Data Wrangling, etc.

Kylie Ariel Bemis

10/9/2018

Data Wrangling, etc.

- Strings with `stringr`
- Factors with `forcats`
- Dates and times with `lubridate`

Strings

```
string1 <- "This is a string"
string2 <- "Include a 'quote' with a single quote"
string3 <- 'Or include a "quote" the other way around'
string4 <- "Here is a string\nwith a newline"
```

Printing strings

```
string1
```

```
## [1] "This is a string"
```

```
string2
```

```
## [1] "Include a 'quote' with a single quote"
```

```
string3
```

```
## [1] "Or include a \"quote\" the other way around"
```

```
string4
```

```
## [1] "Here is a string\nwith a newline"
```

Printing strings (cont'd)

```
cat(string1)
```

```
## This is a string
```

```
cat(string2)
```

```
## Include a 'quote' with a single quote
```

```
cat(string3)
```

```
## Or include a "quote" the other way around
```

```
cat(string4)
```

```
## Here is a string
## with a newline
```

Character vectors

```
c("this", "is", "a", "vector", "of", "strings")
```

```
## [1] "this"    "is"      "a"       "vector"  "of"      "string
```

Strings with `stringr`

The `string` package is a non-default part of the tidyverse. You only need to load it when you need to process strings.

The `stringr` package provides convenience functions for string processing. Most of its functionality can be accomplished in base R as well, but it provides a more consistent (and sometimes faster) interface, following typical tidyverse conventions (e.g., the data is always the first argument).

All `stringr` functions are prefixed with `str_`.

```
library(stringr)
```

String length

```
string <- c("abc", "123", "hello world", NA, "\u03c0\u03c3")
```

Get the length of the character vector.

```
length(string)
```

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

Get the length of the strings in the character vector.

```
str_length(string)
```

```
## [1]  3  3 11 NA  2
```

String length (cont'd)

Get the character length of the strings in the character vector.

```
nchar(string)
```

```
## [1]  3  3 11 NA  2
```

Get the byte length of the strings in the character vector.

```
nchar(string, type="bytes")
```

```
## [1]  3  3 11 NA  4
```

String length (cont'd)

U200B is the "zero-width space" unicode character.

```
nchar("\u200b")
```

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

```
nchar("\u200b", type="bytes")
```

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

```
nchar("\u200b", type="width")
```

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

Concatenating strings

```
str_c("x", "y")
```

```
## [1] "xy"
```

```
str_c("x", "y", sep=", ")
```

```
## [1] "x, y"
```

```
str_c(c("x", "y"), c("1", "2"), sep=" = ")
```

```
## [1] "x = 1" "y = 2"
```

```
str_c(c("x", "y"), c("1", "2"), sep=" = ", collapse=", ")
```

```
## [1] "x = 1, y = 2"
```

Concatenating strings (cont'd)

```
paste0("x", "y")
```

```
## [1] "xy"
```

```
paste("x", "y", sep=", ")
```

```
## [1] "x, y"
```

```
paste(c("x", "y"), c("1", "2"), sep=" = ")
```

```
## [1] "x = 1" "y = 2"
```

```
paste(c("x", "y"), c("1", "2"), sep=" = ", collapse=", ")
```

```
## [1] "x = 1, y = 2"
```

Substrings

```
string <- c("CAT_F", "CAT_F", "DOG_M")
str_sub(string, 1, 3)
```

```
## [1] "CAT" "CAT" "DOG"
```

```
str_sub(string, -2, -1)
```

```
## [1] "_F" "_F" "_M"
```

Locating positions and splitting strings

```
string <- c("CAT_F", "CAT_F", "DOG_M")
str_locate(string, "_")
```

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

```
str_split(string, "_")
```

```
## [[1]]
## [1] "CAT" "F"
##
## [[2]]
## [1] "CAT" "F"
##
## [[3]]
## [1] "DOG" "M"
```

Detecting patterns

```
string <- c("hello", "hi", "hey", "well met")
str_detect(string, "h")
```

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

Detecting patterns

```
library(tidyverse)
words_df <- tibble(i=seq_along(words), words=words)
```

```
words_df
```

```
## # A tibble: 980 x 2
##       i words
##   <int> <chr>
##  1     1 a
##  2     2 able
##  3     3 about
##  4     4 absolute
##  5     5 accept
##  6     6 account
##  7     7 achieve
##  8     8 across
##  9     9 act
## 10    10 active
## # ... with 970 more rows
```

Detecting patterns (cont'd)

Find words that start with "b".

```
filter(words_df, str_detect(words, "^b"))
```

```
## # A tibble: 58 x 2
##         i words
##     <int> <chr>
##  1     66 baby
##  2     67 back
##  3     68 bad
##  4     69 bag
##  5     70 balance
##  6     71 ball
##  7     72 bank
##  8     73 bar
##  9     74 base
## 10     75 basis
## # ... with 48 more rows
```

Padding with white space

```
words_df2 <- mutate(words_df, words=str_pad(words, 30, side="bot
head(words_df2$words)
```

```
## [1] "                             a                             "
## [3] "                         about                             "
## [5] "                        accept                             "
```

```
## [1] "                             able                          "
## [3] "                         absolute                          "
## [5] "                        account                            "
```

Trimming white space

```
words_df3 <- mutate(words_df2, words=str_trim(words))
head(words_df3$words)
```

```
## [1] "a"        "able"     "about"    "absolute" "accept"   "a
```

Strings with `stringi`

The `stringr` package exposes a small subset of the functionality of the `stringi` package, which is a much more complex package for working with strings in R using efficient C++ code under-the-hood.

If `stringr` doesn't provide the functionality you need in string processing, check out the functions in the `stringi` package.

Factors with `forcats`

The `forcats` package is a non-default part of the tidyverse for working with **cat**egorical variables. It provides convenience functions for working with `factors`, and is primarily useful for robustly and conveniently changing the levels of factors.

All functions in `forcats` are prefixed with `fct_`.

```
library(forcats)
```

(Unfortunately, `forcats` provides no additional functionality for working with cats in R. For that, you want https://github.com/Gibbsdavidl/CatterPlots.)

Factors versus character vectors

As we have discussed before, character vectors and factors look similar in R, and some base R functions will silently convert strings to factors.

Internally, factors are stored as integers and character vectors are stored as strings.

It is important to know whether you are working with a factor or a character vector, and which are useful for what purposes.

- Are you representing text data or a categorical variable?
- Do you need to do any string processing?
- How many levels will it have?

Typically, text data should be represented as a character vector and categorical variables should be represented as factors.

Creating factors

```r
month_levels <- c(
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
)
factor(c("Sep", "Apr", "Jun", "Nov"), levels=month_levels)
```

```
## [1] Sep Apr Jun Nov
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

```r
factor(c("Sep", "Apr", "June", "Nov"), levels=month_levels)
```

```
## [1] Sep  Apr  <NA> Nov
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

```r
ordered(c("Sep", "Apr", "Jun", "Nov"), levels=month_levels)
```

```
## [1] Sep Apr Jun Nov
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ...
```

Creating factors (cont'd)

```
fct <- c("C", "B", "A")
factor(fct)
```

```
## [1] C B A
## Levels: A B C
```

```
as.factor(fct)
```

```
## [1] C B A
## Levels: A B C
```

```
factor(fct, levels=unique(fct))
```

```
## [1] C B A
## Levels: C B A
```

```
factor(fct) %>% fct_inorder()
```

```
## [1] C B A
## Levels: C B A
```

Releveling factors

```
fct <- factor(fct)
fct
```

```
## [1] C B A
## Levels: A B C
```

```
relevel(fct, "C")
```

```
## [1] C B A
## Levels: C A B
```

```
fct_relevel(fct, "C")
```

```
## [1] C B A
## Levels: C A B
```

```
fct_relevel(fct, c("C", "B", "A"))
```

```
## [1] C B A
## Levels: C B A
```

Recoding factors

```
recode(fct, A="X", B="Y", C="Z")
```

```
## [1] Z Y X
## Levels: X Y Z
```

```
fct_recode(fct, X="A", Y="B", Z="C")
```

```
## [1] Z Y X
## Levels: X Y Z
```

```
fct_recode(fct, AB="A", AB="B")
```

```
## [1] C  AB AB
## Levels: AB C
```

```
fct_collapse(fct, AB=c("A", "B"))
```

```
## [1] C  AB AB
## Levels: AB C
```

Dates and times with `lubridate`

Dates and times are notoriously difficult. You have to consider multiple components (year, month, day, hour, minute, second, etc.) as well as additional information like time zones. The fact that some years and months are different lengths make dates and times even more of a headache.

The `lubridate` package from the tidyverse provides many convenience functions which make working with dates and date-times much easier.

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date
```

Get current date or date-time

```
today()
```

```
## [1] "2018-10-01"
```

```
now()
```

```
## [1] "2018-10-01 20:48:07 EDT"
```

Parsing dates

```
ymd("2018-01-31")
```

```
## [1] "2018-01-31"
```

```
mdy("January 31st, 2018")
```

```
## [1] "2018-01-31"
```

```
dmy("31-Jan-2018")
```

```
## [1] "2018-01-31"
```

Parsing date-times

```
ymd_hms("2018-01-31 20:11:59")
```

```
## [1] "2018-01-31 20:11:59 UTC"
```

```
mdy_hm("01/31/2018 08:01")
```

```
## [1] "2018-01-31 08:01:00 UTC"
```

```
mdy_hm("01/31/2018 08:01", tz="EST")
```

```
## [1] "2018-01-31 08:01:00 EST"
```

Accessing date-time components

```
datetime <- ymd_hms("2018-02-09 12:34:56")
year(datetime)
```

```
## [1] 2018
```

```
month(datetime)
```

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

```
mday(datetime)
```

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

```
yday(datetime)
```

```
## [1] 40
```

```
wday(datetime)
```

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

Accessing date-time components (cont'd)

```
month(datetime, label=TRUE)
```

```
## [1] Feb
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Se
```

```
month(datetime, label=TRUE, abbr=FALSE)
```

```
## [1] February
## 12 Levels: January < February < March < April < May < June <
```

Accessing date-time components (cont'd)

```
wday(datetime, label=TRUE)
```

```
## [1] Fri
## Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

```
wday(datetime, label=TRUE, abbr=FALSE)
```

```
## [1] Friday
## 7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday <
```

Creating dates from components

```
library(nycflights13)
flights %>%
  select(year, month, day, hour, minute)
```

```
## # A tibble: 336,776 x 5
##     year month   day  hour minute
##    <int> <int> <int> <dbl>  <dbl>
##  1  2013     1     1     5     15
##  2  2013     1     1     5     29
##  3  2013     1     1     5     40
##  4  2013     1     1     5     45
##  5  2013     1     1     6      0
##  6  2013     1     1     5     58
##  7  2013     1     1     6      0
##  8  2013     1     1     6      0
##  9  2013     1     1     6      0
## 10  2013     1     1     6      0
## # ... with 336,766 more rows
```

```r
flights %>%
  select(year, month, day, hour, minute) %>%
  mutate(departure = make_datetime(year, month, day, hour, minut
```

```
## # A tibble: 336,776 x 6
##     year month   day  hour minute departure
##    <int> <int> <int> <dbl>  <dbl> <dttm>
##  1  2013     1     1     5     15 2013-01-01 05:15:00
##  2  2013     1     1     5     29 2013-01-01 05:29:00
##  3  2013     1     1     5     40 2013-01-01 05:40:00
##  4  2013     1     1     5     45 2013-01-01 05:45:00
##  5  2013     1     1     6      0 2013-01-01 06:00:00
##  6  2013     1     1     5     58 2013-01-01 05:58:00
##  7  2013     1     1     6      0 2013-01-01 06:00:00
##  8  2013     1     1     6      0 2013-01-01 06:00:00
##  9  2013     1     1     6      0 2013-01-01 06:00:00
## 10  2013     1     1     6      0 2013-01-01 06:00:00
## # ... with 336,766 more rows
```

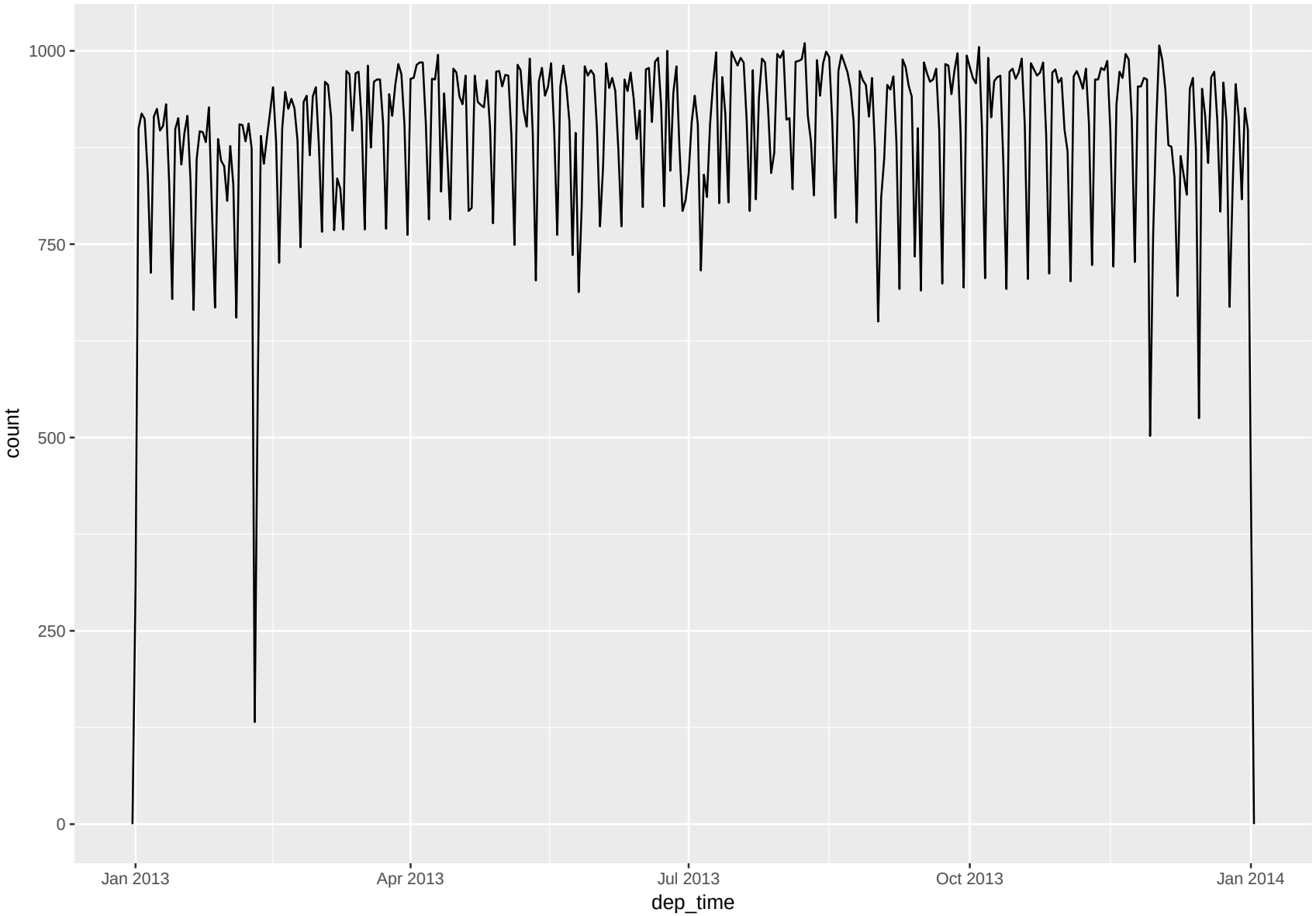
```r
make_datetime_100 <- function(year, month, day, time) {
  make_datetime(year, month, day, time %/% 100, time %% 100)
}

flights_dt <- flights %>%
  filter(!is.na(dep_time), !is.na(arr_time)) %>%
  mutate(
    dep_time = make_datetime_100(year, month, day, dep_time),
    arr_time = make_datetime_100(year, month, day, arr_time),
    sched_dep_time = make_datetime_100(year, month, day, sched_dep_time
    sched_arr_time = make_datetime_100(year, month, day, sched_arr_time
  ) %>%
  select(origin, dest, ends_with("delay"), ends_with("time"))
```

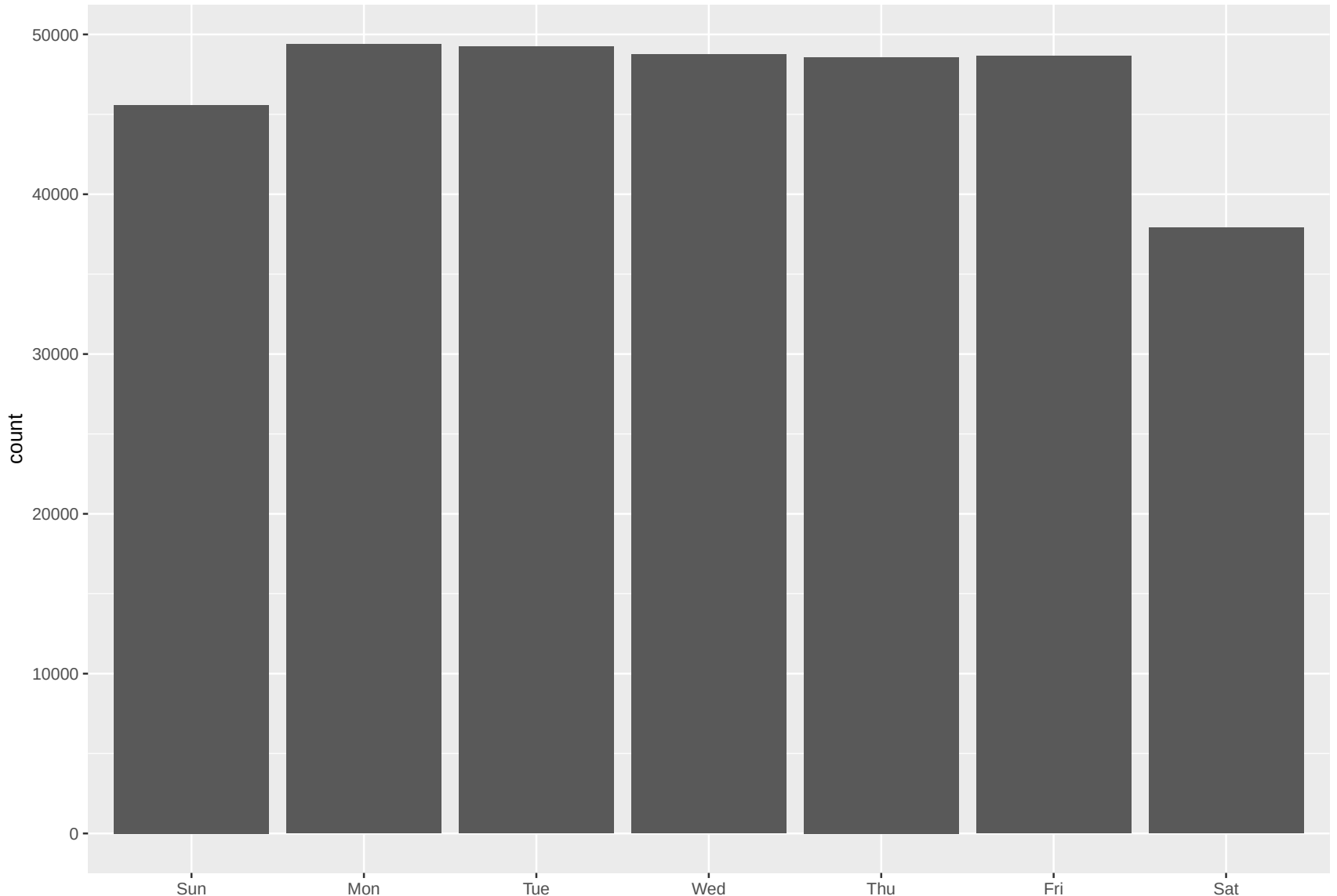
```
flights_dt
```

```
## # A tibble: 328,063 x 9
##    origin dest  dep_delay arr_delay dep_time            sched
##    <chr>  <chr>     <dbl>     <dbl> <dttm>              <dttm
##  1 EWR    IAH           2        11 2013-01-01 05:17:00 2013-
##  2 LGA    IAH           4        20 2013-01-01 05:33:00 2013-
##  3 JFK    MIA           2        33 2013-01-01 05:42:00 2013-
##  4 JFK    BQN          -1       -18 2013-01-01 05:44:00 2013-
##  5 LGA    ATL          -6       -25 2013-01-01 05:54:00 2013-
##  6 EWR    ORD          -4        12 2013-01-01 05:54:00 2013-
##  7 EWR    FLL          -5        19 2013-01-01 05:55:00 2013-
##  8 LGA    IAD          -3       -14 2013-01-01 05:57:00 2013-
##  9 JFK    MCO          -3        -8 2013-01-01 05:57:00 2013-
## 10 LGA    ORD          -2         8 2013-01-01 05:58:00 2013-
## # ... with 328,053 more rows, and 3 more variables: arr_time
## #   sched_arr_time <dttm>, air_time <dbl>
```

```r
flights_dt %>%
  ggplot(aes(dep_time)) +
  geom_freqpoly(binwidth = 24*60*60)
```



```r
flights_dt %>%
  mutate(wday = wday(dep_time, label = TRUE)) %>%
  ggplot(aes(x = wday)) +
    geom_bar()
```



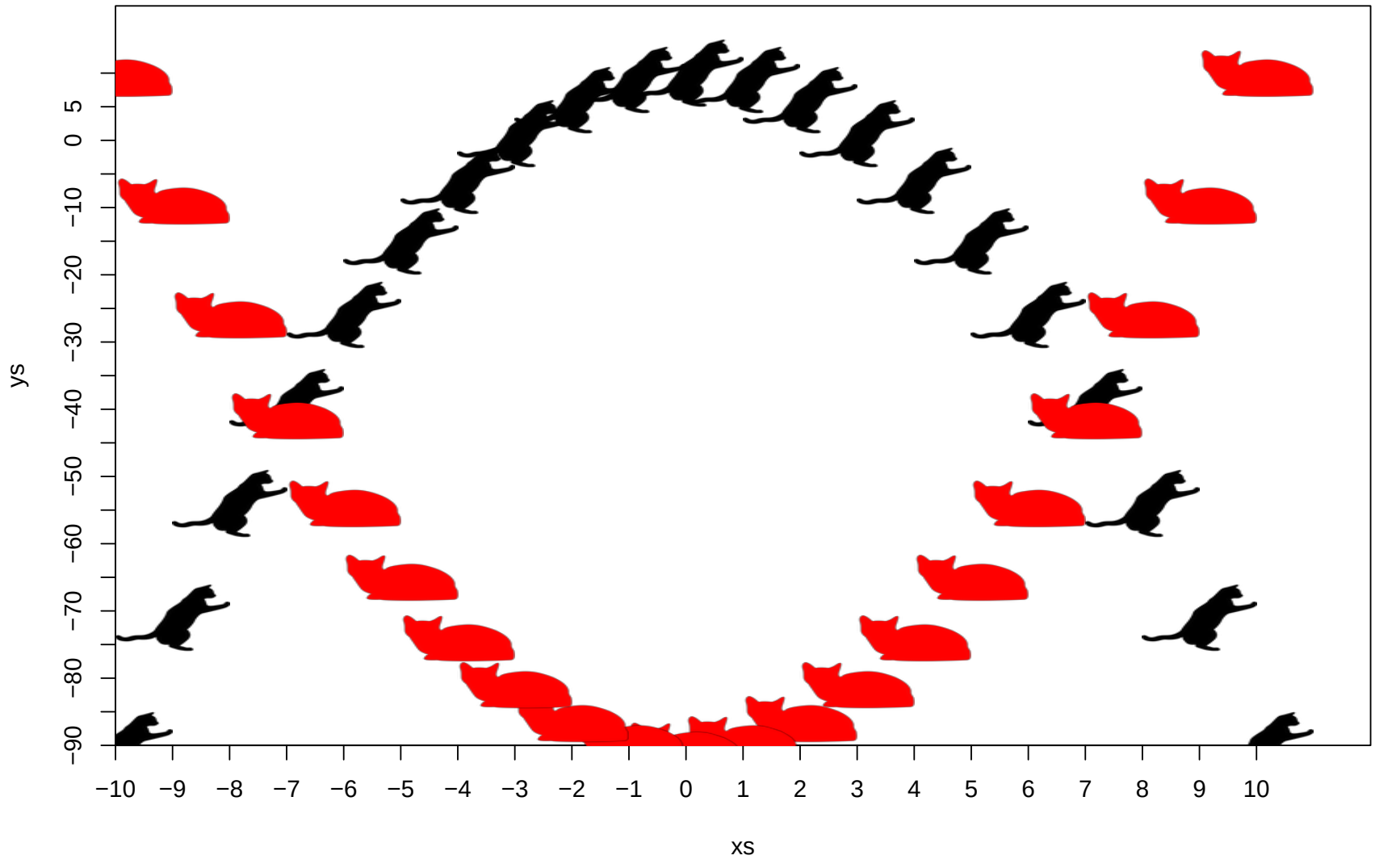
For cats

Did you ever wish you could make scatter plots with cat shaped points? Now you can!

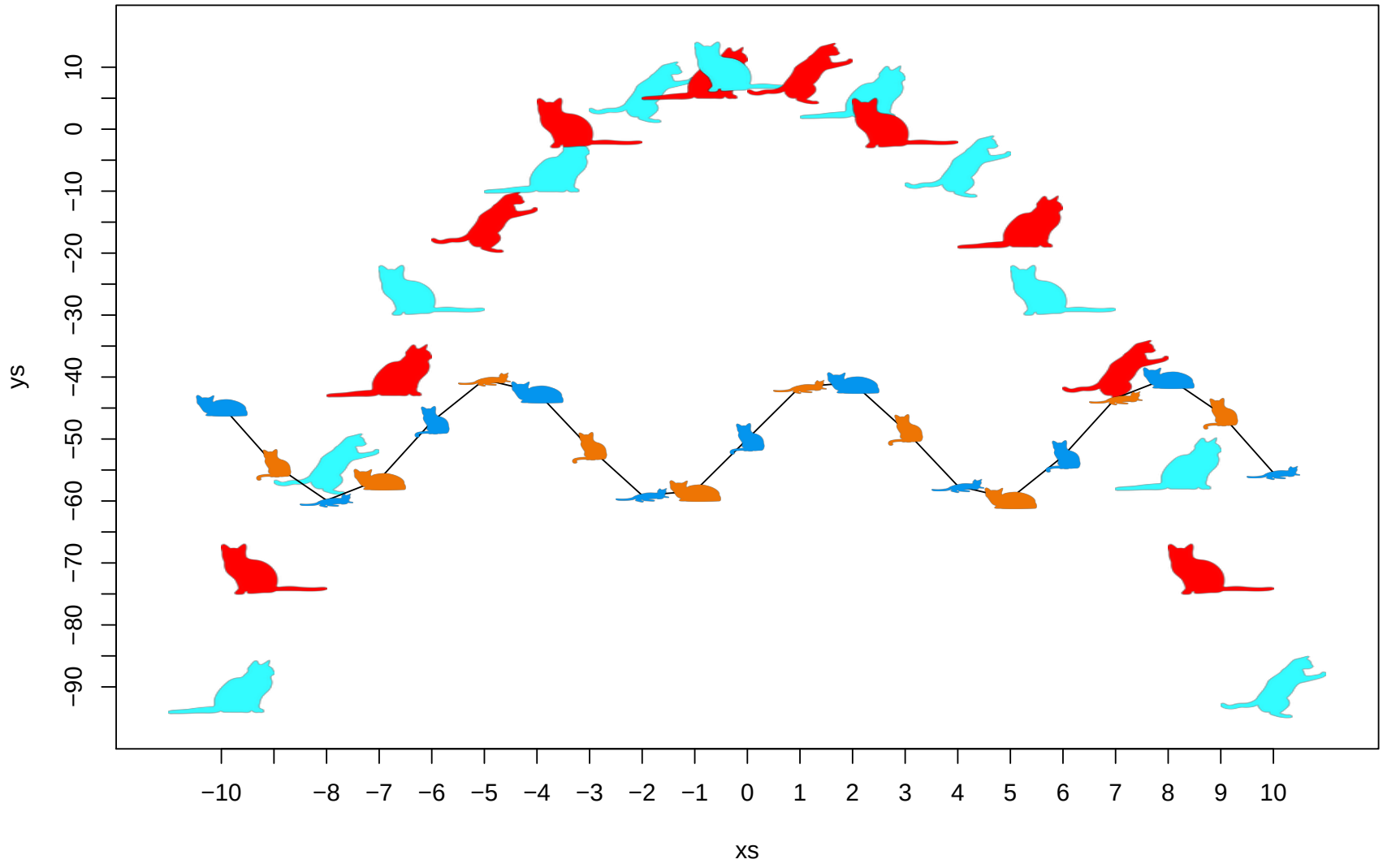
```
library(devtools)
install_github("Gibbsdavidl/CatterPlots")
```

```
library(CatterPlots)
x <- -10:10
y <- -x^2 + 10
purr <- catplot(xs=x, ys=y, cat=3, catcolor='#000000FF')
cats(purr, -x, -y, cat=4, catcolor='#FF0000')
```

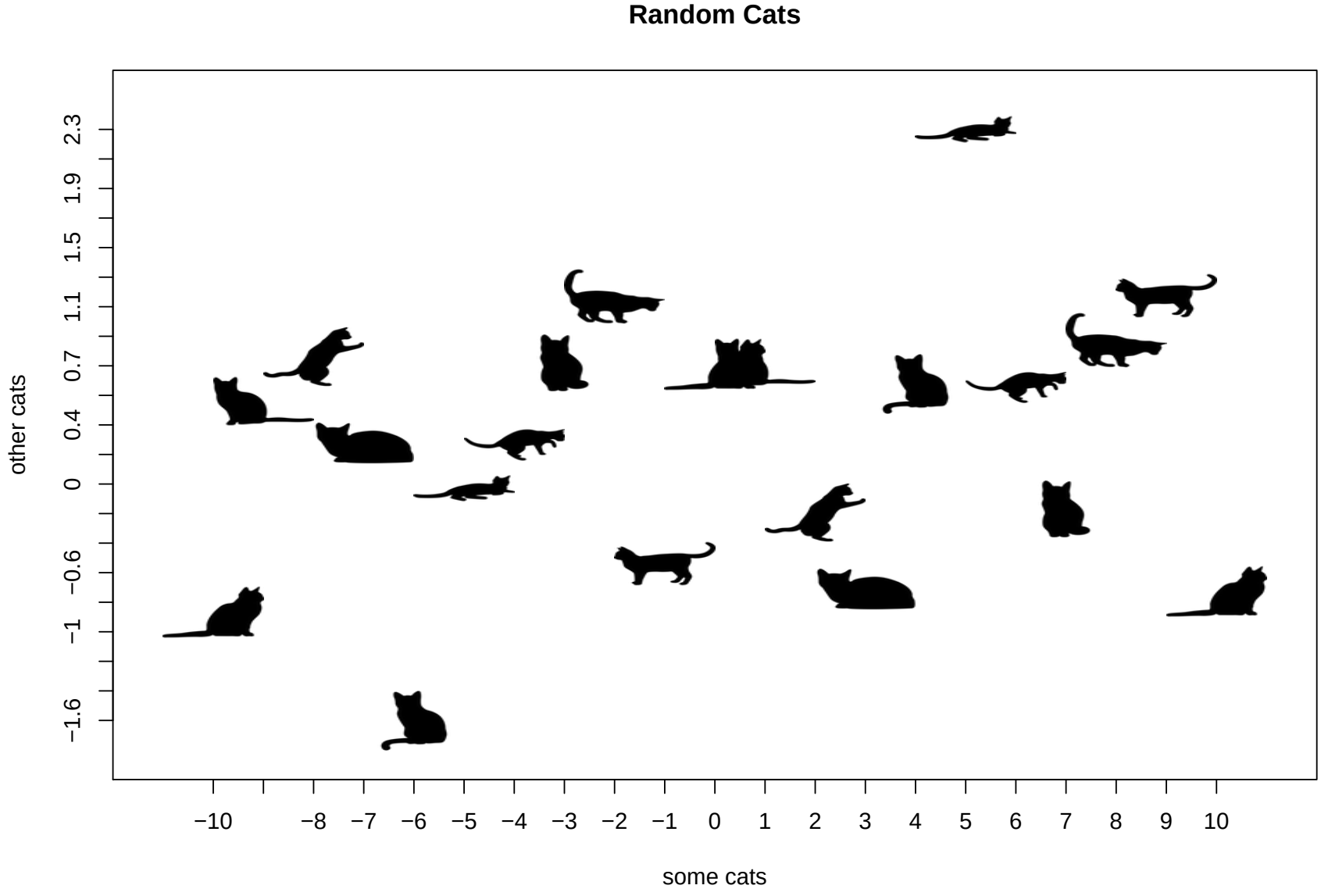
```
##
## Welcome to CatterPlots.
```



```r
# for more fun ...
meow <- multicat(xs=x, ys=y, cat=c(1,2,3),
                 catcolor=list('#33FCFF','#FF0000'),
                 canvas=c(-0.1,1.1, -0.1, 1.1))
morecats(meow, x, 10*sin(x)+40, size=0.05, cat=c(4,5,6),
         catcolor=list('#0495EE','#EE7504'), type="line")
```


ys
10
0
−10
−20
−30
−40
−50
−60
−70
−80
−90
−10
−8
−7
−6
−5
−4
−3
−2
−1
0
1
2
3
4
5
6
7
8
9
10
xs

```r
# random cats
meow <- multicat(xs=x, ys=rnorm(21),
                 cat=c(1,2,3,4,5,6,7,8,9,10),
                 canvas=c(-0.1,1.1, -0.1, 1.1),
                 xlab="some cats", ylab="other cats",
                 main="Random Cats")
```


Random Cats
other cats
2.3
1.9
1.5
1.1
0.7
0.4
0
−0.6
−1
−1.6
−10
−8
−7
−6
−5
−4
−3
−2
−1
0
1
2
3
4
5
6
7
8
9
10
some cats