Applications

Of Data Science

Feature Engineering

Applications of Data Science - Class 15

Giora Simchoni

gsimchoni@gmail.com and add #dsapps in subject

Stat. and OR Department, TAU

2023-05-28

What can you possibly teach us about Feature Engineering?

Our two previous modeling attempts, with Modeling in the Tidyverse and in The Trees were problematic.

- Using a "sinked" split, not using `set.seed()` - bit wasteful
- "Lazy" feature engineering, not using the entire data, not creative
- No interactions in linear models
- Not thinking about Multicollinearity in linear models
- Test once, on the same validation set, for everything
- Data leakage from training set to validation set
- Sometimes tuning params, sometimes not
- No pre-processing of features, if at all by a hunch
- Almost no treatment of class imbalance in classification
- Chaotic feature selection

We had one job:

Predict if an OKCupid user is a Cats or Dogs person, where:

```
cats_categories <- c("has cats", "likes cats", "dislikes dogs and
                     "dislikes dogs and likes cats")
dogs_categories <- c("has dogs", "likes dogs", "has dogs and disli
                     "likes dogs and dislikes cats")
```

And 80% of users are Dogs people, so we have class imbalance.

Model	AUC	Accuracy	Recall Dogs	Precision Dogs	Recall Cats	Precision Cats
GLM	0.74	0.68	0.68	0.92	0.66	0.26
GLM-EN	0.74	0.67	0.67	0.92	0.67	0.26
CART	0.67	0.75	0.79	0.9	0.49	0.28
GBT+Cutoff	0.74	0.70	0.71	0.9	0.65	0.32

And we were not impressed.

Resampling/Cross-Validation

Resampling done right

The main goal of resampling: No Surprises.

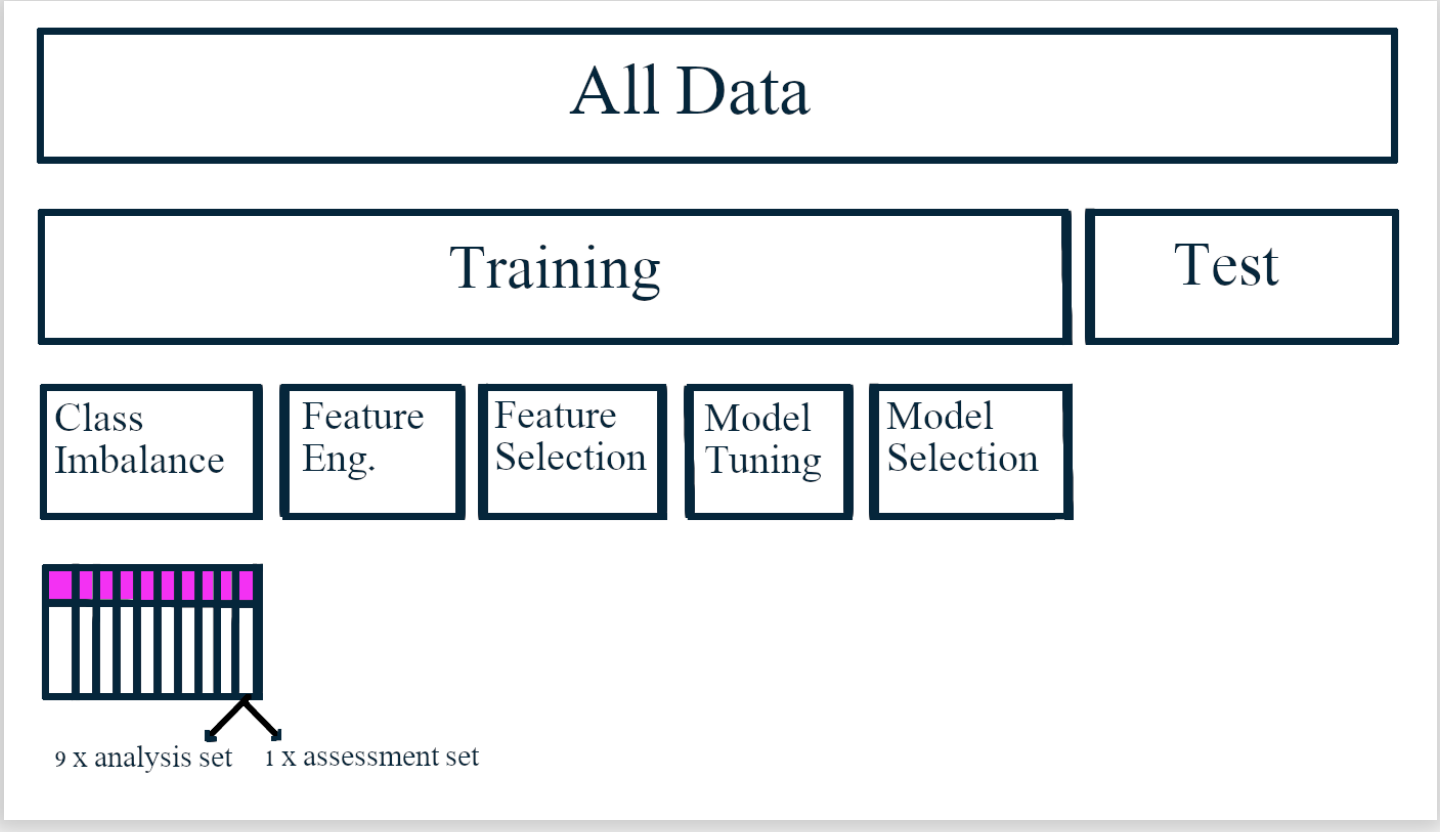
(By generating multiple versions of our data, and watching how our model behaves on unseen fresh data again. And again. And again.)

- Resampling is a key principle when developing a ML solution
- All ML practitioners are aware of the menace of overfitting
- But resampling should also be made for:
  - Feature engineering and transformation (should I use the Box-Cox or Yeo-Johnson transformation?)
  - Feature selection (Should I use 2nd-order interactions here?)
  - Choosing imputation strategy (Simply use the median or regress?)
  - Tuning parameters (100 or 500 trees?)
  - Model selection (LM or MARS?)

- A good resampling scheme will:
  - Have more than one train/valid/test samples!*
  - Be stratified (usually by dependent variable distribution)
  - Use a (paired/repeated/within) statistical test on resamples to reach a decision, or at least a proper plot
  - Avoid data leakage and not include the test set!
  - Encompass all steps in model building
  - Ideally (when data is Big) perform each step on a different part *of the training data*

> 💡 * How many resamples?
>
> – The higher the number the lower the variance of performance metric
>
> – The higher the number the less data used in each resample, the higher the bias of performance metric

Ideally (yes, I know)



And don't forget

Only once we decide on the final pre-processing, features, model and parameters, will we fit the model two more times:

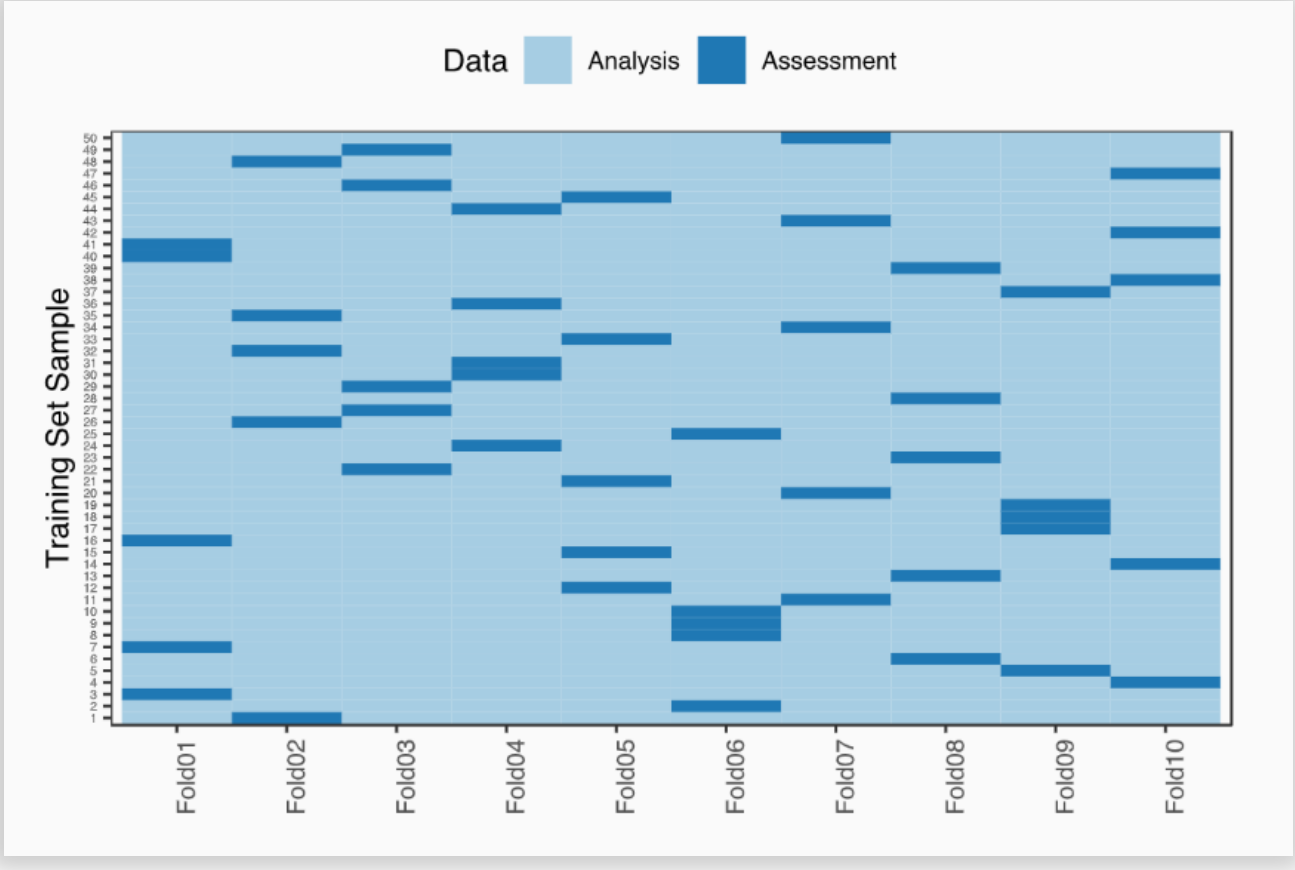
1. On the entire training data, then test on the testing data and get a realistic metric of performance
2. On the *entire* data, right before deployment to production

Many approaches to resampling

But two most common:

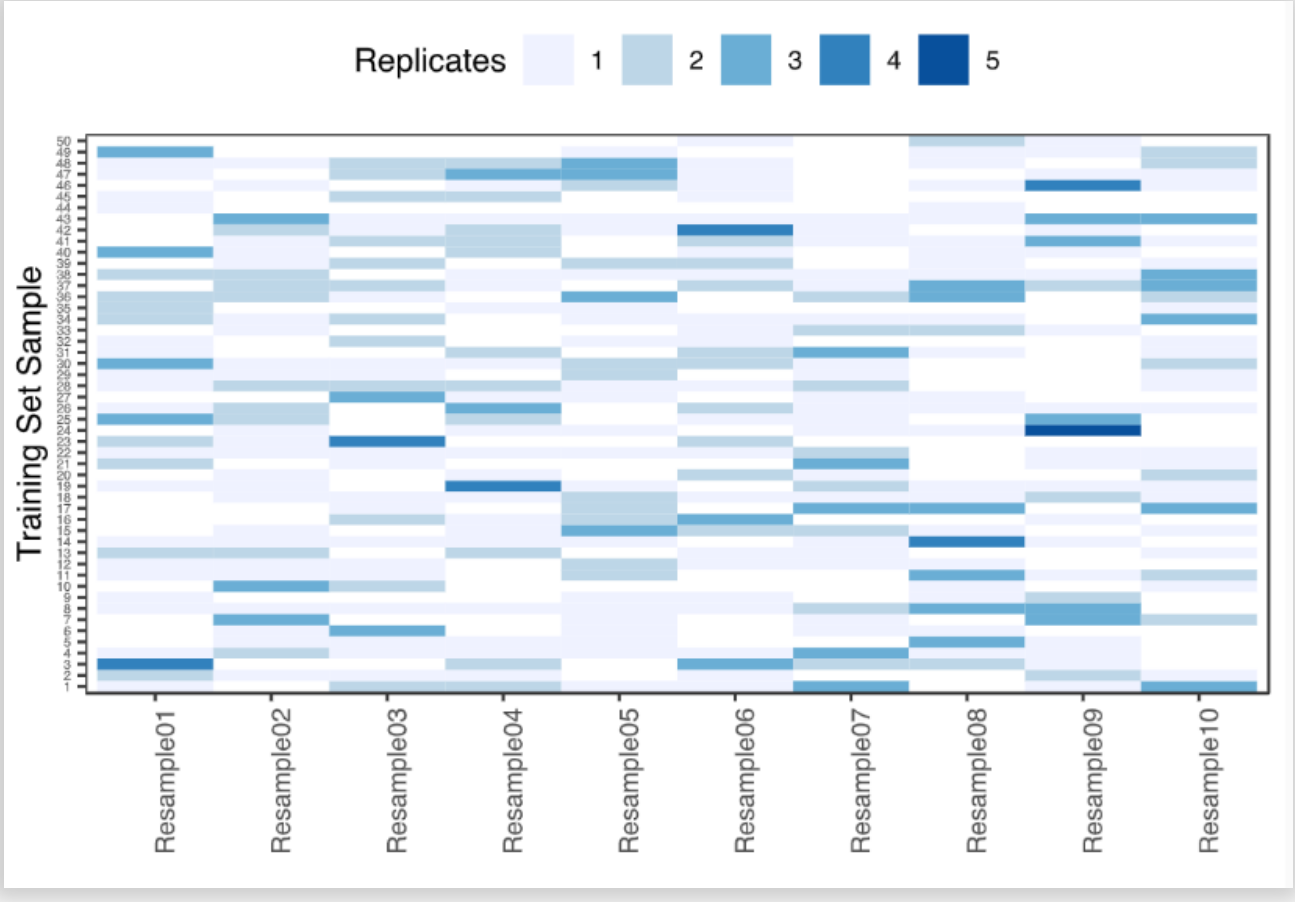
1. V-fold Cross Validation (+ optionally repeating n times)
2. Bootstrap Samples

V-fold Cross Validation



[Applied Machine Learning Workshop at Rstudio-conf 2019 / Max Kuhn]

Bootstrap Samples



💡 What is the probability of entering a fold (at least once)?

Let's Start from the Very Beginning

OkCupid from scratch

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

cats_categories <- c("has cats", "likes cats", "dislikes dogs and
                     "dislikes dogs and likes cats")
dogs_categories <- c("has dogs", "likes dogs", "has dogs and disli
                     "likes dogs and dislikes cats")

okcupid <- okcupid %>%
  mutate(pets = case_when(
           pets %in% cats_categories ~ "cats",
           pets %in% dogs_categories ~ "dogs",
           TRUE ~ NA_character_)) %>%
  drop_na(pets) %>%
  mutate(income = na_if(income, -1)) %>%
  select(-last_online)

okcupid[which(is.na(okcupid$height)), "height"] <- 70 # you don't
```

Using `rsample::initial_split(okcupid, strata = pets)` would have been better, but if we do it now we won't have the same test data to compare.

```
idx <- read_rds("../data/okcupid3_idx.rda")
train_idx <- idx$train_idx
valid_idx <- idx$valid_idx
test_idx <- idx$test_idx

okcupid_train <- okcupid[train_idx, ]
okcupid_valid <- okcupid[valid_idx, ]
okcupid_test <- okcupid[test_idx, ]
```

And let us bind `okcupid_train` and `okcupid_valid` together because we are going to use resampling anyway:

```
okcupid_train <- bind_rows(okcupid_train, okcupid_valid)

glue("train no. of rows: {nrow(okcupid_train)}
     test no. of rows: {nrow(okcupid_test)}")
```

```
## train no. of rows: 12729
## test no. of rows: 4000
```

See that the class balance is more or less the same:

```
okcupid_train %>% count(pets) %>% mutate(pct = n / sum(n))
```

```
## # A tibble: 2 × 3
##   pets      n   pct
##   <chr> <int> <dbl>
## 1 cats   2097 0.165
## 2 dogs  10632 0.835
```

```
okcupid_test %>% count(pets) %>% mutate(pct = n / sum(n))
```

```
## # A tibble: 2 × 3
##   pets      n   pct
##   <chr> <int> <dbl>
## 1 cats    693 0.173
## 2 dogs   3307 0.827
```

Good enough (we could also drop a few cat people at random if they're too much in the test group).

We stash our test data aside, kiss it good bye, we shall not see it again before the final assessment of our model.

That manager again.

Manager mentions casually: oh, and by the way, this model thingy of yours. Yeah, we need it to run in, like 10 milliseconds per user. Marketing team needs that, don't ask me why, and we can't have the users wait.

You: penalized logistic regression it is 😌

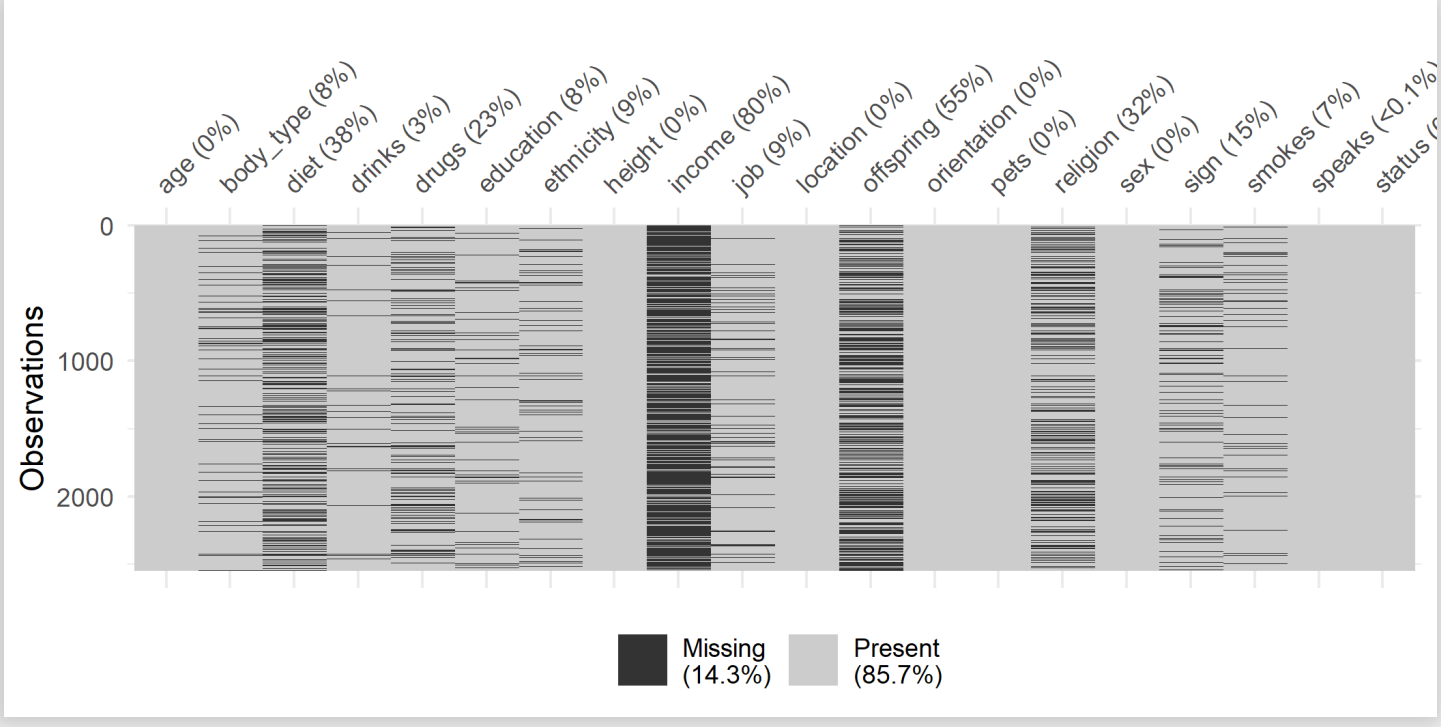
No one mentioned cats/dogs recall/precision to be of any particular importance, so we decide to maximize AUC and take care of cutoff later.

We explore the training data (skipping here at least a couple of hours in which we actually explore the data, I'm assuming we already know quite a bit!):

```
glimpse(okcupid_train)
```

```
## Rows: 12,729
## Columns: 30
## $ age          <dbl> 39, 19, 27, 24, 29, 24, 25, 25, 21, 25, 28, 25, 21,
## $ body_type    <chr> "fit", "curvy", NA, "fit", "fit", "fit", "athletic",
## $ diet         <chr> "mostly anything", "mostly vegetarian", "mostly anyt
## $ drinks       <chr> "socially", "rarely", NA, "socially", "socially", "s
## $ drugs        <chr> "never", "sometimes", NA, NA, "never", NA, "never",
## $ education    <chr> "graduated from college/university", NA, NA, "gradua
## $ essay0       <chr> "i am a very positive person who loves to laugh and
## $ essay1       <chr> "i work in the health and wellness industry and i tr
## $ essay2       <chr> "making people smile or laugh.", "ummmm everything?
## $ essay3       <chr> "my eyes", "my boobs. definitely.<br />\nand the fac
## $ essay4       <chr> "i enjoy malcolm gladwell, blink, tipping point, out
## $ essay5       <chr> "water,steak,chocolate,comfy jeans,transportation,fr
## $ essay6       <chr> "food, naps, women, and health", "caffeine, my baby
## $ essay7       <chr> "doing whatever has come up or has been planned. din
## $ essay8       <chr> "i am totally deaf in my right ear", "i love animals
## $ essay9       <chr> "you enjoy having fun, love to laugh, have a positiv
## $ ethnicity    <chr> "white", "white", NA, "white", "asian, pacific islan
## $ height       <dbl> 70, 60, 72, 65, 69, 72, 70, 67, 66, 67, 63, 63, 62,
## $ income       <dbl> 8e+04, NA, NA, NA, 3e+04, 5e+04, 1e+05, NA, 2e+04, N
## $ job          <chr> "medicine / health", "sales / marketing / biz dev",
## $ location     <chr> "mountain view, california", "pleasant hill, califor
```

```r
library(naniar)

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



Q&A

Q: What type of variables do we have?

A: Numeric: age, height income; Text: essays; Categorical: all others

Q: Can our model(s) handle missing values?

A: Probably gonna use `glmnet` so: NO.

Q: Are missing values a serious issue here?

A: Oh yes. `income` 80+% missing, `offspring` 50+% missing.

Q&A

Q: Can our model(s) handle categorical values?

A: Probably gonna use `glmnet` so: NO.

Q: Categorical variables with many levels?

A: Oh yes. `speaks` has over 2K. And there are those `essay`s.

Q: Numeric variables skewed?

A: Income, but all you need is log.

Q: Classes imbalance scenario?

A: Pretty imbalanced, 5 dogs people for every 1 cat person. Gonna have to take care of it.

You have to have a strategy

Strategy: Divide training data into 6 samples for 6 main decision/steps/tunings:

1. Missing data imputation
2. Class imbalance treatment
3. Feature Engineering for `essay` data
4. Looking into interactions
5. Model Tuning
6. Cutoff choice

> 💡 We assume these decisions are additive or independent of each other.
>
> Does it have to be and what can we do if we suspect not? (Remember we are Statisticians!)

Looking at our strategy, we decide the 12.7K observations need not split evenly between steps.

- Cutoff choice is "peanuts" compared to the other decisions, let's decide in advance that at this stage we would fit the chosen model on ~12K previously seen observations and choose a cutoff by looking at the remaining ~700 observations
- Missing data imputation, class imbalance and interaction steps will each be given 2K observations
- Model tuning 3K observations
- Text feature engineering and selection 3K observations
- Total: 12,729 training observations

We wish to make sure our samples are stratified by `pets`, so we'll use `initial_split()` in stages:

```
library(tidymodels)
tidymodels_prefer()

set.seed(42)
ok_cutoff_split <- initial_split(okcupid_train,
                                 strata = pets, prop = (729 + 1)/1
ok_tr_cutoff <- training(ok_cutoff_split)
ok_train <- testing(ok_cutoff_split)

set.seed(42)
ok_missing_split <- initial_split(ok_train,
                                  strata = pets, prop = (2000 + 1)
ok_tr_missing <- training(ok_missing_split)
ok_train <- testing(ok_missing_split)

set.seed(42)
ok_imbalance_split <- initial_split(ok_train,
                                    strata = pets, prop = (2000 + 1)
ok_tr_imbalance <- training(ok_imbalance_split)
ok_train <- testing(ok_imbalance_split)
```

```
set.seed(42)
ok_interaction_split <- initial_split(ok_train,
                                      strata = pets, prop = (2000
ok_tr_interaction <- training(ok_interaction_split)
ok_train <- testing(ok_interaction_split)

set.seed(42)
ok_tuning_split <- initial_split(ok_train,
                                 strata = pets, prop = 3000/6000)
ok_tr_tuning <- training(ok_tuning_split)
ok_tr_eng <- testing(ok_tuning_split)
```

```
## # A tibble: 7 × 4
##   name        df                    n_rows cats_rate
##   <chr>       <list>                 <dbl>     <dbl>
## 1 missing     <tibble [2,000 × 30]>   2000     0.164
## 2 imbalance   <tibble [2,000 × 30]>   2000     0.164
## 3 engineering <tibble [3,000 × 30]>   3000     0.165
## 4 interaction <tibble [2,000 × 30]>   2000     0.164
## 5 tuning      <tibble [3,000 × 30]>   3000     0.165
## 6 cutoff      <tibble [729 × 30]>      729     0.165
## 7 test        <tibble [4,000 × 30]>   4000     0.173
```

Dealing with Missing Data

Missing Data

We could fill an entire semester on this subject!

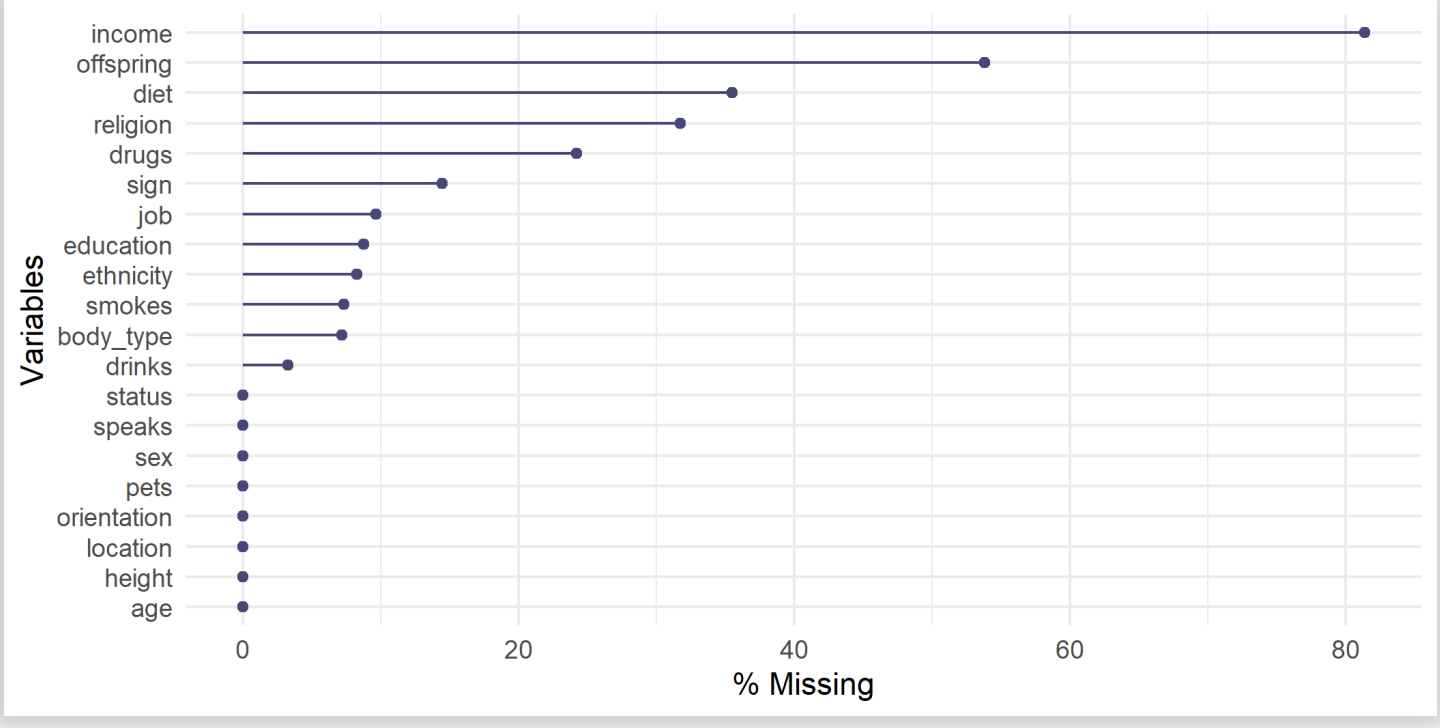
- What causes a missing datum?
  - Survey data
  - Merging data sources
  - Failure of measurement

- What is the missing data mechanism?
  - See Little & Rubin (2019) seminal work
  - MCAR: "on some days the weighing scale batteries drained and we couldn't take the mice weight"
  - MAR: "on the 2024 elections poll a massive Facebook campaign called women to refuse to cooperate with the Seker-Sheker company due to allegations of sexual harassment by the company's head Dani Marom"
  - MNAR: "some of the drug addicts under study missed the final few meetings in the treatment group and their data were discarded"

> 💡 How would you asses the missing data mechanism in your dataset?

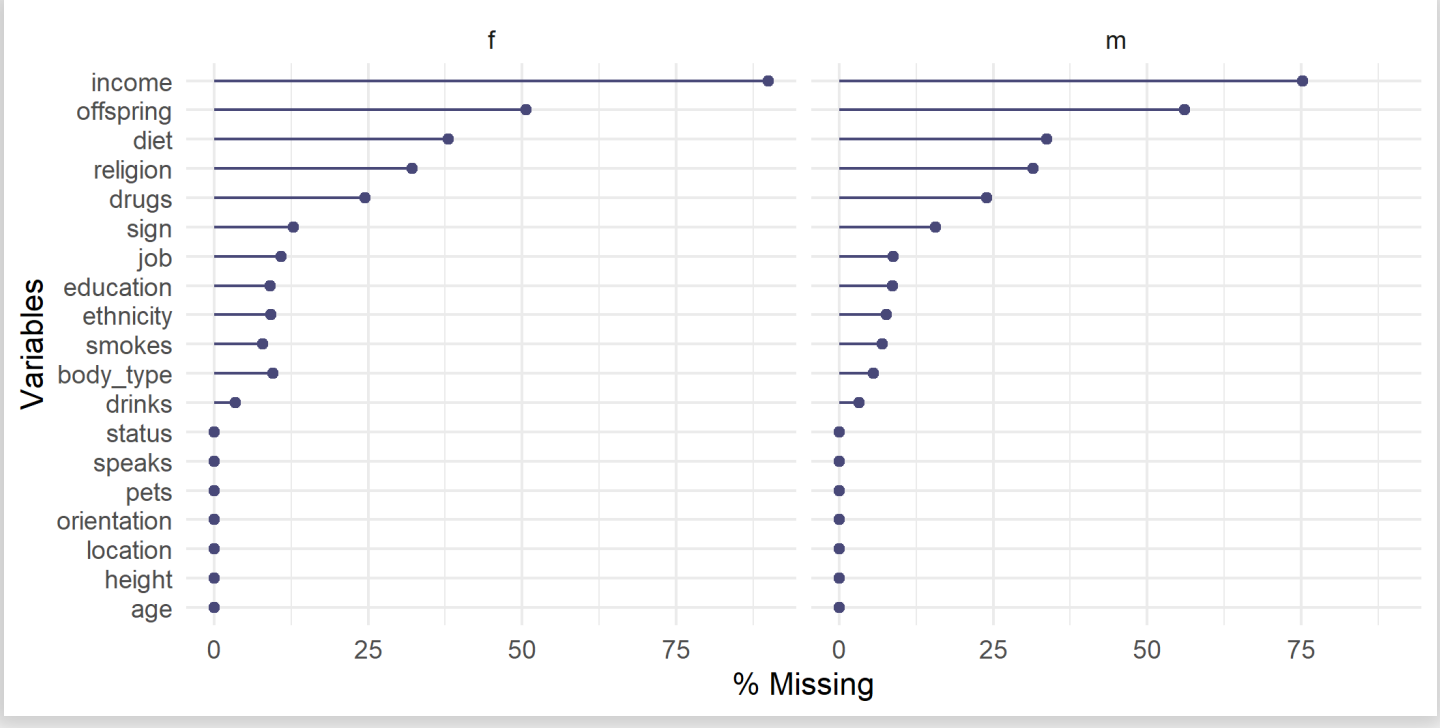
Exploring percent missingness

```
gg_miss_var(ok_tr_missing %>%
              select(-starts_with("essay")),
            show_pct = TRUE)
```



Exploring percent missingness by a predictor

```
gg_miss_var(ok_tr_missing %>%
              select(-starts_with("essay")),
            facet = sex, show_pct = TRUE)
```

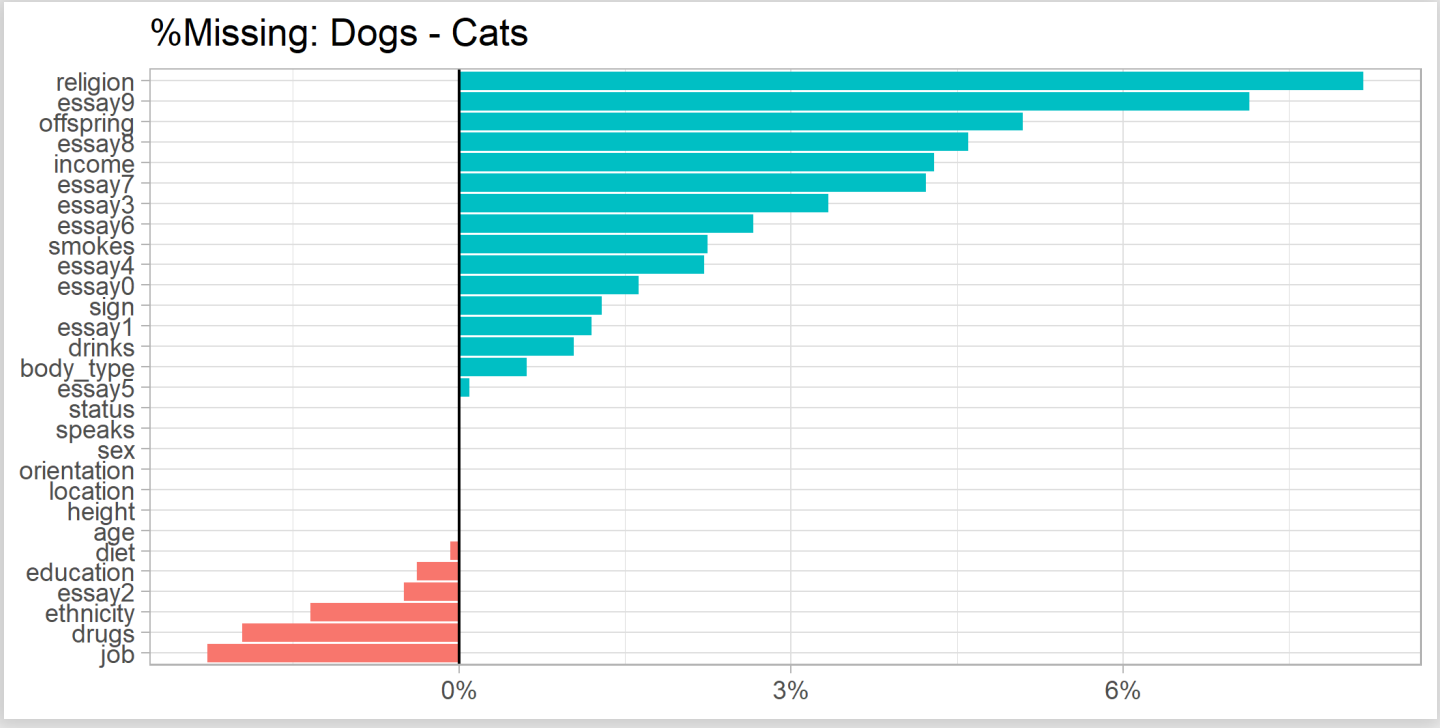


Exploring missingness between a predictor and the response (categorical) variable

```
ok_tr_missing %>%
    mutate(income_missing = is.na(income)) %>%
    count(income_missing, pets) %>%
  pivot_wider(id_cols = pets,
              names_from = income_missing,
              values_from = n) %>%
  mutate(pct_missing = `TRUE` / (`TRUE` + `FALSE`))
```

```
## # A tibble: 2 × 4
##   pets  `FALSE` `TRUE` pct_missing
##   <chr>   <int>  <int>       <dbl>
## 1 cats       73    256       0.778
## 2 dogs      299   1372       0.821
```

But we can do better



Exploring missingness between two (categorical) predictors

```
library(ggmosaic)

ok_tr_missing %>%
  mutate(body_type_missing = is.na(body_type)) %>%
  ggplot() +
  geom_mosaic(aes(x = product(body_type_missing, status),
                  fill = body_type_missing)) +
  labs(x = "", y = "") + theme_light() + theme(axis.text.y=element
```

You don't always need a plot...

```
ok_tr_missing %>%
    mutate(body_type_missing = is.na(body_type)) %>%
    count(body_type_missing, status) %>%
  pivot_wider(id_cols = status,
              names_from = body_type_missing,
              values_from = n) %>%
  mutate(pct_missing = `TRUE` / (`TRUE` + `FALSE`))
```

```
## # A tibble: 4 × 4
##   status         `FALSE` `TRUE` pct_missing
##   <chr>            <int>  <int>       <dbl>
## 1 available           49      3      0.0577
## 2 married              8      2      0.2
## 3 seeing someone      48     12      0.2
## 4 single            1751    127      0.0676
```

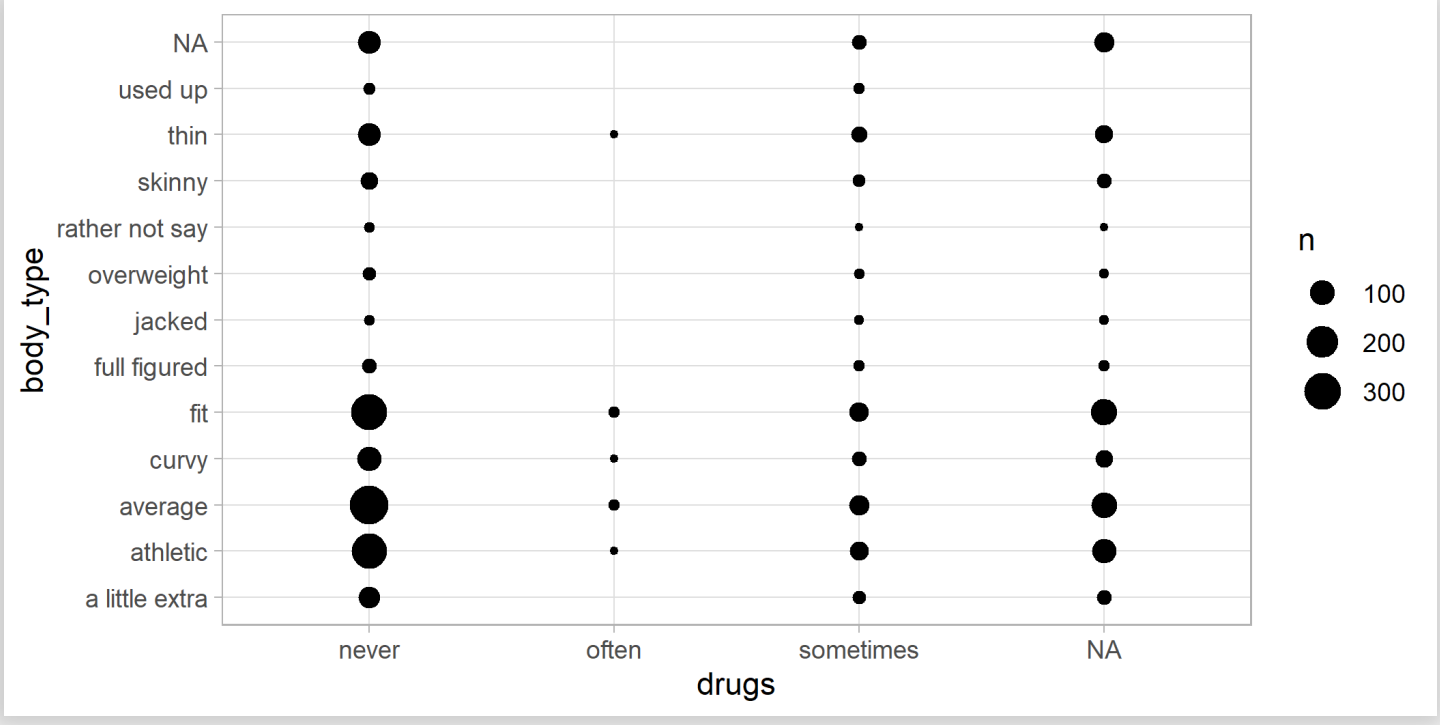
The same with `body_type` missingness by `drugs`:

```
ok_tr_missing %>%
    mutate(body_type_missing = is.na(body_type),
           drugs_missing = is.na(drugs)) %>%
    count(body_type_missing, drugs_missing) %>%
  pivot_wider(id_cols = drugs_missing,
              names_from = body_type_missing,
              values_from = n) %>%
  mutate(pct_missing = `TRUE` / (`TRUE` + `FALSE`))
```

```
## # A tibble: 2 × 4
##   drugs_missing `FALSE` `TRUE` pct_missing
##   <lgl>           <int>  <int>       <dbl>
## 1 FALSE            1423     93      0.0613
## 2 TRUE              433     51      0.105
```

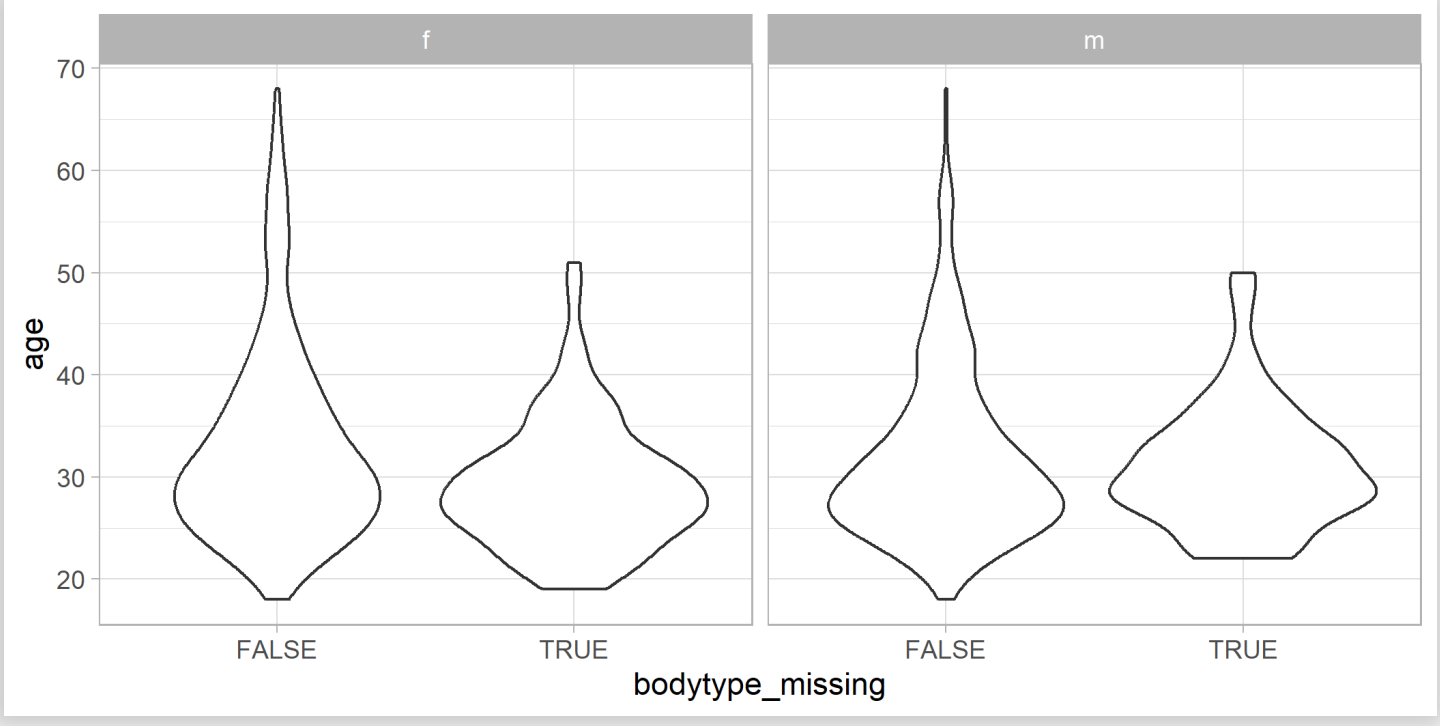
But we can do better with a co-occurrence plot or a heatmap

```
ok_tr_missing %>%
  count(drugs, body_type) %>%
  ggplot(aes(drugs, body_type)) +
  geom_point(aes(size = n)) +
  theme_light()
```



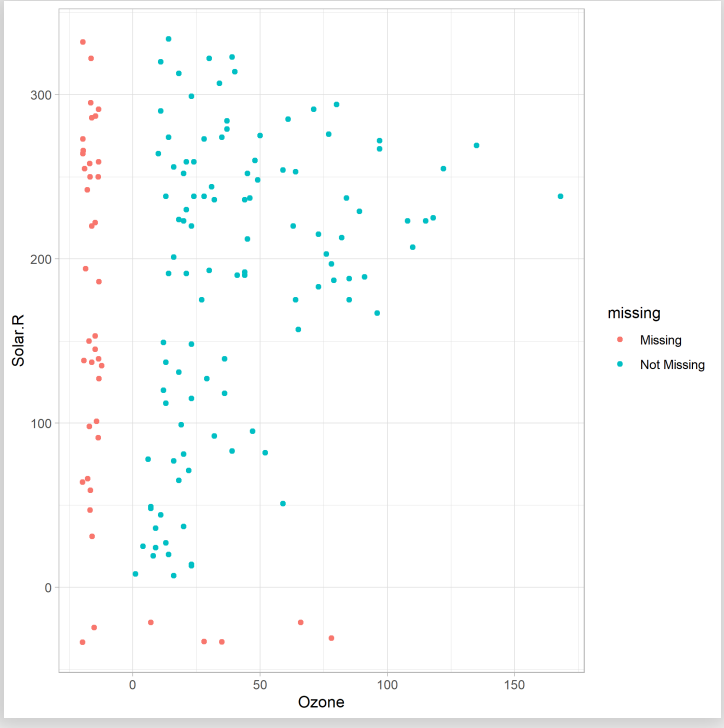
Exploring missingness between numerical and categorical predictors

```
ok_tr_missing %>%
  mutate(bodytype_missing = is.na(body_type)) %>%
  ggplot(aes(bodytype_missing, age)) +
  geom_violin() +
  facet_wrap(. ~ sex) +
  theme_light()
```



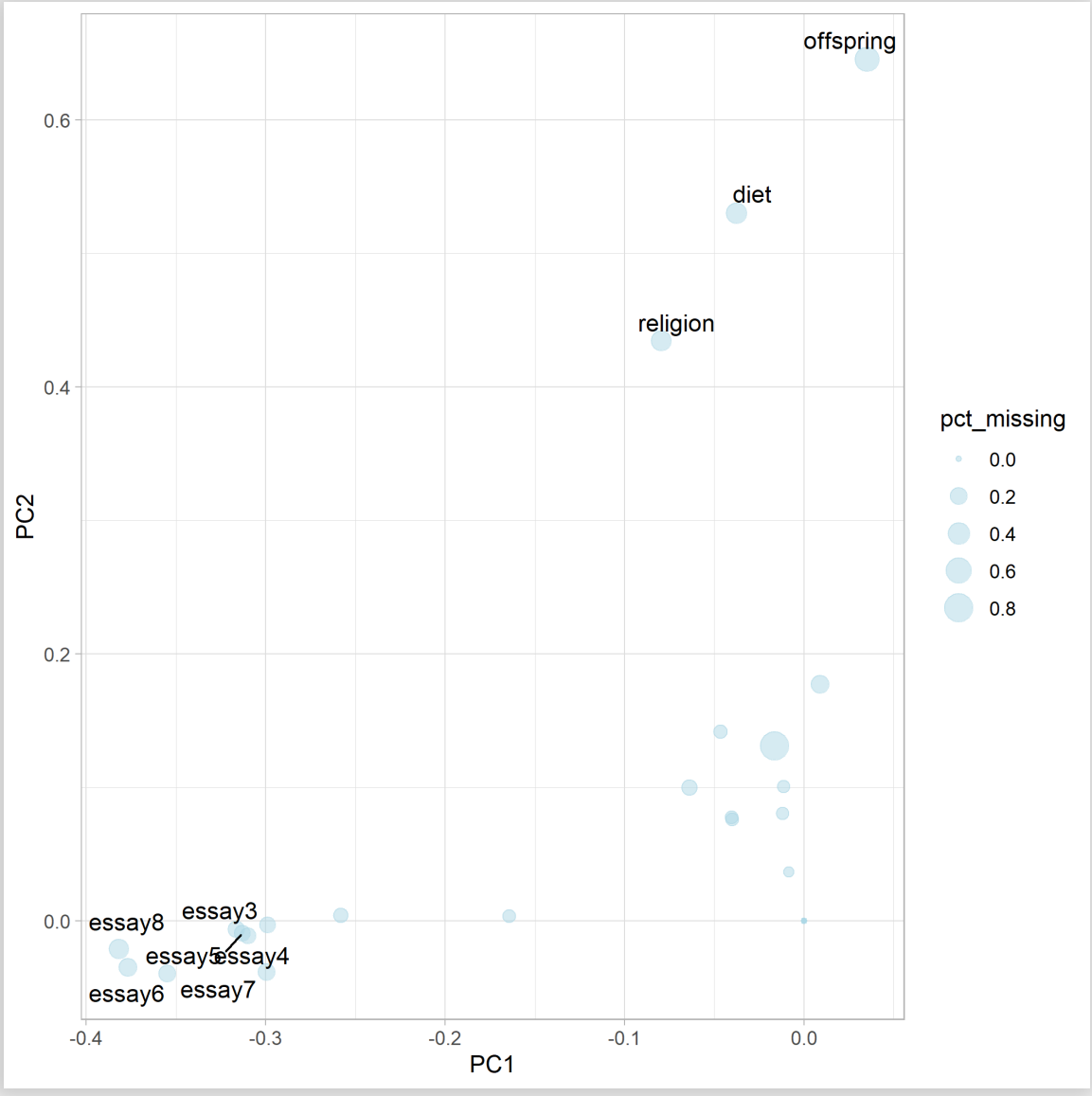
Exploring missingness between two numerical predictors (borrowing `airquality` data)

```
ggplot(airquality, aes(Ozone, Solar.R)) +
  naniar::geom_miss_point() +
  theme_light()
```



When the data is large you could perform PCA on the binary missing/non-missing matrix:

```
binary_mat <- ok_tr_missing %>%
  select(-pets) %>%
  mutate(across(everything(), is.na))

prcomp(binary_mat) %>%
  pluck("rotation") %>%
  as_tibble() %>%
  select(PC1, PC2) %>%
  bind_cols(
    tibble(
      predictor = colnames(binary_mat),
      pct_missing = binary_mat %>% colSums() / nrow(ok_tr_missing)
    )
  ) %>%
  mutate(label = ifelse(PC1 < -0.3 | PC2 > 0.3, predictor, "")) %>
  ggplot(aes(PC1, PC2)) +
  geom_point(aes(size = pct_missing), color = "lightblue", alpha =
  ggrepel::geom_text_repel(aes(label = label)) +
  theme_light()
```


offspring
diet
religion
essay8
essay3
essay5
essay4
essay6
essay7
0.6
0.4
0.2
0.0
PC2
-0.4
-0.3
-0.2
-0.1
0.0
PC1
pct_missing
0.0
0.2
0.4
0.6
0.8

Definitely not in MCAR land

- Keep the data missing, drop it all or impute values?
  - Very much depends on the mechanism of missing data
  - "Drop it all" only if you're "data rich" and assume MCAR
  - Keep or impute: can your model handle missing values?
  - "It's not a bug, it's a feature!": a new category or an additional variable
  - Use resampling to test the best strategy!

- How to impute?
  - Very much depends on the mechanism of missing data
  - Mean/median value if MCAR
  - kNN imputation
  - Iterative Regression/Classification algos (EM algorithm)
  - Use resampling to test the best strategy!
- What do you do with missing values in the response variable?
  - Semi-supervised Learning
  - Active Learning

Let's choose between 2 x 3 x 3 = 18 strategies:

1. Missing categorical features
    - adding an "unknown" category
    - imputing the most common category
2. The `income` feature:
    - Dropping it (it is 80% missing!!)
    - Log transformation, imputing the mean
    - Log transformation, kNN Imputation
3. The `essay` features:
    - Dropping them
    - Adding `essay` length where a missing `essay` has length 0
    - Adding `essay` length where a missing `essay` has length 0 and adding "is_missing" feature to each `essay`

Use `step_mutate_*()` or add what you need beforehand.

```
ok_tr_missing2 <- ok_tr_missing %>%
  mutate(across(essay0:essay9,
                list(
                  "len" = ~ifelse(is.na(.x), 0, str_length(.x)),
                  "isna" = ~ifelse(is.na(.x), 1, 0)
                  ))) %>%
  rename_with(~paste("len", gsub("_len", "", .), sep = "_"),
              ends_with("_len")) %>%
  rename_with(~paste("isna", gsub("_is_na", "", .), sep = "_"),
              ends_with("_isna")) %>%
  select(-starts_with("essay")) %>%
  mutate(across(where(is.character), as.factor))
```

Defining recipes (without `prep()`!), let's start with 2 to simplify.

```
rec_miss_unk_none_none <- recipe(pets ~ ., data = ok_tr_missing2)
  step_rm(income, contains("essay")) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), other = "other1") %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors(), one_hot = FALSE)

rec_miss_mode_none_none <- recipe(pets ~ ., data = ok_tr_missing2)
  step_rm(income, contains("essay")) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), other = "other1") %>%
  step_novel(all_nominal_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors(), one_hot = FALSE)
```

Defining resamples

```
set.seed(42)
cv_splits_missing <- vfold_cv(ok_tr_missing2, v = 10, strata = pet

cv_splits_missing
```

```
## #  10-fold cross-validation using stratification
## # A tibble: 10 × 2
##    splits             id
##    <list>             <chr>
##  1 <split [1799/201]> Fold01
##  2 <split [1800/200]> Fold02
##  3 <split [1800/200]> Fold03
##  4 <split [1800/200]> Fold04
##  5 <split [1800/200]> Fold05
##  6 <split [1800/200]> Fold06
##  7 <split [1800/200]> Fold07
##  8 <split [1800/200]> Fold08
##  9 <split [1800/200]> Fold09
## 10 <split [1801/199]> Fold10
```

For educational purposes Applying (two) recipes to all splits with `prepper()`:

```
cv_splits_missing$rec_miss_unk_none_none <- map(
  cv_splits_missing$splits, prepper, recipe = rec_miss_unk_none_no
cv_splits_missing$rec_miss_mode_none_none <- map(
  cv_splits_missing$splits, prepper, recipe = rec_miss_mode_none_r

map_dbl(cv_splits_missing$rec_miss_unk_none_none, ~nrow(.$term_inf
```

```
##  [1] 92 91 93 93 92 91 90 91 92 91
```

```
cv_splits_missing$rec_miss_unk_none_none[[1]]$term_info %>%
  filter(str_detect(variable, "speaks"))
```

```
## # A tibble: 4 × 4
##   variable                type      role      source
##   <chr>                   <list>    <chr>     <chr>
## 1 speaks_english..fluently. <chr [2]> predictor derived
## 2 speaks_other1           <chr [2]> predictor derived
## 3 speaks_new              <chr [2]> predictor derived
## 4 speaks_unknown          <chr [2]> predictor derived
```

Notice the `role` and `source` columns.

Defining the model specification

```
mod_lr <- logistic_reg(penalty = 0.01) %>%
  set_engine("glmnet")
```

Use `workflow_set()` to fit the two recipes on all folds:

```
rec_list <- list(unk = rec_miss_unk_none_none,
                 mode = rec_miss_mode_none_none)
mod_list <- list(lr = mod_lr)
wset <- workflow_set(rec_list, mod_list)
# also check out the cross arg

wset
```

```
## # A workflow set/tibble: 2 × 4
##   wflow_id info             option    result
##   <chr>    <list>           <list>    <list>
## 1 unk_lr   <tibble [1 × 4]> <opts[0]> <list [0]>
## 2 mode_lr  <tibble [1 × 4]> <opts[0]> <list [0]>
```

```
wset <- wset %>%
  workflow_map("fit_resamples",
               resamples = cv_splits_missing,
               metrics = metric_set(roc_auc))

wset
```

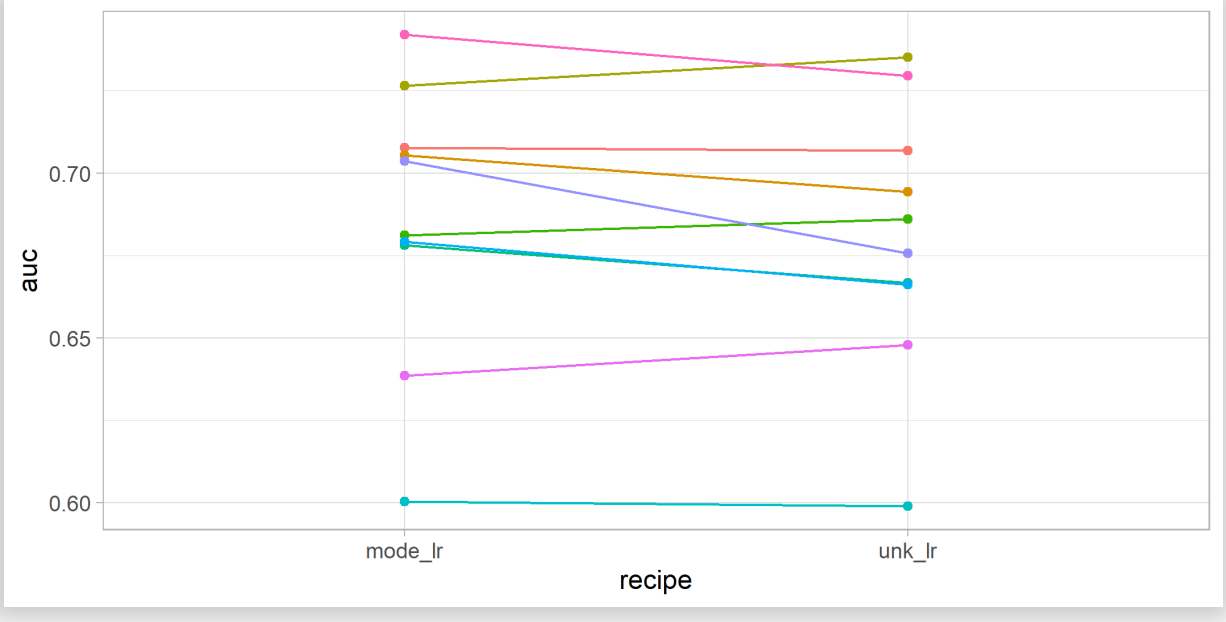
```
## # A workflow set/tibble: 2 × 4
##   wflow_id info             option    result
##   <chr>    <list>           <list>    <list>
## 1 unk_lr   <tibble [1 × 4]> <opts[2]> <rsmp[+]>
## 2 mode_lr  <tibble [1 × 4]> <opts[2]> <rsmp[+]>
```

Can use built-in `autoplot()` to see the results, I dislike it:

```
library(ggrepel)

autoplot(wset, metric = "roc_auc") +
  geom_text_repel(aes(label = wflow_id), nudge_x = 1/8, nudge_y =
  theme_light() +
  theme(legend.position = "none")
```

A type of parallel coordinates plot showing all folds is better IMO:

```
res_df <- wset %>%
  unnest(result) %>%
  mutate(recipe = wflow_id, auc = map_dbl(.metrics, ~.x$.estimate))

res_df %>%
  ggplot(aes(recipe, auc, group = id, color = id)) +
  geom_line() +
  geom_point() +
  guides(color = "none") + theme_light()
```



Comparing AUCs with a statistical test (paired T test)

```
t.test(with(res_df, auc[recipe == "unk_lr"]),
       with(res_df, auc[recipe == "mode_lr"]), paired = TRUE)
```

```
##
##     Paired t-test
##
## data:  with(res_df, auc[recipe == "unk_lr"]) and with(res_df, auc[recipe
## t = -1.5003, df = 9, p-value = 0.1678
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  -0.013950063  0.002824533
## sample estimates:
## mean difference
##    -0.005562765
```

Let's move on to 2 x 2 recipes, adding 2 more recipes for treating the `income` variable with mean imputation, with `step_unknown()` and `step_impute_mode()`.

```
rec_miss_unk_mean_none <- recipe(pets ~ ., data = ok_tr_missing2)
  step_rm(contains("essay")) %>%
  step_log(income, offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), other = "other1") %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors(), one_hot = FALSE)

rec_miss_mode_mean_none <- recipe(pets ~ ., data = ok_tr_missing2)
  step_rm(contains("essay")) %>%
  step_log(income, offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), other = "other1") %>%
  step_novel(all_nominal_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors(), one_hot = FALSE)
```

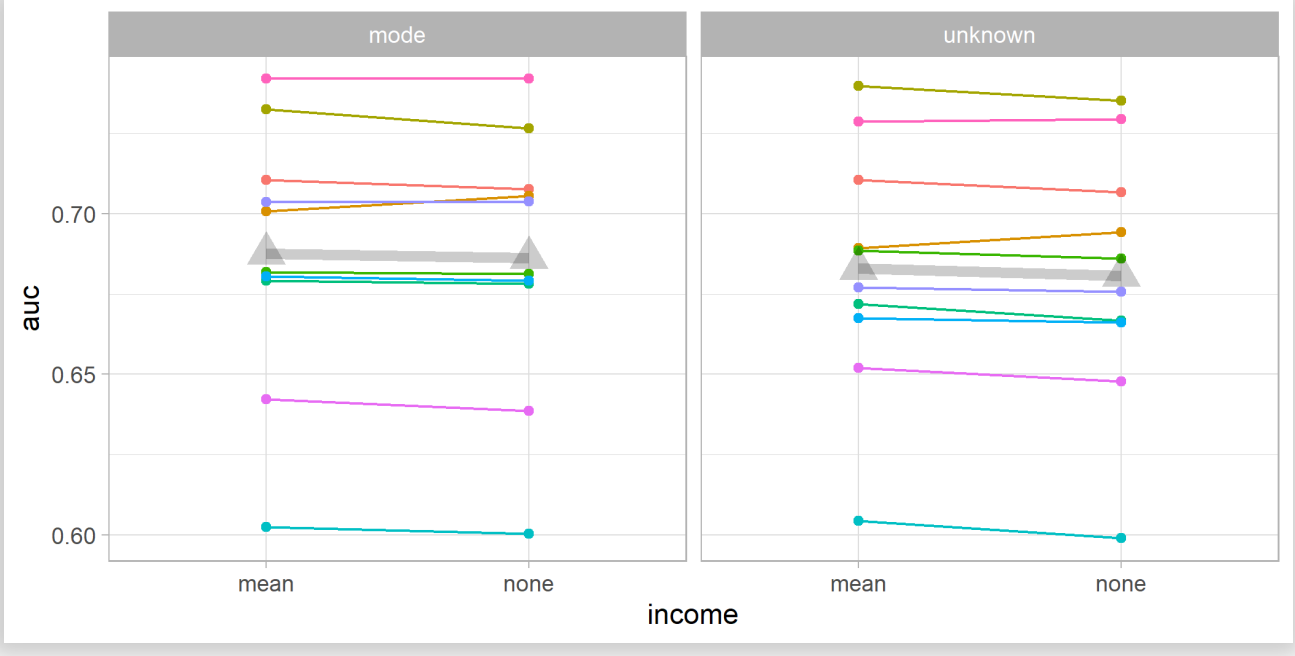
And again, using `workflow_set()`, more concise:

```
rec_list <- list("unknown_none" = rec_miss_unk_none_none,
                 "mode_none" = rec_miss_mode_none_none,
                 "unknown_mean" = rec_miss_unk_mean_none,
                 "mode_mean" = rec_miss_mode_mean_none)

wset <- workflow_set(rec_list, mod_list) %>%
  workflow_map("fit_resamples",
               resamples = cv_splits_missing,
               metrics = metric_set(roc_auc))

cv_res_missing <- wset %>%
  unnest(result) %>%
  mutate(recipe = wflow_id, auc = map_dbl(.metrics, ~.x$.estimate))

cv_res_missing %>% select(wflow_id, id, auc)
```

```
## # A tibble: 40 × 3
##    wflow_id          id       auc
##    <chr>             <chr>  <dbl>
##  1 unknown_none_lr Fold01 0.707
##  2 unknown_none_lr Fold02 0.694
##  3 unknown_none_lr Fold03 0.735
##  4 unknown_none_lr Fold04 0.686
##  5 unknown_none_lr Fold05 0.667
##  6 unknown_none_lr Fold06 0.599
##  7 unknown_none_lr Fold07 0.666
```

Comparing AUCs with a plot:

```
cv_res_missing %>%
  separate_wider_delim(wflow_id, "_", names = c("categorical", "ir
  ggplot(aes(income, auc, group = id, color = id)) +
  geom_line() + geom_line(data = cv_res_missing_meds, color = "bla
  geom_point() + geom_point(data = cv_res_missing_meds, color = "b
    guides(color = "none") + facet_wrap(. ~ categorical) +
    theme_light()
```

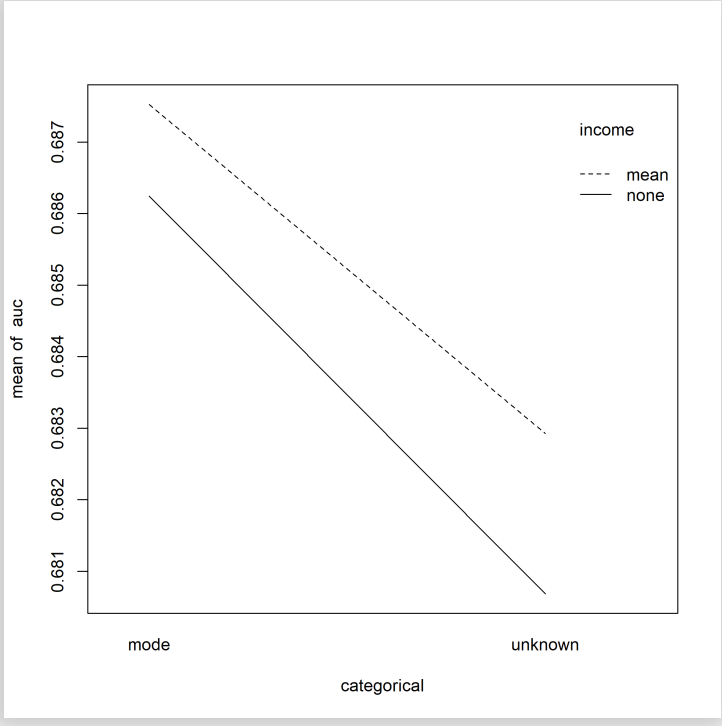


Comparing AUCs with a statistical test (2x2 Repeated measures ANOVA):

```
cv_res_missing <- cv_res_missing %>%
  separate_wider_delim(wflow_id, "_", names = c("categorical", "ir

aov_missing <- aov(
  auc ~ factor(categorical) * factor(income) + Error(factor(id)),
  data = cv_res_missing)
print(summary(aov_missing))
```

```
##
## Error: factor(id)
##           Df  Sum Sq  Mean Sq F value Pr(>F)
## Residuals  9 0.05781 0.006424
##
## Error: Within
##                                    Df    Sum Sq   Mean Sq F value Pr(>F)
## factor(categorical)                 1 0.0002579 2.579e-04   5.333 0.0288
## factor(income)                      1 0.0000310 3.103e-05   0.642 0.4301
## factor(categorical):factor(income)  1 0.0000023 2.340e-06   0.048 0.8275
## Residuals                          27 0.0013059 4.836e-05
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
with(cv_res_missing, interaction.plot(categorical, income, auc))
```



Ok, now let's go back to our original idea of 18 recipes... see code in slides Rmd file.

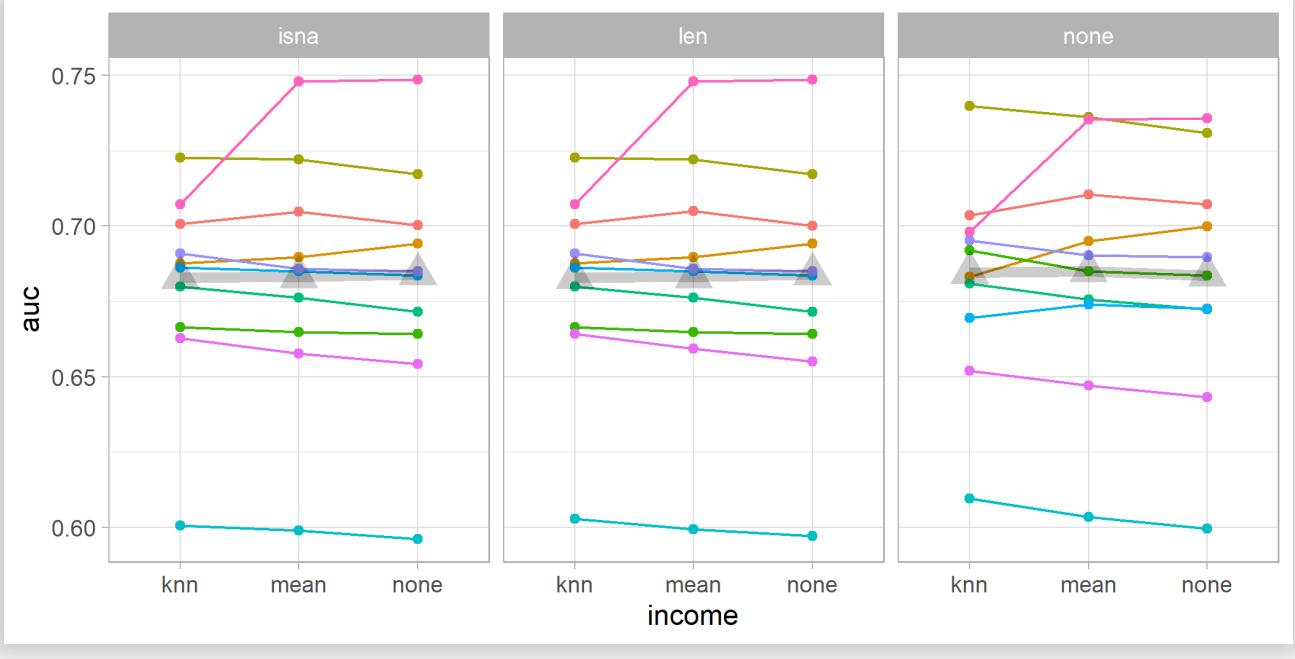
```
rec_list <- list("unknown_none_none" = rec_miss_unk_none_none,
                 "mode_none_none" = rec_miss_mode_none_none,
                 "unknown_mean_none" = rec_miss_unk_mean_none,
                 "mode_mean_none" = rec_miss_mode_mean_none,
                 "unknown_knn_none" = rec_miss_unk_knn_none,
                 "mode_knn_none" = rec_miss_mode_knn_none,
                 "unknown_none_len" = rec_miss_unk_none_len,
                 "mode_none_len" = rec_miss_mode_none_len,
                 "unknown_mean_len" = rec_miss_unk_mean_len,
                 "mode_mean_len" = rec_miss_mode_mean_len,
                 "unknown_knn_len" = rec_miss_unk_knn_len,
                 "mode_knn_len" = rec_miss_mode_knn_len,
                 "unknown_none_isna" = rec_miss_unk_none_isna,
                 "mode_none_isna" = rec_miss_mode_none_isna,
                 "unknown_mean_isna" = rec_miss_unk_mean_isna,
                 "mode_mean_isna" = rec_miss_mode_mean_isna,
                 "unknown_knn_isna" = rec_miss_unk_knn_isna,
                 "mode_knn_isna" = rec_miss_mode_knn_isna
                 )
```

BTW, I just saw each of these is ~700KB before prepping, ridic.

```
set.seed(42)
cv_splits_missing <- vfold_cv(ok_tr_missing2, v = 10, strata = pet

wset <- workflow_set(rec_list, mod_list) %>%
  workflow_map("fit_resamples",
               resamples = cv_splits_missing,
               metrics = metric_set(roc_auc))

cv_res_missing <- wset %>%
  unnest(result) %>%
  mutate(recipe = wflow_id, auc = map_dbl(.metrics, ~.x$.estimate)
  select(wflow_id, id, auc) %>%
  separate_wider_delim(wflow_id, "_", names = c("categorical", "ir
```

Again, see the plot (how would you plot a 2x3x3 experiment results?).

```
cv_res_missing %>%
  group_by(id, income, essay) %>%
  summarise(auc = mean(auc)) %>%
  ggplot(aes(income, auc, group = id, color = id)) +
  geom_line() + geom_line(data = cv_res_missing_meds, color = "bla
  geom_point() + geom_point(data = cv_res_missing_meds, color = "b
  guides(color = "none") + facet_wrap(. ~ essay) +
  theme_light()
```



Repeated measures ANOVA

```
aov_missing <- aov(
  auc ~ factor(categorical) * factor(income) + factor(essay) + Er
  data = cv_res_missing)
summary(aov_missing)
```

```
##
## Error: factor(id)
##           Df Sum Sq Mean Sq F value Pr(>F)
## Residuals  9 0.2215 0.02461
##
## Error: Within
##                                      Df   Sum Sq   Mean Sq F value   Pr(>
## factor(categorical)                   1 0.001649 0.0016491  15.295 0.0001
## factor(income)                        2 0.000239 0.0001193   1.107 0.3330
## factor(essay)                         2 0.000129 0.0000644   0.597 0.5516
## factor(categorical):factor(income)    2 0.000013 0.0000065   0.060 0.9418
## Residuals                           163 0.017574 0.0001078
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Or you can just look at the mean numbers...

```
cv_res_missing %>%
  group_by(categorical, income, essay) %>%
  summarise(auc = mean(auc)) %>%
  arrange(-auc) %>%
  head(10)
```

```
## # A tibble: 10 × 4
## # Groups:   categorical, income [4]
##    categorical income essay   auc
##    <chr>       <chr>  <chr> <dbl>
##  1 mode        mean   none  0.688
##  2 mode        mean   len   0.687
##  3 mode        mean   isna  0.687
##  4 mode        none   none  0.686
##  5 mode        none   len   0.685
##  6 mode        none   isna  0.685
##  7 mode        knn    none  0.685
##  8 mode        knn    len   0.684
##  9 mode        knn    isna  0.683
## 10 unknown     mean   none  0.683
```

💡 So what would you do?

Class Imbalance

Class Imbalance

(We already gave a decent review in last class)

We'll choose between 3 strategies:

- Upsample
- Downsample
- SMOTE and Downsample

Remember how we got here.

```
handle_essays <- function(ok_df) {
  ok_df %>%
    mutate(across(essay0:essay9,
                  list("len" = ~ifelse(is.na(.x), 0, str_length(.x
    rename_with(~paste("len", gsub("_len", "", .), sep = "_"),
                ends_with("_len"))
}

ok_tr_imbalance2 <- handle_essays(ok_tr_imbalance) %>%
  select(-starts_with("essay")) %>%
  mutate(across(where(is.character), as.factor))
```

```r
library(themis)

rec_imb_base <- recipe(pets ~ ., data = ok_tr_imbalance2) %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), other = "other1") %>%
  step_novel(all_nominal_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors(), one_hot = FALSE)

rec_imb_upsample <- rec_imb_base %>%
  step_upsample(pets, over_ratio = 1, seed = 42)

rec_imb_downsample <- rec_imb_base %>%
  step_downsample(pets, under_ratio = 1, seed = 42)

rec_imb_smote <- rec_imb_base %>%
  step_downsample(pets, under_ratio = 1.5) %>%
  step_smote(pets, over_ratio = 1, seed = 42)
```

CV splits, `workflow_set()`.

```
set.seed(42)
cv_splits_imbalance <- vfold_cv(ok_tr_imbalance2, v = 10, strata =

rec_list <- list("upsample" = rec_imb_upsample,
                 "downsample" = rec_imb_downsample,
                 "smote" = rec_imb_smote)

wset <- workflow_set(rec_list, mod_list) %>%
  workflow_map("fit_resamples",
               resamples = cv_splits_imbalance,
               metrics = metric_set(roc_auc))

cv_res_imbalance <- wset %>%
  unnest(result) %>%
  mutate(recipe = wflow_id, auc = map_dbl(.metrics, ~.x$.estimate)
  select(recipe, id, auc)
```

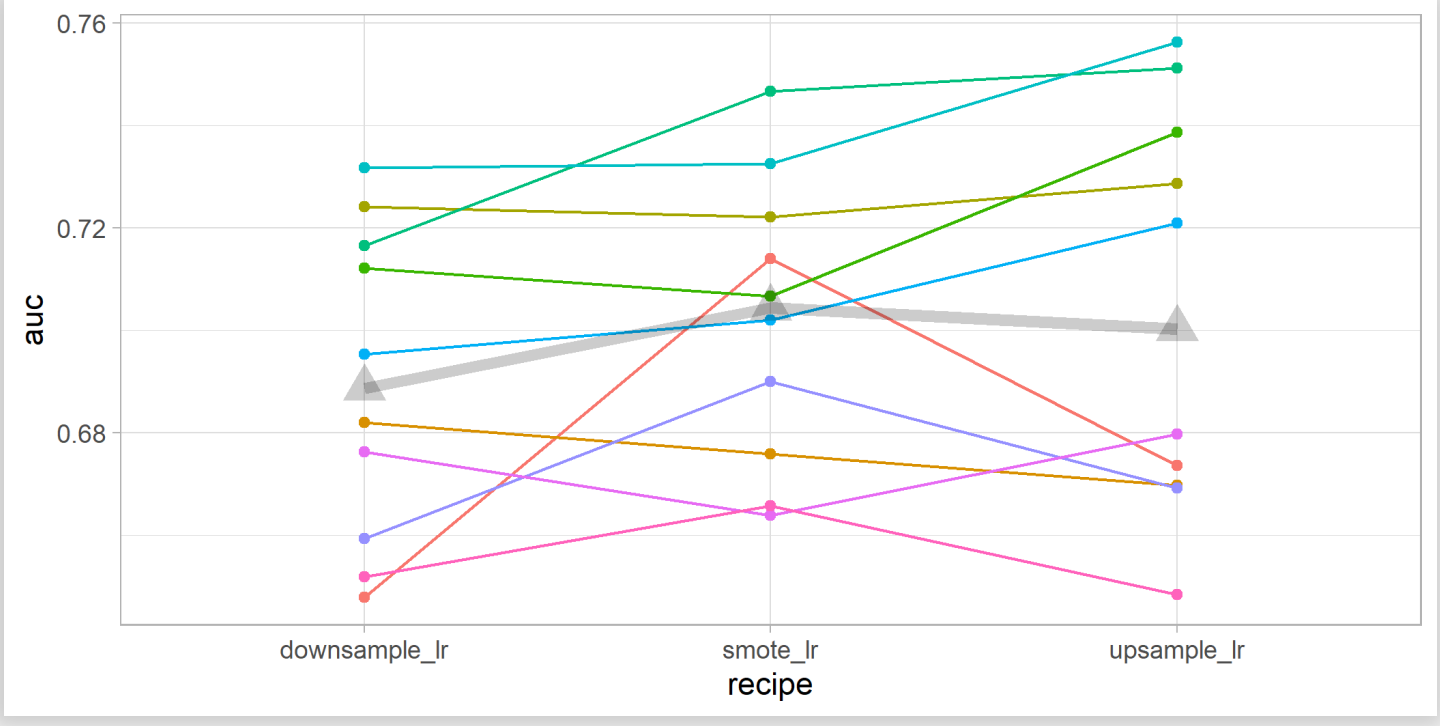
Repeated measures ANOVA:

```
aov_imbalance <- aov(
  auc ~ factor(recipe) + Error(factor(id)),
  data = cv_res_imbalance)
print(summary(aov_imbalance))
```

```
##
## Error: factor(id)
##           Df  Sum Sq  Mean Sq F value Pr(>F)
## Residuals  9 0.02568 0.002854
##
## Error: Within
##                Df   Sum Sq   Mean Sq F value Pr(>F)
## factor(recipe)  2 0.001151 0.0005756   2.587  0.103
## Residuals      18 0.004005 0.0002225
```

Plotting...

```
cv_res_imbalance %>%
  ggplot(aes(recipe, auc, group = id, color = id)) +
    geom_line() + geom_line(data = cv_res_imbalance_meds, color =
  geom_point() + geom_point(data = cv_res_imbalance_meds, color =
    guides(color = "none") +
    theme_light()
```



```
print(cv_res_imbalance %>%
        group_by(recipe) %>%
        summarise(mean_auc = mean(auc), .groups="drop"))
```

```
## # A tibble: 3 × 2
##   recipe        mean_auc
##   <chr>            <dbl>
## 1 downsample_lr    0.690
## 2 smote_lr         0.702
## 3 upsample_lr      0.704
```

💡 So what would you do?

Feat. Engineering and Selection

If anything, remember:

1. Don't be lazy.
2. Be creative.
3. Use resampling!

Let's divide to:

1. Categorical Features
2. Continuous Features
3. Interactions

Categorical Variables

Making dummy variables

(In the case your model can accept categorical levels, skip to the next section - Dealing with categorical variables with many levels)

- One hot encoding: k dummy vars for k levels
- Non one hot encoding: $k-1$ dummy vars for k levels
- Different Contrasts schemes (e.g. for ordered categorical variables ("low", "medium", "high"))
- Feature Hashing

```r
set.seed(42)
df <- tibble(
  x1 = factor(c(rep("a", 4), rep("b", 4))),
  y = runif(8)
)

one_hot <- recipe(y ~ x1, data = df) %>%
  step_dummy(all_nominal(), one_hot = TRUE) %>%
  prep(df) %>%
  bake(new_data = NULL)

non_one_hot <- recipe(y ~ x1, data = df) %>%
  step_dummy(all_nominal(), one_hot = FALSE) %>%
  prep(df) %>%
  bake(new_data = NULL)
```

```
head(one_hot)
```

```
## # A tibble: 6 × 3
##       y  x1_a  x1_b
##   <dbl> <dbl> <dbl>
## 1 0.915     1     0
## 2 0.937     1     0
## 3 0.286     1     0
## 4 0.830     1     0
## 5 0.642     0     1
## 6 0.519     0     1
```

```
head(non_one_hot)
```

```
## # A tibble: 6 × 2
##       y  x1_b
##   <dbl> <dbl>
## 1 0.915     0
## 2 0.937     0
## 3 0.286     0
## 4 0.830     0
## 5 0.642     1
## 6 0.519     1
```

When dealing with linear regression, you should be very careful what you input into the `lm` engine of your choice and how you interpret results!

```
summary(lm(y ~ ., data = non_one_hot))
```

```
##
## Call:
## lm(formula = y ~ ., data = non_one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.7421     0.1434   5.177  0.00206 **
## x1_b         -0.2341     0.2027  -1.155  0.29212
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.1818,    Adjusted R-squared:  0.04546
## F-statistic: 1.333 on 1 and 6 DF,  p-value: 0.2921
```

```
print(mean(df$y) + (mean(df$y[df$x1 == "a"]) - mean(df$y)))
```

This is like:

```
summary(lm(y ~ ., data = df))
```

```
##
## Call:
## lm(formula = y ~ ., data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.7421     0.1434   5.177  0.00206 **
## x1b          -0.2341     0.2027  -1.155  0.29212
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.1818,    Adjusted R-squared:  0.04546
## F-statistic: 1.333 on 1 and 6 DF,  p-value: 0.2921
```

```
summary(lm(y ~ ., data = one_hot))
```

```
##
## Call:
## lm(formula = y ~ ., data = one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.5080     0.1434   3.544   0.0122 *
## x1_a          0.2341     0.2027   1.155   0.2921
## x1_b              NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.1818,    Adjusted R-squared:  0.04546
## F-statistic: 1.333 on 1 and 6 DF,  p-value: 0.2921
```

```
summary(lm(y ~ . - 1, data = one_hot))
```

```
##
## Call:
## lm(formula = y ~ . - 1, data = one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients:
##      Estimate Std. Error t value Pr(>|t|)
## x1_a   0.7421     0.1434   5.177  0.00206 **
## x1_b   0.5080     0.1434   3.544  0.01216 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.8677,    Adjusted R-squared:  0.8236
## F-statistic: 19.68 on 2 and 6 DF,  p-value: 0.002314
```

This is like:

```
summary(lm(y ~ . - 1, data = df))
```

```
##
## Call:
## lm(formula = y ~ . - 1, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients:
##     Estimate Std. Error t value Pr(>|t|)
## x1a   0.7421     0.1434   5.177  0.00206 **
## x1b   0.5080     0.1434   3.544  0.01216 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.8677,    Adjusted R-squared:  0.8236
## F-statistic: 19.68 on 2 and 6 DF,  p-value: 0.002314
```

Plot thickens (check your `options("contrasts")`!):

```
summary(lm(y ~ ., data = df, contrasts = list(x1 = "contr.sum")))
```

```
##
## Call:
## lm(formula = y ~ ., data = df, contrasts = list(x1 = "contr.sum"))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.6251     0.1014   6.167 0.000835 ***
## x11           0.1170     0.1014   1.155 0.292118
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.1818,    Adjusted R-squared:  0.04546
## F-statistic: 1.333 on 1 and 6 DF,  p-value: 0.2921
```

Two variables...

```
set.seed(42)
df <- tibble(
  x1 = factor(c(rep("a", 6), rep("b", 6))),
  x2 = factor(rep(c(rep("c", 2), rep("d", 2), rep("e", 2)), 2)),
  y = runif(12)
)

one_hot <- recipe(y ~ x1 + x2, data = df) %>%
  step_dummy(all_nominal(), one_hot = TRUE) %>%
  prep(df) %>%
  bake(new_data = NULL)

non_one_hot <- recipe(y ~ x1 + x2, data = df) %>%
  step_dummy(all_nominal(), one_hot = FALSE) %>%
  prep(df) %>%
  bake(new_data = NULL)
```

```
head(one_hot)
```

```
## # A tibble: 6 × 6
##       y  x1_a  x1_b  x2_c  x2_d  x2_e
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0.915     1     0     1     0     0
## 2 0.937     1     0     1     0     0
## 3 0.286     1     0     0     1     0
## 4 0.830     1     0     0     1     0
## 5 0.642     1     0     0     0     1
## 6 0.519     1     0     0     0     1
```

```
head(non_one_hot)
```

```
## # A tibble: 6 × 4
##       y  x1_b  x2_d  x2_e
##   <dbl> <dbl> <dbl> <dbl>
## 1 0.915     0     0     0
## 2 0.937     0     0     0
## 3 0.286     0     1     0
## 4 0.830     0     1     0
## 5 0.642     0     0     1
## 6 0.519     0     0     1
```

```
summary(lm(y ~ ., data = non_one_hot))
```

```
##
## Call:
## lm(formula = y ~ ., data = non_one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48619 -0.08138  0.10650  0.15667  0.19636
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.74071    0.15610   4.745  0.00145 **
## x1_b        -0.11986    0.15610  -0.768  0.46465
## x2_d        -0.06112    0.19118  -0.320  0.75738
## x2_e        -0.09636    0.19118  -0.504  0.62783
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2704 on 8 degrees of freedom
## Multiple R-squared:  0.09602,    Adjusted R-squared:  -0.243
## F-statistic: 0.2832 on 3 and 8 DF,  p-value: 0.8362
```

Same as:

```
summary(lm(y ~ ., data = df))
```

```
##
## Call:
## lm(formula = y ~ ., data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48619 -0.08138  0.10650  0.15667  0.19636
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.74071    0.15610   4.745  0.00145 **
## x1b         -0.11986    0.15610  -0.768  0.46465
## x2d         -0.06112    0.19118  -0.320  0.75738
## x2e         -0.09636    0.19118  -0.504  0.62783
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2704 on 8 degrees of freedom
## Multiple R-squared:  0.09602,    Adjusted R-squared:  -0.243
## F-statistic: 0.2832 on 3 and 8 DF,  p-value: 0.8362
```

```
print(mean(df$y) +
        (mean(df$y[df$x1 == "a"]) - mean(df$y)) +
        (mean(df$y[df$x2 == "c"]) - mean(df$y)))
```

```
## [1] 0.7407128
```

```
print(mean(df$y[df$x1 == "a"]) + mean(df$y[df$x2 == "c"]) - mean(d
```

```
## [1] 0.7407128
```

```
print(mean(df$y[df$x1 == "b"]) - mean(df$y[df$x1 == "a"]))
```

```
## [1] -0.1198573
```

```
print(mean(df$y[df$x2 == "d"]) - mean(df$y[df$x2 == "c"]))
```

```
## [1] -0.06112303
```

```
print(mean(df$y[df$x2 == "e"]) - mean(df$y[df$x2 == "c"]))
```

```
## [1] -0.09636022
```

```
summary(lm(y ~ ., data = one_hot))
```

```
##
## Call:
## lm(formula = y ~ ., data = one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48619 -0.08138  0.10650  0.15667  0.19636
##
## Coefficients: (2 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.52450    0.15610   3.360  0.00993 **
## x1_a         0.11986    0.15610   0.768  0.46465
## x1_b             NA         NA      NA       NA
## x2_c         0.09636    0.19118   0.504  0.62783
## x2_d         0.03524    0.19118   0.184  0.85836
## x2_e             NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2704 on 8 degrees of freedom
## Multiple R-squared:  0.09602,    Adjusted R-squared:  -0.243
## F-statistic: 0.2832 on 3 and 8 DF,  p-value: 0.8362
```

```
print(mean(df$y[df$x1 == "b"]) + mean(df$y[df$x2 == "e"]) - mean(d
```

```
## [1] 0.5244952
```

```
print(mean(df$y[df$x1 == "a"]) - mean(df$y[df$x1 == "b"]))
```

```
## [1] 0.1198573
```

```
print(mean(df$y[df$x2 == "c"]) - mean(df$y[df$x2 == "e"]))
```

```
## [1] 0.09636022
```

```
print(mean(df$y[df$x2 == "d"]) - mean(df$y[df$x2 == "e"]))
```

```
## [1] 0.03523718
```

```
summary(lm(y ~ . - 1, data = one_hot))
```

```
##
## Call:
## lm(formula = y ~ . - 1, data = one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48619 -0.08138  0.10650  0.15667  0.19636
##
## Coefficients: (1 not defined because of singularities)
##      Estimate Std. Error t value Pr(>|t|)
## x1_a  0.64435    0.15610   4.128  0.00331 **
## x1_b  0.52450    0.15610   3.360  0.00993 **
## x2_c  0.09636    0.19118   0.504  0.62783
## x2_d  0.03524    0.19118   0.184  0.85836
## x2_e       NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2704 on 8 degrees of freedom
## Multiple R-squared:  0.8914,    Adjusted R-squared:  0.8371
## F-statistic: 16.41 on 4 and 8 DF,  p-value: 0.0006356
```

```
print(mean(df$y[df$x1 == "a"]) + mean(df$y[df$x2 == "e"]) - mean(d
```

```
## [1] 0.6443525
```

```
print(mean(df$y[df$x1 == "b"]) + mean(df$y[df$x2 == "e"]) - mean(d
```

```
## [1] 0.5244952
```

```
print(mean(df$y[df$x2 == "c"]) - mean(df$y[df$x2 == "e"]))
```

```
## [1] 0.09636022
```

```
print(mean(df$y[df$x2 == "d"]) - mean(df$y[df$x2 == "e"]))
```

```
## [1] 0.03523718
```

Categorical variables with many levels

```
ok_tr_eng %>% count(location)
```

```
## # A tibble: 67 × 2
##    location                          n
##    <chr>                         <int>
##  1 alameda, california              49
##  2 albany, california               10
##  3 atherton, california              3
##  4 belmont, california              12
##  5 belvedere tiburon, california     2
##  6 benicia, california              13
##  7 berkeley, california            185
##  8 bolinas, california               1
##  9 brisbane, california              4
## 10 burlingame, california           13
## # … with 57 more rows
```

Many dangers lurk

- OVERFITTING!
- Unseen levels
- Zero variance variables
- Size of p

We'll talk about

- The "Other" and "New" Strategy
- Clustering
- Feature Hashing
- Supervised Numerical Encoding
- Dimensionality Reduction (Embeddings)
- Text Features
- E.g. Simchoni and Rosset (2021, 2023)

The "Other" and "New" Strategy

- "Other": in many scenarios in which a categorical feature has many levels, some of these are so scarce it is doubtful if they have any predictive power - consider grouping them into one "Other" level
- "New": therefore there might be many levels not seen during training, only during testing - consider adding a (zero variance) "New" level

What % threshold should you use for "Other"?

- `step_other()` default is 5% (i.e. any level with less than 5% occurrence is grouped into one super-level "Other")
- or tune it!

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_other(all_nominal_predictors(), threshold = 0.05) %>%
  step_novel(all_nominal_predictors()) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL) %>%
  count(location)
```

```
## # A tibble: 4 × 2
##   location                      n
##   <fct>                     <int>
## 1 berkeley, california        185
## 2 oakland, california         343
## 3 san francisco, california  1615
## 4 other                       857
```

You can also insert some creativity (or plain common sense) in your "Other" strategy:

```
education2 = case_when(
           education == "graduated from high school" ~ "high_schoo
           education == "graduated from two-year college" ~ "colle
           education == "graduated from college/university" ~ "deg
           education == "graduated from masters program" ~ "degree
           education == "graduated from ph.d program" ~ "degree3",
           education == "working on two-year college" ~ "student0"
           education == "working on college/university" ~ "student
           education == "working on masters program" ~ "student2",
           education == "working on ph.d program" ~ "student3",
           is.na(education) ~ NA_character_,
           TRUE ~ "other"
)
```

Clustering

- Manual clustering, using expert knowledge and common sense
- Unsupervised clustering, where it makes sense (e.g. Topic Modeling with LDA on text data)

> 💡 How would you cluster `location`?

The `sign` feature is also a great example:

```
ok_tr_eng %>% count(sign)
```

```
## # A tibble: 47 × 2
##    sign                                                  n
##    <chr>                                             <int>
##  1 aquarius                                             58
##  2 aquarius and it matters a lot                         6
##  3 aquarius and it&rsquo;s fun to think about           70
##  4 aquarius but it doesn&rsquo;t matter                 74
##  5 aries                                                56
##  6 aries and it matters a lot                            1
##  7 aries and it&rsquo;s fun to think about              77
```

```
signs <- c("leo", "libra", "cancer", "virgo", "scorpio", "gemini",
           "taurus", "aries", "pisces", "aquarius", "sagittarius",

recipe(pets ~ ., data = ok_tr_eng %>% select(pets, sign)) %>%
  step_mutate(sign = str_extract(sign, str_c(signs, collapse = "|"
  step_unknown(sign) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL) %>%
  count(sign)
```

```
## # A tibble: 13 × 2
##    sign            n
##    <fct>       <int>
##  1 aries         212
##  2 virgo         232
##  3 leo           223
##  4 taurus        197
##  5 cancer        196
##  6 scorpio       196
##  7 libra         215
##  8 aquarius      208
##  9 gemini        239
## 10 pisces        221
## 11 capricorn     194
## 12 sagittarius   222
## 13 unknown       445
```

Feature Hashing

Suppose I told you "God, just group the locations by the number of times the letter 'l' appears in them!"

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(location = factor(str_count(location, "l"))) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL) %>%
  count(location)
```

```
## # A tibble: 5 × 2
##   location     n
##   <fct>    <int>
## 1 1         1935
## 2 2          875
## 3 3          156
## 4 4           17
## 5 5           17
```

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(locationL = factor(str_count(location, "l"))) %>%
  step_dummy(locationL) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL) %>%
  select(starts_with("location"))
```

```
## # A tibble: 3,000 × 5
##    location                 locationL_X2 locationL_X3 locationL_X4 loca
##    <fct>                           <dbl>        <dbl>        <dbl>
##  1 palo alto, california               0            1            0
##  2 san leandro, california             1            0            0
##  3 hayward, california                 0            0            0
##  4 vallejo, california                 0            1            0
##  5 san francisco, california           0            0            0
##  6 oakland, california                 1            0            0
##  7 berkeley, california                1            0            0
##  8 alameda, california                 1            0            0
##  9 berkeley, california                1            0            0
## 10 san francisco, california           0            0            0
## # … with 2,990 more rows
```

It's not a good idea because

- Differenet `location`s map to the same bin ("collision"), masking possible effect
- The distribution isn't even

But there are advantages too!

- I *did* reduce the dimensionality from 70 to 6 dummy variables
- Fast
- Generalizes well: any new `location` can be mapped this way
- Scales: I can do this for millions of `location`s, no dictionary needed to store in memory

- Feature Hashing (or "hashing trick") does something similar more intelligently
- It lets you choose the hashing algo and no. of features you'd like (the more - the less chance of collision)

```
library(textrecipes)

recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_tokenize(location) %>%
  step_texthash(location, signed = FALSE, num_terms = 2^4) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL)
```

```
## # A tibble: 3,000 × 17
##    texthash_location_01 texthash_location_02 texthash_location_03
##                   <int>                <int>                <int>
##  1                    0                    0                    1
##  2                    0                    0                    1
##  3                    0                    0                    1
##  4                    0                    0                    1
##  5                    0                    0                    1
##  6                    0                    1                    1
##  7                    0                    0                    1
##  8                    0                    0                    1
##  9                    0                    0                    1
## 10                    0                    0                    1
```

- Particularly useful for fast and furious feature engineering of text (see later)
- Beware of columns with zero variance, consider binning them to a "New" category
- Powerful methods like `xgboost` can handle 2^20+ no. of terms easily

> 💡 Why would this work in spite of collisions?
>
> What is still a major disadvantage? Hint: 🎁.

The Washington Post
Democracy Dies in Darkness

Education

‘Creative ... motivating’ and fired

Sarah Wysocki was out of work for only a few days after she was fired by DCPS last year. She is now teaching at Hybla Valley Elementary School in Fairfax County. (Jahi Chikwendiu/The Washington Post)

By **Bill Turque**
March 6, 2012

[Bill Turque, 2012, Washington Post]

Supervised Numerical Encoding

Here's another crazy idea for you:

- For regression, replace a category with y's mean in this category
- For classification, replace a category with the log odds of y in this category

Thus, changing a categorical variable with many levels into a single numerical variable (or a few for a few classes).

```
library(embed)

recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(location2 = location) %>%
  step_lencode_glm(location, outcome = vars(pets)) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL)
```

```
## # A tibble: 3,000 × 3
##    location pets  location2
##       <dbl> <fct> <fct>
##  1    -1.73 dogs  palo alto, california
##  2    -1.58 dogs  san leandro, california
##  3    -1.86 cats  hayward, california
##  4    -2.48 dogs  vallejo, california
##  5    -1.74 dogs  san francisco, california
##  6    -1.47 dogs  oakland, california
##  7    -1.08 dogs  berkeley, california
##  8    -1.49 dogs  alameda, california
##  9    -1.08 cats  berkeley, california
## 10    -1.74 cats  san francisco, california
## # … with 2,990 more rows
```

Make sure you get what just happened

```
mv <- ok_tr_eng$location == "palo alto, california"
p <- mean(ok_tr_eng$pets[mv] == "cats")
log(p / (1 - p))
```

```
## [1] -1.727221
```

But...

- What if a specific location has just 1-2 samples in training set?
- "self-fulfilling prophecy"

The first issue can be somewhat alleviated using a Bayesian approach, *shrinking* somewhat coefficients towards their prior when n is small:

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(location2 = location) %>%
  step_lencode_bayes(location, outcome = vars(pets)) %>%
  prep(ok_tr_eng %>% head(100)) %>%
  bake(new_data = NULL)
```

```
## # A tibble: 100 × 3
##    location pets  location2
##       <dbl> <fct> <fct>
##  1    -1.41 dogs  palo alto, california
##  2    -1.39 dogs  san leandro, california
##  3    -1.36 cats  hayward, california
##  4    -1.43 dogs  vallejo, california
##  5    -1.49 dogs  san francisco, california
##  6    -1.42 dogs  oakland, california
##  7    -1.44 dogs  berkeley, california
```

Dimensionality Reduction (Embeddings)

See our `word2vec` detour when we learned about [Networks Community Detection].

If you have `tensorflow` and `keras` installed, you could train your own embeddings to turn each `location` into a `num_terms` long vector of features:

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(location2 = location) %>%
  step_embed(location, outcome = vars(pets), num_terms = 10,
             options = embed_control(epochs = 10)) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL)
```

Alternatively you could use pre-trained embeddings for text variables such as `essay`s using `step_word_embedding()` from the `textrecipes` package (where `my_embeddings` are your pre-trained embeddings:

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_tokenize(location) %>%
  step_word_embeddings(location, embeddings = my_embeddings) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL)
```

Or, perform PCA on the dummy variables matrix, keeping only a few of the first PCs as features:

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(old_location = location) %>%
  step_dummy(location) %>%
  step_normalize(all_numeric()) %>%
  step_pca(starts_with("location"), num_comp = 10) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL)
```

```
## # A tibble: 3,000 × 12
##    pets  old_location   PC01    PC02    PC03     PC04     PC05     PC06
##    <fct> <fct>         <dbl>   <dbl>   <dbl>    <dbl>    <dbl>    <dbl>
##  1 dogs  palo alto, … -1.28  -0.823   1.33   -1.85    -6.80     1.23
##  2 dogs  san leandro… -1.28  -0.803   1.25   -1.16     2.65     6.85
##  3 cats  hayward, ca… -1.27  -0.773   1.14   -0.720    0.802   -1.04
##  4 dogs  vallejo, ca… -1.26  -0.741   1.04   -0.508    0.455   -0.460
##  5 dogs  san francis…  1.15   0.0647 -0.0406  0.00565 -0.00343  0.00269
##  6 dogs  oakland, ca… -1.61   2.51   -0.588   0.0601  -0.0353   0.0272
##  7 dogs  berkeley, c… -1.40  -1.92   -3.33    0.146   -0.0817   0.0615
##  8 dogs  alameda, ca… -0.445 -0.0625  0.0430 -0.00633  0.00388 -0.00304
##  9 cats  berkeley, c… -1.40  -1.92   -3.33    0.146   -0.0817   0.0615
## 10 cats  san francis…  1.15   0.0647 -0.0406  0.00565 -0.00343  0.00269
## # … with 2,990 more rows, and 3 more variables: PC08 <dbl>, PC09 <dbl>,
## #   PC10 <dbl>
```

Text Features

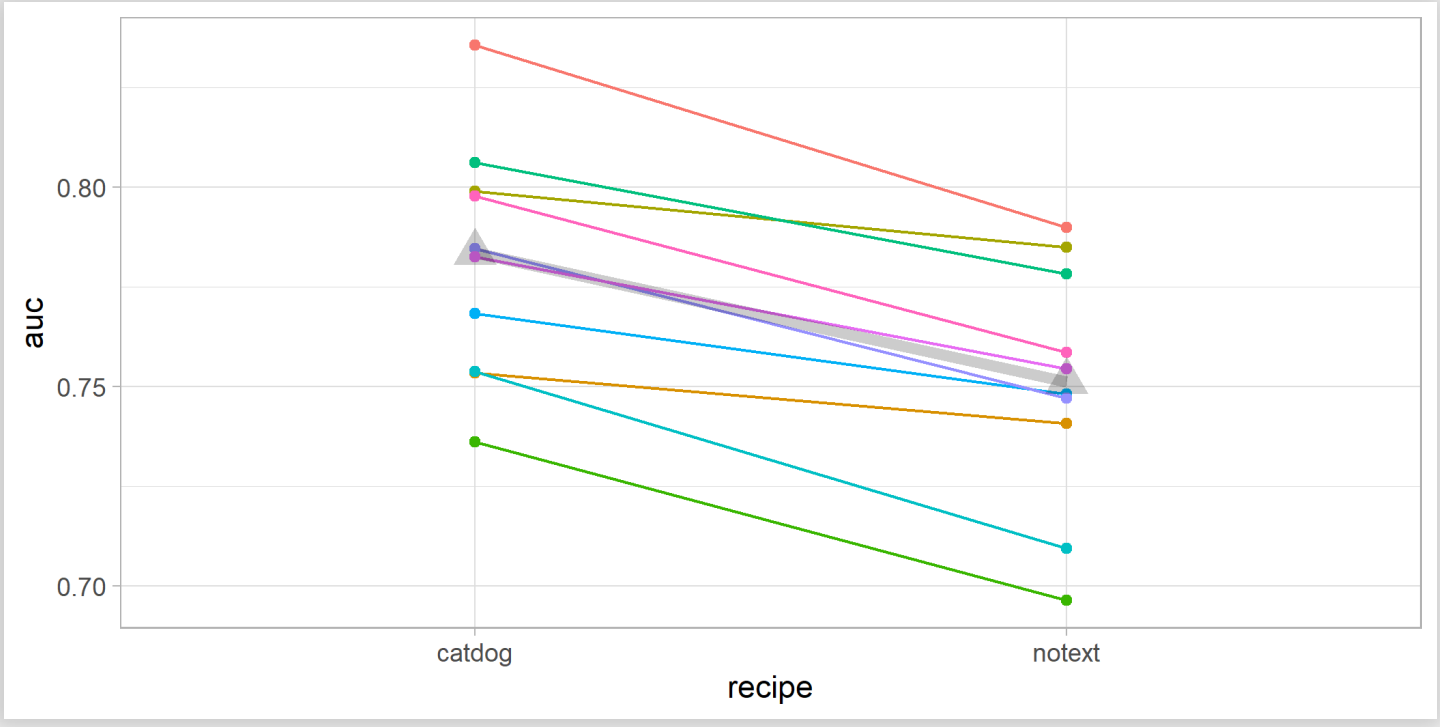
Extracting textual features could be the subject of an entire class or even a course.

Just to demonstrate how worthwhile being creative can get here let us just add two textual features here, `n_cat` and `n_dog` in all `essay`s:

```
ok_tr_eng2 <- handle_essays(ok_tr_eng)

str_count_na <- function(string, pattern = "") {
  n <- str_count(string, pattern)
  ifelse(is.na(n), 0, n)
}
```

```
rec_eng_notext <- recipe(pets ~ ., data = ok_tr_eng2) %>%
  step_rm(starts_with("essay")) %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), other = "other1") %>%
  step_novel(all_nominal_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors(), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```
rec_eng_catdog <- recipe(pets ~ ., data = ok_tr_eng2) %>%
  step_mutate(essays = pmap_chr(
    list(essay0, essay1, essay2, essay3, essay4, essay5,
         essay6, essay7, essay8, essay9), str_c),
              n_cat = str_count_na(essays, "cat"),
              n_dog = str_count_na(essays, "dog")) %>%
  step_rm(starts_with("essay")) %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), other = "other1") %>%
  step_novel(all_nominal_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors(), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```


0.80
0.75
0.70
auc
catdog
notext
recipe

Text Features: some ideas for you

- not just words: do cat people tend to use more the letter "z"? no. of commas, no. of dot-dot-dots, no. of links, no. of hashtags, no. of extra spaces, no. of digits
- to stem or not to stem?
  - "plays", "playing", "playa"
- sentiment features
  - not just no. of "hate" vs. "love"
  - also no. of "you" and "I"
- stop words or not to stop words?
- TF-IDF or not to TF-IDF
- Feature Hashing
- n-grams: "I love my cats"
- topics

Using `textrecipes` + `textfeatures`

- The `textrecipes` package lets you calculate a few features with a single `step_textfeature()`
- The `extract_functions` parameter accepts a named list of functions you wish to define on your text data
- By default it will accept `count_functions` list from the `textfeatures` package by Mike Kearney, which gets you basic counts like `n_words`, `n_digits` etc.

But `textfeatures` also has via the `textfeatures()` function:

- basic sentiment features like `sent_bing`
- first word2vec 2-200 dimensions!

```
print(textfeatures:::sentiment_bing("I hate cats!"))
```

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

```
print(textfeatures:::sentiment_bing("I love cats! Hate dogs though
```

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

```
print(textfeatures::word_dims(
  "Why would Kim Jong-un insult me by calling me \"old,\""))
```

```
## INFO  [09:33:49.248] early stopping at 20 iteration
## INFO  [09:33:49.290] early stopping at 20 iteration
##     w1   w2   w3  w4 w5   w6 w7 w8 w9 w10
## 1 0.05 0.05 0.05 0.4  0 0.35  0  0  0 0.1
```

```
ok_tr_eng2 <- ok_tr_eng2 %>%
  mutate(essays = pmap_chr(
    list(essay0, essay1, essay2, essay3, essay4, essay5,
         essay6, essay7, essay8, essay9), str_c))

my_text_funs <- c(textfeatures::count_functions,
                  "sent_bing" = textfeatures:::sentiment_bing)
```

```
recipe(pets ~ ., data = ok_tr_eng2 %>% select(pets, essays)) %>%
  update_role(essays, new_role = "discarded") %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  prep(ok_tr_eng2 %>% head(10)) %>%
  bake(new_data = NULL) %>%
  select(t_essays_n_uq_words, t_essays_n_hashtags, t_essays_sent_b
```

```
## # A tibble: 10 × 3
##    t_essays_n_uq_words t_essays_n_hashtags t_essays_sent_bing
##                  <dbl>               <dbl>              <dbl>
##  1                   0                   0                  0
##  2                 283                   0                  9
##  3                 345                   0                 16
##  4                   0                   0                  0
##  5                   0                   0                  0
##  6                   0                   0                  0
##  7                   0                   0                  0
##  8                 267                   0                 22
##  9                   0                   0                  0
## 10                 193                   0                 -2
```

Text Features: but the words are most important

We'll take Kuhn & Johnson (2019) approach, and:

1. Filter 1-gram tokens (words) with at least X occurrences
2. Calculate the odds ratio $\frac{P(w|cats)}{1-P(w|cats)} / \frac{P(w|dogs)}{1-P(w|dogs)}$ where $P(w|cats)$ is the % of cats people who used the word w at least once in all essays
3. Calculate a p-value with a FDR correction
4. Choose words with ~~OR of at least |2| and~~ FDR value of maximum 10^{-4}

Then use resampling to see if these have any effect on AUC.

```r
library(tidytext)

tidy_essays <- ok_tr_eng2 %>%
  mutate(id = 1:n()) %>%
  select(id, pets, essays) %>%
  unnest_tokens(word, essays) %>%
  filter(!word %in% stop_words$word,
         str_detect(word, "[a-z]"))

tidy_essays %>% head()
```

```
## # A tibble: 6 × 3
##      id pets  word
##   <int> <chr> <chr>
## 1     2 dogs  bay
## 2     2 dogs  raised
## 3     2 dogs  university
## 4     2 dogs  oregon
## 5     2 dogs  schooled
## 6     2 dogs  hip
```

```r
min_n_words <- tidy_essays %>%
  count(id, word) %>%
  count(word) %>%
  filter(n > 50) %>%
  pull(word)

print(c("cat", "dog", "cats", "dogs") %in% min_n_words)
```

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

```r
n_cats_people <- sum(ok_tr_eng2$pets == "cats")
n_dogs_people <- sum(ok_tr_eng2$pets == "dogs")

fisher_test <- function(cats, dogs) {
  cont_tab <- matrix(c(cats, n_cats_people - cats,
                       dogs, n_dogs_people - dogs),
                     ncol = 2, nrow = 2)
  ft <- fisher.test(cont_tab)
  list(OR = ft$estimate, p_val = ft$p.value)
}
```

```
or_res <- tidy_essays %>%
  filter(word %in% min_n_words) %>%
  count(id, word, pets) %>%
  count(word, pets) %>%
  pivot_wider(id_cols = word, names_from = pets,
              values_from = n, values_fill = list(cats = 0, dogs =
  mutate(fisher_obj = map2(cats, dogs, fisher_test))

or_res <- or_res %>%
  bind_cols(
    bind_rows(or_res$fisher_obj)
  ) %>%
  mutate(fdr = p.adjust(p_val, method = "fdr"))

or_res
```

```
## # A tibble: 818 × 7
##    word        cats  dogs fisher_obj          OR  p_val   fdr
##    <chr>      <int> <int> <list>           <dbl>  <dbl> <dbl>
##  1 ability       17    47 <named list [2]> 1.86  0.0392 0.163
##  2 absolutely    11    75 <named list [2]> 0.736 0.460  0.683
##  3 accent        12    39 <named list [2]> 1.57  0.182  0.398
##  4 act           12    44 <named list [2]> 1.39  0.361  0.603
##  5 action        19   105 <named list [2]> 0.912 0.805  0.928
##  6 active        21   126 <named list [2]> 0.837 0.568  0.784
##  7 activities    15    87 <named list [2]> 0.869 0.686  0.853
##  8 add           15    55 <named list [2]> 1.39  0.255  0.486
##  9 adele          8    47 <named list [2]> 0.859 0.855  0.958
## 10 admit         26   105 <named list [2]> 1.27  0.280  0.513
```

```
cd_words <- or_res %>%
  arrange(fdr) %>%
  filter(fdr < 10e-4)

cd_words
```

```
## # A tibble: 10 × 7
##    word     cats  dogs fisher_obj          OR    p_val      fdr
##    <chr>   <int> <int> <list>           <dbl>    <dbl>    <dbl>
##  1 cat        44    33 <named list [2]> 7.30  2.11e-16 1.73e-13
##  2 dog         7   248 <named list [2]> 0.131 7.65e-13 3.13e-10
##  3 cats       34    34 <named list [2]> 5.36  1.01e-10 2.74e- 8
##  4 ilink      58   100 <named list [2]> 3.19  3.12e-10 6.38e- 8
##  5 class      69   137 <named list [2]> 2.80  5.85e-10 9.58e- 8
##  6 href       61   114 <named list [2]> 2.95  8.41e-10 1.15e- 7
##  7 science    47    84 <named list [2]> 3.02  4.14e- 8 4.84e- 6
##  8 eternal    26    40 <named list [2]> 3.41  5.85e- 6 5.98e- 4
##  9 fantasy    24    35 <named list [2]> 3.59  6.99e- 6 6.35e- 4
## 10 drawing    22    30 <named list [2]> 3.83  7.95e- 6 6.50e- 4
```

So these will be our four options for handling text:

1. Not
2. Text features (zero for `NA`)
3. Text features + our flagged words counts (zero for `NA`)
4. Feature Hashing

```
count_words <- function(essays) {
  counts <- map(cd_words$word, ~str_count_na(essays, .x))
  names(counts) = janitor::make_clean_names(str_c("w_", cd_words$v
  counts
}

ok_tr_eng3 <- ok_tr_eng2 %>%
  bind_cols(
    map_dfr(ok_tr_eng2$essays, count_words)
  ) %>%
  select(-starts_with("essay"), -starts_with("len_essay")) %>%
  bind_cols(ok_tr_eng2 %>% select(essays)) %>%
  mutate(across(where(is.character), as.factor))
```

```
rec_eng_notext <- recipe(pets ~ ., data = ok_tr_eng3) %>%
  step_rm(starts_with("w_")) %>%
  update_role(essays, new_role = "discarded") %>%
  step_log(income, offset = 1) %>%
  step_impute_mean(income) %>%
  step_other(all_nominal_predictors(), -has_role("discarded"), oth
  step_novel(all_nominal_predictors(), -has_role("discarded")) %>%
  step_impute_mode(all_nominal_predictors(), -has_role("discarded"
  step_dummy(all_nominal_predictors(), -has_role("discarded"), one
  step_upsample(pets, over_ratio = 1, seed = 42)

rec_eng_textfeat <- recipe(pets ~ ., data = ok_tr_eng3) %>%
  step_rm(starts_with("w_")) %>%
  update_role(essays, new_role = "discarded") %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("t_"), -t_essays_sent_bing, offset
  step_impute_mean(income) %>%
  step_zv(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), -has_role("discarded"), oth
  step_novel(all_nominal_predictors(), -has_role("discarded")) %>%
  step_impute_mode(all_nominal_predictors(), -has_role("discarded"
  step_dummy(all_nominal_predictors(), -has_role("discarded"), one
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```
rec_eng_fulltext <- recipe(pets ~ ., data = ok_tr_eng3) %>%
  update_role(essays, new_role = "discarded") %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("t_"), -t_essays_sent_bing, offset
  step_impute_mean(income) %>%
  step_zv(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), -has_role("discarded"), oth
  step_novel(all_nominal_predictors(), -has_role("discarded")) %>%
  step_impute_mode(all_nominal_predictors(), -has_role("discarded"
  step_dummy(all_nominal_predictors(), -has_role("discarded"), one
  step_upsample(pets, over_ratio = 1, seed = 42)

rec_eng_hashtext <- recipe(pets ~ ., data = ok_tr_eng3) %>%
  step_rm(starts_with("w_")) %>%
  update_role(essays, new_role = "discarded") %>%
  step_tokenize(essays) %>%
  step_tokenfilter(essays, min_times = 10) %>%
  step_texthash(essays, num_terms = 2^11) %>%
  step_log(income, offset = 1) %>%
  step_impute_mean(income) %>%
  step_zv(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), -has_role("discarded"), oth
  step_novel(all_nominal_predictors(), -has_role("discarded")) %>%
  step_impute_mode(all_nominal_predictors(), -has_role("discarded"
  step_dummy(all_nominal_predictors(), -has_role("discarded"), one
  step_upsample(pets, over_ratio = 1, seed = 42)
```

Using resampling, `workflow_set()`:

```
set.seed(42)
cv_splits_eng <- vfold_cv(ok_tr_eng3, v = 10, strata = pets)

rec_list <- list("notext" = rec_eng_notext,
                 "textfeat" = rec_eng_textfeat,
                 "fulltext" = rec_eng_fulltext,
                 "hashtext" = rec_eng_hashtext)

wset <- workflow_set(rec_list, mod_list) %>%
  workflow_map("fit_resamples",
               resamples = cv_splits_eng,
               metrics = metric_set(roc_auc))

cv_res_eng <- wset %>%
  unnest(result) %>%
  mutate(recipe = wflow_id, auc = map_dbl(.metrics, ~.x$.estimate)
  select(recipe, id, auc) %>%
  separate_wider_delim(recipe, "_", names = c("recipe", "mod"))
```

Haven't forgot about our other categorical variables

Let's use some common sense (see code!).

```
cd_words <- cd_words %>%
  bind_rows(
    tibble(word =
             c("my cat", "my dog", "love cat", "love dog", "kitt",
               "have cat", "have dog", "have a cat", "have a dog"),
           cats = NULL, dogs = NULL, fisher_obj = NULL, OR = 0,
           p_val = 0, fdr = 0))
```

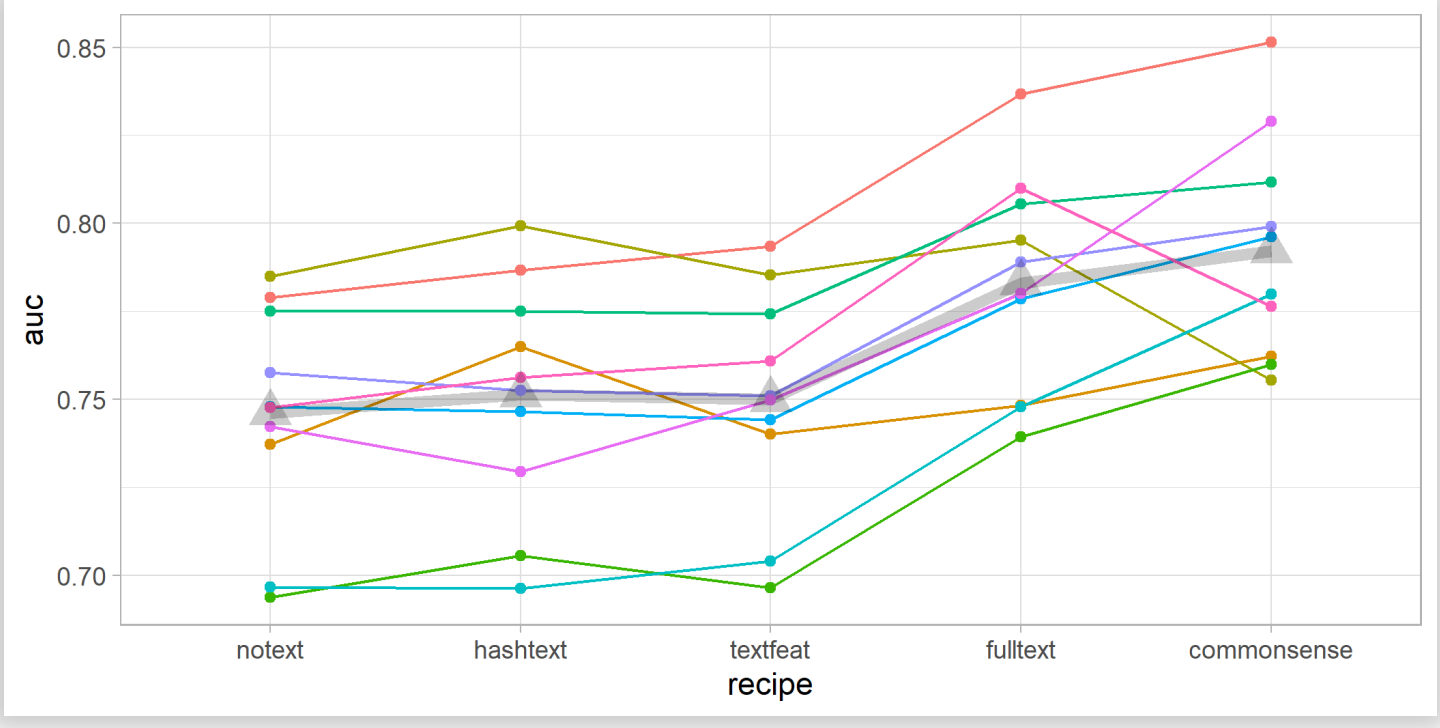
```
ok_tr_eng4 <- full_features(ok_tr_eng)

colnames(ok_tr_eng4)
```

```
##  [1] "age"                "body_type"          "diet"
##  [4] "drinks"             "drugs"              "education"
##  [7] "ethnicity"          "height"             "income"
## [10] "job"                "location"           "orientation"
## [13] "pets"               "religion"           "sex"
## [16] "smokes"             "status"             "len_essay0"
## [19] "len_essay1"         "len_essay2"         "len_essay3"
## [22] "len_essay4"         "len_essay5"         "len_essay6"
## [25] "len_essay7"         "len_essay8"         "len_essay9"
## [28] "essays"             "offspring_no_kids"  "offspring_one_kid"
```

```
rec_eng_commonsense <- recipe(pets ~ ., data = ok_tr_eng4) %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  update_role(essays, new_role = "discarded") %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("len_"), starts_with("t_"),
           -t_essays_sent_bing, offset = 1) %>%
  step_impute_mean(income) %>%
  step_zv(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors(), -has_role("discarded"), oth
  step_novel(all_nominal_predictors(), -has_role("discarded")) %>%
  step_impute_mode(all_nominal_predictors(), -has_role("discarded"
  step_dummy(all_nominal_predictors(), -has_role("discarded"), one
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```
set.seed(42)
cv_splits_eng2 <- vfold_cv(ok_tr_eng4, v = 10, strata = pets)

rec_list <- list("commonsense" = rec_eng_commonsense)

wset <- workflow_set(rec_list, mod_list) %>%
  workflow_map("fit_resamples",
               resamples = cv_splits_eng2,
               metrics = metric_set(roc_auc))

cv_res_commonsense <- wset %>%
  unnest(result) %>%
  mutate(recipe = wflow_id, auc = map_dbl(.metrics, ~.x$.estimate)
  select(recipe, id, auc) %>%
  separate_wider_delim(recipe, "_", names = c("recipe", "mod"))

cv_res_cs_meds <- cv_res_commonsense %>%
  group_by(recipe) %>%
  summarise(auc = mean(auc), id = "All")
```

```
cv_res_eng %>%
  bind_rows(cv_res_commonsense) %>%
  mutate(recipe = fct_relevel(recipe,
                              c("notext", "hashtext", "textfeat",
  ggplot(aes(recipe, auc, group = id, color = id)) +
  geom_line() + geom_line(data = cv_res_eng_meds %>% bind_rows(cv_
  geom_point() + geom_point(data = cv_res_eng_meds %>% bind_rows(c
  guides(color = "none") + theme_light()
```



Numerical Variables

What could the problem be?

The problem almost always depends on the combination between models and variables:

- Varied scale (and kNN, SVM) --> `step_center()`, `step_scale()`,`step_normalize()`,`step_range()`
- Bounded range of outcome (e.g. percentage and, well, almost any model) --> `step_logit()`
- Skewed data and outliers (and linear regression, NN, kNN, SVM, but less for Trees)
- Non-linearity (and LR)
- Co-linearity (and LR, NN)

Skewed Data: Common Transformations

Box-Cox

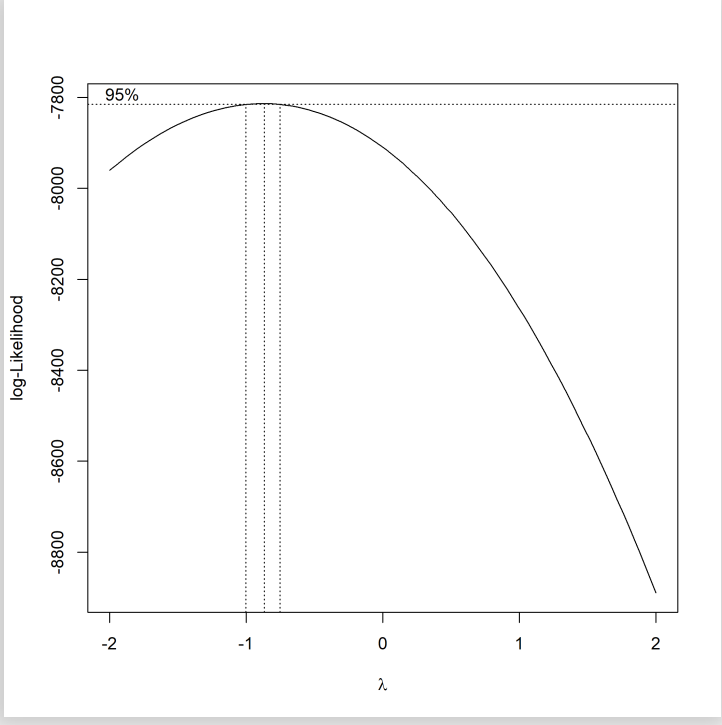
In its simplified form:

$$x_i^{(\lambda)} = \begin{cases} \frac{x_i^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \ln x_i & \text{if } \lambda = 0 \end{cases}$$

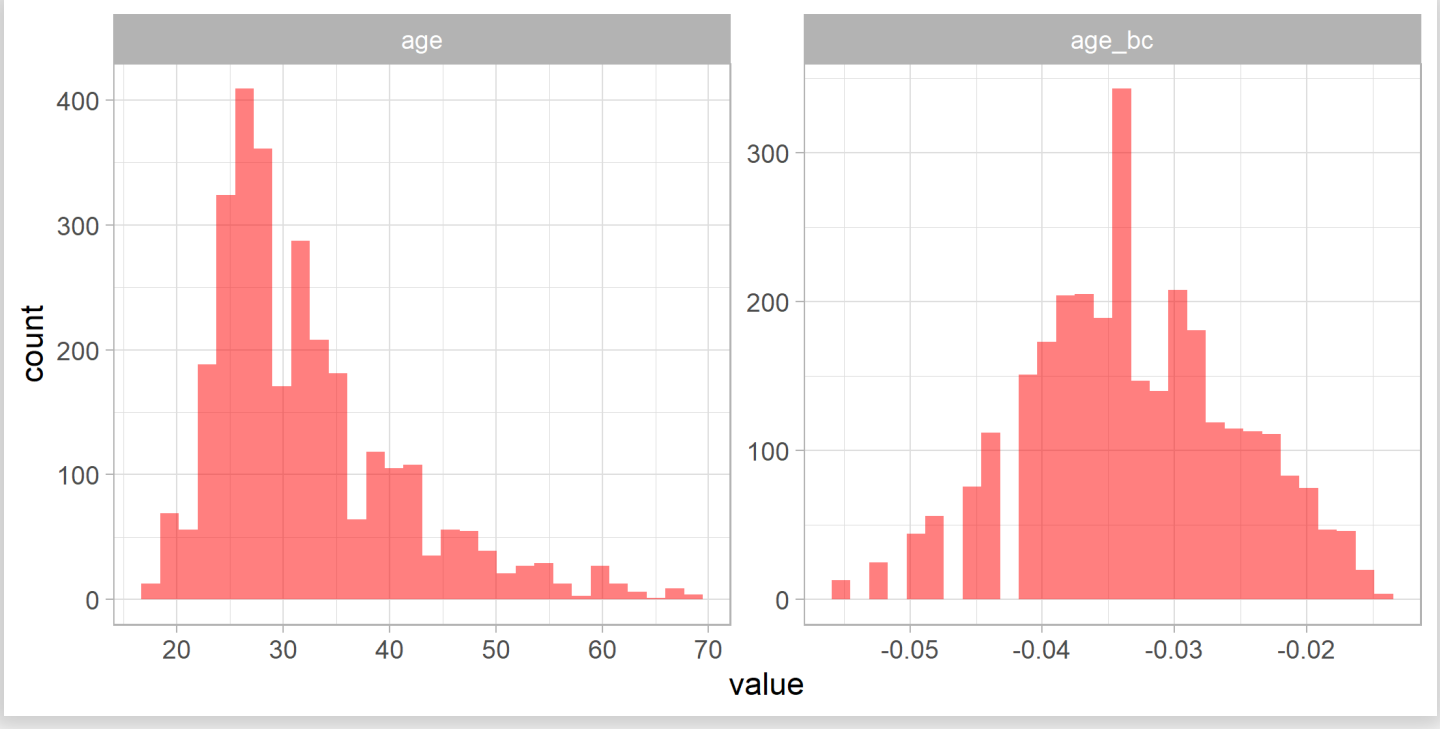
Originally the Box-Cox transformation was developed to transform a dependent y variable by maximizing the log-likelihood of seeing $y^{(\lambda)}$ in a `lm` against some X.

But λ could be estimated in an unsupervised way as well:

```
MASS::boxcox(ok_tr_eng$age ~ 1)
```



```
tibble(age = ok_tr_eng$age,
       age_bc = -1/age) %>%
  pivot_longer(starts_with("age"), names_to = "age") %>%
  ggplot(aes(value)) +
  geom_histogram(bins = 30, fill = "red", alpha = 0.5) +
  facet_wrap(~ age, scales = "free") + theme_light()
```



Yeo-Johnson

Revised Box-Cox to allow for zero and negative values:

$$x_i^{(\lambda)} = \begin{cases} \frac{(x_i+1)^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, y \geq 0 \\ \ln(x_i + 1) & \text{if } \lambda = 0, y \geq 0 \\ \frac{-[(-x_i+1)^{2-\lambda} - 1]}{2-\lambda} & \text{if } \lambda \neq 0, y < 0 \\ -\ln(-x_i + 1) & \text{if } \lambda = 0, y < 0 \end{cases}$$

Both transformations have `steps` in `recipes`:

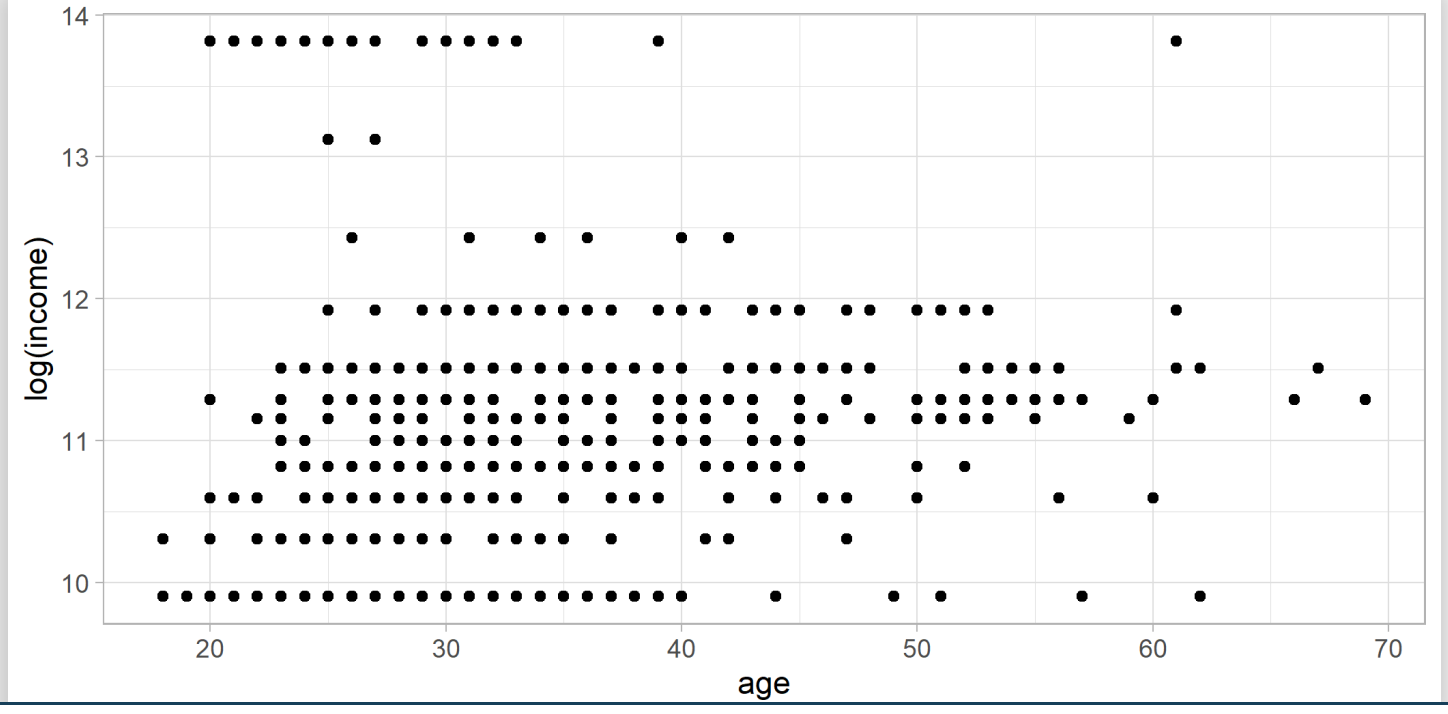
```
recipe(pets ~ ., data = ok_tr_eng) %>%
  step_BoxCox(income)
```

```
recipe(pets ~ ., data = ok_tr_eng) %>%
  step_YeoJohnson(income)
```

Non-Linearity: Basis Expansions and Splines

Let's borrow a continuous y variable like `log(income)` to simplify:

```
ggplot(ok_tr_eng, aes(age, log(income))) +
  geom_point() +
  theme_light()
```



You could do this manually in `lm()` like so

```
mod_lm <- lm(log(income) ~ I(pmax(0, 45 - age)) + I(pmax(0, age -
             data = ok_tr_eng)

summary(mod_lm)
```

```
##
## Call:
## lm(formula = log(income) ~ I(pmax(0, 45 - age)) + I(pmax(0, age -
##     45)), data = ok_tr_eng)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.4298 -0.6799 -0.0352  0.3498  3.1630
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)           11.361687   0.083615 135.880  < 2e-16 ***
## I(pmax(0, 45 - age)) -0.028369   0.005317  -5.335 1.36e-07 ***
## I(pmax(0, age - 45)) -0.007601   0.013507  -0.563    0.574
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8699 on 594 degrees of freedom
##   (2403 observations deleted due to missingness)
## Multiple R-squared:  0.0531,    Adjusted R-squared:  0.04991
## F-statistic: 16.66 on 2 and 594 DF,  p-value: 9.168e-08
```

Or use `splines::bs()` with `degree = 1` and `knots = 45`.

```
library(splines)
mod_lm_bs <- lm(log(income) ~ bs(age, degree = 1, knots = 45),
                data = ok_tr_eng)

summary(mod_lm_bs)
```

```
##
## Call:
## lm(formula = log(income) ~ bs(age, degree = 1, knots = 45), data = ok_tr
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.4298 -0.6799 -0.0352  0.3498  3.1630
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                      10.59574    0.07782 136.155  < 2e-16 **
## bs(age, degree = 1, knots = 45)1  0.76595    0.14356   5.335 1.36e-07 **
## bs(age, degree = 1, knots = 45)2  0.58353    0.28632   2.038    0.042 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8699 on 594 degrees of freedom
##   (2403 observations deleted due to missingness)
## Multiple R-squared:  0.0531,    Adjusted R-squared:  0.04991
## F-statistic: 16.66 on 2 and 594 DF,  p-value: 9.168e-08
```

How will the variables look like?

```
X <- model.matrix(mod_lm)
age_no_na <- ok_tr_eng$age[!is.na(ok_tr_eng$income)]
head(cbind(age_no_na, X), 4)
```

```
##    age_no_na (Intercept) I(pmax(0, 45 - age)) I(pmax(0, age - 45))
## 3         26           1                   19                    0
## 5         24           1                   21                    0
## 10        21           1                   24                    0
## 15        25           1                   20                    0
```

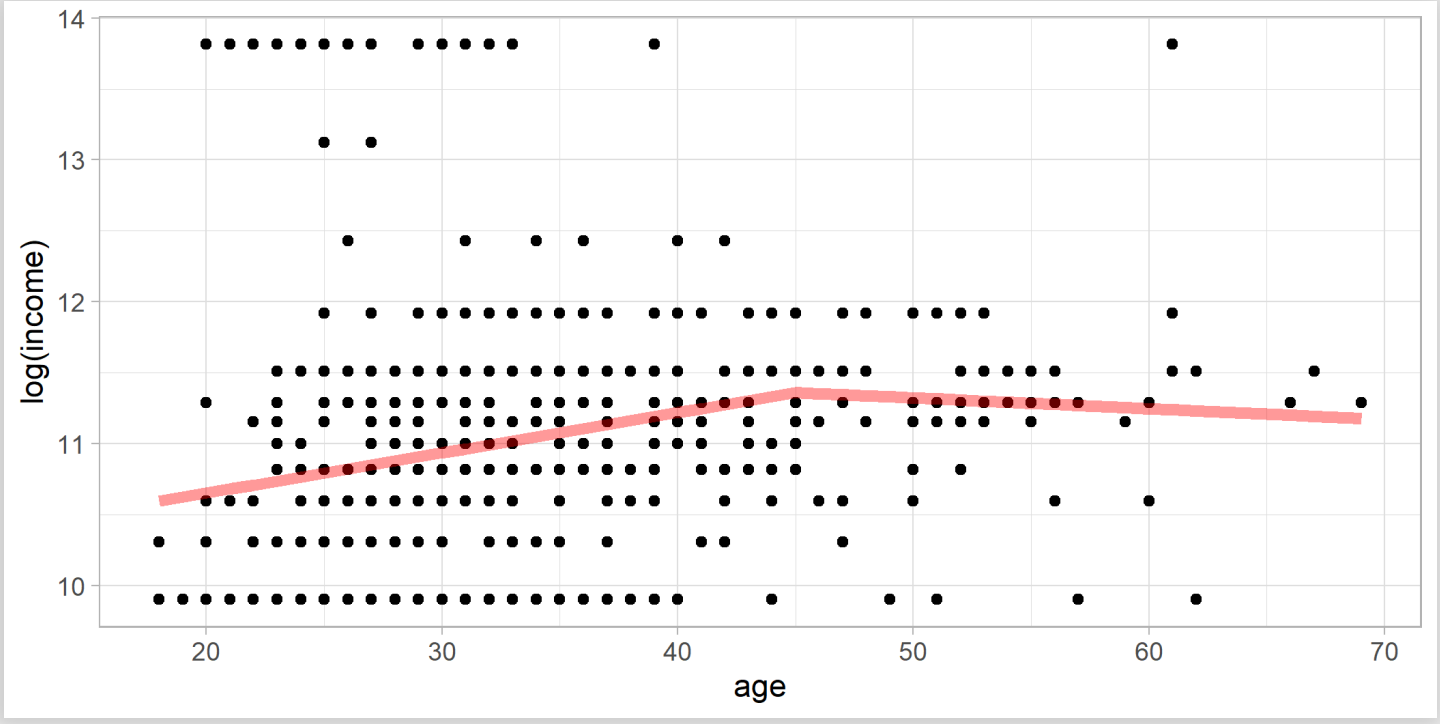
```
X <- model.matrix(mod_lm_bs)
colnames(X)[2:3] <- c("age1", "age2")
head(cbind(age_no_na, X), 4)
```

```
##    age_no_na (Intercept)      age1 age2
## 3         26           1 0.2962963    0
## 5         24           1 0.2222222    0
## 10        21           1 0.1111111    0
## 15        25           1 0.2592593    0
```

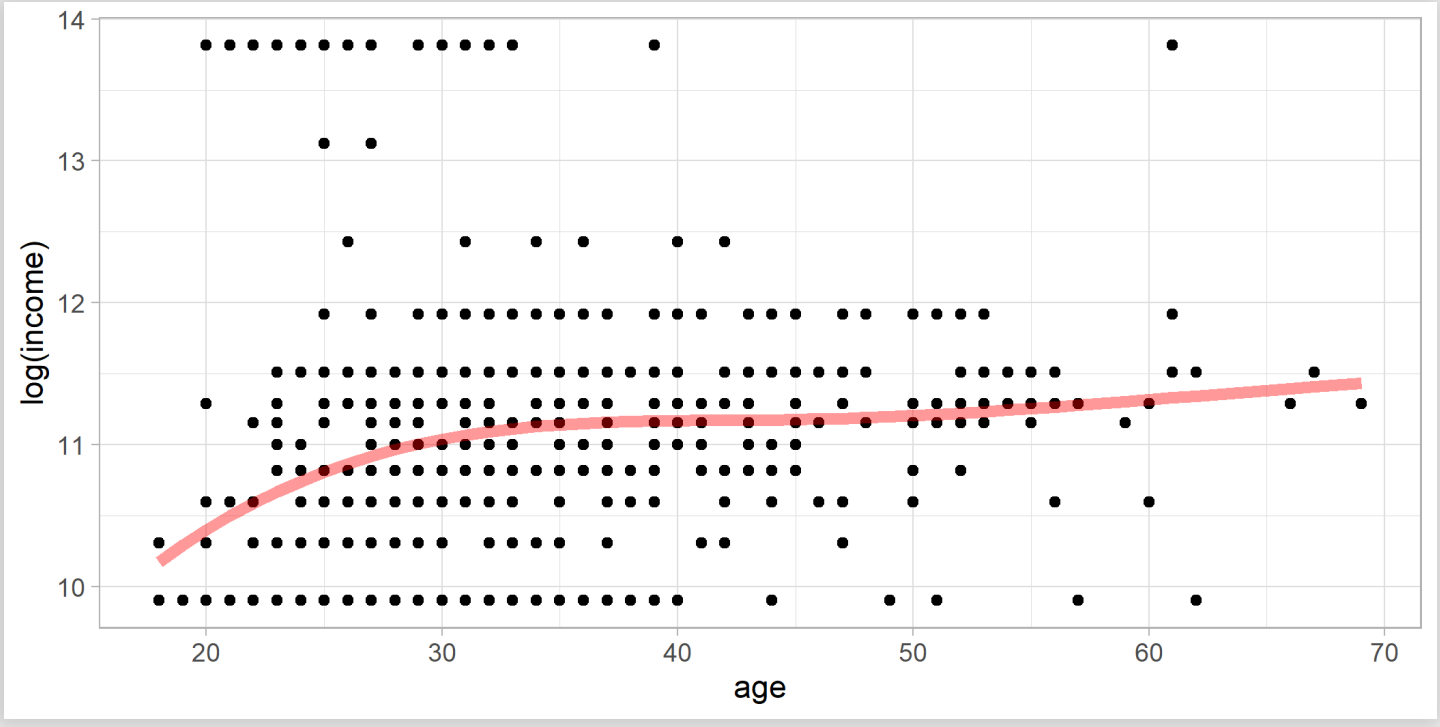
`splines::bs()` re-parametrizes the variables but it is the same model!

How will the fit look like?

```
ggplot(ok_tr_eng[!is.na(ok_tr_eng$income), ]) +
  geom_point(aes(age, log(income))) +
  geom_line(aes(age, mod_lm_bs$fit), color = "red", linewidth = 2,
  theme_light()
```



Or with a simple cubic spline with one knot in `bs()`



Manually with `step_relu()`

```
recipe(income ~ age, ok_tr_eng) %>%
  step_naomit(income) %>%
  step_relu(age, shift = 45, reverse = TRUE) %>%
  step_relu(age, shift = 45) %>%
  prep() %>%
  bake(new_data = NULL) %>%
  head()
```

```
## # A tibble: 6 × 4
##     age income left_relu_age right_relu_age
##   <dbl>  <dbl>         <dbl>           <dbl>
## 1    26  20000            19               0
## 2    24  60000            21               0
## 3    21  20000            24               0
## 4    25  20000            20               0
## 5    35  80000            10               0
## 6    69  80000             0              24
```

For splines and polynomial regression use `step_bs()`, `step_ns()` and `step_poly()`

```
recipe(income ~ age, ok_tr_eng) %>%
  step_naomit(income) %>%
  step_bs(age, degree = 3) %>%
  prep() %>%
  bake(new_data = NULL) %>%
  head()
```

```
## # A tibble: 6 × 4
##   income age_bs_1 age_bs_2 age_bs_3
##    <dbl>    <dbl>    <dbl>    <dbl>
## 1  20000    0.335  0.0622  0.00386
## 2  60000    0.275  0.0366  0.00163
## 3  20000    0.156  0.00977 0.000204
## 4  20000    0.306  0.0488  0.00259
## 5  80000    0.444  0.222   0.0370
## 6  80000    0      0       1
```

Co-linearity: Consider reducing dimensionality

> A conventional thought process is that the more predictors that are included, the better chance the model will have to find the signal. But this is not the case; including irrelevant predictors can have detrimental effects on the final model. The negative impacts include:
>
> - increasing the computational time required for training a model
> - decreasing the predictive performance
> - complicating the predictor importance calculation.
>
> By including any potential predictor, we end up with a sub-optimal model.

Kuhn & Johnson (2019)

Common *dimensionality reduction* methods (some of which we've talked about!):

- PCA, Kernel PCA, Sparse PCA --> `step_pca()`, `step_kpca()`
- NMF --> `step_nnmf()`
- PLS --> `step_pls()`

All of these will find some projection matrix B_{pxk} where k is the number of columns (e.g. PCs) to choose and would typically be much lower than p, the original dimension.

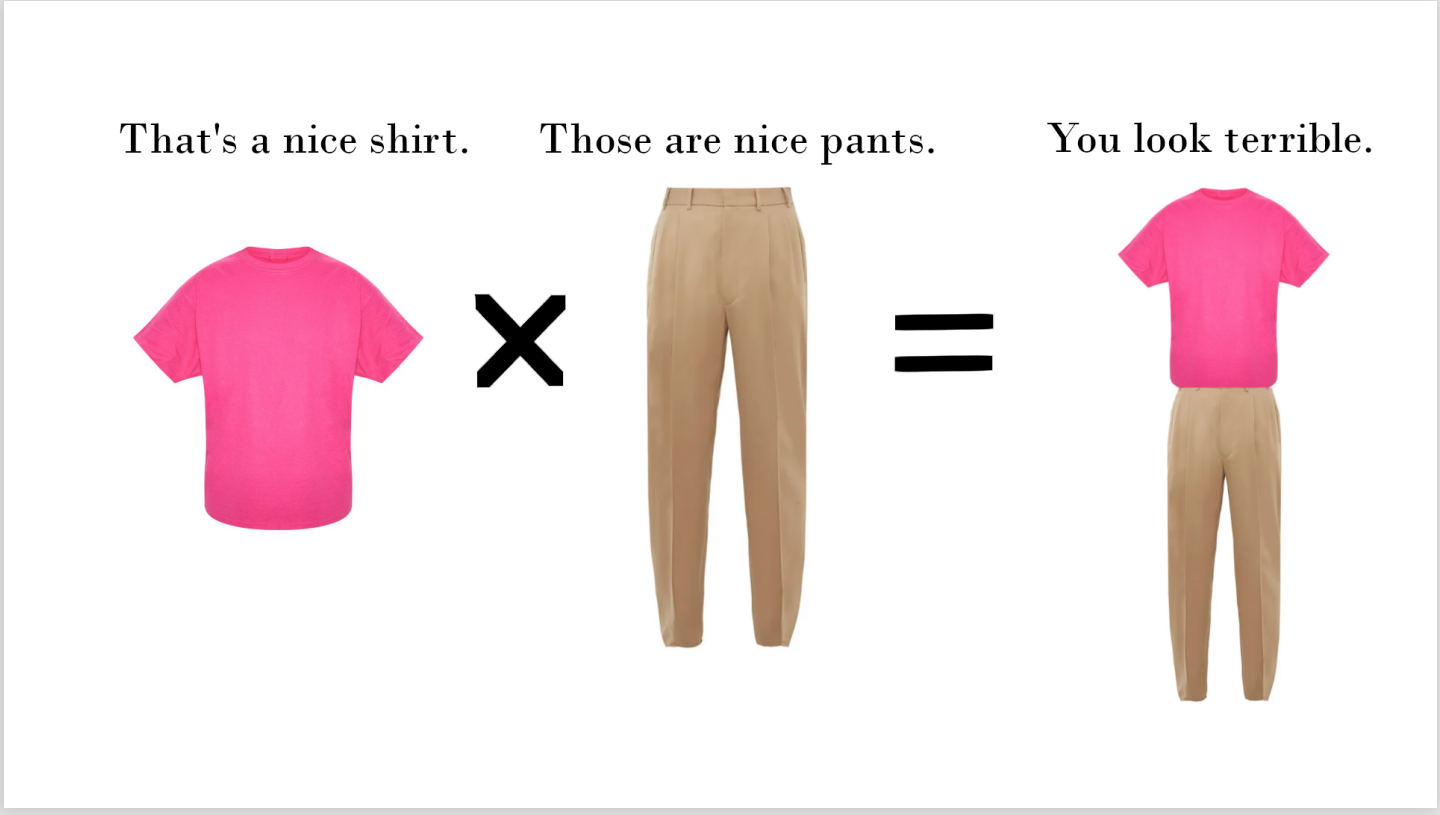
You would then project your X matrix like so: $X^* = XB$ (in most methods X^* columns would also be linearly independent, or "close" to it), and you would fit your data on this transformed X^*

⚠ No guarantee for improvement in performance! In fact...

💡 What is an alternative to reduce dimensionality?

Interaction Effects

What is Interaction?

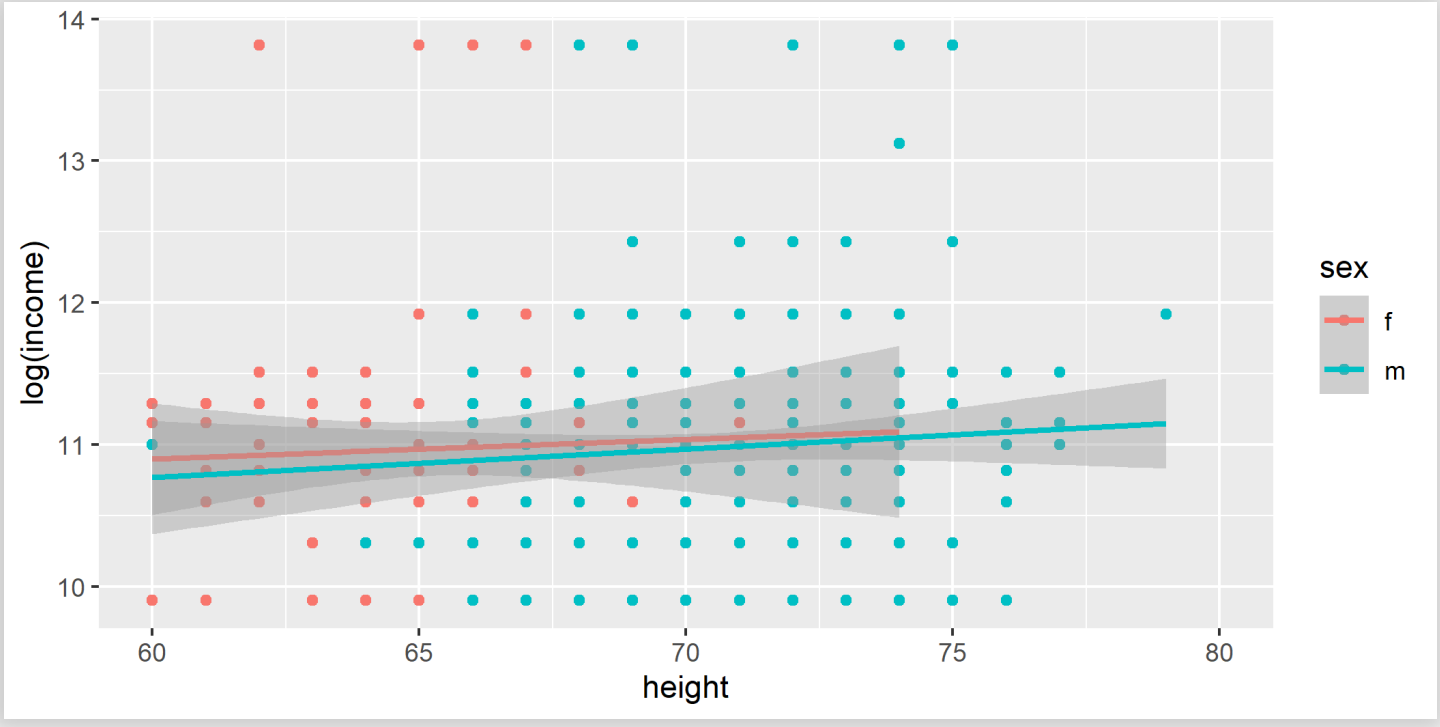


But seriously

```
lm_int <- lm(log(income) ~ sex + height + sex:height,
             data = ok_tr_interaction %>% filter(between(height, 6
summary(lm_int)
```

```
##
## Call:
## lm(formula = log(income) ~ sex + height + sex:height, data = ok_tr_inter
##     filter(between(height, 60, 80)))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.18582 -0.65885 -0.04721  0.50363  2.90622
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.075285   2.011322   5.009 8.38e-07 ***
## sexm        -0.506173   2.411062  -0.210    0.834
## height       0.013731   0.030795   0.446    0.656
## sexm:height  0.006272   0.036081   0.174    0.862
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.907 on 381 degrees of freedom
##   (1602 observations deleted due to missingness)
```

```r
ok_tr_interaction %>% filter(between(height, 60, 80)) %>%
  ggplot(aes(height, log(income), color = sex)) +
  geom_point() +
  geom_smooth(method = "lm")
```



💡 How would you visualize an interaction between two continuous features? Two categorical? Three?...

You get an interaction! You get an interaction!

With the last `rec_eng_commonsense` recipe we reached over 130 features.

So all two-way interactions would mean adding... over 8K features.
All three-way interactions would mean adding... over 350K features.

> 💡 What danger comes to mind? How would you defend yourself?
>
> Which models surface interactions naturally? What about our model?
>
> Should you include interaction terms before or after pre-processing?

Obviously, we need some control for this exploding no. of features:

- Expert knowledge (e.g. when it comes to income most variables probably interact with...?)
- Heredity principle
- Resampling, Penalize
- When p is huge: Two-Stage Modeling combined with your favorite Feature Selection approach
- Whatever you choose, visualize!

So these will be our four options for handling interactions:

1. Not
2. Adding only interactions between `sex` and everything
3. Two Stage: Screening all "main effects" features selected by a single `glmnet` stage then all two-way interactions of those features
4. Two Stage: Screening all two-way interactions by multiple comparisons of a `glm` main effects model vs. main effects + interaction, combined with FDR adjustment for the p-value
5. All two-way interactions 😎 (with a `step_nzv()` though)

For stages 3 and 4 let's use previously available data.

```
ok_tr_combined <- bind_rows(
  ok_tr_missing,
  ok_tr_imbalance,
  ok_tr_eng
  )
ok_tr_combined <- full_features(ok_tr_combined)
```

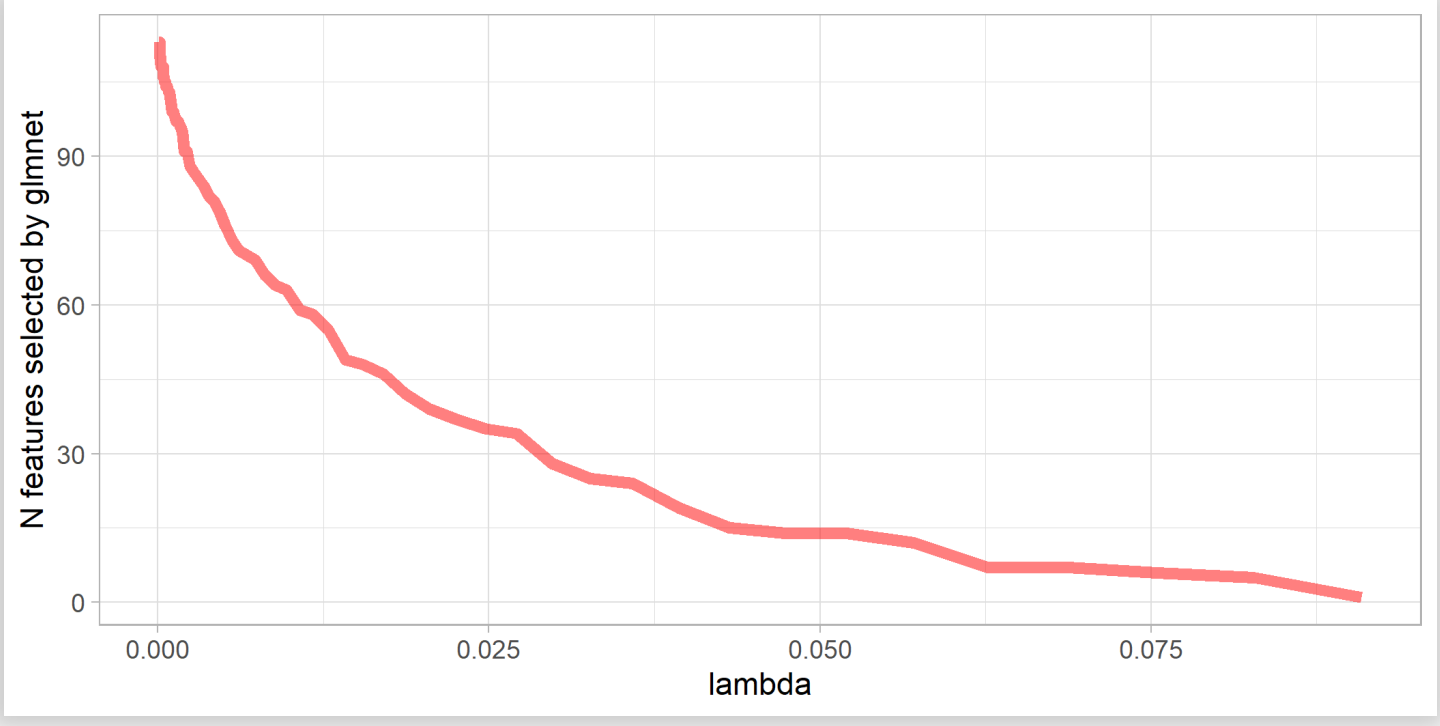
For stage 3 need to screen out "main effects" features which passed `glmnet`:

```
main_data <- rec_eng_commonsense %>%
  prep(ok_tr_combined) %>%
  bake(new_data = NULL)

mod_lr_res <- mod_lr %>%
  fit(pets ~ ., data = main_data) %>%
  pluck("fit") %>%
  tidy()

mod_lr_res
```

```
## # A tibble: 5,450 × 5
##    term         step  estimate lambda dev.ratio
##    <chr>       <dbl>     <dbl>  <dbl>     <dbl>
##  1 (Intercept)     1  5.14e-14 0.0909  8.81e-14
##  2 (Intercept)     2 -4.70e- 3 0.0828  9.49e- 3
##  3 (Intercept)     3 -2.94e- 3 0.0754  2.23e- 2
##  4 (Intercept)     4 -3.97e- 4 0.0687  3.48e- 2
##  5 (Intercept)     5  1.66e- 3 0.0626  4.57e- 2
##  6 (Intercept)     6  3.34e- 3 0.0571  5.87e- 2
##  7 (Intercept)     7  4.84e- 2 0.0520  7.24e- 2
##  8 (Intercept)     8  8.18e- 2 0.0474  8.59e- 2
##  9 (Intercept)     9  1.16e- 1 0.0432  9.84e- 2
## 10 (Intercept)    10  1.45e- 1 0.0393  1.10e- 1
## # … with 5,440 more rows
```

We can plot no. of features selected by `glmnet` by lambda

```
mod_lr_res %>%
  count(lambda) %>%
  ggplot(aes(lambda, n)) +
  geom_line(linewidth = 2, color = "red", alpha = 0.5) +
  labs(y = "N features selected by glmnet") +
  theme_light()
```



We see a plateau around 12-15 features, let's choose those.

```
mod_lr_res %>%
  filter(step == 6) %>%
  select(term, estimate) %>%
  arrange(-abs(estimate))
```

```
## # A tibble: 12 × 2
##    term                estimate
##    <chr>                  <dbl>
##  1 w_my_cat            -0.458
##  2 status_all_else     -0.343
##  3 religion_christian   0.304
##  4 w_my_dog             0.227
##  5 w_dog                0.114
##  6 job_marketing        0.0623
##  7 w_cat               -0.0551
##  8 drugs_sometimes     -0.0368
##  9 drinks_socially      0.0269
## 10 w_cats              -0.0147
## 11 (Intercept)          0.00334
## 12 age                 -0.000539
```

These make sense, let's choose them as our main effects from which we pick all two-way interactions.

```
main_effects <- mod_lr_res %>%
  filter(step == 6, term != "(Intercept)") %>%
  pull(term)
```

For stage 4 we need to iterate over all possible two-way interactions, each time adding one into a full main effects `glm` model, getting the p-value, and applying FDR adjustment.

Start by defining the base model:

```
mod_glm_base <- glm(pets ~ .,
                    data = main_data %>% select(-ends_with("_new")),
                    family = "binomial")
```

Then go over all interactions:

```
extract_interaction_p_val <- function(var1, var2) {
  pb$tick()$print()
  int_term <- str_c(var1, ":", var2)
  int_term2 <- str_c(var2, ":", var1)
  mod_glm <- quietly(stats::update)(
    mod_glm_base, as.formula(str_c(".~.+", int_term))
  )
  if(length(mod_glm$warnings) > 1) {
    return(NA_real_)
  }
  p_val <- tidy(mod_glm$result) %>%
    filter(term %in% c(int_term, int_term2)) %>%
    pull(p.value)
  if (length(p_val) == 0) {
    return(NA_real_)
  }
  p_val
}

all_vars <- colnames(main_data %>% select(-ends_with("_new")))
all_ints <- t(combn(all_vars[-which(all_vars == "pets")], 2))
colnames(all_ints) <- c("var1", "var2")
all_ints <- as_tibble(all_ints)
```

```
pb <- progress_estimated(nrow(all_ints))
all_ints <- all_ints %>%
  mutate(p_val = map2_dbl(var1, var2, extract_interaction_p_val),
         fdr = p.adjust(p_val, method = "fdr"))
```

Let's filter only the *most* convincing interactions.

```
pairwise_ints <- all_ints %>%
  arrange(fdr) %>%
  filter(fdr < 10e-7)

pairwise_ints
```

```
## # A tibble: 18 × 4
##    var1                  var2                      p_val      fdr
##    <chr>                 <chr>                     <dbl>    <dbl>
##  1 offspring_no_kids     offspring_wants_kids   3.60e-19 2.35e-15
##  2 ethnicity_white       job_hardware           3.08e-12 1.01e- 8
##  3 t_essays_n_words      education_kind_working 1.07e-10 1.40e- 7
##  4 t_essays_n_charS      education_kind_working 9.15e-11 1.40e- 7
##  5 t_essays_n_lowers     education_kind_working 8.64e-11 1.40e- 7
##  6 t_essays_n_uq_words   education_kind_working 1.40e-10 1.52e- 7
##  7 t_essays_n_uq_charS   education_kind_working 1.69e-10 1.58e- 7
##  8 speaks_chinese        t_essays_sent_bing     3.20e-10 2.61e- 7
##  9 t_essays_n_extraspaces education_kind_working 3.90e-10 2.83e- 7
## 10 age                   t_essays_n_uq_charS    5.63e-10 3.41e- 7
## 11 body_type_perfect     ethnicity_white        5.73e-10 3.41e- 7
```

First make sure you understand the difference between `other`, `all_else` etc.

```
ok_tr_combined %>% count(ethnicity, sort = TRUE)
```

```
## # A tibble: 6 × 2
##   ethnicity     n
##   <fct>     <int>
## 1 white      3802
## 2 other      1675
## 3 asian       752
## 4 hispanic    401
## 5 black       244
## 6 indian      126
```

```
main_data %>% select(starts_with("ethnicity")) %>% head()
```

```
## # A tibble: 6 × 5
##   ethnicity_hispanic ethnicity_other ethnicity_white ethnicity_all_else
##                <dbl>           <dbl>           <dbl>              <dbl>
## 1                  0               0               1                  0
## 2                  0               0               1                  0
## 3                  0               0               1                  0
## 4                  0               1               0                  0
## 5                  0               0               1                  0
## 6                  0               0               1                  0
```

What are those people with all `ethnicity` columns with zero?

```
main_data %>% select(starts_with("ethnicity")) %>%
  filter(if_all(everything(), ~.x == 0))
```

```
## # A tibble: 1,039 × 5
##    ethnicity_hispanic ethnicity_other ethnicity_white ethnicity_all_else
##                 <dbl>           <dbl>           <dbl>               <dbl>
##  1                  0               0               0                   0
##  2                  0               0               0                   0
##  3                  0               0               0                   0
##  4                  0               0               0                   0
##  5                  0               0               0                   0
##  6                  0               0               0                   0
##  7                  0               0               0                   0
##  8                  0               0               0                   0
##  9                  0               0               0                   0
## 10                  0               0               0                   0
## # … with 1,029 more rows, and 1 more variable: ethnicity_new <dbl>
```

Same with `location`.

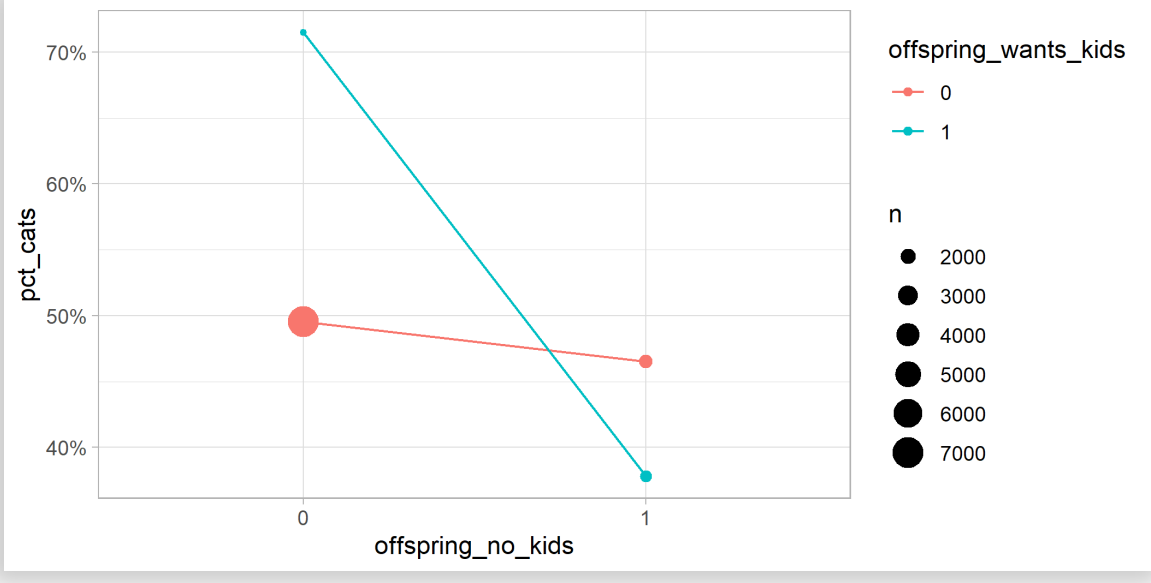
```
ok_tr_combined %>% count(location, sort = TRUE)
```

```
## # A tibble: 4 × 2
##   location     n
##   <fct>    <int>
## 1 sf        3739
## 2 other     2045
## 3 oakland    776
## 4 berkeley   440
```

```
main_data %>% select(starts_with("location")) %>% head()
```

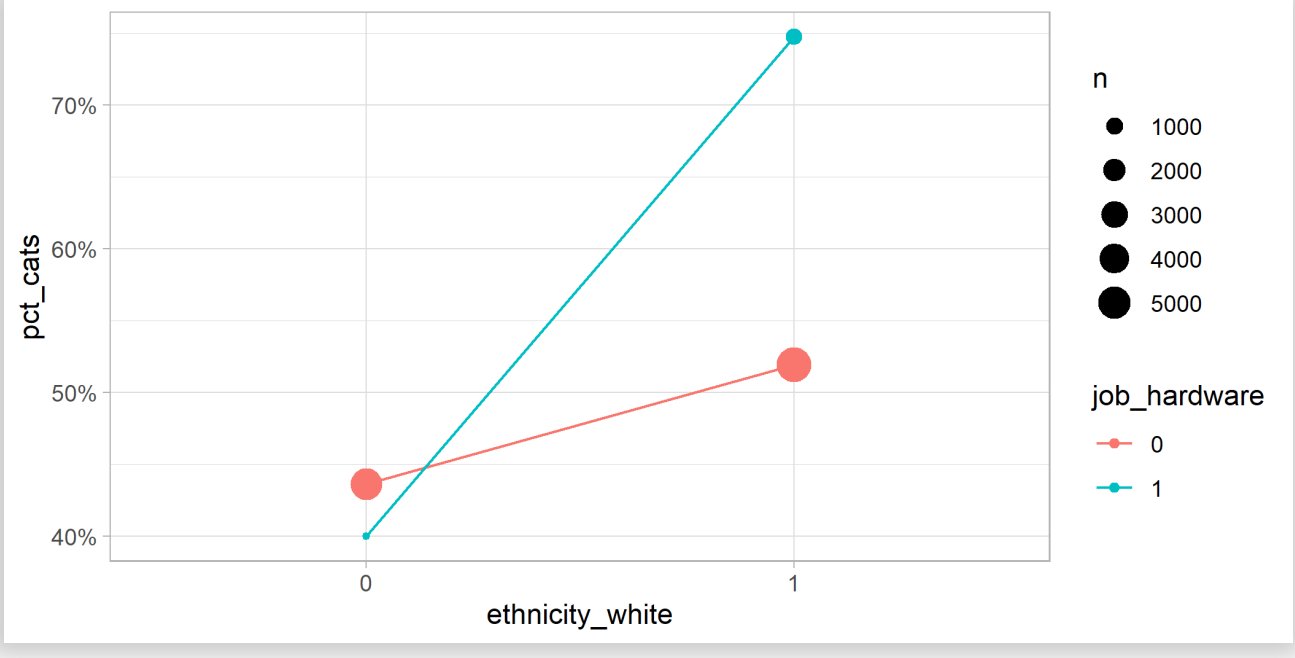
```
## # A tibble: 6 × 4
##   location_oakland location_other location_sf location_new
##              <dbl>          <dbl>       <dbl>        <dbl>
## 1                0              1           0            0
## 2                0              1           0            0
## 3                0              0           1            0
## 4                0              0           1            0
## 5                1              0           0            0
## 6                1              0           0            0
```

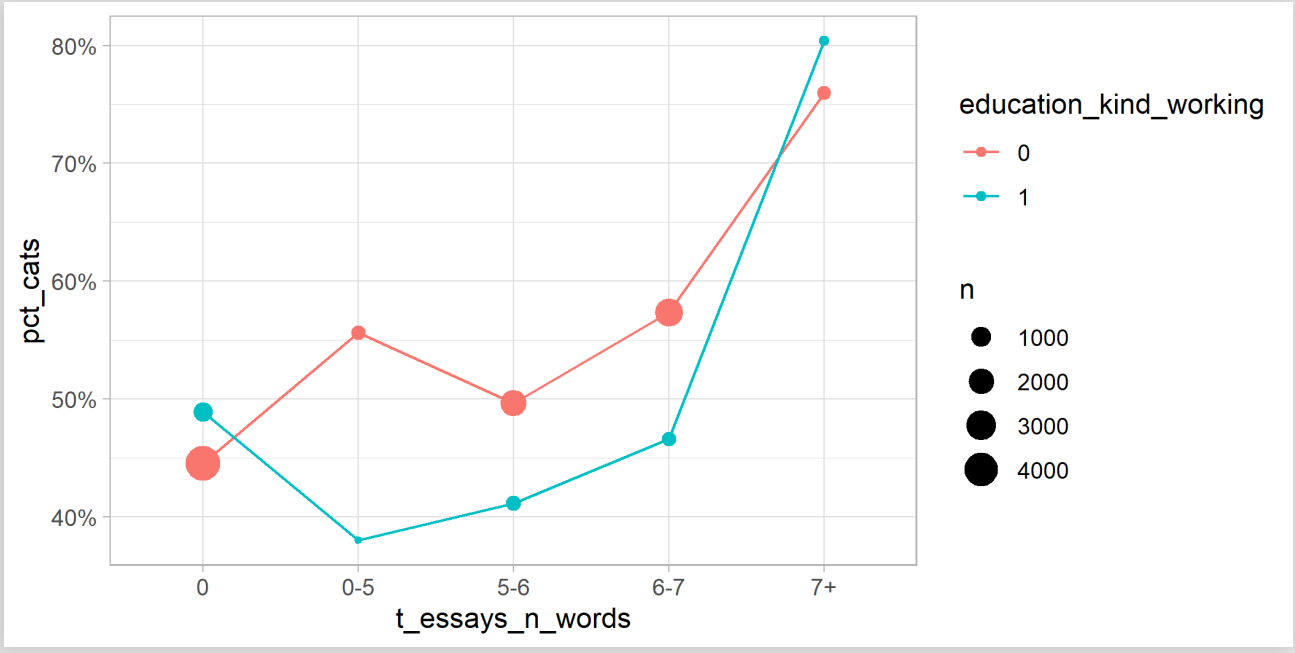
```r
main_data %>% group_by(offspring_no_kids, offspring_wants_kids) %>
    summarise(pct_cats = mean(pets == "cats"), n = n()) %>% ungrou
    mutate(across(c("offspring_no_kids", "offspring_wants_kids"),
    ggplot(aes(offspring_no_kids, pct_cats, color = offspring_want
    geom_line(aes(group = offspring_wants_kids)) + geom_point(aes
    scale_y_continuous(labels = percent_format()) +
    theme_light()
```



💡 Try stating what you see in words.

```
main_data %>% group_by(ethnicity_white, job_hardware) %>%
    summarise(pct_cats = mean(pets == "cats"), n = n()) %>% ungrou
    mutate(across(c("ethnicity_white", "job_hardware"), as.factor)
    ggplot(aes(ethnicity_white, pct_cats, color = job_hardware)) +
    geom_line(aes(group = job_hardware)) + geom_point(aes(size = n
    scale_y_continuous(labels = percent_format()) +
    theme_light()
```




80%
70%
60%
50%
40%
pct_cats
0
0-5
5-6
6-7
7+
t_essays_n_words
education_kind_working
0
1
n
1000
2000
3000
4000

```
ok_tr_interaction2 <- full_features(ok_tr_interaction)

rec_int_noint <- rec_eng_commonsense

rec_int_sex <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  update_role(essays, new_role = "discarded") %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("len_"), starts_with("t_"),
           -t_essays_sent_bing, offset = 1) %>%
  step_impute_mean(income) %>%
  step_other(all_nominal_predictors(), -has_role("discarded"), oth
  step_novel(all_nominal_predictors(), -has_role("discarded")) %>%
  step_impute_mode(all_nominal_predictors(), -has_role("discarded"
  step_dummy(all_nominal_predictors(), -has_role("discarded"), one
  step_interact(~starts_with("sex"):all_numeric_predictors()) %>%
  step_nzv(all_numeric_predictors(), freq_cut = 99/1) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```
# pairwise interactions string code from Max Kuhn's workshop:
# https://github.com/topepo/FES/blob/master/07_Detecting_Interacti

topmain_ints <- t(combn(main_effects, 2))
colnames(topmain_ints) <- c("var1", "var2")
topmain_ints <- topmain_ints %>%
  as_tibble() %>%
  mutate(
    term = str_c(var1, ":", var2)
  ) %>%
  pull(term) %>%
  paste(collapse = "+")

topmain_ints <- str_c("~", topmain_ints)
topmain_ints <- as.formula(topmain_ints)

topmain_ints
```

```
## ~age:w_cat + age:w_dog + age:w_cats + age:w_my_cat + age:w_my_dog +
##     age:drinks_socially + age:drugs_sometimes + age:job_marketing +
##     age:religion_christian + age:status_all_else + w_cat:w_dog +
##     w_cat:w_cats + w_cat:w_my_cat + w_cat:w_my_dog + w_cat:drinks_social
##     w_cat:drugs_sometimes + w_cat:job_marketing + w_cat:religion_christi
##     w_cat:status_all_else + w_dog:w_cats + w_dog:w_my_cat + w_dog:w_my_d
##     w_dog:drinks_socially + w_dog:drugs_sometimes + w_dog:job_marketing
##     w_dog:religion_christian + w_dog:status_all_else + w_cats:w_my_cat +
##     w_cats:w_my_dog + w_cats:drinks_socially + w_cats:drugs_sometimes +
##     w_cats:job_marketing + w_cats:religion_christian + w_cats:status_all
##     w_my_cat:w_my_dog + w_my_cat:drinks_socially + w_my_cat:drugs_sometim
```

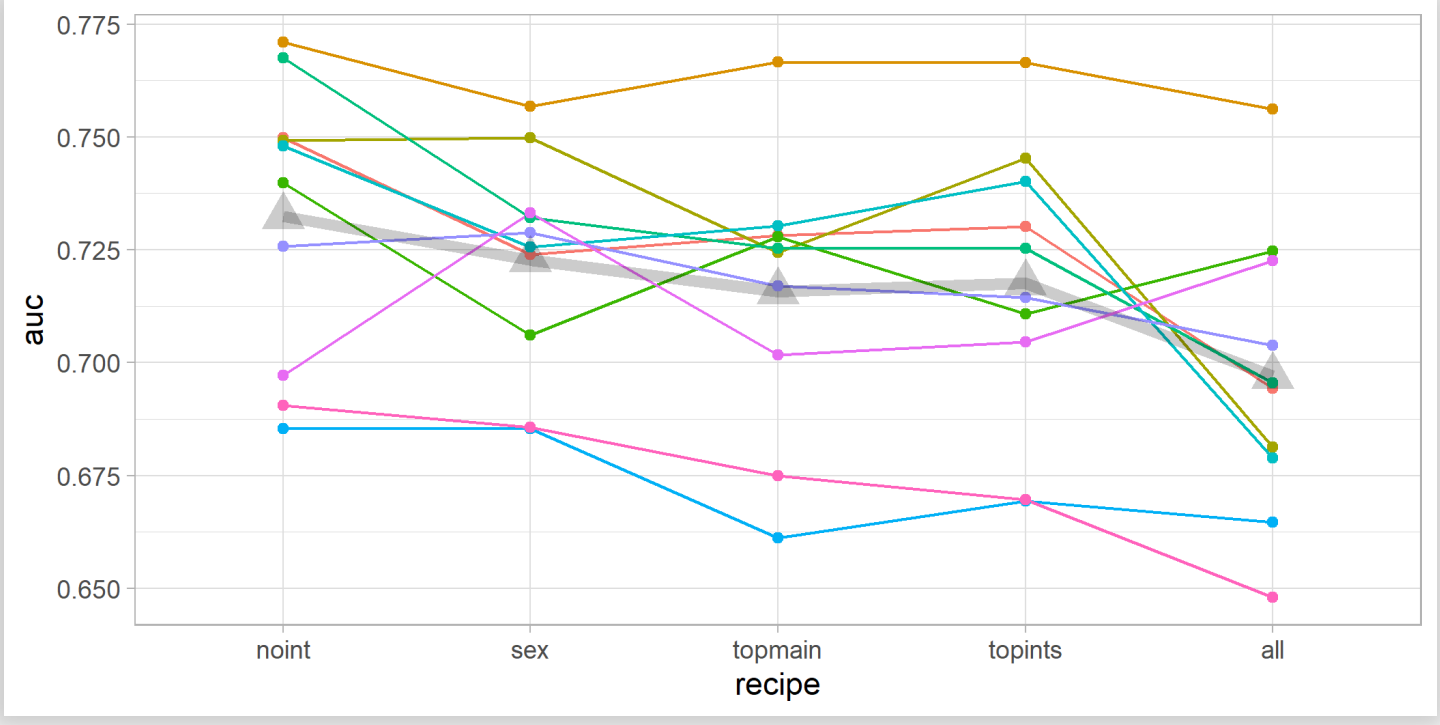
```
rec_int_topmain <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  update_role(essays, new_role = "discarded") %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("len_"), starts_with("t_"),
           -t_essays_sent_bing, offset = 1) %>%
  step_impute_mean(income) %>%
  step_other(all_nominal_predictors(), -has_role("discarded"), oth
  step_novel(all_nominal_predictors(), -has_role("discarded")) %>%
  step_impute_mode(all_nominal_predictors(), -has_role("discarded"
  step_dummy(all_nominal_predictors(), -has_role("discarded"), one
  step_interact(topmain_ints) %>%
  step_nzv(all_numeric_predictors(), freq_cut = 99/1) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```
rec_int_topints <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  update_role(essays, new_role = "discarded") %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("len_"), starts_with("t_"),
           -t_essays_sent_bing, offset = 1) %>%
  step_impute_mean(income) %>%
  step_other(all_nominal_predictors(), -has_role("discarded"), oth
  step_novel(all_nominal_predictors(), -has_role("discarded")) %>%
  step_impute_mode(all_nominal_predictors(), -has_role("discarded"
  step_dummy(all_nominal_predictors(), -has_role("discarded"), one
  step_interact(topint_ints) %>%
  step_nzv(all_numeric_predictors(), freq_cut = 99/1) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```
rec_int_all <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  update_role(essays, new_role = "discarded") %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("len_"), starts_with("t_"),
           -t_essays_sent_bing, offset = 1) %>%
  step_impute_mean(income) %>%
  step_other(all_nominal_predictors(), -has_role("discarded"), oth
  step_novel(all_nominal_predictors(), -has_role("discarded")) %>%
  step_impute_mode(all_nominal_predictors(), -has_role("discarded"
  step_dummy(all_nominal_predictors(), -has_role("discarded"), one
  step_interact(~all_numeric_predictors():all_numeric_predictors()
  step_nzv(all_numeric_predictors(), freq_cut = 95/5) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```
set.seed(42)
cv_splits_int <- vfold_cv(ok_tr_interaction2, v = 10, strata = pet

rec_list <- list("noint" = rec_int_noint,
                 "sex" = rec_int_sex,
                 "topmain" = rec_int_topmain,
                 "topints" = rec_int_topints,
                 "all" = rec_int_all)

wset <- workflow_set(rec_list, mod_list) %>%
  workflow_map("fit_resamples",
               resamples = cv_splits_int,
               metrics = metric_set(roc_auc))

cv_res_int <- wset %>%
  unnest(result) %>%
  mutate(recipe = wflow_id, auc = map_dbl(.metrics, ~.x$.estimate)
  select(recipe, id, auc) %>%
  separate_wider_delim(recipe, "_", names = c("recipe", "mod"))

cv_res_int_meds <- cv_res_int %>%
  group_by(recipe) %>%
  summarise(auc = mean(auc), id = "All")
```

```
cv_res_int %>%
  mutate(recipe = fct_relevel(recipe, c("noint", "sex", "topmain",
  ggplot(aes(recipe, auc, group = id, color = id)) +
  geom_line() + geom_line(data = cv_res_int_meds, color = "black",
  geom_point() + geom_point(data = cv_res_int_meds, color = "blac
  guides(color = "none") + theme_light()
```



Tuning Params

Everything is Tunable

> 💡 I can think of at least 5 semi-random quantitative decisions so far, can you?

But let us simply vary Elastic Net α and λ:

$$\min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1]$$

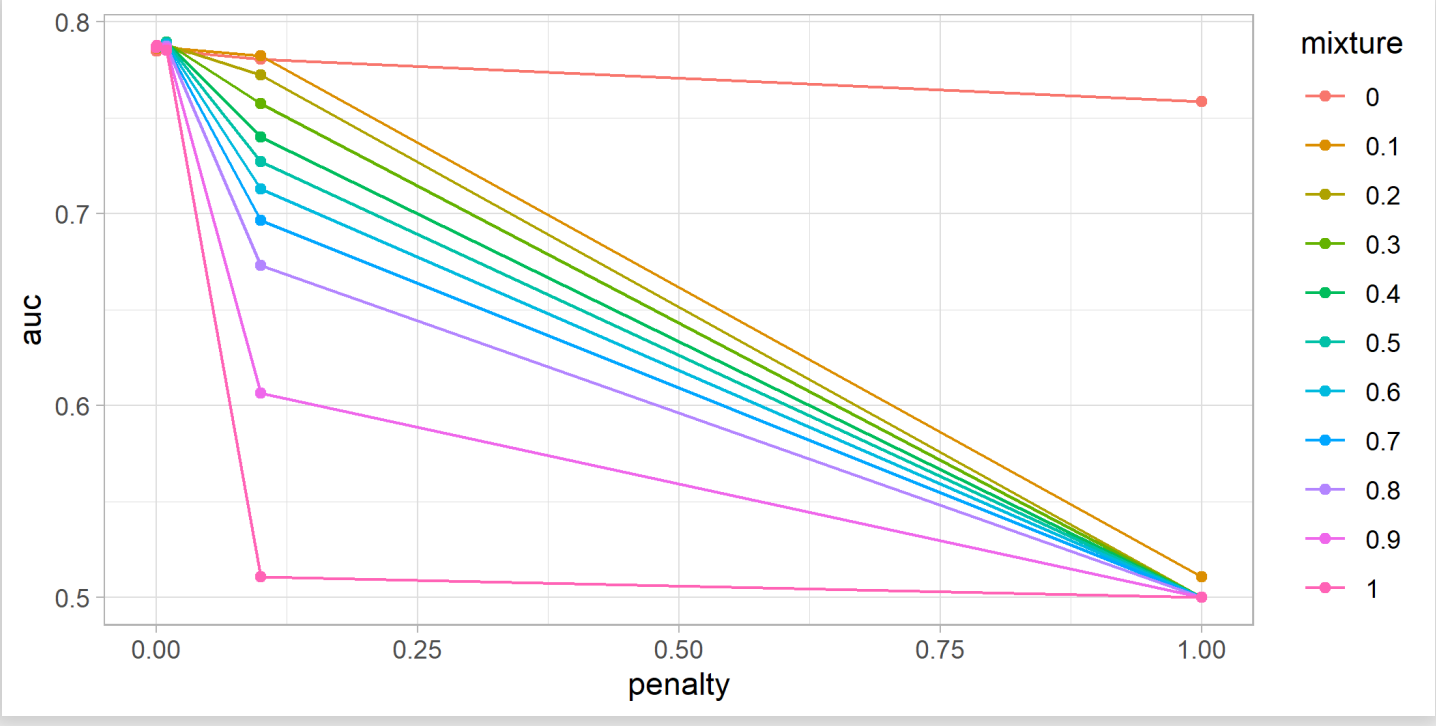
```
mod_en_tune <- logistic_reg(penalty = tune(),
                            mixture = tune()) %>%
  set_engine("glmnet")

en_grid <- grid_regular(penalty(range(-6, 0)),
                        mixture(range(0, 1)), levels = c(7, 11))
```

```
ok_tr_tuning2 <- full_features(ok_tr_tuning)

set.seed(42)
cv_splits_tune <- vfold_cv(ok_tr_tuning2, v = 10, strata = pets)
```

```
tune_res <- tune_grid(object = mod_en_tune,
                      preprocessor = rec_eng_commonsense,
                      resamples = cv_splits_tune,
                      grid = en_grid,
                      metrics = metric_set(roc_auc),
                      control = control_grid(verbose = TRUE))

collect_metrics_tune_res <- collect_metrics(tune_res)

collect_metrics_tune_res
```

```
## # A tibble: 77 × 8
##     penalty mixture .metric .estimator  mean     n std_err .config
##       <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.000001     0   roc_auc binary     0.785    10  0.0120 Preprocessor1
##  2 0.00001      0   roc_auc binary     0.785    10  0.0120 Preprocessor1
##  3 0.0001       0   roc_auc binary     0.785    10  0.0120 Preprocessor1
##  4 0.001        0   roc_auc binary     0.785    10  0.0120 Preprocessor1
##  5 0.01         0   roc_auc binary     0.785    10  0.0120 Preprocessor1
##  6 0.1          0   roc_auc binary     0.780    10  0.0125 Preprocessor1
##  7 1            0   roc_auc binary     0.758    10  0.0135 Preprocessor1
##  8 0.000001     0.1 roc_auc binary     0.787    10  0.0113 Preprocessor1
##  9 0.00001      0.1 roc_auc binary     0.787    10  0.0113 Preprocessor1
```

```
collect_metrics_tune_res %>%
  mutate(auc = mean, mixture = factor(mixture)) %>%
  ggplot(aes(penalty, auc, color = mixture)) +
  geom_line() + geom_point() +
  theme_light()
```



```
show_best(tune_res, n = 10, metric = "roc_auc")
```

```
## # A tibble: 10 × 8
##    penalty mixture .metric .estimator  mean     n std_err .config
##      <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1   0.01      0.4 roc_auc binary     0.789    10  0.0108 Preprocessor1_
##  2   0.01      0.5 roc_auc binary     0.789    10  0.0110 Preprocessor1_
##  3   0.01      0.3 roc_auc binary     0.789    10  0.0110 Preprocessor1_
##  4   0.01      0.6 roc_auc binary     0.788    10  0.0111 Preprocessor1_
##  5   0.01      0.2 roc_auc binary     0.788    10  0.0112 Preprocessor1_
##  6   0.001     1   roc_auc binary     0.788    10  0.0117 Preprocessor1_
##  7   0.01      0.7 roc_auc binary     0.787    10  0.0111 Preprocessor1_
##  8   0.001     0.8 roc_auc binary     0.787    10  0.0117 Preprocessor1_
##  9   0.01      0.8 roc_auc binary     0.787    10  0.0111 Preprocessor1_
## 10   0.001     0.9 roc_auc binary     0.787    10  0.0117 Preprocessor1_
```

💡 So what would you do?

Cutoff choice

The model is there.

Remember it is best to predict a score, let the model's "clients" choose what to do with that score.

Also remember the "I don't know" strategy!

But if we must, let us provide:

- a score to maximize precision of `cats` (while maintaing acceptable levels of recall)
- a score to maximize recall of `cats` (while maintaing acceptable levels of precision)
- a middle score

```
ok_tr_combined <- full_features(
  bind_rows(
    ok_tr_missing,
    ok_tr_imbalance,
    ok_tr_eng,
    ok_tr_interaction,
    ok_tr_tuning))

ok_tr_cutoff2 <- full_features(ok_tr_cutoff)

rec_cutoff <- rec_eng_commonsense %>%
  prep(ok_tr_combined)

ok_tr_for_cutoff <- rec_cutoff %>%
  bake(new_data = NULL)

ok_te_for_cutoff <- rec_cutoff %>%
  bake(ok_tr_cutoff2)

mod_lr_cutoff <- logistic_reg(penalty = 0.01, mixture = 0.5) %>%
  set_engine("glmnet")

mod_lr_cutoff <- mod_lr_cutoff %>%
  fit(pets ~ ., data = ok_tr_for_cutoff)

pred_cutoff <-  mod_lr_cutoff %>%
  predict(new_data = ok_te_for_cutoff, penalty = 0.01, type = "pro
  mutate(truth = ok_te_for_cutoff$pets)
```

```
library(pROC)

cutoffs <- seq(0.1, 0.9, 0.1)

roc_obj <- roc(pred_cutoff$truth, pred_cutoff$.pred_cats,
               levels = c("dogs", "cats"))
coords(roc_obj, x = cutoffs,
               ret = c("accuracy", "recall", "precision",
                       "specificity", "npv"),
               transpose = TRUE)
```

```
##                   [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## accuracy    0.21262003 0.2949246 0.4663923 0.6159122 0.7503429 0.8052126
## recall      1.00000000 0.9916667 0.9250000 0.8416667 0.6833333 0.4666667
## precision   0.17291066 0.1882911 0.2260692 0.2790055 0.3628319 0.4179104
## specificity 0.05747126 0.1576355 0.3760263 0.5714286 0.7635468 0.8719212
## npv         1.00000000 0.9896907 0.9621849 0.9482289 0.9244533 0.8924370
##                   [,7]      [,8]      [,9]
## accuracy    0.8381344 0.8600823 0.8545953
## recall      0.3166667 0.2333333 0.1333333
## precision   0.5135135 0.7368421 0.8888889
## specificity 0.9408867 0.9835796 0.9967159
## npv         0.8748092 0.8668596 0.8537271
```

Predict on Test set

Long time no see!

```
set.seed(42)
random_cats_to_get_rid_of <- okcupid_test %>%
  mutate(id = 1:n()) %>%
  filter(pets == "cats") %>%
  sample_n(40) %>%
  pull(id)

okcupid_test2 <- okcupid_test %>%
  mutate(id = 1:n()) %>%
  filter(!id %in% random_cats_to_get_rid_of) %>%
  select(-id)
okcupid_test2 <- full_features(okcupid_test2)

okcupid_test2 %>% count(pets) %>% mutate(pct = n / sum(n))
```

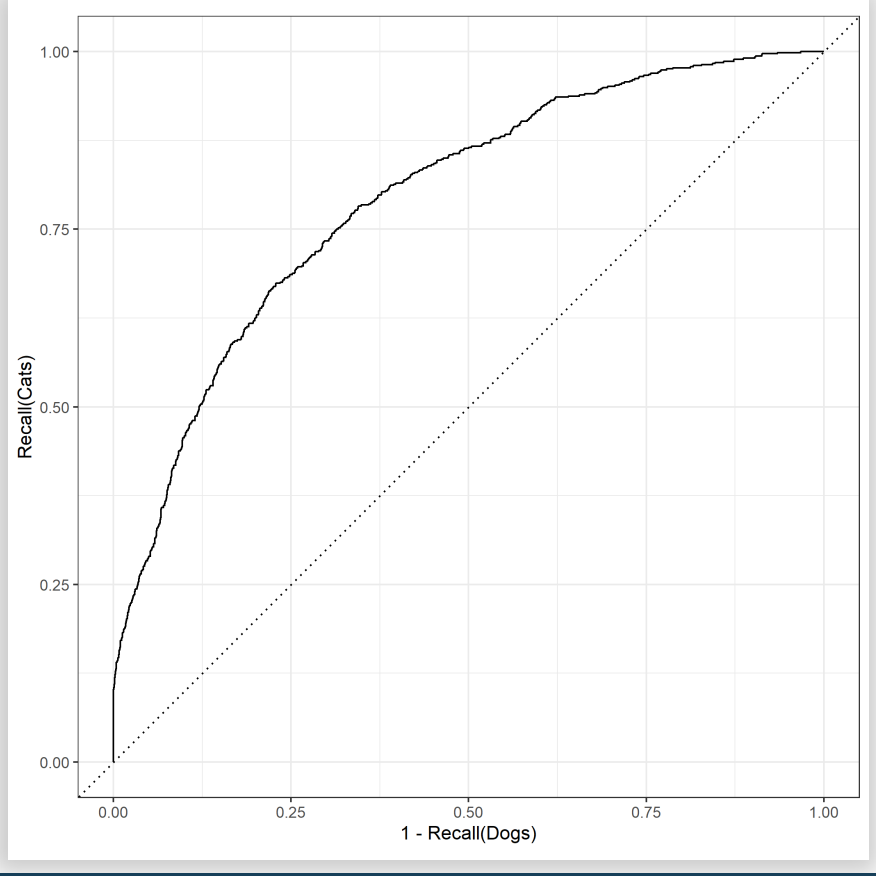
```
## # A tibble: 2 × 3
##   pets      n   pct
##   <fct> <int> <dbl>
## 1 cats    653 0.165
## 2 dogs   3307 0.835
```

```
okcupid_test_baked <- rec_cutoff %>%
  bake(okcupid_test2)

mod_lr_test <- logistic_reg(penalty = 0.01, mixture = 0.5) %>%
  set_engine("glmnet")

mod_lr_test <- mod_lr_test %>%
  fit(pets ~ ., data = ok_tr_for_cutoff)

pred_test <-  mod_lr_test %>%
  predict(new_data = okcupid_test_baked, penalty = 0.01, type = "p
  mutate(truth = okcupid_test_baked$pets)
```

```
roc_auc(pred_test, truth, .pred_cats)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.792
```

```
roc_lr <- roc_curve(
  tibble(truth = pred_test$truth, cats = pred_test$.pred_cats),
  truth, cats)

autoplot(roc_lr) +
  labs(x = "1 - Recall(Dogs)", y = "Recall(Cats)")
```



```
roc_obj <- roc(pred_test$truth, pred_test$.pred_cats,
               levels = c("dogs", "cats"))

cutoffs <- c(0.3, 0.5, 0.9)

coords(roc_obj, x = cutoffs,
               ret = c("accuracy", "recall", "precision",
                       "specificity", "npv"),
               transpose = TRUE)
```

```
##                   [,1]      [,2]      [,3]
## accuracy    0.4608586 0.7361111 0.8520202
## recall      0.9356815 0.6875957 0.1056662
## precision   0.2259615 0.3480620 0.9718310
## specificity 0.3671001 0.7456910 0.9993952
## npv         0.9665605 0.9235955 0.8498329
```

What *is* the model?

```
tidy(mod_lr_test) %>%
  filter(near(penalty, 0.01, tol = 0.0005)) %>%
  arrange(-abs(estimate)) %>%
  select(term, estimate, penalty) %>%
  head(20)
```

```
## # A tibble: 20 × 3
##    term                estimate penalty
##    <chr>                  <dbl>   <dbl>
##  1 w_my_cat              -2.13     0.01
##  2 w_my_dog               1.57     0.01
##  3 w_kitt                -1.25     0.01
##  4 w_love_dog             0.840    0.01
##  5 w_cats                -0.772    0.01
##  6 religion_christian     0.764    0.01
##  7 (Intercept)           -0.652    0.01
##  8 w_dog                  0.648    0.01
##  9 religion_jewish        0.613    0.01
## 10 body_type_thin        -0.587    0.01
## 11 diet_all_else         -0.529    0.01
## 12 drugs_sometimes       -0.509    0.01
## 13 status_all_else       -0.508    0.01
## 14 diet_vegetarian       -0.485    0.01
## 15 job_marketing          0.480    0.01
```

What did we achieve?

- ✔️ Reproducible model
- ✔️ Dependable model
- ✔️ Explainable model
- ✔️ Small and Fast as hell model (put in on a nanochip!)
- ❌ Impressive model

Is that the end?

No.

> 💡 What *didn't* we do?

Few Excellent Books