A VISUALIZATION TECHNIQUE FOR COURSE EVALUATIONS AND OTHER LIKERT SCALE DATA

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Muhammed Saho

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Statistics

May 2018

Fargo, North Dakota

North Dakota State University
Graduate School

Title

A VISUALIZATION TECHNIQUE FOR COURSE EVALUATIONS AND OTHER LIKERT SCALE DATA

By

Muhammed Saho

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Rhonda Magel
Chair

Curt Doetkott

Saeed Salem

Approved:

May 7, 2018
Date

Rhonda Magel
Department Chair

ABSTRACT

Course evaluation is one of the primary ways of collecting feedback from students at NDSU. Since almost every student in every course submits one at the end of the semester, it generates a lot of data. The data is summarized into text based reports with emphasis on average rating of each question. At one page per course, analyzing these reports can be overwhelming. Furthermore, it is very difficult to identify patterns in the text reports.

We combine heat maps and small multiples to introduce a visualization of the data that allows for easier comparison between courses, departments, etc. We defined a data format for storing and transmitting the data. We built an interactive web application that consumes the aforementioned data format and generates the visualizations. We simulated reference data to facilitate interpretation of the visualizations. Finally, we discussed how our research can be applied more generally to Likert scale data.

ACKNOWLEDGEMENTS

I would like to thank the members of my committee: Dr. Rhonda Magel, Curt Doetkott, and Dr. Saeed Salem for their continued support and guidance.

A special thanks is extended to Curt Doetkott for seeding the original idea and generously lending me his time.

PREFACE

A version of this paper was originally attempted by Treasure Sims in 2007. She passed away before she could complete it. May her soul rest in perfect peace.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDIX FIGURES

CHAPTER 1. INTRODUCTION

Student rating of instruction (SROI) or course evaluation as it is more commonly known, is one of the most common forms of gathering feedback from students at NDSU. Towards the end of every course, students fill out a questionnaire about the instructor and the course in general. The responses are collected and coded into a university database, from which various reports are generated. Administrators use the generated reports to inform certain decisions. The reports are also published on the university website for interested parties to review.

The current course evaluation questionnaire at NDSU was introduced in the 2013-2014 academic year and it consists of 16 questions. Prior to the 2013-2014 academic year, only the first 6 questions were on the questionnaire. The additional 10 questions were added to alleviate some concerns of gender bias in the original questions. Each question is answered on a Likert-type scale with five discrete choices: “Very Poor”, “Poor”, “In Between”, “Good”, and “Very Good”. The choices are abbreviated as: VP, P, IB, G, and VG respectively. In datasets, they are coded as: 1, 2, 3, 4, and 5 respectively. The questions are as follows:

1. Your satisfaction with the instruction in this course.
2. The instructor as a teacher.
3. The ability of the instructor to communicate effectively.
4. The quality of this course.
5. The fairness of procedures for grading this course.
6. Your understanding of the course content.
7. Instructor created an atmosphere conducive to learning.
8. Instructor provided well defined course objectives.
9. Instructor provided content/materials clear/organized.

10. I understand how grades were assigned in this course.

11. I met/exceeded the course objectives for this course.

12. Instructor was available to assist students outside class.

13. Instructor provided feedback in a timely manner.

14. Instructor provided relevant feedback that helped me learn.

15. Instructor set and maintained high standards for students.

16. Physical environment was conducive to learning.

Text based reports are generated from the collected data. An interesting feature of the reports is that they rely heavily on mean and standard deviation. The use of mean and standard deviation on this type of data is a subject of great debate. Jamieson (2004) argued that the distance between the various levels (i.e. VP, P, N, G, VG) cannot be presumed equal — therefore interval/ratio descriptive statistics like mean and standard deviation are not applicable. Our visualizations do not depend on interval descriptive statistics. Instead, we rely on categorical statistics like frequency and percentage.

Another subject of great debate is the effectiveness of course evaluations themselves. While there are plenty of studies that examine the merits of student course evaluations, this paper focuses solely on the visualization of the data. We do not attempt to make any statements about the practice itself. The technique we are proposing is generally applicable to most Likert scale data. We use the course evaluation data mostly to demonstrate a practical use case of our method.

Our visualization derives its properties from two main ideas — small multiples popularized by Edward Tufte (1983), and heat maps commonly used in gene expression data (Eisen et al, 1998). Tufte described small multiples as: "*Illustrations of postage-stamp size are* *indexed by category or a label, sequenced over time like the frames of a movie, or ordered by a quantitative variable not used in the single image itself.*" A heat map is a 2D matrix of values represented as colors. Combining these two ideas, we present a visualization that is both simple and information dense.

Our research consisted of three main components. First, we defined a data format that allowed us to efficiently generate the visualizations. Using SAS, we extracted the SROI data and exported it to our predefined data format. Next, we built an interactive web application that ingests the data and outputs the visualizations. We built it using standard web technologies: HTML, CSS, and JavaScript. The web application also serves as a playground for experimenting with different forms of the visualization. It allows us to dynamically adjust features of the visualization like size and color. Finally, we developed SAS code that allowed us to specify probabilities of the different rating levels and simulate data for any number of question-course combination. Using this data, we produced visualizations to serve as references in the interpretation of real data. We conclude our paper by discussing how our method can be applied more generally to other Likert scale data.

CHAPTER 2. LITERATURE REVIEW

In this chapter, we will review some of the historical literature associated with our paper. First, we will focus on the history of Likert scales. More specifically, we will review Likert scales used in student evaluations. Next, we will do a brief review of Tufte's (1983) small multiples. As part of the literature on visualizations, we will also do a brief review of heat maps. Finally, we will tie it all together in a literature review of student evaluations.

Likert Scales

A Likert scale is a type of psychometric rating scale, named after its inventor, Rensis Likert (1932). It is a popular form of collecting responses in questionnaires. It is a psychometric scale, meaning it is designed to measure attitudes or opinions (Bowling, 1997). In student evaluations, there are usually 5 categories of response per question. For example, 1 = strongly disagree, 2 = disagree, 3 = neither agree nor disagree, 4 = agree, and 5 = strongly agree. Some instances of Likert scales have 4, 7, or even 9 categories of response. Technically, Likert scale refers to the format in which a collection of responses are scored. An individual response is a Likert item (Likert, 1932). However, a Likert item is frequently wrongly referred to as a Likert scale.

What differentiates a Likert item from other rating systems is its symmetric nature. Likert items usually have a neutral response, and an equal number of positive and negative responses. There are instances of the scale that do not have a neutral response, forcing a respondent to make a positive or negative choice. A common assumption is that the distance between every successive response is equal (Norman, 2010). However, some have argued that this assumption is invalid because the responses capture intensity of human feeling (Jamieson 2004). These two opposing arguments are the most relevant thing about Likert scales to our paper. Whether Likert scales constitute interval or ordinal data determines which types of analysis can be performed on them.

Another frequent subject in Likert scales are the biases of respondents. Respondents may answer questions incorrectly to put themselves in a more positive light. They may misrepresent for other reasons such as to avoid being perceived as having extremist views, portray their organization in a positive light, etc. These problems are less related to our paper, but they may be relevant when interpreting a set of visualizations. Researchers have found that offering anonymity to respondents improved the accuracy of their answers (Paulhus, 1984). Student evaluations at NDSU do not identify individual students.

There have been papers that examine the visualization of Likert scales. A good example is the Diverging Stacked Bar Chart (Heiberger & Robbins, 2014). Figure 1 shows an example of a diverging stacked bar chart. The peculiar characteristic of this visualization is its adherence to the bipolar nature of Likert items. This makes is good at distinguishing the level of agreement vs disagreement between various questions. The diverging nature of the bars does have one particular drawback — they take up more space. This may not be a concern when analyzing one question across other variables. However, in the case of student evaluations, we are usually trying to compare all the questions across courses, instructors, departments, etc. We tackle this challenge by introducing Tufte's small multiples into our visualization.

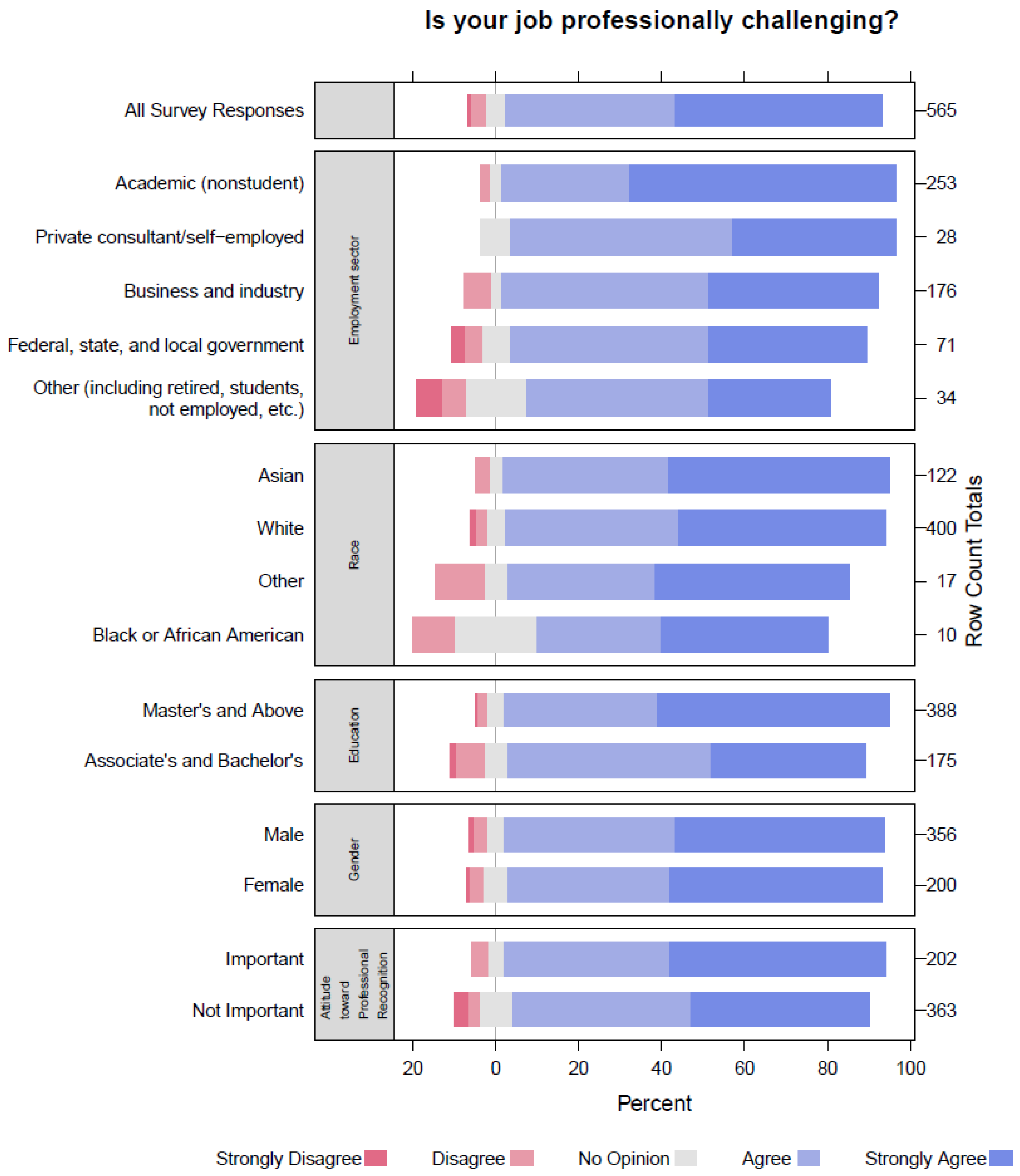


Figure 1. Diverging Stacked Bar. (Heiberger & Robbins, 2014)
Survey responses to a question on job satisfaction (Luo & Keyes 2005). A total of 565 respondents replied to the survey. Each person answered one of five levels of agreement or disagreement with the question “Is your job professionally challenging?” Each panel of the plot shows a breakdown of the respondents into categories defined by the criterion listed in its left strip label.

Small Multiples

Small multiples are a series of small charts with the same axes, usually in a grid to allow them to be easily compared. The concept was popularized by Edward Tufte in his Visual Display of Quantitative Information book (1983). However, this visualization technique has been recorded in much earlier works like Francis Amasa Walker's "Persons with gainful occupations and also as attending school" in the Statistical Atlas of the United States (1874).

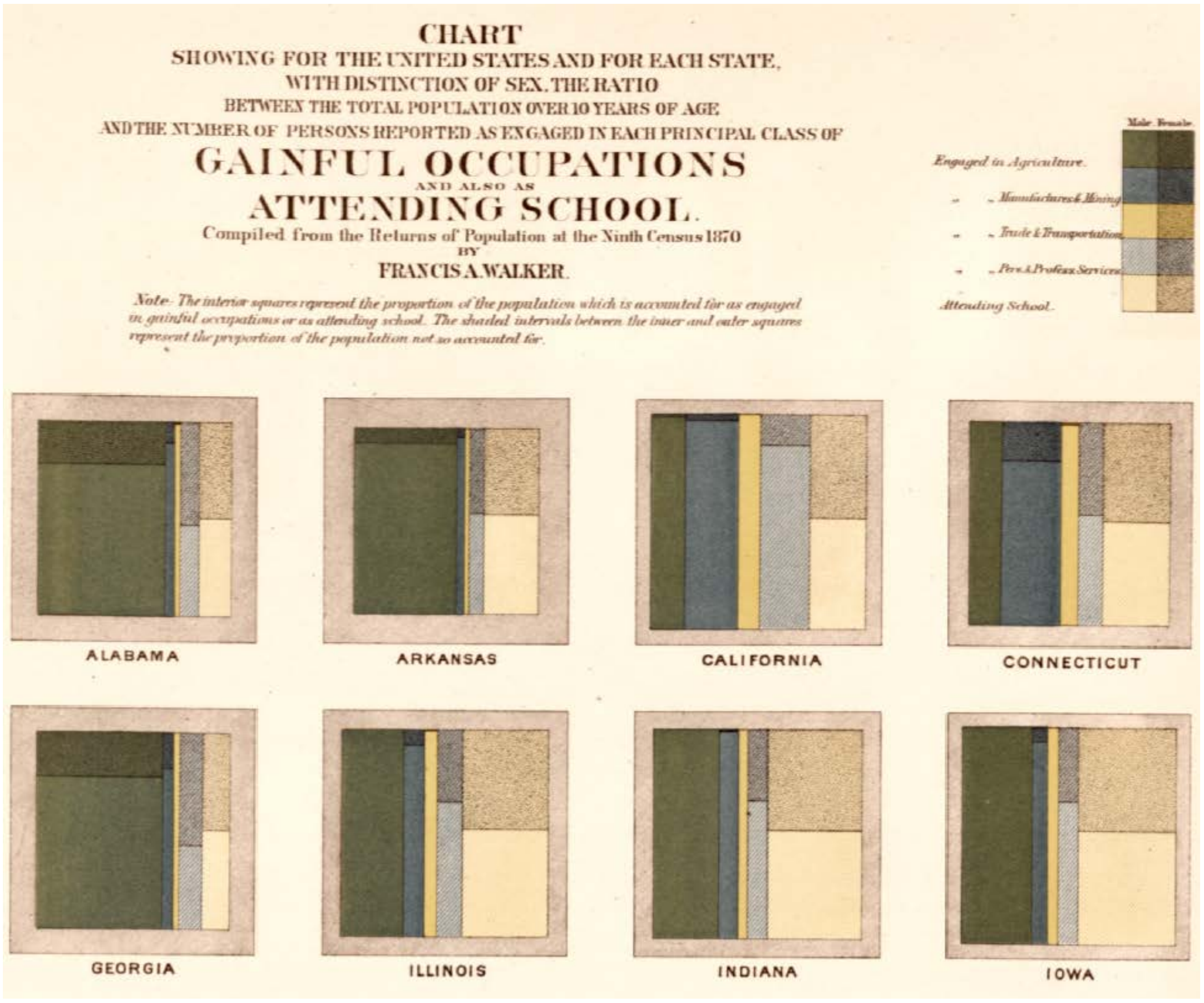


Figure 2. Persons with Gainful Occupations and also as Attending School snippet. (Walker, 1874)

Walker used small multiples to compare the employment levels of males and females from state to state. He also used the same technique to compare church denomination by state. Examples of small multiples have since become more elaborate with the addition of animation and interactivity. An example of this new breed of small multiples is demonstrated in an interactive article by The New York Times ("Kepler's Tally of Planets," 2013) that uses the same concept to display and compare the orbital features of all the planet systems discovered by NASA's Kepler mission.

Heat Maps

A heat map is a 2D matrix of values represented as colors. Similar values are usually clustered near each other. According to Weinstein (2008), it is one of the most widely used graphs in the biological sciences. Wilkinson and Friendly (2009) attribute the lineage of heat maps to two older ideas: shaded matrices and permuting matrices. Wilkinson et al. note, "*Shading a table or matrix is a long-established method of highlighting entries, rows, or columns. Accountants, graphics designers, computer engineers, and others have used this method for years. The most common recent application involves the use of color to shade rows, columns, or cells of a spreadsheet*".

On the other hand, permuting matrices is the practice of ordering matrices to reveal structure. Bertin (1983) demonstrated the usefulness of this method — how it can be used to easily identify relationships between variables. He even built a mechanical device to facilitate the process of reordering matrices.

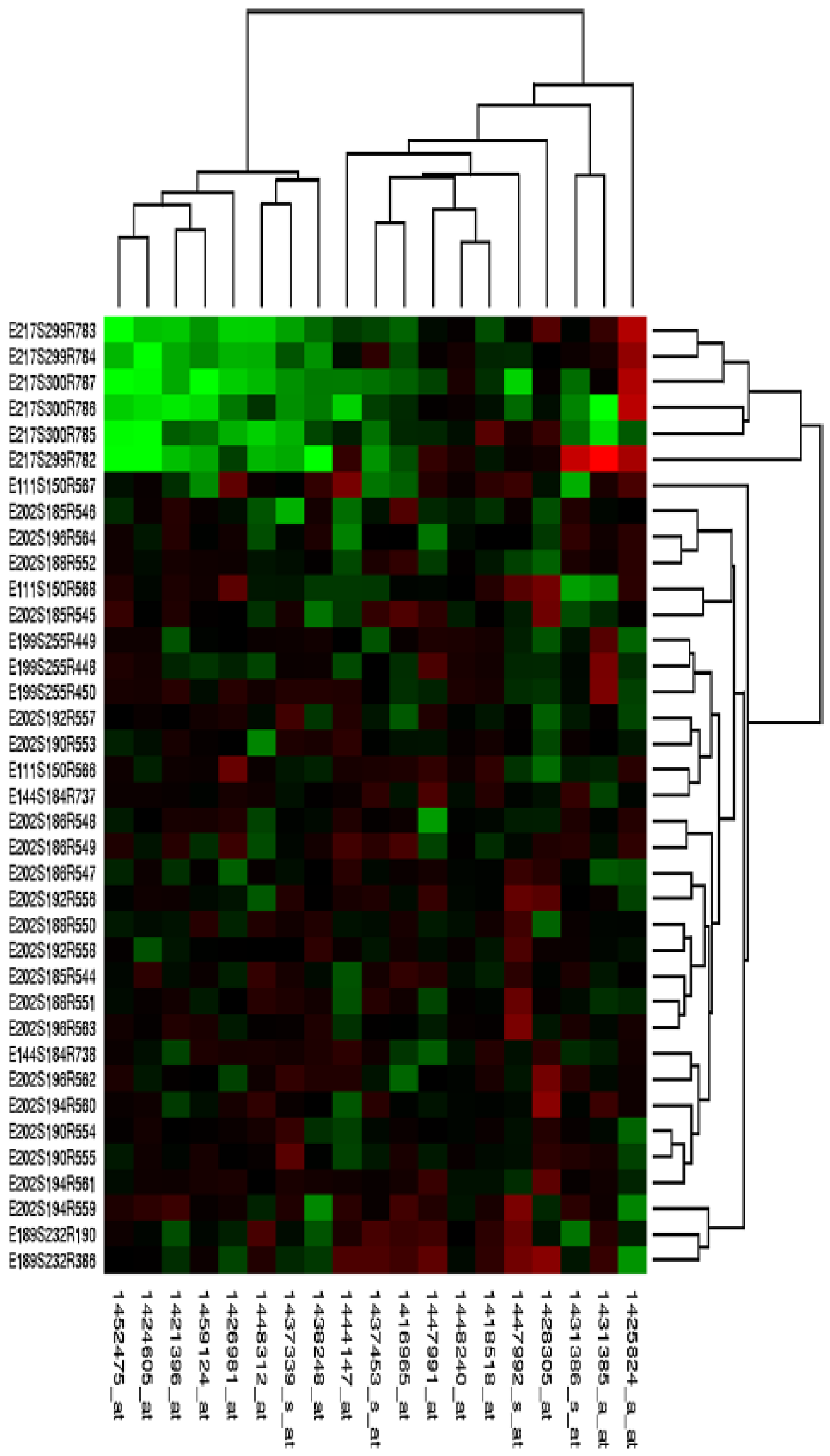


Figure 3. Cluster Heat Map. (Wilkinson & Friendly, 2009)
The rows represent genes, and columns represent samples. Each cell is colorized based on the level of expression of that gene in that sample.

The central feature tying the two ideas is color. Several color schemes are used in heat maps but the most popular is the rainbow color scheme (Borland & Taylor 2007). However, several authors have criticized the rainbow color scheme as being misleading due to its lack of perceptual ordering (Borland & Taylor 2007). This is especially problematic when visualizing data that is ordered if the perceived intensity of different colors do not match the magnitude of the values they represent. We dive deeper into this topic in our color discussion in chapter 4.

Student Evaluations

Student evaluations is one of the most studied forms of personnel evaluation (Marsh 1987). Despite the wide range of materials on this topic, very little focus has been put on the visualization aspect. Instead, a good number of research papers study external factors that affect student ratings (Chang, 1997; Feldman, 1993; Marsh & Roche, 1997; Wachtel, 1998). These papers argue that student evaluations do not represent a fair assessment of the instructors being rated due to external factors. As mentioned earlier, our paper does not examine the merits of student evaluations. Another set of research papers establish guidelines on how to interpret and use the data (Linse, 2017; McKeachie, 1997; Theall & Franklin, 2001). Our research aims to complement these efforts by adding another dimension to the interpretations in the form of visualizations.

CHAPTER 3. METHODOLOGY

In this chapter, we will describe some of the processes we followed to create our visualizations. The following sections describe our data format, various elements of our visualization, the web application, and the data simulation. Since a significant portion of our paper is programming code, we will attach those in the appendix and reference them as needed.

Data

The development of our visualization occurred in two different programming environments. We used SAS to analyze, process, and simulate data. On the other hand, we used web programming (HTML, CSS, JavaScript) to create and interact with the visualizations. Because the data processing and data visualization occurred in two separate environments, we had to come up with a data contract to ensure that the data we generated in SAS could be consistently consumed by our web application.

The first iteration of our data format was a simple Comma-Separated Values (CSV) file. We chose CSV because it is very simple and well supported. SAS natively supports exporting datasets to a CSV file. Several other data processing programs such as Excel, R, and Minitab also support CSV to varying degrees. The structure of a CSV file is documented in the RFC 4180 standard (Shafranovich, 2005). It is a text format typically used for storing tabular data. Each record of a table is represented by a line in the text file. The fields within each record are separated by a comma. In some cases, such as ours, the first line contains the names of the fields corresponding to the rest of the file.

Figure 4 shows an excerpt of course evaluation data in the first iteration of our CSV format. The data is of a 6 question course evaluation questionnaire. Each line captures the ratings of all 6 questions by an individual student. The fields are as follows:

icn - The course number

r1 - The rating for question 1 (1 to 5)

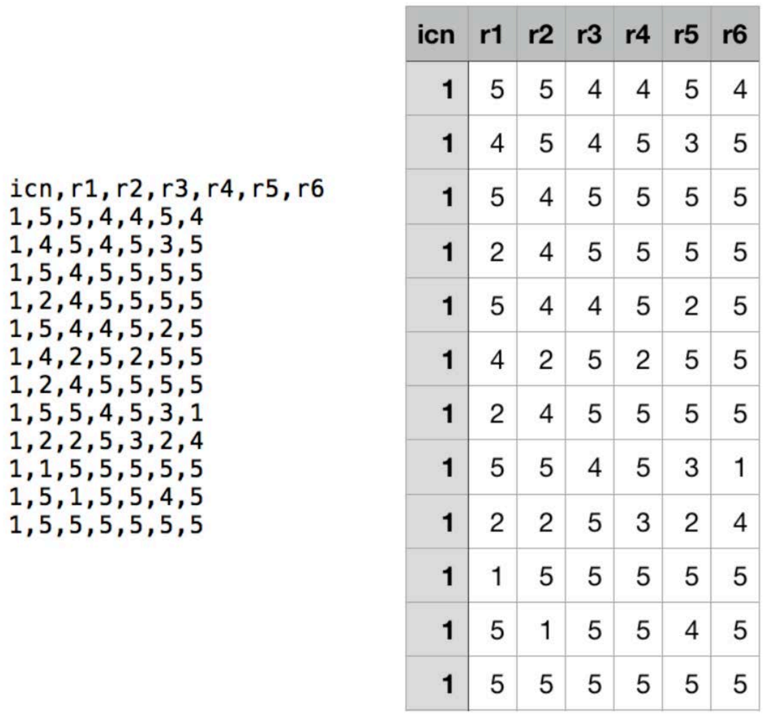
r2 - The rating for question 2 (1 to 5)

r3 - The rating for question 3 (1 to 5)

r4 - The rating for question 4 (1 to 5)

r5 - The rating for question 5 (1 to 5)

r6 - The rating for question 6 (1 to 5)



```
icn,r1,r2,r3,r4,r5,r6
1,5,5,4,4,5,4
1,4,5,4,5,3,5
1,5,4,5,5,5,5
1,2,4,5,5,5,5
1,5,4,4,5,2,5
1,4,2,5,2,5,5
1,2,4,5,5,5,5
1,5,5,4,5,3,1
1,2,2,5,3,2,4
1,1,5,5,5,5,5
1,5,1,5,5,4,5
1,5,5,5,5,5,5
```

icn	r1	r2	r3	r4	r5	r6
1	5	5	4	4	5	4
1	4	5	4	5	3	5
1	5	4	5	5	5	5
1	2	4	5	5	5	5
1	5	4	4	5	2	5
1	4	2	5	2	5	5
1	2	4	5	5	5	5
1	5	5	4	5	3	1
1	2	2	5	3	2	4
1	1	5	5	5	5	5
1	5	1	5	5	4	5
1	5	5	5	5	5	5

Figure 4. Data Format 1 CSV Excerpt and Tabular view

This format worked well during the development of the visualization but its limitations were soon apparent. The main challenge was a way to include other attributes like department and institution while at the same time allowing for a variable number of questions. CSVs are designed such that the number of fields are static. Therefore it would be non-trivial to design a program that generated visualizations from a k question(s) questionnaire while allowing an unspecified number of attributes to also be coded in the data using this format.

The other downside of this format was that it generated very large text files. Each line represents a student in a course; therefore, a year of NDSU course evaluation data coded in this format resulted in a text file with tens of thousands of lines. Generating visualizations from a file of this magnitude is not optimal for a web application that is designed to be fast and interactive. These two issues prompted us to revise the data format.

Although each record in the CSV represents a student, there is no information identifying the particular student nor do we wish to identify individual students. We can greatly reduce the amount of data without losing any information by summarizing it to the number of students that rated each question at each rating level instead of capturing the ratings of individual students. Transforming the data in this way results in the following fields:

icn - The course number

question - The question number

count1 - Count of students that gave this course and question a rating of 1 (VP)

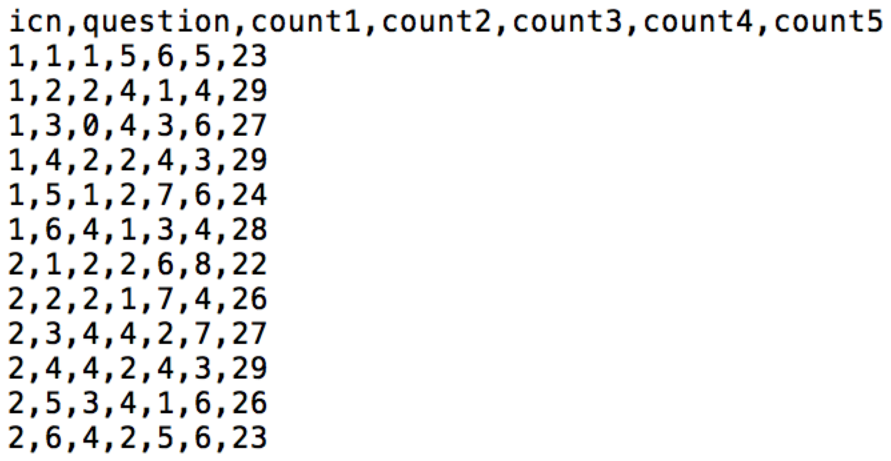
count2 - Count of students that gave this course and question a rating of 2 (P)

count3 - Count of students that gave this course and question a rating of 3 (IB)

count4 - Count of students that gave this course and question a rating of 4 (G)

count5 - Count of students that gave this course and question a rating of 5 (VG)

Incidentally, this transformation also resolves our variable question issue. The question number is now encoded in each record, therefore we can have as many questions as we want without adjusting the number of fields. Also, we know that our data is going to be a 5-point Likert scale, therefore we can add extra fields like “department” and “institution” immediately following the fields for the 5 counts. Our web application can treat all fields after count5 as extra attributes that can be used for grouping and filtering. Finally, we ensure that the data is sorted by course number then by question number so that the web application can consume the data more efficiently. Figure 5 shows an excerpt of the final data format.



```
icn,question,count1,count2,count3,count4,count5
1,1,1,5,6,5,23
1,2,2,4,1,4,29
1,3,0,4,3,6,27
1,4,2,2,4,3,29
1,5,1,2,7,6,24
1,6,4,1,3,4,28
2,1,2,2,6,8,22
2,2,2,1,7,4,26
2,3,4,4,2,7,27
2,4,4,2,4,3,29
2,5,3,4,1,6,26
2,6,4,2,5,6,23
```

Figure 5. Data Format 2 CSV Excerpt

Visualization

The primary goal of the visualization is to make it easy to compare several groups. Examples of groups include comparing courses in a department, comparing different departments in a college, or comparing colleges in the university. Figure 6 shows a comparison of 4 courses, on a 6 question course evaluation using our visualization. In this case, the grouping factor is a course (A,B,C,D). Each group (course) is represented by a rectangular heat map we refer to as a “stamp”. A stamp consists of 6 equal sized columns representing the 6 questions.

We call each vertical column a “bar”. A bar is further divided into “blocks”. A block is color coded to its corresponding rating. The height of a block represents the relative count of a rating on a question.

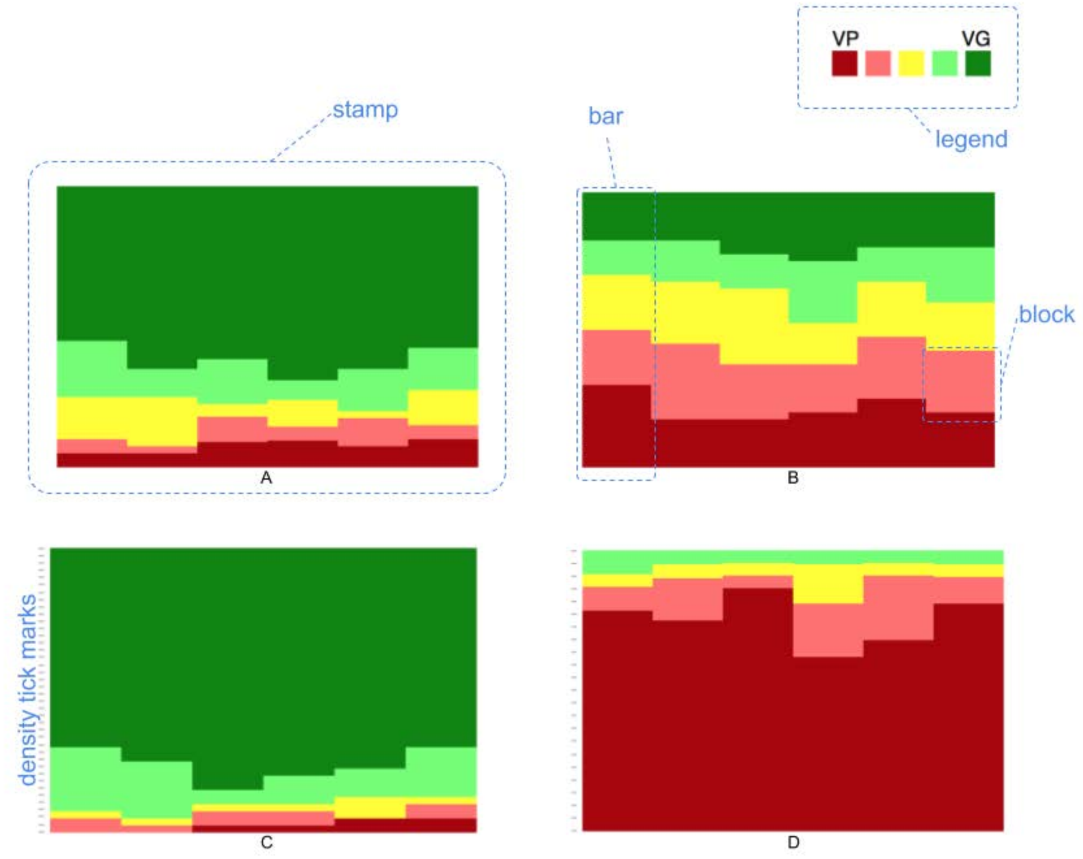


Figure 6. Annotated Diagram of Four Stamps

The sizing of the different elements plays an important role in our visualization. We strived to maintain visual consistency as much as possible. All the stamps on a visualization have the same size. Furthermore, the bars within a stamp are also the same size. The only elements with different sizing are the blocks within a bar. For example, the “very good” blocks in stamp C are much taller than the rest signifying majority positive ratings.

Keeping the major elements equally sized has two distinct advantages. First, it improves space efficiency. We can shrink the overall visualization without worrying about one particular element getting too small. The web application does allow a user to dynamically adjust the overall height and width of the bars. Secondly, equal sizing of stamps allows a user to quickly scan several stamps and apply information learned from one stamp onto others with less effort. These design cues are borrowed directly from small multiples.

Sizing blocks equally does have a drawback; you lose a sense of magnitude when comparing groups. Two groups with vastly different numbers of responses may look similar because their relative counts are the same. To combat this issue, we introduced tick marks adjacent to the leftmost bar of each stamp to signify how many individuals are in that group. Each tick mark represents one individual. Groups with higher numbers of respondents have a higher density of tick marks. For example, the tick marks in stamp D are farther apart than those in stamp C. Stamp D has 20 individuals hence 20 tick marks, whereas stamp C has 40. In the web application, these tick marks can be toggled on/off as they may become more distracting than useful in cases where the groups are roughly the same.

Color also plays a major role in our visualization. We experimented with several color schemes before settling on this subset of the rainbow color scheme. Although papers like Borland and Taylor (2007) warn against using a rainbow color scheme because of its lack of perceptual ordering, it worked better in our case due to the bipolar nature of Likert items. The notion of representing positive as green, negative as red, and neutral as yellow is universally understood. Traffic lights and numerous other everyday objects use the same color scheme. However, we did incorporate some level of perceptual ordering within each pole. The green color representing "very good" is darker than that of "good". Likewise, the red color representing "very poor" is darker than that of "poor". The web application does also have the option of toggling to a monochrome color scheme. This is useful for persons with color blindness and grayscale printing.

Web Application

The web application generates visualizations in two main stages. When a user uploads a CSV data file to the application, the first stage is automatically triggered. In this stage we use the "Papa Parse" CSV library to parse the data file and collect the records. The records from the file are stored in the internal state of the application which triggers the second stage. In the second stage, the application loops through the records in its internal state and renders the blocks sequentially according to the settings specified in the options bar.

The options bar provides the user with controls for adjusting various characteristics of the visualization. When an option is adjusted by the user, the second stage is triggered again and the blocks are re-rendered. The following options are currently available in the web application:

Monochrome - Toggles the color scheme between rainbow and monochrome.

Labels - Toggles the group labels on or off.

Ticks - Toggles the tick marks on or off.

Bar Width - Adjusts the bar width in pixels.

Bar Height - Adjusts the bar height in pixels.

Choose File - Uploads a data file.

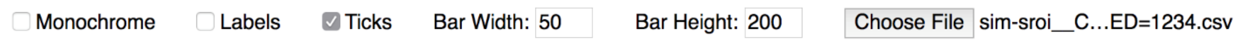

☐ Monochrome ☐ Labels ☑ Ticks Bar Width: 50 Bar Height: 200 Choose File sim-sroi__C…ED=1234.csv

Figure 7. Options Bar.

The rendering of the web application is component based and uses the “React” library. Each element (eg. blocks, bars, stamps, options) is implemented as a self-contained component with parameters. The various components are composed together to form the application interface. The code for the various components of the application is attached in appendix C. The code repository for the entire application is also available at https://goo.gl/ziuodF and a live version of the application is hosted at https://goo.gl/tmj9PW .

Simulation

An important aspect of any visualization is the ease of interpretation. To aid in the interpretation of ours, we programmed a simulation to generate course evaluation data. We then used the data to create reference visualizations. Our simulation is coded in SAS. The core of it uses the “Table” distribution in the RAND function. Wicklin (2011) notes: “*The table distribution enables you to specify the sampling probabilities that are associated with each of k events, and to draw randomly from those events*”. We use this function to simulate a multinomial distribution where the categories correspond to the rating levels of a course evaluation. Our code is designed such that we can adjust the following variables for every simulation:

CLASS_SIZE - The number of students in a course.

NO_OF_CLASSES - The number of courses.

NO_OF_QUESTIONS - The number of questions per course.

VG - The probability of a “very good” rating.

G - The probability of a “good” rating.

N - The probability of an “in between” rating.

P - The probability of a “poor” rating.

VP - The probability of a “very poor” rating.

SEED - The random generator starting seed.

OUTPUT_FILE - The output file.

The result of the simulation is output to a file in the format discussed earlier. We generated several visualizations by identifying patterns in the real data and simulating references for them. For each reference, we simulated 12 6-question courses with 50 respondents per course. We have attached the reference visualizations in appendix B, and we discuss them in chapter 4. The code for the simulation is attached in appendix A.

CHAPTER 4. RESULTS

In this chapter, we will highlight various characteristics of our visualization and how those characteristics can be used in analyzing the course evaluation data. We do not intend to make any conclusions about the data; instead we want to showcase observations that may not be as apparent in a text report. We will also discuss the 3 major visual patterns we identified from the course evaluation data. Finally, we will discuss how adjusting various attributes of our visualization affects its usability.

Although our visualization focused on comparing groups, we found that it was also good at highlighting oddities between questions in a group. Figure 8 shows the visualization of 3 courses, A, B, and C. In courses A and B, the bars for question 5 ("The fairness of procedures for grading this course") are visibly different from the rest. In course A, the ratings are roughly equally distributed in all questions except question 5 where the rating is overwhelmingly "good". Conversely, question 5 is rated worse than the other questions in course B. Course C shows the same phenomenon on a different question, number 4. Overwhelmingly, this phenomenon was observed in question 5 throughout the course evaluation data we visualized. Perhaps students care more about their grades than other facets of their courses. We did not study the reason behind this behavior in the data.

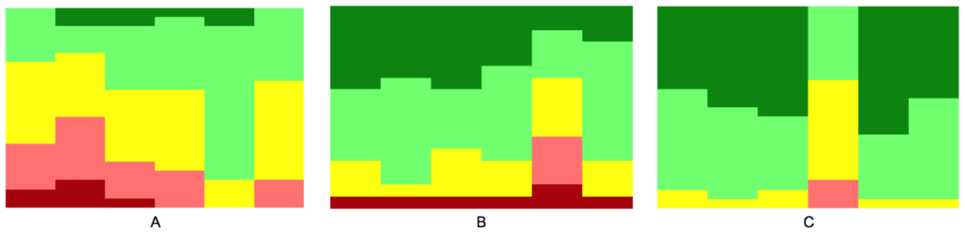


Figure 8. Question Highlight
From real anonymized course evaluation data.

Reference Visualizations

Browsing the visualizations for the course evaluation data, we identified 3 major patterns henceforth referred to as positive, neutral, and negative. We used the simulation described in chapter 3 to simulate the 3 patterns. Each simulated group consists of 50 simulated respondents and 6 questions. It is important to note that the patterns described below may not apply to other datasets. When analyzing new datasets, one should identify patterns of interest for that dataset, and use those for reference.

Figure 9 is an excerpt of a simulation with ratings 50% VG, 45% G, 4% IB, 1% P, and 0% VP. We refer to this pattern as "positive" because it consists mostly of "good" and "very good" ratings. It is the most common rating pattern we found in our data. It is very easy to spot as it consists mostly of greens with an occasional sprinkle of yellow and red. Some groups have no yellow or red.

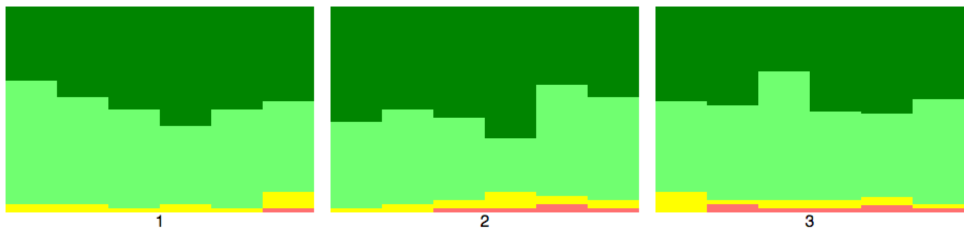


Figure 9. Simulation of Positive Pattern.
Simulated at 50% VG, 45% G, 4% IB, 1% P, and 0% VP.

Another common pattern is one where the ratings are more concentrated in the center. Figure 10 is an excerpt of a simulation with ratings 5% VG, 25% G, 40% IB, 25% P, and 5% VP. We refer to this pattern as "neutral". It manifest as generous blocks of light green, yellow, and light red; sandwiched by a thin strip of dark green at the top and a thin strip of dark red at the bottom.

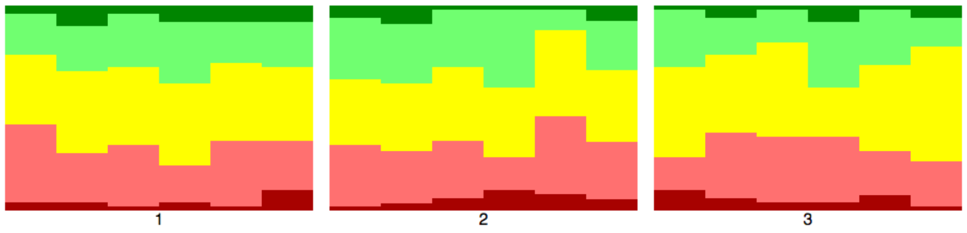


Figure 10. Simulation of Neutral Pattern.
Simulated at 5% VG, 25% G, 40% IB, 25% P, and 5% VP.

A less common pattern we also identified in our data is the "negative" pattern. Figure 11 is an excerpt of a simulation with ratings 5% VG, 5% G, 15% IB, 30% P, and 45% VP. These manifest as patches of green and yellow sitting on big blocks of reds. In the course evaluation data, we found that it was more common to have one or two negatively rated questions than a whole group of questions.

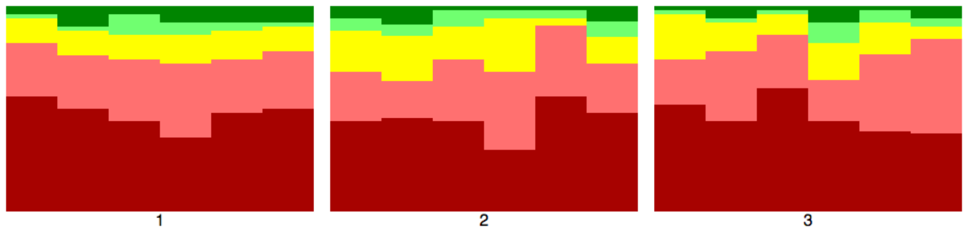


Figure 11. Simulation of Negative Pattern.
Simulated at 5% VG, 5% G, 15% IB, 30% P, and 45% VP.

Attributes

In addition to the rainbow color scheme, we also explored a monochrome color scheme. Figure 12 is a comparison of the two color schemes. The monochrome color scheme maintains a perceptual ordering; "very poor" being the lightest and "very good" the darkest. It is useful in cases where color printing is not available. Persons with color blindness may also find it more useful than the rainbow color scheme. We generally recommend the rainbow color scheme because it is easier to spot the patterns we described earlier.

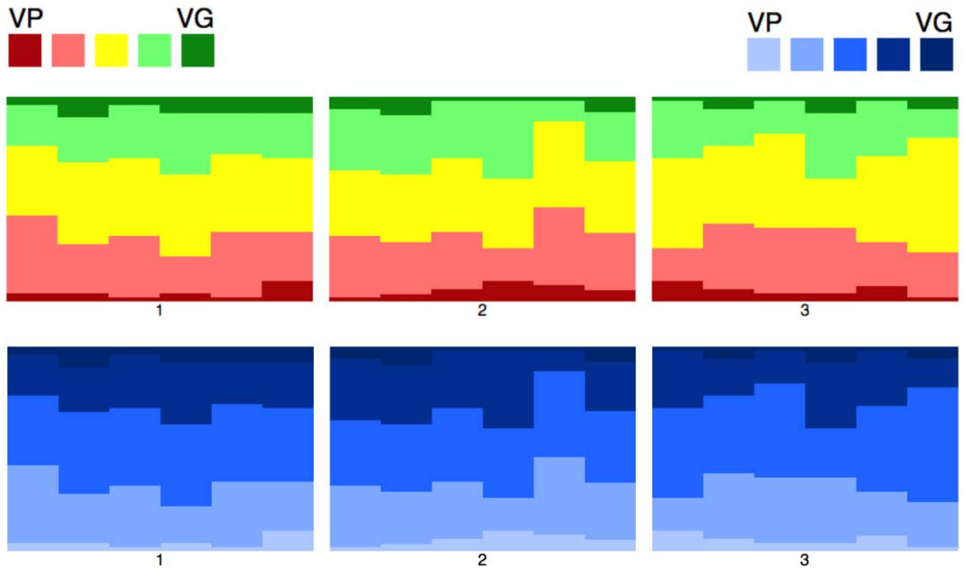


Figure 12. Rainbow vs Monochrome Color Scheme.
Simulated at 5% VG, 25% G, 40% IB, 25% P, and 5% VP.

By default, the web application sizes the bars at 50px width by 200px height. In cases where one is comparing several groups, it may be beneficial to reduce the size of the bars to fit more stamps in the viewport. Figure 13 is a combined simulation of positive, neutral, and negative stamps; shrunk down to a bar size of 10px width by 50px height. At this much smaller size, it is still relatively easy to differentiate the rating patterns.

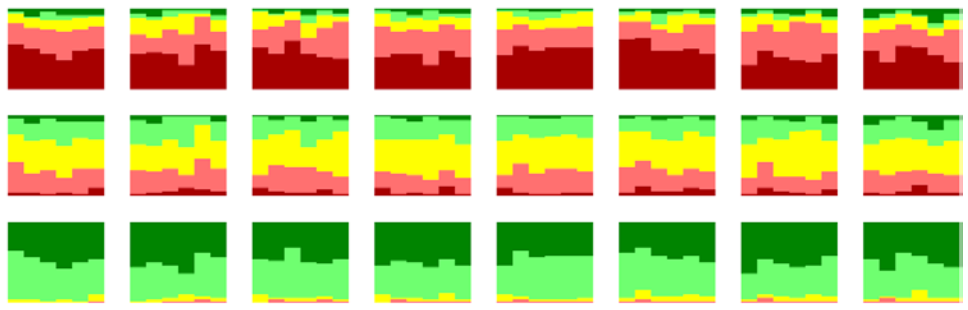


Figure 13. Small Stamps.
Bar size adjusted to 10px Width by 50px Height.

Smaller sizes are also useful in cases where we have a relatively large number of questions. We can fit more stamps in the viewport by shrinking the bar size. Figure 14 is an excerpt of a 16-question simulation with ratings 45% VG, 35% G, 15% IB, 4% P, and 1% VP.

The bar size has been adjusted to 10px width by 100px height.

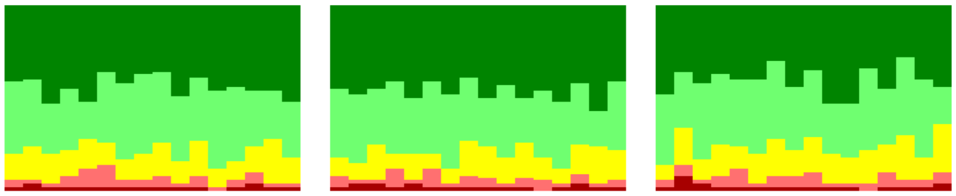


Figure 14. 16-Question Simulation
Simulated at 45% VG, 35% G, 15% IB, 4% P, and 1% VP. Bar size adjusted to 10px width by 100px height.

CHAPTER 5. CONCLUSION

The primary purpose of this paper was to develop a technique for visualizing course evaluation data at NDSU. Additionally, we wanted our technique to be more generally applicable to other Likert data. We accomplished both goals by borrowing ideas from heat maps and small multiples. We implemented our visualization in the form of an interactive web application, and we defined a data format to make it easier to import other data sources into the web application. Finally, we developed code to simulate course evaluation data and export it to our predefined format. We used the simulated data to generate reference visualizations.

Future Considerations

There are some aspects which we would like to improve in the future. The web application can be made more useful by adding some advanced controls to filter and group the data interactively. Currently, if one wants to visualize a subset of their data, they would need to filter the data in a separate program before importing it into the web application. Similarly, if a user wants to group the data by for example "department" instead of "course", they would need to preprocess the data before importing it. In the future, we want to be able to import the data once, then interactively group and filter the data within the web application.

REFERENCES

Bertin, J. (1983). Semiology of Graphics: diagrams, networks, maps. University of Wisconsin Press.

Borland, D., & Taylor, R.M. (2007). Rainbow map (still) considered harmful. IEEE Computer Graphics and Applications. Volume 27. No. 2. pp. 14-17.

Bowling, A. (1997). Data collection methods in quantitative research: questionnaires, interviews and their response rates. Research methods in health: Investigating health and health services, pp. 257-272.

Chang, T. (1997). Student ratings of instruction at teachers college in Taiwan. Hualien, Taiwan: Graduate Institute of Compulsory Education, National Hualien Teachers College. Eric Document Reproduction Service No. ED 414 798.

Eisen, M.B., Spellman, P.T., Brown, P.O., & Botstein, D. (1998). Cluster analysis and display of genome-wide expression patterns. Proceedings of the National Academy of Sciences, Volume 95. No. 25 pp. 14863-14868.

Feldman, K.A. (1993). College students' views of male and female college teachers: Part II—Evidence from students' evaluations of their classroom teachers. Research in Higher Education, Volume 34. pp. 151-211.

Heiberger, R., & Robbins, N. (2014). Design of diverging stacked bar charts for Likert scales and other applications. Journal of Statistical Software, Volume 57. No. 5. pp. 1-32.

Jamieson, S. (2004). Likert scales: how to (ab)use them. Medical education, Volume 38. No 12. pp. 1217-1218.

Kepler’s Tally of Planets. (2013, April 18). Retrieved from http://www.nytimes.com/interactive/science/space/keplers-tally-of-planets.html.

Likert, R. (1932). A technique for the measurement of attitudes. New York: Columbia University Press.

Linse, A.R. (2017). Interpreting and using student ratings data: Guidance for faculty serving as administrators and on evaluation committees. Studies in Educational Evaluation, Volume 54. pp. 94-106.

Luo, A., & Keyes, T. (2005). Second set of results in from the career track member survey. Amstat News, pp. 14-15.

Marsh, H.W. (1987). Students' evaluations of university teaching: Research findings, methodological issues, and directions for future research. International Journal of Educational Research, 11 (Whole issue No. 3).

Marsh, H.W., & Roche, L.A (1997). Making students' evaluations of teaching effectiveness effective: The critical issues of validity, bias, and utility. American Psychologist, Volume 52. No. 11. pp. 1187-1197.

McKeachie, W.J. (1997). Student ratings: The validity of use. American Psychologist, Vol 52. No 11. pp. 1218-1225.

Norman, G. (2010). Likert scales, levels of measurement and the "laws" of statistics. Advances in health sciences education, Volume 15. No. 5. pp. 625.

Paulhus, D.L. (1984). Two-component models of socially desirable responding. Journal of Personality and Social Psychology, Volume 46. No. 3. pp. 598-609.

Shafranovich, Y. (2005). Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC 4180. Retrieved from https://tools.ietf.org/html/rfc4180.

Theall, M. & Franklin, J. (2001). Looking for bias in all the wrong places: A search for truth or a witch hunt in student ratings of instruction? New Directions for Institutional Research, Volume 109. pp. 45-56.

Tufte, E.R. (1983). The visual display of quantitative information. Graphics Press, Cheshire, CT.

Wachtel, H.K. (1998). Student Evaluation of College Teaching Effectiveness: a brief review. Assessment & Evaluation in Higher Education. Volume 23. No. 2. pp. 191-212.

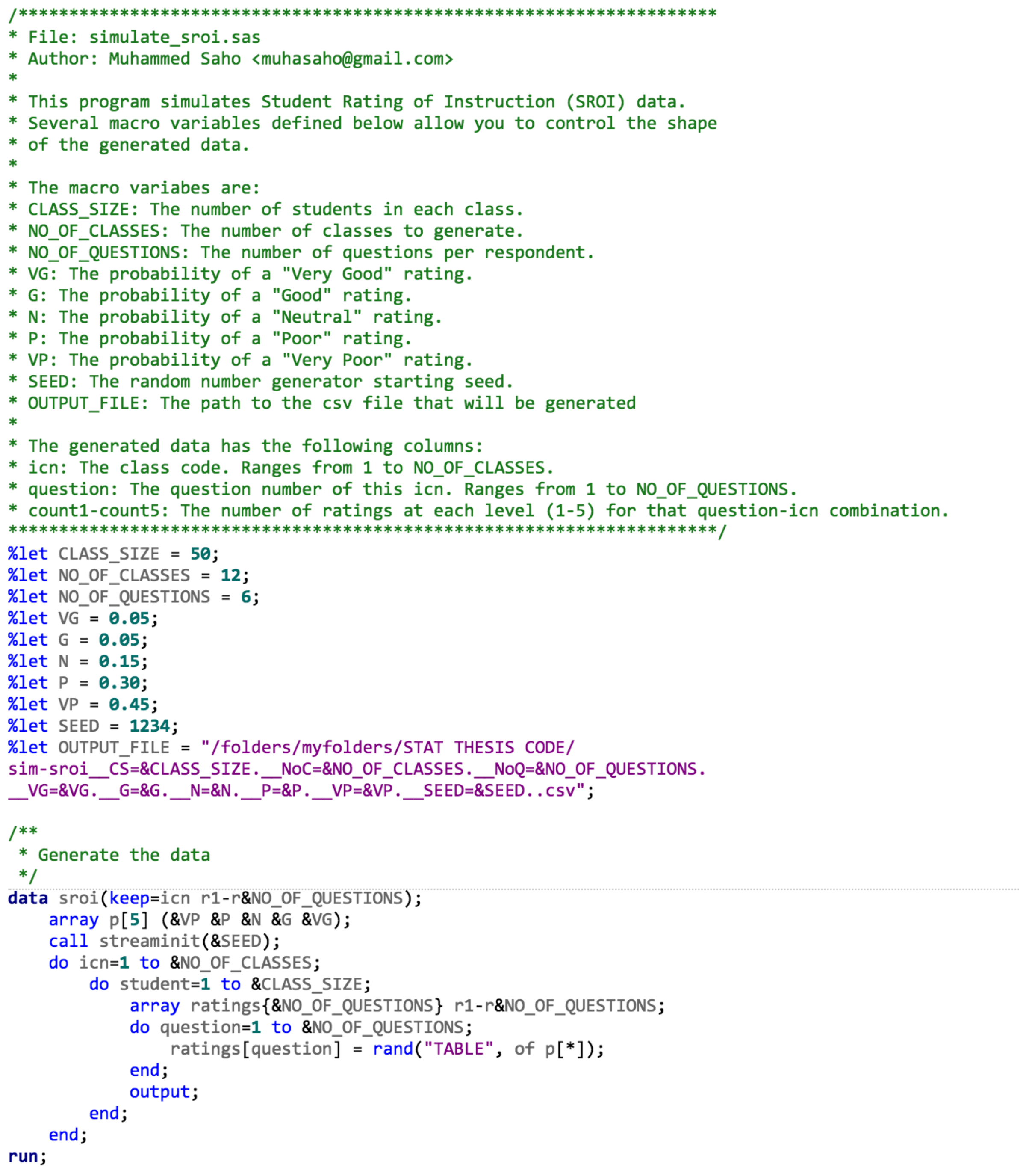
Walker, F.A. (1874). Persons with gainful occupations and also attending school. Statistical atlas of the United States based on the results of the 9th census.

Weinstein, J.N. (2008). A postgenomic visual icon. Science magazine. Volume 319. No. 5871. pp. 1772-1773.

Wicklin, R. (2011, July 13). Simulate Categorical Data in SAS. Retrieved from blogs.sas.com/content/iml/2011/07/13/simulate-categorical-data-in-sas.html.

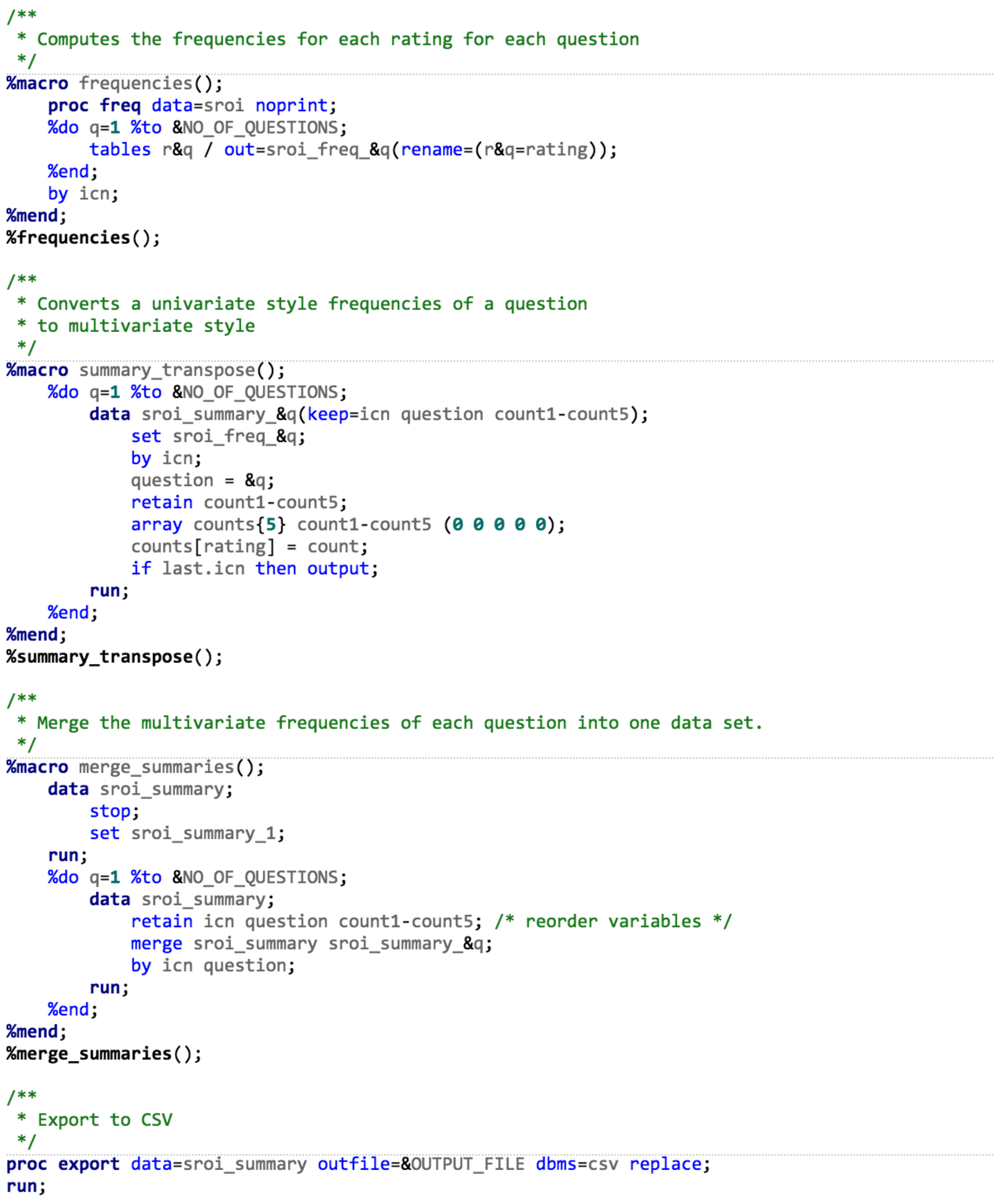
Wilkinson, L., & Friendly, M. (2009). The history of the cluster heat map. The American Statistician. Volume 63. No. 2. pp. 179-184.

APPENDIX A. SAS SIMULATION CODE



```
/**********************************************************************
* File: simulate_sroi.sas
* Author: Muhammed Saho <muhasaho@gmail.com>
*
* This program simulates Student Rating of Instruction (SROI) data.
* Several macro variables defined below allow you to control the shape
* of the generated data.
*
* The macro variabes are:
* CLASS_SIZE: The number of students in each class.
* NO_OF_CLASSES: The number of classes to generate.
* NO_OF_QUESTIONS: The number of questions per respondent.
* VG: The probability of a "Very Good" rating.
* G: The probability of a "Good" rating.
* N: The probability of a "Neutral" rating.
* P: The probability of a "Poor" rating.
* VP: The probability of a "Very Poor" rating.
* SEED: The random number generator starting seed.
* OUTPUT_FILE: The path to the csv file that will be generated
*
* The generated data has the following columns:
* icn: The class code. Ranges from 1 to NO_OF_CLASSES.
* question: The question number of this icn. Ranges from 1 to NO_OF_QUESTIONS.
* count1-count5: The number of ratings at each level (1-5) for that question-icn combination.
**********************************************************************/
%let CLASS_SIZE = 50;
%let NO_OF_CLASSES = 12;
%let NO_OF_QUESTIONS = 6;
%let VG = 0.05;
%let G = 0.05;
%let N = 0.15;
%let P = 0.30;
%let VP = 0.45;
%let SEED = 1234;
%let OUTPUT_FILE = "/folders/myfolders/STAT THESIS CODE/
sim-sroi__CS=&CLASS_SIZE.__NoC=&NO_OF_CLASSES.__NoQ=&NO_OF_QUESTIONS.
__VG=&VG.__G=&G.__N=&N.__P=&P.__VP=&VP.__SEED=&SEED..csv";

/**
 * Generate the data
 */
data sroi(keep=icn r1-r&NO_OF_QUESTIONS);
    array p[5] (&VP &P &N &G &VG);
    call streaminit(&SEED);
    do icn=1 to &NO_OF_CLASSES;
        do student=1 to &CLASS_SIZE;
            array ratings{&NO_OF_QUESTIONS} r1-r&NO_OF_QUESTIONS;
            do question=1 to &NO_OF_QUESTIONS;
                ratings[question] = rand("TABLE", of p[*]);
            end;
            output;
        end;
    end;
run;
```

Figure A1. SAS Simulation Code (Page 1 of 2).



```
/**
 * Computes the frequencies for each rating for each question
 */
%macro frequencies();
    proc freq data=sroi noprint;
    %do q=1 %to &NO_OF_QUESTIONS;
        tables r&q / out=sroi_freq_&q(rename=(r&q=rating));
    %end;
    by icn;
%mend;
%frequencies();

/**
 * Converts a univariate style frequencies of a question
 * to multivariate style
 */
%macro summary_transpose();
    %do q=1 %to &NO_OF_QUESTIONS;
        data sroi_summary_&q(keep=icn question count1-count5);
            set sroi_freq_&q;
            by icn;
            question = &q;
            retain count1-count5;
            array counts{5} count1-count5 (0 0 0 0 0);
            counts[rating] = count;
            if last.icn then output;
        run;
    %end;
%mend;
%summary_transpose();

/**
 * Merge the multivariate frequencies of each question into one data set.
 */
%macro merge_summaries();
    data sroi_summary;
        stop;
        set sroi_summary_1;
    run;
    %do q=1 %to &NO_OF_QUESTIONS;
        data sroi_summary;
            retain icn question count1-count5; /* reorder variables */
            merge sroi_summary sroi_summary_&q;
            by icn question;
        run;
    %end;
%mend;
%merge_summaries();

/**
 * Export to CSV
 */
proc export data=sroi_summary outfile=&OUTPUT_FILE dbms=csv replace;
run;
```

Figure A2. SAS Simulation Code (Page 2 of 2)

APPENDIX B. REFERENCE VISUALIZATIONS

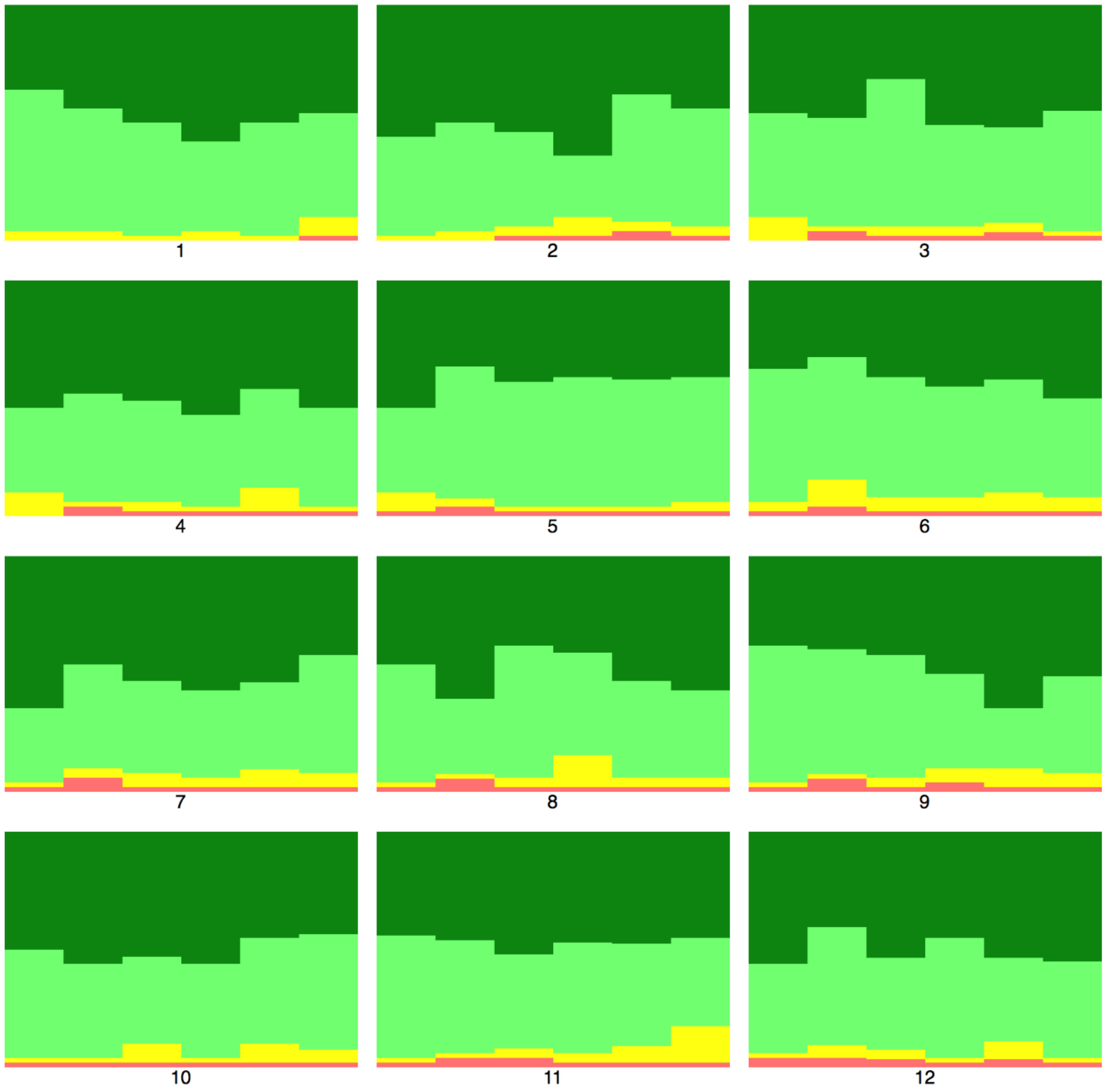


Figure B1. Simulated at 50% VG, 45% G, 4% N, 1% P, and 0% VP.
A positive pattern manifests as mostly green blocks with occasional yellow and red.

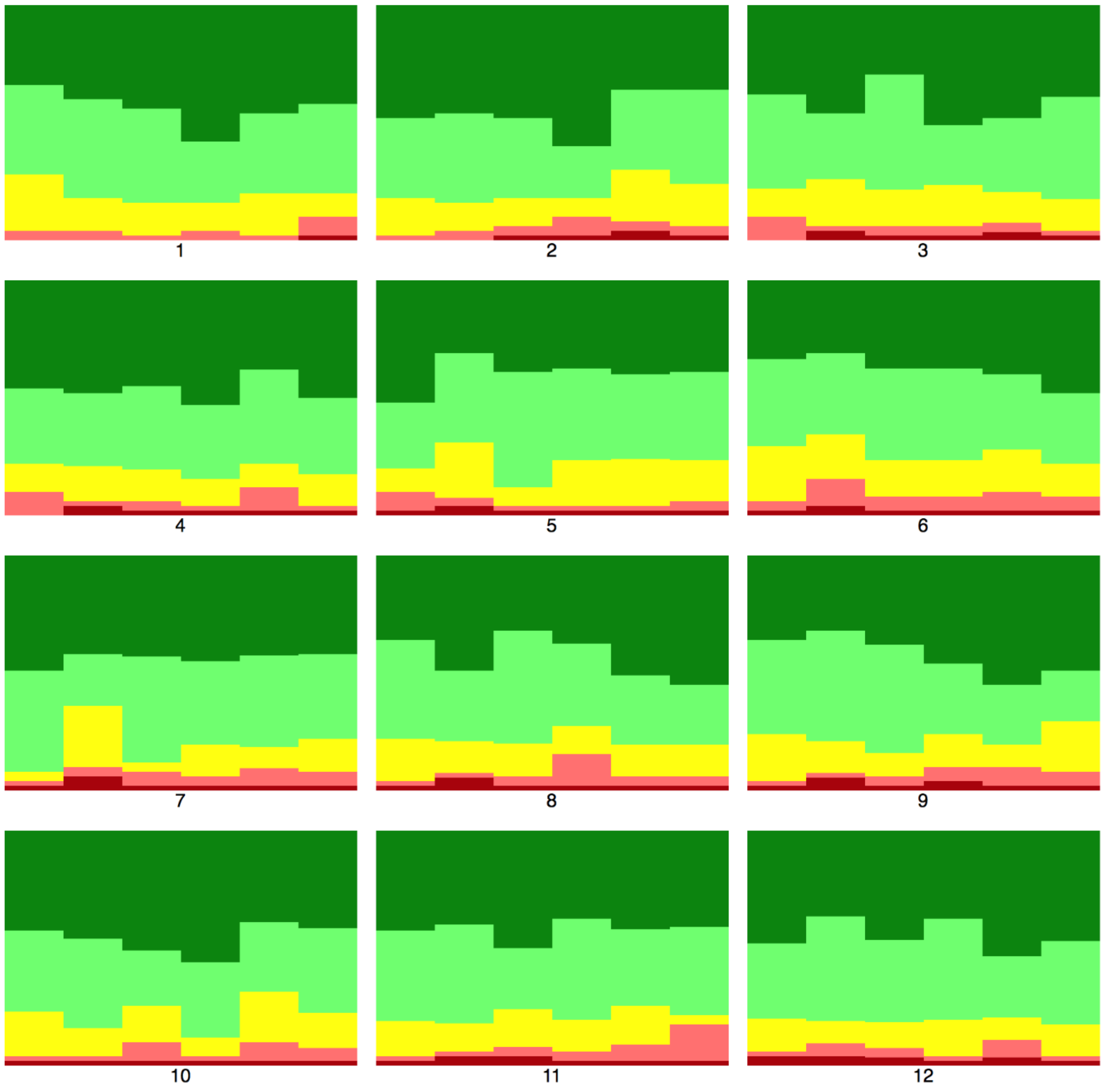


Figure B2. Simulated at 45% VG, 35% G, 15% N, 4% P, and 1% VP.
A generally positive pattern with higher than usual negative ratings.

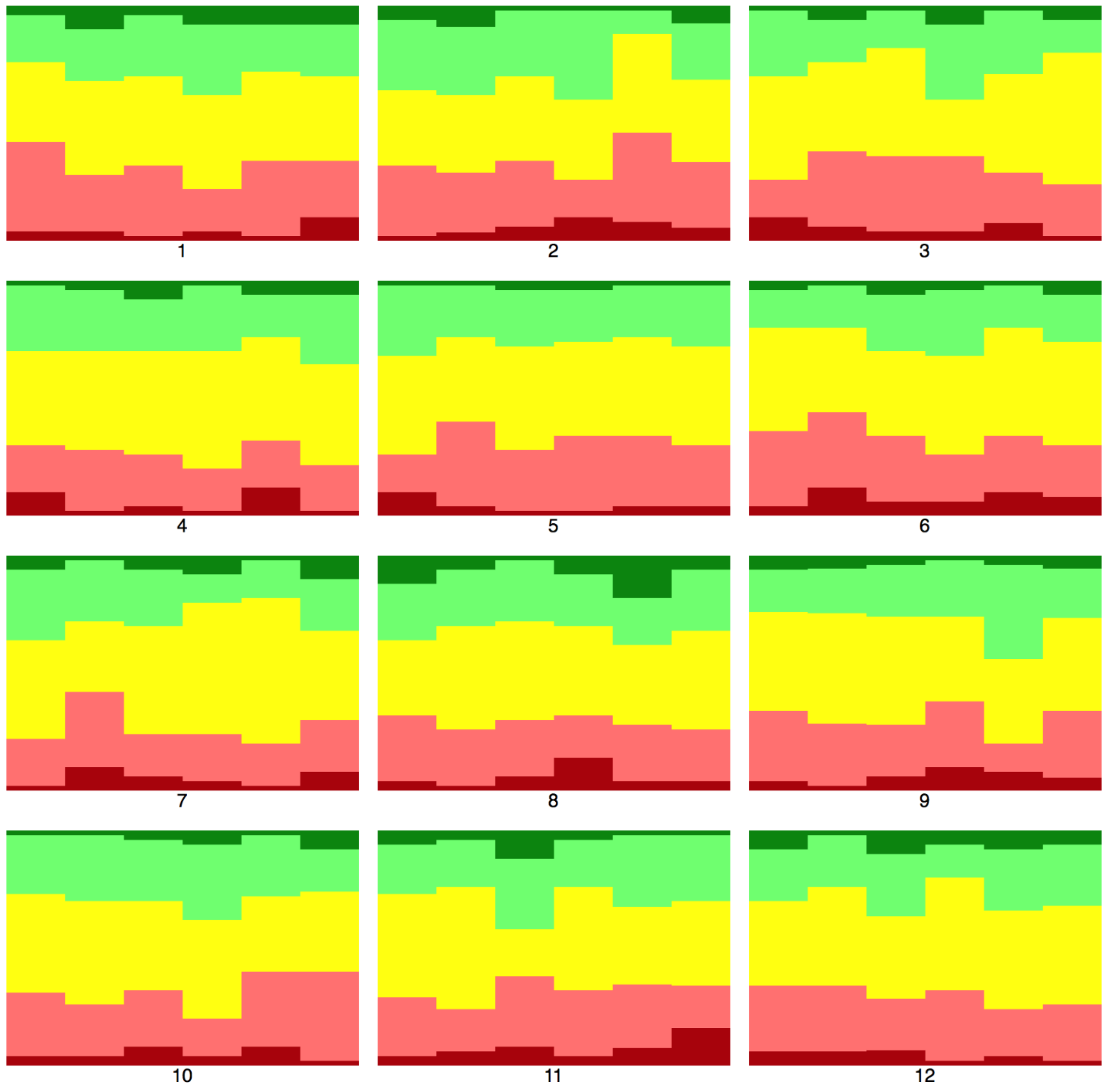


Figure B3. Simulated at 5% VG, 25% G, 40% N, 25% P, and 5% VP.
A neutral pattern manifests as large blocks of light green, yellow, and light red sandwiched between thin strips of dark green and dark red.

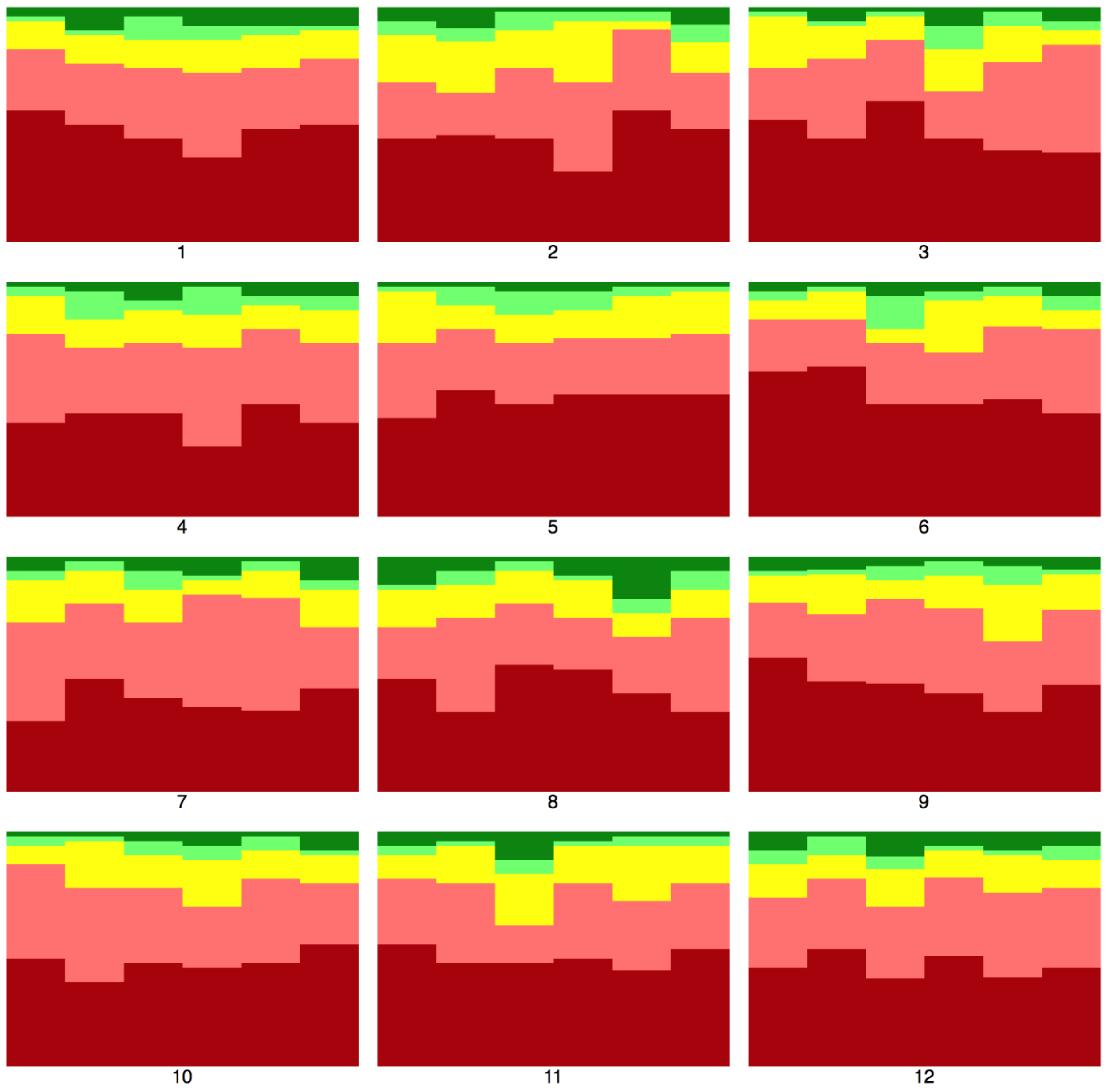
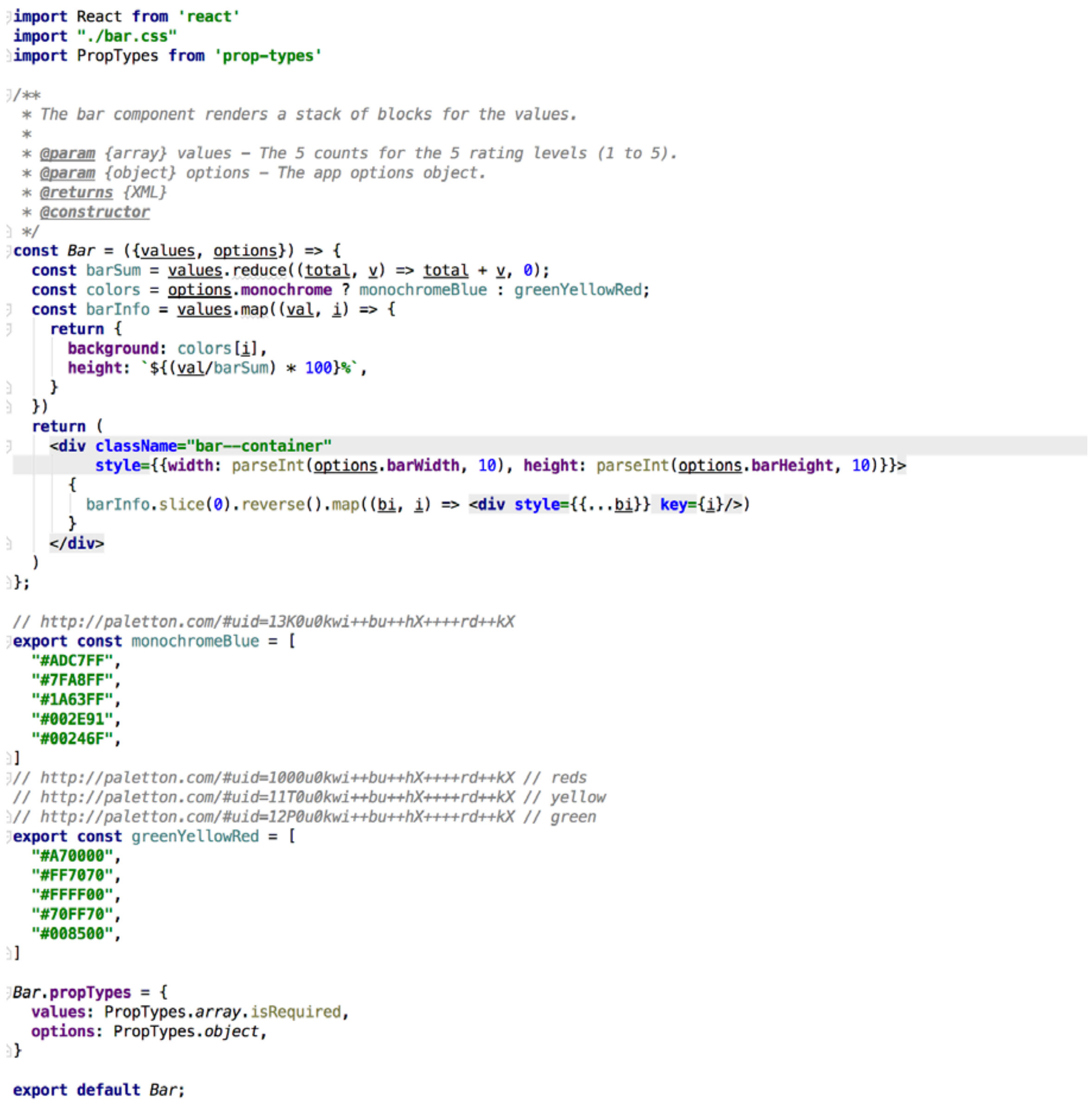


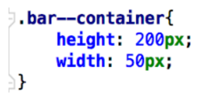
Figure B4. Simulated at 5% VG, 5% G, 15% N, 30% P, and 45% VP.
A negative pattern manifests as smaller green and yellow blocks sitting on big red blocks.

APPENDIX C. WEB APPLICATION CODE



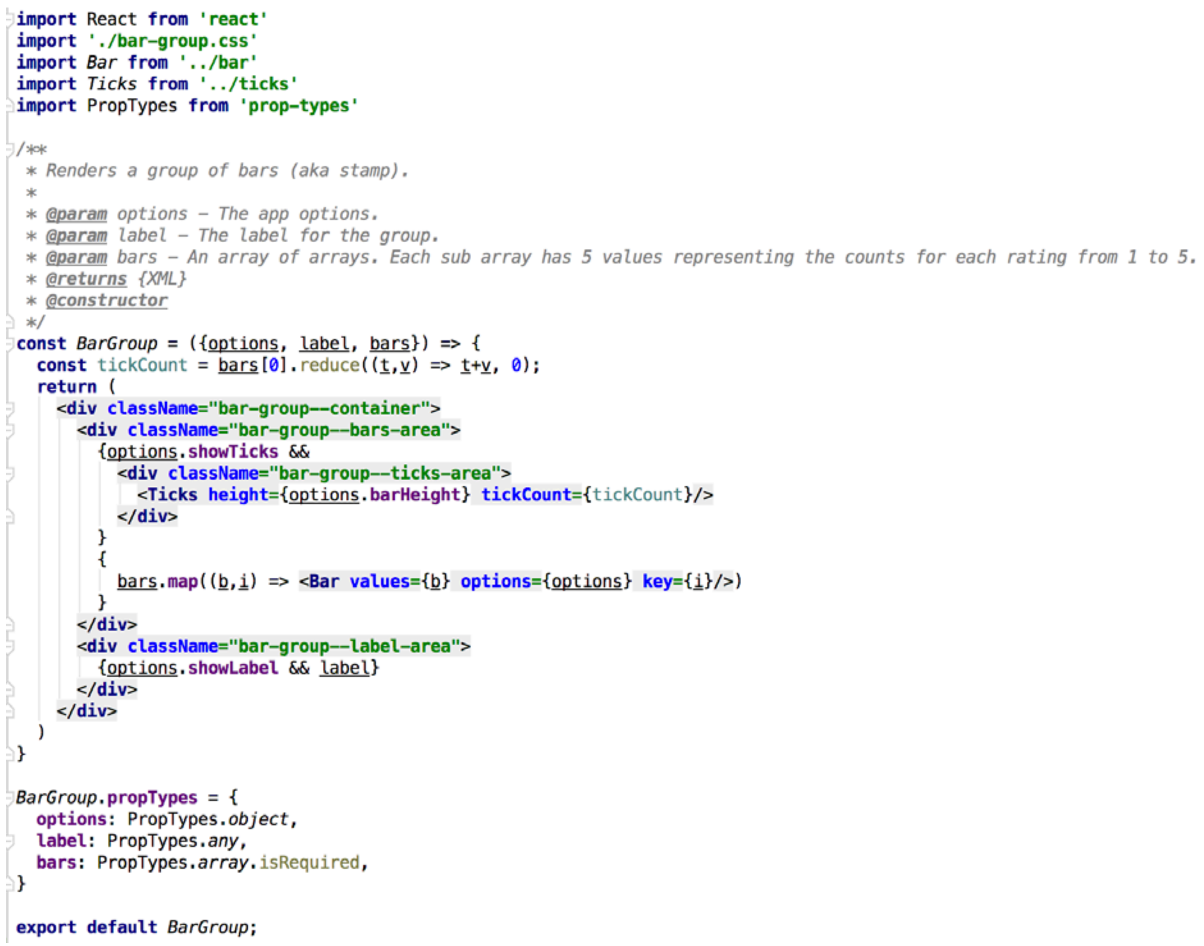
```
import React from 'react'
import "./bar.css"
import PropTypes from 'prop-types'

/**
 * The bar component renders a stack of blocks for the values.
 *
 * @param {array} values - The 5 counts for the 5 rating levels (1 to 5).
 * @param {object} options - The app options object.
 * @returns {XML}
 * @constructor
 */
const Bar = ({values, options}) => {
  const barSum = values.reduce((total, v) => total + v, 0);
  const colors = options.monochrome ? monochromeBlue : greenYellowRed;
  const barInfo = values.map((val, i) => {
    return {
      background: colors[i],
      height: `${(val/barSum) * 100}%`,
    }
  })
  return (
    <div className="bar--container"
         style={{width: parseInt(options.barWidth, 10), height: parseInt(options.barHeight, 10)}}>
      {
        barInfo.slice(0).reverse().map((bi, i) => <div style={{...bi}} key={i}/>)
      }
    </div>
  )
};

// http://paletton.com/#uid=13K0u0kwi++bu++hX++++rd++kX
export const monochromeBlue = [
  "#ADC7FF",
  "#7FA8FF",
  "#1A63FF",
  "#002E91",
  "#00246F",
]
// http://paletton.com/#uid=1000u0kwi++bu++hX++++rd++kX // reds
// http://paletton.com/#uid=11T0u0kwi++bu++hX++++rd++kX // yellow
// http://paletton.com/#uid=12P0u0kwi++bu++hX++++rd++kX // green
export const greenYellowRed = [
  "#A70000",
  "#FF7070",
  "#FFFF00",
  "#70FF70",
  "#008500",
]

Bar.propTypes = {
  values: PropTypes.array.isRequired,
  options: PropTypes.object,
}

export default Bar;
```

Figure C1. Bar.js



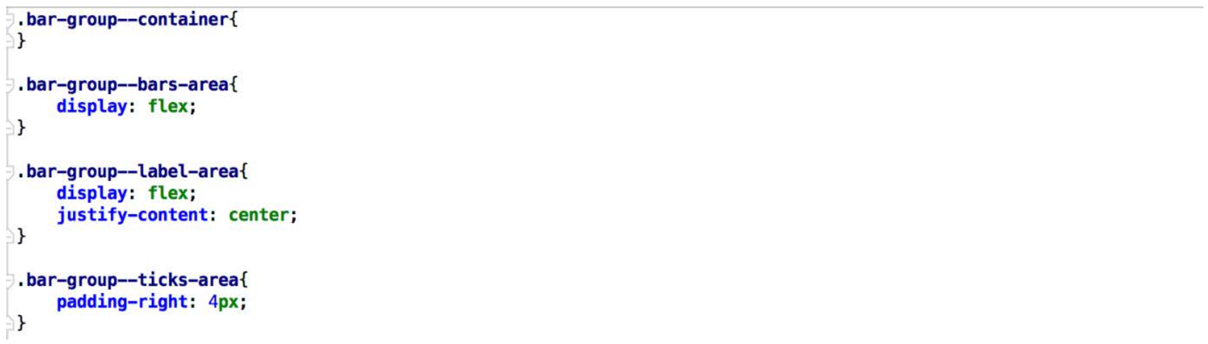
```
.bar--container{
    height: 200px;
    width: 50px;
}
```

Figure C2. Bar.css



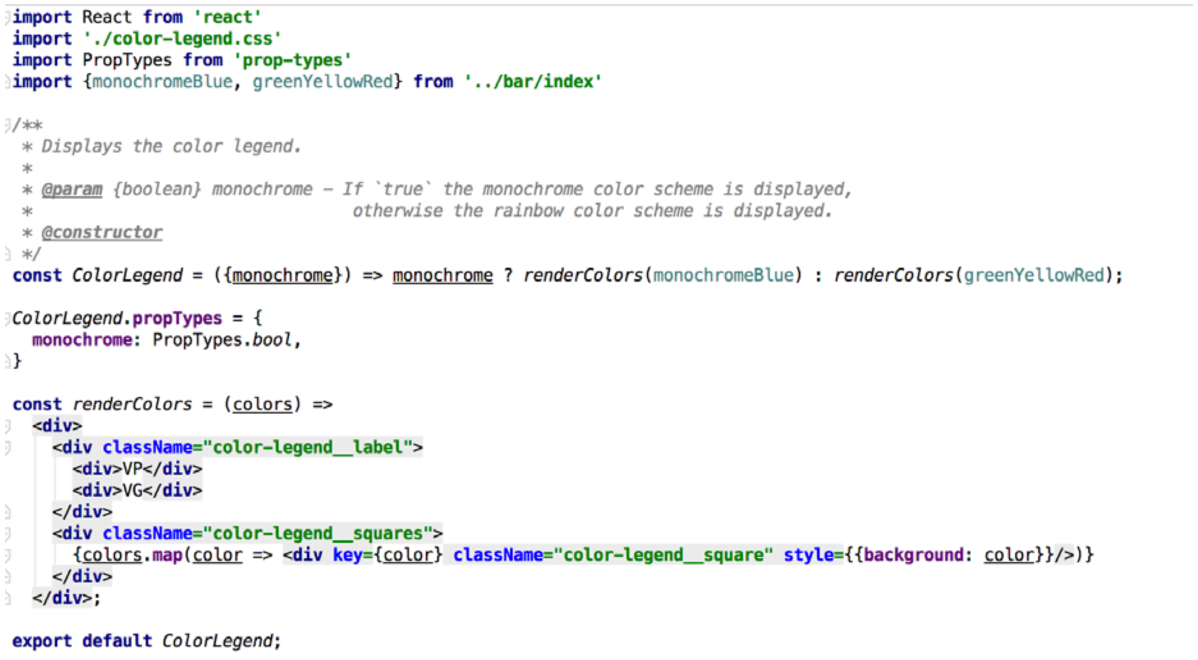
```
import React from 'react'
import './bar-group.css'
import Bar from '../bar'
import Ticks from '../ticks'
import PropTypes from 'prop-types'

/**
 * Renders a group of bars (aka stamp).
 *
 * @param options - The app options.
 * @param label - The label for the group.
 * @param bars - An array of arrays. Each sub array has 5 values representing the counts for each rating from 1 to 5.
 * @returns {XML}
 * @constructor
 */
const BarGroup = ({options, label, bars}) => {
  const tickCount = bars[0].reduce((t,v) => t+v, 0);
  return (
    <div className="bar-group--container">
      <div className="bar-group--bars-area">
        {options.showTicks &&
          <div className="bar-group--ticks-area">
            <Ticks height={options.barHeight} tickCount={tickCount}/>
          </div>
        }
        {
          bars.map((b,i) => <Bar values={b} options={options} key={i}/>)
        }
      </div>
      <div className="bar-group--label-area">
        {options.showLabel && label}
      </div>
    </div>
  )
}

BarGroup.propTypes = {
  options: PropTypes.object,
  label: PropTypes.any,
  bars: PropTypes.array.isRequired,
}

export default BarGroup;
```

Figure C3. Bar-group.js



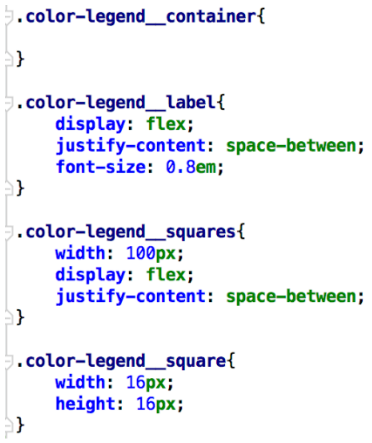
```
.bar-group--container{
}

.bar-group--bars-area{
    display: flex;
}

.bar-group--label-area{
    display: flex;
    justify-content: center;
}

.bar-group--ticks-area{
    padding-right: 4px;
}
```

Figure C4. Bar-group.css



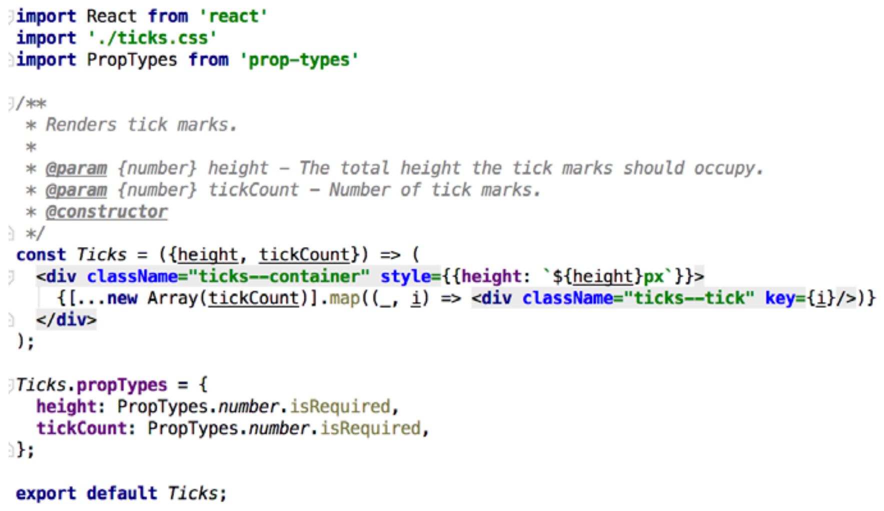
```
import React from 'react'
import './color-legend.css'
import PropTypes from 'prop-types'
import {monochromeBlue, greenYellowRed} from '../bar/index'

/**
 * Displays the color legend.
 *
 * @param {boolean} monochrome - If `true` the monochrome color scheme is displayed,
 *                               otherwise the rainbow color scheme is displayed.
 * @constructor
 */
const ColorLegend = ({monochrome}) => monochrome ? renderColors(monochromeBlue) : renderColors(greenYellowRed);

ColorLegend.propTypes = {
  monochrome: PropTypes.bool,
}

const renderColors = (colors) =>
  <div>
    <div className="color-legend__label">
      <div>VP</div>
      <div>VG</div>
    </div>
    <div className="color-legend__squares">
      {colors.map(color => <div key={color} className="color-legend__square" style={{background: color}}/>)}
    </div>
  </div>;

export default ColorLegend;
```

Figure C5. Color-legend.js



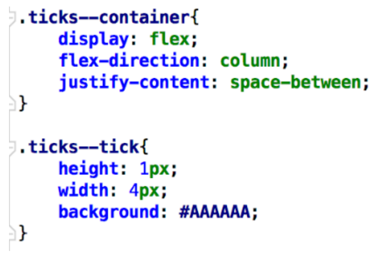
```
.color-legend__container{

}

.color-legend__label{
    display: flex;
    justify-content: space-between;
    font-size: 0.8em;
}

.color-legend__squares{
    width: 100px;
    display: flex;
    justify-content: space-between;
}

.color-legend__square{
    width: 16px;
    height: 16px;
}
```

Figure C6. Color-legend.css



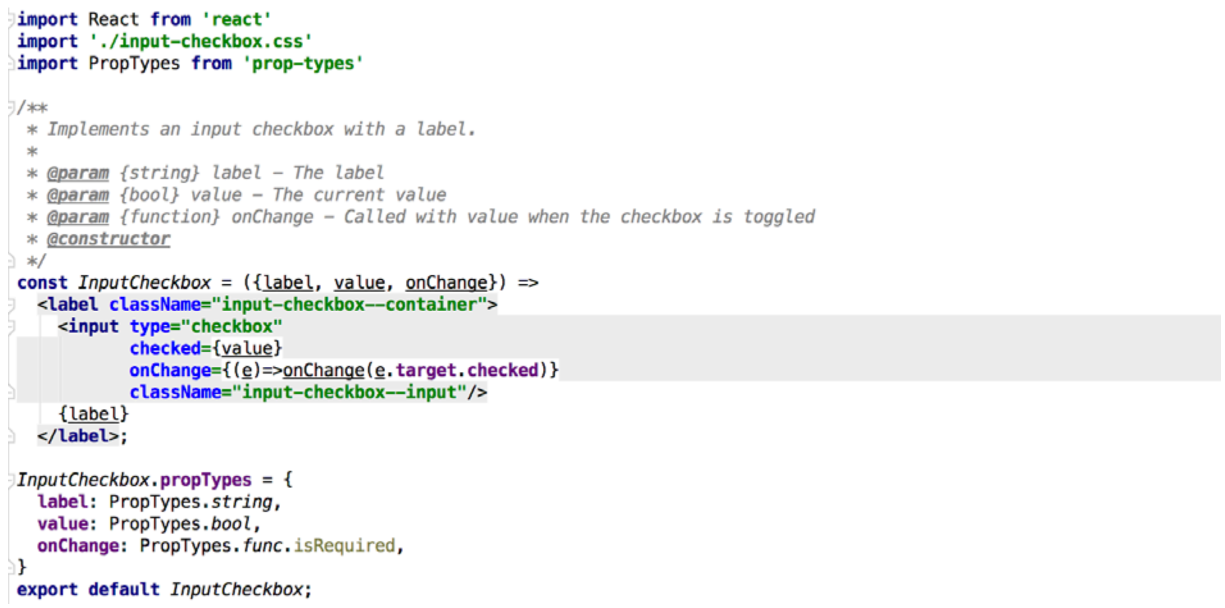
```
import React from 'react'
import './ticks.css'
import PropTypes from 'prop-types'

/**
 * Renders tick marks.
 *
 * @param {number} height - The total height the tick marks should occupy.
 * @param {number} tickCount - Number of tick marks.
 * @constructor
 */
const Ticks = ({height, tickCount}) => (
  <div className="ticks--container" style={{height: `${height}px`}}>
    {[...new Array(tickCount)].map((_, i) => <div className="ticks--tick" key={i}/>)}
  </div>
);

Ticks.propTypes = {
  height: PropTypes.number.isRequired,
  tickCount: PropTypes.number.isRequired,
};

export default Ticks;
```

Figure C7. Ticks.js



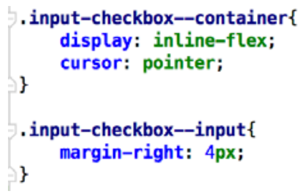
```
.ticks--container{
    display: flex;
    flex-direction: column;
    justify-content: space-between;
}

.ticks--tick{
    height: 1px;
    width: 4px;
    background: #AAAAAA;
}
```

Figure C8. Ticks.css



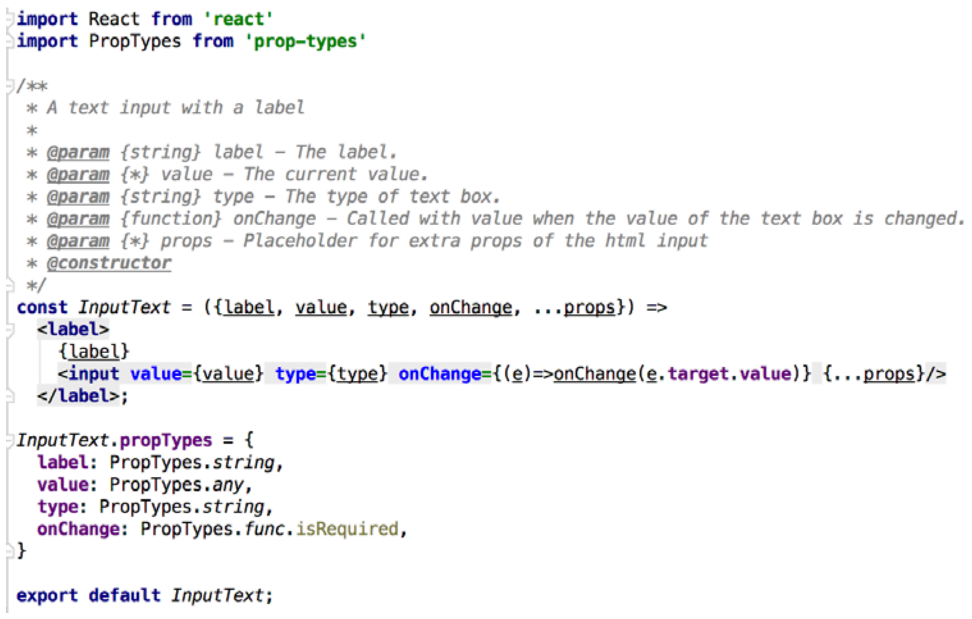
```
import React from 'react'
import './input-checkbox.css'
import PropTypes from 'prop-types'

/**
 * Implements an input checkbox with a label.
 *
 * @param {string} label - The label
 * @param {bool} value - The current value
 * @param {function} onChange - Called with value when the checkbox is toggled
 * @constructor
 */
const InputCheckbox = ({label, value, onChange}) =>
  <label className="input-checkbox--container">
    <input type="checkbox"
           checked={value}
           onChange={(e)=>onChange(e.target.checked)}
           className="input-checkbox--input"/>
    {label}
  </label>;

InputCheckbox.propTypes = {
  label: PropTypes.string,
  value: PropTypes.bool,
  onChange: PropTypes.func.isRequired,
}
export default InputCheckbox;
```

Figure C9. Input-checkbox.js



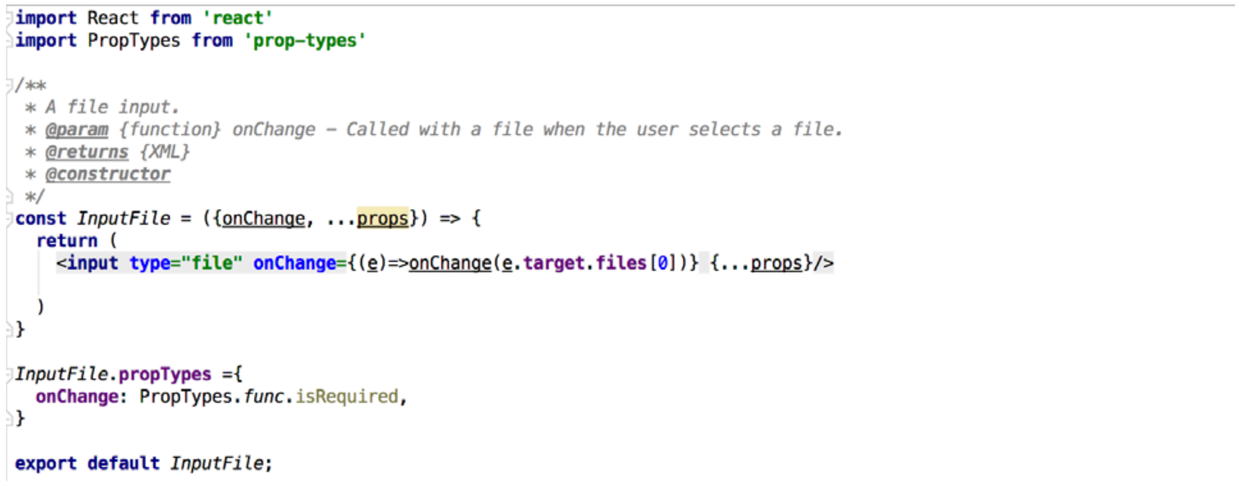
```
.input-checkbox--container{
    display: inline-flex;
    cursor: pointer;
}

.input-checkbox--input{
    margin-right: 4px;
}
```

Figure C10. Input-checkbox.css



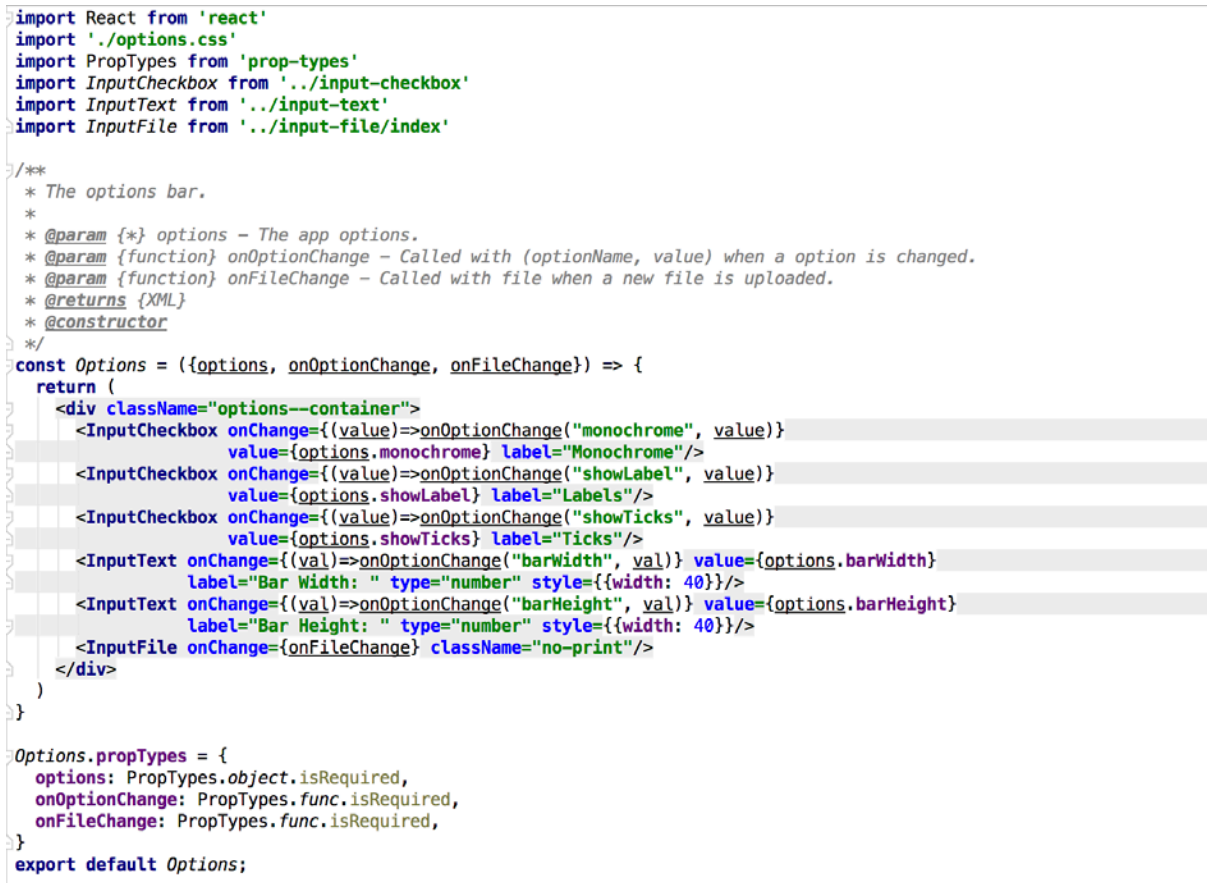
```
import React from 'react'
import PropTypes from 'prop-types'

/**
 * A text input with a label
 *
 * @param {string} label - The label.
 * @param {*} value - The current value.
 * @param {string} type - The type of text box.
 * @param {function} onChange - Called with value when the value of the text box is changed.
 * @param {*} props - Placeholder for extra props of the html input
 * @constructor
 */
const InputText = ({label, value, type, onChange, ...props}) =>
  <label>
    {label}
    <input value={value} type={type} onChange={(e)=>onChange(e.target.value)} {...props}/>
  </label>;

InputText.propTypes = {
  label: PropTypes.string,
  value: PropTypes.any,
  type: PropTypes.string,
  onChange: PropTypes.func.isRequired,
}

export default InputText;
```

Figure C11. Input-text.js



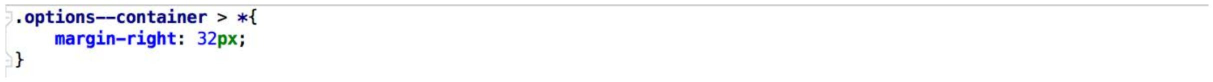
```
import React from 'react'
import PropTypes from 'prop-types'

/**
 * A file input.
 * @param {function} onChange - Called with a file when the user selects a file.
 * @returns {XML}
 * @constructor
 */
const InputFile = ({onChange, ...props}) => {
  return (
    <input type="file" onChange={(e)=>onChange(e.target.files[0])} {...props}/>

  )
}

InputFile.propTypes ={
  onChange: PropTypes.func.isRequired,
}

export default InputFile;
```

Figure C12. Input-file.js



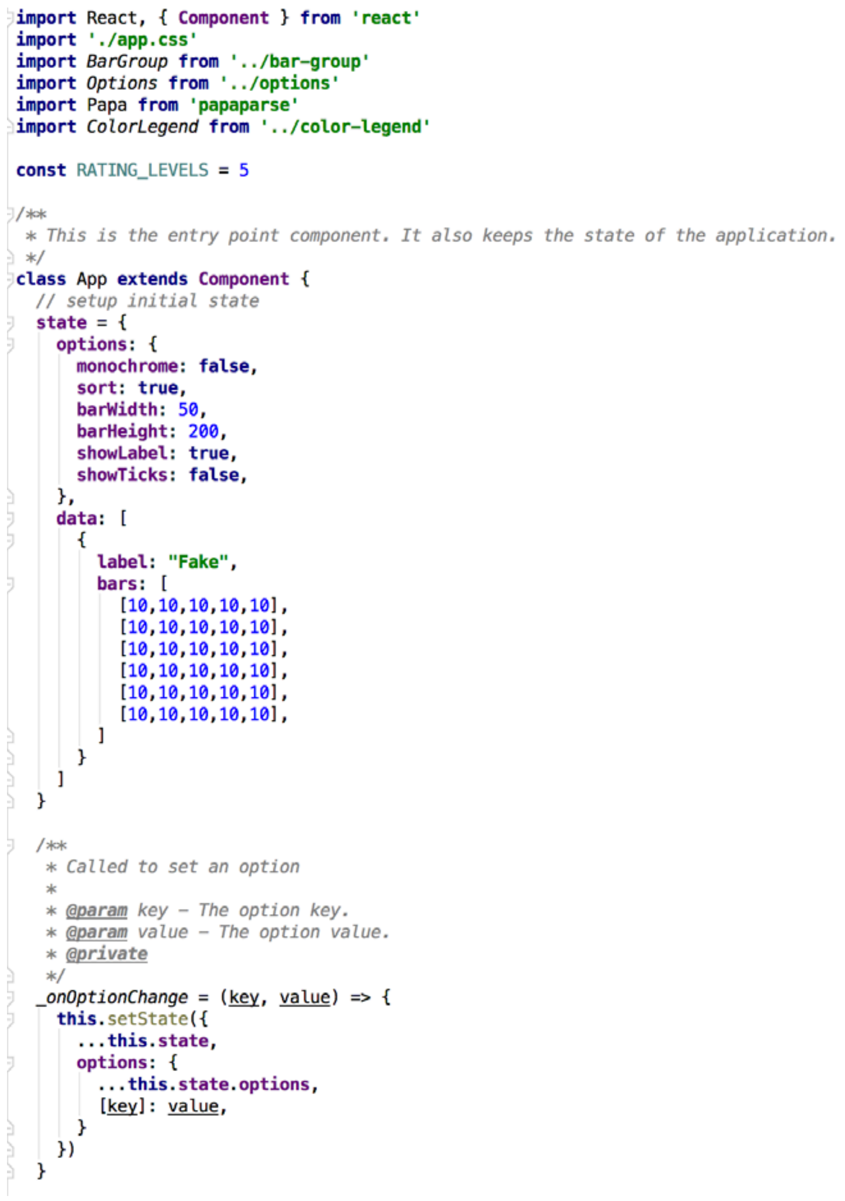
```
import React from 'react'
import './options.css'
import PropTypes from 'prop-types'
import InputCheckbox from '../input-checkbox'
import InputText from '../input-text'
import InputFile from '../input-file/index'

/**
 * The options bar.
 *
 * @param {*} options - The app options.
 * @param {function} onOptionChange - Called with (optionName, value) when a option is changed.
 * @param {function} onFileChange - Called with file when a new file is uploaded.
 * @returns {XML}
 * @constructor
 */
const Options = ({options, onOptionChange, onFileChange}) => {
  return (
    <div className="options--container">
      <InputCheckbox onChange={(value)=>onOptionChange("monochrome", value)}
                     value={options.monochrome} label="Monochrome"/>
      <InputCheckbox onChange={(value)=>onOptionChange("showLabel", value)}
                     value={options.showLabel} label="Labels"/>
      <InputCheckbox onChange={(value)=>onOptionChange("showTicks", value)}
                     value={options.showTicks} label="Ticks"/>
      <InputText onChange={(val)=>onOptionChange("barWidth", val)} value={options.barWidth}
                 label="Bar Width: " type="number" style={{width: 40}}/>
      <InputText onChange={(val)=>onOptionChange("barHeight", val)} value={options.barHeight}
                 label="Bar Height: " type="number" style={{width: 40}}/>
      <InputFile onChange={onFileChange} className="no-print"/>
    </div>
  )
}

Options.propTypes = {
  options: PropTypes.object.isRequired,
  onOptionChange: PropTypes.func.isRequired,
  onFileChange: PropTypes.func.isRequired,
}
export default Options;
```

Figure C13. Options.js



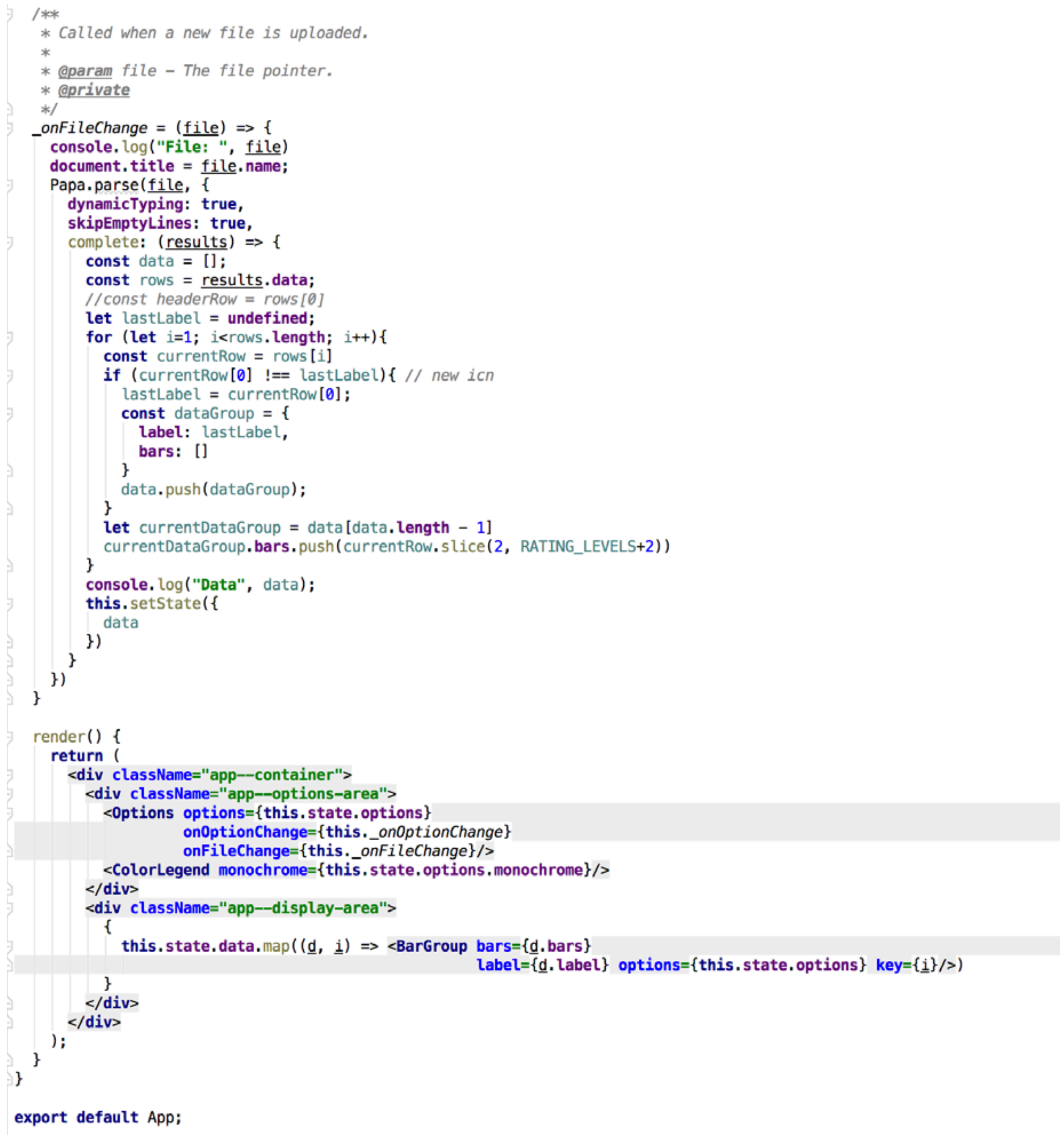
```
.options--container > *{
    margin-right: 32px;
}
```

Figure C14. Options.css



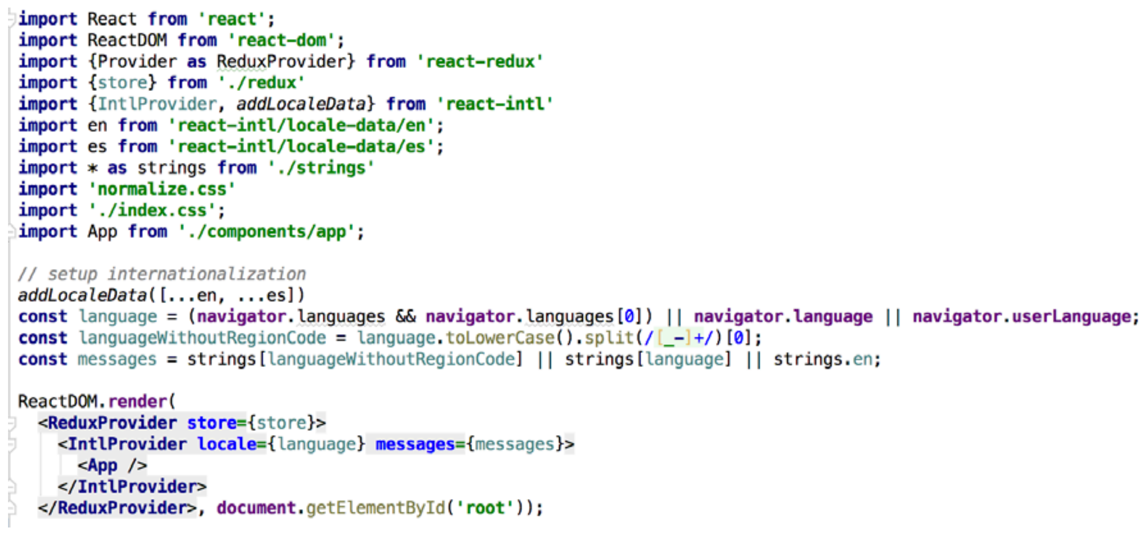
```
import React, { Component } from 'react'
import './app.css'
import BarGroup from '../bar-group'
import Options from '../options'
import Papa from 'papaparse'
import ColorLegend from '../color-legend'

const RATING_LEVELS = 5

/**
 * This is the entry point component. It also keeps the state of the application.
 */
class App extends Component {
  // setup initial state
  state = {
    options: {
      monochrome: false,
      sort: true,
      barWidth: 50,
      barHeight: 200,
      showLabel: true,
      showTicks: false,
    },
    data: [
      {
        label: "Fake",
        bars: [
          [10,10,10,10,10],
          [10,10,10,10,10],
          [10,10,10,10,10],
          [10,10,10,10,10],
          [10,10,10,10,10],
          [10,10,10,10,10],
        ]
      }
    ]
  }

  /**
   * Called to set an option
   *
   * @param key - The option key.
   * @param value - The option value.
   * @private
   */
  _onOptionChange = (key, value) => {
    this.setState({
      ...this.state,
      options: {
        ...this.state.options,
        [key]: value,
      }
    })
  }
```

Figure C15. App.js (1 of 2)



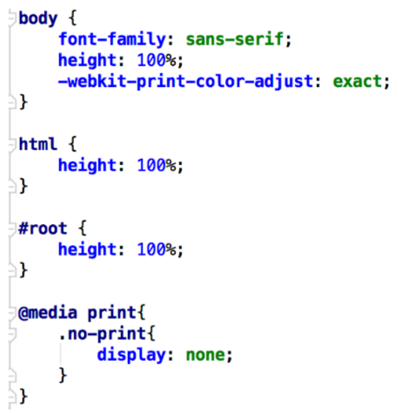
```
  /**
   * Called when a new file is uploaded.
   *
   * @param file - The file pointer.
   * @private
   */
  _onFileChange = (file) => {
    console.log("File: ", file)
    document.title = file.name;
    Papa.parse(file, {
      dynamicTyping: true,
      skipEmptyLines: true,
      complete: (results) => {
        const data = [];
        const rows = results.data;
        //const headerRow = rows[0]
        let lastLabel = undefined;
        for (let i=1; i<rows.length; i++){
          const currentRow = rows[i]
          if (currentRow[0] !== lastLabel){ // new icn
            lastLabel = currentRow[0];
            const dataGroup = {
              label: lastLabel,
              bars: []
            }
            data.push(dataGroup);
          }
          let currentDataGroup = data[data.length - 1]
          currentDataGroup.bars.push(currentRow.slice(2, RATING_LEVELS+2))
        }
        console.log("Data", data);
        this.setState({
          data
        })
      }
    })
  }

  render() {
    return (
      <div className="app--container">
        <div className="app--options-area">
          <Options options={this.state.options}
                   onOptionChange={this._onOptionChange}
                   onFileChange={this._onFileChange}/>
          <ColorLegend monochrome={this.state.options.monochrome}/>
        </div>
        <div className="app--display-area">
          {
            this.state.data.map((d, i) => <BarGroup bars={d.bars}
                                              label={d.label} options={this.state.options} key={i}/>)
          }
        </div>
      </div>
    );
  }
}

export default App;
```

Figure C16. App.js (2 of 2)



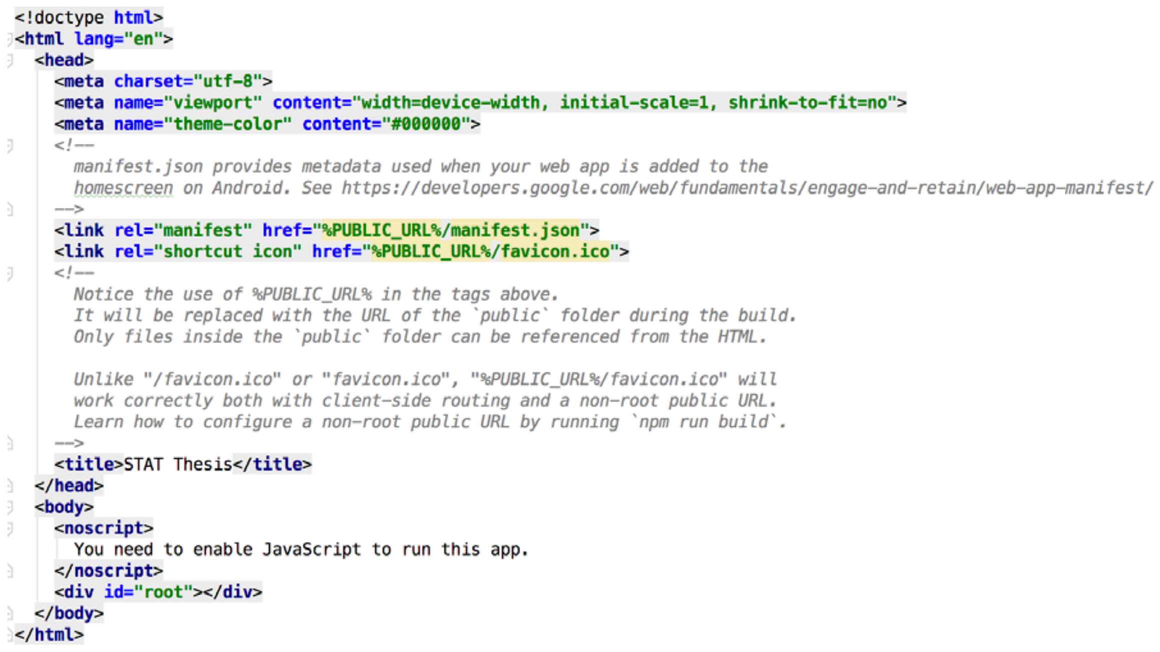
```
import React from 'react';
import ReactDOM from 'react-dom';
import {Provider as ReduxProvider} from 'react-redux'
import {store} from './redux'
import {IntlProvider, addLocaleData} from 'react-intl'
import en from 'react-intl/locale-data/en';
import es from 'react-intl/locale-data/es';
import * as strings from './strings'
import 'normalize.css'
import './index.css';
import App from './components/app';

// setup internationalization
addLocaleData([...en, ...es])
const language = (navigator.languages && navigator.languages[0]) || navigator.language || navigator.userLanguage;
const languageWithoutRegionCode = language.toLowerCase().split(/[_-]+/)[0];
const messages = strings[languageWithoutRegionCode] || strings[language] || strings.en;

ReactDOM.render(
  <ReduxProvider store={store}>
    <IntlProvider locale={language} messages={messages}>
      <App />
    </IntlProvider>
  </ReduxProvider>, document.getElementById('root'));
```

Figure C17. Index.js



```
body {
    font-family: sans-serif;
    height: 100%;
    -webkit-print-color-adjust: exact;
}

html {
    height: 100%;
}

#root {
    height: 100%;
}

@media print{
    .no-print{
        display: none;
    }
}
```

Figure C18. Index.css



```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <!--
      manifest.json provides metadata used when your web app is added to the
      homescreen on Android. See https://developers.google.com/web/fundamentals/engage-and-retain/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>STAT Thesis</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
  </body>
</html>
```

Figure C19. Index.html