Human Genome Variation Lab

January, 2023

Table of Contents

# 1 Homepage

This is the course homepage for Human Genome Variation with Computational Lab (AS.020.321).

#### 1.0.0.1 Class Info

Class is held on **Tuesdays from 3-3:50PM**, in **(location TBD)**.

Please bring your laptop with you to every class. If you don’t have a working laptop, contact me so that we can find a solution.

## 1.1 Syllabus & Logistics

Download a copy of the syllabus here.

### 1.1.1 Assessment & Grading

Final grades will be determined through either:

1. A comprehensive final exam that is graded by the instructors, or
2. A self-grade based upon criteria set forth in your self-evaluations.

To qualify for self-grading, students must demonstrate participation in 12 out of 13 class sessions.

### 1.1.2 Homework

There are required and optional homework assignments for each topic we cover.

## 1.2 Course Description

The Human Genome Variation course exposes you to the power of genomic studies for understanding human evolutionary history, as well as revealing the genetic basis of human traits and disease.

What does real human genomic data look like? How are these data analyzed in practice? Supplementing the main course, this lab module will explore public datasets and computational tools used to analyze human genomic data to better understand how patterns in these data can be used to test hypotheses about evolution and human phenotypes.

### 1.2.1 Educational Objectives

* Explore the ways in which human genomic data is generated, encoded, summarized, and visualized.
* Develop an awareness of potential confounding factors and approaches by which they can be overcome.
* Establish familiarity working with summarized forms of genomic data in R, as well as resources for further independent learning.

## 1.3 RStudio Cloud

All the coding and lectures for this class are conducted on RStudio Cloud.

RStudio is a computing environment for **R**, one of the most widely used coding languages in genomics and statistics.

**RStudio Cloud** is an online version of the RStudio environment. This cloud workspace avoids us having to install R on everyone’s computers, and also allows us to upload and share data files for class.

### 1.3.1 Making an RStudio Cloud account

Go to the [RStudio Cloud website](https://login.rstudio.cloud/register) and sign up for an account, using your JHU email.

Then, join the HGV Lab workspace.

# 2 Genome browsers

In this module, we’ll learn how to use the **UC Santa Cruz (UCSC) Genome Browser** and the **Integrative Genomics Viewer (IGV)**, two extremely popular tools for visualizing genomic data.

#### 2.0.0.1 Learning objectives

After completing this chapter, you’ll be able to:

1. Explain why a reference genome is an important resource for genomics research.
2. Use the UCSC genome browser to find genomic features in a region of interest.
3. Describe the data contained in a file of sequencing reads.
4. Load and interpret sequencing data in IGV.

## 2.1 DNA sequencing data

These days, the vast majority of genomic data is generated through **high-throughput Illumina short-read sequencing**. The broad steps of this sequencing process are:

* Extract DNA
* Fragment DNA
* Prepare for sequencer (add adapters, etc.)
* Amplify DNA
* Sequencing (add fluorescently labeled nucleotides that are read by a digital camera)

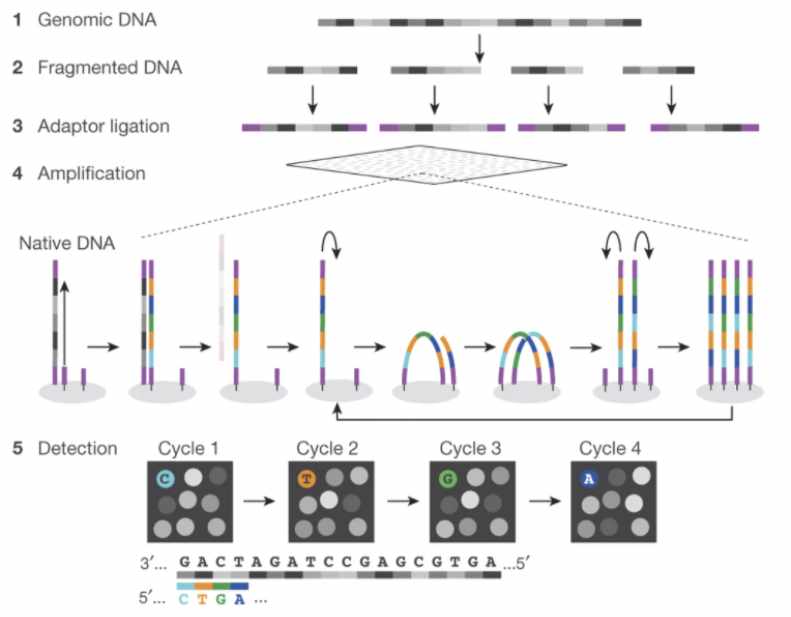


Figure : **Fig. 1 (**[**source**](https://www.researchgate.net/figure/Schematics-of-the-Illumina-and-ONT-sequencing-procedure-A-Illumina-sequencing-1-2_fig1_364792307)**).** Schematic of Illumina short-read sequencing.

This sequencing approach is fast and cost efficient, but introduces two main limitations.

1. Because of the fragmentation step, the resulting sequencing reads are extremely short (~150 bp).
2. We don’t know where in the genome the sequencing reads came from. (This is a limitation common to nearly every sequencing experiment.)

## 2.2 Assembling a genome

When the human genome is 3 billion base pairs long, assembling short sequencing reads into a full genome is a major computational challenge.

How is genome assembly performed?

We can combine sequencing reads that partially overlap with each other into longer sequences.

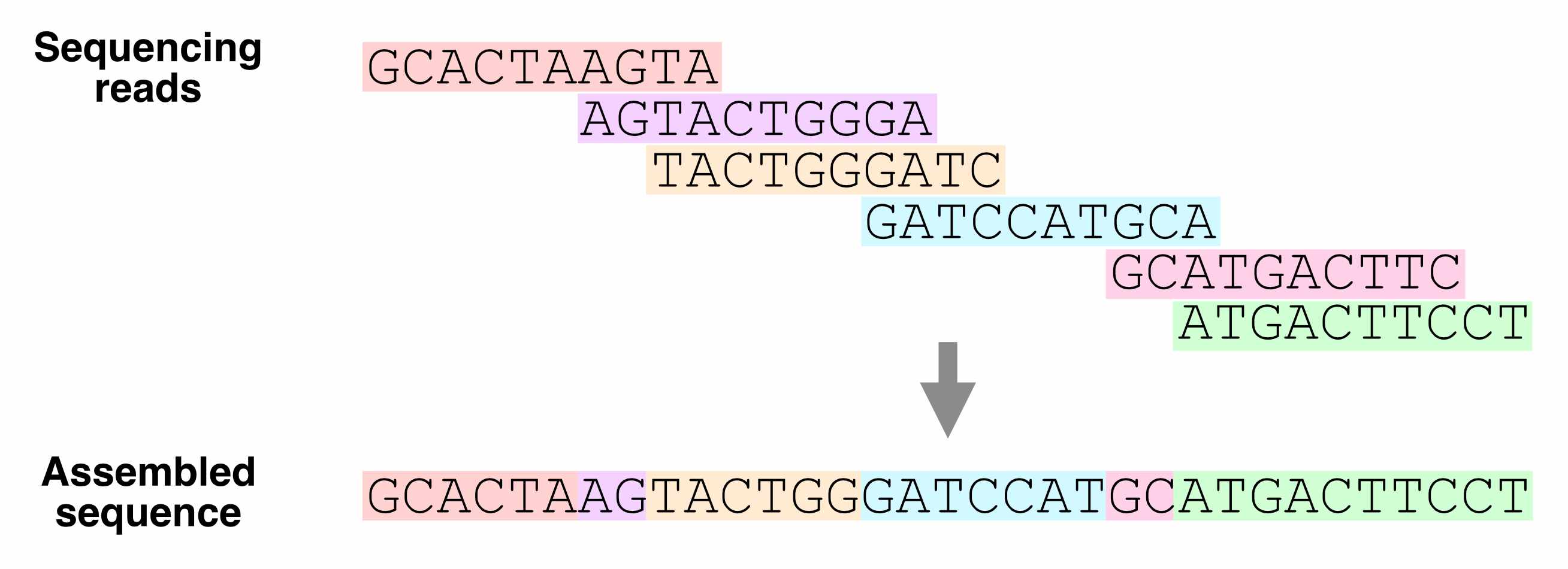


Figure : **Fig. 2.** Using overlapping sequencing reads to assemble a genome.

Which regions of the genome are hardest to assemble?

Ideally, with enough sequencing data, we would be able to reconstruct an entire genome from overlapping reads. In practice, genome assembly is complicated by **repetitive DNA** – sequences in different regions of the genome that are completely or nearly identical.

These repeats make it difficult (or impossible) to determine the order of the sequences around them, or how many copies of the repeat there are.

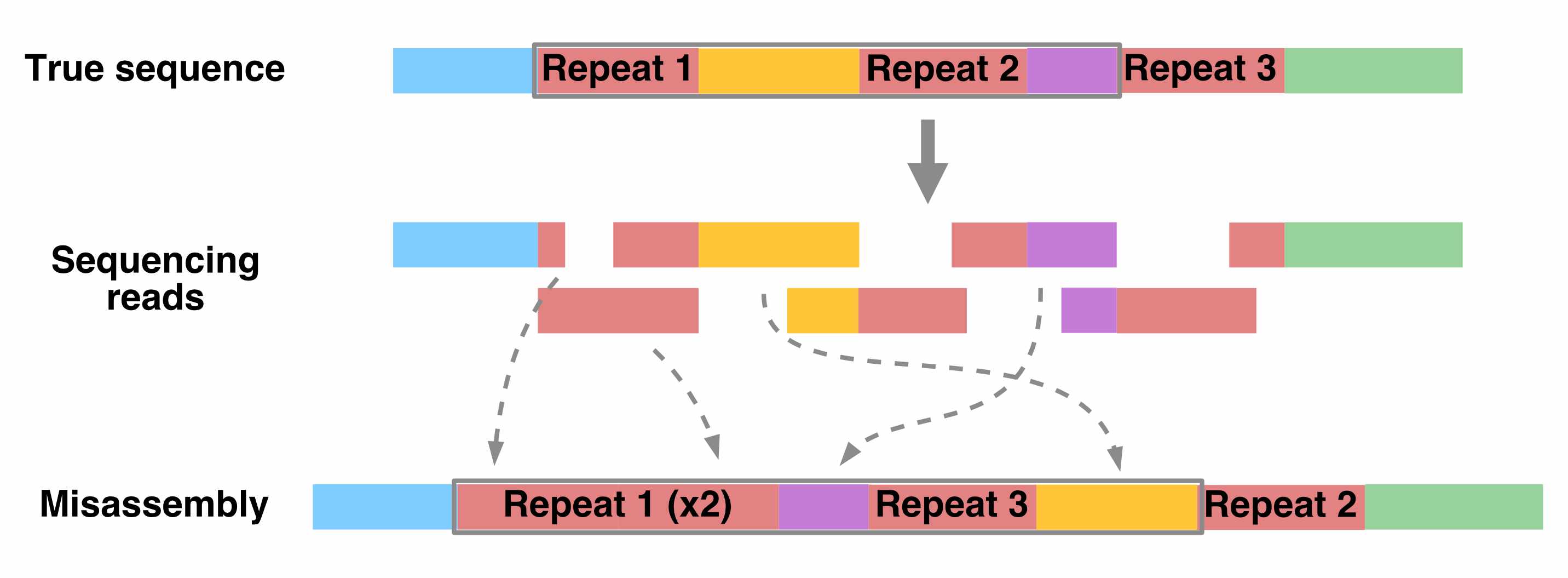


Figure : **Fig. 3.** How repetitive sequences affect genome assembly.

Resolving repetitive regions requires sequencing reads that are longer than the repeat itself, which allow us to determine the flanking sequences on the sides of the repeat.

Using such long-read sequencing technology (i.e., [PacBio](https://www.pacb.com/technology/hifi-sequencing/) and [Nanopore](https://nanoporetech.com/applications/dna-nanopore-sequencing) sequencing), the [Telomere-to-Telomere consortium](https://www.science.org/doi/10.1126/science.abj6987) was able to create a complete, ungapped assembly of the human genome in 2021.

## 2.3 The human reference genome

Having to assemble an entire genome every time you sequence a new individual is a hassle (and often infeasible, if you don’t have enough sequencing data). Instead, we typically align sequencing reads to a **reference genome** – a high-quality genome assembly for that species, which we use to guide our analysis.

The human reference genome was initially assembled in 2000 by the [Human Genome Project](https://www.genome.gov/human-genome-project), and has undergone decades of refinement since. The current version of this reference, which we’ll be using, is **hg38**.

Whose DNA was sequenced for the human reference genome?

DNA from multiple individuals was sequenced to construct the reference genome. Its sequence is a mosaic of these individuals’ DNA.

You can classify the ancestry of different parts of hg38 by comparing its sequence to DNA from different populations. From this, we know that around 70% of hg38 comes from one individual, called RP11, who likely had African American ancestry.

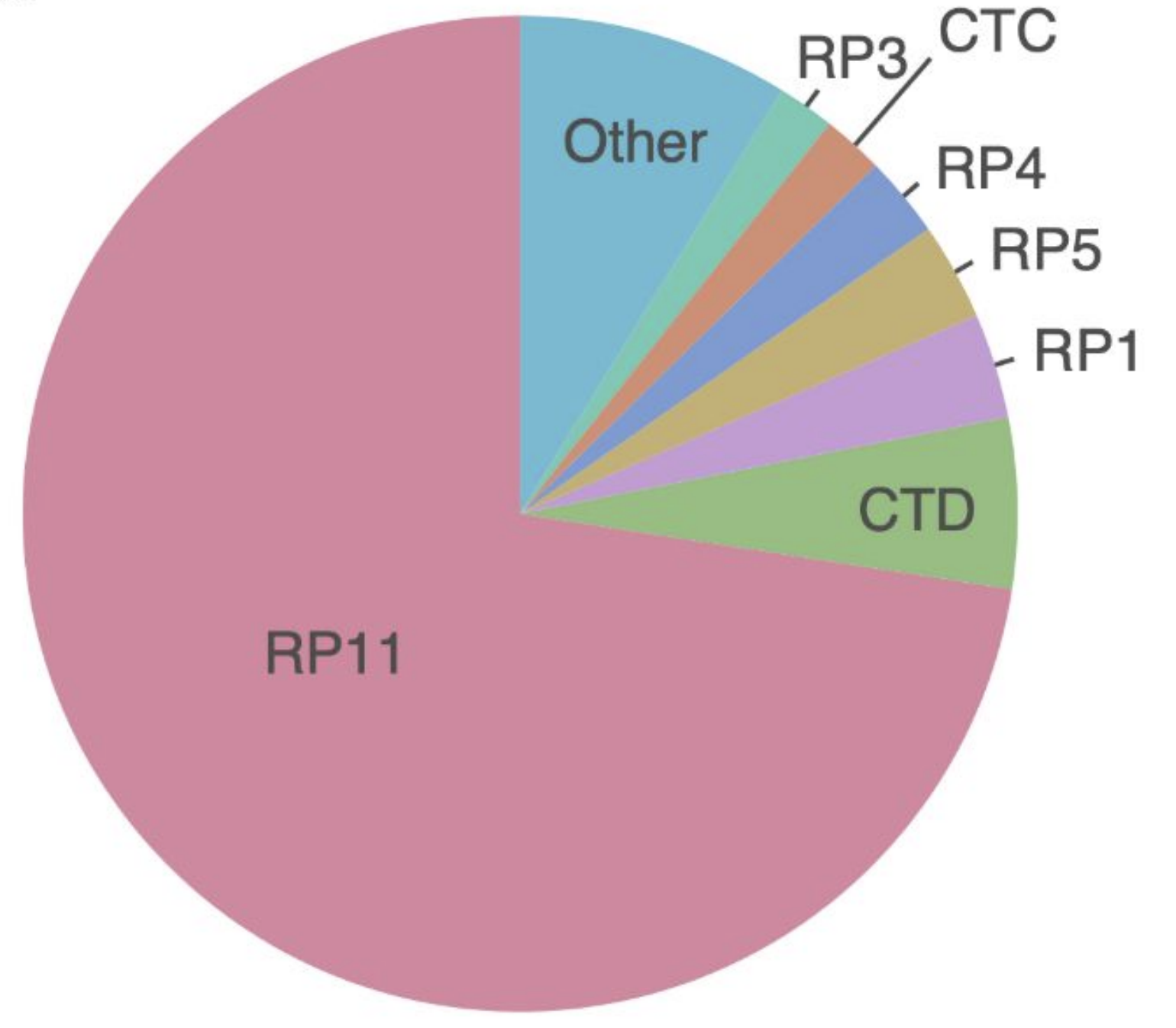


Figure : **Fig. 4.** Sample composition of the human reference genome.

## 2.4 UCSC genome browser

What does the human reference genome actually look like? We can view it in the **UC Santa Cruz (UCSC) genome browser**, an interactive website for viewing genomes – both the human reference and reference genomes for several other species.

The browser also displays **genome annotations**, such as the locations of genes and clinically relevant genetic variants.

Go to the [**UCSC genome browser**](https://genome.ucsc.edu/cgi-bin/hgGateway?hgsid=1537534853_Fxba671lul6Qu1chkXaoY68Kr3lP).

### 2.4.1 Homepage

There are a few key areas of this page:

* Browse/Select Species – choose the species
* Human Assembly – choose the version of the human reference genome
* Position/Search Term – type in a specific position (ex: chr2:25160915), region (ex: chr1:100000-200000), or gene name (ex: HLA-A)

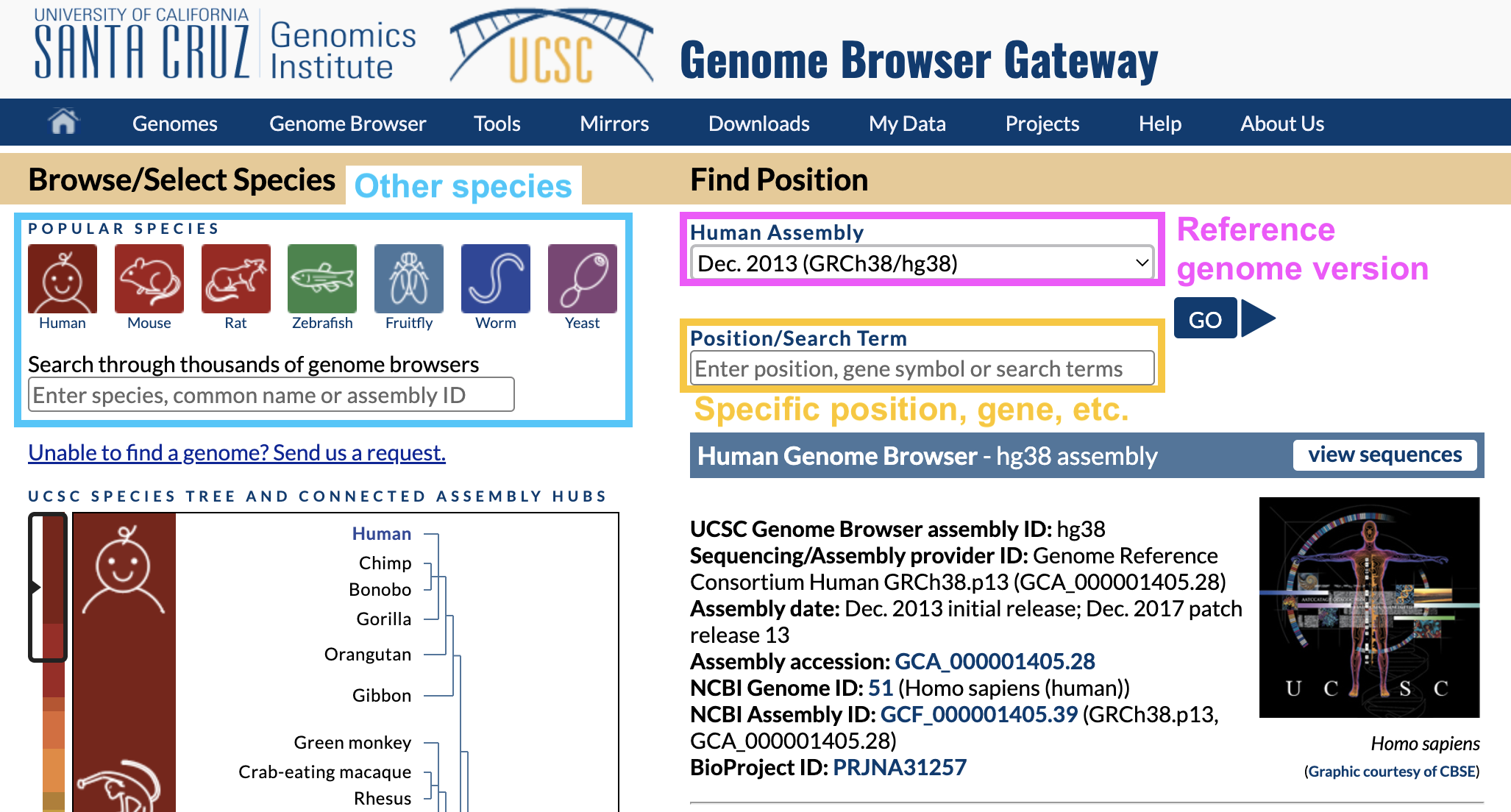


Figure : **Fig. 5.** UCSC genome browser homepage.

In Position/Search Term, type in a gene you’re interested in and hit enter.

Why are there different versions of the reference genome?

You may have noticed that the name of the reference genome we’re using is **GRCh38** or **hg38**, which stands for Genome Reference Consortium Human Build 38 – version 38 of the reference genome.

Over time, the Genome Reference Consortium makes improvements to the reference genome by closing gaps, fixing problems, and resolving repetitive regions. hg38, the most recent version, was released in 2013.

## 2.5 Viewing one region of the genome

Once you hit enter, you should end up on a page like this:

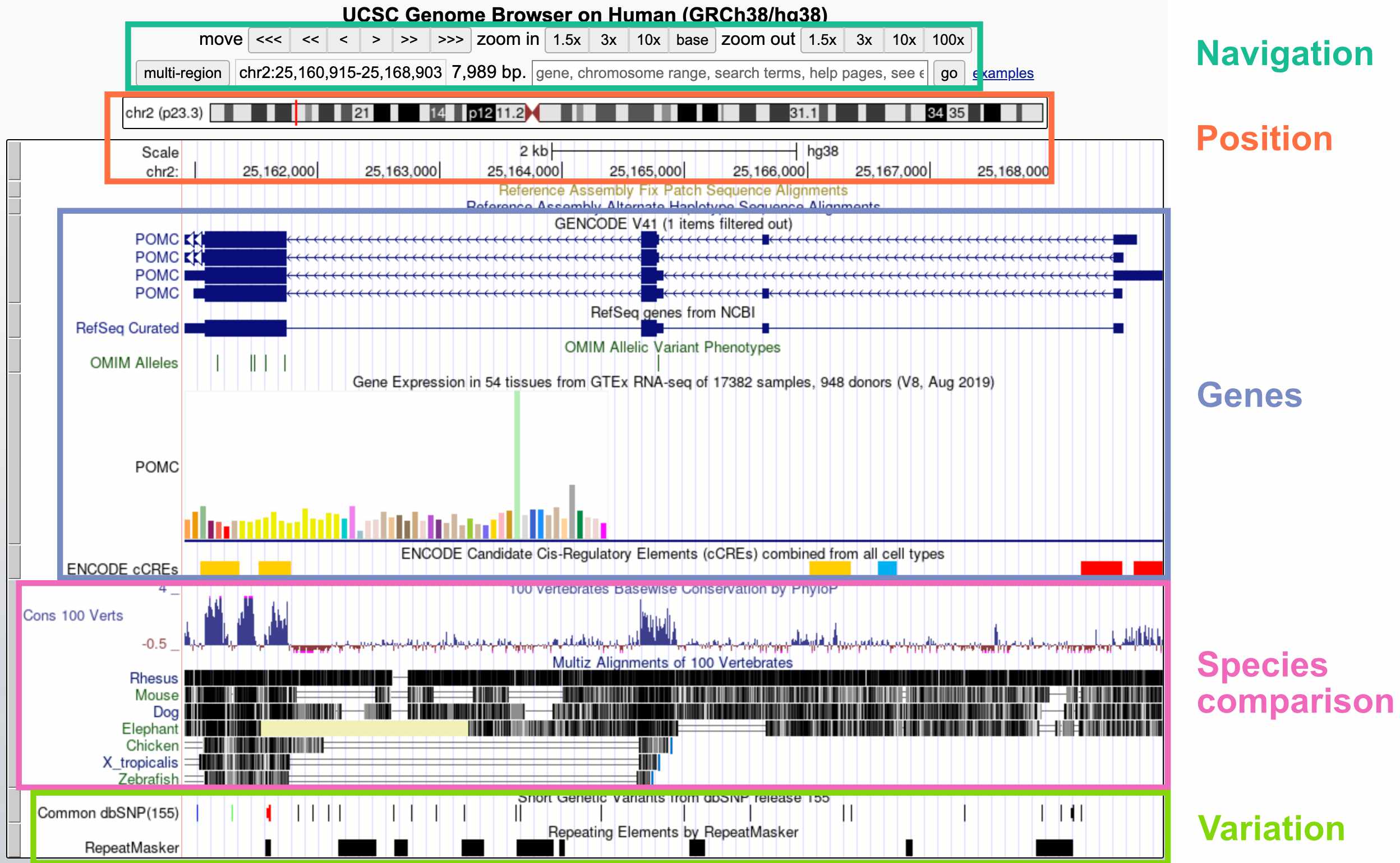


Figure : **Fig. 6.** Viewing one genomic region in the UCSC genome browser.

The default display includes these broad groups of annotations:

* **Navigation**: Buttons for zooming and moving around (you can also move by clicking the display, holding, and moving your mouse); current region; search bar
* **Position**: Current position on the chromosome; current base pair position
* **Genes**: Gene annotations; gene expression by tissue; gene regulatory elements (CREs)
* **Species comparison**: DNA sequence conservation across vertebrates; regions that align with the genomes of other vertebrates
* **Variation**: Genetic variants in the dbSNP database; repeat elements

Inspecting a specific track

If you’re interested in more information about a specific track – for example, the *POMC* gene annotation – you can click on that element to go to a webpage with more details.

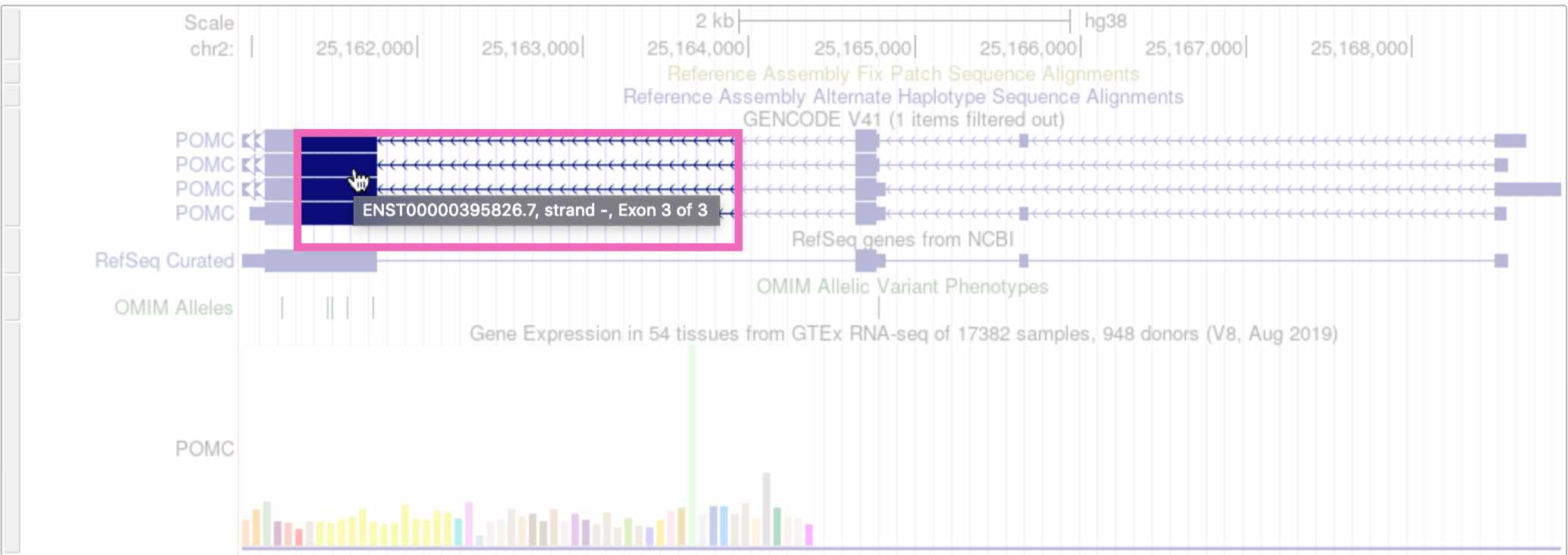


Figure : **Fig. 7.** Clicking on the *POMC* gene track.

(Note that if you click on whitespace instead of an annotation element, it will change the track’s display density instead.)



Figure : **Fig. 8.** Clicking on whitespace to expand the RefSeq genes track.

Customizing the display tracks

The tracks that are automatically displayed are just a small subset of what’s available. You can select which tracks you want to see, and set their display density, by scrolling down on the page.

To add a new track to your browser view, click the drop-down menu below that track and select any of the options besides hide. Here we’re viewing the “Clone Ends” track, which shows the different individuals that were sequenced to create this section of the reference genome.



Figure : **Fig. 9.** Adding the “clone ends” track to the browser.

Click the refresh button in the upper right to reload the genome view. You should see something like this, showing that this region of the reference genome was sequenced in three individuals (CH17, CTD, and RP11):



Figure : **Fig. 10.** Viewing the “clone ends” track.

## 2.6 IGV

While the UCSC genome browser lets you view the reference genome itself, we’re often interested in looking at sequencing data – sequencing reads that are aligned to the reference genome. For this we use the **Integrative Genomics Viewer (IGV)**.

Go to the [IGV web app](https://igv.org/app/).

### 2.6.1 Homepage

The IGV homepage is fairly empty because we haven’t loaded any sequencing reads to look at, and also because we’re zoomed too far out to see anything.

* The Genome tab in the upper right lets you choose which reference genome to work in. The default is **hg38**
* A drop-down menu and search bar below the header allow you to pick a chromosome and genomic position

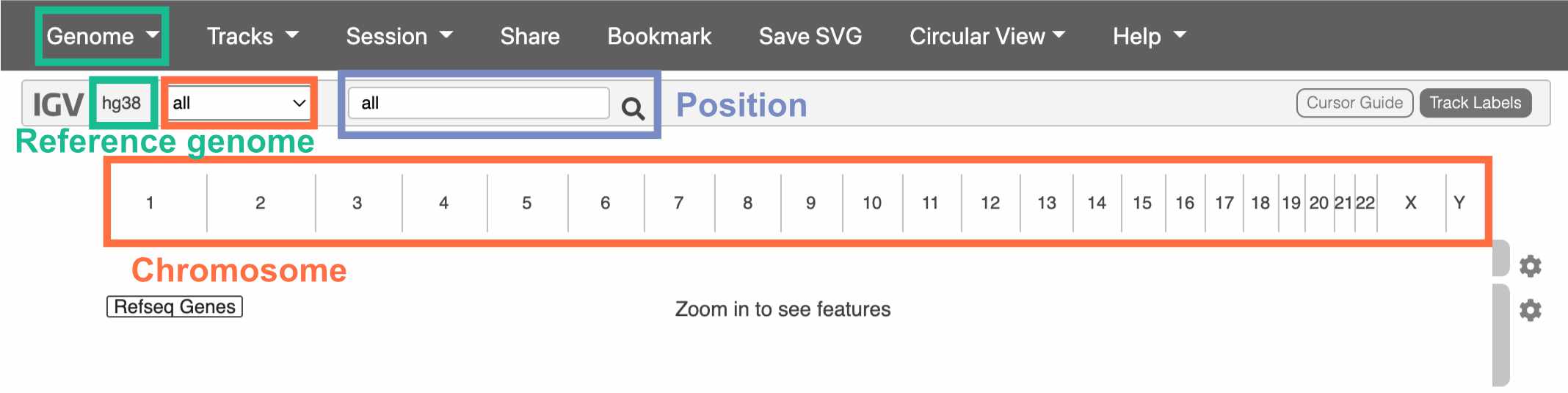


Figure : **Fig. 11.** The IGV homepage.

We haven’t chosen a chromosome yet, so all of them are displayed below the drop-down menu. Click on one to go to a zoomed-in view of that particular chromosome.

## 2.7 Navigating IGV

Once you’ve clicked on a chromosome, zoom in until you can see colors on the top track. This track displays the DNA sequence, colored by nucleotide.

The track below the DNA sequence has gene annotations from RefSeq.

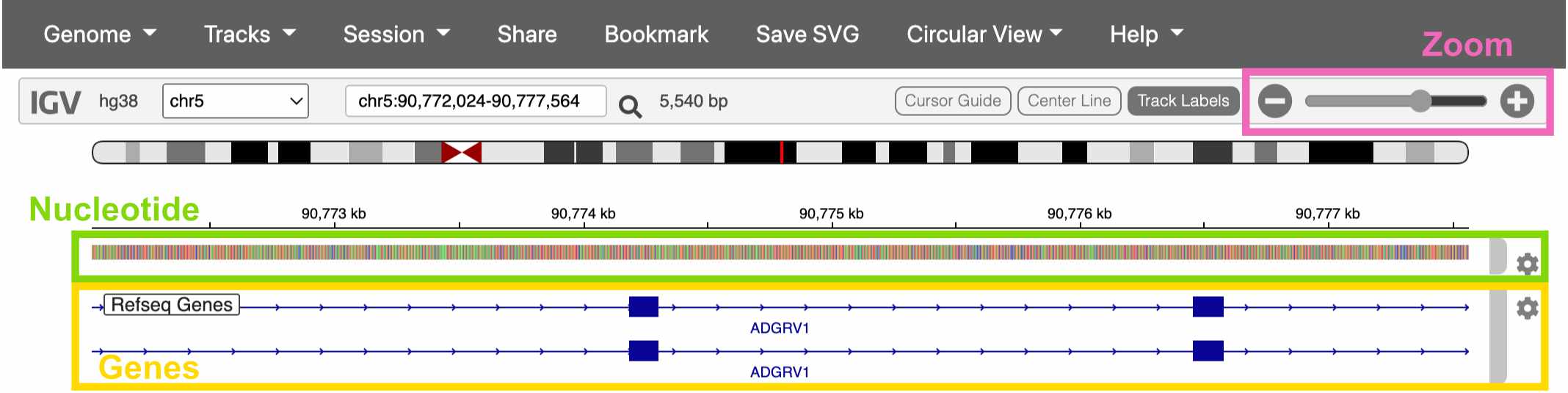


Figure : **Fig. 12.** Viewing a gene in IGV.

## 2.8 Loading sequencing data

Click on the Genome drop-down menu and switch to the Human (hg38 1kg/GATK) reference genome. This version of the hg38 reference has sequencing data already loaded into the IGV web app.

Once you’ve switched references, click Tracks -> 1KG Low Coverage Alignments. This gives you a list of sample to load sequencing data from. Click any sample and then OK.

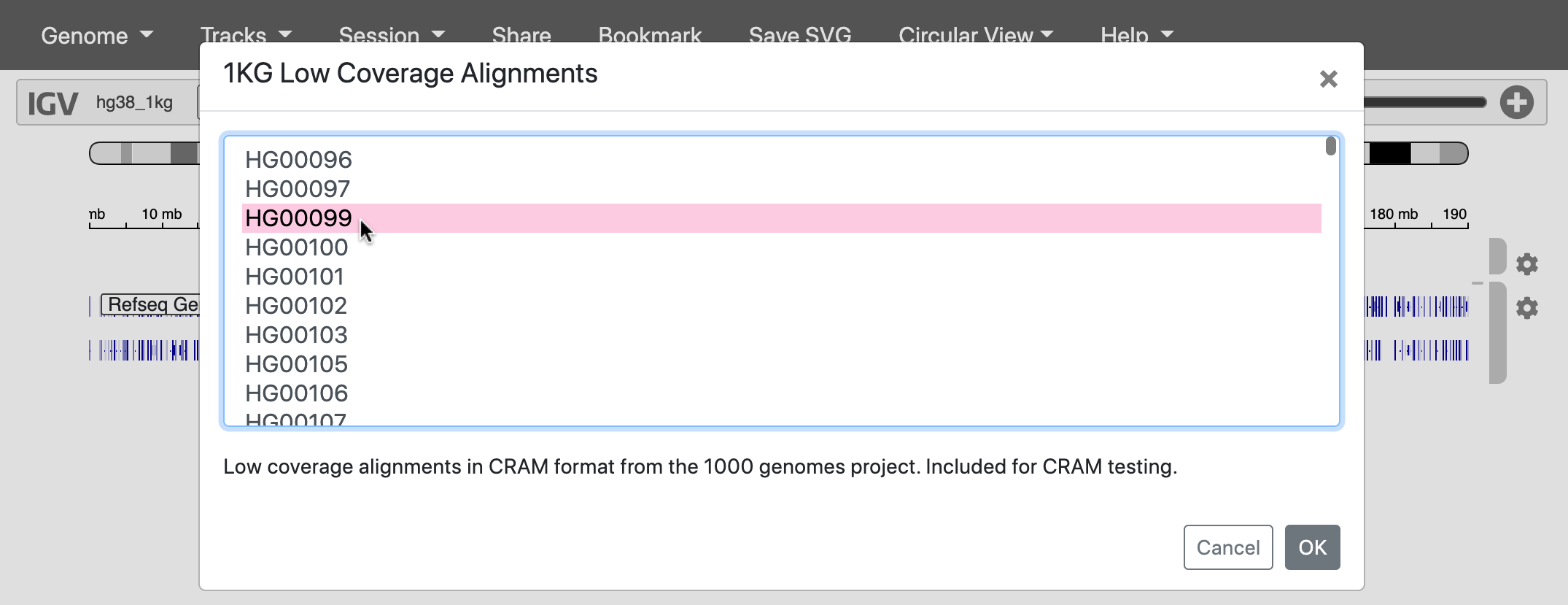


Figure : **Fig. 13.** Loading reads from a 1000 Genomes sample.

## 2.9 The 1000 Genomes Project

Where did this sequencing data come from?

In 2015, a study called the [**1000 Genomes Project**](https://www.nature.com/articles/nature15393) **(1KG or 1KGP)** sequenced 3,202 individuals from 26 globally diverse populations. Because this data is publicly available, it’s become one of the most widely used datasets in human genetics.

Notably, 1KGP still excludes key regions of the world – such as Oceania, the Middle East, native American populations in North America, and many populations within Africa.

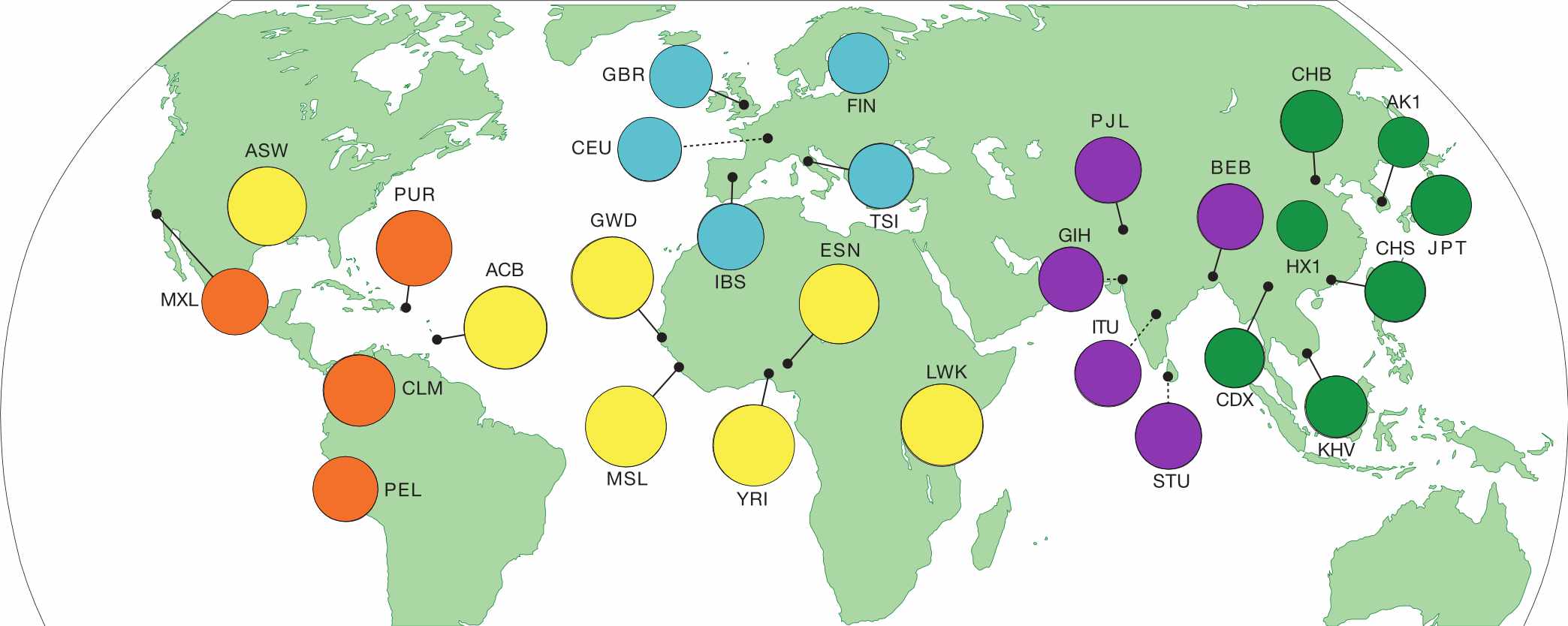


Figure : **Fig. 14.** Regions sampled by the 1000 Genomes Project.

Go to the [**1000 Genomes Project** website](https://www.internationalgenome.org/home) and click the Data tab. Then click the link to the data portal.

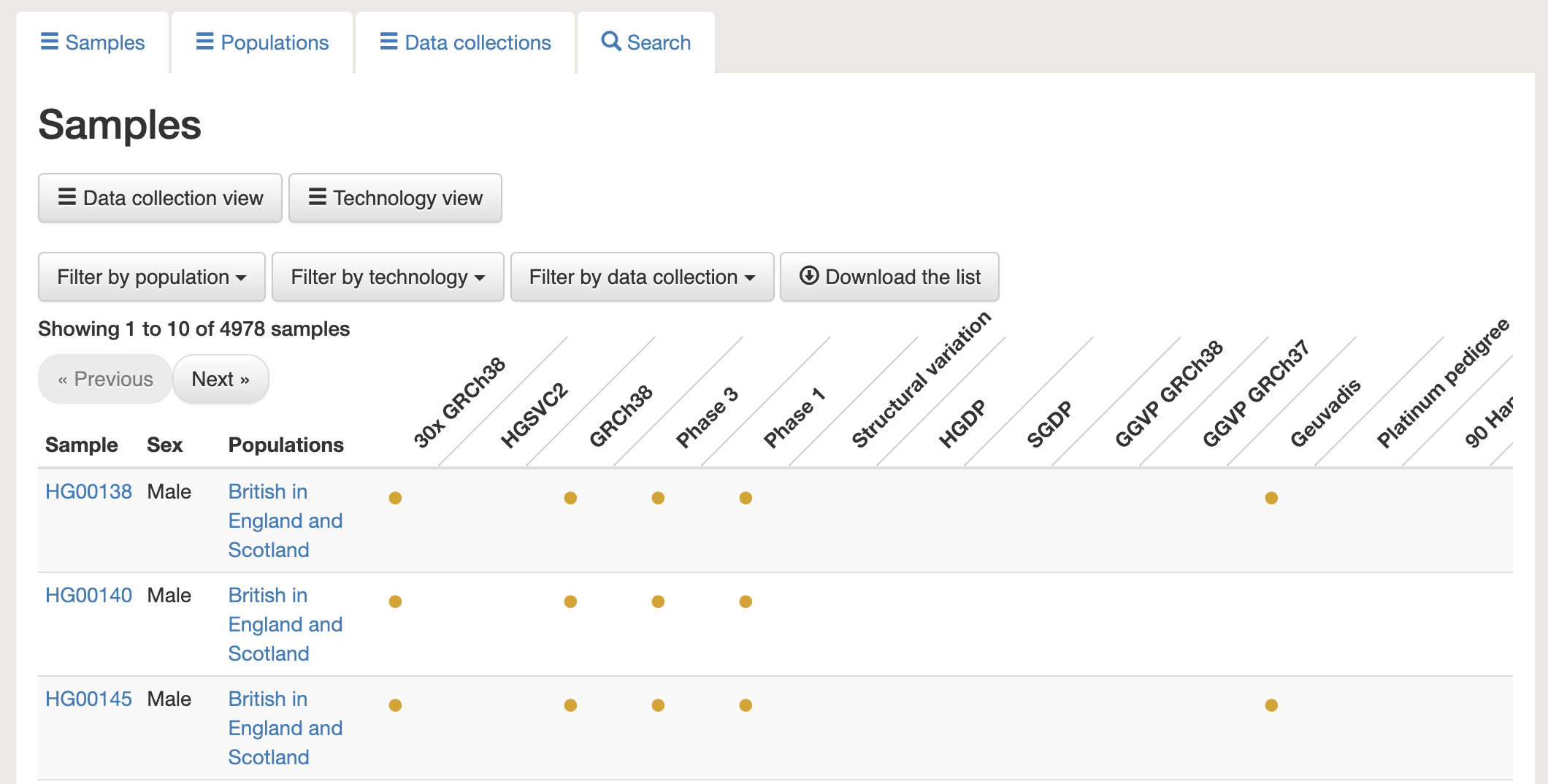


Figure : **Fig. 15.** The data portal includes information about the samples in this dataset.

Choose any individual and copy their sample ID (ex: **HG00138**). We can use this sample ID to find this individual’s raw sequencing data in the [**Sequence Read Archive (SRA)**](https://www.ncbi.nlm.nih.gov/sra).

## 2.10 SRA

Search for the sample ID you chose in SRA. You should see something like this, where every item is a sequencing dataset generated for this sample.

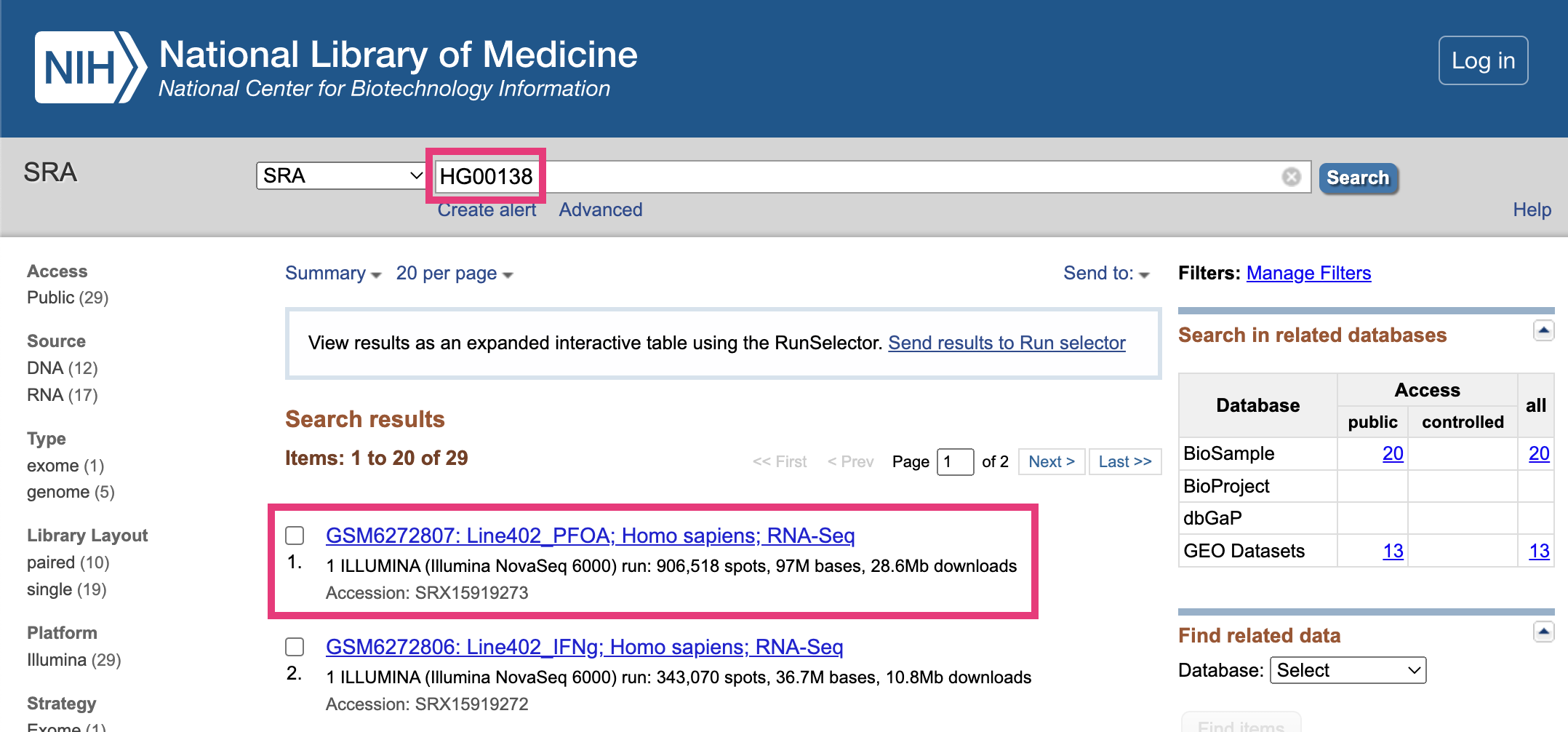


Figure : **Fig. 16.** Finding sequencing data in SRA.

### 2.10.1 Previewing sequencing data

Choose any sequencing dataset, and then click on any item in the **Run** table at the bottom.

This takes you to a page that displays a specific sequencing run (i.e., one use of a sequencing machine). Go to the **Reads** tab.

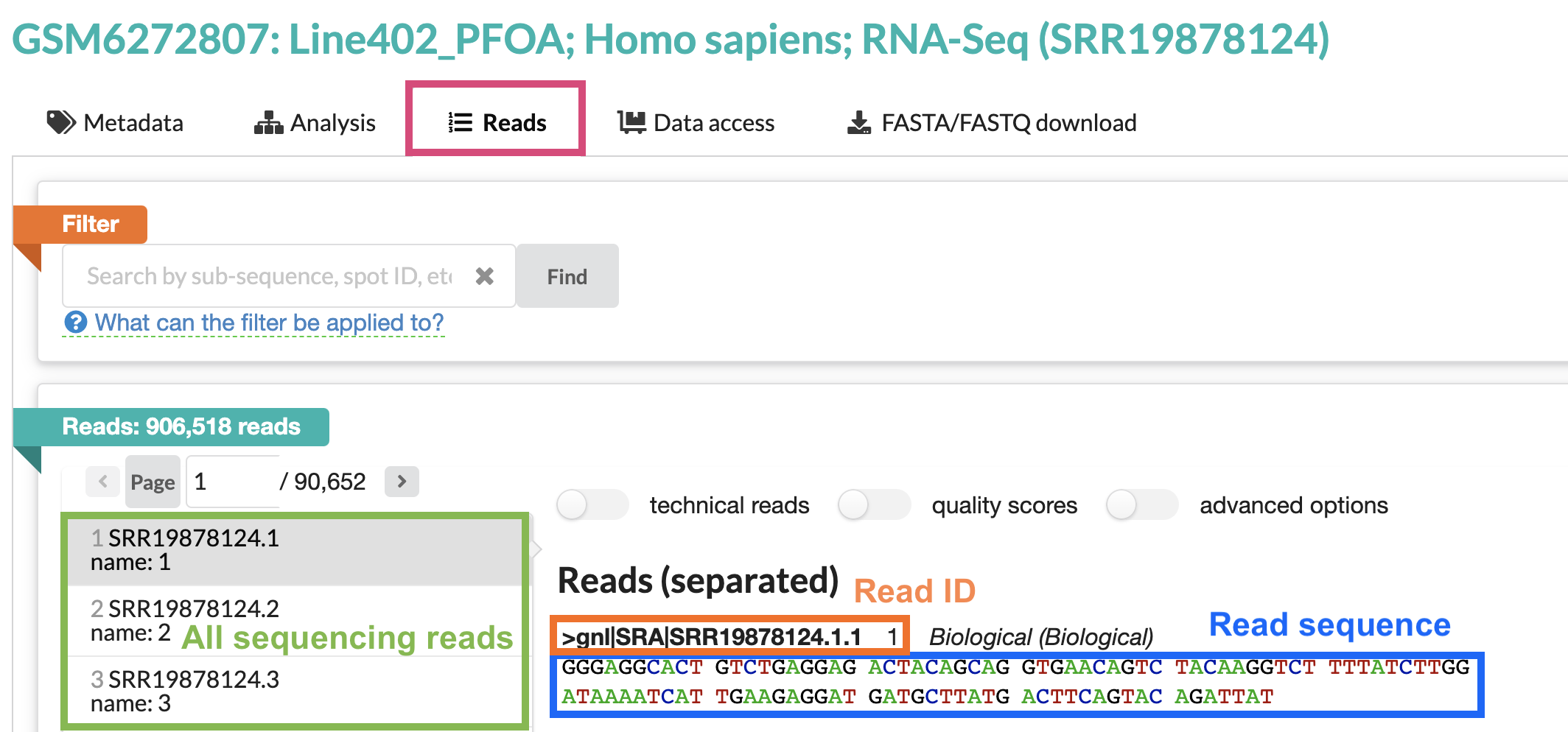


Figure : **Fig. 17.** The Reads tab in SRA.

The right-hand panel shows one sequencing read from this run. Note that the sequence of this read is around 100bp long – the average length for short-read Illumina sequencing.

All sequencing data looks like this! It’s just a text file filled with the IDs and DNA/RNA sequences of your reads.

## 2.11 Viewing sequencing reads in IGV

Now that we’ve seen what raw sequencing data looks like, let’s look at it in IGV.

Return to your IGV tab, where you should have one sample’s DNA sequencing data loaded. Make sure you’re zoomed in enough for the data to display.

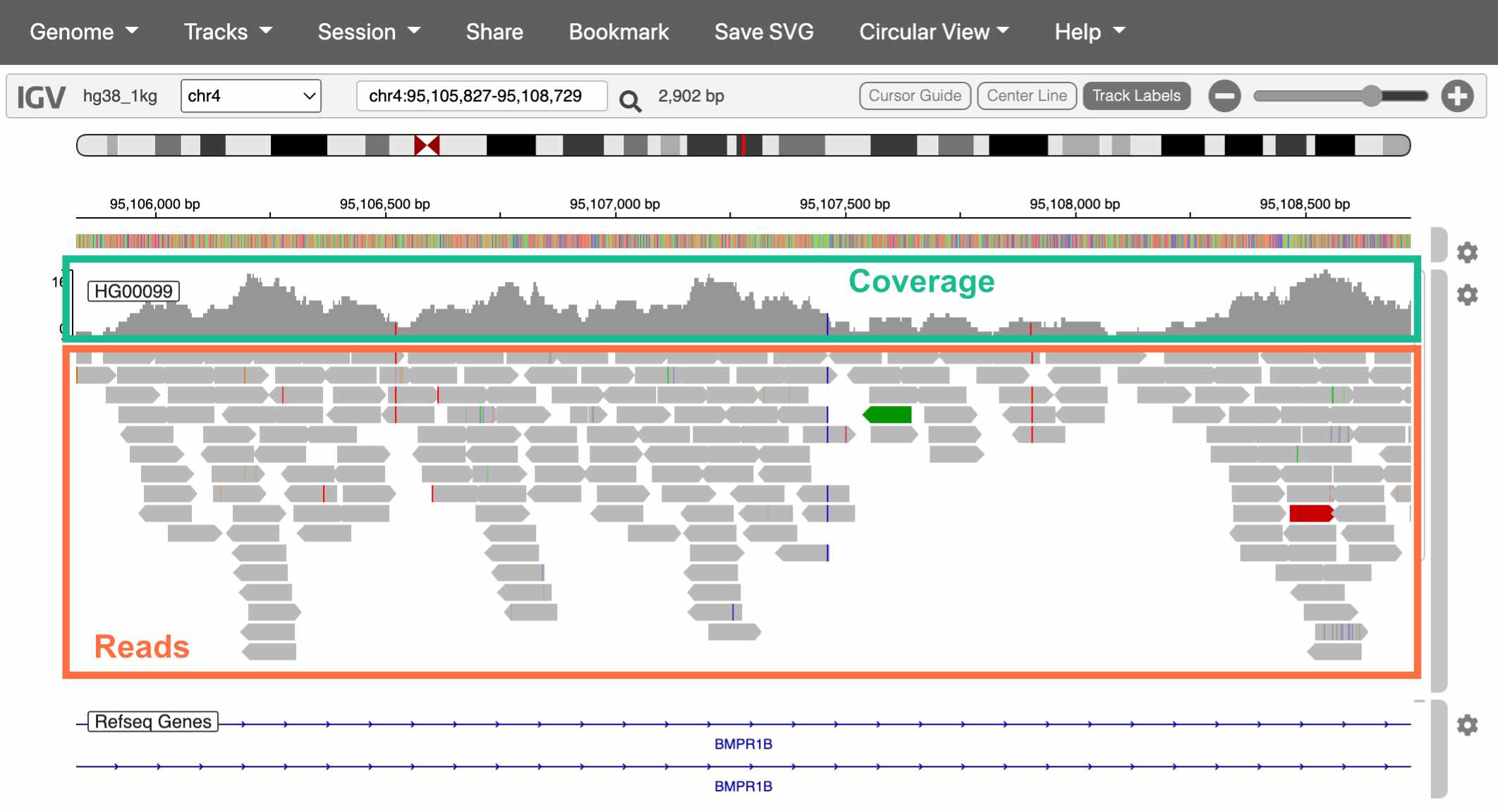


Figure : **Fig. 18.** The Reads tab in SRA.

The top track is a histogram of sequencing coverage (i.e., how many reads there are at that position in the genome). The bottom track shows the reads themselves.

How do we know where in the genome each read belongs?

We match the sequence of the read to the sequence of the reference genome (called **alignment**). With 100bp reads, the probability that a match occurs by chance is , or .

Extracting alignment information in IGV

If you click on a specific read, IGV will display additional information about it, including:

* The exact position it aligns to
* The mapping quality (a score indicating how uniquely it aligns to this position)
* If you’re working with paired-end sequencing data, where its paired read is

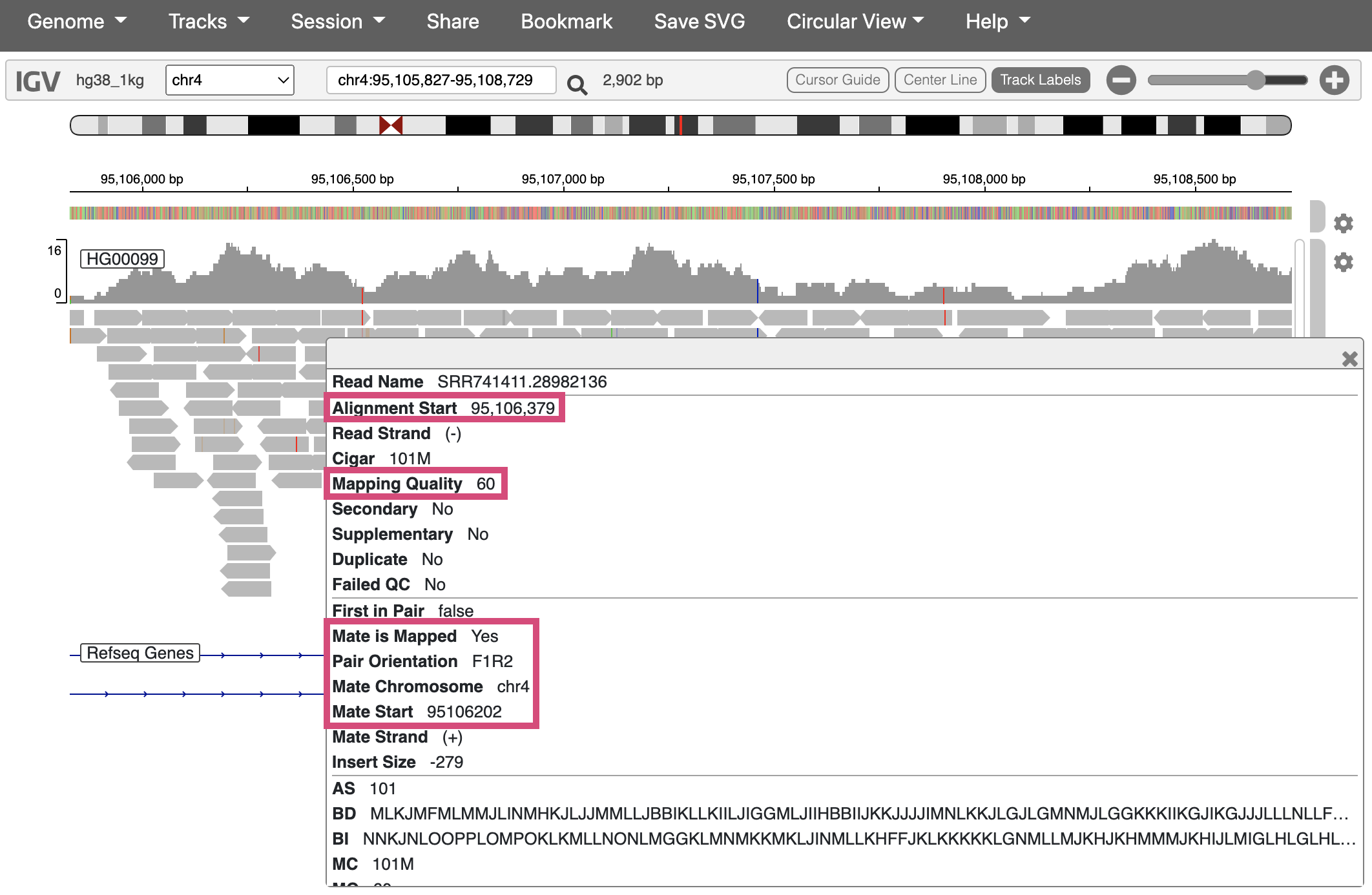


Figure : **Fig. 19.** Viewing additional info for one sequencing read.

## 2.12 Interpreting IGV alignments

Sequencing reads in IGV are colored at bases where they differ from the reference genome. These differences can be caused by either real genetic variation or sequencing error. **How would you distinguish these two?**



Figure : **Fig. 20.** Two of these colored bases are probably real SNPs, and two are probably errors.

The sequencing coverage track also colors the positions that it thinks are real variants.

In the screenshot above, which spans about 2kb, there are two SNPsin the coverage track. This pattern holds more broadly through the genome – humans carry about **one SNP every 1,000 bases**.

Is one SNP every 1,000bp a lot or a little?

Humans actually have much lower amounts of genetic variation than many species, [including many of the great apes](https://www.nature.com/articles/nature12228).

This is mostly the result of human evolutionary history. Because the effective size of human populations has historically been low, with only very recent expansion, the gene pool is still fairly homogenous, with many rare variants and few common ones.

## 2.13 Conclusion

In this lab, we explored several of the most commonly used websites in genomics:

#### 2.13.0.1 Genome browsers

* [**UCSC genome browser**](https://genome.ucsc.edu/): Used to explore features of the human genome
  + If you discover an interesting SNP in your research, you might look it up in the UCSC browser to see which genes it’s in/near, if it overlaps with any repetitive elements, etc.
* [**IGV**](https://igv.org/app/): Used to visualize sequencing data
  + It’s common practice to look at your sequencing reads in IGV to check alignment quality, verify that SNPs look like real variants and not errors, etc.

#### 2.13.0.2 Data repositories

* [**1000 Genomes Project**](https://www.internationalgenome.org/home): One of the largest and most diverse datasets of human sequencing data
  + Data from 1000 Genomes is frequently used in human genetics studies
* [**SRA**](https://www.ncbi.nlm.nih.gov/sra): A repository for publicly available sequencing data
  + Genetics studies deposit their data in SRA if it can be made publicly available (i.e., if it has no identifiable information)

## 2.14 Homework

#### 2.14.0.1 Goals & Learning Objectives

The goal of this homework is to make an account in RStudio Cloud to use for the rest of the semester.

### 2.14.1 Required homework

Follow the instructions [here](https://stephaniemyan.github.io/hgv_modules/rstudio-cloud.html) to create an RStudio Cloud account and join the HGV workspace.

# 3 Discovering mutations

In this module, we’ll use DNA sequencing data from human families to explore the relationship between parental age and *de novo* mutations in their children.

#### 3.0.0.1 Learning objectives

After completing this chapter, you’ll be able to:

1. Create plots to visualize the relationship between two variables.
2. Interpret the results of a linear model.
3. Compare the impact of maternal vs. paternal age on *de novo* mutation counts.
4. Explain what a confidence interval is and why it’s useful.

## 3.1 *De novo* mutations

Mutation and recombination are two biological processes that generate genetic variation. When these phenomena occur during gametogenesis, the changes that they make to DNA are passed down to the next generation through germline cells (i.e., sperm and oocyte).

***De novo* mutations (DNMs)** arise from errors in DNA replication or repair. These mutations can be single-nucleotide polymorphisms (SNPs) or insertions and deletions of DNA. Every individual typically carries around 70 *de novo* SNPs that were not present in either of their parents.

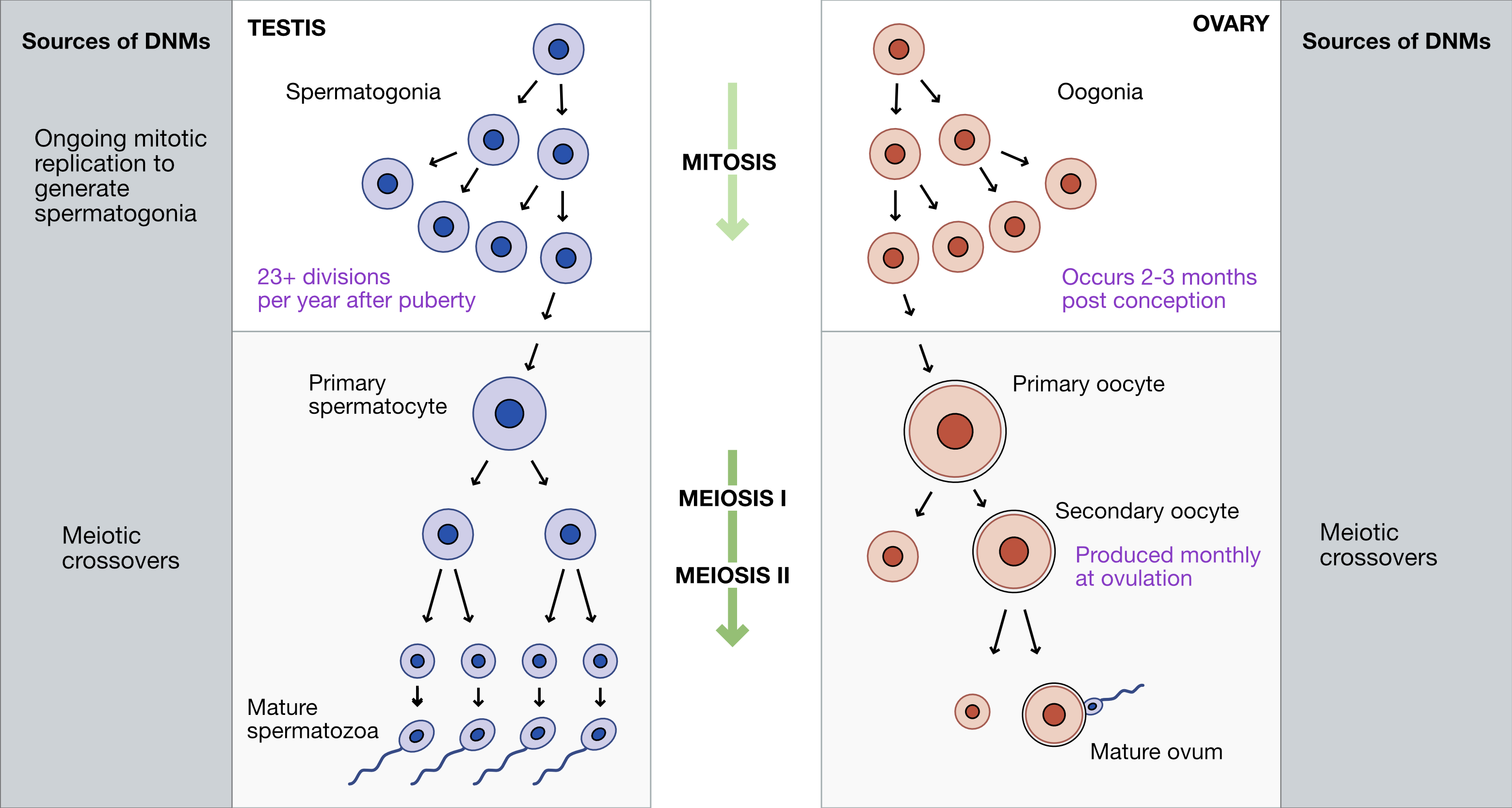


Figure : **Fig. 1.** Sources of DNMs in gametogenesis.

## 3.2 Recombination

**Crossovers**, or meiotic **recombination**, occur during prophase of meiosis I, when homologous chromosomes pair with each other. Double-strand breaks are deliberately generated in the DNA, and are then cut back and repaired based on the sequence of the homologous chromosome. These repairs can sometimes resolve in a crossover event, where sections of DNA are swapped between chromosomes.

Because the sequences of homologous chromosomes differ at sites where they carry different alleles, recombination generates genetic diversity by creating new haplotypes, or combinations of alleles.

Crossovers are required for meiosis because they ensure proper homologous chromosome pairing and segregation (although there are exceptions in some organisms, like male fruit flies). Humans experience 1-4 crossover events per chromosome, with longer chromosomes having more crossovers.

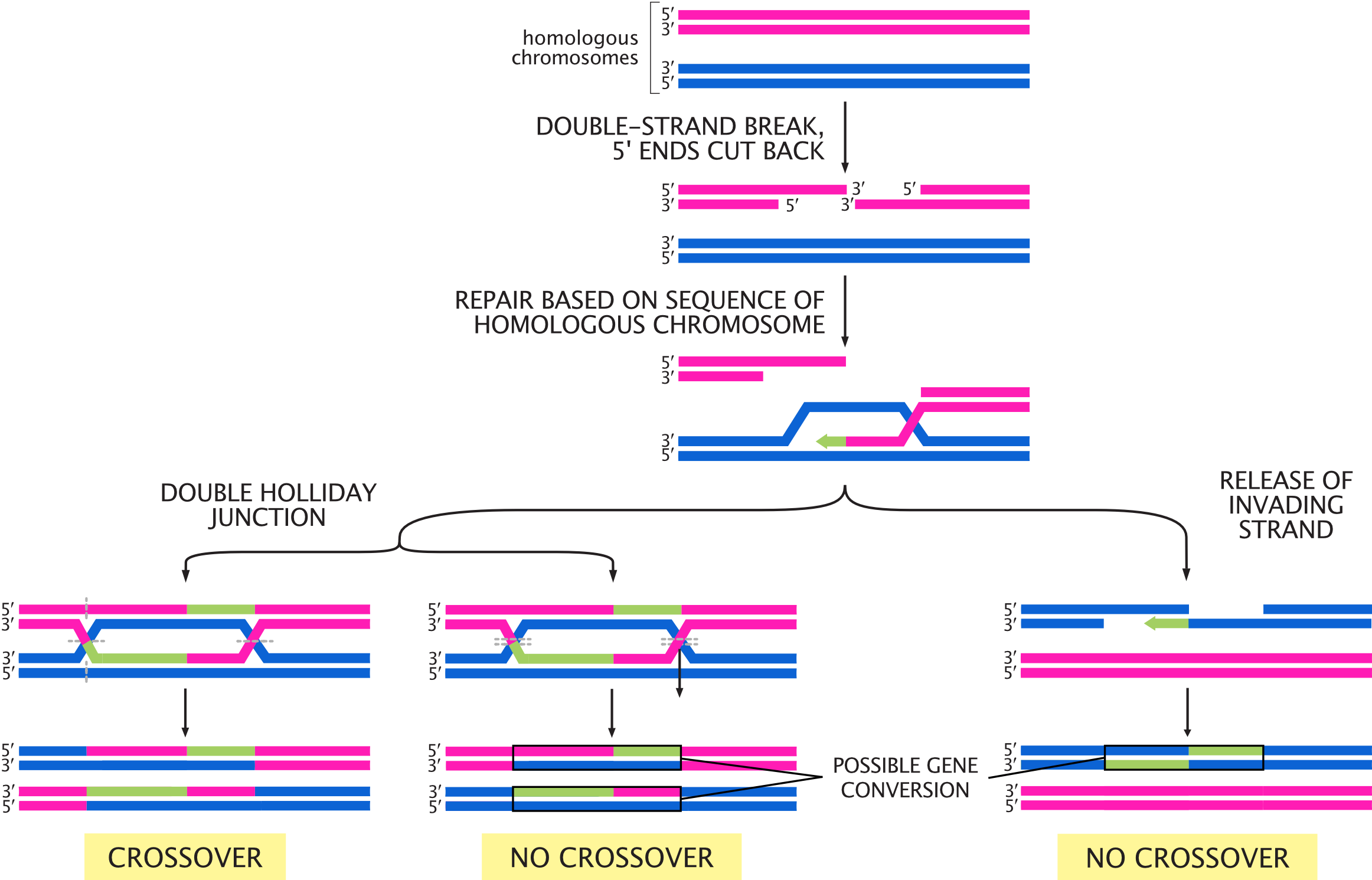


Figure : **Fig. 2.** Possible outcomes for double-strand breaks generated during meiosis I. Adapted from *Molecular Biology of the Cell, 6th Edition* (Alberts et al.)

## 3.3 Setup

In this module, we’ll use sequencing data from families to look at the relationship between DNMs, crossovers, and parental age.

### 3.3.1 R packages

We’re using R’s tidyverse library to analyze our data. You can load this R package by running:

library(tidyverse)

### 3.3.2 Data

Our data comes from the supplementary tables of [this paper by Halldorsson et al.](https://science.sciencemag.org/content/363/6425/eaau1043), which performed whole-genome sequencing on “trios” (two parents and one child) in Iceland. We’ve pre-processed the data to make it easier to work with.

Load the pre-processed data by running the code chunk below.

# read data  
dnm\_by\_age <- read.table("dnm\_by\_age\_tidy\_Halldorsson.tsv",  
 sep = "\t", header = TRUE)  
# preview data  
head(dnm\_by\_age)

## Proband\_id n\_paternal\_dnm n\_maternal\_dnm n\_na\_dnm Father\_age Mother\_age  
## 1 675 51 19 0 31 36  
## 2 1097 26 12 1 19 19  
## 3 1230 42 12 3 30 28  
## 4 1481 53 14 1 32 20  
## 5 1806 61 11 6 38 34  
## 6 2280 63 9 3 38 20

The columns in this table are:

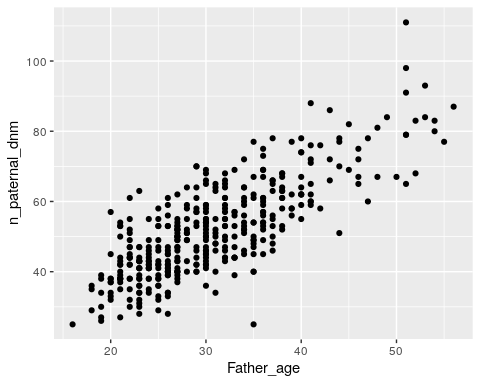
1. Proband\_id: ID of the child (i.e., “proband”)
2. n\_paternal\_dnm: Number of DNMs (carried by the child) that came from the father
3. n\_maternal\_dnm: Number of DNMs that came from the mother
4. n\_na\_dnm: Number of DNMs whose parental origin can’t be determined
5. Father\_age: Father’s age at proband’s birth
6. Mother\_age: Mother’s age at proband’s birth

## 3.4 Visualizing the data

We can use our tidied data to ask questions about the *de novo* mutation rate in these Icelandic individuals. How does parental age affect the number of DNMs for males and females?

Use the dnm\_by\_age data to plot this relationship for *males*.

ggplot(data = dnm\_by\_age,  
 # specify where ggplot should be getting the x location for each data point  
 aes(x = Father\_age,  
 # specify where ggplot should be getting the y location for each data point  
 y = n\_paternal\_dnm)) +  
 # specify that the data should be plotted as points  
 geom\_point()

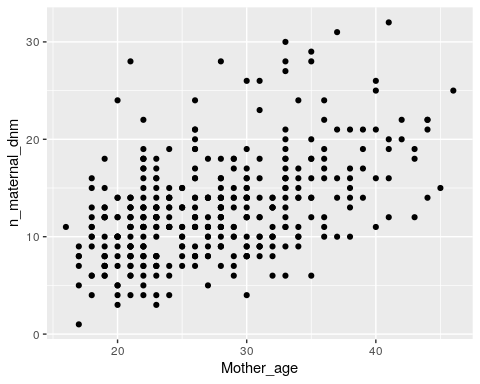


Based on your plot, would you say that there’s an association between paternal age and number of DNMs?

It looks like there’s a pretty strong association between paternal age and number of DNMs, where older males have more DNMs.

Modify your code to plot the relationship between age and number of DNMs for *females*. Does there seem to be an association between maternal age and number of DNMs?

ggplot(data = dnm\_by\_age,  
 aes(x = Mother\_age,  
 y = n\_maternal\_dnm)) +  
 geom\_point()



There’s also a strong positive association between maternal age and number of DNMs, although the slope (i.e., the increase in number of DNMs per year) is shallower.

## 3.5 Linear models

We can visually observe that age seems associated with number of DNMs in both males and females, but we need a way to ask if that this is a statistically meaningful association.

We can do this with a **linear model**. This model fits a line to the plots that we just made, and asks if the slope is significantly different from 0 (i.e., if there’s a significant increase in DNM count as age increases).

If this is a statistical test, what’s the null hypothesis?

The null hypothesis for this linear model is that the slope is 0 – i.e., that there’s no association between parental age and the number of DNMs from that parent.

If the slope is significantly different from 0, we can reject the null hypothesis.

We’ll fit a linear model using R’s lm function. Run the following code block to open a manual describing the function.

?lm

lm requires two arguments:

* The formula or equation it’s evaluating
* A table of data

The formula must be in the format response variable ~ predictor variable(s), where each variable is the name of a column in our data table.

Is our predictor variable the parental age or the number of DNMs?

The predictor variable is parental age. We expect the number of DNMs to change as a *consequence* of parental age.

## 3.6 Fitting a linear model for DNMs

Run the following code to fit a model for the effect of age on paternal DNMs.

# fit linear model for paternal DNMs  
fit\_pat <- lm(formula = n\_paternal\_dnm ~ Father\_age,  
 data = dnm\_by\_age)  
  
# print results of model  
summary(fit\_pat)

##   
## Call:  
## lm(formula = n\_paternal\_dnm ~ Father\_age, data = dnm\_by\_age)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -32.785 -5.683 -0.581 5.071 31.639   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 10.58819 1.70402 6.214 1.34e-09 \*\*\*  
## Father\_age 1.34849 0.05359 25.161 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 8.426 on 388 degrees of freedom  
## Multiple R-squared: 0.62, Adjusted R-squared: 0.619   
## F-statistic: 633.1 on 1 and 388 DF, p-value: < 2.2e-16

How do you interpret results from a linear model?

For our purposes, the only part of the results you need to look at is the line under (Intercept) in the Coefficients section:

Estimate Std. Error t value Pr(>|t|)  
Father\_age 1.34849 0.05359 25.161 < 2e-16 \*\*\*

* The fourth columm, Pr(>|t|), is the **p-value**.

Because this p-value is < 2e-16, we can reject the null hypothesis and say that there is association between paternal age and the number of paternal DNMs.

* The first column, Estimate, is the **slope**, or **coefficient**.

Linear regression fits a line to our plot of paternal age vs. number of DNMs. The coefficient estimate is the slope of that line.

The slope for paternal age given by this linear model is 1.34849. We can interpret this number this way: **For every additional year of paternal age, we expect 1.35 additional paternal DNMs in the child.**

Modify your code to assess the relationship between *maternal* age and number of *maternal* DNMs. Is this relationship significant? How many maternal DNMs do we expect for every additional year of maternal age?

# fit linear model for maternal DNMs  
fit\_mat <- lm(formula = n\_maternal\_dnm ~ Mother\_age,  
 data = dnm\_by\_age)  
  
# print results of model  
summary(fit\_mat)

##   
## Call:  
## lm(formula = n\_maternal\_dnm ~ Mother\_age, data = dnm\_by\_age)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -9.8683 -3.1044 -0.2329 2.2394 17.5379   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 2.51442 0.98193 2.561 0.0108 \*   
## Mother\_age 0.37846 0.03509 10.785 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 4.503 on 388 degrees of freedom  
## Multiple R-squared: 0.2307, Adjusted R-squared: 0.2287   
## F-statistic: 116.3 on 1 and 388 DF, p-value: < 2.2e-16

The p-value is <2e-16 and the Mother\_age slope is 0.37846.

This relationship is significant, and we expect 0.38 more maternal DNMs for every additional year of maternal age.

## 3.7 Confidence intervals

Our models predict that there are 1.35 more DNMs for additional every year of paternal age, and 0.38 more DNMs for every additional year of maternal age. Does this mean that sperm and oocytes accumulate DNMs at different rates?

The maternal and paternal slopes look different, but we need statistical evidence that they actually are. (For example, what if there’s a lot of variability in the maternal DNM data, and the true maternal coefficient could be anywhere between -1 and 10?)

To do this, we compare the **confidence intervals** of our slope estimates.

What is a confidence interval?

We use confidence intervals when estimating a value – in this case, the Mother\_age and Father\_age slope parameters.

A **confidence interval (CI)** is a range of values for which, for some probability, the interval will contain the true value of the slope.

So, a 95% CI contains the true value of the slope 95% of the time.

In R, we get the confidence interval of a parameter from a linear model with the confint function.

?confint

confint requires three arguments:

* A fitted linear model (our fit\_pat variable)
* The parameter we want a CI for (Father\_age)
* The CI’s probability (typically 95%)

## 3.8 Calculate 95% CIs

Run the following code to calculate the 95% confidence interval for the Father\_age slope parameter.

confint(fit\_pat, 'Father\_age', level = 0.95)

## 2.5 % 97.5 %  
## Father\_age 1.243118 1.45386

So, 95% of the time, the number of additional DNMs per year of paternal age is between 1.24 and 1.45.

Modify your code to get the 95% CI for the *Mother\_age* slope. What’s the interpretation of this confidence interval?

confint(fit\_mat, 'Mother\_age', level = 0.95)

## 2.5 % 97.5 %  
## Mother\_age 0.3094713 0.4474528

95% of the time, the number of additional DNMs per year of maternal age is between 0.31 and 0.45.

Now that we have the confidence intervals for both slope parameters, we can finally compare them.

Our two CI ranges are non-overlapping. The paternal range is [1.24, 1.45] and the maternal range is [0.31, 0.45].

If the 95% CIs for two parameters *don’t* overlap, this strongly supports that the parameters are significantly different from one another. **So, it seems likely that paternal and maternal gametes experience different rates of *de novo* mutation.**

If the CIs for two parameters overlap, are they not significantly different?

Not necessarily. More analysis, like a hypothesis test, is needed to make a final decision.

## 3.9 Conclusion

In this lab, we explored the relationship between parental age and the number of *de novo* mutations in their gametes.

* We **plotted** the relationship between maternal/paternal age and DNM count. This visualization suggested that DNM count increases with age for both groups.
* We confirmed this hypothesis by using a **linear model**, which tests if additional years of age have a non-zero effect on the number of DNMs.
* The number of paternal DNMs seemed to increase more quickly with age than maternal DNMs. We confirmed this by comparing the **95% confidence intervals** of the slopes of the two models.

One final question – let’s assume that there really is a difference between the effect of age on DNMs in male and female gametes. What biological reasons might be causing this difference?

## 3.10 Homework

So far, we’ve only looked at the *de novo* mutation data from [the Halldorsson et al. paper](https://science.sciencemag.org/content/363/6425/eaau1043). Now we’ll use their data on the number of maternal and paternal origin crossovers (i.e., how many crossovers occurred across all chromosomes in the maternal and paternal gametes).

#### 3.10.0.1 Goals & Learning Objectives

The goal of this homework is to practice with ggplot.

**Learning Objectives**

* Required homework: Practice visualizing data with ggplot2
* Optional homework: Practice interpreting linear models

## 3.11 Required homework

The data from the paper has been pre-filtered for you. Run this code block to read it in:

# read data  
crossovers <- read.table("crossovers.tsv", header = TRUE)  
  
# preview data  
head(crossovers)

## Proband\_id n\_pat\_xover n\_mat\_xover Father\_age Mother\_age  
## 1 3 22 51 29 28  
## 2 10 26 50 26 26  
## 3 11 25 38 25 22  
## 4 15 24 50 31 26  
## 5 20 27 35 26 24  
## 6 22 28 40 39 31

The columns in this table are:

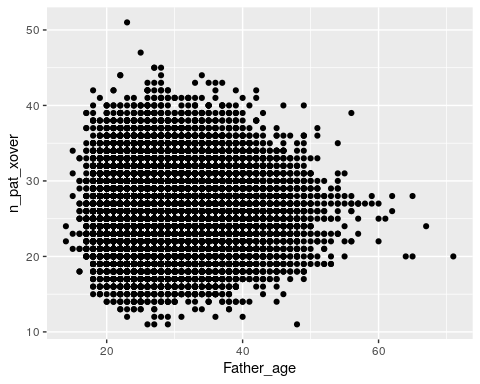
1. Proband\_id: ID of the child
2. n\_pat\_xover: Number of crossovers (carried by the child) that occurred in the paternal gametes
3. n\_mat\_xover: Number of crossovers that occurred in the maternal gametes
4. Father\_age: Father’s age at proband’s birth
5. Mother\_age: Mother’s age at proband’s birth

**Assignment:** Using the ggplot code from this module, plot the relationship between parental age and number of crossovers. As with the DNM data, make one plot for the maternal crossovers and one plot for the paternal. Do you think parental age impacts crossover number?

Solution

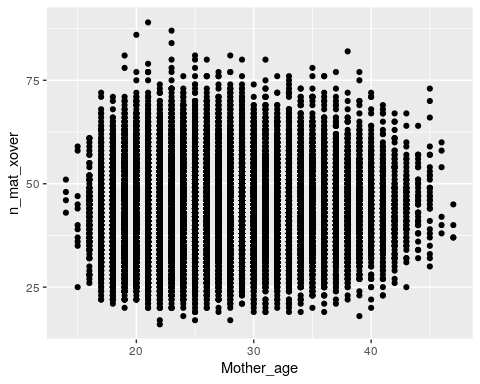
Plot paternal crossovers:

ggplot(data = crossovers,  
 # x axis is paternal age  
 aes(x = Father\_age,  
 # y axis is number of crossovers  
 y = n\_pat\_xover)) +  
 geom\_point()



Plot maternal crossovers:

ggplot(data = crossovers,  
 # x axis is maternal age  
 aes(x = Mother\_age,  
 # y axis is number of crossovers  
 y = n\_mat\_xover)) +  
 geom\_point()



Just by eye, it doesn’t really seem that age affects number of crossovers for either mothers or fathers.

## 3.12 Optional homework

**Assignment:** Fit *two* linear models (one paternal, one maternal) to ask if there is an association between the number of parental crossovers and parental age. If there is an association, how is the number of crossovers predicted to change with every year of maternal/paternal age?

Solution

# fit the model with paternal age  
fit\_pat <- lm(data = crossovers,  
 formula = n\_pat\_xover ~ Father\_age)  
summary(fit\_pat)

##   
## Call:  
## lm(formula = n\_pat\_xover ~ Father\_age, data = crossovers)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.2173 -3.1880 -0.1997 2.8061 24.7652   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 26.369432 0.102736 256.67 <2e-16 \*\*\*  
## Father\_age -0.005852 0.003462 -1.69 0.091 .   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 4.388 on 41090 degrees of freedom  
## Multiple R-squared: 6.953e-05, Adjusted R-squared: 4.519e-05   
## F-statistic: 2.857 on 1 and 41090 DF, p-value: 0.09098

There isn’t a significant association between paternal age and the number of paternal crossovers (p = 0.091).

# fit the model with maternal age  
fit\_mat <- lm(data = crossovers,  
 formula = n\_mat\_xover ~ Mother\_age)  
summary(fit\_mat)

##   
## Call:  
## lm(formula = n\_mat\_xover ~ Mother\_age, data = crossovers)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -27.161 -6.095 -0.425 5.641 45.905   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 41.709271 0.206238 202.24 <2e-16 \*\*\*  
## Mother\_age 0.065989 0.007576 8.71 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 8.685 on 41090 degrees of freedom  
## Multiple R-squared: 0.001843, Adjusted R-squared: 0.001819   
## F-statistic: 75.87 on 1 and 41090 DF, p-value: < 2.2e-16

Surprisingly, there *is* a significant association between maternal age and the number of maternal crossovers (p < 2e-16). For every year of maternal age, we expect the child to carry 0.07 additional maternal origin crossovers.

Although the maternal crossovers plot doesn’t look that impressive, our estimated slope is 0.07, which is probably too small to distinguish visually.

# 4 Simulating evolution

In this lab, we’ll build a simulation to explore genetic drift using the Wright-Fisher model.

#### 4.0.0.1 Learning objectives

After completing this chapter, you’ll be able to:

1. Describe the phenomenon of genetic drift.
2. Explain why random draws from a binomial distribution are a good way to mimic the effect of drift.
3. Interpret allele frequency patterns that occur as a result of drift.
4. Write a for loop in R.
5. Write a function to run code multiple times with different parameters.

## 4.1 Background

### 4.1.1 Genetic drift

In all populations, **genetic drift** acts to change allele frequencies over time. Drift refers to random changes in an allele’s frequency between generations. These random changes occur because individuals carrying different alleles will have different numbers of offspring solely due to chance.

Drift differs from **selection**, which is a deterministic (non-random) change in an allele’s frequency. If an allele is under selection, it’s more likely to increase or decrease in frequency depending on whether it is beneficial or deleterious. Genetic drift, on the other hand, cannot consistently cause an allele’s frequency to increase or decrease.

Although both genetic drift and selection occur in real populations, drift is thought to be the primary driver of allele frequency changes, far outweighing the impacts of selection. This is called the [neutral theory of molecular evolution](https://www.nature.com/scitable/topicpage/neutral-theory-the-null-hypothesis-of-molecular-839/).

### 4.1.2 The Wright-Fisher model

The **Wright-Fisher model** is one of the most commonly used models of genetic drift.

#### 4.1.2.1 Population size

A Wright-Fisher population is a set of individuals who mate randomly. The model assumes that this number remains constant between generations. For simplicity, in this module we’ll also assume that every individual is haploid (so that they can only carry one copy of an allele).

Populations in the real world don’t behave as randomly as an idealized Wright-Fisher population, so their **effective population size** Ne is usually much smaller than their actual population size. The effective population size of the human population is only [12,800–14,400](https://www.pnas.org/content/109/44/17758) individuals, even though its actual size is around 7.8 billion.

#### 4.1.2.2 Allele frequency, fixation, and extinction

The Wright-Fisher model describes the behavior of a single **allele**, which is a variable site in a population (a SNP, insertion/deletion, version of a gene, etc.). If A and a are two alleles, representing two different sequences that exist at the same genomic location, Wright-Fisher allows us to track what happens to one of those alleles under genetic drift.

The allele of interest begins the simulation at some initial **allele frequency (AF)**. This allele frequency is the proportion of individuals in the population who carry that allele, and is always between 0 and 1.

An allele becomes **fixed** in a population when it reaches an allele frequency of 1. At this point, it is no longer considered an allele because everyone in the population carries it. Similarly, an allele goes **extinct** when it reaches an allele frequency of 0 and nobody in the population carries it.

## 4.2 Modeling allele frequencies

In the Wright-Fisher model, we track a population over the course of many generations. When creating each new generation of individuals, we determine how many of them carry A, our allele of interest.

For every individual, we perform a coin flip to determine whether or not they have the A allele. But unlike a coin, the probabilities of having A vs. a aren’t equal.

Instead, the probability of receiving the A allele is equal to A’s **allele frequency** in the current generation. The more common A is in this generation, the more likely it is that someone in the next generation will also carry it.

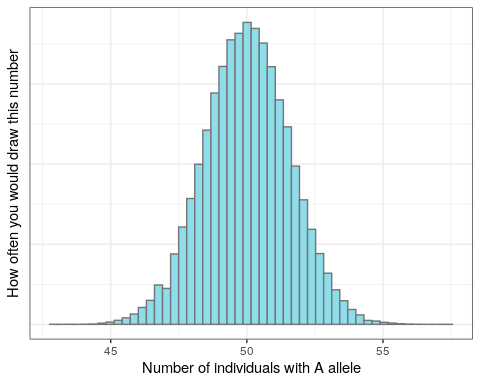


Figure : **Fig. 1. Modeling allelic inheritance with coin flips.** At the end of a generation, every individual flips a weighted coin to determine whether they will carry the blue allele in the next generation. The probability of carrying the allele is equal to the allele’s frequency in the current generation.

We do this coin flip N times, once for each individual in the population, to get the number of people in the next generation who carry the A allele (**Fig. 1**). Conveniently, instead of having to actually simulate all the coin flips, we can get this number by performing a single draw from a **binomial distribution** with size N and success probability p = AF. This distribution tells you how many successes you expect to see from a set of N independent coin flips.

### 4.2.1 The binomial distribution

If we try to draw 100,000 times from a binomial distribution with population size 100 and success probability (AF) 0.5, it’ll look something like this:



The majority of the distribution lies between 48 and 52. Just as we expect based on the allele frequency, the next generation will most likely have around 48-52 individuals with the A allele. But because this is a random draw, there’s a small chance that we might end up with many more or many fewer than that number.

## 4.3 Setup

### 4.3.1 R packages

Load the tidyverse library (which includes ggplot2) to use for the rest of the lab:

library(tidyverse)

## 4.4 A Wright-Fisher simulation

### 4.4.1 Drawing one generation from a binomial distribution

The basis of our simulation is R’s rbinom function, which allows us to sample from a binomial distribution. rbinom takes three arguments:

* n: how many times we’re drawing from the distribution
* size: the size of the population we’re sampling from (i.e. N)
* p: the success probability (i.e. allele frequency)

Every generation, we will draw **once** to produce the number of individuals carrying the A allele in the next generation.

Let’s once again look at a population of size 100, and an A allele currently at AF = 0.5. We use rbinom to get the number of individuals in the next generation who will have A:

rbinom(n = 1, size = 100, prob = 0.5)

## [1] 45

**Coding exercise**: Change the code above so that it returns the allele frequency rather than the number of individuals.

Solution

# divide by the population size to get AF  
rbinom(n = 1, size = 100, prob = 0.5) / 100

## [1] 0.51

**Question**: You’ll notice that every time we run the rbinom line, the number we get is different. Why is that?

Solution

rbinom generates a random number between 1 and 100. Because it’s random, the number it draws will be different every time we run it.

Currently, we’re drawing from a population of 100 individuals. Now let’s see what happens when we increase the population size. (Feel free to run this code block multiple times!)

rbinom(n = 1, size = 10000, prob = 0.5) / 10000

## [1] 0.4934

If you run the code block above multiple times, you’ll observe that the AF is much closer to 0.5 than it was with a population of size 100. This lends to our intuition that an allele’s frequency fluctuates much more when a population is small, and is more stable when the population size is large.

**Question**: As you increase population size, how does that affect an allele’s time to fixation?

Answer

As population size gets larger, the allele will take longer to fix.

### 4.4.2 Simulating multiple generations with a for loop

We can now draw *once* from a binomial distribution to get the number of individuals in one generation who carry the A allele. How do we adapt this to simulate multiple generations?

**Question**: Can we increase n (for example, with rbinom(n = 10, size = 100, prob = 0.5)) to draw multiple times?

Solution

No. Increasing n only gives you multiple replicate draws from the same distribution. It won’t update the AF between generations based on the new number of A alleles, because it uses prob = 0.5 every time.

Instead of drawing multiple times from the same distribution, we write a **for loop** to repeatedly generate and update the number of individuals with the A allele.

A **for loop** allows you to run some code X number of times. For example:

for (i in 1:3) {  
 print(i)  
}

## [1] 1  
## [1] 2  
## [1] 3

This for loop goes through all the values between 1 and 3, and prints each of them out.

We can write a similar for loop that includes rbinom:

for (i in 1:3) {  
 print(rbinom(n = 1, size = 100, prob = 0.5) / 100)  
}

## [1] 0.49  
## [1] 0.45  
## [1] 0.43

This is very close to what we want. However, we’re still running rbinom with the same AF in every iteration.

How do we change this to update the AF each generation? We can add a freq variable that keeps track of the current allele frequency:

# start an initial AF of 0.5  
freq <- 0.5  
  
for (i in 1:3) {  
 # run rbinom to generate the AF for the next generation  
 new\_freq <- rbinom(n = 1, size = 100, prob = freq) / 100  
 # print so that we can see what `freq` is each time  
 print(new\_freq)  
 # update `freq` in each iteration of the loop  
 freq <- new\_freq  
}

## [1] 0.53  
## [1] 0.51  
## [1] 0.52

Now the for loop is updating the allele’s frequency every generation and running rbinom with that new frequency.

One more modification: Providing the variable freq as an input to rbinom allows us to update freq with each for loop iteration. But it also allows us to pass rbinom a variable, rather than hard-coding in a population size.

**Coding exercise**: Add to the code above so that we also provide Ne (effective population size) as a variable (*without* updating it in the for loop).

Solution

# set effective population size outside of for loop  
Ne <- 100  
# start an initial AF of 0.5  
freq <- 0.5  
  
for (i in 1:3) {  
 # run rbinom to generate the AF for the next generation  
 new\_freq <- rbinom(n = 1, size = Ne, prob = freq) / Ne  
 # print so that we can see what `freq` is each time  
 print(new\_freq)  
 # update `freq` in each iteration of the loop  
 freq <- new\_freq  
}

## [1] 0.53  
## [1] 0.47  
## [1] 0.43

## 4.5 Plotting

### 4.5.1 Visualizing changes in AF over generations

Try increasing the number of generations we run the simulation for. What patterns of change do you observe in the allele frequencies?

Ne <- 100  
freq <- 0.5  
  
for (i in 1:20) {  
 new\_freq <- rbinom(n = 1, size = Ne, prob = freq) / Ne  
 print(new\_freq)  
 freq <- new\_freq  
}

## [1] 0.53  
## [1] 0.44  
## [1] 0.41  
## [1] 0.48  
## [1] 0.55  
## [1] 0.49  
## [1] 0.5  
## [1] 0.5  
## [1] 0.44  
## [1] 0.48  
## [1] 0.51  
## [1] 0.46  
## [1] 0.45  
## [1] 0.48  
## [1] 0.46  
## [1] 0.51  
## [1] 0.55  
## [1] 0.58  
## [1] 0.54  
## [1] 0.56

It would be useful to plot how the AF changes over time, so that we can look at it visually. We can do this by storing the AF at each generation in a **vector**, which you can think of as R’s version of a list.

Vectors are formed with the c() function, which stands for “combine”:

my\_vec <- c(0.5, 0.6)  
my\_vec

## [1] 0.5 0.6

You can append elements to a vector called my\_vec by running: my\_vec <- c(my\_vec, new\_element).

**Coding exercise**: Modify the code block with our for loop to create a vector for storing allele frequencies, and then append the updated AF to it every generation.

Hint

Create the vector **before** the for loop. Then append to the vector **within** the for loop.

Solution

Ne <- 100  
freq <- 0.5  
# create vector to store AFs in  
freq\_vector <- freq  
  
for (i in 1:20) {  
 new\_freq <- rbinom(n = 1, size = Ne, prob = freq) / Ne  
 # add new freq to the AF vector  
 freq\_vector <- c(freq\_vector, new\_freq)  
 freq <- new\_freq  
}  
  
freq\_vector

## [1] 0.50 0.52 0.64 0.63 0.65 0.63 0.61 0.68 0.68 0.65 0.63 0.63 0.64 0.62 0.57  
## [16] 0.60 0.61 0.63 0.64 0.65 0.68

We can plot this allele frequency vector with ggplot. First, because ggplot requires its input data to be formatted as a table, we have to convert the vector into some form of table. (We chose tibble form here because it’s easy to convert to.)

sim\_results <- tibble(af = freq\_vector)  
sim\_results

## # A tibble: 21 x 1  
## af  
## <dbl>  
## 1 0.5   
## 2 0.52  
## 3 0.64  
## 4 0.63  
## 5 0.65  
## 6 0.63  
## 7 0.61  
## 8 0.68  
## 9 0.68  
## 10 0.65  
## # … with 11 more rows

This table contains the information that we want on the plot’s y axis. We can now add in a column containing the plot’s x axis data. This should be the **generation** that each AF value corresponds to.

sim\_results <- tibble(af = freq\_vector,  
 gen = 1:21)  
sim\_results

## # A tibble: 21 x 2  
## af gen  
## <dbl> <int>  
## 1 0.5 1  
## 2 0.52 2  
## 3 0.64 3  
## 4 0.63 4  
## 5 0.65 5  
## 6 0.63 6  
## 7 0.61 7  
## 8 0.68 8  
## 9 0.68 9  
## 10 0.65 10  
## # … with 11 more rows

Why does the gens column go from 1 to 21 (instead of 20)?

We add our starting allele frequency to freq\_vector, and then simulate 20 generations of drift. This means that we end up with 21 AFs in our vector.

Now we can finally plot the trajectory of AFs over time with ggplot.

ggplot(data = sim\_results,  
 aes(x = gen, y = af)) +  
 geom\_line()



## 4.6 Creating a simulation function

### 4.6.1 Simulating with different parameters with a function

It would be nice to be able to run the Wright-Fisher simulation with different parameters – like different starting allele frequencies, population sizes, etc. – without having to modify the for loop code every time. We can use a **function** to generalize the code above so we can easily re-run it.

Whenever you have a bunch of lines at the beginning of your code where you set variables, you should always be thinking that you can make it into a function.

The structure of an R function

You’ve already encountered many functions in R, even if you didn’t realize it at the time - rbinom, ggplot, and print are all examples of functions. An R function has [four parts](https://www.tutorialspoint.com/r/r_functions.htm):

<Name> <- function(<Argument(s)>) {  
 <Body>  
 <return()>  
}

* **Name** − The function is stored in your R environment as an object with this name, and you use the name to call it
* **Argument(s)** − Optional; input values that the function performs operations on
* **Body** − The code that describes what the function does
* **Return** − Optional; a return statement allows the function to return a value to the user. Without a return statement, you won’t be able to access the function’s output

Here’s an example function that takes in three parameters for running rbinom, and returns the output of rbinom.

binom\_sim <- function(myN, mySize, myProb) {  
 output <- rbinom(n = myN, size = mySize, prob = myProb)  
 return(output)  
}

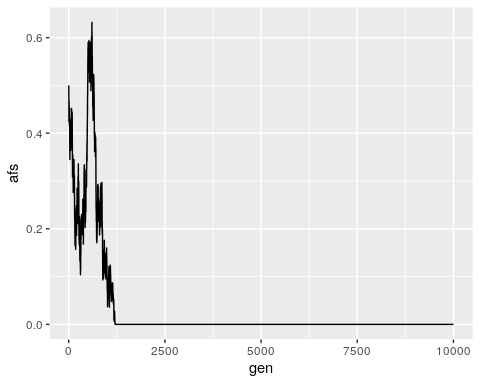
We want our function to take in parameters for the starting allele frequency, population size, and number of generations to simulate. It should return the sim\_results dataframe so that we can plot the allele frequency trajectory.

To write a function, we can place the code that we just wrote into the function body:

run\_sim <- function(Ne, freq, generations) {  
   
 # note how we don't define our initial parameters for Ne, freq, etc.  
 # because we're passing in those parameters as arguments  
   
 freq\_vector <- freq  
 for (i in 1:generations) {  
 new\_freq <- rbinom(n = 1, size = Ne, prob = freq) / Ne  
 freq\_vector <- c(freq\_vector, new\_freq)  
 freq <- new\_freq  
 }  
   
 # convert vector of AFs into a tibble for plotting  
 sim\_results <- tibble(afs = freq\_vector,  
 gen = 1:(generations+1))  
   
 # return the tibble of AFs, so that we can access the results  
 return(sim\_results)  
}

Now we can run the function with parameters of our choosing and plot the output:

# run function  
results <- run\_sim(Ne = 1000, freq = 0.5, generations = 10000)  
  
# plot output  
ggplot(data = results,  
 aes(x = gen, y = afs)) +  
 geom\_line()



**Exercise**: Run your run\_sum function a few times with different input population sizes and allele frequencies. How does changing thesse inputs affect the allele frequency trajectories that you see?

How do I know when to use a function?

Functions are useful whenever you have code that you want to run multiple times with slightly different parameters. If you find yourself copying over code several times and changing just a few things, you should consider writing a function instead.

## 4.7 Conclusion

In this lab, we’ve successfully built a Wright-Fisher simulation for one allele, allowing us to track how we expect its frequency to change over time under the principles of genetic drift.

This simple simulation forms the core of most models used in evolutionary genetics research. For example, the evolutionary simulation software [SLiM](https://messerlab.org/slim/) extends our model to perform Wright-Fisher simulations with multiple alleles simultaneously, allowing us to reconstruct complex phenomena like balancing selection (**Fig. 2**).

Fig. 2. Simulating balancing selection with SLiM. Each vertical line represents a SNP, with height corresponding to the SNP’s current frequency. As the simulation progresses, two distinct haplotypes of mutations begin to rise and fall in frequency together, but neither fully fixes because balancing selection favors intermediate frequencies.

Figure : **Fig. 2. Simulating balancing selection with SLiM.** Each vertical line represents a SNP, with height corresponding to the SNP’s current frequency. As the simulation progresses, two distinct haplotypes of mutations begin to rise and fall in frequency together, but neither fully fixes because balancing selection favors intermediate frequencies.

## 4.8 Homework

#### 4.8.0.1 Goals & Learning Objectives

The goal of this homework is to explore different extensions of the Wright-Fisher model.

**Learning Objectives**

* Required homework: Practice visualizing allele frequencies with ggplot
* Extra credit: Practice writing functions and interpreting allele frequency trajectories

### 4.8.1 Required

Currently, our allele frequency trajectory plot only shows the AF of one allele at a locus. (i.e., if individuals in a population have either an A or a C at some locus, we’re only plotting the trajectory of the A allele.)

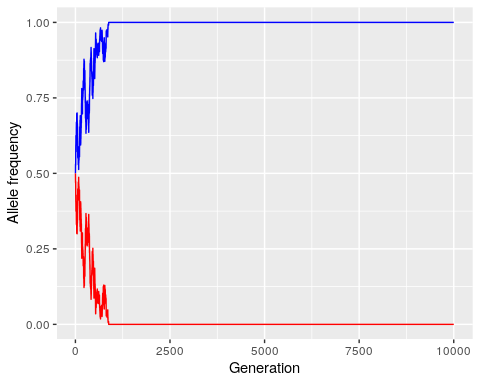
**Assignment:** Add a line to allele frequency trajectory plot that shows the frequency of the other allele at the locus. Give the two alleles different colors.

Hint

You don’t need to modify the simulation function for this.

Solution

# run the simulation again to get output data  
sim <- run\_sim(Ne = 1000, freq = 0.5, generations = 10000) %>%  
 # add in a column with the AF of the minor allele  
 # this is 1 - AF of the major allele  
 dplyr::mutate(afs\_minor = 1 - afs)  
  
ggplot() +  
 geom\_line(data = sim, aes(x = gen, y = afs), color = "blue") +  
 # add in another line with the frequency of the minor allele  
 geom\_line(data = sim, aes(x = gen, y = afs\_minor), color = "red") +  
 ylim(0, 1) +  
 ylab("Allele frequency") +  
 xlab("Generation")



### 4.8.2 Extra credit: Selection

One way to extend our simple Wright-Fisher model is to add in selection as a parameter. Selection affects our model by altering the probability of sampling our allele of interest each generation (e.g., positive selection increases the probability, and negative selection decreases it).

Previously, we assumed that this probability was equivalent to the allele’s frequency, or , where is the population size and is the number of individuals who carry the allele.

For the purposes of this homework, we assume that in a model with selection, this probability is instead:

where is the **selection coefficient**, and ranges from -1 to 1.

**Question**: What does this probability become in the absence of selection (i.e., when )?

The probability becomes , which is the same as the allele frequency.

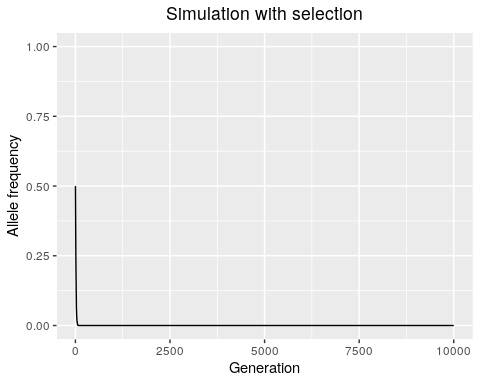
**Assignment:** Modify your run\_sim function so that it takes in a selection coefficient s as a parameter. Run the simulation a few times with and without (s = 0) selection, but keeping other parameters the same (Ne = 10000 – to mimic the Ne of humans, freq = 0.5, generations = 10000). What do you notice about the allele frequency trajectories?

Solution

run\_sim\_selection <- function(Ne, freq, generations, s) {  
   
 freq\_vector <- freq  
 for (i in 1:generations) {  
 # calculate p, the probability of sampling the allele, based on s  
 i <- freq \* Ne # number of individuals who currently carry the allele  
 p <- i\*(1+s) / (Ne - i + i\*(1+s))  
   
 # prob is now `p`, rather than `freq`  
 new\_freq <- rbinom(n = 1, size = Ne, prob = p) / Ne  
 freq\_vector <- c(freq\_vector, new\_freq)  
 freq <- new\_freq  
 }  
   
 # convert vector of AFs into a tibble for plotting  
 sim\_results <- tibble(afs = freq\_vector,  
 gen = 1:(generations+1))  
   
 # return the tibble of AFs, so that we can access the results  
 return(sim\_results)  
}

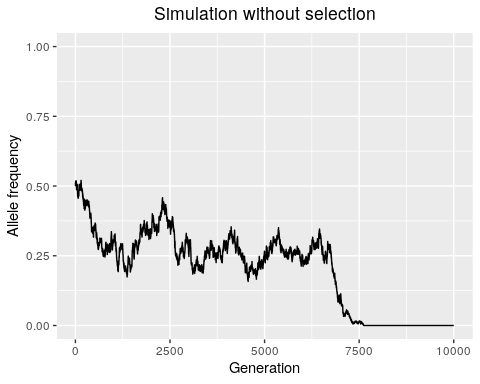
Run and plot the simulation with selection:

results <- run\_sim\_selection(Ne = 10000,  
 freq = 0.5,  
 generations = 10000,  
 s = -0.1)  
ggplot() +  
 geom\_line(data = results, aes(x = gen, y = afs)) +  
 ylim(0, 1) +  
 ylab("Allele frequency") +  
 xlab("Generation") +  
 ggtitle("Simulation with selection") +  
 theme(plot.title = element\_text(hjust = 0.5)) # to center the title



Run and plot the simulation without selection:

results <- run\_sim\_selection(Ne = 10000,  
 freq = 0.5,  
 generations = 10000,  
 s = 0)  
ggplot() +  
 geom\_line(data = results, aes(x = gen, y = afs)) +  
 ylim(0, 1) +  
 ylab("Allele frequency") +  
 xlab("Generation") +  
 ggtitle("Simulation without selection") +  
 theme(plot.title = element\_text(hjust = 0.5)) # to center the title



We observe that selection (at least, strong selection where s = 0.1) tends to decrease the time it takes for an allele to either fix or go extinct. This is because selection directionally biases the probability of sampling that allele.

Decreasing the absolute value of the selection coefficient will make the simulation behave more like drift - most selection coefficients are thought to be [very small](https://academic.oup.com/view-large/figure/326984394/345fig4.jpeg), and the largest known selection coefficients in humans are around [0.05](https://elifesciences.org/articles/63177).

# About the Authors

| Credits | Names |
| --- | --- |
| **Pedagogy** |  |
| Instructor | [Rajiv McCoy](https://mccoy-lab.org/) |
| Content Author | [Stephanie Yan](https://stephaniemyan.github.io/) |
| Content Author | [Kate Weaver](https://kweav.github.io/) |
| **Website** |  |
| Template | [Jeff Leek](https://jtleek.com/) & [The Johns Hopkins Data Science Lab](https://jhudatascience.org/index.html) |
| Design Inspiration | [Ali Madooei](https://engineering.jhu.edu/faculty/ali-madooei/) & [JHU Data Structures](https://cs226sp22.github.io/) |
| **Funding** |  |
| JHU Center for Educational Resources | Techology Fellowship Grant |

## ─ Session info ───────────────────────────────────────────────────────────────  
## setting value   
## version R version 4.0.2 (2020-06-22)  
## os Ubuntu 20.04.3 LTS   
## system x86\_64, linux-gnu   
## ui X11   
## language (EN)   
## collate en\_US.UTF-8   
## ctype en\_US.UTF-8   
## tz Etc/UTC   
## date 2023-01-17   
##   
## ─ Packages ───────────────────────────────────────────────────────────────────  
## package \* version date lib source   
## assertthat 0.2.1 2019-03-21 [1] RSPM (R 4.0.3)   
## backports 1.1.10 2020-09-15 [1] RSPM (R 4.0.2)   
## blob 1.2.1 2020-01-20 [1] RSPM (R 4.0.3)   
## bookdown 0.24 2022-02-15 [1] Github (rstudio/bookdown@88bc4ea)   
## broom 0.7.1 2020-10-02 [1] RSPM (R 4.0.2)   
## callr 3.4.4 2020-09-07 [1] RSPM (R 4.0.2)   
## cellranger 1.1.0 2016-07-27 [1] RSPM (R 4.0.3)   
## cli 2.0.2 2020-02-28 [1] RSPM (R 4.0.0)   
## colorspace 1.4-1 2019-03-18 [1] RSPM (R 4.0.0)   
## crayon 1.3.4 2017-09-16 [1] RSPM (R 4.0.0)   
## DBI 1.1.0 2019-12-15 [1] RSPM (R 4.0.0)   
## dbplyr 1.4.4 2020-05-27 [1] RSPM (R 4.0.0)   
## desc 1.2.0 2018-05-01 [1] RSPM (R 4.0.3)   
## devtools 2.3.2 2020-09-18 [1] RSPM (R 4.0.3)   
## digest 0.6.25 2020-02-23 [1] RSPM (R 4.0.0)   
## dplyr \* 1.0.2 2020-08-18 [1] RSPM (R 4.0.2)   
## ellipsis 0.3.1 2020-05-15 [1] RSPM (R 4.0.3)   
## evaluate 0.14 2019-05-28 [1] RSPM (R 4.0.3)   
## fansi 0.4.1 2020-01-08 [1] RSPM (R 4.0.0)   
## farver 2.0.3 2020-01-16 [1] RSPM (R 4.0.3)   
## forcats \* 0.5.0 2020-03-01 [1] RSPM (R 4.0.0)   
## fs 1.5.0 2020-07-31 [1] RSPM (R 4.0.3)   
## generics 0.0.2 2018-11-29 [1] RSPM (R 4.0.0)   
## ggplot2 \* 3.3.2 2020-06-19 [1] RSPM (R 4.0.1)   
## glue 1.6.1 2022-01-22 [1] CRAN (R 4.0.2)   
## gtable 0.3.0 2019-03-25 [1] RSPM (R 4.0.3)   
## haven 2.3.1 2020-06-01 [1] RSPM (R 4.0.2)   
## highr 0.8 2019-03-20 [1] RSPM (R 4.0.3)   
## hms 0.5.3 2020-01-08 [1] RSPM (R 4.0.0)   
## htmltools 0.5.0 2020-06-16 [1] RSPM (R 4.0.1)   
## httr 1.4.2 2020-07-20 [1] RSPM (R 4.0.3)   
## jsonlite 1.7.1 2020-09-07 [1] RSPM (R 4.0.2)   
## knitr 1.33 2022-02-15 [1] Github (yihui/knitr@a1052d1)   
## labeling 0.3 2014-08-23 [1] RSPM (R 4.0.0)   
## lifecycle 1.0.0 2021-02-15 [1] CRAN (R 4.0.2)   
## lubridate 1.7.9 2020-06-08 [1] RSPM (R 4.0.2)   
## magrittr 2.0.2 2022-01-26 [1] CRAN (R 4.0.2)   
## memoise 1.1.0 2017-04-21 [1] RSPM (R 4.0.0)   
## modelr 0.1.8 2020-05-19 [1] RSPM (R 4.0.3)   
## munsell 0.5.0 2018-06-12 [1] RSPM (R 4.0.3)   
## ottrpal 0.1.2 2022-02-15 [1] Github (jhudsl/ottrpal@1018848)   
## pillar 1.4.6 2020-07-10 [1] RSPM (R 4.0.2)   
## pkgbuild 1.1.0 2020-07-13 [1] RSPM (R 4.0.2)   
## pkgconfig 2.0.3 2019-09-22 [1] RSPM (R 4.0.3)   
## pkgload 1.1.0 2020-05-29 [1] RSPM (R 4.0.3)   
## prettyunits 1.1.1 2020-01-24 [1] RSPM (R 4.0.3)   
## processx 3.4.4 2020-09-03 [1] RSPM (R 4.0.2)   
## ps 1.3.4 2020-08-11 [1] RSPM (R 4.0.2)   
## purrr \* 0.3.4 2020-04-17 [1] RSPM (R 4.0.3)   
## R6 2.4.1 2019-11-12 [1] RSPM (R 4.0.0)   
## Rcpp 1.0.8 2022-01-13 [1] CRAN (R 4.0.2)   
## readr \* 1.4.0 2020-10-05 [1] RSPM (R 4.0.2)   
## readxl 1.3.1 2019-03-13 [1] RSPM (R 4.0.2)   
## remotes 2.2.0 2020-07-21 [1] RSPM (R 4.0.3)   
## reprex 0.3.0 2019-05-16 [1] RSPM (R 4.0.0)   
## rlang 0.4.10 2022-02-15 [1] Github (r-lib/rlang@f0c9be5)   
## rmarkdown 2.10 2022-02-15 [1] Github (rstudio/rmarkdown@02d3c25)  
## rprojroot 2.0.2 2020-11-15 [1] CRAN (R 4.0.2)   
## rstudioapi 0.11 2020-02-07 [1] RSPM (R 4.0.0)   
## rvest 1.0.1 2022-02-15 [1] Github (tidyverse/rvest@4fe39fb)   
## scales 1.1.1 2020-05-11 [1] RSPM (R 4.0.3)   
## sessioninfo 1.1.1 2018-11-05 [1] RSPM (R 4.0.3)   
## stringi 1.5.3 2020-09-09 [1] RSPM (R 4.0.3)   
## stringr \* 1.4.0 2019-02-10 [1] RSPM (R 4.0.3)   
## testthat 3.0.1 2022-02-15 [1] Github (R-lib/testthat@e99155a)   
## tibble \* 3.0.3 2020-07-10 [1] RSPM (R 4.0.2)   
## tidyr \* 1.1.2 2020-08-27 [1] RSPM (R 4.0.2)   
## tidyselect 1.1.0 2020-05-11 [1] RSPM (R 4.0.3)   
## tidyverse \* 1.3.0 2019-11-21 [1] RSPM (R 4.0.3)   
## usethis 2.1.5.9000 2022-02-15 [1] Github (r-lib/usethis@57b109a)   
## utf8 1.1.4 2018-05-24 [1] RSPM (R 4.0.3)   
## vctrs 0.3.4 2020-08-29 [1] RSPM (R 4.0.2)   
## withr 2.3.0 2020-09-22 [1] RSPM (R 4.0.2)   
## xfun 0.26 2022-02-15 [1] Github (yihui/xfun@74c2a66)   
## xml2 1.3.2 2020-04-23 [1] RSPM (R 4.0.3)   
## yaml 2.2.1 2020-02-01 [1] RSPM (R 4.0.3)   
##   
## [1] /usr/local/lib/R/site-library  
## [2] /usr/local/lib/R/library