**Lab 9 – Normality testing; confidence intervals**

**To submit: answers to all numbered questions. When the question asks you to write code or create graphs, submit the code and/or graphs in the Word document as part of your answer. Also submit a single .R file that contains all of your code.**

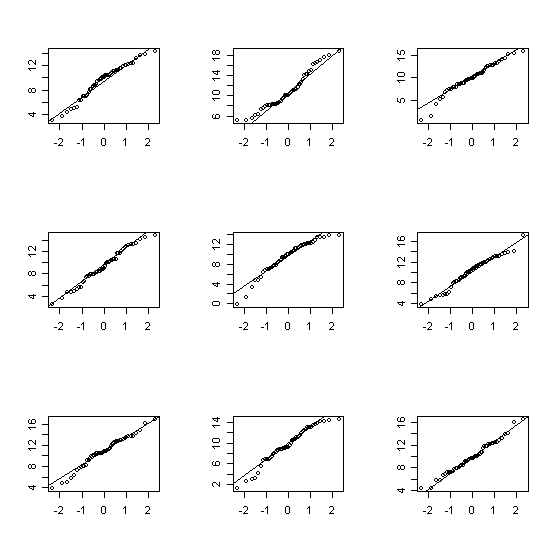
We saw in our last lab that sample means are distributed normally if the population follows a normal distribution, or if the sample is “large enough”. When this is the case, we can construct confidence intervals using the formulas we encountered in class.

In the examples we saw in class, we were told that the population was distributed normally. In real life, however, we often have to determine for ourselves whether this is the case. One way to do that is by producing a **quantile-quantile plot**, which we can do easily in R.

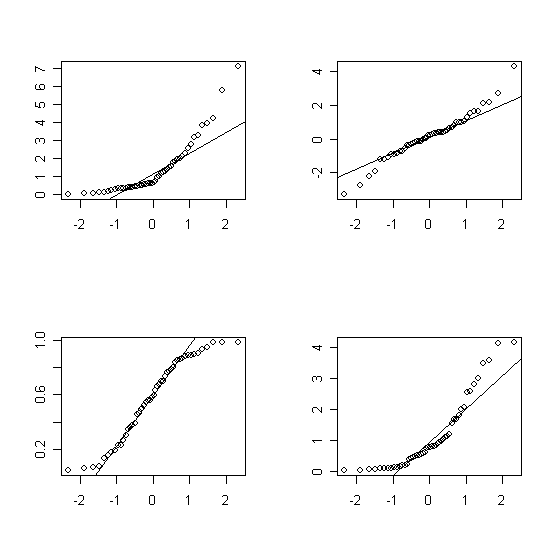
A quantile-quantile plot, or **QQ plot**, is a graph specifically designed to check for normality. The plot usually requires a minimum of 15 – 20 data points in order to accurately reveal normality.

If the data comes from a normal distribution the points should form a line with positive slope.

Examples of QQ plots of normal-ish data:



Examples of QQ plots of non-normal data:

We will do a few examples before delving into the theory.

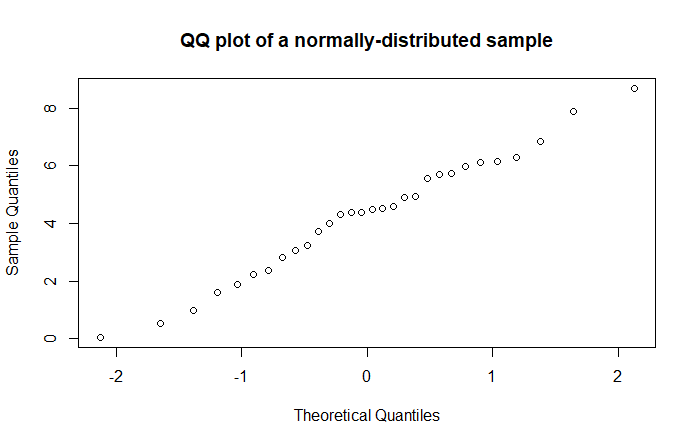
# Quantile-quantile plots of normal data

We will begin by generating a sample that comes from a normal distribution. We have generated similar samples before, using the **rnorm** function. Here are thirty data points that come from a normally-distributed dataset with mean 4 and standard deviation 2, though any mean and standard deviation will do:

> normaldata=rnorm(30,4,2)

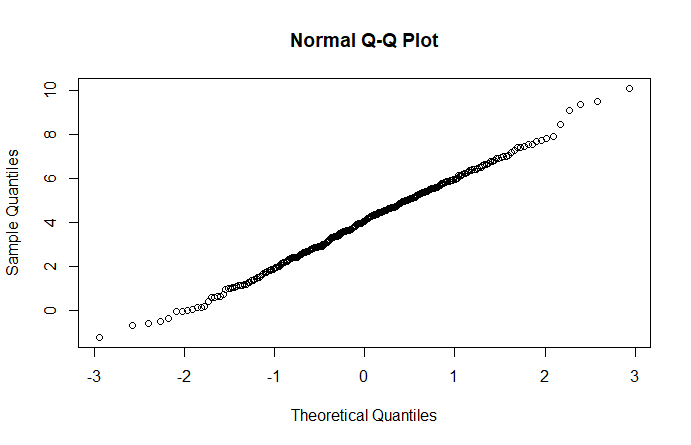
We then produce the QQ plot with the **qqnorm** function:

> qqnorm(normaldata, main="QQ plot of a normally-distributed sample")

The points roughly follow a line. If our sample were larger, we’d see a clearer pattern:

> normaldata=rnorm(300,4,2)

> qqnorm(normaldata)



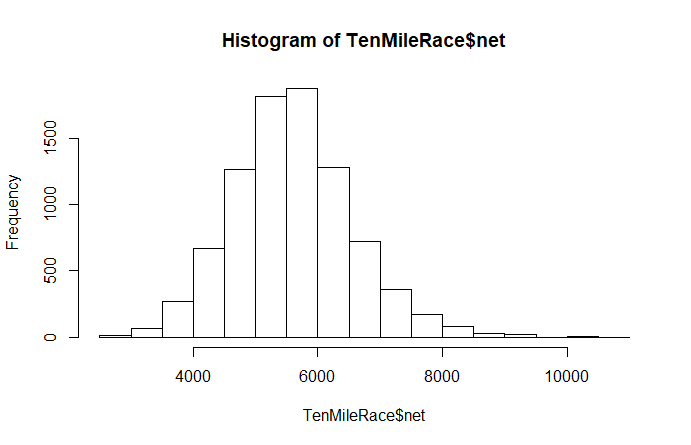
Note that the graph gets a bit weird at the edges. This is typical.

# QQ plots – a bit of theory

A QQ plot plots sample data against the z-scores that correspond to the percentiles of our data. If our data is normally distributed, then we would expect the same proportion of z-scores corresponding to the data as generated z-scores to be between *a* and *b*, for any numbers *a* and *b*. Graphically, this means that if our sample data came from a normally-distributed population, then the horizontal spacing between any two data points will be proportional to the vertical spacing, and our plot will be a straight line with positive slope. The less “normal” our sample data is, the more our plot will differ from being a straight line.

# Confidence intervals

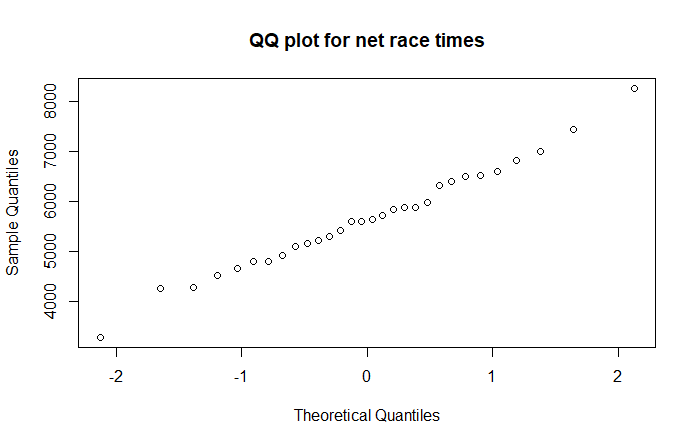
Let’s try this with some real data. Recall the dataset **TenMileRace** from Lab 1. Load this dataset (you will have to load the **mosaicData** library). Here is a histogram of net race times, which certainly appear to follow a normal distribution:



In real life, we often don’t have access to entire populations, or even large samples. When our samples are small, we need to check if they are normal before finding t-values to generate confidence intervals. For small samples, a QQ plot gives us a clearer picture of the distribution than a histogram. Since the population of net times is normally distributed, a sample of times will probably be close to normally distributed as well. Eg, here are 30 randomly-selected race times (the **sample** function gives us 30 indices, and then we retrieve the corresponding race times):

> racetimes30=TenMileRace$net[sample(1:8636, 30)]

> qqnorm(racetimes30, main="QQ plot for net race times")



It certainly appears that the sample of race times, like the population, is normally distributed.

(NOTE: your QQ plot will be different, and it might not look like it reflects a normal distribution. Unfortunately, this sort of thing happens occasionally: samples from normally distributed populations aren’t always normally distributed. Conversely, on occasion we have normally distributed samples that come from populations that are not normally distributed. You may have seen examples of both in Question 1. This sort of thing is why, in statistics, we never “prove” anything with complete certainty – we are only 90%, or 95%, or 99% confident of our conclusion. The other 10% or 5% or 1%, we just happened to get a sample that was not representative of the population. For instance, if the population is large and normally distributed, it’s possible (but not very likely) that through no fault of our own, it turned out that every number in our sample is more than two standard deviations from the mean. That said, if your QQ plot looks very different from a straight line, generate another sample of size 30 and get a QQ plot for it. It will probably look better than your last one.)

We saw in class that we can construct confidence intervals for means of small (n less than or equal to 30) data sets as long as the sample is normally distributed. The **t.test** function in R allows us to construct confidence intervals in a single line:

> t.test(racetimes30)

One Sample t-test

data: racetimes30

t = 35.126, df = 29, p-value < 2.2e-16

alternative hypothesis: true mean is not equal to 0

95 percent confidence interval:

5194.265 5836.535

sample estimates:

mean of x

5515.4

The highlighted line is the one we care about: it tells us that we are 95% sure that the mean net race time is between 5194.265 seconds and 5836.535 seconds.

We can find the mean net race time for the entire population:

> mean(TenMileRace$net)

[1] 5599.065

This value is indeed in the confidence interval.

1. Our theory tells us that 95% of the time, or 19 times out of 20, the true population mean will be contained in the 95% confidence interval. (The other 5% of the time, our sample happened to be “weird” – ie, with a distribution much different from that of the population.) Generate 20 different random samples of size 25 from the population of net race times and construct confidence intervals. How many of them contained the true mean?

Note again that this theory only works for small samples if they come from a normally-distributed population. If we compute a 95% confidence interval the same way for a small sample that is very abnormal, the range we get will not be a true 95% confidence interval. That is, it will not be the case that 95% of confidence intervals obtained with our method will contain the true population mean.

Load the **genotype** dataframe, which is part of the MASS library. Take a minute to read over its help file.

1. Filter your data to obtain four data sets, grouped by litter genotype. Note that these data sets are small, so we require them to be normally distributed if we wish to find a confidence interval using the methods we learned in class. Create QQ plots for weight for each genotype. Which of the four appears to be the most normal? Create a 95% confidence interval for the mean litter weight gain for rats of that genotype, and include a sentence interpreting your result.

Now load the **cats** dataframe.

1. Find 95% confidence intervals for body weight and heart weight for male and female cats. (ie, you will be finding four confidence intervals.) Note that these samples are large enough that we don’t have to worry about whether they are normally distributed. Include sentences interpreting your results. Based on your intervals, does there appear to be a significant difference between the heart weights of male cats versus female cats? How about body weights?
2. Go back to the **TenMileRace** dataframe and obtain 95% confidence intervals for the mean net race times for samples of size:
   1. 30
   2. 230
   3. 430

Include the commands you used, as well as sentences interpreting your conclusions.

Notice that the number of simulations, or equivalently the sample sizes, differed by 200 from part a) to b) and again from part b) to c). However, although the confidence intervals shrunk with each sample increase (if they didn’t, fix your code, because it means you did something wrong), they didn’t shrink by as much the second time around. That is: although larger samples give tighter estimates, there’s a point at which the improvement obtained by increasing the sample size isn’t worth the extra work. This is why we seldom see samples larger than a few hundred.

We can also generate the interpretations of the confidence interval automatically. The help file tells us that we can extract the individual values output by the **t.test** function:

**Value**

A list with class "htest" containing the following components:

|  |  |
| --- | --- |
| statistic | the value of the t-statistic. |
| parameter | the degrees of freedom for the t-statistic. |
| p.value | the p-value for the test. |
| conf.int | a confidence interval for the mean appropriate to the specified alternative hypothesis. |
| estimate | the estimated mean or difference in means depending on whether it was a one-sample test or a two-sample test. |
| null.value | the specified hypothesized value of the mean or mean difference depending on whether it was a one-sample test or a two-sample test. |
| alternative | a character string describing the alternative hypothesis. |
| method | a character string indicating what type of t-test was performed. |
| data.name | a character string giving the name(s) of the data. |

**conf.int** gives us the lower and upper bounds of the confidence interval in the form of a 1x2 array.

> t.test(racetimes30)$conf.int

[1] 5194.265 5836.535

attr(,"conf.level")

[1] 0.95

Notice that a confidence level of 0.95 is provided as well. This is the default confidence level for the **t.test** command. We will be using different confidence levels later.

We can obtain the bounds of the confidence interval individually:

> t.test(racetimes30)$conf.int[1]

[1] 5194.265

> t.test(racetimes30)$conf.int[2]

[1] 5836.535

If we don’t want to run the **t.test** command every time we want to extract a different output value, we can store the output in one variable:

> ttestRaceTimes=t.test(racetimes30)

> ttestRaceTimes$conf.int

[1] 5194.265 5836.535

attr(,"conf.level")

[1] 0.95

Now let’s construct a sentence interpreting the confidence interval. We want to get R to automatically return the sentence “We are 95% sure that the mean net race time is between 5194.265 seconds and 5836.535 seconds.” To do this, we need the bounds of the confidence interval as well as the confidence level.

We can easily obtain the bounds of the confidence interval:

> lowerbound=ttestRaceTimes$conf.int[1]

> upperbound=ttestRaceTimes$conf.int[2]

The confidence level is a bit tricker. Often, we will be specifying it directly when we call the **t.test** command. When we don’t, we can extract the default confidence level:

> attr(t.test(racetimes30)$conf.int, "conf.level")

[1] 0.95

Or, as a percent:

> conflevel=attr(ttestRaceTimes$conf.int, "conf.level")\*100

> conflevel

[1] 95

Now let’s put that together:

> CIsentence=cat("We are", conflevel, "percent sure that the mean race time is between", lowerbound, "and", upperbound, "seconds.")

We are 95 percent sure that the mean race time is between 5194.265 and 5836.535 seconds.

The next four questions will be using the **survey** data (which is part of the MASS library), which we saw in a previous lab.

1. Give a series of commands whose output is a sentence that interprets:
   1. A 95% confidence interval for the mean student age
   2. A 90% confidence interval for the mean student height
   3. A 90% confidence interval for the mean male student height (you’ll need to load a library and use a function that we last saw a few labs ago…)
2. Write a function **confidence(data, conflevel)** that reads in a list of data and a confidence level between 0 and 1, and returns a sentence interpreting the confidence interval. For example, if we want a 99% confidence interval for the mean student age, then the input

> confidence(survey$Age, 0.99)

would return

We are 99 percent sure that the mean is between 19.28241 and 21.46662 .

Note that your script returns a generic sentence that doesn’t include units, or even the quantity being measured (eg, in the case of the example above, the sentence gives no indication that our mean is an age measured in years).

1. Write a function that reads in a list of data, confidence level, information about the mean (ie, that we are measuring student ages) and units, and returns a detailed sentence. For example:

> confidence(survey$Age, 0.99, "student age", "years")

We are 99 percent sure that the mean student age is between 19.28241 and 21.46662 years .

Many commands in R allow users to leave some arguments empty and assigns default values in those cases. The **t.test** command is one such function: if the user doesn’t include a confidence level, R supplies a default of 0.95. Thus the commands t.test(survey$Age, conf.level=0.95) and t.test(survey$Age) give the same output:

One Sample t-test

data: survey$Age

t = 48.447, df = 236, p-value < 2.2e-16

alternative hypothesis: true mean is not equal to 0

95 percent confidence interval:

19.54600 21.20303

sample estimates:

mean of x

20.37451

1. Modify your **confidence** function so that if the user doesn’t specify a confidence level, a default of 0.95 will be applied.

> confidence(survey$Age, 0.90, "student age", "years")

We are 90 percent sure that the mean student age is between 19.68004 and 21.06899 years .

> confidence(survey$Age, 0.95, "student age", "years")

We are 95 percent sure that the mean student age is between 19.546 and 21.20303 years .

> confidence(survey$Age,, "student age", "years")

We are 95 percent sure that the mean student age is between 19.546 and 21.20303 years .

Notice the two commas in the middle of last command. Hint: you may find the **missing** command useful.