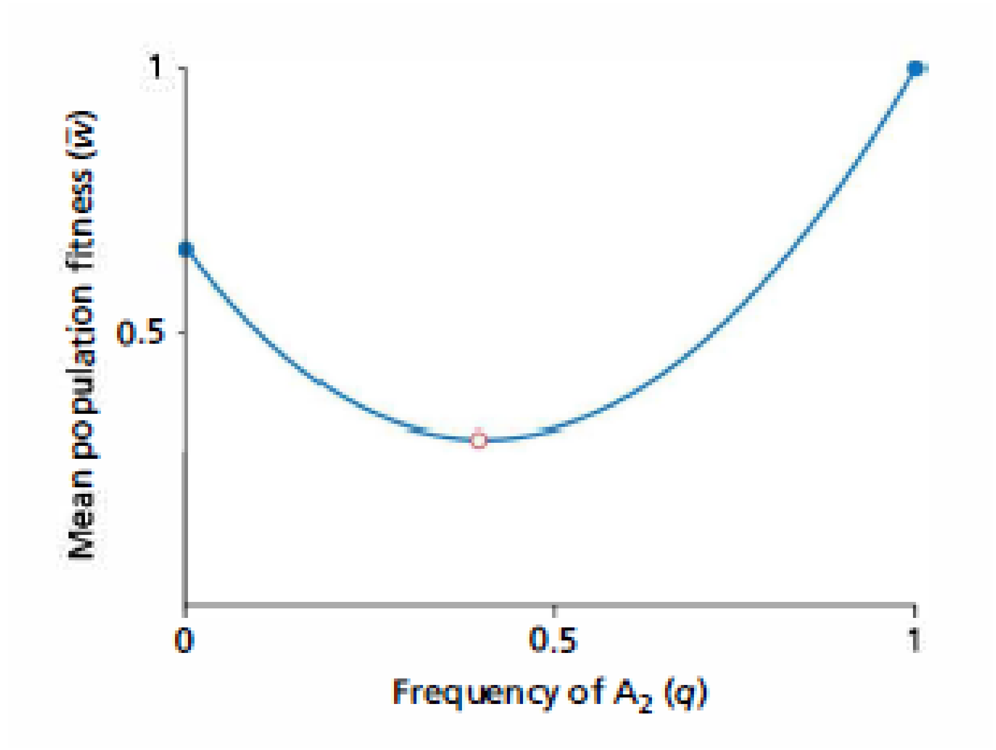
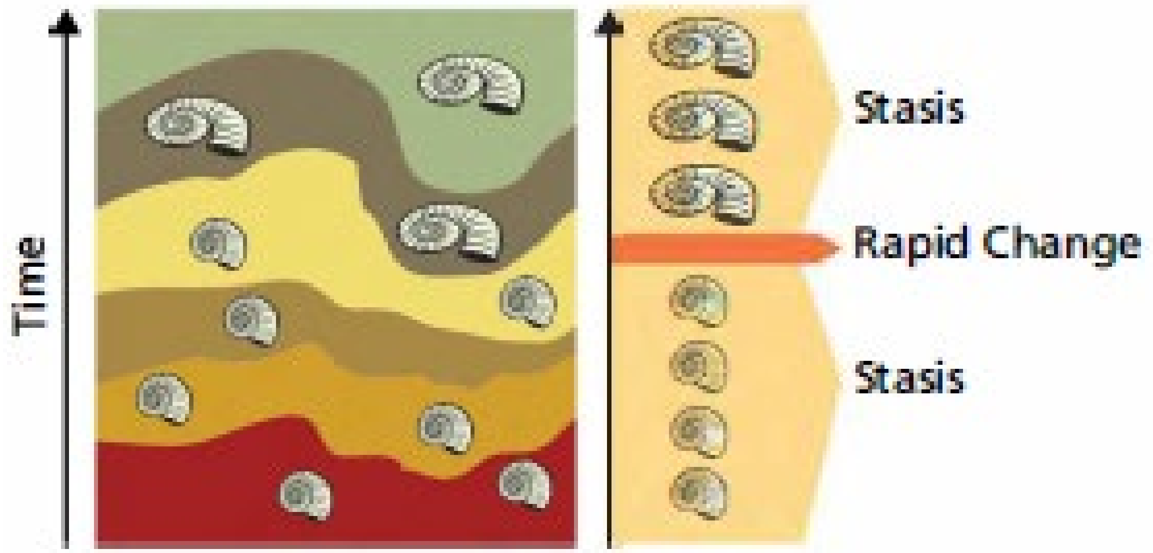
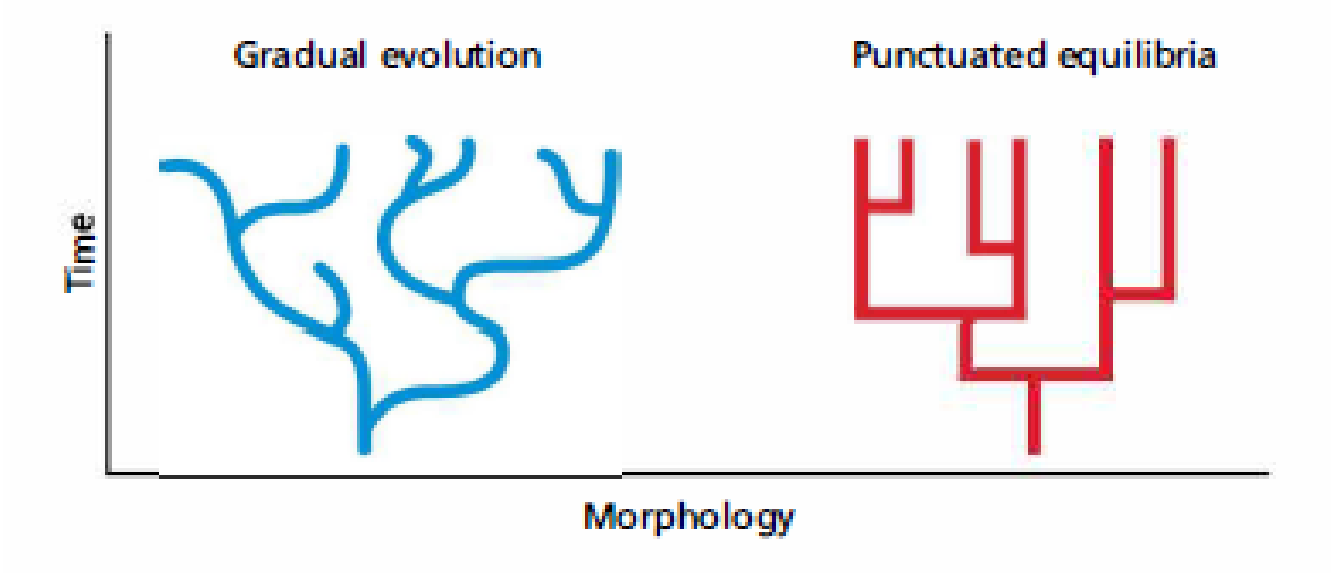
5. Doğal seçilimin gücü

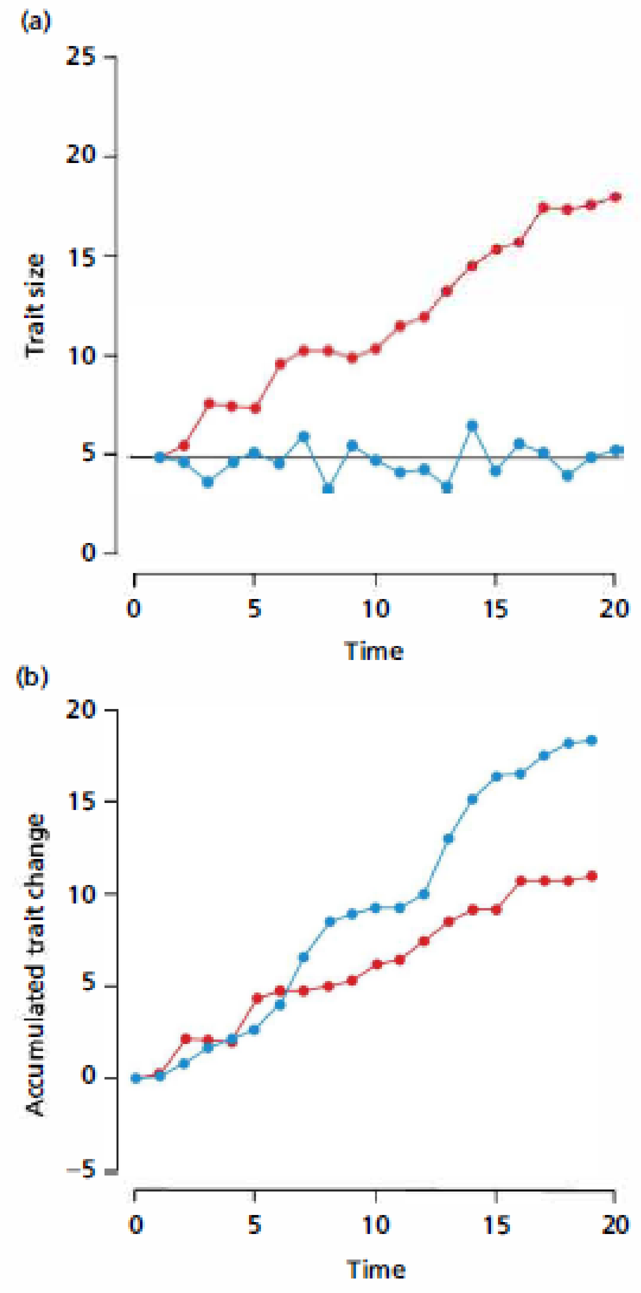
Teorik

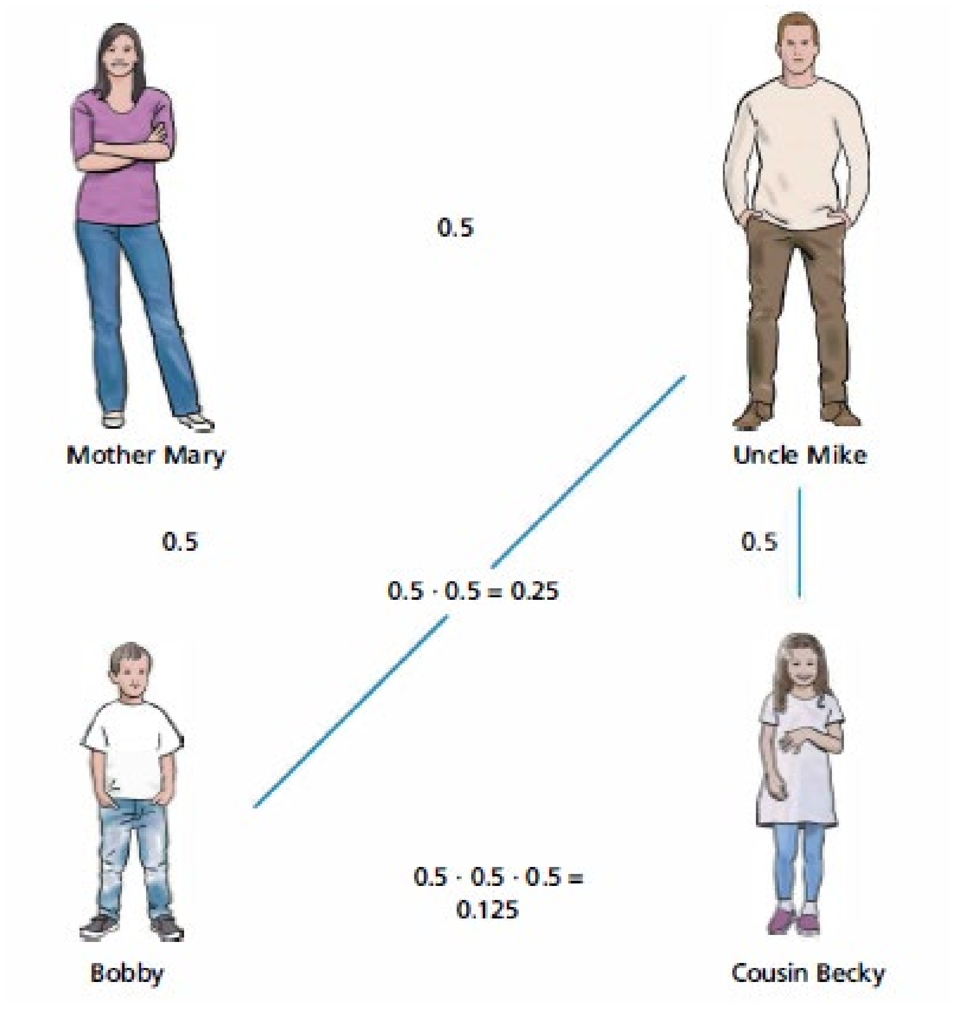


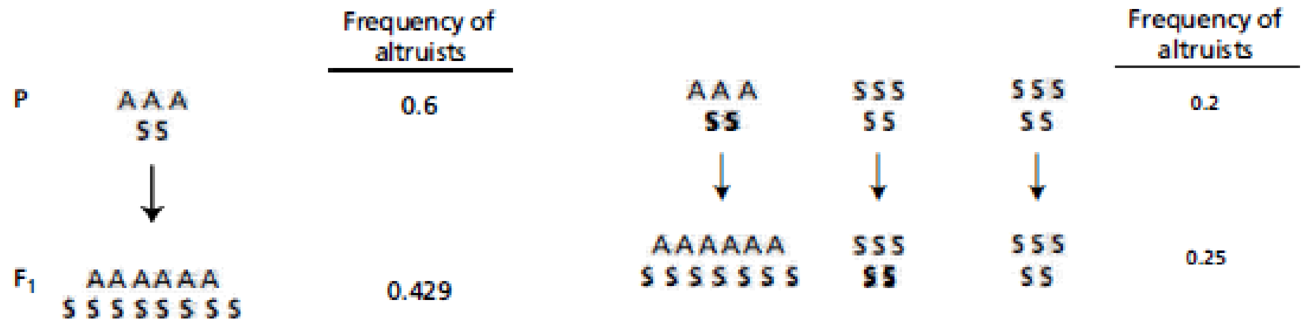


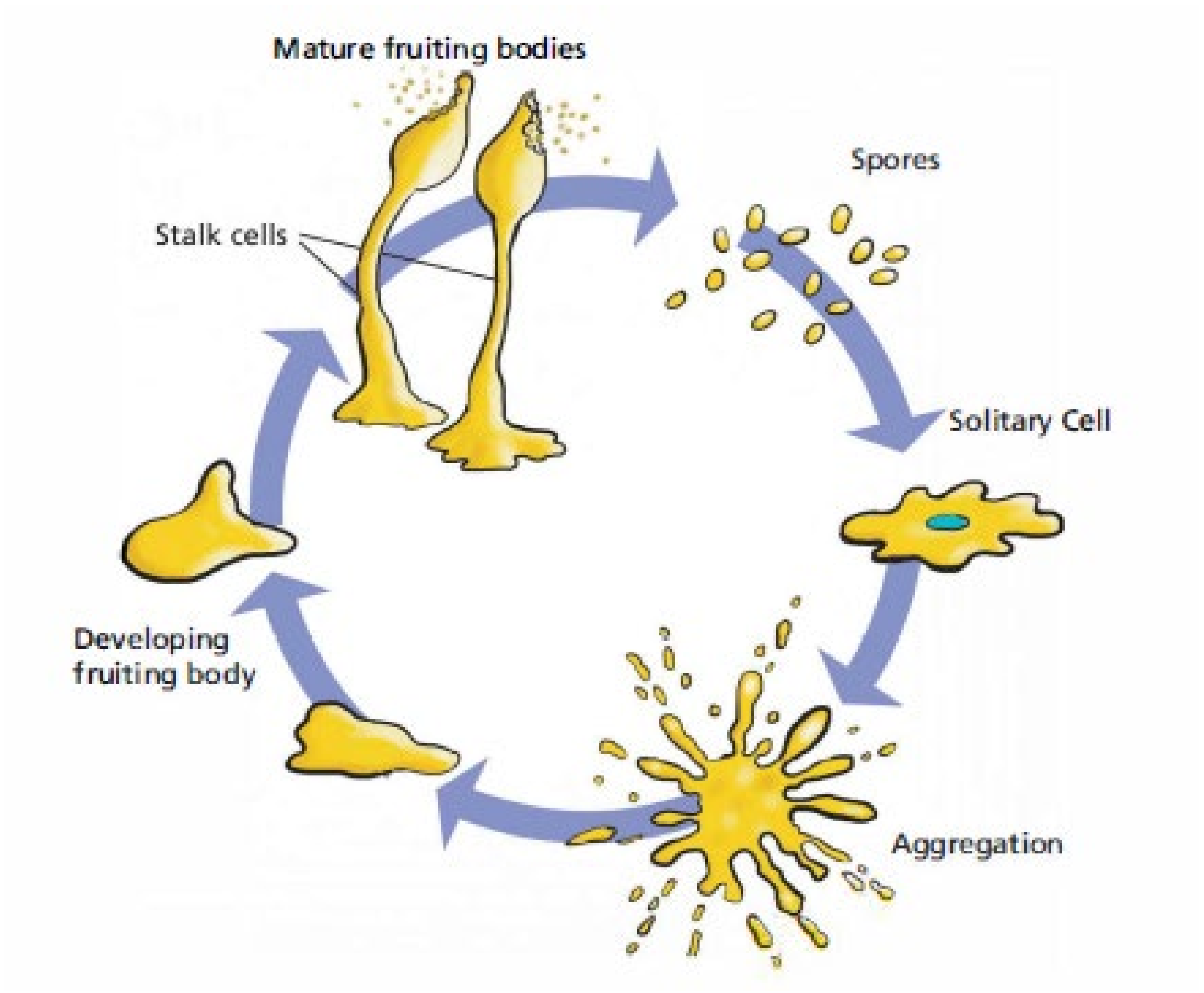
Figures and Tables were taken from Sætre, G. P., & Ravinet, M. (2019). *Evolutionary genetics: Concepts, analysis, and practice*. Oxford University Press, USA. For educational purpose only

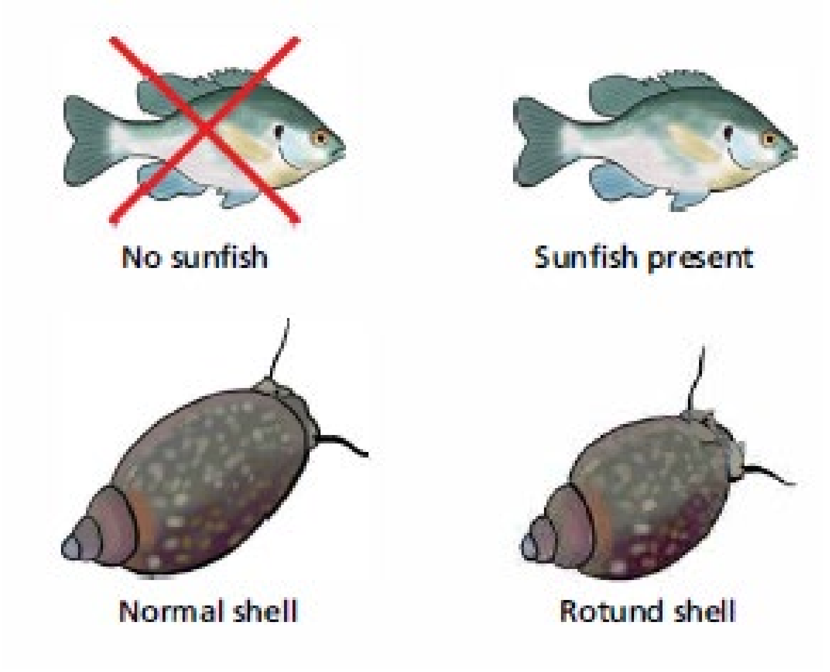

Gradual evolution
Punctuated equilibria
Time
Morphology


(a)
Trait size
Time
(b)
Accumulated trait change
Time

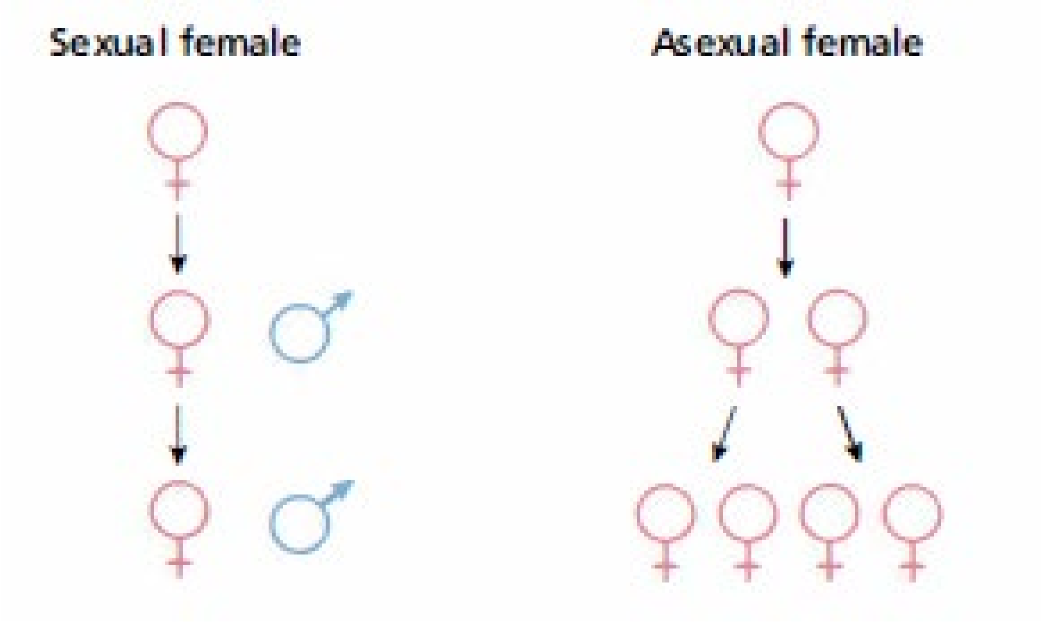

Mother Mary
Uncle Mike
0.5
0.5
0.5
0.5 · 0.5 = 0.25
Bobby
Cousin Becky
0.5 · 0.5 · 0.5 =
0.125

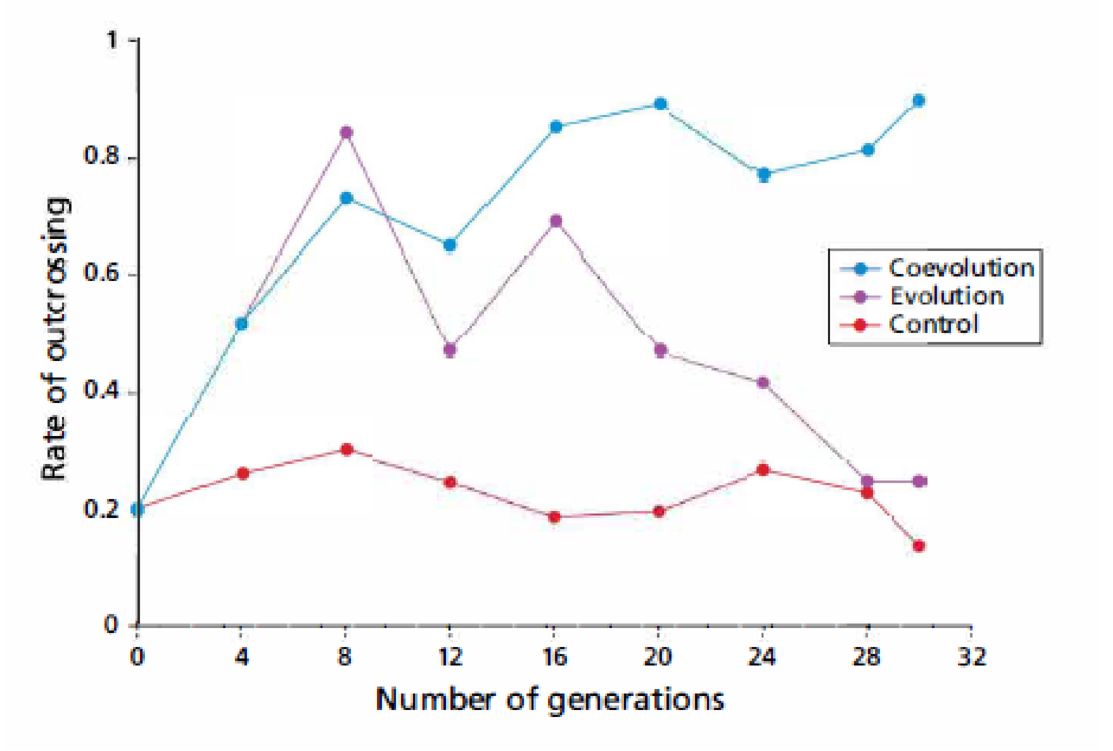

Frequency of altruists
P
AAA
SS
0.6
F1
AAAAAA
SSSSSSSS
0.429
Frequency of altruists
AAA
SS
SSS
SS
SSS
SS
0.2
AAAAAA
SSSSSSS
SSS
SS
SSS
SS
0.25

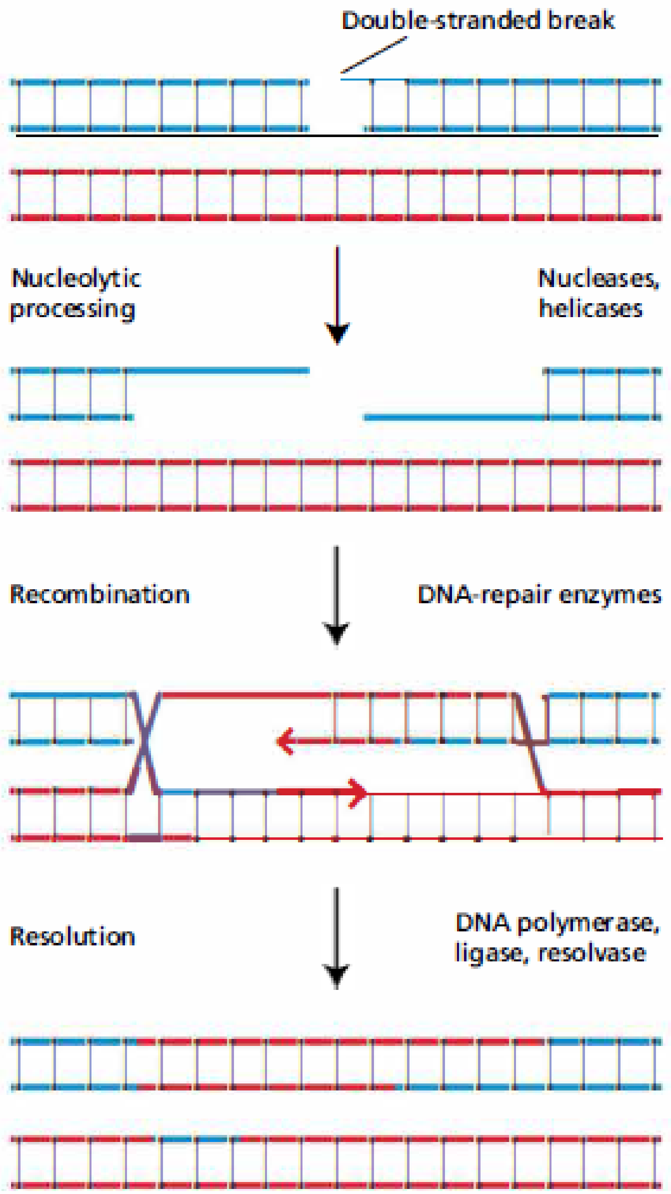


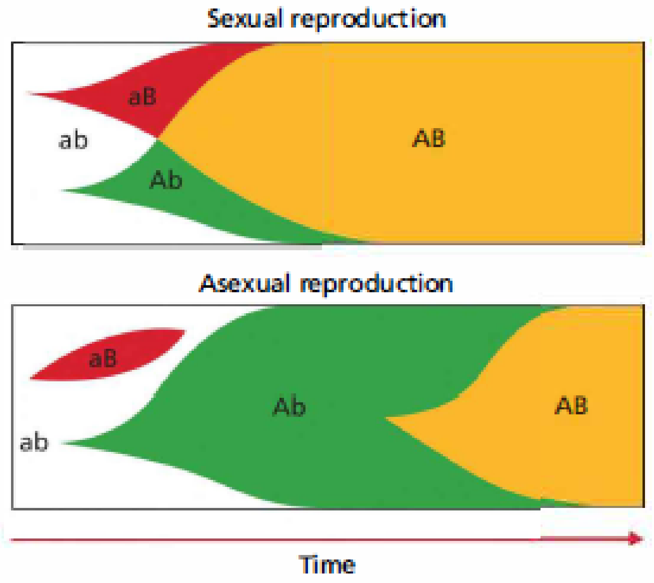


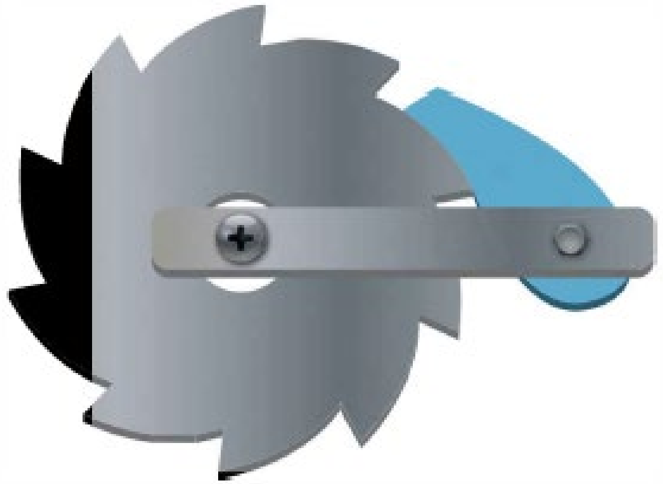
Figures and Tables were taken from Sætre, G. P., & Ravinet, M. (2019). *Evolutionary genetics: Concepts, analysis, and practice*. Oxford University Press, USA. For educational purpose only


Sexual female
Asexual female

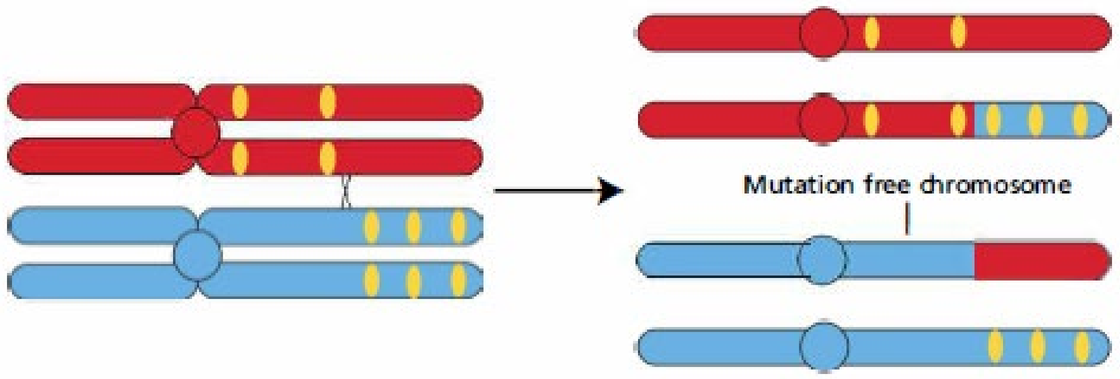

Rate of outcrossing
Number of generations
Coevolution
Evolution
Control
0
0.2
0.4
0.6
0.8
1
0
4
8
12
16
20
24
28
32

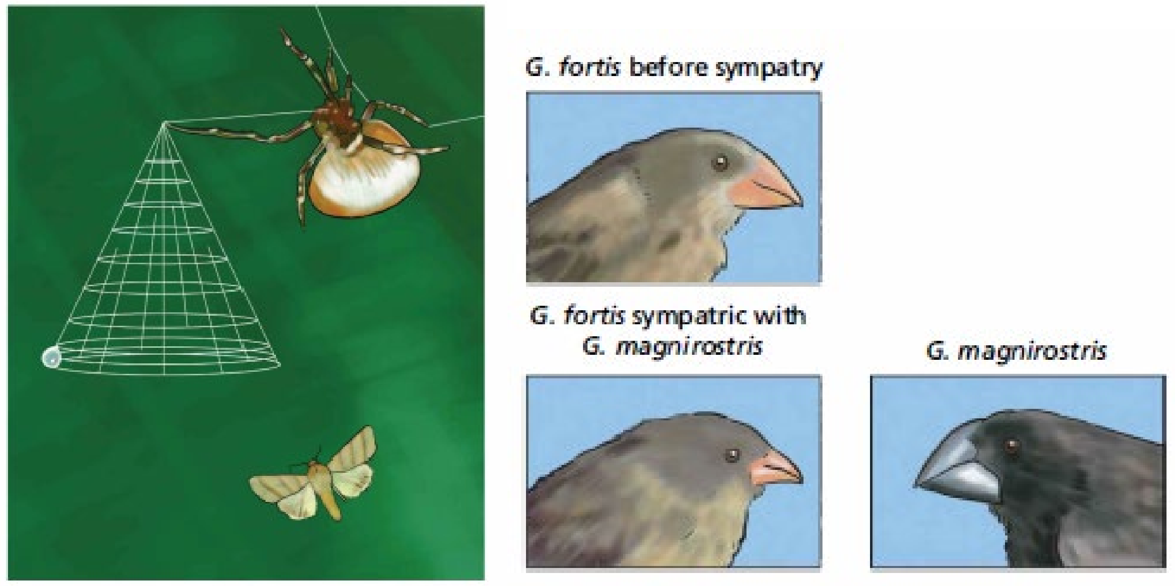






Figures and Tables were taken from Sætre, G. P., & Ravinet, M. (2019). *Evolutionary genetics: Concepts, analysis, and practice*. Oxford University Press, USA. For educational purpose only


Mutation free chromosome


G. fortis before sympatry
G. fortis sympatric with G. magnirostris
G. magnirostris

5. Doğal seçilimin gücü

Uygulama

```
# set up genotype counts
a <- c(48, 28, 4) # americans
d <- c(0, 3, 77) # druze

# get the number of people sampled (same for both)
n <- sum(a)

# calculate the frequency of the T allele - or p
# for americans
p_a <- ((a[1]*2) + a[2])/(2*n)
# for druze
p_d <- ((d[1]*2) + d[2])/(2*n)

# calculating the frequency of C (or q) is then trivial
q_a <- 1 - p_a
q_d <- 1 - p_d
```

Next we can calculate the allele frequencies for the metapopulation - i.e. Americans of European descent and Druze considered as a single population. This is as simple as taking the mean of the two allele frequencies.

```
# calculate total allele frequency
p_t <- (p_a + p_d)/2
q_t <- 1 - p_t
```

With these allele frequencies calculated, we can very easily calculate expected heterozygosities - remember this is just $2pq$.

```
# first calculate expected heterozygosity for the two populations
# americans
hs_a <- 2*p_a*q_a
# druze
hs_d <- 2*p_d*q_d
# then take the mean of this
hs <- (hs_a + hs_d)/2

# next calculate expected heterozygosity for the metapopulations
ht <- 2*p_t*q_t
```

With all the relevant expected heterozygosities in place, we are now ready to calculate F_{ST} which we can do like so:

```
# calculate fst
fst <- (ht - hs)/ht
```

If your calculations were correct, then you should have an F_{ST} estimate of 0.59 - this is very high for between two human populations. One way to interpret the F_{ST} value we have here is that 59% of genetic variance we observe differs between populations. Since population can explain such a large difference in this case, we might expect selection to be responsible…

Writing a set of F_{ST} functions

The code in the previous section was useful to demonstrate how we can calculate F_{ST}, but it would be a lot of work to run through this every single time we want estimate the statistic for a locus. This being R, we can of course easily create a function that will do all of the leg work for us! We will take the code we wrote out in the last section and use it here to write two functions that we can use when we want to calculate F_{ST}. Note that for simplicity, we will only write functions that work for **two populations**.

First, we will write a function called `calc_af` which will take genotype counts from two populations and calculate allele frequencies. The function also allows for sample sizes to vary between the two populations.

```
# a simple function to calculate allele frequencies in two populations
calc_af <- function(pop1, pop2){
  # get the number of samples
  n_1 <- sum(pop1)
  n_2 <- sum(pop2)
  # calculate frequency of 1st allele - p
  # for pop1
  p_1 <- ((pop1[1]*2) + pop1[2])/(2*n_1)
  # for pop2
  p_2 <- ((pop2[1]*2) + pop2[2])/(2*n_2)

  return(c(p_1, p_2))
}
```

Since it is very straightforward for use to calculate the frequency of the second allele once we have the frequency of the first (i.e. $q = 1 - p$), our `calc_af` funct on w ll only calculate p for both populat ons. Let's test t on the data from our prev ous example.

```
# testing our function on the american/druze example
afs <- calc_af(pop1 = c(48, 28, 4), pop2 = c(0, 3, 77))
```

So now that we have a function that calculates allele frequencies in the two populations, we can write our `calc_fst` function to take these frequencies and calculate F_{ST} from them.

```
# a function to calculate fst
calc_fst <- function(p_1, p_2){

  # calculate q1 and q2
  q_1 <- 1 - p_1
  q_2 <- 1 - p_2

  # calculate total allele frequency
  p_t <- (p_1 + p_2)/2
  q_t <- 1 - p_t

  # calculate expected heterozygosity
  # first calculate expected heterozygosity for the two populations
  # pop1
  hs_1 <- 2*p_1*q_1
  # pop2
  hs_2 <- 2*p_2*q_2
  # then take the mean of this
  hs <- (hs_1 + hs_2)/2

  # next calculate expected heterozygosity for the metapopulations
  ht <- 2*p_t*q_t

  # calculate fst
  fst <- (ht - hs)/ht

  # return output
  return(fst)
}
```

Let's test our function on the allele frequencies we calculated with our `calc_af` function.

```
# testing our function on the american/druze example
calc_fst(afs[1], afs[2])
```

This should be the same as you got before, but with a lot less work. Next, we'll look at applying a function to a bigger dataset.

Applying functions to matrices and data frames

Extending our *LCT* and lactase persistence example, let's get some data from multiple human populations. You can download the data here (https://evolutionarygenetics.github.io/lct_count.tsv)

```
lct_counts <- read_delim("./lct_count.tsv", delim = "\t")
```

You should now have a tibble in your R environment with allele counts for the SNP rs4988235 for 53 populations. Again, these data are all from Bersaglieri et al. 2002 (https://www.sciencedirect.com/science/article/pii/S0002929707628389).

What we have is the counts of allleles but what we actually want is the allele frequency for T - that is how we can calculate F_{ST}. However, as you will recall, our `calc_af` function takes the input for **TWO** populations. Calculating frequencies is pretty straightforward so let's simplify our function - we will call it `calc_af_simple`.

```
calc_af_simple <- function(counts){
  # get the number of samples
  n <- sum(counts)
  # calculate frequency of 1st allele - p
  # for pop1
  p <- ((counts[1]*2) + counts[2])/(2*n)
  return(p)
}
```

Let's try this function out on counts for a single population. We use indexing here to select the first row and only columns 2:4, since our function is only expecting the count data, not the population name.

```
calc_af_simple(lct_counts[1, 2:4])
```

Great! So this works well. Now let's get p (the frequency of the T allele) for all the populations. We can do this extremely fast and easily using `apply`.

```
# we can write it this way
p <- select(lct_counts, -pop) %>% apply(1, calc_af_simple)

# or this way
p <- apply(select(lct_counts, -pop), 1, calc_af_simple)
```

`apply` is a similar function to `sapply`, except it works on matrices or data.frames. All it takes three arguments - the object (i.e. matrix/data.frame) which you want to apply it to, the index you apply it over and finally, the function you are applying. In the two examples of how to write an `apply` command above, we first removed the `pop` command with `select` and we then used `1` to denote we are operating over the **rows** of the data.frame. Finally we specify that for each row, we will use our `calc_af_simple` function.

We can now combine our vector of allele frequencies with the population names to create a data.frame of frequencies. Like so

```
lct_freq <- as.tibble(data.frame(pop = lct_counts$pop, p))
```

```
## Warning: `as.tibble()` is deprecated, use `as_tibble()` (but mind the new semantics).
## This warning is displayed once per session.
```

Now we can easily calculate a pairwise F_{ST} with our `calc_fst` function. For example, let's calculate F_{ST} for European Americans and East Asians. We will use `dplyr` commands for this. Recall that `|` means **or**.

```
# extract frequencies
afs <- filter(lct_freq, pop == "European_American" | pop == "East_Asian") %>% pull(p)
# calculate fst
calc_fst(afs[1], afs[2])
```

All the `pull` function does here is return our `p` column as a vector we can use in the `calc_fst` function. As with our previous example, we can see *F*~~ST is actually pretty high between these populations for this SNP. What about if we compared East Asians with the Bedouin people from Israel?

```
# extract frequencies
afs <- filter(lct_freq, pop == "East_Asian" | pop == "Bedouin_Negev_Israel") %>% pull(p)
# calculate fst
calc_fst(afs[1], afs[2])
```

Here we see F_{ST} is substantially lower. Allele frequency differences are lower between these populations.

Visualising F_{ST} along a chromosome

Next, we will combine the skills we learned with `apply` and our custom functions to calculate F_{ST} for a series of SNPs in the vicinity of the *LCT* gene on chromosome 2. This is essentially a genome scan, an approach that can be used to detect signatures of selection in the genome. You can download the data here (https://evolutionarygenetics.github.io/LCT_snps.tsv)

First of all, we will read in the data:

```
lct_snps <- read_delim("./LCT_snps.tsv", delim = "\t")
```

This data is also from from Bersaglieri et al. 2002 (https://www.sciencedirect.com/science/article/pii/S0002929707628389). It is the allele frequencyin various human populations for one allele at a set of 101 biallelic SNP markers close to the *LCT* gene on chromosome 2 in the human gene. Each row is a SNP and there are three frequencies - one for North Americans of European descent, one for African Americans and one for East Asians.

Since we have the allele frequencies, we can easily calculate F_{ST} for each of these SNPs. For our example here, we will do this between `european_americans` and `east_asians`. First of all, let's use our `calc_fst` function on just a single SNP.

```
calc_fst(lct_snps[1, 4], lct_snps[1, 6])
```

How can we scale this up calculate F_{ST} for all those loci rapidly? We can do this in a single line of code using a function called `pmap`.

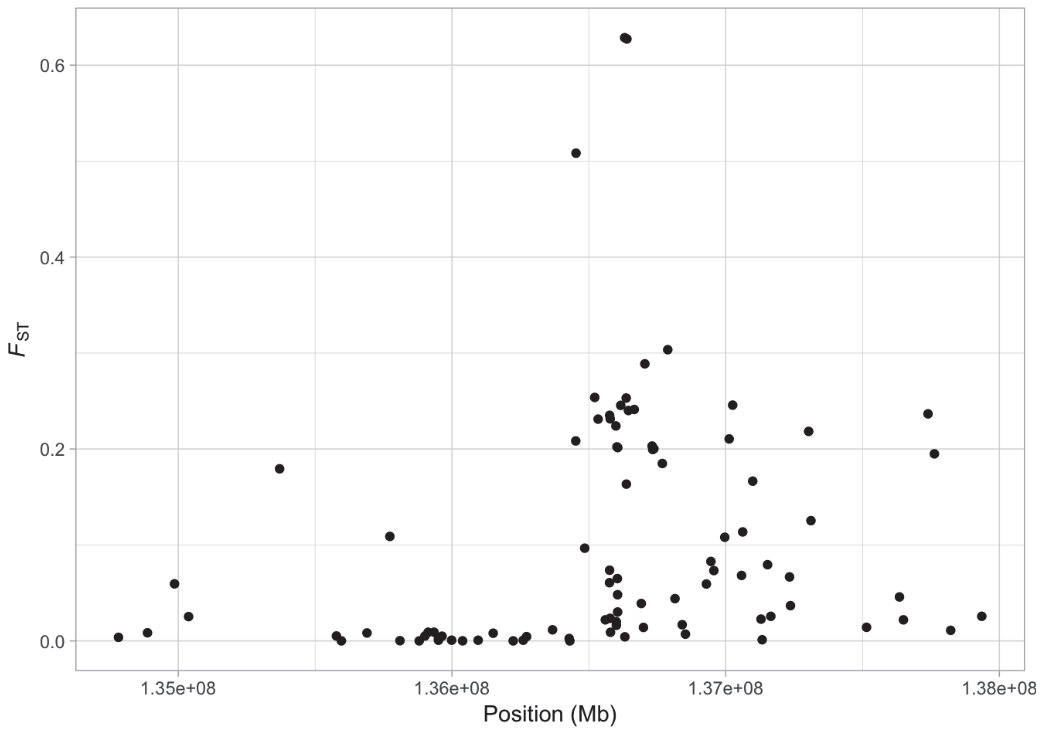
```
# test the code
pmap_dbl(lct_snps, ~ calc_fst(..4, ..6))
# make an fst column
lct_snps$fst <- pmap_dbl(lct_snps, ~ calc_fst(..4, ..6))
```

What did we do here? Well `pmap` can be a tricky function to understand but in this example it is pretty straightforward. It takes 3 main arguments, the first is our data.frame `lct_snps`, then we use `~` to denote we want to apply a function to the rows of this data.frame. Finally, we use our `calc_fst` function but we use `..4` and `..6` to denote that we want to use the 4th and 6th columns respectively (i.e. North Americans of

European descent and East Asians). If you wanted to choose the 5th column for example, you would use `..5` .

Anyway, now that we have F_{ST} estimates for each of our SNPs, we can visualise the variation along the chromosome with `ggplot2` .

```
a <- ggplot(lct_snps, aes(coord, fst)) + geom_point()
a <- a + xlab("Position (Mb)") + ylab(expression(italic(F)[ST]))
a <- a + theme_light()
a
```

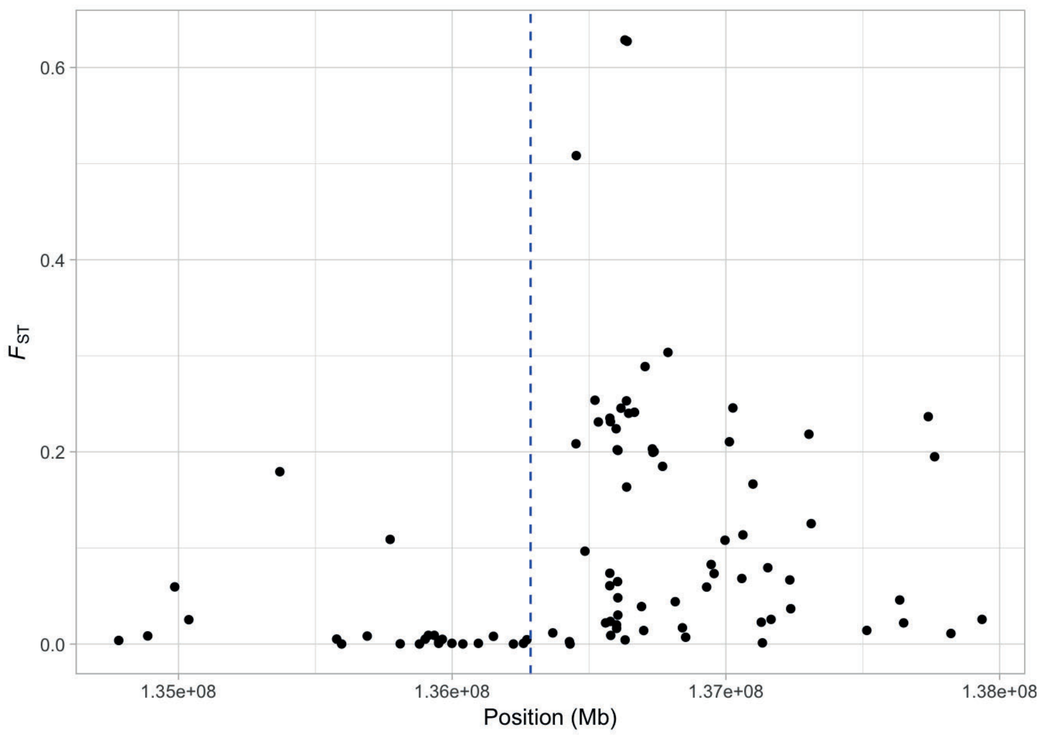


What are we seeing here? Quite clearly, there is a an increase in F_{ST} along the chromosome, with a few SNPs showing extremely high values. It might make things a bit clearer if we mark on our plot the midpoint of the *LCT* gene. We know the gene occurs between 136,261,885 bp and 136,311,220 bp on Chromsome 2 (from the UCSC Genome Browser (https://genome.ucsc.edu/cgi-bin/hgTracks?db=hg18&lastVirtModeType=default&lastVirtModeExtraState=&virtModeType=default&virtMode=0&nonVirtPosition=&position=chr2%3A136261885-136311220&hgsid=690775201_Cttqx4oVutDVGra183ZtAUFm6z2W)). So first we will find the midpoint of the gene.

```
# define the start and stop positions of the gene
lct_start <- 136261885
lct_stop <- 136311220
# calculate the midpoint
lct_mid <- (lct_start + lct_stop)/2
```

All we need to do to add it to our plot is use the `geom_vline` function.

```
a <- a + geom_vline(xintercept = lct_mid, lty = 2, col = "blue")
a
```



When the mid point of the gene is marked, it is clear that there is an increase in F_{ST} just upstream from the *LCT* gene. Perhaps we want to highlight the SNP that we calculated F_{ST} for in our first example? We can do this easily by setting a **factor** in our original data.frame. Recall that the original SNP is called rs4988235.

```
lct_snps <- lct_snps %>%
  mutate(status = if_else(snp_id == "rs4988235", "Yes", "No"))
```

What did we do here? Well first we used the `mutate` function to create a new column. Inside `mutate`, we used `if_else`. This function allows us to set a condition and print values based on whether that condition is met or not. So here we are basically saying, if `snp_id == "rs4988235"` then return `Yes`, if not then return `No`. We named this column `status`. More on how we'll use this in a moment but first we'll use a simple example to demonstrate `if_else` again.

```
if_else(5 > 10, "Surprising", "Not surprising")
```

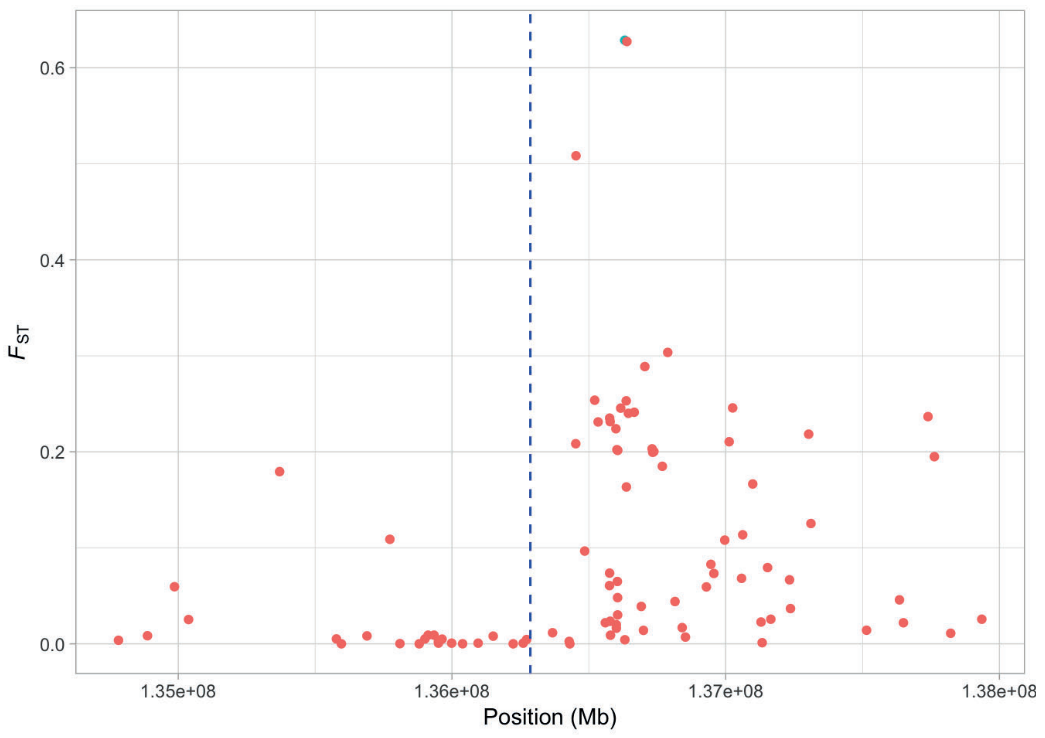
```
## [1] "Not surprising"
```

```
if_else(10 > 5, "Surprising", "Not surprising")
```

```
## [1] "Surprising"
```

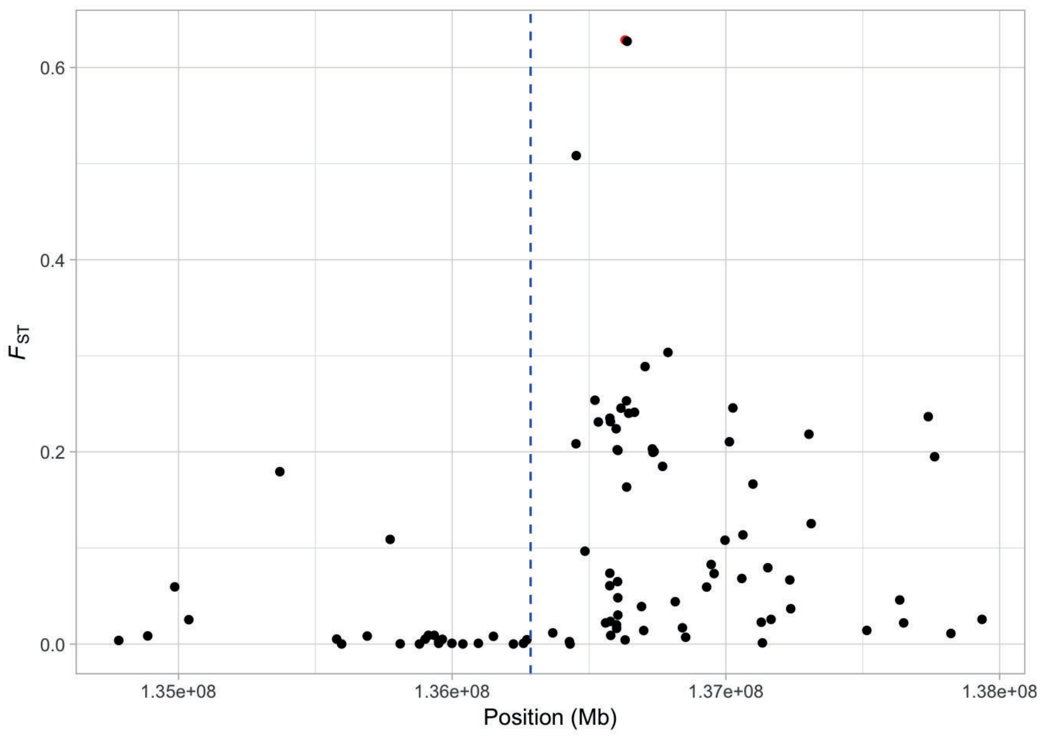
Now to highlight the SNP on our plot. We do this by setting colour as a variable in our `aes`.

```
a <- ggplot(lct_snps, aes(coord, fst, colour = status)) + geom_point()
a <- a + xlab("Position (Mb)") + ylab(expression(italic(F)[ST]))
a <- a + geom_vline(xintercept = lct_mid, lty = 2, col = "blue")
a <- a + theme_light() + theme(legend.position = "none")
a
```



Now we see, our focal SNP is highlighted in the plot. We'll change the colours to make it a little bit clearer.

```
a + scale_colour_manual(values = c("black", "red"))
```



In the next section, we'll demonstrate how we can use the distribution of F_{ST} to identify **outliers** as potential targets of selection.

Distributions in R

A good understanding of statistical distributions is an important part of any evolutionary biologist's toolkit. We have already encountered statistical distributions before during these tutorials - when we used the binomial and chi-squared distributions. Today, we will focus on the **normal distribution** which will be important for understanding how to detect F_{ST} outliers and also for understanding some of the topics that we will encounter in the following tutorial.

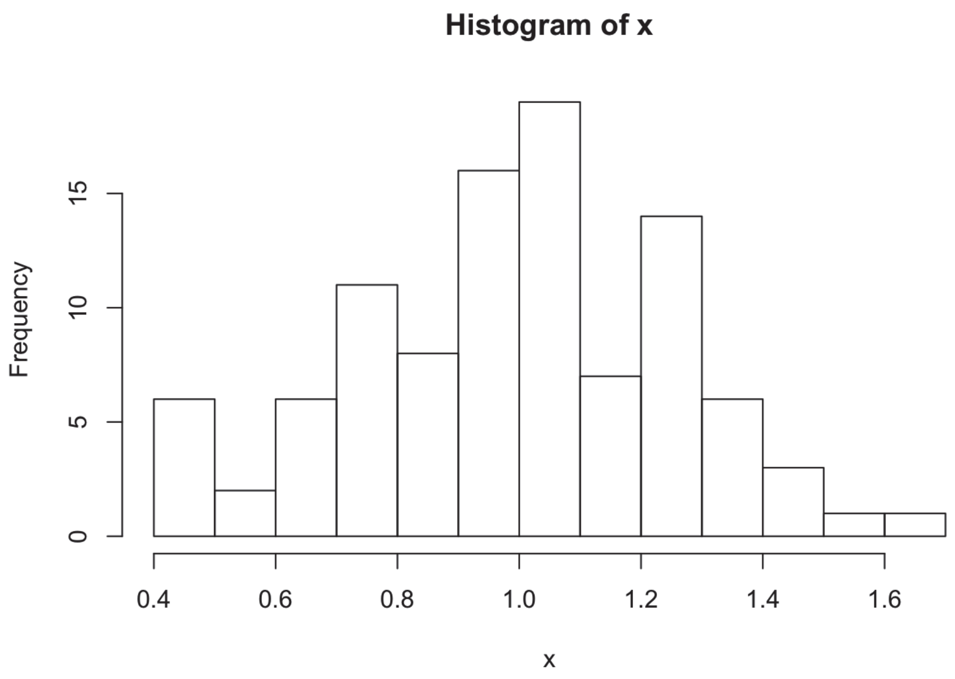
As you might remember, R has a suite of functions that are based around statistical distributions. We will use one of these - `rnorm` - to generate a random draw of 100 values from the normal distribution.

```
rnorm(n = 100, mean = 1, sd = 0.25)
```

What did we do here? First we told `rnorm` we want 100 values - i.e. `n = 100`. We are going to draw them from a distribution with a mean of 1 and a standard deviation of 0.25. Remember, `rnorm` stands for random sample of the normal distribution - so values will differ each time you run this function.

Let's do this again, but this time we will visualise the distribution using a histogram.

```
x <- rnorm(n = 100, mean = 1, sd = 0.25)
hist(x, breaks = 10)
```



You should already be familiar with the normal distribution, but if not then visualising it might help your understanding. The majority of observations occur at the mean (i.e. the peak in our histogram), whereas the extremes of the distribution represent outliers.

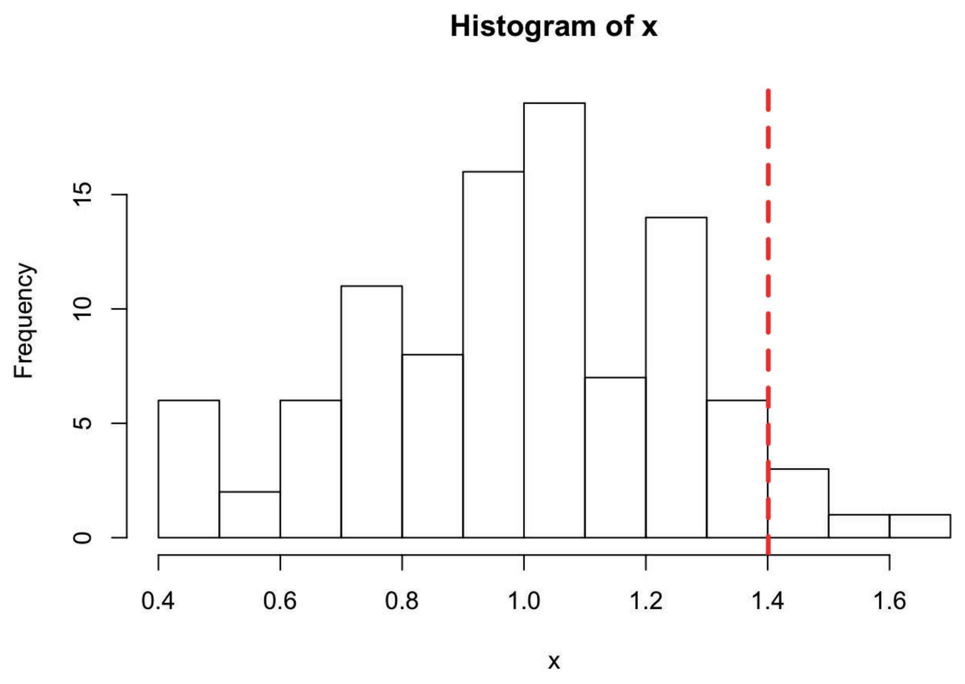
In order to identify something as an outlier, we need to identify an arbitrary **threshold**, above which we consider something an extreme of the distribution. So for example, we could say that the top 5% of values in our distribution are potentially outliers. This means we need to set the percentile or quantile threshold of our distribution to 0.95. We can do this like so

```
quantile(x, 0.95)
```

So this means that if a value is greater than about 1.31, we can consider it in the top 5% of our data distribution - and thus a potential outlier. **(Note, you might get a slightly different answer to this because we used a random sample from the distribution)**

Let's visualise this on our distribution to make it a bit clearer what it means:

```
hist(x, breaks = 10)
abline(v = quantile(x, 0.95), lty = 2, lwd = 3, col = "red")
```

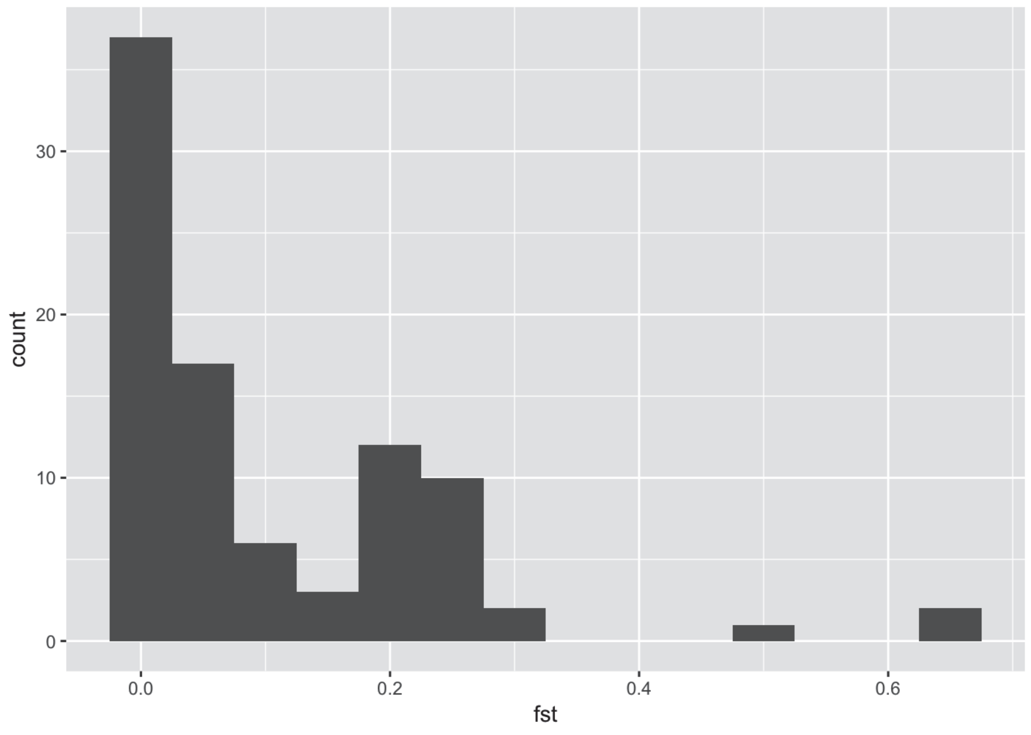


In other words, the red line here shows that this is our threshold, above which we consider values a potential outlier.

Identifying outliers in our F_{ST} distribution

So how can we apply what we just learned about outliers to our F_{ST} data? First of all, we might reasonably ask, is the data normally distributed? Let's plot it and see.

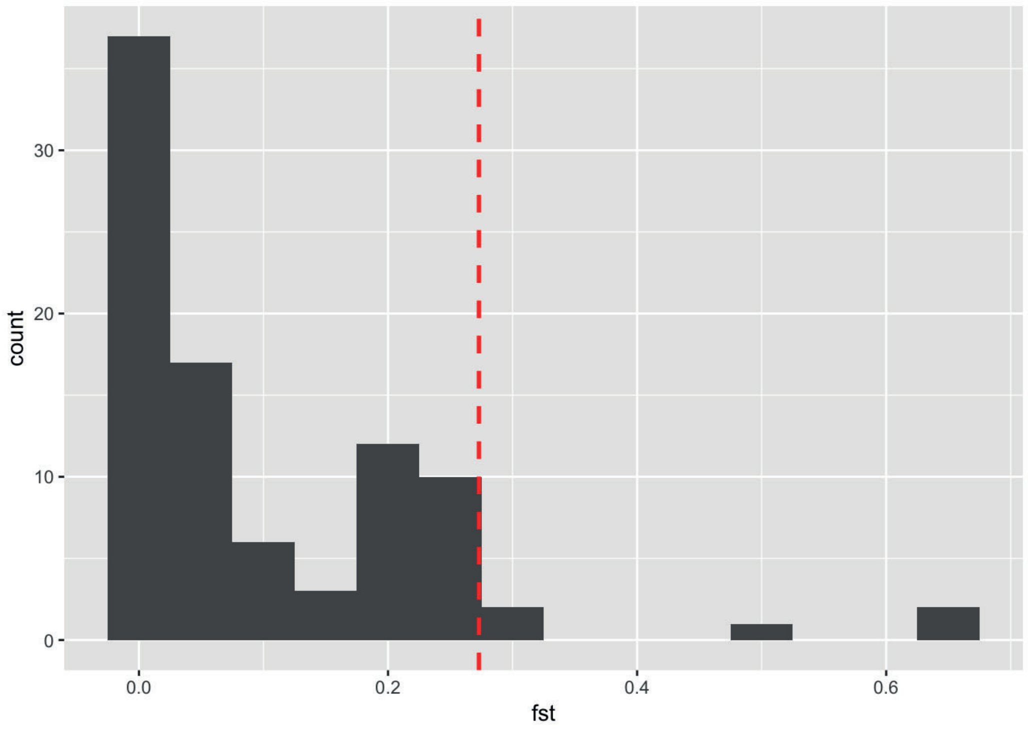
```
ggplot(lct_snps, aes(fst)) + geom_histogram(binwidth = 0.05)
```



Well, obviously it's not quite as nicely distributed as our simulated data was, but we can still use this to identify which values of F_{ST} might be considered potential outliers.

We'll set our threshold at 5% again and identify where on our distribution that falls. Note that this time we need to add `na.rm = T` to our `quantile` function in order to ignore some loci which have no F_{ST} estimates.

```
# set threshold
threshold <- quantile(lct_snps$fst, 0.95, na.rm = T)
# plot histogram with threshold marked
a <- ggplot(lct_snps, aes(fst)) + geom_histogram(binwidth = 0.05)
a + geom_vline(xintercept = threshold, colour = "red", lty = 2, size = 1)
```

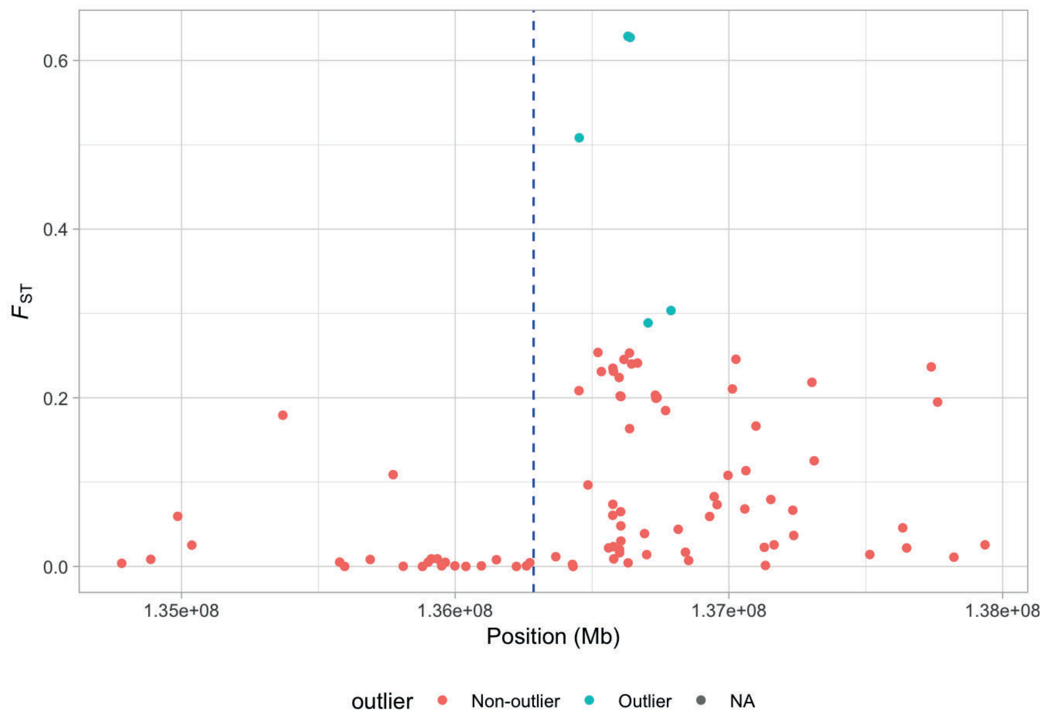


Now what if we want to visualise this on our chromosome-wide plot? Once again, we need to use the `mutate` and `if_else` functions.

```
lct_snps <- lct_snps %>%
  mutate(outlier = if_else(fst > threshold, "Outlier", "Non-outlier"))
```

Take a look at the `lct_snps` tibble - you should now see an additional column which is a character vector with the status of each locus as either outlier or non-outlier. Next we can incorporate this into our plotting:

```
a <- ggplot(lct_snps, aes(coord, fst, colour = outlier)) + geom_point()
a <- a + xlab("Position (Mb)") + ylab(expression(italic(F)[ST]))
a <- a + geom_vline(xintercept = lct_mid, lty = 2, col = "blue")
a <- a + theme_light() + theme(legend.position = "bottom")
a
```



So now our potential outlier SNPs are marked on the figure. There are only 5 of them but they all occur just upstream from the *LCT* locus.

We cannot say for certain that these SNPs have increased F_{ST} values because of selection - other processes such as genetic drift or demographic history (i.e. a bottleneck in one of the two populations) might be responsible. However, given our knowledge that *LCT* is involved in lactase persistence, we can at least hypothesise that this is the case.

One important point to note here is that the threshold we set to identify a SNP as being potentially under seelction is **entirely arbitrary**. In a way this line of thinking forces us to think of selection acting in some binary way on some SNPs and not others. This is obviously not the case. Still, for SNP data like this an F_{ST} scan can be a very useful tool.

Study questions

For study questions on this tutorial, download the `Chapter5_R_questions.R` from Canvas or find it here (https://evolutionarygenetics.github.io/Chapter5_R_questions.R).

Going further

- Graham Coop's notes on F statistics (http://cooplab.github.io/popgen-notes/#summarizing-population-structure)
- A detailed tutorial on calculating population differentiation with several R-based population genetic packages (https://popgen.nescent.org/2015-12-15-microsatellite-differentiation.html)
- A nice, thorough exploration of the normal distribution using R functions and plotting (http://seankross.com/notes/dpqr/)