Exploratory plotting and data analysis

Clara Cohen
Department of Linguistics
cpccohen@berkeley.edu
`http://linguistics.berkeley.edu/~cpccohen/`

D-lab Workshops, Spring 2015

1 Getting started

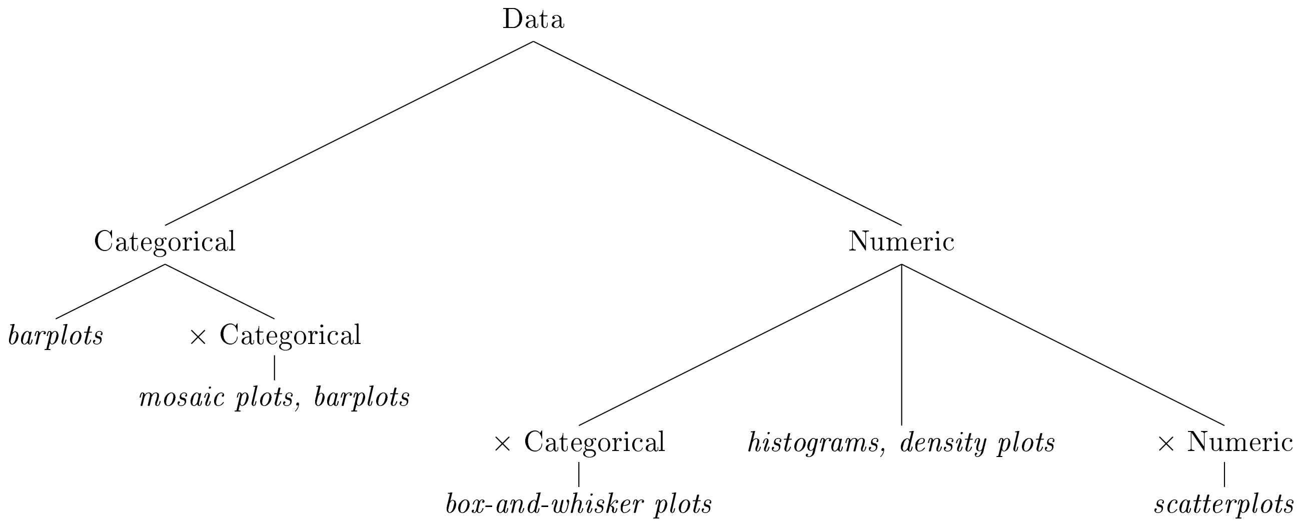
1.1 Prelims

Please load the files `admissions.csv`, `admissionsBig.csv`, `words.csv`, `ratings.csv`, and `states.csv`.

(Hint: use `object <- read.csv("filename.csv", header=TRUE)`)

Assumption You are familiar with the structure of dataframes, including how to extract subsets of a dataframe.

1.2 Data types



2 Summarizing data patterns

2.1 Finding properties of numerical data for different values of a factor

```
aggregate( <Outcome column>, list( <Factor 1> , <Factor 2 >, ...), <function> )
```
(*outputs a dataframe*)

```
tapply( <Outcome column>, list( <Factor 1> , <Factor 2 >, ...), <function> )
```
(*outputs a table*)

- "Dear R: Please find the mean frequency for all words that are animals, and all words that are plants":
  `aggregate( ratings$Frequency, list(ratings$Class), mean )`
- "Find the median length for all words that are complex, and all words that are simplex":
  `tapply( ratings$Length, list(ratings$Complex), median )`
- "Find the average numbers of days with frost for each state region":
  `tapply( states$Frost, list(states$state.region), mean )`

Practice:

- What is the mean Length of animal words and of plant words?
- What is the total area of each state region? What about for each state division?
- What is the mean reaction time for each age and word-valence?

2.2 Tabulating the number of observations of each category or combination of categories

`xtabs( ∼ <Factor 1> + <Factor 2> ...)`

- "Dear R: How many words are plants, and how many are animals?"
  `xtabs( ∼Class, data = ratings )`
- "What is the breakdown of observations for all combinations of Class and by Complexity?"
  `xtabs( ∼ Class + Complex, data = ratings )`
- "How many states have more than the mean value of murders?"
  `xtabs( ∼ assault > mean( assault ), data = states )` #Returns TRUE/FALSE counts
- "How many students of each gender were accepted or rejected?"
  `xtabs( ∼ admitBig$Gender + admitBig$Admit)`

Practice:

- How many states are in each state division?
- For each age group (in `words`), how many reaction times are faster than the overall median reaction time?
- How many students of each gender applied to each department?

3 Plots

3.1 Categorical (× categorical) data

3.1.1 Barplots

Barplots take matrices or tables. Usually these are extracted from your data via `xtabs()`, but you can also turn a dataframe into a table using `tapply()` The important thing is that all the values must be numeric.

Simple barplots:

From `xtabs()`	`dept.xtabs <- xtabs(~ Dept, data = admitBig)`
	`barplot(dept.xtabs)`
From dataframes	`dept.tab <- tapply(admit$Freq, list(admit$Dept), sum)`
	`# tapply()`: like `aggregate()`, but outputs a table
	`barplot(dept.tab)`

Practice:

1. Create a barplot of the number of men and women who applied to this school, using the large `admitBig` dataframe.
2. Create a barplot of the number of applicants who were admitted and rejected, using the condensed `admit` dataframe.

Multi-bar barplots:

Barplots from `xtabs()`	`genderDept.xtabs <- xtabs(~ Gender+Dept, data = admitBig)`
	`barplot(genderDept.xtabs)  #barplot()`: one numerical matrix/vector arg.
From dataframes	`genderDept.tab <- tapply(admit$Freq, list(admit$Gender, admit$Dept), sum)`
	`barplot(genderDept.tab)`
Adding things	
Side-by-side bars	`barplot(genderDept.tab, beside = TRUE)`
Axis labels	`barplot(genderDept.tab, xlab = "Department", ylab = "Number")`
Title	`barplot(GenderDept.tab, main = Applications by Dep't and gender")`
Color	`barplot(GenderDept.tab, col = c("blue", "green"))`
Axis limits	`barplot(genderDept.tab, ylim = c(0, 900))`
Legends	
The easy way	`barplot(genderDept.tab, legend.text = TRUE)`
The more flexible way	`barplot(genderDept.tab, col = c("blue", "green"))`
	`legend("topright", col = c("blue", "green"), legend = c("male", "female"), pch = 15)`

Practice:

1. Create a side-by-side barplot with non-standard colors (use `colors()`) to see a list of the available names), showing the number of applicants who were accepted and rejected in each department. Include a legend so that it is clear which color corresponds to which outcome.

3.1.2 Mosaic plots

Mosaic plots are are similar to barplots, except that each square is proportional to the total amount of data that it represents. Like barplots, they take a matrix or table input.

Mosaic plots	mosaicplot(genderDept.xtabs)
	mosaicplot(genderDept.tab)

Practice

1. Make a mosaic plot counterpart to one of the barplots you've already created. Feel free to play with axis labels, colors, and titles. Do you need a legend?

3.2 Numeric × categorical data

3.2.1 Boxplots

Boxplots are built from dataframes, by means of the following general syntax pattern:

```
<FunctionName> (<OutcomeVariable> ~ <PredictorVariable>, <DataFrame>)
```

Boxplots are better than barplots for graphing numerical distributions, because they show the shape of the distribution, rather than just one measure, such as mean or median.

Graphing means w/ barplots	`RegionIncome.tab <- tapply(states$Income, list(states$states.region), mean))` `barplot(RegionIncome.tab)`
Graphing w/ boxplots	`boxplot(Income ~ state.region, states)`

Practice:

1. Plot some boxplots of state properties by region.

3.3 Numeric data by itself

3.3.1 Histograms

Histograms show how observations are distributed across the range of possible values. They take one numeric vector of values, plus other optional arguments for customizability.

Default histogram	`hist(states$assault)`
Customization	
Number of bins	`hist(states$assault, breaks = seq(0, 350, by = 25))` #14 bins
Y-axis unit	`hist(states$assault, freq = FALSE)` #density rather than # of observations.
Labeling bars	`hist(states$assault, labels = TRUE)` #This is better when `freq = TRUE` (default)

Practice:

1. Find a state property with a very symmetrical distribution, and one with a very lopsided or bimodal distribution. Make a histogram of both properties. Play with the number of bins until the distribution is shown clearly, and label the bins.

3.3.2 Density plots

Density plots are essentially what happens when you have an infinite number of histogram bars, each representing the density of data that occurs at that value. The integral of the density curve is 1, in the same way that the area of each histogram bars (when the y-axis repesents density) sums to 1. Density curves are a great way of showing the shape of your data distribution, and are useful for revealing non-normality, skew, and bi-modality. The input to the `density` function is a numeric vector.

Plotting density	`plot(density(states$assault))`

Superimposing density	`hist(states$assault, freq = FALSE)` #`freq = FALSE` is crucial! `lines(density(states$assault))` # `lines()`: for overlaying curves

Superimposing a histogram	`plot(density(states$assault))`
	`hist(states$assault, freq = FALSE, add = TRUE)`

Practice

1. Superimpose the density curve over the state property you identified earlier that has a non-normal distribution. Then superimpose the histogram over the density. Play with axes and labels until you have a nice plot.

3.4 Numeric × numeric data

Scatterplots are created with the most flexible plotting function that R base graphics can offer: `plot()`. It takes a basic plotting formula (which you saw for boxplots), plus an enormous number of optional arguments. What I will show you is only a small subset of what is possible.

3.4.1 Scatterplots

Straightforward plotting	`plot(assault ~ murder, states)`
Changing axis ranges	`plot(..., ylim = c(40, 350), xlim = c(0, 21))`
Axis labels	`plot(..., xlab = "Murder rate", ylab = "Assault rate")`
Plotting character	`plot(..., pch = 3)` # To see the symbols, `plot(1:25, pch = 1:25)`
Labeling individual points	`plot(assault ~ murder, states)` `identify(states$assault ~ states$murder, labels = states$state.abb)` # Press `esc` when done
Labeling ALL points	`plot(assault ~ murder, states, type = "n")` # leave off points `text(assault ~ murder, states, labels = state.abb)`

Adding extra points	`plot(assault ~ murder, states, type = "n")`
	`points(assault ~ murder, states[states$state.region = = "West", ], col = "red")`

Practice:

1. Create a map of the states! Use `center.latitude` and `center.longitude` for your x- and y-values, and use the state abbreviation for your plotting character.
2. Do the same thing, but have a different color for each state region!
3. Plot the mean size rating of every word against its mean familiarity. What patterns do you observe?
4. Plot the mean size rating of every word against its mean familiarity, but do animal points in red and plant points in green. Include a legend.

4 Stats

4.1 Comparing two means: t-tests

T-tests tell you whether a particular sample's mean is probably different from some value — either a value that you give it, or, more frequently, the mean of another sample.

Is mean different from 0?	`t.test(states$Frost)` # By default, R compares the mean to 0
Is the mean different from x?	`t.test(states$HS.Grad, mu = 100)`
Are two means different?	`t.test(states$murder, states$Murder)`
One-sided t-test	`t.test(states$murder, states$assault, alternative = "less")`
	`t.test(states$assault, states$murder, alternative = "greater")`
Paired t-tests	`t.test(states$murder, states$Murder, paired=TRUE)`

Practice:

1. Are the two measures of state area (`state.area` and `Area`) significantly different?
2. In `ratings`, are animal terms more frequent than plant terms?

4.2 Comparing two or more means: ANOVAs

Anovas tell you whether there is reason to believe two or more samples come from the same distribution. It does NOT tell you anything about their distinctions. You can use two different function combinations to get it done for you: `summary(aov())` is best used for most ANOVAS, especially repeated measures. When all predictors are categorical variables, `anova(lm())` does exactly the same thing. It can also be used for more elaborate linear regression models.

Simple	`summary(aov(RT ~ valence, data = words))`
	`anova(lm(RT ~ valence, data = words))`
Two-way	`summary(aov(RT ~ subjAge + word, data = words))` #No interaction
	`summary(aov(RT ~ subjAge * word, data = words))` #Interaction!
Repeated measures	`summary(aov(RT ~ subjAge * valence + Error(subject/valence), data = words))`

Practice:

1. Perform a one-way anova to determine whether murder rates vary according to state region. If so, create a boxplot to see what the nature of that variation is.

Go here for a more elaborate discussion and example:
`https://personality-project.org/r/r.anova.html`

4.3 Comparing proportions: Chi-squared

Chi-squared tells you whether the proportion of observations distributed across various conditions are different or not. For example:

Are fewer women admitted than men?	`genderAdmit.xtabs <- xtabs(~ Gender+Admit, admitBig)`
	`prop.table(genderAdmit.xtabs, 1)` #Columns sum to 1
	`chisq.test(genderAdmit.xtabs)`

Are some departments more selective?	`deptAdmit.xtabs <- xtabs(~ Dept+Admit, admitBig)`
	`prop.table(deptAdmit.xtabs, 1)` #Rows sum to 1
	`chisq.test(deptAdmit.xtabs)`

Practice:

1. This data set is an excellent example of Simpson's Paradox, which shows the importance of independence of observations during chi-squared tests. On the surface, it seems that women are disproportionately rejected. However, recall that different departments have vastly differ-

ent selectivities. If men and women apply to different departments disproportionately, that could account for the apparent overall differences in acceptance rate. So, do they?

(In other words, perform a chi-squared test to determine whether men and women apply to the various departments at different rates. Create pretty barplots to illustrate the nature of any differences.)

4.4 Rates of change: Regression

4.4.1 Simple linear regression and lines of best fit

Simple linear regression answers the question "For every 1-unit increase in my numerical predictor, how much does the outcome numerical variable change?" This is generally used for showing best-fit lines.

Building the model	`mod.lm <- lm(murder ~ assault, states)`
Inspecting the model	`summary(mod.lm)`
Plotting the model	`plot(murder ~ assault, states)`
	`abline(mod.lm)` `# abline()`: for overlaying straight lines
Built-in model criticism	`plot(mod.lm)`

Practice:

1. Build a regression model for the data in `ratings`, predicting `meanSizeRating` from `meanFamiliarity`. Then plot the data with a regression line.
2. Build some regression models to determine whether and how different factors might affect life expectancy. Then plot one of the relations with a regression line.

4.4.2 Multiple regression

Multiple regression allows you to estimate the effect of each predictor, while taking into account the effect of other possiple predictors. When you have a numerical and categorical predictor, interactions are reasonably simple to plot and interpret.

Two numerical predictors	`mod2 <- lm(Life.Exp ~ assault + urbanPop, states)`
	`summary(mod2)`
Numerical and categorical	
No interaction term	`mod3 <- lm(meanSizeRating ~ meanFamiliarity + Class, ratings)`
With interaction term	`mod4 <- lm(meanSizeRating ~ meanFamiliarity * Class, ratings)`
Plotting it	`modS <- lm(meanSizeRating ~ meanFamiliarity, ratings[Class = = "plant", ])`
	`modN <- lm(meanSizeRating ~ meanFamiliarity, ratings[Class ! = "plant", ])`
	`plot(meanSizeRating ~ meanFamiliarity, ratings, type = "n")`
	`points(meanSizeRating ~ meanFamiliarity, ratings[ratings$Class = = "plant", ], col = "green", pch = 16)`
	`abline(modS, col = "green")`
	`points(meanSizeRating ~ meanFamiliarity, ratings[state$Class ! = "plant", ], col = "red", pch = 16)`

	`abline(modN, col = "red")`

Practice:

1. Determine the effect of `urbanPop` and `state.region` on `Life.Exp` in `states`.
2. Plot the interaction between `urbanPop` and `state.region`, along with best-fit lines. This will take at least 9 lines of code. You can start with `par(mfrow = c(2, 2))` to use a two-by-two plotting space, or overlay everything with `points()`. This will also illustrate the utility of more advanced graphics, which you will learn Friday!

4.4.3 Logistic regression

Logistic regression is one way of determining how various predictors affect the probability of a particular binary outcome. You use the command `glm()`. The syntax is exactly like `lm()`, except that you must also add the argument `family = "binomial"`.

Note: by default, the first level of the outcome variable is considered "failure, " and all subsequent levels are considered "success." So the model is predicting the probability (well, the log-odds) of observing any outcome other than the first value.

One categorical predictor	`glm1 <- glm(Admit ~ Gender, admitBig, family = "binomial")`
	`summary(glm1)`
Categorical and numeric	`glm2 <- glm(Admit ~ Gender * GPA, admitBig, family = "binomial")`
	`summary(glm2)`

Practice:

1. What is the effect of `Dept` on the probability of admission?
2. Does `Gender` interact with `Dept`? If so, how?

4.5 Plotting model coefficients

To plot the effects estimated by your regression model, download and install the `effects` package. This can be loaded for every subsequent use simply by using `library(effects)`.

Build your model	`mod <- lm(Life.Exp ~ murder + urbanPop * state.region, data = states)`
Plot the effects	`plot(allEffects(mod))`
For just one a subset of effect	`plot(Effect(c("state.region", "murder"), mod))`