BiocExomeData

Sean Davis

28 February 2012

Contents

1 Introduction

Second- and third-generation sequencing (NGS) technologies are revolutionizing all fields of biology. However, NGS is still a relatively expensive proposition. Therefore, several strategies have been developed to target regions of interest in the genome for sequencing. The most common approach is to target the exome. Doing so reduces costs by approximately 4-8 fold over whole-genome sequencing while still probing the regions most likely to cause disease. A recent review highlights some of the pros and cons of using the exome sequencing approach over whole-genome sequencing.

Several capture technologies exist and new protocols and reagents are being introduced at a staggering pace given the relative simplicity of the idea. A performance comparison of exome DNA sequencing technologies presents a view of the field as of September, 2011. At the end of the day, though, exome sequencing has had remarkable staying power and will continue to play a role in sequencing experimental design and analysis for some time to come.

1.1 Goals

A "complete" discussion of all aspects of exome capture sequencing is simply not possible. The goal of this vignette is to show use cases for R and Bioconductor that answer questions important in exome capture sequencing.

- Evaluate a capture technology
- Evaluate a capture experiment
- Perform quality control on exome capture variant calls
- Examine variants resulting from a couple of exome capture datasets

This vignette is **NOT** a workflow for performing and analysing exome capture data.

1.2 Description of data

In this tutorial, we will be working with two small datasets, each composed of three exome captured samples. Data from chromosome 22 have been isolated to keep the datasets manageable.

Both datasets have been aligned to the human reference genome to produce BAM files, processed using GATK tools (see figure), and then had variants called using samtools.

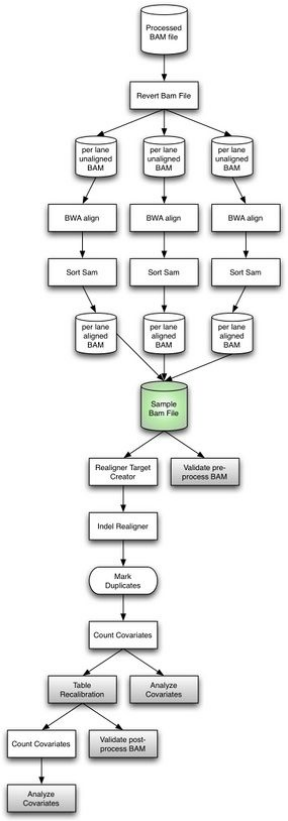


Figure 1: Example workflow for postprocessing a BAM file using GATK toolset (taken from the GATK website)

The first dataset is a subset of data from the 1000 Genomes pilot. The data were downloaded from here and includes samples NA19909, NA19914, and NA19916. The data were prepared using the following commands:

```
#!/bin/bash
/usr/local/samtools/samtools view -h ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical
/usr/local/samtools/samtools view -h ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical
/usr/local/samtools/samtools view -h ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical
/usr/local/samtools/samtools index NA19914.chr22.bam
/usr/local/samtools/samtools index NA19909.chr22.bam
/usr/local/samtools/samtools index NA19916.chr22.bam
```

The second data set is unpublished data from the NCI-60 cell line panel.

Chosen to represent common cancers, the NCI-60 panel has been extensively profiled for copy number variation, gene expression, DNA methylation, drug sensitivity, miRNA expression, and limited DNA sequence variation (Sanger sequencing). The data from three lines, HOP-92, SK-MEL-5, and HCT-15 are included here as illustrative examples.

Table 1: Cell line names and cancer types they represent

Cell Line Name	Cancer Type
HCT-25	Colon Cancer
HOP-92	Lung Cancer
SK-MEL-5	Melanoma

2 Evaluating the Capture Technologies

A first step in designing and analysing an exome capture experiment is to evaluate the technology proposed for the capture. Typically, the manufacturers of a capture kit will provide a BED file that describes either the **targets** or the **baits**. A **target** is a region of the genome which is targeted for sequencing. In exome capture, the targets are typically the coding exons but may additionally include UTRs, splice sites, or promoter regions. A **bait** is a sequence region designed to capture around the targets; typically, baits are short oligos that are used to hybridize to the input DNA. Note that baits and targets are different concepts. When evaluating a capture strategy, we are typically going to be concerned with how well the baits overlap with our targets.

For each of the two capture experiments, I have supplied the capture regions as BED files. As a quick representative example, we will look at the Agilent 38MB SureSelect capture kit that was used for the NCI60 capture data.

```
> options(width=50)
> library(ascii)
> library(rtracklayer)
> ag38mss = import(system.file('extdata/capture/Agilent_SureSelect_Human_All_Exon_38Mb_
     package='BasicExomeExample'))
> class(ag38mss)
[1] "UCSCData"
```

```
attr(,"package")
[1] "rtracklayer"
```

As we can see, this is an object of class *UCSCData*. We are going to be doing some interval operations and examinations on these data, so we first convert to a GRanges object.

```
> ag38mssGR = as(ag38mss,"GRanges")
> ag38mssGR
GRanges with 343850 ranges and 2 elementMetadata cols:
           seqnames               ranges strand
              <Rle>            <IRanges>  <Rle>
       [1]     chr1       [30276, 30395]      +
       [2]     chr1       [30276, 30395]      +
       [3]     chr1       [30276, 30395]      +
       [4]     chr1       [30312, 30431]      -
       [5]     chr1       [30312, 30431]      -
       [6]     chr1       [69070, 69189]      -
       [7]     chr1       [69070, 69189]      +
       [8]     chr1       [69070, 69189]      +
       [9]     chr1       [69190, 69309]      -
       ...      ...                  ...    ...
  [343842]     chrY [59337930, 59338049]      +
  [343843]     chrY [59338050, 59338169]      +
  [343844]     chrY [59338747, 59338866]      +
  [343845]     chrY [59340176, 59340295]      +
  [343846]     chrY [59342461, 59342580]      +
  [343847]     chrY [59342581, 59342700]      +
  [343848]     chrY [59342676, 59342795]      +
  [343849]     chrY [59342901, 59343020]      +
  [343850]     chrY [59343021, 59343140]      +
             |         name     score
             |  <character> <numeric>
       [1]   | A_36_B320275      1000
       [2]   | A_36_B320398      1000
       [3]   | A_36_B320455      1000
       [4]   | A_36_B320991      1000
       [5]   | A_36_B321103      1000
       [6]   | A_36_B161002      1000
       [7]   | A_36_B223635      1000
```

```
      [8]   | A_36_B241957      1000
      [9]   | A_36_B161001      1000
      ... ...          ...       ...
  [343842]   | A_36_B142931         1
  [343843]   | A_36_B142932         1
  [343844]   | A_36_B152403         1
  [343845]   | A_36_B143525         1
  [343846]   | A_36_B144510         1
  [343847]   | A_36_B144511         1
  [343848]   | A_36_B144512         1
  [343849]   | A_36_B144514         1
  [343850]   | A_36_B144515         1
  ---
  seqlengths:
        chr1     chr10 ...       chrX      chrY
   249231325 135516111 ... 155240134  59343140
> seqlevels(ag38mssGR) = sub('chr','',seqlevels(ag38mssGR))
```

Now, we can begin to look at the data. What are the sizes of the baits?

```
> summary(width(ag38mssGR))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    114     120     120     120     120     126
> print(table(width(ag38mssGR)))
   114    115    116    117    118    119    120
    22     47    136     58    128    638 342439
   121    122    123    124    125    126
   183     71     43     78      1      6
```

We might ask the question about how the baits are spaced relative to each other. In particular, how many baits cover a particular region of the genome? We can calculate coverage to answer that question.

```
> baitcov = coverage(ag38mssGR)
```

The `baitcov` object is a list of integer *Rle* objects. We want to know how many bases in the genome have each level of coverage. The goal, then, is to create a two-column table with coverage level as one column and number of bases (at that coverage level) as the second column.

```
> # make a single long vector of each of the lengths and values of
```

```
> # the baitcov object
> rl = do.call(c,sapply(baitcov,runLength))
> rv = do.call(c,sapply(baitcov,runValue))
> # need to use "as.numeric" here to avoid integer overflow
> baseCov = aggregate(rl,by=list(rv),function(tmp) {
     return(sum(as.numeric(tmp)))})
> colnames(baseCov) = c('coverage','numberOfBases')
> baseCov
   coverage numberOfBases
1         0    3053253777
2         1      37393238
3         2        886217
4         3        216572
5         4        110772
6         5         45733
7         6         44051
8         7         26459
9         8          6908
10        9           696
11       10          5347
12       11         15326
13       12           856
14       13           350
15       14           671
16       22           744
```

Agilent advertises this as a 38MB capture kit and the result above suggests that this is, indeed, the case. Interestingly, most bases are captured by only one bait, but some are captured by many more baits than that. There may be good design reasons for this redundancy.

It is also interesting to examine the extent to which the baits cover known targets. For the purposes of this exercise, we will be using the UCSC known genes TranscriptDB package to define our target regions of interest.

```
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
```

We are usually interested in the coding regions of the genes when performing exome capture experiments, so we pull those regions using the `cds` method.

```
> cdsRegions = cds(TxDb.Hsapiens.UCSC.hg19.knownGene)
```

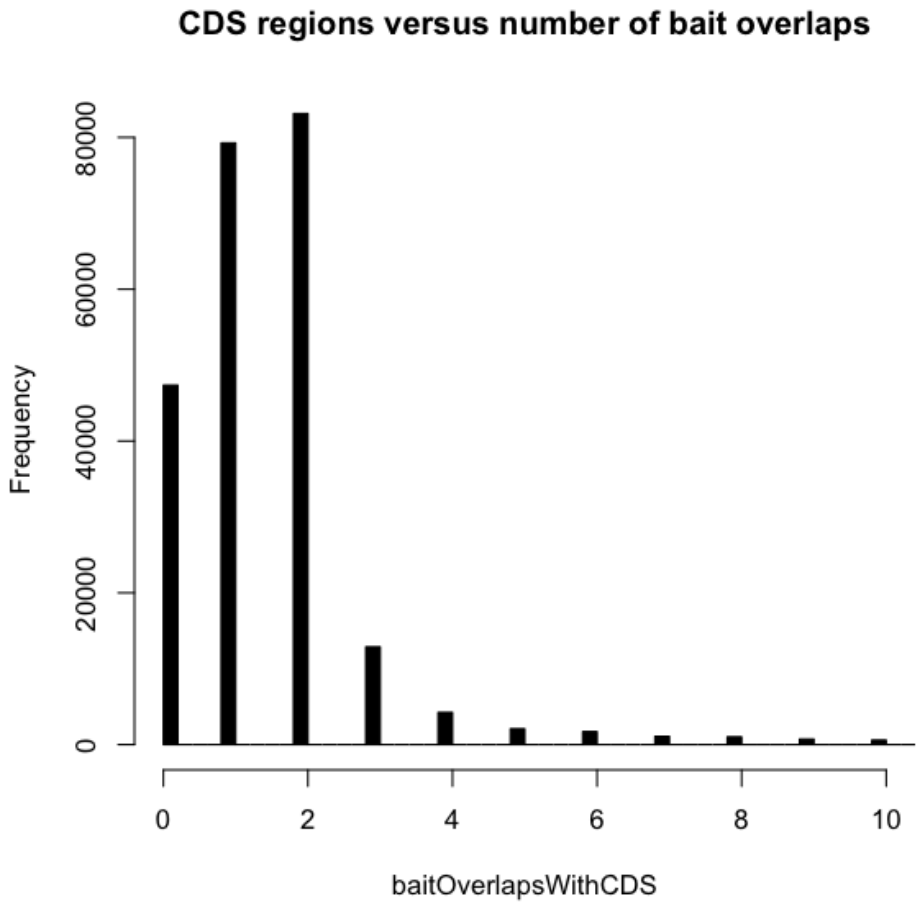
```
> # We are being unsafe and ignoring sequence lengths
> seqlengths(cdsRegions) <- rep(NA,length(seqlengths(cdsRegions)))
> # For the purposes of this tutorial, we will try to stick
> # with the convention of NO 'chr' in chromosome names
> seqlevels(cdsRegions) <- sub('chr','',seqlevels(cdsRegions))
```

Because the baits work by hybridization to the fragmented sample library, the baits can probably be extended by about 50 bp to allow for this extended capture sequence.

```
> # resize regions to be 50bp larger on each end
> ag38mssGRpadded = resize(ag38mssGR,width=width(ag38mssGR)+100,fix="center")
> # and fix so strand is '*' since stranded overlaps are not important
> strand(ag38mssGRpadded) = '*'
```

Finally, we are ready to count the overlaps between the cdsRegions and the expanded baits.

```
> # Get the number of baits overlapping each cdsRegion
> baitOverlapsWithCDS = countOverlaps(cdsRegions,ag38mssGRpadded)
> hist(baitOverlapsWithCDS,xlim=c(0,10),breaks='scott',col='black',
       main='CDS regions versus number of bait overlaps')
```



3 Overall capture quality evaluation

After the exome capture experiment, quality control of the sequencing files, and alignment to the genome are complete, we are often interested to look at the data from a fairly high level. Three questions are often of interest.

1. What is the enrichment of the exome capture regions?
2. What is the per-base coverage of the targets?

To examine these issues, we are going to use three samples from the 1000 Genomes project, but we will limit our analyses to only chromosome 22 to

keep the analyses snappy.

3.1 Capture Enrichment

If no capture had been used, the reads would be "randomly" spread across the genome. We are interested in how well our capture reaction worked. By counting the proportion of aligned reads that fall on our "extended" bait regions versus the number expected if the reads were randomly placed, we can get an estimate of the enrichment availed us by using targeted sequencing over whole-genome sequencing.

The 1000 Genomes project used a different capture reagent than what we worked with in the first exercise, so we load the 1000 Genomes regions.

```
> # This is a non-standard BED format, so must load using
> # read.delim rather than rtracklayer::import
> ag1000gss = read.delim(system.file('extdata/capture/20110225.exome.consensus.annotati
     package='BasicExomeExample'),sep="\t",header=FALSE)
> # convert to GRanges
> ag1000gssGR = GRanges(seqnames=ag1000gss[,1],
     ranges=IRanges(start=ag1000gss[,2],end=ag1000gss[,3]))
> seqlevels(ag1000gssGR) = sub('chr','',seqlevels(ag1000gssGR))
> # expand each region by 50bp
> ag1000gssGR = resize(ag1000gssGR,width=width(ag1000gssGR)+100)
> # collapse overlapping regions onto single regions
> ag1000gssGR = reduce(ag1000gssGR)
> sum(width(ag1000gssGR))
[1] 46899965
```

So, this capture reagent would be expected to capture about 47MB of sequence.

Next, we need to read in a representative BAM file.

```
> alignedGR = readGappedAlignments(system.file('extdata/1000G/bam/NA19909.chr22.bam',pa
```

We limit our capture regions to those on chromosome 22.

```
> chr22captureRegions = ag1000gssGR[seqnames(ag1000gssGR)=='22',]
```

Now, we need to calculate the number of reads that overlap a capture region and the number of reads that fall outside the capture regions.

```
> incapture = countOverlaps(chr22captureRegions,alignedGR)
> outcapture = length(alignedGR)-sum(incapture)
```

And the number of bases of captured and non-captured bases is approximately:

```
> capturedBases = sum(width(chr22captureRegions))
> # This is an approximation!
> notCapturedBases = sum(width(gaps(chr22captureRegions)))
```

Finally, I calculate the enrichment as a ratio of ratios.

```
> enrichment = (sum(incapture)/capturedBases)/(outcapture/notCapturedBases)
> enrichment
[1] 29.05708
```

So, we got roughly 30-fold enrichment over a random distribution of reads. There is some sloppiness in the code above, but a metric like this one can certainly pick up a failed experiment and could be used to compare across runs. Also, while this is an overestimate given that chromosome 22 is acrocentric, this level of enrichment is associated with a good capture sequencing experiment.

3.2 Base-level coverage

Calculating base-level coverage is quite simple using the objects we have already set up.

```
> cvg = coverage(alignedGR)
> class(cvg)
[1] "SimpleRleList"
attr(,"package")
[1] "IRanges"
> cvg
SimpleRleList of length 86
$`1`
'integer' Rle of length 249250621 with 1 run
  Lengths: 249250621
  Values :         0

$`2`
```

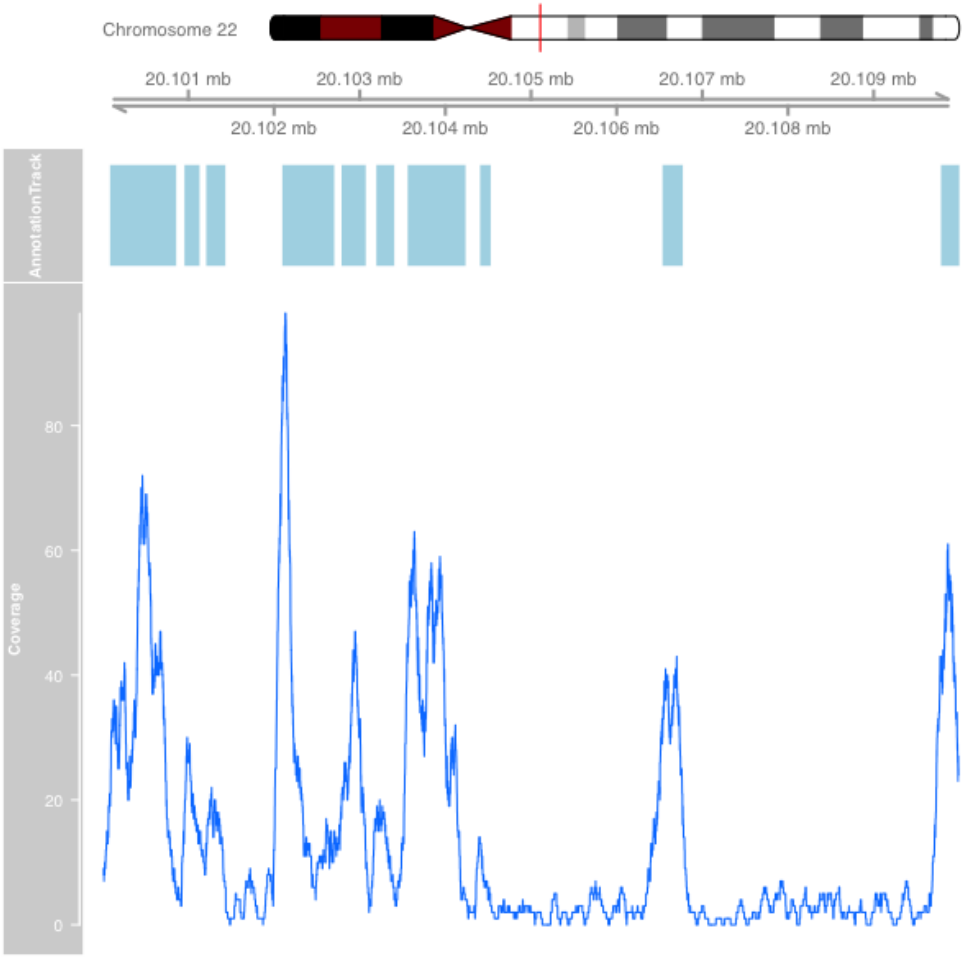
```
'integer' Rle of length 243199373 with 1 run
  Lengths: 243199373
  Values :         0

$`3`
'integer' Rle of length 198022430 with 1 run
  Lengths: 198022430
  Values :         0

$`4`
'integer' Rle of length 191154276 with 1 run
  Lengths: 191154276
  Values :         0

$`5`
'integer' Rle of length 180915260 with 1 run
  Lengths: 180915260
  Values :         0

...
<81 more elements>
> cvg[['22']]
'integer' Rle of length 51304566 with 2590278 runs
  Lengths: 16050083        5 ...    60019
  Values :        0        1 ...        0
```

3.2.1 Visualizing Coverage in Genomic Context

The Gviz package enables UCSC-like track views of data and annotation and is potentially quite useful for visualizing multiple tracks of data on the same plot. We can use the Gviz package (or the ggbio package; choice is a GOOD THING) to visualize our coverage data in the context of our capture regions to show the relative enrichment for capture regions.

```
> library(rtracklayer)
> if(!require(Gviz)) {
     biocLite('Gviz')
     require(Gviz)
     }
> itrack = IdeogramTrack(genome='hg19',chromosome='22')
```

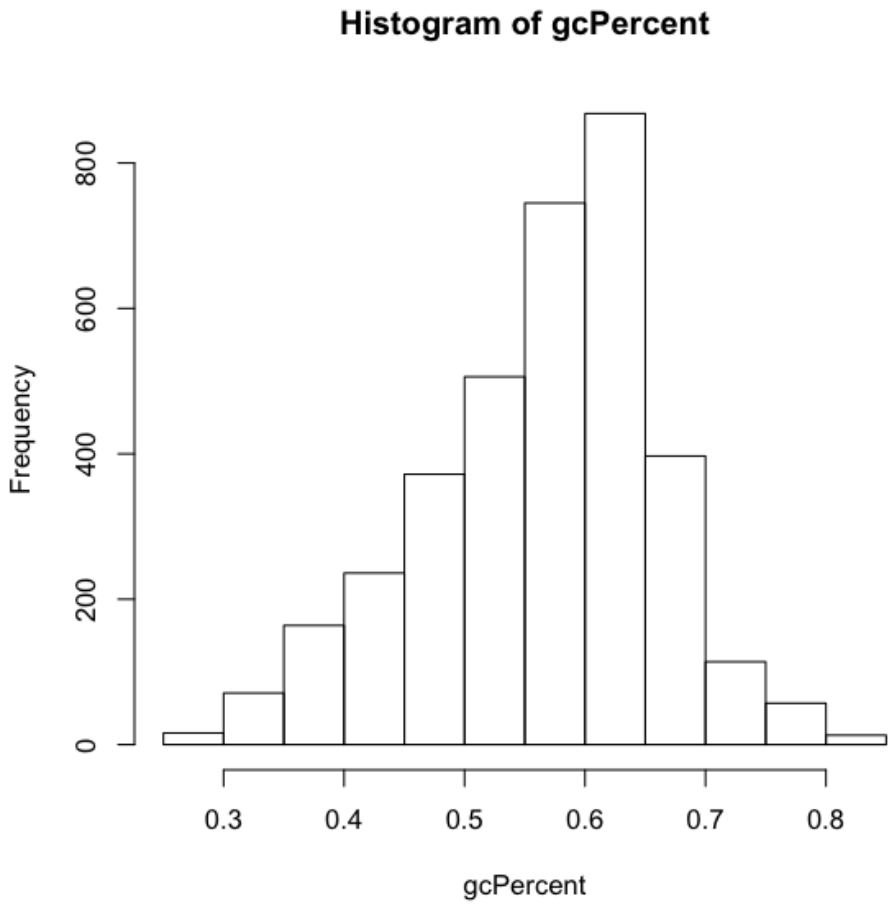
```
> gtrack = GenomeAxisTrack(genome='hg19')
> # funny issue with chromosome names needing to start with 'chr' for Gviz
> chr22CR = chr22captureRegions
> seqlevels(chr22CR) = paste('chr',seqlevels(chr22CR),sep="")
> atrack = AnnotationTrack(chr22CR,chromosome='22',genome='hg19')
> cvgVec = as.integer(cvg$'22')
> dtrack = DataTrack(start=20e6:21e6,
     width=rep(1,1e6+1),
     data=cvgVec[20e6:21e6],chromosome='22',name='Coverage',type='l')
> plotTracks(list(itrack,gtrack,atrack,dtrack),from=20.1e6,to=20.11e6)
```



3.2.2 GC Content and Coverage

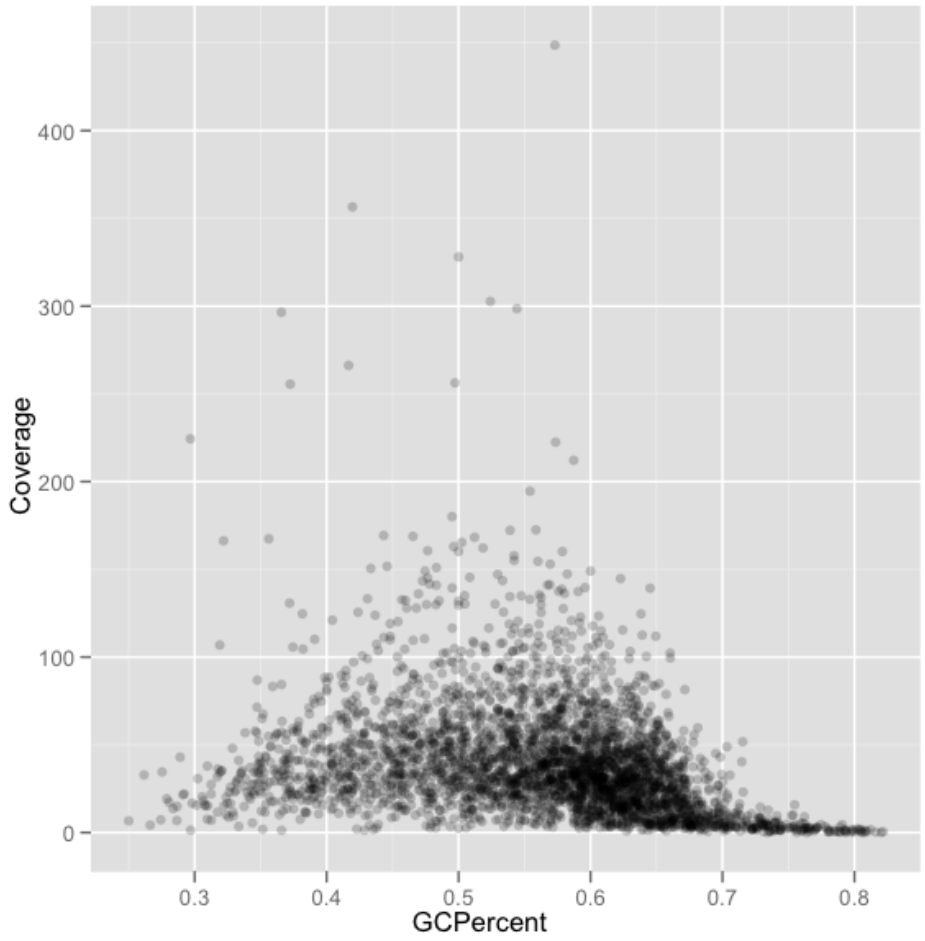
There is a well-known phenomenon in capture sequencing (indeed, with the Illumina sequencing platform itself) whereby GC content of the region being sequenced affects the coverage in that region. Using the Biostrings, BSgenome, and BSgenome data packages, it is just a few lines of code to examine how GC content and capture coverage are related.

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> chr22 = Hsapiens$chr22
> # Make views on the chromosome 22 sequence that correspond to
> # the capture regions
> captSeqViews = Views(chr22,ranges(chr22CR))
> # And calculate the GC content:
> gcContent = letterFrequency(captSeqViews,letters="GC")
> totalBases = letterFrequency(captSeqViews,letters="GCAT")
> gcPercent = gcContent/totalBases
> hist(gcPercent)
```



Calculating the average coverage in each capture region is also straightforward.

```
> library(ggplot2)
> cvgViews = Views(cvg$'22',ranges(chr22CR))
> cvgMeans = viewMeans(cvgViews)
> gcCoverage = data.frame(Coverage=cvgMeans,
    GCPercent=as.numeric(gcPercent))
> print(ggplot(gcCoverage,
              aes(x=GCPercent,y=Coverage))+geom_point(alpha=0.2))
```



There appears to be a strong relationship between GC content and coverage such that it will be nearly impossible to capture and sequence regions with GC content above about 0.7.

4 Evaluating Variant Calls

The primary goal of targeted capture sequencing is to produce a list of variants between a reference genome and the sample being sequenced. There are a number of software packages for producing variant calls given a bam file (or a set of them) including:

- GATK Unified Genotyper

- samtools mpileup
- VarScan
- SnvMix
- ...

For the purposes of this little exercise, I have used samtools mpileup like so:

```
/usr/local/samtools/samtools mpileup -D -uf \
  ../fasta/human_g1k_v37.fasta *.bam | bcftools view -vcg - > 1kg.chr22.vcf
```

4.1 The VCF Format

The most common format for describing variants with respect to a reference genome is the Variant Call Format (VCF). A VCF file is a tab-delimited text format that contains a header describing the context and content of the file, including definitions of Tags that are used in the file to provide details of the variant calls. After the header, the actual variant calls are stored in sequential lines of text, one per variant. Here are a few lines of our example 1000 Genomes VCF file.

I have removed the header, FORMAT, INFO, and specific genotypes to make the output a bit easier to read. The columns are fairly self-explanatory. The ID column can also contain the ID of any known variant such as an rsID from the dbSNP database. The FILTER column can contain user-defined tags for filters that apply to the specific variant in the file, such as "$\text{LOW}_{\text{QUALITY}}$" or "INDEL".

The INFO and Genotype columns are a bit more complicated, so I will not describe it in detail and refer the reader to the VCF Format specification for specifics. However, I will point out that each sample is represented by a column in the VCF file. The GT section of that column will typically be one of "0/0", "0/1", or "1/1" which correspond to homozygous reference, heterozygous, and homozygous for the first variant allele. If more than one variant allele is present, the presence of that allele is noted by a 2 or 3 in the genotype.

To understand the tags in the INFO and GENOTYPE columns, one must examine the VCF file header:

```
> readLines(system.file('extdata/1000G/1kg.chr22.vcf',package='BasicExomeExample'),n=1(
 [1] "##fileformat=VCFv4.0"
 [2] "##samtoolsVersion=0.1.18 (r982:295)"
```

Table 2: VCF record example

X.CHROM	POS	ID	REF	ALT	QUAL	FILTER
22	16136566	.	A	G	5	.
22	16139004	.	G	T	16	.
22	16142489	.	T	A	14	.
22	16143946	.	A	G	14	.
22	16148996	.	C	G	16	.
22	16156691	.	G	A	6	.
22	16159060	.	G	A	49	.
22	16159563	.	G	A	17	.
22	16228619	.	T	C	16	.
22	16248248	.	C	T	119	.
22	16264166	.	C	T	22	.
22	16288776	.	A	G,T	45	.
22	16311474	.	A	G	47	.
22	16344474	.	A	G	46	.
22	16344748	.	T	C	46	.

```
 [3] "##INFO=<ID=DP,Number=1,Type=Integer,Description=\"Raw read depth\">"
 [4] "##INFO=<ID=DP4,Number=4,Type=Integer,Description=\"# high-quality ref-forward bas
 [5] "##INFO=<ID=MQ,Number=1,Type=Integer,Description=\"Root-mean-square mapping qualit
 [6] "##INFO=<ID=VDB,Number=1,Type=Float,Description=\"End distance bias\">"
 [7] "##INFO=<ID=FQ,Number=1,Type=Float,Description=\"Phred probability that sample chr
 [8] "##INFO=<ID=AF1,Number=1,Type=Float,Description=\"Max-likelihood estimate of the s
 [9] "##INFO=<ID=CI95,Number=2,Type=Float,Description=\"Equal-tail Bayesian credible in
[10] "##INFO=<ID=PV4,Number=4,Type=Float,Description=\"P-values for strand bias, baseQ
```

As an example, the INFO tag "DP" will be a single (Number=1) value of Type=Integer and represents "Raw read depth".

4.2 1000 Genomes Data

The goal in this section is to examine some qualities of the variant calls made on three samples from the 1000 Genomes Exome Pilot data.

We start by using the VariantAnnotation package to load the VCF file into a VCF object.

```
> library(VariantAnnotation)
```

Table 3: VCF record example: INFO, FORMAT, and GENOTYPE columns

INFO
DP=24;VDB=0.0535;AF1=0.4947;CI95=0.3333,0.6667;DP4=1,1,8,9;MQ=8;FQ=6.27;PV4=1,0.11,0.3
DP=25;VDB=0.0568;AF1=0.172;CI95=0.1667,0.3333;DP4=8,11,3,2;MQ=17;FQ=16.3;PV4=0.63,0.05
DP=14;VDB=0.0179;AF1=0.2182;CI95=0.1667,0.6667;DP4=3,4,2,3;MQ=16;FQ=14.9;PV4=1,0.0021
DP=6;VDB=0.0198;AF1=0.4462;CI95=0.1667,0.6667;DP4=0,1,1,4;MQ=13;FQ=15;PV4=1,1,1,1
DP=16;VDB=0.0176;AF1=0.4365;CI95=0.1667,0.6667;DP4=3,6,3,3;MQ=14;FQ=18.3;PV4=0.62,1,1,
DP=18;VDB=0.0237;AF1=0.3173;CI95=0.1667,0.5;DP4=6,4,4,2;MQ=13;FQ=6.94;PV4=1,0.0026,1,1
DP=29;VDB=0.0421;AF1=0.4986;CI95=0.5,0.5;DP4=7,10,7,5;MQ=13;FQ=52.3;PV4=0.46,2.6e-07,1,
DP=16;VDB=0.0168;AF1=0.3125;CI95=0.1667,0.5;DP4=2,4,4,3;MQ=15;FQ=18.3;PV4=0.59,0.26,0.1
DP=8;VDB=0.0240;AF1=0.4992;CI95=0.3333,0.6667;DP4=2,0,3,3;MQ=17;FQ=18;PV4=0.46,0.085,0
DP=27;VDB=0.0319;AF1=0.5;CI95=0.5,0.5;DP4=2,3,10,10;MQ=17;FQ=122;PV4=1,1,0.00049,0.097
DP=15;VDB=0.0232;AF1=0.4921;CI95=0.3333,0.5;DP4=4,4,5,1;MQ=19;FQ=24.7;PV4=0.3,0.01,0.3,
DP=13;VDB=0.0144;AF1=0.7741;CI95=0.6667,0.8333;DP4=1,0,7,3;MQ=16;FQ=-12.4;PV4=1,0.12,0
DP=14;VDB=0.0299;AF1=0.4904;CI95=0.3333,0.5;DP4=1,1,6,5;MQ=16;FQ=48.9;PV4=1,0.13,0.002
DP=19;VDB=0.0419;AF1=0.4997;CI95=0.5,0.5;DP4=1,1,12,5;MQ=12;FQ=48.3;PV4=1,1,1,0.087
DP=15;VDB=0.0371;AF1=0.1774;CI95=0.1667,0.3333;DP4=5,4,2,3;MQ=26;FQ=47.4;PV4=1,0.16,0.

```
> vcf = readVcf(system.file('extdata/1000G/1kg.chr22.vcf',
     package='BasicExomeExample'),genome='hg19')
```

4.2.1 Using Known Sites As a Quality Metric

One of the first tasks is to look at the overlap of the called variants with known variants. Here, I am using dbSNP, version 132, as a source of "known" variants. There are probably better choices. The current recommendation from the Broad is to use the latest HapMap VCF file, but I leave that as an exercise. The idea is that "known" variants represent true positives while the "novel" variants (those not in dbSNP) are enriched for false positives.

To get a copy of dbSNP, we can conveniently load one of the SNPlocs packages. After loading, I convert to a GRanges object. Note that I am ignoring the actual genotypes and just grabbing the base pair locations of the variants.

```
> library(SNPlocs.Hsapiens.dbSNP.20101109)
> tmpchr22 = getSNPlocs('ch22')
> dbsnpchr22 = GRanges(seqnames=rep('22',nrow(tmpchr22)),
```

```
    ranges=IRanges(start=tmpchr22$loc,width=1))
```

Finding the overlaps between the variant calls from the VCF file and dbsnp is straightforward.

```
> ovl = findOverlaps(rowData(vcf),dbsnpchr22)
```

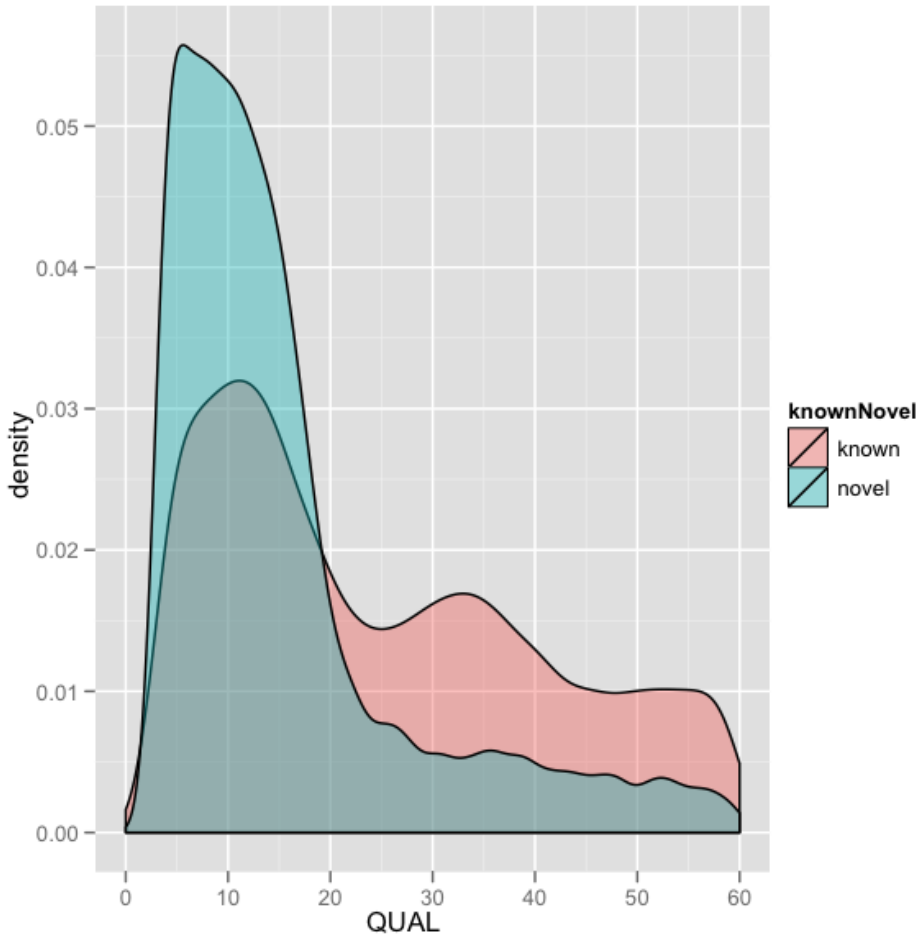
It may be instructive to look at the overlap between the capture regions and the variant calls.

```
> seqlevels(ag1000gssGR)=sub('chr','',seqlevels(ag1000gssGR))
> ovl2 = findOverlaps(rowData(vcf),ag1000gssGR)
```

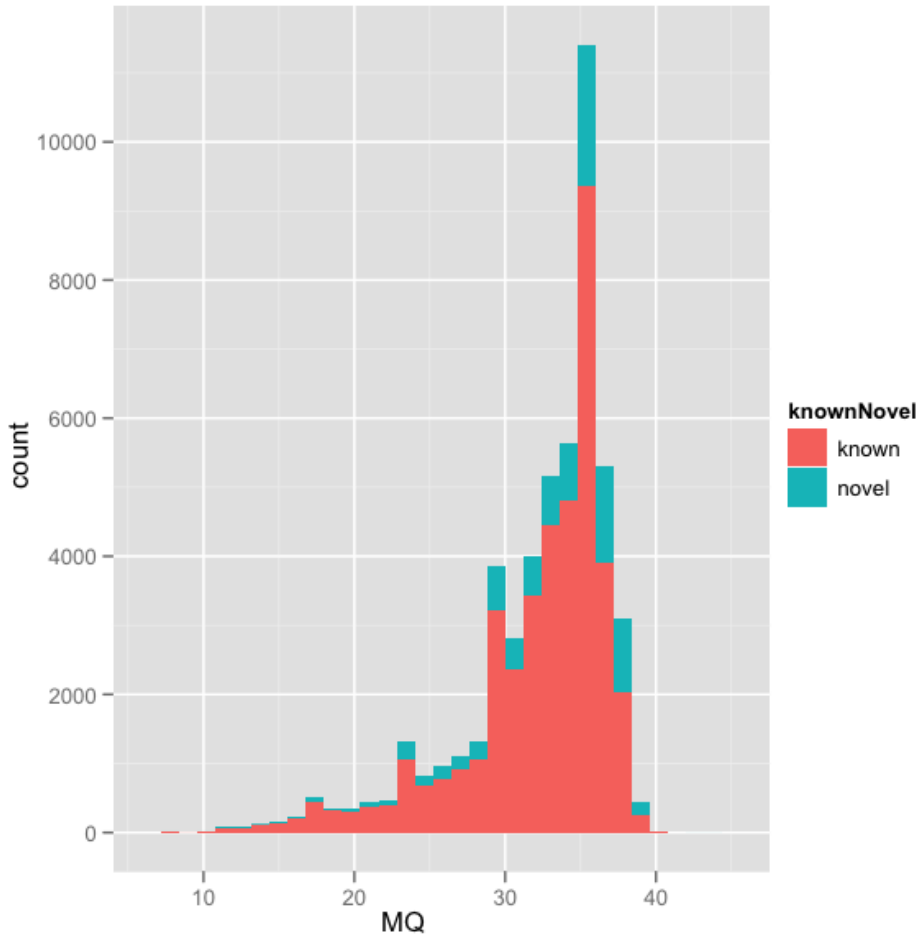
The next code block defines a data.frame of metrics from the VCF file. We will use the resulting data to examine a few quality metrics with regard to known (in dbSNP) and novel (not in dbSNP) variants.

```
> QD = elementMetadata(fixed(vcf))$QUAL/elementMetadata(info(vcf))$DP
> knownNovel = rep("novel",length(QD))
> knownNovel[queryHits(ovl)]="known"
> captured = rep(FALSE,length(QD))
> captured[queryHits(ovl2)]=TRUE
> pv4matrix = do.call(rbind,
    as.list(elementMetadata(info(vcf))$PV4))
> df1 = data.frame(location=start(rowData(vcf)),
    QD=QD,knownNovel=knownNovel,
    QUAL=elementMetadata(fixed(vcf))$QUAL,
    PV1=pv4matrix[,1],PV2=pv4matrix[,2],
    PV3=pv4matrix[,3],PV4=pv4matrix[,4],
    MQ=elementMetadata(info(vcf))$MQ,
    VDB=elementMetadata(info(vcf))$VDB,DP=elementMetadata(info(vcf))$DP)
```

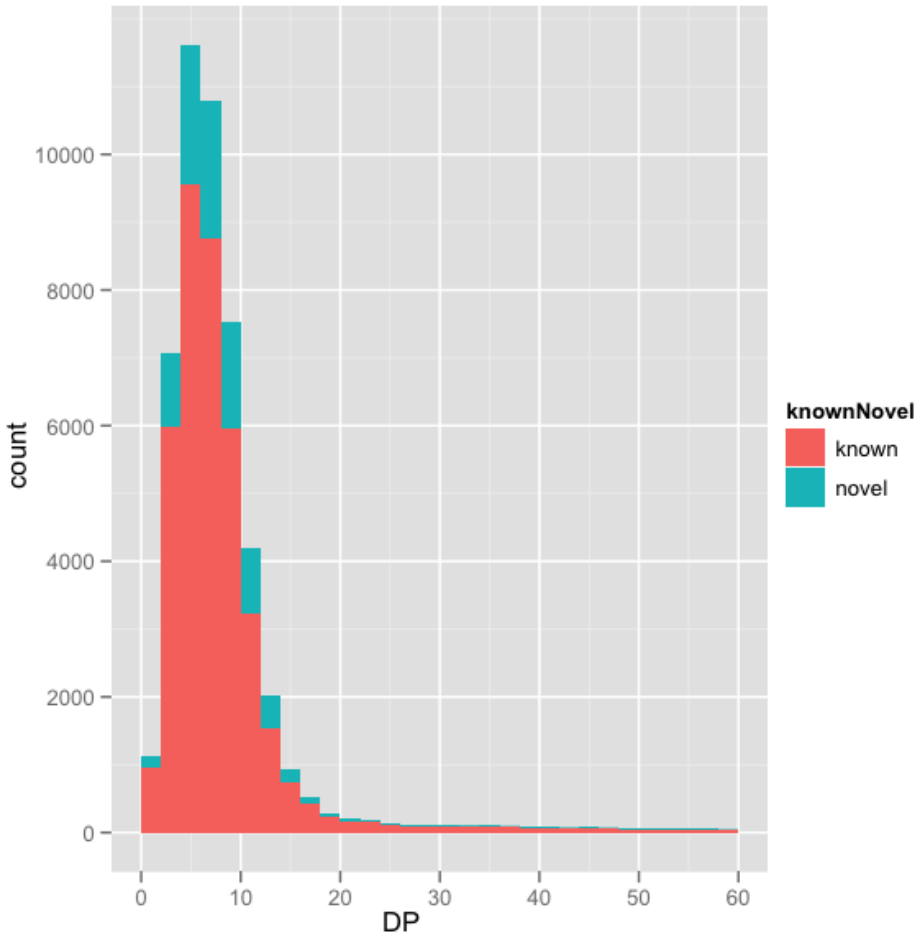
Now, to examine some relationships between the known and novel sites and some VCF file metrics. Variant quality should be important.



Mapping quality may also be important.



And check out read depth at the locus.



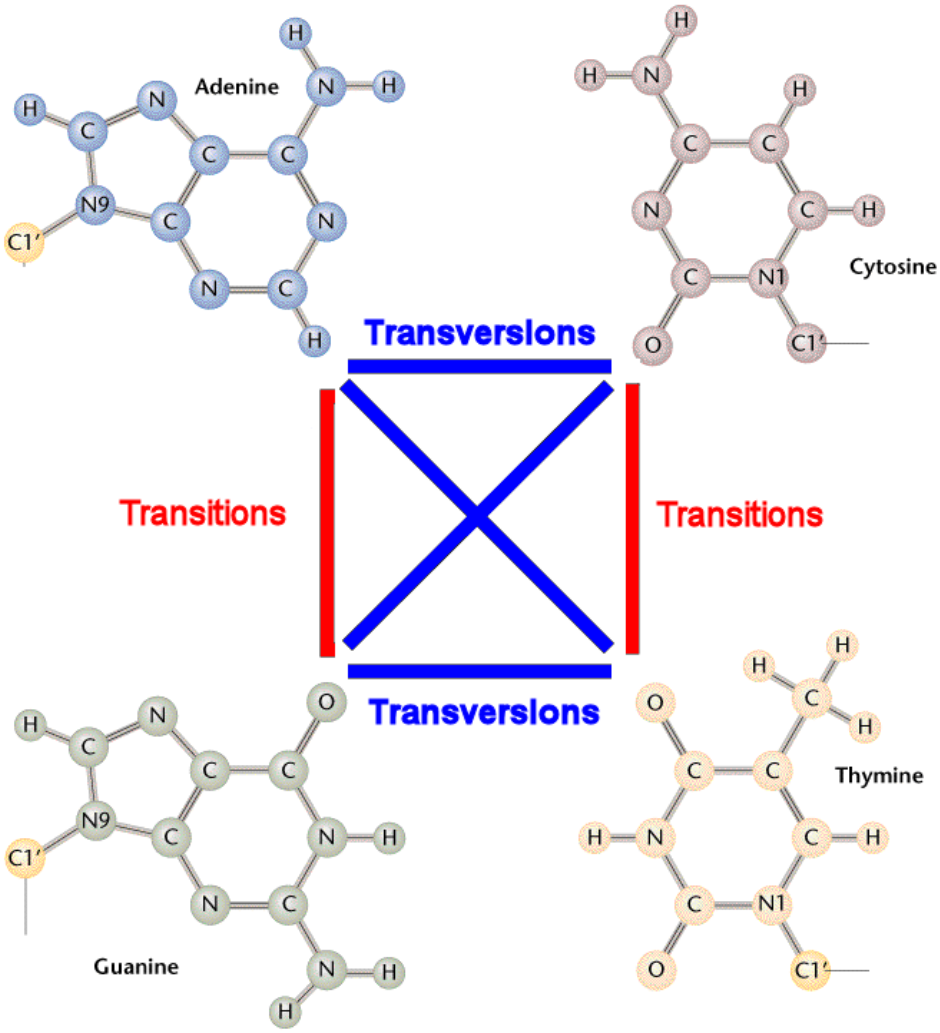
A number of other plots might be of interest. Feel free to try things out to see if you can find parameters that appear to separate known from novel variants.

```
> # some other plots that might be of interest
> ggplot(df1,aes(DP,fill=captured))+
    geom_density(alpha=0.5)+scale_x_continuous(limits=c(0,60))
> ggplot(df1,aes(PV1,fill=knownNovel))+
    geom_bar()+scale_x_continuous(limits=c(0,1))
> ggplot(df1,aes(PV2,fill=knownNovel))+
    geom_bar()+scale_x_continuous(limits=c(0,1))
> ggplot(df1,aes(PV3,fill=knownNovel))+
    geom_bar()+scale_x_continuous(limits=c(0,1))
```

```
> ggplot(df1,aes(PV4,fill=captured))+
    geom_density(alpha=0.4)+scale_x_continuous(limits=c(0,1))
```

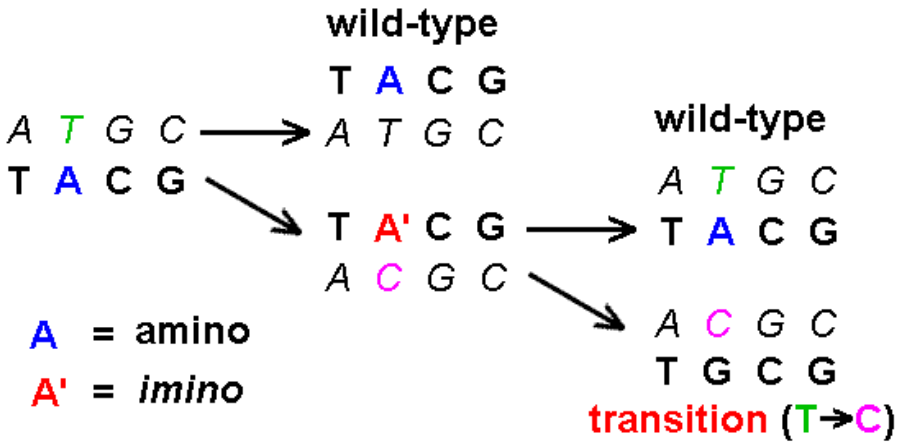
4.2.2 Using Transition/Transversion Ratio To Estimate False Positive Rate

DNA mutation at a single nucleotide can be classified as either a *transition* or a *transversion*. As can be seen from the figure, transversions are twice as likely as transitions *a priori*. Hence, the naive ratio of transitions (ti) to transversions (tv) is 0.5. Figures are from this site



However, a number of biological processes alter this naive ti/tv ratio so

that it is above 2 for natural variation across the human genome and 3.3 for the coding regions of the genome. A partial explanation is given by the next figure which describes the process of "wobble". A tautomeric shift (change from an amino group to an imino group) leads to a mispairing in replication and results in a transition mutation in the DNA. This process occurs naturally for each base and results in SNVs being introduced, all of which are transitions.



Now, let us assume that the observed ti/tv ratio is a mixture of true positive variants and false positive variants.

- If all observed variants are true positives, then we should observe approximately the ti/tv for the exome, which is reliably estimated at 3.3.
- If all observed variants are false positives, then we should observe the naive expected ti/tv, or 0.5.
- In reality, the observed ti/tv ratio is a mixture of true and false positives.

Given the observed ti/tv (titv_{obs}), the false positive ti/tv (titv_{fp}), the true positive ti/tv (titv_{tp}), and α, the proportion of SNVs that are true positives, we can write:

α (titv_{tp}) + (1 - α) (titv_{fp}) = titv_{obs}

Solving for 1 - α gives an estimate of the false positive rate.

FPR = 1 - (titv_{obs} - titv_{fp})/(titv_{tp} - titv_{fp})

Writing a function to calculate titv_{obs} is our first step to using ti/tv ratio as a quality metric.

```
> titv = function(object,sample=NULL) {
     # takes a VCF file as input
     # returns a list:
     #  table: the table of mutations from reference to alternate allele
     #  ti: The number of transition mutations observed
     #  tv: The number of transversion mutations observed
     #  ratio: The ratio of ti/tv
     ref = as.character(values(object)$REF)
     alt = as.character(values(object)$ALT)
     Nrows = (ref=="N" | alt=="N")
     snpsonly = (nchar(ref)==1 & nchar(alt)==1)
     ref = ref[!Nrows & snpsonly]
     alt = alt[!Nrows & snpsonly]
     if(!is.null(sample)) {
       onlySampleVars = (geno(object)$GT[,sample]!='0/0')
       ref = ref[onlySampleVars]
       alt = alt[onlySampleVars]
       }
     y = table(ref,alt)
     ti = y[2,4]+y[4,2]+y[1,3]+y[3,1]
     tv = y[1,2]+y[2,1]+y[1,4]+y[4,1]+y[2,3]+y[3,2]+y[3,4]+y[4,3]
     return(list(table=y,ti=ti,tv=tv,ratio=ti/tv))
     }

> vcf = read.delim(system.file('extdata/1000G/1kg.chr22.vcf',
     package='BasicExomeExample'),skip=17)
> vcfSNP = vcf[nchar(as.character(vcf$ALT))==1
     & nchar(as.character(vcf$REF))==1,]
> head(vcfSNP)
  X.CHROM      POS ID REF ALT  QUAL FILTER
1      22 16136566  .   A   G  4.86      .
2      22 16139004  .   G   T 15.50      .
3      22 16142489  .   T   A 13.60      .
4      22 16143946  .   A   G 14.20      .
5      22 16148996  .   C   G 15.90      .
6      22 16156691  .   G   A  5.70      .

1   DP=24;VDB=0.0535;AF1=0.4947;CI95=0.3333,0.6667;DP4=1,1,8,9;MQ=8;FQ=6.27;PV4=1,0.11
2 DP=25;VDB=0.0568;AF1=0.172;CI95=0.1667,0.3333;DP4=8,11,3,2;MQ=17;FQ=16.3;PV4=0.63,0.0
3      DP=14;VDB=0.0179;AF1=0.2182;CI95=0.1667,0.6667;DP4=3,4,2,3;MQ=16;FQ=14.9;PV4=1,0
```

```
4              DP=6;VDB=0.0198;AF1=0.4462;CI95=0.1667,0.6667;DP4=0,1,1,4;MQ=13;FQ=15;PV
5     DP=16;VDB=0.0176;AF1=0.4365;CI95=0.1667,0.6667;DP4=3,6,3,3;MQ=14;FQ=18.3;PV4=0.62
6         DP=18;VDB=0.0237;AF1=0.3173;CI95=0.1667,0.5;DP4=6,4,4,2;MQ=13;FQ=6.94;PV4=1,0
       FORMAT           NA19909           NA19914
1 GT:PL:DP:GQ 0/1:26,10,29:6:17 0/1:36,21,36:7:15
2 GT:PL:DP:GQ 0/1:50,0,68:11:46  0/0:0,27,27:9:30
3 GT:PL:DP:GQ     0/0:0,6,6:2:9     0/0:0,6,6:2:9
4 GT:PL:DP:GQ  0/1:44,9,44:3:35     0/0:0,3,3:1:3
5 GT:PL:DP:GQ  0/1:38,0,47:7:40     0/1:0,0,9:4:4
6 GT:PL:DP:GQ  0/1:43,9,43:3:33    0/1:7,0,25:9:8
           NA19916
1 0/1:14,4,17:6:12
2 0/0:0,12,12:4:16
3 0/1:47,0,56:8:44
4 0/1:21,6,21:2:15
5 0/1:13,0,22:4:15
6 0/0:0,12,12:4:12
> vcfGR = GRanges(seqnames=vcfSNP[,1],
     ranges=IRanges(start=vcfSNP$POS,width=1),
     REF=vcfSNP$REF,ALT=vcfSNP$ALT,QUAL=vcfSNP$QUAL)
> head(vcfGR)
GRanges with 6 ranges and 3 elementMetadata cols:
      seqnames               ranges strand |
         <Rle>            <IRanges>  <Rle> |
  [1]       22 [16136566, 16136566]      * |
  [2]       22 [16139004, 16139004]      * |
  [3]       22 [16142489, 16142489]      * |
  [4]       22 [16143946, 16143946]      * |
  [5]       22 [16148996, 16148996]      * |
  [6]       22 [16156691, 16156691]      * |
          REF      ALT      QUAL
     <factor> <factor> <numeric>
  [1]        A        G      4.86
  [2]        G        T     15.50
  [3]        T        A     13.60
  [4]        A        G     14.20
  [5]        C        G     15.90
  [6]        G        A      5.70
  ---
  seqlengths:
```

```
22
NA
```

We need another simple function to calculate the false positive rate (FPR) given an observed ti/tv ratio.

```
> # takes an observed ti/tv ratio, the expected titv for false positives,
> # and the expected titv for true positives; default is appropriate for exome data
> fpr = function(titv,titvFP=0.5,titvTP=3.3) {return(1-(titv-titvFP)/(titvTP-titvFP))}
```

To ask a couple of relevant questions, we need to limit our variants to the exome.

```
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> cdsRegions = cds(TxDb.Hsapiens.UCSC.hg19.knownGene)
> cdsRegions = reduce(cdsRegions)
> # We are being unsafe and ignoring sequence lengths
> seqlengths(cdsRegions) <- rep(NA,length(seqlengths(cdsRegions)))
> # For the purposes of this tutorial, we will try to stick
> # with the convention of NO 'chr' in chromosome names
> seqlevels(cdsRegions) <- sub('chr','',seqlevels(cdsRegions))

> # overlaps between cdsRegions and our variants
> ovl = findOverlaps(cdsRegions,vcfGR)
> vcf2 = vcfGR[subjectHits(ovl),]
```

And now, lets look at the quality of known and novel variants.

```
> ovl2 = findOverlaps(vcf2,dbsnpchr22)
> knownNovel=rep("novel",length(vcf2))
> knownNovel[queryHits(ovl2)]="known"
> titvKnown = titv(vcf2[knownNovel=='known',])
> titvNovel = titv(vcf2[knownNovel=='novel',])
> titvKnown$ratio
[1] 3.23348
> titvNovel$ratio
[1] 1.8125
```

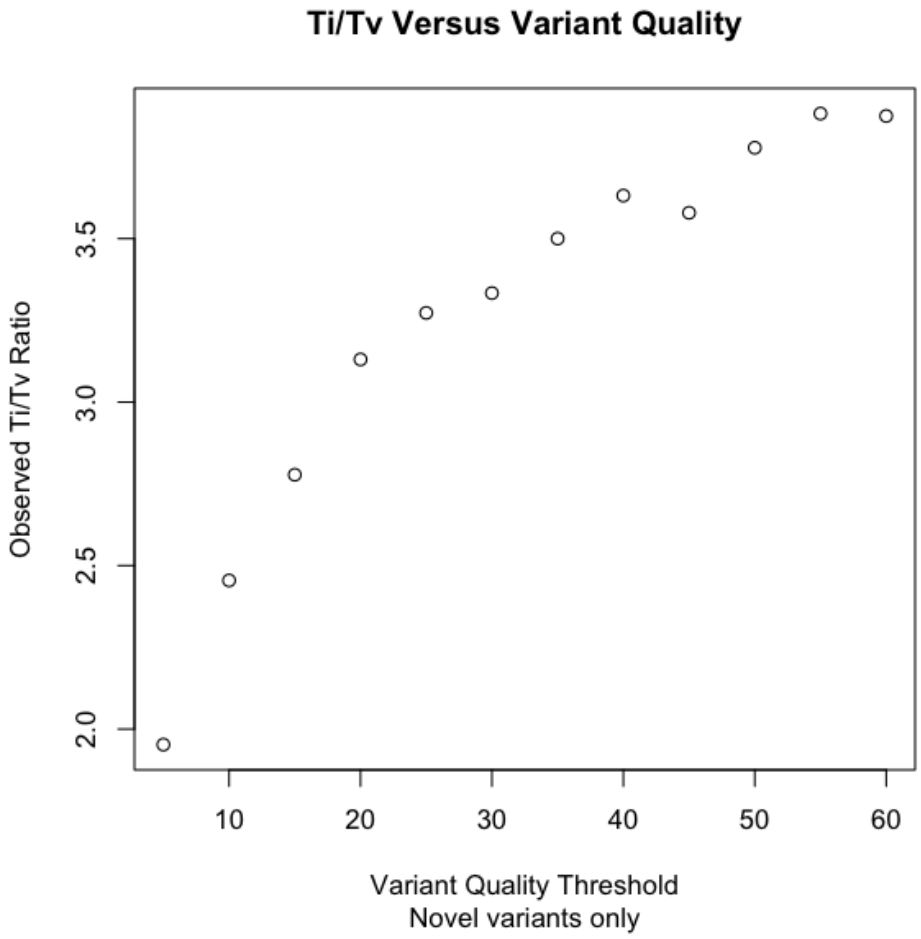
The ratio for known sites is close to that empirically expected for exonic variants while the ratio for novel sites is much lower. An estimate of the FPR is then:

```
> fpr(titvKnown$ratio)
[1] 0.02375708
> fpr(titvNovel$ratio)
[1] 0.53125
```

Our 1000 Genomes subset dataset shows very high quality at known variant sites. However, the "novel" sites appear to be enriched for false positives.

Can we improve on this FPR? We noticed above that the variant quality, QUAL, distinguished known and novel variants well. So, it makes sense that QUAL might enrich for true positives. Let's take a look.

```
> qual = values(vcf2)[["QUAL"]]
> summary(qual)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3.01   79.20  151.00  378.30  999.00  999.00
> qualSeq = seq(5,60,5)
> titvByQuality = sapply(qualSeq,function(threshold) {
     return(titv(vcf2[knownNovel=='novel' & qual>threshold])$ratio)})

> plot(qualSeq,titvByQuality,main="Ti/Tv Versus Variant Quality",
        sub = "Novel variants only",xlab="Variant Quality Threshold",
        ylab = "Observed Ti/Tv Ratio")
```



At this point, one can begin to play with various metrics to try to enrich both known and novel variant calls with true positives. This filtration process, which leads to higher specificity, must be balanced against loss in sensitivity. I leave construction of ROC curves as a further exercise, but note that one can use the known variants as an estimate of true positives.

Consider how you would use other sources of genomic annotation to further improve the quality and biological interpretability of these data.

4.3 NCI60 Data

The NCI60 data are analogous to the 1000 genomes dataset, but the variants in this dataset may be disease-associated. In fact, the remnants of the original disease etiology are still present in the data.

```
> vcf = readVcf(system.file('extdata/NCI60/NCI60.chr22.vcf',package='BasicExomeExample
```

4.3.1 Exercises

- See what you can learn about the biology of melanoma by examining the transition/transversion matrix.
- Make a GRanges object from data from the COSMIC database to look for variant overlaps with known cancer variants.

5 Using SRAdb to Interact With the IGV Browser

Working with sequence data is often best done interactively in a genome browser, a task not easily done from R itself. We have found the Integrative Genomics Viewer (IGV) a high-performance visualization tool for interactive exploration of large, integrated datasets, increasing usefully for visualizing sequence alignments. In SRAdb, functions startIGV, load2IGV and load2newIGV provide convenient functionality for R to interact with IGV. IGV offers a remort control port that allows R to communicate with IGV. The current command set is fairly limited, but it does allow for some IGV operations to be performed in the R console. To utilize this functionality, be sure that IGV is set to allow communication via the "enable port" in the advanced preferences in IGV.

```
> if(!require(SRAdb)) {
     library(BiocInstaller)
     biocLite('SRAdb')
     }
> # start IGV on your computer
> startIGV('mm') # 1.2GB memory used
> # connect to IGV socket
> sock = IGVsocket()
> # Change IGV genome to match our BAM files
> IGVgenome(sock,'b37')
> # Load our example BAM files into IGV
> exampleBams = file.path(system.file('extdata/1000G/bam/',package='BasicExomeExample')
> IGVload(sock,exampleBams)
> gotoVCFRecord <- function(vcfObject,sock,idx=1) {
     # takes a VCF object, an IGV socket, and the index
     # of the variant of interest
```

```
     rd = rd = rowData(vcf)
     IGVgoto(sock,sprintf("%s:%d-%d",seqnames(rd)[idx],start(rd)[idx],end(rd)[idx]))
     }
> vcf = readVcf(system.file('extdata/1000G/1kg.chr22.vcf',package='BasicExomeExample'),
> indelSites = which(values(info(vcf))[['INDEL']])
> gotoVCFRecord(vcf,sock,indelSites[2])
> # and take a pretty picture
> IGVsnapshot(sock)
```

6 sessionInfo

```
> sessionInfo()
R Under development (unstable) (2012-01-19 r58141)
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] grid      stats     graphics  grDevices
[5] utils     datasets  methods   base

other attached packages:
 [1] SNPlocs.Hsapiens.dbSNP.20101109_0.99.6
 [2] VariantAnnotation_1.1.55
 [3] ggplot2_0.8.9
 [4] proto_0.3-9.2
 [5] reshape_0.8.4
 [6] plyr_1.7.1
 [7] BSgenome.Hsapiens.UCSC.hg19_1.3.17
 [8] BSgenome_1.23.2
 [9] Gviz_0.99.0
[10] Rsamtools_1.7.37
[11] Biostrings_2.23.6
[12] TxDb.Hsapiens.UCSC.hg19.knownGene_2.6.2
[13] GenomicFeatures_1.7.28
[14] AnnotationDbi_1.17.22
[15] Biobase_2.15.3
[16] rtracklayer_1.15.7
```

```
[17] GenomicRanges_1.7.28
[18] IRanges_1.13.26
[19] BiocGenerics_0.1.7
[20] ascii_2.1

loaded via a namespace (and not attached):
 [1] biomaRt_2.11.1     bitops_1.0-4.1
 [3] DBI_0.2-5          digest_0.5.1
 [5] lattice_0.20-0     Matrix_1.0-3
 [7] RColorBrewer_1.0-5 RCurl_1.91-1
 [9] RSQLite_0.11.1     snpStats_1.5.4
[11] splines_2.15.0     survival_2.36-12
[13] tools_2.15.0       XML_3.9-4
[15] zlibbioc_1.1.1
```