Applications

Of Data Science

The Trees

Applications of Data Science - Class 12

Giora Simchoni

`gsimchoni@gmail.com and add #dsapps in subject`

Stat. and OR Department, TAU

2020-03-01

The Pros and Cons of Trees

Pros

- It's just a set of if-else statements my 7 y/o can get
- Highly interpretable (when tree is not large)
- Easy to implement
- Fast? (to predict)
- Little pre-processing of predictors needed
- Handle all types of predictors (continuous, categorical)
- Handle missing data, *predict* missing data
- Assumption free
- Feature selection built-in (but when predictors are correlated...)
- Low bias, in general

Cons

- High variance, in general
- Rectangular predictor regions - not always a good thing
- Complexity of prediction limited in no. of leaves! (For a simple CART)
- Not so fast? (to train)
- Greedy
- Selection Bias of predictors with more distinct values?
- Variable Importance when predictors are correlated

Detour: A Regression Problem

OKCupid: Predicting Annual Income

It won't be easy:

```
okcupid <- read_csv("../data/okcupid.csv.zip")

okcupid %>% count(income, sort = TRUE) %>% head(20)
```

```
## # A tibble: 13 x 2
##     income     n
##      <dbl> <int>
##  1      -1 48442
##  2   20000  2952
##  3  100000  1621
##  4   80000  1111
##  5   30000  1048
##  6   40000  1005
##  7   50000   975
##  8   60000   736
##  9   70000   707
## 10  150000   631
## 11 1000000   521
## 12  250000   149
## 13  500000    48
```

We will stick to non-NA (income) observations, and predict $\log_{10}(income/100000)$:

```
okcupid2 <- okcupid %>%
  mutate(income = ifelse(income == -1, NA, log10(income/100000)))
  drop_na(income)
```

In the vector `predictors` (see slides Rmd source) we have 42 continuous and categorical variables which may or may not be predictive to income:

```
okcupid2 <- okcupid2 %>%
  select(income, predictors) %>%
  mutate(id = 1:n())

dim(okcupid2)
```

```
## [1] 11504    44
```

```
glimpse(okcupid2)
```

```
## Observations: 11,504
## Variables: 44
## $ income                 <dbl> -0.09691001, -0.69897000, -0.39794001, -..
## $ age                    <dbl> 35, 23, 28, 30, 29, 40, 31, 22, 35, 31, ..
## $ height_cm              <dbl> 177.80, 180.34, 182.88, 167.64, 157.48, ..
## $ sex                    <fct> m, m, m, f, f, m, f, m, m, f, m, m, f, m..
## $ body_type              <fct> average, thin, average, skinny, thin, fi..
## $ body_type_not_perfect  <fct> TRUE, FALSE, TRUE, FALSE, FALSE, FALSE, ..
## $ diet2                  <fct> other, vegetarian, anything, anything, a..
## $ drinks                 <fct> often, socially, socially, socially, soc..
## $ drugs                  <fct> sometimes, NA, never, never, never, NA, ..
## $ religion2              <fct> atheist, NA, christian, christian, chris..
## $ education2             <fct> other, student1, degree1, high_school, s..
## $ education_kind         <fct> working, working, graduated, graduated, ..
## $ education_academic     <fct> FALSE, FALSE, TRUE, FALSE, FALSE, TRUE, ..
## $ ethnicity2             <fct> white, white, white, white, other, white..
## $ part_black             <fct> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE..
## $ part_white             <fct> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, NA, ..
## $ part_asian             <fct> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE..
## $ part_hispanic          <fct> FALSE, FALSE, FALSE, FALSE, TRUE, FALSE,..
## $ job3                   <fct> travel, student, financial, marketing, o..
## $ orientation            <fct> straight, straight, straight, straight, ..
## $ pets_has_dogs          <fct> FALSE, FALSE, FALSE, TRUE, FALSE, FALSE,..
## $ pets_has_cats          <fct> FALSE, FALSE, FALSE, FALSE, TRUE, FALSE,..
## $ pets_likes_cats        <fct> TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, NA..
## $ pets_likes_dogs        <fct> TRUE, FALSE, FALSE, FALSE, TRUE, TRUE, N..
## $ sign_fun               <fct> FALSE, FALSE, FALSE, NA, FALSE, TRUE, NA..
```

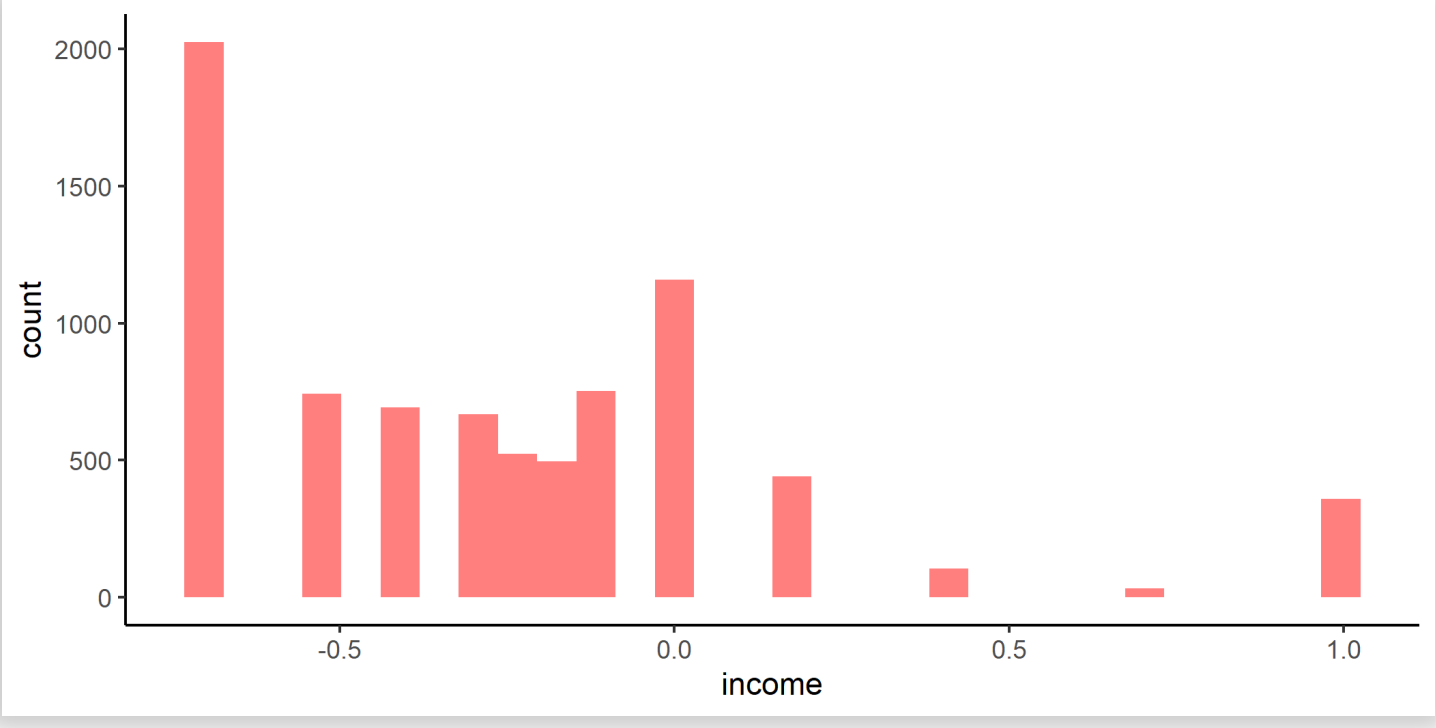
We split the data into training, validation and test sets:

```
# test_idx <- sample(1:nrow(okcupid2), 2000, replace = FALSE)
# train_idx <- okcupid2 %>% filter(!id %in% test_idx) %>% sample_
# valid_idx <- okcupid2 %>% filter(!id %in% test_idx, !id %in% tra
okcupid2 <- okcupid2 %>% select(-id)

idx <- read_rds("../data/okcupid2_idx.rda")
train_idx <- idx$train_idx
valid_idx <- idx$valid_idx
test_idx <- idx$test_idx

okcupid2_train <- okcupid2[train_idx, ]
okcupid2_valid <- okcupid2[valid_idx, ]
okcupid2_test <- okcupid2[test_idx, ]

library(glue)
glue("train no. of rows: {nrow(okcupid2_train)}
     validation no. of rows: {nrow(okcupid2_valid)}
     test no. of rows: {nrow(okcupid2_test)}")
```

```
## train no. of rows: 8000
## validation no. of rows: 1504
## test no. of rows: 2000
```

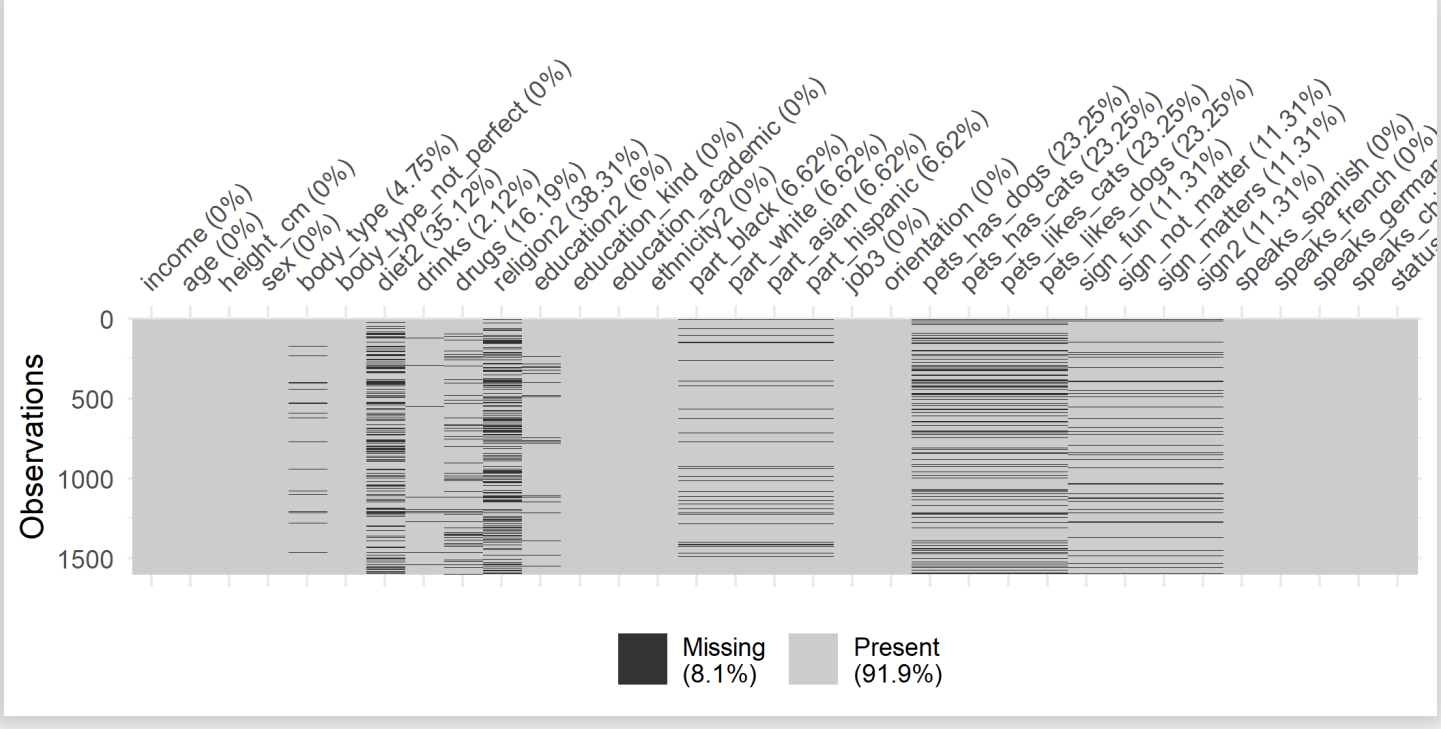
Our transformed income dependent variable behaves "ok":

```
ggplot(okcupid2_train, aes(income)) +
  geom_histogram(fill = "red", alpha = 0.5) +
  theme_classic()
```



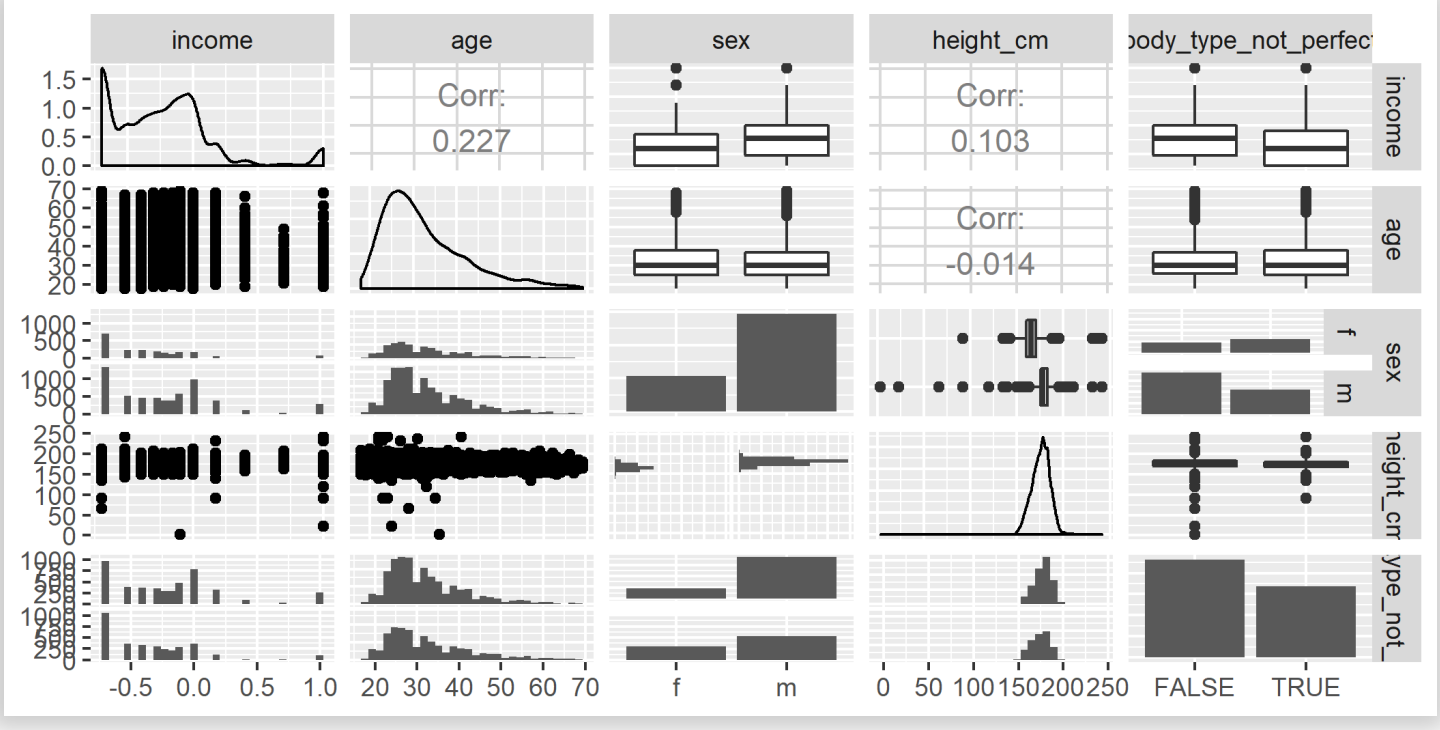
We can quickly see percentage of missing values with [naniar]():

```
library(naniar)

vis_miss(okcupid2_train %>%
           sample_frac(0.2) %>%
           select(-starts_with("essay")))
```



Also worth exploring some basic relations between predictors and income. You can use the work of others, e.g. `ggpairs`:

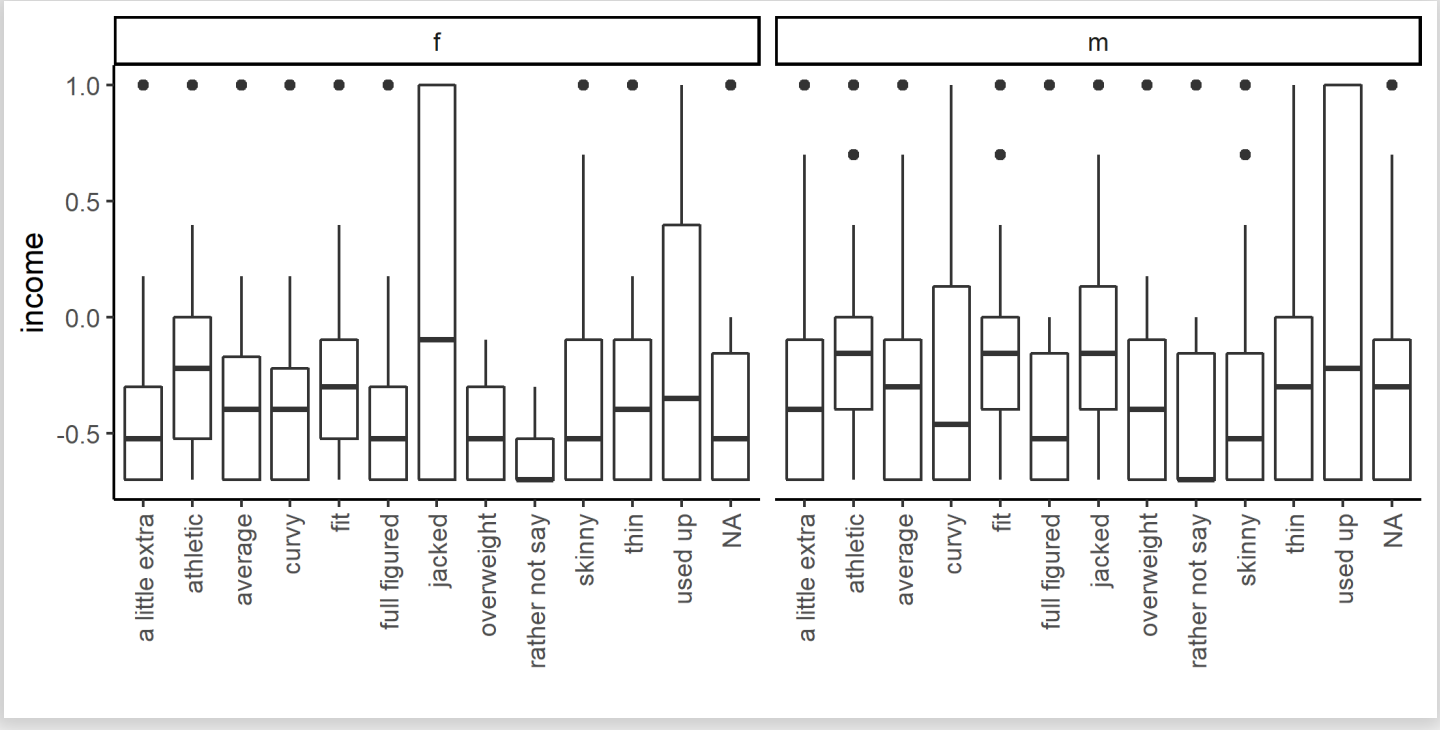
```
library(GGally)

ggpairs(okcupid2_train %>%
          select(income, age, sex, height_cm, body_type_not_perfec
```



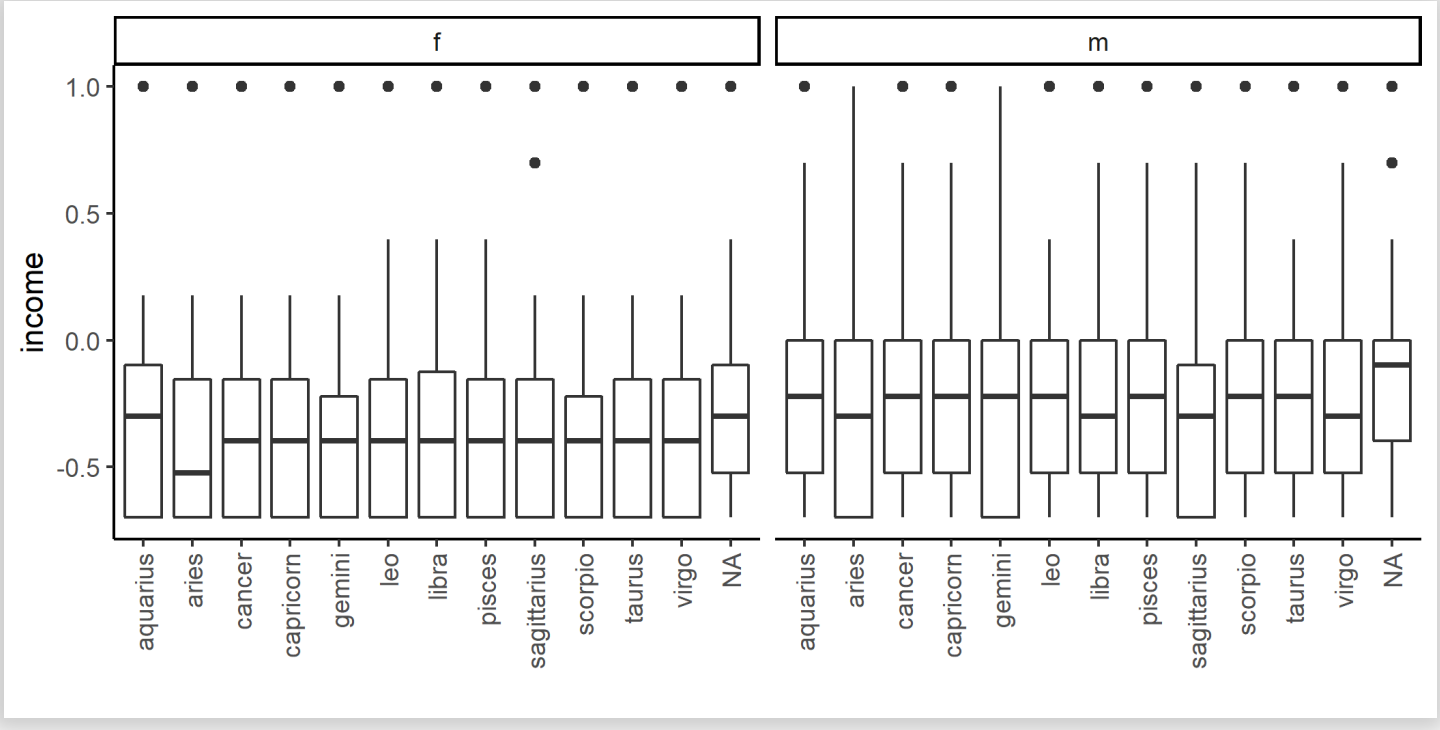
But don't be ashamed of simply exploring on your own:

```
var_vs_income_boxplot <- function(var) {
  ggplot(okcupid2_train, aes({{var}}, income)) +
  geom_boxplot() +
  facet_wrap(~ sex) +
  theme_classic() +
  labs(x = "") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust =
}
```

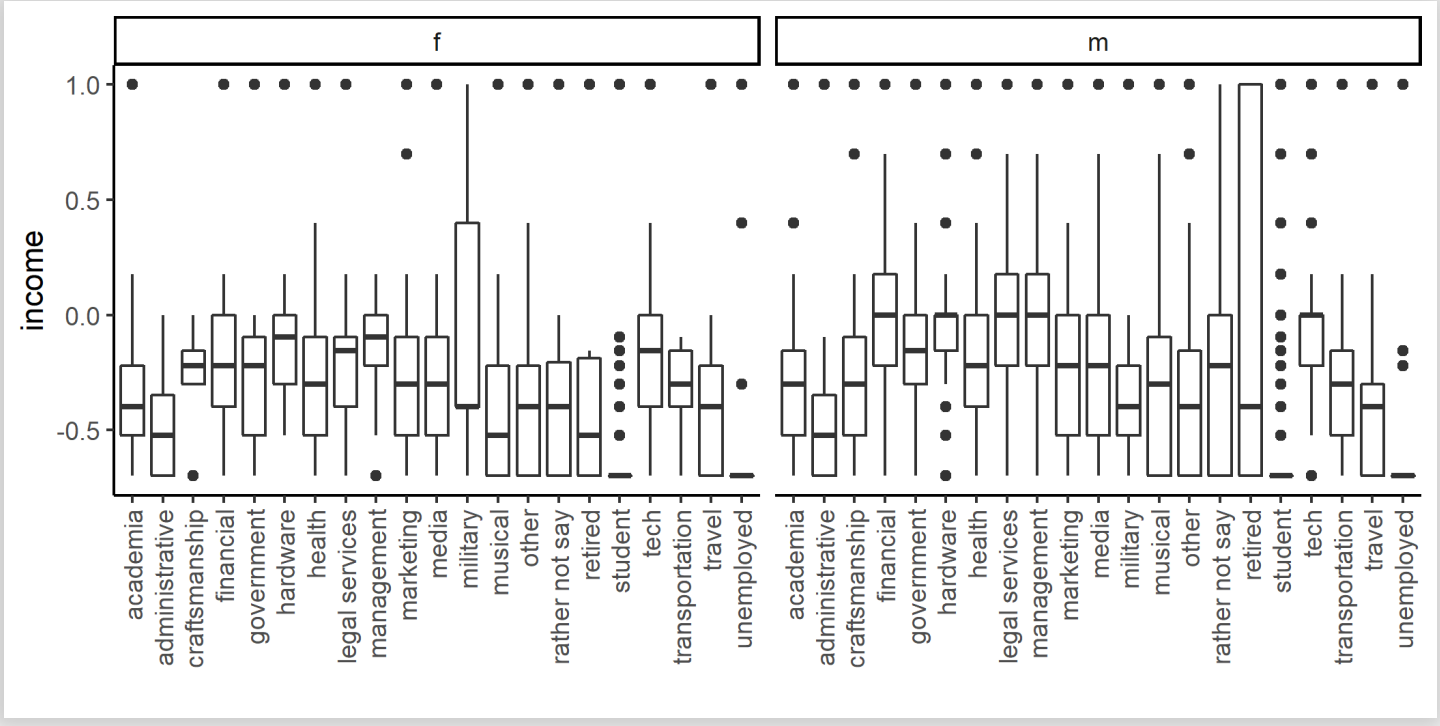
```
var_vs_income_boxplot(body_type)
```



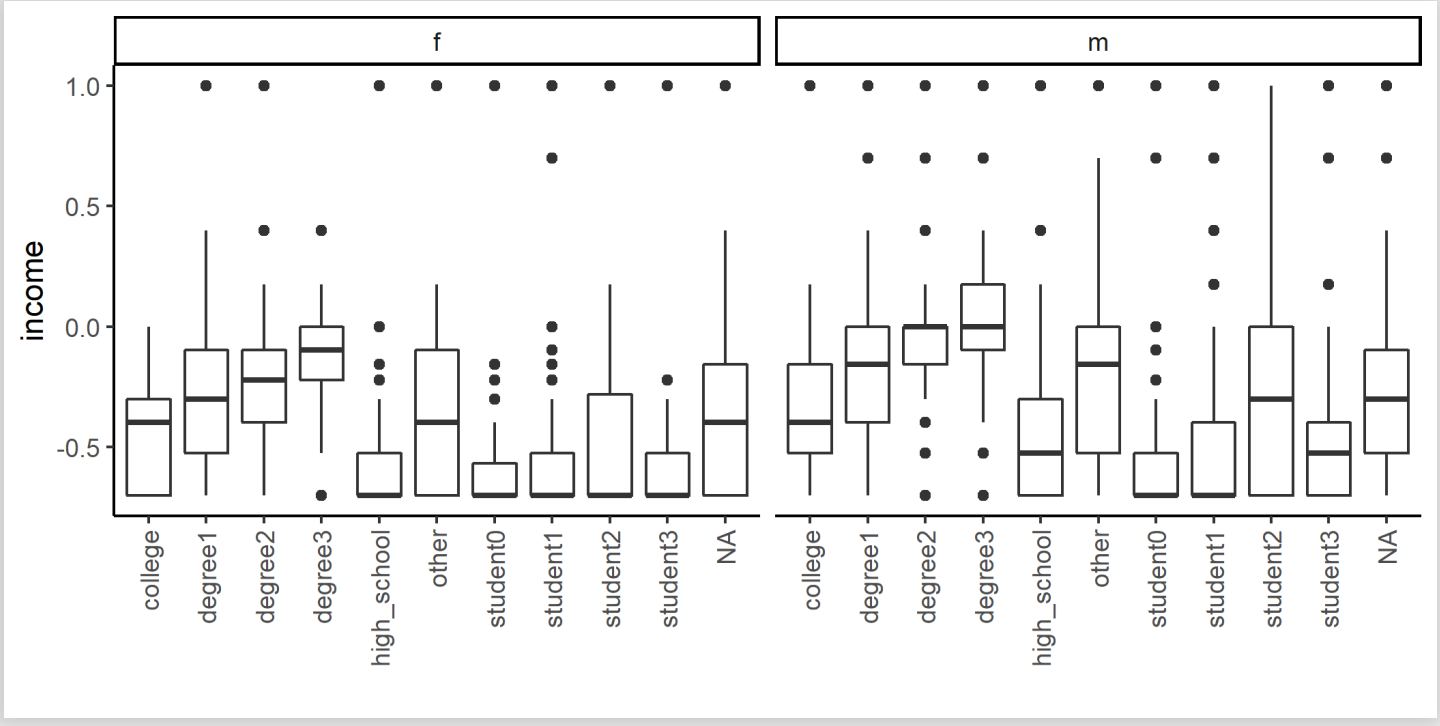
```
var_vs_income_boxplot(sign2)
```



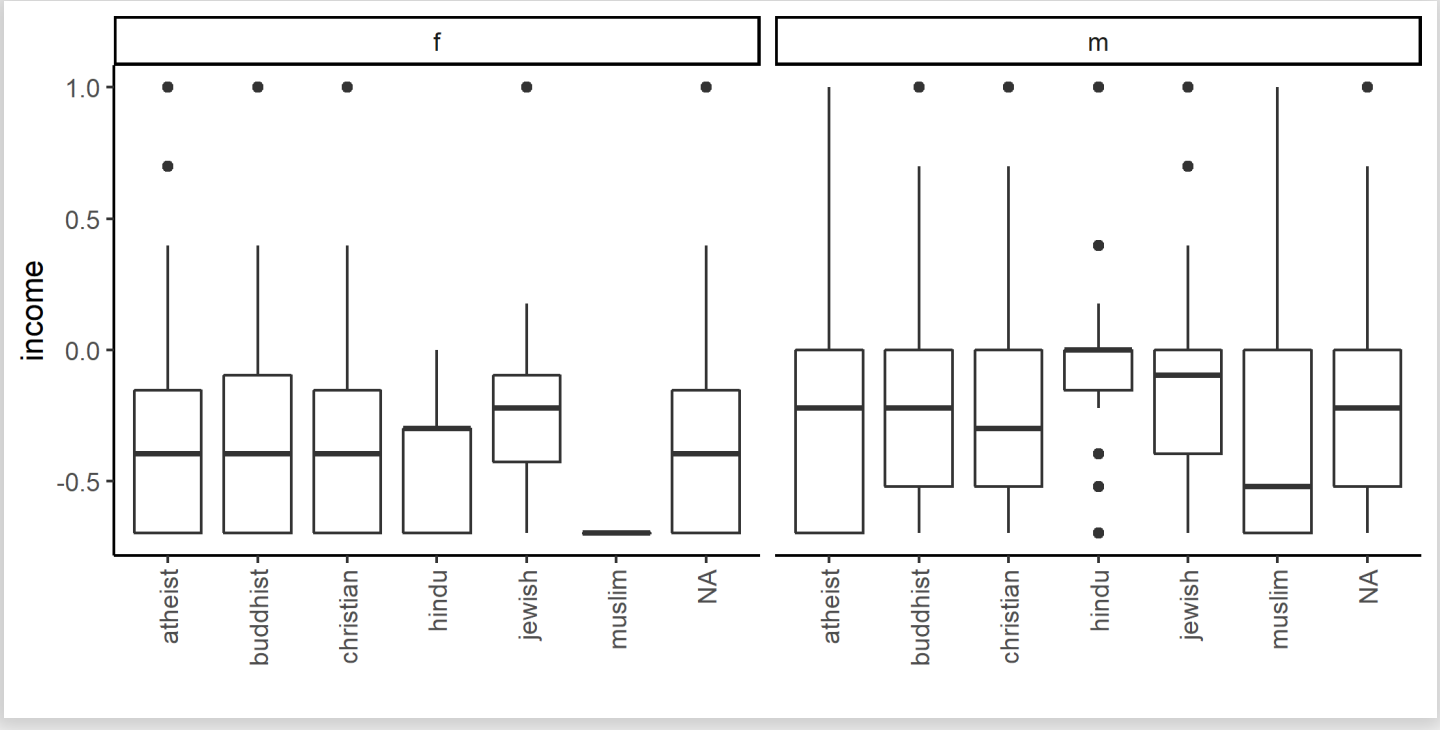
```
var_vs_income_boxplot(job3)
```



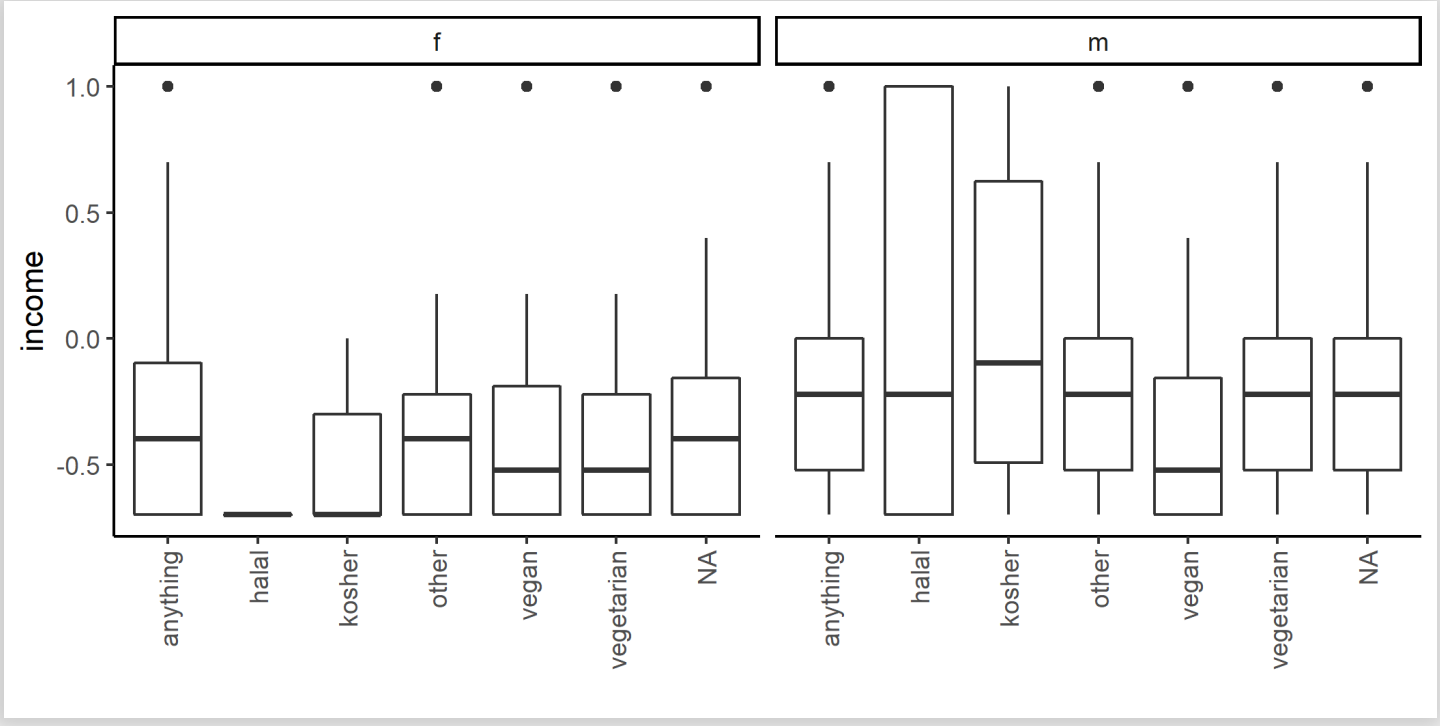
```
var_vs_income_boxplot(education2)
```



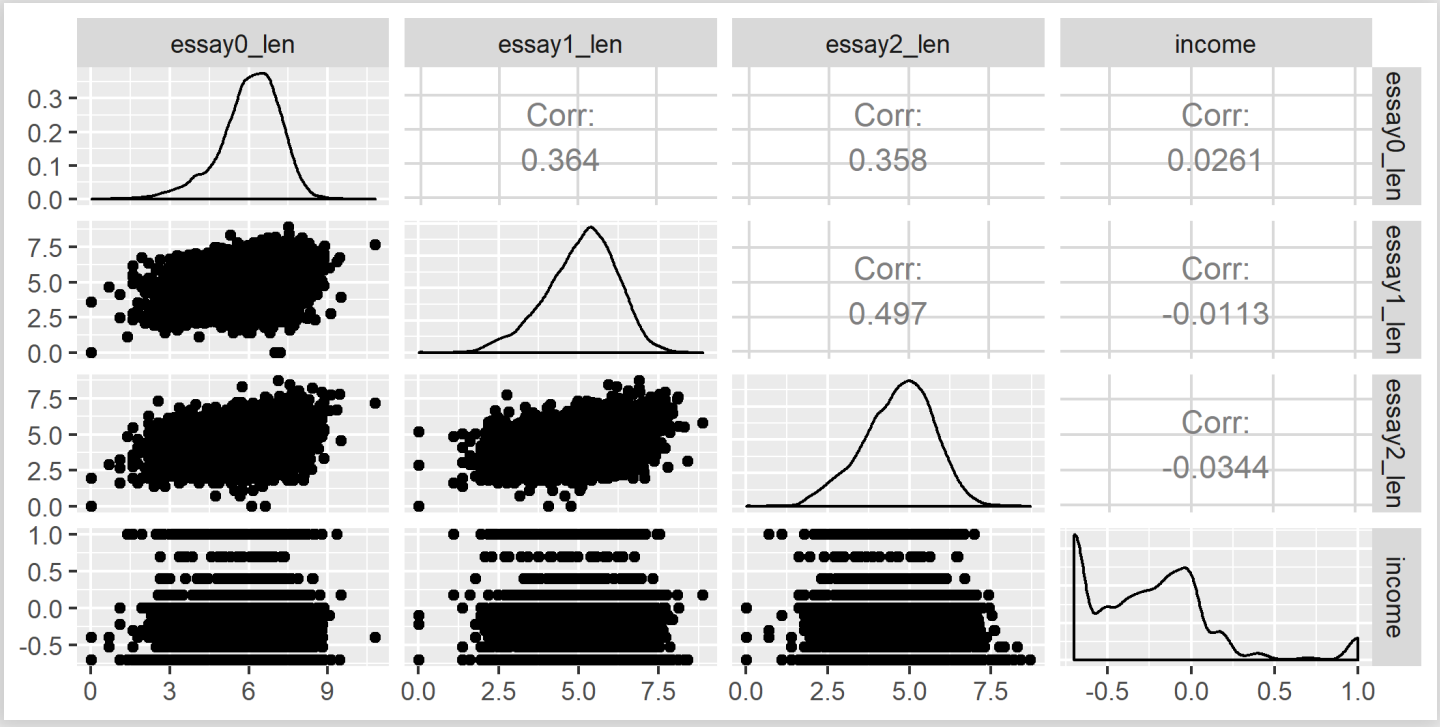
```
var_vs_income_boxplot(religion2)
```



```
var_vs_income_boxplot(diet2)
```



```
ggpairs(okcupid2_train %>%
          select(essay0_len:essay2_len, income))
```



Baseline: Linear Regression

R's `lm` function does not take `NA` values.

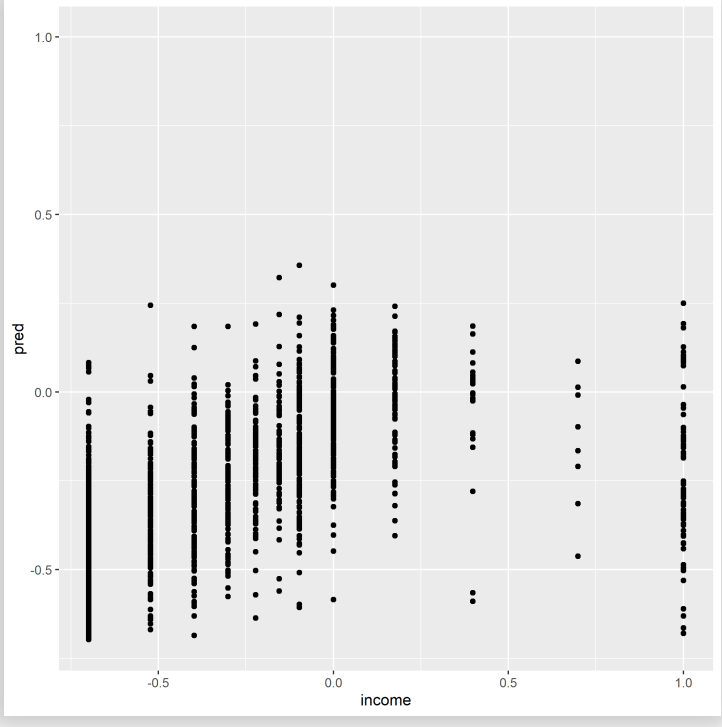
One strategy is to impute these values using a "common" value such as the median for continuous variables and mode for categorical variables. This can easily be achieved with `naniar`:

```
okcupid2_imp <- naniar::impute_median_all(okcupid2)
okcupid2_imp_train <- okcupid2_imp[train_idx, ]
okcupid2_imp_valid <- okcupid2_imp[valid_idx, ]

mod_lm <- lm(income ~ ., data = okcupid2_imp_train)
pred_lm <- predict(mod_lm, okcupid2_imp_valid)

rmse <- function(obs, pred) sqrt(mean((obs - pred)^2))

report_rmse_and_cor <- function(obs, pred) {
  RMSE <- rmse(obs, pred)
  CORR <- cor(obs, pred)
  glue("RMSE: {format(RMSE, digits = 3)}
       CORR: {format(CORR, digits = 3)}")
}
```

```
report_rmse_and_cor(okcupid2_valid$income, pred_lm)
```

```
## RMSE: 0.352
## CORR: 0.501
```

```
tibble(income = okcupid2_valid$income, pred = pred_lm) %>%
  ggplot(aes(income, pred)) + geom_point() + ylim(range(okcupid2_v
```



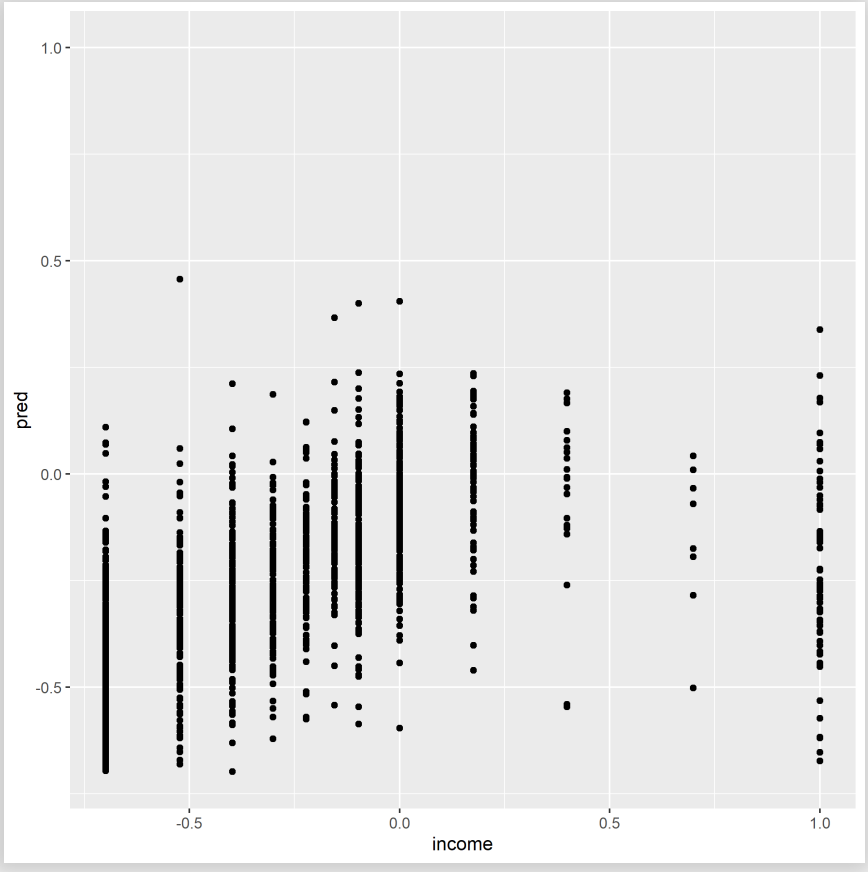
A more intelligent strategy for imputing missing values would be to *predict* them using whatever data is not missing. This can be done quite seamlessly with the `mice` package:

```
# library(mice)
# mice_obj <- mice(okcupid2, m = 1, maxit = 10, seed = 42)
# okcupid2_imp_mice <- complete(mice_obj)

okcupid2_imp_mice <- read_rds("../data/okcupid2_imp_mice.rds")
okcupid2_imp_mice_train <- okcupid2_imp_mice[train_idx, ]
okcupid2_imp_mice_valid <- okcupid2_imp_mice[valid_idx, ]

mod_lm_mice <- lm(income ~ ., data = okcupid2_imp_mice_train)
pred_lm_mice <- predict(mod_lm_mice, okcupid2_imp_mice_valid)

report_rmse_and_cor(okcupid2_valid$income, pred_lm_mice)
```

```
## RMSE: 0.351
## CORR: 0.504
```

> 💡 Can you think of other imputation strategies?

```
tibble(income = okcupid2_valid$income, pred = pred_lm_mice) %>%
  ggplot(aes(income, pred)) +
  geom_point() +
  ylim(range(okcupid2_valid$income))
```

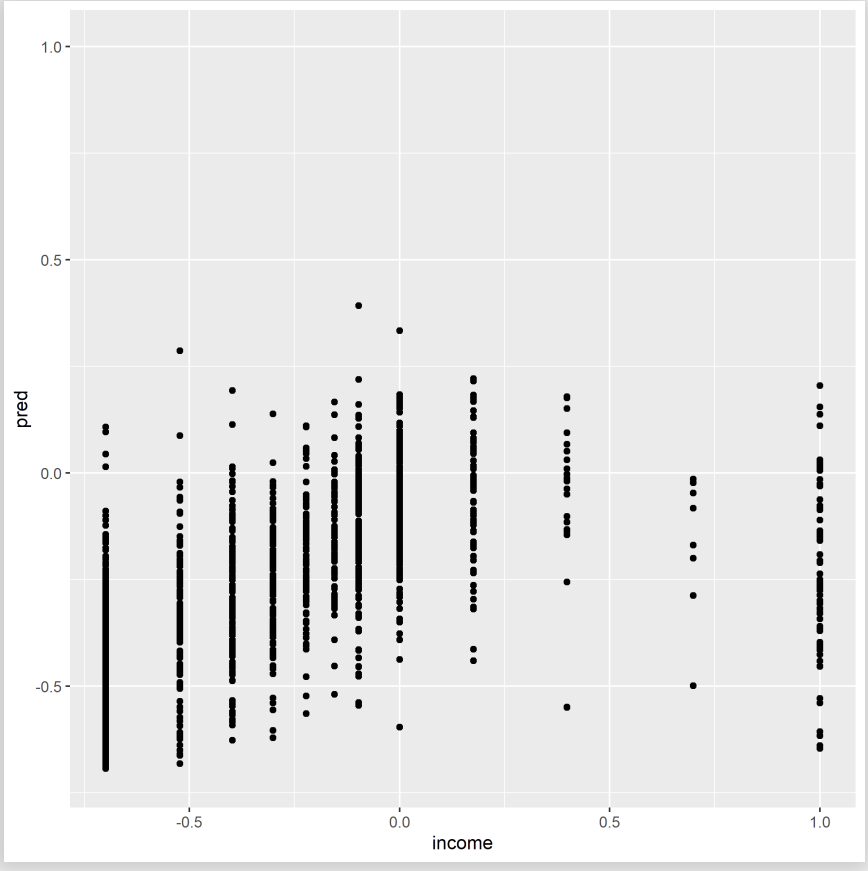


Baseline: Ridge Regression

```
library(glmnet)

okcupid2_imp_mat_train <- model.matrix( ~ ., okcupid2_imp_mice_tra
okcupid2_imp_mat_valid <- model.matrix( ~ ., okcupid2_imp_mice_val

ridge_cv <- cv.glmnet(x = okcupid2_imp_mat_train,
                      y = okcupid2_train$income, alpha = 0)

best_lambda <- ridge_cv$lambda.min

mod_lm_ridge <- glmnet(x = okcupid2_imp_mat_train,
                       y = okcupid2_train$income, alpha = 0,
                       lambda = best_lambda)

pred_lm_ridge <- predict(mod_lm_ridge, okcupid2_imp_mat_valid)

report_rmse_and_cor(okcupid2_valid$income, pred_lm_ridge)
```

```
## RMSE: 0.351
## CORR: 0.505
```

```
tibble(income = okcupid2_valid$income, pred = pred_lm_ridge) %>%
  ggplot(aes(income, pred)) +
  geom_point() +
  ylim(range(okcupid2_valid$income))
```

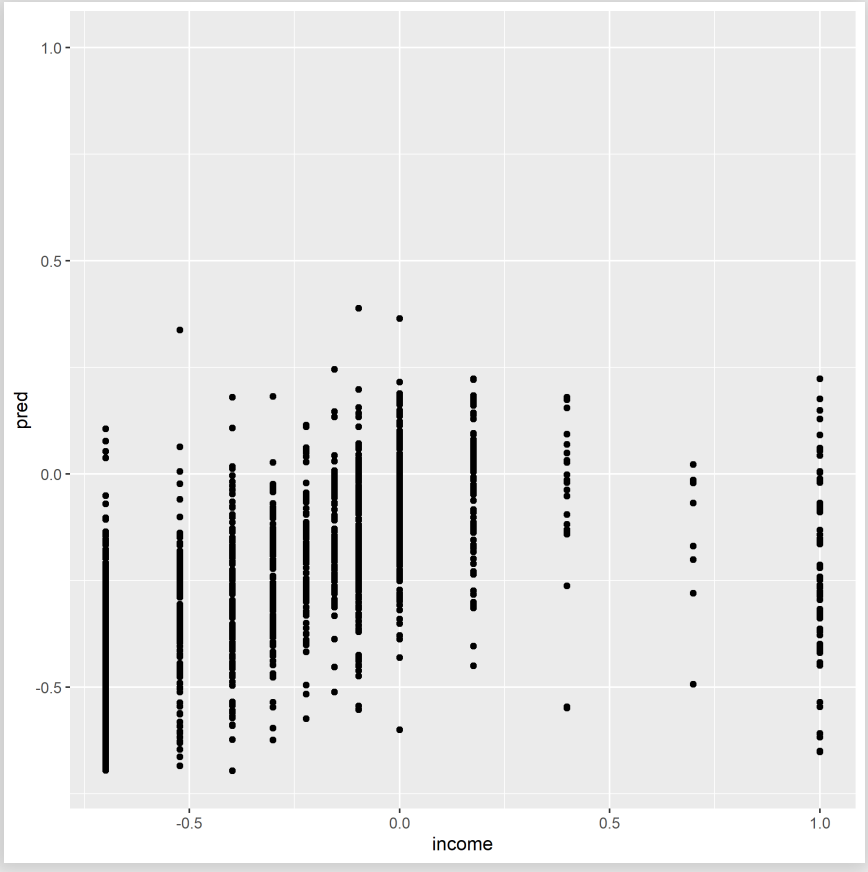


Baseline: Lasso Regression

```
lasso_cv <- cv.glmnet(x = okcupid2_imp_mat_train,
                      y = okcupid2_imp_train$income, alpha = 1)

best_lambda <- lasso_cv$lambda.min

mod_lm_lasso <- glmnet(x = okcupid2_imp_mat_train,
                       y = okcupid2_train$income, alpha = 1,
                       lambda = best_lambda)

pred_lm_lasso <- predict(mod_lm_lasso, okcupid2_imp_mat_valid)

report_rmse_and_cor(okcupid2_valid$income, pred_lm_lasso)
```

```
## RMSE: 0.351
## CORR: 0.506
```

```
tibble(income = okcupid2_valid$income, pred = pred_lm_lasso) %>%
  ggplot(aes(income, pred)) +
  geom_point() +
  ylim(range(okcupid2_valid$income))
```



End of Detour

The CART (Regression)

The OG CART

1. Find the the predictor to (binary) split on and the value of the split, by SSE criterion
2. For each resulting node if $n_{node} > 20$ go to 1
3. Once full tree has been grown, perform pruning using the *cost-complexity parameter* c_p and the SSE_{c_p} criterion
4. Predict the average value at each terminal node

- For each split *surrogate splits* are saved for future `NA`s
- An alternative criterion for complexity could be tree maximum depth (sklearn)
- One can also reduce all tree's unique paths to a set of rules
- Variables can be ranked by "importance"

What if the best split isn't binary?

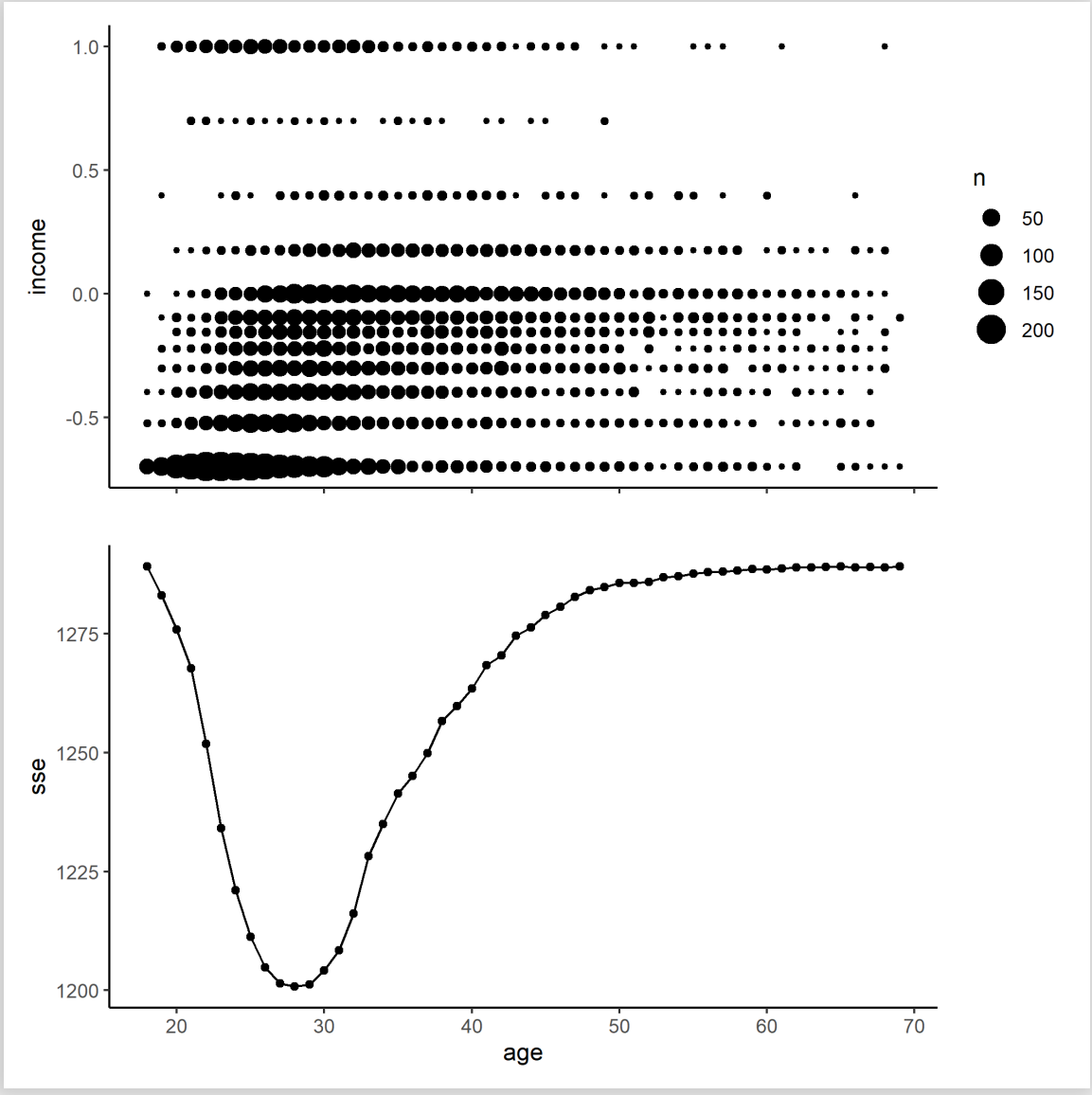
The SSE criterion - continuous predictor

- y is the continuous dependent variable
- a continuous predictor v is nominated for splitting the current node
- with splitting value l
- such that S_1 is the set of observations for which $v_i \leq l$
- and S_2 is the set of observations for which $v_i > l$
- $\bar{y}_1$ is the average of y in set S_1

$$SSE = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2$$

For example if `age` is candidate in splitting `income`:

```r
library(patchwork)

sse <- function(l, df, v) {
  income_above <- df %>% filter({{v}} >= l) %>% pull(income)
  income_below <- df %>% filter({{v}} < l) %>% pull(income)
  sse_above <- sum((income_above - mean(income_above))^2)
  sse_below <- sum((income_below - mean(income_below))^2)
  return(sse_above + sse_below)
}
age <- seq(18,69, 1)
sse_age <- map_dbl(age, sse, df = okcupid2_train, v = age)

p1 <- okcupid2_train %>%
  count(age, income) %>%
  ggplot(aes(age, income)) +
  geom_point(aes(size = n)) +
  theme_classic() +
  labs(x = "") +
  theme(axis.text.x = element_blank())

p2 <- tibble(age = age, sse = sse_age) %>%
  ggplot(aes(age, sse)) +
  geom_line() +
  geom_point() +
  theme_classic()

p1 / p2
```


1.0
0.5
0.0
-0.5
income
n
50
100
150
200
1275
1250
1225
1200
sse
20
30
40
50
60
70
age

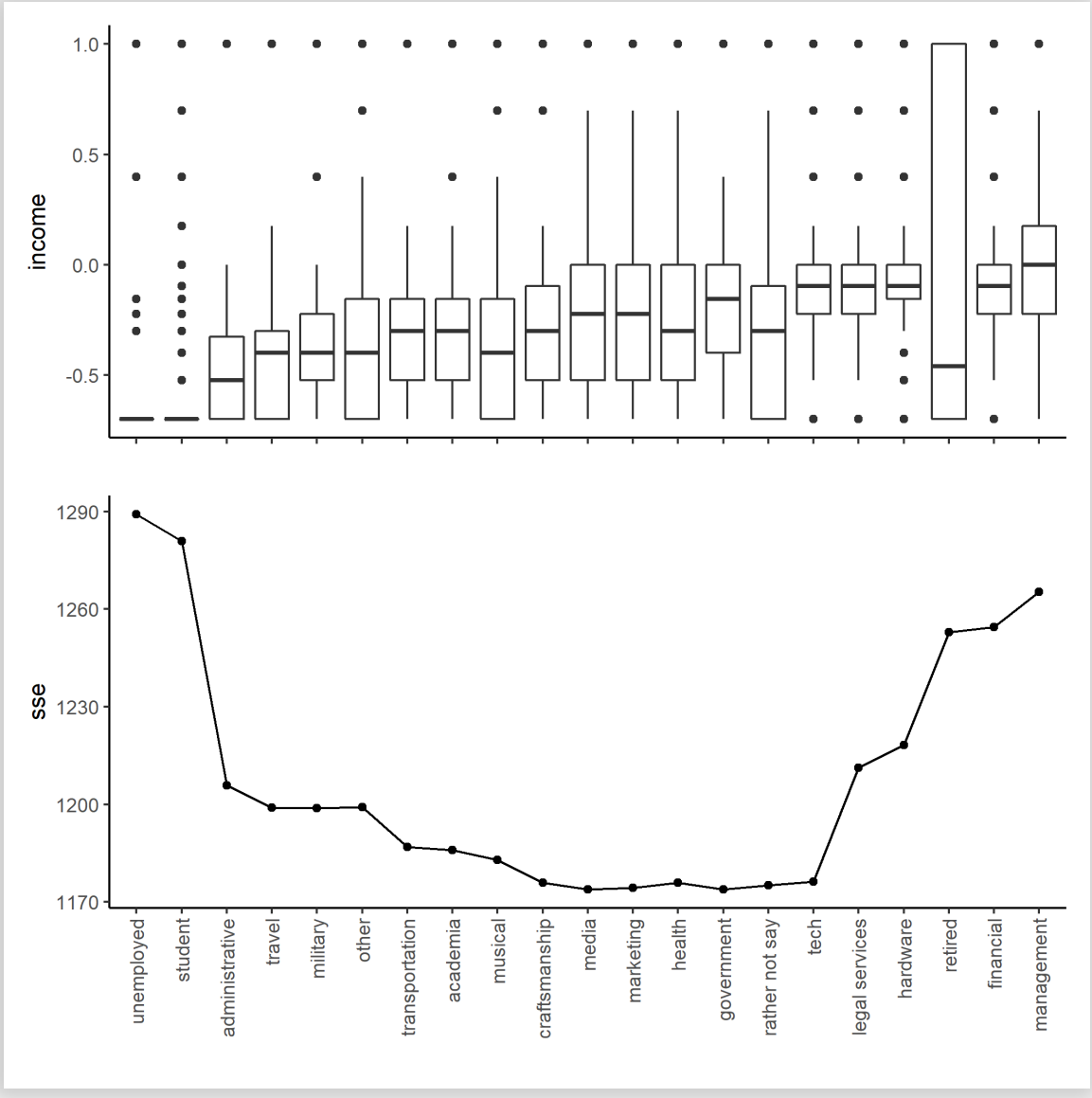
The SSE criterion - categorical predictor

> What could be an issue with a categorical variable with many levels?

- a categorical predictor l is nominated for splitting the current node
  - option 1: use dummy variables, turning each level into a 2-level 0/1 category variable
  - option 2: order levels by some criterion like $\overline{y_j}$ and treat l as continuous from here on

For example if `job3` is candidate in splitting `income`:

```
mean_income_vs_job <- okcupid2_train %>%
  group_by(job3) %>%
  summarise(mean_income = mean(income)) %>%
  arrange(mean_income)
jobs_levels_sorted <- as.character(mean_income_vs_job$job3)
okcupid2_train_job_sorted <- okcupid2_train %>%
  mutate(job_ordered = fct_relevel(job3, jobs_levels_sorted),
         job_ordered_n = as.numeric(job_ordered))

job_rank <- seq(1, length(jobs_levels_sorted), 1)
sse_job <- map_dbl(job_rank, sse, df = okcupid2_train_job_sorted,

p1 <- okcupid2_train_job_sorted %>%
  ggplot(aes(job_ordered, income)) +
  geom_boxplot() +
  theme_classic() + labs(x = "") +
  theme(axis.text.x = element_blank())

p2 <- tibble(job = factor(jobs_levels_sorted, levels = jobs_levels
  ggplot(aes(job, sse, group = 1)) +
  geom_line() +
  geom_point() +
  theme_classic() +
  labs(x = "") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust =

p1 / p2
```


income
1.0
0.5
0.0
-0.5
sse
1290
1260
1230
1200
1170
unemployed
student
administrative
travel
military
other
transportation
academia
musical
craftsmanship
media
marketing
health
government
rather not say
tech
legal services
hardware
retired
financial
management

Pruning: The SSE_{c_p} criterion

The "right-sized" tree should not be too deep to avoid overfitting.

Once the full tree is grown, we check for each split:

$$SSE_{c_p} = SSE_{tot} + c_p \cdot \#\text{terminal nodes}$$

Where c_p is a penalty or regularization parameter usually chosen by cross validation.

And we choose the smallest pruned tree with the minimum SSE_{c_p}.

CART with `rpart`

Growing the full tree (for a numeric `(y) rpart` will guess this is Regression):

```
library(rpart)

mod_tree <- rpart(income ~ ., data = okcupid2_train)
```

Pruning with some c_p:

```
mod_tree <- prune(mod_tree, cp = 0.05)
```

⚠ There is a c_p parameter you can pass to `rpart` while training like so:

```
rpart(income ~ ., data = okcupid2_train, control =
rpart.control(cp = 0.05))
```

But this will only make `rpart` consider this c_p at each split as a minimum criterion *while growing the unpruned tree*. In fact the default of this parameter is 0.01!

So, in order to train a true unpruned tree you would need to pass $c_p = 0$:

```
mod_tree <- rpart(income ~ ., data = okcupid2_train,
                  control = rpart.control(cp = 0))
```

Now. `rpart` by default will perform 10-fold Cross Validation on each split, resulting in this `cptable`:

```
head(mod_tree$cptable)
```

```
##            CP nsplit rel error    xerror       xstd
## 1 0.113597498      0 1.0000000 1.0005314 0.02239107
## 2 0.048179174      1 0.8864025 0.8887768 0.02270631
## 3 0.013140594      2 0.8382233 0.8416209 0.02353824
## 4 0.010043056      3 0.8250827 0.8320991 0.02362029
## 5 0.009147817      4 0.8150397 0.8254387 0.02379713
## 6 0.005634308      5 0.8058919 0.8169666 0.02389871
```

The nature of the `xerror` is unclear from the docs (SSE/n ?) except that it is relative to the error in the root node.

You can either use it like so:

```
best_cp <- mod_tree$cptable[which.min(mod_tree$cptable[,"xerror"]),
mod_tree <- prune(mod_tree, cp = best_cp)
```

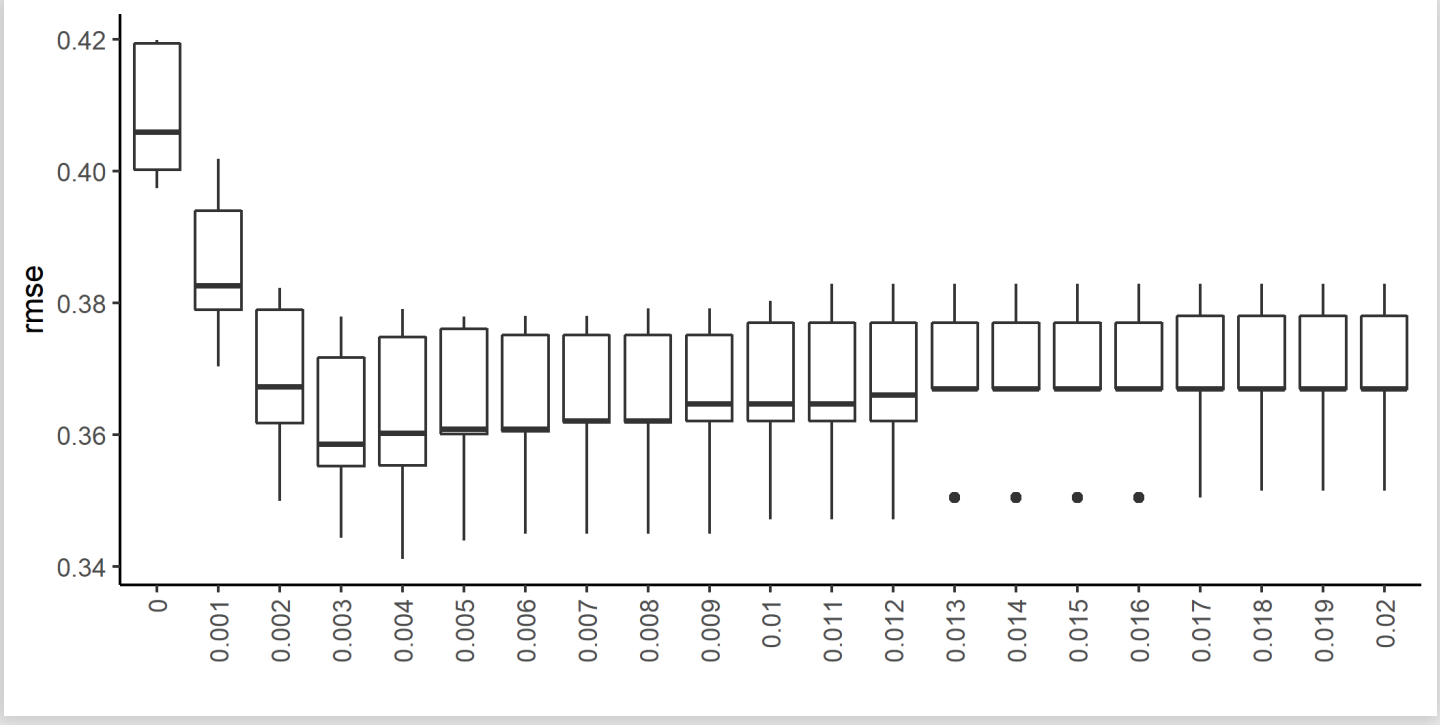
Or you can perfrom CV on your own, passing at each stage a parameter for `rpart` to not perform CV:

```
mod_tree <- rpart(income ~ ., data = okcupid2_train,
                  control = rpart.control(cp = 0), xval = 1)
```

Let's tune c_p for our data using a 5-fold (manual) Cross Validation. The criterion to maximize would be RMSE.

```
n_cv <- 5; cp_seq <- seq(0, 0.02, 0.001)

okcupid2_train_val <- okcupid2_train %>%
  mutate(val = sample(1:n_cv, n(), replace = TRUE))

get_validation_set_rmse <- function(i, .cp) {
  ok_tr <- okcupid2_train_val %>% filter(val != i) %>% select(-val
  ok_val <- okcupid2_train_val %>% filter(val == i) %>% select(-va
  mod <- rpart(income ~ ., data = ok_tr,
               control = rpart.control(cp = 0, xval = 1))
  mod <- prune(mod, cp = .cp)
  pred <- predict(mod, ok_val)
  rmse(ok_val$income, pred)
}

get_cv_rmse <- function(.cp) {
  tibble(cp = rep(.cp, n_cv),
         rmse = map_dbl(1:n_cv, get_validation_set_rmse, .cp = .cp
}
```

```
cv_table <- map_dfr(cp_seq, get_cv_rmse)

cv_table %>%
  mutate(cp = factor(cp)) %>%
  ggplot(aes(cp, rmse)) +
  geom_boxplot() +
  theme_classic() +
  labs(x = "") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust =
```



Training on the entire training set:

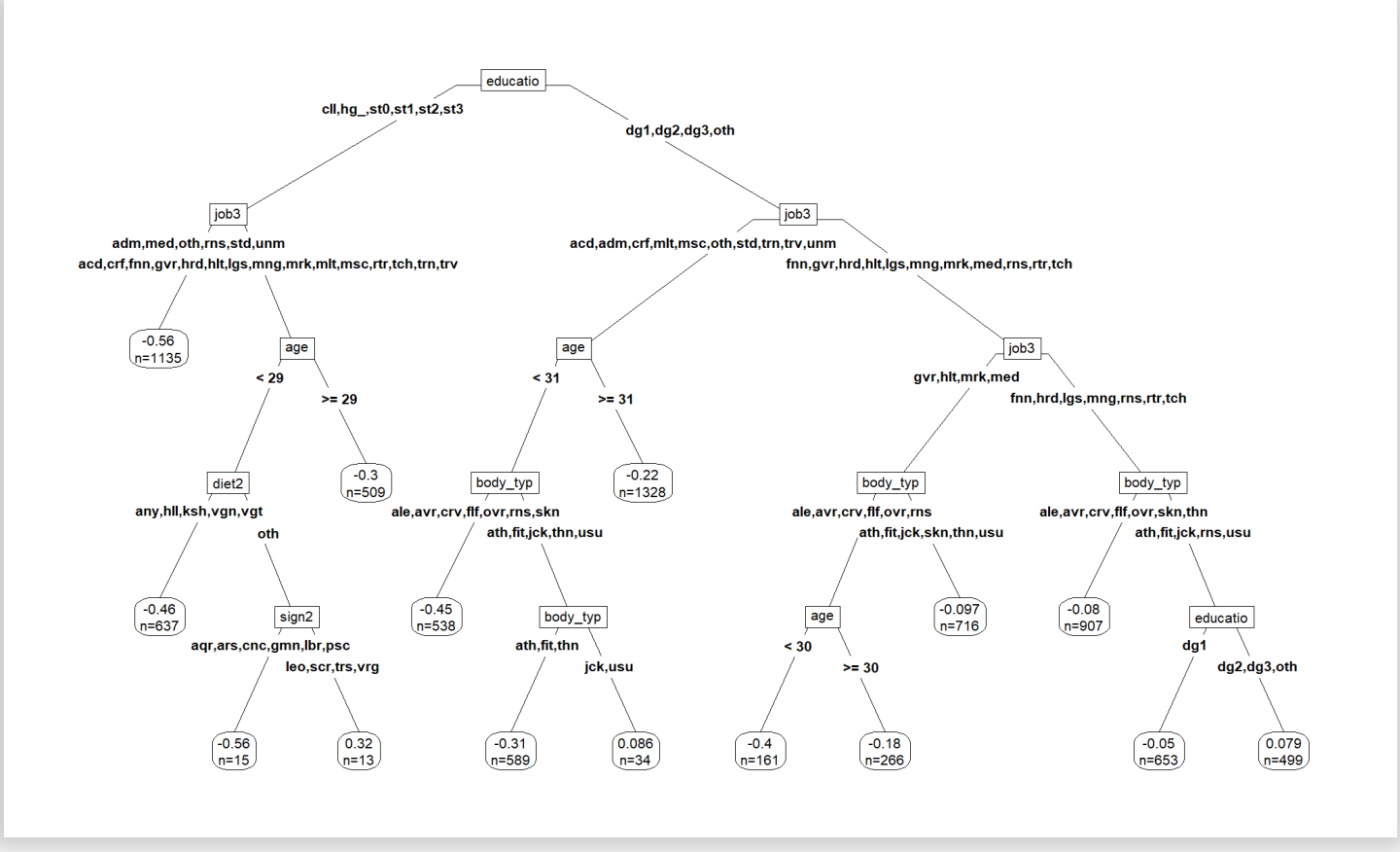
```
mod_tree_na <- rpart(income ~ ., data = okcupid2_train,
                     control = rpart.control(cp = 0, xval = 1))
mod_tree_na <- prune(mod_tree_na, cp = 0.003)
```

You can plot the tree using `rpart`, but...

```
plot(mod_tree_na)
text(mod_tree_na, pretty = 1, use.n = TRUE)
```

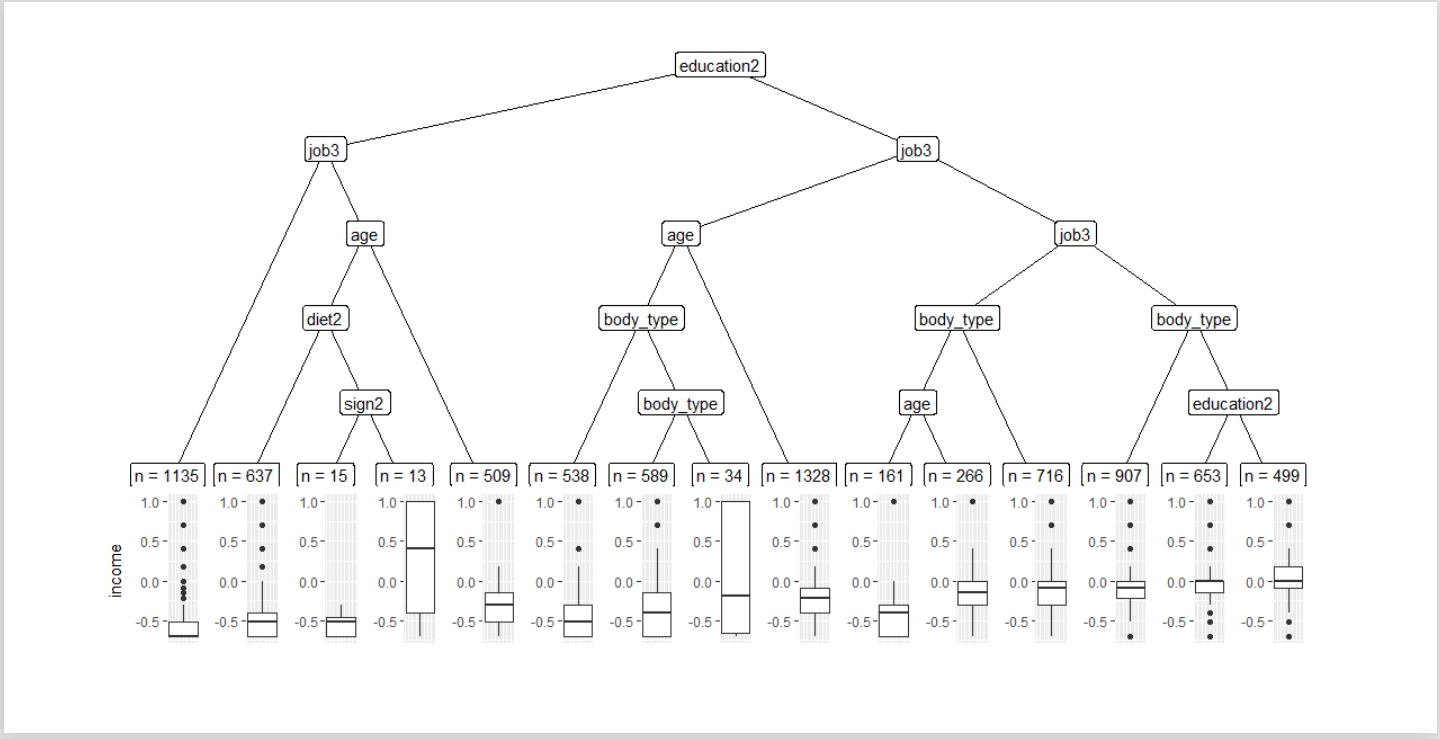
The plotting function in the `rpart.plot` package is slightly nicer:

```
library(rpart.plot)
prp(mod_tree_na, type = 5, extra = 1)
```



You can go fancy and use `partykit` and/or `ggparty`:

```
library(ggparty)

party_obj <- as.party(mod_tree_na)

ggparty(party_obj) +
  geom_edge() +
  # geom_edge_label() +
  geom_node_label(aes(label = splitvar), ids = "inner") +
  geom_node_label(aes(label = str_c("n = ", nodesize)),
                  ids = "terminal", nudge_y = 0.02) +
  geom_node_plot(gglist = list(geom_boxplot(aes(y = income)),
                               theme(axis.text.x=element_blank()
                                     axis.ticks.x=element_blank
                 shared_axis_labels=TRUE)
```


education2
job3
job3
age
age
job3
diet2
body_type
body_type
body_type
sign2
body_type
age
education2
n = 1135
n = 637
n = 15
n = 13
n = 509
n = 538
n = 589
n = 34
n = 1328
n = 161
n = 266
n = 716
n = 907
n = 653
n = 499
income
1.0
0.5
0.0
-0.5

Or you can just print the tree and see what is going on:

```
print(party_obj)
```

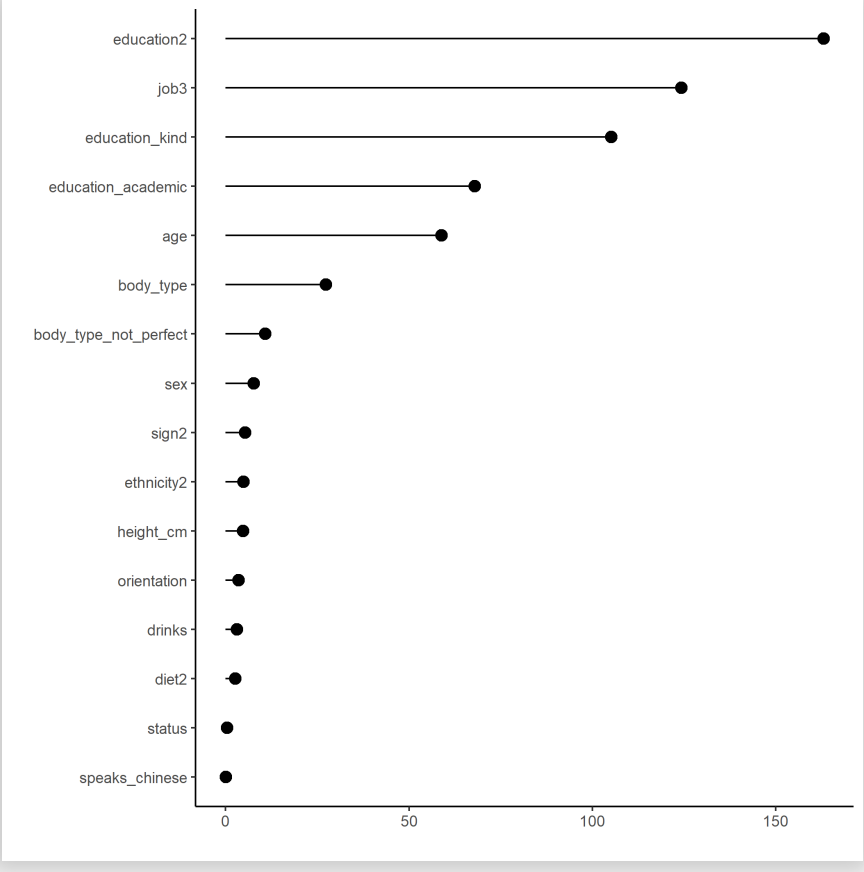
```
Fitted party:
[1] root
|   [2] education2 in college, high_school, student0, student1, student2, student3
|   |   [3] job3 in administrative, media, other, rather not say, student, unemployed: -0.555 (n = 1135, err = 126.9)
|   |   [4] job3 in academia, craftsmanship, financial, government, hardware, health, legal services, management, marketing, militar
|   |   |   [5] age < 28.5
|   |   |   |   [6] diet2 in anything, halal, kosher, vegan, vegetarian: -0.460 (n = 637, err = 84.2)
|   |   |   |   [7] diet2 in other
|   |   |   |   |   [8] sign2 in aquarius, aries, cancer, gemini, libra, pisces: -0.559 (n = 15, err = 0.3)
|   |   |   |   |   [9] sign2 in leo, scorpio, taurus, virgo: 0.316 (n = 13, err = 5.9)
|   |   |   [10] age >= 28.5: -0.301 (n = 509, err = 53.0)
|   [11] education2 in degree1, degree2, degree3, other
|   |   [12] job3 in academia, administrative, craftsmanship, military, musical, other, student, transportation, travel, unemployed
|   |   |   [13] age < 30.5
|   |   |   |   [14] body_type in a little extra, average, curvy, full figured, overweight, rather not say, skinny: -0.450 (n = 538,
|   |   |   |   [15] body_type in athletic, fit, jacked, thin, used up
|   |   |   |   |   [16] body_type in athletic, fit, thin: -0.314 (n = 589, err = 141.6)
|   |   |   |   |   [17] body_type in jacked, used up: 0.086 (n = 34, err = 16.1)
|   |   |   [18] age >= 30.5: -0.221 (n = 1328, err = 171.8)
|   |   [19] job3 in financial, government, hardware, health, legal services, management, marketing, media, rather not say, retired,
|   |   |   [20] job3 in government, health, marketing, media
|   |   |   |   [21] body_type in a little extra, average, curvy, full figured, overweight, rather not say
|   |   |   |   |   [22] age < 29.5: -0.402 (n = 161, err = 13.7)
|   |   |   |   |   [23] age >= 29.5: -0.178 (n = 266, err = 17.3)
|   |   |   |   [24] body_type in athletic, fit, jacked, skinny, thin, used up: -0.097 (n = 716, err = 95.6)
|   |   |   [25] job3 in financial, hardware, legal services, management, rather not say, retired, tech
|   |   |   |   [26] body_type in a little extra, average, curvy, full figured, overweight, skinny, thin: -0.080 (n = 907, err = 83.
|   |   |   |   [27] body_type in athletic, fit, jacked, rather not say, used up
|   |   |   |   |   [28] education2 in degree1: -0.050 (n = 653, err = 48.7)
|   |   |   |   |   [29] education2 in degree2, degree3, other: 0.079 (n = 499, err = 51.3)

Number of inner nodes:    14
Number of terminal nodes: 15
```

Variables Importance

Summing the reduction in SSE for each split variable, we can get a measure of importance.

Unfortunately in `rpart` the "potential" reduction in `(SSE)` for surrogate splits is also summed. You can either ignore or retrain with only the variables chosen by the model.

```
enframe(mod_tree_na$variable.importance) %>%
  arrange(value) %>%
  mutate(variable = as_factor(name)) %>%
  ggplot(aes(variable, value)) +
  geom_segment(aes(x = variable, xend = variable,
                   y = 0, yend = value)) +
  geom_point(size = 3) +
  theme_classic() +
  coord_flip() +
  labs(x = "", y = "")
```



How would this profile look for a couple of very correlated predictors?

Prediction

```
pred_tree_na <- predict(mod_tree_na, okcupid2_valid)

report_rmse_and_cor(okcupid2_valid$income, pred_tree_na)
```

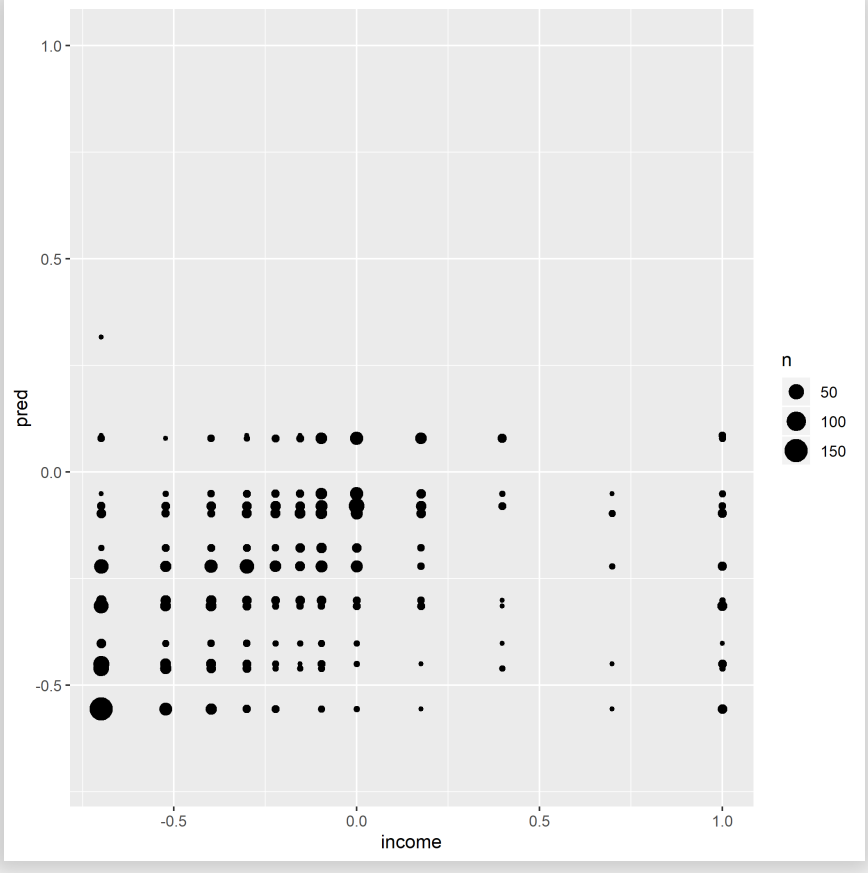
```
## RMSE: 0.363
## CORR: 0.449
```

As expected, far from impressive. Let's try using the imputed `NA` data:

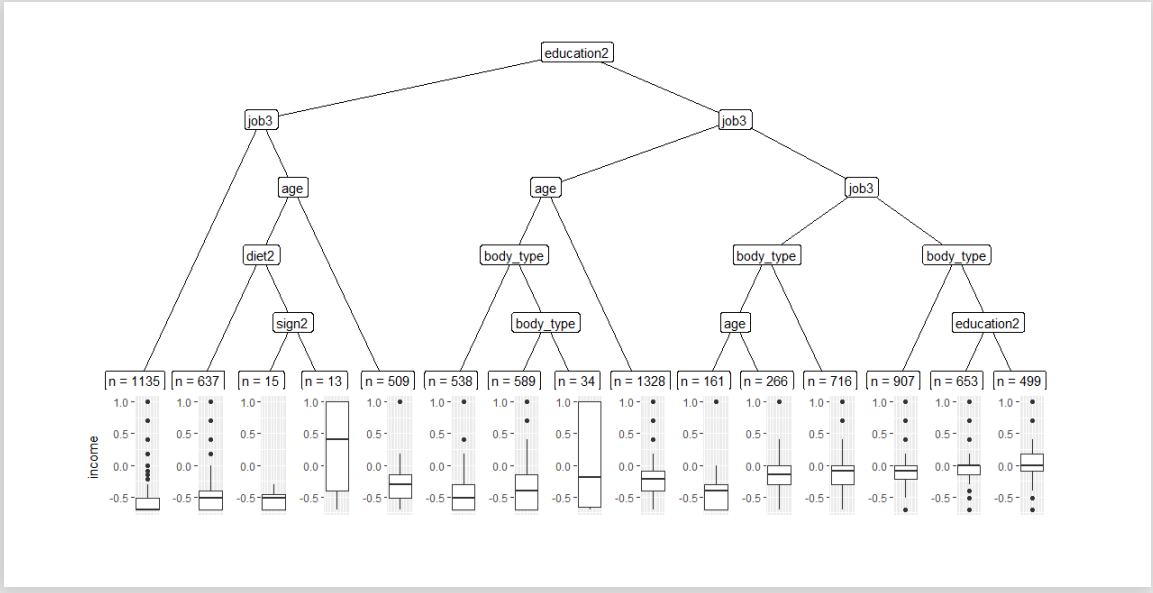
```
mod_tree_imp <- rpart(income ~ ., data = okcupid2_imp_mice_train,
                      control = rpart.control(cp = 0, xval = 1))
mod_tree_imp <- prune(mod_tree_imp, cp = 0.003)
pred_tree_imp <- predict(mod_tree_imp, okcupid2_imp_mice_valid)

report_rmse_and_cor(okcupid2_valid$income, pred_tree_imp)
```

```
## RMSE: 0.369
## CORR: 0.424
```

```
tibble(income = okcupid2_valid$income, pred = pred_tree_na) %>%
  count(income, pred) %>%
  ggplot(aes(income, pred)) +
  geom_point(aes(size = n)) +
  ylim(range(okcupid2_valid$income))
```



The Others



Looking at this one might wonder:

- Since when do we just split, how about a statistical test?
- Predicting a single value for each terminal node? Is that the best we can do?
- Many variables repeat in the same path, can we compact paths into simple rules?

Conditional Inference Trees

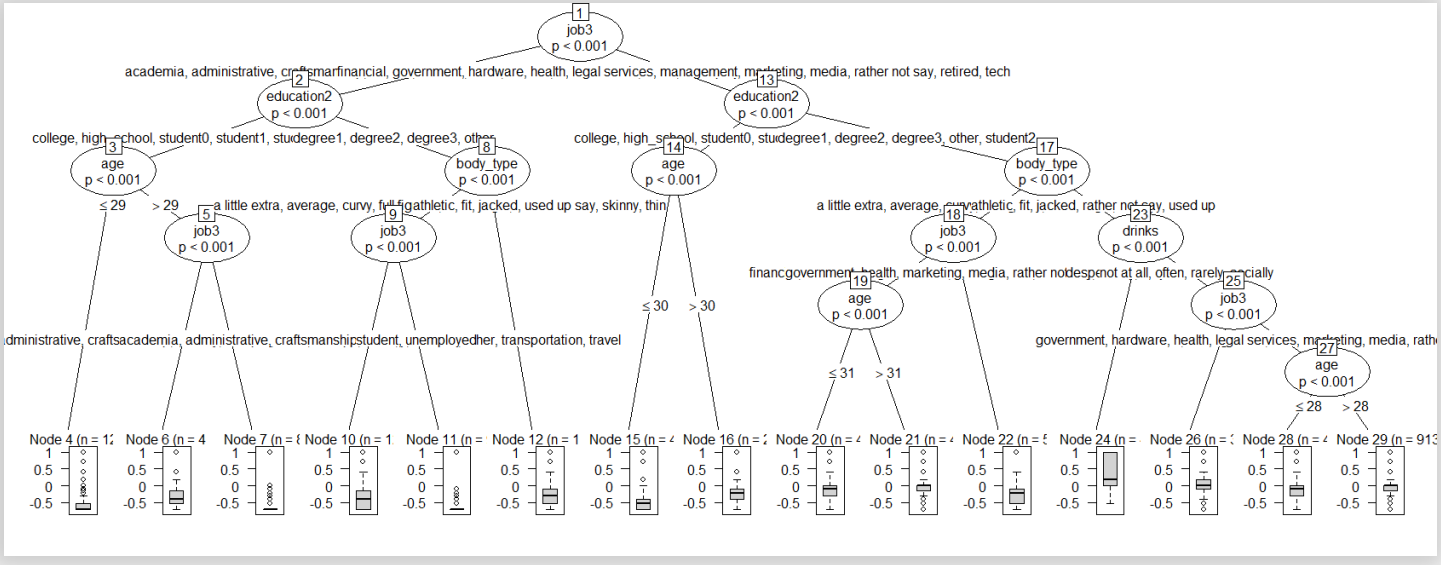
Hothorn et. al. (2006) perform a statistical hypothesis test for each variable to decide whether it is related to y (including multiple comparisons adjustment).

If no variable passes test - stop. No pruning necessary. If a few variable pass - choose the most related, e.g. using a p-value.

```
library(partykit)

mod_ctree_na <- ctree(income ~ ., data = okcupid2_train, alpha = (
pred_ctree_na <- predict(mod_ctree_na, okcupid2_valid)
report_rmse_and_cor(okcupid2_valid$income, pred_ctree_na)
```

```
## RMSE: 0.362
## CORR: 0.454
```

```
plot(mod_ctree_na)
```



Model Trees

Quinlan (1992) and later Wang and Witten (1997) made a few changes to the beloved CART:

- instead of single values at terminal nodes, M5 trees have *linear models*
- sort of a piecewise linear regression only the "pieces" can come from many variables
- can extrapolate to never before seen values (not necessarily good)

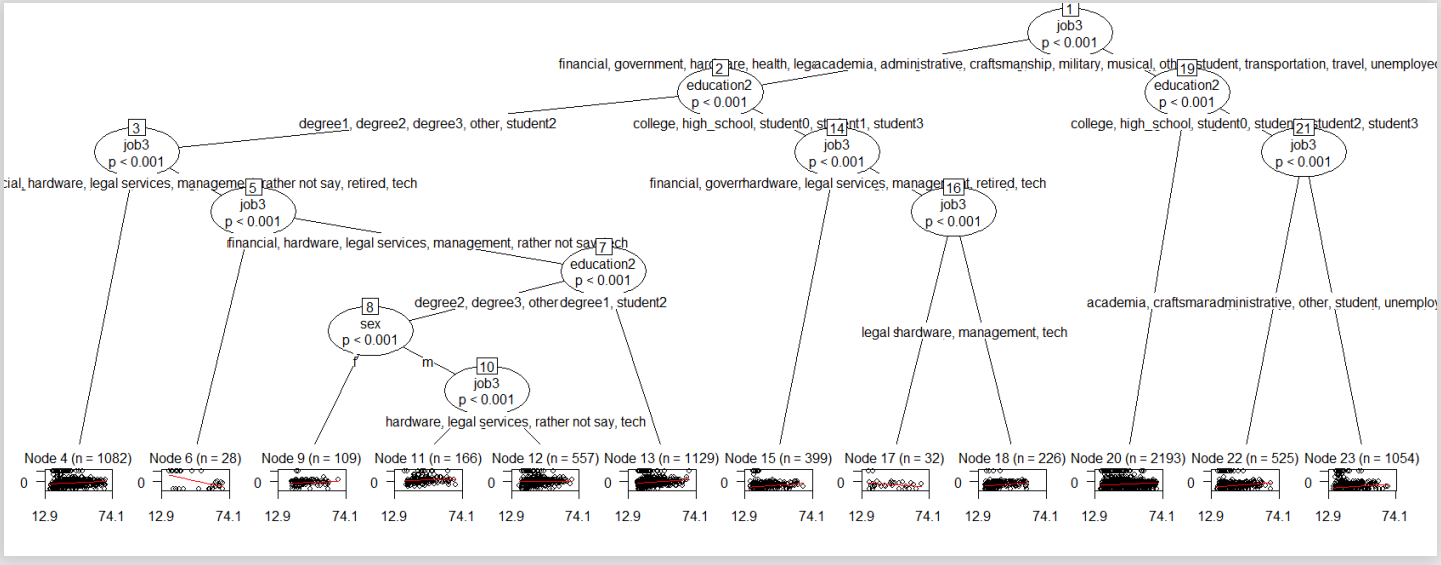
You can find an implementation in R in the `RWeka` package, function `M5P()`.

Zeileis et. al. (2008) approach is slightly simpler to grasp. It allows you to specify variables for splitting and variables for regression.

See `lmtree()` and `glmtree()` functions in the `partykit`

```
mod_lmtree_na <- lmtree(income ~ age | sex + job3 + education2, da

pred_lmtree_na <- predict(mod_lmtree_na, okcupid2_valid)

report_rmse_and_cor(okcupid2_valid$income, pred_lmtree_na)

plot(mod_lmtree_na)
```

```
## RMSE: 0.366
## CORR: 0.437
```



Rule-Based Trees

Holmes et. a. (1993) general approach:

1. Build a model tree (e.g. M5)
2. For each path compute the coverage (= % of relevant observations to that path)
3. Compact the path with largest coverage into a rule
4. Remove previous relevant observations and go back to 1 until all observations are accounted are.

```
library(RWeka)

# limiting predictors just for a nice print!
mod_m5rules_na <- M5Rules(income ~ ., data = okcupid2_train[, c("i
pred_m5rules_na <- predict(mod_m5rules_na, okcupid2_valid)
report_rmse_and_cor(okcupid2_valid$income, pred_m5rules_na)
```

```
## RMSE: 0.374
## CORR: 0.392
```

```
print(mod_m5rules_na)
```

```
M5 pruned model rules
(using smoothed linear models) :
Number of Rules : 2

Rule: 1
IF
        education2=student2,degree1,other,degree2,degree3 > 0.5
        age > 30.5
THEN

income =
        -0.0015 * age
        + 0.1123 * sex=m
        + 0.0003 * education2=student1,high_school,student3,college,student2,degree1,other,degree2,degree3
        + 0.0002 * education2=high_school,student3,college,student2,degree1,other,degree2,degree3
        + 0.0002 * education2=student2,degree1,other,degree2,degree3
        + 0.185 * education2=degree1,other,degree2,degree3
        + 0.0003 * education2=other,degree2,degree3
        + 0.0672 * education2=degree2,degree3
        + 0.0755 * education2=degree3
        - 0.3345 [2977/81.357%]

Rule: 2

income =
        0.009 * age
        + 0.1196 * sex=m
        + 0.0927 * education2=student1,high_school,student3,college,student2,degree1,other,degree2,degree3
        - 0.0384 * education2=high_school,student3,college,student2,degree1,other,degree2,degree3
        + 0.0505 * education2=student3,college,student2,degree1,other,degree2,degree3
        + 0.212 * education2=student2,degree1,other,degree2,degree3
        + 0.111 * education2=degree2,degree3
        - 0.8955 [4523/93.176%]
```

Bagged Trees

Bagging: The Gist

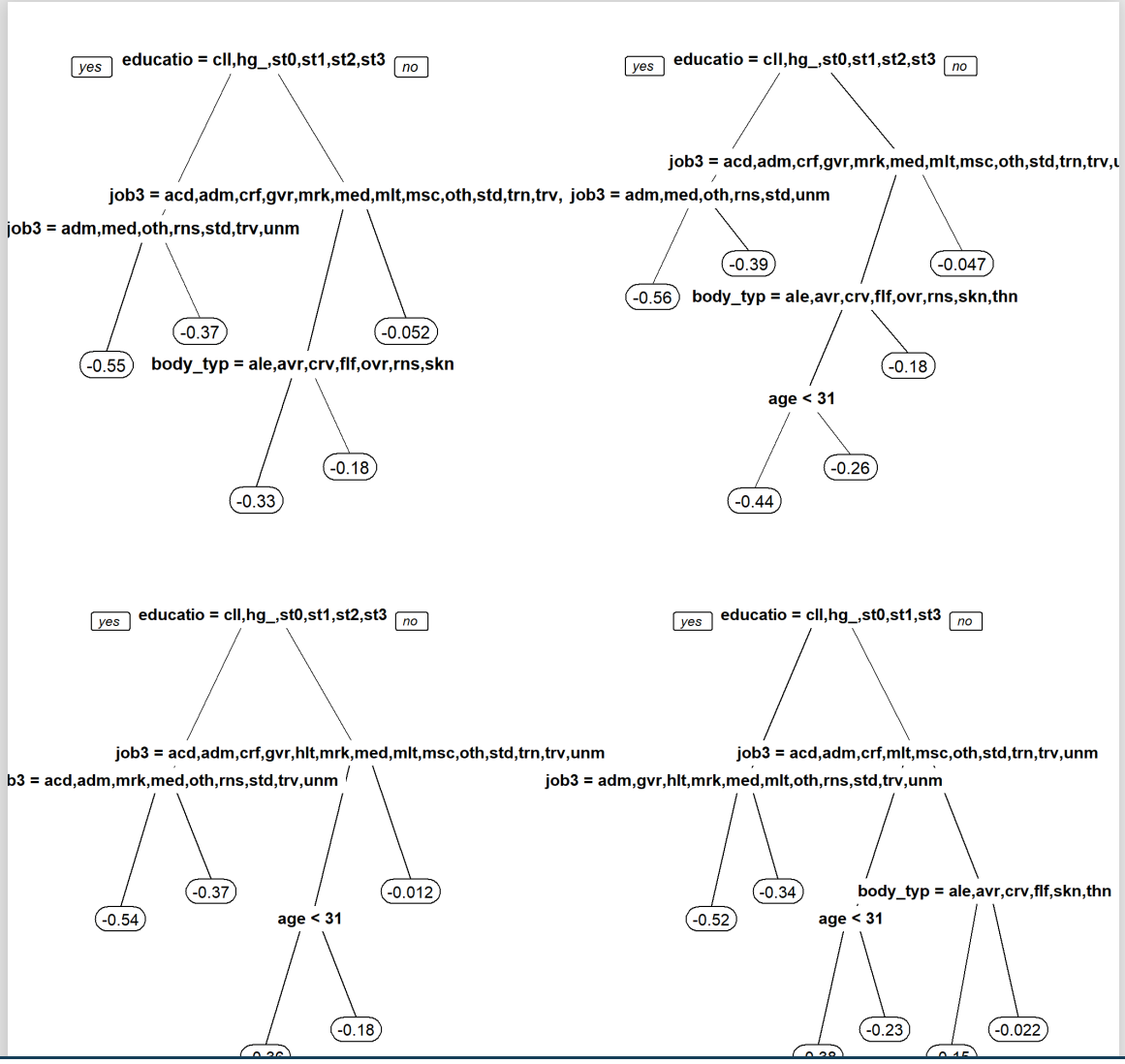
For $j = 1$ to m do

- Generate a bootstrap sample of the original data
- Train an unpruned tree model on this sample

End.

- Predict average through all m trees (or %vote for each class in classification)
- What you get: better prediction, *OOB* error estimates
- What you lose: interpretability, speed (but bagging can be paralleled)

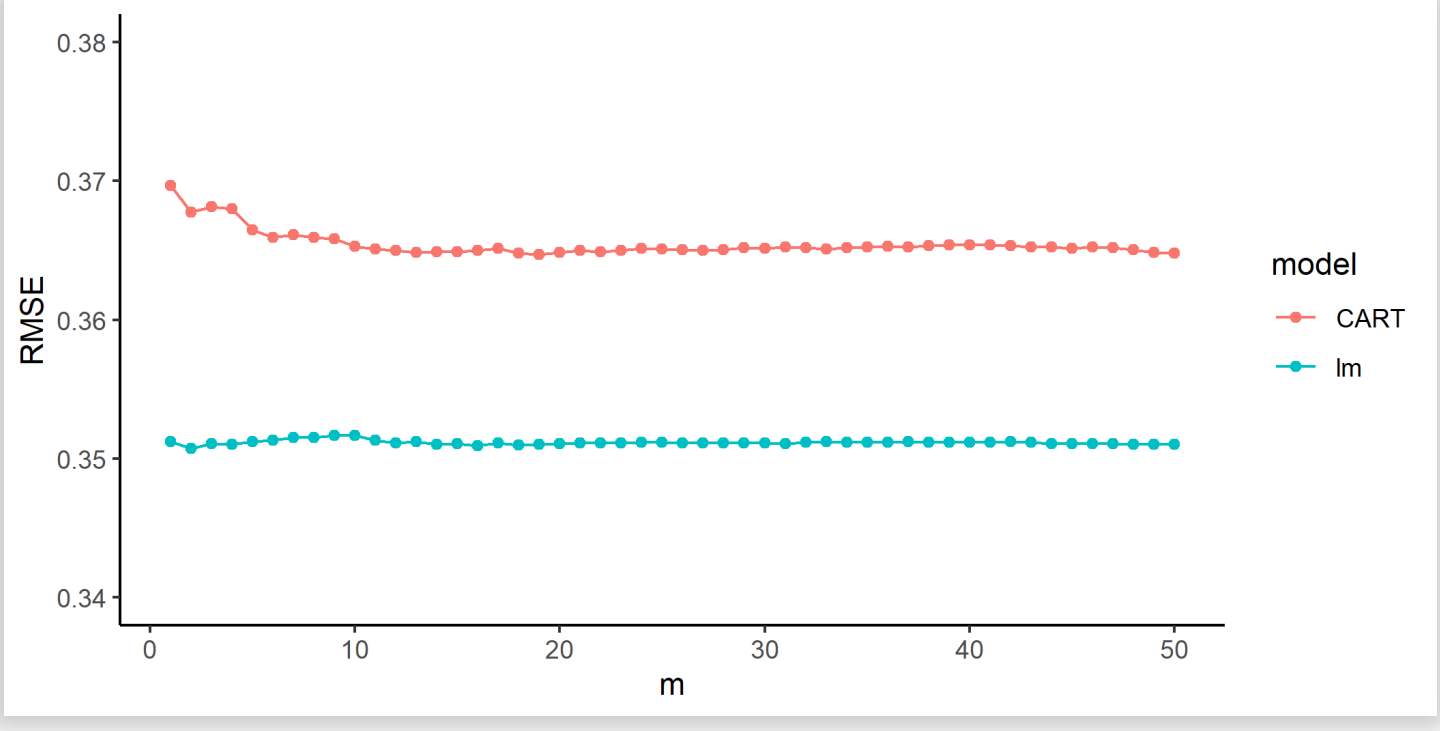
Why does Bagging work?



Which models are likely to benefit from bagging?

```
train_lm <- function(i, .n) {
  samp <- sample(1:.n, .n, replace = TRUE)
  lm(income ~ ., data = okcupid2_imp_mice_train[samp, ])
}

train_tree <- function(i, .n) {
  samp <- sample(1:.n, .n, replace = TRUE)
  rpart(income ~ ., data = okcupid2_imp_mice_train[samp, ])
}

predict_mod <- function(mod) {
  tibble(pred = predict(mod, okcupid2_imp_mice_valid))
}

rmse_m_models <- function(m, .preds) {
  rmse(okcupid2_imp_mice_valid$income, rowMeans(.preds[, 1:m]))
}

n <- nrow(okcupid2_imp_mice_train)
m_max <- 50

mod_lms <- map(1:m_max, train_lm, .n = n)
mod_trees <- map(1:m_max, train_tree, .n = n)

preds_lm <- map_dfc(mod_lms, predict_mod)
preds_tree <- map_dfc(mod_trees, predict_mod)
```

```
rmse_lm <- map_dbl(1:m_max, rmse_m_models, .preds = preds_lm)
rmse_tree <- map_dbl(1:m_max, rmse_m_models, .preds = preds_tree)
tibble(m = rep(1:m_max, 2),
       model = c(rep("lm", m_max), rep("CART", m_max)),
       RMSE = c(rmse_lm, rmse_tree)) %>%
  ggplot(aes(m, RMSE, color = model)) +
  ylim(c(0.34, 0.38)) +
  geom_line() + geom_point() +
  theme_classic()
```



Bagging with `ipred`

```
library(ipred)

mod_bag_na <- bagging(income ~ ., data = okcupid2_train, nbagg = 5
pred_bag_na <- predict(mod_bag_na, okcupid2_valid)
report_rmse_and_cor(okcupid2_valid$income, pred_bag_na)
```

```
## RMSE: 0.367
## CORR: 0.433
```

```
mod_bag_mice <- bagging(income ~ ., data = okcupid2_imp_mice_train
pred_bag_mice <- predict(mod_bag_na, okcupid2_imp_mice_valid)
report_rmse_and_cor(okcupid2_valid$income, pred_bag_mice)
```

```
## RMSE: 0.365
## CORR: 0.438
```

Random Forests

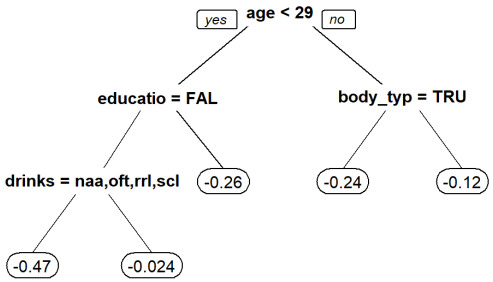
Random Forests: The Gist

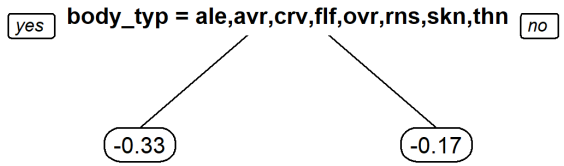
For $j = 1$ to m do

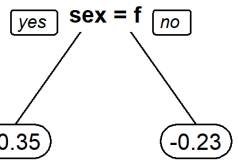
- Generate a bootstrap sample of the original data
- Train an unpruned tree model on this sample
- For each splot:
  - Randomly select m_{try} predictors
  - Select best predictor for split in this subset only

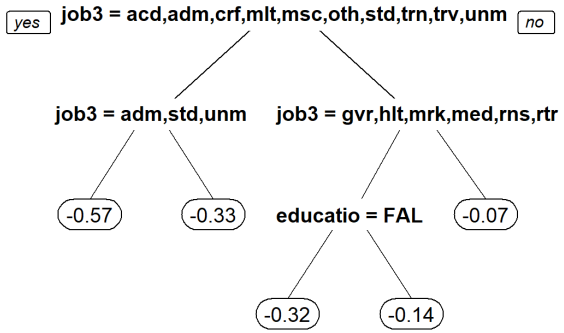
End.

What does RF add on top of Bagging?









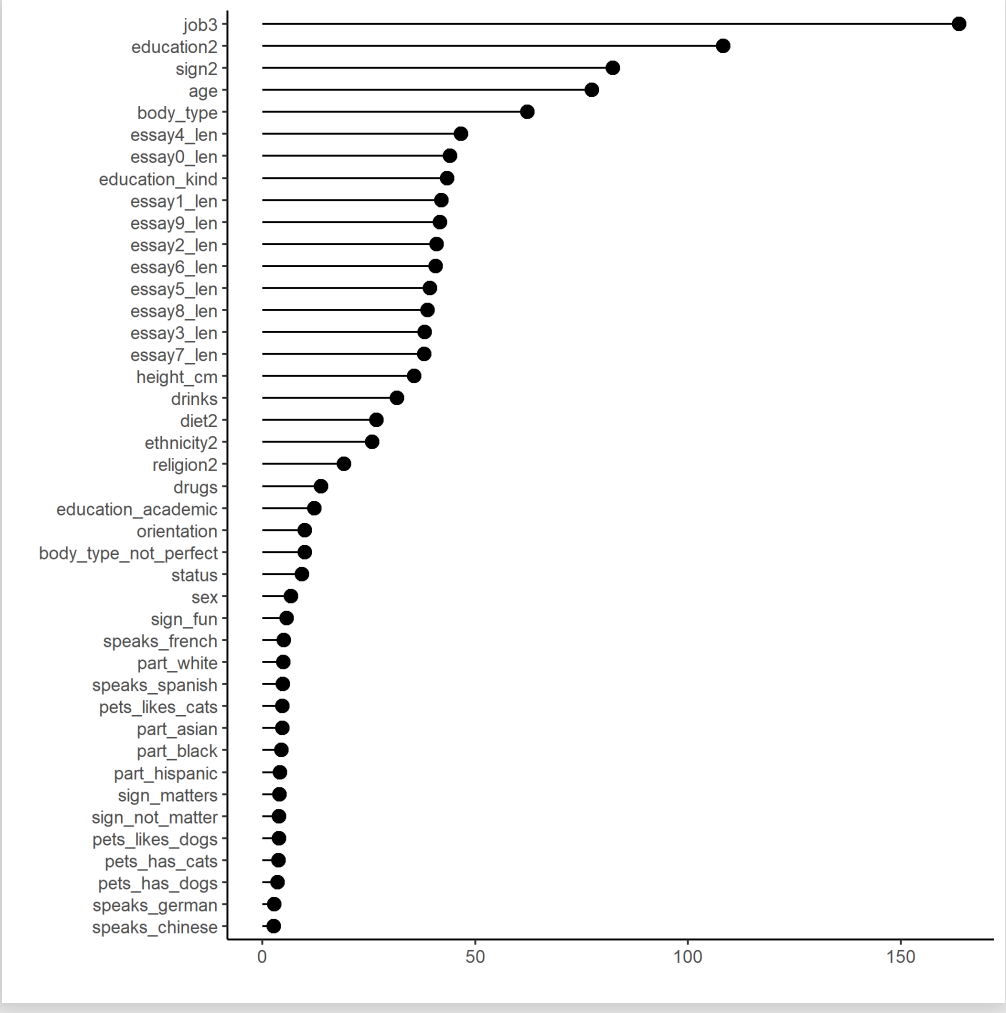
RF with `randomForest`

For some reason this implementation of RF won't accept missing values:

```
library(randomForest)

mod_rf_mice <- randomForest(income ~ ., data = okcupid2_imp_mice_t
                            mtry = 10, ntree = 50, importance = TRUE)
pred_rf_mice <- predict(mod_rf_mice, okcupid2_imp_mice_valid)
report_rmse_and_cor(okcupid2_valid$income, pred_rf_mice)
```

```
## RMSE: 0.354
## CORR: 0.491
```

For variable importance check out the `importance` field of the RF object:

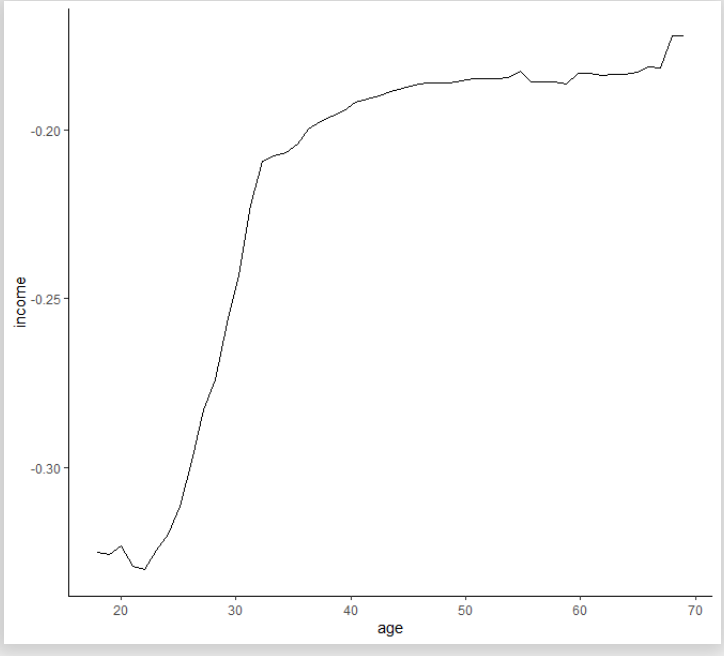


What is troubling about variable importance?

Partial Dependency Plots

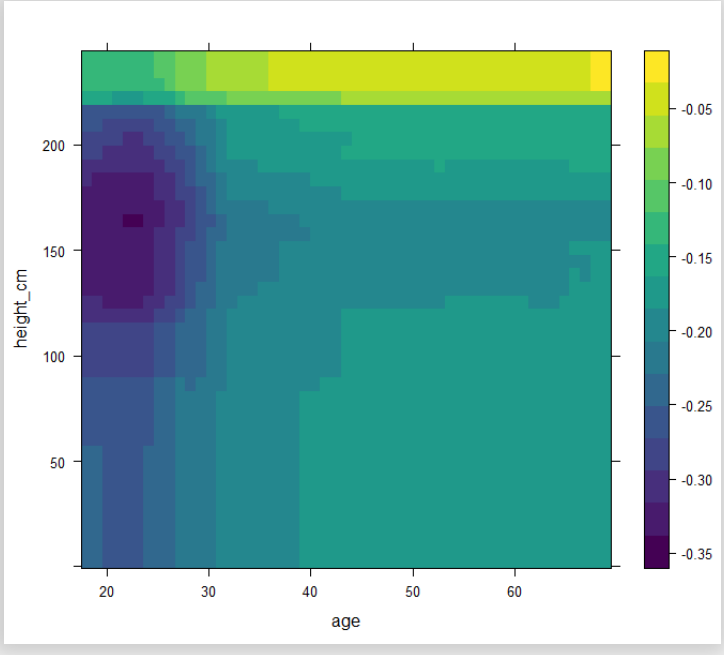
1. Select a subset X_s of variables (usually 1-2) you wish to know the relation to the dependent variable y
2. For each possible permutation of X_s run the model on the original data imputing this specific permutation instead of the values of X_s
3. Average predictions of y
4. Plot y vs. X_s

```
partial(mod_rf_mice, pred.var = "age",
        plot = TRUE, plot.engine = "ggplot2") +
  labs(y = "income") +
  theme_classic()
```



Let's see this for `age` and `height_cm` together (takes some time!):

```
partial(mod_rf_mice, pred.var = c("age", "height_cm"), plot = TRUE
```



Gradient Boosted Trees

Boosting: The Gist

Compute the average response $\overline{y}$, and use this as the initial predicted value for each sample

For $j = 1$ to m do

- Compute the residual for each observation
- Sample a fraction of the original data
- Fit a regression tree of a certain *"interaction depth"* using the residual as response
- Predict each sample using this tree
- Update the predicted value of each sample by adding the previous iteration's value to current predicted value X *learning rate* or *shrinkage* parameter

End.

What you end up with:

$$\hat{y}_i = \overline{y} + \sum_{j=1}^{m} \lambda \cdot tree_j(\mathbf{x}_i)$$

That is, GBT (or GBM) in general is called an additive model.

> 💡 Where does "Gradient" come from?

- The added random sampling of the data was added later, this version is called *Stochastic* Gradient Boosting, the default for the parameter often called "bagging fraction" is 0.5
- The shrinkage λ can be turned into λ_j, a unique weight for each added tree
- Note the **many** tuning parameters here. Do you know how to tune multiple params?
- Not so parallel now, eh?

GBT with gbm

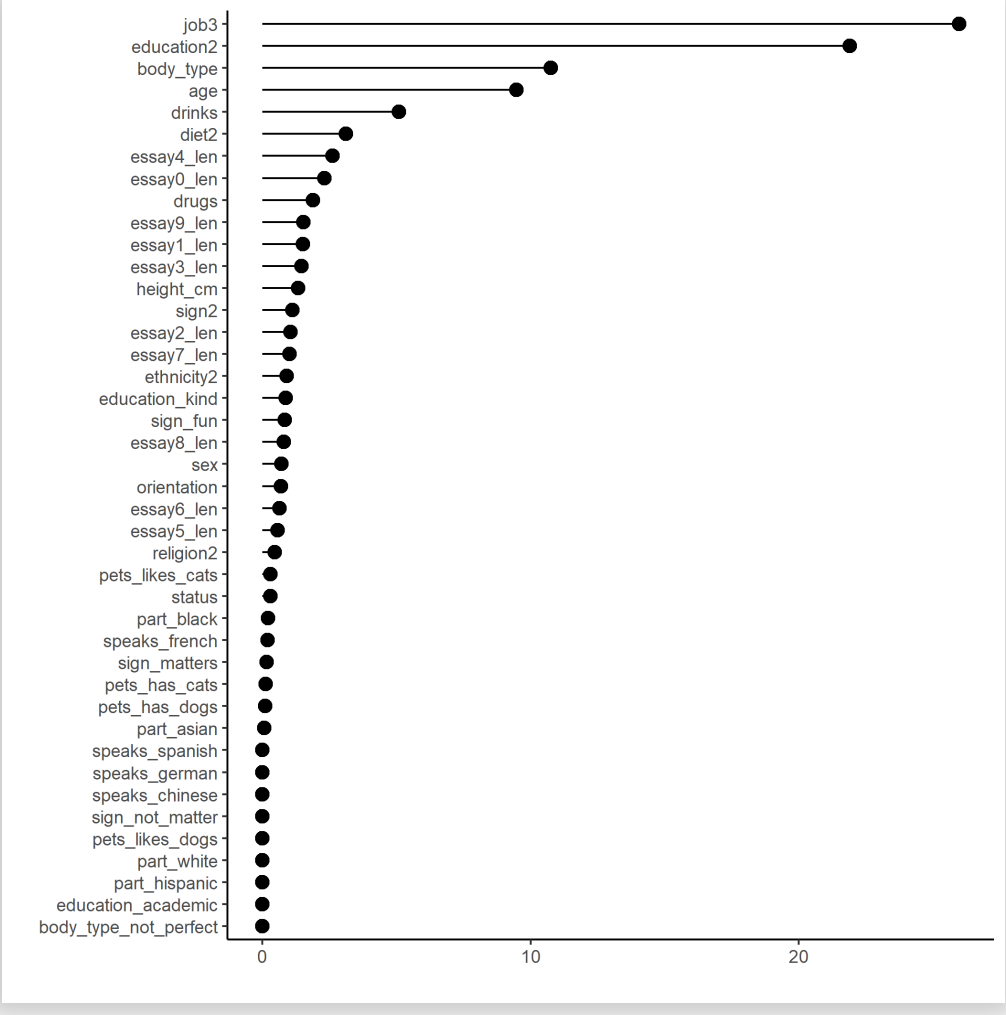
```
library(gbm)

mod_gbm_na <- gbm(income ~ ., data = okcupid2_train,
                  distribution = "gaussian",
                  n.trees = 500, shrinkage = 0.1, bag.fraction = (
pred_gbm_na <- predict(mod_gbm_na, okcupid2_valid, n.trees = 500)
report_rmse_and_cor(okcupid2_valid$income, pred_gbm_na)
```

```
## RMSE: 0.35
## CORR: 0.507
```

```
mod_gbm_mice <- gbm(income ~ ., data = okcupid2_imp_mice_train,
                  distribution = "gaussian", n.trees = 500, shrink
pred_gbm_mice <- predict(mod_gbm_na, okcupid2_imp_mice_valid, n.tr
report_rmse_and_cor(okcupid2_valid$income, pred_gbm_mice)
```

```
## RMSE: 0.352
## CORR: 0.5
```

For variables importance see the `summary` function:



Do you see the difference in the profile of importance between RF and GBT?

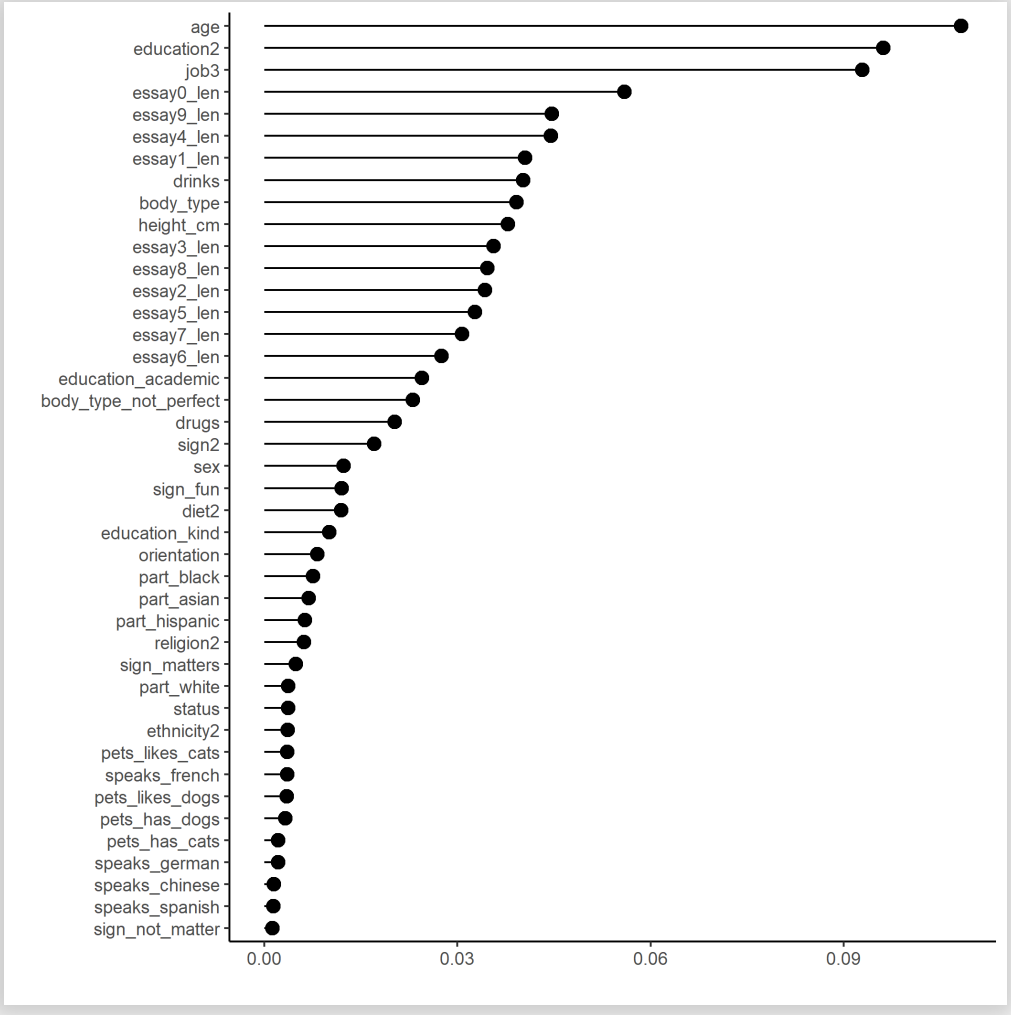
GBT with `xgboost`

`xgboost` is faster and has some additional features, e.g. the ability to train with a validation set until it shows no improvement, similar to neural networks.

```
library(xgboost)

dtrain <- xgb.DMatrix(data = data.matrix(okcupid2_train[, predicto
                      label = okcupid2_train$income)
dval <- xgb.DMatrix(data = data.matrix(okcupid2_valid[, predictors
                    label=okcupid2_valid$income)
watchlist <- list(train=dtrain, test=dval)

mod_xgboost_na <- xgb.train(data = dtrain, watchlist=watchlist,
                            objective = "reg:squarederror", nrounc
                            eta = 0.1, subsample = 0.5,
                            early_stopping_rounds = 10, verbose =
pred_xgboost_na <- predict(mod_xgboost_na, dval)
report_rmse_and_cor(okcupid2_valid$income, pred_xgboost_na)
```

```
## RMSE: 0.355
## CORR: 0.489
```

For variables importance see the `xgb.importance` function:



The CART (Classification)

Detour: A Classification Problem

OKCupid: Predicting Dogs or Cats People

It won't be easy:

```
okcupid %>% count(pets)
```

```
## # A tibble: 16 x 2
##    pets                                n
##    <chr>                           <int>
##  1 dislikes cats                     122
##  2 dislikes dogs                      44
##  3 dislikes dogs and dislikes cats   196
##  4 dislikes dogs and has cats         81
##  5 dislikes dogs and likes cats      240
##  6 has cats                         1406
##  7 has dogs                         4134
##  8 has dogs and dislikes cats        552
##  9 has dogs and has cats            1474
## 10 has dogs and likes cats          2333
## 11 likes cats                       1063
## 12 likes dogs                       7224
## 13 likes dogs and dislikes cats     2029
## 14 likes dogs and has cats          4313
## 15 likes dogs and likes cats       14814
## 16 <NA>                            19921
```

We will define cats/dogs people in the following way and stick to non-NA observations:

```
cats_categories <- c("has cats", "likes cats", "dislikes dogs and
                     "dislikes dogs and has cats")
dogs_categories <- c("has dogs", "likes dogs", "has dogs and disli
                     "likes dogs and dislikes cats")
okcupid3 <- okcupid %>%
  mutate(pets = case_when(
           pets %in% cats_categories ~ "cats",
           pets %in% dogs_categories ~ "dogs",
           TRUE ~ NA_character_)) %>%
  drop_na(pets)

okcupid3 %>% count(pets)
```

```
## # A tibble: 2 x 2
##   pets      n
##   <chr> <int>
## 1 cats   2550
## 2 dogs  13939
```

Notice the classes are very unbalanced, with roughly 11 dogs people to every 2 cats people.

As before in the vector `predictors` we have 36 continuous and categorical variables which may or may not be predictive to being a cats/dogs person:

```
okcupid3 <- okcupid3 %>%
  select(pets, predictors) %>%
  mutate(id = 1:n())

dim(okcupid3)
```

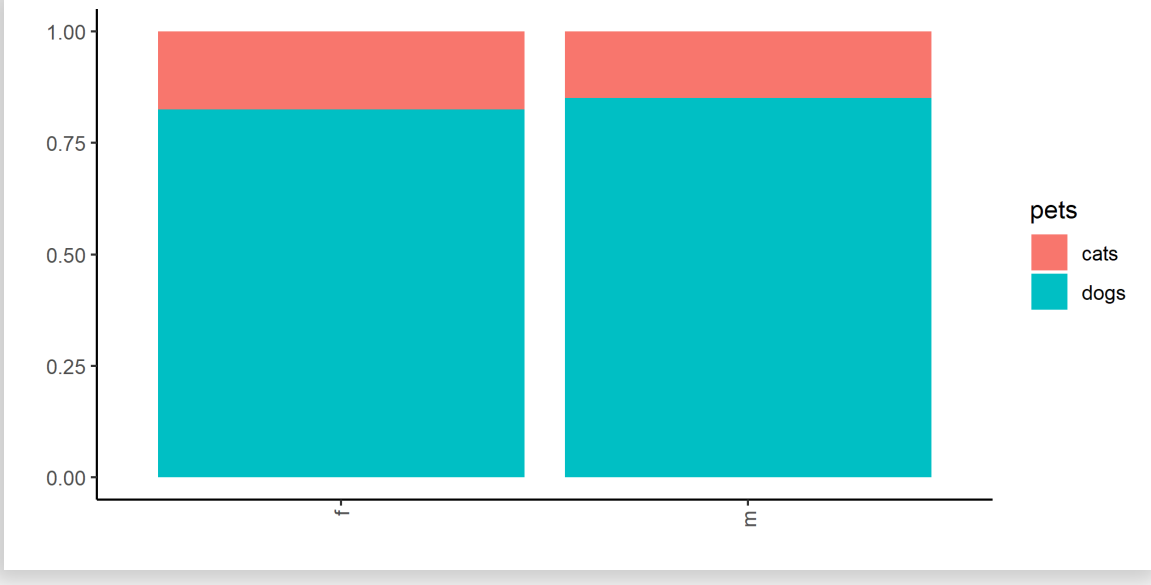
```
## [1] 16489    38
```

And as before we split the data into training, test and validation sets, not shown here:

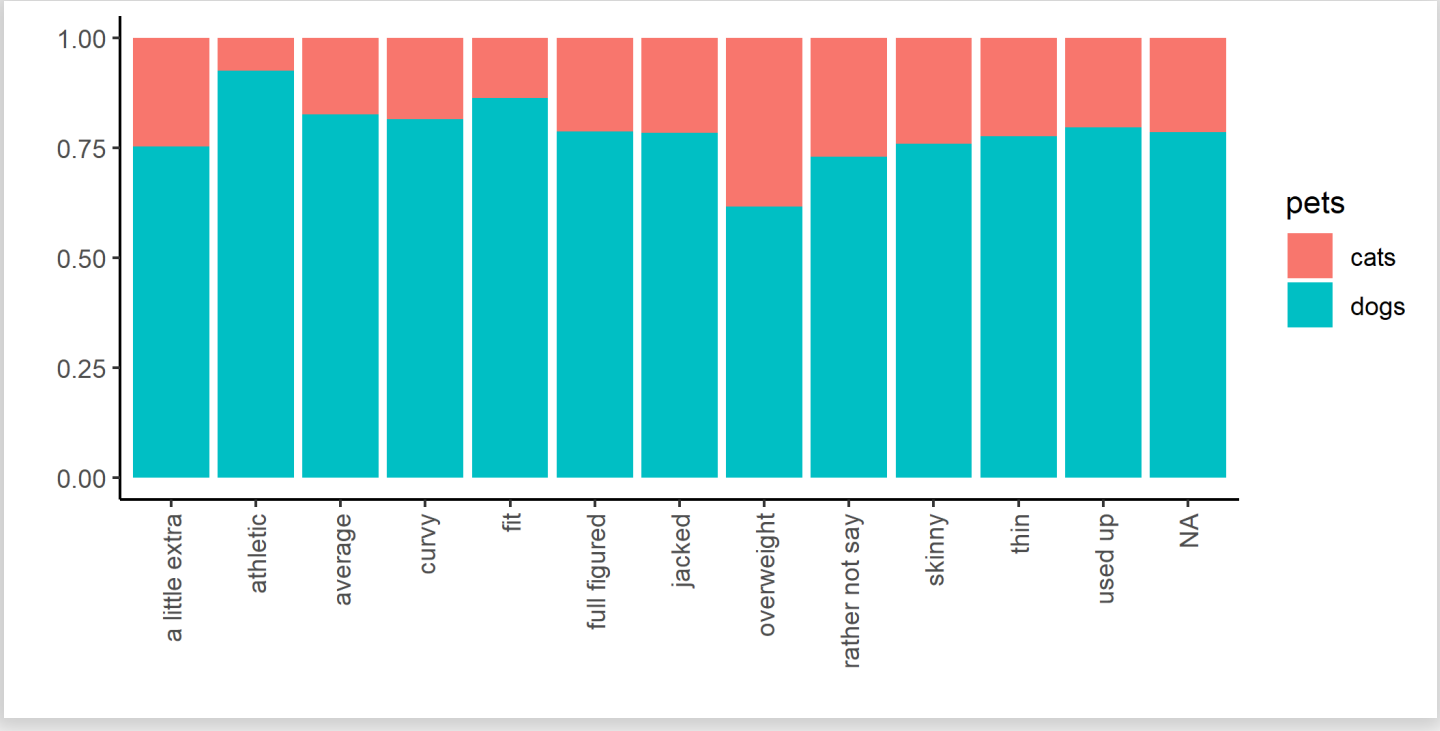
```
## train no. of rows: 10000
## validation no. of rows: 2489
## test no. of rows: 4000
```

Worth exploring relations between predictors and being a cats/dogs person:

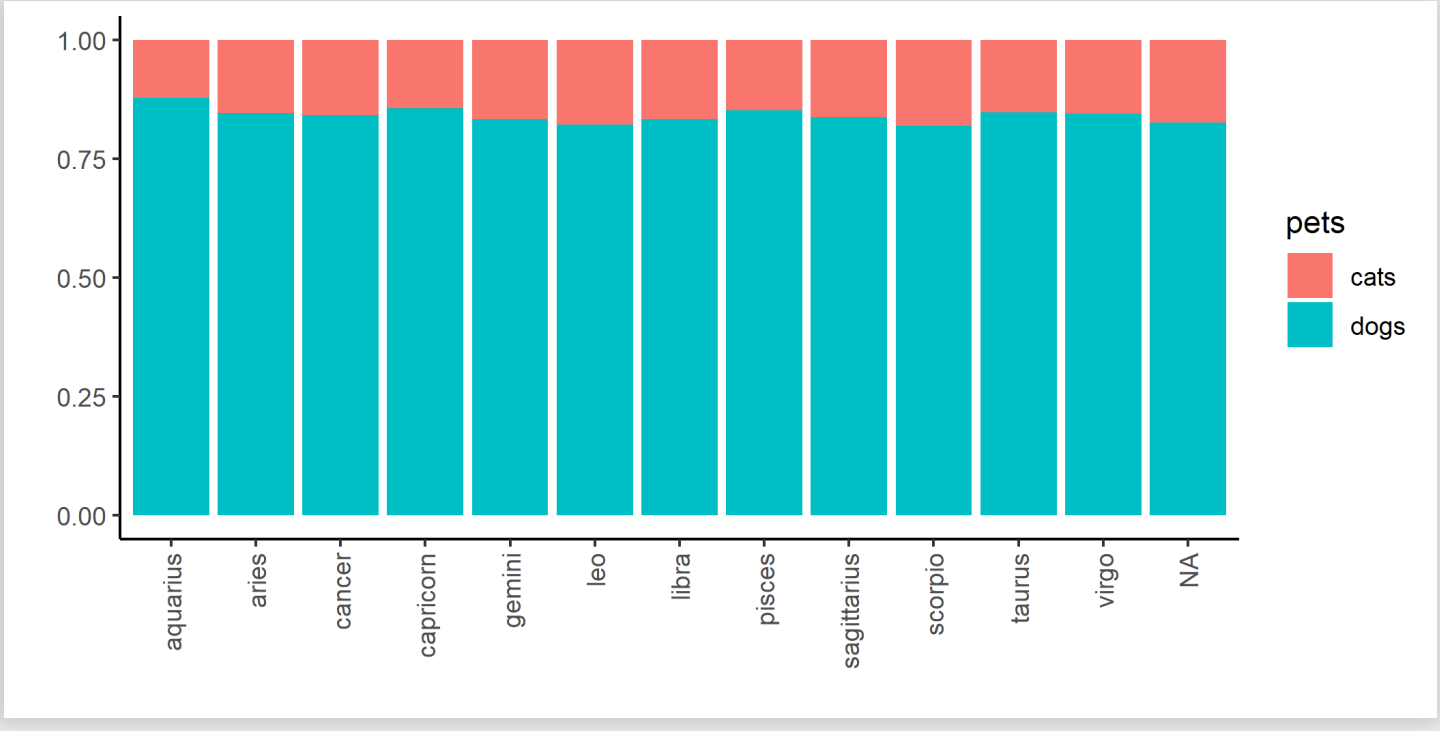
```
var_vs_pets_stackedbar <- function(var) {
  ggplot(okcupid3_train, aes({{var}}, fill = pets)) +
  geom_bar(position = "fill") +
  theme_classic() +
  labs(x = "", y = "") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust =
}

var_vs_pets_stackedbar(sex)
```



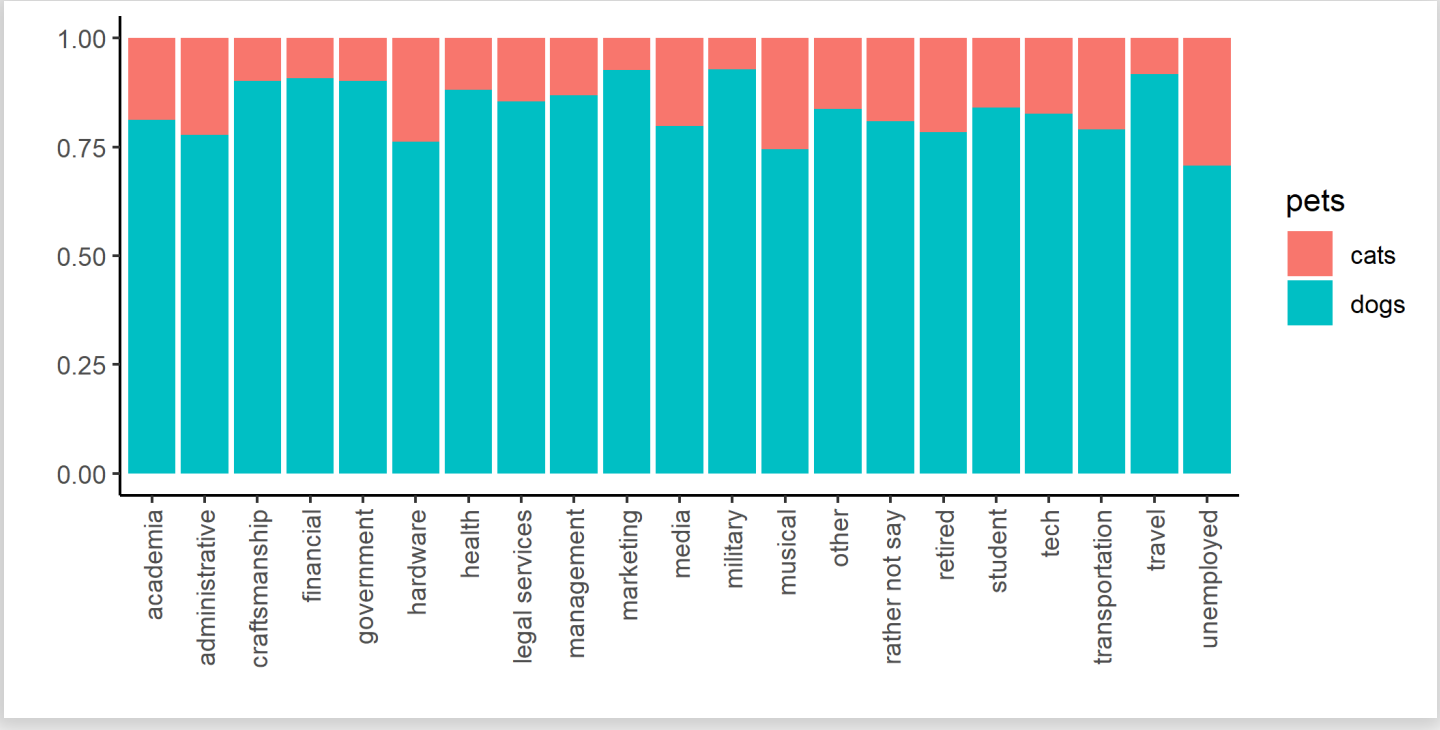
```
var_vs_pets_stackedbar(body_type)
```



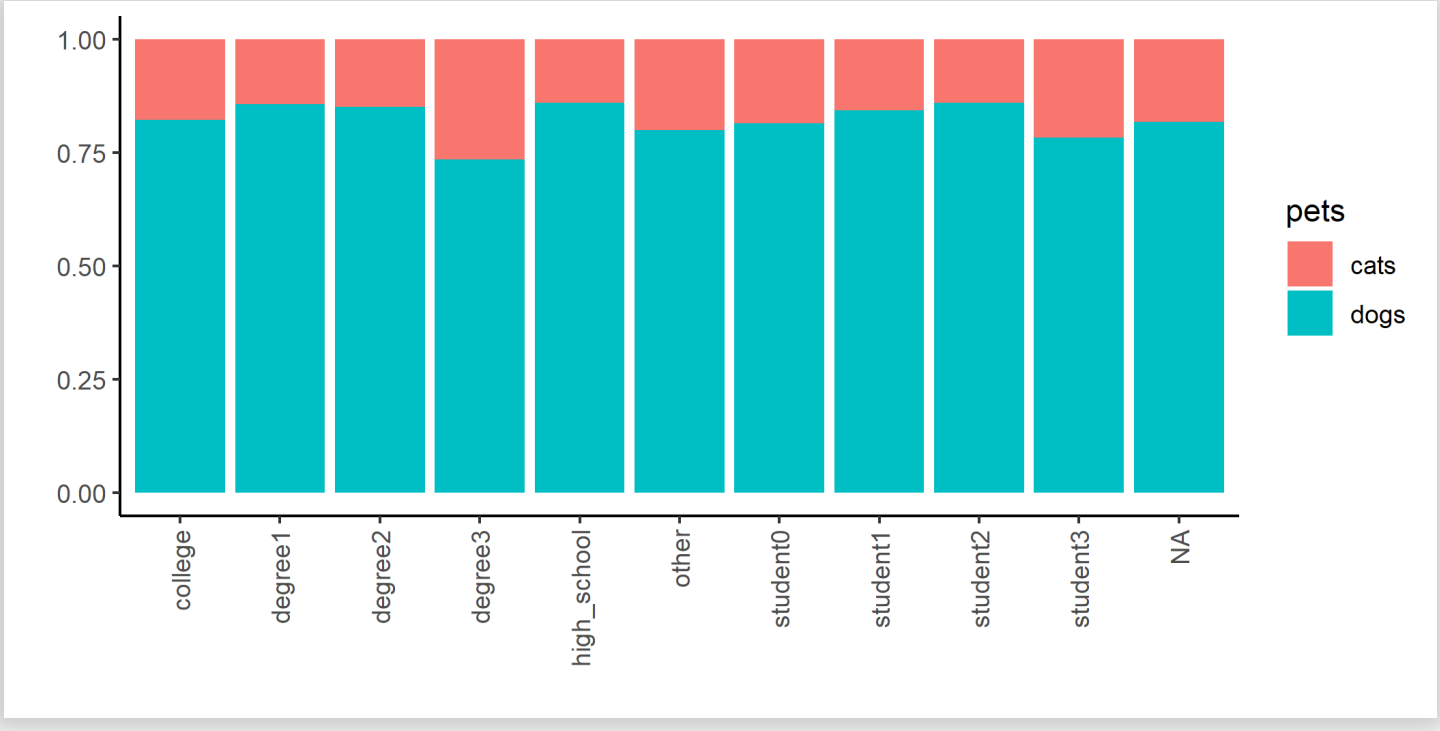
```
var_vs_pets_stackedbar(sign2)
```



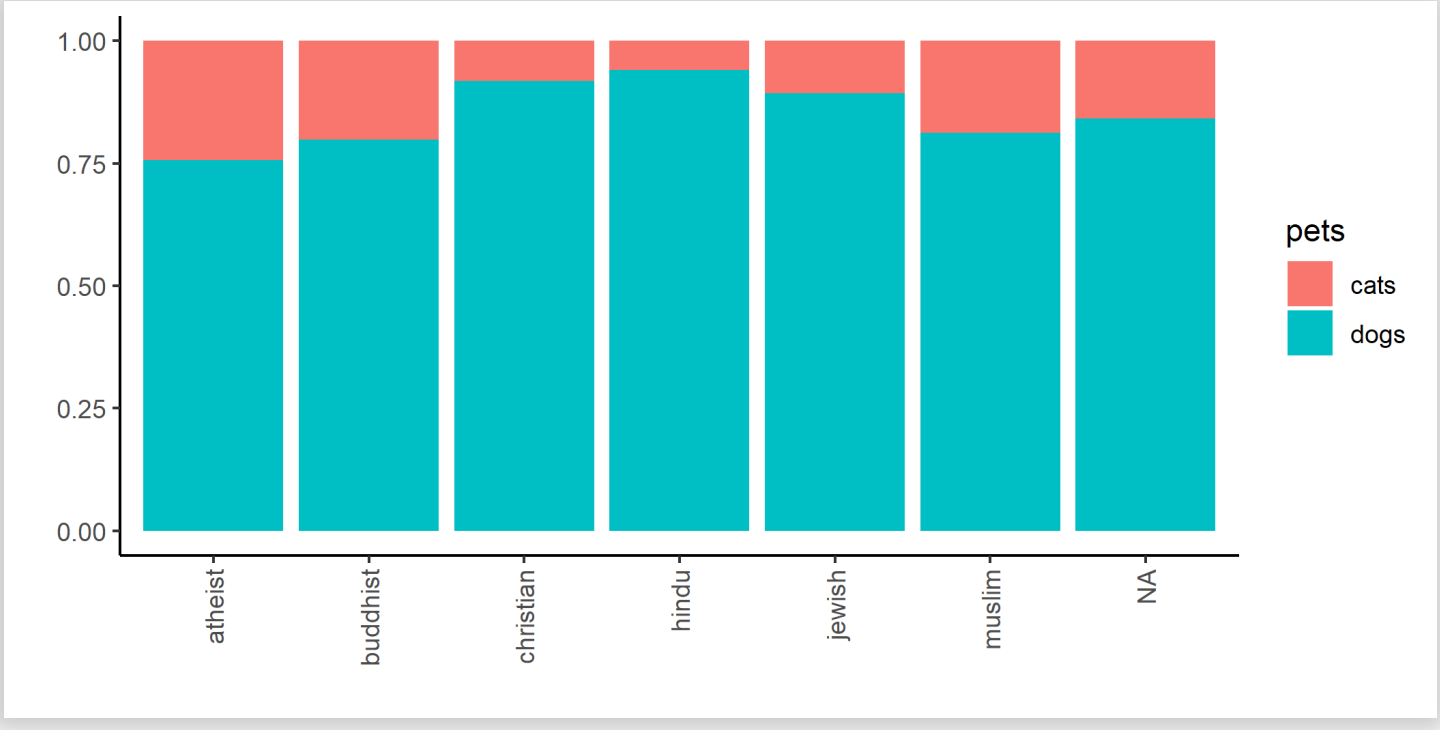
```
var_vs_pets_stackedbar(job3)
```



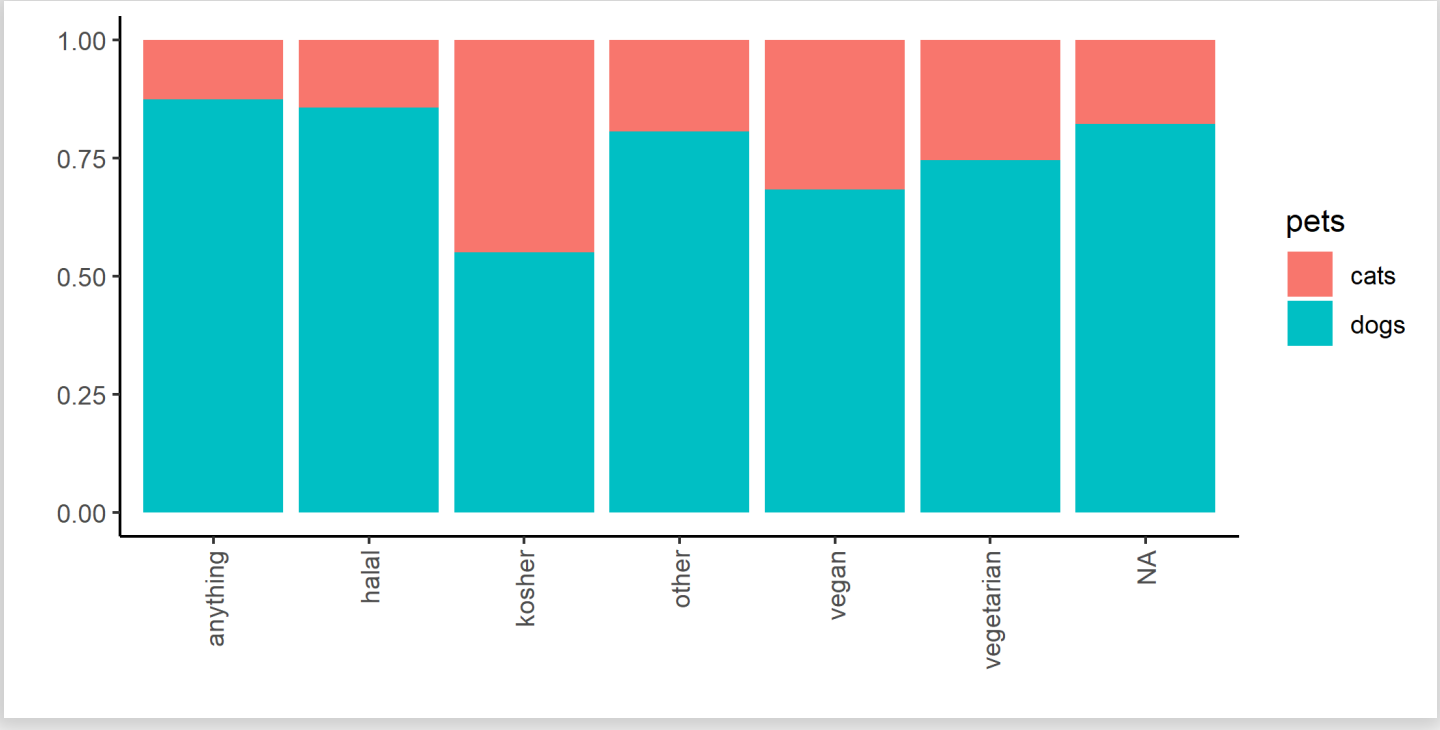
```
var_vs_pets_stackedbar(education2)
```



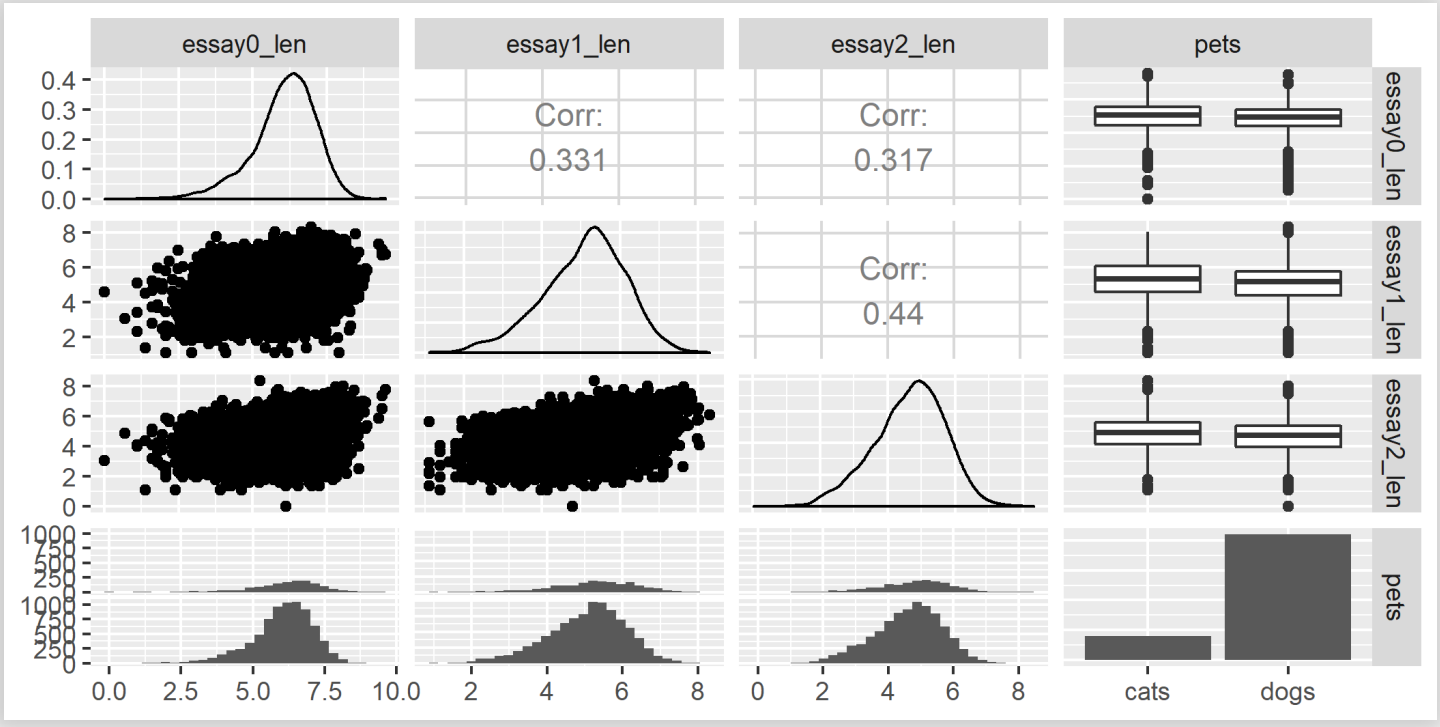
```
var_vs_pets_stackedbar(religion2)
```



```
var_vs_pets_stackedbar(diet2)
```



```
ggpairs(okcupid3_train %>%
          select(essay0_len:essay2_len, pets))
```



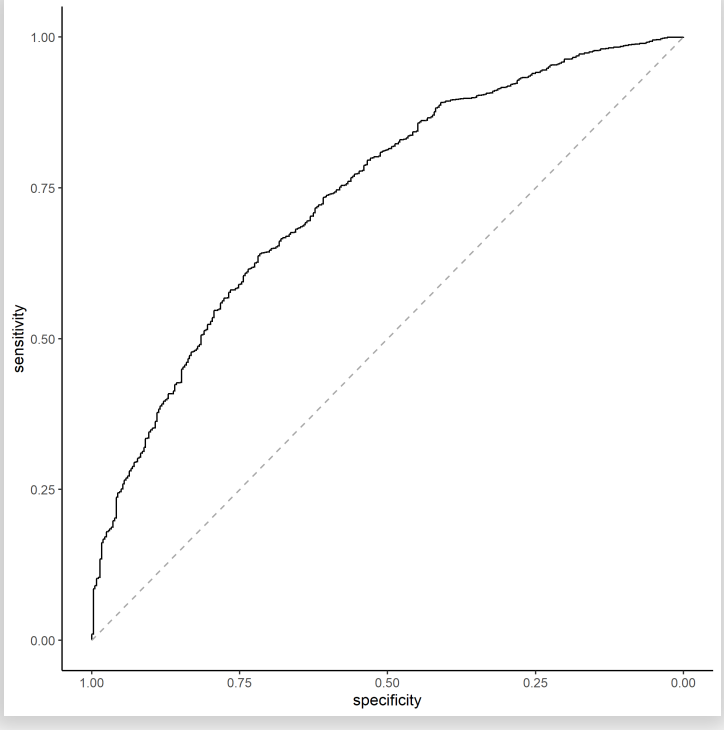
Baseline: Logistic Regression

As in `lm`, it is better to impute missing values than to lose these observations:

```
# library(mice)
# mice_obj <- mice(okcupid3, m = 1, maxit = 10, seed = 42)
# okcupid3_imp_mice <- complete(mice_obj)

okcupid3_imp_mice <- read_rds("../data/okcupid3_imp_mice.rds")
okcupid3_imp_mice_train <- okcupid3_imp_mice[train_idx, ]
okcupid3_imp_mice_valid <- okcupid3_imp_mice[valid_idx, ]

mod_glm_mice <- glm(pets ~ ., data = okcupid3_imp_mice_train, fami
pred_glm_mice <- predict(mod_glm_mice, okcupid3_imp_mice_valid, ty
```

Let's plot the ROC curve:

```
library(pROC)
roc_obj <- roc(okcupid3_valid$pets, pred_glm_mice)

ggroc(roc_obj) +
  theme_classic() +
  geom_segment(aes(x = 1, xend = 0, y = 0, yend = 1), color="darkg
```



Choosing a specific cutoff, the proportion of dogs people in the training sample (the "positive" class), 0.84, let's look at the confusion matrix:

```
cutoff <- mean(okcupid3_train$pets == "dogs")
pred_class <- ifelse(pred_glm_mice > cutoff, "dogs", "cats")
true_class <- okcupid3_valid$pets

table(true_class, pred_class)
```

```
##           pred_class
## true_class cats dogs
##       cats  238  125
##       dogs  678 1448
```

💡 What would be the accuracy? Recall (sensitivity, specificity)? Precision (PPV, NPV)?

We would like to know the model's AUC, accuracy for a given score cutoff, recall and precision:

```
report_accuracy_and_auc <- function(obs, pred, cutoff = mean(okcup
  roc_obj <- roc(obs, pred)
  AUC <- as.numeric(auc(roc_obj))
  res <- coords(roc_obj, x = cutoff,
                ret = c("accuracy", "recall", "precision", "speci
                transpose = TRUE)
  glue("AUC: {format(AUC, digits = 3)}
  ACC: {format(res['accuracy'], digits = 3)}
  Dogs: Recall: {format(res['recall'], digits = 3)}
        Precision: {format(res['precision'], digits = 3)}
  Cats: Recall: {format(res['specificity'], digits = 3)}
        Precision: {format(res['npv'], digits = 3)}")
}

report_accuracy_and_auc(okcupid3_valid$pets, pred_glm_mice)
```

```
## AUC: 0.738
## ACC: 0.677
## Dogs: Recall: 0.681
##       Precision: 0.921
## Cats: Recall: 0.656
##       Precision: 0.26
```

Baseline: GLMNET: Penalized Logistic Regression

```
okcupid3_imp_mat_train <- model.matrix( ~ ., okcupid3_imp_mice_tra
okcupid3_imp_mat_valid <- model.matrix( ~ ., okcupid3_imp_mice_val

glmnet_cv <- cv.glmnet(x = okcupid3_imp_mat_train,
                       y = okcupid3_train$pets, family = "binomial

best_lambda <- glmnet_cv$lambda.min

mod_glm_glmnet <- glmnet(x = okcupid3_imp_mat_train,
                         y = okcupid3_train$pets,
                         family = "binomial", lambda = best_lambda
pred_glm_glmnet <- predict(mod_glm_glmnet, okcupid3_imp_mat_valid,

report_accuracy_and_auc(okcupid3_valid$pets, pred_glm_glmnet[, 1])
```

```
## AUC: 0.739
## ACC: 0.671
## Dogs: Recall: 0.672
##       Precision: 0.922
## Cats: Recall: 0.667
##       Precision: 0.257
```

End of Detour

The OG CART

1. Find the the predictor to (binary) split on and the value of the split, by $Gini$ (a.k.a impurity) criterion
2. For each resulting node if $n_{node} > 20$ go to 1
3. Once full tree has been grown, perform pruning using the *cost-complexity parameter* c_p and the $Gini_{c_p}$ criterion
4. Predict the proportion of each class at each terminal node, or most common class

The $Gini$ criterion

- y is the discrete dependent variable with J classes
- The $Gini_{parent}$ at a parent node is:
  $\sum_{j=1}^{J} P(y=j)[1-P(y=j)] = \sum_j p_j(1-p_j)$
- a continuous predictor v is nominated for splitting the current node with splitting value l
- such that S_1 is the set of observations for which $v_i \leq l$
- and S_2 is the set of observations for which $v_i > l$
- p_{1_j} is the (observed) probability of y equals class j in set S_1

$$Gini_{split} = \frac{\#\text{obs node1}}{\#\text{obs parent}} Gini_{node1} + \frac{\#\text{obs node2}}{\#\text{obs parent}} Gini_{node2} =$$
$$\frac{\#\text{obs node1}}{\#\text{obs parent}} \sum_j p_{1_j}(1-p_{1_j}) + \frac{\#\text{obs node2}}{\#\text{obs parent}} \sum_j p_{2_j}(1-p_{2_j})$$

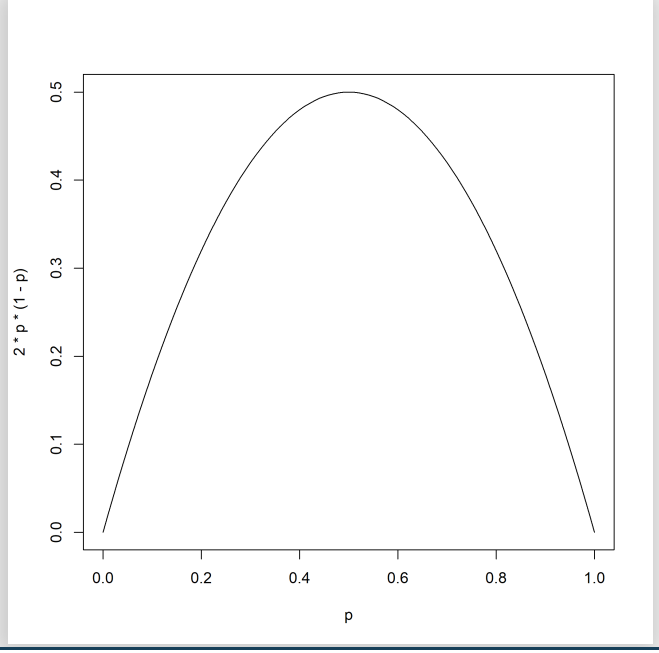
And we find l for which $\Delta G = Gini_{parent} - Gini_{split}$ is greatest.

Why Gini?

For $J = 2$ classes this means:
$p_1(1 - p_1) + p_2(1 - p_2) = 2p_1(1 - p_1)$

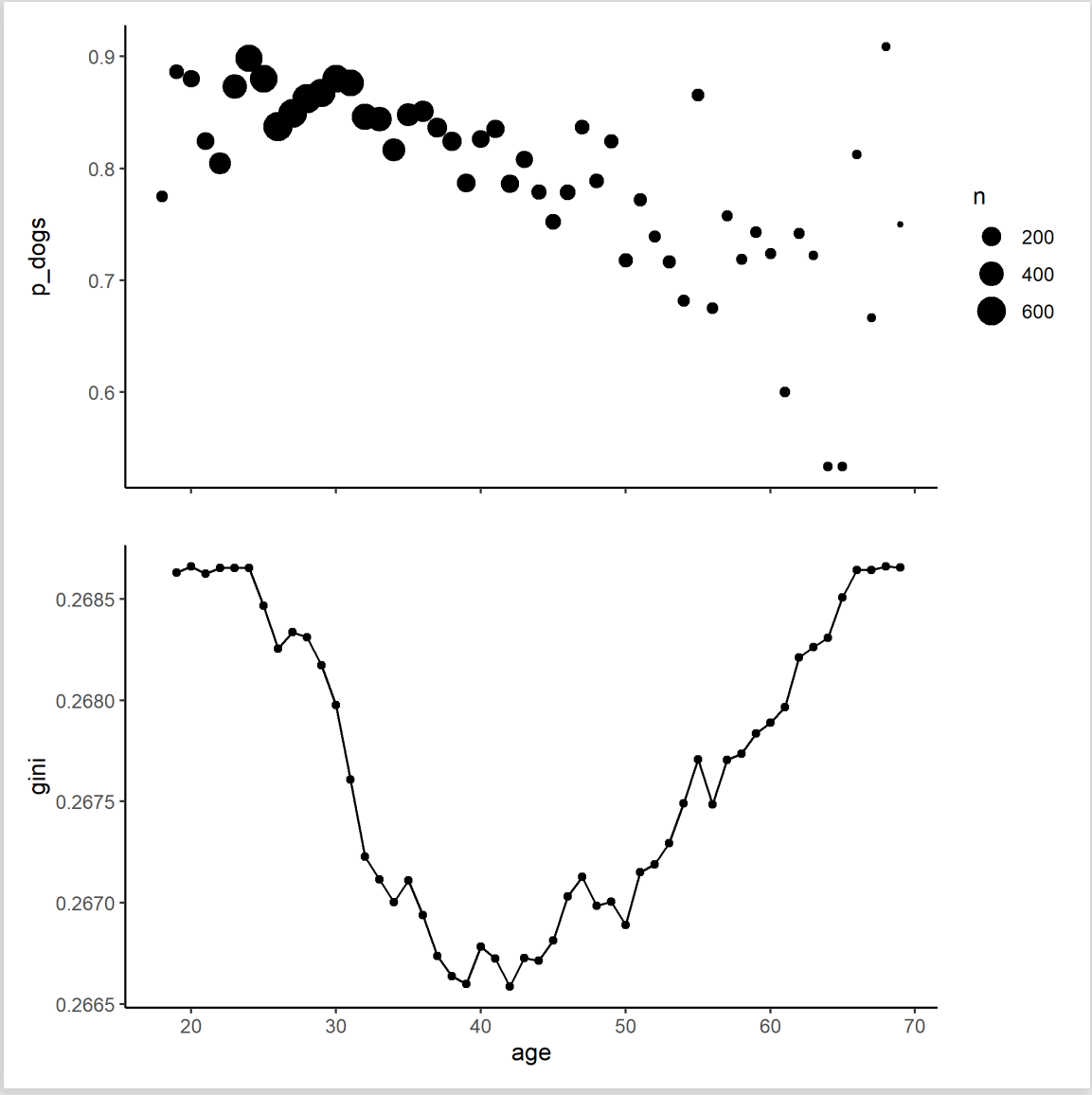
The more "impure" the node, the higher this metric:

```
p <- seq(0, 1, 0.01)
plot(p, 2 * p * (1 - p), type = "l")
```



For example if `religion2` is candidate in splitting `pets`:

```
gini <- function(l, df, v) {
  pets_above <- df %>% filter({{v}} >= l) %>% pull(pets)
  pets_below <- df %>% filter({{v}} < l) %>% pull(pets)
  p_dogs_above <- mean(pets_above == "dogs")
  gini_above <- 2 * p_dogs_above * (1 - p_dogs_above)
  p_dogs_below <- mean(pets_below == "dogs")
  gini_below <- 2 * p_dogs_below * (1 - p_dogs_below)
  gini_weighted <- (length(pets_above) / nrow(df)) * gini_above +
    (length(pets_below) / nrow(df)) * gini_below
  return(gini_weighted)
}

age <- seq(18,69, 1)
gini_age <- map_dbl(age, gini, df = okcupid3_train, v = age)

p1 <- okcupid3_train %>%
  group_by(age) %>%
  summarise(p_dogs = mean(pets == "dogs"), n = n()) %>%
  ggplot(aes(age, p_dogs)) +
  geom_point(aes(size = n)) +
  theme_classic() +
  labs(x = "") +
  theme(axis.text.x = element_blank())

p2 <- tibble(age = age, gini = gini_age) %>%
  ggplot(aes(age, gini)) +
  geom_line() +
```

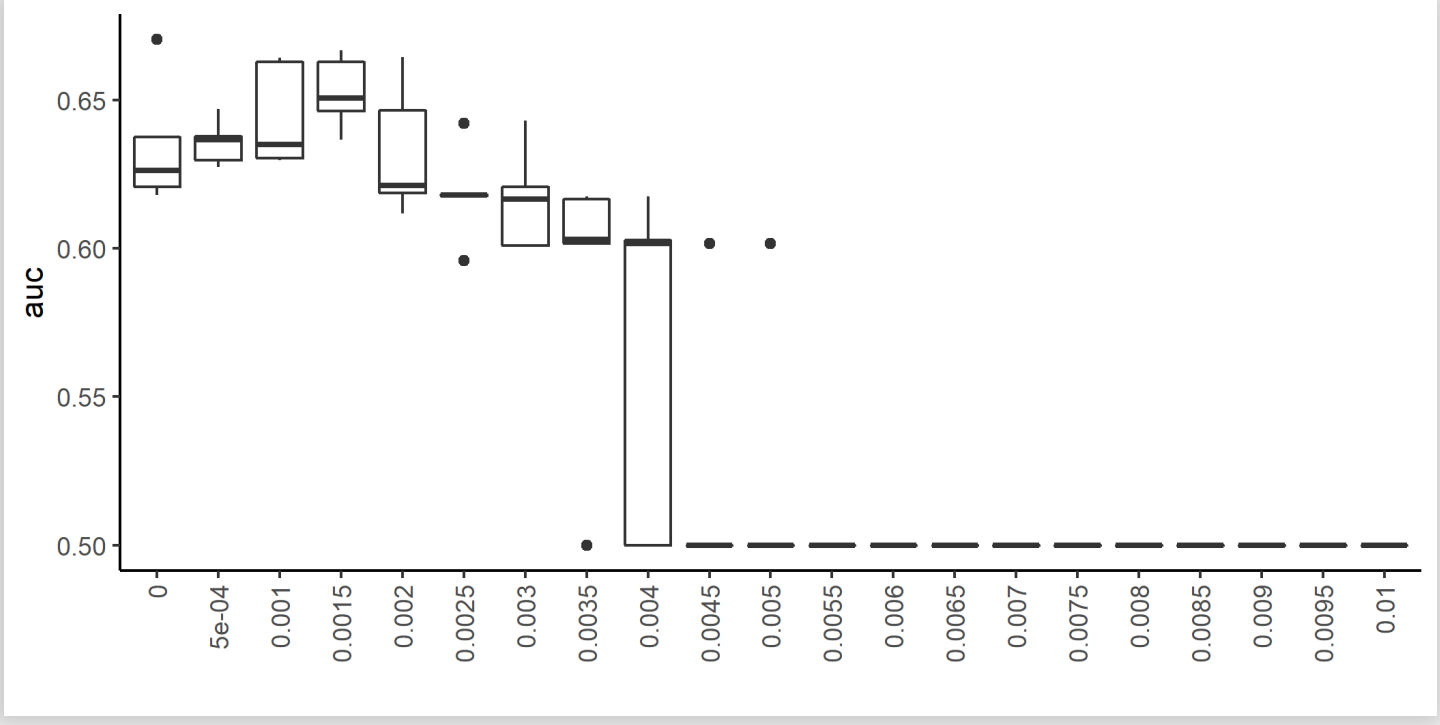

p_dogs
0.9
0.8
0.7
0.6
n
200
400
600
gini
0.2685
0.2680
0.2675
0.2670
0.2665
20
30
40
50
60
70
age

CART with `rpart`

Let's tune c_p for our data using a 5-fold (manual) Cross Validation. The criterion to maximize would be AUC.

```
n_cv <- 5; cp_seq <- seq(0, 0.01, 0.0005)

okcupid3_train_val <- okcupid3_train %>%
  mutate(val = sample(1:n_cv, n(), replace = TRUE))

get_validation_set_auc <- function(i, .cp) {
  ok_tr <- okcupid3_train_val %>% filter(val != i) %>% select(-val
  ok_val <- okcupid3_train_val %>% filter(val == i) %>% select(-va
  mod <- rpart(pets ~ ., data = ok_tr,
               control = rpart.control(cp = 0, xval = 1))
  mod <- prune(mod, cp = .cp)
  pred <- predict(mod, ok_val)[, 2]
  roc_obj <- roc(ok_val$pets, pred)
  as.numeric(auc(roc_obj))
}

get_cv_auc <- function(.cp) {
  tibble(cp = rep(.cp, n_cv),
         auc = map_dbl(1:n_cv, get_validation_set_auc, .cp = .cp))
}
```

```
cv_table <- map_dfr(cp_seq, get_cv_auc)

cv_table %>%
  mutate(cp = factor(cp)) %>%
  ggplot(aes(cp, auc)) +
  geom_boxplot() +
  theme_classic() +
  labs(x = "") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust =
```

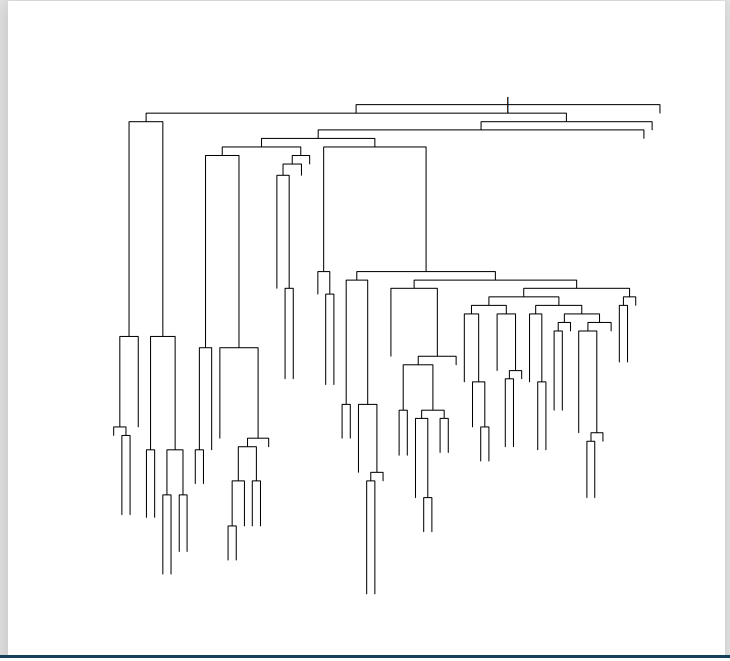


Training on the entire training set:

```
mod_tree_na_class <- rpart(pets ~ ., data = okcupid3_train,
                           control = rpart.control(cp = 0, xval =
mod_tree_na_class <- prune(mod_tree_na_class, cp = 0.0015)
```

There's no way to plot this tree nicely...

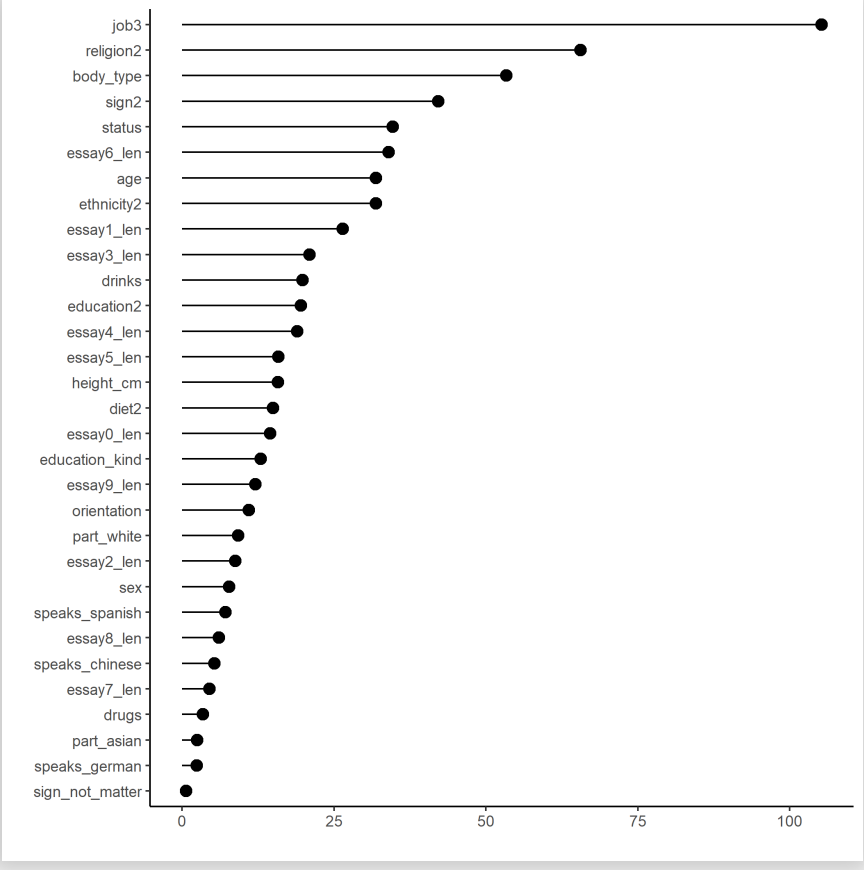
```
plot(mod_tree_na_class)
```



Printing the tree it seems like after splitting observations by religion, those who are Christian, Jewish or Hindu have a 88% of being dogs people, and the rest are then split by Status, Body Type, the length of their 9th Essay, etc...

```
Fitted party:
[1] root
|   [2] religion2 in atheist, buddhist, muslim
|   |   [3] status in available, married, seeing someone
|   |   |   [4] body_type in a little extra, curvy, full figured, skinny, thin
|   |   |   |   [5] job3 in administrative, craftsmanship, hardware, health, management, musical, other, tech
|   |   |   |   |   [6] essay3_len >= 4.45447: cats (n = 35, err = 11.4%)
|   |   |   |   |   [7] essay3_len < 4.45447
|   |   |   |   |   |   [8] sign2 in aquarius, cancer, capricorn, virgo: cats (n = 9, err = 11.1%)
|   |   |   |   |   |   [9] sign2 in aries, gemini, leo, pisces, sagittarius, scorpio: dogs (n = 15, err = 26.7%)
|   |   |   |   [10] job3 in academia, financial, government, legal services, marketing, media, rather not say, student, transportation, travel: dogs
|   |   |   [11] body_type in athletic, average, fit, jacked, overweight, rather not say, used up
|   |   |   |   [12] diet2 in kosher, vegan, vegetarian
|   |   |   |   |   [13] essay0_len < 6.77815: cats (n = 30, err = 23.3%)
|   |   |   |   |   [14] essay0_len >= 6.77815: dogs (n = 16, err = 31.2%)
|   |   |   |   [15] diet2 in anything, halal, other
|   |   |   |   |   [16] essay8_len >= 5.30087
|   |   |   |   |   |   [17] essay2_len < 5.59658: cats (n = 17, err = 17.6%)
|   |   |   |   |   |   [18] essay2_len >= 5.59658: dogs (n = 27, err = 37.0%)
|   |   |   |   |   [19] essay8_len < 5.30087
|   |   |   |   |   |   [20] essay4_len >= 7.88766: cats (n = 9, err = 22.2%)
|   |   |   |   |   |   [21] essay4_len < 7.88766: dogs (n = 150, err = 24.7%)
|   |   [22] status in single
|   |   |   [23] body_type in a little extra, average, curvy, fit, full figured, jacked, overweight, skinny, thin, used up
|   |   |   |   [24] ethnicity2 in native american, other, white
|   |   |   |   |   [25] drinks in desperately, not at all, rarely, very often
|   |   |   |   |   |   [26] job3 in academia, administrative, hardware, legal services, management, media, other, rather not say, tech, transportati
|   |   |   |   |   |   |   [27] education2 in degree3, other, student0, student3
|   |   |   |   |   |   |   |   [28] body_type in a little extra, average, full figured, jacked, overweight, skinny, thin
|   |   |   |   |   |   |   |   |   [29] job3 in academia, administrative, hardware, media, rather not say, tech: cats (n = 39, err = 12.8%)
|   |   |   |   |   |   |   |   |   [30] job3 in legal services, other: dogs (n = 7, err = 28.6%)
|   |   |   |   |   |   |   |   [31] body_type in curvy, fit, used up: dogs (n = 27, err = 33.3%)
|   |   |   |   |   |   |   [32] education2 in college, degree1, degree2, high_school, student1, student2
|   |   |   |   |   |   |   |   [33] body_type in a little extra, curvy, full figured: cats (n = 16, err = 25.0%)
|   |   |   |   |   |   |   |   [34] body_type in average, fit, jacked, overweight, skinny, thin, used up
|   |   |   |   |   |   |   |   |   [35] education2 in degree1, degree2, student2
|   |   |   |   |   |   |   |   |   |   [36] sign2 in capricorn, gemini, libra, pisces, taurus
|   |   |   |   |   |   |   |   |   |   |   [37] essay5_len < 5.02025
|   |   |   |   |   |   |   |   |   |   |   |   [38] job3 in hardware, legal services, management, media: cats (n = 16, err = 18.8%)
|   |   |   |   |   |   |   |   |   |   |   |   [39] job3 in academia, other, tech: dogs (n = 13, err = 38.5%)
|   |   |   |   |   |   |   |   |   |   |   [40] essay5_len >= 5.02025: dogs (n = 8, err = 25.0%)
|   |   |   |   |   |   |   |   |   |   [41] sign2 in aquarius, aries, cancer, leo, sagittarius, scorpio, virgo
```

Variables Importance



Prediction

```
pred_tree_na_class <- predict(mod_tree_na_class, okcupid3_valid)

report_accuracy_and_auc(okcupid3_valid$pets, pred_tree_na_class[,
```

```
## AUC: 0.612
## ACC: 0.788
## Dogs: Recall: 0.87
##       Precision: 0.881
## Cats: Recall: 0.309
##       Precision: 0.289
```

As expected, far from impressive.

Let's try using the imputed `NA` data:

```
mod_tree_imp_class <- rpart(pets ~ ., data = okcupid3_imp_mice_tra
                            control = rpart.control(cp = 0, xval =
mod_tree_imp_class <- prune(mod_tree_imp_class, cp = 0.0015)

pred_tree_imp_class <- predict(mod_tree_imp_class, okcupid3_imp_mi

report_accuracy_and_auc(okcupid3_valid$pets, pred_tree_imp_class[,
```

```
## AUC: 0.672
## ACC: 0.748
## Dogs: Recall: 0.793
##       Precision: 0.9
## Cats: Recall: 0.485
##       Precision: 0.285
```