Assignment 3

Group15

2025-11-06

Sarah Dumont

Sophia Liau

Minh Duong

Question 1

There are several reason why area of genome with high GC content is hard to sequence:

1.1 Thermal Stability and Secondary Structures

- GC base pairs form three hydrogen bonds compared to the two in AT base pairs. As a result, GC-rich regions have a higher melting temperature (T_m) and are more thermally stable. During denaturation and amplification (e.g., in PCR steps of sequencing library preparation), these regions may remain partially double-stranded, leading to incomplete amplification.
- GC-rich sequences tend to form stable secondary structures such as hairpins and G-quadruplexes, which can stall DNA polymerase or cause premature termination of synthesis.

1.2. PCR Amplification Bias

- Most next-generation sequencing (NGS) platforms rely on PCR to amplify DNA fragments before sequencing. GC-rich fragments amplify inefficiently because polymerases can stall or dissociate when encountering secondary structures. This leads to an amplification bias, where GC-rich regions are underrepresented or entirely missing from the sequencing data.

Question 2

We want to find the best global alignment between the two following sequences with the provided scoring matrix:

ATTCGAC

ATCAC

Let's first define the sequence, the parameter and create the scoring matrix. We will split the above strings into separate characters, then convert the resulting list into a vector. We will also define the gap penalty to be -2 for every gap occurring in the resulting sequence.

```r
seq1 <- unlist(strsplit("ATTCGAC", ""))  # horizontal (rows)
seq2 <- unlist(strsplit("ATCAC", ""))    # vertical (cols)
gap_penalty <- -2

#create scoring matrix from the assignment document
score_matrix <- matrix(
  c( 1, -5, -5, -1,
    -5,  1, -1, -5,
    -5, -1,  1, -5,
    -1, -5, -5,  1),
  nrow = 4, byrow = TRUE,
  dimnames = list(c("A","T","C","G"), c("A","T","C","G"))
)
```

Next, we will make a 2 dimensional grid with an extra row and column for the Needleman-Wunsch Method. Each cell dp[i, j] will stores the best alignment score between the first i lettwers of seq2 and the first j letters of seq1. We also fill the first row and column based on the gap penalty.

```r
#initializing the grid
n <- length(seq1)
m <- length(seq2)
dp <- matrix(0, nrow = m + 1, ncol = n + 1)

#fill first row and columns
for (i in 2:(m + 1)) dp[i,1] <- dp[i-1,1] + gap_penalty
for (j in 2:(n + 1)) dp[1,j] <- dp[1,j-1] + gap_penalty
```

We will fill the rest of the grid

```r
for (i in 2:(m + 1)) {
  for (j in 2:(n + 1)) {
    match_score <- score_matrix[seq2[i-1], seq1[j-1]]
    dp[i,j] <- max(
      dp[i-1,j-1] + match_score,  # diagonal (match/mismatch)
      dp[i-1,j] + gap_penalty,    # up (gap in seq1)
      dp[i,j-1] + gap_penalty     # left (gap in seq2)
    )
  }
}

#print the table
dp
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    0   -2   -4   -6   -8  -10  -12  -14
## [2,]   -2    1   -1   -3   -5   -7   -9  -11
## [3,]   -4   -1    2    0   -2   -4   -6   -8
## [4,]   -6   -3    0    1    1   -1   -3   -5
## [5,]   -8   -5   -2   -1   -1    0    0   -2
## [6,]  -10   -7   -4   -3    0   -2   -2    1
```

```r
dp[1:3, 1:3]
```

```
##      [,1] [,2] [,3]
## [1,]    0   -2   -4
## [2,]   -2    1   -1
## [3,]   -4   -1    2
```

Now we will traceback to find the best alignment. Starting from the bottom-right cell, the traceback reconstructs how the optimal alignment was formed. If the score came from the diagonal, both letters were aligned. If from up, a gap was added in seq1. If from left, a gap was added in seq2. The loop continues until both i and j reach 1 (the top-left corner).

```r
i <- m + 1
j <- n + 1
align1 <- c()
align2 <- c()

while (i > 1 || j > 1) {
  if (i > 1 && j > 1 && dp[i,j] == dp[i-1,j-1] + score_matrix[seq2[i-1], seq1[j-1]]) {
    align1 <- c(seq1[j-1], align1)
    align2 <- c(seq2[i-1], align2)
    i <- i - 1
    j <- j - 1
  } else if (i > 1 && dp[i,j] == dp[i-1,j] + gap_penalty) {
    align1 <- c("-", align1)
    align2 <- c(seq2[i-1], align2)
    i <- i - 1
  } else {
    align1 <- c(seq1[j-1], align1)
    align2 <- c("-", align2)
    j <- j - 1
  }
}

cat("Best alignment:\n")
```

```
## Best alignment:
```

```r
cat(paste(align1, collapse = ""), "\n")
```

```
## ATTCGAC
```

Question 3

3.1. We want to load the first 73 lines of the header of the file and print the contents

```r
df <- read.csv("single_cell_RNA_seq_bam.sam", nrows=73, sep="\t", header=FALSE, fill=TRUE)
```

According to the header table in section 1.3 of the BAM/SAM document in the appendix, we have:

- SN tag: this is the reference sequence name

- LN tag: this is reference sequence length that ranges from 1 to $2^{31} - 1$

3.2. The length of the X chromosome, in bp, for our alignment is:

```
#Find the row where SN is "X"
x_row <- df[grepl("SN:X", df$V2), ]

#Extract the length in base pairs
x_length <- sub("LN:", "", x_row$V3)
x_length <- as.numeric(x_length)

x_length
```

```
## [1] 171031299
```

Question 4

4.1. The number of reads in this BAM file is:

```
sam <- read.csv("single_cell_RNA_seq_bam.sam", sep="\t", header=FALSE,
                comment.char="@", col.names = paste0("v", seq_len(30)), fill=TRUE)
sam <- sam[paste0("v", seq_len(11))]
nrow(sam)
```

```
## [1] 146346
```

4.2. Let's first print out the 10th row of a dataframe to look at the format of a read

```
sam[10, ]
```

```
##                                          v1 v2 v3      v4  v5  v6 v7 v8 v9
## 10 NS500668:199:HV73CBGX2:1:11203:20546:3351 16  1 3365976 255 58M  *  0  0
##                                                                        v10
## 10 AATCAAAAAGGGGGCTGTCAGTAGGATGATATAAGATATAGATGTAGTTTATCTCCTA
##                                                                        v11
## 10 EEEEEEEEEEA//AAAAEEEEEE/AEEAEAEEEEEEEEEEEEEAAEE///EEEAAA6A
```

According to section 1.4 of the BAM documentation, we can see that to find the chromosone to which the read was aligned, we should look at column 3, which is the reference sequence name. We can see that this is chromosome 1. This is the **V3** column. The V11 column correspond to **base quality scores (ASCII-encoded)**.

4.3. The number of reads correspond to chromosome X is:

```
numX <-sum(sam$v3 == "X")
numX
```

```
## [1] 5999
```

4.4. The base quality string is in V11 (QUAL field) and each character encode a quality score using Phred+33 encoding. We will convert this score to a numeric score, then average them for all reads aligned to X

```r
#Subset reads aligned to chromosome X
x_reads <- sam[sam$v3 == "X", ]

#Convert QUAL strings to numeric BQ and compute the mean
bq_values <- unlist(lapply(x_reads$v11, function(q) as.integer(charToRaw(q)) - 33))
mean_bq_x <- mean(bq_values, na.rm = TRUE)
cat("The average BQ scores for all read aligned to chromosome X is: ")
```

```
## The average BQ scores for all read aligned to chromosome X is:
```

```r
cat(paste(mean_bq_x, collapse = ""), "\n")
```

```
## 32.7234912715338
```

4.5. We want to plot the distribution of BQs across all bases and reads as a boxplot. We will first convert all QUAL fields to Phred scores

```r
qual_list <- lapply(sam$v11, function(q) utf8ToInt(q) - 33)
max_len <- max(sapply(qual_list, length))
max_len
```

```
## [1] 58
```

```r
# Pad all reads with NA so each read has equal length
qual_mat <- t(sapply(qual_list, function(x) c(x, rep(NA, max_len - length(x)))))
```

We see that the maximum read length is 58, meaning the reads in the SAM/BAM file are only 58 bp long. Now we will create a list of Phred scores for each base position, for example:

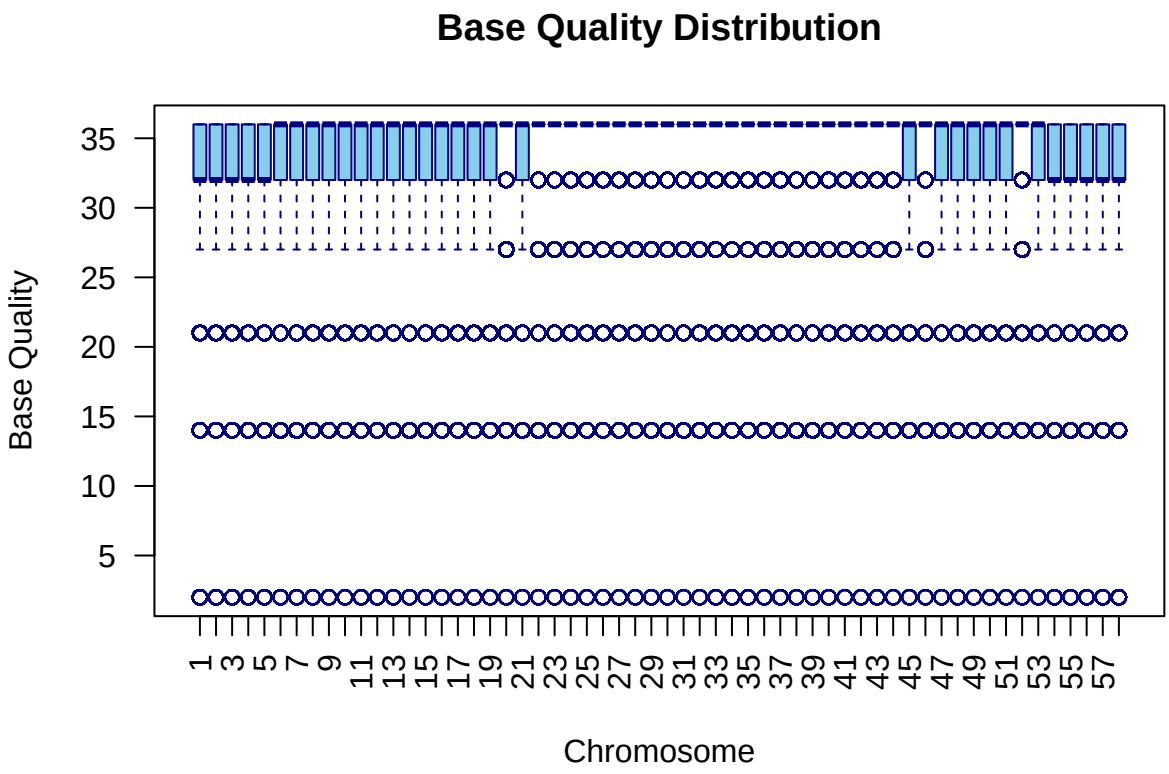
bq_list[[1]] = qualities at position 1 across all reads

```r
bq_list <- lapply(1:max_len, function(i) qual_mat[, i])

names(bq_list) <- paste0(1:max_len)
```

Now we can plot the distribution of basequality across all bases and reads

```r
boxplot(bq_list,
        las = 2,
        col = "skyblue",
        border = "navy",
        main = "Base Quality Distribution",
        xlab = "Chromosome",
        ylab = "Base Quality",
        outline = TRUE)
```



The base quality distribution shows a high median Phred score of around 35 to 36 for all positions, indicating great sequencing accuracy (>99.9%). Most bases fall within the Q30–Q37 range, with only a few low-quality outliers below Q25. This suggests that the dataset is of high overall quality, and downstream analyses can be performed with confidence.

4.6. Section 1.4 of the SAM/BAM documentation indicates V4 as the leftmost mapping position of the reads

4.7. We want to find reads have their leftmost mapping position aligned within bases 40801273 - 40805199 at chromosome 9 which encode protein Hspa8.

```
#Filter reads from chromosome 9 and within the provided coordinates
hspa8_reads <- sam[sam$v3 == "9" & sam$v4 >= 40801273 & sam$v4 <= 40805199, ]
nrow(hspa8_reads)
```

```
## [1] 119
```

4.8. Section 1.4 of the SAM/BAM documentation indicates V5 as the mapping quality. The number of reading iwth mapping quality less than 50 is:

```
sum(sam$v5 < 50)
```

```
## [1] 61527
```

4.9. The mean mapping quality of the reads which have mapping quality less than 50 is:

```
mean(sam$v5[sam$v5 < 50])
```

```
## [1] 0.2418125
```

4.10. The number of reads which align to the tdTomato sequence is:

```
sum(sam$v3 == "tdTomato")
```

```
## [1] 63
```

We see that the cell expresses tdTomato, so it should emit fluorescence under appropriate light. Fluorophore tags can help researchers with visually track gene expression or identify specific cell types under a microscope, separate cell populations using fluorescence.

Question 5

5.1. Let's first obtain the header of the file and a dataframe where each row is a variant.

```
vcf_con <- file("RNA_seq_annotated_variants.vcf", open="r")
vcf_file <- readLines(vcf_con)
close(vcf_con)
vcf <- data.frame(vcf_file)
header <- vcf[grepl("##", vcf$vcf_file), ]
factor(header)
```

```
##   [1] ##fileformat=VCFv4.1
##   [2] ##fileDate=20200930
##   [3] ##source=strelka
##   [4] ##source_version=2.9.2
##   [5] ##startTime=Wed Sep 30 13:12:59 2020
##   [6] ##contig=<ID=1,length=195471971>
##   [7] ##contig=<ID=10,length=130694993>
##   [8] ##contig=<ID=11,length=122082543>
##   [9] ##contig=<ID=12,length=120129022>
##  [10] ##contig=<ID=13,length=120421639>
##  [11] ##contig=<ID=14,length=124902244>
##  [12] ##contig=<ID=15,length=104043685>
##  [13] ##contig=<ID=16,length=98207768>
##  [14] ##contig=<ID=17,length=94987271>
##  [15] ##contig=<ID=18,length=90702639>
##  [16] ##contig=<ID=19,length=61431566>
##  [17] ##contig=<ID=2,length=182113224>
##  [18] ##contig=<ID=3,length=160039680>
##  [19] ##contig=<ID=4,length=156508116>
##  [20] ##contig=<ID=5,length=151834684>
##  [21] ##contig=<ID=6,length=149736546>
##  [22] ##contig=<ID=7,length=145441459>
##  [23] ##contig=<ID=8,length=129401213>
##  [24] ##contig=<ID=9,length=124595110>
##  [25] ##contig=<ID=MT,length=16299>
##  [26] ##contig=<ID=X,length=171031299>
##  [27] ##contig=<ID=Y,length=91744698>
##  [28] ##contig=<ID=JH584299.1,length=953012>
```

```
##   [29] ##contig=<ID=GL456233.1,length=336933>
##   [30] ##contig=<ID=JH584301.1,length=259875>
##   [31] ##contig=<ID=GL456211.1,length=241735>
##   [32] ##contig=<ID=GL456350.1,length=227966>
##   [33] ##contig=<ID=JH584293.1,length=207968>
##   [34] ##contig=<ID=GL456221.1,length=206961>
##   [35] ##contig=<ID=JH584297.1,length=205776>
##   [36] ##contig=<ID=JH584296.1,length=199368>
##   [37] ##contig=<ID=GL456354.1,length=195993>
##   [38] ##contig=<ID=JH584294.1,length=191905>
##   [39] ##contig=<ID=JH584298.1,length=184189>
##   [40] ##contig=<ID=JH584300.1,length=182347>
##   [41] ##contig=<ID=GL456219.1,length=175968>
##   [42] ##contig=<ID=GL456210.1,length=169725>
##   [43] ##contig=<ID=JH584303.1,length=158099>
##   [44] ##contig=<ID=JH584302.1,length=155838>
##   [45] ##contig=<ID=GL456212.1,length=153618>
##   [46] ##contig=<ID=JH584304.1,length=114452>
##   [47] ##contig=<ID=GL456379.1,length=72385>
##   [48] ##contig=<ID=GL456216.1,length=66673>
##   [49] ##contig=<ID=GL456393.1,length=55711>
##   [50] ##contig=<ID=GL456366.1,length=47073>
##   [51] ##contig=<ID=GL456367.1,length=42057>
##   [52] ##contig=<ID=GL456239.1,length=40056>
##   [53] ##contig=<ID=GL456213.1,length=39340>
##   [54] ##contig=<ID=GL456383.1,length=38659>
##   [55] ##contig=<ID=GL456385.1,length=35240>
##   [56] ##contig=<ID=GL456360.1,length=31704>
##   [57] ##contig=<ID=GL456378.1,length=31602>
##   [58] ##contig=<ID=GL456389.1,length=28772>
##   [59] ##contig=<ID=GL456372.1,length=28664>
##   [60] ##contig=<ID=GL456370.1,length=26764>
##   [61] ##contig=<ID=GL456381.1,length=25871>
##   [62] ##contig=<ID=GL456387.1,length=24685>
##   [63] ##contig=<ID=GL456390.1,length=24668>
##   [64] ##contig=<ID=GL456394.1,length=24323>
##   [65] ##contig=<ID=GL456392.1,length=23629>
##   [66] ##contig=<ID=GL456382.1,length=23158>
##   [67] ##contig=<ID=GL456359.1,length=22974>
##   [68] ##contig=<ID=GL456396.1,length=21240>
##   [69] ##contig=<ID=GL456368.1,length=20208>
##   [70] ##contig=<ID=JH584292.1,length=14945>
##   [71] ##contig=<ID=JH584295.1,length=1976>
##   [72] ##contig=<ID=tdTomato,length=2250>
##   [73] ##contig=<ID=SSM2_GFP,length=1619>
##   [74] ##contig=<ID=CreERT2,length=1983>
##   [75] ##content=strelka germline small-variant calls
##   [76] ##INFO=<ID=END,Number=1,Type=Integer,Description="End position of the region described in this
##   [77] ##INFO=<ID=BLOCKAVG_min30p3a,Number=0,Type=Flag,Description="Non-variant multi-site block. Non
##   [78] ##INFO=<ID=SNVHPOL,Number=1,Type=Integer,Description="SNV contextual homopolymer length">
##   [79] ##INFO=<ID=CIGAR,Number=A,Type=String,Description="CIGAR alignment for each alternate indel al
##   [80] ##INFO=<ID=RU,Number=A,Type=String,Description="Smallest repeating sequence unit extended or c
##   [81] ##INFO=<ID=REFREP,Number=A,Type=Integer,Description="Number of times RU is repeated in referen
##   [82] ##INFO=<ID=IDREP,Number=A,Type=Integer,Description="Number of times RU is repeated in indel al
```

```
##  [83] ##INFO=<ID=MQ,Number=1,Type=Integer,Description="RMS of mapping quality">
##  [84] ##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##  [85] ##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##  [86] ##FORMAT=<ID=GQX,Number=1,Type=Integer,Description="Empirically calibrated genotype quality sc
##  [87] ##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Filtered basecall depth used for site genoty
##  [88] ##FORMAT=<ID=DPF,Number=1,Type=Integer,Description="Basecalls filtered from input prior to site
##  [89] ##FORMAT=<ID=MIN_DP,Number=1,Type=Integer,Description="Minimum filtered basecall depth used for
##  [90] ##FORMAT=<ID=AD,Number=.,Type=Integer,Description="Allelic depths for the ref and alt alleles i
##  [91] ##FORMAT=<ID=ADF,Number=.,Type=Integer,Description="Allelic depths on the forward strand">
##  [92] ##FORMAT=<ID=ADR,Number=.,Type=Integer,Description="Allelic depths on the reverse strand">
##  [93] ##FORMAT=<ID=FT,Number=1,Type=String,Description="Sample filter, 'PASS' indicates that all filt
##  [94] ##FORMAT=<ID=DPI,Number=1,Type=Integer,Description="Read depth associated with indel, taken fro
##  [95] ##FORMAT=<ID=PL,Number=G,Type=Integer,Description="Normalized, Phred-scaled likelihoods for gen
##  [96] ##FORMAT=<ID=PS,Number=1,Type=Integer,Description="Phase set identifier">
##  [97] ##FORMAT=<ID=SB,Number=1,Type=Float,Description="Sample site strand bias">
##  [98] ##FILTER=<ID=IndelConflict,Description="Indel genotypes from two or more loci conflict in at lo
##  [99] ##FILTER=<ID=SiteConflict,Description="Site is filtered due to an overlapping indel call filter
## [100] ##FILTER=<ID=LowGQX,Description="Locus GQX is below threshold or not present">
## [101] ##FILTER=<ID=HighDPFRatio,Description="The fraction of basecalls filtered out at a site is grea
## [102] ##FILTER=<ID=HighSNVSB,Description="Sample SNV strand bias value (SB) exceeds 10">
## [103] ##FILTER=<ID=LowDepth,Description="Locus depth is below 3">
## [104] ##FILTER=<ID=NotGenotyped,Description="Locus contains forcedGT input alleles which could not be
## [105] ##FILTER=<ID=PloidyConflict,Description="Genotype call from variant caller not consistent with
## [106] ##FILTER=<ID=NoPassedVariantGTs,Description="No samples at this locus pass all sample filters a
## [107] ##SnpEffVersion="4.3t (build 2017-11-24 10:18), by Pablo Cingolani"
## [108] ##INFO=<ID=ANN,Number=.,Type=String,Description="Functional annotations: 'Allele | Annotation
## [109] ##INFO=<ID=LOF,Number=.,Type=String,Description="Predicted loss of function effects for this va
## [110] ##INFO=<ID=NMD,Number=.,Type=String,Description="Predicted nonsense mediated decay effects for
## 110 Levels: ##content=strelka germline small-variant calls ...
```

```
variants <- read.csv("RNA_seq_annotated_variants.vcf", skip=length(header),
header=TRUE, sep="\t")
```

Let's look at the variants dataframe.

```
colnames(variants)
```

```
##  [1] "X.CHROM"
##  [2] "POS"
##  [3] "ID"
##  [4] "REF"
##  [5] "ALT"
##  [6] "QUAL"
##  [7] "FILTER"
##  [8] "INFO"
##  [9] "FORMAT"
## [10] "ws20171223_MPs_tomatoMuscle8wkQuiescent201"
```

We see that this is consistent with the Variant Call Format standard. Let's extract the first row's REF and ALT alleles which is the reference allele base and the alternative allele:

```
variants[1, ]
```

```
##   X.CHROM      POS ID REF ALT QUAL                                FILTER
## 1       1 12746106  .   G   A    2 LowGQX;LowDepth;NoPassedVariantGTs
##
## 1 SNVHPOL=16;MQ=255;ANN=A|intron_variant|MODIFIER|Sulf1|ENSMUSG00000016918|transcript|ENSMUST0000008
##                                  FORMAT
## 1 GT:GQ:GQX:DP:DPF:AD:ADF:ADR:SB:FT:PL
##             ws20171223_MPs_tomatoMuscle8wkQuiescent201
## 1 0/1:21:1:2:3:1,1:1,1:0,0:0.0:LowGQX;LowDepth:33,0,18
```

```
ref_allele <- variants$REF[1]
alt_allele <- variants$ALT[1]
ref_allele
```

```
## [1] "G"
```

```
alt_allele
```

```
## [1] "A"
```

We see that the reference allele is G and the alternative allele called by Strelka is A.

5.2. We want to obtain the entirety of the ANN info value contents from the INFO field for the first variant.

```
# Get the INFO field of the first variant and convert to string format
info_field <- as.character(variants$INFO[1])

# Split by semicolon
info_split <- strsplit(info_field, ";")[[1]]

# Find the entry that starts with "ANN="
ann_entry <- info_split[grep("^ANN=", info_split)]

# Extract the value part after "ANN="
ann_value <- sub("^ANN=", "", ann_entry)
ann_value
```

```
## [1] "A|intron_variant|MODIFIER|Sulf1|ENSMUSG00000016918|transcript|ENSMUST00000088585.9|protein_codi
```

5.3. From the headers from the first part of the question, we have the following field for the format of ANN value content: ##INFO=<ID=ANN,Number=.,Type=String,Description="Functional annotations: 'Allele | Annotation | Annotation_Impact | Gene_Name | Gene_ID | Feature_Type | Feature_ID | Transcript_BioType | Rank | HGVS.c | HGVS.p | cDNA.pos / cDNA.length | CDS.pos / CDS.length | AA.pos / AA.length | Distance | ERRORS / WARNINGS / INFO' ">

The Annotation field is the second field, we can extract this:

```
# Create a vector with each ANN field
ann_split <- strsplit(ann_value, "\\|")[[1]]
annotation_type <- ann_split[2]
annotation_type
```

```
## [1] "intron_variant"
```

This tells us that the mutation occur in the intron region (non-coding) of a gene.

5.4. We can do the same process for variant 683

```
# Extract the INFO field of variant 683
info_field_683 <- as.character(variants$INFO[683])
info_split_683 <- strsplit(info_field_683, ";")[[1]]
ann_entry_683 <- info_split_683[grep("^ANN=", info_split_683)]
ann_value_683 <- sub("^ANN=", "", ann_entry_683)
```

Similar to 5.3, we will create a vector with each ANN field, the extract the gene name, which is the 4th field indicated in the header section.

```
# Create a vector with each ANN field
ann_first_683 <- strsplit(ann_value_683, ",")[[1]][1]
ann_split_683 <- strsplit(ann_first_683, "\\|")[[1]]

# Extract gene name
gene_683 <- ann_split_683[4]
gene_683
```

```
## [1] "Rps19"
```

5.5. We will construct a function to split the INFO fields at ";" then find the ANN tag. Then remove "ANN=" part which leaves us with only ANN fields of all variants. ann_first corresponds to the first snpEff annotation for each variant. The function will return the annotation type which is the second field of the ANN entry. Using sapply, we will repeat for all the ANN fields.

```
# Extract all ANN fields
info_all <- as.character(variants$INFO)

# Get all ANN entries
ann_all <- sapply(info_all, function(x) {
  ann_tag <- strsplit(x, ";")[[1]]
  ann_entry <- ann_tag[grep("^ANN=", ann_tag)]
  if (length(ann_entry) == 0) return(NA)
  ann_value <- sub("^ANN=", "", ann_entry)
  ann_first <- strsplit(ann_value, ",")[[1]][1]
  strsplit(ann_first, "\\|")[[1]][2]  # annotation type
})
```

We can see the variants type count in the table below:

```
# Count by variant type
table(ann_all)
```

```
## ann_all
##                                                    3_prime_UTR_variant
##                                                                    114
##                                                    5_prime_UTR_variant
```

```
##                                                                                 5
##                                                          downstream_gene_variant
##                                                                               113
## frameshift_variant&splice_acceptor_variant&splice_region_variant&intron_variant
##                                                                                 1
##                                                               intergenic_region
##                                                                              67
##                                                            intragenic_variant
##                                                                               2
##                                                               intron_variant
##                                                                             266
##                                                             missense_variant
##                                                                              71
##                                          missense_variant&splice_region_variant
##                                                                               3
##                                              non_coding_transcript_exon_variant
##                                                                              11
##                                          splice_acceptor_variant&intron_variant
##                                                                               1
##                                                           splice_region_variant
##                                                                               1
##                                            splice_region_variant&intron_variant
##                                                                               6
##                                         splice_region_variant&synonymous_variant
##                                                                               2
##                                               stop_gained&splice_region_variant
##                                                                               2
##                                                           synonymous_variant
##                                                                              31
##                                                          upstream_gene_variant
##                                                                             140
```

```r
sum(table(ann_all))
```

```
## [1] 836
```

5.6. A frameshift variant is an insertion or deletion (indel) whose length is not a multiple of 3. This shifts the triplet reading frame of the mRNA during translation and causes all downstream codons to be read incorrectly. Missense variants only change one amino acid, but a frameshift variant changes all downstream amino acids in translation, so it has a more negative effect.

5.7. We can use grepl() on the INFO field to find those containing "intron_variant". Then we will compare it with the total number of variants.

```r
intronic_variants <- sum(grepl("intron", variants$INFO))
intergenic_variants <- sum(grepl("intergenic", variants$INFO))
total_in <- sum(intronic_variants,intergenic_variants)

# Total number of variant calculation
n_total <- nrow(variants)
percentage_intronic <- (total_in / n_total) * 100
percentage_intronic
```

```
## [1] 83.37321
```

We see that around 83.37% of variants are intronic/intergenic which is consistent since most of the genomic DNA are non-coding.

5.8. In the INFO header, we can see that the 3rd field of ANN outline the impact of a variant. We will repeat what we did in 5.5 but this time the function will return the first snpEff annotation for each variant.

```
ann_firsts <- sapply(info_all, function(x) {
  ann_tag <- strsplit(x, ";")[[1]]
  ann_entry <- ann_tag[grep("^ANN=", ann_tag)]
  if (length(ann_entry) == 0) return(NA)
  ann_value <- sub("^ANN=", "", ann_entry)
  ann_first <- strsplit(ann_value, ",")[[1]][1]
})

# Get components
ann_split_all <- strsplit(ann_firsts, "\\|")
```

Let's create a data frame with only information we are interested in, which is Annotation, Impact and Gene, their position can be found in the INFO header.

```
# Create data frame containing the gene, type of variant and impact
ann_df <- data.frame(
  Annotation = sapply(ann_split_all, `[`, 2),
  Impact = sapply(ann_split_all, `[`, 3),
  Gene = sapply(ann_split_all, `[`, 4)
)
```

Coding variant will have specific keywords, we want only those variants. Then we can filter out the HIGH impact coding variant. Note that we also define splice variants as coding variants despite them being on an intron. This is because it affects how the exons is joined together which ultimately allows them to have the same impact as a coding variant.

```
coding_annots <- c(
  "missense",
  "stop",
  "start",
  "frameshift",
  "inframe",
  "splice",
  "coding",
  "synonymous"
)

high_coding <- ann_df[
  grepl("HIGH", ann_df$Impact) &
  grepl(paste(coding_annots, collapse="|"), ann_df$Annotation),
]

unique(high_coding$Gene)
```

```
## [1] "Ddx1"   "Rps14"  "Rps19"  "Hnrnpl"
```

```r
table(high_coding$Impact)
```

```
##
## HIGH
##    4
```

There are only 4 genes that have HIGH impact variant. Let's go one step further and see these protein's function, specifically those that are in the exon region:

- Rps14: Encodes ribosomal protein S14, a component of the 40S ribosomal subunit.
- Rps19: Encodes ribosomal protein S19, also part of the 40S subunit.
- Hnrnpl: Encodes heterogeneous nuclear ribonucleoprotein L, an RNA-binding protein that regulates pre-mRNA splicing and stability.

We see that a HIGH impact mutation on Rps19 can cause Diamond-Blackfan anemia, affecting erythropoiesis (red-blood-cell formation), so it makes sense for these kind of mutations to be so rare.

5.9. Insertions larger than the read length (60 bp) can't be properly aligned or reconstructed, so Strelka cannot detect them reliably.

5.10. According to section 5 of the VCF documentation, we are interested in the second last and the last column where we can see the order of the tags and their associated value. Let's first extract the tags column which will have the format similar toGT:AD:DP:GQ:PL

```r
tag_col <- variants [, ncol(variants)-1]
head(tag_col)
```

```
## [1] "GT:GQ:GQX:DP:DPF:AD:ADF:ADR:SB:FT:PL"
## [2] "GT:GQ:GQX:DP:DPF:AD:ADF:ADR:SB:FT:PL"
## [3] "GT:GQ:GQX:DP:DPF:AD:ADF:ADR:SB:FT:PL"
## [4] "GT:GQ:GQX:DP:DPF:AD:ADF:ADR:SB:FT:PL"
## [5] "GT:GQ:GQX:DPI:AD:ADF:ADR:FT:PL"
## [6] "GT:GQ:GQX:DP:DPF:AD:ADF:ADR:SB:FT:PL"
```

Let's also look at the last column

```r
# Extract genotype field for all variants
geno_field <- variants[, ncol(variants)]  # usually last column
head(geno_field)
```

```
## [1] "0/1:21:1:2:3:1,1:1,1:0,0:0.0:LowGQX;LowDepth:33,0,18"
## [2] "0/1:27:0:3:1:2,1:2,1:0,0:0.0:LowGQX:30,0,50"
## [3] "0/1:27:0:3:1:2,1:0,0:2,1:0.0:LowGQX:30,0,56"
## [4] "0/1:23:0:5:0:4,1:0,0:4,1:0.0:LowGQX:26,0,80"
## [5] "1/1:9:10:4:0,4:0,4:0,0:LowGQX:90,12,0"
## [6] "0/1:35:2:6:1:4,2:0,0:4,2:0.0:LowGQX:38,0,77"
```

We will first write a function to calculate VAF which takes in the tag column and the field column. It will find the AD (allele depth) values then extract the REF and ALT read counts. The VAF can then be calculated by taking:

$$\frac{x}{x+y}$$

where x is the ALT reads and y is the REF read.

```
library(ggplot2)

# Function to compute VAF
get_vaf <- function(entry_fmt, entry_sample) {

  # Split tag colum and fields
  fmt_fields  <- unlist(strsplit(entry_fmt, ":"))
  samp_fields <- unlist(strsplit(entry_sample, ":"))

  # Locate AD field by name
  ad_idx <- which(fmt_fields == "AD")
  if (length(ad_idx) == 0) return(NA)

  # Extract AD counts
  ad_str  <- samp_fields[ad_idx]
  counts  <- as.numeric(unlist(strsplit(ad_str, ",")))

  # Must have REF, ALT
  if (length(counts) != 2) return(NA)

  ref <- counts[1]
  alt <- counts[2]
  total <- ref + alt
  if (total == 0) return(NA)

  alt / total
}
```

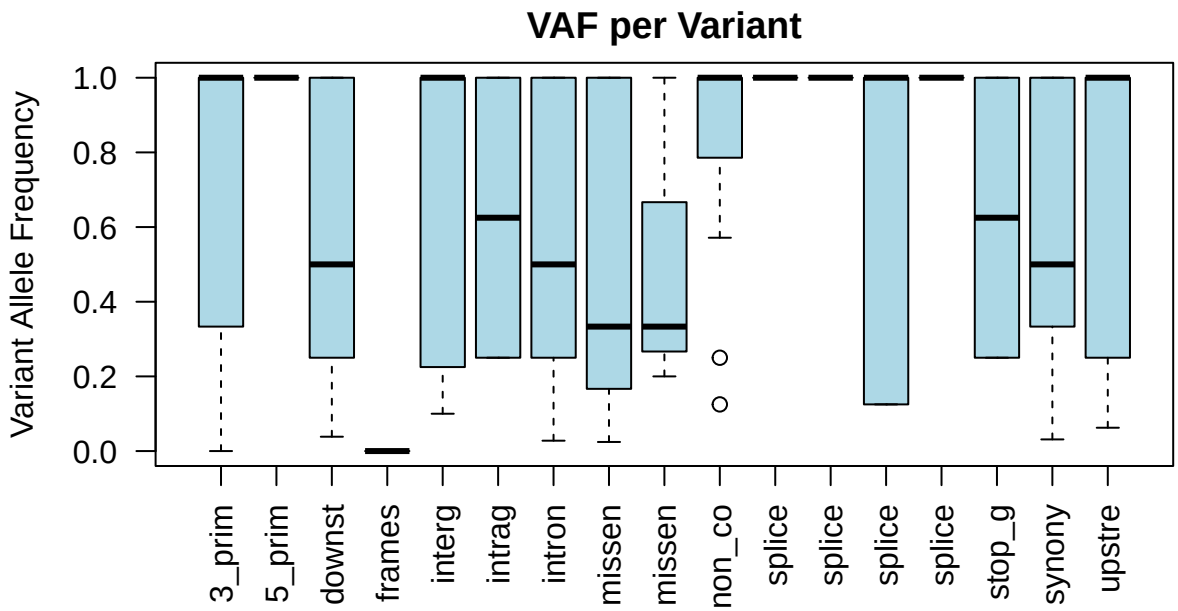
Next, we will apply the function get_vaf to every variants.

```
# Apply to all variants
tag_col <- variants[, ncol(variants)-1]    # FORMAT column
geno_field <- variants[, ncol(variants)]
vafs <- mapply(get_vaf, tag_col, geno_field)
```

Now we will use the list of annotation type that was produced earlier. We will group VAFs by annotation type then plot the distribution of VAFs across all variant type.

```
# Group VAFs by annotation type
vaf_list <- split(vafs, ann_all)
names(vaf_list) <- substr(names(vaf_list), 1, 6)

# Plot
par(mar = c(10, 4, 2, 2))
boxplot(vaf_list, las=2, col="lightblue",
        main="VAF per Variant", ylab="Variant Allele Frequency")
```



Next, we need to find the variants that have VAF greater than 5%.

```
# Count variants with VAF > 5%
vaf_above_5 <- sum(vafs > 0.05, na.rm = TRUE)
vaf_above_5
```

```
## [1] 823
```

Then, we will find the coding variants with VAF greater than 5%. We will first use ann_df which has the following structure:

```
colnames(ann_df)
```

```
## [1] "Annotation" "Impact"     "Gene"
```

We will add a column with the VAF values of each variant, then use coding_keywords to filter for coding variant, then taking only those with VAF > 0.05

```
# Combine VAF and annotation dataframe
ann_df$VAF <- vafs

coding_variants <- ann_df[
  grepl(paste(coding_annots, collapse="|"), ann_df$Annotation), ]

coding_vaf_above_5 <- sum(coding_variants$VAF > 0.05, na.rm = TRUE)
coding_vaf_above_5
```

```
## [1] 123
```

Contribution

All group members did the assignment individually before discussing results. These versions can be seen on github repo: https://github.com/hnimoaht/BMEG310-Assignment-3-ver. There are some discrepencies in the version, particularly in the boxplot sections and finding coding variants with VAF above 0.05. Sarah Dumont's boxplot were used in this final draft. We debugged 5.10 together as a group and reaches a common result. Minh Duong ensures consistency in the final draft and compiles code.