Class 8: GLM, logistic regressions, count data, survival stats, and mixed models
Use of the generalized linear models where error variance is not constant or normal, the lmer function for mixed models, and various survival analysis functions

Goals:

(1) Understand glm components: error structure, linear predictors, and link functions
(2) Logistic regression (proportional count data)
(3) Poisson regression (count data)
(4) Survival analysis (time-to-event data)
(5) Mixed models

When the error structure is not constant or normal, you can (1) transform the data and run lm, (2) use non-parametric models, or (3) define a more appropriate error model and use generalized linear models (glms, pronounced "glims"). To use glm, you must specify the error structure to use (e.g., "binomial") and then a link function (e.g., "logit") that is used to transform the product of the linear predictor function to a predicted value of y.

glm is used to fit generalized linear models which are useful for fitting models where the variance is not constant and/or normally distributed, such as count data (poisson), binomial proportional data (logistic), survival data (time to death, gamma).

lmer (linear mixed-effects models) from the *lme4* package is great for mixed-effects models of many kinds. ***You must install the lme4 package to use the lmer function***. Please install, with dependencies, from the `Packages & Data: Packages Installer` menu. Once installed, you need to load the library with the call: `library(lmer)`.

survival analyses are best done with the *survival* package. ***You must install the survival package for most survival analyses.***

Sample data sets:

```
#Load these data sets to use in the examples
rd<-read.csv("http://people.ucsc.edu/~ggilbert/Rclass_docs/RegressionDataset.csv")

fl<-read.csv("http://people.ucsc.edu/~ggilbert/Rclass_docs/logistregdata.csv")

fl2<-read.csv("http://people.ucsc.edu/~ggilbert/Rclass_docs/logregdata2.csv")

bl<-read.csv("http://people.ucsc.edu/~ggilbert/Rclass_docs/FactorialBlockDataset.csv")

f<-read.csv("http://people.ucsc.edu/~ggilbert/Rclass_docs/mixedmodeldata.csv")

obs2x2<-
matrix(data=c(62,74,35,22),nrow=2,ncol=2,dimnames=list(c("male","female"),c("healthy",
"sick")))

p<- read.csv("http://people.ucsc.edu/~ggilbert/Rclass_docs/ARBUMEflowers.csv")

s<- read.csv("http://people.ucsc.edu/~ggilbert/Rclass_docs/survivaldata.csv")
library(lme4); library(survival)
# end of loading data sets
```

A few notes on playing with distributions in R

You can generate a wide variety of distributions with functions described in Distributions
Some common functions for distributions include (?Distributions for the full list)

```
norm     normal
lnorm    lognormal
binom    binomial
nbinom   negative binomial
pois     poisson
unif     uniform
```

Each function has four variants. For the normal distribution

```
rnorm     normal deviates (e.g., generate a bunch of numbers)
pnorm     distribution function
qnorm     quantile function
dnorm     density function
```

#rnorm(n,mean,sd) generates n numbers from the given distribution

```
plot(x=seq(1,100,1),y=rnorm(n=100,mean=50,sd=10))
```

#plot shows, in order of generation, 100 random values from a normal distribution

#pnorm(q, mean,sd) where q is a vector of probabilities

#the p variant calculates the cumulative distribution function (CDF)
#and returns the probability that a value from the distribution #is less than value x.

```
pnorm(67,mean=50,sd=10) #~95% of values are below 67
```

#qnorm(p,mean,sd) where p is the quantile of the distribution from 0 to 1

#this is the inverse of pnorm, returning the p-th quantile of the distribution

```
qnorm(0.95,mean=50,sd=10) #66.4 is the 95th quantile
```

#dnorm(x, mean, sd) where x is a specific value from the distribution

#the d variant calculates the probability density function (PDF)
#and returns the probability of drawing that value from the distribution.
#does not make much sense for continuous distributions, but does for binomial

```
plot(x=seq(1,100,1),y=dnorm(x=seq(1,100,1), mean=50, sd=10), main="dnorm
mean=500 sd=10")
```

#plot shows probability of drawing each value from 1 to 100

#Three handy ways to look at distributions

```
par(mfrow=c(3,1))
hist(rnorm(n=100,mean=50,sd=10)) #plot histogram of 100 normal values
plot(ecdf(rnorm(n=100,mean=50,sd=10))) #plot continuous dist function
qqnorm(rnorm(n=100,mean=50,sd=10)) #QQ plot to examine normality
par(mfrow=c(1,1))
```

Fitting your data to a distribution

This is tricky, complicated stuff but as a first approximation

```
library(MASS)
fout<-fitdistr(fl$nndist,"lognormal") #finds parameter fit to lognormal
fout; fout$loglik #note loglik comparisons require same # parameters
par(mfrow=c(1,2));hist(fl$nndist); hist(rlnorm(fout$n, fout$estimate[1],
fout$sd[1])); par(mfrow=c(1,1))
```

Cheat Sheet of Family Objects for Error Structure and Link Functions

Generalized linear models (glm) allows response variables to have any distribution (not just normal), so you must specify the appropriate error distribution to be used (called family type; e.g., Gaussian (= normal), binomial, poisson)) and the associated link functions for your analysis (the link function varies normally with the predicted values). The maximum likely estimates of coefficients through glm are from iterative reweighting. not Ordinary Least Squares (as is done for lm)

(Note that for lm, the family object is gaussian(link="identity").)

The available family objects, and their default link functions are:

```
binomial(link = "logit")
gaussian(link = "identity")
Gamma(link = "inverse")   #note the capital G
inverse.gaussian(link = "1/mu^2")
poisson(link = "log")
quasi(link = "identity", variance = "constant")
quasibinomial(link = "logit")
quasipoisson(link = "log")
```

In glm models, you declare the error structure with a call to family=

```
glm(y~x,family=binomial(link="logit")) #this is for a logistic regression
```

If you want to use the default link function, you can omit the link="logit" part and just call the error structure, and it uses the default associated link function:

```
glm(y~x,binomial)
```

However, some families can use multiple link functions, depending on your data structure:

```
gaussian(link = "identity")  #gaussian is the normal distribution
gaussian(link = "log")
gaussian(link = "inverse")
binomial(link="logit") #for logistic
binomial(link="probit") #for normal CDF
binomial(link="cauchit") #for Cauchy CDF
binomial(link="log")
binomial(link="cloglog") #complementary log-log
Gamma(link="inverse")
Gamma(link="identity")
Gamma(link="log")
poisson(link="log")
poisson(link="identity")
poisson(link="sqrt")
inverse.gaussian(link="1/mu^2")
inverse.gaussian(link="inverse")
inverse.gaussian(link="identity")
inverse.gaussian(link="log")
```

Generalized Linear Models (glm)

The glm function allows you to specific the error structure and associated link functions appropriate to analysis of your data, so you are not limited to normal distributions (as you are for lm).

The glm algorithm finds coefficients for the models by maximum likelihood rather than through ordinary least squares (as is done for lm). This means you don't get the same F-statistics and R-squared values as you do in OLS – this relates to the philosophy of the approaches, and is beyond the scope of this class.

However, let's look briefly at running a simple linear regression using lm and glm, to compare the outputs. The lm function assumes normal distribution for the error structure; to do the same in glm, we need to specify that the family is Gaussian (= normal) and the link function is the "identity" function.

#here is the same analysis run using lm and glm

```
summary(lout<-lm(precip~temp,data=rd))
Call:
lm(formula = precip ~ temp, data = rd)

Residuals:
    Min      1Q  Median      3Q     Max
-148.43  -86.78   -2.99   75.30  151.42

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  125.227     51.396   2.437   0.0214 *
temp           2.051      3.270   0.627   0.5356
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 89.7 on 28 degrees of freedom
Multiple R-squared:  0.01385, Adjusted R-squared:  -0.02137
F-statistic: 0.3934 on 1 and 28 DF,  p-value: 0.5356

summary(gout<-glm(precip~temp, data=rd, family=gaussian (link="identity")))
Call:
glm(formula = precip ~ temp, family = gaussian(link = "identity"),
    data = rd)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-148.43   -86.78    -2.99    75.30   151.42

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  125.227     51.396   2.437   0.0214 *
temp           2.051      3.270   0.627   0.5356
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 8046.519)
 #NOTE that the Dispersion parameter is square of the Residual Standard Error
    Null deviance: 228468  on 29  degrees of freedom
Residual deviance: 225303  on 28  degrees of freedom
AIC: 358.86
Number of Fisher Scoring iterations: 2
```

Note: AIC=2k-2ln(L) where k=df and ln(L) is the log-likelihood

```
logLik(gout) #logLik -176.4 df=3
2*3-2*(-176.4282) #which is same as AIC for gout
```

Logistic regression v.1 (data are binary observations as 0 or 1)

Logistic Regression is used when you want to model how a quantitative independent variable affects the outcome of a binary response. For instance, imagine you are interested in how the size of a plant (number of leaves) and the density of plants (as distance to nearest neighbor) affect the likelihood that individual plants in a population produce flowers.
For each of 40 plants you record if the plant flowered during the season (flowered = 1 if yes, 0 if no), the number of leaves it had at the end of the season (leaves), and the distance to the nearest conspecific neighbor (nndist). (`see data set fl`). You want to ask if probability of flowering is a function of leaves, nndist, or an interaction between them.

```
leaves nndist flowered
   5   2.19        0
   1   1.21        0
   2   2.85        0
   2   3.22        0
   5   2.30        1
   4   3.04        1
etc.
```

Flowering is a binomial response. Binomial errors have
asymptotes toward zero as the proportion approach 0 and 1 so the variance is hump-shaped.
Remember that a logistic curve is an S-curve.
The linear predictor used in this logistic regression is $\eta_i = x_{i,lvs}\beta_{lvs} + x_{i,nnd}\beta_{nnd} + x_{i,lvs.xnnd}\beta_{lvs.xnnd}$
where *x* are the explanatory variables and the β are the parameters to be estimated. The values given by the linear predictor with binomial error are logits, where logit(p) = ln(p/(1-p)). The inverse of the logit linking function can then be to transform the logits back to the value of p where p=exp(logit(p)) / (1 + exp(logit(p)).

Logistic regression is performed using the glm function – generalized linear models.
GLM does linear models where you can specify the error structure (here "binomial"), and connect it to the model with a link function (here, "logit"). You specify the error type as "family", and the link function as "link". (Note: in glm, if you set family=binomial, the link function defaults to "logit", so you can specify it explicitly or not, as you choose).

The basic format for this logistic regression is then :

```
glm(flowered~leaves*nndist,family=binomial(link="logit"),data=fl)
```

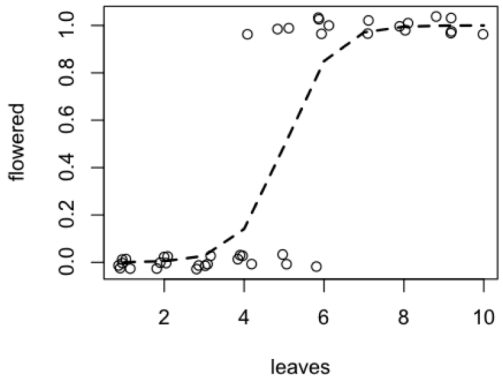
Use AIC to find the minimum model (f~leaves, f~nndist, f~leaves+nndist, etc.).

```
flout<-glm(flowered~leaves*nndist,family=binomial(link="logit"),data=fl)
flout2<-glm(flowered~leaves+nndist,family=binomial(link="logit"),data=fl)
flout3<-glm(flowered~leaves,family=binomial(link="logit"),data=fl)
flout4<-glm(flowered~nndist,family=binomial(link="logit"),data=fl)
AIC(flout,flout2,flout3,flout4) #compare reduced models
summary(flout3) #show summary of best model
#alternatively get the same place with the stepwise function
summary(step(glm(flowered~leaves*nndist,family=binomial(link="logit"),data=fl)))

plot(jitter(flowered,.2)~jitter(leaves),data=fl)  #plot the original data with jitter
points(flout3$fitted.values[order(flout3$data$leaves)]~
flout3$data$leaves[order(flout3$data$leaves)],type="l",lty="dashed",lwd=2)
```

#look at what happens if you do not include the order function
#Note: jitter only affects the display, not the data used in analysis.
#it makes it easier to see how many data points are present

The output looks like this:

```
> AIC(flout,flout2,flout3,flout4) #compare
reduced models
       df      AIC
flout   4 24.08837
flout2  3 22.12320
flout3  2 20.26362   #this is the best model
flout4  2 57.31238
```




```
> summary(flout3) #show summary of best model

Call:
glm(formula = flowered ~ leaves, family = binomial(link = "logit"), data = fl)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.94568  -0.23503  -0.04037   0.10149   1.97943

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -8.8804     3.1987  -2.776  0.00550 **
leaves        1.7683     0.6345   2.787  0.00532 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 54.548  on 39  degrees of freedom
Residual deviance: 16.264  on 38  degrees of freedom
AIC: 20.264

Number of Fisher Scoring iterations: 7

> anova(flout3, test="Chisq") #test the fit of the model
Analysis of Deviance Table
Model: binomial, link: logit
Response: flowered

Terms added sequentially (first to last)

Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                       39     54.548
leaves  1   38.285        38     16.264 6.114e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

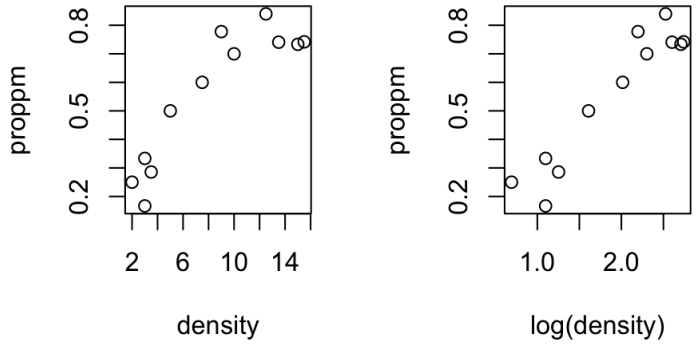
note: the model is logit(flower)=-8.8804+1.7683*leaves
recall that the antilogit is exp(logit(p)) / (1 + exp(logit(p))

```
#try this to show the curve
antilogit<-function(x) {exp(x)/(1+exp(x))}
lvs<-seq(1,10,.1) #create points along the desired x domain
logithat<--8.8804+1.7683*lvs #predict the logits from the formula
plot(lvs,antilogit(logithat),type= "l")
```

Logistic regression v.2 (where data are available as frequency counts)
Sometimes, you have frequency counts, rather than a single line for each individual. For instance, imagine you have a study where you measure the density of *Stachys bullata* (plants/m^2) in a series of random plot, and for each plant in the plot you record if the plant has powdery mildew (sick) or not (healthy). From those data you can calculate proportion with powdery mildew (proppm). You want to ask if probability of disease incidence (proportion of plants) in the plot is a function of host density.

```
#look at plots of proportion with density and log(density) in data frame fl2
par(mfrow=c(1,2))
plot(proppm~density,data=fl2)  #linear density
plot(proppm~log(density),data=fl2)  #log density
par(mfrow=c(1,1))
```

```
fl2  #use data frame fl2
   density sick healthy proppm
1      2.0    1       3 0.2500
2      3.0    1       5 0.1667
3      3.0    2       4 0.3333
4      3.5    2       5 0.2857
5      5.0    4       6 0.4000
6      7.5    5       5 0.5000
7      9.0   10       8 0.5556
8     10.0   14       6 0.7000
9     12.5   21       4 0.8400
10    13.5   20       7 0.7407
11    15.0   22       8 0.7333
12    15.5   23       8 0.7419
```




```
# To do analysis you first need to bind sick and healthy counts
# together into one object
y<-cbind(fl2$sick,fl2$healthy)

#logistic regression takes the form
#out<-glm(Counts~IndepVar, family=errortype(link="linkFunction")
lrm_out<-glm(y~fl2$density, family=binomial(link="logit"))
summary(lrm_out)  #get coef, deviances, and model fits

plot(proppm~density,ylab="proportion with powdery mildew", data=fl2)
lines(fl2$density,predict(lrm_out,list(density),type="response"))
```

```
> y
      [,1] [,2]
 [1,]    1    3
 [2,]    1    5
 [3,]    2    4
 [4,]    2    5
 [5,]    5    5
 [6,]    9    6
 [7,]   14    4
 [8,]   14    6
 [9,]   21    4
[10,]   20    7
[11,]   22    8
[12,]   23    8
#success and
failure (1/0) in
one object, as
paired columns
```

Note: compare these

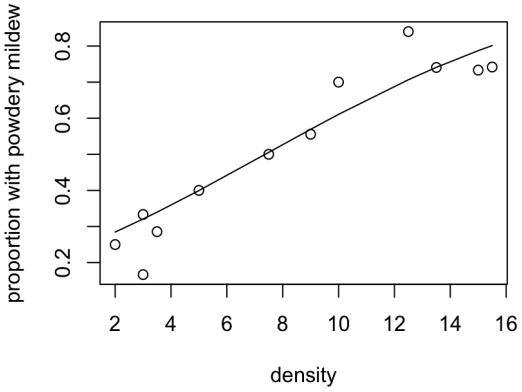
```
antilogit<-function(x) {exp(x)/(1+exp(x))} #useful
lrm_out$fitted.values  #fitted values in model object
fitted(lrm_out) #extract the fitted values from model object
predict(lrm_out,list(density),type="response") #gives fitted probs
predict(lrm_out,list(density)) #predict gives logits
antilogit(predict(lrm_out,list(density))) #convert logits to fitted
```

Logistic regression v.3 (using proportions and weights)

```
summary(glm(I(fl2$sick/(fl2$sick+fl2$healthy))~fl2$density, binomial,
weights=(fl2$sick+fl2$healthy)))
#can do the logistic regression on proportions if you include weights
#that indicate the total number of individuals for that proportion.
```

The output from logistic regression v.2 looks like this:

```
Call:
glm(formula = y ~ fl2$density, family = binomial(link = "logit"))

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.85736  -0.40473  -0.07568   0.01858   1.55260

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.26495    0.43828  -2.886   0.0039 **
fl2$density  0.17153    0.03833   4.475 7.63e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 27.0511  on 11  degrees of freedom
Residual deviance:  5.1075  on 10  degrees of freedom
AIC: 43.272

Number of Fisher Scoring iterations: 4

> anova(lrm_out, test="Chisq")
Analysis of Deviance Table

Model: binomial, link: logit

Response: y

Terms added sequentially (first to last)


            Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                           11    27.0511
fl2$density  1   21.944        10     5.1075 2.808e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```



So, from the summary output, the model is
logit(sick)=-1.26495 + 0.17153*density

```
#to see if logistic regression has a better fit with log-transformed density
llrm_out<-glm(y~log(fl2$density), family=binomial(link="logit"))
AIC(lrm_out,llrm_out)
          df      AIC
lrm_out    2 43.27156
llrm_out   2 41.86101

#log transformation might be marginally better, but not much
```

A few random notes about examining models that I have found confusing

Let's go back and look at the various models from Logistic Regression v1.

```
flout<-glm(flowered~leaves*nndist,family=binomial(link="logit"),data=fl)
flout2<-glm(flowered~leaves+nndist,family=binomial(link="logit"),data=fl)
flout3<-glm(flowered~leaves,family=binomial(link="logit"),data=fl)
flout4<-glm(flowered~nndist,family=binomial(link="logit"),data=fl)
```

There are several ways to compare models

1. Stepwise does it all for you (running blind) using AIC (not recommended – use your eyes and brain)

```
step(glm(flowered~leaves*nndist,family=binomial(link="logit"),data=fl))
```

2. Fit multiple, specified models and then compare with AIC
compares reduced models looking for lowest AIC with rule of thumb that must be different by 2 units to be considered different. Here flout3 is best.

```
AIC(flout,flout2,flout3,flout4)
       df      AIC
flout   4 24.08837
flout2  3 22.12320
flout3  2 20.26362
flout4  2 57.31238
```

3. Summary of the model object gives p-values from Wald tests for dropping each coefficient compared to the full model with all the coefficients

```
summary(flout)
Estimate Std. Error z value Pr(>|z|)
(Intercept)    -8.7160     6.7912  -1.283    0.199
leaves          1.6393     1.2713   1.289    0.197
nndist         -0.2971     2.9565  -0.101    0.920
leaves:nndist   0.1116     0.5970   0.187    0.852
```

4. ANOVA Likelihood Ratio Test on model object adds terms sequentially, comparing smaller model with the next most complex (leaves *vs.* leaves + nndist) (kind of weird comparisons)

```
anova(glm(flowered~leaves*nndist,family=binomial(link="logit"),data=fl), test="LRT")
             Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                            39     54.548
leaves         1   38.285        38     16.264 6.114e-10 ***
nndist         1    0.140        37     16.123    0.7079
leaves:nndist  1    0.035        36     16.088    0.8520
```

4. ANOVA Likelihood Ratio Test, model by model (note the difference from 4; here it compares leaves *vs.* leaves + nndist + leaves*nndist)

```
anova(flout,flout3,test="LRT")
Analysis of Deviance Table

Model 1: flowered ~ leaves * nndist
Model 2: flowered ~ leaves
  Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1        36     16.088
2        38     16.264 -2 -0.17525   0.9161
```

*Recall that AIC=2*DF – 2ln(L) and logLikelihood (L) is -2*Residual Deviance*

```
logLik(flout3)  #'log Lik.' -8.131812 (df=2)
2*2-2*(-8.131812) #20.26362 which is AIC from AIC(flout3)
and Resid. Dev. From anova(flout3) is 16.264 which is also -2*(-8.131812)
```

Poisson Regression (Log Linear models)
Poisson regression is useful when response variables are counts, rather than continuous. This is particularly useful when counts of something happening are available, but we don't now know often something does not happen (when you know the proportion of total times something happens, look instead at logistic regression).

For example, every two weeks we record all the species of flowers found in 41 litter traps on the UCSC Forest Ecology Research Plot. Data set p includes the number of times that flowers of *Arbutus menziesii* have been found in each trap, as well as the number of stems and the total basal area of *A. menziesii* within a 10-m radius of that trap. Let's ask if the number of times flowers of *A. menziesii* are found in a trap is a function of the number of individuals (stems) or the size (basal area) of *A. menziesii* nearby. Because the flower data are counts, and bounded by zero, Poisson regression is appropriate.

```
head(p)

summary(pout1<-glm(flowers~1,family=poisson(link="log"),data=p)) #just the
intercept

summary(pout2<-glm(flowers~stems,family=poisson(link="log"),data=p)) #number
of stems

summary(pout3<-glm(flowers~basalarea,family=poisson(link="log"),data=p))
#basal area

AIC(pout1, pout2, pout3) #AIC comparison of the three and pout3 is best
par(mfrow=c(2,2)) #set up to compare poisson and gaussian
plot(flowers~basalarea,data=p, main="poisson") #make a graph of the data
poutpred<-predict(pout3,list(basalarea= seq(min(p$basalarea),max(p$basalarea),.01)))
#gets predicted values
#but the link function returns the log value, so you must first
#do the antilog of the predicted values
lines(seq(min(p$basalarea),max(p$basalarea),.01),exp(poutpred))

plot(pout3,which=1) #show Residuals vs Fitted plot

#Now compare these results to fitting with a Gaussian error structure
#Compare to what you would get if you did the analysis with lm
summary(gout<-glm(flowers~basalarea,family=gaussian,data=p))
summary(lm(flowers~basalarea, data=p))

plot(flowers~basalarea,data=p, main="gaussian")
goutpred<-predict(gout,list(basalarea=seq(min(p$basalarea),max(p$basalarea),.01)))
#gets predicted values
lines(seq(min(p$basalarea),max(p$basalarea),.01),goutpred)

plot(gout,which=1) #show Residuals vs Fitted plot
par(mfrow=c(1,1))
```

Look at the residual deviance of the models – should be close to the df. Poisson is closer than Gaussian.

Output of Poisson regression

```
Call:  glm(formula = flowers ~ basalarea, family = poisson(link = "log"), data = p)
Coefficients:
(Intercept)    basalarea
     0.5045       1.2392

Degrees of Freedom: 40 Total (i.e. Null);  39 Residual
Null Deviance:          126.8
Residual Deviance: 68.21   AIC: 167.3
Analysis of Deviance Table

          Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                         40    126.769
basalarea  1   58.557        39     68.213 1.975e-14 ***

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0256  -1.8200   0.0208   0.8978   2.2686  #should be normal with median=0

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.5045     0.1402   3.599 0.000319 ***
basalarea     1.2392     0.1543   8.033 9.51e-16 ***

    Null deviance: 126.769  on 40  degrees of freedom
Residual deviance:  68.213  on 39  degrees of freedom  #Residual should be ~ d.f.
AIC: 167.33

> pchisq(deviance(pout),df.residual(pout),lower=F)
[1] 0.002605805  # significant goodness of fit says error structure not well captured
```

Output of Gaussian regression (compare to what you would get from lm)

```
Call:  glm(formula = flowers ~ basalarea, family = gaussian, data = p)

Coefficients:
(Intercept)    basalarea
      1.249        5.199

Degrees of Freedom: 40 Total (i.e. Null);  39 Residual
Null Deviance:          346.4
Residual Deviance: 117.4   AIC: 165.5
Analysis of Deviance Table

          Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                         40     346.44
basalarea  1   229.04        39     117.40 < 2.2e-16 ***

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-3.41713  -1.24948  -0.05991   1.21499   3.94009  # even more off than poisson

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    1.250      0.351   3.560 0.000994 ***
basalarea      5.199      0.596   8.723 1.06e-10 ***

(Dispersion parameter for gaussian family taken to be 3.010280)

    Null deviance: 346.44  on 40  degrees of freedom
Residual deviance: 117.40  on 39  degrees of freedom  #further off than poisson
AIC: 165.49
```

Analysis of Simple Contingency Tables

Categorical Count Data (using 2x2 count table in matrix obs2x2)

Count data in categories can be analyzed by fisher's exact tests, chisquare teste, or glm.

```
obs2x2 #start with this table
       healthy sick
male        62   35
female      74   22
```

 # Fishers exact test

```
obs2x2
fisher.test(obs2x2)

      Fisher's Exact Test for Count Data

data:  obs
p-value = 0.05806
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.2654182 1.0349204
sample estimates:
odds ratio
 0.5283995
```

#chisq.test(obs)

```
chisq.test(obs2x2)

      Pearson's Chi-squared test with Yates' continuity correction

data:  obs
X-squared = 3.4109, df = 1, p-value = 0.06477
```

#Note: for complex, multi-way contingency tables, use glm with family=poisson.

Survival statistics (time-to-event analysis)

Often how long it take for an event to happen (death, germination, pupation) is more interesting than whether it happens (everything dies). Analysis of time-to-event (also called failure time or time-to-death or survival analysis) allows you to do that. You can't just analyze the data with a linear model because of two problems.

(1) Failure times generally have a Gamma distribution.
(2) Observations are often censored (the experiment ends before all individuals die).

There are a number of ways to handle survival analyses.

Survival v. 1. Everyone dies (no censored observations). The glm version (limited use)

Here we will use data frame s, ignoring the "died" column. Assume the all the values in weeks indicate how long the birds lived. We want to know if there is a difference in the rate of mortality between males and females. +
Data of this sort can be analyzed using glm, provided there is no right censoring.

```
allout<-glm(weeks~gender, Gamma, data=s)
summary(allout)

Call:
glm(formula = weeks ~ gender, family = Gamma, data = s)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1195  -0.6111  -0.2628   0.4128   1.3427

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.33333    0.03661   9.104 8.45e-14 ***
gendermale  -0.08009    0.04598  -1.742   0.0856 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Gamma family taken to be 0.4705659)

    Null deviance: 39.179  on 77  degrees of freedom
Residual deviance: 37.711  on 76  degrees of freedom
AIC: 332.01

# Coefficients table suggests a marginally significant difference with
# males surviving less time than females.
```

Setting which reference level using relevel()

Because the first record in s is for a female, the above analysis treats female as the reference level, and provides the coefficient for effect of being a male. If you want to switch the reference level, use relevel.

```
s$gender<-relevel(s$gender,"male") #makes male the reference level
summary(allout<-glm(weeks~gender, Gamma, data=s))
```

Think of this in terms of alive or dead – alive would be the reference, and you are looking to see how your predictors affect whether death occurs.

Survival v. 2. Surv and survfit with Right-censored observations: Making graphs

Most commonly, survival data are right censored (you can't wait until all of them die, or they die from unrelated causes). Here the Surv and survfit functions are useful. Use Surv to make a survival object, then survfit to create a suitable graph of survival curves.

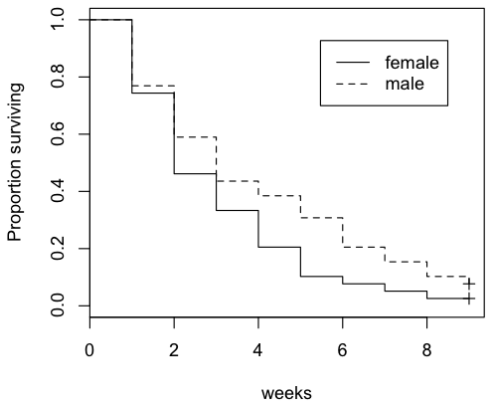
There are two ways to use Surv, depending on your data format:

1. Surv(time,event) #time is the follow-up time, event is 0=alive 1=dead at end for right censoring
2. Surv(time,time2,event) #time is start time and time2 is end time

Here is an example with the type 1, with the data in

```
s<- read.csv("http://people.ucsc.edu/~ggilbert/Rclass_docs/survivaldata.csv")
library(survival)
pwfit<- survfit(Surv(time=s$weeks,event=s$died)~s$gender)
pwfit; summary(pwfit)
plot(pwfit,lty=c("solid","dashed"),ylab="Proportion surviving",xlab="weeks")
legend(locator(1),legend=c("female","male"),lty=c("solid","dashed"))
#note: locator(1) need to click on graph where you want to place the legend
```

```
Call: survfit(formula = Surv(s$weeks, s$status) ~ s$gender)

                records n.max n.start events median 0.95LCL 0.95UCL
s$gender=female      39    39      39     38      2       2       4
s$gender=male        39    39      39     36      3       2       5

                s$gender=female
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    1     39      10   0.7436  0.0699       0.6184        0.894
    2     29      11   0.4615  0.0798       0.3288        0.648
    3     18       5   0.3333  0.0755       0.2139        0.520
    4     13       5   0.2051  0.0647       0.1106        0.380
    5      8       4   0.1026  0.0486       0.0405        0.260
    6      4       1   0.0769  0.0427       0.0259        0.228
    7      3       1   0.0513  0.0353       0.0133        0.198
    8      2       1   0.0256  0.0253       0.0037        0.177

                s$gender=male
 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    1     39       9   0.7692  0.0675       0.6477        0.914
    2     30       7   0.5897  0.0788       0.4539        0.766
    3     23       6   0.4359  0.0794       0.3050        0.623
    4     17       2   0.3846  0.0779       0.2586        0.572
    5     15       3   0.3077  0.0739       0.1922        0.493
    6     12       4   0.2051  0.0647       0.1106        0.380
    7      8       2   0.1538  0.0578       0.0737        0.321
    8      6       2   0.1026  0.0486       0.0405        0.260
    9      4       1   0.0769  0.0427       0.0259        0.228
```



Survival v. 3. Non-parametric cox proportional hazards model (constant hazard)
Survival analyses use a special object called the Kaplan-Meier suvivorship object, which is included in the model as Surv(timetoevent,censoring)
For censoring, 1=event occurred, and 0=event did not occur during study

```
library(survival)
cphout<-coxph(Surv(s$weeks,s$died)~s$gender)
summary(cphout)

Call:
coxph(formula = Surv(s$weeks, s$died) ~ s$gender)

  n= 78, number of events= 74

                coef exp(coef) se(coef)      z Pr(>|z|)
s$gendermale -0.4142    0.6609   0.2371 -1.747   0.0806 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

             exp(coef) exp(-coef) lower .95 upper .95
s$gendermale    0.6609      1.513    0.4153     1.052

Concordance= 0.552  (se = 0.044 )
Rsquare= 0.038   (max possible= 0.999 )
Likelihood ratio test= 3.05  on 1 df,   p=0.08092 #marg sig male/female diff
Wald test            = 3.05  on 1 df,   p=0.08063
Score (logrank) test = 3.09  on 1 df,   p=0.07869
```

Survival v. 4. Parametric version with a Weibull distribution (non-constant hazard)

```
pwout<-survreg(Surv(s$weeks,s$died)~s$gender)
summary(pwout)
Call:
survreg(formula = Surv(s$weeks, s$died) ~ s$gender)
              Value Std. Error     z        p
(Intercept)   1.209     0.1127 10.73 7.30e-27
s$gendermale  0.319     0.1580  2.02 4.33e-02
Log(scale)   -0.387     0.0909 -4.25 2.10e-05

Scale= 0.679   #Note that <1 indicates mortality accelerates with age

Weibull distribution
Loglik(model)= -161.5   Loglik(intercept only)= -163.5
      Chisq= 3.96 on 1 degrees of freedom, p= 0.047
Number of Newton-Raphson Iterations: 5
n= 78

#Suggests marginally significant difference in survival time
```

Note: survreg can take any of several distributions that describe distribution of times to event. These include "weibull", "exponential", "gaussian", "logistic", "lognormal" and "loglogistic". Can use fitdistr (see bottom p 2) to help decide appropriate distribution.

Mixed-effects Models

Sometimes models include both fixed effects (e.g., treatments) and random effects (e.g., split plots, repeated measures, blocks, individuals). Fixed effects have informative factor levels (e.g., fertilized vs. control), whereas random effects are generally uninformative (e.g., individual A, individual B). How the error terms are handled in such models is very important. We won't review all the ways this can happen, but generally introduce how lme4 is used to handle mixed-effects models.
Mixed models are best handled using functions found in the **lme4** package. Please install lme4 including dependencies, from the `Packages & Data: Packages Installer` menu. Then don't forget to load them into your workspace: `library(lme4).`

`lmer` function

The `lmer` function is the current workhorse function for mixed-effects models. It allows you to fixed effects and random effects explicitly. It is analogous to PROC MIXED in SAS, but has several differences in output, that relate to differences in opinion of what is more correct between the author (Douglas Bates) of lme4 and SAS (see https://stat.ethz.ch/pipermail/r-help/2006-May/094765.html if you are interested). In particular, it delves into why lmer does not give you p values! You can get the manual for lmer at http://cran.r-project.org/web/packages/lme4/lme4.pdf. It is a great reference for how to do all kinds of things with mixed models.

Let's do this by example of a single fixed effect with nested random effects.

Imagine you measure how many species of fungi are in leaves of a number of trees. Half the trees were treated with nitrogen fertilizer, and half are controls (a fixed effect called `treat`). The sampling was done on three trees per treatment; within each tree three leaves were collected from each of three branches, in a nested design. `Tree and branch` are random effects, in that order from biggest to smallest nested scale. Note that you need to *make sure your random effects are factors, not numbers*. Do this either by using text, or by specifying (use as.factor: e.g., f$tree<-as.factor(f$tree)).

The dataframe "f" looks like this:

```
> f
      treat tree branch fungi
1  nitrogen   TA     B1    16
2  nitrogen   TA     B1    11
3  nitrogen   TA     B1     6
4  nitrogen   TA     B2    12
5  nitrogen   TA     B2    16
...
51  control   TF    B17     9
52  control   TF    B18    13
53  control   TF    B18     7
54  control   TF    B18     6
```

```
lmer(depvar~fixedeffects + (1 | randomeffects), data=mydata)
```

The lmer function includes first the model formula, written as usual as `response ~ FixedEffects`.
This is followed by random-effects terms, enclosed in parentheses, separated by a "+" symbol.
The random effects terms have two parts, separated by "|", which means "by".
The model matrix comes to the left of "|"; To the right of "|" are the grouping factors.
Most often the model matrix is "1", for intercept; it can be a factor for the slope term.
"/" is used to indicate nesting, including all the levels, from big to small. Or one can specify nesting explicitly using ":", thus: (1 | tree/branch) is equivalent to (1 | tree:branch) + (1 | tree).

So, a model for the effect of treat (a fixed effect) on fungi, with the random effects of branch nested within tree, would be:

```
lmer(fungi~treat + (1 | tree/branch), data=f)  #or equivalently
lmer(fungi~treat + (1 | tree:branch) + (1 | tree), data=f)
```

if there were no fixed effect treatment, and you just wanted to look at variance across the random effects,
`lmer(fungi~1 + (1 | tree/branch), data=f)`

The output of lmer is an object of type "mer", which is a huge, nasty, gnarly beast.

```
summary(mmout<- lmer(fungi~treat + (1 | tree/branch), data=f))
anova(mmout)

Linear mixed model fit by REML
Formula: fungi ~ treat + (1 | tree/branch)
   Data: f
   AIC   BIC logLik deviance REMLdev
 329.2 339.1 -159.6    322.6   319.2
Random effects:
 Groups      Name        Variance   Std.Dev.
 branch:tree (Intercept) 2.3405e-19 4.8379e-10
 tree        (Intercept) 5.8485e-11 7.6476e-06
 Residual                2.3887e+01 4.8875e+00
Number of obs: 54, groups: branch:tree, 18; tree, 6

Fixed effects:
             Estimate Std. Error t value
(Intercept)     6.4074     0.9406   6.812
treatnitrogen   3.8889     1.3302   2.924

Correlation of Fixed Effects:
            (Intr)
treatnitrgn -0.707

Analysis of Variance Table
      Df Sum Sq Mean Sq F value
treat  1 204.17  204.17   8.547
```

There are various useful extractor functions for the output object of type “mer”, detailed in the manual

```
fixef(mmout)   #this will give the coefficients for fixed effects
ranef(mmout)   #this extracts the coefficients for random effects
fitted(mmout)  #the fitted values of the model
```

To compare different models, use the anova function:

```
mmout<- lmer(fungi~treat + (1 | tree/branch), data=f)
mmout2<- lmer(fungi~treat + (1 | tree), data=f)
anova(mmout,mmout2)
Data: f
Models:
mmout2: fungi ~ treat + (1 | tree)
mmout: fungi ~ treat + (1 | tree/branch)
       Df    AIC    BIC  logLik Chisq Chi Df Pr(>Chisq)
mmout2  4 330.57 338.52 -161.28
mmout   5 332.57 342.51 -161.28     0      1          1  #don't reject simpler mmout2
```

Analysis of a factorial design with blocks

Read in the data set FactorialBlockDataset.csv as data frame "bl"

```
bl<-read.csv("http://people.ucsc.edu/~ggilbert/Rclass_docs/FactorialBlockDataset.csv")
```

Imagine a factorial data set with two fixed effects: cultivar (wildtype and its transformed GM counterpart) and treatment (control, nitrogen, or potassium fertilizer). There are five blocks (random effect), and the response variable is plant_mass. Each block is divided into plots, and each combination of factors (e.g., control-GM, wildtype-potassium) is randomly assigned within the block.

```
bl
   cultivar treatment block plant_mass
1  wildtype   control     1       2.91
2  wildtype   control     2       2.49
3  wildtype   control     3       2.32
…
29       GM potassium     4       4.30
30       GM potassium     5       5.30
```

note: block defaults to type integer, but there is no order to the blocks. Best to convert to factor
`bl$block<-as.factor(bl$block)`

```
summary(blout<-lmer(plant_mass~cultivar*treatment + (1 | block),data=bl))

Linear mixed model fit by REML
Formula: plant_mass ~ cultivar * treatment + (1 | block)
   Data: bl
   AIC   BIC logLik deviance REMLdev
 98.58 109.8 -41.29    84.46   82.58
Random effects:
 Groups   Name        Variance Std.Dev.
 block    (Intercept) 0.49630  0.70448
 Residual             0.96674  0.98323
Number of obs: 30, groups: block, 5

Fixed effects:
                                  Estimate Std. Error t value
(Intercept)                         3.3020     0.5409   6.104
cultivarwildtype                   -0.7460     0.6218  -1.200
treatmentnitrogen                   0.1820     0.6218   0.293
treatmentpotassium                  0.0740     0.6218   0.119
cultivarwildtype:treatmentnitrogen  1.8040     0.8794   2.051
cultivarwildtype:treatmentpotassium -0.0020     0.8794  -0.002

Correlation of Fixed Effects:
                    (Intr) cltvrw trtmntn trtmntp cltvrwldtyp:trtmntn
cltvrwldtyp         -0.575
trtmntntrgn         -0.575  0.500
trtmntptssm         -0.575  0.500  0.500
cltvrwldtyp:trtmntn  0.406 -0.707 -0.707  -0.354
cltvrwldtyp:trtmntp  0.406 -0.707 -0.354  -0.707   0.500

anova(blout)
Analysis of Variance Table
                   Df Sum Sq Mean Sq F value
cultivar            1 0.1584  0.1584  0.1639
treatment           2 7.3417  3.6708  3.7972
cultivar:treatment  2 5.4300  2.7150  2.8084
```

Although Douglas Bates would not approve, you can get SAS-equivalent p-values like this:

```
pf(.1639,df1=1,df2=8,lower.tail=F) #pvalue for cultivar df2=8=2*(5-1)
pf(3.7972, df1=2, df2= 16,lower.tail=F) #pvalue for treatment df2= 16 = 2*(5-1)*(3-1)
```

For comparison for a balanced design you can do this analysis using aov. But be careful with unbalanced designs!

```
summary(blout2<-aov(plant_mass~cultivar*treatment +Error(block),data=bl))
```