Analysis of Terascope Benchmarking Data

Mark R. Tyrrell

23/01/2019

1. Introduction

Visualization methods panoramic images show potential for visualizing multiscale urban data. accessible route to supercomputer visualizations on many low cost devices. Visual supercomputing deployed 14 PFlop cloud supercomputer, larger than any GPU HPC system in UK. cost in total for one result (a scaling graph) ~£20,000 using > £10 million computer. Azure improved (from K80 to V100) during the project, we immediately benefited. provides access to performance approx. 20 years ahead of current desktop systems.

2. Background

Terapixel images offer an intuitive, accessible way to present information sets to stakeholders, allowing viewers to interactively browse big data across multiple scales. The challenge we addressed here is how to deliver the supercomputer scale resources needed to compute a realistic terapixel visualization of the city of Newcastle upon Tyne and its environmental data as captured by the Newcastle Urban Observatory.

Our solution is a scalable architecture for cloud-based visualization that we can deploy and pay for only as needed. The three key objectives of this work are to: create a supercomputer architecture for scalable visualization using the public cloud; produce a terapixel 3D city visualization supporting daily updates; undertake a rigorous evaluation of cloud supercomputing for compute intensive visualization applications.

We demonstrate that it is feasible to produce a high quality terapixel visualization using a path tracing renderer in under a day using public IaaS cloud GPU nodes. Once generated the terapixel image supports interactive browsing of the city and its data at a range of sensing scales from the whole city to a single desk in a room, accessible across a wide range of thin client devices.

Problem Description

The dataset below was created from application checkpoint and system metric output from the production of a terapixel image. There are a variety of problems which can be addressed using the TeraScope dataset. Each will commence with an exploratory data analysis. Examples of questions you may wish to be able to answer through the EDA process are as follows:

Which event types dominate task runtimes? What is the interplay between GPU temperature and performance? What is the interplay between increased power draw and render time? Can we quantify the variation in computation requirements for particular tiles? Can we identify particular GPU cards (based on their serial numbers) whose performance differs to other cards? (i.e. perpetually slow cards). What can we learn about the efficiency of the task scheduling process? A project may focus entirely on the EDA process, or you may also wish to develop a data product to allow interactive exploration of the data. For example, you could develop a Shiny dashboard to display the results of your analysis into render performance and system operation, displaying timelines of execution based on the application checkpoint data.

3. Methodology

Data analysis was conducted using the R programming language and the R Studio IDE, with the report composed in Rmarkdown. The **ProjectTemplate** library was used to organise the project and ensure reproducability. Version control was managed using git and Github.

An archived folder is available containing the original Rmarkdown .rmd file and git log. The folder is structured for standard data mining projects, including subfolders for data, cache, munge and source files. As the data analysis involves operations which require extended processing time, key objects have been cached as .Rdata files. The objects are called along the progression of the script (and Rmarkdown report), while the original code is included in the munge folder. Detailed instructions for reproducing the analysis are included in the main README file.

The original csv data files are too large for Github at 300+ MB. Therefore they are not included in the archived project folder. A download link can be found in the README file.

3.1 Data Understanding

The data made available for this analysis consists of 3 tabular data files in comma-separated value (.csv) format covering multiple aspects of the terapixel rendering process. The data files are detailed below.

3.1.1 Application Checkpoints (application-checkpoints.csv 111.2MB)

This file contains 660,400 observations over 6 variables detailing application checkpoint events throughout the execution of the render job.

1. **timestamp** Timestamp of observation entry
2. **hostname** Unique hostname of the virtual machine auto-assigned by the Azure batch system (Total unique: 1024)
3. **eventName** Name of the event occuring within the rendering application. Possible Values:
    - **Tiling** Tiling is where post processing of the rendered tile is taking place
    - **Saving** Config Saving Config is simply a measure of configuration overhead
    - **Render** Render is when the image tile is is being rendered
    - **TotalRender** TotalRender is the entire task
    - **Uploading** Uploading is where the output from post processing is uploaded to Azure Blob Storage
4. **eventType** Possible Values: **START**, **STOP**
5. **jobId** ID of the Azure batch job (Total: 3 one for each level 4, 8 and 12)
6. **taskId** ID of the Azure batch task (Total: 65793 level 4 1x1; level 8 16x16; level 12 256x256)

3.1.2 GPU (gpu.csv 208.7MB)

This file contains 1,543,681 observations over 8 variables detailing metrics that were output regarding the status of the GPU on the virtual machine.

1. **timestamp** Timestamp of observation entry
2. **hostname** Unique hostname of the virtual machine auto-assigned by the Azure batch system (Total unique: 1024)
3. **gpuSerial** Unique serial number of the physical GPU card (Total unique: 1024)
4. **gpuUUID** Unique system id assigned by the Azure system to the GPU unit (Total unique: 1024)
5. **powerDrawWatt** Power draw of the GPU in watts
6. **gpuTempC** Temperature of the GPU in Celsius
7. **gpuUtilPerc** Percent utilisation of the GPU Core(s)
8. **gpuMemUtilPerc** Percent utilisation of the GPU memory

3.1.3 Task X Y (task-x-y.csv 6.2MB)

This file contains 65,793 observations over 5 variables detailing the tile x,y co-ordinates of which part the image was being rendered for each task.

1. **jobId** ID of the Azure batch job (Total: 3 one for each level 4, 8 and 12)
2. **taskId** ID of the Azure batch task (Total: 65793 level 4 1x1; level 8 16x16; level 12 256x256)
3. **x** X co-ordinate of the image tile being rendered (Total unique: 256)
4. **y** Y co-ordinate of the image tile being rendered (Total unique: 256)
5. **level** The visualisation created is a zoomable "google maps style" map. In total we create 12 levels. Level 1 is zoomed right out and level 12 is zoomed right in. You will only see levels 4, 8 and 12 in the data as the intermediate level are derived in the tiling process.

3.2 Data Preparation

The csv files were read into R and reviewed. Data types were changed as required (eg. timestamps, factors to char etc). In the case of the GPU dataset, the extraneous column **gpuUUID** was dropped, as it was redundant with **gpuSerial** as a unique identifier. The Task X Y dataset was fully merged into the Application Checkpoint dataset, as it contained only 3 unique columns. The resulting object **app_task** was cached along with the **gpu** object for the GPU dataset, after checking for NAs. These cached objects form the basis of the analysis.

The analysis made heavy use of the Dplyr package to transform and filter **app_task** and **gpu**. Multiple analyses were based on transformed datasets that required extensive processing time. These datasets were cached as objects, with the original code included in the munge folder. This function was particularly exemplified in the form of the **gpu_task** dataset, which was the output of the computation of mean resource usage for each event under each of the 65793 level 4 tasks in the **app_task** dataset. This computation took three hours on a Corei5 system.

4. Exploratory Data Analysis

As detailed in section 2.2, the data show 1024 unique VMs, each with an attached GPU. These 1024 GPUs were used to render 65,793 pixels generation operations identified by a unique task ID. The four phases of pixel generation are: Saving Config, Rendering, Tiling and Uploading. Each of these phases is logged in the **app_task** dataset, once at the start of the phase, and once at the end. In addition, a fifth entry is made for the total rendering process. This results in 10 observations for each pixel generation.

4.1 Examination of Single GPU

The performance data over the first 10 minutes for a sample GPU (S/N: 323617042631) are displayed in Figure 1. The cycles for each tiling job can be clearly seen in the plot for each metric. Each tiling job took appears to take approximately 40 seconds. GPU utlisation percentage appears relatively uniform for each tiling job at approximately 90%, whereas there is more variation on the other metrics. This is possibly a result of a percentage usage software limit on the GPU. Unsurprisingly, the temperature builds up from a cold start over successive tiling jobs.

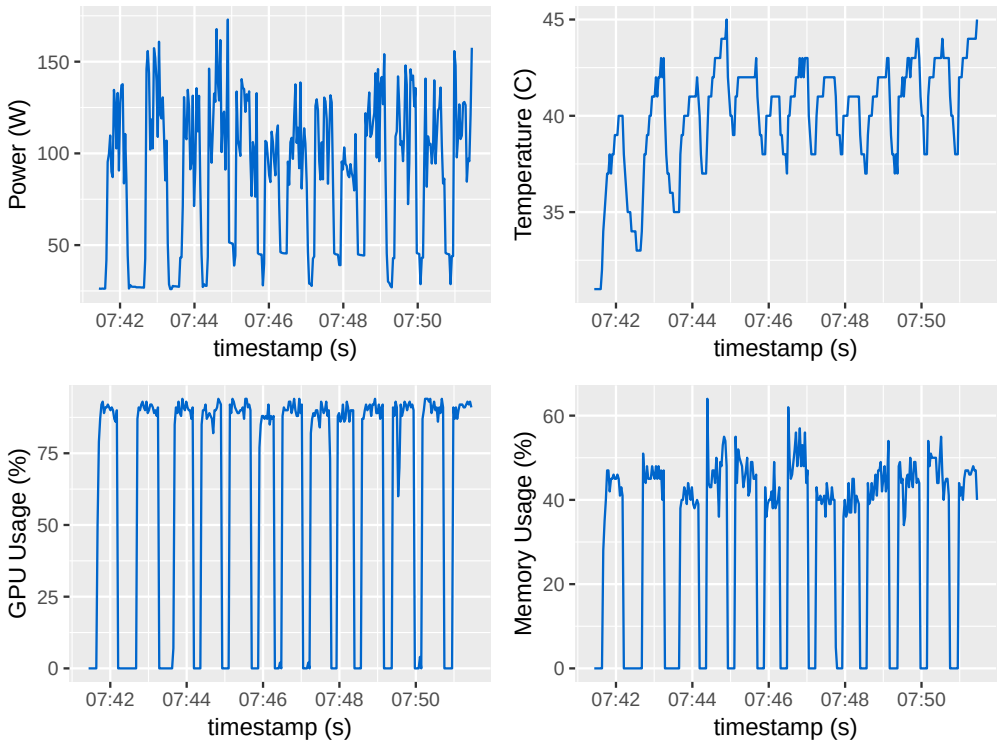

Figure 1: Sample GPU Performance Metrics over Time

4.2 GPU Performance Metrics Correlation

Figure 2 displays scatter plots of the GPU performance metrics correlations. As the GPU dataset is extremely large at 1.5 million observations, a random sample of 10,000 was selected for the plots. As demonstrated, there is a postive correlation between all of the metrics. However, there is also significant variation. This finding matches the observations taken from Figure 1. The positive correlation is to be expected, as the increased workload on the GPU in the form of GPU utilisation percentage will naturally require associated memory for the tiling job, and the demands of this utilisation on the hardware components will result in increased power consumption and rising temperature.

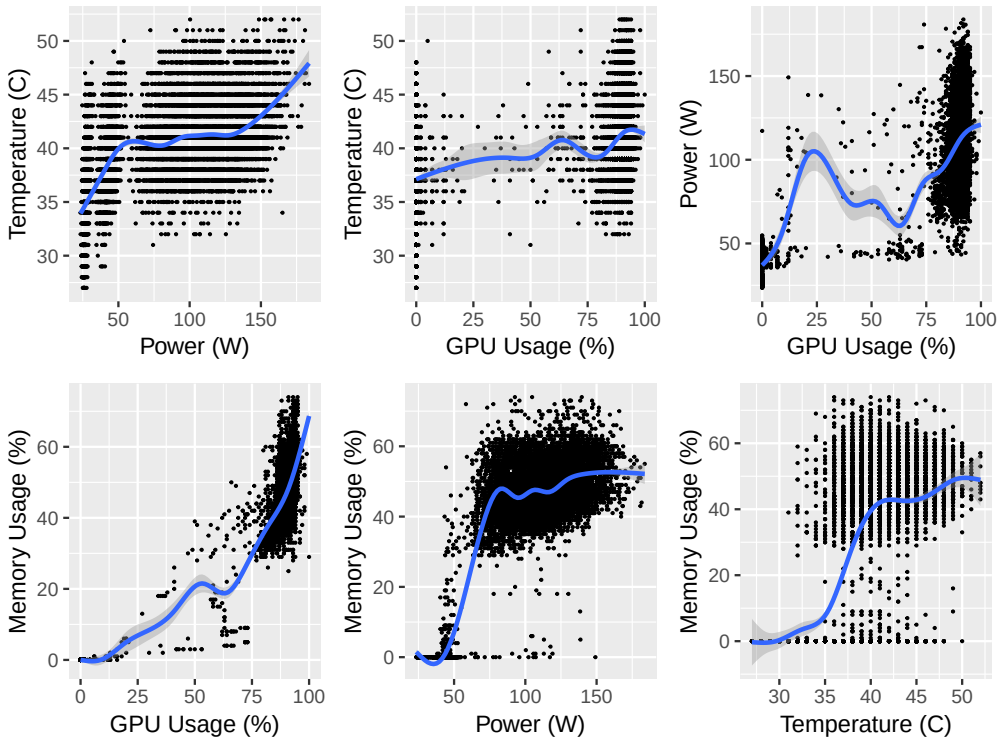

Figure 2: GPU Performance Metrics Correlation

4.3 GPU Performance Metrics Analysis

4.3.1 GPU Performance Summary Statistics

Table 1 displays the mean of each GPU performance metric by each event. The rendering task is easily identifiable as the most energy-intense phase of the tiling job, with power consumption more than double that of the second highest, tiling. There is much less variation in relative temperatures, though rendering is still noteably higher in mean temperature than the other phases.

Table 1: Mean GPU Performance by Event (all Levels)

Event	Power (W)	Temperature (s)	GPU Usage (%)	Memory Usage (%)	n
Render	95.80	40.56	71.467	37.438	65546
Saving Config	36.09	37.96	0.000	0.000	108
Tiling	44.76	39.50	0.000	0.000	31694
Uploading	44.01	39.34	0.000	0.000	35080

There are noteable missing data in Table 1 for the Saving Config, Uploading and Tiling phases over GPU and memory utilisation. The dataset used to compute the table was taken from **gpu_task**. This dataset is a join of **gpu** with **app_task**, where each gpu observation is matched by the VM hostname and timestamp for each event. Despite the high logging resolution of the operation (each GPU logged performance metrics approximately every 2 seconds), on average these events occur over very short intervals (less than 1 second). As such, they were statistically less likely to be 'sampled', and consequently have a much smaller sample size relative to the rendering phase.

Table 2 provides quantified insight into this effect by detailing the mean runtime durations for each event.

The phases in question display very short average durations. This effect is particularily exemplified in the case of Saving Config. With an average interval of 2ms, it only appears in 108 of the GPU observations.

The presence of power and temperature data for the underrepresented phases in the absence of corresponding GPU and memory data remains questionable. It is likely the temperature metric follows a heat loading dynamic, taking time to cool off after each rendering phase. The negative corrollary of this effect was already observed (ref. Figure 1). The power metric can assumed to be exhibiting baseline consumption. With regard to GPU and memory, the phases in question can be assumed to put only relatively low strain on the hardware resources. Therefore it is not surprising to see zero utilisation on these metrics, as any small percent utilisation caught by the sampling process would be effectively zeroed out by the mean operation.

Table 2: Mean Execution Time by Event (Level 12 only)

Event	Duration (s)	n
Render	41.206	65536
Saving Config	0.002	65536
Tiling	0.973	65536
Uploading	1.373	65536

4.3.2 GPU Performance Metrics Distribution

The summary statisics for the GPU performance metrics provide a succinct method for comparing the various phases in terms of impact on system resources. However, they provide only limited insight the actual system performance. A more robust method of benchmarking high performance computational systems is to analyse quantiles[1].

Figure 3 displays the distributions of execution time for each pixel generation. The top plot presents the execution time for rendering and the bottom for the tiling phase. Both distributions appear bimodal, with the larger mode appearing mostly normal with a skew to the right. The smaller mode appears to represents a group of GPUs which exhibit significantly higher performance.

The features of the render distribution highlight the importance of documenting system performance beyond summary statistics. The long tail to the right demonstrates the large variance in system performance. Were one to assume a normal distribution, the standard deviation for the render execution time distribution would be calculated as 6.047s. Therefore the maximum execution time at 81.51s would not be conceivable (as a 6.7 sigma event) on such a small dataset.

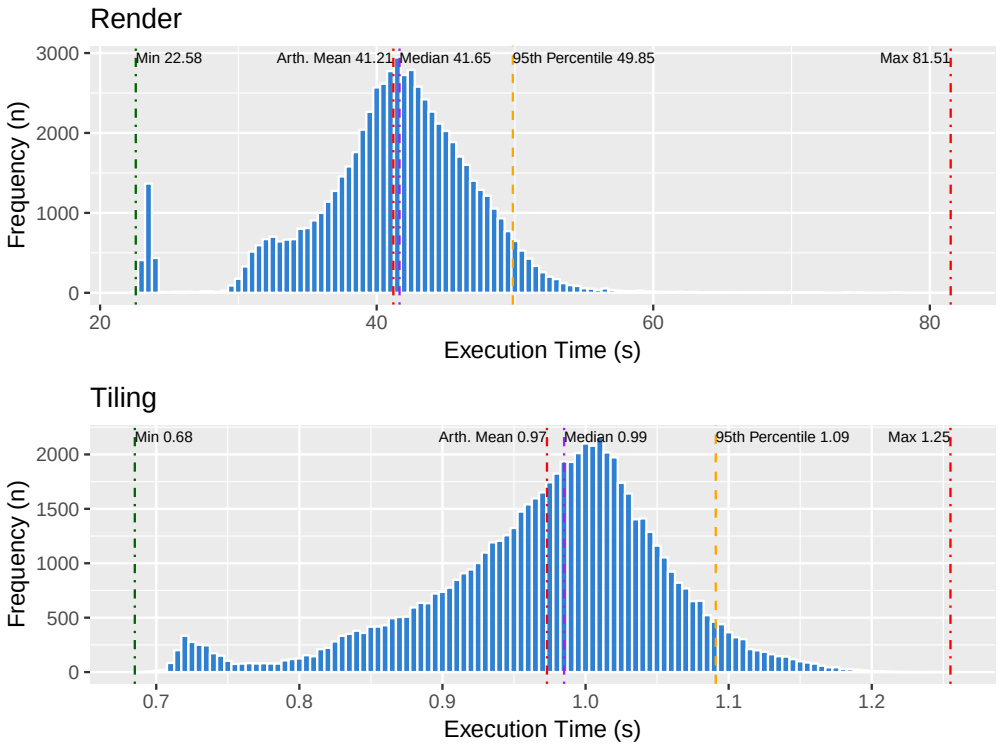

Figure 3: Distribution by Execution Time: Render & Tiling

The bimodal characteristics of the distributions in Figure 3 are more pronounced for the rendering phase distribution; a finding which is logical given the doubled sample size for the rendering phase, as well as the vastly larger quantity of GPU logging data available over the 40+ second phase duration. Additionally as the time scale for the rendering phase is relatively large compared to tiling, it is easier to inherently comprehend the effects of variance on the operation through visualisation. As the comparative performance and relationship between the render phase and all other phases has already been established in previous sections, from this point the render phase will be examined exlcusively in order to further evaluate system performance.

Figure 4 displays the distribution of Power Consumption and Temperature

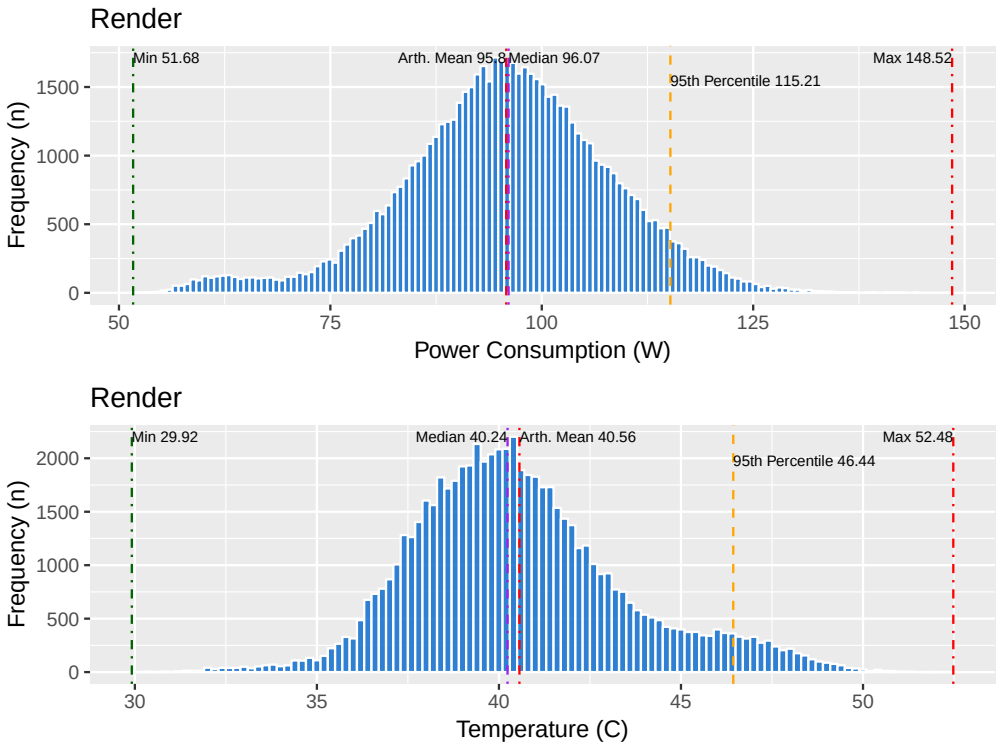

Figure 4: Distribution by Power Consumption & Temperature: Render

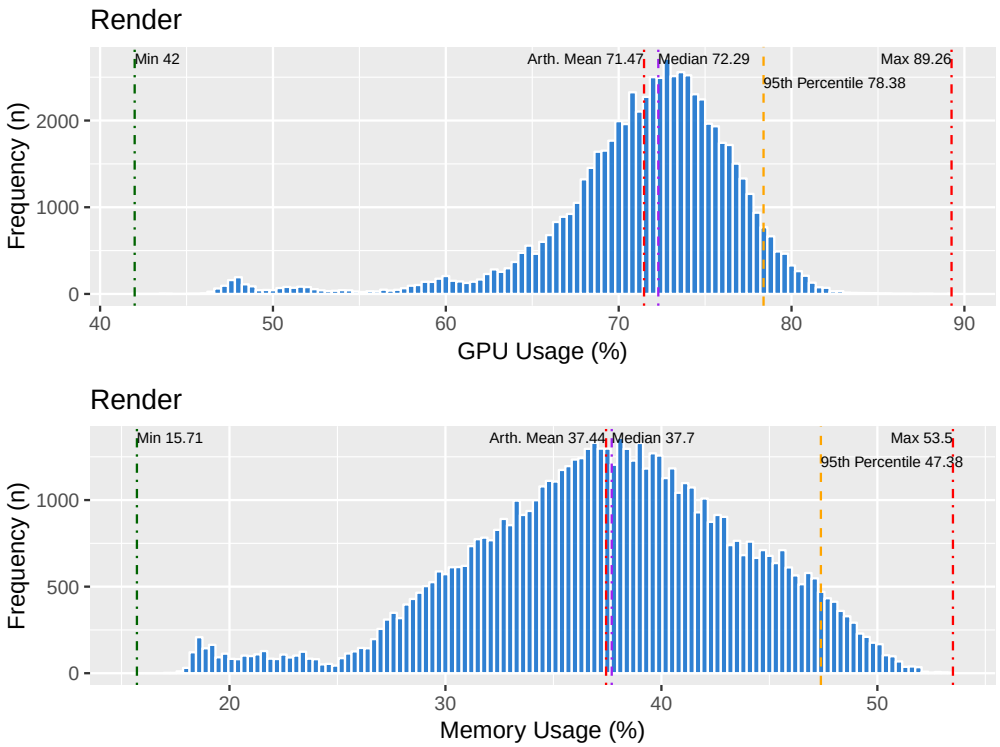

Figure 5: Distribution by GPU & Memory % Utilisation: Render

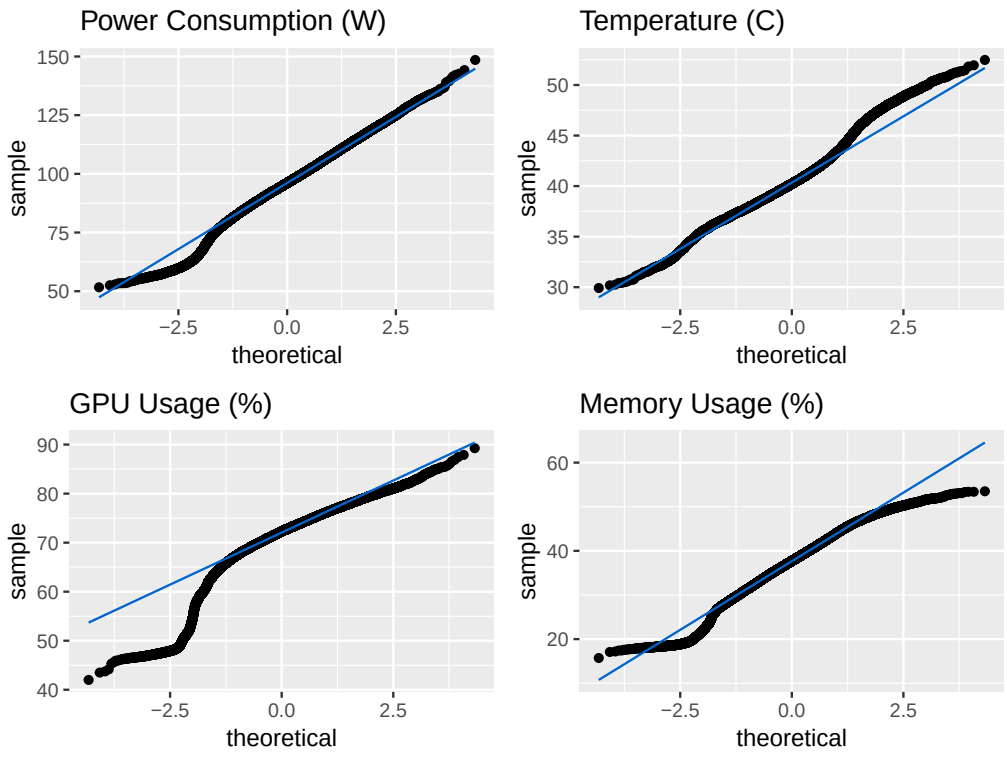

Figure 6: QQ Plot for GPU Performance Metrics

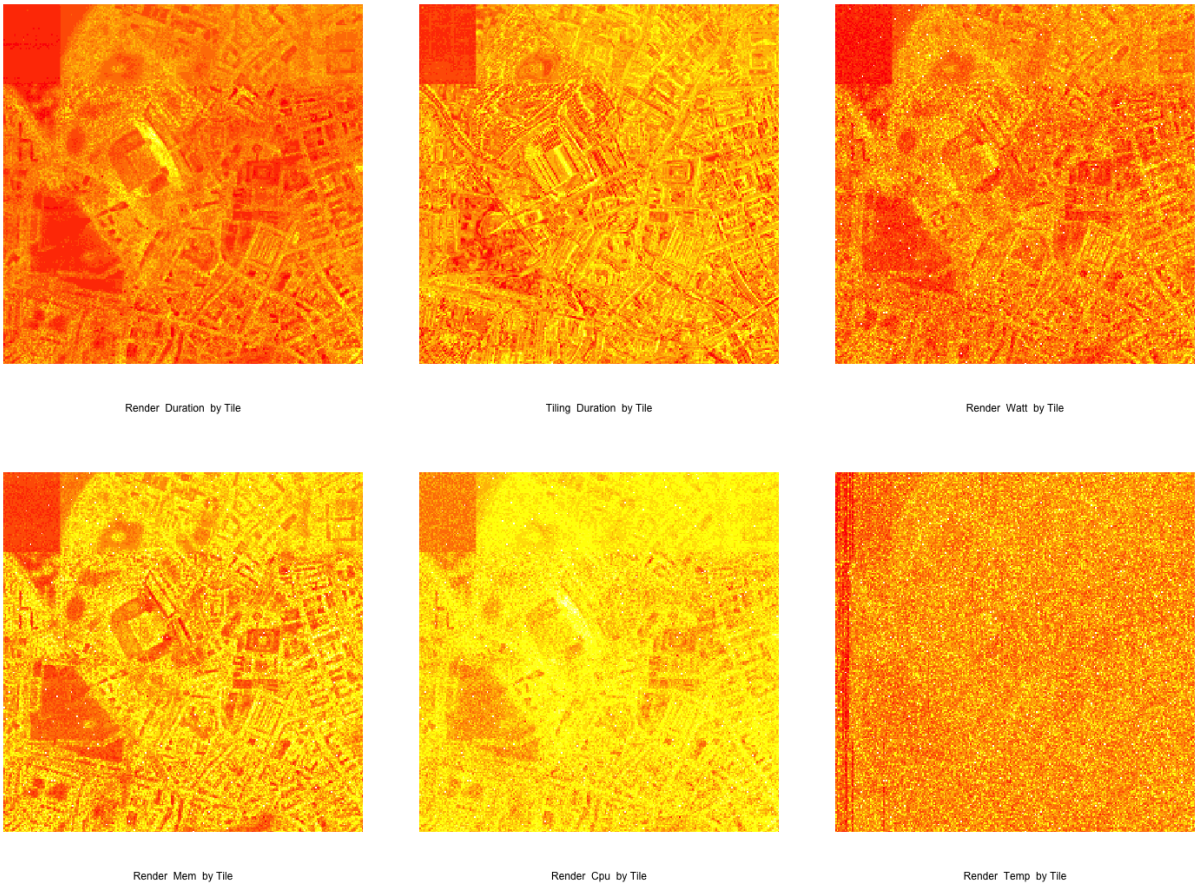

Figure 7: Distribution by GPU & Memory: Render

4. Summary of Results Evaluation and Business Conclusions

The analysis returned numerous results relevant to the business objectives. These results are summarised and evaluated below. Summary conclusions and reccomendations are included where applicable.

4.1 Increase Enrolment

Gender Gap
The difference in course enrollment between genders demonstrates that there is room for growth in female student usage. It may be possible to boost female enrollment by differential marketing strategies or tailoring the VLE in a gender-focused way.

Overly Educated Student Base
There is room for growth in lesser educated demographics

Largely UK Focused
There is opportunity for student base expansion to English-speaking developing countries. Review of platform multi-language options could provide insights into how non-english audiences could be better reached.

4.2 Improve Course Completion Rate

Completion Rate
At 5.8 percent, the completion rate is standard performance for the sector, but lessons learned by other MOOCs could be applied here to increase completion rates. For instance, is the platform providing the latest in terms of user interface and pedagocical approaches?

Course Duration
The 71 day median (43 day mean) for course completion contrasts with the course estimate of 3 weeks. It is possible students sign up expecting an easier or quicker course. The stated completion estimate should be reviewed and either changed, or course structure change to more accurately reflect the real course duration.

Day 90 Course Completion Upswing
The bimodal nature of the dropout by step findings raises questions about why there is such a notable increase in course completions from the 3-month mark. Is it in fact auto-scheduled email reminders? This should be reviewed as it would prove efficacy of this functionality and provide an evidence base for other such automated reminders.

Step 3.21 (Duration Uutlier)
Taking on average 8 days to complete, this step was a major outlier. The particulars of this step should be reviewed and the step possibly reformatted, as it correlates with a high dropout for the same step number.

Downward Trend in Dropout Follows Step Durations
Though this could be simply a platform-based learning curve, it could signify a less than optimum structuring. For instance, easier steps earlier in the course might result in greater step achievement and course completion on average.

Gender Performance Trend
Review gender-focused course accessibility for females. Why are women doing worse than men? It may be possible to tailor the VLE in a gender-focused way based on provided demographic student data.

Age Performance Trend
Review course age-focused attributes. For instance, could a different interface for younger learners improve performance?

Retirees Performance Trend
Retirees show above average completion rates. Marketing to this demographic could improve overall course completion rates.

Nationality Performance Trend
Review platform multi-language options as well as current VLE user interface with regards to english as a second language (ESL) users.

4.3 Increase Percentage of Paid Upgrades

Course Purchase Correlation with Completion
Increased data from future cohorts could further elucidate the dynamics behind these metrics. A review of causation could be possible with an online-based (operational) randomised control trial (RCT). This could involve offering free/discounted evaluation/certification to random learners and comparing results with a control group.

Conclusions for future data mining

The large quantity of missing demographic data (approximately 90%) limited meaningful analysis of subsets of the sample. For instance, though there were 289 observations in the dataset of paid learners, only 13 of these obervations had complete demographic variables. Therefore it was impossible to draw statistical inference about demographic factors contributing to paid upgrades. If there was someway of enticing users

to enter demographic data more frequently, it could vastly increase the utitlity of the learning analytics for this course.

References

1. http://www.policyconnect.org. uk/hec/sites/site_hec/files/ report/419/fieldreportdownload/ frombrickstoclicks- hecreportforweb.pdf

2. Chuang, Isaac and Ho, Andrew, HarvardX and MITx: Four Years of Open Online Courses – Fall 2012-Summer 2016 (December 23, 2016). Available at SSRN: https://ssrn.com/abstract=2889436 or http://dx.doi.org/10.2139/ssrn.2889436

3. National Science Foundation, National Center for Science and Engineering Statistics, Scientists and Engineers Statistical Data System (SESTAT) and National Survey of College Graduates (NSCG) (1993, 2013), http://sestat.nsf.gov.

Appendix: R-Code

The code below is provided for reference and reproducibility purposes. The sections detail the individual data files used for the analysis, and the order reflects the loading seqence.

```
################################### Munge: 01-A.R ######################################

##### Read csv data files, combine by type (rbind), clean/transform as required and cache as .RData

##### Enrolments
# Load and combine all datasets
list_sel = list.files(path = "data/", pattern = "*enrolments*")
x <- lapply(list_sel, function(i) read.csv(file = paste("data/",i,sep = "")))
df_enrolments <- do.call('rbind', x)

# Clean & Transform
df_enrolments = df_enrolments %>%
        #Convert date columns to POSIX UTC
        mutate(enrolled_at = as.character(enrolled_at)) %>%
        mutate(unenrolled_at = as.character(unenrolled_at)) %>%
        mutate(fully_participated_at = as.character(fully_participated_at)) %>%
        mutate(purchased_statement_at = as.character(purchased_statement_at)) %>%
        mutate(enrolled_at = ymd_hms(enrolled_at, tz="UTC")) %>%
        mutate(unenrolled_at = ymd_hms(unenrolled_at, tz="UTC")) %>%
        mutate(fully_participated_at = ymd_hms(fully_participated_at, tz="UTC")) %>%
        mutate(purchased_statement_at = ymd_hms(purchased_statement_at, tz="UTC")) %>%
        #Convert country to character vector
        mutate(detected_country = as.character(detected_country)) %>%
        # Add derived variables to display Days Enrolled, Days to Completion, and binary completion var
        mutate(days_enrolled = as.numeric(round(difftime(unenrolled_at, enrolled_at, unit="days"),0))) %
        mutate(days_to_completion = as.numeric(round(difftime(fully_participated_at, enrolled_at, unit=
        mutate(completed = ifelse(is.na(fully_participated_at), 0, 1)) %>%
        mutate(learner_id = as.character(learner_id))

cache("df_enrolments")

##### Step Activity
```

```
# Load and combine all datasets
list_sel = list.files(path = "data/", pattern = "*activity*")
x <- lapply(list_sel, function(i) read.csv(file = paste("data/",i,sep = "")))
df_step_activity <- do.call('rbind', x)

# Clean & Transform
df_step_activity = df_step_activity %>%
        #Convert date columns to POSIX UTC
        mutate(first_visited_at = ymd_hms(first_visited_at, tz="UTC")) %>%
        mutate(last_completed_at = ymd_hms(last_completed_at, tz="UTC")) %>%
        mutate(learner_id = as.character(learner_id))

cache("df_step_activity")


##### Functions
#Tabulate function (shows percentage)
tblFun <- function(x){
        tbl <- table(x)
        res <- cbind(tbl,round(prop.table(tbl)*100,2))
        colnames(res) <- c('Count','Percentage')
        res
}

cache("tblFun")

################################ EDA EDA_Enrolment.R ######################################

# DF Step Activity

# Create working object from cached dateset
df_enrol = df_enrolments

cache("df_enrol")


### 3.1 Profile existing student base

#Student data summary statistics (filtering out unknowns and insufficient samples)
df_enrol_dt = filter(df_enrol, gender != "Unknown" & gender != "nonbinary" & gender != "other"
                     & age_range != "Unknown" & detected_country != "--" & highest_education_level != "

#Reorder factor levels and remove unknown
df_enrol_dt$age_range = factor(df_enrol_dt$age_range, c("<18","18-25","26-35","36-45","46-55","56-65","
df_enrol_dt$gender = factor(df_enrol_dt$gender, c("male", "female"))
df_enrol_dt$highest_education_level = factor(df_enrol_dt$highest_education_level, c("apprenticeship","l
df_enrol_dt$employment_status = factor(df_enrol_dt$employment_status, c("full_time_student","looking_fo
#Rename factor levels
df_enrol_dt$highest_education_level = mapvalues(df_enrol_dt$highest_education_level,
        from = c("apprenticeship","less_than_secondary","professional","secondary","tertiary","universi
        to = c("apprentice","< secondary","professional","secondary","tertiary","bachelors","masters","
df_enrol_dt$employment_status = mapvalues(df_enrol_dt$employment_status,
        from = c("full_time_student","looking_for_work","not_working","retired","self_employed","unempl
```

```
        to = c("student","seeking work","not working","retired","self-employed","unemployed","full-time


cache("df_enrol_dt")


#Gender histogram
ggplot(df_enrol_dt) + stat_count(aes(gender), fill=I("#0066CC"), col=I("white"), alpha=I(0.8)) +
        labs(x = "Gender", y = "Count", title = "Gender")

tblFun(df_enrol_dt$gender)

#Age group histogram
ggplot(df_enrol_dt) + stat_count(aes(age_range), fill=I("#0066CC"), col=I("white"), alpha=I(0.8)) +
        labs(x = "Age Groups", y = "Count", title = "Age Groups")

tblFun(df_enrol_dt$age_range)

#Education histogram
ggplot(df_enrol_dt) + stat_count(aes(highest_education_level), fill=I("#0066CC"), col=I("white"), alpha=
        labs(x = "Education", y = "Count", title = "Education")

tblFun(df_enrol_dt$highest_education_level)

#Employment histogram
ggplot(df_enrol_dt) + stat_count(aes(employment_status), fill=I("#0066CC"), col=I("white"), alpha=I(0.8
        labs(x = "Employment Status", y = "Count", title = "Employment Status")

tblFun(df_enrol_dt$employment_status)

#Country
#Transform data for detected country, aggregate by count for each country, select top countries by x am
df_enrol_ct = df_enrol %>%
        dplyr::select(detected_country) %>%
        filter(detected_country != "--") %>%
        mutate(count = 1) %>%
        group_by(detected_country) %>%
        summarise(sum(count)) %>%
        rename(Country = detected_country, Count = "sum(count)") %>%
        mutate(Percentage = round(Count/sum(Count)*100,2)) %>%
        filter(Count > 95) %>%
        arrange(desc(Count))

cache("df_enrol_ct")

#Convert to names for table
df_enrol_ct_name = df_enrol_ct
df_enrol_ct_name$Country = countrycode(df_enrol_ct_name$Country, "iso2c", "country.name")
cache("df_enrol_ct_name")

#Plot and table
ggplot(df_enrol_ct_name, aes(Country, Count)) + geom_col(fill=I("#0066CC"), col=I("white"), alpha=I(0.8
        labs(x = "Country", y = "Count", title = "Nationality")
```

```
df_enrol_ct_name[1:10,]

#Create map object
map_country = invisible(joinCountryData2Map(df_enrol_ct, joinCode = "ISO2", nameJoinColumn = "Country",
cache("map_country")
#creating a user defined colour palette
# op = palette(c('green','yellow','orange','red'))
# #find quartile breaks
# cutVector = quantile(df_enrol_ct$Count, prob = seq(0, 1, length = 11), type = 5)
# #classify the data to a factor
# df_enrol_ct$Count = cut(df_enrol_ct$Count, cutVector, include.lowest = TRUE)
# #rename the categories
# levels(df_enrol_ct$Count) = c('low', 'med', 'high', 'vhigh', 't1', 't2','t3','t4','t5','t6')
# #mapping by shade
# par(mai=c(0,0,0.2,0),xaxs="i",yaxs="i")
mapCountryData(map_country, nameColumnToPlot = "Count", catMethod='logFixedWidth', mapTitle='Learners by

#mapping by bubble
# par(mai=c(0,0,0.2,0),xaxs="i",yaxs="i")
# mapBubbles(dF=map_country, nameZSize="Count", nameZColour="Count", numCats = 10, catMethod='categoric

### 3.2 Performance Metrics

##### Completion Data (n = 2154)
df_enrol_days_cp = filter(df_enrol, !is.na(days_to_completion))

cache("df_enrol_days_cp")

#Plot Course Completion Duration frequency
ggplot(df_enrol_days_cp, aes(days_to_completion)) + geom_histogram(binwidth = 10, fill=I("#0066CC"), col
        labs(x = "Days", y = "Count", title = "Histogram: Successful Course Completion Duration (days)"
        scale_x_continuous(breaks = round(seq(min(df_enrol_days_cp$days_to_completion), max(df_enrol_day
ggsave(file.path('graphs', 'course_completion_duration.pdf'))


################################ EDA EDA_Step_Activity.R ###################################

#DF Step Activity

# Create working object from cached dateset
df_step = df_step_activity

# 1. Analysis of Completion Duration by Step
#Compute number of average days take to complete each step
df_step_avg = df_step %>%
        #Add derived variable for duration per step
        mutate(total_days = as.numeric(round(difftime(last_completed_at, first_visited_at, unit="days")
        #Strip all variables except step and total_days
        dplyr::select(step, total_days) %>%
        #Aggregate
        group_by(step) %>%
        summarise(total_days = round(mean(total_days, na.rm = TRUE),1)) %>%
        #Remove sub-levels
```

```
        filter(step < 1.11 | step > 1.19) %>%
        filter(step < 2.11 | step > 2.19) %>%
        filter(step < 2.21 | step > 2.29) %>%
        filter(step < 3.11 | step > 3.19)

cache("df_step_avg")

#Plot average days per step
ggplot(df_step_avg, aes(step, total_days)) + geom_col(size = 1, fill=I("#0066CC"), colour = "#0066CC", a
        title = "Average Completion Duration per Step (days)")  +
        scale_x_continuous(breaks = round(seq(min(df_step_avg$step), max(df_step_avg$step), by = 0.1),1)
ggsave(file.path('graphs', 'avg_comp_duration_step.pdf'))


# 2. Analysis of Dropout Frequency by Step
df_step_max = df_step %>%
        #Compute maximum level completed for each unique learner ID
        filter(is.na(last_completed_at)) %>%
        dplyr::select(learner_id, step) %>%
        group_by(learner_id) %>%
        summarise(step = max(step))

cache("df_step_max")

df_step_drop = df_step_max %>%
        # Add Count variable (= 1 for each row) and drop learner_id
        mutate(Count = ifelse(!is.na(learner_id), 1, 0)) %>%
        mutate(learner_id = NULL) %>%
        # Remove sub-levels
        filter(step < 1.11 | step > 1.19) %>%
        filter(step < 2.11 | step > 2.19) %>%
        filter(step < 2.21 | step > 2.29) %>%
        filter(step < 3.11 | step > 3.19)

cache("df_step_drop")

#Plot frequency of drop out by step number
ggplot(df_step_drop, aes(step)) + geom_histogram(binwidth = 0.1, fill=I("#0066CC"), col=I("white"), alpl
        labs(x = "Step", y = "Count", title = "Histogram: Last Step Completed") +
        scale_x_continuous(breaks = round(seq(min(df_step_drop$step), max(df_step_drop$step), by = 0.1)
        theme(text = element_text(size=10))
ggsave(file.path('graphs', 'dropout_by_step.pdf'))

#Percentage dropout by step 1.1
dim(filter(df_step_drop, step == 1.1))[1]/dim(df_step_drop)[1]


################################## EDA EDA_Join.R ##########################################


### Join enrolments data (age, gender etc) with step activity on learner_id and look at performance base
# Left join on enrolements data (as opposed to step activity data) because demographic (interesting) da
#Dataset containing full demographic variables (n = 3819)
```

```r
df_step_perf = left_join(df_enrol_dt, df_step_max, copy = FALSE)
df_step_perf = df_step_perf %>%
        #Drop unused columns
        dplyr::select(-(enrolled_at:fully_participated_at)) %>%
        mutate(country = NULL) %>%
        #Convert all "completed" steps to 3.9 (otherwise shows NA in some cases)
        mutate(step = ifelse(is.na(step) & completed == 1, 3.9, step)) %>%
        #Create new binary var to show course purchase
        mutate(purchased = ifelse(!is.na(purchased_statement_at),1,0))

cache("df_step_perf")

####Predictor Gender
p1 = ggplot(data = df_step_perf, mapping = aes(x = gender, y = step)) + geom_boxplot(fill=I("#0066CC"),
        labs(x = "Gender", y = "Step", title = "a) Gender") +
        scale_y_continuous(breaks = round(seq(min(df_step_perf$step, na.rm = TRUE), max(df_step_perf$st
dat1 = ggplot_build(p1)$data[[1]]
p1 = p1 + geom_segment(data=dat1, aes(x=xmin, xend=xmax, y=middle, yend=middle), colour="white", size=0
cache("p1")

####Predictor Age
p2 = ggplot(data = df_step_perf, mapping = aes(x = age_range, y = step)) + geom_boxplot(fill=I("#0066CC
        labs(x = "Age Range", y = "Step", title = "b) Age Range") +
        scale_y_continuous(breaks = round(seq(min(df_step_perf$step, na.rm = TRUE), max(df_step_perf$st
dat2 = ggplot_build(p2)$data[[1]]
p2 = p2 + geom_segment(data=dat2, aes(x=xmin, xend=xmax, y=middle, yend=middle), colour="white", size=0
cache("p2")

####Predictor Education
p3 = ggplot(data = df_step_perf, mapping = aes(x = highest_education_level, y = step)) + geom_boxplot(f
        labs(x = "Education Level", y = "Step", title = "c) Education") +
        scale_y_continuous(breaks = round(seq(min(df_step_perf$step, na.rm = TRUE), max(df_step_perf$st
        theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5))
dat3 = ggplot_build(p3)$data[[1]]
p3 = p3 + geom_segment(data=dat3, aes(x=xmin, xend=xmax, y=middle, yend=middle), colour="white", size=0
cache("p3")

####Predictor Employment Status
p4 = ggplot(data = df_step_perf, mapping = aes(x = employment_status, y = step)) + geom_boxplot(fill=I(
        labs(x = "Employment Status", y = "Step", title = "d) Employment Status") +
        scale_y_continuous(breaks = round(seq(min(df_step_perf$step, na.rm = TRUE), max(df_step_perf$st
        theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5))
dat4 = ggplot_build(p4)$data[[1]]
p4 = p4 + geom_segment(data=dat4, aes(x=xmin, xend=xmax, y=middle, yend=middle), colour="white", size=0
cache("p4")

pred_grid = grid.arrange(p1, p2, p3, p4, ncol=2)


##Country boxplot
#Derive and sort top 10 countries from df_enrol_ct dataset
ordered <- order(df_enrol_ct$Count, decreasing = TRUE)
top_countries = df_enrol_ct[ordered,][1:20,]
```

```
#Filter dataset for top 10 countries
df_step_perf_ct = df_step_perf %>%
        filter(detected_country %in% top_countries$Country) %>%
        mutate(step = ifelse(is.na(step), 0, step)) %>%
        group_by(detected_country) %>%
        summarise(step = mean(step))
        # arrange(step)
df_step_perf_ct$detected_country = countrycode(df_step_perf_ct$detected_country, "iso2c", "country.name
cache("df_step_perf_ct")
####Predictor Country
ggplot(data = df_step_perf_ct, mapping = aes(x = reorder(detected_country, step), y = step)) + geom_col
        theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5)) +
        labs(x = "Country", y = "Step", title = "Step Performance by Country (Top 20)")


################################ LM Var Tests  ##########################################

#Ages
levs = levels(df_predictors$age_range)
p = c()
for(i in 1:length(levs)){
        x = dim(filter(df_predictors, age_range == levs[i] & completed == 1))[1]
        y = dim(filter(df_predictors, age_range == levs[i]))[1]
        z = x/y
        p[i] = log(z/ (1 - z))
}
plot(p)

#Education
levs = levels(df_predictors$highest_education_level)
p = c()
for(i in 1:length(levs)){
        x = dim(filter(df_predictors, highest_education_level == levs[i] & completed == 1))[1]
        y = dim(filter(df_predictors, highest_education_level == levs[i]))[1]
        z = x/y
        p[i] = log(z/ (1 - z))
}
plot(p)

############################### Linear Modelling  ########################################

### Predictive Modelling

df_predictors = df_step_perf

#Transform predictor dataset
df_predictors = df_predictors %>%
        #Remove extra vars
        dplyr::select(-(learner_id:purchased_statement_at)) %>%
        dplyr::select(-(employment_area:days_to_completion)) %>%
        dplyr::select(-(step:purchased))
```

```r
#Convert nominal categorical var to binary integer
df_predictors$gender = (as.integer(df_predictors$gender)-1)
#Convert categorical vars to integers as per results of ordinalt tests (ref. Ordinal_test.R))
df_predictors$age_range = as.integer(df_predictors$age_range)
df_predictors$highest_education_level = as.integer(df_predictors$highest_education_level)



###### PRELIM MODEL
#Deconstruct, standardise and rebuild predictors dataset (& remove employment status because of convers
df_predictors_std = scale(as.matrix(df_predictors[,1:3]))
df_predictors_lm = data.frame(df_predictors_std, df_predictors[,5])
names(df_predictors_lm) = c("gender","age_range","highest_education_level","completed")

#Fit the Linear model using the dataset
lr_fit = glm(completed ~ ., data = df_predictors_lm, family = "binomial")
summary(lr_fit)
#Compute prediction probabilities
phat = predict(lr_fit, df_predictors_lm, type = "response")
#Compute fitted (i.e. predicted) values
yhat = ifelse(phat > 0.5, 1, 0)
#Calculate confusion matrix
table(Observed=df_predictors_lm$completed, Predicted = yhat)
#Compute training error
1 - mean(df_predictors_lm$completed == yhat)



###### FINAL MODEL

#Final model Age + Gender
df_predictors_lm_red = df_predictors_lm[,-3]
names(df_predictors_lm_red) = c("gender","age_range","completed")

set.seed(20)
#Create training and testing sets
len_set = dim(df_predictors_lm_red)[1]
train_qt = round(len_set*0.9,0)
random_sel = sample(len_set, len_set, replace = FALSE)
#Create test/training indices
train_sel = random_sel[1:train_qt]
test_sel = random_sel[(train_qt+1):len_set]
#Slice predictors df into test/training dfs
df_train = filter(df_predictors_lm_red, row_number() %in% train_sel)
df_test = filter(df_predictors_lm_red, row_number() %in% test_sel)

#Fit the Linear model using the dataset
lr_fit_final = glm(completed ~ ., data = df_train, family = "binomial")

cache("lr_fit_final")

summary(lr_fit_final)
```

```r
#Compute prediction probabilities
phat_test = predict(lr_fit_final, df_test, type = "response")
#Compute fitted (i.e. predicted) values
yhat_test = ifelse(phat_test > 0.5, 1, 0)
#Calculate LR confusion matrix
conf_matrix = table(Observed=df_test$completed, Predicted = yhat_test)
#Compute LR test error
lr_test_error = 1 - mean(df_test$completed == yhat_test)

cache("conf_matrix")
cache("lr_test_error")



###### CROSS VALIDATION

n=nrow(df_predictors_lm_red)

set.seed(20)
#10-fold cross validation
nfolds = 10
#Sample fold-assingment index
fold_index = sample(nfolds, n, replace=TRUE)
fold_sizes = numeric(nfolds)
#Compute fold sizes
for(k in 1:nfolds){
        fold_sizes[k] = length(which(fold_index==k))
        fold_sizes
}

#Assign vector for avg MSE
cv_lsq_errors = numeric(nfolds)

#Loop through folds fitting model to k-1 training data and predicting values for k, assign errors to ve
for(k in 1:nfolds){
        #Fit model by least squares using all but the k-th fold
        lsq_tmp_fit = glm(completed ~ ., data=df_predictors_lm_red[fold_index!=k,], family = "binomial"
        #Compute fitted values for the k-th fold
        phat = predict(lsq_tmp_fit, df_predictors_lm_red[fold_index == k,], type = "response")
        #Work out the MSE for the k-th fold
        yhat = ifelse(phat > 0.5, 1, 0)
        cv_lsq_errors[k] = mean((df_predictors_lm_red[fold_index==k,]$completed - yhat)^2)
}

#Take mean of error vector
cross_val_test_error = weighted.mean(cv_lsq_errors, w=fold_sizes)

cache("cross_val_test_error")
```