Model evaluation and analysis: the *modEvA* R package in a nutshell

A. Marcia Barbosa, CIBIO/InBIO - University of Evora (Portugal),
barbosa@uevora.pt

updated 7 May 2015

modEvA is an R package for **analysing and evaluating species distribution models**. Most functions are meant for generalized linear models (GLMs) with binomial distribution and a logit link function (i.e., logistic regression), although many can be applied to other models as well. Most functions can also be applied to values of observed and corresponding predicted values, as long as they are bounded between 0 and 1.

Installing and loading *modEvA*

The ***modEvA*** package works within the free and open-source R statistical software, so you first need to **download, install and open R** (available at http://www.r-project.org). In this tutorial, in monospaced font are the commands that you need to type (or copy and paste) into the R console (and then press the *enter* key to execute them). For commands that generate visible results in R, these are usually shown below them, preceded by hash marks (##). Note that all **commands are case-sensitive**, so you must respect upper- and lower-case letters; that you must always use **straight** (', ") **rather than curly quotes and apostrophes**; and that **R is only ready to receive a new command when there's a prompt sign (>) at the end of the R console**; if not, it's still waiting for an operation to be finished or for you to complete a previous command -- watch out for unclosed parentheses and such.

Install *modEvA* by pasting the command below in the R console (when connected to the internet):

```
install.packages("modEvA", repos = "http://R-Forge.R-project.org")
```

This should work if you have the **latest version of R**; otherwise, it may either fail (producing a message like "*package* '*modEvA*' *is not available for your R version*") or show a warning and install an older version of *modEvA*. To **check the version that you have actually installed**, type citation(package="modEvA"). To install the latest version of the package, you can either upgrade R *or* download the compressed *modEvA* package **source files** to your disk -- *.zip* for Windows or *.tar.gz* for Linux and Mac, available at the package development page or at this Dropbox folder and then install the

package locally, e.g. with R menu "*Packages - Install packages from local zip files*" (Windows), or "*Packages & Data - Package installer, Packages repository - Local source package*" (Mac), or "*Tools - Install packages - Install from: Package Archive File*" (RStudio).

You only need to install the package once (unless a new version becomes available), but you need to **load it every time you open a new R session** in which you intend to use *modEvA* (no need for an internet connection anymore), by pasting the following command in R:

```
library(modEvA)
```

Analysing models

Load the *rotif.mods* sample dataset that comes with the *modEvA* package, to use as an example:

```
data(rotif.mods)
```

You can **get more information on this dataset** (the following command should open an R Documentation window):

```
help(rotif.mods)
```

You can see that *rotif.mods* is a list containing two elements: a list of *models* and a dataframe with their *predictions*. Let's leave the predictions alone for now, and work on the model objects. Let's start by checking out their names:

```
names(rotif.mods$models)
```

```
##  [1] "Abrigh" "Afissa" "Apriod" "Bangul" "Bcalyc" "Bplica" "Bquadr"
##  [8] "Burceo" "Cgibba" "Edilat" "Flongi" "Kcochl" "Kquadr" "Ktropi"
## [15] "Lbulla" "Lclost" "Lhamat" "Lluna"  "Llunar" "Lovali" "Lpatel"
## [22] "Lquadr" "Mventr" "Ppatul" "Pquadr" "Pvulga" "Specti" "Tpatin"
## [29] "Tsimil" "Ttetra"
```

These names correspond to species codes, and each object in this list is a GLM of the presence-absence of the corresponding species. Now let's **assign the first model to an individual object**, and use it **to try out *modEvA* functions**. The following alternate commands should both create a model object named "mod" from the 1st model in the *rotif.mods$models* list, which is named "Abrigh":

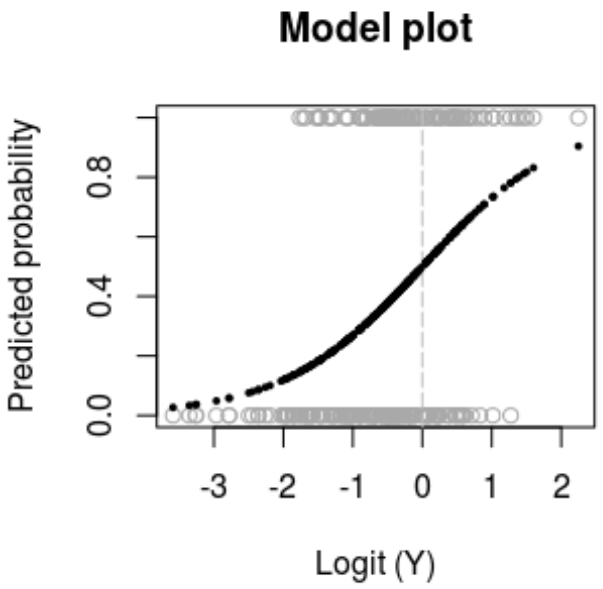
```
mod <- rotif.mods$models[[1]]
mod <- rotif.mods$models[["Abrigh"]]
```

In most *modEvA* functions, instead of a model object, you can provide observed and predicted values that you've obtained elswehere. Let's extract such values from this model, to be able to use them in examples as well:

```
mydata <- data.frame(observed = mod$y, predicted = mod$fitted.values)
```

Let's now try *modEvA* functions on this model's predictions (or you can use any other GLM object or observed-predicted values that you may have). Let's start with function ***plotGLM***, which shows **how observed (grey) and predicted (black) values vary** along the regression equation:

```
plotGLM(model = mod, xlab = "Logit (Y)", ylab = "Predicted probability", main = "Model plot")
```



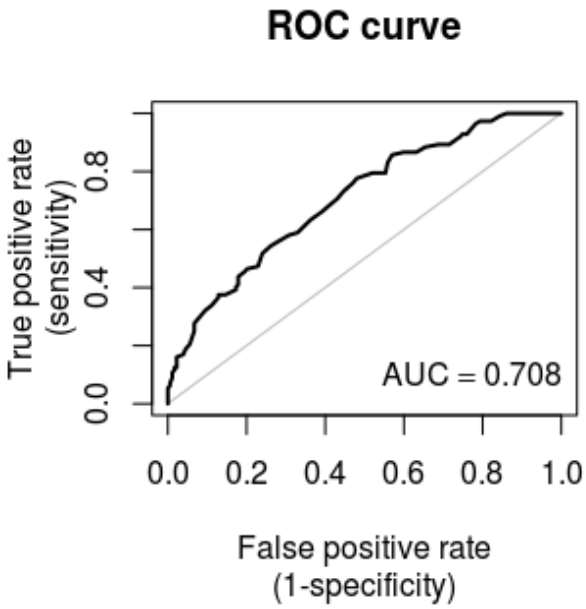
Instead of the *model* object, you can provide your **observed and predicted values**:

```
plotGLM(obs = mydata[ , "observed"], pred = mydata[ , "predicted"], xlab = "Logit (Y)", ylab = "Predicted probability", main = "Model plot")
```

Evaluating discrimination capacity

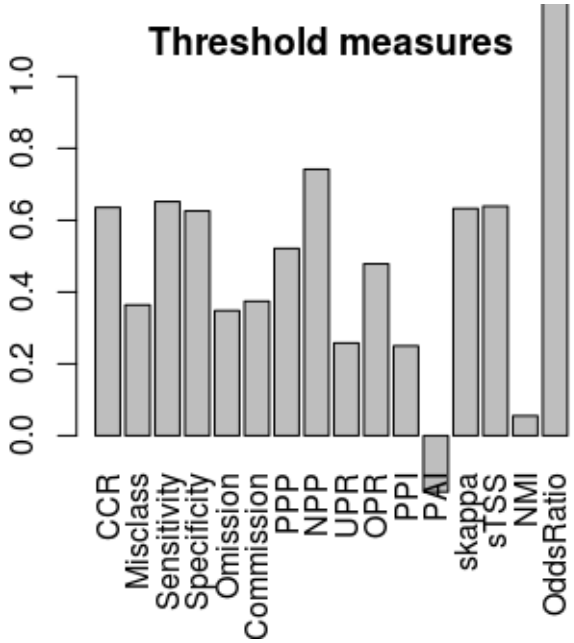
Let's now **calculate the area under the ROC curve (AUC)** for this model. The *AUC* function produces the plot displayed below and also some text results, which will appear in your R console but will not be shown here. Again, as in most *modEvA* functions, you can provide *obs = mydata[, "observed"], pred = mydata[, "predicted"]* instead of *model = mod*:

```
AUC(model = mod)
```



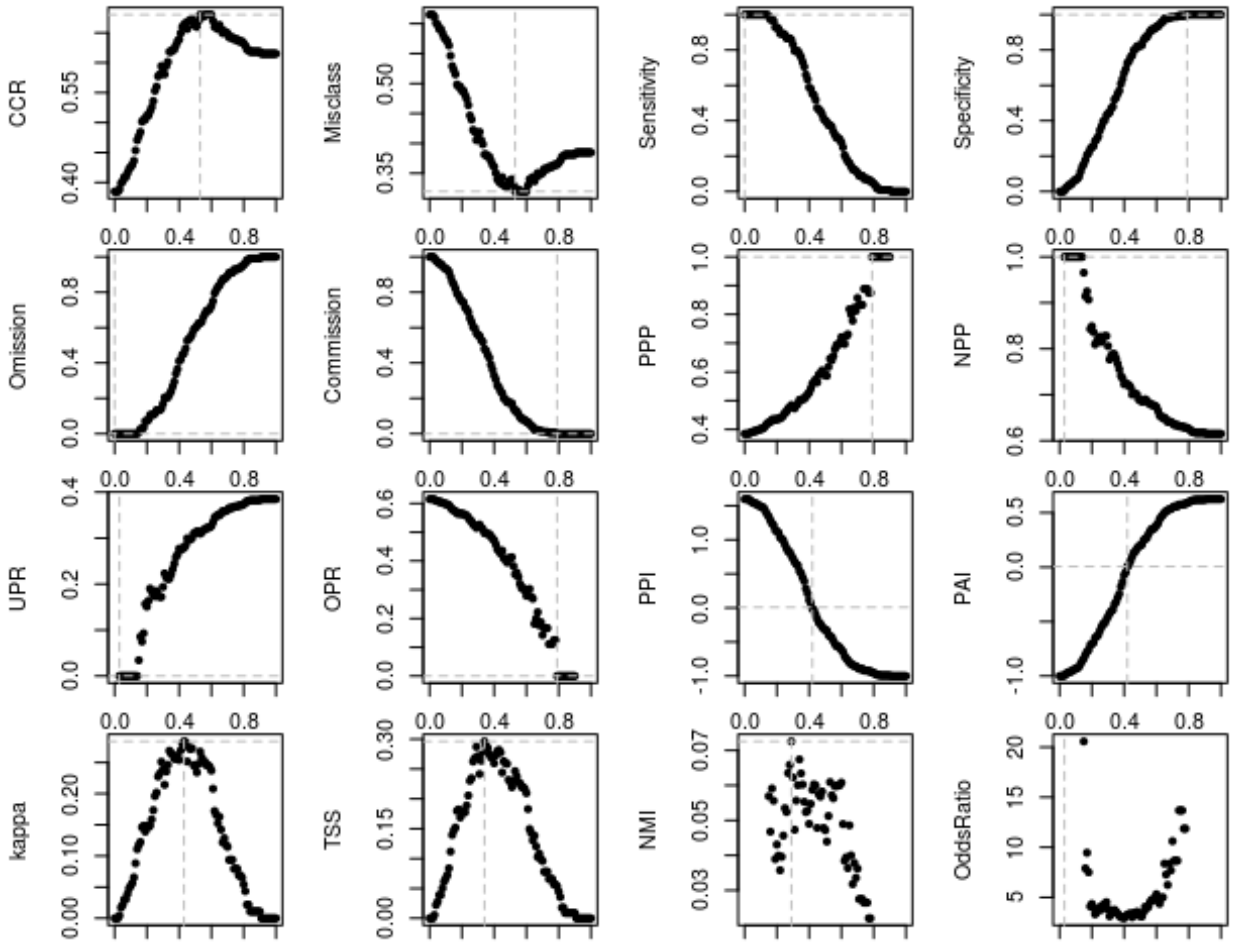
Now **calculate some threshold-based evaluation measures** for this model, using the species' prevalence (proportion of presences) as the threshold value above which to consider that the model predicts the species to be present. Only the plot is shown here, but additional text results should appear in your R console as well:

```
par(mar = c(5.6, 4.1, 2, 2.1))  # changes figure margins
threshMeasures(model = mod, thresh = "preval", ylim = c(0, 1), main = "Threshold measures")
```



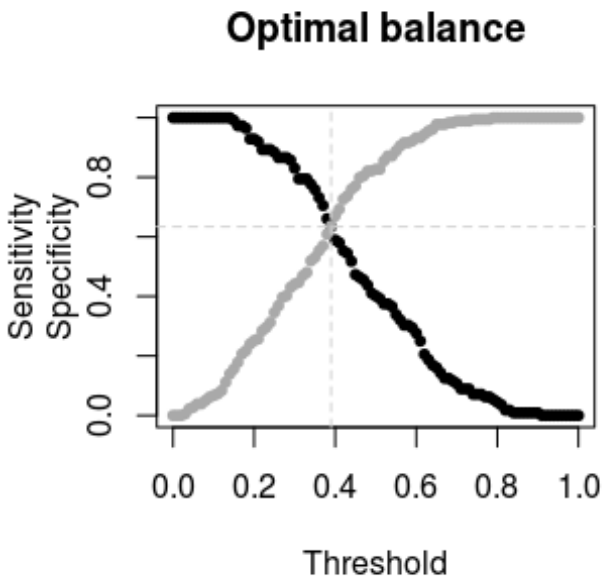
Note that the *thresh*old value used above was *preval*ence, which is adequate when your predicted values are of probability. If, however, you are **evaluating *favourability* predictions** (see e.g. this other tutorial on favourability modelling), **prevalence is not an adequate threshold: 0.5 is its equivalent**. But there are many **different criteria for choosing the threshold** value for a model (see Details in help(threshMeasures)). Let's **see how each threshold-based measure varies** along **with** the chosen prediction **threshold**, to maybe **identify optimal thresholds** according to particular criteria:

```
optiThresh(model = mod, pch = 20)
```



You can also calculate the **optimal threshold balancing two complementary evaluation measures**:

```
optiPair(model = mod, measures = c("Sensitivity", "Specificity"), main = "Optimal balance")
```



You can try the command above with other pairs of related *measures*, such as c("Omission", "Commission"), c("PPI", "PAI"), etc..

Assessing model explanatory power, fit and calibration

Let's now assess the **proportion of variation that the model accounts for**. For GLMs there isn't a single consensual measure for this; *modEvA* can calculate the **explained deviance** (*D-squared*), optionally adjusted for the number of observations and parameters; and some **pseudo R-squared** values (see help(Dsquared)and help(RsqGLM) for further info):

```
Dsquared(model = mod)
```

```
## [1] 0.1114199
```

```
Dsquared(model = mod, adjust = TRUE)
```

```
## [1] 0.09264705
```

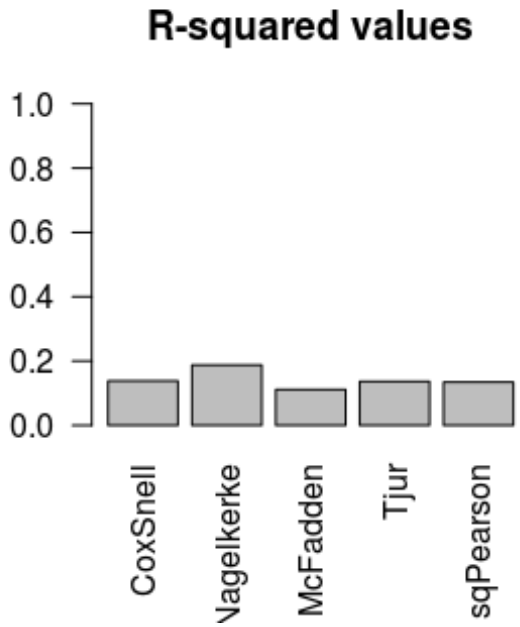
```
RsqGLM(model = mod)
```

```
## $CoxSnell
## [1] 0.1380008
##
## $Nagelkerke
## [1] 0.187434
##
## $McFadden
```

```
## [1] 0.1114199
##
## $Tjur
## [1] 0.1365661
##
## $sqPearson
## [1] 0.134168
```

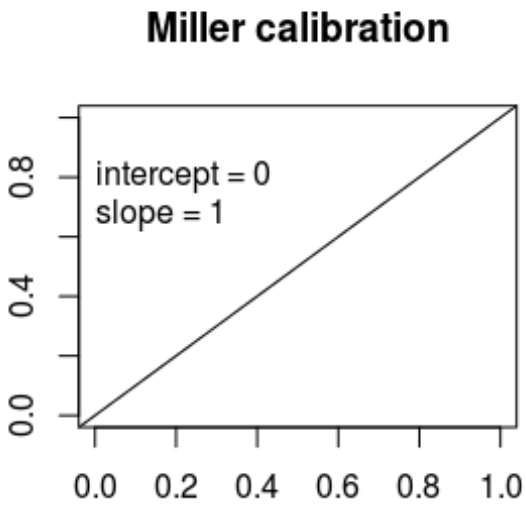
You can **visualise these R-squared measures** with the *barplot* R function:

```
barplot(unlist(RsqGLM(model = mod)), ylim = c(0, 1), las = 2, main = "R-squared values")
```



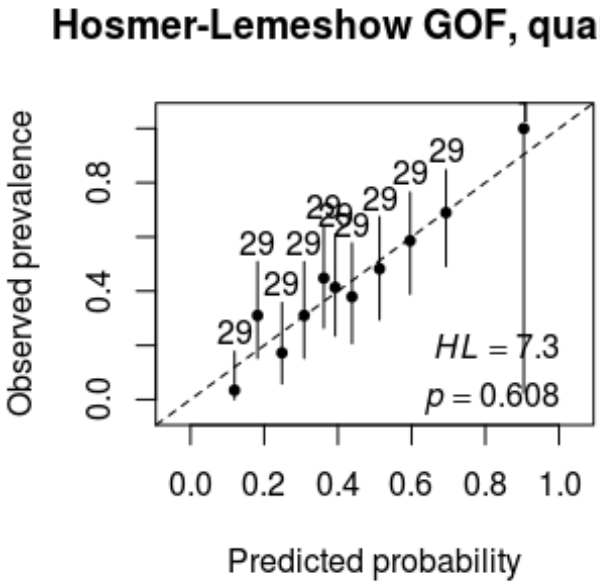
We can also take a look at some model calibration measures, such as **Miller's calibration statistics** and the **Hosmer-Lemeshow goodness-of-fit**. Note, however, that the former is not useful for evaluating a model on the same data used for building it (the results will always look good); and that the latter can depend strongly on the *bin.method* used for grouping the values. See help(MillerCalib) and help(HLfit) to find out how these measures are calculated and how their results can be interpreted.

```
MillerCalib(model = mod)
```



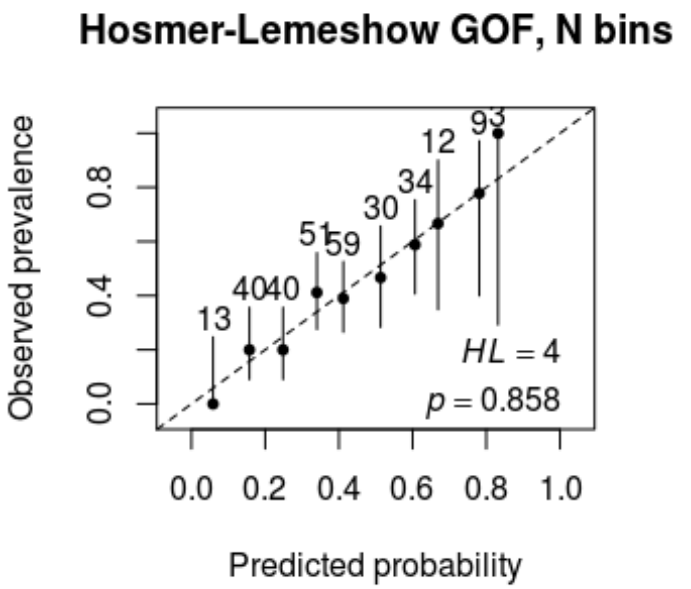

```
## $intercept
## [1] 1.799596e-13
##
## $slope
## [1] 1
```

```
HLfit(model = mod, bin.method = "quantiles", main = "Hosmer-Lemeshow GOF,
quantiles")
```



```
## $bins.table
##    BinCenter NBin     BinObs   BinPred BinObsCIlower BinObsCIupper
## 1  0.1194811   29 0.03448276 0.1016246  0.0008726469     0.1776443
## 2  0.1819756   29 0.31034483 0.1789304  0.1528459396     0.5083234
## 3  0.2482370   29 0.17241379 0.2445406  0.0584560830     0.3577476
## 4  0.3084812   29 0.31034483 0.3091572  0.1528459396     0.5083234
## 5  0.3618102   29 0.44827586 0.3607436  0.2644553037     0.6430613
## 6  0.3927488   29 0.41379310 0.3942488  0.2352402098     0.6106372
## 7  0.4383741   29 0.37931034 0.4372018  0.2068686995     0.5773954
## 8  0.5130304   29 0.48275862 0.5063462  0.2944855830     0.6746850
## 9  0.5960986   29 0.58620690 0.5910230  0.3893627914     0.7647598
## 10 0.6933976   29 0.68965517 0.7070658  0.4916766462     0.8471541
## 11 0.9044237    1 1.00000000 0.9044237  0.0250000000     1.0000000
##
## $chi.sq
## [1] 7.278077
##
## $DF
## [1] 9
##
## $p.value
## [1] 0.6081922
```

```
HLfit(model = mod, bin.method = "n.bins", main = "Hosmer-Lemeshow GOF, N bins")
```




```
## $bins.table
##                  BinCenter NBin    BinObs    BinPred BinObsCIlower
## (0.026,0.114] 0.05859488   13 0.0000000 0.06339589    0.00000000
```

```
## (0.114,0.202] 0.15729121   40 0.2000000 0.15678701    0.09052241
## (0.202,0.29]  0.24840042   40 0.2000000 0.24634002    0.09052241
## (0.29,0.378]  0.34035838   51 0.4117647 0.33974064    0.27584296
## (0.378,0.466] 0.41206526   59 0.3898305 0.41510104    0.26549147
## (0.466,0.554] 0.51322862   30 0.4666667 0.50772771    0.28341808
## (0.554,0.642] 0.60663508   34 0.5882353 0.60041852    0.40696943
## (0.642,0.729] 0.66899210   12 0.6666667 0.67465548    0.34887551
## (0.729,0.817] 0.78101033    9 0.7777778 0.77077823    0.39990643
## (0.817,0.905] 0.83199370    3 1.0000000 0.85136910    0.29240177
##             BinObsCIupper
## (0.026,0.114]     0.2470526
## (0.114,0.202]     0.3564780
## (0.202,0.29]      0.3564780
## (0.29,0.378]      0.5583072
## (0.378,0.466]     0.5256215
## (0.466,0.554]     0.6567448
## (0.554,0.642]     0.7535293
## (0.642,0.729]     0.9007539
## (0.729,0.817]     0.9718550
## (0.817,0.905]     1.0000000
##
## $chi.sq
## [1] 3.995296
##
## $DF
## [1] 8
##
## $p.value
## [1] 0.8575476
```

Evaluating multiple models

You can calulate a set of **evaluation measures for several models simultaneously**, if you have them in a list like *rotif.mods$models* (results not shown here):

```
model_eval <- multModEv(models = rotif.mods$models, thresh = "preval", bin.method = "quantiles")
model_eval
```

With the *multModEv* function, if you want to use observed and predicted values rather than a list of model objects, get all your values in a table and call them by their column index numbers -- e.g., if your observed data are in columns 2 to 8 and your predicted data are in columns 9 to 15 (note that species must be **in the same order** on both *observed* and *predicted* columns!), you can use this command:

```
multModEv(obs.data = mydata[ , 2:8], pred.data = mydata[ , 9:15], thresh = "preval")
```

If you've successfully executed any of the two commands above, you can save these results to a file on your disk. The following command will save a comma-separated values (CSV) file in your working directory (type getwd() to find out where it is):

```
write.csv(model_eval, file = "model_eval.csv", row.names = FALSE)
```

Additional options

Some other potentially useful *modEvA* functions are (still) not covered in this tutorial, but you can find out about them as well:

```
help(evenness)
help(prevalence)
help(OA)
help(MESS)
help(varPart)
```

You can find out additional options and further info on any function with *help(function.name)* (e.g., help(multModEv)).

That's it! You can e-mail me with any suggestions or concerns, but first remember to check for updates to the package or this tutorial at http://modEvA.r-forge.r-project.org. This tutorial was built with *RStudio* + *rmarkdown* + *knitr*. Thanks!