To Import Data into R from excel or text files (small files < 10000 elements)

1. Convert .xlsx file to .csv or .txt(windows compatible)
2. Read into R
   1. d <- read.csv(“file.csv”, header=T)
   2. d< <- read.table(“file.txt”,header=T,sep=”\t”)
3. List the structure of the data.frame with Str function
   1. str(d) #will give objects, variables, classes, length info
   2. attributes(d) # $names vector lists headers in order starting at Col. 1
4. To add columns to data frame
   1. df$newcolname <- object #object is a vector
   2. df[ ,”newcolname”] <- object
   3. can use colnames(df) function to change column names
   4. colnames(df) <- c(“col1”,”col2”,…,”coln”)

To Import Data into R for larger files (medium to large files > 10^4 elements)

1. Convert file
2. Read into R using nrows argument
   1. sample <- read.table(“file.txt”,header=T,sep=”\t”,nrows=100)
   2. classes <- sapply(sample,class) # creates vector of col. data classes
   3. head(sample) #gives first 7 rows of file so you can see data org.
   4. classes # lists the data class of each column as vector
   5. d <- read.table(“file.txt”,colClasses=classes,header=T,sep=”\t”)
   6. str(d) #ensure levels and classes are same str(sample)

To do Summary Statistics

1. Stack data into proper data.frame
   1. stack(d) -> D
   2. Alternative method is to split data into a list (Go to 7)
2. Remove all rows containing NA
   1. G <- omit.na(D)
   2. Go to 6 for alternative method
3. Identify attributes of list (data.frame is turned into list upon stack)
   1. attributes(G)
   2. names(G) # will give column names
   3. str(G) will do this also
4. Assign vectors to each column set
   1. a <- G[,1] # w is numeric vector with values of col. 1
   2. b <- G[,2] # e is factor vector with levels/strings of col. 2
5. To generate summary statistics for each factor, use a general function
   1. tmpfn = function(x) c(mean=mean(x), var=var(x), n=length(x))
   2. tapply(a,b,tmpfn)
6. Alternatively, you can unstack, data will remain as a list
   1. g <- unstack(G)
7. Split data into a list by factor using the split() function
   1. s <- split(d,d$YOURFACTORHERE)
8. To do stats, call by factor name.
   1. mean(g$N2), or
   2. tapply(g$N2, tmpfn) # check if this works might not

To test for normality by Shapiro Wilk or Chi-Square Test

1. Load Outliers package
   1. library(outliers)
   2. chisq.out.test(x) # call by factor or location [a, g$N2, g(23:54,3)]
   3. if needed can make new vector
   4. t <- g(23:54,3), o <- g(1:22,3) # row(s), col with factor label
2. Load mvoutlier package
   1. library(mvoutlier) #may need to load this package to use
   2. shapiro.test(x) #null hypothesis is data are normally distributed
3. Use Boxplot(x) function to visualize outliers, outer whisker is 90%.
   1. boxplot(x) #must be numeric list
4. Use Histogram function to visualize outliers
   1. hist(x) #must be numeric list

To do individual t-tests

1. Create variables to compare with appropriate columns of interest
   1. x <- d[,c(column#)]
   2. y <- d[,c(column#)]
   3. z <- d[c(,column#)]
2. Remove NA values
   1. badx <- is.na(x)
   2. x <- x[!badx]
3. Use t.test function to perform Two Sample t-test
   1. t.test(x,y)
   2. Default is Welch Two Sample t-test, will use variances to estimate correct number of degrees of freedom
   3. IF DATA UNSTACKED: call by factor name [t.test(g$N2, g$arr-1)]
4. If only need t-statistic or p.value or confident interval perform following
   1. ttest <- t.test(x,y)
   2. names(ttest)
   3. ttest$YOURPARAMETER (i.e. ttest$p.value for only p-value)
   4. You can reassign variables to make this more universal (ttest <- t.test(u,v), u <- y, v <- z, ttest(u,v) #yields t-test between z and y)
5. To test for mean of group x lower than group y (one-sided/tailed t-test)
   1. t.test(x,y,alt=”less”)
   2. + , var.equal=TRUE) #assume equal variance

To do Wilcoxon rank sum test for nonparametric (non-normal distribution) data

1. This test compares medians, so null hypothesis will be medians are equal
   1. wilcox.test(x,y)

To do One-Way ANOVA

1. To do One-Way ANOVA to test null hypotheses that means are equal
   1. oneway.test(values ~ ind, data=G) #not assume equal variance
   2. + ,var.equal=T) #assume equal variance
   3. p > 0.05 indicates means are equal
2. To get more detail use anova function
   1. anova(lm(values ~ ind, data=G)
   2. aov function will give same data but must use summary command in the output of aov to get same data as anova(lm())

To do Kruskal-Wallis nonparametric test (Alternative to One-Way ANOVA)

1. This test does not depend on normality of data, similar to Wilcoxen signed- rank test used in place of t-test with data that are not normally distributed.
   1. kruskal.test(values ~ ind, data=G)
   2. p-values may be small, relationship testing null hypothesis of no difference in means

To Test for Correlation and Covariance

1. Use the cor, cor.test, and rcorr functions from Hmisc package
   1. library(Hmisc)
   2. cor(x,y) # x, y are vectors or x is df with y set to NULL
   3. cor.test(x,y) # default correlation method is Pearson
   4. If data distribution is not normal can use Spearman correlation by adding method argument
   5. cor.test(x,y, method=”spearman”) # alternative is kendall, ranks values and compares rankings not numbers themselves
2. Use the cov function to measure Covariance
   1. Cov(x,y, method=”pearson”, “kendall”, “spearman”)

To Visualize Data

1. To plot data
   1. library(lattice)
2. To see histogram
   1. hist(x, n) #x is object, n is number of intervals
3. To see kernal density plot
   1. plot(density(x))
4. To see dot plot
   1. plot(D)
5. To see barchart
   1. j <- G[,1]
   2. k <- G[,2]
   3. barchart(j~k) # display numeric var j for each level of factor k
6. To see box and whisker plot
   1. boxplot(d)
   2. bwplot(j~k)
7. To see boxplot for each combination of two factors (i.e. drug vs dxOD by strain)
   1. bwplot(drug~dxOD|strain, D, xlab=”dOD/XO”)
8. To see Quantile-Quantile (QQ) plot
   1. qqnorm(x)
   2. qqline(x,col=”red”)
9. To use scatterplot (lattice/Trellis package)
   1. library(lattice)
   2. To plot dxOD by factors drug and strain of data frame A
   3. xyplot(A$dxOD~A$drug|A$strain)
   4. To rotate axis labels 90 degrees use scales argument
   5. scales=list(x=list(rot=90))
   6. To set axes ranges use xlim or ylim arguments
   7. ylim=c(0,0.02)
   8. If variable in data frame is a character vector of factors you can modify labels of graph using xlim/ylim set as a new vector
   9. v <- c(“a”,”b”,”c”,”…”)
   10. xlim = v
10. To plot several groups of data on the same plot
    1. Use group argument, group = factor to group by
    2. xyplot(object1, object2, group=factor to group by) # groups can be distinguished by color # group is a general plot argument for use with any plotting package # colors can be set with col argument

To Subset Data

1. To subset data based on multiple factors
   1. subset(data, colname1==”factor/level” & colnameX==”factor/level”, select= colnames) #use select=c(col1,col2,colx) for multiple factors
   2. first argument is your data file name (data or d)
   3. second argument defines subset (only col1, only col1 > x & col2)
   4. third argument, the select function, specifies which columns will be used for output of data (only data that meets second argument, and output those rows data but only from colx, leave blank if want all columns)

To Store Data with associated metadata after data read into R

1. For single object use dput
   1. dput(data.frame, file=”new.R”) #dput -- save as R code
   2. d <- dget(“new.R”) #dget -- retrieve as object
   3. compare str(data.frame) and str(d), should be identical
2. For saving multiple objects use dump
   1. dump(c(x,y,z,n), file=”new.R” ) #dump – save as R code
   2. source(“new.R”) #source – read code into R
   3. Can now call each object vector using original name into R
3. To write a data table/frame to a csv file readable by Excel
   1. write.csv(object, “output.csv”, row.names=TRUE)