The Basics of the R Programming Language

Charlene Garridos

2023-04-27

Contents

1 R Basic

2 Data in R

This <- operator is used to assign names

```
my_sum <- 3+3 # store the result
my_sum +3
```

```
## [1] 9
```

```
my_sum <- 3+3
my_list <- list
my_sum +3
```

```
## [1] 9
```

```
my_sum # shows the value of my_sum
```

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

```
(new_sum <- my_sum + 3) # assign my_sum + 3 to new_sum and show its value
```

```
## [1] 9
```

```
(another_sum <- 5+6)
```

```
## [1] 11
```

2.1 Data Types

The **typeof()** function can be used to see the type of a single scalar value.

Numeric data can be in integer form or double (decimal) form.

```
my_integer <- 1L # integers can be signified by adding an 'L' to the end
typeof(my_integer)
```

```
## [1] "integer"
```

```
my_integer <- 9
typeof(my_integer)
```

```
## [1] "double"
```

Character data is text data surrounded by single or double quotes.

```r
my_character <- "This is Text"
typeof(my_character)
```

```
## [1] "character"
```

Logical data takes the form TRUE or FALSE.

```r
(my_logical <- FALSE)
```

```
## [1] FALSE
```

```r
typeof(my_logical)
```

```
## [1] "logical"
```

```r
my_double <- 5.6
typeof(my_double)
```

```
## [1] "double"
```

2.2 Homogeneous Data Structure

Vectors are one-dimensional structures that carry the same type of data and are denoted by the function **c()**. The type of the vector may also be checked with the **typeof()** function, however the **str()** method can display both the contents and the type of the vector.

```r
double_vec <- c(3.1, 31, 311, 3111, 3.111)
str(double_vec)
```

```
##  num [1:5] 3.1 31 311 3111 3.11
```

Categorical data, which has a finite number of possible values, can be represented or stored as a factor vector to facilitate grouping and processing.

```r
categories <- factor(c("C", "H", "A", "R", "L"))
str(categories)
```

```
##  Factor w/ 5 levels "A","C","H","L",..: 2 3 1 5 4
```

```r
categories_char <- c("C", "H", "A", "R", "L")
str(categories_char)
```

```
##  chr [1:5] "C" "H" "A" "R" "L"
```

```r
ranking <- c("Medium", "High", "Low") # character vector, factors can be given order.
str(ranking)
```

```
##  chr [1:3] "Medium" "High" "Low"
```

```r
# turn it into an ordered factor
ranking_factors <- ordered(ranking, levels = c("Low", "Medium", "High"))
str(ranking_factors)
```

```
##  Ord.factor w/ 3 levels "Low"<"Medium"<..: 2 3 1
```

The **length()** function returns the number of entries or elements in a vector.

```r
length(categories)
```

```
## [1] 5
```

```r
length(ranking_factors)
```

```
## [1] 3
```

```r
(my_sequence <- 1:10) # Simple numeric sequence vectors can be created using shorthand notation.
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
(my_sequence <- seq(from = 1, to = 10))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
my_seq_two <- seq(from = 1, to = 10, by = 5)
my_seq_two
```

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

```r
my_seq_three <- seq(from = 1, to = 10, by = 2)
my_seq_three
```

```
## [1] 1 3 5 7 9
```

```r
vec <- 1:5 # numeric sequence vector
str(vec)
```

```
##  int [1:5] 1 2 3 4 5
```

```r
new_vec <- c(vec, "hello") # creating a new vector containing vec and the character "hello"
str(new_vec) # numeric values have been coerced into their character equivalents
```

```
##  chr [1:6] "1" "2" "3" "4" "5" "hello"
```

```r
# attempt a mixed logical and numeric, sometimes logical or factor types will be coerced to numeric.
mix <- c(TRUE, 6)
str(mix) # logical has been converted to binary numeric (TRUE = 1)
```

```
##  num [1:2] 1 6
```

```r
new_categories <- c(categories, 1) # try to add a numeric to our previous categories factor vector
str(new_categories) # categories have been coerced to background integer representations
```

```
##  num [1:6] 2 3 1 5 4 1
```

```r
str(categories)
```

```
##  Factor w/ 5 levels "A","C","H","L",..: 2 3 1 5 4
```

Matrices are two-dimensional data structures of the same type that are created by determining the number of rows and columns in a vector. Data is read into the matrix down the columns, from left to right. It are rarely used for non-numeric data types.

```r
(m <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)) # create a 2x2 matrix with the first four integers
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

Arrays are n-dimensional data structures with the same data type and are not used extensively by most R users.

```r
(m <- matrix(vec, nrow = 5, ncol = 2))
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    2
## [3,]    3    3
## [4,]    4    4
## [5,]    5    5
```

2.3 Heterogeneous Data Structure

Lists are one-dimensional data structures that can take data of any type.List elements can be any data type and any dimension. Each element can be given a name.

```r
my_list <- list(6, TRUE, "hello")
str(my_list)
```

```
## List of 3
##  $ : num 6
##  $ : logi TRUE
##  $ : chr "hello"
```

```
new_list <- list(scalar = 6, vector = c("Heloo", "Goodbye"),
                 matrix = matrix(1:4, nrow = 2, ncol = 2))
str(new_list)
```

```
## List of 3
##  $ scalar: num 6
##  $ vector: chr [1:2] "Heloo" "Goodbye"
##  $ matrix: int [1:2, 1:2] 1 2 3 4
```

This symbol **$** is used to accesses the named list elements

```
new_list$matrix
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
new_list$vec
```

```
## [1] "Heloo"   "Goodbye"
```

Dataframes are the most used data structure in R, due to its similarity to data tables found in databases and spreadsheets. They are effectively a named list of vectors of the same length, with each vector as a column. As such, a dataframe is very similar in nature to a typical database table or spreadsheet

```
names <- c("Charl", "Garridos") # two vectors of different types but same length
ages <- c(20, 20)
(df <- data.frame(names, ages)) # create a dataframe
```

```
##      names ages
## 1    Charl   20
## 2 Garridos   20
```

```
str(df) # get types of columns
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ names: chr  "Charl" "Garridos"
##  $ ages : num  20 20
```

```
dim(df) # get dimensions of df
```

```
## [1] 2 2
```

3 Working with Data frames

3.1 Loading and Tidying Data in Dataframes

```r
# url of data set
url <- "https://raw.githubusercontent.com/msuiitdmsgabriel/datasets-regression/main/salespeople.csv"
```

The **read.csv()** function can accept a URL address of the file if it is online.

```r
salespeople <- read.csv(url) #load the data set and store it as a dataframe called salespeople
local_salespeople <- read.csv("salespeople.csv")
```

The function head, which displays thee first rows of a dataframe, has only one required argument x: the name of the dataframe

```r
high_sales <- subset(salespeople, subset = sales >= 700)
head(high_sales)
```

```
##    promoted sales customer_rate performance
## 6         1   918          4.54           2
## 12        1   716          3.16           3
## 20        1   937          5.00           2
## 21        1   702          3.53           4
## 25        1   819          4.45           2
## 26        1   736          3.94           4
```

```r
# to select specific columns use the select argument
subset(salespeople, select = c("sales", "performance"))
```

the **head()** function to display just the first few rows.

```r
dim(salespeople)
```

```
## [1] 351   4
```

```r
head(salespeople) # hundreds of rows, so view first few
```

```
##   promoted sales customer_rate performance
## 1        0   594          3.94           2
## 2        0   446          4.06           3
## 3        1   674          3.83           4
## 4        0   525          3.62           2
## 5        1   657          4.40           3
## 6        1   918          4.54           2
```

View a specific column by using symbol **$**, and use **square brackets** to view a specific entry.

```r
salespeople$sales[6]
```

```
## [1] 918
```

```
salespeople$sales
```

```
##   [1] 594 446 674 525 657 918 318 364 342 387 527 716 557 450 344 372 258 338
##  [19] 410 937 702 469 535 342 819 736 330 274 341 717 478 487 239 825 400 728
##  [37] 773 425 943 510 389 270 945 497 329 389 475 383 432 619 578 411 445 440
##  [55] 359 419 840 393 754 441 803 444 753 688 431 511 464 473 532 280 342 320
##  [73] 531 373 547 611 825 431 401 517 803 586 444 693 659 416 423 756 245 419
##  [91] 757 617 909 516 317 425 528 416 645 390 393 394 387 450 487 607 369 489
## [109] 324 417 694 651 395 442 422 404 381 501 944 753 591 735 538 451 477 436
## [127] 738 902 464 944 285 453 382 414 335 935 203 348 800 436 360 674 425 901
## [145] 453 350 362 486 471 459 506 262 825 291 464 802 818 736 364 308 862 349
## [163] 375 423 938 456 517 373 898 777 470 545 699 697 300 677 497 669 596 492
## [181] 346 590 592 780 432 418 662 678 716 330 414 416 403 362 284 363 655 597
## [199] 794 818 409 681 606 489 475 590 396 420 857 371 421 828 594 533 462 392
## [217] 475 752 659 650 496 211 898 388 383 455 319 756 377 940 757 469 394 484
## [235] 491 547 519 739 479 943 742 357 432 584 595 401 460 753 466 362 361 338
## [253] 882 293 922 793 787 400 516 295 307 151 441 406 270 680 662 347 453 309
## [271] 592 540 886 420 718 284 323 513 841 362 842 321 516 428 383 521 358 489
## [289] 252 720 610 871 594 522 379 454 450 317 835 297 516 355 858 305 410 707
## [307] 798 265 576 448 590 456 930 412 286 440 546 385 544 505 732 506 394 674
## [325] 458 251 429 348 789 795 509 754 580 289 390 787 241 522 412 359 489 940
## [343] 592 796 653 459 586 401 500 373  NA
```

```
View(salespeople)
```

Alternatively, use a **[row, column]** index to get a specific entry in the dataframe.

```
salespeople[34, 4]
```

```
## [1] 3
```

```
salespeople[34,]
```

```
##    promoted sales customer_rate performance
## 34        1   825          3.32           3
```

```
salespeople[,4]
```

```
##   [1]  2  3  4  2  3  2  3  1  3  3  3  3  2  3  2  3  1  4  2  2  4  2  2  1  2
##  [26]  4  2  1  2  2  2  1  4  3  2  3  3  1  4  3  4  2  4  3  3  4  3  2  3  3
##  [51]  4  4  3  2  1  3  4  1  3  2  3  2  4  2  4  2  3  2  1  2  2  3  4  2  1
##  [76]  4  2  3  2  3  3  1  4  3  1  3  3  4  2  2  3  1  3  1  1  3  2  1  2  4
## [101]  1  2  3  3  3  4  1  2  3  1  2  4  2  1  3  3  4  4  2  3  4  4  3  2  3
## [126]  2  3  4  1  4  3  2  2  2  3  3  2  3  2  3  1  3  3  3  2  3  2  1  2  3
## [151]  3  2  3  2  3  2  3  3  3  1  4  4  2  3  3  1  2  1  4  3  3  4  4  3  2
## [176]  3  1  4  2  3  2  2  3  4  2  2  4  3  3  1  1  2  1  3  3  1  3  3  3  1
## [201]  1  1  3  2  3  3  2  2  2  2  3  4  1  2  2  3  3  2  2  2  3  1  3  1  2
## [226]  3  4  3  3  3  3  3  1  2  4  2  4  3  2  4  2  1  3  2  2  3  2  2  2  2
## [251]  2  3  3  2  1  2  3  2  4  2  1  2  2  1  2  4  2  3  1  1  2  4  3  4  4
## [276]  2  3  3  4  1  3  1  3  4  3  1  2  3  2  3  4  2  3  2  3  2  2  2  1  4
## [301]  3  2  3  3  3  1  3  2  3  1  3  3  4  2  1  1  1  3  1  1  2  3  4  2  4
## [326]  2  1  3  3  1  3  4  1  3  3  1  2  2  2  2  3  4  4  3  3  3  2  3  2  1
## [351] NA
```

Use **str()** function to view the data type.

```
str(salespeople)
```

```
## 'data.frame':    351 obs. of  4 variables:
##  $ promoted     : int  0 0 1 0 1 1 0 0 0 0 ...
##  $ sales        : int  594 446 674 525 657 918 318 364 342 387 ...
##  $ customer_rate: num  3.94 4.06 3.83 3.62 4.4 4.54 3.09 4.89 3.74 3 ...
##  $ performance  : int  2 3 4 2 3 2 3 1 3 3 ...
```

Use **summary()**, to see a statistical summary of each column, which depicts various statistics depending on the type of the column

```
summary(salespeople)
```

```
##     promoted          sales       customer_rate    performance
##  Min.   :0.0000   Min.   :151.0   Min.   :1.000   Min.   :1.0
##  1st Qu.:0.0000   1st Qu.:389.2   1st Qu.:3.000   1st Qu.:2.0
##  Median :0.0000   Median :475.0   Median :3.620   Median :3.0
##  Mean   :0.3219   Mean   :527.0   Mean   :3.608   Mean   :2.5
##  3rd Qu.:1.0000   3rd Qu.:667.2   3rd Qu.:4.290   3rd Qu.:3.0
##  Max.   :1.0000   Max.   :945.0   Max.   :5.000   Max.   :4.0
##                   NA's   :1       NA's   :1       NA's   :1
```

```
sum(is.na(salespeople)) #confirm no NAs
```

```
## [1] 3
```

```
is.na(salespeople)
```

complete.cases() function identifies the rows that have no NAs, and then select those rows from the dataframe based on that condition

```
complete.cases(salespeople)
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [49]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [61]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [73]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [85]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [97]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [109]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [121]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [133]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [145]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [157]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [169]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
## [181]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [193]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [205]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [217]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [229]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [241]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [253]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [265]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [277]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [289]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [301]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [313]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [325]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [337]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [349]  TRUE  TRUE FALSE
```

```
salespeople[complete.cases(salespeople),]
```

```
salespeople <- salespeople[complete.cases(salespeople),]
```

```
is.na(salespeople)
```

```
sum(is.na(salespeople))
```

```
## [1] 0
```

Use the **unique()** function to see the unique values of a vector or column.

```
unique(salespeople$performance)
```

```
## [1] 2 3 4 1
```

to change the type of a column in a dataframe, use the **as.numeric(), as.character(), as.logical() or as.factor()** functions.

```
salespeople$performance
```

```
##   [1] 2 3 4 2 3 2 3 1 3 3 3 3 2 3 2 3 1 4 2 2 4 2 2 1 2 4 2 1 2 2 2 1 4 3 2 3 3
##  [38] 1 4 3 4 2 4 3 3 4 3 2 3 3 4 4 3 2 1 3 4 1 3 2 3 2 4 2 4 2 3 2 1 2 2 3 4 2
##  [75] 1 4 2 3 2 3 3 1 4 3 1 3 3 4 2 2 3 1 3 1 1 3 2 1 2 4 1 2 3 3 3 4 1 2 3 1 2
## [112] 4 2 1 3 3 4 4 2 3 4 4 3 2 3 2 3 4 1 4 3 2 2 2 3 3 2 3 2 3 1 3 3 3 2 3 2 1
## [149] 2 3 3 2 3 2 3 2 3 3 3 1 4 4 2 3 3 1 2 1 4 3 3 4 4 3 2 3 1 4 2 3 2 2 3 4 2
## [186] 2 4 3 3 1 1 2 1 3 3 1 3 3 3 1 1 1 3 2 3 3 2 2 2 2 3 4 1 2 2 3 3 2 2 2 3 1
## [223] 3 1 2 3 4 3 3 3 3 3 1 2 4 2 4 3 2 4 2 1 3 2 2 3 2 2 2 2 2 3 3 2 1 2 3 2 4
## [260] 2 1 2 2 1 2 4 2 3 1 1 2 4 3 4 4 2 3 3 4 1 3 1 3 4 3 1 2 3 2 3 4 2 3 2 3 2
## [297] 2 2 1 4 3 2 3 3 3 1 3 2 3 1 3 3 4 2 1 1 1 3 1 1 2 3 4 2 4 2 1 3 3 1 3 4 1
## [334] 3 3 1 2 2 2 2 3 4 4 3 3 3 2 3 2 1
```

```
as.factor(salespeople$performance)
```

```
##   [1] 2 3 4 2 3 2 3 1 3 3 3 3 2 3 2 3 1 4 2 2 4 2 2 1 2 4 2 1 2 2 2 1 4 3 2 3 3
##  [38] 1 4 3 4 2 4 3 3 4 3 2 3 3 4 4 3 2 1 3 4 1 3 2 3 2 4 2 4 2 3 2 1 2 2 3 4 2
##  [75] 1 4 2 3 2 3 3 1 4 3 1 3 3 4 2 2 3 1 3 1 1 3 2 1 2 4 1 2 3 3 3 4 1 2 3 1 2
## [112] 4 2 1 3 3 4 4 2 3 4 4 3 2 3 2 3 4 1 4 3 2 2 2 3 3 2 3 2 3 1 3 3 3 2 3 2 1
## [149] 2 3 3 2 3 2 3 2 3 3 3 1 4 4 2 3 3 1 2 1 4 3 3 4 4 3 2 3 1 4 2 3 2 2 3 4 2
## [186] 2 4 3 3 1 1 2 1 3 3 1 3 3 3 1 1 1 3 2 3 3 2 2 2 2 3 4 1 2 2 3 3 2 2 2 3 1
## [223] 3 1 2 3 4 3 3 3 3 3 1 2 4 2 4 3 2 4 2 1 3 2 2 3 2 2 2 2 2 3 3 2 1 2 3 2 4
## [260] 2 1 2 2 1 2 4 2 3 1 1 2 4 3 4 4 2 3 3 4 1 3 1 3 4 3 1 2 3 2 3 4 2 3 2 3 2
## [297] 2 2 1 4 3 2 3 3 3 1 3 2 3 1 3 3 4 2 1 1 1 3 1 1 2 3 4 2 4 2 1 3 3 1 3 4 1
## [334] 3 3 1 2 2 2 2 3 4 4 3 3 3 2 3 2 1
## Levels: 1 2 3 4
```

```
salespeople$performance <- as.factor(salespeople$performance)
str(salespeople)
```

```
## 'data.frame':    350 obs. of  4 variables:
##  $ promoted     : int  0 0 1 0 1 1 0 0 0 0 ...
##  $ sales        : int  594 446 674 525 657 918 318 364 342 387 ...
##  $ customer_rate: num  3.94 4.06 3.83 3.62 4.4 4.54 3.09 4.89 3.74 3 ...
##  $ performance  : Factor w/ 4 levels "1","2","3","4": 2 3 4 2 3 2 3 1 3 3 ...
```

3.2 Manipulating Dataframes

The use of == is to test for precise equality. Similarly, selevting columns are based on inequalities (> for 'greater than', < for 'less than', >= for 'greater than or equal to', <= for 'less than or equal to', or != for 'not equal to').

```
(sales_720 <- subset(salespeople, subset = sales == 720))
```

```
##     promoted sales customer_rate performance
## 290        1   720          3.76           3
```

```
unique(salespeople$sales)
```

```
##   [1] 594 446 674 525 657 918 318 364 342 387 527 716 557 450 344 372 258 338
##  [19] 410 937 702 469 535 819 736 330 274 341 717 478 487 239 825 400 728 773
##  [37] 425 943 510 389 270 945 497 329 475 383 432 619 578 411 445 440 359 419
##  [55] 840 393 754 441 803 444 753 688 431 511 464 473 532 280 320 531 373 547
##  [73] 611 401 517 586 693 659 416 423 756 245 757 617 909 516 317 528 645 390
##  [91] 394 607 369 489 324 417 694 651 395 442 422 404 381 501 944 591 735 538
## [109] 451 477 436 738 902 285 453 382 414 335 935 203 348 800 360 901 350 362
## [127] 486 471 459 506 262 291 802 818 308 862 349 375 938 456 898 777 470 545
## [145] 699 697 300 677 669 596 492 346 590 592 780 418 662 678 403 284 363 655
## [163] 597 794 409 681 606 396 420 857 371 421 828 533 462 392 752 650 496 211
## [181] 388 455 319 377 940 484 491 519 739 479 742 357 584 595 460 466 361 882
## [199] 293 922 793 787 295 307 151 406 680 347 309 540 886 718 323 513 841 842
## [217] 321 428 521 358 252 720 610 871 522 379 454 835 297 355 858 305 707 798
## [235] 265 576 448 930 412 286 546 385 544 505 732 458 251 429 789 795 509 580
## [253] 289 241 796 653 500
```

```
(subset(salespeople, subset = sales >= 700))
```

```
##     promoted sales customer_rate performance
## 6          1   918          4.54           2
## 12         1   716          3.16           3
## 20         1   937          5.00           2
## 21         1   702          3.53           4
## 25         1   819          4.45           2
## 26         1   736          3.94           4
## 30         1   717          2.98           2
## 34         1   825          3.32           3
## 36         1   728          2.66           3
## 37         1   773          4.89           3
## 39         1   943          4.40           4
## 43         1   945          4.31           4
## 57         1   840          5.00           4
## 59         1   754          3.74           3
## 61         1   803          4.89           3
## 63         1   753          5.00           4
## 77         1   825          4.66           2
## 81         1   803          4.15           3
## 88         1   756          3.55           4
## 91         1   757          3.11           3
## 93         1   909          3.21           3
## 119        1   944          5.00           2
## 120        1   753          4.43           3
## 122        1   735          4.03           4
## 127        1   738          3.05           3
## 128        1   902          5.00           4
## 130        1   944          3.92           4
## 136        1   935          5.00           3
## 139        1   800          4.24           2
## 144        1   901          2.70           3
## 153        1   825          4.95           3
## 156        1   802          3.78           2
## 157        1   818          4.24           3
## 158        1   736          3.78           3
## 161        1   862          4.17           4
## 165        1   938          3.69           3
## 169        1   898          2.26           4
## 170        1   777          4.86           3
## 184        1   780          4.15           4
## 189        1   716          3.44           3
## 199        1   794          3.83           3
## 200        1   818          2.70           1
## 209        1   857          3.85           2
## 212        1   828          1.37           4
## 218        1   752          4.89           2
## 223        1   898          3.51           3
## 228        1   756          3.09           3
## 230        1   940          2.82           3
## 231        1   757          3.55           3
## 238        1   739          3.99           3
```

```
## 240        1   943          3.21           4
## 241        1   742          4.17           2
## 248        1   753          4.89           2
## 253        1   882          2.63           3
## 255        1   922          4.15           1
## 256        1   793          4.08           2
## 257        1   787          2.56           3
## 273        1   886          4.68           3
## 275        1   718          4.03           4
## 279        1   841          5.00           4
## 281        1   842          3.99           3
## 290        1   720          3.76           3
## 292        1   871          5.00           2
## 299        1   835          3.90           1
## 303        1   858          3.67           3
## 306        1   707          2.38           1
## 307        1   798          4.72           3
## 313        1   930          4.22           4
## 321        1   732          3.57           2
## 329        1   789          3.71           3
## 330        1   795          4.31           1
## 332        1   754          4.33           4
## 336        1   787          3.14           1
## 342        1   940          5.00           4
## 344        1   796          5.00           3
```

```
salespeople_sales_part <- subset(salespeople, select = c("sales", "performance"))
head(salespeople_sales_part)
```

```
##   sales performance
## 1   594           2
## 2   446           3
## 3   674           4
## 4   525           2
## 5   657           3
## 6   918           2
```

head(salespeople_sales_perf)

```
low_sales <- subset(salespeople, subset = sales < 400)
```

```
# binds the rows of low_sales and high_sales together
low_and_high_sales = rbind(low_sales, high_sales)
head(low_and_high_sales)
```

```
##    promoted sales customer_rate performance
## 7         0   318          3.09           3
## 8         0   364          4.89           1
## 9         0   342          3.74           3
## 10        0   387          3.00           3
## 15        0   344          3.02           2
## 16        0   372          3.87           3
```

```
head(high_sales)
```

```
##    promoted sales customer_rate performance
## 6         1   918          4.54           2
## 12        1   716          3.16           3
## 20        1   937          5.00           2
## 21        1   702          3.53           4
## 25        1   819          4.45           2
## 26        1   736          3.94           4
```

```
head(low_and_high_sales)
```

```
##    promoted sales customer_rate performance
## 7         0   318          3.09           3
## 8         0   364          4.89           1
## 9         0   342          3.74           3
## 10        0   387          3.00           3
## 15        0   344          3.02           2
## 16        0   372          3.87           3
```

To select specific columns use the **select argument**.

```
# two dataframes with two columns each
sales_perf <- subset(salespeople, select = c("sales", "performance"))
prom_custrate <- subset(salespeople, select =  c("promoted", "customer_rate"))
```

```
# bind the columns to create a dataframes with four
full_df <- cbind(sales_perf, prom_custrate)
head(full_df)
```

```
##   sales performance promoted customer_rate
## 1   594           2        0          3.94
## 2   446           3        0          4.06
## 3   674           4        1          3.83
## 4   525           2        0          3.62
## 5   657           3        1          4.40
## 6   918           2        1          4.54
```

```
head(sales_perf)
```

```
##   sales performance
## 1   594           2
## 2   446           3
## 3   674           4
## 4   525           2
## 5   657           3
## 6   918           2
```

```
head(prom_custrate)
```

```
##   promoted customer_rate
## 1        0          3.94
## 2        0          4.06
## 3        1          3.83
## 4        0          3.62
## 5        1          4.40
## 6        1          4.54
```

```
head(full_df)
```

```
##   sales performance promoted customer_rate
## 1   594           2        0          3.94
## 2   446           3        0          4.06
## 3   674           4        1          3.83
## 4   525           2        0          3.62
## 5   657           3        1          4.40
## 6   918           2        1          4.54
```

```
head(full_df, n=5)
```

```
##   sales performance promoted customer_rate
## 1   594           2        0          3.94
## 2   446           3        0          4.06
## 3   674           4        1          3.83
## 4   525           2        0          3.62
## 5   657           3        1          4.40
```

4 Functions, Packages and Libraries

4.1 Using functions

Functions are operations that take certain defined inputs and return an output. It exist to perform common useful operations and usually take one or more arguments. Often there are a large number of arguments that a function can take, but many are optional and not required to be specified by the user. A second argument is optional, n: the number of rows to display. If n is not entered, it is assumed to have the default value n = 6. When running a function, you can either specify the arguments by name or you can enter them in order without their names. If you enter without naming, R expects the arguments to be entered in exactly the right order.

```
head(salespeople) # see the head of salespeople, with the default of six rows
```

```
##   promoted sales customer_rate performance
## 1        0   594          3.94           2
## 2        0   446          4.06           3
## 3        1   674          3.83           4
## 4        0   525          3.62           2
## 5        1   657          4.40           3
## 6        1   918          4.54           2
```

```
head(x = salespeople)
```

```
##   promoted sales customer_rate performance
## 1        0   594          3.94           2
## 2        0   446          4.06           3
## 3        1   674          3.83           4
## 4        0   525          3.62           2
## 5        1   657          4.40           3
## 6        1   918          4.54           2
```

```
head(salespeople, 3) # see fewer rows - arguments need to be in the right order if not named
```

```
##   promoted sales customer_rate performance
## 1        0   594          3.94           2
## 2        0   446          4.06           3
## 3        1   674          3.83           4
```

This case is when you don't know the right order, name your arguments and you can put them in any order

```
head(x = salespeople, n = 4)
```

```
##   promoted sales customer_rate performance
## 1        0   594          3.94           2
## 2        0   446          4.06           3
## 3        1   674          3.83           4
## 4        0   525          3.62           2
```

```
head(n = 6, x = salespeople)
```

```
##   promoted sales customer_rate performance
## 1        0   594          3.94           2
## 2        0   446          4.06           3
## 3        1   674          3.83           4
## 4        0   525          3.62           2
## 5        1   657          4.40           3
## 6        1   918          4.54           2
```

4.2 Help with Functions

Type **help(head)** or **?head** which will display the results in the Help browser window in Rstudio. Alternatively you can open the Help browser window directly in Rstudio and do a search there. Users can write their own functions to perform tasks that are helpful to their objective. Functions are not limited to those that come packaged in R. The help page normally shows the following:

- Description of the purpose of the function
- Usage examples, so you can quickly see how it is used
- Arguments list so you can see the names and order of arguments

- Details or notes on further considerations on use
- Expected value of the output (for example head() is expected to return a similar object to its first input x)
- Examples to help orient you further (sometimes examples can be very abstract in nature and not so helpful to users)

```
help()
```

```
## starting httpd help server ... done
```

```
help(head)
```

```
?head
```

4.3 Writing Your Own Functions

Experienced programmers in most languages subscribe to a principle called **DRY (Don't Repeat Yourself)**.Whenever a task needs to be done repeatedly, it is poor practice to write the same code numerous times.

```
df_report <- function(df){paste("This dataframe contains", nrow(df),
                                "rows and", ncol(df), "columns. There are", sum(is.na(df)),
                                "NA entries")} # create df_report function
```

```
df_report(mtcars)
```

```
## [1] "This dataframe contains 32 rows and 11 columns. There are 0 NA entries"
```

```
paste("This is how the paste", "functioning works")
```

```
## [1] "This is how the paste functioning works"
```

4.4 Installing Packages

The R programming language allows users to write and share their own functions and resources via packages, which are additional modules that can be easily installed to make resources available that are not in the base R installation. Use functions from both base R and popular packages, such as the **MASS package** for statistical modeling. To use an external package like MASS, it must be installed into the user's package library using the **"install.packages()"** command in the R console, which will download and install the package from the internet repository for R packages (CRAN), including any other packages that may be required.

```
my_packages <- c("MASS", "DescTools", "dplyr")
install.packages(my_packages)
```

```
## Installing packages into 'C:/Users/User/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)
```

```
## Error in contrib.url(repos, "source"): trying to use CRAN without setting a mirror
```

```
installed.packages("MASS")
```

```
##      Package LibPath Version Priority Depends Imports LinkingTo Suggests
##      Enhances License License_is_FOSS License_restricts_use OS_type Archs
##      MD5sum NeedsCompilation Built
```

```
my_packages <- c("DescTools", "dplyr")
```

```
installed.packages("my_packages")
```

```
##      Package LibPath Version Priority Depends Imports LinkingTo Suggests
##      Enhances License License_is_FOSS License_restricts_use OS_type Archs
##      MD5sum NeedsCompilation Built
```

4.5 Using Packages

```
library(MASS)
```

```
help(package = "MASS")
```

4.6 The Pipe Operator

The **pipe operator** makes code more natural to read and write and reduces the typical computing problem of many nested operations inside parentheses. It comes inside many R packages, particularly magrittr and dplyr.

For instance, imagine we wanted to do the following two operations in one command:

1. Subset salespeople to only the sales values of those with sales less than 500
2. Take the mean of those values

```
sales <- subset(salespeople, subset = sales < 500)
```

```
mean(sales$sales)
```

```
## [1] 388.6684
```

```
mean(subset(salespeople, subset = sales <500) $ sales)
```

```
## [1] 388.6684
```

```
mean(subset(salespeople$sales, subset = salespeople$sales<500))
```

```
## [1] 388.6684
```

This is nested and needs to be read from the inside out in order to align with the instructions. The **pipe operator** %>% takes the command that comes before it and places it inside the function that follows it (by default as the first argument). This reduces complexity and allows you to follow the logic more clearly.

```
library(magrittr) # load magrittr library to get the pipe operator
```

```
# use the pipe operator to lay out the steps more logically
subset(salespeople$sales, subset = salespeople$sales < 500) %>%
mean()
```

```
## [1] 388.6684
```

```
library(magrittr)
subset(salespeople, subset = sales < 500)$sales %>%
mean()
```

```
## [1] 388.6684
```

```
salespeople$sales %>% # start with all data
  subset(subset = salespeople$sales < 500) %>% # get the subsetted data
  mean() %>% # take the mean value
  round() # round to the nearest time
```

```
## [1] 389
```

5 Errors, Warnings and Messages

Errors are serious problems which usually result in the halting of your code and a failure to return your requested output. They usually come with an indication of the source of the error, and these can sometimes be easy to understand and sometimes frustratingly vague and abstract.

```
subset(salespeople, subset = sales = 700)
```

```
## Error: <text>:1:36: unexpected '='
## 1: subset(salespeople, subset = sales =
##                                       ^
```

The error message helps on what is wrong. It is an error to used sales = 720 as a condition to subset your data, when it should be sales == 720 for precise equality.

```
subset(salespeople, subset = sales == 700)
```

```
## [1] promoted      sales         customer_rate performance
## <0 rows> (or 0-length row.names)
```

```
head[salespeople]
```

```
## Error in head[salespeople]: object of type 'closure' is not subsettable
```

```
head(salespeople)
```

```
##   promoted sales customer_rate performance
## 1        0   594          3.94           2
## 2        0   446          4.06           3
## 3        1   674          3.83           4
## 4        0   525          3.62           2
## 5        1   657          4.40           3
## 6        1   918          4.54           2
```

```
salespeople[1,0]
```

```
## data frame with 0 columns and 1 row
```

Warnings are less serious and usually alert you to something that you might be overlooking and which could indicate a problem with the output. In many cases you can ignore warnings, but sometimes they are an important reminder to go back and edit your code.

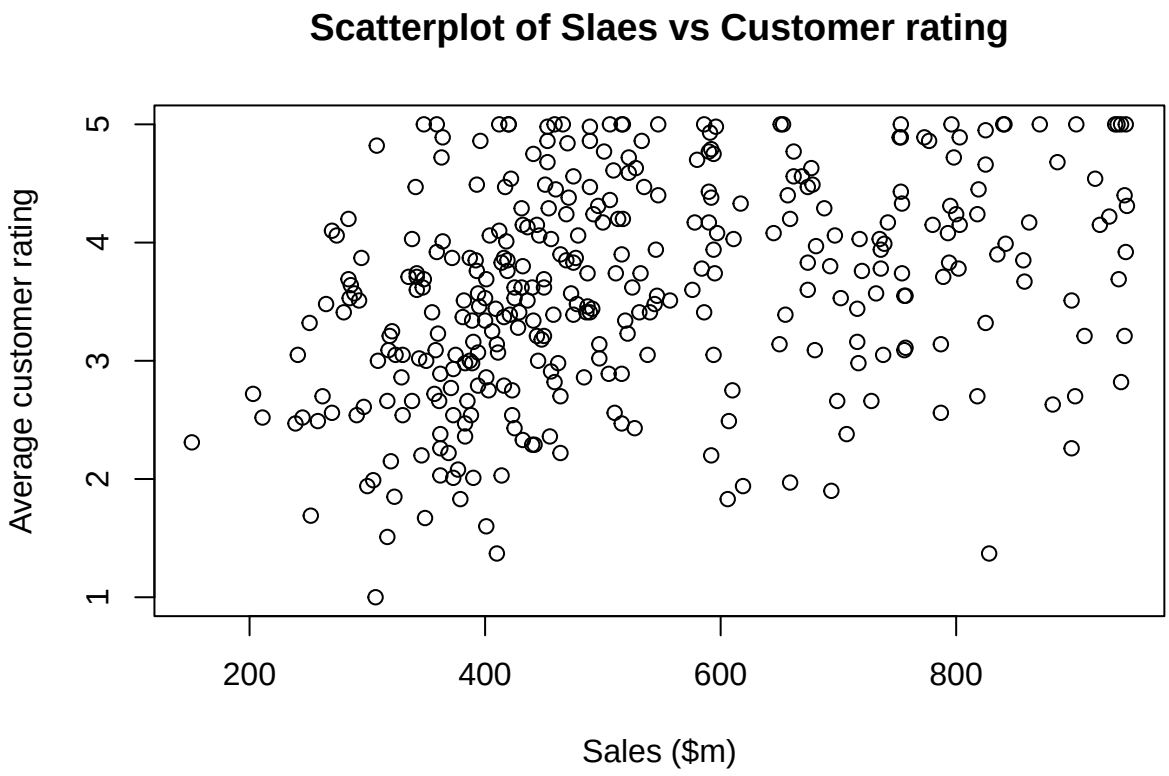
Messages are pieces of information that may or may not be useful to you at a particular point in time. Sometimes you will receive messages when you load a package from your library. Sometimes messages will keep you up to date on the progress of a process that is taking a long time to execute.

6 Plotting and Graphing

6.1 Plotting in Base R

The simplest plot function in base R is **plot()**. This performs basic X-Y plotting. As an example, this code will generate a scatter plot of customer_rate against sales in the salespeople data set. \Note the use of the arguments **main, xlab and ylab** for customizing the axis labels and title for the plot.

```
# scatter plot of customer_rate against sales
plot(x =  salespeople$sales, y =  salespeople$customer_rate,
     xlab = "Sales ($m)", ylab = "Average customer rating",
     main = "Scatterplot of Slaes vs Customer rating")
```

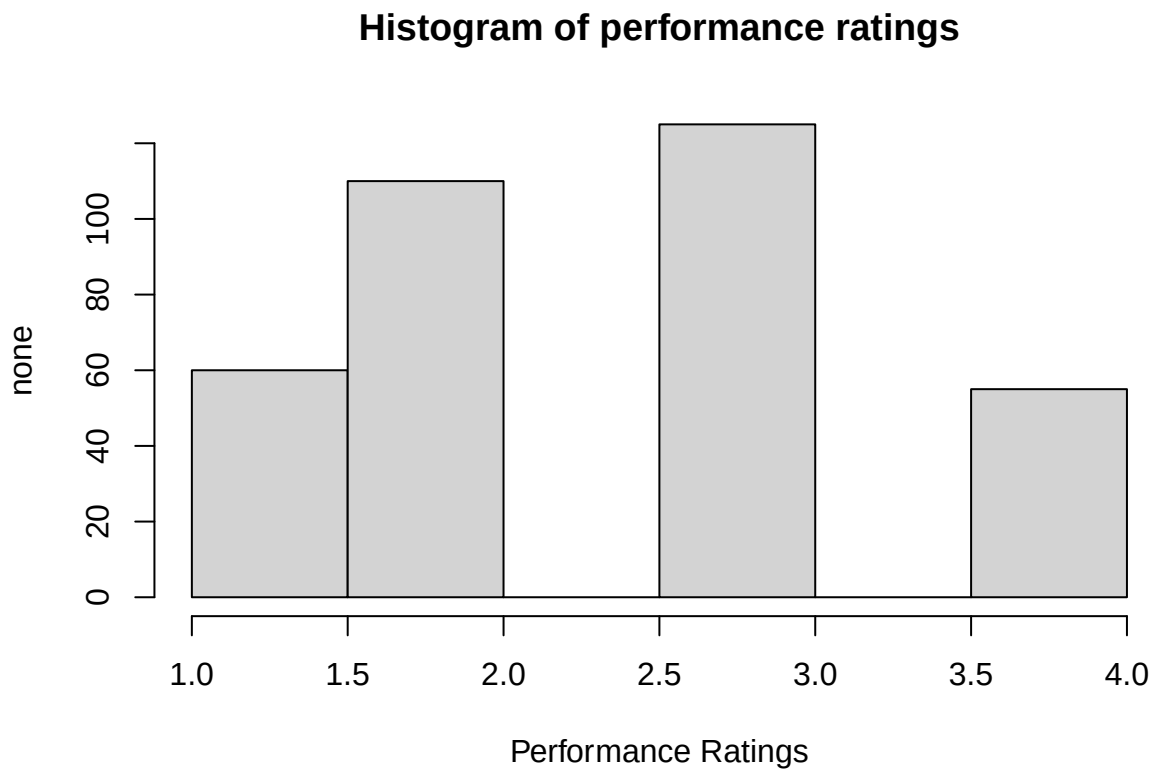


Histograms of data can be generated using the **hist()**. \Note the use of **breaks** to customize how the bars appear.

```
# Convert performance ratings back to numeric data type for histogram
salespeople$performance <- as.numeric(salespeople$performance)
```

```
hist(salespeople$performance) #histogram of performance ratings
```

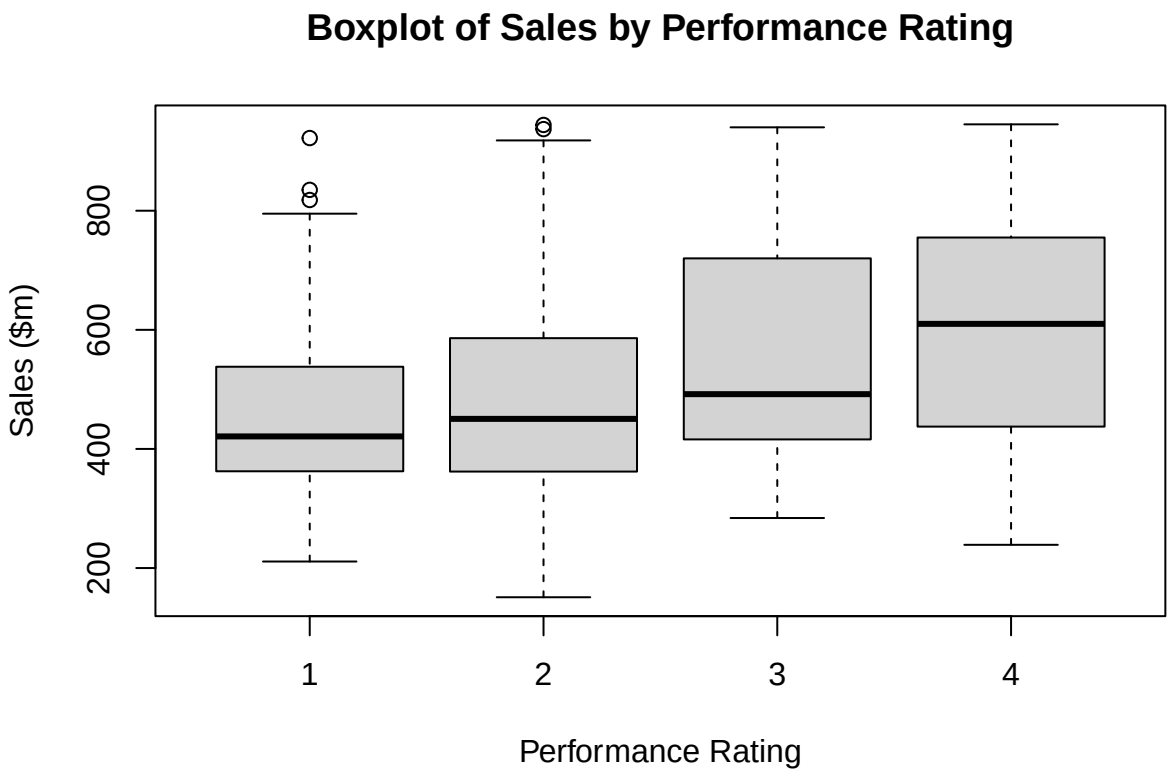

```
hist(salespeople$performance, xlab = "Performance Ratings",
     ylab = "none", main = "Histogram of performance ratings")
```




```
hist(salespeople$performance, breaks = 0:4)
```

Box and whisker plots are excellent ways to see the distribution of a variable, and can be grouped against another variable to see bivariate patterns. \Note the use of the **formula** and **data** notation here to define the variable we are interested in and how we want it grouped. We will study this formula notation in greater depth later in this book.

```
boxplot(formula = sales ~ performance, data = salespeople,
        xlab = "Performance Rating", ylab = "Sales ($m)",
        main = "Boxplot of Sales by Performance Rating") # box plot of sales by performance rating
```



6.2 Specialist plotting and graphing packages

ggplot2 is the most commonly used and allows for flexible construction of a wide range of charts and graphs, though it uses a specific command grammar that may take some getting used to. **Plotly** is an excellent package for interactive graphing, and **GGally** extends ggplot2 to allow easy combination of charts and graphs, which is particularly useful for exploratory data analysis. **GGally's ggpairs()** function produces a pairplot, which is a visualization of all univariate and bivariate patterns in a data set

```
install.packages(
   "ggplot2",
   repos = c("http://rstudio.org/_packages",
   "http://cran.rstudio.com")
) # convert performance and promotion to categorical
```

```
## Installing package into 'C:/Users/User/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)
```

```
## package 'ggplot2' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\User\AppData\Local\Temp\RtmpcfRtF5\downloaded_packages
```

```
library(GGally)
```

```
## Loading required package: ggplot2
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
salespeople$promoted <- as.factor(salespeople$promoted)
salespeople$performance <- as.factor(salespeople$performance)
```

6.3 Pairplot of salespeople

```
GGally::ggpairs(salespeople) # Pairplot of salespeople
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```