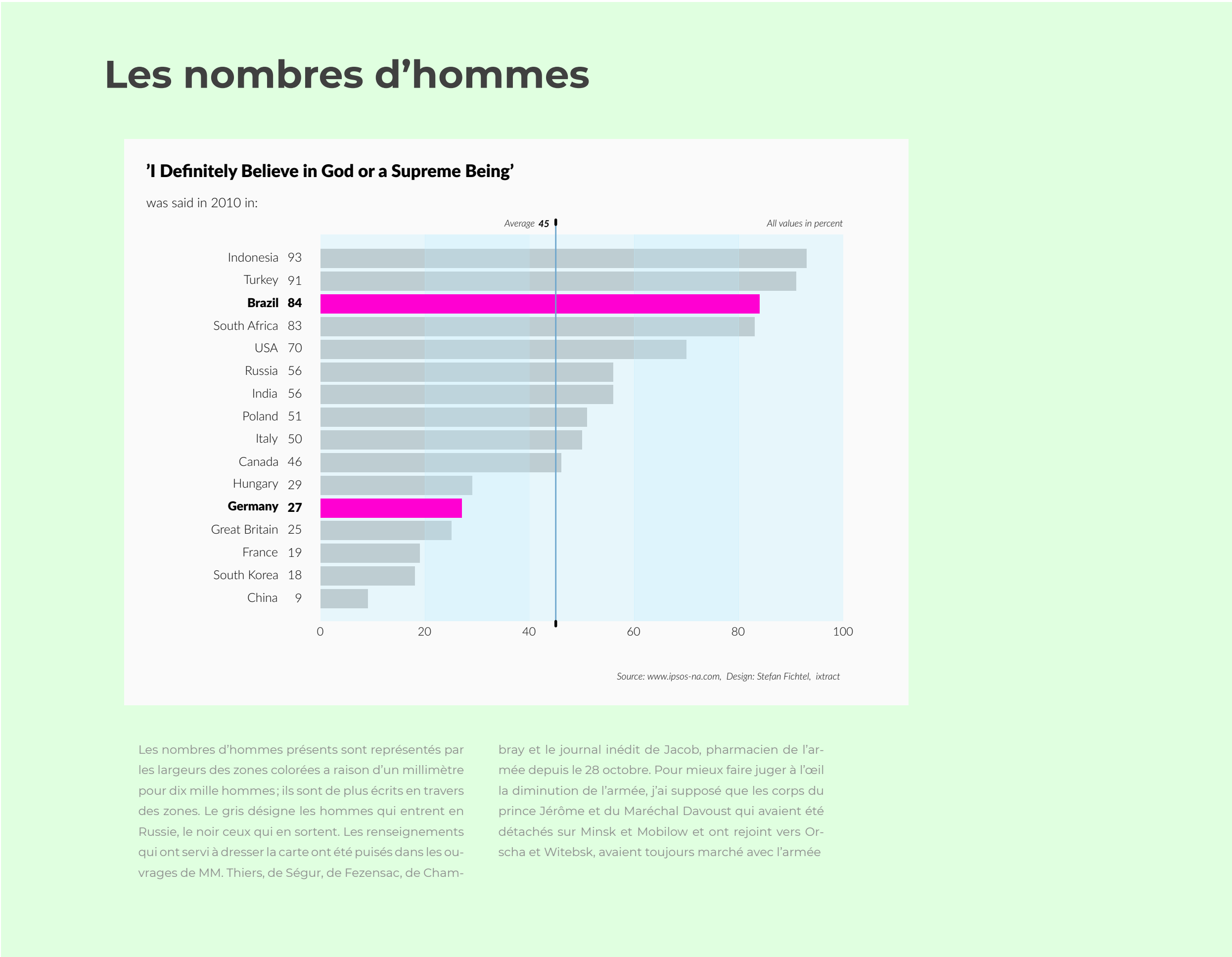
f

Can Web Technologies Help R Generate Print Quality Graphics?



NOTE: Image to be updated with better example

CompSci 791: Research Paper [Working version]

Kane Cullimore

November 16, 2020

Abstract

Abstract paragraph to cover the following main topics:

- Introduction and problem description
- layoutEngine and its limitations
- RSelenium backend
- NodeJS backend
- Future suggestions

Document Structure

This report is organized in the following way:

- A problem definition is given to explain the core functionality the layoutEngine library is provided
- A brief explanation of how the layoutEngine is intended to be used along with its current shortcomings is covered
- The RSelenium layoutEngine back-end is introduced
- The NodeJS layoutEngine back-end is introduced
- A comparison of each back-end is given to identify areas of advancement
- Finally, a review of the overall layoutEngine approach is given and compared to some existing R extensions

Temporary Notes

The planned size of document:

- Background = 10 pages
- RSelenium Backend = 10 pages
- NodeJS Backend = 10 pages
- Comparisons and Wrap-up = 10 pages

Presentation:

- Time 10 mins with 4 mins Qs
- Slide: Background and What problem it solves
- Quick demo (with backup slide)
- Slide: Diagram of lE system
- Slide: Identification of lE limitations
- Slide: New development
- Demo of latest and greatest examples
- Slide: Qs

Contents

1 Introduction

The **R programming language** is a popular open-source statistical analysis tool. The language has many libraries that support sophisticated statistical techniques. Many of these rely on graphical output to communicate results. A strong appeal of this programming language is the ease at which its **core graphics system** (TODO: add glossary item) generates graphical output that is both accurate and effective at communicating this type of information.

Few open-source alternatives offer an equivalent set of sophisticated statistical methods married with a flexible and powerful graphics system. The Python programming language is a strong competitor. However, its focus is more as general purpose programming language with fewer specialized statistics libraries that generate these types of graphic types with equivalent rigor.

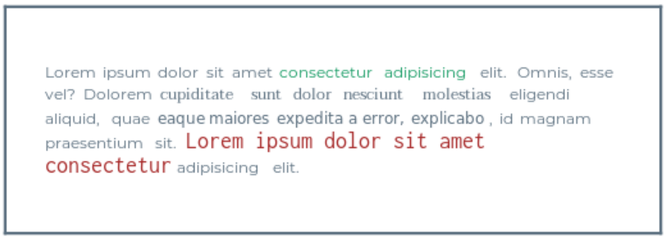
The number of specialized applications R is used within has grown with its popularity. One such use-case is the incorporation of the statistical graphic within published articles. While the raw dots and dashes used to generate these graphics is of sufficient digital quality there are several fundamental publishing requirements which are not supported.

2 Problem Description

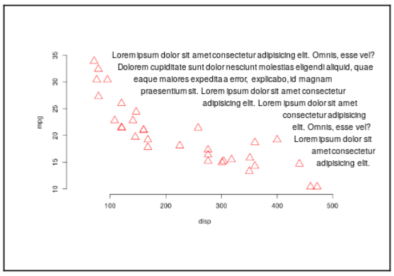
The publishing industry has a long history dating back to the 15^{th} century when movable type printing was first invented. As the industry integrated within digital media it has brought with it a system of long-standing expectations for content. As a result, digital publications often have a myriad of requirements for graphics which are referred to in this report as **print quality graphics**.

Several of the more important requirements include **(1)** writing system and font specifications, **(2)** document layout and typesetting, **(3)** sophisticated content rendering, and **(4)** control over output resolution and file format.

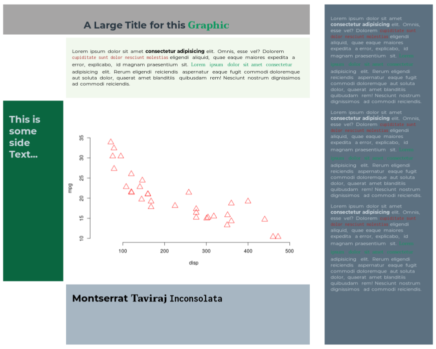
R users have a powerful tool for generating statistics based graphics but they will struggle to support many of these requirements. Several examples are presented below which show features which are not currently possible with the core R graphics system.



Example 1: Using multiple font types in the same graphical component

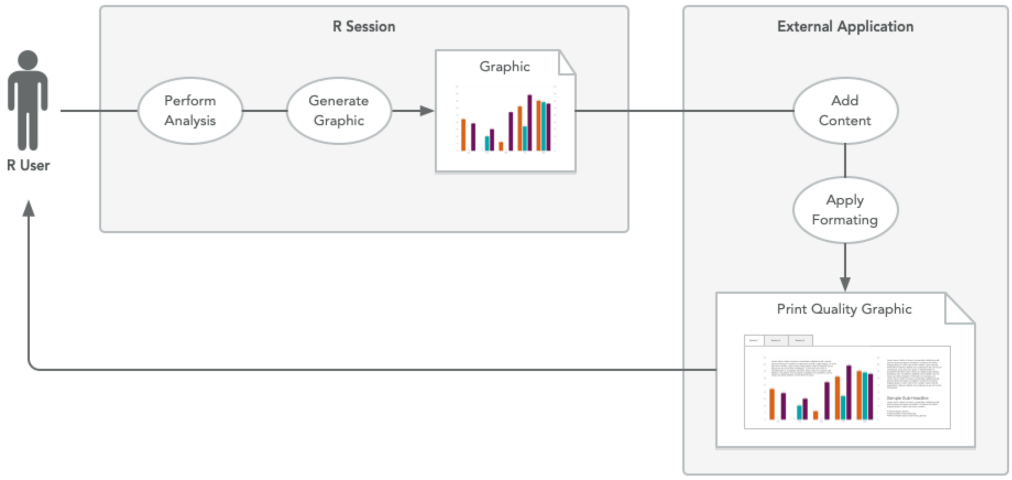


Example 2: Embedded graphics with wrapped text



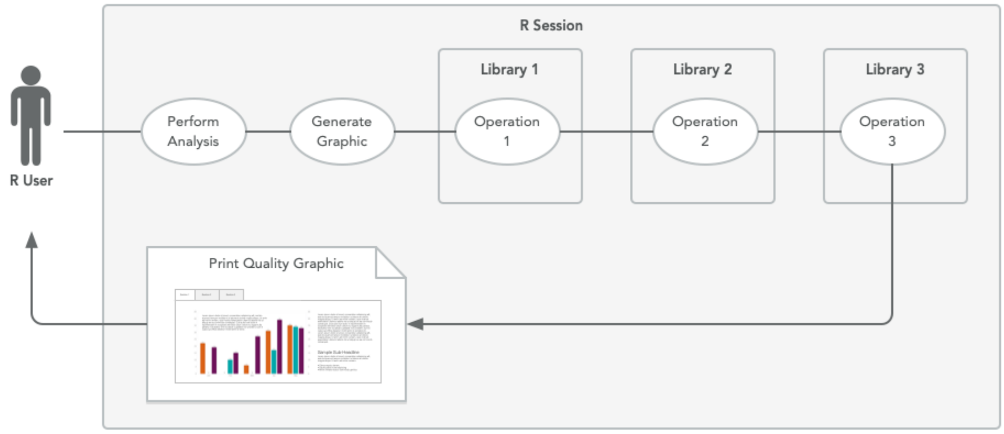
Example 3: Complex layouts of text and graph components

An existing solution to produce print quality graphics is to modify the R graphical output with external tools such as LaTeX or *Adobe Illustrator*. The user must either be proficient in both environments or have a specialist available to help.



External Applications Used to Produce Print Quality Graphics

Another solution would rely on several existing R libraries. Such libraries offer bits of this functionality ad hoc. The user would remain within the R ecosystem but might need several libraries depending on the publishing requirements. This modular feature-set composition is the *standard R approach* to extend its functionality.



Standard R Approach to Produce Print Quality Graphics

This research explores an alternative approach, referred to as the *layoutEngine approach*, which offers a general purpose solution to generate print quality graphics from within the R ecosystem. This approach is encapsulated by the layoutEngine R library which incorporates web technologies to extend the functionality of the R graphics system.

The motivation of this approach is based on the understanding that web technologies and modern browsers have long supported the special needs of the publishing industry. As a result, there is much functionality to be gained by leveraging this technology. The layoutEngine intends to create access to a full-suite of industry leading functionality. This hinges on whether it can successfully utilize the browsers graphics system in an easy to use R package with accessible dependencies.

Please note that this research does not address the relative performance and functionality difference between the *layoutEngine approach* and the *standard R approach*. Rather, it *first* explores the efficacy of this *layoutEngine approach*, and *second*, attempts to improve upon the implementation of the existing layoutEngine library to address several existing limitations.

3 The layoutEngine R Library

The intent of the **layoutEngine**[TODO: Add to References] R library is to extend R's graphical system by adopting functionality available in web technologies. Its core functionality is to act as an interface between R and a web browser to provide access to the rich feature set. The library is available to review via the layoutEngine repository however it is still in development and not yet available in **CRAN**.

3.1 The Standard R Approach

The *standard R approach* to its extend functionality is paramount to the success of open-source programming languages. Available libraries are freely shared within the community which can be loaded into the scope of an environment to gain specific functionality. If a broader feature set is needed then several libraries are loaded which acts as a modular system. There are many advantages to this approach and it is a key reason why these languages and user communities have thrived.

Several **available CRAN packages** offer functionality which meet some publishing requirements. Many of these libraries are well executed and perform admirably. The *ggtext* package enables multiple font types to be specified in the same graphic. The *patchwork* package offers a similar functionality specifically for arranging several ggplots with claims of increased simplicity and flexibility. In addition, the base package *grid* has a `layout` function which creates a `Grid` layout object that enables plots from different systems to be arranged together. Many other libraries extend R towards the realm of print quality graphics but no general purpose solution exists at this point in time.[TODO: add all to References] Together they establish that a need does exists for this type of extended functionality for R users.

NOTE: Also references Paul's list per HTML Rendering article

The standard R approach might eventually succeed in offering a general purpose solution to generate print quality graphics. However, several difficulties exist would first need to be overcome. First, the publishing requirements represent an extensive set of functionality. In addition, the variety of graphical output this must operate with is also large. As a result, the task of coordinating a suite of purpose-built libraries that is flexible enough to cover all scenarios would be significant. This is both difficult from both a developer and user perspective due to the large number of functions and objects to handle.

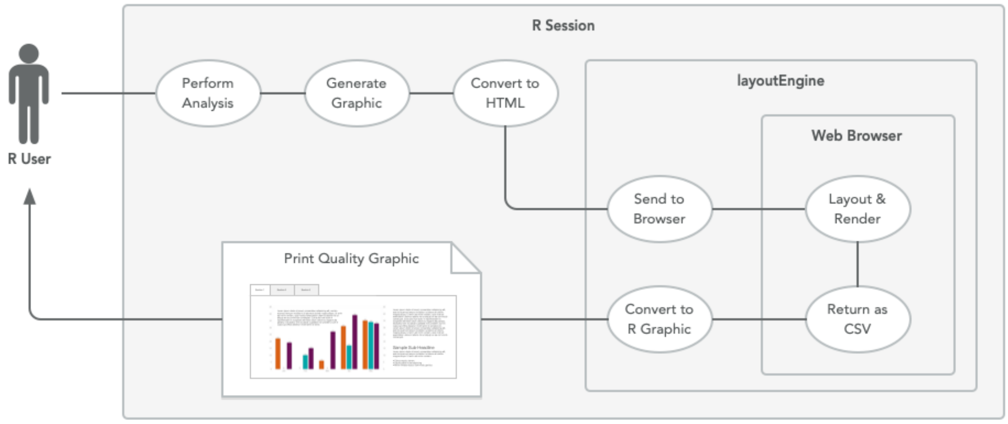
3.2 The layoutEngine Approach

The *layoutEngine approach* differs with the way it extends the functionality of R. It acts as an interface between the graphics system in R and another technology ecosystem; the web browser. This general purpose solution attempts to adopt from an industry that has a long history supporting publishing requirements so as to avoid reinventing the wheel.

This approach bypasses the need to build a complex general purpose solution from the ground up and instead employs existing tools. These tools are part of the web-based languages (HTML, CSS and JavaScript) and R packages which can be used generate HTML from R objects. The table below identifies some of these technologies which can be used to generate a print quality graphic.

Functionality	**Technology**
Layout and Typesetting	HTML elements styled with CSS Grid or Flexbox
Font Type and Styling	CSS @font-face with traditional CSS styles
Text Wrapping	Standard HTM, CSS shape-outside
R to HTML Conversion	xtable, formattable, htmltools, gridSVG, rmarkdown, etc...

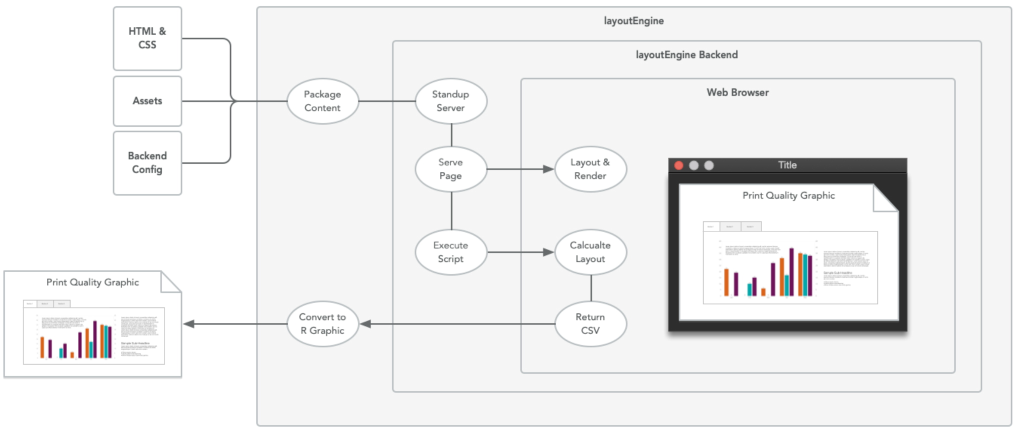
The simplified diagram below demonstrates how the layoutEngine is used for a general use-case. The graphic is first defined as an R object. This is converted to HTML where some additional definition could be added. The browser readable definition is transferred and loaded into a web browser. The browser's layout and rendering engine generates the desired graphic in the browser window. A JavaScript function is then executed to calculate the position of each component on the page. This data is then sent back to R in CSV format where the layoutEngine will convert it to a R readable graphic object. This can then be displayed in the R graphics window or sent directly to an image file.



Produce Print Quality Graphics Using the layoutEngine

3.3 Solution Design

The layoutEngine solution design is comprised of two components as shown in the figure blow. The **layoutEngine** is configured to interface with one of several available **layoutEnging Backends**[TODO: Add to References]. The layoutEngine acts as the interface for the R user while the layoutEngine Backend is the interface between R and the web browser. This solution was chosen as much of the complexity exists in the Backend. This abstracts much of the complexity away from the user and allows various Backend designs to be swapped out with little impact to the main layoutEngine interface.



layoutEngine Solution Design

3.3.1 layoutEngine Package

The layoutEngine is responsible for handling all the web-based data to send to the layoutEngine Backend. It must also take the returned data from the Backend and render this within the R graphics display. A major piece of this involves locating the correct fonts within the host machine to ensure the browser calculations are made using the exact font specifications.

Requirements:

- Interface for R Users to execute commands and configure the Backend
- Handling of web page content (HTML, CSS, Fonts, Assets)

- Call of Backend primary interface
- Handle returned layout calculation
- Rendering of new content within R Graphics Display

Challenges:

- Ensuring ease-of-use for R Users by accepting various web-based content formats
- Cross-platform access to system fonts
- Support conversion of web-based graphic data to the R graphics display

3.3.2 layoutEngine Backend Package

The layoutEngine Backend is where much of complexity exists required to interface with a web browser. Since there are many possible ways to implement this functionality it is contained within a separate R package. The main purpose of the Backend is to serve the R graphic to a browser, execute the layout calculation script and return data to the layoutEngine. The Backend contains the primary challenge of the *layoutEngine Approach*.

Requirements:

- Locate and manage a modern web browser session
- Send and receive data between a R session and web browser
- Query and modify the web-page DOM [TODO: Add glossary term]

Challenges:

- Variability in Host Machine
  - Cross-platform system calls (macOS, Windows and Linux)
  - System and R dependencies

There are three layoutEngine backends available for use with the layoutEngine at the time of this research. These backends successfully demonstrate the viability of this approach.

3.4 Benefits

3.4.1 General

- Integration of the image within other content that is accessible programmatically
- Not just an embedded graphic
- HTML knows about the interior of the R graphic and is NOT just a dumb blob
- Access to a huge amount of functionality of the web technology stack
- Web Tech is a vibrant community
- Browsers are extremely sophisticated and competitively being enhanced each year
- Take advantage of the large variety of packages and methods that currently generaate HTML
  - Knitr (markdown?)
  - xtable
  - htmltools (RStudio `https://www.stat.auckland.ac.nz/~paul/Reports/HTML/layoutengine/layoutengine.html#pkg:htmltools`)
  - Others?
- layoutEngine Backends

3.4.2 DOM Backend

- [TODO]
- Based on Paul's DOM package
- Live visual feedback for debugging, reviewing output
- Access to latest web browser and therefore latest HTML, CSS and JS specs

3.4.3 PhantomJS Backend

- [TODO]
- Simple, lightweight, few dependencies
- Per Ref
  - Based on WebKit browser engine (Apple from Chrome)
  - Does not require a GUI and performs layout off-screen

3.4.4 CSSBox Backend

- Per Ref
  - Based on CSSBox Java library
  - Generates HTML layout information directly (i.e. standalone HTML layout engine)
  - Generates information for every line of text after layout which is better than most web browsers
- [TODO: get some feedback on this from Paul]

3.5 Limitations

3.5.1 General

- Have to learn and write in HTML/CSS/JS
- Security layer around using a browser
- System dependencies across all OS is trouble
- Font managing software customized per OS
- Per Ref
  - Cairo-based R graphics devices (and pdf & postscript)
  - Matching or converting X11 fonts for the X11 device to fonts the layout backends can use would be hard
  - Support for native Windows and MacOS graphics devices
  - Smallish list of CSS that is support in layoutEngine currently do nothing when pushed back to R from the backend
  - Issues with hyphens in CSS (as string variables in R)
  - Pixel resolution compatibility (resolution of graphics device should be set to 96 dpi)
  - Not a fast process (rendering HTMl) => speed cost for expanded functionality

3.5.2 DOM Backend

- [TODO]

- Default browser opens every call
- Per Ref
  - See article about managing font types

3.5.3 PhantomJS Backend

- [TODO]
- Lack of visual feedback
- No longer developed so will eventually lose support
- Based on older WebKit engine so behind on HTML and CSS specs

3.5.4 CSSBox Backend

- Per Ref
  - Have to keep track of levels of accuracy based on what device HTML will be rendered on
  - Lags browsers support of web standards (modern CSS specifically)
- [TODO]

3.5.5 Summary

The following list summarizes the desired requirements for a layoutEngine backend:

- **Cross Platform:** Compatibility with all major operating systems (Linux, MacOS and Windows)
- **Simple Dependencies:** Simple installation with few dependencies that are consistent across platforms
- **Industry Support:** Robust industry support of any incorporated tools, technologies and standards
- **Modern Web Standards:** Support for modern web standards including HTML, CSS and JavaScript
- **Visual Feedback:** Ability to view graphics within live browser

Limitation	DOM	PhantomJS	CSSBox
Cross Platform	Low	Low	Low
Simple Dependencies	Low	Low	Low
Industry Support	Low	None	Low
Modern Web Standards	Med	Low	Low
Visual Feedback	High	None	?

4 layoutEngine Development

4.1 Objectives

The viability of the layoutEngine approach is still being explored and it is the layoutEngine backend where the majority of the limitations reside. There are several existing backends ([TODO:] see reference) however each has certain limitations that must be rectified before community adoption is possible.

This report introduces two newly developed layoutEngine backends which attempt to address the limitations of the existing designs. One relies on a **Selenium** server hosted within a **Docker container** container. The second is a custom **NodeJS** server also hosted within a Docker container.

4.2 layoutEngine and Backend Interfaces

Development for a layoutEngine Backend requires meeting the interface requirements for the layoutEngine package itself. In the example below, the Backend is specified and then the layoutEngine **`grid.html()`** function is called with the specified arguments. Since the Backend is isolated, this *R User interface* remains the same for all Backends.

```r
## Load libraries for layoutEngine and Backend and set the Backend
library(layoutEngine)
library(layoutEngineRSelenium)
options(layoutEngine.backend=RSeleniumEngine)
## Basic example supplying HTML, FONTS, CSS, ASSETS
HTML <- c('<div class="content">',
          '<h1>This is some simple text</h1>',
          '</div>')
FONTS <- cssFontFamily("FreeMono")
CSS <- '.content {font-family: FreeMono; border: solid; width: 800px; height: 100px; display: flex; justify-content:
↪  right;}'
## Call layoutEngine
png("essential-example.png", width=800, height=100)
grid.html(html=HTML, fonts=FONTS, css=CSS)
dev.off()
```

This is some simple text

The *Backend interface* is contained within the engine.R component of all the Backends. This file specifies a **Layout** and **fontFile** function that is used to instantiate a layoutEngine Backend Engine object using the **`makeEngine()`** function. The requirements and flexibility of this object is documented within the layoutEngine::makeEngine() help page.

The **`layout`** argument is the minimum requirement needed to instantiate Backend object. This function should perform take the supplied web-page data and then layout the page with a layout engine. This engine is expected to be provided within a typical browser but as seen with the CSS-Box Backend solution can also be a non-typical implementation of a layout engine.

Once the web-page layout has been performed within the browser (or equivalent), the next requirement is the layout is captured within a R readable form. This is a critical piece of the layoutEngine functionality. For this research, the script that executes this step was taken from the DOM Backend. In general, this JavaScript script updates the web-page DOM to isolate all text characters within a single bounding box. Once each element has its bounding box, the locations and content for each box is exported from the DOM and transformed into CSV format.

The layout CSV is then transferred back to the layoutEngine primary package where it redraws the graphic within the R graphic display. For all of this to work well, each component must be defined and tracked through this entire process. Of special concern is that of the specified fonts. Since all fonts vary dramatically in their sizes, the actual font files must be identified on the host machine and then transferred to the browser where they are incorporated into the web-page using the `@font-face` CSS feature.

The layoutEngine Backend has much flexibility in the way it can be implemented provided this core functionality is provided. In general, these minimum requirements of the `layout` function include:

1. Start up a server used to host the web-page
2. Find the specified font files within the host system
3. Copy asset files to the server (fonts, images, embedded HTML, etc)
4. Open a web browser
5. Serve the web-page
6. Execute the layout calculation script within the browser
7. Send the returned CSV back to the layoutEngine

4.3 Technical Requirements

The layoutEngine Backend must meet several key technical requirements to adequately perform the necessary functions as outline in the previous section. These requirements can be addressed in a variety of ways as is evident in comparing the existing Backends ([TODO:] see reference).

Technical requirements have been categorized to improve the ease of comparison.

4.3.1 Locally Hosted Server

- ► Live refresh of the browser upon updates to the DOM as a debugging tool
  - ▷ Server to host web-page

4.3.2 Browser Integration

- ► Maintain a single web browser instance for each layoutEngine call to prevent multiple browsers from sending layout calculations back to R
- ► Use of latest web browser to ensure ability to utilize the best features of HTML, CSS and JavaScript as they become available
  - ▷ Control of browser
    - ⋆ Automated (Selenium, or “open” call)
    - ⋆ Manual control
  - ▷ Access to latest web technologies

4.3.3 Bi-directional Communication

The backend must support a browser compatible communication protocol. Data must be transferred between R and the browser in both directions. The backend must first send the raw HTML based data to the browser. It must

- Communication Protocol
  - Access to webpage to POST and GET
- Has to support the transfer information in both directions
  - To the browser
    - Asset files such as images and font files to the hosted webpage from the host machine
    - DOM components from the R layoutEngine call
    - Call to execute the layout calculation script
  - From the browser
    - Either request or accept the layout calculation CSV as a file or in memory data
- Control of DOM
  - Add content to the page
  - Execute a JavaScript script and send the contents back

4.3.4 Installation Requirements

- Few dependencies on other development teams for critical components that might either create bugs or prevent the support of the latest web technologies
- As simple implementation as possible to
  - reduce the weight of the installed package
  - Enable easy future support or extensions by others
  - Provide as few opportunities for issues, bugs, etc and improve the maintainability and comprehension by other contributors to the project
- Cross platform support
  - Reduce the dependencies of OS specific packages to reduce the number cross-platform issues
- No security vulnerabilities
  - Opening up graphics device or network access of the host machine
  - Cross-platform support
    - Installation dependencies
    - Graphics device usage

4.3.5 Performance & Maintainability

- While not a high-priority, there should not be a large performance hit for the full function call
  - Performance
  - Bugs / Issues
  - Low dependencies on other embedded technologies
  - Ease of support and maintenance by other contributors

4.4 Target Enhancements

4.4.1 Cross Platform

An emphasis is placed on new layoutEngine backends to support all three major platforms (Linux, MacOS and Windows). While the existing backends prove the viability of the layoutEngine functionality it is deemed absolutely necessary for the package to be easily used on all platforms. There is little chance the package would be found useful across the industry if it were only available on Linux platforms. A primary reason for this being that a majority of users are on either Windows or MacOS.

4.4.2 Simple Dependencies

Secondary to cross platform support, the backend must also have relatively simple installation requirements for all platforms. The intention here is to improve the user experience by making the installation as easy as possible. In addition, with fewer requirements there is less opportunities for future incompatibilities to arise.

4.4.3 Industry Support

It is critical that any technologies that are incorporated into the layoutEngine backend have development support into the future. The more common and widely used such technologies are the less likely there will be technical issues as other parts of the ecosystem advance.

4.4.4 Modern Web Standards

It is preferred the backend design is able to support modern web standards for **Web Design and Applications** as defined by W3C. If the latest and greatest standards are not fully supported then an acceptable lag of 1 to 2 years from the most recent release. This feature should be considered as relatively important as many users will be turned off from too much lag between what is seen as industry standard versus cutting edge.

4.4.5 Visual Feedback

It should be considered valuable to have access to a live browser for several reasons. Although headless browsers might be considered more easily implemented there is significant value in being able to see the graphical output within the browser. For example, the user can see quickly identify any discrepancies between in supported web technologies between the browser and R graphics display.

5 RSelenium Backend

The RSelenium backend has been developed with the use of two preexisting technologies. The first is the **Selenium WebDriver** which is a robust browser automation tool. The second is the **RSelenium R library** which provides an interface to the Selenium WebDriver from within R.

The primary interest in using Selenium is that is a powerful and popular tool that offers a more robust platform to control a web browser. The fact the RSelenium library allowed for easy use of this tool off-the-shelf further increased its appeal. While the benefits for this solution design are significant there are also several limitations (or downsides) to its use.

This section first presents the solution design of the layoutEngine backend using Selenium. This is then followed by a review of the benefits and limitations of the design.

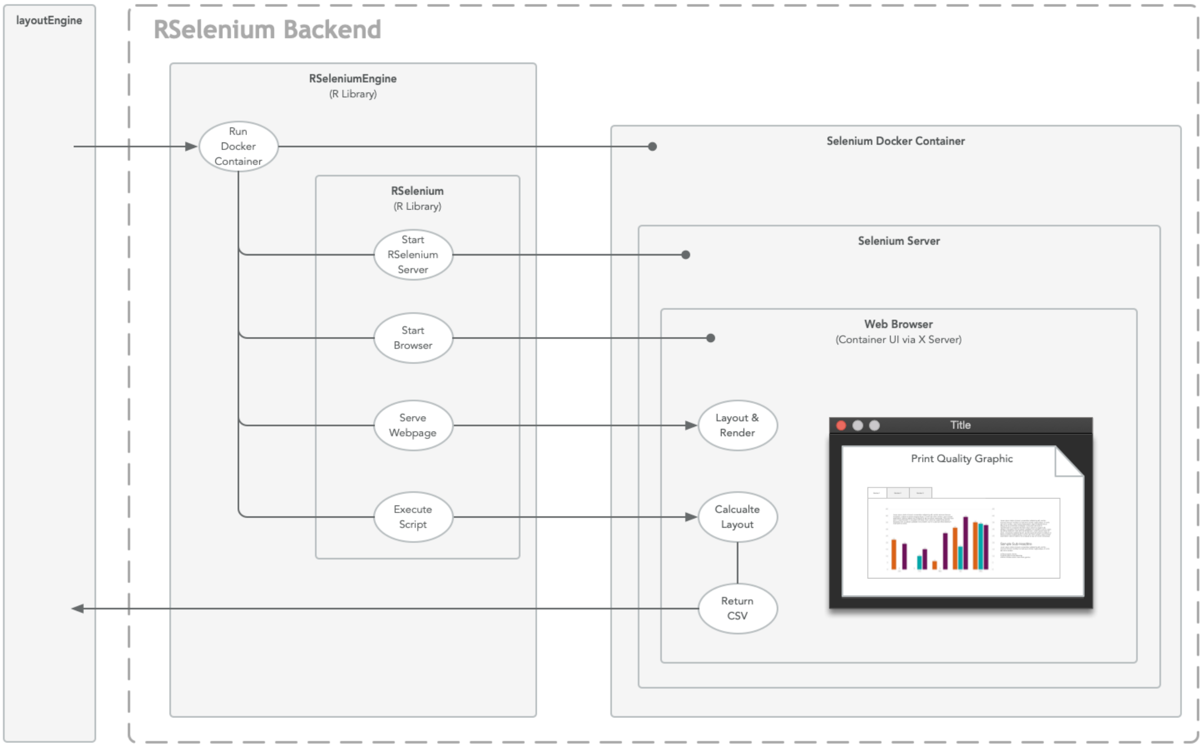
5.0.1 Overview

The motivation behind the RSelenium Backend package includes the following:

- Selenium Server
  - Robust solution that offers a large set of ways to control a browser
  - Strong technical community with evidence of future support due to wide industry use
  - Support of docker containers
  - Docker container includes controlled web browser types which improves consistency
  - Direct access to the webpage DOM via the Selenium server commands
- RSelenium
  - R interface to Selenium Server already exists
  - Tests shows relatively simple controls

5.0.2 Solution Design

- RSelenium R Package
- Selenium Docker images
- Docker container
- Browser launches from docker container (good and bad)



RSelenium Backend Design

1. RSelenium Backend Arguments
   - url='127.0.0.1'
   - portRS=4444
   - portClient=4444
   - network='bridge'
   - shm_size='1g'
   - browser_type='firefox'
   - headless=FALSE
   - image_request=NULL
   - fresh_pull=FALSE
2. layoutEngineRSelenium Backend Setup

```
library(layoutEngine)
library(layoutEngineRSelenium)
options(layoutEngine.backend=RSeleniumEngine)
## Firefox Selenium docker image
firefox_build <- "3.141.59-20200525"
firefox_image <- paste0("selenium/standalone-firefox-debug:", firefox_build)
## Chrome Selenium docker image
# chrome_build <- "3.141.59-20200525"
# chrome_image <- paste0("selenium/standalone-chrome-debug:", chrome_build)
## Select which image to test
test <- list(name="firefox", image=firefox_image)
# test <- list(name="chrome", image=chrome_image)
```

3. Docker Container Component
   The Selenium Docker container call `dockerContainer()` will start up the Selenium Docker container per the argument settings. It also has methods to get status and stop the container.

```r
## User settings for component test
settings <- list(url="127.0.0.1", portRS=4444, portClient="4444", network="bridge", shm_size="1g",
                 browser_type=test$name, headless=FALSE, image_request=test$image, fresh_pull=FALSE)
## Setup RSelenium Backend Docker container based on user settings
container <- dockerContainer(settings)
## Run the RSelenium docker container
container$run()
## Get RSelenium docker container status
container$getInfo()
## Stop and delete the RSelenium Docker container
container$close()
```

4. Selenium Server Component
   The Selenium server call **`rSServer()`** will start up the Selenium Docker container with the **`dockerContainer()`** call shown above. It will also setup the RSelenium server hosted within the Docker container with the **`open()`** method. It also has methods to get status and stop the container.

```r
## Setup RSelenium Backend Selenium server hosted on the docker container
RSServer <- rSServer(settings)
## Get Selenium server status
RSServer$getStatus()
## Open a browser instance hosted within the Docker container
RSServer$open()
## Close RSelenium Backend Selenium server hosted on the docker container
RSServer$close()
```

5. Selenium Browser Session Component
   The RSelenium Server call **`rSServer()`** will start up the RSelenium Docker container with the **`dockerContainer()`** call shown above. It will also setup the RSelenium server hosted within the Docker container with the **`open()`** method. It also has methods to get status and stop the container.

```r
## The outer most component starts all encapusalated components
RSSession <- rSSSession(url="127.0.0.1", portRS=4444, portClient="4444", network="bridge", shm_size="1g",
                        browser_type=test$name, headless=FALSE, image_request=test$image, fresh_pull=FALSE)
## Open a browser instance hosted on a Selenium server within the Docker container
RSSession$open()
## Get status of all the components
RSSession$getStatus()
## Close all components (browser session, server and Docker container)
RSSession$close()
```

5.0.3 Benefits

The backend performs quite well when working properly.

- The use of the Selenium server and browser session made it easy to ensure only a single browser session was used for each R layoutEngine call.
- The communication protocol used to control the Selenium browser's DOM was abstracted away from our development efforts. Since the Selenium server's primary functionality is offering tight controls of a browser we

were able to take full advantage of this. This meant that JavaScript DOM scripting methods can be directly applied to modify its content without the need for worrying about using a communication protocol such at HTTP or a WebSocket.

- Since we can directly modify the contents of the webpage DOM via the Selenium server controls we see live updates in the browser without having to refresh the page. This both improves the user feedback of the using the browser as a debugging device and adds the ability to execute the layout calculation script immediately upon the DOM update.
- Stand alone server seperate from R

5.0.4 Limitations

Some downsides of the RSelenium Backend include the following:

- Scheduling the start-up of the container, Selenium Server and then the web browser session created some difficulties
- Time required to open the full stack was near 5-10 seconds
- Opening the browser application hosted within the docker container in full mode (i.e. no headless) poses difficulties for Windows and MacOS due to the sharing of the host graphics system. It can be finicky to get this working on all host machines easily.
- The browser version is controlled by the Selenium development team which must be supported via one of their docker images. This creates a dependency on this external development team to ensure the latest browsers (and therefore latest web specifications) can be maintained.
- It was found that a latest docker image created a bug with the RSelenium R package that was no longer up to date. Dependencies on both the Selenium docker iamges and RSelenium R Package poses some risk of future bugs or lack of support for the latest features.

5.0.5 Summary

In general, the RSelenium backend offers a simple way of controlling a browser. However, there are several issues that this approach presents to future development. As a result, it does not seem to be a robust solution to continue development with.

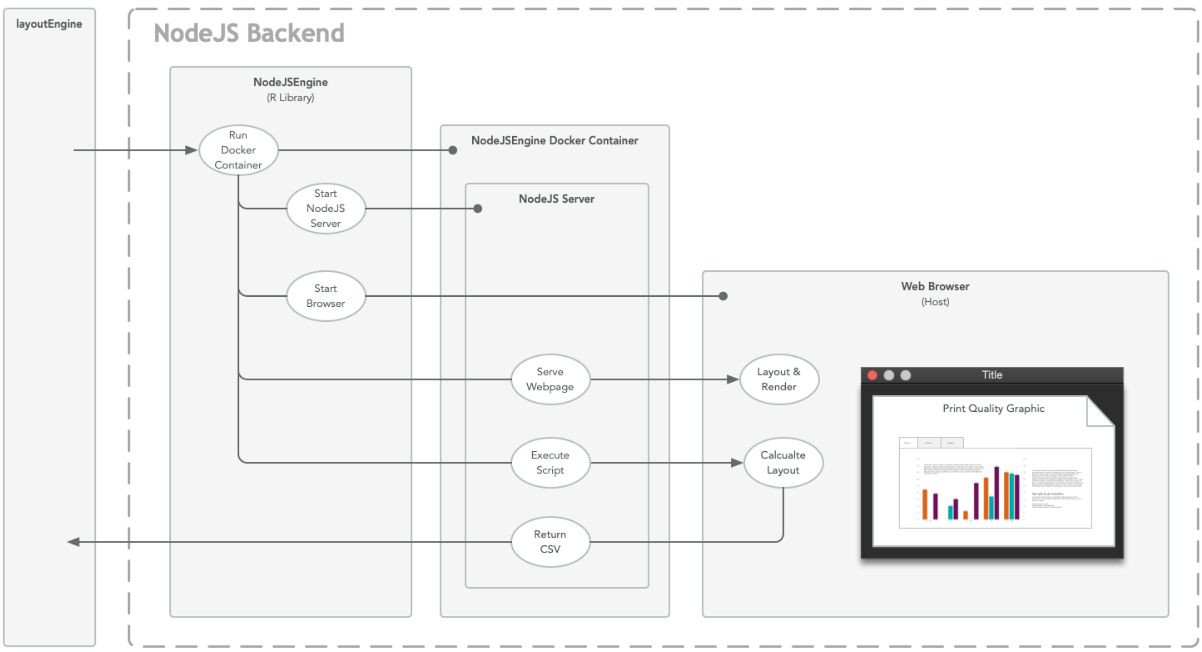
6 NodeJS Backend

6.0.1 Overview

The motivation behind the NodeJS Backend package includes the following:

- Building the server from scratch would result in very light-weight package
- Easily customize the functionality of it to suit the layoutEngine
- NodeJS is a very popular language with a large community and strong support into the future
- Several Node packages/libraries can be used to take advantage of
- Ability to stay at the very fore-front of web language support since we're in control of all depencies
- Few risky dependencies (i.e. RSlenenium and Selenium images)
- Easily work within Docker image for support multiple OS types via communication protocol such as WebSocket or HTTP API
- Can extend functionality of the backend service in an endless variety

6.0.2 Solution Design



NodeJS Backend Design

- Webpack Dev Server
- Express Server
- Web Socket
- Docker Container

6.0.3 Benefits

- Ability to design server side functionality to extend the usefulness and robustness of the solution
  - For example, transformations or formatting can be performed on the server before sending back to R

  - Very tight compatibiity between a Node server and the browser ensures strong solution design on that side
  - Aim would be to simplify the function calls as much as possible on the R interface
- Live reload still exists via WebPack dev server
- Lightweight dependencies
- Local browser (good and bad)
- Docker container
- Stand alone server separate from R
- Server could reside external to the host if there were ever a need or use-case for

6.0.4 Limitations

- Have to implement communication protocol
- Managing multiple browser and/or open tabs needs managing
- Performance hit to stand-up docker container and node server
- Unforeseen issues due to new design with little testing

6.0.5 Summary

7 Comparison

1. Summary of solution features, benefits and limitations
2. How do they rank with the existing **backends**?

8 Recommendations

1. Overview of layoutEngine as a solution to generating print quality graphics
2. Do the new backends improve its performance?
3. Where should future development work concentrate?

A Appendix

A.1 Development Environment

A single Docker container is used to perform research, experimentation, R package development and documentation. This environment was chosen to easily share the development content with others for collaboration and feedback. It will also ensure that any future return to this research can be resurrected with a working code-base independent of software changes.

The report and R development have been performed within Emacs and ESS environment inside of the Docker container. The report is written within the Emacs org-mode markdown language which abstracts some LaTeX syntax while also providing literate programming options which are more flexible than generic markdown or Rmarkdown.

Some basic Docker and Emacs commands are provided to walk the user through some of aspects of the build and editing processes.

A.1.1 Docker container description

Overview: The Docker container is publicly available on Docker Hub with the following image name **kcull\layoutengine-research**. The container is built from the Ubuntu 18.04 image and has R 3.6.3 and Emacs 27.1 installed. The container has been configured to run Emacs in its GUI environment on the host machine.

User and Home Directory: The user is logged in as a sudo-user with `/home/user/` as the $HOME directory. The sudo password is "password." The working directory is `/project/` which both the shell and Emacs will initialize into.

Directory Organization: The project also has the primary layoutEngine repositories cloned in the `\opt` directory.

Directory Hierarchy:

```
# Emacs configuration files
/home/user/.emacs.d/
# Github repository for research paper
/project/
# Github repository for layoutEngine
/opt/layoutengine
# Github repository for layoutEngineDOM
/opt/layoutenginedom
# Experimental code for layoutEngineRSelenium
/opt/layoutenginerselenium
# Experimental code for layoutEngineNodeJS
/opt/layoutenginenodejs
```

A.1.2 Host setup and Docker run instructions

The following instructions are provided to recreate the development environment. This has only been tested from within a host machine running Ubuntu 18.04 but is assumed to be compatible with other Debian derivatives.

- GitHub Repository: kcullimore/layoutengine-research
- Docker Image: kcull\layoutengine-research

1 - Download the docker image:

```
$ docker pull kcull/layoutengine-research:latest
```

2 - Create a working directory on the host machine and clone the github repository:

```
$ mkdir /home/$USER/layoutengine-research
$ git clone git@github.com:kcullimore/layoutengine-research.git /home/$USER/layoutengine-research
```

3 - Grant local access to your X server to allow Emacs to run in a local window and the run the docker container (setting is reset upon reboot): **Warning: this exposes your computer. Read more here.**

```
$ xhost +local:
```

4 - Run the docker container:

```
$ docker run --rm -it \
        --network host \
        --privileged=true \
        --env DISPLAY=unix$DISPLAY \
        --volume /tmp/.X11-unix:/tmp/.X11-unix \
        --volume /var/run/docker.sock:/var/run/docker.sock \
        --mount type=bind,source=/home/$USER/layoutegine-research/,target=/project/ \
        --name layoutengine-research \
        kcull/layoutengine-research:latest
```

5 - Once the docker container is up and running verify folder structure has correctly mapped the host directories.

6 - Open Emacs in the container's terminal: `$ Emacs`. The host should launch Emacs in its GUI form (i.e. not within the shell). If this doesn't occur verify steps 4 were followed thoroughly (NOTE: After reboot the display device will have to be provided access again with the `{\`$xhost + local : command$).

7 - From within Emacs perform the following operations to open and recreate the current report

- Opens Treemacs with `M-0`
- Open folder structure to `/project/paper/` with Tab-Enter or Mouse
- Open org-mode markdown file `layoutengine-research-paper.org` with Enter or Mouse double-click
- Make some edits to the file and save with `C-x C-s`
- Launch Export Dispatcher menu with `C-c C-e`
- Create new PDF file with `C-l C-o`

8 - The PDF should have opened automatically which you can scroll through with arrow keys or the mouse scroll wheel. Use `q` key to minimize the PDF buffer.

9 - Close Emacs with `C-x C-c` and exit the container by typing `exit` at the terminal.

10 - Navigate to the project directory on the host machine and verify the new PDF and edited org-mode file were correctly saved.

11 - If the above worked the project appears to be correctly established on the host machine.

A.1.3 Emacs within Docker Container

Emacs Terminology

- **buffer:** 'Screen' or 'window' user operates within
- **marking:** Highlighting region of window

Often used commands can be found at `https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf`.

Customized keybindings

- Open emacs configuration file with `C-c e` (Emacs must be restarted for changes)
- Expand all nested/hidden text within *.org file with `Shift-Tab Shift-Tab Shift-Tab`

- Copy, cut and paste with standard keybindings per **Cua Mode**
- Switch visual line wrap with `M-9`
- Switch to truncate long-line view with `M-8`
- Enter/Exit rectangle edit mode with `C-^`
- Enter/Exit multi-edit mode by highlighting word and then `C-u`
- Auto-indent R script (via ESS) by highlighting buffer with `C-x h` and then C-M-}

Document Export

When a PDF version of the document is produced a standard TEX file (*.tex) is also produced after transpilation. This TEX file can be edited and used with a standard LATEX command: `latex report.tex`.

To be continued...

A.2 Org-mode examples

A.2.1 Font definitions

Using LaTeX fontspec package [1]

Sans

Internet based applications are an increasingly popular way to communicate and interact with complex data.

Sans italic

Internet based applications are an increasingly popular way to communicate and interact wtih complex data.

Sans italic bold

This might include a business application that assist employees unverstand the current state of the market.

Serif

It might also include a news website communicating techincal details from a story such census data.

Serif italic

It might also include a news website communicating techincal details from a story such census data.

Serif italic bold

It might also include a news website communicating techincal details from a story such census data.

`Mono type`

`It might also include a news website communicating techincal details from a story such census data.`

`Mono Bold type`

`The quick brown fox 012456789`

A.2.2 Sample R code highlighting

```r
##*******10********20********30********40********50********60********70********80
## Problem 2: START => Optical Illusion Example
##*******10********20********30********40********50********60********70********80
## Generate pdf file of plot (capture ends with dev.off() below)
pdf("prob-02.pdf", width = 3, height = 6)
## Create theta values  for each line segments (i.e. 180 degs / 4 = 45 segments)
## Remove elements in the center of vector (i.e. 80-100 degree section)
theta <- seq(0, pi, length = 45)[-(20:26)]
## Set parameters to be used in plot() (R = dummy radius, B = slope of lines)
R <- 1
B <- sin(theta) / cos(theta)
## Setup plot space and define coordinate axes (also remove 'edge buffer')
plot.new()
par(mar = c(0.1, 0.1, 0.1, 0.1))
plot.window(xlim = c(-R, R), ylim = c(-R, R), asp = 1)
## Create the black line segments
for (i in 1:length(B)) abline(a = 0, b = B[i], lwd = 2)
## Create the 2 red vertical lines
abline(v = c(-R/2, R/2), col = "red", lwd = 4)
## Stop image capture
invisible(dev.off())
##*******10********20********30********40********50********60********70********80
## Problem 2: END
##*******10********20********30********40********50********60********70********80
```

A.2.3 Sample HTML code highlighting

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1,
            maximum-scale=1.0, user-scalable=0"
    />
    <!-- favicon -->
  </head>
  <body>
    <title>DOM - Testing Application</title>
    <div id="AppDiv" class="app-div"></div>
  </body>
</html>
```

A.2.4 Sample CSS code highlighting

```css
.iah-text-Raleway {
  font-family: 'Raleway', sans-serif;
  font-weight: 500;
}

.iah-text-black {
  font-family: 'Roboto Mono', monospace;
  font-weight: 500;
  font-size: 2em;
  overflow-wrap: break-word;
  margin: 10px;
  color: var(--iah-grey-dark);
}
```

A.2.5 Sample JavaScript code highlighting

```
var args = []; // Empty array, at first.
for (var i = 0; i < arguments.length; i++) {
    args.push(arguments[i])
} // Now 'args' is an array that holds your arguments.

// ES6 arrow function
var multiply = (x, y) => { return x * y; };

// Or even simpler
var multiply = (x, y) => x * y;
```

B References

Murrell, P. (2018). “Rendering HTML Content in R Graphics” Technical Report 2018-13, Department of Statistics, The University of Auckland.

References

[1] package used to manage fonts within xelatex (or luatex) fontspec: http://ctan.math.washington.edu/tex-archive/macros/latex/contrib/fontspec/fontspec.pdf

[2] docker socket solution https://jpetazzo.github.io/2015/09/03/do-not-use-docker-in-docker-for-ci/