R By Example::Cheat Sheet

Source Code

General conventions: functions as `function()`, packages as `{package}` and objects as `object`

Data

General Workflow

1. Initialize a Project.
2. Open RMarkdown File and save in project directory.
3. Load packages you'll need in first chunk.
   - e.g., `{tidyverse}`, `{rio}`, `{ggeffects}`, `{car}`
4. Write prose and R code in RMarkdown file.

Add-on Packages

- `install.packages(<package name>)` downloads the package from the web to your computer - only need to do once per R version.
- `library(<package name>)` makes the package functions available for your R session - need to do once per R session.

Importing Data

Use the `{rio}` package to import data from Stata, SPSS, etc…:

- `dat <- import(<path>)` where `<path>` is the path to a file.
- `export(dat, <filename>)` will export the data to another proprietary format (Stata, SPSS, etc…).

Data Types

- Numeric - numbers without labels
- Factors - categorical variables - numbers with labels.
  - `factor()` and `as.factor()` can turn numeric and character data into factors.
  - `factorize()` from the `{rio}` package will take labels read in with the data from Stata or SPSS and apply them to the variables.
- Strings - character strings with no attached numerical information.

Investigating Properties of Data For a dataset named `data` and a variable `x`.

- `str(data)` – shows the properties of all the variables in `data`.
- `str(data$x)` – shows the properties of the `x` variable in the `data`.
- `names(data)` – shows the names of all the variables in `data`.
- `head(data)` – shows the first six rows of all the variables in `data`.
- `tail(data)` – shows the last six rows of all the variables in `data`.

`{dplyr}` Operations

- `%>%` pipe chains functions together.
- `select()` chooses columns based on logical expression, name or number.
- `filter()` chooses observations based on logical expression.
- `group_by()` groups observations so operations are performed within groups.
- `mutate()` adds or modifies variables to existing dataset.
- `summarise()` collapses/aggregates across observations (within group).
  - common operations: `first()`, `last()`, `mean()`, `sd()`, `median()`, `min()`, `max()`, `quantile()`, `n()`

Data, Viz and Linear Models

Recoding Variables

- Using the `recode()` function from the `{car}` package:
  - `recode(x, <recode statements>)`
  - `lo` and `hi` stand in for the minimum and maximum of the variable, `else` is short for all other values not explicitly mentioned.
  - Example: `college_ed = recode(educ, "lo:12='None'; 13:15='Some'; 16:hi='Degree'", as.factor=TRUE, levels=c("None", "Some", "Degree"))`
- Using `case_when()` from the `{dplyr}` package.
  - Example: `college_ed = case_when(educ <= 12 ~ 'None', educ > 12 & educ < 16 ~ 'Some', educ >=16 ~ 'Degree', TRUE ~ NA_character_)`

Data Viz

In the `{ggplot2}` framework, data can be mapped to the following aesthetics - `shape`, `colour`, `fill`, `alpha`, `linetype`, `size` and the properties controlled with the `scale_*_manual()` functions.

- `geom_bar()` to make bar plots, use `stat="identity"` if you are supplying the height of the bar with `y` and `position=position_dodge()` for side-by-side bar plots.
- `geom_line()` for lines connecting points.
- `geom_histogram()` for histograms, use `position="identity"` for overlapping (superposed) histograms.
- `geom_tile()` for heatmap plots, you can add the text onto the tiles with `geom_text()`.
- `geom_mosiac()` from the `{ggmosaic}` package for making mosaic plots.

Other useful functions:

- `coord_flip()` to swap `x` and `y` axes.
- `facet_wrap(~group)` to make different panels for each value of `group`.
- `theme(axis.text.x=element_text(rot=45, hjust=1))` to rotate `x`-axis labels 45 degrees.

Model Formulas

- Generally take the form of `<dependent variable> ~ <independent variables>`
  - `x1 + x2` - additive relationship.
  - `x1 * x2` - multiplicative including main effects.
  - `x1:x2` - multiplicative excluding main effects.
  - `log(x)` - natural logarithm of `x`.
  - `I(x^p)` - raise `x` to the `p` power.
  - `poly(x, p)` - include a `p` degree polynomial in `x`.

(Generalized) Linear Models

Linear Models

- Estimate with `model <- lm(formula, data=data)`
- Presentation
  - `stargazer(model)` from the `{stargazer}` package to make publication-ready tables.
  - `ggpredict(model)` from the `{ggeffects}` package for making predicted effects plots.
  - `intQualQuant()` from the `{DAMisc}` package for categorical-continuous interactions.
    - `type='slopes'` with `plot=FALSE` prints the simple slopes.
  - `DAintfun2()` and `DAintfun()` from the `{DAMisc}` package for continuous-continuous interaction conditional effect graphs.
    - `changeSig()` to identify changes in the significance of conditional effects.
- Diagnostics
  - Linearity/Functional Form `crPlots(model)` from the `{car}` package.
  - Heteroskedasticity - `ncvTest(model)` from the `{car}` package or residual vs fitted plot.
  - Normality of Residuals - `shaprio.test(model$residuals)` or density plot of residuals.
  - Outliers and Influential Observations - `InfluencePlot(model)` from the `{car}` package.
    - Leverage - `hatvalues(model)`
    - Discrepancy - `rstudent(model)` or `rstandard(model)`
    - Cook's D - `cooks.distance(model)`
    - DFBETAS - `dfbetas(model)`

Generalized Linear Models

- Estimate with `glm(formula, data, family)`
- Model fit
  - `binfit()` from the `{DAMisc}` package for pseudo-R^2 measures.
  - `pre()` from the `{DAMisc}` package to calculate the proportional reduction in error.
  - `AIC()` and `BIC()` in base R to calculate information criterion measures.
  - `logLik()` to display the log-likelihood.
  - `lrtest()` from the `{lmtest}` or `anova()` from base R to test nested models.
  - `clarke_test()` from the `{clarkeTest}` package to test non-nested models.

General conventions: functions as `function()`, packages as `{package}` and objects as `object`

GLMs and Categorical Models

Generalized Linear Models

- Calculating Effects.
  - Odds Ratios for Logit: `exp(code(model))`
  - First difference at reasonable values - `glmChange()` from the `{DAMisc}` package.
    - Find "central" values of all values except x_j
    - Calculate predicted pobabilities for $x_j = x_0 + \delta$ and $x_j = x_0$
    - Subtract the latter from the former
  - Average first difference - `glmChage2()` from the `{DAMisc}` package.
    - Calculate the predicted probabilities for all observations.
    - Calculate the predicted probabilities for all observations setting $x_j = x_j + \delta$
    - Subtract the former from the latter to get the individual first differences and take the average.
  - Marginal Effect - `margins()` from the `{margins}` package.
    - Calculate the partial first difference of the predicted probabilities w.r.t. x_j for each observation.
    - Average across all of the marginal effects.
- Plotting Effects
  - At Reasonable Values - `ggpredict()` from the `{ggeffects}` package
    - Hold all variables except x_j at central values.
    - Calculate predicted probabilities at $x_j = \{min(x_j), \ldots, max(x_j)\}$
    - Plot predicted probabilities on y against $\{min(x_j), \ldots, max(x_j)\}$ on x.
  - Average Effect - `aveEffPlot()` from the `{DAMisc}` package.
    - Calculate predicted probability for every observation at each value of $x_j = \{min(x_j), \ldots, max(x_j)\}$
    - For each value of $x_j = \{min(x_j), \ldots, max(x_j)\}$, calculate the average predicted probability.
    - Plot the average predicted probabilities against $\{min(x_j), \ldots, max(x_j)\}$.

GLM Interactions

- Two separable questions (that are inseperable in the linear model).
  1. Is the product term needed?
     - Evaluate with statistical significance of interaction term(s).
  2. Is there an interaction?
     - Calculate the second difference/derivative
     - `seconDiff()` from `{DAMisc}` calculates this.
     - For a cross-partial derivative (difference in first derivatives), you could do it by hand.

Categorical, Survey and Multilevel Models

Ordinal Models

- Estimate with `clm()` fromthe `{ordinal}` package or `polr()` from the `{MASS}` package.
- `ordfit()` and `pre()` from the `{DAMisc}` package, `AIC()` and `BIC()` from base R as well as `lrtest()` from `{lmtest}` and `clarke_test()` from `{clarkeTest}` all work.
- `ordChange()` and `ordChange2()` from `{DAMisc}` calculate first differences using the 'reasonable values' and 'observed values' approaches, respectively.
- `ggpredict()` from `{ggeffects}` and `ordAveEffPlot()` from `{DAMisc}` plot effects for the 'reasonable values' and 'observed values' approaches, respectively.
- `nominal_test()` from `{ordinal}` for `clm()` objects and `poTest()` from `{car}` for `polr()` objects do the Brant test.

Multinomial Models

- Estimate with `multinom()` from the `{nnet}` package or `mlogit()` from the `{mlogit}` package.
  - `mlogit()` also allows estimation of the conditional logit model.
- `mnlgit()` and `pre()` from the `{DAMisc}` package, `AIC()` and `BIC()` from base R as well as `lrtest()` from `{lmtest}` and `clarke_test()` from `{clarkeTest}` all work.
- `mnlChange()` and `mnlChange2()` from `{DAMisc}` calculate first differences using the 'reasonable values' and 'observed values' approaches, respectively.
- `ggpredict()` from `{ggeffects}` and `mnlAveEffPlot()` from `{DAMisc}` plot effects for the 'reasonable values' and 'observed values' approaches, respectively.

Complex Survey Data

- You can set the survey design properties with `svydesign()` from the `{survey}` package or `as_survey_design()` from the `{srvyr}` package.
  - Use `as_survey()` from `{srvyr}` to convert an object created with the `{survey}` package into a 'survey tibble' to be used with `{dplyr}` and other `{srvyr}` functions.
- `sumStats()` from the `{DAMisc}` package makes summary statistics (optionally by the values of other variables),
- `xt()` from `{DAMisc}` makes weighted cross-tabulations with measure of fit for survey data.
- `svyglm()` and `svyolr()` from `{survey}` will estimate GLM and ordinal models. `svymle()` will estimate any MLE model for complex survey data.
- `ggpredict()` from `{ggeffects}` will generate predicted probabilities as will `glmChange2()` and `probci()` from the `{DAMisc}` package.
  - `glmChange2()` and `probci()` use unweighted averages of individual first differences.

Multilevel and Measurement Models

Multilevel Models

- Estimated with `lmer()` or `glmer()` from `{lme4}` for (G)LMs. For ordinal models, use `clmm()` from `{ordinal}` and for multinomial models us `mlogit()` from `{mlogit}`.
  - For `lmer()` and `glmer()`, including in the formula `(1 | group)` will make random intercepts by `group` and `(1 + l2var | group)` will make random intercepts and random slopes for `l2var` by `group`.
- `coef()` will provide `group`-level intercepts and coefficients from the model.
- `ggpredict()` from `{ggeffects}` will graph either fixed overall effects or group-level effects - with confidence intervals that include the fixed-effect (and optionally random-effect) variances.
- `brm()` from `{brms}` will estimate a Bayesian version of the model using Stan.
  - NOTE: this could take a long time depending on the size of the model/data.

PCA/Exploratory FA

- `princomp()` from base R does principal components analysis.
- `ggbiplot()` from the `{ggbiplot}` package makes a biplot for the PCA.
- `fa()` from the `{psych}` package does EFA with many options.
  - `rotate=<rotation>` will rotate the solution see `?fa` for options.
- `scree()` from `{psych}` makes a scree plot for evaluating dimensionality.

CFA, SEM and Growth Models

- Estimated with `cfa()`, `sem()` or `growth()` from the `{lavaan}` package with primary operators:
  - `lv =~ ind1 + ind2` specifies that `ind1` and `ind2` are indicators of latent variable `lv`.
  - `dv ~ iv1 + iv2` specifies a regression of `dv` on `iv1` and `iv2`
  - `ac := a*c` specifies that the computed value `ac` is the product of `a` and `c`.
  - `ind1 ~~ ind2` indicates that the covariance between `ind1` and `ind2` should be free and estimated by the model.
- `modificationIndices()` print the modification indices for `{lavaan}` models.
- `summary(model)` summarizes the model object with the following arguments:
  - `standardize=TRUE` will give the standardized solution.
  - `fit.measures=TRUE` give more scalar measures of fit for the model.
- `predict(model)` predicts the latent variables for the observed dataset used in the model.