

NGS, from **data** to **table** to **figure**

From downloading data to final publication ready figures

Meng Li

Background

- I introduce what I learned from mine supervisors these years. This report also include some knowledge which I learn by myself.

Object

1. I hope everyone can follow me, and I ensure you will meet these problems if you do NGS analysis or data analysis.
2. If you have any question at any time, please **don't hesitate** to ask a question, actually some of the materials are very difficult to me, and I also learn it while this class running.
3. Although it may take some time to learn, but it will save a lot of time in the future.
4. I hope after each class, you can make some practice and read the supplement files.

Object

5. All the course materials can be download from https://github.com/regSNPs/from_data_2_table_2_figure

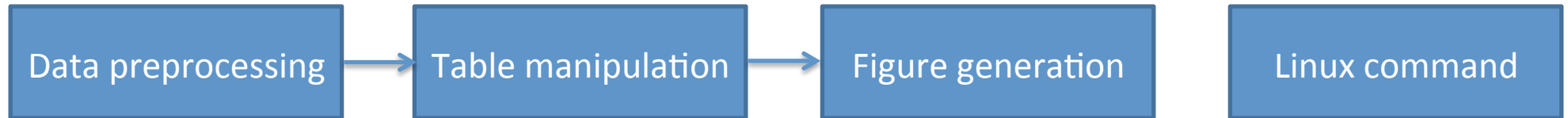
6. This is not a advance class, but I confirm you will learn some 'latest knowledge', **if you search R question in stackoverflow you will see a lot of question use the knowledge here.**

7. I think after the report, the students can learn how to perform data analysis when supervisor give us a task.

Workflow of the report

- 1. Data preprocessing (First we need to know where we can get the data and how to preprocess the data)
- 2. Table manipulation (We usually don't use the raw data directly, we need process them into a desired format for further investigation)
- 3. Figure generation (Human beings can't read thousands of rows in table simultaneously, we need figure to visualize them)
- 4. Linux (Besides R, we also need Linux to facilitate some work)

Let's learn together



The required knowledge include:

Biology (understand the biology process of the problem)

Programming skill (understand the code)

Statistic graphic (understand the figure)

The example problem

We will follow a problem during the course.

Problem to solve: RNA binding protein binding gene analysis.

We want to study the relationship between multi RBPs and gene expression. i.e the object is to study if the number of RBPs bind the gene can affect the gene expression level in different genes in same cell line.

The example problem

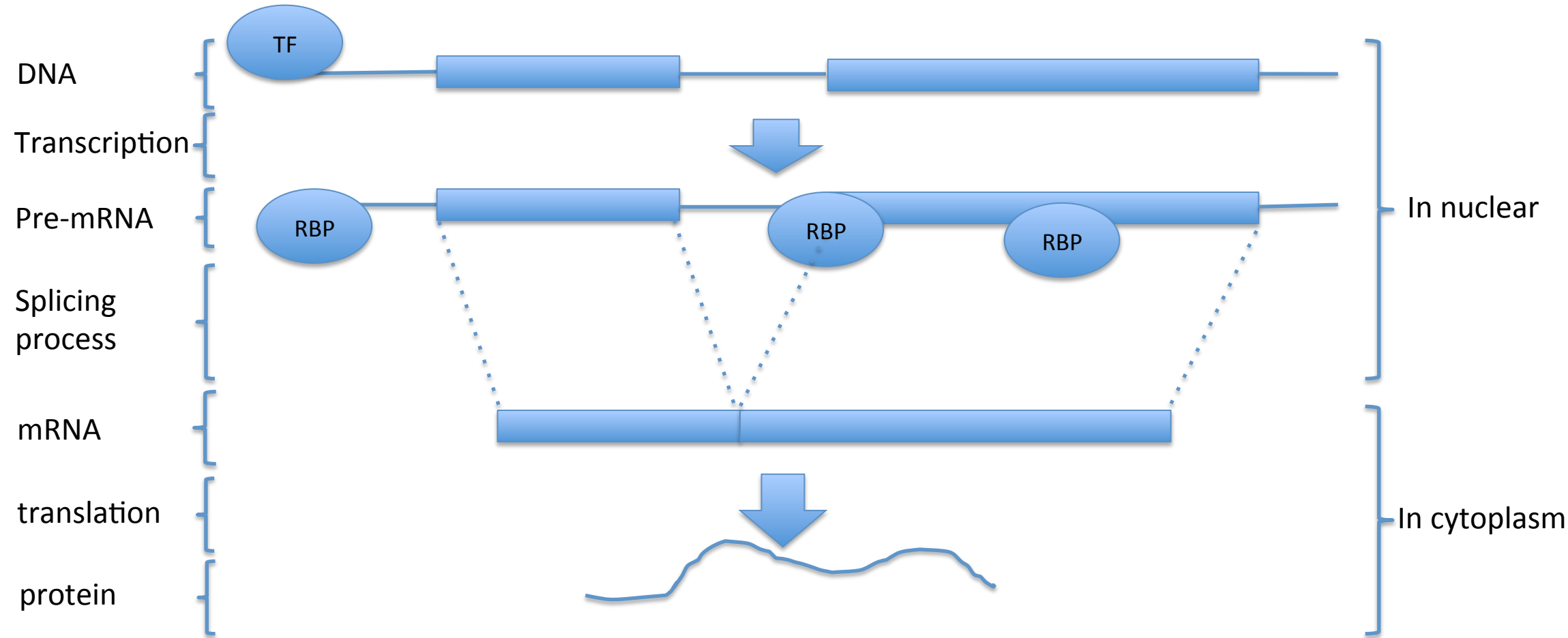
- We can also study the RBP and gene expression analysis in different cell lines, **but I didn't have that data currently.**

The biology process of this

Although I'm not very good at biology, but **the first thing is understanding the biology process of the problem.** which will make the analysis meaningful.

I will try to explain the biology process of this problem first.

The biological process of gene expression



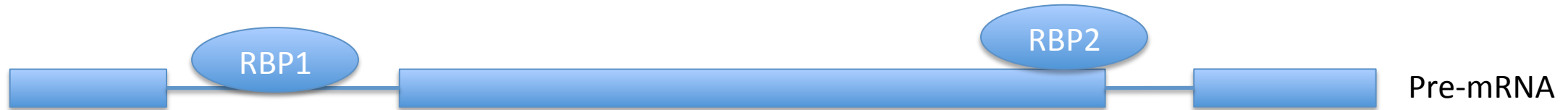
We all know the TF can affect the gene expression, but **will the RBPs regulate the gene expression through regulating the transcription of pre-mRNA?** I don't know either, let's do it!!

The analysis process

- 1. We need data to know where the RBP binds? This kind of data can be got from **CLIP-seq technique**.
- 2. We need to know how the gene expressed in a cell line. This data can be got from **RNA-seq technique**.
- 3. We need to ensure that both two kinds of dataset **come from the same cell line**. After we got the above information, our final analysis is doing **correlation between RBP binding count and gene expression**.

The CLIP-seq technique

First we need to know where the RBP binds? The below is a demonstration of RBP binding region, **in reality, we don't know where the RBP binds.**



The technique is called CLIP-seq, which can locate the target of RBP binding sites in the pre-mRNA.

The CLIP-seq experiment

Got from Wikipedia

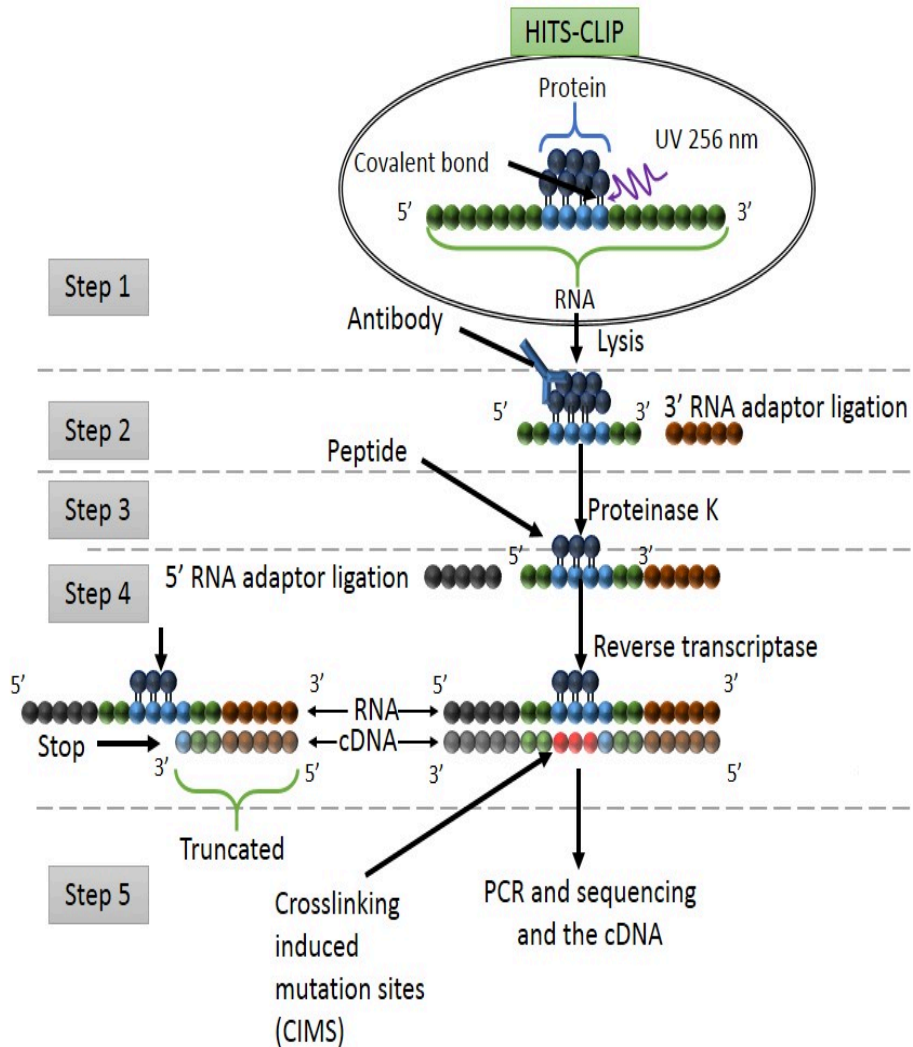


Figure 2: HITS-CLIP

Step 1

HITS-CLIP begins with the in-vivo cross-linking of RNA-protein complexes using ultraviolet light. The cell is lysed and the protein of interest is isolated using immunoprecipitation.

Step 2

Washing is performed to remove free RNA, and RNA adaptors are ligated at the 3' ends.

Step 3

Proteinase K digestion is performed. This leaves a peptide at the cross-link site that modifies the chemical structure of the nucleotide.

Step 4

5' RNA adaptors are ligated and cDNA is synthesized using reverse transcription.

Step 5

PCR and sequencing of the cDNA.

CLIP-seq workflow

It is a technique just like the CHIP-seq, whereas work on pre-mRNA and RBPs.

The goal of the technique is extracting the sequence in the RBP binding sites.

We just need to know that the reads we will see later come from the binding sites.

I will show you the standard protocol later.

Sequence Database and annotation database

- There are **two kinds of biology database**. We need sequence database to get the sequence and annotation database to get gene annotation.
- 1. The sequence database store the sequence from sequencer, this kind of database is very very large.
- 2. The annotation database stores the genome annotation.

The sequence database

- Once we decide the data (**CLIP-seq** dataset) we need. Next step is selecting the database that we are going to extract the data.
- There are various kinds of sequence database, eg: **Encode, modEncode, GEO, SRA, EBI, GTEx, TCGA, 1000 Genome Project, ESP6500**
- I will give a brief introduction for these databases.

The annotation database

- There are many annotation database available which include: UCSC, refseq, and Ensembl, Uinprot, PDB et.al.
- I will give a bries introduction to these databases.
- Usually, refseq is more accurate while Ensembl is more comprehensive.
- For refseq hg19 (GRCh37), it contains around 50000 transcripts.
- For ensembl hg19 (GRCh37), it contains around 200000 transcripts.

Encode&UCSC

- Encode
- Encode dataset matrix:
 - <https://www.encodeproject.org/matrix/?type=Experiment&x.limit=>
- UCSC
- Table browser, download various kinds of annotation from different assemble.

Encode&UCSC database

- I will show you the two databases.
- One is the encode database, another is the UCSC database.
- **For encode, I will show how to get CLIP data for SRSF1 in K562. (SRSF1.web)**
<https://www.encodeproject.org/experiments/ENCSR432XUP/>
- **For UCSC, I will show how to get the annotation, which will be used in various kinds of NGS analysis. (e.g. How to use the table browser, how to use download)**

UCSC

- Besides the web interface, UCSC also provide two kinds of methods to extracting data from their database.
- 1. DAS server
[E.g. http://genome.ucsc.edu/cgi-bin/das/hg19/dna?segment=chr4:35654,35695](http://genome.ucsc.edu/cgi-bin/das/hg19/dna?segment=chr4:35654,35695)
- 2. mysql client:

Users can also use various kinds of mysql client (e.g. mysql workbench, JDBC et.al.)

```
Mysql -user=genome -host=genom-mysql.cse.ucsc.edu -A
```

Data formats

The download eCLIP data from Encode can be many different formats.

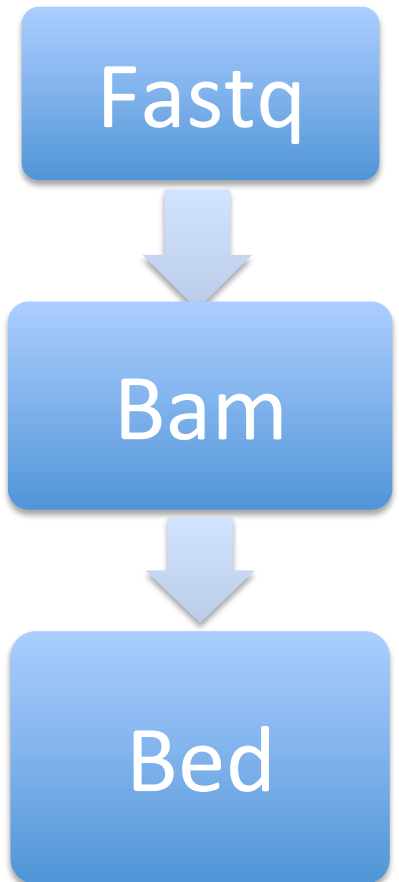
Including:

fastq format (original reads),
bam format (alignment file),
bed format (peaks region file).

Here we assume the raw format is fastq format, I will show you the above formats.
(UCSC format help:

<http://genome.ucsc.edu/FAQ/FAQformat.html>)

Data preprocessing
workflow



From fastq to bam

- The main command here is **read alignment**.

e.g `bowtie2 [options]* -x <bowtie-index> {-1 read1 -2 read2 | -U <r>} [-S <sam>]`

- But there are possibly other command, eg:

1. **Demultiplexing** (sometimes, multi samples are sequenced once a time, multiplexing is a technique to index them)

2. **Cutadaptor** (adaptor is used to do sequencing, we need to remove the adapters from head or tail of the adaptor)

3. **FastQC** (quality control)

4. **rmRep** (remove repetitive elements)

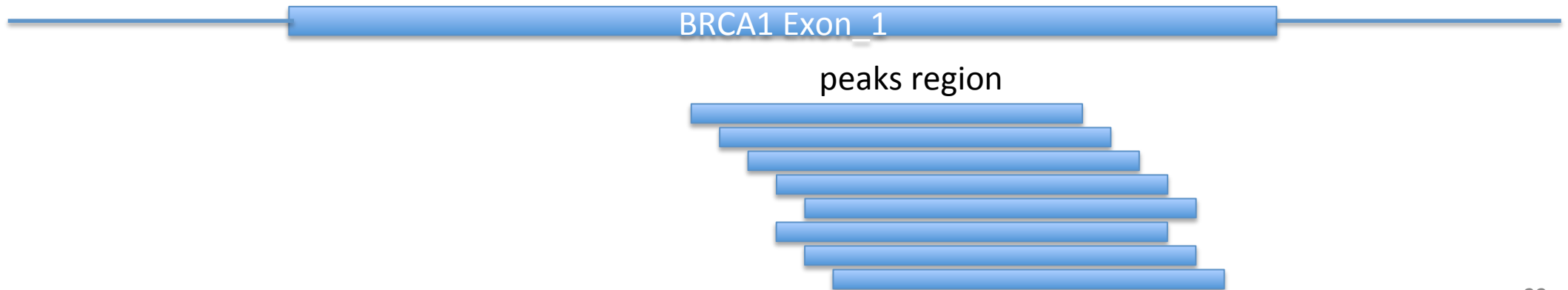
The BAM file we got

A bam file looks like below:

```
chr1      13232      33421      ATAAFCFAD  &%dfd2$@$ ...  
chr2      61323      62121      ATCTTGCGT  *&dfd2)a# ...
```

....

The bam file can be viewed through IGV viewer.
I will show you a IGV image.



Quality control

- Usually, we need to check the quality of the result bam file.
- The possible criterion include:
 - % of reads mapped to the genome.
 - % of reads mapped to the exon region.
 - % of reads mapped to the positive strand.
 - % of reads mapped to the negative strand.

From bam to bed

- Next we need to call peaks in the **bam** file. It extracted the regions that contain the reads.
- The main command is the **Clipper**.
- It also include:
- **Samtools sort** (sort the bam file by position)
- **Samtools index** (index the bam file to do fast retrieve)
- **BigbedTobed** (convert Bigbed to bed file)

The full pipeline here

- eCLIP_analysisSOP_v1.P.pdf. (the computation pipeline)
- eCLIP_SOP_v1.P_110915.pdf. (the experiment pipeline)
- I will show you the standard analysis pipeline in the supplement file.

The data we got (SRSF_rep1.bed)

chr1	15212	15250	ENSG00000227232.4_0_3	15	-	-1	-1	0.0292794233248	15230
chr1	16239	16287	ENSG00000227232.4_1_4	19	-	-1	-1	0.0104649442093	16263
chr1	16441	16485	ENSG00000227232.4_2_5	25	-	-1	-1	0.00256249972915	16462
chr1	17451	17517	ENSG00000227232.4_3_10	63	-	-1	-1	4.50322291742e-07	17481
chr1	90235	90280	ENSG00000239945.1_0_4	19	-	-1	-1	0.0105097488596	90256
chr1	90240	90275	ENSG00000238009.2_0_4	20	-	-1	-1	0.00879079737544	90256
chr1	109158	109192	ENSG00000238009.2_1_5	29	-	-1	-1	0.00115927397829	109175
chr1	113824	113881	ENSG00000238009.2_2_8	49	-	-1	-1	1.21214630854e-05	113848
chr1	115708	115784	ENSG00000238009.2_3_22	145	-	-1	-1	2.6562287759e-15	115737
chr1	116372	116428	ENSG00000238009.2_4_13	74	-	-1	-1	3.4175167505e-08	116397
chr1	135196	135226	ENSG00000237683.5_0_3	15	-	-1	-1	0.0270886550409	135213
chr1	135196	135226	ENSG00000268903.1_0_3	15	-	-1	-1	0.0274502908066	135213

Each row is a peak region. **Now we know where the RBP binds!**

Column 9 is the p-value. We can filter some peaks using p-value.

Column 4 is the Ensembl gene id. The gene of the peaks locate.

Basic quality control

- Usually, we need to check the file to ensure the file we got is right, this can be done by basic Linux command.
- Some possible criterion is:
 - The number of peaks in the files.
 - How many chromosomes in the file.
 - The number of peaks in the chromosome 1.
 - The p-values of the peaks.

Most used Linux command

#Count how many peaks in the file:

```
wc -l SRSF1_rep1.bed; (39116)
```

#Count how many peaks in chr12:

```
cut -f 1 SRSF1_rep1.bed | grep chr12 | wc -l (2133)
```

#Get the p-value column:

```
cut -f 9 SRSF1_rep1.bed > SRSF1_rep1_pvalue.txt
```

Most used Linux command

```
#how many chromosome in it
```

```
cut -f 1 SRSF1_rep1.bed | sort -u | wc -l
```

```
#remove the 'chr' in each line
```

```
cut -c 4- SRSF1_rep1.bed >SRSF1_rep1_noChr.bed
```

```
#count the peaks in each of the file and output to a new file
```

```
for i in `ls *.bed`
```

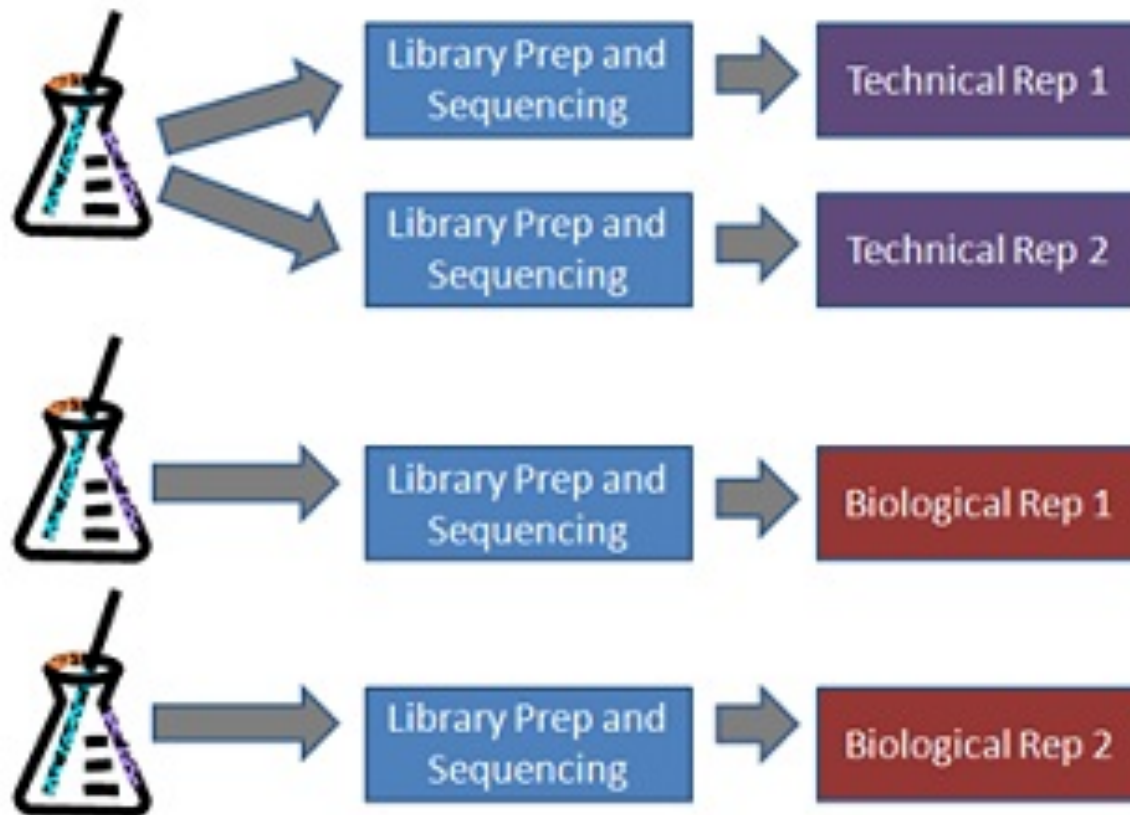
```
do
```

```
    `wc -l $i >> peaks_for_each_file.txt`;
```

```
done;
```

Biology replicate

- Usually we use at least one biological replicate to ensure the peaks we found are not due to variance.



Overlap of two biological replicates

- One possible solution is finding the overlapped regions between the two biology replicates, and **treat the overlapped region as peaks.**
- But how to find the overlapped region between two bed files?

The two bed files

Bed_rep_1

chr1 14924	14954	ENSG00000227232.4_0_3	15	-	-1	-1	0.0286089808565	14938
chr1 17454	17567	ENSG00000227232.4_1_15	95	-	-1	-1	2.52804288143e-10	17488
chr1 89861	89871	ENSG00000238009.2_0_3	16	-	-1	-1	0.020756022461	89862
chr1 89861	89871	ENSG00000239945.1_0_3	16	-	-1	-1	0.0211020769049	89862
chr1 89871	89904	ENSG00000238009.2_1_3	16	-	-1	-1	0.0246792521386	89885
chr1 89871	89904	ENSG00000239945.1_1_3	15	-	-1	-1	0.0258746724881	89885

Bed_rep_2

chr1 15212	15250	ENSG00000227232.4_0_3	15	-	-1	-1	0.0292794233248	15230
chr1 16239	16287	ENSG00000227232.4_1_4	19	-	-1	-1	0.0104649442093	16263
chr1 16441	16485	ENSG00000227232.4_2_5	25	-	-1	-1	0.00256249972915	16462
chr1 17451	17517	ENSG00000227232.4_3_10	63	-	-1	-1	4.50322291742e-07	17481
chr1 90235	90280	ENSG00000239945.1_0_4	19	-	-1	-1	0.0105097488596	90256
chr1 90240	90275	ENSG00000238009.2_0_4	20	-	-1	-1	0.00879079737544	90256

The general problem of overlapping

Here we need to find the overlap between the two bed files, actually sometimes we also need to overlap with different kinds of files.

The most common file formats we encountered are:
BAM, BED, GTF, VCF

They are all interval range formats, i.e. they all contain a chromosome name and at least a start position. A **generic solution to this problem** is needed.

Find overlap of biology replicates

- R package GenomicRanges:
- Bed file is table range format, also some other file formats like VCF, GFF, GTF. We need efficient methods to overlap between these kinds of files.
- 1. One Linux solution is using **Tabix**, while a java solution a **tribble**.
- 2. A more powerful solution is using R package **GenomicRanges**. It can efficiently map between ranges format files. [GenomicRangesIntroduction.pdf](#)

VCF, BED, GTF file format

- I will show you the above VCF, BED, GTF file formats in UCSC, and show that they are all the general chromosome, coordinate format.
- <https://genome.ucsc.edu/FAQ/FAQformat.html#format10.1>

R package GenomicRanges

To try to use it, First we need to read the file and then converting Bed, VCF, GFF, GTF to **GenomicRange** objects.
Reference:

Codes:

```
#First read bed file into table biology_rep_1
SRSF1_rep_1_bed<-read.table("Bed_rep_1")
```

Bed format is 0-based and its left coordinate is inclusive, while the right coordinate is exclusive

```
SRSF1_rep_1_range<-with(SRSF1_rep_1_bed,Granges(seqnames=chr,
+ranges=IRanges(start=start+1,end=end),strand=strand))
```

```
SRSF1_rep_2_range<-with(SRSF1_rep_2_bed,Granges(seqnames=chr,
+ranges=IRanges(start=start+1,end=end),strand=strand))
```

R package GenomicRanges

Find the peaks which overlap between the two biology replicates.

```
Codes:
##overlap the two biological replicates.
SRSF1_overlap_mch<-findoverlaps(SRSF1_rep_1_range, SRSF1_rep_2_range);
##get the overlap in replicate 1.
SRSF1_rep_1_overlap<-
SRSF1_rep_1_range[unique(queryHits(SRSF1_overlap_mch)),];
##get the overlap in replicate 2.
SRSF1_rep_2_overlap<-
SRSF1_rep_2_range[unique(subjectHits(SRSF1_overlap_mch)),];
```

What the overlap result looks like

#RBP_overlap_mtch is revord the line number of each overlap.

>RBP_overlap_mtch

	queryHits	subjectHits
[1,]	2	1
[2,]	3	1
[3,]	4	1
[4,]	5	2
[5,]	6	2

What we got currently

What we got is the overlapped peak regions between the two biological replicates.

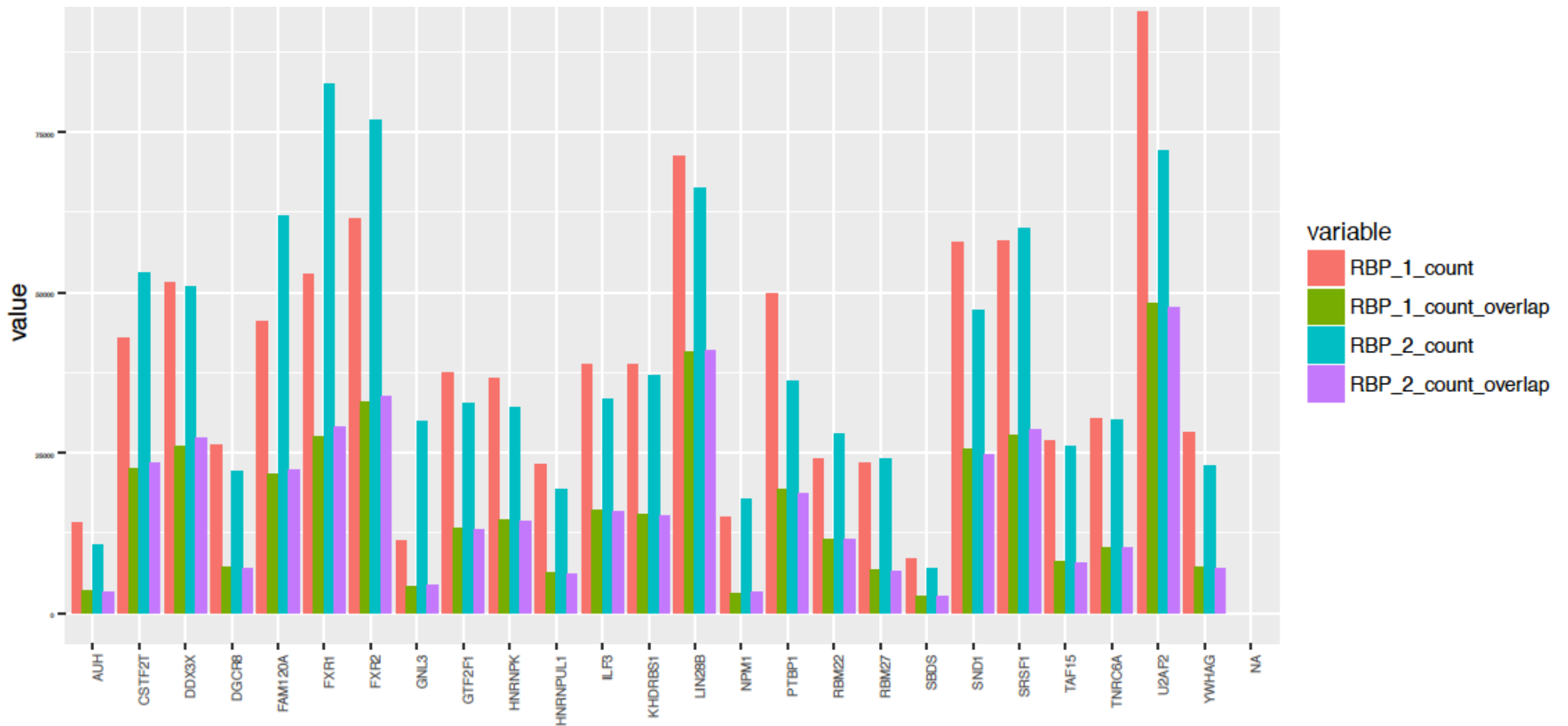
chr1	1212	1321
chr2	1312	1521
chr1	2212	3321
...		

How the result influence the result

How much the biological replicate influence the peaks region? The details of below program will be elucidate later.

```
##The number of overlapped peaks in each replicate.
overLapNum1<-length(unique(queryHits(oneMtch)));
overLapNum2<-length(unique(subjectHits(oneMtch)));
##combine the information
RBPs_replicate_overlap_frame<-c(rbpsnum1,overLapNum1,rbpsnum2,
overLapNum2)
##melt the data frame
RBPs_replicate_overlap_melt<-
melt(RBPs_replicate_overlap_frame,id.vars="RBP");
##plot
p<-ggplot(RBPs_replicate_overlap_melt, aes(x=RBP,
y=value,fill=variable)) + geom_bar(stat="identity",position="dodge")
```


How many peaks are overlapped between the two biological replicates



R package GenomicRanges

Besides overlap between two genome interval regions. GenomicRanges also include many additional functions. These include:

Set operations: ()

union, setdiff, intersect;

Interval operation: (resize the interval, change the flank region)

resize, flank, width, shift;

Overlap operations: (overlap with a region)

countOverlaps, subsetByOverlaps, findOverlaps;

Split operations: split, c(); split a GenomicRanges into multi ones.

Basic operations: tail, head, rev; get subset of the genomicRanges

Multi RBPs CLIP-seq bed files

Sometimes, we need to analysis two or more RBPs to study their interaction. We need **GRangesList** structure. Here we take HNRNPK as example.

```
##Get the overlap regions of two HNRNPK replicates
HNRNPK_rep_1_overlap<-
HNRNPK_rep_1_range[unique(queryHits(HNRNPK_overlap_mch)),];

##Combine HNRNPK and SRSF1 into a single GRangesList object
RBPs_range_list<-GRangesList(SRSF1_rep_1_overlap,
HNRNPK_rep_1_overlap);
```

Count RBP binding sites in each gene

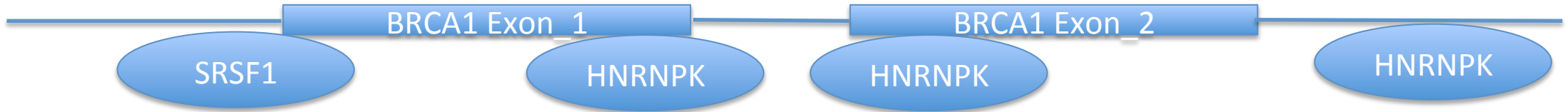
Currently we have **two RBPs (SRSF1, HNRNPK)** in a **GRangeList object**. Next, we want to count RBP binding sites in each transcript and ensure the information we got is enough to do further analysis. We need human genome annotation, here we use hg19. GTF which can be download from UCSC table Browser.

File: Table Browser

GTF is 1-based system and both the left and right coordinate are inclusive.

```
1  pseudogene gene      11869      14412 .   +   .   gene_id "ENSG00000223972";
2  processed_transcript transcript 11869 14409 .   +   .   gene_id
.....
.....
```

Count RBP binding sites in each gene



Example: BRCA1 contain 3 HNRNPK binding sites and 1 SRSF1 binding sites.

Codes:

```
##read the GTF file into R
gtf_frame<-read.table("hg19_ens.gtf",header=F,as.is=T);
##construct a GenomicRange object for the GTF
gtf_range<-with(gtf_frame,Granges(seqnames=chr,
+ranges=IRanges(start=start,end=end),strand=strand))
```

Overlap with RBPs binding sites and genome annotation

Next we overlap the `RBPs_range_list` with `gRBP_range` to get number of RBPs in each gene.

Code:

```
GeneName_RBPName<-data.frame(gene_name,RBP_name) #store the result
for(i in 1:2){##for each RBP.
  RBP_overlap_mtch<-
    findOverlaps(RBPs_range_list[[i]], gtf_range);#RBP overlap GTF
  RBP_gtf_overlap<-
    RBPs_range_list[[i]][queryHits(RBP_overlap_mtch),];#overlapped RBP

  Gtf_RBP_overlap<-
    gtf_range[[1]][subjectHits(RBP_overlap_mtch),];#overlapped GTF
```

Overlap with RBPs binding sites and genome annotation

Code:

```
gene_names<-seqnames(GRBP_RBP_overlap);#overlapped RBP names
RBP_names<-names(RBPs_range_list)[i];#overlapped gene names
GeneName_RBPName<-
  cbind(GeneName_RBPName,c(gene_names, RBP_names) );#store the result
}
```

```
>GeneName_RBPName
```

```
#each line represent a overlap
```

Gene_names	RBP_names
TP53	SFRS1
TP53	SFRS1
TP53	HNRNPK

```
... . .
```

An intro to bioconductor

- R package **GenomicRanges** is part of Bioconductor project, which contains many packages related to bioinformatics, including NGS analysis, microarray analysis and so on.
- Besides the **GenomicRanges**, another general used package I found very useful is the **BSGenome**, which provide different annotation for many genomes.
- There are also other packages in it which target on different aspect, like Chip-seq, RNA-seq (EdgeR) and so on.
- I list two papers which describe the bioconductor packages in the supplement files.

Bioconductor

Many papers are published in Bioconductor by successfully build a package in it for specific use.

R Package reshape2

Next we need to do transformation to count each gene's binding sites.

Here, we first introduce the **long table format** and **short table format**.

Long table and short table

Short Table format

Gene_name	RBP_name
TP53	HNRNPK
TP53	HNRNPK
BRCA1	SRSF1
TP53	HNRNPK
TP53	SRSF1

As you can see,
The data in the short
table is redundant. I
use more cell to store
the same data.

Column name become header

Long Table format

Gene_name	HNRNPK	SRSF1
TP53	xxx	xxx
BRCA1	xxx	xxx

The tidy data

- Usually, we think long table format is more tidy than short table format.
- **A tidy table means each row is a observation and each column is a variable.**

R package reshape2

R package reshape contain two fundamental function:

melt: Convert long table to short table.

cast: Convert short table to long table.

R package reshape2

```
>GeneName_RBPName
#each line represent a overlap
  Gene_names      RBP_names
  TP53            SFRS1
  TP53            SFRS1
  TP53            HNRNPK
```

Codes:

```
#convert the short table format to long table format
GeneName_RBPName_cast<-
Cast(GeneName_RBPName_melt, Gene_names~RBP_names, length)
#Gene_names~RBP_names is folumar, where the left side is #row id and
left side is column label
#Where length is the function to count.
```

R package reshape2

```
>GeneName_RBPName_cast
```

Gene_name	HNRNPK	SRSF1
TP53	3	1
BRCA1	1	3

#Now we have the table we need, we got each gene's RBP #binding sites for each RBP.

```
#  
#  
#  
#
```

From table to figure

Human beings can't read hundreds of rows simultaneously, we need visualization method to check the dataset.

We now have the table, we want to represent it using figures to do visualization. Maybe it is not very useful for small table, but for large table, it means a lot.

We need R package ggplot2 to convert table to figure.

R package ggplot2

ggplot2 usually receive short table format data and convert it into statistic figures. Its main advantage is productive and high versatile. It make use of **graphic grammar**.

Codes:
#ggplot2 receive short table format, we need to melt the #table.

```
>GeneName_RBPName_cast
```

Gene_name	HNRNPK	SRSF1
TP53	3	1
BRCA1	1	3

melt long table to short table format

Codes:

```
##convert the long to short format  
GeneName_RBPName_melt<-  
melt(GeneName_RBPName_cast);  
  
##melt result  
>GeneName_RBPName_melt
```

Gene_name	RBP_name	count
TP53	HNRNPK	3
TP53	SRSF1	1
BRCA1	HNRNPK	1
BRCA1	SRSF1	3

Give some ggplot figure example

Codes:

```
##count the number RBP binding in each gene's exon region.
```

```
GeneName_RBPName_point2<-ggplot(se_target_data_melt)+  
  geom_point( aes(x=gene_name,y=value,color=variable) )
```

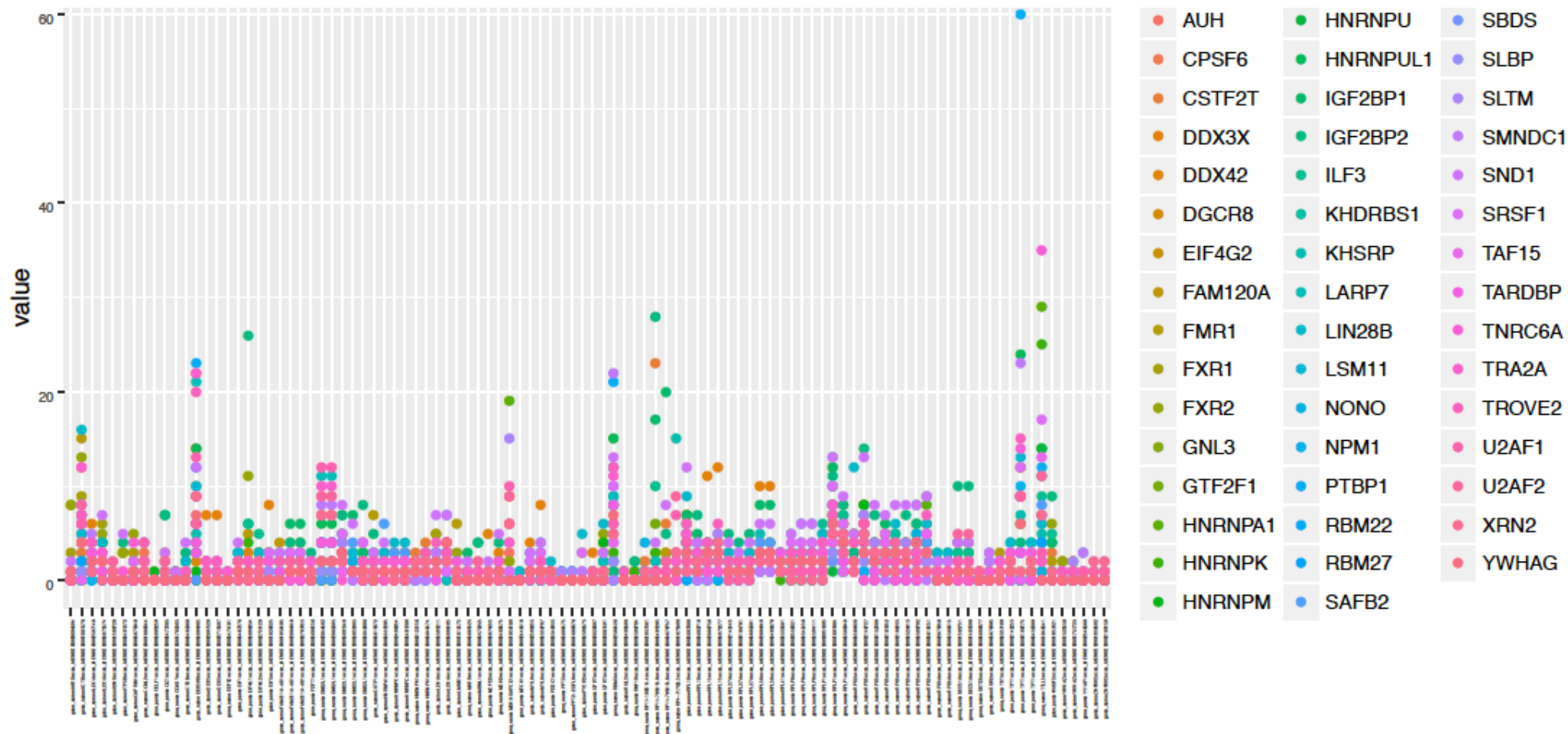
```
##plot
```

```
GeneName_RBPName_bar<-ggplot(se_target_data_melt)+  
geom_bar( aes(x=gene_name,y=value,fill=variable),stat="identity",posit  
ion="dodge" )+
```

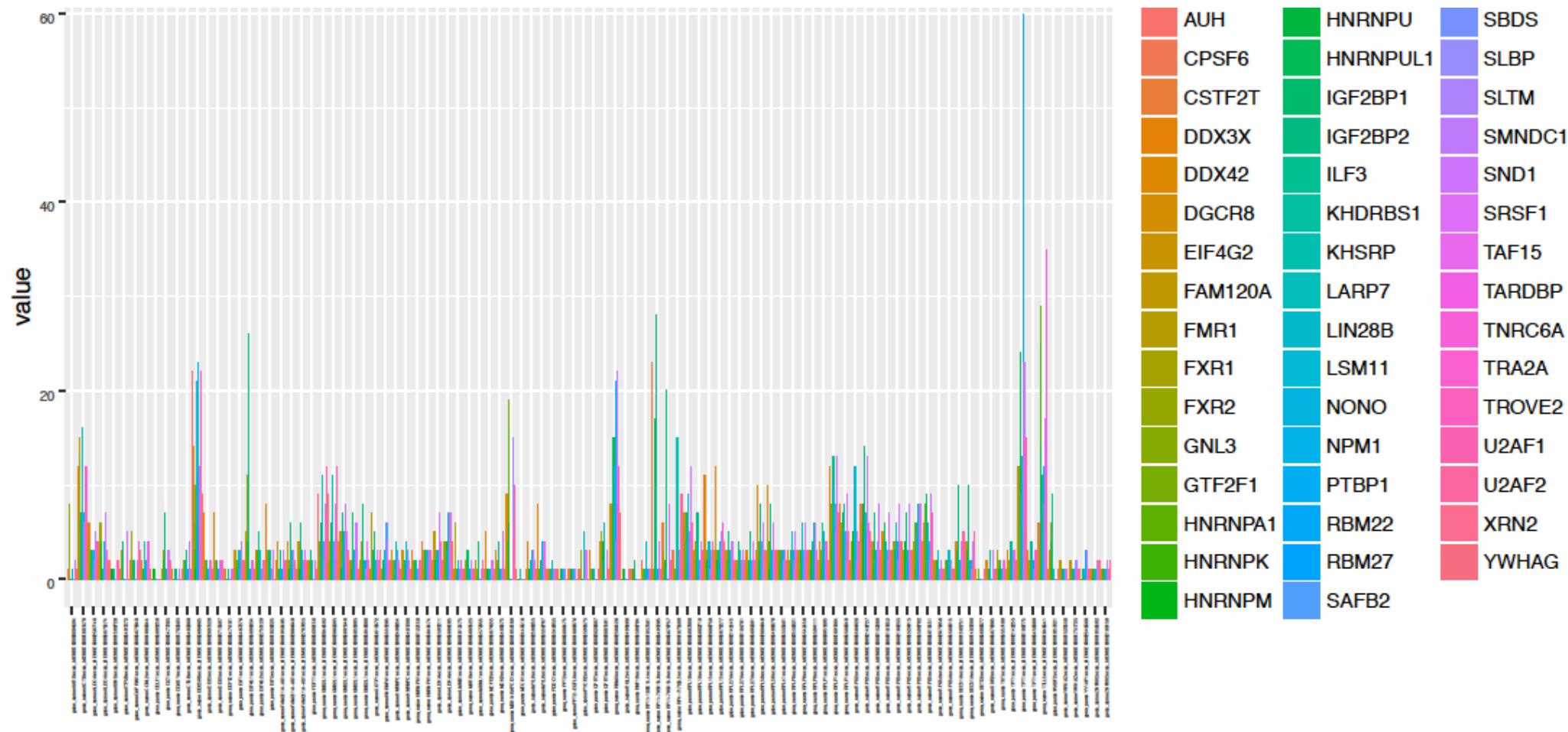
```
GeneName_RBPName_tile<-ggplot(se_target_data_melt)+  
  geom_tile( aes(x=gene_name,y=variable, fill=value) )
```

```
print(GeneName_RBPName_p1);
```

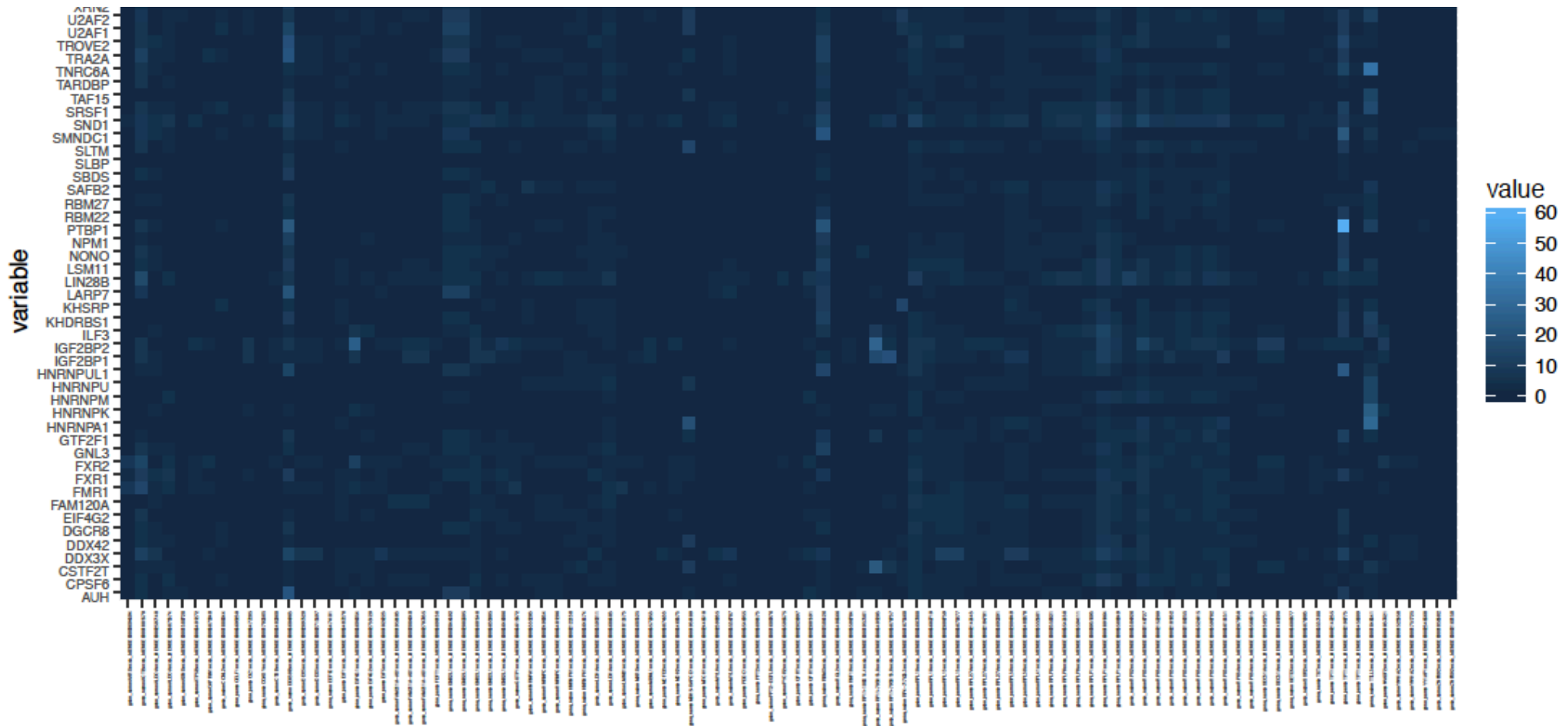
Figure_1



Figure_2



Figure_3



Which one is better?

- **I will show you the raw PDF here**, which is much clear than this one.
- Scatter plot is very easy to check which gene contain highest count, but if two RBP has same count in the gene, they will overlap. So scatter plot is wrong.
- Bar plot avoid the overlap problem by putting each point beside each other, but the 47 colors is very hard to distinguish.
- So I think the best one is the **heatmap**.

R package ggplot2

You can see the power of ggplot2. only change a few codes can do different kinds of plots, very efficient and productivity.

R package ggplot2

ggplot is multi layers figure, as you can see, each '+' add a layer to the final figure.

aes: map variable to figure variable (color, x-axis, y-axis, shape, fill).

geom: The figure type, eg. Bar_plot, histogram, scatter plot.

stat: the statistic transformation, eg. bin, identity.

R package ggplot2

- I list the figure types that support by ggplot2.
Histogram, boxplot, barplot, contour, density, scatter, tileplot, hex plot, violin, pieplot, line and so on.
- If you want to combine two or ggplot2 object in one figure, you may need R function **multiplot.r**, which can be search and got from web.

R package ggplot2

- Besides productivity, ggplot is very versatile.
- These include:
- `geom_xx` (different kinds of figures)
- `stat_xx` (different kinds of statistics transformation)
- Scales (change the scales like color, legend, fill et.al.)
- Coordinate (cartesian, polar, flip, trans)
- Faceting (multi plots in one figure)
- Position adjustments (dodge, fill, identity, stack, jitter)
- Annotation (custom, logticks, map, raster)
- Fortify
- Themes (font, size, label orientation et.al.)
- Aesthetics (map from variable to axis)
- Others ()

Calculate the number of gene for each RBP

We want to summarize which RBP binds most genes.

```
>GeneName_RBPPName
#each line represent a overlap
Gene_names      RBP_names
TP53             SRSF1
TP53             SRSF1
TP53             HNRNPK
```

Generalize the problem

The generalize of the problem is **split-aggregate** problem, what we want is split the data.frame into multi small ones, and do some operation on each group.

There is a R package called 'plyr' which can be used to do these kinds of job.

The 'for' loop equivalent

R's **for loop** is seldom used in practice. But R provide other function do to iteration:

sapply: apply a function to each element in list.

apply: apply a function to row or column.

lapply: apply a function to each element in list.

R package plyr

Although R provide some functions for doing iteration. But they are not that powerful enough for some problems, **the main problem is that the input and output format is ambiguous.** While R package plyr solved this problem.

R package plyr

It includes many functions:

Where d means table, l means list, _ means nothing, a means vector, so:

d_ply means input is data.frame, output nothing.

ddply means input is data.frame, output data.frame.

dlply means input is data.frame, output list.

daply means input is data.frame, output vector.

R package plyr

R package plyr is a more powerful solution for this kind of analysis. **It group the table and apply a function. (also called split+modify+aggregate)**

For our problem, we need to **treat each RBP a separate group and count the number of genes in each group.**

R package plyr

```
RBP_bind_fre<-ddply(GeneName_RBPName,.(RBP_names),nrow)
```

```
#each line represent a overlap
```

Gene_names	RBP_names
TP53	SRSF1
TP53	SRSF1
TP53	HNRNPK

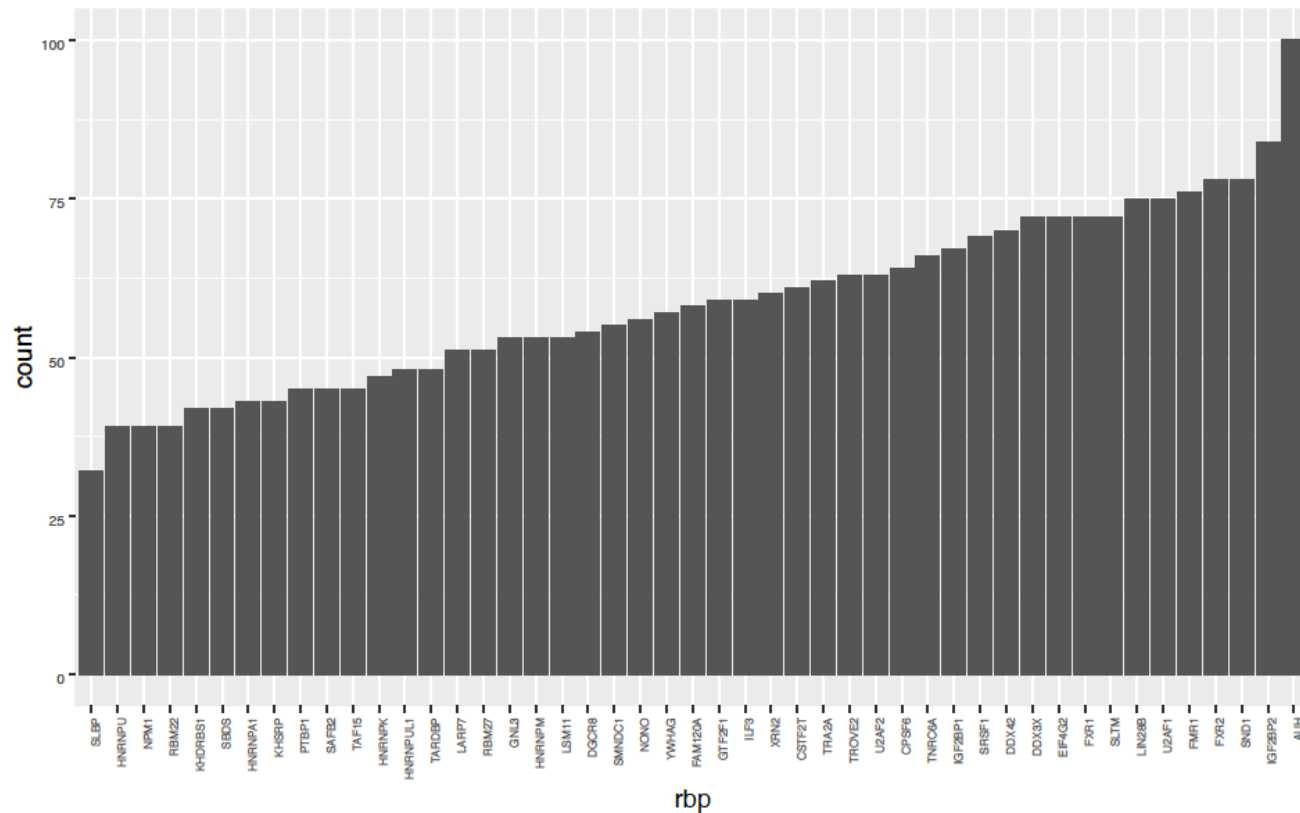
```
>RBP_bind_fre
```

RBP_names	Var1
SRSF1	2
HNRNPK	2

Calculate number of gene for each RBP

With a little ggplot2, we can do the below figure.

```
rbp_gene_count_plot<-ggplot(rbp_gene_count)+  
  geom_bar(aes(x=rbp,y=count),stat="identity")+
```



R package plyr

- Besides group analysis, R package plyr can also do .
- 1. data.frame operations, like rbind.fill, progress bar,
- 2. generalized for loop, executing a function multiple times and return something (like data.frame).

The gene expression data

- Next we add the gene expression data, which can be got from **RNA-seq technique**, and do the correlation between gene expression and RBP binding information
- The key things to know is **RNA-seq can quantatively determine the expression level of gene.**

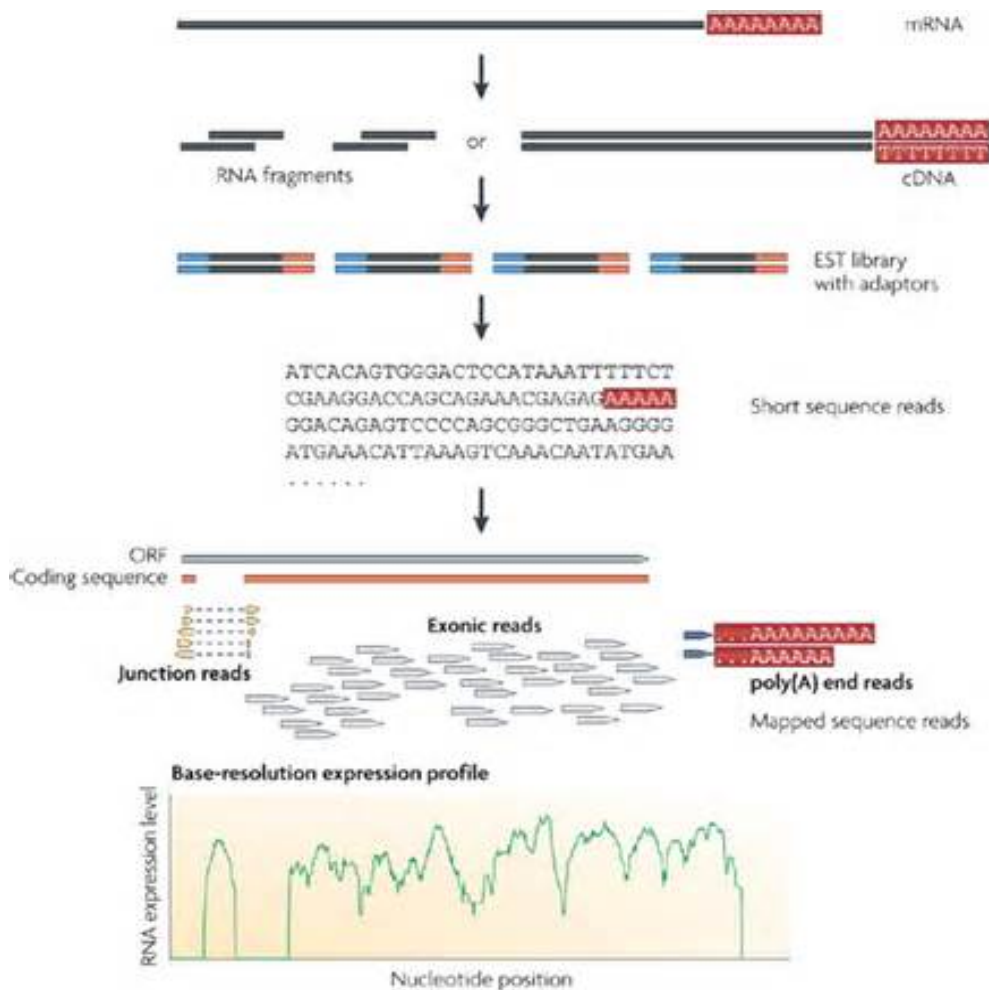
Add gene expression

Here, we use **NGSUtils** to calculate each sample's FPKM, to avoid the bias due to a single sample. We use 115 samples here and take the median FPKM as the gene's expression level.

We have 115 samples and we need to write a small **pipeline** to do it.

RNA-seq workflow

Got from Cufflinks paper



There are many kinds of RNA-seq technique which I also don't know very much.

Different sequencer will also give different result.

The RNA-seq data in Encode

The RNA-seq data set also **come from the ENCODE project**, we also assume the download format is fastq format.

Unlike sequence alignment for CLIP-seq, for RNA-seq, **we must also consider the junction reads to get the bam file.**

One software is the **tophat2**.

The RNA-seq data set

To ensure the expression data we got doesn't come from big variance, we use 104 samples' RNA-seq dataset from a same cell line (K562).

We use a tool called NGSUtils which can calculate the gene expression data from bam file.

A small Linux pipeline

After we got the bam files, we will try to call the **FPKM** values from the BAM files.

For large samples (like 104 samples here), we usually use pipeline to do batch process.

A lot of language can be used to write pipeline, here, I use bash shell, I think writing pipeline in **bash shell** is a nature way (bash.quickref.pdf).

The pipeline for **NGSUtils** here

```
for i in `ls -d dirContainBAMFiles`
do
    echo $i;
    sample=`basename $i`;
    my i_dir=${i/.bam};
    mkdir $i_dir;

    samtools sort $i $i.sort;
    samtools index $i.sort;

    script="bamutils count -gRBP Homo_sapiens.GRCh37.75_4.gRBP -rpkm -norm all $i.sort >
$sample.count";
    echo $script > $sample.count.job;

    `$script`
    #qsub -l nodes=1:ppn=1,walltime=4:00:00,mem=4gb -M li487@iupui.edu -d /N/dc2/scratch/liulab/
limeng/RBP_network/ -m ae -N $sample.count $sample.count.job
```

Process Gene expression

For each sample, now we got a **gene expression table** .

#gene	geneid	isoid	chrom	strand	txstart	txend	length	count	count (CPM)	RPKM
WASH7P	ENSG00000227232	ENSG00000227232	chr1	-	14362	29806	2073	309	1.76326905472	0.850588063058
DDX11L1	ENSG00000223972	ENSG00000223972	chr1	+	11868	14412	1756	41	0.23396126616	0.133235345194
MIR1302-10	ENSG00000243485	ENSG00000243485	chr1	+	29553	31109	1021	65	0.370914202449	0.363285212977
FAM138A	ENSG00000237613	ENSG00000237613	chr1	-	34553	36081	1219	224	1.27822740536	1.04858687889
OR4G4P	ENSG00000268020	ENSG00000268020	chr1	+	52472	54936	947	10	0.0570637234537	0.0602573637315
OR4G11P	ENSG00000240361	ENSG00000240361	chr1	+	62947	63887	940	0	0.0	0.0
OR4F5	ENSG00000186092	ENSG00000186092	chr1	+	69090	70008	918	3	0.0171191170361	0.0186482756385
RP11-34P13.7	ENSG00000238009	ENSG00000238009	chr1	-	89294	133566	3569	2050	11.698063308	3.2776865531
RP11-34P13.8	ENSG00000239945	ENSG00000239945	chr1	-	89550	91105	1319	561	3.20127488575	2.42704691869

Process Gene expression

Next, we need to **combine multi sample's gene expression into one table**. This can be done in multi ways since each file is already ordered by the gene name, including using R read each file. Use linux command cut and paste.

X.gene	ENCFF000DV	ENCFF000DV	ENCFF000DV	ENCFF000DV	ENCFF000DY	ENCFF000DY
WASH7P	148.049887	167.275891	401.776058	428.998374	19.3728753	68.0487607
DDX11L1	2.33384883	2.42390186	0	0.61342398	1.11666424	0.25012929
MIR1302-10	0.90201027	1.44107576	0.36055293	2.11003431	3.88415134	2.11511547
FAM138A	0.83104818	1.16389709	0	1.59057468	0.2938757	0
OR4G4P	0.09724947	0	0	0	0.11945795	0.23190445
OR4G11P	0	0	0	0	0.00668597	0
OR4F5	0	0	0	0	0.02738482	0
RP11-34P13.	25.8042164	24.1905387	83.6506025	74.306493	2.14483193	4.22530905
RP11-34P13.	6.07451601	8.4458933	0	0	2.5301266	0.38850003
RP11-34P13.	59.1633562	73.6970671	155.569658	144.734635	11.2270541	13.258401
RP11-34P13.	30.8065091	33.6749976	81.0133226	75.8572195	15.2915755	4.76860331
RP11-34P13.	32.7517539	46.144623	30.7176767	29.3904024	14.7089653	1.35743453
CICP27	0.06039825	0.01378484	0.19313984	0	1.07329875	1.67072191
AL627309.1	16.6470554	22.1368215	17.3617548	16.4348757	8.526187	1.07289579

Gene expression quality control

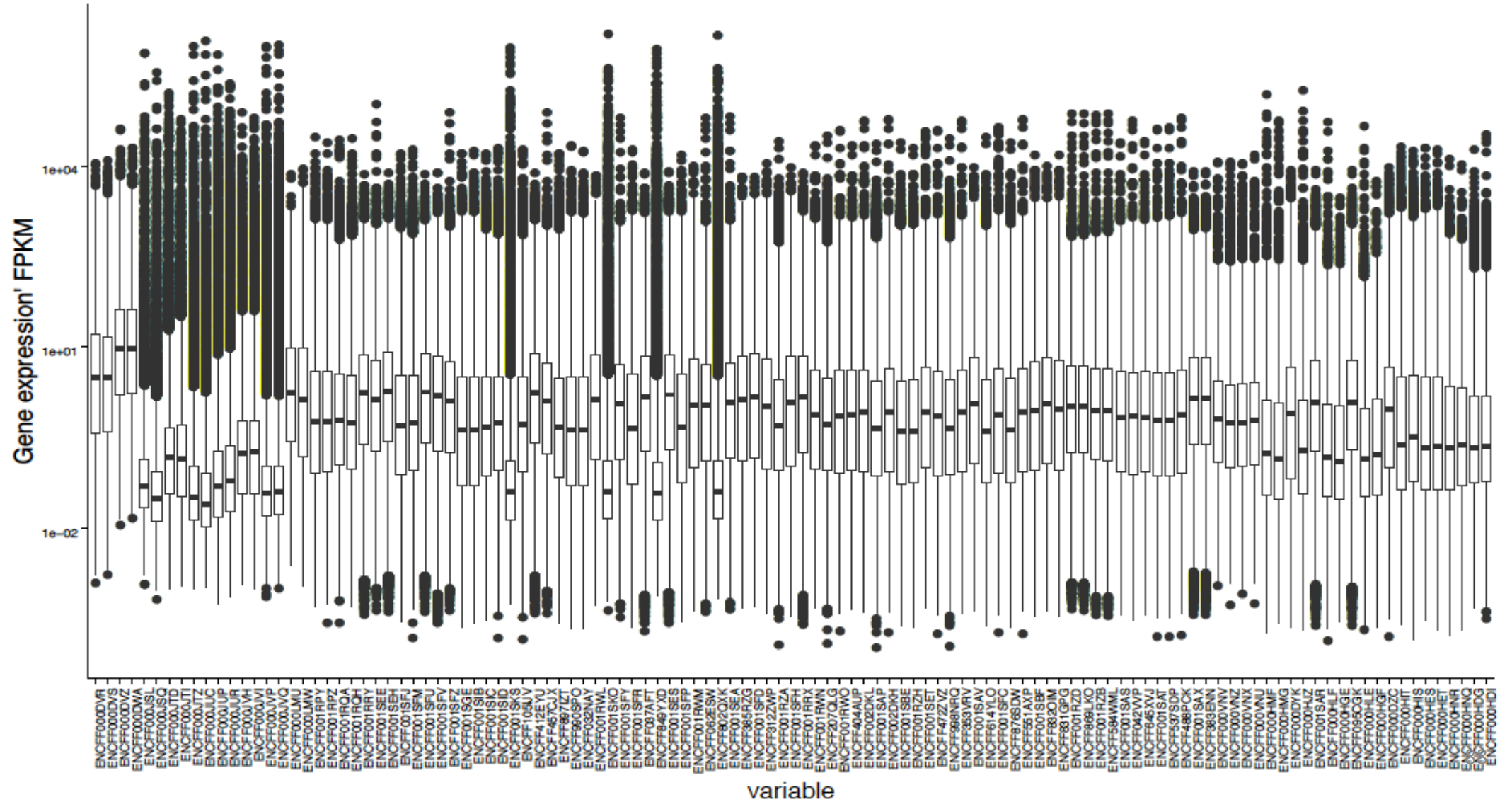
Usually, before we do a experiment, we need to validate the result we got and the feasibility of next step. (quality control)

Here we use heatmap and boxplot to do quality control.

Boxplot code

```
gene_expression_tbl<-  
read.table("gene.expression.tsv",sep="\t",as.is=TRUE);  
  
#ggplot receive short format table  
gene_expression_tbl_melt<-melt(gene_expression_tbl);  
  
gene_exp_box_plot<-ggplot(gene_expression_tbl_melt)+  
#geom_boxplot are used to plot box plot.  
  geom_boxplot(aes(x=variable, y=value  ) ) +  
  theme_classic()+scale_y_log10();
```

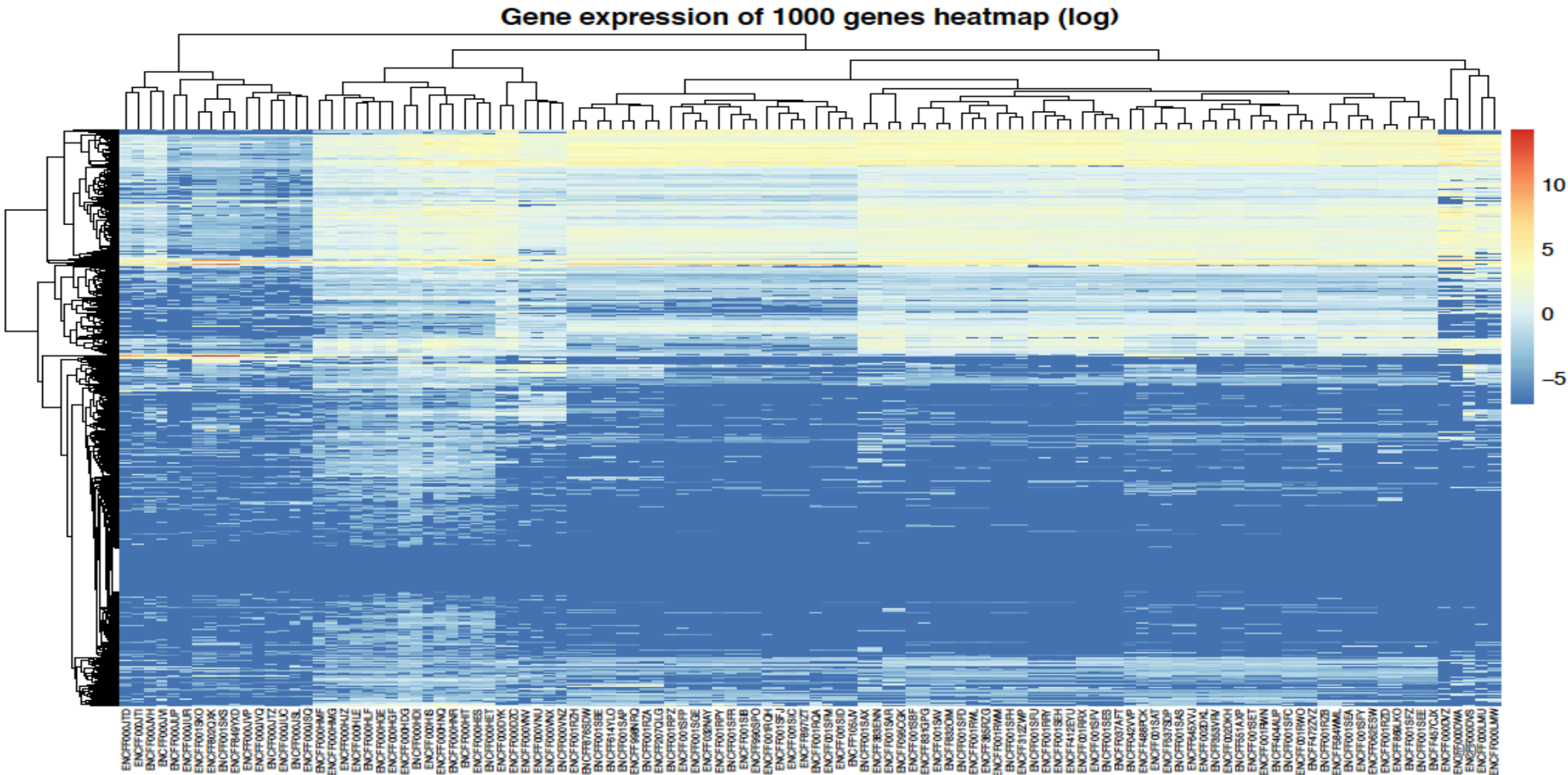
Figure



heatmap

A more generalize way to check gene expression is using heatmap, however, ggplot2 doesn't provide this, here I use R package 'pheatmap'.

Figure



What we can see from the figures?

Seems the there are **two different groups of samples**, although they come from only one cell line 'K562'. It turns out the some of the samples come from a independent culture 'K562'.

Gene Expression

Now we take the median of gene expression among samples.

And the gene expression.

Gene_name	Sample1	Sample2	Sample3	...
TP53	113.0	313	112	
BRCA1	313	113.1	132	
...				



And the gene expression.

Gene_name	Median(FPKM)	...
TP53	213.4	
BRCA1	313.5	
...		

Combine the RBP and gene expression table

Now we have a table contain RBP binding information.

Gene_name	SRSF1	BX1
TP53	1	3
BRCA1	3	1
...

And the gene expression.

Gene_name	Median(FPKM)	...
TP53	213.4	
BRCA1	313.5	
...

Generalize of the problem

- **Combine two tables** is a daily task for SQL, and it include many kinds of operations, like `left_join`, `inner_join`, `right_join` and `full_join`.
- `Left_join`: keep all the data in left table in the result table.
- `Right_join`: keep all the data in the right table in the result table.
- `Inner_join`: keep the data in both the tables.
- `Full_join`: keep the data in either the table.

R package dplyr

Here we use R package dplyr. Table manipulation language. A very powerful tool to process tables in R. It implements the above join operations in R, and make multi tables analysis more easily.

R package dplyr

Its function include:

filter – select a subset of the rows of a data frame.

arrange – works similarly to filter, except that instead of filtering or selecting rows, it reorders them.

select – select columns of a data frame.

mutate – add new columns to a data frame that are functions of existing columns.

summarize – summarize values.

group_by – describe how to break a data frame into groups of rows.

SQL JOIN – sql style joining multi tables together.

Join two tables

```
>geneName_RBPName_cast
```

Gene_name	SRSF1	BX1
TP53	1	3
BRCA1	3	1

```
>gene_expression_tbl_melt
```

Gene_name	Median(FPKM)...
TP53	213.4
BRCA1	313.5

```
Gene_RBP_exp_join<-
```

```
inner_join(geneName_RBPName_cast,  
gene_expression_tbl_melt,by=c("Gene_name"="Gene_name"));
```

Join two tables

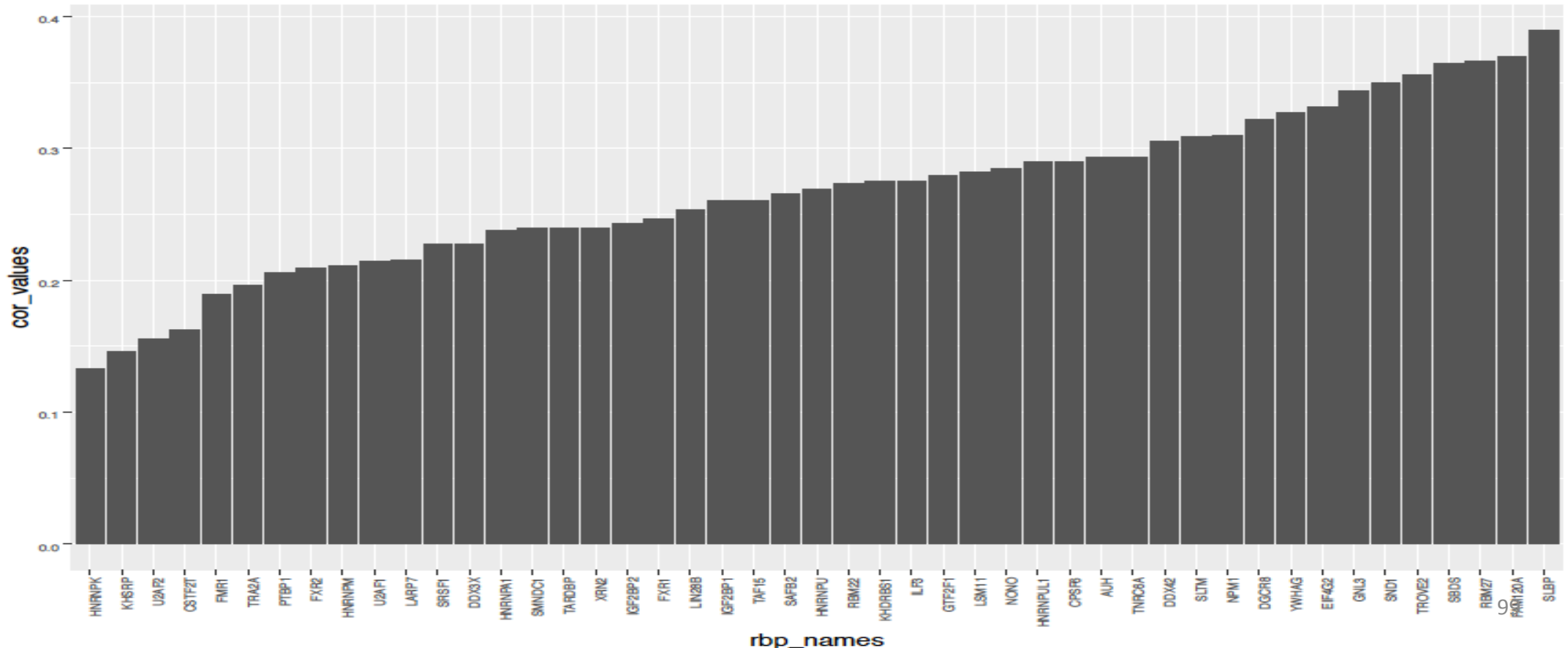
Gene_name	Median(FPKM)	SRSF1	BX1
TP53	213.4	1	3
BRCA1	313.5	3	1
...

Now we have all the data in one table.

Correlation of RBP and Gene expression

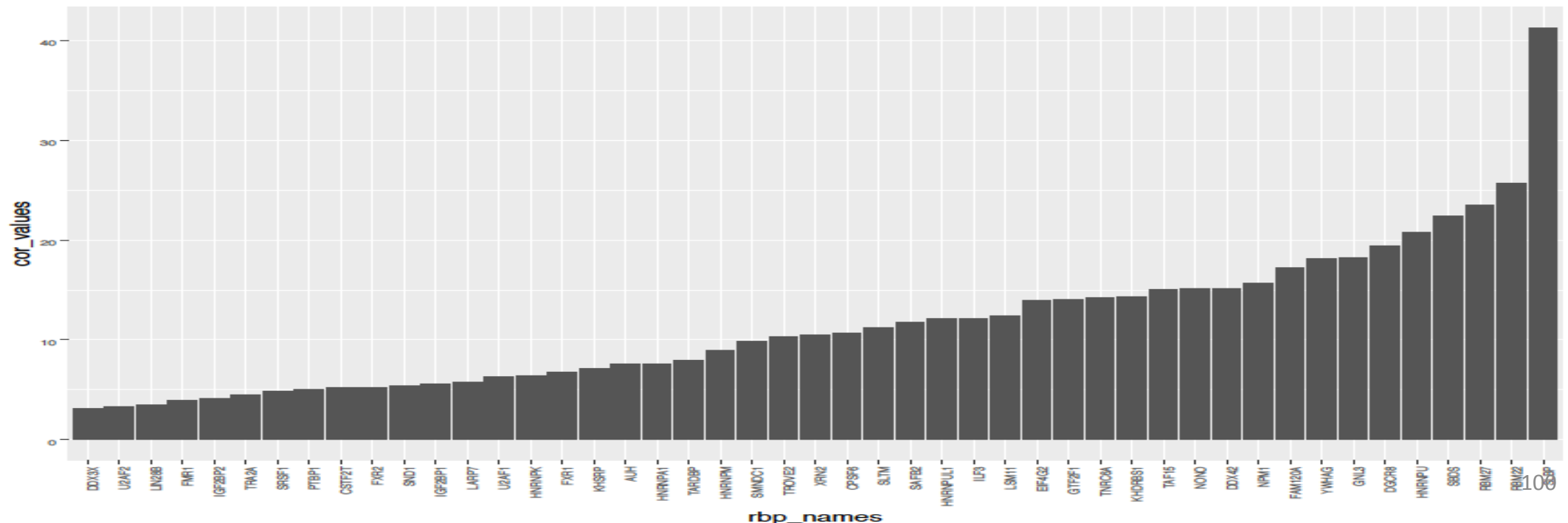
With a little ggplot:

```
rbp_cor_values_data_plot<-ggplot(rbp_exp_data)  
+geom_bar(aes(x=rbp_names,y=cor_values),stat="identity")
```



Linear regression of RBP and Gene expression

Another consideration is longer gene region has higher possibility to contain more RBPs, so we need to consider the gene's length.

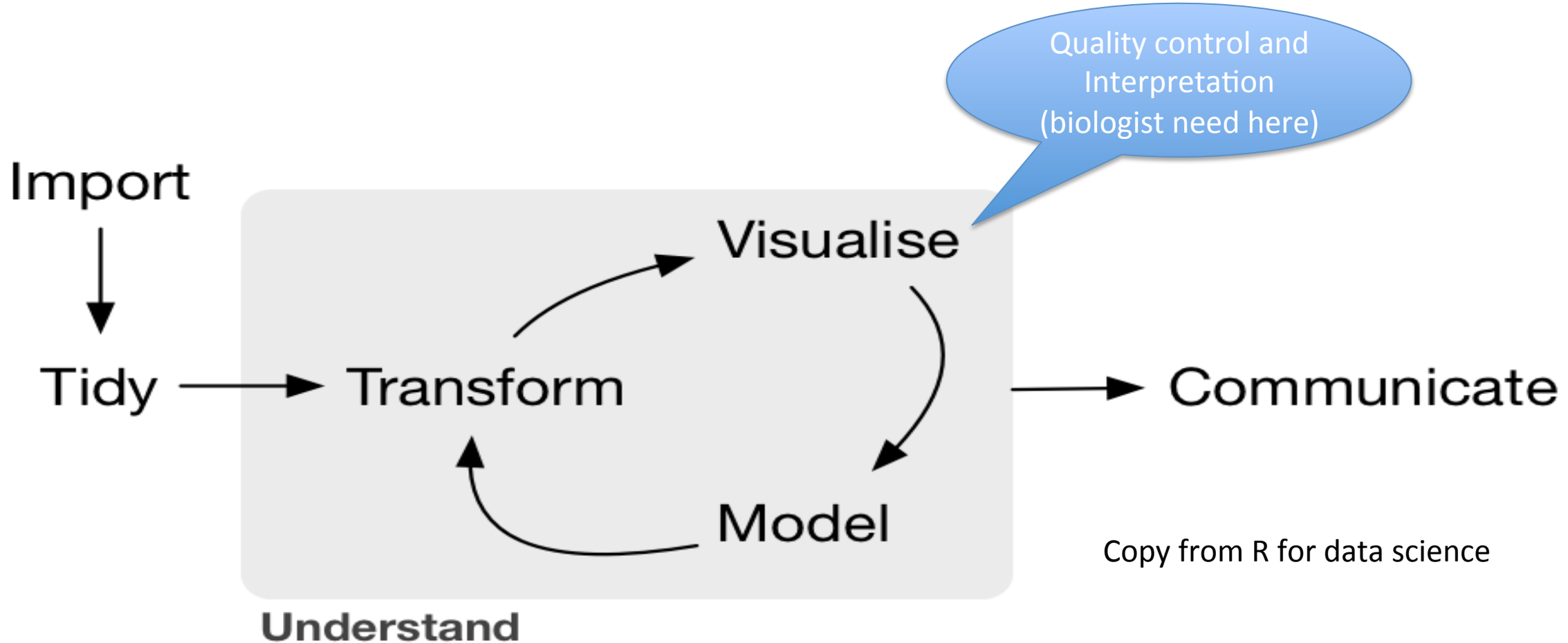


The pipe

- Usually, we will do many modification on the raw data to get the final data we need, which will make the code very complicate.
- Linux has a pipe which facilitate complicate operation. R also has a pipe command, and here it is:
`%>%`

Check the `magnitr` package for more details.

Summary: a daily life for data scientist



A few advice to newbie when do science

Actually I also am a newbie, but I also some advice to the more newer ones.

1. Tidy up the workspace of your project. (like make some folders: result, code, figures, reference, document, discussion, tables, annotation, data, et.al.)
2. Record every command ever used. (So if result go wrong, we know where we do mistake, thus can avoid make same mistake twice)
3. Double check every step!!

Further reading

<http://hadley.nz>

Advance R. <http://adv-r.had.co.nz/>

R for data science. <http://r4ds.had.co.nz/pipes.html>

ggplot2: elegant graphics for data analysis. (This one is not free)

ggplot2: http://docs.ggplot2.org/0.9.3.1/geom_bar.html

Bimедical data science: <http://genomicsclass.github.io/book/>

Bioconductor tutorial: <https://www.bioconductor.org/help/course-materials/>

I recommend read the “R for data science” first.

Acknowledgement

冯伟兴教授和其它实验室的老师 and 同学们。