Basics

Contents

1 General terminology **2**

2 Markers **2**

3 Data **2**

4 Public databases **3**

5 Data management **3**

5.1 Python . . . 3
5.2 GRanges . . . 6
5.3 DNAStrings . . . 16
5.4 Data classes . . . 18
5.5 Expression array archives . . . 23
5.6 Obtaining the ExpressionSet for an EMBL series . . . 23
5.7 Storing data . . . 24
5.8 Reference genomes . . . 25
5.9 Annotations . . . 26

1 General terminology

Term	Description
Genome	The entire DNA content of an individual's chromosome the the articular
Marker	A measurable variation (polymorphism) at the DNA level
Locus	A genomic region that explains phenotypic differences due to genetic polymorphisms
Gene	A basic unit of heredity and a sequence of nucleotides in DNA that encodes the synthesis of a gene product
Isoform	Each mRNA maps to a gene but one gene can be mapped by numerous mRNAs. A gene encodes different mRNA transcripts, or isoforms
Variant	A locus in the genome where there are differences between individuals
Allele	One of the possible bases/sequences that can occur at the variant
SNP	Single nucleotide polymorphism: only one nucleotide substitution within a short DNA sequence A type of variant
SNP array	Short DNA sequences, with their variant nucleotides at their ends, who constitute probes
VNTRs	Variable number of tandem repeats: multiple copies of the same sequence of DNA A type of variant
Haplotype	Blocks of correlated SNPs that tend to be inherited together (e.g., HLA alleles in the major histocompatibility complex)
QTL	A locus that correlates with the variation of a quantitative trait in the phenotype of a population
Linkage-Disequilibrium	It is a measure of the statistical association between the allele frequencies at two different loci in a population
Transcriptome	The complete set of RNA transcripts that are produced by the genome
Transcript	Refers to which exons are spliced to form the RNA molecule
Epigenome	Non-sequence modifications of DNA that are heritable (e.g., methylation)
Exposome	The measure of all the exposures of an individual in a lifetime

2 Markers

In general terms, the markers are used to identify regions harboring quantitative trait loci (QTL).

Types:

- Restriction fragment length polymorphism (RFLP)
- Random amplification of polymorphic DNA (RAPD)
- Amplified fragment length polymorphism (AFLP)
- Single-stranded conformation polymorphism (SSCP)
- Copy number variation (CNV)
- Microsatellites:
  - A type of VNTR where the length of the repeating unit is less than five base pairs
  - A microsatellite can have many alleles (10-30)
  - They are widely used in QTL (quantitative trait locus) mapping projects
- Single nucleotide polymorphism (SNP):
  - 0: homozygous (AA, for the minor allele A)
  - 1: heterozygous (AC/CA, for the minor allele A)
  - 2: variant homozygous (CC, for the minor allele A)

3 Data

- Transcriptomic data:
  - Generation methods:

        * Microarray method: mRNA is collected and converted back to cDNAs which are then dyed and hybridized on a chip containing probes
        * RNA-seq method: the application of high throughput sequencing to RNA
- Epigenomic data:
    - Methylomic data: bounded by the number of CpG sequences in the genome
    - Others
- Genomic data:
    - Strands
        * 5' to 3' (+)
        * 3' to 5' (-)

4 Public databases

- dbGaP:
    - genomic, epigenomic, somatic mutations, transcriptomic and microbiomic data, with associated phenotypes
    - Permission is required
- EGA: same as above
- GEO:
    - Only transcriptomic data from microarray and RNA-seq
    - Data can be directly retrieved with Bioconductor
- 1000 Genomes: genomic data from the sequencing of 2,504 individuals from 26 different ancestries
- GTEx: collected transcriptomic data for 714 donors, 635 of which were also genotyped
- TCGA: high-throughput DNA and RNA sequencing, SNP, DNA methylation, and reverse-protein arrays of 33 cancers

5 Data management

5.1 Python

5.1.1 Read, count, and reverse complement

Read genomic data:

```
def readGenome(filename):
    genome = ''
    with open(filename, 'r') as f:
        for line in f:
            if not line[0] == '>':
                genome += line.rstrip()
    return genome

lambda_virus_genome = readGenome('Data/lambda_virus.txt')
print(lambda_virus_genome[:20])
```

```
## GGGCGGCGACCTCGCGGGTT
```

Number of bases in the strand:

```
def count(genome):
    counts = {'A': 0, 'C': 0, 'G': 0, 'T': 0}
    for base in genome:
        counts[base] += 1
    print(counts)
```

```
count(lambda_virus_genome)
```

```
## {'A': 12334, 'C': 11362, 'G': 12820, 'T': 11986}
```

Get the antisense strand:

```
def reverseComplement(senseStrand):
    complement = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C', 'N': 'N'}
    antisenseStrand = ''
    for base in senseStrand:
        antisenseStrand = complement[base] + antisenseStrand
    return antisenseStrand

lambda_virus_genomeAntisense = reverseComplement(lambda_virus_genome)
print(lambda_virus_genomeAntisense[:20])
```

```
## CGTAACCTGTCGGATCACCG
```

5.1.2 Quality scores

Read genome reads (sequences and quality values):

```
def readReads(filename):
    sequences = []
    qualities = []
    with open(filename, 'r') as f:
        while True:
            f.readline()
            seq = f.readline().rstrip()
            f.readline()
            qual = f.readline().rstrip()
            if len(seq) == 0:
                break
            sequences.append(seq)
            qualities.append(qual)
    return sequences, qualities

human_seqs, human_quals = readReads('Data/SRR835775_1.first1000.fastq')
print(human_seqs[1])
```

```
## TAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACC
```

```
print(human_quals[1])
```

```
## CCCFFFFFGHHGHJJJJJIJGIIJJJJJJJIJIJJJJJFJJFGIIIIH=CBFCF=CCEG)=>EHB2@@DEC>;;?=;(=?BBD?59?BA############
```

Convert quality values to numbers:

```
def phred33ToQ(qual):
    return ord(qual) - 33
```

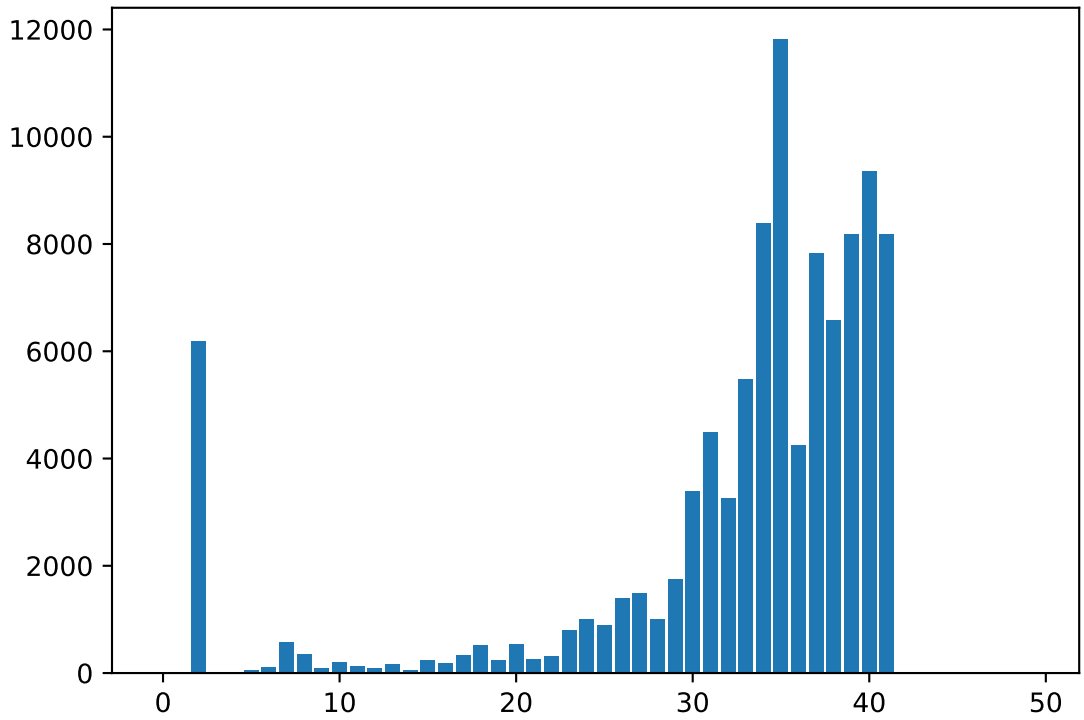
Create a histogram of the frequency of qualities:

```
import matplotlib.pyplot as plt
def createHist(qualities):
    hist = [0] * 50
    for qual in qualities:
        for phred in qual:
```

```
            q = phred33ToQ(phred)
            hist[q] += 1
    return hist
h = createHist(human_quals)
plt.bar(range(len(h)), h)
```

```
## <BarContainer object of 50 artists>
```

```
plt.show()
```

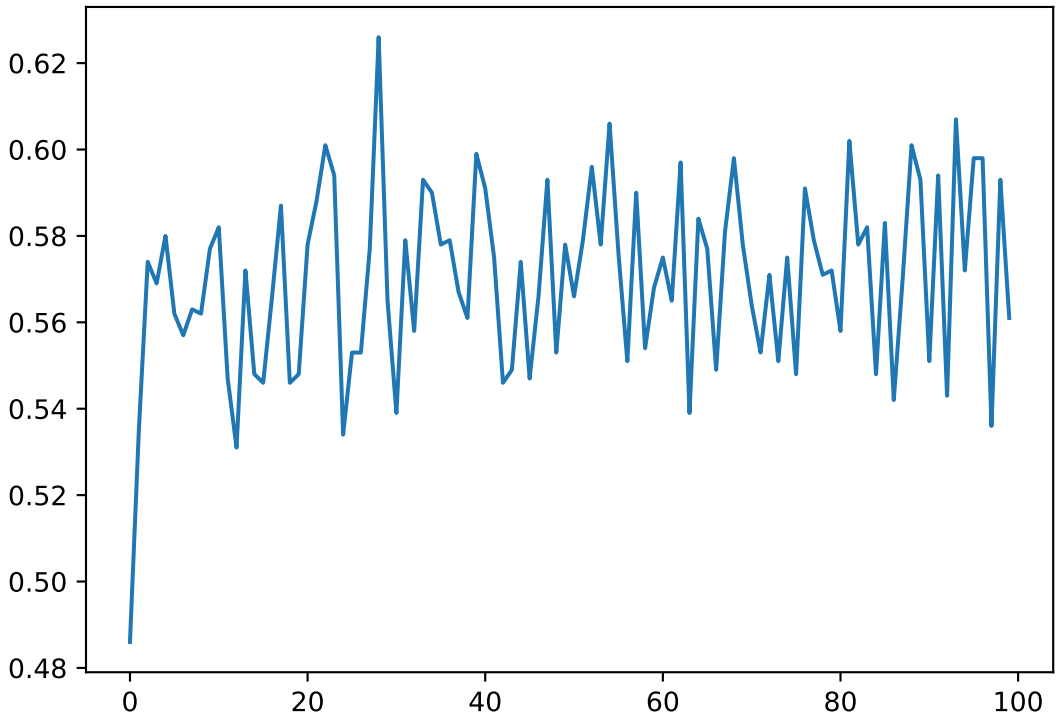


5.1.3 Find GCs

Find GCs by position:

```
def findGCByPos(reads):
    gc = [0] * 100
    totals = [0] * 100
    for read in reads:
        for base in range(len(read)):
            if read[base] == 'C' or read[base] == 'G':
                gc[base] += 1
            totals[base] += 1
    for i in range(len(gc)):
        if totals[i] > 0:
            gc[i] /= float(totals[i])
    return gc
```

```
gc = findGCByPos(human_seqs)
plt.plot(range(len(gc)), gc)
plt.show()
```



5.2 GRanges

Let's start by creating a sample gene:

```
# Import packages
library(GenomicRanges)

# Create an example gene
gen1 <- GRanges("chr1", IRanges(1000001, 1001000), strand = "+")
```

Now, let's check its specification:

```
# Get gene spec
## Start
start(gen1)
```

```
## [1] 1000001
```

```
## End
end(gen1)
```

```
## [1] 1001000
```

```
## Width
width(gen1)
```

```
## [1] 1000
```

```
## Strand
strand(gen1)
```

```
## factor-Rle of length 1 with 1 run
##   Lengths: 1
##   Values : +
## Levels(3): + - *
```

```
## The 'metadata columns', any information stored alongside each range
mcols(gen1)
```

```
## DataFrame with 1 row and 0 columns
```

```
## IRanges
ranges(gen1)
```

```
## IRanges object with 1 range and 0 metadata columns:
##           start       end     width
##       <integer> <integer> <integer>
##   [1]   1000001   1001000      1000
```

```
## The chromosomes for each ranges
seqnames(gen1)
```

```
## factor-Rle of length 1 with 1 run
##   Lengths:    1
##   Values : chr1
## Levels(1): chr1
```

```
## The possible chromosomes
seqlevels(gen1)
```

```
## [1] "chr1"
```

```
## The lengths for each chromosome
seqlengths(gen1)
```

```
## chr1
##   NA
```

5.2.1 Intra-range methods

Affects ranges independently.

function	description
shift	moves left/right
narrow	narrows by relative position within range
resize	resizes to width, fixing start for +, end for -
flank	returns flanking ranges to the left +, or right -
promoters	similar to flank
restrict	restricts ranges to a start and end position
trim	trims out of bound ranges
+/-	expands/contracts by adding/subtracting fixed amount

function	description
*	zooms in (positive) or out (negative) by multiples

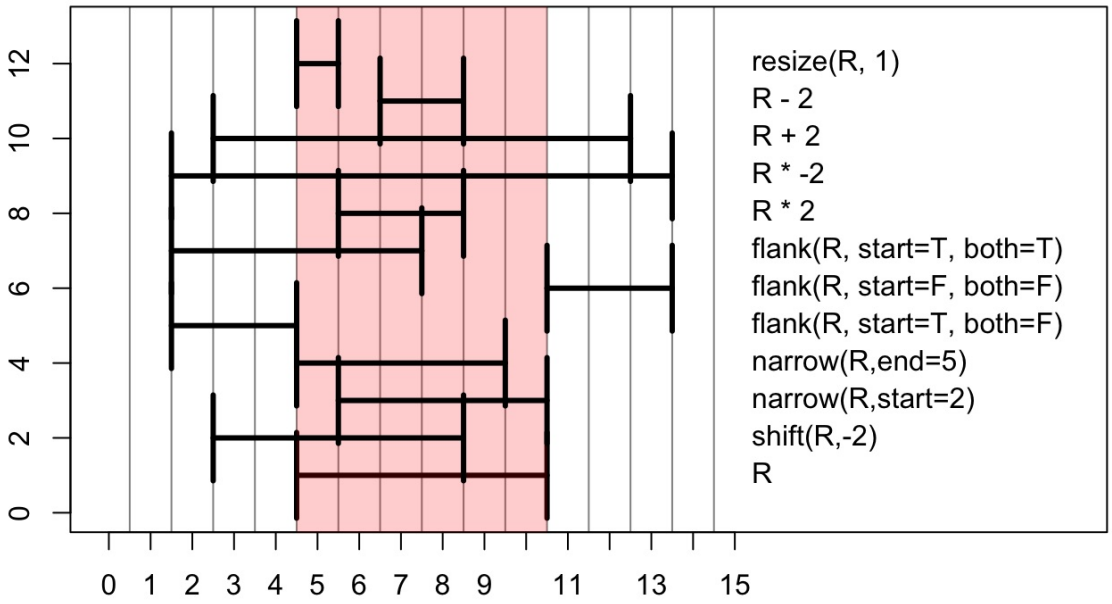


Figure 1: Intra-range methods 1

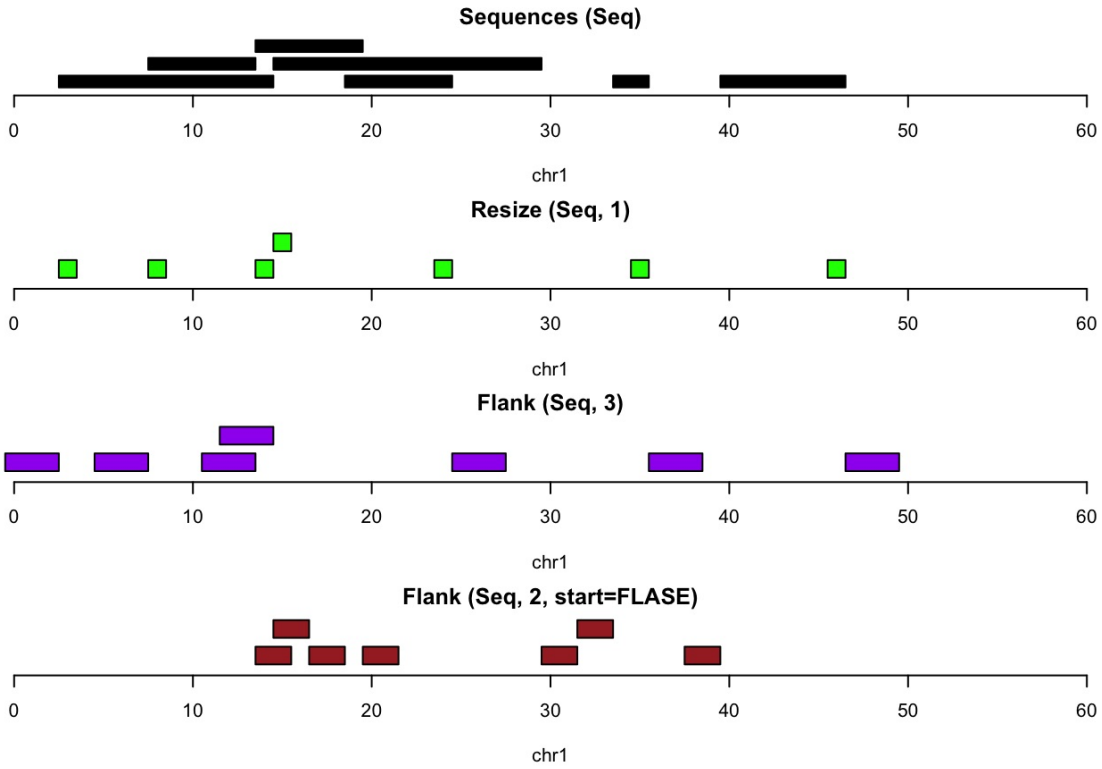


Figure 2: Intra-range methods 2

5.2.2 Inter-range methods

Affects ranges as a group.

function	description
range	one range, leftmost start to rightmost end
reduce	cover all positions with only one range
gaps	uncovered positions within range
disjoin	breaks into discrete ranges based on original starts/ends

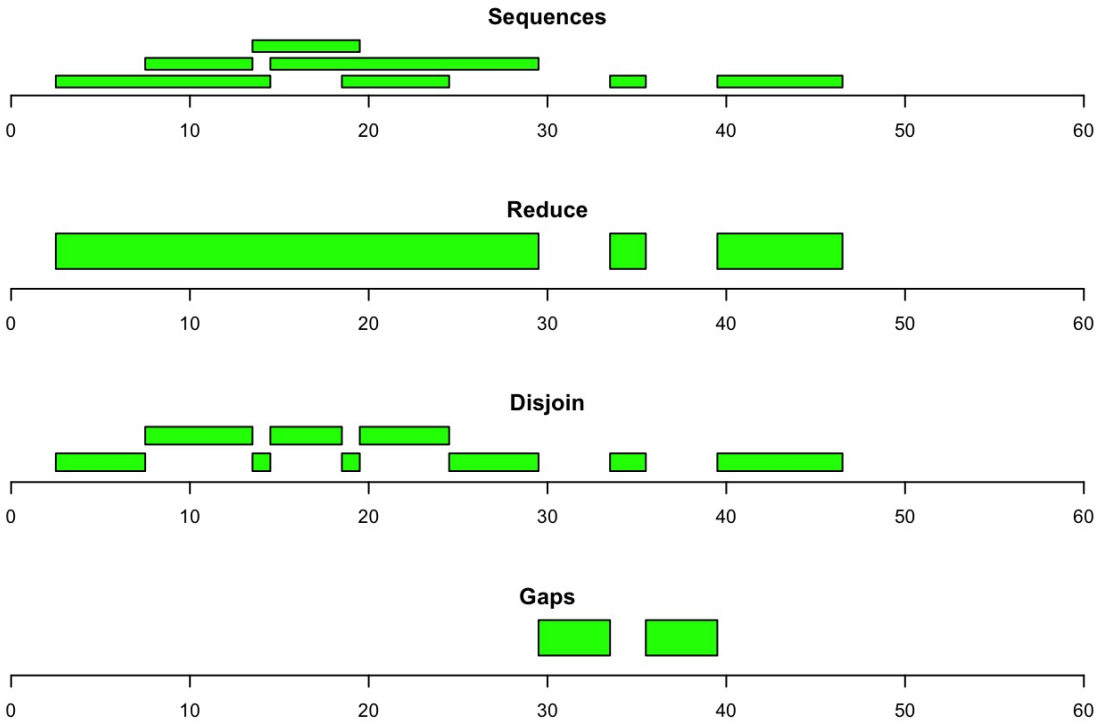
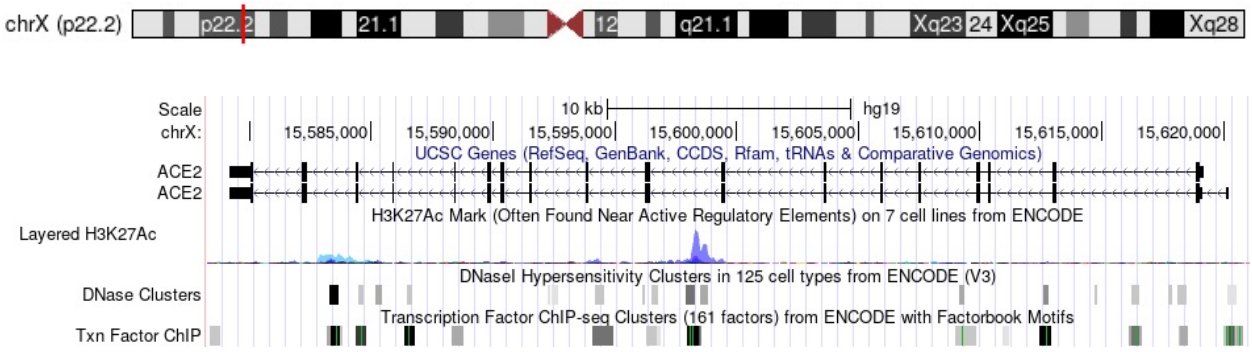


Figure 3: Inter-range methods

5.2.3 ENCODE project experiment

Scenario: A ChIP-seq experiment to identify the ESRRA nuclear protein binding sites in HepG2 cell line (of liver origin) and GM12878 cell line (of lymphoblastoid origin).

ESRRA: Estrogen Related Receptor Alpha



Let's start by importing and preprocessing the data:

```
# Load data
library(ERBS)
data(HepG2)
data(GM12878)

# Check data
HepG2[1:5, 4:5]
```

```
## GRanges object with 5 ranges and 2 metadata columns:
##       seqnames              ranges strand | signalValue    pValue
##          <Rle>           <IRanges>  <Rle> |   <numeric> <numeric>
##   [1]     chr2   20335378-20335787      * |      58.251    75.899
##   [2]    chr20       328285-329145      * |      10.808    69.685
##   [3]     chr6 168135432-168136587      * |      17.103    54.311
##   [4]    chr19     1244419-1245304      * |      12.427    43.855
##   [5]    chr11   64071828-64073069      * |      10.850    40.977
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# Extract data as Rle data
## Extract seqnames (chromosomes)
chr <- seqnames(HepG2)[1:5,]
## Extract ranges
r <- ranges(HepG2)[1:5,]
## Extract values
vals <- values(HepG2)[1:5,]

# Convert extracted data to character
chr <- as.character(chr)
chr
```

```
## [1] "chr2"  "chr20" "chr6"  "chr19" "chr11"
```

```
r <- as.data.frame(r)
r
```

```
##       start       end width
## 1  20335378  20335787   410
## 2    328285    329145   861
## 3 168135432 168136587  1156
## 4   1244419   1245304   886
## 5  64071828  64073069  1242
```

```
vals <- as.data.frame(vals)
vals[, 4:7]
```

```
##   signalValue pValue       qValue peak
## 1      58.251 75.899 6.143712e-72  195
## 2      10.808 69.685 5.028065e-66  321
## 3      17.103 54.311 7.930665e-51  930
## 4      12.427 43.855 1.359756e-40  604
## 5      10.850 40.977 7.333863e-38  492
```

```
# Cross-tab
table(chr)
```

```
## chr
## chr11 chr19  chr2 chr20  chr6
##     1     1     1     1     1
```

```
# Subset operation
chr20 <- HepG2[seqnames(HepG2) == "chr20", ]
chr20[1:5, 7]
```

```
## GRanges object with 5 ranges and 1 metadata column:
##       seqnames            ranges strand |      peak
```

```
##          <Rle>         <IRanges>  <Rle> | <integer>
##   [1]    chr20     328285-329145      * |       321
##   [2]    chr20 22410891-22411863      * |       660
##   [3]    chr20 56039583-56040249      * |       315
##   [4]    chr20 16455811-16456232      * |       199
##   [5]    chr20   3140243-3140774      * |       315
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# Re-order the data by seqnames and ranges
HepG2_ordered = HepG2[order(HepG2),]
HepG2_ordered[1:5, 7]
```

```
## GRanges object with 5 ranges and 1 metadata column:
##       seqnames            ranges strand |      peak
##          <Rle>         <IRanges>  <Rle> | <integer>
##   [1]     chr1   9294157-9294957      * |       182
##   [2]     chr1 15911325-15911873      * |       264
##   [3]     chr1 16339152-16339862      * |       332
##   [4]     chr1 26146200-26147004      * |       337
##   [5]     chr1 27264576-27265423      * |       191
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

Now, we are going to find binding sites and extract their ranges:

```
# Find binding sites common to both HepG2 and GM12878
res = findOverlaps(HepG2, GM12878)
res[1]
```

```
## Hits object with 1 hit and 0 metadata columns:
##       queryHits subjectHits
##       <integer>   <integer>
##   [1]         1          12
##   -------
##   queryLength: 303 / subjectLength: 1873
```

```
HepG2[1]
```

```
## GRanges object with 1 range and 7 metadata columns:
##       seqnames            ranges strand |      name     score       col
##          <Rle>         <IRanges>  <Rle> | <numeric> <integer> <logical>
##   [1]     chr2 20335378-20335787      * |        NA         0      <NA>
##       signalValue    pValue      qValue      peak
##         <numeric> <numeric>   <numeric> <integer>
##   [1]      58.251    75.899 6.14371e-72       195
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
GM12878[12]
```

```
## GRanges object with 1 range and 7 metadata columns:
##       seqnames            ranges strand |      name     score       col
##          <Rle>         <IRanges>  <Rle> | <numeric> <integer> <logical>
##   [1]     chr2 20335155-20336108      * |        16         0      <NA>
##       signalValue    pValue    qValue      peak
##         <numeric> <numeric> <numeric> <integer>
##   [1]       55.95   223.815        32       438
```

```
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# Ranges from the query for which we found a hit in the subject
index = queryHits(res)
erbs = HepG2[index,]
erbs[1:5]
```

```
## GRanges object with 5 ranges and 7 metadata columns:
##       seqnames            ranges strand |      name     score       col
##          <Rle>         <IRanges>  <Rle> | <numeric> <integer> <logical>
##   [1]     chr2 20335378-20335787      * |        NA         0      <NA>
##   [2]    chr20     328285-329145      * |        NA         0      <NA>
##   [3]    chr19   1244419-1245304      * |        NA         0      <NA>
##   [4]    chr11 64071828-64073069      * |        NA         0      <NA>
##   [5]     chr2 16938364-16938840      * |        NA         0      <NA>
##       signalValue    pValue      qValue      peak
##         <numeric> <numeric>   <numeric> <integer>
##   [1]      58.251    75.899 6.14371e-72       195
##   [2]      10.808    69.685 5.02806e-66       321
##   [3]      12.427    43.855 1.35976e-40       604
##   [4]      10.850    40.977 7.33386e-38       492
##   [5]      12.783    38.004 5.36029e-35       255
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# Extract only the ranges
erbs = granges(erbs)
erbs[1:5]
```

```
## GRanges object with 5 ranges and 0 metadata columns:
##       seqnames            ranges strand
##          <Rle>         <IRanges>  <Rle>
##   [1]     chr2 20335378-20335787      *
##   [2]    chr20     328285-329145      *
##   [3]    chr19   1244419-1245304      *
##   [4]    chr11 64071828-64073069      *
##   [5]     chr2 16938364-16938840      *
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

5.2.3.1 Acquisition of gene transcription start sites

Using the binding sites, we want to know which genes are preceded by each binding site.

Note: This information is strand-based, meaning that "preceding" depends on the direction of which we read the genome.

```
# Define human genes
library(Homo.sapiens)
ghs = genes(Homo.sapiens)
ghs[1:5]
```

```
## GRanges object with 5 ranges and 1 metadata column:
##         seqnames              ranges strand |          GENEID
##            <Rle>           <IRanges>  <Rle> | <CharacterList>
##       1    chr19   58858172-58874214      - |               1
##      10     chr8   18248755-18258723      + |              10
```

```
##     100    chr20   43248163-43280376      - |            100
##    1000    chr18   25530930-25757445      - |           1000
##   10000     chr1 243651535-244006886      - |          10000
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# First three ERBS ranges
erbs[1:3]
```

```
## GRanges object with 3 ranges and 0 metadata columns:
##       seqnames            ranges strand
##          <Rle>         <IRanges>  <Rle>
##   [1]     chr2 20335378-20335787      *
##   [2]    chr20     328285-329145      *
##   [3]    chr19   1244419-1245304      *
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# For each range in ERBS, find the closest preceding genes in GHS
index = precede(erbs, ghs)

# First three closest preceding genes in GHS to ERBS ranges
ghs[index[1:3]]
```

```
## GRanges object with 3 ranges and 1 metadata column:
##         seqnames            ranges strand |          GENEID
##            <Rle>         <IRanges>  <Rle> | <CharacterList>
##    9741     chr2 20232411-20251789      - |            9741
##   57761    chr20     361308-378203      + |           57761
##   90007    chr19   1248552-1259142      + |           90007
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# Distance between binding sites and nearest preceding genes
distance(erbs, ghs[index])[1:3]
```

```
## [1] 83588 32162  3247
```

Now, let's find the transcription start site nearest to each binding site:

```
# Get all TSS of human genome
tssgr = resize(ghs, 1)
tssgr[1:3]
```

```
## GRanges object with 3 ranges and 1 metadata column:
##       seqnames    ranges strand |          GENEID
##          <Rle> <IRanges>  <Rle> | <CharacterList>
##     1    chr19  58874214      - |               1
##    10     chr8  18248755      + |              10
##   100    chr20  43280376      - |             100
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# Distance between erbs binding sites and nearest TSS
d = distanceToNearest(erbs, tssgr)
d[1:3]
```

```
## Hits object with 3 hits and 1 metadata column:
##       queryHits subjectHits |  distance
```

```
##       <integer>   <integer> | <integer>
##   [1]         1       22817 |     83588
##   [2]         2       19883 |       914
##   [3]         3       13475 |      2669
##   -------
##   queryLength: 75 / subjectLength: 23056
```

```
dists = values(d)$distance

# Get the TSS of genes that are less than 1000 bases away from the erbs BS
index = subjectHits(d)[dists < 1000]
index[1:3]
```

```
## [1] 19883  6316 18488
```

```
tssgr[index, ]
```

```
## GRanges object with 41 ranges and 1 metadata column:
##          seqnames    ranges strand |          GENEID
##             <Rle> <IRanges>  <Rle> | <CharacterList>
##    80023    chr20    327370      + |           80023
##     2101    chr11  64073044      + |            2101
##     7086     chr3  53290130      - |            7086
##     5478     chr7  44836241      + |            5478
##   286262     chr9 140095163      - |          286262
##      ...      ...       ...    ... .             ...
##    55090    chr17  17380300      + |           55090
##    11165     chr6  34360457      - |           11165
##    56181     chr1  26146397      + |           56181
##   347734     chr6  44225283      - |          347734
##     2948     chr1 110198698      + |            2948
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# Extract those genes IDs
keys = as.character(values(tssgr[index])$GENEID)
keys[1:3]
```

```
## [1] "80023" "2101"  "7086"
```

```
# Extract the symbole and names of those IDs
res = select(
  Homo.sapiens,
  keys = keys,
  columns = c("SYMBOL", "GENENAME"),
  keytype = "GENEID"
)
res[1:3,]
```

```
##   GENEID                       GENENAME SYMBOL
## 1  80023                    neurensin 2  NRSN2
## 2   2101 estrogen related receptor alpha  ESRRA
## 3   7086                   transketolase    TKT
```

These genes are the genes that are:

- common between both HepG2 and GM12878 cell lines;
- their TSS are close (<1000 bases far from) to the ESRRA nuclear protein BS;

- as a result, most probably activated by estrogen via Estrogen Related Receptor Alpha (ESRRA).

Conclusion: So basically, we went from the protein binding sites of a specific nuclear protein to the genes that are activated via it.

5.3 DNAStrings

Let's start by creating a sample DNA string:

```
library(Biostrings)

# Define a DNAString
dna <- DNAString("TCGAGCAAT")
dna
```

```
## 9-letter DNAString object
## seq: TCGAGCAAT
```

In *DNAStrings* there are the four bases (ACGT), the wild card or unknown base (N), and the dash representing a gap (-).

```
# Create a set of DNAStrings (DNAStringSet)
set1 <- DNAStringSet(c("TCA", "AAATCG", "ACGTGCCTA", "CGCGCA", "GTT", "TCA"))
set1
```

```
## DNAStringSet object of length 6:
##     width seq
## [1]     3 TCA
## [2]     6 AAATCG
## [3]     9 ACGTGCCTA
## [4]     6 CGCGCA
## [5]     3 GTT
## [6]     3 TCA
```

```
# Extract a base from a seq
set1[[1]][2]
```

```
## 1-letter DNAString object
## seq: C
```

```
# Number of DNAStrings in the set
length(set1)
```

```
## [1] 6
```

```
# Width of each DNAString in the set
width(set1)
```

```
## [1] 3 6 9 6 3 3
```

Some other functions:

function	description
duplicated	detect which sequences are duplicated
unique	keep only unique sequences
sort	sort sequences alphabetically
letterFrequency	find the freq of one character
dinucleotideFrequency	find the freq of all dinucleotides
trinucleotideFrequency	find the freq of all trinucleotides
countPattern	find the freq of a pattern

function	description
matchPattern	find the locations of a pattern
reverseComplement	find the reverse complement of a DNAString
translate	find the amino acid translation of a DNAString

Let's check a specific pattern in our DNAString sample:

```
# Check for a pattern in a DNAString
dna_seq <- DNAString("ATGCGCGCGGCTA")
matchPattern("CGC", dna_seq)
```

```
## Views on a 13-letter DNAString subject
## subject: ATGCGCGCGGCTA
## views:
##       start end width
##   [1]     4   6     3 [CGC]
##   [2]     6   8     3 [CGC]
```

Because DNA is double-stranded, when checking for a pattern, you should also check for its complement:

```
# Check for a the reverse complement of a pattern in a DNAString
matchPattern(reverseComplement(DNAString("CGC")), dna_seq)
```

```
## Views on a 13-letter DNAString subject
## subject: ATGCGCGCGGCTA
## views:
##       start end width
##   [1]     3   5     3 [GCG]
##   [2]     5   7     3 [GCG]
##   [3]     7   9     3 [GCG]
```

A somehow similar process goes with DNAStringSets, only with a different command name:

```
# Check for a pattern in a DNAStringSet
set2 <- DNAStringSet(c("AACCGGTTTCGA", "CGATCGCGCCGG"))
vmatchPattern("CG", set2)
```

```
## MIndex object of length 2
## [[1]]
## IRanges object with 2 ranges and 0 metadata columns:
##           start       end     width
##       <integer> <integer> <integer>
##   [1]         4         5         2
##   [2]        10        11         2
##
## [[2]]
## IRanges object with 4 ranges and 0 metadata columns:
##           start       end     width
##       <integer> <integer> <integer>
##   [1]         1         2         2
##   [2]         5         6         2
##   [3]         7         8         2
##   [4]        10        11         2
```

Now let's get back to the ENCODE project and find DNAStrings where ChIP-seq binding peaks are:

```
# Load human reference genome
library(BSgenome.Hsapiens.UCSC.hg19)

# Get DNAStrings of human genome where ChIP-seq binding peaks are
hepseq = getSeq(Hsapiens, HepG2)
hepseq[1:5]
```

```
## DNAStringSet object of length 5:
##     width seq
## [1]   410 GAGACAGGGTTTCACCATGTTGGCCAGGCTGGTT...CCTTCCAGGAAGCAGAAATGTTCAAGGACTCTC
## [2]   861 TGGGAAGGACACAACTGAATGAGGCTGTGCAGAG...AGCAGAACCTCCAACCGTGTGTGTGTGTGTGTG
## [3]  1156 GACACCTGCCACCCCGGACCCCACAGAATGGGCA...GCTTCGTGTCTGCTTTCTTATGTGTTTTTGTTT
## [4]   886 GTGAAGGCCCTGGAGTAGGCGGTGCGTACCCGGT...AGTGTTTTTGGCACCTCCGTGGGCACCTAGGCT
## [5]  1242 CATCCTCCACCTTAACACTCAGCACCCTTAGAGA...TTTTGTGTCCTACAAGCAGCCGGCGGCGCCGCC
```

Note: A DNA motif is defined as a nucleic acid sequence pattern that has some biological significance such as being DNA binding sites for a regulatory protein, i.e., a transcription factor.

Count occurrences of a motif in the gathered DNAStrings:

```
mot = "TCAAGGTCA"
sum(vcountPattern(mot, hepseq),
    vcountPattern(mot, reverseComplement(hepseq)))
```

```
## [1] 55
```

5.4 Data classes

In this section, we will work with different forms of genome data classes.

5.4.1 PLINK

PLINK format includes three files:

- .bed: Contains the genomic SNP data (Homozygous normal 00, Heterozygous 10, Homozygous variant 11, missing 01)
- .bim: Contains SNPs annotations
- .fam: Contains the subject's information

```
library(snpStats)
snps <- read.plink(bed = "Output/obesity.bed",
  bim = "Data/obesity.bim",
  fam = "Data/obesity.fam")
geno <- snps$genotypes
pheno <- snps$fam
annotation <- snps$map
```

5.4.1.1 Coordinating information from diverse tables

We will use a dataset which has three objects, like those of the PLINK format: a SnpMatrix and two dataframes.

```
# Load sample data
library(GSE5859Subset)
data(GSE5859Subset)
```

Upon attachment and loading of data, we have three data elements:

- geneAnnotation
- geneExpression

- sampleInfo

```
# Check elements
geneExpression[1:2, 1:4]
```

```
##           GSM136508.CEL.gz GSM136530.CEL.gz GSM136517.CEL.gz GSM136576.CEL.gz
## 1007_s_at         6.543954         6.401470         6.298943         6.837899
## 1053_at           7.546708         7.263547         7.201699         7.052761
```

```
geneAnnotation[1, ]
```

```
##     PROBEID  CHR   CHRLOC SYMBOL
## 1 1007_s_at chr6 30852327   DDR1
```

```
sampleInfo[1, ]
```

```
##     ethnicity       date         filename group
## 107       ASN 2005-06-23 GSM136508.CEL.gz     1
```

Now to integrate data:

```
# Integrate data
cbind(sampleInfo[1:3, ], colnames(geneExpression)[1:3], t(geneExpression)[1:3, 1:4])
```

```
##     ethnicity       date         filename group colnames(geneExpression)[1:3]
## 107       ASN 2005-06-23 GSM136508.CEL.gz     1              GSM136508.CEL.gz
## 122       ASN 2005-06-27 GSM136530.CEL.gz     1              GSM136530.CEL.gz
## 113       ASN 2005-06-27 GSM136517.CEL.gz     1              GSM136517.CEL.gz
##     1007_s_at  1053_at   117_at   121_at
## 107  6.543954 7.546708 5.402622 7.892544
## 122  6.401470 7.263547 5.050546 7.707754
## 113  6.298943 7.201699 5.024917 7.461886
```

5.4.2 ExpressionSet

An ExpressionSet is another form of data container which is designed to combine several different sources of information into a single convenient structure. The data in an ExpressionSet consists of expression data from microarray experiments, 'meta-data' describing samples in the experiment, annotations and meta-data about the features on the chip and information related to the protocol used for processing each sample.

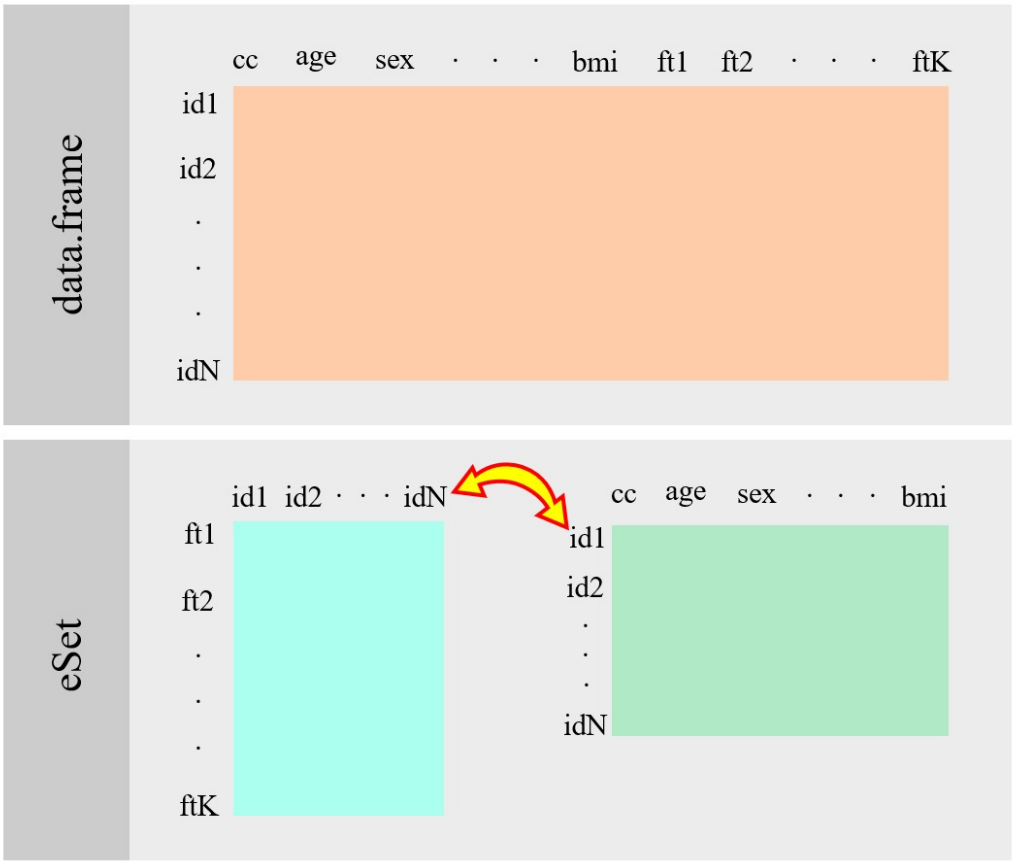


Figure 4: ExpressionSet data

```
# Import packages
library(Biobase)
```

Let's convert our previous PLINK data to an ExpressionSet:

```
# Bind the previous tables in an ExpressionSet
rownames(sampleInfo) = sampleInfo$filename
rownames(geneAnnotation) = geneAnnotation$PROBEID
es5859 = ExpressionSet(assayData = geneExpression)
pData(es5859) = sampleInfo
fData(es5859) = geneAnnotation
```

And we can easily subset data like the following:

```
# Subsetting data from an ExpressionSet
es5859_Y = es5859[which(fData(es5859)$CHR == "chrY"), ]
```

5.4.2.1 Annotation in ExpressionSet

annotation argument in an ExpressionSet points to a character describing the platform on which the samples were assayed. This is often the name of a Bioconductor chip annotation package, which facilitated down-stream analysis.

```
# Set annotation for the ExpressionSet
annotation(es5859) = "hgfocus.db"
```

5.4.2.2 ExperimentData in ExpressionSet

experimentData argument in an ExpressionSet points to an optional MIAME (Minimum Information About a Microarray Experiment) instance with meta-data (e.g., the lab and resulting publications from the analysis) about the experiment.

```
# Set experimentData for the ExpressionSet
library(annotate)
experimentData(es5859) = pmid2MIAME("17206142")
```

```
# Check experimentData
experimentData(es5859)
```

```
## Experiment data
##   Experimenter name: Spielman RS
##   Laboratory: NA
##   Contact information:
##   Title: Common genetic variants account for differences in gene expression among ethnic groups.
##   URL:
##   PMIDs: 17206142
##
##   Abstract: A 145 word abstract is available. Use 'abstract' method.
```

```
# Check abstract
abstract(es5859)
```

```
## [1] "Variation in DNA sequence contributes to individual differences in quantitative traits, but in
```

5.4.3 SummarizedExperiment

SummarizedExperiment is a comprehensive data structure that can be used to store expression and methylation data from microarrays or read counts from RNA-seq experiments.It can contain slots for one or more omic datasets, feature annotation (e.g. genes, transcripts, SNPs, CpGs), individual phenotypes and experimental details,

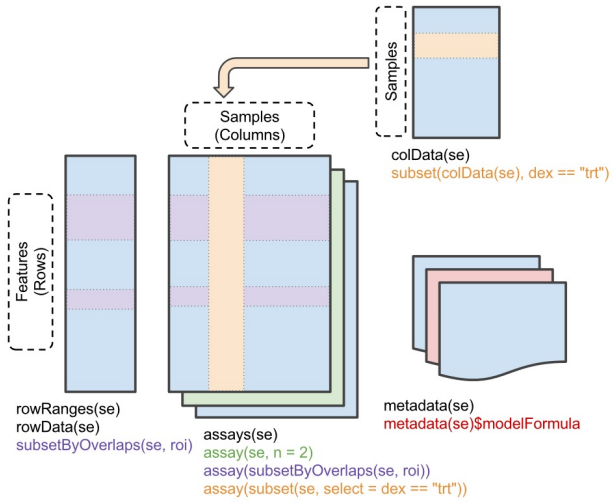


Figure 5: SummarizedExperiment data

Let's load a sample SummarizedExperiment data:

```r
# Load sample data (RNA-seq study)
library(airway)
library(SummarizedExperiment)
data(airway)
```

Check the data:

```r
# Get metadata about the experiment
metadata(airway)
```

```
## [[1]]
## Experiment data
##   Experimenter name: Himes BE
##   Laboratory: NA
##   Contact information:
##   Title: RNA-Seq transcriptome profiling identifies CRISPLD2 as a glucocorticoid responsive gene tha
##   URL: http://www.ncbi.nlm.nih.gov/pubmed/24926665
##   PMIDs: 24926665
##
##   Abstract: A 226 word abstract is available. Use 'abstract' method.
```

```r
# Get the first four feature names
rownames(airway)[1:4]
```

```
## [1] "ENSG00000000003" "ENSG00000000005" "ENSG00000000419" "ENSG00000000457"
```

```r
# Get the first four exon coordinates for a specific gene
rowRanges(airway)$ENSG00000172057[1:4]
```

```
## GRanges object with 4 ranges and 2 metadata columns:
##       seqnames            ranges strand |   exon_id       exon_name
##          <Rle>         <IRanges>  <Rle> | <integer>     <character>
##   [1]       17 38077294-38078938      - |    549057 ENSE00001316037
##   [2]       17 38077296-38077570      - |    549058 ENSE00002684279
##   [3]       17 38077929-38078002      - |    549059 ENSE00002697088
##   [4]       17 38078351-38078552      - |    549060 ENSE00002718599
##   -------
##   seqinfo: 722 sequences (1 circular) from an unspecified genome
```

```r
# Get the sample-level data
colData(airway)
```

```
## DataFrame with 8 rows and 9 columns
##             SampleName     cell      dex    albut        Run avgLength
##               <factor> <factor> <factor> <factor>   <factor> <integer>
## SRR1039508 GSM1275862  N61311     untrt    untrt SRR1039508       126
## SRR1039509 GSM1275863  N61311     trt      untrt SRR1039509       126
## SRR1039512 GSM1275866  N052611    untrt    untrt SRR1039512       126
## SRR1039513 GSM1275867  N052611    trt      untrt SRR1039513        87
## SRR1039516 GSM1275870  N080611    untrt    untrt SRR1039516       120
## SRR1039517 GSM1275871  N080611    trt      untrt SRR1039517       126
## SRR1039520 GSM1275874  N061011    untrt    untrt SRR1039520       101
## SRR1039521 GSM1275875  N061011    trt      untrt SRR1039521        98
##             Experiment    Sample    BioSample
##               <factor>  <factor>     <factor>
## SRR1039508  SRX384345 SRS508568 SAMN02422669
## SRR1039509  SRX384346 SRS508567 SAMN02422675
```

```
## SRR1039512  SRX384349 SRS508571 SAMN02422678
## SRR1039513  SRX384350 SRS508572 SAMN02422670
## SRR1039516  SRX384353 SRS508575 SAMN02422682
## SRR1039517  SRX384354 SRS508576 SAMN02422673
## SRR1039520  SRX384357 SRS508579 SAMN02422683
## SRR1039521  SRX384358 SRS508580 SAMN02422677
```

```
# Check for the existence of overlapping regions
# in the exon coordinates of one specific gene
reduce(rowRanges(airway)$ENSG00000172057)
```

```
## GRanges object with 8 ranges and 0 metadata columns:
##       seqnames            ranges strand
##          <Rle>         <IRanges>  <Rle>
##   [1]       17 38077294-38078938      -
##   [2]       17 38079365-38079516      -
##   [3]       17 38080283-38080478      -
##   [4]       17 38081008-38081058      -
##   [5]       17 38081422-38081624      -
##   [6]       17 38081876-38083099      -
##   [7]       17 38083320-38083482      -
##   [8]       17 38083737-38083854      -
##   -------
##   seqinfo: 722 sequences (1 circular) from an unspecified genome
```

```
# Check sample groups
table(airway$dex)
```

```
##
##   trt untrt
##     4     4
```

5.5 Expression array archives

Expression array archives contain the ExpressionSet for a lot of various species and experiments. Expression array archive in the US:

- Gene Expression Omnibus (GEO)
- Package: *GEOquery*

Expression array archive in Europe:

- European Molecular Biology Laboratories (EMBL)
- Package: *ArrayExpress*

5.5.1 Obtaining the ExpressionSet for a GEO series

```
library(GEOquery)
# glioMA = getGEO("GSE78703")[[1]]
```

5.6 Obtaining the ExpressionSet for an EMBL series

```
library(ArrayExpress)
# glioMA = getAE("E-MTAB-5797")
```

5.7 Storing data

In this section, we will learn how to store data in a database and manipulate that data.

This dataset can quite easily be handled directly in R, but for larger datasets dimensionality can become a problem. The simplest workaround is to store the data in a SQL query and retrieve only the parts of the data that are needed at any given time.

In here, we will use two files:

- a genotypes file: contains all genotype calls for all SNP and all samples.
- a map file: holds mapping information of the SNP, e.g., chromosome, physical location, etc.

We want to create a database. There are two ways:

5.7.1 Using command line

For our database we need to create a schema (a schema is simply a text file that describes the database structure and is used to create the initial empty DB). A simple schema will consist of two tables, one for each of the files and one column for each source of information.

```
cat Data/snpDB.sql
```

We are ready to create the database, named "SNPsmall":

```
sqlite3 Data/SNPsmall < Data/snpDB.sql
```

5.7.2 Using R

This process does not need an schema.

```
# Load package
library(RSQLite)

# Create a new blank database called SNPsmall
con = dbConnect(dbDriver("SQLite"), dbname = "Output/SNPsmall")

# Upload files to the database
dbWriteTable(
  con,
  "snpmap",
  "Data/SNPmap.txt",
  header = TRUE,
  append = T,
  sep = "\t"
)

dbWriteTable(
  con,
  "SNP",
  "Data/SNPSample1.txt",
  append = T,
  header = TRUE,
  skip = 0,
  sep = "\t"
)

# Take a look at the tables and fields in the databse
```

```
con = dbConnect(dbDriver("SQLite"), dbname = "Output/SNPsmall")
dbListTables(con)
```

```
## [1] "SNP"    "snpmap"
```

```
dbListFields(con, "snpmap")
```

```
## [1] "name"       "chromosome" "position"
```

```
dbListFields(con, "SNP")
```

```
## [1] "snp"     "animal"  "allele1" "allele2" "x"       "y"       "gcscore"
```

You can add indexes to the database to make it faster.

```
dbGetQuery(con, "CREATE INDEX chromosome_idx ON snpmap(chromosome)")
```

```
## data frame with 0 columns and 0 rows
```

```
dbGetQuery(con, "CREATE INDEX snp_idx ON SNP(animal)")
```

```
## data frame with 0 columns and 0 rows
```

```
dbGetQuery(con, "CREATE INDEX ID_idx ON SNP(snp)")
```

```
## data frame with 0 columns and 0 rows
```

The function **dbGetQuery** is used to send an SQL query to the DB and return the data in a single step.

```
# Retrieve the number of records in a table
dbGetQuery(con, "select count (*) from snpmap")
```

```
##   count (*)
## 1     20000
```

```
# Retrieve sample ids
sampleids_s1 = as.vector(dbGetQuery(con, "select distinct animal from SNP"))$animal
head(sampleids_s1)
```

```
## [1] "\"sample1\""  "\"sample10\"" "\"sample11\"" "\"sample12\"" "\"sample13\""
## [6] "\"sample14\""
```

```
# Retrieve all data associated with the first sample
hold_s1 = dbGetQuery(con,
                  paste("select * from SNP where animal='", sampleids_s1[1], "'", sep = ""))
head(hold_s1)
```

```
##                             snp    animal allele1 allele2     x     y gcscore
## 1 "250506CS3900065000002_1238.1" "sample1"     "A"     "B" 0.833 0.707  0.8446
## 2  "250506CS3900140500001_312.1" "sample1"     "B"     "B" 0.018 0.679  0.9629
## 3  "250506CS3900176800001_906.1" "sample1"     "B"     "B" 0.008 1.022  0.9484
## 4 "250506CS3900211600001_1041.1" "sample1"     "B"     "B" 0.010 0.769  0.9398
## 5 "250506CS3900218700001_1294.1" "sample1"     "B"     "B" 0.000 0.808  0.9272
## 6  "250506CS3900283200001_442.1" "sample1"     "B"     "B" 0.019 0.583  0.9552
```

When we are finished with the database we should close the connection.

```
dbDisconnect(con)
```

5.8 Reference genomes

Let's check how to load some refernce genomes and specially human reference genome:

```r
# Import packages
library(BSgenome)
library(Biostrings)
```

```r
# Check some of the available reference genomes
available.genomes()[1:5]
```

```
## [1] "BSgenome.Alyrata.JGI.v1"
## [2] "BSgenome.Amellifera.BeeBase.assembly4"
## [3] "BSgenome.Amellifera.NCBI.AmelHAv3.1"
## [4] "BSgenome.Amellifera.UCSC.apiMel2"
## [5] "BSgenome.Amellifera.UCSC.apiMel2.masked"
```

```r
# Load Hsapiens.UCSC.hg19 reference seq
library(BSgenome.Hsapiens.UCSC.hg19)
hs = BSgenome.Hsapiens.UCSC.hg19
```

```r
# Acquire a chromosome's sequence
hs$chr17
```

```
## 81195210-letter DNAString object
## seq: AAGCTTCTCACCCTGTTCCTGCATAGATAATTGCAT...GTGGGTGTGGGTGTGGTGTGTGGGTGTGGGTGTGGT
```

5.9 Annotations

Let's see how to annotate the data now. We will try with human reference genome annotation.

5.9.1 UCSC annotation

```r
# Load Hsapiens.UCSC.hg19 transcripts and genes
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
hs_txdb = TxDb.Hsapiens.UCSC.hg19.knownGene
hs_txdb
```

```
## TxDb object:
## # Db type: TxDb
## # Supporting package: GenomicFeatures
## # Data source: UCSC
## # Genome: hg19
## # Organism: Homo sapiens
## # Taxonomy ID: 9606
## # UCSC Table: knownGene
## # Resource URL: http://genome.ucsc.edu/
## # Type of Gene ID: Entrez Gene ID
## # Full dataset: yes
## # miRBase build ID: GRCh37
## # transcript_nrow: 82960
## # exon_nrow: 289969
## # cds_nrow: 237533
## # Db created by: GenomicFeatures package from Bioconductor
## # Creation time: 2015-10-07 18:11:28 +0000 (Wed, 07 Oct 2015)
## # GenomicFeatures version at creation time: 1.21.30
## # RSQLite version at creation time: 1.0.0
## # DBSCHEMAVERSION: 1.1
```

```
# Get the addresses of genes by Entrez gene IDs
genes(hs_txdb)
```

```
## GRanges object with 23056 ranges and 1 metadata column:
##         seqnames              ranges strand |     gene_id
##            <Rle>           <IRanges>  <Rle> | <character>
##       1    chr19   58858172-58874214      - |           1
##      10     chr8   18248755-18258723      + |          10
##     100    chr20   43248163-43280376      - |         100
##    1000    chr18   25530930-25757445      - |        1000
##   10000     chr1 243651535-244006886      - |       10000
##     ...      ...                 ...    ... .         ...
##    9991     chr9 114979995-115095944      - |        9991
##    9992    chr21   35736323-35743440      + |        9992
##    9993    chr22   19023795-19109967      - |        9993
##    9994     chr6   90539619-90584155      + |        9994
##    9997    chr22   50961997-50964905      - |        9997
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# Get the addresses of exons
exons(hs_txdb)
```

```
## GRanges object with 289969 ranges and 1 metadata column:
##                  seqnames      ranges strand |   exon_id
##                     <Rle>   <IRanges>  <Rle> | <integer>
##        [1]           chr1 11874-12227      + |         1
##        [2]           chr1 12595-12721      + |         2
##        [3]           chr1 12613-12721      + |         3
##        [4]           chr1 12646-12697      + |         4
##        [5]           chr1 13221-14409      + |         5
##        ...            ...         ...    ... .       ...
##   [289965] chrUn_gl000241 35706-35859      - |    289965
##   [289966] chrUn_gl000241 36711-36875      - |    289966
##   [289967] chrUn_gl000243 11501-11530      + |    289967
##   [289968] chrUn_gl000243 13608-13637      + |    289968
##   [289969] chrUn_gl000247   5787-5816      - |    289969
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# Get the addresses of transcripts
transcripts(hs_txdb)
```

```
## GRanges object with 82960 ranges and 2 metadata columns:
##                  seqnames        ranges strand |     tx_id     tx_name
##                     <Rle>     <IRanges>  <Rle> | <integer> <character>
##        [1]           chr1   11874-14409      + |         1  uc001aaa.3
##        [2]           chr1   11874-14409      + |         2  uc010nxq.1
##        [3]           chr1   11874-14409      + |         3  uc010nxr.1
##        [4]           chr1   69091-70008      + |         4  uc001aal.1
##        [5]           chr1 321084-321115      + |         5  uc001aaq.2
##        ...            ...           ...    ... .       ...         ...
##   [82956] chrUn_gl000237        1-2686      - |     82956  uc011mgu.1
##   [82957] chrUn_gl000241   20433-36875      - |     82957  uc011mgv.2
##   [82958] chrUn_gl000243   11501-11530      + |     82958  uc011mgw.1
```

```
##   [82959] chrUn_gl000243   13608-13637      + |     82959  uc022brq.1
##   [82960] chrUn_gl000247     5787-5816      - |     82960  uc022brr.1
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

```
# Filter all exons identified for two different genes (by their Entrez gene IDs)
exons(
  hs_txdb,
  columns = c("EXONID", "TXNAME", "GENEID"),
  filter = list(gene_id = c(100, 101))
)
```

```
## GRanges object with 39 ranges and 3 metadata columns:
##        seqnames              ranges strand |    EXONID
##           <Rle>           <IRanges>  <Rle> | <integer>
##    [1]    chr10 135075920-135076737      - |    144421
##    [2]    chr10 135077192-135077269      - |    144422
##    [3]    chr10 135080856-135080921      - |    144423
##    [4]    chr10 135081433-135081570      - |    144424
##    [5]    chr10 135081433-135081622      - |    144425
##    ...      ...                 ...    ... .       ...
##   [35]    chr20   43254210-43254325      - |    256371
##   [36]    chr20   43255097-43255240      - |    256372
##   [37]    chr20   43257688-43257810      - |    256373
##   [38]    chr20   43264868-43264929      - |    256374
##   [39]    chr20   43280216-43280376      - |    256375
##                                TXNAME          GENEID
##                           <CharacterList> <CharacterList>
##    [1] uc009ybi.3,uc010qva.2,uc021qbe.1             101
##    [2]            uc009ybi.3,uc021qbe.1             101
##    [3] uc009ybi.3,uc010qva.2,uc021qbe.1             101
##    [4]                        uc009ybi.3             101
##    [5]            uc010qva.2,uc021qbe.1             101
##    ...                              ...             ...
##   [35]                        uc002xmj.3             100
##   [36]                        uc002xmj.3             100
##   [37]                        uc002xmj.3             100
##   [38]                        uc002xmj.3             100
##   [39]                        uc002xmj.3             100
##   -------
##   seqinfo: 93 sequences (1 circular) from hg19 genome
```

5.9.2 ENSEMBL annotation

```
# Load Hsapiens.v75 transcripts, genes, and proteins
library(ensembldb)
library(EnsDb.Hsapiens.v75)
hs_edb = EnsDb.Hsapiens.v75
hs_edb
```

```
## EnsDb for Ensembl:
## |Backend: SQLite
## |Db type: EnsDb
## |Type of Gene ID: Ensembl Gene ID
## |Supporting package: ensembldb
```

```
## |Db created by: ensembldb package from Bioconductor
## |script_version: 0.3.0
## |Creation time: Thu May 18 09:15:45 2017
## |ensembl_version: 75
## |ensembl_host: localhost
## |Organism: homo_sapiens
## |taxonomy_id: 9606
## |genome_build: GRCh37
## |DBSCHEMAVERSION: 2.0
## | No. of genes: 64102.
## | No. of transcripts: 215647.
## |Protein data available.
```

```
# List attributes
listTables(hs_edb)
```

```
## $gene
## [1] "gene_id"          "gene_name"        "gene_biotype"     "gene_seq_start"
## [5] "gene_seq_end"     "seq_name"         "seq_strand"       "seq_coord_system"
## [9] "symbol"
##
## $tx
## [1] "tx_id"            "tx_biotype"       "tx_seq_start"     "tx_seq_end"
## [5] "tx_cds_seq_start" "tx_cds_seq_end"   "gene_id"          "tx_name"
##
## $tx2exon
## [1] "tx_id"    "exon_id"  "exon_idx"
##
## $exon
## [1] "exon_id"        "exon_seq_start" "exon_seq_end"
##
## $chromosome
## [1] "seq_name"    "seq_length"  "is_circular"
##
## $protein
## [1] "tx_id"            "protein_id"       "protein_sequence"
##
## $uniprot
## [1] "protein_id"           "uniprot_id"           "uniprot_db"
## [4] "uniprot_mapping_type"
##
## $protein_domain
## [1] "protein_id"            "protein_domain_id"     "protein_domain_source"
## [4] "interpro_accession"    "prot_dom_start"        "prot_dom_end"
##
## $entrezgene
## [1] "gene_id"  "entrezid"
##
## $metadata
## [1] "name"  "value"
```

```
# Return seq_name (i.e., chr name) for transcripts of a specific gene
transcripts(hs_edb,
            filter = GenenameFilter("ZBTB16"),
            columns = "seq_name")
```

```
## GRanges object with 9 ranges and 2 metadata columns:
##                    seqnames              ranges strand |           tx_id
##                       <Rle>           <IRanges>  <Rle> |     <character>
##   ENST00000335953       11 113930315-114121398      + | ENST00000335953
##   ENST00000541602       11 113930447-114060486      + | ENST00000541602
##   ENST00000544220       11 113930459-113934368      + | ENST00000544220
##   ENST00000535700       11 113930979-113934466      + | ENST00000535700
##   ENST00000392996       11 113931229-114121374      + | ENST00000392996
##   ENST00000539918       11 113935134-114118066      + | ENST00000539918
##   ENST00000545851       11 114051488-114118018      + | ENST00000545851
##   ENST00000535379       11 114107929-114121279      + | ENST00000535379
##   ENST00000535509       11 114117512-114121198      + | ENST00000535509
##                     gene_name
##                   <character>
##   ENST00000335953      ZBTB16
##   ENST00000541602      ZBTB16
##   ENST00000544220      ZBTB16
##   ENST00000535700      ZBTB16
##   ENST00000392996      ZBTB16
##   ENST00000539918      ZBTB16
##   ENST00000545851      ZBTB16
##   ENST00000535379      ZBTB16
##   ENST00000535509      ZBTB16
##   -------
##   seqinfo: 1 sequence from GRCh37 genome
```

5.9.3 OrgDb annotation

```
# Import packages
library(AnnotationDbi)
library(org.Hs.eg.db)
```

```
# View the first five columns' names
columns(org.Hs.eg.db)[1:5]
```

```
## [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS"
```

```
# View the first 5 keys (i.e., values) in a specific column (i.e., keytype)
keys(org.Hs.eg.db, keytype = "ENSEMBL")[1:5]
```

```
## [1] "ENSG00000121410" "ENSG00000175899" "ENSG00000291190" "ENSG00000171428"
## [5] "ENSG00000156006"
```

```
# Map the ID from a specific key (e.g., ENSG00000175899) from its
# corresponding keytype (e.g., ENSEMBL) to another keytype (e.g., ENTREZID)
mapIds(org.Hs.eg.db,
       keys = "ENSG00000175899",
       column = "ENTREZID",
       keytype = "ENSEMBL")
```

```
## ENSG00000175899
##             "2"
```

```
# A process similar to the mapIds function from the AnnotationDbi package,
# this time in the biomaRt package
library(biomaRt)
```

```
mart <-
  useMart(biomart = "ensembl",
          dataset = "hsapiens_gene_ensembl",
          host = "https://useast.ensembl.org")
getBM(
  mart = mart,
  attributes = "entrezgene_id",
  filters = "ensembl_gene_id",
  values = "ENSG00000175899"
)
```

```
##   entrezgene_id
## 1             2
```

```
# List the first 5 attributes for a specific mart object
listAttributes(mart)[1:5,]
```

```
##                            name                 description         page
## 1               ensembl_gene_id               Gene stable ID feature_page
## 2       ensembl_gene_id_version       Gene stable ID version feature_page
## 3         ensembl_transcript_id         Transcript stable ID feature_page
## 4 ensembl_transcript_id_version Transcript stable ID version feature_page
## 5            ensembl_peptide_id            Protein stable ID feature_page
```

```
# List the first 5 filters for a specific mart object
listFilters(mart)[1:5,]
```

```
##              name              description
## 1 chromosome_name Chromosome/scaffold name
## 2           start                    Start
## 3             end                      End
## 4      band_start                Band Start
## 5        band_end                  Band End
```

5.9.4 Gene sets and pathways

5.9.4.1 Gene Ontology Gene Ontology (GO) organizes terms relevant to the roles of genes and gene products in:

- biological processes
- molecular functions
- cellular components

```
# Import packages
library(GO.db)
library(AnnotationDbi)
```

```
# View the columns' names
columns(GO.db)
```

```
## [1] "DEFINITION" "GOID"       "ONTOLOGY"   "TERM"
```

```
# View the first 5 keys (i.e., values) in a specific column (i.e., keytype)
keys(GO.db, keytype = "ONTOLOGY")[1:5]
```

```
## [1] "BP"        "CC"        "MF"        "universal" NA
```

```r
# Import packages
library(KEGGREST)
```

```r
# Get data for the gene with the Entrez ID 675 (i.e., BRCA2 gene)
brca2K = keggGet("hsa:675")

# List gene attributes
names(brca2K[[1]])
```

5.9.4.2 KEGG: Kyoto Encyclopedia of Genes and Genomes

```
##  [1] "ENTRY"     "SYMBOL"    "NAME"      "ORTHOLOGY" "ORGANISM"  "PATHWAY"
##  [7] "NETWORK"   "DISEASE"   "BRITE"     "POSITION"  "MOTIF"     "DBLINKS"
## [13] "STRUCTURE" "AASEQ"     "NTSEQ"
```

```r
# Get its seq data
brca2K[[1]]$NTSEQ
```

```
## DNAStringSet object of length 1:
##     width seq
## [1] 10257 ATGCCTATTGGATCCAAAGAGAGGCCAACATTTT...CAGGACACAATTACAACTAAAAAATATATCTAA
```

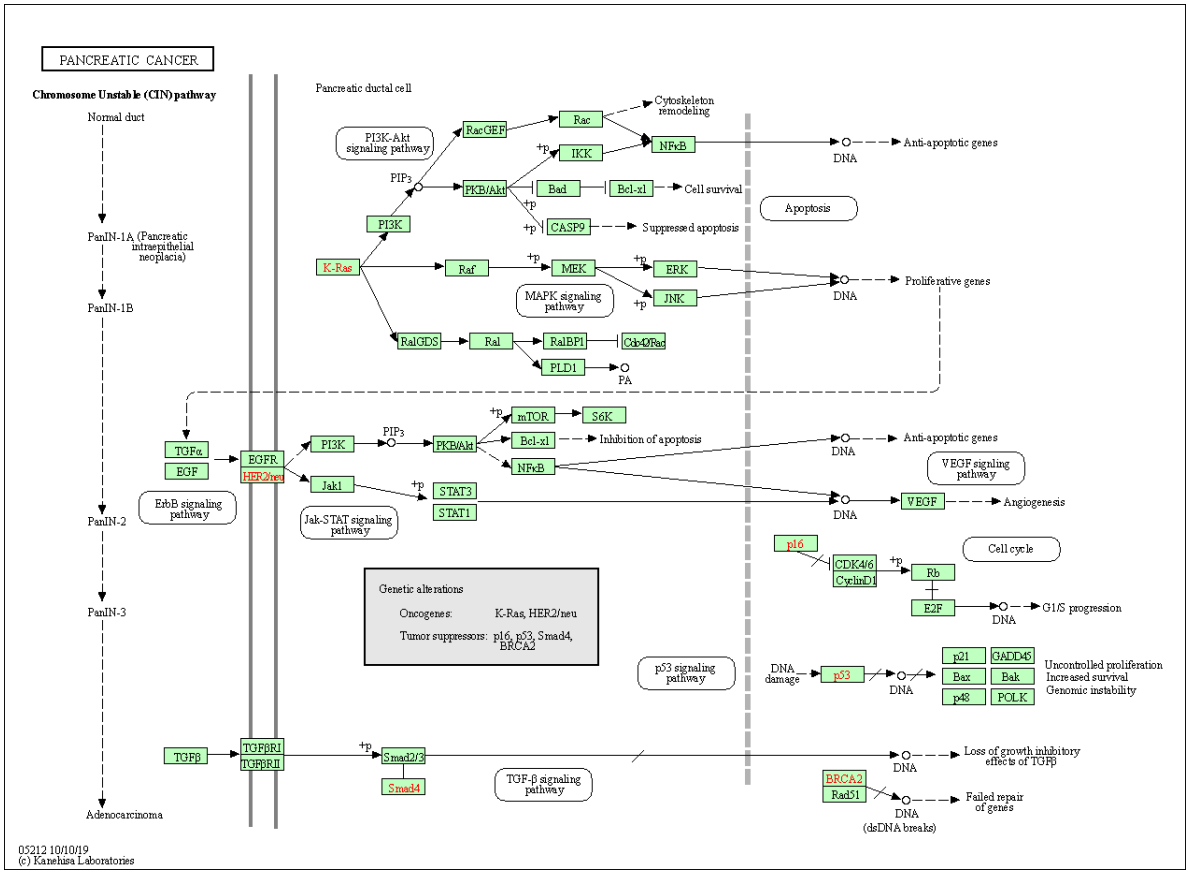
```r
# Get the list of genes making up a pathway model
brpat = keggGet("path:hsa05212")

# List pathway attributes
names(brpat[[1]])
```

```
##  [1] "ENTRY"       "NAME"        "DESCRIPTION" "CLASS"       "PATHWAY_MAP"
##  [6] "NETWORK"     "DISEASE"     "DRUG"        "ORGANISM"    "GENE"
## [11] "COMPOUND"    "REL_PATHWAY" "KO_PATHWAY"  "REFERENCE"
```

```r
# Get the Entrez IDs for the first five genes
brpat[[1]]$GENE[seq(1, 10, 2)]
```

```
## [1] "3845" "5290" "5293" "5291" "5295"
```

```r
# Get an illustration of the pathway
library(png)
library(grid)
brpng = keggGet("hsa05212", "image")
grid.raster(brpng)
```


PANCREATIC CANCER
Chromosome Unstable (CIN) pathway
Normal duct
PanIN-1A (Pancreatic intraepithelial neoplasia)
PanIN-1B
PanIN-2
PanIN-3
Adenocarcinoma
Pancreatic ductal cell
PI3K-Akt signaling pathway
RacGEF
Rac
Cytoskeleton remodeling
NFκB
Anti-apoptotic genes
DNA
IKK
PIP3
PKB/Akt
Bad
Bcl-xl
Cell survival
Apoptosis
PI3K
CASP9
Suppressed apoptosis
K-Ras
Raf
MEK
ERK
JNK
MAPK signaling pathway
Proliferative genes
RalGDS
Ral
RalBP1
Cdc42/Rac
PLD1
PA
TGFα
EGF
EGFR
HER2/neu
ErbB signaling pathway
PI3K
PIP3
PKB/Akt
mTOR
S6K
Bcl-xl
Inhibition of apoptosis
NFκB
Anti-apoptotic genes
DNA
VEGF signaling pathway
Jak1
STAT3
STAT1
Jak-STAT signaling pathway
VEGF
Angiogenesis
DNA
p16
CDK4/6
CyclinD1
Cell cycle
Rb
E2F
G1/S progression
DNA
Genetic alterations
Oncogenes: K-Ras, HER2/neu
Tumor suppressors: p16, p53, Smad4, BRCA2
p53 signaling pathway
DNA damage
p53
DNA
p21
GADD45
Bax
Bak
p48
POLK
Uncontrolled proliferation
Increased survival
Genomic instability
TGFβ
TGFβRI
TGFβRII
Smad2/3
Smad4
TGF-β signaling pathway
DNA
Loss of growth inhibitory effects of TGFβ
BRCA2
Rad51
DNA
(dsDNA breaks)
Failed repair of genes
05212 10/10/19
(c) Kanehisa Laboratories