Overview of regression techniques

Point count data analysis workshop 2025

Péter Sólymos

2025-11-12

Table of contents

Preamble

```
suppressPackageStartupMessages({
    library(dplyr)
    library(ggplot2)
```

```
    library(mefa4)
    library(detect)
})
```

Covariates

Variables that co-vary with the response variable. Also called as independent variables, predictors.

Let's continue with the JOSM data set:

```
x <- detect::josm$surveys |>
    select(
        Longitude,
        Latitude,
        WindStart,
        TSSR,
        DAY,
        Open,
        Water,
        Decid,
        OpenWet,
        Conif,
        ConifWet,
        Agr,
        UrbInd,
        SoftLin,
        Roads
    )
```

STOP AND EXPLAIN EACH VARIABLE!

Variable types

WHat type of variables we have? You can use the `str()` function to reveal the structure of R objects:

```
str(x)
```

```
'data.frame':   4569 obs. of  15 variables:
 $ Longitude: num  -113 -113 -113 -113 -113 ...
 $ Latitude : num  55.2 55.2 55.2 55.2 55.2 ...
 $ WindStart: int  0 0 0 0 0 1 2 0 0 0 ...
 $ TSSR     : num  0.0132 0.0666 0.0125 0.041 0.1263 ...
 $ DAY      : num  0.471 0.471 0.471 0.471 0.471 ...
 $ Open     : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Water    : num  0.02055 0.00752 0.00752 0.03284 0.00416 ...
 $ Decid    : num  0.8569 0.0443 0.0443 0.8823 0.8139 ...
 $ OpenWet  : num  0.00315 0.58355 0.58355 0.06961 0.11164 ...
 $ Conif    : num  0.0249 0 0 0 0 ...
 $ ConifWet : num  0.0368 0.3456 0.3456 0 0.0018 ...
 $ Agr      : num  0 0 0 0 0 0 0 0 0 0 ...
 $ UrbInd   : num  0 0 0 0.01056 0.00839 ...
 $ SoftLin  : num  0.00521 0.01811 0.01811 0.00141 0.01349 ...
 $ Roads    : num  0.052461 0.000913 0.000913 0.003307 0.046578 ...
```

We see that these variables are all continuous.

However, `WindStart` has very few distinct values, so we could treat it as ordinal (ordered factor):

```
table(x$WindStart)
```

```
   0    1    2    3    4    5    6
2471  926  844  287   31    5    3
```

```
x$WindOrd <- as.ordered(x$WindStart)
str(x$WindOrd)
```

```
 Ord.factor w/ 7 levels "0"<"1"<"2"<"3"<..: 1 1 1 1 1 2 3 1 1 1 ...
```

```
levels(x$WindOrd)
```

```
[1] "0" "1" "2" "3" "4" "5" "6"
```

Sometimes variables are binary. In R these can be logical (`TRUE`/`FALSE`) or coded as 0/1. Often we make such variables by discretizing other continuous or ordinal variables. E.g. We can create a binary wind variable:

```r
x$Wind01 <- ifelse(x$WindStart > 0, 1, 0)
table(x$WindStart, x$Wind01)
```

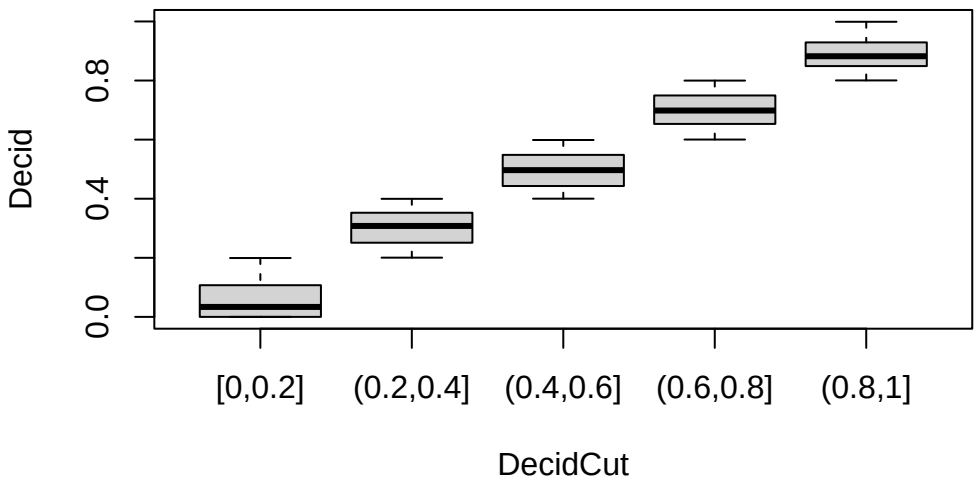
```
     0    1
0 2471    0
1    0  926
2    0  844
3    0  287
4    0   31
5    0    5
6    0    3
```

Some categorical variables depend on some kind of classification, for example we can cut a continuous variable into bins:

```r
x$DecidCut <- cut(x$Decid, seq(0, 1, 0.2), include.lowest = TRUE)
table(x$DecidCut)
```

```
[0,0.2] (0.2,0.4] (0.4,0.6] (0.6,0.8]   (0.8,1]
   1592       878       857       638       604
```

```r
boxplot(Decid ~ DecidCut, x)
```



We can also inspect the land cover proportions that add up to 1 for each row.

```
## define column names
cn <- c(
    "Open", "Water", "Agr", "UrbInd", "SoftLin", "Roads", "Decid",
    "OpenWet", "Conif", "ConifWet"
)
## these sum to 1
summary(rowSums(x[, cn]))
```

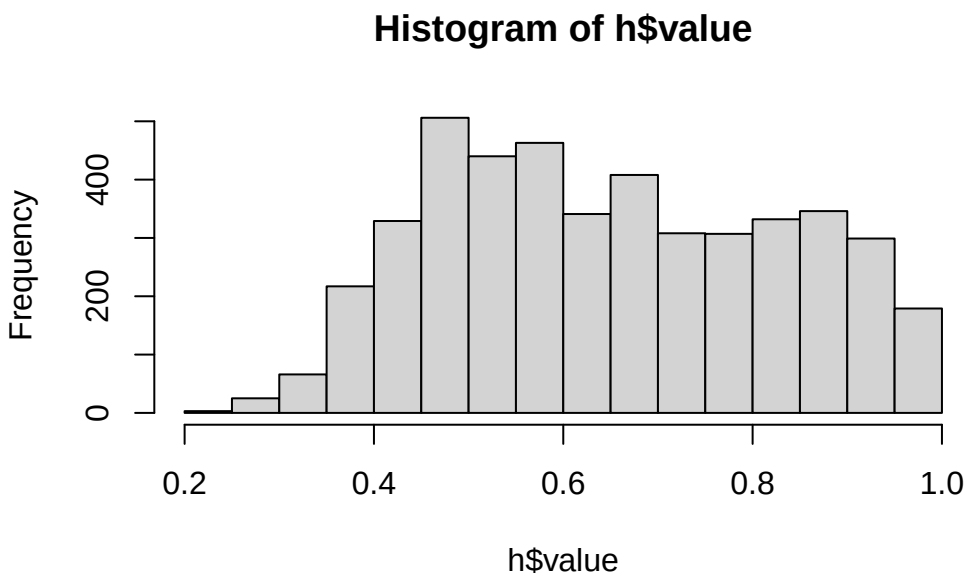
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       1       1       1       1       1
```

The `find_max()` function finds the maximum value in each row, the output contains the value and the column where it was found, we can turn that into the dominant land cover type encoded in `HAB`:

```
h <- find_max(x[, cn])
head(h)
```

```
          index     value
CL10102   Decid 0.8569106
CL10106 OpenWet 0.5835472
CL10108 OpenWet 0.5835472
CL10109   Decid 0.8822829
CL10111   Decid 0.8139365
CL10112   Decid 0.8139365
```

```
hist(h$value)
```




```
table(h$index)
```

```
    Open    Water      Agr   UrbInd  SoftLin    Roads    Decid  OpenWet
      12       10        4       14        0        2     2084      160
   Conif ConifWet
     745     1538
```

```
x$HAB <- droplevels(h$index) # drop empty levels
x$DEC <- ifelse(x$HAB == "Decid", 1, 0)
```

Other types of categorical variables are truly discrete, like observer, where there are no underlying continuous data:

```
table(detect::josm$surveys$ObserverID)
```

```
  2  14  15  19  22  26  27  28  32  41  48  57  59  63  64  65  69  75  82  86
186 223  88 267   7 191 307 227 231 140 170 126 222 240 127 154 302 127 173 280
 87  89  93  98
166  99 244 272
```

When the number of categories increase and approach the sample size, we can consider treating these variables as random effects. E.g. the `SurveyArea` variable that has 271 levels.

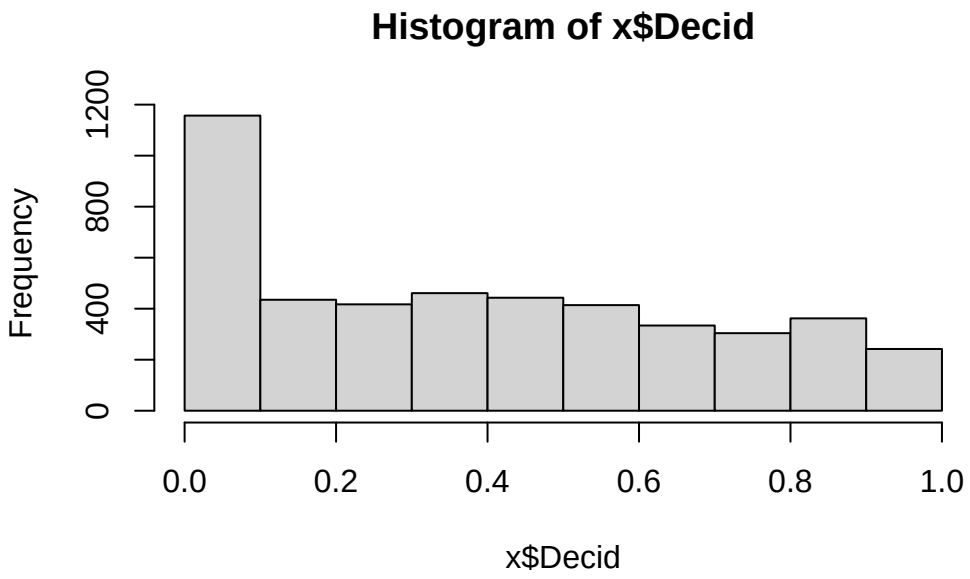
Data exploration

We should inspect each variable that we want to use as a covariate. Here are some of the most important functions:

```
summary(x$Decid) # check mean, range, missing values
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.09852 0.36132 0.38717 0.63232 0.99891
```

```
hist(x$Decid) # check skew and outliers
```



A nice way of getting all of the above nicely formatted is to use the skimr package:

```
skimr::skim(x)
```

Table 1: Data summary

Name	x
Number of rows	4569
Number of columns	20
Column type frequency:	
factor	3

numeric	17
Group variables	None

Variable type: factor

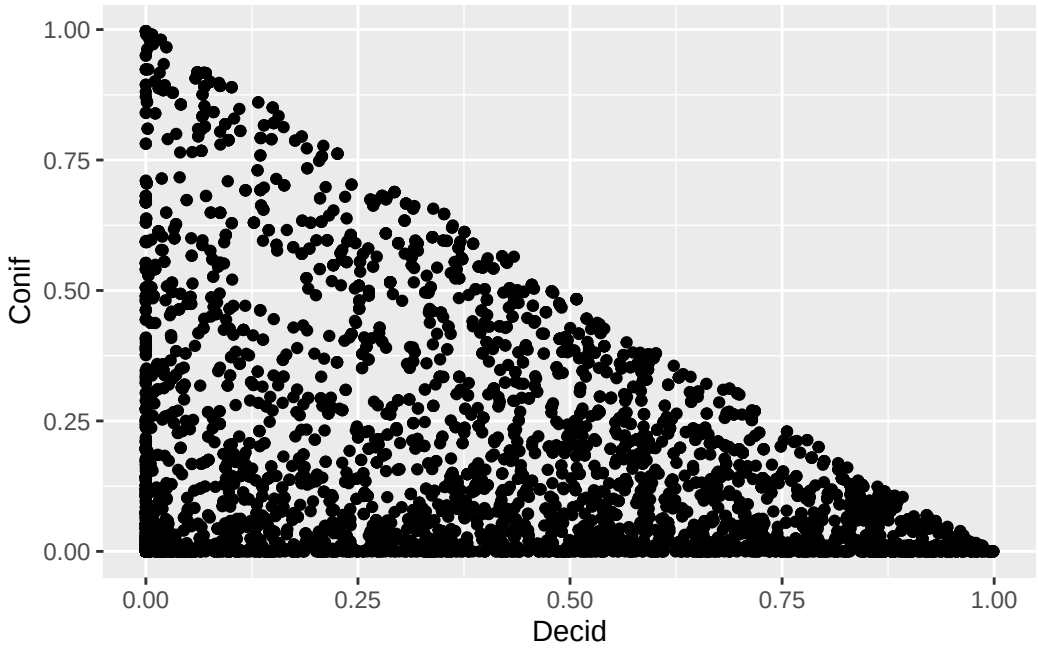
skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
WindOrd	2	1	TRUE	7	0: 2471, 1: 926, 2: 844, 3: 287
DecidCut	0	1	FALSE	5	[0,: 1592, (0.: 878, (0.: 857, (0.: 638
HAB	0	1	FALSE	9	Dec: 2084, Con: 1538, Con: 745, Ope: 160

Variable type: numeric

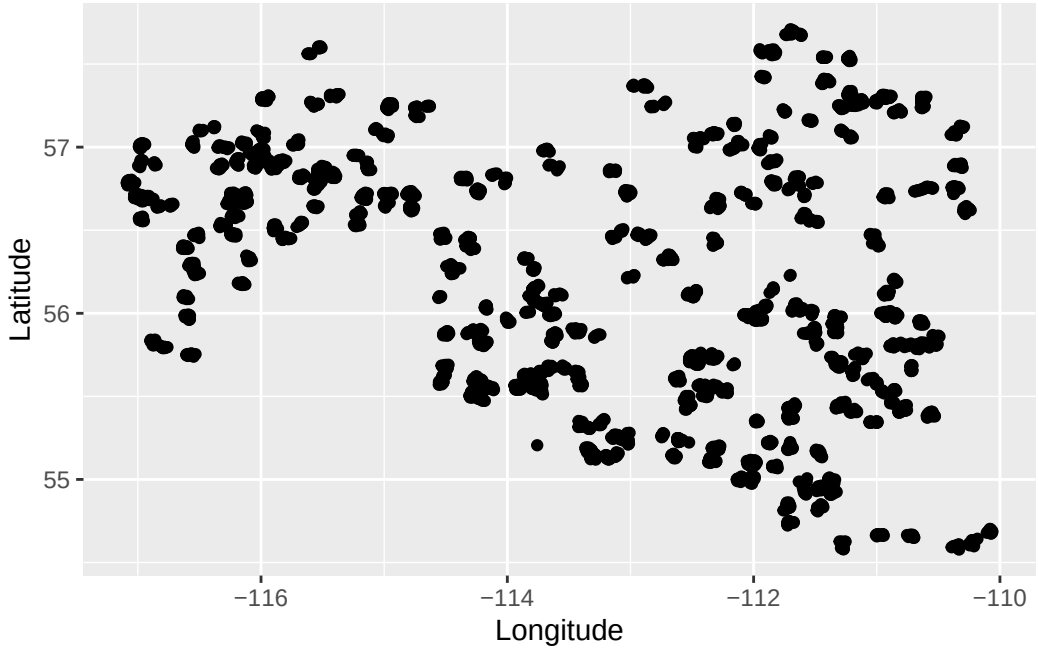
skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Longitude	0	1	-113.20	1.96	-117.09	-114.55	-112.75	-111.49	-110.06	
Latitude	0	1	56.20	0.74	54.58	55.58	56.15	56.81	57.71	
WindStart	2	1	0.80	1.02	0.00	0.00	0.00	2.00	6.00	
TSSR	0	1	0.10	0.06	-0.03	0.05	0.10	0.16	0.24	
DAY	0	1	0.45	0.03	0.39	0.42	0.45	0.47	0.50	
Open	0	1	0.01	0.04	0.00	0.00	0.00	0.00	0.65	
Water	0	1	0.01	0.04	0.00	0.00	0.00	0.00	0.81	
Decid	0	1	0.39	0.30	0.00	0.10	0.36	0.63	1.00	
OpenWet	0	1	0.07	0.12	0.00	0.00	0.02	0.08	0.87	
Conif	0	1	0.18	0.22	0.00	0.01	0.08	0.27	1.00	
ConifWet	0	1	0.30	0.29	0.00	0.04	0.20	0.50	1.00	
Agr	0	1	0.00	0.02	0.00	0.00	0.00	0.00	0.79	
UrbInd	0	1	0.01	0.05	0.00	0.00	0.00	0.01	1.00	
SoftLin	0	1	0.02	0.02	0.00	0.01	0.01	0.03	0.25	
Roads	0	1	0.01	0.02	0.00	0.00	0.00	0.01	0.27	
Wind01	2	1	0.46	0.50	0.00	0.00	0.00	1.00	1.00	
DEC	0	1	0.46	0.50	0.00	0.00	0.00	1.00	1.00	

To explore relationships between variables, make scatter and box plots:

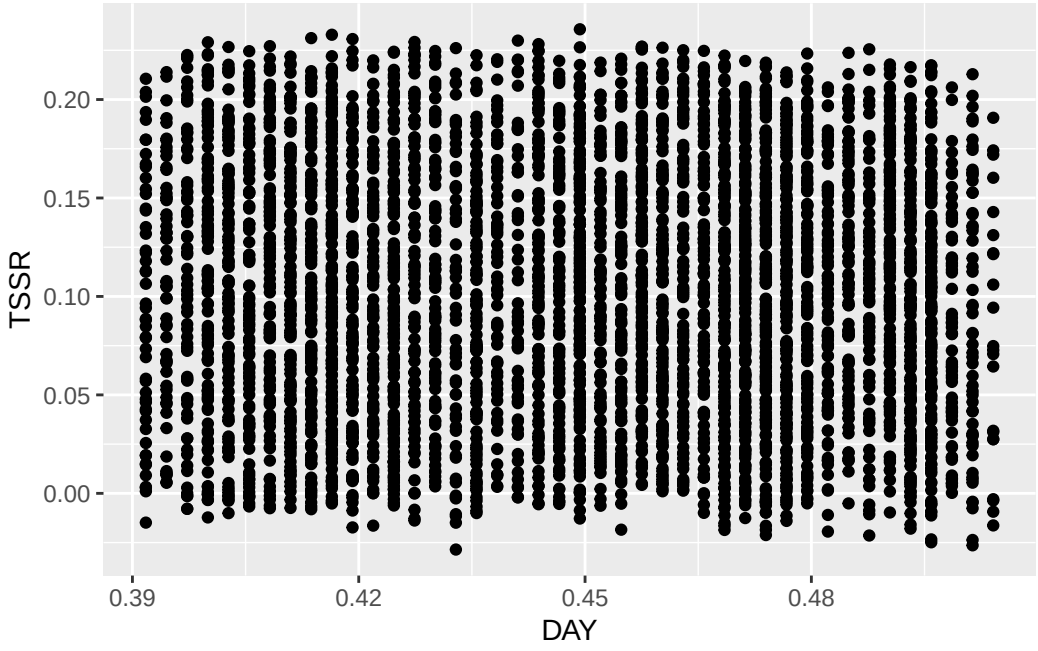
```r
x |> ggplot(aes(x = Decid, y = Conif)) +
    geom_point()
```



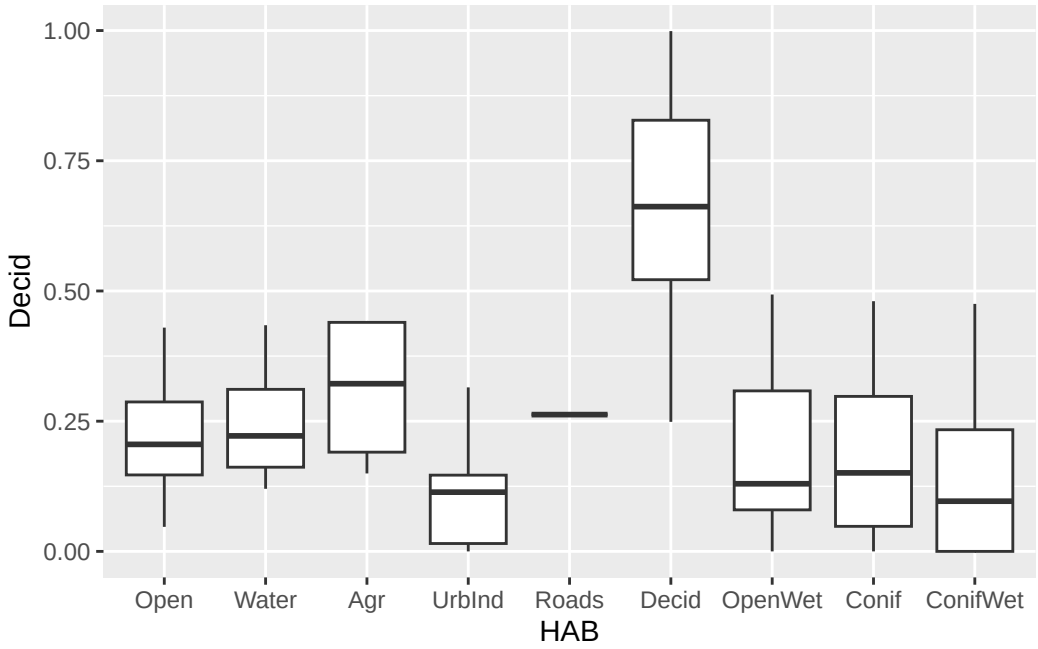

```r
x |> ggplot(aes(x = Longitude, y = Latitude)) +
    geom_point()
```




```
x |> ggplot(aes(x = DAY, y = TSSR)) +
    geom_point()
```

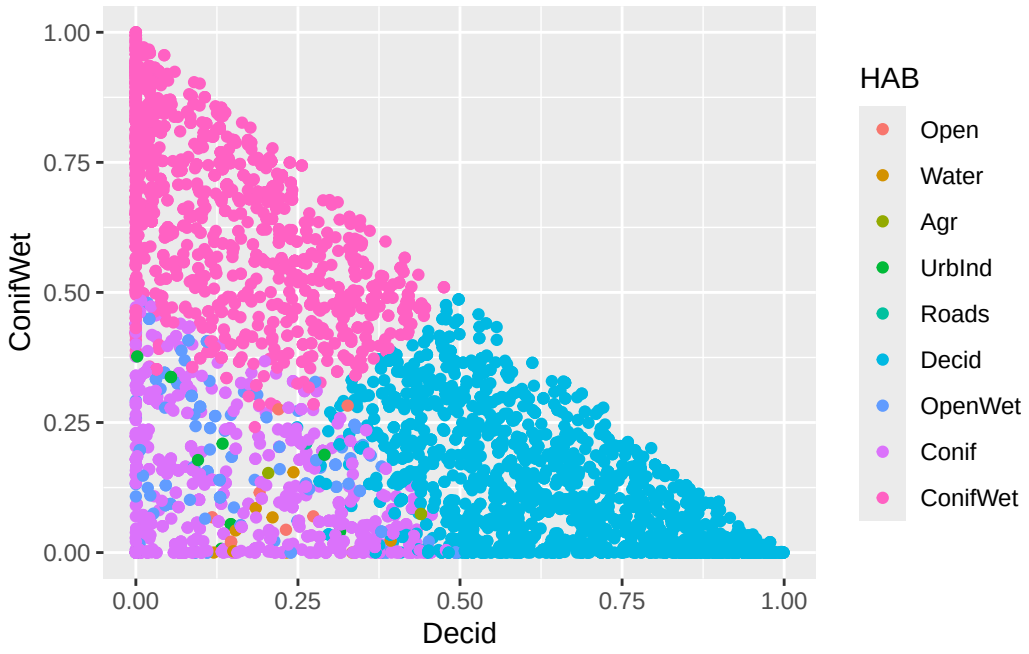


```
x |> ggplot(aes(x = HAB, y = Decid)) +
    geom_boxplot()
```



We can present 3 variables as color scatter or bubble plots:

```
x |> ggplot(aes(x = Decid, y = ConifWet, col = HAB)) +
    geom_point()
```



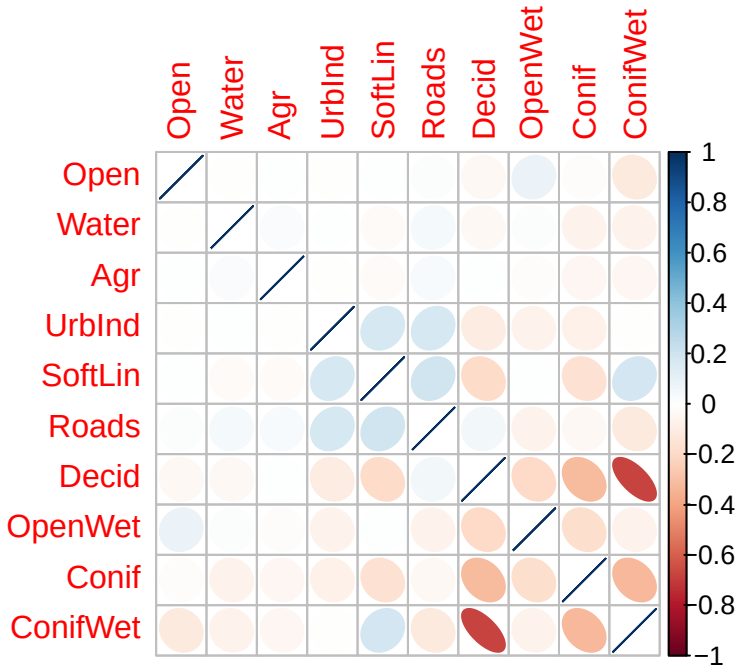
Multivariate exploration include checking correlations:

```
round(cor(x[, cn]), 3)
```

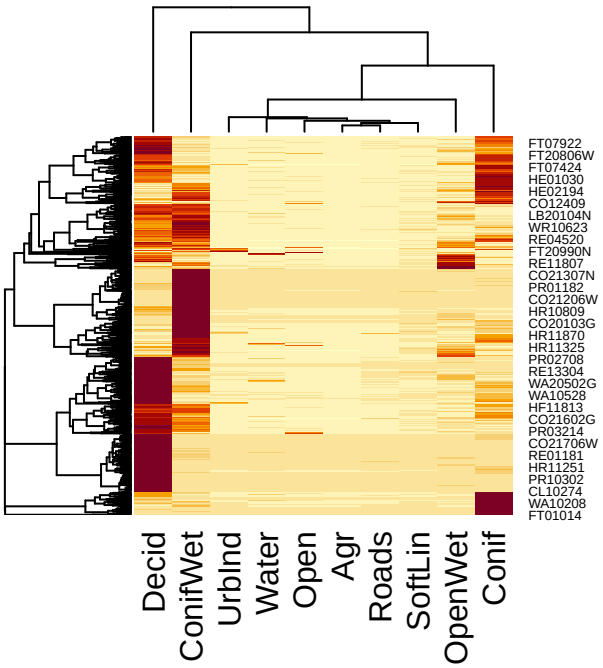
```
            Open  Water    Agr UrbInd SoftLin  Roads  Decid OpenWet  Conif
Open       1.000 -0.002  0.004 -0.007   0.006  0.011 -0.038   0.081 -0.019
Water     -0.002  1.000  0.023  0.007  -0.026  0.048 -0.039   0.012 -0.063
Agr        0.004  0.023  1.000 -0.004  -0.027  0.036  0.008  -0.012 -0.040
UrbInd    -0.007  0.007 -0.004  1.000   0.178  0.174 -0.100  -0.060 -0.070
SoftLin    0.006 -0.026 -0.027  0.178   1.000  0.193 -0.185   0.001 -0.152
Roads      0.011  0.048  0.036  0.174   0.193  1.000  0.055  -0.063 -0.037
Decid     -0.038 -0.039  0.008 -0.100  -0.185  0.055  1.000  -0.197 -0.314
OpenWet    0.081  0.012 -0.012 -0.060   0.001 -0.063 -0.197   1.000 -0.173
Conif     -0.019 -0.063 -0.040 -0.070  -0.152 -0.037 -0.314  -0.173  1.000
ConifWet  -0.115 -0.068 -0.048 -0.008   0.187 -0.120 -0.674  -0.069 -0.330
          ConifWet
Open        -0.115
Water       -0.068
Agr         -0.048
UrbInd      -0.008
SoftLin      0.187
Roads       -0.120
Decid       -0.674
OpenWet     -0.069
```

```
Conif      -0.330
ConifWet    1.000
```

```
corrplot::corrplot(cor(x[, cn]), "ellipse")
```




```
heatmap(as.matrix(x[, cn]))
```



Variable transformations

We have see examples of these:

- indicator variables (0/1)
- discretization (cut)

Other transformations include:

- sqrt: to tame outliers (not suitable for negative values)
- log: we'll see many use cases later
- polynomials: nonlinear terms (`x^2`, `x^3`, etc.)
- centering: keeps the distribution but shifts the mean
- scaling: keeps the distribution but shifts the range (often used with centering)

Compound variables

We can reduce correlation by combining variables together when those are additive:

```
x$FOR <- x$Decid + x$Conif + x$ConifWet
x$HF <- x$Agr + x$UrbInd + x$Roads + x$SoftLin
x$WET <- x$OpenWet + x$ConifWet + x$Water
```

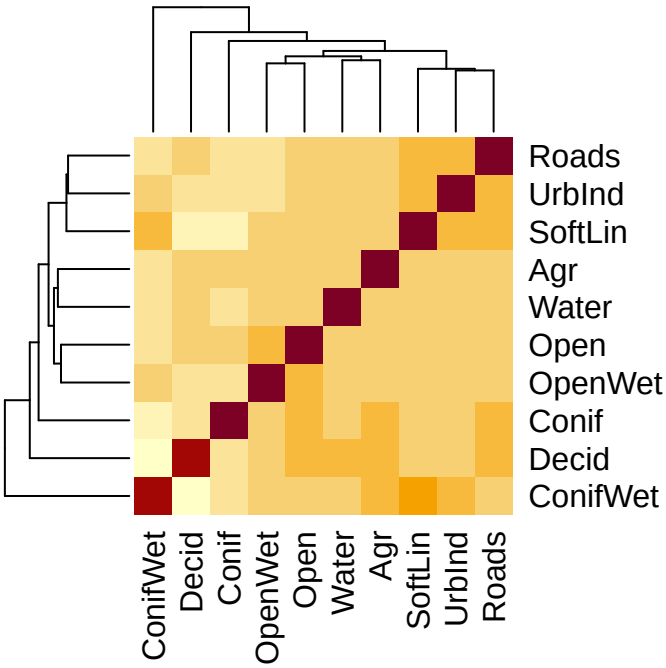
We can reduce colinearity by calculating variables relative to each other, like proportions or ratios (watch out for division by 0):

```
x$pDecid <- ifelse(x$FOR > 0, x$Decid / x$FOR, 0)
cor(x[, c("FOR", "pDecid")])
```

```
              FOR     pDecid
FOR     1.00000000 0.01124794
pDecid 0.01124794 1.00000000
```

We can also merge categories, e.g. based on their similarity:

```
heatmap(cor(x[, cn]))
```



Modeling

We have manipulated the covariates and the species counts. Let's put them together in the same data frame.

```
spp <- "OVEN" # change here if you want to use another species
detect::josm$species[spp, ]
```

```
     SpeciesID SpeciesName       ScientificName
OVEN      OVEN    Ovenbird Seiurus aurocapillus
```

```
y <- mefa4::Xtab(~ SiteID + SpeciesID, detect::josm$counts)[, spp, drop = FALSE]
x$Count <- y[rownames(x), ]
```

Let's check for missing values:

```
data.frame(missing = colSums(is.na(x))) |> filter(missing > 0)
```

```
          missing
WindStart       2
WindOrd         2
Wind01          2
```

There are only 2 rows with missing values, we could use `na.omit(x)` to drop these, or we can impute the values:

- pick a value randomly
- use the mean
- use the most common value (mode)
- Use other variables to predict the possible value

After inspecting the full data set, it turns out that site FT21204W experiences wind according to `WindEnd`:

```
sites <- rownames(x)[is.na(x$WindStart)]
detect::josm$surveys[detect::josm$surveys$SiteID %in% c("FT07424", "FT21204W"), ]
```

```
           SiteID SurveyArea Longitude Latitude       Date  StationID
FT07424   FT07424      FT074 -111.7273 55.18270 2012-06-28  FT07424-1
FT21204W FT21204W      FT212 -111.5415 57.15522 2014-06-30 FT21204W-1
         ObserverID  TimeStart VisitID WindStart PrecipStart TempStart
FT07424          57 8:20:00 AM       1        NA           0        NA
FT21204W         32 8:30:00 AM       1        NA          NA      20.3
         CloudStart WindEnd PrecipEnd TempEnd CloudEnd    TimeFin Noise
FT07424           0      NA         0      NA        0 8:30:00 AM     0
```

```
FT21204W         NA       3         0    20.3        0 8:40:00 AM     1
         OvernightRain            DateTime         SunRiseTime SunRiseFrac
FT07424           TRUE 2012-06-28 08:20:00 2012-06-28 04:49:30   0.2010437
FT21204W         FALSE 2014-06-30 08:30:00 2014-06-30 04:34:45   0.1907999
              TSSR OrdinalDay       DAY Open        Water Agr      UrbInd
FT07424  0.1461785        179 0.4904110    0 0.0023198943   0 0.006284186
FT21204W 0.1633668        180 0.4931507    0 0.0006266884   0 0.082233995
             SoftLin       Roads     Decid    OpenWet     Conif  ConifWet
FT07424  0.000000000 0.001462686 0.2005029 0.03073732 0.4908488 0.2678442
FT21204W 0.008925667 0.061308954 0.2785886 0.00000000 0.0000000 0.5683161
```

Let's set the value to 3 for site FT21204W, and the mode (0) for site FT07424:

```
x[sites, "WindStart"] <- c(0, 3)
x[sites, "WindOrd"] <- c("0", "3")
x[sites, "Wind01"] <- c(0, 1)
```

Any more `NA`s left?

```
any(is.na(x))
```

```
[1] FALSE
```

Let's inspect the response variable:

```
table(x$Count)
```

```
   0    1    2    3    4    5    6
2492  881  654  365  134   30   13
```

The distribution looks somewhat 0 inflated. What are the possible reasons for that?

- conditions lead to the absence of the species
- detected counts are lower than the actual counts

The types of models used most often to model count data and the functions used to fit them:

- Poisson: `stats::glm()`
- Negative Binomial (higher or lower variance than Poisson): `MASS::glm.nb()`
- Zero-inflated Poisson (ZIP): `pscl::zeroinfl()`

- Zero-inflated Negative Binomial (ZINB): : `pscl::zeroinfl()`

Other approaches include additive models (`mgcv::gam()`) and tree based methods (see the gbm and xgboost packages).

Let us start with the Poisson model.

Poisson null model

The null model states that the expected values of the count at all locations are identical: $E[Y_i] = \lambda$ $(i = 1, ..., n)$, where Y_i is a random variable that follows a Poisson distribution with mean λ: $(Y_i \mid \lambda) \sim Poisson(\lambda)$. The observation (y_i) is a realization of the random variables Y at site i, these observations are independent and identically distributed (i.i.d.), and we have n observations in total.

Saying the the distribution is Poisson is an assumption in itself. For example we assume that the variance equals the mean $(V(\mu) = \mu)$.

```
mP0 <- glm(Count ~ 1, data = x, family = poisson)
```

The `family=poisson` specification implicitly assumes that we use a logarithmic link functions, that is to say that $log(\lambda) = \beta_0$, or equivalently: $\lambda = e^{\beta_0}$. The mean of the observations equal the mean of the fitted values, as expected:

```
mean(mP0$y)
```

```
[1] 0.8859707
```

```
mean(fitted(mP0))
```

```
[1] 0.8859707
```

```
exp(coef(mP0))
```

```
(Intercept)
  0.8859707
```

The logarithmic function is called the link function, its inverse, the exponential function is called the inverse link function. The model family has these conveniently stored for us:

```
mP0$family
```

```
Family: poisson
Link function: log
```

```
mP0$family$linkfun
```

```
function (mu)
log(mu)
<environment: namespace:stats>
```

```
mP0$family$linkinv
```

```
function (eta)
pmax(exp(eta), .Machine$double.eps)
<environment: namespace:stats>
```

Inspect the summary

```
summary(mP0)
```

```
Call:
glm(formula = Count ~ 1, family = poisson, data = x)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.12107    0.01572  -7.703 1.33e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 7447.9  on 4568  degrees of freedom
Residual deviance: 7447.9  on 4568  degrees of freedom
AIC: 12603

Number of Fisher Scoring iterations: 6
```

Notice that the residual deviance much higher than residual degrees of freedom. This indicates that our parametric model (Poisson error distribution, constant expected value) is not quite right. See if we can improve that somehow and explain more of the variation.

We can pick an error distribution that would fit the residuals around the constant expected value better (i.e. using random effects). But this way we would not learn about what is driving the variation in the counts. We would also have a really hard time predicting abundance of the species for unsurveyed locations. We would be right on average, but we wouldn't be able to tell how abundance varies along e.g. a disturbance gradient or with tree cover.

An alternative approach would be to find predictors that could explain the variation.

Main effects

We fit a parametric (Poisson) linear model using `Decid` as a predictor:

```
mP1 <- glm(Count ~ Decid, data = x, family = poisson)
mean(mP1$y)
```

```
[1] 0.8859707
```

```
mean(fitted(mP1))
```

```
[1] 0.8859707
```

```
coef(mP1)
```

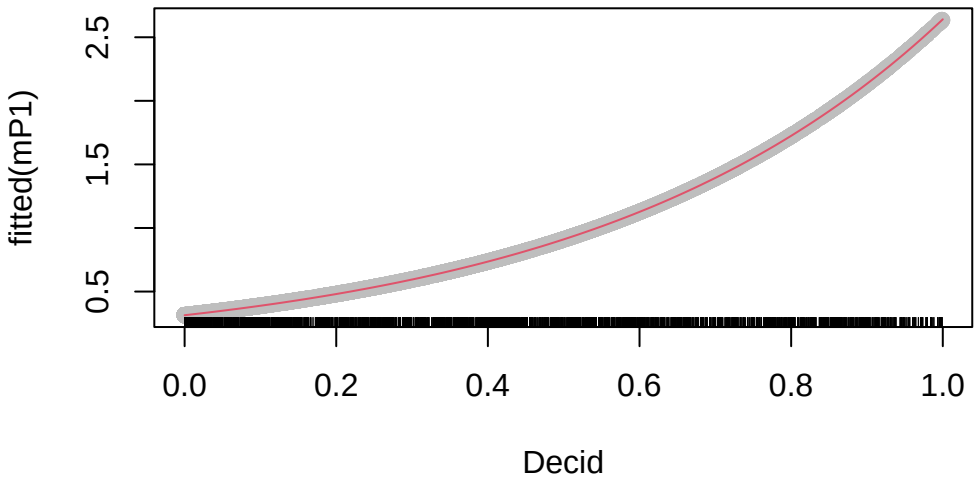
```
(Intercept)       Decid
  -1.158833    2.130040
```

Same as before, the mean of the observations equal the mean of the fitted values. But instead of only the intercept, now we have 2 coefficients estimated. Our linear predictor thus looks like: $log(\lambda_i) = \beta_0 + \beta_1 x_{1i}$. This means that expected abundance is e^{β_0} where `Decid`=0, $e^{\beta_0} e^{\beta_1}$ where `Decid`=1, and $e^{\beta_0 + \beta_1 x_1}$ in between.

The relationship can be visualized by plotting the fitted values against the predictor, or using the coefficients to make predictions using our formula:

```r
## make a sequence between 0 and 1
dec <- seq(from = 0, to = 1, by = 0.01)
## predict lambda
lam <- exp(coef(mP1)[1] + coef(mP1)[2] * dec)

plot(fitted(mP1) ~ Decid, x, pch = 19, col = "grey") # fitted
lines(lam ~ dec, col = 2) # our predicted
rug(x$Decid) # observed x values
```



The model summary tells us that residuals are not quite right The residual deviance is still higher than residual degrees of freedom (these should be close if the Poisson assumption holds, but it is much better than what we saw for the null model).

We also learned that the `Decid` effect is significant (meaning that the effect size is large compared to the standard error):

```r
summary(mP1)
```

```
Call:
glm(formula = Count ~ Decid, family = poisson, data = x)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.15883    0.03507  -33.04   <2e-16 ***
Decid        2.13004    0.05359   39.75   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 7447.9  on 4568  degrees of freedom
Residual deviance: 5760.2  on 4567  degrees of freedom
AIC: 10917

Number of Fisher Scoring iterations: 6
```

Note: we see a significant (<0.05) P-value for the intercept as well. It is totally meaningless. That P-value relates to the hull hypothesis of the intercept (β_0) being 0. There is nothing special about that, it is like saying the average abundance is different from 1.

But when β_1 is significantly different from 0, it means that the main effect has non-negligible effect on the mean abundance.

We can compare this model to the null (constant, intercept-only) model:

```
AIC(mP0, mP1)
```

```
    df      AIC
mP0  1 12602.84
mP1  2 10917.16
```

```
BIC(mP0, mP1)
```

```
    df      BIC
mP0  1 12609.27
mP1  2 10930.02
```

```
MuMIn::model.sel(mP0, mP1)
```

```
Model selection table
    (Intrc) Decid df    logLik    AICc   delta weight
mP1 -1.1590  2.13  2 -5456.581 10917.2    0.00      1
mP0 -0.1211        1 -6300.422 12602.8 1685.68      0
Models ranked by AICc(x)
```

AIC uses the negative log likelihood and the number of parameters as penalty. Smaller value indicate a model that is closer to the (unknowable) true model (caveat: this statement is true only asymptotically, i.e. it holds for large sample sizes). For small samples, we often use BIC (more penalty for complex models when sample size is small), or AICc (as in `MuMIn::model.sel()`).

Non-linear effects

We can use polynomial terms to capture non (log) linear effects:

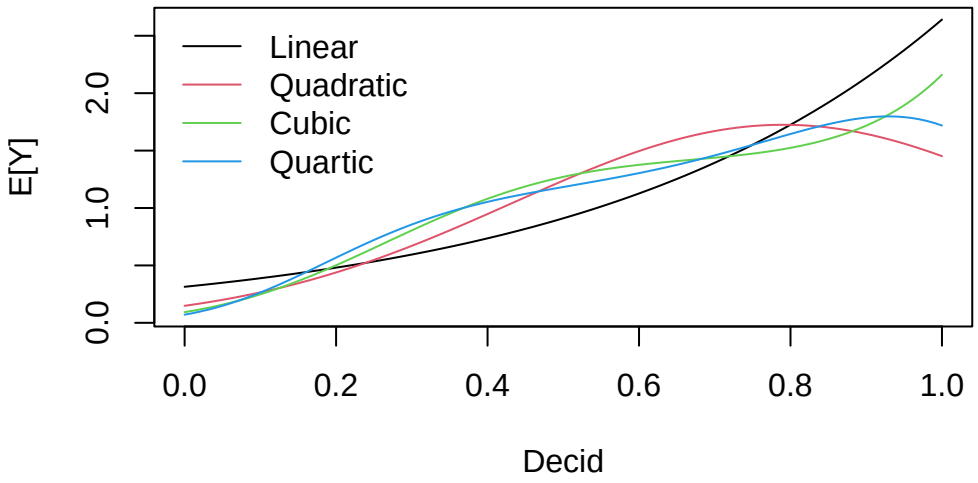
```
mP12 <- glm(Count ~ Decid + I(Decid^2), data = x, family = poisson)
mP13 <- glm(Count ~ Decid + I(Decid^2) + I(Decid^3), data = x, family = poisson)
mP14 <- glm(Count ~ Decid + I(Decid^2) + I(Decid^3) + I(Decid^4), data = x, family = poisson)
MuMIn::model.sel(mP0, mP1, mP12, mP13, mP14)
```

```
Model selection table
     (Intrc)  Decid Decid^2 Decid^3 Decid^4 df    logLik    AICc   delta weight
mP14 -2.6470 16.610 -38.740   41.78  -16.47  5 -5232.629 10475.3    0.00      1
mP13 -2.3730 11.340 -16.250    8.06          4 -5243.979 10496.0   20.70      0
mP12 -1.9090  6.209  -3.927                  3 -5286.517 10579.0  103.77      0
mP1  -1.1590  2.130                          2 -5456.581 10917.2  441.89      0
mP0  -0.1211                                1 -6300.422 12602.8 2127.57      0
Models ranked by AICc(x)
```

Not a surprise that the most complex model won, we had enough degrees of freedoms to spare.

```
xnew <- data.frame(Decid = seq(0, 1, 0.01))

pr <- cbind(
    predict(mP1, xnew, type = "response"),
    predict(mP12, xnew, type = "response"),
    predict(mP13, xnew, type = "response"),
    predict(mP14, xnew, type = "response")
)
matplot(xnew$Decid, pr,
    lty = 1, type = "l",
    xlab = "Decid", ylab = "E[Y]"
)
legend("topleft",
    lty = 1, col = 1:4, bty = "n",
    legend = c("Linear", "Quadratic", "Cubic", "Quartic")
)
```



Categorical variables

Categorical variables are expanded into a *model matrix* before parameter estimation. The model matrix usually contains indicator variables for each level (value 1 when factor value equals a particular label, 0 otherwise) except for the *reference category* (check `relevel` if you want to change the reference category).

The estimate for the reference category comes from the intercept, the rest of the estimates are relative to the reference category. In the log-linear model example this means a ratio.

```
head(model.matrix(~DEC, x))
```

```
        (Intercept) DEC
CL10102           1   1
CL10106           1   0
CL10108           1   0
CL10109           1   1
CL10111           1   1
CL10112           1   1
```

```
mP2 <- glm(Count ~ DEC, data = x, family = poisson)
summary(mP2)
```

```
Call:
glm(formula = Count ~ DEC, family = poisson, data = x)
```

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.85106    0.03070  -27.72   <2e-16 ***
DEC          1.21104    0.03574   33.89   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 7447.9  on 4568  degrees of freedom
Residual deviance: 6123.1  on 4567  degrees of freedom
AIC: 11280

Number of Fisher Scoring iterations: 6
```

```
coef(mP2)
```

```
(Intercept)         DEC
 -0.8510608   1.2110412
```

The estimate for a non-deciduous landscape is e^{β_0}, and it is $e^{\beta_0}e^{\beta_1}$ for deciduous landscapes. (Of course such binary classification at the landscape (1 km^2) level doesn't really makes sense.)

```
MuMIn::model.sel(mP1, mP2)
```

```
Model selection table
    (Intrc) Decid   DEC df    logLik    AICc  delta weight
mP1 -1.1590  2.13        2 -5456.581 10917.2   0.00      1
mP2 -0.8511       1.211  2 -5638.039 11280.1 362.92      0
Models ranked by AICc(x)
```

Having estimates for each land cover type improves the model, but the model using continuous variable is still better

```
mP3 <- glm(Count ~ HAB, data = x, family = poisson)
summary(mP3)
```

```
Call:
```

```
glm(formula = Count ~ HAB, family = poisson, data = x)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.3863     0.5774  -2.401   0.0163 *
HABWater      1.0296     0.6901   1.492   0.1357
HABAgr        0.6931     0.9129   0.759   0.4477
HABUrbInd     0.1335     0.7638   0.175   0.8612
HABRoads    -10.9163   201.2853  -0.054   0.9567
HABDecid      1.7463     0.5776   3.023   0.0025 **
HABOpenWet    0.4220     0.5914   0.714   0.4755
HABConif      0.9214     0.5792   1.591   0.1117
HABConifWet   0.2942     0.5790   0.508   0.6114
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 7447.9  on 4568  degrees of freedom
Residual deviance: 6023.2  on 4560  degrees of freedom
AIC: 11194

Number of Fisher Scoring iterations: 10
```

```
MuMIn::model.sel(mP1, mP2, mP3)
```

```
Model selection table
    (Intrc) Decid   DEC HAB df    logLik    AICc  delta weight
mP1 -1.1590  2.13            2 -5456.581 10917.2   0.00      1
mP3 -1.3860               +  9 -5588.059 11194.2 276.99      0
mP2 -0.8511       1.211       2 -5638.039 11280.1 362.92      0
Models ranked by AICc(x)
```

The prediction in this case would look like: $log(\lambda_i) = \beta_0 + \sum_{j=1}^{k-1} \beta_j x_{ji}$, where we have k factor levels (and $k-1$ indicator variables besides the intercept).

Here is a general way of calculating fitted values or making predictions based on the design matrix (`X`) and the coefficients (`b`) (column ordering in `X` must match the elements in `b`) given a parametric log-linear model `object` and data frame `df` (the code won't run as is, `object` is just a placeholder for your GLM model object):

```
b <- coef(object)
X <- model.matrix(formula(object), df)
exp(X %*% b)
```

Multiple main effects

We can add main effects by providing a new formula to a new model. Or we can use the `update()` function. The `. ~ .` means to keep both sides of the formula but add (`+`) or remove (`-`) terms from the original scope. Note: the `update` function can be used to update any function call, not just GLM models.

```
# mP4 <- glm(Count ~ Decid + ConifWet, data=x, family=poisson)
mP4 <- update(mP1, . ~ . + ConifWet)
summary(mP4)
```

```
Call:
glm(formula = Count ~ Decid + ConifWet, family = poisson, data = x)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.69255    0.05547 -12.485   <2e-16 ***
Decid        1.61480    0.07169  22.525   <2e-16 ***
ConifWet    -0.98628    0.09906  -9.957   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 7447.9  on 4568  degrees of freedom
Residual deviance: 5659.7  on 4566  degrees of freedom
AIC: 10819

Number of Fisher Scoring iterations: 6
```

```
MuMIn::model.sel(mP0, mP1, mP4)
```

```
Model selection table
    (Intrc) Decid   CnfWt df    logLik    AICc   delta weight
mP4 -0.6925 1.615 -0.9863  3 -5406.310 10818.6    0.00      1
```

```
mP1 -1.1590 2.130          2 -5456.581 10917.2   98.54      0
mP0 -0.1211                1 -6300.422 12602.8 1784.22      0
Models ranked by AICc(x)
```

Here are some functions that can automate model selection:

- `drop1`: evaluate which variable to drop to lower AIC the most
- `add1`: evaluate which variable to add to lower AIC the most
- `step`: perform forward/backward model selection using `add1`/`drop1`

Interactions

When we consider interactions between two variables (say x_1 and x_2), we refer to adding another variable to the model matrix that is a product of the two variables ($x_{12} = x_1 x_2$):

```
head(model.matrix(~ x1 * x2, data.frame(x1 = 1:4, x2 = 10:7)))
```

```
  (Intercept) x1 x2 x1:x2
1           1  1 10    10
2           1  2  9    18
3           1  3  8    24
4           1  4  7    28
```

Let's consider interaction between our two predictors from before:

```
mP5 <- glm(Count ~ Decid * ConifWet, data = x, family = poisson)
summary(mP5)
```

```
Call:
glm(formula = Count ~ Decid * ConifWet, family = poisson, data = x)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    -0.55157    0.05643  -9.775   <2e-16 ***
Decid           1.20486    0.07799  15.448   <2e-16 ***
ConifWet       -2.32015    0.14875 -15.598   <2e-16 ***
Decid:ConifWet  5.34691    0.35615  15.013   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 7447.9  on 4568  degrees of freedom
Residual deviance: 5415.8  on 4565  degrees of freedom
AIC: 10577

Number of Fisher Scoring iterations: 6
```

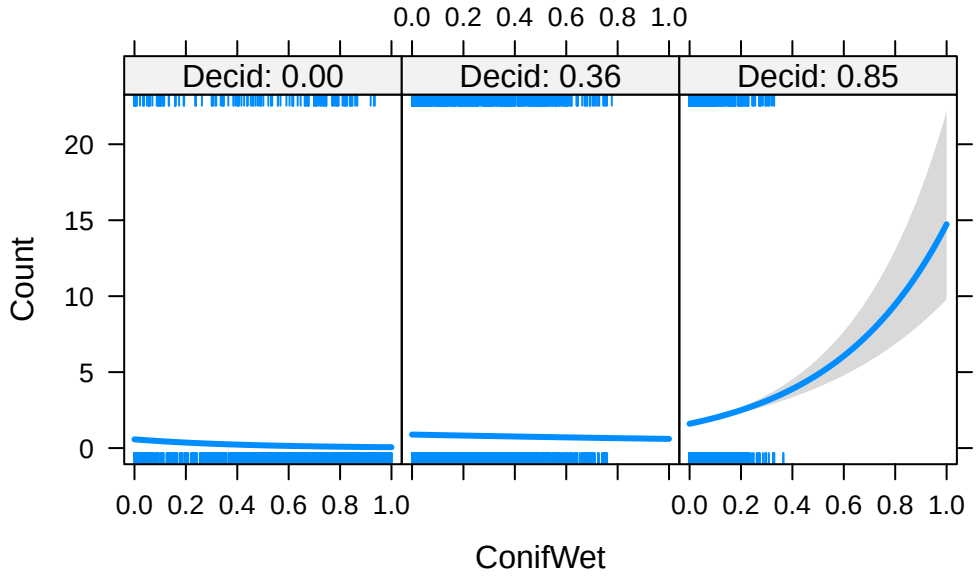
```
MuMIn::model.sel(mP0, mP1, mP4, mP5)
```

```
Model selection table
      (Int)   Dcd     CnW CnW:Dcd df    logLik    AICc   delta weight
mP5 -0.5516 1.205 -2.3200   5.347  4 -5284.389 10576.8    0.00      1
mP4 -0.6925 1.615 -0.9863          3 -5406.310 10818.6  241.84      0
mP1 -1.1590 2.130                  2 -5456.581 10917.2  340.38      0
mP0 -0.1211                        1 -6300.422 12602.8 2026.06      0
Models ranked by AICc(x)
```

The model with the interaction is best supported, but how do we make sense of this relationship? We can't easily visualize it in a single plot. We can either

1. fix all variables (at their mean/meadian) and see how the response is changing along a single variable: this is called a *conditional* effect (conditional on fixing other variables), this is what `visreg::visreg()` does
2. or plot the fitted values against the predictor variable (one at a time), this is called a *marginal* effect, and this is what `ResourceSelection::mep()` does

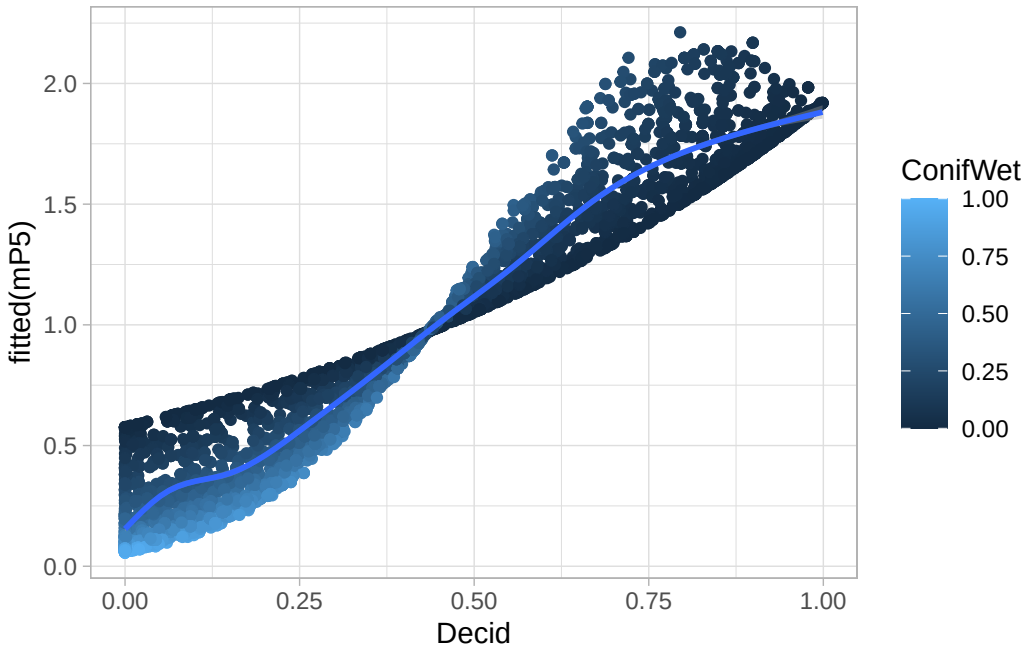
```
visreg::visreg(mP5, scale = "response", xvar = "ConifWet", by = "Decid")
```




```
ggplot(x, aes(x = Decid, y = fitted(mP5), col = ConifWet)) +
    geom_point() +
    geom_smooth() +
    theme_light()
```

```
`geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```

```
Warning: The following aesthetics were dropped during statistical transformation:
colour.
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?
```



Final battle of Poisson models:

```
MuMIn::model.sel(mP0, mP1, mP12, mP13, mP14, mP2, mP3, mP4, mP5)
```

```
Model selection table
       (Int)    Dcd   Dcd^2 Dcd^3  Dcd^4   DEC HAB     CnW CnW:Dcd df    logLik
mP14 -2.6470 16.610 -38.740 41.78 -16.47                            5 -5232.629
mP13 -2.3730 11.340 -16.250  8.06                                   4 -5243.979
mP5  -0.5516  1.205                                -2.3200   5.347  4 -5284.389
mP12 -1.9090  6.209  -3.927                                         3 -5286.517
mP4  -0.6925  1.615                                -0.9863          3 -5406.310
mP1  -1.1590  2.130                                                 2 -5456.581
mP3  -1.3860                                    +                   9 -5588.059
mP2  -0.8511                           1.211                        2 -5638.039
mP0  -0.1211                                                        1 -6300.422
        AICc   delta weight
mP14 10475.3    0.00      1
mP13 10496.0   20.70      0
mP5  10576.8  101.52      0
mP12 10579.0  103.77      0
mP4  10818.6  343.35      0
mP1  10917.2  441.89      0
mP3  11194.2  718.89      0
mP2  11280.1  804.81      0
```

```
mP0  12602.8 2127.57      0
Models ranked by AICc(x)
```

Of course, the most complex model wins but the Chi-square test is still significant (indicating lack of fit). Let's try different error distribution.

Different error distributions

We will use the 2-variable model with interaction:

```
mP <- glm(Count ~ Decid * ConifWet, data = x, family = poisson)
```

Let us try the Negative Binomial distribution first. This distribution is related to Binomial experiments (number of trials required to get a fixed number of successes given a binomial probability). It can also be derived as a mixture of Poisson and Gamma distributions (see Wikipedia), which is a kind of hierarchical model. In this case, the Gamma distribution acts as an i.i.d. random effect for the intercept: $Y_i \sim Poisson(\lambda_i)$, $\lambda_i \sim Gamma(e^{\beta_0+\beta_1 x_{1i}}, \gamma)$, where γ is the Gamma variance.

The Negative Binomial variance (using the parametrization common in R functions) is a function of the mean and the scale: $V(\mu) = \mu + \mu^2/\theta$.

```
mNB <- MASS::glm.nb(Count ~ Decid * ConifWet, data = x)
summary(mNB)
```

```
Call:
MASS::glm.nb(formula = Count ~ Decid * ConifWet, data = x, init.theta = 3.524756181,
    link = log)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    -0.58101    0.06291  -9.236   <2e-16 ***
Decid           1.23695    0.08919  13.868   <2e-16 ***
ConifWet       -2.36485    0.16040 -14.743   <2e-16 ***
Decid:ConifWet  5.70380    0.40132  14.212   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Negative Binomial(3.5248) family taken to be 1)
```

```
    Null deviance: 6086.2  on 4568  degrees of freedom
Residual deviance: 4388.4  on 4565  degrees of freedom
AIC: 10464

Number of Fisher Scoring iterations: 1

              Theta:  3.525
          Std. Err.:  0.411

 2 x log-likelihood:  -10453.887
```

Next, we look at zero-inflated models. In this case, the mixture distribution is a Bernoulli distribution and a count distribution (Poisson or Negative Binomial, for example). The 0's can come from both the zero and the count distributions, whereas the >0 values can only come from the count distribution: $A_i \sim Bernoulli(\varphi)$, $Y_i \sim Poisson(A_i\lambda_i)$.

The zero part of the zero-inflated models are often parametrized as probability of zero $(1-\varphi)$, as in the `pscl::zeroinfl` function:

```
## Zero-inflated Poisson
mZIP <- pscl::zeroinfl(Count ~ Decid * ConifWet | 1, x, dist = "poisson")
summary(mZIP)
```

```
Call:
pscl::zeroinfl(formula = Count ~ Decid * ConifWet | 1, data = x, dist = "poisson")

Pearson residuals:
    Min      1Q  Median      3Q     Max
-1.2160 -0.7011 -0.3391  0.3766  8.9394

Count model coefficients (poisson with log link):
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    -0.31127    0.06503  -4.787 1.69e-06 ***
Decid           1.05891    0.08588  12.330  < 2e-16 ***
ConifWet       -2.45044    0.15622 -15.686  < 2e-16 ***
Decid:ConifWet  5.94422    0.38368  15.493  < 2e-16 ***

Zero-inflation model coefficients (binomial with logit link):
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.5989     0.1021  -15.65   <2e-16 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Number of iterations in BFGS optimization: 12
Log-likelihood: -5218 on 5 Df
```

```
## Zero-inflated Negative Binomial
mZINB <- pscl::zeroinfl(Count ~ Decid * ConifWet | 1, x, dist = "negbin")
summary(mZINB)
```

```
Call:
pscl::zeroinfl(formula = Count ~ Decid * ConifWet | 1, data = x, dist = "negbin")

Pearson residuals:
    Min      1Q  Median      3Q     Max
-1.1874 -0.6902 -0.3375  0.3616  8.9455

Count model coefficients (negbin with log link):
               Estimate Std. Error z value Pr(>|z|)
(Intercept)    -0.37625    0.07312  -5.146 2.66e-07 ***
Decid           1.10981    0.09105  12.189  < 2e-16 ***
ConifWet       -2.43272    0.15872 -15.327  < 2e-16 ***
Decid:ConifWet  5.93085    0.39641  14.962  < 2e-16 ***
Log(theta)      2.61874    0.51590   5.076 3.85e-07 ***

Zero-inflation model coefficients (binomial with logit link):
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.8513     0.1917  -9.655   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Theta = 13.7184
Number of iterations in BFGS optimization: 22
Log-likelihood: -5215 on 6 Df
```

Now we compare the four different parametric models:

```
AIC(mP, mNB, mZIP, mZINB)
```

```
      df      AIC
mP     4 10576.78
```

```
mNB    5 10463.89
mZIP   5 10445.40
mZINB  6 10442.56
```

```
MuMIn::model.sel(mP, mNB, mZIP, mZINB)
```

```
Model selection table
        (Int)    CnW   Dcd CnW:Dcd cnt_(Int) cnt_CnW cnt_Dcd cnt_CnW:Dcd
mZINB                                -0.3762  -2.433   1.110           +
mZIP                                 -0.3113  -2.450   1.059           +
mNB   -0.5810 -2.365 1.237   5.704
mP    -0.5516 -2.320 1.205   5.347
      zer_(Int)         family    class init.theta link dist df    logLik
mZINB    -1.851         b(lgt) zeroinfl                 ngbn  6 -5215.281
mZIP     -1.599         b(lgt) zeroinfl                 pssn  5 -5217.698
mNB             NB(3.5248,log)   negbin       3.52  log       5 -5226.943
mP                      p(log)      glm                        4 -5284.389
         AICc  delta weight
mZINB 10442.6   0.00  0.804
mZIP  10445.4   2.83  0.196
mNB   10463.9  21.32  0.000
mP    10576.8 134.21  0.000
Abbreviations:
 family: b(lgt) = 'binomial(logit)',
         NB(3.5248,log) = 'Negative Binomial(3.5248,log)',
         p(log) = 'poisson(log)'
 dist:   ngbn = 'negbin', pssn = 'poisson'
Models ranked by AICc(x)
```

Our best model is the ZINB. The probability of observing a zero as part of the zero distribution is back transformed from the zero coefficient using the inverse logit function:

```
unname(plogis(coef(mZIP, "zero"))) # P of 0 (not 1!)
```

```
[1] 0.1681348
```

Mixed models

It is also common practice to consider generalized linear mixed models (GLMMs) for count data. These mixed models are usually considered as Poisson-Lognormal mixtures. The simplest, so

called i.i.d., case is similar to the Negative Binomial, but instead of Gamma, we have Lognormal distribution: $Y_i \sim Poisson(\lambda_i)$, $log(\lambda_i) = \beta_0 + \beta_1 x_{1i} + \epsilon_i$, $\epsilon_i \sim Normal(0, \sigma^2)$, where σ^2 is the Lognormal variance on the log scale.

We can use the `lme4::glmer` function: use `SiteID` as random effect (we have exactly n random effects).

```
mPLN1 <- lme4::glmer(Count ~ Decid * ConifWet + (1 | SiteID),
    data = data.frame(SiteID = rownames(x), x), family = poisson
)
summary(mPLN1)
```

```
Generalized linear mixed model fit by maximum likelihood (Laplace
  Approximation) [glmerMod]
 Family: poisson  ( log )
Formula: Count ~ Decid * ConifWet + (1 | SiteID)
   Data: data.frame(SiteID = rownames(x), x)

     AIC      BIC   logLik deviance df.resid
 10445.9  10478.0  -5218.0  10435.9     4564

Scaled residuals:
    Min      1Q  Median      3Q     Max
-1.1475 -0.6291 -0.2855  0.4156  5.4234

Random effects:
 Groups Name        Variance Std.Dev.
 SiteID (Intercept) 0.2988   0.5466
Number of obs: 4569, groups:  SiteID, 4569

Fixed effects:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)    -0.74536    0.06743  -11.05   <2e-16 ***
Decid           1.27742    0.09201   13.88   <2e-16 ***
ConifWet       -2.34745    0.16248  -14.45   <2e-16 ***
Decid:ConifWet  5.63872    0.41229   13.68   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
            (Intr) Decid  ConfWt
Decid       -0.895
ConifWet    -0.621  0.644
```

```
Decid:CnfWt  0.176 -0.379 -0.700
```

Note: the number of unknowns we have to somehow estimate is now more than the number of observations we have. How is that possible?

Alternatively, we can use `SurveyArea` as a grouping variable. We have now $m < n$ random effects, and survey areas can be seen as larger landscapes within which the sites are clustered: $Y_i j \sim Poisson(\lambda_i j)$, $log(\lambda_i j) = \beta_0 + \beta_1 x_{1ij} + \epsilon_i$, $\epsilon_i \sim Normal(0, \sigma^2)$. The index i $(i = 1, ..., m)$ defines the cluster (survey area), the j $(j = 1, ..., n_i)$ defines the sites within survey area i $(n = \sum_{i=1}^{m} n_i)$.

```r
mPLN2 <- lme4::glmer(Count ~ Decid * ConifWet + (1 | SurveyArea),
    data = data.frame(SurveyArea = detect::josm$surveys$SurveyArea, x),
    family = poisson
)
summary(mPLN2)
```

```
Generalized linear mixed model fit by maximum likelihood (Laplace
  Approximation) [glmerMod]
 Family: poisson  ( log )
Formula: Count ~ Decid * ConifWet + (1 | SurveyArea)
   Data: data.frame(SurveyArea = detect::josm$surveys$SurveyArea, x)

     AIC      BIC   logLik deviance df.resid
 10047.2  10079.3  -5018.6  10037.2     4564

Scaled residuals:
    Min      1Q  Median      3Q     Max
-1.7379 -0.6434 -0.3200  0.3580  6.4556

Random effects:
 Groups     Name        Variance Std.Dev.
 SurveyArea (Intercept) 0.2934   0.5417
Number of obs: 4569, groups:  SurveyArea, 271

Fixed effects:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)    -0.73773    0.07807   -9.45   <2e-16 ***
Decid           1.19195    0.09820   12.14   <2e-16 ***
ConifWet       -2.32770    0.16824  -13.84   <2e-16 ***
Decid:ConifWet  5.53058    0.39698   13.93   <2e-16 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
            (Intr) Decid  ConfWt
Decid       -0.808
ConifWet    -0.609  0.628
Decid:CnfWt  0.162 -0.325 -0.670
```

In the battle of distributions (keeping the linear predictor part the same) the clustered GLMM was best supported:

```
tmp <- AIC(mP, mNB, mZIP, mZINB, mPLN1, mPLN2)
tmp$delta_AIC <- tmp$AIC - min(tmp$AIC)
tmp[order(tmp$AIC), ]
```

```
      df      AIC delta_AIC
mPLN2  5 10047.19    0.0000
mZINB  6 10442.56  395.3703
mZIP   5 10445.40  398.2047
mPLN1  5 10445.91  398.7171
mNB    5 10463.89  416.6952
mP     4 10576.78  529.5872
```

Next

Naïve estimates of occupancy and abundance