Ve572 Lecture 7

Manuel and Jing

UM-SJTU Joint Institute

June 7, 2018

- We have only dealt with small datasets, for which efficiency is not an issue.

```
> system.time({
+   n = 1e3       # number of data points to load
+   dota2items.df =
+     read.table("~/Desktop/purchase_log.csv",
+     sep = ",", header = TRUE,
+     nrows = n) # Max rows
+
+   item.name.df =
+     read.table("~/Desktop/item_ids.csv",
+                sep = ",", header = TRUE,
+                stringsAsFactors = FALSE)
+ })
```

```
   user  system elapsed
  0.004   0.001   0.003
```

```
> str(dota2items.df, vec.len = 1)
```

```
'data.frame':    1000 obs. of  4 variables:
 $ item_id    : int  44 29 ...
 $ time       : int  -81 -63 ...
 $ player_slot: int  0 0 ...
 $ match_id   : int  0 0 ...
```

```
> nrow(item.name.df); head(item.name.df)
```

```
[1] 189
  item_id          item_name
1       1              blink
2       2  blades_of_attack
3       3          broadsword
4       4          chainmail
5       5           claymore
6       6 helm_of_iron_will
```

- We need to be careful when the dataset becomes even moderately big!

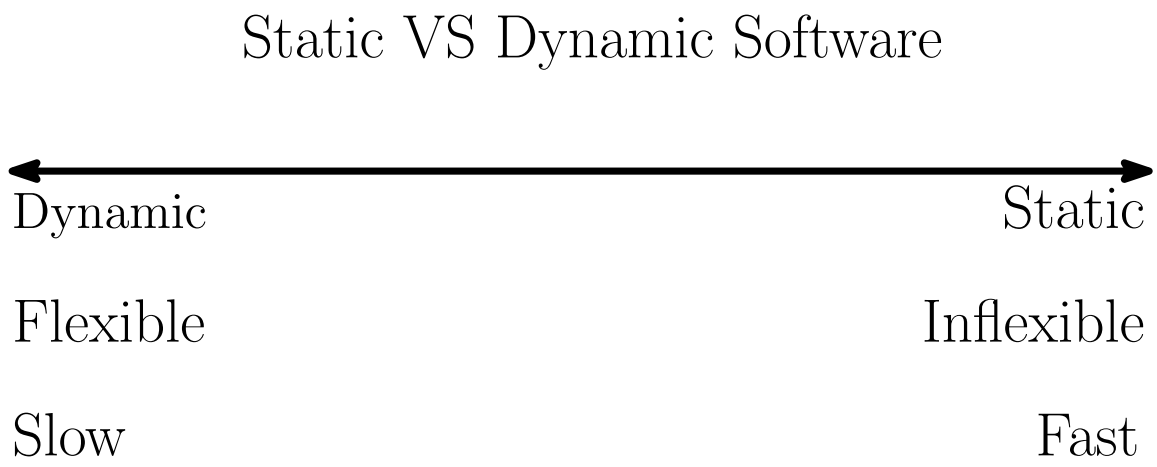
```
> system.time({
+   dota2items.df = read.table(
+     "~/Desktop/purchase_log.csv",
+     sep = ",", header = TRUE,
+     nrows = 1e7)
+ })
```

```
   user  system elapsed
 20.874   0.283  21.791
```

```
> format(object.size(dota2items.df),
+        units = "auto")
```

```
[1] "152.6 Mb"
```

- R was originally designed to be extremely dynamic and flexible.



  which means there are various ways of doing the same job in R.

- Currently, there are three main approaches of manipulating data in R.
  - base
  - dplyr
  - data.table
- We will cover data.table next, which is designed to be efficient for big data.

```
> library(data.table)
```

- Before data.table, here is a textbook example of trading flexibility for speed

```
> m = 1e5; rbenchmark::benchmark(
+   "1.Slow for loop" = {
+     x = NULL
+     for (i in 1:m){x[i] = sqrt(i)}
+   },
+   "2.Preallocation" = {
+     x = double(m);
+     for (i in 1:m){x[i] = sqrt(i)}
+   },
+   "3.Vectorisation" = {
+     x = sqrt(1:m)
+   }, replications = 5, order = "relative",
+   columns = c("test", "replications",
+               "elapsed", "relative"))
```

	test	replications	elapsed	relative
3	3.Vectorisation	5	0.003	1
2	2.Preallocation	5	0.414	138
1	1.Slow for loop	5	102.528	34176

```
> rbenchmark::benchmark( # Recall this has 189 rows
+   "Base" = {
+     item.name.df =
+       read.table("~/Desktop/item_ids.csv",
+                  sep = ",", header = TRUE,
+                  stringsAsFactors = FALSE)
+   },
+   "data.table" = {
+     item_name_dt =
+       fread("~/Desktop/item_ids.csv",
+             sep = ",", header = TRUE,
+             stringsAsFactors = FALSE)
+   }, replications = 5, order = "relative",
+   columns = c("test", "replications",
+               "elapsed", "relative"))
```

	test	replications	elapsed	relative
2	data.table	5	0.001	1
1	Base	5	0.004	4

```
> rbenchmark::benchmark( # This has 18,193,745 rows
+   "Base" = {             # Significantly bigger!
+     dota2items.df =
+       read.table("~/Desktop/purchase_log.csv",
+                  sep = ",", header = TRUE)
+   },
+   "data.table" = {
+     dota2items_dt =
+       fread("~/Desktop/purchase_log.csv",
+             sep = ",", header = TRUE)
+   },
+   replications = 5, order = "relative",
+   columns = c("test", "replications",
+               "elapsed", "relative"))
```

	test	replications	elapsed	relative
2	data.table	5	10.427	1.000
1	Base	5	205.571	19.715

```
> rbenchmark::benchmark( # sort the data frame
+   "Base" = {
+     order.base.df =
+       dota2items.df[order(dota2items.df$match_id,
+                           dota2items.df$time,
+                           decreasing = TRUE), ]
+   },

+   "data.table" = {
+     order_dt =
+       dota2items_dt[order(-match_id,-time)]
+   },
+   replications = 5, order = "relative",
+   columns = c("test", "replications",
+               "elapsed", "relative"))
```

	test	replications	elapsed	relative
2	data.table	5	5.237	1.000
1	Base	5	11.140	2.127

```
> head(order.base.df)
```

	item_id	time	player_slot	match_id
18193441	147	2849	1	49999
18193549	158	2768	4	49999
18193482	96	2760	2	49999
18193480	58	2742	2	49999
18193481	24	2742	2	49999
18193548	55	2742	4	49999

```
> head(order_dt)
```

	item_id	time	player_slot	match_id
1:	147	2849	1	49999
2:	158	2768	4	49999
3:	96	2760	2	49999
4:	58	2742	2	49999
5:	24	2742	2	49999
6:	55	2742	4	49999

```
> rbenchmark::benchmark( # Subset columns
+   "Base" = {
+     col.base.df =
+       order.base.df[, !names(order.base.df)
+                       %in% c("player_slot")]
+   },

+   "data.table" = {
+     col_dt = order_dt[, !"player_slot"]
+   },
+   replications = 5, order = "relative",
+   columns = c("test", "replications",
+               "elapsed", "relative"))
```

	test	replications	elapsed	relative
1	Base	5	0.001	1
2	data.table	5	0.229	229

```
> rbenchmark::benchmark( # Add a column
+   "Base" = {
+     col.base.df$time_r =
+       rank(order.base.df$time)
+   },

+   "data.table" = {
+     col_dt[, time_r := rank(time)]
+   },
+   replications = 5, order = "relative",
+   columns = c("test", "replications",
+               "elapsed", "relative"))
```

	test	replications	elapsed	relative
1	Base	5	46.444	1.000
2	data.table	5	48.019	1.034

```
> rbenchmark::benchmark( # Subset rows
+   "Base" = {
+     row.base.df =
+       dota2items.df[dota2items.df$player_slot == 0
+                     & dota2items.df$time>0, ]
+   },

+   "data.table" = {
+     row_dt =
+       dota2items_dt[player_slot == 0 & time>0 ]
+   },
+   replications = 5, order = "relative",
+   columns = c("test", "replications",
+               "elapsed", "relative"))
```

	test	replications	elapsed	relative
2	data.table	5	1.578	1.000
1	Base	5	2.457	1.557

```
> rbenchmark::benchmark(
+   "Base" = {
+     item.counts.df =
+       aggregate(time~item_id,
+                 data = row.base.df,
+                 FUN = length)
+     item.median.time.df =
+       aggregate(time~item_id,
+                 data = row.base.df,
+                 FUN = median)
+     item.summary.base.df =
+       merge(item.name.df,
+             item.counts.df,
+             by = "item_id")
+     item.summary.base.df =
+       merge(item.summary.base.df,
+             item.median.time.df,
+             by = "item_id")
```

```
+       colnames(item.summary.base.df)[-1:-2] =
+         c("counts", "median_time")
+     },
+     "data.table" = {
+       item_counts_dt =
+         row_dt[, length(time), by = item_id ]
+       colnames(item_counts_dt)[2] = "counts"
+       item_median_time_dt =
+         row_dt[, .(median_time = median(time)),
+                  by = item_id ]
+       item_summary_dt =
+         merge(item_name_dt, item_counts_dt,
+                  by = "item_id")
+       item_summary_dt =
+         merge(item_summary_dt, item_median_time_dt,
+                  by = "item_id")
+     },
+     replications = 5, order = "relative",
+     columns = c("test", "replications",
+                 "elapsed", "relative"))
```

```
        test replications elapsed relative
2 data.table            5   0.256    1.000
1       Base            5  16.817   65.691
```

```
> head(item_summary_dt)
```

```
   item_id           item_name counts median_time
1:       1               blink  14740      1173.0
2:       2  blades_of_attack  26309       518.0
3:       3         broadsword  10834      1445.0
4:       4          chainmail  14214      1310.5
5:       5           claymore   8365      1056.0
6:       6 helm_of_iron_will   6008      1056.0
```

- Another problem with large datasets in R:

  R objects live in memory entirely

  which means R reads Data into RAM all at once!
- This feature improves speed until there is NOT enough memory.
- Solution:
  1. Subset data before loading into R
  2. Workarounds within R (< 10GB)
  3. Connect and interact with database within R (> 10GB)
  4. Hadoop and Spark (> 10GB)

```
> # Trade flexibility/Convenience for speed

> flights.df =
+   read.table("flights14.csv", header = TRUE,
+              sep = ",", stringsAsFactors = FALSE)

> classes.vec =
+   c(rep("integer", 8), rep("character", 2),
+     "integer", rep("character", 2),
+     rep("integer", 4))

> # read.table specifying colClasses
> flights.df =
+   read.table("flights14.csv", header = TRUE,
+              sep = ",", stringsAsFactors = FALSE,
+              colClasses = classes.vec)
```

	test	replications	elapsed	relative
2	With colClasses	10	19.005	1.000
1	Without colClasses	10	21.310	1.121

- That said, `fread` does everything automatically,

```
> library(data.table)
> flights.df =
+   fread("flights14.csv", data.table = FALSE)
```

and is much faster, the trade-off is flexibility for it can only read regular csv.

	test	replications	elapsed	relative
3	fread	10	1.460	1.000
2	With colClasses	10	19.521	13.371
1	Without colClasses	10	21.838	14.958

```
> flights_DT =
+   fread("flights14.csv", data.table = TRUE)
```

	test	replications	elapsed	relative
1	fread data frame	10	1.381	1.000
2	fread data table	10	1.454	1.053

- A data table is also an enhance data frame, it is a data frame with a pointer

```
> DT = data.table( ID = c("a","b","a"), No = 1:3,
+                  Mon = c("Jan","Jan", "Feb"))
```

```
> attributes(DT)
```

```
$names
[1] "ID"  "No"  "Mon"

$row.names
[1] 1 2 3

$class
[1] "data.table" "data.frame"

$.internal.selfref
<pointer: 0x10200df78>
```

- There is not much difference in printing,

```
> DT
```

```
   ID No Mon
1:  a  1 Jan
2:  b  2 Jan
3:  a  3 Feb
```

```
> # Things that are the same
> DT[[1]]; DT[["No"]]; DT$Mon; DT[1,]
```

```
[1] "a" "b" "a"
[1] 1 2 3
[1] "Jan" "Jan" "Feb"
   ID No Mon
1:  a  1 Jan
```

- But the syntax within the frame of data table, i.e. `[...]`, is vastly different.

- Selecting elements

```
> df = data.frame( ID = c("a","b","a"), No = 1:3,
+                  Mon = c("Jan","Jan", "Feb"),
+                  stringsAsFactors = FALSE)
> df
```

```
  ID No Mon
1  a  1 Jan
2  b  2 Jan
3  a  3 Feb
```

```
> # Data frame drops its structure by default
> df[2,3]
```

```
[1] "Jan"
```

```
> # Data table retains its structure by default
> DT[2,3]
```

```
   Mon
1: Jan
```

- Selecting columns

```
> # Data frame drops its structure by default
> df[, 1]; df[, "ID"]
```

```
[1] "a" "b" "a"
[1] "a" "b" "a"
```

```
> # Data table retains its structure by default
> DT[, 1]; DT[, "ID"]
```

```
   ID
1:  a
2:  b
3:  a
   ID
1:  a
2:  b
3:  a
```

```
> # Selecting columns using names without ""
> DT[, ID]
```

```
[1] "a" "b" "a"
```

```
> # .() is an alias to list()
> DT[, .(ID)]
```

```
   ID
1:  a
2:  b
3:  a
```

```
> # Renaming while selecting columns
> DT[, .(colID = ID, colNo = No)]
```

```
   colID colNo
1:     a     1
2:     b     2
3:     a     3
```

```
> df[1:2] # Selecting columns
```

```
  ID No
1  a  1
2  b  2
3  a  3
```

```
> DT[1:2] # Selecting rows
```

```
   ID No Mon
1:  a  1 Jan
2:  b  2 Jan
```

```
> # grepl on column names
> DT[, names(DT) %like% "o", with = FALSE]
```

```
   No Mon
1:  1 Jan
2:  2 Jan
3:  3 Feb
```

- Subsetting or filtering rows

```
> # Identical to the syntax in data frame
> DT[ ID == "a" & No == 3, ]
```

```
   ID No Mon
1:  a  3 Feb
```

```
> # Subsetting rows is assumed without the comma
> DT[ ID == "a" & No == 3]
```

```
   ID No Mon
1:  a  3 Feb
```

- Notice data table always gives new row names, while data frame gives old

```
> df[df$ID == "a" & df$No == 3, ] # Need the comma
```

```
  ID No Mon
3  a  3 Feb
```

- Sort

```
> # by ID in descending order,
> # and then by No in ascending order
> DT[order(-ID, No)]
```

```
   ID No   M
1:  b  2 Jan
2:  a  1 Jan
3:  a  3 Feb
```

- In addition to pointers, data.table uses binary search algorithm to speed up

```
> setkey(DT, ID, No) # the followings are equivalent
> DT[.("a", 3)]; DT[ ID == "a" & No == 3]
```

```
   ID No Mon
1:  a  3 Feb
   ID No Mon
1:  a  3 Feb
```

- In general, `j` in `DT[i, j, by]` is not necessary the column index.

```
> str(flights_DT)
```

```
Classes ?data.table? and 'data.frame':  253316 obs. of  17 variables:
 $ year      : int  2014 2014 2014 2014 2014 2014 2014 2014 2014 2014 ...
 $ month     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ day       : int  1 1 1 1 1 1 1 1 1 1 ...
 $ dep_time  : int  914 1157 1902 722 1347 1824 2133 1542 1509 1848 ...
 $ dep_delay: int  14 -3 2 -8 2 4 -2 -3 -1 -2 ...
 $ arr_time  : int  1238 1523 2224 1014 1706 2145 37 1906 1828 2206 ...
 $ arr_delay: int  13 13 9 -26 1 0 -18 -14 -17 -14 ...
 $ cancelled: int  0 0 0 0 0 0 0 0 0 0 ...
 $ carrier   : chr  "AA" "AA" "AA" "AA" ...
 $ tailnum   : chr  "N338AA" "N335AA" "N327AA" "N3EHAA" ...
 $ flight    : int  1 3 21 29 117 119 185 133 145 235 ...
 $ origin    : chr  "JFK" "JFK" "JFK" "LGA" ...
 $ dest      : chr  "LAX" "LAX" "LAX" "PBI" ...
 $ air_time  : int  359 363 351 157 350 339 338 356 161 349 ...
 $ distance  : int  2475 2475 2475 1035 2475 2454 2475 2475 1089 2422 ...
 $ hour      : int  9 11 19 7 13 18 21 15 15 18 ...
 $ min       : int  14 57 2 22 47 24 33 42 9 48 ...
 - attr(*, ".internal.selfref")=<externalptr>
```

- Compute or do `j`

```
> # How many trips have had total delay < 0
> flights_DT[, sum((arr_delay + dep_delay) < 0)]
```

```
[1] 141814
```

- Subsetting in `i` and do in `j`

```
> # Compute the average arrival and departure delay
> # for all flights with JFK as the origin in June.
> flights_DT[origin == "JFK" & month == 6L,
+            .(m_arr = mean(arr_delay),
+              m_dep = mean(dep_delay))]
```

```
      m_arr    m_dep
1: 5.839349 9.807884
```

- The followings are the same

```
> # How many trips have been made in 2014
> # from JFK airport in the month of June
> flights_DT[origin == "JFK" & month == 6L, .N]
>
> # The function length() requires an input argument
> flights_DT[origin == "JFK" & month == 6L,
+            length(dest)]
```

```
[1] 8422
```

- Grouping using by in DT[i, j, by]

```
> flights_DT[, .(.N), by = .(origin)]
```

```
   origin     N
1:    JFK 81483
2:    LGA 84433
3:    EWR 87400
```

- Multiple grouping variables

```
> head(flights_DT[, .N, by = .(origin,dest)])
```

```
   origin dest     N
1:    JFK  LAX 10208
2:    LGA  PBI  2307
3:    EWR  LAX  4226
4:    JFK  MIA  2750
5:    JFK  SEA  1815
6:    EWR  MIA  2094
```

- Subsetting in `i`, computing in `j` and grouping in `by`

```
> head(flights_DT[carrier == "AA",
+                 .(marrdel = mean(arr_delay),
+                   mdepdel = mean(dep_delay)),
+                 by = .(origin, dest, month)], 3)
```

```
   origin dest month    marrdel    mdepdel
1:    JFK  LAX     1   6.590361 14.2289157
2:    LGA  PBI     1 -7.758621  0.3103448
3:    EWR  LAX     1   1.366667  7.5000000
```

- Statements in `by`

```
> flights_DT[, .N, .(dep_delay > 0, arr_delay > 0)]
```

```
   dep_delay arr_delay      N
1:      TRUE      TRUE  72836
2:     FALSE      TRUE  34583
3:     FALSE     FALSE 119304
4:      TRUE     FALSE  26593
```

Q: What does the following do?

```
> DT[, print(.SD), by = ID]
```

```
   No  Mon
1:  1  Jan
2:  3  Feb
   No  Mon
1:  2  Jan
Empty data.table (0 rows) of 1 col: ID
```

Q: What does the following mean?

```
> DT[, lapply(.SD, mean),
+     by = ID, .SDcols = c("No")]
```

```
   ID  No
1:  a   2
2:  b   2
```

Reshaping

- Consider the following

```
> tmp = "family_id age_mother dob_child1 dob_child2 dob_child3
+ 1          30 1998-11-26 2000-01-29          NA
+ 2          27 1996-06-22          NA          NA
+ 3          26 2002-07-11 2004-04-05 2007-09-02
+ 4          32 2004-10-10 2009-08-27 2012-07-21
+ 5          29 2000-12-05 2005-02-28          NA"
```

```
> (DT = fread(tmp))
```

```
   family_id age_mother dob_child1 dob_child2 dob_child3
1:         1          30 1998-11-26 2000-01-29          NA
2:         2          27 1996-06-22          NA          NA
3:         3          26 2002-07-11 2004-04-05 2007-09-02
4:         4          32 2004-10-10 2009-08-27 2012-07-21
5:         5          29 2000-12-05 2005-02-28          NA
```

```
> str(DT)
```

```
Classes ?data.table? and 'data.frame':  5 obs. of  5 variables:
 $ family_id : int  1 2 3 4 5
 $ age_mother: int  30 27 26 32 29
 $ dob_child1: chr  "1998-11-26" "1996-06-22" "2002-07-11" "2004-10-10" ...
 $ dob_child2: chr  "2000-01-29" NA "2004-04-05" "2009-08-27" ...
 $ dob_child3: chr  NA NA "2007-09-02" "2012-07-21" ...
 - attr(*, ".internal.selfref")=<externalptr>
```

- Convert to long form

```
> cols = c("dob_child1", "dob_child2", "dob_child3")
> head({DT_m1 = melt(DT, measure.vars = cols,
+                    variable.name = "child",
+                    value.name = "dob")})
```

```
   family_id age_mother      child        dob
1:         1         30 dob_child1 1998-11-26
2:         2         27 dob_child1 1996-06-22
3:         3         26 dob_child1 2002-07-11
4:         4         32 dob_child1 2004-10-10
5:         5         29 dob_child1 2000-12-05
6:         1         30 dob_child2 2000-01-29
```

- Convert to wide form

```
> dcast(DT_m1,
+       family_id + age_mother ~ child,
+       value.var = "dob")
```

```
   family_id age_mother dob_child1 dob_child2 dob_child3
1:         1         30 1998-11-26 2000-01-29         NA
2:         2         27 1996-06-22         NA         NA
3:         3         26 2002-07-11 2004-04-05 2007-09-02
4:         4         32 2004-10-10 2009-08-27 2012-07-21
5:         5         29 2000-12-05 2005-02-28         NA
```

- There are people running R with data.table on servers with 1TB of memory.
- However, most of us don't have 1TB of memory readily available:

```
> library(ff)

> library(ffbase)

> # Specify a path to a folder
> # to store the binary file created by R
> setwd("~/Desktop/")
>
> system("mkdir ffdf")
>
> options(fftempdir = "~/Desktop/ffdf")
```

- The data we are going to use is from the Bureau of Transportation Statistics,

`flights_sep_oct15.txt`

it contains all flights to and from all American airports in Sept. - Oct. 2015.

```
> system.time({
+   flights.ff =
+     read.table.ffdf(file="flights_sep_oct15.txt",
+                     sep = ",", VERBOSE = TRUE,
+                     header = TRUE, next.rows = 1e5,
+                     colClasses = NA)})
```

```
read.table.ffdf 1..100000 (100000)  csv-read=3.928sec ffdf-write=0.54sec
read.table.ffdf 100001..200000 (100000)  csv-read=4.146sec ffdf-write=0.401sec
read.table.ffdf 200001..300000 (100000)  csv-read=4.141sec ffdf-write=0.401sec
read.table.ffdf 300001..400000 (100000)  csv-read=4.15sec ffdf-write=0.406sec
read.table.ffdf 400001..500000 (100000)  csv-read=4.245sec ffdf-write=0.401sec
read.table.ffdf 500001..600000 (100000)  csv-read=4.224sec ffdf-write=0.396sec
read.table.ffdf 600001..700000 (100000)  csv-read=4.211sec ffdf-write=0.405sec
read.table.ffdf 700001..800000 (100000)  csv-read=4.128sec ffdf-write=0.399sec
read.table.ffdf 800001..900000 (100000)  csv-read=4.145sec ffdf-write=0.405sec
read.table.ffdf 900001..951111 (51111)  csv-read=2.264sec ffdf-write=0.225sec
 csv-read=39.582sec  ffdf-write=3.979sec  TOTAL=43.561sec
   user  system elapsed
 40.019   1.413  43.574
```

```
> ncol(flights.ff)
```

```
[1] 28
```

```
> system.time({ airlines.ff =
+   read.csv.ffdf(file = "airline_id.csv",
+                 VERBOSE = TRUE, header = TRUE,
+                 next.rows = 1e5, colClasses = NA)})
```

```
read.table.ffdf 1..1607 (1607)  csv-read=0.03sec ffdf-write=0.017sec
 csv-read=0.03sec  ffdf-write=0.017sec  TOTAL=0.047sec
   user  system elapsed
  0.031   0.004   0.049
```

```
> names(airlines.ff)
```

```
[1] "Code"        "Description"
```

```
> names(flights.ff)
```

```
 [1] "YEAR"              "MONTH"             "DAY_OF_MONTH"      "DAY_OF_WEEK"
 [5] "FL_DATE"           "UNIQUE_CARRIER"    "AIRLINE_ID"        "TAIL_NUM"
 [9] "FL_NUM"            "ORIGIN_AIRPORT_ID" "ORIGIN"            "ORIGIN_CITY_NAME"
[13] "ORIGIN_STATE_NM"   "ORIGIN_WAC"        "DEST_AIRPORT_ID"   "DEST"
[17] "DEST_CITY_NAME"    "DEST_STATE_NM"     "DEST_WAC"          "DEP_TIME"
[21] "DEP_DELAY"         "ARR_TIME"          "ARR_DELAY"         "CANCELLED"
[25] "CANCELLATION_CODE" "DIVERTED"          "AIR_TIME"          "DISTANCE"
```

```
> names(airlines.ff) = c("AIRLINE_ID", "AIRLINE_NM")
```

- Join the two datasets

```
> flights.data.ff =
+   merge.ffdf(flights.ff, airlines.ff,
+              by="AIRLINE_ID")

> class(flights.data.ff)
```

```
[1] "ffdf"
```

```
> dim(flights.data.ff)
```

```
[1] 951111     29
```

```
> #The new object is only 551.2 Kb in size
> object.size(flights.data.ff)
```

```
562144 bytes
```

- If we load the whole dataset into the memory,

```
> airlines.df = read.csv("airline_id.csv",
+                        header = TRUE)
> system.time({
+   flights.df =
+     read.table("flights_sep_oct15.txt",
+                sep = ",", header = TRUE)})
```

```
   user  system elapsed
 37.871   0.877  39.438
```

it will be slightly faster at this scale than reading into `ffdf`

```
   user  system elapsed
 40.019   1.413  43.574
```

```
> names(airlines.df) = c("AIRLINE_ID", "AIRLINE_NM")
> flights.df = merge(flights.df, airlines.df,
+                    by="AIRLINE_ID")
```

```
> #The new object is already 105.7 Mb in size
> #A rapid spike in RAM use when processing
```

```
> object.size(flights.df)
110803296 bytes
```

```
> rbenchmark::benchmark(
+    "ff" = {
+      origin_st=unique(flights.data.ff$ORIGIN_STATE_NM)
+    },
+    "base" = {
+      origin.st=unique(flights.df$ORIGIN_STATE_NM)
+    },
+    replications = 10, order = "relative",
+    columns = c("test", "replications",
+                "elapsed", "relative")
+ )
```

```
  test replications elapsed relative
2 base           10   0.167    1.000
1   ff           10   0.560    3.353
```

- Basic modelling can be done using base func together with `ff` and `ffbase`.

```
> flights.2008.data = read.csv.ffdf(
+    file="2008.csv.bz2",header=TRUE,VERBOSE=TRUE)
```

```
csv-read=353.575sec  ffdf-write=25.144sec  TOTAL=378.719sec
```

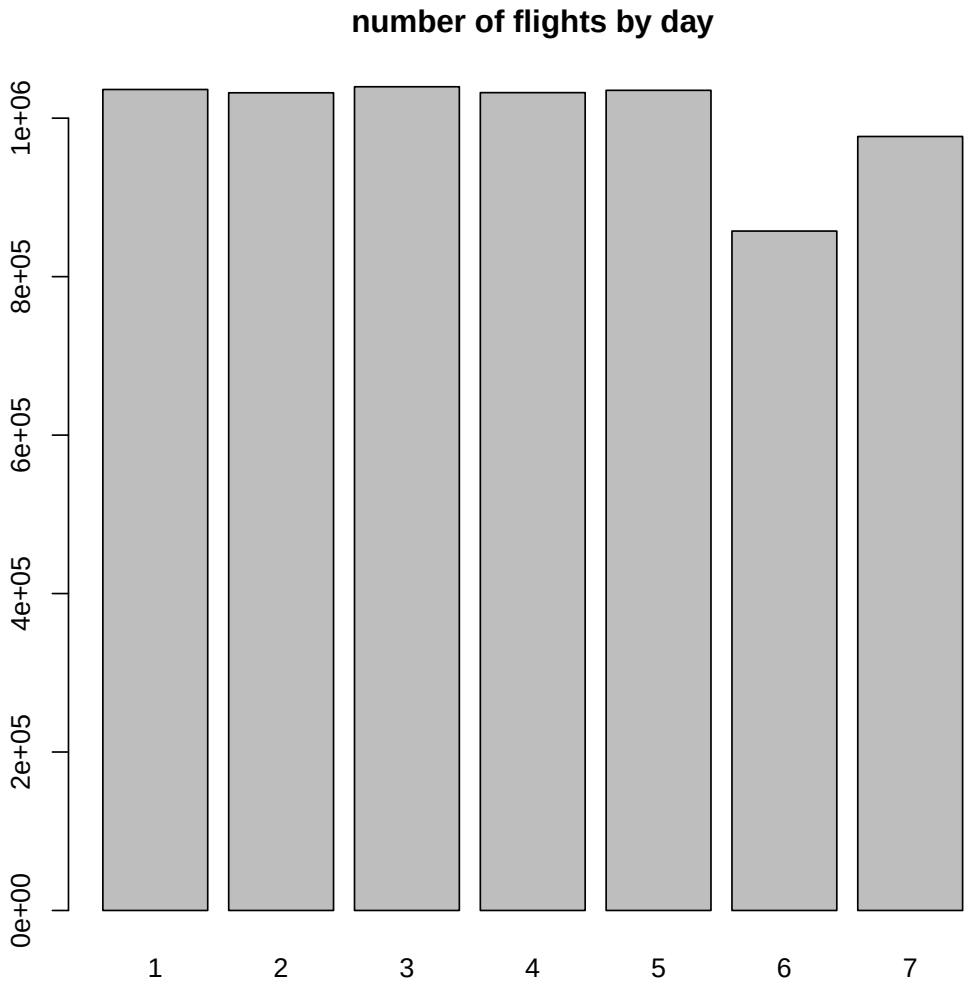
```
> dim(flights.2008.data)
```

```
[1] 7009728      29
```

```
> names(flights.2008.data)
```

```
 [1] "Year"              "Month"             "DayofMonth"        "DayOfWeek"
 [5] "DepTime"           "CRSDepTime"        "ArrTime"           "CRSArrTime"
 [9] "UniqueCarrier"     "FlightNum"         "TailNum"           "ActualElapsedTime"
[13] "CRSElapsedTime"    "AirTime"           "ArrDelay"          "DepDelay"
[17] "Origin"            "Dest"              "Distance"          "TaxiIn"
[21] "TaxiOut"           "Cancelled"         "CancellationCode"  "Diverted"
[25] "CarrierDelay"      "WeatherDelay"      "NASDelay"          "SecurityDelay"
[29] "LateAircraftDelay"
```

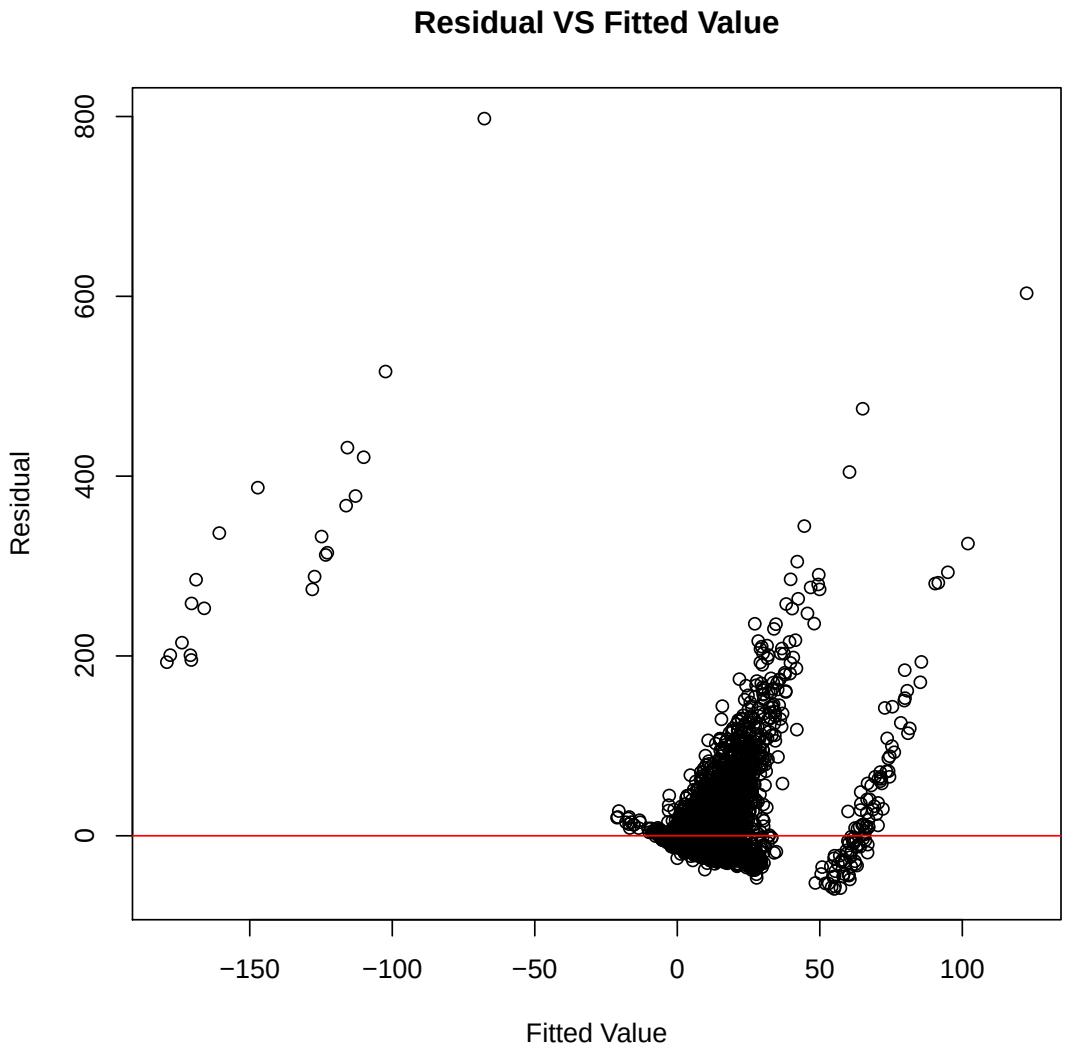
```
> barplot(table.ff(flights.2008.data$DayOfWeek),
+           main="number of flights by day")
```



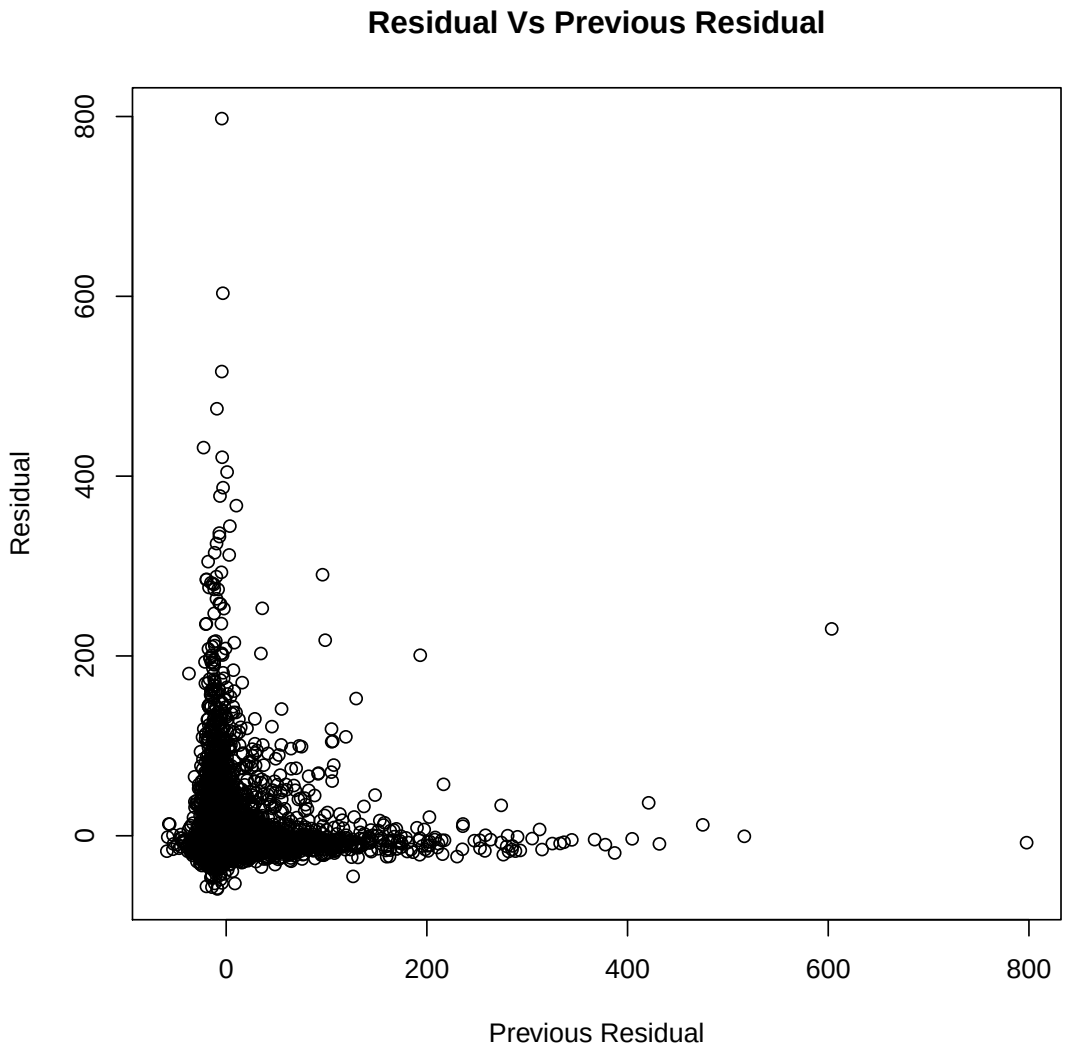
```
> object.size(flights.2008.data)
```

```
435856 bytes
```

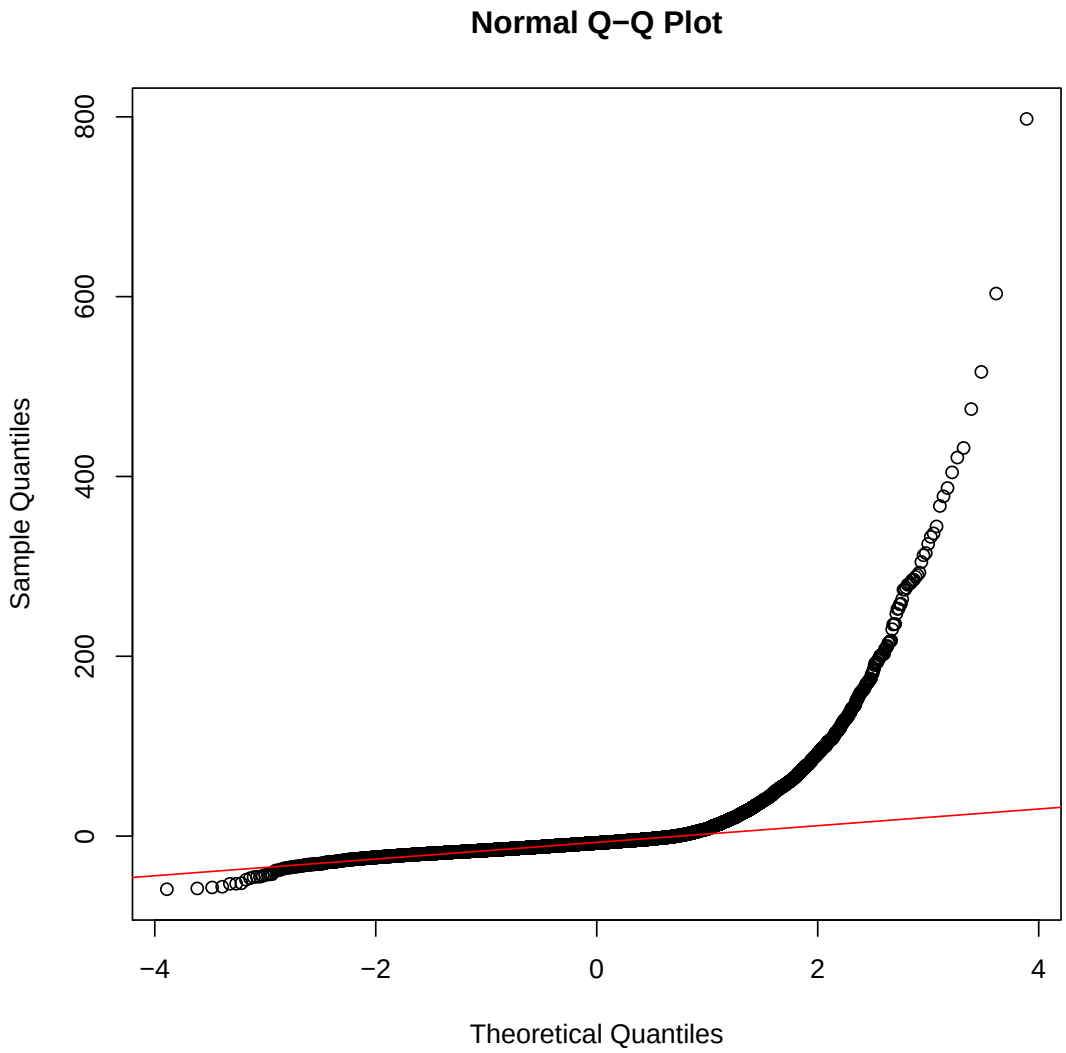
```
> flights.LM =
+   lm(DepDelay~DayOfWeek+DepTime+CRSDepTime
+       +ArrTime+CRSArrTime+UniqueCarrier,
+       data = flights.2008.data)

> res = flights.LM$residuals
> fit = flights.LM$fitted.values
> sample = sample(1:length(res), 1e4)
> res = res[sample]
> fit = fit[sample]

> plot(fit, res,
+       xlab = "Fitted Value", ylab = "Residual",
+       main = "Residual VS Fitted Value")
> abline(h = 0, col = "red")
```




```
> plot(res[-length(res)], res[-1],
+       main = "Residual Vs Previous Residual",
+       ylab = "Residual", xlab = "Previous Residual")
```




```
> qqnorm(res); qqline(res, col = "red")
```


Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles
0
200
400
600
800
−4
−2
0
2
4