# Lit Review

Data Visualisation is key to understanding large sets of data. Graphically representing data allows the human mind to better comprehend abstracted views and makes it easier to identify trends, patterns and anomalies not easily identifiable when looking at data in more traditional forms such as spreadsheets.

# For Group Report UI:

## List of Tutorials

Installing Node.js on a Raspberry-PI: <http://thisdavej.com/beginners-guide-to-installing-node-js-on-a-raspberry-pi/>

Node.js File System: <http://www.tutorialspoint.com/nodejs/nodejs_file_system.htm>

Node.js Web Module: <http://www.tutorialspoint.com/nodejs/nodejs_web_module.htm>

Node.js Express Framework: <http://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm>

Node.js Restful API: <http://www.tutorialspoint.com/nodejs/nodejs_restful_api.htm>

Node.js – Documentation: <https://nodejs.org/dist/latest-v4.x/docs/api/synopsis.html>

D3 – Let’s Make a Bar Chart (Parts I, II & III): <https://bost.ocks.org/mike/bar/>

D3 – How Selections Work: <https://bost.ocks.org/mike/selection/>

D3 – Scot Murry’s D3 Tutorials – Fundamentals: <http://alignedleft.com/tutorials/d3/fundamentals>

## List of References to Technologies used

D3: <https://d3js.org/>

D3 – Examples: <https://github.com/d3/d3/wiki/Gallery>

D3 – Streamgraph Example: <http://bl.ocks.org/mbostock/4060954>

D3 – Calendar View Example: <http://bl.ocks.org/mbostock/4063318>

D3 – Day/Hour Heat map Example: <http://bl.ocks.org/tjdecke/5558084>

D3 – Show Reel (For Streamgraph) Example: <http://bl.ocks.org/mbostock/1256572>

D3 Queue: https://github.com/d3/d3-queue

Word Cloud: <https://github.com/timdream/wordcloud2.js>

Node.js 4.4.7: <https://nodejs.org/en/>

Bootstrap: <http://getbootstrap.com/>

Raspberry-PI: <https://www.raspberrypi.org/>

Raspbain: <https://www.raspbian.org/>

JQuery: <https://jquery.com/>

## How to setup the solution

The Twit-Con-Pro Data Visualisation component is a stand-alone web server component using Node.js combined with the Express Framework. The solution only depends on Node.js to be installed and can run on any platform. It was tested on a Windows 8 Desktop and Raspberry-PI 3 running a Raspbian OS.

1. Install Node.js from <https://nodejs.org/en/download/>
2. Getting the Twit-Con-Pro solution:
   1. Option1 – Checkout The Git Project: <https://github.com/garethstephenson/ELEN7046_Group2_2016.git>
   2. Option2 – Download the Git Project from: <https://github.com/garethstephenson/ELEN7046_Group2_2016>

### Running the solution

1. For Windows OSes:
   1. Open Command Prompt
2. For Linux and Mac Oses (Including Raspberry-PI):
   1. Open up a Terminal Window locally. Alternatively, open a remote SSH terminal session to the machine hosting Node.js and the solution.
3. Navigate the {solution}\visual folder
4. Execute the following command: “node main.js”
5. The following line of text should appear: “Twit-Con-Pro started on http://:::8081”

### Changing the Port

1. Open up the main.js located in the {solution}\visual folder in any text based editor.
2. Navigate to line 50 “var server = app.listen(8081, function () {”
3. Change the first parameter of the listen function, by default set to ‘8081’ to the desired port.
4. Save the main.js file
5. Follow the “Running the solution” section and test the new port setting

## Key Risks/Challenges

### How the team dealt with technical/project risks

As early as the first team meeting, the team identified core risks and formulated an approach around these risks. The concept, borrowed from the Agile Principals, was ‘early failure’. In order to understand if something is going to work, build it and use it. The main approach was to develop POC (Proof of Concept) projects too deal with specific concepts. Within the first week after the meeting, all main concepts had a POC prototype solution behind it and these POC’s became the basis of the final product.

### The Raspberry-PI infrastructure

Raspberry-PI’s limitations around processing and technologies – Initially unclear of the processing capability of the PIs, a risk was raised as to if the Raspberry-PI’s could handle the processing and visualisation of data. This risk was early mitigated by running various POCs on the PI infrastructure.

To further mitigate this risk, a decision was made to use a web based visualisation solution in order to leverage of a client device to share the processing load.

### Collaboration

As the team was not co-located challenges around communication and general collaboration was raised before the first team meeting.

The team made use of the following, online collaboration tools to meet their needs:

* WhatsApp – to initiate any discussion that required an immediate response
* Slack – for knowledge sharing and lengthy, more general discussions
* Trello – to manage collaborative tasks and to provide an understanding of the scope of work
* GitHub – Source Control and Content Management
* Google Hangouts – For collaborative, interactive knowledge sharing

To ensure frequent, face-to-face communication and ensuring the project moved forward, the team decided to meet every Sunday for the duration of the project.

## Assumptions and Constraints

### Assumptions

* Web-client devices can render the graphical elements
* Datasets produced by the data processing component will be small enough to easily be transmitted over the internet to web-client browsers without error

### Technical Constraints

* With the data visualisation component being web based, this component was constraint to use HTML CSS and JavaScript technologies.
* Required to run on commodity hardware and open-source software.

### Project Constraints

* Time – The team could only meet over weekends

## Design Decisions

### Solution Conceptual

Major solution design concepts include cohesion and coupling, portability and extensibility. To support this, the solution was divided into three components, Data Collection, Processing and Visualisation. JSON files were chosen to transfer data between the components. None of the components are aware of the internal workings or composition of the other components and are only dependant to receive or produce data in the agreed formats.

### Visual Component – Conceptual

The Visual component implements the MVC pattern to separate the View and controlling logic from the Data. This allows for the data components to be develop separately from the Views. Between the Views and Data Models where Controllers responsible to load the views based on the set configuration and provide the views with the required data model.

The Data Visualisation component was designed to be agnostic of the content and allows for comparison of categories in a topic agnostic of the topic or categories.

All charts were design to be scalable. All graphs will render as the view point change allowing for a pleasant user experience.

### Visual Component – Technical

The Node.js Express Framework was chosen as the server-side technology as it allows for serving both static and dynamic content with little coding necessary.

Bootstrap was chosen to allow for a responsive UI. This allowed for creating a response base without having to specifically code for it.

JQuery was chosen to aid cross browser support.

Twig was chosen as a HTML templating tool to allow the controllers to embed data when rending the views.

The D3 JavaScript charting library was chosen as the base to develop the majority of the charts as it provides the ability to visualise data by attaching the provided data to the DOM (Document Object Model).

## Diagram Description

This diagram depicts both the core framework components (on the left) and the graph views (on the right) structured according to the MVC pattern modularising the sub components into Views, Controllers and Data Models.

Controllers are made possible using the Node.js Express framework’s Router component. This allows for a combination of paths (also known as routes) and HTTP Verbs (GET, POST, PUT…) to be assigned to specific Controller methods.

As an example, in the right, The Index Controller receives a request from the client web browser to load the route/home page by navigating the browser to http://localhost:8081. This intern is interpreted by the Router component as a GET request on the ‘/’ route. In the Index Controller (index.js) this route is configured to call a function that will fetch the Index Model (index.json) and render the index.html.twig template to produce the Index Page.