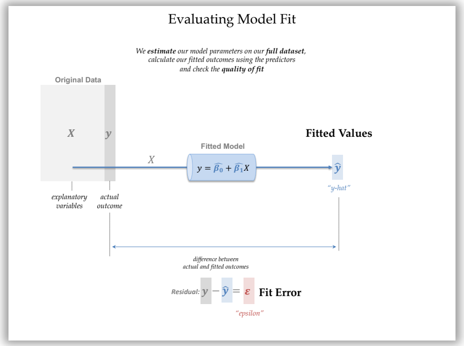
Business Analytics Using Computational Statistics

Structural Equation Modeling

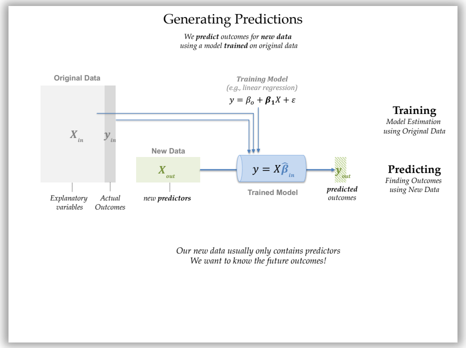
Predictions

Ensemble Methods

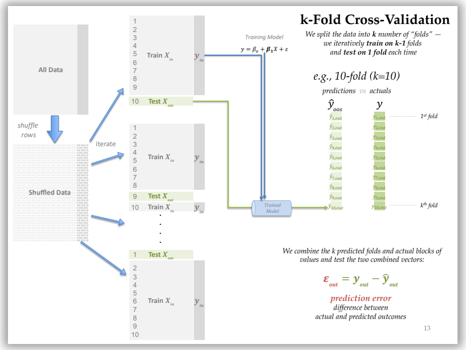
Model Fitting



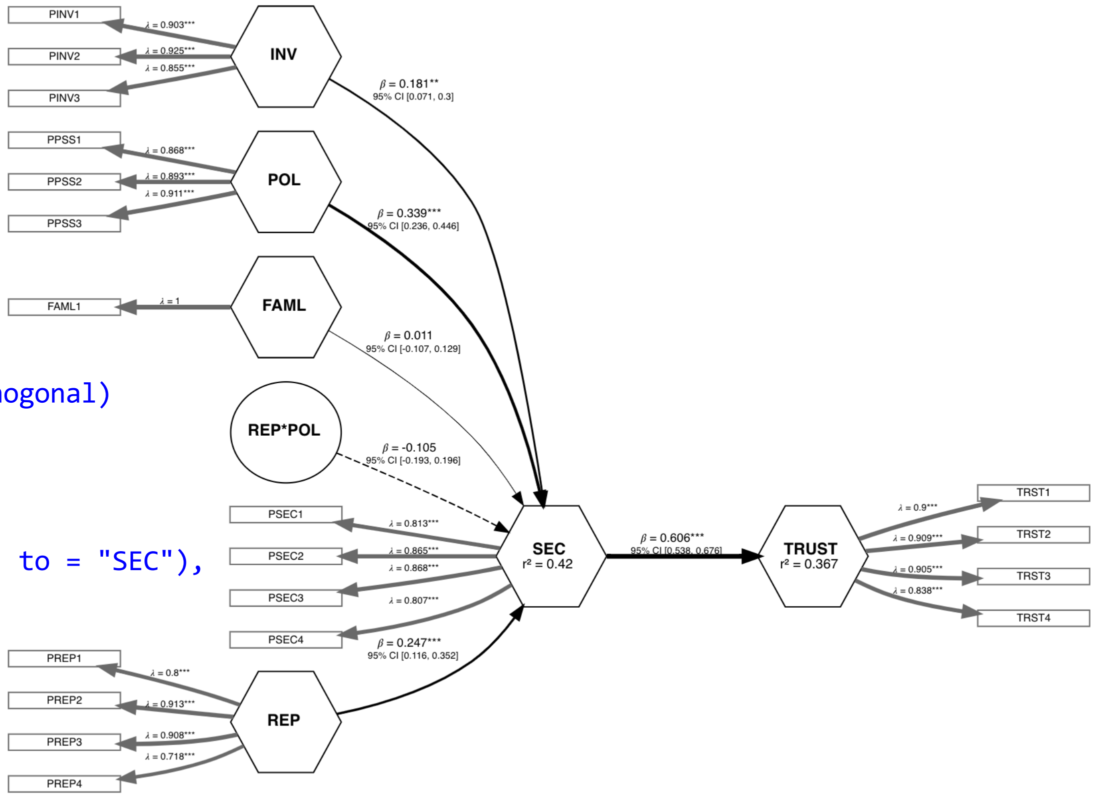
Generating Predictions



Testing Predictions



Applying Structural Equation Modeling

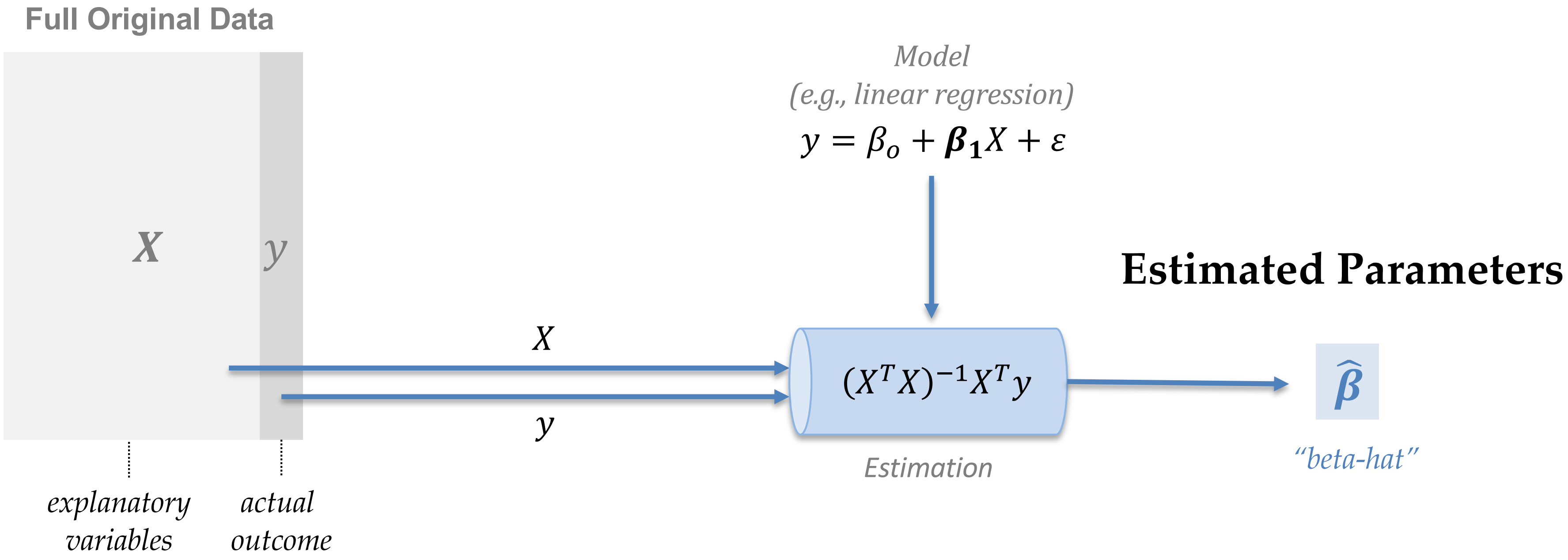



```r
library(seminr)

sec = read.csv("security_data.csv")

# Measurement Model
sec_mm <- constructs(
  composite("TRUST",  multi_items("TRST", 1:4)),
  composite("SEC",    multi_items("PSEC", 1:4)),
  composite("REP",    multi_items("PREP", 1:4)),
  composite("INV",    multi_items("PINV", 1:3)),
  composite("POL",    multi_items("PPSS", 1:3)),
  composite("FAML",   single_item("FAML1")),
  interaction_term(iv="REP", moderator="POL", method=orthogonal)
)

# Structural Model
sec_sm <- relationships(
  paths(from = c("REP", "INV", "POL", "FAML", "REP*POL"), to = "SEC"),
  paths(from = "SEC", to = "TRUST")
)

sec_pls <- estimate_pls(
  data = sec,
  measurement_model = sec_mm,
  structural_model = sec_sm
)

sec_pls_boot <- bootstrap_model(sec_pls, nboot = 1000)

summary(sec_pls_boot)
                 Original Est. Bootstrap Mean Bootstrap SD T Stat. 2.5% CI 97.5% CI
REP  ->  SEC             0.247          0.242        0.061   4.068   0.116    0.352
INV  ->  SEC             0.181          0.188        0.058   3.110   0.071    0.300
POL  ->  SEC             0.339          0.341        0.056   6.060   0.236    0.446
FAML  ->  SEC            0.011          0.012        0.059   0.177  -0.107    0.129
REP*POL  ->  SEC        -0.105         -0.021        0.126  -0.831  -0.193    0.196
SEC  ->  TRUST           0.606          0.610        0.035  17.260   0.538    0.676

plot(sec_pls_boot)
```

Fitting a Model

In-Sample ***Estimation***

*We **estimate** our model parameters by **fitting** them to our **full original dataset***



Estimation

Data

```
cars <- read.table("auto-data.txt", header=FALSE, na.strings = "?")
names(cars) <- c("mpg", "cylinders", "displacement", "horsepower",
                 "weight", "acceleration", "model_year",
                 "origin", "car_name")
cars$car_name <- NULL
cars <- na.omit(cars)
```

Model

$$y = \beta_o + \boldsymbol{\beta_1} x_1 + \varepsilon$$

```
X <- cbind("(intercept)" = rep(1, nrow(cars)),
           displacement  = cars$displacement)
y <- cars$mpg
```

Linear Algebra

$$mpg = \beta_o + \boldsymbol{\beta_1} displacement + \varepsilon$$

```
mpg ~ displacement
```

Using R function

Estimating Model Parameters

$$\widehat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T y = \begin{bmatrix} \widehat{\beta_0} \\ \widehat{\beta_1} \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

```
beta_hat <- solve(t(X)%*%X) %*% t(X)%*%y
```

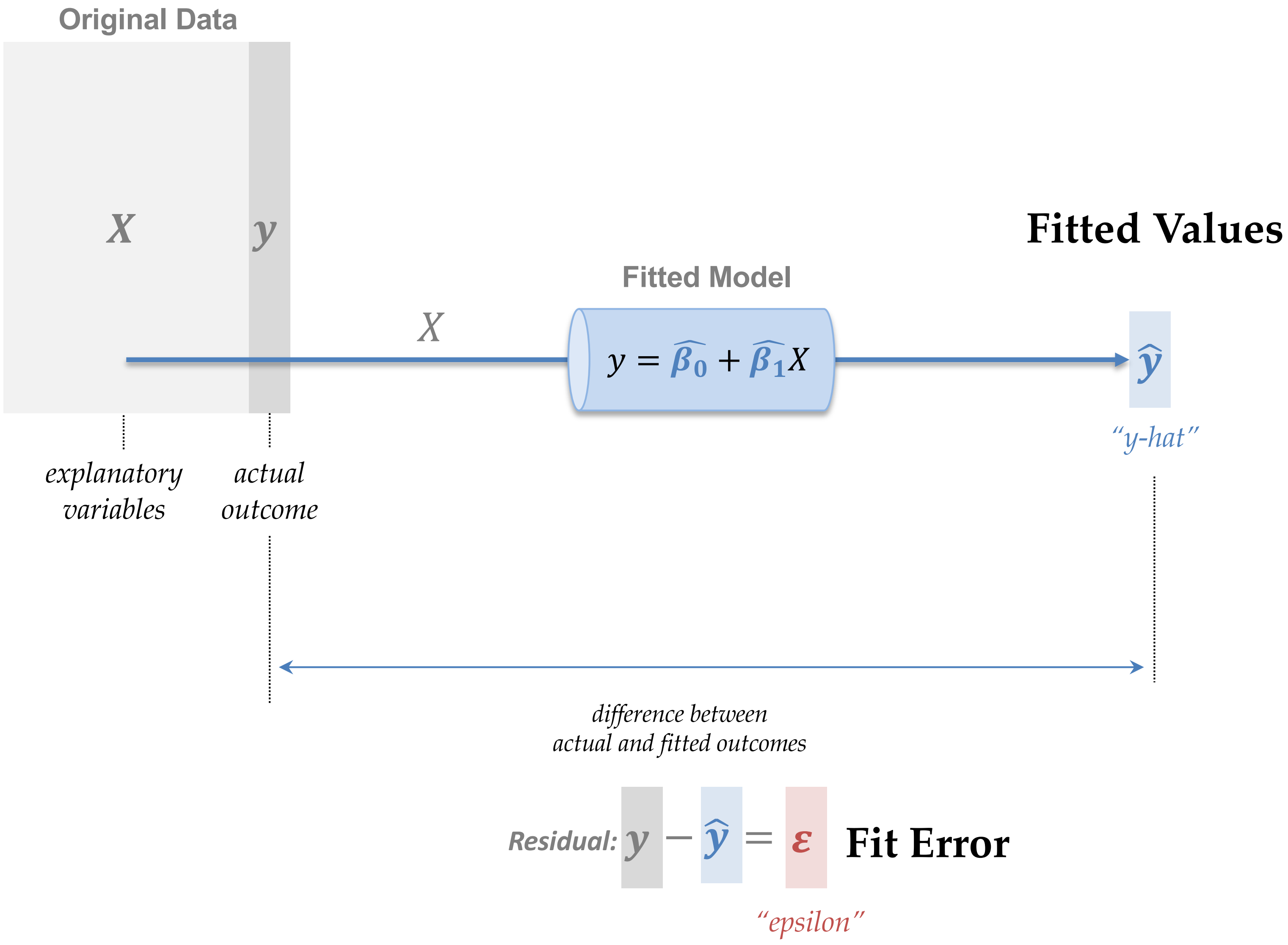
Linear Algebra

```
mpg_lm <- lm(mpg ~ displacement, data=cars)
mpg_lm$coefficients
```

Using R function

Evaluating Model Fit

*We **estimate** our model parameters on our **full dataset**,*
calculate our fitted outcomes using the predictors
*and check the **quality of fit***



Evaluating **In-Sample** Model Fit

We are using the same sample we fitted our data on, to test our fitted values $\hat{y}_{in}$

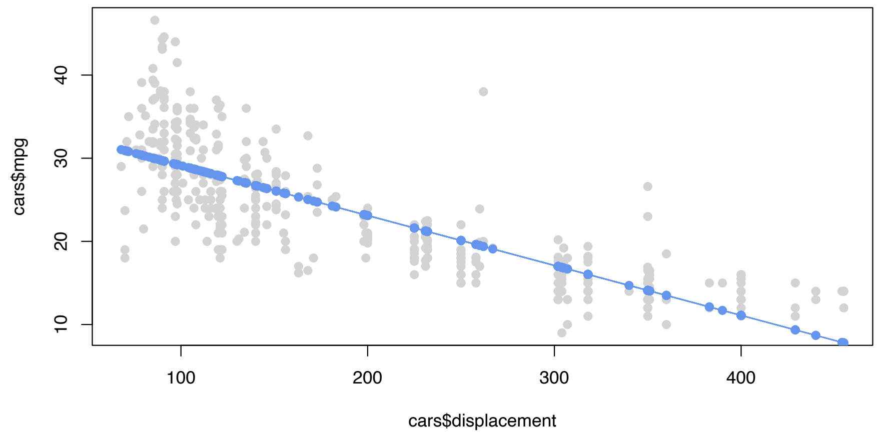
Fitted Values

$$\hat{y}_{in} = X\hat{\beta}_{in}$$

```
y_hat <- X %*% beta_hat
```

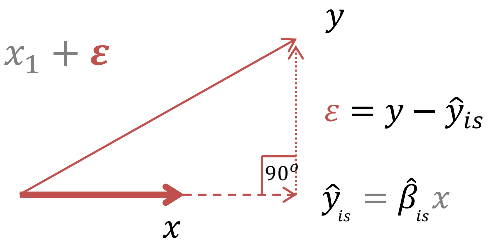
```
mpg_fitted <- fitted(mpg_lm)
        or
mpg_fitted <- mpg_lm$fitted.values
```

```
plot(cars$displacement, cars$mpg, pch=19, …)
points(cars$displacement, mpg_fitted, …
points(cars$displacement, mpg_fitted, type="l", …)
```



Fit Error (Residuals)

$$y = \beta_o + \beta_1 x_1 + \boldsymbol{\varepsilon}$$




```
fit_error <- y - y_hat
```

```
fit_error <- cars$mpg - mpg_fitted
        or
fit_error <- residuals(mpg_lm)
```

Mean Squared Fit Error

$$MSE_{in} = \frac{\overbrace{\sum(y - \hat{y}_{in})^2}^{SSE_{\text{in}}}}{n}$$

$$= \frac{\sum(\varepsilon)^2}{n}$$

Recall: $R^2 = 1 - \dfrac{SSE}{SST}$

```
mse_is <- mean((y - y_hat)^2)
```

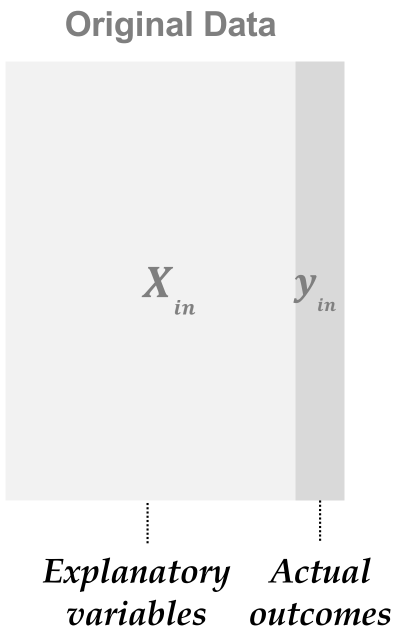
```
mse_is <- mean((cars$mpg - fitted(mpg_lm))^2)
        or
mse_is <- mean(residuals(mpg_lm)^2)
```

Out-of-sample Predictions

We can reapply our fitted model to predict outcomes in a new sample

Original Sample

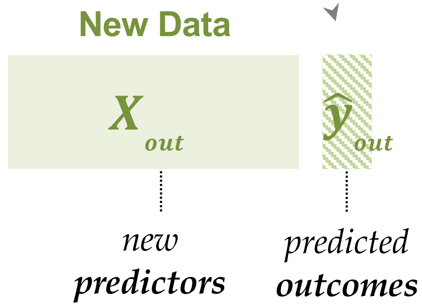
Data from Original Context and Time



$$y_{in} = X_{in}\hat{\beta}_{in}$$

New Sample

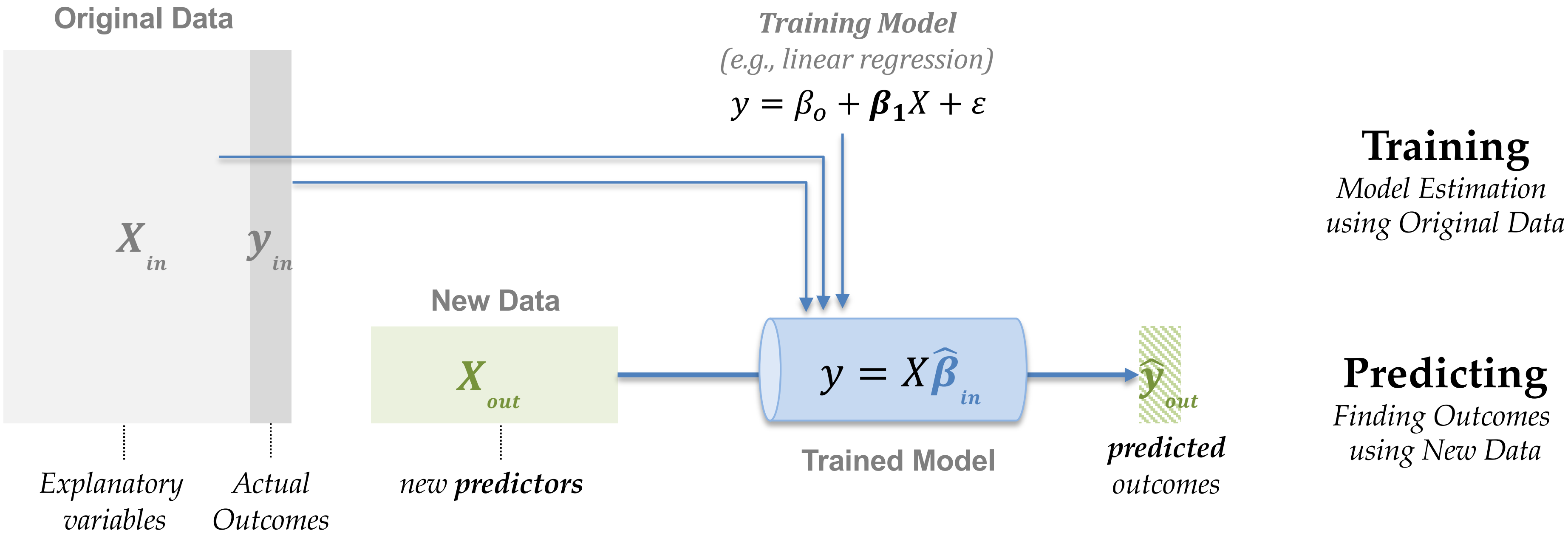
Data from New Context or Time
or, New Data from Original Context and Time



Usually, we want to make predictions
*when we do **not** have actual outcomes for our new data*

Generating Predictions

*We **predict** outcomes for **new data***
*using a model **trained** on original data*



Our new data usually only contains predictors
We want to know the future outcomes!

Generating Predictions

```r
old_cars <- subset(cars, model_year <= 81)
new_cars <- subset(cars, model_year == 82)
```

Let's pretend we only had ***older cars*** *(1981 and before) to train with*
We now want to predict the mpg of ***next year's*** *(1982) cars*

Training *(model estimation on orignal data)*

```r
X_old <- cbind("(intercept)"=rep(1, nrow(old_cars)), displacement=old_cars$displacement)
y_old <- old_cars$mpg
beta_hat_old <- solve(t(X_old)%*%X_old) %*% t(X_old)%*%y_old
```

Linear Algebra

```r
lm_old <- lm(mpg ~ displacement, data=old_cars)
```

Using R function

Predicting *(outcomes based on trained model and new predictors)*

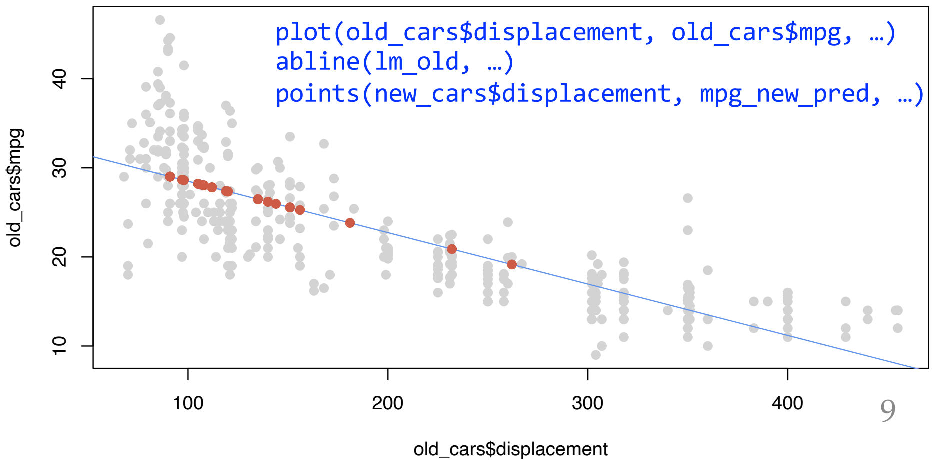
```r
X_new <- cbind("(intercept)"=rep(1, nrow(new_cars)), displacement=new_cars$displacement)
y_hat_new <- X_new %*% beta_hat_old
```

Linear Algebra

```r
mpg_new_pred <- predict(lm_old, new_cars)
```

Using R function

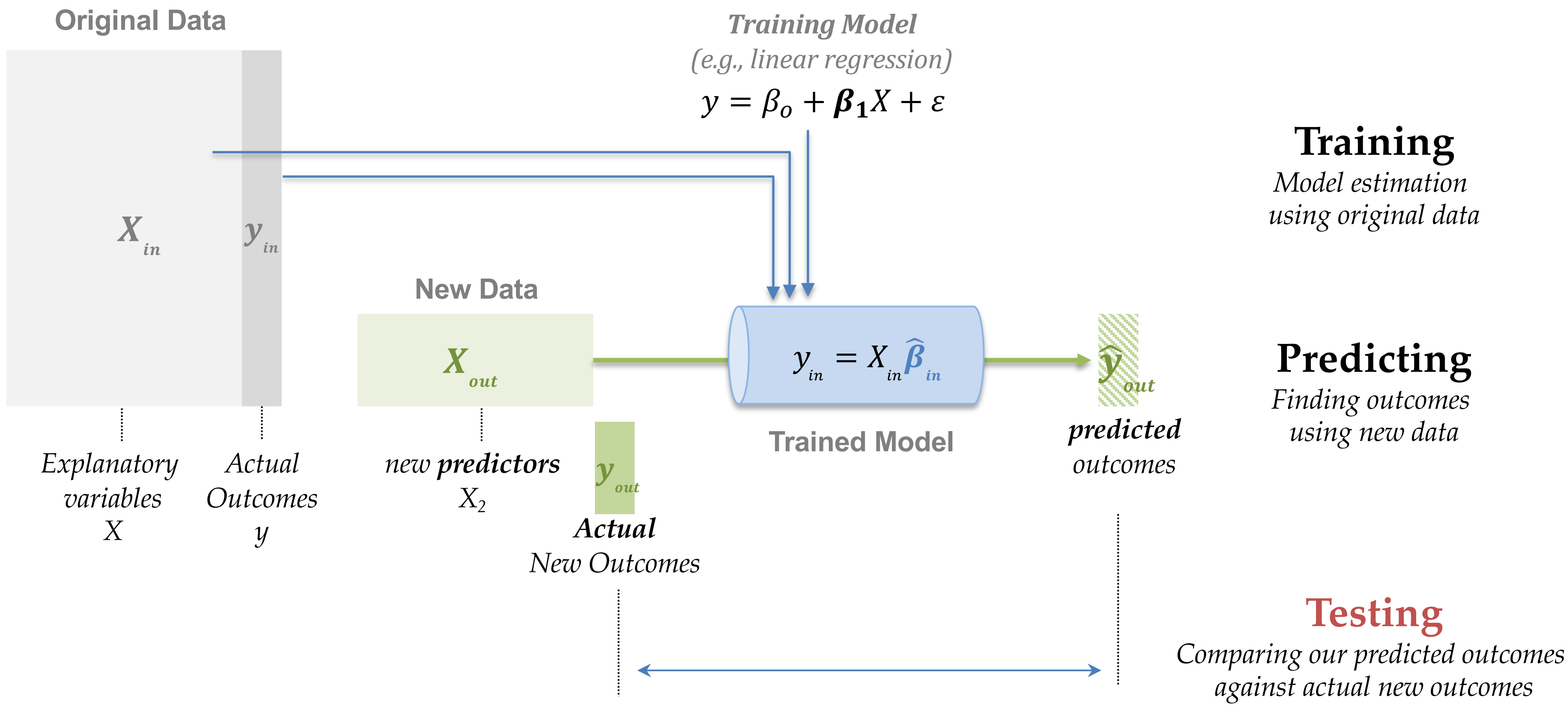
How can we ***test*** *how good our predictions are?*



Testing Predictions

*We **train** models on training data, to **predict** outcomes of new predictors, and **test** against actual new outcomes*

To test how good our predictions are, we would ideally compare our predictions against new outcomes values that happen in future



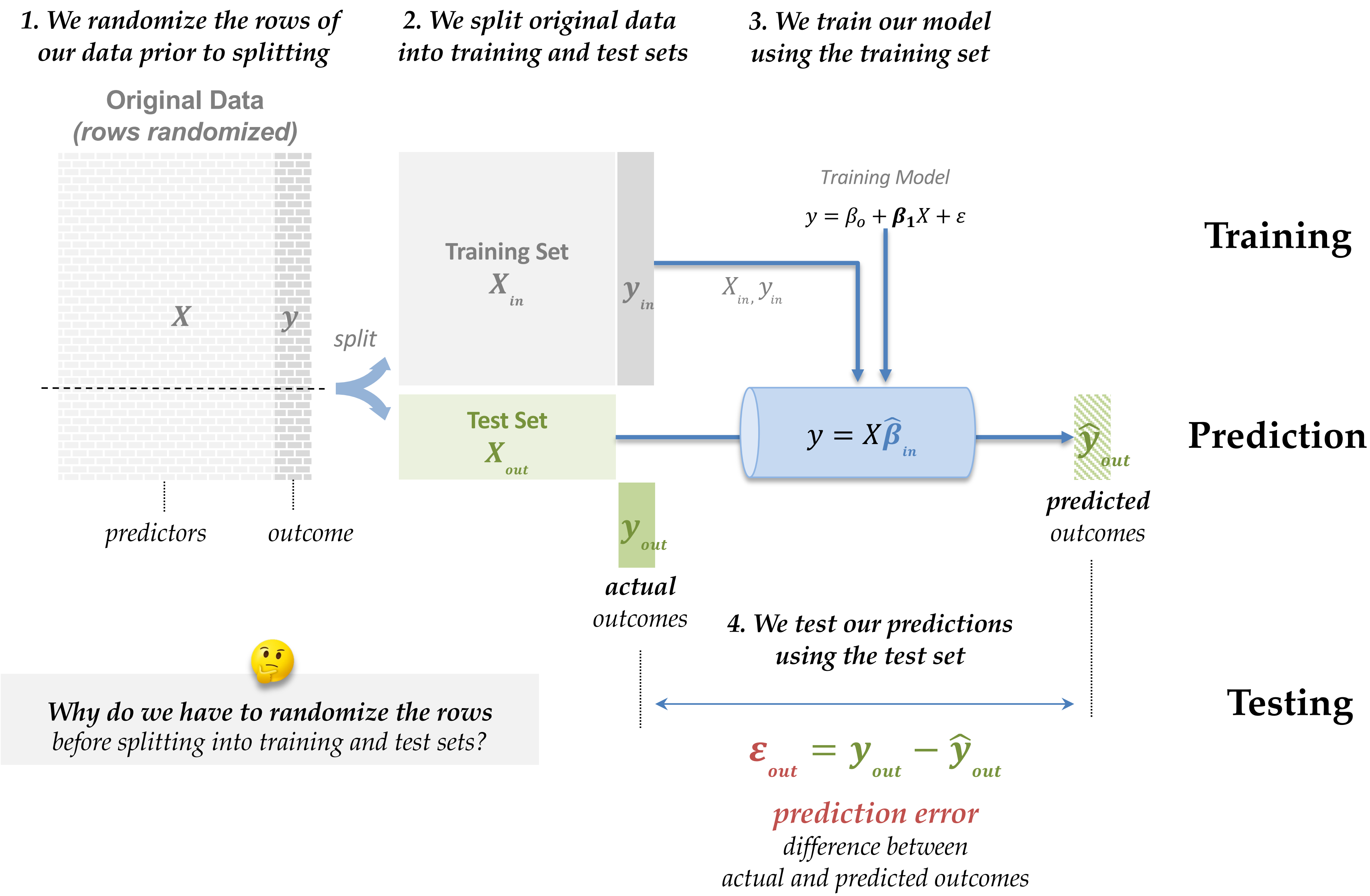
$$\varepsilon_{out} = y_{out} - \hat{y}_{out}$$

prediction error
difference between actual and predicted outcomes

But we usually do not have actual new outcomes!
How can we test how good our predictions might be in general?

Split-Sample Testing

*To **test** against "actual outcomes", we split our original data into **training** and **test** sets*



Split-Sample Testing

*e.g., **random** 70:30 split*

Training

We train on 70% of our data, chosen at random

```
set.seed(27935752)
train_indices <- sample(1:nrow(cars), size=0.70*nrow(cars))
train_set <- cars[train_indices,]
lm_trained <- lm(mpg ~ displacement, data=train_set)
```

Training Set
(70%)

We want our training set to be much larger than our test set

Predicting

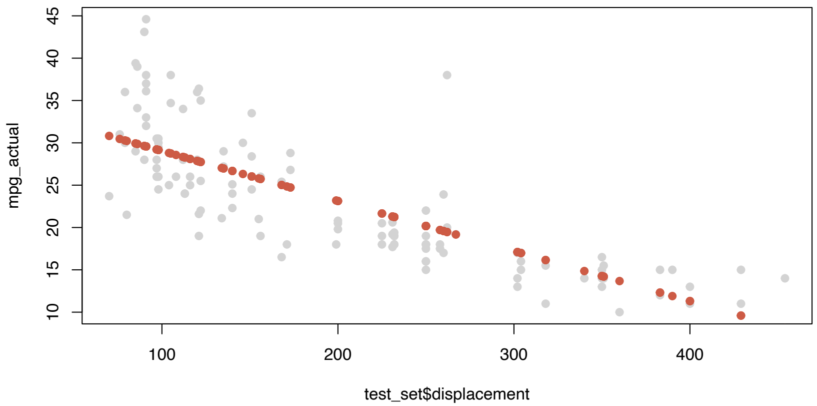
We predict and test on the remaining 30% of data

```
test_set <- cars[-train_indices,]
mpg_predicted <- predict(lm_trained, test_set)
```

Test Set
(30%)

```
plot(test_set$displacement, mpg_actual, …)
points(test_set$displacement, mpg_predicted, …)
```

*Will our **Prediction Error** be the same, better, or worse than **Fit Error**?*



Mean-Square Predictive Error

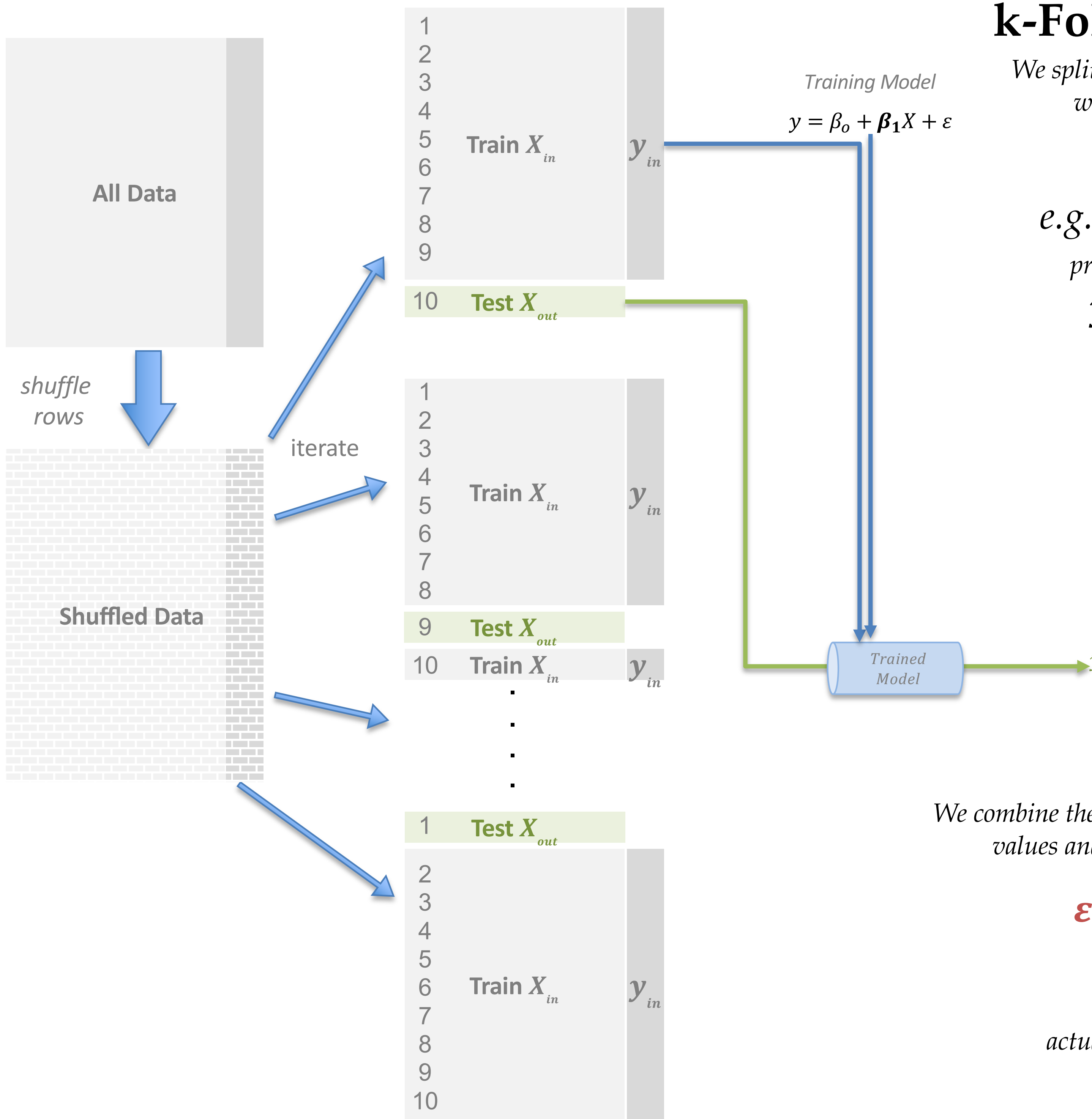
$$MSE_{out} = \frac{\sum \overbrace{(y_{out} - \hat{y}_{out})}^{\text{prediction error } \varepsilon_{out}}{}^2}{n}$$

```
mpg_actual <- test_set$mpg
pred_err <- mpg_actual - mpg_predicted
mse_out  <- mean( (mpg_predicted - mpg_actual)^2 )
```

What if our chosen training/test sets are unusual?

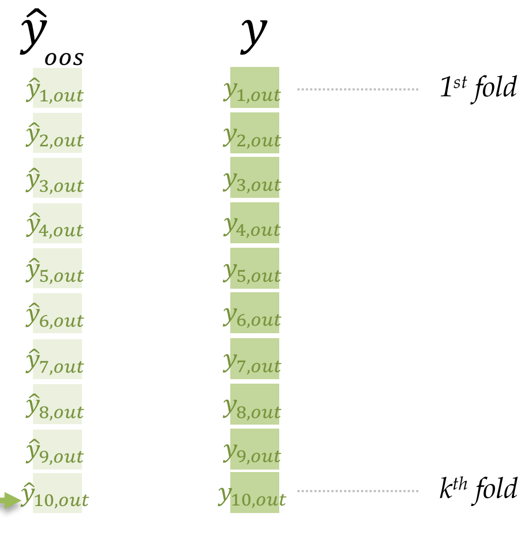
k-Fold Cross-Validation

*We split the data into **k** number of "folds" — we iteratively **train on k-1** folds and **test on 1 fold** each time*



e.g., 10-fold (k=10)

predictions vs actuals



We combine the k predicted folds and actual blocks of values and test the two combined vectors:

$$\varepsilon_{out} = y_{out} - \hat{y}_{out}$$

prediction error

difference between actual and predicted outcomes

k-Fold Cross-Validation

Implementation Hints!

… = *yours to fill in!*

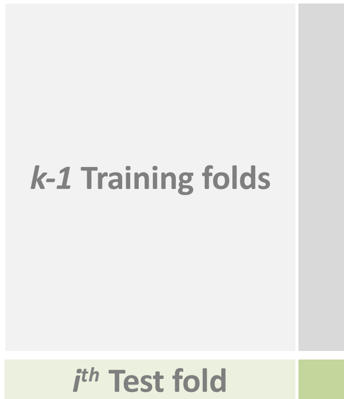
```r
# Calculate mse_out across all folds
k_fold_mse <- function(dataset, k=10, …) {
  fold_pred_errors <- sapply(1:k, \(i) {
    fold_i_pe(i, k, dataset, …)
  })
  pred_errors <- unlist(fold_pred_errors)

  mean(pred_errors^2)
}
```

```r
# Calculate prediction error for fold i out of k
fold_i_pe <- function(i, k, dataset, …) {
  folds <- cut(…, labels = FALSE)

  test_indices <- which(…)
  test_set  <- dataset[…]
  train_set <- dataset[…]
  trained_model <- …

  predictions <- predict(…)
  actuals - predictions
}
```

$$MSE_{out} = \frac{\sum(y_{out} - \hat{y}_{out})^2}{n}$$

Reference to new functions:

```r
quarters <- cut(1:100, 4, labels = FALSE)
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

2nd_quarter <- which(quarters == 2)
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```




```r
mpg_lm <- lm(mpg ~ displacement, data=cars)
mean((cars$mpg - mpg_lm$fitted.values)^2)
21.37454
```

```r
lm_mse <- k_fold_mse(mpg_lm, cars, cars$mpg)
23.94120
```

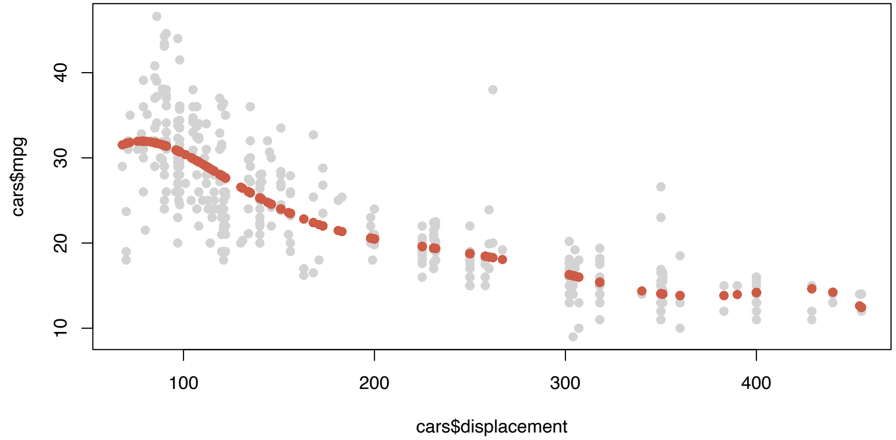
More Complex Models

Models can seem to match data better if we increase their complexity

Complex Global Models

We can use bigger model formulas with more parameters to estimate to create a more complex response surface

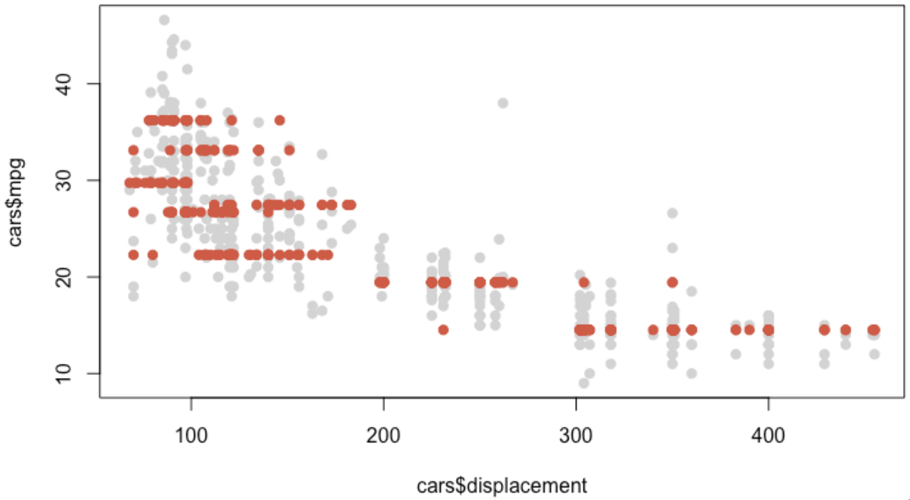
e.g., Polynomial Regression



Complex Partitioned Models

We can divide the data space into smaller spaces and make simpler predictions in each space

e.g., Regression Trees



Polynomial Regression

*We can put **higher-order terms** into regression to explicitly take into account non-linearities*

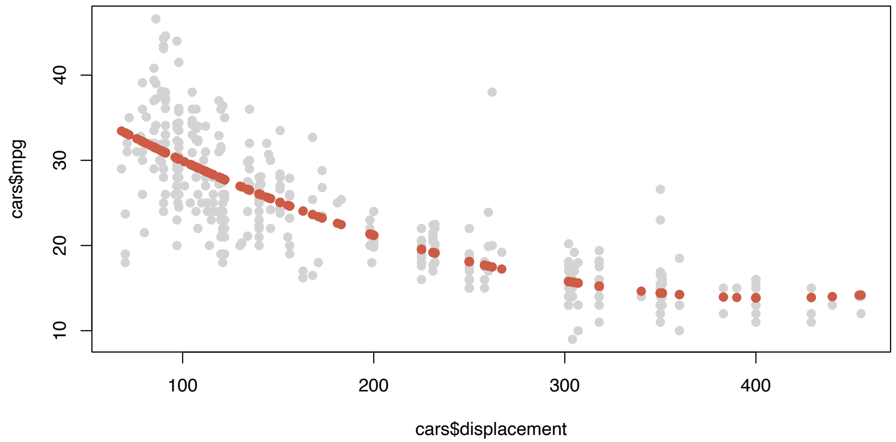
$$y = \beta_o + \beta_1 \boldsymbol{x} + \beta_2 \boldsymbol{x^2} + \cdots + \beta_p \boldsymbol{x^p} + \varepsilon$$

Is using many polynomial terms useful for explanation or prediction?

Quadratic Model:

$$y = \beta_o + \beta_1 \boldsymbol{x} + \beta_2 \boldsymbol{x^2} + \varepsilon$$

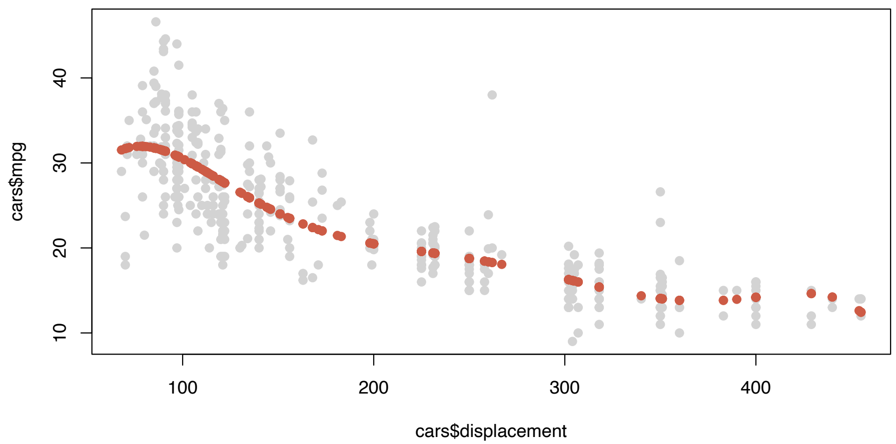
```
pm_regr2 <- lm(mpg ~ poly(displacement, 2), data=cars)
plot(cars$displacement, cars$mpg, …)
points(cars$displacement, pm_regr2$fitted.values, …)

k_fold_mse(pm_regr2, cars, cars$mpg)
```



6th-degree Polynomial Model:

$$y = \beta_o + \beta_1 \boldsymbol{x} + \beta_2 \boldsymbol{x^2} + \beta_2 \boldsymbol{x^3} + \beta_2 \boldsymbol{x^4} + \beta_2 \boldsymbol{x^5} + \beta_2 \boldsymbol{x^6} + \varepsilon$$

```
pm_regr6 <- lm(mpg ~ poly(displacement, 6), data=cars)
plot(cars$displacement, cars$mpg, …)
points(cars$displacement, pm_regr6$fitted.values, …)

k_fold_mse(pm_regr6, cars, cars$mpg)
```



*Which model might **fit** the original data better?*

*Which model might **predict** new data better?*

Why would there be a difference between fit and prediction?!?

Regression Trees

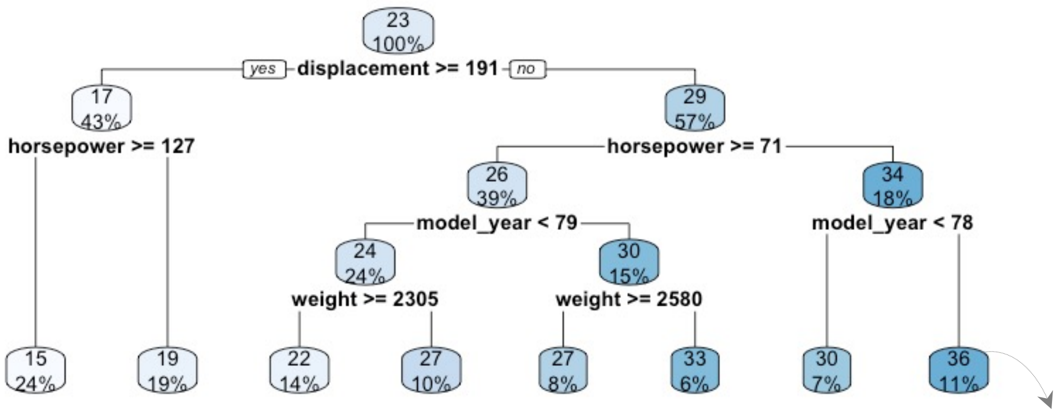
*Model that uses **Decision Tree** structure to relate explanatory variables to an outcome*

outcome variable

$$mpg \sim \overbrace{cyl + disp + hp + weight + acc + year + origin}^{\textit{predictor variables}}$$

$$y \sim x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + \varepsilon$$

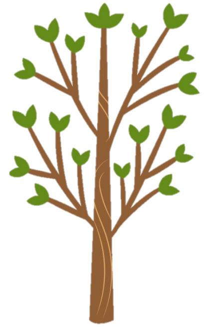
Root of Tree



Leaves of Tree

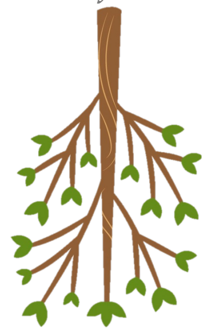
all cases at a leaf
*get same **predicted value***

Leaves of tree



Root of tree

Root of tree



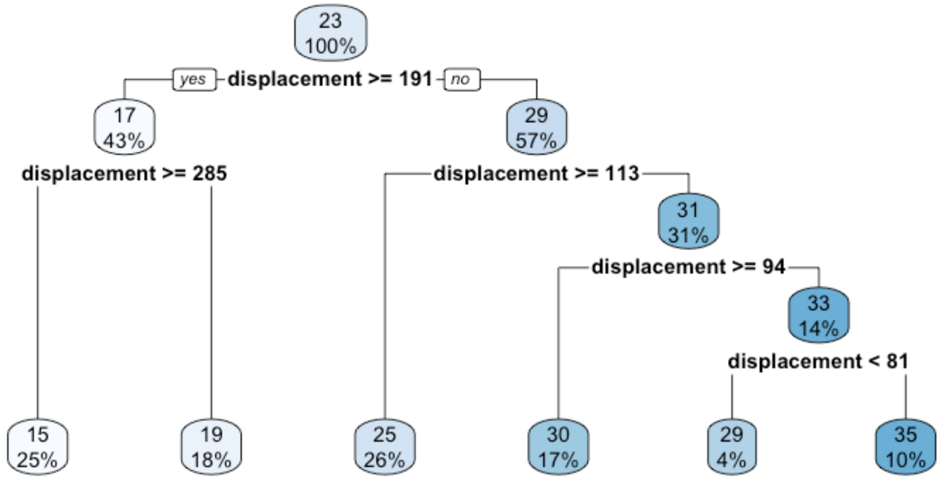
Leaves of tree

Regression Trees

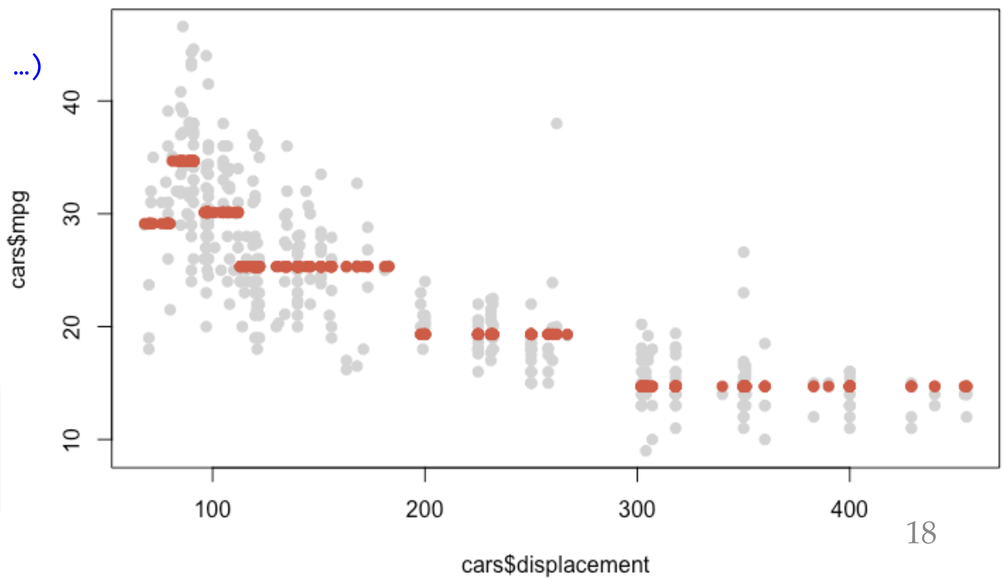
Predictive model that uses Decision Tree structure to relate outcomes of observations to its predictors

Single predictor example: `mpg ~ displacement`

```r
library(rpart)
library(rpart.plot)

# Defining the model
cars_tree <- rpart(mpg ~ displacement, data=cars)


# Plotting the tree
rpart.plot(cars_tree)
```




```r
plot(cars$displacement, cars$mpg, pch=19, col="lightgray")
points(cars$displacement, predict(cars_tree, newdata = cars), …)

k_fold_mse(cars_tree, cars, cars$mpg)
```



Plotting the fitted response is not feasible for more than one predictor; we usually look at the figure of the tree rather than a scatterplot