

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

2020-05-02

APPLICATIONS



OF DATA SCIENCE

What can you possibly teach us about Feature Engineering?

APPLICATIONS



OF DATA SCIENCE

Our two previous modeling attempts, with [Modeling in the Tidyverse](#) and in [The Trees](#) where problematic.

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

1. Predict the income of an OKCupid user (from those who reported income)
2. 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  
                      "ikes dogs and dislikes cats")
```

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

On our validation set we got...

- Regression Task

Model	RMSE	R
LM	0.35	0.5
LM-L2	0.35	0.5
LM-L1	0.35	0.5
CART	0.36	0.45
Bagging	0.37	0.44
RF	0.35	0.49
GBT-gbm	0.35	0.5
GBT-XGBOOST	0.35	0.5

- Classification Task

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

APPLICATIONS



OF DATA SCIENCE

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.)

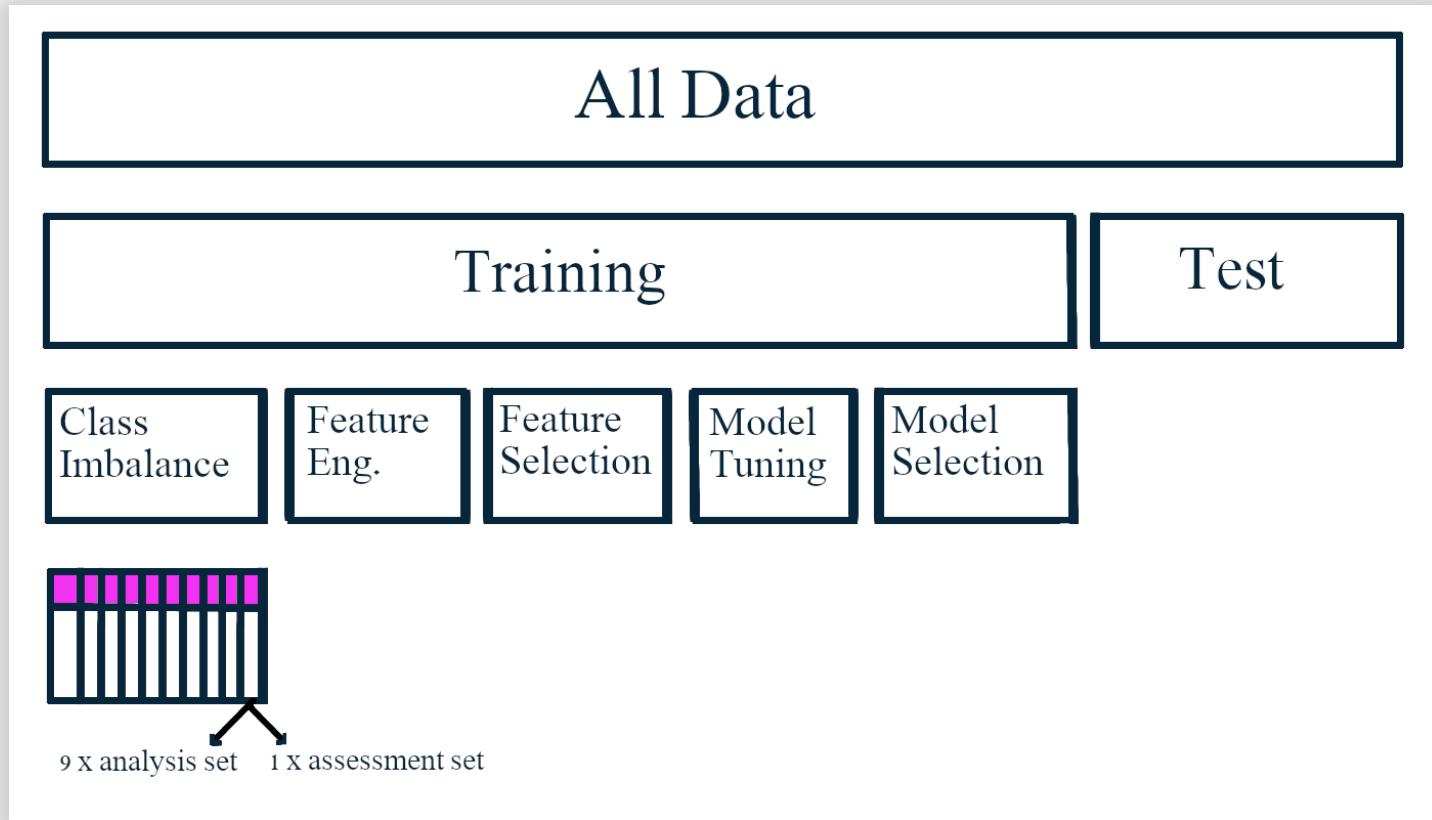
- Resampling is a key principle when developing a ML solution
- All ML practitioners are aware of the menace of overfitting
- But resampling should also be made for:
 - Feature engineering and transformation (should I use the Box-Cox or Yeo-Johnson transformation?)
 - Feature selection (Should I use 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:

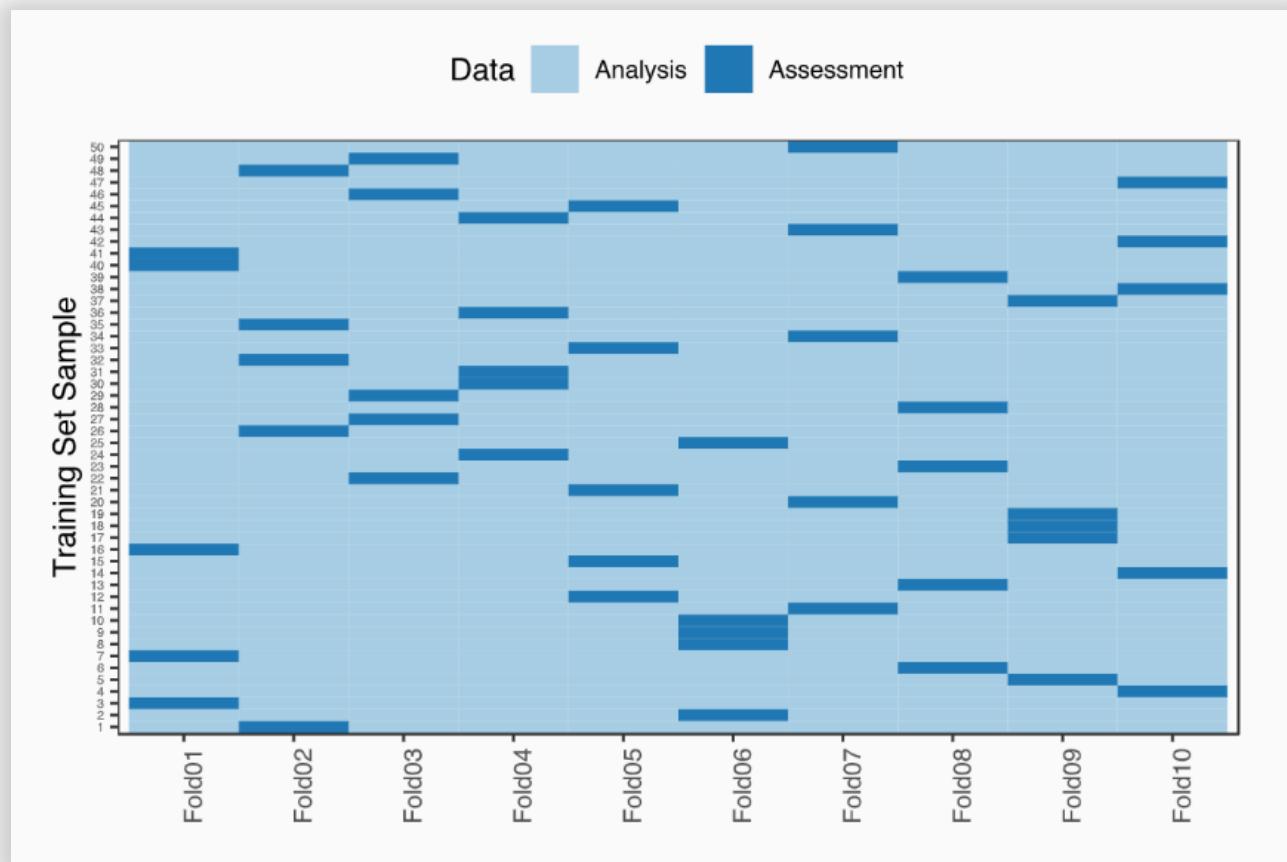
1. On the entire training data, then test on the testing data and get a realistic metric of performance
2. On the *entire* data, right before deployment to production

Many approaches to resampling

But two most common:

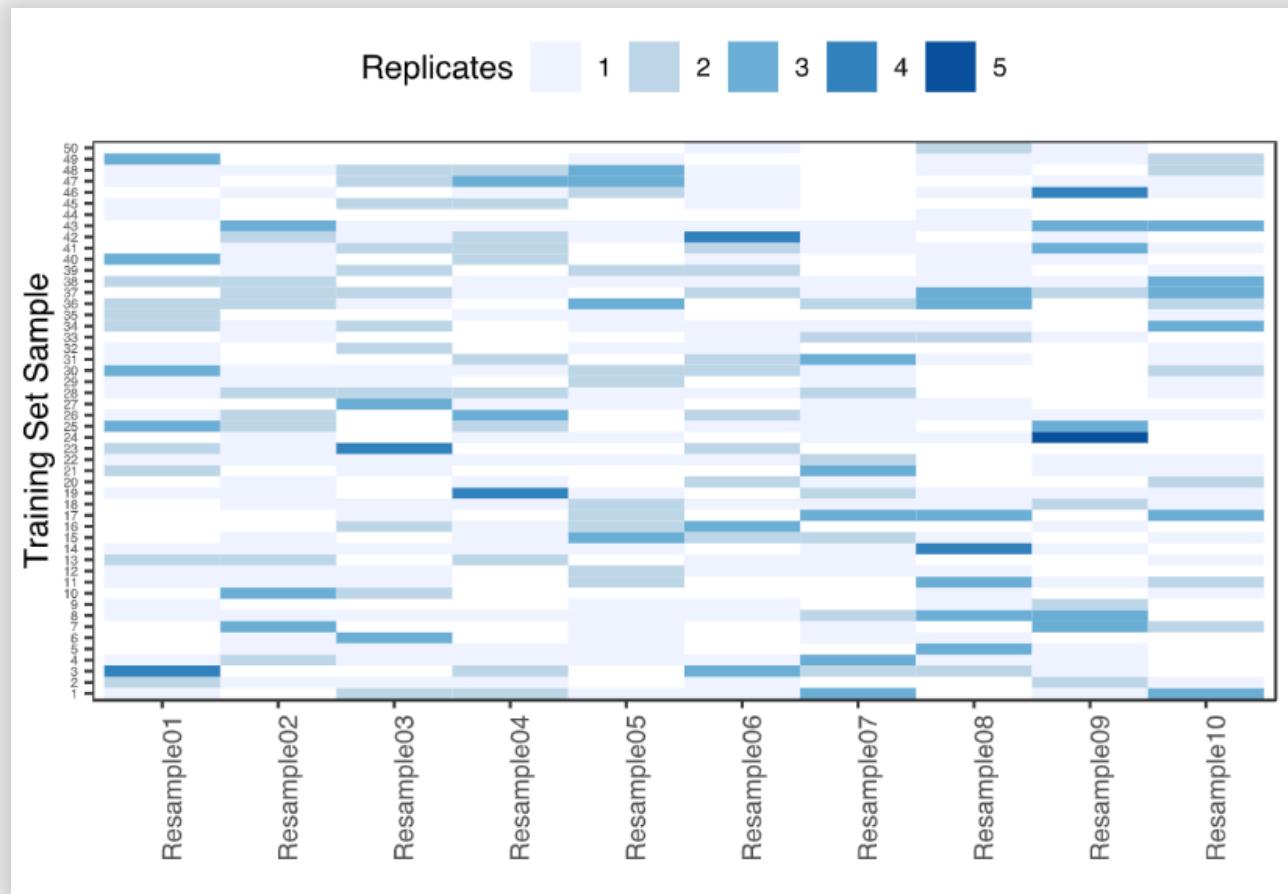
1. V-fold Cross Validation (+ optionally repeating n times)
2. Bootstrap Samples

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

APPLICATIONS



OF DATA SCIENCE

OkCupid from scratch

```
okcupid <- read_csv("../data/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 dislikes  
                      "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) %>%  
  na_if(-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)}\n      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 x 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 x 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.

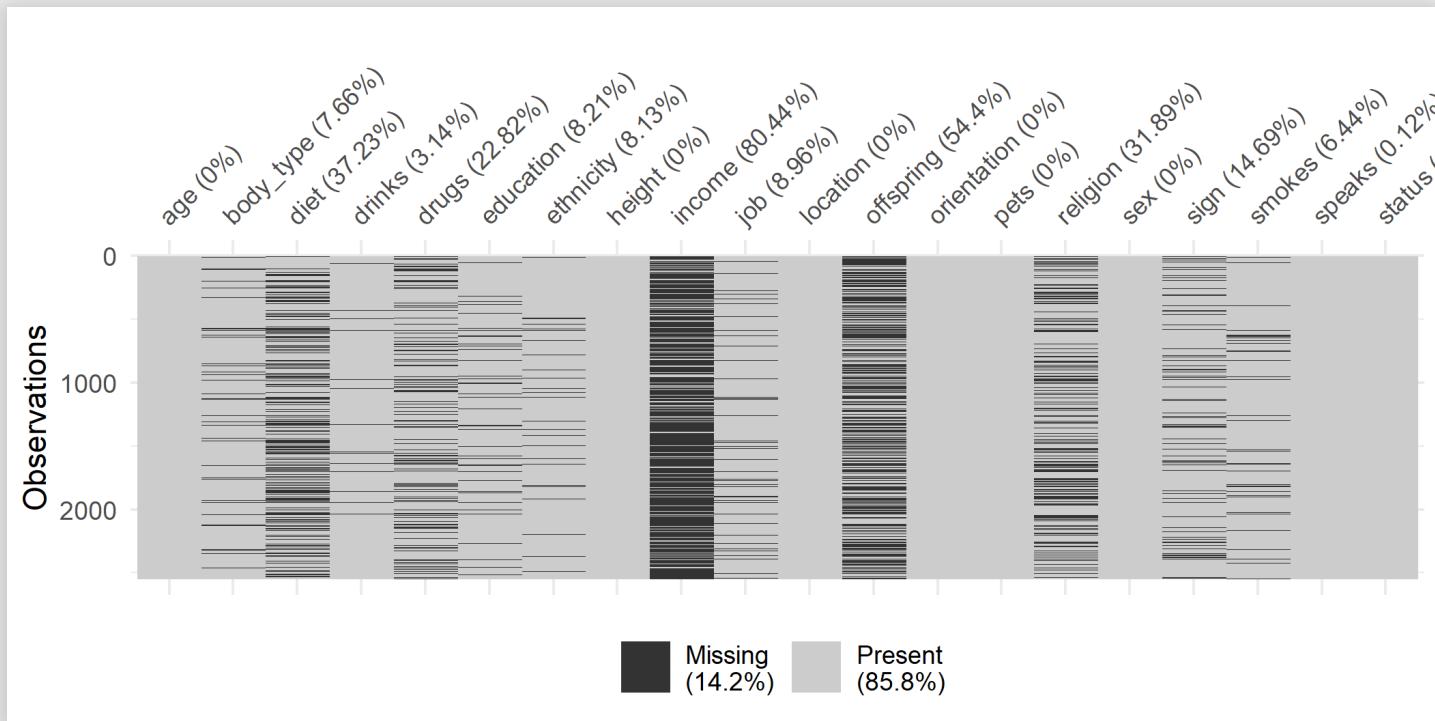
We explore the training data (skipping here at least a couple of hours in which we actually explore the data, I'm assuming we already know quite a bit!):

```
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 an...
## #> #> $ drinks <chr> "socially", "rarely", NA, "socially", "socially", ...
## #> #> $ drugs <chr> "never", "sometimes", NA, NA, "never", NA, "never"...
## #> #> $ education <chr> "graduated from college/university", NA, NA, "grad...
## #> #> $ essay0 <chr> "i am a very positive person who loves to laugh an...
## #> #> $ essay1 <chr> "i work in the health and wellness industry and i ...
## #> #> $ essay2 <chr> "making people smile or laugh.", "ummmm everything...
## #> #> $ essay3 <chr> "my eyes", "my boobs. definitely.<br />\nand the f...
## #> #> $ essay4 <chr> "i enjoy malcolm gladwell, blink, tipping point, o...
## #> #> $ essay5 <chr> "water,steak,chocolate,comfy jeans,transportation,..
## #> #> $ essay6 <chr> "food, naps, women, and health", "caffeine, my bab...
## #> #> $ essay7 <chr> "doing whatever has come up or has been planned. d...
## #> #> $ essay8 <chr> "i am totally deaf in my right ear", "i love anima...
## #> #> $ essay9 <chr> "you enjoy having fun, love to laugh, have a posit...
## #> #> $ ethnicity <chr> "white", "white", NA, "white", "asian, pacific isl...
## #> #> $ 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, ...
## #> #> $ job <chr> "medicine / health", "sales / marketing / biz dev"...
## #> #> $ location <chr> "mountain view, california", "pleasant hill, calif...
```

```
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. Gonna have to take care of it.

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 `essays`.

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 a 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)

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 / 3000))
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 - 1) / 3000)
ok_tr_tuning <- training(ok_tuning_split)
ok_tr_eng <- testing(ok_tuning_split)

```

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

Dealing with Missing Data

APPLICATIONS



OF DATA SCIENCE

Missing Data

We could fill an entire semester on this subject!

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

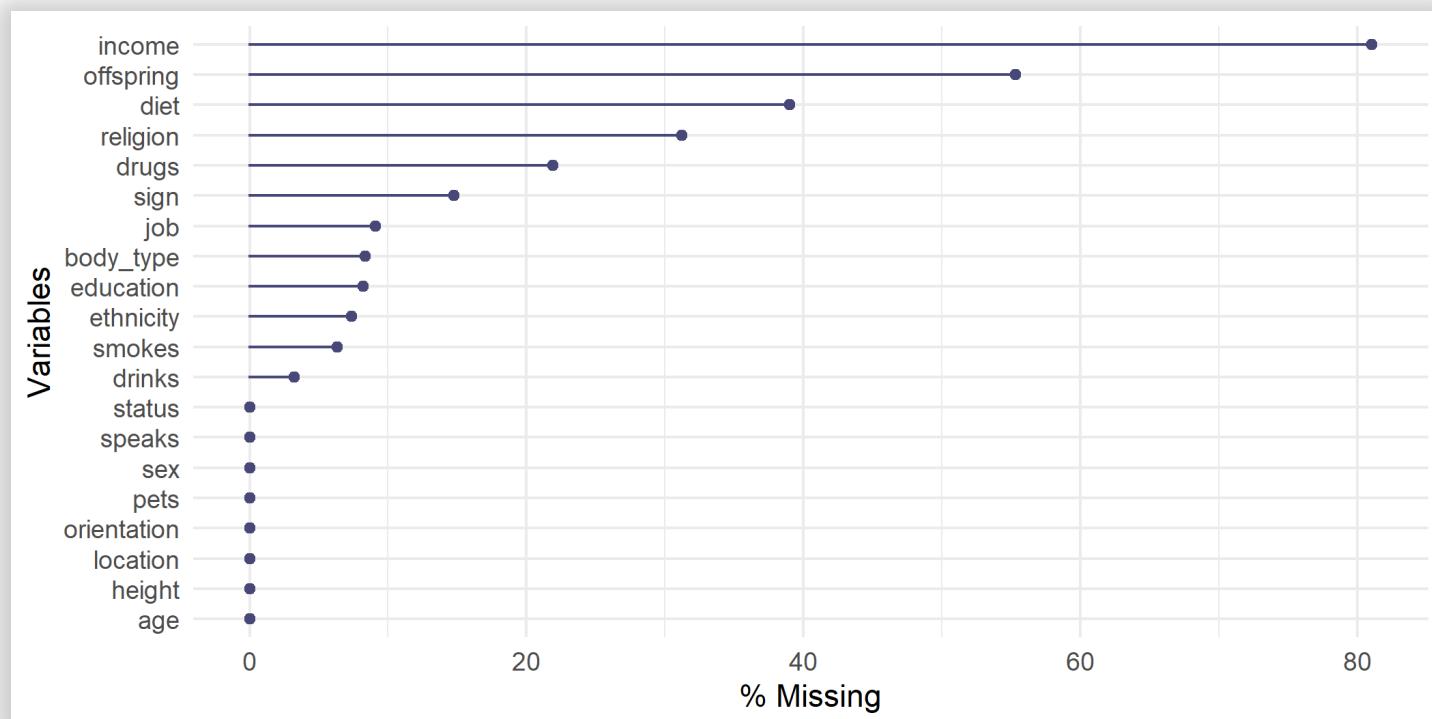
- What is the missing data mechanism?
 - See [Little & Rubin \(2019\)](#) seminal work
 - MCAR: "on some days the weighing scale batteries drained and we couldn't take the mice weight"
 - MAR: "on the 2024 elections poll a massive Facebook campaign called all women to refuse to cooperate with the Seker-Sheker 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?

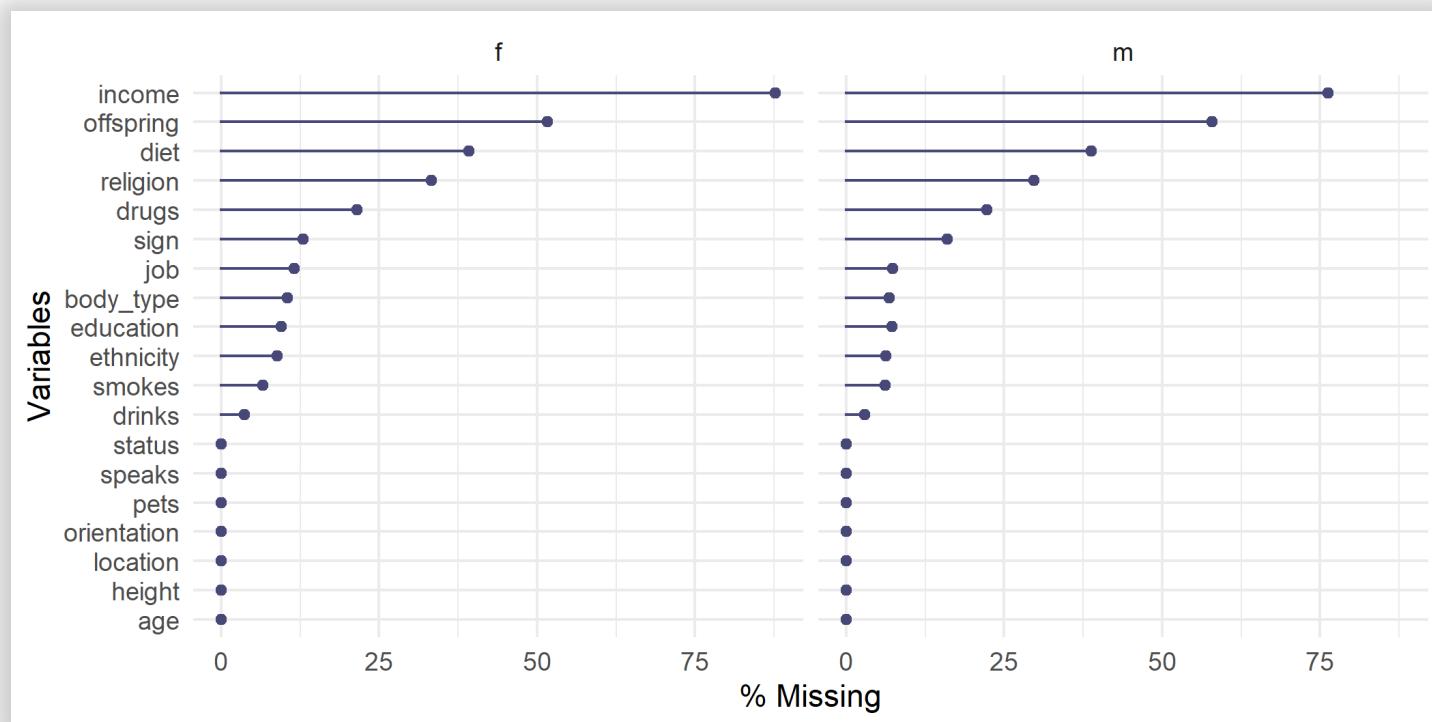
Exploring percent missingness

```
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)
```

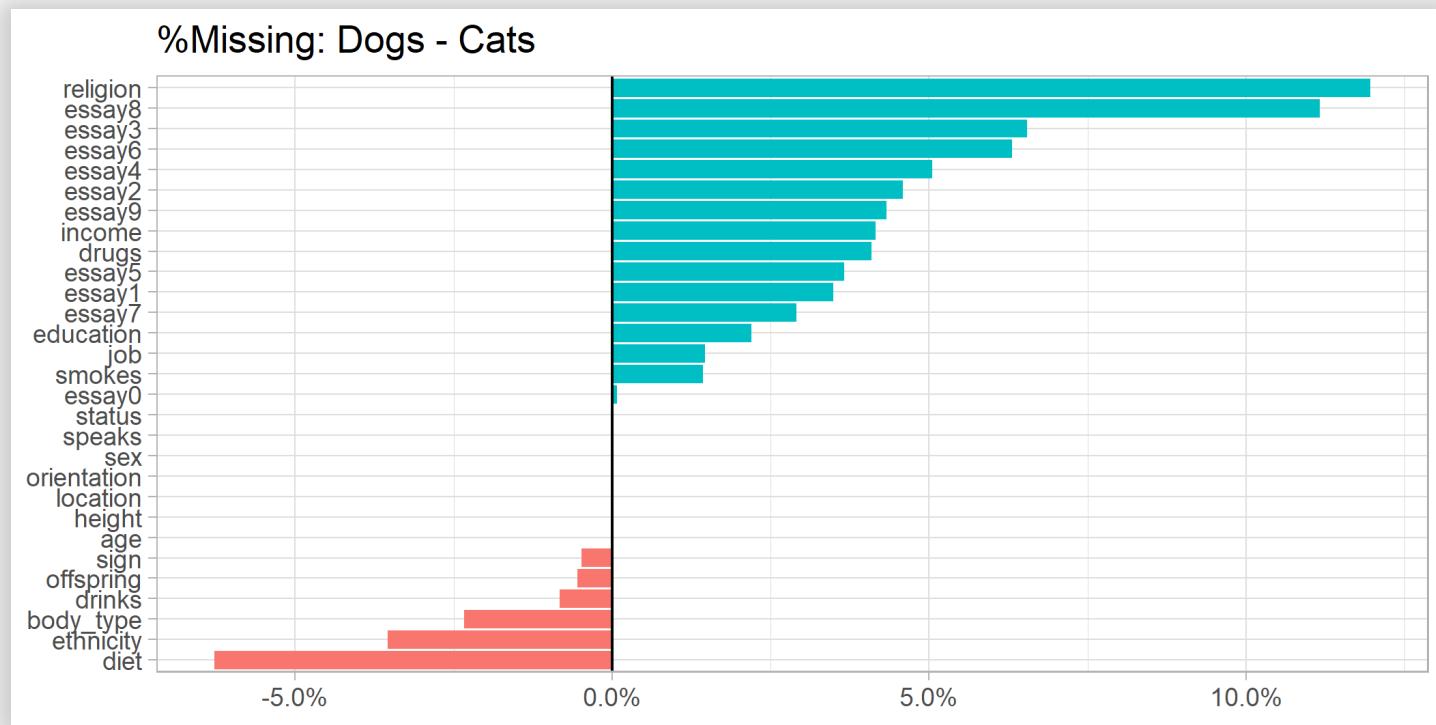


Exploring missingness between a predictor and the response (categorical) variable

```
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 x 4
##   pets    `FALSE` `TRUE` pct_missing
##   <chr>    <int>   <int>      <dbl>
## 1 cats       74     256      0.776
## 2 dogs      305    1365      0.817
```

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 = NULL, y = NULL) + theme_light()
```



Girl, 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 x 4
##   status      `FALSE` `TRUE` pct_missing
##   <chr>       <int>   <int>     <dbl>
## 1 available     46      8     0.148
## 2 married        7      1     0.125
## 3 seeing someone 42      5     0.106
## 4 single       1738    153    0.0809
```

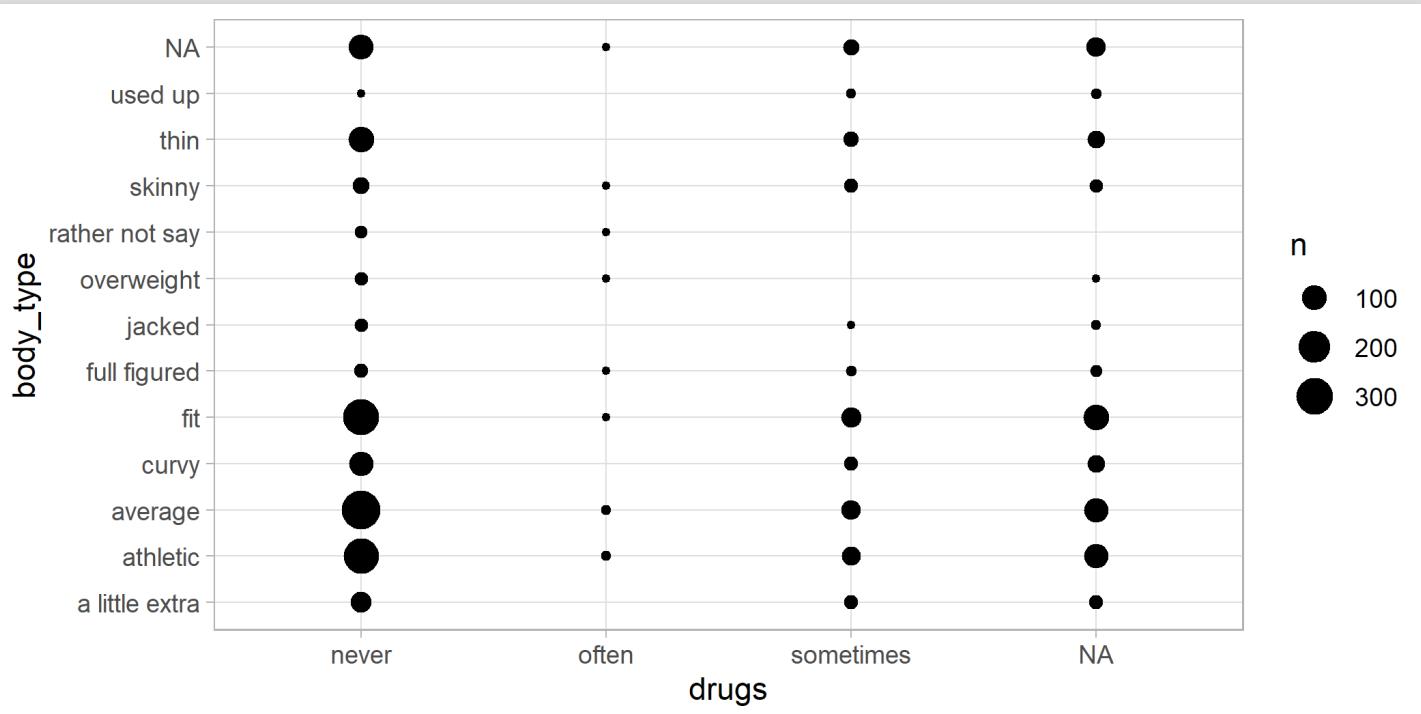
The same with body_type missingness by drugs:

```
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 x 4
##   drugs_missing `FALSE` `TRUE` pct_missing
##   <lgl>          <int>  <int>     <dbl>
## 1 FALSE           1440    122      0.0781
## 2 TRUE            393     45       0.103
```

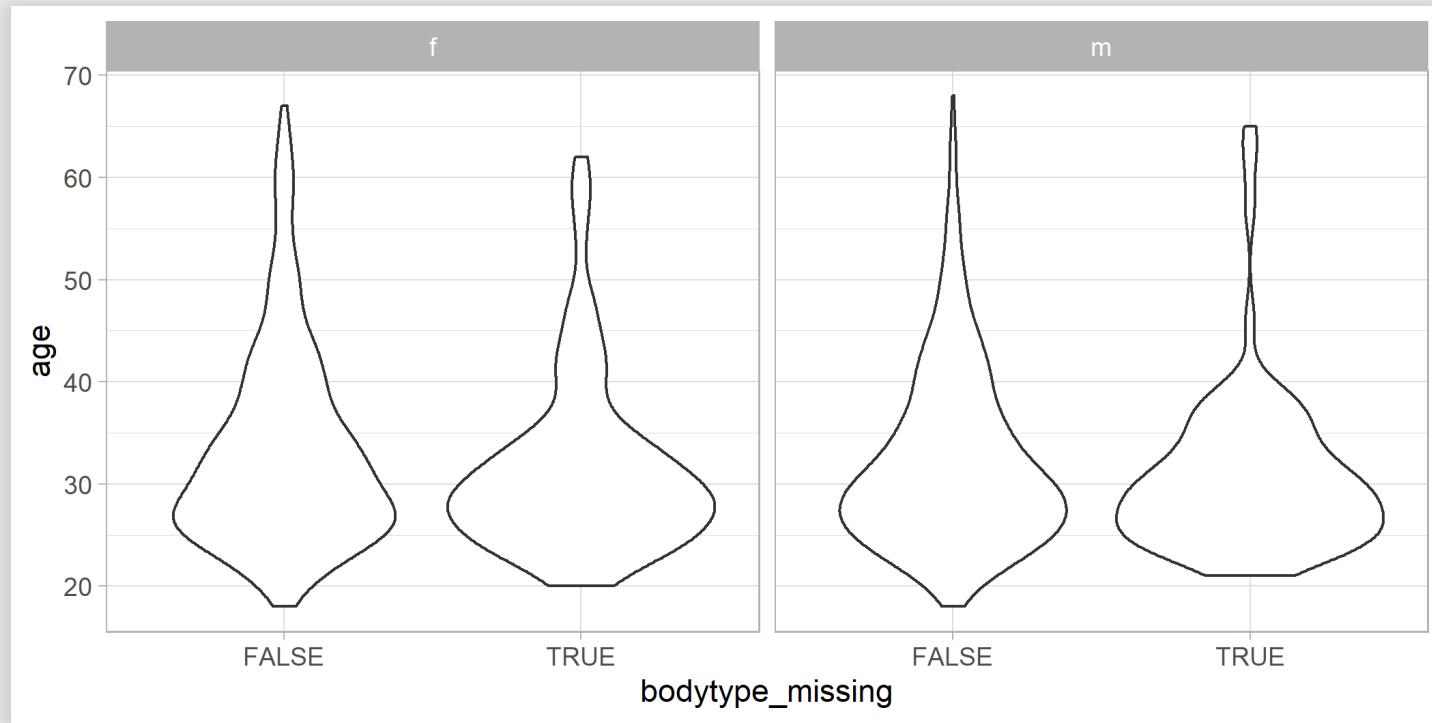
But we can do better with a co-occurrence plot or a heatmap

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



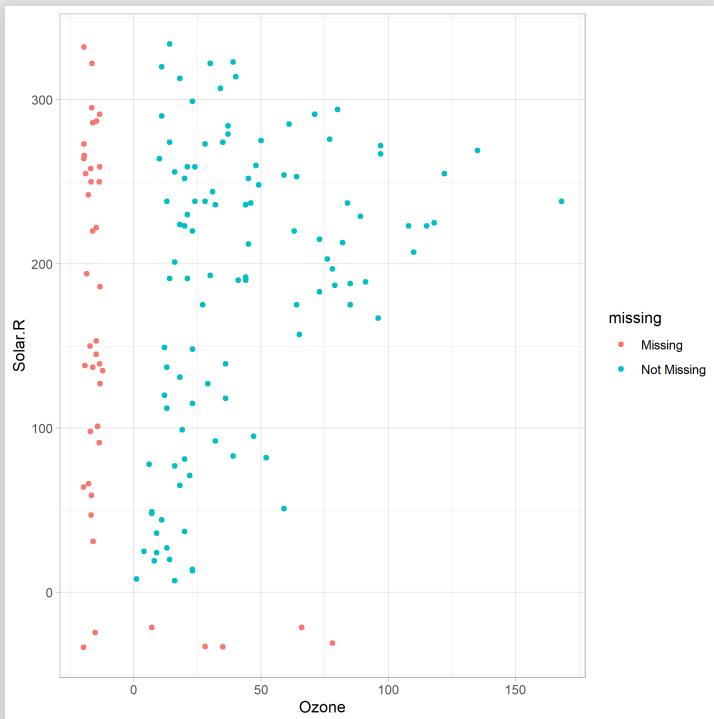
Exploring missingness between numerical and categorical predictors

```
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)

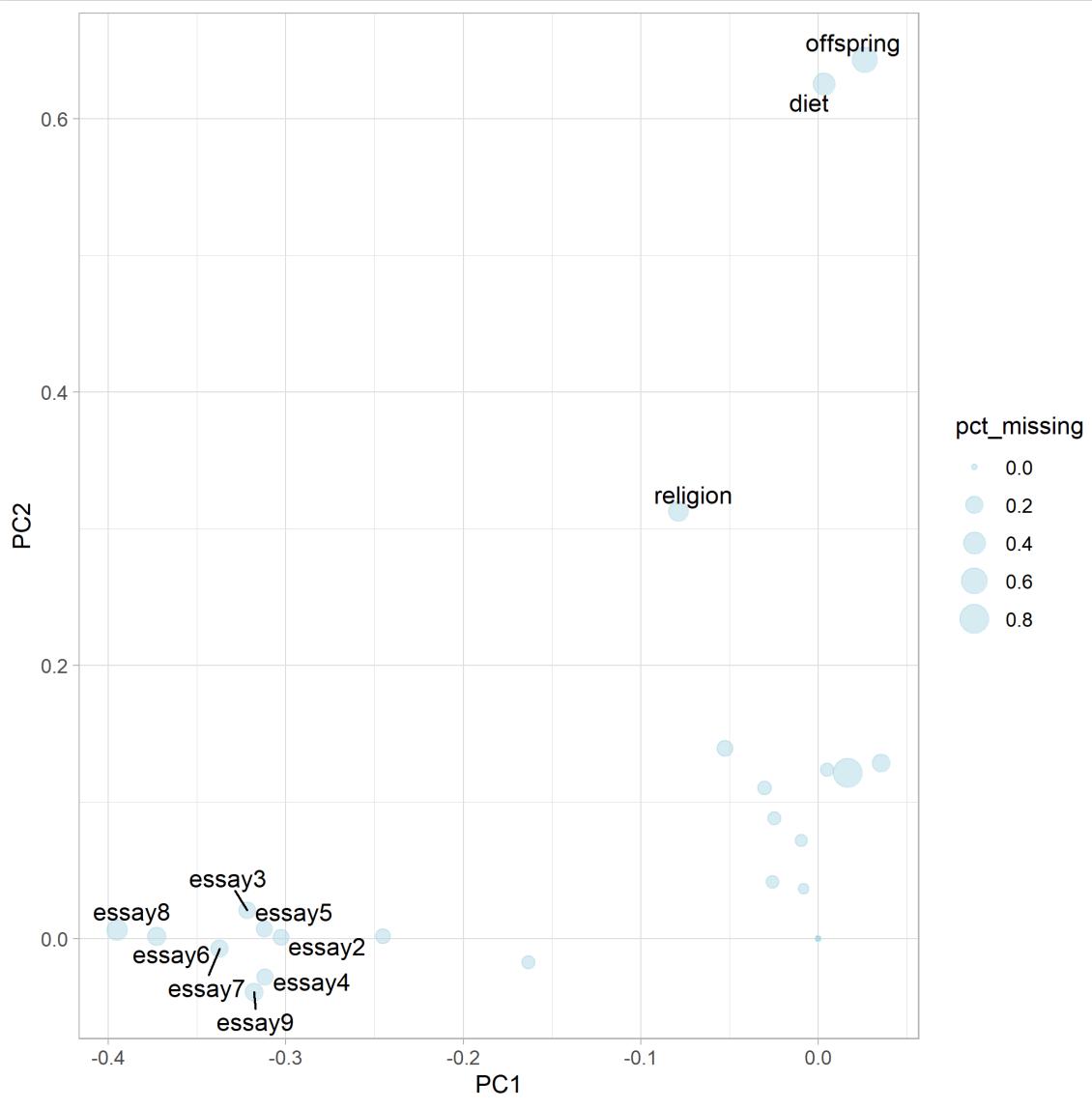
```
ggplot(airquality, aes(Ozone, Solar.R)) +  
  naniar::geom_miss_point() +  
  theme_light()
```



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

```
binary_mat <- ok_tr_missing %>%
  select(-pets) %>%
  mutate_all(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()
```



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 $2 \times 3 \times 3 = 18$ 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 %>%
  select(-starts_with("essay")) %>%
  bind_cols(
    ok_tr_missing %>%
      transmute_at(vars(essay0:essay9),
                   list("len" = ~ifelse(is.na(.x), 0, str_length(.x))))
  ) %>%
  bind_cols(
    ok_tr_missing %>%
      transmute_at(vars(essay0:essay9),
                   list("isna" = ~ifelse(is.na(.x), 1, 0)))
  ) %>%
  rename_at(vars(contains("_len")),
            list(~paste("len", gsub("_len", "", .),
                       sep = " ") ) ) %>%
  rename_at(vars(contains("_isna")),
            list(~paste("isna", gsub("_isna", "", .),
                       sep = " ") ) ) %>%
  mutate_if(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)
  update_role(income, contains("essay"), new_role = "discarded") %
    step_normalize(all_numeric(), -has_role("discarded")) %>%
    step_other(all_nominal(), -has_role("discarded"), other = "other")
    step_novel(all_nominal(), -has_role("discarded"), -all_outcomes)
    step_unknown(all_nominal(), -has_role("discarded"), -all_outcomes)
    step_dummy(all_nominal(), -all_outcomes(),
              -has_role("discarded"), one_hot = FALSE)

rec_miss_mode_none_none <- recipe(pets ~ ., data = ok_tr_missing2)
  update_role(income, contains("essay"), new_role = "discarded") %
    step_normalize(all_numeric(), -has_role("discarded")) %>%
    step_other(all_nominal(), -has_role("discarded"), other = "other")
    step_novel(all_nominal(), -has_role("discarded"), -all_outcomes)
    step_modeimpute(all_nominal(), -has_role("discarded"), -all_outcomes)
    step_dummy(all_nominal(), -all_outcomes(),
              -has_role("discarded"), one_hot = FALSE)
```

(pretty sure `step_rm` is a way to discard predictors altogether)

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 x 2
##   splits          id
##   <named list>  <chr>
## 1 <split [1.8K/200]> Fold01
## 2 <split [1.8K/200]> Fold02
## 3 <split [1.8K/200]> Fold03
## 4 <split [1.8K/200]> Fold04
## 5 <split [1.8K/200]> Fold05
## 6 <split [1.8K/200]> Fold06
## 7 <split [1.8K/200]> Fold07
## 8 <split [1.8K/200]> Fold08
## 9 <split [1.8K/200]> Fold09
## 10 <split [1.8K/200]> Fold10
```

Applying (two) recipes to all splits

```
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_info)

##   1   2   3   4   5   6   7   8   9   10
## 112 113 113 113 113 113 113 112 112 112

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

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

Defining the model specification

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

Fitting the model to all splits

```
fit_lr <- function(rec_obj)
  fit(mod_lr, pets ~ .,
       data = juice(rec_obj, all_predictors(), all_outcomes()))

cv_splits_missing$mod_lr_unk <- map(
  cv_splits_missing$rec_miss_unk_none_none, fit_lr)
cv_splits_missing$mod_lr_mode <- map(
  cv_splits_missing$rec_miss_mode_none_none, fit_lr)
```

Predicting on assessment data of each split

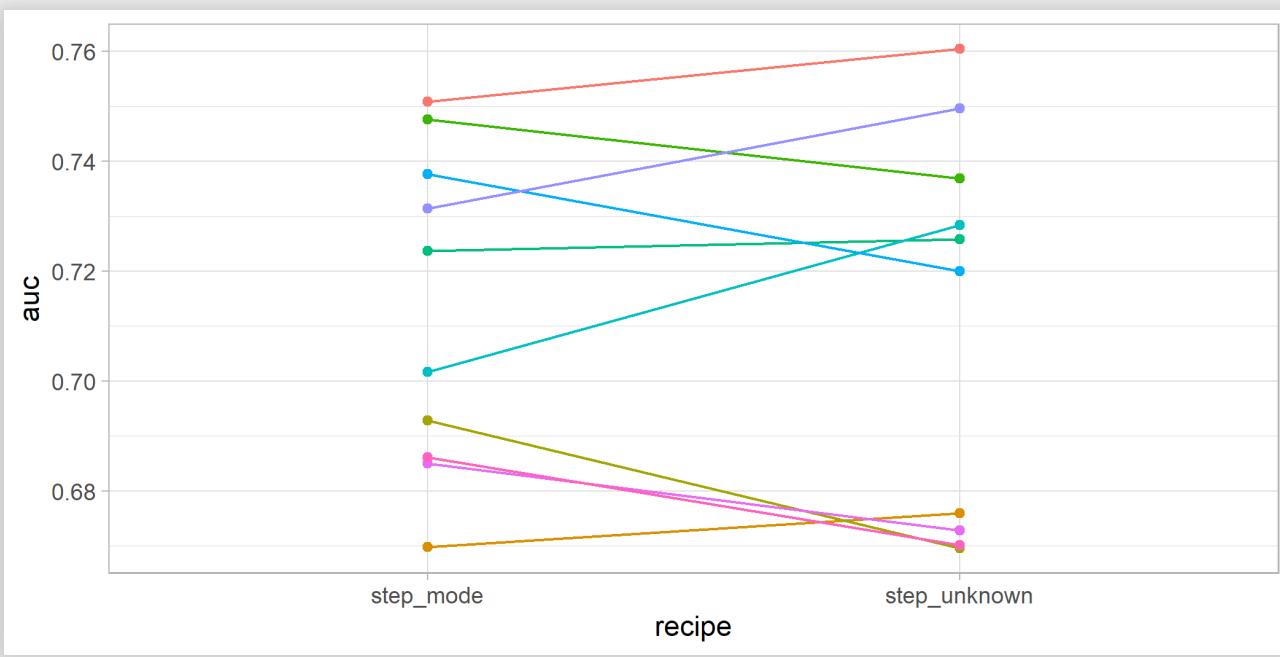
```
pred_lr <- function(split_obj, rec_obj, model_obj) {  
  mod_data <- bake(rec_obj,  
                     new_data = assessment(split_obj),  
                     all_predictors(), all_outcomes())  
  out <- mod_data %>% select(pets)  
  out$predicted <- predict(model_obj, mod_data, type = "prob",  
                            penalty = 0.0001)$ .pred_cats  
  out  
}  
  
cv_splits_missing$pred_unk <-  
  pmap(  
    lst(  
      split_obj = cv_splits_missing$splits,  
      rec_obj = cv_splits_missing$rec_miss_unk_none_none,  
      model_obj = cv_splits_missing$mod_lr_unk  
    ),  
    pred_lr)  
cv_splits_missing$pred_mode <- pmap(lst(  
  split_obj = cv_splits_missing$splits,  
  rec_obj = cv_splits_missing$rec_miss_mode_none_none,  
  model_obj = cv_splits_missing$mod_lr_mode  
),  
  pred_lr)
```

Measuring the AUC of each split

```
cv_splits_missing$step_unknown <- map_dfr(  
  cv_splits_missing$pred_unk, roc_auc, pets, predicted)$estimate  
cv_splits_missing$step_mode <- map_dfr(  
  cv_splits_missing$pred_mode, roc_auc, pets, predicted)$estimate
```

Comparing AUCs with a plot

```
cv_splits_missing %>%
  select(id, step_unknown, step_mode) %>%
  pivot_longer(cols = c(step_unknown, step_mode),
               names_to = "recipe", values_to = "auc") %>%
  ggplot(aes(recipe, auc, group = id, color = id)) +
  geom_line() +
  geom_point() +
  guides(color = FALSE) + theme_light()
```



Comparing AUCs with a statistical test (paired T test)

```
t.test(cv_splits_missing$step_unknown,  
       cv_splits_missing$step_mode, paired = TRUE)  
  
##  
##      Paired t-test  
##  
## data:  cv_splits_missing$step_unknown and cv_splits_missing$step_mode  
## t = -0.31928, df = 9, p-value = 0.7568  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -0.01364404  0.01026897  
## sample estimates:  
## mean of the differences  
##                  -0.001687534
```

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

```
rec_miss_unk_mean_none <- recipe(pets ~ ., data = ok_tr_missing2)
  update_role(contains("essay"), new_role = "discarded") %>%
  step_log(income, offset = 1) %>%
  step_meanimpute(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_unknown(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_dummy(all_nominal(), -all_outcomes(),
            -has_role("discarded"), one_hot = FALSE)

rec_miss_mode_mean_none <- recipe(pets ~ ., data = ok_tr_missing2)
  update_role(contains("essay"), new_role = "discarded") %>%
  step_log(income, offset = 1) %>%
  step_meanimpute(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_dummy(all_nominal(), -all_outcomes(),
            -has_role("discarded"), one_hot = FALSE)
```

Our approach with 2 recipes does not scale, and we don't need to keep all data just final AUC for each recipe. We are writing a `train_recipe()` function!

```
lst_recos <- 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)

train_recipe <- function(rec, rec_name, splits) {
  splits_prep <- map(splits, prepper, recipe = rec)
  splits_fit <- map(splits_prep, fit_lr)
  splits_pred <- pmap(
    lst(
      split_obj = splits,
      rec_obj = splits_prep,
      model_obj = splits_fit
    ),
    pred_lr
  )
  res <- map_dfr(splits_pred, roc_auc, pets, predicted)$estimate
  names(res) <- rec_name
  res
}
```

Training multiple recipes, saving the AUC for each recipe, each split.

```
cv_splits_missing <- cv_splits_missing %>%
  bind_cols(
    map2_dfc(lst_recipes, names(lst_recipes), train_recipe,
             splits = cv_splits_missing$splits)
  )
```

Processing the results into a nice table:

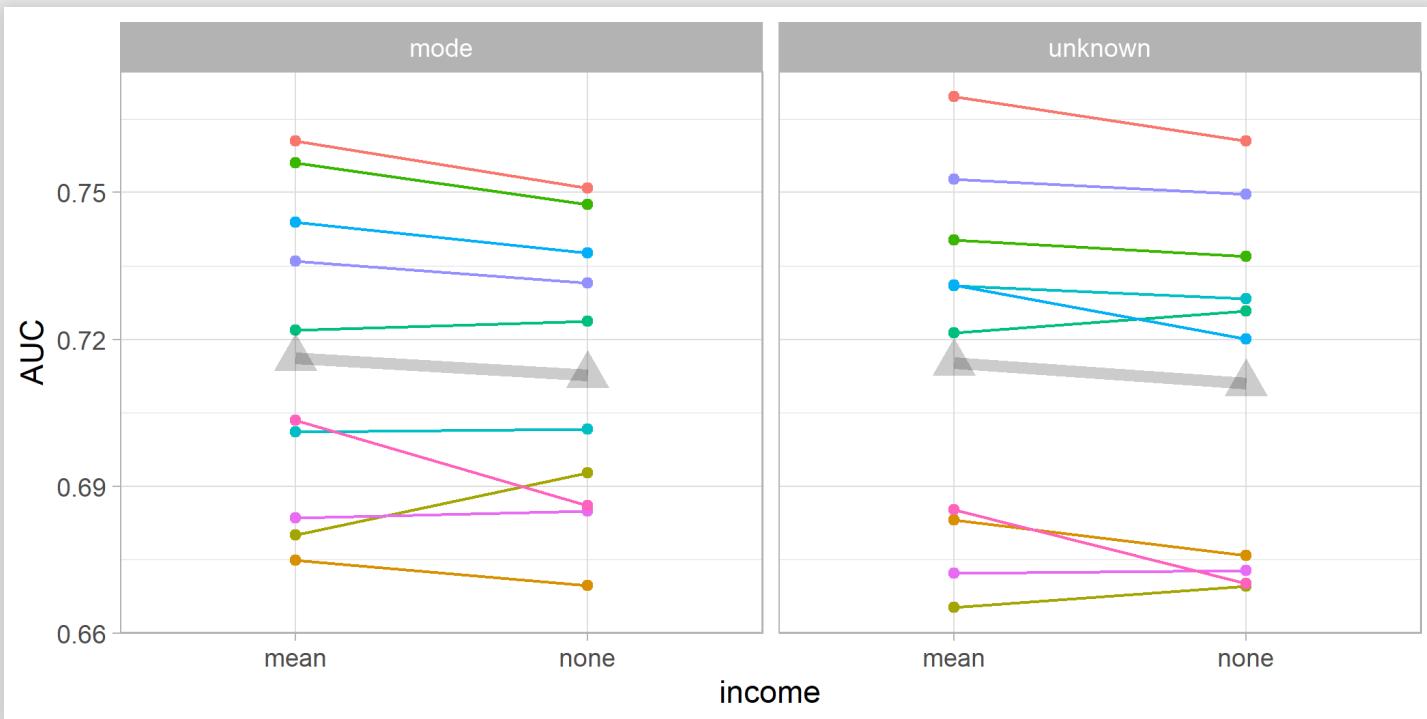
```
cv_res_missing <- cv_splits_missing %>%
  pivot_longer(cols = names(lst_recipes),
               names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC) %>%
  separate(recipe, c("categorical", "income"))

cv_res_missing
```

```
## # A tibble: 40 x 4
##       id   categorical income     AUC
##   <chr>   <chr>      <chr>   <dbl>
## 1 Fold01 unknown    none    0.760
## 2 Fold01 mode       none    0.751
## 3 Fold01 unknown    mean    0.770
## 4 Fold01 mode       mean    0.760
## 5 Fold02 unknown    none    0.676
## 6 Fold02 mode       none    0.670
```

Comparing AUCs with a plot

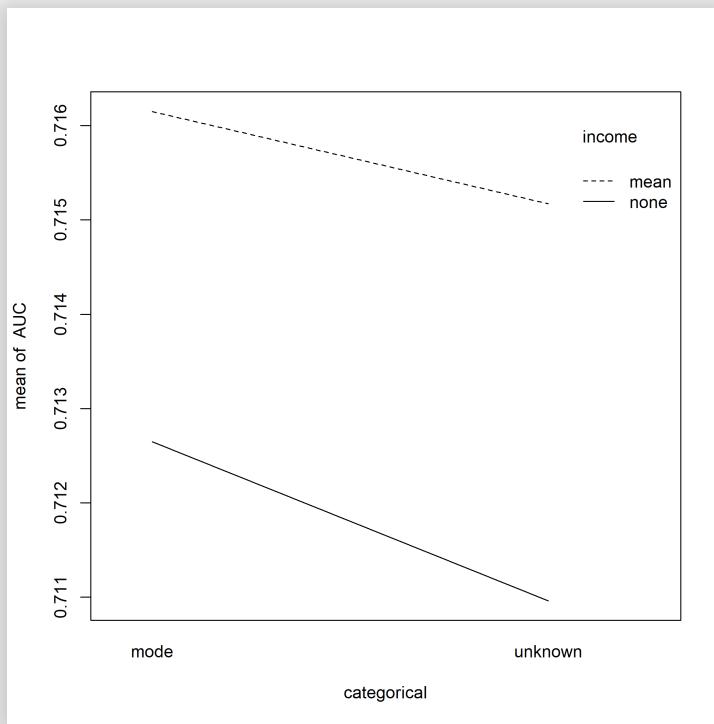
```
cv_res_missing %>%
  ggplot(aes(income, AUC, group = id, color = id)) +
  geom_line() + geom_line(data = cv_res_missing_meds, color = "black") +
  geom_point() + geom_point(data = cv_res_missing_meds, color = "black") +
  guides(color = FALSE) + facet_wrap(. ~ categorical) +
  theme_light()
```



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

```
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 0.03699 0.00411  
##  
## Error: Within  
##           Df   Sum Sq Mean Sq F value Pr(>F)  
## factor(categorical)      1 0.0000178 1.779e-05 0.165 0.688  
## factor(income)          1 0.0001487 1.487e-04 1.379 0.251  
## factor(categorical):factor(income)  1 0.0000013 1.250e-06 0.012 0.915  
## Residuals            27 0.0029116 1.078e-04
```

```
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.

```
lst_recos <- 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  
                )
```

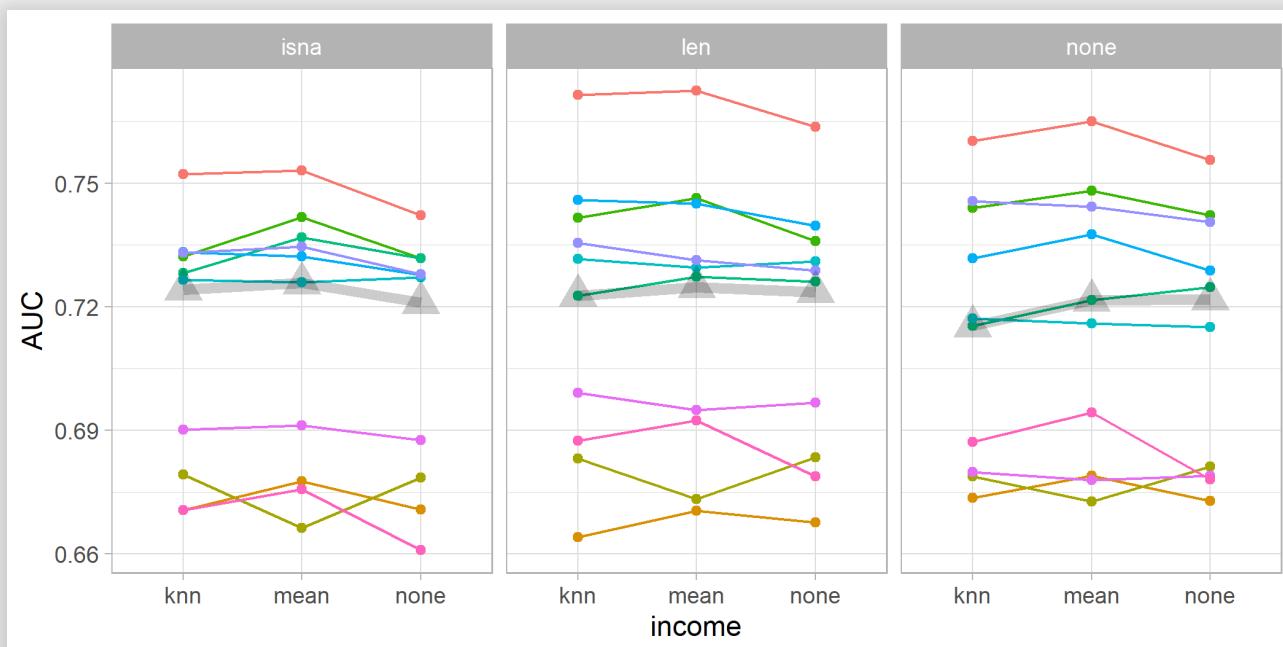
```
set.seed(42)
cv_splits_missing <- vfold_cv(ok_tr_missing2, v = 10, strata = pet
```

```
cv_splits_missing <- cv_splits_missing %>%
  bind_cols(
    map2_dfc(lst_rec, names(lst_rec), train_recipe,
             splits = cv_splits_missing$splits)
  )
```

```
cv_res_missing <- cv_splits_missing %>%
  pivot_longer(cols = names(lst_rec),
               names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC) %>%
  separate(recipe, c("categorical", "income", "essay"))
```

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 = "k")
  geom_point() + geom_point(data = cv_res_missing_meds, color = "k")
  guides(color = FALSE) + facet_wrap(. ~ essay) +
  theme_light()
```



Repeated measures ANOVA

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

## 
## Error: factor(id)
##           Df Sum Sq Mean Sq F value Pr(>F)
## Residuals   9 0.1595 0.01772
##
## Error: Within
##                               Df Sum Sq Mean Sq F value Pr(>F)
## factor(categorical)          1 0.000191 0.0001911 1.975 0.1617
## factor(income)               2 0.000483 0.0002415 2.497 0.0854
## factor(essay)                2 0.001101 0.0005503 5.689 0.0040
## factor(categorical):factor(income) 2 0.000007 0.0000036 0.038 0.9630
## Residuals                   163 0.015766 0.0000967
##
## factor(categorical)
## factor(income)
## factor(essay)                    .
## factor(categorical):factor(income) **
## Residuals
## ---
## 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 x 4
## # Groups:   categorical, income [6]
##       categorical income essay     AUC
##       <chr>        <chr>  <chr>  <dbl>
## 1 mode          mean    len    0.719
## 2 mode          knn     len    0.719
## 3 unknown       knn     len    0.718
## 4 unknown       mean    len    0.718
## 5 mode          mean    none   0.716
## 6 mode          none    len    0.716
## 7 mode          mean    isna   0.715
## 8 unknown       mean    none   0.715
## 9 mode          knn    none   0.715
## 10 unknown      none   len    0.715
```

 So what would you do?

Class Imbalance

APPLICATIONS



OF DATA SCIENCE

Class Imbalance

(We already gave a decent review in [last class](#))

We'll choose between 3 strategies:

- Upsample
- Downsample
- SMOTE and Downsample

```

ok_tr_imbalance2 <- ok_tr_imbalance %>%
  select(-starts_with("essay")) %>%
  bind_cols(
    ok_tr_imbalance %>%
      transmute_at(vars(essay0:essay9),
                   list("len" = ~ifelse(is.na(.x), 0, str_length(.x))))
  ) %>%
  bind_cols(
    ok_tr_imbalance %>%
      transmute_at(vars(essay0:essay9),
                   list("isna" = ~ifelse(is.na(.x), 1, 0)))
  ) %>%
  rename_at(vars(contains("_len")),
            list(~paste("len", gsub("_len", "", .),
                       sep = " "))) %>%
  rename_at(vars(contains("_isna")),
            list(~paste("isna", gsub("_isna", "", .),
                       sep = " "))) %>%
  mutate_if(is.character, as.factor)

```

```

library(themis)

rec_imb_upsample <- recipe(pets ~ ., data = ok_tr_imbalance2) %>%
  update_role(starts_with("isna"), new_role = "discarded") %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_meanimpute(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other") %>%
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes) %>%
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outcomes) %>%
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1)

rec_imb_downsample <- recipe(pets ~ ., data = ok_tr_imbalance2) %>%
  update_role(starts_with("isna"), new_role = "discarded") %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_meanimpute(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other") %>%
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes) %>%
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outcomes) %>%
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_downsample(pets, under_ratio = 1)

```

```
rec_imb_smote <- recipe(pets ~ ., data = ok_tr_imbalance2) %>%
  update_role(starts_with("isna"), new_role = "discarded") %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_meanimpute(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_downsample(pets, under_ratio = 1.5) %>%
  step_smote(pets, over_ratio = 1)
```

CV splits

```
set.seed(42)
cv_splits_imbalance <- vfold_cv(ok_tr_imbalance2, v = 10, strata =
lst_recs <- list("upsample" = rec_imb_upsample,
                  "downsample" = rec_imb_downsample,
                  "smote" = rec_imb_smote)

cv_splits_imbalance <- cv_splits_imbalance %>%
  bind_cols(
    map2_dfc(lst_recs, names(lst_recs), train_recipe,
            splits = cv_splits_imbalance$splits)
  )
```

```
cv_res_imbalance <- cv_splits_imbalance %>%
  pivot_longer(cols = names(lst_rec),
               names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC)
```

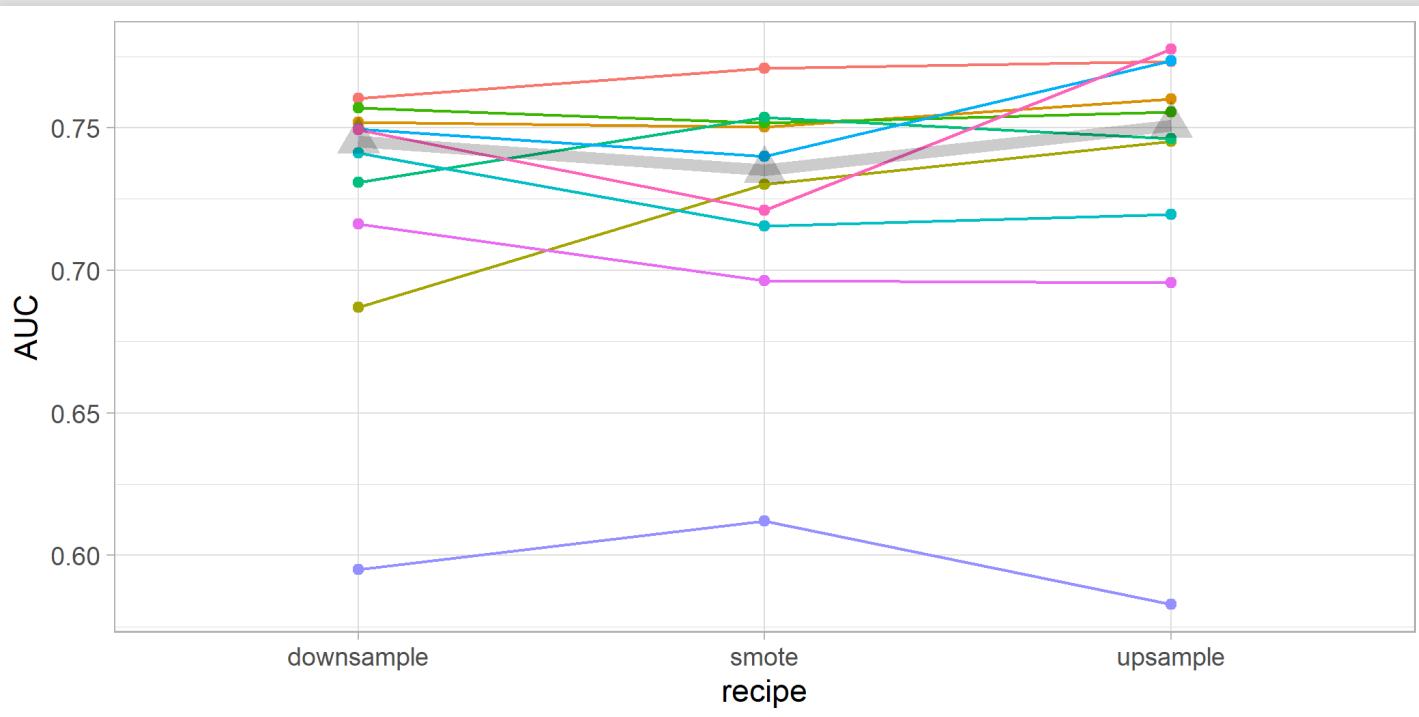
Repeated measures ANOVA:

```
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 0.06694 0.007438
##
## Error: Within
##           Df  Sum Sq  Mean Sq F value Pr(>F)
## factor(recipe)  2 0.000527 0.0002637   0.946  0.407
## Residuals      18 0.005016 0.0002787
```

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 = FALSE) +
  theme_light()
```



```
print(cv_res_imbalance %>%
      group_by(recipe) %>%
      summarise(mean_auc = mean(AUC)) )
```

```
## # A tibble: 3 x 2
##   recipe     mean_auc
##   <chr>       <dbl>
## 1 downsample  0.724
## 2 smote       0.724
## 3 upsample    0.733
```

 So what would you do?

Feat. Engineering and Selection

APPLICATIONS



OF DATA SCIENCE

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

APPLICATIONS



OF DATA SCIENCE

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

```
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) %>%
  juice()

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

```
head(one_hot)
```

```
## # A tibble: 6 x 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 x 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) %>%
  juice()

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

```
head(one_hot)
```

```
## # A tibble: 6 x 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 x 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(df$y))
```

```
## [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(c  
## [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(df$y[df$x1 == "b"] + df$y[df$x2 == "d"]))
```

[1] 0.6443525

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

[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: 71 x 2
##   location             n
##   <chr>                <int>
## 1 alameda, california      53
## 2 albany, california        9
## 3 ashland, california       1
## 4 atherton, california       8
## 5 belmont, california      12
## 6 belvedere tiburon, california    7
## 7 benicia, california      13
## 8 berkeley, california     186
## 9 brisbane, california        4
## 10 burlingame, california     23
## # ... with 61 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

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(), threshold = 0.05) %>%
  step_novel(all_nominal()) %>%
  prep(ok_tr_eng) %>%
  juice() %>%
  count(location)
```

```
## # A tibble: 4 x 2
##   location              n
##   <fct>                <int>
## 1 berkeley, california    186
## 2 oakland, california     312
## 3 san francisco, california 1610
## 4 other                   892
```

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

```
education2 = case_when(
  education == "graduated from high school" ~ "high_school",
  education == "graduated from two-year college" ~ "college0",
  education == "graduated from college/university" ~ "degree1",
  education == "graduated from masters program" ~ "degree2",
  education == "graduated from ph.d program" ~ "degree3",
  education == "working on two-year college" ~ "student0",
  education == "working on college/university" ~ "student1",
  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 x 2
##       sign          n
##   <chr>     <int>
## 1 aquarius      61
## 2 aquarius and it's fun to think about    77
## 3 aquarius but it doesn't matter        83
## 4 aries         73
## 5 aries and it's fun to think about    68
## 6 aries but it doesn't matter        69
## 7 cancer        59
```

```

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) %>%
  juice() %>%
  count(sign)

```

```

## # A tibble: 13 x 2
##       sign      n
##   <fct>    <int>
## 1 capricorn     192
## 2 libra        218
## 3 taurus        247
## 4 leo           226
## 5 aries         210
## 6 aquarius      221
## 7 pisces        223
## 8 sagittarius    176
## 9 scorpio        214
## 10 cancer        203
## 11 gemini        218
## 12 virgo          214
## 13 unknown       438

```

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) %>%
  juice() %>%
  count(location)
```

```
## # A tibble: 6 x 2
##   location     n
##   <fct>     <int>
## 1 0             2
## 2 1            1934
## 3 2            871
## 4 3            155
## 5 4            23
## 6 5            15
```

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

```
## # A tibble: 3,000 x 6
##   location locationL_X1 locationL_X2 locationL_X3 locationL_X4
##   <fct>     <dbl>       <dbl>       <dbl>       <dbl>
## 1 mountai~      1          0          0          0
## 2 oakland~      0          1          0          0
## 3 san fra~      1          0          0          0
## 4 san pab~      0          1          0          0
## 5 oakland~      0          1          0          0
## 6 san fra~      1          0          0          0
## 7 san raf~      0          1          0          0
## 8 san fra~      1          0          0          0
## 9 san fra~      1          0          0          0
## 10 alameda~     0          1          0          0
## # ... with 2,990 more rows, and 1 more variable: locationL_X5 <dbl>
```

It's not a good idea because

- Different locations 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 100 to 6 dummy variables
- Fast
- Generalizes well: any new location can be mapped this way
- Scales: I can do this for millions of locations, 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) %>%
  juice()
```

```
## # A tibble: 3,000 x 17
##   location_hash01 location_hash02 location_hash03 location_hash04
##       <dbl>          <dbl>          <dbl>          <dbl>
## 1 0             0               0               1
## 2 0             1               1               0
## 3 0             0               0               1
## 4 0             0               0               1
## 5 0             1               1               0
## 6 0             0               0               1
## 7 0             0               0               1
## 8 0             0               0               1
## 9 0             0               0               1
## 10 0            0               0               1
## # ... with 2,990 more rows, and 13 more variables:
## #   location_hash05 - location_hash17 <dbl>
```

- 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: 🏷️.

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) %>%
  juice()

## # A tibble: 3,000 x 3
##       location   pets location2
##   <dbl> <fct> <fct>
## 1 -0.539 dogs   mountain view, california
## 2 -1.20  dogs   oakland, california
## 3 -1.84  dogs   san francisco, california
## 4 -2.40  dogs   san pablo, california
## 5 -1.20  dogs   oakland, california
## 6 -1.84  dogs   san francisco, california
## 7 -0.571  cats   san rafael, california
## 8 -1.84  dogs   san francisco, california
## 9 -1.84  dogs   san francisco, california
## 10 -0.930 cats   alameda, california
## # ... with 2,990 more rows
```

Make sure you get what just happened

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

```
## [1] -0.5389965
```

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)) %>%
  juice()
```

```
## # A tibble: 100 x 3
##       location  pets  location2
##       <dbl> <fct> <fct>
## 1      -1.50 dogs  mountain view, california
## 2      -1.72 dogs  oakland, california
## 3      -1.99 dogs  san francisco, california
## 4      -1.78 dogs  san pablo, california
## 5      -1.72 dogs  oakland, california
## 6      -1.99 dogs  san francisco, california
## 7      -1.35 cats   san rafael, 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) %>%
  juice()
```

Alternatively you could use pre-trained embeddings for text variables such as essays 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) %>%
  juice()
```

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) %>%
  juice()
```

```
## # A tibble: 3,000 x 12
##   pets   old_location     PC01     PC02     PC03     PC04     PC05     PC06
##   <fct> <fct>       <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 dogs   mountain    vi~ -1.27    -0.634    -0.985    0.433   -9.37e-2  0.538
## 2 dogs   oakland, ca~ -1.58     2.66     0.711   -0.0657   9.48e-3 -0.0482
## 3 dogs   san francis~ 1.15     0.0528    0.0410  -0.00550  8.27e-4 -0.00425
## 4 dogs   san pablo, ~ -1.26    -0.617    -0.937    0.372   -7.59e-2  0.427
## 5 dogs   oakland, ca~ -1.58     2.66     0.711   -0.0657   9.48e-3 -0.0482
## 6 dogs   san francis~ 1.15     0.0528    0.0410  -0.00550  8.27e-4 -0.00425
## 7 cats   san rafael, ~ -1.28    -0.678    -1.13     0.720   -2.19e-1  1.50
## 8 dogs   san francis~ 1.15     0.0528    0.0410  -0.00550  8.27e-4 -0.00425
## 9 dogs   san francis~ 1.15     0.0528    0.0410  -0.00550  8.27e-4 -0.00425
## 10 cats  alameda, ca~ -0.447   -0.0513   -0.0430   0.00613 -9.29e-4  0.00478
## # ... with 2,990 more rows, and 4 more variables: PC07 <dbl>, PC08 <dbl>
## #   PC09 <dbl>, PC10 <dbl>
```

Text Features

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

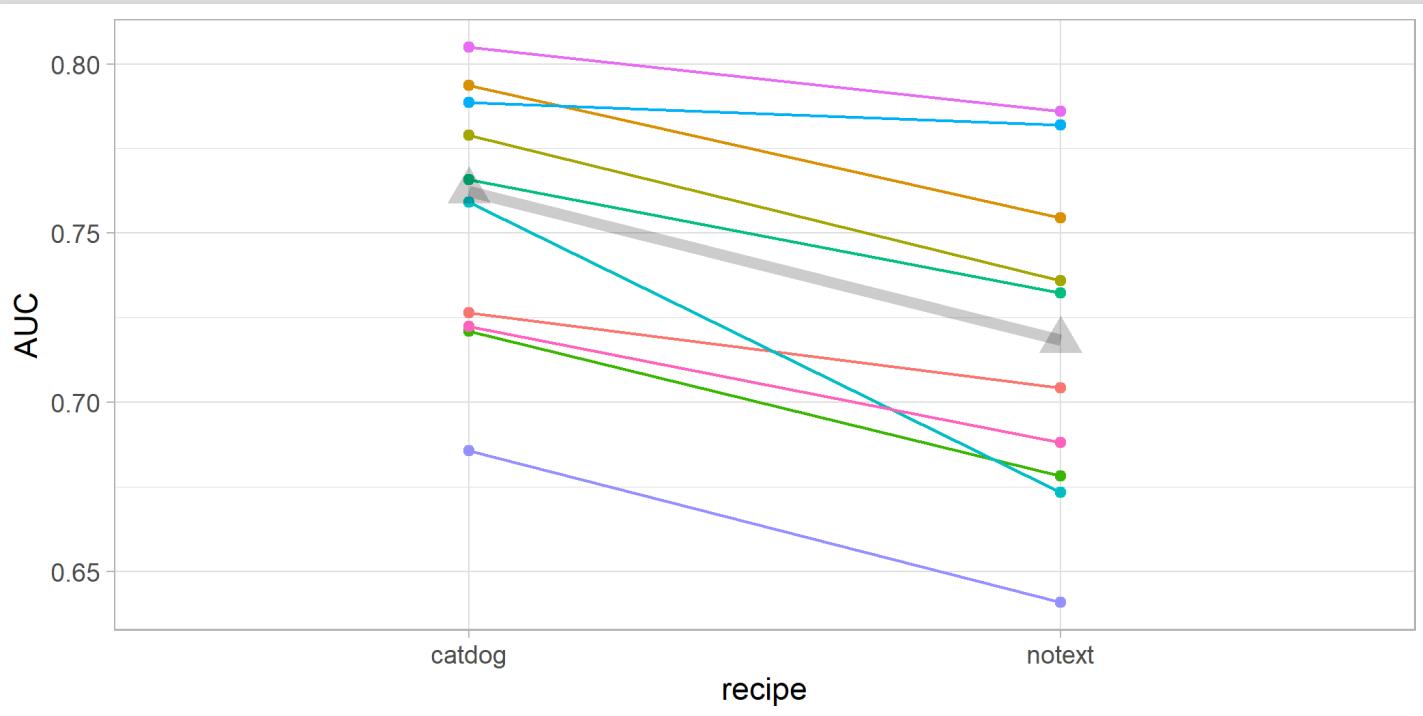
Just to demonstrate how worthwhile being creative can get here let us just add two textual features here, `n_cat` and `n_dog` in all essays:

```
ok_tr_eng2 <- ok_tr_eng %>%
  mutate_at(vars(essay0:essay9),
            list("len" = ~ifelse(is.na(.x), 0, str_length(.x)))) %
  rename_at(vars(contains("_len")),
            list(~paste("len", gsub("_len", "", .), sep = "_") ) )
```

```
rec_eng_notext <- recipe(pets ~ ., data = ok_tr_eng2) %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_meanimpute(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_dummy(all_nominal(), -all_outcomes(),
            -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1)

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

```
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")) %>%
  update_role(starts_with("essay"), new_role = "discarded") %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_meanimpute(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_dummy(all_nominal(), -all_outcomes(),
            -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1)
```



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](#) 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  [11:52:54.899] early stopping at 20 iteration  
## INFO  [11:52:54.956] early stopping at 20 iteration  
##    w1    w2    w3    w4    w5    w6    w7    w8    w9    w10  
## 1    0    0.05   0.3    0    0    0.5   0.05   0    0.1    0
```

```
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)) %>%
  juice() %>%
  select(t_essays_n_uq_words, t_essays_n_hashtags, t_essays_sent_k)

```

```

## # A tibble: 10 x 3
##   t_essays_n_uq_words t_essays_n_hashtags t_essays_sent_bing
##             <dbl>                  <dbl>                  <dbl>
## 1                 114                      0                     18
## 2                   0                      0                     0
## 3                 116                      0                     11
## 4                 250                      0                     11
## 5                 356                      0                     8
## 6                 405                      0                    40
## 7                   0                      0                     0
## 8                 622                      0                    36
## 9                   0                      0                     0
## 10                465                      0                     23

```

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^{-5}

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

```
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 x 3
##       id   pets   word
##   <int> <chr> <chr>
## 1     1  dogs  positive
## 2     1  dogs    person
## 3     1  dogs     loves
## 4     1  dogs     laugh
## 5     1  dogs     enjoy
## 6     1  dogs      life
```

```
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
```

```
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: 808 x 7
##   word      cats  dogs fisher_obj          OR  p_val     fdr
##   <chr>     <int> <int> <list>        <dbl> <dbl> <dbl>
## 1 ability      17    46 <named list [2]>  1.91  0.0369 0.112
## 2 absolutely   18    62 <named list [2]>  1.49  0.167  0.317
## 3 accent       14    38 <named list [2]>  1.90  0.0561 0.148
## 4 action        23    96 <named list [2]>  1.23  0.378  0.559
## 5 active        21   134 <named list [2]>  0.788 0.373  0.557
## 6 activities    16    91 <named list [2]>  0.891 0.791  0.892
## 7 add           17    46 <named list [2]>  1.91  0.0369 0.112
## 8 admit         17    92 <named list [2]>  0.937 0.896  0.950
## 9 adult          13    38 <named list [2]>  1.76  0.0861 0.195
## 10 adventure    25   145 <named list [2]>  0.870 0.595  0.749

```

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

```
cd_words
```

```
## # A tibble: 37 x 7
##   word      cats  dogs fisher_obj          OR    p_val     fdr
##   <chr>     <int> <int> <list>        <dbl>    <dbl>    <dbl>
## 1 cat       55    37 <named list [2]>  8.37  1.06e-21 8.53e-19
## 2 cats      43    21 <named list [2]> 11.3   5.37e-20 2.17e-17
## 3 class     73   151 <named list [2]>  2.71   4.72e-10 9.54e- 8
## 4 href      59   106 <named list [2]>  3.08   4.40e-10 9.54e- 8
## 5 ilink     57   101 <named list [2]>  3.11   6.79e-10 1.10e- 7
## 6 dog        12   247 <named list [2]> 0.228  2.19e- 9 2.95e- 7
## 7 writing    62   127 <named list [2]>  2.69   1.08e- 8 1.25e- 6
## 8 strange    24    28 <named list [2]>  4.53   4.03e- 7 4.07e- 5
## 9 dr         24    29 <named list [2]>  4.37   6.26e- 7 5.62e- 5
## 10 dogs      5   138 <named list [2]> 0.176  1.31e- 6 1.03e- 4
## # ... with 27 more rows
```

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$word, ~str_count_na(essays, .x))
  names(counts) = str_c("w_", cd_words$word)
  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_if(is.character, as.factor)
```

```

rec_eng_notext <- recipe(pets ~ ., data = ok_tr_eng3) %>%
  update_role(starts_with("w_"), essays, new_role = "discarded") %>%
  step_log(income, offset = 1) %>%
  step_meanimpute(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other") %>%
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes) %>%
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outcomes) %>%
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1)

rec_eng_textfeat <- recipe(pets ~ ., data = ok_tr_eng3) %>%
  update_role(starts_with("w_"), essays, new_role = "discarded") %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funcs) %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)) %>%
  step_log(income, starts_with("t_"), -t_essays_sent_bing, offset = 1) %>%
  step_meanimpute(income) %>%
  step_zv(all_numeric()) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other") %>%
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes) %>%
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outcomes) %>%
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1)

```

```

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_meanimpute(income) %>%
  step_zv(all_numeric()) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_dummy(all_nominal(), -all_outcomes(),
    -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1)

rec_eng_hashtext <- recipe(pets ~ ., data = ok_tr_eng3) %>%
  update_role(starts_with("w_"), 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_meanimpute(income) %>%
  step_zv(all_numeric()) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outcomes)
  step_dummy(all_nominal(), -all_outcomes(),
    -has_role("discarded"), one_hot = FALSE) %>%

```

```

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

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

cv_res_eng <- cv_splits_eng %>%
  bind_cols(
    map2_dfc(lst_recos, names(lst_recos), train_recipe,
             splits = cv_splits_eng$splits)
  ) %>%
  pivot_longer(cols = names(lst_recos),
               names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC)

cv_res_eng_meds <- cv_res_eng %>%
  group_by(recipe) %>%
  summarise(AUC = median(AUC), id = "All")

```

Haven't forgot about our other categorical variables

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

```
ok_tr_eng4 <- full_features(ok_tr_eng)

colnames(ok_tr_eng4)
```



```
## [1] "age"                      "body_type"
## [3] "diet"                      "drinks"
## [5] "drugs"                     "education"
## [7] "ethnicity"                  "height"
## [9] "income"                     "job"
## [11] "location"                  "orientation"
## [13] "pets"                       "religion"
## [15] "sex"                        "sign"
## [17] "smokes"                     "status"
## [19] "len_essay0"                 "len_essay1"
## [21] "len_essay2"                 "len_essay3"
## [23] "len_essay4"                 "len_essay5"
## [25] "len_essay6"                 "len_essay7"
## [27] "len_essay8"                 "len_essay9"
## [29] "essays"                     "offspring_no_kids"
## [31] "offspring_one_kid"          "offspring_many_kids"
```

```
rec_eng_commonsense <- recipe(pets ~ ., data = ok_tr_eng4) %>%
  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("len_"), starts_with("t_"),
    -t_essays_sent_bing, offset = 1) %>%
  step_meanimpute(income) %>%
  step_zv(all_numeric()) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(
    body_type, diet, drugs, job, sign, religion, smokes, drinks, e
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outco
  step_dummy(all_nominal(), -all_outcomes(),
    -has_role("discarded")), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1)
```

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

lst_recbs <- list("commonsense" = rec_eng_commonsense)
```

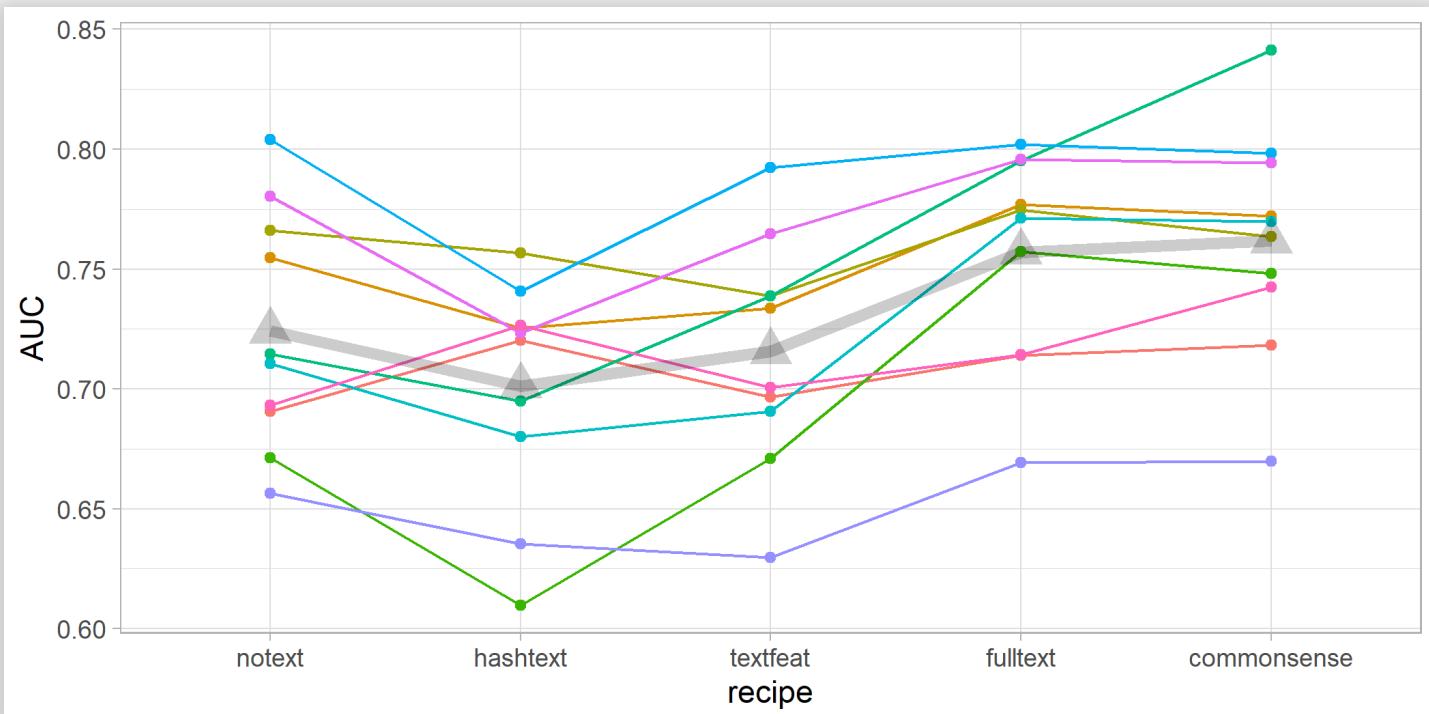
```
cv_res_commonsense <- cv_splits_eng2 %>%
  bind_cols(
    map2_dfc(lst_recbs, names(lst_recbs), train_recipe,
             splits = cv_splits_eng2$splits)
  ) %>%
  pivot_longer(cols = names(lst_recbs),
               names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC)

cv_res_commonsense_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",
                                 "fulltext", "commonse")))
  ggplot(aes(recipe, AUC, group = id, color = id)) +
  geom_line() + geom_line(data = cv_res_eng_meds %>% bind_rows(cv_res_commonsense),
                           group = 1, color = "#999999") +
  geom_point() + geom_point(data = cv_res_eng_meds %>% bind_rows(cv_res_commonsense),
                           group = 1, color = "#999999") +
  guides(color = FALSE) + theme_light()

```



Numerical Variables

APPLICATIONS



OF DATA SCIENCE

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

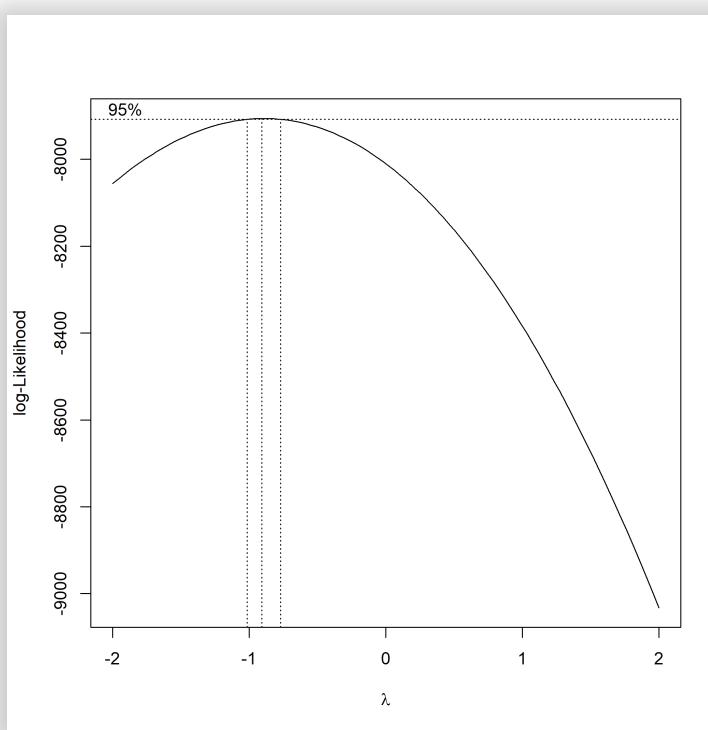
In its simplified form:

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

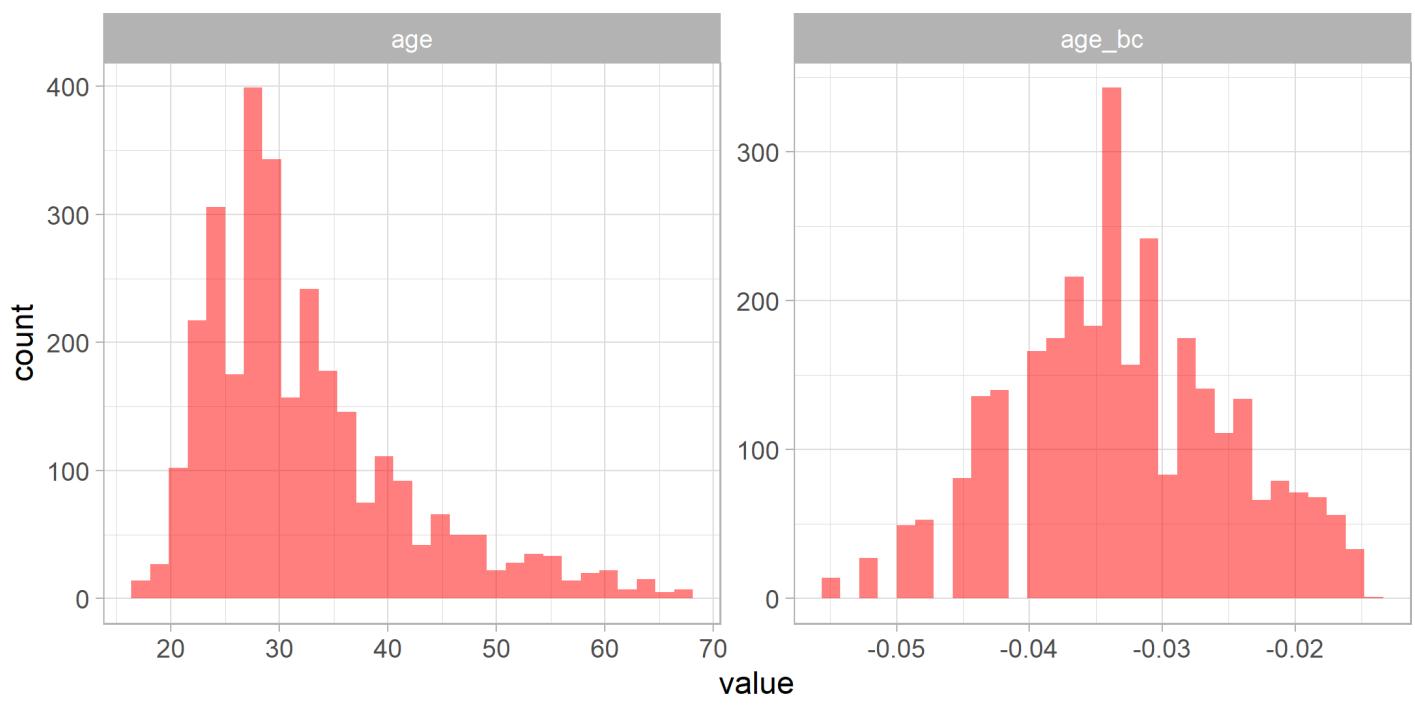
Originally the Box-Cox transformation was developed to transform a dependent y variable by maximizing the log-likelihood of seeing $y^{(\lambda)}$ in a `lm` against some X .

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:

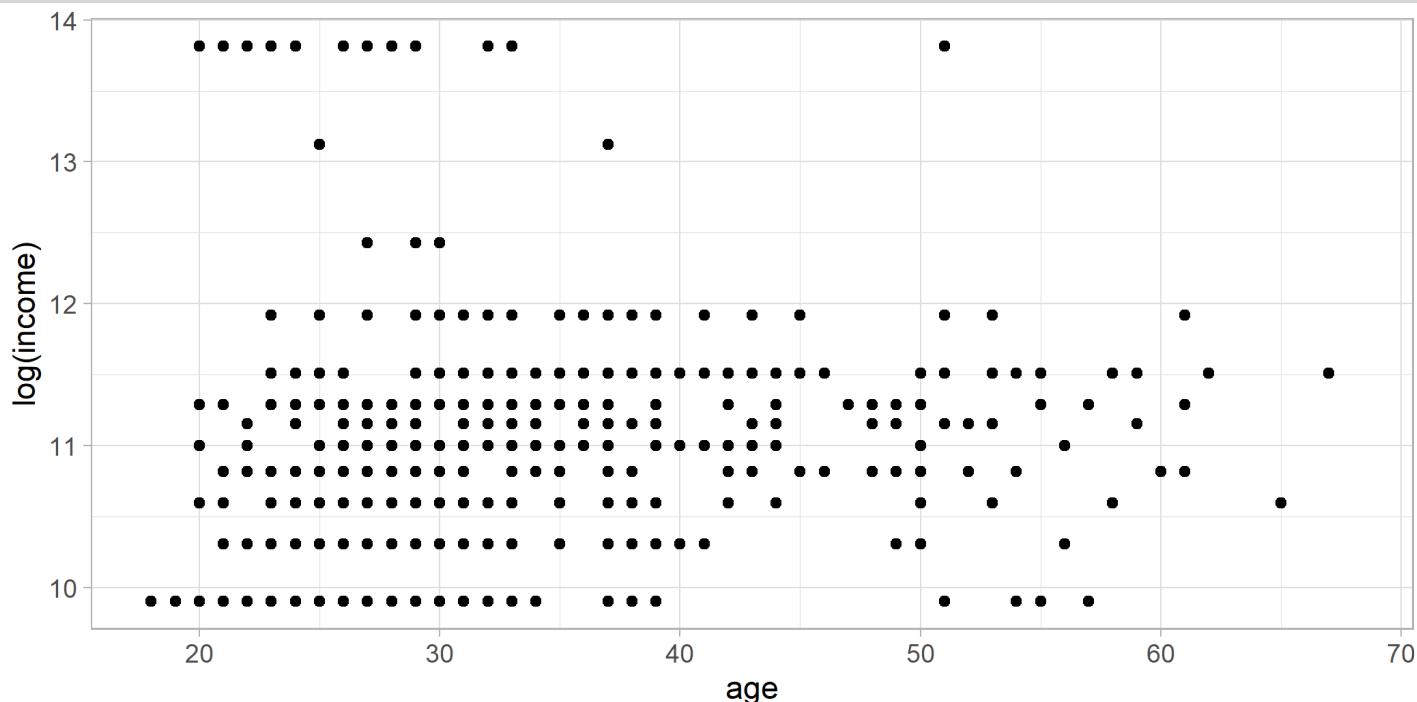
```
recipe(pets ~ ., data = ok_tr_eng) %>%
  step_BoxCox(income)
```

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

Non-Linearity: Basis Expansions and Splines

Let's borrow a continuous y variable like $\log(\text{income})$ 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  
## -1.3331 -0.6472 -0.0632  0.3654  3.2316  
##  
## Coefficients:  
##                               Estimate Std. Error t value Pr(>|t|)  
## (Intercept)           11.414705   0.093229 122.437 < 2e-16 ***  
## I(pmax(0, 45 - age)) -0.033230   0.005738  -5.791 1.16e-08 ***  
## I(pmax(0, age - 45)) -0.029685   0.013992  -2.122  0.0343 *  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## Residual standard error: 0.856 on 562 degrees of freedom  
## (2435 observations deleted due to missingness)  
## Multiple R-squared:  0.05805,    Adjusted R-squared:  0.05469  
## F-statistic: 17.32 on 2 and 562 DF,  p-value: 5.038e-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
## -1.3331 -0.6472 -0.0632  0.3654  3.2316
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                10.51749   0.07891 133.280 < 2e-16 ***
## bs(age, degree = 1, knots = 45)1  0.89722   0.15492   5.791 1.16e-08 ***
## bs(age, degree = 1, knots = 45)2  0.21447   0.27510   0.780   0.436
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.856 on 562 degrees of freedom
## (2435 observations deleted due to missingness)
## Multiple R-squared:  0.05805,    Adjusted R-squared:  0.05469
## F-statistic: 17.32 on 2 and 562 DF,  p-value: 5.038e-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))
## 1            39           1                   6          0
## 4            29           1                  16          0
## 12           32           1                  13          0
## 15           22           1                  23          0
```

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

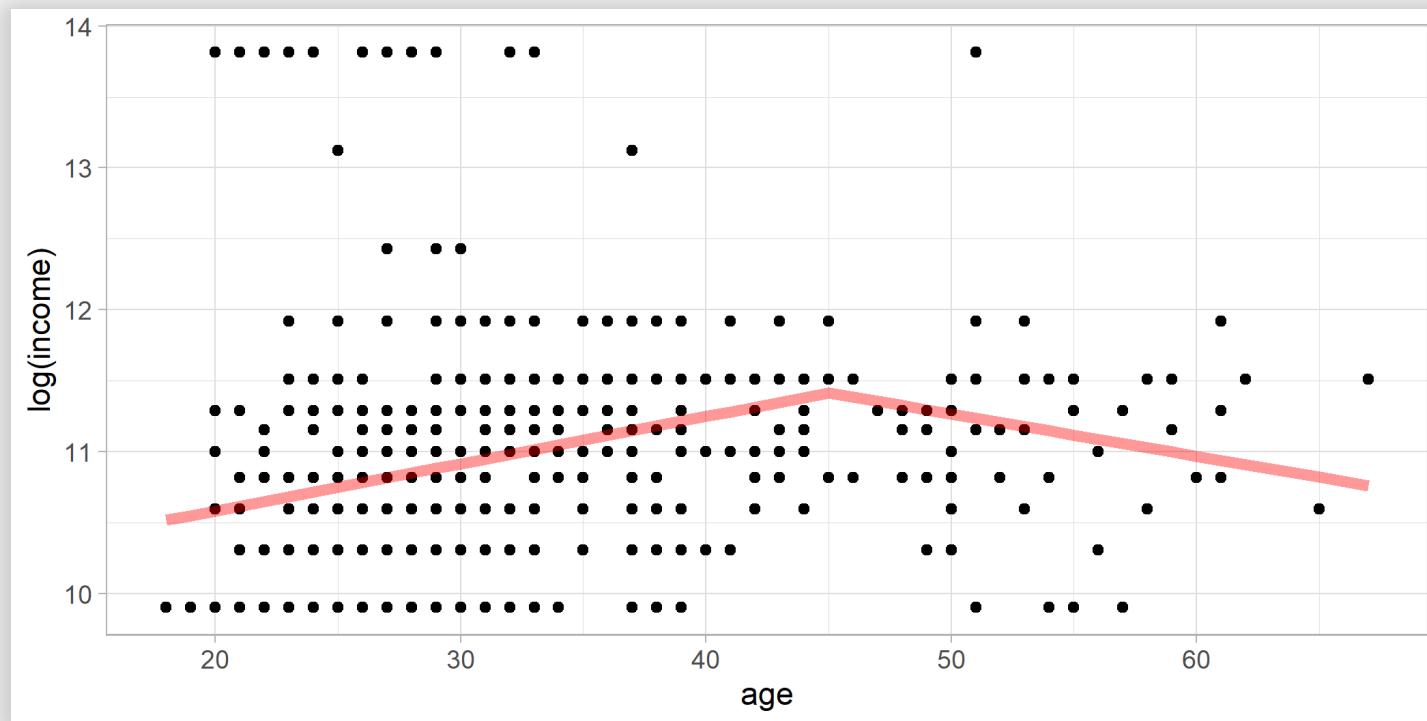
head(cbind(age_no_na, X), 4)
```

```
##      age_no_na (Intercept)      age1  age2
## 1            39           1 0.7777778  0
## 4            29           1 0.4074074  0
## 12           32           1 0.5185185  0
## 15           22           1 0.1481481  0
```

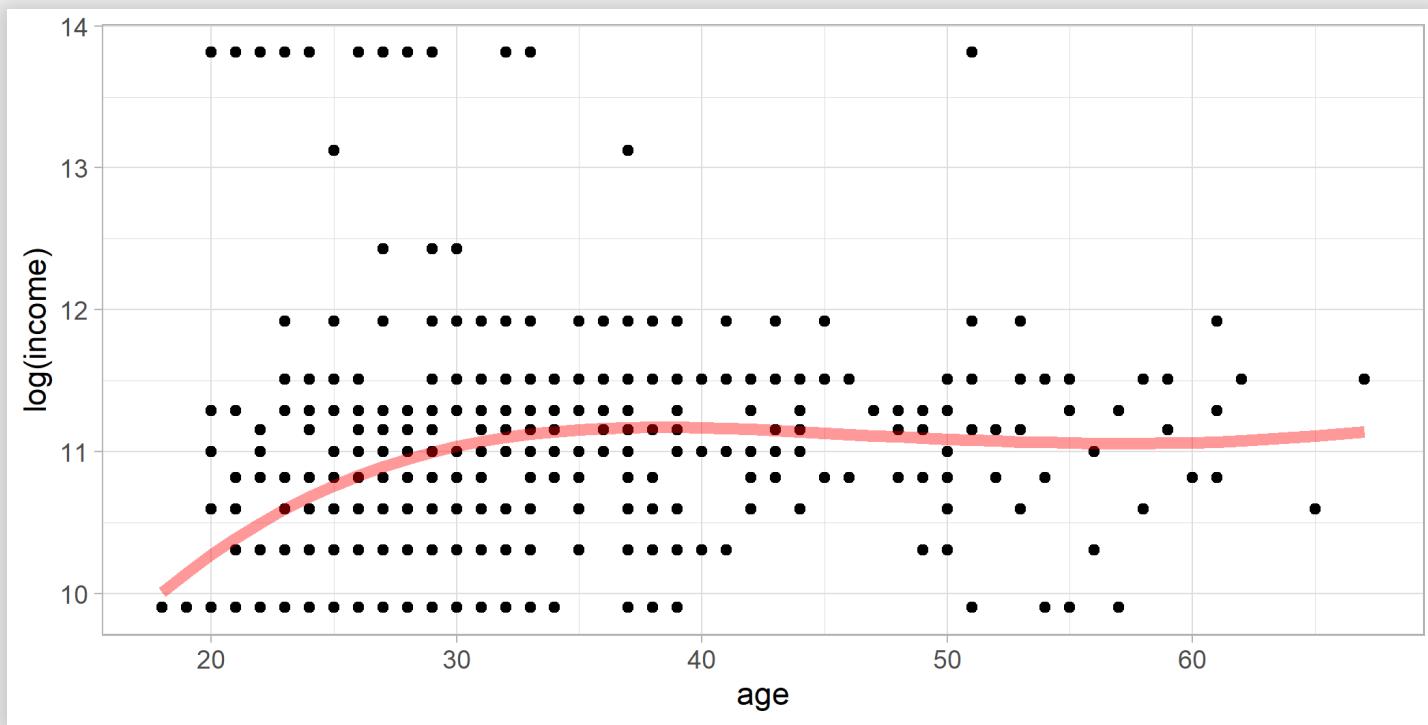
splines::bs() re-parametrizes the variables but it is the same model!

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", lwd = 2, alpha = 0.5)  
  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() %>%
  juice() %>%
  head()
```

```
## # A tibble: 6 x 4
##       age   income left_relu_age right_relu_age
##     <dbl>    <dbl>        <dbl>        <dbl>
## 1     39    80000          6          0
## 2     29    30000         16          0
## 3     32  1000000         13          0
## 4     22    20000         23          0
## 5     51  1000000          0          6
## 6     40    30000          5          0
```

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() %>%
  juice() %>%
  head()
```

```
## # A tibble: 6 x 4
##   income    age_bs_1    age_bs_2    age_bs_3
##   <dbl>     <dbl>     <dbl>     <dbl>
## 1 80000     0.420     0.315     0.0787
## 2 30000     0.405     0.117     0.0113
## 3 1000000   0.437     0.175     0.0233
## 4 20000     0.207     0.0184    0.000544
## 5 1000000   0.215     0.444     0.305
## 6 30000     0.409     0.333     0.0905
```

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

APPLICATIONS



OF DATA SCIENCE

What is Interaction?

That's a nice shirt.



×

Those are nice pants.



=

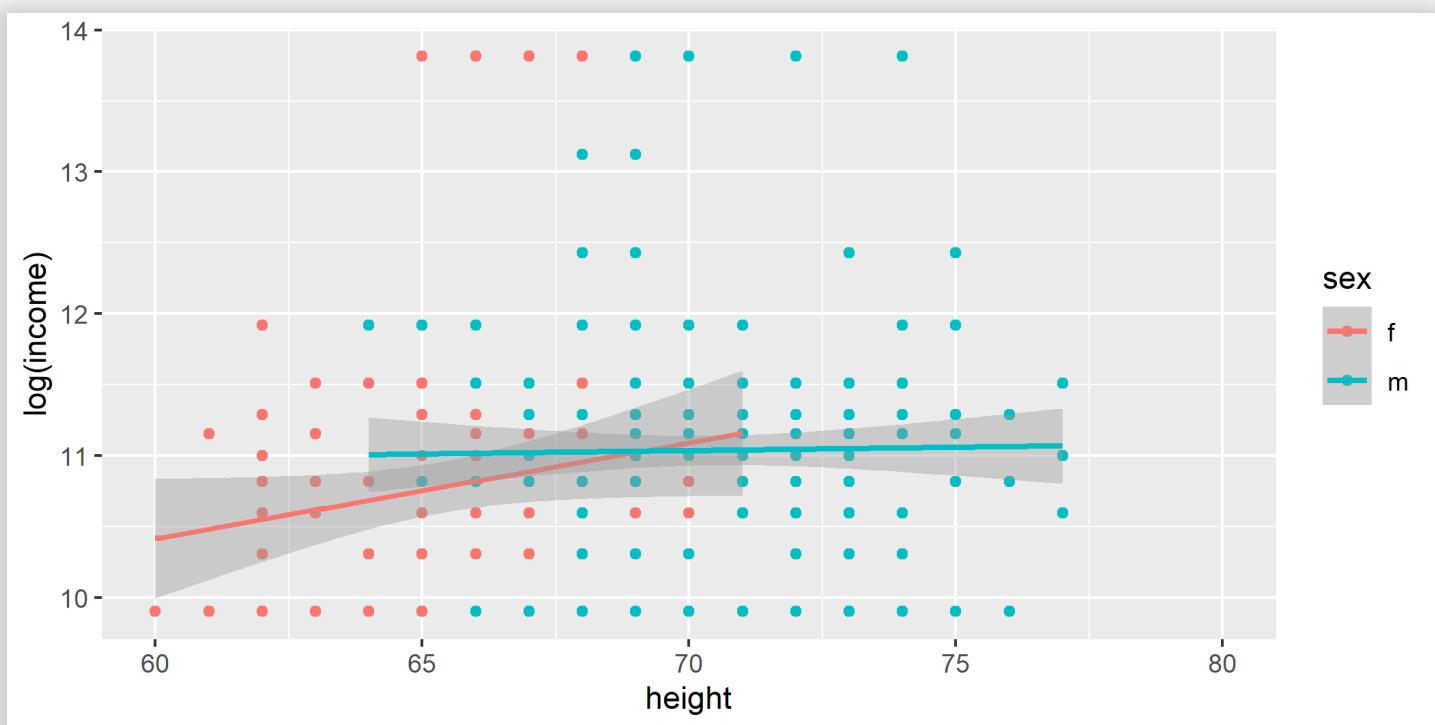
You look terrible.



But seriously

```
lm_int <- lm(log(income) ~ sex + height + sex:height,  
             data = ok_tr_interaction %>% filter(between(height, 60, 80))  
summary(lm_int)  
  
##  
## Call:  
## lm(formula = log(income) ~ sex + height + sex:height, data = ok_tr_interaction  
##       filter(between(height, 60, 80)))  
##  
## Residuals:  
##      Min        1Q    Median        3Q       Max  
## -1.16063 -0.71711 -0.02872  0.46308  3.06165  
##  
## Coefficients:  
##                 Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 6.36464   2.23738   2.845  0.00468 **  
## sexm        4.33800   2.61968   1.656  0.09855 .  
## height      0.06753   0.03421   1.974  0.04912 *  
## sexm:height -0.06277   0.03929  -1.597  0.11098  
## ---  
## Signif. codes:  0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1  
##  
## Residual standard error: 0.9044 on 386 degrees of freedom  
##   (1593 observations deleted due to missingness)  
##
```

```
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 150 features.

So all two-way interactions would mean adding... over 11K features.
All three-way interactions would mean adding... over 550K 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

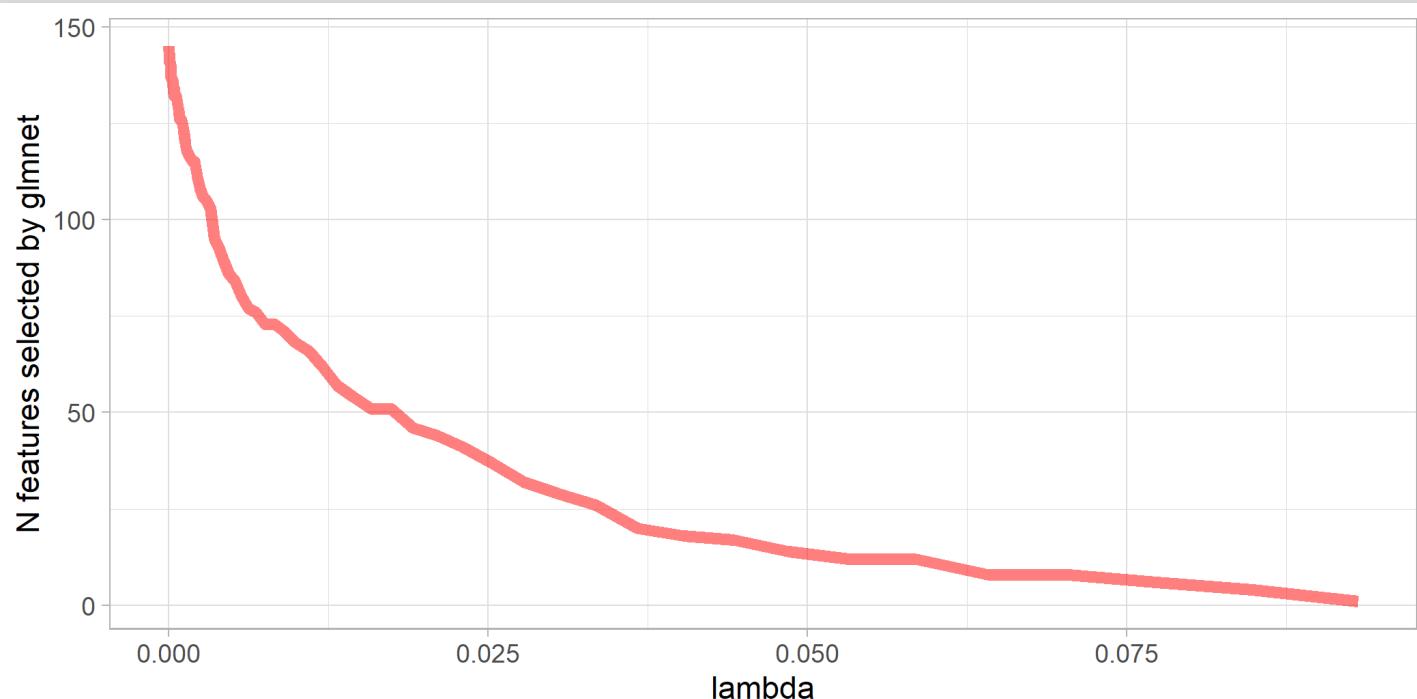
```
main_data <- rec_eng_commonsense %>% prep(ok_tr_combined) %>% juice(mod_lr_res <- mod_lr %>% fit(pets ~ ., data = main_data) %>% pluck("fit") %>% tidy()) mod_lr_res
```



```
## # A tibble: 7,962 x 5
##   term                  step  estimate lambda dev.ratio
##   <chr>                 <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept)           1 -2.45e-13  0.0931  2.60e-13
## 2 (Intercept)           2 5.09e- 2  0.0848  5.99e- 3
## 3 w_cat                 2 -3.54e- 3  0.0848  5.99e- 3
## 4 religion_christian   2 3.20e- 2  0.0848  5.99e- 3
## 5 part_white_TRUE.     2 -7.29e- 2  0.0848  5.99e- 3
## 6 (Intercept)           3 8.87e- 2  0.0773  1.55e- 2
## 7 w_cat                 3 -2.19e- 2  0.0773  1.55e- 2
## 8 w_cats                3 -7.26e- 3  0.0773  1.55e- 2
## 9 w_dog                 3 1.07e- 3  0.0773  1.55e- 2
## 10 religion_christian  3 9.52e- 2  0.0773  1.55e- 2
## # ... with 7,952 more rows
```

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

```
mod_lr_res %>%
  count(lambda) %>%
  ggplot(aes(lambda, n)) +
  geom_line(lwd = 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 == 8) %>%
  select(term, estimate) %>%
  arrange(-abs(estimate))
```

```
## # A tibble: 14 x 2
##   term                  estimate
##   <chr>                 <dbl>
## 1 part_white_TRUE.     -0.342
## 2 religion_christian    0.323
## 3 status_other1        -0.208
## 4 drinks_socially      0.183
## 5 w_dog                  0.179
## 6 drugs_sometimes      -0.122
## 7 offspring_no_kids_TRUE. 0.119
## 8 location_sf            0.104
## 9 w_cat                  -0.0842
## 10 w_cats                 -0.0577
## 11 job_marketing          0.0426
## 12 w_art                  -0.0191
## 13 t_essays_n_commas     -0.00513
## 14 (Intercept)           0.000973
```

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 == 8, 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.

```

extract_interaction_p_val <- function(var1, var2) {
  pb$tick() $print()
  int_term <- str_c(var1, ":", var2)
  mod_formula <- formula(str_c("pets ~ . + ", int_term))
  mod_glm <- glm(mod_formula,
                  data = main_data %>% select(-ends_with("_new")),
                  family = "binomial")
  mod_summary <- summary(mod_glm)
  p_val <- tidy(mod_glm) %>%
    filter(term == int_term) %>%
    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"))

```

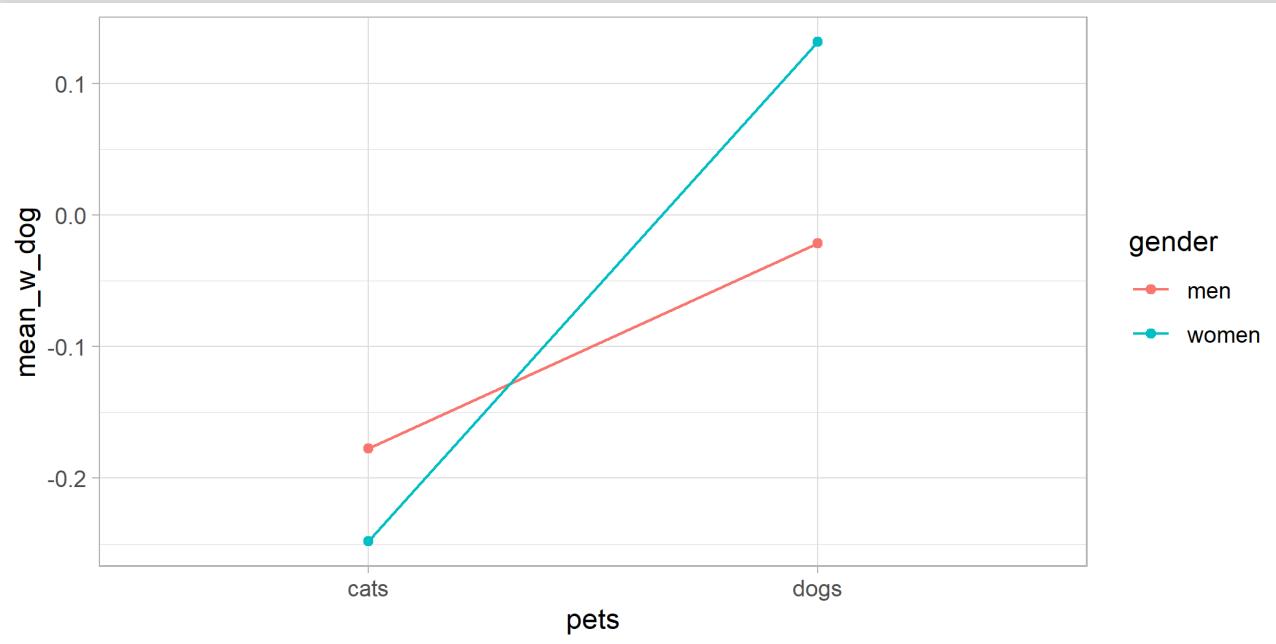
Let's filter only the *most* convincing interactions.

```
pairwise_ints <- all_ints %>%
  arrange(fdr) %>%
  filter(fdr < 10e-6)
```

```
pairwise_ints
```

```
## # A tibble: 19 x 4
##   var1                 var2           p_val      fdr
##   <chr>                <chr>        <dbl>     <dbl>
## 1 w_dog                 w_strange    5.70e-13  0.00000000296
## 2 w_strange              orientation_other1 3.79e-13  0.00000000296
## 3 w_dog                 sign_not_matter_TRUE. 2.58e-12  0.00000000892
## 4 len_essay0              job_student  1.03e-11  0.00000000215
## 5 len_essay4              job_student  8.78e-12  0.00000000215
## 6 orientation_straight religion_christian 2.68e-10  0.000000463
## 7 job_financial          location_other_location 3.72e-10  0.000000551
## 8 drinks_socially         education_kind_other 4.44e-10  0.000000576
## 9 w_class                  w_dog        8.95e-10  0.00000103
## 10 len_essay1             part_asian_TRUE. 1.57e- 9  0.00000163
## 11 w_href                  w_dog        2.78e- 9  0.00000215
## 12 w_ilink                 w_dog        2.90e- 9  0.00000215
## 13 w_dog                  sex_m        2.71e- 9  0.00000215
## 14 w_strange               diet_other1  2.44e- 9  0.00000215
## 15 part_asian_TRUE.       sign_not_matter_TRUE. 3.58e- 9  0.00000248
## 16 w_books                 speaks_spanish_TRUE. 6.22e- 9  0.00000403
## 17 len_essay6              smokes_when.drinking 7.99e- 9  0.00000488
```

```
main_data %>%
  group_by(sex_m, pets) %>%
  summarise(mean_w_dog = mean(w_dog)) %>% ungroup() %>%
  mutate(gender = factor(ifelse(sex_m == 0, "women", "men"))) %>%
  ggplot(aes(pets, mean_w_dog, color = gender)) +
  geom_line(aes(group = gender)) + geom_point() +
  theme_light()
```



💡 Try stating what you see in words.

```

ok_tr_interaction2 <- full_features(ok_tr_interaction)

rec_int_noint <- rec_eng_commonsense

rec_int_sex <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  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("len_"), starts_with("t_"),
    -t_essays_sent_bing, offset = 1) %>%
  step_meanimpute(income) %>%
  step_zv(all_numeric()) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(
    body_type, diet, drugs, job, sign, religion, smokes, drinks, e
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outco
  step_dummy(all_nominal(), -all_outcomes(),
    -has_role("discarded")), one_hot = FALSE) %>%
  step_interact(~starts_with("sex"):all_numeric()) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_nzv(all_numeric(), freq_cut = 99/1) %>%
  step_upsample(pets, over_ratio = 1)

```

```

# 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

```

```

## ~w_cat:w_cats + w_cat:w_dog + w_cat:w_art + w_cat:t_essays_n_commas +
##   w_cat:drinks_socially + w_cat:drugs_sometimes + w_cat:job_marketing +
##   w_cat:location_sf + w_cat:religion_christian + w_cat:status_other1 +
##   w_cat:offspring_no_kids_TRUE. + w_cat:part_white_TRUE. +
##   w_cats:w_dog + w_cats:w_art + w_cats:t_essays_n_commas +
##   w_cats:drinks_socially + w_cats:drugs_sometimes + w_cats:job_marketing +
##   w_cats:location_sf + w_cats:religion_christian + w_cats:status_other1 +
##   w_cats:offspring_no_kids_TRUE. + w_cats:part_white_TRUE. +
##   w_dog:w_art + w_dog:t_essays_n_commas + w_dog:drinks_socially +
##   w_dog:drugs_sometimes + w_dog:job_marketing + w_dog:location_sf +
##   w_dog:religion_christian + w_dog:status_other1 + w_dog:offspring_no_kids

```

```
rec_int_topmain <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  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("len_"), starts_with("t_"),
           -t_essays_sent_bing, offset = 1) %>%
  step_meanimpute(income) %>%
  step_zv(all_numeric()) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(
    body_type, diet, drugs, job, sign, religion, smokes, drinks, e
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outco
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_interact(topmain_ints) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_nzv(all_numeric(), freq_cut = 99/1) %>%
  step_upsample(pets, over_ratio = 1)
```

```

rec_int_topoints <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  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("len_"), starts_with("t_"),
    -t_essays_sent_bing, offset = 1) %>%
  step_meanimpute(income) %>%
  step_zv(all_numeric()) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(
    body_type, diet, drugs, job, sign, religion, smokes, drinks, e
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outco
  step_dummy(all_nominal(), -all_outcomes(),
    -has_role("discarded")), one_hot = FALSE) %>%
  {if (!"smokes_when.drinking" %in% .$var_info$variable)
    step_mutate(., smokes_when.drinking = 0)} %>%
  step_interact(topint_ints) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_nzv(all_numeric(), freq_cut = 99/1) %>%
  step_upsample(pets, over_ratio = 1)

```

```
rec_int_all <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  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("len_"), starts_with("t_"),
    -t_essays_sent_bing, offset = 1) %>%
  step_meanimpute(income) %>%
  step_zv(all_numeric()) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other")
  step_novel(
    body_type, diet, drugs, job, sign, religion, smokes, drinks, e
  step_modeimpute(all_nominal(), -has_role("discarded"), -all_outco
  step_dummy(all_nominal(), -all_outcomes(),
    -has_role("discarded"), one_hot = FALSE) %>%
  step_interact(~all_numeric():all_numeric()) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_nzv(all_numeric(), freq_cut = 95/5) %>%
  step_upsample(pets, over_ratio = 1)
```

```
set.seed(42)
cv_splits_int <- vfold_cv(ok_tr_interaction2, v = 10, strata = pet
lst_recos <- list("noint" = rec_int_noint,
                  "sex" = rec_int_sex,
                  "topmain" = rec_int_topmain,
                  "topints" = rec_int_topints,
                  "all" = rec_int_all)
```

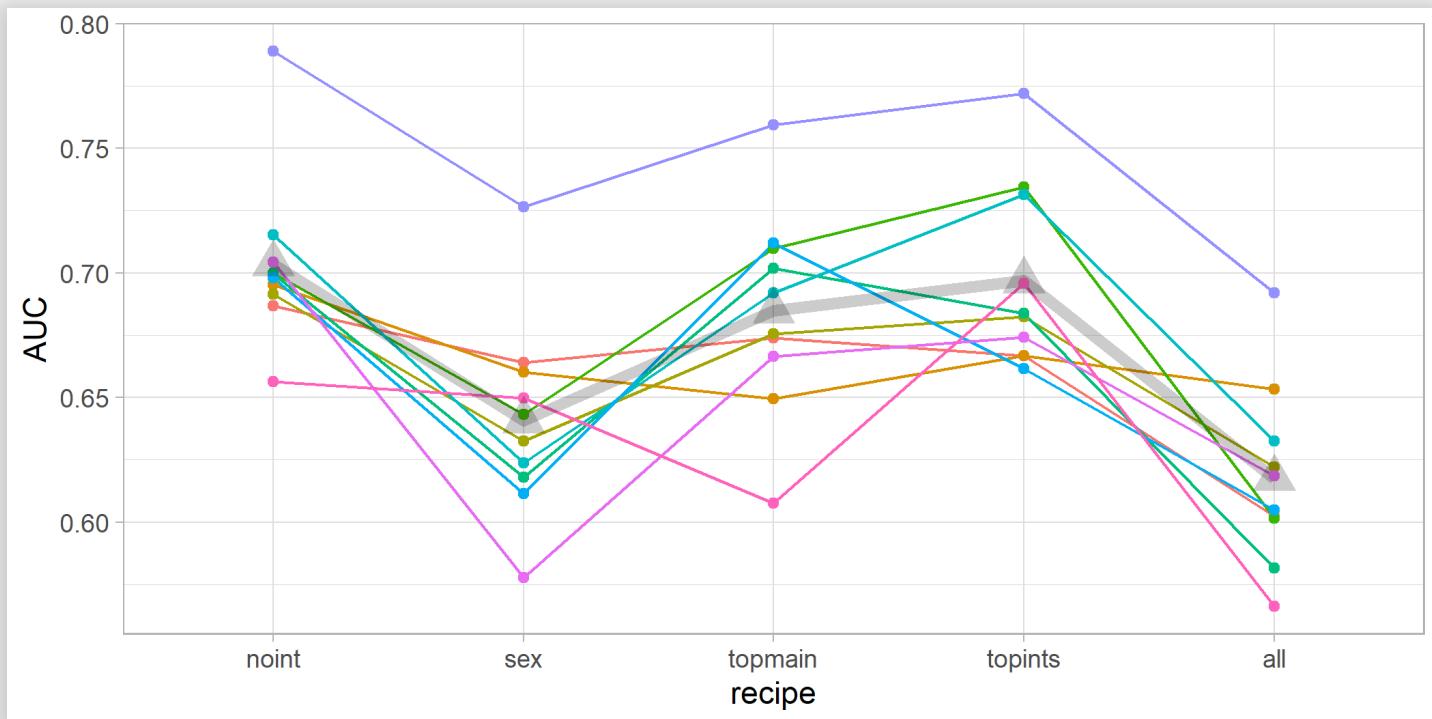
```
cv_res_int <- cv_splits_int %>%
  bind_cols(
    map2_dfc(lst_recos, names(lst_recos), train_recipe,
             splits = cv_splits_int$splits)
  ) %>%
  pivot_longer(cols = names(lst_recos),
               names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC)

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 = "black",
  guides(color = FALSE) + theme_light()

```



Tuning Params

APPLICATIONS



OF DATA SCIENCE

Everything is a 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 :

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

```
library(tune)

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))
```

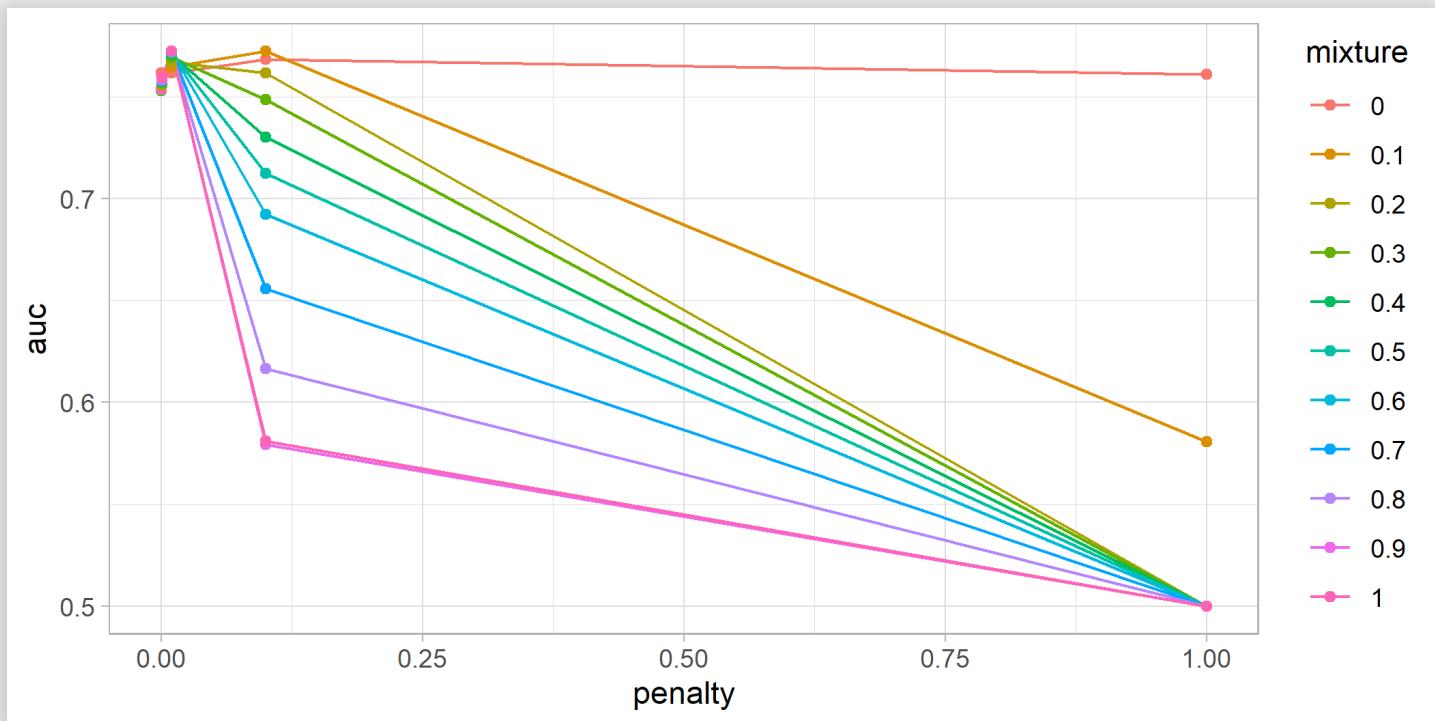
```
ok_tr_tuning2 <- full_features(ok_tr_tuning)

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_int_topints,
                        resamples = cv_splits_tune,
                        grid = en_grid,
                        metrics = metric_set(roc_auc))
tune_res
```

```
## # 10-fold cross-validation using stratification
## # A tibble: 10 × 4
##   splits              id   .metrics      .notes
##   <list>            <chr> <list>        <list>
## 1 <split [2.7K/301]> Fold01 <tibble [77 × 5]> <tibble [11 × 1]>
## 2 <split [2.7K/301]> Fold02 <tibble [77 × 5]> <tibble [11 × 1]>
## 3 <split [2.7K/301]> Fold03 <tibble [77 × 5]> <tibble [11 × 1]>
## 4 <split [2.7K/301]> Fold04 <tibble [77 × 5]> <tibble [11 × 1]>
## 5 <split [2.7K/300]> Fold05 <tibble [77 × 5]> <tibble [11 × 1]>
## 6 <split [2.7K/300]> Fold06 <tibble [77 × 5]> <tibble [11 × 1]>
## 7 <split [2.7K/299]> Fold07 <tibble [77 × 5]> <tibble [11 × 1]>
## 8 <split [2.7K/299]> Fold08 <tibble [77 × 5]> <tibble [11 × 1]>
## 9 <split [2.7K/299]> Fold09 <tibble [77 × 5]> <tibble [11 × 1]>
## 10 <split [2.7K/299]> Fold10 <tibble [77 × 5]> <tibble [11 × 1]>
```

```
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 x 7
##   penalty mixture .metric .estimator  mean     n std_err
##   <dbl>    <dbl> <chr>   <chr>     <dbl> <int>   <dbl>
## 1 0.01      1    roc_auc binary    0.772    10  0.0144
## 2 0.1       0.1   roc_auc binary    0.772    10  0.0146
## 3 0.01      0.9   roc_auc binary    0.772    10  0.0148
## 4 0.01      0.7   roc_auc binary    0.772    10  0.0151
## 5 0.01      0.8   roc_auc binary    0.772    10  0.0149
## 6 0.01      0.6   roc_auc binary    0.772    10  0.0155
## 7 0.01      0.5   roc_auc binary    0.772    10  0.0157
## 8 0.01      0.4   roc_auc binary    0.771    10  0.0161
## 9 0.01      0.3   roc_auc binary    0.769    10  0.0163
## 10 0.1      0    roc_auc binary    0.768    10  0.0153
```

 So what would you do?

Cutoff choice

APPLICATIONS



OF DATA SCIENCE

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 maintaining acceptable levels of recall)
- a score to maximize recall of `cats` (while maintaining 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_int_topints %>%
  prep(ok_tr_combined)

ok_tr_for_cutoff <- rec_cutoff %>%
  juice()

ok_te_for_cutoff <- rec_cutoff %>%
  bake(ok_tr_cutoff2)

mod_lr_cutoff <- logistic_reg(penalty = 0.01, mixture = 1) %>%
  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)

```

	0.1	0.2	0.3	0.4	0.5	0.6
## accuracy	0.18655693	0.2496571	0.4046639	0.5514403	0.7009602	0.7901235
## recall	1.00000000	0.9583333	0.9250000	0.7666667	0.5750000	0.3750000
## precision	0.16830295	0.1750381	0.2070896	0.2352941	0.2923729	0.3658537
## specificity	0.02627258	0.1100164	0.3021346	0.5090312	0.7257800	0.8719212
## npv	1.00000000	0.9305556	0.9533679	0.9171598	0.8965517	0.8762376
	0.7	0.8	0.9			
## accuracy	0.8340192	0.83950617	0.83813443			
## recall	0.2583333	0.08333333	0.03333333			
## precision	0.4920635	0.58823529	0.66666667			
## specificity	0.9474548	0.98850575	0.99671593			
## npv	0.8663664	0.84550562	0.83955740			

Predict on Test set

APPLICATIONS



OF DATA SCIENCE

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 x 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 = 1) %>%
  set_engine("glmnet")

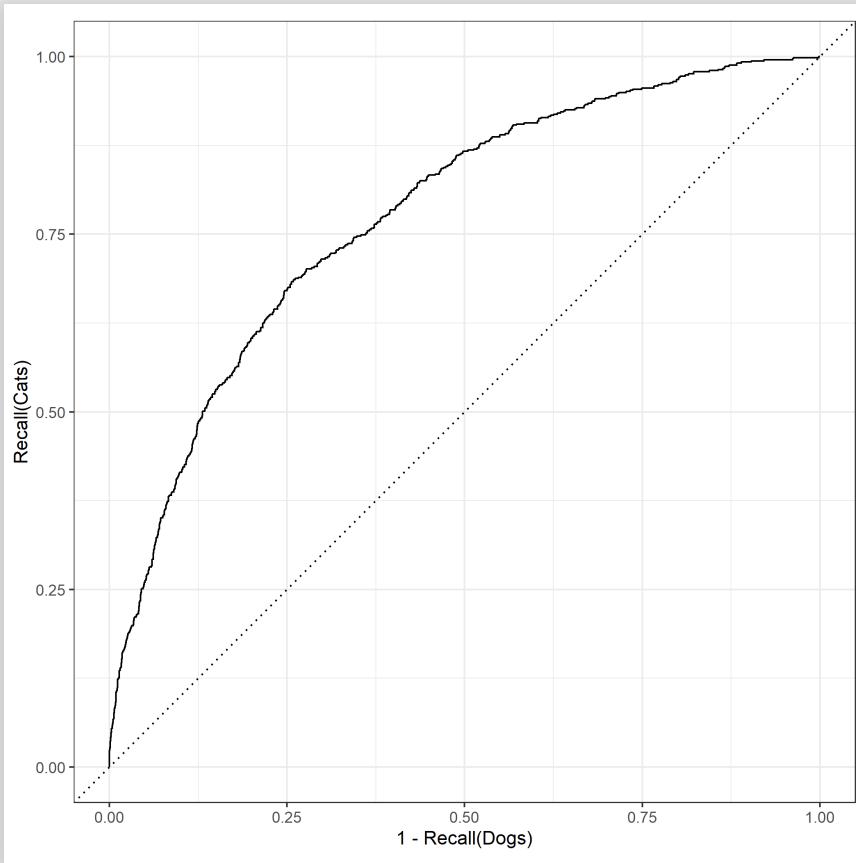
mod_lr_test <- mod_lr_test %>%
  fit(pets ~ ., data = ok_tr_for_cutoff)

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

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

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>       <dbl>
## 1 roc_auc binary     0.776
```

```
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)

```

	0.3	0.5	0.9
## accuracy	0.4042929	0.7189394	0.83888889
## recall	0.9418070	0.6967841	0.02756508
## precision	0.2094687	0.3321168	0.85714286
## specificity	0.2981554	0.7233142	0.99909283
## npv	0.9628906	0.9235521	0.83879157

What *is* the model?

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

```
## # A tibble: 20 x 3
##   term                estimate    lambda
##   <chr>              <dbl>     <dbl>
## 1 w_dog               0.643    0.00961
## 2 (Intercept)          0.329    0.00961
## 3 w_cats              -0.181   0.00961
## 4 religion_christian  0.179    0.00961
## 5 age                 -0.174   0.00961
## 6 w_cat               -0.161   0.00961
## 7 drinks_socially     0.157    0.00961
## 8 job_marketing        0.155    0.00961
## 9 location_sf           0.147   0.00961
## 10 drugs_sometimes    -0.141   0.00961
## 11 body_type_thin      -0.140   0.00961
## 12 part_white_TRUE.   -0.125   0.00961
## 13 religion_jewish      0.120   0.00961
## 14 job_hardware        -0.119   0.00961
## 15 status_other1       -0.113   0.00961
## # ... with 16 more rows, and 1 more variable:
## #   std.error <dbl>
```

What did we achieve?

- ✓ Reproducible model
- ✓ Dependable model
- ✓ Explainable model
- ✓ Small and Fast as hell model
- ✗ Impressive model

Is that the end?

No.

💡 What *didn't* we do?

Few Excellent Books

