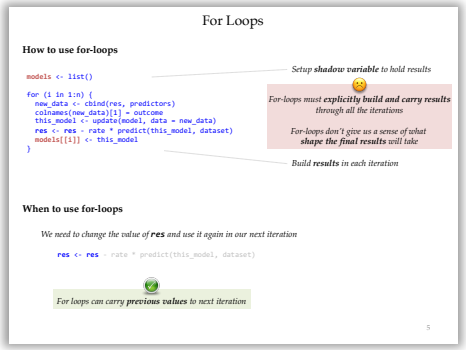
Business Analytics Using Computational Statistics

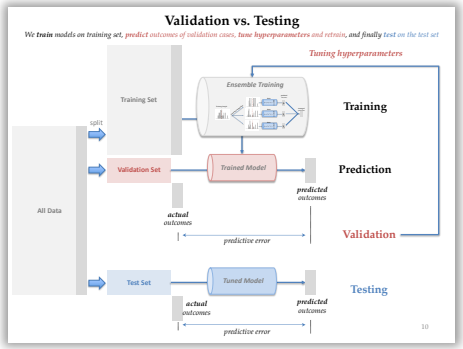
Ensemble Models

Validation and Looking Ahead

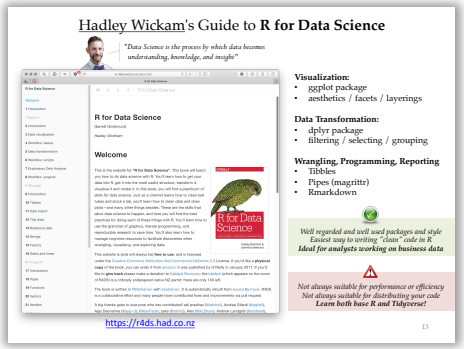
Loops Revisited



Validation and Testing



Looking Ahead…



Insurance Dataset

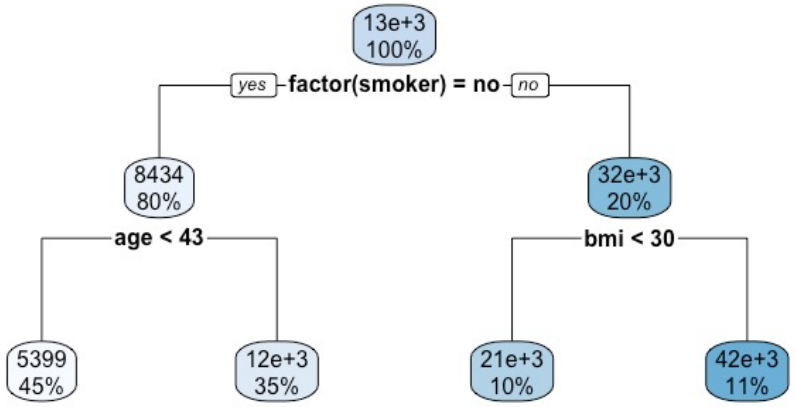
Dataset

```
insurance <- read.csv("insurance.csv") |> na.omit()
str(insurance)
'data.frame':1338 obs. of  7 variables:
 $ age     : int  19 18 28 33 32 31 46 37 37 60 ...
 $ sex     : chr  "female" "male" "male" "male" ...
 $ bmi     : num  27.9 33.8 33 22.7 28.9 ...
 $ children: int  0 1 3 0 0 0 1 3 2 0 ...
 $ smoker  : chr  "yes" "no" "no" "no" ...
 $ region  : chr  "southwest" "southeast" "southeast" "northwest" ...
 $ charges : num  16885 1726 4449 21984 3867 ...
```

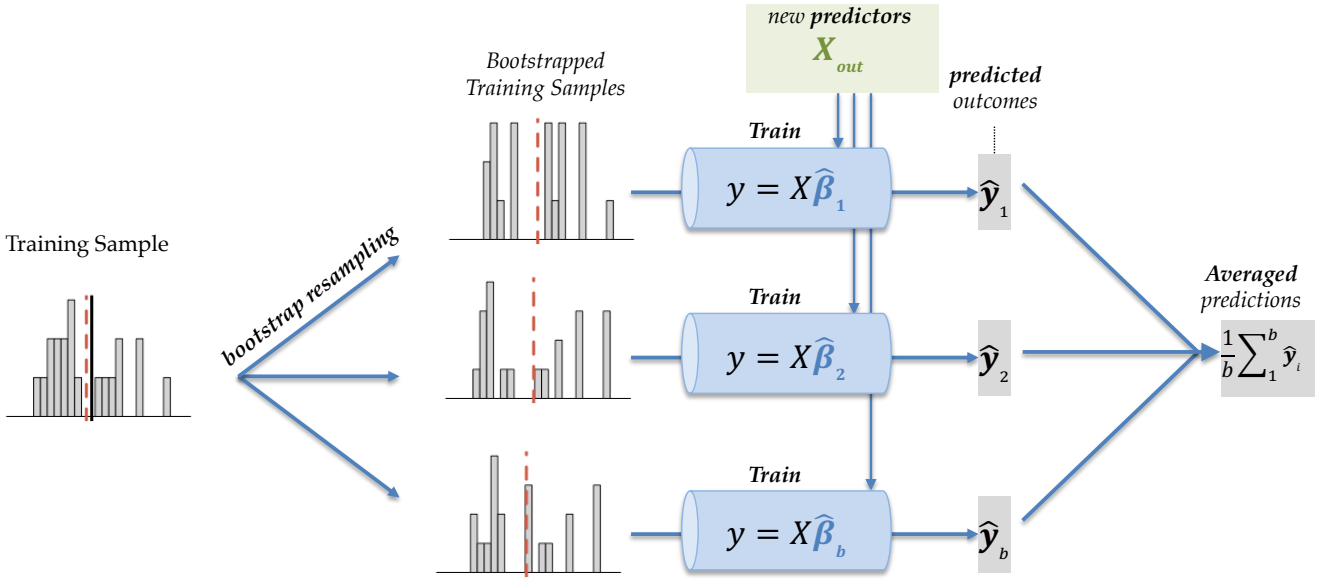
Models

```
charges_formula <- charges ~ age + factor(sex) + bmi + children + factor(smoker) + factor(region)

lm_model <- lm(charges_formula, data=insurance)
Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)        -11938.5      987.8 -12.086  < 2e-16 ***
age                   256.9       11.9  21.587  < 2e-16 ***
factor(sex)male      -131.3      332.9  -0.394 0.693348
bmi                   339.2       28.6  11.860  < 2e-16 ***
children              475.5      137.8   3.451 0.000577 ***
factor(smoker)yes   23848.5      413.1  57.723  < 2e-16 ***
factor(region)northwest  -353.0      476.3  -0.741 0.458769
factor(region)southeast -1035.0      478.7  -2.162 0.030782 *
factor(region)southwest  -960.0      477.9  -2.009 0.044765 *

tree_model <- rpart(charges_formula, data=insurance)
rpart.plot(tree_model)
```

Should we be interested in the inferential results?



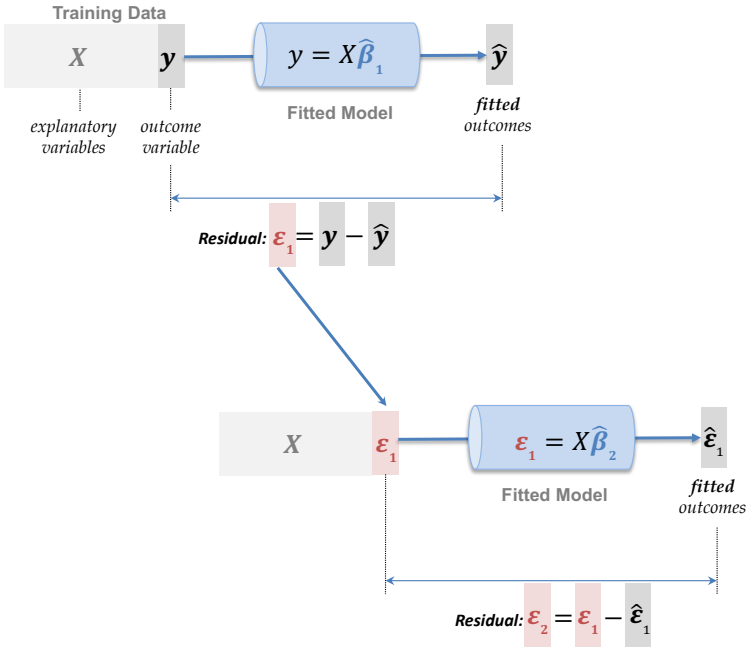
Implementing Bagging




```r
bagged_learn <- function(model, dataset, b=100) {
  lapply(1:b, \(i) {
    data_i <- dataset[sample(nrow(dataset), replace=TRUE),]
    update(model, data=data_i)
  })
}
```

```r
bagged_predict <- function(bagged_models, new_data) {
  predictions <- lapply(bagged_models, \(m) predict(m, new_data))
  as.data.frame(predictions) |> apply(FUN=mean, MARGIN=1)
}
```

Implementing Boosting




```
boost_learn <- function(model, dataset, outcome, n=100, rate=0.1) {
  predictors <- dataset[,-which(names(dataset) == outcome)]

  # Initialize residuals and models
  res <- dataset[,outcome] # actuals
  models <- list()

  for (i in 1:n) {
    new_data <- cbind(res, predictors)
    colnames(new_data)[1] = outcome
    this_model <- update(model, data = new_data)
    res <- res - rate * predict(this_model, dataset)
    models[[i]] <- this_model
  }

  list(models=models, rate=rate)
}
```

$$\varepsilon = \varepsilon - \alpha\hat{y}$$

```
boost_predict <- function(boosted_learning, new_data) {
  boosted_models <- boosted_learning$models
  rate <- boosted_learning$rate

  predictions <- lapply(boosted_models, \(this_model) {
    predict(this_model, new_data)
  })

  pred_frame <- as.data.frame(predictions) |> unname()
  apply(pred_frame, FUN = \(preds) rate * sum(preds), MARGIN=1)
}
```

$$\sum_{i=1}^{\nu} \alpha f_i(X_{out}) = \alpha \sum_{i=1}^{\nu} f_i(X_{out})$$

Functional Iteration

produces a list

```
lapply(bagged_models, \(model) predict(model, new_data))
```

Apply functions:
- *Produce **predictable output structures***
- *Require **little code** to implement*
- *Can be **more readable***

processes one dimension of a matrix (e.g., row or column)

```
apply(pred_frame, FUN = \(preds) { rate * sum(preds) }, MARGIN=1)
```

See also: `purrr:map()`

https://purrr.tidyverse.org/reference/map.html

Why would we use any other form of iteration?

For Loops

How to use for-loops

```
models <- list()

for (i in 1:n) {
  new_data <- cbind(res, predictors)
  colnames(new_data)[1] = outcome
  this_model <- update(model, data = new_data)
  res <- res - rate * predict(this_model, dataset)
  models[[i]] <- this_model
}
```

*Setup **shadow variable** to hold results*

*For-loops must **explicitly build and carry results** through all the iterations*

*For-loops don't give us a sense of what **shape the final results** will take*

*Build **results** in each iteration*

When to use for-loops

*We need to change the value of **res** and use it again in our next iteration*

```
res <- res - rate * predict(this_model, dataset)
```

*Use For loops when you need to carry **previous values** to next iteration*

Testing our Predictive Models

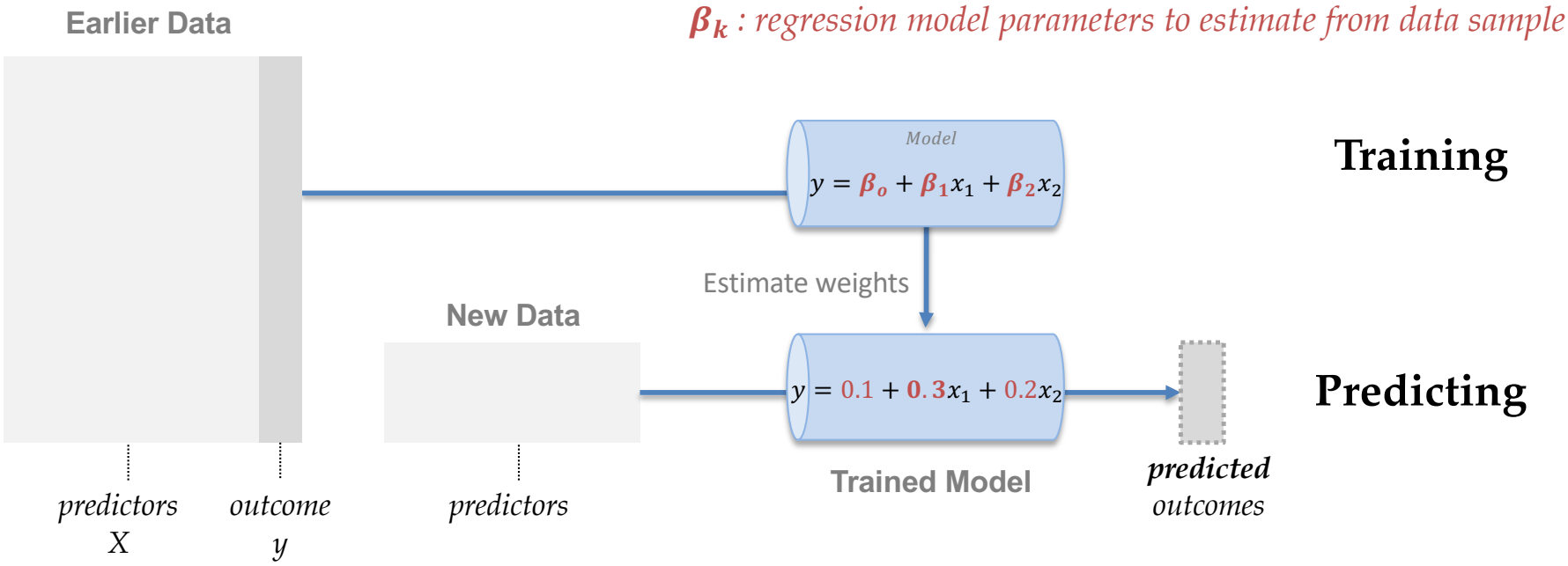
```
# K-FOLD CROSS VALIDATION
k_fold_rmse(lm_model, insurance, "charges")
 is      oos
6041.680 6087.388

k_fold_rmse(tree_model, insurance, "charges")
 is      oos
5029.781 5135.175


# SPLIT SAMPLE TESTING OF ENSEMBLE MODELS
set.seed(423079)
train_indices <- sample(1:nrow(insurance), 0.80*nrow(insurance))
train_set <- insurance[train_indices,]
test_set <- insurance[-train_indices,]

bagged_learn(lm_model, train_set) |>
  bagged_predict(test_set) |>
  rmse_out(test_set$charges, preds = _)  # [1] 6381.847

bagged_learn(tree_model, train_set) |>
  bagged_predict(test_set) |>
  rmse_out(test_set$charges, preds = _)  # [1] 4974.225

boost_learn(lm_model, train_set, outcome="charges") |>
  boost_predict(test_set) |>
  rmse_out(test_set$charges, preds = _)  # [1] 6380.757

boost_learn(tree_model, train_set, outcome="charges") |>
  boost_predict(test_set) |>
  rmse_out(test_set$charges, preds = _)  # [1] 4751.576
```

Bagged methods do not work well for stable learners
Boosted methods do not work well for strong methods

Bagged methods seems to favor unstable learners
Boosted methods seem to favor weak methods

Parameters

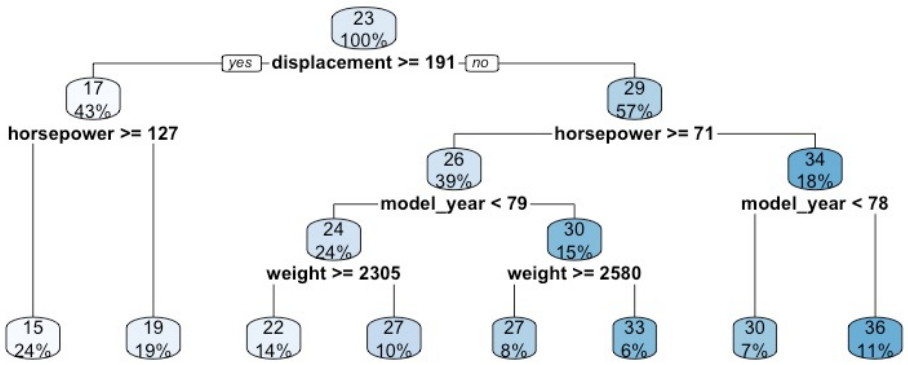
values estimated during training



Hyperparameters

*values used to **tune** the learning process*

manually adjust or change to get better results



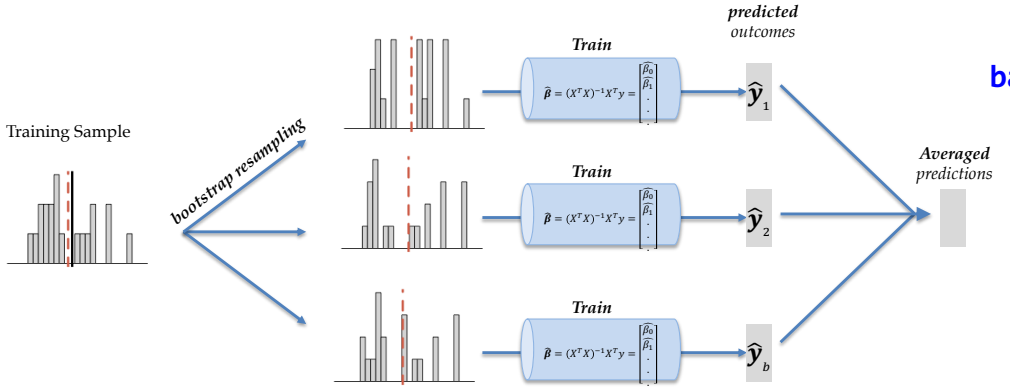
***max_depth**: how many levels of children a tree can have*

***min_samples_split**: the least number of cases a node must have to be considered for splitting*

***max_features**: the most number of features from dataset that can be used to generate the tree*

Hyperparameters in Ensemble Models

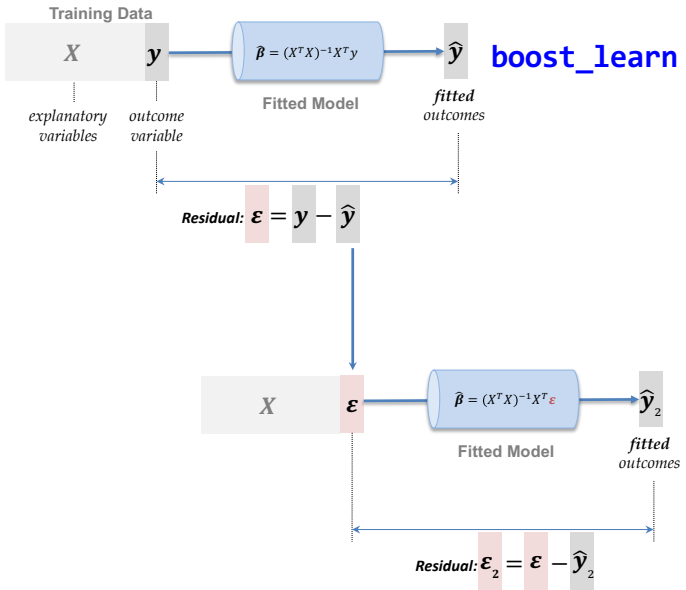
Bagged Models




```
bagged_learn <- function(model, dataset, b=100) {…}
```

b: *Number of bootstrapped samples*

Boosted Models



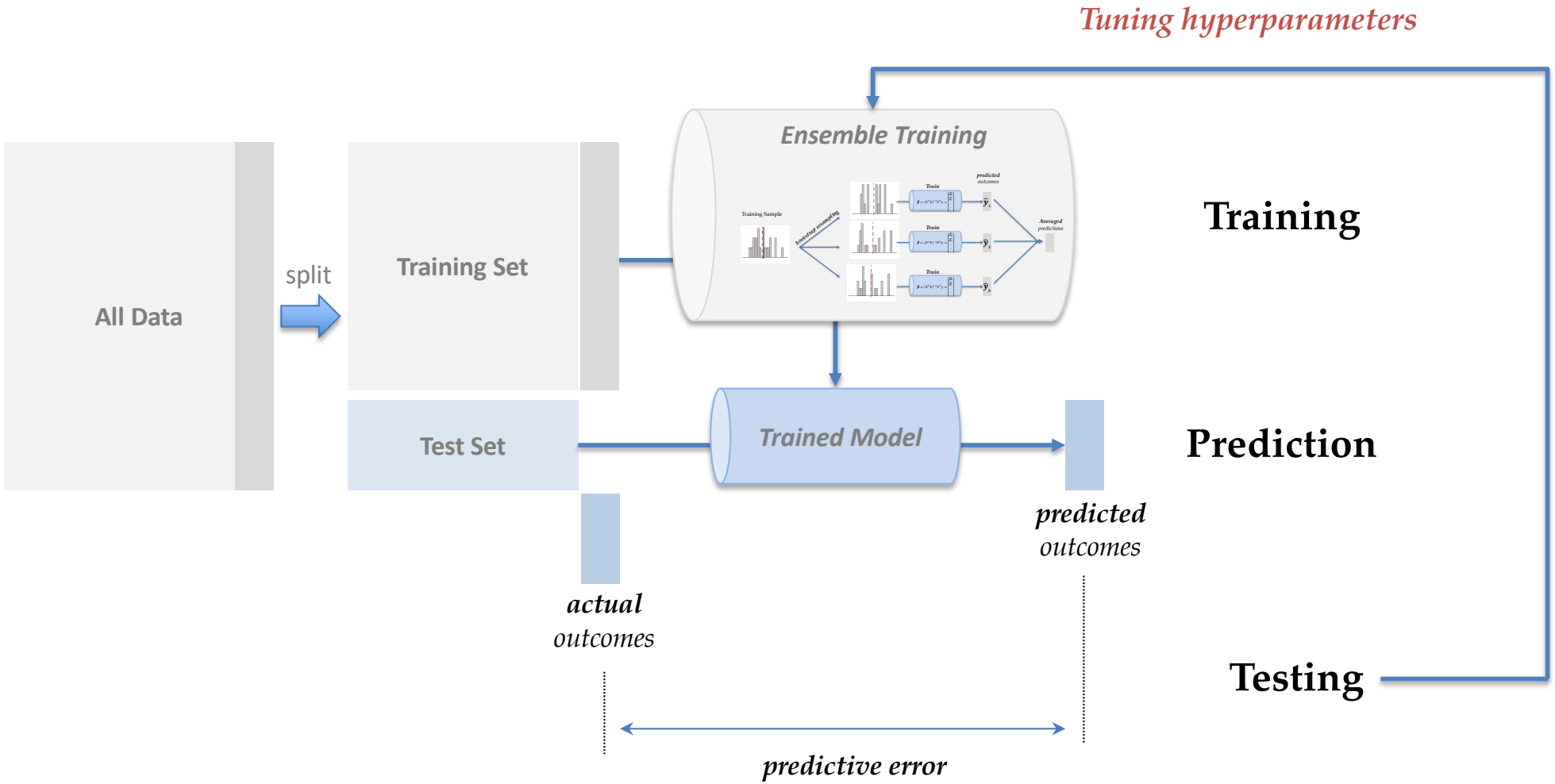

```
boost_learn <- function(model, dataset, outcome, n=100, rate=0.1) {…}
```

rate: *learning rate*

n: *number of rounds*

Risks in Tuning Hyperparameters

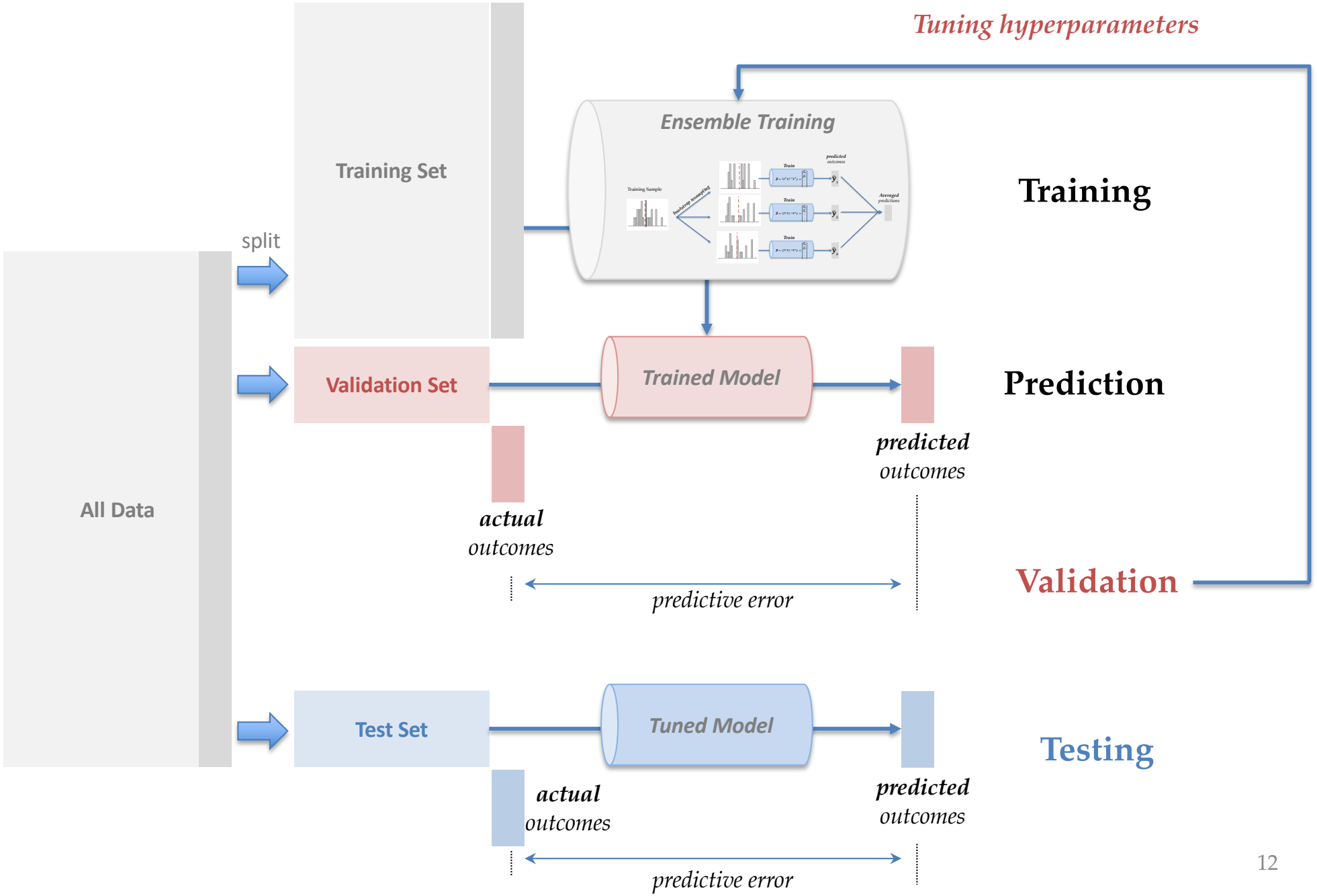
*Consider if we **train** models on training set, **predict** outcomes of test set, and then **tune hyperparameters** and retrain*



We risk overfitting to the test set!
Our estimated predictive performance ($RMSE_{out}$) will be too optimistic (too low)

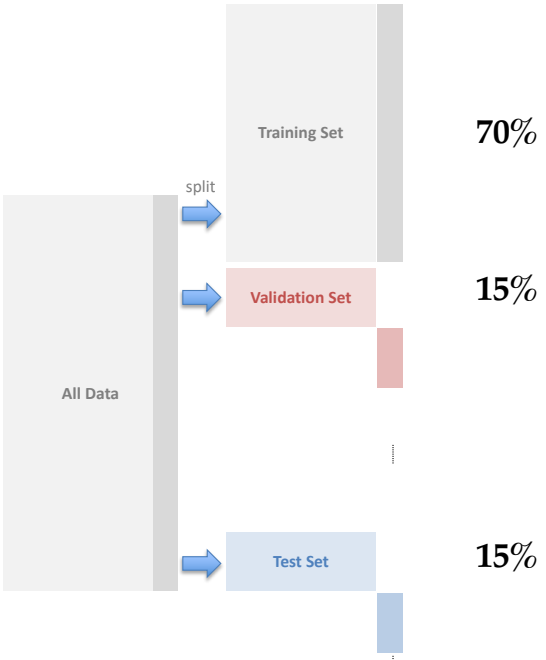
Validation vs. Testing

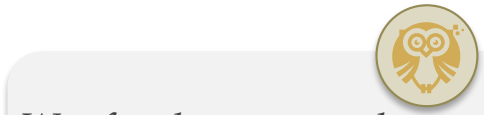
*We **train** models on training set, **predict** outcomes of validation cases, **tune hyperparameters** and retrain, and finally **test** on the test set*



Hyperparameter Optimization

```
# 70:15:15 HYPERPARAMETER VALIDATION SETUP
set.seed(472638912)
n <- nrow(insurance)
rand_indices <- sample(1:n)
n70 <- round(0.70*n, 0)
n85 <- round(0.85*n, 0)
train_set <- insurance[rand_indices[1:n70],]
validate_set <- insurance[rand_indices[(n70 + 1):n85],]
test_set <- insurance[rand_indices[(n85 + 1):n],]
```





We often have to search a space of hyperparameters to find the optimal set of values that gives us the best predictive performance

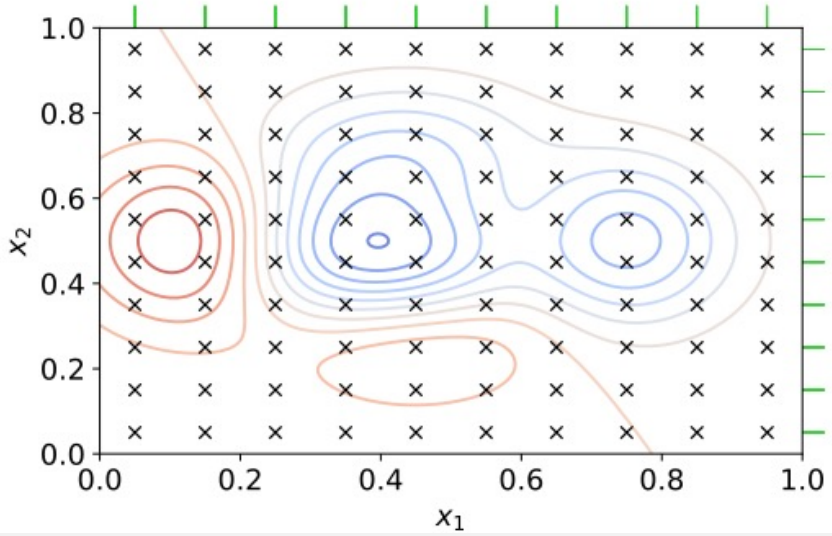
Hyperparameter optimization methods:

- ***Manual Search***
- ***Grid Search***

} *We will try these methods*

- Random Sampling
- Evolutionary Algorithms
- Gradient-based optimization
- etc.

image: Wikipedia (hyperparameter optimization)



Two define a surface with varying predictive performance

Hyperparameter tuning for Bagging

```
rebagging <- function(stump, train_data, validate_data) {
  bagged_learn(stump, train_data) |>
    bagged_predict(validate_data) |>
    rmse_out(validate_data$charges, preds = _)
}

set.seed(76534235)

# baseline untuned parameter comparison
stump <- update(tree_model, cp=0, maxdepth=1)
rebagging(stump, train_set, test_set)                  # [1] 7882.932

# find winning hyperparameters using validation
stump1 <- update(tree_model, cp=0, maxdepth=1)
rebagging(stump1, train_set, validate_set)             # [1] 7403.726

stump2 <- update(tree_model, cp=0, maxdepth=2)
rebagging(stump2, train_set, validate_set)             # [1] 4668.749

stump3 <- update(tree_model, cp=0, maxdepth=3)
rebagging(stump3, train_set, validate_set)             # [1] 4223.991

stump4 <- update(tree_model, cp=0, maxdepth=4)
rebagging(stump4, train_set, validate_set)             # [1] 4154.712

stump5 <- update(tree_model, cp=0, maxdepth=5)
rebagging(stump5, train_set, validate_set)             # [1] 4132.281

stump6 <- update(tree_model, cp=0, maxdepth=6)
rebagging(stump6, train_set, validate_set)             # [1] 4156.272

# test winning hyperparameters on test set
rebagging(stump5, train_set, test_set)                 # [1] 4651.94
```

Manual Search

we use our own intuition to search a set of hyperparamters

Our tuned hyperparameters give us a better predictions!

Why are is the final test performance worse than the validation performance?

Hyperparameter tuning for Boosting

```
reboosting <- function(stump, train_data, validate_data, rate) {
  boost_learn(stump, train_data, outcome="charges", rate = rate) |>
    boost_predict(validate_data) |>
    rmse_out(validate_data$charges, preds = _)
}

# untuned parameter comparison
boost_stump <- update(tree_model, cp=0, maxdepth=1)
reboosting(boost_stump, train_set, test_set, rate=0.10)  # [1] 6339.255

# find winning hyperparameters using validation
boost_stump <- update(tree_model, cp=0, maxdepth=2)
reboosting(boost_stump, train_set, validate_set, rate=0.01)
reboosting(boost_stump, train_set, validate_set, rate=0.05)
reboosting(boost_stump, train_set, validate_set, rate=0.10)
reboosting(boost_stump, train_set, validate_set, rate=0.15)

boost_stump <- update(tree_model, cp=0, maxdepth=3)
reboosting(boost_stump, train_set, validate_set, rate=0.01)
reboosting(boost_stump, train_set, validate_set, rate=0.05)
reboosting(boost_stump, train_set, validate_set, rate=0.10)
reboosting(boost_stump, train_set, validate_set, rate=0.15)

boost_stump <- update(tree_model, cp=0, maxdepth=4)
reboosting(boost_stump, train_set, validate_set, rate=0.01)
reboosting(boost_stump, train_set, validate_set, rate=0.05)
reboosting(boost_stump, train_set, validate_set, rate=0.10)
reboosting(boost_stump, train_set, validate_set, rate=0.15)

# test winning hyperparameters on test set
boost_stump <- update(tree_model, cp=0, maxdepth=3)
reboosting(boost_stump, train_set, test_set, rate=0.05)  # [1] 4616.174
```

Grid Search

we systematically try equally spaced sets of hyperparameter values

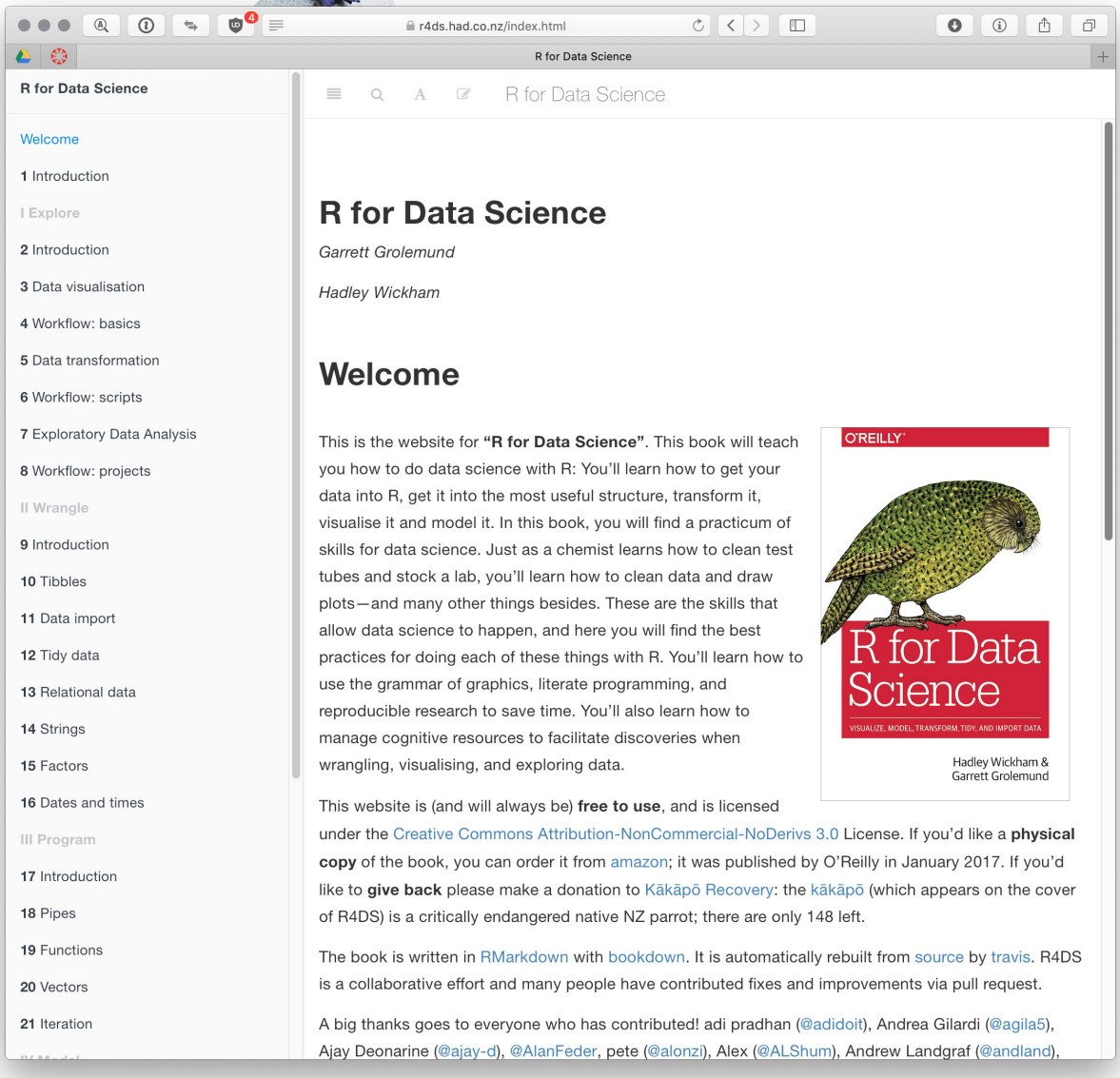
maxdepth	learning rate 0.01	0.05	0.10	0.15
2	7446.568	4215.605	4178.164	4198.649
3	7264.562	**4132.652**	4181.78	4223.391
4	7201.843	4191.672	4265.846	4291.165

Our tuned hyperparameters give us a better predictions!

Where to next?

Hadley Wickam's Guide to **R for Data Science**

"Data Science is the process by which data becomes understanding, knowledge, and insight"



https://r4ds.had.co.nz

Visualization:

- ggplot package
- aesthetics / facets / layerings

Data Transformation:

- dplyr package
- filtering / selecting / grouping

Wrangling, Programming, Reporting

- Tibbles
- Pipes (magrittr)
- Rmarkdown/Quarto

Well regarded and well used packages and style
Easiest way to writing "clean" code in R
Ideal for analysts working on business data

Not always suitable for performance or efficiency
Not always suitable for distributing your code
Learn both base R and Tidyverse!

Interactive Visualizations:
From Manipulate to Shiny

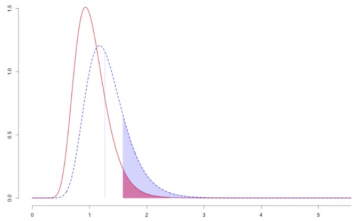
manipulate()

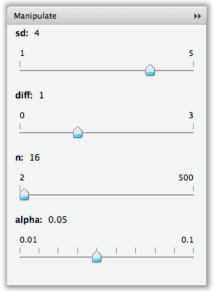
Create a User Interface to Interact with Visualizations

```
install.packages("manipulate")      Install new package

library(manipulate)

                                    Your function to plot visualization
manipulate(
  t.test.plot(diff, sd, n, alpha),
  diff  = slider(0, 3, step = 0.1, initial = 1),
  sd    = slider(1, 5, step = 0.1, initial = 4),
  n     = slider(2, 500, step = 1, initial = 16),
  alpha = slider(0.01, 0.1, step = 0.01, initial = 0.05)
)
```

UI controller components

Define your parameters as UI components

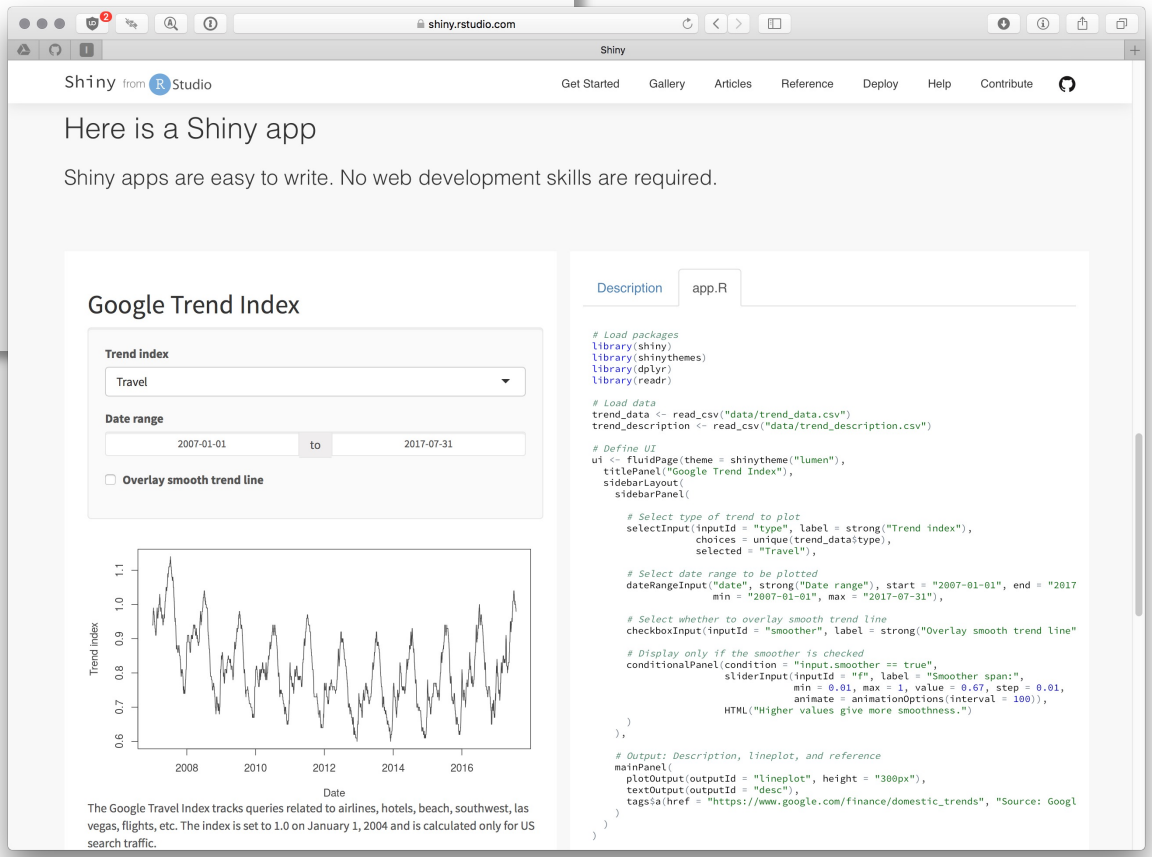




Manipulate: Make local interactive visualizations

Shiny: https://shiny.rstudio.com

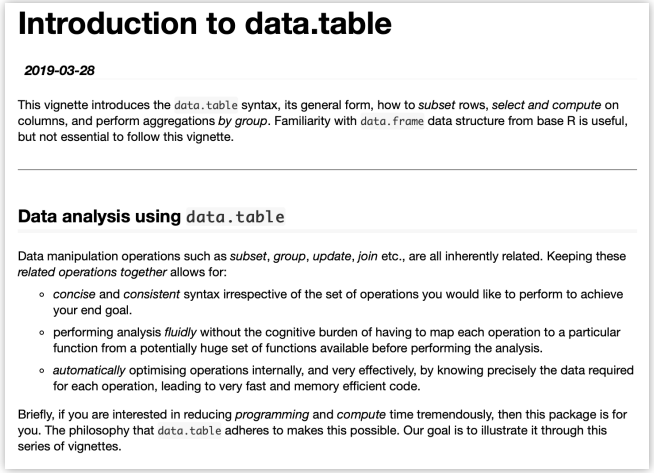
Make web-based interactive visualizations



High Performance: data.table for loading, aggregating

`data.table` – *package for high performance read/writes and computation*

Introduction



Introduction to data.table

2019-03-28

This vignette introduces the `data.table` syntax, its general form, how to *subset* rows, *select and compute* on columns, and perform aggregations *by group*. Familiarity with `data.frame` data structure from base R is useful, but not essential to follow this vignette.

Data analysis using `data.table`

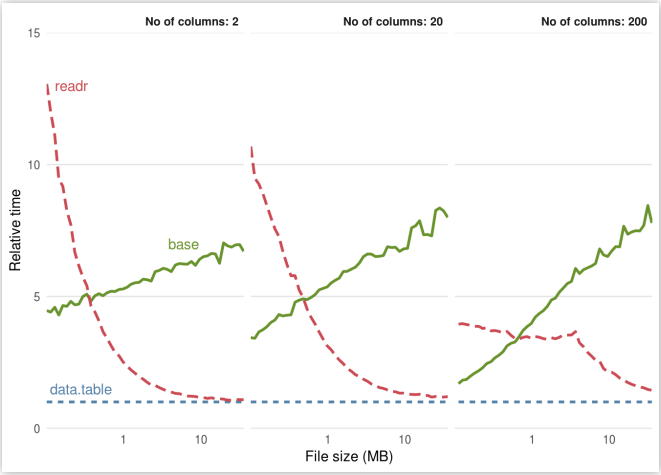
Data manipulation operations such as *subset*, *group*, *update*, *join* etc., are all inherently related. Keeping these *related operations together* allows for:

- *concise* and *consistent* syntax irrespective of the set of operations you would like to perform to achieve your end goal.
- performing analysis *fluidly* without the cognitive burden of having to map each operation to a particular function from a potentially huge set of functions available before performing the analysis.
- *automatically* optimising operations internally, and very effectively, by knowing precisely the data required for each operation, leading to very fast and memory efficient code.

Briefly, if you are interested in reducing *programming* and *compute* time tremendously, then this package is for you. The philosophy that `data.table` adheres to makes this possible. Our goal is to illustrate it through this series of vignettes.

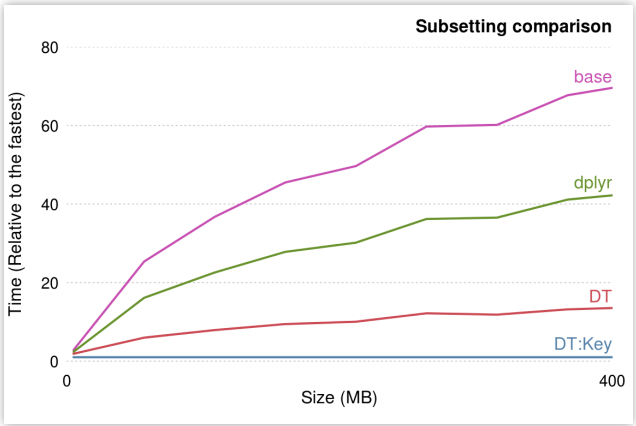
https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html

Fast Reads



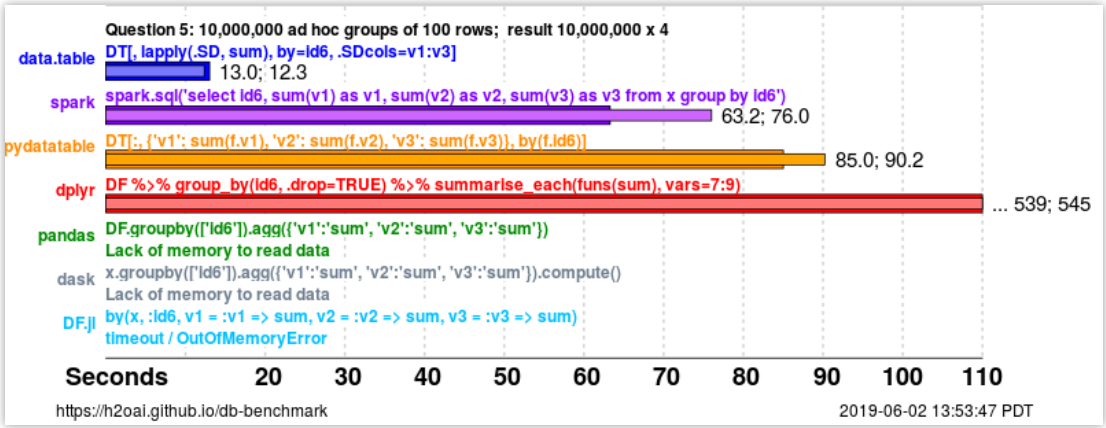
https://csgillespie.github.io/efficientR/input-output.html#top-5-tips-for-efficient-data-io

Fast Subsetting



https://csgillespie.github.io/efficientR/data-processing-with-data-table.html

Fast Aggregation and Summary



https://h2oai.github.io/db-benchmark/

High Performance: Sparse Matrices

```
# native matrix: Create sparse matrices
random_x <- sample(1:2000, 100)
random_y <- sample(1:2000, 100)

m1 <- matrix(0, nrow = 2000, ncol = 2000)
m1[random_x, random_y] <- 1

object.size(m1)            # 32000200 bytes
system.time( m1 %*% m1 )   # 5.397 seconds
```

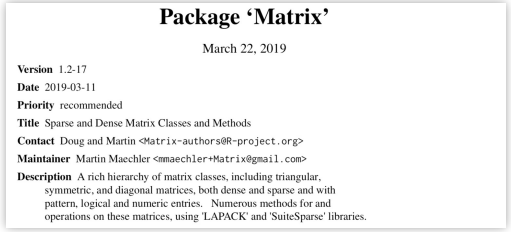
Large matrix operations (e.g., cosine similarity)
(a) take a lot of memory!
(b) take a lot of time!

Matrix – *package for high performance sparse matrix computations*

```
# Matrix package: Create sparse matrix using
library(Matrix)
m2 <- Matrix(0, nrow = 2000, ncol = 2000, sparse = TRUE)
m2[random_x, random_y] <- 1

object.size(m2)            # 129424 bytes
system.time( m2 %*% m2 )   # 0.001 seconds
```

We can take advantage of sparsity (emptiness) in matrices:
(a) takes a lot less memory!
(b) take a lot less time!

Works with many modeling packages like glmnet



Package 'Matrix'

March 22, 2019

Version 1.2-17
Date 2019-03-11
Priority recommended
Title Sparse and Dense Matrix Classes and Methods
Contact Doug and Martin <Matrix-authors@R-project.org>
Maintainer Martin Maechler <mmaechler+Matrix@gmail.com>
Description A rich hierarchy of matrix classes, including triangular, symmetric, and diagonal matrices, both dense and sparse and with pattern, logical and numeric entries. Numerous methods for and operations on these matrices, using 'LAPACK' and 'SuiteSparse' libraries.

https://cran.r-project.org/package=Matrix

John Myles White

"Who refuses to do arithmetic is doomed to talk nonsense."

Browse: Home / 2011 / October / 31 / Using Sparse Matrices in R

Using Sparse Matrices in R

By John Myles White on 10.31.2011

http://www.johnmyleswhite.com/notebook/2011/10/31/using-sparse-matrices-in-r/

Parallel Processing

Multi-core CPU and Distributed Computing

http://www.win-vector.com/blog/2016/01/parallel-computing-in-r/

```
lm_data <- cbind(y=rnorm(400), as.data.frame(replicate(10, rnorm(400))))

boot_lm = function(i, dat) {
  boot_dat = dat[sample(nrow(dat), replace=TRUE), ]
  lm(y~., data=boot_dat)$coefficients
}

system.time( sapply(1:6000, boot_lm, dat=lm_data) )
```

This code runs on a single core of your CPU
It is not using all the processing power of your machine

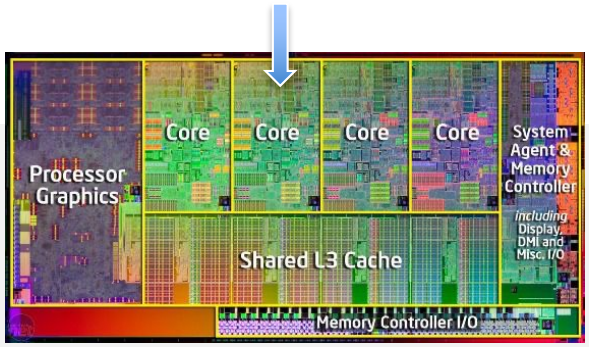


Image: https://www.cnet.com/news/what-became-of-multi-core-programming-problems/

We can parallelize our work over the multiple cores of our CPU

Parallel computing is a type of computation in which many calculations are carried out simultaneously."

1. Find out how many cores are available on your CPU
2. Allocate cores into a work cluster for processing work
3. Give the cluster your local variables and environment with libraries loaded
4. Run parallelized equivalent of apply family function
5. Free up cluster for other tasks (important!)

```
library(parallel)

num_cores <- detectCores()

cluster <- makeCluster(num_cores)

clusterExport(cl = cluster,
              varlist = c("lm_data"),
              envir = environment())

system.time( parSapply(cluster, 1:3000, boot_lm, dat=lm_data) )

stopCluster(cluster)
```

Parallelizing across machines: http://www.win-vector.com/blog/2016/01/running-r-jobs-quickly-on-many-machines/
https://www.r-bloggers.com/running-r-jobs-quickly-on-many-machines/

RStudio Server

Download RStudio Server v1.4.1717

RStudio Server enables you to provide a browser based interface to a version of R running on a remote Linux server, bringing the power and productivity of the RStudio IDE to server-based deployments of R.

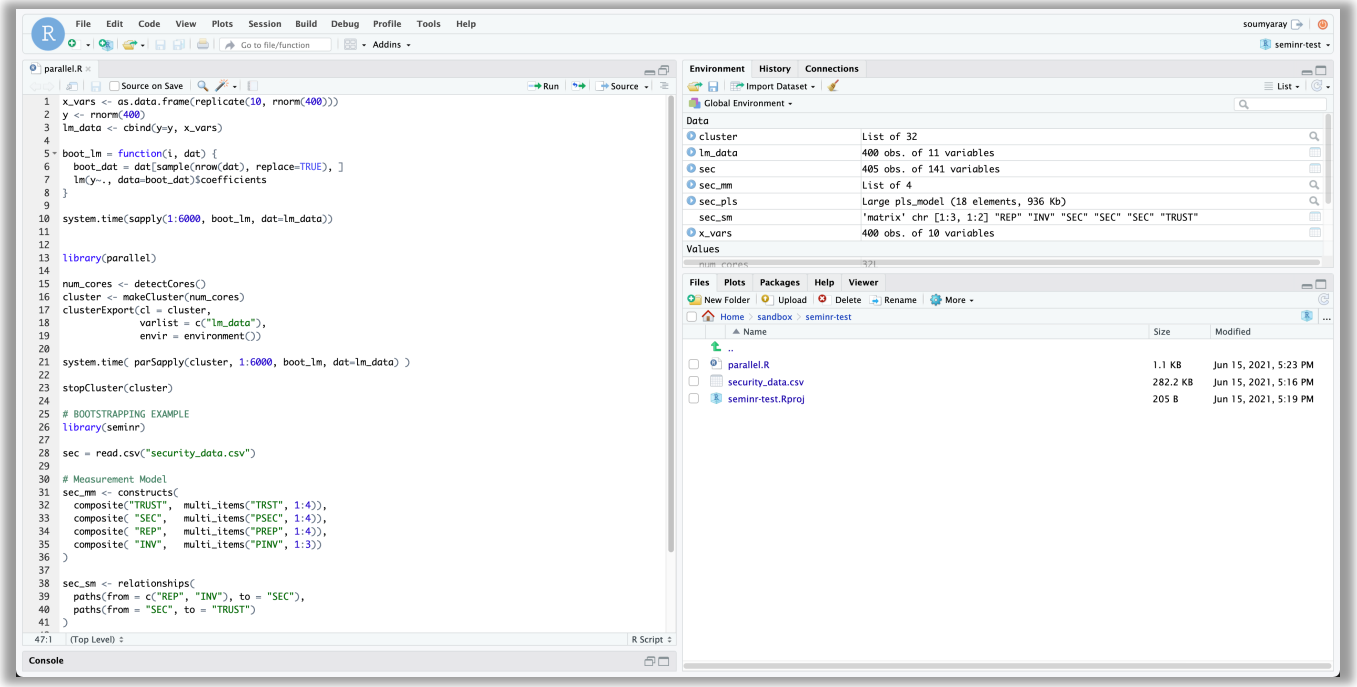
Do you need support or a commercial license? Compare our commercial and open source products.

Managing Packages

To install packages on Linux faster and easier, consider RStudio Package Manager.

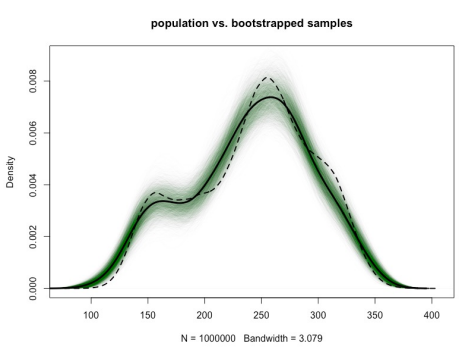
Choose your Linux Platform:

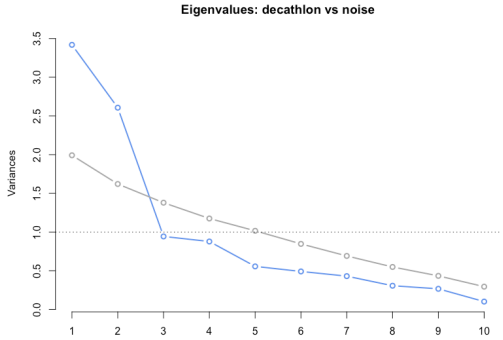
Red Hat/CentOS | Debian/Ubuntu | SLES/openSUSE | Other Platforms



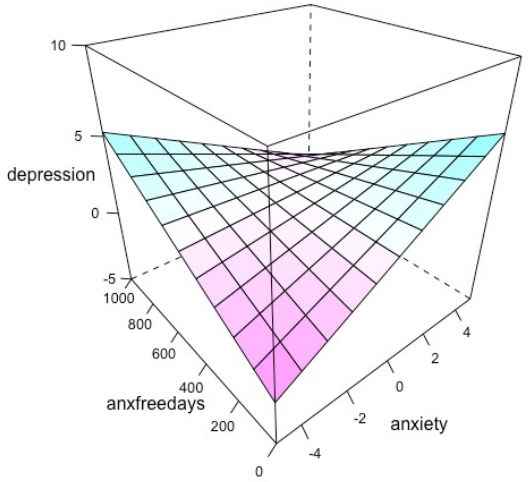
Data Tool Users ➡ Data Tool Makers

Resample and Simulate for Inference

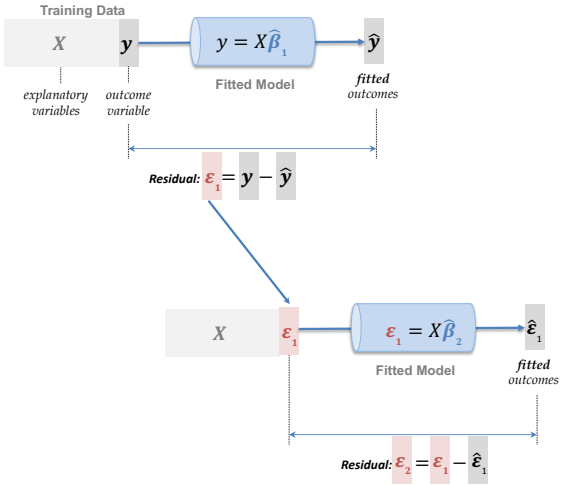




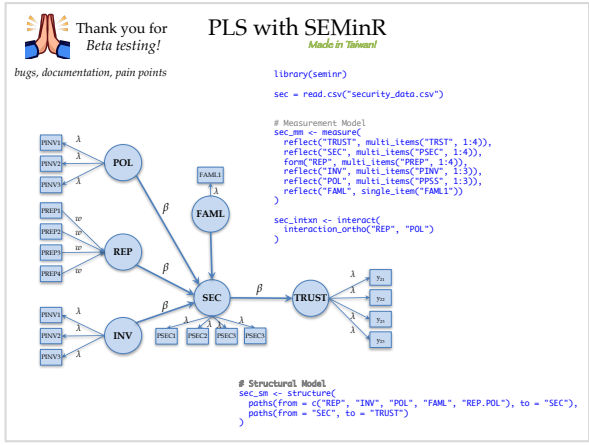
Interactive animations to understand your data



Engage with Machine Learning



Don't be afraid to make your own tools



Thank you students for Engaging this Semester!

Big Thank You to our Teaching Assistants:

Edgar Durón Ramos
Daren Smith
Hsuan-Jung Lin