Applications

Of Data Science

Modeling in the Tidyverse

Applications of Data Science - Class 5

Giora Simchoni

gsimchoni@gmail.com and add #dsapps in subject

Stat. and OR Department, TAU

2020-03-01

The Problem

Inconsistency, Inextensibility

```
n <- 10000
x1 <- runif(n)
x2 <- runif(n)
t <- 1 + 2 * x1 + 3 * x2
y <- rbinom(n, 1, 1 / (1 + exp(-t)))
```

```
glm(y ~ x1 + x2, family = "binomial")
```

```
glmnet(as.matrix(cbind(x1, x2)), as.factor(y), family = "binomial'
```

```
randomForest(as.factor(y) ~ x1 + x2)
```

```
gbm(y ~ x1 + x2, data = data.frame(x1 = x1, x2 = x2, y = y))
```

Compare this with `sklearn`

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
  GradientBoostingClassifier

LogisticRegression(penalty='none').fit(X, y)

LogisticRegression(penalty='l2', C=0.001).fit(X, y)

RandomForestClassifier(n_estimators=100).fit(X, y)

GradientBoostingClassifier(n_estimators=100).fit(X, y)
```

Detour: A Regression Problem

IPF-Lifts: Predicting Bench Lifting

- Dataset was published as part of the TidyTuesday intiative
- Comes from Open Powerlifting
- Wikipedia: Powerlifting is a strength sport that consists of three attempts at maximal weight on three lifts: squat, bench press, and deadlift

The raw data has over 40K rows: for each athlete, for each event, stats about athlete gender, age and weight, and the maximal weight lifted in the 3 types of Powerlifting.

We will be predicting `best3bench_kg` based on a few predictors, no missing values:

```
library(lubridate)

ipf_lifts <- read_csv("https://raw.githubusercontent.com/rfordatas

ipf_lifts <- ipf_lifts %>%
  drop_na(best3bench_kg, age) %>%
  filter(between(age, 18, 100), best3bench_kg > 0, equipment != "W
  select(sex, event, equipment, age, division, bodyweight_kg, best
  drop_na() %>%
  mutate(year = year(date), month = month(date),
         dayofweek = wday(date)) %>%
  select(-date) %>%
  mutate_if(is.character, as.factor)

dim(ipf_lifts)
```

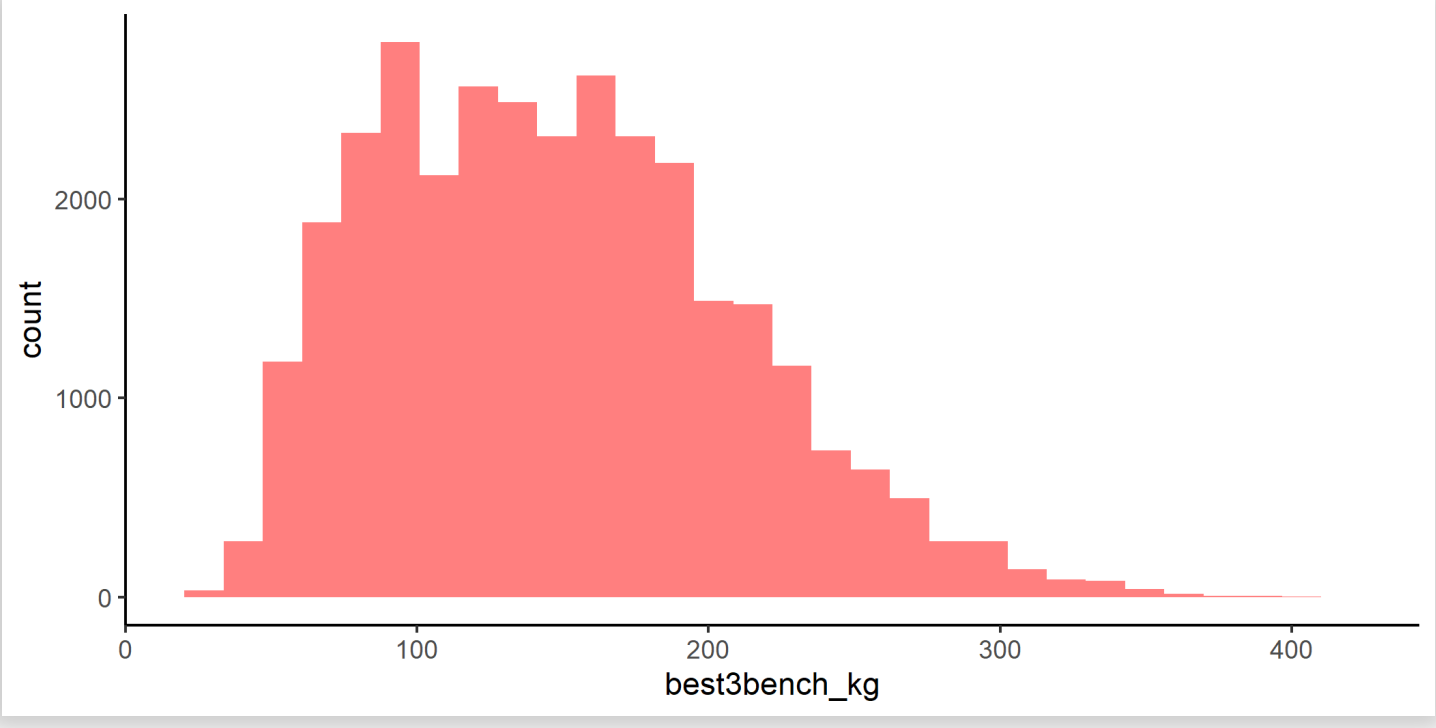
```
## [1] 32047    11
```

```
glimpse(ipf_lifts)
```

```
## Observations: 32,047
## Variables: 11
## $ sex            <fct> F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, ..
## $ event          <fct> SBD, SBD, SBD, SBD, SBD, SBD, SBD, SBD, SBD, SBD..
## $ equipment      <fct> Single-ply, Single-ply, Single-ply, Single-ply, ..
## $ age            <dbl> 33.5, 34.5, 23.5, 27.5, 37.5, 25.5, 33.5, 26.0, ..
## $ division       <fct> Open, Open, Open, Open, Open, Open, Open, Open, ..
## $ bodyweight_kg  <dbl> 44, 44, 44, 44, 44, 44, 48, 48, 48, 48, 48, 48, ..
## $ best3bench_kg  <dbl> 60.0, 62.5, 62.5, 60.0, 65.0, 45.0, 62.5, 77.5, ..
## $ meet_name      <fct> World Powerlifting Championships, World Powerlif..
## $ year           <dbl> 1989, 1989, 1989, 1989, 1989, 1989, 1989, 1989, ..
## $ month          <dbl> 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, ..
## $ dayofweek      <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ..
```

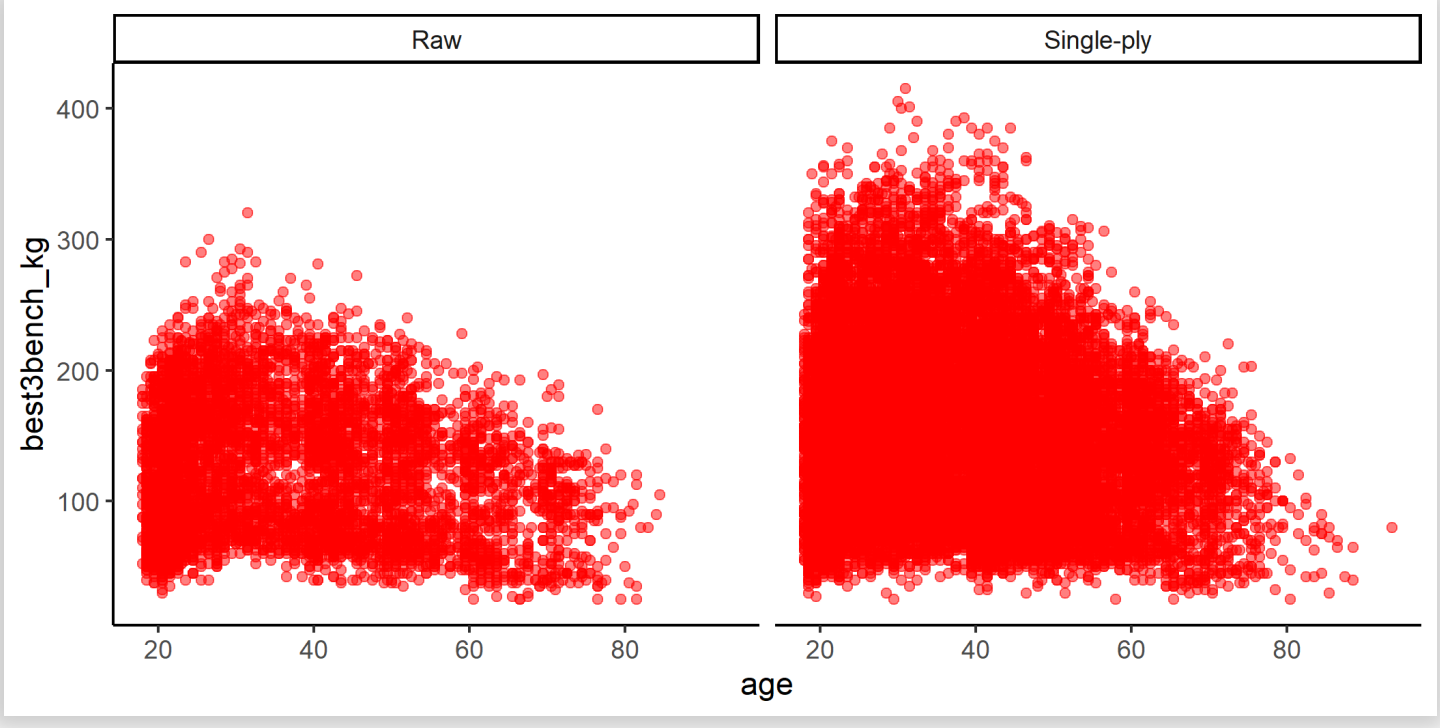
See the dependent variable distribution:

```
ggplot(ipf_lifts, aes(best3bench_kg)) +
  geom_histogram(fill = "red", alpha = 0.5) +
  theme_classic()
```



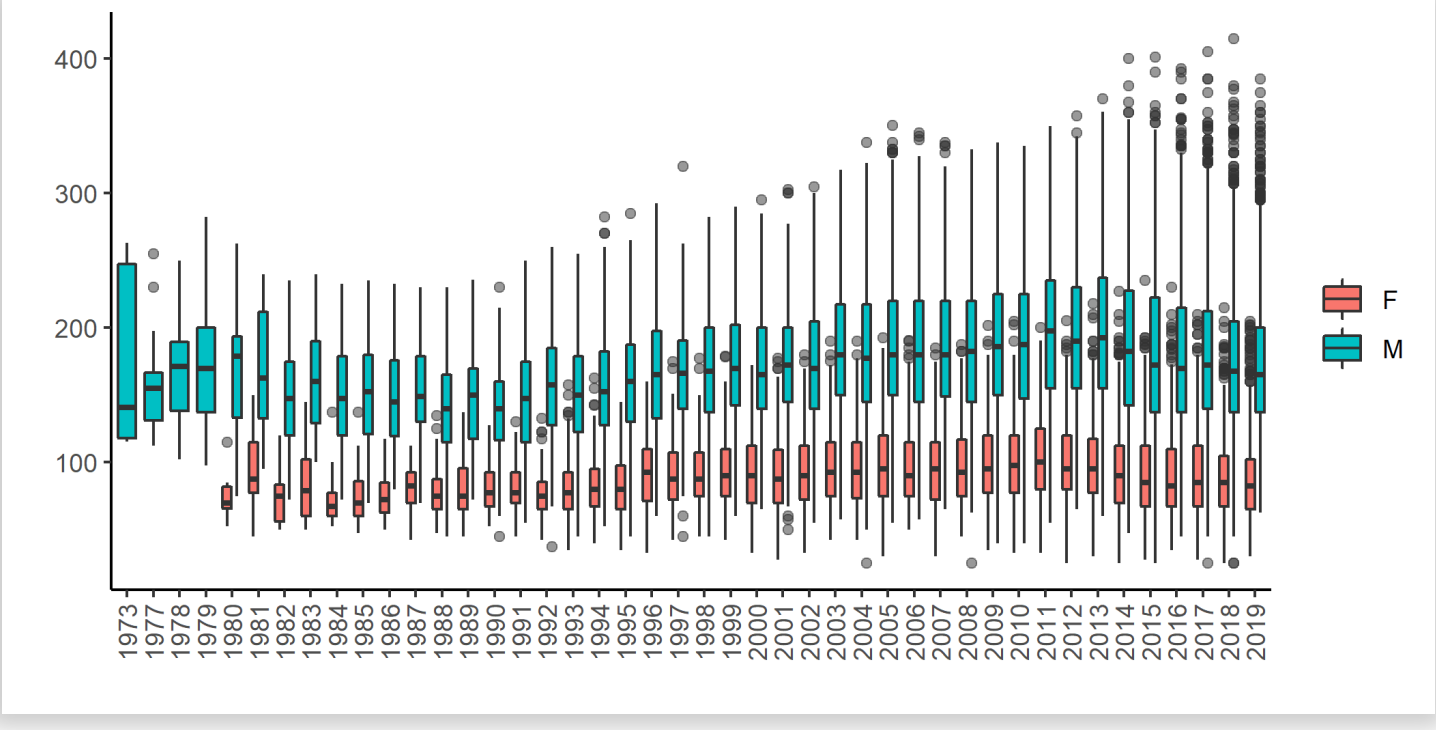
See it vs. say age, facetted by equipment:

```
ggplot(ipf_lifts, aes(age, best3bench_kg)) +
  geom_point(color = "red", alpha = 0.5) +
  facet_wrap(~ equipment) +
  theme_classic()
```



See it vs. year, by gender:

```
ggplot(ipf_lifts, aes(factor(year), best3bench_kg, fill = sex)) +
  geom_boxplot(outlier.alpha = 0.5) +
  labs(fill = "", x = "", y = "") +
  theme_classic() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust =
```



Maybe add age^2 and $year^2$ to make Linear Regression's life easier?

```
ipf_lifts <- ipf_lifts %>%
  mutate(age2 = age ^ 2, year2 = year ^2)
```

End of Detour

The Present Solution: `caret`

Split Data

```
library(caret)

train_idx <- createDataPartition(ipf_lifts$best3bench_kg,
                                 p = 0.6, list = FALSE)

ipf_tr <- ipf_lifts[train_idx, ]
ipf_te <- ipf_lifts[-train_idx, ]

library(glue)
glue("train no. of rows: {nrow(ipf_tr)}
     test no. of rows: {nrow(ipf_te)}")
```

```
## train no. of rows: 19230
## test no. of rows: 12817
```

Here you might consider some preprocessing.

`caret` has some nice documentation here.

Tuning and Modeling

Define general methodology, e.g. 10-fold Cross-Validation:

```
fit_control <- trainControl(method = "cv", number = 5)

ridge_grid <- expand.grid(alpha=0, lambda = 10^seq(-3, 1, length =
lasso_grid <- expand.grid(alpha=1, lambda = 10^seq(-3, 1, length =
rf_grid <- expand.grid(splitrule = "variance",
                       min.node.size = seq(10, 30, 10),
                       mtry = seq(2, 10, 2))

mod_ridge <- train(best3bench_kg ~ ., data = ipf_tr, method = "glm
                   trControl = fit_control, tuneGrid = ridge_grid,
                   metric = "RMSE")

mod_lasso <- train(best3bench_kg ~ ., data = ipf_tr, method = "glm
                   trControl = fit_control, tuneGrid = lasso_grid,
                   metric = "RMSE")

mod_rf <- train(best3bench_kg ~ ., data = ipf_tr, method = "ranger
                   trControl = fit_control, tuneGrid = rf_grid,
                   num.trees = 50, metric = "RMSE")
```

Evaluating Models

```
mod_ridge
```

```
## glmnet
##
## 19230 samples
##    12 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 15385, 15384, 15384, 15384, 15383
## Resampling results across tuning parameters:
##
##   lambda        RMSE      Rsquared   MAE
##    0.001000000  26.84827  0.8099138  20.55394
##    0.001206793  26.84827  0.8099138  20.55394
##    0.001456348  26.84827  0.8099138  20.55394
##    0.001757511  26.84827  0.8099138  20.55394
##    0.002120951  26.84827  0.8099138  20.55394
##    0.002559548  26.84827  0.8099138  20.55394
##    0.003088844  26.84827  0.8099138  20.55394
##    0.003727594  26.84827  0.8099138  20.55394
##    0.004498433  26.84827  0.8099138  20.55394
##    0.005428675  26.84827  0.8099138  20.55394
##    0.006551286  26.84827  0.8099138  20.55394
##    0.007906043  26.84827  0.8099138  20.55394
```

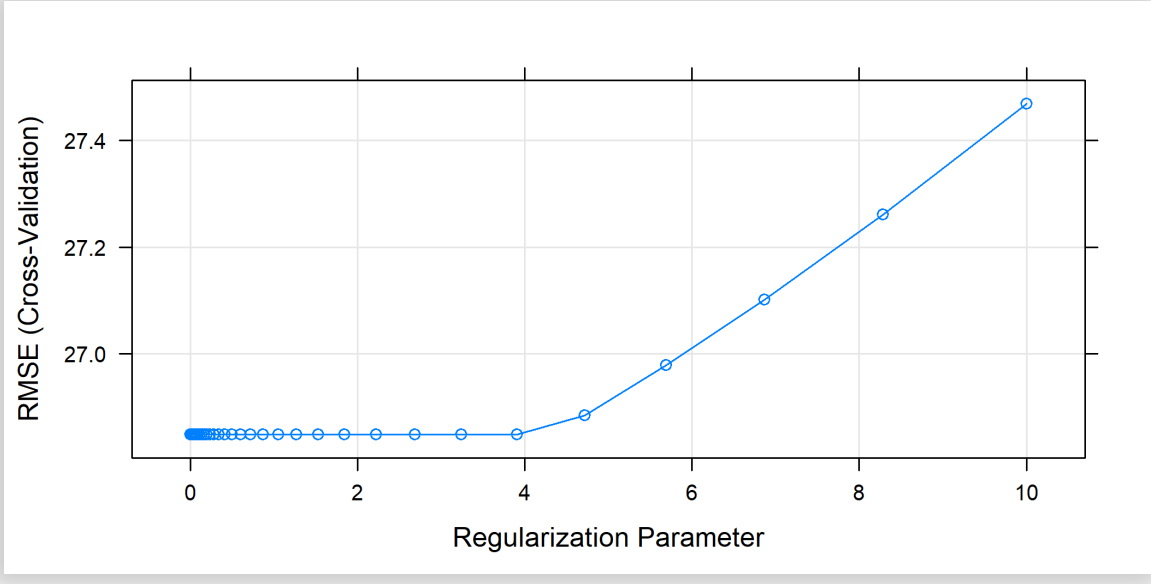
```
mod_lasso
```

```
## glmnet
##
## 19230 samples
##    12 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 15386, 15383, 15384, 15384, 15383
## Resampling results across tuning parameters:
##
##   lambda        RMSE      Rsquared   MAE
##    0.001000000  26.43438  0.8137774  20.25171
##    0.001206793  26.43438  0.8137774  20.25171
##    0.001456348  26.43438  0.8137774  20.25171
##    0.001757511  26.43438  0.8137774  20.25171
##    0.002120951  26.43438  0.8137774  20.25171
##    0.002559548  26.43438  0.8137774  20.25171
##    0.003088844  26.43438  0.8137774  20.25171
##    0.003727594  26.43438  0.8137774  20.25171
##    0.004498433  26.43438  0.8137774  20.25171
##    0.005428675  26.43438  0.8137774  20.25171
##    0.006551286  26.43438  0.8137774  20.25171
##    0.007906043  26.43438  0.8137774  20.25171
##    0.009540955  26.43438  0.8137774  20.25171
##    0.011513954  26.43438  0.8137774  20.25171
##    0.013894955  26.43438  0.8137774  20.25171
##    0.016768329  26.43438  0.8137774  20.25171
```

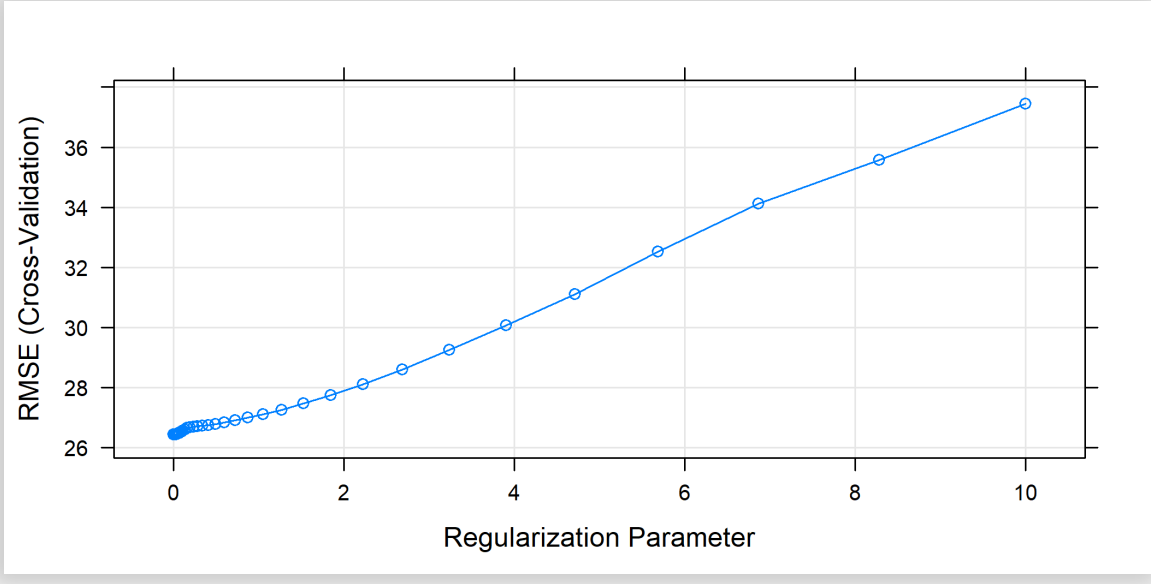
```
mod_rf
```

```
## Random Forest
##
## 19230 samples
##    12 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 15384, 15383, 15384, 15384, 15385
## Resampling results across tuning parameters:
##
##   min.node.size  mtry  RMSE      Rsquared   MAE
##   10              2    42.97906  0.7374511  34.38019
##   10              4    33.58867  0.7893763  26.34715
##   10              6    28.47842  0.8179302  22.01305
##   10              8    25.65320  0.8359859  19.62846
##   10             10    24.71868  0.8424168  18.83413
##   20              2    44.40910  0.7168126  35.67302
##   20              4    34.16609  0.7813087  26.80165
##   20              6    28.29841  0.8191724  21.81372
##   20              8    25.85930  0.8350953  19.81877
##   20             10    24.62077  0.8432578  18.77625
##   30              2    43.63977  0.7206202  34.90311
##   30              4    33.66368  0.7827368  26.32678
##   30              6    28.19928  0.8202997  21.73367
##   30              8    26.05553  0.8339279  19.93278
##   30             10    24.80283  0.8419966  18.92405
##
```

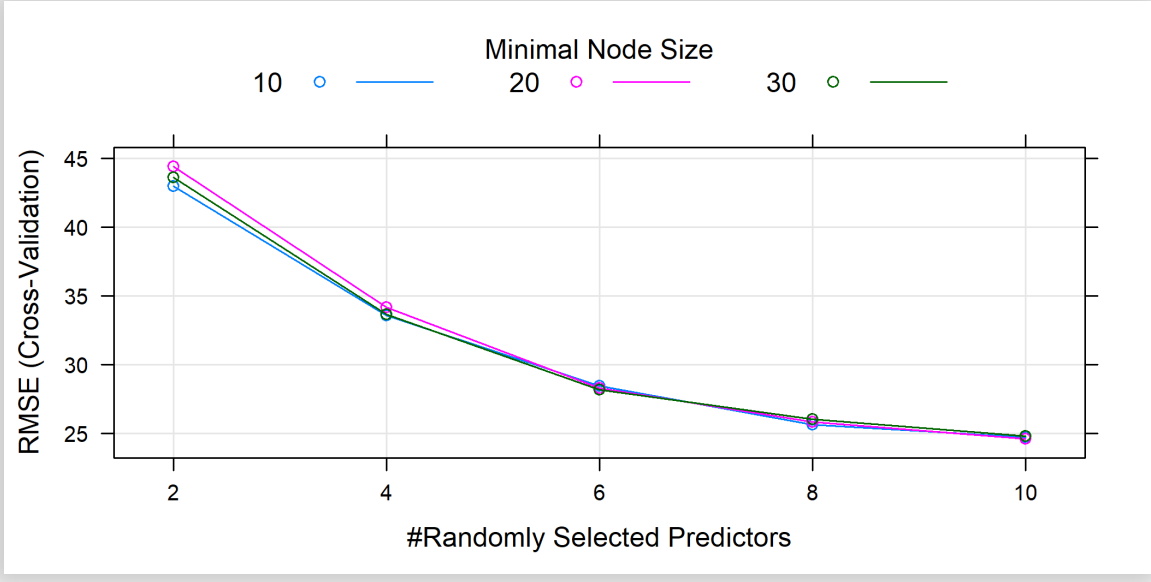
```
plot(mod_ridge)
```



```
plot(mod_lasso)
```



```
plot(mod_rf)
```

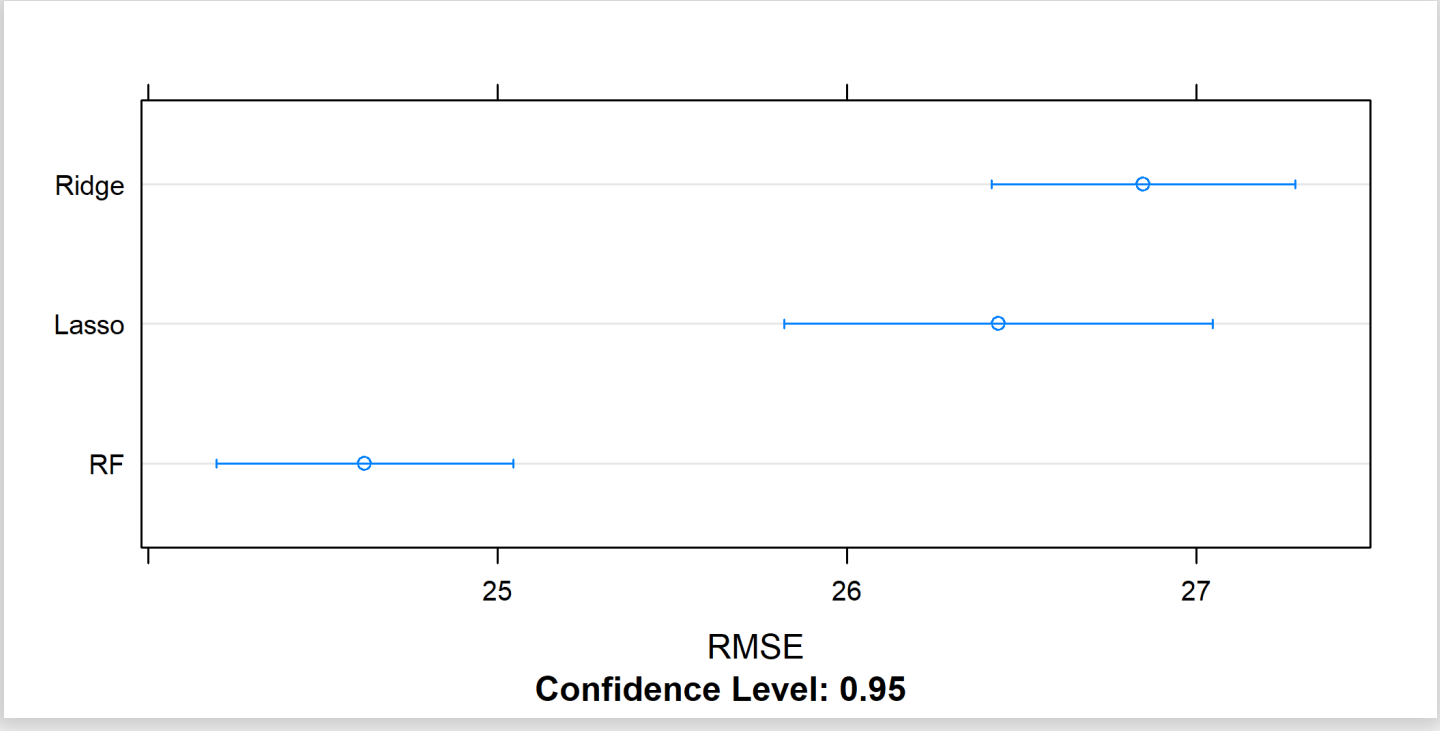

Minimal Node Size
10
20
30
RMSE (Cross-Validation)
45
40
35
30
25
2
4
6
8
10
#Randomly Selected Predictors

Comparing Models

```
resamps <- resamples(list(Ridge = mod_ridge,
                          Lasso = mod_lasso,
                          RF = mod_rf))
summary(resamps)
```

```
##
## Call:
## summary.resamples(object = resamps)
##
## Models: Ridge, Lasso, RF
## Number of resamples: 5
##
## MAE
##           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## Ridge 20.39991 20.45566 20.53437 20.55394 20.53526 20.84448    0
## Lasso 19.88140 19.94063 20.43765 20.25164 20.48677 20.51177    0
## RF    18.40720 18.66643 18.85038 18.77625 18.94520 19.01202    0
##
## RMSE
##           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## Ridge 26.50302 26.66880 26.73841 26.84827 26.91660 27.41453    0
## Lasso 25.87176 25.91696 26.76098 26.43435 26.79996 26.82210    0
## RF    24.04310 24.64214 24.67820 24.62077 24.82394 24.91648    0
##
```

```
dotplot(resamps, metric = "RMSE")
```



Predicting

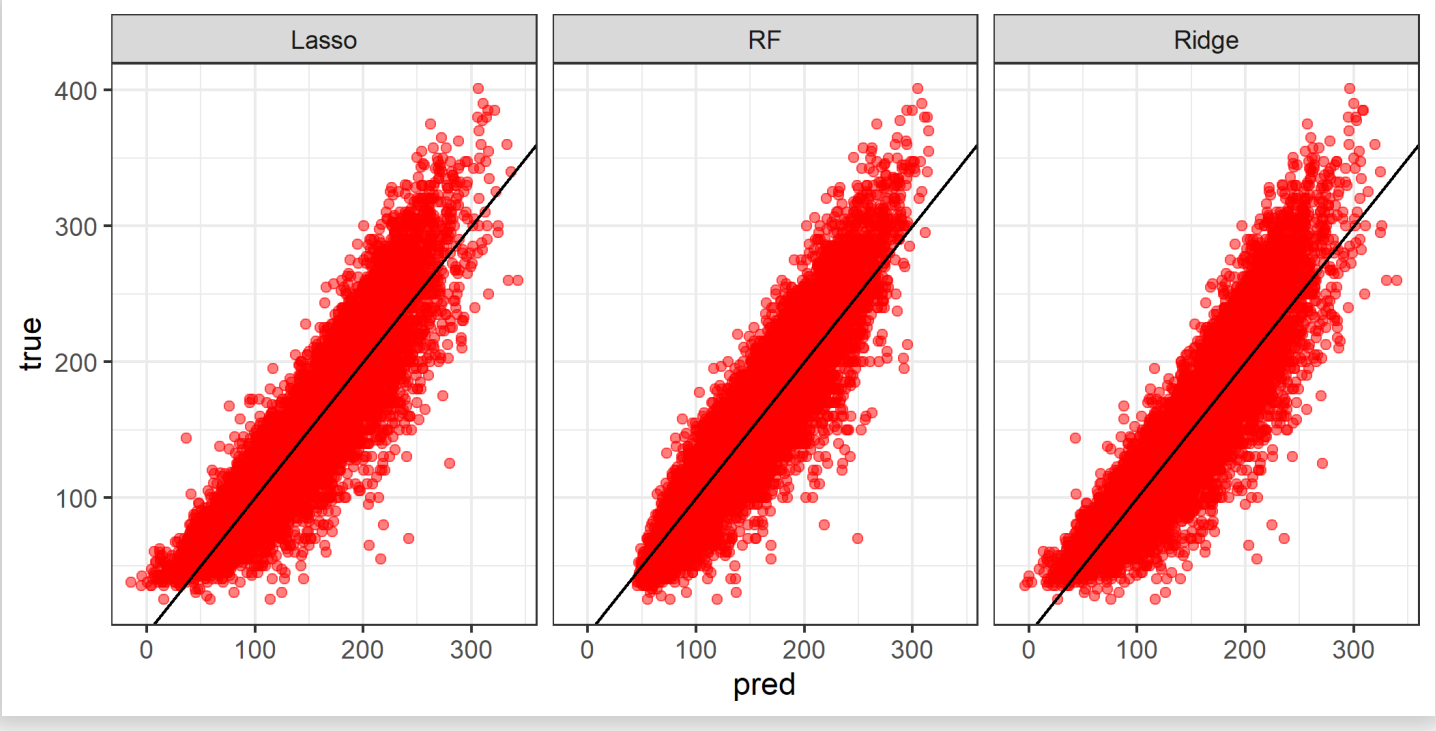
```
pred_ridge <- predict(mod_ridge, newdata = ipf_te)
pred_lasso <- predict(mod_lasso, newdata = ipf_te)
pred_rf <- predict(mod_rf, newdata = ipf_te)

rmse_ridge <- RMSE(pred_ridge, ipf_te$best3bench_kg)
rmse_lasso <- RMSE(pred_lasso, ipf_te$best3bench_kg)
rmse_rf <- RMSE(pred_rf, ipf_te$best3bench_kg)

glue("Test RMSE Ridge: {format(rmse_ridge, digits = 3)}
     Test RMSE Lassoe: {format(rmse_lasso, digits = 3)}
     Test RMSE RF: {format(rmse_rf, digits = 3)}")
```

```
## Test RMSE Ridge: 26.5
## Test RMSE Lassoe: 26
## Test RMSE RF: 24.2
```

⚠ Is using a "regular" regression model the natural approach for these data?

Ask yourself what is this model good for, if at all 😮

```
bind_rows(
  tibble(method = "Ridge", pred = pred_ridge, true = ipf_te$best3k
  tibble(method = "Lasso", pred = pred_lasso, true = ipf_te$best3k
  tibble(method = "RF", pred = pred_rf, true = ipf_te$best3bench_k
  ggplot(aes(pred, true)) +
  geom_point(color = "red", alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0) +
  facet_wrap(~ method) +
  theme_bw()
```



The Future Solution: `tidymodels`

Inspired by Julia Silge

Packages under tidymodels

- `paresnip`: tidy `caret`
- `dials` and `tune`: specifying and tuning model parameters
- `rsample`: sampling, data partitioning
- `recipes` and `embed`: preprocessing and creating model matrices
- `infer`: tidy statistics
- `yardstick`: measuring models performance
- `broom`: convert models output into tidy tibbles

And more.

> ⚠ All `tidymodels` packages are under development!

Split Data

The `initial_split()` function is from the `rsample` package:

```
library(tidymodels)

ipf_split_obj <- ipf_lifts %>%
  initial_split(prop = 0.6, strata = equipment)

ipf_tr <- training(ipf_split_obj)
ipf_te <- testing(ipf_split_obj)

glue("train no. of rows: {nrow(ipf_tr)}
     test no. of rows: {nrow(ipf_te)}")
```

```
## train no. of rows: 19229
## test no. of rows: 12818
```

```
print(ipf_split_obj)
```

```
## <19229/12818/32047>
```

Preprocess (but we're not gonna use it)

The `recipe()` function is from the `recipes` package. It allows you to specify a python-like pipe you can later apply to any dataset, including all preprocessing steps:

```
ipf_rec <- recipe(best3bench_kg ~ ., data = ipf_tr)
ipf_rec
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         12
```

`recipes` contains more preprocessing `step_`s than you imagine:

```
ipf_rec <-  ipf_rec %>%
  step_normalize(all_numeric())
```

After you have your `recipe` you need to `prep()` materials...

```
ipf_rec <- ipf_rec %>% prep(ipf_tr)

ipf_rec
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         12
##
## Training data contained 19229 data points and no missing data.
##
## Operations:
##
## Centering and scaling for age, bodyweight_kg, year, month, ... [trained]
```

At this point our `recipe` has all necessary `sd` and `means` for numeric variables.

And then we `bake()`:

```
ipf_tr2 <- ipf_rec %>% bake(ipf_tr)
ipf_te2 <- ipf_rec %>% bake(ipf_te)

glue("mean of age in orig training: {format(mean(ipf_tr$age), digi
     mean of age in baked training: {format(mean(ipf_tr2$age), dig
```

```
## mean of age in orig training: 36.7, sd: 14.3
## mean of age in baked training: 0, sd: 1
```

```
glue("mean of age in orig testing: {format(mean(ipf_te$age), digit
     mean of age in baked testing: {format(mean(ipf_te2$age), digi
```

```
## mean of age in orig testing: 36.5, sd: 14.3
## mean of age in baked testing: -0, sd: 0.997
```

Or you can do it all in a single pipe:

```
ipf_rec <- recipe(best3bench_kg ~ ., data = ipf_tr) %>%
  step_normalize(all_numeric()) %>%
  prep(ipf_tr)

ipf_tr2 <- ipf_rec %>% bake(ipf_tr)
ipf_te2 <- ipf_rec %>% bake(ipf_te)

glue("mean of age in orig training: {format(mean(ipf_tr$age), digi
     mean of age in baked training: {format(mean(ipf_tr2$age), dig
```

```
## mean of age in orig training: 36.7, sd: 14.3
## mean of age in baked training: 0, sd: 1
```

```
glue("mean of age in orig testing: {format(mean(ipf_te$age), digit
     mean of age in baked testing: {format(mean(ipf_te2$age), digi
```

```
## mean of age in orig testing: 36.5, sd: 14.3
## mean of age in baked testing: -0, sd: 0.997
```

Modeling

For now let us use the original `ipf_tr` data.

Functions `linear_reg()` and `set_engine()` are from the `parsnip` package:

```
mod_ridge_spec <- linear_reg(mixture = 0, penalty = 0.001) %>%
  set_engine(engine = "glmnet")

mod_ridge_spec
```

```
## Linear Regression Model Specification (regression)
##
## Main Arguments:
##   penalty = 0.001
##   mixture = 0
##
## Computational engine: glmnet
```

```
mod_ridge <- mod_ridge_spec %>%
  fit(best3bench_kg ~ ., data = ipf_tr)

mod_ridge
```

```
## parsnip model object
##
## Fit in:  51ms
## Call:  glmnet::glmnet(x = as.matrix(x), y = y, family = "gaussian",
##
##     Df    %Dev Lambda
## 1   51 0.00000  43070
## 2   51 0.00398  39250
## 3   51 0.00437  35760
## 4   51 0.00479  32580
## 5   51 0.00526  29690
## 6   51 0.00577  27050
## 7   51 0.00632  24650
## 8   51 0.00694  22460
## 9   51 0.00761  20460
## 10  51 0.00834  18640
## 11  51 0.00915  16990
## 12  51 0.01003  15480
## 13  51 0.01100  14100
## 14  51 0.01206  12850
## 15  51 0.01322  11710
## 16  51 0.01449  10670
## 17  51 0.01588   9721
## 18  51 0.01740   8858
```

In a single pipe:

```
mod_lasso <- linear_reg(mixture = 1, penalty = 0.001) %>%
  set_engine(engine = "glmnet") %>%
  fit(best3bench_kg ~ ., data = ipf_tr)

mod_lasso
```

```
## parsnip model object
##
## Fit in:  41ms
## Call:  glmnet::glmnet(x = as.matrix(x), y = y, family = "gaussian",
##
##    Df    %Dev Lambda
## 1   0 0.00000 43.070
## 2   1 0.08401 39.250
## 3   2 0.16420 35.760
## 4   2 0.24370 32.580
## 5   2 0.30980 29.690
## 6   2 0.36460 27.050
## 7   2 0.41010 24.650
## 8   2 0.44780 22.460
## 9   2 0.47920 20.460
## 10  2 0.50520 18.640
## 11  2 0.52690 16.990
## 12  2 0.54480 15.480
## 13  2 0.55970 14.100
## 14  3 0.57220 12.850
```

Can also use `fit_xy()` a-la `sklearn`:

```
mod_rf <- rand_forest(mode = "regression", mtry = 4, trees = 50, r
  set_engine("ranger") %>%
  fit_xy(x = ipf_tr[, -7],
         y = ipf_tr$best3bench_kg)

mod_rf
```

```
## parsnip model object
##
## Fit in:  641msRanger result
##
## Call:
##  ranger::ranger(formula = formula, data = data, mtry = ~4, num.trees = ~
##
## Type:                             Regression
## Number of trees:                  50
## Sample size:                      19229
## Number of independent variables:  12
## Mtry:                             4
## Target node size:                 30
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       562.6713
## R squared (OOB):                  0.8499346
```

Notice how easy it is to get the model's results in a tidy way using the `tidy()` function:

```
tidy(mod_ridge)
```

```
## # A tibble: 5,200 x 5
##    term                 step  estimate lambda dev.ratio
##    <chr>               <dbl>     <dbl>  <dbl>     <dbl>
##  1 (Intercept)             1  1.48e+ 2 43072.  2.59e-36
##  2 sexM                    1  8.51e-35 43072.  2.59e-36
##  3 eventSB                 1  3.20e-36 43072.  2.59e-36
##  4 eventSBD                1 -2.44e-35 43072.  2.59e-36
##  5 equipmentSingle-ply     1  3.04e-35 43072.  2.59e-36
##  6 age                     1 -5.54e-37 43072.  2.59e-36
##  7 divisionJuniors         1 -4.85e-36 43072.  2.59e-36
##  8 divisionLight           1 -1.81e-35 43072.  2.59e-36
##  9 divisionMasters 1       1 -1.11e-36 43072.  2.59e-36
## 10 divisionMasters 2       1 -1.54e-35 43072.  2.59e-36
## # ... with 5,190 more rows
```

Predicting

```
results_test <- mod_ridge %>%
  predict(new_data = ipf_te, penalty = 0.001) %>%
  mutate(
    truth = ipf_te$best3bench_kg,
    method = "Ridge"
  ) %>%
  bind_rows(mod_lasso %>%
    predict(new_data = ipf_te) %>%
    mutate(
      truth = ipf_te$best3bench_kg,
      method = "Lasso"
  )) %>%
  bind_rows(mod_rf %>%
    predict(new_data = ipf_te) %>%
    mutate(
      truth = ipf_te$best3bench_kg,
      method = "RF"
  ))

dim(results_test)
```

```
## [1] 38454     3
```

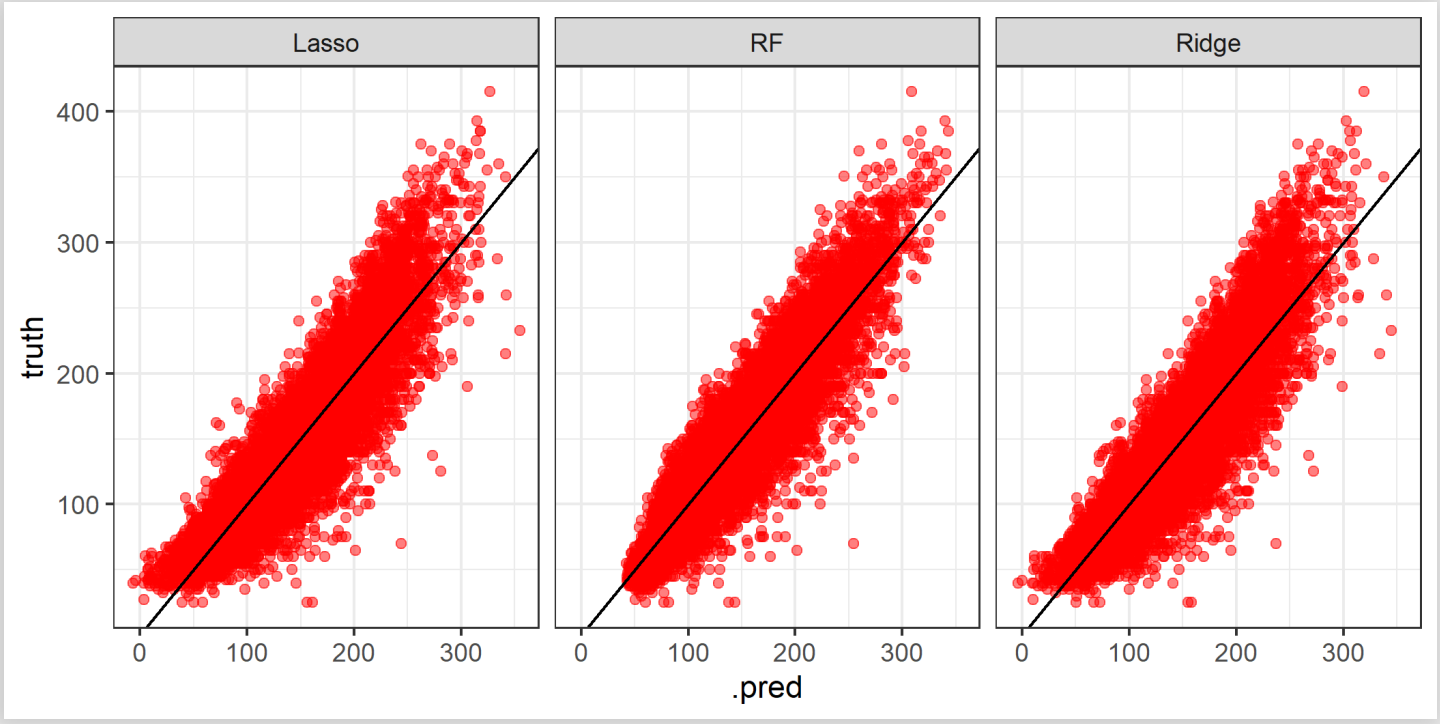
Comparing Models

The package `yardstick` has tons of performance metrics:

```
results_test %>%
  group_by(method) %>%
  yardstick::rmse(truth = truth, estimate = .pred)
```

```
## # A tibble: 3 x 4
##   method .metric .estimator .estimate
##   <chr>  <chr>   <chr>          <dbl>
## 1 Lasso  rmse    standard        26.4
## 2 RF     rmse    standard        23.7
## 3 Ridge  rmse    standard        26.8
```

```
results_test %>%
  ggplot(aes(.pred, truth)) +
  geom_point(color = "red", alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0) +
  facet_wrap(~ method) +
  theme_bw()
```



Tuning

This isn't completely clear to me, but it seems to work:

Define your model spec, using `tune()` from the `tune` package (needs to be installed separately) for a paramter you wish to tune:

```
library(tune)

mod_rf_spec <- rand_forest(mode = "regression",
                           mtry = tune("mtry"),
                           min_n = tune("min_n")) %>%
  set_engine("ranger")
```

Define the `grid` on which you train your params, with the `dials` package:

```
rf_grid <- grid_regular(mtry(range(2, 10)), min_n(range(10, 30)),
                        levels = c(5, 3))

rf_grid
```

```
## # A tibble: 15 x 2
##     mtry min_n
##    <int> <int>
##  1     2    10
##  2     4    10
##  3     6    10
##  4     8    10
##  5    10    10
##  6     2    20
##  7     4    20
##  8     6    20
##  9     8    20
## 10    10    20
## 11     2    30
## 12     4    30
## 13     6    30
## 14     8    30
## 15    10    30
```

Split your data into a few folds for Cross Validation with `vfold_cv()` from the `rsample` package:

```
cv_splits <- vfold_cv(ipf_tr, v = 5)

cv_splits
```

```
## #  5-fold cross-validation
## # A tibble: 5 x 2
##   splits               id
##   <named list>         <chr>
## 1 <split [15.4K/3.8K]> Fold1
## 2 <split [15.4K/3.8K]> Fold2
## 3 <split [15.4K/3.8K]> Fold3
## 4 <split [15.4K/3.8K]> Fold4
## 5 <split [15.4K/3.8K]> Fold5
```

Now perform cross validation with `tune_grid()` from the `tune` package:

```
tune_res <- tune_grid(recipe(best3bench_kg ~ ., data = ipf_tr),
                      model = mod_rf_spec,
                      resamples = cv_splits,
                      grid = rf_grid,
                      metrics = metric_set(rmse))
tune_res
```

```
## #  5-fold cross-validation
## # A tibble: 5 x 4
##   splits               id    .metrics          .notes
## * <list>               <chr> <list>            <list>
## 1 <split [15.4K/3.8K]> Fold1 <tibble [15 x 5]> <tibble [0 x 1]>
## 2 <split [15.4K/3.8K]> Fold2 <tibble [15 x 5]> <tibble [0 x 1]>
## 3 <split [15.4K/3.8K]> Fold3 <tibble [15 x 5]> <tibble [0 x 1]>
## 4 <split [15.4K/3.8K]> Fold4 <tibble [15 x 5]> <tibble [0 x 1]>
## 5 <split [15.4K/3.8K]> Fold5 <tibble [15 x 5]> <tibble [0 x 1]>
```

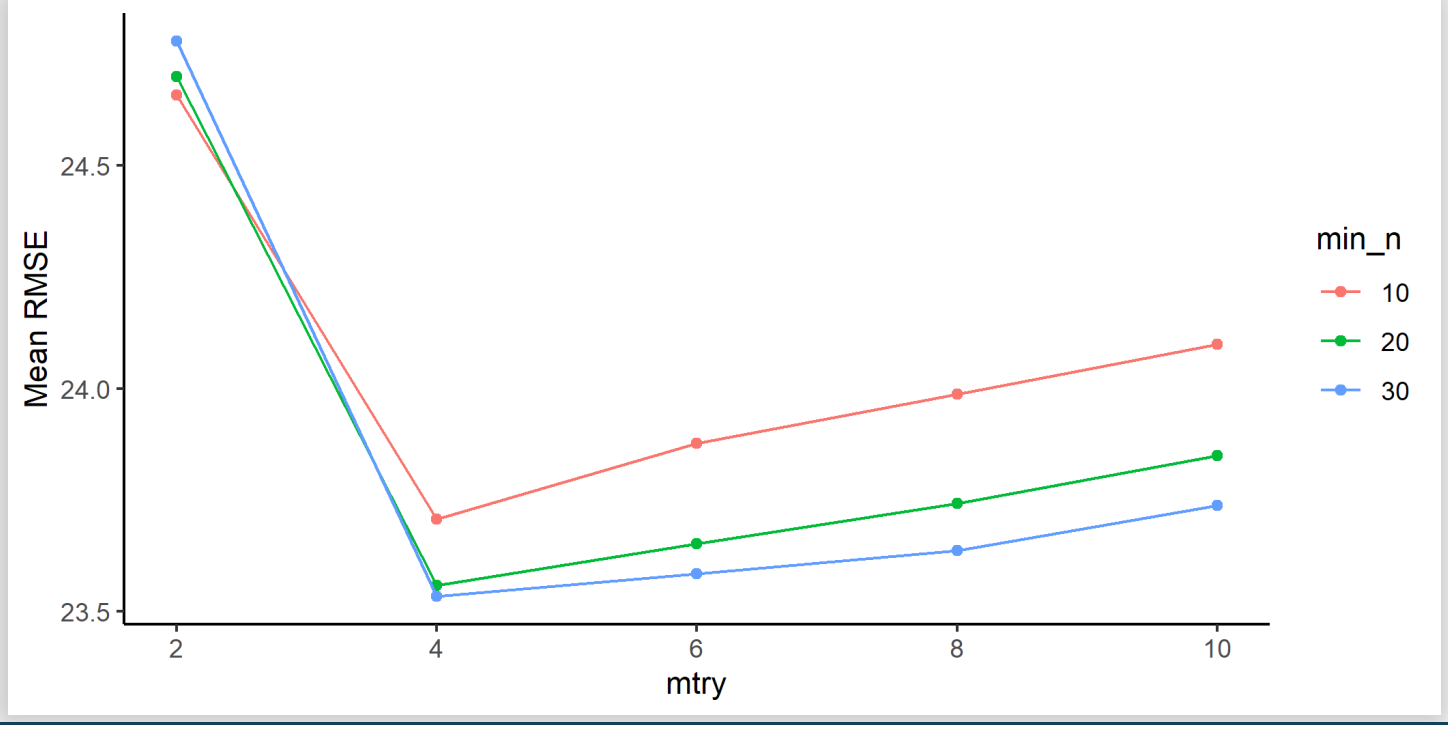
Collect the mean metric across folds:

```
estimates <- collect_metrics(tune_res)

estimates
```

```
## # A tibble: 15 x 7
##     mtry min_n .metric .estimator  mean     n std_err
##    <int> <int> <chr>   <chr>      <dbl> <int>   <dbl>
##  1     2    10 rmse    standard    24.7     5   0.197
##  2     2    20 rmse    standard    24.7     5   0.190
##  3     2    30 rmse    standard    24.8     5   0.192
##  4     4    10 rmse    standard    23.7     5   0.186
##  5     4    20 rmse    standard    23.6     5   0.192
##  6     4    30 rmse    standard    23.5     5   0.181
##  7     6    10 rmse    standard    23.9     5   0.191
##  8     6    20 rmse    standard    23.7     5   0.186
##  9     6    30 rmse    standard    23.6     5   0.186
## 10     8    10 rmse    standard    24.0     5   0.203
## 11     8    20 rmse    standard    23.7     5   0.185
## 12     8    30 rmse    standard    23.6     5   0.185
## 13    10    10 rmse    standard    24.1     5   0.206
## 14    10    20 rmse    standard    23.8     5   0.181
## 15    10    30 rmse    standard    23.7     5   0.183
```

Choose best paramter:

```
estimates %>%
  mutate(min_n = factor(min_n)) %>%
  ggplot(aes(x = mtry, y = mean, color = min_n)) +
  geom_point() +
  geom_line() +
  labs(y = "Mean RMSE") +
  theme_classic()
```



`infer`: Tidy Statistics

Statistical Q1

Is there a relation between men and women and the type of equipment they use in 2019? Assume observations are independent.

```
sex_vs_equipment <- ipf_lifts %>%
  filter(year == 2019) %>%
  select(sex, equipment) %>%
  table()

sex_vs_equipment
```

```
##     equipment
## sex Raw Single-ply
##   F 678         186
##   M 854         287
```

```
prop.table(sex_vs_equipment, margin = 1)
```

```
##     equipment
## sex       Raw Single-ply
##   F 0.7847222  0.2152778
##   M 0.7484663  0.2515337
```

Statistical Q2

Is there a difference between men and women age in 2019? Assume observations are independent.

```
ipf_lifts %>%
  filter(year == 2019) %>%
  group_by(sex) %>% summarise(avg = mean(age), sd = sd(age), n = n
```

```
## # A tibble: 2 x 4
##   sex     avg    sd     n
##   <fct> <dbl> <dbl> <int>
## 1 F      36.0  15.5   864
## 2 M      38.8  16.7  1141
```

Same Problem!

Varied interface, varied output.

```
prop.test(sex_vs_equipment[,1], rowSums(sex_vs_equipment))
```

```
##
##     2-sample test for equality of proportions with continuity
##     correction
##
## data:  sex_vs_equipment[, 1] out of rowSums(sex_vs_equipment)
## X-squared = 3.3872, df = 1, p-value = 0.0657
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  -0.001975717  0.074487646
## sample estimates:
##    prop 1    prop 2
## 0.7847222 0.7484663
```

```
t.test(age ~ sex, data = ipf_lifts %>% filter(year == 2019))
```

```
##
##     Welch Two Sample t-test
##
## data:  age by sex
## t = -3.8797, df = 1921.8, p-value = 0.0001081
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -4.228319 -1.388844
## sample estimates:
## mean in group F mean in group M
##        35.97801        38.78659
```

The `generics::tidy()` Approach

(Also available when you load several other packages, like `broom` and `yardstick`)

```
tidy(prop.test(sex_vs_equipment[,1], rowSums(sex_vs_equipment)))
```

```
## # A tibble: 1 x 9
##   estimate1 estimate2 statistic p.value parameter conf.low conf.high met
##       <dbl>     <dbl>     <dbl>   <dbl>     <dbl>    <dbl>     <dbl> <ch
## 1     0.785     0.748      3.39  0.0657         1 -0.00198    0.0745 2-s
## # ... with 1 more variable: alternative <chr>
```

```
tidy(t.test(age ~ sex, data = ipf_lifts %>% filter(year == 2019)))
```

```
## # A tibble: 1 x 10
##   estimate estimate1 estimate2 statistic p.value parameter conf.low
##      <dbl>     <dbl>     <dbl>     <dbl>   <dbl>     <dbl>    <dbl>
## 1    -2.81      36.0      38.8     -3.88 1.08e-4     1922.    -4.23
## # ... with 3 more variables: conf.high <dbl>, method <chr>,
## #   alternative <chr>
```

The `infer` Approach

> infer implements an expressive grammar to perform statistical inference that coheres with the tidyverse design framework

4 main verbs for a typical flow:

- `specify()` - dependent/independent variables, formula
- `hypothesize()` - declare the null hypothesis
- `generate()` - generate data reflecting the null hypothesis (the permutation/bootstrap approach)
- `calculate()` - calculate a distribution of statistics from the generated data, from which you can extract conclusion based on a p-value for example

`infer` Diff in Proportions Test

Get the observed statistic (here manually in order to not confuse you, there *is* a way via `infer`):

```
#    equipment
# sex Raw Single-ply
#   F 678         186
#   M 854         287
p_F <- sex_vs_equipment[1, 1] / (sum(sex_vs_equipment[1, ]))
p_M <- sex_vs_equipment[2, 1] / (sum(sex_vs_equipment[2, ]))
obs_diff <- p_F - p_M
obs_diff
```

```
## [1] 0.03625596
```

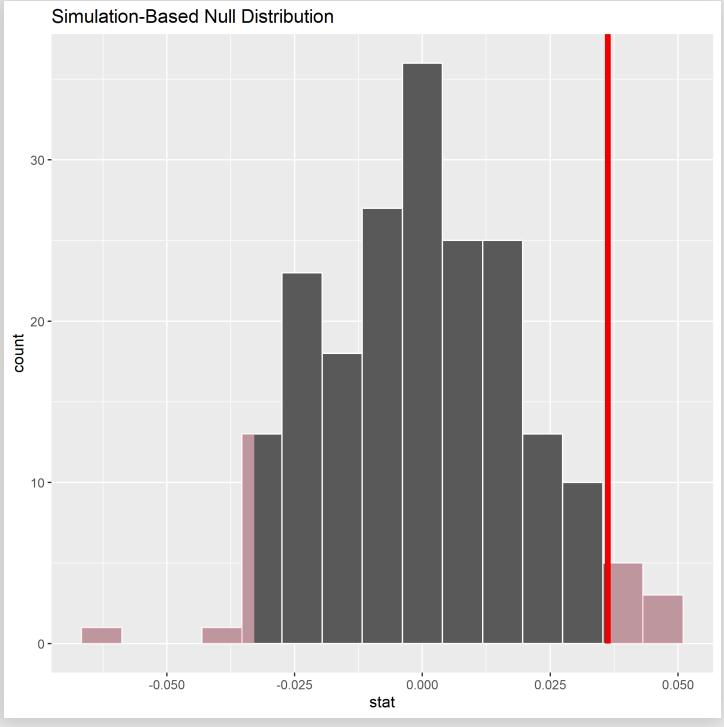
Get distribution of the difference in proportions under null hypothesis

```
diff_null_perm <- ipf_lifts %>%
  filter(year == 2019) %>%
  specify(equipment ~ sex, success = "Raw") %>%
  hypothesize(null = "independence") %>%
  generate(reps = 200, type = "permute") %>%
  calculate(stat = "diff in props", order = c("F", "M"))

diff_null_perm
```

```
## # A tibble: 200 x 2
##    replicate      stat
##        <int>     <dbl>
##  1         1 -0.00442
##  2         2  0.0220
##  3         3 -0.000353
##  4         4 -0.0126
##  5         5  0.00371
##  6         6 -0.0105
##  7         7  0.00575
##  8         8 -0.0105
##  9         9  0.00982
## 10        10  0.0180
## # ... with 190 more rows
```

Visualize the permuted difference null distribution and the p-value

```
visualize(diff_null_perm) +
  shade_p_value(obs_stat = obs_diff, direction = "two_sided")
```



Get the actual p-value:

```
diff_null_perm %>%
  get_p_value(obs_stat = obs_diff, direction = "two_sided")
```

```
## # A tibble: 1 x 1
##   p_value
##     <dbl>
## 1    0.08
```

`infer` t Test (independent samples)

Get the observed statistic (here via `infer`):

```
obs_t <- ipf_lifts %>%
  filter(year == 2019) %>%
  specify(age ~ sex) %>%
  calculate(stat = "t", order = c("F", "M"))
obs_t
```

```
## # A tibble: 1 x 1
##    stat
##   <dbl>
## 1 -3.88
```

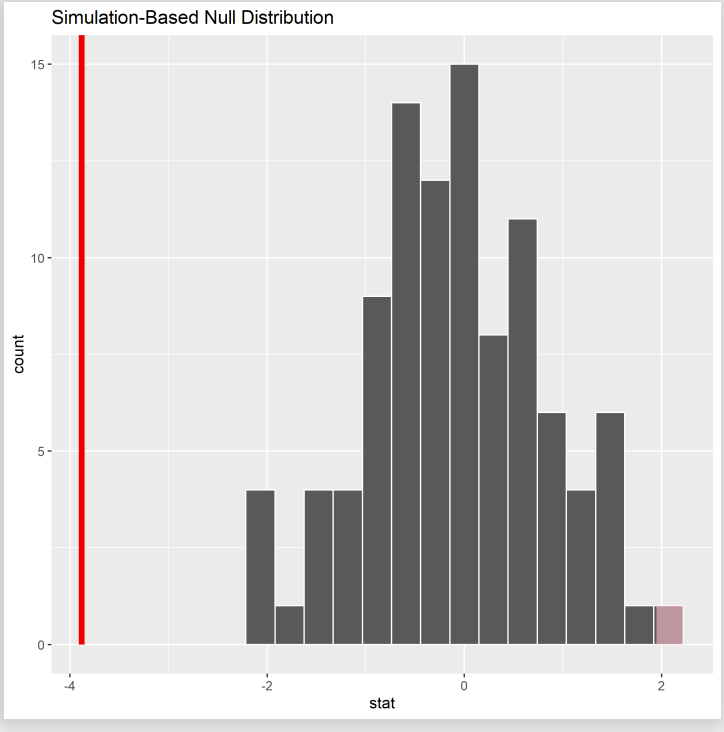
Get distribution of the t statistic under null hypothesis

```
t_null_perm <- ipf_lifts %>%
  filter(year == 2019) %>%
  specify(age ~ sex) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 100, type = "permute") %>%
  calculate(stat = "t", order = c("F", "M"))

t_null_perm
```

```
## # A tibble: 100 x 2
##    replicate   stat
##        <int>  <dbl>
##  1         1  0.190
##  2         2 -0.750
##  3         3 -0.338
##  4         4  1.74
##  5         5  0.267
##  6         6  1.58
##  7         7  0.358
##  8         8 -1.36
##  9         9 -1.37
## 10        10  0.162
## # ... with 90 more rows
```

Visualize the permuted t statistic null distribution and the two-sided p-value

```
visualize(t_null_perm) +
  shade_p_value(obs_stat = obs_t, direction = "two_sided")
```



Get the actual p-value:

```
t_null_perm %>%
  get_p_value(obs_stat = obs_t, direction = "two_sided")
```

```
## # A tibble: 1 x 1
##   p_value
##     <dbl>
## 1       0
```