Variant Analysis

Quality Control

First important thing during an analysis is: check the quality of the raw data. We do this by using Biopython, and plotting the Phred quality of a sample of 5,000 reads from the original file.

```
from Bio import SeqIO
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as pyplot
import numpy as np
import pandas as pd

def seq_quality(reads):
  data = SeqIO.parse( reads, 'fastq')
  pList = []
  phredList =[]
  lengths = []
  for record in data:
    pArray = []
    phredArray = []
    qualities = record.letter_annotations["phred_quality"]
    for Q in qualities:
        # convert the PHRED score to a probability
        p = 10**(-float(Q)/10)
        # append this specific probabity to array for this sequence
        pArray = pArray + [p]
        phredArray += [Q]
    # now append the sequence's probablity to a list of all of them
    pList = pList + [pArray]
    # also append the phred qualities as they are
    phredList += [phredArray]
    # store the length of the probability array (same as length of sequence)
    lengths += [len(pArray)]
  phredData = pd.DataFrame(phredList)
  return(phredData)

forwardQuality = seq_quality('/home/rstudio/data/markdown_exercise/reads/normal_sample_1.fq')
reverseQuality = seq_quality('/home/rstudio/data/markdown_exercise/reads/normal_sample_2.fq')
```

Once we do this in python, we can then manage the data we have extracted from within R. The *py* object contains the objects generated within python.

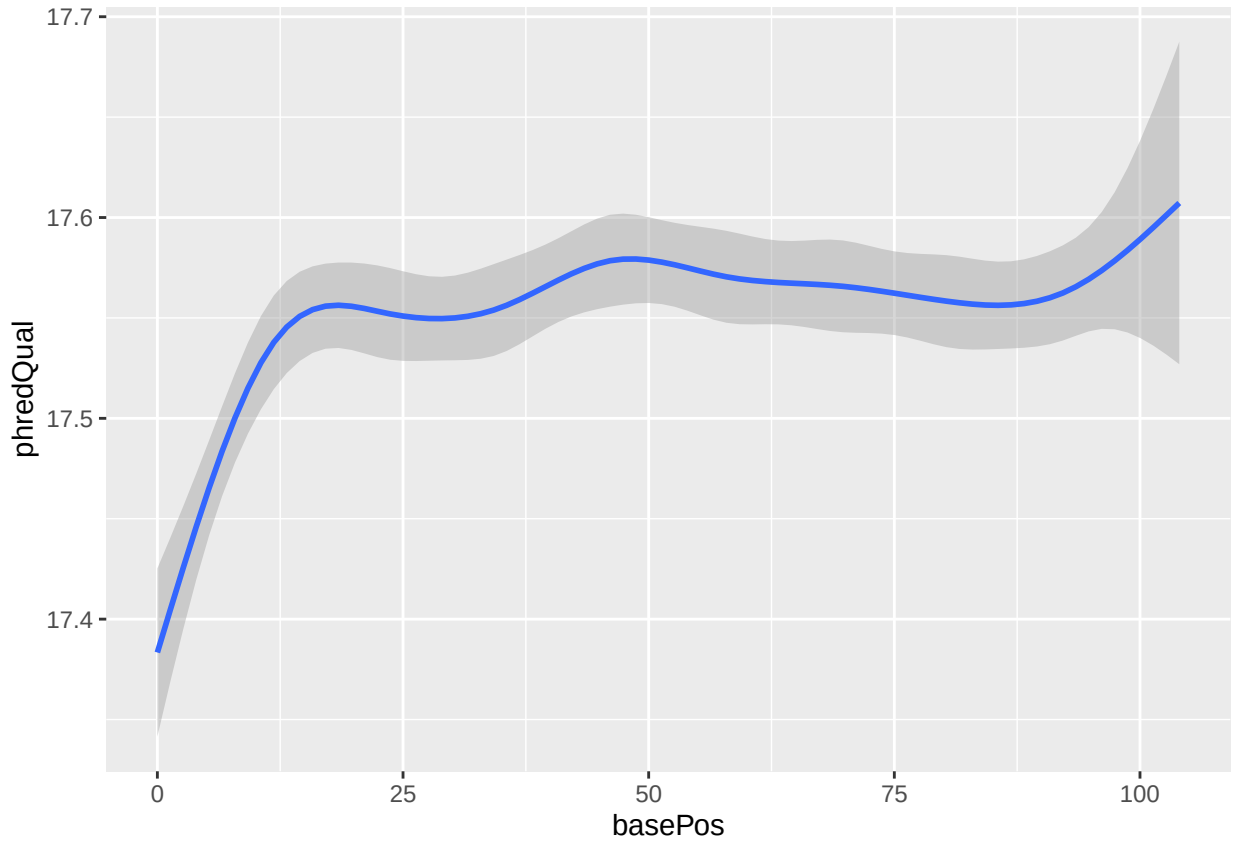
```
fwQualData = py$forwardQuality
fwQualData$record = paste0("read_", 1:dim(fwQualData)[1])
```

```
fwQualData = fwQualData %>%
  pivot_longer(cols = c(1:105), names_to = "basePos", values_to = "phredQual")
fwQualData$basePos = as.numeric(fwQualData$basePos)
head(fwQualData)
```

```
## # A tibble: 6 x 3
##   record basePos phredQual
##   <chr>    <dbl>     <dbl>
## 1 read_1       0        17
## 2 read_1       1        17
## 3 read_1       2        17
## 4 read_1       3        15
## 5 read_1       4        17
## 6 read_1       5        18
```

Now we can plot the results from the forward reads.

```
ggplot(fwQualData, aes(x=basePos, y=phredQual))+
  geom_smooth()
```



Variant Calling

Characteristics of the variants

We import a VCF file by using the library **VariantAnnotation. *Inside this package, the function* readVcf* allows us to point to the VCF we want to import: we need to specify which genome version was used.

```
vcf <- readVcf("variants/results_ann.vcf", genome = "hg38")
```

Let's look at the structure: the VCF class has a few accessors which allow us to inspect different sections of the VCF file.

The *rowRanges* accessor allows us to see the coordinates and alleles:

```
head(rowRanges(vcf))
```

```
## GRanges object with 6 ranges and 5 metadata columns:
##                       seqnames    ranges strand | paramRangeID            REF
##                          <Rle> <IRanges>  <Rle> |     <factor> <DNAStringSet>
##    chr21:6456942_G/A    chr21   6456942      * |           NA              G
##    chr21:7819308_C/T    chr21   7819308      * |           NA              C
##    chr21:7819461_G/A    chr21   7819461      * |           NA              G
##   chr21:14162955_C/G    chr21  14162955      * |           NA              C
##            rs61740029    chr21  14165268      * |           NA              A
##          rs140307100    chr21  14964781      * |           NA              T
##                                    ALT      QUAL      FILTER
##                       <DNAStringSetList> <numeric> <character>
##    chr21:6456942_G/A                  A    106.91           .
##    chr21:7819308_C/T                  T     51.91           .
##    chr21:7819461_G/A                  A     67.91           .
##   chr21:14162955_C/G                  G     37.91           .
##            rs61740029                 G    483.48           .
##          rs140307100                  C    645.48           .
##   -------
##   seqinfo: 1 sequence from hg38 genome
```

The *info* accessor allows us to see the annotations:

```
head(info(vcf))
```

```
## DataFrame with 6 rows and 21 columns
##                                AC            AF        AN BaseQRankSum        DB
##                     <IntegerList> <NumericList> <integer>    <numeric> <logical>
## chr21:6456942_G/A              1          0.25         4        5.250     FALSE
## chr21:7819308_C/T              1          0.25         4        2.910     FALSE
## chr21:7819461_G/A              1          0.25         4        1.630     FALSE
## chr21:14162955_C/G             1          0.25         4        0.792     FALSE
## rs61740029                     2           0.5         4           NA      TRUE
## rs140307100                    2           0.5         4           NA      TRUE
##                           DP       END ExcessHet        FS InbreedingCoeff
##                     <integer> <integer> <numeric> <numeric>       <numeric>
## chr21:6456942_G/A         137        NA    3.0103     3.550              NA
```

```
## chr21:7819308_C/T        103        NA    3.0103     0.000              NA
## chr21:7819461_G/A         71        NA    3.0103     7.911              NA
## chr21:14162955_C/G        54        NA    3.0103     0.000              NA
## rs61740029               106        NA    0.7918     0.000              NA
## rs140307100               64        NA    0.7918     0.000              NA
##                          MLEAC          MLEAF        MQ MQRankSum        QD
##                    <IntegerList> <NumericList> <numeric> <numeric> <numeric>
## chr21:6456942_G/A              1          0.25     29.98    -0.115      1.32
## chr21:7819308_C/T              1          0.25     33.71    -1.335      1.13
## chr21:7819461_G/A              1          0.25     30.57    -0.147      3.99
## chr21:14162955_C/G             1          0.25     60.00     0.000      5.42
## rs61740029                     2           0.5     60.00        NA     15.60
## rs140307100                    2           0.5     60.00        NA     22.26
##                       RAW_MQandDP ReadPosRankSum       SOR
##                     <IntegerList>      <numeric> <numeric>
## chr21:6456942_G/A          NA,NA          0.016     0.367
## chr21:7819308_C/T          NA,NA         -1.004     0.884
## chr21:7819461_G/A          NA,NA          2.010     0.515
## chr21:14162955_C/G         NA,NA         -0.652     0.495
## rs61740029                 NA,NA             NA     1.402
## rs140307100                NA,NA             NA     1.255
##                                                                          ANN
##                                                                <CharacterList>
## chr21:6456942_G/A  A|synonymous_variant..,A|synonymous_variant..,A|synonymous_variant..,...
## chr21:7819308_C/T                            T|synonymous_variant..,T|synonymous_variant..
## chr21:7819461_G/A                            A|synonymous_variant..,A|synonymous_variant..
## chr21:14162955_C/G G|intron_variant|MOD..,G|intron_variant|MOD..,G|intron_variant|MOD..,...
## rs61740029         G|missense_variant|M..,G|missense_variant|M..,G|missense_variant|M..,...
## rs140307100                                                  C|missense_variant|M..
##                              LOF             NMD
##                    <CharacterList> <CharacterList>
## chr21:6456942_G/A
## chr21:7819308_C/T
## chr21:7819461_G/A
## chr21:14162955_C/G
## rs61740029
## rs140307100
```

and the *geno* accessor allows us to access the genotypes of the samples represented in the VCF file

```
head(geno(vcf))
```

```
## List of length 6
## names(6): GT AD DP GQ MIN_DP PGT
```

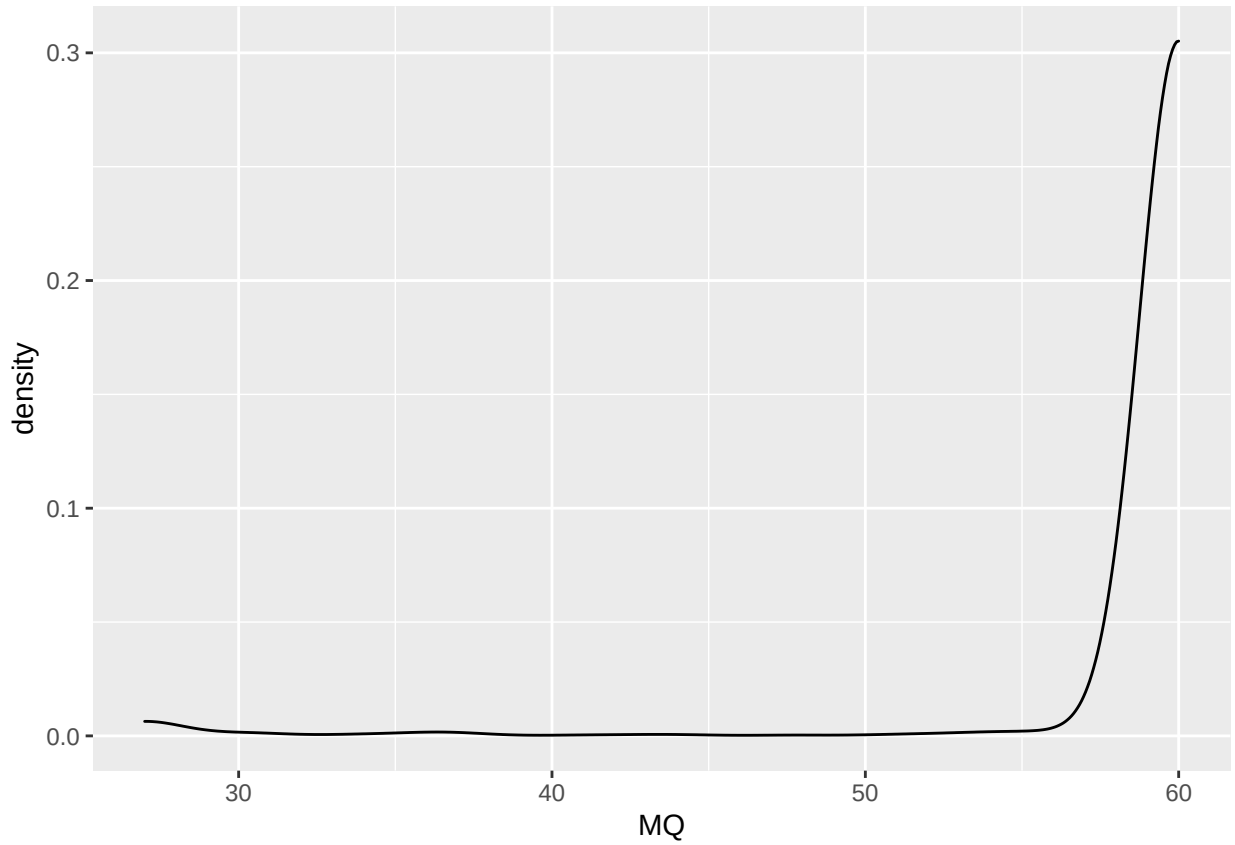
To simplify our analyses we can create a tibble, which we need in order to plot or create tables. Using *lapply* and some functions provided by the teacher, we can unnest the annotations created by **snpEff** and simplify the structure of the data.

```
variants <- as_tibble(rowRanges(vcf))
variants$variantName = names(rowRanges(vcf))
variants = cbind(variants, as_tibble(geno(vcf)$GT))
variants$DP = info(vcf)$DP
```

```
variants$QD = info(vcf)$QD
variants$MQ = info(vcf)$MQ
variants$gene <- unlist(lapply(info(vcf)$ANN, get_most_severe_gene))
variants$consequence <- unlist(lapply(info(vcf)$ANN, get_most_severe_consequence))
variants$impact <- unlist(lapply(info(vcf)$ANN, get_most_severe_impact))
```

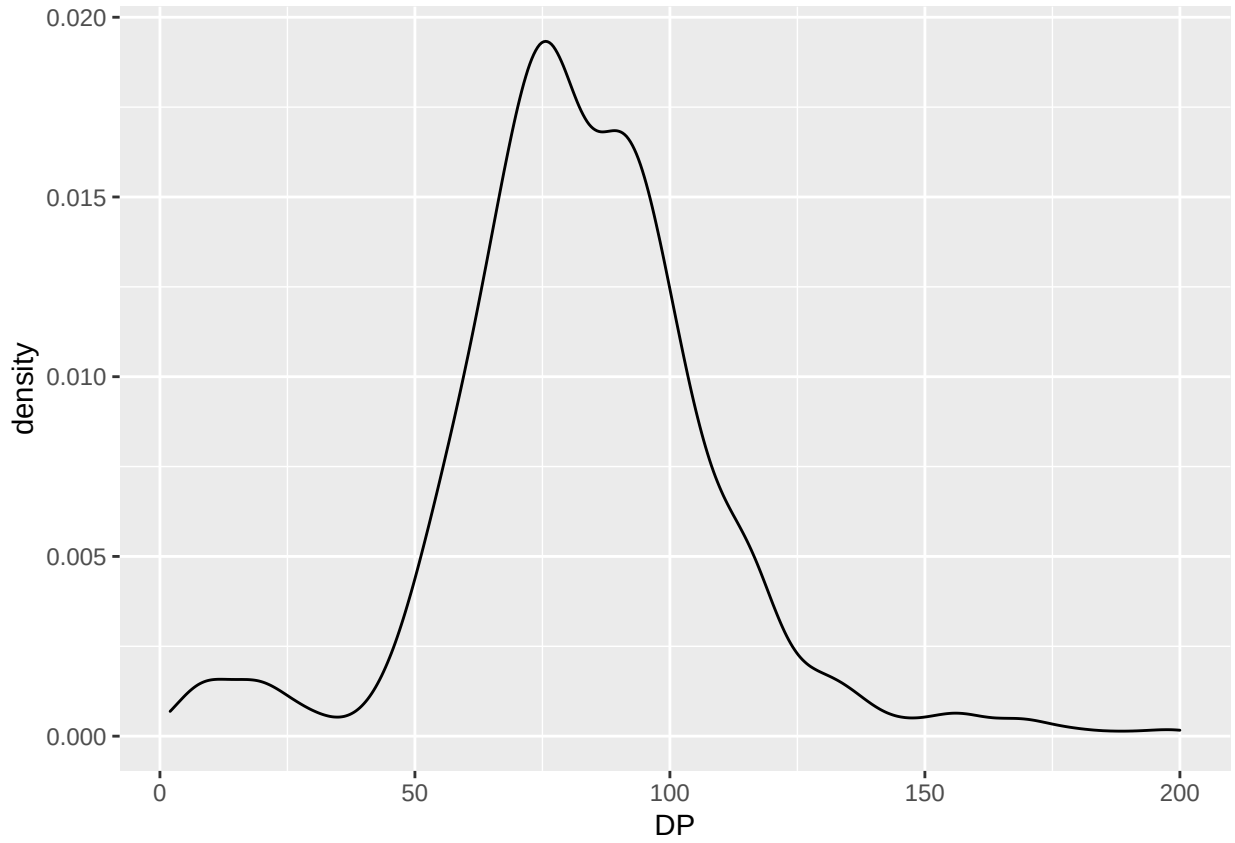
Now that we have a tibble, we can use it to plot some quality data, like the *mapping quality* of the variants:

```
ggplot(variants, aes(x=MQ,y=..density..))+
  geom_density()
```



or the *depth* they have been sequenced at:

```
ggplot(variants, aes(x=DP,y=..density..))+
  geom_density()
```

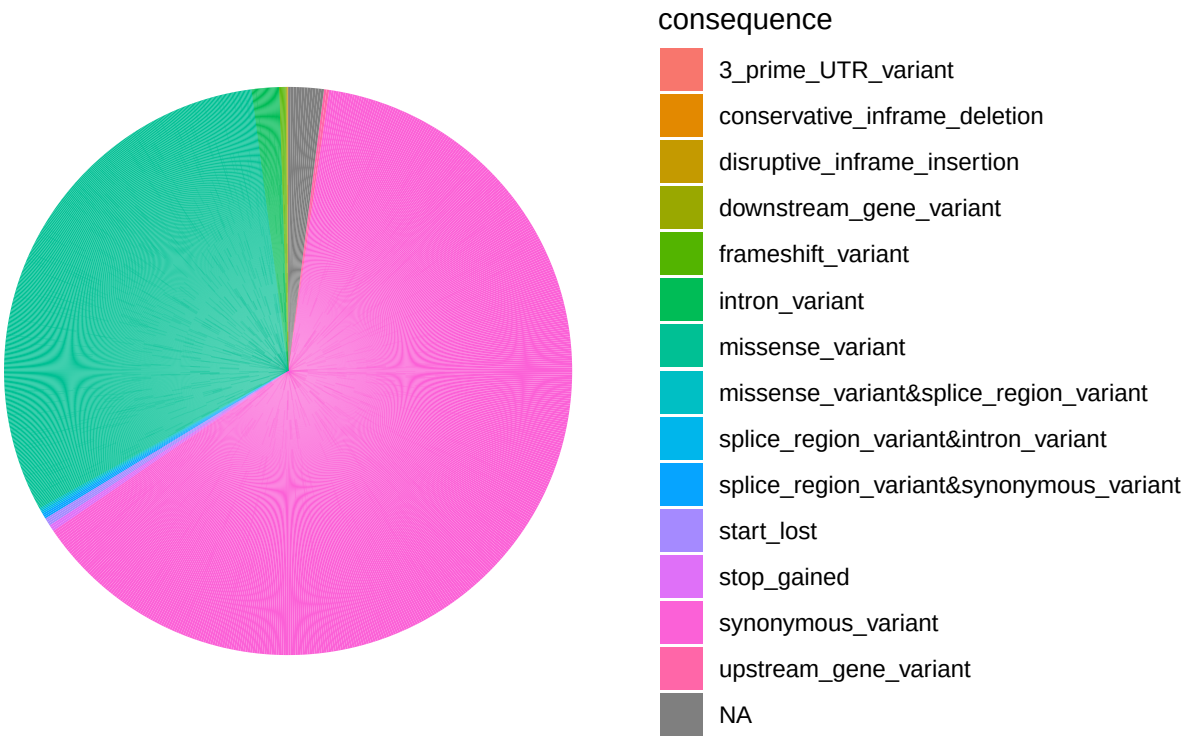


Variant consequences

Naturally, an interesting thing is looking at the variant consequences once we have annotated the variants.

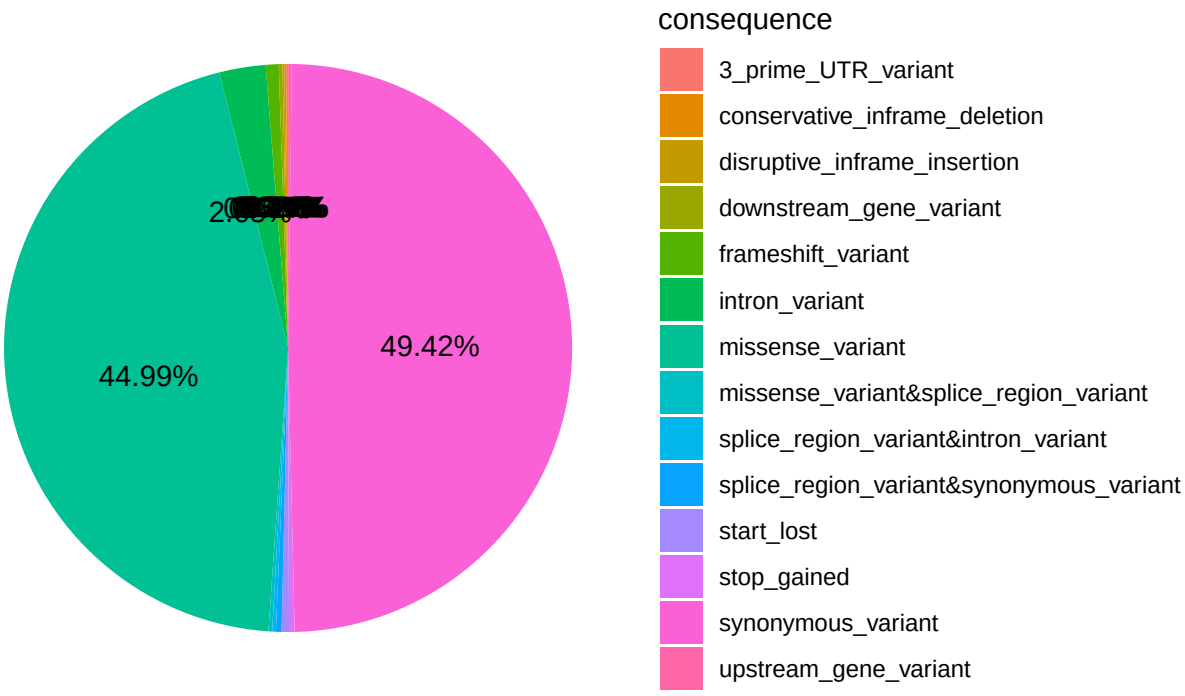
A Pie Chart is in fact a Bar plot with polar coordinates: let's see how we can do it in R

```
ggplot(variants, aes(x="", y=consequence, fill=consequence))+
  geom_bar(width = 1, stat = "identity")+
  coord_polar(theta = "y")+
  theme_void()
```



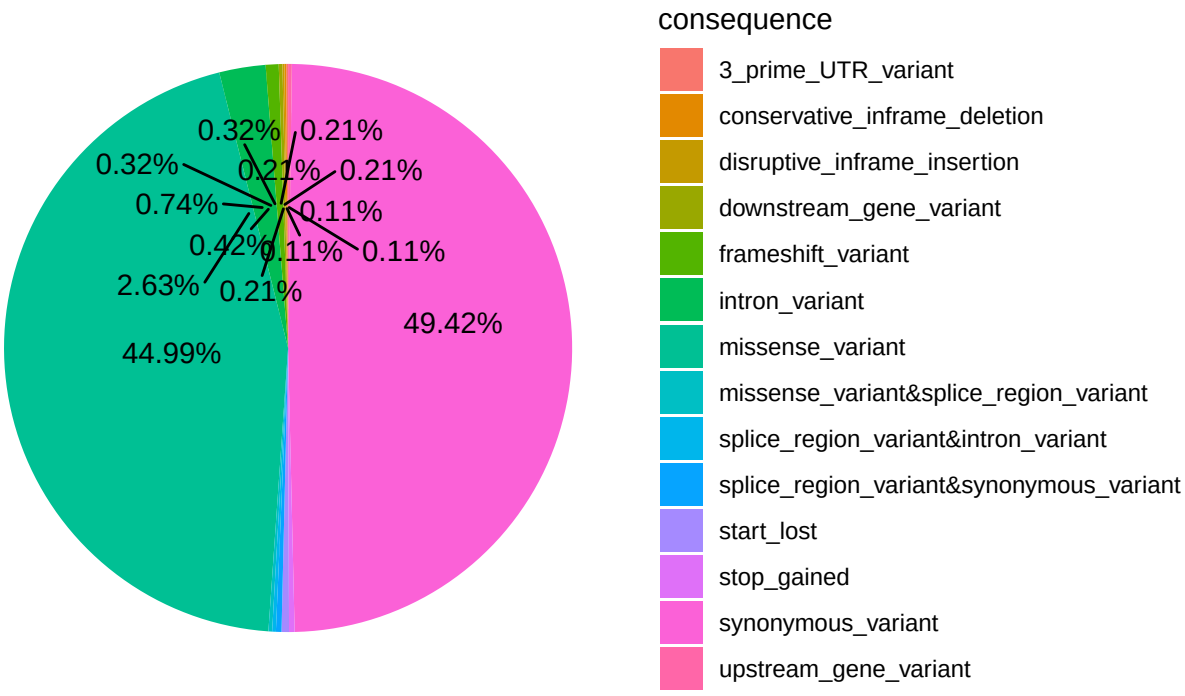
There is a better way to plot this, and we can improve the plot as follows

```
variants %>%
  filter(!is.na(consequence)) %>%
  count(consequence) %>%
  mutate(count = n,
         percent = paste0(round(count/sum(count) * 100, digits = 2), "%"))%>%
  arrange(desc(count)) %>%
  mutate(lab.ypos = cumsum(count) - 0.5 * count) %>%
  ggplot(aes(x="", y=count, fill=consequence))+
  geom_bar(width = 1, stat = "identity")+
  coord_polar(theta = "y")+
  geom_text(aes(y=lab.ypos, label=percent))+
  theme_void()
```



But the labels are still awfully overlapping and they are not readable. In different situations we can use **ggrepel** to avoid labels overlaps, like follows:

```
variants %>%
  filter(!is.na(consequence)) %>%
  count(consequence) %>%
  mutate(count = n,
         percent = paste0(round(count/sum(count) * 100, digits = 2), "%"))%>%
  arrange(desc(count)) %>%
  mutate(lab.ypos = cumsum(count) - 0.5 * count) %>%
  ggplot(aes(x="", y=count, fill=consequence))+
  geom_bar(width = 1, stat = "identity")+
  coord_polar(theta = "y")+
  ggrepel::geom_text_repel(aes(y=lab.ypos, label=percent), max.overlaps = Inf)+
  theme_void()
```



In this case it does not help us very much, due to the nature of the data. Let's table the results instead:

```
variants %>%
  filter(!is.na(consequence)) %>%
  group_by(consequence) %>%
  summarise(count=n()) %>%
  arrange(desc(count)) %>%
  kable()
```

consequence	count
synonymous_variant	469
missense_variant	427
intron_variant	25
frameshift_variant	7
start_lost	4
splice_region_variant&synonymous_variant	3
stop_gained	3
downstream_gene_variant	2
missense_variant&splice_region_variant	2
splice_region_variant&intron_variant	2
upstream_gene_variant	2
3_prime_UTR_variant	1
conservative_inframe_deletion	1
disruptive_inframe_insertion	1

Disease candidates

We can filter the VCF as we did on bash, but this time is going to be easier:

```
selectedVars = variants %>%
  filter(
    impact == "HIGH",
    normal == "0/0"
  )
```

Let's use this to table the result in a simple table:

seqnames	start	REF	ALT	gene	consequence
chr21	32576780	A	AC	TCP10L	frameshift_variant
chr21	32631618	A	AGTATT	SYNJ1	frameshift_variant
chr21	33576378	G	GA	SON	frameshift_variant
chr21	37472761	A	G	DYRK1A	start_lost
chr21	43741551	A	G	PDXK	start_lost
chr21	44339194	T	C	CFAP410	start_lost
chr21	44406660	C	T	TRPM2	stop_gained
chr21	45504485	TTCGGCTCC	T	COL18A1	frameshift_variant
chr21	45504493	CA	C,	COL18A1	frameshift_variant
chr21	45504507	CCCCCGGCCCCCCAGGCCCCCCAGGCCCA	, C	COL18A1	frameshift_variant
chr21	45989090	C	T	COL6A1	stop_gained

Now, like we did in class, we want to use websites like *ClinVar* or *gnomAD* to add information to our table and present it: let's imagine we have checked on clinvar if the variant is pathogenic or not, and whether the gene is constrained for LoF variation on gnomad.

The information below does not correspond to a real search result:

```
selectedVars$clinvar = c("benign", "benign", "benign", "benign", "benign", "benign",
                         "benign", "benign", "benign", "pathogenic", "pathogenic")
selectedVars$constrained = c("not constrained", "not constrained", "not constrained",
                             "not constrained", "not constrained", "not constrained",
                             "not constrained", "not constrained", "not constrained",
                             "constrained", "constrained")
```

Now we can table this information and present it in our report. We might want to select only some columns to improve the readability of the table:

```
selectedVars %>%
  select(seqnames, start, gene, consequence, clinvar, constrained) %>%
  kable()
```

seqnames	start	gene	consequence	clinvar	constrained
chr21	32576780	TCP10L	frameshift_variant	benign	not constrained
chr21	32631618	SYNJ1	frameshift_variant	benign	not constrained
chr21	33576378	SON	frameshift_variant	benign	not constrained
chr21	37472761	DYRK1A	start_lost	benign	not constrained

seqnames	start	gene	consequence	clinvar	constrained
chr21	43741551	PDXK	start_lost	benign	not constrained
chr21	44339194	CFAP410	start_lost	benign	not constrained
chr21	44406660	TRPM2	stop_gained	benign	not constrained
chr21	45504485	COL18A1	frameshift_variant	benign	not constrained
chr21	45504493	COL18A1	frameshift_variant	benign	not constrained
chr21	45504507	COL18A1	frameshift_variant	pathogenic	constrained
chr21	45989090	COL6A1	stop_gained	pathogenic	constrained