Summer 2015 Python Notes (copied from MoPad)
NOTE: These are rough notes taken in/by Python Group. Use with caution as they may contain errors! If you find an error (or have a better solution), we would love for you to share it with us!

This pad text is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents!

Please be cognizant of whether you are using a public pad or private/team pad, and take appropriate precautions with data you post here!

This is an open forum to type your Cyclismo tutorial notes in! Feel free to type as much/as little as you find useful :)

06/08/15

Chapter 1: Input http://www.cyclismo.org/tutorial/R/input.html
Open a .csv file and assign the contents to variable "var" (comma-separated file format)
var = read.csv("filename.txt", head = T)
var = read.csv(file.choose(), head = T)
#head denotes that the first row are header names
#filename is the file you wish to open; it must be in the current working directory
#"file.choose()" opens a window and allows you to select a file

read.table #same as read.csv, but for a tab-delimited file
getwd() #returns the current working directory
setwd() #sets the working directory
dir() #lists the contents of the working directory

Chapter 2: http://www.cyclismo.org/tutorial/R/types.html#
Variable types:
numbers,
strings, factors, data frames,logical

Chapter3: http://www.cyclismo.org/tutorial/R/basicOps.html
Doing math between vectors requires the vectors to have the same number of elements
names() #gives the names of columns
table$name #gives the items in the column

# mean, median, quantiles, minimum, maximum, variance, standard deviation, ~all:
mean()
median()
quantile()
min()
max()
var()
sd()

summary()

# sort increasing numerical or alpha order [,decreasing=TRUE]
sort()

dataFrame <- dataFrame[order(dataFrame$column2),]
# This takes the data in column2, puts it in numerical order, sorting the entire data
# frame, and putting it back into dataFrame

A similar method to reorganize your data:
> dataFrame$Names <- with(dataFrame, reorder(Names, column2))

If your data frame has row numbers and you want to get rid of them:
row.names(dataFrame) <- dataFrame$Names
Replaces the name for the data in each row with the info stored in the column
dataFrame$Names

dataFrame <- dataFrame(,2:ncol(dataFrame))
# Removes the old column with useless numbers

What we want to learn in R:
Plots: (ggplot2) # see http://ggplot2.org/resources/2007-vanderbilt.pdf
boxplot
scatterplots (for large datasets)
heatmaps
plot manipulations

Tests:
Check distribution (Shapiro-Wilks, qqplot)
ANOVA
Tukey
test for outliers
nonparametric tests (Kruskal Wallis)lim
clustering

Big Data Analysis:
Bioconductor
- DESeq2/EdgeR
- microarray analysis

Regression:
linear regression
multiple linear regression

modeling

Limitations of each test

*TO DO for Monday, 15 June 2015:
1. Install ggplot2 package
   -install.packages("ggplot2")
   -library(ggplot2) to run it
   ->see link above for more info on ggplot
2. Familiarize yourself with mtcars dataset (type "mtcars"; automatically stored as a variable in R)

To see which data sets are available to you: Type data() into R to view all different sets

Heatmaps
dataF <- data.frame(read.csv("myFile", head=TRUE, sep=","))
x <- as.matrix(dataF)
heatmap(x) # x is a matrix of values
# load package "gplots"
heatmap.2(x)
# using matrix, see http://sebastianraschka.com/Articles/heatmaps_in_r.html

from a data frame(ggplot2) https://learnr.wordpress.com/2010/01/26/ggplot2-quick-heatmap-plotting/

Kmeans clustering
myDF # a dataframe with your data
myMatrix <- as.matrix(myDF) #only include those columns/rows with numbers (no text)
kmeans20 <- kmeans(myMatrix, 20, nstart=200)
  # 20 is the number of centers you want to end up with,
  # try many different numbers (ie 4, 10, 66)
  # nstart is the number of times it should attempt the clustering more is usually better
# simple visualization for a matrix with 2 columns
plot(myMatrix, col = kmeans20$cluster)
points(kmeans20$centers, col = 1:20, pch = 8)

adding a column to a matrix for personalized heatmap generation
library("gplots")
myclusterMatrix <- transform(myMatrix, cluster=kmeans20$cluster) #add the column
myclusterDf <- myclusterMatrix[do.call(order, as.data.frame(myclusterMatrix$cluster)),]
    #sort based on the cluster number
myorderedMatrix <- as.matrix(myclusterDf) #turn it back into a matrix
heatmap.2(myorderedMatrix[,1:2], Rowv=NA, Colv=NA, scale="none", trace="none", density.info="none")
  # omit the cluster column(3)

remove the sorting of rows and the dendogram
remove the sorting of columns and the dendogram
use the actual numbers given(assuming you already have normalized data)
remove the default blue tracing
remove the blue tracing from the key

Scatter plot, and subsequent graph manipulations: GOOD LUCK!
Using the mtcars dataset in R, we can plot mpg vs. disp (displacement) as follows:
(I'm actually including spaces more legibility; text to type in R is italicized)

plot(mpg ~ disp, data = mtcars)

Alternatively, if the mtcars dataset is attached:

attach(mtcars)
plot(mpg ~ disp)

The following are commands to manipulate graphs:

bty = "n" #box is not drawn around the plot
frame.plot = F #box is not drawn around the plot

Titles:
xlab = "X-axis title"
ylab = "Y-axis title"
main = "Graph title"
sub = "Subtitle"
las = 2 #axis labels are perpendicular to axis

Change axis range (both options are vectors of two values that define the min and max):
xlim = c(0,500)
ylim = c(0,40)
axes = F #axes are not plotted (allows custom axes)

Add custom axis:
axis(side, at = c(a1, a2, a3, …), labels = c("l1", "l2", "l3", …), pos = num)
#where "side" defines side of the plot to draw the axis (1=below, 2=left, 3=above, 4=right), "at" defines points at which tick marks are to be drawn, "labels" defines the labels for the tick marks, "pos" is the coordinate at which the axis line is to be drawn

Change plotting/line styles (size change is by defined factor):
col = "tomato" #change points/line colour; to list the colours available in R, type "colors()"
pch = 18 #specify a symbol 0-25, or a character, e.g. "A"
cex = 2 #point size
lty = 2 #line type (1-6)

lwd = 2 #line width

Title/label sizes (change is by defined factor):
cex.axis = 0.8 #axis annotation
cex.label = 1.5 #axis label
cex.main = 2 #graph title
cex.sub = 1.2 #subtitle

Title/label attributes:
- for defining "font" 1=plain, 2=bold, 3=italic, 4=bold italic
- for defining "family", values are "serif", "sans", "mono"
font.axis = 3 #italicize axis annotations
font.lab = 4 #bold and italicize axis labels
font.main = 1 #plain graph title
font.sub = 2 #bold subtitle
family = "serif"
col.axis = "navy" #change colours of axes
col.lab = "peru" #change colours of axis labels
col.main = "gold" #change title colour
col.sub = "orchid" #change subtitle colour

Put everything together.....
plot(mpg ~ disp, main="MPG vs. Displacement", sub = "Gas mileage decreases with increased displacement", xlab = "Displacement", ylab = "MPG", col.main="gold", col.sub="orchid", col.axis="navy", col.lab="peru", pch=18, cex=1.5, col="tomato", xlim=c(0,500), ylim=c(0,40), cex.axis=0.8, cex.lab=1.5, cex.main=2, cex.sub=1.2, font.axis=3, font.lab=4, font.main=1, font.sub=2, family="serif")

Admittedly, this is an extremely long script to type for each graph, but we can store graph preferences using par() as follows:
par() #view current settings (notice that there are many, many, many more settings...)
opar = par() #make a copy of current settings
par(opar) #return to original settings

#Note:
"par" is also used if you want to create a figure composed of multiple graphs -
par(mfrow=c(3,4)) #defines a graph space of 4 plots wide x 3 plots high

Define plotting preferences:
par(col.main="gold", col.sub="orchid", col.axis="navy", col.lab="peru", pch=18, cex=1.5, col="tomato", cex.axis=0.8, cex.lab=1.5, cex.main=2, cex.sub=1.2, font.axis=3, font.lab=4, font.main=1, font.sub=2, family="serif")
These preferences will be used until the graph window is closed

Plot with these preferences (much shorter script):

plot(mpg ~ disp, main="MPG vs. Displacement", xlab = "Displacement", ylab = "MPG", xlim=c(0,500), ylim=c(0,40))

Add a line of best fit (uses the "lm" function for linear regression):
abline(lm(mpg ~ disp), col = "purple", lty = 2, lwd = 2)
#where lty is line type (1-6) and lwd is line width (factor to change)

Add a second series to the graph (obviously the x-axis label no longer applies):
points(hp, mpg, pch=16, col = "tomato4")
#points adds new data to an open plot with the syntax "points(x, y)"

Add a legend:
legend("topright", inset = 0.05, title = "Legend", c("Displacement","Horsepower"), col=c("tomato","tomato4"), pch = c(18,16))
#where topright defines the location on the plot and inset defines how far to inset the legend from the plot border; location can also be defined as an x,y coordinate on the plot

Add additional graphs to plot (ensure axis range is large enough to accommodate extra graphs):
Use "add=T, at=y" as options for additional graphs, where "y" defines the level on the y-axis to add the graph

Add text to the plot:
text(x, y, labels) #specifies x-y coordinates to add text defined by "labels"; normally characters outside the plot area will be ignored, but can be overridden with the "xpd=T" option

BONUS -- Barplot with error bars (because this is not quite trivial):
First define some means and standard deviations to plot...
- parse "cyl", "wt" and "gear" columns from mtcars (these have a similar range of numbers)
data = data.frame(mtcars$cyl, mtcars$wt, mtcars$gear)

Determine means and standard deviations for this new dataset
- use "apply" function to do this quickly
- the apply function is very versatile, as it applies a function across multiple dimensions in a dataframe
- apply(data, dimension, function), where "dimension" is the direction over which the function will be applied (1 = across rows, 2 = across columns)

means = apply(data, 2, mean)
sds = apply(data, 2, sd)

Use the "barplot" function, but must define barplot as a variable
*b.plot = barplot(means, ylim = c(0, max(means)+(max(sds)*2)))* #increase range of y-axis with ylim to accommodate error bars
arrows(b.plot, means+sds, b.plot, means, angle=90, code=1, length=0.05) #add +error bars
arrows(b.plot, means-sds, b.plot, means, angle=90, code=1, length=0.05) #add -error bars

Plotting discrete data:
- plotting discrete data is difficult because values will overlap
plot(x, y) #overlap of discrete x and y values makes plot useless
mosaicplot(table(x, y)) #mosaic plot of a count table of the data
smoothScatter(x, y) #density plot better describes data
grid(a, b) #add gridlines; a vertical lines x b horizontal lines

##Using ggplot2 http://www.statmethods.net/advgraphs/ggplot2.html
library(ggplot2) # Should provide access to all of ggplot2

qplot() # Basic plotting, guesses the plot based on the data given
geom = ["boxplot", "path", "jitter", "dotplot", "density"...] # to manually set the type of graph
  see http://docs.ggplot2.org/current/
facets= data # breaks up the graph based on the factors in data, multiple data (ie. data1 ~ data2) has a tiling effect
size = # sets the size of the points
colour = # sets the color of the points

examples
qplot(mpg, wt, data=mtcars, colour=factor(cyl))
qplot(mpg, disp, data=mtcars, size=wt)

create factors with value labels
mtcars$gear <- factor(mtcars$gear,levels=c(3,4,5), labels=c("3gears","4gears","5gears"))
mtcars$am <- factor(mtcars$am,levels=c(0,1), labels=c("Automatic","Manual"))
mtcars$cyl <- factor(mtcars$cyl,levels=c(4,6,8), labels=c("4cyl","6cyl","8cyl"))

Separate regressions of mpg on weight for each number of cylinders
qplot(wt, mpg, data=mtcars, geom=c("point", "smooth"), method="lm", formula=y~x, color=cyl, main="Regression of MPG on Weight", xlab="Weight", ylab="Miles per Gallon")

To make graphs that have a similar feel use theme() and add it to your graphs
http://docs.ggplot2.org/current/theme.html
White background and black grid lines
qplot() + *theme_bw()*

Large brown bold italics labels
and legend placed at top of plot
qplot + *theme(axis.title=element_text(face="bold.italic", size="12", color="brown"), legend.position="top")*

A large portion of the population is colorblind, use these to avoid confusion
http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/
The palette with grey:

cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
The palette with black:
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

library(scales) # allows use of alpha
*ggplot(mtcars, aes(mpg, wt*100)) + geom_density2d(aes(colour=factor(gear)),size=1) + xlim(0,40) + ylim(0,600) + scale_colour_manual(values = alpha(cbPalette,I(2/5))) + theme_bw() + guides(colour = guide_legend(override.aes = list(alpha = 1)))*
use the mtcars dataset
plot x=mpg and y=weight, adjust weight values
make a density plot, Break up the data based on gears, line width = 1
set the x-axis limits 0 - 40
set the y-axis limits 0 - 600
scale the colors to the colorblind palette, and make them not as vivid for overlaps
change the background to the default black and white theme
make the legend appear readable

to plot:expression clusters
bethany <- ggplot(rnaseq_matrix_pearson, aes(treatment ,expression))
bethany + geom_line(aes(colour=cluster))

X23_mock =
as.data.frame(rep("X23_mock",12936),rnaseq_matrix_pearson$X23_Mock,rnaseq_matrix_pearson$cluster)

Generation of random distributions:
R can generate data from pre-installed distributions. For every distribution there are four commands, which are a function prefix with that distribution:
"d" returns the height of the probability density function
"p" returns the cumulative density function
"q" returns the inverse cumulative density function (quantiles)
"r" returns randomly generated numbers
For a normal distribution: dnorm, pnorm, qnorm, rnorm
Define size, mean, sd; e.g. rnorm(50, 5, 2) will randomly generate 50 values of mean=5 and sd=2 from a normal distribution.

Normality assessment:
Common tests for normal distribution are the Shapiro-Wilk test and Anderson-Darling test. Both test the null hypothesis that data are distributed normally; if the P-value is below the cutoff, normality should be rejected.
Shapiro-Wilk test:

shapiro.test(x)
Anderson-Darling test:
Load “nortest” package
ad.test(x) #requires n > 7

To test normality on a subset of one column of data (Ex. mpg for only 4 cyl cars), use data subset:
> shapiro.test(subset(mtcars, mtcars$cyl == "4")$mpg)

These tests test the null hypothesis that a sample came from a normally distributed population; even if the data fails the normality test, it does not mean the population is non-normal. It is just as helpful to visualize the distribution of your data.
The q-q plot is a plot of the sample quantiles against the theoretical quantiles and is a nice method to visualize whether the data deviates from normality.
qqnorm(x)
qqline(x) #draws a line through the first and third quartiles; normally distributed data should follow this line

Exercises
1. Test normality of “mpg”, “gear”, “disp” in mtcars with a Shapiro test. e.g.
shapiro.test(mtcars$mpg)
2. Use a q-q plot to visualize the distribution of these data.
3. Generate random normal distributions with statistics from a variable and test for comparison with a shapiro.test and visualize with qqplot.
- use length (n), mean, and sd functions
Example:
shapiro.test(a <- rnorm(length(mtcars$disp), mean(mtcars$disp), sd(mtcars$disp))
qqplot(a)

Statistical tests

Parametric tests (i.e. assumption of normality)
t-test:
t.test(y, mu=3) #one-sample t-test; test null hypothesis that mean(y) = mu

Example 1. Is the mean of mpg (mtcars) different from 20? 23?

t.test(y1,y2) #two-sample t-test where y1 and y2 are independent groups of numeric data
#options:
#paired=T paired t-test
#var.equal=T equal sample variance

Example 2. Is the mpg of 4-cylinder cars different from 8-cylinder cars? (Subset 4- and 8-cylinder data)

```
cyl4=subset(mtcars,mtcars$cyl=="4")$mpg
cyl8=subset(mtcars,mtcars$cyl=="8")$mpg
t.test(cyl4,cyl8)
```

The output from these tests is not conducive for export. To parse out the p-value:

```
t.test(y1,y2)$p.value
#Use attribute(t.test(y1,y2)) to identify the output of interest
```

T-test is fine for comparing two groups, but does not work for examining the effect of a factor with multiple levels.

One-Way ANOVA:

```
aov(y~x)        #returns SS, df, residuals
summary(aov(y~x))        #returns summary statistics, including P-value
```

Example 3. Do the number of engine cylinders have an effect on car mpg? What about car weight?

Note: In the case of cylinders, R will interpret these as numerical data. However, in this case, we may want cylinder number treated as a factor levels. In cases where a number should not be treated as a number, define it as a character.

```
summary(aov(mpg~as.character(cyl), data=mtcars))
```

Again, this output is not good for exporting. To parse the summary into a data frame:

```
as.data.frame(summary(aov(y~x))[[1]])
```

This is interesting, but what if you want to know which levels of x are significantly different? (e.g. Which cylinder comparisons are significantly different?) For this, a post-hoc test is needed to correct for multiple testing. The Tukey HSD test is applied on the results of an ANOVA.

Tukey HSD test:

```
TukeyHSD(aov(y~x))              #multiple comparisons of means
```

Example 4. Which cylinder comparisons are significant at $P<0.01$?

Once again, parsing the relevant data (p-values) is not intuitive:

```
TukeyHSD(aov(y~x))[[1]]
```

Or even more complicated:pvalues

```
as.data.frame(TukeyHSD(aov(y~x))[[1]][,4])
```

What if there are multiple factors involved? More complex multi-factor models can be defined for ANOVA.

Two-Way ANOVA:

```
aov(y~x+w)      #examines the effect of factors x and w on y
aov(y~x*w)      #examines the effect of factors x and w, and their interaction, on y
```

Example 5. What is the effect of engine cylinder number and car weight on mpg? Do they have a significant interactive effect?

A Tukey test cannot be performed on an ANOVA with multiple factors, but can be used to examine a single factor within a multi-factor ANOVA.
TukeyHSD(aov(y~x*w, 'x')) #returns the effect of the levels of only x

Nonparametric tests (i.e. no assumption of normality)
Many nonparametric tests involve testing data based on rankings. A simple option is to rank-transform the data, then run ANOVA and Tukey HSD.

Kruskal-Wallis test:
The Kruskal-Wallis test (sometimes also called "one-way ANOVA on ranks") is a rank-based nonparametric test. It cannot analyze multiple variables.
kruskal.test(y ~ x)

Example 6. Using the Kruskal-Wallis test, does cylinder number have an effect on mpg?

Post-hoc tests
Nemenyi test
require(PMCMR)
posthoc.kruskal.nemenyi.test(y, x, method = “Tukey”)
#y is dependent variable, x is independent variable, method defines the approx. distribution for error rate correction and is “Tukey” or “Chisquare”
#output is the lower triangle of the matrix containing the p-values of the pairwise comparisons

Pairwise Wilcoxon rank sum test
pairwise.wilcox.test(y, x, p.adj = method)
#use p.adject method to correct for multiple testing and control for family-wise error-rate (FWER) bonferroni is most conservative, BH is least stringent (but most powerful), hochberg is superior for controlling FWER

Example 7. Use a pairwise Wilcoxon rank sum test to determine which cylinder comparisons are significant at P<0.01.

08/03/15
DESeq2
1. *Format your read counts file*
   -verify the data do not have decimals(round or truncate)
   -verify your column headers do not have inappropriate symbols (no beginning numbers, on minus signs...).
   -leave the first column header blank; R will use this as the row.names column
   -save the file as a .csv
2. *Install DESeq2 from Bioconductor in R*
   -source: http://bioconductor.org/biocLite.R

> biocLite("DESeq2") #to install DESeq2

3. *Load count data:*
> countTable = read.table(file.choose(), header=T, row.names=1)
DESeq requires count data, not normalized reads, in the format of rows representing genes, columns representing samples, and unique identifiers of each for column and row headers
for technical replicates, sum the counts to generate a single biological sample

4. Load metadata/experimental design:
> metadata = read.table(file.choose(), header=T, row.names=1)
metadata describes the count data with the first column listing the samples in the count table (i.e. column headers in countTable) and the second column describing the sample condition (in this example, the header is named "condition")
ensure that the sample names are unique; different conditions are defined by this metadata

5. Load DESeq2:
> library("DESeq2") # to open DESeq

6. Create DESeq2 Data Set (this method reads data from a matrix of read counts)
> dds = DESeqDataSetFromMatrix(countData = countTable, colData = metadata, design = ~condition)
where "design = ~condition" refers to the metadata definition column (in this example, the second column, for which the header is named "condition")

DESeq will set levels in alphabetical order
To ensure your base level is what you want, use the "relevel" command
> dds$condition = relevel(dds$condition, "WT")
where "WT" is the control for comparisons, e.g. the denominator for the fold change of the comparison
> dds = DESeq(dds)
> resultsNames(dds)

To test significance for difference in expression between the conditions "Mutant" and "WT" defined in the metadata, and order results by the adjusted p-value
> results.condition <- results(dds, contrast=list("conditionMutant", "conditionWT"))
> results.condition <- results.condition[order(results.condition$padj),]
> head(results.condition)

some options:
> head(dds)
Ex output:
class: DESeqDataSet
dim: 1 12
exptData(0):
assays(3): counts mu cooks
rownames(1): AT1G01010

rowData metadata column names(37): baseMean baseVar ... deviance maxCooks
colnames(12): BH1 BH2 ... BH11 BH12
colData names(2): Treatment sizeFactor

Information about which variables and tests were used can be found by calling the function mcols on the results object.
> mcols(results.condition)$description
Ex output:
[1] "mean of normalized counts for all samples"
[2] "log2 fold change (MAP): Treatment 30CMock vs 23CMock"
[3] "standard error: Treatment 30CMock vs 23CMock"
[4] "Wald statistic: Treatment 30CMock vs 23CMock"
[5] "Wald test p-value: Treatment 30CMock vs 23CMock"
[6] "BH adjusted p-values"

> summary(res, alpha=0.01)
Ex. output
out of 24845 with nonzero total read count
adjusted p-value < 0.01
LFC > 0 (up) : 1322, 5.3%
LFC < 0 (down) : 2232, 9%
outliers [1] : 110, 0.44%
low counts [2] : 4868, 20%
(mean count < 4)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results

To test the significance while applying a log-fold change (lfc) threshold for the significance test:
(in theory, this approach is preferable if you plan to use a fold change cutoff for your data; in this way, both fold change and p-value cutoffs will be applied to the same test)
>results.condition <-results(dds, lfcThreshold=0.1, altHypothesis="greaterAbs", contrast=list("conditionMutant", "conditionWT"))

# to do a pairwise comparison
contrast=list("conditionMutant", "conditionWT"))
OR
> contrast=c("condition","Mutant","WT")
The contrast argument of the function results takes a character vector of length three: the name of the variable, the name of the factor level for the numerator of the log2 ratio, and the name of the factor level for the denominator

dds=nbinomLRT(dds, maxit=1000, reduced = (formula(~xtreatment))

Run standard differential expression analysis (steps are wrapped into a single function)
dds = DESeq(dds)
res = results(dds)

Export results
write.table(res, file="data_results.txt", sep="\t")