Creating New Variables

Contents

1 Creating New Variables

1.1 Why create new variables ?

Often the raw data won't have a value you are looking for You will need to transform the data to get the values you would like Usually you will add those values to the data frames you are working with Common variables to create - Missingness indicators - "Cutting up" quantitative variables - Applying transforms

1.2 Creating sequences

Sometimes you need an index for your data set

```
s1 <- seq(1,10,by=2)
s1
```

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

```
s2 <- seq(1,10,length=3)
s2
```

```
## [1]  1.0  5.5 10.0
```

```
x <- c(1,3,8,25,100);
seq(along = x)
```

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

When you use the along argument in the seq() function, it generates a sequence from 1 to the length of the object you have specified.

1.3 Subsetting variables

```
rest <- read.csv("./data/rest.csv")
```

```
rest$nearme <- rest$nghbrhd %in% c("Roland Park","Homeland")
```

```
table(rest$nearme)
```

```
##
## FALSE  TRUE
##  1314    13
```

1.4 Creating binary variables

```
rest$zipWrong <- ifelse(rest$zipcode < 0, TRUE, FALSE)
table(rest$zipWrong, rest$zipcode < 0)
```

```
##
##         FALSE
##   FALSE  1327
```

1.5 Creatingc ategoricavl ariables

```
rest$zipcode <- as.numeric(rest$zipcode)
```

```
## Warning: NAs introduced by coercion
```

```
rest$zipGroups <- cut(rest$zipcode, breaks = quantile(rest$zipcode))
```

```
## Error in quantile.default(rest$zipcode): missing values and NaN's not allowed if 'na.rm' is FALSE
```

```
table(rest$zipGroups)
```

```
## < table of extent 0 >
```

1.6 Easier cutting

```
library(Hmisc)
```

```
##
## Attaching package: 'Hmisc'
```

```
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
rest$zipGroups <- cut2(rest$zipcode, g=4)
```

1.7 Creating factor variables

```
rest$zcf <- factor(rest$zipcode)
rest$zcf[1:10]
```

```
##  [1] 21206 21231 21224 21211 21223 21218 21205 21211 21205 21231
## 31 Levels: 21201 21202 21205 21206 21207 21208 21209 21210 21211 ... 21287
```

```
class(rest$zcf)
```

```
## [1] "factor"
```

1.8 Levels of factor variables

```
yesno <- sample(c("yes", "no"), size = 10, replace = TRUE)
yesnofac <- factor(yesno, levels = c("yes","no"))
```

```
yesno
```

```
##  [1] "yes" "no"  "yes" "yes" "yes" "yes" "no"  "yes" "yes" "no"
```

```
yesnofac
```

```
##  [1] yes no  yes yes yes yes no  yes yes no
## Levels: yes no
```

```
relevel(yesnofac, ref = "yes")
```

```
##  [1] yes no  yes yes yes yes no  yes yes no
## Levels: yes no
```

```
as.numeric(yesnofac)
```

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

1.9 Using the mutate function

```
library(Hmisc)
library(plyr)
```

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:Hmisc':
##
##     is.discrete, summarize
```

```
rest2 <- mutate(rest, zipGroups=cut2(zipcode, g=4))
```

```
table(rest2$zipGroups)
```

```
##
## [21201,21205) [21205,21220) [21220,21227) [21227,21287]
##           337           375           300           314
```

The mutate() function is a part of the dplyr package (and also available in plyr) in R. It's used to create new variables or modify existing ones in a data frame.

```
str(mutate)
```

```
## function (.data, ...)
```

1.10 Common transforms

`abs` (x) absolute value `sqrt` (x) square root `ceiling` (x) ceiling(3.475) is 4 `floor` (x) floor(3.475) is 3 `round` (x, digi ts=n) roun(3.475,digits=2) is 3.48 `signif` (x, digi ts=n) signif(3.475,digits=2) is 3.5 `cos(x)`, sin(x) etc. `log ( x)` natural logarithm `log2 ( x)`, logl o (x) other common logs `exp ( x)` exponentiating x