Challenge Documentation

Mobina Kazemi

September, 2023

```
library(ggplot2)
library(cowplot)
library(qdapRegex)
library(dplyr)
library(tidyverse)
library(knitr)
library(kableExtra)
library(httr)
library(jsonlite)
library(xml2)
library(papeR)
library(BiocManager)
```

The input data for this project is a VCF file which I need to read in the first step.

Read VCF file

A Variant Call Format (VCF) file is a standardized text-based format used to represent genetic variants, such as single nucleotide polymorphisms (SNPs), insertions, deletions, and other types of genomic variations. The components of a VCF file include:

1. Header: The header contains metadata and information about the source of the data, the reference genome being used, file format version, and other relevant information.
2. Columns and Data Rows: The main data section contains information about each variant as individual rows. Each row corresponds to a genomic position where a variant is found. The attributes of this section is listed:

CHROM: Chromosome where the variant is located

POS: Position of the variant on the chromosome

ID: Identifier for the variant

REF: Reference allele

ALT: Alternate alleles

QUAL: Quality score representing the confidence in the variant call

FILTER: Filters or flags indicating whether the variant passed quality control

INFO: Information fields containing additional data about the variant

FORMAT: Format field specifying the data format for each sample

Sample Data: Sample-specific data in the format specified by the “FORMAT” field

In order to find the Data Rows, I read the VCF file from the line where the first variant is located, this way I skip the Header information. For this purpose I search where "#CHROM" is located and read the variant information from that line.

```
file<-read.delim("~/Desktop/Tempus_exam/Challenge_data (1).vcf")
line=which(file$X..fileformat.VCFv4.1=="#CHROM")
vcf<-read.delim("~/Downloads/Challenge_data (1).vcf",skip = line)
```

Data preparation

INFO section of a VCF file contains the detailed information regarding each variant. It is a semicolon delimited list of information (AD=a;CP=b;…) which I need to parse out in order to have these detailed tags in separate columns for each variant.

To split the tags and their values in the INFO section, I used the following code. In order to understand the code, imagine the INFO string is AD=a;CP=b,NS=c;RO=d. I split the INFO string into pieces using ";" and "=" pattern. The output of this action is this string: "AD a CP b NS c RO d".

Now the odd indexes refer to attributes and even indexes refer to actual values. Using the same concept, I extracted the information in the INFO section of each variant and saved them in separate columns.

```
Data_final <- lapply(seq(nrow(vcf)), function(i) {
    INFO <- which(colnames(vcf) == "INFO")
    data_remove_info_column <- vcf[i, -INFO]

    length_attributes = length(unlist(strsplit(unlist(strsplit(vcf$INFO[i], split = ";")),
        split = "=")))
    # value of attributes
    Data <- t(as.data.frame(unlist(strsplit(unlist(strsplit(vcf$INFO[i], split = ";")),
        split = "="))[seq(2, length_attributes, 2)]))
    # name of the attributes
    colnames(Data) <- unlist(strsplit(unlist(strsplit(vcf$INFO[i], split = ";")),
        split = "="))[seq(1, length_attributes, 2)]

    rownames(Data) <- i

    Data_info_parsed <- cbind(data_remove_info_column, Data)
    return(Data_info_parsed)
})
Data_final <- do.call("rbind", Data_final)


knitr::kable(Data_final[1:4, c(1, 2, 4, 5, seq(11, 17))], caption = "Subset of Data_final",
    format = "pandoc")
```

Table 1: Subset of Data_final

X.CHROM	POS	REF	ALT	AB	ABP	AC	AF	AN	AO	CIGAR
1	931393	G	T	0	0	0	0	6	95	1X
1	935222	C	A	0.574956	58.3503	4	0.666667	6	652	1X
1	1277533	T	C	0	0	6	1	6	786	1X
1	1284490	G	A	0	0	6	1	6	228	1X

Q1: Depth of sequence coverage at the site of variation

Looking at the Header section of a VCF file, one can find the descriptions provided for the meaning of each attribute in the INFO section. Based on this information in the VCF file, DP tag shows the depth of coverage at the site of variation.

##INFO=<ID=DP,Number=1,Type=Integer,Description="Total read depth at the locus

```
# column DP in Data_final shows the depth of coverage at the site of variation
DP = which(colnames(Data_final) == "DP")
# Data_answers is the result of answering questions cumulatively
Data_answers <- Data_final[, c(1:10, DP)]
Data_answers$DP <- as.numeric(Data_answers$DP)
knitr::kable(Data_answers[1:4, c(1, 2, 4, 5, 11)], caption = "Subset of Data_answers",
    format = "pandoc")
```

Table 2: Subset of Data_answers

X.CHROM	POS	REF	ALT	DP
1	931393	G	T	4124
1	935222	C	A	1134
1	1277533	T	C	786
1	1284490	G	A	228

Q2: Number of reads supporting the variant

Based on the Header section of the VCF file, AO tag shows the alternate allele observation count.

##INFO=<ID=AO,Number=A,Type=Integer,Description="Alternate allele observations, with partial observations recorded fractionally">

```
# column AO in the VCF file shows the alternate allele observation count
AO = which(colnames(Data_final) == "AO")
# Data_answers is the result of answering questions cumulatively
Data_answers <- Data_final[, c(1:10, DP, AO)]
knitr::kable(Data_answers[1:4, c(1, 2, 4, 5, seq(11, 12))], caption = "Subset of Data_answers",
    format = "pandoc")
```

Table 3: Subset of Data_answers

X.CHROM	POS	REF	ALT	DP	AO
1	931393	G	T	4124	95
1	935222	C	A	1134	652
1	1277533	T	C	786	786
1	1284490	G	A	228	228

Q3: Percentage of reads supporting the variant versus those supporting reference reads.

Based on the Header section of the VCF file, RO tag shows the reference allele observation count.

##INFO=<ID=RO,Number=1,Type=Integer,Description="Reference allele observation count, with partial observations recorded fractionally">

For this question I need to divide AO by RO. For some cases, there are multiple alternative alleles reported and therefore multiple AO values exist in a row. For these cases, the AO attribute has comma separated components (a,b) which I need to divide each of them by RO. AO/RO value for cases with RO=0 is reported as Inf as we cannot divide any number by zero.

```
#column RO in the VCF file shows the reference allele observation count
RO=which(colnames(Data_final)=="RO")
#Data_answers is the result of answering questions cumulatively
Data_answers<- Data_final[,c(1:10,DP,AO,RO)]
Data_answers$RO<-as.numeric(Data_answers$RO)

AOvsRO<-lapply(seq(nrow(Data_answers)), function(i) {
  length_AO_value<-length(unlist(strsplit(Data_answers$AO[i],split = ",")))
  #For the cases of only one alternative allele
  if(length_AO_value==1){
    AOvsRO_value<-as.numeric(Data_answers$AO[i])/Data_answers$RO[i]
    #round values to 4 of significant digits
    AOvsRO_value<-round(AOvsRO_value,digits = 4)
    AOvsRO_value<-as.data.frame(AOvsRO_value)
    colnames(AOvsRO_value)<-"AO/RO"
    Data<-cbind(Data_answers[i,],AOvsRO_value)
  }
  #For the cases of multiple alternative alleles
  else{
    length_AO_value<-length(unlist(strsplit(Data_answers$AO[i],split = ",")))
    vec=c()
    for (j in seq(length_AO_value)) {
      AO_values<-unlist(strsplit(Data_answers$AO[i],split = ","))
      AOvsRO_value<-as.numeric(AO_values[j])/Data_answers$RO[i]
      #round values to 4 of significant digits
      AOvsRO_value<-round(AOvsRO_value,digits = 4)
      vec=c(vec,AOvsRO_value)
    }

    AOvsRO_value<-paste0(vec,collapse = ",")
    AOvsRO_value<-as.data.frame(AOvsRO_value)
    colnames(AOvsRO_value)<-"AO/RO"
    Data<-cbind(Data_answers[i,],AOvsRO_value)


  }

return(Data)
})
Data_answers <- do.call("rbind", AOvsRO)


knitr::kable(Data_answers[1:4,c(1,2,4,5,seq(11,14))],
             caption = "Subset of Data_answers",format = "pandoc")
```

Table 4: Subset of Data_answers

X.CHROM	POS	REF	ALT	DP	AO	RO	AO/RO
1	931393	G	T	4124	95	4029	0.0236
1	935222	C	A	1134	652	480	1.3583

X.CHROM	POS	REF	ALT	DP	AO	RO	AO/RO
1	1277533	T	C	786	786	0	Inf
1	1284490	G	A	228	228	0	Inf

Q5: The minor allele frequency of the variant if available

The format of this question is very similar to the previous questions and I prefered to answer this question before question 4.

Minor Allele Frequency (MAF) is a concept used in genetics and genomics to describe the frequency of the less common allele at a specific genetic variant locus within a population. MAF is calculated based on the percentage of allelic depth (AD) vs sequence depth (DP).

$$MAF = \min(AD)/DP$$

Here, AD is not present among the attributes of the INFO section. However, AO and RO together are the representation of AD and I generated this column myself in order to answer this question.

Note: It is possible for the reference allele to be the minor allele at a genetic variant locus. The reference allele is simply the allele that is used as the basis for comparison in the context of the reference genome being used. The choice of reference allele does not necessarily determine whether an allele is major or minor.

```
Data_answers <- Data_answers %>%
    mutate(AD = paste(AO, RO, sep = ","))

Data_answers$DP <- as.numeric(Data_answers$DP)
# for each variant we get the min value of AD and divide that to DP to
# calculate MAF
MAF <- lapply(seq(nrow(Data_answers)), function(i) {

    min_AD <- min(as.numeric(unlist(strsplit(Data_answers$AD[i], split = ","))))
    MAF_value <- min_AD/Data_answers$DP[i]
    MAF_value <- round(MAF_value, digits = 4)
    MAF_value <- as.data.frame(MAF_value)
    colnames(MAF_value) <- "MAF"
    Data <- cbind(Data_answers[i, ], MAF_value)

    return(Data)
})
Data_answers <- do.call("rbind", MAF)


knitr::kable(Data_answers[1:4, c(1, 2, 4, 5, seq(11, 16))], caption = "Subset of Data_answers",
    format = "pandoc")
```

Table 5: Subset of Data_answers

X.CHROM	POS	REF	ALT	DP	AO	RO	AO/RO	AD	MAF
1	931393	G	T	4124	95	4029	0.0236	95,4029	0.0230
1	935222	C	A	1134	652	480	1.3583	652,480	0.4233
1	1277533	T	C	786	786	0	Inf	786,0	0.0000
1	1284490	G	A	228	228	0	Inf	228,0	0.0000

Q4: Using the VEP hgvs API

This was the first time that I had to use Ensembl REST API endpoints. My previous experience with Variant Effect Predictor (VEP) was through Web interface and Command line tool. Therefore, after some research I could write the code to fetch variant consequences for cases with single nucleotide variation when the reference allele and the alternative one are both one nucleotide. However, I had some trouble incorporating such method for other scenarios (refer to following Note). For that reason I split the data into Solved_Q4 and Unsolved_Q4.

Note: I was able to solve the problem for other cases as well (Unsolved_Q4). You would find the update at the end of the answer for this question (refer to Update Q4 section in page 9)

```
# Counting the length of the variant string (ATT=3)
Data_answers <- Data_answers %>%
    mutate(Len_Ref = nchar(REF)) %>%
    mutate(Len_Alt = nchar(ALT))

# Solved_Q4
condition_Solved_Q4 <- Data_answers$Len_Ref == 1 & Data_answers$Len_Alt == 1
#-c(17,18) is to remove those columns after diving the data
Data_Solved_Q4 <- Data_answers[which(condition_Solved_Q4), -c(17, 18)]

# Unsolved_Q4
condition_Unsolved_Q4 <- Data_answers$Len_Ref != 1 | Data_answers$Len_Alt != 1
Data_Unsolved_Q4 <- Data_answers[which(condition_Unsolved_Q4), -c(17, 18)]
```

Note: Based on the REST API documentation, in order to connect to the correct database, you need to specify the server correctly. One of my challenges was finding the correct server in the first step. In the header section of a VCF file, there is typically information that specifies the reference genome that was used for variant calling. Based on my understanding of the data (##reference=/data/human_g1k_v37.fasta), reference genome "human_g1k_v37.fasta" is likely based on the GRCh37 assembly (Genome Reference Consortium Human Build 37) and I used the server accordingly (https://grch37.rest.ensembl.org).

Next step is to specify the endpoint and the information about the request. Since here we are focused on variant effect prediction for human, in the URL, "/vep/homo_sapiens/hgvs/" is fixed and we just need to generate the dynamic segment of the URL based on each variant. Since here, I have the position and the information about the reference and alternative allele for each variant, I can implement my request based on HGVS notation as shown in the following example:

```
#vep/homo_sapiens/hgvs/12:g.133233705T>C?content-type=application/json

server<-"https://grch37.rest.ensembl.org"
ext<-"/vep/homo_sapiens/hgvs/12:g.133233705T>C?"
r <- GET(paste(server, ext, sep = ""), content_type("application/json"),

         accept("application/json"))

stop_for_status(r)


temp <- fromJSON(toJSON(content(r)))
example<-matrix(data = 0,nrow = 1,ncol = 6)
example<-as.data.frame(example)
colnames(example)<-c("CHROM","POS","REF","ALT","gene_symbol","consequence")
example[1,]<-c(12,133233705,"T","C",temp$transcript_consequences[[1]]$gene_symbol[[1]],
               temp$transcript_consequences[[1]]$consequence_terms[[1]][1])

example
```

```
##   CHROM       POS REF ALT gene_symbol    consequence
## 1    12 133233705   T   C        POLE intron_variant
```

Same as the above example, I generated the results for 5333 (number of variants in the Solved_Q4 data) out of 6977 variant of the question.

Note: For the cases that the gene symbol was not returned using the API, I reported gene symbol as NA and just reported the consequence.

```
knitr::opts_chunk$set(eval = FALSE)
library(httr)
library(jsonlite)
library(xml2)
library(papeR)

GeneSymbol_Consequence_VEP_API<-function(Chrom,Pos,Ref,Alt){
server<-"https://grch37.rest.ensembl.org"
ext<-"/vep/homo_sapiens/hgvs/"
#generating the dynamic segment of the URL based on the Pos, Ref, Alt
varinat=paste0(Chrom,":g.",Pos,Ref,">",Alt,"?")
Url=paste0(ext,varinat)
#connection to database
r <- GET(paste(server, Url, sep = ""), content_type("application/json"),

         accept("application/json"))
#evaluating my request
stop_for_status(r)

#save the returned values in a variable
temp <- fromJSON(toJSON(content(r)))
#extract gene symbol
gene_symbol=temp$transcript_consequences[[1]]$gene_symbol[[1]]
#extract consequence
consequence=temp$transcript_consequences[[1]]$consequence_terms[[1]][1]
#For cases that the gene symbol is not returned
if(is.null(gene_symbol)==F){
result=c(gene_symbol,consequence)
}else{
consequence=temp$most_severe_consequence[[1]]
result=c(NA,consequence)
}
result<-t(as.data.frame(result))
colnames(result)<-c("gene_symbol","consequence")
return(result)
}

#Check the function for 100 of variants.
# VEP_API_data<-lapply(seq(100), function(i){
#   Chrom=Data_Solved_Q4$X.CHROM[i]
#   Pos=Data_Solved_Q4$POS[i]
#   Ref=Data_Solved_Q4$REF[i]
#   Alt=Data_Solved_Q4$ALT[i]
#   GeneSymbol_Consequence=GeneSymbol_Consequence_VEP_API(Chrom,Pos,Ref,Alt)
#   Data<-cbind(Data_Solved_Q4[i,],GeneSymbol_Consequence)
# return(Data)
# })
# a<-do.call("rbind", VEP_API_data)
```

```r
# head(a)


#Run the fucntion for all the variants in the Solved_Q4 data
VEP_API_data<-lapply(seq(1,nrow(Data_Solved_Q4)), function(i){
  #extract the parameters of the GeneSymbol_Consequence_VEP_API function
  Chrom=Data_Solved_Q4$X.CHROM[i]
  Pos=Data_Solved_Q4$POS[i]
  Ref=Data_Solved_Q4$REF[i]
  Alt=Data_Solved_Q4$ALT[i]

  #run the GeneSymbol_Consequence_VEP_API function for each variant
  GeneSymbol_Consequence=GeneSymbol_Consequence_VEP_API(Chrom,Pos,Ref,Alt)
  #save the result for each variant besides other information about a variant
  Data<-cbind(Data_Solved_Q4[i,],GeneSymbol_Consequence)
  return(Data)
})

Data_Solved_Q4 <- do.call("rbind", VEP_API_data)

#Save the result
write.table(Data_Solved_Q4,
            file = "~/Desktop/Tempus_exam/Data_Solved_Q4.txt",quote = F,row.names = F)
```

Now that I have the requested results for Solved_Q4, I just put NA for gene symbol and consequence of the Unsolved_Q4 data and then combine the Solved_Q4 and Unsolved_Q4 together.

```r
# Read the saved result for Solved_Q4
Data_Solved_Q4 <- read.delim(file = "~/Desktop/Tempus_exam/Data_Solved_Q4.txt", sep = "")
colnames(Data_Solved_Q4)[14] <- "AO/RO"

# Put NA for the GeneSymbol and Consequence columns of Unsolved_Q4
Data_Unsolved_Q4$gene_symbol <- NA
Data_Unsolved_Q4$consequence <- NA

# rbind the Data_Solved_Q4 and Data_Unsolved_Q4
Data_answers <- rbind(Data_Solved_Q4, Data_Unsolved_Q4)

knitr::kable(Data_answers[1:4, c(1, 2, 4, 5, seq(17, 18))], caption = "Subset of Data_answers",
    format = "pandoc")
```

Table 6: Subset of Data_answers

X.CHROM	POS	REF	ALT	gene_symbol	consequence
1	931393	G	T	HES4	downstream_gene_variant
1	935222	C	A	HES4	intron_variant
1	1277533	T	C	DVL1	synonymous_variant
1	1284490	G	A	MXRA8	downstream_gene_variant

For the last requirement of this question, I need to report the type of variation (substitution,insertion, CNV, etc.) for each variant. Based on the Header section of the VCF file, TYPE tag shows the type of variation.

##INFO=<ID=TYPE,Number=A,Type=String,Description="The type of allele, either snp, mnp, ins, del, or complex.">

```
#add type of each variant
TYPE=which(colnames(Data_final)=="TYPE")
Data_answers<-merge(Data_answers,Data_final[,c(seq(5),TYPE)],

                    by = colnames(Data_final)[seq(5)])
```

Update Q4

To complete the remaining part of question 4 (Unsolved_Q4), I found the way to request from REST API for del, ins, mnp, snp and complex. Here I add the gene symbol and consequence for those variants as well. Since for each type of variation the connection to REST API needs different parameters, I split the data into 5 categories; del, ins, complex, mnp and snp and did the analysis separately for each of them.

```
Data_Unsolved_Q4 <- Data_answers[which(is.na(Data_answers$consequence) == T), ]
Data_Solved_Q4 <- Data_answers[-(which(is.na(Data_answers$consequence) == T)), ]
# Here I split the data into 5 categories: del, ins, complex, mnp, snp

Data_del <- Data_Unsolved_Q4[grep(pattern = "del", x = Data_Unsolved_Q4$TYPE), ]
Data_remain <- Data_Unsolved_Q4[-(grep(pattern = "del", x = Data_Unsolved_Q4$TYPE)),
    ]

Data_ins <- Data_remain[grep(pattern = "ins", x = Data_remain$TYPE), ]
Data_remain <- Data_remain[-(grep(pattern = "ins", x = Data_remain$TYPE)), ]

Data_complex <- Data_remain[grep(pattern = "complex", x = Data_remain$TYPE), ]
Data_remain <- Data_remain[-(grep(pattern = "complex", x = Data_remain$TYPE)), ]

Data_mnp <- Data_remain[grep(pattern = "mnp", x = Data_remain$TYPE), ]
Data_snp <- Data_remain[-(grep(pattern = "mnp", x = Data_remain$TYPE)), ]
```

1. Data_del

```
knitr::opts_chunk$set(eval = FALSE)
#For each category the connection to REST API needs different parameters

##Data_del

VEP_del<-lapply(seq(1,nrow(Data_del)), function(i){
  #extract the parameters of the GeneSymbol_Consequence_VEP_API function
  Chrom=Data_del$X.CHROM[i]
  Pos=Data_del$POS[i]

  server<-"https://grch37.rest.ensembl.org"
  ext<-"/vep/homo_sapiens/hgvs/"
  #generating the dynamic segment of the URL
  varinat=paste0(Chrom,":g.",Pos,"del?")
  Url=paste0(ext,varinat)

  r <- GET(paste(server, Url, sep = ""), content_type("application/json"),

         accept("application/json"))
  #evaluating my request
  stop_for_status(r)
  #save the returned values in a variable
```

```
  temp <- fromJSON(toJSON(content(r)))
  #extract gene symbol
  gene_symbol=temp$transcript_consequences[[1]]$gene_symbol[[1]]
  #extract consequence
  consequence=temp$transcript_consequences[[1]]$consequence_terms[[1]][1]
  if(is.null(gene_symbol)==F){
        Data_del$gene_symbol[i]<-gene_symbol
        Data_del$consequence[i]<-consequence
  }else{
        Data_del$consequence[i]=temp$most_severe_consequence[[1]]
        }

  return(Data_del[i,])
})
Data_del <- do.call("rbind", VEP_del)
```

2. Data_ins

```
knitr::opts_chunk$set(eval = FALSE)
#For each category the connection to REST API needs different parameters

##Data_ins
VEP_ins<-lapply(seq(1,nrow(Data_ins)), function(i){
  #extract the parameters of the GeneSymbol_Consequence_VEP_API function
  #hgvs/12:g.122762763_122762764insAAA?content-type=application/json
  Chrom=Data_ins$X.CHROM[i]
  Pos=Data_ins$POS[i]
  Ref=Data_ins$REF[i]
  Alt=Data_ins$ALT[i]
  TYPE=Data_ins$TYPE[i]
  #find which of the alternative variants is reported as insertion
  alt_n<-which(unlist(strsplit(TYPE,","))=="ins")[1]
  alt_str<-unlist(strsplit(Alt,","))[alt_n]

  #Find the insertion part of sequence
  alignment=pairwiseAlignment(Ref,alt_str)
  ins_pos=which(unlist(str_split(as.character(alignment@pattern),""))=="-")
  if(length(ins_pos)!=0){
    ins_len=length(ins_pos)
    ins_str=substring(alt_str,ins_pos[1],ins_pos[ins_len])
  }else{
    ins_str=substring(alt_str,(nchar(as.character(alignment@pattern))+1),nchar(alt_str))
  }

  server<-"https://grch37.rest.ensembl.org"
  ext<-"/vep/homo_sapiens/hgvs/"
  #generating the dynamic segment of the URL
  varinat=paste0(Chrom,":g.",Pos,"_",(Pos+1),"ins",ins_str,"?")
  Url=paste0(ext,varinat)

  r <- GET(paste(server, Url, sep = ""), content_type("application/json"),

         accept("application/json"))
  #evaluating my request
  stop_for_status(r)
  #save the returned values in a variable
```

```
  temp <- fromJSON(toJSON(content(r)))
  #extract gene symbol
  gene_symbol=temp$transcript_consequences[[1]]$gene_symbol[[1]]
  #extract consequence
  consequence=temp$transcript_consequences[[1]]$consequence_terms[[1]][1]
  if(is.null(gene_symbol)==F){
        Data_ins$gene_symbol[i]<-gene_symbol
        Data_ins$consequence[i]<-consequence
  }else{
        Data_ins$consequence[i]=temp$most_severe_consequence[[1]]
        }

  return(Data_ins[i,])
})
Data_ins <- do.call("rbind", VEP_ins)
```

3. Data_complex

```
knitr::opts_chunk$set(eval = FALSE)
#For each category the connection to REST API needs different parameters

##Data_complex
VEP_complex<-lapply(seq(1,nrow(Data_complex)), function(i){
  #extract the parameters of the GeneSymbol_Consequence_VEP_API function
  #hgvs/1:g.100843090_100843093CTTG>TTTT?content-type=application/json
  Chrom=Data_complex$X.CHROM[i]
  Pos=Data_complex$POS[i]
  Ref=Data_complex$REF[i]
  Alt=Data_complex$ALT[i]
  TYPE=Data_complex$TYPE[i]
  #find which of the alternative variants is reported as insertion
  alt_n<-which(unlist(strsplit(TYPE,","))=="complex")[1]
  alt_str<-unlist(strsplit(Alt,","))[alt_n]


  server<-"https://grch37.rest.ensembl.org"
  ext<-"/vep/homo_sapiens/hgvs/"
  #generating the dynamic segment of the URL
  varinat=paste0(Chrom,":g.",Pos,"_",(Pos+nchar(Ref)-1),Ref,">",alt_str,"?")
  Url=paste0(ext,varinat)

  r <- GET(paste(server, Url, sep = ""), content_type("application/json"),

         accept("application/json"))
  #evaluating my request
  stop_for_status(r)
  #save the returned values in a variable
  temp <- fromJSON(toJSON(content(r)))
  #extract gene symbol
  gene_symbol=temp$transcript_consequences[[1]]$gene_symbol[[1]]
  #extract consequence
  consequence=temp$transcript_consequences[[1]]$consequence_terms[[1]][1]
  if(is.null(gene_symbol)==F){
        Data_complex$gene_symbol[i]<-gene_symbol
        Data_complex$consequence[i]<-consequence
  }else{
```

```
        Data_complex$consequence[i]=temp$most_severe_consequence[[1]]
        }

  return(Data_complex[i,])
})
Data_complex <- do.call("rbind", VEP_complex)
```

4. Data_mnp

```
knitr::opts_chunk$set(eval = FALSE)
#For each category the connection to REST API needs different parameters

##Data_mnp
VEP_mnp<-lapply(seq(1,nrow(Data_mnp)), function(i){
  #extract the parameters of the GeneSymbol_Consequence_VEP_API function
  #hgvs/11:g.125503051_125503052GG>TT?content-type=application/json
  Chrom=Data_mnp$X.CHROM[i]
  Pos=Data_mnp$POS[i]
  Ref=Data_mnp$REF[i]
  Alt=Data_mnp$ALT[i]
  TYPE=Data_mnp$TYPE[i]
  #find which of the alternative variants is reported as insertion
  alt_n<-which(unlist(strsplit(TYPE,","))=="mnp")[1]
  alt_str<-unlist(strsplit(Alt,","))[alt_n]


  server<-"https://grch37.rest.ensembl.org"
  ext<-"/vep/homo_sapiens/hgvs/"
  #generating the dynamic segment of the URL
  varinat=paste0(Chrom,":g.",Pos,"_",(Pos+nchar(Ref)-1),Ref,">",alt_str,"?")
  Url=paste0(ext,varinat)

  r <- GET(paste(server, Url, sep = ""), content_type("application/json"),

         accept("application/json"))
  #evaluating my request
  stop_for_status(r)
  #save the returned values in a variable
  temp <- fromJSON(toJSON(content(r)))
  #extract gene symbol
  gene_symbol=temp$transcript_consequences[[1]]$gene_symbol[[1]]
  #extract consequence
  consequence=temp$transcript_consequences[[1]]$consequence_terms[[1]][1]
  if(is.null(gene_symbol)==F){
        Data_mnp$gene_symbol[i]<-gene_symbol
        Data_mnp$consequence[i]<-consequence
  }else{
        Data_mnp$consequence[i]=temp$most_severe_consequence[[1]]
        }

  return(Data_mnp[i,])
})
Data_mnp <- do.call("rbind", VEP_mnp)
```

5. Data_snp

```
knitr::opts_chunk$set(eval = FALSE)
#For each category the connection to REST API needs different parameters

##Data_snp
VEP_snp<-lapply(seq(1,nrow(Data_snp)), function(i){
  #extract the parameters of the GeneSymbol_Consequence_VEP_API function
  #hgvs/11:g.125503051_125503052GG>TT?content-type=application/json
  Chrom=Data_snp$X.CHROM[i]
  Pos=Data_snp$POS[i]
  Ref=Data_snp$REF[i]
  Alt=Data_snp$ALT[i]
  TYPE=Data_snp$TYPE[i]
  #find which of the alternative variants is reported as insertion
  alt_n<-which(unlist(strsplit(TYPE,","))=="snp")[1]
  alt_str<-unlist(strsplit(Alt,","))[alt_n]


  server<-"https://grch37.rest.ensembl.org"
  ext<-"/vep/homo_sapiens/hgvs/"
  #generating the dynamic segment of the URL
  varinat=paste0(Chrom,":g.",Pos,"_",(Pos+nchar(Ref)-1),Ref,">",alt_str,"?")
  Url=paste0(ext,varinat)

  r <- GET(paste(server, Url, sep = ""), content_type("application/json"),

         accept("application/json"))
  #evaluating my request
  stop_for_status(r)
  #save the returned values in a variable
  temp <- fromJSON(toJSON(content(r)))
  #extract gene symbol
  gene_symbol=temp$transcript_consequences[[1]]$gene_symbol[[1]]
  #extract consequence
  consequence=temp$transcript_consequences[[1]]$consequence_terms[[1]][1]
  if(is.null(gene_symbol)==F){
        Data_snp$gene_symbol[i]<-gene_symbol
        Data_snp$consequence[i]<-consequence
  }else{
        Data_snp$consequence[i]=temp$most_severe_consequence[[1]]
        }

  return(Data_snp[i,])
})
Data_snp <- do.call("rbind", VEP_snp)
```

After getting the results for each category, I combined all the results with the results that I had from previous analysis (Solved_Q4) and saved the final data in Data_answers.csv.

```
# Combine the 5 categories together with the Solved_Q4
Data_answers <- rbind(Data_Solved_Q4, Data_del, Data_ins, Data_complex, Data_mnp,
    Data_snp)


# Arrange in a way that NAs move to the end of data
Data_answers <- Data_answers %>%
    arrange(rowSums(is.na(.)))
```

```r
# Save the result
write.table(Data_answers, file = "~/Desktop/Tempus_exam/Data_answers.txt")

write.table(Data_answers, file = "~/Desktop/Tempus_exam/Data_answers.csv", quote = F,
    row.names = F, sep = ",")
```

```r
Data_answers<-read.delim("~/Desktop/Tempus_exam/Data_answers.txt",sep = "")
sample_row=sample(x = seq(nrow(Data_answers)),size = 10)

knitr::kable(Data_answers[sample_row,c(1,4,5,seq(17,19))],
             caption = "Subset of Data_answers",format = "pandoc")
```

Table 7: Subset of Data_answers

	X.CHROM	REF	ALT	gene_symbol	consequence	TYPE
5118	5	C	T	FAT2	synonymous_variant	snp
4007	20	G	A	CTCFL	intron_variant	snp
2587	16	T	C	FANCA	missense_variant	snp
3850	2	T	G	MOB1A	5_prime_UTR_variant	snp
6018	7	C	T	HOXA3	synonymous_variant	snp
64981	5	AATA	AATATA	DIAPH1	intron_variant	ins
321	1	T	C	CDK11A	synonymous_variant	snp
110	1	T	C	RP11-325P15.1	non_coding_transcript_exon_variant	snp
6602	9	T	C	NOTCH1	splice_polypyrimidine_tract_variant	snp
54	1	A	G	NGF	intron_variant	snp

Q6: Any additional annotations that you feel might be relevant

In the Data preparation section, I parsed out all the information in the INFO parameter, therefore one can gain additional information about the variants by looking at each attribute of the INFO section that have not been covered in the previous questions. I summarize these detailed tags and their description here:

AB: Allele balance at heterozygous sites: a number between 0 and 1 representing the ratio of reads showing the reference allele to all reads, considering only reads from individuals called as heterozygous

ABP: Allele balance probability at heterozygous sites: Phred-scaled upper-bounds estimate of the probability of observing the deviation between ABR and ABA given E(ABR/ABA) ~ 0.5, derived using Hoeffding's inequality

AC: Total number of alternate alleles in called genotypes

AF: Estimated allele frequency in the range (0,1]

AN: Total number of alleles in called genotypes

CIGAR: The extended CIGAR representation of each alternate allele, with the exception that '=' is replaced by 'M' to ease VCF parsing

DPB: Total read depth per bp at the locus; bases in reads overlapping / bases in haplotype

DPRA: Alternate allele depth ratio. Ratio between depth in samples with each called alternate allele and those without

EPP: End Placement Probability: Phred-scaled upper-bounds estimate of the probability of observing the deviation between EL and ER given E(EL/ER) ~ 0.5, derived using Hoeffding's inequality

EPPR: End Placement Probability for reference observations: Phred-scaled upper-bounds estimate of the probability of observing the deviation between EL and ER given E(EL/ER) ~ 0.5, derived using Hoeffding's inequality

GTI: Number of genotyping iterations required to reach convergence or bailout

LEN: allele length

MEANALT: Mean number of unique non-reference allele observations per sample with the corresponding alternate alleles

MQM: Mean mapping quality of observed alternate alleles

MQMR: Mean mapping quality of observed reference alleles

NS: Number of samples with data

NUMALT: Number of unique non-reference alleles in called genotypes at this position

ODDS: The log odds ratio of the best genotype combination to the second-best

PAIRED: Proportion of observed alternate alleles which are supported by properly paired read fragments

PAIREDR: Proportion of observed reference alleles which are supported by properly paired read fragments

PAO: Alternate allele observations, with partial observations recorded fractionally

PQA: Alternate allele quality sum in phred for partial observations

PQR: Reference allele quality sum in phred for partial observations

PRO: Reference allele observation count, with partial observations recorded fractionally

QA: Sum of quality of the alternate observations

QR: Sum of quality of the reference observations

RPL: Reads Placed Left: number of reads supporting the alternate balanced to the left (5') of the alternate allele

RPP: Read Placement Probability: Phred-scaled upper-bounds estimate of the probability of observing the deviation between RPL and RPR given E(RPL/RPR) ~ 0.5, derived using Hoeffding's inequality

RPPR: Read Placement Probability for reference observations: Phred-scaled upper-bounds estimate of the probability of observing the deviation between RPL and RPR given E(RPL/RPR) ~ 0.5, derived using Hoeffding's inequality

RPR: Reads Placed Right: number of reads supporting the alternate balanced to the right (3') of the alternate allele

RUN: Run length: the number of consecutive repeats of the alternate allele in the reference genome

SAF: Number of alternate observations on the forward strand

SAP: Strand balance probability for the alternate allele: Phred-scaled upper-bounds estimate of the probability of observing the deviation between SAF and SAR given E(SAF/SAR) ~ 0.5, derived using Hoeffding's inequality

SAR: Number of alternate observations on the reverse strand

SRF: Number of reference observations on the forward strand

SRP: Strand balance probability for the reference allele: Phred-scaled upper-bounds estimate of the probability of observing the deviation between SRF and SRR given E(SRF/SRR) ~ 0.5, derived using Hoeffding's inequality

SRR: Number of reference observations on the reverse strand

In addition to the information above, we can focus on the data for each sample and similar to the INFO section, parse out the data for each sample separately. Looking at the NS value in the VCF file, we have 2 samples in this study. Based on the format of the VCF file, it seems that the analysis was conducted on normal sample and some sort of disease sample. I have access to the normal attribute from the VCF file which has the specified information about the normal sample. Looking at the VCF file, I do not think there is any information specifically about the disease sample. One can think the vaf5 attribute is the disease sample data. However, from my understanding in 5195 out of 6977 rows of the data, the value of normal attribute is exactly equal to the vaf5 attribute. This information gives me the idea that the vaf5 attribute is not related to disease sample and could potentially refer to sample data assuming a threshold (for example 5%) on the "Variant Allele Frequency" (VAF) of the sample.

Based on the information above, I parsed out the detailed tags for the normal sample and named them tag_normal.

```
Data_final_list <- lapply(seq(nrow(Data_final)), function(i) {
    normal <- which(colnames(Data_final) == "normal")
    FORMAT <- which(colnames(Data_final) == "FORMAT")
    data_remove_normal_column <- Data_final[i, -c(normal, FORMAT)]

    Data <- t(as.data.frame(unlist(strsplit(Data_final$normal[i], split = ":"))))
    colnames(Data) <- paste0(unlist(strsplit(Data_final$FORMAT[i], split = ":")),
        "_normal")

    rownames(Data) <- i

    Data_normal_parsed <- cbind(data_remove_normal_column, Data)
    return(Data_normal_parsed)
})
Data_final <- do.call("rbind", Data_final_list)

cols <- sample(x = seq(ncol(Data_final)), 5)
knitr::kable(Data_final[1:4, cols], caption = "Subset of Data_final", format = "pandoc")
```

Table 8: Subset of Data_final

GTI	POS	QA_normal	EPP	ODDS
0	931393	0	9.61615	591.29
0	935222	12246	44.7878	53.367
0	1277533	15766	94.5216	307.075
0	1284490	4551	463.972	92.2152

Here I parsed out all the additional attributes regarding the variants and saved the final data in Data_more_attr.csv.

```
#Other attributes that have not been covered in the previous questions
diff_attr<-setdiff(colnames(Data_final),colnames(Data_answers))
FORMAT=which(colnames(Data_answers)=="FORMAT")
normal=which(colnames(Data_answers)=="normal")

#merge with Data_final and add the additional attributes for each variant

Data_more_attr<-merge(Data_answers[,-c(FORMAT,normal)],

                      Data_final[,c(colnames(Data_final)[seq(5)],diff_attr)],
                      by=colnames(Data_final)[seq(5)])

Data_more_attr<-Data_more_attr %>%
  arrange(rowSums(is.na(.)))


write.table(Data_more_attr,
            file = "~/Desktop/Tempus_exam/Data_more_attr.txt")

write.table(Data_more_attr,
            file = "~/Desktop/Tempus_exam/Data_more_attr.csv",quote = F,row.names = F,sep = ",")

cols<-sample(x = seq(ncol(Data_more_attr)),5)

knitr::kable(Data_more_attr[1:4,cols]
             ,caption = "Subset of Data_more_attr",format = "pandoc")
```

Table 9: Subset of Data_more_attr

MQMR	RPL	ID	RUN	POS
0	872	.	1	100818464
70	72	.	1	100819424
70	2	.	1	100843084
70	6	.	1	100843090

Based on the question of the project and the depth of analysis, one can use any of these additional attributes to answer scientific questions. There is also possibility of comparison between attributes and finding patterns between them for more advanced analyses. For the sake of this project, I just extracted all the possible information from the VCF file.