Preliminaries:

Before you begin, ensure you have done the following (instructions are contained within the slides):

- Open your script from yesterday, **problems.R** into RStudio.
- Load your session, **problems.RData** into RStudio.
- Set the working directory as the R_Course folder

Your dataset should contain the entire dataframe for colon_cancer_data_set.txt, and two subsets, one containing gene expression for the tumour (`affected`) samples and other containing data for the normal (`unaffected`) samples.

Note, you can use the `ls()` function to list the variables contained within your R session. Alternatively (or simultaneously), you can look at the **environment window** (top right) in RStudio, which should also list the contents of your session.

Plots can saved by using the Export option in the plotting window.

Note, I use the notation `df[]` to illustrate sample pieces of code. In these cases, `df` is just a placeholder representing any dataframe.

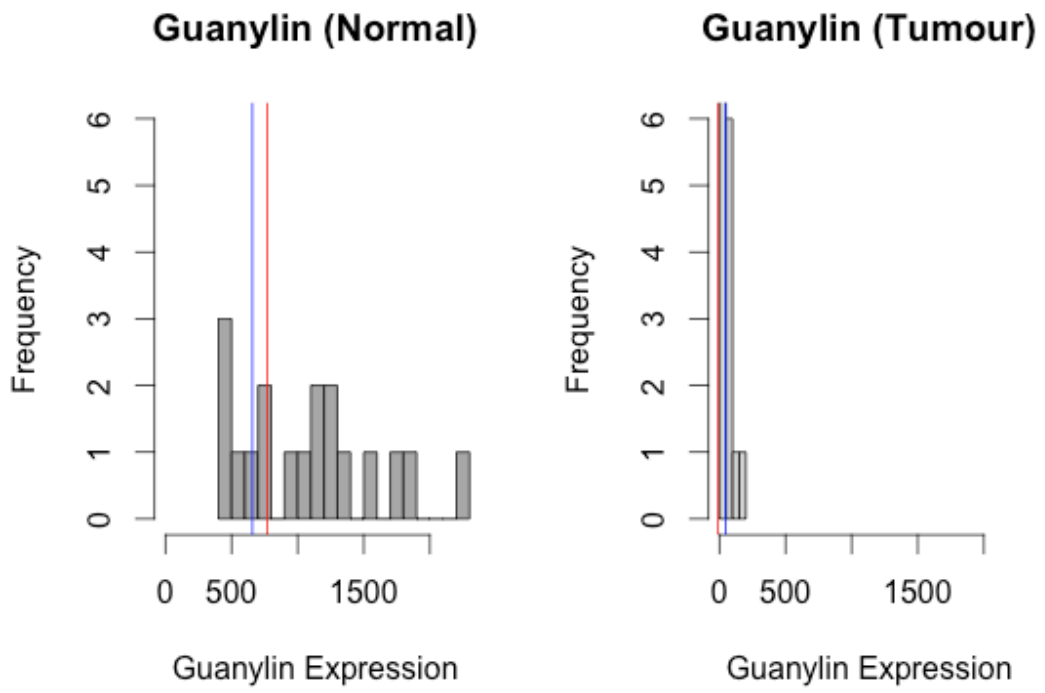
Q1. For the purpose of this analysis, we are interested in three genes, guanylin, pyrroline reductase and apolipoprotein A. These genes can be identified from the accessions M97496, M77836 and M10373, respectively.

1.1 For the affected data, make a new dataframe called `affected_genes` containing only expression for M97496, M77836 and M10373. (Hint: You can extract data from a data frame by indexing using the column or row names, e.g. `df[,c('name_1', 'name_2')]` for columns or `df[c('name_1','name_2')]` for rows.

1.2 For the unaffected data, do the same as above. Name the data frame `unaffected_genes`. Keep the order of genes the same for above.

1.3 Accession numbers are a bit annoying to keep track of. Replace the Accession number columns names with 'guanylin', 'pyrroline_reductase' and 'apolipoprotein_A'. (Hint: `names(df)` gives the column names of a dataframe. To replace, try `names(df) <- c('vector', 'of', 'new', 'names')`

Q2. Using the `par()` and `hist()` functions, create a panel of histograms for pyrroline reductase gene expression in tumour and normal samples. The aim is to achieve a plot similar to that for guanylin expression below.



2.1 Note, you can use the `main=` argument to specify a header.

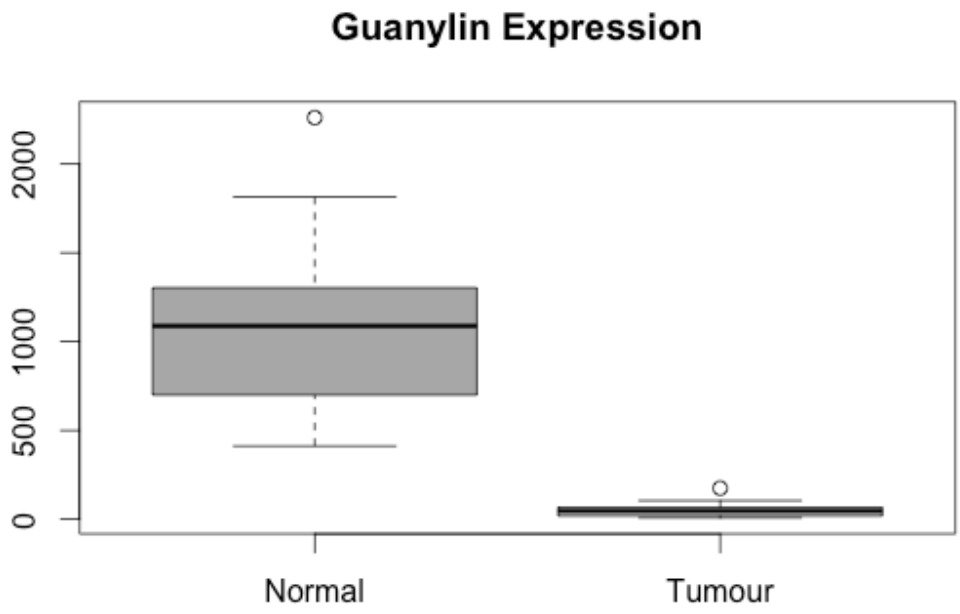
2.2 When comparing distributions it is useful fi they have the same range of values on their axes. To set the limits of the x-axis, you can use the `max()` and `min()` functions on the affected_genes and unaffected_genes and select a maximum and minimum that satisfies both data sets.

2.3 Use the `abline()` function to place lines for the mean and median (`median()`) gene expression for both histograms and colour them red and blue respectively.

2.4 Looking at the graphs you have made, determine an appropriate y-axis limit and set using the `ylim=` argument.

2.4 Experiment with `breaks=` argument within the `hist()` function to find a bin size that gives a nice histogram.

Q3. Using the `boxplot()` function, plot boxplots for the apolioprotein A gene in normal and tumour samples. Below is an example for guanylin.



3.1 Supply a vector of colours for the two boxplots.

3.2 Add a title and 'Normal', 'Tumour' labels to the plots.

Q4. Use the `plot()` function to create a scatterplot of Normal (x-axis) vs Tumour (y-axis) apolipoprotein expression. Experiment with the `pch=` and `cex=` arguments to make the plot appealing.

Notice, there is an outlier in the Normal data ruining your plot! Try the following:

4.1 Find which row the outlier occupies.

```
which(unaffected_genes[,3]==max(unaffected_genes[,3]),
      arr.ind=T)
```

This will tell you the position of that one outlier in the array.

4.2 Remake the plot by removing that outlier. (Hint: to ignore a point, the syntax is as follows `df[ ,3][-INDEX]` (or `df$gene[-INDEX])` where the index is from 4.1.

Q5. Once completed:

5.1 Save your script.

5.2 Save your session.

Note! Please ensure to save your work as we will use this data set continuously throughout the course.