Linear Modeling with R

Clay Ford

What are Linear Models?

- Linear Models are mathematical representations of the process that (*we think*) gave rise to our data.
- They seek to explain the relationship between a continuous variable of interest, our *response* or *dependent* variable, and one or more *predictor* or *independent* variables.
- Linear models are basically weighted sums of our independent variables.
- We call them "linear models" because the weights (or parameters) are additive. They don't appear as an exponent or get multiplied or divided by each other.
- Linear often refers to straight lines, but linear models can be curved.

Example of a linear model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

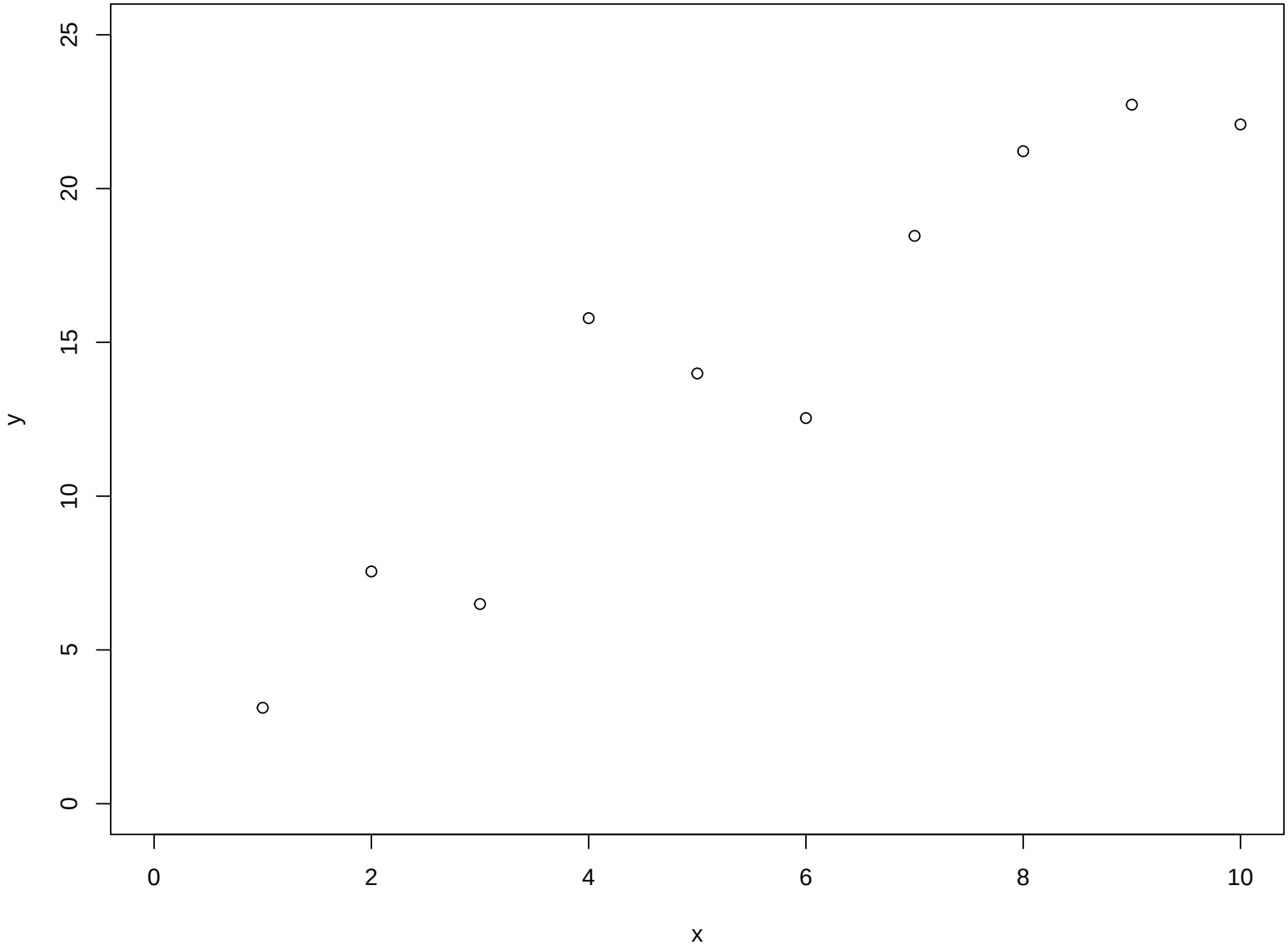
- Y is the response
- X_1 and X_2 are the predictors
- $\beta_0, \beta_1, \beta_2$ are coefficients
- ϵ is random error. Our model will not perfectly predict Y. It will be off by some random amount. We assume this amount is a random draw from a Normal distribution with mean 0 and standard deviation σ

Building a linear model means we propose a linear model and then estimate the coefficients and the standard deviation of the error term. Above, this means estimating $\beta_0, \beta_1, \beta_2$ and σ. This is what we do in R.

Proposing a model in 2 dimensions

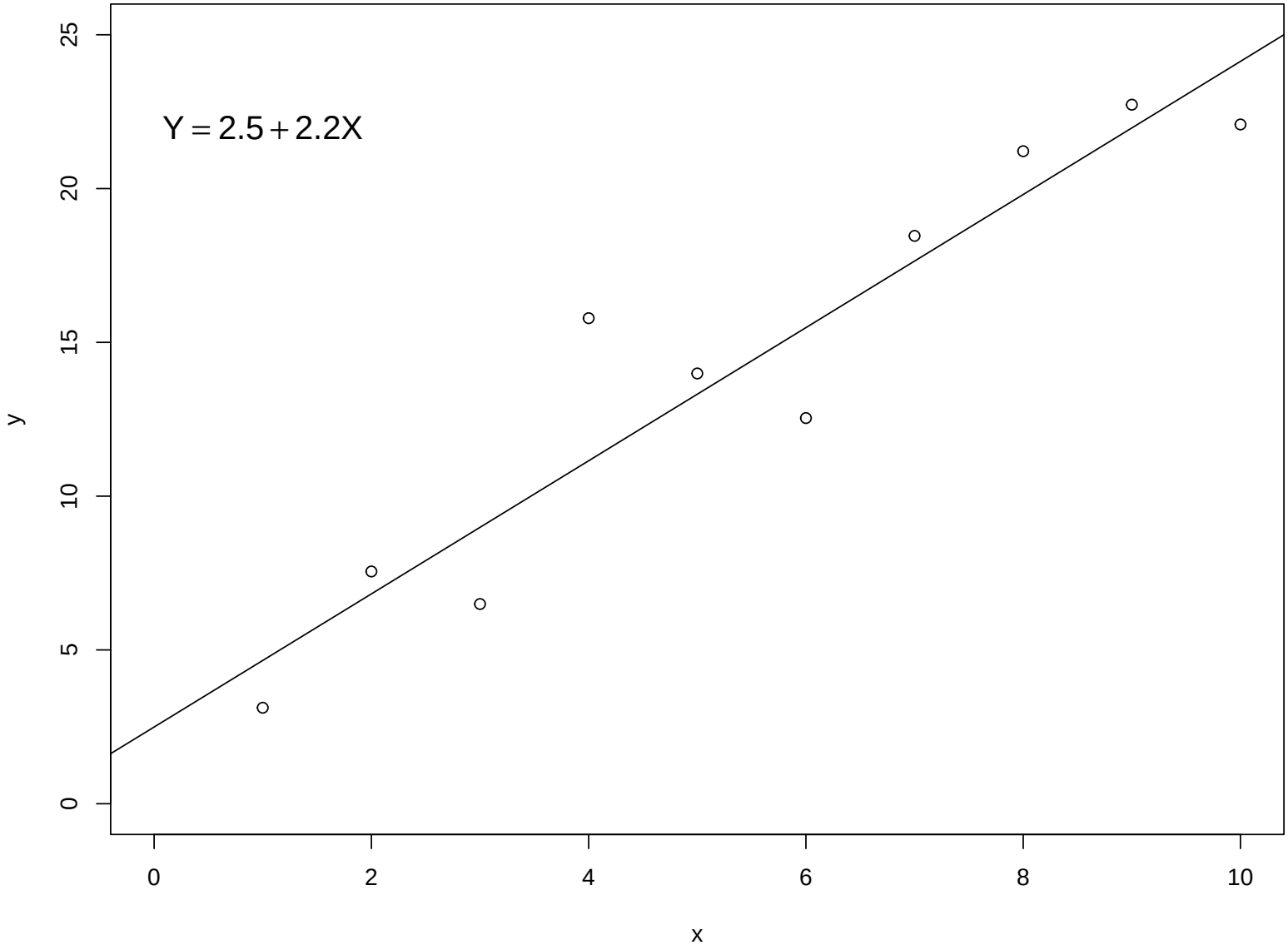
It appears this data came from a straight line model:

$Y = \beta_0 + \beta_1 X + \epsilon$.



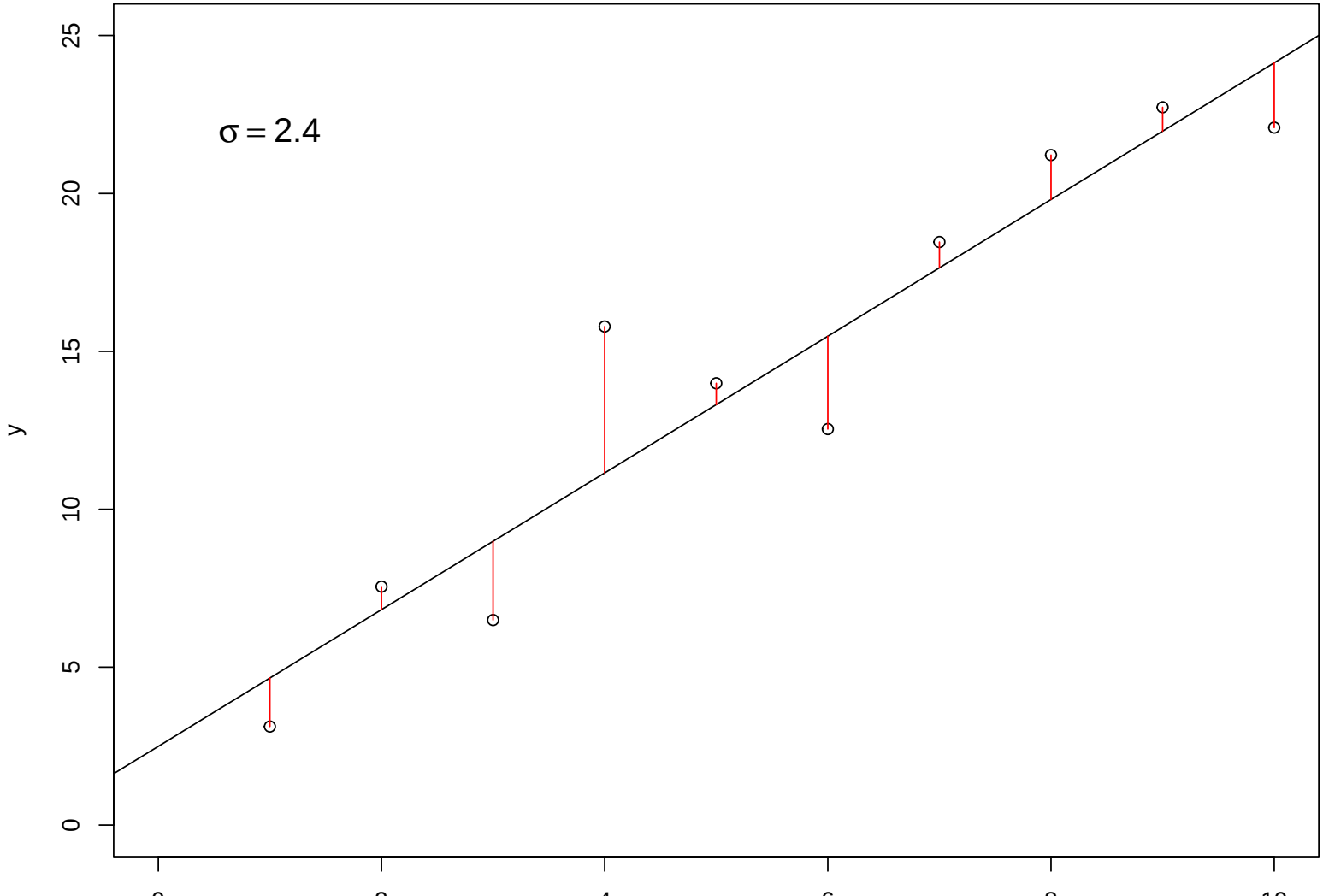
Building a model

Statistics allows us to *fit* a model to the data. Below is the best fitting line for this data.



Model error

Our model doesn't perfectly predict Y. The length of the red lines (the residuals) are used to estimate the error of our model. We can think of the error as sort of like the average absolute distance of the red lines.



fitting a linear model in R: import data

Most any kind of data can be read into R. The usual form of the function is `read.x`. Examples:

- **CSV**: `mydata <- read.csv(file="file.csv")`
- **TXT**: `mydata <- read.table(file="file.txt", header=TRUE)`
- **Fixed-width**: `mydata <- read.fwf(file="file.dat", widths = c(4,3,9))`

You can also use point-and-click in R Studio: "Import Dataset" button.

The `haven` package allows you to read in data from other programs, such as SPSS, Stata and SAS.

fitting a linear model in R: structure of data

Data should be *tidy*, that is

- Each variable should be a column
- Each observation should be a row
- Repeated observations on an object should be separate rows

Example

```
##   Subject time conc
## 1       1 0.25 1.50
## 2       1 0.50 0.94
## 3       1 0.75 0.78
```

Also, there is no need to create dummy variables or interactions in advance. R can do this for us.

fitting a linear model in R: examine data

Make sure you and R both agree what your data look like.

- numbers were read in as numbers
- known missing data is coded as missing (`NA`, not 999)
- how much missing data do you have?
- how are your variables distributed? (lots of zeros? Outliers?)

Two basic functions:

1. `summary(mydata)`: Statistical summaries of all variables
2. `str(mydata)`: Structure of data

fitting a linear model in R: exploratory plots

Good idea to visually examine your data before building a model. Helps you...

- spot potential outliers or errors
- determine if transformations may be necessary

Three basic plots:

1. `hist(response)`: Histogram of response
2. `plot(response ~ predictor, data=mydata)`: scatterplot/boxplots
3. `pairs(mydata)`: pairwise scatter plots

The `car` package has two nice exploratory plotting functions: `scatterplot` and `scatterplotMatrix`.

Fitting a linear model in R: the lm function

- The basic function is `lm`. The main arguments are `formula` and `data`.
- The "formula" is the linear model expressed in what is called *Wilkinson-Rogers* notation. (*More on that later*)
- To fit $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$ do the following:
  `lm(formula=Y ~ X1 + X2, data=mydata)`
- Or more concisely:
  `lm(Y ~ X1 + X2, mydata)`

Saving and working with a linear model in R

- Results of a linear model can be saved, like so:
  `lm1 <- lm(Y ~ X1 + X2, mydata)`
- `lm1` is a linear model *object* that contains various quantities of interest.
- Use *extractor* functions to view the different quantities.
  Common ones are `summary`, `coef`, `residuals`, and `fitted`.
- For example, `summary(lm1)`.

A closer look at the R model summary

```
Call:
lm(formula = price ~ finsqft + bedrooms + lotsize, data = s
```

Repeat of the function call. Useful if result is saved and then printed later.

```
Residuals:
    Min      1Q  Median      3Q     Max
-219284  -38284   -5933   29009  376303
```

Quick check of the distributional (Normal) assumptions of residuals. Median should not be far from 0. Max and Min, and 1Q and 3Q, should be roughly equal in absolute value.

A closer look at the R model summary

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -8.761e+04  1.418e+04  -6.177 1.33e-09 ***
finsqft      1.634e+02  5.804e+00  28.149  < 2e-16 ***
...
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘
```

- **Estimate**: $\hat{\beta}$
- **Std. Error**: standard error of the $\hat{\beta}$
- **t value**: statistic for test $H_0 : \hat{\beta} = 0$ *in the model* (Estimate $\div$ Std. Error)
- **Pr(>|t|)**: p-value of hypothesis test (2-sided)
- **Signif. codes**: indicators of significance; one star means $0.01 < p < 0.05$

A closer look at the R model summary

```
Residual standard error: 0.7679 on 89 degrees of freedom
Multiple R-squared:  0.5893,  Adjusted R-squared:  0.557
F-statistic: 18.24 on 7 and 89 DF,  p-value: 7.694e-15
```

- **Residual standard error**: $\hat{\sigma}$
- **degrees of freedom**: # of obs - # of parameters (the $\hat{\beta}$s)
- **Multiple R-squared**: measure of model fit (0,1)
- **Adjusted R-squared**: measure of model fit adjusted for number of parameters (0,1)
- **F-statistic**: statistic for hypothesis test all coefficients (other than intercept) $= 0$
- **p-value**: p-value of hypothesis test

Confidence intervals for $\hat{\beta}$

- Confidence intervals allow us to express uncertainty in our estimates.
- They tell us about plausible values for parameters that hypothesis tests cannot.
- To extract from model in R, use `confint` function. Output includes lower and upper bounds.
- Default is 95%, but can be modified using the `level` argument.

Confidence intervals for predictions

- We can make predictions using our linear model, but there will be uncertainty. We'd like to quantify that uncertainty.
- Two forms of confidence intervals (CI) for prediction:

1. CI of predicted mean (predicted *mean* given these predictors)
2. CI of predicted value (predicted *value for an individual* with given predictors)

- The second is wider due to the increased uncertainty of predicting a specific value versus predicting a mean.
- To make predictions for a fitted model, use the `predict` function. Output includes fit and lower/upper bounds.

Using the `predict` function in `R`

- The only required argument to `predict` is the fitted model object.
- Use the `interval` argument to specify type of confidence interval:
  `interval="confidence"` for predicted mean
  `interval="prediction"` for predicted value
- Use the `newdata` argument to make predictions using new data (ie, data not used to build model). New data must be a data frame.

Model specification in R

- R uses the *Wilkinson-Rogers* notation for specifying models:

  response variable $\sim$ *explanatory variable(s)*

- The tilde ($\sim$) can be read as "is modeled as a function of" or "described by" or "regressed on".
- Model formulation is used throughout R in plotting, aggregation and statistical tests.

Model formula symbols

Symbols are used differently in R models:

- $+$ inclusion of variable
- $-$ deletion of variable (not subtraction)
- $*$ inclusion of variables *and their interactions* (not multiplication)
- : interaction of variables
- $\wedge$ interaction of variables to specified degree (not exponent)

To override model symbol, use the `I()` function.

See `help(formula)` for more information.

Examples of model specifications

- `y ~ x` (simple linear regression)
- `y ~ x + z` (multiple regression)
- `y ~ .` (multiple regression for all variables in data set)
- `y ~ x + z - 1` (multiple regression without intercept)
- `y ~ x + z + x:z` (multiple regression with interaction)
- `y ~ x * z` (same as previous)
- `y ~ u + x + z + u:x + u:z + x:z + u:x:z` (multiple regression with all interactions)
- `y ~ u * x * z` (same as previous)
- `y ~ (u + x + z)^2` (multiple regression with all 2-way interactions)
- `y ~ x + I(x^2)` ("raw" polynomial regression, degree 2)
- `y ~ poly(x, 2)` (orthogonal polynomial regression, degree 2)
- `y ~ ns(x, 2)` (spline regression, df $= 2$; `library(splines)`)

Using categorical predictors in model building

- So far we have only considered numerical predictors. What about categorical predictors such as Male/Female, Democrat/Republican/Independent, Low/Medium/High, etc.?
- R requires categorical predictors be encoded as *factors*.
- A factor is a set of integer codes with associated *levels*. With categorical variables encoded as factors, R automatically and correctly incorporates them into a linear model.
- Either define a variable as a factor in a data frame or in the `lm` formula:
  - `sales$quality <- factor(sales$quality)`
  - `lm(price ~ factor(quality), data=sales)`
- Recommended to define variable as factor in the data frame.

How factors are modeled

- By default factors are modeled using *treatment contrasts*.
- This means one level is treated as baseline and the other levels have coefficients that express change from baseline.
- To make this happen R automatically codes the factor levels as dummy variables.
- Say we have variable `level` with three levels: low, medium, high. R codes as follows:

```
##        medium high
## low         0    0
## medium      1    0
## high        0    1
```

How factors are reported in output

The baseline is not listed per se but is pulled into the intercept. The coefficients on the other levels represent difference from baseline.

```
## lm(formula = resp ~ level, data = test)

##                Estimate
## (Intercept) 10.12061
## levelmedium  9.86895
## levelhigh   14.86689
```

- mean "resp" for level=low is about 10
- mean "resp" for level=medium is about 10 more than "low", so 20
- mean "resp" for level=high is about 15 more than "low", so 25

How factors work in interactions

- If the effect of a variable depends on another variable, we say the variables *interact*.
- We're often not sure if variables interact or not. That's why we include them in the model: to see if they significantly improve the model.
- Interacting a k-level factor with a numeric variable yields a separate parameter estimate for the numeric variable at $k-1$ levels of the factor.
- Interacting a k-level factor with a j-level factor yields parameter estimates for $(k-1) \times (j-1)$ combinations of levels.

How `factor:numeric` interactions are reported in output

```
## lm(formula = resp ~ level * num, data = test)

##                      Estimate
## (Intercept)        9.48891334
## levelmedium        9.47340251
## levelhigh         13.54821543
## num                0.21753681
## levelmedium:num   -0.08427390
## levelhigh:num     -0.06587283
```

How `factor:numeric` interactions work

- With one factor and one numeric variable we have a *varying intercept and slope* model. The intercept and slope vary based on the level.
- Model when level=low
  - $9.489 + 0.217 * num$
- Model when level=high (note how the intercept and slope change)
  - $(9.489 + 13.548) + (0.217 - 0.066) * num$
- This is also knows as an Analysis of Covariance (ANCOVA).

How `factor:factor` interactions are reported in output

```
## lm(formula = resp ~ level * pos, data = test)

##                            Estimate
## (Intercept)              10.0399195
## levelmedium              10.0784283
## levelhigh                14.7078325
## poscenter                -0.4224549
## posright                  0.6645173
## levelmedium:poscenter     0.2980911
## levelhigh:poscenter       0.7959860
## levelmedium:posright     -0.9265268
## levelhigh:posright       -0.3188136
```

How `factor:factor` interactions work

- Mean response value when level=low and position=left
  - 10.0399
- Mean response value when level=low and position=center
  - $10.0399 - 0.4225 = 9.6174$
- Mean response value when level=high and position=right
  - $10.0399 + 14.7078 + 0.6645 - 0.3188 = 25.0934$
- This is also known as a 2-Factor Analysis of Variance (ANOVA).

Interpreting coefficients when a model has interactions

- If we include an interaction in our model for X_1 and X_2, then we cannot directly interpret the coefficients for X_1 and X_2. Their effects depend on each other.
- To get an idea of how they interact we can create an interaction plot or visualize the model using a package such as `ggeffects`.
- We usually want to find out if the interaction is significant. We can do this with the `anova` function.
- The `anova` function tells us whether or not the interaction appears to explain a significant amount of variation in our response.
- Judge the significance of an interaction based on the `anova` output, not the (possibly many) individual hypothesis tests in the linear model output.

Linear Modeling vs. ANOVA

- Two sides of the same coin.
  - Linear Model: Do levels of a categorical variable affect the response?
  - ANOVA: Does the mean response differ between levels of a categorical variable?
- Do ANOVA same way you fit linear model, except use `aov` instead of `lm`:

```
aov1 <- aov(Y ~ X1 + X2, data=mydata)
summary(aov)
TukeyHSD(aov1) # multilple comparisons of means
```

Non-linear effects

- Often the simple assumption of a linear effect of a predictor is unrealistic.
- the `ns()` function from the splines package allows us to fit non-linear effects, similar to a polynomial. ns stands for natural splines. Its second argument is the degrees of freedom. It may help to think of that as the number of times the smooth line changes directions.
- 3 to 5 degrees of freedom is almost always sufficient. Example:

```
sales_mod7 <-  lm(log(price) ~ ns(finsqft, df = 3) +
                        bedrooms + lotsize +
                        bathrooms + garagesize +
                        quality + highway,
                    data = sales)
```

Partial residual plots

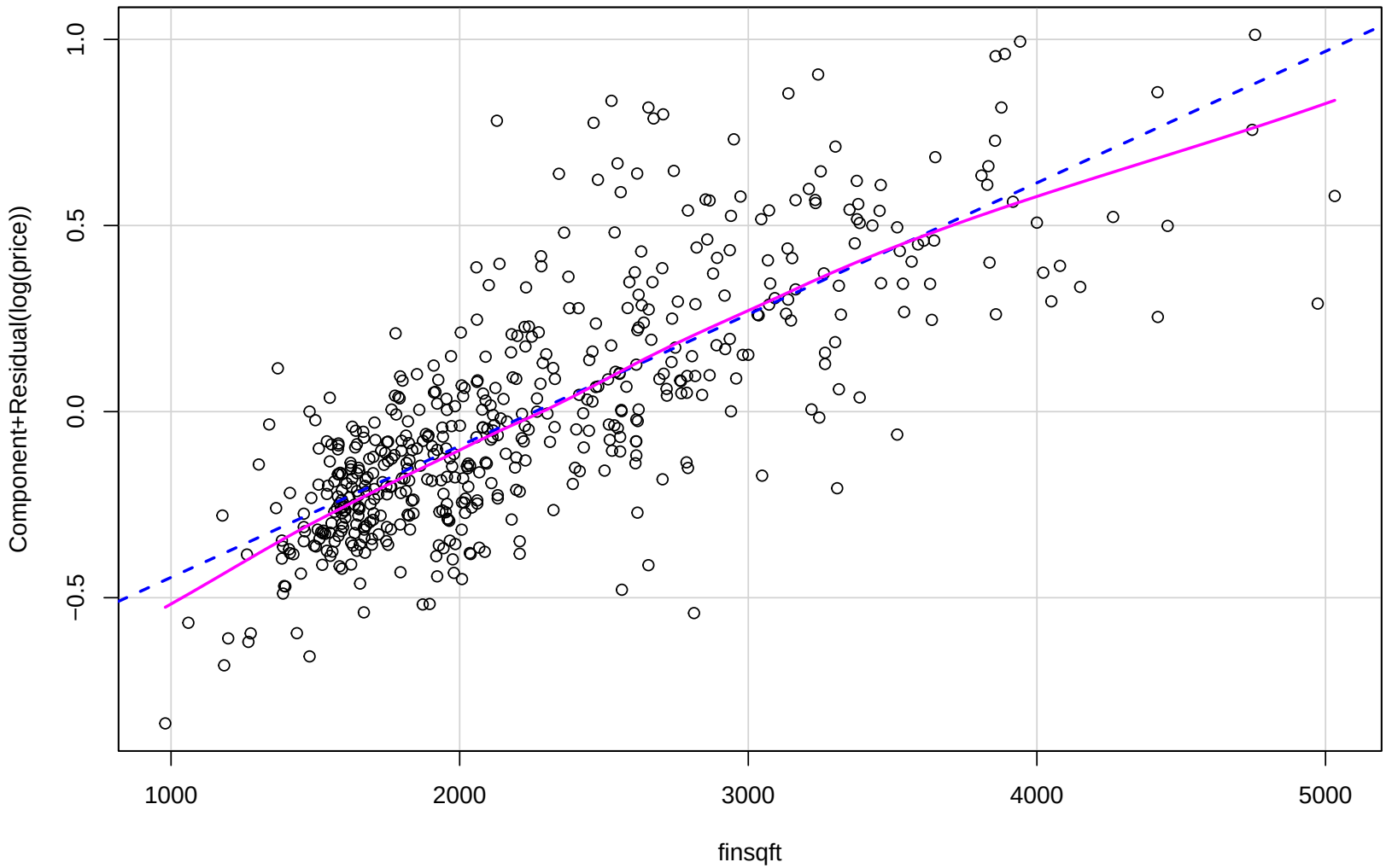
- Partial residual plots can help us determine if we need non-linear effects. Also called term plots and component-residual plots.
- The `crPlots()` function from the car package is very useful for this. Example:

```
crPlots(sales_mod6, terms = ~ finsqft + bedrooms +
                bathrooms + garagesize + lotsize)
```

- The blue dashed line is the fitted slope. The purple line is a smooth trend line. A curving purple line indicates a non-linear effect may be warranted.

Example of partial residual plot

```
crPlots(sales_mod6, terms = ~ finsqft)
```



Regression Diagnostics

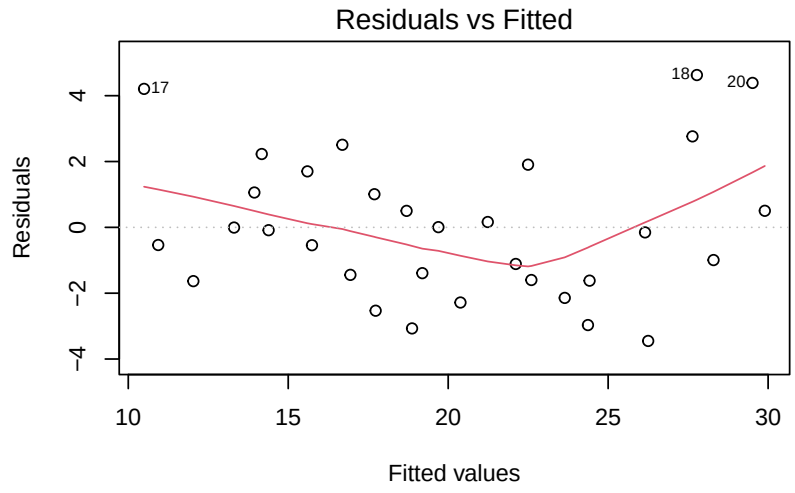
- Estimation and inference from a linear model depend on several assumptions.
- We check these assumptions using *regression diagnostics*. We assume...
  - errors are independent
  - errors have constant variance
  - errors are normally distributed
  - all observations "fit" the model and none have large influence on the model
- Violations of these assumptions can invalidate our model.

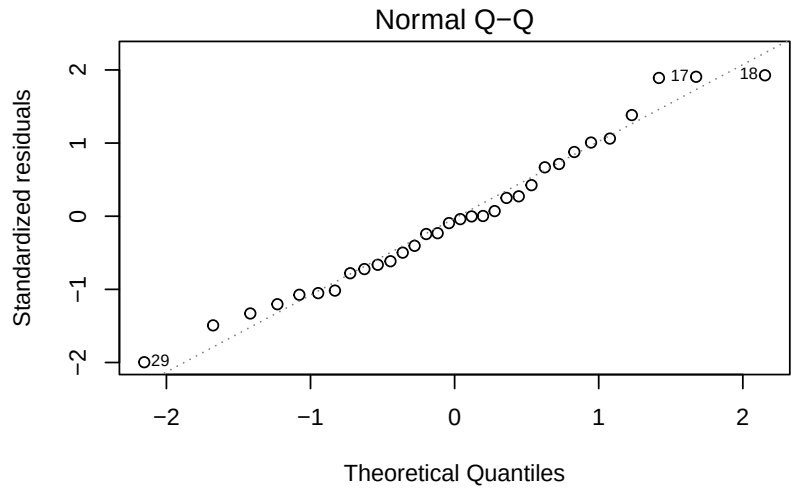
Quick visual diagnostics using `plot`

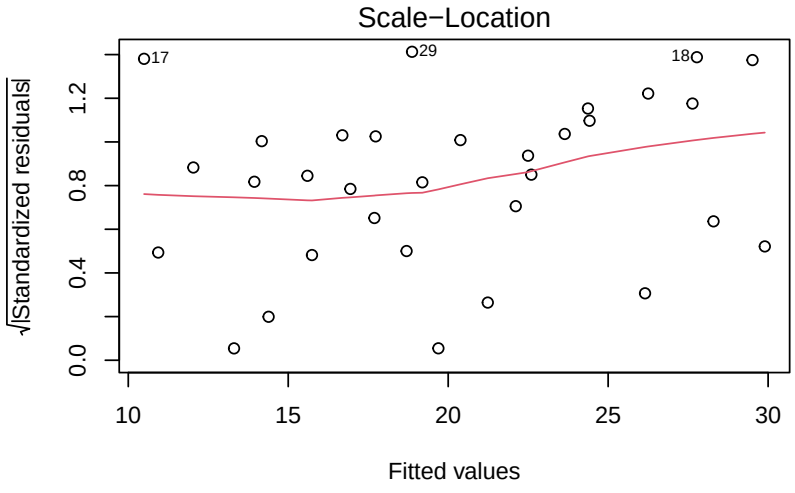
Calling the `plot` function on the model object produces four diagnostic plots.

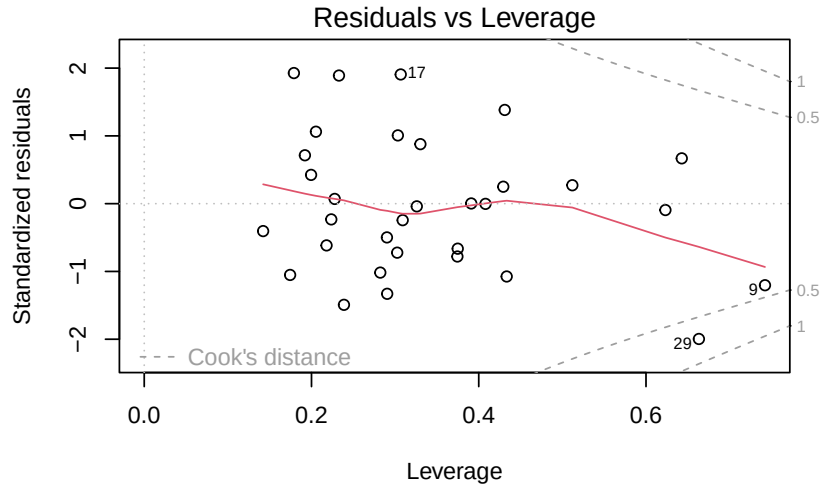
1. Residuals vs Fitted (*check constant variance assumption*)
2. Normal Q-Q (*check normality assumption*)
3. Scale-Location (*check constant variance assumption*)
4. Residuals vs Leverage (*check for influential observations*)

Example of plotting model object









How to interpret plots

1. Residuals vs Fitted: should have a horizontal line with uniform scatter of points
2. Normal Q-Q: points should lie close to diagonal line
3. Scale-Location: should have a horizontal line with uniform scatter of point; (*similar to #1 but easier to detect trend in dispersion*)
4. Residuals vs Leverage: points should lie *within* contour lines

Updating linear models

- After fitting a model, looking at coefficients and their standard errors, and examining diagnostics, we frequently need to update the model.
- This means removing or adding predictors and/or removing observations.
- Adding/removing predictors can be accomplished with the `update` function.
- Removing observations can be accomplished with the `subset=` argument in the `lm` function

Model selection

The task of adding or removing predictors is often referred to as *model selection* or *variable selection*.

Reasons for using a subset of predictors instead of all of them:

1. Simplicity. The simplest explanation is the best.
2. Better precision. Unnecessary predictors add noise.
3. Avoiding Collinearity. Can hide relationships.
4. Future efficiency. Save time and money not measuring redundant predictors

Two main types of model selection

1. Testing-based approach: compare successive models using hypothesis tests
2. Criterion-based approach: optimize a measure of goodness

Which to choose? That's up to you. Whatever you do, expect to do some experimentation and iteration to find better models.

Also expect to use a great deal of subjective judgment.

Testing-based approach

- Compare nested models with a *partial F-test* using the `anova` function. For example:

```
lm1 <- lm(y ~ x1 + x2 + x3 + x4)
lm2 <- update(lm1, . ~ . - x3 - x4)
anova(lm2, lm1)
```

- Null hypothesis: both models the same (smaller model fits just as well as bigger model).
- A low p-value says reject null; the larger model has more explanatory power.

Criterion-based approaches - AIC/BIC

- Two common criteria is the Akaike Information Criterion (AIC) and the Bayesian information criterion (BIC)
- These values estimate the out-of-sample accuracy if we were to use these models to make predictions on new data.
- Models with lower AIC and BIC are usually preferred.
- This approach requires *no hypothesis testing* unlike the testing-based procedure.
- Beware of "close finishes". Just because a model has a slightly lower AIC/BIC doesn't mean it's necessarily better. Would we get the same result with a new sample?

More R packages for modeling

- `effects`: visualizing model effects; similar to `ggeffects` but with a few more features
- `broom`: Convert statistical analysis objects into tidy data frames; great if you want to do further analysis or plotting of statistical results; works great with `dplyr` and `ggplot2`
- `lme4`: fit linear mixed-effect models
- `coefplot`: plot model coefficients; visualize coefficient magnitudes along with their standard errors

References

- Faraway, J. (2005). *Linear Models in R*. London: Chapman & Hall.
- Fox, J. (2002). *A R and S-Plus Companion to Applied Regression*. London: Sage.
- Harrell, F. (2015). *Regression Modeling Strategies* (2nd ed.). New York: Springer.
- Kutner, M., et al. (2005). *Applied Linear Statistical Models* (5th ed.). New York: McGraw-Hill.
- Maindonald J., Braun, J.W. (2010). *Data Analysis and Graphics Using R* (3rd ed.). Cambridge: Cambridge Univ Press.

StatLab

- Thanks for coming today!
- For help and advice with your data analysis, contact the StatLab to set up an appointment: statlab@virginia.edu
- Sign up for more workshops or see past workshops: http://data.library.virginia.edu/training/
- Register for the Research Data Services newsletter to stay up-to-date on StatLab events and resources: http://data.library.virginia.edu/newsletters/