Applied Machine Learning

Max Kuhn (RStudio)

Why R for Modeling?

- *R has cutting edge models.*

  Machine learning developers in some domains use R as their primary computing environment and their work often results in R packages.

- *It is easy to port or link to other applications.*

  R doesn't try to be everything to everyone. If you prefer models implemented in C, C++, `tensorflow`, `keras`, `python`, `stan`, or `Weka`, you can access these applications without leaving R.

- *R and R packages are built by people who* ***do*** *data analysis.*

- *The S language is very mature.*

- The machine learning environment in R is extremely rich.

Downsides to Modeling in R

- R is a data analysis language and is not C or Java. If a high performance deployment is required, R can be treated like a prototyping language.
- R is mostly memory-bound. There are plenty of exceptions to this though.

The main issue is one of *consistency of interface*. For example:

- There are two methods for specifying what terms are in a model[1]. Not all models have both.
- 99% of model functions automatically generate dummy variables.
- Sparse matrices can be used (unless they can't).

[1] There are now three but the last one is brand new and will be discussed later.

Syntax for Computing Predicted Class Probabilities

Function	Package	Code
`lda`	`MASS`	`predict(obj)`
`glm`	`stats`	`predict(obj, type = "response")`
`gbm`	`gbm`	`predict(obj, type = "response", n.trees)`
`mda`	`mda`	`predict(obj, type = "posterior")`
`rpart`	`rpart`	`predict(obj, type = "prob")`
`Weka`	`RWeka`	`predict(obj, type = "probability")`
`logitboost`	`LogitBoost`	`predict(obj, type = "raw", nIter)`
`pamr.train`	`pamr`	`pamr.predict(obj, type = "posterior", threshold)`

We'll see a solution for this later in the class.

tidymodels Collection of Packages

```r
library(tidymodels)
```

```
## Registered S3 method overwritten by 'xts':
##   method     from
##   as.zoo.xts zoo
```

```
## ── Attaching packages ─────────────────────────────── tidymodels 0.0.2 ──
```

```
## ✔ broom     0.5.1          ✔ purrr     0.3.3
## ✔ dials     0.0.3.9002     ✔ recipes   0.1.7.9001
## ✔ dplyr     0.8.3          ✔ rsample   0.0.5
## ✔ ggplot2   3.2.1          ✔ tibble    2.1.3
## ✔ infer     0.4.0          ✔ yardstick 0.0.4
## ✔ parsnip   0.0.4
```

```
## ── Conflicts ────────────────────────────────── tidymodels_conflicts() ──
## ✖ purrr::discard()  masks scales::discard()
## ✖ dplyr::filter()   masks stats::filter()
## ✖ dplyr::lag()      masks stats::lag()
## ✖ ggplot2::margin() masks dials::margin()
## ✖ dials::offset()   masks stats::offset()
## ✖ recipes::step()   masks stats::step()
```

Plus `tidypredict`, `tidyposterior`, `tidytext`, and more in development.

Example Data Set - House Prices

For our examples, we will use the Ames IA housing data. There are 2,930 properties in the data.

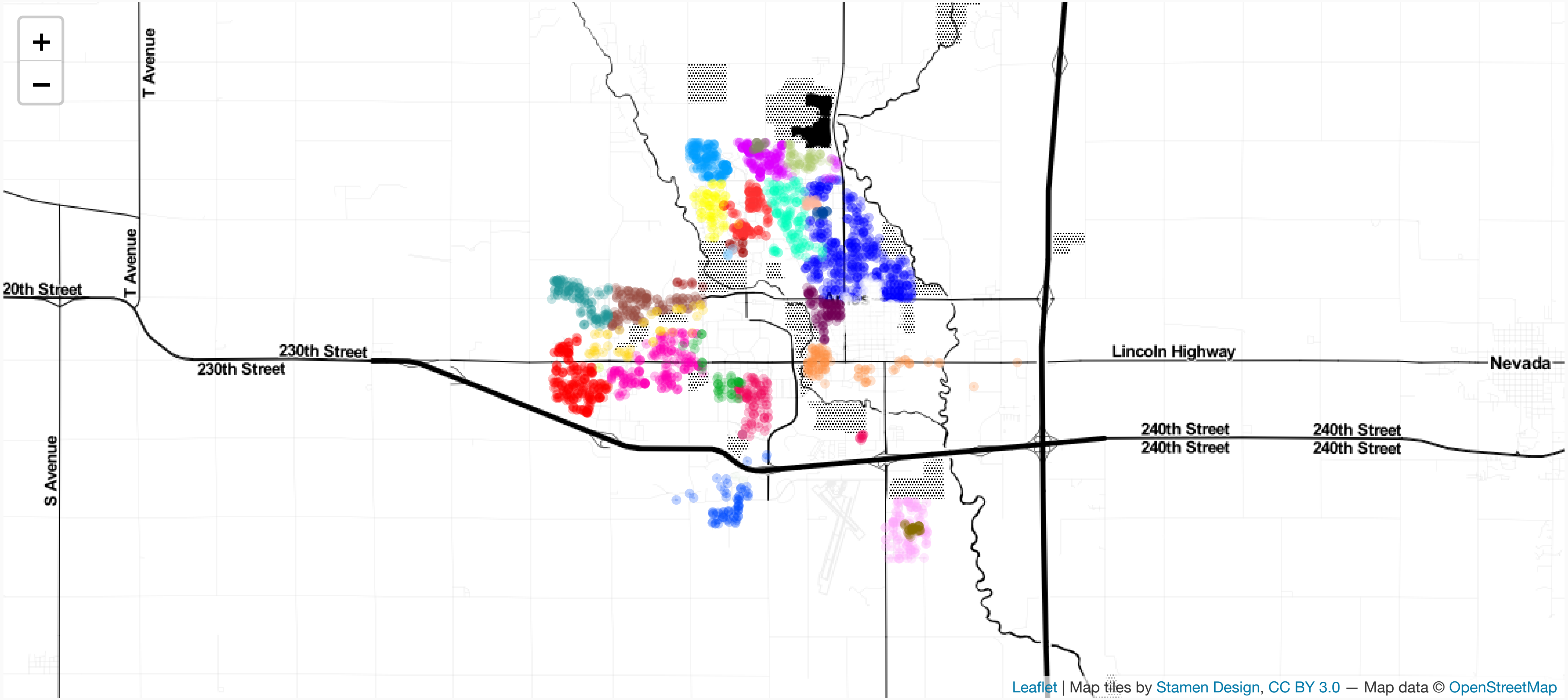
The Sale Price was recorded along with 81 predictors, including:

- Location (e.g. neighborhood) and lot information.
- House components (garage, fireplace, pool, porch, etc.).
- General assessments such as overall quality and condition.
- Number of bedrooms, baths, and so on.

More details can be found in De Cock (2011, Journal of Statistics Education).

The raw data are at `http://bit.ly/2whgsQM` but we will use a processed version found in the `AmesHousing` package.

Example Data Set - House Prices



Tidyverse Syntax

Many tidyverse functions have syntax unlike base R code. For example:

- Vectors of variable names are eschewed in favor of *functional programming*. For example:

```
contains("Sepal")

# instead of

c("Sepal.Width", "Sepal.Length")
```

- The *pipe* operator is preferred. For example:

```
merged <- inner_join(a, b)

# is equal to

merged <- a %>%
  inner_join(b)
```

- Functions are more *modular* than their traditional analogs (`dplyr`'s `filter()` and `select()` vs `base::subset()`)

Some Example Data Manipulation Code

```r
library(tidyverse)

ames_prices <- "http://bit.ly/2whgsQM" %>%
  read_delim(delim = "\t", guess_max = 2000) %>%
  rename_at(vars(contains(' ')), list(~gsub(' ', '_', .))) %>%
  dplyr::rename(Sale_Price = SalePrice) %>%
  dplyr::filter(!is.na(Electrical)) %>%
  dplyr::select(-Order, -PID, -Garage_Yr_Blt)

ames_prices %>%
  group_by(Alley) %>%
  summarize(
    mean_price = mean(Sale_Price / 1000),
    n = sum(!is.na(Sale_Price))
  )
```

```
## # A tibble: 3 x 3
##   Alley mean_price     n
##   <chr>      <dbl> <int>
## 1 Grvl        124.   120
## 2 Pave        177.    78
## 3 <NA>        183.  2731
```

Examples of `purrr::map*`

purrr contains functions that *iterate over lists* without the explicit use of loops. They are similar to the family of apply functions in base R, but are type stable.

```r
# purrr loaded with tidyverse or tidymodels package

mini_ames <- ames_prices %>%
  dplyr::select(Alley, Sale_Price, Yr_Sold) %>%
  dplyr::filter(!is.na(Alley))

head(mini_ames, n = 5)
```

```
## # A tibble: 5 x 3
##   Alley Sale_Price Yr_Sold
##   <chr>      <dbl>   <dbl>
## 1 Pave      190000    2010
## 2 Pave      155000    2010
## 3 Pave      151000    2010
## 4 Pave      149500    2010
## 5 Pave      152000    2010
```

```r
by_alley <- split(mini_ames, mini_ames$Alley)
# map(.x, .f, ...)
map(by_alley, head, n = 2)
```

```
## $Grvl
## # A tibble: 2 x 3
##   Alley Sale_Price Yr_Sold
##   <chr>      <dbl>   <dbl>
## 1 Grvl       96500    2010
## 2 Grvl      109500    2010
##
## $Pave
## # A tibble: 2 x 3
##   Alley Sale_Price Yr_Sold
##   <chr>      <dbl>   <dbl>
## 1 Pave      190000    2010
## 2 Pave      155000    2010
```

Examples of `purrr::map*`

```
map(by_alley, nrow)
```

```
## $Grvl
## [1] 120
##
## $Pave
## [1] 78
```

`map()` always returns a list. Use suffixed versions for simplification of the result.

```
map_int(by_alley, nrow)
```

```
## Grvl Pave
##  120   78
```

Complex operations can be specified using a *formula notation*. Access the current thing you are iterating over with `.x`.

```
map(
  by_alley,
  ~summarise(.x, max_price = max(Sale_Price))
)
```

```
## $Grvl
## # A tibble: 1 x 1
##   max_price
##       <dbl>
## 1    256000
##
## $Pave
## # A tibble: 1 x 1
##   max_price
##       <dbl>
## 1    345000
```

{purrr} and list-columns

Rather than using `split()`, we can `tidyr::nest()` by `Alley` to get a data frame with a *list-column*. We often use these when working with *multiple models*.

```r
ames_lst_col <-
  nest(mini_ames, data = c(Sale_Price, Yr_Sold))

# or
# nest(mini_ames, data = c(-Alley))

ames_lst_col
```

```
## # A tibble: 2 x 2
##   Alley             data
##   <chr> <list<df[,2]>>
## 1 Pave        [78 × 2]
## 2 Grvl       [120 × 2]
```

```r
ames_lst_col %>%
  mutate(
    n_row = map_int(data, nrow),
    max   = map_dbl(data, ~ max(.x$Sale_Price))
  )
```

```
## # A tibble: 2 x 4
##   Alley           data n_row    max
##   <chr> <list<df[,2]>> <int>  <dbl>
## 1 Pave        [78 × 2]    78 345000
## 2 Grvl       [120 × 2]   120 256000
```

Quick Data Investigation

To get warmed up, let's load the real Ames data and do some basic investigations into the variables, such as exploratory visualizations or summary statistics. The idea is to get a feel for the data.

Let's take 10 minutes to work on your own or with someone next to you. Collaboration is highly encouraged!

To get the data:

```r
library(AmesHousing)
ames <- make_ames()
```

10:00

Resources

- `http://www.tidyverse.org/`
- R for Data Science
- Jenny's `purrr` tutorial or Happy R Users Purrr
- Programming with `dplyr` vignette
- Selva Prabhakaran's `ggplot2` tutorial
- `caret` package documentation
- CRAN Machine Learning Task View

About these slides.... they were created with Yihui's `xaringan` and the stylings are a slightly modified version of Patrick Schratz's Metropolis theme.

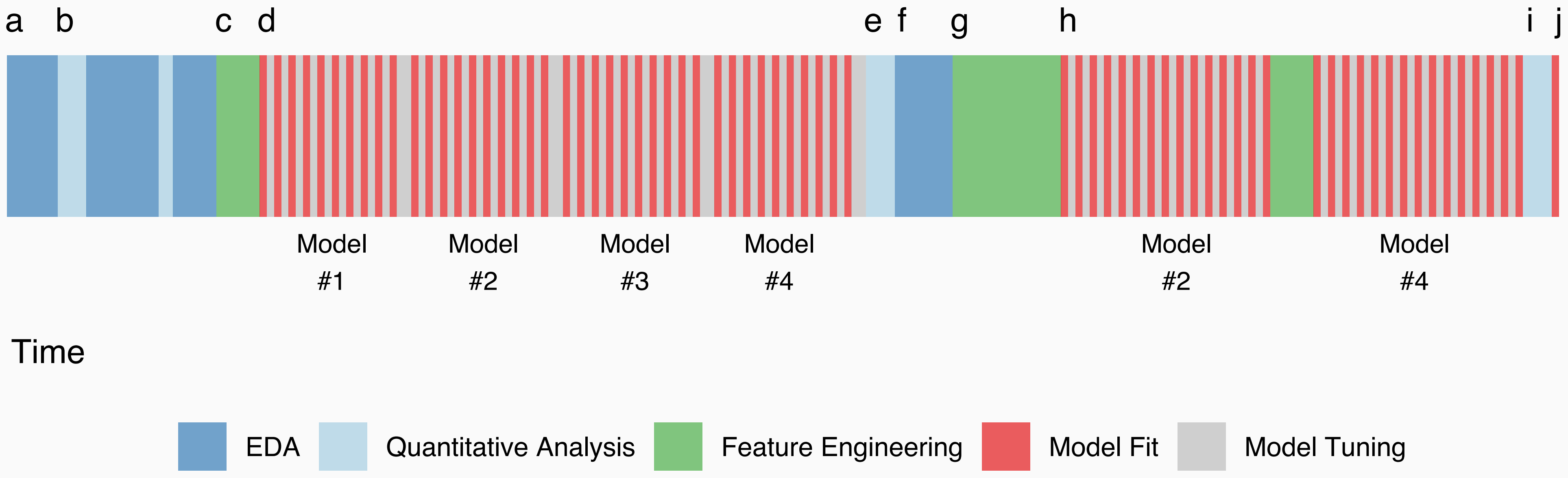
The Modeling *Process*

Common steps during model building are:

- estimating model parameters (i.e. training models)
- determining the values of *tuning parameters* that cannot be directly calculated from the data
- model selection (within a model type) and model comparison (between types)
- calculating the performance of the final model that will generalize to new data

Many books and courses portray predictive modeling as a short sprint. A better analogy would be a marathon or campaign (depending on how hard the problem is).

What the Modeling Process Usually Looks Like

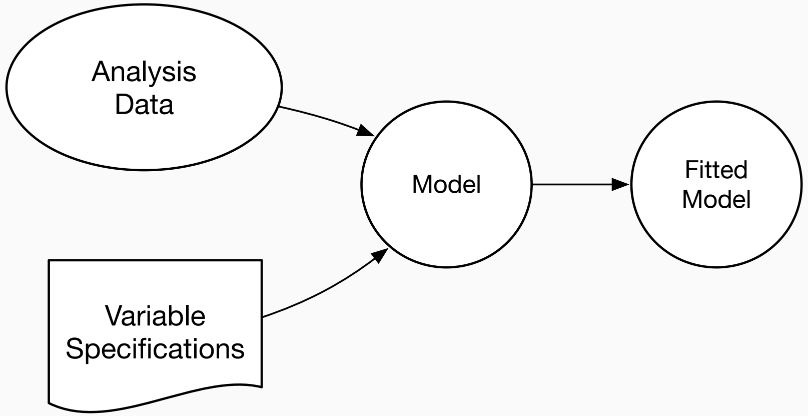


What Are We Doing with the Data?

We often think of the model as the *only* real data analysis step in this process.

However, there are other procedures that are often applied before or after the model fit that are data-driven and have an impact.

If we only think of the model as being important, we might end up accidentally overfitting to the data in-hand. This is very similar to the problems of "the garden of forking paths" and "p-hacking" (pdf).



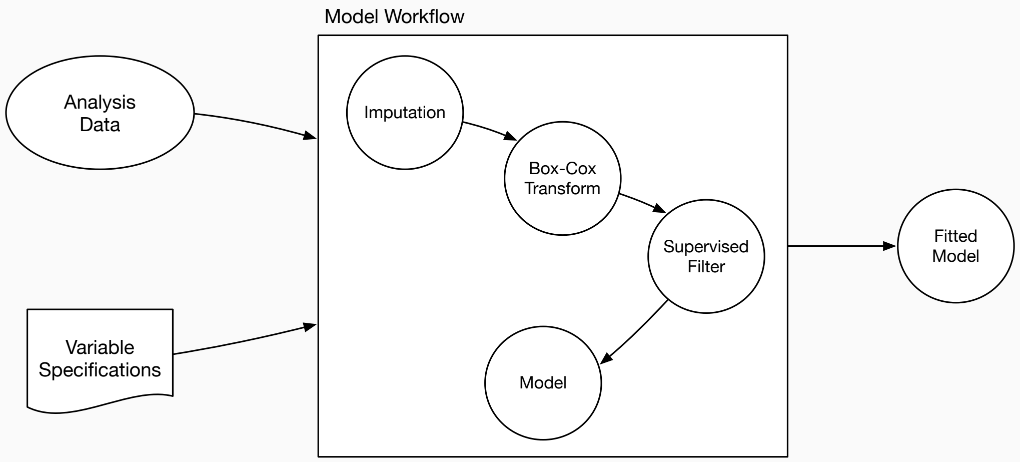
Define the Data Analysis Process

Let's conceptualize a process or *workflow* that involves all of the steps where the data are analyzed in a significant way. The includes the model but might also include other *estimation* steps:

- data preparation steps (e.g. imputation, encoding, transformations, etc)
- selection of which terms go into the model

and so on.

Admittedly, there is some grey area here.



This concept will become important when we talk about measuring performance of the modeling process.