Building Functions

Eugen Buehler

November 14th, 2018

R Functions are objects

R is a functional programming language. This means that functions are “objects”, just like data frames, vectors, and other things that are assigned to variables and passed to other functions.

A rose by any other name

The name of a functions is actually the name of a variable that contains the function, in the same way that the

```
log
```

```
## function (x, base = exp(1))  .Primitive("log")
```

This means that we can create a copy of a function by assigning its value to a new variable.

```
myLogFunction <- log
myLogFunction
```

```
## function (x, base = exp(1))  .Primitive("log")
```

Functions are a kind of data and have a class

```
myNumber <- 7
class(myNumber)
```

```
## [1] "numeric"
```

```
class(log)
```

```
## [1] "function"
```

Creating a function

We can create a new function using the word “function” followed by the functions arguments and one or more R statements.

```
myDumbFunction <- function() 42
myDumbFunction()
```

```
## [1] 42
```

This is a function with **no** arguments. Usually functions have arguments, which we will see next. Here, `myDumbFunction` gives the same answer whenever it’s called

Creating a multi-statement function

If there is more than one statement in a function, they should be enclosed in curly brackets:

```
doubleIt <- function(x) {
  myResult <- x * 2
  myResult # or, explicitly, return(myResult)
}
doubleIt(5)
```

```
## [1] 10
```

The last statement within the curly brackets will be the value returned by the function.

x is the function argument, in that it is a placeholder we can replace with an actual value when calling the function

Functions live in their own little world

Inside a function, variables that existed in your environment can be used and even changed. However, any changes made, including changing data stored in variables and creating new variables, happens solely within the function. Your environment stays the same.

```
exists("myResult")
```

```
## [1] FALSE
```

```
myResult <- 1000
doubleItOutput <- doubleIt(2)
myResult
```

```
## [1] 1000
```

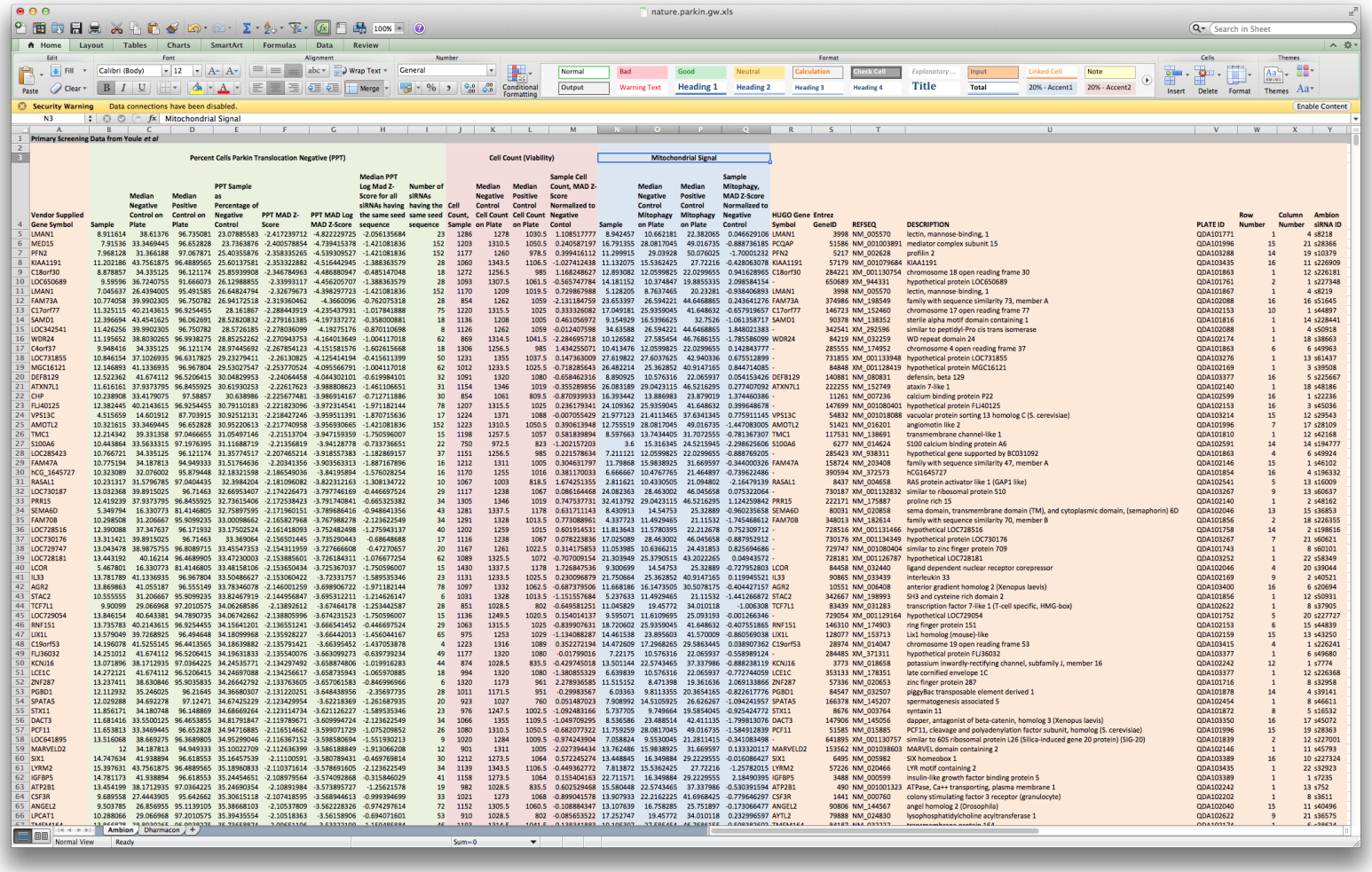
Example Data Set

The data set used in today's lecture comes from an siRNA screen that we published a few years ago. The screen looked for genes that influence parkin translocation.

High-content genome-wide RNAi screens identify regulators of parkin upstream of mitophagy. Hasson SA, Kane LA, Yamano K, Huang CH, Sliter DA, Buehler E, Wang C, Heman-Ackah SM, Hessa T, Guha R, Martin SE, Youle RJ. Nature. 2013.

The data set will be available for download from the lectures portion of the class web page, also here.

Preview the Data



Import the data

```
library(readxl)
ambion <- read_excel("lecture_functions_data/nature.parkin.gw.xlsx", skip = 3)
str(ambion)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    65196 obs. of  25 variables:
##  $ Vendor Supplied Gene Symbol                                          : chr
##  $ Sample                                                               : num
##  $ Median Negative Control on Plate                                     : num
##  $ Median Positive Control on Plate                                     : num
##  $ PPT Sample as Percentage of Negative Control                         : num
##  $ PPT MAD Z-Score                                                      : num
##  $ PPT MAD Log MAD Z-Score                                              : num
##  $ Median PPT Log Mad Z-Score for all siRNAs having the same seed sequence: num
##  $ Number of siRNAs having the same seed sequence                       : num
##  $ Cell Count, Sample                                                   : num
##  $ Median Negative Control Cell Count on Plate                          : num
##  $ Median Positive Control Cell Count on Plate                          : num
##  $ Sample Cell Count, MAD Z-Score Normalized to Negative Contol         : num
##  $ Sample__1                                                            : num
##  $ Median Negative Control Mitophagy on Plate                           : num
```

Check for missing data

```
options(dplyr.width = Inf) # show all cols
ambion %>% summarize_all(function(x) sum(is.na(x)))
```

```
## # A tibble: 1 x 25
##   `Vendor Supplied Gene Symbol` Sample `Median Negative Control on Plate`
##                           <int>  <int>                                <int>
## 1                           441    441                                  441
##   `Median Positive Control on Plate`
##                                <int>
## 1                                441
##   `PPT Sample as Percentage of Negative Control` `PPT MAD Z-Score`
##                                            <int>             <int>
## 1                                            441               441
##   `PPT MAD Log MAD Z-Score`
##                       <int>
## 1                       441
##   `Median PPT Log Mad Z-Score for all siRNAs having the same seed sequence`
##                                                                      <int>
## 1                                                                      441
##   `Number of siRNAs having the same seed sequence` `Cell Count, Sample`
```

Investigate Missing Data

```
#ambion[is.na(ambion[,1]),][1,]
ambion %>% filter(is.na(.[,1])) %>% slice(1)
```

```
## # A tibble: 1 x 25
##   `Vendor Supplied Gene Symbol` Sample `Median Negative Control on Plate`
##   <chr>                          <dbl>                                <dbl>
## 1 <NA>                              NA                                   NA
##   `Median Positive Control on Plate`
##                                <dbl>
## 1                                 NA
##   `PPT Sample as Percentage of Negative Control` `PPT MAD Z-Score`
##                                             <dbl>             <dbl>
## 1                                              NA                NA
##   `PPT MAD Log MAD Z-Score`
##                       <dbl>
## 1                        NA
##   `Median PPT Log Mad Z-Score for all siRNAs having the same seed sequence`
##                                                                      <dbl>
## 1                                                                       NA
##   `Number of siRNAs having the same seed sequence` `Cell Count, Sample`
```

Eliminate Missing Data

```
# ambion <- ambion[! is.na(ambion[,2]), ]
ambion <- ambion %>% filter(!is.na(Sample))
#apply(ambion, 2, function(x) sum(is.na(x)))
 ambion %>% summarize_all(function(x) sum(is.na(x)))
```

```
## # A tibble: 1 x 25
##   `Vendor Supplied Gene Symbol` Sample `Median Negative Control on Plate`
##                           <int>  <int>                                <int>
## 1                             0      0                                    0
##   `Median Positive Control on Plate`
##                                <int>
## 1                                0
##   `PPT Sample as Percentage of Negative Control` `PPT MAD Z-Score`
##                                            <int>             <int>
## 1                                              0                 0
##   `PPT MAD Log MAD Z-Score`
##                       <int>
## 1                         0
##   `Median PPT Log Mad Z-Score for all siRNAs having the same seed sequence`
##                                                                      <int>
```

Simplify the Data

Often it will be helpful to create a new data frame with only the data we wish to analyze.

```
#ambion.simple <- ambion[,c(19,25,1,21,7,13,17)]
ambion.simple <- select(ambion, 19,25,1,21,7,13,17)
ambion.simple[1,]
```

```
## # A tibble: 1 x 7
##   `Entrez GeneID` `Ambion siRNA ID` `Vendor Supplied Gene Symbol`
##             <dbl> <chr>             <chr>
## 1            3998 s8218             LMAN1
##   DESCRIPTION                `PPT MAD Log MAD Z-Score`
##   <chr>                                          <dbl>
## 1 lectin, mannose-binding, 1                     -4.82
##   `Sample Cell Count, MAD Z-Score Normalized to Negative Contol`
##                                                    <dbl>
## 1                                                   1.11
##   `Sample Mitophagy, MAD Z-Score Normalized to Negative Control`
##                                                    <dbl>
## 1                                                 0.0466
```

Simplify our Column Names

```
library(knitr, quietly = TRUE)
colnames(ambion.simple) <- c("GeneID","siRNA","Symbol","Description",
                             "PPT","Cells","Mitophagy")
kable(head(ambion.simple, n=4), format = "markdown")
```

GeneID	siRNA	Symbol	Description	PPT	Cells	Mitophagy
3998	s8218	LMAN1	lectin, mannose-binding, 1	-4.822230	1.1085178	0.0466291
51586	s28366	MED15	mediator complex subunit 15	-4.739415	0.2405872	-0.8887362
5217	s10379	PFN2	profilin 2	-4.539309	0.3994161	-1.7000123
57179	s226909	KIAA1191	KIAA1191	-4.516443	-1.0274124	-0.4280631

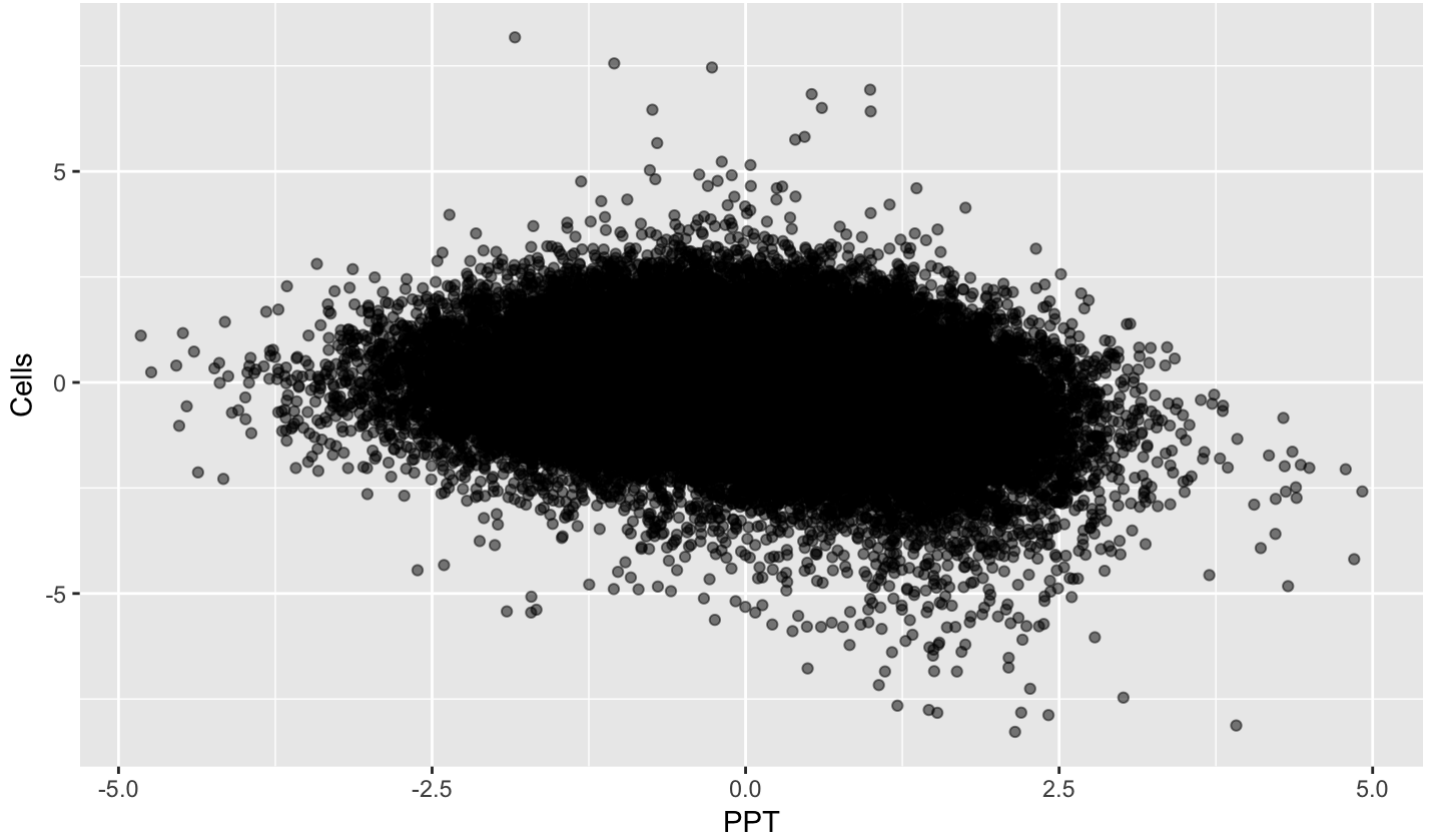
Simplify our Column Names

```
library(knitr, quietly = TRUE)
ambion.simple <- ambion.simple %>% set_names(c("GeneID","siRNA","Symbol","Descriptic
                                "PPT","Cells","Mitophagy"))
head(ambion.simple, n = 4) %>% kable(format='markdown')
```

GeneID	siRNA	Symbol	Description	PPT	Cells	Mitophagy
3998	s8218	LMAN1	lectin, mannose-binding, 1	-4.822230	1.1085178	0.0466291
51586	s28366	MED15	mediator complex subunit 15	-4.739415	0.2405872	-0.8887362
5217	s10379	PFN2	profilin 2	-4.539309	0.3994161	-1.7000123
57179	s226909	KIAA1191	KIAA1191	-4.516443	-1.0274124	-0.4280631

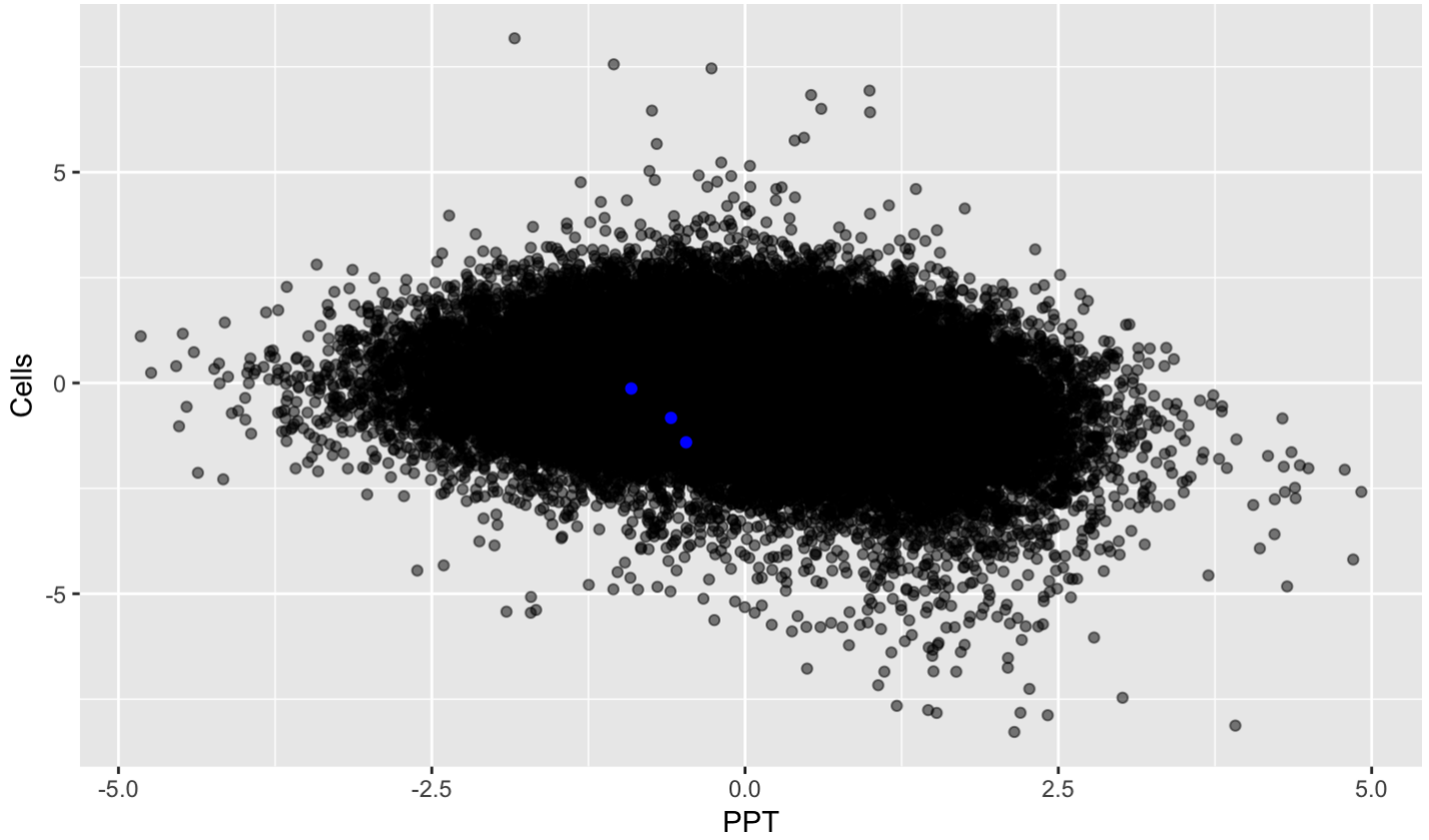
Evaluate how our variables interact

```
library(ggplot2, quietly = TRUE)
ggplot(ambion.simple, aes(x=PPT, y=Cells)) + geom_point(alpha=0.5)
```



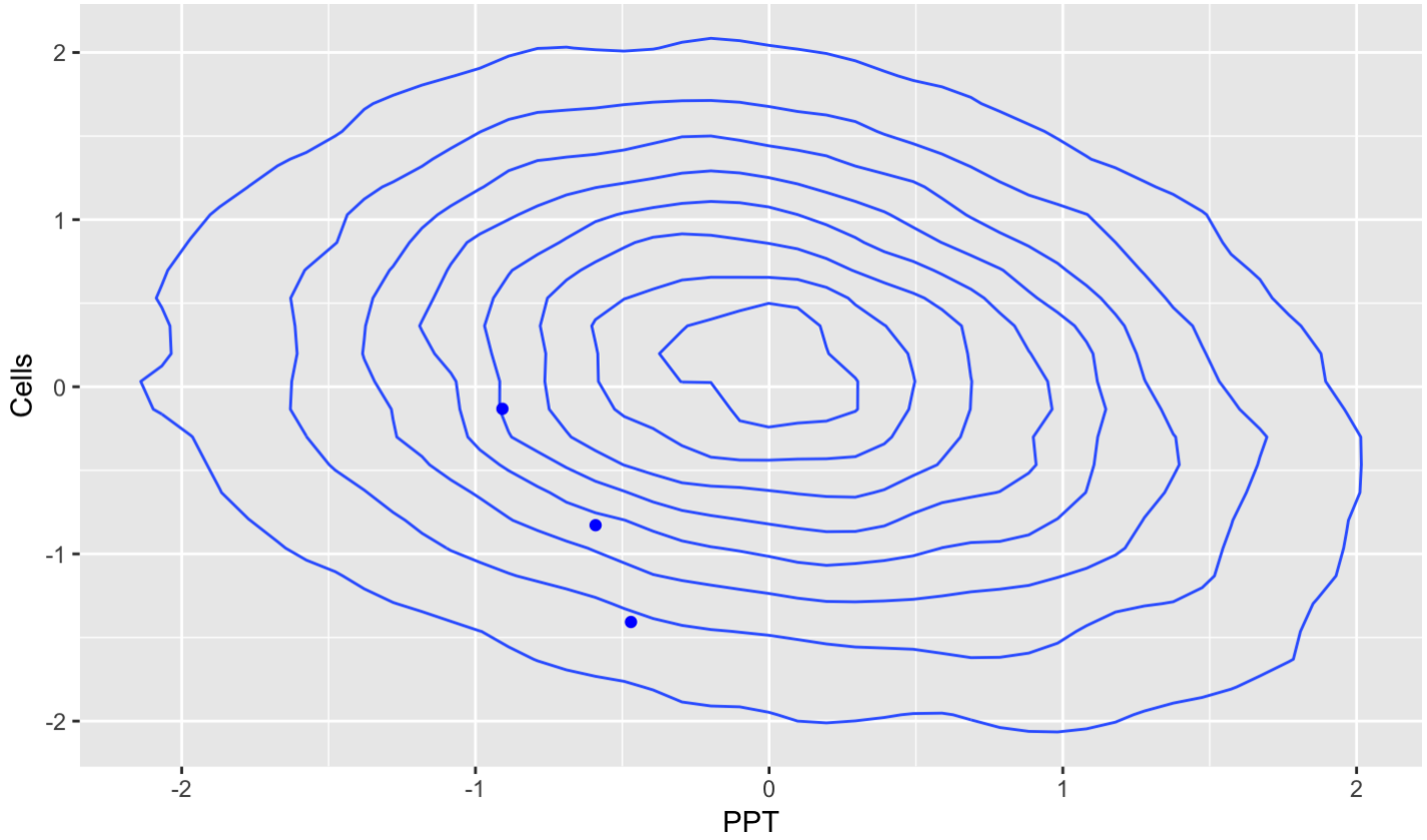
Evaluate how our variables interact

```
ggplot(ambion.simple, aes(x=PPT, y=Cells)) + geom_point(alpha=0.5) +
    geom_point(data = ambion.simple %>% filter(Symbol=='PARK2'),
               #ambion.simple[ambion.simple$Symbol == "PARK2",],
               color="blue")
```



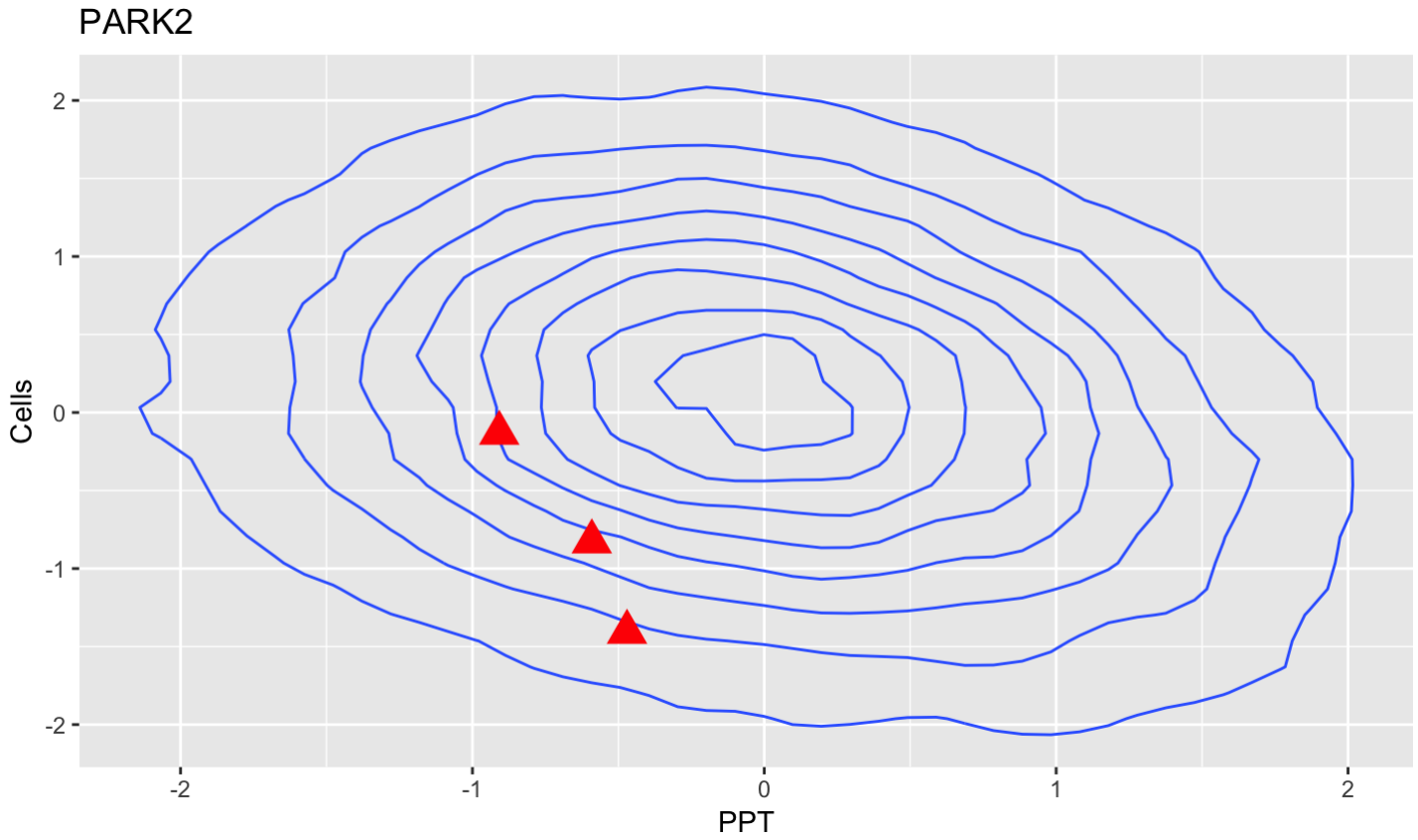
Refine the plot

```
ggplot(ambion.simple, aes(x=PPT, y=Cells)) + geom_density2d() +
    geom_point(data = ambion.simple %>% filter(Symbol=='PARK2'),
               color="blue")
```



Further refine the plot

```
ggplot(ambion.simple, aes(x=PPT, y=Cells)) + geom_density2d() +
    geom_point(data = ambion.simple %>% filter(Symbol=="PARK2"),
               color="red", shape=17, size =5) +
        ggtitle("PARK2")
```



Adding gene description

```
description <- ambion.simple$Description[ambion.simple$Symbol == "PARK2"][1]
description
```

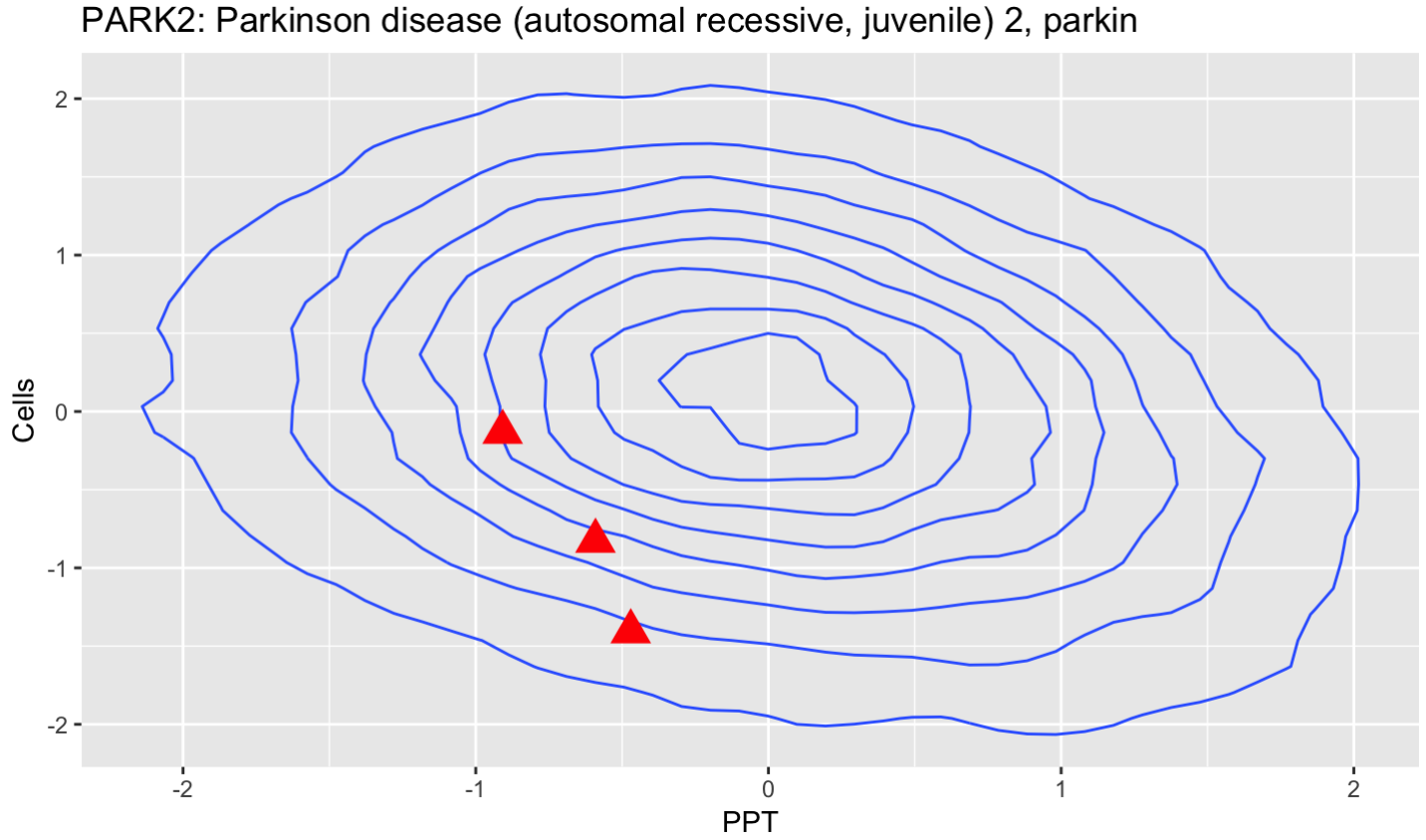
```
## [1] "Parkinson disease (autosomal recessive, juvenile) 2, parkin"
```

```
myTitle <- paste("PARK2",description,sep=": ")
myTitle
```

```
## [1] "PARK2: Parkinson disease (autosomal recessive, juvenile) 2, parkin"
```

Final version of plot

```
ggplot(ambion.simple, aes(x=PPT, y=Cells)) + geom_density2d() +
    geom_point(data = ambion.simple %>% filter(Symbol == "PARK2"),
               color="red", shape=17, size =5) +
        ggtitle(myTitle)
```



Making the refined plot into a function

Now that we have our custom plot looking right, we would like to be able to do the same for other genes but without so much typing. First, make a new R Script in RStudio:

Constructing a new function from your history

Frequently, making a function will simply be a function of selecting the right parts of your history and hitting the “to source” button.

Function with PARK2 hard coded

```
graphGene <- function(gene) {
  description <- ambion.simple$Description[ambion.simple$Symbol == "PARK2"][1]
  myTitle <- paste("PARK2",description,sep=": ")
  ggplot(ambion.simple, aes(x=PPT, y=Cells)) + geom_density2d() +
    geom_point(data = ambion.simple %>% filter(Symbol=="PARK2"),
               color="red", shape=17, size =5) +
    ggtitle(myTitle)
}
```

Function made generic

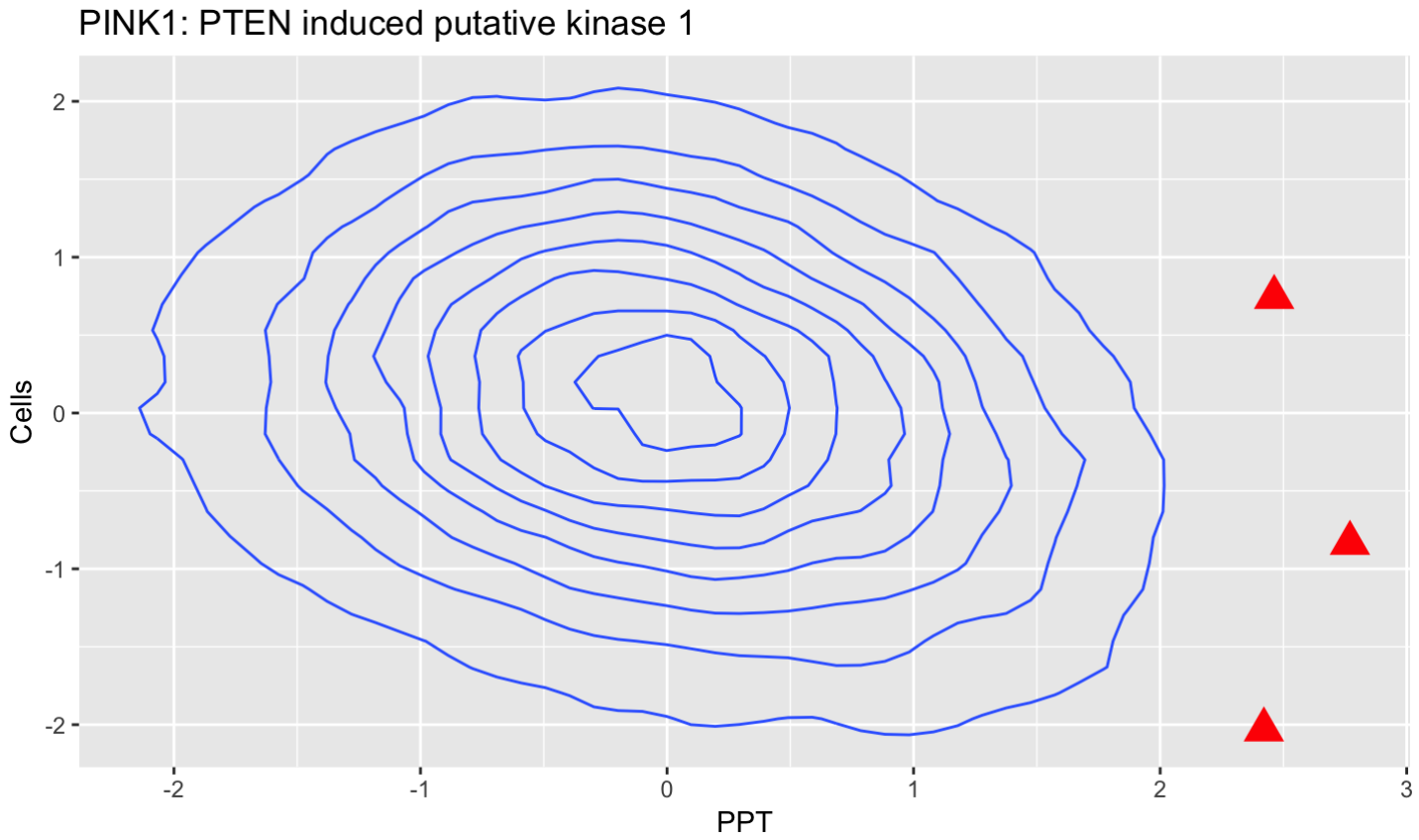
```
graphGene <- function(gene) {
  description <- ambion.simple$Description[ambion.simple$Symbol == gene][1]
  myTitle <- paste(gene,description,sep=": ")
  ggplot(ambion.simple, aes(x=PPT, y=Cells)) + geom_density2d() +
    geom_point(data = ambion.simple %>% filter(Symbol == gene),
               color="red", shape=17, size =5) +
    ggtitle(myTitle)
}
```

Function made generic, with checks

```
graphGene <- function(gene) {
  if(!is.character(gene)) stop("Need to provide a string")
  if(!(gene %in% ambion.simple$Symbol)) stop("Need to provide a valid gene")
  description <- ambion.simple$Description[ambion.simple$Symbol == gene][1]
  myTitle <- paste(gene,description,sep=": ")
  ggplot(ambion.simple, aes(x=PPT, y=Cells)) + geom_density2d() +
    geom_point(data = ambion.simple %>% filter(Symbol == gene),
               color="red", shape=17, size =5) +
    ggtitle(myTitle)
}
```

Our function in action

```
graphGene("PINK1")
```



Default values for function arguments

```
pdfGene <- function(gene, file=paste(gene,".pdf", sep="")) {
    pdf(file, width=5, height=5)
    graphGene(gene)
    dev.off()
}
```

Passing on extra arguments to our function

We can use the ellipse notation (…) to indicate that extra arguments to our function should be passed on to a function that is inside our function (in this case pdf).

```
pdfGene <- function(gene, file=paste(gene,".pdf", sep=""), ...) {
    pdf(file, ...)
    graphGene(gene)
    dev.off()
}
pdfGene("PINK1", width=10, height=10)
```

Control of Flow: If/Else

We can decide whether something happens in our function using “if” and “if/else”.

```
sillyFunction <- function(x) {
  if (x < 5) {
    returnValue <- x
  }
  else {
    returnValue <- x / 2
  }
  return(returnValue)
}
sillyFunction(12)
```

```
## [1] 6
```

Control of Flow: For

```
pdfGenes <- function(genes, ...) {
    for (gene in genes) { # This will work through gene by gene
      pdfGene(gene, file = paste0(gene, '.pdf'), ...)
    }
}
pdfGenes(c("PLK1","PINK1","BRCA1"))
```