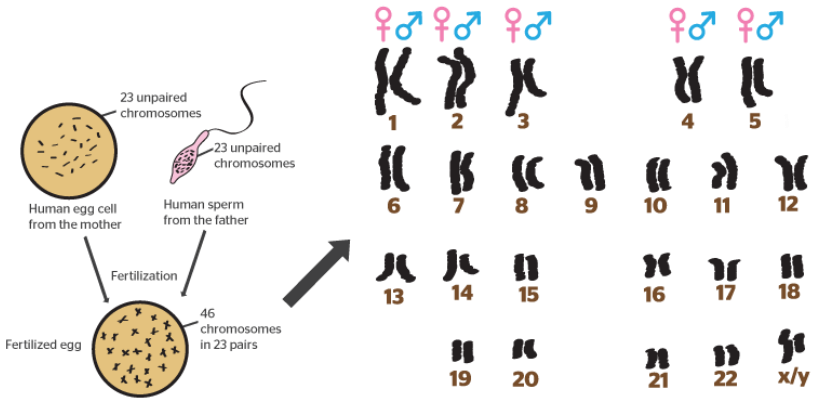
DNA-Based Prediction of Height

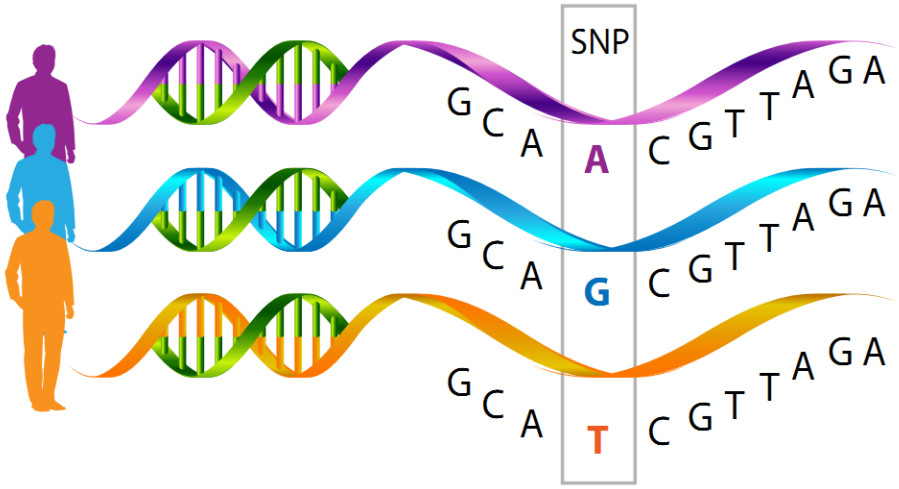
David Halvorsen

The essential biological code for a human consists of approximately 3 billion base pairs of DNA in a haploid (haploid means half the chromosome #) human sex cell. When a sperm fertilizes an egg the two haploid sex cells combine to collectively consist of ~ 6 billion base pairs of DNA. Those 6 billion base pairs are stored in 23 pairs of chromosomes. So to reiterate, human biological code is made up of 6 billion base pairs of DNA inside 46 chromosomes. Here's a helpful visual describing that in more detail.



http://b4fa.org/bioscience-in-brief/introduction-genes-crops/how-are-chromosomes-inherited/

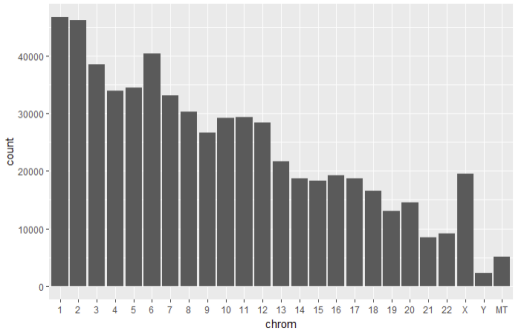
23andMe is a Mountain View based biology company that will sequence a small portion of your genome, called Single Nucleotide Polymorphisms (SNPs), for $200. Collectively, humans have most of the same biological code, BUT small DNA differences in your SNPs can change your biological characteristics; for more information about 23andMe, check out their scientific explanation page: https://www.23andme.com/genetic-science/. Here's an illustration of different *single* SNP changes across different humans.



The lower left of the next figure is the first couple of rows of my 23andMe raw data. First, let's talk about the column headings. Rsids are names for different genetic locations. Chromosome can be 1-22, X, Y or MT; 1-22 for the first 22 chromosomes, X or Y for sex differentiation (which we'll talk about more later) and MT for mitochondrial DNA. Position is the location of the DNA on the chromosome. Genotype is the two-letter code that describes the genes in your chromosomes. Remember how you inherit one chromosome from each parent? That's why you have two letters in your genotype. The R code you see is for reading in my rawdata.txt file, labeling columns, and then using ggplot to show the number of reads that were done for each of my chromosomes.

```{r}
library(ggplot2)
d <- read.table("/media/david/New_Volume/2018-06-13-DropboxLINUX/Dropbox/Linux_Sharing/Data_Analysis/Final_Project/genome_David_Halvorsen_v4_Full_20180510202712.txt",
                sep="\t", header=FALSE,
                colClasses=c("character", "character", "numeric", "character"),
                col.names=c("rsid", "chrom", "position", "genotype"))
tmp <- d$chrom
d$chrom = ordered(d$chrom, levels=c(seq(1, 22), "X", "Y", "MT"))
## It's never a bad idea to check your work
stopifnot(all(as.character(tmp) == as.character(d$chrom)))
ggplot(d) + geom_bar(aes(chrom))
```

```
# rsid  chromosome      position        genotype
rs12564807      1       734462  AA
rs3131972       1       752721  GG
rs148828841     1       760998  CC
rs12124819      1       776546  AA
rs115093905     1       787173  GG
rs11240777      1       798959  GG
rs7538305       1       824398  AA
rs4970383       1       838555  AC
rs4475691       1       846808  CC
rs7537756       1       854250  AA
rs13302982      1       861808  GG
rs55678698      1       864490  CC
i6019299        1       871267  CC
rs1110052       1       873558  GT
rs147226614     1       878697  GG
i6019302        1       881843  GG
rs2272756       1       882033  GG
rs67274836      1       884767  GG
i6019303        1       888554  CC
rs13302945      1       889159  CC
```



I want to predict physical characteristics from genetic data and that's not possible to do with *just* my data, so let's find more data. I decided to use Harvard's Personal Genome Project (PGP) as a data source: https://my.pgp-hms.org/public_genetic_data?data_type=23andMe. That webpage contains 941 rows of anonymous human names (labeled as 'Participant') and 941 rows of Download links. We'll need web scraping to get at this data. But first, here's a picture of the general layout for the Personal Genome Project data page. Additionally, there are 14 medical surveys that all of the participants have answered. We'll talk more about the medical surveys later.

	Participant	Published	Data type	Source	Name	Download	Report
	hu481B1D	2018-06-05	23andMe	Participant	23andMe_shala	Download (16.1 MB)	View report • female • 615,343 positions covered • ref. b37

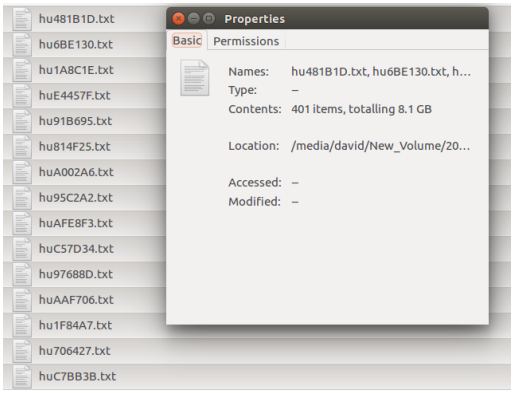
It's challenging to get the data because there is no easily accessible central download link. I used the rvest plugin to web scrape the PGP webpage with the read_html command. Before I go on, I should explain that HTML creates the layout for websites and CSS styles the structure. I used the SelectorGadget plugin (http://selectorgadget.com) to identify the css for the data download links. BUT, these data download links are not complete. As you can see from the example code below, all the code returned was the HTML for creating the link to the data. We can't do much with "Download", so we'll need more code to automate getting all the data.

```
## CREATE LIST OF SNP DATA DOWNLOAD LINKS (UNUSABLE HTML FOR FILES TO DOWNLOAD)
```{r}
# This code block visits the PGP website, saves the page as HTML, AND THEN selects the download links for all of the specimen data.
# rvest function read_html is used to save all the HTML
library(rvest)
public_SNP_data_HTML <- read_html("https://my.pgp-hms.org/public_genetic_data?data_type=23andMe")
# rvest function html_nodes is used to select CSS FOR ONLY the download links of interest
unusable_23andMe_HTML <- public_SNP_data_HTML %>%
  html_nodes(css="td:nth-child(7) a")
print(paste("This data has this many rows: ", length(unusable_23andMe_HTML), sep=''))
print(paste("This is the first row of the HTML data we'll need to trim: ", unusable_23andMe_HTML[1], sep=''))
```

 [1] "This data has this many rows: 941"
 [1] "This is the first row of the HTML data we'll need to trim: Download"
```

I used regexpr and substr text editing tools to splice together the full download link for the data. At first, it looked like the links in the format of 'https://my.pgp-hms.org/user_file/download/3546' were what I needed … I really did think I was done. But, then I realized it was going to get much more complicated.

```
## CREATE FUNCTION THAT TAKES UNUSABLE HTML AND TRIMS TO THE RELEVANT DOWNLOAD LINK TEXT
```{r}
# ^That HTML needs to get trimmed to have ONLY the relevant download link text. This function performs that task.
# regexpr is used to identify the text's positions of interest; those locations are used to trim the HTML to have
# ONLY the relevant download link text. I'll call this function inside a while loop later in the code.
funcUSABLE_download_path <- function(unusuable_HTML) {
  # print(paste('Current item is:', item, sep=''))
  starting = regexpr('user_file', unusuable_HTML)
  # print(paste("Path starting is", starting))
  stopping = regexpr('" ', unusuable_HTML)
  # print(paste("Path stopping is", stopping))
  name_starting = regexpr('download/', unusuable_HTML)
  # print(paste("File name starting is", name_starting))
  name_stopping = regexpr('" rel', unusuable_HTML)
  # print(paste("File name stopping is", name_stopping))
  file_name <- substr(unusuable_HTML, (name_starting+9), (name_stopping-1))
  # print(paste("file name ending is: ", file_name, sep=''))
  full_file_path <- paste('https://my.pgp-hms.org/user_file/download/', file_name, sep='')
  # print(paste("full file path is: '", full_file_path, "'", sep=''))
  return(full_file_path)
}
print(paste("We still have # of rows of: ", length(unusable_23andMe_HTML), sep=''))
print(paste("We've made those download links readable! Here's the first one: '", funcUSABLE_download_path(unusable_23andMe_HTML[1]), "' ", sep=''))
```

 [1] "We still have # of rows of: 941"
 [1] "We've made those download links readable! Here's the first one: 'https://my.pgp-hms.org/user_file/download/3546' "
```

Those links are actually redirection links, which I didn't initially think could be a problem. But then I noticed that the database that I was web scraping from had heterogenous file extensions (there's .PDF, .txt, .rtf, .zip and all sorts of others). I needed to know the file extension to properly save the files, so I decided to use getURL, from the RCurl package, to get the full file path of my files … that worked for the first ~100 files, so I felt comfortable leaving my code to download overnight, but then I woke up the following morning to this:

```
library(RCurl)
getURL("https://my.pgp-hms.org/user_file/download/2732")
```

```
Loading required package: bitops
rsid\tchromosome\tposition\tgenotype\r\nrs4477212\t1\t82154\tAA\r\nrs3094315\t1\t752566\tAA\r\nrs3131972\t1\t752721\tGG\r\nrs12124819\t1\t776546\tAA\r\nrs11240777\t1\t798959\tGG\r\n
.
\r\nrs3748597\t1\t888659\tCC\r\nrs13303106\t1\t891945\tGG\r\nrs28415373\t1\t893981\tCC\r\nrs13303010\t1\t894573\tAA\r\nrs6696281\t1\t903104\tCC\r\nrs28391282\t1\t904165\tGG\r\nrs234
.
\r\nrs6687776\t1\t1030565\tCC\r\nrs9651273\t1\t1031540\tGG\r\nrs11579015\t1\t1036959\tTT\r\nrs6671356\t1\t1040026\tTT\r\nrs4970405\t1\t1048955\tAA\r\nrs12726255\t1\t1049950\tAA\r\nr
.
\r\nrs12021879\t1\t1439671\tCC\r\nrs6690515\t1\t1447325\tAG\r\nrs6669795\t1\t1450947\tAC\r\nrs10159041\t1\t1453921\tCT\r\nrs3813216\t1\t1458567\tAG\r\nrs9439462\t1\t1462766\tCT\r\nr
.
\r\nrs2296716\t1\t1497824\tCC\r\nrs9439468\t1\t1499298\tAG\r\nrs6603793\t1\t1505255\tCT\r\nrs7520996\t1\t1509034\tCT\r\nrs7519837\t1\t1510801\tCT\r\nrs6687029\t1\t1519068\tAC\r\nrs6
.
\r\nrs28508199\t1\t1852484\tAA\r\nrs2144687\t1\t1853394\tTT\r\nrs2250833\t1\t1854321\tAG\r\nrs2803316\t1\t1865298\tAG\r\nrs12758705\t1\t1873625\tGG\r\nrs2803329\t1\t1874581\tAG\r\nr
.
\r\nrs947344\t1\t3423467\tCT\r\nrs11585362\t1\t3428608\tAG\r\nrs7543634\t1\t3431235\tAA\r\nrs12093117\t1\t3441264\tAG\r\nrs4648392\t1\t3446813\tGG\r\nrs10797400\t1\t3450637\tCC\r\nr
```

For some reason, getURL doesn't always just 'getURL'. Sometimes it downloads the entirety of a 600,000 line .txt file and posts it to the R window (crashing R). It'd have been nice if I noticed the problem right away, but I didn't experience it until download 122 and I had no tryCatch system on the download.file, so my downloading was canceled. My solution for this problem was to use the httr package for working with the internet HTTP messaging protocol to get the file name in an obscure way. I wrote a function that uses the HEAD command to request HTTP messaging protocol information for the desired file. I tinkered around until I discored that "HEAD(as.character(link))[3]$headers[5]" could be used to specifically select the file and extension info. See this example:

```
## FUNCTION THAT RETURNS THE FILENAME FROM THE FILENAME))))
```{r}
library(httr)
FUNC_attachment_filename <- function(link){
  file_name <- HEAD(as.character(link))[3]$headers[5]
  start <- regexpr("filename", as.character(file_name))
  start <- start + 10
  file_name <- substr(file_name, start, nchar(file_name))
  stop <- regexpr('"', as.character(file_name))
  file_name <- substr(file_name, 1, stop-1)
  return(file_name)
}
link <- "https://my.pgp-hms.org/user_file/download/2732"
print(paste("Download link is: ", link, sep=''))
print(paste("Filename is: ", FUNC_attachment_filename(link), sep=''))
```
```

```
[1] "Download link is: https://my.pgp-hms.org/user_file/download/2732"
[1] "Filename is: hu5BB600_20170422005743.txt"
```

BUT, this method of getting file and extension DID NOT work for all files; this method *ONLY* worked for the weird .txt files that getURL would post the entirety of to the screen. I used this other code to download the more normal files. It uses the getURL method to grab the download name as well as substr and regxpr string editing methods to get the complete file path. I used if/else logic to run the first httr HEAD file path grab => download.file, and if that failed, it ran the normal full file path grab method => download.file.

```
## SECTION 8: THIS FUNCTION GETS THE TRUE DOWNLOAD PATH AND FILE NAME AND EXTENSION
```{r}
library(RCurl)
# The PGP website stores links like 'https://my.pgp-hms.org/user_file/download/3539' BUT, those links DO NOT have the specimen.
# They redirect to warehouse.pgp-hms.org. This function repeats tools that were initially used in this function: funcUSABLE_download_path
funcGET_file_path <- function (url) {
  # Returns a text block WITH the file path within it. try is very important here because otherwise it'd crash the whole thing.
  true_file_location <- try(getURL(url))
  # print(paste("file path is:", true_file_location))
  # This identifies the starting point of the true file path
  file_location_start <- regexpr('http://', true_file_location)
  # print(paste("true path start is:", file_location_start))
  # This identifies the ending point of the true file path
  file_location_end <- regexpr('\">redirected', true_file_location)
  # print(paste("true path end is:", file_location_end))
  true_file_path <- substr(true_file_location, (file_location_start), (file_location_end-1))
  # print(paste("the full file path is: \'", true_file_path, "\'", sep=""))
  file_name <- basename(true_file_path)
  # print(paste("the file name is: ", file_name))
  # IF the code doesn't work, this makes it return 'the funcGET_file_path returned empty'
  if(nchar(true_file_path, keepNA=FALSE) == 0) {
    true_file_path <- 'the funcGET_file_path returned empty'
  }
  return(true_file_path)
}
# print(funcGET_file_path('https://my.pgp-hms.org/user_file/download/3546'))
```
```

```
[1] "http://warehouse.pgp-hms.org/warehouse/f18d303bd0308cf1d394fffee8eded09+89/genome_v5_Full_20180512193425.txt"
```

BUT, we want to connect these data files to the medical surveys for predicting traits AND we don't have the specimen names! The initial redirection download links have seemingly random numbers at the end and the other download links ~occassionally~ have helpful information, but they're mostly random. Some of the time, the file path has the full legal name of an individual, but that's not really something we want to have to count on for data accuracy. We want to compare DNA reads to Medical Surveys, so *ANY* mistaken row changes will break the connection between the sources of data and make everything else meaningless.

```
+   else {
+     # print("Normal warehouse routine triggered!")
+     print(PGP_linkDL_list[i])
+     print(funcGET_file_path(PGP_linkDL_list[i]))
+     if(file_ext(funcGET_file_path(PGP_linkDL_list[i])) == "txt"){
+       number_of_txt_files <- number_of_txt_files + 1
+     }
+   }
+   i <- i + 1
+ }
[1] "Current i is:  10  and # remaining is:  929"
[[1]]
[1] "https://my.pgp-hms.org/user_file/download/3516"

[1] "http://warehouse.pgp-hms.org/warehouse/ecb42e54e8ec123b13d4f15047375612+101/genome_Roxy_Hanson_v4_Full_20180218115824.rtf"
```

So I used the rvest package to web scrape the specimen names from that same download page. This code is mostly the same as the previous code, BUT the last code stopped after using html_nodes to select CSS. This new code pipes that command into html_text *because* I no longer was HTML code; I want the exact text of the link (which is contained in the HTML text). See below for an example of a sample identifier.

```
## SECTION 4: WEB SCRAPING FOR THE SPECIMEN NAMES
```{r}
# This code block visits the Personal Genome Project website and creates a list of all of the sample specimen_names
# rvest function read_html saves all the HTML
library(rvest)
public_SNP_data_HTML <- read_html("https://my.pgp-hms.org/public_genetic_data?data_type=23andMe")
# the web-scraped public_SNP_data is piped into the rvest 'html_nodes' command with a CSS selector for the desired specimen names.
# I used the "SelectorGadget" chrome extension to get the specimen names CSS of ".collection-column+ td a"
# You can find the "SelectorGadget" chrome extension here: https://chrome.google.com/webstore/detail/selectorgadget/mhjhnkcfbdhnjickkkdbjoemdmbfginb?hl=en
# Note that I piped html_nodes into html_text. That allows us to get the presented text; leaving it with html_nodes would give us the HTML instead.
specimen_names <- public_SNP_data_HTML %>%
html_nodes(".collection-column+ td a") %>%
html_text()
# Here's some debug code to make sure that we're getting the right output
typeof(specimen_names)
print(paste("This data has this many rows: ", length(specimen_names), sep=''))
print(paste("The first element of the specimen list is: ", specimen_names[1], sep=''))
```

[1] "character"
[1] "This data has this many rows: 941"
[1] "The first element of the specimen list is: hu481B1D"
```

That's everything needed for downloading the data and properly labeling it! I managed to get 8.1 GB. I wanted all 941 files, but I didn't know how to handle all the different extensions, so I stuck with .txt and that choice dropped the total number of workable files down to 401.



I have created a *simple* predictive model that can identify gender from DNA. It's predictions for 313 DNA files, and the matching medical surveys, were statistically indistinguishable from the reported survey #s for the frequency of men and women. Here's an excerpt from the PGPParticipantSurvey. The 14 medical surveys are located here: https://my.pgp-hms.org/google_surveys

Participant	Timestamp	Do not touch!	Year of birth	Gender
hu4A2DC0	7/15/2011 19:08	c62aa7e5166c0648f3858f971c43e2ce	30-39 years	Male
hu831FD6	7/15/2011 19:27	86800126c53ef3950abb82323cb3773f	21-29 years	Female
huD09050	7/15/2011 19:28	50c3404a74b779f67a3b89404a56533d	60-69 years	Male
hu5DA1EE	7/15/2011 19:36	73032adc432e8ca2277a09f238d13e84	40-49 years	Male
hu295B81	7/15/2011 21:08	6cf7cef3878c1668896c1c4f412cafdd	21-29 years	Male
huD90F39	7/15/2011 21:11	4430689b69965c36303b5c5bf89c6dcc	40-49 years	Female
hu34DC82	7/15/2011 21:23	82eb7d5627dafbf671cfb97701ec40e3	30-39 years	Female
hu4DFBAF	7/15/2011 21:44	f439798ad861178e8372aca4db65e560	30-39 years	Female
hu1D45E6	7/15/2011 22:16	dd5b63beb1bd19c0e218d111ec71974b	21-29 years	Male
huFCB8FD	7/15/2011 22:40	c2ec2bebd02df2b34b0d30112422e79c	60-69 years	Female
hu59502A	7/15/2011 23:07	517a89beed78fa40a2a18b2ad6a86a8a	30-39 years	Male
hu600DFB	7/16/2011 0:01	7c517ab62a70d673dea7f99d150859b5	21-29 years	Male

*Note that I moved the Gender column several columns to the left to be able to show this image.

I want to predict gender from the DNA and then independently check the survey responses to see if I'm correct. First, I'll get *reported* gender from the medical surveys. We have 4,103 rows of participant gender survey responses and 401 .txt files of DNA, so obviously we'll need to cut this group of items down to just have the matching ones.

I used list.files to obtain a complete list of all the DNA data files and created lists of those files with and without the .txt extension. I'll use the ones without the .txt extension to search for matches in surveys.

```
## Create list of 23andMe data WITH and WITHOUT .txt extension
```{r}
# does this maintain order? I don't trust it.
list_files <- list.files(path="/media/david/New_Volume/2018-06-13_DOWNLOAD_PGP/")
# it's of type list ... I should cycle throught he current list to create  a permanent list
# typeof(list_of_files_and_extensions)
i <- 1
list_of_files_WITH_EXTensions <- list()
list_of_files_WITHOUT_EXTensions <- list()
while(i <= length(list_files)){
  list_of_files_WITH_EXTensions[i] <- list_files[i]
  # -1 cause we don't want the "."
  right_before_start_of_TXT_extension <- regexpr(".txt", list_files[i]) - 1
  list_of_files_WITHOUT_EXTensions[i] <- substr(list_files[i], 1, right_before_start_of_TXT_extension)
  i <- i + 1
}
```
```

Here's reading in the medical survey with read.csv

```
## Read in the CSV table
```{r}
# I moved gender to the left to find it easier. ALSO changed column name to Gender
PGP_Participant_Gender_CSV <- read.csv(file="/media/david/New_Volume/Personal_Genome_Project/Surveys/PGPParticipantSurvey-20180609003013-GENDERmovedLEFT.csv")
PGP_Participant_Gender_CSV
```
```

I whittled down the TOTAL list of downloaded DNA data files to *ONLY* files that I had medical survey data by cycling through all of the file names and checking the CSV data for it with %in% to search through list_of_files_WITHOUT_EXTensions[i].

```
## generate list of files that HAVE GENDER INFO
```{r}
list_of_specimens_HAVE_GENDER_INFO <- list()
i <- 1
j <- 1
matches <- 0
while(i <= length(list_of_files_WITHOUT_EXTensions)){
  if(list_of_files_WITHOUT_EXTensions[i] %in% PGP_Participant_Gender_CSV$Participant){
    list_of_specimens_HAVE_GENDER_INFO[j] <- list_of_files_WITHOUT_EXTensions[i]
    j <- j + 1
    #print(paste("Matches: ", matches))
    matches <- matches + 1
  }
  i <- i + 1
}
matches
list_of_specimens_HAVE_GENDER_INFO
```
```

We grabbed the list of specimen names for DNA data files we have, so now we need to grab the reported genders from the medical surveys. I used careful iteration on I and j to make sure that I was consistently maintaining the match between specimen name and medical survey data.

```
## get gender info for each list_of_specimens_HAVE_GENDER_INFO
```{r}
list_of_specimens_HAVE_GENDER_INFO_REPORTEDGENDER <- list()
i <- 1
j <- 1
matches <- 0
while(i <= length(list_of_specimens_HAVE_GENDER_INFO)){
  #while(j <= length(PGP_Participant_Gender_CSV$Participant)){
  for(item in PGP_Participant_Gender_CSV$Participant){
    if(item %in% list_of_specimens_HAVE_GENDER_INFO_REPORTEDGENDER){
      next
    }
    else(item == list_of_specimens_HAVE_GENDER_INFO[i]) {
      print("MATCH TRIGGERED")
      print(paste("specimen name: ", list_of_specimens_HAVE_GENDER_INFO[i], " PGP_Survey Participant: ", item, " matches # is: ", matches, sep=''))
      list_of_specimens_HAVE_GENDER_INFO_REPORTEDGENDER[j] <- item
      matches <- matches + 1
      j <- j + 1
    }
    i <- 1
  }
  i <- 1 + i
}

```
```

We have medical survey reported gender, matching specimen names and matching DNA data files. Now it's time to cycle through the DNA data files to make predictions on gender. Chromosome 23 can be X or Y; a pair of X's (XX) mean that the person is female and a pair of an X and a Y means that the person is male. I thought that I could simply make an immediate prediction of gender based on reads of X and Y chromosomes, BUT 23andMe will occasionally report Y chromosomes for a female (supposed to just be XX) and it will occasionally report 2 DNA letters of data of XX for a biological male (supposed to just be XY). I had a hard time getting more information about this until I stumbled upon this Reddit post: "I have the raw data file of a male. The Illumina chips are made to probe the X chromosome twice so the X chromosome could be probed twice in the male and why they get C twice."

https://www.reddit.com/r/23andme/comments/6wahdo/im_a_woman_yet_23_and_me_shows_a_partial_y/

I created a more complicated predictive set of code to handle these weird extra chromosome reads. I couldn't fit all of the code in this screenshot, so you'll have to look at my codebase to grab it. These files are 600,000 rows long and the X and Y chromosomes are at the very end of the file, so I started the row iterator out at the length of the rows and it iterates backwards from there. Also, you can't immediately make the gender prediction from a single event of an absence of a Y chromosome (to predict female) or the absence of an X chromosome (to predict male) because sometimes the DNA sequencing machine had a failure while reading that specific DNA site. This code reads 5 different DNA sites on the Y chromosome. If there are three or more Y chromosome blank reads, the gender is assumed to be female. If there are 2 or less Y chromosome blanks the gender is assumed to be male.

```
while(searching_for_Y_chromosome){
  #print(paste("Current file number is: ", current_file_iterator, " and # remaining files is: ", length(list_of_specimens_HAVE_GENDER_INFO)-current_file_iterator, sep=''))
  #print(paste("Current row is: ", current_row_iterator, " Current specimen is: ", as.character(list_of_specimens_HAVE_GENDER_INFO[current_file_iterator]), sep=''))
  if(current_23andme_file$chrom[current_row_iterator] == "Y"){
    Y_chromosome_reads <- Y_chromosome_reads + 1
    if(current_23andme_file$genotype[current_row_iterator] == "--"){
      number_of_Y_blanks <- number_of_Y_blanks + 1
      # predicted_gender <- "FEMALE"
      # searching_for_Y_chromosome <- FALSE
    }
  }
  if(Y_chromosome_reads == 5){
    if(number_of_Y_blanks >= 3){
      predicted_gender_list[current_file_iterator] <- "FEMALE"
      print(paste("Current row is: ", current_row_iterator, " Current specimen is: ", as.character(list_of_specimens_HAVE_GENDER_INFO[current_file_iterator]), sep=''))
      print(paste("FEMALE predicted and ", predicted_gender_list[current_file_iterator], "stored", sep=''))
      searching_for_Y_chromosome <- FALSE
    }
    else if(number_of_Y_blanks <= 2){
      predicted_gender_list[current_file_iterator] <- "MALE"
      print(paste("Current row is: ", current_row_iterator, " Current specimen is: ", as.character(list_of_specimens_HAVE_GENDER_INFO[current_file_iterator]), sep=''))
      print(paste("MALE predicted and ", predicted_gender_list[current_file_iterator], "stored", sep=''))
      searching_for_Y_chromosome <- FALSE
    }
    else{
      predicted_gender_list[current_file_iterator] <- "UNKNOWN_GENDER_ERROR"
      print(paste("Current row is: ", current_row_iterator, " Current specimen is: ", as.character(list_of_specimens_HAVE_GENDER_INFO[current_file_iterator]), sep=''))
      print(paste("ERROR predicted and ", predicted_gender_list[current_file_iterator], "stored", sep=''))
      searching_for_Y_chromosome <- FALSE
    }
  }
  current_row_iterator <- current_row_iterator - 100
  if(current_row_iterator <= 45000){
    predicted_gender_list[current_file_iterator] <- "NO_MATCH_ERROR"
    print("NO_MATCH_ERROR")
    searching_for_Y_chromosome <- FALSE
  }
}
```

I created a dataframe from the specimen names, the medical survey reported genders and the genders that I predicted.

Specimen <fctr>	Gender <fctr>	Predicted_Gender <chr>
hu0684DB	Female	FEMALE
hu092771	Female	FEMALE
hu09D626	Female	FEMALE
hu0A6570	Male	MALE
hu0B4CF6	Female	FEMALE
hu0B51B8	Female	FEMALE
hu0B9C47	Male	MALE
hu0C0779	Female	FEMALE
hu0C8573	Male	MALE
hu0DEE68	Male	MALE

11-20 of 100 rows

This code compared my predictions to the actual reported genders of the participants.

```
## compare predicitons to survey responses
```{r}
i <- 1
matches <- 0
while(i <= length(predicted_gender_list)){
#while(i <= 64){
  print(i)
  print(paste("Current specimen is: ", list_of_specimens_HAVE_GENDER_INFO[i], sep=''))
  print(paste("Gender prediction is: ", predicted_gender_list[i], sep=''))
  print(paste("Reported gender: ", list_of_specimens_HAVE_GENDER_INFO_REPORTEDGENDER[i], sep=''))
  if(tolower(as.character(predicted_gender_list[i])) == tolower(as.character(list_of_specimens_HAVE_GENDER_INFO_REPORTEDGENDER[i]))){
    print("HECK YEA!")
    matches <- matches + 1
  }
  i <- i + 1
}
print(paste("The number of matches is: ", matches, sep=''))
```
```

I used a chi-square test to compare the medical survey reported genders against the genders that I predicted. My p-value was incredibly low!

```{r}
library(MASS)
chisq.test(tolower(predicted_gender_list[1:100]), tolower(list_of_specimens_HAVE_GENDER_INFO_REPORTEDGENDER[1:100]))
```

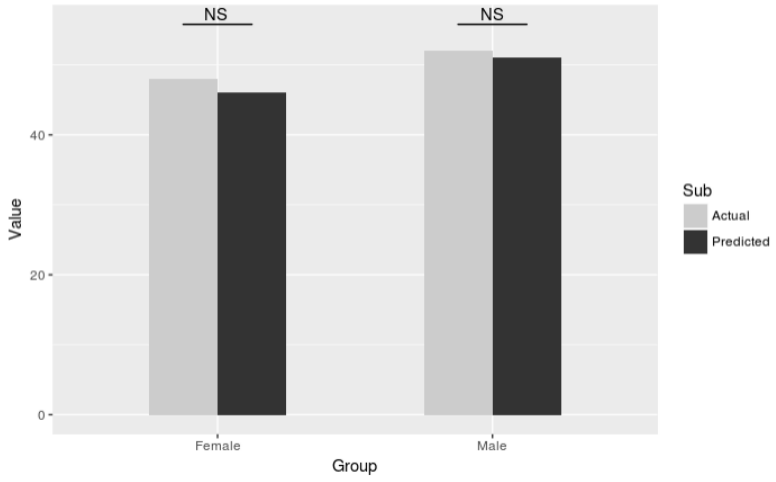
```
Chi-squared approximation may be incorrect
        Pearson's Chi-squared test

data:  tolower(predicted_gender_list[1:100]) and tolower(list_of_specimens_HAVE_GENDER_INFO_REPORTEDGENDER[1:100])
X-squared = 100, df = 3, p-value < 2.2e-16
```

I also used the ggsignif package to create a bar graph of my findings.

```{r}
library(ggplot2)
library(ggsignif)

dat <- data.frame(Group = c("Female", "Female", "Male", "Male"),
                  # Sub is the data sub-type (predicted vs. actual)
                  Sub = c("Actual", "Predicted", "Actual", "Predicted"),
                  # Value is the values
                  Value = c(Female,FEMALE,Male,MALE))

ggplot(dat, aes(Group, Value)) +
  geom_bar(aes(fill = Sub), stat="identity", position="dodge", width=.5) +
  geom_signif(stat="identity",
              data=data.frame(x=c(0.875, 1.875), xend=c(1.125, 2.125),
                              y=c(55.8, 55.8), annotation=c("NS ", "NS")),
              aes(x=x,xend=xend, y=y, yend=y, annotation=annotation)) +
  geom_signif(comparisons=list(c("S1", "S2")), annotations="***",
              y_position = 69.3, tip_length = 0, vjust=0.4) +
  scale_fill_manual(values = c("grey80", "grey20"))
```



I wanted to predict much more than just gender, but I was significantly unprepared for how intensive the data handling methods would be for this project. However, I did find an appreciation for the complexities of data cleaning. This is definitely the most challenging data analysis project I have ever undertaken. I had also attempted to predict lactose intolerance, height and eye color, BUT I was unable to identify anything significant. There's 2 reasons I believe that I ran into trouble with those more complicated tasks: 1) I only used one SNP at a time for comparisons (23andMe uses multiple SNPs for it's "genoset" predictions) and 2) I may have made a data handling issue and lost the connection between the specimen names, the DNA data or the CSV surveys. I managed to get the gender prediction after I had carefully reworked my eye color prediction code and focusing on being VERY careful with maintaining which data belong to what specimen. I will be repeating the other prediction methods to see if I can get them to work.

I also have a much longer term vision for the future of this project. In the near term, I'd like to use genosets to make predictions. Underneath this text, to the left, you'll see a 23andMe "genoset" for blue eye color. On the right you'll see me selecting CSV rows for mentioning "blue" eyes.

Gs237/criteria

< Gs237

```
and(
        rs4778241(C;C),
        rs12913832(G;G),
        rs7495174(A;A),
        rs8028689(T;T),
        rs7183877(C;C),
        rs1800401(C;C))
```

```
## print Left_Eye ONLY if 'blue' is within it
```{r}
i <- 1
blue_eye_count <- 1
while(i <= length(Phenotype_Survey$Participant)){
  print(i)
  if(grepl("blue", tolower(Phenotype_Survey$Left_Eye[i]))){
    print(Phenotype_Survey$Left_Eye[i])
    blue_eye_count <- blue_eye_count + 1
  }
  i <- i + 1
}
```
```

```
[1] 1
[1] greyish blue
500 Levels:  11 #16 is as close as it gets 17 a light hazel/
[1] 2
[1] blue
```

This is the start of that much more complicated eye color prediction analytical route in a dataframe:

specimen <chr>	rs12913832 <chr>	rs1667394 <chr>	rs1800407 <chr>	rs1393350 <chr>	rs1800401 <chr>	rs7495174 <chr>
hu00698E	GG	TT	CC	AG	GG	AA
hu0257DA	GG	TT	CC	GG	GG	AA
hu029DBB	AG	TT	CC	GG	AG	AA
hu034DB1	GG	CC	CC	GG	GG	AA
hu05CA34	AG	TT	CC	AG	GG	AA
hu092771	GG	TT	CC	AG	AG	AA
hu0B51B8	AA	TT	CC	AG	GG	AA
hu0B9C47	GG	CT	CC	GG	GG	AG
hu0B9C47	GG	TT	CC	AA	GG	AA
hu0C0779	AG	TT	CC	GG	GG	AA

1-10 of 145 rows

Previous 1 2 3 4 5 6 … 15 Next

Also, looking much farther into the future, I'd like to try to create a predictive model for a much more complicated condition. I managed to find a machine learning paper that predicted high blood pressure from the metrics in the table to the lower left. Using machine learning to predict hypertension from a clinical dataset https://ieeexplore.ieee.org/document/7849886/. They had data for physically measurable traits like blood pressure and cholesterol. I have metrics for those kinds of data, like SNPs that influence high vs. low cholesterol, BMI, triglycerides and general heart function. In the lower right, you'll see a few dozen rows of medical patients that had reported hypertension. In a future project, I'd like to try to predict those much more complicated outcomes.

TABLE I. NETWORK INPUT FACTORS

#	Variable (unit of measure)	Type (mean ± standard deviation)
1	Birth Year	Numeric (56.59±14.65)
2	Gender	Categorical (0,1)
3	Body Mass Index (kg/m^2)	Numeric (28.57±5.29)
4	Systolic Blood Pressure (mmHg)	Numeric (127.22±14.81)
5	Diastolic Blood Pressure (mmHg)	Numeric (78.39±9.08)
6	High Density Lipoprotein (mmol/L)	Numeric (1.40±0.33)
7	Low Density Lipoprotein (mmol/L)	Numeric (2.96±0.72)
8	Triglycerides (mmol/L)	Numeric (1.44±0.64)
9	Cholesterol (mmol/L)	Numeric (4.99±0.83)
10	Micro-albumin (mg/L)	Numeric (29.46±31.68)
11	Urine Albumin-Creatinine Ratio (mg/mmol)	Numeric (4.14±4.87)

341	hu46125C	Hypertension
342	huE31062	Hypertension
343	huAEC1B0	Hypertension
344	hu5FD75D	Hypertension
345	hu034DB1	Hypertension
346	hu474789	Hypertension
347	hu3EBF80	Hypertension
348	hu3CC2C0	Hypertension
349	hu94040B	Hypertension
350	hu904B18	Hypertension
351	huBC03A7	Hypertension
352	huEBD467	Hypertension
353	huA9D22A	Hypertension
354	hu0FB755	Hypertension
355	huE8554D	Hypertension
356	hu6E46C7	Hypertension
357	hu74B85B	Hypertension
358	hu165E62	Hypertension
359	hu20C5DA	Hypertension
360	hu0D1FA1	Hypertension
361	hu578827	Hypertension