Assignment 4

Dakota Hawkins & Mae Rose Gott

Welcome to Assignment 4

Before we begin

Unlike Assignments 1-3, you will no longer be provided with the RMarkdown code. All of the code blocks here are empty and accompanied by instructions for what is expected in the code block. Do not neglect your `main.R` skeleton file as the functions there will constitute most of the points you earn.

THE BULK OF YOUR GRADE WILL BE ON YOUR WORK IN `main.R`

Anything that is specified to occur in functions you create MUST be in the `main.R` file. While you theoretically CAN do some or all of the homework in this .Rmd file, it is bad practice to have your 'back end' coding in the same files as your 'front end' report. Your `report.html` may be referenced to visually check any graphical outputs you may generate for this and all future assignments, but your `main.R` functions will be run outside of their respective markdown environments when grading them.

1. Reading and filtering count data

To begin, let's read in the raw count matrix, named `verse_counts.tsv`, as a `tibble` by implementing the `read_data()` function in `main.R`. Then filter out all zero-variance genes with the `filter_zero_var_genes()` function, also in `main.R`. Filtering out zero-variance genes is necessary for several common analysis steps, such as Principal Component Analysis (PCA), and can be helpful to reduce the size of our dataset if memory issues are a constraint.

1a. import data

```
#Read in verse_counts.tsv and display the head of the resulting tibble
verse_counts <- read_tsv("/usr4/bf528/sv/Documents/BF_591_Assignment_4/verse_counts.tsv"
)
```

```
## Rows: 55416 Columns: 9
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: "\t"
## chr (1): gene
## dbl (8): vP0_1, vP0_2, vP4_1, vP4_2, vP7_1, vP7_2, vAd_1, vAd_2
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(verse_counts)
```

```
## # A tibble: 6 × 9
##   gene                 vP0_1 vP0_2 vP4_1 vP4_2 vP7_1 vP7_2 vAd_1 vAd_2
##   <chr>                <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 ENSMUSG00000102693.2     0     0     0     0     0     0     0     0
## 2 ENSMUSG00000064842.3     0     0     0     0     0     0     0     0
## 3 ENSMUSG00000051951.6    19    24    17    17    17    12     7     5
## 4 ENSMUSG00000102851.2     0     0     0     0     1     0     0     0
## 5 ENSMUSG00000103377.2     1     0     0     1     0     1     1     0
## 6 ENSMUSG00000104017.2     0     3     0     0     0     1     0     0
```

```
#Display the top 10 rows of verse counts
```

1b. Apply filter_zero_var_genes()

```
#Filter out zero variance genes using the function you created in `main.R
count_data <- filter_zero_var_genes(verse_counts)
no_header <- count_data[-c(1)] #we are removing the header of the count_data dataframe
head(count_data)
```

```
## # A tibble: 6 × 9
##   gene                 vP0_1 vP0_2 vP4_1 vP4_2 vP7_1 vP7_2 vAd_1 vAd_2
##   <chr>                <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 ENSMUSG00000051951.6    19    24    17    17    17    12     7     5
## 2 ENSMUSG00000102851.2     0     0     0     0     1     0     0     0
## 3 ENSMUSG00000103377.2     1     0     0     1     0     1     1     0
## 4 ENSMUSG00000104017.2     0     3     0     0     0     1     0     0
## 5 ENSMUSG00000103201.2     0     0     0     0     0     0     0     1
## 6 ENSMUSG00000103161.2     0     0     1     0     0     0     0     0
```

```
#display the head of the counts data from the zero variance filter function's output
```

2. Constructing sample meta data from sample names

It’s often beneficial to separate data modes from each other during analysis (i.e. to have one data frame containing gene expression data, and separate data frame for any gene or subject level information). Here we’ll construct a `tibble` from sample names- which are located in the columns of our count matrix. To help you along, `main.R` has the two helper functions for you to implement first, `timepoint_from_sample` which extracts the ages of the subjects (P0, P4, P7, and Ad) from the sample names and `sample_replicate` which extracts the sample replicate number (“1” or “2”). Then, use these functions to implement the `meta_info_from_labels` function and extract the sample name, timepoint, and replicate number from each sample name. The function should then return a tibble with all the relevant information.

```
#Create and display the sample meta data tibble
sample_meta <- meta_info_from_labels(colnames(no_header))
knitr::kable(sample_meta)
```

Sample	Timepoint	Replicate
vP0_1	P0	1
vP0_2	P0	2
vP4_1	P4	1
vP4_2	P4	2
vP7_1	P7	1
vP7_2	P7	2
vAd_1	Ad	1
vAd_2	Ad	2

3. Normalization

Often we will want to normalize gene counts between samples so that comparisons are more reliable. Given read counts are a relative measure of abundance, normalization is a necessary step during RNAseq analysis when samples may have differing numbers of total reads.

3a. Normalizing with CPM (counts per million)

Now that we've explored our dataset prior to count normalization, we should normalize our data to see how things change. A simple and intuitive approach to count normalization is *counts per million*: that is finding the number of counts for a given gene in a given sample for each million reads observed. It can be easily calculated using this formula

$$cpm = \frac{X_{i,j}}{\sum_{i=1}^{P} X_{i,j}} \cdot 10^6$$

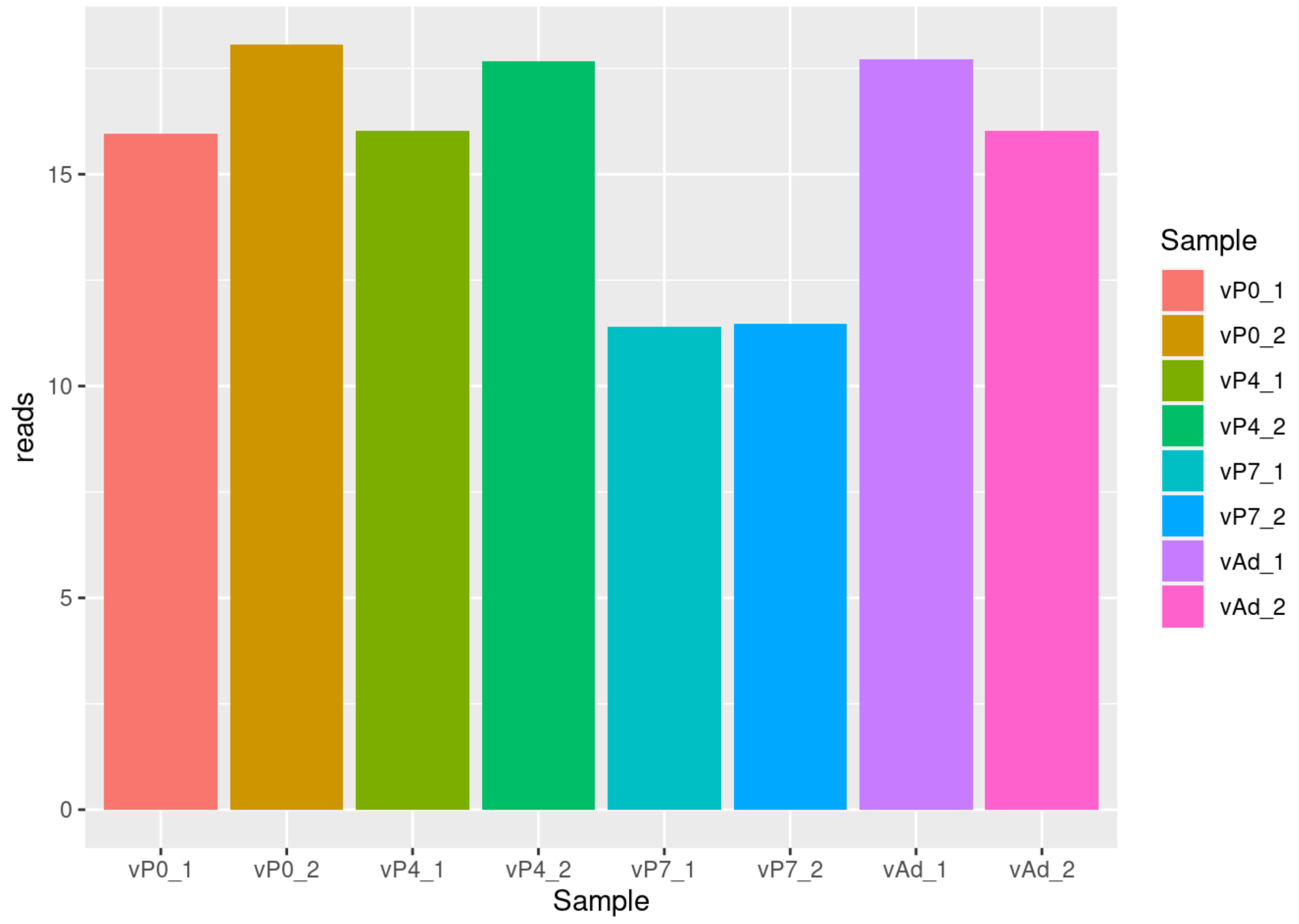
Where X is a $P \times N$ count matrix with genes as rows and samples as columns.

In other words, the counts per million is 10^6 * (percentage of reads that a gene represents in the sample's library). Therefore, for any gene within a sample, the CPM is calculated with: (#read for sample)/(#total reads for all filtered genes in sample) * 10^6

Plot library size

Implement the `get_library_size()` function and create a barplot of the number of millions of reads per sample. You may either create your own function to make the barplot or write the code in the RMarkdown at your discretion.

```
#Calculate the number of reads in each sample and plot as a barplot
#Make a barplot showing the number of millions of reads for each sample
library <- get_library_size(count_data)
pivot_longer(library/1000000, everything(), names_to = "Sample", values_to = "reads") %
>%
  mutate(Sample=factor(Sample, levels = colnames(library), exclude = NA, ordered = is.or
dered(Sample), nmax = NA)) %>%
  ggplot(aes(x = Sample, y = reads, fill = Sample)) +
  geom_bar(stat = "identity")
```



CPM calculation

Implement the `normalize_by_cpm` function to normalize a raw count matrix given the provided formula.

```
#Normalize your filtered counts matrix with CPM and display the head
cpm <- normalize_by_cpm(count_data)
head(cpm)
```

```
## # A tibble: 6 × 9
##   gene                  vP0_1 vP0_2 vP4_1 vP4_2 vP7_1 vP7_2 vAd_1 vAd_2
##   <chr>                 <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 ENSMUSG00000051951.6  2.25  3.18  2.12  2.12  2.58  1.72  1.45  1.34
## 2 ENSMUSG00000102851.2  1.06  1.06  1.06  1.06  1.17  1.06  1.06  1.09
## 3 ENSMUSG00000103377.2  1.12  1.06  1.09  1.11  1.06  1.11  1.13  1.09
## 4 ENSMUSG00000104017.2  1.06  1.25  1.09  1.06  1.06  1.18  1.06  1.06
## 5 ENSMUSG00000103201.2  1.09  1.06  1.06  1.06  1.06  1.09  1.06  1.12
## 6 ENSMUSG00000103161.2  1.09  1.06  1.12  1.09  1.06  1.06  1.06  1.06
```

3b. Normalizing with DESeq2

DESeq2 is a common and prolific tool for performing differential expression analysis (DEA) in RNAseq data. Within that process, DESeq2 implements its own normalization procedure prior to performing DEA for more reliably identified differential genes. You can read more about it here (http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html#data-transformations-and-visualization).

Normalizing with DESeq2

Implement the `deseq_normalize()` function to compute and extract the normalized counts using DESeq2. The `DESeq2DataSet` object requires a design formula to be specified. This formula is used as the differential expression model, but we don't need to provide a formula to normalize counts (http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html#do-normalized-counts-correct-for-variables-in-the-design). You may pass `~1` as the design formula to pass a trivial formula when creating the `DESeq2DataSet` object.

```
#Normalize your filtered data and display the head of the resulting tibble
dseqnormalized <- deseq_normalize(count_data, sample_meta)
```

```
## converting counts to integer mode
```

```
head(dseqnormalized)
```

```
## # A tibble: 6 × 9
##   gene                  vP0_1 vP0_2  vP4_1  vP4_2 vP7_1 vP7_2 vAd_1 vAd_2
##   <chr>                 <dbl> <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 ENSMUSG00000051951.6 15.9   18.2  15.3   13.1   22.3  15.5   7.44  6.04
## 2 ENSMUSG00000102851.2  0      0     0      0      1.31  0     0     0
## 3 ENSMUSG00000103377.2  0.839  0     0      0.773  0     1.29  1.06  0
## 4 ENSMUSG00000104017.2  0      2.28  0      0      0     1.29  0     0
## 5 ENSMUSG00000103201.2  0      0     0      0      0     0     0     1.21
## 6 ENSMUSG00000103161.2  0      0     0.900  0      0     0     0     0
```

4 Visualization

Now that the data has been filtered and normalized, plot your data in the methods described below. Your filtered data will be plotted first. You will be instructed to plot your CPM and DESeq-normalized afterwards, in part 5 and 6 below.

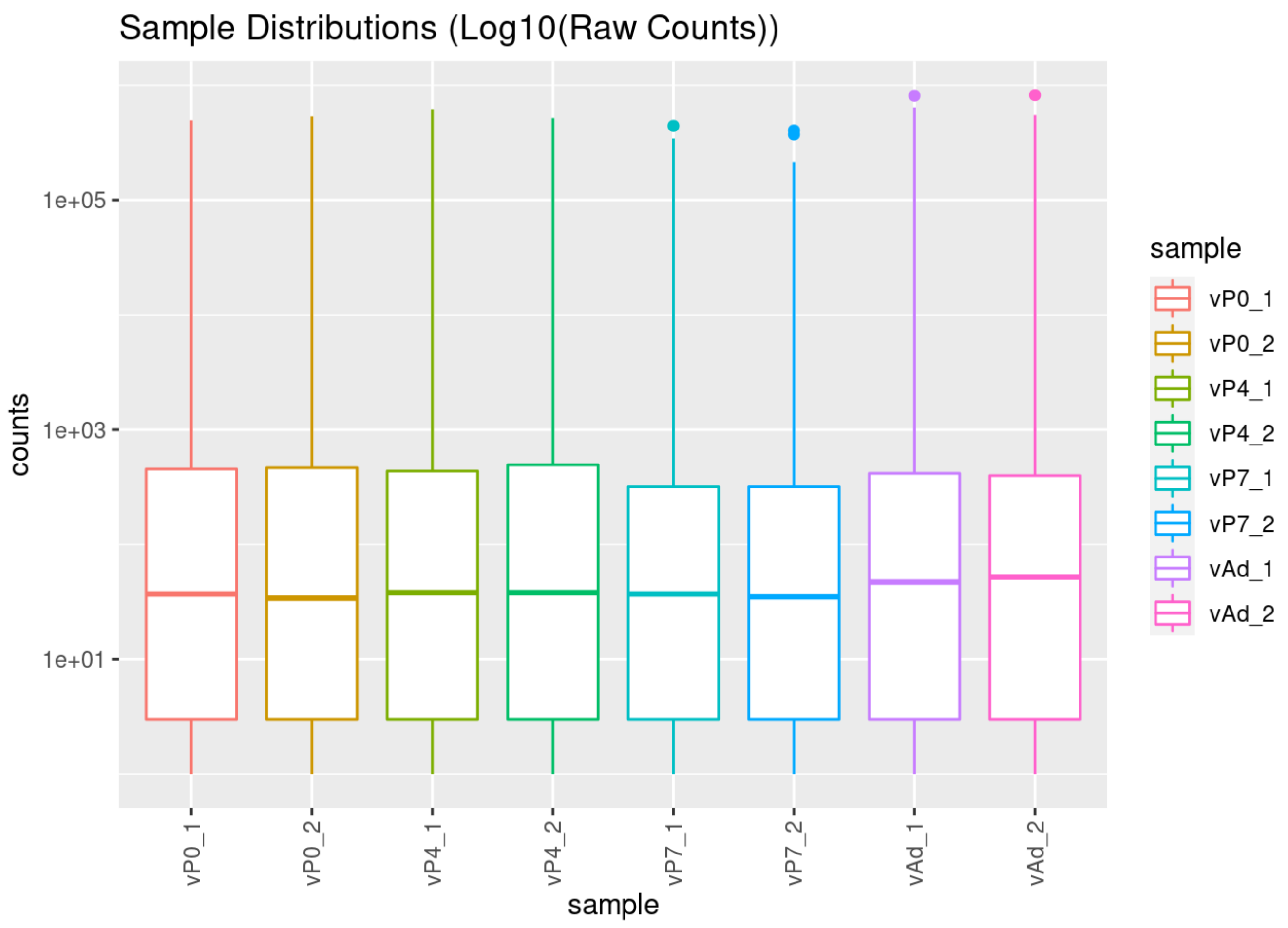
4a. Visualizing count distributions via boxplots

Implement the `plot_sample_distributions()` function using `ggplot2` to create boxplots visualizing the read counts for each sample over all genes. Ensure the function is able to scale the y-axis to $log10$ values if specified by the `scale_y_axis` parameter. Color each boxplot by its respective sample name.

```
#Create and display your count distributions boxplot

plot_sample_distributions(no_header, scale_y_axis = TRUE, title = "Sample Distributions
 (Log10(Raw Counts))")
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

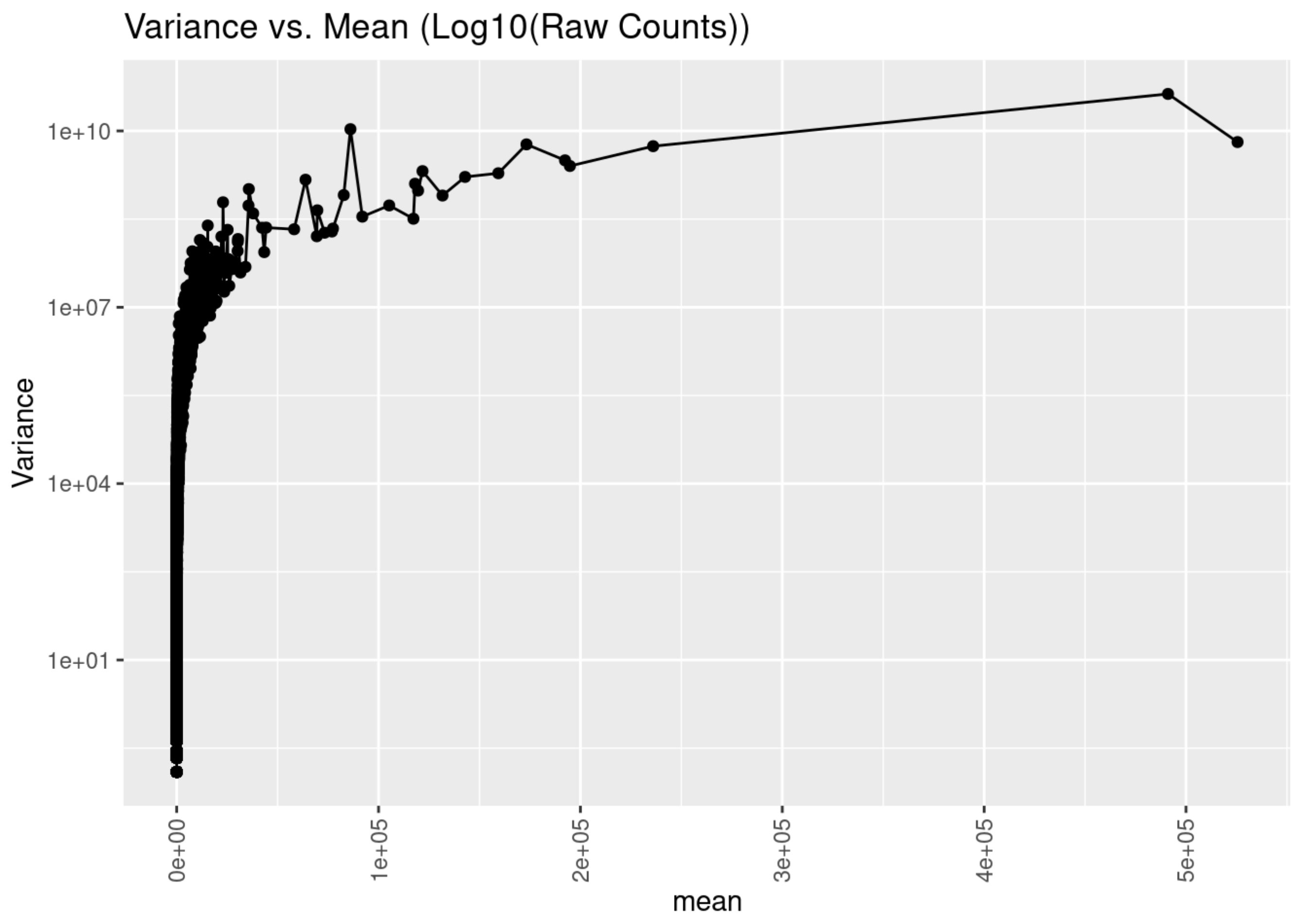
```
## Warning: Removed 68152 rows containing non-finite values (stat_boxplot).
```



4b. Count variance vs mean count

Another aspect of RNAseq data that often needs to be accounted for the is relationship between variance and mean expression: that is, more highly expressed genes will exhibit higher variance values. This is problematic as interesting patterns controlled by genes at naturally lower expression values will be washed out by more highly expressed genes. To visualize this phenomenon, implement the `plot_variance_vs_mean()` function to plot gene variance on the y-axis vs mean count rank on the x-axis, where with p genes, the most highly expressed gene will have rank p and the most lowly expressed gene will have rank 1.

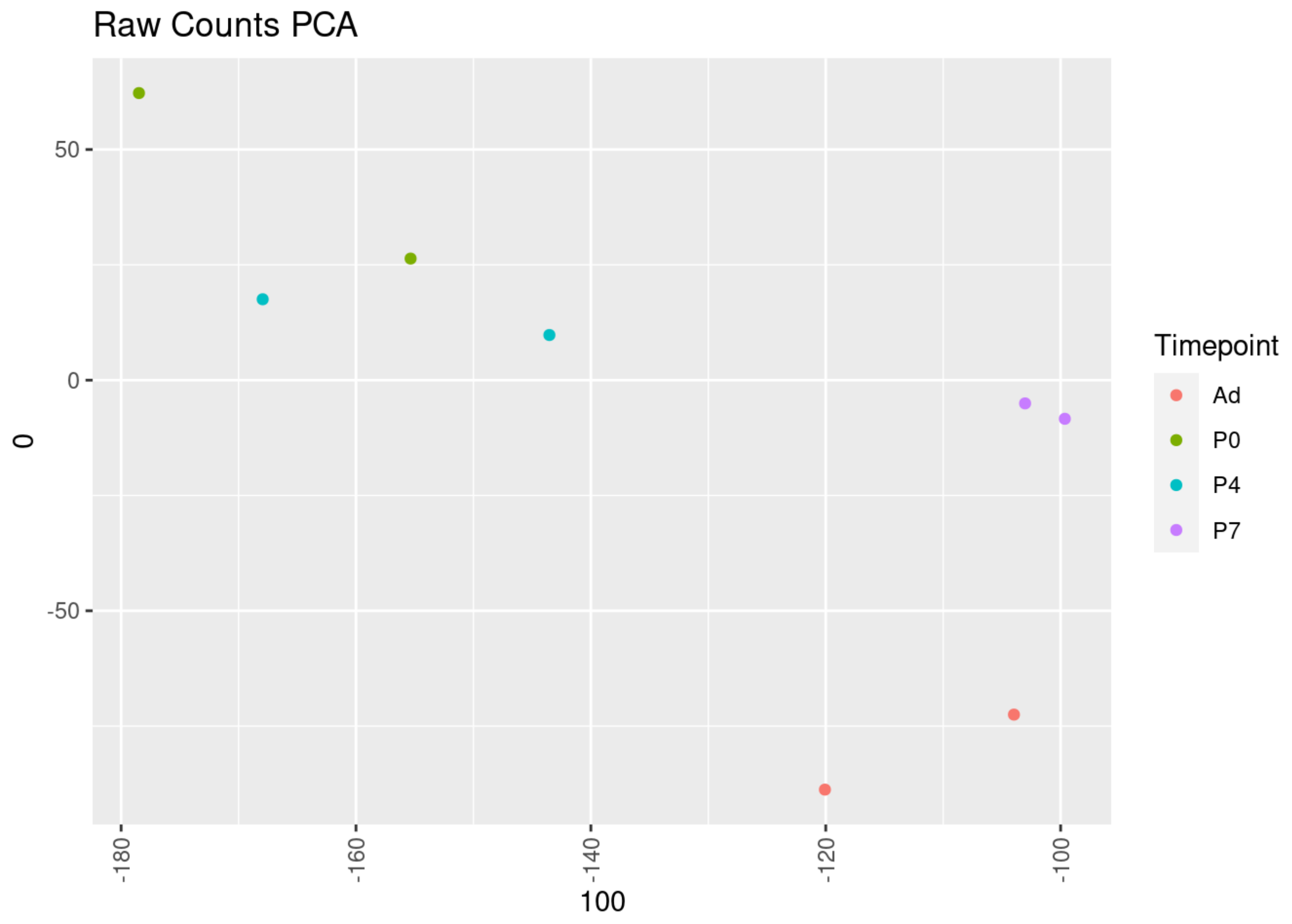
```
#Create and display your variance vs mean plot
plot_variance_vs_mean(no_header, TRUE, "Variance vs. Mean (Log10(Raw Counts))")
```



4c. Visualizing patterns with PCA

Principal Component Analysis (PCA) is a dimension reduction technique often used to extract the most meaningful structure in a dataset to fewer dimensions. This is incredibly useful when trying to visualize your data in only a few dimensions (e.g. 2 PCs compared to 2000 genes), and when trying to build models with only a few orthogonal data features. To visualize the structure in our dataset, implement the `plot_pca()` function to both run PCA on a provided count matrix, but then to also plot the first two PCs of the dataset using a scatterplot, where each dot represents a sample and is colored by its respective sample name.

```
#Create and display your PCA plot
plot_pca(no_header, sample_meta, 'Raw Counts PCA')
```



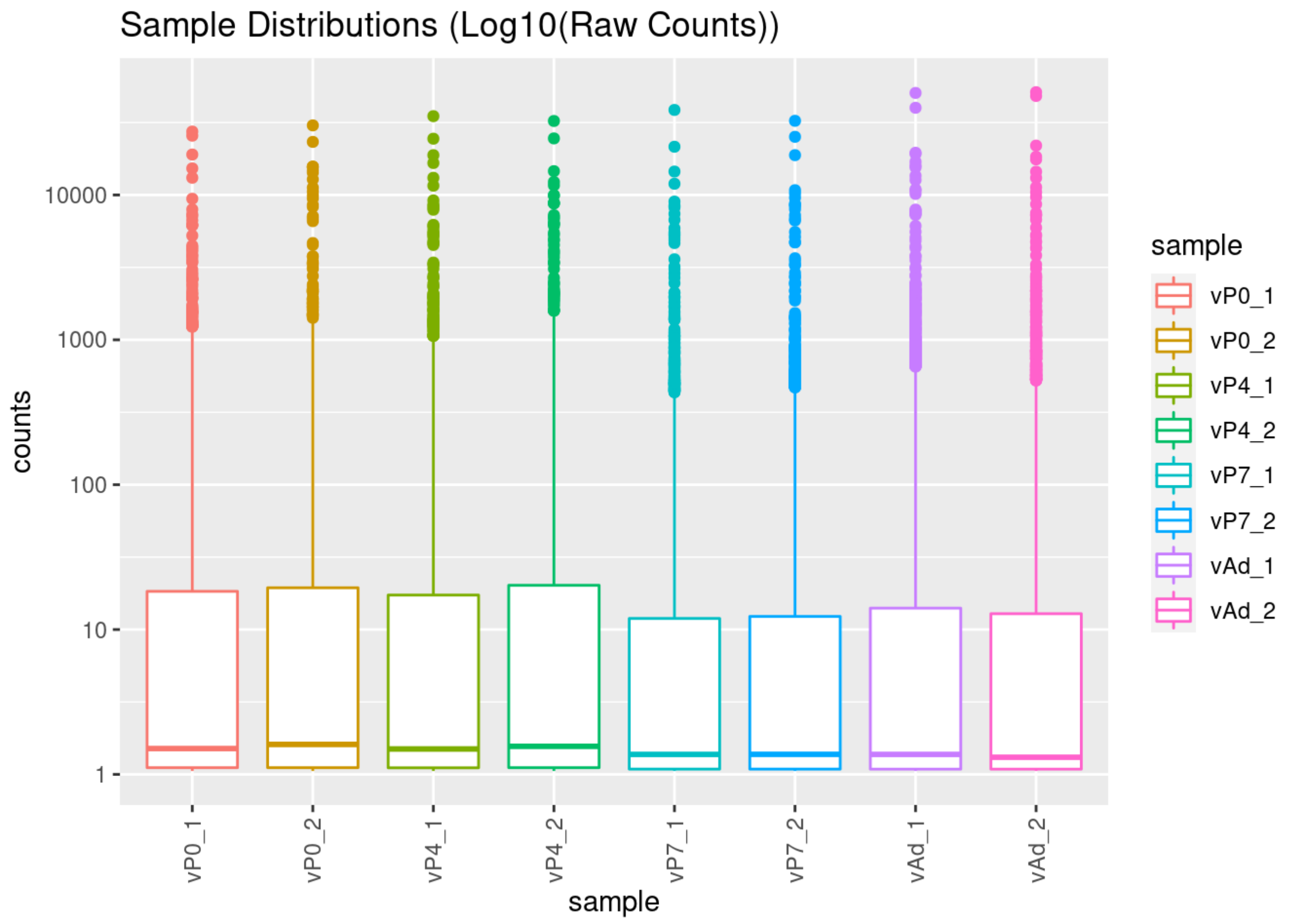
5. Visualizing CPM

Visualize the effect of CPM normalization on the dataset by visualizing the sample distributions, the relationship between variance and average expression values, and plotting the samples along the first two PCs. Does the normalization seem effective? What, if any, are the major differences observed between the plots produced by raw counts?

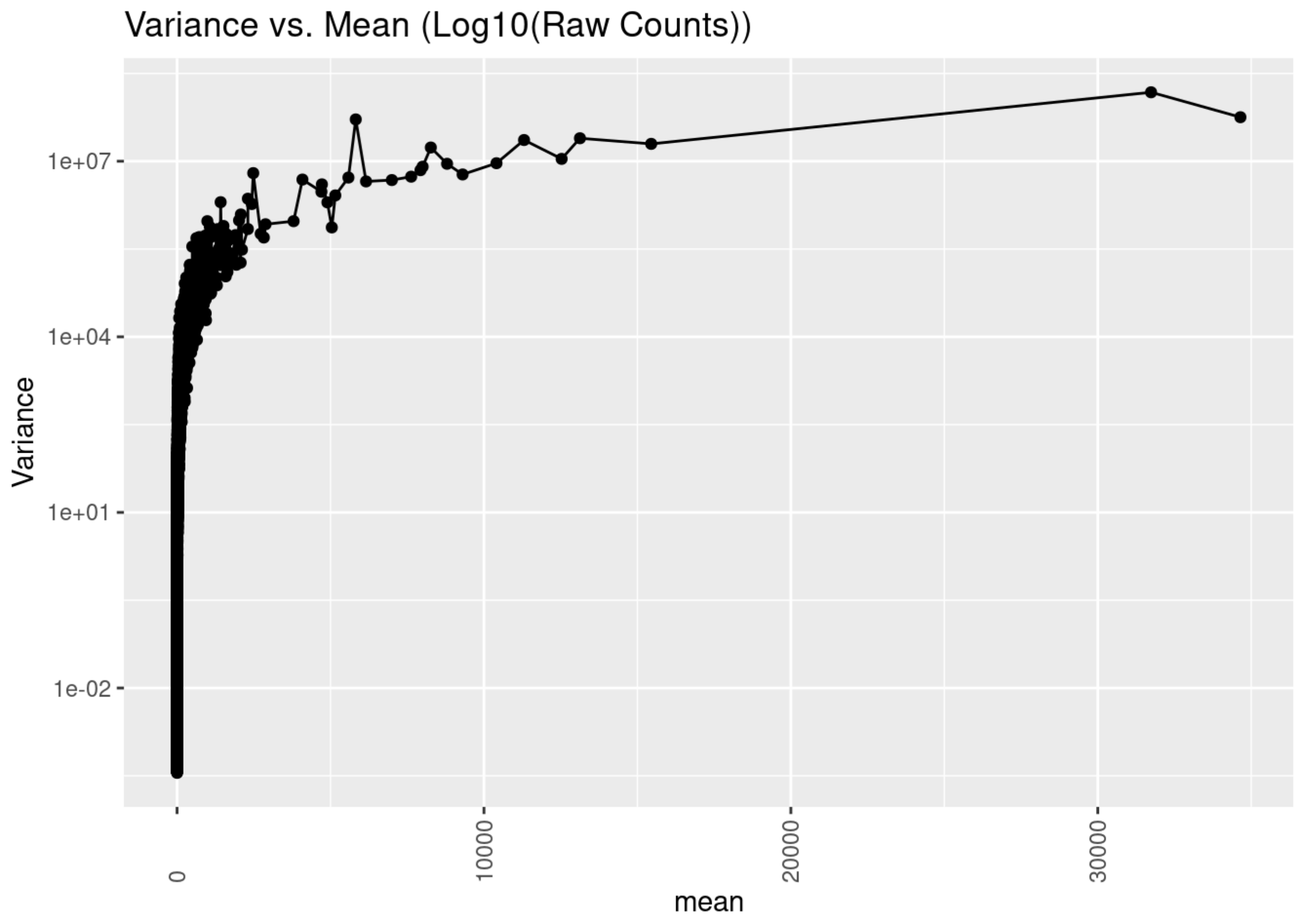
CPM plots

Plot your CPM data using the same three functions you used in part 4 and display your Sample Distribution, Variance vs Mean, and PCA plots. Direct the function to `scale_y_axis` when calling functions that have that parameter.

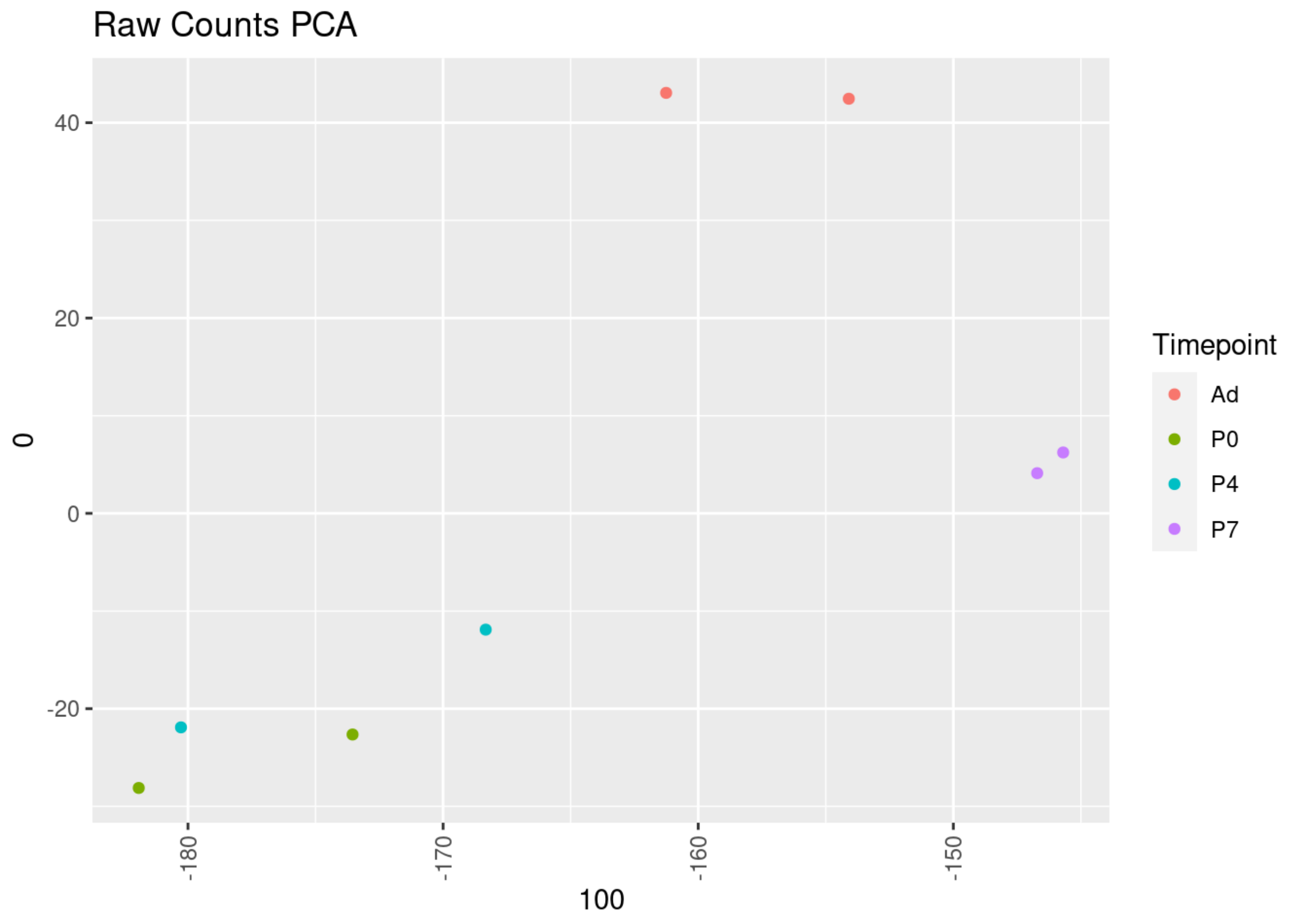
```
#Create and display 3 plots of your cpm matrix: Sample Distribution, Variance vs Mean, a
nd PCA
plot_cpm <- cpm[c(-1)]
plot_sample_distributions(plot_cpm, TRUE, "Sample Distribution (log10(CPM))")
```



```
plot_variance_vs_mean(plot_cpm, TRUE, "Variance vs Mean (DESeq2)")
```



```
plot_pca(plot_cpm, sample_meta, "CPM PCA")
```



6. Visualizing DESeq2 Normalizion

Visualize the effect of DESeq2 normalization on the dataset by visualizing the sample distributions, the relationship between variance and average expression values, and plotting the samples along the first two PCs. How does DESeq2 normalization compare to CPM? Raw values? Which method, if any, seems most effective?

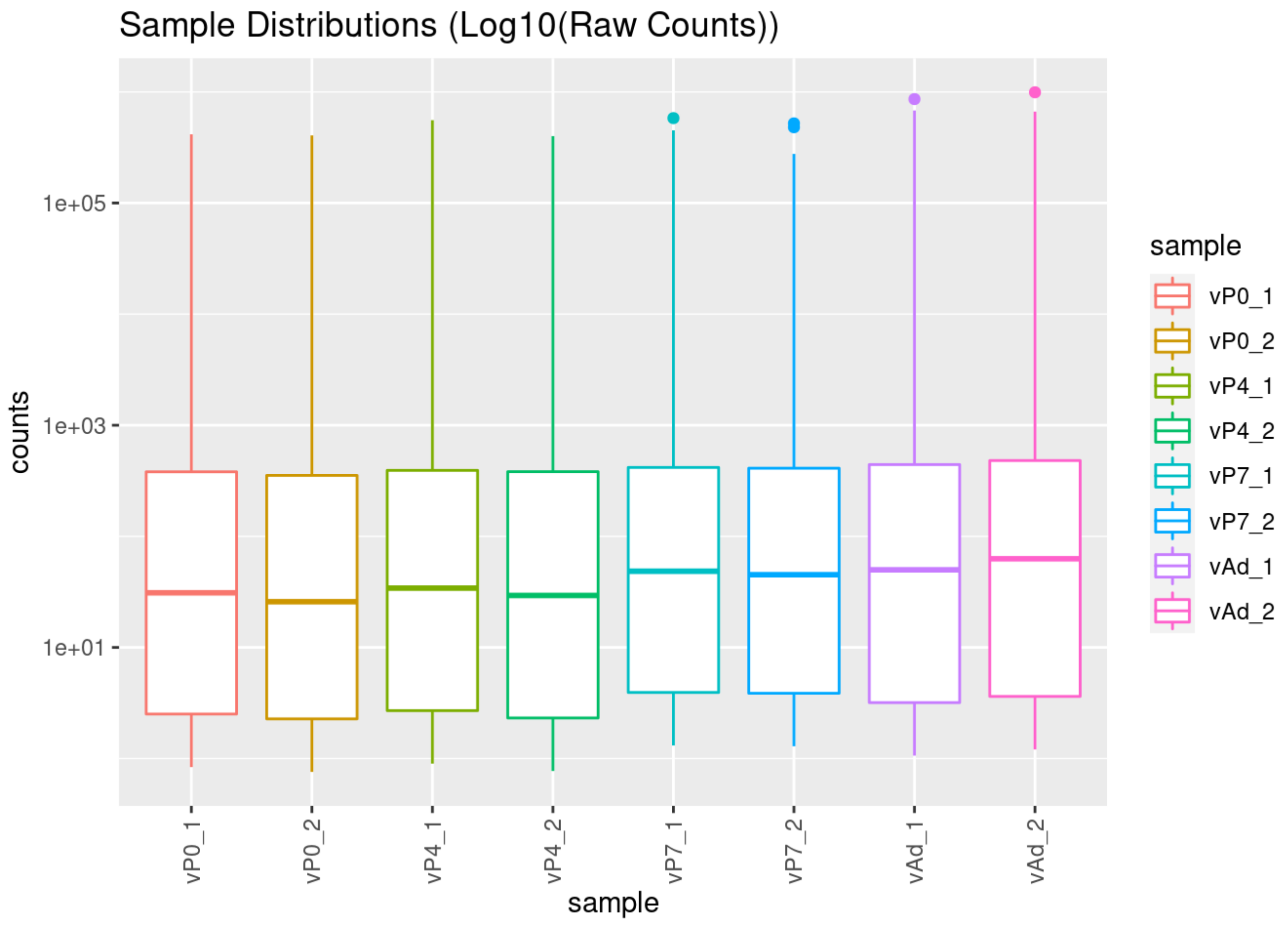
DESeq2 plots

Plot your DESeq2 data using the same three functions you used in part 4 and display your Sample Distribution, Variance vs Mean, and PCA plots.

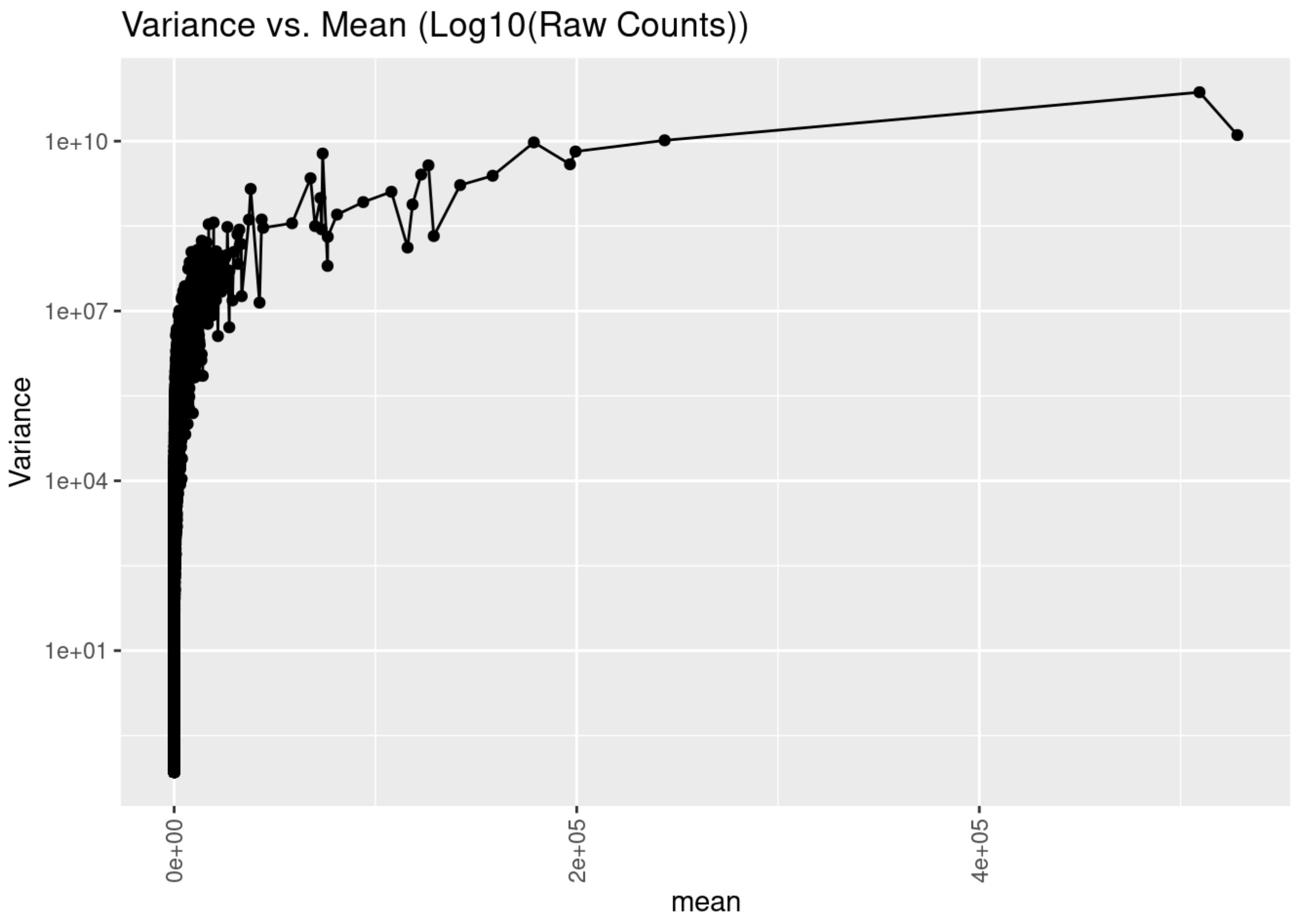
```
#Create and display 3 plots of your deseq-normalized data: Sample Distribution, Variance
vs Mean, and PCA

plot_dseqnormalized <- dseqnormalized[c(-1)]
plot_sample_distributions(plot_dseqnormalized, TRUE, "Sample Distribution (log10(DESeq2
 norm counts" )
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

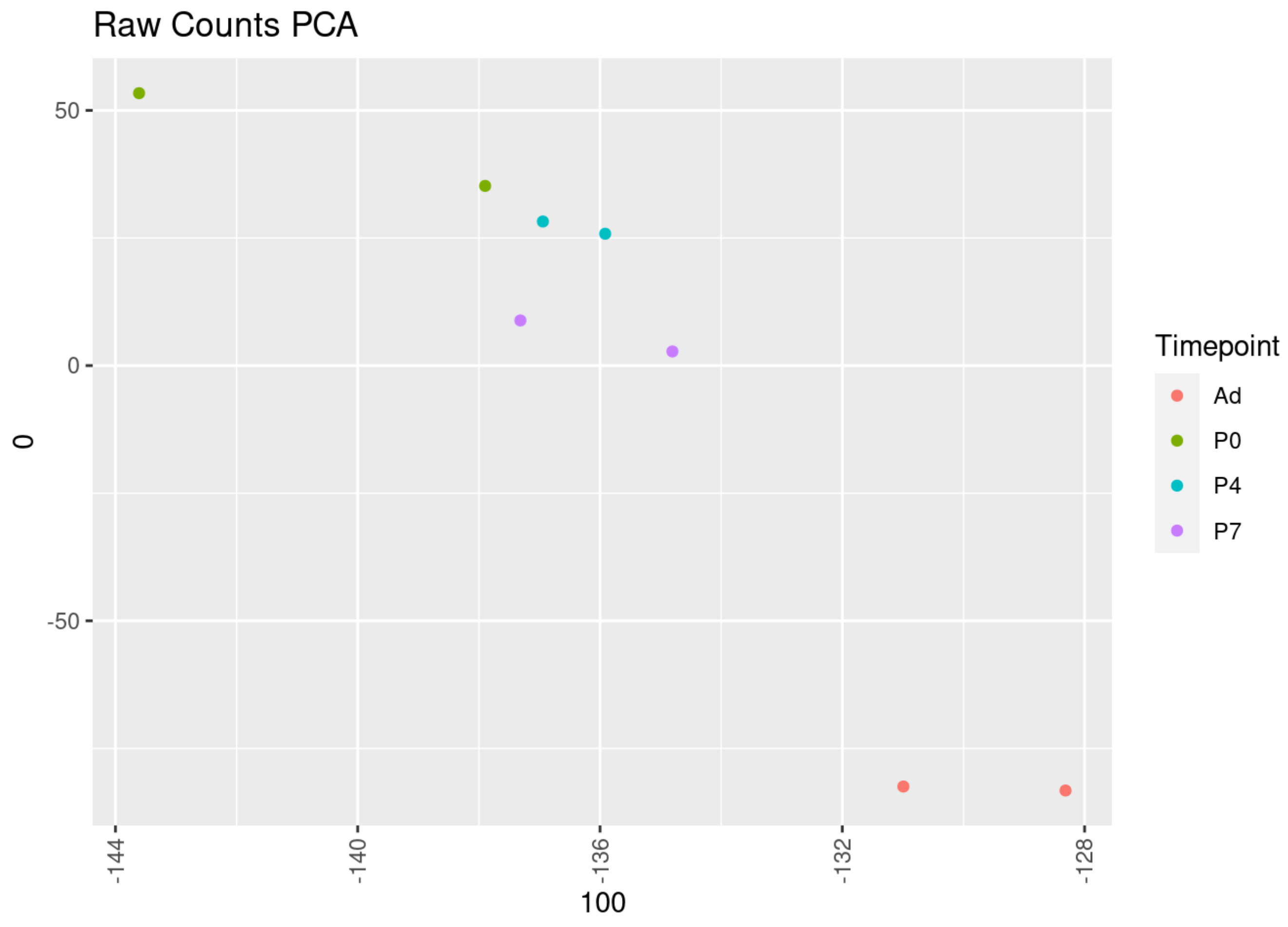
```
## Warning: Removed 68152 rows containing non-finite values (stat_boxplot).
```



```
plot_variance_vs_mean(plot_dseqnormalized, TRUE, "Variance vs Mean (CPM)" )
```



```
plot_pca(plot_dseqnormalized, sample_meta, "DESeq2 Normalized PCA" )
```


Raw Counts PCA
50
0
-50
0
-144
-140
-136
-132
-128
100
Timepoint
Ad
P0
P4
P7