Using the R package Shiny to create web applications that facilitate quantitative gene expression analysis

Emilia Gan

A thesis

submitted in partial fulfilment of the

requirements of the degree of

Master of Science

University of Washington

2016

Committee:

Timothy Rose, Chair

Michael Lagunoff

James Brinkley

Program Authorized to Offer Degree:

Pathobiology

University of Washington

Abstract

Using the R package Shiny to create web applications that facilitate quantitative gene expression analysis

Emilia Gan

Chair of the supervisory committee:
Timothy Rose, PhD
Pediatric Infectious Diseases

Kaposi's Sarcoma-associated Herpesvirus (KSHV), the etiologic agent of Kaposi's Sarcoma (KS) infects host cells and predominantly enters a latent state. Tumorigenesis, however, likely requires lytic replication. The signals that cause the virus to transition from latency to active viral replication are not understood completely. RNA-sequencing provides investigators with a way to study the viral transcriptome to determine patterns of gene expression. However, processing and analyzing the very large data sets obtained via high-throughput sequencing can be challenging. Numerous software packages have been written using the R programming language to assist investigators in working with large biological data sets. However, the steep learning curve required to use R effectively makes these packages potentially difficult to use. The new R package Shiny is a web application framework for R. For this project, two applications were developed using Shiny – 1) an application to generate figures with side-by-side box plots for selected genes, demonstrating the variance in the expression of these genes across samples; and 2) an application to perform principal components analysis on a dataset. Both applications provide the user with a graphical user interface, thus bypassing the need for any coding skills on the part of the investigator.

Table of Contents

1. Introduction

Kaposi's Sarcoma-Associated Herpesvirus (KSHV) is the etiologic agent of Kaposi's Sarcoma (KS), the most commonly occurring cancer in people infected with the Human Immunodeficiency Virus (HIV). Kaposi's Sarcoma was first described by Moritz Kaposi in 1872 (1), but the etiology of the disease remained unknown for more than 100 years. The virus responsible for KS was discovered in 1994 (2), when the mortality and morbidity of AIDS-associated KS spurred research into identifying the cause of the disease. KSHV was later shown to be the cause of Primary Effusion Lymphoma (PEL) and Multicentric Castleman's disease, in addition to KS (1).

Like all herpesviruses, KSHV exists in the host cell either as latent or as actively replicating virus. The conditions and triggers that cause the virus to transition from a state of latency to one of active viral replication are not clearly understood. Numerous researchers have studied viral gene expression with the aim of elucidating this key stage in the viral life cycle. RNA-sequencing provides researchers with a powerful tool in such investigations, but comes with associated challenges of its own in terms of processing and analyzing the large quantity of data produced.

Our lab has been using RNA-sequencing to study patterns of viral gene expression in KS tumor biopsy specimens. As a consequence, we have had to consider a number of issues related to working with data obtained via next-generation sequencing methods.

This thesis project recognizes the importance of computational tools in analyzing RNA-sequencing data. Two web applications were built using the R package, Shiny, with the goal of making common data exploration and data analytic methods more convenient for use by lab researchers. The first application creates custom plots of side-by-side boxplots showing gene expression variation between samples, while the second application performs principal components analysis of a gene expression counts data set. Both applications provide researchers with a graphical user interface (GUI) to use while interacting with R. The second part of this thesis project demonstrates the use of the above-described web applications as applied to research data sets.

2. Background

2.1 KSHV Background

2.1.1 KSHV Epidemiology

In areas where both KSHV and HIV are endemic, such as in sub-Saharan Africa, KSHV represents a significant public health concern. It is possible for KSHV to be transmitted via sexual contact, but the primary mode of transmission is likely through saliva (1, 2, 3). In KSHV-infected individuals, the virus tends to be localized within host cells, and viremia is rare. However, KSHV is found to be shed in the saliva of many, but not all, infected individuals, regardless of their HIV status (3). In areas where KSHV is endemic, this non-sexual mode of transmission infection appears to result in most individuals becoming infected during childhood, with the virus being transmitted from mother to children (1, 3), between siblings, and from outside contacts (3). Children and females are affected by KS to a much greater extent in Africa than in any other part of the world, with many childhood infections occurring before puberty (4). (3) With the emergence of HIV-1, KS incidence in endemic areas has surged, with a concomitant increase in morbidity due to Kaposi's Sarcoma (KS). By the early 1990s, approximately one quarter of all childhood cancers in Zambia were KS (3), and in Uganda and Zimbabwe, KS is the most common cancer in men and the second most common in women (1). The prevalence of KSHV in these regions is estimated to be quite high, ranging from 25% to 30% for Cameroon and South Africa to as high as over 75% for countries such as Botswana and Congo(1). KSHV thus presents a significant public health burden.

2.1.2 KSHV Infection and KS Pathogenesis

The life cycle of KSHV is notable for the existence of two types of viral infection. Most commonly, the virus enters host cells, reaching the nucleus and establishing a latent infection. During this latency period, a minimal subset of viral proteins are expressed. These viral proteins serve to help the virus evade immune detection and eradication (5, 6, 7, 8). In addition to latency, the virus can enter a lytic state with widespread

production of viral proteins. Lytic infections generally lead to cell lysis with concomitant release of newly produced virions (5). The factors that lead to the transition from latency to lytic replication are not completely understood (9, 10, 11). It is probable that a combination of host cell and viral factors are required. Similarly, the factors that lead to the development of KS are also not completely elucidated (8, 12), but immunosuppression, along with angiogenesis and differentiation of circulating cells of the hematopoietic lineage are all thought to play critical roles in oncogenesis (12, 13, 14,15,).

KSHV infection alone is not sufficient to lead to development of KS (2). In most infected immunocompetent individuals, KS infection results in minimal symptomology. Typically, any pigmented lesions that appear are slow-growing and do not result in significant pathology. KS tumors don't typically appear unless the host is immunocompromised (16). Before the emergence of HIV, KS tumors occurred primarily in elderly men of eastern European and Mediterranean heritage and was rarely fatal (12). KS was also seen in organ transplant patients whose immune systems were being artificially suppressed to prevent organ rejection (2).

KS tumors are highly vascularized and are characterized by the proliferation of inflammatory cells (8), accompanied by angiogenesis. KSHV infected cells in KS tumors tend to be highly proliferative and poorly differentiated (8). In addition, host cells undergo a dramatic conformational change, assuming a characteristic spindle shape (12). Clinically, KS tumors appear as dermatological lesions that assume a number of morphologies ranging from flat, pigmented macular lesions to raised nodules, to aggressive fungating lesions (8). These lesions may occur on the skin, mucosal surfaces, or viscerally (8). Whether KS tumors progress or regress is dependent on host immune status. Even in the context of co-infection with HIV, KS tumors will generally spontaneously regress upon initiation of antiretroviral (HAART) therapy (2).

2.1.3 Genomic Structure of KSHV

KSHV is one of 8 known human herpesviruses. It is a member of the gammaherpesvirus subgroup, and is closely related to Epstein-Barr virus (EBV), another oncogenic herpesvirus (4). The gammaherpesviruses have evolved to efficiently subvert cellular signaling pathways (6, 17, 18, 19, 20). The pathways disrupted by the virus favor prolonged cell survival and include cell cycle progression, apoptosis, and immune surveillance pathways (8). In this manner, the virus is able to establish a persistent lifelong infection of the host. However, it is thought that these changes, which disrupt the normal regulation of the cellular life cycle, contribute to the development of neoplasia (8).

KSHV is a large double-stranded linear DNA virus. Its genome is between 165 - 170 kb (8, 20) and encodes more than 80 open reading frames (ORFs) (21). The genome is organized with a long unique region (LUR) situated between two GC-rich terminal repeat (TR) sequences that can vary in length (8). The KSHV genome contains many ORFs that are conserved in α- and β-herpesviruses (8). There are also a number of ORFs that are specific to KSHV, which are named K1 through K15 based on their location in the genome (8). KSHV also contains genes that are homologs of cellular genes (8). KSHV genes are divided into groups based on where the genes are expressed in the viral life cycle. A core subset of genes are known as latency genes. These genes are located in the “latency locus” (22), and include the standard marker of latency, the latency-associate nuclear antigen (LANA-1) encoded by ORF73 (23). Recent work, however, suggests that the division between “latent” genes and “lytic” genes is not as clear cut as previously assumed, and that a more complex model that considers cell type and other, potentially unknown, factors, is needed to describe the pattern of gene expression during latency.

2.1.4 KSHV Latency and Reactivation to Lytic Replication

As is the case with other herpesviruses, the KSHV lifecycle has two distinct phases (8), latency, where very few viral genes are expressed, and lytic replication, where all viral genes are expressed and new virions are produced. Upon initial infection with KSHV,

the virus briefly expresses some lytic genes (24, 25). Within the first 24 hours following *de novo* infection, the virus travels to the nucleus and the linear viral genome circularizes to become an episome (26). At this point, a small proportion of infected cells may contain virus undergoing lytic activation, but in most cells the virus is in the latent state. KS lytic replication is associated with the production and release of new viral particles, and thus with KSHV viremia (2), leading in turn to an increased risk of developing KS (2). Upon reactivation from latency, all KSHV genes are expressed in a well-studied lytic replicative cascade. Once KS has developed, more than 95% of the KS lesions will contain viral DNA (8), but in any given lesion, only 5% or less of the virus present may be actively replicating lytic virus (27, 28). This lytic program can be initiated via the expression of a single viral gene, the replication and transcriptional activator (Rta) encoded by ORF50 (29, 30).

2.1.5 Expression of KSHV Viral Proteins

KSHV viral gene expression is tightly regulated (31). Based on the results of transcript mapping experiments, KSHV gene expression is divided into three categories: class I, class II, and class III (31). Class I consists of genes whose expression is considered to be constitutive, while classes II and III consist of genes with inducible expression (31). As mentioned earlier, when the virus is in the latent state, as it is in the vast majority of infected cells in KS lesions, gene expression is severely restricted (31). LANA (ORF73) is considered the signature of latency, and is believed to be constitutively expressed in all KSHV infected cells, both *in vitro* and *in vivo* (31). Using antibodies to LANA, a characteristic pattern of punctate immunoreactivity can be seen in the nuclei of most KSHV-infected spindle cells and endothelial cells in KS tumors (31).

2.2 RNA-Sequencing Background

2.2.1 Transcriptome Analysis

The transcriptome was first studied through the use of microarrays, such as those produced by Affymetrix. Microarrays use a hybridization-based approach to

transcriptome discovery. Known sequences of DNA are used as probes to capture transcripts, thus requiring prior knowledge of the genome under study. Microarray technology is also limited by the high levels of background signal due to cross-hybridization and by a narrow dynamic range of detection due to both background signals and signal saturation.

RNA-Sequencing (RNA-Seq), as the name implies, is a sequence-based approach to studying the transcriptome. As with microarrays, RNA-Seq provides a “snapshot” of the transcriptome. This is a more complete snapshot, however, than the one obtained from a microarray experiment. The actual mRNA transcripts produced from the genome at a given point in time are all included in the data output, whereas in a microarray experiment, only those transcripts for which there are probes on the microarray chip are included in the output. In addition, since the entire sequence of each RNA-Seq read is included in the data output, RNA-Seq can be used to help distinguish isoforms of transcripts.

Transcriptome discovery via RNA-Seq thus opens new areas for investigation that were inaccessible via microarray studies. It is important to remember, however, that not all transcribed mRNA are actually translated into proteins.

RNA-Seq provides one set of data, but to gain a full picture of how the cell uses its genetic information, ideally, proteomics studies would also be done. By comparing the set of transcripts to the set of translated proteins, information can be obtained on post-translational modifications and on non-coding mRNA. However, proteomics technologies have not advanced at the same pace as sequencing technologies have, and remain costly.

2.2.2 The Illumina Platform

The Illumina sequencing platform has long been a standard in the next-generation sequencing field (32).Illumina uses reversible terminator-based sequencing to synthesize short reads of 75 to 100 bases (33). These modified nucleotides contain

fluorescent labels that are cleaved after each sequencing cycle and that do not interfere with subsequent chain extension (33). Illumina machines generate FASTQ files (32). The FASTQ format, an extension of the FASTA format that includes a quality component for each nucleotide, has become a commonly used file format for sharing sequencing data. (32). The quality score most often included in a FASTQ file is the PHRED score, which is calculated using an estimate of the probability of an erroneous base call determined (32).

2.2.3 Processing of RNA-Sequencing data

In analyzing RNA-Seq data, the first step is to filter out poor quality reads. Once this has been done, the remaining reads can be aligned to a reference genome or used to perform *de novo* assembly of the genome. Differences seen between the experimentally-obtained reads and the reference genome are potentially variants in the genomic sequence. Deeper coverage correlates with greater confidence in the observations made (34). Depending on the study being conducted, it may make sense to filter the RNA before even carrying out the sequencing. When analyzing the total RNA content of a cell, more than 80% of the reads may come from ribosomal RNA (26). Because such a high proportion of RNA reads come from mitochondrial and ribosomal sources, fewer reads are obtained from other RNAs. Increasing coverage to increase the numbers of these other RNA types also increases the cost of the experiment. Filtering for polyA RNA can reduce the number of rRNA reads, as can filtering to remove rRNA directly (26).

The Integrative Genome Viewer (IGV) (35) is one of several freely available tools that graphically display RNA-Sequencing reads aligned to the genome. The scale of the view provided can be adjusted so that the user may obtain a panoramic view of the overall expression level pattern or may zoom in to examine the actual nucleotide sequences and note the location, type, and quantity of variants. However, to obtain the maximal benefit from these tools, accurate annotations of existing reference sequences are necessary.

Accurate annotations are also critical in collecting gene expression counts data from an RNA-Sequencing experiment. Software that generates counts files, such as HTSeq (36), like genome viewing software, also relies on accurate annotation data. Typically, these programs reference data contained in an annotation file to know which reads to attribute to which coding region.

2.2.4 Limitations of RNA-Sequencing

Manipulations of the RNA sample must be conducted cautiously, since removal of part of the data set can potentially skew the proportions of the RNA detected and give unreliable information on the levels of gene expression. Most sequencing experiments require that the nucleic acids be fragmented, as the next generation sequencing methods generally require samples under 500 base pairs in length (37). The fragmentation process itself negatively effects the ability of the experiment to detect shorter RNAs, such as non-coding RNAs (26). Selecting for polyadenylated RNA has the potentially undesirable effect of eliminating noncoding RNA transcripts from the sample set (26). Finally, rRNA depletion efficiency is dependent upon the quality of the RNA sample and the number of probes used in the removal process (26). Paired-read methods, which can provide useful information on splicing, involve ligation and amplification steps that can introduce errors and potentially return numerous artifactual chimeras (26).

Another issue comes to light when attempting to compare data across experiments. First, in order to compare data from different experimental runs, RNA-Seq results need to be normalized. This is commonly done by multiplying the read counts by some constant factor to generate 1,000,000 reads per sample (reads per million RPM). This value is then subjected to another normalization step to provide a value for gene expression in terms of reads per kilobase per million mapped (RKPM). However, there is not clear agreement that presenting data in terms of RKPM is the best method to use to normalize RNA-Seq data (32, 38, 39, 40).

Another normalization method proposed for use with RNA-sequencing data is the Trimmed Mean of M-values (TMM) method (41). This method uses the raw data and estimates scaling factors in order to ensure that genes with the same expression level in two samples will not be flagged as being differentially expressed. To this end, the method starts with the base assumption that most genes are not differentially expressed and compares the overall expression levels of genes between different samples (41, 42). Estimates of the ratio of RNA production between samples are calculated using a weighted trimmed mean of the log expression ratios (the M-values). When used on a data set containing more than two samples, the method selects one sample to serve as a reference for the others. TMM scaling factors are then calculated using this reference. Data are actually doubly trimmed using this method. The M-values are trimmed by a default 30% and the absolute expression levels are trimmed by a default 5% (these defaults are typically adjustable by the investigator running the analysis) (41). In cases where the gene expression for a given sample is zero, the trimming is done in advance of calculating the log-fold change in expression. The TMM normalization method has been found to be robust in situations where multiple genes may have low or zero counts, which may make it more suitable to studying a virus that typically exists in a latent state, such as KSHV (41, 42).

2.3 KSHV Transcriptome Analysis

In studying KSHV gene expression, researchers are faced with several challenges. The first issue to consider is that the virus enters a latent state shortly after infecting the host cell, and few viral genes are expressed. Another issue is that there is not an ideal model system with which to study KSHV. In tumors and in infected cell cultures, the virus enters latency, making the effects of specific viral genes difficult to study. To overcome this limitation, cells infected with KSHV in culture are typically induced to enter the viral replicative stage, either with chemicals such as sodium butyrate or the phorbol ester 12-0-tetradecanoyl phorbol 13-acetate, or via overexpression of ORF50, the replication and transcriptional activator protein (Rta). Another approach has been to study closely related viruses in macaques (44, 45, 46), but this has its own set of limitations and

challenges. Consequently, very little research has been done on the natural state of KSHV in the context of a tumor, and there is much that remains to be discovered in how the virus replicates in an actual tumor. By using both cells from KSHV tumor biopsies and cells cultured in vitro, but not treated to enter the replicative state, our lab seeks to understand the patterns of “natural” viral gene expression.

KSHV has a large genome containing over 80 coding regions (8, 20, 26). In addition, the potential for variability in viral gene expression is further increased by the existence of alternate splicing patterns, alternate transcription start sites and polycistronic messages (47, 48, 49, 50). This variability in gene expression patterns can be seen when comparing the gene expression patterns from different cell types or when comparing expression *in vitro* to expression *in vivo* (31). Tools that can be used to quickly and easily look at different data sets, or that help present large complex data sets in a “condensed” form are extremely useful in conducting exploratory data analysis and in providing insights into areas of research that may be worth pursuing further.

3. Specific Aims

The analysis of biological data has become increasing complex given the magnitude of the data sets generated by such techniques as high-throughput sequencing. It has become virtually impossible to conduct biological research without computational assistance in managing and processing these large datasets. This can present challenges for biological researchers, many of whom have neither the time nor the inclination to become expert programmers in addition to excellent bench researchers. One response to addressing the computational demands associated with high throughput sequencing is to form interdisciplinary research teams or collaborations. Ideally, in this way, the complementary skillsets available to the team would seamlessly cover all potential needs. In practice, however, certain computational analyses are carried out with such frequency that creating tools that enable all team members to conduct them easily makes sense, despite the initial effort required to set up the necessary infrastructure. The specific aims of this thesis project are as follows:

I. Aim 1: Design and build web applications using the R programming language that facilitate the transcriptome analysis KSHV RNA-Sequencing data. To this end, applications will be made that permit the user to construct customized box plots and to conduct principal component analyses via a graphical user interface (GUI). The code used to create the applications will be made publicly available on GitHub, allowing for easy download by any investigator who may find them useful.

II. Aim 2: Write “User’s Guide” documents detailing the features and use of the Shiny applications described in Aim 1, above. These guide will also include information on the systems setup required to run the applications.

III. Aim 3: Utilize Next-Generation RNA-Sequencing techniques to characterize the viral RNA transcriptomes of KSHV-infected cutaneous Kaposi’s Sarcoma (KS) tumor specimens and of cultured blood and lymphatic endothelial precursor cell specimens. Conduct a comparison of the viral gene expression patterns seen in both the in vivo and the in vitro specimens.

4. Methods

4.1 Shiny Application Design

The Shiny package is a web application framework for use with the R programming language (55, 57). The goal in designing these Shiny apps was for the layout to be uncluttered so that users would be able to quickly and easily understand how to interact with each app’s components. As much as possible, the layout followed the same structure in each app. The arrangement of application elements has a title and tabs to different application pages arranged horizontally at the top of the application (Region A). Below this header region, a sidebar panel (Region B), primarily containing Shiny widgets requesting user input is located to the left and a larger main panel (Region C) for displaying data and plots is located to the right. This layout is illustrated below in Figure 1.

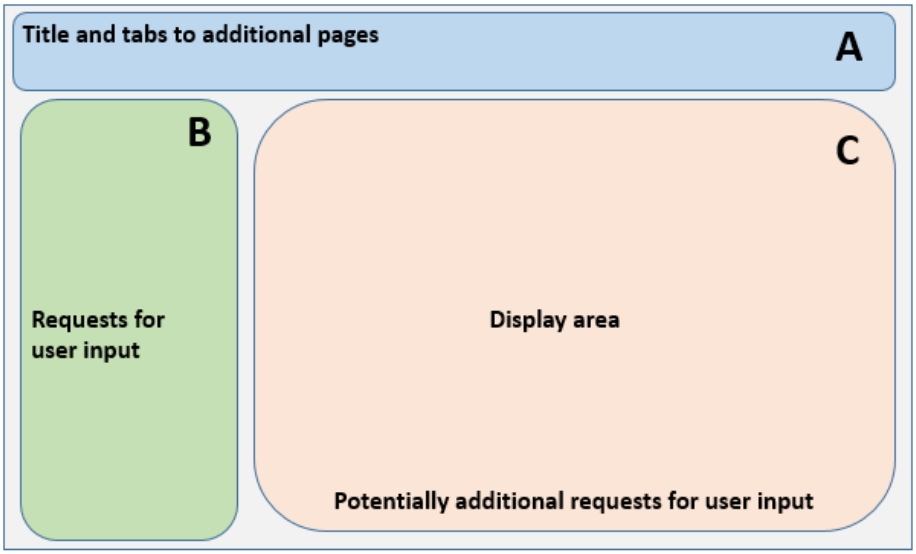


Figure 1: Shiny Application Layout
Region A - title and tab information.
Region B - primary user input area
Region C – primary output display area

This layout uses Shiny's default application layout theme, which gives it the additional benefit of being relatively simple to implement, while at the same time providing a clean, intuitive user interface design.

4.1.1 Consistency of Design between Applications

Both applications share a number of features. In addition to standardizing the layout used in the applications, whenever possible, the arrangement and function of the Shiny "widgets" used in the sidebar panel is similar, if not identical. The following sections describe components of the sidebar panel that are similar or identical in both of the applications. These features can be seen in the screenshots provided in Figure 3 and Figure 4.

i. Uploading a Data File

In both applications, the first Shiny “widget” in the sidebar panel is the file upload box. This enables the user to select a data file containing the data to be processed by the application from any location in their local file system. The application can upload data files in the following formats: comma-separated values (.csv or .txt), tab-separated values (.tsv or .txt), and semi-colon separated values (.txt). The selected data file can be changed at any point while using the applications by simply selecting a new data file to upload.

Both applications require that uploaded data files conform to a specific format. Specifically, neither the top-left cell nor the bottom-right cell of the data file may be empty (shown highlighted red in the figure), although the words ‘Sample’ and “filler,” themselves, are not critical. In addition, supplying type data is optional. The decision to have the applications accommodate type data in the same file as the actual numerical counts was intended to support data “cohesiveness” and to minimize the need for multiple files to completely describe a single set of data. However, if such data are included in the data file, they may only be positioned in the last column (for categorizing row elements) or in the last row (for categorizing column elements. The word “Type” as either the column header and/or the row name, as appropriate, is required to identify type data to the application. An example of acceptable file formatting is displayed in Figure 2. Additional permitted variations of this format are illustrated in the User’s Guides provided in Appendix A.

Sample	Gene1	Gene2	Gene3	Gene4	Gene5	Type
1_A	100	432	22	253	415	sampleTypeA
2_A	256	36	85	112	417	sampleTypeB
3_A	113	576	26	76	373	sampleTypeC
4_A	76	47	36	97	210	sampleTypeA
5_A	34	12	34	45	211	sampleTypeA
6_A	0	8	896	13	110	sampleTypeB
7_A	24	11	26	96	142	sampleTypeC
8_A	487	6	345	35	143	sampleTypeC
9_A	224	30	117	18	322	sampleTypeC
10_A	53	76	32	86	17	sampleTypeA
Type	geneTypeA	geneTypeA	geneTypeB	geneTypeB	geneTypeA	filler

Figure 2: Proper Formatting of Data
There must be text in the top-left and bottom-right cells. However, the actual words "Sample" and "filler" are not critical and may be changed. Including type data is optional, but if it is included for either samples or genes, the word "Type" is critical, as the data will not be processed appropriately if this keyword is missing. Permitted variations of this file format that omit part or all of the type information are included in the User's Guide (included in Appendix A).

ii. Data Orientation

The applications process the data in an orientation-specific manner. For the Box Plots Application, the data may need to be transposed if the desired elements (typically genes) are not appearing in the drop down menu. In the Principal Components Application, the application is designed to work with data in any orientation, i.e. either with sample data in rows and gene data in columns or with these positions reversed. The rationale behind this flexibility was that clustering both to group samples with similar gene expression patterns together and to group genes with similar expression patterns across samples could provide useful information to an investigator. Toggling between both of these scenarios is thus facilitated by the application, with no preference given to either. However, in order to function properly, the applications must know whether data for genes is arranged in rows or columns. A radio button in the left sidebar panel allows

the user to enter this information in for use by the program. Inaccurately entering in data orientation information may lead to invalid results being displayed by the applications or even in abrupt program termination.

iii. TMM Normalization

This radio button option lets the investigator choose whether to look at the data with or without trimmed mean of M-values (TMM) normalization being applied. Normalization in the context of RNA-Sequencing data analysis is discussed in detail in section 2.2.4. Allowing the user to determine whether or not to normalize the data before plotting enables the visualization of raw counts data or of data normalized prior to being uploaded to the application.

iv. Displaying the Uploaded Data

After the application uploads a data file, it processes the file to extract type information, if supplied, and to ensure that the data are formatted appropriately for the analyses to be performed. Once the data are reformatted appropriately, the data set is displayed in tabular format at the bottom of the application. This lets the user view the reformatted data and ensure that the data file used is correct.

v. Downloading Plots

Both applications are capable of producing picture files (.pdf format) of the plots displayed. To create such a file, click the “Download” button located below the plot.

4.1.2 Application 1: Box Plots Application

The first Shiny application provides users with the ability to generate custom box plots using data from an uploaded data file. Box plots and their interpretation are discussed further in the User’s Guide in Appendix A. Briefly, a box plot provides a “snap shot” of a data set. The central “box” gives the range occupied by the middle 50% of the values in the data set. The horizontal line through the box represents the median of the data set.

The two vertical lines emerging from the top and bottom surfaces of the box represent the highest and lowest non-outlier values, respectively. Outlier values in the data set are represented by points above and below these vertical lines.

Typically, the box plots generated by this application would be used to view the variability in expression levels of a specified set of genes across all experimental samples. The number of genes for which data can be displayed in one plot is limited by the maximum plot width generated by the R plotting function. While items may continue to be added, spacing of the box plots will become cramped, impeding legibility, after about 12 to 16 individual box plots are present in the plot. The user can select the genes to display from a drop down menu generated when the data file is uploaded into the application. Multiple genes can be selected for plotting on the same box plot. Selected genes do not need to be selected in the same order that they appear in the drop down list, nor do they need to be selected as consecutive “blocks” of genes. As genes are selected, the box plot display dynamically adjusts to update the figure. Once gene selection has been concluded, clicking outside the drop down menu will cause the menu to close. It may be reopened at any later point, allowing for modification of the list of genes selected, with concomitant replotting in the display.

The initial plot generated by the application is scaled along the y-axis so that all data, including extreme outliers, are displayed. Given the extreme range in expression levels from gene to gene within a sample and from sample to sample, the application provides a slider bar widget that allows for customization of the y-axis range. This will enable the user to “enlarge” box plots that would otherwise be compressed due to the presence of outliers in the plot data and also provide the means to standardize the y-axis across multiple plots. Changing the “Set Custom Y Range” radio buttons from the “No” default to “Yes” causes the slider input widget to appear, along with a duplicate box plot towards the bottom of the display. The y-axis readjustment will affect only this second plot, allowing the investigator to efficiently compare both unscaled and scaled portrayals of the data. The Box Plot Application’s features can be seen in the screen shot in Figure 3.

Figure 3: Box Plot Application Screenshot
The sidebar panel features and a representative box plot are displayed. Not shown are the slider input widget and the second box plot that would be displayed once the "Set Custom Y Range" selection was switched to "Yes."

4.1.3 Application 2: Principal Component Analysis Application

The second Shiny application makes conducting principal component analysis convenient to do. Principal components analysis is a technique used to reduce the complexity of a multi-dimensional data set. In essence, the data points are transformed so that the majority of the information contained in the data set can be represented

using a small number of “principal components.” The plots generates by this application are generated by using the first two principal components.

Once the data file is uploaded, a plot of the first two principal components, PC 1 and PC 2, appears in the main panel. The default setting is for the plot to display the elements stored in columns in the data file. The rationale behind this choice is that this is typically how sample data are arranged in our lab. To see the row elements displayed instead, the user need only check the appropriate radio button in the sidebar panel. Thus, the decision of whether to arrange sample data by rows or columns does not affect the utility of the PCA application.

If type information is supplied for either the column elements or the row elements, the corresponding PC1 versus PC2 plot can be displayed with points that are colored and/or shaped to reflect the type category of each point. These options are enabled by selecting the “Color” radio button and the “Shapes” checkbox, respectively. If these options are selected in error, the application will display a notice that the data do not support them, and the plot display will remain unchanged, with the default round, black data points. A legend providing a guide to the colors and/or symbols used is automatically generated and displayed to the right of the plot. If no type information is provided, the application provides a basic black-and-white default plot with round data points.

Mousing over the plot causes the cursor to transform to a small cross. Using this cross to locate and click on a given point in the plot will result in information about the selected point being displayed in the grey box located below the plot. This information is drawn from the principal component analysis output and the data file and includes the plot coordinates of the selected plot and its sample identifier or gene identifier. In Figure 4, note that the information displayed in the grey box below the plot is giving the sample ID and plot coordinates of the green triangle located in the lower-left corner of the plot.

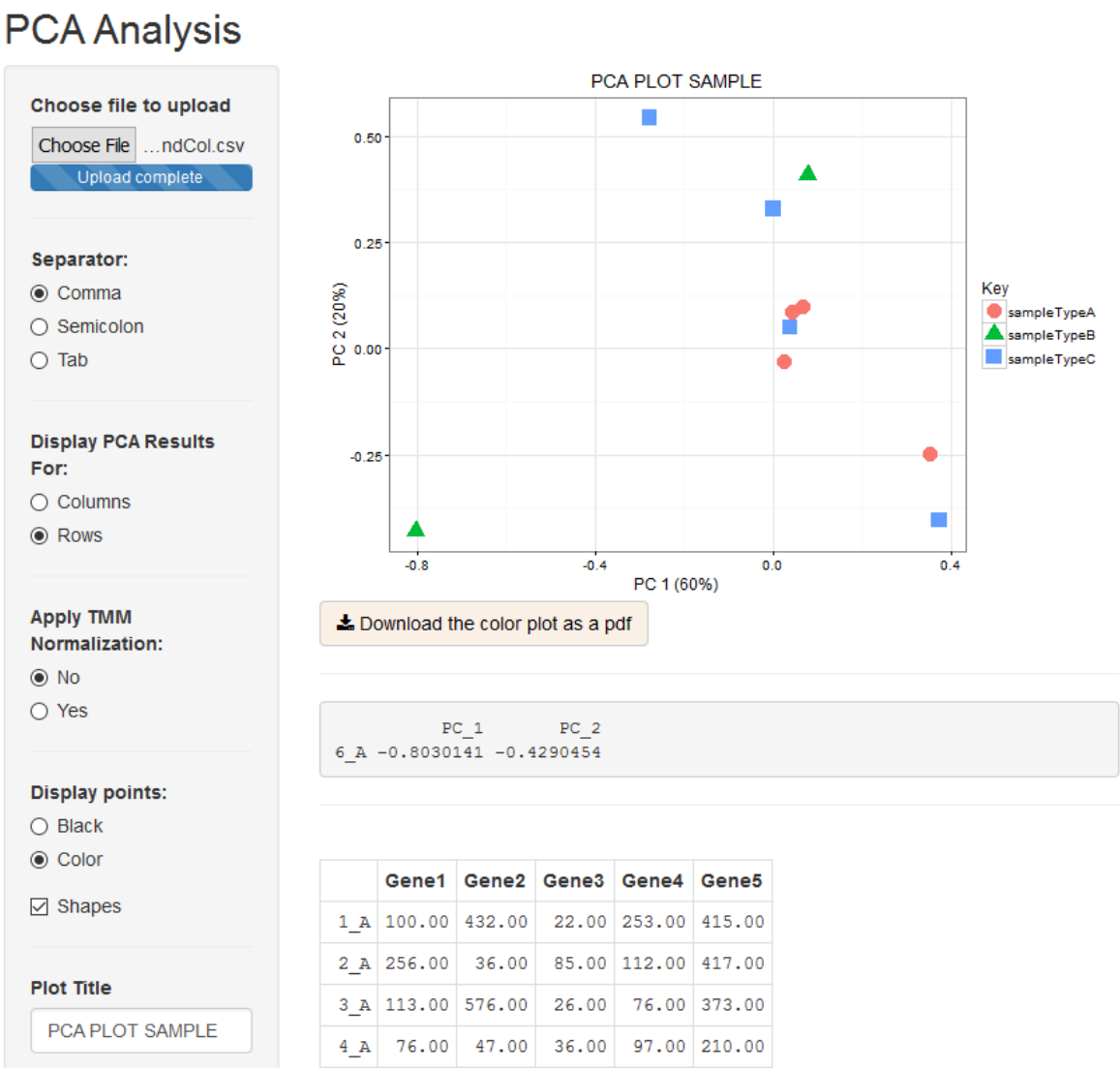


Figure 4: Principal Components Analysis Application Screenshot
The sidebar panel features and a representative PC 1 vs. PC 2 plot are displayed. Note the grey box below the plot that displays name and coordinate information for selected point(s). Currently, this section is displaying information on the green triangle in the lower left corner of the plot.

4.2 RNA-Sequencing and Generation of Gene Expression Counts Data Sets

RNA-sequencing data were obtained from 45 punch biopsy specimens taken from KS tumors. Biopsies were performed on 26 patients (21 male and 5 female) from the Uganda Cancer Institute in Kampala, Uganda, with some patients providing multiple biopsies. All of the patients were co-infected with KSHV and HIV. Each tumor biopsied

had one of the following three morphologies: macular, nodular, or fungating. Two biopsies were taken from patients who had received anti-retroviral therapy.

Sequencing was done on the Illumina platform. While total reads (host cell reads and viral reads, combined) numbered approximately 100 million per sample, the number of viral reads varied greatly from sample to sample, averaging approximately 100,000 across all the samples (range: < 50 to > 800,000). Read depth ranged from 0 to 20,000.

We were also fortunate in being given access to RNA-sequencing datasets from infected blood (BEC) and lymphatic (LEC) endothelial precursor cells cultured in the Lagunoff lab at the University of Washington, Seattle. These cells are significant because KSHV may induce angiogenic phenotypes that promote tumorigenesis (14). These cells were also studied in a “natural” state, i.e. without exposure to chemical inducing agents or overexpression of ORF50 (Rta).

Analyzing the RNA-sequencing data requires a multi-step process. Programs were either run via the command line or via a local installation of the Galaxy platform (51) or via a web-based Galaxy instance. First, FASTQ Groomer (52) was used to ensure read quality. TopHat2 (53) was then used to align reads to a reference genome. We used the NCBI reference sequence for KSHV, NC_009333.1 (54), along with either its associated GFF3 annotation file (56) or a custom annotation file prepared by our lab. The customized annotation file was designed to eliminate the ambiguity that exists in attempting to assign reads to coding regions in KSHV. Specifically, the customized annotation file excluded regions where reads could potentially be claimed by multiple transcriptional regions as a result of polycistronic messages, alternate splicing, alternative transcription start sites, and the like. The GFF files were used to visualize the read distributions in a viewer such as IGV (described earlier in Section 2.2.3) and to generate gene expression counts file using the HTSeq program (36).

4.3 Application Testing

The goal of a project such as this one is to produce a useful and functional application. Ideally, the program would be robust and able to gracefully handle incorrect user inputs. These applications were tested with all purportedly legal file formats using small, “toy” datasets. The actual test files used are provided in the supplementary materials and described in Appendix B. In brief, the applications were tested with files of the following formats:

i. No type data present.

ii. Row elements’ type data present, stored in last column.

iii. Column elements’ type data present, stored in last row.

iv. Type data present for both row and column elements.

Each of the formats listed above was tested as a comma-separated values (.csv) file, a tab-separated values (.tsv) file, and a semicolon-separated values (.txt) file. In addition, each file was tested with sample data in rows and gene data in columns and, conversely, with gene data in rows and sample data in columns.

After verifying that test data files following these permitted formatting options uploaded correctly and produced the requisite results, the applications were used to analyze lab-generated data files. These “real-world” data files were generated from Excel spreadsheets containing the counts files obtained from the RNA-Sequencing data sets. The files were saved from their Excel spreadsheets as .csv files, with samples arranged in columns and genes arranged in rows.

5. Results and Discussion

5.1 Shiny Applications

Both the Box Plot Application and the Principal Components Analysis Application are complete and functioning as intended. The code for the applications has been placed online in the GitHub repository hosting service, where it is freely available for download under an MIT license (meaning that there are no restrictions placed on the use of the code, and also no liability to the original author of the code related to its use). User's guides for both applications have also been placed on GitHub.

5.2 Using the Shiny Applications

5.2.1 Box Plot Application

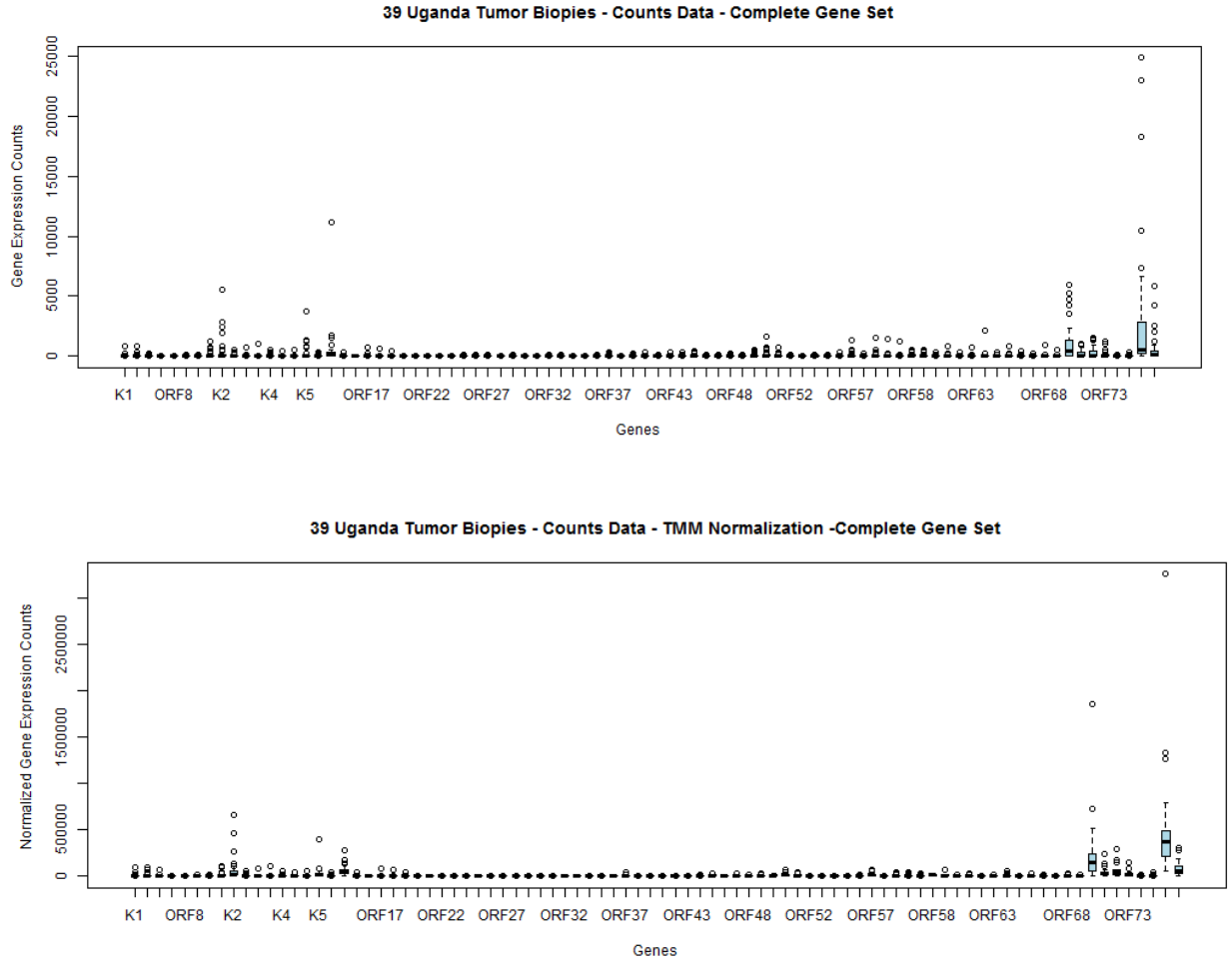


Figure 5: Boxplots of 39 tumor biopsies.

These plots include all KSHV genes in the Human Herpesvirus 8 reference standard (NC_009333.1). The data used to generate the top plot has not been normalized. TMM normalization has been applied to the data shown in the second plot.

At this level of detail, normalizing the data before plotting does not seem to make much of a difference. However, scaling the results to account for the total number of reads obtained for each sample allows us to better compare the samples to one another. In recognition of the importance of normalization, from this point on, plots will display only normalized data. As there is not general agreement on the optimal normalization method to use for RNA-Sequencing data, three normalization methods will be considered: 1) TMM (Trimmed Mean of M-values), 2) RPKM (Reads Per Kilobase per Million Mapped), and TPM (Transcripts Per Million).

TMM, discussed in Section 2.4.4, does not adjust for differences in gene lengths. RPKM and TPM do consider gene length in addition to read depth. The difference between these two methods comes from the order in which each normalization step is done. In RPKM, read counts are first normalized for sequencing depth and then normalized for gene length. This results in differences in the normalized “total reads” count. In the TPM method, the first step is to normalize for gene length, and then to normalize for sequencing depth. The result of switching the order of these steps is that the normalized “total reads” count becomes constant across samples, allowing for more straightforward sample-to-sample comparisons of relative gene counts. The Box Plot Application and the PCA Application will be used to demonstrate the effect of each normalization method.

By including all the genes in the counts file, an overview of gene expression across the entire genome is obtained. We see that there is significant expression of genes from the “latency locus” at the right side of the plot, but there is also some expression from genes outside this region towards the left side of the plot (Fig. 5). However, the scale of the plot makes identification of these genes difficult. Accordingly, a new plot, shown in Figure 6, was generated to focus in on the genes at the left end of the genome.

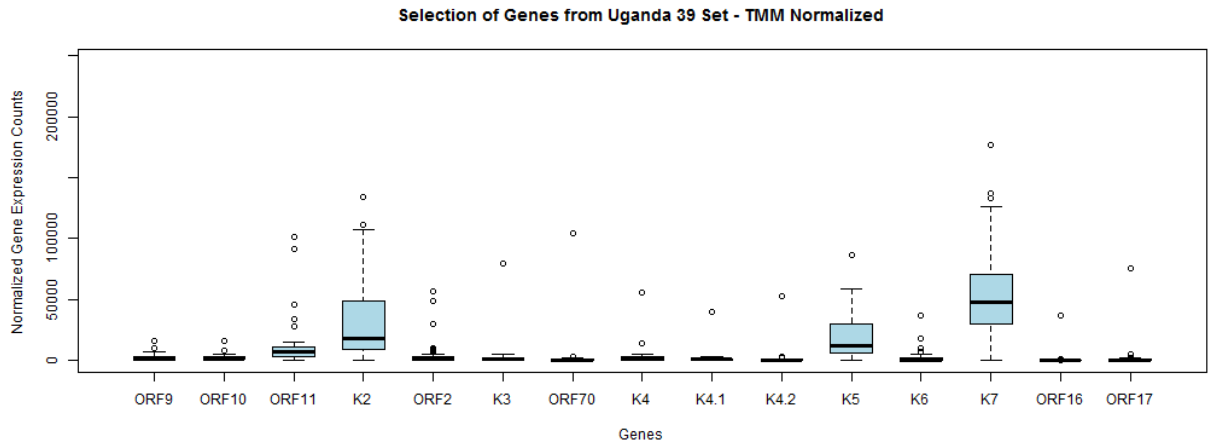


Figure 6: Selection of genes chosen for closer examination of their boxplots. The y-axis range was custom-selected to enhance box plot visibility. Some high outlier values were excluded as a result.

This closer view of the expression data shows that there are reads being produced from the ORF11, K2, K5, and K7 gene regions. All of these genes are generally considered “lytic” genes (1), although it has been previously noted that K2, which encodes a homolog of IL-6, may be expressed independent of other lytic genes and has been seen expressed in KS along with the canonical latency gene LANA (ORF73) (1). These genes could be studied further to determine the likelihood that they produce proteins that have a function in latency that could account for possible transient expression. K5 is known to be involved in immune modulation, while K7 has anti-apoptotic function (1). These pro-cell-survival functions could contribute to the prolonged survival of the latently-infected cell. It is possible, however, that the expression of these lytic genes is due to the 1% to 5% of tumor cells that typically are in a reactivated state. During lytic replication, expression of all viral genes would be expected, but it is possible that only a subset of genes is expressed high enough levels to be detected, given that the vast majority of the virally infected tumor cells would be in the latent state.

Looking more closely at the latency locus (Figure 7), we observe that ORF73, which is generally believed to be constitutively expressed, does not have many read counts. This, of course, does not imply that LANA, the protein encoded by ORF73, is not

present. Since ORF73 protein is detected in most tumor cells, the ORF73 mRNA may be continually expressed at low levels or is transiently expressed, while the protein is stable.

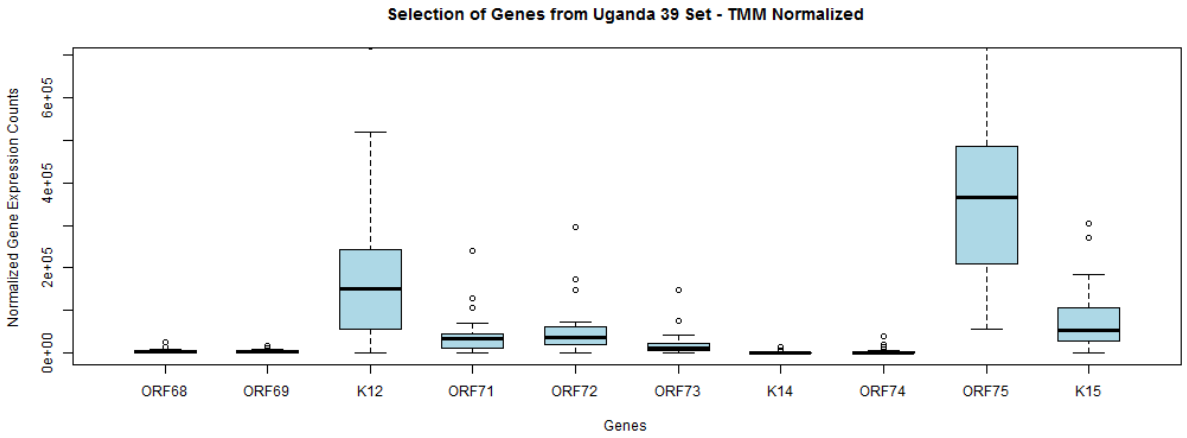
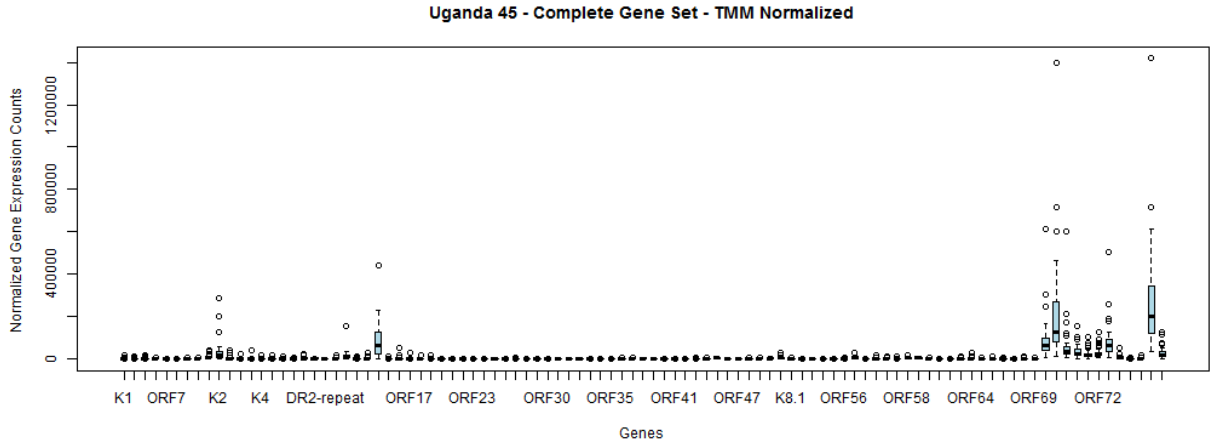


Figure 7: Closer view of the region containing the latency locus.
Of note is the relatively low expression of ORF73, the gene that encodes LANA, the classic marker of latency.

The plots shown to this point were all generated from counts determined using the NC_009333.1 reference sequence and GFF obtained from the NCBI website. The following plots were prepared from a larger dataset of 45 biopsy specimens using our lab-generated, custom GFF genome annotation file. Again, in Figure 8, we'll look at normalized plots of the complete gene set to get a sense of gene expression across the entire genome.



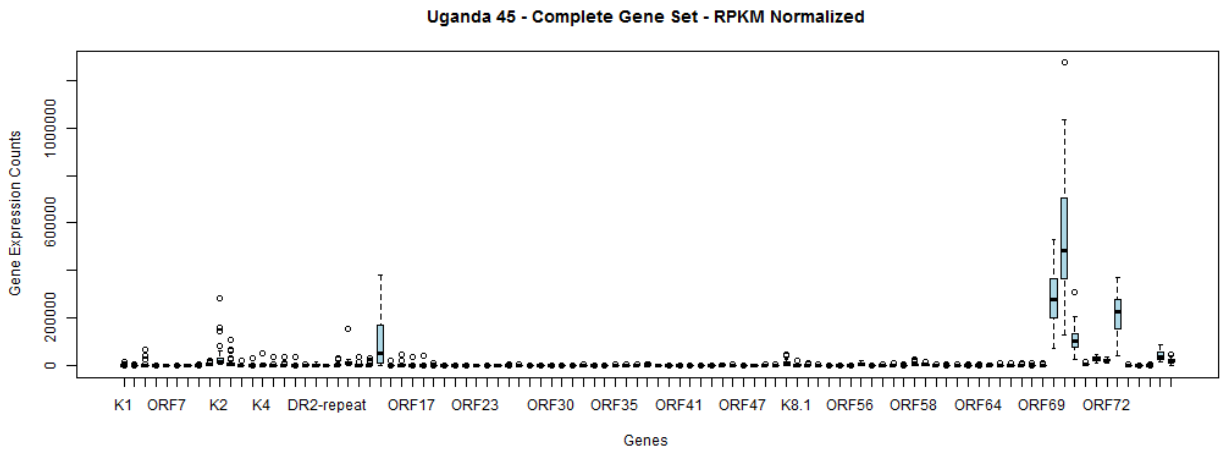


Figure 8: Normalized gene expression boxplots for tumor biopsy samples. The top plot was generated using data normalized with the TMM method, while the bottom plot uses data normalized with the RPKM method.

The top plot was normalized using the TMM method, which calculates scaling factors based on the expression levels of each gene and the total reads in the library of each sample, while the lower plot was normalized using the Reads Per Kilobase per Million mapped reads (RPKM) method. The RPKM method was developed to facilitate comparisons both within and between samples and normalizes gene counts to correct for differences both in library sizes and in gene lengths (42). However, the RPKM method has been shown to introduce a bias in the per-gene variances, particularly in genes expressed at low levels (42). This issue must be taken into consideration given the relatively low numbers of viral reads obtained from each biopsy's sequencing data.

There is no consensus on what the "best" normalization method might be for RNA-sequencing data in general (42), but the choice of normalization method can significantly affect the results obtained. Looking more closely at the areas with

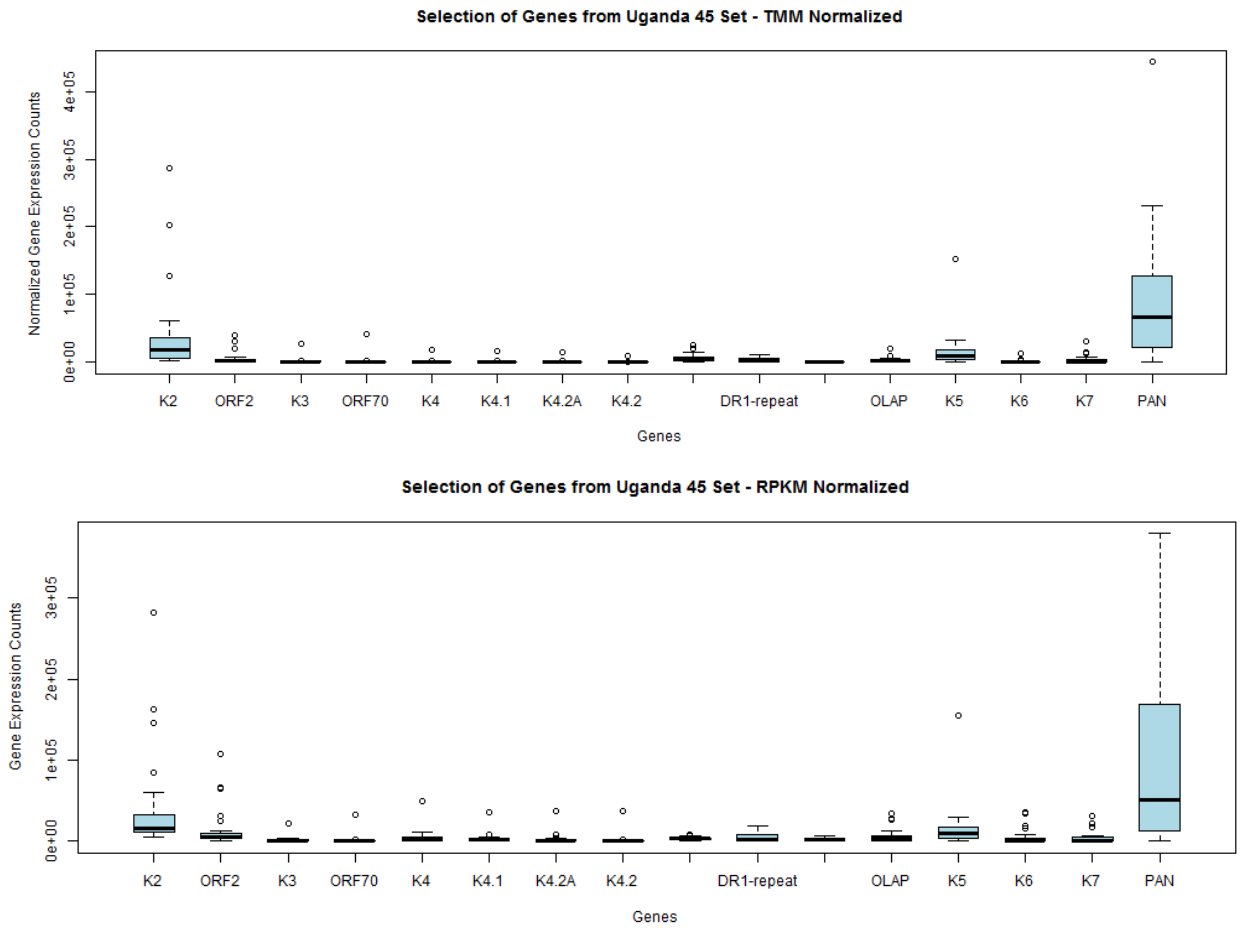


Figure 9: Comparing TMM normalization to RPKM normalization, K2 through PAN. For this region of the genome, the choice of normalization method does not appear to cause a significant difference in the relative expression levels seen.

significant gene expression occurring, we see that for the genes towards the beginning of the genome (Figure 9), the same genes appear to be expressed at similar levels, regardless of method used. For genes near the end of the genome (Figure 10), some notable differences stand out. ORF75 appears to be expressed at a significantly higher level than other genes when TMM normalization is used, and at a much lower level, when RPKM normalization is used. Conversely, K12 (K12A and K12Aa in the custom annotation file) expression, while evident in the TMM normalized plot, appears much more significant in the RPKM normalized plot. This difference can be attributed to gene size as the ORF75 GFF feature is 20 times larger than the K12A feature.

It is also critical to point out issues regarding validity of read counts determined using different GFF files. The GFF files contain location positions for each gene feature used to map the RNAseq reads. For complicated genomes such as KSHV, in which there are many overlapping transcripts and ORFs, care must be taken to ensure that reads mapping to a specific gene feature are actually derived from that gene, rather than to an overlapping feature. As shown above, using the NC_009333 KSHV genome Reference Sequence (RefSeq) GFF file available from the NCBI, the ORF K7 is determined to be highly expressed. In contrast, using the custom GFF that was developed in our lab to avoid overlapping features, the ORFK7 shows few mapped reads. This discrepancy is due to the fact that the poly adenylated nuclear (PAN) transcript overlaps the ORFK7 transcript and essentially all of the reads mapping to the ORF K7 feature in the NC_009333 GFF are actually derived from the PAN transcript.

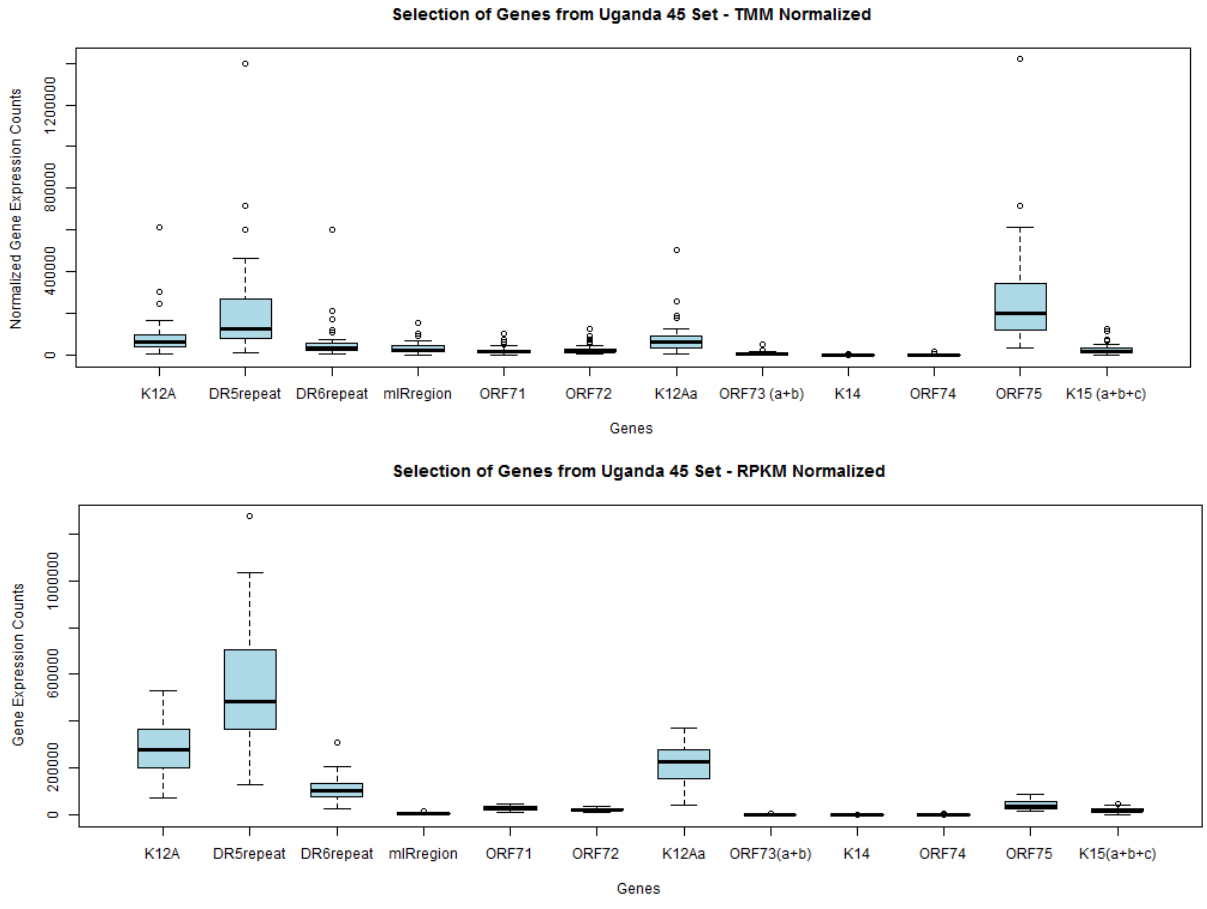
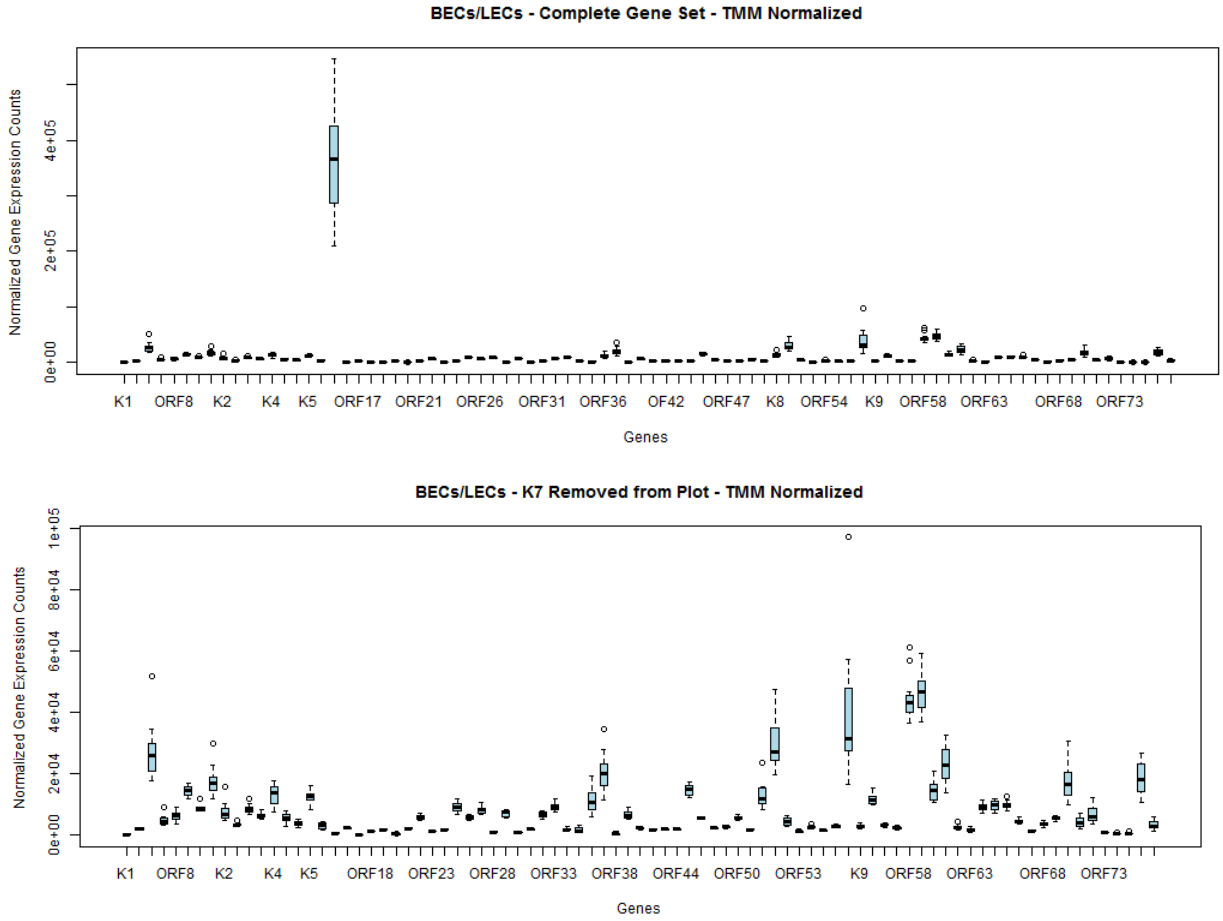


Figure 10: Comparing TMM normalization to RPKM normalization, K12A though K15. In this region of the genome, the choice of normalization method significantly affects the normalized expression levels for K12 and ORF75.

Being able to easily visualize and compare the effects of the normalization methods used can help investigators ensure that no significant results are masked by the method chosen. It would be imprudent to assume that the results obtained via one method are more valid than those from another method. Instead, the results from both procedures should be considered within their biological contexts.

The third lab-generated data set was obtained from cell cultures of blood endothelial precursor cells (BECs) and lymphatic endothelial precursor cells (LECs). These cultures were prepared to ensure that approximately 95% of the cells contained virus



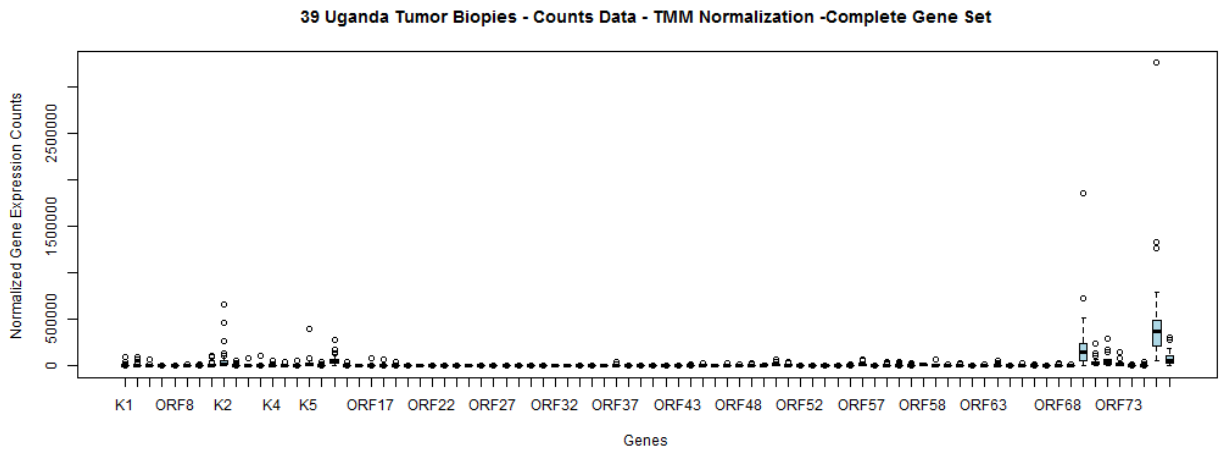


Figure 11: Normalized gene expression boxplots for cultured cell samples. The plot from the tumor biopsies is included for comparison, as the global expression pattern for cultured cells is significantly different from that for tumor cells. The box plot for K7 has been removed from the middle plot to allow expression levels of the other genes to be appreciated.

Consequently, many more viral reads were obtained in the RNA-sequencing data than were obtained from the tumor specimens, where much of the sample could have contained tissue composed of cells that did not harbor the virus.

Figure 11 shows the genome-wide expression for the cultured cells, with the tumor data presented again for comparison. In this analysis, the NC_009333.1 standard annotation was used. The most obvious difference in the two data sets is the high level of expression seen across the entire genome in the “latently-infected” cell cultures. This could provide further evidence that the latent state is more complex than generally thought, with at least transient expression of many genes traditionally considered as “lytic.” As noted earlier, however, the effect of having a small percentage of cells in a latent infection that are in the lytic phase could account for the observed results. This is an area that needs further study. Interestingly, even in these cultures displaying much wider gene expression than anticipated in un-induced cells, the level of reads originating from ORF73 (LANA) is surprisingly low (Figure 12). The kinetics of ORF73 expression also seem to warrant further study.

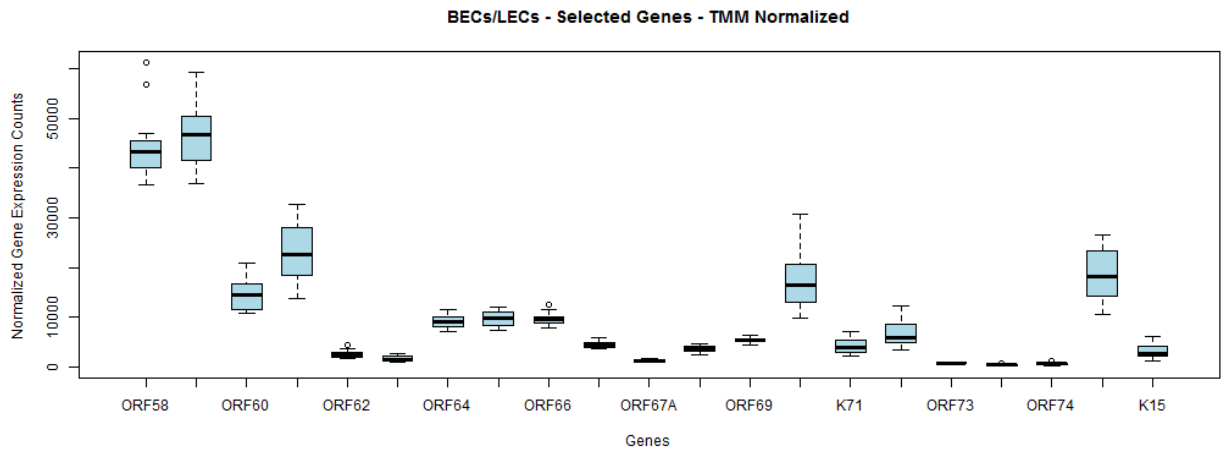


Figure 12: Closer view of the latency region in the cultured cell sample. The absence of reads from ORF73 is notable.

From the plots shown in Figures 5-12, normalizing for both gene length and sequencing depth (via the RPKM method) appears to be the better choice for KSHV viral read data than simply normalizing to reduce the effect of extreme values (TMM method). Figure 13 demonstrates how TPM compares to TMM.

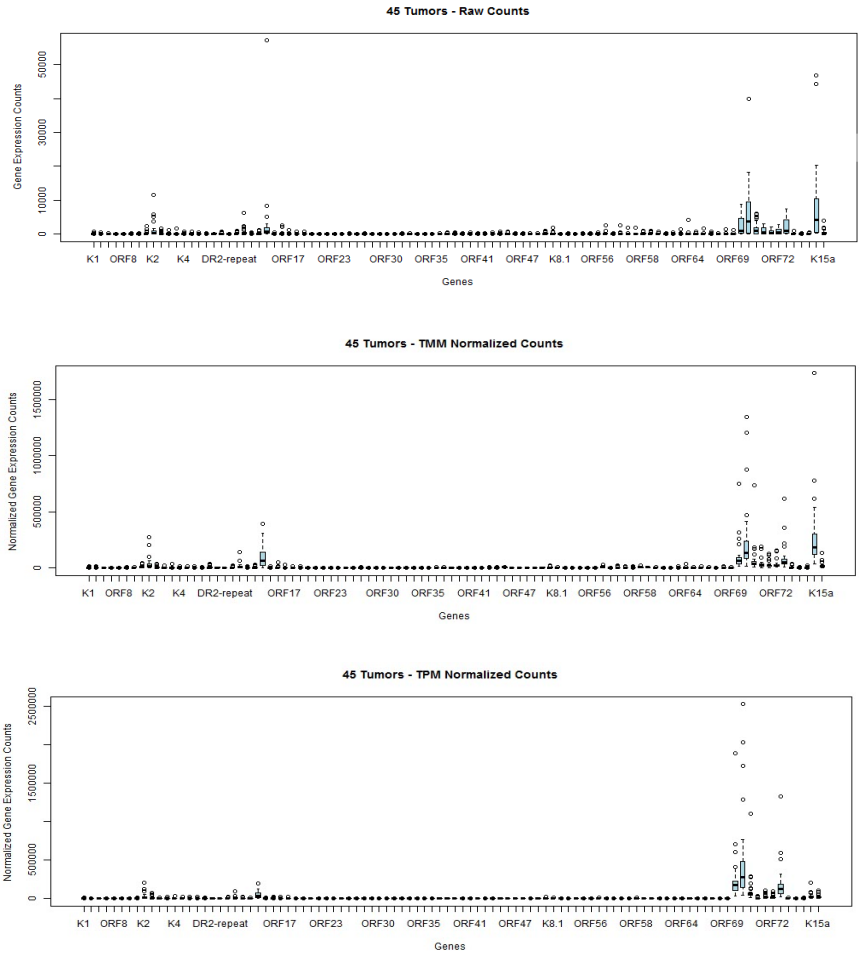


Figure 13: Comparison of TMM and TPM normalization methods.

The advantage TPM has over RPKM is in setting an even baseline for gene count comparisons. The similarity in the plots for TPM and RPKM (see Figure 10 for comparison), is indicative of the similarity in the normalization approach used by each method. TPM provides normalized counts that can be easily compared directly, as each sample is “rescaled” to the same normalized total read count value, making it the preferred method over RPKM.

5.2.2 Principal Components Analysis Application

Principal components analysis provides investigators with a way of viewing a data set of high dimensionality. The two principal component directions shown in the plots in this section are the directions in feature space along which the original data are highly variable (58). Together, they define a 2-dimensional plane which best fits the data and onto which the data are projected (58). Using average squared Euclidean distance as a measure of closeness, these plots display the best two-dimensional representations of the data set (58).

Since we have type data for the Uganda tumor biopsy specimens (morphology data), this was included in the file uploaded to the PCA application. Similarly, the cultured cell specimens had type data provided that indicated whether the viral gene counts were from BECs or LECs. Type data is communicated using different colors for the plotted points, depending on the category of the point. PC 1 versus PC 2 plots were generated using both TMM normalized data and non-normalized data, so that the effect of normalization could be seen.

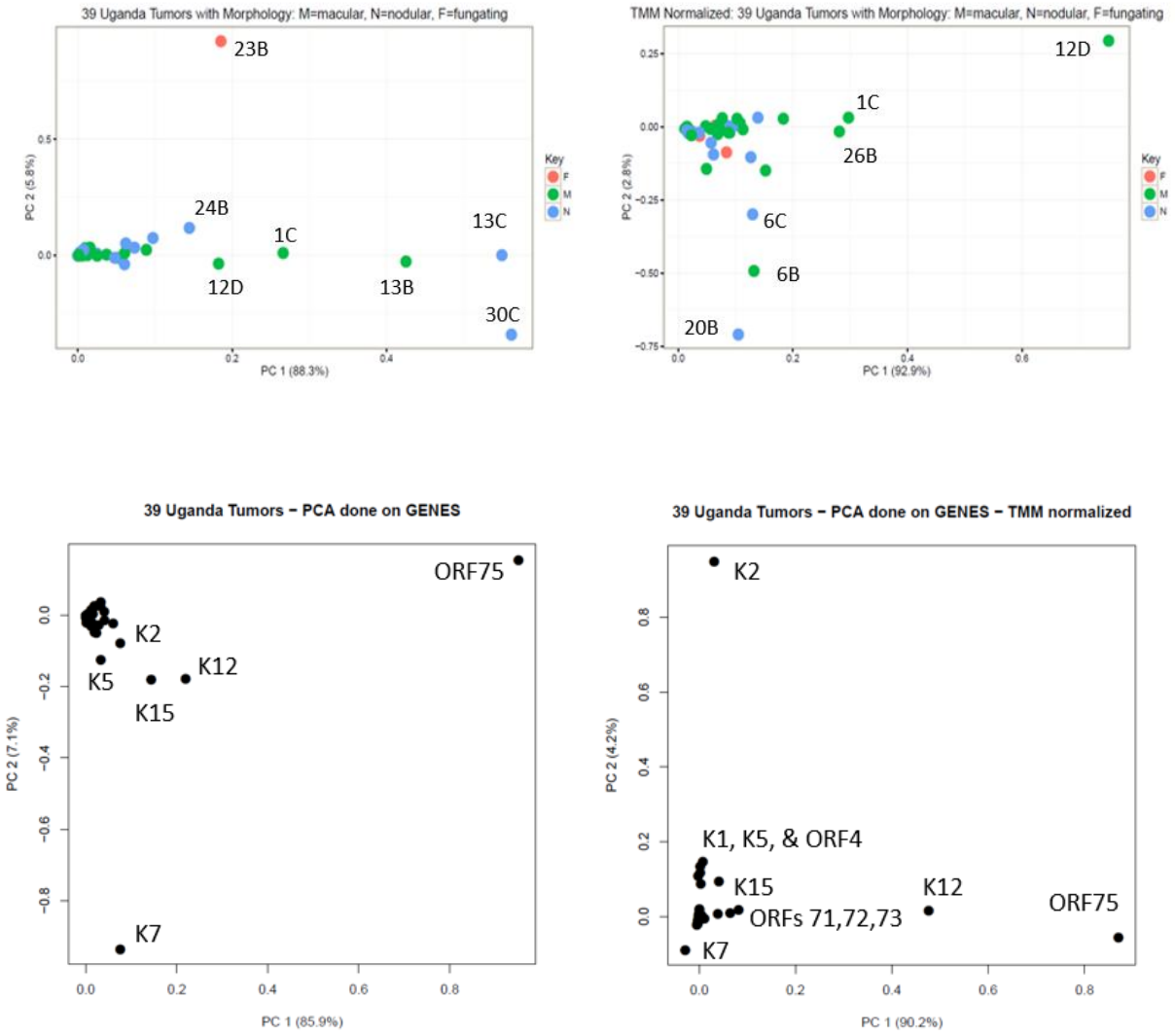


Figure 14: PCA Analysis with and without TMM normalization for Uganda tumor data.

While all of the genes not clustering together in the non-normalized plot are also present as distinct from the primary cluster in the normalized plot, the normalization procedure brings additional genes of potential interest to light. The primary cluster is likely made up of the genes with minimal to no detectable expression, and thus not of much additional interest. What is notable, however, is that ORF73 is separated away from the main cluster, along with other genes from the latency locus. In comparison, the box plot analysis had displayed lower than expected expression of ORF73. The PCA analysis suggests that even though expression levels were not as high as anticipated, there is

still a notable difference in ORF73 expression compared to the majority of the other viral genes in the tumor specimen.

When looking at the analysis done on samples, rather than on genes, there does not seem to be a recognizable pattern based on morphology – the tumor specimens did not separate into clusters based on the macular, nodular, or fungating appearance of the tumor.

The normalized plot for BECs and LECs does seem to show the samples separating to some degree based on cell type, with the BECs localizing to the lower left region of the plot (Figure 15).

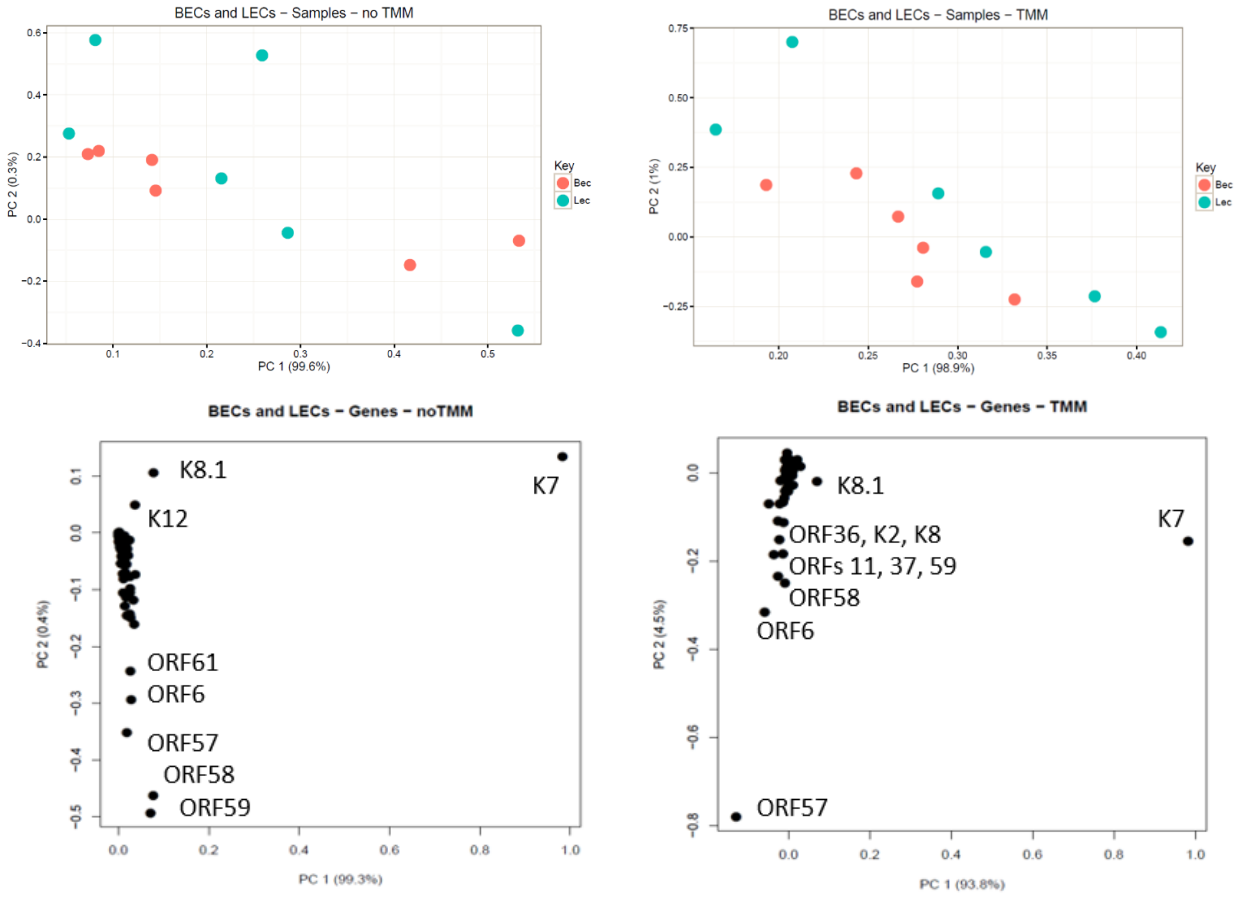


Figure 15: PCA Analysis, with and without TMM normalization for cultured cell data.

When plotting PC 1 versus PC 2 for genes, the normalized plot displays a number of genes that separate from the main cluster. This could be useful in choosing which genes to select first for further study, especially since the box plot, which showed notable expression across most of the genome, didn't provide many clues as to which genes might be most deserving of further investigation.

Counts files were also generated using the customized GFF developed in the lab. These plots are shown in Figures 16 (genes plotted in PCA plot) and 17 (samples plotted in PCA plot). For both Figure 16 and Figure 17, a plot is also included for a data set on which RPKM normalization was done before uploading the file into the application. This gives us the opportunity, again, of comparing normalization methods.

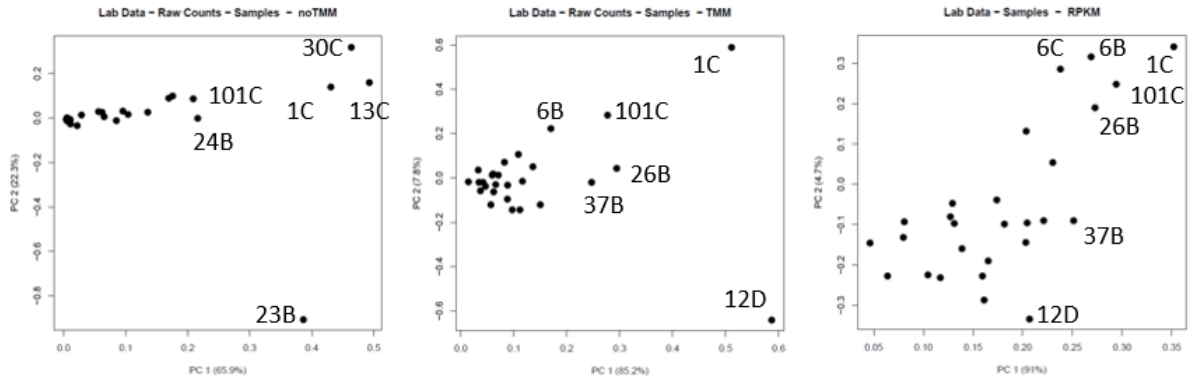


Figure 16: Counts file generated using customized GFF. Samples plotted.

It is difficult to speculate on the significance of these plots. It is likely that using non-normalized is not the right approach to take, but the two normalization methods each give quite different results. When TMM normalization is used, most samples are grouped together toward the left side of the plot, and only a small number of samples are removed from this group. When RPKM normalization is used, however, the genes seems to spread out evenly across the plot, without forming clusters at all. This again points to the necessity of considering the strengths and weaknesses of each

normalization method and realizing that while PCA can suggest potentially fruitful areas for further study, the results must be evaluated in context with other available information. Interestingly, when genes, rather than samples, are plotted, the results are much more consistent (Figure 17).

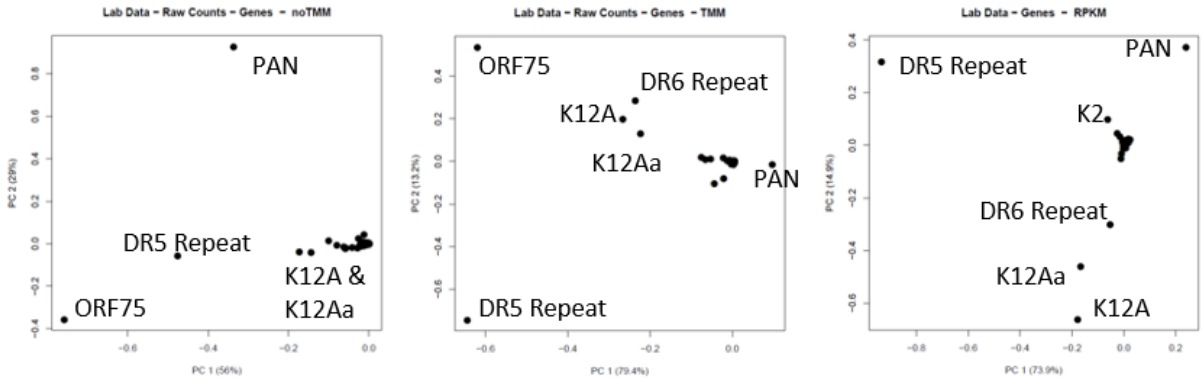


Figure 17: Counts file generated using customized GFF. Genes plotted.

In Figure 18, the effect of TPM normalization is considered. In this plot, tumor specimens were classified based on the total number of raw viral reads obtained from each specimen. The plot using non-normalized data show that the samples separate

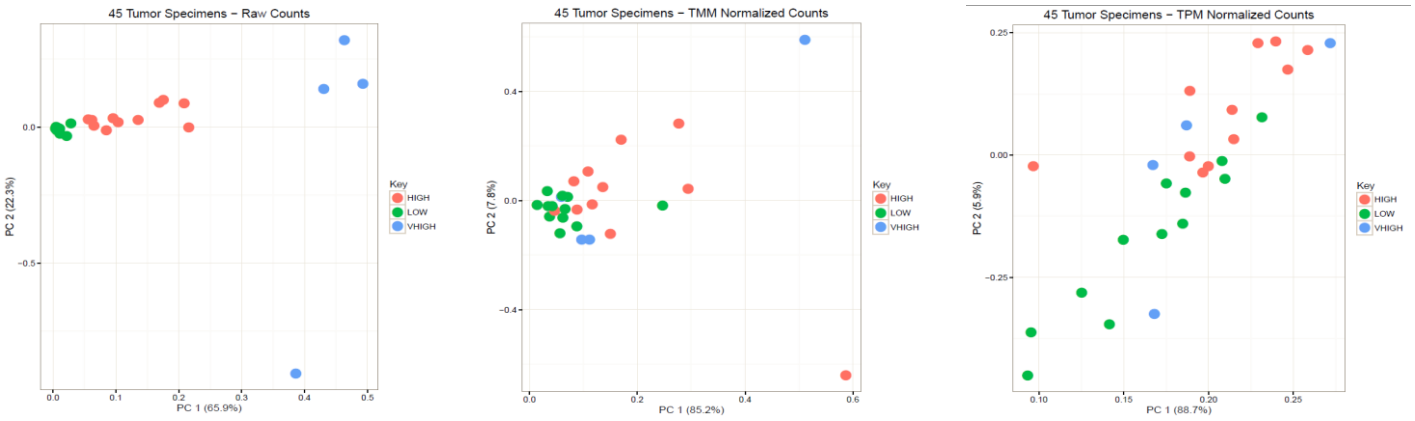


Figure 18: Comparison of no normalization, TMM normalization, and TPM normalization in PCA analysis of tumor specimens.

based on how many reads each had. This is a bias that should be eliminated from the analysis. TMM normalization reduces this effect somewhat, but specimens with low reads are still predominately clustering together. When TPM normalization is used, however, the artificial grouping of these specimens disappears and reads are more evenly spread out across the PC1 versus PC2 plot. In referring back to Figure 16, we see a similar spread in the data points using RPKM. Given that RPKM and TPM are similar normalization methods, this is not surprising, but it serves to emphasize the importance of normalizing for both gene length and sequencing depth.

Tools such as the Box Plot Application and the PCA Application can thus make exploring a data set more convenient and provide suggestions on areas worth investigating further. However, the investigator will still need to interpret the information these tools offer. As shown in this discussion, there remain questions about how RNA-sequencing data should be processed, and choices made can result in very different data analytic outcomes. From the analysis described, TPM normalization appears to provide advantages for analyzing KSHV viral read data that the other methods considered do not. RPKM offers some of the advantages obtained with TPM, but still leaves the investigator with the issue of having a non-standard baseline from which to compare gene count proportions. TPM offers all the advantages of RPKM, while eliminating this issue.

6. Future Directions

As data intensive research techniques such as RNA-sequencing become standard practice in biological research, the need for tools that assist researchers in analyzing or managing their data increases. R, while a powerful and versatile programming language, has a steep learning curve that limits its usefulness for many researchers. The two applications presented here remove this barrier and provide the additional benefit of interactive, dynamic visualizations. Given the extensive collection of R packages designed specifically for use in biological data analysis, it would appear that this approach could be used to increase their accessibility to a wider pool of users.

A natural progression would be to create applications that could use the results of the principal components analysis and query some of the gene ontology (GO) or pathway enrichment analysis functions in the edgeR package to determine if the genes that are being grouped together share complementary functions. Another potentially helpful application would be one that applied different normalization methods to a data file of raw counts, creating files that could then be used in applications such as the ones demonstrated here. A third possible future application would be one that performed hierarchical clustering analyses and created heat maps. As stated earlier, it is possible to do all of these tasks directly in R, but the process of going from a raw data set to the desired final result is not at all straightforward. By changing these tasks from coding problems to interactions with a graphical user interface, the Shiny package makes it possible for a research team with a small number of computationally-inclined members to generate tools in-house that can be used by everyone.

7. Appendices

7.1 Appendix A: Shiny Application User's Guides

7.1.1 Appendix A1: Box Plot Application User's Guide

Box Plot Application User's Guide

Emilia Gan

10 June 2016

1 Introduction

This application uses the R package Shiny (ref**) to create a graphical user interface with which a user can upload a data file and create box plots of selected genes to view and analyze variability in the expression of these genes across multiple samples. In addition to providing the capability of R to perform this task to users who are not familiar with the R programming language, the application has the further advantage of providing a level of flexibility beyond the typical R script. Specifically, the ease with which genes can be selected and deselected by simple mouse clicks and the ease with which new data files can be loaded, make analysis using the Shiny application more efficient even for expert R programmers.

2 Setting Up Your R Environment

2.1 Installation of R, RStudio, and Required R Packages

In order to run this application locally, you will need to have the free R software environment and the open-source RStudio integrated development environment (IDE) installed. Downloads for R are available for Unix platforms, Windows, and Mac OS. Installation instructions for each of these can be found using the following links:

- The R Project for Statistical Computing: https://www.r-project.org/
- RStudio IDE: https://www.rstudio.com/

Once you have this software installed, you need to install the following R packages:

- *shiny*: Web application framework for R
- *edgeR*; Empirical analysis of digital gene expression data in R
- *ggplot2*: An implementation of the grammar of graphics
- *limma*: Linear models (for all gene expression technologies)

Installing R packages from within the RStudio IDE is quite straightforward:

1) Open the RStudio IDE.

2) Click on the “Packages” tab to see a listing of currently installed packages.

3) All of the packages required to run the Box Plot Application are in the Comprehensive R Archive Network (CRAN) repository. Accordingly, they can be easily installed from within the RStudio IDE by following this sequence:

a) Click on the “Install” tab. A pop-up window will appear.

b) Type in the name of the package you wish to install.

c) Check the box labeled “install dependencies” to ensure that any packages required by the package you are requesting are automatically installed as well.

A screenshot illustrating the package installation process is provided below:

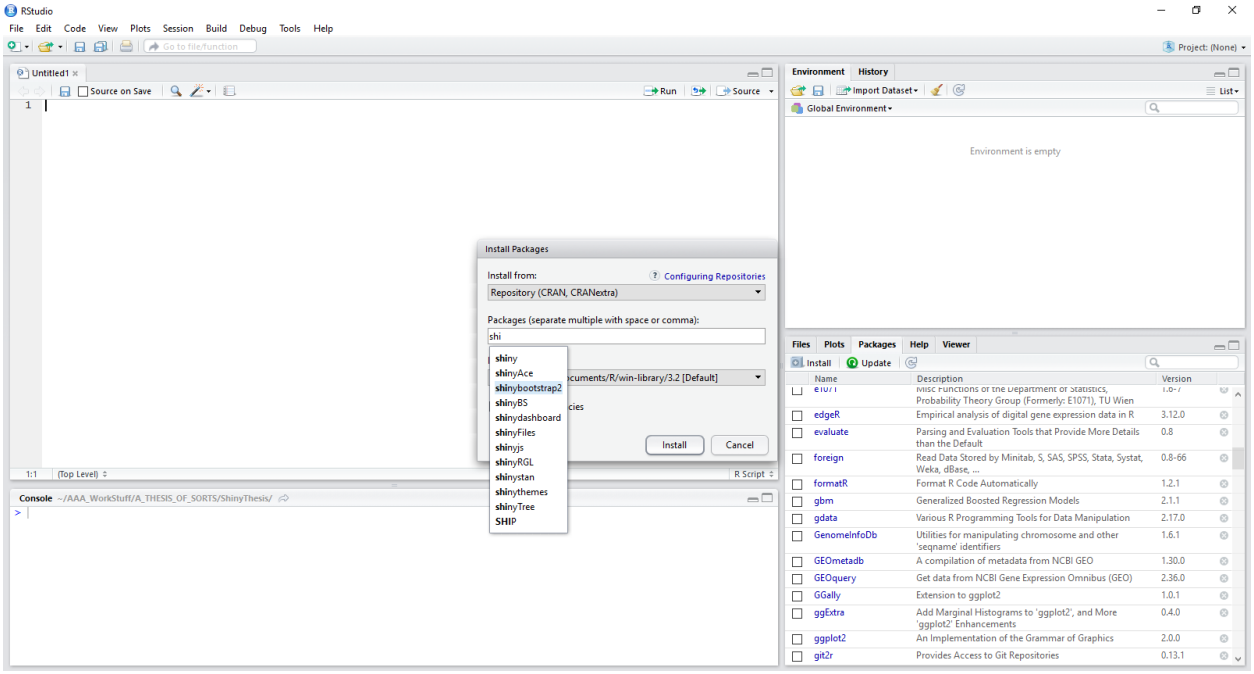


Figure 1: Installing R packages within the RStudio IDE.

2.2 Downloading the Box Plot Application Files

All R code files needed to run the Box Plot Application are publically available online on GitHub. The work is shared under an MIT License, meaning that it may be used without limitations,

provided attribution is given to the author and the author is released from any liability relating to use of the code.

Once downloaded, the application files may be stored anywhere in your local file system. For the applications to work most efficiently, the downloaded files for the Box Plot Application should be kept together in a single folder. R requires that each Shiny application be names “app.R” in order to be recognized and run as an application, without the code needing to be manually selected and highlighted. In addition, files that contain material the Shiny application needs to access, such as local image files, need to be stored in a folder named “www.” Keeping each app.R file and its associated www folder in its own, informatively named directory makes organizing this, and other, Shiny applications possible. The file hierarchical diagram shown below illustrates one option for organizing Shiny application files.

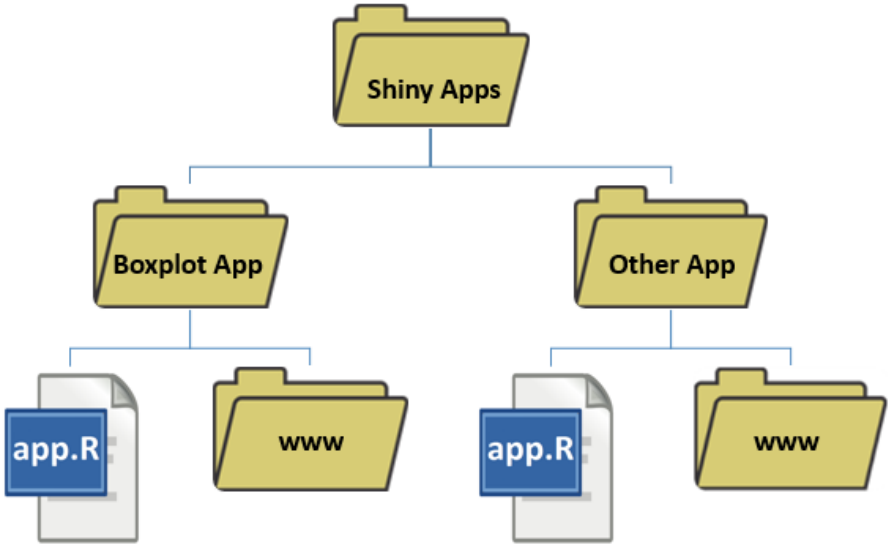


Figure 2: File organization for Shiny applications.

2.3 Running the Box Plot Application

Once the R environment has been set up, the Box Plot Application can be run from within R Studio. To do this, open RStudio and open the Box Plot Application’s “app.R” file. After the code

has opened in RStudio, the application can be deployed by clicking the “Run app” button as illustrated in the screen shot provided in Figure 3. Once the application has deployed, its functionality is accessed exclusively through the graphical user interface (GUI).

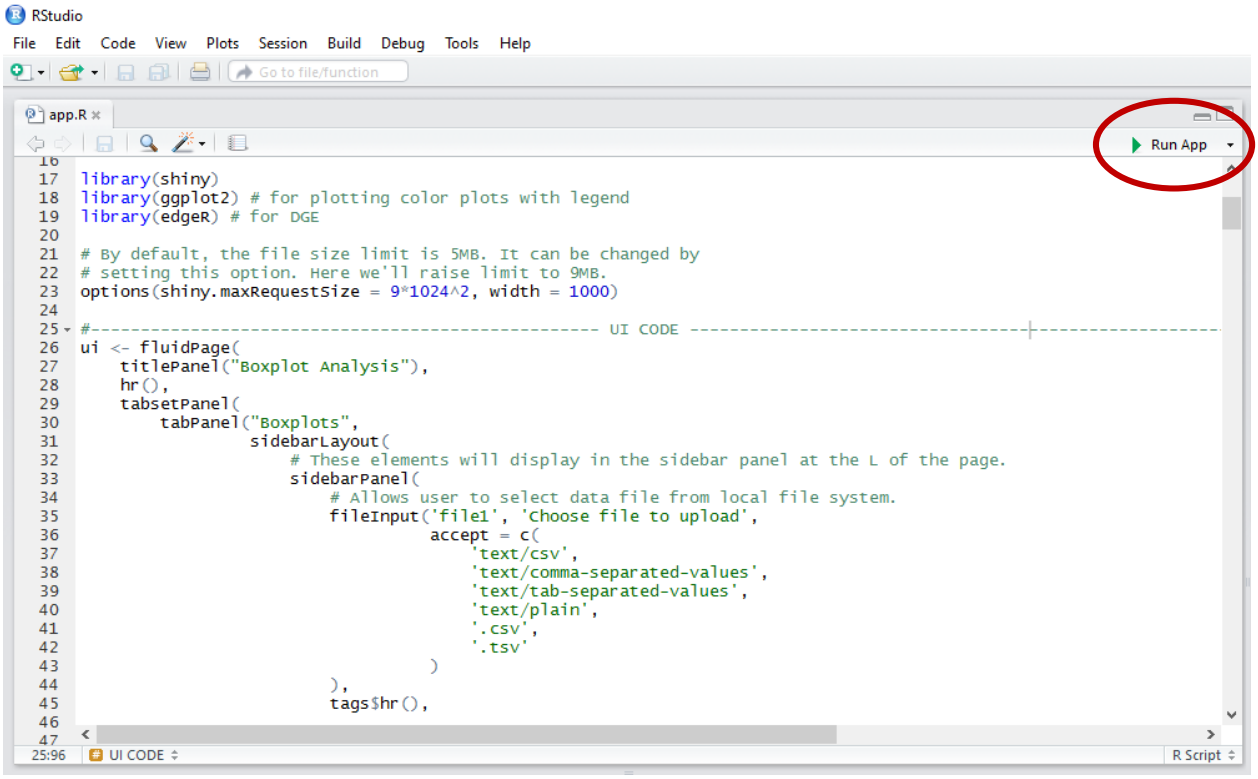


Figure 3: Deploying the Box Plot Application
Open the Box Plot Application’s app.R file in RStudio, then deploy the application from within RStudio by clicking the “Run App” arrow in the top right corner (outlined in red).

3 Overview of features

3.1 Box Plot Application Layout

Upon launching the Box Plot Application, the GUI opens. Figure 4 points out the various features of the Box Plot Application layout.

A. File upload box that allows the user to select an appropriately formatted file to upload from any location in the local file system.

B. Radio buttons to provide information on the file type to the application.

C. Radio buttons to provide information to the application on how the data are arranged in the data file.

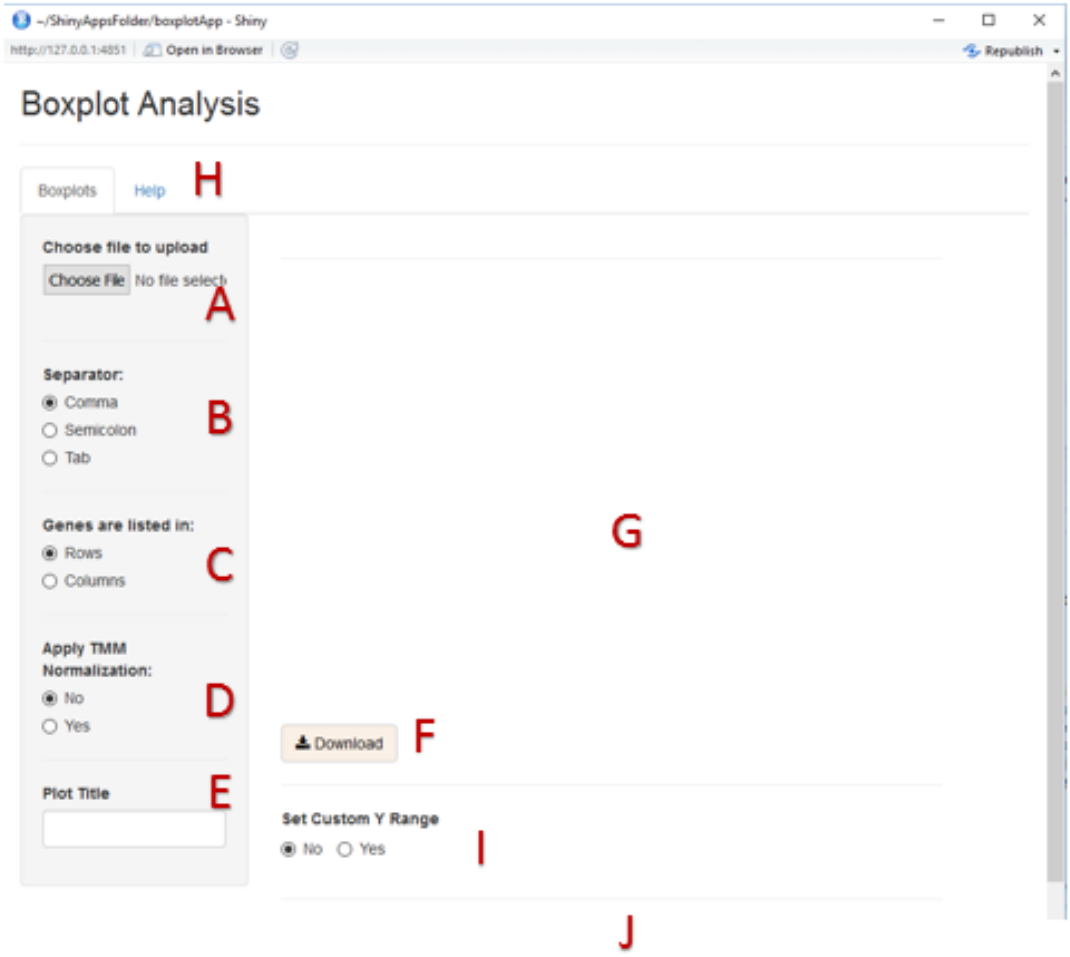


Figure 4: Box Plot Application Layout

D. Radio buttons to enable the user to indicate whether or not TMM normalization should be applied to the data. TMM normalization is discussed in Section 3.6 of this document.

E. Text entry box that allows the user to input a title for the plot(s) created.

F. Download button to save the plot created as a .pdf file.

G. Main plot display area.

H. Tab to a “Help” page that describes the format required of the data files uploaded to the application.

I. Radio buttons that allow the user to generate a second copy of the plot with a customized y-axis range. Selecting “Yes” in this set of radio buttons will cause a slider input to appear to the right of the radio buttons, along with a duplicate plot below (as displayed in Figures 5).

J. Area where data file itself will be displayed in tabular form.

3.2 Uploading Files and Selecting Features for Plotting

Upon uploading an appropriately formatted file, a data entry box appears. When this box is clicked, a drop down menu will appear with a list of the features (typically genes) to use in creating the plot. This is shown in Figure 5.

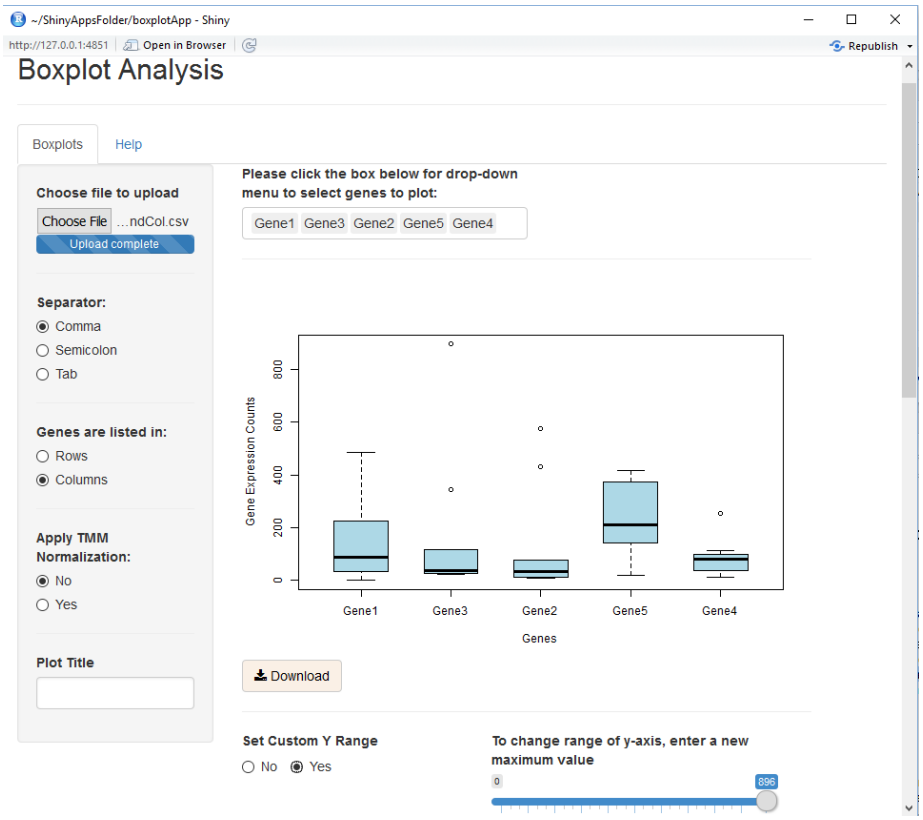


Figure 5: Box Plot Application with Uploaded Data File and Generated Plot.

The plot is generated once the first feature is selected. As additional features are selected for plotting, the plot dynamically updates itself. Features may be removed from the plot by selecting them in the data entry box and pushing the “Backspace” key or the “Delete” key on the keyboard. Additional features may be added at any time by selecting them from the drop down menu. A title may be entered via the text entry box labeled “Plot Title” in the sidebar panel. As mentioned, clicking inside the data entry box causes the drop down menu to appear. Clicking outside the box causes the drop down menu to close. At the same time that a data file is uploaded, the data in the data file are displayed at the bottom of the application in tabular format, as shown in Figure 8.

To select a new data file, all that is required is to return to the file upload box and select a new file from the local file system.

3.3 Permitted File Formats

The Box Plot Application can accept files in comma-separated value format (.csv or .txt), tab-separated value format (.tsv or .txt), or in semicolon-separated value format (.txt). The file should have both column names and row names for the data. In addition, the file may contain optional type information, if the elements (typically samples being sequences and genes) belong in different categories. While this data is not used by the Box Plot Application, it is used by its “partner” application, the Principal Component Analysis Application. The same data file can thus be used with both Shiny Applications.

The formatting of the data file is critical to the proper operation of the Box Plot Application. Figure 6 provides an example of the format that should be followed when preparing data for use with the application.

Sample	Gene1	Gene2	Gene3	Gene4	Gene5	Type
1_A	100	432	22	253	415	sampleTypeA
2_A	256	36	85	112	417	sampleTypeB
3_A	113	576	26	76	373	sampleTypeC
4_A	76	47	36	97	210	sampleTypeA
5_A	34	12	34	45	211	sampleTypeA
6_A	0	8	896	13	110	sampleTypeB
7_A	24	11	26	96	142	sampleTypeC
8_A	487	6	345	35	143	sampleTypeC
9_A	224	30	117	18	322	sampleTypeC
10_A	53	76	32	86	17	sampleTypeA
Type	geneType	geneTypeA	geneTypeB	geneTypeB	geneTypeA	filler

Figure 6: Proper Formatting of Data
There must be text in the top-left and bottom-right cells. However, the actual words “Sample” and “filler” are not critical and may be changed. Including type data is optional, but if it is included for either samples or genes, the word “Type” is critical, as the data will not be processed appropriately if this keyword is missing.

Figure 6 displays a data file as it might appear in a program such as Excel, before being converted to a comma-separated value file, or other permitted file format. All data files are required to include both column names and row names. For the use the application was designed for, these names will identify both the samples and genes included in the data set. The cells highlighted in red MUST contain text, as the data file is not permitted to contain any empty cells, but the actual word chosen is not critical. Type (or categorizing) information is optional. However, when such data are included, the word “Type” must be used in the cells highlighted in green in Figure 6. Keeping this format in mind, the red-highlighted cell in the lower-right corner is only an issue when BOTH row elements and column elements have associated type data. Figure 7 displays thumbnail versions of each of the permitted formats.

Sample	Gene1	Gene2	Gene3	Gene4	Gene5	Type
1_A	100	432	22	253	415	sampleTypeA
2_A	256	36	85	112	417	sampleTypeB
3_A	113	576	26	76	373	sampleTypeC
4_A	76	47	36	97	210	sampleTypeA
5_A	34	12	34	45	211	sampleTypeA
6_A	0	8	896	13	110	sampleTypeB
7_A	24	11	26	96	142	sampleTypeC
8_A	487	6	345	35	143	sampleTypeC
9_A	224	30	117	18	322	sampleTypeC
10_A	53	76	32	86	17	sampleTypeA
Type	geneTypeA	geneTypeA	geneTypeB	geneTypeB	geneTypeA	filler

Figure 7A: Sample and gene Type data included (data in both last row and last column).

Sample	Gene1	Gene2	Gene3	Gene4	Gene5
1_A	100	432	22	253	415
2_A	256	36	85	112	417
3_A	113	576	26	76	373
4_A	76	47	36	97	210
5_A	34	12	34	45	211
6_A	0	8	896	13	110
7_A	24	11	26	96	142
8_A	487	6	345	35	143
9_A	224	30	117	18	322
10_A	53	76	32	86	17
Type	geneTypeA	geneTypeA	geneTypeB	geneTypeB	geneTypeA

Figure 7B: Only gene Type data included (data in last row only).

Sample	Gene1	Gene2	Gene3	Gene4	Gene5	Type
1_A	100	432	22	253	415	sampleTypeA
2_A	256	36	85	112	417	sampleTypeB
3_A	113	576	26	76	373	sampleTypeC
4_A	76	47	36	97	210	sampleTypeA
5_A	34	12	34	45	211	sampleTypeA
6_A	0	8	896	13	110	sampleTypeB
7_A	24	11	26	96	142	sampleTypeC
8_A	487	6	345	35	143	sampleTypeC
9_A	224	30	117	18	322	sampleTypeC
10_A	53	76	32	86	17	sampleTypeA

Figure 7C: Only sample Type data included (data in lasr column only).

Sample	Gene1	Gene2	Gene3	Gene4	Gene5
1_A	100	432	22	253	415
2_A	256	36	85	112	417
3_A	113	576	26	76	373
4_A	76	47	36	97	210
5_A	34	12	34	45	211
6_A	0	8	896	13	110
7_A	24	11	26	96	142
8_A	487	6	345	35	143
9_A	224	30	117	18	322
10_A	53	76	32	86	17

Figure 7D: No Type data included.

Figure 7: Permitted file formats
Examples of permitted file formats illustrating, once again, that column names and row names are required (first column and first row and that type (categorical) information is optional, but, if present, must be identified using the keyword “Type.”

3.4 Rescaling the y-axis

Due to potentially wide variation in the expression levels of genes, certain box plots may appear excessively “compressed.” Accordingly, the user is given the ability to change the range of the y-axis at will by clicking on the “Yes” option of the “Set Custom Y Range” radio buttons. Changing this setting to “Yes” causes a slider input widget to be displayed. In addition, once the slider has been clicked, its position may also be controlled via the right and left arrow keys on the keyboard. The slider input and its effect on the y-axis range are illustrated in Figure 8.

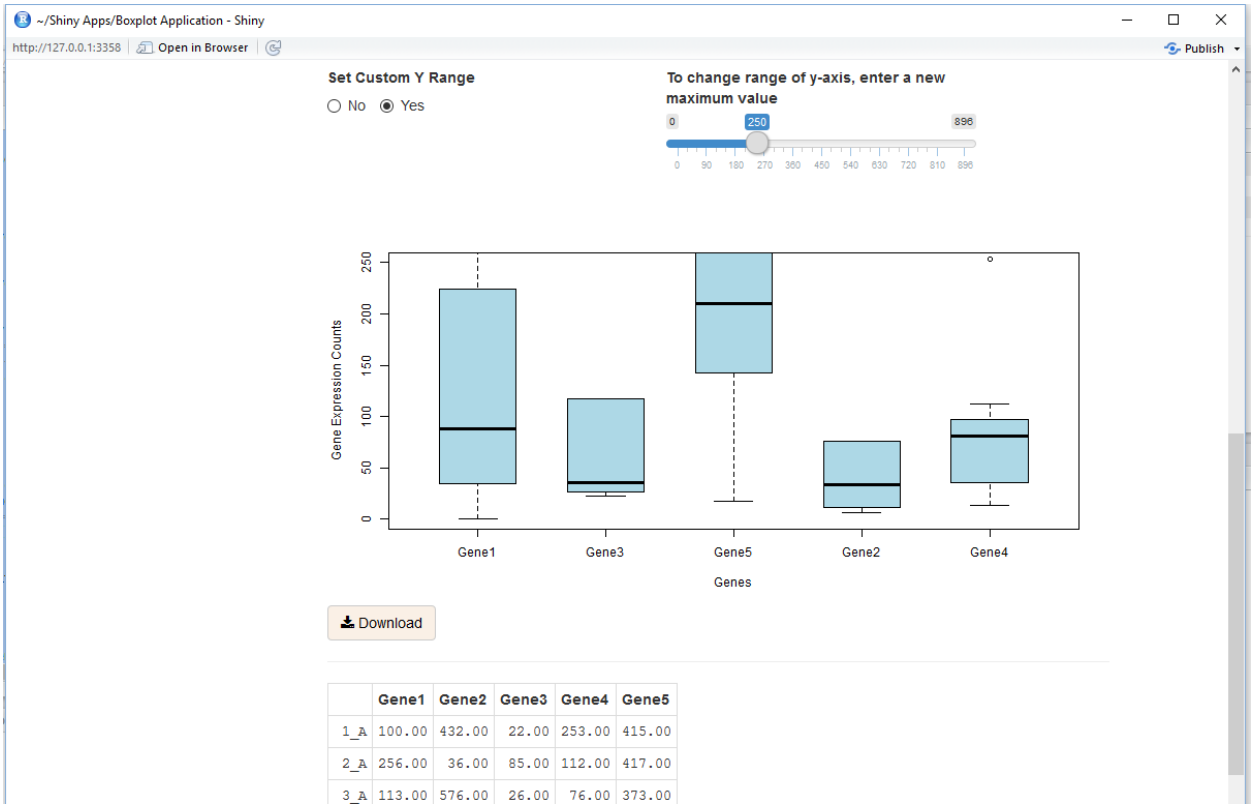


Figure 8: Rescaling the y-axis
The y-axis range is controlled by the slider input widget that appears on the screen once the "Set Custom Y Range" radio buttons are set to "Yes." Note the presence of the top portion of the data file in tabular form under the scaled plot.

The rescaled (lower) plot will have the same plot title as the initial (upper) plot, or no title if no text was entered in the text entry box labeled "Plot Title" in the sidebar panel.

3.5 Interpreting the Box Plots

The box is a graphical representation of a data set. The various sections of the box plot represent different values. The top, middle, and bottom lines of the plot represent the 3rd, 2nd, and 1st quartiles, respectively. The "whiskers" represent the highest and lowest data values that fall within 1.5 IQR of the 3rd quartile and 1st quartile, respectively. Outlier are represented by points in line with, but falling above or below the ends of, the whiskers.

This application was designed to create box plots of gene expression counts data across multiple samples. Selecting more genes from the drop down menu will result in a plot with correspondingly more aligned box plots. The plots created by the Box Plot Application can

contain between 12 and 18 box plots before the box plots become so cramped as to be difficult to interpret.

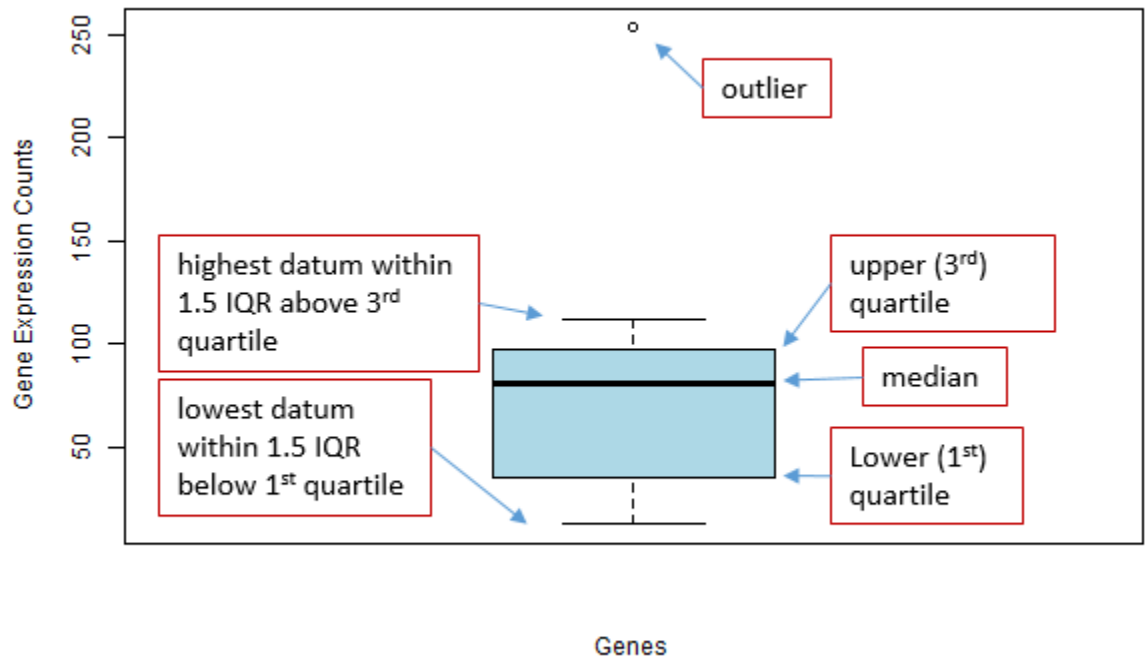


Figure 9: The Box Plot
The box plot, also known as a box-and-whisker plot, graphically depicts sets of numerical data.

3.6 Trimmed Mean of M-values Normalization

Normalizing the RNA-Sequencing counts data is a standard step in the data analysis pipeline. However, there is no consensus on the “best” normalization method to use, and a number of different methods are commonly used. The PCA Application offers the option of normalizing the uploaded data using the Trimmed Mean of M-values (TMM) method described in the article by Robinson and Oshlack (6).

The TMM normalization method uses raw counts data to estimate scaling factors to ensure that genes being expressed at the same level in different samples are not erroneously flagged as being differentially expressed. The base assumption of the method is that the majority of genes are not differentially expressed. The method estimates the ratio of RNA production between samples by calculating a weighted trimmed mean of log expression ratios. The PCA Application

uses the default setting in the edgeR package for the percentage of data trimmed before the mean calculation.

In acknowledgement of the existence of other normalization techniques and the possibility that the user of the application may wish to use some method other than TMM, whether or not the data being plotted undergo the normalization or not is user-controlled via the “Apply TMM Normalization” set of radio buttons. The user thus has the option of conducting the normalization step prior to uploading the data file for plotting, or bypassing this step altogether to look at raw counts data.

3.7 Downloading Plots

The plots generated by the Box Plot Application may be saved to the local file system as .pdf files by clicking on the “Download” button located under each plot.

4 Troubleshooting and How to Get Help

All attempts have been made to thoroughly test this application. Should features fail to work as described, the most common source of the problem is likely to be an improperly formatted data file. Please refer to the diagrams showing accepted formats.

It is also important to click the appropriate radio button for the file format being uploaded. The app will not process a comma-separated value file, for example, despite its being a permitted format, if the radio button clicked indicates that it should be uploaded a tab-separated values file.

This software is being made publicly available for all uses, including modification. It is thus offered with no guarantees or support provided. However, the software author may be contacted at efgan@uw.edu, and will provide assistance when possible.

5 References

1. R Core Team (2015). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

2. RStudio Team (2015). RStudio: Integrated Development for R. RStudio, Inc., Boston, MA URL http://www.rstudio.com/.

3. Institute for Statistics and Math (Wirtschaftsuniversitat Wien). The Comprehensive R Archive Network, https://cran.r-project.org/ , accessed 8 May 2016.

4. The Open Source Initiative. The MIT License (MIT), https://opensource.org/licenses/MIT , accessed 8 May 2016.

5. Robinson, Mark D and Alicia Oshlack. “A scaling normalization method for differential expression analysis of RNA-seq data.” *Genome Biology* (2010), 11:R225.

6. Robinson MD, McCarthy DJ and Smyth GK (2010). edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26, 139-140

Principal Component Analysis (PCA) Application User's Guide

Emilia Gan

10 June 2016

1 Introduction

This application uses the R package Shiny (ref**) to create a graphical user interface with which a user can upload a data file and perform principal component analysis on the uploaded data. The application produces a plot of the first two principal components, which can be downloaded. In addition to providing the capability of R to perform this task to users who are not familiar with the R programming language, the application has the further advantage of providing a level of flexibility beyond the typical R script. Specifically, the ease with which the user can toggle between having the row elements or the column elements displayed in the plot and the ease with which new data files can be loaded, make analysis using the Shiny application more efficient even for expert R programmers.

2 Setting Up Your R Environment

2.1 Installation of R, RStudio, and Required R Packages

In order to run this application locally, you will need to have the free R software environment and the open-source RStudio integrated development environment (IDE) installed. Downloads for R are available for Unix platforms, Windows, and Mac OS. Installation instructions for each of these can be found using the following links:

- The R Project for Statistical Computing: https://www.r-project.org/
- RStudio IDE: https://www.rstudio.com/

Once you have this software installed, you need to install the following R packages:

- *shiny*: Web application framework for R
- *edgeR*; Empirical analysis of digital gene expression data in R
- *ggplot2*: An implementation of the grammar of graphics
- *limma*: Linear models (for all gene expression technologies)

Installing R packages from within the RStudio IDE is quite straightforward:

1) Open the RStudio IDE.

2) Click on the “Packages” tab to see a listing of currently installed packages.

3) All of the packages required to run the PCA Application are in the Comprehensive R Archive Network (CRAN) repository. Accordingly, they can be easily installed from within the RStudio IDE by following this sequence:

a) Click on the “Install” tab. A pop-up window will appear.

b) Type in the name of the package you wish to install.

c) Check the box labeled “install dependencies” to ensure that any packages required by the package you are requesting are automatically installed as well.

A screenshot illustrating the package installation process is provided below:

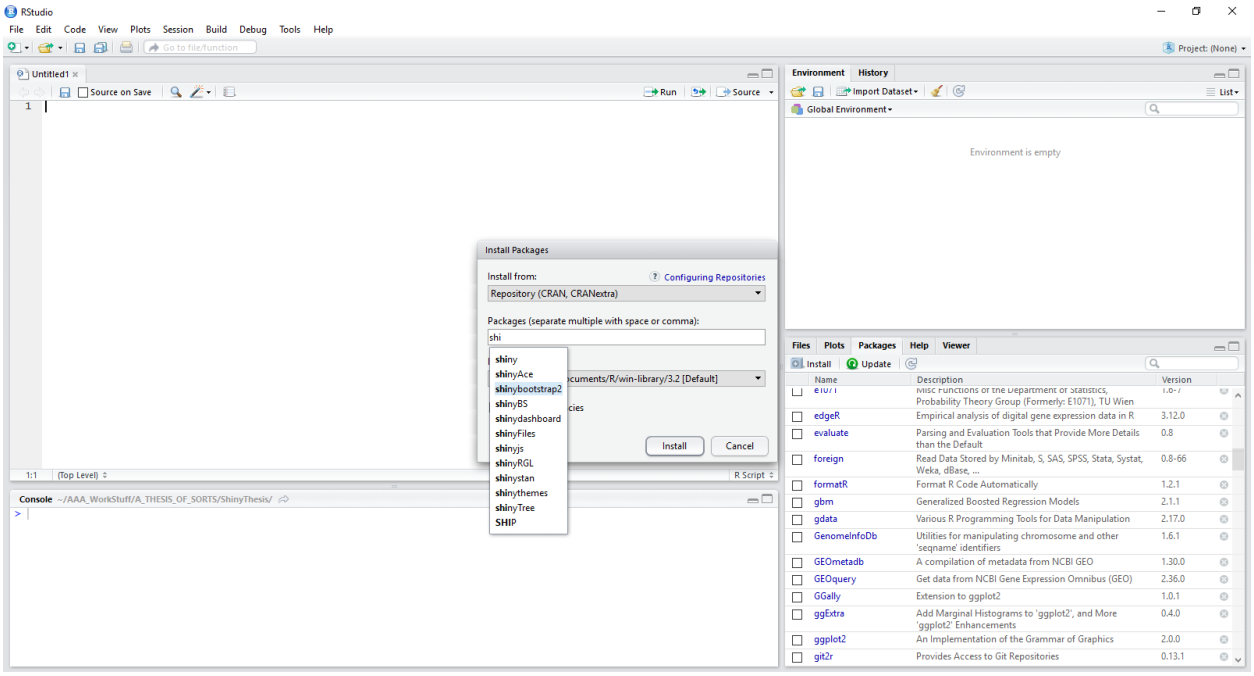


Figure 1: Installing R packages within the RStudio IDE.

2.2 Downloading the PCA Application Files

All R code files needed to run the PCA Application are publically available online on GitHub. The work is shared under an MIT License, meaning that it may be used without limitations, provided attribution is given to the author and the author is released from any liability relating to use of the code.

Once downloaded, the application files may be stored anywhere in your local file system. For the applications to work most efficiently, the downloaded files for the PCA Application should be kept together in a single folder. R requires that each Shiny application be names “app.R” in order to be recognized and run as an application, without the code needing to be manually selected and highlighted. In addition, files that contain material the Shiny application needs to access, such as local image files, need to be stored in a folder named “www.” Keeping each app.R file and its associated www folder in its own, informatively named directory makes organizing this, and other, Shiny applications possible. The file hierarchical diagram shown below illustrates one option for organizing Shiny application files.

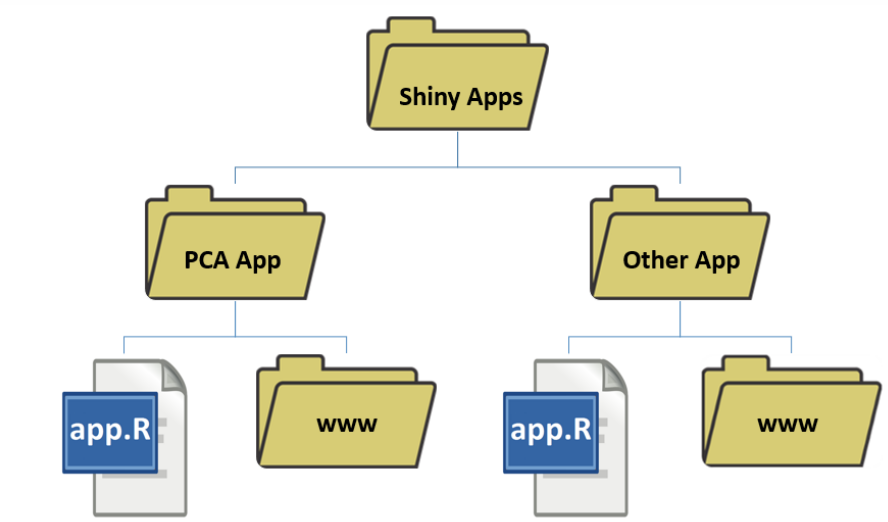


Figure 2: File organization for Shiny applications.

2.3 Running the PCA Application

Once the R environment has been set up, the PCA Application can be run from within R Studio. To do this, open RStudio and open the PCA Application’s “app.R” file. After the code has

opened in RStudio, the application can be deployed by clicking the “Run app” button as illustrated in the screen shot provided in Figure 3. Once the application has deployed, its functionality is accessed exclusively through the graphical user interface (GUI).

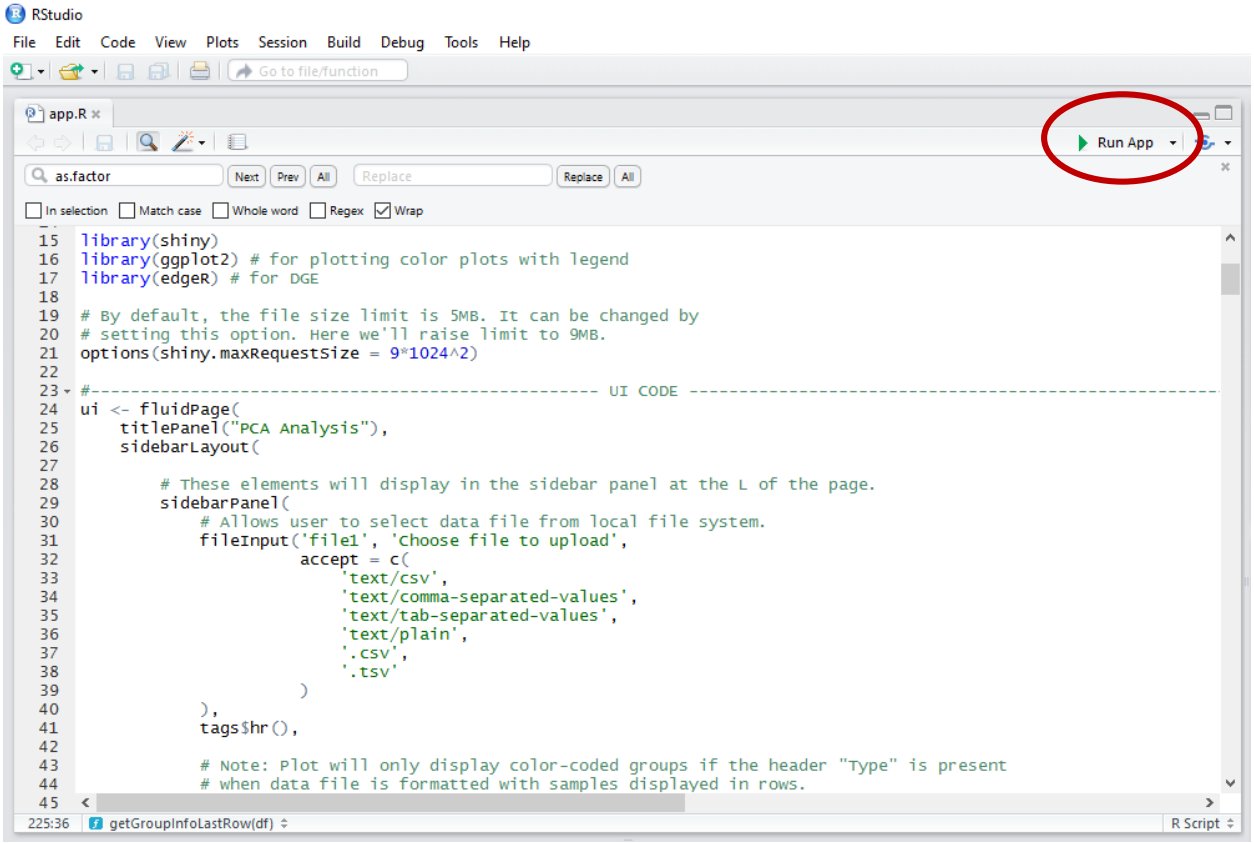


Figure 3: Deploying the PCA Application
Open the PCA Application’s app.R file in RStudio, then deploy the application from within RStudio by clicking the “Run App” arrow in the top right corner (outlined in red).

3 Overview of features

3.1 PCA Application Layout

Upon launching the PCA Application, the GUI opens. Figure 4 points out the various features of the PCA Application layout.

A. File upload box that allows the user to select an appropriately formatted file to upload from any location in the local file system.

B. Radio buttons to provide information on the file type to the application.

C. Radio buttons to provide information to the application on how the data are arranged in the data file. It is CRITICAL that the program know whether genes are arranged listed by row or by column. Inaccurate information will cause invalid results and, potentially, abrupt program termination.

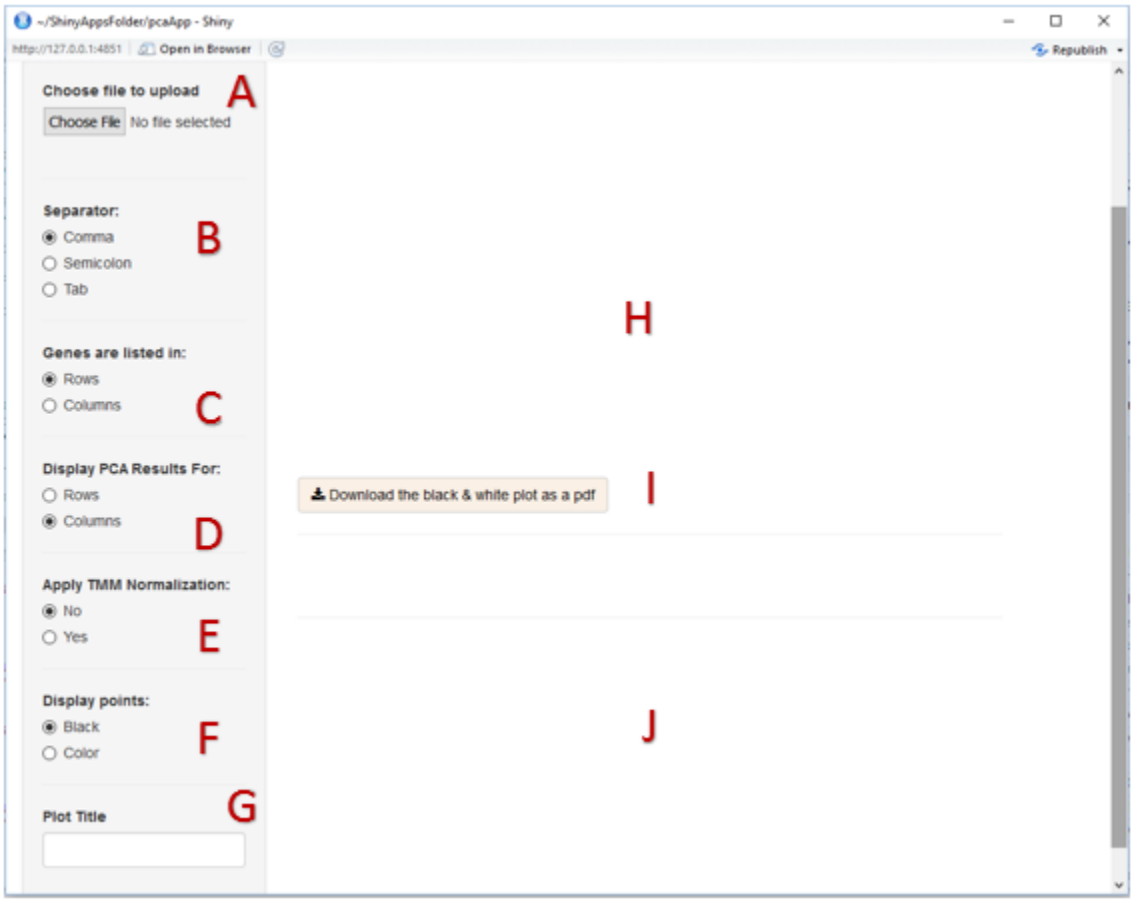


Figure 4: PCA Application Layout

D. Radio buttons to indicate whether the PC 1 versus PC 2 plot should plot the elements that arranged in rows or in columns in the uploaded data file.

E. Radio buttons to enable the user to indicate whether or not TMM normalization should be applied to the data. TMM normalization is discussed in Section 3.6 of this document.

F. Radio buttons to indicate if the plot's points should be displayed black or in color. Color is only an option when group (categorical) information is provided.

G. Text entry box that allows the user to input a title for the plot(s) created.

H. Main plot display area.

I. Download button to save the plot created as a .pdf file.

J. Area for displaying data file contents in tabular form.

(Not shown in the figure: Tab to a "Help" page that describes the format required of the data files uploaded to the application and provides usage instructions. Located offscreen in this screenshot, at the top of the application.)

3.2 Uploading Files and Selecting Features for Plotting

Upon uploading an appropriately formatted file, a scatter plot of the first two principal components is displayed. The plot axes include information about the variance accounted for by each principal component.

Figure 5: PCA Application with Uploaded Data File and Generated Plot.

Also visible is the tabular representation of the data file at the bottom of the display. The plot, itself, is interactive. Mousing over the plot causes the cursor to change to a “+” symbol, which can be moved over any point in the plot. Upon clicking, that point’s name and plot coordinates are displayed in the grey box below the plot. In Figure 5, the point selected was the point representing sample 6_A, which is located in the lower left corner of the plot. A title may be entered via the text entry box labeled “Plot Title” in the sidebar panel (off screen). The data being examined can be easily changed.To select a new data file, all that is required is to return to the file upload box and select a new file from the local file system.

3.3 Permitted File Formats

The PCA Application can accept files in comma-separated value format (.csv or .txt), tab-separated value format (.tsv or .txt), or in semicolon-separated value format (.txt). The file should have both column names and row names for the data. In addition, the file may contain optional type information, if the elements (typically samples being sequences and genes) belong in different categories.

The formatting of the data file is critical to the proper operation of the PCA Application. Figure 6 provides an example of the format that should be followed when preparing data for use with the application.

Sample	Gene1	Gene2	Gene3	Gene4	Gene5	Type
1_A	100	432	22	253	415	sampleTypeA
2_A	256	36	85	112	417	sampleTypeB
3_A	113	576	26	76	373	sampleTypeC
4_A	76	47	36	97	210	sampleTypeA
5_A	34	12	34	45	211	sampleTypeA
6_A	0	8	896	13	110	sampleTypeB
7_A	24	11	26	96	142	sampleTypeC
8_A	487	6	345	35	143	sampleTypeC
9_A	224	30	117	18	322	sampleTypeC
10_A	53	76	32	86	17	sampleTypeA
Type	geneTypeA	geneTypeA	geneTypeB	geneTypeB	geneTypeA	filler

Figure 6: Proper Formatting of Data
There must be text in the top-left and bottom-right cells. However, the actual words “Sample” and “filler” are not critical and may be changed. Including type data is optional, but if it is included for either samples or genes, the word “Type” is critical, as the data will not be processed appropriately if this keyword is missing.

Figure 6 displays a data file as it might appear in a program such as Excel, before being converted to a comma-separated value file, or other permitted file format. All data files are required to include both column names and row names. For the use the application was designed for, these names will identify both the samples and genes included in the data set. The cells highlighted in red MUST contain text, as the data file is not permitted to contain any empty cells, but the actual word chosen is not critical. Type (or categorizing) information is optional. However, when such data are included, the word “Type” must be used in the cells highlighted in green in Figure 6. Keeping this format in mind, the red-highlighted cell in the lower-right corner is only an issue when BOTH row elements and column elements have associated type data. Figure 7 displays thumbnail versions of each of the permitted formats.

Sample	Gene1	Gene2	Gene3	Gene4	Gene5	Type
1_A	100	432	22	253	415	sampleTypeA
2_A	256	36	85	112	417	sampleTypeB
3_A	113	576	26	76	373	sampleTypeC
4_A	76	47	36	97	210	sampleTypeA
5_A	34	12	34	45	211	sampleTypeA
6_A	0	8	896	13	110	sampleTypeB
7_A	24	11	26	96	142	sampleTypeC
8_A	487	6	345	35	143	sampleTypeC
9_A	224	30	117	18	322	sampleTypeC
10_A	53	76	32	86	17	sampleTypeA
Type	geneTypeA	geneTypeA	geneTypeB	geneTypeB	geneTypeA	filler

Figure 7A: Sample and gene Type data included (data in both last row and last column).

Sample	Gene1	Gene2	Gene3	Gene4	Gene5
1_A	100	432	22	253	415
2_A	256	36	85	112	417
3_A	113	576	26	76	373
4_A	76	47	36	97	210
5_A	34	12	34	45	211
6_A	0	8	896	13	110
7_A	24	11	26	96	142
8_A	487	6	345	35	143
9_A	224	30	117	18	322
10_A	53	76	32	86	17
Type	geneTypeA	geneTypeA	geneTypeB	geneTypeB	geneTypeA

Figure 7B: Only gene Type data included (data in last row only).

Sample	Gene1	Gene2	Gene3	Gene4	Gene5	Type
1_A	100	432	22	253	415	sampleTypeA
2_A	256	36	85	112	417	sampleTypeB
3_A	113	576	26	76	373	sampleTypeC
4_A	76	47	36	97	210	sampleTypeA
5_A	34	12	34	45	211	sampleTypeA
6_A	0	8	896	13	110	sampleTypeB
7_A	24	11	26	96	142	sampleTypeC
8_A	487	6	345	35	143	sampleTypeC
9_A	224	30	117	18	322	sampleTypeC
10_A	53	76	32	86	17	sampleTypeA

Figure 7C: Only sample Type data included (data in lasr column only).

Sample	Gene1	Gene2	Gene3	Gene4	Gene5
1_A	100	432	22	253	415
2_A	256	36	85	112	417
3_A	113	576	26	76	373
4_A	76	47	36	97	210
5_A	34	12	34	45	211
6_A	0	8	896	13	110
7_A	24	11	26	96	142
8_A	487	6	345	35	143
9_A	224	30	117	18	322
10_A	53	76	32	86	17

Figure 7D: No Type data included.

Figure 7: Permitted file formats
Examples of permitted file formats illustrating, once again, that column names and row names are required (first column and first row and that type (categorical) information is optional, but, if present, must be identified using the keyword “Type.” The decision to have genes in columns and samples in rows or to reverse them is up to the user. The application can process files in either orientation.

3.4 Displaying Categorical (Type) Information

If type/categorical information is included for the elements being plotted, selecting the “Color” radio button under “Display points” will color code points according to their category. A legend listing the categories appears to the right of the plot. In addition, upon selection the “Color” option a “Shapes” checkbox also appears, further emphasizing the different categories by assigning each category a unique shape. The “Shapes” option is offered to facilitate the creation of plots in which categories remain distinguishable even in black and white reproductions of the plot. Figure 8 illustrates each of these features.

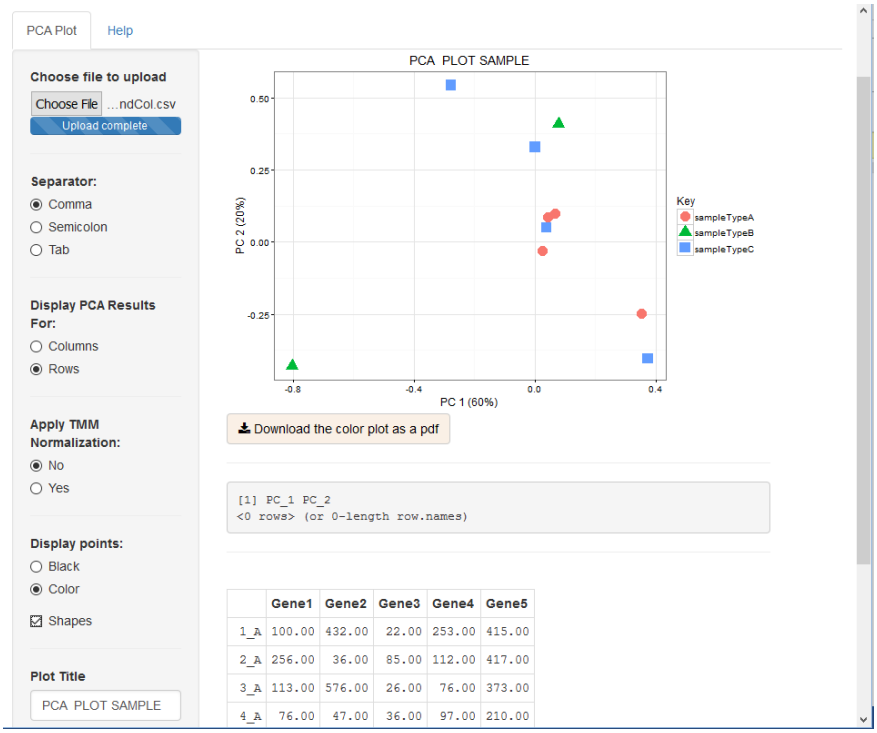


Figure 8: Displaying Type (categorical) Information
If type/categorical information is included for the elements being plotted, these categories may be indicated using different colors and shapes.

3.5 Interpreting the PC1 versus PC2 Plot

Principal component analysis is a method that provides investigators with a low-dimensional representation of the data set that still captures much of the variation in the data set. The first principal component is the normalized linear combination of the set of features on which data were collected (for this application, generally the gene expression counts) which has the greatest variance. The second principal component is the linear combination with the highest variance that is also uncorrelated with the first principal component. In effect, the original data are being projected into a lower dimensional subspace -- a plane created by the two principal components. These projected points are used to create the plot.

The proportion of the data set's variation that is being captured by each principal component is stated in the axes labels. In Figure 8, PC 1 accounts for 60% of the variation, with PC 2 explaining 20% more, bringing the cumulative variance captured by the plot to 80%. The

visualization created using these two principal components has the potential to reveal interesting patterns in the data, and is a valuable tool for exploratory data analysis.

3.6 Trimmed Mean of M-values Normalization

Normalizing the RNA-Sequencing counts data is a standard step in the data analysis pipeline. However, there is no consensus on the “best” normalization method to use, and a number of different methods are commonly used. The PCA Application offers the option of normalizing the uploaded data using the Trimmed Mean of M-values (TMM) method described in the article by Robinson and Oshlack (6).

The TMM normalization method uses raw counts data to estimate scaling factors to ensure that genes being expressed at the same level in different samples are not erroneously flagged as being differentially expressed. The base assumption of the method is that the majority of genes are not differentially expressed. The method estimates the ratio of RNA production between samples by calculating a weighted trimmed mean of log expression ratios. The PCA Application uses the default setting in the edgeR package for the percentage of data trimmed before the mean calculation.

In acknowledgement of the existence of other normalization techniques and the possibility that the user of the application may wish to use some method other than TMM, whether or not the data being plotted undergo the normalization or not is user-controlled via the “Apply TMM Normalization” set of radio buttons. The user thus has the option of conducting the normalization step prior to uploading the data file for plotting, or bypassing this step altogether to look at raw counts data.

3.7 Downloading Plots

The plots generated by the PCA Application may be saved to the local file system as .pdf files by clicking on the “Download” button located under each plot.

4 Troubleshooting and How to Get Help

All attempts have been made to thoroughly test this application. Should features fail to work as described, the most common source of the problem is likely to be an improperly formatted data file. Please refer to the diagrams showing accepted formats.

It is also important to click the appropriate radio button for the file format being uploaded. The app will not process a comma-separated value file, for example, despite its being a permitted format, if the radio button clicked indicates that it should be uploaded a tab-separated values file.

This software is being made publicly available for all uses, including modification. It is thus offered with no guarantees or support provided. However, the software author may be contacted at efgan@uw.edu, and will provide assistance when possible.

5 References

1. R Core Team (2015). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

2. RStudio Team (2015). RStudio: Integrated Development for R. RStudio, Inc., Boston, MA URL http://www.rstudio.com/.

3. Institute for Statistics and Math (Wirtschaftsuniversitat Wien). The Comprehensive R Archive Network, https://cran.r-project.org/ , accessed 8 May 2016.

4. The Open Source Initiative. The MIT License (MIT), https://opensource.org/licenses/MIT , accessed 8 May 2016.

5. James, Gareth, et al. *An Introduction to Statistical Learning*. New York: Springer, 2014.

6. Robinson, Mark D and Alicia Oshlack. “A scaling normalization method for differential expression analysis of RNA-seq data.” *Genome Biology* (2010), 11:R225.

7. Robinson MD, McCarthy DJ and Smyth GK (2010). edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26, 139-140

7.2 Appendix B: Testing Process

The following “toy data sets” are representative of those used in the first stage of testing the applications. All data sets are presented in comma-separated values format, only, to cut down on redundancy. If desired, similar files using alternate separators may be generated by copying these files and replacing the commas with tabs or semi-colons (the other two permitted separators). The second stage of testing involved the use of actual lab data, which will not be included here in its raw form. Results of analyses performed using the applications on this data are presented in the Results section.

1. No type data included - .csv format

```
Sample,Gene1,Gene2,Gene3,Gene4,Gene5
1_A,100,432,22,253,415
2_A,256,36,85,112,417
3_A,113,576,26,76,373
4_A,76,47,36,97,210
5_A,34,12,34,45,211
6_A,0,8,896,13,110
7_A,24,11,26,96,142
8_A,487,6,345,35,143
9_A,224,30,117,18,32200
10_A,53,76,32,86,17
```

2. Type data in the last column, but not in the last row - .csv format

```
Sample,Gene1,Gene2,Gene3,Gene4,Gene5,Type
1_A,100,432,22,253,415,sampleTypeA
2_A,256,36,85,112,417,sampleTypeB
3_A,113,576,26,76,373,sampleTypeC
4_A,76,47,36,97,210,sampleTypeA
5_A,34,12,34,45,211,sampleTypeA
6_A,0,8,896,13,110,sampleTypeB
7_A,24,11,26,96,142,sampleTypeC
8_A,487,6,345,35,143,sampleTypeC
9_A,224,30,117,18,322,sampleTypeC
10_A,53,76,32,86,17,sampleTypeA
```

3. Type data in the last row, but not in the last column - .csv format

```
Gene,1_A,2_A,3_A,4_A,5_A
Gene1,100,432,22,253,415
Gene2,256,36,85,112,417
Gene3,113,576,26,76,373
Gene4,76,47,36,97,210
Gene5,34,12,34,45,211
Gene6,0,8,896,13,110
Gene7,24,11,26,96,142
Gene8,487,6,345,35,143
Gene9,224,30,117,18,322
Gene10,53,76,32,86,17
Type,geneTypeA,geneTypeA,geneTypeB,geneTypeB,geneTypeA
```

4. Type data in both the last row and the last column - .csv format

```
Gene,Sample1,Sample2,Sample3,Sample4,Sample5,Type
Gene1,100,432,22,253,415,sampleTypeA
Gene2,256,36,85,112,417,sampleTypeB
Gene3,113,576,26,76,373,sampleTypeC
Gene4,76,47,36,97,210,sampleTypeA
Gene5,34,12,34,45,211,sampleTypeA
Gene6,0,8,896,13,110,sampleTypeB
Gene7,24,11,26,96,142,sampleTypeC
Gene8,487,6,345,35,143,sampleTypeC
Gene9,224,30,117,18,322,sampleTypeC
Gene10,53,76,32,86,17,sampleTypeA
Type,geneTypeA,geneTypeA,geneTypeB,geneTypeB,geneTypeA,filler
```

As can be seen, the data sets used for testing are much smaller than actual data sets would be and the numerical values contained in each toy data set cover a very narrow range. Real data sets are likely to have much more variation and when testing actual experimental data, certain parts of the program had to be recoded to account for the possibility of a given gene having count values of zero across all samples sequenced. In addition, the assumption was made that the applications would be used only with valid data sets, so no effort was made to challenge the applications with “bad” data, such as counts files that included negative gene counts or non-numerical entries.

7.3 Appendix C: R Code

R code is being provided in this appendix for completeness, but is better viewed after being downloaded from GitHub (See Appendix D: Supplemental Materials).

7.3.1 Box Plot Application Code

```
# Emilia F Gan
# Student Number: 1138607
# Email: efgan@uw.edu
# Copyright E Gan 2016
# Submitted in partial fulfillment of the requirements
# of the degree Master of Science

# Box Plot Application

# Opening a CSV File and reading into R
# Modeled after: http://shiny.rstudio.com/gallery/upload-file.html
# Downloading plots learned from: https://www.youtube.com/watch?v=LSnWGmVkB6A

library(shiny)
library(ggplot2) # for plotting color plots with legend
library(edgeR) # for DGE
library(limma)

# By default, the file size limit is 5MB. It can be changed by
# setting this option. Here we'll raise limit to 9MB.
options(shiny.maxRequestSize = 9*1024^2)

#---------------------------------------------------- UI CODE ----------------------------------------------------------------
ui <- fluidPage(
   titlePanel("Boxplot Analysis"),
   hr(),
   tabsetPanel(
      tabPanel("Boxplots",
               sidebarLayout(
                  # These elements will display in the sidebar panel at the L of the page.
                  sidebarPanel(
                     # Allows user to select data file from local file system.
                     fileInput('file1', 'Choose file to upload',
                               accept = c(
                                  'text/csv',
                                  'text/comma-separated-values',
                                  'text/tab-separated-values',
                                  'text/plain',
                                  '.csv',
                                  '.tsv'
                               )
                     ),
                     tags$hr(),
```

```
    # Indicates whether or not the data file has column headers.
    # Note: Plot will only display color-coded groups if the header "Type" is present
    # when data file is formatted with samples displayed in rows.
    radioButtons('sep', 'Separator:',
                 c(Comma=',',
                   Semicolon=';',
                   Tab='\t'),
                 ','),
    tags$hr(),

    # Collects sample formatting data for proper analysis.
    # For PCA function, samples should be listed one per row.
    radioButtons('dir', 'Genes are listed in:',
                 c(Rows = 'R',
                   Columns = 'C'),
                 'R'),

    tags$hr(),

    # Determines whether counts file represents normalized or unnormalized data.
    # Normalized data are not processed further prior to PCA.
    # Unnormalized data are normalized by the app using the TMM method as recommended
    # by Peter Linsley of BRI using function written by Kristen Dang.
    radioButtons('norm', 'Apply TMM Normalization:',
                 c(No = 'No_Norm',
                   Yes = 'Norm'),
                 'No_Norm'),
    tags$hr(),

    # Allows user to input a custom title for the plot.
    # If left blank, no title will be displayed.
    textInput('title', "Plot Title", ''),

    # Suppresses error messages about NULL while awaiting user selection.
    # Credit to: Joe Cheng[R Studio] at
    # https://groups.google.com/forum/#!topic/shiny-discuss/FyMGa2R_Mgs
    tags$style(type="text/css",
               ".shiny-output-error { visibility: hidden; }",
               ".shiny-output-error:before { visibility: hidden; }"
    ),
    width = 3
),

# These elements will display in the main panel (R side of page).
mainPanel(
    uiOutput('outSelections'),
    tags$hr(),
    plotOutput('boxPlot'),
    downloadButton('download1', 'Download', class = "buttCol"),
    tags$hr(),
    fluidRow(
       column(4,
```

```
                        radioButtons('customY', "Set Custom Y Range",  c(No = 'NotCustom', Yes =
'Custom'), 'NotCustom', inline = TRUE)
                    ),
                    column(7, offset = 1,
                        conditionalPanel(
                        condition = "input.customY == 'Custom'",
                        uiOutput('yMaxSelector')
                        )

                    )
                ),  # end fluidRow
                conditionalPanel(
                    condition = "input.customY == 'Custom'",
                    plotOutput('boxPlotCustomY'),
                    downloadButton('download2', 'Download', class = "buttCol")
                ),
                tags$head(tags$style(".buttCol{background-color:#faf0e6;} .buttCol{color: black;}")),
                tags$hr(),
                tableOutput('table'),
                tags$hr()
              ) # end mainPanel
            ) # end sidebarLayout

    ), # end tabPanel
    tabPanel("Help",
            tags$h1("File Formatting Issues", style = "color:darkblue"),
            tags$p(" It is anticipated that most issues that arise while trying to use this application will be
related
                to file formatting. A very specific file format is required. This format has several variations,
                depending on the type of data included in the file. The figures below illustrate accepatable
file formats,
                as they would appear in a spreadsheet program (such as Excel) before being coverted
into one of the accepted
                file formats (.csv, .tsv, or .txt).", style="font-size:20px"),
            tags$img(height = 443, width = 904, src = "Permitted.PNG"),
            tags$p(" In the figure above, genes are arranged by column and samples by row. Transposing
this arrangement,
                so that genes are in rows, while samples are in columns is permitted and should not cause
any issues
                in the functioning of the application.", style="font-size:20px")
            )  # end tabPanel
            )  # end tabsetPanel
            )  #end fluidPage


#---------------------------------------------- SERVER CODE ---------------------------------------
server <- function(input,output){
   # Function to read data from a file, after checking file in not NULL.
   getData <- function(dataFile){
      inFile <- dataFile
      if (is.null(inFile)) {
         return(NULL)
      }
      else {
```

```
        read.csv(inFile$datapath, header = TRUE, sep = input$sep, quote = input$quote, row.names = 1,
stringsAsFactors = FALSE)
      }
   }

   # Read in the data from the input file.
   myData <- reactive({getData(input$file1)})

   # Get data dimensions as 2 item list [#rows, rcols].
   dataDimensions <- reactive(dim(myData()))

   # Get information on whether sample data are in ROWS or COLUMNS.
   direction <- reactive({input$dir})

   #-----------------------------------------------------------------------------------------
   #          TASK ONE: FORMAT THE DATA
   #-----------------------------------------------------------------------------------------

   # Get the data into a "tidier" format with row names identified as such and
   # Sample/Gene identifiers separated from the numerical data table elements.

   # This section of code checks whether data are arranged with genes or samples in rows and
   # separates any type data present from the counts data.

   # 1) Check that ROW TYPE DATA is present
   # i.e. Last COLUMN contains TYPE data
   hasGroupDataInLastCol <- function(df){
      if(!is.null(df$Type)){
         return(TRUE)
      }
      return(FALSE)
   }

   # 2) Check that COLUMN TYPE DATA is present
   # i.e. Last ROW contains TYPE data
   hasGroupDataInLastRow <- function(df){
      numRows = dataDimensions()[1]
      if(row.names(df)[numRows] == "Type"){
         return(TRUE)
      }
      return(FALSE)
   }

   # 3) Now, we're ready to do the actual reformatting:
   # Remove the first column from the data frame -- it should contain text (gene or sample names).
   # If it contains data, the input file is NOT PROPERLY FORMATTED!
   # At the end of this code block, only numerical values should be in the body of the data frame.
   # The data frame's column and row names should now be appropriately recognized as such by R.
   formattedData <- function(){

      dimensions <- dim(myData())
      numRows <- dimensions[1]
      numCols <- dimensions[2]
```

```
    preformatted <- myData()
    # If there is GROUP data in the last column, it needs to be removed.
    if(isTRUE(hasGroupDataInLastCol(preformatted)) == TRUE) {
       numCols = numCols - 1
    }
    # If there is group data in the last row, it needs to be removed.
    if(isTRUE(hasGroupDataInLastRow(preformatted)) == TRUE){
       numRows = numRows - 1
    }
    # reformat
    preformatted <- preformatted[1:numRows, 1:numCols]
    # get the row names and store them as "r_names"
    r_names <- row.names(preformatted)

    preformatted <- lapply(preformatted, as.numeric)
    preformatted <- as.data.frame(preformatted)

    # Add back the row name info as formal row names since they got lost
    # in the two steps above.
    row.names(preformatted) <- r_names
    if(input$dir == 'C'){
       preformatted <- t(preformatted)
    }
    return(preformatted)
  }

  #-----------------------------------------------------------------------------------
  #          TASK TWO: BE READY TO NORMALIZE DATA IF THIS OPTION IS SELECTED
  #-----------------------------------------------------------------------------------

  # Normalize the counts data using TMM, if user selects this option.
  # TMM normalization requires that genes be in ROWS -- This is very important for correct functioning
  # of the TMM algorithm. In addition to incorrect results, improper orientation could cause the program
  # to crash if a gene has zero counts across all samples.
  normalized <- function(){

    counts_m <- formattedData()

    dge <- DGEList(counts=counts_m)
    # can now use TMM to normalize the counts data
    # this function Calculates normalization factors to scale the raw library sizes
    # using the weighted trimmed mean of M-values (to the reference),
    # where the weights are from the delta method on Binomial data
    data.dge = calcNormFactors(dge, method=c("TMM"), refColumn = NULL,logratioTrim = .3, sumTrim
= 0.05, doWeighting=TRUE, Acutoff=-1e10, p=0.75)
    #### NEXT: Use this function from Kristen Dang ####
    # This function uses the library size from the datafile and the
    # normalization factors calculated to return a matrix of normalized counts
    # See DGEList documentation: default for lib.size = colSums(counts)
    # default norm.factors = rep(1, ncol(counts)) (i.e. a vector of '1's)
    return_TMM_matrix_noRound=function(inDGEObject){
       eff.libsize = inDGEObject$samples$lib.size * inDGEObject$samples$norm.factors
       tndata = 1e6*t(inDGEObject$counts)/eff.libsize
       return(t(tndata))
```

```
        }

        # Call the function, then convert matrix obtained to a data frame:
        data.tmm = return_TMM_matrix_noRound(data.dge) #### A function from Kristen Dang
        data.tmm = as.data.frame(data.tmm)

        return(data.tmm)
    }

    #-----------------------------------------------------------------------------------------
    #           TASK THREE: PREPARE GENE NAMES AND SLIDER INPUT MAX VALUE
    #-----------------------------------------------------------------------------------------

    # Get gene names for drop down menu from file read in to the app.
    output$outSelections <- renderUI({
        genes <- row.names(formattedData())
        selectInput("genes", "Please click the box below for drop-down menu to select genes to plot:",
choices = genes, multiple=TRUE, selectize=TRUE)

    })

    # Get maximum count value in data to use as y-max for slider input widget.
    output$yMaxSelector <- renderUI({
        if(input$norm == 'Norm'){
            maxNorm <- round(max(normalized()), 0)
            #print("Printing maxNorm:")
            #print(maxNorm)
            sliderInput('yRange', "To change range of y-axis, enter a new maximum value", 0, maxNorm,
value = maxNorm)
        }
        else {
            maxForm <- round(max(formattedData()), 0)
            #print("Printing maxForm")
            #print(maxForm)
            sliderInput('yRange', "To change range of y-axis, enter a new maximum value", 0, maxForm,
value = maxForm)
        }
    })

    #-----------------------------------------------------------------------------------------
    #           TASK FOUR: CREATE BOXPLOT INPUTS
    #-----------------------------------------------------------------------------------------

    # Create the boxplots
    boxPlotInput <- function(){

        # Build the y-axis label
        if(input$norm == 'Norm'){
            normalBool = "Normalized"
            dataForPlot <- normalized()
        }
        else {
            normalBool = ""
            dataForPlot <- formattedData()
```

```
        }

        # Combine y-axis label components
        yaxisLabel = paste(normalBool," Gene Expression Counts ", sep="")

        dataForPlot <- t(dataForPlot)
        dataToPlot <- dataForPlot[,input$genes]

        boxplot(dataToPlot, col="lightblue", pars = list(boxwex = 0.6), main=input$title, xlab="Genes", ylab =
yaxisLabel)
    }

    # Creates boxplot with customized y-axis range.
    boxPlotCustomYInput <- function(){
        # store the data to plot
        dataForPlot <- formattedData()
        # Build the y-axis label
        if(input$norm == 'Norm'){
            normalBool = "Normalized"
            dataForPlot <- normalized()
        }
        else {
            normalBool = ""
            dataForPlot <- formattedData()
        }

        # Combine y-axis label components
        yaxisLabel = paste(normalBool," Gene Expression Counts ", sep="")
        dataForPlot <- t(dataForPlot)
        dataToPlot <- dataForPlot[,input$genes]

        boxplot(dataToPlot, col="lightblue", pars = list(boxwex = 0.6), main=input$title, xlab="Genes", ylab =
yaxisLabel, ylim = c(0, input$yRange))
    }

    #-------------------------------------------------------------------------------------------
    #           TASK FIVE: DISPLAY BOXPLOTS
    #-------------------------------------------------------------------------------------------

    output$boxPlot <- renderPlot({
        boxPlotInput()
    })

    output$boxPlotCustomY <- renderPlot({
        boxPlotCustomYInput()
    })

    #-------------------------------------------------------------------------------------------
    #           TASK SIX: DOWNLOAD PLOTS
    #-------------------------------------------------------------------------------------------

    # Download handler for boxplot plot no scaling (top plot).
    output$download1 <- downloadHandler(
        filename = 'downloadedPlot.pdf',
```

```
    content = function(file) {
      pdf(file)
      dataForPlot <- formattedData()
      # Build the y-axis label
      if(input$norm == 'Norm'){
        normalBool = "Normalized"
        dataForPlot <- normalized()
      }
      else {
        normalBool = ""
        dataForPlot <- formattedData()
      }

      # Combine y-axis label components
      yaxisLabel = paste(normalBool," Gene Expression Counts ", sep="")
      dataForPlot <- t(dataForPlot)
      dataToPlot <- dataForPlot[,input$genes]

      boxplot(dataToPlot, col="lightblue", main=input$title, xlab="Genes", ylab = yaxisLabel)
      dev.off()
    }
  )

  # Download handler for boxplot plot with scaling (bottom plot).
  output$download2 <- downloadHandler(
    filename = 'downloadedPlot.pdf',
    content = function(file) {
      pdf(file)
      dataForPlot <- formattedData()
      # Build the y-axis label
      if(input$norm == 'Norm'){
        normalBool = "Normalized"
        dataForPlot <- normalized()
      }
      else {
        normalBool = ""
        dataForPlot <- formattedData()
      }

      # Combine y-axis label components
      yaxisLabel = paste(normalBool," Gene Expression Counts ", sep="")
      dataForPlot <- t(dataForPlot)
      dataToPlot <- dataForPlot[,input$genes]

      boxplot(dataToPlot, col="lightblue", main=input$title, xlab="Genes", ylab = yaxisLabel, ylim = c(0,
input$yRange))
      dev.off()

    }
  )

  #-------------------------------------------------------------------------------------
  #          TASK SEVEN: DISPLAY DATA FILE
  #-------------------------------------------------------------------------------------
```

```
    # Display data table below plot.
    # Data are displayed in the same format as they are in the input file.
    output$table <- renderTable(
        if(input$dir == 'C'){
            t(formattedData())
        }
        else {
            formattedData()
        }
    )
}

shinyApp(ui = ui, server = server)
```

7.3.2 Principal Components Analysis Application Code

```
# Emilia F Gan
# Student Number: 1138607
# Email: efgan@uw.edu
# Copyright E Gan 2016
# Submitted in partial fulfillment of the requirements
# of the degree Master of Science

# Principal Component Analysis (PCA) Analysis

# Opening a CSV File and reading into R
# Modeled after: http://shiny.rstudio.com/gallery/upload-file.html
# Downloading plots learned from: https://www.youtube.com/watch?v=LSnWGmVkB6A

library(shiny)
library(ggplot2) # for plotting color plots with legend
library(edgeR) # for DGE
library(limma)

# By default, the file size limit is 5MB. It can be changed by
# setting this option. Here we'll raise limit to 9MB.
options(shiny.maxRequestSize = 9*1024^2)

#---------------------------------------------------- UI CODE ----------------------------------------------------
ui <- fluidPage(
   titlePanel("PCA Analysis"),
   hr(),
   tabsetPanel(
      tabPanel("PCA Plot",
            sidebarLayout(

               # These elements will display in the sidebar panel at the L of the page.
               sidebarPanel(
                  # Allows user to select data file from local file system.
                  fileInput('file1', 'Choose file to upload',
                        accept = c(
                           'text/csv',
                           'text/comma-separated-values',
                           'text/tab-separated-values',
                           'text/plain',
                           '.csv',
                           '.tsv'
                        )
                  ),
                  tags$hr(),

                  # Note: Plot will only display color-coded groups if the header "Type" is
                  # present when data file is formatted with samples displayed in rows.
                  radioButtons('sep', 'Separator:',
                           c(Comma=',',
                             Semicolon=';',
                             Tab='\t'),
```

```
            ','),
tags$hr(),

# Collects formatting data for proper analysis.
radioButtons('geneDir', 'Genes are listed in:',
            c(Rows = 'Row',
              Columns = 'Col'),
            'Row'),

tags$hr(),


# Collects information on how to do PCA.
radioButtons('dir', 'Display PCA Results For:',
            c(Rows = 'R',
              Columns = 'C'),
            'C'),

tags$hr(),

# Determines whether counts file represents normalized or
   # unnormalized data.
# Normalized data are not processed further prior to PCA.
# Unnormalized data are normalized by the app using the TMM
    # method as recommended
# by Peter Linsley of BRI using function written by Kristen Dang.
radioButtons('norm', 'Apply TMM Normalization:',
            c(No = 'No_Norm',
              Yes = 'Norm'),
            'No_Norm'),
tags$hr(),

# If no grouping data provided for samples, default color is BLACK.
# If grouping data are provided, but color plot output not desired,
# the color and color key can be suppressed by selecting 'Black' radio button.
radioButtons('col', 'Display points:',
            c(Black = 'BLK',
              Color = 'CLR'),
            'BLK'),

# Option to have points on plot displayed with different symbols for
   # each sample subtype.
conditionalPanel(
    condition = "input.col == 'CLR'",
    checkboxInput('Shapes', 'Shapes', FALSE)
),

tags$hr(),

# Allows user to input a custom title for the plot.
# If left blank, no title will be displayed.
textInput('title', "Plot Title", ''),

# Suppresses error messages about NULL while awaiting user
```

```
                # selection.
                # Credit to: Joe Cheng[R Studio] at
                # https://groups.google.com/forum/#!topic/shiny-
                # discuss/FyMGa2R_Mgs
                tags$style(type="text/css",
                        ".shiny-output-error { visibility: hidden; }",
                        ".shiny-output-error:before { visibility: hidden; }"
                ),
                width = 3
            ),

            # These elements will display in the main panel (R side of page).
            mainPanel(
                plotOutput('pcaPlot',click = "plot_click"),
                conditionalPanel(
                    condition = "input.col == 'CLR'",
                    downloadButton('downloadColor', 'Download the color plot as a pdf', class = "buttCol")
                ),
                conditionalPanel(
                    condition = "input.col == 'BLK'",
                    downloadButton('downloadBlack', 'Download the black & white plot as a pdf', class =
"buttCol")
                ),
                tags$head(tags$style(".buttCol{background-color:#faf0e6;} .buttCol{color: black;}")),
                tags$hr(),
                conditionalPanel(
                    condition = "input.col == 'CLR'",
                    textOutput('warning')
                ),
                verbatimTextOutput("info"),
                tags$hr(),
                tableOutput('table')
            )  # end mainPanel
        )  # end sidebarLayout
    ),  # end first tabPanel
    tabPanel("Help",
        tags$h1("File Formatting Issues", style = "color:darkblue"),
        tags$p(" It is anticipated that most issues that arise while trying to use this application will be
            related to file formatting. A very specific file format is required. This format has several
variations, depending on the type of data included in the file. The figures below illustrate
accepatable file formats, as they would appear in a spreadsheet program (such as Excel) before
being coverted into one of the accepted file formats (.csv, .tsv, or .txt).", style="font-size:20px"),
        tags$img(height = 414  , width = 671 , src = "Permitted.PNG"),
        tags$p(" In the figure above, genes are arranged by column and samples by row.
Transposing this arrangement,
            so that genes are in rows, while samples are in columns is permitted and should not
cause any issues
            in the functioning of the application.", style="font-size:20px")
        )
        )  # end tabsetPanel
        )  # end fluidPage

#---------------------------------------------- SERVER CODE ----------------------------------------
server <- function(input,output){
```

```
# Function to read data from a file, after checking file in not NULL.
getData <- function(dataFile){
   inFile <- dataFile
   if (is.null(inFile)) {
      return(NULL)
   }
   else {
      read.csv(inFile$datapath, header = TRUE, sep = input$sep, quote = input$quote, row.names =
  1, stringsAsFactors = FALSE)
   }
}

# Read in the data from the input file.
myData <- reactive({getData(input$file1)})

# Get data dimensions as 2 item list [#rows, rcols].
dataDimensions <- reactive(dim(myData()))

# Get information on whether sample data are in ROWS or COLUMNS.
direction <- reactive({input$dir})

#-----------------------------------------------------------------------------------------
#          TASK ONE: FORMAT THE DATA
#-----------------------------------------------------------------------------------------

# Get the data into a "tidier" format with row names identified as such and
# Sample/Gene identifiers separated from the numerical data table elements.

# 1) Check that ROW TYPE DATA is present
# i.e. Last COLUMN contains TYPE data
hasGroupDataInLastCol <- function(df){
   if(!is.null(df$Type)){
      return(TRUE)
   }
   return(FALSE)
}

# 2) Check that COLUMN TYPE DATA is present
# i.e. Last ROW contains TYPE data
hasGroupDataInLastRow <- function(df){
   numRows = dataDimensions()[1]
   if(row.names(df)[numRows] == "Type"){
      return(TRUE)
   }
   return(FALSE)
}

# 3) Get GROUP TYPE information from last COLUMN
# (check if present first using hasGroupDataInLastCol)
# Need to specify to R that these should be considered a factor with levels.
getGroupInfoLastCol <- function(df){
   dimensions <- dim(df)
   numRows <- dimensions[1]
   if(isTRUE(hasGroupDataInLastRow(df))) {
```

```
        return(as.factor(df$Type[1:numRows-1]))
    }
    else {
        return(as.factor(df$Type))
    }
}

# 4) Get GROUP TYPE information from last ROW
# (check if present first using hasGroupDataInLastRow before calling this function)
# Need to specify to R that these should be considered a factor with levels.
getGroupInfoLastRow <- function(df){
    dimensions <- dim(df)
    numRows <- dimensions[1]
    numCols <- dimensions[2]

    if(isTRUE(hasGroupDataInLastCol(df))){
        lastRowInfo <- df[numRows,1:numCols-1]
    } else {
        lastRowInfo <- df[numRows,1:numCols]
    }

    lastRowInfo <- data.frame(t(lastRowInfo))

    return(t(lastRowInfo$Type))
}

# 5) Now, we're ready to do the actual reformatting:
# Remove the first column from the data frame -- it should contain text (gene or sample names).
# If it contains data, the input file is NOT PROPERLY FORMATTED!
# At the end of this code block, only numerical values should be in the body of the data frame.
# The data frame's column and row names should now be appropriately recognized as such by R.
formattedData <- function(){
    dimensions <- dim(myData())
    numRows <- dimensions[1]
    numCols <- dimensions[2]

    formatted <- myData()
    # If there is GROUP data in the last column, it needs to be removed.
    if(isTRUE(hasGroupDataInLastCol(formatted)) == TRUE) {
        numCols = numCols - 1
    }
    # If there is group data in the last row, it needs to be removed.
    if(isTRUE(hasGroupDataInLastRow(formatted)) == TRUE){
        numRows = numRows - 1
    }
    # reformat
    formatted <- formatted[1:numRows, 1:numCols]
    # get the row names and store them as "r_names"
    r_names <- row.names(formatted)

    formatted <- lapply(formatted, as.numeric)
    formatted <- as.data.frame(formatted)

    # Add back the row name info as formal row names since they got lost
```

```
    # in the two steps above.
    row.names(formatted) <- r_names

    return(formatted)
}

# Normalize the counts data using TMM, if user selects this option.
normalized <- function(){
    if(input$geneDir == 'Col') {
        counts_m <- t(formattedData())
    }
    else {
        counts_m <- formattedData()
    }
    dge <- DGEList(counts=counts_m)
    # can now use TMM to normalize the counts data
    # this function Calculates normalization factors to scale the raw library sizes
    # using the weighted trimmed mean of M-values (to the reference),
    # where the weights are from the delta method on Binomial data
    data.dge = calcNormFactors(dge, method=c("TMM"), refColumn = NULL,logratioTrim = .3,
  sumTrim = 0.05, doWeighting=TRUE, Acutoff=-1e10, p=0.75)
    #### NEXT: Use this function from Kristen Dang ####
    # This function uses the library size from the datafile and the
    # normalization factors calculated to return a matrix of normalized counts
    # See DGEList documentation: default for lib.size = colSums(counts)
    # default norm.factors = rep(1, ncol(counts)) (i.e. a vector of '1's)
    return_TMM_matrix_noRound=function(inDGEObject){
        eff.libsize = inDGEObject$samples$lib.size * inDGEObject$samples$norm.factors
        tndata = 1e6*t(inDGEObject$counts)/eff.libsize
        return(t(tndata))
    }

    # Call the function, then convert matrix obtained to a data frame:
    data.tmm = return_TMM_matrix_noRound(data.dge) #### A function from Kristen Dang
    data.tmm = as.data.frame(data.tmm)

    if(input$geneDir == 'Col'){
        return(t(data.tmm))
    }

    return(data.tmm)
}

#-----------------------------------------------------------------------------------------
#           TASK TWO: PERFORM PCA
#-----------------------------------------------------------------------------------------

# Perform PCA
# If 'R' selected, remember to transpose the data.
pcaTest <- function(){
    if(input$norm == 'Norm'){
        #forPCA <- as.data.frame(normalized())
        forPCA <- normalized()
    }
```

```
    else {
        forPCA <- formattedData()
    }

    if(input$dir == 'R'){
        pr.out <- prcomp(t(forPCA))
    }
    else {
        pr.out <- prcomp(forPCA)
    }
    return(pr.out)
}

# create a data frame with the sample names and PC1 and PC2 eigenvectors
# Number of rows will vary depending on whether plotting samples or genes in PC1 v Pc2 plot.
dataframe <- reactive({
    if(input$geneDir == 'Row') {
        if(input$dir == 'R'){
            if(input$norm != 'Norm'){
                Names <- row.names(formattedData())

            }
            else {
                Names <- row.names(normalized())
            }
        }
        else {
            if(input$norm != 'Norm'){
                Names <- colnames(formattedData())
            }
            else {
                Names <- colnames(normalized())
            }
        }
    }
    else {
        if(input$dir == 'R'){
            if(input$norm != 'Norm'){
                Names <- row.names(formattedData())

            }
            else {
                Names <- row.names(normalized())
            }
        }
        else {
            if(input$norm != 'Norm'){
                Names <- colnames(formattedData())
            }
            else {
                Names <- colnames(normalized())
            }
        }
    }
```

```
    data.frame("PC_1" = pcaTest()$rotation[,1], "PC_2" = pcaTest()$rotation[,2], row.names = Names)
})

# Get the proportions of variance info for the axes labels
proportions <- reactive({
    cumsum(pcaTest()$sdev^2 / sum(pcaTest()$sdev^2))
})

#-------------------------------------------------------------------------------------
#           TASK THREE: PLOT PCA RESULTS AND DOWNLOAD PLOT
#-------------------------------------------------------------------------------------

# Get GROUP MEMBERSHIP information for COLOR-CODING the PC1 v PC2 Plot
typeInfo <- function(){
    # If elements to show in plot are arranged in Rows and attributes in Columns:
    if(input$dir == 'R') {
        #   then Element Group data located in last column
        if(isTRUE(hasGroupDataInLastCol(myData()))){
            Groups = getGroupInfoLastCol(myData())
        }
        else{
            Groups = NULL
        }
    }
    # If elements to show in plot are arranged in Columns and attributes in Rows:
    else {
        #   then Element Group data in last row
        if(isTRUE(hasGroupDataInLastRow(myData()))){
            Groups = t(getGroupInfoLastRow(myData()))
        }
        else{
            Groups = NULL
        }
    }
    return(Groups)
}

# Display noted to user that color selections are only functional
# when appropriate group type data is supplied.
output$warning <- renderText({
    if(input$col == 'CLR' && is.null(typeInfo())){
        "COLOR options will be non-functional as no GROUPING DATA has been supplied."
    }
    else {""}
})

# Plot the 2 pricipal components.
plotInput = function(){
    axes <- proportions()
    xaxis <- axes[1]
    yaxis <- axes[2] - xaxis
    df = dataframe()

    if(is.null(typeInfo()) | input$col == "BLK") {
```

```
        print("TypeInfo seems to be null")
        plot(df$PC_1, df$PC_2, pch = 19, cex = 1.5,main = input$title, xlab = paste(
            "PC 1 (", round(xaxis,3)*100, "%)", sep=""), ylab = paste("PC 2 (", round(yaxis,3)*100, "%)",
  sep=""))

    } else {

        Key = typeInfo()

        if(input$Shapes == TRUE){
            ggplot(df, aes(x = df$PC_1, y = df$PC_2)) +
                geom_point(mapping = aes(colour = Key, shape = Key), size = 5) + labs(x = paste("PC 1 (",
  round(xaxis,3)*100, "%)", sep=""), y = paste("PC 2 (", round(yaxis,3)*100, "%)", sep="")) +
                labs(title = input$title) + theme_bw()
        }
        else{
            ggplot(df, aes(x = df$PC_1, y = df$PC_2)) +
                geom_point(mapping = aes(colour = Key), size = 5) + labs(x = paste("PC 1 (",
  round(xaxis,3)*100, "%)", sep=""), y = paste("PC 2 (", round(yaxis,3)*100, "%)", sep="")) +
                labs(title = input$title) + theme_bw()
        }
    }
}

output$pcaPlot <- renderPlot({
    plotInput()
})

# Display point data for points on plot close to location of mouse click
# If multiple points are located very close together, all will be listed.
output$info <- renderPrint({
    nearPoints(dataframe(), input$plot_click, xvar = "PC_1" , yvar = "PC_2")
})

#--------------------------------------------------------------------------------
# Download handler for ggplot2 plot.
output$downloadColor <- downloadHandler(
    filename = 'downloadedPlot.pdf',
    content = function(file) {
        device <- function(..., width, height) {
            grDevices::pdf(..., width = width, height = height)
        }
        ggsave(file, plot = plotInput(), device = device)
    }
)

# Download handler for basic R plot.
output$downloadBlack <- downloadHandler(
    filename = 'downloadedPlot.pdf',
    content = function(file) {
        pdf(file)
        plot(dataframe()$PC_1, dataframe()$PC_2, pch = 19, cex = 1.5,main = input$title, xlab = paste(
            "PC 1 (", round(proportions()[1],3)*100, "%)", sep=""), ylab = paste("PC 2 (",
  round((proportions()[2] - proportions()[1]),3)*100, "%)", sep=""))
```

```
            dev.off()

        }
    )

    # Display data table below plot.
    output$table <- renderTable(formattedData())
}

shinyApp(ui = ui, server = server)
```

7.4 Appendix D: Supplemental Materials

Code files and User's Guide for the Box Plot Shiny Application may be downloaded from: https://github.com/efran/Shiny-Boxplot-Application

Code files and User's Guide for the Principal Components Analysis Application may be downloaded from: https://github.com/efran/Shiny-PCA-Application

Both applications can also be viewed and used online at present. However, the site hosting them has limited monthly "active hours" under the free hosting plan, so the application is not guaranteed to be accessible there at all times. The links for the online apps are provided below:

https://efg-ds.shinyapps.io/boxplotApp/

https://efg-ds.shinyapps.io/pcaApp/

8. References

1. Damania, Blossom and James M. Pipas (eds.) *DNA Tumor Viruses*. Springer, 2009.
2. Uldrick, Thomas, S. and Denise Whitby. "Update on KSHV epidemiology, Kaposi Sarcoma pathogenesis, and treatment of Kaposi Sarcoma." *Cancer Letters* (2011): 150-162.
3. Minhas, Veenu and Charles Wood. "Epidemiology and Transmission of Kaposi's Sarcoma-Associated Herpesvirus." *Viruses* (2014): 4178-4194.
4. Dourmishev, Lyubomir A., et al. "Molecular Genetics of Kaposi's Sarcoma-Associated Herpesvirus (Human Herpesvirus 8) Epidemiology and Pathogenesis." *Microbiology and Molecular Biology Reviews* (2003): 175-212.
5. Chandriani, Sanjay and Ganem. Don. "Array-Based Transcript Profiling and Limiting-Dilution Reverse Transcription-PCR Analysis Identify Additional Latent Genes in Kaposi's Sarcoma-Associated Herpesvirus." *Journal of Virology* (2010): 5565-5573.
6. Rossetto, Cyprian C. and Gregory S. Pari. "Kaposi's Sarcoma-Associated Herpesvirus Noncoding Polyadenylated Nuclear RNA Interacts with Virus- and Host Cell-Encoded Proteins and Suppresses Expression of Genes Involved in Immune Modulation." *Journal of Virology* (2011): 13290-13297.
7. Taylor, Jennifer L., et al. "Transcriptional Analysis of Latent and Inducible Kaposi's Sarcoma-Associated Herpesvirus Transcripts in the K4 to K7 Region." *Journal of Virology* (2005): 15099-15106.
8. Wen, Kwun Wah and Blossom Damania. "Kaposi sarcoma-associated herpesvirus (KSHV): Molecular biology and oncogenesis." *Cancer Letters* (2010): 140-150.
9. Majerciak, Vladimir, et al. "A Viral Genome Landscape of RNA Polyadenylation from KSHV Latent to Lytic Infection." *PLOS Pathogens* (2013): e1003749.
10. McClure, Lydia V., et al. "Comprehensive Mapping and Analysis of Kaposi's Sarcoma-Associated Herpesvirus 3' UTRs Identify Differential Posttranscriptional Control of Gene Expression in Lytic versus Latent Infection." *Journal of Virology* (2013): 12838-12849.

11. Pan, Hongyi, Fuchun Zhou and Shou-Jiang Gao. "Kaposi's Sarcoma-Associated Herpesvirus Induction of Chromosome Instability in Primary Human Endothelial Cells." *Cancer Research* (2004): 4064-4068.
12. Mutlu, Agata D'Agostino and et al. "In Vivo-Restricted and Reversible Malignancy Induced by Human Herpesvirus-8 KSHV: A Cell and Animal Model of Virally Induced Kaposi's Sarcoma." *Cancer Cell* (2007): 245-258.
13. Carroll, Patrick A., Elizabeth Brazeau and Michael Lagunoff. "Kaposi's sarcoma-associated herpesvirus infection of blood endothelial cells induces lymphatic differentiation." *Virology* (2004): 7-18.
14. DiMaio, Terri A. and Michael Lagunoff. "KSHV induction of angiogenic and lymphangiogenic phenotypes." *Frontiers in Microbiology* (2012).
15. Cesarman, Ethel. "Gammaherpesvirus and lymphoproliferative disorders in immunocompromised patients." *Cancer Letters* (2011): 163-174.
16. Greene, Whitney, et al. "Molecular Biology of KSHV in Relation to AIDS-Associated Oncogenesis." *Cancer Treat Res.* (2007): 133:69-127.
17. Arvey, Aaron and et al. "An Atlas of the Epstein-Barr Virus Transcriptome and Epigenome Reveals Host-Virus Regulatory Interactions." *Cell Host & Microbe* (2012): 233-245.
18. Chang, Henry H. and Don Ganem. "A Unique Herpesviral Transcriptional Program in KSHV-Infected Lymphatic Endothelial Cells Leads to mTORC1 Activation and Rapamycin Sensitivity." *Cell Host & Microbe* (2013): 429-440.
19. Johnson, Andrew S., Nicole Maronian and Jeffrey Vieira. "Activation of Kaposi's Sarcoma-Associated Herpesvirus Lytic Gene Expression during Epithelial Differentiation." *Journal of Virology* (2005): 13769-13777.
20. Rossetto, Cyprian C., et al. "Regulation of Viral and Cellular Gene Expression by Kaposi's Sarcoma-Associated Herpesvirus Polyadenylated Nuclear RNA." *Journal of Virology* (2013): 5540-5553.
21. Tang, Shuang and Zhi-Ming Zheng. "Kaposi's Sarcoma-associated Herpesvirus K8 Exon 3 Contains Three 5'-Splice Sites and Harbors a K8.1 Transcription Start Site." *The Journal of Biological Chemistry* (2002): 14547-14556.

22. Dittmer, Dirk, et al. "A Cluster of Latently Expressed Genes in Kaposi's Sarcoma-Associated Herpesvirus." *Journal of Virology* (1998): 8309-8315.
23. Kedes, Dean H., et al. "Identification of the Gene Encoding the Major Latency-associated Nuclear Antigen of the Kaposi's Sarcoma-associated Herpesvirus." *The Journal of Clinical Investigation* (1997): 2606-2610.
24. Bechtel, Jill, Adam Grundhoff and Don Ganem. "RNAs in the Virion of Kaposi's Sarcoma-Associated Herpesvirus." *Journal of Virology* (2005): 10138-10146.
25. Purushothaman, Pravinkumar, Suhani Thakker and Subhash C. Verma. "Transcriptome Analysis of Kaposi's Sarcoma-Associated Herpesvirus during De Novo Primary Infection of Human B and Endothelial Cells." *Journal of Virology* (2015): 3093-3111.
26. Renne, Rolf, et al. "The Size and Conformaiton of Kaposi's Sarcoma-Associated Herpesvirus (Human Herpesvirus 8) DNA in Infected Cells and Virions." *Journal of Virology* (1996): 8151-8154.
27. Adang, Laura A., Christopher H. Parsons and Dean H. Kedes. "Asynchronous Progression through the Lytic Cascade and Variations in Intracellular Viral Loads Revealed by High-Throughput Single-Cell Analysis of Kaposi's Sarcoma-Associated Herpesvirus Infeciton." *Journal of Virology* (2006): 10073-10082.
28. Renne, Rolf, et al. "Lytic Growth od Kaposi's sarcoma-associated herpesvirus (human herpesvirus 8) in culture." *Nature Medicine* (1996): 342-346.
29. Lukac, David M., et al. "Reactivation of Kaposi's Sarcoma-Associated Herpesvirus Infection from Latency by Expression of ORF 50 Transactivator, a Homolog of the EBV R Protein." *Virology* (1998): 304-312.
30. Majerciak, Vladimir, et al. "Kaposi's Sarcoma-Associated Herpesvirus ORF57 Functions as a Viral Splicing Factor and Promotes Expression of Intron-Containing Viral Lytic Genes in Spliceosome-Mediated RNA Splicing." *Journal of Virology* (2008): 2792-2801.
31. Parravicini, Carlo, et al. "Differential Viral Protein Expression in Kaposi's Sarcoma-Associated Herpesvirus-Infected Diseases." *American Journal of Pathology* (2000): 743-749.

32. Cock, P, et al. “The Sanger FASTQ file format for sequences with quality scores and the Solexa/Illumina FASTQ variants.” *Nucleic Acids Research* (2010): 38(6) 1767-1771.
33. Metzker, M. “Sequencing technologies – the next generation.” *Nature Reviews: Genetics* (2010): 11:31-46.
34. Rudy, G. A Hitchhiker’s Guide to Next Generation Sequencing – Part 2. 2010 Dec 9. In *@gabeinformatics: An “Our 2 SNPs…” Blog by Golden Helix*. Accessed 9 May 2016. http://blog.goldenhelix.com/?p=490
35. Robinson, James T., et al. Integrative Genomics Viewer. *Nat Biotechnol* (2011): 29(1):24-26.
36. Anders, Simon, Pyl, Paul Theodor, and Wolfgang Huber.”HTSeq – a Python framework to work with high-throughput sequencing data.” *Bioinformatics* (2014): 166-169.
37. Maguerat, S. and J. Bahler. "RNA Seq: from technology to biology." *Cell Mol Life Sci* (2010): 569-579.
38. Cole, Trapnell, et al. "Transcript assembly and quantification by RNA-Seq revelas unannotated transcripts and isoform switching during cell differentiation." *Nature Biotechnology* (2010): 511-515.
39. Wagner, G., Kin, K., and V Lynch. “Measurement of mRNA abundance using RNA-seq data: RPKM measure is inconsistent among samples.” *Theory Biosci* (2012): 10.1007/S12064-012-0612-3.
40. Raz, T, er al. “Protocol Dependence of Sequencing-Based Gene Expression Measurements. *PLOS One* (2011): e19287.
41. Robinson, Mark D. and Alicia Oshlack. "A scaling normalization method for differential expression analysis of RNA-seq data." *Genome Biology* (2010): 11:R25.
42. Dillies, Marie-Agnes et al. "A comprehensive evaluation of normalization methods for Illumina high-throughput RNA sequencing data analysis." *Briefings in Bioinformatics* (2012).
43. Aanes, Halvard, et al. "Normalization of RNA-Sequencing Data from Samples with Varying mRNA Levels." *PLOS One* (2014).

44. Bruce, A Gregory, et al. "Next-Generation Sequence Analysis of the Genome of RFHVMn, the Macaque Homolog of Kaposi's Sarcoma (KS)-Associated Herpesvirus, from a KS-Like Tumor of a Pig-Tailed Macaque." *Journal of Virology* (2013): 13676-13693.
45. Bruce, A Gregory, et al. "The ORF59 DNA polymerase processivity factor homologs of Old World primate RV2 rhadinoviruses are highly conserved nucelar antigens expressed in differentiated epithelium in infected macaques." *Virology Journal* (2009): 6:205.
46. DeMaster, Laura K. and Timothy M. Rose. "A critical Sp1 element in the rhesus rhadinovirus (RRV) Rta promoter confers high-level activity that correlates with cellular permissivity for viral replication." *Virology* (2014): 196-209.
47. Ajiro, Masahiko and Zhi-Ming Zheng. "Oncogenes and RNA Splicing of human tumor viruses." *Emerging Microbes and Infections* (2014): e63.
48. Arias, Carolina, et al. "KSHV 2.0: A Comprehensive Annotation of the Kaposi's Sarcoma-Associated Herpesvirus Genome Using Next-Generation Sequencing Reveals Novel Genomic and Funcitonal Features." *PLOS Pathogens* (2014): e1003847.
49. Bai, Zhiqiang, et al. "Genomewide Mapping and Screening of Kaposi's Sarcoma-Associated Herpesvirus (KSHV) 3' Untranslated Regions Identify Bicistronic and Polycistronic Viral Transcripts as Frequent Targets osf KSHV MicroRNAs." *Journal of Virology* (2014): 377-392.
50. Chang, Ting-Yu, et al. "Differentially regulated splice variants and systems biology analysis of Kaposi's sarcoma-associated herpesvirus-infected lymphatic endothelial cells." *Nucleic Acids Research* (2011): 6970-6985.
51. Giardine, B., et al. “Galaxy: a platform for interactive large-scale genome analysis.” Genome Research (2005): 1451-1455.
52. Blankenberg, Daniel, et al. “Manipulation of FASTQ data with Galaxy.” *Bioinformatics* (2010): 1783-1785.

53. Kim, D., et al. “TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology* (2013): doi 10.1186/gb-2013014040r36.
54. National Center for Biotechnology Information. Human herpesvirus 8, complete genome. 23 Apr 2010. www.ncbi.nlm.nih.gov/nuccore/139472801
55. Chang, Winston, et al. shiny: Web Application Framework for R. R package version 0.13.0 (2016). https://CRAN.R-projrct.org/package=shiny
56. Stein, Lincoln. “Generic Feature Format Version 3 (GFF3).” 26 Feb 2013. The Sequence Ontology Project. Accessed 9 May 2016. http://www.sequenceontology.org/gff3.shtml
57. R Core Team (2015). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.
58. James, Gareth, et al. *An Introduction to Statistical Learning*. New York: Springer, 2014.