Linear models

SDS 323
James Scott (UT-Austin)

Reference: Introduction to Statistical Learning Chapter 3

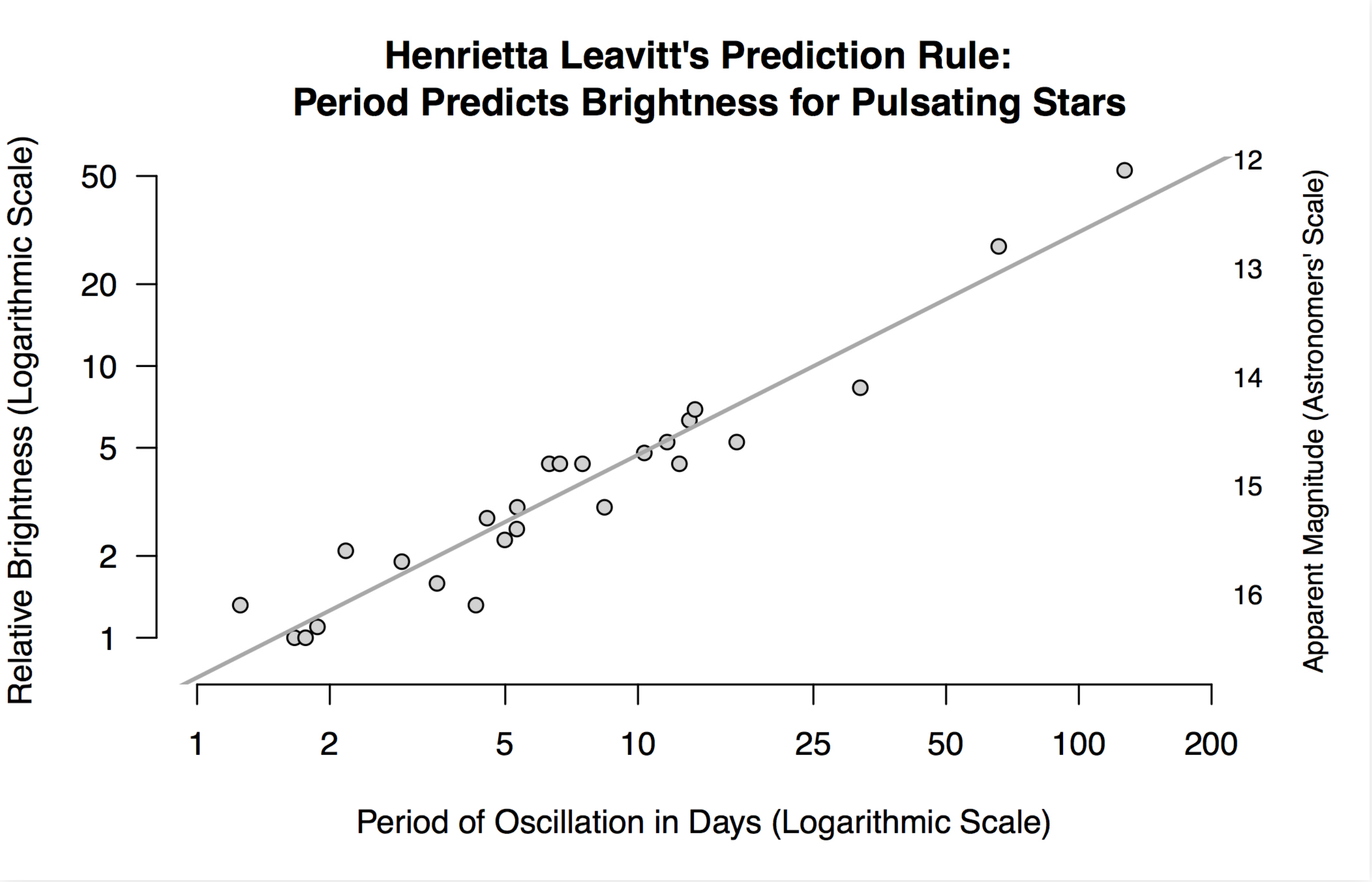
Linear models

Linear modeling is the most widely used tool in the world for fitting a predictive model of the form $y = f(x) + e$.

They are used throughout the worlds of science and industry.

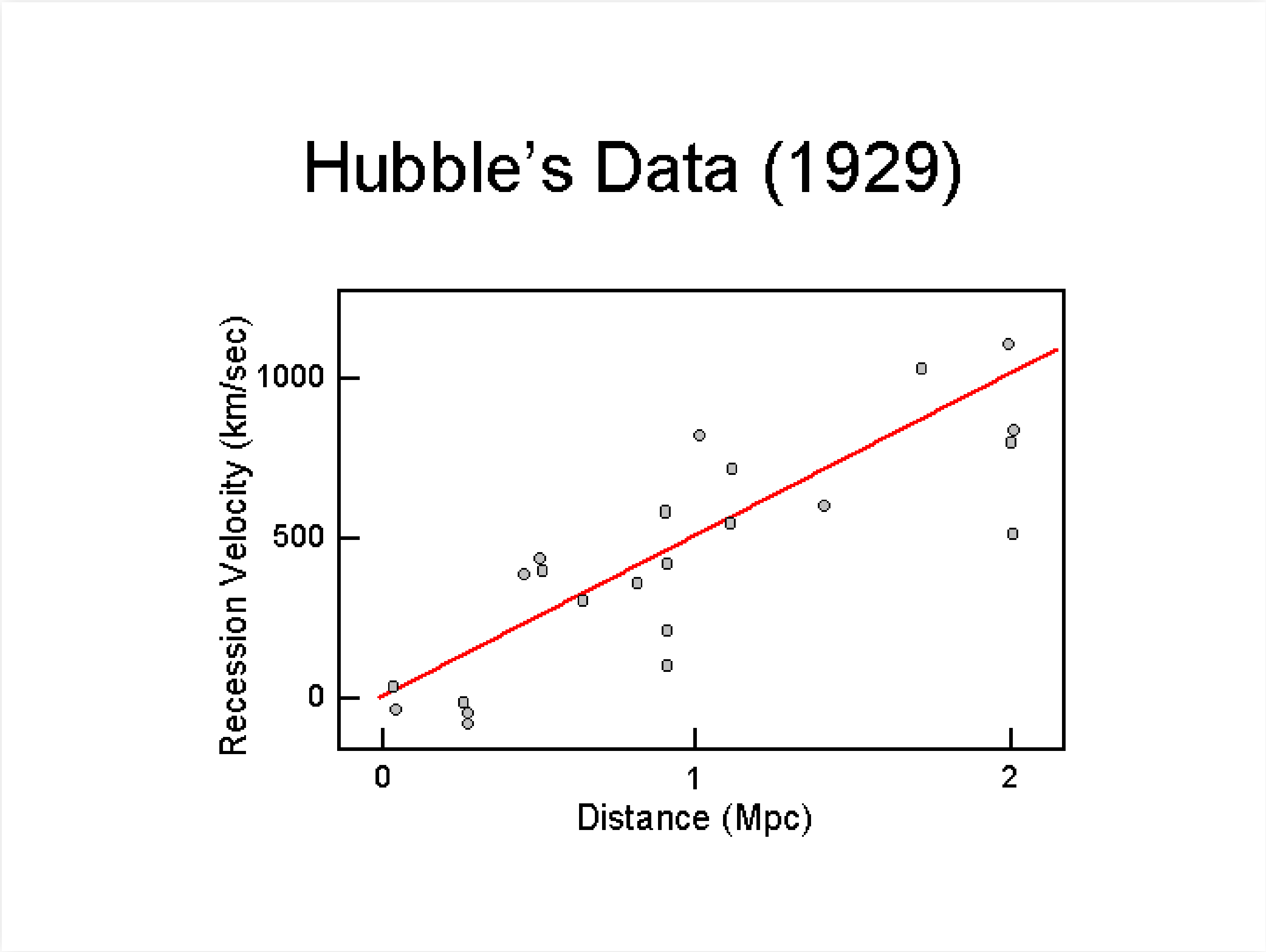
They have been at the heart of some of history's greatest scientific discoveries.

Linear models

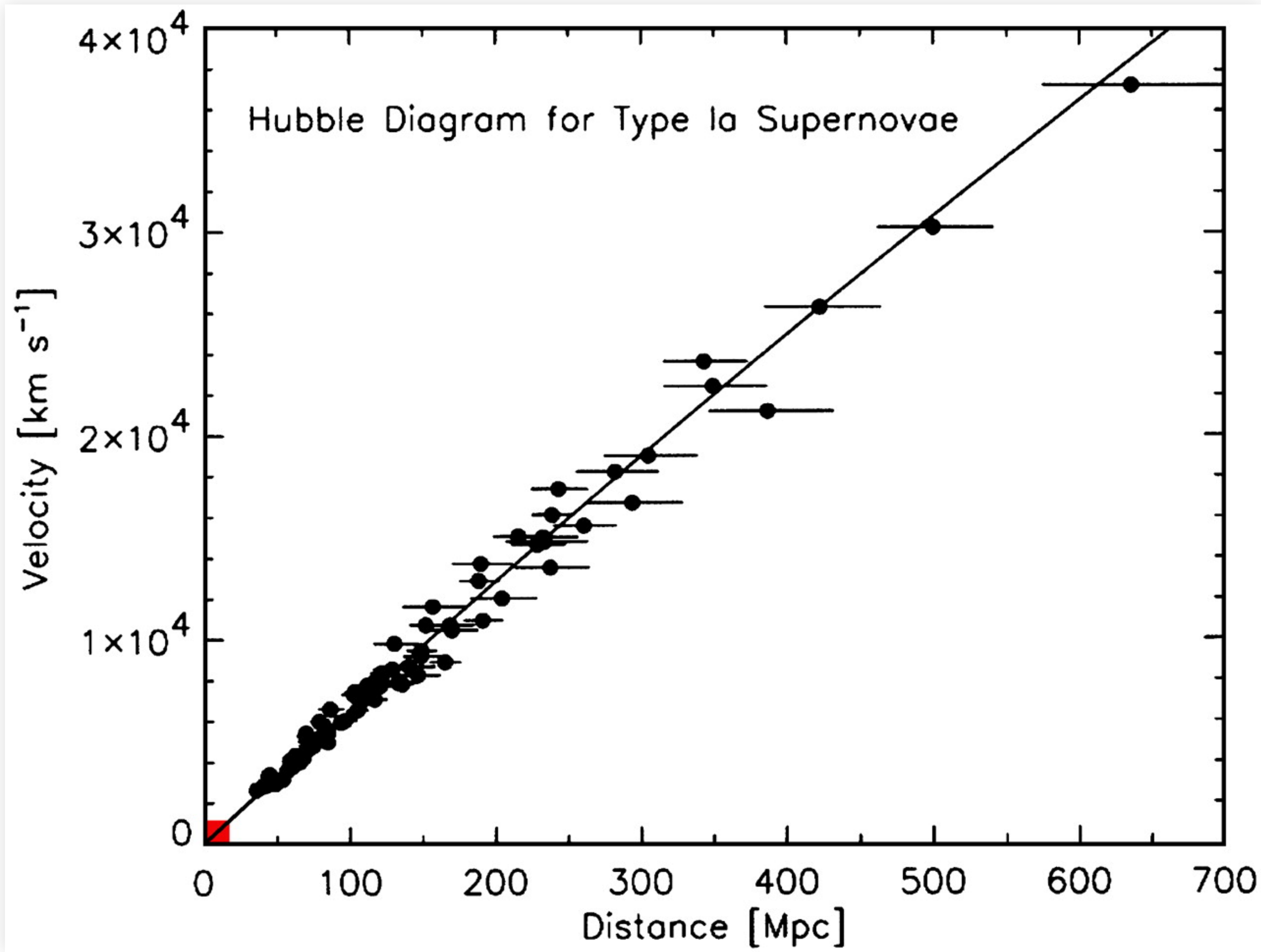


From *AIQ: How People and Machines are Smarter Together*

Linear models



Linear models



Linear models

A linear model is parametric model; we can can write down $f(x)$ in the form of an equation:

$$
\begin{aligned}
y &= \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + e \\
&= x \cdot \beta + e
\end{aligned}
$$

A notational convenience: the intercept gets absorbed into the vector of predictors by including a leading term of 1:

- $x = (1, x_1, x_2, \ldots, x_p)$
- $\beta = (\beta_0, \beta_1, \beta_2, \ldots, \beta_p)$

Linear models: major pros

- They often work pretty well for prediction.
- They're a simple foundation for more sophisticated techniques.
- They're easy to estimate, even for extremely large data sets.

Linear models: major pros

- They have lower estimation variance than most nonlinear/nonparametric alternatives.

- They're easier to interpret than nonparametric models.

- Techniques for feature selection (choosing which x_j's matter) are very well developed for linear models.

Linear models: major cons

- Linear models are pretty much always *wrong*, sometimes subtly and sometimes spectacularly. If the truth is nonlinear, then the linear model will provide a biased estimate.

- Linear models depend on which transformation of the data you use (e.g. x versus $\log(x)$).

- Linear models don't handle *interactions* among the predictors unless you explicitly build them in.

Huh? What's an interaction?

We use the term **interaction** in statistical learning to describe situations where the effect of some feature x on the outcome y is **context-specific.**:

- Biking in a low gear: easy (say 2 out of 10)
- Biking in a high gear: a bit hard (say 4 out of 10)
- Biking uphill in a low gear: a bit hard (say 5 out of 10)
- Biking uphill in a high gear: *very* hard (say 9 out of 10)

Huh? What's an interaction?

So what's the “effect” of change the gear from low to high? There is no one answer… **it depends on context!**

- On flat ground: “high gear” effect = 4 - 2 = 2.
- Uphill: “high gear” effect = 9 - 5 = 4.

One feature (slope) changes how another feature (gear) affects y.

That's an interaction.

Other simple examples

What's the effect of a 5 mph breeze on comfort? It depends!

- When it's hot outside, a 5 mph breeze makes things more pleasant.
- What about the effect of the same breeze when it's cold outside?

What's the effect of two Tylenol pills on a headache? It depends!

- This dose will help a headache in a 165 pound adult human being.
- What about the effect of that same dose on a 13,000 pound African elephant?

Linear models and interactions

Interactions are about capturing these context-specific effects by correctly modeling the *joint effect* of more than one feature at once.

The real world is full of interactions!

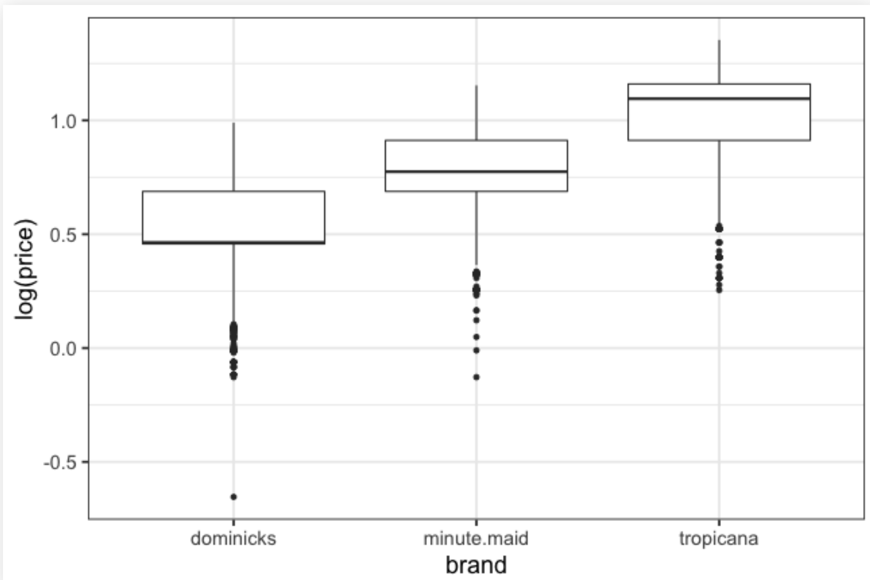
- house price versus (square footage, distance to downtown)
- college GPA versus (SAT Math score, college major)
- health outcomes vs (ACE inhibitors, pregnancy)

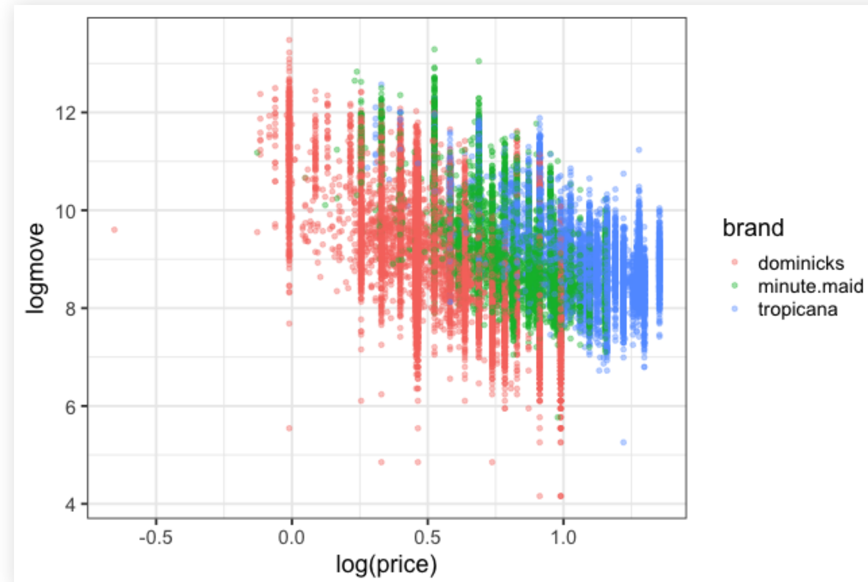
Linear models *can* accommodate interactions among feature variables—**but only** if we build these interaction terms in "by hand."

Example: orange juice sales

- Three brands of OJ: Tropicana, Minute Maid, Dominicks
- 83 Chicago-area stores
- Demographic info for each store
- Price, sales (log units moved), and whether advertised (`feat`)
- data in `oj.csv`, code in `oj.R`.

Example: orange juice sales





Each brand occupies a well-defined price range.

Sales decrease with price (duh).

OJ example: teaching points

- Log-linear models: thinking about *scale* in linear models.
- Interpreting regression models when some of the predictors are categorical (factor effects, model matrices).
- Interactions.

Log-linear models

- When fitting a linear model (*this* goes up, *that* goes down), think about the *scale* on which you expect to find linearity.

- A very common scale is to model the mean for $\log(y)$ rather than y. Remember, this allows us to model *multiplicative* rather than *additive* change.

$$log(y) = \log(\beta_0) + \beta_1 x \iff y = \beta_0 e^{\beta_1 x}$$

- Change x by one unit $\longrightarrow$ multiply $E(y \mid x)$ by e^{β_1} units.

Log-linear models

Whenever y changes on a percentage scale, use $\log(y)$ as a response:

- prices: “Foreclosed homes sell at a 20\% discount…”
- sales: “Our Super Bowl ad improved y.o.y. sales by 7% on average, across all product categories.”
- volatility, failures, weather events… lots of things that are non-negative are expressed most naturally on a log scale.

OJ: price elasticity

A simple "elasticity model" for orange juice sales y might be:

$$\log(y) = \gamma \log(\text{price}) + x \cdot \beta$$

The rough interpretation of a log-log regression like this: for every 1% increase in price, we can expect a $\gamma\%$ change in sales.

Let's try this in R, using `brand` as a feature (x):

```
reg = lm(logmove ~ log(price) + brand, data=oj)
coef(reg) ## look at coefficients
```

```
     (Intercept)       log(price) brandminute.maid   brandtropicana
      10.8288216       -3.1386914        0.8701747        1.5299428
```

OJ: model matrix and dummy variables

What happened to `branddominicks`?

Our regression formulas look like $\beta_0 + \beta_1 x_1 + \ldots$ But brand is not a number, so you can't do $\beta \cdot \mathrm{brand}$.

The first step of `lm` is to create a numeric "model matrix" from the input variables:

OJ: model matrix and dummy variables

Input:

Brand
dominicks
minute maid
tropicana

Variable expressed as a factor.

Output:

Intercept	**brand=minute maid**	**brand = tropicana**
1	0	0
1	1	0
1	0	1

Variable coded as a set of numeric "dummy variables" that we can multiply against β coefficients. This is done using `model.matrix`.

OJ: model matrix and dummy variables

Our OJ model used `model.matrix` to build a 4 column matrix:

```
> x <- model.matrix( ~ log(price) + brand, data=oj)
> x[1,]
Intercept log(price) branBDinute.maid brandtropicana
  1.00000   1.353255         0.000000       1.000000
```

Each factor's reference level is absorbed by the intercept. Coefficients are "change relative to reference level" (dominicks here).

To check the reference level of your factors, use

```
levels(oj$brand)
```

```
[1] "dominicks"   "minute.maid" "tropicana"
```

The first level is reference.

OJ: model matrix and dummy variables

To *change* the reference level, use `relevel`:

```
oj$brand2 = relevel(oj$brand, 'minute.maid')
```

Now if you re-run the regression, you'll see a different baseline category. But crucially, the price coefficient doesn't change:

```
reg2 = lm(logmove ~ log(price) + brand2, data=oj)
coef(reg)   # old model
```

```
      (Intercept)        log(price) brandminute.maid   brandtropicana
       10.8288216        -3.1386914        0.8701747        1.5299428
```

```
coef(reg2) # new model
```

```
     (Intercept)       log(price) brand2dominicks brand2tropicana
      11.6989962       -3.1386914      -0.8701747       0.6597681
```

Interactions

Remember: an interaction is when one feature changes how another feature acts on y.

In regression, an interaction is expressed as the product of two features:

$$E(y \mid x) = f(x) = \beta_0 + \beta_1 x_2 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \cdots$$

so that the effect on y of a one-unit increase in x_1 is $\beta_1 + \beta_{12} x_2$. (It depends on x_2!)

Interactions

Interactions play a *massive* role in statistical learning. They are important for making good predictions, and they are often central to social science and business questions:

- Does gender change the effect of education on wages?
- Do patients recover faster when taking drug A?
- How does brand affect price sensitivity?

Interactions in R: use *

This model statement says: elasticity (the `log(price)` coefficient) is different from one brand to the next:

```
reg3 = lm(logmove ~ log(price)*brand, data=oj)
coef(reg3)
```

```
                (Intercept)                      log(price)
                10.95468173                     -3.37752963
            brandminute.maid                  brandtropicana
                 0.88825363                      0.96238960
log(price):brandminute.maid   log(price):brandtropicana
                 0.05679476                      0.66576088
```

This output is telling us that the elasticities are:

dominicks: -3.4   minute maid: -3.3   tropicana: -2.7

Where do these numbers come from? Do they make sense?

Interactions: OJ

- A key question: what changes when we feature a brand? Here, this means in-store display promo or flier ad (`feat` in `oj.csv`):

  1. You could model the additive effect of `feat` on log sales volume…
  2. Or (1) *and* an overall effect of `feat` on price elasticity…
  3. Or you could model a *brand-specific* effect of `feat` on elasticity.

- Let's see the R code in `oj.R` for runs of all three models.

- Goal: connect the regression formula and output to a specific equation, i.e. $f(x) = E(y \mid x)$ for each brand individually.

Brand-specific elasticities

	Dominicks	**Minute Maid**	**Tropicana**
Not featured	-2.8	-2.0	-2.0
Featured	-3.2	-3.6	-3.5

Findings:

- Ads always decrease elasticity.
- Minute Maid and Tropicana elasticities drop 1.5% with ads, moving them from less to more price sensitive than Dominicks.

Why does marketing increase price sensitivity? And how does this influence pricing/marketing strategy?

Brand-specific elasticities

Before including feat, Minute Maid behaved like Dominicks.

```
reg_interact = lm(logmove ~ log(price)*brand, data=oj)
coef(reg_interact)
```

```
                (Intercept)                  log(price)
                10.95468173                 -3.37752963
           brandminute.maid              brandtropicana
                 0.88825363                  0.96238960
log(price):brandminute.maid   log(price):brandtropicana
                 0.05679476                  0.66576088
```

(Compare the elasticities!)

Brand-specific elasticities

With feat, Minute Maid looks more like Tropicana. Why?

```
reg_ads3 <- lm(logmove ~ log(price)*brand*feat, data=oj)
coef(reg_ads3)
```

```
                   (Intercept)                     log(price) 
                   10.40657579                    -2.77415436 
             brandminute.maid                 brandtropicana 
                    0.04720317                     0.70794089 
                          feat      log(price):brandminute.maid 
                    1.09440665                     0.78293210 
     log(price):brandtropicana                log(price):feat 
                    0.73579299                    -0.47055331 
         brandminute.maid:feat            brandtropicana:feat 
                    1.17294361                     0.78525237 
log(price):brandminute.maid:feat   log(price):brandtropicana:feat 
                   -1.10922376                    -0.98614093 
```

(Again, compare the elasticities!)

What happened?

Minute Maid was more heavily promoted!

```
    brand
feat dominicks minute.maid tropicana
   0     0.743       0.711     0.834
   1     0.257       0.289     0.166
```

Because Minute Maid was more heavily promoted, AND promotions have a negative effect on elasticity, we were confounding the two effects on price when we estimated an elasticity common to all brands.

Including `feat` helped deconfound the estimate!

Take-home messages: transformations

- Transformations help us find the most natural scale on which to express the relationship between x and y. The log transformation is easily the most common.
- Exponential growth/decay in y versus x:
$$\widehat{log(y)} = \beta_0 + \beta_1 x \iff \hat{y} = e^{\beta_0} \cdot e^{\beta_1 x}$$
- Power law (elasticity model) in y versus x:
$$\widehat{log(y)} = \beta_0 + \beta_1 \log(x) \iff \hat{y} = e^{\beta_0} \cdot x^{\beta_1}$$

Take-home messages: dummy variables

- Categorical variables are encoded using dummies. Happens behind the scenes, but to interpret the output correctly, it's important to know how it works.

Intercept	**brand=minute maid**	**brand = tropicana**
1	0	0
1	1	0
1	0	1

- One category is arbitrarily chosen as the baseline/intercept. Any baseline is as good as another:
  - JJ Watt is 6'5", Yao Ming is +13 inches, Simone Biles is -21 inches.
  - Simone Biles is 4'8", Yao Ming is +34 inches, JJ Watt is +21 inches.

Take-home messages: interactions

- An interaction is when one variable changes the effect of another variable on y.

- In linear models, we express interactions as products of variables:
$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \cdots$$
so that the effect on y of a one-unit increase in x_1 is $\beta_1 + \beta_{12} x_2$.

Out-of-sample fit: linear models

- Just like with K-nearest neighbors in one x variable, it's important to consider out of sample fit for a linear model.
- Especially in a model with lots of variables, the in-sample fit and out-of-sample fit can be very different.
- Let's see these ideas in practice, by comparing three predictive models for house prices in Saratoga, New York.

Variables in the data set

- price (y)
- lot size, in acres
- age of house, in years
- living area of house, in square feet
- percentage of residents in neighborhood with college degree
- number of bedrooms
- number of bathrooms
- number of total rooms
- number of fireplaces
- heating system type (hot air, hot water, electric)
- fuel system type (gas, fuel oil, electric)
- central air conditioning (yes/no)

Three models for house prices

We'll consider three possible models for price constructed from these 11 predictors.

- Small model: price versus lot size, bedrooms, and bathrooms (4 total parameters, including the intercept).
- Medium model: price versus all variables above, main effects only (14 total parameters, including the dummy variables).
- Big model: price versus all variables listed above, together with all pairwise interactions between these variables (90 total parameters, include dummy variables and interactions).

Three models for house prices

A starter script is in `saratoga_lm.R`. Goals for today:

- See if you can "hand-build" a model for price that outperforms all three of these baseline models! Use any combination of transformations, polynomial terms, and interactions that you want.
- When measuring out-of-sample performance, there is *random variation* due to the particular choice of data points that end up in your train/test split. Make sure your script addresses this by averaging the estimate of out-of-sample RMSE over many different random train/test splits.

Three models for house prices

```
# Split into training and testing sets
n = nrow(SaratogaHouses)
n_train = round(0.8*n)  # round to nearest integer
n_test = n - n_train
train_cases = sample.int(n, n_train+1, replace=FALSE)
test_cases = setdiff(1:n, train_cases)
saratoga_train = SaratogaHouses[train_cases,]
saratoga_test = SaratogaHouses[test_cases,]


# Fit to the training data
lm1 = lm(price ~ lotSize + bedrooms + bathrooms, data=saratoga_train)
lm2 = lm(price ~ . - sewer - waterfront - landValue - newConstruction,
data=saratoga_train)
lm3 = lm(price ~ (. - sewer - waterfront - landValue - newConstruction)^2,
data=saratoga_train)
```

Three models for house prices

```
# Predictions out of sample
yhat_test1 = predict(lm1, saratoga_test)
yhat_test2 = predict(lm2, saratoga_test)
yhat_test3 = predict(lm3, saratoga_test)

rmse = function(y, yhat) {
  sqrt( mean( (y - yhat)^2 ) )
}

# Root mean-squared prediction error
rmse(saratoga_test$price, yhat_test1)
```

```
[1] 76718.89
```

```
rmse(saratoga_test$price, yhat_test2)
```

```
[1] 66710.4
```

```
rmse(saratoga_test$price, yhat_test3)
```

```
[1] 79040.66
```

Comparison with K-nearest-neighbors

- A linear model makes strong assumptions about the form of $f(x)$.
- If these assumptions are wrong, then a linear model can predict poorly (large bias).
- A nonparametric model like KNN is different:
  - It makes fewer assumptions (and thus can have lower bias).
  - You don't have to “hand build" interactions into the model.
  - But it typically has higher estimation variance.
- Which approach leads to a more favorable spot along the bias-variance tradeoff?

Comparison with K-nearest-neighbors

Let's fit a KNN model to the Saratoga house-price data using all the same variables we used in our “medium” model. Recall the basic recipe:

- Given a value of K and a point x_0 where we want to predict, identify the K nearest training-set points to x_0. Call this set $\mathcal{N}_0$.
- Then estimate $\hat{f}(x_0)$ using the average value of the target variable y for all the points in $\mathcal{N}_0$.

$$\hat{f}(x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} y_i$$

(The summation over $i \in \mathcal{N}_0$ means “sum over all data points i that fall in the neighborhood.”)

Measuring closeness

There's a caveat here: how do we measure which K points are the closest to x_0? To see why this is trickier than it sounds at first, consider three houses:

- House A: 4 bedrooms, 2 bathrooms, 2200 sq ft
- House B: 4 bedrooms, 2 bathrooms, 2300 sq ft
- House C: 2 bedrooms, 1.5 bathrooms, 2125 sq ft

Which house (B or C) should be “closer to” A?

Measuring closeness

Most people would that A and B are the most similar: they both have 4 BR/2 BA, and their size is similar. House C is has only 2BR/1.5 BA; it's *enormous* for a house with that number of bedrooms. Yet if we naively try to calculate pairwise Euclidean distances, we find the following:

$$\begin{aligned}
d(A, B) &= \sqrt{(4-4)^2 + (2-2)^2 + (2200-2300)^2} \\
&= 100 \\
d(A, C) &= \sqrt{(4-2)^2 + (2-1.5)^2 + (2200-2125)^2} \\
&\approx 75.03
\end{aligned}$$

So A's nearest neighbor is C, not B!

Measuring closeness

- What happened here is that the `sqft` variable completely overwhelmed the BR and BA variables in the distance calculations.

- The root of the problem: square footage is measured on an *entirely different scale* than bedrooms or bathrooms. Treating them as if they're on the same scale leads to counter-intuitive distance calculations.

- The simple way around this: weighting. That is, we treat some variables as more important than others in the distance calculation.

Weighting

Ordinary Euclidean distance:

$$d(x_1, x_2) = \sqrt{\sum_{j=1}^{p} \left\{x_{1,j} - x_{2,j}\right\}^2}$$

Weighted Euclidean distance:

$$d_w(x_1, x_2) = \sqrt{\sum_{j=1}^{p} \left\{w_j \cdot (x_{1,j} - x_{2,j})\right\}^2}$$

This depends upon the choice of weights $w_1, \ldots, w_p$ for each feature variable.

Weighting

We can always choose the scales/weights to reflect our substantive knowledge of the problem. For example:

- a "typical" bedroom is about 200 square feet (roughly 12x16).
- a "typical" bathroom is about 50 square feet (roughly 8x6).

This might lead us to scales like the following:

$$d_w(x, x') = \sqrt{(x_1 - x_1')^2 + \left\{200(x_2 - x_2')\right\}^2 + \left\{50(x_3 - x_3')\right\}^2}$$

where x_1 is square footage, x_2 is bedrooms, and x_3 is bathrooms. Thus a difference of 1 bedroom counts 200 times as much as a difference of 1 square foot.

Weighting by scaling

But choosing weights by hand can be a real pain.

A "default" choice of weights that requires no manual specification is to weight by the inverse standard deviation of each feature variable:

$$d_w(x_1, x_2) = \sqrt{\sum_{j=1}^{p} \left(\frac{x_{1,j} - x_{2,j}}{s_j}) \right)^2}$$

where s_j is the sample standard deviation of feature j across all cases in the training set.

Weighting

This is equivalent to calculating "ordinary" distances using a rescaled feature matrix, where we center and scale each feature variable to have mean 0 and standard deviation 1:

$$\tilde{X}_{ij} = \frac{(x_{ij} - \mu_j)}{s_j}$$

where (μ_j, s_j) are the sample mean and standard deviation of feature variable j.

Then we can run ordinary (unweighted) KNN using Euclidean distances based on $\tilde{X}$.

Example: KNN on the house-price data

```
# construct the training and test-set feature matrices
# note the "-1": this says "don't add a column of ones for the intercept"
Xtrain = model.matrix(~ . - (price + sewer + waterfront + landValue +
newConstruction) - 1, data=saratoga_train)
Xtest = model.matrix(~ . - (price + sewer + waterfront + landValue +
newConstruction) - 1, data=saratoga_test)

# training and testing set responses
ytrain = saratoga_train$price
ytest = saratoga_test$price

# now rescale:
scale_train = apply(Xtrain, 2, sd)  # calculate std dev for each column
Xtilde_train = scale(Xtrain, scale = scale_train)
Xtilde_test = scale(Xtest, scale = scale_train)  # use the training set
scales!
```

Example: KNN on the house-price data

```
head(Xtrain, 2)
```

```
    lotSize age livingArea pctCollege bedrooms fireplaces bathrooms rooms
142    0.33  32       1898         64        3          1       1.5     7
51     0.24  10       1782         52        3          0       2.5     6
    heatinghot air heatinghot water/steam heatingelectric fuelelectric
142              0                      1               0            0
51               1                      0               0            0
    fueloil centralAirNo
142       0            1
51        0            0
```

```
head(Xtilde_train, 2) %>% round(3)
```

```
    lotSize    age livingArea pctCollege bedrooms fireplaces bathrooms
142  -0.248  0.147      0.237      0.818    -0.17      0.731    -0.611
51   -0.394 -0.612      0.050     -0.351    -0.17     -1.091     0.911
     rooms heatinghot air heatinghot water/steam heatingelectric
142 -0.011         -1.335                  2.171          -0.475
51  -0.441          0.749                 -0.460          -0.475
    fuelelectric fueloil centralAirNo
142       -0.483  -0.372        0.760
51        -0.483  -0.372       -1.314
```

Example: KNN on the house-price data

```
library(FNN)
K = 10

# fit the model
knn_model = knn.reg(Xtilde_train, Xtilde_test, ytrain, k=K)

# calculate test-set performance
rmse(ytest, knn_model$pred)
```

```
[1] 67610.25
```

```
rmse(ytest, yhat_test2)  # from the linear model with the same features
```

```
[1] 66710.4
```

Example: KNN on the house-price data

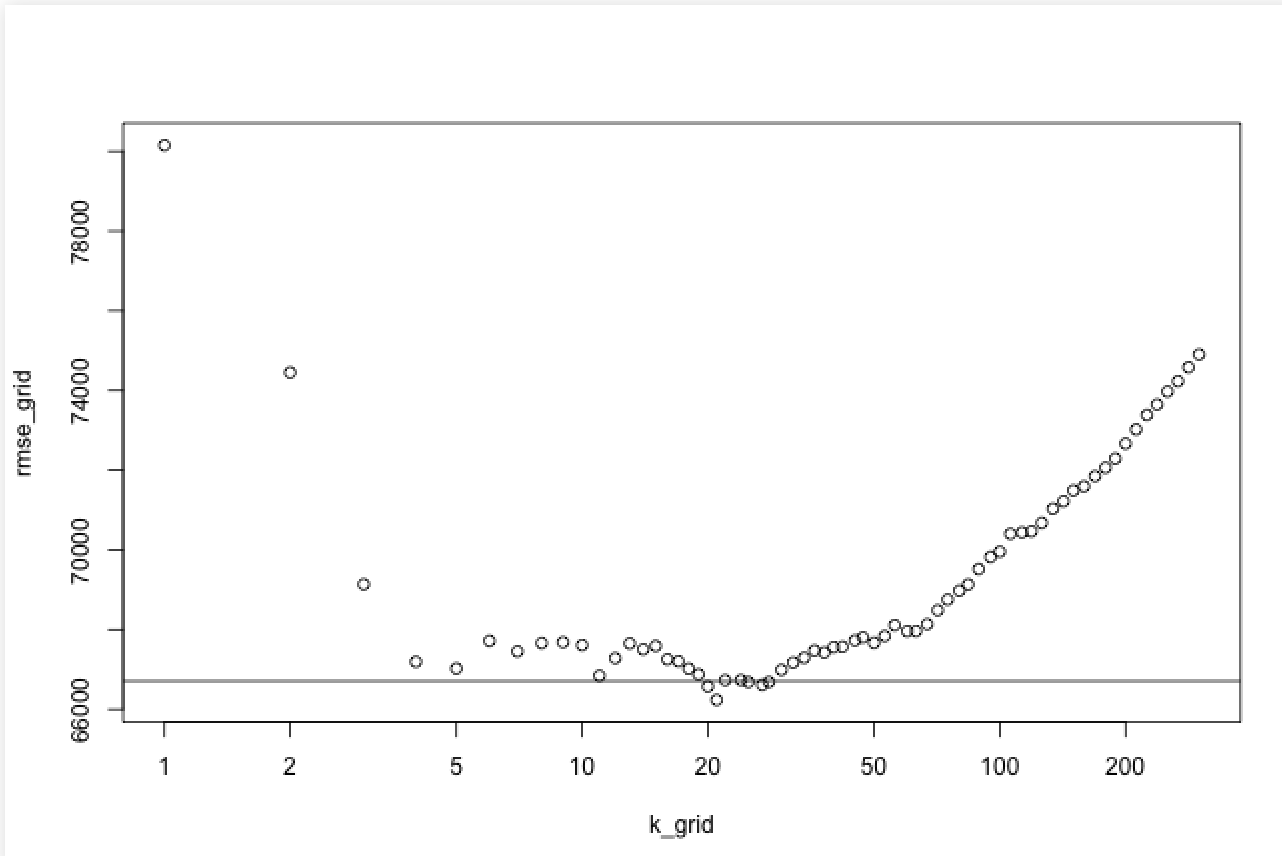
Let's try many values of K:

```
library(foreach)

k_grid = exp(seq(log(1), log(300), length=100)) %>% round %>% unique
rmse_grid = foreach(K = k_grid, .combine='c') %do% {
  knn_model = knn.reg(Xtilde_train, Xtilde_test, ytrain, k=K)
  rmse(ytest, knn_model$pred)
}
```

Example: KNN on the house-price data

```
plot(k_grid, rmse_grid, log='x')
abline(h=rmse(ytest, yhat_test2)) # linear model benchmark
```



Back to your "favorite" linear model

- Return to your “hand-built” model for price, which might have included transformations, newly defined variables, etc.
- See if you can turn that linear model into a better-performing KNN model.
- Note: don't explicitly include interactions or polynomial terms in your KNN model! It is sufficiently adaptable to find them, if they are there.

Take-home messages

- "Feature selection" is an important component of building a linear model.
- It can be used for its own sake (i.e. to build a linear predictive model) or as a pre-processing step to choose features for a non-linear model.
- But selecting features by hand is laborious, and selecting interactions by hand is even worse! Stay tuned for more automated methods :-)