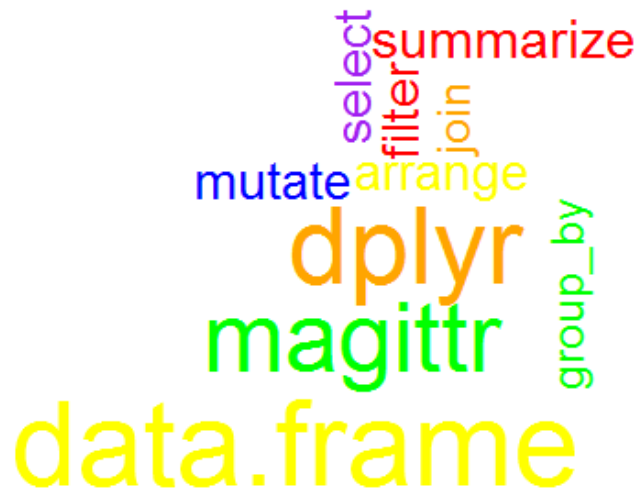


dplyr



```
wordcloud(dplyr_verb$verbs,dplyr_verb$power,random.order=T,random.color=T,colors=c("black","red","green","yellow","blue","orange","magenta","pink","purple"),rot.per=.51)
```

dplyr with pipe

```
url<-"https://sites.google.com/site/pocketecoworld/sampledata.csv"
```

```
dframe<-read.csv(url,sep="," ,head=T)
```

```
require(magrittr) # pipe %>%
```

```
require(dplyr)
```

```
dframe %>% filter( Index=='A')
```

```
# without pipe → filter(dframe,Index=='A')
```

```
dframe %>% filter(Y2005>1200000 & Index=='A')
```

```
# without pipe → filter(dframe, Y2005>1200000 & Index=='A')
```

Verbs of dplyr

filter

select

summarize

mutate

arrange

join

group_by

dplyr:select

`dframe %>% select(Index, State)`

Same as

`select(dframe,Index,State)`

Verbs of dplyr

Filter

select

summarize

Mutate

arrange

Join

Group_by

dplyr: combining verbs

```
dframe %>% filter(Y2005>1200000 & Index=='A') %>%  
select (State,Y2002,Y2007,Y2008)
```

Using magittr.pipe

Regular way

```
select (filter(dframe,Y2005>1200000 & Index=='A') State,Y2002,Y2007,Y2008)
```

mutating (changing)

```
mutate(select(filter(dframe, Index=='A'), State, Y2002, Y2004),  
+ Y2002onY2004=Y2004/Y2002-1)
```

```
dframe %>% filter(Index == 'A') %>% select (State, Y2002, Y2004) %>%  
mutate(Y2002onY2004=Y2004/Y2002-1)
```

Note that mutate creates a new variable (or column)

Select NOT commutative!!!!!! with mutate or filter than...

```
dframe %>% mutate(Y2002onY2004=Y2004/Y2002-1)  
%>% select (State, Y2002, Y2004)  
%>% filter(Index == 'A')  
## WILL FAIL ... WHY?  
## AND what happened to my new variable....
```

```
dframe %>% filter(Index == 'A') %>%  
select (State, Y2002, Y2004) %>%  
mutate(Y2002onY2004=Y2004/Y2002-1)
```

arrange:sorting

```
dframe %>% filter (Index == 'A') %>%  
arrange(Y2002) %>% select  
(State,Y2002,Y2004)
```

```
select(arrange(filter(dframe,Index == 'A'),Y2002),  
State,Y2002,Y2004)
```

```
dframe %>% filter (Index == 'A') %>%  
arrange(desc(Y2002)) %>% select  
(State,Y2002,Y2004)
```

summarize

```
summarize_at(group_by(dframe, Index), vars(Y2002:Y2004),  
  funs(n(), mean(., na.rm=TRUE)))
```

```
dframe %>% group_by (Index) %>%  
  summarize_at(vars(Y2002:Y2004),  
    funs(n(), mean(., na.rm=TRUE)))
```

```
dframe %>% filter (Index %in% c("A", "C", "I"))  
%>% group_by (Index) %>% do(head(., 2))
```

Operating on subgroups

```
t = mydata %>% select(Index, Y2015) %>%  
  filter(Index %in% c("A", "C", "I")) %>%  
  group_by(Index) %>%  
  do(arrange(., desc(Y2015)))
```

```
rkannan@F4Linux1:~  
> t = mydata %>% select(Index, Y2015) %>%  
+   filter(Index %in% c("A", "C", "I")) %>%  
+   group_by(Index) %>%  
+   do(arrange(., desc(Y2015)))  
> t  
Source: local data frame [11 x 2]  
Groups: Index [3]  
  
# A tibble: 11 x 2  
  Index      Y2015  
  <fctr>    <int>  
1     A 1979143  
2     A 1916661  
3     A 1647724  
4     A 1329341  
5     C 1718072  
6     C 1644607  
7     C 1330736  
8     I 1757171  
9     I 1641866  
10    I 1583516  
11    I 1467614
```


Loading titanic

```
require(magrittr)
require(dplyr)
```

```
titanic<-read.csv("./titanic-train.csv",head=T)
```

```
t_cc<-titanic[complete.cases(titanic),]
```

```
# what difference does it make?
```

```
dim(titanic) == dim(t_cc)
```

```
#examine
```

```
head(t_cc)
```

```
#let us find the youngest and oldest person
```

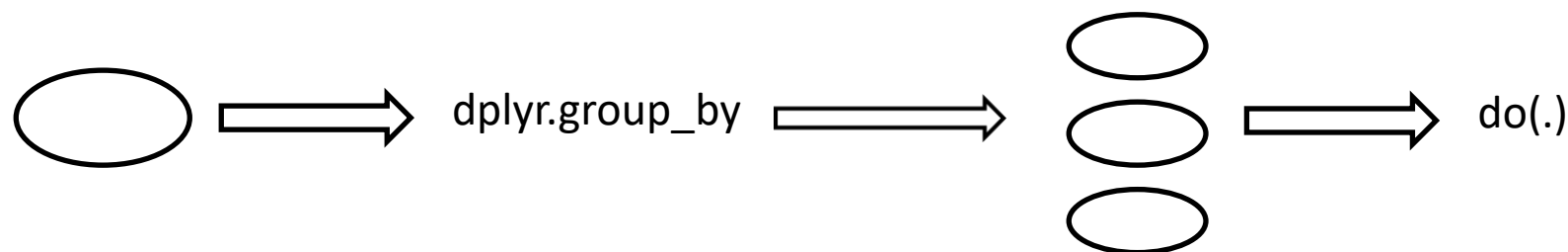
```
t_cc %>% group_by (Survived,Sex) %>%
```

```
summarize_at(vars(Survived,Sex,Age),
```

```
funs(n(),max(Age),min(Age),median(Age)))
```

```
# A tibble: 4 x 6
  Survived Sex      n    max    min median
  <int> <fctr> <int> <dbl> <dbl> <dbl>
1       0 female   64    57  2.00  24.5
2       0  male  360    74  1.00  29.0
3       1 female  197    63  0.75  28.0
4       1  male   93    80  0.42  28.0
```

Group_by: processing each subgroup



```
> titanic %>% group_by(Sex) %>% do(head(select(., Age, Sex, Name, Survived), 3))
Source: local data frame [6 x 4]
Groups: Sex [2]

# A tibble: 6 x 4
   Age    Sex Name                               Survived
<dbl> <fctr> <fctr>                               <int>
1    38 female Cumings, Mrs. John Bradley (Florence Briggs Thayer) 1
2    26 female                               Heikkinen, Miss. Laina 1
3    35 female Futrelle, Mrs. Jacques Heath (Lily May Peel) 1
4    22 male                               Braund, Mr. Owen Harris 0
5    35 male                               Allen, Mr. William Henry 0
6    NA male                               Moran, Mr. James 0
```

Join

```
df1<-data.frame(first_name =  
c("john", "bill", "madison",  
"abby", "zzz"),  
stringsAsFactors = FALSE)
```

```
df2<-data.frame(name= c("john", "bill",  
"madison", "abby", "thomas"),  
gender=c("M", "either", "M", "either",  
"M"),stringsAsFactors = FALSE)
```

```
> inner_join(df1,df2)  
Error: No common variables. Please specify `by` param.
```

```
> df3<-df2  
> names(df3)  
[1] "name" "gender"  
> names(df1)  
[1] "first_name"  
> names(df3)[1]<-names(df1)  
> names(df3)  
[1] "first_name" "gender"  
> inner_join(df1,df3)  
Joining, by = "first_name"  
  first_name gender  
1      john      M  
2      bill either  
3    madison      M  
4      abby either
```

We will keep df2 for other experiments.

We will copy df2 to df3 and make the names of df3 same as df1 and then do the inner_join

A data science brain must ask many questions at this time!!!

Join

```
df1<-data.frame(first_name =  
c("john", "bill", "madison",  
"abby", "zzz"),  
stringsAsFactors = FALSE)
```

```
df2<-data.frame(name= c("john", "bill",  
"madison", "abby", "thomas"),  
gender=c("M", "either", "M", "either",  
"M"),stringsAsFactors = FALSE)
```

```
> inner_join(df1,df2)  
Error: No common variables. Please specify `by` param.
```

```
> inner_join(df1,df2,by=c("first_name" ="name"))  
  first_name gender  
1      john      M  
2      bill either  
3    madison      M  
4      abby either  
>
```

Join

```
df1<-data.frame(first_name =  
c("john", "bill", "madison",  
"abby", "zzz"),  
stringsAsFactors = FALSE)
```

```
df2<-data.frame(name= c("john", "bill",  
"madison", "abby", "thomas"),  
gender=c("M", "either", "M", "either",  
"M"),stringsAsFactors = FALSE)
```

```
> left_join(df1,df2,by=c("first_name" = "name"))  
  first_name gender  
1      john      M  
2      bill either  
3    madison      M  
4      abby either  
5       zzz  <NA>  
> left_join(df1,df2,c("first_name" = "name"))  
  first_name gender  
1      john      M  
2      bill either  
3    madison      M  
4      abby either  
5       zzz  <NA>
```

Join

```
df1<-data.frame(first_name =  
c("john", "bill", "madison",  
"abby", "zzz"),  
stringsAsFactors = FALSE)
```

```
df2<-data.frame(name= c("john", "bill",  
"madison", "abby", "thomas"),  
gender=c("M", "either", "M", "either",  
"M"),stringsAsFactors = FALSE)
```

```
> right_join(df1,df2,by=c("first_name" ="name"))  
  first_name gender  
1      john      M  
2      bill either  
3    madison      M  
4      abby either  
5    thomas      M  
>
```

Random → uncorrelated

```
x<-rnorm(100)
y<-rnorm(100)
```

What levels of correlation is acceptable between X and Y given they were randomly generated?

```
cor(x,y)
[1] 0.1480642
```

Is 14% correlation acceptable?
0% is perfectly uncorrelated.

```
LMOBJ<-lm(y~x)
```

R functions return something...mostly gets evaluated by the R-Shell. Default evaluation is to print it to the console. Assigning to LMOBJ prevents further evaluation.

```
> lm(y~x)
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)          x
    0.09244      0.17713

> invisible(lm(y~x))
>
```

Here lm returns an object and it is evaluated. Printing partially.

We can suppress it with invisible.

Inside R objects

You just downloaded a great new package and you want to find what the package functions return. Let us start with good old OLS function `lm`.

```
x<-rnorm(100)
```

```
y<-rnorm(100)
```

R functions return something...mostly gets evaluated by the R-Shell. Default evaluation is to print it to the console. Assigning to `LMOBJ` prevents further evaluation.

```
LMOBJ<-lm(y~x)
```

```
> lm(y~x)
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)          x
    0.09244      0.17713

> invisible(lm(y~x))
>
```

Here `lm` returns an object and it is evaluated. Printing partially.

We can suppress it with `invisible`.

Inside an lm object:str(lmobj)

```
> str(LMOBJ)
List of 12
 $ coefficients : Named num [1:2] 0.0924 0.1771
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "x"
 $ residuals    : Named num [1:100] -0.125 0.736 -0.959 -2.665 -0.209 ...
  ..- attr(*, "names")= chr [1:100] "1" "2" "3" "4" ...
 $ effects      : Named num [1:100] -1.056 1.645 -0.763 -2.581 -0.257 ...
  ..- attr(*, "names")= chr [1:100] "(Intercept)" "x" "" "" ...
 $ rank         : int 2
 $ fitted.values: Named num [1:100] 0.2392 0.0797 0.5325 0.2821 -0.0165 ...
  ..- attr(*, "names")= chr [1:100] "1" "2" "3" "4" ...
 $ assign       : int [1:2] 0 1
 $ qr          : List of 5
  ..$ qr       : num [1:100, 1:2] -10 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:100] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "(Intercept)" "x"
  .. ..- attr(*, "assign")= int [1:2] 0 1
  ..$ graux: num [1:2] 1.1 1.02
  ..$ pivot: int [1:2] 1 2
  ..$ tol   : num 1e-07
  ..$ rank  : int 2
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 98
 $ xlevels      : Named list()
 $ call         : language lm(formula = y ~ x)
 $ terms        :Classes 'terms', 'formula' language y ~ x
  .. ..- attr(*, "variables")= language list(y, x)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
```


`LMOBJ$residuals == (LMOBJ$model$y - LMOBJ$fitted.values)`

Lm returns a list of 12 objects

```
$ call      : language lm(formula = y ~ x)
$ terms     :Classes 'terms', 'formula' language y ~ x
.. ..- attr(*, "variables")= language list(y, x)
.. ..- attr(*, "factors")= int [1:2, 1] 0 1
.. ..- attr(*, "dimnames")=List of 2
.. ..$ : chr [1:2] "y" "x"
.. ..$ : chr "x"
.. ..- attr(*, "term.labels")= chr "x"
.. ..- attr(*, "order")= int 1
.. ..- attr(*, "intercept")= int 1
.. ..- attr(*, "response")= int 1
.. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
.. ..- attr(*, "predvars")= language list(y, x)
.. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
.. ..- attr(*, "names")= chr [1:2] "y" "x"
$ model     :'data.frame': 100 obs. of 2 variables:
..$ y: num [1:100] 0.114 0.816 -0.426 -2.383 -0.225 ...
..$ x: num [1:100] 0.828 -0.072 2.484 1.071 -0.615 ...
..- attr(*, "terms")=Classes 'terms', 'formula' language y ~ x
.. ..- attr(*, "variables")= language list(y, x)
.. ..- attr(*, "factors")= int [1:2, 1] 0 1
.. ..- attr(*, "dimnames")=List of 2
.. ..$ : chr [1:2] "y" "x"
.. ..$ : chr "x"
.. ..- attr(*, "term.labels")= chr "x"
.. ..- attr(*, "order")= int 1
.. ..- attr(*, "intercept")= int 1
.. ..- attr(*, "response")= int 1
.. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
.. ..- attr(*, "predvars")= language list(y, x)
.. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
.. ..- attr(*, "names")= chr [1:2] "y" "x"
- attr(*, "class")= chr "lm"
```

- [1] "coefficients"
- [2] "residuals"
- [3] "effects"
- [4] "rank"
- [5] "fitted.values"
- [6] "assign"
- [7] "qr"
- [8] "df.residual"
- [9] "xlevels"
- [10] "call"
- [11] "terms"
- [12] "model"

Basics in R:dissecting R Objects

```
> uLMOBJ<-unclass(LMOBJ)
> names(uLMOBJ)
 [1] "coefficients"  "residuals"      "effects"        "rank"
 [5] "fitted.values" "assign"         "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
> uLMOBJ$coefficients
(Intercept)          x
 0.09244066  0.17712573
> uLMOBJ$coefficients[[1]]
[1] 0.09244066
> uLMOBJ$coefficients[1]
(Intercept)
 0.09244066
> 
```

Dissecting lm

```
> names(LMOBJ)
[1] "coefficients" "residuals"      "effects"        "rank"
[5] "fitted.values" "assign"         "qr"            "df.residual"
[9] "xlevels"      "call"          "terms"         "model"
> 
```

```
> typeof(LMOBJ)
[1] "list"
> LMOBJ[[1]]
(Intercept)          x
0.09244066  0.17712573
```

SVD

```
r1<-c(3, -1, 2, -5, 4, 1)
r2<-c(4, 2, -3, -1, 1, 3)
r3<-c(-1, 5, 4, 2, -3, -2)
r4<-c(3, 1, 2, -3, 0, -3)
r5<-c(-5, 4, -2, 2, -5, -1)
```

```
> m2
      [,1] [,2] [,3] [,4] [,5] [,6]
r1      3   -1    2   -5    4    1
r2      4    2   -3   -1    1    3
r3     -1    5    4    2   -3   -2
r4      3    1    2   -3    0   -3
r5     -5    4   -2    2   -5   -1

> svd_m2<-svd(m2)
> str(svd_m2)
List of 3
 $ d: num [1:5] 12.43 7.56 5.71 3.28 2.66
 $ u: num [1:5, 1:5] -0.546 -0.235 0.424 -0.173 0.661 ...
 $ v: num [1:6, 1:5] -0.549 0.376 -0.029 0.455 -0.563 ...

> ??svd
> svd_m2$d
[1] 12.427510  7.561830  5.706228  3.278029  2.658798
> svd_m2$u %*% svd_m2$d %*% t(svd_m2$v)
Error in svd_m2$u %*% svd_m2$d %*% t(svd_m2$v) :
  non-conformable arguments
> svd_m2$u %*% diag(svd_m2$d) %*% t(svd_m2$v)
      [,1] [,2] [,3] [,4]      [,5] [,6]
[1,]      3   -1    2   -5  4.000000e+00    1
[2,]      4    2   -3   -1  1.000000e+00    3
[3,]     -1    5    4    2 -3.000000e+00   -2
[4,]      3    1    2   -3 -2.220446e-16   -3
[5,]     -5    4   -2    2 -5.000000e+00   -1

> round(svd_m2$u %*% diag(svd_m2$d) %*% t(svd_m2$v))
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]      3   -1    2   -5    4    1
[2,]      4    2   -3   -1    1    3
[3,]     -1    5    4    2   -3   -2
[4,]      3    1    2   -3    0   -3
[5,]     -5    4   -2    2   -5   -1

> |
```