Week_2_IntroToRmd-Part2-Fancy

Seema Plaisier

06/21/22

Fancier data structures and plotting

In this Rmd, we will take some more complicated tools in R and break them down so that you can see how they work.

Things we will cover in this Rmd are:

1) installing packages: pull in publicly available code for your own use
2) complex data structure DGEList: used in the differential expression pipeline you will be using
3) detailed plots using 'ggplot': fancy plotting tool to compose really pretty figures

Installing packages

One of the main reasons for learning to use R is that there is an enormous collection of developed and tested code written in R that you can use to do analysis. Functions are included in bundles called packages and made available for use in the program by using the 'library' command in R. This included code will extend the functionality of R beyond the basic structures. In order to include a library, it has to be installed (code has to be downloaded to your local R environment). In this next chunk, the 'require' function is used to check to see if specific packages are currently installed, returning TRUE if yes and FALSE if no. If 'require' returns FALSE, the '!' ("not") operator returns the opposite value of TRUE and calls the 'install.packages' function to install the package. For packages in the large biology focused set of packages called Bioconductor, there is a special install function called 'BiocManager::install' which itself will be installed if you don't have it in your R environment.

```
if(!require(tidyverse)){
    install.packages("tidyverse")
}
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(BiocManager)){
    install.packages("BiocManager")
}
```

```
## Loading required package: BiocManager
```

```
if(!require(ggplot2)){
    BiocManager::install("ggplot2")
}
if(!require(edgeR)){
  BiocManager::install("edgeR")
}
```

```
## Loading required package: edgeR
```

```
## Loading required package: limma
```

```
library(tidyverse)
library(BiocManager)
library(ggplot2)
library(edgeR)
```

Custom data types: DGEList

Using a combination of basic data types in R, specific packages use specialized data types as input for functions. The construction of these data types should be covered in the documentation for the package. As an example, we will use the DGEList data structure from the package edgeR. Here we will go over it in detail so you understand how to work with it since will use it to identify differentially expressed genes.

There are two main elements of a DGEList object: 1) a matrix of counts 2) a data frame of sample information: a) row for each sample, columns for groups those samples belong to b) lib.size (number of genes that have counts) c) normalization factors (which start as all 1, but we can fill in using the appropriate functions)

```
# ? or ?? can be used to look up documentation for classes
# and functions
`?`(`?`(edgeR::DGEList))
```

```
## starting httpd help server ... done
```

```
# first entry in the Help tab or in a newly opened browser
# is edgeR::DGEList-class, click on it to read

# DGEList objects contains a matrix of read counts and
# other information that we will need to normalize and
# process the read count data

# Read counts are the output from programs like
# featureCounts, which take the sequences read that align
# to particular genes and count them up as a measure of how
# much those genes are expressed
```

```
# in this example, we are going to create a matrix of
# random numbers to stand in for counts data and numbers 1
# and 2 to represent groups for our fake samples
fake_counts = matrix(rnbinom(10000, mu = 5, size = 2), ncol = 4)
fake_groups = rep(1:2, each = 2)

# now we are going to construct a DGEList object with our
# fake counts and groups as features and set the lib.size
# feature to the sum of all the counts in each column
d <- DGEList(counts = fake_counts, group = fake_groups, lib.size = colSums(fake_counts))
dim(d)  # will show you the dimensions of the counts matrix
```

```
## [1] 2500    4
```

```
colnames(d)  # will show you the sample names (filled in by default since we did not specify here)
```

```
## [1] "Sample1" "Sample2" "Sample3" "Sample4"
```

```
d$samples  # will show you the sample information data.frame created from the parameters we passed in
```

```
##         group lib.size norm.factors
## Sample1     1    12189            1
## Sample2     1    12492            1
## Sample3     2    12631            1
## Sample4     2    12601            1
```

```
# you should see that the norm.factors are 1 by default we
# will use the calcNormFactors function to get factors that
# we can multiply by the counts to keep the range of counts
# comparable between samples

# in our real data, we will also pass in some of the
# optional elements such as genes (name of rows)

# once all this data is filled in, you can pass the DGEList
# variable d into functions from the edgeR package that are
# designed to take DGEList objects as input parameters,
# keeping the code short and tidy using this data type
# allows the functions in edgeR to assume the name of the
# elements to be counts, samples, etc
```

Set working directory and data directories

Your working directory is where all your output files will be stored by default. You can set it by copying the path of the directory from the explorer windows. Make sure you change the slashes to forward slashes '/' if needed.

The 'setwd' function sets your working directory to the path you specify. The 'list.files' and 'list.dirs' functions can be used to print what's inside that directory.

```
# store the path to the working directory in a variable
# make sure to include the / at the end of the directory path if you plan to put it together with the p
working_directory = "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/Week 2/"  # THIS WILL NEED TO BE CHAN
#working_directory = "/scratch/splaisie/test_CURE"  # example of path to working directory on Agave

# set the working directory to where you want your output to go
setwd(working_directory)

# see what's currently in your working directory
# files:
list.files(working_directory)
```

```
##  [1] "~$tes-R_Walkthrough_video.docx"
##  [2] "~WRL3226.tmp"
##  [3] "dp_plot.pdf"
##  [4] "emp_salary_plot.pdf"
##  [5] "Extras.docx"
##  [6] "Notes-R_Walkthrough_video.docx"
##  [7] "Resources.docx"
##  [8] "subset.csv"
##  [9] "subset.txt"
## [10] "Week_2_IntroToRmd-Map_of_RStudio.pdf"
## [11] "Week_2_IntroToRmd-Map_of_RStudio.Rmd"
## [12] "Week_2_IntroToRmd-Part1-Basics.pdf"
## [13] "Week_2_IntroToRmd-Part1-Basics.Rmd"
## [14] "Week_2_IntroToRmd-Part2-Fancy.pdf"
## [15] "Week_2_IntroToRmd-Part2-Fancy.Rmd"
## [16] "Week_2_IntroToRmd.pdf"
## [17] "Week_2_IntroToRmd.Rmd"
```

```
# directories:
list.dirs(working_directory)
```

```
## [1] "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/Week 2/"
```

```
# set the path to other directories you might need to reference
untrimmed_directory = "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/Week 3/"
print(untrimmed_directory)
```

```
## [1] "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/Week 3/"
```

```
# don't need to set it to working directory

# can instead use paste0 command to add the file path of a directory to the file name
# make sure you have the / at the end to make sure the file name is not added to the deepest directory
file_path = paste0(untrimmed_directory, "DE_Pipeline_UntrimmedData.Rmd")
print (file_path)
```

```
## [1] "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/Week 3/DE_Pipeline_UntrimmedData.Rmd"
```

Fancier Data Plotting with ggplot

In addition to basic plotting functions in R, there are a lot of awesome packages for making fancier plots. One of the most widely used is ggplot or newer version ggplot2.

Here is a gallery of examples to give you a taste of how many beautiful types of figures/plots you can make: https://r-graph-gallery.com/

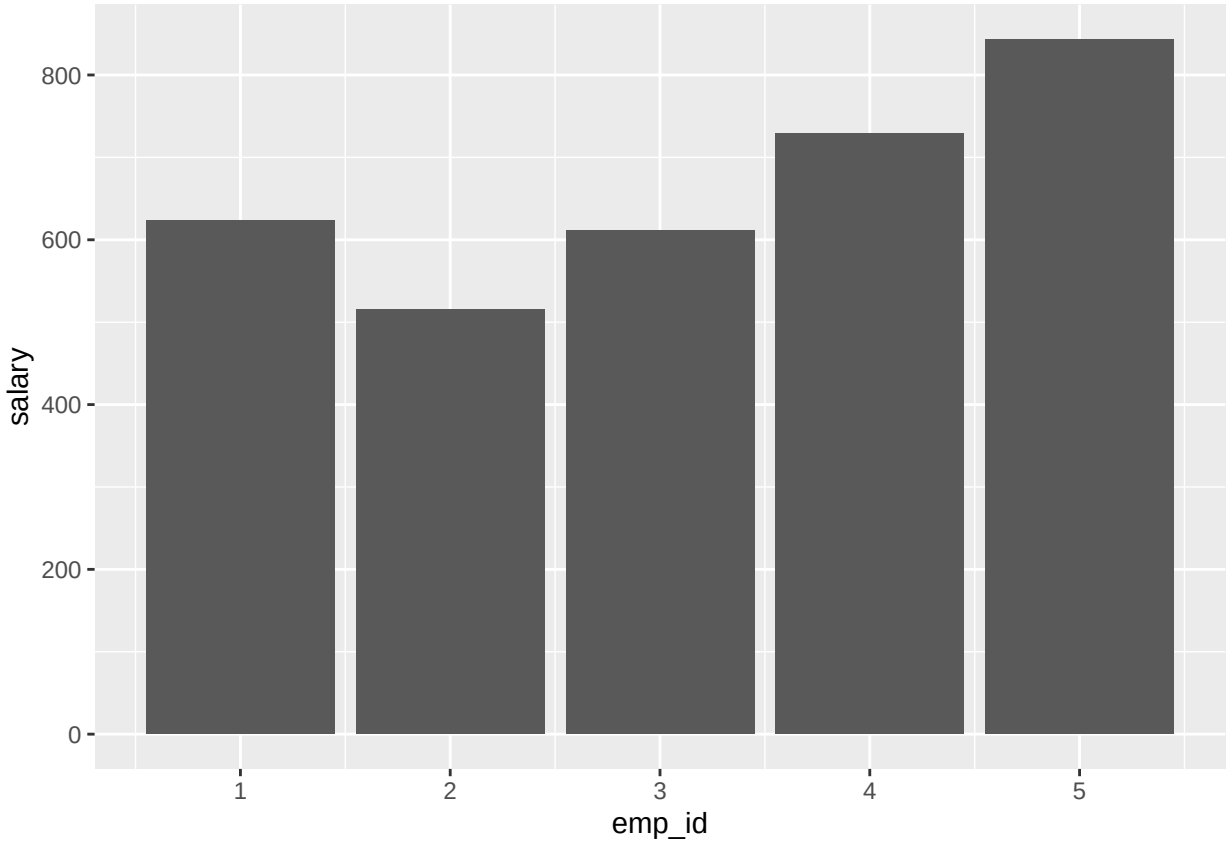
The way that ggplot works is that you set the data you are wanting to plot to a ggplot variable and then add in all the features you would like to use to customize your plot. There are a ton of options for plots and customization, you just have to go step by step figuring out how to get what you want.

In this introductory example, we are adding features one at a time so you can get a better idea of what the different modifications do. If you look at ggplot code other people have written, you will often see all the modifications added together in one line before plotting. Either way works.

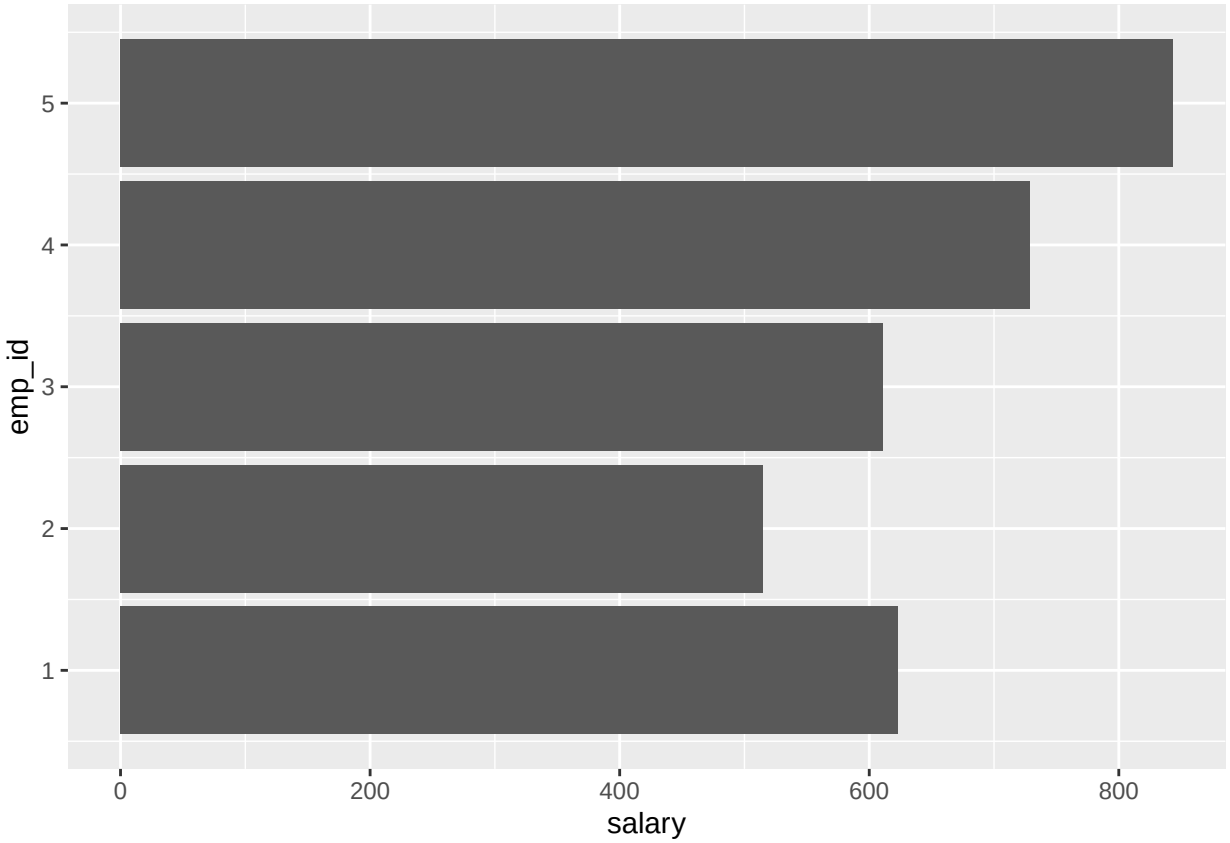
```
# have to call library(ggplot2) which we did above

# set up data to work with as an example
emp.data <- data.frame(emp_id = c(1:5), emp_name = c("Rick",
    "Dan", "Michelle", "Ryan", "Gary"), salary = c(623.3, 515.2,
    611, 729, 843.25))
print(emp.data)  # show the contents of employee data frame
```

```
##   emp_id emp_name salary
## 1      1     Rick 623.30
## 2      2      Dan 515.20
## 3      3 Michelle 611.00
## 4      4     Ryan 729.00
## 5      5     Gary 843.25
```

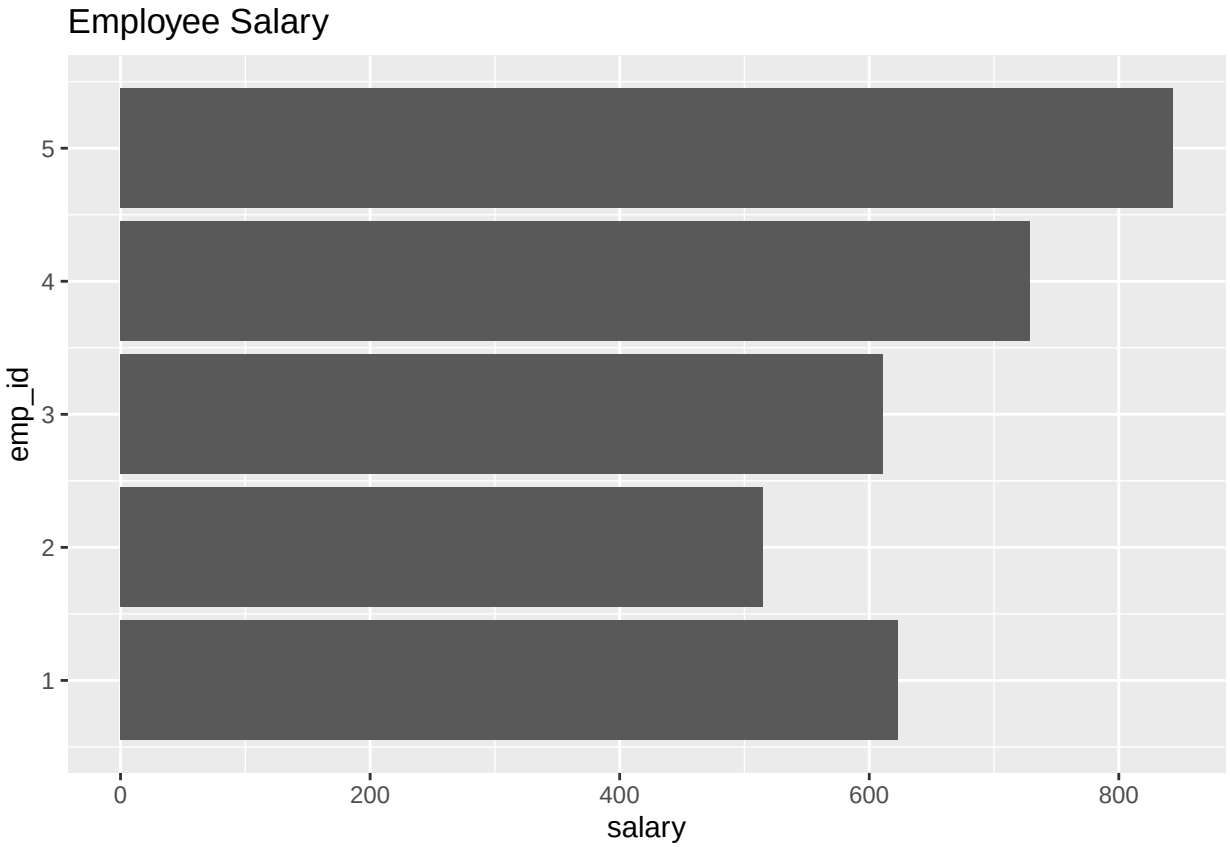
```
# Initialize plot variable with data and basic type
p = ggplot(data = emp.data, aes(x = emp_id, y = salary)) + geom_bar(stat = "identity")

# plot the ggplot variable p
plot(p)
```



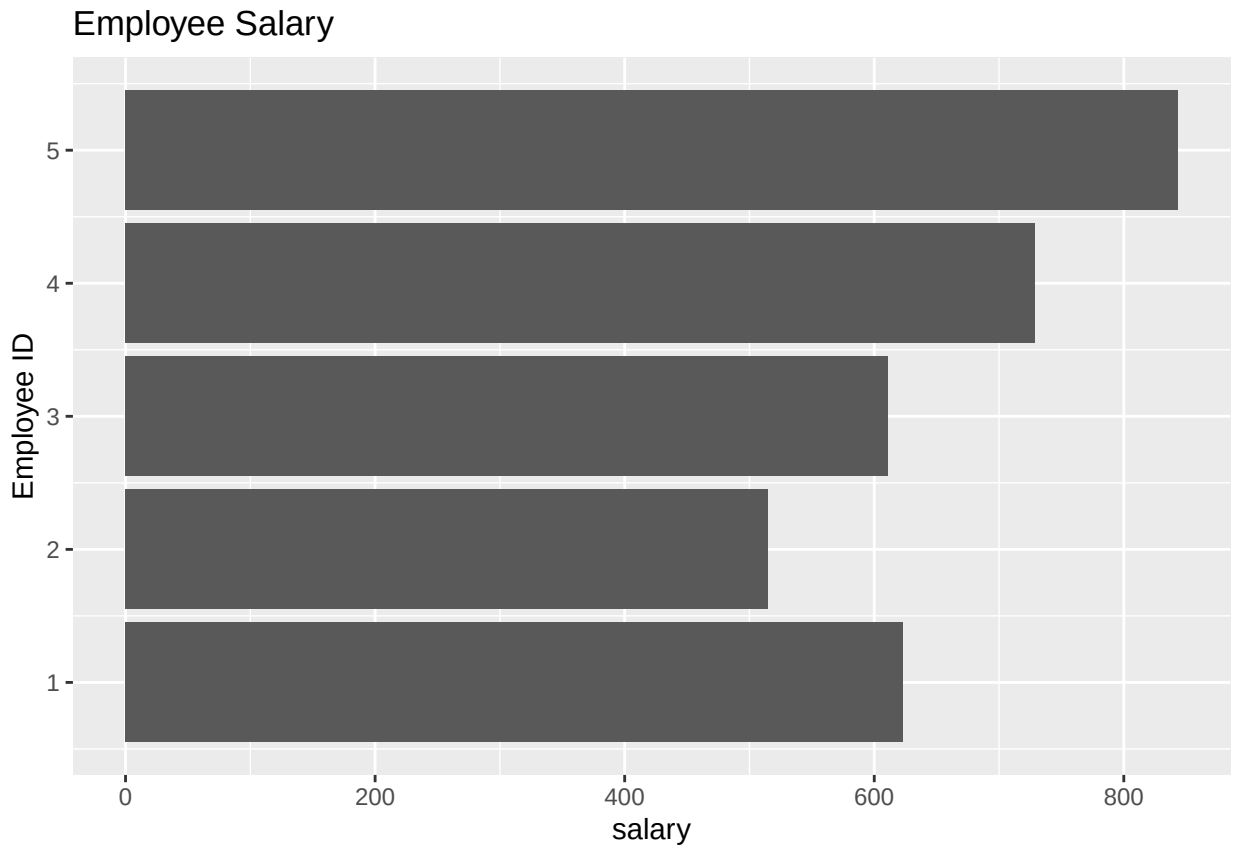

```
# let's say we wanted the plot to be going left to right
p = p + coord_flip()
plot(p)
```



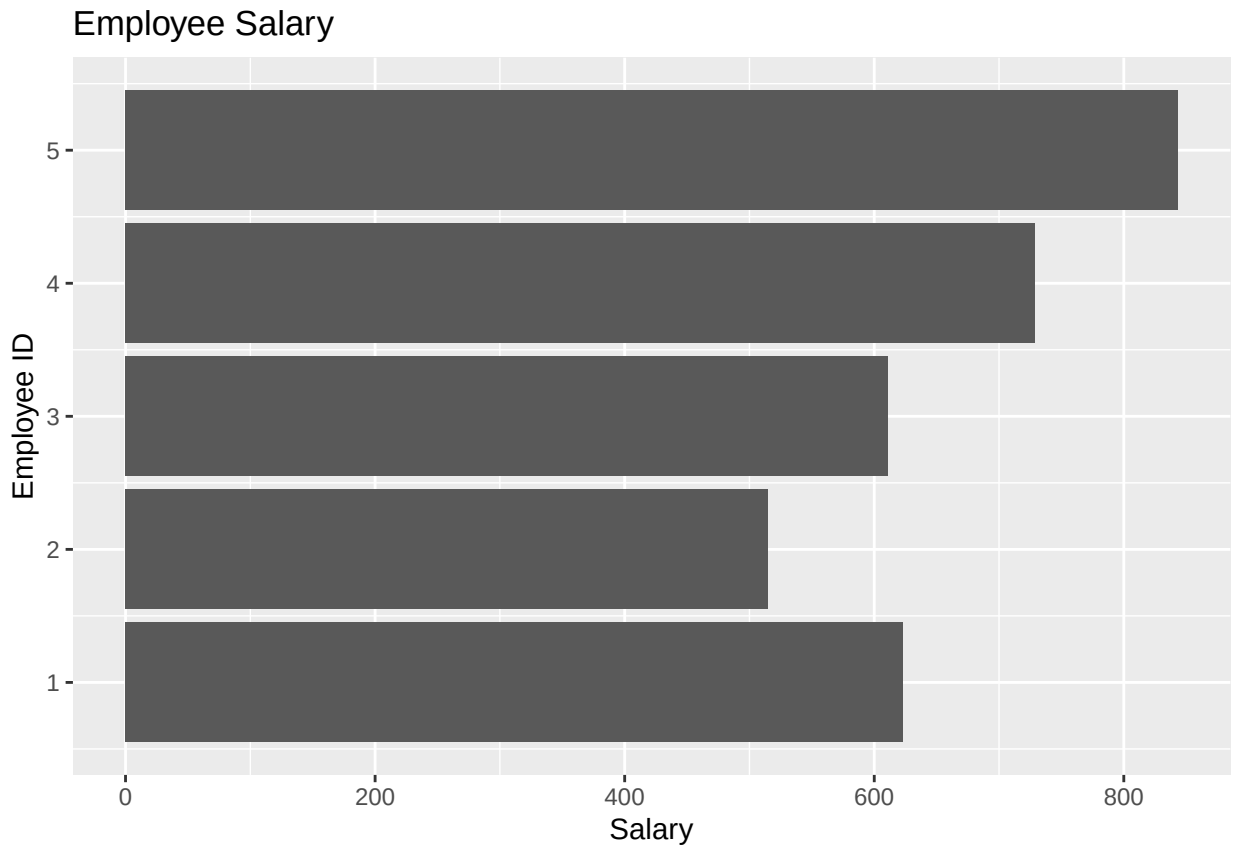

```
# add a title to the plot
p = p + ggtitle("Employee Salary")
plot(p)
```



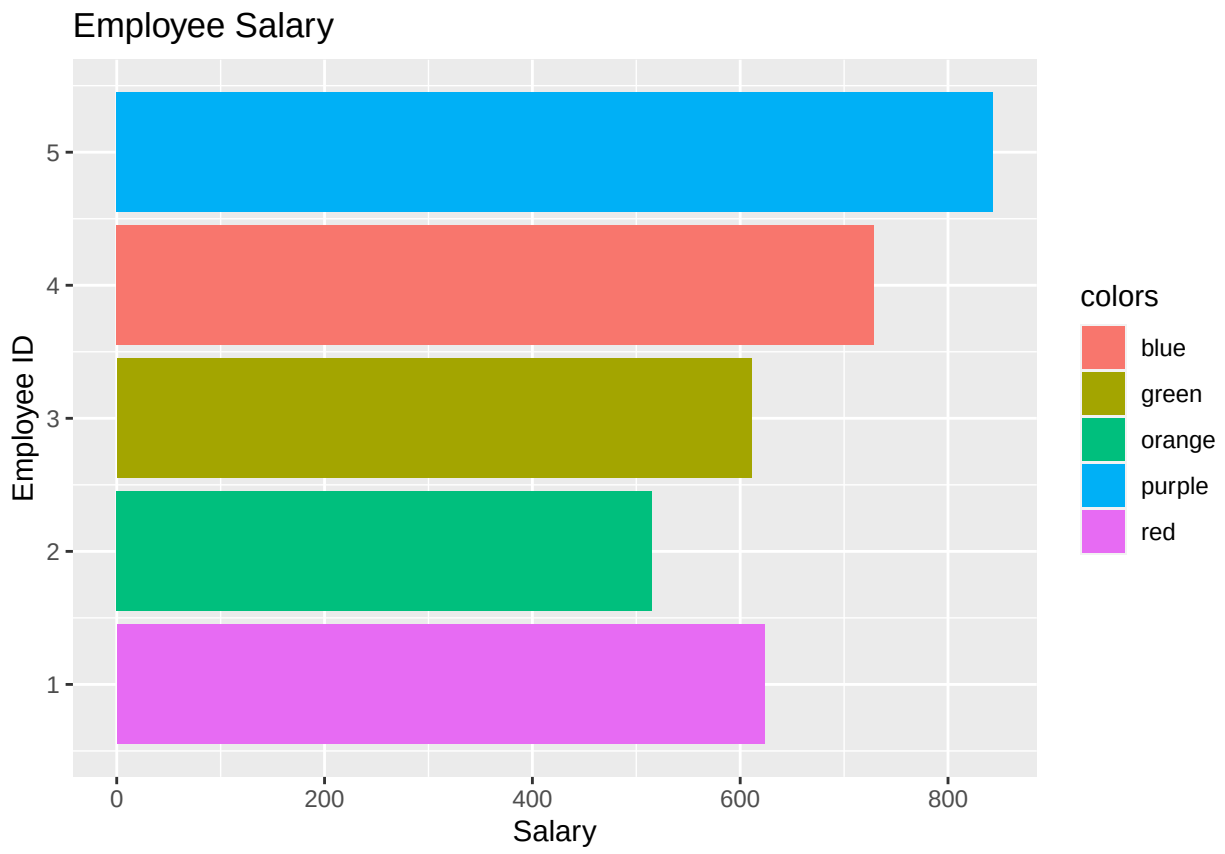

```
# change the axis labels
p = p + xlab("Employee ID")
plot(p)
```



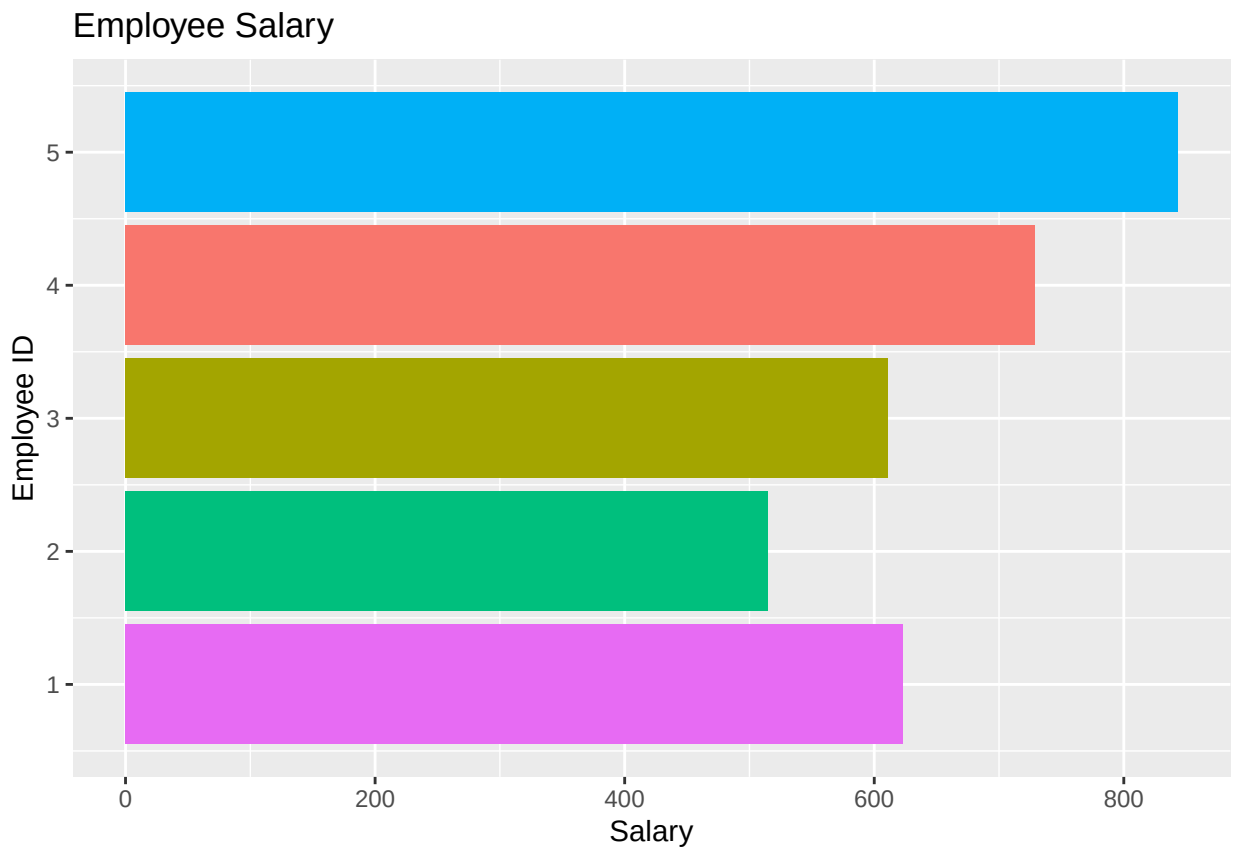

```
p = p + ylab("Salary")
plot(p)
```




```
# if you want to use specific colors, for the bars, you can
# use the fill parameter when setting your ggplot variable
# setting the colors automatically adds a legend, I am
# showing how to remove it in our case

# here I will show you how to use ggplot all in one line

# specific colors for the bars
colors = c("red", "orange", "green", "blue", "purple")

# construct ggplot object all in one line
p = ggplot(data = emp.data, aes(x = emp_id, y = salary, fill = colors)) +
    geom_bar(stat = "identity") + coord_flip() + ggtitle("Employee Salary") +
    xlab("Employee ID") + ylab("Salary")
plot(p)
```




```
# remove the legend because it's not saying anything
# important
p = ggplot(data = emp.data, aes(x = emp_id, y = salary, fill = colors)) +
    geom_bar(stat = "identity") + coord_flip() + ggtitle("Employee Salary") +
    xlab("Employee ID") + ylab("Salary") + theme(legend.position = "none")
plot(p)
```



Save plot

To save a figure, we use the 'ggsave' function. The type of file saved is determined by the file type extension you give. That is, if you give a file name ending in ".pdf", your output file will be saved as a PDF with the plot in it. If you give it a file name ending in ".png", you will get the figure in a PNG image file. The output file will be saved in your working directory by default.

```
ggsave("emp_salary_plot.pdf", plot = p)
```

```
## Saving 6.5 x 4.5 in image
```

```
# check to make sure you can find your output file after it is saved
# when you Knit a report, this save function will be run, so you
#    should see your output files and report in your working directory
```

List all the packages used for future reference

```
sessionInfo()
```

```
## R version 4.2.0 (2022-04-22 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
```

```
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] edgeR_3.38.1        limma_3.52.1        BiocManager_1.30.18
##  [4] forcats_0.5.1       stringr_1.4.0       dplyr_1.0.9
##  [7] purrr_0.3.4         readr_2.1.2         tidyr_1.2.0
## [10] tibble_3.1.7        ggplot2_3.3.6       tidyverse_1.3.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.8.3     locfit_1.5-9.5   lubridate_1.8.0  lattice_0.20-45
##  [5] assertthat_0.2.1 digest_0.6.29    utf8_1.2.2       R6_2.5.1
##  [9] cellranger_1.1.0 backports_1.4.1  reprex_2.0.1     evaluate_0.15
## [13] highr_0.9        httr_1.4.3       pillar_1.7.0     rlang_1.0.2
## [17] readxl_1.4.0     rstudioapi_0.13  rmarkdown_2.14   labeling_0.4.2
## [21] munsell_0.5.0    broom_0.8.0      compiler_4.2.0   modelr_0.1.8
## [25] xfun_0.31        pkgconfig_2.0.3  htmltools_0.5.2  tidyselect_1.1.2
## [29] fansi_1.0.3      crayon_1.5.1     tzdb_0.3.0       dbplyr_2.1.1
## [33] withr_2.5.0      grid_4.2.0       jsonlite_1.8.0   gtable_0.3.0
## [37] lifecycle_1.0.1  DBI_1.1.2        magrittr_2.0.3   formatR_1.12
## [41] scales_1.2.0     cli_3.3.0        stringi_1.7.6    farver_2.1.0
## [45] fs_1.5.2         xml2_1.3.3       ellipsis_0.3.2   generics_0.1.2
## [49] vctrs_0.4.1      tools_4.2.0      glue_1.6.2       hms_1.1.1
## [53] fastmap_1.1.0    yaml_2.3.5       colorspace_2.0-3 rvest_1.0.2
## [57] knitr_1.39       haven_2.5.0
```