Data cleanup and summary statistics with R

Jasleen Grewal

2019-02-11

Contents

Getting started

This tutorial assumes you know how to load data into an RStudio session, view a dataframe and explore columns/rows of a dataframe in R. Knowing how to visualize data as scatterplots will also be helpful, though not essential.

We will be using a package called ggpubr today.
> Ggpubr is a graphing package, that lets you create publication ready ggplots, and automatically add significance levels to your figures.

Let us also load up ggplot2 into our current environment, just in case we want to make pretty plots.

```
# If you don't have ggplot2 installed already install.packages('ggplot2')
# Otherwise, load it into the environment
require(ggplot2)
library(ggpubr)
library(reshape2)
```

R provides excellent support for statistical analysis. The data we will be working with is cell-line expression data from the LINCS1000 dataset. I have adapted this dataset for our use, which is available at the same spot you found this tutorial. You can also download the original data from here.

Cleaning my data

```
## Load the data
data_df <- read.table("data_sp_scaled.txt", sep = "\t", stringsAsFactors = F, header = T)
## and the covariate information
metadata_df <- read.table("metadata.txt", sep = "\t", stringsAsFactors = F, header = T)
```

As always, let us start by figuring out what we are working with. The `dim()` function prints the dimensions of a dataframe, and `head()` function shows the first 6 rows of a dataframe. You can also print only the row-count (or column-count) with the commands `nrow()` and `ncol()`.
The command `dim(data_df)` tells us that the data has 35 rows and 33842 columns. Similarly, the row count and column count values for the metadata dataframe, `metadata_df`, are 35, 5, respectively.

Notice that if we try to print the first 6 rows of `data_df`, the output is immense. Thus, instead of `head(data_df)`, we will print the first 6 rows, with the first 10 columns. We can select these columns with `data_df[,1:10]`.

	TSPAN6	TNMD	DPM1	SCYL3	C1orf112	FGR	CFH	FUCA2	GCLC	NF
HCC1806	3.505880	0	NA	NA	3.534551	NA	2.8251175	5.727614	NA	4.600
MCF10A	4.027427	0	5.549405	2.482252	3.963324	0.0000000	5.2812262	5.292880	6.521847	4.172
SKBR3	2.684550	0	6.747064	3.199656	4.189142	0.1014213	0.0096204	5.240616	4.322539	3.518
HS578T	NA	0	NA	1.424340	3.363768	0.0369093	5.2759113	5.841921	3.851815	4.461
MDAMB231	4.287758	NA	5.693864	2.217807	4.619957	0.0000000	0.9724045	5.677292	4.046207	5.442
BT20	3.335776	0	6.602087	2.648148	3.904788	0.0000000	0.1888233	4.781958	5.623793	4.138

The column names correspond to genes, and the rows represent samples. These sample names correspond to the column `cl_id` in `metadata_df` (can you quickly verify this using `head`?).
More interestingly, we have some missing values in our data! Before we try and figure out a fix for this, let us calculate how many genes have missing values, or if the problem is only in a single sample.
R has a handy command, `complete.cases`, for checking if there are any rows containing missing values. It returns a TRUE/FALSE value for every row. We can summarize the results of this **list** in tabular form, using the function `table()`.

```
table(complete.cases(data_df))
```

Var1	Freq
FALSE	5
TRUE	30

It appears 5 samples have atleast 1 gene with a missing value. We can redo this test for the genes, after **transposing** our data. This is done using the function `t()`.

```
table(complete.cases(t(data_df)))
```

Var1	Freq
FALSE	18941
TRUE	14901

Dealing with missing data

Over 50% of the genes across 5 samples are missing. We can deal with this either using imputation strategies, or by discarding the problematic samples. As imputation strategies are an entire discussion by themselves, we will into dive into them today (additional resources available at end of tutorial). Instead, we will take the easy way out and remove the samples with NAs. Good thing we have already covered a quick way to unselect these samples!.

```
data_df_clean <- data_df[complete.cases(data_df), ]
```

After this, we have `nrow(data_df_clean)` samples and `ncol(data_df_clean)` genes. We can do a quick 'smell-test' on this data, by using the dataframe function `summary()`. This function calculates summary statistics for each column in the dataframe. We can transpose the dataframe so that the samples become columns (instead of rows).

```
summary(t(data_df_clean))
```

BT20	MCF7	PDX1258	PDX1328	BT549	HCC38	HCC
Min. : 0.0000	Min. : 0.0000	Min. : 0.0000	Min. : 1.000	Min. : 0.0000	Min. : 0.0000	Min. :
1st Qu.: 0.0000	1st Qu.: 0.0000	1st Qu.: 0.0000	1st Qu.: 1.000	1st Qu.: 0.0000	1st Qu.: 0.0000	1st Qu.:
Median : 0.1565	Median : 0.2083	Median : 0.3089	Median : 1.123	Median : 0.1346	Median : 0.2004	Median :
Mean : 1.6150	Mean : 1.6330	Mean : 1.7137	Mean : 3.433	Mean : 1.5792	Mean : 1.6433	Mean :
3rd Qu.: 3.2142	3rd Qu.: 3.2164	3rd Qu.: 3.2897	3rd Qu.: 3.648	3rd Qu.: 3.0660	3rd Qu.: 3.1915	3rd Qu.:
Max. :13.9749	Max. :13.5385	Max. :400.7026	Max. :310.614	Max. :12.7921	Max. :12.9747	Max. :1

Well, its still hard to read! Enter ggplot! However, we need to set up our data such that we can pass in a column with the sample name, and a column with the values being plotted.

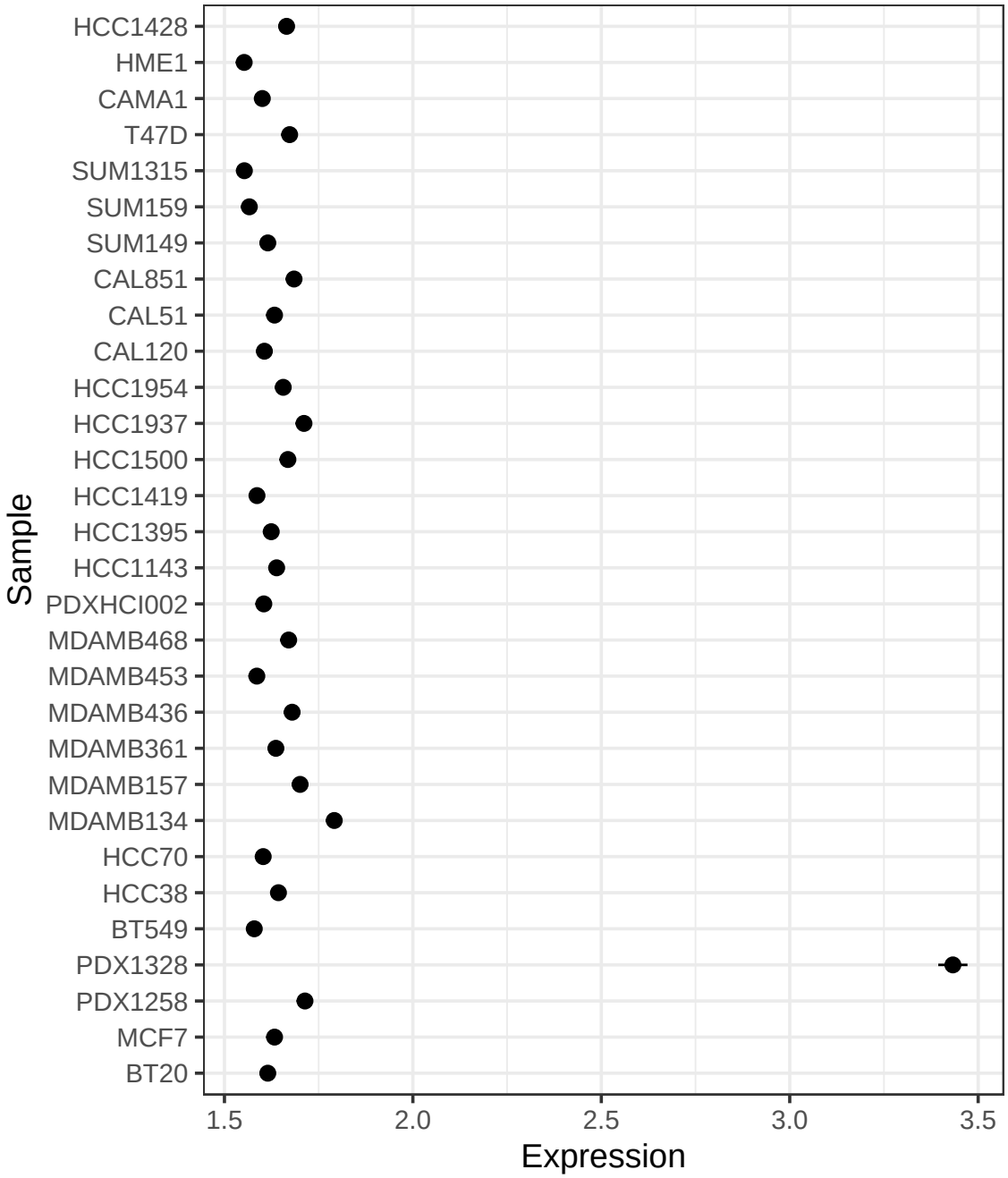
For this, we will use the `melt` function from the **reshape2** package. The melt function is helpful in converting your data from the *long* to *wide* format. A similar function, `cast()`, can be used when you wish to calculate summary statistics on your data.

```
data_compact_df = melt(t(data_df_clean))
colnames(data_compact_df) = c("Gene", "Sample", "Expression")
```

Dealing with outliers

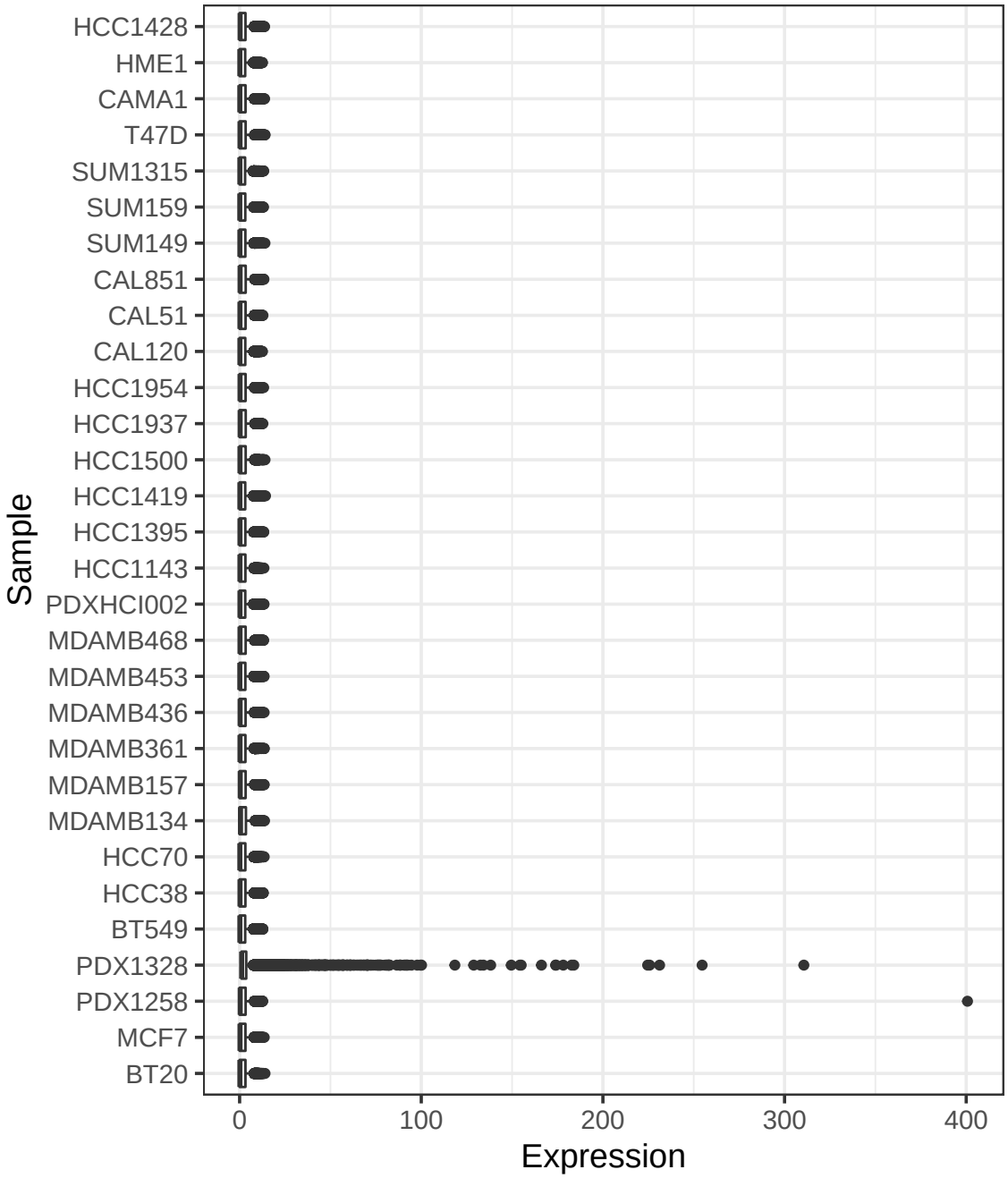
ggplot has a handy geom_object (remember these from the ggplot tutorial?). The `stat_summary()` method allows us to plot a pointrange plot showing the mean and standard error.

```
## stat_summary() defaults to the categories defined on the x axis, and summarizes
## the numeric spread on the y-axis We use coord_flip() to switch the categories
## to the y-axis, to improve readability
ggplot(data_compact_df, aes(x = Sample, y = Expression)) + stat_summary() + theme_bw(base_size = 14) +
    coord_flip()
```



However, we have an anomalous sample! The sample `PDX1328` has readings that lie extremely outside the range of the rest of the samples. We can see this more clearly with a boxplot version of the datapoints, plotted using `geom_boxplot()`

```
ggplot(data_compact_df, aes(x = Sample, y = Expression)) + geom_boxplot() + theme_bw(base_size = 14) +
    coord_flip()
```

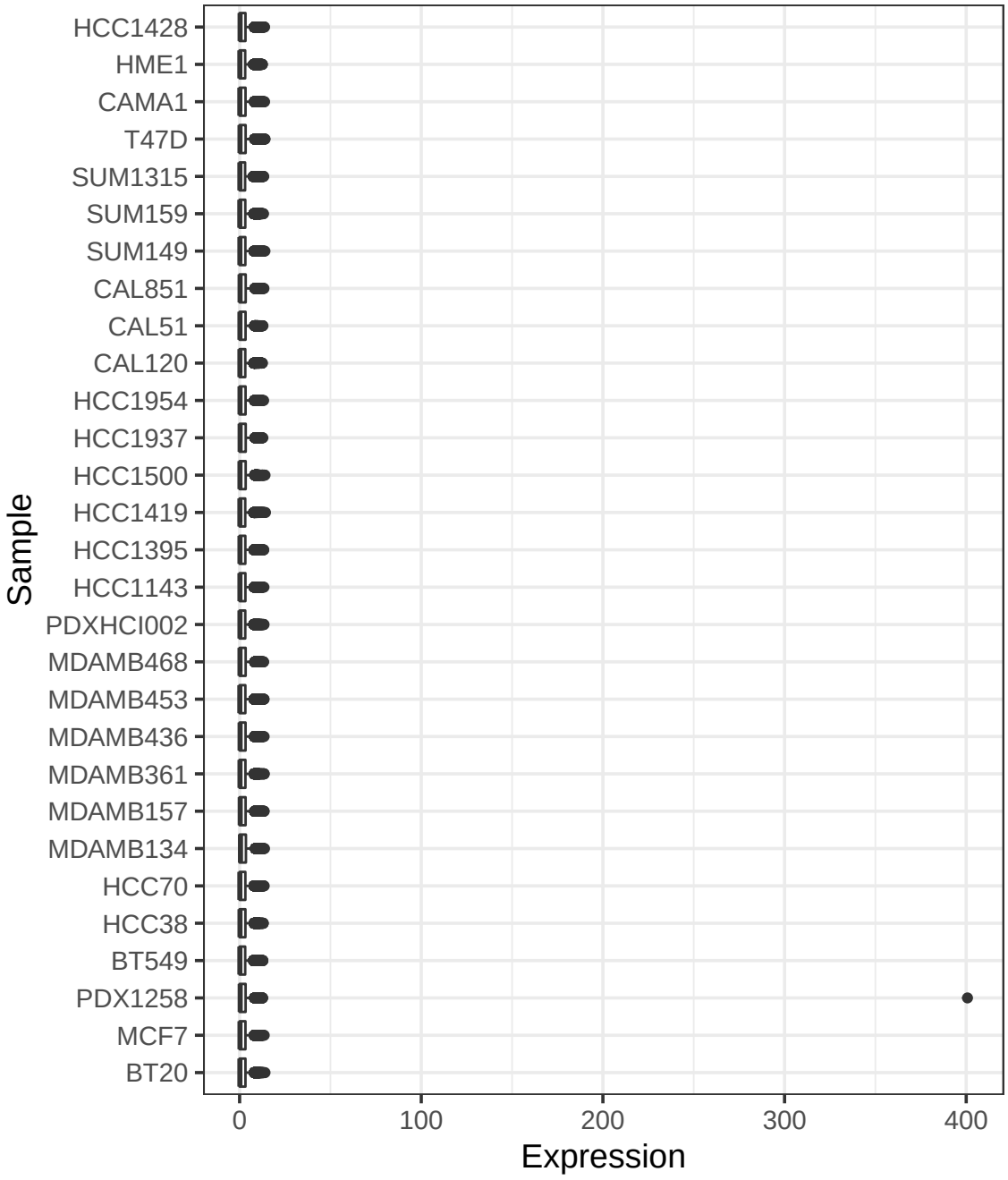


We can remove the outlier sample using the following command (note that we are making changes to the samplxe x gene dataframe, not the melted version).

```
data_df_clean2 = data_df_clean[!(rownames(data_df_clean) %in% c("PDX1328")), ]
```

Alright, so what does the data look like after that?

```
# First we melt our data
data_compact_df = melt(t(data_df_clean2))
colnames(data_compact_df) = c("Gene", "Sample", "Expression")
# Then we plot it
ggplot(data_compact_df, aes(x = Sample, y = Expression)) + geom_boxplot() + theme_bw(base_size = 14) +
    coord_flip()
```



It looks like PDX1258 has an outlier gene. We can easily print out the gene from our melted dataframe, with the command `data_compact_df[data_compact_df$Expression > 300, "Gene"]`. This returns TSPAN6. We can either remove this gene entirely, or replace it with the mean value. Sample metadata information can come in handy at this point. Let us see what information the metadata dataframe can provide:

```
kable(head(metadata_df))
```

cl_id	cl_provider_name	cl_provider_catalog_id	cl_cell_type	cl_disease_detail
CAL51	Leibniz Institute	ACC-302	epithelial-like	breast carcinoma
MCF7	ATCC	HTB-22	epithelial	breast adenocarcinoma
HME1	ATCC	CRL-4010	epithelial	normal
SKBR3	ATCC	HTB-30	epithelial	breast adenocarcinoma
MDAMB231	ATCC	HTB-26	epithelial	breast adenocarcinoma
BT20	ATCC	HTB-19		breast ductal carcinoma

PDX1258 is a breast carcinoma. We can see the different disease categories by summarizing the contents of the column `cl_disease_detail`.

```
table(metadata_df$cl_disease_detail)
```

Var1	Freq
breast adenocarcinoma	10
breast carcinoma	5
breast ductal carcinoma	12
breast fibrocystic disease	1
breast medullary carcinoma	2
normal	1
squamous cell carcinoma	1
unknown	3

As there are 5 `breast carcinoma`s in this dataset, we can potentially set the value of PDX1258 to the average value of the gene in other breast carcinomas. If we connect our metadata with our expression data, it will be easy to select the gene and samples of interest. For this we use the `merge` function. Merging requires a column that has the same values in the 2 different dataframes we are joining. Note that we can specify the column using `by="shared column"` if the column has the same name in the 2 dataframes.

```
## Merge expression and metadata
data_merged = merge(data_compact_df, metadata_df, by.x = "Sample", by.y = "cl_id")
## Select PDX1258's outlier gene
brca_tspan_df = data_merged[data_merged$Gene == "TSPAN6" & data_merged$cl_disease_detail ==
    "breast carcinoma", ]
```

	Sample	Gene	Expression	cl_provider_name	cl_provider_catalog_id	cl_ce
101527	CAL51	TSPAN6	5.6334571	Leibniz Institute	ACC-302	epith
679083	MDAMB453	TSPAN6	0.2958427	ATCC	HTB-131	
750048	PDX1258	TSPAN6	400.7026020	Dan Stover (Harvard Medical School)		epith
913735	T47D	TSPAN6	3.9403834	ATCC	HTB-133	

Notice that there are only 4 samples here, and CAL51 is similar to PDX1258 as both are epithelial-like cell lines. We can set the TSPAN6 value for PDX1258 the same as sample CAL51, or the average of the 3 breast carcinomas.

```
## Firstly we update data_df_clean2 Notice how we select the row with the sample
## name, and gene with the gene name
data_df_clean2["PDX1258", "TSPAN6"] = mean(brca_tspan_df[brca_tspan_df$Sample !=
    "PDX1258", "Expression"])
## Then we recalculate the melted version of this dataframe
data_compact_df = melt(t(data_df_clean2))
colnames(data_compact_df) = c("Gene", "Sample", "Expression")
```

Comparing groups (and plotting the significance values)

For the last bit, we will use an in-built dataset, `airquality`. You can laod it into your current environment by typing `data(airquality)`.

```
data(airquality)
```

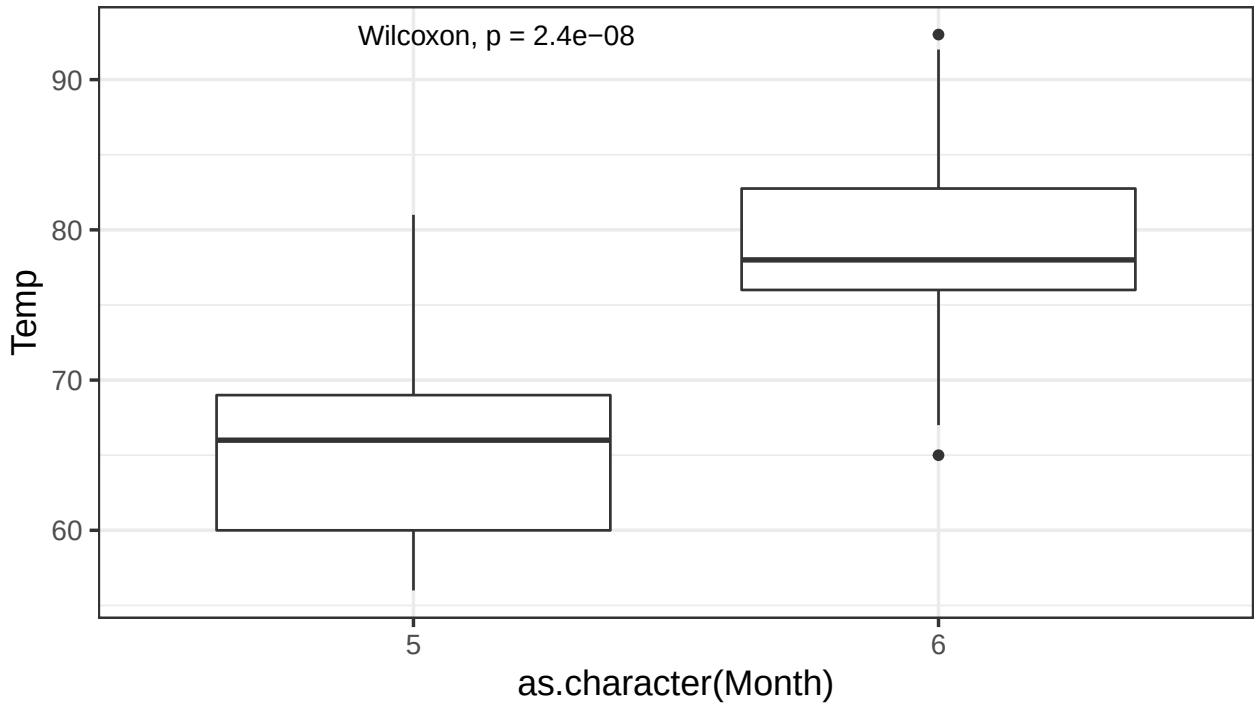
Ozone	Solar.R	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

We will use the package `ggpubr`. This package is quite similar to ggplot, but it has additional methods that make it easy to create publication-ready figures in R. One of these methods is `stat_compare_means()`.

```
ggplot(airquality[airquality$Month %in% c(5, 6), ], aes(x = Month, y = Temp)) + geom_boxplot() +
    stat_compare_means() + theme_bw(base_size = 14)
```

You have probably run into an error message as you run the code above.
Warning message: Continuous x aesthetic – did you forget aes(group=...)? .
This is because the categories we are passing to perform the paired test for significance is numeric (hence 'continuous'). We can overcome this by treating the category column's values (Month) as strings.

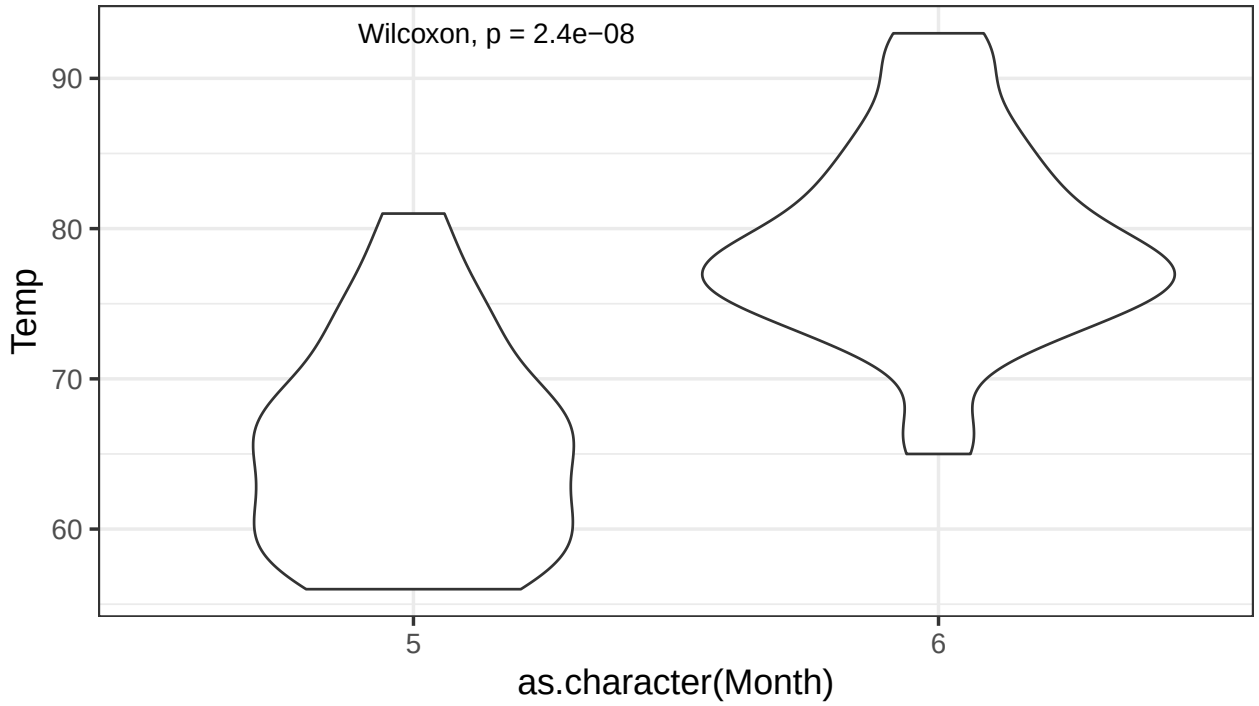
```
ggplot(airquality[airquality$Month %in% c(5, 6), ], aes(x = as.character(Month),
    y = Temp)) + geom_boxplot() + stat_compare_means() + theme_bw(base_size = 14)
```



- You can change the type of test that is performed. For example, try updating `stat_compare_means()` to `stat_compare_means(method="t.test")`

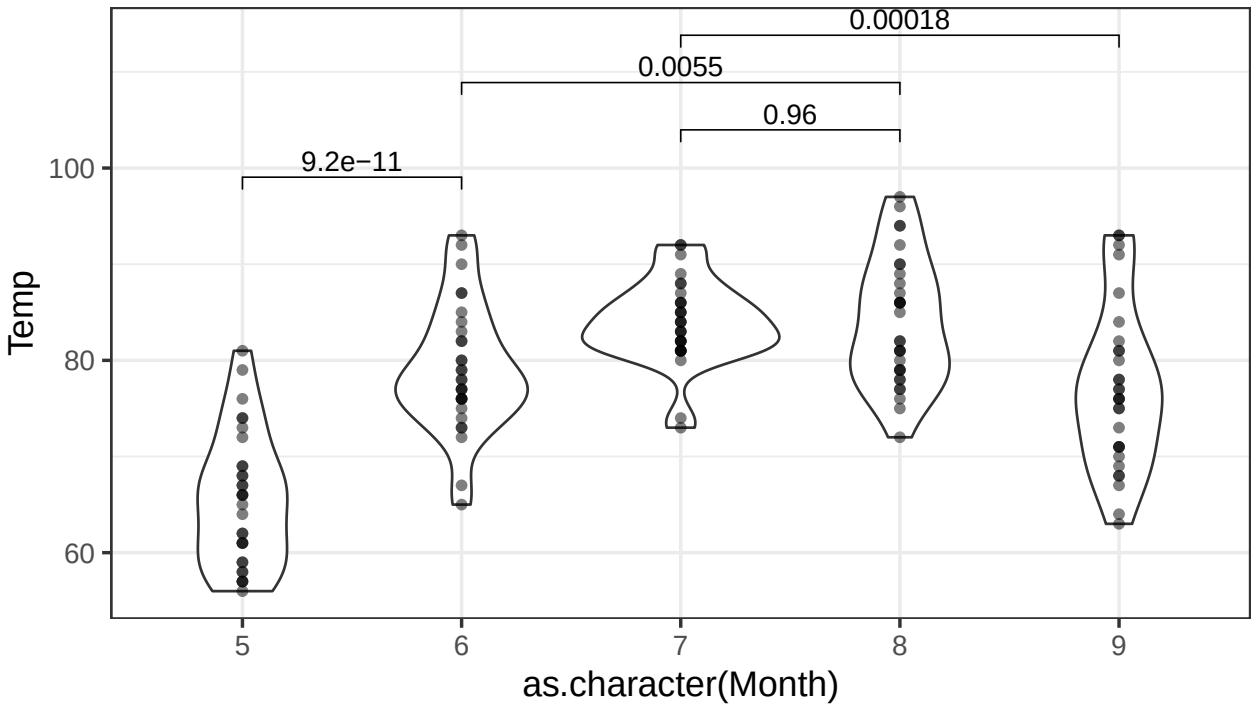
We can visualize the spread of data in the different categories using other geometric objects. A **violin plot** is an extension of a box-plot that shows the kernel density distributions of the data points, in addition to the median value and spread.

```
ggplot(airquality[airquality$Month %in% c(5, 6), ], aes(x = as.character(Month),
    y = Temp)) + geom_violin() + stat_compare_means() + theme_bw(base_size = 14)
```



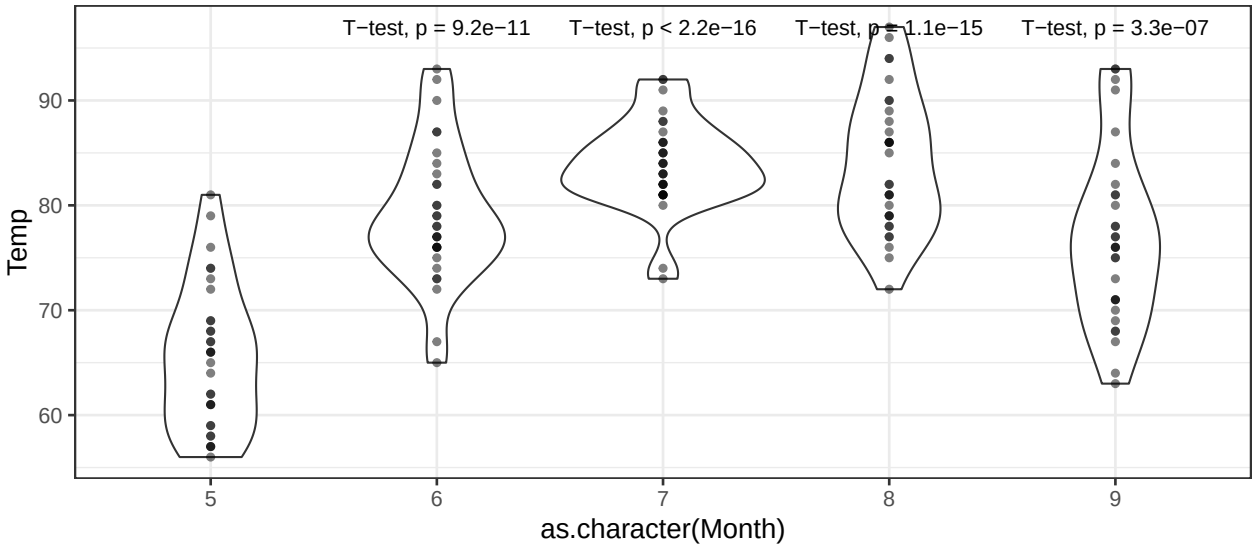
We can also extend the comparison to more than two groups. This, however, requires a bit of work. We first need to define the various pairwise comparisons we wish to perform. Subsequently we pass this list of comparisons to `stat_compare_means`.

```
my_comparisons <- list(c("5", "6"), c("7", "8"), c("6", "8"), c("7", "9"))
## Plot p-values for specified comparisons
ggplot(airquality, aes(x = as.character(Month), y = Temp)) + geom_violin() + geom_point(alpha = 0.5) +
    stat_compare_means(comparisons = my_comparisons, method = "t.test") + theme_bw(base_size = 14)
```



You can also calculate the significance of difference in means between all groups relative to a reference, like so:

```
ggplot(airquality, aes(x = as.character(Month), y = Temp)) + geom_violin() + geom_point(alpha = 0.5) +
    stat_compare_means(method = "t.test", ref.group = "5") + theme_bw(base_size = 14)
```

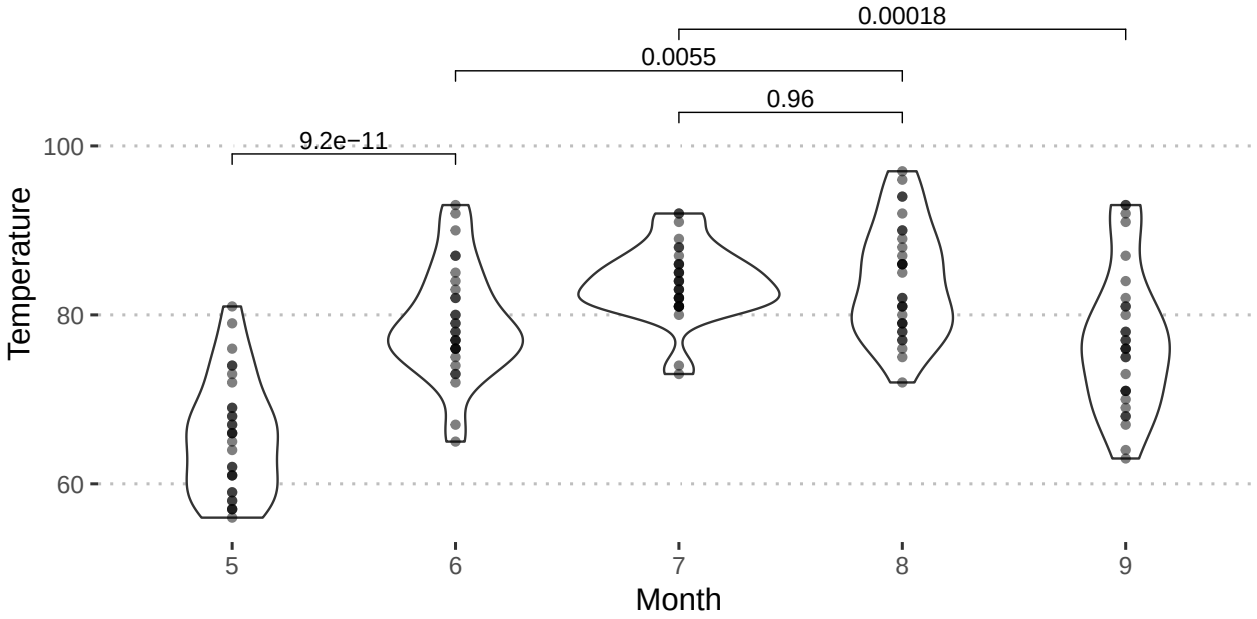


ggpubr magic

ggpubr's methods `theme_pubclean` and `theme_pubr` shift the focus of your plot to your data.

```
my_comparisons <- list(c("5", "6"), c("7", "8"), c("6", "8"), c("7", "9"))

ggplot(airquality, aes(x = as.character(Month), y = Temp)) + geom_violin() + geom_point(alpha = 0.5) +
    stat_compare_means(comparisons = my_comparisons, method = "t.test") + labs(x = "Month",
    y = "Temperature") + theme_pubclean(base_size = 14)
```



Additional Resources

Outlier detection with R
Detailed lecture on data cleanup with R
Using ggpubr to calculate significance

Take-aways

1. Basic smell-tests on your data
2. Removing cases with missing data
3. Identifying outliers
4. Using ggpubr for descriptive statistics
5. Using ggpubr to create publication-ready figures

Extra

Pairs plots for expression data

Given a few samples (observations) with a large number of genes (variables), we can quickly evaluate if certain samples are outliers, simply by comparing the pair-wise scatterplots for all the samples

```
## We can also plot pairwise scatterplots
brca_df = data_merged[data_merged$cl_disease_detail %in% c("normal", "breast medullary carcinoma"),
    ]
## Reverse the melt step
brca_df_recast = dcast(brca_df[, c("Sample", "Gene", "Expression")], Sample ~ Gene)
rownames(brca_df_recast) = brca_df_recast$Sample
brca_df_recast$Sample <- NULL
## Remove the outlier gene
brca_df_recast = brca_df_recast[, !(colnames(brca_df_recast) %in% c("TSPAN6"))]
pairs(t(brca_df_recast), panel = function(...) smoothScatter(..., add = TRUE))
```

Filtering genes by average value or standard deviation

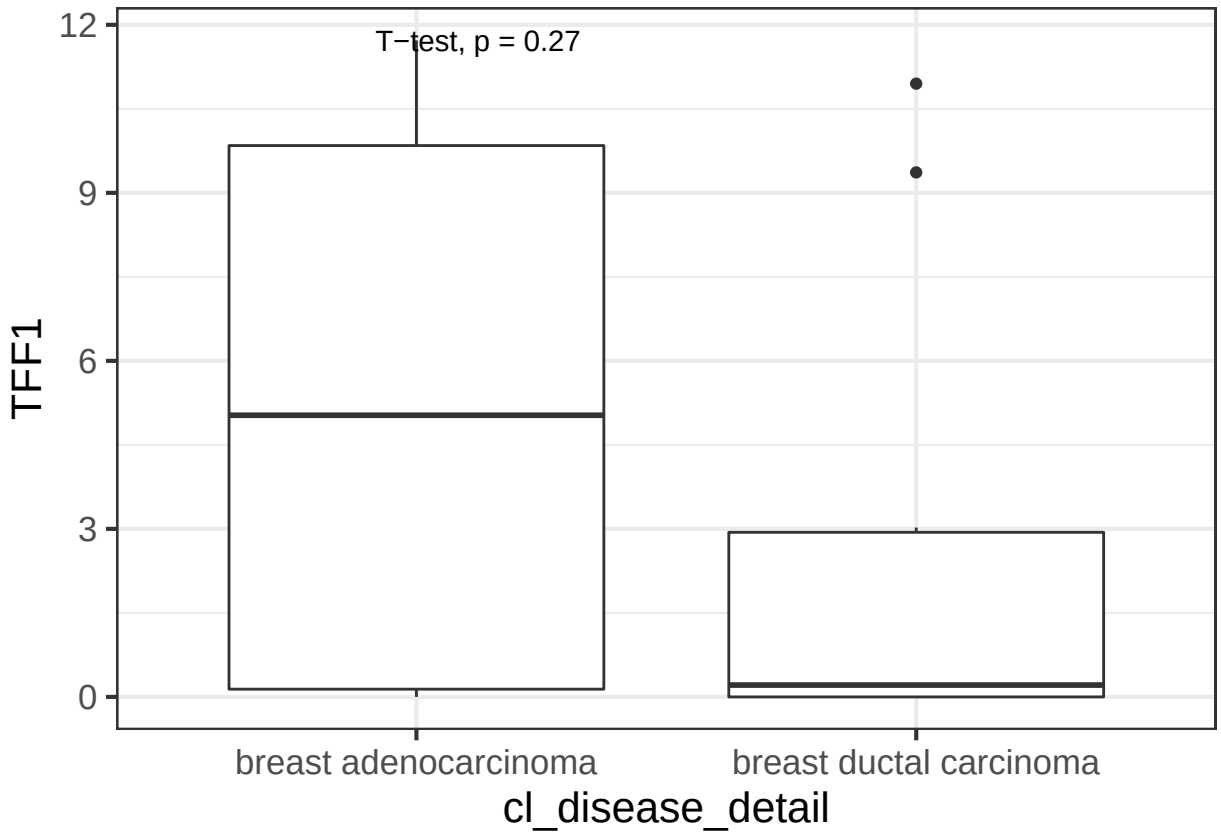
We firstly identify genes that vary within disease types. We will compare breast adenocarcinomas and breast ductal carcinomas.

```
## Select the samples from the metadata dataframe
samples_brca = metadata_df[metadata_df$cl_disease_detail %in% c("breast adenocarcinoma",
    "breast ductal carcinoma"), ]
## Filter our dataframe based on this list
brca_cohorts_df = data_df_clean2[rownames(data_df_clean2) %in% samples_brca$cl_id,
    ]
### Compare this dataframe to what you get with the following command
brca_cohorts_testdf = data_df_clean2[samples_brca$cl_id, ]
```

We will do some filtering to identify the most variable genes. Bioconductor's package genefilter also has some of these pre-implemented.

```
# Calculate mean of each gene
avg_genes = sapply(brca_cohorts_df, mean)
# Calculate standard deviation of each gene
sd_genes = sapply(brca_cohorts_df, sd)
# Filter dataframe based on an SD threshold of your choosing
brca_cohorts_filt = brca_cohorts_df[, names(sd_genes[sd_genes == max(sd_genes)]),
    drop = FALSE]
brca_cohorts_filt$cl_id = rownames(brca_cohorts_filt)
brca_cohorts_filt = merge(brca_cohorts_filt, metadata_df)

ggplot(brca_cohorts_filt, aes(x = cl_disease_detail, y = TFF1)) + geom_boxplot() +
    theme_bw(base_size = 16) + stat_compare_means(method = "t.test")
```


T−test, p = 0.27
12
9
6
3
0
TFF1
breast adenocarcinoma
breast ductal carcinoma
cl_disease_detail