Calculating predator-derived measures of prey availability (pCPUE)

Cathy Cavallo ccavallo.ecology@gmail.com

8/01/2019

Calculating predator-derived measures of prey availability (pCPUE)

This code will allow the user to subset and filter a weighbridge dataset, calculate measures of foraging success and effort and combine with diet information to calculate prey-specific catch per unit effort (pCPUE) and overall catch per unit effort (CPUE).

Install. packages

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr

## Conflicts with tidy packages ----------------------------------------------

## filter(): dplyr, stats
## lag():    dplyr, stats
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following object is masked from 'package:purrr':
##
##     transpose

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(tidyr)
library(ggplot2) #for plotting nice figures
library(reshape2) #use this to convert dataframes from wideform to longform
```

```
##
## Attaching package: 'reshape2'

## The following objects are masked from 'package:data.table':
##
##     dcast, melt

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(lubridate) #for use with dates
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday,
##     week, yday, year

## The following object is masked from 'package:base':
##
##     date
```

Load scat sampling date data to use to subset APMS (weighbridge) data.

```
SampleDates<-read.csv("SampleDatesWDummy.csv",stringsAsFactors = FALSE)
SampleDates$Poo.collection.date<-as.Date(as.character(SampleDates$Poo.collection.date,format="%d/%m/%Y"
#if the above doesn't give the right day format, use the other one (both options below)
#as.POSIXct( SampleDates$Poo.collection.date, format = "%d/%m/%Y)
#as.Date(as.character(SampleDates$Poo.collection.date,format="%d/%m/%Y"))
```

Subset the data to 1 week before scat sampling

add two columns to create date range

Note that if you have brought Poo.collection.date in as POSIXct then these columns won't work. Make sure it is as.Date(as.character())
This is because the following function will try and knock 1, 4 and 7 seconds off, not days if you have it in the wrong format!

```
SampleDates$Poo.deposited<-SampleDates$Poo.collection.date-1
SampleDates$Week.before<-SampleDates$Poo.collection.date-7
```

Bring in APMS (weighbridge data)

```
APMS<-read.csv('APMS_20180516.csv',stringsAsFactors = FALSE)
```

```
APMS$Burrow<-as.factor(APMS$burrow.apms)
APMS$weight<-as.numeric(APMS$weight)
```

```
## Warning: NAs introduced by coercion
```

```
APMS$datetime<-as.POSIXct( APMS$datetime, format = "%d/%m/%Y %H:%M",tz="UTC")
APMS$date<-as.Date(APMS$date, format = "%d/%m/%Y")



APMS$Localdatetime<-format(APMS$datetime, tz="Australia/Sydney",usetz=TRUE)
APMS$Localdatetime<-as.POSIXct(APMS$Localdatetime,tz="Australia/Sydney")
APMS$Localdate<-date(APMS$Localdatetime)
```

Subset APMS data to only trips within your sampling dates

```
MyTrips<-data.frame()
class(APMS)
```

```
## [1] "data.frame"
```

```
#i=9
for (i in 1:length(SampleDates$Poo.collection.date)){
  #We first subsample the data from the weighbridge we want matching our dates from CPUFERRA data.
  NewAPMS<-APMS[which(APMS$Localdate<=SampleDates$Poo.deposited[i]&
                  APMS$Localdate>=SampleDates$Week.before[i]),]

  #We then create the columns we need in our weighbridge data subsample
  NewAPMS$Poo.collection.date<-SampleDates$Poo.collection.date[i] #but this stage stuffs it up
  NewAPMS$Poo.deposited<-SampleDates$Poo.deposited[i]
  NewAPMS$Week.before<-SampleDates$Week.before[i]

  #We rowbind the weighbridge data subsample to our Final data set
 MyTrips<-rbind(MyTrips,NewAPMS)
  }


 #rm(NewAPMS,i)
currentDate <- Sys.Date()
csvFileName <- paste("My Trips APMS",currentDate,".csv",sep="")
write.csv(MyTrips, file=csvFileName)
```

Calculate trip duration & mass gain

create functions to calculate trip duration and mass gain

First, make sure that the APMS data is sorted by the datetime column.

```
MyTrips<-MyTrips[order(MyTrips$datetime, decreasing = FALSE),]
```

Trip Duration

```
data<-MyTrips
tag<-"tag"
datetime<-"Localdatetime"
direction<-"direction"

tripDurationdf<-function(data, tag, datetime, direction){
  DateTimeColumn<-which(names(data)== datetime)
  TagColumn<-which(names(data)== tag)
  DirectionColumn<-which(names(data)== direction)
  data$TripDuration<-NA

  data$sortorder<-1:nrow(data) #To preserve the original order of the data

  data<-data[order(data[,DateTimeColumn]),] #Sort by time so that ins that should follow outs are conse
  DataHold<-data[1,]
  DataHold[1,]<-NA

  #i<-unique(data[,TagColumn])[1]
  for(i in unique(data[,TagColumn])){
    ActiveTagRows<-data[which(data[,TagColumn]==i),]

    for(j in 1:nrow(ActiveTagRows)){
      if(j==nrow(ActiveTagRows)){
        break
      }
      if(ActiveTagRows[j,DirectionColumn] == "out" & ActiveTagRows[j+1,DirectionColumn]=="in"){
        TripDuration<-difftime(time1 = ActiveTagRows[j+1,DateTimeColumn], time2 = ActiveTagRows[j,DateTi
        ActiveTagRows$TripDuration[(j+1)]<-as.numeric(TripDuration) #insert it into the TripDuration co
      }
    }
  DataHold<-rbind(DataHold,ActiveTagRows)
  }
  data<-DataHold[-1,]
  data<-data[order(data$sortorder),]
  return(data)
}
```

Mass Change

```
data<-MyTrips
tag<-"tag"
datetime<-"Localdatetime"
direction<-"direction"
mass<-"weight"
MassChangedf<-function(data, tag, datetime, direction,mass){
  DateTimeColumn<-which(names(data)== datetime)
  TagColumn<-which(names(data)== tag)
  DirectionColumn<-which(names(data)== direction)
  MassColumn<-which(names(data)==mass)
  data$MassChange<-NA
data$sortorder<-1:nrow(data) #To preserve the original order of the data

data<-data[order(data[,DateTimeColumn]),] #Sort by time so that ins that should follow outs are conse
DataHold<-data[1,]
DataHold[1,]<-NA
i<-unique(data[,TagColumn])[2]
for(i in unique(data[,TagColumn])){
  ActiveTagRows<-data[which(data[,TagColumn]==i),]

  for(j in 1:nrow(ActiveTagRows)){
    if(j==nrow(ActiveTagRows)){
      break
    }
    if(ActiveTagRows[j,DirectionColumn] == "out" & ActiveTagRows[j+1,DirectionColumn]=="in"){
      if(!is.na(ActiveTagRows[j+1,MassColumn]) & !is.na(ActiveTagRows[j,MassColumn])){
        MassChange<-(ActiveTagRows[j+1,MassColumn]-ActiveTagRows[j,MassColumn])
        ActiveTagRows$MassChange[j+1]<-MassChange #insert it into the TripDuration column of the "in"
      }
    }
  }
DataHold<-rbind(DataHold,ActiveTagRows)
}
data<-DataHold[-1,]
data<-data[order(data$sortorder),]
DropColumn<-which(names(data)== "sortorder")
data<-data[,-DropColumn]
return(data)
}
```

Run these functions on your trip data

```
MyTrips<-tripDurationdf(data=MyTrips,tag = "tag",datetime = "Localdatetime",direction = "direction")
MyTrips<-MassChangedf(data=MyTrips,tag="tag",datetime="Localdatetime",direction="direction",mass="weigh
```

Filtering

Filter to retain only trips that returned on the day poo was deposited

```
MyTrips$DateOfReturn<-as.Date(NA)
MyTrips$DateOfReturn[MyTrips$direction == "in"]<-MyTrips$Localdate[MyTrips$direction == "in"]


FilterReturns<-which(MyTrips$direction=='in'&MyTrips$DateOfReturn==MyTrips$Poo.deposited)

MyTrips<-MyTrips[FilterReturns,]
class(MyTrips$date)
```

```
## [1] "Date"
```

```
class(MyTrips$direction)
```

```
## [1] "character"
```

```
class(MyTrips$DateOfReturn)
```

```
## [1] "Date"
```

```
class(MyTrips$Localdatetime)
```

```
## [1] "POSIXct" "POSIXt"
```

My Trips

When checking these (and the one before) remember that the first "in" direction rows will have NA for all calculated trip data!
###Convert trip duration hours to days

```
MyTrips$TripDurationDays<-ceiling(MyTrips$TripDuration/24)
```

Limits

Limit trips to =<7 days
Limit mass gain to between 0 & 420 g

```
MyTripsAC<-MyTrips
#Mass gain from published papers
FilterMassAC<-which(MyTripsAC$MassChange>(0) & MyTripsAC$MassChange<420 &MyTripsAC$TripDurationDays<=7)

MyTripsAC$ValidMassChangeAC<-NA
MyTripsAC$ValidMassChangeAC[FilterMassAC]<-MyTripsAC$MassChange[FilterMassAC]
MyTripsAC$ValidMassChangeAC[!FilterMassAC]<-NA



#Filter to only trips that would match diet data
FilterIndices<-which(MyTripsAC$TripDurationDays<=7)

MyTripsAC$trip.duration.daysValid<-NA
MyTripsAC$trip.duration.daysValid[FilterIndices]<-MyTrips$TripDurationDays[FilterIndices]
MyTripsAC$trip.duration.daysValid[!FilterIndices]<-NA

currentDate <- Sys.Date()
csvFileNameAC <- paste("My Trips APMS Valid trip data_AC ",currentDate,".csv",sep="")
write.csv(MyTripsAC, file=csvFileNameAC)
```

Stage Summary Get tripdata number of stages per date

This isn't needed but it is good for checking whether you have balanced breeding stages per each date. You probably won't unless your species/colony nests asynchronously.

```
MyTrips2<-MyTrips
MyTrips2$MONTH<-month(MyTrips2$datetime)
TripStageSummary<-MyTrips2%>%
  group_by(date,MONTH,stage)%>%
  summarise(Nstage=n())

TripStageSummary$stage<-as.character(TripStageSummary$stage)
TripStageSummary<-TripStageSummary[TripStageSummary$stage == "I"|TripStageSummary$stage == "G"| TripSta

#make stage a factor again
TripStageSummary$fStage<-as.factor(TripStageSummary$stage)
class(TripStageSummary$date)
```

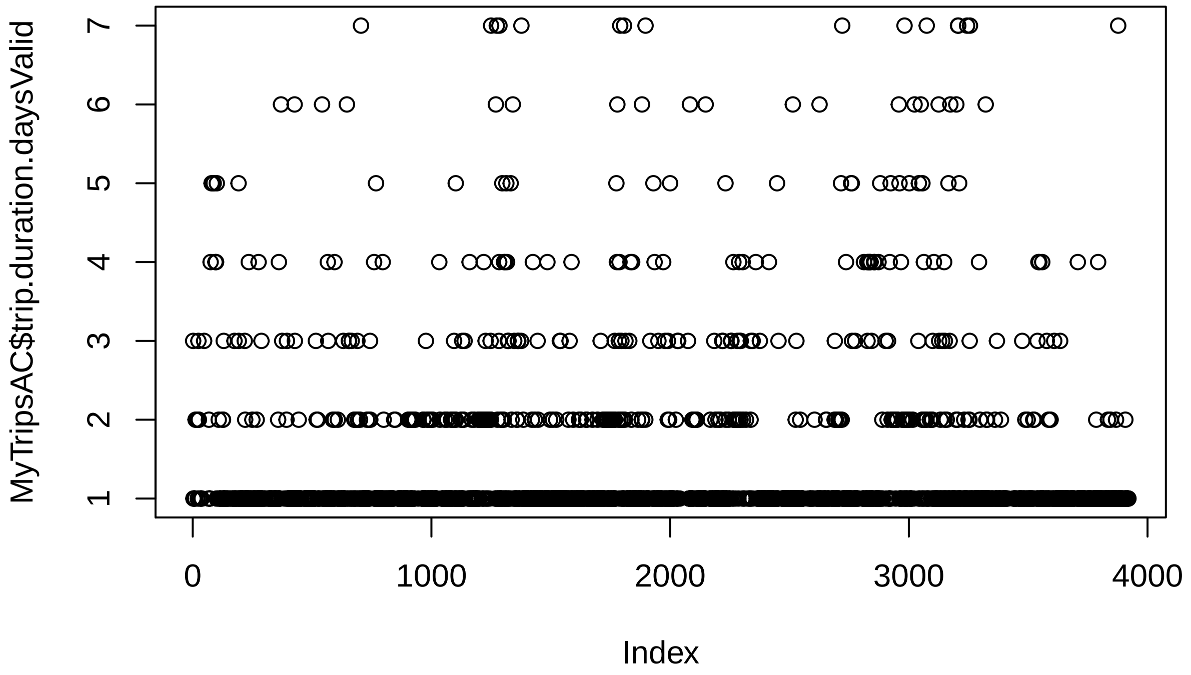
```
## [1] "Date"
```

```
write.csv(TripStageSummary,"TripStageSummary_20181102.csv")
```

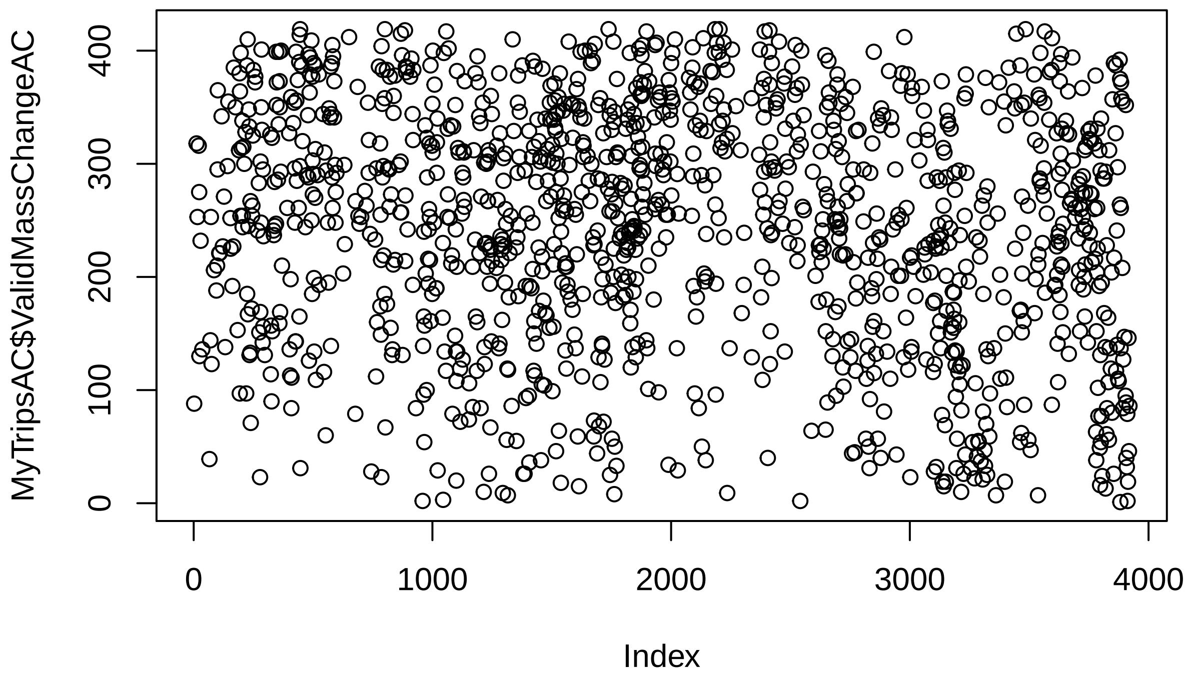
Which average to use? Mean, median or mode?

Test visually and statistically.

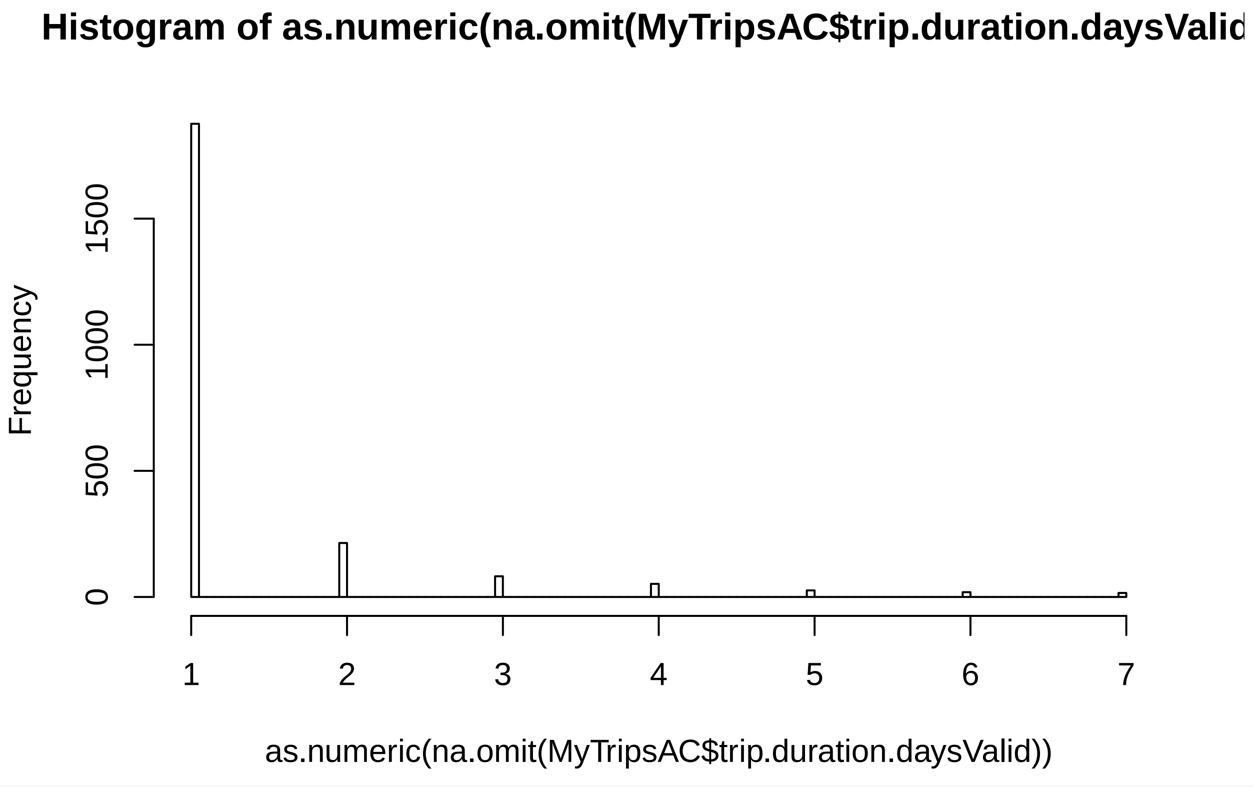
```
plot(MyTripsAC$trip.duration.daysValid)
```



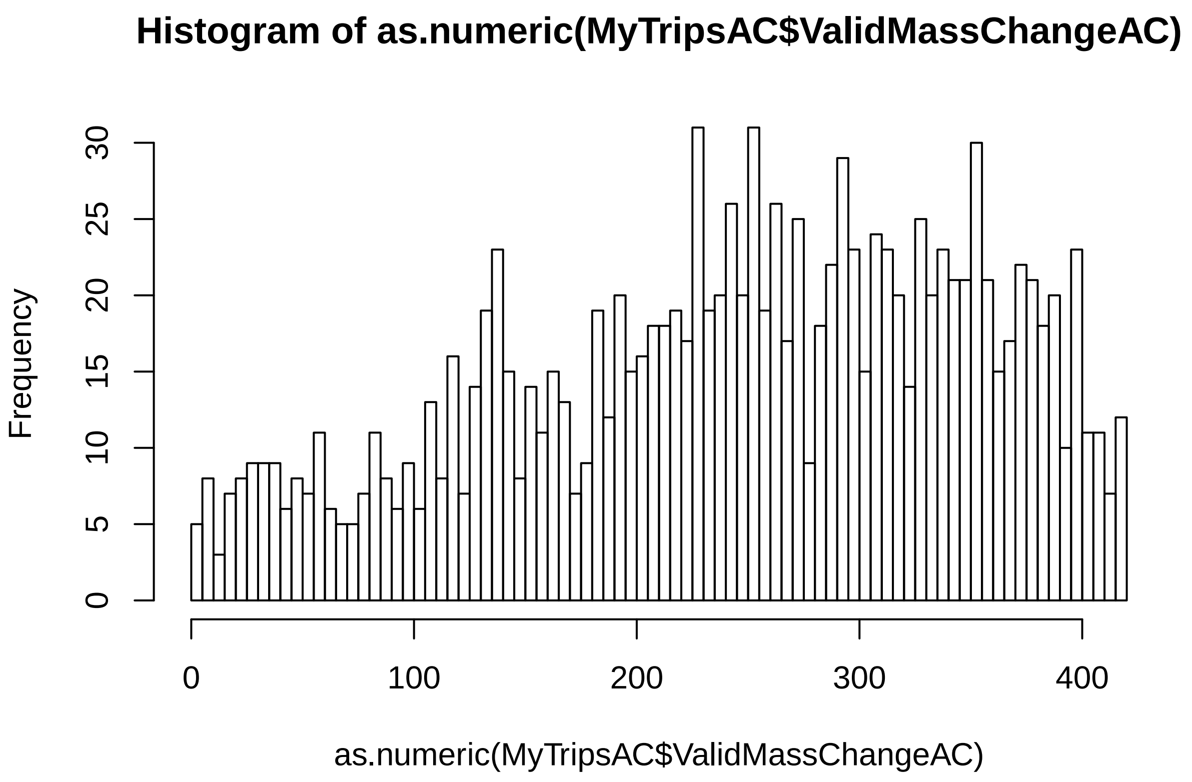
```
plot(MyTripsAC$ValidMassChangeAC)
```



```
hist(as.numeric(na.omit(MyTripsAC$trip.duration.daysValid)),breaks=100)
```



```
hist(as.numeric(MyTripsAC$ValidMassChangeAC),breaks=100)
```



```
library(moments)
agostino.test(MyTripsAC$trip.duration.daysValid)
```

```
##
##  D'Agostino skewness test
##
## data:  MyTripsAC$trip.duration.daysValid
## skew = 3.4331, z = 34.3520, p-value < 2.2e-16
## alternative hypothesis: data have a skewness
```

```
agostino.test(MyTripsAC$ValidMassChangeAC)
```

```
##
##  D'Agostino skewness test
##
## data:  MyTripsAC$ValidMassChangeAC
## skew = -0.40237, z = -5.69360, p-value = 1.244e-08
## alternative hypothesis: data have a skewness
```

create a function that calculates mode

Remember that the mode is the value that occurs most frequently

```
getmode<-function(d){
  uniqd<-na.omit(unique(d))
  uniqd[which.max(tabulate(match(d,uniqd)))]
}
```

Average foraging trip duration and mass gain by stage and date

I have left in the ability for people to choose median, mean or mode as their average because it may differ between groups. However, judging by the fact that foragers will most likely try and fill up before returning from a foraging trip I expect the frequency distribution of mass change will always be negatively skewed.

```
i=5
#rm(i)
HoldingFrame<-data.frame()

  for (i in 1:length(SampleDates$Poo.collection.date)){#for each unique Poo.collection.date

ActiveTripsStage<-MyTripsAC[which(MyTripsAC$Poo.collection.date==SampleDates$Poo.collection.date[i]),]


NewColumnsStage<-ActiveTripsStage%>% #In ActiveTrips, average all the trip durations for that Poo.colle
#Return averaged values in columns  with suffix _mean, _median, _mode.
  group_by(stage)%>%
  summarise(trip.duration.daysValid_count=sum(!is.na(trip.duration.daysValid)),trip.duration.daysValid_m
NewColumnsStage$Poo.collection.date<-SampleDates$Poo.collection.date[i]
HoldingFrame<-rbind(HoldingFrame,NewColumnsStage)
  }
#now rename holding frame
AveragedTripsACStage_all<-HoldingFrame
class(AveragedTripsACStage_all$stage)
```

```
## [1] "character"
```

```
AveragedTripsACStage_all$StageTrips<-as.factor((AveragedTripsACStage_all$stage))
```

Load the diet (scat sample) data

```
RRA16<-read.csv("FishRRA_2018-10-07.csv",stringsAsFactors = TRUE)
RRA16$Weekbeginspoo<-floor_date(as.Date(RRA16$DATE, "%Y-%m-%d"), unit="week")
RRA16$Poo.collection.date<-as.Date(RRA16$DATE, "%Y-%m-%d")
RRA16$SampleField<-as.character(RRA16$SampleField)
RRA16$SEASON<-as.factor(RRA16$SEASON)
RRA16$StageDiet<-RRA16$STAGE
```

Subset

```
RRA16en<-RRA16[,c("SampleField","Poo.collection.date","BURROW","SITE","SEASON","StageDiet","MONTH","BAR
```

Check for interactions with no values because these will need to be removed from the analysis

```
table(with(RRA16en, interaction(MONTH,StageDiet,SEASON)))
```

```
##
##  1.0.2015  2.0.2015  3.0.2015  8.0.2015  9.0.2015 10.0.2015 11.0.2015
##         0         0         0         0         0         0         0
## 12.0.2015  1.E.2015  2.E.2015  3.E.2015  8.E.2015  9.E.2015 10.E.2015
##         1         0         0         0         4        29         6
## 11.E.2015 12.E.2015  1.G.2015  2.G.2015  3.G.2015  8.G.2015  9.G.2015
##          7         7         3         5         0         0        18
## 10.G.2015 11.G.2015 12.G.2015  1.P.2015  2.P.2015  3.P.2015  8.P.2015
##         30        12        16         2        26         0         1
##  9.P.2015 10.P.2015 11.P.2015 12.P.2015  1.0.2016  2.0.2016  3.0.2016
##          7        14        29        35         0         0         0
##  8.0.2016  9.0.2016 10.0.2016 11.0.2016 12.0.2016  1.E.2016  2.E.2016
##          0         0         0         0         0         1         0
##  3.E.2016  8.E.2016  9.E.2016 10.E.2016 11.E.2016 12.E.2016  1.G.2016
##          0         0         8         6         4         1        11
##  2.G.2016  3.G.2016  8.G.2016  9.G.2016 10.G.2016 11.G.2016 12.G.2016
##          3         0         0        12        51        22         5
##  1.P.2016  2.P.2016  3.P.2016  8.P.2016  9.P.2016 10.P.2016 11.P.2016
##          8         7         6         0         5        22        40
## 12.P.2016
##         11
```

```r
table(with(RRA16en, interaction(MONTH,StageDiet)))
```

```
##
##  1.0  2.0  3.0  8.0  9.0 10.0 11.0 12.0  1.E  2.E  3.E  8.E  9.E 10.E 11.E
##    0    0    0    0    0    0    0    1    1    0    0    4   37   12   11
## 12.E  1.G  2.G  3.G  8.G  9.G 10.G 11.G 12.G  1.P  2.P  3.P  8.P  9.P 10.P
##    8   14    8    0    0   30   81   34   21   10   33    6    1   12   36
## 11.P 12.P
##   69   46
```

```r
table(with(RRA16en, interaction(MONTH,SEASON)))
```

```
##
##  1.2015  2.2015  3.2015  8.2015  9.2015 10.2015 11.2015 12.2015  1.2016
##       5      31       0       5      54      50      48      59      20
##  2.2016  3.2016  8.2016  9.2016 10.2016 11.2016 12.2016
##      10       6       0      25      79      66      17
```

```r
table(with(RRA16en, interaction(StageDiet,SEASON)))
```

```
##
## 0.2015 E.2015 G.2015 P.2015 0.2016 E.2016 G.2016 P.2016
##      1     53     84    114      0     20    104     99
```

```r
table(with(RRA16en, interaction(SITE,SEASON)))
```

```
##
##  PP.2015 RTB.2015  PP.2016 RTB.2016
##      126      126       95      128
```

```r
table(with(RRA16en, interaction(SEASON)))
```

```
##
## 2015 2016
##  252  223
```

CPUFE preparation

create date range in diet

Note that if you have brought Poo.collection.date in as POSIXct then these columns won't work. Make sure it is as.Date(as.character()) This is because the following function will try and knock 1, 4 and 7 seconds off, not days if you have it in the wrong format!

```
RRA16noNA<-RRA16en[complete.cases(RRA16en),]
class(RRA16noNA$Poo.collection.date)
```

```
## [1] "Date"
```

```
RRA16noNA$Poo.deposited<-RRA16noNA$Poo.collection.date-1
RRA16noNA$Week.before<-RRA16noNA$Poo.collection.date-7
```

Replace E's with I's

Use this to clean up data when you have used different names in different datasets. The breeding stages recorded during fieldwork are always assigned "I" for incubation, because it is part of a pre-defined long-term monitoring set up. Unfortunately, I used "E" for eggs in my scat sampling data.

```
class(RRA16noNA$StageDiet)
```

```
## [1] "factor"
```

```
levels(RRA16noNA$StageDiet)[levels(RRA16noNA$StageDiet)=="E"] <- "I"
RRA16ForCPUFE<-RRA16noNA
names(RRA16noNA)
```

```
##  [1] "SampleField"            "Poo.collection.date"
##  [3] "BURROW"                 "SITE"
##  [5] "SEASON"                 "StageDiet"
##  [7] "MONTH"                  "BARRACOUTA.Total"
##  [9] "RED.COD.Total"          "JACKMACKERAL.Total"
## [11] "SARDINE.Total"          "WAREHOU.Total"
## [13] "Bluefin.leatherjacket"  "Velvet.leatherjacket"
## [15] "Spiny.gurnard"          "Weedfish.sp."
## [17] "Acanthaluteres.sp."     "ANCHOVY.Total"
## [19] "Upeneichythys.sp."      "Poo.deposited"
## [21] "Week.before"
```

```
class(RRA16ForCPUFE$StageDiet)
```

```
## [1] "factor"
```

```
levels(RRA16ForCPUFE$StageDiet)
```

```
## [1] "0" "I" "G" "P"
```

```
RRA16ForCPUFE$StageDiet<-as.character(RRA16ForCPUFE$StageDiet)
RRA16ForCPUFE<-RRA16ForCPUFE[RRA16ForCPUFE$StageDiet != "0",]
RRA16ForCPUFE$fStageDiet<-as.factor(RRA16ForCPUFE$StageDiet)
```

drop NA/0 levels in APMS (weighbridge) data

Sometimes we don't have reproductive information about the birds crossing the weighbridge. These get dropped because we care about breeding-stage effects in this species.

```
#drop levels
AveragedTripsACStage_all$StageTrips<-as.character(AveragedTripsACStage_all$StageTrips)
AveragedTripsACStage_all<-AveragedTripsACStage_all[AveragedTripsACStage_all$StageTrips == "I"|AveragedTr

#make stage a factor again
AveragedTripsACStage_all$fStageTrips<-as.factor(AveragedTripsACStage_all$StageTrips)
```

Merge APMS & diet data

```
ForCPUFE_ACStageAll<-data.frame()
i=2
for(i in 1:nrow(RRA16ForCPUFE)){
  ActiveSample<-RRA16ForCPUFE[i,c("Poo.collection.date","fStageDiet")]# isolating that one cell (date i
  TripsRow<-AveragedTripsACStage_all[AveragedTripsACStage_all$Poo.collection.date == ActiveSample$Poo.co
                                  AveragedTripsACStage_all$fStageTrips == ActiveSample$fStageDiet,]#subset
  BurrowRow<-RRA16ForCPUFE[i,] #i is always the active row
  if( length(TripsRow$fStageTrips)>0){
    RowsBound<-cbind(BurrowRow,TripsRow)
    }
    ForCPUFE_ACStageAll<-rbind(ForCPUFE_ACStageAll,RowsBound)

}#ends function
```

Insert the seasonal relative read abundance of fish (Actinopterygii) from each year for pCPUE calculations

```
ForCPUFE_ACStageAll$FishRRA<-ifelse(test = ForCPUFE_ACStageAll$SEASON=="2015",62,64)
```

CPUFE calculation (w stage)

Define the fish you are interested in as "My Species".
Here I have left it so that the user can decide whether to use the median or mean values for foraging trip duration and mass change.

```
MySpecies<-c("BARRACOUTA.Total","RED.COD.Total","JACKMACKERAL.Total","SARDINE.Total","WAREHOU.Total","Bl
i="BARRACOUTA.Total"
#i="RED.COD.Total"
k=1
ForCPUFE_ACStageAll<-as.data.frame(ForCPUFE_ACStageAll)
CPUFE_ACStageAll<-as.data.frame(ForCPUFE_ACStageAll)
for (i in MySpecies){
  print(i)#this will print after each loop so you can see progress
  ColumnIndexStage<-which(names(ForCPUFE_ACStageAll) == i) #Which column number is the current fish in
  ColumnSubsetStage<-ForCPUFE_ACStageAll[,(ColumnIndexStage)]#subset to all rows, for the single column
```

```
CPUFE_meanStage<-vector()
CPUFE_medianStage<-vector()#create a empty vector (a vector because we want to bind a single vector to
  for(k in 1:nrow(ForCPUFE_ACStageAll)){#for each row get the DeltaMass and FTD.
    ActiveMass_meanStage<-ForCPUFE_ACStageAll$ValidMassChangeAC_mean[k]
    ActiveMass_medianStage<-ForCPUFE_ACStageAll$ValidMassChangeAC_median[k]
    ActiveFTD_meanStage<-as.numeric(ForCPUFE_ACStageAll$trip.duration.daysValid_mean[k])
    ActiveFishRRA<-ForCPUFE_ACStageAll$FishRRA[k]
     ActiveFTD_medianStage<-as.numeric(ForCPUFE_ACStageAll$trip.duration.daysValid_median[k])
    ActiveRRAStage<-ColumnSubsetStage[k]#then use the same row index to select our fish column of inter
    ActiveCPUFEmeanStage<-(((ActiveRRAStage/100)*(ActiveFishRRA/100))*ActiveMass_meanStage)/as.numeric(A
    CPUFE_meanStage<-c(CPUFE_meanStage,ActiveCPUFEmeanStage)#bind them.
      ActiveCPUFEmedianStage<-(((ActiveRRAStage/100)*(ActiveFishRRA/100))*ActiveMass_medianStage)/as.dou
      #compute CPUFE for one row of the fish column of interest at a timeas.double(ActiveFTD_median,uni
    CPUFE_medianStage<-c(CPUFE_medianStage,ActiveCPUFEmedianStage)#bind them.
    ColumnSeedStage<-data.frame(CPUFE_meanStage,CPUFE_medianStage)
  }#closes the row loop. At this bracket we have computed all of the rows for a single fish column
  CPUFEcolumnsStage<-ColumnSeedStage#rename vector seed as CPUFEcolumn
CPUFE_ACStageAll<-cbind(CPUFE_ACStageAll,CPUFEcolumnsStage) #bind the new column to the existing burrow
  names(CPUFE_ACStageAll)[names(CPUFE_ACStageAll) == 'CPUFE_meanStage'] <-paste(i,"_CPUFE_mean", sep =
names(CPUFE_ACStageAll)[names(CPUFE_ACStageAll) == 'CPUFE_medianStage'] <-paste(i,"_CPUFE_median", sep =
}
```

```
## [1] "BARRACOUTA.Total"
## [1] "RED.COD.Total"
## [1] "JACKMACKERAL.Total"
## [1] "SARDINE.Total"
## [1] "WAREHOU.Total"
## [1] "Bluefin.leatherjacket"
## [1] "Velvet.leatherjacket"
## [1] "Spiny.gurnard"
## [1] "Weedfish.sp."
## [1] "Acanthaluteres.sp."
## [1] "ANCHOVY.Total"
## [1] "Upeneichythys.sp."
```

Calculate total CPUFE

This is a simple measure of foraging success divided by foraging effort. It allows us to see whether predators are having to expend more effort to catch ANY prey. This is useful for generalist predators that are able to switch prey sources.

```
CPUFE_ACStageAll<-CPUFE_ACStageAll[,-c(28)]
CPUFE_ACStageAll<-CPUFE_ACStageAll%>%
 group_by(Poo.collection.date)%>%
  mutate(CPUFEtotal=ValidMassChangeAC_median/as.double(trip.duration.daysValid_median,units='secs'))
```

Export your data

I use this version to record a new version each time and avoid overwriting previous versions.

```r
currentDate <- Sys.Date()
csvFileName <- paste("CPUFE_ALL_ACStage_FishRRA_",currentDate,".csv",sep="")
write.csv(CPUFE_ACStageAll, file=csvFileName)
```