Lists and Data Frames

Data Analysis with R and Python

Deepayan Sarkar

Lists

- Lists are vectors with arbitrary types of components

Lists

- Lists are vectors with arbitrary types of components
- Individual elements can be extracted using `x[[i]]`
- Vector indexing by `x[i]` also works in the usual way

Lists

- Lists are vectors with arbitrary types of components
- Individual elements can be extracted using `x[[i]]`
- Vector indexing by `x[i]` also works in the usual way
- A list may or may not have names
- Lists with names have a special type of extraction operator: `$`

Example: Results of an optimization procedure

- Problem:
    - A farmer has 100 meters of fencing
    - He wants to enclose a rectangular area to grow spinach
    - His goal is to maximize the enclosed area of the field
    - What dimensions (length and width) should he use?

Example: Results of an optimization procedure

- Problem:
  - A farmer has 100 meters of fencing
  - He wants to enclose a rectangular area to grow spinach
  - His goal is to maximize the enclosed area of the field
  - What dimensions (length and width) should he use?
- The answer is obviously that the area should be a square (with each side 25m)

Example: Results of an optimization procedure

- Problem:
  - A farmer has 100 meters of fencing
  - He wants to enclose a rectangular area to grow spinach
  - His goal is to maximize the enclosed area of the field
  - What dimensions (length and width) should he use?
- The answer is obviously that the area should be a square (with each side 25m)
- Easy to formulate and solve mathematically:

$$
\begin{aligned}
2 * (a + b) = 100 &\implies b = 50 - a \\
\text{To maximize } a * b = a * (50 - a) = 50a - a^2 &\iff \text{to minimize } a^2 - 50a \\
\text{Differentiate and solve } &: \ 2a - 50 = 0
\end{aligned}
$$

Example: Results of an optimization procedure

- Suppose we wanted to use numerical optimization instead

```
L <- function(a) a^2 - 50 * a
res <- optimize(L, interval = c(0, 100))
str(res)
```

```
List of 2
 $ minimum  : num 25
 $ objective: num -625
```

Printing lists

```
res
```

```
$minimum
[1] 25

$objective
[1] -625
```

Scalar indexing

- Standard indexing using `x[[index]]`

```
res[[1]]
```

```
[1] 25
```

```
res[["objective"]]
```

```
[1] -625
```

Scalar indexing

- Standard indexing using `x[[index]]`

```r
res[[1]]
```

```
[1] 25
```

```r
res[["objective"]]
```

```
[1] -625
```

- Extracting element by name using `x$name`

```r
res$minimum
```

```
[1] 25
```

Vector indexing

```r
res[1]
```

```
$minimum
[1] 25
```

```r
res["objective"]
```

```
$objective
[1] -625
```

Vector indexing

```r
res[1]
```

```
$minimum
[1] 25
```

```r
res["objective"]
```

```
$objective
[1] -625
```

```r
res[c(2, 1, 2)]
```

```
$objective
[1] -625

$minimum
[1] 25

$objective
[1] -625
```

Lists as containers

- This is an important use of lists
    - Represent result of an analysis
    - Collects various relevant quantities
    - Result contains all quantities in single object

Other algorithms for "root finding"

- Exercise: Can we write our own optimization method?

Other algorithms for "root finding"

- Exercise: Can we write our own optimization method?
- Simple algorithms (root finding)
    - Grid search
    - Bisection method
    - Regula falsi

Example: Confidence interval for probability p

- Given data: $X_1, X_2, \ldots, X_n$ i.i.d. sample from Bernoulli(p)
- We want to estimate unknown success probability p

Example: Confidence interval for probability p

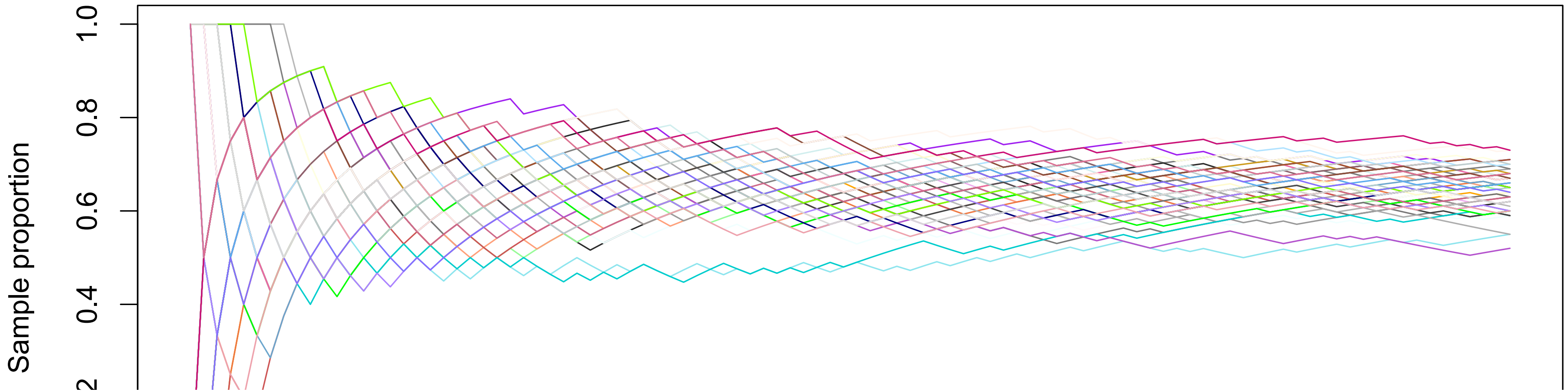
- Given data: $X_1, X_2, \ldots, X_n$ i.i.d. sample from Bernoulli(p)
- We want to estimate unknown success probability p
- Natural estimator is the sample proportion $\hat{p} = \bar{X}$

Example: Confidence interval for probability p

- Given data: $X_1, X_2, \ldots, X_n$ i.i.d. sample from Bernoulli(p)
- We want to estimate unknown success probability p
- Natural estimator is the sample proportion $\hat{p} = \bar{X}$
- But $\hat{p}$ will usually not be exactly equal to p
- What values of p are *plausible*? (Consistent with observed data)

Recall: distribution of sample proportion

```r
p <- 0.65
plot(1:100, type = "n", ylim = c(0, 1), ylab = "Sample proportion")
for (i in 1:50) {
    z <- rbinom(100, size = 1, prob = p)
    lines(1:100, cumsum(z) / 1:100, col = sample(colors(), 1))
}
```



Example: Does a random quadratic equation have real roots?

- Suppose $A, B, C \sim N(0,1)$
- Are the roots of $Ax^2 + Bx + C = 0$ real?

Example: Does a random quadratic equation have real roots?

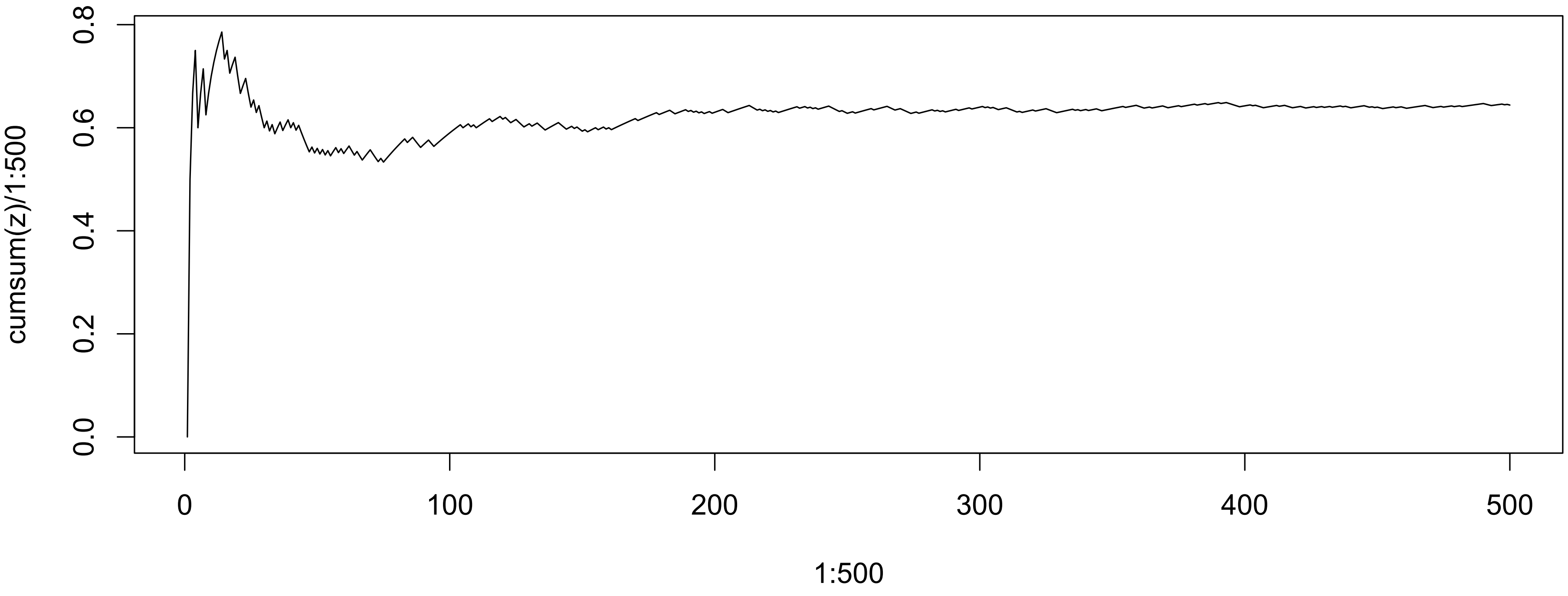
- Suppose $A, B, C \sim N(0, 1)$
- Are the roots of $Ax^2 + Bx + C = 0$ real?

```
z <- logical(500)
for (i in 1:500) {
    A <- rnorm(1); B <- rnorm(1); C <- rnorm(1)
    D <- B^2 - 4 * A * C
    z[i] <- D >= 0
}
str(z)
```

```
logi [1:500] FALSE TRUE TRUE TRUE FALSE TRUE ...
```

Example: Does a random quadratic equation have real roots?

```r
plot(1:500, cumsum(z) / 1:500, type = "l")
```



Example: Does a random quadratic equation have real roots?

- One method implemented in R function `prop.test()`

```
tt500 <- prop.test(sum(z), 500)
str(tt500, give.attr = FALSE)
```

```
List of 9
 $ statistic  : Named num 40.9
 $ parameter  : Named int 1
 $ p.value    : num 1.6e-10
 $ estimate   : Named num 0.644
 $ null.value : Named num 0.5
 $ conf.int   : num [1:2] 0.6 0.686
 $ alternative: chr "two.sided"
 $ method     : chr "1-sample proportions test with continuity correction"
 $ data.name  : chr "sum(z) out of 500, null probability 0.5"
```

Example: Does a random quadratic equation have real roots?

- One method implemented in R function `prop.test()`

```
tt500 <- prop.test(sum(z), 500)
str(tt500, give.attr = FALSE)
```

```
List of 9
 $ statistic  : Named num 40.9
 $ parameter  : Named int 1
 $ p.value    : num 1.6e-10
 $ estimate   : Named num 0.644
 $ null.value : Named num 0.5
 $ conf.int   : num [1:2] 0.6 0.686
 $ alternative: chr "two.sided"
 $ method     : chr "1-sample proportions test with continuity correction"
 $ data.name  : chr "sum(z) out of 500, null probability 0.5"
```

```
tt500$conf.int
```

```
[1] 0.6000604 0.6856917
attr(,"conf.level")
[1] 0.95
```

Example: Does a random quadratic equation have real roots?

- What if we only had 100 replications?

```
tt100 <- prop.test(sum(z[1:100]), 100)
str(tt100, give.attr = FALSE)
```

```
List of 9
 $ statistic  : Named num 2.89
 $ parameter  : Named int 1
 $ p.value    : num 0.0891
 $ estimate   : Named num 0.59
 $ null.value : Named num 0.5
 $ conf.int   : num [1:2] 0.487 0.686
 $ alternative: chr "two.sided"
 $ method     : chr "1-sample proportions test with continuity correction"
 $ data.name  : chr "sum(z[1:100]) out of 100, null probability 0.5"
```

```
tt100$conf.int
```

```
[1] 0.4870348 0.6859677
attr(,"conf.level")
[1] 0.95
```

Lists as containers

```
tt500
```

```
    1-sample proportions test with continuity correction

data:  sum(z) out of 500, null probability 0.5
X-squared = 40.898, df = 1, p-value = 1.604e-10
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.6000604 0.6856917
sample estimates:
    p
0.644
```

Why doesn't output look like a list?

- This is an important feature of R
- Uses concepts such as attributes and class
- We will discuss these later

Data Frames

- R analog of a spreadsheet

Example

Manufacturer	Model	Type	Min.Price	Price	Max.Price	MPG.city	MPG.highway
Acura	Integra	Small	12.9	15.9	18.8	25	31
Acura	Legend	Midsize	29.2	33.9	38.7	18	25
Audi	90	Compact	25.9	29.1	32.3	20	26
Audi	100	Midsize	30.8	37.7	44.6	19	26
BMW	535i	Midsize	23.7	30.0	36.2	22	30

Data Frames

- R analog of a spreadsheet
- Rectangular (matrix-like) structure

Data Frames

- R analog of a spreadsheet
- Rectangular (matrix-like) structure
- Each column is (usually) an atomic vector
- Different columns can be of different types

Data Frames

- R analog of a spreadsheet
- Rectangular (matrix-like) structure
- Each column is (usually) an atomic vector
- Different columns can be of different types
- Every column must have the same length
- Every column must have a name

Data Frames

- Most built-in data sets in R are data frames

```r
data(Cars93, package = "MASS")
Cars93
```

```
   Manufacturer          Model    Type Min.Price Price Max.Price MPG.city MPG.highway
1         Acura        Integra   Small      12.9  15.9      18.8       25          31
2         Acura         Legend Midsize      29.2  33.9      38.7       18          25
3          Audi             90 Compact      25.9  29.1      32.3       20          26
4          Audi            100 Midsize      30.8  37.7      44.6       19          26
5           BMW           535i Midsize      23.7  30.0      36.2       22          30
6         Buick        Century Midsize      14.2  15.7      17.3       22          31
7         Buick        LeSabre   Large      19.9  20.8      21.7       19          28
8         Buick     Roadmaster   Large      22.6  23.7      24.9       16          25
9         Buick        Riviera Midsize      26.3  26.3      26.3       19          27
10     Cadillac        DeVille   Large      33.0  34.7      36.3       16          25
11     Cadillac        Seville Midsize      37.5  40.1      42.7       16          25
12    Chevrolet       Cavalier Compact       8.5  13.4      18.3       25          36
13    Chevrolet        Corsica Compact      11.4  11.4      11.4       25          34
14    Chevrolet         Camaro  Sporty      13.4  15.1      16.8       19          28
```

Data Frames

- Data frames are internally stored as lists (with constraints)

```
str(Cars93)
```

```
'data.frame':	93 obs. of  27 variables:
 $ Manufacturer      : Factor w/ 32 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ Model             : Factor w/ 93 levels "100","190E","240",..: 49 56 9 1 6 24 54 74 73 35 ...
 $ Type              : Factor w/ 6 levels "Compact","Large",..: 4 3 1 3 3 3 2 2 3 2 ...
 $ Min.Price         : num  12.9 29.2 25.9 30.8 23.7 14.2 19.9 22.6 26.3 33 ...
 $ Price             : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ Max.Price         : num  18.8 38.7 32.3 44.6 36.2 17.3 21.7 24.9 26.3 36.3 ...
 $ MPG.city          : int  25 18 20 19 22 22 19 16 19 16 ...
 $ MPG.highway       : int  31 25 26 26 30 31 28 25 27 25 ...
 $ AirBags           : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
 $ DriveTrain        : Factor w/ 3 levels "4WD","Front",..: 2 2 2 2 3 2 2 3 2 2 ...
 $ Cylinders         : Factor w/ 6 levels "3","4","5","6",..: 2 4 4 4 2 2 4 4 4 5 ...
 $ EngineSize        : num  1.8 3.2 2.8 2.8 3.5 2.2 3.8 5.7 3.8 4.9 ...
 $ Horsepower        : int  140 200 172 172 208 110 170 180 170 200 ...
 $ RPM               : int  6300 5500 5500 5500 5700 5200 4800 4000 4800 4100 ...
 $ Rev.per.mile      : int  2890 2335 2280 2535 2545 2565 1570 1320 1690 1510 ...
```

Data Frames

- List-like behaviour: Columns can be extracted like a list

```r
Cars93$MPG.city
```

```
 [1] 25 18 20 19 22 22 19 16 19 16 16 25 25 19 21 18 15 17 17 20 23 20 29 23 22 17 21 18
[29] 29 20 31 23 22 22 24 15 21 18 46 30 24 42 24 29 22 26 20 17 18 18 17 18 29 28 26 18
[57] 17 20 19 23 19 29 18 29 24 17 21 24 23 18 19 23 31 23 19 19 19 20 28 33 25 23 39 32
[85] 25 22 18 25 17 21 18 21 20
```

- Vector indexing extracts multiple columns

```r
head(Cars93[c(1, 4, 7)])
```

```
  Manufacturer Min.Price MPG.city
1        Acura      12.9       25
2        Acura      29.2       18
3         Audi      25.9       20
4         Audi      30.8       19
5          BMW      23.7       22
6        Buick      14.2       22
```

Data Frames

```
carsub <- Cars93[c("Make", "MPG.city", "Weight",
                   "Length", "EngineSize", "Man.trans.avail")]
str(carsub)
```

```
'data.frame':	93 obs. of  6 variables:
 $ Make           : Factor w/ 93 levels "Acura Integra",..: 1 2 4 3 5 6 7 9 8 10 ...
 $ MPG.city       : int  25 18 20 19 22 22 19 16 19 16 ...
 $ Weight         : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
 $ Length         : int  177 195 180 193 186 189 200 216 198 206 ...
 $ EngineSize     : num  1.8 3.2 2.8 2.8 3.5 2.2 3.8 5.7 3.8 4.9 ...
 $ Man.trans.avail: Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 1 1 1 1 ...
```

Data Frames

- Two-dimensional indexing

```
carsub[1:6, ]
```

```
            Make MPG.city Weight Length EngineSize Man.trans.avail
1 Acura Integra       25   2705    177        1.8             Yes
2  Acura Legend       18   3560    195        3.2             Yes
3       Audi 90       20   3375    180        2.8             Yes
4      Audi 100       19   3405    193        2.8             Yes
5      BMW 535i       22   3640    186        3.5             Yes
6 Buick Century       22   2880    189        2.2              No
```

```
carsub[1:6, c(1, 4, 6)]
```

```
            Make Length Man.trans.avail
1 Acura Integra    177             Yes
2  Acura Legend    195             Yes
3       Audi 90    180             Yes
4      Audi 100    193             Yes
5      BMW 535i    186             Yes
6 Buick Century    189              No
```

Data Frames

- Two-dimensional indexing

```
nrow(carsub)
```

```
[1] 93
```

```
carsub[sample(nrow(carsub), 6), ]
```

```
                 Make MPG.city Weight Length EngineSize Man.trans.avail
15    Chevrolet Lumina       21   3195    198        2.2              No
37         Ford Taurus       21   3325    192        3.0              No
41       Honda Prelude       24   2865    175        2.3             Yes
78            Saab 900       20   2775    184        2.1             Yes
91 Volkswagen Corrado       18   2810    159        2.8             Yes
65       Nissan Altima       24   3050    181        2.4             Yes
```

Data Frames

- Two-dimensional indexing

```
carsub[sample(nrow(carsub), 6), c("MPG.city", "Weight", "Length")]
```

```
   MPG.city Weight Length
22       20   3570    203
58       20   2920    175
78       20   2775    184
87       18   3785    187
3        20   3375    180
85       25   2950    174
```

Data import

- Statistical data are usually structured like a spreadsheet (Excel / CSV)

Data import

- Statistical data are usually structured like a spreadsheet (Excel / CSV)
- Typical approach: read data from spreadsheet file into data frame

Data import

- Statistical data are usually structured like a spreadsheet (Excel / CSV)
- Typical approach: read data from spreadsheet file into data frame
- Easiest route:
  - R itself cannot read Excel files directly
  - Save as CSV file from Excel / LibreOffice / Google Sheets
  - Read with `read.csv()` or `read.table()` (more flexible)

Data import

- Statistical data are usually structured like a spreadsheet (Excel / CSV)
- Typical approach: read data from spreadsheet file into data frame
- Easiest route:
    - R itself cannot read Excel files directly
    - Save as CSV file from Excel / LibreOffice / Google Sheets
    - Read with `read.csv()` or `read.table()` (more flexible)
- Alternative: Use "Import Dataset" tool in RStudio

Data import

- Statistical data are usually structured like a spreadsheet (Excel / CSV)
- Typical approach: read data from spreadsheet file into data frame
- Easiest route:
  - R itself cannot read Excel files directly
  - Save as CSV file from Excel / LibreOffice / Google Sheets
  - Read with `read.csv()` or `read.table()` (more flexible)
- Alternative: Use "Import Dataset" tool in RStudio
- Data frames can be exported as a spreadsheet file using `write.csv()` or `write.table()`

Exporting data as CSV file

```
str(carsub)
```

```
'data.frame':	93 obs. of  6 variables:
 $ Make           : Factor w/ 93 levels "Acura Integra",..: 1 2 4 3 5 6 7 9 8 10 ...
 $ MPG.city       : int  25 18 20 19 22 22 19 16 19 16 ...
 $ Weight         : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
 $ Length         : int  177 195 180 193 186 189 200 216 198 206 ...
 $ EngineSize     : num  1.8 3.2 2.8 2.8 3.5 2.2 3.8 5.7 3.8 4.9 ...
 $ Man.trans.avail: Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 1 1 1 1 ...
```

```
write.csv(carsub, file = "cars.csv")
```

Other data import / export utilities

- These are the most basic data input / output functions
- There are many other other specialized functions
- Low-level utilties: `scan()`, `readLines()`, `readChar()`, `readBin()`
- Various packages provide import / export to formats used by other software
- R has its own "serialization" format using `save()` and `load()`

Example: air quality data

```r
aqi <- read.csv("https://deepayan.github.io/BSDS/2024-01-DE/data/rkpuram-aqi.csv")
aqi
```

```
         date pm25 pm10  o3 no2 so2  co
1   2024/11/1  300  260  40  19   8  17
2   2024/11/2  306  249  57  16   5  19
3   2024/11/3  308  298  53  21   3  17
4   2024/11/4  300  246  56  12   9  24
5   2024/11/5  282  227  52  15   8  22
6   2024/11/6  267  251  47  16   5  19
7   2024/11/7  307  205  40  13   9  21
8   2024/11/8  275  198  46   9  10  22
9   2024/11/9  269  195  53   8  11  19
10 2024/11/10  260  264  46   9  18  32
11 2024/11/11  254  176  42   6  11  20
12 2024/11/12  228  492  50   6   8  27
13 2024/11/13  463  324   8   4  12  35
14 2024/11/14  385   NA  NA  NA  NA  NA
```