Statistics for a data rich world—some explorations*

Stefano Allesina (adapted by Lin Chen) *University of Chicago*

Goal: More and more often we need to analyze large and complex data sets. However, the statistical methods we've been taught in college have evolved in a data-poor world. Modern biology requires new tools, which can cope with the new questions and methods that arise in a data-rich world. Here we are going to discuss problems that often arise in the analysis of large data sets. We're going to review hypothesis testing (and what happens when we have many hypotheses) and discuss model selection. We're going to see the effects of selective reporting and p-hacking, and how they contribute to the *reproducibility crisis* in the sciences. **Audience:** Biologists with some programming background.

I. Review of hypothesis testing

Statistics is the science of collecting and analyzing data from samples in order to estimate and make inference regarding the parameters in the population. A sample is a smaller and random subset of the population of interest and the sample is collected to represent the population. Hypothesis testing is a major component of statistical inference.

The basic idea of hypothesis testing is the following: we have devised an hypothesis on our system, which we call H_1 (the "alternative hypothesis"). We have collected our data, and we would like to test whether the data are consistent (or inconsistent) with the so-called "null-hypothesis" (H_0), which is associated with a contradiction to the hypothesis we would like to prove.

The simplest example is that of a bent coin: my friend Aiden likes to bet on a coin toss with people, and he often chooses head and wins. I suspect that he has a bent coin in favor of head (H_1: the coin is bent). We therefore toss the coin several times and check whether the number of heads we observe is consistent with the null hypothesis of a fair coin (H_0: the coin is a fair coin).

In `R` we can toss many coins in no time at all. Call `p` the probability of obtaining a head, and initially toss a fair coin (`p = 0.5`) a thousand times:

First, when simulating a data, we always want to set seed to make sure the results are reproducible.

```
set.seed(222)
p <- 0.5 # probability of a head (fair coin)
flips <- 1000 # number of times we flip the coin
data <- sample(c("H", "T"),
                flips, prob = c(p, 1 - p),
                replace = TRUE)
heads <- sum(data == "H")
```

*This document is included as part of the Statistics for large data sets packet for the U Chicago BSD qBio6 boot camp, @MBL, 2020. **Current version**: 2020; **Corresponding author**: sallesina@uchicago.edu. Adapted by Lin Chen (lchen@health.bsd.uchicago.edu).

There is also a faster way to flip the coins and count the heads by assuming that the number of heads out of 1000 flips follows a binomial distribution:

```
heads <- rbinom(1, flips, p)
```

If the coin is fair, we expect approximately 500 heads, but of course we might have small variations due to the randomness of the process. We therefore need a way to distinguish between "bad luck" and an incorrect hypoythesis.

What is customarily done is to compute the probability of recovering the observed or a more extreme version of the pattern under the null hypothesis: if the probability is very small, it implies that when the null hypothesis is true, there is a very small probability (i.e., very unlikely) that you can observe things as or more extreme than what you have observed in the current data. We call this probability a *p-value*. A small *p-value* (typically less than 0.05) indicates strong evidence against the null hypothesis, so you reject the null hypothesis and conclude that the data is inconsistent with the null hypothesis. A large *p-value* (> 0.05) only means that when the null hypothesis is true, you are likely to observe what has been observed in the current data but that is not enough to prove the null hypothesis is true, so you fail to reject the null hypothesis and the conclusion is inconclusive. When the null hypothesis is indeed true or when the alternative hypothesis is true but you do not have enough power to reject the null, you may end up with a large p-value.

For example, if the coin is fair, the number of heads should follow the binomial distribution. The probability of observing a larger number of heads than what we've got is therefore

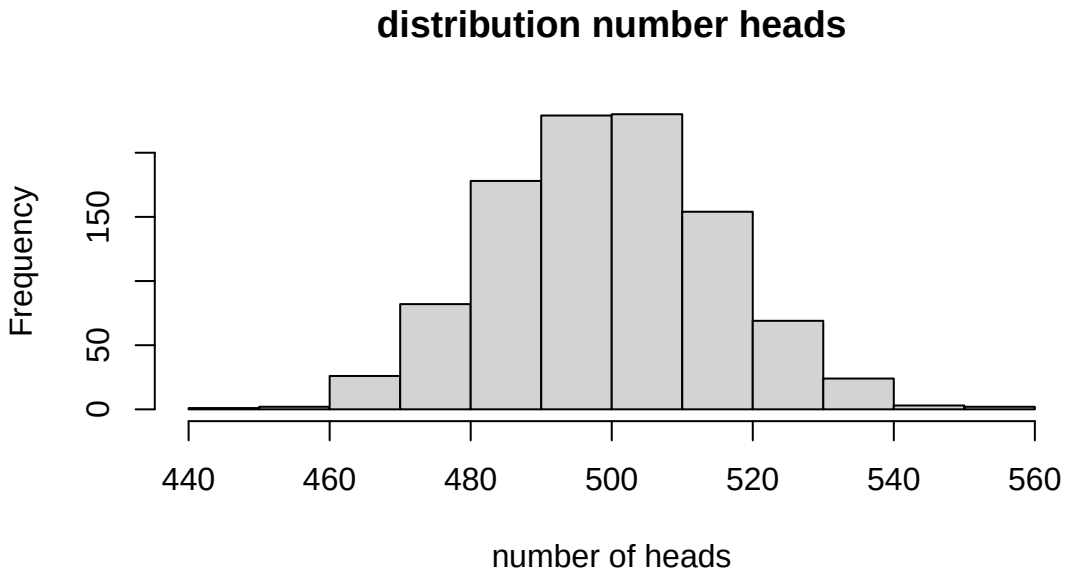
```
one.sided.pvalue <- 1 - pbinom(heads, flips, 0.5)
```

Here and in the following sections, we calculated a one-sided p-value testing $H_0 : p = 0.5$ versus $H_A : p > 0.5$. Note that in many other cases, we recommend a two-sided test, and a two-sided p-value together with a 95% confidence interval can be obtained via:

```
heads <- rbinom(1, flips, p)
pvalue <- binom.test(heads, flips, 0.5, alternative="two.sided")
```

What if we repeat the tossing many, many times?

```
# flip 1000 coins 1000 times, and count the number of heads in each of the 1000 experiments
# produce histogram of number of heads
heads_distribution <- rbinom(1000, flips, p)
hist(heads_distribution, main = "distribution number heads", xlab = "number of heads")
```



You can see that it is very unlikely to get more than 560 (or less than 440) heads when flipping a fair coin 1000 times. Therefore, if we were to observe say 400 heads (or 600), we would tend to believe that the coin is biased (though of course this could have happened by chance if we are repeating the tossing 1 trillion times!).

Type I and type II errors

When testing an hypothesis, we can make two types of errors:

- **Type I error**: reject H_0 when it is in fact true. Also known as false positive.
- **Type II error**: fail to reject H_0 when in fact it is not true. Also known as false negative.

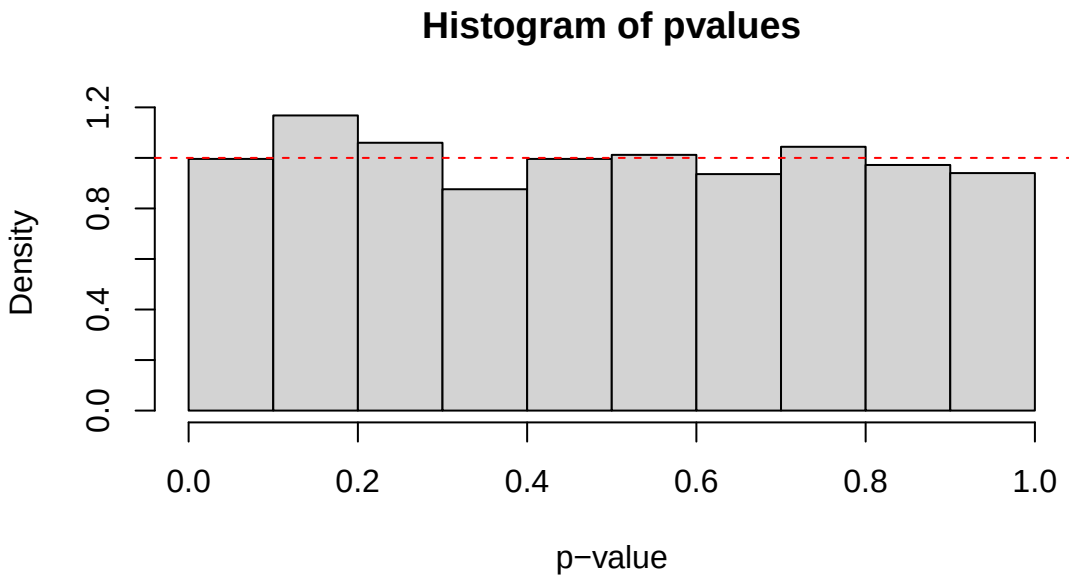
We call α the probability of making a type I error (or type I error rate), and β as type II error rate. And power is in fact $1 - \beta$. We can calculate the p-value based on the data and compare the p-value with the significance threshold α. The p-value quantifies how strongly the data contradicts the null hypothesis, and if $p < \alpha$, we reject the null hypothesis. Type I and Type II error rates are inversely related. In choosing a significance level for a test, you are actually deciding how much you want to risk committing a type I error — rejecting the null hypothesis when it is. The more stringent α is (0.01 versus 0.05) in controlling type I error rate, the less likely you would make a rejection decision and consequently the power will be reduced too.

The distribution of p-values

Suppose that we are tossing each of several fair coins 1000 times. For each, we compute the corresponding (one-sided) p-value testing the null hypothesis `p = 0.5` against the alternative `p > 0.5`. How are the p-values distributed?

```
ncoins <- 2500
heads <- rbinom(ncoins, flips, p)
pvalues <- 1-pbinom(heads, flips, 0.5)
```

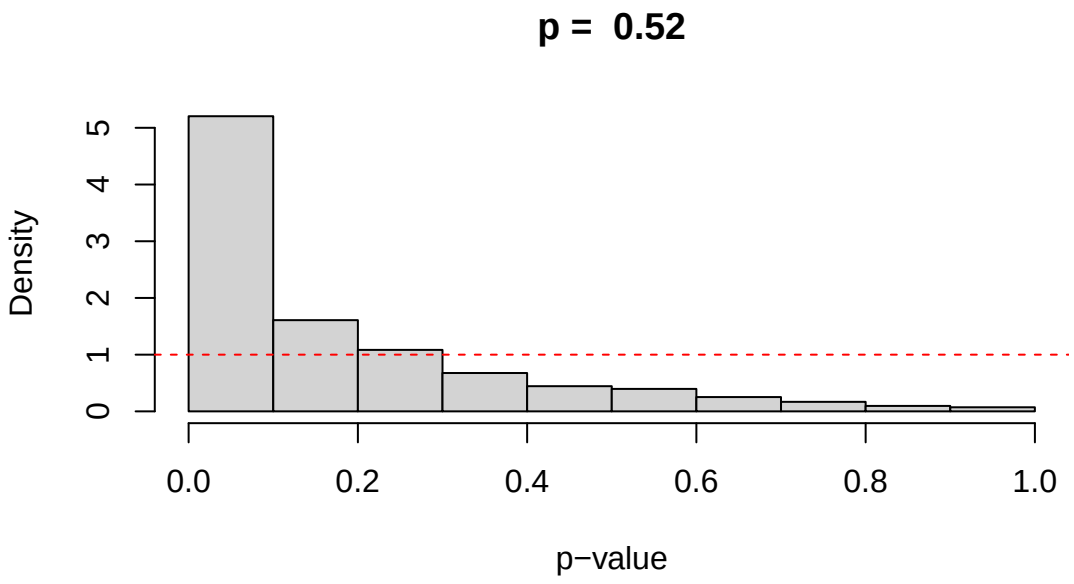
```r
hist(pvalues, xlab = "p-value", freq = FALSE)
abline(h = 1, col = "red", lty = 2)
```



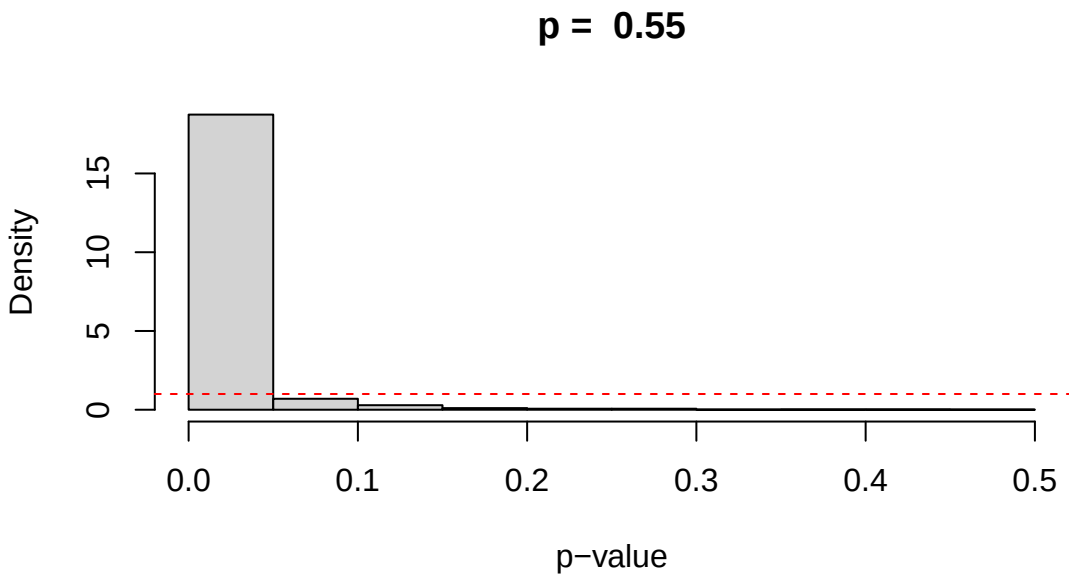
As you can see, if the data were generated under the null hypothesis, the distribution of the p-values would be approximately uniform between 0 and 1. This means that if we set $\alpha = 0.05$, we would reject the null hypothesis 5% of the time (even though in this case we know the hypothesis is correct!).

What is the distribution of the p-values if we are tossing biased coins? We will find an enrichment in small p-values, with stronger effects for larger biases:

```r
p <- 0.52 # the coin is biased
heads <- rbinom(ncoins, flips, p)
pvalues <- 1 - pbinom(heads, flips, 0.5)
hist(pvalues, xlab = "p-value", main = paste0("p =  ", p), freq = FALSE)
abline(h = 1, col = "red", lty = 2)
```




```r
p <- 0.55 # the coin is biased
heads <- rbinom(ncoins, flips, p)
pvalues <- 1 - pbinom(heads, flips, 0.5)
hist(pvalues, xlab = "p-value", main = paste0("p =  ", p), freq = FALSE)
abline(h = 1, col = "red", lty = 2)
```



II. The challenges with p-values

Selective reporting

Articles reporting positive results are easier to publish than those containing negative results. Authors might have little incentive to publish negative results, which could go directly into the file-drawer.

This tendency is evidenced in the distribution of p-values in the literature: in many disciplines, one finds a sharp decrease in the number of tests with p-values just below 0.05 (which is customarily–and arbitrarily–chosen as a threshold for "significant results"). For example, we find many a sharp decrease in the number of reported p-values of 0.051 compared to 0.049–while we expect the p-value distribution to decrease smoothly.

Selective reporting leads to irreproducible results: we always have a (small) probability of finding a "positive" result by chance alone. For example, suppose we toss a fair coin many times, until we find a "signficant" result.

On the other hand, more and more journals would require us to report effect size estimates and confidence intervals – "Were this procedure to be repeated on numerous samples, the fraction of calculated confidence intervals (which would differ for each sample) that encompass the true population parameter would tend toward 90%."(source: Cox D.R., Hinkley D.V. (1974) Theoretical Statistics, Chapman & Hall, p49, p209.)

Problem: p-hacking

The problem is well-described by Simonsohn et al. (J. Experimental Psychology, 2014): "While collecting and analyzing data, researchers have many decisions to make, including whether to collect more data, which method to use, which measure(s) to analyze, which covariates to use, what to do with outliers and missing data, and so on. If these decisions are not made in advance but rather are made as the data are being analyzed, then researchers may make them in ways that self-servingly increase their odds of publishing. Thus, rather than placing entire studies in the file-drawer, researchers may file merely the subsets of analyses that produce nonsignificant results. We refer to such behavior as *p-hacking*." The term p-hacking describes the conscious or subconscious manipulation of data in a way that produces a desired p-value.

The same authors showed that with careful p-hacking, almost anything can become significant (read their hylarious article in Psychological Science, where they show that listening to a song can change the listeners' age!).

Discussion on p-values

Selective reporting and p-hacking are only two of the problems associated with the widespread use and misuse of p-values. The discussion in the scientific community on this issue is extremely topical. I have collected some of the articles on this problem in the `readings` folder. Importantly, in 2016 the American Statistical Association released a statement on p-values every scientist should read.

Reproducibility crisis

P-values and hypothesis testing contribute considerably to the so-called *reproducibility crisis* in the sciences. A survey promoted by *Nature* magazine found that "More than 70% of researchers have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments."

This problem is due to a number of factors, and addressing it will likely be one of the main goals of science in the next decade.

Exercise: p-hacking

Go to goo.gl/a3UOEF and try your hand at p-hacking, showing that your favorite party is good (bad) for the economy.

III. Multiple comparisons (also known as multiple testing)

The problem of multiple comparisons arises when we perform multiple statistical tests. Since each test is subject to some small chance of producing a false positive result, when jointly considering many many tests, the chances of producing some false positive findings are much higher.

Suppose we perform our coin tossing exercise, flipping 1000 coins 1000 times each. For each coin, we determine whether our data differs significantly from what expected by contrasting our p-value with a significance level $\alpha = 0.05$.

Even if the coins are all perfectly fair, we would expect to find approximately $0.05 \cdot 1000 = 50$ coins that lead to the rejection of the null hypothesis.

In fact, we can calculate the probability of making at least one type I error (reject the null when in fact it is true). This probability is called the Family-Wise Error Rate (FWER). It can be computed as 1 minus the probability of making no type I error at all. If we set $\alpha = 0.05$, and assume the tests to be independent, the probability of making no errors in m tests is $1 - (1 - 0.05)^m$. Therefore, if we perform 10 tests, we have about 40% probability of making at least a mistake; if we perform 100 tests, the probability grows to more than 99%. If the tests are not independent, we can still say that in general $FWER \leq m\alpha$.

This means that setting an α per test does not control for FWER.

Moving from tossing coins to biology, consider the following examples:

- **Gene expression** In a typical RNAseq experiment, we compare the differential expression levels of tens of thousands of genes in the treatment and control tissues.
- **GWAS** In Genome-Wide Association Studies we want to find single-nucleotide polymorphisms (SNPs) associated with a given phenotype. It is common to test millions of SNPs for signficant associations.
- **Identifying binding sites** Identifying candidate binding sites for a transcriptional regulator requires scanning the whole genome, yielding tens of millions of tests.

Organizing the tests in a table

Suppose that we're testing m hypotheses. Of these, an unknown subset m_0 is true, while the remaining $m_1 = m - m_0$ are false. We would like to correctly call the true/false hypotheses (as much as possible). We can summarize the results of our tests in a table, of which the elements are unobservable:

	not rejected	rejected
H is True	U (true negatives)	V (false positives)
H is False	T (false negatives)	S (true positives)

What we would like to know is $m_1 = T + S$ and $m_0 = U + V$. Then V is the number of type I errors (rejected H when in fact it is true), and T is the number of type II errors (failed to reject a false H). However, we can only observe $V + S$ (the number of "discoveries"), and $U + T$ (number of "failures").

The type I error rate is $E[V]/m$ (where $E[X]$ stands for expectation). When there are many tests (m) being considered, controlling for type I error rates at 0.05 means that by random chance, one could make $0.05 \times m$ false positive findings even if all m tests are under the null. For example, when testing genetic associations between 10 million genetic variants to the risk of a disease, if still using 0.05 as the significance threshold, there could be 500k significant findings by random chance even if the disease is not heritable and has no genetic association. Apparently, we need to choose a much more strigent significance threshold to account for the number of tests, and there are some other error measures. The Family-wise error rate is defined as $P(V > 0)$. Another quantity of interest is the False Discovery Rate (FDR), measured as the proportion of true discoveries $FDR = E[V/(V + S)]$ when $V + S > 0$. FDR measures the proportion of falsely rejected hypotheses.

Importantly, FWER guards against any single false positive finding among all m tests, and is a more stringent significance criteria than FDR in multiple comparison problems. It also means that when we control for FWER, we're automatically controlling for the FDR, but not vice versa. It should be noted that when a large number of tests is performed, controlling FWER could be quite conservative and may lose power.

Methods for multiple testing correction: Bonferroni correction

One of the simplest and most widely-used procedures to control for FWER is Bonferroni's correction. This procedure controls for FWER in the case of independent or dependent tests. It is typically quite conservative, especially when the tests are not independent (in practice, it becomes "too conservative" when the number of tests is moderate to high). Fundamentally, for a desired FWER α we choose as a significance treshold of α/m for each single test, where m is the number of tests we're performing. Equivalently, we can "adjust" the p-values as $q_i = \min(m \cdot p_i, 1)$, and call significant the values $q_i < \alpha$. In `R` it is easy to perform this correction:

```
original_pvals <- c(0.012, 0.06,  0.77, 0.001, 0.32)
adjusted_pvals <- p.adjust(original_pvals, method = "bonferroni")
print(adjusted_pvals)
```

```
## [1] 0.060 0.300 1.000 0.005 1.000
```

With these adjusted p-values, and an $\alpha = 0.05$, we would still single out as significant the fourth test, but not the first. The strength of Bonferroni is its simplicity, and the fact that we can perform the operation in a single step. Moreover, the order of the tests does not matter.

Other procedures: the Holm–Bonferroni method (or the Holm method)

There are several refinements of Bonferroni's correction, some of which use the sequence of ordered p-values. For example, the Holm method starts by sorting the p-values in increasing order $p_{(1)} \leq p_{(2)} \leq p_{(3)} \leq \dots p_{(m)}$. The hypothesis $H_{(i)}$ is rejected if $p_{(j)} \leq \alpha/(m - j + 1)$ for all

$j = 1, \ldots, i$. Equivalently, we can adjust the p-values as $q_{(i)} = \min(1, \max((m - i + 1)p_{(i)}, q_{(i-1)}))$. In this way, we use the most stringent threshold to determine whether the smallest p-value is significant, the next smallest p-value uses a slightly higher threshold and so on. The Holm method is uniformly more powerful than the Bonferroni correction.

For example, using the same p-values above:

```
original_pvals <- c(0.012, 0.06,  0.77, 0.001, 0.32)
adjusted_pvals <- p.adjust(original_pvals, method = "holm")
print(adjusted_pvals)
```

```
## [1] 0.048 0.180 0.770 0.005 0.640
```

We see that we would be calling the first test significant, contrary to what obtained with Bonferroni.

The function `p.adjust` offers several choices for p-value correction. Also, the package `multcomp` provides a quite comprehensive set of functions for multiple hypothesis testing.

An example: testing mixed coins

We're going to test these concepts by tossing repeatedly many coins. In particular, we're going to toss 1000 times 50 biased coins ($p = 0.55$) and 950 fair coins ($p = 0.5$). For each coin, we're going to compute a p-value, and count the number of type I, type II, etc. errors when using unadjusted p-values as well as when correcting using the Bonferroni or Holm procedure.

```
toss_coins <- function(p, flips){
  # toss a coin with probability p of landing on head several times
  # return a data frame with p, number of heads, pval and
  # H0 = TRUE if p = 0.5 and FALSE otherwise
  heads <- rbinom(1, flips, p)
  pvalue <- 1 - pbinom(heads, flips, 0.5)
  if (p == 0.5){
    return(data.frame(p = p, heads = heads, pval = pvalue, H0 = TRUE))
  } else {
    return(data.frame(p = p, heads = heads, pval = pvalue, H0 = FALSE))
  }
}

# To ensure everybody gets the same results, we're setting the seed
set.seed(8)
data <- data.frame()
# the biased coins
for (i in 1:50) data <- rbind(data, toss_coins(0.55, 1000))
# the fair coins
for (i in 1:950) data <- rbind(data, toss_coins(0.5, 1000))
# here's the data structure
head(data)
```

```
##      p heads         pval    H0
## 1 0.55   535 1.235282e-02 FALSE
## 2 0.55   558 1.061983e-04 FALSE
## 3 0.55   567 9.546428e-06 FALSE
## 4 0.55   532 1.988964e-02 FALSE
## 5 0.55   547 1.322765e-03 FALSE
## 6 0.55   574 1.178147e-06 FALSE
```

Now we write a function that adjusts the p-values and builds the table above

```
get_table <- function(data, adjust, alpha = 0.05){
  # produce a table counting U, V, T and S
  # after adjusting p-values for multiple comparisons
  data$pval.adj <- p.adjust(data$pval, method = adjust)
  data$reject <- FALSE
  data$reject[data$pval.adj < alpha] <- TRUE
  return(table(data[,c("reject","H0")]))
}
```

First, let's see what happens if we don't adjust the p-values:

```
no_adjustment <- get_table(data, adjust = "none", 0.05)
print(no_adjustment)
```

```
##        H0
## reject  FALSE TRUE
##   FALSE     2  905
##   TRUE     48   45
```

We correctly declared 48 of the biased coins "significant", but we also incorrectly called 2 biased coins "not significant" (Type II error). More worringly, we called 45 fair coins biased when they were not (Type I error). To control for the family-wise error rate, we can correct using Bonferroni:

```
bonferroni <- get_table(data, adjust = "bonferroni", 0.05)
print(bonferroni)
```

```
##        H0
## reject  FALSE TRUE
##   FALSE    40  950
##   TRUE     10    0
```

With this correction, we dramatically reduced the number of type I errors (from 45 to 0), but at the cost of increasing type II errors (from 2 to 40) and losing power. In this way, we would make only 10 discoveries instead of 50.

In this case, Holm's procedure does not help:

```
holm <- get_table(data, adjust = "holm", 0.05)
print(holm)
```

```
##        H0
## reject  FALSE TRUE
##   FALSE    40  950
##   TRUE     10    0
```

More sophisticated methods, for example the Benjamini-Hochberg (BH) procedure based on controlling false discovery rate (FDR=E(false discoveries/signifiant tests), where E stands for expectation), can reduce the type II errors and improve power, at the cost of a few and estimable type I errors:

```
BH <- get_table(data, adjust = "BH", 0.05)
print(BH)
```

```
##        H0
## reject  FALSE TRUE
##   FALSE    17  946
##   TRUE     33    4
```

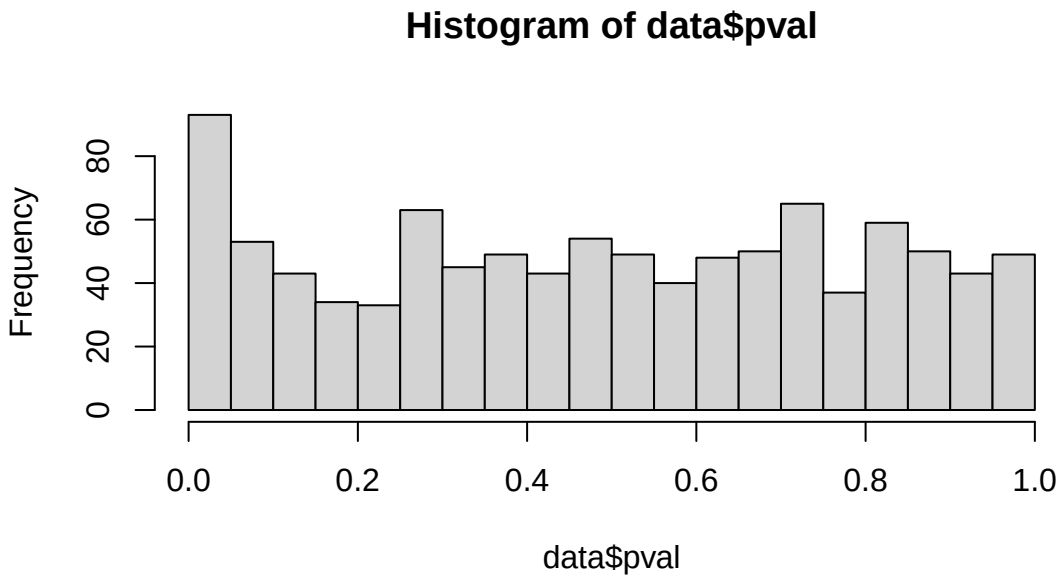
FDR and q-values

Inspired by the need for controlling for FDR in genomics, Storey and Tibshirani (PNAS 2003) have proposed the idea of a q-value, measuring the probability that a feature that we deemed significant turns out to be not significant after all.

One uses p-values to control for the false positive rate (# false positive / total test): when determining significant p-values we control for the rate at which null features in the data are called significant. The False Discovery Rate (# false positive / # signficant test), on the other hand, measures the rate at which results that are deemed significant are truly null. While setting $PCER = 0.05$ we are stating that about 5% of the truly null features will be called significant, an $FDR = 0.05$ means that among the features that are called significant, about 5% will turn out to be null.

They proposed a method that uses the ensamble of p-values to determine the approximate (local) FDR. The idea is simple. If you plot your histogram of p-values when you have few true effect, and many nulls, you will see something like:

```
hist(data$pval, breaks = 25)
```



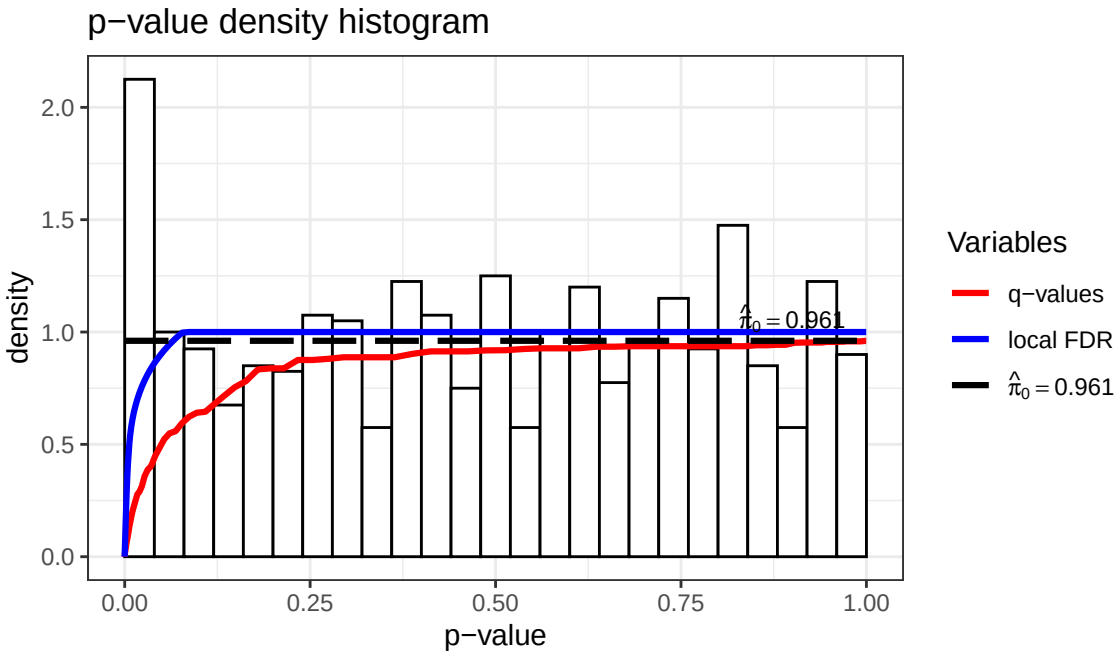
where the right side of the histogram is close to a uniform distribution. We could use the high p-values to find how tall the histogram would be if all effects were null, thereby estimating the proportion of truly null features $\pi_0 = m_0/m$.

Storey has built an `R`-package for this type of analysis:

```
# To install:
#install.packages("devtools")
#library("devtools")
#install_github("jdstorey/qvalue")
library("qvalue")
qobj <- qvalue(p = data$pval)
```

Here's the estimation of the π_0

```
hist(qobj)
```



which is quite good (in this case we know that $\pi_0 = 0.95$). The small p-values under the dashed line represent our false discoveries. Even better, through randomizations one can associate a q-value to each test, representing the probability of making a mistake when calling a result significant (formally, the q-value is the minimum FDR that can be attained when calling that test significant).

For example:

```
table((qobj$pvalues < 0.05) & (qobj$qvalues < 0.05), data$H0)
```

```
##
##          FALSE TRUE
##   FALSE     17  946
##   TRUE      33    4
```

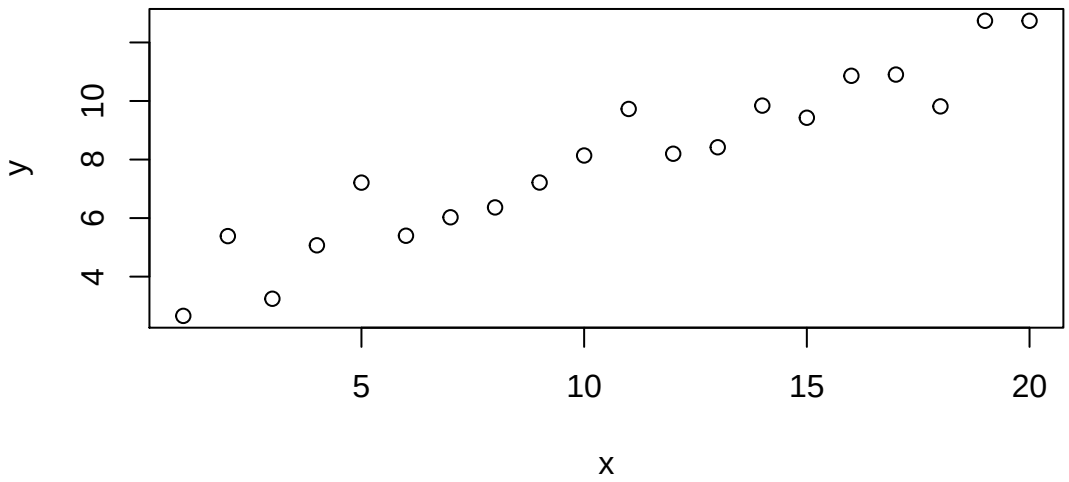
Note that the estimation of FDR is unstable if the demoniator (# significant test) is expected to be small. Therefore, you may notice that FDR was widely used in detecting differentially expressed genes in diseased versus normal samples where the expected number of non-null tests is large. In contast, in GWAS, researchers use the Bonferroni-adjusted p-value threshold of 5×10^{-8} to declare significance.

IV. Linear regression, logistic regression, and model selection

Linear regression

In statistics, linear regression is an approach to model the linear relationship between one response variable (or dependent variable) and one or more explanatory variables (or independent variables, or predictor). If there is only one exlanatory variable, it is called simple linear regression. If the linear regression involves more than one explanatory variable, it is a multiple linear regression.

```r
# create fake data for a simple linear regression
set.seed(5)
x <- 1:20
y <- 3 + 0.5 * x + rnorm(20)
plot(y ~ x)
```

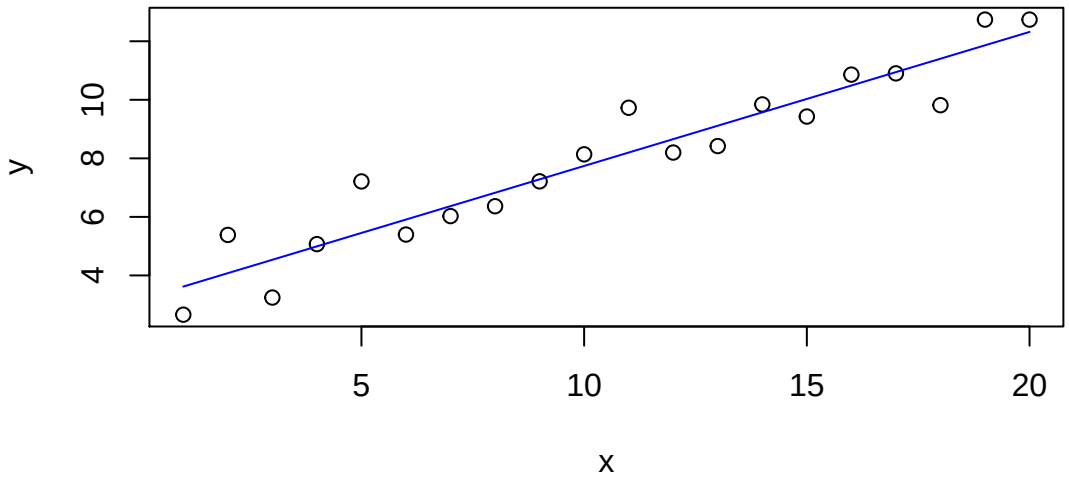


We can fit a simple linear regression to the data

```r
model1 <- lm(y ~ x)
summary(model1)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.58733 -0.53439 -0.05563  0.40359  1.76021
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.16195    0.42731    7.40 7.31e-07 ***
## x            0.45786    0.03567   12.84 1.70e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9199 on 18 degrees of freedom
## Multiple R-squared:  0.9015, Adjusted R-squared:  0.896
## F-statistic: 164.8 on 1 and 18 DF,  p-value: 1.695e-10
```

```r
plot(y~x)
points(model1$fitted.values~x, type = "l", col = "blue")
```



In a data rich world, often, we need to select a model out of a set of reasonable alternatives with different combinations and/or (even nonlinear) patterns of explanatory variables. However, we run the risk of overfitting the data (i.e., fitting the noise as well as the pattern). The best fitted model for the current data from the current sample may not be the best model representing the pattern in the population of interest. Here is a simple example of a overfitted regression:

For the above data, we can also fit a more complex polynomial function of x.

```
model2 <- lm(y ~ poly(x, 7))
```

Let's see the residuals etc.

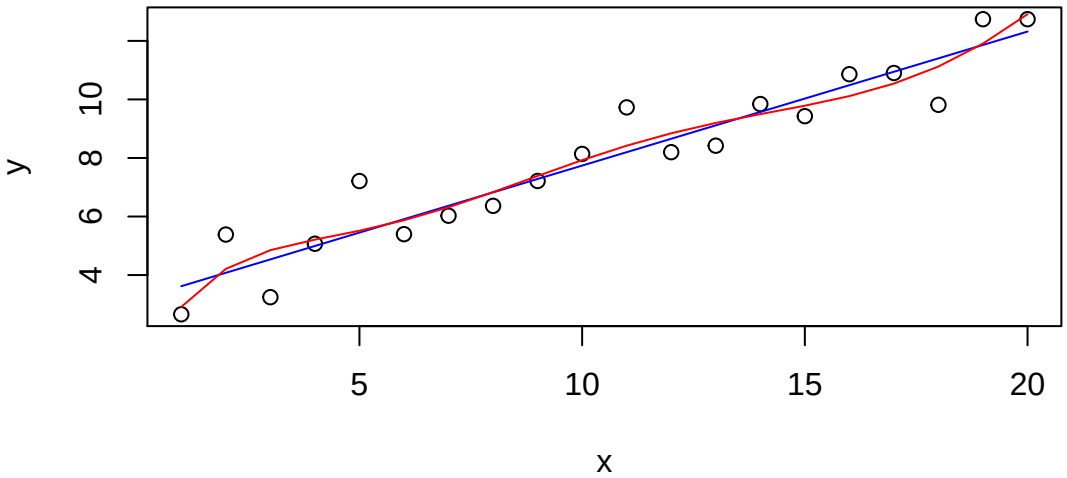
```
summary(model1)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.58733 -0.53439 -0.05563  0.40359  1.76021
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.16195    0.42731    7.40 7.31e-07 ***
## x            0.45786    0.03567   12.84 1.70e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9199 on 18 degrees of freedom
## Multiple R-squared:  0.9015, Adjusted R-squared:  0.896
## F-statistic: 164.8 on 1 and 18 DF,  p-value: 1.695e-10
```

```r
summary(model2)
```

```
##
## Call:
## lm(formula = y ~ poly(x, 7))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6066 -0.4714 -0.1719  0.4577  1.6960
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.96944    0.23826  33.448 3.24e-13 ***
## poly(x, 7)1 11.80701    1.06553  11.081 1.17e-07 ***
## poly(x, 7)2 -0.13536    1.06553  -0.127    0.901
## poly(x, 7)3  0.82397    1.06553   0.773    0.454
## poly(x, 7)4  0.28455    1.06553   0.267    0.794
## poly(x, 7)5  0.81336    1.06553   0.763    0.460
## poly(x, 7)6 -0.40726    1.06553  -0.382    0.709
## poly(x, 7)7  0.03159    1.06553   0.030    0.977
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.066 on 12 degrees of freedom
## Multiple R-squared:  0.9119, Adjusted R-squared:  0.8605
## F-statistic: 17.74 on 7 and 12 DF,  p-value: 1.917e-05
```

```r
plot(y~x)
points(model1$fitted.values~x, type = "l", col = "blue")
points(model2$fitted.values~x, type = "l", col = "red")
```



Our second model has a much greater R^2, i.e., more variation in the response variable can be explained by the model, but the second model also has many more parameters. The first model is more parsimonious. Which is a model we should choose?

Model selection tries to address this and similar problems. Most model fitting and model selection procedures are based on likelihoods (e.g., Bayesian models, maximum likelihood, minimum description length). The likelihood $L(\theta|D, M)$ is (proportional to) the probability of observing the data D under the model M and parameters θ. Because likelihood can be very small when you have much data, typically one works with log-likelihoods. For example:

```
logLik(model1)
```

```
## 'log Lik.' -25.65474 (df=3)
```

```
logLik(model2)
```

```
## 'log Lik.' -24.54 (df=9)
```

Typically, more complex models will yield better (less negative) log-likelihoods and a better fit for the current data. However, more parameters would also increase the variation of the model. A complex model may not best represent the population (not to say computational burden). We will need to find a balance between bias and variance. We therefore want to penalize more complex models in some way.

AIC

One of the simplest methods to select among competing models is the Akaike Information Criterion (AIC). It penalizes models accoding to the **number of parameters**: $AIC = 2p - 2\log L(\theta|D, M)$, where p is the number of parameters. Note that **smaller** values of AIC stand for "better" models. In `R` you can compute AIC using:

```
AIC(model1)
```

```
## [1] 57.30948
```

```
AIC(model2)
```

```
## [1] 67.08001
```

As you can see, AIC would favor the first (and simpler) model, which is also the model we used to simulate the data.

AIC is rooted in information theory and measures (asymptotically) the loss of information when using the model instead of the data. There are several limitations of AIC: a) it only holds asymptotically (i.e., for very large data sets; for smaller data you need to correct it); it penalizes each parameter equally (i.e., parameters that have a large influence on the likelihood have the same weight as parameters that do no influence the likelihood much); it can lead to overfitting, favoring more complex models in simulated data generated by simpler models.

BIC

In a similar vein, BIC (Bayesian Information Criterion) uses a slightly different penalization: $BIC = \log(n)p - 2\log L(\theta|D, M)$, where n is the number of data points. You may see that for large data, BIC penalizes a complex model more than AIC. Again, smaller values stand for "better" models:

```
BIC(model1)
```

```
## [1] 60.29667
```

```
BIC(model2)
```

```
## [1] 76.0416
```

Here in this simple example, AIC and BIC agree with each other, and the simpler model is favored.

Logistic regression

Logistic regression is often used to model the nonlinear relationship (modeled by a logistic function) between a binary response variable and a linear combination of explantory variables. When the outcome is binary, for example pass/fail, win/lose, alive/dead, yes/no, one would consider a logistic regression. It can be extended to model several classes of a categorical variable as response.

We will start with an example. Fox et al. (Research Integrity and Peer Review, 2017) analyzed the invitations to review for several scientific journals, and found that "The proportion of invitations that lead to a submitted review has been decreasing steadily over 13 years (2003–2015) for four of the six journals examined, with a cumulative effect that has been quite substantial". Their data is stored in `../data/FoxEtAl.csv`. We're going to build models trying to predict whether a reviewer will agree (or not) to review a manuscript.

```
# read the data
reviews <- read.csv("../data/FoxEtAl.csv", sep = "\t")
# take a peek
head(reviews)
```

```
##   Sort   Journal  msID Year ReviewerID ReviewerInvited ReviewerResponded
## 1    1 Evolution 34152 2007    8426852               1                 1
## 2    2 Evolution 34152 2007    8425970               1                 1
## 3    3 Evolution 34152 2007    8425116               1                 1
## 4    4 Evolution 34152 2007    8426128               1                 1
## 5    5 Evolution 34152 2007    9327585               1                 1
## 6    6 Evolution 34152 2007    8423528               1                 1
##   ReviewerAgreed ReviewerAssigned ReviewSubmitted
## 1              0                0              NA
## 2              0                0              NA
```

```
## 3              0                 0              NA
## 4              0                 0              NA
## 5              1                 1               1
## 6              0                 0              NA
```

```
# set NAs to 0
reviews[is.na(reviews)] <- 0
# how big is the data?
dim(reviews)
```

```
## [1] 113876     10
```

```
# that's a lot! Let's take 5000 review invitations for our explorations;
# we will fit the whole data set later
set.seed(101)
small <- reviews[order(runif(nrow(reviews))),][1:5000,]
```

The response variable of interest is a reviewer i agreeing to review a manuscript or not, and is binary; and so we decided to use a logistic regression. Call π_i the probability that a reviewer i agree to review a manuscript. We model $logit(\pi_i) = \log(\pi_i/(1-\pi_i))$ as a linear function.

Constant rate

As a null model we build a model in which the probability of agreeomg to review does not change in time/for journals:

```
# suppose the rate at which reviewers agree is a constant
mean(small$ReviewerAgreed)
```

```
## [1] 0.466
```

```
# fit a logistic regression
model_null <- glm(ReviewerAgreed~1, data = small, family = "binomial")
summary(model_null)
```

```
##
## Call:
## glm(formula = ReviewerAgreed ~ 1, family = "binomial", data = small)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.120  -1.120  -1.120   1.236   1.236
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept) -0.13621    0.02835  -4.805 1.55e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6908.3  on 4999  degrees of freedom
## Residual deviance: 6908.3  on 4999  degrees of freedom
## AIC: 6910.3
##
## Number of Fisher Scoring iterations: 3
```

```
# interpretation:
exp(model_null$coefficients[1]) / (1 + exp(model_null$coefficients[1]))
```

```
## (Intercept)
##       0.466
```

Declining trend

We now build a model in which the probability to review declines steadily from year to year:

```
# Take 2003 as baseline
model_year <- glm(ReviewerAgreed~I(Year - 2003), data = small, family = "binomial") #The I() fun
summary(model_year)
```

```
##
## Call:
## glm(formula = ReviewerAgreed ~ I(Year - 2003), family = "binomial",
##     data = small)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.3441  -1.0879  -0.9686   1.2372   1.4017
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)     0.383697   0.065482   5.860 4.64e-09 ***
## I(Year - 2003) -0.074753   0.008491  -8.804  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6908.3  on 4999  degrees of freedom
## Residual deviance: 6829.7  on 4998  degrees of freedom
```

```
## AIC: 6833.7
##
## Number of Fisher Scoring iterations: 4
```

Journal dependence

Reviewers might be more likely to agree for more prestigious journals:

```
# Take the first journal as baseline
model_journal <- glm(ReviewerAgreed~Journal, data = small, family = "binomial")
summary(model_journal)
```

```
##
## Call:
## glm(formula = ReviewerAgreed ~ Journal, family = "binomial",
##     data = small)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.208  -1.110  -1.033   1.247   1.329
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.07187    0.06812   1.055  0.29141
## JournalFE    -0.27908    0.09411  -2.965  0.00302 **
## JournalJANIM -0.16544    0.09598  -1.724  0.08476 .
## JournalJAPPL -0.23319    0.09323  -2.501  0.01237 *
## JournalJECOL -0.26183    0.09403  -2.785  0.00536 **
## JournalMEE   -0.42223    0.12868  -3.281  0.00103 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6908.3  on 4999  degrees of freedom
## Residual deviance: 6892.7  on 4994  degrees of freedom
## AIC: 6904.7
##
## Number of Fisher Scoring iterations: 3
```

Model journal and year

Finally, we can build a model combining both features: we fit a parameter for each journal/year combination

```
# Take the first journal as baseline, the colon mark stands for interaction term only.
model_journal_yr <- glm(ReviewerAgreed~Journal:I(Year-2003),
                        data = small, family = "binomial")
#summary(model_journal_yr)
```

Likelihoods

In R, you can extract the log-likelihood from a model object calling the function `logLik`

```
logLik(model_null)
```

```
## 'log Lik.' -3454.167 (df=1)
```

```
logLik(model_year)
```

```
## 'log Lik.' -3414.845 (df=2)
```

```
logLik(model_journal)
```

```
## 'log Lik.' -3446.335 (df=6)
```

```
logLik(model_journal_yr)
```

```
## 'log Lik.' -3395.365 (df=7)
```

Interpretation: because we're dealing with binary data, the likelihood is the probability of correctly predicting the agree/not agree for all the 5000 invitations considered. Therefore, the probability of guessing a (random) invitation correctly under the first model is:

```
exp(as.numeric(logLik(model_null)) / 5000)
```

```
## [1] 0.5011582
```

while the most complex model yields

```
exp(as.numeric(logLik(model_journal_yr))/ 5000)
```

```
## [1] 0.5070869
```

We didn't improve our guessing much by considering many parameters! This could be due to specific data points that are hard to predict, or mean that our explanatory variables are not sufficient to model our response variable.

AIC

We can also calculate AIC for logistic models. In R you can compute AIC using:

```
AIC(model_null)
```

```
## [1] 6910.334
```

```
AIC(model_year)
```

```
## [1] 6833.691
```

```
AIC(model_journal)
```

```
## [1] 6904.669
```

```
AIC(model_journal_yr)
```

```
## [1] 6804.73
```

As you can see, the model `model_journal_yr` has the smallest AIC among all and is preferred here.

BIC

In a similar vein, BIC (Bayesian Information Criterion) uses a slightly different penalization: $BIC = \log(n)p - 2\log L(\theta|D, M)$, where n is the number of data points. Again, smaller values stand for "better" models:

```
BIC(model_null)
```

```
## [1] 6916.851
```

```
BIC(model_year)
```

```
## [1] 6846.725
```

```
BIC(model_journal)
```

```
## [1] 6943.772
```

```
BIC(model_journal_yr)
```

```
## [1] 6850.35
```

Note that according to BIC, `model_year` is favored. As mentioned before, BIC would penalize a complex model more.

Cross validation

One very robust method to perform model selection, often used in machine learning, is cross-validation. The idea is simple: split the data in three parts: a small data set for exploring; a large set for fitting; a small set for testing (for example, 5%, 75%, 20%). You can use the first data set to explore freely and get inspired for a good model. The data will be then discarded. You use the largest data set for accurately fitting your model(s). Finally, you validate your model or select over competing models using the last data set.

Because you haven't used the test data for fitting, this should dramatically reduce the risk of overfitting. The downside of this is that we're wasting precious data. There are less expensive methods for cross validation, but if you have much data, or data are cheap, then cross-validation has the virtue of being fairly robust.

Let's try our hand at cross-validation. First, we split the data into three parts:

```
reviews$cv <- sample(1:3, nrow(reviews), prob = c(0.05, 0.75, 0.2), replace = TRUE)
dataexplore <- reviews[reviews$cv == 1,]
datafit <- reviews[reviews$cv == 2,]
datatest <- reviews[reviews$cv == 3,]
# We've already done our exploration.
# Let's fit the data using model_journal
# and model_journal_yr, which seem to be the most promising
cv_model1 <- glm(ReviewerAgreed~I(Year-2003), data = datafit, family = "binomial")
cv_model2 <- glm(ReviewerAgreed~Journal:I(Year-2003), data = datafit, family = "binomial")
```

Now that we've fitted the models, we can use the function `predict` to find the fitted values for the `testdata`:

```
mymodel <- cv_model1
# compute probabilities
pi <- predict(mymodel, newdata = datatest, type = "resp")
# compute log likelihood
mylogLik <- sum(datatest$ReviewerAgreed * log(pi) +
                  (1 - datatest$ReviewerAgreed) * log(1 - pi))
print(mylogLik)
```

```
## [1] -15520.91
```

repeat for the other model

```r
mymodel <- cv_model2
# compute probabilities
pi <- predict(mymodel, newdata = datatest, type = "resp")
# compute log likelihood
mylogLik <- sum(datatest$ReviewerAgreed * log(pi) +
                  (1 - datatest$ReviewerAgreed) * log(1 - pi))
print(mylogLik)
```

```
## [1] -15473.93
```

Cross validation supports the choice of the more complex model here.

Other approaches

Bayesian models are gaining much traction in biology. The advantage of these models is that you can get a posterior distribution for the parameter values, reducing the need for p-values and AIC. The downside is that fitting these models is computationally much more expensive (you have to find a distribution of values instead of a single value).

There are three main ways to perform model selection in Bayesian models:

- **Reversible-jump MCMC** You build a Monte Carlo Markov Chain that is allowed to "jump" between models. You can choose a prior for the probability of being in each of the models; the posterior distribution gives you an estimate of how much the data supports each model. Upside: direct measure. Downside: difficult to implement in practice – you need to avoid being "trapped" in a model.
- **Bayes Factors** Ratio between the probability of two competing models. Can be computed analytically for simple models. Can also be interpreted as the average likelihood when parameters are chosen according to their prior (or posterior) distribution. Upside: straightforward interpretation — it follows from Bayes theorem; Downside: in most cases, one needs to approximate it; can be tricky to compute for complex models.
- **DIC** Similar to AIC and BIC, but using distributions instead of point estimates.

Another alternative paradigm for model selection is Minimum-Description Length. The spirit is that a model is a way to "compress" the data. Then you want to choose the model whose total description length (compressed data + description of the model) is minimized.

A word of caution

The "best" model you've selected could still be a terrible model (best among bad ones). Out-of-fit prediction (such as in the cross-validation above) can give you a sense of how well you're modeling the data.

When in doubt, remember the (in)famous paper in Nature by Tatem et al. 2004, which used some flavor of model selection to claim that, according to their linear regression, in the 2156 Olympics the fastest woman would run faster than the fastest man. One of the many memorable letters that ensued reads:

> Sir — A. J. Tatem and colleagues calculate that women may out-sprint men by the middle of the twenty-second century (Nature 431,525; 2004). They omit to mention, however, that (according to their analysis) a far more interesting race should occur in about 2636, when times of less than zero seconds will be recorded.
>
> In the intervening 600 years, the authors may wish to address the obvious challenges raised for both time-keeping and the teaching of basic statistics.
>
> Kenneth Rice

Prediction for population outside the current data is prophecy.

V. Programming Challenge

The secret alphabetic relationship to COVID-19

To try your hand at p-hacking and overfitting, and show how these practices can lead to completely inane results.

You can download today's data on the geographic distribution of COVID-19 cases worldwide from the source below. The data is updated daily and contains the lastest publicly available data on COVID-19 by country. Our task is to show significant and secret/superstitious relationships between the number of COVID-19 cases and the country name's first letter. Those relationships can be alphabetic order, or some secret patterns (e.g., vowels versus consonants, letters before M versus after). For example, you may show that the number of COVID-19 cases is increasingly/decreasingly associated with the alphabetic position of the first letter of the country name in a particular day.

The data reported the "cumulative number for 14 days of COVID-19 cases per 100k individuals" for each country or territory. There are also several other covariates in the data set and you may also consider their interaction effects with the country name's alphabetic order. You may choose one particular day and report your findings, or choose an overall average to date. Note that if you analyze multiple days in your analysis, you may need to consider the longitudinal data analysis issues and we do not recommend here.

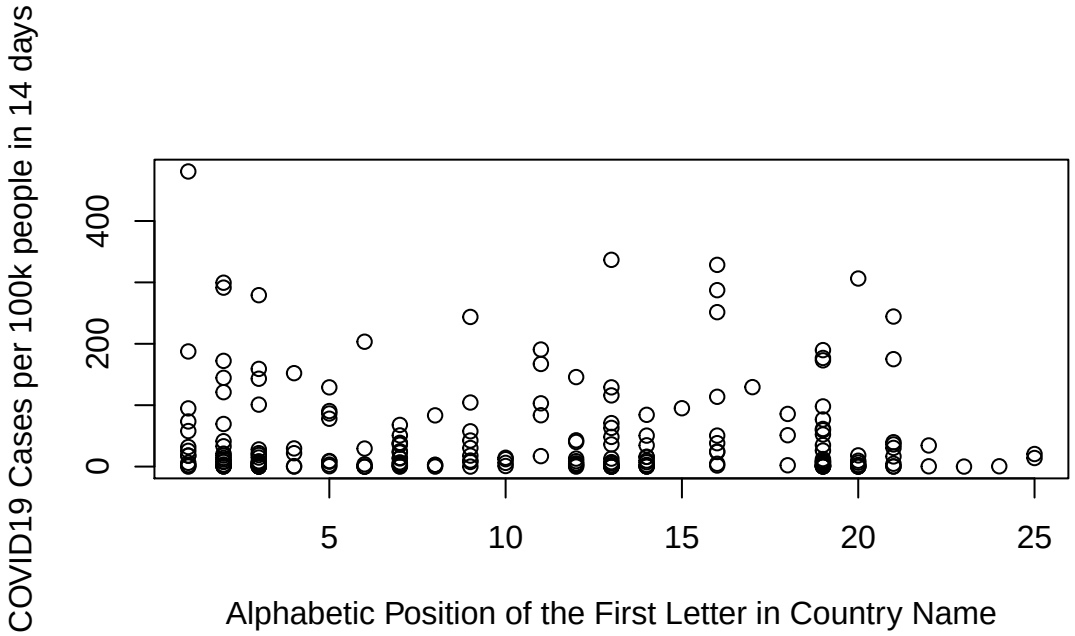
```
# read data
library(utils)

#read the Dataset sheet into "R". The dataset will be called "data".
data <- read.csv("https://opendata.ecdc.europa.eu/covid19/casedistribution/csv", na.strings = ""

# the code below shows how you may recode the country name's first letter to a numeric alphabeti
## also you may pick a particular date and analyze data for only that day.
data1 <- data[(data$dateRep=="11/08/2020"), ]  ## you may code a for loop and search for a day t
CountryAb <- as.integer(as.factor(substr(data1$countriesAndTerritories,1,1)))


# analyze the correlation between Cumulative_number_for_14_days_of_COVID.19_cases_per_100000 and
```

```r
plot(CountryAb, data1$Cumulative_number_for_14_days_of_COVID.19_cases_per_100000, xlab="Alphabet
```



Not surprisingly, by choosing a random day 8/11/2020, I did not find a significant relationship between alphabetic order of country name and COVID-19 cases.

```r
model1<- lm(Cumulative_number_for_14_days_of_COVID.19_cases_per_100000~CountryAb, data=data1)
summary(model1)
```

```
##
## Call:
## lm(formula = Cumulative_number_for_14_days_of_COVID.19_cases_per_100000 ~
##     CountryAb, data = data1)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -52.99 -44.82 -35.16   7.99 427.68
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  53.4215    10.0108   5.336 2.48e-07 ***
## CountryAb    -0.4363     0.7875  -0.554     0.58
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 78.42 on 207 degrees of freedom
## Multiple R-squared:  0.001481,   Adjusted R-squared:  -0.003343
## F-statistic: 0.307 on 1 and 207 DF,  p-value: 0.5801
```

Now try other days and possibly consider interaction terms and other alphabetic patterns, can

you find a secret relationship to tell? Finally, the ultimate task, can you explain your findings in a scientific way? Either supporting your conclusions and explaining it, or criticizing what you have done and explaining why you got a significant p-value would be fine.