EmoViz

Final Report

CS 4624 Multimedia, Hypertext, and Information Access Capstone
Spring 2018

Instructor: Dr. Edward A. Fox
Virginia Tech, Blacksburg, VA 24061

Authors

Thomas Evans
Matthew Barnett

Additional Team Members

Fuadul Islam
Yibing Zhang

Client

Professor Steven D. Sheetz

May 2, 2018

Table of Contents

Table of Tables 4
Table of Figures 5
I. Executive Summary 6
II. Introduction 7
A. Goals and Objectives 7
B. Statement of Scope 7
C. Client Background 7
D. Major Constraints 7
E. Organization of Report 8
III. Requirements 8
A. Use of Apache Spark 8
B. Support for Multiple Types of Visualizations 8
C. Filter and Compare Multiple Data Sets 9
D. View Precise Changes in Data 9
E. Local Web Application 9
IV. Design 9
A. Front-End Design 10
B. Back-End Design 10
V. Implementation 11
A. Execution of an EmoViz Job 11
B. Data Normalization 11
C. Colors and Emotions 12
D. Optimizations 12
VI. Testing & Evaluation 12
VII. User's Manual 13
A. Installation & Setup 13
B. Running EmoViz 14
C. Uploading New Data 16
D. Choosing Visualization Types 16
E. Generating Visualizations 17

F. Exporting Visualizations 18
VIII. Developer's Manual 19
A. Configuring Your Development Environment 19
1. Hardware Requirements 19
2. Software Requirements 20
B. Project Structure 21
1. File Listing 21
3. Plot.ly Library 21
C. Adding New Visualization Types 21
IX. Lessons Learned 22
A. Problems and Solutions 22
X. Acknowledgements 23
XI. References 23

Table of Tables

Table 1 -> Colors corresponding to each emotion · **12**

Table 2 -> Relationship between type of visualization and number of data files that can possibly be selected · **16**

Table of Figures

Figure 1 -> Screenshot of EmoViz GitHub repository showing green Clone or download button · · · · · · · · · · · **13**

Figure 2 -> Screenshot of EmoViz Dashboard · **15**

Figure 3 -> EmoViz Dashboard Reference Names · **15**

Figure 4 -> Screenshot of error message generated when invalid input is used · **17**

Figure 5 -> Screenshot of "Download plot as a .png" button · **18**

I. Executive Summary

This report describes the EmoViz project for the Multimedia, Hypertext, and Information Access Capstone at Virginia Tech during the spring 2018 semester. The goal of the EmoViz project is to develop a tool that generates and displays visualizations made from Facial Action Coding System (FACS) emotion data.

The client, Dr. Steven D. Sheetz, is a Professor of Accounting and Information Systems at Virginia Tech. Dr. Sheetz conducted a research project in 2009 to determine how human emotions are affected when a subject is confronted with analyzing a business audit. In the study, an actor was hired to record a five minute video of a simulated business audit in which they read a prewritten script containing specific visual cues at highlighted points throughout the duration of the audit. Participants of the study were divided into two groups, each of which was given a distinct set of accounting data to review prior to watching the simulation video. The first group received accounting data that had purposely been altered in a way that would indicate the actor was committing fraud by lying to the auditor. The second group received accounting data that correctly corresponded to the actor's script so that it would appear there was no fraud committed. All participants watched the simulation video while their face movements were tracked using the Noldus FaceReader software to catalog emotional states. FaceReader samples data points on the face every 33 milliseconds and uses a proprietary algorithm to quantify the following emotions at each sampling: neutral, happy, sad, angry, surprise, and disgust.

After cataloging roughly 9,000 data rows per participant, Dr. Sheetz adjusted the data and exported each set into .csv files. From there, the EmoViz team uploaded these files into the newly developed system where the data was then processed using Apache Spark [1]. Using Spark's powerful key-value mapping algorithm for cluster computing, the .csv data was transformed into Dataframes [2] which helped to map each emotion to a named column. These named columns were then queried in order to generate visualizations and display certain emotions over time. Additionally, the queries helped to compare and contrast different data sets so the client could analyze the visualizations. After the analysis, the client could draw conclusions about how human emotions are affected when confronted with a business audit.

II. Introduction

A. Goals and Objectives

The objective was to develop a system that allowed our client to analyze Facial Action Coding System emotion data and generate visualizations from this data. In order to achieve this objective, several processes had to come together to form one fully functioning system. These processes included uploading .csv files into a directory to be processed, mapping the .csv data into Spark Dataframes, using Dataframes to organize and normalize the data for simple querying, generating Plot.ly [3] visualizations using these queries, and then presenting these visualizations to view, analyze, and export. Our goal was to provide our client with a simple but effective way to view trends and compare data sets.

B. Statement of Scope

The scope of this project entailed the development of a streamlined process for converting emotion datasets into rich, interactive visualizations. Although our client had a fairly technical background and experience working with command-line applications, it was determined that our software package was best suited to a windowed application. We were provided with 51 anonymized data files from the initial study; however, our project allowed for the addition of data sets, in the event that future iterations of this study are conducted.

C. Client Background

Steven Sheetz is a Professor of Accounting and Information Systems at Virginia Tech and guides our project team. Professor Sheetz' academic focus surrounds database management and systems development. The client conducted a research project in 2009 to determine how human emotions are affected when a subject is confronted with analyzing a business audit.

D. Major Constraints

The largest constraint for this project was determining breadth and depth of features given the timeframe for a deliverable. After preliminary research, there was a lot of different functionality that could be included in the software. Unfortunately, we had to limit our ambitions to a manageable amount of work so that we were able to deliver software that met the minimum requirements with adequate testing.

Another constraint involved the tools used to develop this project. After researching several possibilities, the EmoViz team came to the conclusion that paying for a subscription of certain software would not be viable for the long term success of the EmoViz system. Consequently, strictly open-source alternatives were required which constrained the flexibility of the development stack.

E. Organization of Report

This report contains eleven sections in total, beginning with the Executive Summary. This is followed by an introduction to the project and the requirements from the client. The design and implementation of the main software deliverable is explained in detail, followed by a brief word on testing and evaluation of current results. Next there is a user's manual which provides tutorials for using EmoViz and answers to common questions. The developer's manual explains the structure of the EmoViz program and development environment so future students can expand, maintain, or revise this project. The report concludes with lessons learned throughout the semester, acknowledgements, and finally referenced work.

III. Requirements

A. Use of Apache Spark

The client asked us to use Apache Spark in order to help speed up the data processing and access. Since we had so much data to process and also wanted to support dynamic filtering of the data, we knew we would need a robust tool to help query and organize the data. Spark uses a data structure called a Dataframe which is a distributed collection of data organized into named columns. These named columns can easily be queried and filtered into smaller datasets which could then be used to generate visualizations.

B. Support for Multiple Types of Visualizations

Multiple types of visualizations are required because no single graph can simultaneously convey the same message as a variety of graph types can. The client wants to generate multiple types of graphs because they all provide different benefits in terms of visualization. Some graph types are linear and two dimensional, which can reveal a lot about a set of data but tend to appear fairly bland. Other graphs can contain multiple axes and show the data dynamically. This will help our client understand the data better, and therefore draw better conclusions.

C. Filter and Compare Multiple Data Sets

Complex visualizations can be hard to read, so the client asked to have the ability to filter traces by each emotion. This allows him to focus on changes in individual emotions rather than showing all emotions at once. This also allows for the ability to compare data between multiple students at the same time, which is important in learning how students in different sample groups responded to the video audit.

D. View Precise Changes in Data

Throughout the five minutes of facial emotion data taken, the client expects to see changes in emotion over time. The client especially wishes to see if there are precise changes at specific moments in time where deceptive cues are utilized by the actor in the video. This can help the client understand how students react to possible feelings of happiness, sadness, disgust, anger, and surprise.

E. Local Web Application

We have decided to use a local web application as a user interface, and we are using PHP for it. A local web application will make it easy for our client to use all other tools that are needed for visualization of his data sets. The local web application was decided over an external web server since the client only has one computer in his office that he will use to generate these visualizations. Additionally, this helps to simplify complications over compatibility with different computer systems. An external web application would allow the client to access the system and visualizations from any computer, however this was not a priority for the client.

IV. Design

The EmoViz software package is separated into two levels of abstraction: front-end and back-end. The front-end of the project is the user interface and is implemented as a graphical web application, because our client is most familiar and comfortable working in this type of environment. The back-end of the project is where heavy data processing, Spark cluster management, and system setup occurs. The user provides program inputs on the front-end to be sent for processing on the back-end. Subsequently, outputs from the back-end are displayed for the user to examine, modify, and export on the front-end.

A. Front-End Design

The front-end component of EmoViz is a local web application built with HTML and CSS. Although a command-line application would be perfectly feasible for implementing EmoViz, the decision to have a dedicated front-end allows our client to operate more efficiently and view generated visualizations in a single location. The main feature of the front-end is the viewing window, in which generated visualizations are displayed. Requests sent to the Plot.ly server are returned as HTML files. By default, they are displayed in the browser through a registered Plot.ly account; however, the Plot.ly package offers offline functionality for direct downloads of HTML files. Upon response from the Plot.ly server, HTML files are loaded into the widget and the user is able to immediately see the visualizations. In order to generate a visualization, the user selects FACS .csv files from their filesystem before running a job. Visualizations can be generated for a single data sheet if the client wishes to see how a participant's emotions change and differ through the duration of the video. Additionally, multiple data sheets can plotted on the same visualization, enabling the client to examine how the two groups of his study differ emotionally at certain critical points of the video.

B. Back-End Design

The back-end component of EmoViz is comprised of various Python data processing primitives such as Numpy, Pandas, etc. in combination with the powerful data processing capabilities of Apache Spark. Apache Spark is an open-source cluster computing framework for executing data-intensive jobs in parallel. It is similar to Apache Hadoop in terms of partitioning data across the cluster, but accomplishes faster execution by keeping the data in memory rather than writing it to disk. Although Spark runs through the JVM, it offers the PySpark API which allows direct interfacing with the python implementation for EmoViz.

The Python implementation for EmoViz is a combination of setup, file reading, and plotting scripts. These scripts combine to transform the .csv spreadsheets into the mapped Spark Dataframes in order to generate visualizations. Python does this with ease due to the open source toolkit PySpark and a library called Py4J [9] working together to interface with JVM objects such as Spark Dataframes. Additionally, Spark utilizes two types of methods: transformations and actions. Any function that returns a Dataframe is a transformation, while any function that returns a value is called an action. These methods are easily implemented in Python through

operations such as map() and reduceByKey() for transformations, and take() and reduce() for actions. EmoViz utilizes Spark and Python's compatibility to quickly process the large datasets and generate filtered data on the fly, which is then displayed through each visualization.

V. Implementation

The EmoViz codebase is heavily dependent on conforming to the structure of the APIs and tools it aims to bring together. Though the front-end and back-end were initially written separately, their union provides the user with a simple interface with a powerful engine in the background. EmoViz grants users the power to quickly execute jobs on large sets of FACS data, and then render dense, complex images right on their local machine.

A. Execution of an EmoViz Job

An EmoViz job is the abstraction for the full process of visualizing data. The inputs for a job are one or more FACS data sets and a choice for type of visualization to generate. The output of a job is an interactive data visualization as an HTML element. The job begins with user interaction on the front-end, where a visualization type is selected along with associated data files to visualize. All visualization types can accommodate at least one data sheet, and a subset of these types have the ability to show multiple data sheets on a single plot. When a job is run, a list of parameters corresponding to user selections are sent to the Python back-end, where processing begins. The program reads the list of files to visualize and translates them to a list of paths where the files are located on disk. These paths are used to open and read the CSV files and then convert them to Dataframes. These Dataframes are delegated to an object which corresponds to the visualization type passed from the front-end. When the object is created, the Dataframes are queried for traces of data which are sent to the Plot.ly REST API through a call to the built-in `plotly.plot()` function of the Python package. This call returns a reference to an HTML file, which is written back to disk. Upon successful completion of a job, a PHP script finds the HTML file in the appropriate directory and displays it in the viewing window on the front-end.

B. Data Normalization

Some of the data provided by the client was marked as 'Adjusted' and had timestamps offset by a variable number of seconds. This was due to experimental error in which the video was started a few seconds after the FaceReader software began to track the participant's face. To account for

adjusted files, they would be preprocessed when a job was run and then saved for future use. EmoViz first checks the starting timestamp to determine how much offset there is in the data. Each value in the 'time' column is subsequently reduced by this offset such that readings begin at zero milliseconds.

C. Colors and Emotions

All visualizations represent a number of emotions with respect to timestamps in the video, so to make the visualizations appealing a color was assigned to each emotion tracked by the FaceReader software. The colors correspond to the common feelings of each emotion and are described in Table 1 below.

Emotion	Color	Hexadecimal Value
Neutral	Black	#000000
Happy	Yellow	#F9E03D
Sad	Blue	#1560BD
Angry	Red	#C00000
Surprise	Orange	#FF7F50
Scared	Teal	#4FCFCB
Disgust	Green-Brown	#5B4A14

Table 1 Colors corresponding to each emotion

D. Optimizations

The main optimization technique in EmoViz deals with the processing of data sets. It would be time consuming to convert each CSV file to a Python object every time a job is run, especially if the data is adjusted. When a file is adjusted it is also overwritten on disk so that it does not need to be adjusted again in the future; the original dataset is still preserved on disk so that users may reference original sources if needed. EmoViz also makes use of the Python shelf operation to encode objects to a small index file, similar to how caching works. As a result, loading requested data sets back into memory as Python objects is faster than reading them in full from disk.

VI. Testing & Evaluation

The main source of testing done on the EmoViz platform occurred during the development of the input data validation. In order to ensure the front end does not send an improper command to the

back end, several error flags and checks are used before any exec call is made. This means that any invalid input given by the user is caught and displayed before actually generating any source code errors. This data validation was tested thoroughly by several team members where all possible data file and visualization type combinations were generated. The result of these tests showed that only the proper combination of input selections made an exec call and generated a visualization.

VII. User's Manual

A. Installation & Setup

1. Visit the EmoViz GitHub repository at https://github.com/tsevans/EmoViz
2. Select the green "Clone or download" button as shown in Figure 1.

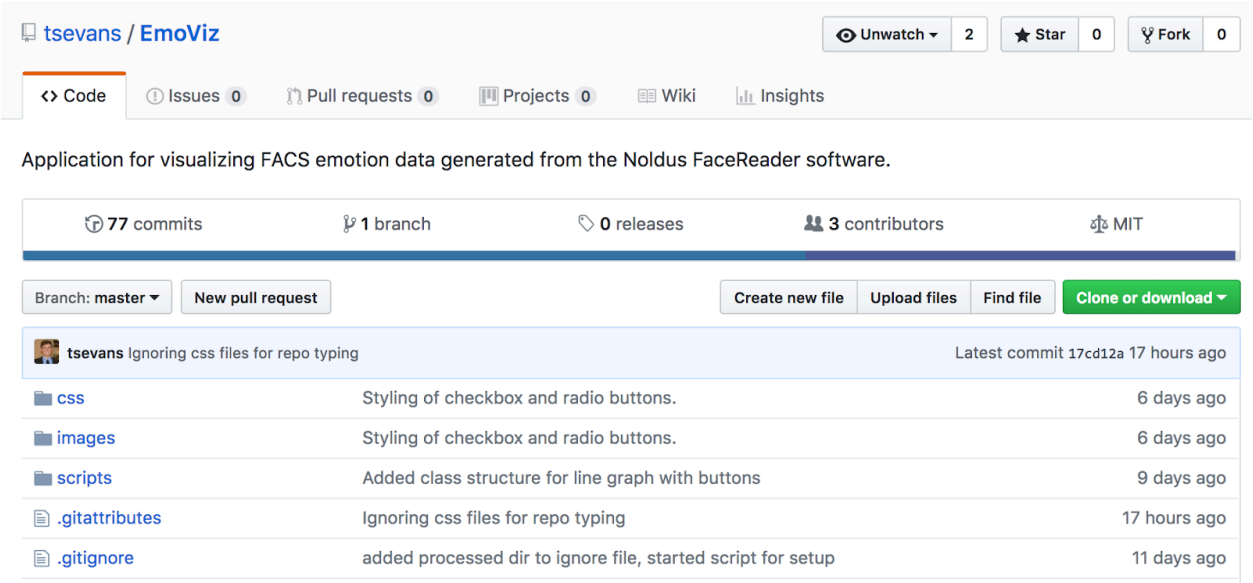

Figure 1 Screenshot of EmoViz GitHub repository showing green Clone or download button

3. Select the "Download ZIP" option.
4. Visit the Downloads folder where EmoViz-master.zip should be located.
5. Double click the EmoViz-master.zip folder in order to unzip the folder.
6. An unzipped EmoViz-master folder will appear.
7. Drag this unzipped folder into the preferred location where EmoViz will reside.
8. In a file browsing window, navigate to the EmoViz directory as chosen above in step 7.

9. Double click on the setup.sh script. This script will create the appropriate folders and start the local PHP built-in web server.

B. Running EmoViz

This section explains how the execution script works by describing what a user would do if the script did not exist/was not working. If section A is completed successfully, then skip section B (this section) and go straight to section C: Uploading New Data.

1. Open the Terminal/Command Line and navigate to the EmoViz directory where the index.php file is located.
2. Type the command:
   ```
   $ php -S localhost:8000
   ```

If successful, you should see the message:

```
"PHP 7.1.14 Development Server started at Tue May 1 20:00:00
2018 Listening on http://localhost:8000
Document root is <EmoViz directory location>"
```

3. This will start PHP's built-in web server
4. Open a web browser and navigate to:
   http://localhost:8000

5. The user should see the EmoViz dashboard as shown in Figure 2.

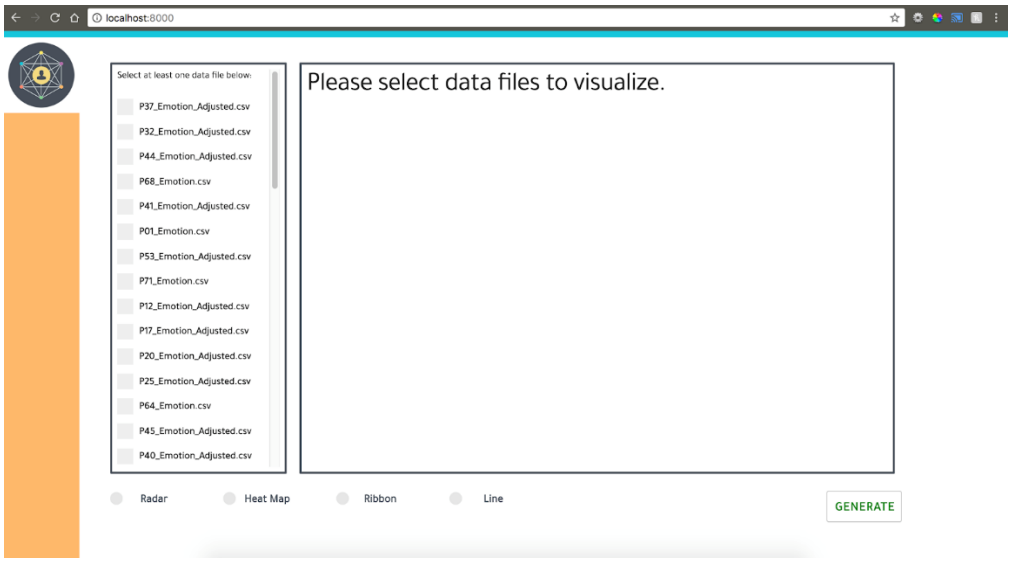

Figure 2 Screenshot of EmoViz Dashboard

6. The EmoViz application is now up and running.

For future reference, the following EmoViz dashboard features will be referenced and depicted in Figure 3.

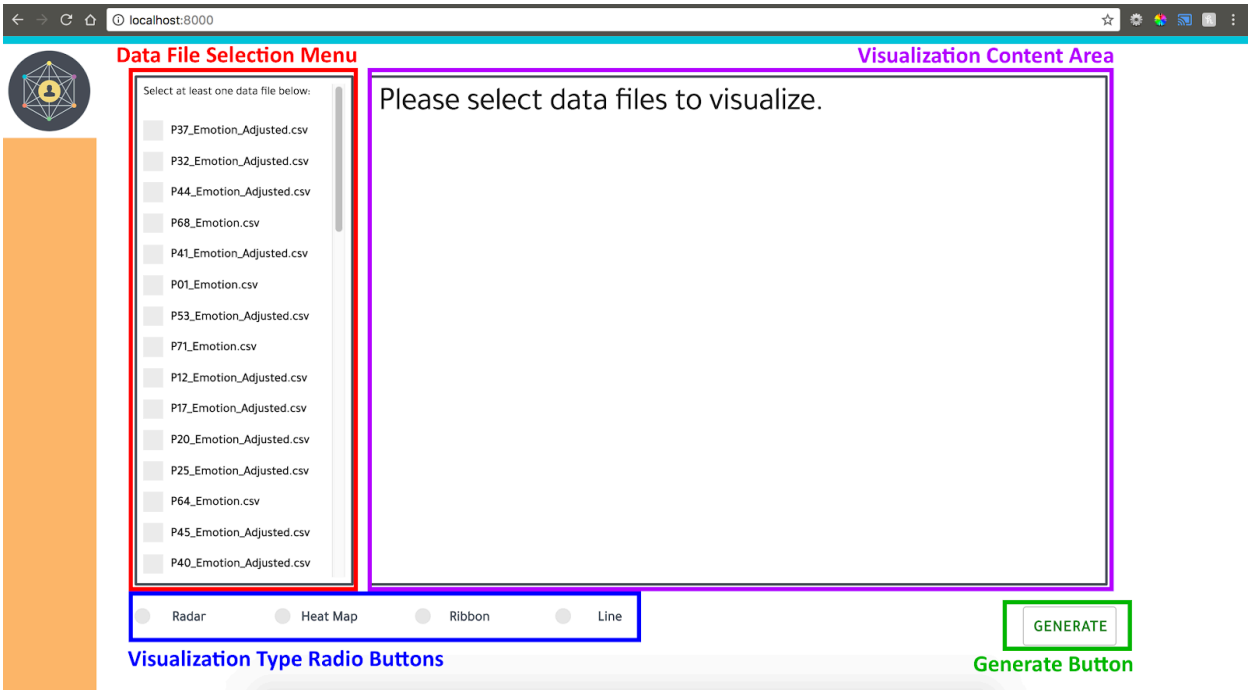

Figure 3 EmoViz Dashboard Reference Names

C. Uploading New Data

If the EmoViz application is not up and running, please follow the steps above in Section A of the User's Manual. Once the EmoViz application is up and running, the user may wish to upload new data files to be visualized. In order to do this, the user must first ensure the files are named consistently with the Face Reader Software standard output. For example, a correctly named file would be titled "EmotionByMillisecond_P01.csv". Place these newly renamed data files into the `data_raw/` folder, which is the location of all unprocessed data files. Go back to the EmoViz application in the web browser and refresh the page. The new data file should be in the Data File Selection Menu. If you are not able to find the file, try scrolling to the bottom of the list.

D. Choosing Visualization Types

1. If the EmoViz application is not up and running, please follow the steps above in section A of the User's Manual.
2. In order to choose a visualization type, the user should first select one or many data files from the Data File Selection Menu. The amount of data files selected depends on the type of visualization desired, as described in Table 2.

Type of Visualization	Number of Data Files Possible
Radar	1, 2, 3, 4, 5
Heat Map	1
Ribbon	1
Line	1, 2

Table 2 Relationship between type of visualization and number of data files that can possibly be selected.

3. Select one of the Visualization Type Radio Buttons by clicking on it.

E. Generating Visualizations

1. If the EmoViz application is not up and running, please follow the steps above in section A of the User's Manual.
2. If Data Files and a Visualization Type have not been selected, please follow the steps above in sections C and D of the User's Manual.
3. Once the above steps are completed, select the Generate Button.
4. The Visualization Content Area should be filled with the user's chosen visualization.

NOTE: If the input was improperly selected, a red error message will display similar to the one shown in Figure 4.

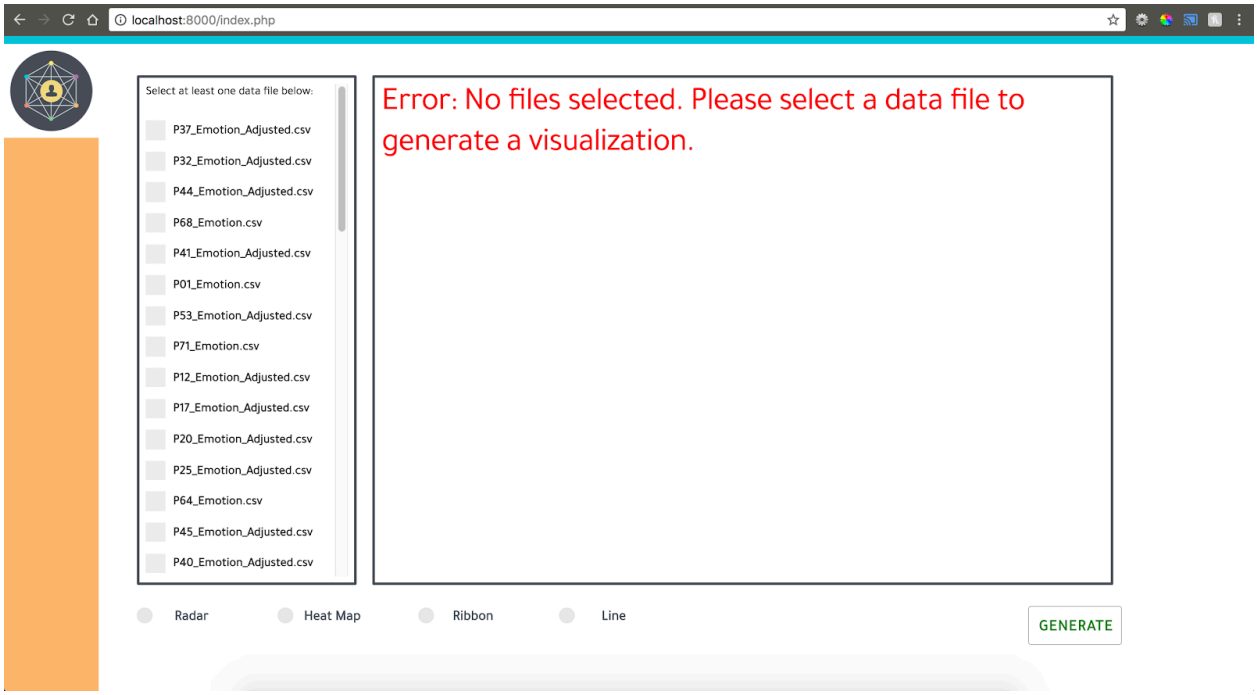

Figure 4 Screenshot of error message generated when invalid input is used.

F. Exporting Visualizations

1. After following all of the steps in the sections above successfully, the Visualization Content Area should be filled with the user's chosen visualization.
2. In order to export the generated visualization, simply select the "Download plot as a .png" button in the top-right hand corner of the Visualization Content Area. This button is denoted by the small camera icon as shown in Figure 5.

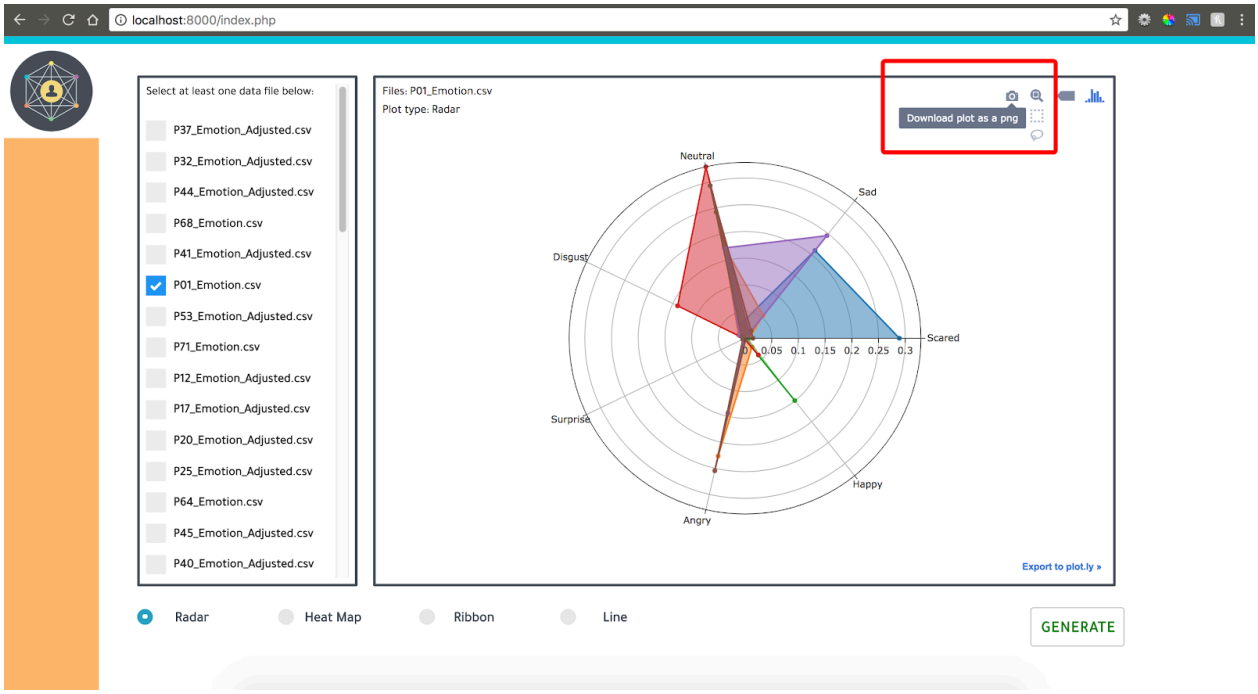

Figure 5 Screenshot of "Download plot as a png" button

VIII. Developer's Manual

This section describes the development and execution environment configurations, code structure for the project, and specific semantics regarding the functionality of the software implementation. Development can be done through any preferred editor, but the team found that it was easiest to use JetBrains PyCharms IDE [7] because of its extensive set of built-in features, including code completion. The community edition of the IDE can be downloaded for free, and if you are a student you can get a free license for the full version on JetBrains' website.

A. Configuring Your Development Environment

1. Hardware Requirements

Development of EmoViz was done primarily on Linux and Linux-based systems, so configurations will be given in this context. The instructions assume a Debian-based distribution, such as Ubuntu, which uses the 'apt' package manager. If your computer is running a different Linux-based operating system, you may use the default package manager. For development on Windows operating systems, packages can be downloaded from their distribution websites and

installed manually with the provided installer. A Linux-based system is helpful for working with EmoViz but not required.

2. Software Requirements

The following instructions can be referenced if the user wishes to install necessary packages from the command line.

Begin by cloning the git repository for EmoViz, located at https://github.com/tsevans/EmoViz. This is accomplished by executing the following commands:

```
$ cd /dir/where/you/want/to/install/emoviz
$ git clone https://github.com/tsevans/EmoViz.git
```

EmoViz is written in Python version 2.7.12 [4], which can be installed with the following command:

```
$ sudo apt-get install python
```

To ensure that it was installed properly, enter the following command:

```
$ python --version
```

Plot.ly is the primary graphing API used for generating visualizations. If you wish to generate animated plots, you must register a free account at https://plot.ly/accounts/login/?action=login. An account is not required to generate static plots. Plot.ly is one of many Python modules required to run EmoViz. These dependencies are acquired via the pip package manager, which can be installed with the following command:

```
$ sudo apt-get install python-pip
```

Once pip is installed, run the following commands to install modules:

```
$ pip install plotly [3]
$ pip install pyspark [5]
$ pip install pandas [6]
```

Users must now install Apache Spark version 2.2.1. These files can be downloaded for free from the Apache website. Most of the setup should be handled by the provided installer, but it is important that users set the SPARK_HOME environment variable to the location where Spark was installed. By default, Spark should be installed in the /usr/bin directory. You can set the environment variable with the command:

```
$ export SPARK_HOME=/usr/bin/spark
```

B. Project Structure

1. File Listing

The primary file called from the front-end script is main.py, and it reads parameters for execution and delegates the appropriate methods. The scripts filereader.py, csv_processor.py, and time_utils.py are used to read and process files. Finally, graph_static_plotly.py is where calls to the Plot.ly API are made and execution of the job ends.

3. Plot.ly Library

The plotly offline API allows for the writing of richly linked and annotated visualizations to HTML files. Plotly graphs tend to consist of three parts: traces, layouts, and figures. Traces are subsets of a Dataframe and contain data for a single aspect of a plot, such as a line in a line graph or category in a histogram. Traces are implemented as a Python dictionary, where keys are various attributes such as the color, name, and visual properties of the trace, and the values are the values for those properties as described in the plotly documentation. For a single dataset, there would be a single trace for each of the seven emotions. For multiple datasets, there would be a trace for each dataset with subtraces for the emotions within each Dataframe. Multiple traces are placed in a Python list, named ‘data’ by standard, to then be described by the layout. The layout of a plot determines the properties for how traces are displayed. These are also implemented as a Python dictionary, where the keys are properties of the graph such as size, axis labels, color, etc. and the values are the values for each item. Figures bring together the list of traces and the layout for a visualization by storing each of them as an entry in a graph_objects dict. The figure is what is directly plotted and various configurations can be changed such as online vs. offline plotting as well as the name of the file that is returned.

C. Adding New Visualization Types

The object-oriented structure of the code allows for new visualization types to be easily added. A visualization is represented by name as a Python class in the file graph_static_plotly.py. To add a new type, find the reference for it in the Plot.ly documentation to see how it works, and then simply copy the format of an existing class. The classes use static methods so they can be called directly. For example, the RadarChart class contains the methods generate(), build_data_traces(),

and build_layout(). The data is passed into the traces, which are passed into the layout, and then a call to the Plot.ly API is made and the method returns a handle to the HTML file response.

IX. Lessons Learned

This section will discuss the progress made throughout the project including the timeline of deliverables, problems met along the way, solutions to those problems, and then lessons learned from overcoming obstacles.

At the beginning of the project, the EmoViz team was set on using a tool called DataBricks [11] to help analyze data and generate visualizations. This was a tool the client was familiar with and urged the team to research further. After conducting research, however, the team found that the free version of DataBricks would not be suitable for the scope of the project and that something else would be needed. DataBricks is simply the official maintenance company for Apache Spark and provides small improvements in speed. Not only were the optimizations negligible for this use case, but the free version did not allow any version control integration which would have heavily impeded workflow of the team. Multiple software stacks were compared and ultimately the team decided to use PySpark for two reasons: Python's native bindings to Spark would allow us to maintain fast data processing and the team was most familiar with Python development compared to C#, Scala, and JavaScript.

A. Problems and Solutions

The following paragraphs describe some of the problems encountered throughout this project as well as how we approached them to come up with solutions.

Early on the research process, we had trouble in deciding on which software stack to use for development of the EmoViz project. Four main options were available: Python, Java, C# and JavaScript; each came with its own advantages and drawbacks. Our whole team was familiar with Java, but we could not find bindings to Spark which didn't involve the use of Scala. C# would have made for easy development of a Windows application, but it does not have native bindings to Spark and only has an unofficial mapping called Mobius [12] which was not as well documented as Spark. JavaScript has many libraries for visualization, but only one member of the team had worked with it before and there was no need for this application to be written exclusively as a web application. After weighing pros and cons of each option, we decided to go

with Python since it is directly compatible with Spark and easy for all members of a team to learn.

One of the biggest challenges when working with Apache Spark is setting it up on a computer. Our team spent a great deal of time setting this up so we wanted to make sure our client wouldn't have to deal with the same issues. The solution for this was to write many setup scripts.

Towards the end of the semester there was limited communication between the client and the team through no fault on either side. This lack of in-person meetings and email correspondence meant some questions were unable to be answered in a timely manner as the team wrapped up the project. This limited the implementation of some features such as unique epoch analysis.

X. Acknowledgements

Client: Steven D. Sheetz, Professor of Accounting and Information Systems
Contact Information: sheetz@vt.edu

Capstone Professor: Dr. Edward A. Fox, Professor of Computer Science
Contact Information: fox@vt.edu

XI. References

1. Apache Spark Foundation. Apache Spark Language Reference. Available at https://spark.apache.org/documentation.html [Accessed: 02-May-2018]
2. Xin, R., Armbrust, M., & Liu, D. (2018, February 06). Introducing DataFrames in Apache Spark for Large Scale Data Science. Retrieved March 26, 2018, from https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html [Accessed: 30-April-2018]
3. Plotly Technologies Inc. Collaborative data science. Plotly Technologies Inc. Montréal, QC. 2015. Available at https://plot.ly [Accessed: 01-May-2018]
4. Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org [Accessed: 01-May-2018]
5. Apache Spark Foundation. Apache Spark Python API Reference, version 1.6.7. Available at https://spark.apache.org/docs/latest/api/python/index.html [Accessed: 30-April-2018]

6. Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010) http://conference.scipy.org/proceedings/scipy2010/mckinney.html [Accessed: 30-April-2018]
7. Jet Brains. PyCharm, version 2017.3.3. Available at https://www.jetbrains.com/pycharm/ [Accessed: 01-May-2018]
8. International Organization for Standardization. Date and time format - ISO 8601. Available at https://www.iso.org/iso-8601-date-and-time-format.html [Accessed: 01-May-2018]
9. Python Software Foundation. Python Py4j API. https://www.py4j.org/ [Accessed: 01-May-2018]
10. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing." Available: https://cs.stanford.edu/~matei/papers/2012/nsdi_spark.pdf [Accessed: 02-May-2018].
11. "Making Big Data Simple." *Databricks*, databricks.com/. [Accessed: 02-May-2018]
12. Microsoft. "Microsoft/Mobius." *GitHub*, 8 Apr. 2017, github.com/Microsoft/Mobius. [Accessed: 02-May-2018]