Lab 2

Maria Bassem_20011141

2024-02-21

Part 1: Data Visualization:

Task 1.1: Visualizing Trend

In this part, I learnt how to select certain rows from the data. Then I calculated the mean of each gene so, colMeans() was used to facilitate the calculation of mean expression levels across all genes. After that, I created a dataframe for plotting the data. Then, I plotted a scatter plot between the first sample and the mean gene expression. After that, a new data.frame containing the first 10 sorted genes based on the mean expression was created. A bar chart (column chart) was created and then a box plot comparing the first gene only against all the cancer types (phenotype)

1. Import ggplot2 package. For more information on ggplot2 check the references section.

```
# we can use library(ggplot2) or library(tidyverse)
library(tidyverse) # include the ggplot2 and other tools
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.4     ✔ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.4.4     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.1
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

2. Create a smaller dataframe of the first 150 genes

```
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

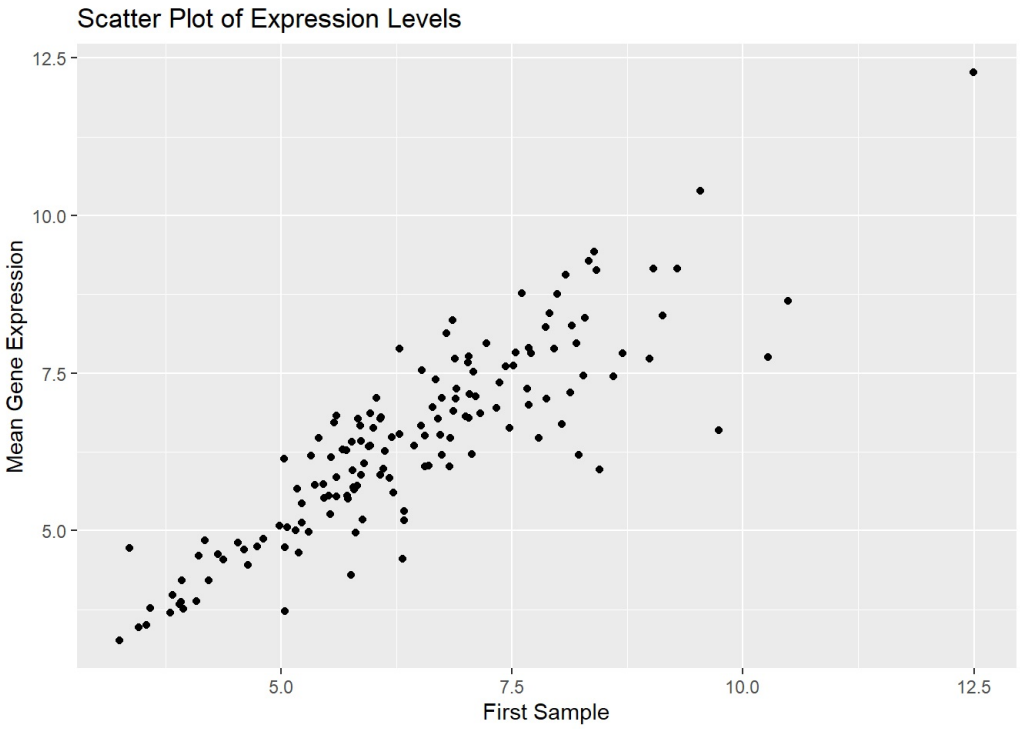
```
brain_data <- fread("D:\\Third Year Computer\\Term 2\\Bio\\Labs\\Lab 1\\Brain_GSE50161.csv")
brain_data <- as.data.frame(brain_data)

small_data <- select(brain_data, names(brain_data) [1:152])
```

3. Calculate the mean expression for all genes

```
mean_fun <- colMeans(small_data[, -c(1,2)])

gene_names <- names(small_data[, -c(1,2)])

# Create a data frame with genes and their corresponding mean expression values
mean_fun_df <- data.frame(
  Gene = gene_names,
  Mean_Expression = mean_fun
)
```

4. Create a ggplot2 scatter plot showing expression levels of the first sample and the mean gene expression

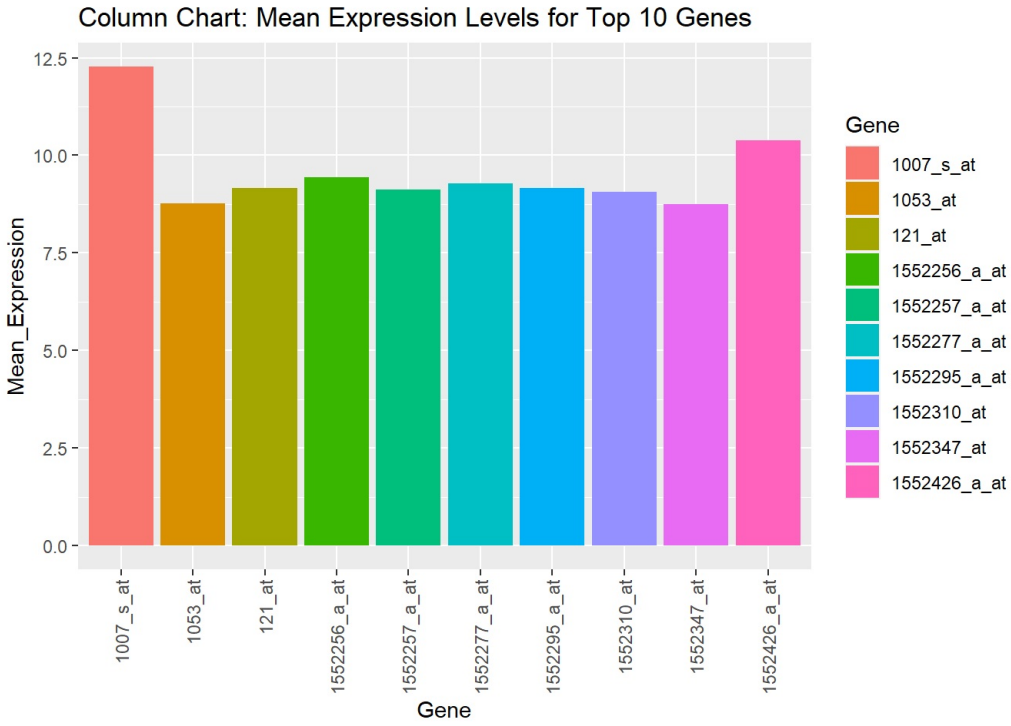
```
# Extracting the first sample (first row)
first_row <- small_data[1, -c(1, 2)]

combined_data <- data.frame(
  mean_expression = mean_fun,
  X1 = t(first_row) # transpose the first_sample
)
combined_data <- rename(combined_data, first_sample = X1)
# Create a scatter plot
ggplot(combined_data, aes(x = first_sample, y = mean_expression)) + geom_point() +
  labs(title = "Scatter Plot of Expression Levels",
       x = "First Sample",
       y = "Mean Gene Expression")
```



Task 1.2: Advanced Visualization:

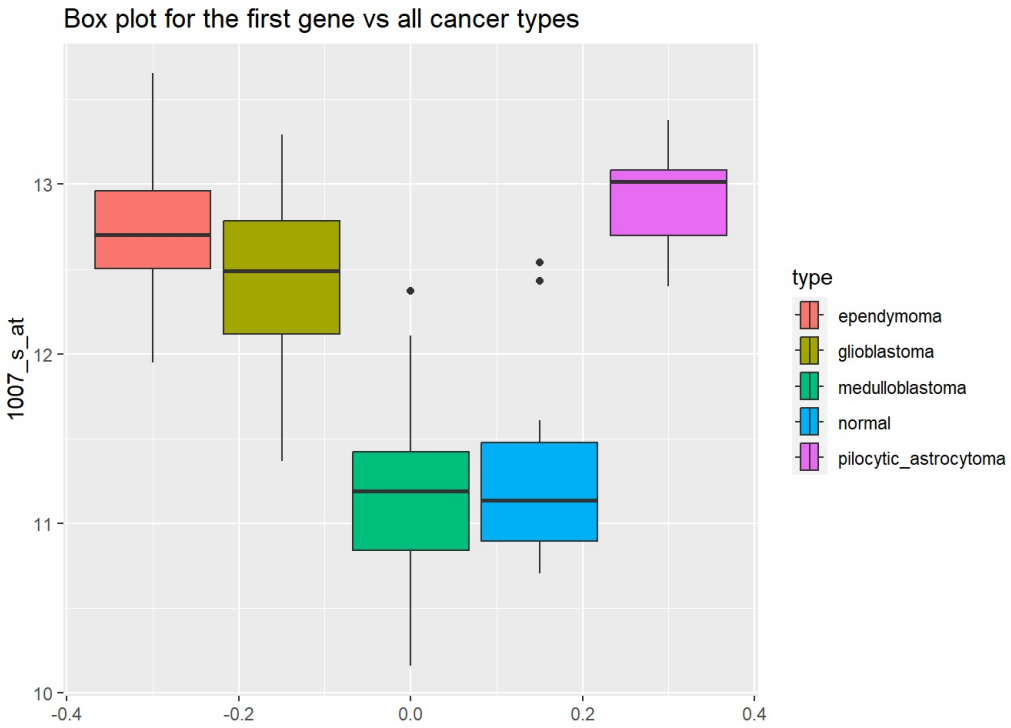
1. Generate a ggplot2 bar plot showing the mean expression levels for the top 10 genes

```
# Sort the data frame based on Mean_Expression in descending order
mean_fun_df <- mean_fun_df[order(-mean_fun_df$Mean_Expression), ]

# Select the first 10 rows
top_10_mean_fun_df <- head(mean_fun_df, 10)
# stat="identity" uses the actual values supplied in the data for the heights of the bars.
ggplot(top_10_mean_fun_df, aes(x = Gene, y = Mean_Expression, fill=Gene)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1)) +
  labs(title = "Column Chart: Mean Expression Levels for Top 10 Genes")
```



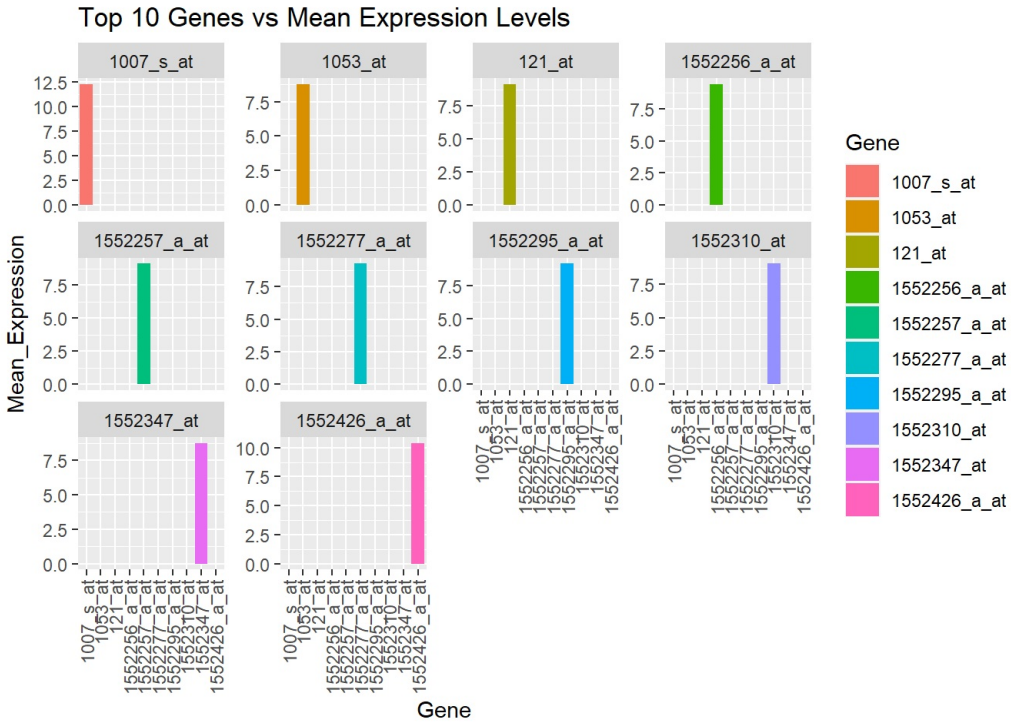
2. Compare the first gene only for all the cancer types (phenotype) by drawing a box plot (hint: specify the fill as the cancer type)

```
ggplot(small_data, aes(x=`1007_s_at`, fill=type)) +
 geom_boxplot() +
 coord_flip() +
 labs(title = "Box plot for the first gene vs all cancer types")
```

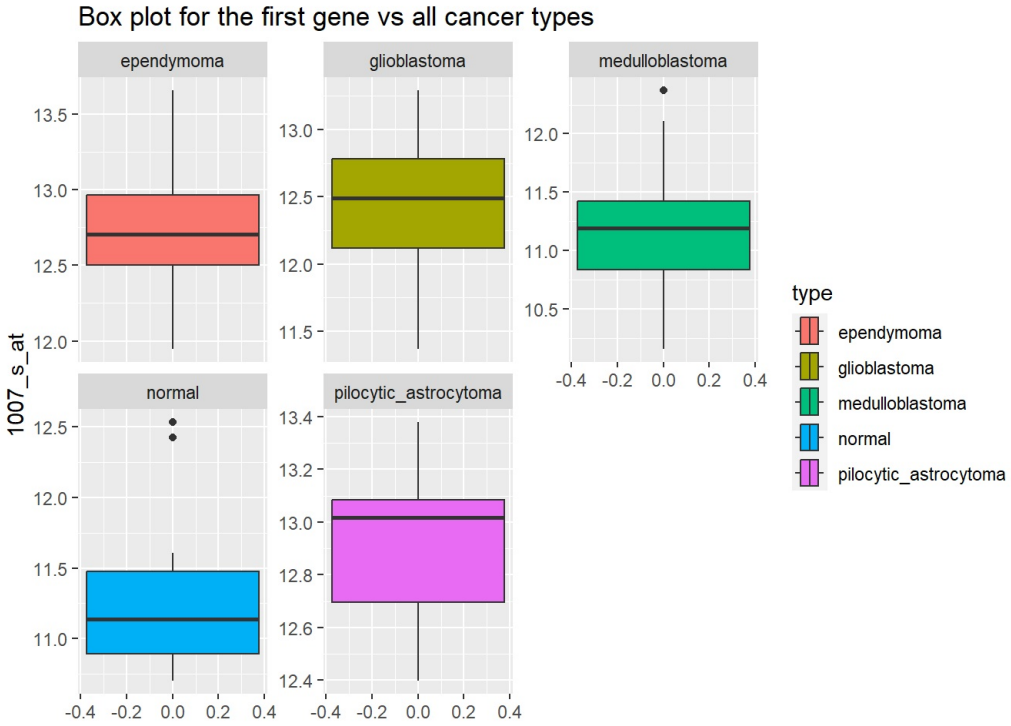


3. Split the plots on different frames (hint: Facets)

```
ggplot(top_10_mean_fun_df, aes(x = Gene, y = Mean_Expression, fill=Gene)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1)) +
  labs(title = "Top 10 Genes vs Mean Expression Levels") +
  facet_wrap(~Gene, scale="free_y")
```




```
# facet_wrap(~type, scale="free_y") --> creates separate facets (windows) for each unique value of the type varia
ble.
ggplot(small_data, aes(x=`1007_s_at`, fill=type)) +
 geom_boxplot() +
 coord_flip() +
 labs(title = "Box plot for the first gene vs all cancer types") +
 facet_wrap(~type, scale="free_y")
```



Part 2: Sequence Alignment:

In this task, I learnt about the library called “Biostrings” which contained sequence alignment. I learnt how to use the `pairwiseAlignment` function to align sequences, and how to extract the output from the `PairwiseAlignmentsSingleSubject` object which is a form of S4

Task 2.1: Installing Biostrings

Use the below Sequences in your execution: 1. Sequence A: AGCTGAACTAGCTAGCTGACTGACTGACTAGCTAGCTGACTAGCTG 2. Sequence B: AGCGAACTAGCTGACTGACGACTGACTAGCTGACTAGCTGACTAGC

```r
#install.packages("BiocManager")
library(BiocManager)
#BiocManager::install("Biostrings")
library(Biostrings)
```

```
## Loading required package: BiocGenerics
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:lubridate':
##
##     intersect, setdiff, union
```

```
## The following objects are masked from 'package:dplyr':
##
##     combine, intersect, setdiff, union
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which.max, which.min
```

```
## Loading required package: S4Vectors
```

```
## Loading required package: stats4
```

```
##
## Attaching package: 'S4Vectors'
```

```
## The following objects are masked from 'package:data.table':
##
##     first, second
```

```
## The following objects are masked from 'package:lubridate':
##
##     second, second<-
```

```
## The following objects are masked from 'package:dplyr':
##
##     first, rename
```

```
## The following object is masked from 'package:tidyr':
##
##     expand
```

```
## The following object is masked from 'package:utils':
##
##     findMatches
```

```
## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname
```

```
## Loading required package: IRanges
```

```
##
## Attaching package: 'IRanges'
```

```
## The following object is masked from 'package:data.table':
##
##     shift
```

```
## The following object is masked from 'package:lubridate':
##
##     %within%
```

```
## The following objects are masked from 'package:dplyr':
##
##     collapse, desc, slice
```

```
## The following object is masked from 'package:purrr':
##
##     reduce
```

```
## The following object is masked from 'package:grDevices':
##
##     windows
```

```
## Loading required package: XVector
```

```
##
## Attaching package: 'XVector'
```

```
## The following object is masked from 'package:purrr':
##
##     compact
```

```
## Loading required package: GenomeInfoDb
```

```
##
## Attaching package: 'Biostrings'
```

```
## The following object is masked from 'package:base':
##
##     strsplit
```

Task 2.2: Run Pairwise Alignment

```
# initializing the sequences
sequenceA <- DNAString("AGCTGAACTAGCTAGCTGACTGACTGACTAGCTAGCTGACTAGCTG")
sequenceB <- DNAString("AGCGAACTAGCTGACTGACGACTGACTAGCTGACTAGCTGACTAGC")

# pairwise alignment calculation
alignment <- pairwiseAlignment(
  pattern = sequenceA,
  subject = sequenceB,
  substitutionMatrix = NULL,
  gapOpening = -2,
  gapExtension = -8,
  type = "global"
)
```

Notes on the `pairwiseAlignment()` function:

- substitutionMatrix = NULL –> diagonal values and off-diagonal values are set to 0 and 1 respectively.
- gapOpening –> the cost for opening a gap in the alignment
- gapExtension –> incremental cost along the length of the gap in the alignment
- type = “global” –> aligns whole strings with end gap penalties

```
# Print the results
cat("Aligned Sequence A:", as.character(alignedPattern(alignment)), "\n")
```

```
## Aligned Sequence A: AGCTGAACTAGCTAGCTGACTGACTGACTAGCT----AGCTGACTAGCTG
```

```
cat("Aligned Sequence B:", as.character(alignedSubject(alignment)), "\n")
```

```
## Aligned Sequence B: AGC-GAACTAGCTGACTGAC-GACTGACTAGCTGACTAGCTGACTAGC--
```

```
# Display alignment score
cat("Alignment Score:", alignment@score, "\n")
```

```
## Alignment Score: -4.528324
```