*Working with* 

A University of Queensland Advanced Workshop

# Session 4: Case Studies

Bill Venables, CSIRO/Data61, Dutton Park

Rhetta Chappell, Griffith University

2–5 February, 2021

# Contents

# 1 Darwins Finches

The *DarwinsFinches* data set comes from an historical expedition to the Galapagos Islands in 1898-99. It gives various anatomical measurements for one hundred and forty-six finch specimens, classified into five species. Our task will be to show to what extent the anatomical measurements alone can separate out the species. We will use the classical discriminant function method and display the results in two dimensions only.

We will do the computations "by hand" to illustrate how things can be done, elegantly, using elementary tools.

```
find("DarwinsFinches")

[1] "package:WWRData"

str(DarwinsFinches)

'data.frame': 146 obs. of  11 variables:
 $ Id     : chr  "B01" "B02" "B03" "B04" ...
 $ Species: Factor w/ 5 levels "Geospiza fortis fortis",..: 1 1 1 1 1 1 1 1 1 1 ..
 $ BodyL  : num  115 117 118 118 120 120 125 125 127 127 ...
 $ WingL  : num  71 66 65 67 64 66 68 70 68 68.5 ...
 $ TailL  : num  41 39 38.5 38.5 34 40 48 42 49 44 ...
 $ BeakW  : num  9.7 8.3 8.5 8.7 9 8.3 9.7 11 9.3 9 ...
 $ BeakH  : num  13 11 11.3 12 11 11.5 12.5 14 12.3 14 ...
 $ LBeakL : num  9.7 8 8.5 8.5 8 8.5 9 10 8.7 8.3 ...
 $ UBeakL : num  17.7 16 16.5 16.7 16 15 17.5 19 17 17 ...
 $ N.UBkL : num  12 10.5 11 11.5 11 10 12 12.3 11.7 11 ...
 $ TarsusL: num  22 20 20 20.7 21 19.5 20.5 23 21.5 22 ...
```

Note: Factors with *level names with leading or trailing blanks* can cause problems in *ggplot* graphics. Sometimes useful to check this.

```
with(DarwinsFinches, levels(Species))
```

```
[1] "Geospiza fortis fortis"
[2] "Geospiza fortis platyrhyncha"
[3] "Geospiza heliobates"
[4] "Geospiza fuliginosa parvula"
[5] "Geospiza prosthemelas prosthemelas"
```

We also set a few items useful later:

```
## choose species colours
spCols <- c(`Geospiza fortis fortis` =                "#006400",  ## "darkgreen"
            `Geospiza fortis platyrhyncha` =          "#9BCD9B",  ## "darkseagreen3"
            `Geospiza heliobates`  =                  "#B22222",  ## "firebrick"
            `Geospiza fuliginosa parvula` =           "#8B4513",  ## "chocolate4"
            `Geospiza prosthemelas prosthemelas` = "#F4A460")  ## "sandybrown"
sub_text <-  ## for a citation line
  "(Darwins Finch data, Hopkins-Stanford Galapagos Expedition, 1898-99)"
```

## Waffly outline

A simple discriminant function analysis is really just a matrix version of a single classification analysis of variance, with some extensions and matrix operations of various kinds replacing simpler arithmetic ones in the univariate case.

```
Species <- with(DarwinsFinches, Species) ## Species factor
n <- length(Species)                     ## sample size (= 146)
species <- levels(Species)               ## species names
p <- length(species)                     ## number of species (= 5)

Y <- as.matrix(select(DarwinsFinches, BodyL:TarsusL)) ## responses

M0 <- apply(Y, 2, ave)                   ## grand mean
M1 <- apply(Y, 2, ave, Species)          ## species means

B <- crossprod(M1 - M0)/(p - 1)          ## between species MSq
W <- crossprod(Y  - M1)/(n - p)          ## withing species MSq
```

Now that we have the "Between" and "Within" mean SSQ matrices, the main discriminant function calculation is to solve the generalized eigenvalue problem

$$(B - \lambda W)\alpha = \mathbf{0}$$

The eigenvalues $\lambda_i$ are the analogue(s) of the $F-$statistics, and the eigenvectors, $\alpha_i$ the discriminant function coefficients.

We have a special function *eigen2* in `WWRUtilities` to solve these generalized eigenvalue problems, for symmetric matrices, in a stable way.

```
ev <- eigen2(B, W)                                ## critical computation
print(with(ev, zapsmall(values)), digits = 3) ## "F-statistics" should be (p-1)
```

```
[1] 482.97 185.40  32.58   5.38   0.00   0.00   0.00   0.00   0.00
```

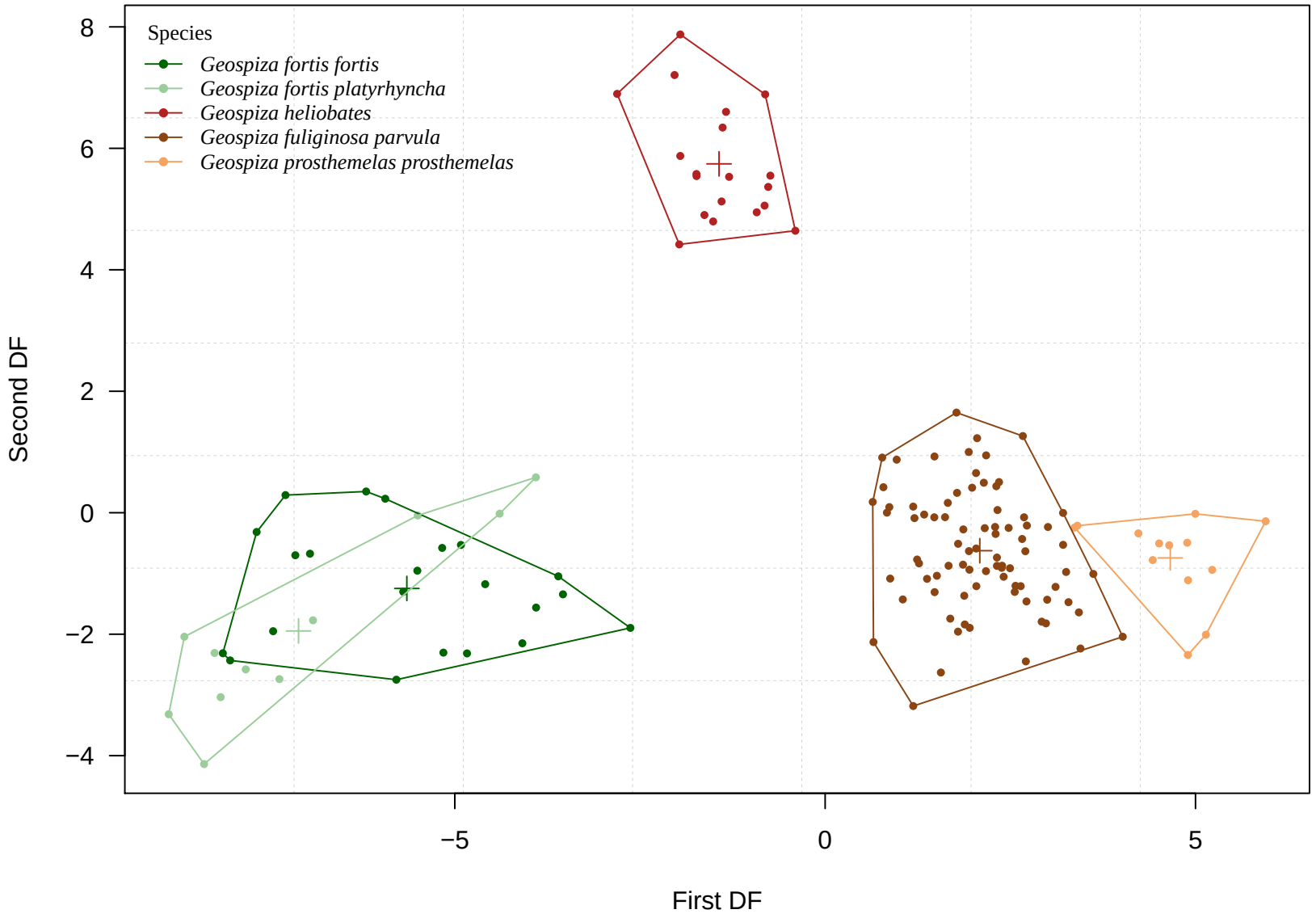
```
alpha <- with(ev, vectors[, 1:(p-1)])                    ## coefficients
Scores <- (Y - M0) %*% alpha                             ## discriminant functions
Finch <- cbind(DarwinsFinches, data.frame(Scores))       ## augmented data
noquote(setdiff(names(Finch), names(DarwinsFinches))) ## extra names
```

```
[1] X1 X2 X3 X4
```

## 1.1 Display the results

We compare a traditional graphics plot with a *ggplot2* version. First, a traditional plot.

```
with(Finch, plot(X1, X2, xlab="First DF", ylab="Second DF", las=1,
                 col=spCols[Species], pch=16, cex=0.7,
                 panel.first=grid(lty="dashed", lwd=0.5, nx=7)))
by(Finch, Species, FUN=function(dat) with(dat, {
  spColour <- spCols[Species[1]]                  ## get species col
  z <- complex(real=X1, imaginary=X2)             ## complex trick!
  points(mean(z), pch=3, cex=1.5, col=spColour) ## centroid
  polygon(z[chull(z)], col="transparent",
          border=spColour)                        ## convex hulls
})) %>% invisible()                               ## toss dummy output
mtext(sub_text, side=1, line=4, adj=1, cex=0.75, family="serif")
par(family="serif")                             ## outside legene()
legend("topleft", species, pch=16, lty="solid", col=spCols, cex=0.8,
       title="Species", text.font=3, title.adj=0.025, bty="n",
       inset=c(0.0125, 0.0125))
```




(Darwins Finch data, Hopkins−Stanford Galapagos Expedition, 1898−99)

Now for a *ggplot()* version. The computations need to be done prior to the construction of the plot object.

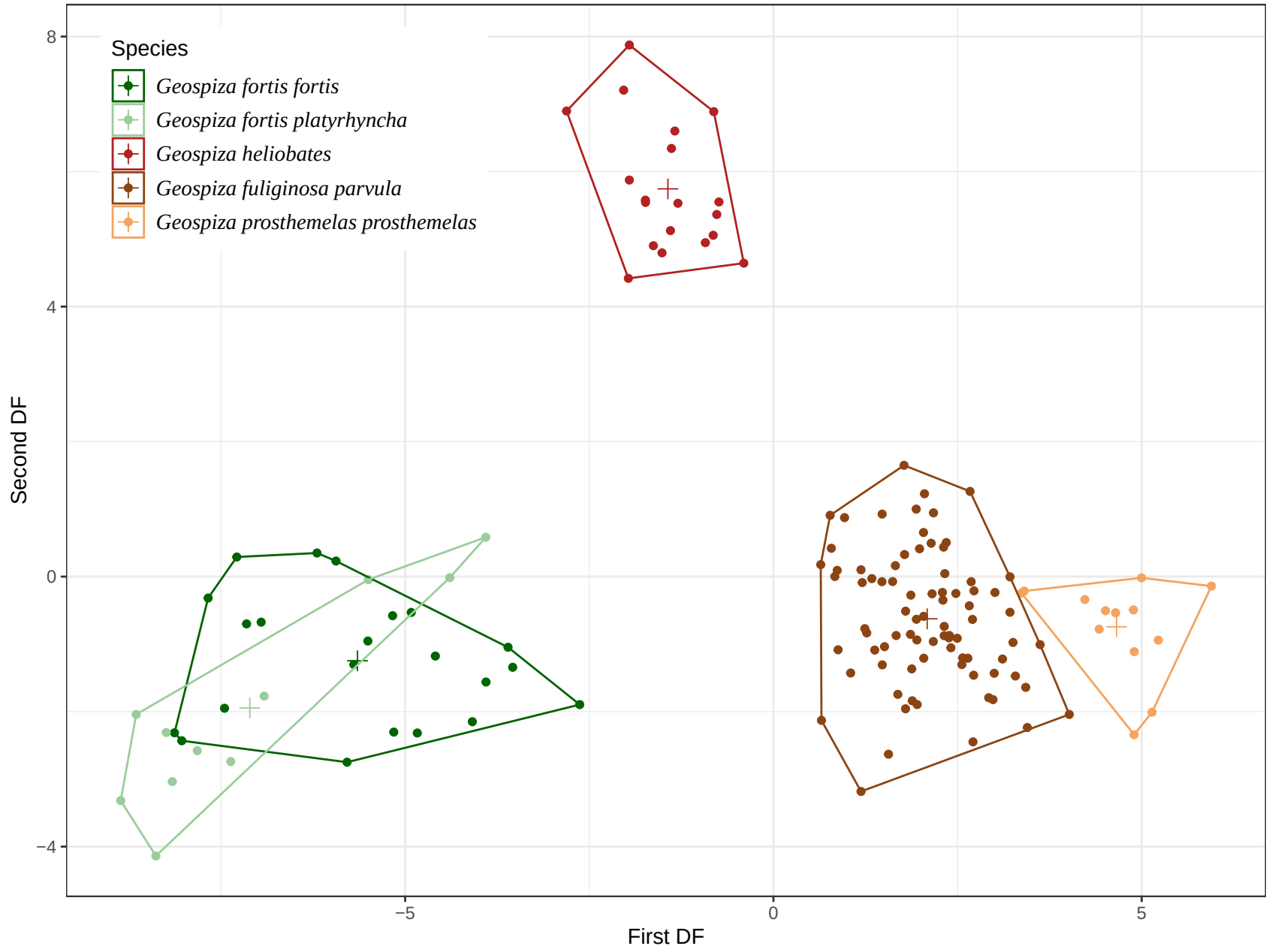
```
finch_centroids <- Finch %>%            ## for the central points
  group_by(Species) %>%
  summarise(X1 = mean(X1), X2 = mean(X2)) %>%
  ungroup()

finch_hulls <- Finch %>%                ## this is the tricky one
  group_by(Species) %>%
  do(., with(., {                       ## the complex trick does
    h <- chull(cbind(X1, X2))           ## not work with ggplot()
    data.frame(Species = Species[h], X1 = X1[h], X2 = X2[h])
  })) %>%
  ungroup()
```

An alternative using a combination of new and more traditional tools:

```
separate_species <- split(Finch, Species)  ## a list of 5 data frames
finch_centroids_1 <- separate_species %>%
  lapply(function(dat) with(dat,
                            data.frame(Species = Species[1],
                                       X1 = mean(X1),
                                       X2 = mean(X2)))) %>%
  do.call(rbind, .)
finch_hulls_1 <- separate_species %>%
  lapply(function(dat) with(dat, {
    h <- chull(cbind(X1, X2))
    data.frame(Species = Species[h],
               X1 = X1[h],
               X2 = X2[h])
  })) %>% do.call(rbind, .)
```

The display is now straightforward

```
plt <- ggplot() +
  aes(x = X1, y = X2, color = Species) +  ## inherited to all
  geom_point(data = Finch) +
  geom_point(data = finch_centroids, shape = 3, size = 3) +
  geom_polygon(data = finch_hulls, fill = "transparent") +
  scale_color_manual(values = spCols) +
  labs(caption = sub_text, x = "First DF", y = "Second DF") +
  theme_bw() +
  theme(legend.position = c(0.19, 0.85),
        legend.text = element_text(family = "serif",
                                   face = "italic", size = 11))
plt
```




(Darwins Finch data, Hopkins−Stanford Galapagos Expedition, 1898−99)

# 2 The churn data

> "Customer churn" refers to [the situation] when a customer (player, subscriber, user, &c.) ceases his or her relationship with a company. Online businesses typically treat a customer as churned once a particular amount of time has elapsed since the customer's last interaction with the site or service.
>
> Dr Google, 2019-01-24

- A *churn* data set is provided, in raw form, in the `extdata` folder of the `WWRCourse` package

```
dir(system.file("extdata", package = "WWRCourse"))
```

```
[1] "Boab.jpg"                  "churnData.csv.gz"
[3] "creditCard.csv"            "creditCard.csv.gz"
[5] "Genesis.txt.gz"            "mammogram_original.csv.gz"
[7] "mammogram.csv.gz"          "Revelations.txt.gz"
[9] "stata"
```

The file is compressed but to read it in does not require it to be de-compressed first

```
fname <- system.file("extdata", "churnData.csv.gz", package = "WWRCourse")
churnData <- read_csv(gzfile(fname))      ## initial read


-- Column specification --------------------------------------------------
cols(
.default = col_double(),
state = col_character(),
area_code = col_character(),
international_plan = col_character(),
voice_mail_plan = col_character(),
churn = col_character(),
sample = col_character()
)
i Use `spec()` for the full column specifications.
```

To suppress this warning, use the output to make a neater read for your eventual script

```
churnData <- read_csv(gzfile(fname),      ## neater read (for eventual notebook)
                      col_types = cols(.default = col_double(),
                                       state = col_character(),
                                       area_code = col_character(),
                                       international_plan = col_character(),
                                       voice_mail_plan = col_character(),
                                       churn = col_character(),
                                       sample = col_character()))
names(churnData) %>% noquote()
```

```
 [1] state                          account_length
 [3] area_code                      international_plan
 [5] voice_mail_plan                number_vmail_messages
 [7] total_day_minutes              total_day_calls
 [9] total_day_charge               total_eve_minutes
[11] total_eve_calls                total_eve_charge
[13] total_night_minutes            total_night_calls
[15] total_night_charge             total_intl_minutes
[17] total_intl_calls               total_intl_charge
[19] number_customer_service_calls churn
[21] sample
```

- The data frame is from a *Kaggle* competition and details of the data, (assumed genuine), are sparse.
- The sampled entities are previous customers of a large telecommunications company.
- The response is *churn*, a character variable with two possible values.
- The data is already split into *train* and *test* samples, as given by the final column.
- The challenge is to build a predictor for the response.

```
dim(churnData)
```

```
[1] 5000   21
```

```
with(churnData, table(churn, sample))
```

```
     sample
churn test train
  no  1443  2850
  yes  224   483
```

## 2.1 Character variables to factors: chaining

- The function *read_csv* produces a result of class *"tibble"*, an enhanced *data.frame*.
- By default, "*tibble*"s have character variables where usually *data.frame*s have *factor*s. [Until R 4.0.0 changed this!]
- For modelling purposes, factors are needed rather than character strings. How do we do it?

```
churnData <- data.frame(unclass(churnData), stringsAsFactors = TRUE)
```

- The *chain operator*, *%>%*, (also known as a *pipe* operator) allows this kind of nested expression to be written in a linear form that is easier to disentangle:

```
churnData <- churnData %>%                    ## start with the data...
  unclass(.)           %>%                    ## then remove the class()...
  data.frame(., stringsAsFactors = TRUE)      ## then make it a data.frame()
```

The "dot" place-holder, ., may be omitted if it is the first argument.

## 2.2 A first look at the data

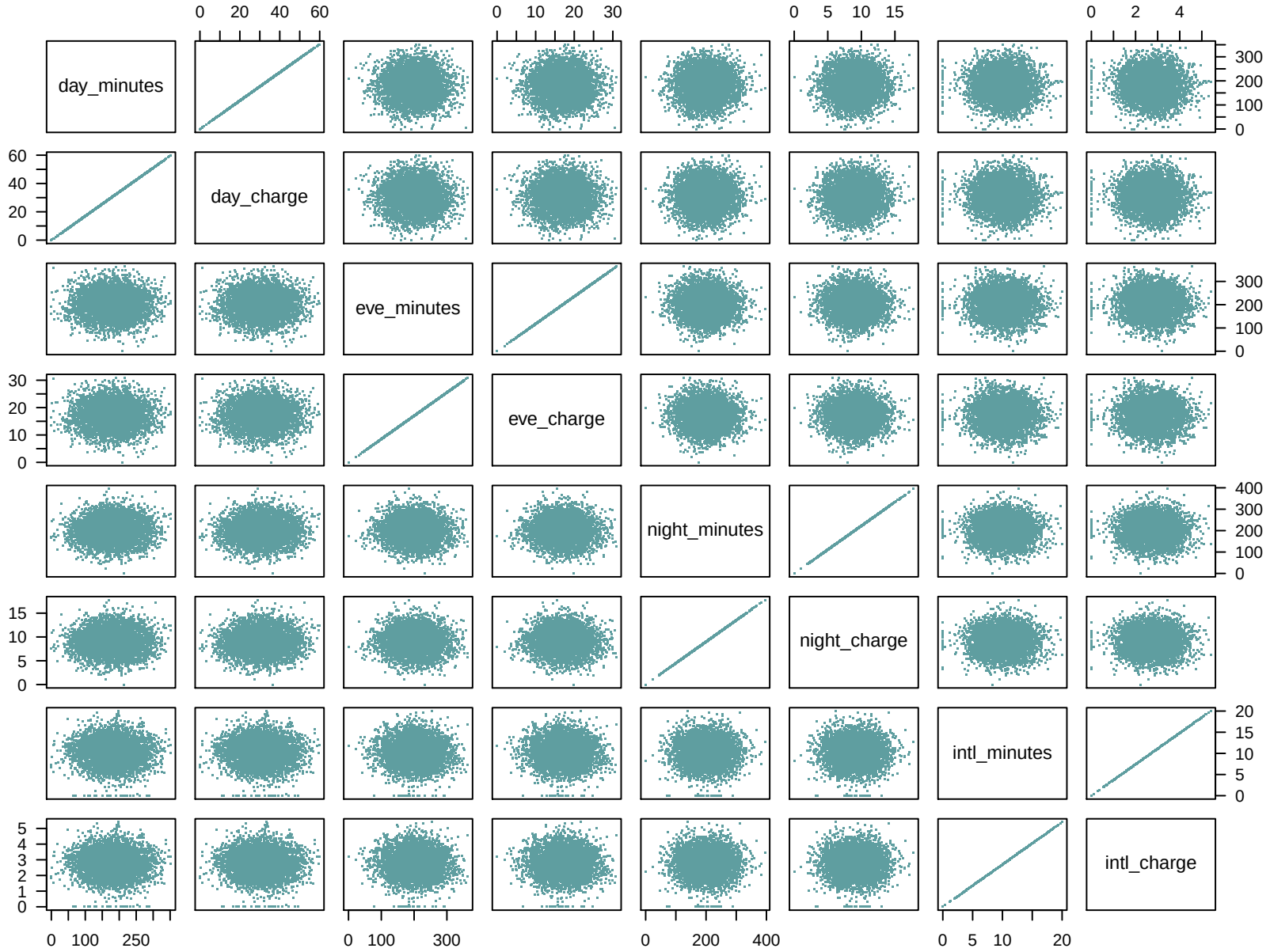
First let's shorten the names a bit and make a small change to the *area_code* variable

```
churnData <- churnData %>%
  within({
    area_code <- sub("^area_code_", "", area_code) %>%
      factor()
  })
with(churnData, table(area_code))
```

```
area_code
 408  415  510
1259 2495 1246
```

There is some redundancy in the data. A simple graphical exploration:

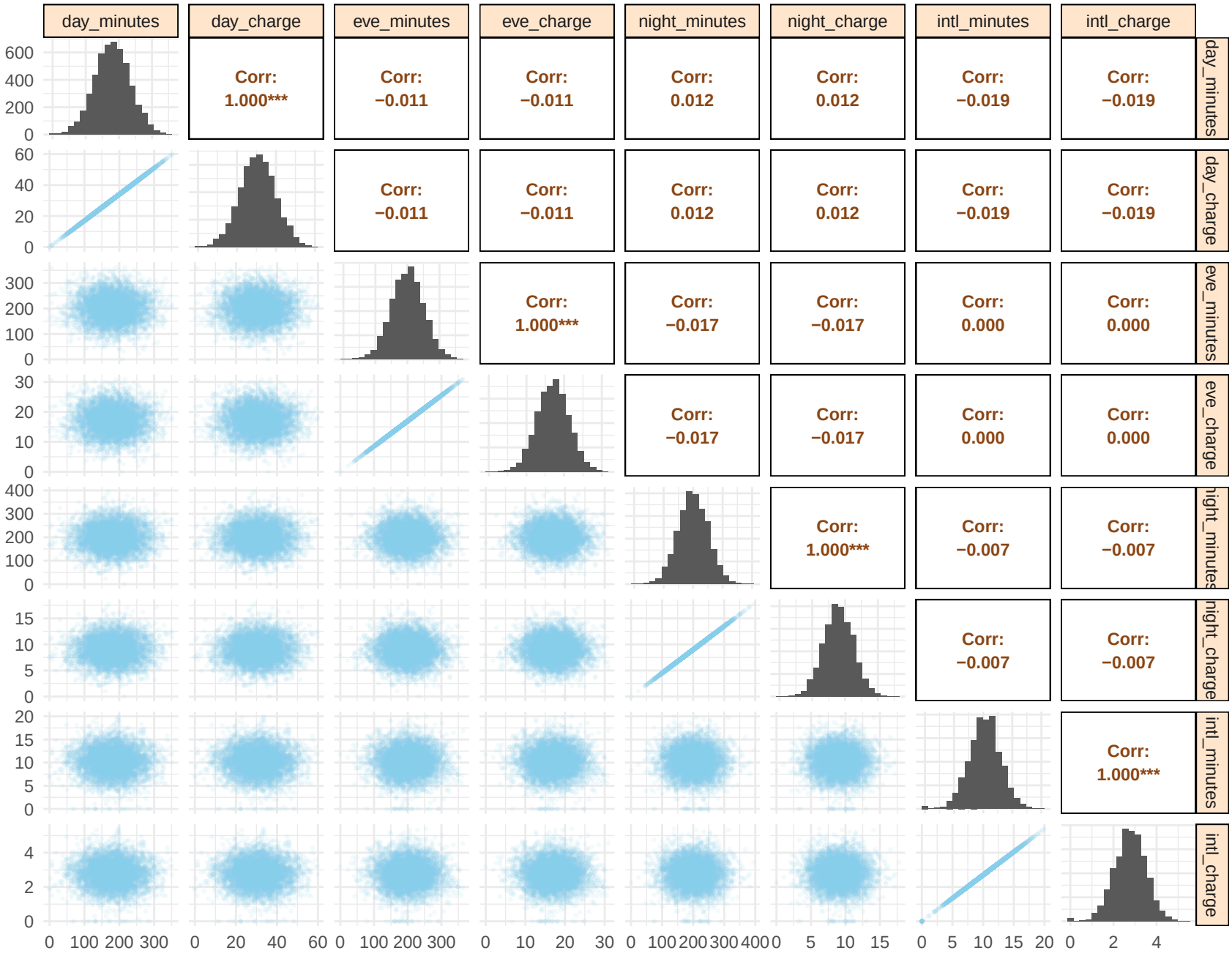
```
names(churnData) <- sub("^(total|number)_", "", names(churnData)) ## neater
tots <- churnData %>% select(ends_with("_minutes"),
                             ends_with("_charge"),
                             ends_with("_calls"))
pairs(tots, pch = ".")          ## old technology! (result not shown)
######
## cut down the number of panels and group variables for effect
######
tots <- tots %>%
  select(starts_with("day"),   starts_with("eve"),
         starts_with("night"), starts_with("intl"),
         -ends_with("_calls"))                ## remove the calls
pairs(tots, pch = ".", col = "cadet blue")  ## notice anything odd?
```


day_minutes
day_charge
eve_minutes
eve_charge
night_minutes
night_charge
intl_minutes
intl_charge

### 2.2.1 The *ggplot pairs alternative*

The *ggplot* version of *pairs* is non-standard with regards to the normal *ggplot* call structure. It is provided in the (somewhat eccentric) *GGally* package. (The *Corr:* annotation is hard-wired!)

```
suppressPackageStartupMessages({
  library(GGally)
})
ggpairs(data = tots,
        upper = list(continuous=wrap("cor", size=3, fontface="bold",
                                     colour="saddlebrown")),
        diag = list(continuous=wrap("barDiag", bins=20)),
        lower = list(continuous=wrap("points", size=0.5,
                                     colour=alpha("skyblue", 1/10))))
```


day_minutes
day_charge
eve_minutes
eve_charge
night_minutes
night_charge
intl_minutes
intl_charge
Corr: 1.000***
Corr: −0.011
Corr: 0.012
Corr: −0.019
Corr: 1.000***
Corr: −0.017
Corr: 0.000
Corr: 1.000***
Corr: −0.007
Corr: 1.000***

The "_charge" variables are almost an exact re-scaling of the corresponding "_minutes" variables. Look at the key block or correlations:

```
cor(churnData %>% select(ends_with("_minutes")),
    churnData %>% select(ends_with("_charge")))

               day_charge    eve_charge night_charge   intl_charge
day_minutes    0.99999995 -0.0107600217  0.011782533 -0.0194147972
eve_minutes   -0.01074730  0.9999997749 -0.016642042  0.0001593155
night_minutes  0.01180143 -0.0166489191  0.999999207 -0.0066545873
intl_minutes  -0.01948970  0.0001320238 -0.006716538  0.9999926570
```

Two issues:

- The correlations between corresponding minutes and charge variables are *far too high*, indicating they are essentially the *same* variable, just expressed in different units.
- The correlations between other pairs of variables are *uniformly too low* to be fully credible!

The data is almost certainly, *fake*! Disappointing, but not surprising.

## 2.3 Looking at proportions

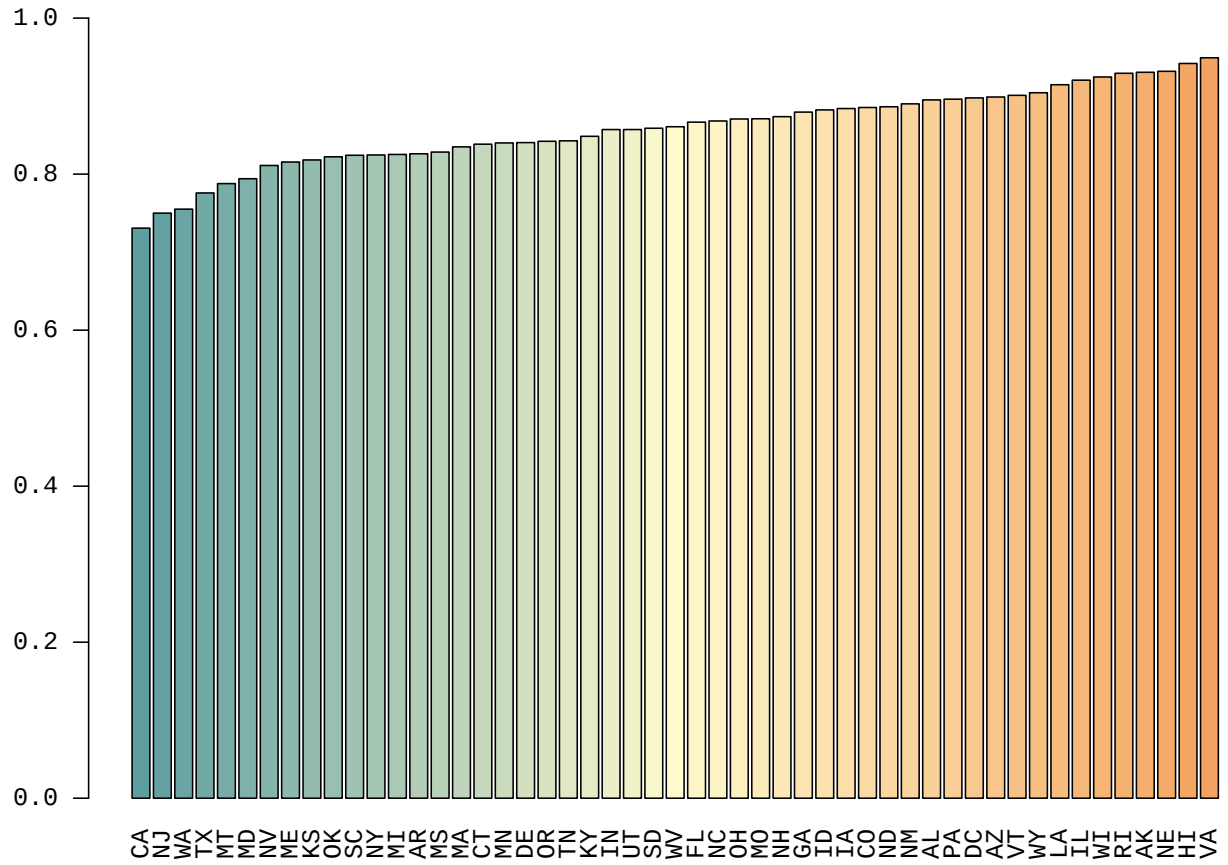
- So far only considered *predictors*. What about the response?
- *state* and *day_minutes* are likely to be useful, so consider a graphical exploration of the *marginal* dependency patterns.

```
state_props <- churnData                          %>%
  group_by(state)                                 %>%
  summarise(n = n(),                                    ## state total
            p = mean(churn == "no")) %>%                ## state proportion
  arrange(p)                                            ## 'league table' ordering
head(state_props %>% untibble(), 6)                     ## Californians!
```

```
  state   n         p
1    CA  52 0.7307692
2    NJ 112 0.7500000
3    WA  98 0.7551020
4    TX 116 0.7758621
5    MT  99 0.7878788
6    MD 102 0.7941176
```

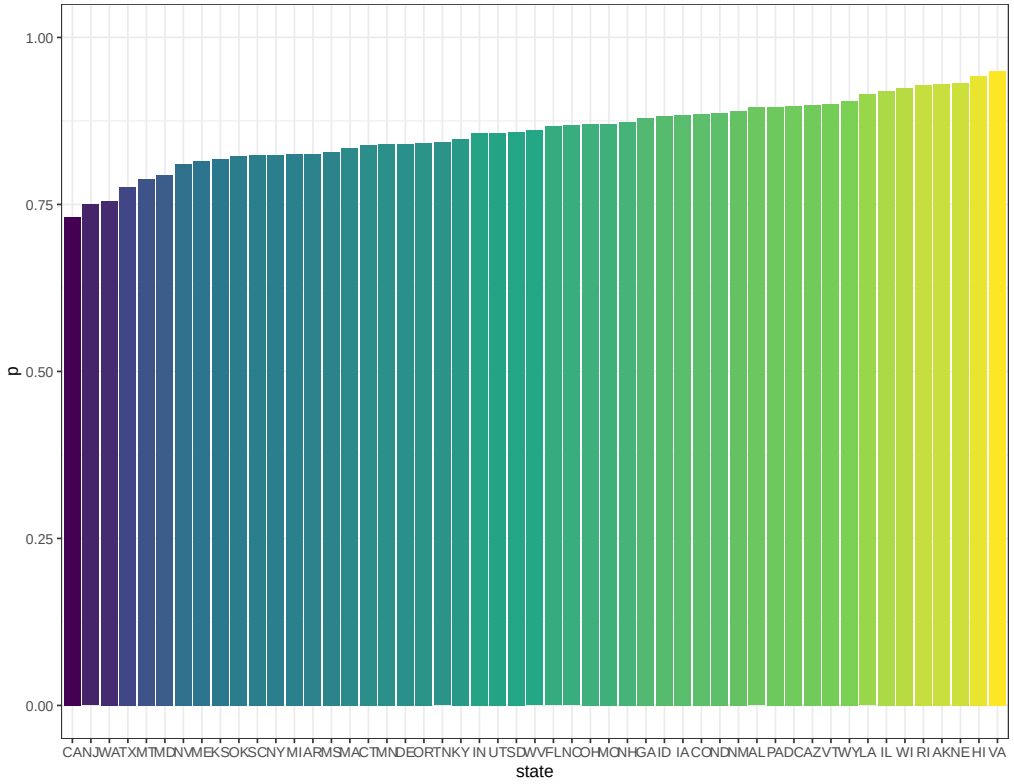
An 'old technology' display:

```
with(state_props,
     barplot(p, names.arg = state, las = 2, ylim = c(0,1),
             family = "mono", fill = pal_sea2sand)) ## colours are pointless!!
```



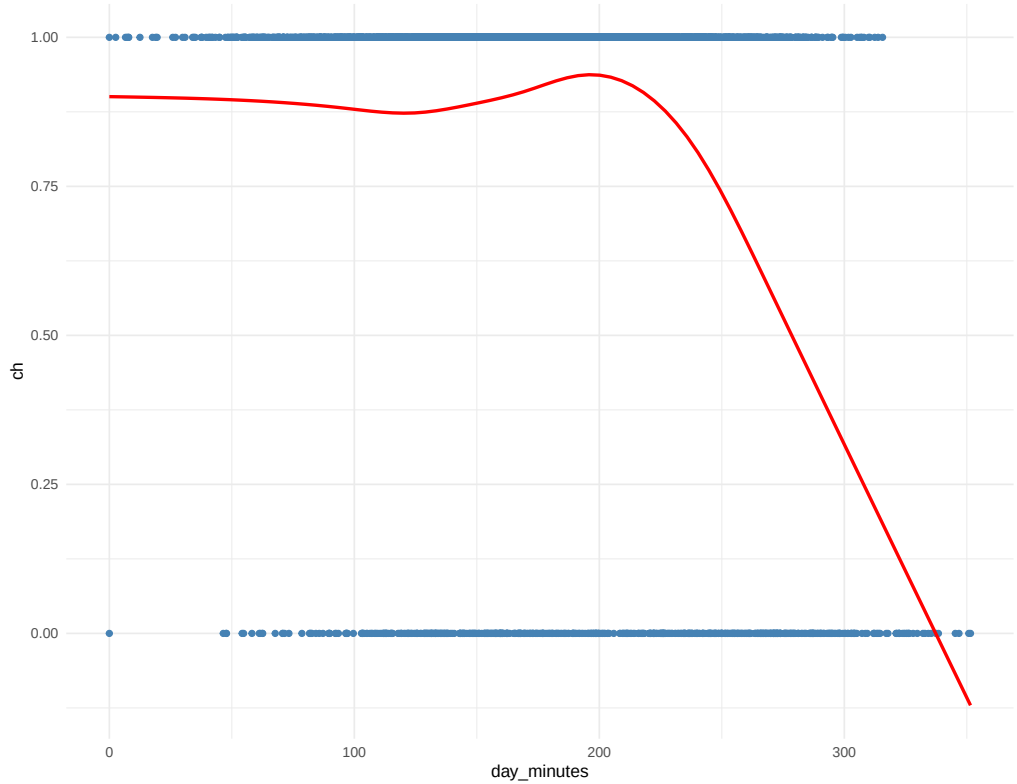
A new technology version:

```
state_props <- state_props %>%
  mutate(state = factor(as.character(state), levels = as.character(state)))
ggplot(state_props) + aes(x = state, y = p, fill = p) + ylim(0, 1) +
  geom_bar(stat = "identity") + scale_fill_viridis_c() + theme_bw() +
  theme(legend.position="none")
```



# A continuous predictor: how do we handle that?

```
churn_tmp <- churnData %>%
  mutate(ch = (churn == "no") + 0) ## binary numeric
ggplot(churn_tmp) + aes(x = day_minutes, y = ch) +
  geom_point(colour = "steelblue") +
  geom_smooth(se = FALSE, method = "gam", formula = y ~ s(x, bs = "cs"),
              colour = "red")
```



### 2.3.1 Local weighting: compiled code

Try to do something similar using lower level tools. Two functions

```
localWeighting <- function(x, y, scale) {
  y_wtd <- numeric(length(y))
  for(i in seq_along(y)) {
    y_wtd[i] <- weighted.mean(y, w = exp(-((x-x[i])/scale)^2))
  }
  y_wtd
}
```

Now for a compiled version using Rcpp:

```cpp
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector localWeightCpp(NumericVector x,
                             NumericVector y,
                             double scale) {
  int n = y.size();
  NumericVector y_wtd(n);

  for(int i = 0; i < n; i++) {
    double w, swy = 0.0, sw = 0.0;
    for(int j = 0; j < n; j++) {
      w = exp(-pow((x[i] - x[j])/scale, 2));
      swy += w * y[j];
      sw += w;
    }
    y_wtd[i] = swy/sw;
  }
  return y_wtd;
}
```

## Try it out

```
yw1 <- with(churn_tmp, localWeighting(day_minutes, ch, scale = 25))
yw2 <- with(churn_tmp, localWeightCpp(day_minutes, ch, scale = 25))
all.equal(yw1, yw2)
```

```
[1] TRUE
```

## But what about timing?

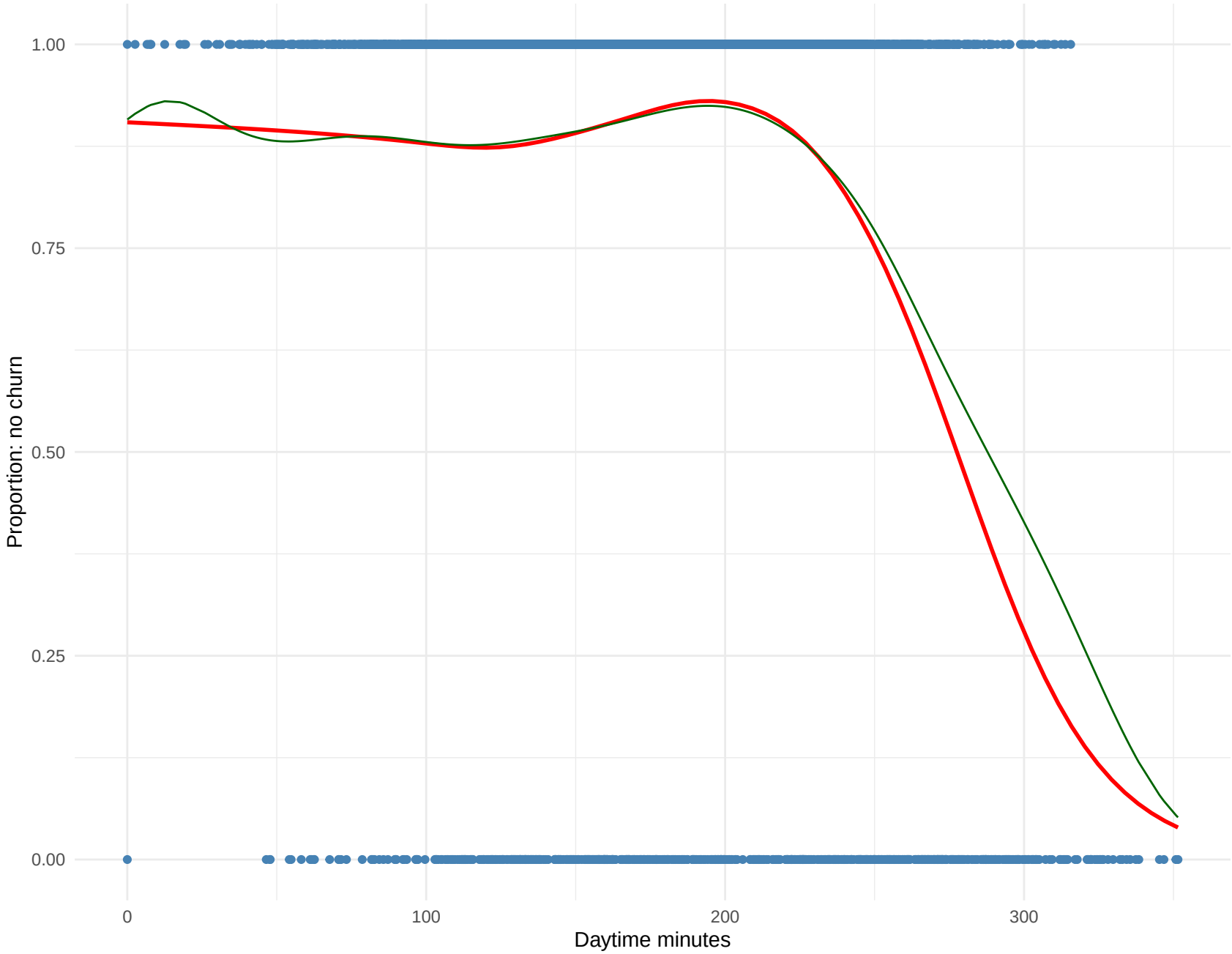
```
library(rbenchmark)
with(churn_tmp,
     benchmark(localWeighting(day_minutes, ch, scale = 25),
               localWeightCpp(day_minutes, ch, scale = 25),
     replications = 10, columns = c("test", "elapsed", "relative")))
```

```
                                       test elapsed relative
2 localWeightCpp(day_minutes, ch, scale = 25)   1.845    1.000
1 localWeighting(day_minutes, ch, scale = 25)   3.610    1.957
```

## Finally, how does it compare with the smooth?

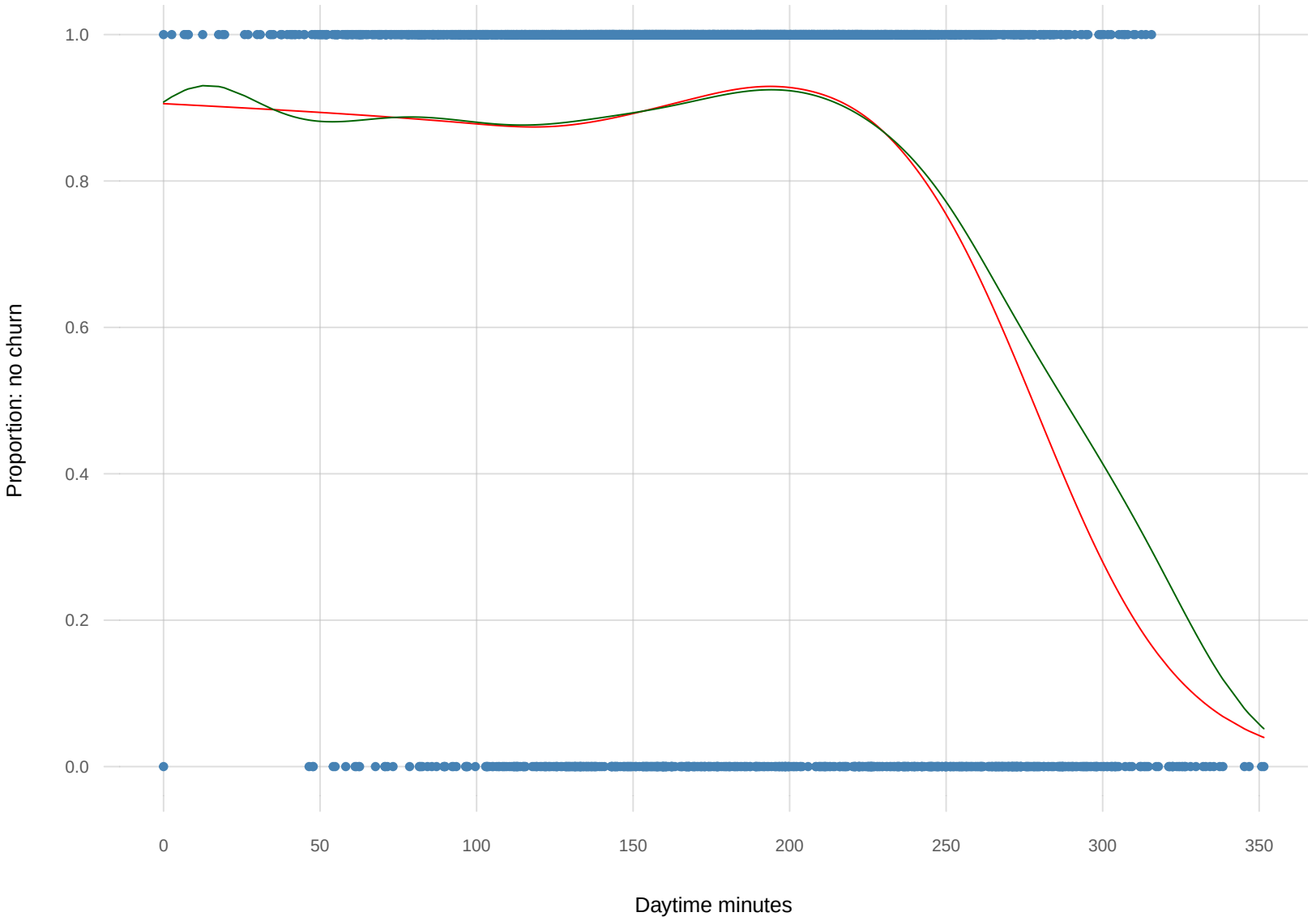
```
churn_tmp <- churn_tmp %>%
  mutate(p_wt = localWeightCpp(day_minutes, ch, scale = 25))
ggplot(churn_tmp) + aes(x = day_minutes, y = ch) +
```

```
  geom_point(colour = "steelblue") +
  geom_smooth(se = FALSE,
              method = "gam", formula = y ~ s(x, bs = "cs"),
              method.args = list(family = "binomial"),
              colour = "red") +
  geom_line(aes(x = day_minutes, y = p_wt), colour = "dark green") +
  ylab("Proportion: no churn") + xlab("Daytime minutes")
###
### There seems to be broad qualitative agreement.
```


1.00
0.75
0.50
0.25
0.00
Proportion: no churn
0
100
200
300
Daytime minutes

An old technology plot to achieve the same purpose:

```
suppressPackageStartupMessages(library(mgcv))
greyish <- grey(0.35)
ghostgrey <-  alpha("grey", 0.5)
churn_tmp %>%
  arrange(day_minutes) %>%   # x now has to be ordered.
  with({
    plot(day_minutes, ch, pch = 20, axes = FALSE, bty = "n", type = "n",
         xlab = "Daytime minutes", ylab = "Proportion: no churn",
         cex.lab = 0.8)
    axis(1, col = "transparent", col.axis = greyish,
         cex.axis = 0.7, col.ticks = ghostgrey)
    axis(2, col = "transparent", col.axis = greyish,
         cex.axis = 0.7, col.ticks = ghostgrey)
    grid(lty = "solid", col = ghostgrey)
    points(day_minutes, ch, pch = 20, col = "steelblue")
    smooth <- fitted(gam(ch ~ s(day_minutes, bs = "cs"), binomial))
    lines(day_minutes, smooth, col = "red")
    lines(day_minutes, p_wt, col = "dark green")
  })
```


1.0
0.8
0.6
0.4
0.2
0.0
Proportion: no churn
0
50
100
150
200
250
300
350
Daytime minutes

## 2.4 Splitting the data

The data need not be split, but it is handy to do so for convenience in modelling. The split data sets should have the *sample* column removed. We look at a couple of ways to do this.

**The antediluvian method**:

```
sample_column <- which(names(churnData) == "sample")  ## calculate it!
churn_train <- churnData[churnData$sample == "train", -sample_column]
churn_test  <- churnData[churnData$sample == "test" , -sample_column]
```

**The classical method**:

```
churn_train <- subset(churnData, sample == "train", select = -sample)
churn_test  <- subset(churnData, sample == "test" , select = -sample)
```

**The modern method**:

```
churn_train <- churnData %>% filter(sample == "train") %>% select(-sample)
churn_test  <- churnData %>% filter(sample == "test" ) %>% select(-sample)
```

(To be continued?)

# 3 The Quine data: spreading, gathering and merging

- Children from Walgett, New South Wales, Australia, were classified by Culture, Age, Sex and Learner status and the number of days absent from school in a particular school year was recorded.
- The quine data frame has 146 rows and 5 columns.

```
find("quine")  ## Check that you have it.  Otherwise quine <- MASS::quine
```

```
[1] "package:WWRData"
```

```
str(quine)
```

```
'data.frame': 146 obs. of  5 variables:
 $ Eth : Factor w/ 2 levels "A","N": 1 1 1 1 1 1 1 1 1 1 ...
 $ Sex : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ Age : Factor w/ 4 levels "F0","F1","F2",..: 1 1 1 1 1 1 1 1 2 2 ...
 $ Lrn : Factor w/ 2 levels "AL","SL": 2 2 2 1 1 1 1 1 2 2 ...
 $ Days: int  2 11 14 5 5 13 20 22 6 6 ...
```

Task 1: tables of counts, means and standard deviations.

```
counts <- with(quine, table(Eth, Sex, Age, Lrn)) %>%
  as.data.frame(responseName = "Count")
pivot_wider(counts, names_from = Age, values_from = Count) %>% booktabs()
```

| Eth | Sex | Lrn | F0 | F1 | F2 | F3 |
|---|---|---|---|---|---|---|
| A | F | AL | 4 | 5 | 1 | 9 |
| N | F | AL | 4 | 6 | 1 | 10 |
| A | M | AL | 5 | 2 | 7 | 7 |
| N | M | AL | 6 | 2 | 7 | 7 |
| A | F | SL | 1 | 10 | 8 | 0 |
| N | F | SL | 1 | 11 | 9 | 0 |
| A | M | SL | 3 | 3 | 4 | 0 |
| N | M | SL | 3 | 7 | 3 | 0 |

## A second display

```
tab <- counts %>%
  mutate(AgeSex = Age:Sex) %>%
  pivot_wider(id_cols = c(Eth, Lrn),
              names_from = AgeSex, values_from = Count) %>%
  arrange(Lrn, Eth)
booktabs(tab)
```

| Eth | Lrn | F0:F | F0:M | F1:F | F1:M | F2:F | F2:M | F3:F | F3:M |
|---|---|---|---|---|---|---|---|---|---|
| A | AL | 4 | 5 | 5 | 2 | 1 | 7 | 9 | 7 |
| N | AL | 4 | 6 | 6 | 2 | 1 | 7 | 10 | 7 |
| A | SL | 1 | 3 | 10 | 3 | 8 | 4 | 0 | 0 |
| N | SL | 1 | 3 | 11 | 7 | 9 | 3 | 0 | 0 |

How would we put this table back into the original form?

I'm glad you asked.

```
counts1 <- tab %>%
  pivot_longer(names_to = "AgeSex", values_to = "Count",
               `F0:F`:`F3:M`) %>%  ## note `...`
  separate(AgeSex, into = c("Age", "Sex"), sep = ":") %>%
  select(names(counts)) %>%
  unclass() %>% data.frame() %>%
  arrange(Lrn, Age, Sex, Eth)
# cbind(counts, ".  " = "    ", counts1) ## crude check
all.equal(counts, counts1)
```

```
[1] "Component \"Sex\": 'current' is not a factor"
[2] "Component \"Age\": 'current' is not a factor"
```

The content of *counts* and *counts1* is the same, but columns are in a different order.[a]

[a] **Problem:** Are there any other differences? How would you check, and would you adjust *counts1* *gracefully* so that it becomes identical to the original *counts*?

## 3.1 Mean-variance relationship

- The main purpose is to look at the mean-variance relationship:
  - An approximately linear one would indicate a *poisson* or *quasipoisson* model for the count response, *Days*, is appropriate,
  - *An approximately linear relationship between the mean and the* standard deviation *suggest something like a Negative Binomial model for the count response, Days, is appropriate.*
  - *More particularly, the Negative Binomial distribution has a mean-variance relationship of the form*

$$Var[Y] = \mu + \mu^2/\theta$$

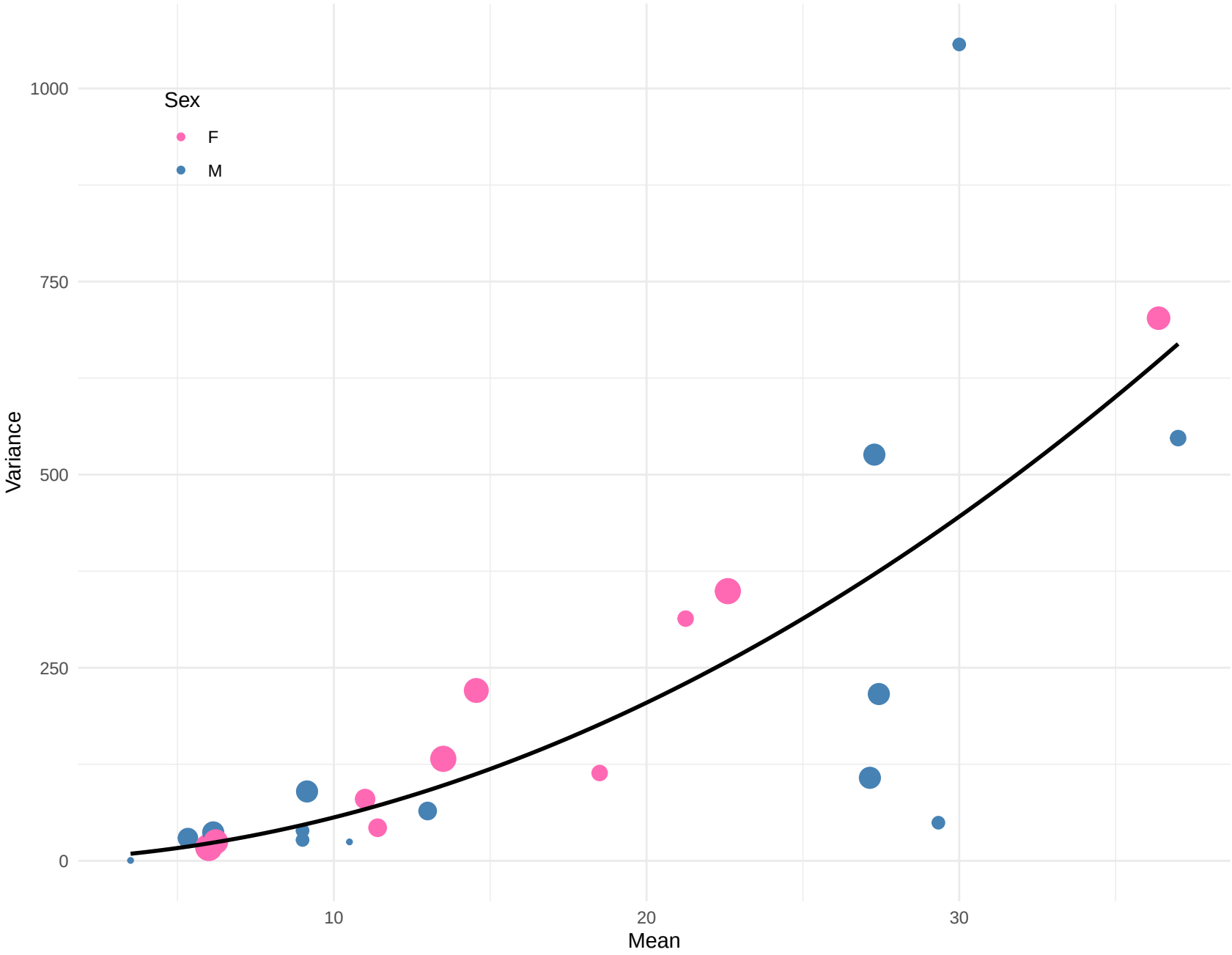
*We will go to this form directly, as we strongly suspect the NB will be a good model.*

- The plot should contain extra information, but not too much!

```
Stats <- quine %>%
  group_by(Age, Sex, Eth, Lrn) %>%
  summarise(Count = n(), Mean = mean(Days), S2 = var(Days)) %>%
  ungroup() %>% untibble()
```

```
`summarise()` has grouped output by 'Age', 'Sex', 'Eth'.  You can override
using the `.groups` argument.
```

```
Stats1 <- na.omit(Stats)  ## it will happen anyway!
#####
##### mean-variance plot
#####
ggplot(Stats1) + aes(x=Mean, y=S2, size=Count, colour=Sex) +
  geom_point()  + ylab("Variance") +
  scale_colour_manual(values = c(F = "hotpink", M = "steelblue")) +
  stat_smooth(data=Stats1, aes(x=Mean, y=S2, weight = Count),
              method="lm", formula = y ~ 0+offset(x)+I(x^2),
              colour="black", se=FALSE) +
  guides(size = "none") + theme(legend.position = c(0.1, 0.85))
```


Sex
F
M
1000
750
500
250
0
Variance
10
20
30
Mean

This suggests we could get a good initial estimate for $\theta$, which we can then compare with the maximum likelihood estimate:

```
mv0 <- lm(S2 ~ 0 + offset(Mean) + I(Mean^2), Stats, weights = Count)
summary(mv0)$coefficients
```

```
           Estimate Std. Error  t value     Pr(>|t|)
I(Mean^2) 0.4618429 0.05250985 8.795357 8.125603e-09
```

```
(theta_0 <- as.vector(1/coef(mv0)["I(Mean^2)"]))
```

```
[1] 2.165238
```

```
(theta_ml <- with(quine, theta.ml(Days, ave(Days, Eth, Sex, Age, Lrn))))
```

```
[1] 1.92836
attr(,"SE")
[1] 0.2688968
```

```
as.vector(theta_0 - theta_ml)/attr(theta_ml, "SE")  ## "quasi-t-statistic"
```

```
[1] 0.8809264
```

(To be continued.)

# 4 Synopsis

| Issue | Tools |
|---|---|
| Input | `gzfile`, `read_csv`, `tibble`, ... |
| Chaining | %>% binary operator |
| Manipulation | Five key functions: |
| | `select, filter, mutate, group_by, summarise` |
| | Helper functions: |
| | `begins_with, ends_with, contains, everything, ...` |
| Shaping | Two key functions: `pivot_longer, pivot_wider` |
| | Helper functions: `separate` |
| Old technology | `with, within, subset, tapply, sapply, lapply, ...` |

# Session information

**Date: 2021-01-29**

- R version 4.0.3 (2020-10-10), `x86_64-pc-linux-gnu`
- Running under: `Ubuntu 20.04.1 LTS`
- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0`
- LAPACK: `/usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: dplyr 1.0.3, english 1.2-5, forcats 0.5.1, GGally 2.1.0, ggplot2 3.3.3, ggthemes 4.2.4, gridExtra 2.3, knitr 1.31, lattice 0.20-41, mgcv 1.8-33, nlme 3.1-151, patchwork 1.1.1, purrr 0.3.4, rbenchmark 1.0.0, readr 1.4.0, scales 1.1.1, stringr 1.4.0, tibble 3.0.5, tidyr 1.1.2, tidyverse 1.3.0, WWRCourse 0.2.3, WWRData 0.1.0, WWRGraphics 0.1.2, WWRUtilities 0.1.2, xtable 1.8-4
- Loaded via a namespace (and not attached): assertthat 0.2.1, backports 1.2.1, broom 0.7.3, cellranger 1.1.0, cli 2.2.0, colorspace 2.0-0, compiler 4.0.3, crayon 1.3.4, DBI 1.1.1, dbplyr 2.0.0, digest 0.6.27, ellipsis 0.3.1, evaluate 0.14, fansi 0.4.2, farver 2.0.3, fractional 0.1.3, fs 1.5.0, generics 0.1.0, glue 1.4.2, grid 4.0.3, gtable 0.3.0, haven 2.3.1, highr 0.8, hms 1.0.0, httr 1.4.2, iterators 1.0.13, jsonlite 1.7.2, labeling 0.4.2, lazyData 1.1.0, lifecycle 0.2.0, lubridate 1.7.9.2, magrittr 2.0.1, MASS 7.3-53, Matrix 1.3-2, modelr 0.1.8, munsell 0.5.0, parallel 4.0.3, PBSmapping 2.73.0, pillar 1.4.7, pkgconfig 2.0.3, plyr 1.8.6, R6 2.5.0, randomForest 4.6-14, RColorBrewer 1.1-2, Rcpp 1.0.6, readxl 1.3.1, reprex 1.0.0, reshape 0.8.8, rlang 0.4.10, rpart 4.1-15, rstudioapi 0.13, rvest 0.3.6, SOAR 0.99-11, splines 4.0.3, stringi 1.5.3, tidyselect 1.1.0, tools 4.0.3, vctrs 0.3.6, viridisLite 0.3.0, withr 2.4.1, xfun 0.20, xml2 1.3.2