Building Responsible, Data-Driven Visualization Recommendation Systems
CMSC828D Project Report

Suleyman Aslan*
Department of Computer Science
University of Maryland

Chen Chen†
Department of Computer Science
University of Maryland

Ethan Remsberg‡
Department of Computer Science
University of Maryland

Tianshu Xu§
Department of Computer Science
University of Maryland

Yufan Zheng¶
Department of Computer Science
University of Maryland

ABSTRACT

Multiple state-of-the-art visualization tools, such as D3 and Vega-Lite, emerge to make visualization generalization easy for researchers, data scientists and even for users without programming experience and background knowledge. People could largely benefit from these easy-to-use systems and tools to create visualizations and run analyses. However, there are data issue detection and data issue cleaning either not supported in the tools or can not be easily used. Moreover, there is a need that visualization systems should not only to output required charts but also to flag and explain potential issues within users' data. Therefore, based upon a previous prototype, we present a visualization recommendation system which can detect potential statistical data issues and suggest possible solutions. We propose refined methods on building collection of implementable and commonly-seen data issues, a function that accepts datasets from the user's side, an interactive user interface that is in a clear layout, and an educative visual recommendation panel including necessary explanations and possible solutions for detected data issues. We also present a user study involving 4 participants to validate the effectiveness and usability of our system.

Index Terms: Database—Data Visualization—Statistics; Informative Prediction—Recommendation System

1 INTRODUCTION

Visualizations are probably the most important tool that people from various backgrounds use to illustrate their ideas and findings. This is also the reason that most programming languages have their own built-in visualization toolkits, such as ggplot2 in R [44] and pyplot in Python [1]. To meet the needs of non-programmers who do not use programming languages much in their daily work, many visualization tools under the framework of the Grammar of graphics [45] emerges, e.g., Vega [3], D3 [9] and Vega-Lite [34]. Without much experience with programming, even novice users can build expressive visualizations using their own data after a short learning period.

Despite the effectiveness and convenience, these visualization tools have brought, potential problems related to the data itself (e.g., data quality) can be overlooked easily: easy-to-use systems allow users to create fine visualizations and run analyses even without having to see and understand their data. When problems occur, it is mostly left up to the users to determine which parts of the data have issues by really looking at the data itself, which is difficult and inconvenient especially in the big data regime. Also, it cannot be expected much that users can appropriately check and clean their data since the visualization tools are built for broad users many of whom might have less experience of database and programming. Thus, there is a need that visualization systems not only to output required charts but also to flag and explain potential issues within users' data. In this way, users can be educated to learn their data better and potentially address some issues using their own domain knowledge by interacting with the system and further produce correct and statistically sound visualizations.

Designing this kind of system is certainly not an easy task, and we list three major challenges here. **First**, which set of data issues to target in the system need to be considered carefully. Basically, we want the issues that the system can detect to be commonly seen in general datasets, not too hard to detect and address, and influential in the quality of the generated visualization. The automation of the checking needs to be fast (ideally as quick as the generalization of the visualization) to avoid too much latency, which poses challenges over the implementation of the detection process. Apart from the efficiency, automatic data cleaning without domain knowledge is known to be a difficult problem [4]. **Second**, the user should be able to interact with the system easily and intuitively even without a sufficient statistics background. This requires the system to output easy-to-understand and clear explanations over the detected issues, and provide possible solutions to them. The information display should be obvious other than disruptive and should balance between quick information and explanatory details. **Third**, the checking for data issues should work for personal datasets from the user. The system is built for helping users better understand and debug their own datasets, making the ability of generalization to different kinds of datasets of the system important. In general, dealing with a large dataset space without user inputs is a hard task, let alone there could potentially exist several kinds of statistical issues. In summary, balancing all of the challenges above is a tough exercise in good user-driven design and implementation.

There is literature in the field of visualization studying the pitfalls and challenges of visual analysis [10, 25, 27], which, however, start with the visual perception side without flagging potential data quality problems as an issue. In [25], the authors indeed raised the existence of potential data quality issues by saying "either designed to be insensitive to data quality issues through the employment of data cleaning methods or to explicitly visualize errors and uncertainty in the application to make the analyst aware of the problem", however, without discussing possible solutions. Instead, they assumed the user has sufficient expert knowledge to tackle the data issues accordingly, which is not realistic nowadays given the developing trend

*e-mail: aslan@umd.edu
†e-mail: cchen24@cs.umd.edu
‡e-mail: eremsber@umd.edu
§e-mail: txu@umd.edu
¶e-mail: yfzheng@umd.edu

and the broadly targeted users of visualization tools as discussed before. In the field of database and data management, people who consider the data cleaning task mostly operate directly upon the data itself [4, 31, 33, 37], meaning that this is not a feature injected into any visualization system and the user may not be able to visually learn the detected issues and erroneous data. A few works, such as [29, 32], considered combining some visual components and the data cleaning process, however, in an opposite manner: using visualizations to better illustrate and detect potential data issues and perform cleaning. The visual components used are rather basic, and the final goal of the two papers is to produce data with the good quality other than high-quality visualizations generated from clean data. Apart from the literature from the two fields above, we do notice that some visual analytic software does have some basic data quality checking, e.g., Tableau [2] automatically filters out nulls in the final visualizations. However, to the best of our knowledge, a visualization recommendation system embedded with meaningful statistical checking over data to help the user better understand and explore their datasets is currently missing.

In this project, we build upon a previous prototype called *StatCheck1.0*, which we would refer to as *StatCheck1.0* later in this report, to produce a visualization recommendation system that helps the user perform certain kinds of data issue checking and suggest potential solutions. Based on our experiences testing *StatCheck1.0*, we propose to improve it by reorganizing the layout to make it more clear and informative and supporting more meaningful interactions between the recommendation interface and the user. In summary, our contributions lie in three folders:

1. We re-organized and modified the original layout of the system, making it easier for the user to follow the pipeline and the positions of different charts more balanced. We also added a personal data uploading feature to allow customized dataset checking;

2. We enhanced the interaction between the user and the checking results, e.g., for the outlier detection, a slider is added to allow the user to control the percentage of data being filtered out; when multiple issues are presented, we allow the user to delete some of them based on their domain knowledge and update the visualization accordingly;

3. We presented a user study which involved 4 participant. We validated the usability and effectiveness of our system using the evaluation results compared to *StatCheck1.0*.

2 Related Work

Data visualization is crucial for researchers and data analyzers during the process of data analyzing and processing. The visualization results normally explains the significance of data to people who are visually oriented [30], and provides users with intuitive means to interactively explore and analyze data, enabling them to effectively identify interesting patterns, infer correlations and causalities, and supports sense-making activities [8].

Although there are different kinds of data visualization emerging and developed in recent years, challenges and unsophisticated are still largely existing in this area induced by people with different visualization-related skills. For example, there might be gap between data characterization tools, visualization design tools, and development platforms under the condition that many teams may include both designers who create new visualization designs and developers who implement the resulting visualization software [40]. While database researchers designing a new interactive analysis system for exploring large datasets, pitfalls may also occur in the design if visual perception is ignored. To ease this challenge, researchers investigated properties of crowdsourced for graphical perception research and using Amazon's Mechanical Turk to evaluate visualizations [20].

It is common for making prediction and recommendation in interaction data visualization tools [41]. The idea of predicting future is achieved by making passively observations clicks from the past and reported an average 95% on prediction accuracy. Along with that, the authors also present a framework for automatically learning future click events during data exploration and demonstrate. A visualization recommendation engine SEEDB [38] also developed to facilitate fast visual analysis. By study a given subset of data, SEEDB explores the space of visualizations, evaluates promising visualizations for trends, and recommends those it deems most "useful" or "interesting".

The idea of exploratory analysis via faceted browsing of visualization recommendations [46] is to make visualization recommendations according to statistical and perceptual measures. Brown et al. [11] accurately predict a user's task performance and infer some user personality traits by using machine learning techniques to analyze interaction data. More recently, Battle et al. [7] designed a tool for exploratory browsing of large datasets. By utilizing a client-server architecture, the authors present a two-level prediction engine, with an SVM classifier at the top level to predict the user's current analysis phase, and recommendation models at the bottom to predict low-level interaction patterns. Early work experiments [17] made contribution to visualization recommendation by monitoring users' behavior for implicit signals of user intent, and therefore provide more effective recommendations. In order to make visual exploration tool scale well and more adaptive to huge datasets, as well as to support exploratory activities [15], the authors apply semantic caching of active query sets and several prefetching strategies to exploiting characteristics of the visual exploration environment.

Statistical data visualization via interactive dashboard has been studied extensively in recent years. There are existing work making statistical graphics using Python, matplotlib, and integrate with pandas library [43]. The statistical, interactive data visualization also introduced to the field of medical and health care. In [26], Ko et al. present a study that suggests the procedures of efficient visualization of big data for general healthcare researchers. Statistical data visualization is also be concerned by replace the dataset with a distribution based proxy representation that summarizes scalar information into a much reduced memory footprint [42]. Statistical modeling of data is largely being used in many other different fields such as financial [23], agriculture [13], and biological [5].

There has been a significant surge in data error detection research [18, 31]. The integration of large amount of data always induced data errors due to different data resources, different levels of data quality and inconsistency in data attributes [39]. Our work is inspired by the work of [4] which classify data issues into four categories: Outlier, Duplicates, Rule Violations, and Pattern Violations. One closely related technique addressing the data error within dataset is detect duplication. In order to detecting and addressing data duplication issue, Kolcz et al. [14] consider the consequences of duplicate presence of data and Balaji et al. [6] address the issue by combining data duplication and then using degree sorting to eliminate the overheads of optimization and improves the efficiency of data duplication. Master data management is introduced in [19] to reduce data redundancy in an organization. It is important to note that the goal to detect asymmetrical distribution within the dataset using method such as computer-intensive [36] and approximate numerical [47]. On the other hand, the existing methods are helping with outliers detection. For example, Jiang et al. [22] proposed a two-phase clustering algorithm by firstly modifying the k-mean algorithm and then construct the minimum spanning tree and remove the longest edge. Moreover, Cateni et al. [12] present an outlier detection method based on a fuzzy inference system. There has been a long time research problem on how to dealing with missing

values in data analysis and data visualization. Previous work by Scheffer [35] has compared eight different methods of imputation with different amounts of missing data, but their focus is on how mean and standard deviation are affected by different imputation methods. Some common, simple methods for missing value imputation such as using mean, median, or k-nearest neighbor are discussed in [24]. Other approaches are introduced such as principal component analysis [16, 21] and the use of penalized splines of the propensity score [28].

3 Design Goals

As we mentioned before, the system should not assume an expert user. Thus, the overall system should have a clear layout, an easy-to-use pipeline, and necessary explanations for components/results. Based on these requirements we have for the system, we conclude the following four design goals:

(G1): A collection of implementable and commonly-seen data issues.

As we mentioned before, we don't assume the user to have any specific domain knowledge (though if any, they can use their knowledge to better interact with the recommended information). Thus, we seek for commonly-seen, not too hard to detect and address, and potentially influential in the quality of the generated visualization. If we can help the user to address such a set of basic issues, they can be more focused on either the visualization itself or domain-specific data quality checking.

(G2): A function which can accept datasets from the user side and present them using Vega-Lite specifications.

This is a key feature, which, however, is missing in *StatCheck1.0*. At the end of the day, we would like this toolkit to be beneficial to users' daily routine, meaning that users can test any datasets they would encounter. Although it is generally hard to support a huge dataset space, allowing the user to try certain kinds of datasets definitely can enhance the usability of the toolkit.

(G3): An interface that is in a clear layout, within which visual components are separated appropriately.

We require the overall interface to be easy-to-understand and clear: users should be able to comprehend how to navigate through the system in a short learning period; We also require the components within the interface to be consistent: at the same time, each component should be presenting something meaningful in terms of the same dataset. We don't want to confuse or mislead the user, which is the case we usually observe when trying *StatCheck1.0*.

(G4): An interactive and educative visual recommendation component containing necessary explanations and possible solutions for detected data issues.

In addition to presenting the detected data issues and a recommended visualization (as what it is in *StatCheck1.0*), we argue that more interactions should be added in the recommendation component to (1) allow the user to adapt their domain knowledge (if any) to accept or ignore some detected issues, and (2) educate the user about how the detection process is and potential solutions one can choose from. We want to make sure that we are making suggestions over possible issues the dataset has, and ensure the user the control over which suggestions to take.

In summary, our design goals focus on the validness of the data issues we provide, the organized and consistent components that can work together properly, the ability of the user to test their own datasets and interact with the detected results to better understand and visualize their datasets.

4 Architecture

In this section, we give a high-level overview of the different components of our system. We follow the client-server architecture from *StatCheck1.0*, with a front-end who processes the user input, passes it to the back-end and then displays the information returned accordingly, as well as a back-end who runs the statistical anomaly

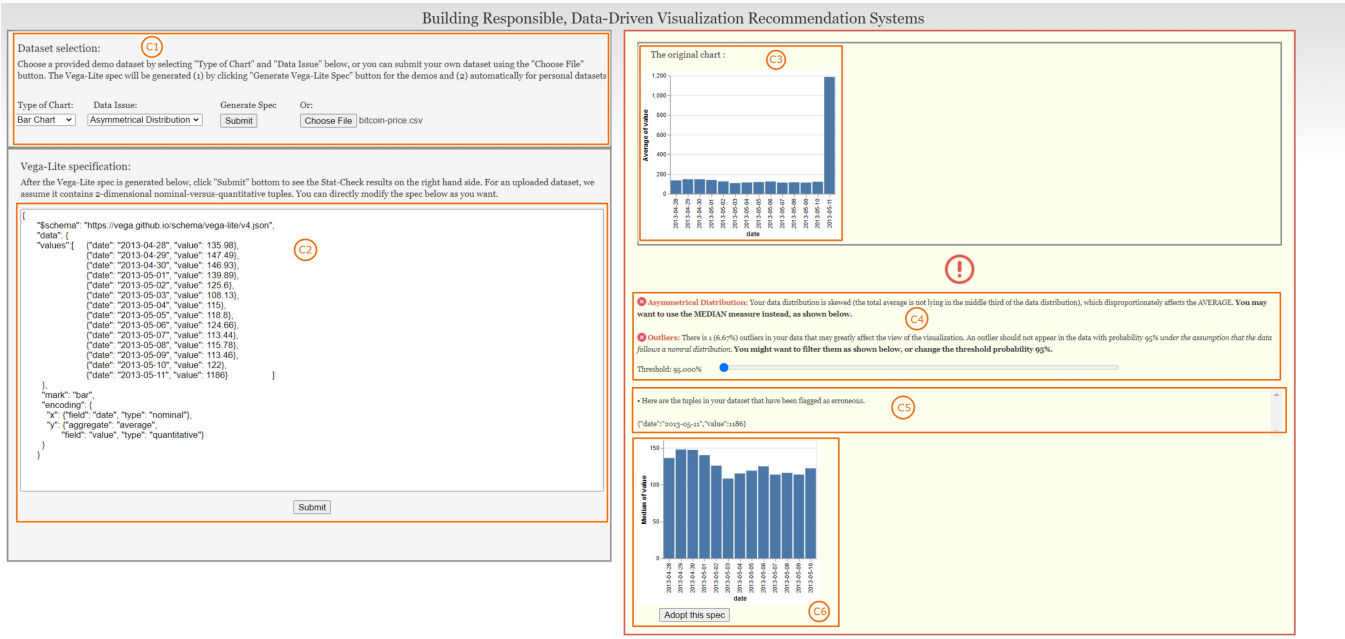


Figure 1: The overall front-end structure of our system

detection functions on the given dataset filters erroneous data and passes all the necessary results back to the front-end. We present the overall front-end structure of our system in Figure 1 and introduce the functionality of each highlighted component. We will detail the back-end in the next section.

There are six major components in the front-end, as shown in Figure 1:

(**C1**): This component receives the user inputs specifying they are testing either a demo dataset or a customized dataset. For demos, the user can select the chart type and the targeted data issue to display; for a customized dataset the system would automatically generate the corresponding Vega-lite specification and run data issue checking.

(**C2**): This component displays the corresponding Vega-lite specification to the user input over the dataset. Notice that the user can directly edit the code inside to meet any personal need.

(**C3**): This component presents the generated visualization without any dataset modification.

(**C4**): This component presents detected data issues if any as well as necessary explanations and potential solutions. For certain issues, the system has interactive widgets that allow the user to test different solutions.

(**C5**): This component displays all the data points related to any detected data issues. If there are too many, the system will provide a slider on the right.

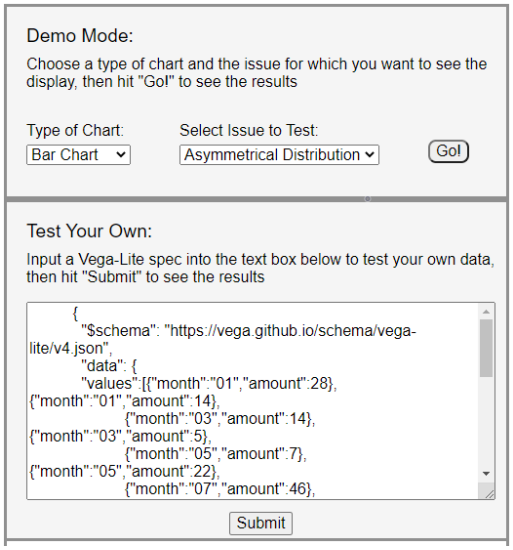
(**C6**): This component displays the recommended visualization after the solutions to detected data issues is applied. The user can accept this chart by clicking the ”Accept this spec” button below if they are satisfied.

5 METHODS

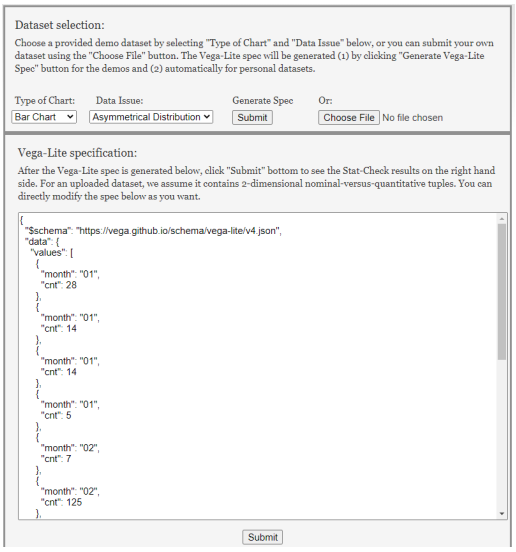
5.1 Improved Design

Based on ***G2*** and ***G3***, we notice that the previous iteration of this prototype provides an unintuitive, complicated interface. The old system has two modes, called “Demo Mode” and “Test Your Own” modes, however, this design is very confusing. There is a Vega-Lite specification created for the “Test Your Own” mode, and the “Demo Mode” provides preset specifications for the issues, however this creates a scenario where there are two different specifications and it is not clear which one is displayed. Also considering that “Demo Mode” fetches the specification from the system and does not show it to the user, it is not easy to use these ‘modes’ for the users. In fact, our team was confused by this design as the “users” of the system when we were gaining familiarity with the project. In addition, we observe that the old system does not enable users to quickly import their external datasets, which makes the data checking process inconvenient. Because of these reasons, we implement an alternative design. Our rationale for the design is that the components of the interface should be consistent and should not lead users to misleading conclusions, and the users should always be able to see the specification of the visualization and iteratively interact with it so that when our system displays an issue, they can easily match the issue with the specification. Also, we expect that users will have a dataset file, such as a “.csv” file, instead of a Vega-Lite specification, therefore importing an external dataset and automatically creating the Vega-Lite specification is essential. Otherwise, the functionalities of the system would not be very beneficial to the users. The comparison of the data and Vega-Lite specification panels for the different versions of *StatsCheck* is given in Figure 2.

We first remove the “Demo” and “Test Your Own” modes, and merge the functionalities into one. We use the same preset specifications from the previous version for various types of charts and data



(a) Data and Vega-Lite specification panel of the previous prototype.



(b) Our design of data and Vega-Lite specification panel.

Figure 2: Comparison of data and Vega-Lite specification panels.

issues and also add new preset specifications for our implemented issues. The users can analyze the provided demo datasets by generating a specification with the desired chart and issue, as seen in ***C1***. When the dataset is fetched from the system, the corresponding Vega-Lite specification is generated in ***C2*** and the system starts detecting data issues, if there are any. Accordingly, ***C3***, ***C4***, ***C5***, and ***C6*** are updated depending on the output of the issue detection system.

The demo datasets are provided so that the users can gain familiarity with the program, however, in order to produce a meaningful results, we need to process the users’ custom datasets. The users can submit their datasets either in a tabular format containing comma-separated values as the data records, or as a Vega-Lite specification file. If they submit a tabular formatted data, we automatically generate the Vega-Lite specification, and display it in ***C2***. The system then proceeds to check the issues in the same way as before. Moreover, the users can interactively edit the specification code in ***C2***, so that they can directly see the effect of a modification to the dataset, even

Figure 3: Error message displayed to the users.

if the modification to the visualization are for exploratory purposes.

Apart from the aforementioned issues, there is another problem of the previous iteration. If there are errors in the format of the Vega-Lite specification in ***C2***, which can easily and regularly happen, the old version of the system is completely unresponsive to the user. The program internally throws an error, however, the user is not notified of this error at all. Also considering that the system keeps the output of the previous visualization, this creates a scenario where the output and the specification are not consistent again. A non-expert user is not likely to realize this, and will continue to use the system and accept the output, which is misleading. We alleviate this problem by removing the previously generated output if there are errors and by displaying an error message to the user instead. An example of the error message is given in Figure 3. For the user side, taking an action based on the error message might require some technical knowledge, since our message includes terms such as "JSON" and "SyntaxError", however, even if the user is not able to fix the error, we notify the user that something went wrong and we do not display a potentially misleading output.

With our improvements, our design effectively achieves ***G2*** and ***G3***, as all components are always in synchrony and consistent, users can utilize the system using their dataset, and there are no misleading outputs in the presence of errors.

5.2 Duplicate Entry Check

As we mentioned, one of our goals, ***G1***, is to detect commonly seen data issues. The previous iteration of *StatsCheck* deals with a set of issues, however, we notice that there is another candidate statistical issue that we can solve, which is the detection of duplicate entries. We add this to the set of data issues the system is checking to cover more potential issues, and raise a warning if there are multiple entries with the same independent variable and they are not properly handled, e.g., by aggregation.

When we detect duplicate values, we first display the entries that are flagged as erroneous and offer two main options to resolve, aggregating the values (default option) or omitting them from the dataset. Within the aggregation option, we also have minor options to choose an aggregation function as well, which include average, maximum, median, minimum, and sum. Since there might be various reasons behind the duplicate entries in the dataset, instead of assuming a 'best practice', we make our system flexible with these multiple resolution options. In the end, it should be easy for users to find the most suitable approach, as we allow them to preview the resolutions one by one and provide visualization for each. An example of this functionality using the aggregate option with minimum as the function is given in Figure 4.

5.3 Improved Outlier and Asymmetrical Distribution Detection

To better achieve ***G1***, we further improved the outlier detection and asymmetrical distribution detection of the previous iteration of *StatsCheck*. The original outlier detection used a hard-coded $1.5 \times IQR$ rule to determine outliers, which we consider inflexible. Instead, our new outlier detection method based on the assumption that quantitative columns of the data follow a Gaussian distribution. Note that the user can choose to omit the issue if she does not think that the data follows a Gaussian distribution, as we will see in Section 5.5. Formally, the process goes as follows: Assume that we have n data points. First, we compute the mean $\vec{\mu}$ and the covariance matrix Σ of the dataset. For notational simplicity we assume that Σ is diagonal. Note that for the general case, we can unitarily diagonalize Σ as $\Sigma = U^{\dagger}DU$ and then apply the linear transformation U to data points. This way the problem is reduced to the case of diagonal Σ. Then, suppose there is a suspicious data point $\vec{x_0}$ and we want to calculate how unlikely it appears. To do that, let us assume that we have a dataset that consists of n samples drawn from $\vec{x} \sim \mathcal{N}(\vec{\mu}, \Sigma)$ i.i.d. The probability of us *not* spotting a data point "as suspicious as $\vec{x_0}$" will be

$$p := \left(\Pr_{\vec{x} \sim \mathcal{N}(\vec{\mu}, \Sigma)} [|D^{-1/2}\vec{x}| < |D^{-1/2}\vec{x_0}|] \right)^n .$$

Therefore $1 - p$ correctly measures how certain $\vec{x_0}$ is an outlier. By default, we label all $\vec{x_0}$ with associated probability higher than 0.95

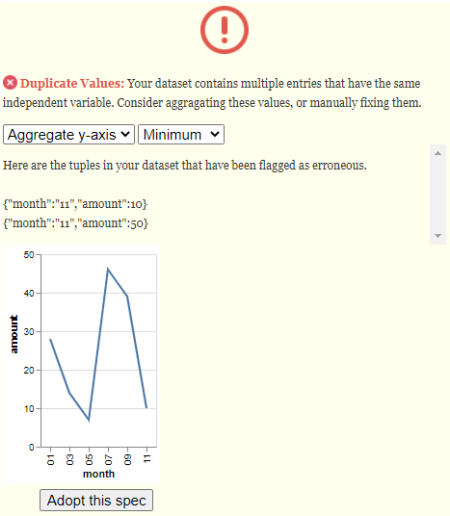


Figure 4: Duplicate entry check.

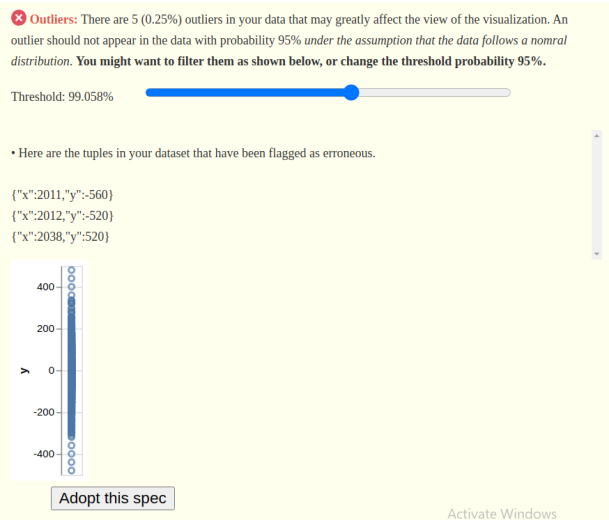


Figure 5: Outlier detection.

(i.e., $p < 0.05$). The user, however, is possible to choose a different threshold value she likes, as shown in Figure 5.

We also updated the asymmetrical distribution check. The basic idea is that the system will recommend to use aggregation method "median" rather than "averge" if the data distribution is highly biased. The original method does not make much sense: they simply check, using the 68 rule, that whether the data distribution is Gaussian. However, being Gaussian or not is irrelevant to the bias. Instead, our approach is that the data distribution will be considered biased if the average does not lie in the middle third of the data distribution, i.e., between 33% and 67% quantiles. An example is shown in Figure 6.

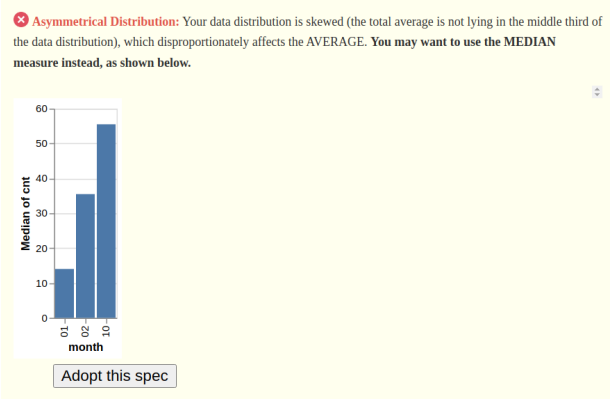


Figure 6: Asymmetric distribution check.

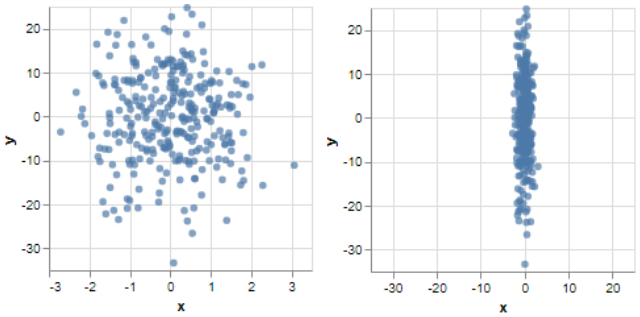
5.4 Axis-aligned Spread Check

We also introduce an additional, minor issue checking process which is more closely related to the Vega-Lite framework than the dataset. However, this process is still based on the dataset statistics and can be beneficial for the users. Therefore, we relate this functionality to our ***G1*** goal, as it has the aim of increasing the quality of the generated visualization. When we look at the spread of the distribution for each attribute in order to detect if there is an asymmetrical distribution, we can only explain the spread of the data parallel to the axes of the attributes. However, if we consider a 2D feature space, e.g. in x and y dimensions, it is likely that the distribution will have a diagonal correlation that cannot be explained by $\sigma(x)$ and $\sigma(y)$. Therefore we also need to perform checks based on the covariance of the data to specifically address the distribution in the 2D feature space.

By utilizing the covariance matrix, we can infer information about the shape of the distribution. Here, we assume that the data comes from a multivariate normal distribution. Based on the covariance matrix, we can capture the diagonal and axis-aligned spread. We use this information to overcome one weakness of Vega-Lite.

In Vega-Lite, when we plot a 2D data, the system creates default ranges for each axis separately. However, if there is a significant difference between the axis-aligned spread for each axis, this can result in having a false sense of the shape of the distribution. An example plot demonstrating this behavior is given in Figure 7a. In the plot, it seems like the data has a 'circular' shape, but this is misleading because y values have an approximate range of $[-30, 20]$ whereas x values have an approximate range of $[-3, 3]$. Therefore the data actually has a 'vertical' shape, as seen in Figure 7b. A non-expert user might not realize this issue, and proceed to draw conclusions by "visually" analyzing the data distribution while ignoring the discrepancies in the feature domains caused by the different magnitudes of spread. Because of this reason, our system warns the users when this problem is detected.

When a user enters a data containing 2 quantitative attributes, assuming that it has a multivariate normal distribution, we check



(a) Example plot demonstrating the auto-matically generated ranges created by Vega-Lite for each axis. Note the differ-ence between the range of x and y-axes.

(b) Example output showing that the sug-gested ranges display the correct shape of the distribution.

Figure 7: Displaying the correct shape of the data based on the axis-aligned spread.

each of the axis-aligned spread. Knowing that the Vega-Lite "zooms in" each axis separately, we calculate the ratio of larger spread to the smaller one and if the ratio is greater than a threshold value (defaulted to 2), our system suggests custom ranges to the user to display the correct shape of the data. The suggested ranges are the same, the range is the largest span that covers each data point. Therefore, this method is quite simple yet effective. An example output that uses the suggested ranges is given in Figure 7b.

5.5 User Interactions

In the previous iteration of *StatsCheck*, the process is just that the user submits a Vega-Lite specification and she will then see a static recommendation. This is barely interactive. To achieve ***G4***, we added the following ingredients:

Issue Explanation. For each issue we have detected, an explanation is shown, which is supposed to be understandable to non-experts.

Multiple Solution. For almost each type of issue, the user has freedom to choose a recommendation from multiple options. Two examples are the slider in Figure 5 and the selection boxes in Figure 4. Or, the user can choose to *omit* certain issues by clicking on the small red cross symbol to the left of a detected issue type shown in ***C4***.

Specification Adoption. The user can choose to adopt the specification recommended to ***C2***. Sometimes this is helpful for iterative issue checking as we *cannot* do that by default: For example, if there is a detected outlier, removing it will result in an update of the mean and the covariance matrix (refer to Section 5.3), which may result in new outliers detected. However, we cannot remove the initial outlier until the user explicitly confirms.

6 Evaluation

This section details the small-scale study we conducted in order to evaluate our prototype system. We hope to gauge how useful our system is for finding potential statistical issues in visualizations and, further, that the system is effective at interactively resolving these potential issues.

The design of this study was inspired by a study done on *StatCheck1.0*. Many of the design choices we made were inspired by the feedback received from the previous evaluation. The hope is that, by conducting a similar study and comparing the results, we will have shown that the tool has been improved and received better than its previous iteration.

All surveys were done remotely via *Zoom*. The participant would be accompanied by one of the group members, who had the tool running on their machine. This would be shared with the participant via a screen share; in order to interact with the tool, the participant would tell us what interactions they wanted to perform, and we would do it for them.

Before the survey began, each participant was given the same introduction on how the tool worked. The participant would learn how to load the example datasets, how the tool provides feedback, and how the user could resolve each of the issues interactively.

Each participant would then go through five sample datasets. These correspond to the four main issues we check for (asymmetrical distribution, outliers, missing values, and duplicate entries), and a fifth dataset with no issues. Once the participant had generated the Vega-Lite specification for a dataset via the built-in functionality, they would be asked to interact with the tool until they felt that the updated plot had been resolved of all issues. This resolution could be done by either using one of the suggested resolution methods or by dismissing the issues that were raised. Note that the task ended when the user felt the visualization was absolved of all issues, not when the tool thought the visualization was absolved of all issues.

For each example dataset, the user answered three questions, all of which were provided on a Google form:

1. What potential problems were flagged, if any?
2. Did you find the feedback provided useful?
3. If an issue was detected, were you able to find a resolution that you felt solved the issue?

Question 1 was asked as a means of quality control. We wanted to ensure that the user understood what the tool was communicating. Should a user answer this question incorrectly, the data would have been removed from our results. Participants entered their responses into a text field for this question.

The results of Questions 2 were meant to gauge how well-received the feedback was. We wanted to see if the tool was effective at flagging issues that are both informative to the user and things that should be remedied. Question 2 was answered on a 5 point Likert scale, with 1 denoting definitely not useful and 5 denoting definitely useful. If no issue was raised, this question was omitted. Both Questions 1 and 2 were similar to the study conducted on *StatCheck1.0*.

Finally, Question 3 is concerned with how well the user is able to interact with our tool. Compared to *StatCheck1.0*, we wanted to make the system more interactive and flexible for the user. Thus, we wanted to collect data on how well the system is able to resolve the potential issues it raises from the user's perspective. The previous study asked a similar question: "Would you have selected the updated visualization?" We note that the resolution process now involves a back-and-forth exchange between the user and the system, and we updated this question to reflect this change. Question 3 was recorded as a yes or no if an issue was detected, and was omitted otherwise.

Upon completion of the above three questions for all five sample datasets, we asked follow-up questions to gauge the user's overall experience with the tool and the study itself. In particular, we asked the following:

- How easy were the questions in this survey to answer? This was collected on a 5 point Likert scale, with 1 denoting very hard and 5 denoting very easy.
- Rate your agreement with the following statement: "The tool would be useful to design better visualizations." This was also collected on a 5 point Likert scale, with 1 denoting strong disagreement and 5 denoting strong agreement.
- What additional features, if any, do you wish the system included? Users entered their responses into a text box.
- Are there any other comments or suggestions you might have? Users entered their responses into a text box.

The first question was asked to ensure that the survey and its corresponding tasks were straightforward and well-defined, while the second question was asked to gauge the overall user satisfaction with the system. The last two questions were added to allow for open ended feedback and critique from the user, should they feel the need to express their thoughts outside the scope of the survey.

We neglected to collect demographical information on the participants of our study. This was mainly due to the small scale of the study. Informally, the participants were all young adults who were both technologically savvy and data savvy. The race and gender identifications varied across participants. We note that the target audience of our tool also includes less data-inclined users, and the study may not accurately reflect this group.

In total, we amassed four participants for our study. While certainly not a large enough sample to make generalizations about the system's performance, it does provide some initial insight. Further, we note that the previous study of *StatCheck1.0* also included four participants, so it is possible to compare the results between studies without extrapolating.

Finally, the participants were encouraged to ask clarifying questions and voice their thoughts during the study. While these were not formally recorded, they may be referenced to provide context to the results we present.

7 Results

This section presents the results of the evaluation study we performed, described in detail in Section 6. We provide the results from the perspective of three success metrics. To begin with, we look at the users' overall sentiment by analyzing the questions asked at the end of the survey. Next, we connect the results of working through the example datasets to the design goals listed in Section 3 to determine if we have met these goals. Finally, we compare our results to the study performed on *StatCheck1.0* to demonstrate the tool has been improved.

7.1 Analyzing Overall User Experience

We begin by first analyzing the questions asked at the end of the survey. These questions were asked to gauge the overall experience the users had with our tool and the study itself. This will give us a high-level view of the sentiment toward our tool before delving deeper into the specific examples.

First, we note that, when asked how easy the questions in the survey were to answer, all four participants responded with "Very Easy", i.e. a 5 on the Likert scale. This indicates that the survey was indeed well-defined and straightforward for the users to complete and that we should have no qualms with accepting this data as accurate.

The other required question was to rate agreement with the statement "the tool would be useful to design better visualizations". The results are presented in Figure 8. With an average of 4, we can see that the users generally felt that our system helps with designing visualizations. It is also worth noting that the user who submitted a 3 viewed the system more as a data-refining tool than a visualization tool.

Finally, the users were able to leave comments about other features they would like to see incorporated into the system. Two users both mentioned that they wish there was a way to export the results that had been freed of the issues we raised. Indeed, this would be a useful feature, but fell beyond the scope of our project; our goals were more focused with getting data into the system rather than exporting it out. We leave such a feature to future work.

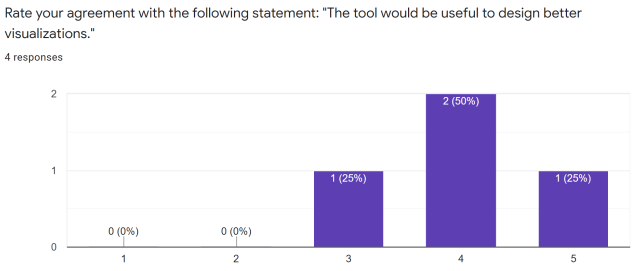


Figure 8: Results of Agreement, with 5 denoting strong agreement

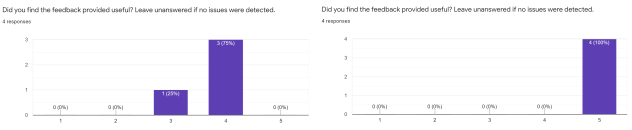
We conclude that the overall sentiment towards our system was generally positive and that most of the negative sentiments were caused by a lack of of features rather than the system itself.

7.2 Examining Example Dataset Survey Responses

Of the three questions participants answered for each sample dataset, we begin by examining the first question. They were asked to confirm the issues that the tool raised for the dataset, and they entered them into a text box. Each user was correctly able to reiterate what issues were being raised, if any. We submit this as evidence for **G3**, as the users were able to load the pre-built examples and identify the issues raised without much confusion.

The other question where users seemed unanimous was the third question, where we asked if they were able to resolve all of the issues raised by the system. Across all datasets, all four participants responded that they were able to resolve the issues with each visualization. Note that the user could resolve issues by accepting or rejecting the proposed solutions; we view this as an achievement of the interactive nature of our system and accept this as evidence towards completing **G4**.

For the second question, we collected information on how useful each participant found the information displayed for each dataset. When looking at the results, we see an emerging dichotomy of the data issues. The first group of issues consists of outliers and asymmetrical distribution, which we label as "statistical issues". The second group contains duplicate entries and missing values, henceforth "integrity issues". Each feature in the two groups had the same distribution of responses, so we present a graph for each of the two groups in Figure 9.



(a) How useful each participant found the check for outliers and asymmetrical distribution, with 5 denoting very useful.

(b) How useful each participant found the check for missing values and duplicate entries, with 5 denoting very useful.

Figure 9: A side-by-side comparison of the reception of statistical issues versus integrity issues.

In particular, we see that integrity issues are very well received, with every participant finding this warning very useful. Conversely, while the statistical issues are still useful, they are significantly less useful than the integrity issues, with the average dropping from a 5 to 3.75. While not formally recorded, we did observe that users were more likely to reject the suggestions for the statistical issues than the integrity issues, which may explain this discrepancy.

Given that all four issues are positively received, we consider **G1** to be a success. While certainly not an exhaustive list, we have incorporated several issues that users want to know about and resolve. We make note of the more lukewarm reception to the statistical issues and conjecture that this is due to the users' inclination to reject our recommendations rather than accept them.

7.3 Comparison to Previous Study

Finally, we can compare the results of our study to the study conducted on *StatCheck1.0*. The studies were similar in many ways; both studies amassed four participants and guided the users through datasets containing issues the system would check for. As such, it is fair to make direct comparisons between the two studies.

Perhaps unsurprisingly, we see similar patterns emerge from both studies. The treatment of missing values was well received by both sets of participants, while asymmetrical distribution was more poorly (but still positively) received. Further, most of the information provided by the system was deemed useful, with only one instance in the previous study of a user finding the information returned not useful.

The one key difference between the two studies lies in how the users went about resolving the issues the system raised. In *StatCheck1.0*, there was no interaction during the resolution phase of issues; users were asked how likely they would be to adopt the system's recommendations on a 5 point Likert scale. Conversely, our version of StatCheck allows the user to interactively solve issues or ignore them entirely; we instead asked if the user had found a way to resolve all of the issues with the plot.

As such, we saw a marked improvement in this area. The previous study made note of at least four instances where the users indicated they may or may not accept the recommendations or would definitely not accept the recommendations. In our study, every single instance was able to be solved in the opinions of our users. We chalk this up as another success towards **G4**, as we have quantifiably shown that the interactions we have added allow the users to create new visualizations that consider correct.

We conclude with some notes about **G2**. Our study did not examine how well the system is able to accept user-made datasets as it instead opted to work with pre-made datasets that highlight the issues we check for. However, we did receive a comment about this feature as we introduced the tool. One user expressed confusion over the chart type selection option as it pertained to uploaded datasets. This selection only applies to the example datasets; to change the type chart for an uploaded dataset, one would have to directly edit the Vega-Lite specification. We admit this is a failure of both **G2** and **G3**, as the layout is confusing when uploading datasets and we do not have a simple way for users to change the chart types for their uploaded data.

8 Discussion and Future Work

In this project, we extend *StatCheck1.0*, a previous prototype, to make a more powerful and interactive visualization recommendation system that automatically helps the user check potential data issues within their datasets. We test *StatCheck1.0* and fundamentally improve the system mainly through (1) cleaning up and reorganizing the front end, (2) supporting user-specified datasets, and (3) allowing more interactions between the user and the system. We present a study in which 4 participants tested the system with five datasets. We use the resulting data to validate the effectiveness and usability of our improved system. In the following subsections we talk about the major takeaways, limitations and future directions.

8.1 Takeaways

Based on what we have learned along the way, we provide the following takeaways that one could get from our project.

1. **Injecting data issue checking into visualization tools will benefit many users.**

As we explained before, nowadays visualization tools tend to be pretty easy-to-use even for users without sufficient knowledge about programming. Users can easily generate visualizations in an end-to-end manner (from data to charts). However, such systems cannot guarantee the validness of the visualizations from the data perspective, meaning that they cannot help fix any data issues that might exist within the users' datasets. It is usually left up to the user to have some requisite training to finish necessary data cleaning, checking, and analyzing jobs so as to ensure the quality of the data, which cannot be much expected since novice users who may not have data processing experiences are also encouraged to try those visualization systems. It is best to have some data checking tools built inside the visualization systems that can help resolve the above problem.

On the play-test-day, we got a comment from one audience saying "this tool looks cool, could it be implemented in Jupyter Notebook?", which indicates that performing data issue checking is indeed a useful feature for broad users while is currently missing in most of the programming languages or visualization tools. The results of our user study also show that all participants are mostly positive toward the usefulness of our data-issue-checking-based recommendation system.

2. **Interaction is easy, educational interaction is hard.**

   One of our major contributions is the design and implementation of several additional interactions between the user and the detected results. The motivation behind this is that we want to "educate" the user to better understand what is going on inside the checking procedure through interacting with the results provided by the system. This might also help the user understand their dataset better. However, how to achieve clear educational guidance without elaborating much with statistics is found to be hard. For example, our outlier checking is based on the assumption of normal distribution, and a slider controlling the filter threshold is provided (please see Section 5.5 for details). We observe in the user study that several participants didn't get the functionality of the slider at the beginning. Although we provide some textual explanations above the slider, it is always the visual object, a.k.a., the slider, that attracts the user's attention first. We also find it hard to achieve a balance between space efficiency and understandable statistical explanations.

3. **We should always pay attention to any bias concern when designing visualization systems.**

   When brainstorming over possible extensions of the data issues, we came up with an idea that recommending to the user attribute combinations that have higher correlations than the user specifies if more attributes are available. At that time, we thought this could be a cool feature that is useful to novice users. However, in the progress meeting, we've been told that this falls into the category of p-hacking, which is a common issue related to bias concerns. Later we took the bias lecture and gained a better understanding of the bias concerns in visualization and more generally data science. On this specific project, we have to keep in mind that we are only making suggestions over the user's data other than directly or indirectly telling the user what is right or wrong; ignoring the bias concerns could result in invalid visualizations especially for novice users who just start in this field. This is an important lesson we learned no only for this project, but also for any systems we will build in the future.

8.2 Limitations

Due to technique difficulty and limited time, we list three major limitations of out system:

1. **Only support two-dimensional datasets.**

   Currently the system only supports data issue checking over two-dimensional datasets, which is implementation-friendly and doable in a limited time period. However, it is definitely needed to support datasets with more columns since the most important goal of this project is to make the tool useful for customized datasets (which varies in many ways like number of columns, types of attributes and number of rows). So without extending to more general datasets, our system is to some extent limited in terms of capacity.

   Obviously, extending to more general datasets is not an easy task. Several things need to be taken care of, such as generalizing the current detection functions to multi-dimensional cases and determining what certain user inputs are needed for complex user-specified datasets. Since this more lies in the statistics field other than visualization, we didn't consider it as a goal in our project.

2. **Optimization over the detection procedure is needed.**

   Currently, there is no back-end optimization existing (all the calculations happen at the Python server), making a noticeable latency for certain kinds of data issue checking, e.g., outlier checking for the example dataset we provided. It is possible that the user uploads a huge dataset (even it is two-dimensional), and observing a latency probably would downgrade the user's experience with our system. How to handle this case efficiently is currently not well-considered, and we leave it as future work.

3. **The user study is not perfect.**

   Our current user study involves 4 participants, which is not ideally large for us to obtain enough insights and further improve the system. Also, due to the inconvenience of conducting evaluations remotely, we didn't include the personal dataset uploading feature as a part of the evaluation, which might cause a loss of suggestions regarding that feature.

8.3 Future Work

Apart from the above limitations that can be considered, we list three potential future directions that can be explored to further improve our system:

1. **Supporting user-specified checking criterion.** In the case where the user wants to check over some domain-knowledge-related issues, we can consider letting the user input the checking criterion in a specific form such that we can integrate that criterion to check the dataset given.

2. **Providing options to export the recommendation visualization into codes using other visualization languages.** This was suggested by one of our participants in the user study, who wanted to turn the resulting Vega-Lite specifications into D3 codes. We agree with this advice and think this could benefit users from various backgrounds.

3. **Making the interface more visually appealing.** Our current implementation for the interface follows *StatCheck1.0* and is a bit old-school. It would be interesting to modernize the interface to make it visually more attractive.

9 Conclusion

In this project, we proposed fundamental improvements over a previous prototype *StatCheck1.0* mainly regarding front-end designs, refined data checking methods and educative user interactions to provide a visualization recommendation system within which an automatic statistical data issue checking feature is implemented. Such

a system helps novice users get started with visualization and more broadly data science, and assists expert users with data-cleaning to save time. We validate the usability and effectiveness of our system either textually using the learnt principles or by our user study. Although the goals of this project is in general fulfilled, we do think there are many ways we can further improve our system to allow better sense of control over both the dataset and the checking criterion from the user side, which makes it a useful system for broad users. Ultimately we would like to see the ideas within our project would be maturely engineered into valid features in modern visualization frameworks.

10 Team Member Contributions

- Suleyman: He contributed to the alternative front-end design with *Chen* by synchronizing the components, and implemented detecting errors in the Vega-Lite specification and notifying the users about them. He added the options as the aggregation functions for the duplicate entry check, and designed and implemented the axis-aligned spread checking. He wrote sections 5.1, 5.2, and 5.4, and came up with and implemented two ideas that did not make into the final version, checking the statistical significance which is abandoned due to p-hacking issue, and syntax highlighting of Vega-Lite specification which is disabled due to the bugs in the underlying framework Prism. He prepared the presentation video with *Ethan*.

- Chen: He led most of the team meetings and the task separations. As for the implementation, he (1) reorganized the visual components into the current shape; (2) added the personal data uploading function & automatic Vega-Lite specification generalization. As for the ideas, he assisted *Yufan* to form the interaction design of the outlier checking. As for the user study, he collected one data point and assisted *Ethan* to get one another. As for the writing, he accounted for the Introduction / Design Goals / Architecture / Discussion and Future Work / Conclusion sections. He together with *Yufan* led the progress meeting.

- Ethan: He wrote the Evaluation and Results sections of the report. In the code, he implemented the checking for duplicate entries, the interactive resolutions for missing and duplicate entries (***C4***), and printing the erroneous tuples (***C5***). He also led most of the playtest demo alongside *Tianshu* and presented most of the presentation video alongside *Suleyman*. He designed the evaluation study and conducted most of the studies, alongside *Chen*.

- Tianshu: He led part of the team meeting, the play test day demo along with *Ethan*, and led on the presentation video slides. He also make contribution to sections in the project report. Moreover, he contribute ideas to proposing on user interface design, and solving data issues including out-of-domain entries, missing time series data, longitude, latitude, zip etc.

- Yufan: He implemented the new outlier and asymmetrical distribution detection (Section 5.3). He realized the "omit" buttons mentioned in Section 5.5. He also contributed minor changes and bug fixes to other components, e.g., data uploading feature and ***C5***. For ideas, he, with assistance of *Chen*, came up with ideas about functionalities just mentioned. He and *Chen* also proposed an interesting idea of *differential influence* of data points, which was finally abandoned on the progress meeting because this notion is visualization-based but not statistics-based. He together with *Chen* led the progress meeting. He was responsible for writing of Section 5.3 and 5.5.

References

[1] The pyplot package. `https://matplotlib.org/stable/tutorials/introductory/pyplot.html`. Accessed: 2021-05-16.

[2] Tableau. `https://www.tableau.com/`. Accessed: 2021-05-16.

[3] Vega: a visualization grammar, Apr. 2021. original-date: 2013-02-03T18:36:30Z.

[4] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment*, 9(12):993–1004, 2016.

[5] J. Aerts, N. Gehlenborg, G. E. Marai, and K. K. Nieselt. Visualization of biological data-crossroads (dagstuhl seminar 18161). In *Dagstuhl Reports*, vol. 8. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[6] V. Balaji and B. Lucia. Combining data duplication and graph reordering to accelerate parallel graph processing. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 133–144, 2019.

[7] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, p. 1363–1375. Association for Computing Machinery, New York, NY, USA, 2016. doi: 10.1145/2882903.2882919

[8] N. Bikakis. Big data visualization tools, 2018.

[9] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[10] S. Bresciani and M. J. Eppler. The pitfalls of visual representations: A review and classification of common errors made while designing and interpreting visualizations. *Sage Open*, 5(4):2158244015611451, 2015.

[11] E. T. Brown, A. Ottley, H. Zhao, Q. Lin, R. Souvenir, A. Endert, and R. Chang. Finding waldo: Learning about users from their interactions. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1663–1672, 2014. doi: 10.1109/TVCG.2014.2346575

[12] S. Cateni, V. Colla, and M. Vannucci. A fuzzy logic-based method for outliers detection. In *Artificial Intelligence and Applications*, pp. 605–610, 2007.

[13] W. Chen, P. Liu, C. Zhang, M. Yan, R. Zhao, and B. Li. Development and research of agricultural big data visualization system.

[14] A. Chowdhury and J. Alspector. Data duplication: an imbalance problem? In *ICML'2003 Workshop on Learning from Imbalanced Data Sets (II), Washington, DC*, 2003.

[15] P. R. Doshi, E. A. Rundensteiner, and M. O. Ward. Prefetching for visual data exploration. In *Proceedings of the Eighth International Conference on Database Systems for Advanced Applications*, DASFAA '03, p. 195. IEEE Computer Society, USA, 2003.

[16] S. Dray and J. Josse. Principal component analysis with missing values: a comparative survey of methods. *Plant Ecology*, 216(5):657–667, 2015.

[17] D. Gotz and Z. Wen. Behavior-driven visualization recommendation. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*, IUI '09, p. 315–324. Association for Computing Machinery, New York, NY, USA, 2009. doi: 10.1145/1502650.1502695

[18] T. Gschwandtner, J. Gärtner, W. Aigner, and S. Miksch. A taxonomy of dirty time-oriented data. In *International Conference on Availability, Reliability, and Security*, pp. 58–72. Springer, 2012.

[19] F. Haneem, R. Ali, N. Kama, and S. Basri. Resolving data duplication, inaccuracy and inconsistency issues using master data management. In *2017 International Conference on Research and Innovation in Information Systems (ICRIIS)*, pp. 1–6. IEEE, 2017.

[20] J. Heer and M. Bostock. Crowdsourcing graphical perception: Using mechanical turk to assess visualization design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, p. 203–212. Association for Computing Machinery, New York, NY, USA, 2010. doi: 10.1145/1753326.1753357

[21] A. Ilin and T. Raiko. Practical approaches to principal component analysis in the presence of missing values. *The Journal of Machine Learning Research*, 11:1957–2000, 2010.

[22] M.-F. Jiang, S.-S. Tseng, and C.-M. Su. Two-phase clustering process for outliers detection. *Pattern recognition letters*, 22(6-7):691–700, 2001.

[23] M. Jimichi, D. Miyamoto, C. Saka, and S. Nagata. Visualization and statistical modeling of financial big data: double-log modeling with skew-symmetric error distributions. *Japanese Journal of Statistics and Data Science*, 1(2):347–371, 2018.

[24] J. Kaiser. Dealing with missing values in data. *Journal of systems integration*, 5(1):42–51, 2014.

[25] D. A. Keim, F. Mansmann, J. Schneidewind, and H. Ziegler. Challenges in visual data analysis. In *Tenth International Conference on Information Visualisation (IV'06)*, pp. 9–16. IEEE, 2006.

[26] I. Ko and H. Chang. Interactive data visualization based on conventional statistical findings for antihypertensive prescriptions using national health insurance claims data. *International Journal of Medical Informatics*, 116:1–8, 2018. doi: 10.1016/j.ijmedinf.2018.05.003

[27] J. Kohlhammer, D. Keim, M. Pohl, G. Santucci, and G. Andrienko. Solving problems with visual analytics. *Procedia Computer Science*, 7:117–120, 2011.

[28] R. Little and H. An. Robust likelihood-based analysis of multivariate data with missing values. *Statistica Sinica*, pp. 949–968, 2004.

[29] S. Liu, G. Andrienko, Y. Wu, N. Cao, L. Jiang, C. Shi, Y.-S. Wang, and S. Hong. Steering data quality with visual analytics: The complexity challenge. *Visual Informatics*, 2(4):191–197, 2018.

[30] Y. Luo, X. Qin, N. Tang, and G. Li. Deepeye: Towards automatic data visualization. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pp. 101–112, 2018. doi: 10.1109/ICDE.2018.00019

[31] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[32] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, vol. 1, pp. 381–390, 2001.

[33] S. Salloum, J. Z. Huang, and Y. He. Exploring and cleaning big data with random sample data blocks. *Journal of Big Data*, 6(1):1–28, 2019.

[34] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization & Computer Graphics (Proc. InfoVis)*, 2017. doi: 10.1109/tvcg.2016.2599030

[35] J. Scheffer. Dealing with missing data. 2002.

[36] C. D. Sutton. Computer-intensive methods for tests about the mean of an asymmetrical distribution. *Journal of the American Statistical Association*, 88(423):802–810, 1993.

[37] J. Van den Broeck, S. A. Cunningham, R. Eeckels, and K. Herbst. Data cleaning: detecting, diagnosing, and editing data abnormalities. *PLoS Med*, 2(10):e267, 2005.

[38] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: Efficient data-driven visualization recommendations to support visual analytics. *Proc. VLDB Endow.*, 8(13):2182–2193, Sept. 2015. doi: 10.14778/2831360.2831371

[39] L. Visengeriyeva and Z. Abedjan. Metadata-driven error detection. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, pp. 1–12, 2018.

[40] J. Walny, C. Frisson, M. West, D. Kosminsky, S. Knudsen, S. Carpendale, and W. Willett. Data changes everything: Challenges and opportunities in data visualization design handoff. *CoRR*, abs/1908.00192, 2019.

[41] R. Wan, R. Garnett, and A. Ottley. Learning and anticipating future actions during exploratory data analysis. *CoRR*, abs/1809.09664, 2018.

[42] K.-C. Wang, K. Lu, T.-H. Wei, N. Shareef, and H.-W. Shen. Statistical visualization and analysis of large data using a value-based spatial distribution. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 161–170, 2017. doi: 10.1109/PACIFICVIS.2017.8031590

[43] M. L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021. doi: 10.21105/joss.03021

[44] H. Wickham and M. H. Wickham. The ggplot package, 2007.

[45] L. Wilkinson. The grammar of graphics. In *Handbook of computational statistics*, pp. 375–414. Springer, 2012.

[46] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, 2016. doi: 10.1109/TVCG.2015.2467191

[47] Y. Yokomizu, Y. Kito, and T. Matsumura. Approximate numerical method for deriving an asymmetrical distribution of radiation intensity in a gas blasted arc. gas fukitsuke arc ni okeru hitaishona hosha kyodo bunpu no kinji kyukai shuho. *Denki Gakkai Ronbunshi, B (Transactions of the Institute of Electrical Engineers of Japan);(Japan)*, 110(7), 1990.