

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

2020-03-28

APPLICATIONS



OF DATA SCIENCE

The Problem

APPLICATIONS



OF DATA SCIENCE

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

APPLICATIONS



OF DATA SCIENCE

IPF-Lifts: Predicting Bench Lifting

- Dataset was published as part of the [TidyTuesday](#) initiative
- 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/rfordatascience/tidyverse-tutorial/gh-pages/data/ipf.csv")

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

dim(ipf_lifts)

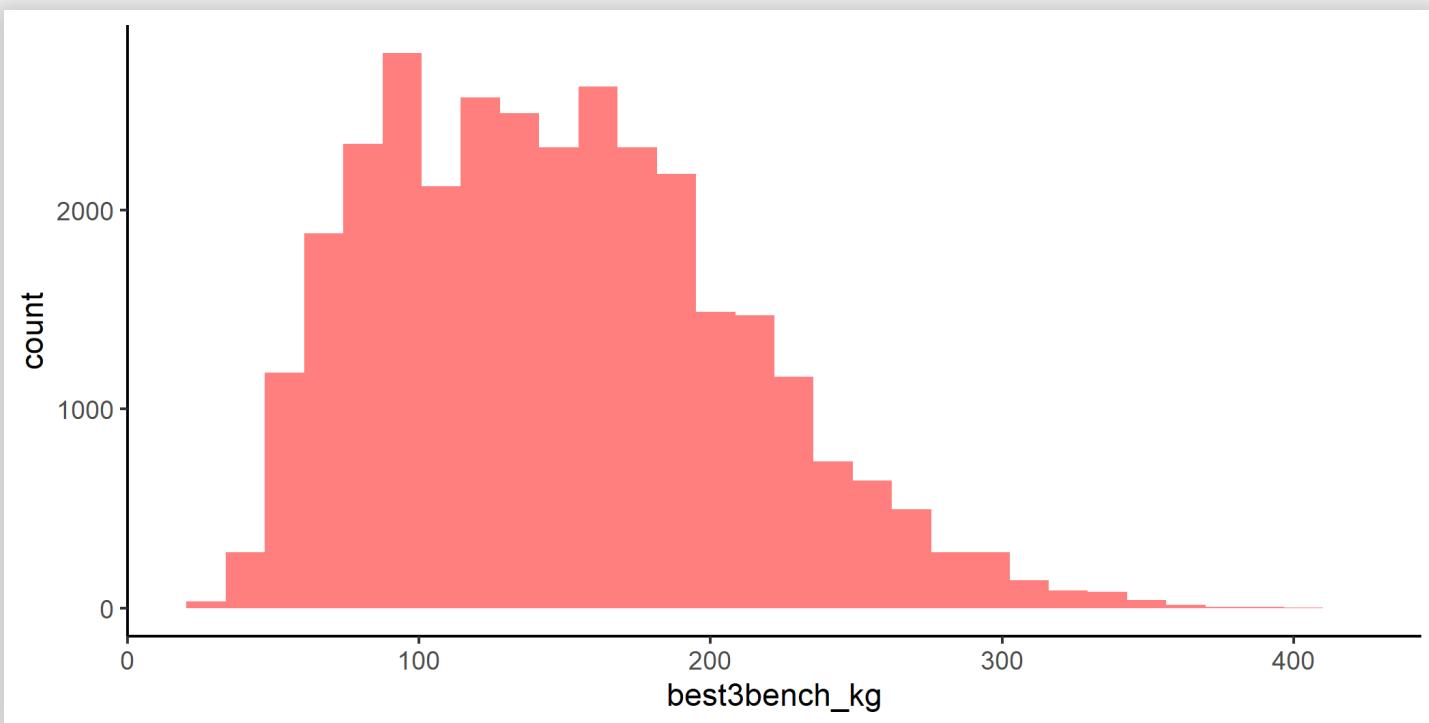
## [1] 32047     11
```

```
glimpse(ipf_lifts)
```

```
## Observations: 32,047
## Variables: 11
## $ sex <fct> F, ...
## $ event <fct> 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, ...
```

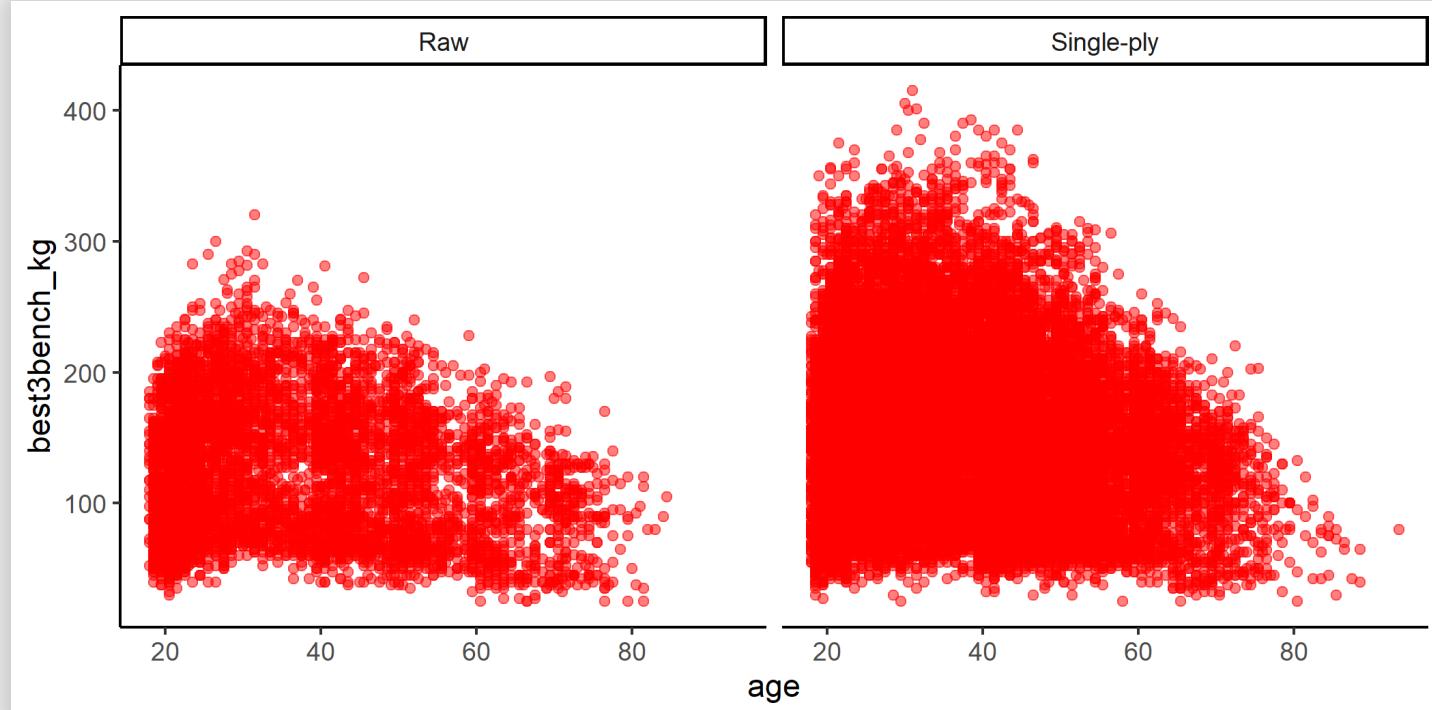
See the dependent variable distribution:

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



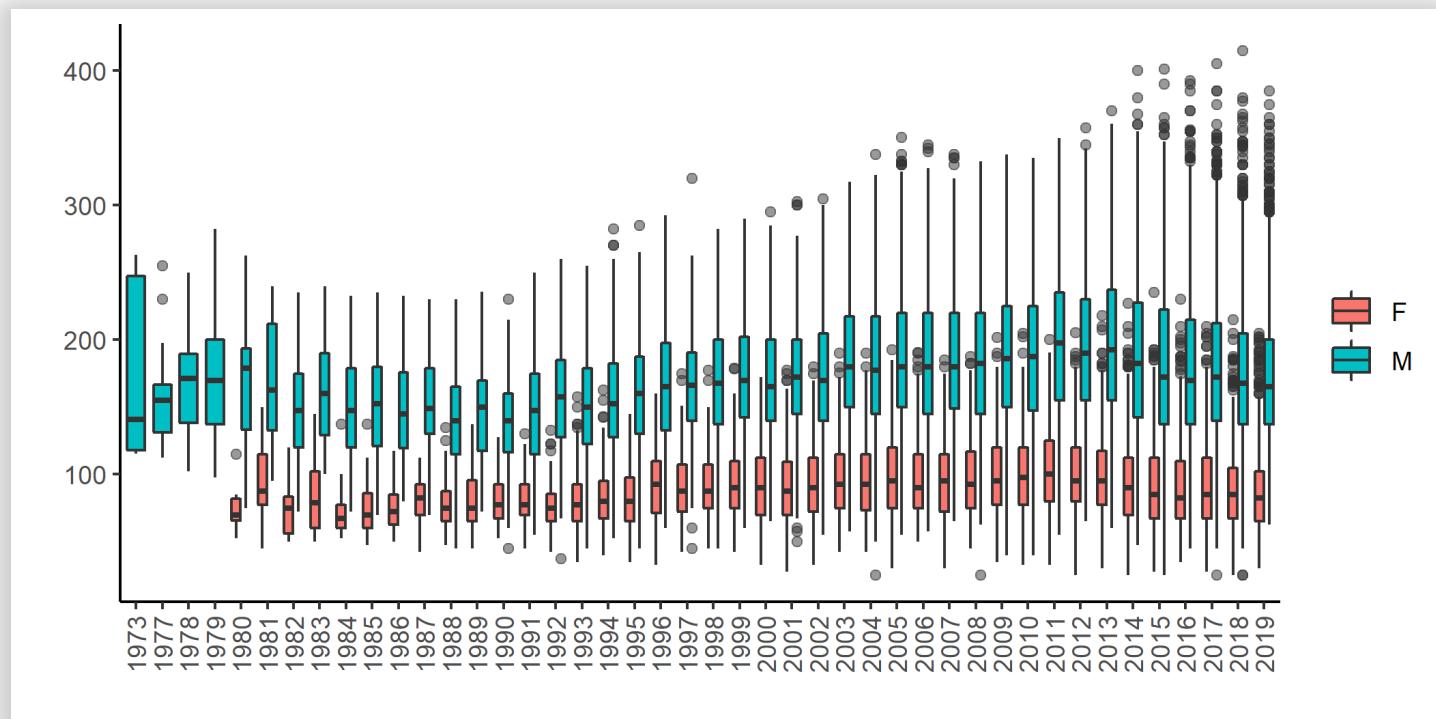
See it vs. say age, faceted by equipment:

```
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

APPLICATIONS



OF DATA SCIENCE

WARNING

 What you're about to see is not a good modeling/prediction flow. This is just an intro to tidy modeling. Some of the issues with how things are done here will be raised, some will have to wait till later in the course.

The Present Solution: caret

APPLICATIONS



OF DATA SCIENCE

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. 5-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: 15384, 15384, 15384, 15383, 15385
## Resampling results across tuning parameters:
##
##     lambda      RMSE    Rsquared    MAE
## 0.001000000  26.62116  0.8131891  20.33717
## 0.001206793  26.62116  0.8131891  20.33717
## 0.001456348  26.62116  0.8131891  20.33717
## 0.001757511  26.62116  0.8131891  20.33717
## 0.002120951  26.62116  0.8131891  20.33717
## 0.002559548  26.62116  0.8131891  20.33717
## 0.003088844  26.62116  0.8131891  20.33717
## 0.003727594  26.62116  0.8131891  20.33717
## 0.004498433  26.62116  0.8131891  20.33717
## 0.005428675  26.62116  0.8131891  20.33717
## 0.006551286  26.62116  0.8131891  20.33717
## 0.007006042  26.62116  0.8131891  20.33717
```

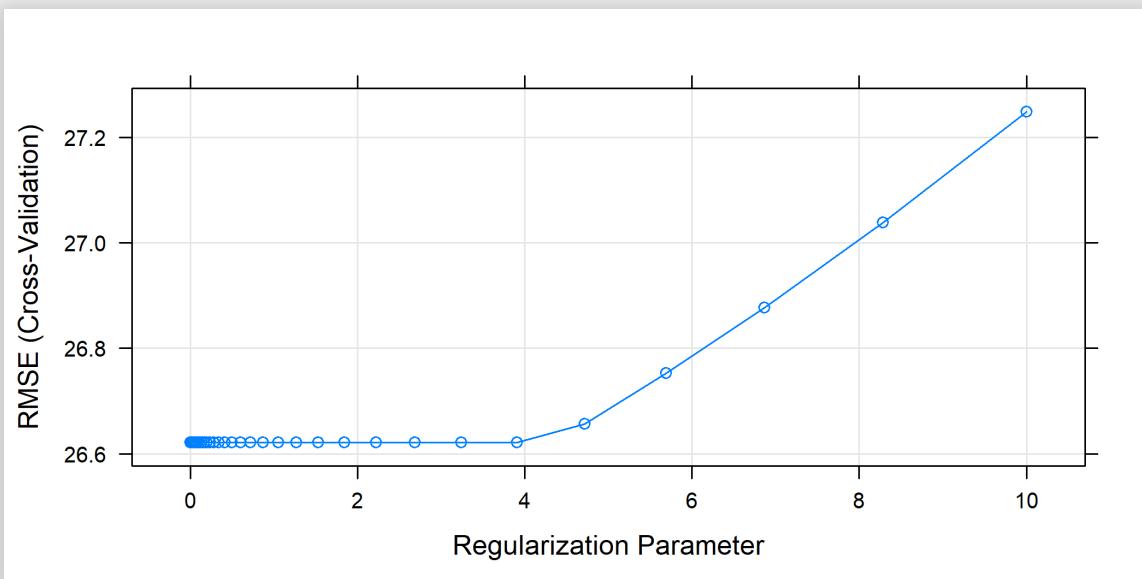
```
mod_lasso
```

```
## glmnet
##
## 19230 samples
##    12 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 15385, 15383, 15384, 15384, 15384
## Resampling results across tuning parameters:
##
##     lambda      RMSE    Rsquared    MAE
## 0.001000000 26.17306 0.8175264 20.02156
## 0.001206793 26.17306 0.8175264 20.02156
## 0.001456348 26.17306 0.8175264 20.02156
## 0.001757511 26.17306 0.8175264 20.02156
## 0.002120951 26.17306 0.8175264 20.02156
## 0.002559548 26.17306 0.8175264 20.02156
## 0.003088844 26.17306 0.8175264 20.02156
## 0.003727594 26.17306 0.8175264 20.02156
## 0.004498433 26.17306 0.8175264 20.02156
## 0.005428675 26.17306 0.8175264 20.02156
## 0.006551286 26.17306 0.8175264 20.02156
## 0.007906043 26.17306 0.8175264 20.02156
## 0.009540955 26.17306 0.8175264 20.02156
## 0.011513954 26.17306 0.8175264 20.02156
## 0.013894955 26.17306 0.8175264 20.02156
## 0.016768329 26.17306 0.8175264 20.02156
```

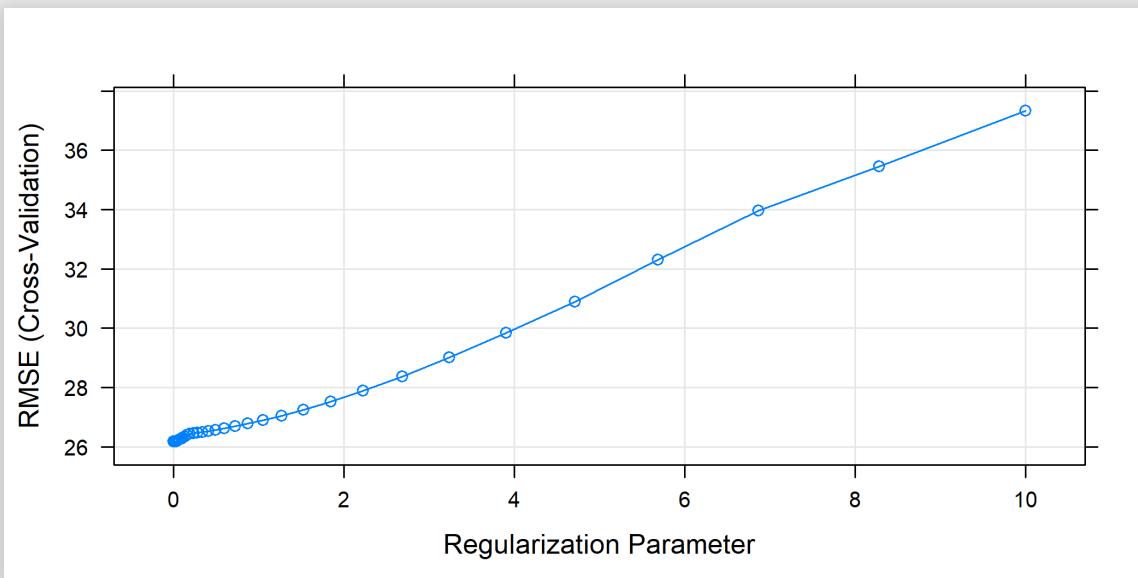
```
mod_rf
```

```
## Random Forest
##
## 19230 samples
##     12 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 15383, 15384, 15385, 15385, 15383
## Resampling results across tuning parameters:
##
##     min.node.size  mtry   RMSE      Rsquared    MAE
##     10            2     44.40672  0.7004199  35.68288
##     10            4     34.03673  0.7885218  26.78077
##     10            6     28.22751  0.8205982  21.77966
##     10            8     25.77867  0.8363904  19.74503
##     10           10    24.59374  0.8435989  18.71429
##     20            2     43.79755  0.7232273  35.13259
##     20            4     33.79883  0.7851855  26.46222
##     20            6     28.84054  0.8154826  22.23134
##     20            8     25.94377  0.8355417  19.81437
##     20           10    24.65925  0.8431627  18.74507
##     30            2     44.56685  0.7097288  35.74805
##     30            4     33.24043  0.7890227  25.97921
##     30            6     28.49393  0.8190458  21.96644
##     30            8     25.70164  0.8368138  19.67986
##     30           10    24.67475  0.8443715  18.76560
##
```

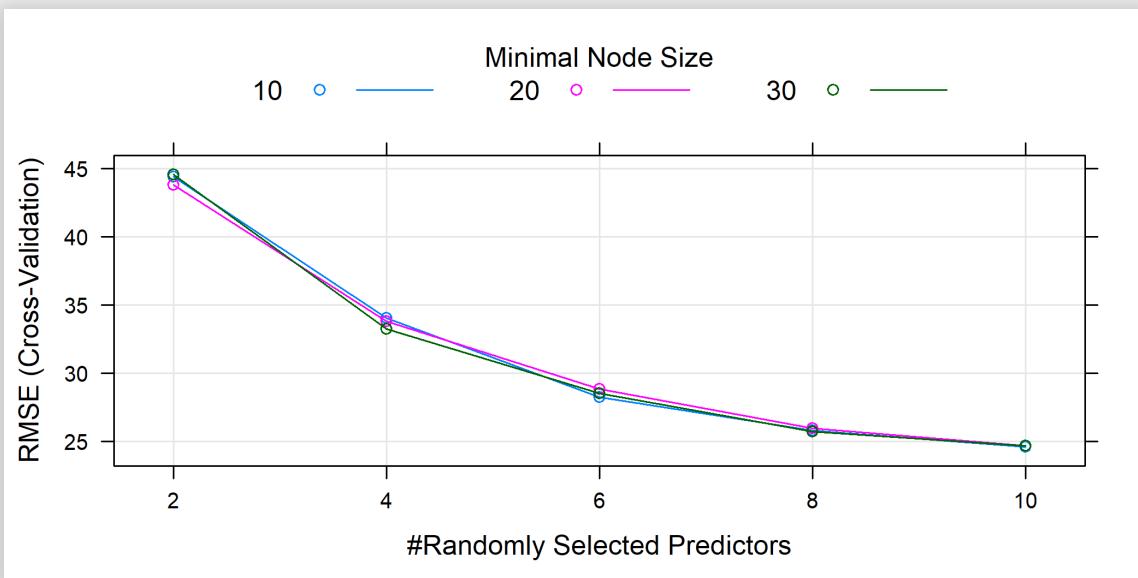
```
plot(mod_ridge)
```



```
plot(mod_lasso)
```



```
plot(mod_rf)
```

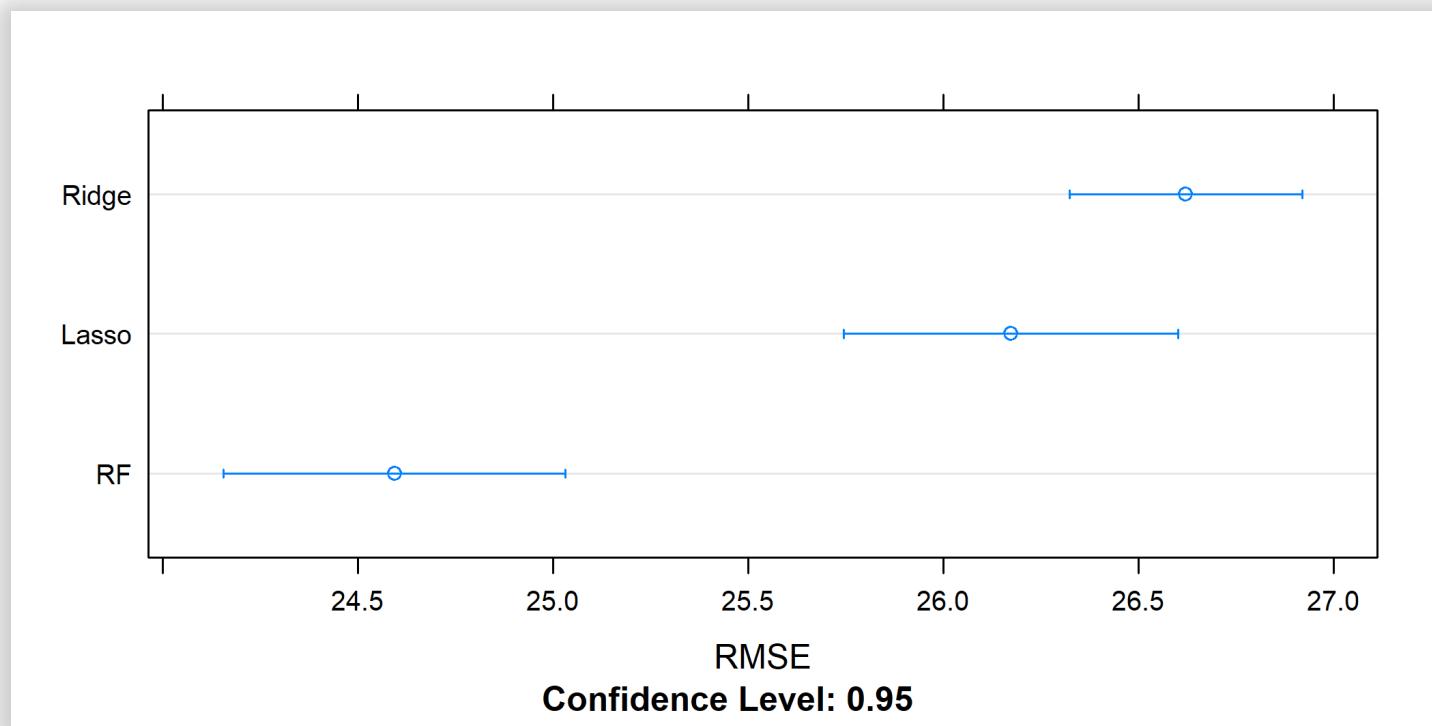


Comparing Models

```
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 20.08838 20.22315 20.40625 20.33717 20.42442 20.54366 0
## Lasso 19.78475 19.93539 19.95993 20.02156 19.97522 20.45251 0
## RF    18.43908 18.55445 18.60816 18.71429 18.75826 19.21148 0
##
## RMSE
##           Min.   1st Qu.   Median   Mean   3rd Qu.   Max. NA's
## Ridge 26.37970 26.38121 26.69809 26.62116 26.70952 26.93727 0
## Lasso 25.85918 26.00483 26.10097 26.17306 26.14174 26.75859 0
## RF    24.25151 24.28312 24.64581 24.59374 24.67083 25.11742 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)

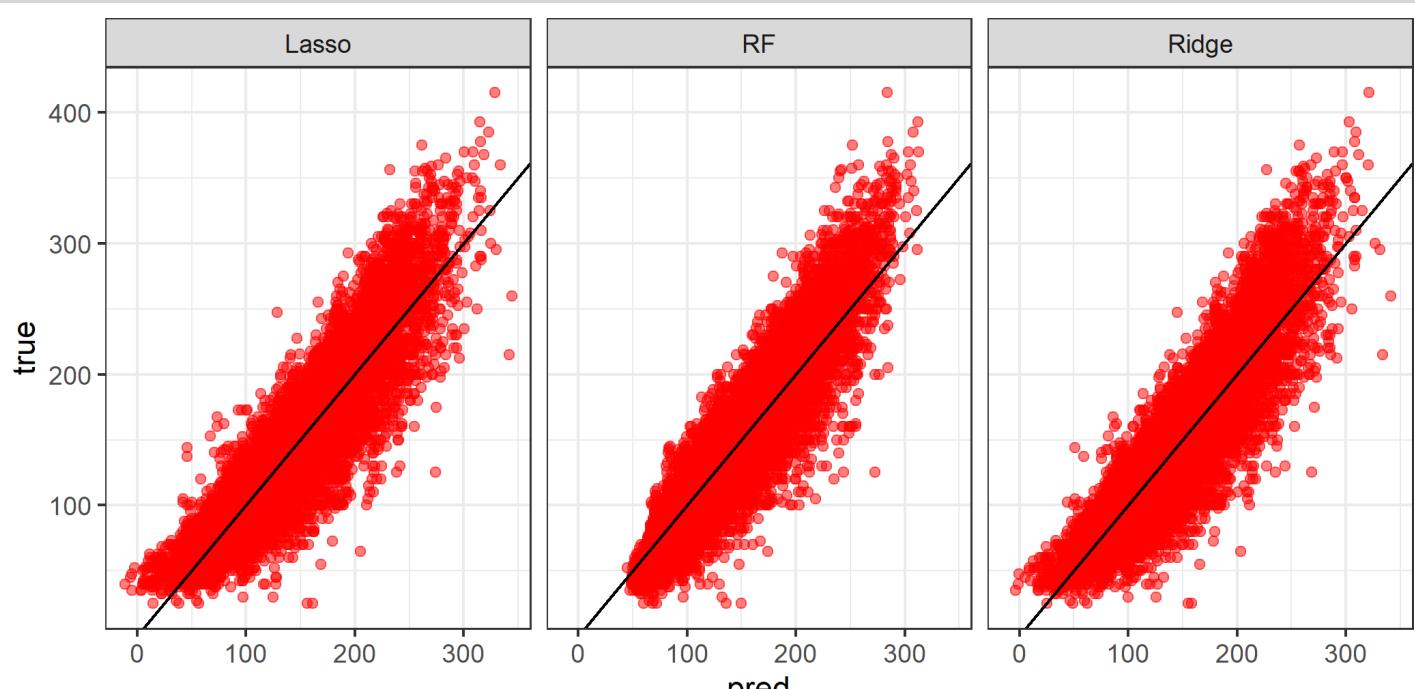
glue("Test RMSE Ridge: {format(rmse_ridge, digits = 3)}
      Test RMSE Lassoe: {format(rmse_lasso, digits = 3)}
      Test RMSE RF: {format(rmse_rf, digits = 3)}")
```

```
## Test RMSE Ridge: 26.8
## Test RMSE Lassoe: 26.4
## Test RMSE RF: 24.6
```

⚠️ 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](#)

APPLICATIONS



OF DATA SCIENCE

Packages under `tidymodels`

- `parsnip`: 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)}\n"
     "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.

```
ipf_rec$var_info
```

```
## # A tibble: 13 x 4
##   variable      type    role    source
##   <chr>        <chr>   <chr>   <chr>
## 1 sex          nominal predictor original
## 2 event         nominal predictor original
## 3 equipment    nominal predictor original
## 4 age           numeric predictor original
## 5 division     nominal predictor original
## 6 bodyweight_kg numeric predictor original
## 7 meet_name    nominal predictor original
## 8 year          numeric predictor original
## 9 month         numeric predictor original
## 10 dayofweek   numeric predictor original
## 11 age2         numeric predictor original
## 12 year2        numeric predictor original
## 13 best3bench_kg numeric outcome   original
```

```
ipf_rec$levels$meet_name
```

```
## $values
## [1] "3rd World University Powerlifting Cup"
## [2] "48th World Open Championships"
## [3] "6th World Classic Championships"
## [4] "Disabled Bench Press World Championships"
## [5] "Men's World Powerlifting Championships"
## [6] "Reykjavík International Games"
## [7] "Student's World Cup"
## [8] "University Powerlifting Cup"
## [9] "Women's World Powerlifting Championship"
## [10] "Women's World Powerlifting Championships"
## [11] "World Bench Press Championships"
## [12] "World Classic Bench Press Championships"
## [13] "World Classic Powerlifting Championships"
## [14] "World Classic Powerlifting Cup"
## [15] "World Disabled Bench Press Championships"
## [16] "World Games"
## [17] "World Juniors & Sub-Juniors Championships"
## [18] "World Juniors Powerlifting Championships"
## [19] "World Masters Bench Press Championships"
## [20] "World Masters Championships"
## [21] "World Masters Powerlifting Championships"
## [22] "World Men's Powerlifting Championship"
## [23] "World Open Bench Press Championships"
## [24] "World Open Championships"
## [25] "World Powerlifting Championships"
## [26] "World Students Cup"
```

```
ipf_rec$steps[ [1] ]$means
```

```
##          age bodyweight_kg      year      month      dayofweek
## 3.661433e+01 8.146179e+01 2.006976e+03 7.753029e+00 3.576005e+00
##          age2      year2 best3bench_kg
## 1.544206e+03 4.028041e+06 1.481720e+02
```

```
ipf_rec$steps[ [1] ]$sds
```

```
##          age bodyweight_kg      year      month      dayofweek
## 14.269106 24.865236 9.326106 2.716649 1.740679
##          age2      year2 best3bench_kg
## 1193.043744 37383.445309 60.975633
```

And then we `bake()` (or [juice\(\)](#)):

```
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$age), digits = 1)}  
      mean of age in baked training: {format(mean(ipf_tr2$age), digits = 1)}
```



```
## mean of age in orig training: 36.6, sd: 14.3
## mean of age in baked training: 0, sd: 1

glue("mean of age in orig testing: {format(mean(ipf_te$age), digits = 1)}  
      mean of age in baked testing: {format(mean(ipf_te2$age), digits = 1)}
```



```
## mean of age in orig testing: 36.6, sd: 14.3
## mean of age in baked testing: 0, sd: 1
```

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$age), digits = 1)}")
     mean of age in baked training: {format(mean(ipf_tr2$age), digits = 1)}

## mean of age in orig training: 36.6, sd: 14.3
## mean of age in baked training: 0, sd: 1

glue("mean of age in orig testing: {format(mean(ipf_te$age), digits = 1)}")
     mean of age in baked testing: {format(mean(ipf_te2$age), digits = 1)}

## mean of age in orig testing: 36.6, sd: 14.3
## mean of age in baked testing: 0, sd: 1
```

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  42890
## 2   51  0.00398  39080
## 3   51  0.00436  35610
## 4   51  0.00478  32450
## 5   51  0.00525  29560
## 6   51  0.00576  26940
## 7   51  0.00632  24540
## 8   51  0.00693  22360
## 9   51  0.00760  20380
## 10  51  0.00833  18570
## 11  51  0.00914  16920
## 12  51  0.01002  15410
## 13  51  0.01098  14050
## 14  51  0.01204  12800
## 15  51  0.01320  11660
## 16  51  0.01447  10620
## 17  51  0.01586   9681
## 18  51  0.01737   8821
```

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: 50ms
## Call: glmnet::glmnet(x = as.matrix(x), y = y, family = "gaussian",
##
##          Df      %Dev Lambda
## 1    0 0.00000 42.890
## 2    1 0.08402 39.080
## 3    2 0.16380 35.610
## 4    2 0.24350 32.450
## 5    2 0.30960 29.560
## 6    2 0.36450 26.940
## 7    2 0.41000 24.540
## 8    2 0.44790 22.360
## 9    2 0.47930 20.380
## 10   2 0.50540 18.570
## 11   2 0.52700 16.920
## 12   2 0.54500 15.410
## 13   2 0.55990 14.050
## 14   2 0.57230 12.800
```

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

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

```
## parsnip model object  
##  
## Fit in: 720msRanger 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):     560.9305  
##   R squared (OOB):                 0.8491322
```

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 42893.  2.58e-36
## 2 sexM                 1  8.47e-35 42893.  2.58e-36
## 3 eventSB               1 -8.25e-36 42893.  2.58e-36
## 4 eventSBD               1 -2.53e-35 42893.  2.58e-36
## 5 equipmentSingle-ply     1  2.93e-35 42893.  2.58e-36
## 6 age                   1 -5.11e-37 42893.  2.58e-36
## 7 divisionJuniors        1 -5.96e-36 42893.  2.58e-36
## 8 divisionLight            1 -1.34e-35 42893.  2.58e-36
## 9 divisionMasters 1       1 -9.26e-37 42893.  2.58e-36
## 10 divisionMasters 2      1 -1.42e-35 42893.  2.58e-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
```

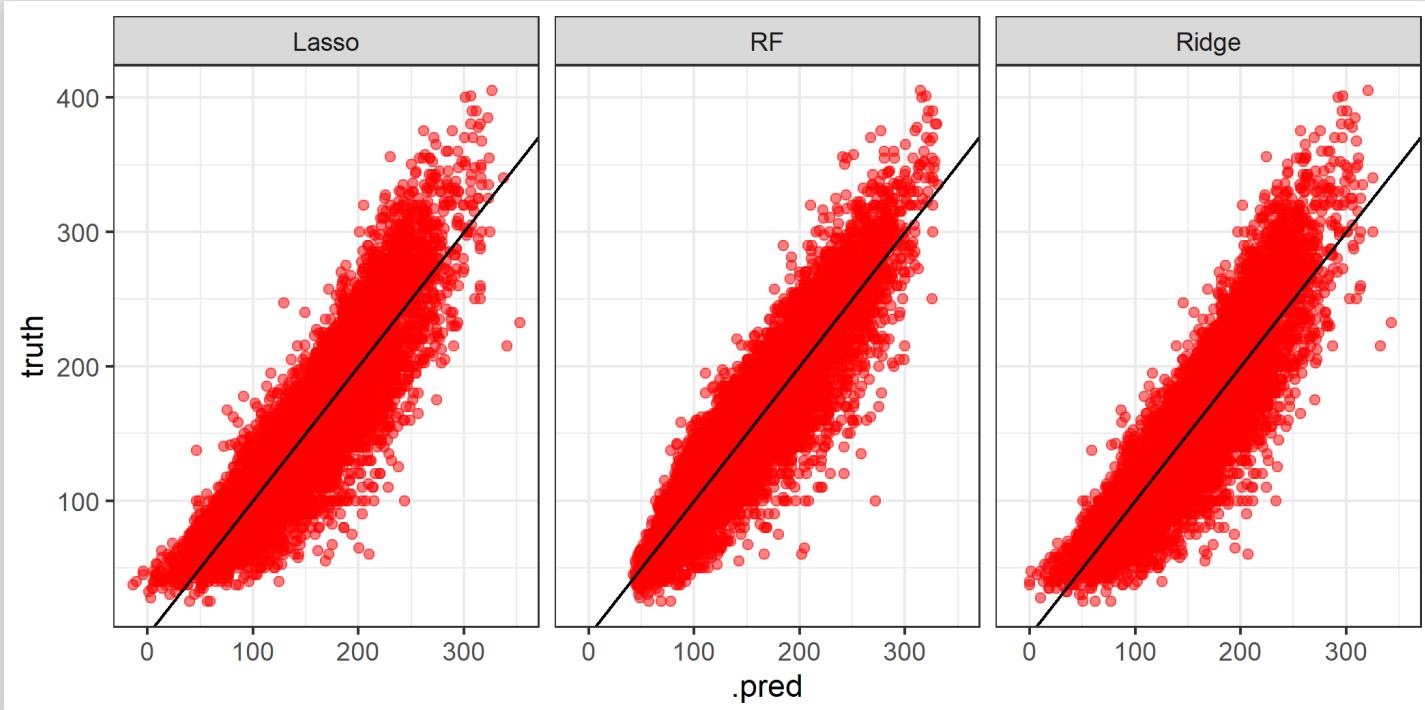
Comparing Models

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     26.9
```

```
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

Define your model spec, using `tune()` from the `tune` package for a parameter you wish to tune:

```
library(tune)

mod_rf_spec <- rand_forest(mode = "regression",
                           mtry = tune(),
                           min_n = tune(),
                           trees = 100) %>%
  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]>
```

```
tune_res$.metrics[[1]]
```

```
## # A tibble: 15 x 5
##       mtry min_n .metric .estimator .estimate
##       <int> <int> <chr>   <chr>        <dbl>
## 1      2     10  rmse    standard     25.4
## 2      4     10  rmse    standard     24.4
## 3      6     10  rmse    standard     24.6
## 4      8     10  rmse    standard     24.8
## 5     10     10  rmse    standard     24.9
## 6      2     20  rmse    standard     25.4
## 7      4     20  rmse    standard     24.3
## 8      6     20  rmse    standard     24.4
## 9      8     20  rmse    standard     24.5
## 10    10     20  rmse    standard     24.6
## 11    2      30  rmse    standard     25.5
## 12    4      30  rmse    standard     24.2
## 13    6      30  rmse    standard     24.3
## 14    8      30  rmse    standard     24.4
## 15    10     30  rmse    standard     24.4
```

Collect the mean metric across folds:

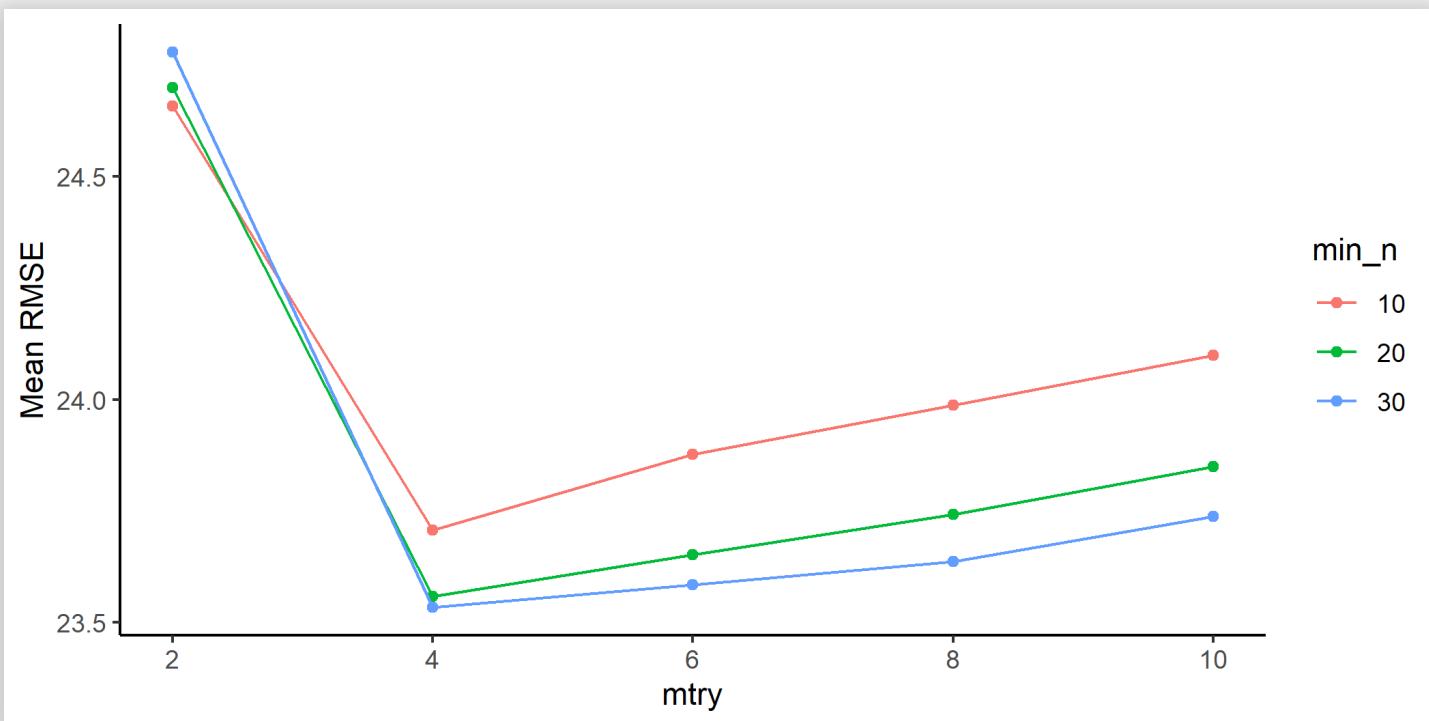
```
estimates <- collect_metrics(tune_res)
```

```
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 parameter:

```
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()
```



There are of course also methods for helping us choose best params and final model.

```
best_rmse <- tune_res %>% select_best(metric = "rmse", maximize =  
best_rmse
```

```
## # A tibble: 1 x 2  
##   mtry min_n  
##   <int> <int>  
## 1     4     30
```

See also `?select_by_one_std_err`.

```
mod_rf_final <- finalize_model(mod_rf_spec, best_rmse)  
mod_rf_final
```

```
## Random Forest Model Specification (regression)  
##  
## Main Arguments:  
##   mtry = 4  
##   trees = 100  
##   min_n = 30  
##  
## Computational engine: ranger
```

```
mod_rf_final %>%
  fit(best3bench_kg ~ ., data = ipf_tr) %>%
  predict(new_data = ipf_te) %>%
  mutate(truth = ipf_te$best3bench_kg)
```

```
## # A tibble: 12,818 x 2
##       .pred truth
##   <dbl> <dbl>
## 1     62.5 62.5
## 2     63.0  60
## 3     65.3  65
## 4     65.9  62.5
## 5     72.5  67.5
## 6     70.3  52.5
## 7     81.0  82.5
## 8     77.0  72.5
## 9     76.5  62.5
## 10    80.9  105
## # ... with 12,808 more rows
```

infer: Tidy Statistics

APPLICATIONS



OF DATA SCIENCE

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 = r
```

```
## # 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 method
##       <dbl>      <dbl>     <dbl>    <dbl>      <dbl>      <dbl>      <dbl> <chr>
## 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
##   estimate1 estimate2 statistic p.value parameter conf.low conf.high method
##       <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

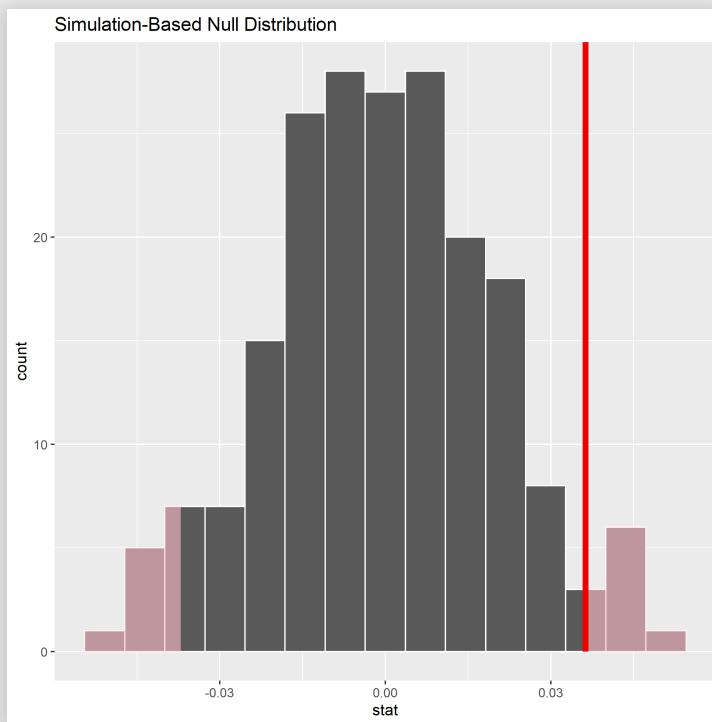
```
diff_null_perm <- ipf_lifts %>%
  filter(year == 2019) %>%
  specify(equipment ~ sex, success = "Raw") %>%
  hypothesize(null = "independence") %>%
  generate(reps = 200, type = "permute") %>%
  calculate(stat = "diff in props", order = c("F", "M"))

diff_null_perm
```

```
## # A tibble: 200 x 2
##       replicate      stat
##       <int>     <dbl>
## 1          1  0.00575
## 2          2  0.0363
## 3          3 -0.00645
## 4          4  0.0118
## 5          5 -0.0532
## 6          6  0.00982
## 7          7  0.00168
## 8          8 -0.0105
## 9          9  0.0403
## 10         10  0.0200
## # ... 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 0.1
```

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

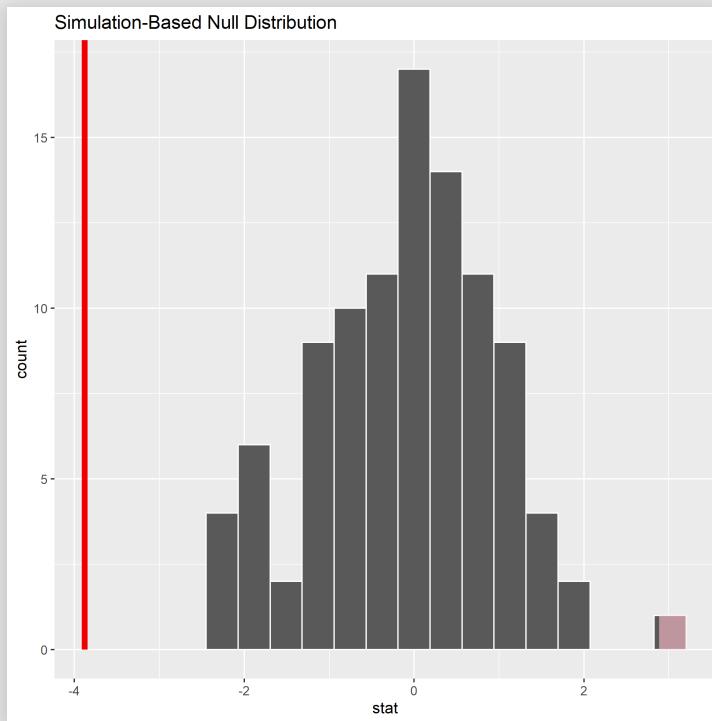
```
t_null_perm <- ipf_lifts %>%
  filter(year == 2019) %>%
  specify(age ~ sex) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 100, type = "permute") %>%
  calculate(stat = "t", order = c("F", "M"))

t_null_perm
```

```
## # A tibble: 100 x 2
##       replicate   stat
##           <int>   <dbl>
## 1             1 -0.658
## 2             2 -0.923
## 3             3  0.0529
## 4             4  1.67 
## 5             5  0.802
## 6             6  1.27 
## 7             7  1.20 
## 8             8  1.31 
## 9             9  0.0975
## 10            10 -2.20
## # ... 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
```