7-r-data-structures

April 22, 2024

1 7. R Data Structures

1.1 7.1 Vectors

Recall that vectors may have mode logical, numeric or character.

7.1.1 Subsets of Vectors

Remember how we can extract subsets of vectors (as discussed in section 2.6.2): - We can specify the indices of the elements to be extracted, where negative indices omit elements. - We can use a logical vector to specify which elements to extract based on their logical value. - Additionally, for vectors with named elements, there's a third option:

[ ]:
```
data <- c(Andreas = 178, John = 185, Jeff = 183)[c("John", "Jeff")]
data


#A vector of names has been used to extract the elements.
```

John 185 **Jeff** 183

7.1.2 Patterned Data

Use 5:15 to generate the numbers 5, 6, …, 15. Entering 15:5 will generate the sequence in the reverse order. To repeat the sequence (2, 3, 5) four times over, enter rep(c(2,3,5), 4) thus:

[ ]:
```
rep(c(2,3,5),4)
```

1. 2 2. 3 3. 5 4. 2 5. 3 6. 5 7. 2 8. 3 9. 5 10. 2 11. 3 12. 5

If instead one wants four 2s, then four 3s, then four 5s, enter rep(c(2,3,5), c(4,4,4)).

[ ]:
```
rep(c(2,3,5),c(4,4,4)) # An alternative is rep(c(2,3,5), each=4)
```

1. 2 2. 2 3. 2 4. 2 5. 3 6. 3 7. 3 8. 3 9. 5 10. 5 11. 5 12. 5

You can simplify it like this:

Instead of `c(4, 4, 4)`, you can use `rep(4, 3)`. So instead of `rep(c(2, 3, 5), c(4, 4, 4))`, you can use `rep(c(2, 3, 5), rep(4, 3))`.

Also, remember that `rep()` has a `length.out` argument, which repeats the sequence until it reaches the specified length.

1.2 7.2 Missing Values

In R, NA represents a missing value. Any arithmetic operation or relation involving NA results in NA. This includes comparisons such as <, <=, >, >=, ==, and !=. The first four compare magnitudes, == checks for equality, and != checks for inequality. It's essential to handle NA values properly to avoid unexpected results. For instance, x == NA produces NA, so it's better to use is.na(x) to determine which values of x are NA. Using x == NA doesn't provide any useful information about x.

[ ]:
```
x <- c(1,6,2,NA)
is.na(x) # TRUE for when NA appears, and otherwise FALSE
```

1. FALSE 2. FALSE 3. FALSE 4. TRUE

[ ]:
```
x==NA  #All elements are set to NA
```

1. <NA> 2. <NA> 3. <NA> 4. <NA>

[ ]:
```
NA==NA
```

<NA>

Missing values in subscripts: In R

[ ]:
```
y[x>2]  <- x[x>2]    #generates the error message "NAs are not allowed in␣
↪subscripted assignments".
```

```
Error: object 'y' not found
Traceback:
```

Users are advised to use !is.na(x) to limit the selection, on one or both sides as necessary, to those elements of x that are not NAs. We will have more to say on missing values in the section on data frames that now follows

1.3 7.3 Data frames

Data frames are like tables in R that store data. They're more flexible than matrices because each column can have a different type of data, like numbers, text, or categories. However, all values in a column must be of the same type. If you have a data frame with only numbers in each column, it's similar to a matrix, but not exactly the same. You can convert a data frame to a matrix using the as.matrix() function. We'll talk more about lists, another important concept, later on.

7.3.1 Extraction of Component Parts of Data frames

Consider the data frame barley that accompanies the lattice package:

[ ]:
```
barley <- read.csv("/content/barley.csv")
names(barley)
```

1. 'rownames' 2. 'yield' 3. 'variety' 4. 'year' 5. 'site'

[ ]:
```
# Check the structure of the barley dataset
str(barley)
```

```
'data.frame':	120 obs. of  5 variables:
 $ rownames: int  1 2 3 4 5 6 7 8 9 10 …
 $ yield   : num  27 48.9 27.4 39.9 33 …
 $ variety : chr  "Manchuria" "Manchuria" "Manchuria" "Manchuria" …
 $ year    : int  1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 …
 $ site    : chr  "University Farm" "Waseca" "Morris" "Crookston" …
```

[ ]:
```
# If the "site" variable is not a factor, convert it to factor
barley$site <- factor(barley$site)

# Check the levels of the "site" variable
levels(barley$site)
```

1. 'Crookston' 2. 'Duluth' 3. 'Grand Rapids' 4. 'Morris' 5. 'University Farm' 6. 'Waseca'

[ ]:
```
Duluth1932  <- barley[barley$year == "1932"  & barley$site == "Duluth",␣
  ↪c("variety", "yield")]
Duluth1932
```

A data.frame: 10 × 2

	variety <chr>	yield <dbl>
66	Manchuria	22.56667
72	Glabron	25.86667
78	Svansota	22.23333
84	Velvet	22.46667
90	Trebi	30.60000
96	No. 457	22.70000
102	No. 462	22.50000
108	Peatland	31.36667
114	No. 475	27.36667
120	Wisconsin No. 38	29.33333

The first column holds the row labels, which in this case are the numbers of the rows that have been extracted. In place of c("variety","yield") we could have written, more simply, c(2,4).

7.3.2 Data Sets that Accompany R Packages

To see a list of available datasets, just type data() in your R console. If you want to know what datasets are included in a specific package like 'datasets', use data(package='datasets').

For most packages, datasets are automatically accessible once the package is loaded. For example, you don't need to load the 'airquality' dataset separately; it's available once you load the 'datasets' package.

When you install R, it comes with many commonly used packages already included. However, some packages need to be installed explicitly. In these notes, we'll use the MASS package occasionally, which is included in the default distribution.

At the beginning of your session, R automatically loads some packages including the base package. You can load other installed packages using the library() command.

1.4 7.4 Data Entry Issues

7.4.1 Idiosyncrasies

The read.table() function is great for reading numeric data arranged in rows and columns. However, if your data has text in one of the columns, R will automatically turn it into a 'factor', which is like a category with different levels for each unique text string.

Sometimes, R may misunderstand your data, especially if there are small mistakes like using 'O' instead of '0' or 'l' instead of '1'. If you use symbols like '*',.', or blank spaces, R might also think the column contains text instead of numbers.

To avoid this confusion, you can use the as.is = TRUE parameter with read.table(). This tells R to keep the columns as they are without turning them into factors. Later, if you need to use a column as a factor, you can convert it manually or some functions will do it automatically for you.

7.4.2 Missing values when using read.table()

When using read.table(), remember to code missing values as NA. If your data file comes from SAS, you'll likely need to set na.strings = c("."). Sometimes, there might be different characters indicating missing values, like "NA", ".", "", or just empty cells. You can handle multiple indicators by setting na.strings = c("NA",".",""). The empty quotes"" ensure that empty cells are also treated as NAs.

7.4.3 Separators when using read.table()

With data from spreadsheets37, it is sometimes necessary to use tab ("\t") or comma as the separator. The default separator is white space. To set tab as the separator, specify sep="\t".

1.5 7.5 Factors and Ordered Factors

In our earlier discussion, we learned about factors (section 2.6.4). They're great for efficiently storing character strings when there are many repeats of the same strings. They're especially useful for including qualitative effects in model and graphics formulas.

Factors act like two things in one. They're stored as integer vectors, but each value is interpreted based on a table of levels.

For example, let's look at the islandcities dataset that comes with these notes. It contains the populations of 19 island nation cities with urban center populations of 1.4 million or more in 1995. The row names are city names, the first column is the country name, and the second column is the urban center population in millions. Here's a table showing how many times each country appears:

- Australia: 3
- Cuba: 1
- Indonesia: 4
- Japan: 6
- Philippines: 2
- Taiwan: 1
- United Kingdom: 2

When you print the column named "country," you see the names, not the underlying integer values. This translation happens automatically in R for most factor operations, but there are some exceptions that can be tricky. To ensure you get the country names, specify them explicitly.

[ ]:
```
islandcities <- read.csv("/content/islandcities.csv")
as.character(islandcities$country)
```

1. 'UGA' 2. 'UGA' 3. 'ITA' 4. 'ITA' 5. 'ITA' 6. 'ALD' 7. 'PSE' 8. 'VAT' 9. 'FRA' 10. 'FRA' 11. 'FRA' 12. 'ATA' 13. 'ZMB' 14. 'UGA' 15. 'UGA' 16. 'UGA' 17. 'UGA' 18. 'UGA' 19. 'UGA' 20. 'NAM' 21. 'ITA' 22. 'ITA' 23. 'ITA' 24. 'ITA' 25. 'ITA' 26. 'ITA' 27. 'ITA' 28. 'ITA' 29. 'ITA' 30. 'FRA' 31. 'JPN' 32. 'FRA' 33. 'FRA' 34. 'FRA' 35. 'FRA' 36. 'FRA' 37. 'FRA' 38. 'FRA' 39. 'FRA' 40. 'FRA' 41. 'FRA' 42. 'FRA' 43. 'FRA' 44. 'SRB' 45. 'BIH' 46. 'GBR' 47. 'GBR' 48. 'GBR' 49. 'SMR' 50. 'NLD' 51. 'NLD' 52. 'LIE' 53. 'ATA' 54. 'ATA' 55. 'ATA' 56. 'ATA' 57. 'ATA' 58. 'ATA' 59. 'ATA' 60. 'ATA' 61. 'ATA' 62. 'ATA' 63. 'ATA' 64. 'ATA' 65. 'ATA' 66. 'ATA' 67. 'ATA' 68. 'ATA' 69. 'ATA' 70. 'ATA' 71. 'ATA' 72. 'ATA' 73. 'ATA' 74. 'ATA' 75. 'ATA' 76. 'ATA' 77. 'ATA' 78. 'ATA' 79. 'ATA' 80. 'ATA' 81. 'ATA' 82. 'ATA' 83. 'ATA' 84. 'ATA' 85. 'ATA' 86. 'ATA' 87. 'ATA' 88. 'ATA' 89. 'ATA' 90. 'ATA' 91. 'ATA' 92. 'GBR' 93. 'SWZ' 94. 'GBR' 95. 'GBR' 96. 'NOR' 97. 'LUX' 98. 'ITA' 99. 'FRA' 100. 'JPN' 101. 'FRA' 102. 'FRA' 103. 'FRA' 104. 'FRA' 105. 'FRA' 106. 'GEO' 107. 'PRT' 108. 'SDN' 109. 'NAM' 110. 'NAM' 111. 'ITA' 112. 'GEO' 113. 'FSM' 114. 'MHL' 115. 'USA' 116. 'TUV' 117. 'PLW' 118. 'SAH' 119. 'MCO' 120. 'KIR' 121. 'COM' 122. 'CHN' 123. 'AND' 124. 'USA' 125. 'USA' 126. 'USA' 127. 'GBR' 128. 'PAK' 129. 'KOR' 130. 'NGA' 131. 'CHN' 132. 'CHN' 133. 'CHN' 134. 'CHN' 135. 'CHN' 136. 'CHN' 137. 'CHN' 138. 'CHN' 139. 'CHN' 140. 'CHN' 141. 'CHN' 142. 'IND' 143. 'IND' 144. 'IND' 145. 'IND' 146. 'IND' 147. 'IND' 148. 'IND' 149. 'IND' 150. 'IND' 151. 'IND' 152. 'IND' 153. 'IND' 154. 'IND' 155. 'IND' 156. 'IND' 157. 'IND' 158. 'IND' 159. 'URY' 160. 'USA' 161. 'USA' 162. 'USA' 163. 'USA' 164. 'USA' 165. 'USA' 166. 'USA' 167. 'USA' 168. 'USA' 169. 'USA' 170. 'USA' 171. 'USA' 172. 'USA' 173. 'USA' 174. 'USA' 175. 'USA' 176. 'USA' 177. 'USA' 178. 'USA' 179. 'USA' 180. 'USA' 181. 'USA' 182. 'USA' 183. 'USA' 184. 'USA' 185. 'USA' 186. 'USA' 187. 'USA' 188. 'USA' 189. 'USA' 190. 'USA' 191. 'USA' 192. 'USA' 193. 'USA' 194. 'VEN' 195. 'USA' 196. 'USA' 197. 'USA' 198. 'USA' 199. 'USA' 200. 'USA' 201. 202. 'GBR' 203. 'GBR' 204. 'TON' 205. 'SOL' 206. 'SYC' 207. 'STP' 208. 'WSM' 209. 'MLT' 210. 'MDV' 211. 'ISR' 212. 'CPV' 213. 'BHS' 214. 'CYP' 215. 'TWN' 216. 'CHN' 217. 'CHN' 218. 'USA' 219. 'USA' 220. 'USA' 221. 'USA' 222. 'USA' 223. 'USA' 224. 'USA' 225. 'USA' 226. 'VEN' 227. 'USA' 228. 'USA' 229. 'USA' 230. 'USA' 231. 'USA' 232. 'VNM' 233. 'VNM' 234. 'TUR' 235. 'HUN' 236. 'YEM' 237. 'ESP' 238. 'ROU' 239. 'SYR' 240. 'SYR' 241. 'CHE' 242. 'PRT' 243. 'SDN' 244. 'SAU' 245. 'SAU' 246. 'NOR' 247. 'PAK' 248. 'PAK' 249. 'ZAF' 250. 'RUS' 251. 'MEX' 252. 'MEX' 253. 'NGA' 254. 'POL' 255. 'PRK' 256. 'TZA' 257. 'IDN' 258. 'IRL' 259. 'LBR' 260. 'ITA' 261. 'ITA' 262. 'MYS' 263. 'CHN' 264. 'CHN' 265. 'CHN' 266. 'CHN' 267. 'CHN' 268. 'CHN' 269. 'CHN' 270. 'IDN' 271. 'IDN' 272. 'ECU' 273. 'COL' 274. 'COL' 275. 'CUB' 276. 'EGY' 277. 'DEU' 278. 'DEU' 279. 'DEU' 280. 'CZE' 281. 'KWT' 282. 'CHN' 283. 'CHN' 284. 'CHN' 285. 'CHN' 286. 'CHN' 287. 'CHN' 288. 'CHN' 289. 'CHN' 290. 'CHN' 291. 'CHN' 292. 'CHN' 293. 'CHN' 294. 'JPN' 295. 'CHN' 296. 'CHN' 297. 'CHN' 298. 'JPN' 299. 'DOM' 300. 'GHA' 301. 'IND' 302. 'IND' 303. 'IND' 304. 'IND' 305. 'LBY' 306. 'ISR' 307. 'FIN' 308. 'IRN' 309. 'IND' 310. 'IND' 311. 'IND' 312. 'IND' 313. 'IND' 314. 'IND' 315. 'DNK' 316. 'CIV' 317. 'BRA' 318. 'BRA' 319. 'BRA' 320. 'BRA' 321. 'BRA' 322. 'BRA' 323. 'CAN' 324. 'CAN' 325. 'BRA' 326. 'BRA' 327. 'BEL' 328. 'BGD' 329. 'AGO' 330. 'DZA' 331. 'BGD' 332. 'AUS' 333. 'MMR' 334. 'USA' 335. 'USA' 336. 'USA' 337. 'USA' 338. 'USA' 339. 'USA' 340. 'VEN' 341. 'UKR' 342. 'ARE' 343. 'UZB' 344. 'ESP' 345. 'CHE' 346. 'SWE' 347. 'THA'

348. 'PER' 349. 'SEN' 350. 'ZAF' 351. 'NLD' 352. 'MAR' 353. 'KOR' 354. 'PHL' 355. 'MEX' 356. 'NZL' 357. 'DEU' 358. 'CHN' 359. 'CHN' 360. 'JPN' 361. 'COD' 362. 'IND' 363. 'IND' 364. 'GRC' 365. 'IRQ' 366. 'ETH' 367. 'IRN' 368. 'CAN' 369. 'CAN' 370. 'ARG' 371. 'AFG' 372. 'AUT' 373. 'AUS' 374. 'TWN' 375. 'USA' 376. 'USA' 377. 'USA' 378. 'GBR' 379. 'TUR' 380. 'SAU' 381. 'ZAF' 382. 'RUS' 383. 'MEX' 384. 'NGA' 385. 'ITA' 386. 'CHN' 387. 'KEN' 388. 'IDN' 389. 'COL' 390. 'EGY' 391. 'CHN' 392. 'JPN' 393. 'IND' 394. 'FRA' 395. 'CHL' 396. 'IND' 397. 'BRA' 398. 'BRA' 399. 'AUS' 400. 'SGP' 401. 'CHN'

[ ]:
```
#to get the integer values, specify
unclass(islandcities$country)
```

1. 'UGA' 2. 'UGA' 3. 'ITA' 4. 'ITA' 5. 'ITA' 6. 'ALD' 7. 'PSE' 8. 'VAT' 9. 'FRA' 10. 'FRA' 11. 'FRA' 12. 'ATA' 13. 'ZMB' 14. 'UGA' 15. 'UGA' 16. 'UGA' 17. 'UGA' 18. 'UGA' 19. 'UGA' 20. 'NAM' 21. 'ITA' 22. 'ITA' 23. 'ITA' 24. 'ITA' 25. 'ITA' 26. 'ITA' 27. 'ITA' 28. 'ITA' 29. 'ITA' 30. 'FRA' 31. 'JPN' 32. 'FRA' 33. 'FRA' 34. 'FRA' 35. 'FRA' 36. 'FRA' 37. 'FRA' 38. 'FRA' 39. 'FRA' 40. 'FRA' 41. 'FRA' 42. 'FRA' 43. 'FRA' 44. 'SRB' 45. 'BIH' 46. 'GBR' 47. 'GBR' 48. 'GBR' 49. 'SMR' 50. 'NLD' 51. 'NLD' 52. 'LIE' 53. 'ATA' 54. 'ATA' 55. 'ATA' 56. 'ATA' 57. 'ATA' 58. 'ATA' 59. 'ATA' 60. 'ATA' 61. 'ATA' 62. 'ATA' 63. 'ATA' 64. 'ATA' 65. 'ATA' 66. 'ATA' 67. 'ATA' 68. 'ATA' 69. 'ATA' 70. 'ATA' 71. 'ATA' 72. 'ATA' 73. 'ATA' 74. 'ATA' 75. 'ATA' 76. 'ATA' 77. 'ATA' 78. 'ATA' 79. 'ATA' 80. 'ATA' 81. 'ATA' 82. 'ATA' 83. 'ATA' 84. 'ATA' 85. 'ATA' 86. 'ATA' 87. 'ATA' 88. 'ATA' 89. 'ATA' 90. 'ATA' 91. 'ATA' 92. 'GBR' 93. 'SWZ' 94. 'GBR' 95. 'GBR' 96. 'NOR' 97. 'LUX' 98. 'ITA' 99. 'FRA' 100. 'JPN' 101. 'FRA' 102. 'FRA' 103. 'FRA' 104. 'FRA' 105. 'FRA' 106. 'GEO' 107. 'PRT' 108. 'SDN' 109. 'NAM' 110. 'NAM' 111. 'ITA' 112. 'GEO' 113. 'FSM' 114. 'MHL' 115. 'USA' 116. 'TUV' 117. 'PLW' 118. 'SAH' 119. 'MCO' 120. 'KIR' 121. 'COM' 122. 'CHN' 123. 'AND' 124. 'USA' 125. 'USA' 126. 'USA' 127. 'GBR' 128. 'PAK' 129. 'KOR' 130. 'NGA' 131. 'CHN' 132. 'CHN' 133. 'CHN' 134. 'CHN' 135. 'CHN' 136. 'CHN' 137. 'CHN' 138. 'CHN' 139. 'CHN' 140. 'CHN' 141. 'CHN' 142. 'IND' 143. 'IND' 144. 'IND' 145. 'IND' 146. 'IND' 147. 'IND' 148. 'IND' 149. 'IND' 150. 'IND' 151. 'IND' 152. 'IND' 153. 'IND' 154. 'IND' 155. 'IND' 156. 'IND' 157. 'IND' 158. 'IND' 159. 'URY' 160. 'USA' 161. 'USA' 162. 'USA' 163. 'USA' 164. 'USA' 165. 'USA' 166. 'USA' 167. 'USA' 168. 'USA' 169. 'USA' 170. 'USA' 171. 'USA' 172. 'USA' 173. 'USA' 174. 'USA' 175. 'USA' 176. 'USA' 177. 'USA' 178. 'USA' 179. 'USA' 180. 'USA' 181. 'USA' 182. 'USA' 183. 'USA' 184. 'USA' 185. 'USA' 186. 'USA' 187. 'USA' 188. 'USA' 189. 'USA' 190. 'USA' 191. 'USA' 192. 'USA' 193. 'USA' 194. 'VEN' 195. 'USA' 196. 'USA' 197. 'USA' 198. 'USA' 199. 'USA' 200. 'USA' 201. 202. 'GBR' 203. 'GBR' 204. 'TON' 205. 'SOL' 206. 'SYC' 207. 'STP' 208. 'WSM' 209. 'MLT' 210. 'MDV' 211. 'ISR' 212. 'CPV' 213. 'BHS' 214. 'CYP' 215. 'TWN' 216. 'CHN' 217. 'CHN' 218. 'USA' 219. 'USA' 220. 'USA' 221. 'USA' 222. 'USA' 223. 'USA' 224. 'USA' 225. 'USA' 226. 'VEN' 227. 'USA' 228. 'USA' 229. 'USA' 230. 'USA' 231. 'USA' 232. 'VNM' 233. 'VNM' 234. 'TUR' 235. 'HUN' 236. 'YEM' 237. 'ESP' 238. 'ROU' 239. 'SYR' 240. 'SYR' 241. 'CHE' 242. 'PRT' 243. 'SDN' 244. 'SAU' 245. 'SAU' 246. 'NOR' 247. 'PAK' 248. 'PAK' 249. 'ZAF' 250. 'RUS' 251. 'MEX' 252. 'MEX' 253. 'NGA' 254. 'POL' 255. 'PRK' 256. 'TZA' 257. 'IDN' 258. 'IRL' 259. 'LBR' 260. 'ITA' 261. 'ITA' 262. 'MYS' 263. 'CHN' 264. 'CHN' 265. 'CHN' 266. 'CHN' 267. 'CHN' 268. 'CHN' 269. 'CHN' 270. 'IDN' 271. 'IDN' 272. 'ECU' 273. 'COL' 274. 'COL' 275. 'CUB' 276. 'EGY' 277. 'DEU' 278. 'DEU' 279. 'DEU' 280. 'CZE' 281. 'KWT' 282. 'CHN' 283. 'CHN' 284. 'CHN' 285. 'CHN' 286. 'CHN' 287. 'CHN' 288. 'CHN' 289. 'CHN' 290. 'CHN' 291. 'CHN' 292. 'CHN' 293. 'CHN' 294. 'JPN' 295. 'CHN' 296. 'CHN' 297. 'CHN' 298. 'JPN' 299. 'DOM' 300. 'GHA' 301. 'IND' 302. 'IND' 303. 'IND' 304. 'IND' 305. 'LBY' 306. 'ISR' 307. 'FIN' 308. 'IRN' 309. 'IND' 310. 'IND' 311. 'IND' 312. 'IND' 313. 'IND' 314. 'IND' 315. 'DNK'

316. 'CIV' 317. 'BRA' 318. 'BRA' 319. 'BRA' 320. 'BRA' 321. 'BRA' 322. 'BRA' 323. 'CAN' 324. 'CAN' 325. 'BRA' 326. 'BRA' 327. 'BEL' 328. 'BGD' 329. 'AGO' 330. 'DZA' 331. 'BGD' 332. 'AUS' 333. 'MMR' 334. 'USA' 335. 'USA' 336. 'USA' 337. 'USA' 338. 'USA' 339. 'USA' 340. 'VEN' 341. 'UKR' 342. 'ARE' 343. 'UZB' 344. 'ESP' 345. 'CHE' 346. 'SWE' 347. 'THA' 348. 'PER' 349. 'SEN' 350. 'ZAF' 351. 'NLD' 352. 'MAR' 353. 'KOR' 354. 'PHL' 355. 'MEX' 356. 'NZL' 357. 'DEU' 358. 'CHN' 359. 'CHN' 360. 'JPN' 361. 'COD' 362. 'IND' 363. 'IND' 364. 'GRC' 365. 'IRQ' 366. 'ETH' 367. 'IRN' 368. 'CAN' 369. 'CAN' 370. 'ARG' 371. 'AFG' 372. 'AUT' 373. 'AUS' 374. 'TWN' 375. 'USA' 376. 'USA' 377. 'USA' 378. 'GBR' 379. 'TUR' 380. 'SAU' 381. 'ZAF' 382. 'RUS' 383. 'MEX' 384. 'NGA' 385. 'ITA' 386. 'CHN' 387. 'KEN' 388. 'IDN' 389. 'COL' 390. 'EGY' 391. 'CHN' 392. 'JPN' 393. 'IND' 394. 'FRA' 395. 'CHL' 396. 'IND' 397. 'BRA' 398. 'BRA' 399. 'AUS' 400. 'SGP' 401. 'CHN'

By default, R sorts the level names in alphabetical order. If we form a table that has the number of times that each country appears, this is the order that is used:

[ ]:
```
table(islandcities$country)
```

```
AFG AGO ALB ALD AND ARE ARG ARM ATA ATG AUS AUT AZE BDI BEL BEN BFA BGD BGR BHR 
  4   7   1   1   1   2  20   1  40   1  36   1   1   1   1   3   2   4   1   1 
BHS BIH BLR BLZ BOL BRA BRB BRN BTN BWA CAF CAN CHE CHL CHN CIV CMR COD COG COL 
  2   2   2   1   6  46   1   1   1   4   4  45   3  13 100   3   6  15   3   9 
COM CPV CRI CUB CYP CZE DEU DJI DMA DNK DOM DZA ECU EGY ERI ESP EST ETH FIN FJI 
  1   1   2   2   1   1   5   1   1   6   2   6   4   8   2   8   1   5   4   1 
FRA FSM GAB GBR GEO GHA GIN GMB GNB GNQ GRC GRD GTM GUY HND HRV HTI HUN IDN IND 
 30   1   3  13   3   1   3   1   1   1   2   1   2   1   1   1   1   1  23  69 
IRL IRN IRQ ISL ISR ITA JAM JOR JPN KAZ KEN KGZ KHM KIR KNA KOR KOS KWT LAO LBN 
  1   9   5   1   2  21   1   1  18  13   4   1   3   1   1   5   1   1   1   1 
LBR LBY LCA LIE LKA LSO LTU LUX LVA MAR MCO MDA MDG MDV MEX MHL MKD MLI MLT MMR 
  1   7   1   1   3   1   1   1   1   7   1   1   5   1  27   1   1   6   1   4 
MNE MNG MOZ MRT MUS MWI MYS NAM NER NGA NIC NLD NOR NPL NZL OMN PAK PAN PER PHL 
  1   5   8   4   1   3   4   7   5  13   1   4   7   1   8   1   9   2  11   8 
PLW PNG POL PRK PRT PRY PSE QAT ROU RUS RWA SAH SAU SDN SEN SGP SLB SLE SLV SMR 
  1   5   3   3   3   2   2   1   3  81   1   1   5   9   3   1   1   1   1   1 
SOL SOM SRB SSD STP SUR SVK SVN SWE SWZ SYC SYR TCD TGO THA TJK TKM TLS TON TTO 
  2   1   2   3   1   2   1   1   4   2   1   2   3   2   4   1   4   1   1   1 
TUN TUR TUV TWN TZA UGA UKR URY USA UZB VAT VCT VEN VNM VUT WSM YEM ZAF ZMB ZWE 
  2   7   1   3   7  10   8   2 114   5   1   1   7   4   1   1   3  13   6   3 
```

This order of the level names is purely a convenience. We might prefer countries to appear in order of latitude, from North to South. We can change the order of the level names to reflect this desired order:

[ ]:
```
# Convert country column to factor
islandcities$country <- as.factor(islandcities$country)

lev <- levels(islandcities$country)

lev
```

1. 'AFG' 2. 'AGO' 3. 'ALB' 4. 'ALD' 5. 'AND' 6. 'ARE' 7. 'ARG' 8. 'ARM' 9. 'ATA' 10. 'ATG' 11. 'AUS' 12. 'AUT' 13. 'AZE' 14. 'BDI' 15. 'BEL' 16. 'BEN' 17. 'BFA' 18. 'BGD' 19. 'BGR' 20. 'BHR' 21. 'BHS' 22. 'BIH' 23. 'BLR' 24. 'BLZ' 25. 'BOL' 26. 'BRA' 27. 'BRB' 28. 'BRN' 29. 'BTN' 30. 'BWA' 31. 'CAF' 32. 'CAN' 33. 'CHE' 34. 'CHL' 35. 'CHN' 36. 'CIV' 37. 'CMR' 38. 'COD' 39. 'COG' 40. 'COL' 41. 'COM' 42. 'CPV' 43. 'CRI' 44. 'CUB' 45. 'CYP' 46. 'CZE' 47. 'DEU' 48. 'DJI' 49. 'DMA' 50. 'DNK' 51. 'DOM' 52. 'DZA' 53. 'ECU' 54. 'EGY' 55. 'ERI' 56. 'ESP' 57. 'EST' 58. 'ETH' 59. 'FIN' 60. 'FJI' 61. 'FRA' 62. 'FSM' 63. 'GAB' 64. 'GBR' 65. 'GEO' 66. 'GHA' 67. 'GIN' 68. 'GMB' 69. 'GNB' 70. 'GNQ' 71. 'GRC' 72. 'GRD' 73. 'GTM' 74. 'GUY' 75. 'HND' 76. 'HRV' 77. 'HTI' 78. 'HUN' 79. 'IDN' 80. 'IND' 81. 'IRL' 82. 'IRN' 83. 'IRQ' 84. 'ISL' 85. 'ISR' 86. 'ITA' 87. 'JAM' 88. 'JOR' 89. 'JPN' 90. 'KAZ' 91. 'KEN' 92. 'KGZ' 93. 'KHM' 94. 'KIR' 95. 'KNA' 96. 'KOR' 97. 'KOS' 98. 'KWT' 99. 'LAO' 100. 'LBN' 101. 'LBR' 102. 'LBY' 103. 'LCA' 104. 'LIE' 105. 'LKA' 106. 'LSO' 107. 'LTU' 108. 'LUX' 109. 'LVA' 110. 'MAR' 111. 'MCO' 112. 'MDA' 113. 'MDG' 114. 'MDV' 115. 'MEX' 116. 'MHL' 117. 'MKD' 118. 'MLI' 119. 'MLT' 120. 'MMR' 121. 'MNE' 122. 'MNG' 123. 'MOZ' 124. 'MRT' 125. 'MUS' 126. 'MWI' 127. 'MYS' 128. 'NAM' 129. 'NER' 130. 'NGA' 131. 'NIC' 132. 'NLD' 133. 'NOR' 134. 'NPL' 135. 'NZL' 136. 'OMN' 137. 'PAK' 138. 'PAN' 139. 'PER' 140. 'PHL' 141. 'PLW' 142. 'PNG' 143. 'POL' 144. 'PRK' 145. 'PRT' 146. 'PRY' 147. 'PSE' 148. 'QAT' 149. 'ROU' 150. 'RUS' 151. 'RWA' 152. 'SAH' 153. 'SAU' 154. 'SDN' 155. 'SEN' 156. 'SGP' 157. 'SLB' 158. 'SLE' 159. 'SLV' 160. 'SMR' 161. 'SOL' 162. 'SOM' 163. 'SRB' 164. 'SSD' 165. 'STP' 166. 'SUR' 167. 'SVK' 168. 'SVN' 169. 'SWE' 170. 'SWZ' 171. 'SYC' 172. 'SYR' 173. 'TCD' 174. 'TGO' 175. 'THA' 176. 'TJK' 177. 'TKM' 178. 'TLS' 179. 'TON' 180. 'TTO' 181. 'TUN' 182. 'TUR' 183. 'TUV' 184. 'TWN' 185. 'TZA' 186. 'UGA' 187. 'UKR' 188. 'URY' 189. 'USA' 190. 'UZB' 191. 'VAT' 192. 'VCT' 193. 'VEN' 194. 'VNM' 195. 'VUT' 196. 'WSM' 197. 'YEM' 198. 'ZAF' 199. 'ZMB' 200. 'ZWE'

[ ]:
```
lev[c(7,4,6,2,5,3,1)]
```

1. 'ARG' 2. 'ALD' 3. 'ARE' 4. 'AGO' 5. 'AND' 6. 'ALB' 7. 'AFG'

[ ]:
```
country <- factor(islandcities$country, levels=lev[c(7,4,6,2,5,3,1)])
```

[ ]:
```
table(country)
```

```
country
ARG ALD ARE AGO AND ALB AFG
 20   1   2   7   1   1   4
```

In ordered factors, the levels have a specific order, so there are inequalities between them.

Factors can lead to unexpected results, so it's important to keep these two things in mind: 1. When a character vector becomes a column in a data frame, R automatically converts it into a factor. If you want to keep it as character, wrap it in the `I()` function. 2. Sometimes factors are treated as numeric vectors. To ensure you get the character vector, use `as.character(country)`. If you want the numeric levels (1, 2, 3, …), use `as.numeric(country)`.

1.6 7.6 Ordered Factors

In ordered factors, it's the levels that have a specific order. You can create an ordered factor, or convert a regular factor into an ordered one, using the `ordered()` function. Ordered factors are used when the levels represent positions on a scale, like small, medium, large.

[ ]:
```
stress.level<-rep(c("low","medium","high"),2)
```

[ ]:
```
ordf.stress<-ordered(stress.level, levels=c("low","medium","high"))
```

[ ]:
```
ordf.stress
```

1. low 2. medium 3. high 4. low 5. medium 6. high

Levels: 1. 'low' 2. 'medium' 3. 'high'

[ ]:
```
ordf.stress<"medium"
```

1. TRUE 2. FALSE 3. FALSE 4. TRUE 5. FALSE 6. FALSE

[ ]:
```
ordf.stress>="medium"
```

1. FALSE 2. TRUE 3. TRUE 4. FALSE 5. TRUE 6. TRUE

Ordered factors inherit properties from regular factors, and they have an additional attribute for ordering. When you check the class of an object, you'll see both its own class and any classes it inherits from.

[ ]:
```
class(ordf.stress)
```

1. 'ordered' 2. 'factor'

1.7 7.7 Lists

Lists allow you to gather various R objects into one container. This could include vectors, matrices, functions, or any other type of data. They can be a mix of different types of objects. As an example, we'll look at the list created by R when you run a linear model (lm) calculation, like the elastic.lm object mentioned in sections 1.1.4 and 2.1.4.

[77]:
```
elasticband <- data.frame(stretch=c(46,54,48,50,44,42,52),
 distance=c(148,182,173,166,109,141,166))
# it is readily verified that elastic.lm consists of a variety of diffrent␣
↪kinds of objects,
# stored as alist. You can get the names of these objects by typing in
elastic.lm <- lm(distance~stretch, data=elasticband)
names(elastic.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'

[79]:
```
# The first list element is:
elastic.lm$coefficients
```

(Intercept)	-63.5714285714285	**stretch**	4.55357142857143

[80]:
```
#Alternative ways to extract this first list element are:
elastic.lm[["coefficients"]]
elastic.lm[[1]]
```

(Intercept) -63.5714285714285 **stretch** 4.55357142857143

(Intercept) -63.5714285714285 **stretch** 4.55357142857143

To get the sublist containing only the vector `elastic.lm$coefficients`, you can use either `elastic.lm["coefficients"]` or `elastic.lm[1]`. These commands give slightly different outputs but essentially refer to the same sublist. The output will display information preceded by `$coefficients`, indicating the list element named `coefficients`.

[81]:
```
#The second list element is a vector of length 7
options(digits=3)
elastic.lm$residuals
```

1 2.10714285714287 **2** -0.321428571428572 **3** 18 **4** 1.89285714285714 **5** -27.7857142857143 **6** 13.3214285714286 **7** -7.21428571428571

[82]:
```
# The tenth list element documents the function call:
elastic.lm$call
```

```
lm(formula = distance ~ stretch, data = elasticband)
```

[83]:
```
mode(elastic.lm$call)
```

'call'

1.8 7.8 Matrices and Arrays

Matrices are simpler than data frames because they contain elements of the same type throughout, such as all numbers or all characters. This makes them useful for mathematical operations that data frames can't handle. Matrices are especially important for users interested in creating new regression or multivariate methods. Matrices can also have more than two dimensions, in which case they are called arrays. It's important to remember that matrices are stored column by column.

[89]:
```
xx <- matrix(1:6,ncol=3) # Equivalently, enter matrix(1:6,nrow=2)
xx
```

A matrix: 2 × 3 of type int

1	3	5
2	4	6

[90]:
```
#if xx is any matrix , the assignment
x <- as.vector(xx)
#places columns of xx, in order, into the vector x. In the example above, we
 ↪get back the elements 1, 2, . . . , 6.
```

[92]:
```
# Matrices have the attribute "dimension".
dim(xx)
```

1. 2 2. 3

[93]:
```r
#in fact a matrix is avector whose dimension attribute has length 2.
x34 <- matrix(1:12, ncol=4)
x34
```

A matrix: 3 × 4 of type int

1	4	7	10
2	5	8	11
3	6	9	12

[94]:
```r
# the extraction of a column or rows or submatrix
x34[2:3, c(1,4)]     # extract rows 2 & 3 & columns 1 & 4
```

A matrix: 2 × 2 of type int

2	11
3	12

[95]:
```r
x34[2,]     #extract the second row
```

1. 2 2. 5 3. 8 4. 11

[96]:
```r
x34[-2,]    #extract all rows except the second
```

A matrix: 2 × 4 of type int

1	4	7	10
3	6	9	12

[97]:
```r
x34[-2,-3]   # extract the matrix obtained by omitting row 2 and col 3
```

A matrix: 2 × 3 of type int

1	4	10
3	6	12

The dimnames() function helps manage row and column names in matrices. It returns a list containing row names as the first element and column names as the second. This concept extends straightforwardly to arrays, which we'll cover next.

7.8.1 Arrays

Extending beyond matrices (which have two dimensions) leads to arrays, which can have more than two dimensions. A matrix is essentially a two-dimensional array. For example, if we have a numeric vector with 24 elements, we can organize them into an array to maintain their order.

[98]:
```r
x <- 1:24
```

[100]:
```r
dim(x) <- c(2,12)   # turn this into 2 * 12 matrix.
x
```

A matrix: 2 × 12 of type int

1	3	5	7	9	11	13	15	17	19	21	23
2	4	6	8	10	12	14	16	18	20	22	24

[101]:
```r
dim(x) <- c(3,4,2)
x
```

1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 18. 18 19. 19 20. 20 21. 21 22. 22 23. 23 24. 24

7.8.2 Conversion of Numeric Data frames into Matrices

You can perform certain operations on matrices that you can't do with data frames. If you need to convert between the two, you can use the as.matrix() function.

1.9 7.9 Exercises

1. Generate the numbers 101, 102, …, 112, and store the result in the vector x.

[103]:
```
x <- 101:112
x
```

1. 101 2. 102 3. 103 4. 104 5. 105 6. 106 7. 107 8. 108 9. 109 10. 110 11. 111 12. 112

2. Generate four repeats of the sequence of numbers (4, 6, 3)

[108]:
```
x <- rep(c(4,6,3),4)
x
```

1. 4 2. 6 3. 3 4. 4 5. 6 6. 3 7. 4 8. 6 9. 3 10. 4 11. 6 12. 3

3. Generate the sequence consisting of eight 4s, then seven 6s, and finally nine 3s. Store the numbers obtained , in order, in the columns of a 6 by 4 matrix.

[109]:
```
# Create the sequence
sequence <- c(rep(4, 8), rep(6, 7), rep(3, 9))

# Create a 6 by 4 matrix and fill it with the sequence
matrix_sequence <- matrix(sequence, nrow = 6, ncol = 4, byrow = TRUE)
matrix_sequence
```

A matrix: 6×4 of type dbl

4	4	4	4
4	4	4	4
6	6	6	6
6	6	6	3
3	3	3	3
3	3	3	3

4. Create a vector consisting of one 1, then two 2's, three 3's, etc., and ending with nine 9's.

[110]:
```
# Generate the vector
sequence <- unlist(sapply(1:9, function(x) rep(x, times = x)))

# Print the vector
sequence
```

1. 1 2. 2 3. 2 4. 3 5. 3 6. 3 7. 4 8. 4 9. 4 10. 4 11. 5 12. 5 13. 5 14. 5 15. 5 16. 6 17. 6 18. 6 19. 6 20. 6 21. 6 22. 7 23. 7 24. 7 25. 7 26. 7 27. 7 28. 7 29. 8 30. 8 31. 8 32. 8 33. 8 34. 8 35. 8 36. 8 37. 9 38. 9 39. 9 40. 9 41. 9 42. 9 43. 9 44. 9 45. 9

5. For each of the following calculations, what you would expect? Check to see if you were right!

a)

```
 answer <- c(2, 7, 1, 5, 12, 3, 4)
for (j in 2:length(answer)){ answer[j] <- max(answer[j],answer[j-1])}
```

b)

```
answer <- c(2, 7, 1, 5, 12, 3, 4)
for (j in 2:length(answer)){ answer[j] <- sum(answer[j],answer[j-1])}
```

[111]:
```
#(1)# Given vector
answer <- c(2, 7, 1, 5, 12, 3, 4)

# Perform the calculation
for (j in 2:length(answer)) {
  answer[j] <- max(answer[j], answer[j-1])
}

# Print the result
answer
```

1. 2 2. 7 3. 7 4. 7 5. 12 6. 12 7. 12

[112]:
```
#(2)# Given vector
answer <- c(2, 7, 1, 5, 12, 3, 4)

# Perform the calculation
for (j in 2:length(answer)) {
  answer[j] <- sum(answer[j], answer[j-1])
}

# Print the result
answer
```

1. 2 2. 9 3. 10 4. 15 5. 27 6. 30 7. 34

6. In the built-in data frame airquality (datasets package):

(a) Determine, for each of the columns of the data frame airquality (datasets package), the median, mean, upper and lower quartiles, and range;

(b) Extract the row or rows for which Ozone has its maximum value;

(c) extract the vector of values of Wind for values of Ozone that are above the upper quartile.

[114]:
```
# Load the dataset
airquality  <- read.csv("/content/airquality.csv")

# (a) Summary statistics for each column
summary_stats <- function(x) {
```

```
  med <- median(x, na.rm = TRUE)
  mean_val <- mean(x, na.rm = TRUE)
  q1 <- quantile(x, 0.25, na.rm = TRUE)
  q3 <- quantile(x, 0.75, na.rm = TRUE)
  range_val <- range(x, na.rm = TRUE)

  return(c(median = med, mean = mean_val, Q1 = q1, Q3 = q3, Range = range_val))
}

# Apply the summary function to each column
summary_df <- sapply(airquality, summary_stats)

# Print the summary dataframe
print(summary_df)
```

```
       rownames Ozone Solar.R  Wind Temp Month  Day
median       77  31.5     205  9.70 79.0  7.00 16.0
mean         77  42.1     186  9.96 77.9  6.99 15.8
Q1.25%       39  18.0     116  7.40 72.0  6.00  8.0
Q3.75%      115  63.2     259 11.50 85.0  8.00 23.0
Range1        1   1.0       7  1.70 56.0  5.00  1.0
Range2      153 168.0     334 20.70 97.0  9.00 31.0
```

[115]:
```
# (b) Row with maximum Ozone value
max_ozone_row <- airquality[which.max(airquality$Ozone), ]
print(max_ozone_row)
```

```
    rownames Ozone Solar.R Wind Temp Month Day
117      117   168     238  3.4   81     8  25
```

[116]:
```
# (c) Values of Wind for Ozone > upper quartile
upper_quartile <- quantile(airquality$Ozone, 0.75, na.rm = TRUE)
wind_above_upper_quartile <- airquality$Wind[airquality$Ozone > upper_quartile]
print(wind_above_upper_quartile)
```

```
 [1]   NA   NA   NA   NA   NA  5.7   NA   NA   NA   NA   NA   NA   NA 13.8   NA
[16]   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA  4.1   NA
[31]  4.6  5.1  6.3  5.7  7.4   NA   NA  5.1   NA   NA  8.6  8.0  7.4  7.4  6.9
[46]  4.6  4.0 10.3  8.0   NA   NA  9.7   NA   NA  3.4  8.0   NA  9.7  2.3  6.3
[61]  6.3  6.9  5.1  2.8  4.6   NA
```

7. Refer to the Eurasian snow data that is given in Exercise 1.6 .

Find the mean of the snow cover

(a) for the oddnumbered years and

(b) for the even-numbered years.

[117]:
```r
# Eurasian snow dataset (assuming it's already loaded)
years <- 1970:1979
snow_cover <- c(6.5, 12.0, 14.9, 10.0, 10.7, 7.9, 21.9, 12.5, 14.5, 9.2)
# Combine years and snow_cover into a single data vector
snow_data <- data.frame(year = years, snow_cover = snow_cover)
snow_data
# Extracting odd-numbered and even-numbered years
odd_years <- snow_data$snow_cover[snow_data$year %% 2 != 0]
even_years <- snow_data$snow_cover[snow_data$year %% 2 == 0]

# Calculating mean snow cover for odd-numbered and even-numbered years
mean_snow_cover_odd <- mean(odd_years, na.rm = TRUE)
mean_snow_cover_even <- mean(even_years, na.rm = TRUE)

# Printing the results
cat("Mean snow cover for odd-numbered years:", mean_snow_cover_odd, "\n")
cat("Mean snow cover for even-numbered years:", mean_snow_cover_even, "\n")
```

A data.frame: 10×2

year	snow_cover
<int>	<dbl>
1970	6.5
1971	12.0
1972	14.9
1973	10.0
1974	10.7
1975	7.9
1976	21.9
1977	12.5
1978	14.5
1979	9.2

```
Mean snow cover for odd-numbered years: 10.3
Mean snow cover for even-numbered years: 13.7
```

8. Determine which columns of the data frame Cars93 (MASS package) are factors. For each of these factor columns, print out the levels vector. Which of these are ordered factors?

[118]:
```r
# Load the MASS package
library(MASS)

# Load the Cars93 dataset
data(Cars93)

# Identify factor columns
factor_columns <- sapply(Cars93, is.factor)

# Print factor columns and their levels
for (column_name in names(Cars93[factor_columns])) {
```

```r
  cat("Column:", column_name, "\n")
  cat("Levels:", levels(Cars93[[column_name]]), "\n")

  # Check if the factor is ordered
  if (is.ordered(Cars93[[column_name]])) {
    cat("Ordered factor:", column_name, "\n")
  }

  cat("\n")
}
```

```
Column: Manufacturer
Levels: Acura Audi BMW Buick Cadillac Chevrolet Chrylser Chrysler Dodge Eagle
Ford Geo Honda Hyundai Infiniti Lexus Lincoln Mazda Mercedes-Benz Mercury
Mitsubishi Nissan Oldsmobile Plymouth Pontiac Saab Saturn Subaru Suzuki Toyota
Volkswagen Volvo

Column: Model
Levels: 100 190E 240 300E 323 535i 626 850 90 900 Accord Achieva Aerostar Altima
Astro Bonneville Camaro Camry Capri Caprice Caravan Cavalier Celica Century
Civic Colt Concorde Continental Corrado Corsica Corvette Cougar Crown_Victoria
Cutlass_Ciera DeVille Diamante Dynasty ES300 Eighty-Eight Elantra Escort Eurovan
Excel Festiva Firebird Fox Grand_Prix Imperial Integra Justy Laser LeBaron
LeMans LeSabre Legacy Legend Loyale Lumina Lumina_APV MPV Maxima Metro Mirage
Mustang Passat Prelude Previa Probe Protege Q45 Quest RX-7 Riviera Roadmaster
SC300 SL Scoupe Sentra Seville Shadow Silhouette Sonata Spirit Stealth Storm
Summit Sunbird Swift Taurus Tempo Tercel Town_Car Vision

Column: Type
Levels: Compact Large Midsize Small Sporty Van

Column: AirBags
Levels: Driver & Passenger Driver only None

Column: DriveTrain
Levels: 4WD Front Rear

Column: Cylinders
Levels: 3 4 5 6 8 rotary

Column: Man.trans.avail
Levels: No Yes

Column: Origin
Levels: USA non-USA

Column: Make
```

```
Levels: Acura Integra Acura Legend Audi 100 Audi 90 BMW 535i Buick Century Buick
LeSabre Buick Riviera Buick Roadmaster Cadillac DeVille Cadillac Seville
Chevrolet Astro Chevrolet Camaro Chevrolet Caprice Chevrolet Cavalier Chevrolet
Corsica Chevrolet Corvette Chevrolet Lumina Chevrolet Lumina_APV Chrylser
Concorde Chrysler Imperial Chrysler LeBaron Dodge Caravan Dodge Colt Dodge
Dynasty Dodge Shadow Dodge Spirit Dodge Stealth Eagle Summit Eagle Vision Ford
Aerostar Ford Crown_Victoria Ford Escort Ford Festiva Ford Mustang Ford Probe
Ford Taurus Ford Tempo Geo Metro Geo Storm Honda Accord Honda Civic Honda
Prelude Hyundai Elantra Hyundai Excel Hyundai Scoupe Hyundai Sonata Infiniti Q45
Lexus ES300 Lexus SC300 Lincoln Continental Lincoln Town_Car Mazda 323 Mazda 626
Mazda MPV Mazda Protege Mazda RX-7 Mercedes-Benz 190E Mercedes-Benz 300E Mercury
Capri Mercury Cougar Mitsubishi Diamante Mitsubishi Mirage Nissan Altima Nissan
Maxima Nissan Quest Nissan Sentra Oldsmobile Achieva Oldsmobile Cutlass_Ciera
Oldsmobile Eighty-Eight Oldsmobile Silhouette Plymouth Laser Pontiac Bonneville
Pontiac Firebird Pontiac Grand_Prix Pontiac LeMans Pontiac Sunbird Saab 900
Saturn SL Subaru Justy Subaru Legacy Subaru Loyale Suzuki Swift Toyota Camry
Toyota Celica Toyota Previa Toyota Tercel Volkswagen Corrado Volkswagen Eurovan
Volkswagen Fox Volkswagen Passat Volvo 240 Volvo 850
```

9. Use summary() to get information about data in the data frames attitude (both in the datasets package), and cpus (MASS package). Write brief notes, for each of these data sets, on what this reveals.

[119]:
```r
# Load the datasets package
library(datasets)

# Load the MASS package
library(MASS)

# Summary of the attitude dataset
cat("Summary of attitude dataset:\n")
summary(attitude)
cat("\n")

# Summary of the cpus dataset
cat("Summary of cpus dataset:\n")
summary(cpus)
cat("\n")
```

Summary of attitude dataset:

```
    rating        complaints      privileges       learning         raises
 Min.   :40.0   Min.   :37.0   Min.   :30.0   Min.   :34.0   Min.   :43.0
 1st Qu.:58.8   1st Qu.:58.5   1st Qu.:45.0   1st Qu.:47.0   1st Qu.:58.2
 Median :65.5   Median :65.0   Median :51.5   Median :56.5   Median :63.5
 Mean   :64.6   Mean   :66.6   Mean   :53.1   Mean   :56.4   Mean   :64.6
 3rd Qu.:71.8   3rd Qu.:77.0   3rd Qu.:62.5   3rd Qu.:66.8   3rd Qu.:71.0
 Max.   :85.0   Max.   :90.0   Max.   :83.0   Max.   :75.0   Max.   :88.0
```

```
   critical        advance
 Min.   :49.0   Min.   :25.0
 1st Qu.:69.2   1st Qu.:35.0
 Median :77.5   Median :41.0
 Mean   :74.8   Mean   :42.9
 3rd Qu.:80.0   3rd Qu.:47.8
 Max.   :92.0   Max.   :72.0
```

```
Summary of cpus dataset:

             name          syct             mmin             mmax
 ADVISOR 32/60 :  1   Min.   :  17   Min.   :   64   Min.   :   64
 AMDAHL 470/7A :  1   1st Qu.:  50   1st Qu.:  768   1st Qu.: 4000
 AMDAHL 470V/7 :  1   Median : 110   Median : 2000   Median : 8000
 AMDAHL 470V/7B:  1   Mean   : 204   Mean   : 2868   Mean   :11796
 AMDAHL 470V/7C:  1   3rd Qu.: 225   3rd Qu.: 4000   3rd Qu.:16000
 AMDAHL 470V/8 :  1   Max.   :1500   Max.   :32000   Max.   :64000
 (Other)       :203
      cach            chmin            chmax             perf           estperf
 Min.   :  0.0   Min.   : 0.0   Min.   :  0.0   Min.   :   6   Min.   :  15
 1st Qu.:  0.0   1st Qu.: 1.0   1st Qu.:  5.0   1st Qu.:  27   1st Qu.:  28
 Median :  8.0   Median : 2.0   Median :  8.0   Median :  50   Median :  45
 Mean   : 25.2   Mean   : 4.7   Mean   : 18.3   Mean   : 106   Mean   :  99
 3rd Qu.: 32.0   3rd Qu.: 6.0   3rd Qu.: 24.0   3rd Qu.: 113   3rd Qu.: 101
 Max.   :256.0   Max.   :52.0   Max.   :176.0   Max.   :1150   Max.   :1238
```

10. From the data frame mtcars (MASS package) extract a data frame mtcars6 that holds only the information for cars with 6 cylinders.

[120]:
```
# Load the MASS package
library(MASS)

# Extract cars with 6 cylinders
mtcars6 <- mtcars[mtcars$cyl == 6, ]

# View the first few rows of mtcars6
head(mtcars6)
```

A data.frame: 6 × 11

	mpg <dbl>	cyl <dbl>	disp <dbl>	hp <dbl>	drat <dbl>	wt <dbl>	qsec <dbl>	vs <dbl>
Mazda RX4	21.0	6	160	110	3.90	2.62	16.5	0
Mazda RX4 Wag	21.0	6	160	110	3.90	2.88	17.0	0
Hornet 4 Drive	21.4	6	258	110	3.08	3.21	19.4	1
Valiant	18.1	6	225	105	2.76	3.46	20.2	1
Merc 280	19.2	6	168	123	3.92	3.44	18.3	1
Merc 280C	17.8	6	168	123	3.92	3.44	18.9	1

11. From the data frame Cars93 (MASS package), extract a data frame which holds only information for small and sporty cars.

```
# This is formatted as code
```

[121]:
```
# Load the MASS package
library(MASS)

# Extract small and sporty cars
small_sporty_cars <- subset(Cars93, Type %in% c("Small", "Sporty"))

# View the first few rows of the extracted data frame
head(small_sporty_cars)
```

A data.frame: 6 × 27

	Manufacturer <fct>	Model <fct>	Type <fct>	Min.Price <dbl>	Price <dbl>	Max.Price <dbl>	MPG.city <int>	MP <i
1	Acura	Integra	Small	12.9	15.9	18.8	25	31
14	Chevrolet	Camaro	Sporty	13.4	15.1	16.8	19	28
19	Chevrolet	Corvette	Sporty	34.6	38.0	41.5	17	25
23	Dodge	Colt	Small	7.9	9.2	10.6	29	33
24	Dodge	Shadow	Small	8.4	11.3	14.2	23	29
28	Dodge	Stealth	Sporty	18.5	25.8	33.1	18	24