Welcome and Introduction

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Data Scientists, Win Vector LLC

datacamp

What is Regression?

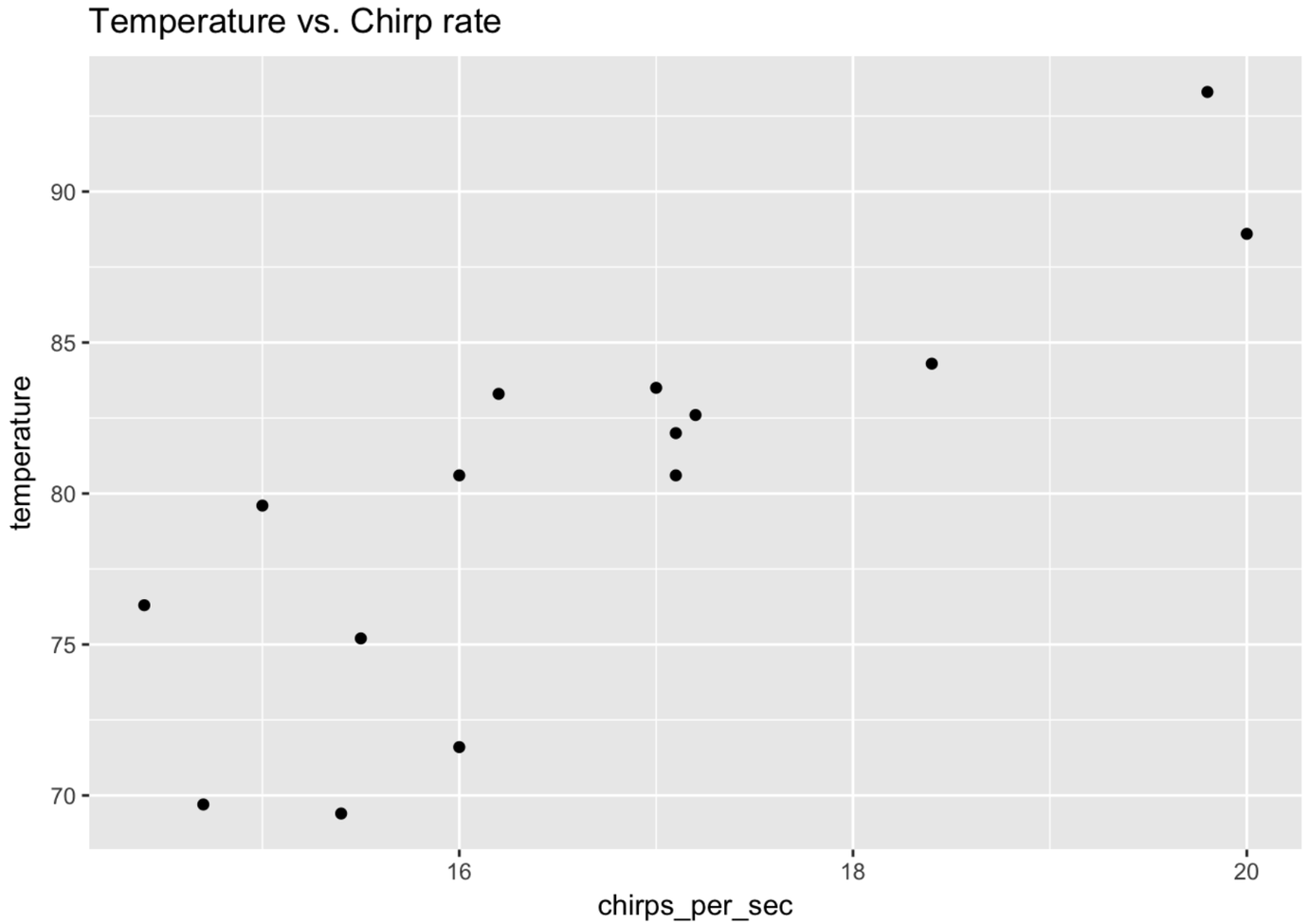
Regression: Predict a numerical outcome ("dependent variable") from a set of inputs ("independent variables").

- *Statistical Sense*: Predicting the expected value of the outcome.
- *Casual Sense*: Predicting a numerical outcome, rather than a discrete one.

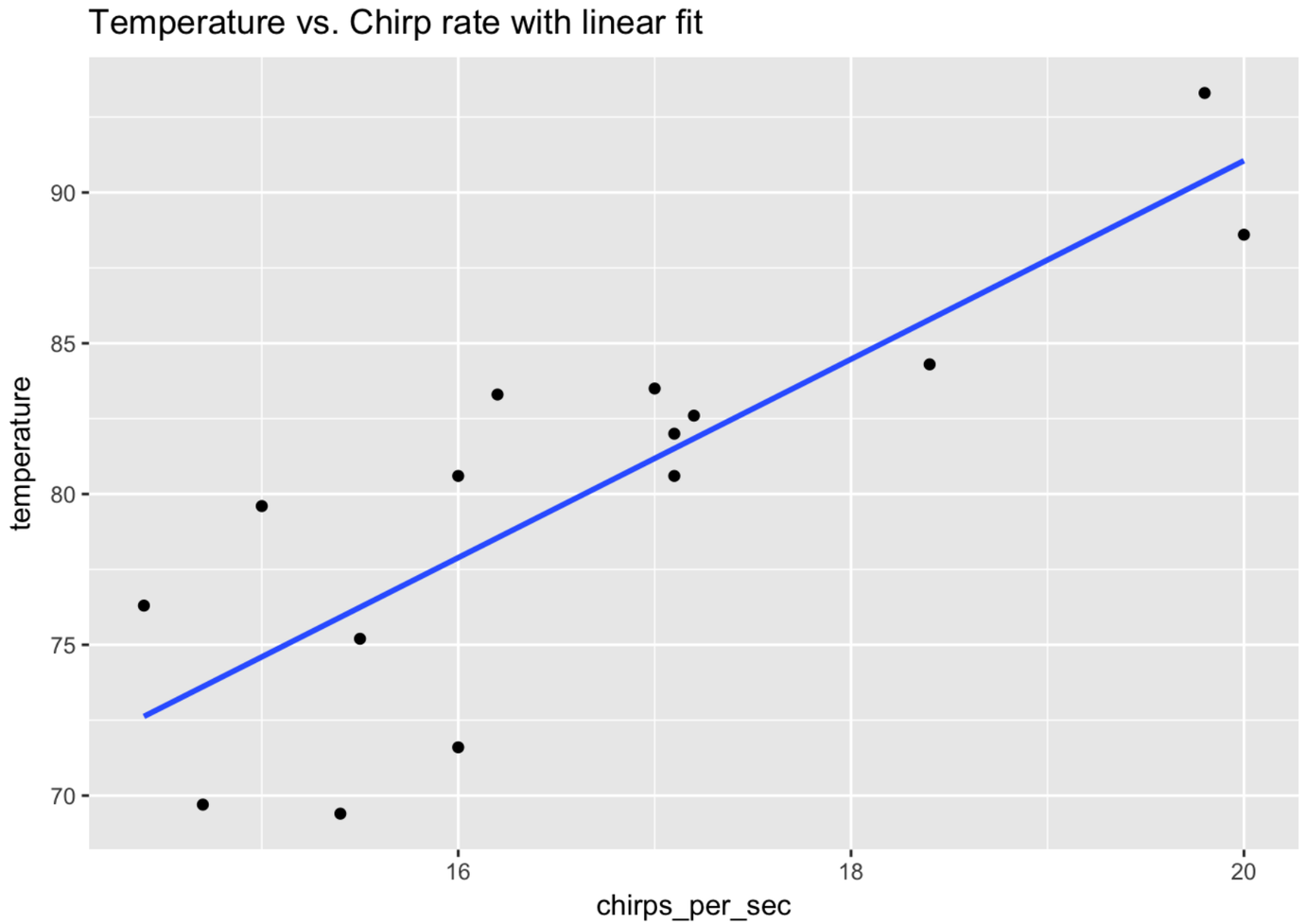
What is Regression?

- *How many units will we sell?* (**Regression**)
- *Will this customer buy our product (yes/no)?* (**Classification**)
- *What price will the customer pay for our product?* (**Regression**)

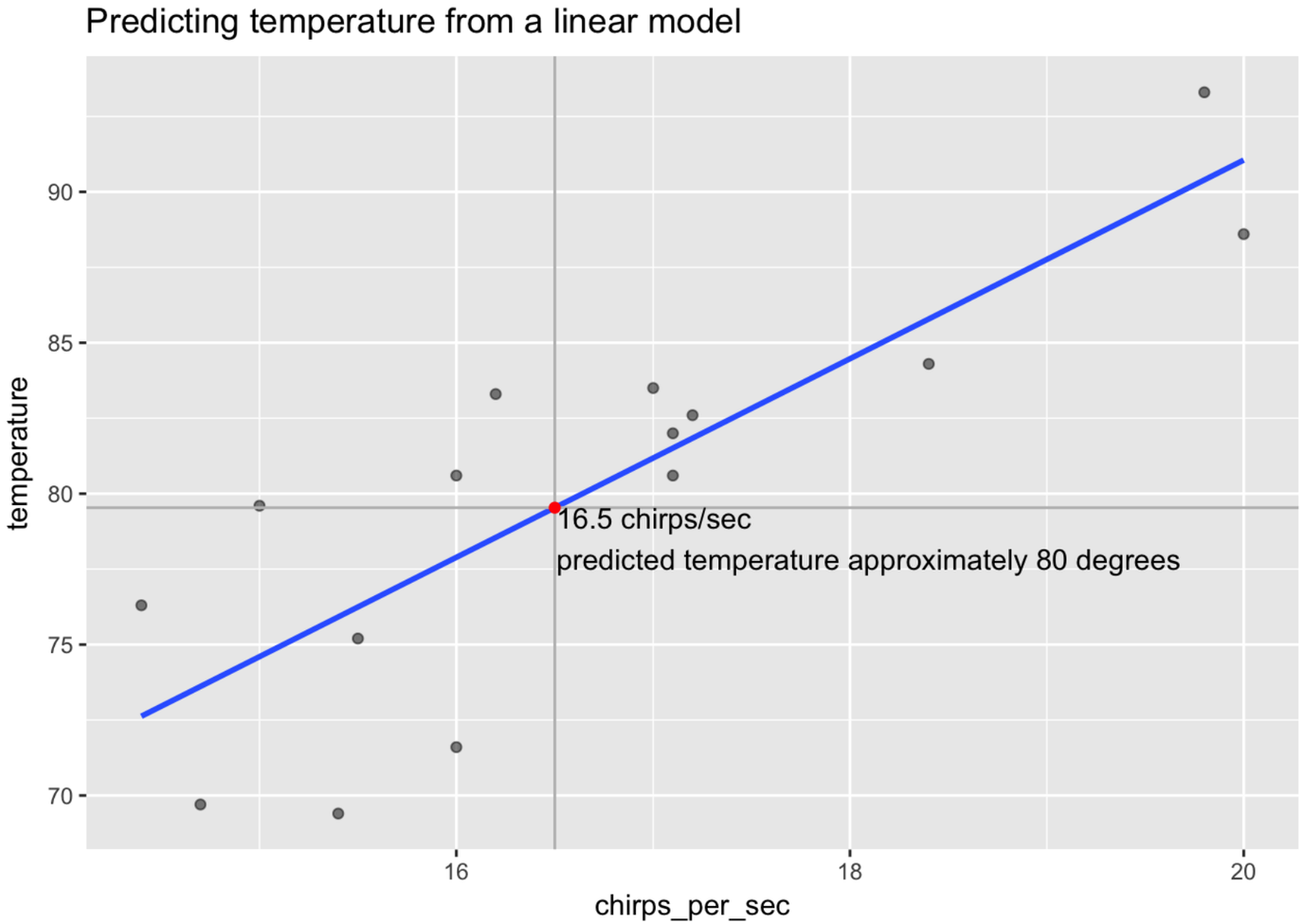
Example: Predict Temperature from Chirp Rate



Predict Temperature from Chirp Rate



Predict Temperature from Chirp Rate



Regression from a Machine Learning Perspective

- *Scientific mindset*: Modeling to understand the data generation process
  - *Engineering mindset*: *Modeling to predict accurately

Machine Learning: Engineering mindset

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

Linear regression - the fundamental method

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector LLC

datacamp

Linear Regression

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots$$

- y is *linearly* related to each x_i
- Each x_i contributes *additively* to y

Linear Regression in R: lm()

```
cmodel <- lm(temperature ~ chirps_per_sec, data = cricket)
```

- formula: `temperature ~ chirps_per_sec`
- data frame: `cricket`

Formulas

```
fmla_1 <- temperature ~ chirps_per_sec
fmla_2 <- blood_pressure ~ age + weight
```

- LHS: outcome
- RHS: inputs
  - use `+` for multiple inputs

```
fmla_1 <- as.formula("temperature ~ chirps_per_sec")
```

Looking at the Model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots$$

```
cmodel
```

```
Call:
lm(formula = temperature ~ chirps_per_sec, data = cricket)

Coefficients:
   (Intercept)  chirps_per_sec
        25.232           3.291
```

More Information about the Model

```
summary(cmodel)
```

```
Call:
lm(formula = fmla, data = cricket)

Residuals:
   Min     1Q Median     3Q    Max
-6.515 -1.971  0.490  2.807  5.001

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)     25.2323    10.0601   2.508 0.026183 *
chirps_per_sec   3.2911     0.6012   5.475 0.000107 ***

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.829 on 13 degrees of freedom
Multiple R-squared:  0.6975, Adjusted R-squared:  0.6742
F-statistic: 29.97 on 1 and 13 DF,  p-value: 0.0001067
```

More Information about the Model

```
broom::glance(cmodel)


sigr::wrapFTest(cmodel)
```

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

Predicting once you fit a model

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector LLC

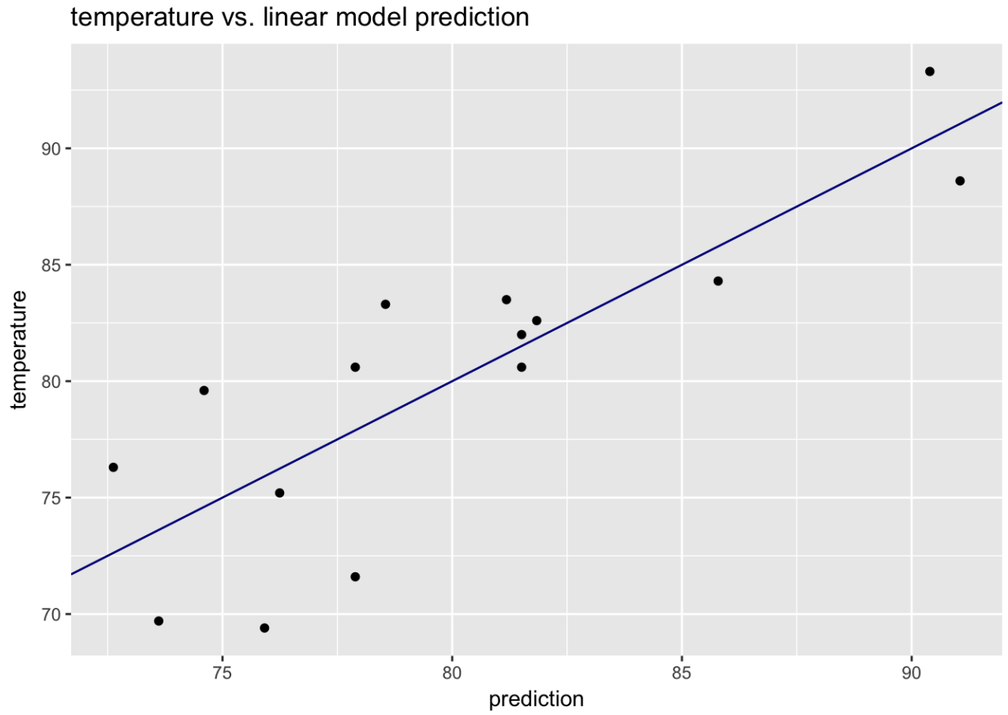
Predicting From the Training Data

```
cricket$prediction <- predict(cmodel)
```

- `predict()` by default returns training data predictions

Looking at the Predictions

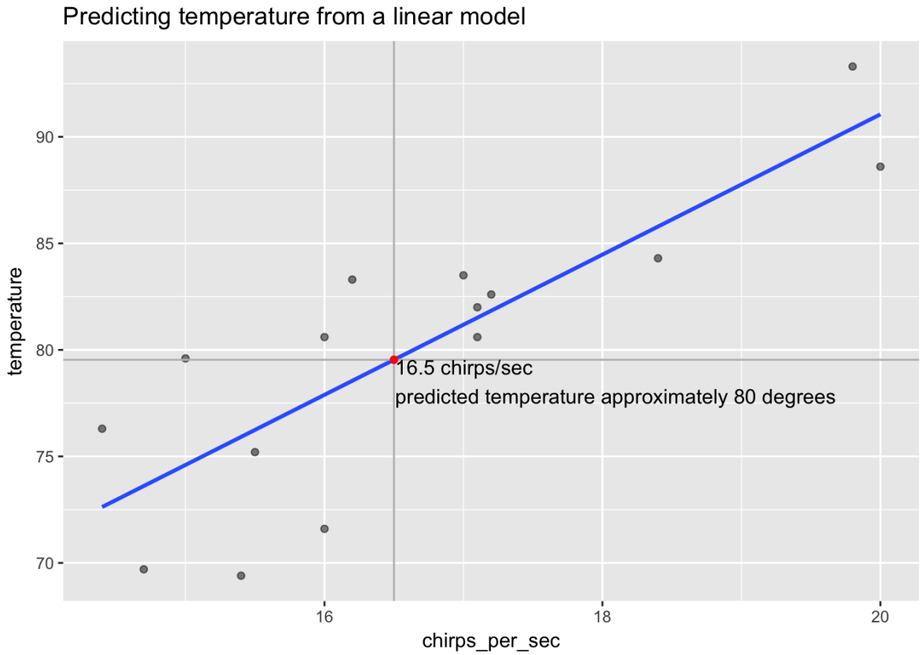
```
ggplot(cricket, aes(x = prediction, y = temperature)) +
     geom_point() +
     geom_abline(color = "darkblue") +
     ggtitle("temperature vs. linear model prediction")
```



Predicting on New Data

```
newchirps <- data.frame(chirps_per_sec = 16.5)
newchirps$prediction <- predict(cmodel, newdata = newchirps)
newchirps
```

```
  chirps_per_sec     pred
1           16.5 79.53537
```



Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

Wrapping up linear regression

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector, LLC

Pros and Cons of Linear Regression

- Pros
  - Easy to fit and to apply
  - Concise
  - Less prone to overfitting

Pros and Cons of Linear Regression

- Pros
  - Easy to fit and to apply
  - Concise
  - Less prone to overfitting
  - **Interpretable**

```
Call:
lm(formula = blood_pressure ~ age + weight, data = bloodpressure)

Coefficients:
(Intercept)          age       weight
    30.9941       0.8614       0.3349
```

Pros and Cons of Linear Regression

- Pros
  - Easy to fit and to apply
  - Concise
  - Less prone to overfitting
  - Interpretable
- Cons
  - Can only express linear and additive relationships

Collinearity

- **Collinearity** -- when input variables are partially correlated.

```
Call:
lm(formula = blood_pressure ~ age + weight, data = bloodpressure)

Coefficients:
(Intercept)          age       weight
    30.9941       0.8614       0.3349
```

Collinearity

- **Collinearity** -- when variables are partially correlated.
- Coefficients might change sign

```
Call:
lm(formula = blood_pressure ~ age + weight, data = bloodpressure)

Coefficients:
(Intercept)          age       weight
    30.9941       0.8614       0.3349
```

Collinearity

- **Collinearity** -- when variables are partially correlated.
- Coefficients might change sign
- High collinearity:
  - Coefficients (or standard errors) look too large
  - Model may be unstable

```
Call:
lm(formula = blood_pressure ~ age + weight, data = bloodpressure)

Coefficients:
(Intercept)          age       weight
    30.9941       0.8614       0.3349
```

Coming Next

- Evaluating a regression model
- Properly training a model

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

Evaluating a model graphically

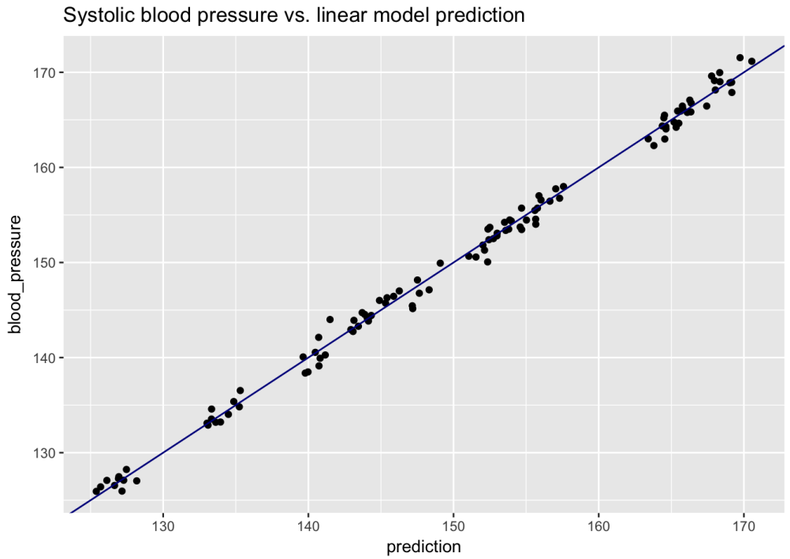
SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector LLC

datacamp

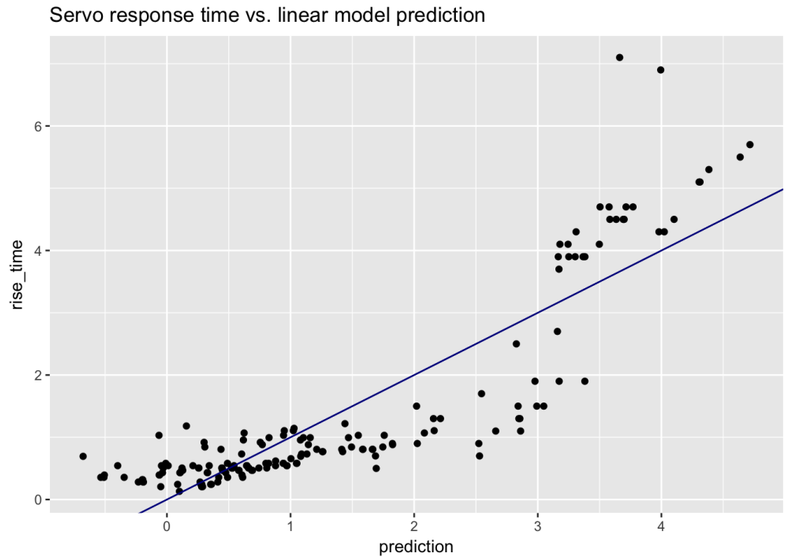
Plotting Ground Truth vs. Predictions

A well fitting model



- *x = y* line runs through center of points
- "line of perfect prediction"

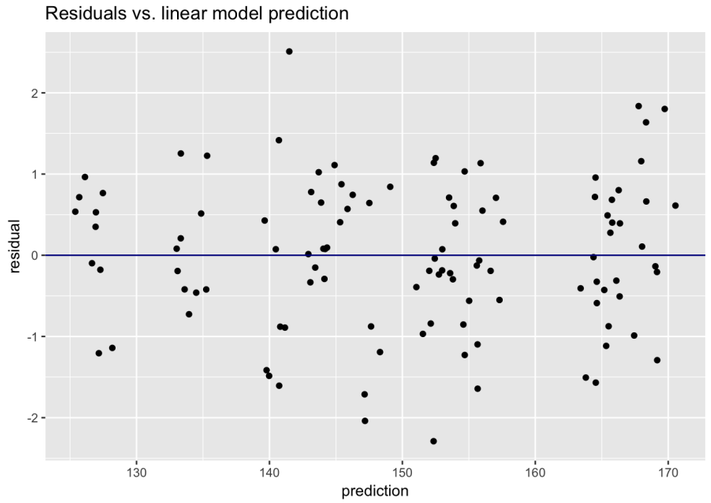
A poorly fitting model



- Points are all on one side of *x = y* line
- Systematic errors

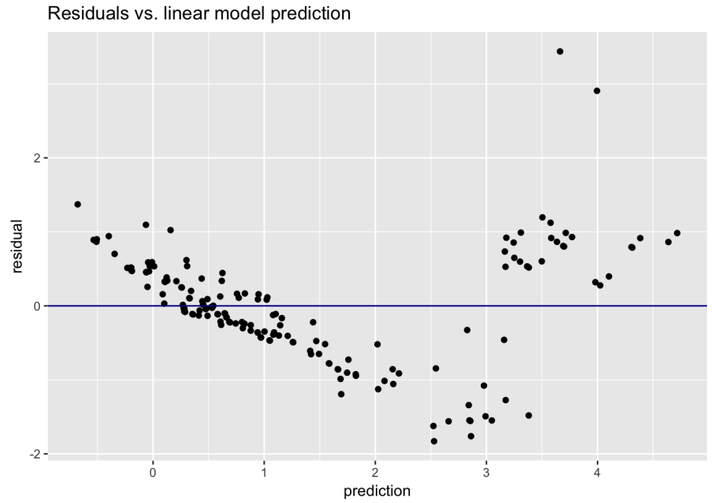
The Residual Plot

A well fitting model



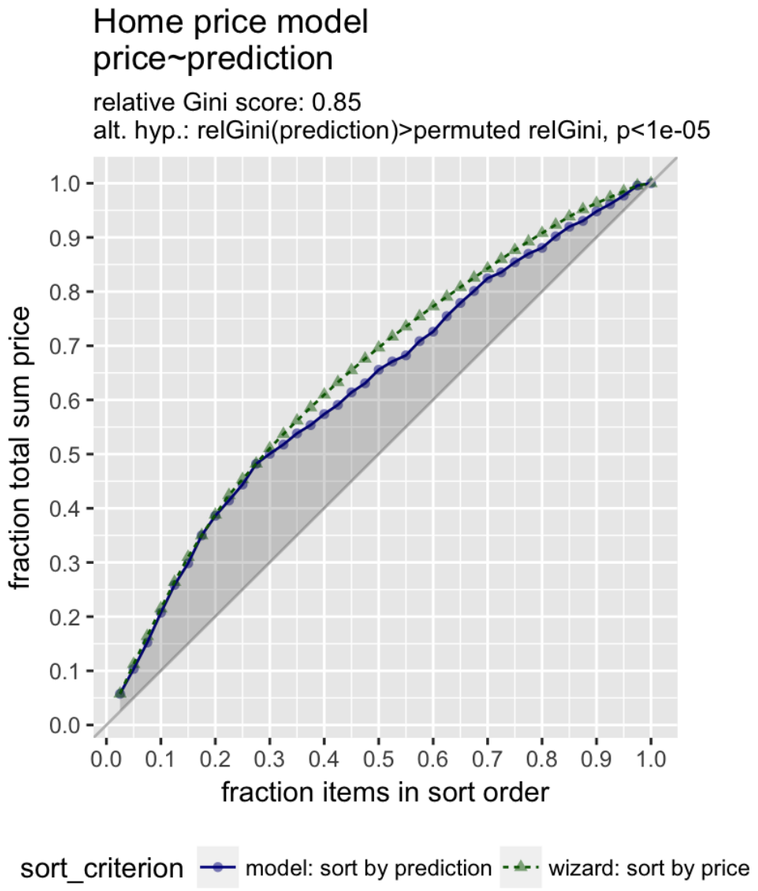
- Residual: actual outcome - prediction
- Good fit: no systematic errors

A poorly fitting model



- Systematic errors

The Gain Curve

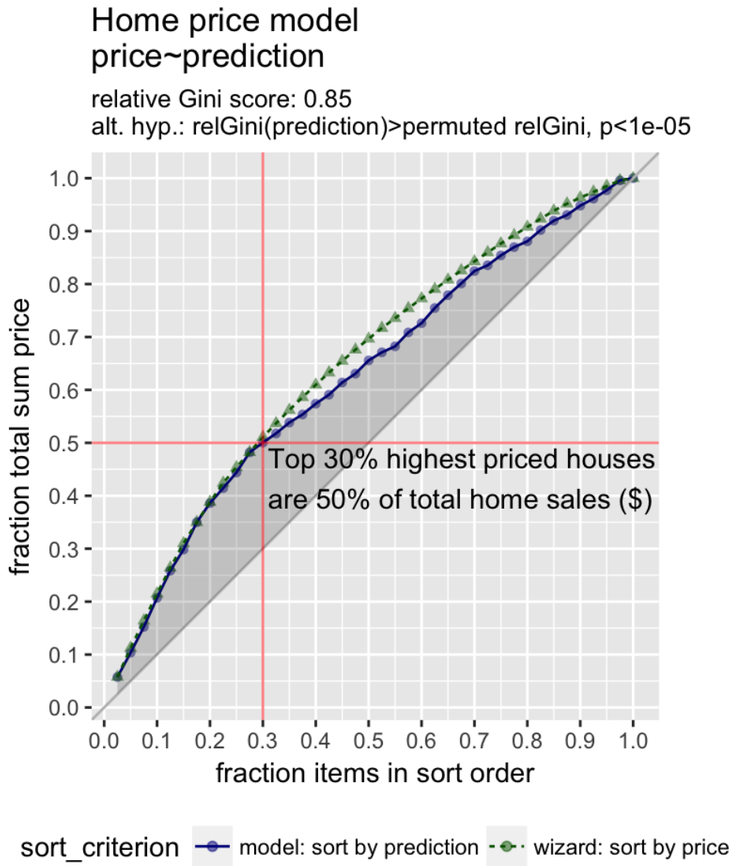


Measures how well model sorts the outcome

- **x-axis**: houses in model-sorted order (decreasing)
- **y-axis**: fraction of total accumulated home sales

Wizard curve: perfect model

Reading the Gain Curve




```
GainCurvePlot(houseprices, "prediction", "price", "Home price model")
```

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

Root Mean Squared Error (RMSE)

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector LLC

What is Root Mean Squared Error (RMSE)?

$$RMSE = \sqrt{\overline{(pred - y)^2}}$$

where

- $pred - y$: the error, or residuals vector
- $\overline{(pred - y)^2}$: mean value of $(pred - y)^2$

RMSE of the Home Sales Price Model

```
# Calculate error
err <- houseprices$prediction - houseprices$price
```

- `price` : column of actual sale prices (in thousands)
- `prediction` : column of predicted sale prices (in thousands)

RMSE of the Home Sales Price Model

```
# Calculate error
err <- houseprices$prediction - houseprices$price

# Square the error vector
err2 <- err^2
```

RMSE of the Home Sales Price Model

```
# Calculate error
err <- houseprices$prediction - houseprices$price

# Square the error vector
err2 <- err^2

# Take the mean, and sqrt it
(rmse <- sqrt(mean(err2)))
```

```
58.33908
```

- $RMSE \approx 58.3$

Is the RMSE Large or Small?

```
# Take the mean, and sqrt it
(rmse <- sqrt(mean(err2)))
```

```
58.33908
```

```
# The standard deviation of the outcome
(sdtemp <- sd(houseprices$price))
```

```
135.2694
```

- $RMSE \approx 58.3$
- $sd(price) \approx 135$

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

R-squared (R^2)

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector LLC

What is R^2?

A measure of how well the model fits or explains the data

- A value between 0-1
  - near 1: model fits well
  - near 0: no better than guessing the average value

Calculating R^2

R^2 is the *variance explained by the model.*

$$R^2 = 1 - \frac{RSS}{SS_{Tot}}$$

where

- $RSS = \sum (y - prediction)^2$
  - Residual sum of squares (variance from model)
- $SS_{Tot} = \sum (y - \overline{y})^2$
  - Total sum of squares (variance of data)

Calculate R^2 of the House Price Model: RSS

- Calculate error

```
err <- houseprices$prediction - houseprices$price
```

- Square it and take the sum

```
rss <- sum(err^2)
```

- `price` : column of actual sale prices (in thousands)
- `pred` : column of predicted sale prices (in thousands)
- $RSS \approx 136138$

Calculate R^2 of the House Price Model: SS_{Tot}

- Take the difference of prices from the mean price

```
toterr <- houseprices$price - mean(houseprices$price)
```

- Square it and take the sum

```
sstot <- sum(toterr^2)
```

- $RSS \approx 136138$
- $SS_{Tot} \approx 713615$

Calculate R^2 of the House Price Model

```
(r_squared <- 1 - (rss/sstot) )
```

```
0.8092278
```

- $RSS \approx 136138$
- $SS_{Tot} \approx 713615$
- $R^2 \approx 0.809$

Reading R^2 from the lm() model

```
# From summary()
summary(hmodel)
```

```
...
Residual standard error: 60.66 on 37 degrees of freedom
Multiple R-squared:  0.8092, Adjusted R-squared:  0.7989
F-statistic: 78.47 on 2 and 37 DF,  p-value: 4.893e-14
```

```
summary(hmodel)$r.squared
```

```
0.8092278
```

```
# From glance()
glance(hmodel)$r.squared
```

```
0.8092278
```

Correlation and R^2

```
rho <- cor(houseprices$prediction, houseprices$price)
```

```
0.8995709
```

```
rho^2
```

```
0.8092278
```

- ρ = `cor(prediction, price)` = 0.8995709
- ρ^2 = 0.8092278 = R^2

Correlation and R^2

- True for models that minimize squared error:
  - Linear regression
  - GAM regression
  - Tree-based algorithms that minimize squared error
- True for training data; **NOT** true for future application data

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

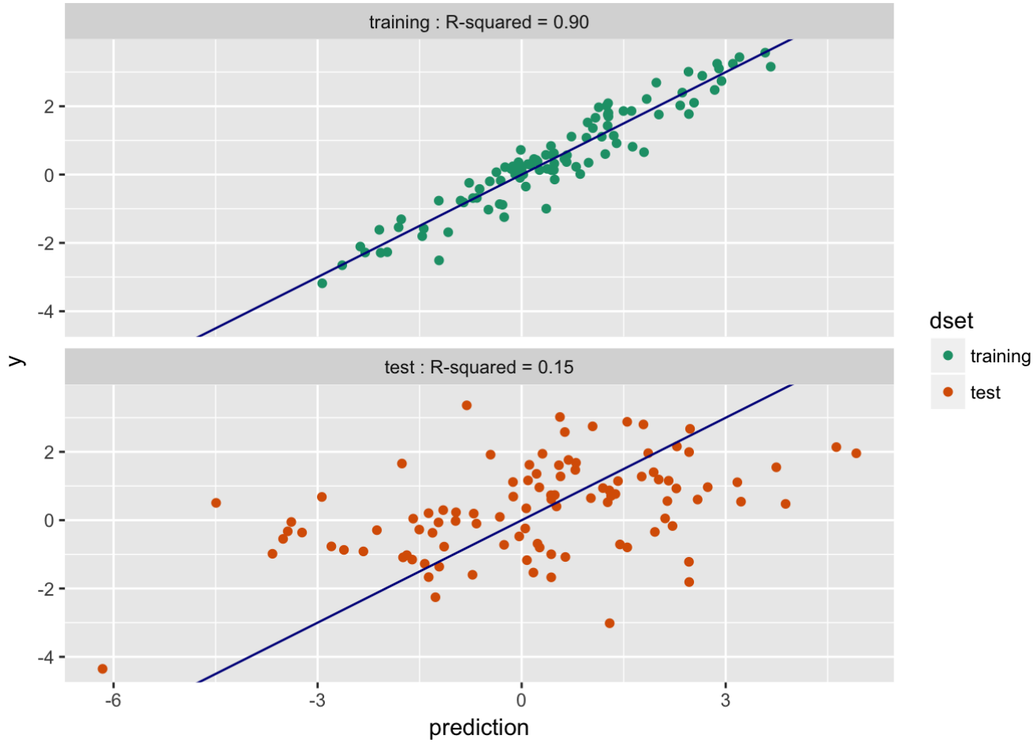
Properly Training a Model

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector, LLC

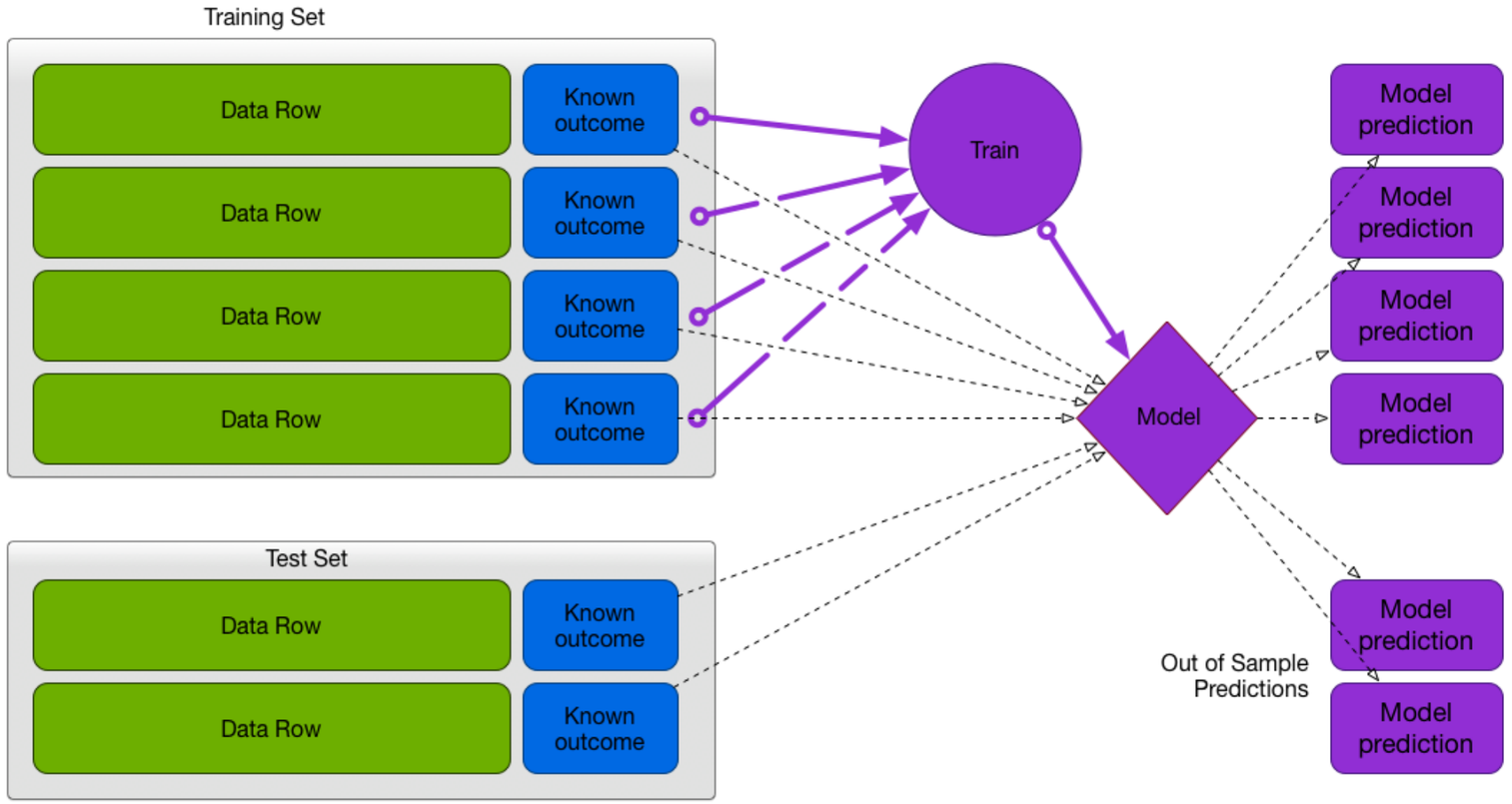
datacamp

Models can perform much better on training than they do on future data.



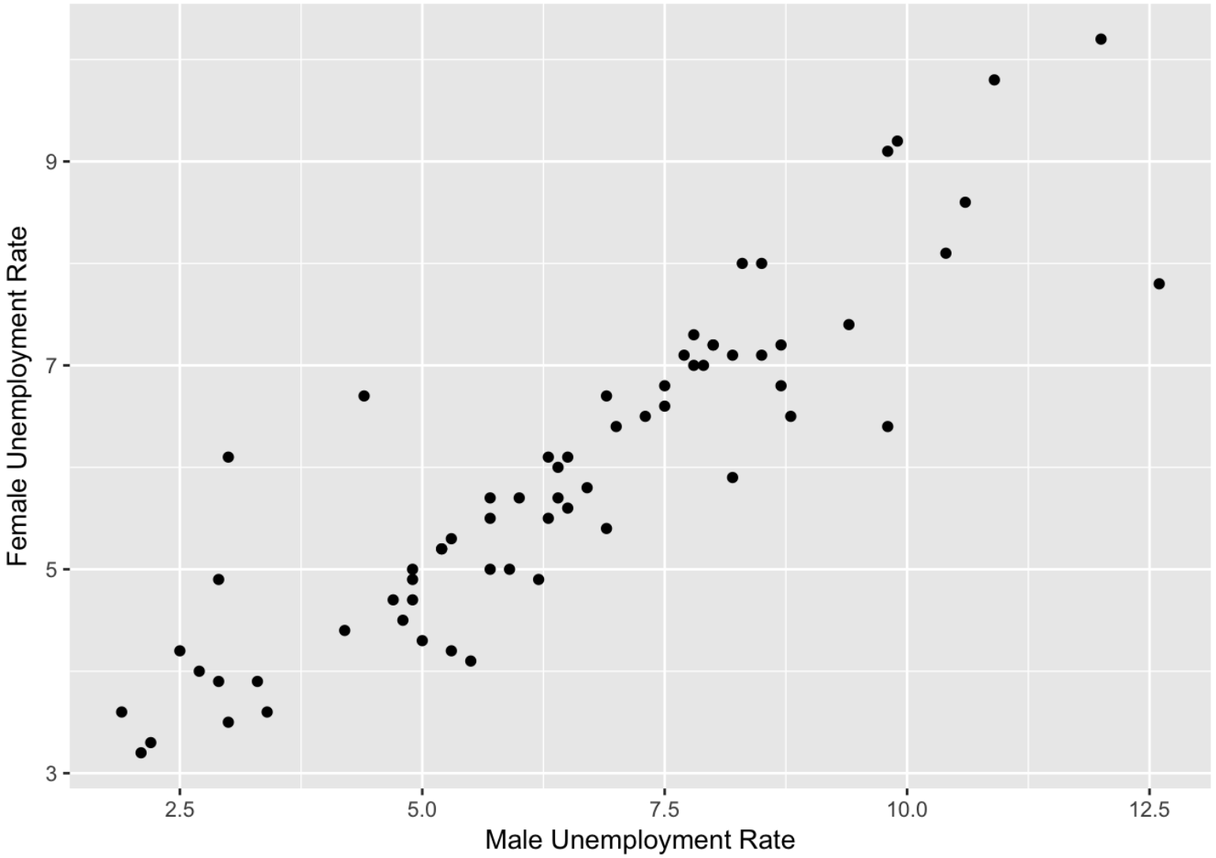
- Training R^2: 0.9; Test R^2: 0.15 -- **Overfit**

Test/Train Split



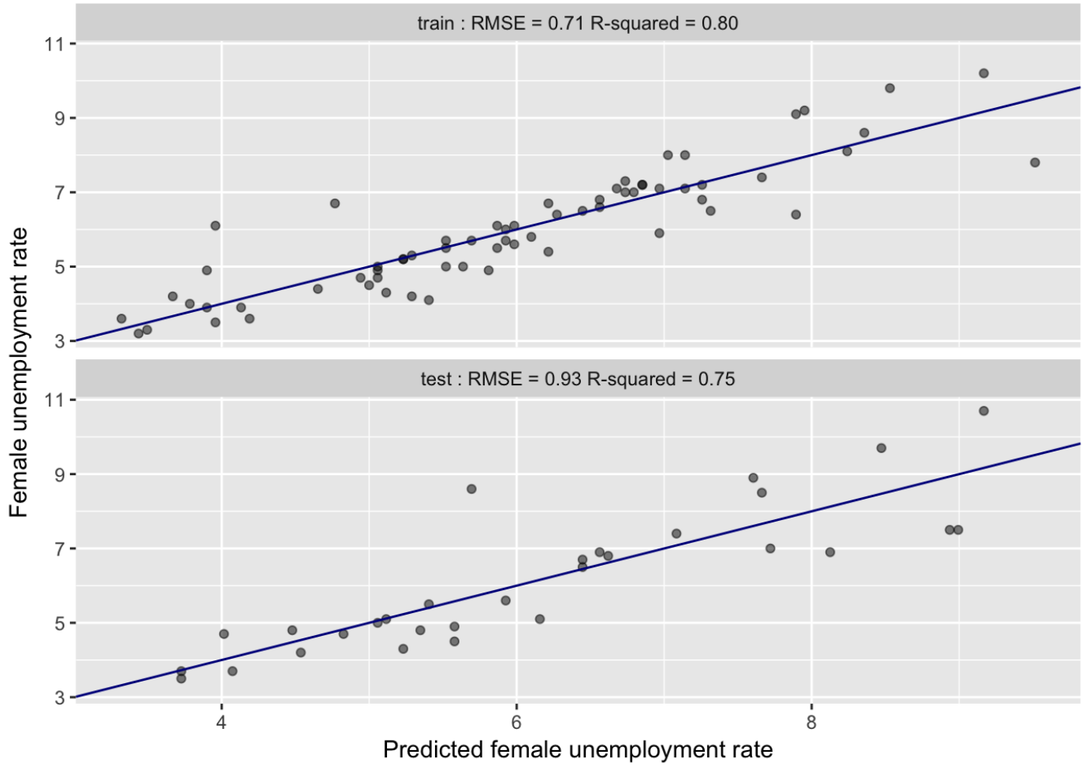
Recommended method when data is plentiful

Example: Model Female Unemployment



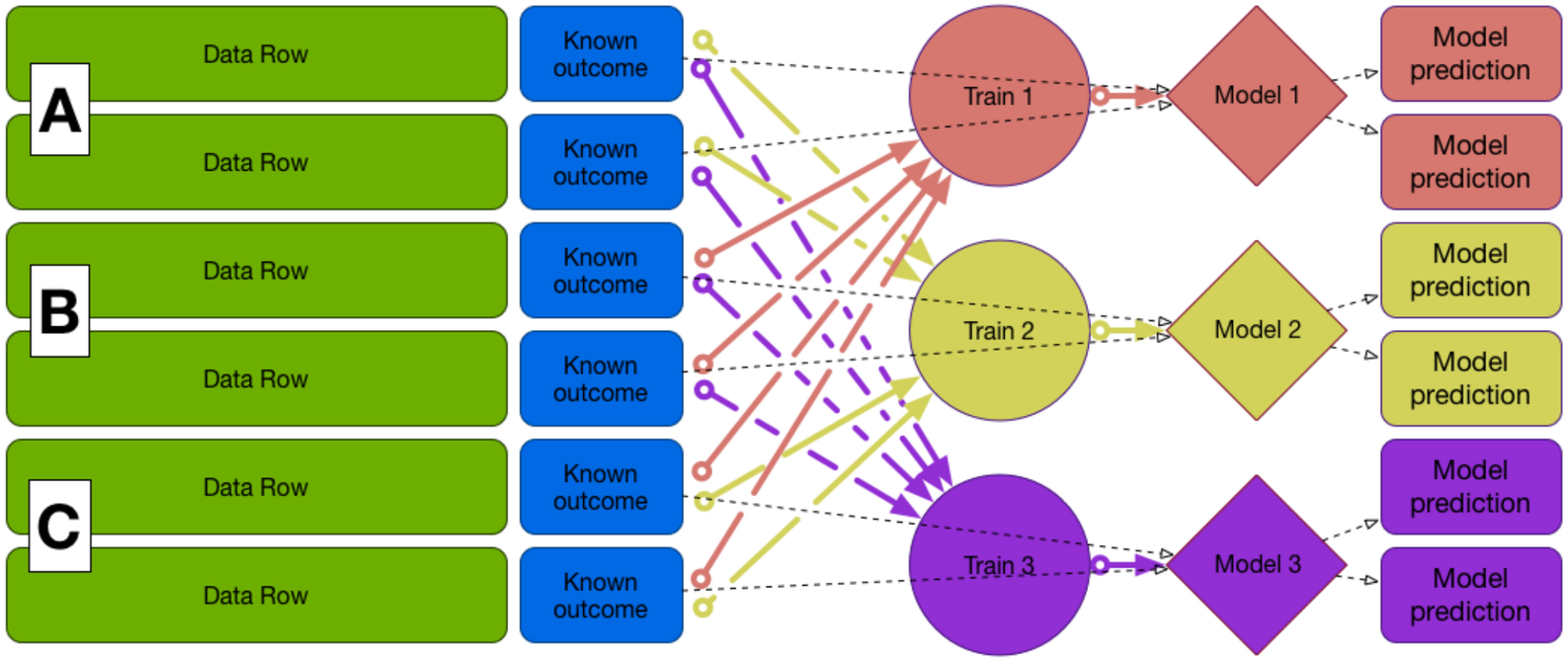
- Train on 66 rows, test on 30 rows

Model Performance: Train vs. Test



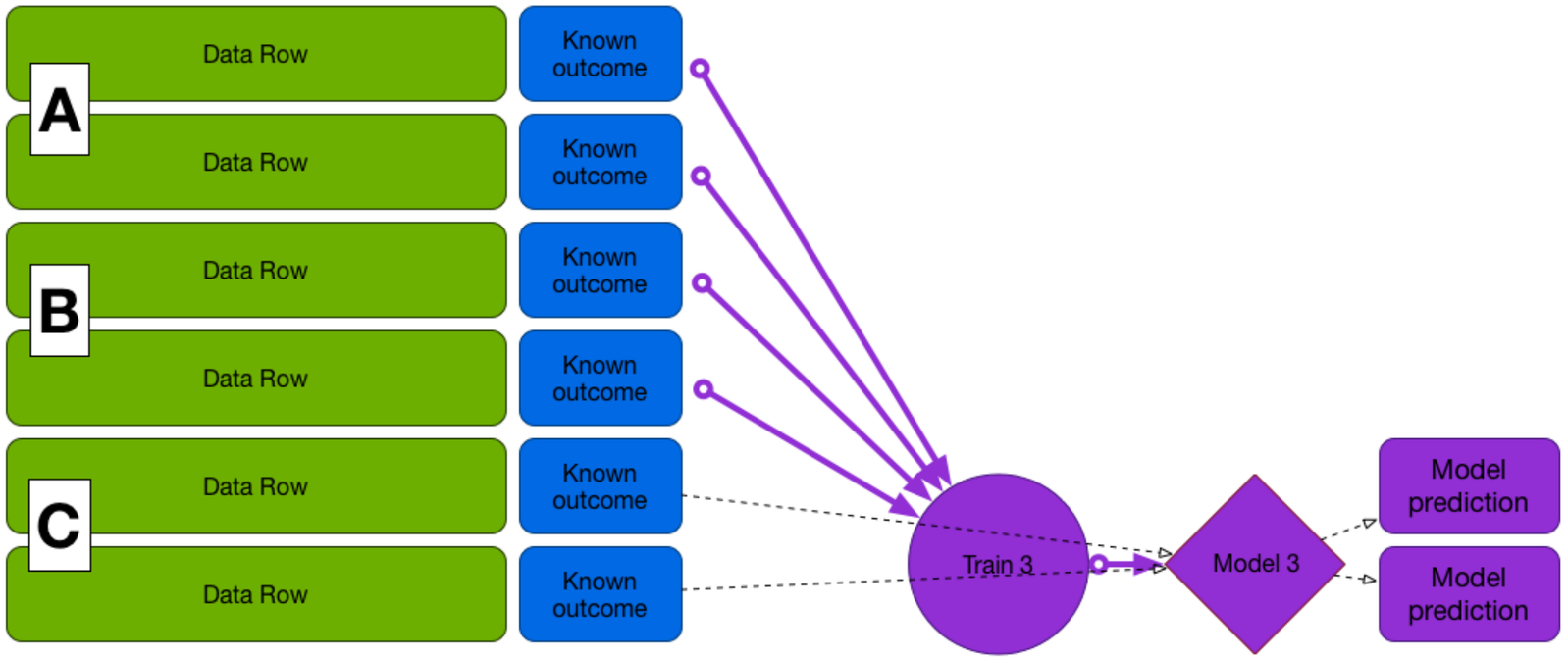
- Training: RMSE 0.71, R^2 0.8
- Test: RMSE 0.93, R^2 0.75

Cross-Validation

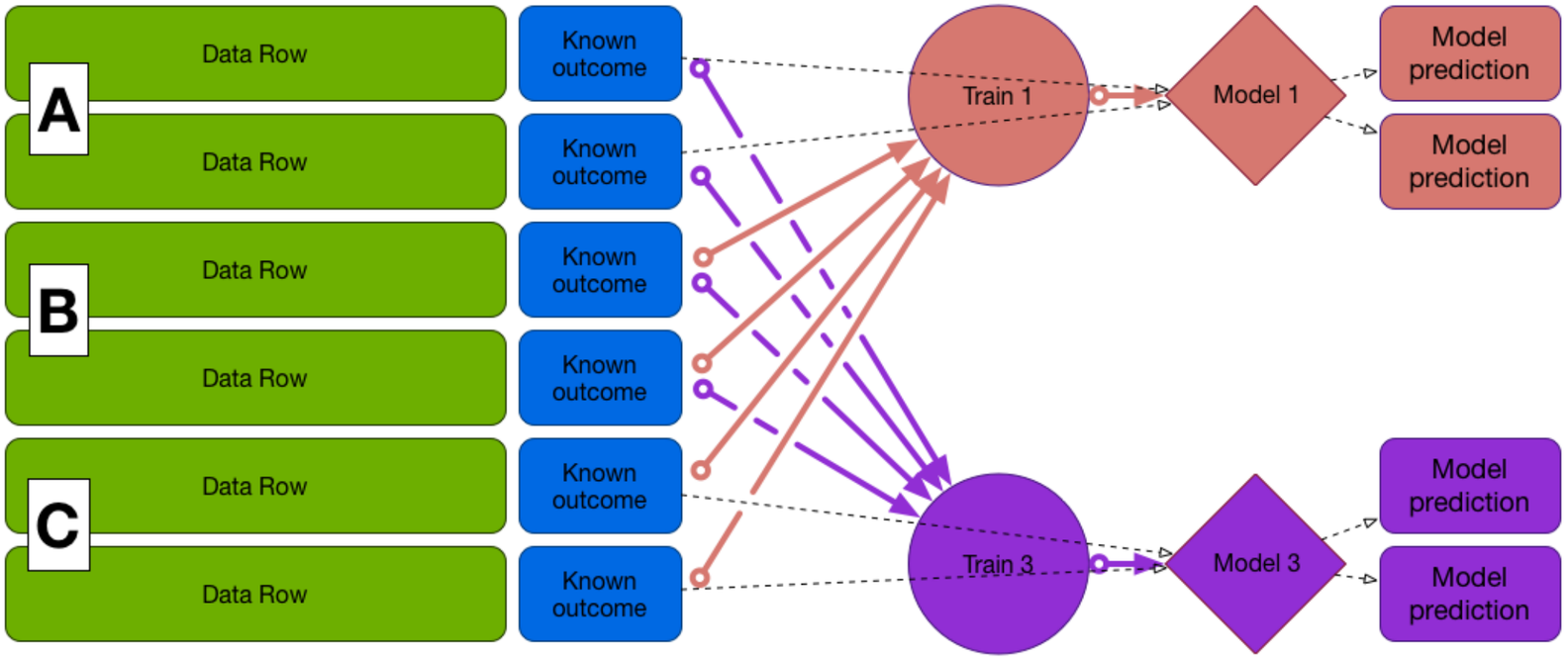


Preferred when data is not large enough to split off a test set

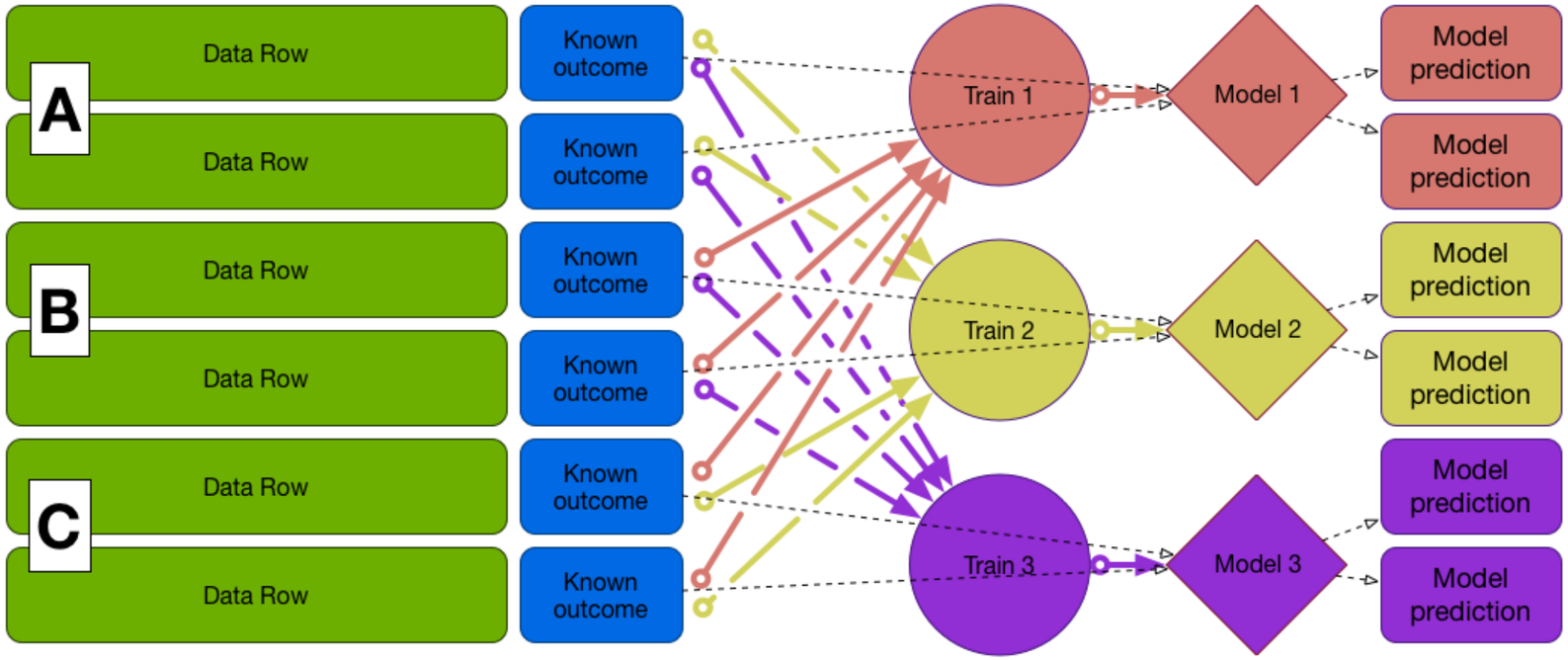
Cross-Validation



Cross-Validation



Cross-Validation



Create a cross-validation plan

```
library(vtreat)
splitPlan <- kWayCrossValidation(nRows, nSplits, NULL, NULL)
```

- `nRows` : number of rows in the training data
- `nSplits` : number folds (partitions) in the cross-validation
  - e.g, nfolds = 3 for 3-way cross-validation
- remaining 2 arguments not needed here

Create a cross-validation plan

```
library(vtreat)
splitPlan <- kWayCrossValidation(10, 3, NULL, NULL)
```

First fold (A and B to train, C to test)

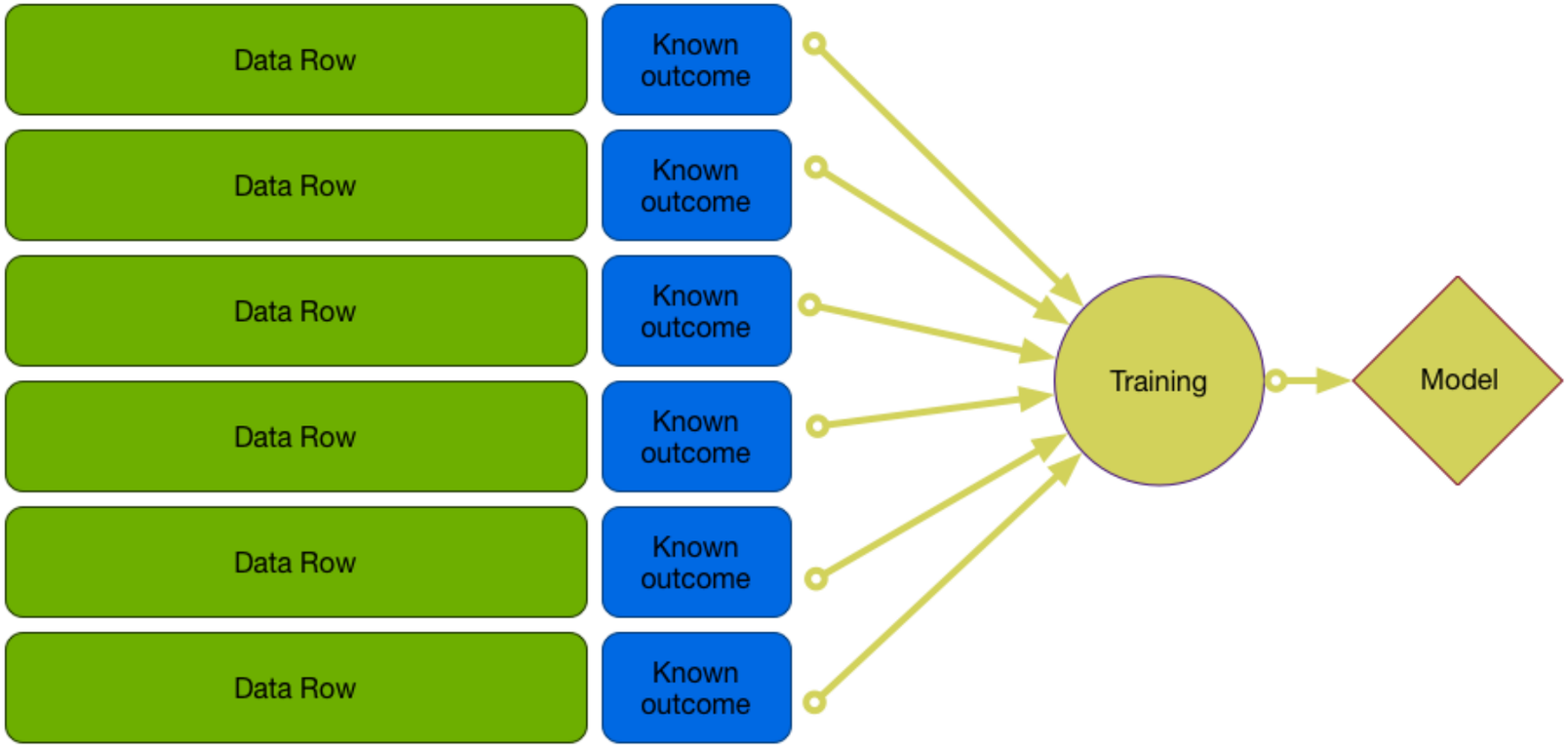
```
splitPlan[[1]]
```

```
$train
1  2  4  5  7  9 10
$app
3 6 8
```

Train on A and B, test on C, etc...

```
split <- splitPlan[[1]]
model <- lm(fmla, data = df[split$train,])
df$pred.cv[split$app] <- predict(model, newdata = df[split$app,])
```

Final Model


Data Row
Data Row
Data Row
Data Row
Data Row
Data Row
Known outcome
Known outcome
Known outcome
Known outcome
Known outcome
Known outcome
Training
Model

Example: Unemployment Model

Measure type	RMSE	R^2
train	0.7082675	0.8029275
test	0.9349416	0.7451896
cross-validation	0.8175714	0.7635331

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

Categorical inputs

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector, LLC

datacamp

Example: Effect of Diet on Weight Loss

```
WtLoss24 ~ Diet + Age + BMI
```

Diet	Age	BMI	WtLoss24
Med	59	30.67	-6.7
Low-Carb	48	29.59	8.4
Low-Fat	52	32.9	6.3
Med	53	28.92	8.3
Low-Fat	47	30.20	6.3

model.matrix()

```
model.matrix(WtLoss24 ~ Diet + Age + BMI, data = diet)
```

- All numerical values
- Converts categorical variable with N levels into N - 1 indicator variables

Indicator Variables to Represent Categories

Original Data

Diet	Age	...
Med	59	...
Low-Carb	48	...
Low-Fat	52	...
Med	53	...
Low-Fat	47	...

Model Matrix

(Int)	DietLow-Fat	DietMed	...
1	0	1	...
1	0	0	...
1	1	0	...
1	0	1	...
1	1	0	...

- reference level: "Low-Carb"

Interpreting the Indicator Variables

Linear Model:

$$WtLoss24 = \beta_0 + \beta_{DietLow}x_{DietLow} + \beta_{DietMed}x_{DietMed} + \beta_{Age}x_{Age} + \beta_{BMI}x_{BMI}$$

```
lm(WtLoss24 ~ Diet + Age + BMI, data = diet))
```

```
Coefficients:
      (Intercept)        DietLow-Fat     DietMed
         -1.37149           -2.32130    -0.97883
              Age                BMI
          0.12648            0.01262
```

Issues with one-hot-encoding

- Too many levels can be a problem
  - Example: ZIP code (about 40,000 codes)
- Don't hash with geometric methods!

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

Interactions

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector, LLC

Additive relationships

Example of an additive relationship:

```
plant_height ~ bacteria + sun
```

- Change in height is the sum of the effects of bacteria and sunlight
  - Change in sunlight causes same change in height, independent of bacteria
  - Change in bacteria causes same change in height, independent of sunlight

What is an Interaction?

The simultaneous influence of two variables on the outcome is not additive.

```
plant_height ~ bacteria + sun + bacteria:sun
```

- Change in height is more (or less) than the sum of the effects due to sun/bacteria
- At higher levels of sunlight, 1 unit change in bacteria causes more change in height

What is an Interaction?

The simultaneous influence of two variables on the outcome is not additive.

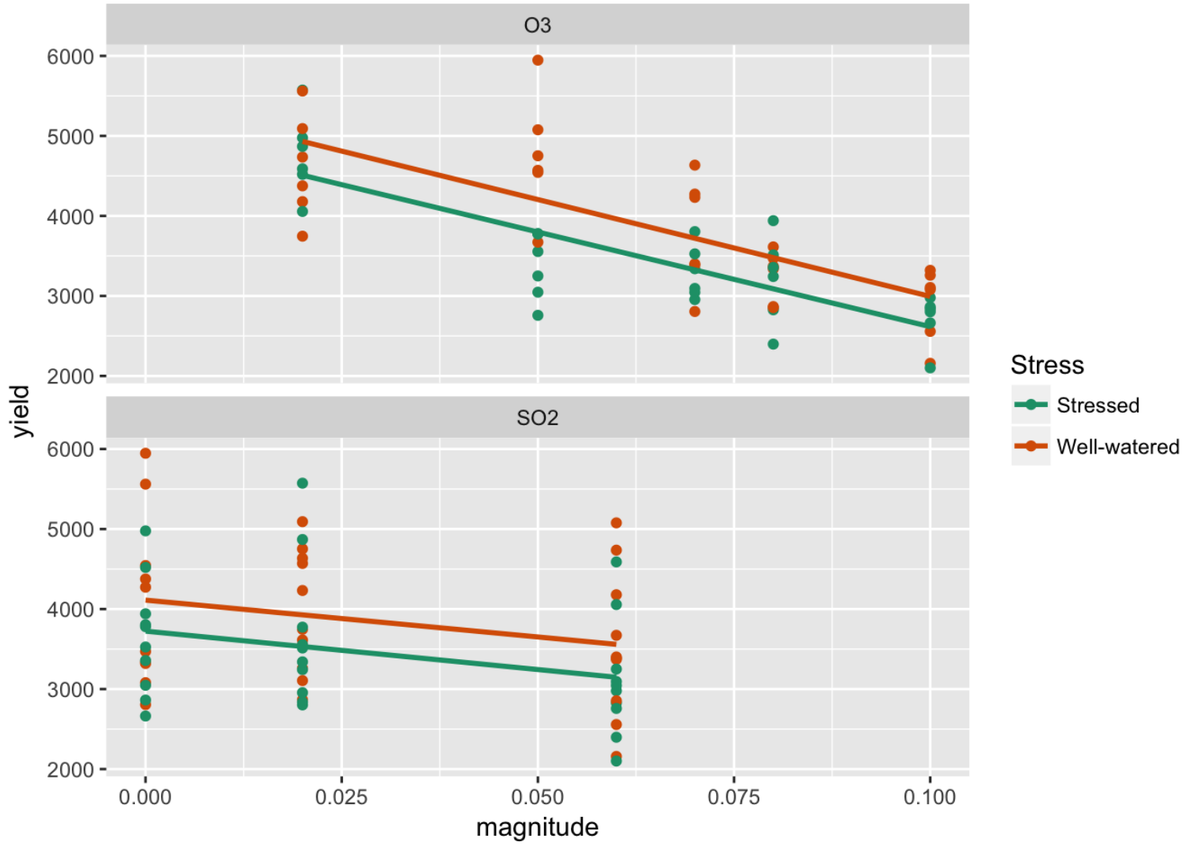
```
plant_height ~ bacteria + sun + bacteria:sun
```

- `sun` : categorical {"sun", "shade"}
- In sun, 1 unit change in bacteria causes *m* units change in height
- In shade, 1 unit change in bacteria causes *n* units change in height

Like two separate models: one for sun, one for shade.

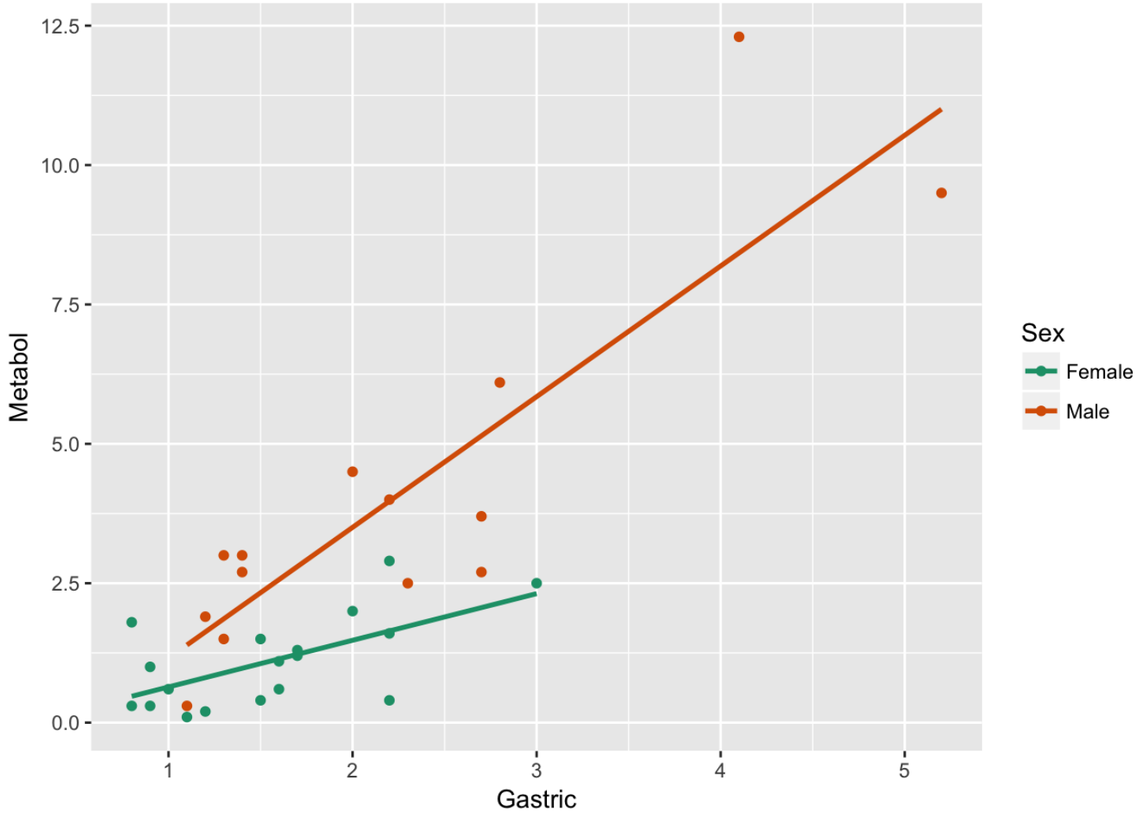
Example of no Interaction: Soybean Yield

```
yield ~ Stress + SO2 + O3
```



Example of an Interaction: Alcohol Metabolism

```
Metabol ~ Gastric + Sex
```



Expressing Interactions in Formulae

- Interaction - Colon (`:`)

```
y ~ a:b
```

- Main effects and interaction - Asterisk (`*`)

```
y ~ a*b
# Both mean the same
y ~ a + b + a:b
```

- Expressing the product of two variables - **I**

```
y ~ I(a*b)
```

same as $y \propto ab$

Finding the Correct Interaction Pattern

Formula	RMSE (cross validation)
`Metabol ~ Gastric + Sex`	1.46
`Metabol ~ Gastric * Sex`	1.48
`Metabol ~ Gastric + Gastric:Sex`	1.39

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

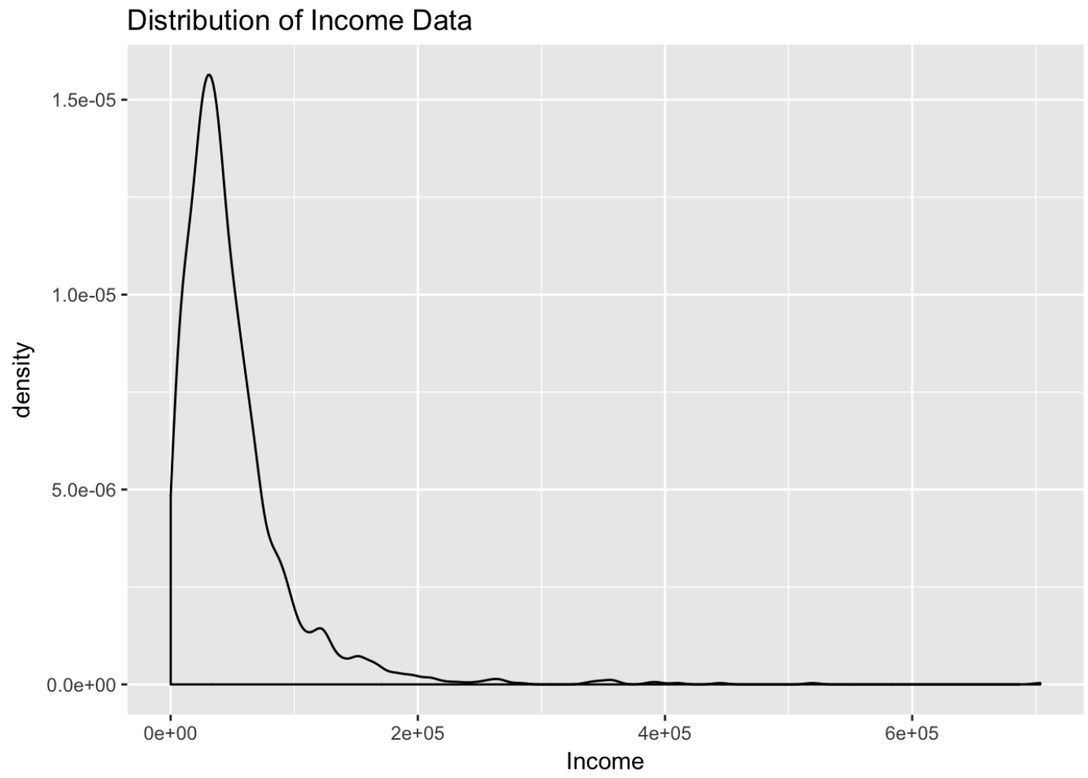
Transforming the response before modeling

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector, LLC

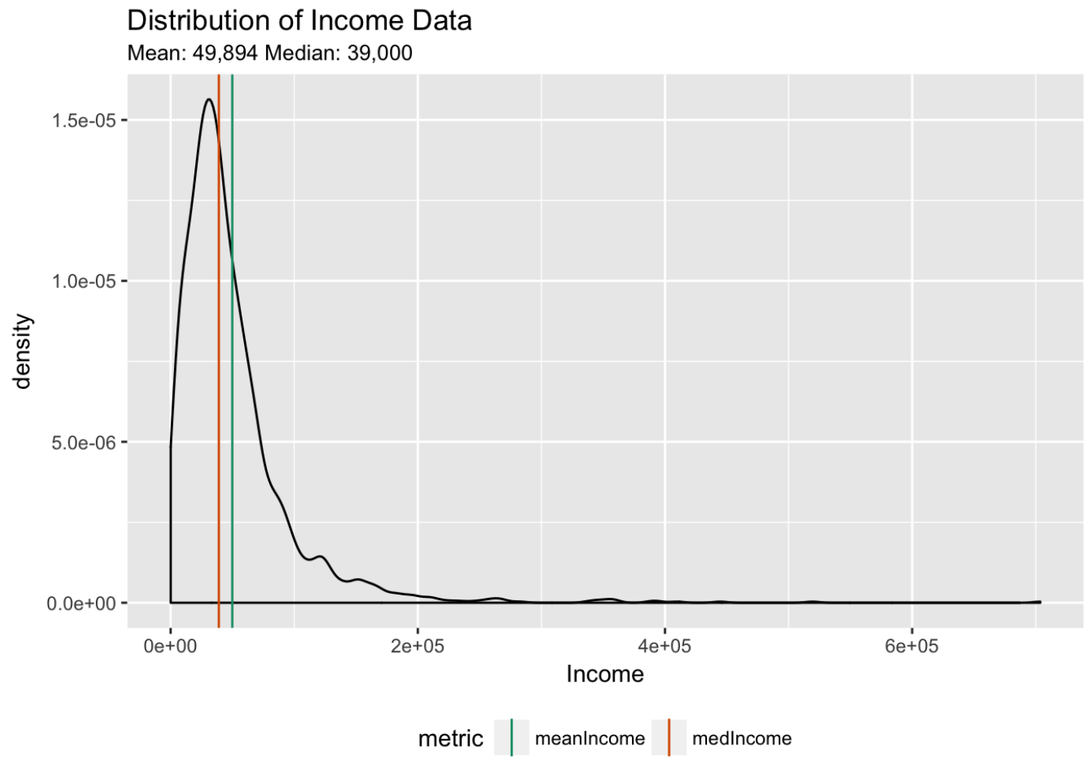
datacamp

The Log Transform for Monetary Data



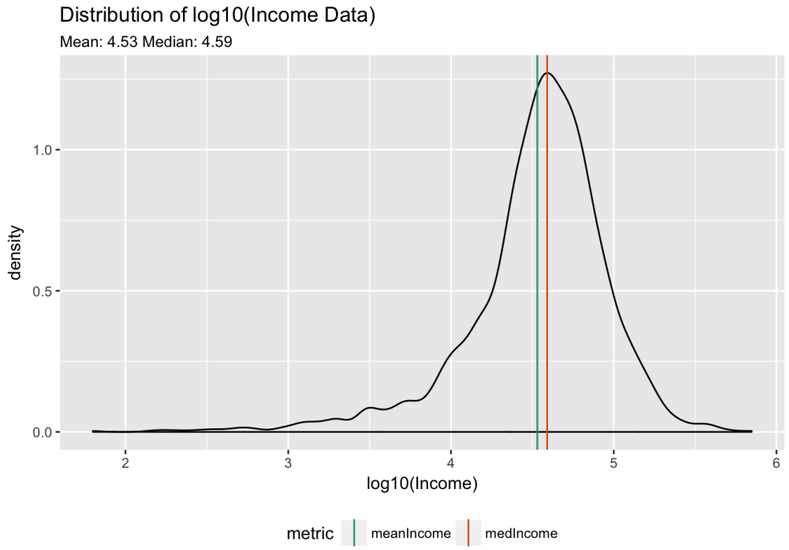
- Monetary values: lognormally distributed
- Long tail, wide dynamic range (60-700K)

Lognormal Distributions



- mean > median (~ 50K vs 39K)
- Predicting the mean will overpredict typical values

Back to the Normal Distribution



For a Normal Distribution:

- mean = median (here: 4.53 vs 4.59)
- more reasonable dynamic range (1.8 - 5.8)

The Procedure

1. Log the outcome and fit a model

```
model <- lm(log(y) ~ x, data = train)
```

The Procedure

1. Log the outcome and fit a model

```
model <- lm(log(y) ~ x, data = train)
```

2. Make the predictions in log space

```
logpred <- predict(model, data = test)
```

The Procedure

1. Log the outcome and fit a model

```
model <- lm(log(y) ~ x, data = train)
```

2. Make the predictions in log space

```
logpred <- predict(model, data = test)
```

3. Transform the predictions to outcome space

```
pred <- exp(logpred)
```

Predicting Log-transformed Outcomes: Multiplicative Error

$$log(a) + log(b) = log(ab)$$

$$log(a) - log(b) = log(a/b)$$

- Multiplicative error: $pred/y$
- Relative error: $(pred - y)/y = \frac{pred}{y} - 1$

Reducing multiplicative error reduces relative error.

Root Mean Squared Relative Error

RMS-relative error = $\sqrt{\overline{(\frac{pred - y}{y})^2}}$

- Predicting log-outcome reduces RMS-relative error
- But the model will often have larger RMSE

Example: Model Income Directly

```
modIncome <- lm(Income ~ AFQT + Educ, data = train)
```

- `AFQT` : Score on proficiency test 25 years before survey
- `Educ` : Years of education to time of survey
- `Income` : Income at time of survey

Model Performance

```
test %>%
+     mutate(pred = predict(modIncome, newdata = test),
+            err = pred - Income) %>%
+     summarize(rmse = sqrt(mean(err^2)),
+               rms.relerr = sqrt(mean((err/Income)^2)))
```

RMSE	RMS-relative error
36,819.39	3.295189

Model log(Income)

```
modLogIncome <- lm(log(Income) ~ AFQT + Educ, data = train)
```

Model Performance

```
test %>%
+     mutate(predlog = predict(modLogIncome, newdata = test),
+            pred = exp(predlog),
+            err = pred - Income) %>%
+     summarize(rmse = sqrt(mean(err^2)),
+               rms.relerr = sqrt(mean((err/Income)^2)))
```

RMSE	RMS-relative error
38,906.61	2.276865

Compare Errors

`log(Income)` model: smaller RMS-relative error, larger RMSE

Model	RMSE	RMS-relative error
On `Income`	36,819.39	3.295189
On `log(Income)`	38,906.61	2.276865

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

Transforming inputs before modeling

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector LLC

datacamp

Why To Transform Input Variables

- Domain knowledge/synthetic variables
  - $Intelligence \sim \frac{mass.brain}{mass.body^{2/3}}$

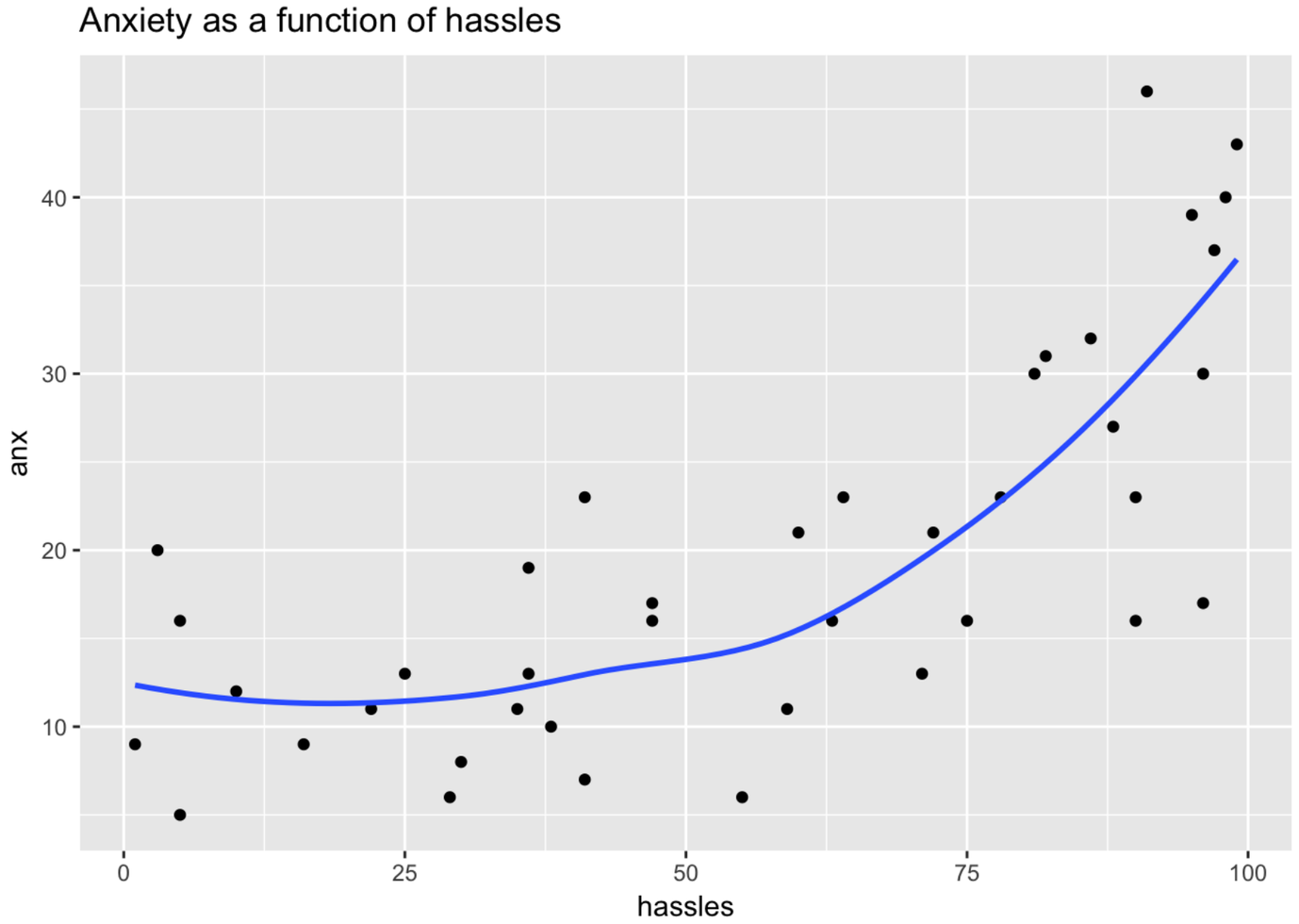
Why To Transform Input Variables

- Domain knowledge/synthetic variables
  - $Intelligence \sim \frac{mass.brain}{mass.body^{2/3}}$
- Pragmatic reasons
  - Log transform to reduce dynamic range
  - Log transform because meaningful changes in variable are multiplicative

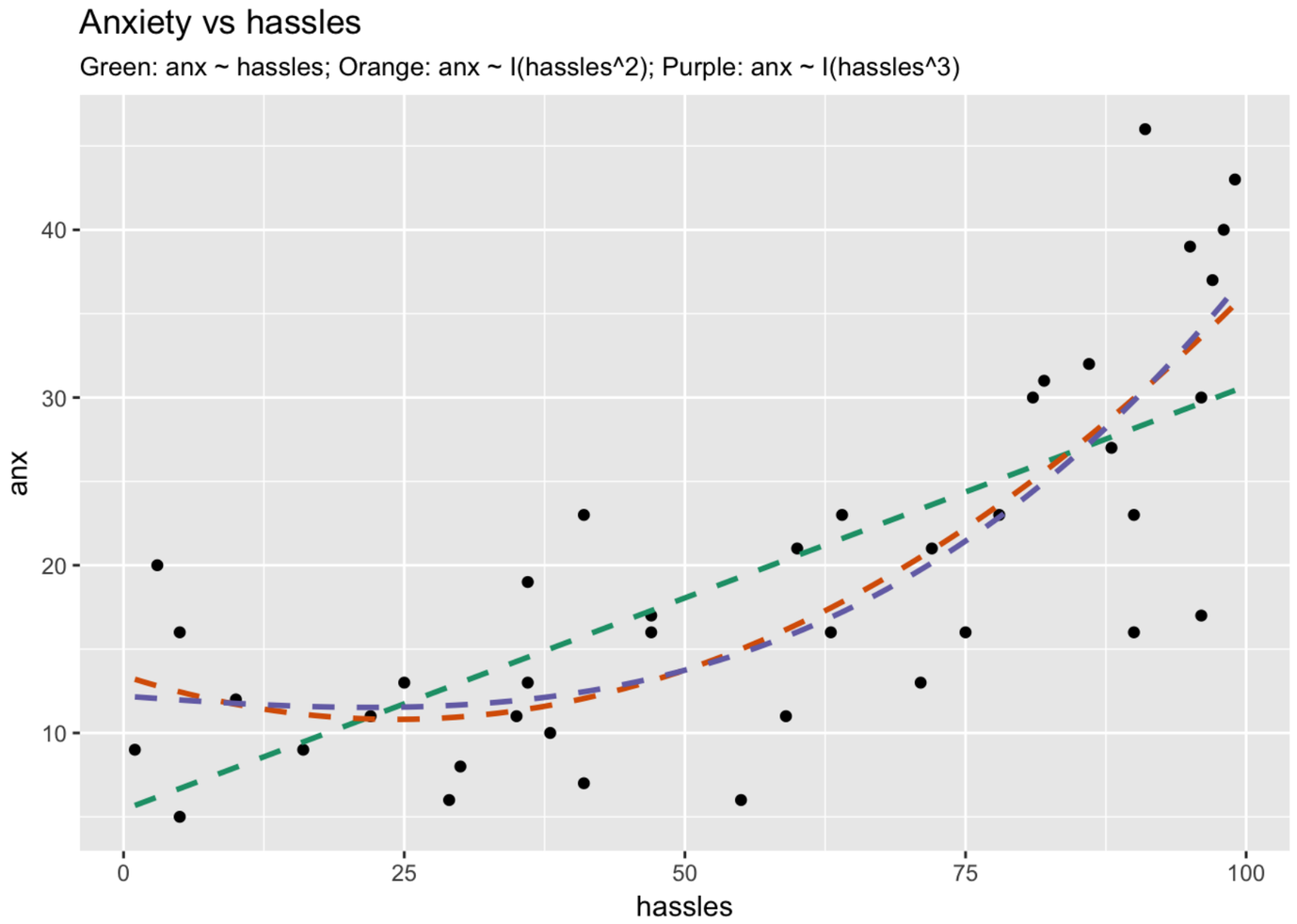
Why To Transform Input Variables

- Domain knowledge/synthetic variables
  - $Intelligence \sim \frac{mass.brain}{mass.body^{2/3}}$
- Pragmatic reasons
  - Log transform to reduce dynamic range
  - Log transform because meaningful changes in variable are multiplicative
  - y approximately linear in $f(x)$ rather than in x

Example: Predicting Anxiety



Transforming the hassles variable



Different possible fits

Which is best?

- `anx ~ I(hassles^2)`
- `anx ~ I(hassles^3)`
- `anx ~ I(hassles^2) + I(hassles^3)`
- `anx ~ exp(hassles)`
- ...

`I()` : treat an expression literally (not as an interaction)

Compare different models

Linear, Quadratic, and Cubic models

```
mod_lin <- lm(anx ~ hassles, hassleframe)
summary(mod_lin)$r.squared
```

```
0.5334847
```

```
mod_quad <- lm(anx ~ I(hassles^2), hassleframe)
summary(mod_quad)$r.squared
```

```
0.6241029
```

```
mod_tritic <- lm(anx ~ I(hassles^3), hassleframe)
summary(mod_tritic)$r.squared
```

```
0.6474421
```

Compare different models

Use cross-validation to evaluate the models

Model	RMSE
Linear ($hassles$)	7.69
Quadratic ($hassles^2$)	6.89
Cubic ($hassles^3$)	6.70

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

Logistic regression to predict probabilities

SUPERVISED LEARNING IN R: REGRESSION

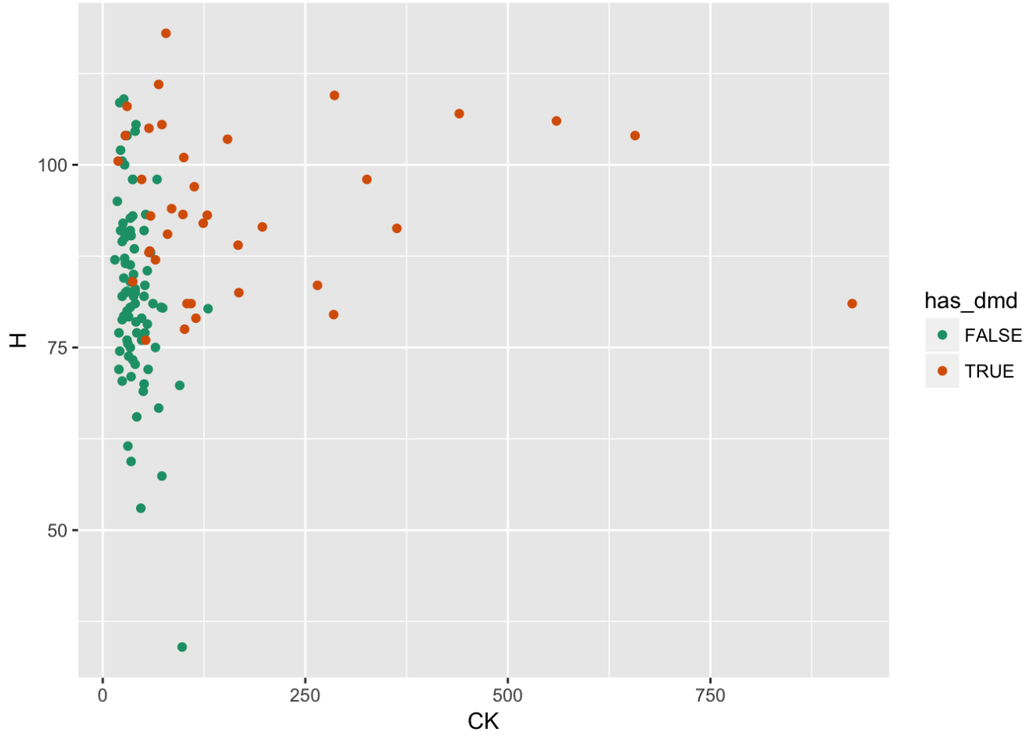
Nina Zumel and John Mount
Win-Vector LLC

datacamp

Predicting Probabilities

- Predicting *whether* an event occurs (yes/no): **classification**
- Predicting *the probability* that an event occurs: **regression**
- Linear regression: predicts values in $[-\infty, \infty]$
- Probabilities: limited to [0,1] interval
  - So we'll call it non-linear

Example: Predicting Duchenne Muscular Dystrophy (DMD)



- outcome: `has_dmd` inputs: `CK` , `H`

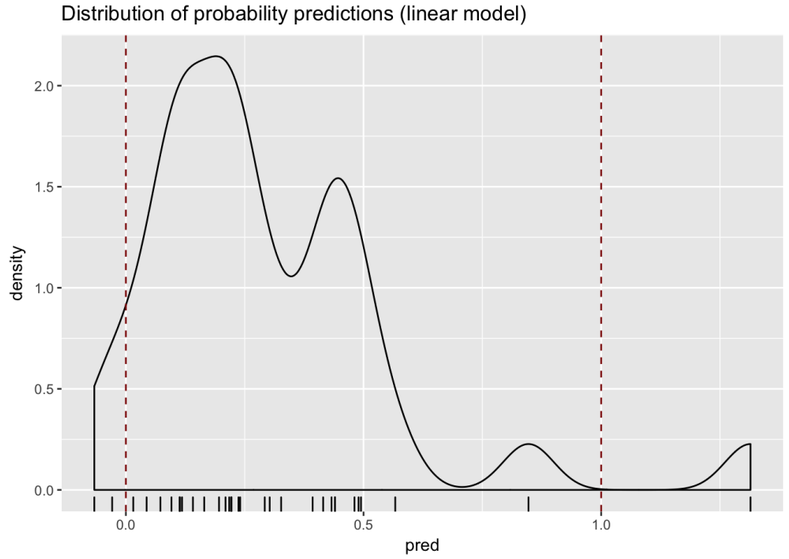
A Linear Regression Model

```
model <- lm(has_dmd ~ CK + H,
            data = train)

test$pred <- predict(
    model,
    newdata = test
)
```

outcome: `has_dmd` $\in \{0,1\}$

- 0: FALSE
- 1: TRUE

Model predicts values outside the range [0:1]



Logistic Regression

$$log(\frac{p}{1-p}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...$$

```
glm(formula, data, family = binomial)
```

- Generalized linear model
- Assumes inputs additive, linear in *log-odds*: $log(p/(1-p))$
- family: describes error distribution of the model
  - logistic regression: `family = binomial`

DMD model

```
model <- glm(has_dmd ~ CK + H, data = train, family = binomial)
```

- outcome: two classes, e.g. a and b
- model returns $Prob(b)$
  - Recommend: 0/1 or FALSE/TRUE

Interpreting Logistic Regression Models

```
model
```

```
Call:  glm(formula = has_dmd ~ CK + H, family = binomial, data = train)

Coefficients:
(Intercept)           CK            H
  -16.22046      0.07128      0.12552

Degrees of Freedom: 86 Total (i.e. Null);  84 Residual
Null Deviance:       110.8
Residual Deviance: 45.16     AIC: 51.16
```

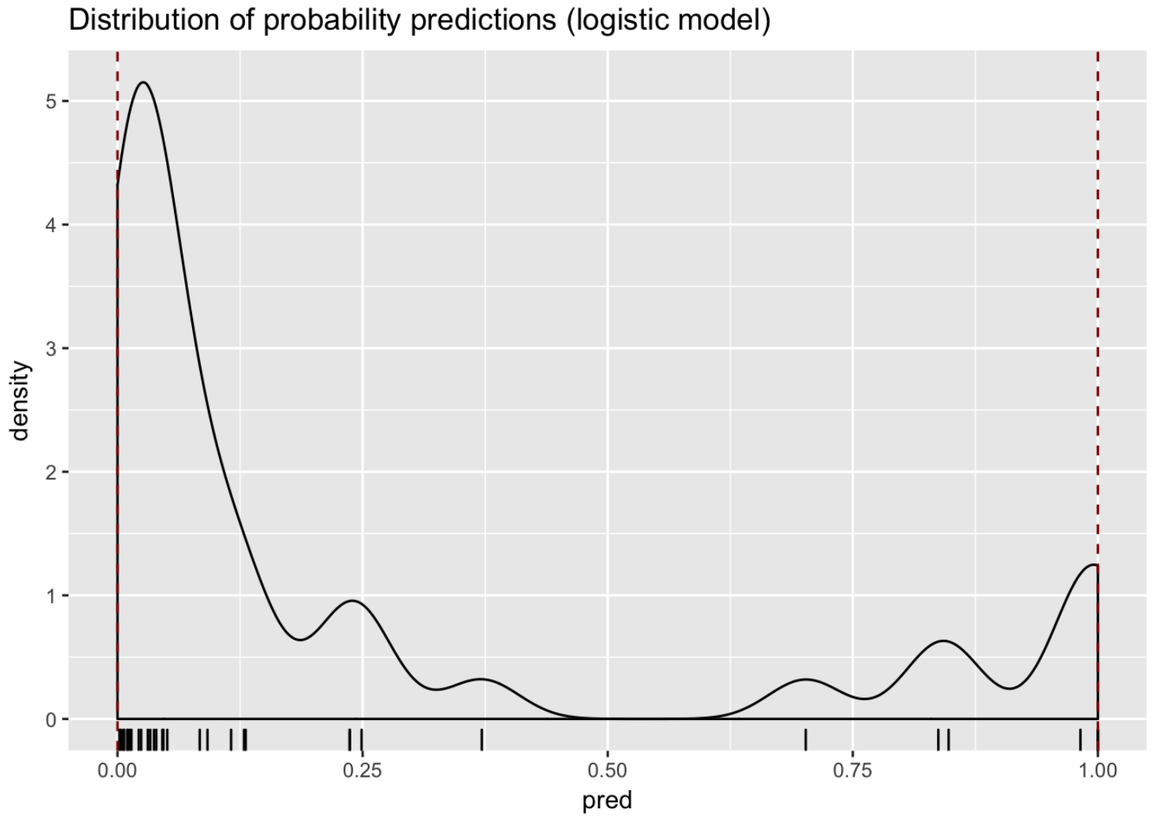
Predicting with a glm() model

```
predict(model, newdata, type = "response")
```

- `newdata` : by default, training data
- To get probabilities: use **`type = "response"`**
  - By default: returns log-odds

DMD Model

```
model <- glm(has_dmd ~ CK + H, data = train, family = binomial)
test$pred <- predict(model, newdata = test, type = "response")
```



Evaluating a logistic regression model: pseudo-R^2

$$R^2 = 1 - \frac{RSS}{SS_{Tot}}$$

$$pseudoR^2 = 1 - \frac{deviance}{null.deviance}$$

- Deviance: analogous to variance (RSS)
- Null deviance: Similar to SS_{Tot}
- pseudo R^2: Deviance explained

Pseudo-R^2 on Training data

Using `broom::glance()`

```
glance(model) %>%
  summarize(pR2 = 1 - deviance/null.deviance)
```

```
   pseudoR2
1 0.5922402
```

Using `sigr::wrapChiSqTest()`

```
wrapChiSqTest(model)
```

```
"... pseudo-R2=0.59 ..."
```

Pseudo-R^2 on Test data

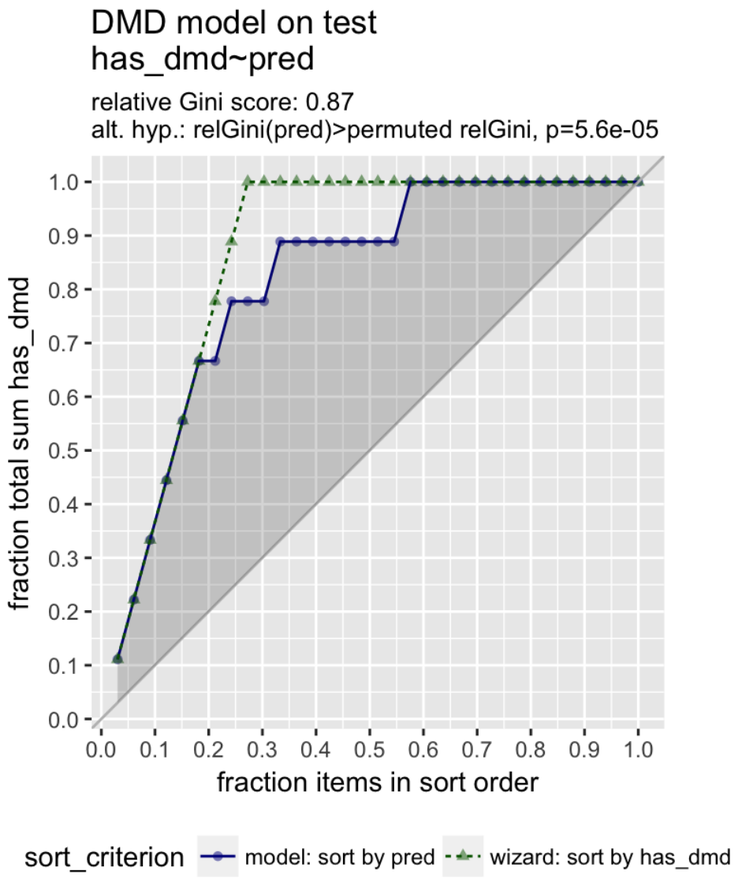
```
# Test data
test %>%
  mutate(pred = predict(model, newdata = test, type = "response")) %>%
  wrapChiSqTest("pred", "has_dmd", TRUE)
```

Arguments:

- data frame
- prediction column name
- outcome column name
- target value (target event)

The Gain Curve Plot

```
GainCurvePlot(test, "pred","has_dmd", "DMD model on test")
```



Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

Poisson and quasipoisson regression to predict counts

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector, LLC

datacamp

Predicting Counts

- Linear regression: predicts values in $[-\infty, \infty]$
- Counts: integers in range $[0, \infty]$

Poisson/Quasipoisson Regression

```
glm(formula, data, family)
```

- family: either `poisson` or `quasipoisson`
- inputs additive and linear in log(count)

Poisson/Quasipoisson Regression

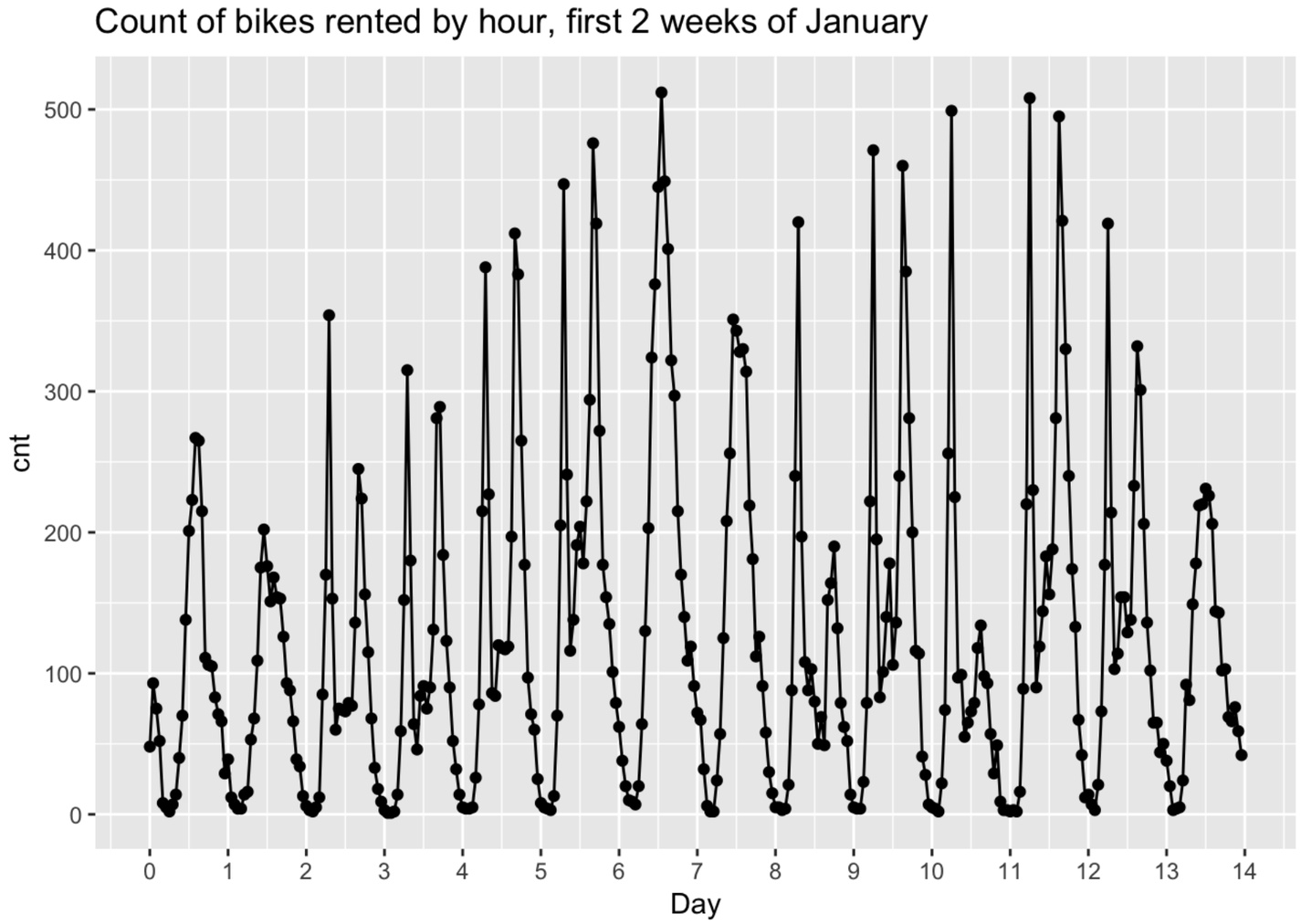
```
glm(formula, data, family)
```

- family: either `poisson` or `quasipoisson`
- inputs additive and linear in log(count)
- outcome: *integer*
  - counts: e.g. number of traffic tickets a driver gets
  - rates: e.g. number of website hits/day
- prediction: expected *rate* or *intensity* (not integral)
  - expected # traffic tickets; expected hits/day

Poisson vs. Quasipoisson

- Poisson assumes that `mean(y) = var(y)`
- If `var(y)` much different from `mean(y)` - quasipoisson
- Generally requires a large sample size
- If rates/counts >> 0 - regular regression is fine

Example: Predicting Bike Rentals



Fit the model

```
bikesJan %>%
  summarize(mean = mean(cnt), var = var(cnt))
```

```
      mean      var
1 130.5587 14351.25
```

Since `var(cnt)` >> `mean(cnt)` $\rightarrow$ *use quasipoisson*

```
fmla <- cnt ~ hr + holiday + workingday +
  weathersit + temp + atemp + hum + windspeed

model <- glm(fmla, data = bikesJan, family = quasipoisson)
```

Check model fit

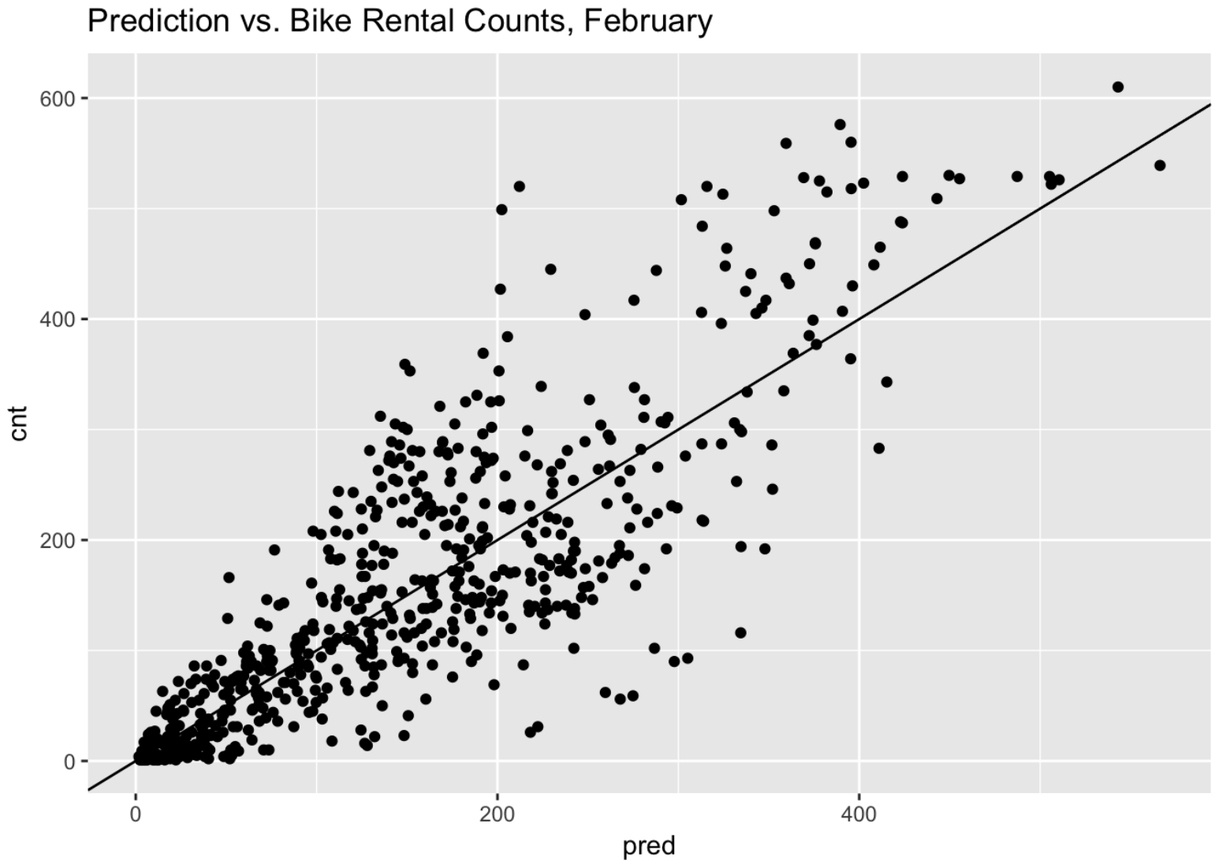
$$pseudoR^2 = 1 - \frac{deviance}{null.deviance}$$

```
glance(model) %>%
  summarize(pseudoR2 = 1 - deviance/null.deviance)
```

```
   pseudoR2
1 0.7654358
```

Predicting from the model

```
predict(model, newdata = bikesFeb, type = "response")
```



Evaluate the model

You can evaluate count models by RMSE

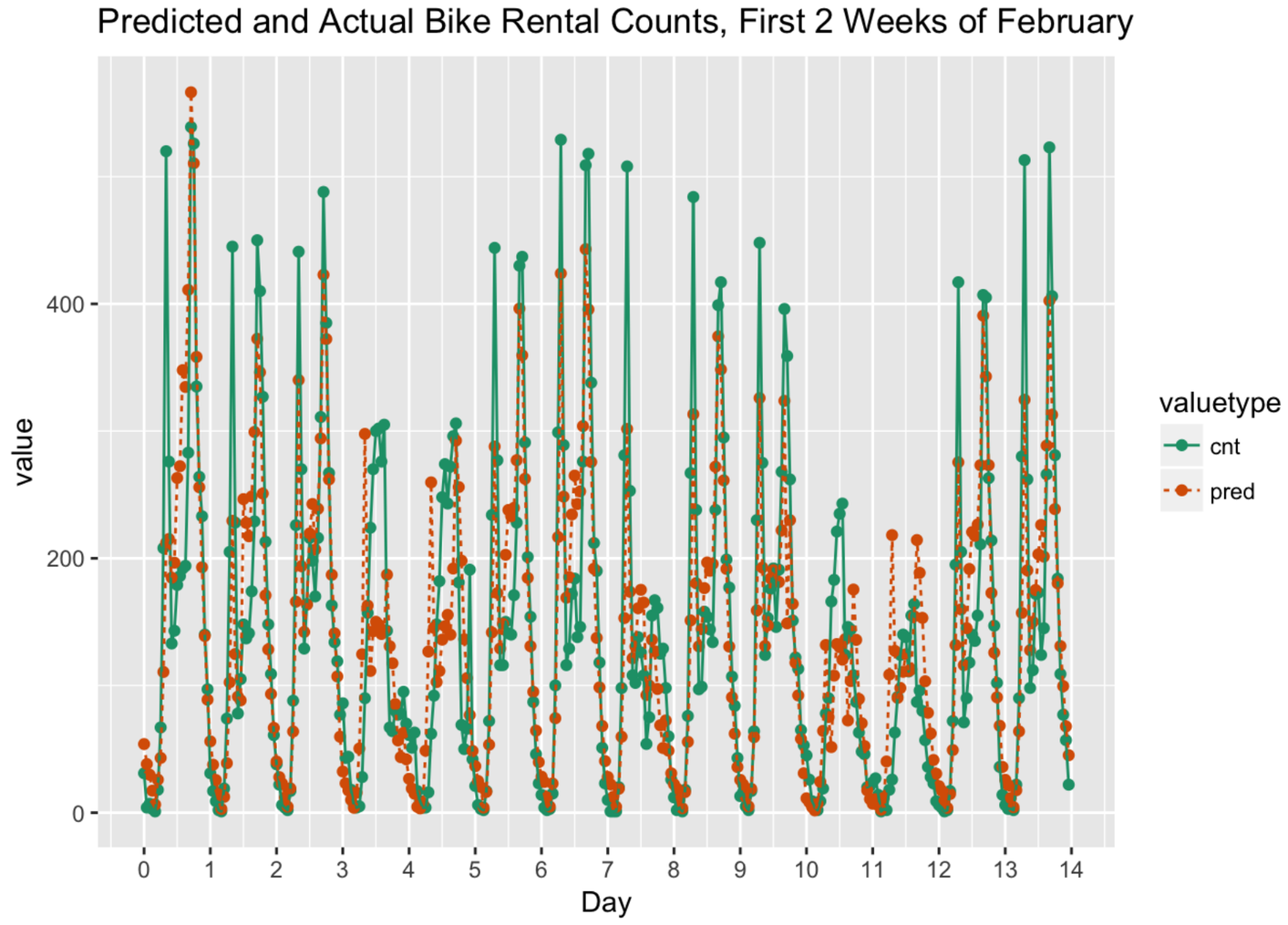
```
bikesFeb %>%
  mutate(residual = pred - cnt) %>%
  summarize(rmse = sqrt(mean(residual^2)))
```

```
      rmse
1 69.32869
```

```
sd(bikesFeb$cnt)
```

```
134.2865
```

Compare Predictions and Actual Outcomes



Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

GAM to learn non-linear transformations

SUPERVISED LEARNING IN R: REGRESSION

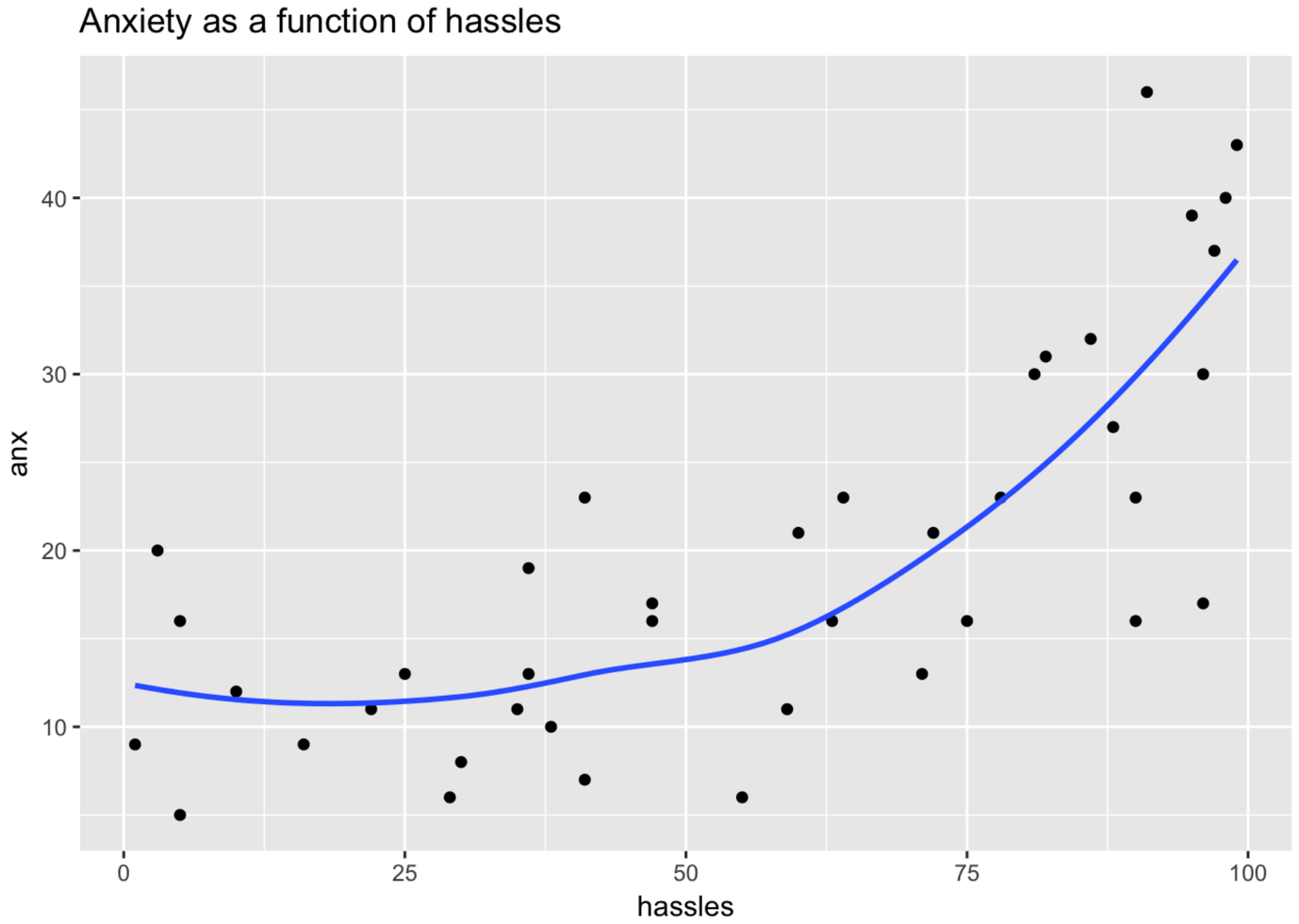
Nina Zumel and John Mount
Win-Vector, LLC

datacamp

Generalized Additive Models (GAMs)

$$y \sim b0 + s1(x1) + s2(x2) + ...$$

Learning Non-linear Relationships



gam() in the mgcv package

```
gam(formula, family, data)
```

family:

- gaussian (default): "regular" regression
- binomial: probabilities
- poisson/quasipoisson: counts

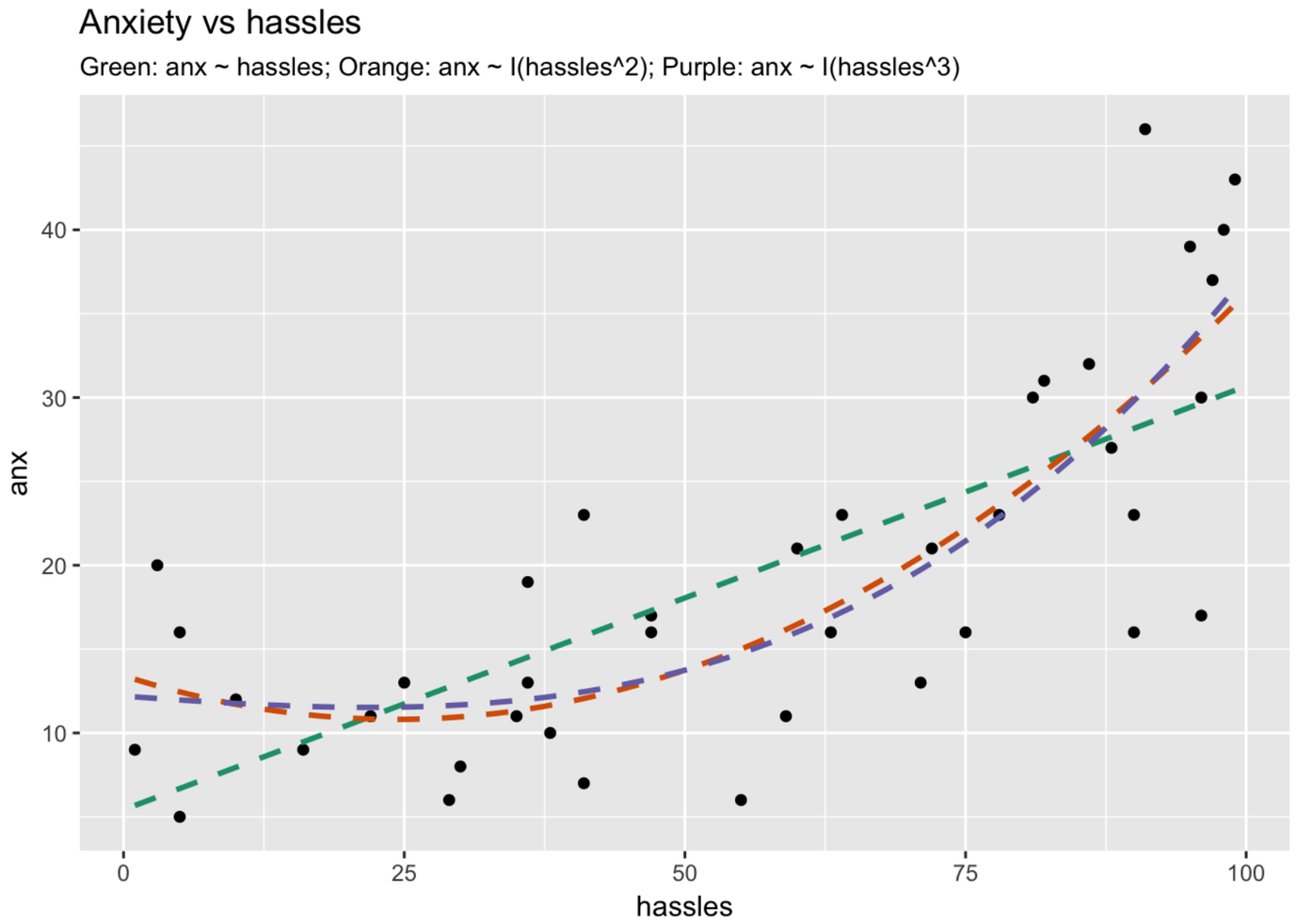
Best for larger datasets

The s() function

```
anx ~ s(hassles)
```

- `s()` designates that variable should be non-linear
- Use `s()` with continuous variables
  - More than about 10 unique values

Revisit the hassles data



Revisit the hassles data

Model	RMSE (cross-val)	R^2 (training)
Linear ($hassles$)	7.69	0.53
Quadratic ($hassles^2$)	6.89	0.63
Cubic ($hassles^3$)	**6.70**	**0.65**

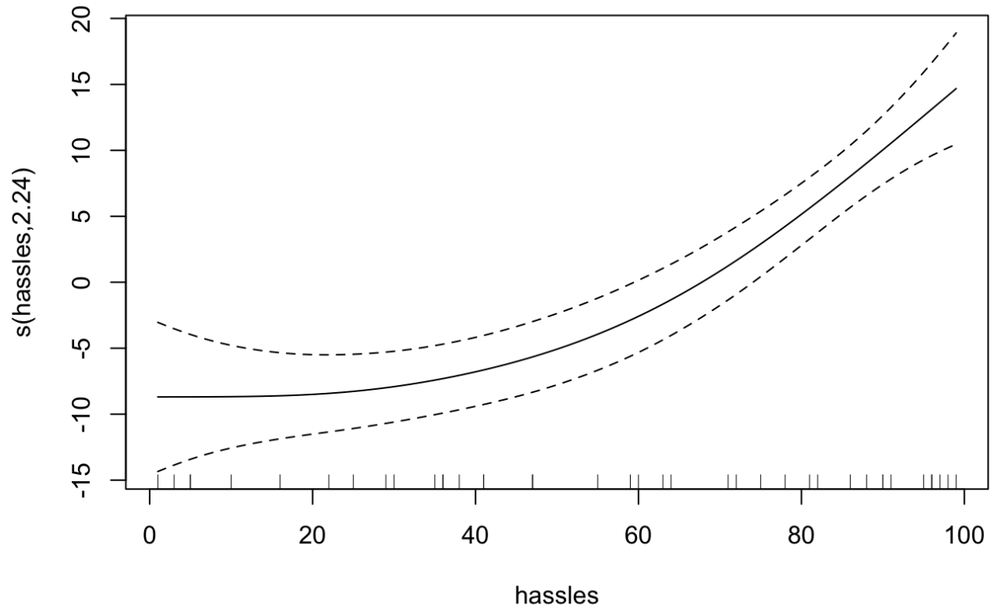
GAM of the hassles data

```
model <- gam(
  anx ~ s(hassles),
  data = hassleframe,
  family = gaussian
)

summary(model)
```

```
...

R-sq.(adj) =  0.619   Deviance explained = 64.1%
GCV = 49.132  Scale est. = 45.153    n = 40
```

Examining the Transformations

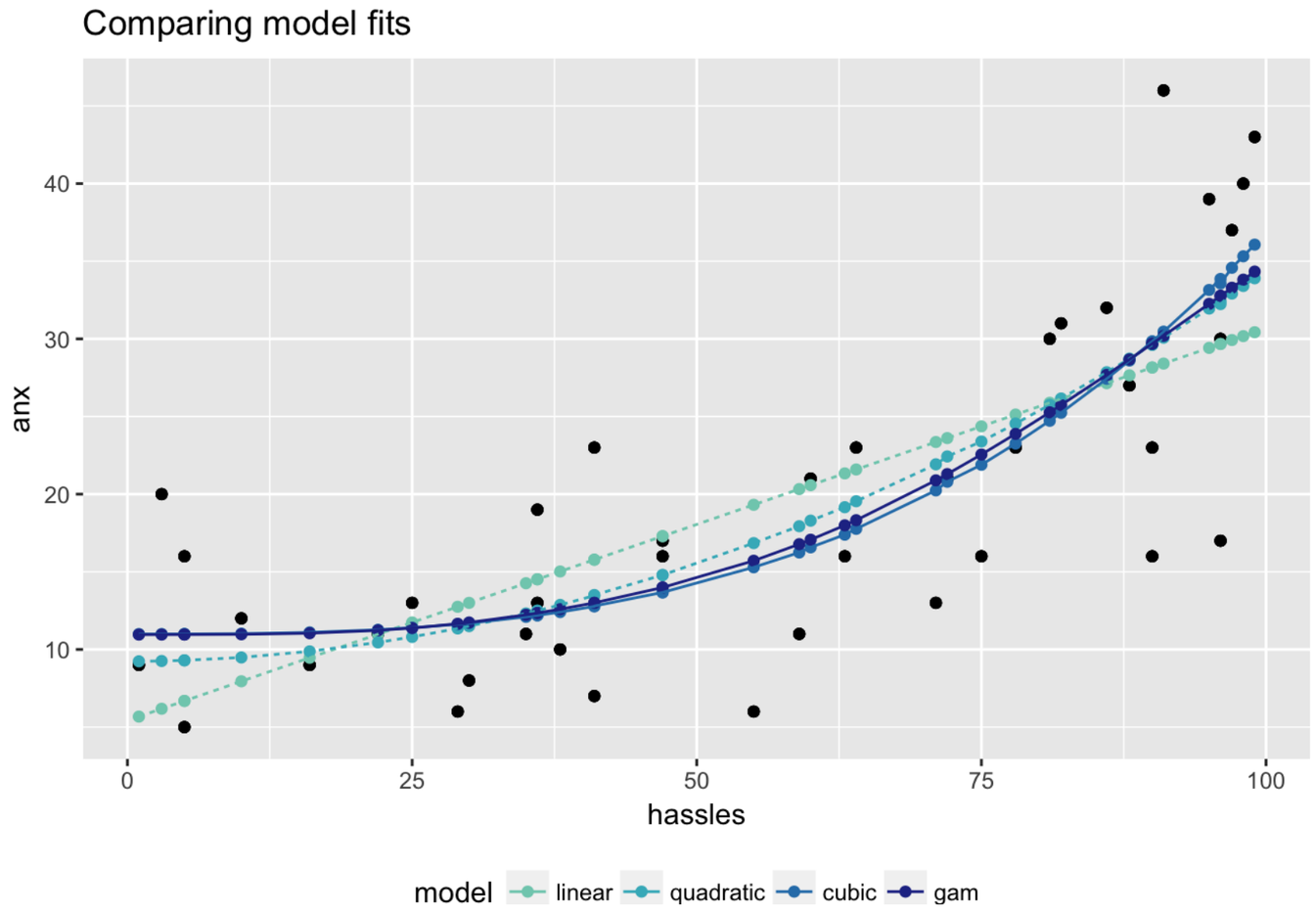
```
plot(model)
```



y values: `predict(model, type = "terms")`

Predicting with the Model

```
predict(model, newdata = hassleframe, type = "response")
```



Comparing out-of-sample performance

Knowing the correct transformation is best, but GAM is useful when transformation isn't known

Model	RMSE (cross-val)	R^2 (training)
Linear ($hassles$)	7.69	0.53
Quadratic ($hassles^2$)	6.89	0.63
Cubic ($hassles^3$)	6.70	0.65
GAM	**7.06**	0.64

- Small dataset $\rightarrow$ noisier GAM

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

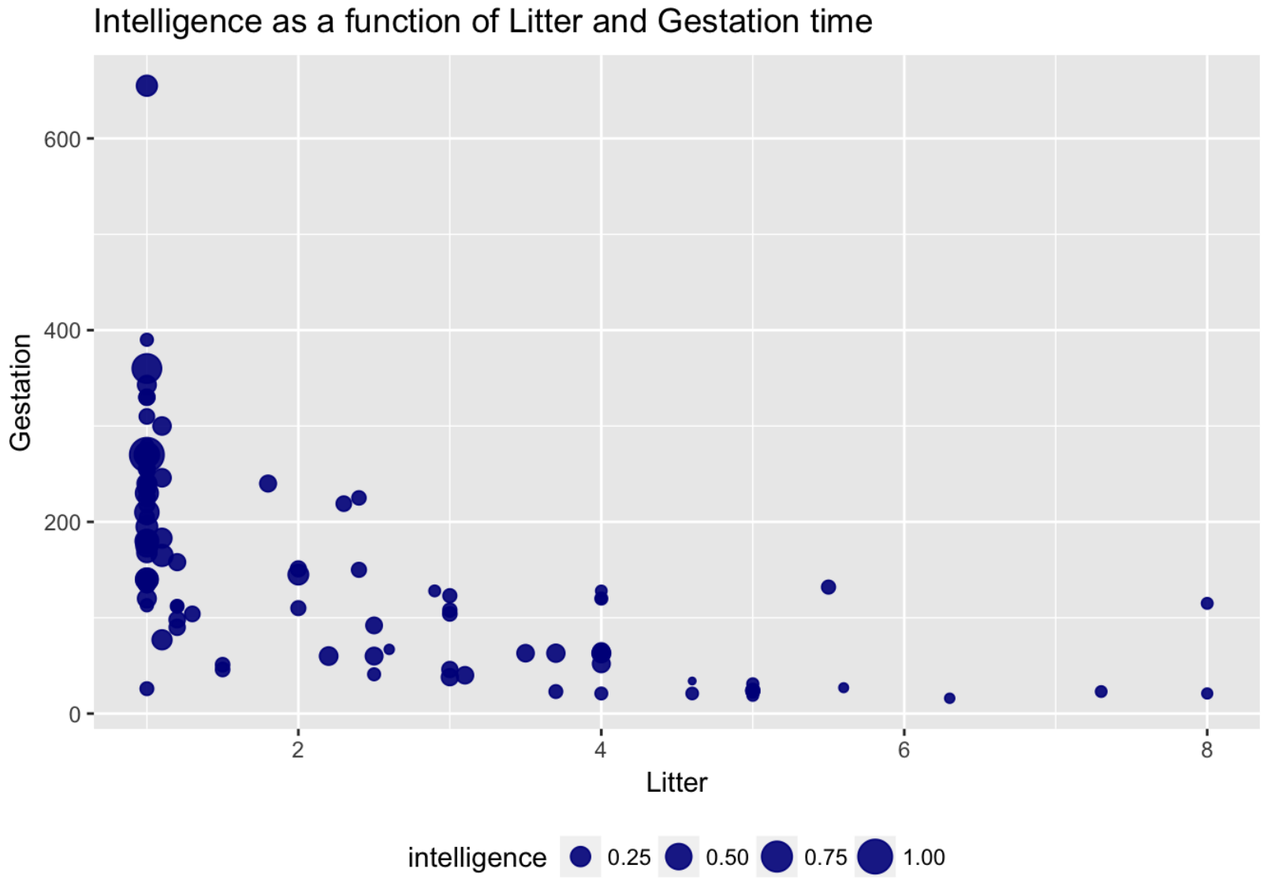
The intuition behind tree-based methods

SUPERVISED LEARNING IN R: REGRESSION

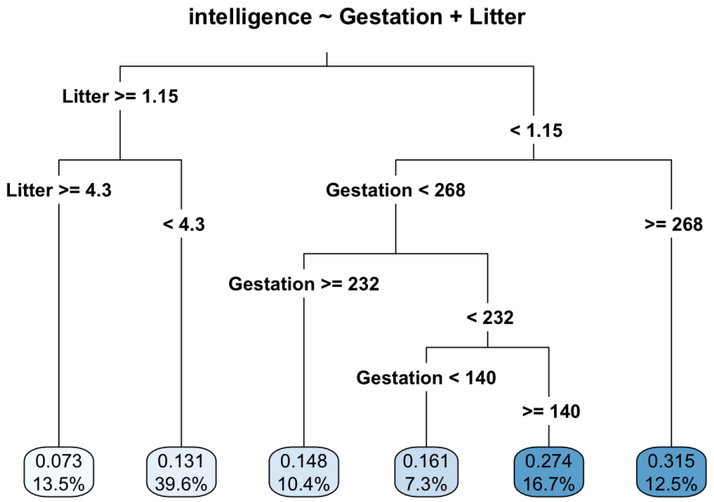
Nina Zumel and John Mount
Win-Vector, LLC

datacamp

Example: Predict animal intelligence from Gestation Time and Litter Size



Decision Trees



Rules of the form:

- *if a AND b AND c THEN y*

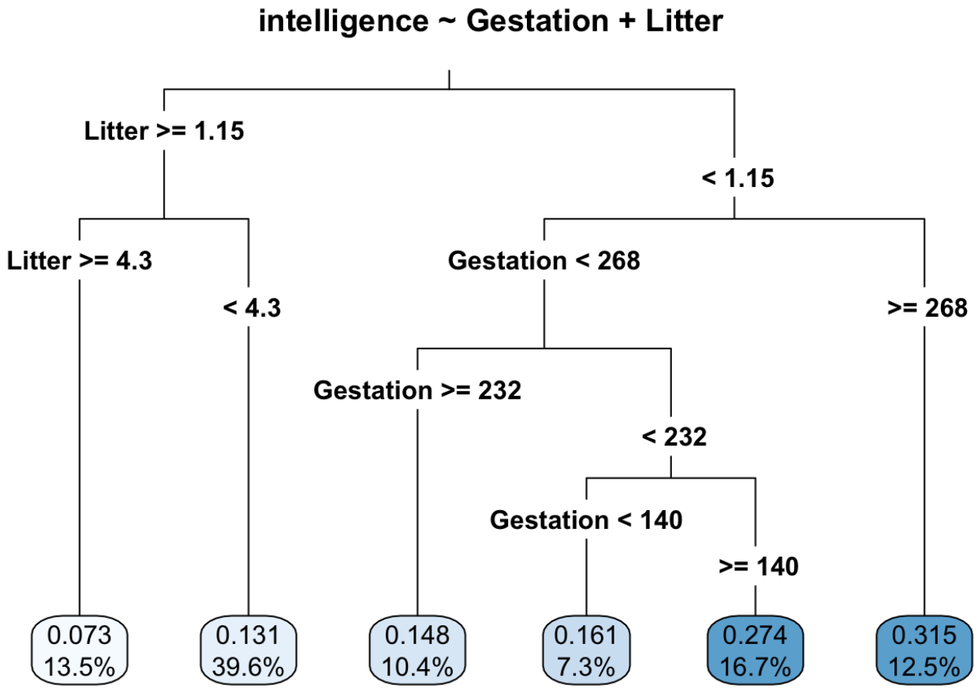
Non-linear concepts

- intervals
- non-monotonic relationships

non-additive interactions

- AND: similar to multiplication

Decision Trees



- IF Litter < 1.15 AND Gestation $\geq$ 268 $\rightarrow$ intelligence = 0.315
- IF Litter IN [1.15, 4.3) $\rightarrow$ intelligence = 0.131

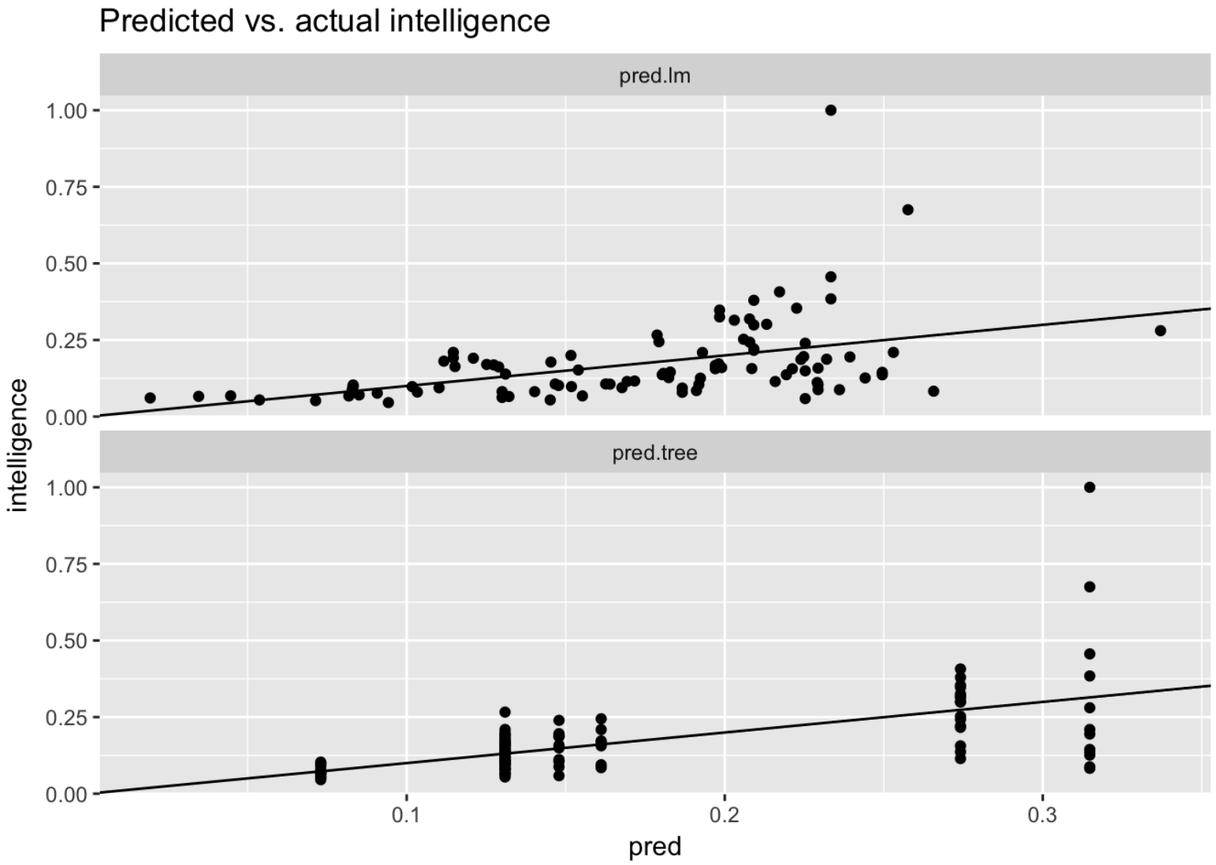
Decision Trees

Pro: Trees Have an *Expressive Concept Space*

Model	RMSE
linear	0.1200419
tree	0.1072732

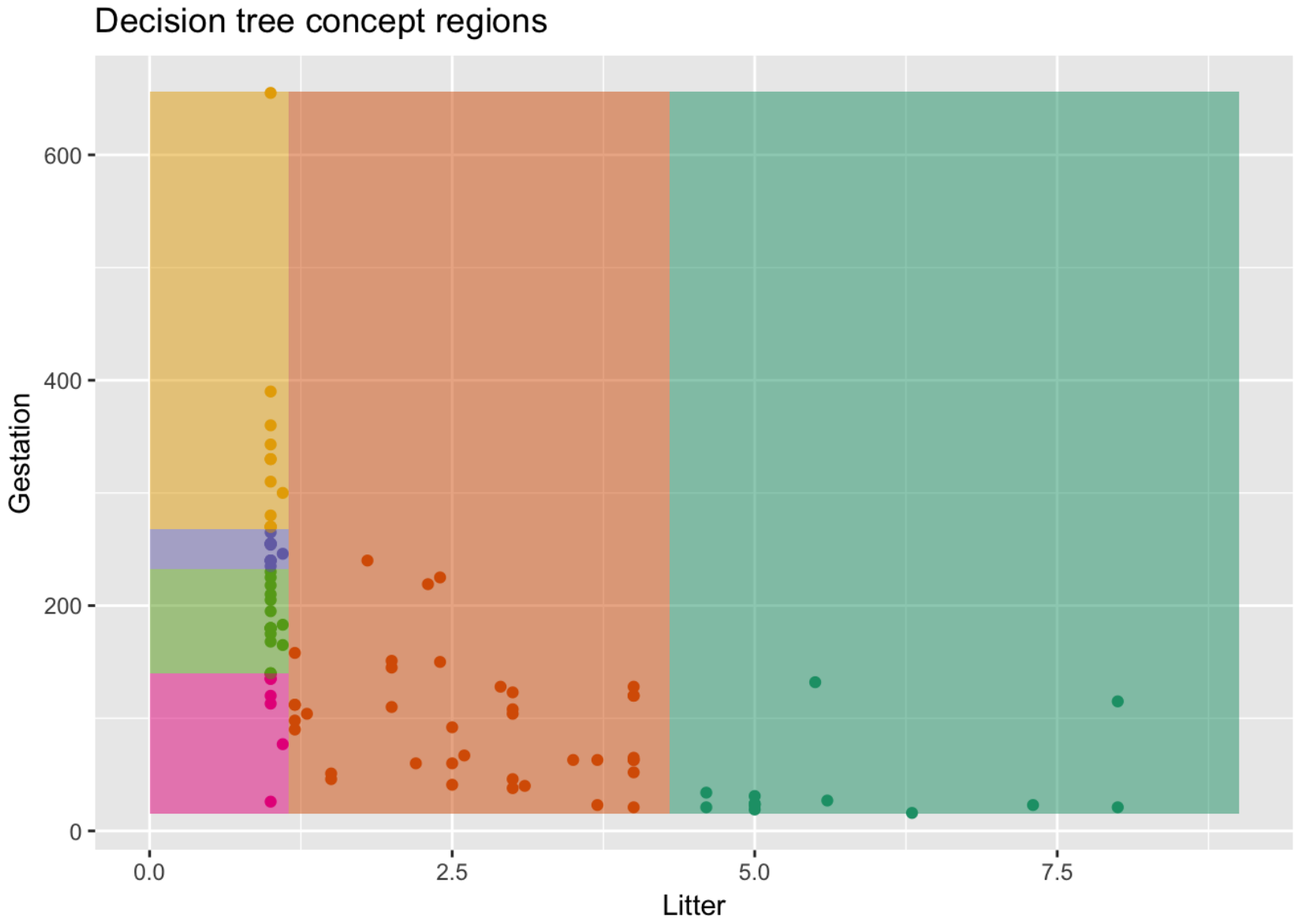
Decision Trees

Con: *Coarse-Grained Predictions*



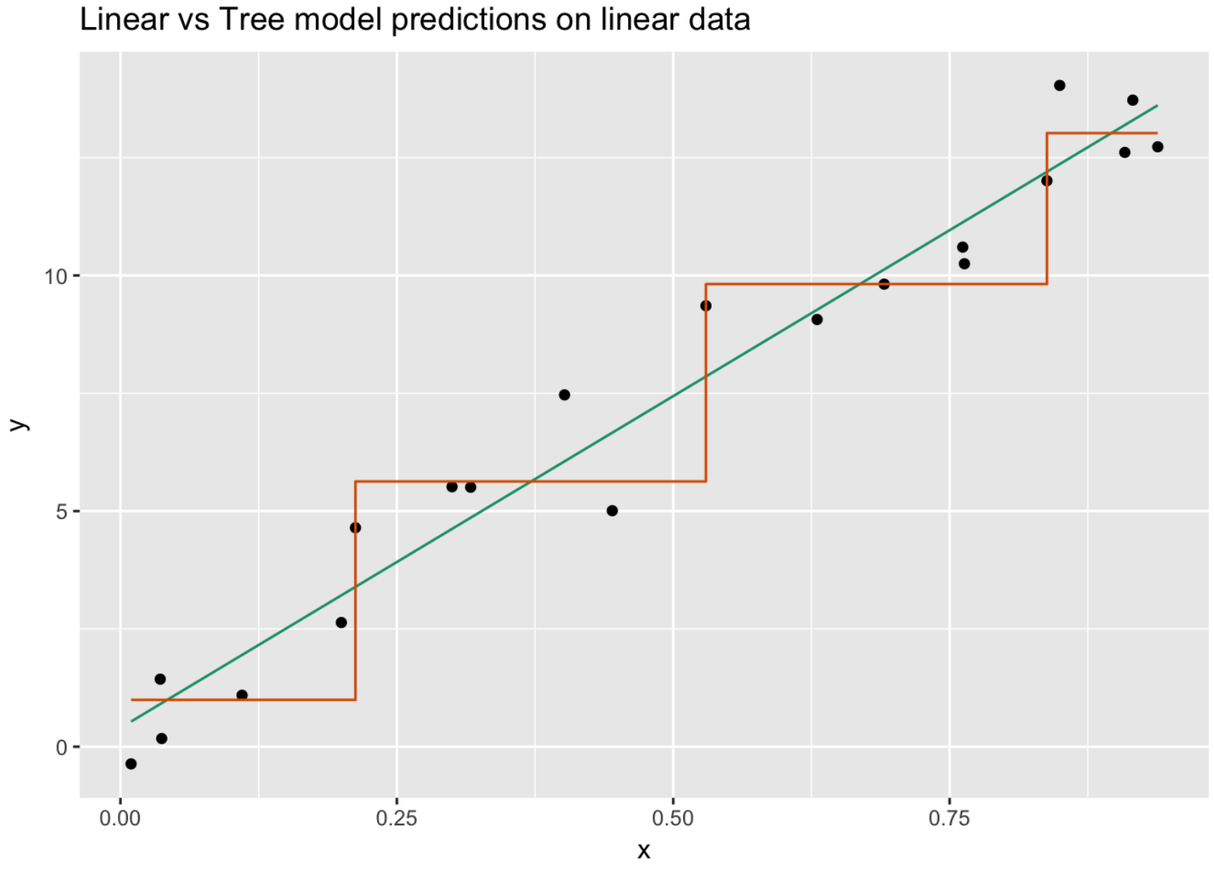
It's Hard for Trees to Express Linear Relationships

Trees Predict Axis-Aligned Regions



It's Hard for Trees to Express Linear Relationships

It's Hard to Express Lines with Steps

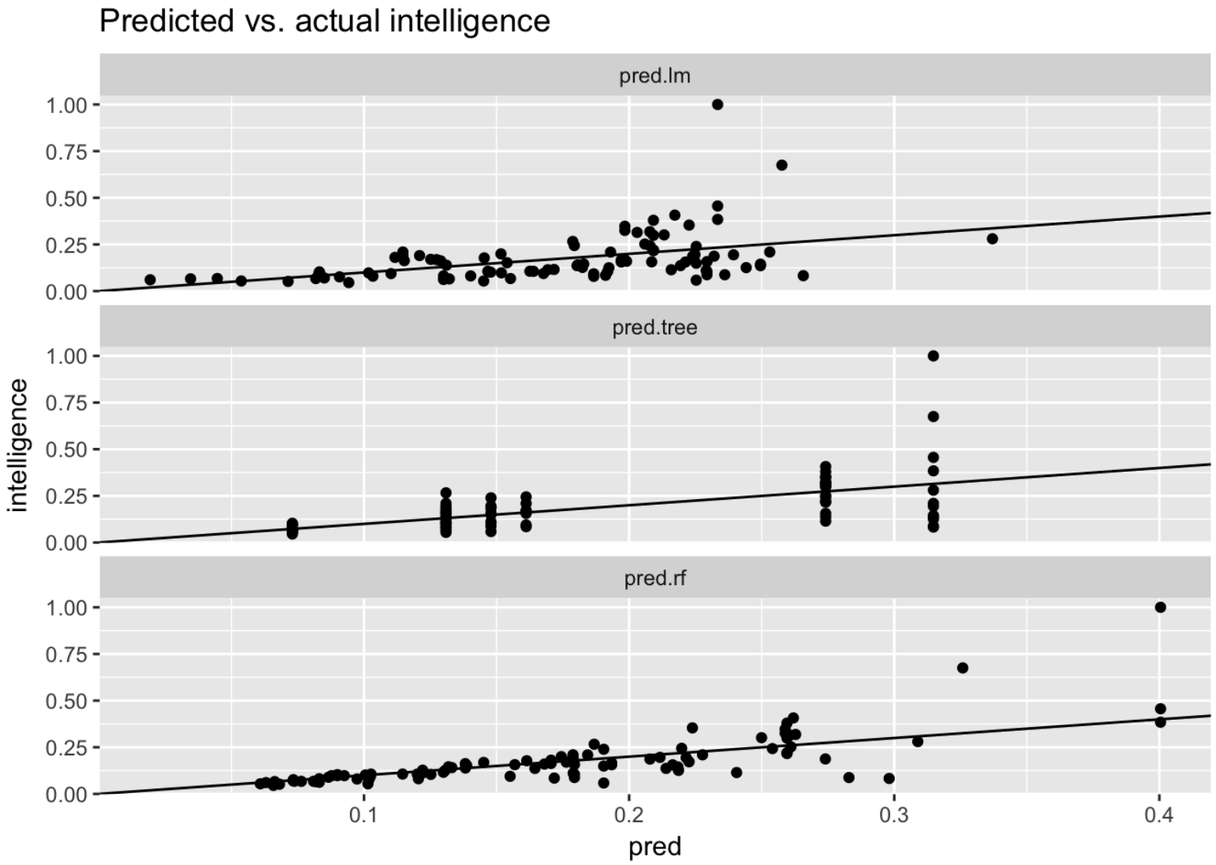


Other Issues with Trees

- Tree with too many splits (deep tree):
  - Too complex - danger of overfit
- Tree with too few splits (shallow tree):
  - Predictions too coarse-grained

Ensembles of Trees

Ensembles Give Finer-grained Predictions than Single Trees



Ensembles of Trees

Ensemble Model Fits Animal Intelligence Data Better than Single Tree

Model	RMSE
linear	0.1200419
tree	0.1072732
random forest	0.0901681

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

Random forests

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector, LCC

Random Forests

Multiple diverse decision trees averaged together

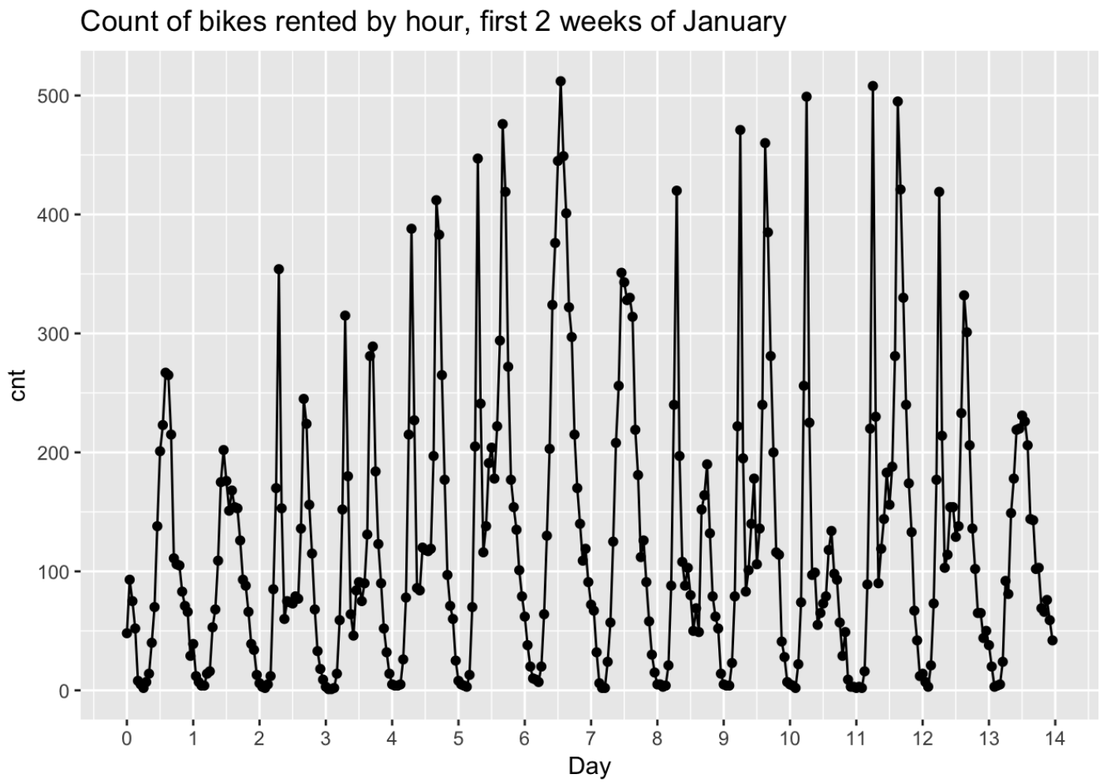
- Reduces overfit
- Increases model expressiveness
- Finer grain predictions

Building a Random Forest Model

1. Draw bootstrapped sample from training data
2. For each sample grow a tree
   - At each node, pick best variable to split on (from a random subset of all variables)
   - Continue until tree is grown
3. To score a datum, evaluate it with all the trees and average the results.

Example: Bike Rental Data

```
cnt ~ hr + holiday + workingday +
  weathersit + temp + atemp + hum + windspeed
```



Random Forests with ranger()

```
model <- ranger(fmla, bikesJan,
                num.trees = 500,
                respect.unordered.factors = "order")
```

- `formula`, `data`
- `num.trees` (default 500) - use at least 200
- `mtry` - number of variables to try at each node
  - default: square root of the total number of variables
- `respect.unordered.factors` - recommend set to "order"
  - "safe" hashing of categorical variables

Random Forests with ranger()

```
model
```

```
Ranger result
...
OOB prediction error (MSE):       3103.623
R squared (OOB):                  0.7837386
```

Random forest algorithm returns estimates of out-of-sample performance.

Predicting with a ranger() model

```
bikesFeb$pred <- predict(model, bikesFeb)$predictions
```

`predict()` inputs:

- model
- data

Predictions can be accessed in the element `predictions` .

Evaluating the model

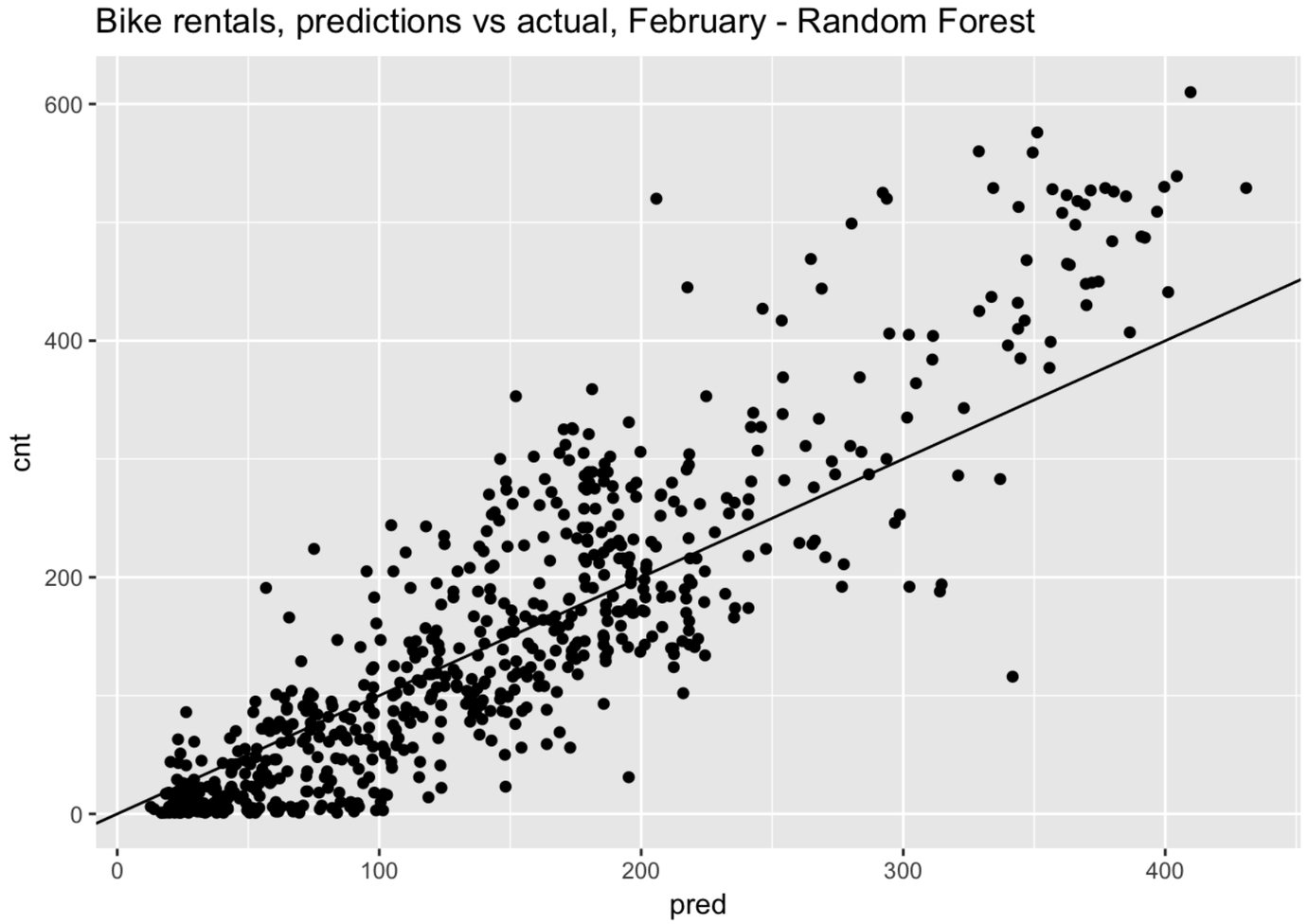
Calculate RMSE:

```
bikesFeb %>%
  mutate(residual = pred - cnt) %>%
  summarize(rmse = sqrt(mean(residual^2)))
```

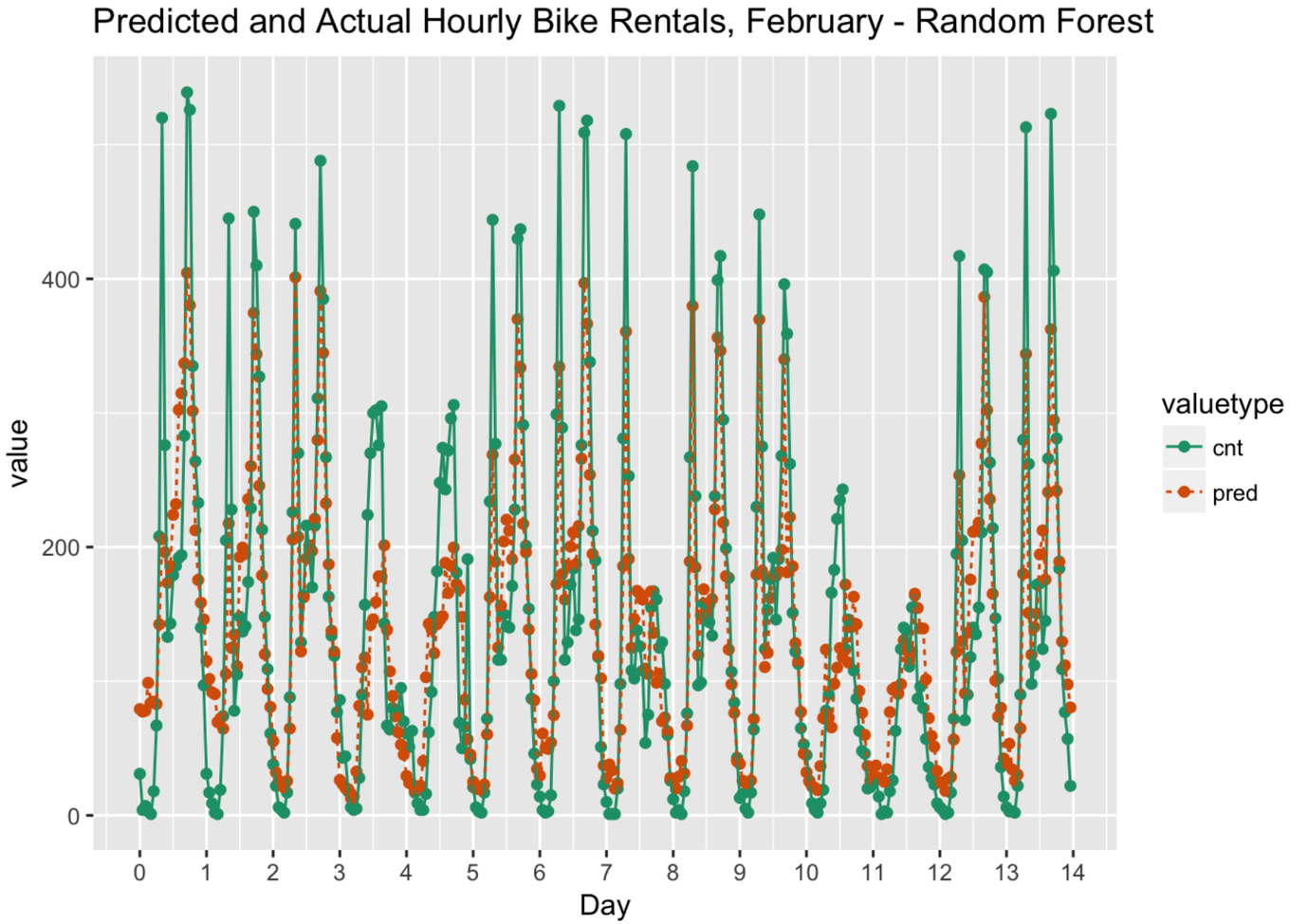
```
      rmse
1 67.15169
```

Model	RMSE
Quasipoisson	69.3
Random forests	67.15

Evaluating the model



Evaluating the model



Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

One-Hot-Encoding Categorical Variables

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector, LLC

datacamp

Why Convert Categoricals Manually?

- Most R functions manage the conversion for you
  - `model.matrix()`
- `xgboost()` does not
  - Must convert categorical variables to numeric representation
- Conversion to indicators: *one-hot encoding*

One-hot-encoding and data cleaning with `vtreat`

Basic idea:

- `designTreatmentsZ()` to design a *treatment plan* from the training data, then
- `prepare()` to created "clean" data
  - all numerical
  - no missing values
    - use `prepare()` with treatment plan for all future data

A Small vtreat Example

Training Data

x	u	y
one	44	0.4855671
two	24	1.3683726
three	66	2.0352837
two	22	1.6396267

Test Data

x	u	y
one	5	2.6488148
three	12	1.5012938
one	56	0.1993731
two	28	1.2778516

Create the Treatment Plan

```
vars <- c("x", "u")
treatplan <- designTreatmentsZ(dframe, varslist, verbose = FALSE)
```

Inputs to `designTreatmentsZ()`

- `dframe` : training data
- `varlist` : list of input variable names
- set verbose = FALSE to suppress progress messages

Get the New Variables

The scoreFrame describes the variable mapping and types

```
(scoreFrame <- treatplan$scoreFrame %>%
     select(varName, origName, code))
```

```
        varName origName  code
1   x_lev_x.one        x   lev
2 x_lev_x.three        x   lev
3   x_lev_x.two        x   lev
4        x_catP        x  catP
5       u_clean        u clean
```

Get the names of the new `lev` and `clean` variables

```
(newvars <- scoreFrame %>%
     filter(code %in% c("clean", "lev")) %>%
     use_series(varName))
```

```
"x_lev_x.one"   "x_lev_x.three" "x_lev_x.two"   "u_clean"
```

Prepare the Training Data for Modeling

```
training.treat <- prepare(treatmentplan, dframe, varRestriction = newvars)
```

Inputs to `prepare()` :

- `treatmentplan` : treatment plan
- `dframe` : data frame
- `varRestriction` : list of variables to prepare (optional)
  - default: prepare all variables

Before and After Data Treatment

Training Data

x	u	y
one	44	0.4855671
two	24	1.3683726
three	66	2.0352837
two	22	1.6396267

Treated Training Data

x_lev_x.one	x_lev_x.three	x_lev_x.two	u_clean
1	0	0	44
0	0	1	24
0	1	0	66
0	0	1	22

Prepare the Test Data Before Model Application

```
(test.treat <- prepare(treatplan, test, varRestriction = newvars))
```

```
  x_lev_x.one x_lev_x.three x_lev_x.two u_clean
1           1             0           0       5
2           0             1           0      12
3           1             0           0      56
4           0             0           1      28
```

vtreat Treatment is Robust

Previously unseen `x` level: *four* *four* encodes to (0, 0, 0)

x	u	y
one	4	0.2331301
two	14	1.9331760
three	66	3.1251029
four	25	4.0332491

```
prepare(treatplan, toomany, ...)
```

x_lev_x.one	x_lev_x.three	x_lev_x.two	u_clean
1	0	0	4
0	0	1	14
0	1	0	66
0	**0**	**0**	25

Let's practice!

SUPERVISED LEARNING IN R: REGRESSION

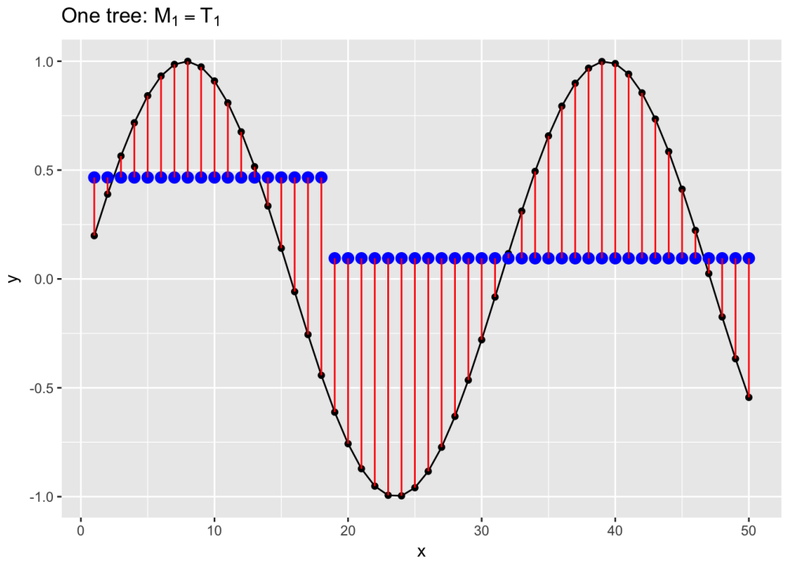
Gradient boosting machines

SUPERVISED LEARNING IN R: REGRESSION

Nina Zumel and John Mount
Win-Vector, LLC

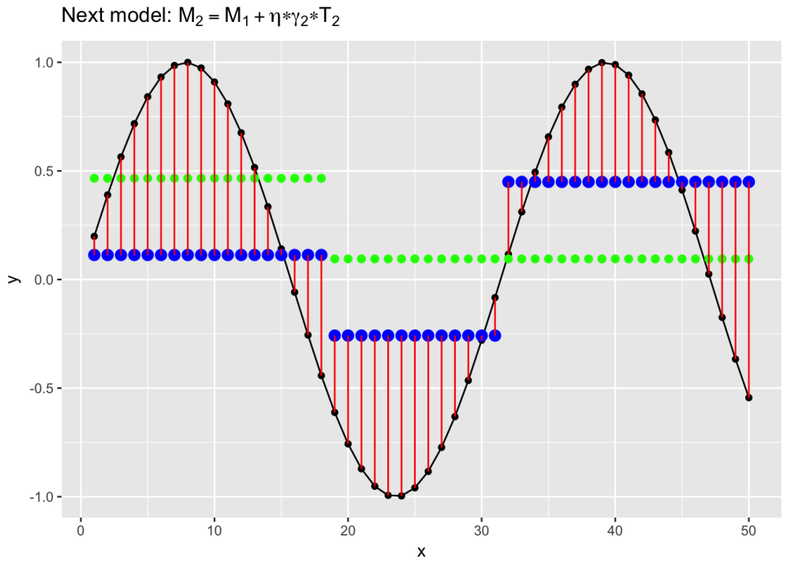
datacamp

How Gradient Boosting Works



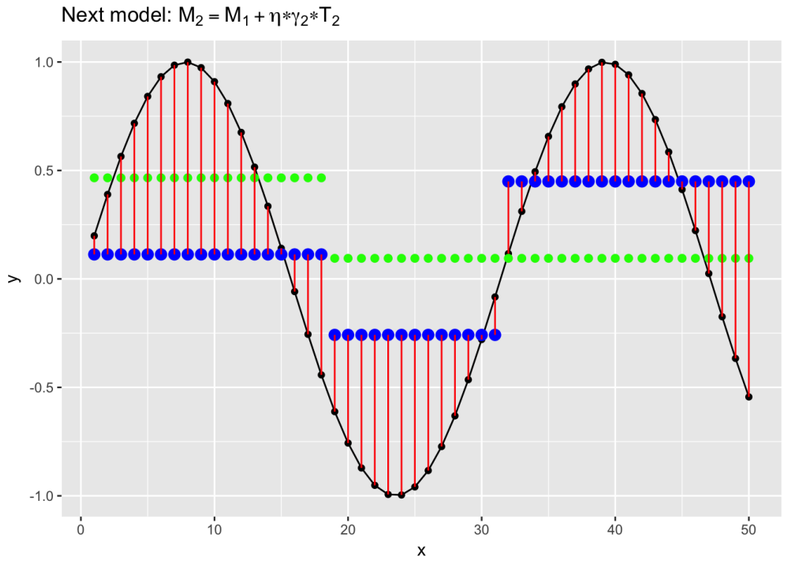
1. Fit a shallow tree T_1 to the data: $M_1 = T_1$

How Gradient Boosting Works



1. Fit a shallow tree T_1 to the data: $M_1 = T_1$
2. Fit a tree T_2 to the residuals. Find γ such that $M_2 = M_1 + \gamma T_2$ is the best fit to data

How Gradient Boosting Works



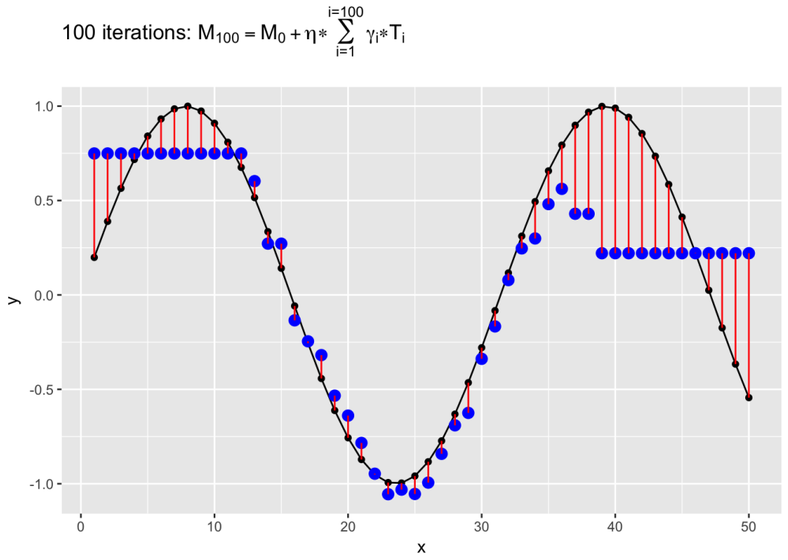
Regularization: learning rate

$\eta \in (0, 1)$

$$M_2 = M_1 + \eta\gamma T_2$$

- Larger η: faster learning
- Smaller η: less risk of overfit

How Gradient Boosting Works

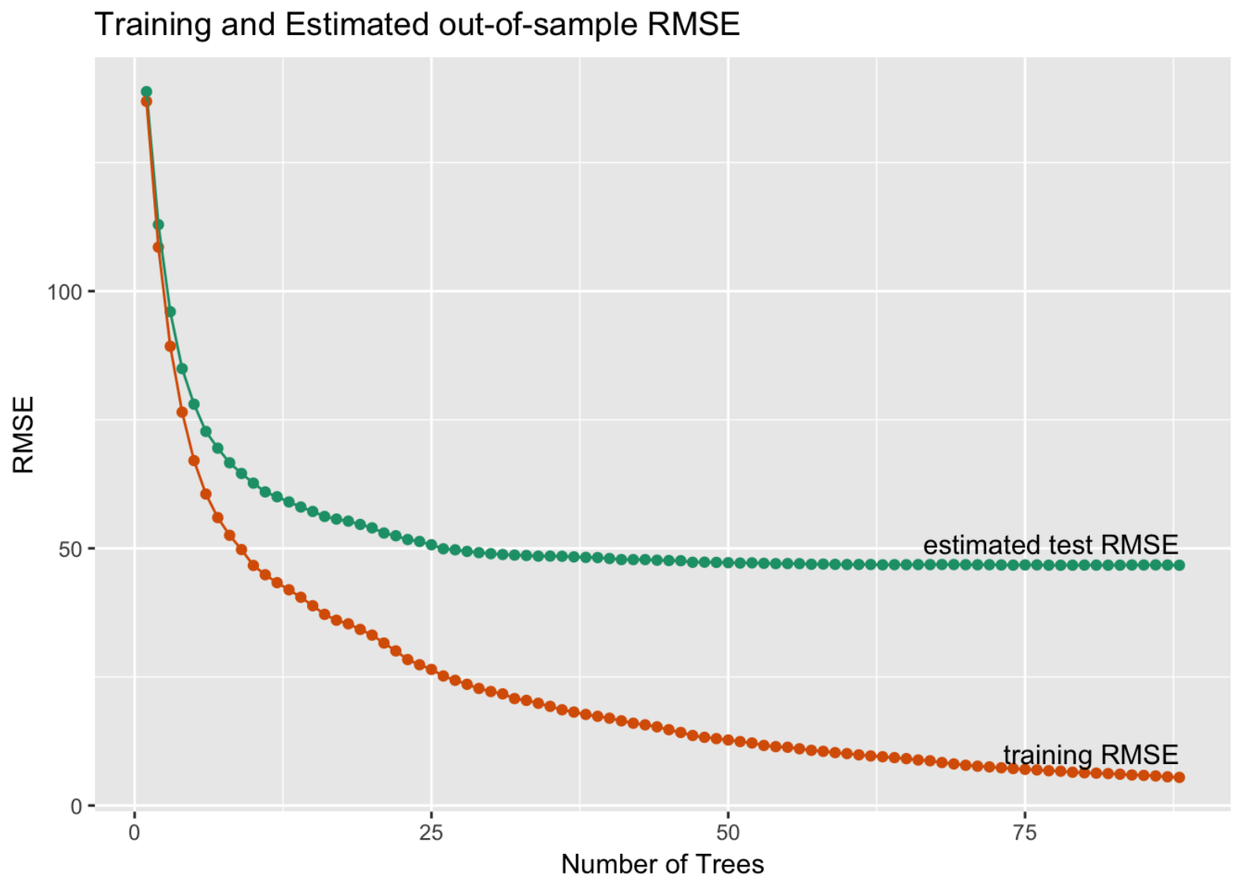


1. Fit a shallow tree T_1 to the data
   - $M_1 = T_1$
2. Fit a tree T_2 to the residuals.
   - $M_2 = M_1 + \eta\gamma_2 T_2$
3. Repeat (2) until stopping condition met

Final Model:

$$M = M_1 + \eta \sum \gamma_i T_i$$

Cross-validation to Guard Against Overfit



Training error keeps decreasing, but test error doesn't

Best Practice (with xgboost())

1. Run `xgb.cv()` with a large number of rounds (trees).

Best Practice (with xgboost())

1. Run `xgb.cv()` with a large number of rounds (trees).
2. `xgb.cv()$evaluation_log` : records estimated RMSE for each round.
   - Find the number of trees that minimizes estimated RMSE:

     n_{best}

Best Practice (with xgboost())

1. Run `xgb.cv()` with a large number of rounds (trees).
2. `xgb.cv()$evaluation_log` : records estimated RMSE for each round.
   - Find the number of trees that minimizes estimated RMSE: n_{best}
3. Run `xgboost()` , setting `nrounds` = n_{best}

Example: Bike Rental Model

First, prepare the data

```
treatplan <- designTreatmentsZ(bikesJan, vars)
newvars <- treatplan$scoreFrame %>%
     filter(code %in% c("clean", "lev")) %>%
     use_series(varName)

bikesJan.treat <- prepare(treatplan, bikesJan, varRestriction = newvars)
```

For `xgboost()`:

- Input data: `as.matrix(bikesJan.treat)`
- Outcome: `bikesJan$cnt`

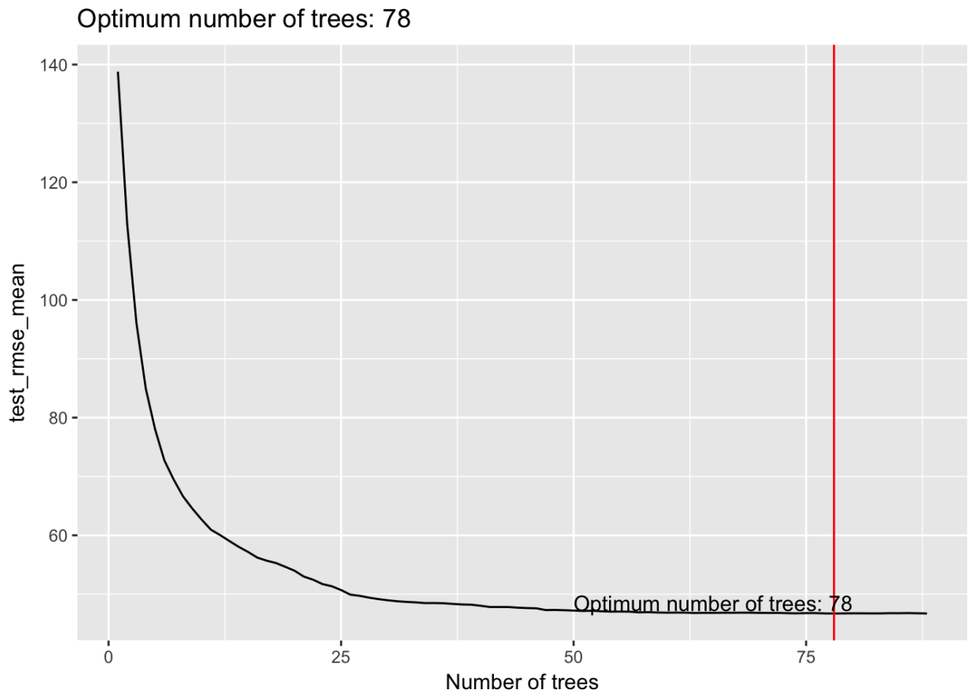
Training a model with xgboost() / xgb.cv()

```
cv <- xgb.cv(data = as.matrix(bikesJan.treat), label = bikesJan$cnt,
             objective = "reg:squarederror",
             nrounds = 100, nfold = 5, eta = 0.3, max_depth = 6)
```

Key inputs to `xgb.cv()` and `xgboost()`

- `data` : input data as matrix ; `label` : outcome
- `objective` : for regression - `"reg:squarederror"`
- `nrounds` : maximum number of trees to fit
- `eta` : learning rate
- `max_depth` : maximum depth of individual trees
- `nfold` (`xgb.cv()` only): number of folds for cross validation

Find the Right Number of Trees




```
elog <- as.data.frame(cv$evaluation_log)
(nrounds <- which.min(elog$test_rmse_mean))
```

```
78
```

Run xgboost() for final model

```
nrounds <- 78

model <- xgboost(data = as.matrix(bikesJan.treat),
                 label = bikesJan$cnt,
                 nrounds = nrounds,
                 objective = "reg:squarederror",
                 eta = 0.3,
                 max_depth = 6)
```

Predict with an xgboost() model

Prepare February data, and predict

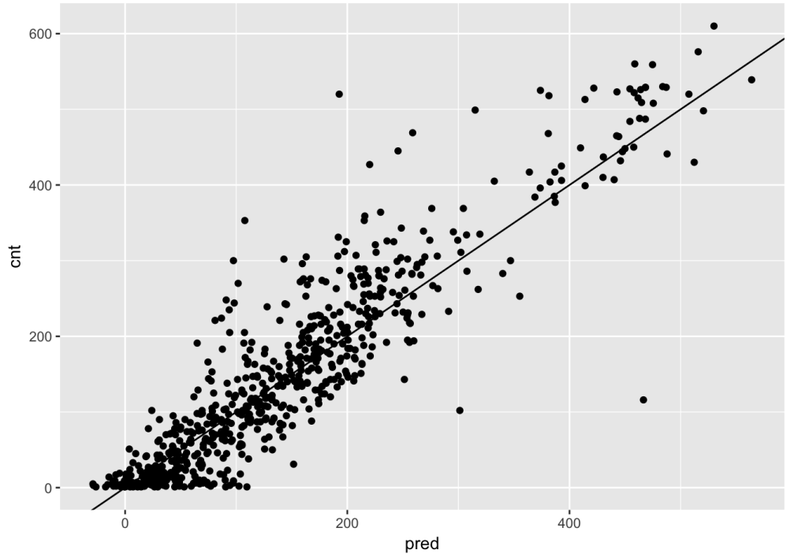
```
bikesFeb.treat <- prepare(treatplan, bikesFeb, varRestriction = newvars)

bikesFeb$pred <- predict(model, as.matrix(bikesFeb.treat))
```

Model performances on Febrary Data

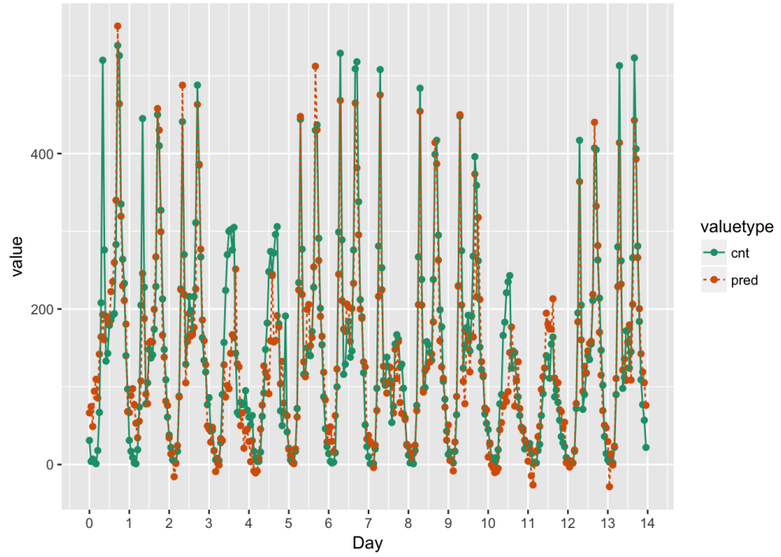
Model	RMSE
Quasipoisson	69.3
Random forests	67.15
Gradient Boosting	**54.0**

Visualize the Results

Predictions vs. Actual Bike Rentals, February



Predictions and Hourly Bike Rentals, February



Let's practice!

SUPERVISED LEARNING IN R: REGRESSION