Using the Pipeline

Tom August

9 July 2015

Introduction

This document shows how to use the indicator pipeline to create biodiversity indicators such as those for DEFRA's Biodiversity Indicators in Your Pocket. The pipeline is shared in the form of an R package called 'BRCindicators' making it easy to share and maintain.

The functions in BRCindicators work with yearly estimates of species abundance or occurrence and aggregate them into an scaled indicator value with bootstrapped confidence intervals

This package has the ability to read in the output of occupancy models created in the R package sparta, a package for estimating species trends from occurrence data. This package can be installed from Github and details of how to use the package are given in the package vignette. There is no need to use sparta to create your yearly species estimates as BRCindicators can also work with other data.

To create an indicator we first need to have species trends, let's create some using the sparta R package.

Creating yearly estimates of occurrence in sparta

If you already have yearly estimates of abundance or occurrence for your species you can skip this stage. Here we show how you can create these estimates from raw species observation data using sparta.

```
# Install the package from CRAN
# THIS WILL WORK ONLY AFTER THE PACKAGE IS PUBLISHED
install.packages('sparta')

# Or install the development version from GitHub
library(devtools)
install_github('biologicalrecordscentre/sparta')
```

Let's assume you have some raw data already, we can under take occupancy modelling like this

```
# Load the sparta package
library(sparta)
```

```
## Loading required package: lme4
## Loading required package: Matrix
## Loading required package: Rcpp
```

For demonstration purposes I have a faked dataset of 8000 species observations. In my dataset the species are named after the letters in the alphabet. Below I show how I can use the Bayesian occupancy models in sparta to create yearly estimates of occurrence. For more information please see the vignette for sparta

```
# Preview of my data
head(myData)
```

```
##   taxa site time_period
## 1    r  A51  1970-01-14
## 2    v  A87  1980-09-29
## 3    e  A56  1996-04-14
## 4    z  A28  1959-01-16
## 5    r  A77  1970-09-21
## 6    x  A48  1990-02-25
```

```
# First format our data
formattedOccData <- formatOccData(taxa = myData$taxa,
                                  site = myData$site,
                                  time_period = myData$time_period)
```

```
## Warning in errorChecks(taxa = taxa, site = site, time_period =
## time_period): 94 out of 8000 observations will be removed as duplicates
```

```
# Here we are going to use the package snowfall to parallelise
library(snowfall)
```

```
## Loading required package: snow
```

```
# I have 4 cpus on my PC so I set cpus to 4
# when I initialise the cluster
sfInit(parallel = TRUE, cpus = 4)
```

```
## Warning in searchCommandline(parallel, cpus = cpus, type = type,
## socketHosts = socketHosts, : Unknown option on commandline:
## rmarkdown::render('W:/PYWELL_SHARED/Pywell Projects/BRC/Tom August/R
## Packages/BRCindicators/vignette/using_the_pipeline.Rmd', encoding
```

```
## R Version:  R version 3.2.0 (2015-04-16)
```

```
## snowfall 1.84-6 initialized (using snow 0.3-13): parallel execution on 4 CPUs.
```

```
# Export my data to the cluster
sfExport('formattedOccData')

# I create a function that takes a species name and runs my model
occ_mod_function <- function(taxa_name){

  library(sparta)

  # Note that this will write you results to your computer
  # the location is set to your user folder
  occ_out <- occDetFunc(taxa_name = as.character(taxa_name),
                        n_iterations = 200,
                        burnin = 15,
                        occDetdata = formattedOccData$occDetdata,
                        spp_vis = formattedOccData$spp_vis,
                        write_results = TRUE,
                        output_dir = '~/Testing_indicator_pipe',
```

```
                        seed = 123)
}

# I then run this in parallel
system.time({
para_out <- sfClusterApplyLB(unique(myData$taxa), occ_mod_function)
})
```

```
##    user  system elapsed
##    0.55    0.06  157.42
```

```
# Stop the cluster
sfStop()
```

```
##
## Stopping cluster
```

```
# We can see all the files this has created
list.files('~/Testing_indicator_pipe')
```

```
##  [1] "a.rdata" "b.rdata" "c.rdata" "d.rdata" "e.rdata" "f.rdata" "g.rdata"
##  [8] "h.rdata" "i.rdata" "j.rdata" "k.rdata" "l.rdata" "m.rdata" "n.rdata"
## [15] "o.rdata" "p.rdata" "q.rdata" "r.rdata" "s.rdata" "t.rdata" "u.rdata"
## [22] "v.rdata" "w.rdata" "x.rdata" "y.rdata" "z.rdata"
```

Installing BRCindicators

Installing the package is easy and can be done in a couple of lines

```
library(devtools)
install_github(repo = 'biologicalrecordscentre/BRCindicators')
```

Summarising sparta output for indicator

Now that we have some species trends data to work with (no doubt you already have your own) we can use the first function in BRCindicators. This function reads in all the output files from sparta (which are quite large and complex) and returns a simple summary table that we can use for calculating the indicator. If you have done your analysis without using sparta you can skip to the next step.

```
library(BRCindicators)

# All we have to supply is the directory where out data is saved
# You will note this is the 'output_dir' passed to sparta above.
trends_summary <- summarise_occDet(input_dir = '~/Testing_indicator_pipe')
```

```
## Loading data...done
```

```r
# Lets see the summary
head(trends_summary[,1:5])
```

```
##      year          a         b         c         d
## [1,] 1950 0.70478495 0.6649462 0.5961290 0.5584946
## [2,] 1951 0.79693548 0.3507527 0.4298925 0.3935484
## [3,] 1952 0.58172043 0.5009677 0.2550538 0.3822043
## [4,] 1953 0.29016129 0.5557527 0.5461290 0.3884946
## [5,] 1954 0.42155914 0.3305914 0.6818817 0.2032258
## [6,] 1955 0.07209677 0.3506452 0.6155914 0.3051613
```

Returned from this function is a summary of the data as a matrix. In each row we have the year, specified in the first column, and each subsequent column is a species. The values in the table are the mean of the posterior for the predicted proportion of sites occupied, a measure of occurrence.

Calculating indicator values

The next step is to re-scale the data so that the value for all species in the first year is the same. Once this is done we calculate the geometric mean across species for each year creating the indicator value. This function also accounts for species that have no data at the beginning of the dataset by entering them at the geometric mean for that year, this stops them dramatically changing the indicator value in the year they join the dataset. It also accounts for species that leave the dataset before the end by holding them at their last value. Finally limits to species values can be given, preventing extremely high or low values biasing the indicator.

The data I have generated in 'trends_summary' is very easy to work with but to show off what this function can do I'm going to mess it up a bit.

```r
trends_summary[1:3, 'a'] <- NA
trends_summary[1:5, 'b'] <- NA
trends_summary[2:4, 'c'] <- 1000
trends_summary[45:50, 'd'] <- NA

# Let's have a look at these changes
head(trends_summary[,1:5])
```

```
##      year          a         b            c         d
## [1,] 1950         NA        NA    0.5961290 0.5584946
## [2,] 1951         NA        NA 1000.0000000 0.3935484
## [3,] 1952         NA        NA 1000.0000000 0.3822043
## [4,] 1953 0.29016129        NA 1000.0000000 0.3884946
## [5,] 1954 0.42155914        NA    0.6818817 0.2032258
## [6,] 1955 0.07209677 0.3506452    0.6155914 0.3051613
```

```r
tail(trends_summary[,1:5])
```

```
##       year         a          b         c  d
## [45,] 1994 0.1623118 0.66333333 0.0544086 NA
## [46,] 1995 0.4450000 0.27978495 0.7768817 NA
## [47,] 1996 0.5569355 0.58451613 0.5604301 NA
## [48,] 1997 0.2257527 0.77548387 0.8226344 NA
## [49,] 1998 0.7445161 0.07645161 0.4878495 NA
## [50,] 1999 0.2616129 0.51586022 0.7089247 NA
```

Now that I have 'messed up' the data a bit we have two species with data missing at the beginning and one species with data missing at the end. We also have one species with some very high values.

Now lets run this through the re-scaling function.

```
# Let's run this data through our scaling function (all defaults used)
rescaled_trends <- rescale_species(Data = trends_summary)

# Here's the result
head(rescaled_trends[,c('year', 'indicator', 'a', 'b', 'c', 'd')])
```

```
##      year indicator         a        b          c         d
## [1,] 1950 100.00000        NA       NA   100.0000 100.00000
## [2,] 1951 110.73239        NA       NA 10000.0000  70.46592
## [3,] 1952 125.08623        NA       NA 10000.0000  68.43473
## [4,] 1953 112.72013 112.72013       NA 10000.0000  69.56103
## [5,] 1954  94.66601 163.76479       NA   114.3849  36.38814
## [6,] 1955  99.38391  28.00773 99.38391   103.2648  54.63997
```

```
tail(rescaled_trends[,c('year', 'indicator', 'a', 'b', 'c', 'd')])
```

```
##       year indicator         a         b          c        d
## [45,] 1994  90.99655  63.05393 188.00962   9.126984 120.8895
## [46,] 1995 108.99098 172.87095  79.29989 130.321068 120.8895
## [47,] 1996 115.65614 216.35498 165.67033  94.011544 120.8895
## [48,] 1997 114.85272  87.69906 219.79662 137.996032 120.8895
## [49,] 1998  99.44455 289.22519  21.66880  81.836219 120.8895
## [50,] 1999 102.80190 101.62982 146.21108 118.921356 120.8895
```

You can see that species 'a' and 'b' enter the dataset at the geometric mean (the indicator value), all species are indexed at 100 in the first year and the very high values in 'c' are capped at 10000 at the end 'd' has been held at it's end value.

The 'indicator' column that is returned here is our indicator, calculated as the geometric mean of all the species in the data set.

Getting confidence intervals for the indicator

We can get confidence intervals for this indicator by bootstrapping across species. We have a function for that too!

```
# This function takes just the species columns
scaled_species <- rescaled_trends[,!colnames(rescaled_trends) %in% c('year', 'indicator')]
indicator_CIs <- bootstrap_indicator(Data = scaled_species)
```

```
## Running bootstrapping for 10000 iterations...done
```

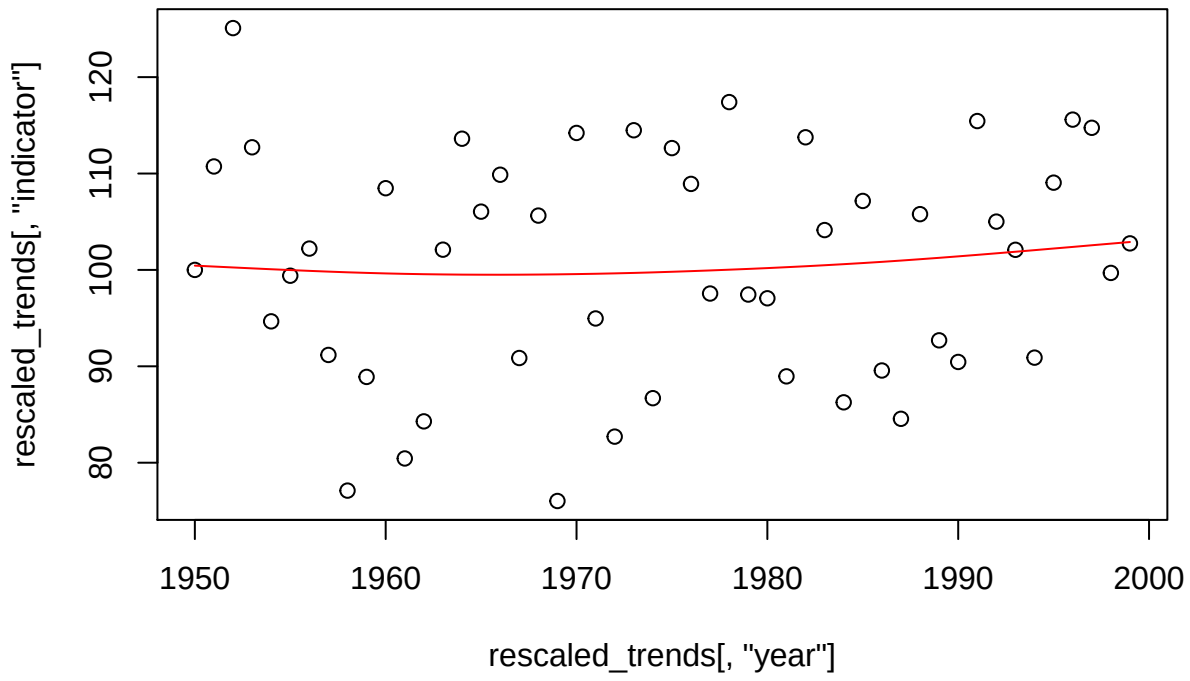
```
# Returned are the CIs for our indicator
head(indicator_CIs)
```

```
##      quant_025 quant_975
## [1,] 100.00000  100.0000
## [2,]  75.34724  180.9767
## [3,]  91.35183  197.7696
## [4,]  72.11378  184.6584
## [5,]  76.44115  117.3195
## [6,]  81.07227  120.7961
```

Creating a smoothed indicator

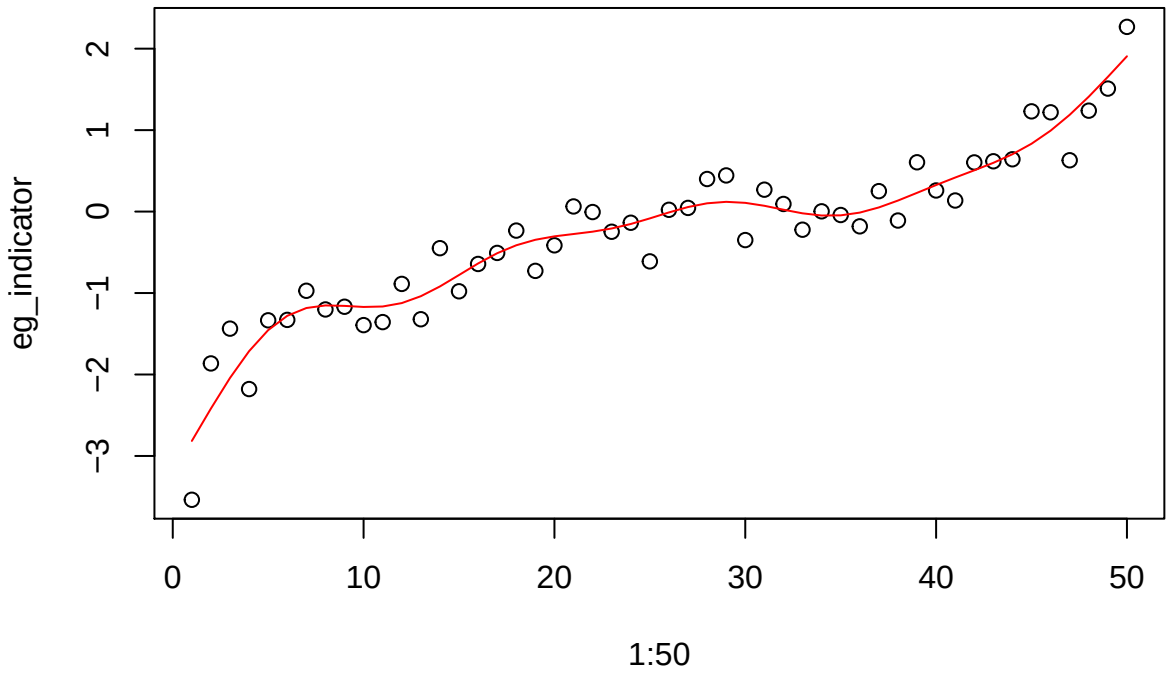
It is sometimes desirable to create a smoothed indicator value from the raw values. This can be achieved by fitting a GAM (general additive model) to the indicator using a spline. This spline is a smoothed curve that goes through the raw values for the indicator and is fitted using the function 'gam' in the 'mgcv' R package.

```
# The smoothing function takes the indicator values
smoothed_indicator <- GAM_smoothing(rescaled_trends[,'indicator'])

# In this example there is little support for a non-linear trend and
# so the line almost linear
plot(x = rescaled_trends[,'year'], y = rescaled_trends[,'indicator'])
lines(x = rescaled_trends[,'year'], y = smoothed_indicator, col = 'red')
```




```
# But if our indicator did support a non-linear trend it might look
# like this
eg_indicator <- jitter(sort(rnorm(50)), amount = 0.5)
```

```r
eg_smoothed <- GAM_smoothing(eg_indicator)
plot(x = 1:50, y = eg_indicator)
lines(x = 1:50, y = eg_smoothed, col = 'red')
```

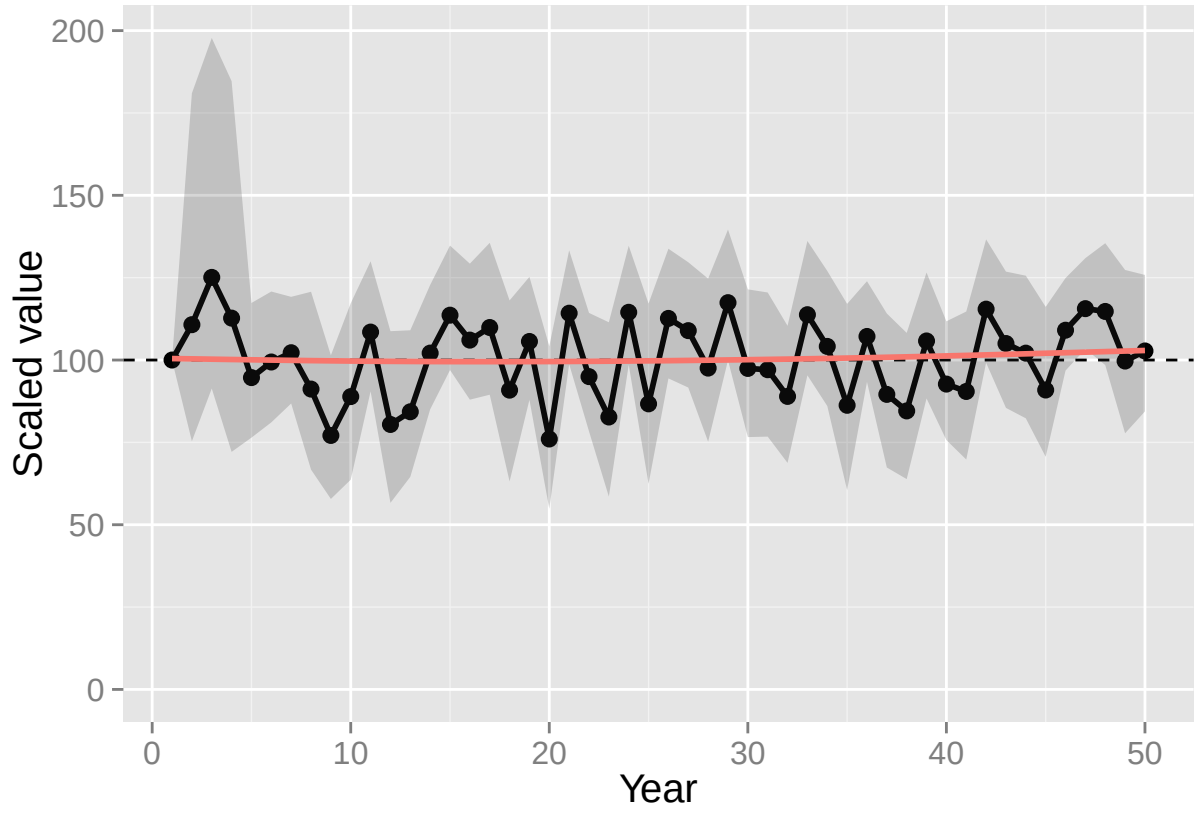


Where there is little support for a non-linear trend a GAM smoothed line will tend towards linear. Where there is good support for a non-linear trend the smoothed line will become more 'bendy'.

Plotting the indicator

We now have our indicator and the confidence intervals around it. The next step is to plot it. We have included a function that creates a simple plot using ggplot2, however you could easily create your own plots in R using the data.

```r
# Plot our indicator.
plot_indicator(indicator = rescaled_trends[,'indicator'],
               smoothed_line = smoothed_indicator,
               CIs = indicator_CIs)
```




```
## NULL
```

In this plot you can see the high upper confidence interval in years 2-4, this is due to the artificially high values we gave to species 'c'.

Creating a custom pipeline function

We have demonstrated how you might run the indicator functions one at a time, however in a 'pipeline' we want data to flow through seamlessly. Additionally there are a number of parameters in the functions that we have not shown you that you might find useful. Here is an example of how you can create your own pipeline function. Our function will wrap around the functions described above, setting the parameters to meet our needs. Once we have done this it will allow use to execute our pipeline in one line.

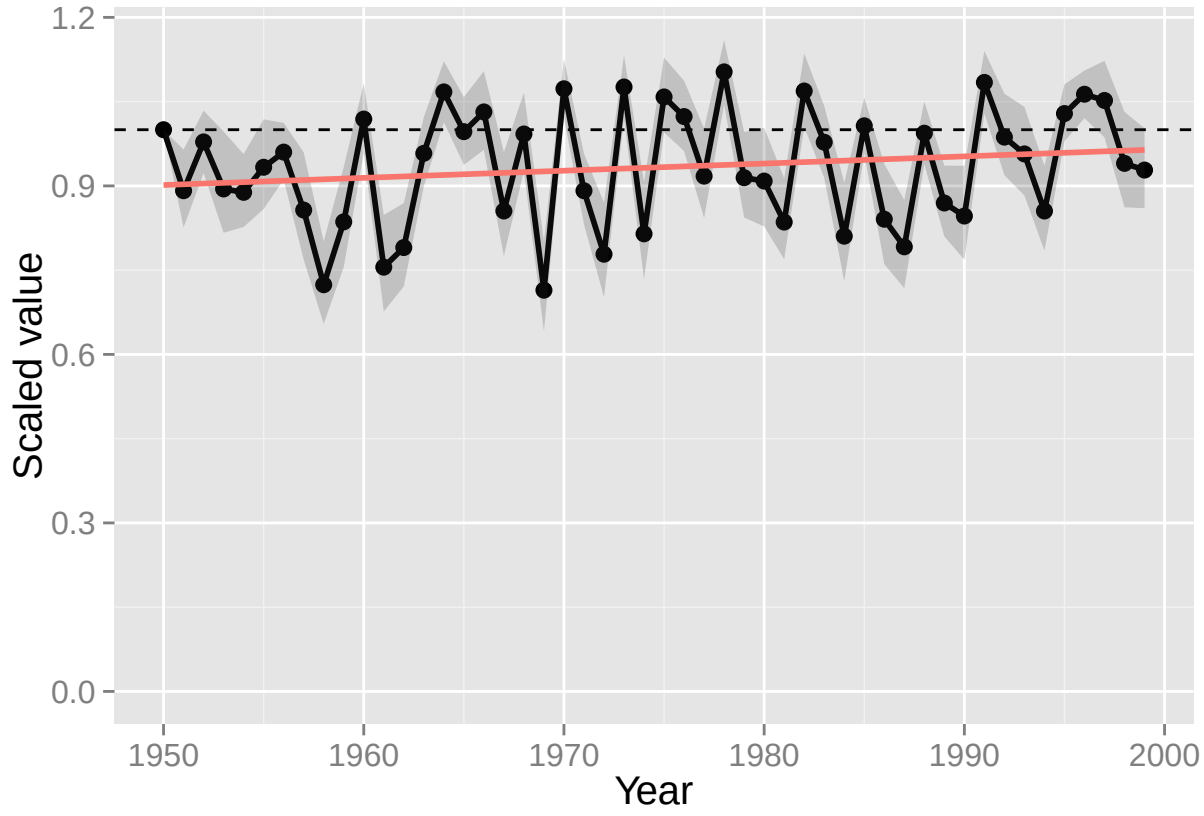
```
# I call my function 'run_pipeline' and the only arguement it
# takes is the directory of sparta's output
run_pipeline <- function(input_dir){

  require(sparta)
  require(BRCindicators)

  # Create the trends summary
  trends_summary <- summarise_occDet(input_dir = input_dir)

  # Rescale the values and get the indicator values
```

```
  # Here I set the index to 1 and change the value limits
  rescaled_trends <- rescale_species(Data = trends_summary,
                                     index = 1,
                                     max = 100,
                                     min = 0.001)

  # Bootstrap the indicator to get CIs
  scaled_species <- rescaled_trends[,!colnames(rescaled_trends) %in% c('year', 'indicator')]
  # This time I set the iterations to twice the default and
  # use custom confidence intervals
  indicator_CIs <- bootstrap_indicator(Data = scaled_species,
                                       CI_limits = c(0.25, 0.75),
                                       iterations = 20000)

  # Get the smoothed indicator line
  smoothed_indicator <- GAM_smoothing(rescaled_trends[,'indicator'])

  # This time I specify the years and index value
  plot_indicator(indicator = rescaled_trends[,'indicator'],
                 year = rescaled_trends[,'year'],
                 index = 1,
                 CIs = indicator_CIs,
                 smoothed_line = smoothed_indicator)

  ## I'll return all my data
  return(cbind(smoothed_indicator, indicator_CIs, as.data.frame(trends_summary)))
 }
```

Once we have created this function we can run this pipeline on a directory in one line, or put it in a loop to run across many directories.

```
# Now we can run the pipeline in one line, like a boss
indicator_data <- run_pipeline(input_dir = '~/Testing_indicator_pipe')
```

```
## Loading data...done
## Running bootstrapping for 20000 iterations...done
```




```
head(indicator_data)
```

```
##   smoothed_indicator  quant_25  quant_75 year          a         b
## 1          0.9015525 1.0000000 1.0000000 1950 0.70478495 0.6653763
## 2          0.9028268 0.8259136 0.9647625 1951 0.79693548 0.3511290
## 3          0.9041012 0.9228537 1.0338799 1952 0.58172043 0.4996774
## 4          0.9053755 0.8165306 0.9967839 1953 0.29016129 0.5541935
## 5          0.9066499 0.8267958 0.9566039 1954 0.42155914 0.3294086
## 6          0.9079242 0.8587982 1.0184956 1955 0.07209677 0.3494624
##           c         d         e         f         g         h         i
## 1 0.5974194 0.5584946 0.3966667 0.4308602 0.6446774 0.69876344 0.30220430
## 2 0.4287097 0.3935484 0.4836559 0.6439785 0.5614516 0.11301075 0.63258065
## 3 0.2547849 0.3822043 0.4333871 0.2561828 0.7812366 0.53962366 0.67741935
## 4 0.5459677 0.3884946 0.5011290 0.8284409 0.3850538 0.02548387 0.71102151
## 5 0.6819892 0.2032258 0.8316129 0.9126344 0.3236559 0.33016129 0.42682796
## 6 0.6162366 0.3051613 0.7269892 0.5935484 0.7659677 0.70693548 0.08295699
##           j         k         l         m         n         o         p
## 1 0.6470968 0.7858065 0.5540860 0.5243548 0.8059677 0.3745699 0.2611290
## 2 0.5486022 0.3579032 0.8134409 0.8210215 0.6904301 0.8947849 0.7148387
## 3 0.4660215 0.9073656 0.5485484 0.7880108 0.7336559 0.7823118 0.5344086
## 4 0.7581183 0.5990323 0.5588710 0.5783333 0.8790323 0.7585484 0.4128495
## 5 0.5518280 0.4172043 0.6714516 0.5704839 0.7836559 0.8179032 0.6227957
## 6 0.5869355 0.5444624 0.7652688 0.8324731 0.8024731 0.8157527 0.8816667
##           q         r         s         t         u         v         w
## 1 0.8543011 0.4128495 0.8133333 0.8835484 0.9443548 0.7822043 0.8990860
```

```
## 2 0.4618817 0.3774731 0.7354301 0.3758065 0.9224194 0.4482258 0.8893548
## 3 0.9264516 0.8145699 0.9229570 0.4051075 0.9124731 0.6686559 0.6076344
## 4 0.8619892 0.7209677 0.7547312 0.9301075 0.5318817 0.8969892 0.6652688
## 5 0.8366129 0.5326344 0.3087634 0.7564516 0.3589247 0.8827957 0.7860215
## 6 0.8315054 0.4775806 0.6786559 0.6886559 0.9084946 0.9076882 0.8343011
##           x         y         z
## 1 0.7770430 0.8680108 0.9267204
## 2 0.5982796 0.8382258 0.7405376
## 3 0.8288710 0.7811828 0.7397849
## 4 0.8540323 0.7969892 0.6544624
## 5 0.5259677 0.9719355 0.6649462
## 6 0.8943011 0.7767204 0.7719892
```