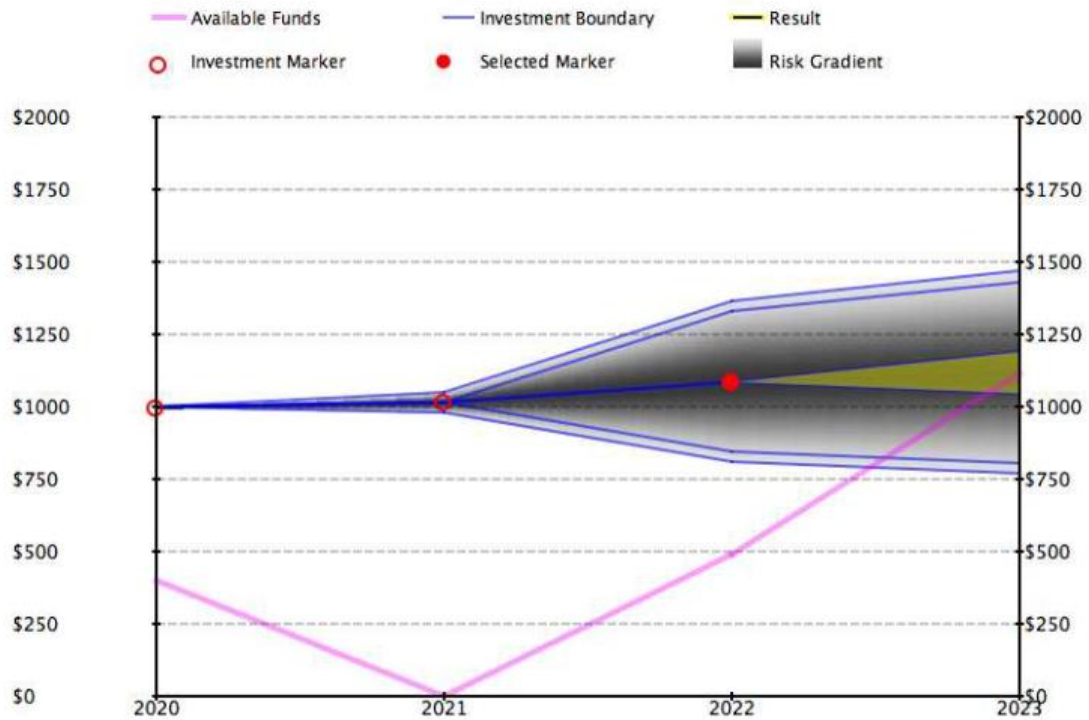


Figure 2.4.3.1. A screenshot of the visualisation for the investments from Rudolph et al. (2009). The purple line shows the remaining amount to invest. The blue line and the grey area offer the risk range. The filled red marker is the selected investment located at the start year of the investment. The unfilled red marker is the unselected investment. The yellow part shows the risk contribution of the selected investment.

2.4.4 Economic Indicators

Ruppert et al. (2017) provided an interactive visualisation system to ease decision-making in the mining sector. Their work is based on Simon's (1960) decision-making model. To achieve their goal, they provided a matrix view (Figure 2.4.4.1), a statistic view for exploring an individual country's data, and a detailed view for the detailed question. Additionally, they enable similarity search, helping users find similar countries and developing a country comparison view for comparing different countries. They used static multivariate questionnaire data. A user test with quantitative and qualitative questions validated their work.

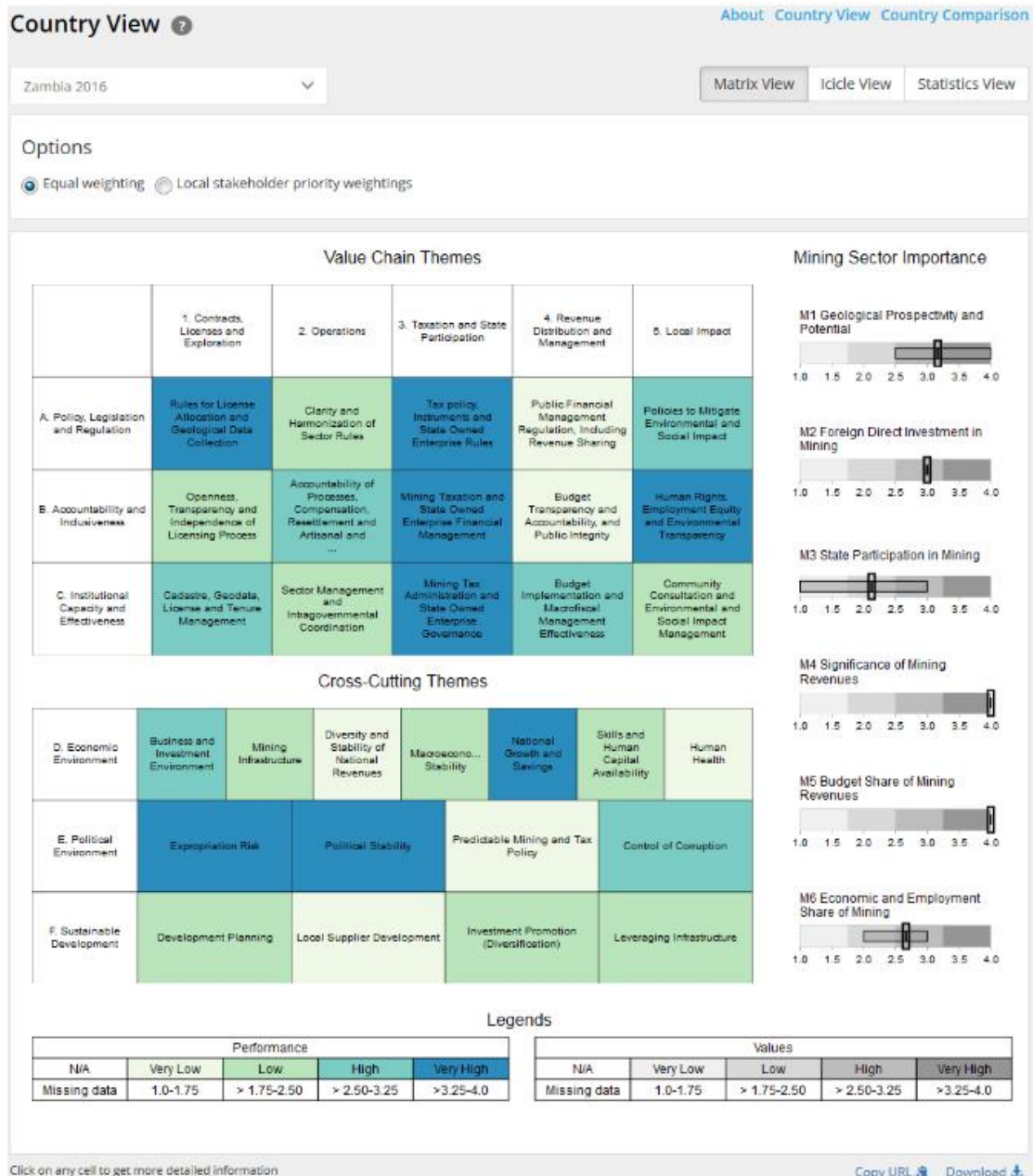


Figure 2.4.4.1. A screenshot of “Weighted Matrix View” (Ruppert et al. 2017, p. 159). The chart shows an overview of the dataset of an individual country. The top-left chart shows the topics' performance with “mining value” (Ruppert et al. 2017, p. 158). The bottom-left chart shows those without mining value. The right chart shows the “mining sector importance scores” (Ruppert et al. 2017, p. 158). The buttons at the bottom right corner can export the charts.

Hermes (Leite, Arleo, et al., 2020) is a visualisation tool (see Figure 2.4.4.2) for exploring economic networks. This work closely relates to Ceneda et al. (2019). They visualised their data through 2 pages of multiple coordinated views. They utilised a guidance-enriched approach for sub-graph searching. The data used is temporal, multivariate economic graph data with 172 nodes over 20000 edges. Their system was evaluated by user study, case study and experts' feedback.

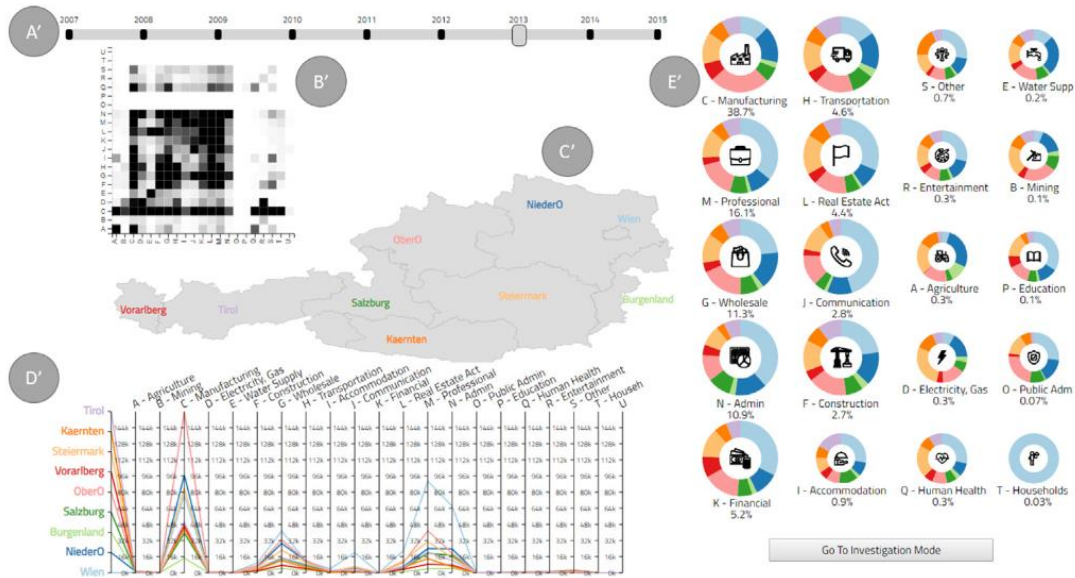


Figure 2.4.4.2. “Hermes” (Leite, Arleo, et al., 2020, p. 11). (A) Time slider for selecting a year. (B) The Connection Matrix shows the connection of sectors for the country. (C) A map indicating the colour of each region. (D) Parallel coordinates showing the contribution of each sector of different regions. (E) Sector doughnut charts presenting each sector’s regional distribution.

2.4.5 Transaction

Arleo et al. (2019) proposed a visual analytics tool to help investigate time-series national economies data. Their work relates to previous visual analytic research (Didimo et al., 2012; Kirkland et al., 1999; Leite, Gschwandtner, Miksch, Kriglstein, et al., 2018; Pham & Lee, 2017). Arleo et al. (2019) visualised inter-company transactions using micro and macro data from different sources, see figure 2.4.5.1. Domain knowledge was used for the incremental model-building process. They used spatiotemporal and multivariate transaction data and national economic data. They validated the work through a user study.

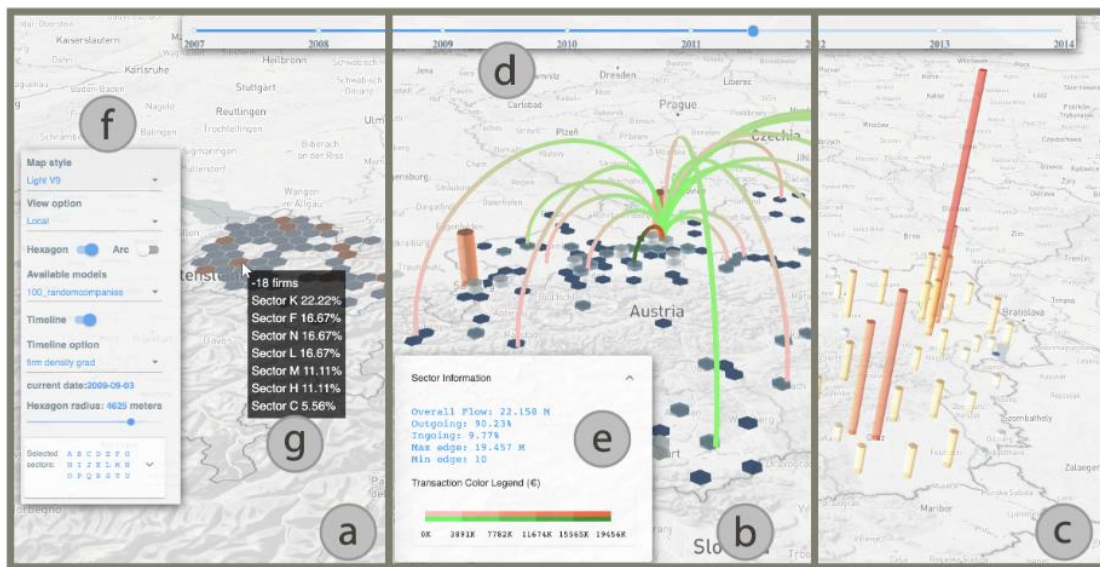


Figure 2.4.5.1. A screenshot of “Sabrina” (Arleo et al. 2019). (a, b, and c) Map showing an overview of the density of the companies and transaction data in Austria. Each hexagon represents a group of aggregated firms. (d) Timeline controlling the time of the shown data. (e)

Detailed information for the aggregated selected transactions. (f) Control panel for encoding and filtering. (g) Tooltips showing detailed information about the hexagon.

Arleo et al. (2023) then developed an improved version, Sabrina2.0 (see Figure 2.4.5.2), with advice from domain experts. Sabrina2.0 show the data from different levels, including the transactions between companies and between regions. It provided the options for different models. Sabrina2.0 used spatial-temporal data from a heterogeneous source, with only 1000 companies were chosen. User studies and case studies evaluated the system.

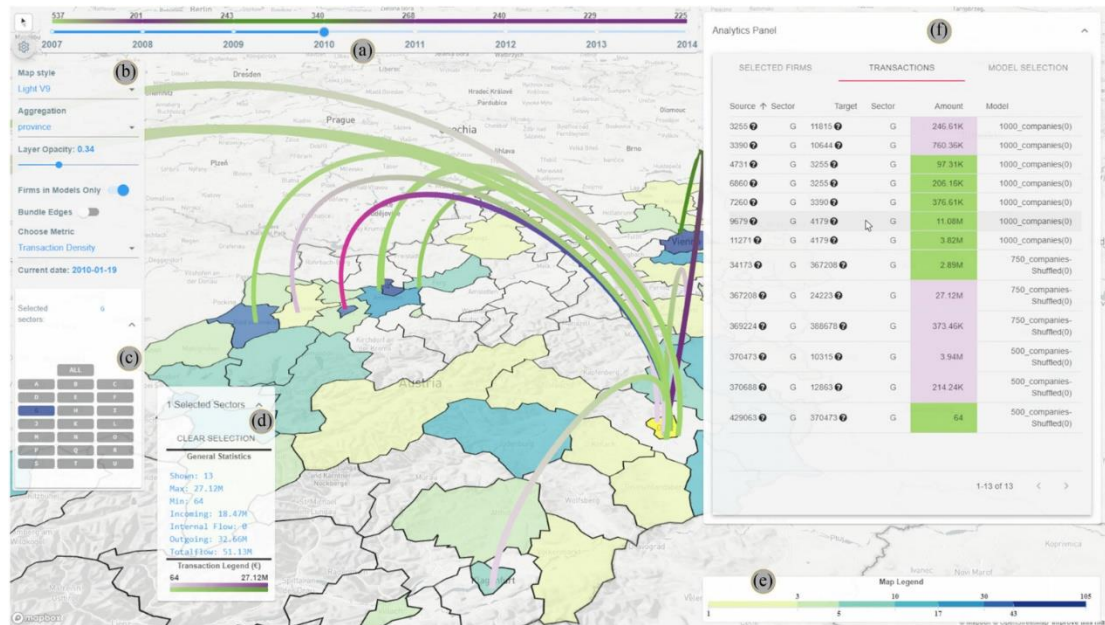


Figure 2.4.5.2. Sabrina2.0 from Arleo et al. (2023). (a) Timeline. (b) Configuration panel. (c) Selection panel. (d) Detailed information. (e) Colour legend. (f) Table showing transactions.

Leite et al. (2016) developed a visual analytics tool (Figure 2.4.5.3) to improve financial fraud analysis. Their work is closely related to Carminati et al. (2014). They prototyped an automatic transaction evaluation system with visualisation to help users understand and improve the evaluation system. They used temporal financial transaction events data.

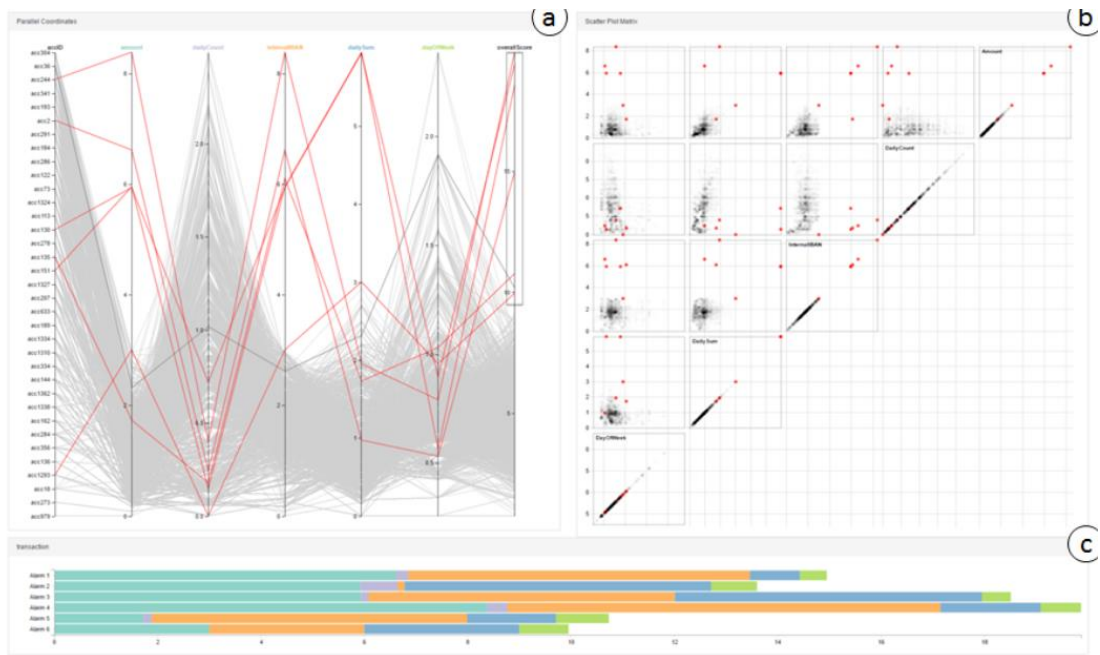


Figure 2.4.5.3. A screenshot of the visual analytic tool with linked views from Leite et al. (2016). (a) Parallel coordinate where each line represents a transaction. (b) Scatter plots where each point represents a transaction. (c) A stacked bar chart shows the score of each highlighted transaction.

Leite et al. (2018) provided a visual analytics system (i.e., EVA) (see Figure 2.4.5.4) to help with financial fraud detection. Their work used a similar approach proposed by Kirkland et al. (1999), which is the integration of “AI, visualisation, pattern recognition, and data mining” (Kirkland et al., 1999; Leite et al., 2018, p. 331), and extended with interactive visualisation techniques. They iteratively developed their system with domain experts utilising a scoring algorithm to find unauthorised transactions and highlighted them in their visualisation system. They used money transaction data, which is multivariate geospatial and temporal. There are thirteen million records over fifteen months. User studies and interviews evaluated the usability and effectiveness of their system.

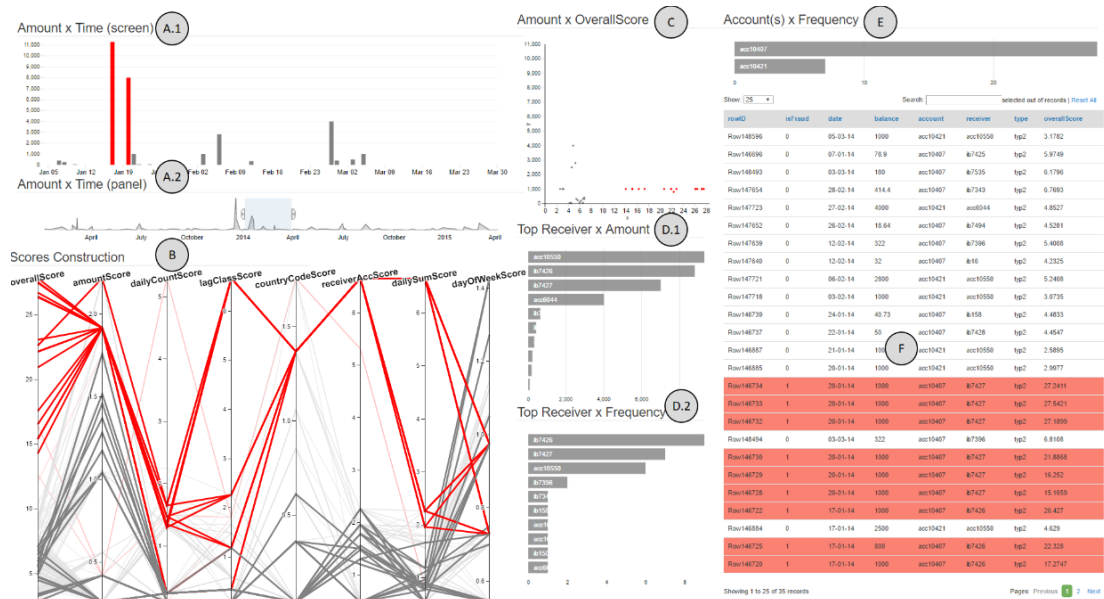


Figure 2.4.5.4. A screenshot of “EVA” (Leite, Gschwandtner, Miksch, Kriglstein, et al., 2018) (A.1, A.2) Temporal views. (A.2) Filter. (B) Parallel coordinates plot showing transactions and their score. (C) Scatter plot showing the amount and overall score. (D) Sorted bar chart showing the select account’s amount (D.1) and frequency (D.2). (E) Bar chart as accounts selector showing each account’s number of transactions. (F) Table view showing detailed data.

Didimo et al. (2014) provided a visualisation system (see Figure 2.4.5.5) to ease the analysis of financial activity networks. Their work closely relates to “VisForFraud” (Di Giacomo et al., 2010, p. 393). The interaction method used in their work was inspired by Eades & Huang (2004). They “combined enhanced graph drawing techniques to devise novel algorithms and interaction functionalities for the visual exploration of network dataset, together with tools for SNA and for the automatic generation of reports.” Didimo et al. (2014, p. 433). They used temporal transaction data collected by financial institutions such as “banks and casinos” (Didimo et al., 2014). Case studies and user tests were conducted on the usefulness of their system.

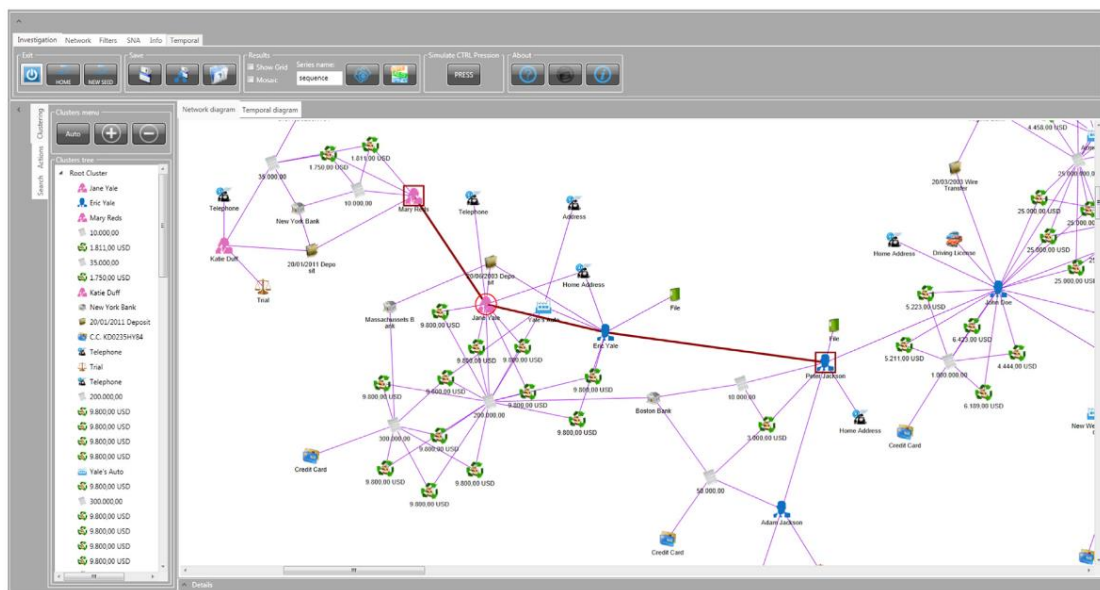


Figure 2.4.5.5. VISFAN from Didimo et al. (2014).

Leite et al. (2020) developed a visual analytics tool (see Figure 2.4.5.6) to improve fraud analysis. Their work is closely related to “EVA” (Leite, Gschwandtner, Miksch, Kriglstein, et al., 2018). By collaborating with domain experts, they developed a visualisation system fitting into the financial fraud detection workflow and enabling “guidance-enriched pattern search” to facilitate analysis at different complexity levels. They used in their system is multivariate time-series network transaction data spanning over two years from 77,000 accounts. They conducted user studies and interviews to evaluate their work.

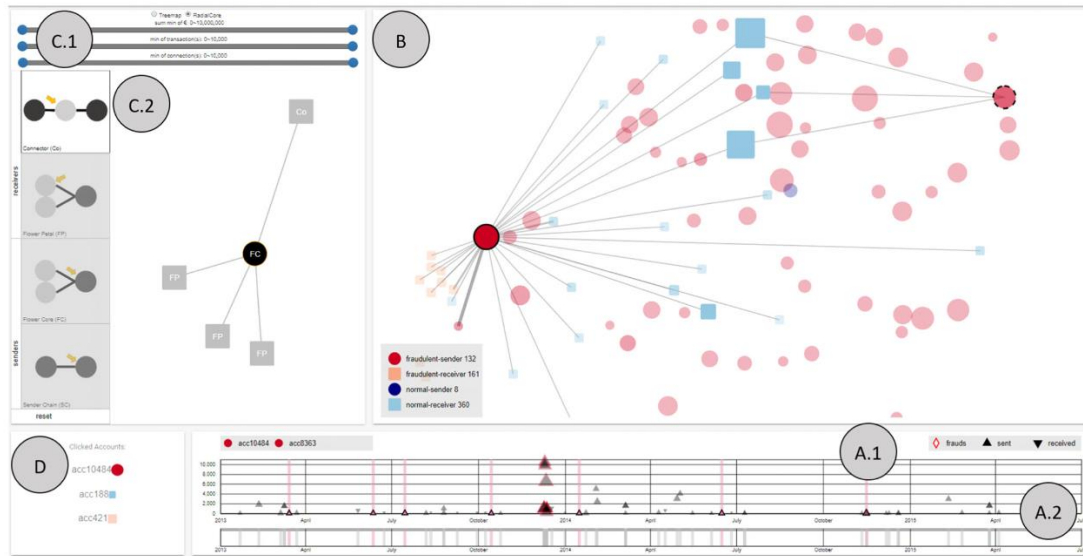


Figure 2.4.5.6. A screenshot of “NEVA” (Leite et al., 2020, p. 349). (A) Temporal views. (A.1) Detailed temporal view. (A.2) Overview of temporal view. (B) Node-Link view showing the network of the accounts to analyse. (C) Pattern search panel with guidance. (D) History of the clicked nodes. Suspicious data are highlighted.

WireVis (Chang et al., 2007, p.3) is a visualisation tool facilitating the exploration of wire transaction data, see Figure 2.4.5.7. Their work filled the research gap in visualising financial transaction data by multiple coordinated views, which had been proved feasible by North & Shneiderman (2000). They designed filterable coordinated multiple views showing both overview and detail of data over time and provided a search-by-example technique to show the similar transactions. They used binning techniques, grouping the accounts based on the keyword frequency of each account’s transactions. They used temporal multivariate transaction data sampled over twelve months. They assessed their system through two case studies.

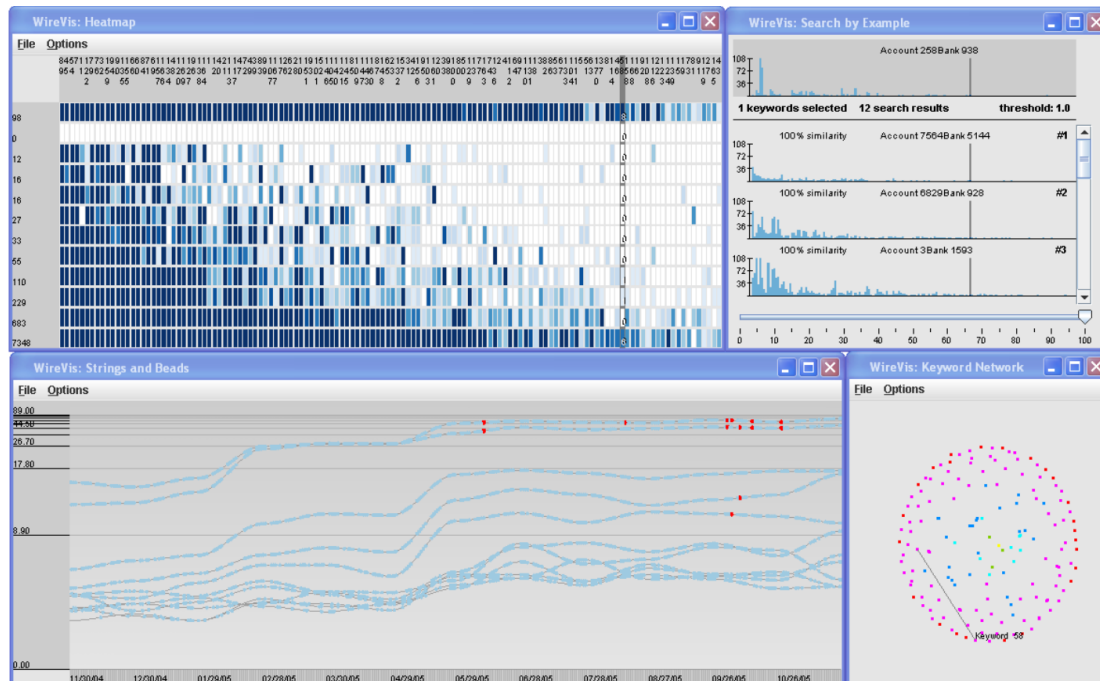


Figure 2.4.5.7. “WireVis” (Chang et al., 2007, p.3). The top-left view is a heap map showing relationships between accounts and keywords. The top-right window is the search-by-example

tool. The bottom-left window is the string and bead view showing the temporal pattern of the transactions. The bottom-right view is the keyword graph.

Their later work (Chang et al., 2008) improved the scalability of WireVis by using SQL Server 2005. As Figure 2.4.5.8 demonstrates, they build six database tables for raw data and five database tables for visualisation views. The procedures for the data process are also stored in the database. The data is temporal and relational, with 46000k records. They validated their work through performance tests.

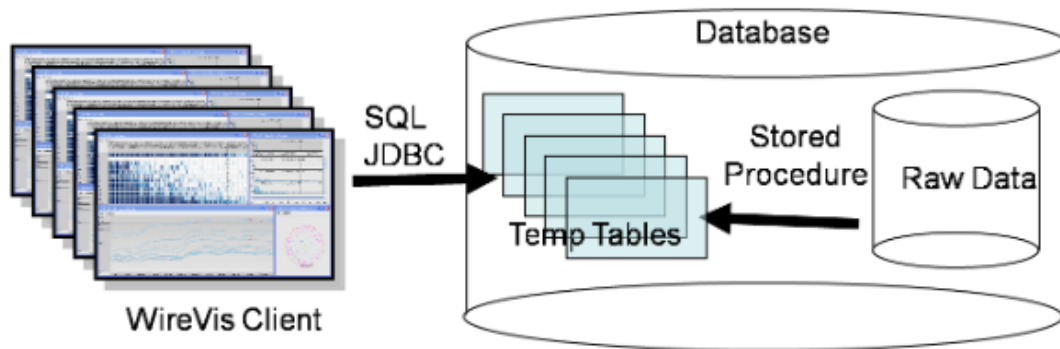


Figure 2.4.5.8. The connection between “WireVis to a database” (Chang et al., 2008, p. 7).

Inspired by Wasserman & Faust's (1994) research on social network analysis, Singh & Best (2019) proposed AMLink to help detect money laundering. Before visualising the data, they imported the data and then processed the data by SQL query. Sequentially, data are converted to the format that can be rendered by GraphViz. Finally, GraphViz rendered the data as node-link chart (Figure 2.4.5.9). They used multivariate and temporal anonymised transaction data. Domain experts' feedback was used for assessing their system.

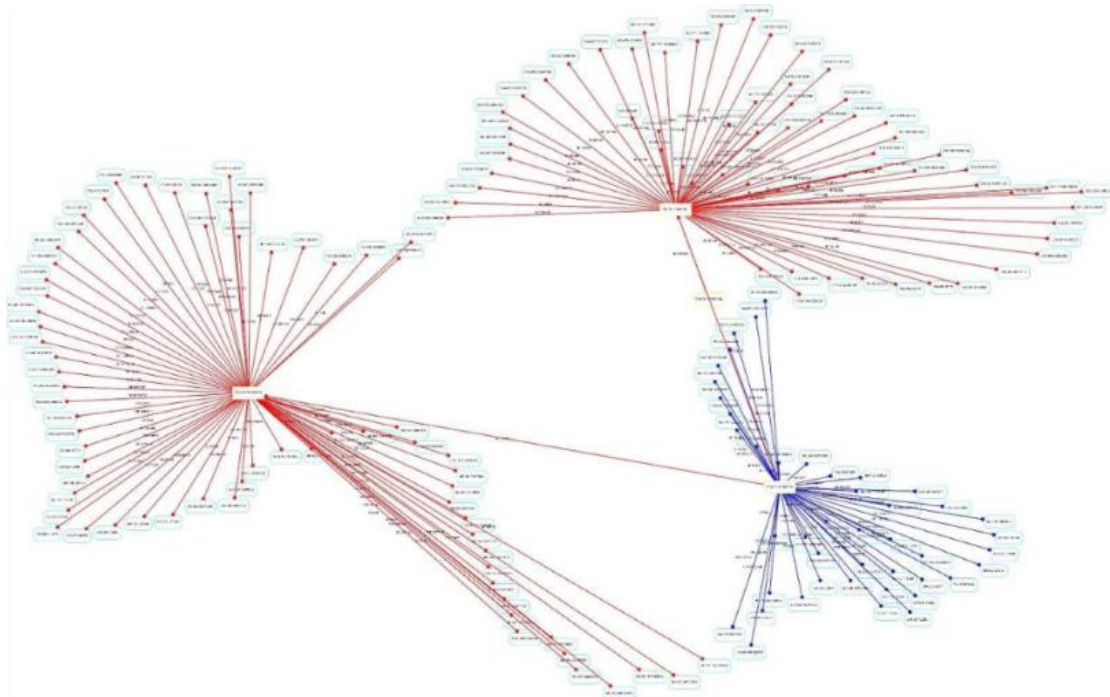


Figure 2.4.5.9. Node-link chart used in AMLink from Singh & Best (2019).

ATOVis (Maçãs et al., 2022) is a visual analytic system that helps fraud detection, see Figure

2.4.5.10. The multilevel timeline used in their system was inspired by Brehmer et al. (2017). With the help of e-commerce experts, they provided a multiscale timeline showing both the overview and detail of a transaction over time, a main view of customers' transaction behaviour and a details area for the details of the transaction within a period. Moreover, clustering and aggregation were applied to avoid cluttering. They used multivariate, temporal transaction data consisting of more than 4 million transactions. A user study tested their system.

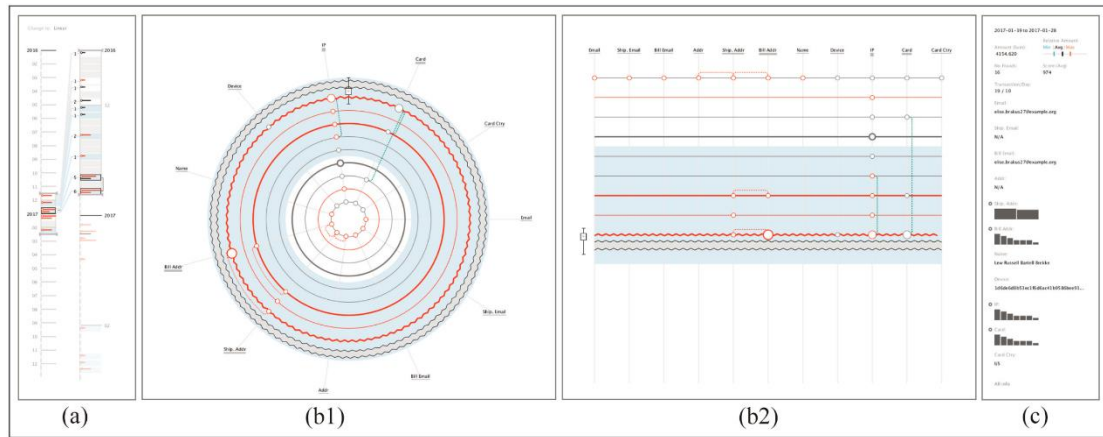


Figure 2.4.5.10. A screenshot of “ATOVis” (Maças et al., 2022) (a) Timeline area showing both the overview and the detailed temporal pattern of transactions. (b) The main view provides an overview of a user’s transaction behaviour. In (b), each straight line represents transactions aggregated by day, and each curvy line represents clustered transactions. The red line is the fraudulent transactions. Circles indicate the change of the attribute. The thickness of the lines and circles represents the number of transactions. The dotted line connects the related attributes. (b1) is a radial representation where the inner line happens earlier. (b2) is the linear representation where the top line happens earlier. (c) Details of the selected transactions.

Yue et al. (2019) provided an interactive visualisation tool (Figure 2.4.5.11) to ease the comparison and analysis between Bitcoin exchanges, which had been overlooked in previous research. They utilised multiple coordinated views with interactive visualisation techniques to show the pattern and trend of the Bitcoin market. They used multivariate, time-series transaction data, which is bigger than 10GB spanning seven years. Their research was evaluated by case studies and interviewing domain experts.



Figure 2.4.5.11. A screenshot of “BitExTract” (Yue et al., 2019, p. 162). (A) Comparison view for comparing the different indices of the exchanges. (B) Exchange list illustrating the history transactions volume of different bitcoin exchanges. (C) Massive sequence view providing an overview of the entire market. (D) Connection view showing the behaviour between different exchanges.

Maçãs et al. (2020) eased the visual analysis of banking transactions fraud. Their work was inspired by Wirevis (Chang et al., 2007) and Self-Organising-Map (Rogovschi et al., 2011) (i.e., a dimension reduction technique). With the help of a fraud detection company, they built a visualisation tool representing each transaction by glyph and having a history view (Figure 2.4.5.12) and a transaction topology view. Their data set is anonymised bank transactions, which are multivariate and temporal. They validated their work through user studies.



Figure 2.4.5.12. Transaction history view from Maçãs et al. (2020). (A) GUI panel. (B) Glyphs matrix mapping x to time and y to transaction amount and amount histograms. (C) Time histogram and mini self-organising-map. (D) Timeline.

Firat et al. (2023) proposed the first open retail bank transaction dataset. The previous publicly unavailable transaction datasets inspired them. They first anonymised their data and applied manual, semi-automatic, and automatic categorisation. The characteristics of their data set will be introduced in section 2.3. The usability of their dataset was validated by the initial visualisation (Figure 2.4.5.13).

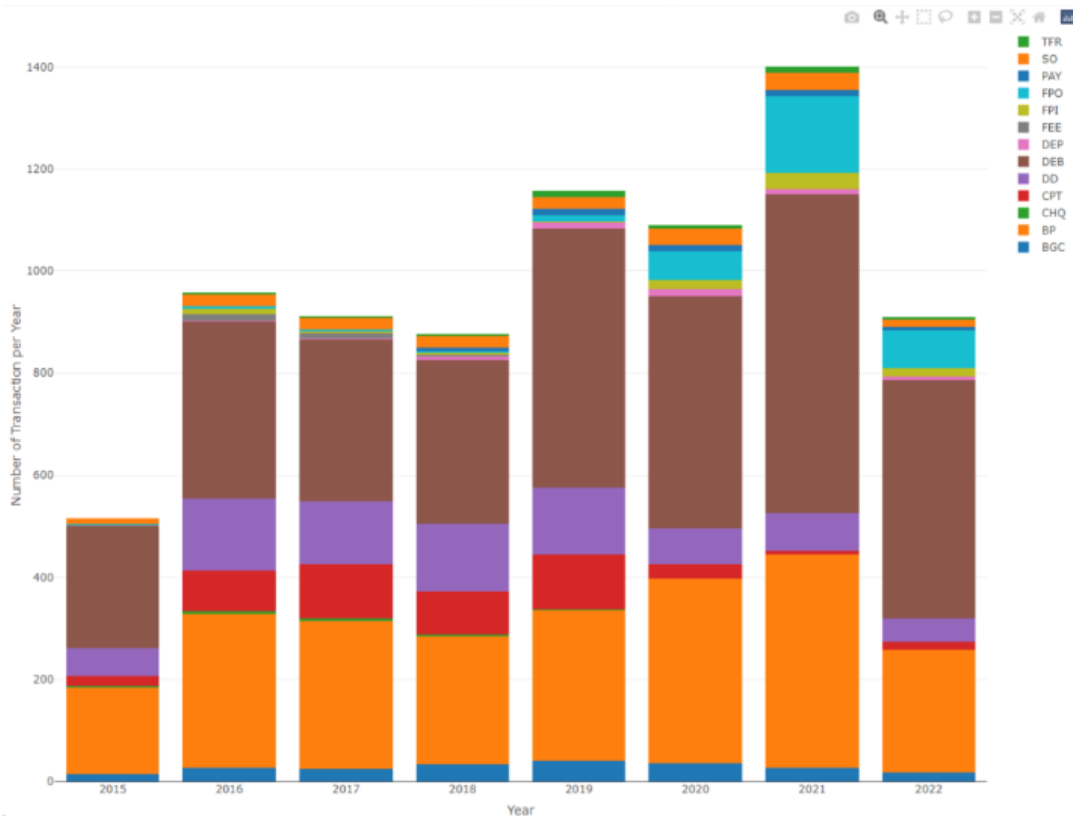


Figure 2.4.5.13. A stacked bar chart showing the transaction type automatically defined by banks of different years from Firat et al. (2023).

2.5 Visualisation Surveys

There are surveys in information visualisation, which help understand the current knowledge of information visualisation.

McNabb & Laramée (2017) provided the first survey of survey for information visualisation to help researchers understand this field. They searched 86 information visualisation survey papers (Figure 2.5.1) through linear and relation searches, used a 2-d classification scheme that uses subject (i.e., “Data-Centric, Multivariate & Hierarchical, Graphs & Networks, Coordinated Multiple Views, Real-World Applications, Overview” (McNabb & Laramée, 2017, p. 591)) and information visualisation pipeline (i.e. “Data Enhancement & Transformation, Visual Mapping & Structure, Exploration & Rendering, Interaction & Analysis and Perception” (McNabb & Laramée, 2017, p. 593)).

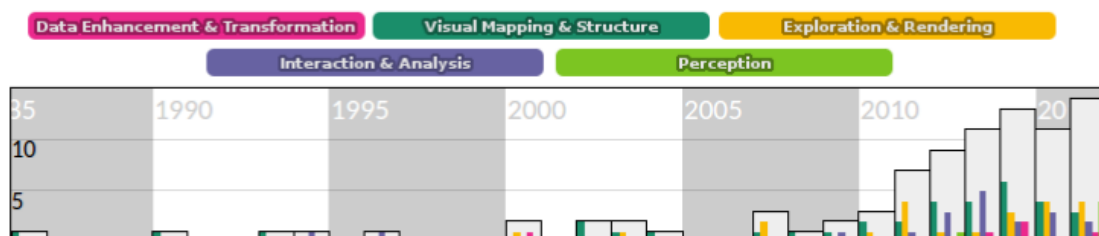


Figure 2.5.1. A histogram where x represents the year of the survey and height represents the number of the survey. The colour represents the category of the survey. Figure from McNabb & Laramée (2017).

However, the SoS (McNabb & Laramée, 2017) did not focus on visualisation books, so Rees

& Laramee (2019) provided a survey for the books. They searched the books on their shelves and in online bookstores like Amazon, John Smith, etc. The number of books they have searched is over 23000 pages. They provided a two-level classification based on books and chapters topics (Figure 2.5.2).

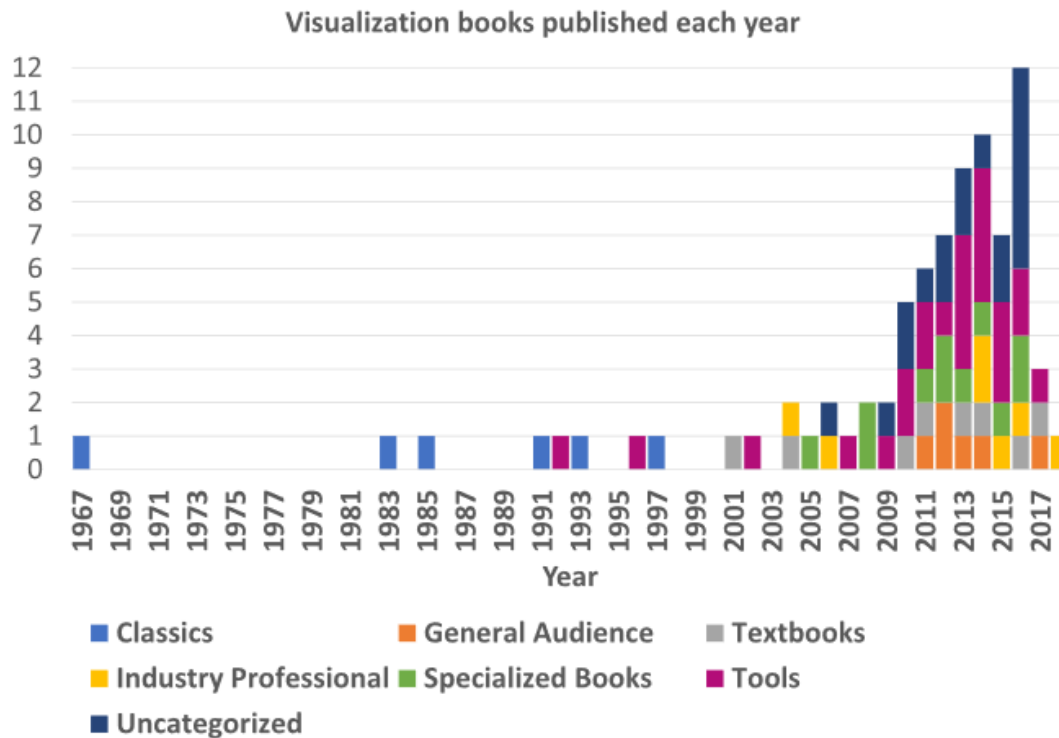


Figure 2.5.2. Metadata of the books included in the Survey of Books from Rees & Laramee (2019).

Liu et al. (2023) provided an overview of data visualisation resources to help different audiences find resources, see Table 2.5.1. Their work is closely related to Liu et al.'s (2021) short survey and shares a similar idea of providing a collection of resources with Matrix & SUS (2011). To achieve their goal, they provided collections of resources. They searched the collections of resources based on their rich experience in visualisation and the resources related to those resources. They classified them based on the resource type, including website, literature, software collection, colour, geospatial, and blogs. Moreover, they provided a website to update the collections.

Table 2.5.1. A table showing the target audiences of different resource collection subjects from Liu et al. (2023).

Subject \ Audience	Refereed Literature	Websites	Software	Software-Developer	Color	geospatial	Blogs
Students	✓	✓	✓		✓	✓	✓
Researchers	✓	✓			✓	✓	✓
Practitioners		✓	✓	✓	✓	✓	
Developers		✓	✓	✓	✓	✓	

Ko et al. (2016) provided a survey focusing on visual analytics for financial data. This survey benefits from the previous classification for the business application domain (Tegarden, 1999). They achieved their goal by reviewing 50 studies (Figure 2.5.3), interviewing financial domain experts and extracting task requirements of the exploratory financial data visualisation. Then,

they classified the literature by different categorisation schemes, including “data sources, applied automated techniques, visualisation techniques, interaction, and evaluation methods” (Ko et al. 2016, p. 599).

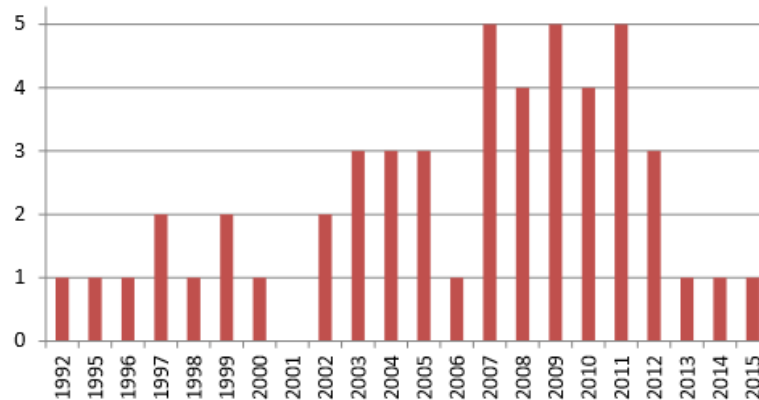


Figure 2.5.3. A bar chart showing the year distribution of the literature searched by Ko et al. (2016). Figure from Ko et al. (2016).

Roberts & Laramee (2018) provided a more comprehensive survey for business data visualisation. Their scope is closely related to the financial visualisation that Ko et al. (2016) included. They searched the literature for combined terms related to business visualisation from popular sources or tools like IEEE Xplore. Then, they classified the literature into three categories, including “business intelligence, business ecosystem and customer-centric” (Roberts & Laramee, 2018, p. 1) (Figure 2.5.4). 70 pieces of literature were included in their survey.

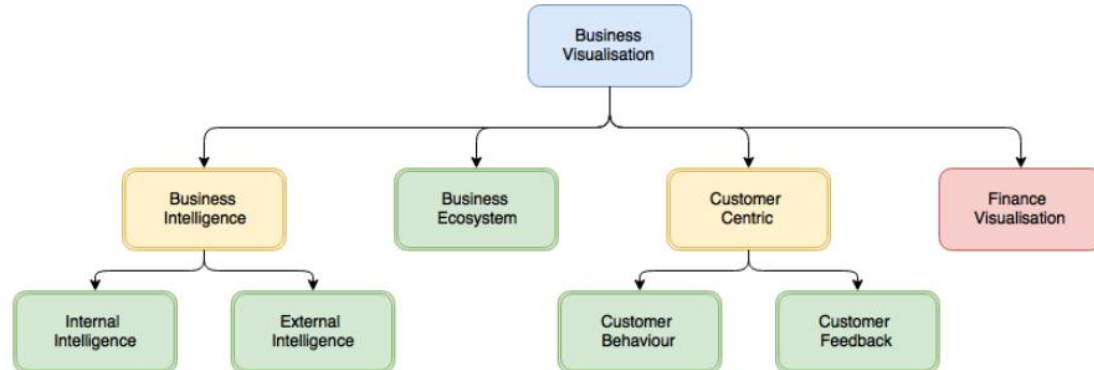


Figure 2.5.4. This tree shows the top-level classification of business visualisation literature from Roberts & Laramee (2018). Green ones are the leaf nodes. Yellow ones are the umbrella classification. Financial visualisation is related to this survey but has been included by Ko et al. (2016).

2.6 Visualisation Techniques

Many studies focus on visualisation techniques, such as multiple coordinated views, clustering, and glyphs, helping form the appropriate design of iBankEx.

Multiple coordinated view is powerful for data exploration, synchronising the views through interactive techniques like brush and linking (i.e., selecting a view and another view will update) (Langner et al., 2019; J. C. Roberts, 2007). The use of multiple views in iBankEx is influenced by their popularity among the previous financial visualisation research (Chang et al., 2007; Didimo et al., 2014; Leite et al., 2016; Leite, Gschwandtner, et al., 2020; Leite, Gschwandtner,

Miksch, Gstrein, et al., 2018; Maças et al., 2020, 2022; Singh & Best, 2019; Yue et al., 2019), see Section 2.1.2 and Table 2.6.1.

Table 2.6.1. Visualisation techniques used in financial visualisation research using transaction data. Orange or “*” indicates MoneyVis cited them. Green means they are related to Mao (2023), Ko et al. (2016), or Lei & Zhang (2010). “!” means the research is not a visualisation system but is an improvement related to financial visualisation. The classifications were proposed by Ko et al. (2016).

	Search source			!			*		*		*		
		Leite et al. (2016)	Arleo et al. (2019)	Chang et al. (2007)	Chang et al. (2008)	Didimo et al. (2014)	Leite et al. (2018)	Leite et al. (2020)	Singh & Best (2019)	Yue et al. (2019)	Maças et al. (2020)	Maças et al. (2022)	Arleo et al. (2023)
Visualisation Techniques	Bar charts	✓	✓	✓			✓			✓	✓	✓	7
	Graph		✓	✓		✓	✓	✓	✓	✓			7
	Table			✓		✓	✓					✓	4
	Clustering		✓								✓	✓	4
	Timeline						✓				✓	✓	4
	PCP	✓				✓			✓				3
	Scatter plot	✓				✓							2
	Small multiples			✓						✓			2
	Glyph						✓				✓		2
	Map		✓									✓	2
	Line chart					✓							1
	Pie, doughnut or ring					✓							1
	Customised											✓	1
	Area chart						✓						1
	Plot Matrix	✓											1
	SOM										✓		1
	String and Beads			✓									1
	Heatmap			✓									1
	Massive sequence view								✓				1

Moreover, Bach et al. (2022) provided some guidelines for designing a visualisation dashboard, sharing a similar concept with multiple coordinated views. Their work closely relates to the dashboards collected by Sarikaya et al. (2019). They provided a taxonomy of dashboard design patterns by reviewing 144 dashboards based on structure, visual design, and interactivity. Then, they grouped the dashboard into six genres. Sequentially, they discussed the design trade-offs and possible design frameworks for dashboards (Figure 2.6.1). They surveyed 144 dashboards and conducted a qualitative dashboard design workshop to evaluate and improve their work. They suggested that an ideal dashboard should minimise the screen space, abstraction level, number of pages, and interaction, which impacts the design of the current project.

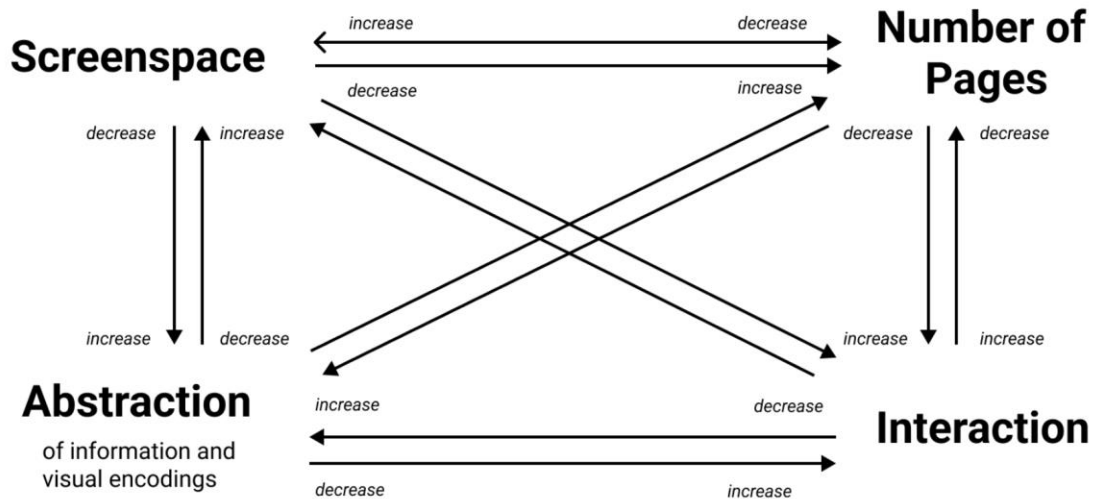


Figure 2.6.1. The trade-off between dashboard design (Bach et al., 2022). Increasing one will decrease the others.

As Table 2.6.1 presents, **clustering** is popular in visualisation research due to its ability to resolve cluttering (Ellis & Dix, 2007) by grouping similar entities. IBankEx is based on two clustering algorithms (i.e., k-mean and linkage-based clustering algorithms) introduced by Fuchs et al. (2019). They provided an online interactive visualisation tool (Figure 2.6.2) to help understand clustering algorithms. Their work was inspired by Hundhausen et al. (2002) and Shaffer et al. (2010). They developed their system with web technology and d3 library. They used 2D static data and evaluated their work by user feedback.

Moreover, Santos et al. (2018) introduced character-based, vector-based, and hybrid-based string measure methods in their research, providing the options for choosing string measures in iBankEx.

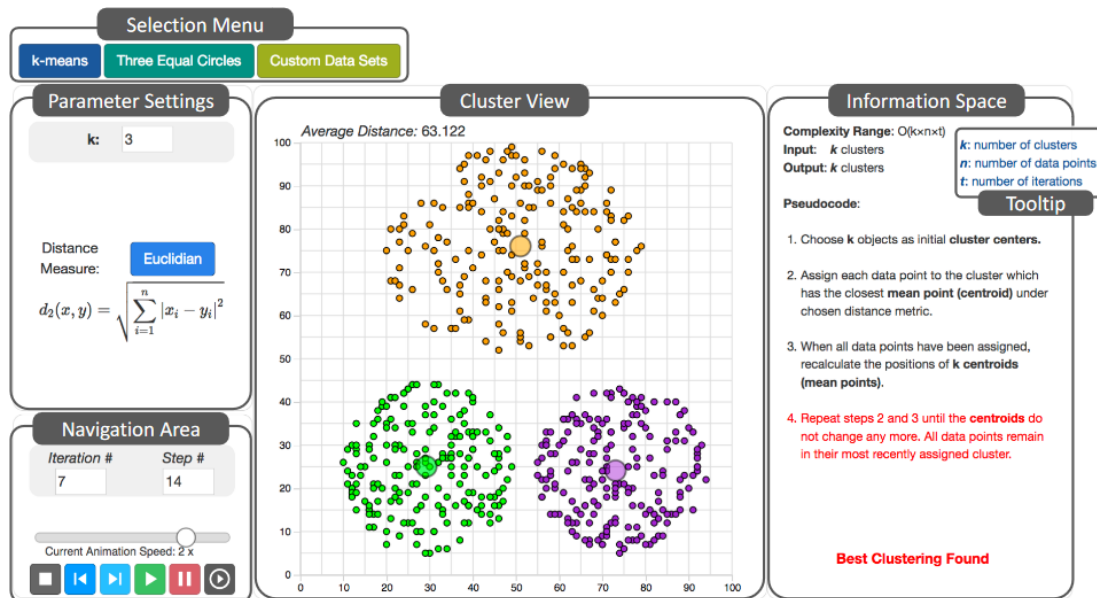


Figure 2.6.2. A screenshot of EduClust from Fuchs et al. (2019). This tool helps learn clustering algorithms by providing an interactive visualisation with a scatter plot and dendrogram. The selection menu allows the user to choose a dataset. The parameter settings panel allows the change of the parameters for the algorithm. The navigation area allows the user to control the steps of the clustering process. The cluster view shows the clusters on a scatter plot. The

information space shows the current step of the clustering algorithm and explains the algorithm through pseudocode. The tooltip provides more detail for the algorithm.

Glyph-based visualisation effectively shows multiple data attributes on a single visual object (Borgo et al., 2013). There are several glyph-related guidelines and research. Borgo et al. (2013) presented a comprehensive report related to glyph-based visualisation. They linked the underlying theories in semiotics, perception, and cognition with glyph-based visualisation, searched the existing guidelines for design and implementation, and surveyed the applications of glyph-based visualisation.

Fuchs (2015) aimed to help design and choose glyphs for temporal and multi-dimensional data. They reviewed the experiments conducted in the last 70 years and conducted experiments for both temporal and multi-dimensional glyphs (Figure 2.6.3). They provided guidelines for designing effective glyphs.

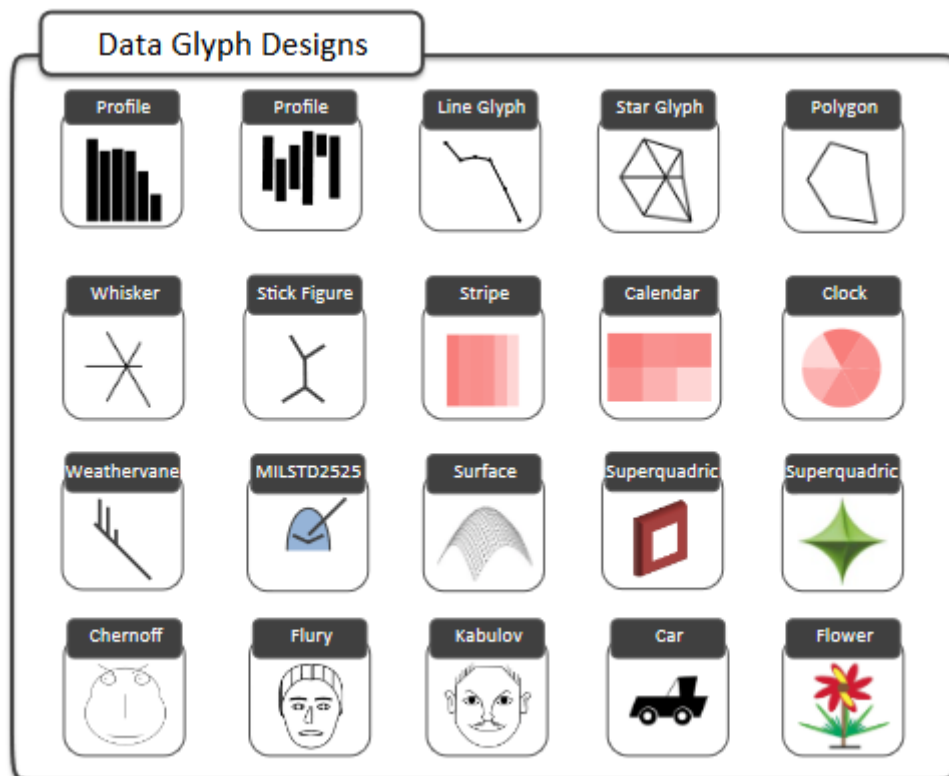


Figure 2.6.3. The Glyphs used in Fuchs' (2015) experiments.

Rees et al. (2021) improved glyph visualisation for agent behaviour. Their work is closely related to the grid-based cluster aggregation technique (Fuchs et al., 2016) and glyph placement strategies (Ward, 2002). With the help of domain experts, Rees et al. (2021) designed the hierarchical glyphs (Figure 2.6.4) that can avoid overlapping by aggregation. They used context and focus (i.e., greyscale for unimportant attributes) to highlight the important attributes, which is also used in iBankEx. They used multi-variate temporal call centre data with more than 5 million records. Case studies and domain expert feedback evaluated their work.

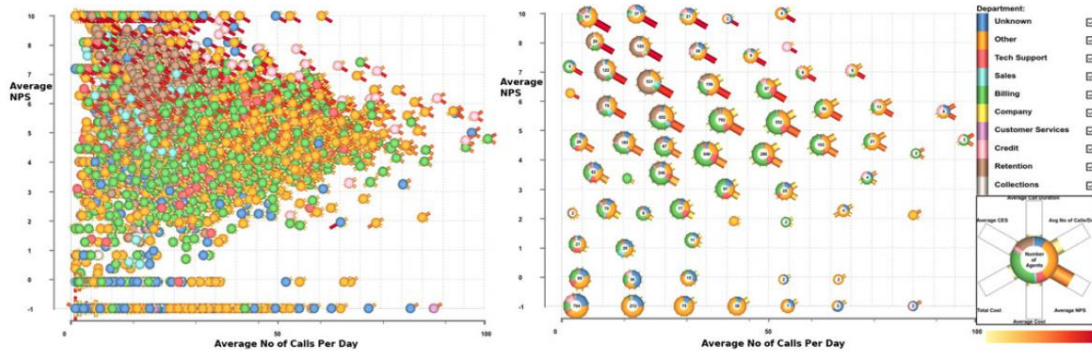


Figure 2.6.4. A screenshot of AgentVis from Rees et al. (2021). The left chart displays more than 6500 overplotted glyphs. The right chart clustered the glyphs and used size to represent the number of children in the group.

McNabb & Laramee (2019b) also utilised context and focus, improving the glyph placement mechanism on the map. Their work is closely related to the previous survey and visualisation tool focus on glyph placement (Ward, 2002, 2008; Ward & Lipchak, 2000). Pie, polar area, bar, and star glyphs (see Table 2.6.2) were chosen in their work and also used in the iBankEx. They designed a novel algorithm pipeline and enabled different interactivity techniques like zooming and filtering. They tested their idea through 3 case studies and compared them with standard glyph placement strategies. The case studies used different dataset which are multi-variate and temporal, up to more than 3000 entities.

Table 2.6.2. Glyphs used by McNabb & Laramee (2019b).

Glyph Design	Hidden Density Indicators				
	No Indicator	Outline	Size	Shadow	Size + Outline
Pie Chart					
Polar Area Chart					
Bar Chart					
Star Chart					

3 Data Characteristics

The MoneyVis dataset (Firat et al., 2023) is a time-series multivariate retail bank transaction data from a single account. There are 6567 entries of transactions spanning from the July of 2015 to the July of 2022. The dataset has the following columns:

1. Transaction number: unique id of transaction.

2. Transaction type: categories automatically generated by the bank.
3. Credit: the amount of incoming transaction.
4. Debit: the amount of outgoing transaction.
5. City: with many missing values.
6. Country: with many missing values.
7. Transaction description: the description of each transaction.
8. Date: transaction happens on Saturday and Sunday are put into the following Monday.
9. Category: a category of the transaction added by Firat et al. (2023).

The dataset is provided online by Firat et al. (2023) at this URL: <https://tinyurl.com/y4e8yevn>.

Before developing the app, preliminary data exploration was conducted by Tableau. A dashboard (Figure 3.1) was developed to show the patterns and trends of the data.



Figure 3.1. A Tableau dashboard for exploring the dataset before developing iBankEx.

Some errors were found during the preliminary data exploration. They were fixed before developing the software. Firstly, some cities are represented incorrectly (Figure 3.2). For instance, Nottingham is represented by “Nottingham”, “Nottingham ”, and “nottingham”. To handle this, the city and country column were capitalised, and the right space that appeared in city names were removed. The value “Almer??a” was replaced by “Almeria”.

```

sortedUniqueCityNameString = sorted([str(item) for item in df['Location City'].unique()],
                                     key=lambda x: x.lower())
for index, cityString in enumerate(sortedUniqueCityNameString):
    if (index+1) % 5 == 0:
        print(f' {cityString}')
    else:
        print(f' {cityString}', end=',')

```

✓ 0.0s

```

'Aarhus', 'Almer??a', 'Amsterdam', 'Baltimore', 'Bargoed'
'Belfast', 'Belfast ', 'Berlin', 'Birmingham', 'Boston'
'Bristol', 'Brno', 'Budapest', 'Bushmills', 'Cambridge'
'Cardiff', 'Casablanca', 'Derby', 'Genova', 'Houston'
'Huston', 'Illes Balears', 'Lampeter', 'Leeds', 'Leicester'
'Lisbon', 'London', 'London/birmingham', 'Madrid', 'Malia'
'Manchester', 'Mansfield', 'Middleton', 'nan', 'New Brighton'
'Newcastle upon Tyne', 'Nottingham', 'nottingham', 'Nottingham ', 'Oakham'
'Paris', 'Phoenix', 'Plymouth', 'Plymouth/Newcatle', 'Porto'
'Prague', 'Reading', 'Roma', 'Santa Cruz de Tenerife', 'Setubal'
'Sheerness', 'Sheffield ', 'Shrewsbury', 'Solihull', 'Swansea'
'swansea', 'Vancouver', 'Vinohrady', 'Worcester',

```

Figure 3.2. This figure shows the printed unique values that appear in the “Location City” column in the dataset. Nottingham is represented incorrectly by three forms, including “Nottingham”, “Nottingham ”, and “nottingham”. Swansea was represented by two forms: “Swansea” and “swansea”. Almería was represented incorrectly by “Almer??a”.

Secondly, some transaction records whose city is in the UK used empty values for the country name (Figure 3.3). To handle this, the data whose location city is Nottingham, Swansea or Bristol were fixed by setting their country as “Uk”.

```

check city-country incorrect pairs

```

```

def checkCityCountry():
    cities = list(df["Location City"])
    countries = list(df["Location Country"])
    assert(len(cities) == len(countries))
    uniquePairs = list(set(zip(cities, countries)))
    # check if there exists duplicate cities in the uniquePairs
    citiesFromPairs = [city for city,_ in uniquePairs]
    duplicatedCities = []
    uniqueCities = []
    for city in citiesFromPairs:
        if city in uniqueCities:
            duplicatedCities.append(city)
        else:
            uniqueCities.append(city)
    print("duplicatedCities:", duplicatedCities)
    return [(city, country) for city, country in uniquePairs if city in duplicatedCities]
checkCityCountry()

```

[42] ✓ 0.0s

```

... duplicatedCities: ['Bristol', 'Swansea', 'Nottingham']

... [('Nottingham', 'UK'),
      ('Bristol', nan),
      ('Bristol', 'UK'),
      ('Swansea', nan),
      ('Swansea', 'UK'),
      ('Nottingham', nan)]

```

Figure 3.3. The checkCityCountry function returns the cities that have incorrect country names. Some transactions in Nottingham, Bristol and Swansea have the wrong country value.

4 Project Specification

This section presents the features and technology choices.

4.1 Feature Specification

This section introduces the must-have features and optional features of iBankEx as suggested by the project guideline (Laramee, 2021). As the program is developed incrementally, the features are organised to many small features to make it possible to put them into the project timeline (Table 5.1).

4.1.1 Must-have Features

1. Web-based UI
2. Calendar View with Year Selection
3. Basic Bar and Pie Glyph
4. Calendar View Glyph Option
5. Cluster View with Logarithmic Scale and Sliders
6. Transaction Amount View with Logarithmic Scale and Sliders
7. Table View for Detailed Data
8. Brush and Linking
9. Consistent Axis Style and Colour
10. Calendar View Detail-on-Demand
11. Transaction Clustering
12. Transaction Description Grouping
13. Colour Legends and Highlighting
14. Context and Focus
15. Transaction Frequency

4.1.2 Optional Features

1. Bar Glyph Reordering
2. Bar Glyph Bandwidth and Height options
3. Pie Glyph Radius Option
4. Polar Area Glyph
5. Polar Area Glyph Radius Options
6. Star Glyph
7. Star Glyph Radius Options
8. "Superpositioned" (Gleicher, 2018, p. 417) Calendar View
9. Calendar View Expanding
10. Transaction Amount View Axis Swapping
11. Transaction Amount View Expanding
12. Cluster View Expanding
13. Cluster View Colour Option
14. Cluster View Axis Options and Swapping
15. Table View Page Option
16. Table View Clear-All Button
17. Table View Sorting
18. Table View Synchronised Row Colour
19. Colour Legend Sorting
20. Border Colour for Brusher and Calendar View
21. Table View Consistent Radio Button Colour
22. Transaction Description Grouping Advanced Mode

4.2 Technology Choices

In this project, Python and TypeScript are used. The libraries used in this project are React.js, Redux.js, D3.js, Next.js, Material UI, Tailwind CSS, Fastapi, Uvicorn, Pandas, Numpy, Scikit-learn, Jellyfish, SciPy. SVG and Canvas are strategically used for rendering the charts. Tools like Git, GitHub, Vscod, Prettier, “ES7 + React/Redux/React-Native snippets”, Docker, FileZilla, Vultr, and Node Package Manager are used.

Section 4.2.1 to section 4.2.4 introduces the technologies and explain the reason for using them.

4.2.1 Programming Language

Python and TypeScript are the programming languages used in iBankEx. TypeScript is a programming language built upon JavaScript (*JavaScript With Syntax For Types.*, n.d.), and JavaScript is an interpreted non-typed programming language running in a browser or another environment such as node.js. TypeScript provides type checking (Figure 4.2.1.1) and code auto-completing suggestions (Figure 4.2.1.2). Python is an interpreted programming language with plenty of libraries. In the present project, Python is used for data manipulation, including data loading, cleaning, clustering, and aggregation, and for the API server, which returns transaction and cluster data. TypeScript code is compiled into JavaScript code and running in the browser for fetching data, data manipulation, state management logic, visualising, etc. There are three reasons for choosing them.

The first reason is that they share the nature of the “best tools”, which are “(1) open source, (2) cross-platform, and (3) provide a forum in which to ask questions” (Laramee, 2021, p. 7). Python and TypeScript are open-source (Wagner, 2018; *Welcome to Python.Org*, 2023). Python is cross-platform, as it can be run on Windows, Linux and MacOS (*Python Operating Systems List*, n.d.). TypeScript is also cross-platform, as it can be run in a browser. Additionally, by using frameworks like React Native, TypeScript can be used for building mobile applications (Brito et al., 2018). Moreover, TypeScript can also be run in a node.js environment, which can be run on Linux, Windows and macOS. Finally, TypeScript and Python have strong community support and forums (e.g., discuss.python.org, stackoverflow.com/questions/tagged/typescript) for asking questions.

```
const user = {
  firstName: "Angela",
  lastName: "Davis",
  role: "Professor",
}

console.log(user.name)

Property 'name' does not exist on type '{ firstName: string;
lastName: string; role: string; }'.
```

Figure 4.2.1.1. Typescript type checking. If using JavaScript, the “user.name” value will be “undefined”, which might lead to unintentional error. TypeScript raises this error early in the code editor. Figure from *JavaScript With Syntax For Types.* (n.d.).

```
import express from "express"
const app = express()

app.get("/", function (req, res) {
  res.send
})

app.listen
```

send
sendDate
sendfile
sendFile

Figure 4.2.1.2. TypeScript provides auto-completing suggestions based on the properties of an object. Figure from *JavaScript With Syntax For Types*. (n.d.).

The second reason is the popularity of Python and JavaScript for data visualisation (X. Liu et al., 2023). Python libraries like pandas, numpy, and SciPy can help manipulate data. There are many Python visualisation libraries, such as ggplot, plotly and seaborn (Luraschi, 2015/2023). Meanwhile, there are many JavaScript libraries, like d3.js, chart.js, etc., for visualisation (Luraschi, 2015/2023). Moreover, many existing works used JavaScript. For example, Andrews et al. (2016) used JavaScript for geospatial visualisation, and Lavoué et al. (2013) used JavaScript for visualising compressed 3D data. TypeScript is a superset of JavaScript, which means those visualisation libraries can also be used when writing TypeScript code. The reason for using Typescript but not JavaScript is that Typescript can potentially increase development experience and productivity by type error detecting and code auto-completion (Fischer & Hanenberg, 2015; Gao et al., 2017; *What Are the Advantages of TypeScript over JavaScript?*, 2023).

The third reason is that they are the author’s most familiar programming languages, as the author had done previous coursework with Python and JavaScript, which is good for the prospect of project management. If using other suitable programming languages, such as Java for visualisation or C# for API, there might not be enough time to learn.

4.2.2 Libraries

It is worth noting that there are many existing JavaScript visualisation libraries, such as chart.js, plotly.js, and React-Vis (Luraschi, 2015/2023). The present project do not use these high-level chart library for three reasons. The first reason is to provide the flexibility for changing the code, as using d3.js with React and Redux allows implementing the features at a more detailed level than using high-level visualisation libraries. The second reason is that the author of the present project has previous project experience in building a d3-based app but has no experience in JavaScript high-level chart library. The third reason is to make the code more consistent to ease the change. For example, it is possible to render bar glyphs, pie glyphs and scatter plots using one existing library. However, if some new features are needed to be added and cannot be supported by the chosen library, the new library needs to be used, and it takes time to learn the new library. Therefore, all the charts and their features are chosen to be implemented by d3, React, and Redux.

The Graphic user interface is built with React.js, Next.js, Material UI.js, and Tailwind CSS. The underlying data for the GUI is managed with the help of Redux.js and d3.js. Dataset preprocessing and clustering are done by a Python API server which uses FastAPI, Uvicorn, Pandas, numpy, scikit-learn, jellyfish and SciPy.

React.js is an open-source library for building UI components using one-way data binding (data

can only flow from the parents to the children) (Ferreira et al., 2022; *React*, n.d.). Unlike plain HTML and JavaScript, react maintains and updates a virtual DOM first and only updates the real DOM when needed, leading to better performance (Dwivedi et al., 2022). The benefits of using react in the present project are 1) easier to write an app with better performance and 2) better maintainability and reusability due to componentisation, see Figure 4.2.2.1.

```
export default function FolderableContainer({ children, label, initIsFolded }: { children: React.ReactNode, label: string, initIsFolded: boolean }) {
  const [isFolded, setIsFolded] = useState(initIsFolded);
  const handleToggleFold = () => {
    setIsFolded(!isFolded)
  }
  return (<button className="mx-auto" style={{ width: '100%' }} onClick={handleToggleFold}>{isFolded ? 'Show ' : 'Hide ' } {label}</button>{isFolded && children}</>)
}

export function FolderableContainerInTable({ children, label, initIsFolded, colSpan }: { children: React.ReactNode, label: string, initIsFolded: boolean, colSpan: number }) {
  const [isFolded, setIsFolded] = useState(initIsFolded);
  const handleToggleFold = () => {
    setIsFolded(!isFolded)
  }
  return (
    <tr>
      <td colSpan={colSpan}>
        <button className="mx-auto" style={{ width: '100%' }} onClick={handleToggleFold}>{isFolded ? 'Show ' : 'Hide ' } {label}</button>
      </td>
    </tr>
    {isFolded && children}
  </>)
}
```

Figure 4.2.2.1. The code of FolderableContainer components. It has been reused many times.

Redux.js (for React) is a state management library for React apps (*React Redux* | *React Redux*, n.d.). It can centralise the state and provide them globally. The consumer of the state must access the state through the selector function and update the state by dispatch actions. This makes the app behave more predictable, debuggable and flexible (*Redux - A Predictable State Container for JavaScript Apps.* | *Redux*, n.d.). The reason for using redux is due to the app's interactive features, such as expanding the chart, brush and linking, selecting a day, etc. These features will lead to many states, see Figure 4.2.2.2. In this app, the official redux toolkit (*Redux Toolkit* | *Redux Toolkit*, n.d.) is also used, simplifying the creation of the states, reducers, and selectors.

```
// reference for all the exports: https://react-redux.js.org/tutorials/quick-start#install-redux-toolkit-and-react-redux
import { configureStore } from "@reduxjs/toolkit";
import barDayViewReducer from "@app/dashboard/components/CalendarView3/DayViews/barDayViewSlice";
import calendarViewReducer from "@app/dashboard/components/CalendarView3/calendarViewSlice";
import clusterViewReducer from "./dashboard/components/ClusterView/clusterViewSlice";
import colourLegendReducer from "./dashboard/components/colourLegend/colourLegendSlice";
import pieDayViewReducer from "./dashboard/components/CalendarView3/DayViews/pieDayViewSlice";
import polarAreaDayViewReducer from "./dashboard/components/CalendarView3/DayViews/polarAreaDayViewSlice";
import starDayViewReducer from "./dashboard/components/CalendarView3/DayViews/starDayViewSlice";
import scatterPlotReducer from "@app/dashboard/components/TransactionAmountView.tsx/scatterPlotSlice";
import interactivityReducer from "@app/dashboard/components/Interactivity/interactivitySlice";
import colourChannelReducer from "@app/dashboard/components/colourChannel/colourChannelSlice";
import popUpReducer from "@app/dashboard/components/PopupWindow/PopupSlice";
export const store = configureStore({
  reducer: {
    barDayView: barDayViewReducer,
    pieDayView: pieDayViewReducer,
    polarAreaDayView: polarAreaDayViewReducer,
    starDayView: starDayViewReducer,
    calendarView: calendarViewReducer,
    clusterView: clusterViewReducer,
    colourLegend: colourLegendReducer,
    interactivity: interactivityReducer,
    scatterPlot: scatterPlotReducer,
    colourChannel: colourChannelReducer,
    popUp: popUpReducer,
  },
});
```

Figure 4.2.2.2. Code for the Redux store. Code closely related to a feature is created in the same file.

D3.js is an open-source data visualisation library for the web app (*D3 by Observable* | *The JavaScript Library for Bespoke Data Visualization*, n.d.). It provides a novel and efficient way

for manipulating the DOM element. It provides convenient ways for creating axes and mapping data to visual form by scale functions such as from transaction amount to the location in the x-axis. However, React.js is also used in the present project, leading to a contradiction of DOM manipulation with d3.js. To solve this problem, one solution is applying the “useEffect”, and the “useRef” hook provided by React, which allows direct DOM manipulation, and another solution is to give up using D3’s DOM manipulation and only utilise its scaling function and use react to creating visual SVG elements (*Using React with D3.js*, n.d.). In this project, the latter is chosen as the former increases the risk of bugs in a React App by “conflicting with the changes React is making” (*Manipulating the DOM with Refs – React*, n.d.). Therefore, D3.js will only be used for creating scale functions (see figure 4.2.2.3), which maps data from the dataset to visual form, for generating SVG property such as path and line for the glyphs and for providing the locations and colour of the points for drawing points by Canvas API.

```
case "warm":
  interpolateFunction = d3.interpolateWarm;
  break;
default:
  // reference for exhaustiveCheck: https://www.typescriptlang.org/docs/handbook/exhaustive-checks.html
  const _exhaustiveCheck: never = colourScheme;
  throw new Error("this should never happen");
}
const colourRange = d3
  .quantize((t) => interpolateFunction(t), colourDomain.length)
  .reverse();
const scale = d3.scaleOrdinal(colourDomain, colourRange);
return scale;
}, [colourDomain, colourScheme]);
```

Figure 4.2.2.3. Code for creating scaling function for colour. With “quantize”, “interpolateXXXX”, and “scaleOrdinal”, a function that maps data to colour was created by d3.

Next.js (*Next.js by Vercel - The React Framework*, n.d.) is an open-source framework for developing React applications (Goel et al., 2023). As is currently suggested by the React official documentation (*Create React App*, n.d.), it is used in the present project for creating an environment for the React app conveniently by hiding the complexity of the configuration environment and optimising the production file. The present project also benefits from the hot reloading feature of the development mode (Figure 4.2.2.4) provided by Next.js.

```
PS D:\Projects\summer-project-msc\frontend> npm run dev
> frontend@0.1.0 dev
> next dev -p 80
- ready started server on 0.0.0.0:80, url: http://localhost:80
- event compiled client and server successfully in 809 ms (20 modules)
- wait compiling...
- event compiled client and server successfully in 266 ms (20 modules)
- wait compiling /dashboard/page (client and server)...
- event compiled client and server successfully in 12.7s (3660 modules)
- wait compiling...
- event compiled successfully in 1170 ms (1857 modules)
- wait compiling...
- event compiled client and server successfully in 2.3s (3660 modules)
- wait compiling...
- event compiled client and server successfully in 4s (3660 modules)
```

Figure 4.2.2.4. The development mode of Next.js can automatically re-compile and refresh the app when the code changes.

Material UI (n.d.) is an open-source GUI library for React-based apps. It provides a wide range of interactive UI components, see Figure 4.2.2.5. Using this can lead to a prettier user interface.

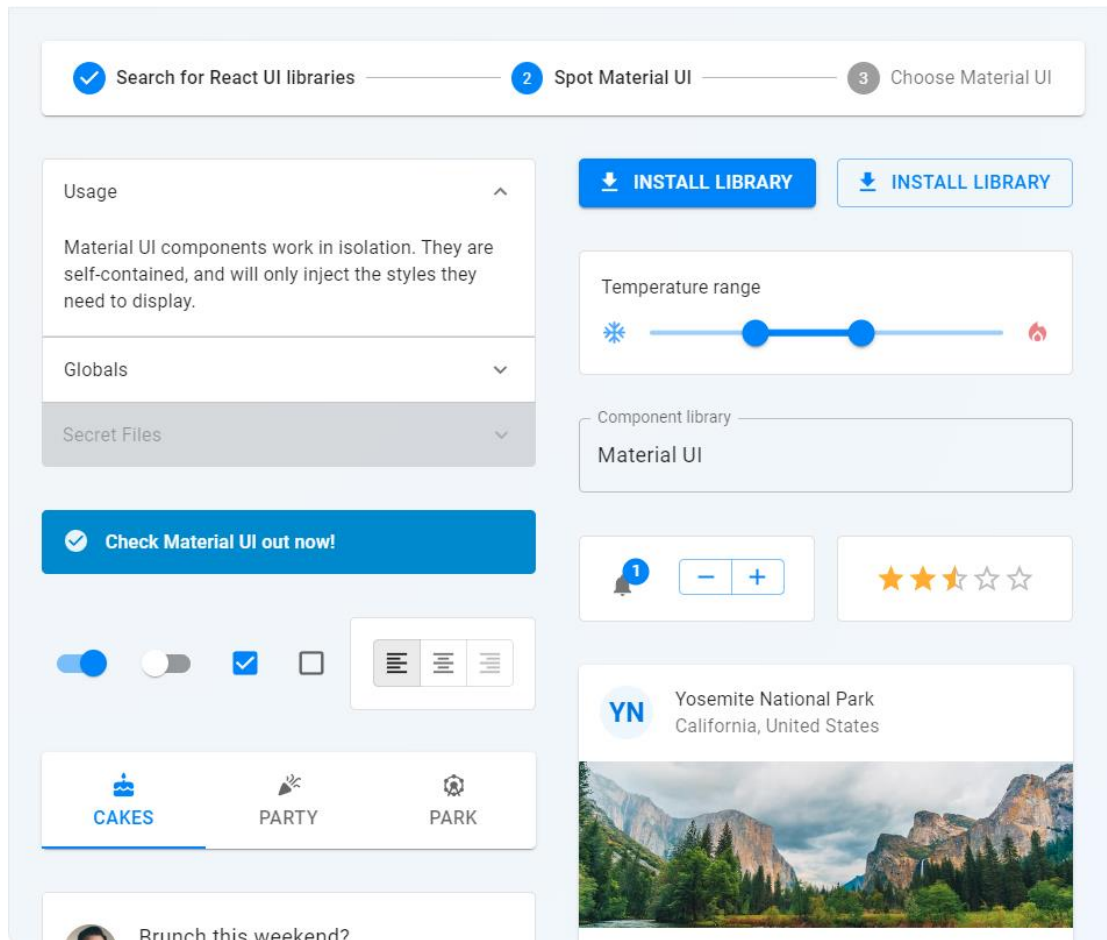


Figure 4.2.2.5. A screenshot of material components from *Material UI* (n.d.).

Tailwind CSS is an open-source CSS framework that provides predefined CSS classes (*Tailwind CSS - Rapidly Build Modern Websites without Ever Leaving Your HTML.*, n.d.). Tailwind can simplify styling an HTML element without writing a CSS class (Goel et al., 2023). Tailwind CSS (see Figure 4.2.2.6) and React CSS Properties (see Figure 4.2.2.7) will be used, as it is easy to use Tailwind for the style that will not change, and React CSS Properties can calculate the style dynamically.

```
<div className="grid grid-cols-12">
  <div className="col-span-5">
```

Figure 4.2.2.6. Example of Tailwind CSS used in the Project. The classNames are defined and provided by Tailwind.

```
return (
  <div style={{ paddingRight: isExpanded ? CALENDAR_VIEW_EXPANDED_LEGEND_WIDTH : CALENDAR_VIEW_FOLDED_LEGEND_WIDTH }}>
    <table className="smallLetterTable" >
      <thead>
        <tr>
          <td></td>
          {(Array.from(Array(31).keys())).map(i => <td key={i + 1}
            style={{ color: detailDay && detailDay.day === i + 1 && detailDay.year === currentYear ? 'red' : 'black' }}>
            {i + 1}
          </td>)}
        </tr>
      </thead>
      <tbody>
```


Figure 4.2.2.7. Example of using React CSS Properties Object: the first style is calculated based on the variable `isExpanded`, and the second style is calculated based on the detailed day.

FastAPI is an open-source, high-performance Python web framework for Application Programming Interfaces (Bansal & Ouda, 2022; *FastAPI*, n.d.). Comparing with Flask (i.e., another Python web framework), it is faster and better for handling huge amounts of data (Bansal & Ouda, 2022). It is used in this project for providing API service due to its performance and the author's previous coursework experience.

Uvicorn is an open-source Python web server framework. It is used in this project for running a FastAPI application. The reason for using Uvicorn is because it is one of the officially suggested ways of deploying the FastAPI application (*Run a Server Manually - Uvicorn - FastAPI*, n.d.), and the other reason is its simplicity and the author's previous coursework experience.

Pandas is an open-source Python library for data manipulation and analysis (*Pandas - Python Data Analysis Library*, n.d.). It can be argued that Pandas is the best Python library for data preparation (Mckinney, 2011; Stančin & Jović, 2019). Therefore, pandas is used in this project for data preparation tasks such as loading CSV data, transforming data, converting data to JSON, etc.

Numpy is an open-source Python library with good performance for array and matrix manipulation (*NumPy*, n.d.; Ranjani et al., 2019). It is used in the present project for matrix manipulation, see Figure 4.2.2.8, and for vectorising the distance functions, see Figure 4.2.2.9, leading to less code and better performance as the function provided by numpy can avoid writing slow Python loops.

Scikit-learn is an open-source Python machine-learning library (Hao & Ho, 2019; *Scikit-Learn: Machine Learning in Python — Scikit-Learn 1.3.0 Documentation*, n.d.). The present project uses it for normalising data and running the K-means clustering algorithm, see Figure 4.2.2.8.

```
def clusterByKMeans(self, metric1, metric2, numberOfCluster, maxIteration: int = 300, nInit=10):
    """
    assume metric1 and metric2 exist in the column names.
    run KMean clustering algorithm based on the two metrics
    if any metric is not int or float, they will be try to convert to float.
    set the clusterId column to be the result of clustering algorithm
    this method will mutate self.dataframe
    maxIteration should >VALID_KMEAN_ITERATION[0] and <VALID_KMEAN_ITERATION[1]
    """
    assert metric1 in self.dataframe.columns, f"{metric1} does not exist"
    assert metric2 in self.dataframe.columns, f"{metric2} does not exist"
    if (maxIteration < VALID_KMEAN_ITERATION[0] or maxIteration > VALID_KMEAN_ITERATION[1]):
        raise ValueError(
            'maxIteration should <1 and >2000, given: ' + str(maxIteration))
    if (numberOfCluster < 1):
        raise ValueError(
            'invalid number of cluster, it must be at least 1')
    # get the numerical value of two columns
    x1 = self.getColumn(metric1, toNumerical=True).to_numpy()
    x2 = self.getColumn(metric2, toNumerical=True).to_numpy()

    # reference for normalise the data: https://scikit-learn.org/stable/modules/preprocessing.html#normalization
    # normalise the data so that one of the dimension won't dominant the clustering algorithm
    x1Norm = sklearn.preprocessing.normalize([x1])
    x2Norm = sklearn.preprocessing.normalize([x2])

    X = np.dstack((x1Norm, x2Norm))[0] # type: ignore
    # run kmean
    # reference: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
    kmeans = KMeans(n_clusters=numberOfCluster, random_state=0,
                    max_iter=maxIteration, n_init=nInit).fit(X)
    # update the clusterId column
    self.dataframe['cluster'] = kmeans.labels_
    return True
```


Figure 4.2.2.8. The highlighted line used numpy to combine two vectors into a matrix. Scikit-learn (code named “sklearn” and “KMeans”) is used for normalising and clustering the data.

```
if vectorise:
    return np.vectorize(function)
else:
    return function
```

Figure 4.2.2.9. Code that used numpy to vectorise the function. The vectorised function can be applied efficiently on an array.

SciPy is an open-source Python library for scientific computing with algorithms that help optimisation, integration, interpolation, etc. (*SciPy*, n.d.; Virtanen et al., 2020). In the present project, SciPy is utilised for string clustering as it provides a convenient way to run linkage-based clustering algorithms, see Figure 4.2.2.10.

```
def __updateDistanceMatrix(self):
    """
    update self.distanceMatrix based on the preprocessedStringArray and distance function
    """
    vectorisedDistanceFunction = self._getDistanceFunction(
        self.distanceMetric, True)
    distanceMatrix = pdist(self.preprocessedStringArray,
                           metric=vectorisedDistanceFunction)

    self.distanceMatrix = distanceMatrix

def __updateLinkageMatrix(self):
    """
    update self.linkageMatrix based on the distanceMatrix and linkageMethod
    """
    # create linkage_matrix
    self.linkageMatrix = linkage(
        self.distanceMatrix, method=self.linkageMethod)
```

Figure 4.2.2.10. Code that uses SciPy for hierarchical clustering. Scipy provides the function “linkage” and “pdist”.

Jellyfish is an open-source Python string matching library with many string distance functions, including “Levenshtein Distance”, “Damerau-Levenshtein Distance”, “Jaro Distance”, “Jaro-Winkler Distance”, “Match Rating Approach Comparison”, and “Hamming Distance” (Turk, n.d.). Jellyfish implements the string distance functions (Figure 4.2.2.11), saving plenty of time.

```

def _getDistanceFunction(self, distanceMetric, vectorise=True):
    """
    Based on the distanceMetric, Return a string distance function that compares two string
    If vectorise is True, the returned function is vectorised.
    """
    function = None
    match distanceMetric:
        case 'levenshtein':
            function = jellyfish.levenshtein_distance
        case 'damerauLevenshtein':
            function = jellyfish.damerau_levenshtein_distance
        case 'hamming':
            function = jellyfish.hamming_distance
        case 'jaroSimilarity':
            function = jellyfish.jaro_similarity
        case 'jarowinklerSimilarity':
            function = jellyfish.jaro_winkler_similarity
        case 'MatchRatingApproach':
            def f(str1, str2):
                if jellyfish.match_rating_comparison(str1, str2):
                    return 0.9
                else:
                    return 0.1
            function = f
        case _:
            raise ValueError('invalid distance metric')
    if vectorise:
        return np.vectorize(function)
    else:
        return function

```

Figure 4.2.2.11. Code for using jellyfish library. A vectorised string distance function will be returned by the method `_getDistanceFunction`.

4.2.3 Rendering Mechanism

HTML is the markup language for web page rendering. JSX, integrated into React applications, merges HTML-like syntax with JavaScript (*Writing Markup with JSX – React*, n.d.). React application efficiently render web page by first updating the virtual DOM tree and then update the minimum number of nodes in the real DOM tree.

There are two popular rendering options for charts, Scalable Vector Graphics (SVG) and Canvas, which offer different advantages. SVG is a vector-based two-dimensional graphic markup language which can be bonded with DOM properties like event handlers and CSS styles (Lin et al., 2019). Canvas utilises the JavaScript API “getContext” to render at the pixel level (Lin et al., 2019), providing better performance than SVG (D. Li et al., 2018; X. Li & Bao, 2014), especially for many elements, as X. Li & Bao (2014) suggests (Figure 4.2.3.1). SVG is size-independent as it is vector-based, can resize without quality loss (X. Li & Bao, 2014). SVG is simpler as it is declarative and can bond with DOM (Lin et al., 2019). As a result, SVG shares great popularity in information visualisation research due to its simplicity (Chang et al., 2008), while using Canvas leads to “non-trivial complexity”(X. Li & Bao, 2014, p. 256).

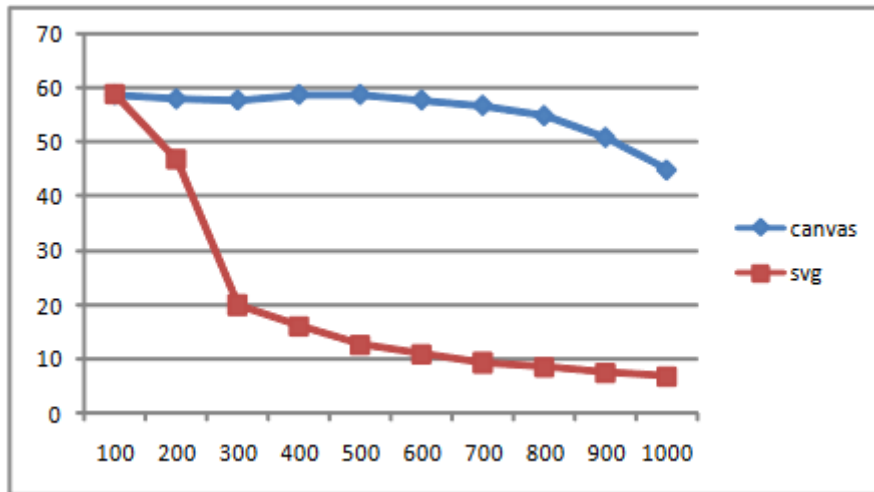


Figure 4.2.3.1. Result of the frame rate observation concerning the number of elements rendered by Canvas vs. SVG. THE X-axis represents the number of elements, and the Y-axis represents the frame rate. Figure from Li & Bao (2014).

In iBankex, Canvas is strategically employed for rendering scatter plots within the cluster view and transaction amount view, involving many data points and frequent updating. This approach ensures a smooth user experience while optimising development efforts. Rather than dedicating resources to enhancing individual glyphs through Canvas, the project prioritises broader improvements such as UI enhancements, debugging, and adding additional glyphs, including star glyphs and polar area glyphs, which yield more significant enhancements than write Canvas version glyph.

4.2.4 Tools and the Others

Git is a version control tool that can help roll back the versions and collaborate in a team. **GitHub** (Figure 4.2.4.1) is an online code repository for hosting the code. By using git and GitHub, code can be accessed on any computer. **Vscode** is a code editor that supports many extensions. It is useful in this project as it helps refactor the code, highlighting errors and provides a graphic user interface for git, see Figure 4.2.4.2. **Prettier** is a Vscode extension for formatting the number of letters in a line and the indentation size. “**ES7 + React/Redux/React-Native snippets**” is a Vscode extension with many hotkeys for code snippets, which can make the development faster by avoiding writing repetitive simple react codes. **Vultr** is a server provider. The server used in this project is provided by it. **FileZilla** is an FTP software with GUI that helps upload files to the server, see Figure 4.2.4.3. **Pylint** (*Pylint - Code Analysis for Python* | www.Pylint.Org, n.d.) and **Graphviz** (*Graphviz*, n.d.) are the tools used to generate class diagrams from Python code. **Docker** is an open-source “container virtualisation technology” (Anderson, 2015, p.102). It helps “build, share, and run container applications” (*Docker*, 2022). The present project uses docker to ease deploying a Python server. It ensures that the Python and the Python libraries are the right version, reducing the risk that the code works well on the development laptop but not on the Ubuntu server. Without Docker, extra efforts must be made to create a virtual environment for Python. **Node package manager** (*Run JavaScript Everywhere.*, n.d.) helps manage JavaScript package conveniently. With NPM, a line of “npm install” can install all the packages.

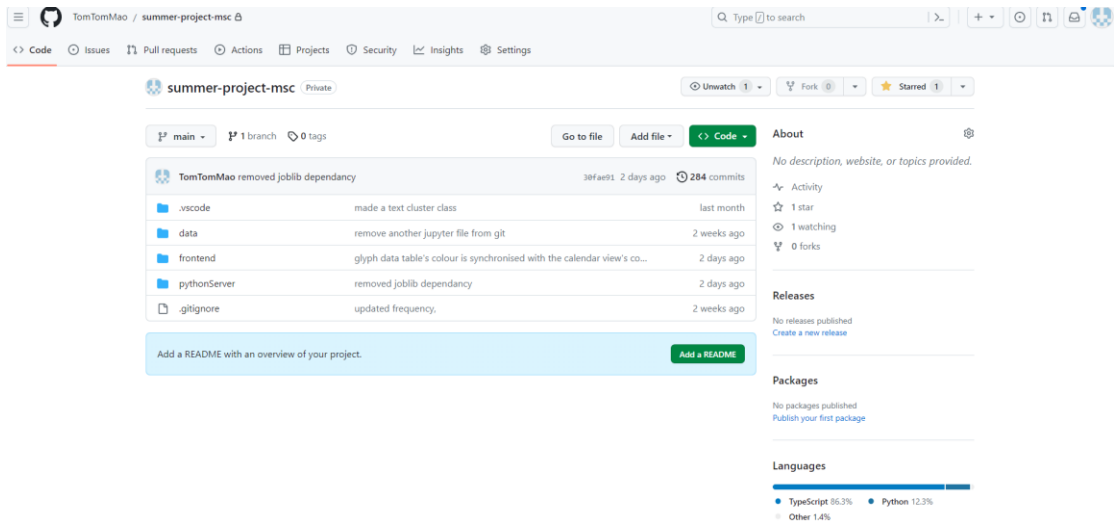


Figure 4.2.4.1. A Screenshot of the Github Page of iBankEx.

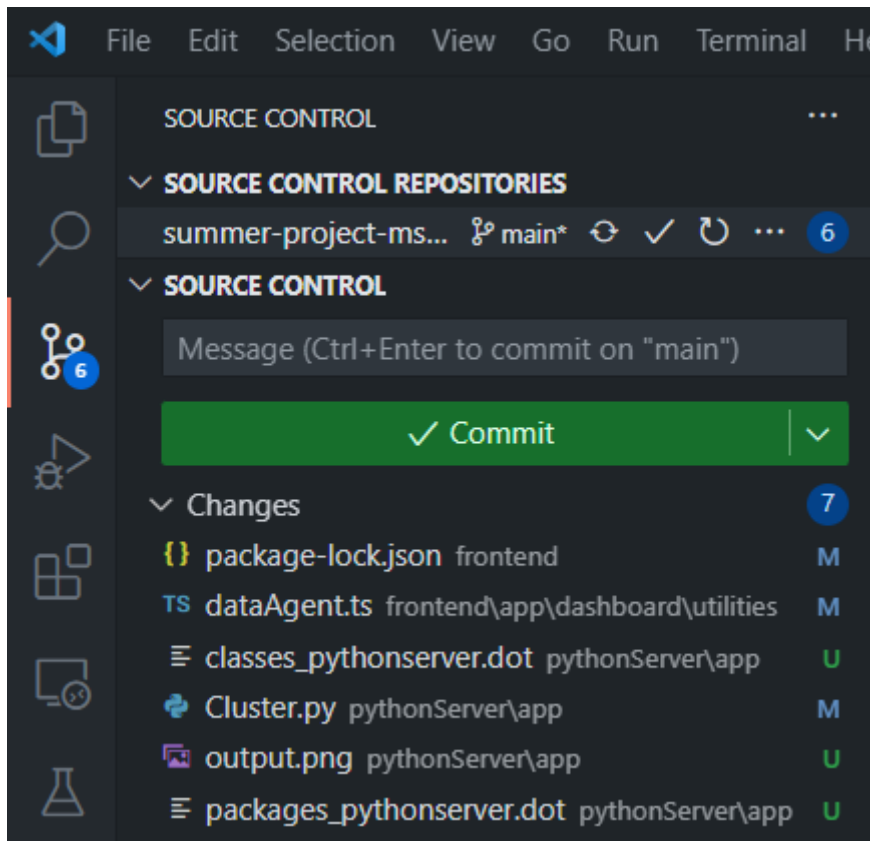


Figure 4.2.4.2. A Screenshot of the graphical Git user interface provided by Vscod.

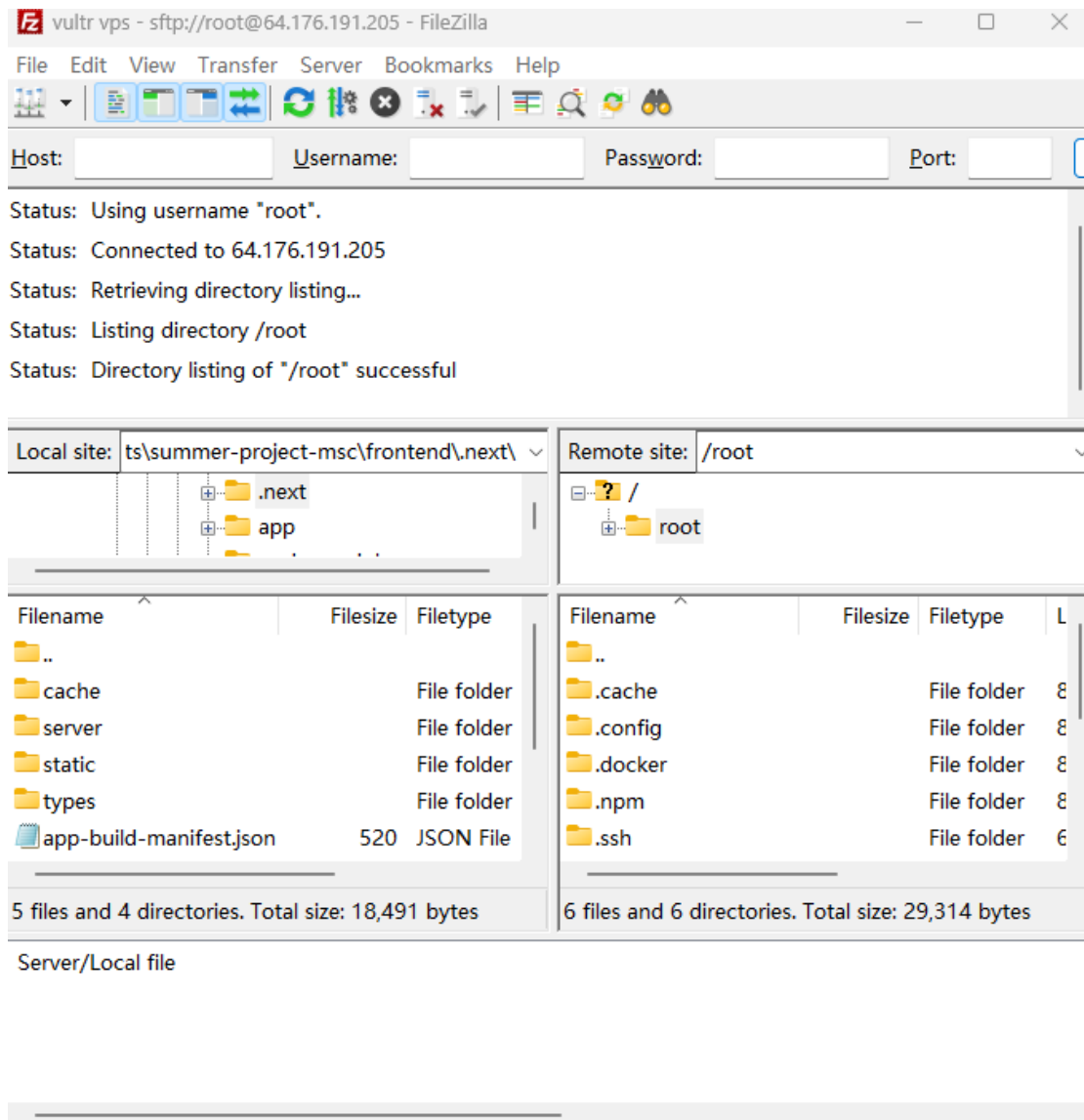


Figure 4.2.4.3. A Screenshot of Filezilla’s GUI. File can be uploaded into the server by dragging.

5 Project Plan and Timetable

The project is developed through an iterative process. After each meeting with the supervisor, what has been done and discussed were recorded in the meeting minutes (Laramee, 2021a), see appendix 1. As a result, subsequent works follow on the outcomes of the meeting.

The project's timeline and the completion dates are provided, as suggested by Laramee (2021). The project spans three months, from June to September. During June, the primary focus is conducting a literature review and getting familiar with the dataset. In July and August, the emphasis shifts to developing and refining features. Table 5.1 provides the detail of the completion date.

Table 5.1. Detailed Project Timetable. Table Style from Laramee (2021).

	Description	Date
1	Literature Review and Technology Review Initial Project Plan	July 5, 2023

2	Calendar View with Year Selection Table View for Detailed Data Calendar View Detail-on-Demand by Table Table View Clear-All Button Table View Sorting	July 12, 2023
3	Basic Bar and Pie Glyph Calendar view Control Panel Transaction Amount View with Logarithmic Scale Brushing and Linking between the Transaction Amount View and Detail View Bar Glyph Reordering, Bandwidth, Height Options Transaction Amount View Axis Swapping Context and Focus by Opacity	July 19, 2023
4	Calendar View Expanding Transaction Amount View Expanding Transaction Colour Legends and Highlighting	July 26, 2023
5	Calendar View UI Improvement Pie Glyph Radius Option	Aug 2, 2023
6	Python Server App Deployed by Docker Cluster View with Logarithmic Scale Transaction Clustering Cluster View Colour Option Cluster View Axis Option and Swapping	Aug 9, 2023
7	Cluster View Colour Legend and Highlighting Transaction Description Grouping Transaction Frequency Superpositioned Calendar view Further Calendar View UI improvement	Aug 16, 2023
8	Context and Focus by Greyscale Cluster View Expanding Border Colour for Brusher and Calendar view Transaction Description Grouping Advanced Mode Better Positioning for the Colour Legends Prettier Colour Scatter Plot Performance Improvement	Aug 23, 2023
9	Consistent Axis Style and Colour Colour Legend Sorting Star Glyph and Polar Area Glyph with Radius Option Table View Page Option Table View Consistent Radio Button Colour Table View Synchronised Row Colour	Aug 30, 2023
10	Table View Page Option Colour Legend Sorting	Sep 6, 2023

6 Project Design

This section will introduce the project's system design with diagrams, as Laramee (2021) suggested. Visualisation design will also be included to explain the visual encoding and interactivities. This section also provides a diagram for the system architecture, a class diagram of the API server, and an indented list showing the component hierarchy of the top-level components of the graphic user interface.

6.1 Visualisation Design

iBankEx (Figure 6.1) provides overview, zoom and filter, and detail-on-demand. Context and

focus, and multiple coordinate views are used. As multiple coordinated views share a similar concept with the “Analytic dashboard” (Bach et al., 2022, p. 3), their design trade-off and guidelines, such as not overwhelming the viewers and views’ consistency, were considered. As a result, efforts are taken to avoid too many views and reduce the screen size and number of pages, and the size of the chart is 1654.43px * 798.511px, which is suitable for a full-size browser on a 1920px*1080px screen. The consistency is considered so that the axis format for the same attribute between different views is the same, the colour of a given attribute between different views is also consistent, and the border colour of the brusher of the first radio button in the table view and of the glyph container are both blue and the border of the select glyph’s container and the second radio button in the table view (Figure 6.1D) are both red.

Sections 6.1.1 to 6.1.4 will introduce each view's visual design and interactivities.

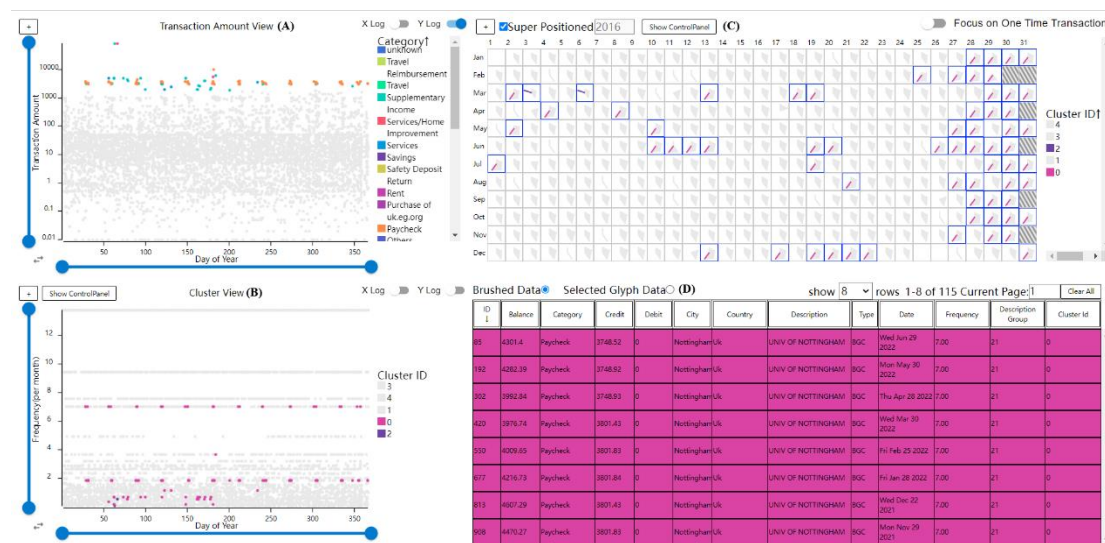


Figure 6.1. iBankEx consists of 4 interactive views. A) Transaction Amount View is designed to provide an overview of transaction amount vs. the day of the year. B) Cluster view allows the user to observe the cluster information concerning the different axes. C) Calendar view uses different glyphs to demonstrate the monthly pattern effectively. D) Table view shows the detail-on-demand. Both (A) and (B) provide sliders to control what to display.

6.1.1 Transaction Amount View

The transaction amount view, a scatterplot, aims to provide an overview of the temporal patterns of all the transactions. As Figure 6.1.1.1 illustrates, the x-axis represents the day of the year, and the y-axis represents the transaction amount. In this view, each point represents a row of data in the dataset, and its colour represents the transaction category. This view can visualise data at a lower abstract level than other charts, such as a bar chart and is less cluttered than a line chart.

The transaction amount view is highly interactive. A button (Figure 6.1.1.1B) can expand or fold the transaction amount view. It has a brusher (Figure 6.1.1.1A) with a blue border to select user-interested transactions. The other views will also focus on the data brushed in this view. It has an axis swapping button (Figure 6.1.1.1C) for swapping the x and y axis. It has two sliders for changing the range of the x and y, see Figure 6.1.1.1D&E. It has a sortable category colour legend where the user can click and highlight the category across all views, see Figure 6.1.1.1F. It has two radio buttons for choosing a logarithmic scale or linear scale for the x and y axis separately, see Figure 6.1.1.1G.

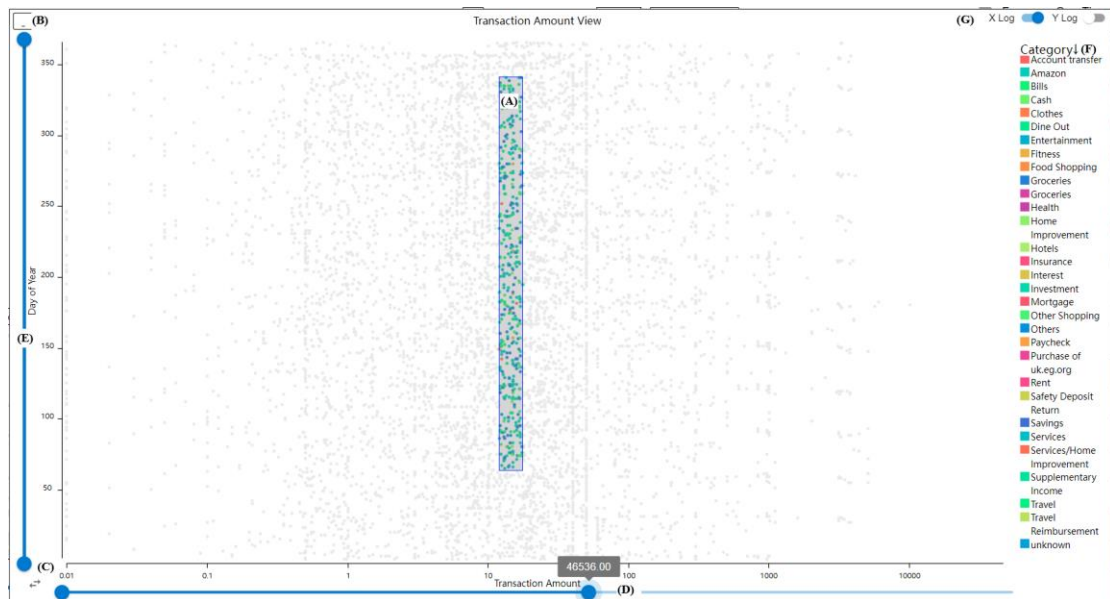


Figure 6.1.1.1. A Screenshot of the Expanded Transaction Amount View. (A) Brusher. (B) Expand/Fold button. (C) Swap the axis button. (D) Slider for the x-axis. (E) Slider for the y-axis. (F) Colour legend. (G) Buttons for controlling if use logarithmic scale for the x and y axis.

6.1.2 Cluster View

The Cluster View demonstrates the cluster information of transactions. As Figure 6.1.2.1 illustrates, by default, it maps x to the day of the year, y to frequency, and colour to cluster id. Each point represents a transaction. The reason for using scatter plots for cluster view is the popularity of scatter plots for visualising clustering results (Fuchs et al., 2019).



Figure 6.1.2.1. A Screenshot of the Expanded Cluster View. Note that the Transaction Group option in the middle was added after taking the screenshot.

The interactivity provided in the cluster view covers all the interactivity in the transaction view. Additionally, it has a control panel for observing data from different perspectives. The control panel has a visual mapping section for choosing the attribute for the x, y, and colour channels and two radio buttons for turning off/on the logarithmic scale for the x and y axis separately.

The control panel also has a clustering section which allows the user to choose the target number of clusters and the clustering metrics, including transaction amount, frequency (per month) and category. Moreover, transaction groups can also be specified by the user. The user can decide how the frequency is calculated based on the grouping option. Users can set the clustered transaction description, category, or initial transaction description as the key for grouping. Although the default parameters are chosen, it also provides an advanced mode for those who want to modify the string clustering algorithm's string distance measure, linkage method and number of transaction description clusters.

6.1.3 Calendar View

The calendar view demonstrates the information of each day's transactions using glyphs. Bar glyph, pie glyph, polar area glyph and star glyph are used in the calendar view, see Figure 6.1.3.1.

Bar glyph: Each bar is a transaction record. The height of the bar represents the transaction amount, and the bar's colour represents the category.

Pie glyph: Each fan represents a transaction. The angle of the fan represents the transaction amount, and the fan's colour represents the category. The radius of the pie represents the total transaction amount of the day.

Polar area glyph: Each fan represents a category. The angle is fixed. The radius of the fan represents the total transaction amount of the category. The colour of the fan represents the category.

Star glyph: Each bar represents a cluster. The colour of the bar represents cluster-ID. The length of the bar represents the total transaction amount of the cluster. The direction of the bar represents the cluster-ID too.

Like the cluster view, the calendar view also has a control panel with four sections for the four types of glyphs, see Figure 6.1.3.1.

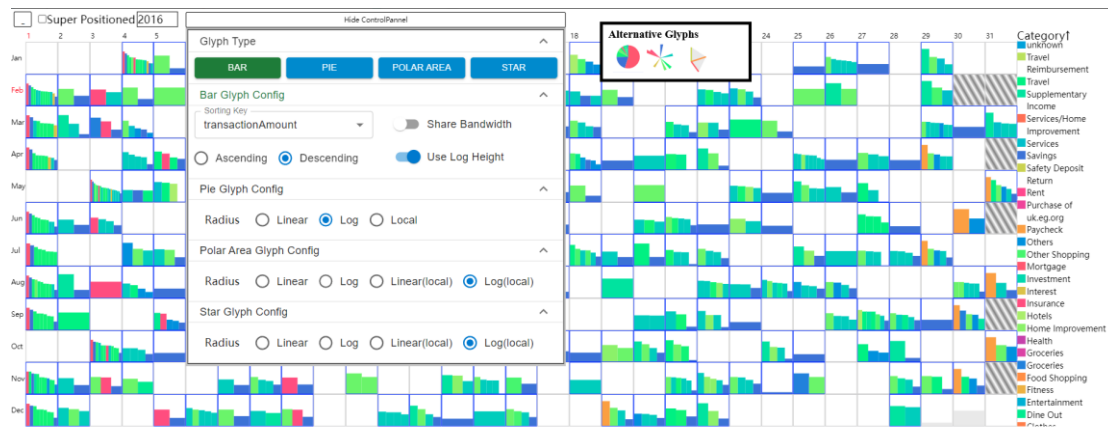


Figure 6.1.3.1. A Screenshot of the Expanded Calendar View. The calendar view currently displays a bar glyph. The glyphs in the black borders are the alternative glyphs, which were added manually on the screenshot. The user can choose which one to use and can configure the glyphs.

6.1.4 Table View

The purpose of Table View (Figure 6.1.4.1) is to show the detail-on-demand. It provides colours for each row of data. The colour can represent cluster ID, transaction description group ID, or

category. It depends on the last place the user selects or highlights data. The table shows all the columns in the dataset, but the column names are shortened for better display. The table consists of an “ID” column representing the unique “Transaction Number”, a “Balance” column representing the amount of money in the bank account, a “credit” column for the transaction amount of the incoming money, a “debit” column for the outgoing money, a “City” column and a “Country” column represents the location of the transaction, a “description” column for transaction description, a “Type” column for transaction type, a “Date” column for the transaction date, a “Frequency” column representing the number of transactions in the same description group happened per month, a “Description group” column representing the group that the transaction description belongs to, and a “Cluster Id” column showing the result of clustering algorithms with the selected attributes.

Several interactivities are provided. The rows can be sorted based on the selected attributes. The table can switch between the table for the brushed data or the selected glyph data. Buttons for changing the pages and number of rows are also provided at the top of the table. “8” and “17” are selected as two special number-of-rows options to fit the screen size well. If it shows eight rows, the screen size of the entire software will not change. If it shows seventeen rows, the table fits the screen height well. If users choose more rows to display, scrolling is supported for easily navigating to the target transaction, as they can sort the rows on any column and do not have to click the changing page button frequently.

Brushed Data Selected Glyph Data show 8 rows 57-64 of 64 Current Page:8 Clear All

ID	Balance	Category	Credit	Debit	City	Country	Description	Type	Date	Frequency	Description Group	Cluster Id
5238	18505.71	Amazon	0	3.99	Swansea	Uk	AMAZON UK RETAIL A	DEB	Wed Mar 01 2017	9.13	78	2
5254	16851.8	Amazon	0	3.63	Swansea	Uk	Amazon UK Marketpl	DEB	Tue Feb 21 2017	9.13	78	2
5985	11817.42	Amazon	0	2.75	Swansea	Uk	Amazon Svcs Europe	DEB	Mon Mar 14 2016	9.13	78	2
5988	11821.27	Others	0	3.49	Swansea	Uk	RJR-CC.COM 1 88846	DEB	Mon Mar 14 2016	1.00	188	2
5993	11725.13	Services	0	3.54	Swansea	Uk	ACTBLUE*BERNIE.SAN	DEB	Mon Mar 14 2016	0.22	136	2
6003	12970.06	Services	0	3.57	Swansea	Uk	ACTBLUE*BERNIE.SAN	DEB	Fri Mar 04 2016	0.22	136	2
6013	12414.27	Interest	4.74	0	Swansea	Uk	INTEREST (NET)		Tue Mar 01 2016	1.48	50	0
6034	9964.33	Amazon	0	4.75	Swansea	Uk	Amazon UK Marketpl	DEB	Mon Feb 22 2016	9.13	78	2

Figure 6.1.4.1. Table view sorted by ID. The current number of rows is 8, which makes the iBankEx not overflow the screen height. The description group determines the colour as the user just brushed the data on a chart representing the transaction description group by colour.

6.2 System Overview

Figure 6.2.1 presents the iBankEx’s system architecture at a top-level abstraction. This system is web-based, just like the system proposed by Camisetty et al. (2019), Ghosh et al. (2019) and Lin et al. (2019). The system consists of 2 parts, an API Server and a Next.js-based React-Redux APP. The API server is responsible for loading the MoneyVis dataset, caching data which needs expensive calculation (i.e., distance matrix for transaction description), providing cluster information, and providing API for the user interface. Meanwhile, the React-Redux App is responsible for fetching data from the server, rendering views based on the user options and the data, and providing interactivity.

As Figure 6.2.1 illustrates, when the user interacts with a view, the event handler will handle the event raised by the browser and update the corresponding states in the redux store. Then, once the state gets updated, the affected components will re-render themselves.

This architecture can separate the expensive calculation, such as distance matrix for string clustering, away from the browser, easing the installation process for user as it is web-based. It also improves the maintainability because the server, UI, data agent, state, event handlers, reducers, and selectors logics are separated.

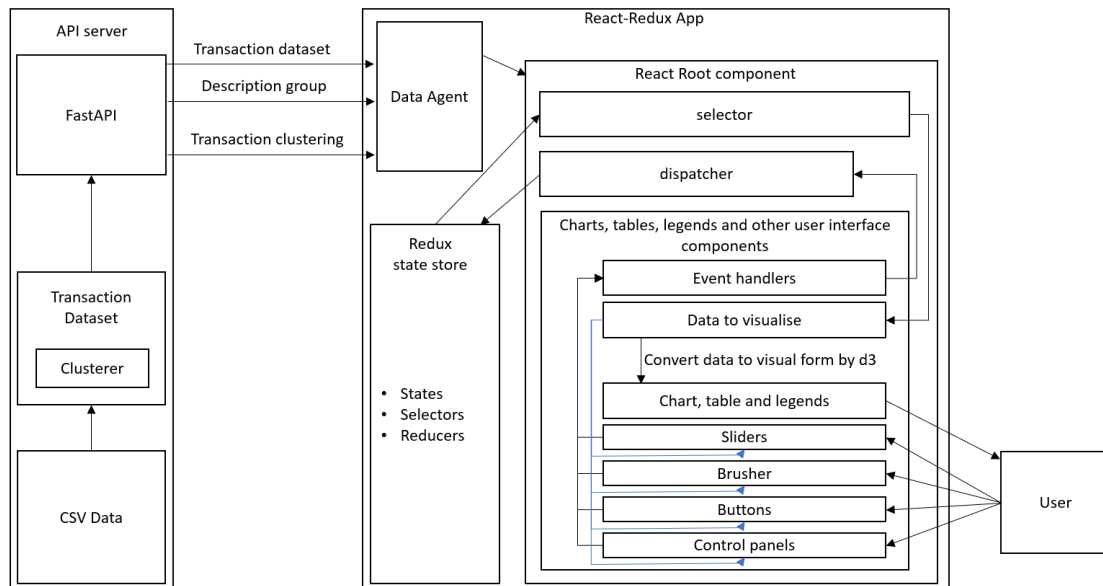


Figure 6.2.1. System overview of the iBankEx explaining the typical implementation logic of the features. The system consists of two parts. The API server app provides a transaction dataset, transaction description group and cluster-ID. The UI is powered by React and Redux libraries, providing views and interactivity. The user interacts with the features through the sliders, brushers, buttons, and control panels. Then, these components will call event handlers, which will dispatch actions. The redux state store will use a reducer to update the states based on the actions. Once the states get updated, the affected components will update themselves. Some states will let the root component (i.e., the App component) call API for updating the data. When the server receives an API call, it will operate the TransactionDataset object. If it is related to string clustering, the LinkageBasedStringClusterer object will be used. When the server is initialised, the CSV raw data will be imported into the TransactionDataset class.

6.2.1 API Server Design

There is an API Server that provides three simple APIs, which are “/transactionData”, “/transactionData/updateFrequencyInfo” and “/transactionData/kmean”. The API is just the functions decorated by “@app.get” in the main.py. These functions are the clients of the TransactionDataset class, which relates to the LinkageBasedStringCluster class and FrequencyOption class, see Figure 6.2.1.1.

When the “transactionData” API gets called, it will get the data from the TransactionDataset object by the getDataframe method and return the value in JSON format.

When the “/transactionData/updateFrequencyInfo” gets called, it will use the setFrequencyOption method provided by the TransactionDataset class, and then call the clusterByKMeans method provided by the TransactionDataset class, and finally call the getDataframe method provided by TransactionDataset to get the updated data and return the data in JSON format.

When the “/transactionData/kmean” gets called, it will call the clusterByKMeans method provided by TransactionDataset to cluster the data and call the

getClusterIdOfTransactionNumber method to get cluster information of each transaction. The information will be returned in JSON format.

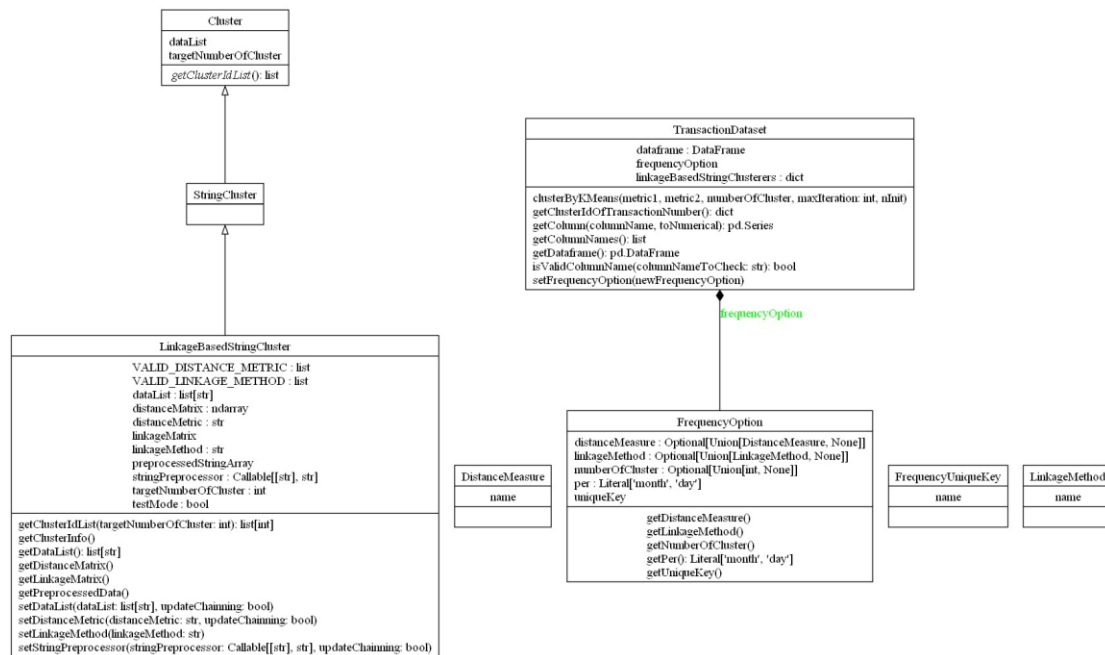


Figure 6.2.1.1. Class Hierarchy for the Python server application. Note there is a property of the TransactionDataset called “linkageBasedStringClusterers”, but it is a Python dictionary, so the 1-many relationships between TransactionDataset and LinkageBasedStringClusterers are not shown in the diagram. DistanceMeasure, FrequencyUniqueKey and LinkageMethod are the Enum types to constrain the parameters of the FrequencyOption’s constructor. TransactionDataset maintains the dataset and provides the ability to cluster the data and calculate the frequency. LinkageBasedStringCluster is used by TransactionDataset for grouping transaction descriptions. Graphviz generates the figure.

6.2.2 Web-based UI Design

The codes are for the web-based UI utilised React and Redux library, as they build reusable interactive user interface components. It does not use an object-oriented pattern. However, it is possible to provide a components hierarchy diagram similar to the class composition diagram suggested by Laramee (2021).

Figure 6.2.2.1 illustrates the is-part-of relationship between the top-level components using the indented list. Transaction amount view, calendar view, and cluster view are embedded in the ExpandableContainers, and the control panels are embedded in the FolderableContainers. The TransactionAmountView and the ClusterView both use LogScaleSwitcherGroup and InteractiveScatterPlot. The TableViewCollection renders and switches the table for the brushed and selected glyph data. There is also a Popup component used for showing feedback. The FormControlLabel is the button for showing one-time transactions. The communication between these high-level components is through the global data store provided by the top-level ReactRedux.Provider. A more detailed implementation will be introduced in section 7.



Figure 6.2.2.1. Components hierarchy tree for the web-based interface, generated by *React Developer Tools – React*, (n.d.).

7 Implementation

As Laramée (2021) suggested, this section is for the reader to understand how the features defined in section 3 are implemented and how to use iBankEx.

It is essential to briefly explain how React and Redux work, as they are the libraries used for implementing the software. The modern React use functional components, which are function that return JSX elements (syntax that like HTML). The components can maintain states, which will be checked before the re-rendering. The components can also receive properties. React will call the component’s render function when the state or the properties change. The returned JSX will be checked by React, and the virtual DOM will be updated. Then, the browser’s DOM tree will be conditionally updated based on what has been changed in the virtual DOM. Redux has a store for storing publicly shared data. Using the latest redux toolkit, the store can be created by separate slices like clusterViewSlice and calendarViewSlice for the closely related features and then put together into the store.ts file. In each slice, states, reducers, and selectors must be created. States are used for storing the information, reducers for updating the state based on actions, and selectors for getting data from the states with customised logic. The slices export the action creator and selector. The react components use AppSelector to select data from the store and use AppDispatcher to dispatch actions.

The rest of this section will explain how each feature is implemented. However, the thesis will not introduce every function's specification. The capitalised, camel-cased, and uppercased words are the variable names that can be searched in the source code (Appendix 4). Reviewing the source code when reading section 7 is suggested if the reader wants to re-implement iBankEx. Additionally, a demo video is provided at <https://youtu.be/TQSMgKitDu4>.

7.1 Basic Features

This section will introduce how the basic features introduced in section 4.1.1 work and are implemented. Additionally, the data structures like array and hash map that are used will be mentioned.

7.1.1 Web-based UI

The web-based UI can be accessed through <http://moneyvis.wentaom.com/dashboard>. It can be accessed through a modern browser like Chrome, Firefox, or Safari on either Windows or MacOS.

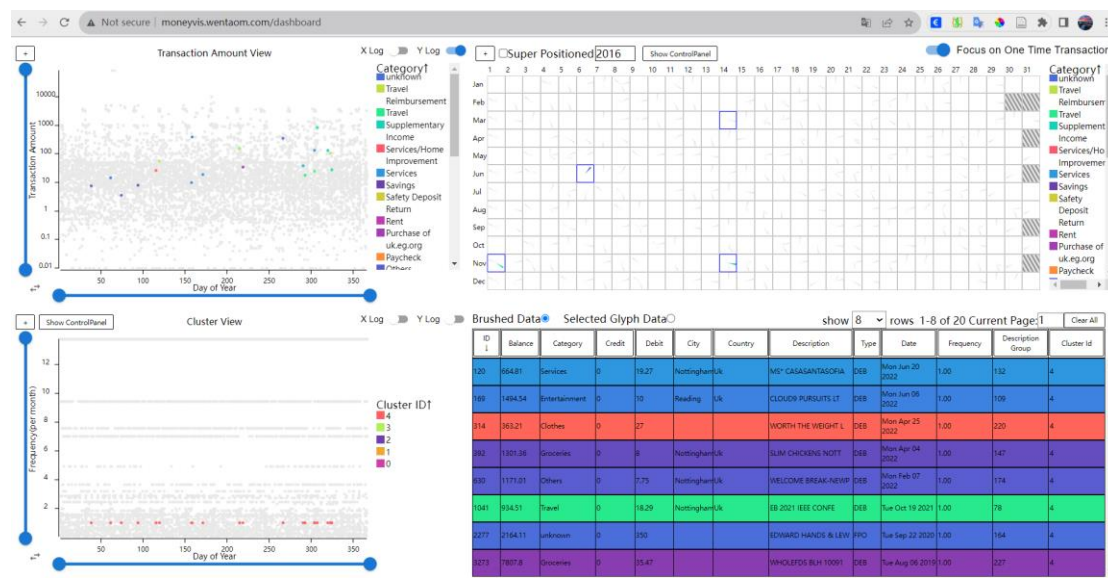


Figure 7.1.1.1. The page can be accessed through an URL: <http://moneyvis.wentaom.com/dashboard>.

To make sure the user can access iBankEx, web technology is used. The system can be divided into two parts, 1) a Python server for API calls and 2) a Next.js server for the user interface. The application is developed in a Windows environment and uploaded into a server rented from Vultr running Ubuntu 22.04x64 in Manchester with 1GB RAM, 25 GM NVMe storage and one virtual CPU. Google Domain Name Service provides the domain name.

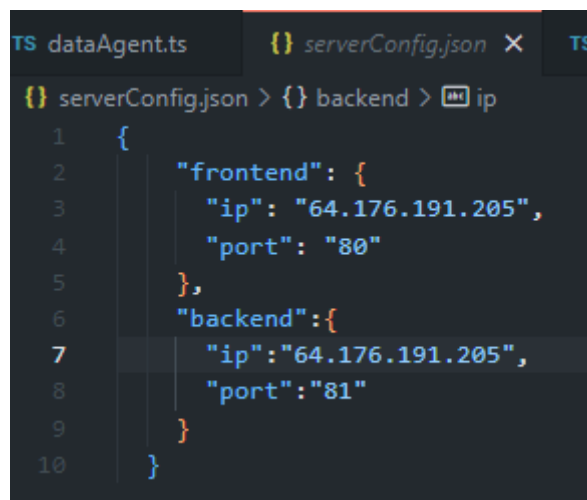
The server uses two port numbers, “81” for the API application and “80” for the user interface. To ensure that the user interface can fetch data from the server application, which runs in a different process, CORS is allowed in the API application, see Figure 7.1.1.2. The file serverConfig.json (Figure 7.1.1.3) must be created manually to set the server's IP address.

```

16  app = FastAPI()
17
18  # allow cors
19  # reference: https://fastapi.tiangolo.com/tutorial/cors/
20
21  app.add_middleware(
22      CORSMiddleware,
23      allow_origins=['*'],
24      allow_credentials=True,
25      allow_methods=["*"],
26      allow_headers=["*"],
27  )

```

Figure 7.1.1.2. Code for Allowing CORS. Requests from all the other process are valid.



```

TS dataAgent.ts  {} serverConfig.json X  TS
{} serverConfig.json > {} backend > ip
1  {
2      "frontend": {
3          "ip": "64.176.191.205",
4          "port": "80"
5      },
6      "backend": {
7          "ip": "64.176.191.205",
8          "port": "81"
9      }
10 }

```

Figure 7.1.1.3. A Screenshot of serverConfig.json.

Docker is used for the Python server to deploy the app more easily. The pipeline in the Dockfile tells Docker how to build the application, which is modified from the FastAPI deployment tutorial (*FastAPI in Containers - Docker - FastAPI*, n.d.). The deployment is by building the docker image for the Python server and then start the docker container on port 81 (Figure 7.1.1.4).

```

install docker following this tutorial:
https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04

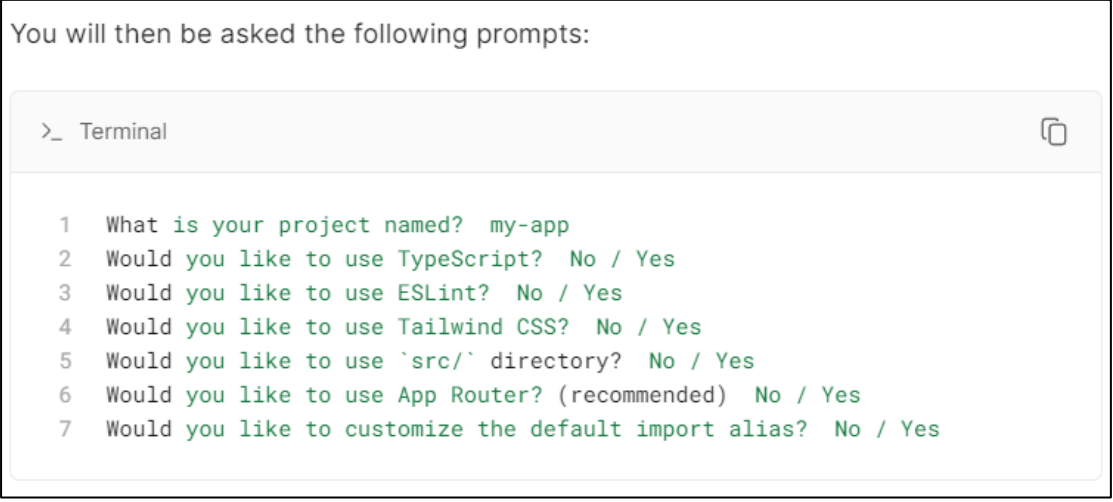
1. build pythonserver: docker build -t pythonserver .
2. run python server: docker run -d --name pythonserver -p 81:81
pythonserver

```

Figure 7.1.1.4. Screenshot of the steps for deploying Python applications.

However, the next.js server program does not use docker as the node package manager is simple to use. The server was created by the create-next-app (*API Reference*, n.d.) (Figure 7.1.1.5), which sets up an environment for the React app with the settings for TypeScript and Tailwind CSS and the command line code (i.e., “npm run build” and “npm run start”) for build and

deploy the app.



The image shows a terminal window titled ">_ Terminal" with a copy icon in the top right corner. The terminal displays seven prompts for the 'create-next-app' command, each followed by a user input. The prompts and their corresponding inputs are:

```
1 What is your project named? my-app
2 Would you like to use TypeScript? No / Yes
3 Would you like to use ESLint? No / Yes
4 Would you like to use Tailwind CSS? No / Yes
5 Would you like to use `src/` directory? No / Yes
6 Would you like to use App Router? (recommended) No / Yes
7 Would you like to customize the default import alias? No / Yes
```

Figure 7.1.1.5. Options for create-next-app. Figure source: (*API Reference*, n.d.).

The way to upload the application to the server is through either Filezilla or Git. Using git and build on the server takes less time, whereas uploading the already-built app takes longer. However, as the server has limited memory, sometimes the server cannot build the Next app (i.e., the UI). In this case, the Next app must be built on a better computer with npm and then uploaded to the server through Filezilla.

To connect the web-based UI with the API server, there is a function called `useSyncTransactionDataAndClusterData` in `useSyncTransactionDataAndClusterData.tsx`. This function will fetch the transaction data and the clustered id from the API server when the `frequencyUniqueKey`, `distanceMeasure`, `linkageMethod`, `numberOfClusterForString`, `numberOfCluster`, `metric1`, and `metric2` from the redux are updated. Once the data gets fetched, it will update the data stored in the redux store by dispatching the `setTransactionDataArr` and `setClusterDataArr` action defined in the `interactivitySlice`. The data stored in the redux store a linear array of `transactionData` objects and an array of `ClusterData` objects. All the AJAX request functions are stored in the file `dataAgent.ts` for better maintainability.

The App component holds the responsibility of synchronising the data.

7.1.2 Calendar View with Year Selection

The `CalendarView` is a child component of the App component. It receives `transactionDataArr` from the App and uses the `useClusterDataMap` function to get a hash map called `clusterDataMap`, which maps `transactionNumber` to the cluster id in $O(1)$. This function will only re-create the map when the `clusterDataArr` from the redux store is updated, avoiding repetitive expansive calculations.

This component rendered the data on an html-table-based layout. It first creates 12 components called `MonthView`, which renders glyphs of each day in a table row. The `CalendarView` will prepare the essential information for the `MonthView`, including the month of the month view, the current year of the calendar, and the data object with several transaction data maps which can help get the transaction data of a day or of a “superpositioned” day in $O(1)$, the scale functions (i.e., function which takes a domain value and returns a value for visual representation) for different glyphs which the `CalendarViewComponent` calculates.

The `MonthView` itself is a row of a table, see Figure 7.1.2.1A. When the `MonthView` renders

glyphs, it will check the glyphType to choose the glyph to render. Then, it passes the essential properties for the glyphs, including day, month, currentYear, the data object, and scale function. Then, the glyphs will render themselves based on the properties. Glyphs will be explained in 7.1.3 and 7.2.1 to 7.2.7.

The year control component (Figure 7.1.2.1C) is rendered by the CalendarViewYearController component. It will check the min and max number of years. Once the user changes the value, it will update the currentYear state in the calendarViewSlice in the redux store.

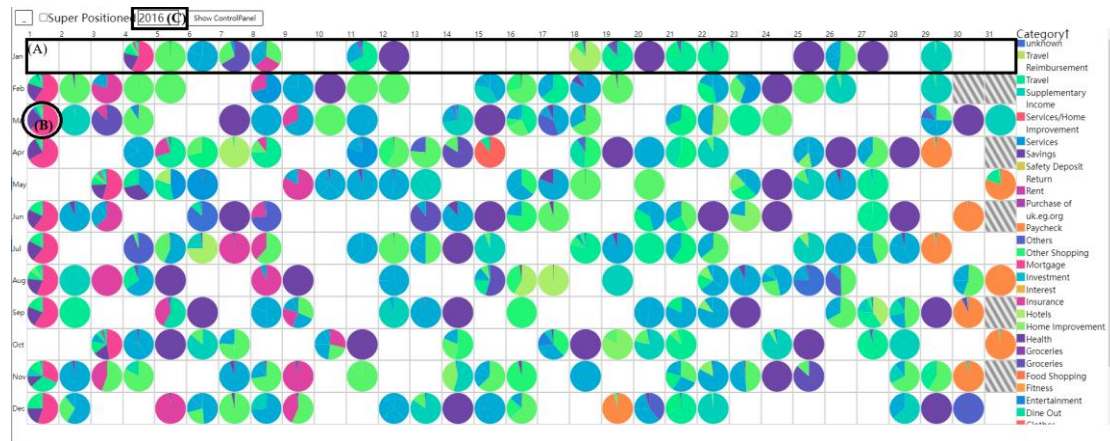


Figure 7.1.2.1. A Screenshot of the expanded CalendarView. (A) is what a MonthView component renders. (B) is a PieDayGlyph. (C) is the year selection.

7.1.3 Basic Bar and Pie Glyph

The bar glyph (Figure 7.1.3.1A) is rendered by the BarDayView component. To render the basic bar glyph, the component first gets the transaction data (an array) of the day based on the properties passed by its parent (i.e., monthView). Then, the data will be mapped to bars by passing the data to the scale functions. Sequentially, the mapped data will be put into the “<rect>” element as properties and rendered in an SVG element by the browser.

The pie glyph (Figure 7.1.3.1B) is rendered by the PieDayView component. Similarly, the components get the day's data to render the basic pie glyph. The logic for getting a day's data is refactored into the useCalendarDayGlyphTransactionDataArr function. Then, the radius of the pie is set the same as half of the containerWidth. Sequentially, d3.pie and d3.arc are used for preparing the properties of the SVG “<path>” element, which is rendered in an SVG element.



Figure 7.1.3.1. A) bar glyph. B) pie glyph.

7.1.4 Calendar View Glyph Option

The calendar view has a control panel (Figure 7.1.4.1) for choosing which glyph to use. It is implemented in the CalendarViewControlPanel component. The control panel is embedded in a FolderableContainer component. It has four buttons. The button for the selected glyph type is displayed as green colour. The state of what glyph to use is stored in the redux store's calendarViewSlice section. When the user clicks a button for the glyph type, the button raises an onClick event, handled by handleUseGlyph, dispatching the setGlyphType action with the chosen glyph as payload. Then, the reducer will update the state in the redux store based on the glyph type. Finally, when the state gets updated, the buttons will be re-rendered. The glyphs

config options will be introduced in the section 7.2.

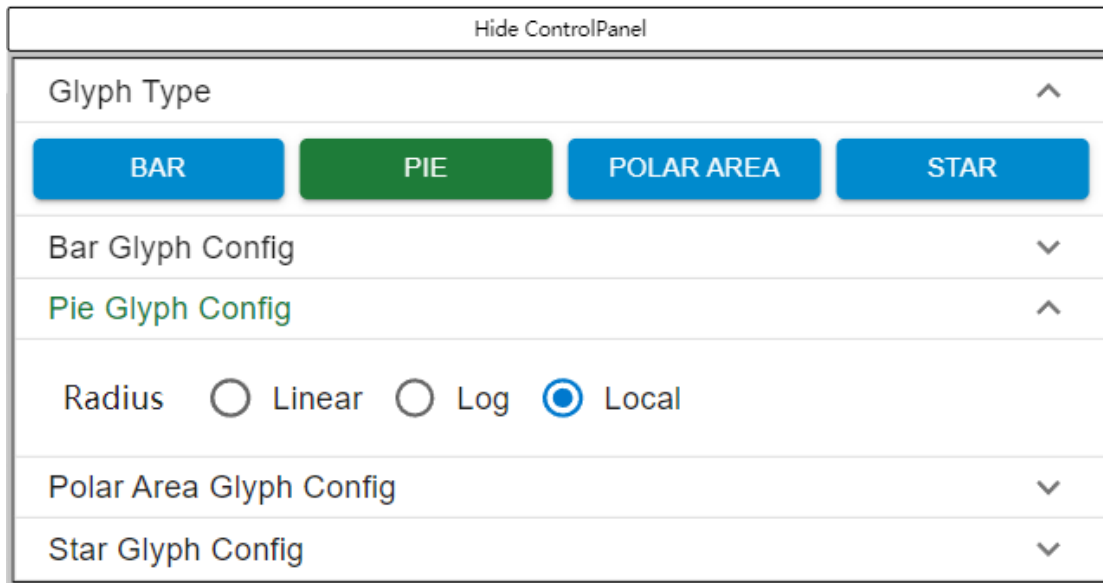


Figure 7.1.4.1. A Screenshot of the Calendar View Control Panel. The UI is based on Material UI.

7.1.5 Cluster View with Logarithmic Scale and Sliders

The component `ClusterView` receives its configurations, including the layout and the axis labels, from the redux store's `clusterViewSlice`. It also gets the dataset from the redux store. Instead of receiving an array of objects with multiple attributes, it stores each column in a single array. Additionally, the colour scale functions are provided by the App component. The Cluster View will check its `colourLabel` state and choose the right colour scale function. Then, those values are passed into `InteractiveScatterPlot` components and rendered by the `InteractiveScatterPlot` component.

```
// axis lable
const xlabel = useAppSelector(clusterViewSlice.selectXAxisLabel)
const ylabel = useAppSelector(clusterViewSlice.selectYAxisLabel)
const { categoryColourScale, clusterIdColourScale, frequencyUniqueKeyColourScale } = props.colourScales
const colourLabel = useAppSelector(clusterViewSlice.selectColourLabel) // for decide wich colour scale to use
const colourScale = colourLabel === 'category' ? categoryColourScale :
  colourLabel === 'cluster' ? clusterIdColourScale : frequencyUniqueKeyColourScale // can't be put in the st
```

Figure 7.1.5.1. This is the code for getting the axis information from the redux store. The code is part of the `ClusterView` component, which takes props from the App component. The third line retrieves the colour scales object from the props. The colour scale is determined by the colour label state by the ternary operation.

The information on an axis using a logarithmic or linear scale is stored in the redux store's `clusterViewSlice`. This information will be passed into an interactive scatterplot, too. The user can click the switch buttons (Figure 7.1.5.2A) to use the logarithmic scale, leading the redux store to update the corresponding state (i.e., `xLog` or `yLog`), the button for switching and the chart will be re-rendered.

The user can also change the range of the sliders (Figure 7.1.5.2B). As a result, the data rendered in the cluster view will be filtered. The slider component is implemented inside the `InteractiveScatterPlot` component to make it more usable. However, the states of the min and max values and of the sliders (i.e., `sliderXMin`, `sliderXMax`, `sliderYMin`, and `sliderYMax`) and the event handlers for changing the slider's value are provided by the `ClusterView` component.

When the user controls the slider, the event handler updates the state, dispatching the corresponding actions.

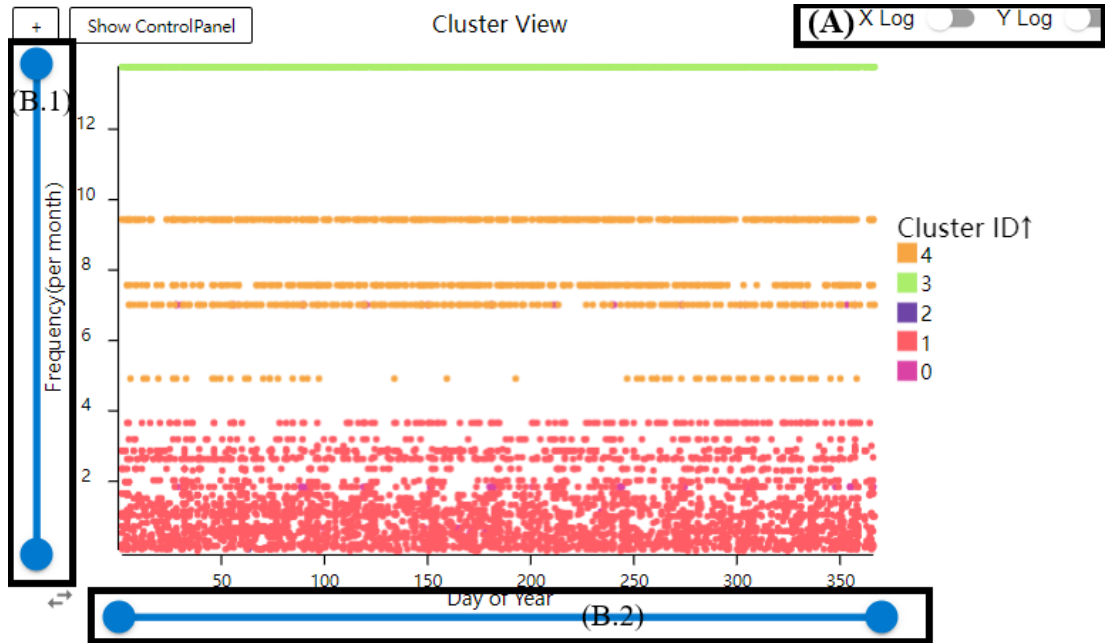


Figure 7.1.5.2. A Screenshot of Cluster View. (A) switch button based on Material UI. (B) sliders based on Material UI.

7.1.6 Transaction Amount View with Logarithmic Scale and Sliders

The implementation of the TransactionAmountView component is like the implementation mentioned by section 7.1.5. The difference is that the TransactionAmountView use the ScatterPlotSlice in the redux store, and the y-axis state is set to be transaction amount rather than frequency.

7.1.7 Table View for Detailed Data

The table view (Figure 7.1.7.1) uses an HTML table element for rendering the table for the detailed transactions. As there is no separate control panel for the table view, the state of the table view is only stored inside the TableView and TableViewCollection components rather than in the redux store. The TableView is wrapped by the TableViewCollection component, which allows the user to choose which table (table for brushed data or for the selected glyph) to display. It maintains a currentTable state for deciding which table to show. When clicking the table radio button (Figure 7.1.7.1A), the state will be updated to the corresponding value, and the table will be switched to the corresponding table.

To display the data, the TableViewCollection component receives properties, including an array of transaction data, a set of brushed transaction numbers, and a set of transaction numbers of the selected glyph from its parent App component. How these data are prepared will be introduced in 7.1.8. Additionally, it will use the useClusterDataMap function to get a map called clusterDataMap whose key is transaction number and value is cluster id. Then, transactionNumberSet for the transaction number of the selected transactions of the table and transactionDataArr are provided to the TableView components.

The TableView component renders charts based on the given data in an HTML table. The first thing to render is a row of column names. Then, it creates filteredTransactionDataArr, an array of transactions whose transaction number is in the transactionNumberSet. This array will be

looped through and then be rendered row by row. The colour feature will be introduced in 7.1.9 and 7.2.18.

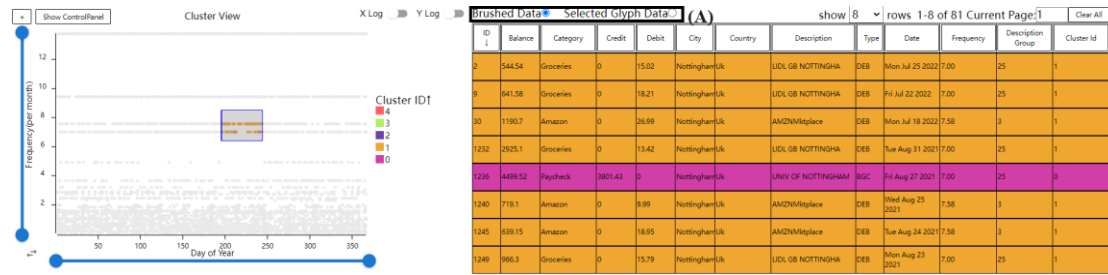


Figure 7.1.7.1. A screenshot of the table view and the cluster view. The table shows the detailed information of the transactions selected by the brusher in the cluster view. The colour in the table and the colour in the cluster view represent the cluster-ID. (A) The radio button for switching between tables.

7.1.8 Brush and linking

Brush and linking are implemented as a brusher in the cluster view and transaction amount view, see Figure 7.1.8.1A. The states, including `scatterPlotSelectedTransactionNumberArr` (for transaction amount view), `clusterViewSelectedTransactionNumberArr` (for cluster view), and `currentSelector` in the `interactivitySlice` in the `redux` store are used for the knowledge of the selected data.

The `selectedSelectedTransactionNumberArr` selector checks which brusher the user uses based on the `currentSelector` state and returns an array of the selected transaction number. The `selectSelectedTransactionNumberSet` use this selector to get the array and create a set of selected transaction number so that the search operation can be done in constant time. Then, the memorised version of `selectSelectedTransactionNumberSet` is provided to avoid returning two sets with identical elements.

The actions created by `setScatterPlotSelectedTransactionNumberArr` and `setClusterViewSelectedTransactionNumberArr` that are defined in the `interactivitySlice` will update the `selectedTransactionNumberArr` and the `currentSelector` states.

The brusher, implemented using the `d3` library, is in the `InteractiveScatterPlot` component, which receives the corresponding event handler `onSelectTransactionNumberArr` and the state `shouldShowBrusher`. The state controls if the brusher should display, and the event handler dispatches the `setXXXXXSelectedTransactionNumberArr` action to update the `brushedTransactionNumber` and the `currentSelector` state. When the user moves the brusher, the brusher will raise an event with the top-left and bottom-right locations information. The `handleBrush` event handler will handle the event. Then, in the `handleBrush` event handler, the locations are converted to the corresponding value in the dataset, and the dataset will be filtered to get the transaction number of the transactions in the value range. Finally, the `handleBrush` will call the `onSelectTransactionNumberArr` function to update the data in the `redux` state store.

Brush and linking is one of the most difficult parts. Because when the axis swaps or the slider moves, the log scale changes or the chart gets expanded or folded, the `d3` brusher will not update automatically as it is not a `react` component. To handle this problem, a state maintained by the `InteractiveScatterPlot` component named `lastBrushValueExtent` is introduced. When the user moves the brusher, it stores the top left and bottom right domain values. When the chart gets changed (i.e., one of the following values in the `InteractiveScatterPlot` changed:

width, height, xLog, yLog, xLabel, yLabel, xArr, yArr, filteredXDomainMin, filteredXDomainMax, filteredYDomainMin, filteredYDomainMax), the brusher will move to the right place by the API brusher.move based on the x and y values calculated by the new scale functions and the lastBrushValueExtent state.

The views are linked, so the other views will also be updated to focus on the brushed transaction. In the scatter plot and the bar glyph, pie glyph, the brushed transaction will have their initial colour, and the transactions which are not selected will use the constant value “GREY1”. For the polar area glyph, if the fan’s corresponding category appears in the set of the brushed data of the day (or the superpositioned day), the colour will be the initial colour. Otherwise, it will be GREY1. The rule is similar for the star-glyph but for cluster-Id rather than category. In the table, only the brushed transaction will be displayed.

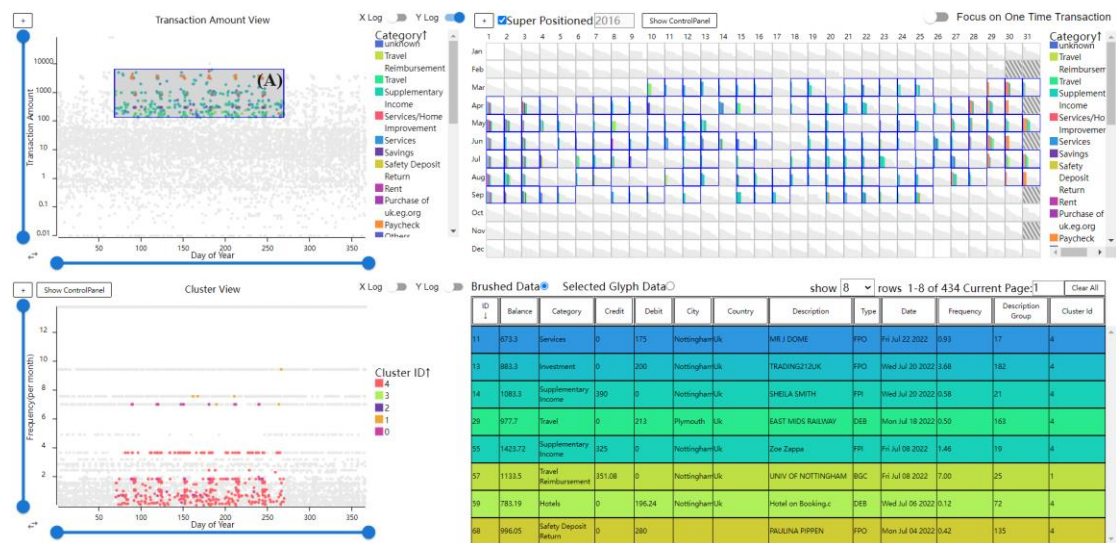


Figure 7.1.8.1. When brushing on the transaction amount view, the other views are updated. The grey elements are for the transactions that are not in the brusher. (A) The brusher. The background colour of the rows in the table represents the transaction category. The colour mapping for the categories in the transaction amount view, the calendar view and the table are the same.

7.1.9 Consistent Axis Style and Colour

Like Figure 7.1.8.1 presents, the axis ticks are consistent, and all the numbers are displayed as numbers without “k” (e.g., 10000 rather than 10k). They have at most two decimal point. Additionally, the colours of a given variable (i.e., cluster ID, transaction description group, or category) are the same between different views. This is implemented by preparing the colour scale function at the top level (i.e., the App component).

Moreover, the background colour of each row in the table view is set the same as the colour mapping of the user’s selected brusher or colour legend, which will be introduced by section 7.2.18.

7.1.10 Calendar View Detail-on-Demand

As Figure 7.1.10.1 illustrates, the day August 21 is selected. As a result, the transaction in the day is displayed on the table view. It can happen because the CalendarView component has a handleShowDayDetail function, updating the calendarViewSlice.detailDay state. This function will then be passed into the month view as the onShowDayDetail property. Then, the month view will put the onShowDayDetail in the <td> element of the table’s onClick event listener.

As a result, when the user clicks the <td> element, the detailDay state will be updated. Then, the App component will prepare the selectedGlyphTransactionNumberSet based on the detailDay, transactionDataArr and isSuperPositioned state. After that, the transactionNumberSet will be passed into the tableView, and the table will render the corresponding transactions. The colour scale function of the table for glyph is determined by the App component based on the colourLabelForTable data selected by the selectCurrentSelectorColourScaleType selector.

Additionally, tooltip is provided. When the user hover on a day, the transaction amount of each element in the chart will be displayed, see Figure 7.1.10.1. The contents of the tooltip are stored in the CalendarViewSlice, updated by the onHover event handler in the BarDayView, PieDayVieww, PolarAreaDayView and StarDayView components.

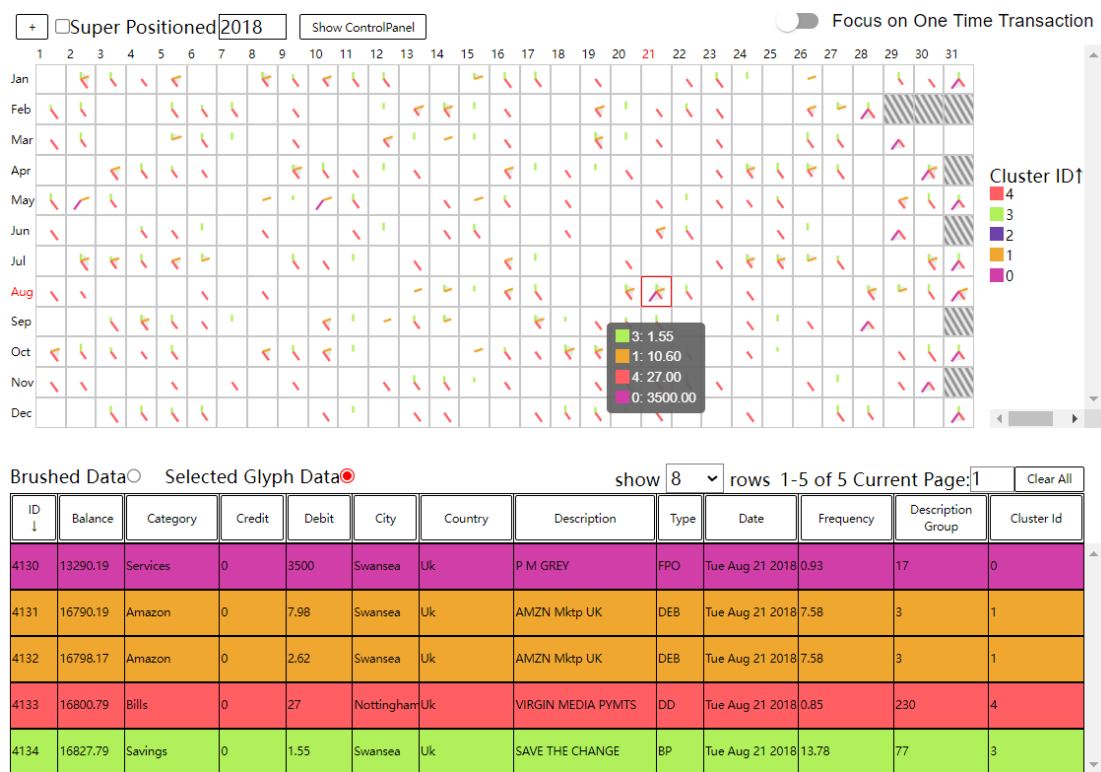


Figure 7.1.10.1. August 21, 2018, is selected. A red border highlights the glyph. The day and month in the calendar view are also highlighted in red colour. The detail of the transactions that happened on that day is displayed in the table view. The tooltip shows the transaction amount of each cluster.

7.1.11 Transaction Clustering

The clustering algorithm is implemented by the KMean algorithm due to its simplicity. It is implemented in the Python server application by the library scikit-learn. In the TransactionDataset.clusterByKMeans method, the transactions are based on the given metrics and number of clusters. To avoid one variable dominating the algorithm, the data are normalised. When a clustering API call comes to the server, the decorated function getClusterId in the main.py will call the transaction dataset.clusterByKMeans after validating the parameters. Then, it will return the cluster information of the transaction number in JSON format to the browser, as Figure 7.1.11.1 shows.

```

X Headers Payload Preview Response Initiator Timing
1 {
-   "1": {
-     "cluster": 4
-   },
-   "2": {
-     "cluster": 6
-   },
-   "3": {
-     "cluster": 7
-   },
-   "4": {
-     "cluster": 7
-   },
-   "5": {
-     "cluster": 3
-   }
- }

```

Figure 7.1.11.1. Cluster-ID of the Transaction Number returned by the API server.

In the user interface, the clustering has the following options: cluster metrics and number of clusters. These options are included in the ClusterAlgorithmControlPanel, as Figure 7.1.11.2 shows. Tooltips and feedback are also provided. When the user clicks update, the clustering metrics state and number of cluster states in the redux state store will be updated. As mentioned in section 7.1.1, the App component will detect the states and fetch the data through an API call. Once the data gets fetched successfully, all the other views will update as the state clusterDataArr is updated. A reset button is also provided to cancel any change before clicking the update button.

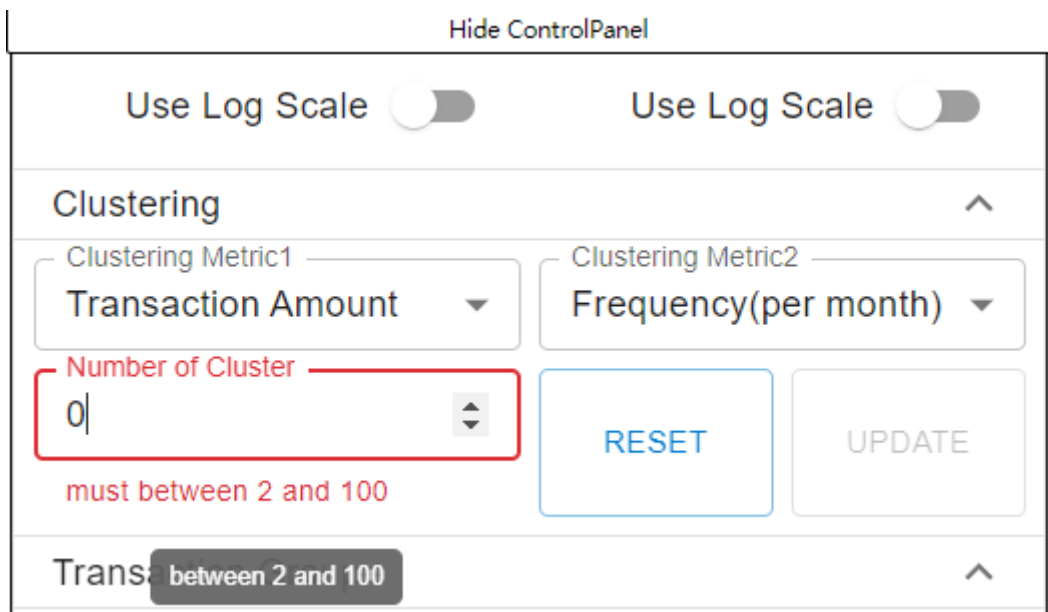


Figure 7.1.11.2. Transaction control panel for the transaction clustering option.

7.1.12 Transaction Description Grouping

There must be a unique value for grouping transactions to calculate the frequency. The first option is to use a transaction description. The second choice is to use category. The third way is to use clusteredTransactionDescription. The first two values are already there in the database. Whereas the third needs to be calculated on the server side. There is a

LinkageBasedStringCluster Class, which can use a LinkageBased clustering algorithm to cluster a list of strings. The transactionDataset uses this class to group similar transaction descriptions, see Figure 7.1.12.1. When the server receives an updateUniqueKey call, it will update the frequencyUnqiueKey information of the transactionDataset by calling transactionDataset.setFrequencyOption. Then, the KMeans algorithm for the transactions will be run because the change in the transaction description group might lead to a change in frequency, which might cause a change in cluster information.

Brushed Data Selected Glyph Data show 8 rows 1-8 of 76 Current Page:1 Clear All

ID	Balance	Category	Credit	Debit	City	Country	Description	Type	Date	Frequency	Description Group 1	Cluster Id
3259	6483.63	Cash	0	50	Swansea	Uk	LNK SWANSEA CITY C	CPT	Mon Aug 12 2019	1,11	29	0
3285	7903.64	Bills	0	147	Swansea	Uk	CC SWANSEA C.TAX	DD	Thu Aug 01 2019	1,11	29	0
3378	5611.82	Bills	0	147	Swansea	Uk	CC SWANSEA C.TAX	DD	Mon Jul 01 2019	1,11	29	0
3384	6191.92	Cash	0	100	Swansea	Uk	LNK SWANSEA 1	CPT	Mon Jul 01 2019	1,11	29	0
3474	7529.51	Bills	0	147	Swansea	Uk	CC SWANSEA C.TAX	DD	Mon Jun 03 2019	1,11	29	0
3518	5146.18	Cash	0	50	Swansea	Uk	LNK SWANSEA CITY C	CPT	Mon May 20 2019	1,11	29	0
3573	7787.57	Bills	0	147	Swansea	Uk	CC SWANSEA C.TAX	DD	Wed May 01 2019	1,11	29	0
3651	15084.75	Bills	0	145.69	Swansea	Uk	CC SWANSEA C.TAX	DD	Mon Apr 01 2019	1,11	29	0

Figure 7.1.12.1. A screenshot of the table view. The transactions with similar descriptions are clustered into the group 29.

Figure 7.1.12.2 shows these options at the user interface in the FrequencyControlPanel. The reset and update buttons are like those of the transaction clustering option explained in section 7.1.11. The data fetching logic and state updating logic are also like what happens after updating the transaction clustering option. Feedback is also provided, just like for the transaction clustering feature.

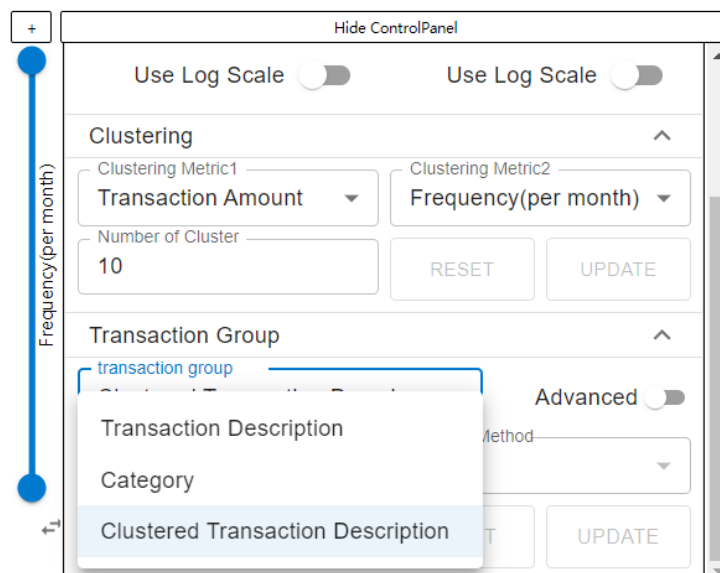


Figure 7.1.12.2. Transaction Group Options. The user can choose transaction description, category or clusteredTransactionDescription as the key for grouping transactions as one transaction for calculating the frequency.

7.1.13 Colour Legends and Highlighting

There are colour legends (Figure 7.1.13.1) for different variables. When the user clicks a rectangle or title in the colour legend, all the views will be updated so that only the data that matches the selected colour legend will be focused, see Figure 7.1.13.2.

The colour legends are rendered by the CategoryColourLegend component, FrequencyUniqueKeyColourLegend, and ClusterIdColourLegend components. They get the colourScale by the useXXXColourScale functions, which returns a colour scale object with a getColour method that maps the colour domain value to the colour string. These colour scales also have the information of the selected transaction number, and the getColour method will return the string stored in “GREY1” when the transaction is not selected. Then, the colour legends use the LegendList component to render the legends.

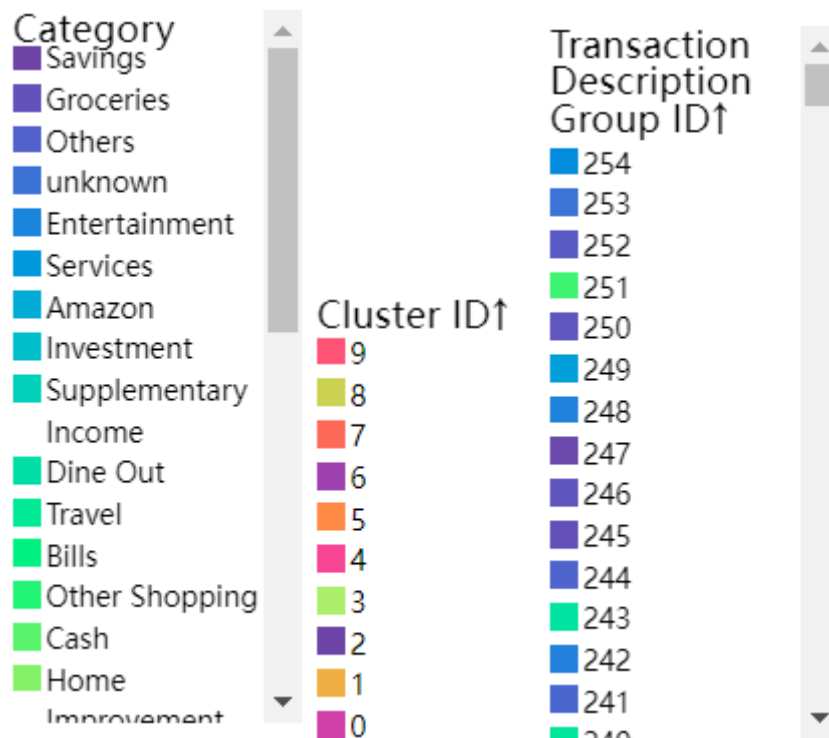


Figure 7.1.13.1. Colour Legends for the Different Variables.



Figure 7.1.13.2. The cluster ID 4 is selected. As a result, the transaction amount view and the calendar view set the colour of the visual elements for the transactions whose cluster-ID is not four as grey. The table view only shows the data whose cluster-ID is 4.

The highlighting is implemented by the function `handleToggleSelect`, which dispatch the toggle actions to update the `currentSelector`, `selectedCategoryArr`, `selectedFrequencyUniqueKeyArr` (for transaction description group) and `selectedClusterIdArr` state. The `currentSelector` will be set to be “category”, “clusterId”, or “frequencyUniqueKey” if the user clicks in the category legend, cluster id legend or transaction description group legend, respectively. They were used by the `selectSelectedTransactionNumberArr` mentioned before. Additionally, the `selectSelectedTransactionNumberArr` needs to have three new conditions (i.e., “category”, “clusterId”, and “frequencyUniqueKey”).

When the other view uses the `selectSelectedTransactionNumberSet` just like section 7.1.8 mentioned, based on the `currentSelector`’s value, the `selectSelectedTransactionNumberArr` return an array of transaction number of those satisfy category, clusterId or frequencyUniqueKey, which is displayed as transaction description group.

7.1.14 Context and Focus

As section 7.1.8 mentioned, brush and linking are supported. To highlight the brushed data, context and focus are used. As Figure 7.1.14.1 illustrates, the transaction in the brusher is highlighted, and the other visual elements are grey. This is implemented by giving the colour scale object the knowledge of the brushed transaction data number. So, when any chart tries to get data from the colour scale’s `getColour` method, it will return a grey colour for the context transaction. For the star-glyph, which plots transaction clusters, only the clusters with the selected transaction will not be grey. Similarly, for the polar area glyph, only the category with the selected transaction will not be grey.

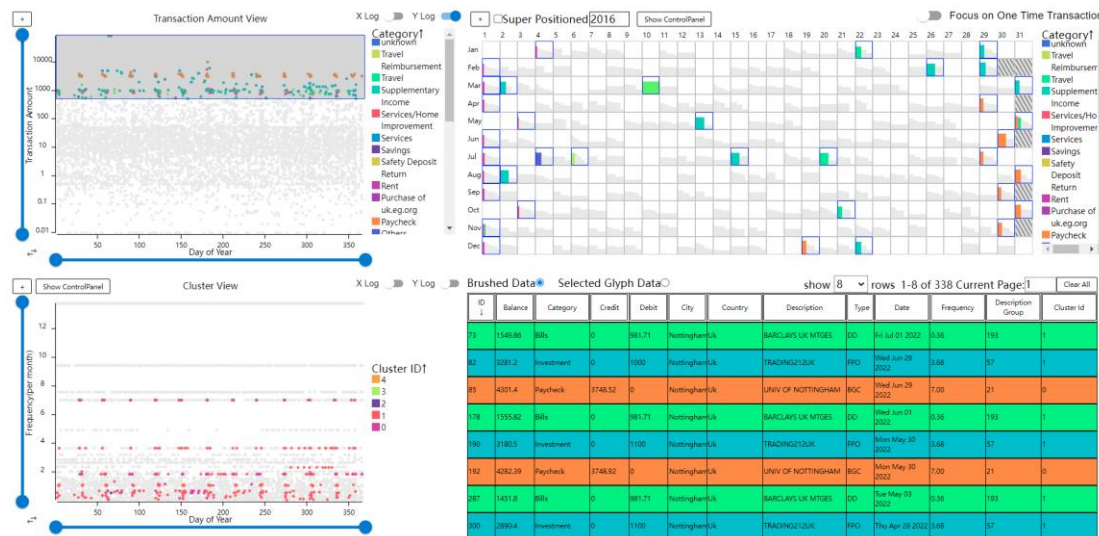


Figure 7.1.14.1. Example of Context and Focus. The context is rendered in greyscale.

7.1.15 Transaction Frequency

Transaction Frequency is calculated by dividing the number of transactions by the number of months between the first day and the last day the transaction happens.

There is some transaction that only happens for one time. To find them, they can be highlighted by the “focus on one time transaction” button, see Figure 7.1.15.1.

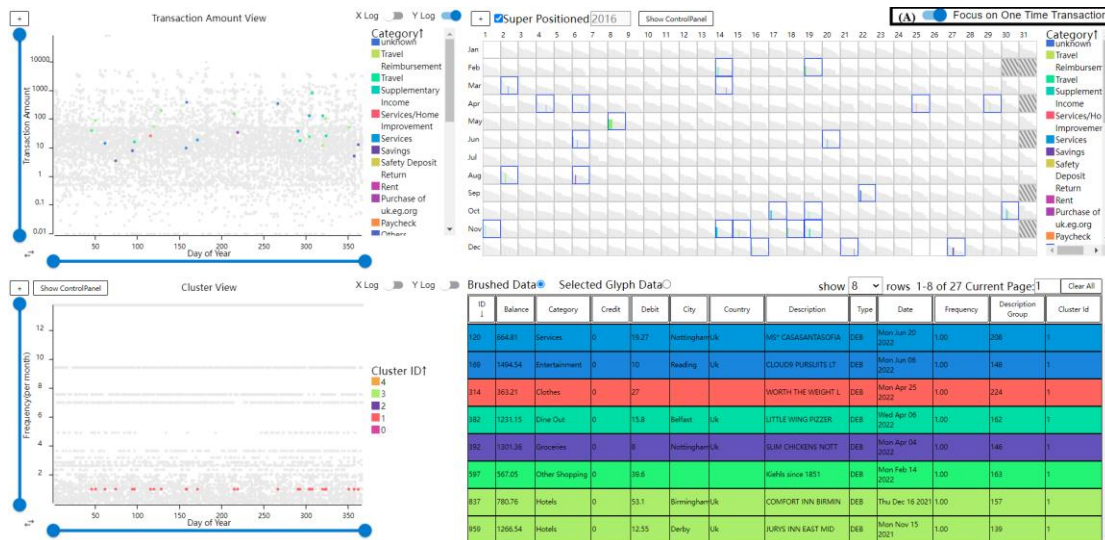


Figure 7.1.15.1. The views focus on the one-time transaction. (A) Button for showing the data.

Another possible value (i.e., “oneTimeTransaction”) of the currentSelector state is added to implement this feature. Additionally, selectSelectedTransactionNumberArr will call getOneTimeTransactionNumberArr (see Figure 7.1.15.2) to get the one-time transactions from all the transactions when the currentSelector is “oneTimeTransaction”. The one-time transaction button handles the event by dispatching the toggleShowOneTimeTransaction action for toggling the currentSelector value between “” and “oneTimeTransaction”.

```

function getOneTimeTransactionNumberArr(transactionDataArr: TransactionData[]):
Array<TransactionData['transactionNumber']>
/**
 * given
 * @param given a transaction data array, return a Arr of transaction number such that the transactionData with this number has a unique description group id
 * (or frequencyUniqueKey)
 * @return @param transactionDataArr
function getOneTimeTransactionNumberArr(transactionDataArr: TransactionData[]): Array<TransactionData['transactionNumber']> {

```

Figure 7.1.15.2. Specification of the getOneTimeTransactionNumberArr used by selectSelectedTransactionNubmerArr.

The API server calculates the frequency with the transaction group ID mentioned in 7.1.12. In the API server, when calling the TransactionDataset.setFrequencyOption method, not only the transaction description group column will be updated, but the frequency will also be calculated by the __updateFrequency method, which uses the __getFrequencyOfGroup method.

7.2 Optional Features

This section introduces the implementation of the optional features mentioned in section 4.1.2.

7.2.1 Bar Glyph Reordering

As Figure 7.2.1.1 shows, the order of the bars can be sorted. These options are in the BarDayViewControlPanel. It is implemented by letting the CalendarView sort the day's transaction data before creating the xScale in the BarDayView component. A comparator is needed to sort the transaction data generated by TransactionData. The curryCompare method takes an attribute name of the transaction data and the descending/ascending option and returns a comparator. The configuration is stored in the sortingKey and isDesc state in the redux store's barDayViewSlice. When the user changes the options in the calendar view control panel, the sortingKey and isDesc get updated by the actions that are dispatched in the event handlers (i.e., handleSetBarGlyphSortingKey and handleSetBarGlyphSortingOrder respectively).

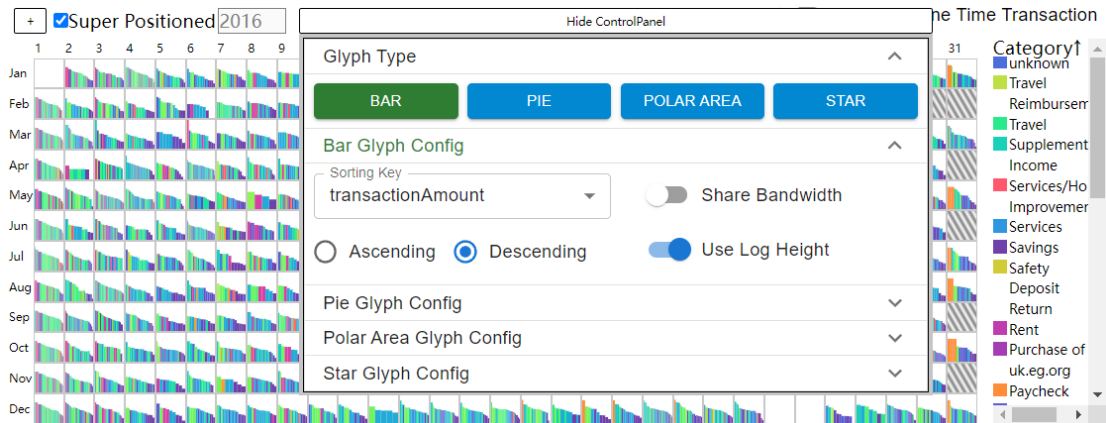


Figure 7.2.1.1. Bars get sorted by the transaction amount in descending order.

7.2.2 Bar Glyph Bandwidth and Height Options

The width of the bars can be the same across all the bar glyphs in Figure 7.2.2.1 or can be set to make the bars fill the container width like in Figure 7.2.1.1. The height of the bars can be either a logarithmic scale or linear scale, see figure 7.2.2.1. The user can control these options in the control panel implemented in the BarDayViewControlPanel.

To implement these, `isSharedBandWidth`, `heightAxis`, and `maxTransactionCountOfDay` states are stored in the `barDayViewSlice`. If shared bandwidth is true, the calendar view will calculate the maximum number of transactions of a day and store it in the `maxTransactionCountOfDay` state. Then, before the bar glyph calculates the x scale, the domain will be filled by the empty value to make the domain length the same as the `maxTransactionCountOfDay`. The `heightAxis` state will be checked by the bar glyph component. The linear height scale (i.e., `heightScaleLinear`) function and the logarithmic height scale (i.e., `heightScaleLog`) function will be prepared and put inside BarDayView's props. The BarDayView will use the `heightScaleLinear` function for converting the transaction amount into height if the `heightAxis` is linear. Otherwise, it will use the `heightScaleLog` function.

The options are in the calendar view control panel. When the user changes the option, the state will also be updated through the event handlers, which dispatch the corresponding actions, see Figure 7.2.2.2.

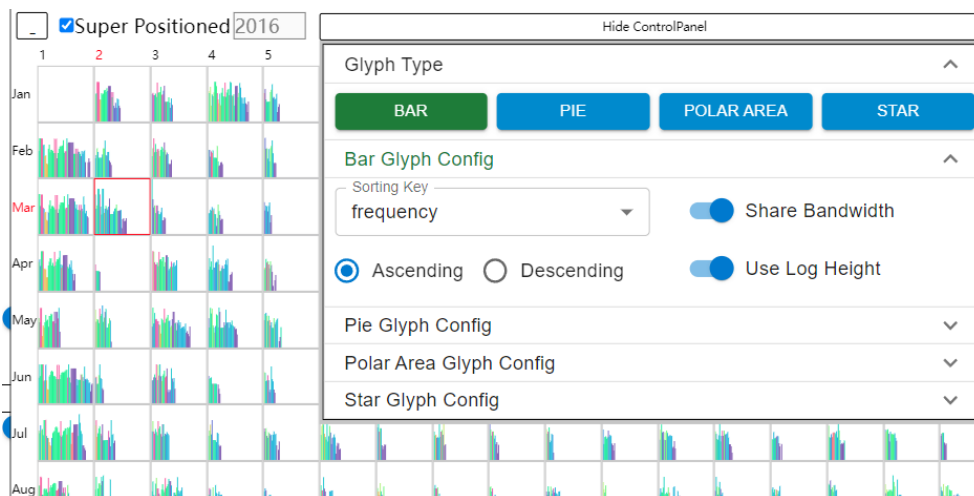


Figure 7.2.2.1. Screenshot of the Bar glyphs with shared bandwidth and logarithmic height scale. The user can control the bandwidths, height scale, sorting key, sorting order of the bars.

```

const handleToggleBarDayViewShareBandwidth = () => {
  dispatch(barDayViewSlice.toggleShareBandwidth())
}
const handleToggleBarDayViewHeightAxis = () => {
  dispatch(barDayViewSlice.toggleHeightAxis())
}

```

Figure 7.2.2.2. The event handler for the shared bandwidth feature and the uses-log-height feature of the bar glyph.

7.2.3 Pie Glyph Radius Option

The pie glyph can be set to use linear scale (Figure 7.2.3.1) or logarithmic scale (Figure 7.2.3.2) for radius or use fixed radius (Figure 7.2.3.3). This is controlled by the radiusAxis state in the PieDayViewSlice in the state store. The state updating logic is also to let the radio buttons for the options in the PieDayViewControlPanel use an event handler called handleSetPieGlyphRadiusAxis, which dispatches the setRadiusAxis action. When the calendar view renders the pie glyphs, it prepares both the linear and logarithmic scale functions for the pie glyphs. The pie glyph will retrieve the radiusAxis from the redux store and choose the corresponding scale function for generating the radius based on the radiusAxis state. If the radius option is local, then the glyph will use the width of the container size as the radius.

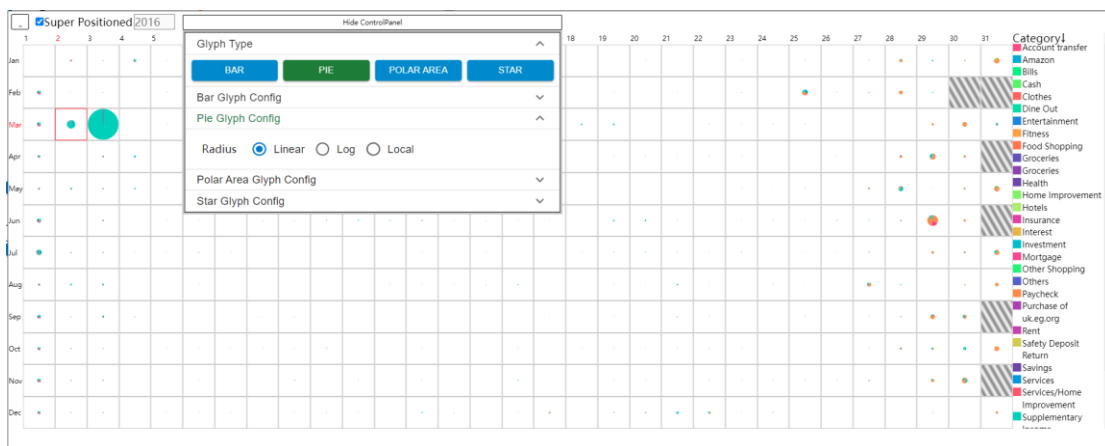


Figure 7.2.3.1. A screenshot of the expanded calendar view. The calendar view uses pie glyphs to show the transactions. Colour represents a category. Radius represents the total amount of transactions of each day. The radius uses a linear scale.



Figure 7.2.3.2. Pie glyph with logarithmic scale.

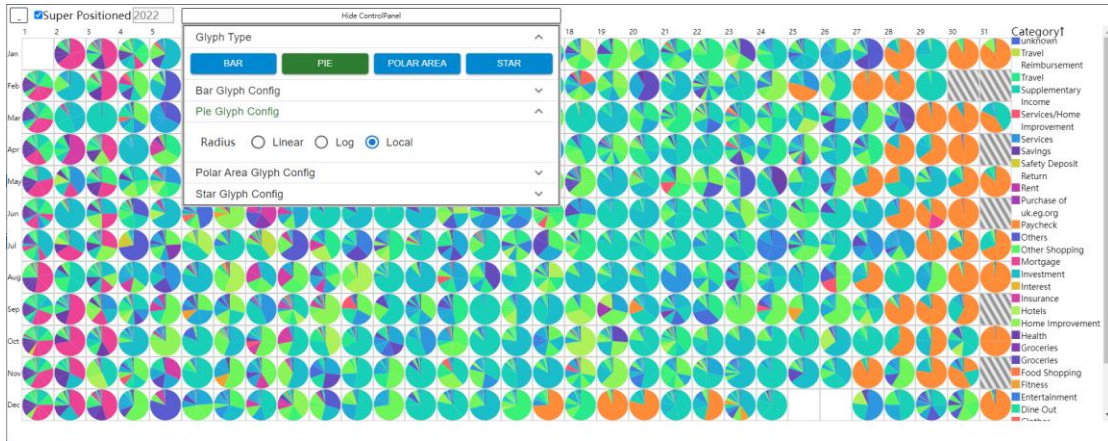


Figure 7.2.3.3. Pie glyph with a fixed radius.

7.2.4 Polar Area Glyph

The polar area glyph (Figure 7.2.5.1) is implemented. The PolarAreaView component is used to get the day's transactions in $O(1)$ from the data property and prepare the data and scale functions for the polarAreaChart components. Then, the polarAreaChart will render the glyph based on the data and scale functions.

7.2.5 Polar Area Glyph Radius Options

The polar area glyph radius options use the radiusAxis state, which provides both the information of logarithmic/linear and the information of local/shared domain, stored in the polarAreaViewSlice in the redux store. The polarAreaView will retrieve the radiusAxis state from the state store and provide different radius scale functions to the polarAreaChart component for rendering. If the scale is local, then the scale function for the radius of each slice is calculated based on the day's transactions. If the scale is global, the scale function will be the polarAreaCalendarViewSharedRadialScales provided by the CalendarView component. If the radiusAxis is a logarithmic scale, the function for creating the scale will be `d3.scaleLogarithmic`; otherwise, it will be `d3.scaleLinear`.

The calendar view control panel (Figure 7.2.5.1) provides the option buttons, and the state updating logic is the same as the logic of the pie glyph.

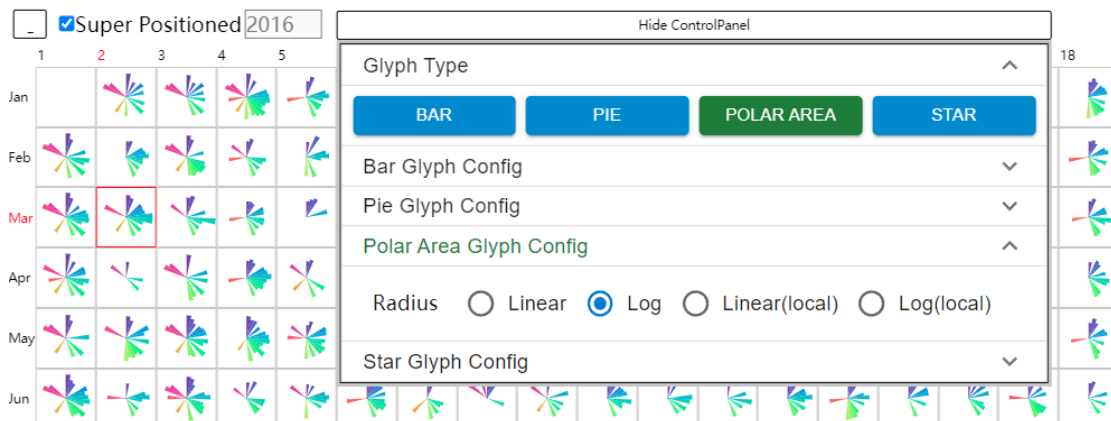


Figure 7.2.5.1. A Screenshot of the polar area glyphs with shared Logarithmic scale. The colour represents the category.

7.2.6 Star Glyphs

Star Glyphs shows the transaction amounts of each cluster in a day, see Figure 7.2.7.1. It is implemented by the `starDayView` component and `starChart` component. The `starDayView` component efficiently gets the day data from its property. Then, it prepares the data and scale functions for the `starChart` for rendering. The `starChart` will first convert the data to the points in the polar coordinate. Then, the polar coordinate will be mapped to the point in the x-y coordinate through simple geometry, see Figure 7.2.6.1. Sequentially, triangles and lines will be generated based on the points. Finally, these elements will be rendered in the glyph.

```
const x = radius * Math.sin(theta)
const y = - radius * Math.cos(theta)
const currentPoint: CartesianPoint = { x, y }
```

Figure 7.2.6.1. Code for Converting a Polar Point to an X-Y Point.

7.2.7 Star Glyph Radius Options

The star-glyph has similar radius options, just like the polar area glyph. The implementation is similar, except that the star-glyph and its control buttons use the `radiusAxis` state stored in the `starDayViewState`.

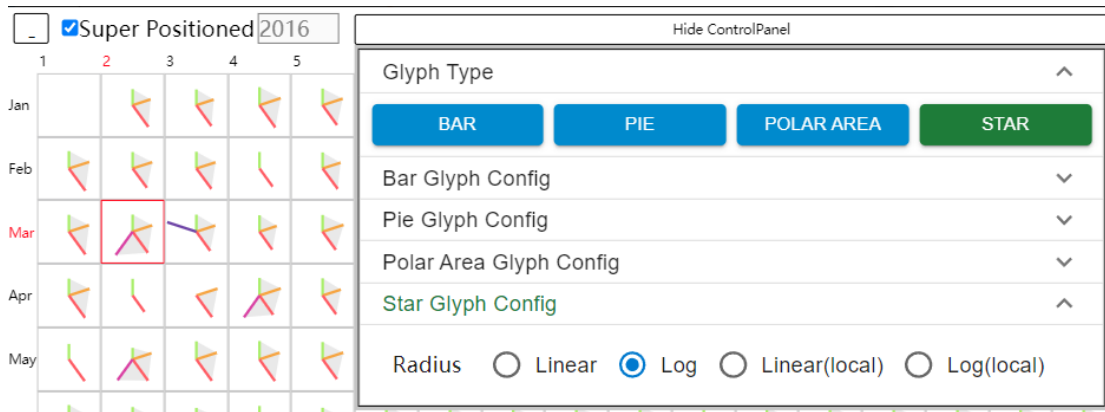


Figure 7.2.7.1. A Screenshot of the star glyphs with shared Logarithmic scale. The colour represents cluster ID.

7.2.8 “Superpositioned” Calendar View

The glyph can not only show the transaction of a day of the chosen year (e.g., 2020/1/3), but it can also show all the transactions of the day of all the years (e.g., 2016/1/3, 2017/1/3, 2018/1/3, 2019/1/3, 2020/1/3, 2021/1/3, 2022/1/3).

The feature is implemented by adding an `isSuperPositioned` state into the `calendarViewSlice` in the redux store. When the glyphs retrieve day data from the entire dataset, they will check this state to select the day data of the current year of the calendar or the day data of all the years. Then, the other logic in the glyph components does not need to be changed, as only the `dayData` selecting code is now superpositioned.

Additionally, after adding the superpositioned feature, the code at other places needs to be changed. In the App component, when calculating the `selectedGlyphTransactionNumber`, it will not check the year but only the month and day of the data if the value of the `isSuperPositioned` state is true. In the CalendarView component, when preparing shared scale functions, if the `isSuperpositioned` is true, the year will be used as part of the key when grouping the data.

The isSuperpositioned option (Figure 7.2.8.1) can be selected. When the user selects it, the year filed will be disabled. The state updating logic, like the others, uses an event handler to dispatch the corresponding actions (i.e., enableSuperPosition and disableSuperPosition) defined in the calendarViewSlice.

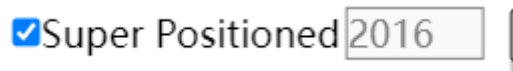


Figure 7.2.8.1. The superpositioned option. When the check box is checked, the year input field is disabled.

7.2.9 Calendar View Expanding

There is an expand/fold button for the calendar view. By clicking the expanding button (Figure 7.2.9.1A), the calendar view will be centred like a separate window (Figure 7.2.9.1). When the calendar view is expanded, the expand button will become a fold button (Figure 7.2.9.1A) for making the calendar view back to the normal size (Figure 7.2.9.2).

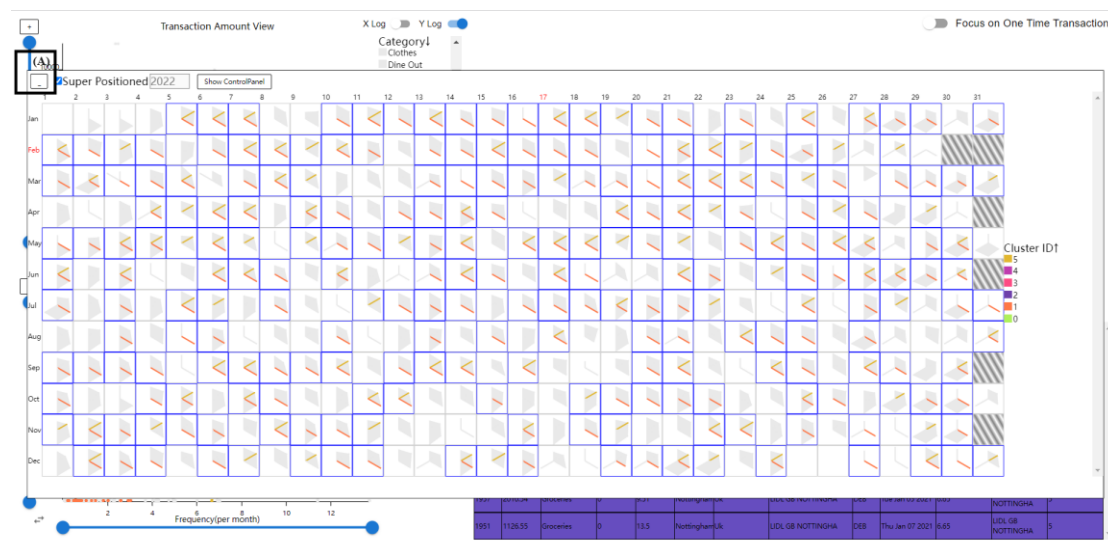


Figure 7.2.9.1. Expanded Calendar View. (A) Folding button.

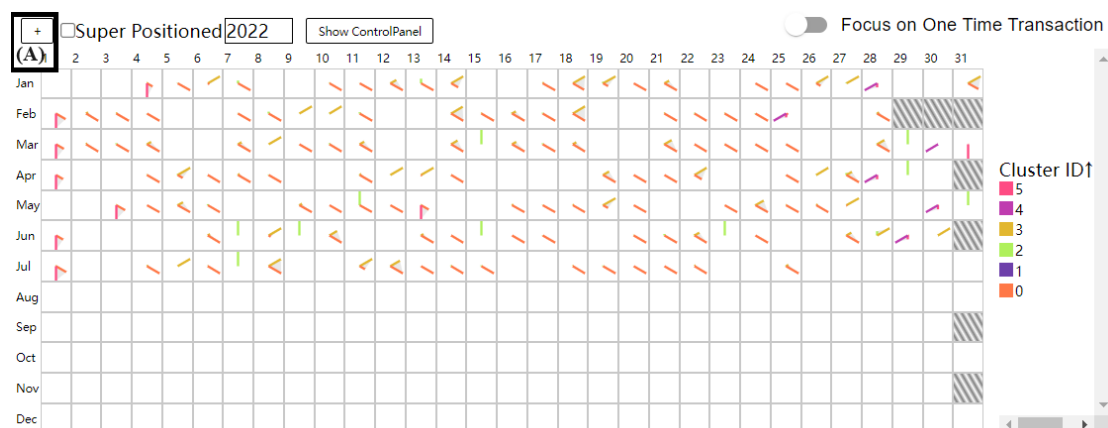


Figure 7.2.9.2. Regular Calendar View. (A) Expanding button.

To implement this feature, containerWidth, containerHeight, expandedContainerWidth, expandedContainerHeight and isExpanded state was added to the calendarViewSlice in the redux state store. The interfaces called selectCurrentContainerHeight and

selectCurrentContainerWidth were also added to the calendarViewSlice. These two selector functions can return the containerWidth and containerHeight if isExpanded is false. Otherwise, they will return the expandedContainerWidth and expandedContainerHeight. These two selector functions were used by the CalendarView component to retrieve the glyph size.

Additionally, a reusable component, ExpandableContainer, was developed. It can provide a button that shows “+” or “-” for the folded or expanded status (see Figures 7.2.9.2A and x7.2.9.1A). It uses the onSetExpand event handler, defined by the App component, to change the child element’s style when the user clicks the expand/fold button. Its storage is a state by itself to determine if the container is expanded and uses the initStyle, a CSS style, property when not expanded, and expandedStyle when expanded.

The calendar view component is wrapped as a child component inside an ExpandableContainer component. The onSetExpand event handler passes into the container is a function that dispatches the calendarViewSlice.expand or “.fold” action, which changes the isExpanded state of the calendarViewSlice.

The difficulty of implementing the expand feature for the calendar view and the other two views is to decide which component is responsible for the style property. It can be put inside the ExpandableContainer or as a property that needs to be passed into the container. The decision is made to use the latter for maintainability reasons. So that in the future, if different ExpandableContainer needs to be implemented, such as an ExpandableContainer centred at the left, at the right, with or without a border, they can be implemented by simply putting the ExpandableContainer inside a new component which provides styles.

7.2.10 Transaction Amount View Axis Swapping

The user can click the button (Figure 7.2.10.1A) to swap the x-y axis. When the axis is swapped, the brusher will move with the axis. Additionally, the sliders' range will remain the same as before swapped, and the option of using a logarithmic scale for each axis will also be memorised, see Figure 7.2.10.1 and 7.2.10.2.

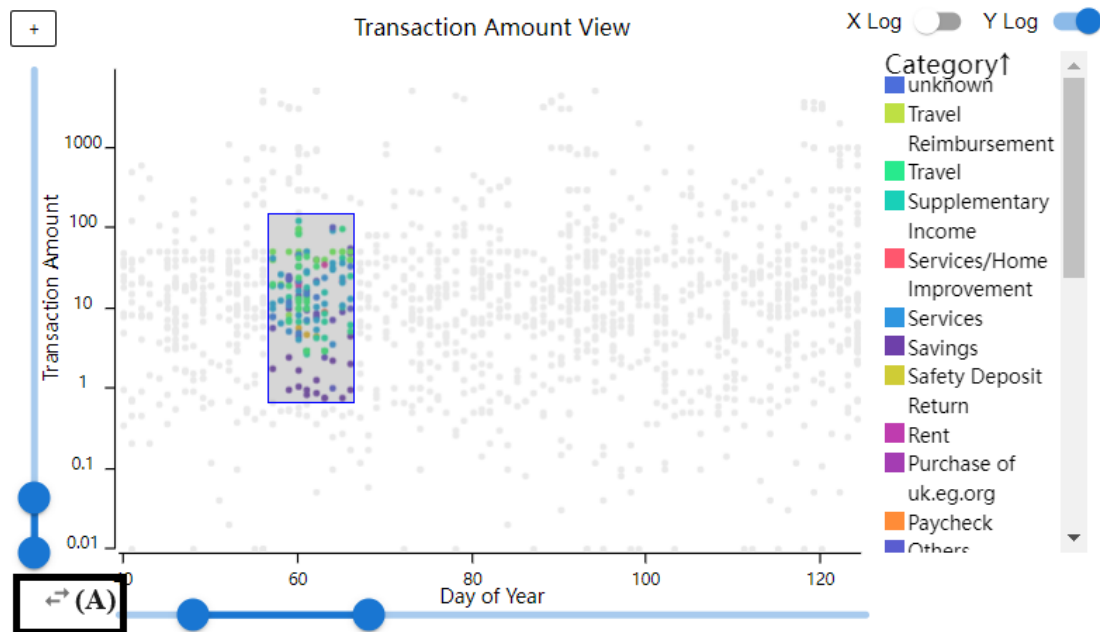


Figure 7.2.10.1. Transaction Amount View. (A) is the button for swapping the x and y axis.

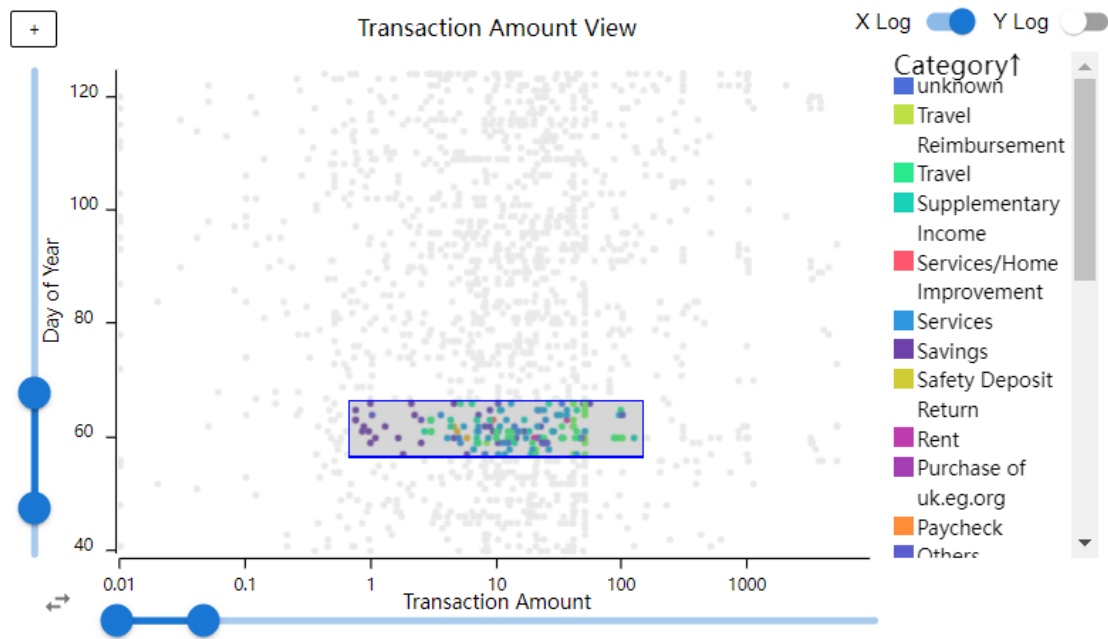


Figure 7.2.10.2. Transaction Amount View after swapping.

To explain how the swapping feature works, it needs to clarify how the InteractiveScatterPlot and TransactionAmountView render the charts.

The InteractiveScatterPlot is a reusable component that both Transaction Amount View and Cluster View can use. The axis information, the event handler for updating the states, and the data are passed by its parent components, TransactionAmountView and ClusterView. The InteractiveScatterPlot receives the properties called `xArr` and `yArr`, which are two linear arrays of numbers representing the data for the x-axis and y-axis. Then, based on the other configurations, such as those for the layout, logarithmic options, etc., scale functions for mapping x and y data will be created. Sequentially, the data in `xArr` and `yArr` will be converted to the arrays called `xVisualData` and `yVisualData`, which are the x and y locations on the chart. These two arrays will be passed into the InteractiveScatterPlot's child Circles Component to render an array of points.

To prepare the `xArr` and `yArr` for InteractiveScatterPlot, the TransactionAmountView component gets the `xArr` and `yArr` from the `scatterPlotSlice` in the redux store. In the `scatterPlotSlice`, two states (i.e., x and y) represent the attribute names of the x-axis and the y-axis. Two value selectors called `selectXdata` and `selectYdata` are used to get the x and y-axis data. Those two selectors select the data based on the states for the attribute names.

To implement the swapping feature, a "swap" action is added to the `scatterPlotSlice`. It swaps the states for the attribute names (i.e., x and y), options for logarithmic (i.e., `xLog` and `yLog` mentioned in sections 7.1.5 and 7.1.6), and the sliders (i.e., `sliderXMin`, `sliderXMax`, `sliderYMin`, and `sliderYMax` mentioned in section 7.1.5 and 7.1.6). The TransactionAmountView create the `onSwap` event handler, which dispatches the swap action and provides this event handle as the `handleSwap` property of TransactionAmountView's child component InteractiveScatterPlot. Finally, the swap button is added to the InteractiveScatterPlot.

The major difficulty is that this feature causes a bug for the brusher, which is the brusher's location does not move when the axis is swapped. The description of this challenge and the solution was introduced in section 7.1.8.

7.2.11 Transaction Amount View Expanding

The transaction amount view also has an expanding feature. The user can expand and fold the transaction amount view (see Figures 7.2.11.1 and 7.2.11.2).

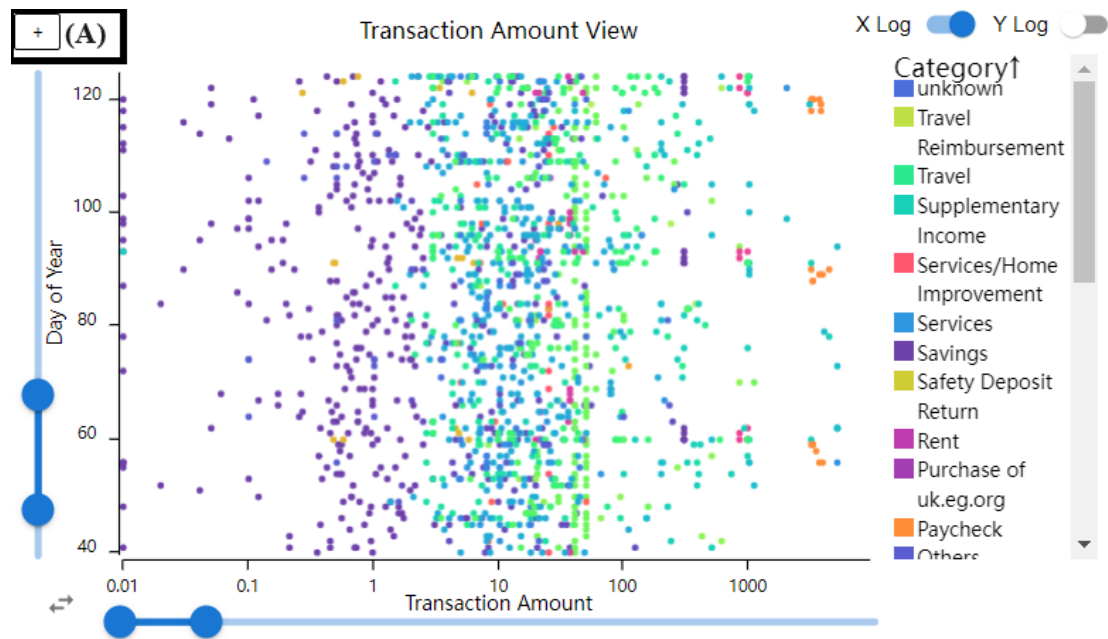


Figure 7.2.11.1. Folded Transaction Amount View. (A) Expanding button.



Figure 7.2.11.2. Expanded Transaction Amount View. (A) Fold button.

The implementation of this expanding feature for the transaction amount view is almost the same as the expanding feature for the calendar view, except for two differences. The first one is that the states, including `containerWidth`, `containerHeight`, `expandedContainerWidth`, `expandedContainerHeight` and `isExpanded`, and the interfaces (or called selectors), including `selectCurrentContainerHeight` and `selectCurrentContainerWidth`, are included in the `ScatterPlotSlice` rather than the `CalendarViewSlice`. The second difference is that the event handler for expanding dispatches the `expand` and `fold` action for the `scatterPlotSlice` rather than the `calendarViewSlice`, see Figure 7.2.11.3.

```

// expanding handler
/**
 * tell the components which are wrapped inside the expandablecontainer it is expanded or folded
 * @param chartToExpand chart to expand
 */
function handleSetExpand(nextIsExpand: boolean, chartToExpand: 'calendar view' | 'cluster view' | 'scatter plot') {
  if (chartToExpand === 'calendar view') {
    if (nextIsExpand) {
      dispatch(calendarViewSlice.expand())
    } else {
      dispatch(calendarViewSlice.fold())
    }
  } else if (chartToExpand === 'scatter plot') {
    if (nextIsExpand) {
      dispatch(scatterPlotSlice.expand())
    } else {
      dispatch(scatterPlotSlice.fold())
    }
  } else if (chartToExpand === 'cluster view') {
    if (nextIsExpand) {
      dispatch(clusterViewSlice.expand())
    } else {
      dispatch(clusterViewSlice.fold())
    }
  }
}
}

```

Figure 7.2.11.3. The event handler will be called when the user clicks the expand and fold button. It is in the App component and will be passed into the expand/fold buttons in the ExpandableContainers with the chartToExpand argument based on the child component of the ExpandableContainers.

The difficulty is also the bug for the brusher, which is the brusher’s location does not move when the size of the chart changes. The solution is just like the solution mentioned in 7.1.8.

7.2.12 Cluster View Expanding

The expanding feature of the cluster view is almost the same as the one of the transaction amount views, but the states and the event handler are related to the clusterViewSlice.

7.2.13 Cluster View Colour Option

The user can choose what variable to be represented by the colours in the cluster view control panel. The supported variables are category, cluster ID, and transaction description group.

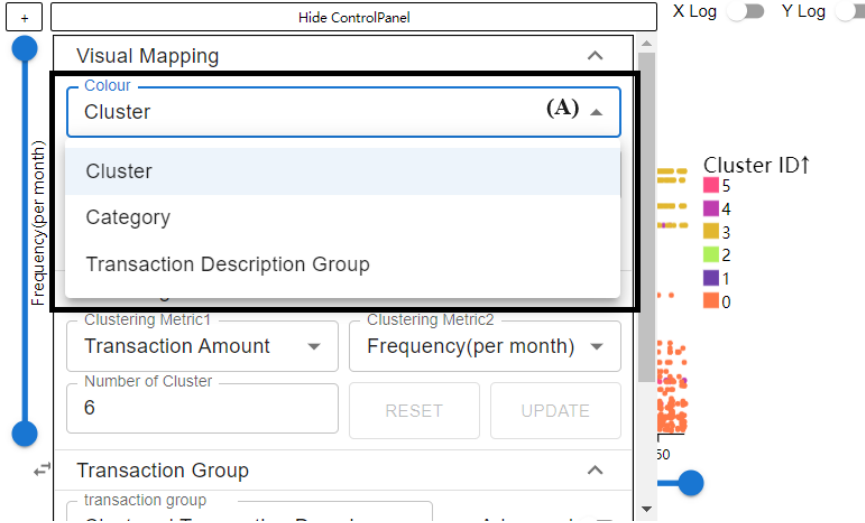


Figure 7.2.13.1. Cluster View Control Panel. (A) is the colour option. The user can choose what is represented by the colours of the points in the cluster view.

To implement this feature, a state called colour is added to the clusterViewSlice of the state. A selector called selectColourLabel is used for providing the knowledge of what is represented by the colours in the cluster view, and another selector called selectColourDomainMemorised is created for providing a linear array of ColourDomainData object (which has the information of transaction number and its value for the corresponding colour variable defined by the colour state). An action called setColour is used for updating the colour label.

Then, the ClusterView choose one of the colour scales (i.e., categoryColourScale, clusterIdColourScale, frequencyUniqueKeyColourScale) from the props provided by the App component based on the knowledge retrieved by selectColourLabel selector of what variable represented by colours. And use selectColourDomainMemorised to get the variable for each data. The colour scale function, colour domain array and colour label will be passed into its InteractiveScatterPlot component and rendered by the InteractiveScatterPlot component.

The interactivity of the colour option (Figure 7.2.13.1A) is implemented in the ClusterViewMappingControlPanel component by providing an event handler called handleChangeColour for the selection box, which uses Material UI. The event handler will dispatch the setColour action to update the colour state.

7.2.14 Cluster View Axis Options and Swapping

Like the transaction amount view, the cluster view also provides a swap button. Additionally, the user can choose what variable to be represented by the x and y axis in the cluster view control panel, see Figure 7.2.14.1.

As the cluster view also uses the InteractiveScatterPlot component, the swapping implementation is like the transaction amount view implementation. Still, the states, selectors and actions are in clusterViewSlice.

Additionally, two actions called setXLabel and setYLabel are created in the clusterViewSlice. They are used for changing the x and y states and separately reset the states for the sliders. The axis options (see figure 7.2.14.1A&B) are implemented in the ClusterViewMappingControlPanel component, and they are achieved by the event handlers (i.e., handleChangeXLabel and handleChangeYLabel) that dispatch the setXLabel and setYLabel actions, respectively.

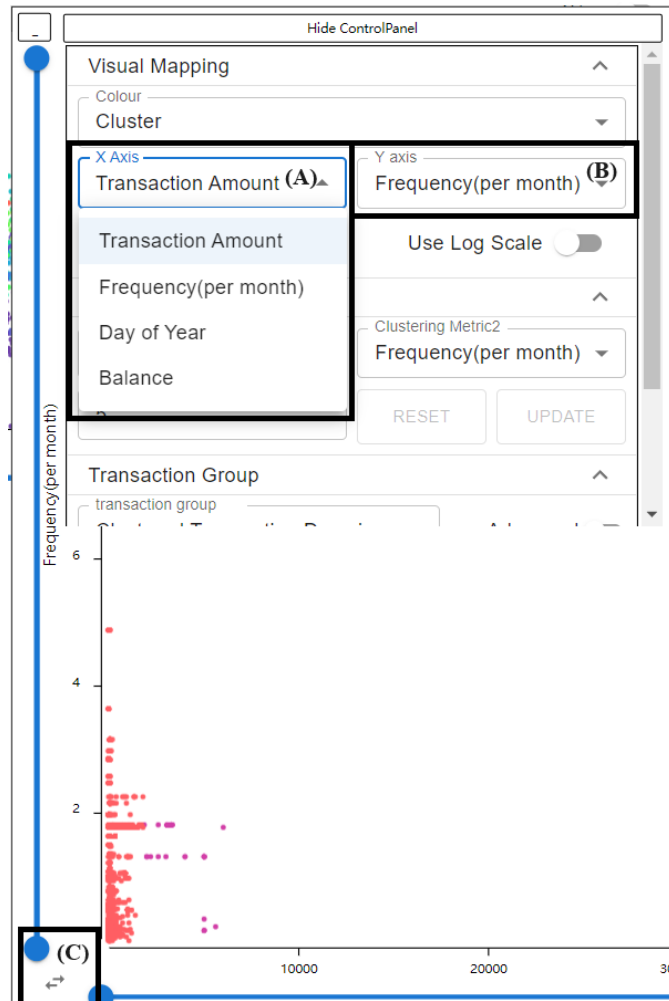


Figure 7.2.14.1. Cluster View Control panel. (A) X-axis Option. (B) Y-axis Option. (C) Axis swap Button.

7.2.15 Table View Page Option

The user can determine the number of rows displayed in the table view. There is a drop-down menu for choosing the number of rows (Figure 7.2.15A). Additionally, the information on which rows is shown on the table is also displayed next to the drop-down menu (Figure 7.2.15B). The user can switch between rows by changing the current page (Figure 7.2.15C).

The candidate numbers of rows are 8, 17, 50, 100, 200, and all. The first two numbers are carefully chosen as eight rows can make the web page not overflow, and 17 can make the table's height become the height of the web page. The other larger number allows the user to scroll on the table when they want. Additionally, when the table overflows the screen, the screen will automatically scroll so that the entire table will be displayed. As these options are only used by the table view, the configuration state, reducers, and actions are just maintained by the TableView component. There is a numberOfRowsPerPage attribute defined in the table view's configuration state. It is used for deciding how many rows to display. The action whose type is 'change number of rows' is defined, and the reducer will be updated with the value of numberOfRowsPerPage when the action is dispatched. An event handler called handleChangeNumberOfRowsPerPage is defined in the component and is provided to the drop-down menu. This event handler will dispatch the changeNumberOfRowsPerPage action when the user chooses a value in the drop-down menu. The component will also be checking the numberOfRowsPerPage. If this value changes, it will automatically scroll the window using the

DOM API called scrollIntoView.

To provide a current page option, currentPage is defined in the configuration. An action called ‘change page’ is defined, and the reducer updates the value of the currentPage when the action gets dispatched. An event handler called handleChangePage is created to dispatch the “change page” option after checking if the value exceeds the number of pages. The event handler is passed into the page number input box (Figure 7.2.15.1C).

To decide which rows to render, the startIndex and endIndex are calculated based on the value of numberOfRowPerPage and currentPage.

The numbers in (Figure 7.2.15.1B) are calculated based on startIndex, endIndex and numberOfRowPerPage values.

The difficulty is to decide the best number of rows. Initially, each row's size may change based on its content, so given the number of rows, the height of the table might change, which means the table may or may not overflow. To solve this problem, each row's height and the column's width are set to be constant. Additionally, if the data in a cell exceeds the cell size, the exceeded part will be hidden. Therefore, tooltips are provided so the user can hover over the cell and see the full data.

Brushed Data Selected Glyph Data

(A) show 8 rows 1-8 of 25 Current Page: 1 (C) Clear All (D)

ID	Balance	Category	Credit	Debit	City	Country	Description	Type	Date	Frequency	Description Group	Cluster Id
2314	3226.74	Savings	0	0.86	Nottingham	Uk	SAVE THE CHANGE	BP	Fri Sep 04 2020	13.80	128	3
3147	5736.68	Groceries	0	13.55			SAINSBURYS S/MKTS	DEB	Wed Sep 04 2019	0.53	88	0
3148	5750.23	Entertainment	0	5.95	Worcester	Uk	WWW.FREEDOM-LEISUR	DEB	Wed Sep 04 2019	0.28	240	0
3149	5756.18	Amazon	0	2.75	Swansea	Uk	AMZNMktplace	DEB	Wed Sep 04 2019	8.13	63	1
3150	5758.93	Cash	0	50	Swansea	Uk	LNK TESCO FABIAN W	CPT	Wed Sep 04 2019	1.39	192	0
3151	5808.93	Savings	0	0.19	Swansea	Uk	SAVE THE CHANGE	BP	Wed Sep 04 2019	13.80	128	3
4091	13816.47	Amazon	3	0	Swansea	Uk	CLUB LLOYDS WAIVED	DEP	Tue Sep 04 2018	2.00	11	0
4092	13813.47	Bills	0	3	Swansea	Uk	CLUB LLOYDS FEE	PAY	Tue Sep 04 2018	2.00	11	0

Figure 7.2.15.1. Table View. (A) option for the number of rows to display on a page at once. (B) information of the current rows. (C) option for changing the current page. (D) Clear all buttons that will cancel the selection for the current table. In this case, if the user clicks the clear all button, the day selection on the calendar view will be cancelled because the current table is “Selected Glyph Data”.

7.2.16 Table View Clear-All Button

The “clear all” button in the table view is one of the ways to cancel the selection. When the user clicks it, the day selection in the calendar view will be cancelled if the current table is for the glyph data. If the current table is for the brushed data or the data for the selected colour legend, the selection will be cancelled.

To implement this feature, two event handlers called handleClearBrush and handleClearGlyph are created in the App component and passed into the brushed and glyph data tables, respectively. Depending on the current table, one of these event handlers will be called when the user clicks the “clear all” button. The handleClearBrush will dispatch the clearBrush action, which is created in interactivitySlice. It will set the currentSelector to be an empty string, which will cause the selectSelectedTransactionNumberArr to return an empty array. The handleClearBrush will dispatch the action called clearDetailDay, which is defined in the

calendarViewSlice. It will set the detailDay to be null. As a result, if the set of transaction numbers is empty or the detailDay is null, the brushed table or the glyph detail table will be empty.

7.2.17 Table View Sorting

If the user clicks the table view's column name in the table view, the data's order will be changed. Initially, the table is sorted by the ID (i.e., transactionNumber) in ascending order. If the user clicks the column name that is the current column name for sorting, then the order will be toggled. If the user clicks the other column name, the table will be sorted by the new column name, and the order will be the same as the previous sorting order. The column name (Figure 7.2.17.1A) will be with an arrow after it if it is the sorting key.

Brushed Data Selected Glyph Data show 8 rows 1-8 of 2949 Current Page: 1 Clear All

ID	Balance	Category	Credit	Debit	City	Country	Description ↑ (A)	Type	Date	Frequency	Description Group	Cluster Id
4070	13160.36	Hotels	0	42.99	Birmingham	Uk	easyHotel Birmingh	DEB	Wed Sep 19 2018	0.15	106	4
4048	14424.88	Hotels	0	10	Birmingham	Uk	easyHotel Birmingh	DEB	Mon Oct 01 2018	0.15	106	4
4750	13241.67	Insurance	0	19.36			ZURICH	DD	Wed Oct 18 2017	1.46	19	4
4394	20161.86	Insurance	0	19.36	Swansea	Uk	ZURICH	DD	Thu Apr 05 2018	1.46	19	4
4326	21365.01	Insurance	0	19.36	Swansea	Uk	ZURICH	DD	Tue May 01 2018	1.46	19	4
4281	17382.54	Insurance	0	19.36	Swansea	Uk	ZURICH	DD	Fri Jun 01 2018	1.46	19	4
4217	16209.47	Insurance	0	19.36	Swansea	Uk	ZURICH	DD	Tue Jul 03 2018	1.46	19	4
4164	18626.74	Insurance	0	19.36	Swansea	Uk	ZURICH	DD	Wed Aug 01 2018	1.46	19	4

Figure 7.2.17.1. Table View. The transactions are sorted by their description in ascending order. (A) is the column name, it has an up arrow indicating that transactions are sorted by their transaction description in ascending order. The column name can be clicked like a button.

To implement this feature, a sortingKey and isDesc are stored in the sortingConfig state in the TableView component. Actions “change sorting key” and “toggle order” are defined. The reducer will update sortingKey and isDesc. Like other features, event handlers dispatch actions when the user clicks the buttons. However, the state is stored locally in the component rather in the redux store.

When rendering the table, the sortingKey and isDesc are used to create a comparator function, which will be used to sort the array of the selected transaction data. The TableView component will render the sorted transaction data array. The column names will also be rendered based on the value of sortingKey and isDesc.

7.2.18 Table View Synchronised Row Colour

The colour for each row is determined by how the user selects the transaction. Suppose the user uses the brusher on the transaction amount view. In that case, the colour on the table will represent the category of the transaction, just like in Figure 7.2.18.1. If the user uses the brusher on the cluster view, the colour will be determined by the colour option for the cluster view chosen by the user, just like Figure 7.2.18.2. If the user selects the data by colour legend, the colour of the rows in the table will be determined by the colour legend, just like in Figure 7.2.18.3.

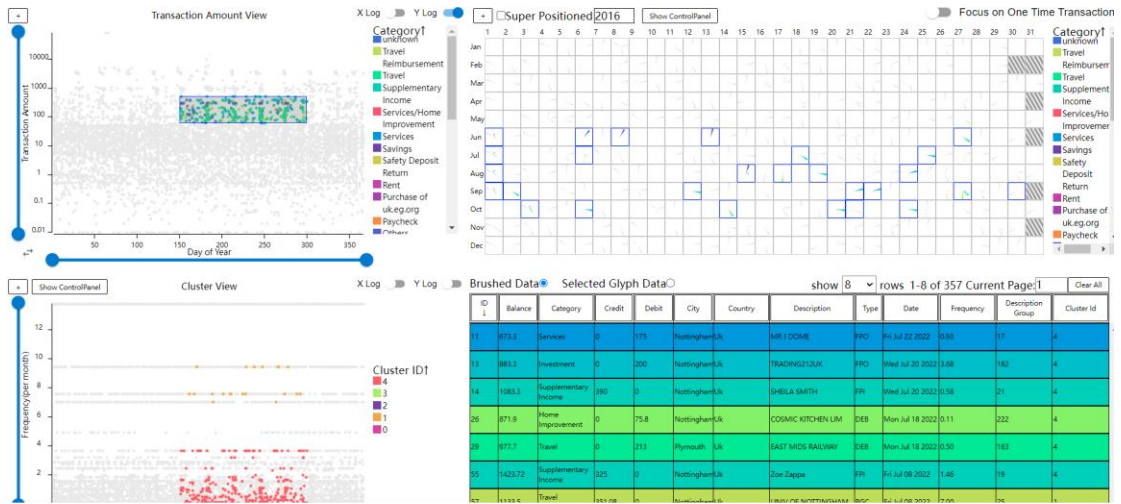


Figure 7.2.18.1. A Screenshot of iBankEx. The user selected the data through the brusher in the transaction amount view. The category of each transaction determines the colour on the brushed data table.

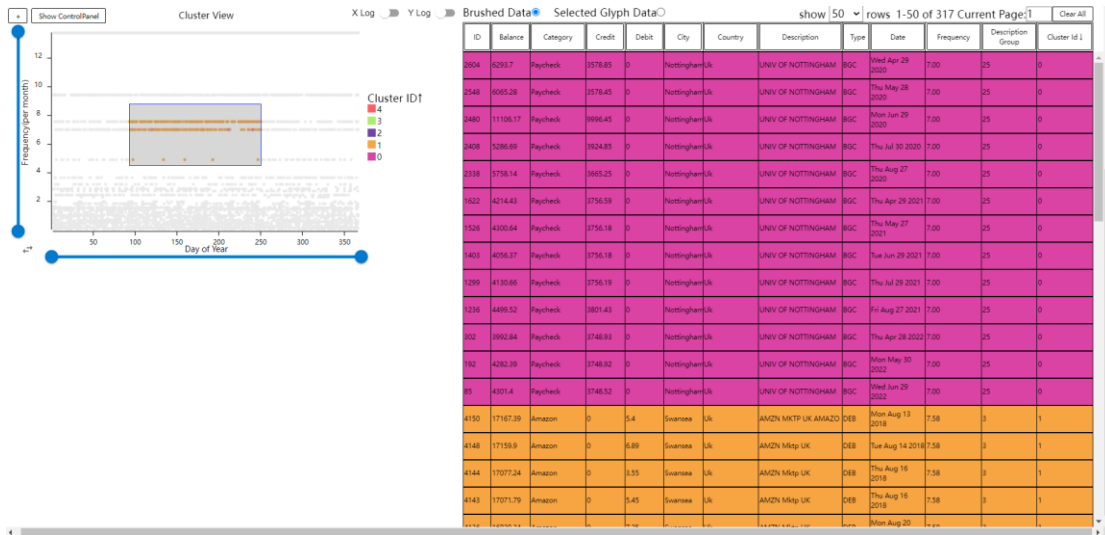


Figure 7.2.18.2. A Screenshot of the cluster view and table view. The user selected the data through the brusher in the cluster view. The cluster ID of each transaction determines the colour on the brushed data table.

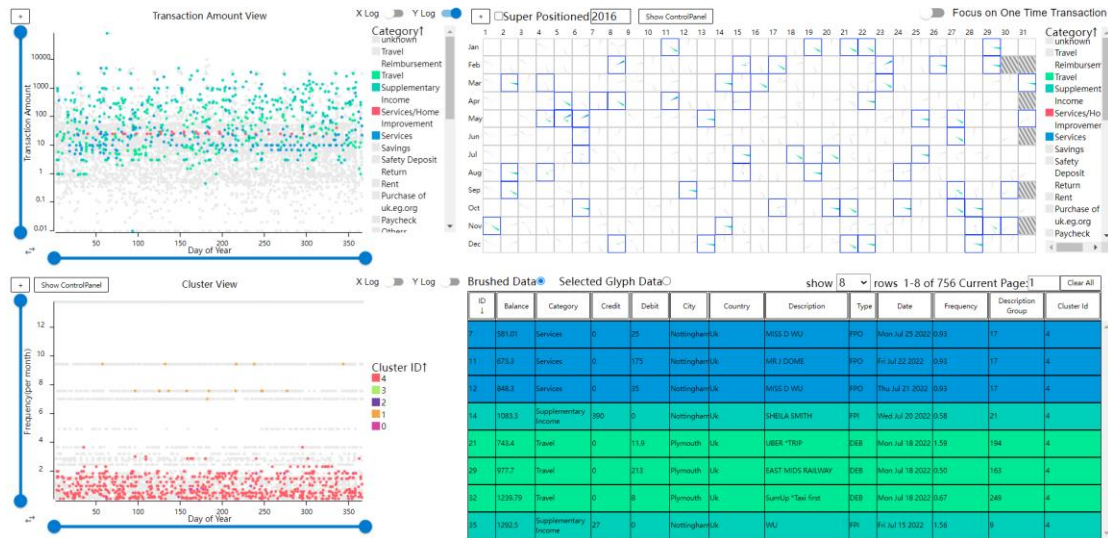


Figure 7.2.18.3. A Screenshot of iBankEx. The user selected data through the category legend. The category of each transaction determines the colour in the table.

To implement this feature, `selectCurrentSelectorColourScaleType` is defined in the `interactivitySlice`. This selector tells the colour scale type by checking the `interactivitySlice`'s `currentSelector` state and the colour label of the cluster view. The App component will choose the right colour scale function for the table views, see Figure 7.2.18.4. Then, the table view sets the background of each row based on the colour string returned by the colour scale function.

```

41 export default function App() {
42   const brushedTransactionNumberArr = useAppSelector(interactivitySlice.selectSelectedTransactionNumberArrMemorised)
43   const brushedTransactionNumberSet = useMemo(() => new Set(brushedTransactionNumberArr), [brushedTransactionNumberArr])
44
45   useSyncTransactionDataAndClusterData(); // app is responsible for checking the relative states in the redux store and update
46   const categoryColourScale = useCategoryColourScale()
47   const clusterIdColourScale = useClusterIdColourScale()
48   const frequencyUniqueKeyColourScale = useFrequencyUniqueKeyColourScale()
49   const transactionDataArr = useTransactionDataArr();
50
51   /**table's colour scale type */
52   const colourLabelForTable = useAppSelector(interactivitySlice.selectCurrentSelectorColourScaleType)
53   const tableColourScale = colourLabelForTable === 'category' ? categoryColourScale :
54     (colourLabelForTable === 'cluster' ? clusterIdColourScale : frequencyUniqueKeyColourScale)
55   const tableViewColourDomainDataArr = useAppSelector(colourChannelSlice.selectTableViewColourDomainDataMemorised)
56
57   /**glyph table's colour scale type, synch with glyph's colour */
58   const colourLabelForGlyphTable = useAppSelector(interactivitySlice.selectGlyphDataTableColourScaleType)
59   const glyphTableColourScale = colourLabelForGlyphTable === 'category' ? categoryColourScale :
60     (colourLabelForGlyphTable === 'cluster' ? clusterIdColourScale : frequencyUniqueKeyColourScale)
61   const glyphTableViewColourDomainDataArr = useAppSelector(colourChannelSlice.selectGlyphTableViewColourDomainDataMemorised)

```

Figure 7.2.18.4. Code in App component. Lines 42 and 43 prepare the colour scale for the selected data detail table. Lines 49 and 50 prepare the colour scale function for the glyph detail table.

7.2.19 Colour Legend Sorting

To help the user find the category, cluster ID or transaction description group more easily, the colour legend can be sorted by clicking their title. By default, the legend is sorted by colour. If the user clicks the title, the legends will be sorted by the letters in ascending order, see Figure 7.2.19.1. If the user clicks the title again, the order will be descending. If the user clicks the title when the order is descending, the order will be changed to the default order.



Figure 7.2.19.1. A Screenshot of Category colour legends with different sorting order. (A) is sorted by colour. (B) is sorted by the words in ascending order. (C) is sorted by the words in descending order.

This feature is implemented in the LegendList component, see Figure 7.2.19.2. A sortBy state is maintained inside the LegendList component. An array of data for each legend is called sortedColourMappingArr, which is sorted based on the sortBy value. When rendering the colour legend, the order is the same as the data in the sortedColourMappingArr. Additionally, to provide interactivity, the handleToggleSorting is created to change the sortBy state when the user clicks the title.

```

95     const isDomainNumberLike = colourMappingArr.every(colourMapping => isNumber(colourMapping.domain))
96     const comparatorAscending = isDomainNumberLike ?
97       ((a: ColourMapping, b: ColourMapping) => parseFloat(a.domain) - parseFloat(b.domain)) :
98       ((a: ColourMapping, b: ColourMapping) => a.domain > b.domain ? 1 : -1)
99     const comparatorDescending = (a: ColourMapping, b: ColourMapping) => -comparatorAscending(a, b)
100    const [sortBy, setSortBy] = useState<'domainDescending' | 'domainAscending' | 'colour'>('domainDescending')
101    let sortedColourMappingArr: ColourMapping[] = [...colourMappingArr]
102    switch (sortBy) {
103      case 'domainAscending':
104        sortedColourMappingArr.sort(comparatorAscending)
105        break;
106      case 'domainDescending':
107        sortedColourMappingArr.sort(comparatorDescending)
108        break;
109      case 'colour':
110        break;
111      default:
112        const _exhaustiveCheck: never = sortBy
113        throw new Error("exhaustive check error");
114    }
115    function handleToggleSortting() {
116      if (sortBy === 'colour') {
117        setSortBy('domainAscending')
118      } else if (sortBy === 'domainAscending') {
119        setSortBy('domainDescending')
120      } else if (sortBy === 'domainDescending') {
121        setSortBy('colour')
122      } else {
123        const _exhaustiveCheck: never = sortBy
124        throw new Error("exhaustive check error");
125      }
126    }
127    const Arrow = (sortBy === 'colour' ? <span></span> : sortBy === 'domainAscending' ? DOWNARROW : UPARROW)

```

Figure 7.2.19.2. Code for sorting the colour legend in the LegendList component. The sortedColourMappingArr stores the information for rendering the colour legend. It is sorted based on the sortBy state. The handleToggleSortting function is used for handling the user’s interaction.

The challenge is to sort the colour legends that only use numbers. If comparing the number by “>”, the order will be like Figure 7.2.19.3. That’s because the data type of the numbers is string. To handle this problem, they were converted to float and compared in line 98 in Figure 7.2.19.2.

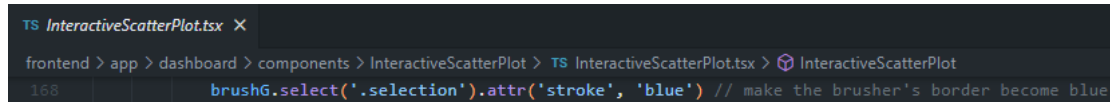


Figure 7.2.19.3. Unintended error, showing the wrong order of sorted cluster-ID.

7.2.20 Border Colour for Brusher and Calendar View

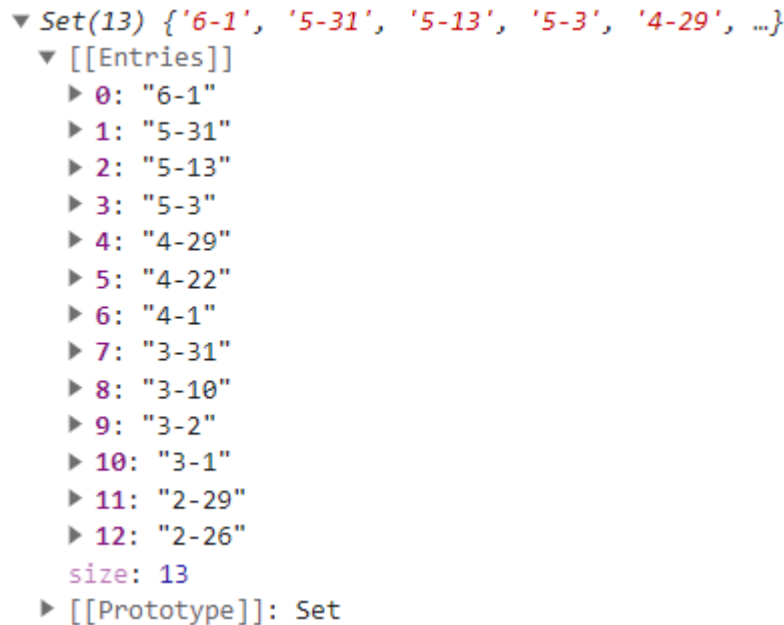
As **Figure 7.1.8.1** illustrates, the brusher's border colour and the rectangles wrapping the glyphs are blue.

The brusher's colour is set to be blue in the `InteractiveScatterPlot` component, see **Figure 7.2.20.1**. To set the border colour in the calendar view to be blue, the `CalendarView` component created a set of date string (see **Figure 7.2.20.2**) when the selected transactions happened. This Set is called `highLightedCalendarDayBorderMMDDSet` and is passed into the `MonthView` component introduced in 7.1.2. When the `MonthView` renders each day's glyph, the border will be blue if the date is in the `highLightedCalendarDayBorderMMDDSet`.



```
TS InteractiveScatterPlot.tsx X
frontend > app > dashboard > components > InteractiveScatterPlot > TS InteractiveScatterPlot.tsx > InteractiveScatterPlot
168 brushG.select('.selection').attr('stroke', 'blue') // make the brusher's border become blue
```

Figure 7.2.20.1. The code that sets the border colour of the brusher to be blue.



```
▼ Set(13) {'6-1', '5-31', '5-13', '5-3', '4-29', ...}
  ▼ [[Entries]]
    ▶ 0: "6-1"
    ▶ 1: "5-31"
    ▶ 2: "5-13"
    ▶ 3: "5-3"
    ▶ 4: "4-29"
    ▶ 5: "4-22"
    ▶ 6: "4-1"
    ▶ 7: "3-31"
    ▶ 8: "3-10"
    ▶ 9: "3-2"
    ▶ 10: "3-1"
    ▶ 11: "2-29"
    ▶ 12: "2-26"
    size: 13
  ▶ [[Prototype]]: Set
```

Figure 7.2.20.2. A set of string representing the month and day when the selected transactions happened.

There are two reasons for using string rather than the JavaScript date class. One reason is to support the superpositioned calendar view feature, which does not need to know the year information needed in the date class. Another reason is to get the correct Boolean value when checking if a date is in the set. The return value will be false if the date object to check is not in the set, even if there is a date object with same time in the set (see **Figure 7.2.20.3**), which makes date checking inconvenient. The primitive nature of string ensures that the date string created anywhere can be regarded as the same by the set.

```

> const d1 = new Date(2020,1,1); const d2 = new Date(2020,1,1);
< undefined
> console.log(d1); console.log(d2);
Sat Feb 01 2020 00:00:00 GMT+0000 (Greenwich Mean Time)
Sat Feb 01 2020 00:00:00 GMT+0000 (Greenwich Mean Time)
< undefined
> s = new Set()
< ▶ Set(0) {size: 0}
> s.add(d1)
< ▶ Set(1) {Sat Feb 01 2020 00:00:00 GMT+0000 (Greenwich Mean Time)}
> s.has(d2)
< false
> s.has(d1)
< true

```

Figure 7.2.20.3. When using the date object in the set, two dates with the same time will not be regarded as the same thing when checked by the “has” method.

7.2.21 Table View Consistent Radio Button Colour

The colour for the brusher (Figure 7.2.21.1A), the border of the glyphs with selected transactions (Figure 7.2.21.1B), and the radio button (Figure 7.2.21.1C) are blue.

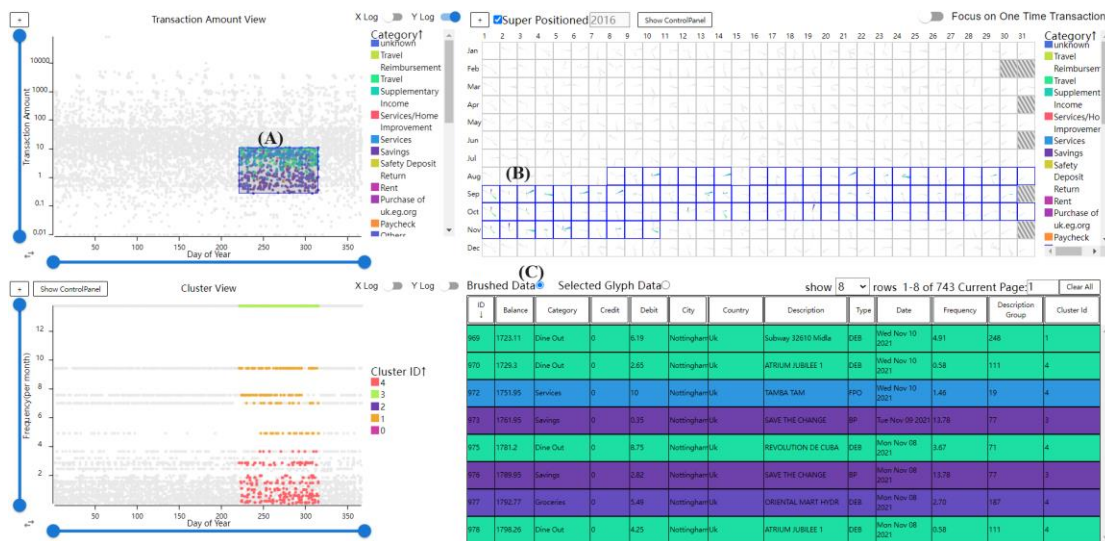
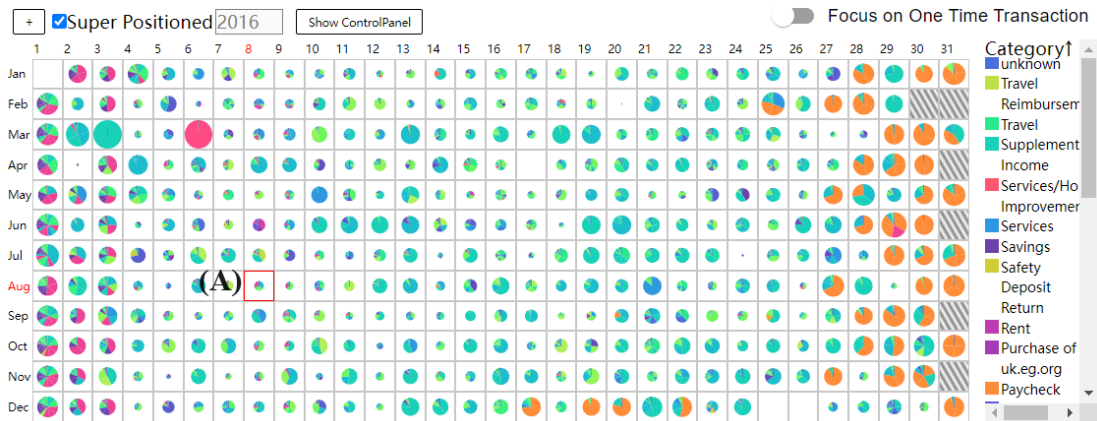


Figure 7.2.21.1. A Screenshot of iBankEx (A) Brusher (B) Glyph with blue border (C) Blue radio button.

Similarly, the colour for the selected glyph’s border (Figure 7.2.21.2A) and the radio button (Figure 7.2.21.2B) are red. This is implemented by the MonthView component. It checks the detailDay state provided by the CalendarView and sets the colour to red if the date of the glyph is the detailDay.



(B)

Brushed Data Selected Glyph Data show 8 rows 1-8 of 8 Current Page: 1 Clear All

ID ↓	Balance	Category	Credit	Debit	City	Country	Description	Type	Date	Frequency	Description Group	Cluster Id
3267	6690.72	Investment	0	100	Swansea	Uk	WWW.III.CO.UK.DE	DEB	Thu Aug 08 2019	0.69	177	4
4151	17172.79	Insurance	0	37.12	Swansea	Uk	LV LIFE	DD	Wed Aug 08 2018	1.23	22	4
4892	14754.68	Amazon	0	8.39	Swansea	Uk	Amazon UK Marketpl	DEB	Tue Aug 08 2017	9.42	8	1
4893	14763.07	Other Shopping	0	38.8			THE HEALTH SHOP	DEB	Tue Aug 08 2017	0.18	51	4
4894	14801.87	Cash	0	50	Swansea	Uk	LNK UNI - UNIVERSI	CPT	Tue Aug 08 2017	2.46	42	4
4895	14851.87	Savings	0	0.81	Swansea	Uk	SAVE THE CHANGE	BP	Tue Aug 08 2017	13.78	77	3
5650	14645.03	Amazon	0	2.81	Swansea	Uk	Amazon UK Marketpl	DEB	Mon Aug 08 2016	9.42	8	1
5651	14647.84	Insurance	0	37.12	Swansea	Uk	LV LIFE	DD	Mon Aug 08 2016	1.23	22	4

Figure 7.2.21.2. A Screenshot of the right part of iBankEx (A) Selected Glyph with Red Border (B) Red Radio button.

7.2.22 Transaction Description Grouping Advanced Mode

An advanced mode is provided for the expert user. The user can choose what distance measure, linkage method and target number of transaction description group in the cluster view control panel, see Figure 7.2.22.1. This feature allows the user to explore different parameters for the transaction description grouping, as there might be better parameters than the default one. The user must turn on the advanced mode button to enable these options.

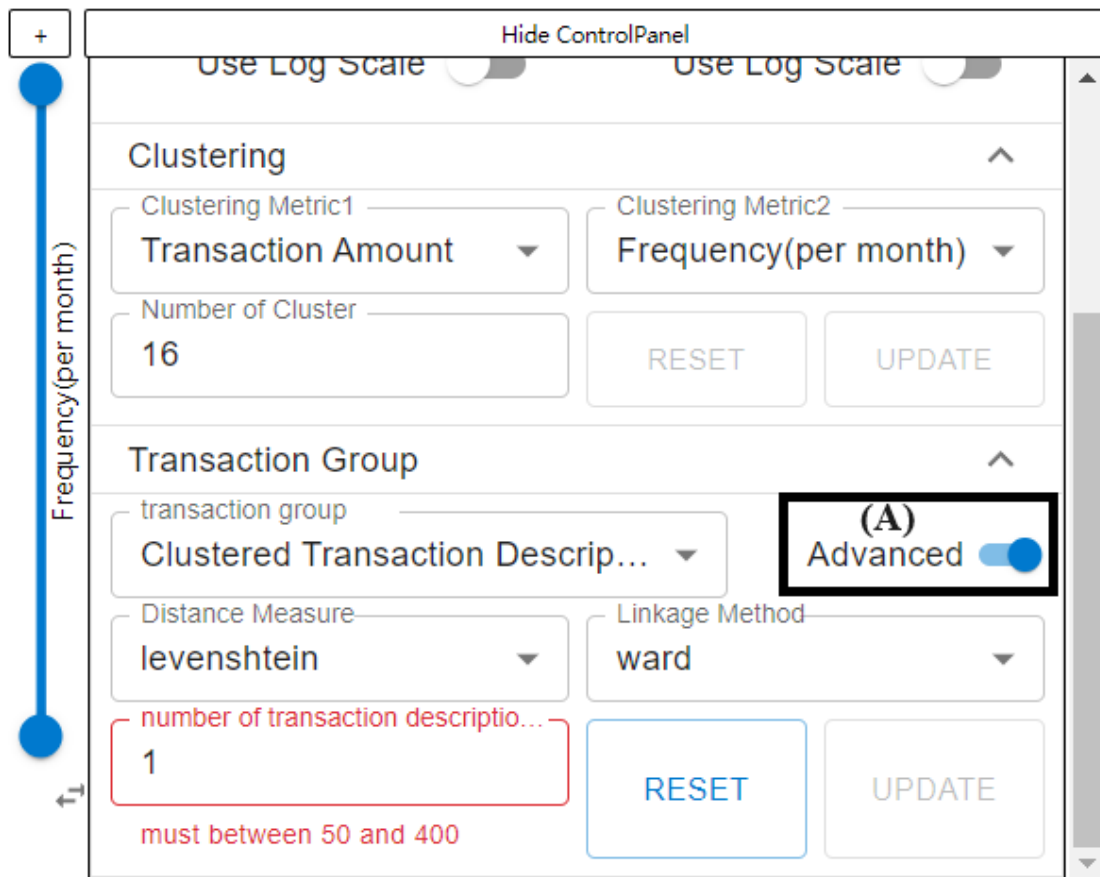


Figure 7.2.22.1. A Screenshot of the cluster view control panel. (A) is the button for the advanced mode. When the button is switched off, the distance measure, linkage method and number of transaction description groups will not be editable.

The distance measures are just those jellyfish provide (Turk, n.d.). Scipy provides the method for calculating the distance matrix and linkage matrix. When the user clicks update, the frequencyUniqueKey state defined by clusterViewSlice in the redux store will be updated by the handleSave event handler that distich the setFrequency behaviour in the clusterViewSlice in the redux state store. Then, once this state gets updated, the App component will call the updateFrequencyInfo API through the function useSyncTransactionDataAndClusterData. Once the API is returned, the entire dataset stored in the interactivitySlice in the redux store will be updated. As a result, all the views using transaction data will be re-rendered.

When the API server receives the API call, the transactionDataset object will use the corresponding LinkageBasedClusterer object, which has a precalculated distance matrix to generate the new transaction description group id for the transactions. In detail, the LinkageBasedClusterer will update its linkage matrix based on the linkage method by __updateLinkageMatrix method. Then, _searchOptimalThreshold will use binary search to find the threshold that can produce the number of clusters closest to the user-defined target number of clusters. The getClusterIdList method will use the threshold to generate the cluster id list through the fcluster function provided by Scipy.

8 Testing and Evaluation

This section describes the finding of the dataset and shows the usefulness by case study and performance test.

8.1 Results

This section presents a case study and the findings of the data Idiosyncrasies that not has been mentioned by Firat et al. (2023).

8.1.1 Case Study: Data Exploration 1 – paycheck

This case study shows the usefulness of iBankEx for understand the data, where the paycheck transaction is explored from the overview to the detail. It starts from the obvious orange bars at the end of each month (Figure 8.1.1.1). This case study shows the usefulness of the superpositioned calendar view, bar glyph, star glyph, the clustering algorithm, the table view, the cluster view, the transaction amount view, colour legend sorting, context and focus, table reordering, transaction frequency, description grouping option, transaction clustering, view expansion, and brush and linking.

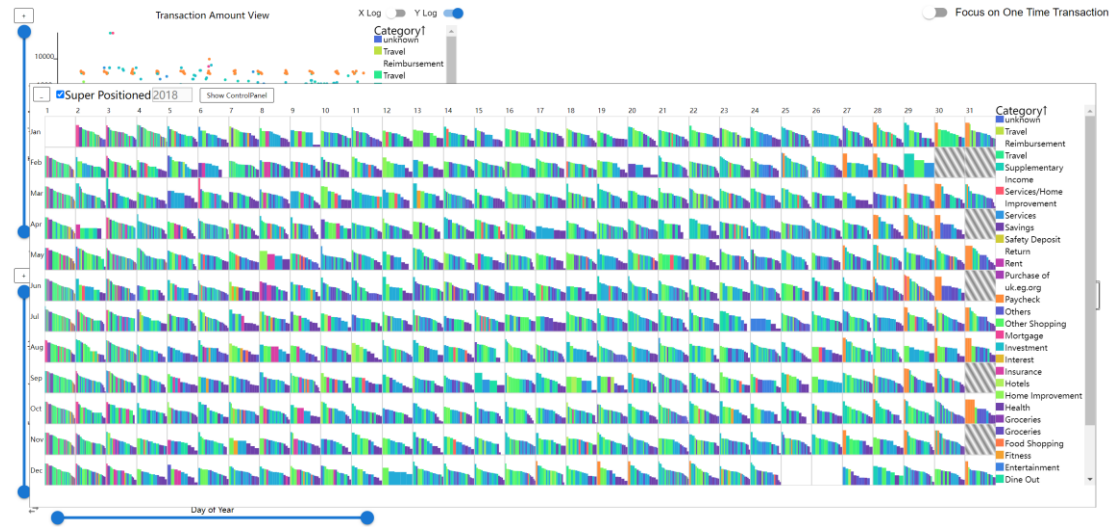


Figure 8.1.1.1. Expanded superpositioned calendar view with bar glyphs. Each bar represents a transaction record. The bars are sorted by their transaction amount, represented by height with logarithmic scale in descending order.

By clicking the colour legend, iBankEx presents the detail of the paycheck transactions, see Figure 8.1.1.2, whose cluster view shows that most paycheck transactions are in the cluster 0, and a few are in the cluster 1.

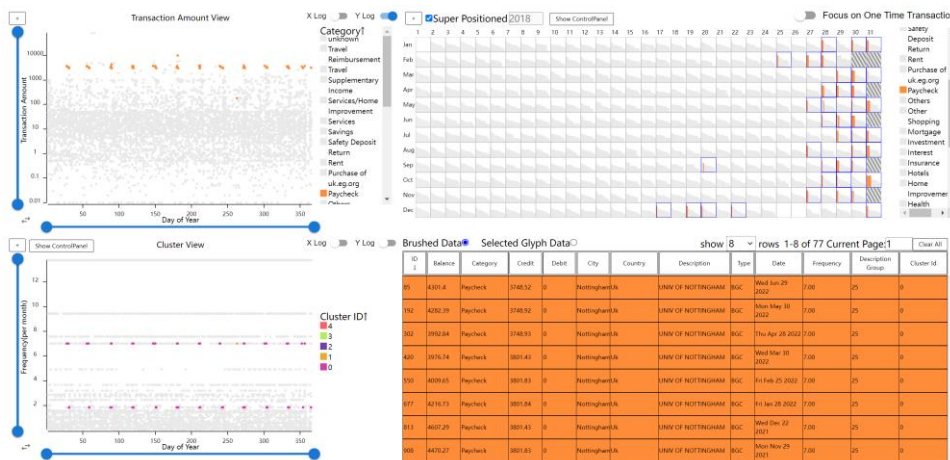


Figure 8.1.1.2. A Screenshot of iBankEx. The paycheck transactions are highlighted.

To get more information, star glyph is chosen, and the x axis of the cluster view is set to be transaction amount with logarithmic scale, see Figure 8.1.1.3, showing that the paycheck whose cluster id is 1 happens in the September 20, 2019. The reason it's different from the others is because its low amount (i.e., 186.29). The cluster view shows that some paycheck transaction has two frequencies (Figure 8.1.1.4A&B). By checking the date column, both transactions happen each month. The reason for the difference might be the incorrectness of transaction description grouping.

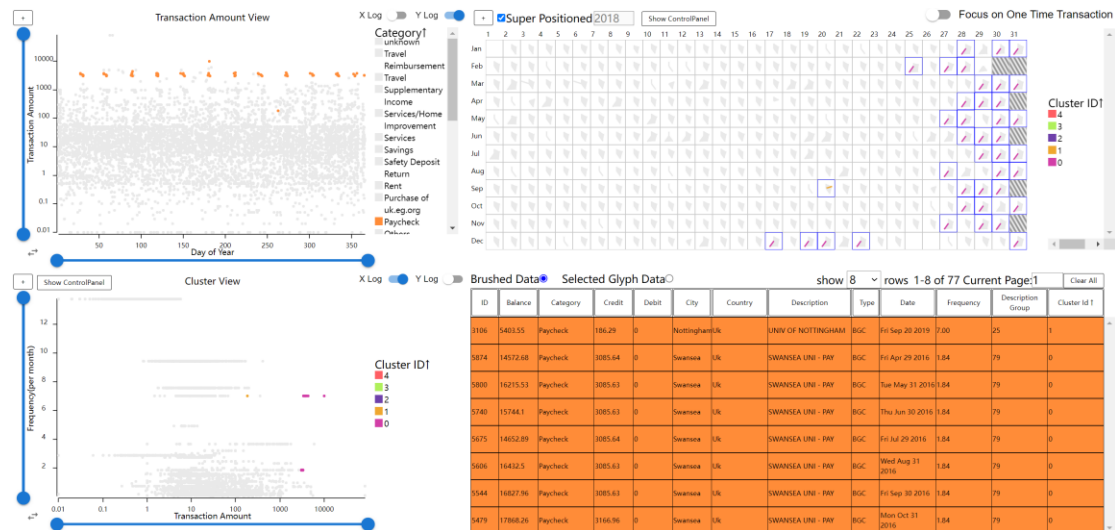


Figure 8.1.1.3. A Screenshot of iBankEx focusing on the paycheck transactions. The calendar view presents star glyphs. The rows in the table are ordered by cluster id.

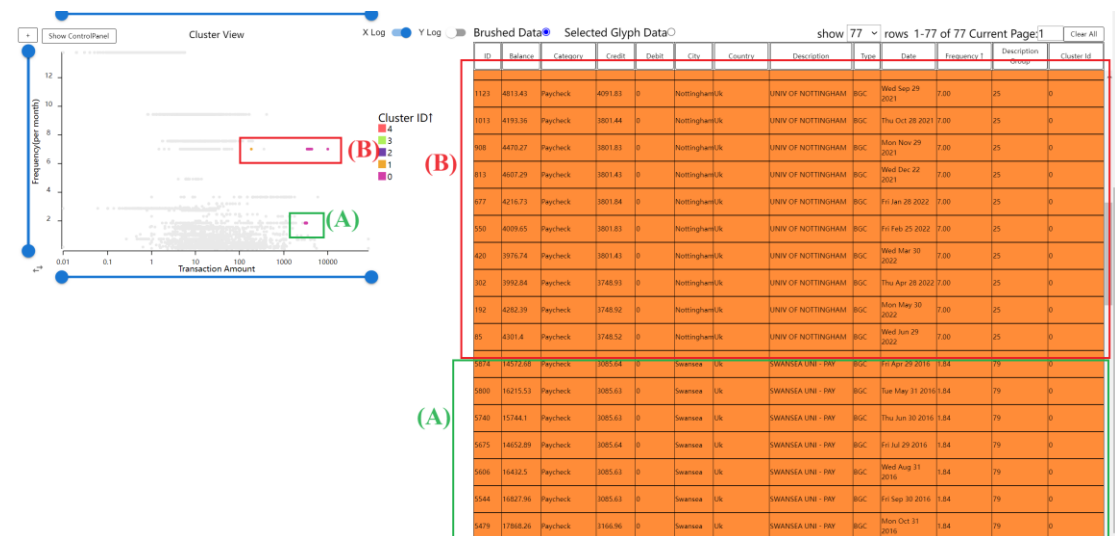


Figure 8.1.1.4. A Screenshot of cluster view and table view. (A) Transactions with lower frequency. (B) Transaction with higher frequency.

To investigate this hypothesis, the description group 25 is selected by the colour legend. As Figure 8.1.1.5 present, “LIDL GB NOTTINGHAM”, “KFC NOTTINGHAM” etc., are clustered in the same group with “UNIV OF NOTTINGHAM”. This means the transaction group is not clustered well for these descriptions.

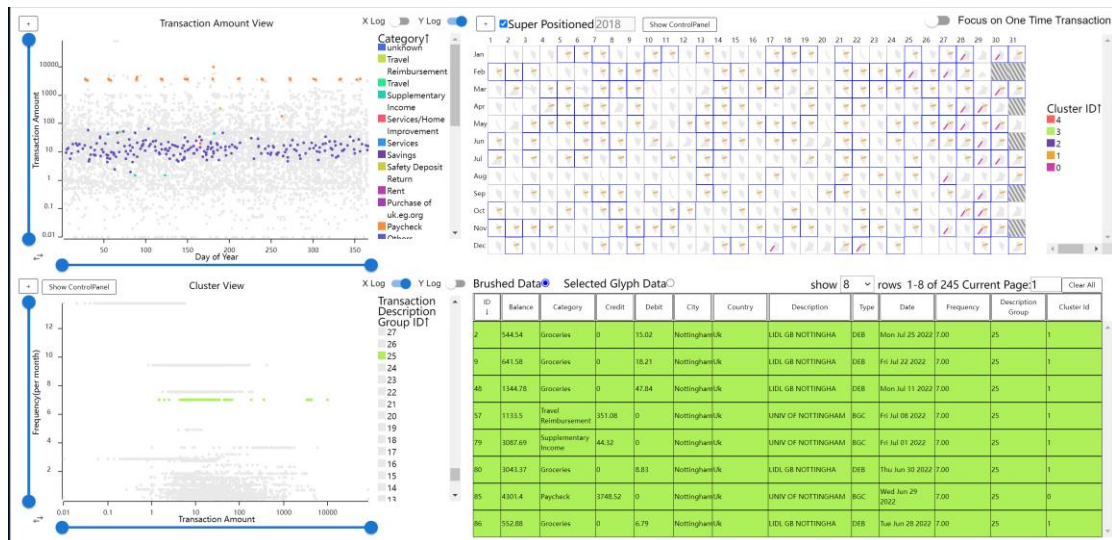


Figure 8.1.1.5. A Screenshot of iBankEx focusing on the description group 25.

To handle this, the transaction group unique key is adjusted to transaction description. As a result, the new frequency of these paycheck is become normal (i.e., 0.94 per month from Swansea university payment and 1 per month from Nottingham). This means this person has stable income.

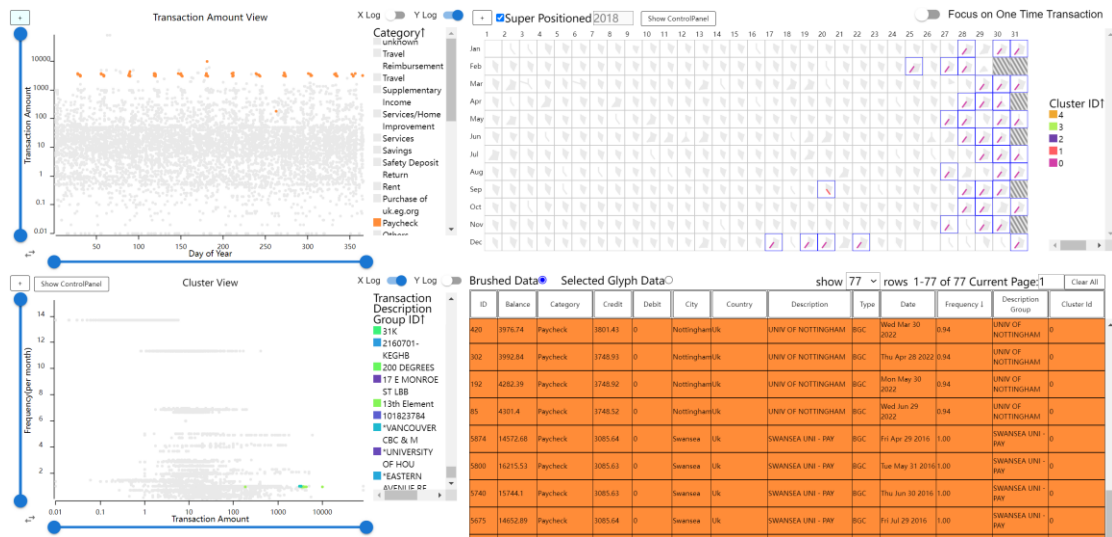


Figure 8.1.1.6. A screenshot of iBankEx. The transaction description is grouped by the description.

Another observation is that there is a paycheck (Figure 8.1.1.7A) much higher than the others. By brushing on this transaction, we can see the detail of it in the table view (see Figure 8.1.1.8). The transaction happens on Jun 29, 2019.

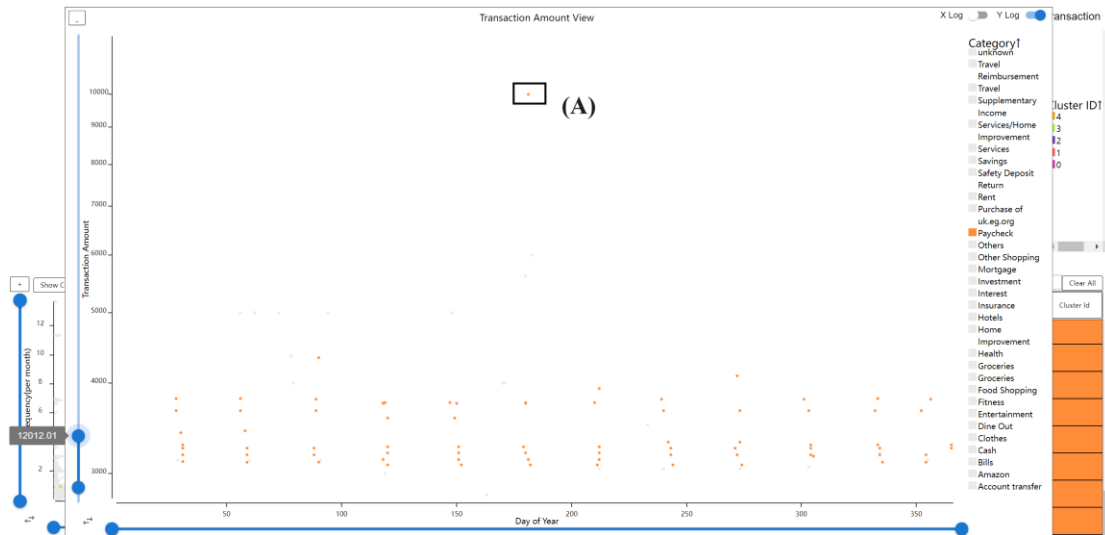


Figure 8.1.1.7. Expanded Transaction Amount View focusing on paycheck transaction. The view is zoomed in along the Y-Axis. (A) is the high amount paycheck transaction.

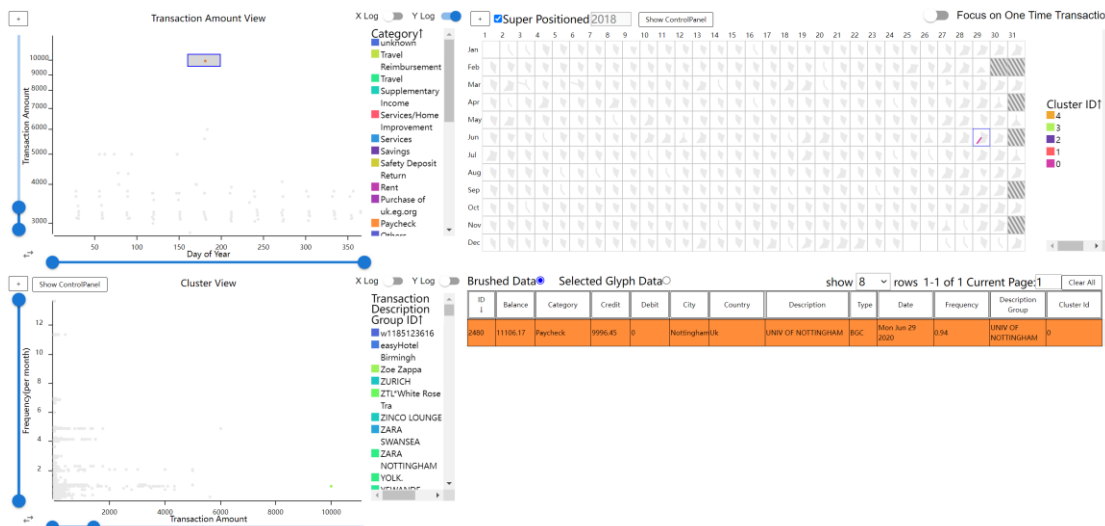


Figure 8.1.1.8. iBankEx focusing on the high amount paycheck transaction. The brusher is used for selecting the transaction.

8.1.2 Data Idiosyncrasies

The following were found during the project:

1. Incorrect location value as mentioned in section 3.
2. Category “Gorceries” with extra space at the end.
3. Inconsistent transaction description such as “LIDL GB NOTTINGHA” and “LIDL GB NOTTINGHAM”, “Amazon *Mktplce” EU and “AMAZON MKTPLACE PM”. This is solved by the “Transaction Description Grouping” feature provided by MoneyVis.
4. Consecutive empty space for transaction description, see Figure 8.1.2.1.

Table 8.2.1 present the response time when the user brushing on views, choose a colour label and click glyph to show detail in the table. Most of the response time are below 500 milliseconds. Most interaction are smooth enough except for the one when using bar glyph.

Table 8.2.1. The response time with different calendar view configuration for user brushing on views, choose a colour label or click glyph to show detail in the table.

Behaviour	Glyph	Calendar Year	Response Time
brush on transaction amount view	bar	2016	575ms
	pie		199ms
	polar area		243ms
	star		218ms
	bar	Superpositioned	491ms
	pie		358ms
	polar area		287ms
	star		229ms
brush on cluster view	bar	2016	528ms
	pie		209ms
	polar area		234ms
	star		214ms
	bar	Superpositioned	496ms
	pie		218ms
	polar area		319ms
	star		250ms
choose a colour label	bar	2016	552ms
	pie		242ms
	polar area		249ms
	star		225ms
	bar	Superpositioned	540ms
	pie		348ms
	polar area		297ms
	star		260ms
click glyph to show detail in the table	bar	2016	273ms
	pie		240ms
	polar area		262ms
	star		249ms
	bar	Superpositioned	257ms
	pie		231ms
	polar area		236ms
	star		239ms
change calendar view page	bar	2015 to 2022	171ms
	pie		177ms
	polar area		204ms
	star		194ms

The other behaviours including changing the page number of the table, expanding the views,

moving the sliders, swapping the axis, toggle the logarithmic scales, show one-time transactions, loading the page, updating the clustering configuration and transaction description group configuration and change the colour coding for the cluster view are also tested. Their response time are shown in the Table 8.2.2.

Table 8.2.2. Response time for the other behaviours.

Behaviour	Additional Notes	Response Time
table change page	8 rows per page	136ms
	17 rows per page	247ms
	50 rows per page	592ms
	100 rows per page	1230ms
	200 rows per page	2356ms
expanding transaction amount view		101ms
expanding calendar view		554ms
expanding cluster view		91ms
moving sliders	cluster view	62ms
swap axis		149ms
change log scales		115ms
focus on one time transaction		229ms
loading page		3119ms
update KMeans config		724ms
update transaction description group config		2873ms
change cluster view colour coding		96ms

9 Conclusion

In this study, a web-based interactive visualisation tool, iBankEx, was developed for exploring the first open retail bank transaction dataset of a single account. The tool has visualisation features such as multiple coordinated views, brush and linking, context and focus to facilitate the exploration of the data and utilised clustering algorithms. The tool presents overview of the data through a transaction amount view, a cluster view, and a novel superpositioned calendar view with different glyphs and a table view for detail. Sliders are provided for zooming and filtering. Transaction clustering and transaction description clustering are used for providing more insight of the dataset. The project is evaluated by case study and performance test. Additionally, this thesis contributes to the new knowledge of the data idiosyncrasies.

10 Future Work

In the future the following things can be done. The performance can be improved by using Canvas to render the calendar view and table view. Table lens can be applied to improve the effectiveness of the table view. Transactions can be aggregated to present derived data of the groups. These aggregated data can be presented as sortable glyphs like AgentVis (Rees et al., 2021) provides. Hovering and linking can also be provided to allow the user hover on a visual element and highlight the rest views. Undo and redo is also an useful feature for exploration (Shneiderman, 1996), which can be added in the future. The future researcher can also do more case studies to present more insight of the dataset. User test can be applied to gather the feedbacks for better user experience.

Reference List

- Anderson, C. (2015). Docker [Software engineering]. *IEEE Software*, 32(3), 102-c3. <https://doi.org/10.1109/MS.2015.62>
- Andrews, K., Traunmüller, T., Wolking, T., Goldgruber, E., Gutounig, R., & Ausserhofer, J. (2016). *Styrian Diversity Visualisation: Visualising Statistical Open Data with a LeanWeb App and Data Server*. The Eurographics Association. <https://doi.org/10.2312/eurp.20161150>
- API Reference: Create-next-app | Next.js*. (n.d.). Retrieved 5 September 2023, from <https://nextjs.org/docs/pages/api-reference/create-next-app>
- Arleo, A., Tsigkanos, C., Jia, C., Leite, R. A., Murturi, I., Klaffenböck, M., Dustdar, S., Wimmer, M., Miksch, S., & Sorger, J. (2019). Sabrina: Modeling and Visualization of Financial Data over Time with Incremental Domain Knowledge. *2019 IEEE Visualization Conference (VIS)*, 51–55. <https://doi.org/10.1109/VISUAL.2019.8933598>
- Arleo, A., Tsigkanos, C., Leite, R. A., Dustdar, S., Miksch, S., & Sorger, J. (2023). Visual Exploration of Financial Data with Incremental Domain Knowledge. *Computer Graphics Forum*, 42(1), 101–116. <https://doi.org/10.1111/cgf.14723>
- Bach, B., Freeman, E., Abdul-Rahman, A., Turkey, C., Khan, S., Fan, Y., & Chen, M. (2022). *Dashboard Design Patterns* (arXiv:2205.00757). arXiv. <https://doi.org/10.48550/arXiv.2205.00757>
- Bansal, P., & Ouda, A. (2022). Study on Integration of FastAPI and Machine Learning for Continuous Authentication of Behavioral Biometrics. *2022 International Symposium on Networks, Computers and Communications (ISNCC)*, 1–6. <https://doi.org/10.1109/ISNCC55209.2022.9851790>
- Borgo, R., Kehrer, J., Chung, D. H. S., Maguire, E., Laramée, R. S., Hauser, H., Ward, M., & Chen, M. (2013). *Glyph-based Visualization: Foundations, Design Guidelines, Techniques and Applications*. <https://doi.org/10.2312/conf/EG2013/stars/039-063>
- Brehmer, M., Lee, B., Bach, B., Riche, N. H., & Munzner, T. (2017). Timelines Revisited: A Design Space and Considerations for Expressive Storytelling. *IEEE Transactions on Visualization and Computer Graphics*, 23(9), 2151–2164. <https://doi.org/10.1109/TVCG.2016.2614803>
- Brito, H., Gomes, A., Santos, Á., & Bernardino, J. (2018). JavaScript in mobile applications: React native vs ionic vs NativeScript vs native development. *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, 1–6. <https://doi.org/10.23919/CISTI.2018.8399283>
- Byron, L., & Wattenberg, M. (2008). Stacked Graphs – Geometry & Aesthetics. *IEEE Transactions on Visualization and Computer Graphics*, 14(6), 1245–1252. <https://doi.org/10.1109/TVCG.2008.166>
- Camisetty, A., Chandurkar, C., Sun, M., & Koop, D. (2019). Enhancing Web-based Analytics Applications through Provenance. *IEEE Transactions on Visualization and Computer Graphics*, 25(1), 131–141. <https://doi.org/10.1109/TVCG.2018.2865039>
- Carminati, M., Caron, R., Maggi, F., Epifani, I., & Zanero, S. (2014). BankSealer: An Online Banking Fraud Analysis and Decision Support System. In N. Cuppens-Boulahia, F. Cuppens, S. Jajodia, A. Abou El Kalam, & T. Sans (Eds.), *ICT Systems Security and Privacy Protection* (pp. 380–394). Springer. https://doi.org/10.1007/978-3-642-55415-5_32

Ceneda, D., Gschwandtner, T., & Miksch, S. (2019). A Review of Guidance Approaches in Visual Data Analysis: A Multifocal Perspective. *Computer Graphics Forum*, 38(3), 861–879. <https://doi.org/10.1111/cgf.13730>

Chang, R., Ghoniem, M., Kosara, R., Ribarsky, W., Yang, J., Suma, E., Ziemkiewicz, C., Kern, D., & Sudjianto, A. (2007). WireVis: Visualization of Categorical, Time-Varying Data From Financial Transactions. *2007 IEEE Symposium on Visual Analytics Science and Technology*, 155–162. <https://doi.org/10.1109/VAST.2007.4389009>

Chang, R., Lee, A., Ghoniem, M., Kosara, R., Ribarsky, W., Yang, J., Suma, E., Ziemkiewicz, C., Kern, D., & Sudjianto, A. (2008). Scalable and Interactive Visual Analysis of Financial Wire Transactions for Fraud Detection. *Information Visualization*, 7(1), 63–76. <https://doi.org/10.1057/palgrave.ivs.9500172>

Comaniciu, D., & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5), 603–619. <https://doi.org/10.1109/34.1000236>

Create React App. (n.d.). Retrieved 27 March 2023, from <https://create-react-app.dev/>

D3 by Observable | The JavaScript library for bespoke data visualization. (n.d.). Retrieved 14 September 2023, from <https://d3js.org/>

Deshpande, A., & Barmish, B. R. (2016). A general framework for pairs trading with a control-theoretic point of view. *2016 IEEE Conference on Control Applications (CCA)*, 761–766. <https://doi.org/10.1109/CCA.2016.7587910>

Di Giacomo, E., Didimo, W., Liotta, G., & Palladino, P. (2010). Visual analysis of financial crimes: [System paper]. *Proceedings of the International Conference on Advanced Visual Interfaces*, 393–394. <https://doi.org/10.1145/1842993.1843073>

Didimo, W., Liotta, G., & Montecchiani, F. (2012). *VIS4AUI: VISUAL ANALYSIS OF BANKING ACTIVITY NETWORKS*. 2, 799–802. <https://doi.org/10.5220/0003933407990802>

Didimo, W., Liotta, G., & Montecchiani, F. (2014). Network visualization for financial crime detection. *Journal of Visual Languages & Computing*, 25(4), 433–451. <https://doi.org/10.1016/j.jvlc.2014.01.002>

Dingen, D., van't Veer, M., Houthuizen, P., Mestrom, E. H. J., Korsten, E. H. H. M., Bouwman, A. R. A., & van Wijk, J. (2019). RegressionExplorer: Interactive Exploration of Logistic Regression Models with Subgroup Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 25(1), 246–255. <https://doi.org/10.1109/TVCG.2018.2865043>

Docker: Accelerated Container Application Development. (2022, May 10). <https://www.docker.com/>

Dorogovtsev, S. N., Goltsev, A. V., & Mendes, J. F. F. (2006). K-core architecture and k-core percolation on complex networks. *Physica D: Nonlinear Phenomena*, 224(1), 7–19. <https://doi.org/10.1016/j.physd.2006.09.027>

Dwivedi, P., Kshamta, & Joshi, A. (2022). ReactJS For Trading Applications. *2022 International Conference on Cyber Resilience (ICCR)*, 01–07. <https://doi.org/10.1109/ICCR56254.2022.9995932>

Eades, P., & Huang, M. L. (2004). Navigating Clustered Graphs Using Force-Directed Methods. In *Graph Algorithms and Applications 2* (pp. 191–215). WORLD SCIENTIFIC.

https://doi.org/10.1142/9789812794741_0010

Ellis, G., & Dix, A. (2007). A Taxonomy of Clutter Reduction for Information Visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6), 1216–1223. <https://doi.org/10.1109/TVCG.2007.70535>

FastAPI. (n.d.). Retrieved 4 May 2023, from <https://fastapi.tiangolo.com/>

FastAPI in Containers—Docker—FastAPI. (n.d.). Retrieved 5 September 2023, from <https://fastapi.tiangolo.com/deployment/docker/>

Ferreira, F., Borges, H. S., & Valente, M. T. (2022). On the (un-)adoption of JavaScript front-end frameworks. *Software: Practice and Experience*, 52(4), 947–966. <https://doi.org/10.1002/spe.3044>

Firat, E. E., Vytla, D., Singh, N. V., Jiang, Z., & Laramée, R. S. (2023). *MoneyVis: Open Bank Transaction Data for Visualization and Beyond*.

Fischer, L., & Hanenberg, S. (2015). An empirical investigation of the effects of type systems and code completion on API usability using TypeScript and JavaScript in MS visual studio. *ACM SIGPLAN Notices*, 51(2), 154–167. <https://doi.org/10.1145/2936313.2816720>

Fuchs, J. (2015). *Glyph Design for Temporal and Multi-Dimensional Data: Design Considerations and Evaluation*. <https://kops.uni-konstanz.de/handle/123456789/33691>

Fuchs, J., Isenberg, P., Bezerianos, A., Miller, M., & Keim, D. (2019). *EduClust -A Visualization Application for Teaching Clustering Algorithms*. 1. <https://inria.hal.science/hal-02103293>

Fuchs, J., Jäckle, D., Weiler, N., & Schreck, T. (2016). Leaf Glyphs: Story Telling and Data Analysis Using Environmental Data Glyph Metaphors. In J. Braz, J. Pettré, P. Richard, A. Kerren, L. Linsen, S. Battiato, & F. Imai (Eds.), *Computer Vision, Imaging and Computer Graphics Theory and Applications* (pp. 123–143). Springer International Publishing. https://doi.org/10.1007/978-3-319-29971-6_7

Gao, Z., Bird, C., & Barr, E. T. (2017). To Type or Not to Type: Quantifying Detectable Bugs in JavaScript. *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 758–769. <https://doi.org/10.1109/ICSE.2017.75>

Ghosh, S., Datta, A., Tan, K., & Choi, H. (2019). SLIDE – a web-based tool for interactive visualization of large-scale – omics data. *Bioinformatics*, 35(2), 346–348. <https://doi.org/10.1093/bioinformatics/bty534>

Gleicher, M. (2018). Considerations for Visualizing Comparison. *IEEE Transactions on Visualization and Computer Graphics*, 24(1), 413–423. <https://doi.org/10.1109/TVCG.2017.2744199>

Goel, A. K., Bisht, V. S., & Chaudhary, S. (2023). Multisignature Crypto Wallet Paper. *2023 8th International Conference on Communication and Electronics Systems (ICCES)*, 476–479. <https://doi.org/10.1109/ICCES57224.2023.10192591>

Graphviz. (n.d.). Graphviz. Retrieved 5 September 2023, from <https://graphviz.org/>

Guo, H., Liu, M., Yang, B., Sun, Y., Qu, H., & Shi, L. (2022). RankFIRST: Visual Analysis for Factor Investment By Ranking Stock Timeseries. *IEEE Transactions on Visualization and Computer Graphics*, 1–10. <https://doi.org/10.1109/TVCG.2022.3209414>

Hao, J., & Ho, T. K. (2019). Machine Learning Made Easy: A Review of Scikit-learn Package

in Python Programming Language. *Journal of Educational and Behavioral Statistics*, 44(3), 348–361. <https://doi.org/10.3102/1076998619832248>

Havre, S., Hetzler, B., & Nowell, L. (2000). ThemeRiver: Visualizing theme changes over time. *IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings*, 115–123. <https://doi.org/10.1109/INFVIS.2000.885098>

He, Y., & Li, H. (2022). Optimal layout of stacked graph for visualizing multidimensional financial time series data. *Information Visualization*, 21(1), 63–73. <https://doi.org/10.1177/14738716211045005>

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages & Computing*, 13(3), 259–290. <https://doi.org/10.1006/jvlc.2002.0237>

JavaScript With Syntax For Types. (n.d.). Retrieved 2 September 2023, from <https://www.typescriptlang.org/>

Joachims, T. (2002). Optimizing Search Engines using Clickthrough Data. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. <https://doi.org/10.1145/775047.775067>

Kirkland, J. D., Senator, T. E., Hayden, J. J., Dybala, T., Goldberg, H. G., & Shyr, P. (1999). The NASD Regulation Advanced-Detection System (ads). *AI Magazine*, 20(1), 55–67. <https://doi.org/10.1609/aimag.v20i1.1440>

Ko, S., Cho, I., Afzal, S., Yau, C., Chae, J., Malik, A., Beck, K., Jang, Y., Ribarsky, W., & Ebert, D. S. (2016). A Survey on Visual Analysis Approaches for Financial Data. *Computer Graphics Forum*, 35(3), 599–617. <https://doi.org/10.1111/cgf.12931>

Langner, R., Kister, U., & Dachsel, R. (2019). Multiple Coordinated Views at Large Displays for Multiple Users: Empirical Findings on User Behavior, Movements, and Distances. *IEEE Transactions on Visualization and Computer Graphics*, 25(1), 608–618. <https://doi.org/10.1109/TVCG.2018.2865235>

Laramee, R. S. (2010). How to Write a Visualization Research Paper: A Starting Point. *Computer Graphics Forum*, 29(8), 2363–2371. <https://doi.org/10.1111/j.1467-8659.2010.01748.x>

Laramee, R. S. (2011). How to Read a Visualization Research Paper: Extracting the Essentials. *IEEE Computer Graphics and Applications*, 31(3), 78–82. <https://doi.org/10.1109/MCG.2011.44>

Laramee, R. S. (2021a). *Bob's Minutes of Meeting Protocol: Incentive and a Description*.

Laramee, R. S. (2021b). *Bob's Project Guidelines: Writing a Dissertation for a BSc or MSc in Computer Science*.

Lavoué, G., Chevalier, L., & Dupont, F. (2013). *Streaming compressed 3D data on the web using JavaScript and WebGL*. 19–27. <https://doi.org/10.1145/2466533.2466539>

Lei, S. T., & Zhang, K. (2010). A visual analytics system for financial time-series data. *Proceedings of the 3rd International Symposium on Visual Information Communication*, 1–9. <https://doi.org/10.1145/1865841.1865868>

Leite, R. A., Arleo, A., Sorger, J., Gschwandtner, T., & Miksch, S. (2020). Hermes: Guidance-enriched Visual Analytics for economic network exploration. *Visual Informatics*, 4(4), 11–22.

<https://doi.org/10.1016/j.visinf.2020.09.006>

Leite, R. A., Gschwandtner, T., Miksch, S., Gstrein, E., & Kuntner, J. (2016). *Visual Analytics for Fraud Detection: Focusing on Profile Analysis*. The Eurographics Association. <https://doi.org/10.2312/eurp.20161138>

Leite, R. A., Gschwandtner, T., Miksch, S., Gstrein, E., & Kuntner, J. (2018). Visual analytics for event detection: Focusing on fraud. *Visual Informatics*, 2(4), 198–212. <https://doi.org/10.1016/j.visinf.2018.11.001>

Leite, R. A., Gschwandtner, T., Miksch, S., Gstrein, E., & Kuntner, J. (2020). NEVA: Visual Analytics to Identify Fraudulent Networks. *Computer Graphics Forum*, 39(6), 344–359. <https://doi.org/10.1111/cgf.14042>

Leite, R. A., Gschwandtner, T., Miksch, S., Kriglstein, S., Pohl, M., Gstrein, E., & Kuntner, J. (2018). EVA: Visual Analytics to Identify Fraudulent Events. *IEEE Transactions on Visualization and Computer Graphics*, 24(1), 330–339. <https://doi.org/10.1109/TVCG.2017.2744758>

Li, D., Mei, H., Shen, Y., Su, S., Zhang, W., Wang, J., Zu, M., & Chen, W. (2018). ECharts: A declarative framework for rapid construction of web-based visualization. *Visual Informatics*, 2(2), 136–146. <https://doi.org/10.1016/j.visinf.2018.04.011>

Li, X., & Bao, Z. (2014). Performance Characterization of Web Applications with HTML5 Enhancements. *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, 252–258. <https://doi.org/10.1109/DASC.2014.52>

Lin, C., Sun, Q., & Wang, C. (2019). Web-based management and visualization of myocardial ischemia data. *2019 Chinese Automation Congress (CAC)*, 5424–5428. <https://doi.org/10.1109/CAC48633.2019.8997288>

Liu, Q. Q., Li, Q., Tang, C. F., Lin, H. B., Peng, Z. H., Li, Z. W., & Chen, T. J. (2020). *Visual Analysis of Car-hailing Reimbursement Data for Overtime*. The Eurographics Association. <https://doi.org/10.2312/eurp.20201119>

Liu, Q. Q., Li, Q., Zhu, Z., Ye, T., & Ma, X. (2021). Inspecting the Process of Bank Credit Rating via Visual Analytics. *2021 IEEE Visualization Conference (VIS)*, 136–140. <https://doi.org/10.1109/VIS49827.2021.9623312>

Liu, X., Alharbi, M., Best, J., Chen, J., Diehl, A., Firat, E., Rees, D., Wang, Q., & Laramée, R. S. (2021). Visualization Resources: A Starting Point. *2021 25th International Conference Information Visualisation (IV)*, 160–169. <https://doi.org/10.1109/IV53921.2021.00034>

Liu, X., Alharbi, M. S., Chen, J., Diehl, A., Rees, D., Firat, E. E., Wang, Q., & Laramée, R. S. (2023). Visualization Resources: A Survey. *Information Visualization*, 22(1), 3–30. <https://doi.org/10.1177/14738716221126992>

Luraschi, J. (2023). *Awesome Dataviz* [Computer software]. <https://github.com/javierluraschi/awesome-dataviz> (Original work published 2015)

Lusardi, A., & Mitchell, O. S. (2005). *Financial Literacy and Planning: Implications for Retirement Wellbeing*. Michigan Retirement Research Center Research.

Maaten, L. van der, & Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86), 2579–2605.

Maçãs, C., Polisciuc, E., & Machado, P. (2020). VaBank: Visual Analytics for Banking

- Transactions. *2020 24th International Conference Information Visualisation (IV)*, 336–343. <https://doi.org/10.1109/IV51561.2020.00062>
- Maças, C., Polisciuc, E., & Machado, P. (2022). ATOVis – A visualisation tool for the detection of financial fraud. *Information Visualization*, 21(4), 371–392. <https://doi.org/10.1177/14738716221098074>
- Manipulating the DOM with Refs – React*. (n.d.). Retrieved 2 September 2023, from <https://react.dev/learn/manipulating-the-dom-with-refs>
- Mao, W. (2023). *finVis STAR: The State of the Art in Financial Information Visualisation*.
- Material UI: React components based on Material Design*. (n.d.). Retrieved 2 September 2023, from <https://mui.com/material-ui/>
- Matrix, & SUS. (2011). *UsabiliTEST. Usability testing: Card Sorting, Prioritization*. <https://www.usabilitest.com/>
- Mckinney, W. (2011). pandas: A Foundational Python Library for Data Analysis and Statistics. *Python High Performance Science Computer*.
- McNabb, L., & Laramée, R. S. (2017). Survey of Surveys (SoS)—Mapping The Landscape of Survey Papers in Information Visualization. *Computer Graphics Forum*, 36(3), 589–617. <https://doi.org/10.1111/cgf.13212>
- McNabb, L., & Laramée, R. S. (2019a). *How to Write a Visualization Survey Paper: A Starting Point*.
- McNabb, L., & Laramée, R. S. (2019b). Multivariate Maps—A Glyph-Placement Algorithm to Support Multivariate Geospatial Visualization. *Information*, 10(10), Article 10. <https://doi.org/10.3390/info10100302>
- Nakano, M., Takahashi, A., & Takahashi, S. (2017). Generalized exponential moving average (EMA) model with particle filtering and anomaly detection. *Expert Systems with Applications*, 73, 187–200. <https://doi.org/10.1016/j.eswa.2016.12.034>
- Next.js by Vercel—The React Framework*. (n.d.). Retrieved 2 September 2023, from <https://nextjs.org/>
- North, C., & Shneiderman, B. (2000). Snap-together visualization: A user interface for coordinating visualizations via relational schemata. *Proceedings of the Working Conference on Advanced Visual Interfaces*, 128–135. <https://doi.org/10.1145/345513.345282>
- NumPy*. (n.d.). Retrieved 3 September 2023, from <https://numpy.org/>
- pandas—Python Data Analysis Library*. (n.d.). Retrieved 8 May 2023, from <https://pandas.pydata.org/>
- Park, N. J., George, K. M., & Park, N. (2010). A multiple regression model for trend change prediction. *2010 International Conference on Financial Theory and Engineering*, 22–26. <https://doi.org/10.1109/ICFTE.2010.5499430>
- Pham, T., & Lee, S. (2017). *Anomaly Detection in Bitcoin Network Using Unsupervised Learning Methods* (arXiv:1611.03941). <http://arxiv.org/abs/1611.03941>
- Pylint—Code analysis for Python* | www.pylint.org. (n.d.). Retrieved 5 September 2023, from <https://www.pylint.org/>

Python Operating Systems List. (n.d.). Python.Org. Retrieved 2 September 2023, from <https://www.python.org/downloads/operating-systems/>

Ranjani, J., Sheela, A., & Meena, K. P. (2019). Combination of NumPy, SciPy and Matplotlib/PyLab -a good alternative methodology to MATLAB - A Comparative analysis. *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, 1–5. <https://doi.org/10.1109/ICIICT1.2019.8741475>

React. (n.d.). Retrieved 2 September 2023, from <https://react.dev/>

React Developer Tools – React. (n.d.). Retrieved 4 September 2023, from <https://react.dev/learn/react-developer-tools>

React Redux | React Redux. (n.d.). Retrieved 14 September 2023, from <https://react-redux.js.org/>

Redux Toolkit | Redux Toolkit. (n.d.). Retrieved 14 September 2023, from <https://redux-toolkit.js.org/>

Redux—A predictable state container for JavaScript apps. | Redux. (n.d.). Retrieved 2 September 2023, from <https://redux.js.org/>

Rees, D., & Laramée, R. S. (2019). A Survey of Information Visualization Books. *Computer Graphics Forum*, 38(1), 610–646. <https://doi.org/10.1111/cgf.13595>

Rees, D., Laramée, R. S., Brookes, P., D’Cruze, T., Smith, G. A., & Miah, A. (2021). AgentVis: Visual Analysis of Agent Behavior With Hierarchical Glyphs. *IEEE Transactions on Visualization and Computer Graphics*, 27(9), 3626–3643. <https://doi.org/10.1109/TVCG.2020.2985923>

Roberts, J. C. (2007). State of the Art: Coordinated & Multiple Views in Exploratory Visualization. *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007)*, 61–71. <https://doi.org/10.1109/CMV.2007.20>

Roberts, R., & Laramée, R. (2018). Visualising Business Data: A Survey. *Information*, 9(11), 285. <https://doi.org/10.3390/info9110285>

Rogovschi, N., Lebbah, M., & Bennani, Y. (2011). A SELF-ORGANIZING MAP FOR MIXED CONTINUOUS AND CATEGORICAL DATA. *International Journal of Computing*, 24–32. <https://doi.org/10.47839/ijc.10.1.733>

Rudolph, S., Savikhin, A., & Ebert, D. S. (2009). FinVis: Applied visual analytics for personal financial planning. *2009 IEEE Symposium on Visual Analytics Science and Technology*, 195–202. <https://doi.org/10.1109/VAST.2009.5333920>

Run a Server Manually—Uvicorn—FastAPI. (n.d.). Retrieved 3 September 2023, from <https://fastapi.tiangolo.com/deployment/manually/>

Run JavaScript Everywhere. (n.d.). Run JavaScript Everywhere. Retrieved 18 September 2023, from <https://nodejs.dev/en/>

Ruppert, T., Bannach, A., Bernard, J., Lokanc, M., & Kohlhammer, J. (2017). *Visual Access to Performance Indicators in the Mining Sector*. The Eurographics Association. <https://doi.org/10.2312/eurovisshort.20171150>

Santos, R., Murrieta-Flores, P., & Martins, B. (2018). Learning to combine multiple string similarity metrics for effective toponym matching. *International Journal of Digital Earth*, 11(9),

913–938. <https://doi.org/10.1080/17538947.2017.1371253>

Sarikaya, A., Correll, M., Bartram, L., Tory, M., & Fisher, D. (2019). What Do We Talk About When We Talk About Dashboards? *IEEE Transactions on Visualization and Computer Graphics*, 25(1), 682–692. <https://doi.org/10.1109/TVCG.2018.2864903>

Scikit-learn: Machine learning in Python—Scikit-learn 1.3.0 documentation. (n.d.). Retrieved 3 September 2023, from <https://scikit-learn.org/stable/>

SciPy. (n.d.). Retrieved 3 September 2023, from <https://scipy.org/>

Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S., & Edwards, S. H. (2010). Algorithm Visualization: The State of the Field. *ACM Transactions on Computing Education*, 10(3), 9:1-9:22. <https://doi.org/10.1145/1821996.1821997>

Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. *Proceedings 1996 IEEE Symposium on Visual Languages*, 336–343. <https://doi.org/10.1109/VL.1996.545307>

Simon, H. A. (1960). *The New Science of Management Decision*. Harper.

Singh, K., & Best, P. (2019). Anti-Money Laundering: Using data visualization to identify suspicious activity. *International Journal of Accounting Information Systems*, 34, 100418. <https://doi.org/10.1016/j.accinf.2019.06.001>

Stančin, I., & Jović, A. (2019). An overview and comparison of free Python libraries for data mining and big data analysis. *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 977–982. <https://doi.org/10.23919/MIPRO.2019.8757088>

Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. (n.d.). Retrieved 2 September 2023, from <https://tailwindcss.com/>

Tegarden, D. P. (1999). Business Information Visualization. *Communications of the Association for Information Systems*, 1(1). <https://doi.org/10.17705/1CAIS.00104>

Tsang, K. W., Li, H., Lam, F. M., Mu, Y., Wang, Y., & Qu, H. (2020). TradAO: A Visual Analytics System for Trading Algorithm Optimization. *2020 IEEE Visualization Conference (VIS)*, 61–65. <https://doi.org/10.1109/VIS47514.2020.00019>

Turk, J. (n.d.). *jellyfish: Approximate and phonetic matching of strings*. (1.0.0) [Python, Rust; OS Independent].

Using React with D3.js. (n.d.). Retrieved 2 September 2023, from <https://wattenberger.com/blog/react-and-d3>

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... van Mulbregt, P. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3), Article 3. <https://doi.org/10.1038/s41592-019-0686-2>

Wagner, B. (2018). *Learning Web Programming with TypeScript 2.0 and Angular 2.0* (1st edition). Addison Wesley.

Ward, M. O. (2002). A Taxonomy of Glyph Placement Strategies for Multidimensional Data Visualization. *Information Visualization*, 1(3–4), 194–210.

<https://doi.org/10.1057/PALGRAVE.IVS.9500025>

Ward, M. O. (2008). Multivariate Data Glyphs: Principles and Practice. In C. Chen, W. Härdle, & A. Unwin (Eds.), *Handbook of Data Visualization* (pp. 179–198). Springer. https://doi.org/10.1007/978-3-540-33037-0_8

Ward, M. O., & Lipchak, B. N. (2000). A visualization tool for exploratory analysis of cyclic multivariate data. *Metrika*, *51*(1), 27–37. <https://doi.org/10.1007/s001840000042>

Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.

Welcome to Python.org. (2023, August 29). Python.Org. <https://www.python.org/about/>

What are the Advantages of TypeScript over JavaScript? | Altogic - Backend as a service platform. (2023, February 5). <https://www.altogic.com/blog/what-are-the-advantages-of-typescript-over-javascript>

Writing Markup with JSX – React. (n.d.). Retrieved 4 September 2023, from <https://react.dev/learn/writing-markup-with-jsx>

Yao, J., Lin, C., Xie, X., Wang, A. J., & Hung, C.-C. (2010). Path Planning for Virtual Human Motion Using Improved A* Star Algorithm. *2010 Seventh International Conference on Information Technology: New Generations*, 1154–1158. <https://doi.org/10.1109/ITNG.2010.53>

Yue, X., Gu, Q., Wang, D., Qu, H., & Wang, Y. (2021). iQUANT: Interactive Quantitative Investment Using Sparse Regression Factors. *Computer Graphics Forum*, *40*(3), 189–200. <https://doi.org/10.1111/cgf.14299>

Yue, X., Shu, X., Zhu, X., Du, X., Yu, Z., Papadopoulos, D., & Liu, S. (2019). BitExTract: Interactive Visualization for Extracting Bitcoin Exchange Intelligence. *IEEE Transactions on Visualization and Computer Graphics*, *25*(1), 162–171. <https://doi.org/10.1109/TVCG.2018.2864814>

Zhao, J., Forer, P., & Harvey, A. S. (2008). Activities, ringmaps and geovisualization of large human movement fields. *Information Visualization*, *7*(3–4), 198–209. <https://doi.org/10.1057/PALGRAVE.IVS.9500184>

Appendix 1. Meeting Minutes

The meeting minutes can be accessed by this link:

<https://docs.google.com/document/d/1jvSTDCbzuLoCJahr0fWbhZxJNiqvHVj1KQ0yNIy46b0/edit#heading=h.eenmnoe2v7f8>.

Appendix 2. Performance Test

The screenshot of the performance test can be accessed by this link:

<https://github.com/TomTomMao/iBankEx-Performance>.

Appendix 3. Python Documentation

The documentation was generated by pydoc and Fastapi.

The documentation can be accessed via this link: <https://github.com/TomTomMao/iBankEx-Python-doc>.

Appendix 4. Source Code

This source code is in the git repository: <https://github.com/TomTomMao/summer-project-msc>.