````
---
title: "Raw Data Merge"
output:
  html_document:
    df_print: paged
  html_notebook: default
  pdf_document: default
---

```{r}
library(tidyverse)
library(googlesheets4)
library(qualtRics)
library(lubridate)
```

Note:  the Qualtrics pretest data still has participant names, so this script cannot be fully executed without the Qualtrics credentials, which are stored elsewhere.


## Load list of valid IDs
```{r download}
gs4_auth(path=Sys.getenv("SHINY_PROTOCOL"))

#gs4_auth(email="russell.g.almond@gmail.com",
scopes="https://www.googleapis.com/auth/spreadsheets")
#gs4_deauth()

ValidIDs <- read_sheet("1lBp0LhNdSf0r1OtwC2cyqInB2npLbjH0KUMh4z_nzfM")$StudyID
AllIDs <- read_sheet("1BPkBMEE5DOtZ3MBt5C1ab4djddAOhUyAIPMiR4WRhBQ")

AllIDs %>% filter(StudyID %in% ValidIDs) -> PPIDs
AllIDs %>% filter(!(StudyID %in% ValidIDs)) -> InvIDs
```
# Survey Metadata

Experiment in trying to mine the QSF information to automagically build data dictionary.

```{r QualtricsSurveyIDs}
qualtricsIDs <- c(PRE="SV_bkALP80IYWnEgkK",ECT="SV_3BKybDb3f2v4QU6",
                  POT="SV_1MotNHu1dPEAJnM",Assent="SV_07esYssbhDccXs1",
                  Consent="SV_8vsIO1klt7DSPMG", PosttestA="SV_aeBB1Tw7bYKeIqq",
                  PosttestB="SV_4ISgzDcDm6BLiom", PretestA="SV_4IpHb99rMIEKtIq",
                  PretestB="SV_6eTzhLhhnhXrPmK")
```

## Question Titles
```{r grabMetadata}
pretestQ <- survey_questions(qualtricsIDs["PRE"])
potQ <- survey_questions(qualtricsIDs["POT"])
ectQ <- survey_questions(qualtricsIDs["ECT"])
posttestB <- survey_questions(qualtricsIDs["PosttestB"])
```


```{r cleanHTML}
````
````

```
pretestQuestions <- gsub("</?p>","",pretestQ$question)
names(pretestQuestions) <- pretestQ$qname
potQuestions <- gsub("</?p>","",potQ$question)
names(potQuestions) <- potQ$qname
ectQuestions <- gsub("</?p>","",ectQ$question)
names(ectQuestions) <- ectQ$qname
postBQuestions <- gsub("</?p>","",posttestB$question)
names(postBQuestions) <- posttestB$qname
```

Qualtrics QSF

```{r LoadQSF}
preQSF <- jsonlite::fromJSON("PP_IES_Pretest.qsf",FALSE)
potQSF <- jsonlite::fromJSON("PP_IES_Posttest_POT (2).qsf",FALSE)
ectQSF <- jsonlite::fromJSON("PP_IES_Posttest_ECT.qsf",FALSE)
postBQSF <- jsonlite::fromJSON("Posttest_B_Summer_Camp.qsf",FALSE)
```

Fetch the elements marked "SQ". Use the DataExportTag as the name, this should allow searching the metadata by name.

```{r preSQ}
preSQ <- lapply(preQSF$SurveyElements[which(sapply(preQSF$SurveyElements,
                                    function(el) el$Element=="SQ"))],
                function(sq) sq$Payload)
names(preSQ) <- sapply(preSQ, function(q) q$DataExportTag)
potSQ <- lapply(potQSF$SurveyElements[which(sapply(potQSF$SurveyElements,
                                    function(el) el$Element=="SQ"))],
                function(sq) sq$Payload)
names(potSQ) <- sapply(potSQ, function(q) q$DataExportTag)
ectSQ <- lapply(ectQSF$SurveyElements[which(sapply(ectQSF$SurveyElements,
                                    function(el) el$Element=="SQ"))],
                function(sq) sq$Payload)
names(ectSQ) <- sapply(ectSQ, function(q) q$DataExportTag)
postBSQ <- lapply(postBQSF$SurveyElements[which(sapply(postBQSF$SurveyElements,
                                    function(el) el$Element=="SQ"))],
                function(sq) sq$Payload)
names(postBSQ) <- sapply(postBSQ, function(q) q$DataExportTag)
```

Names for IMI questions

```{r IMI}
choices <- ectSQ$IMI$Choices
IMI.choices <- rep(NA_character_, max(as.numeric(names(choices))))
for (choi in names(choices)) {
  IMI.choices[as.numeric(choi)] <- choices[[choi]]$Display
}
chorder <- as.numeric(unlist(ectSQ$IMI$ChoiceOrder))
IMI.stems <- IMI.choices[chorder]
names(IMI.stems) <- paste("IMI",1L:length(IMI.stems),sep="_")
IMI.stems <- gsub("\t","",IMI.stems)  # Clean extra tabs.
```

Names for PA questions.

PA questions only appear on ECD form.

```{r PA.metadata}
PA.names <- paste("PA",1:7,sep="")
PA.stems <- sapply(ectSQ[PA.names],function (sq) sq$QuestionText)
PA.labels <- sapply(ectSQ[PA.names],function(sq) unlist(sq$Choices))
PA.labels
```

Names for the ethnicity questions

```{r ethnicityMetadata}
ethnames <- preSQ$Ethnicity$Choices
ethcolnames <- paste("Ethnicity",names(ethnames),sep="_")
ethnames <- unlist(ethnames)
ethnames <- ethnames[grepl("Display",names(ethnames))]
names(ethnames) <- ethcolnames
ethnames
```

Extract the keys

```{r getKeys}
preKey <- sapply(preSQ,function(sq) {
  if (is.null(sq$GradingData) || length(sq$GradingData)==0L) return (NA)
  as.numeric(sq$GradingData[[1]]$ChoiceID)
})
preKey <- preKey[!is.na(preKey)]
potKey <- sapply(potSQ,function(sq) {
  if (is.null(sq$GradingData) || length(sq$GradingData)==0L) return (NA)
  as.numeric(sq$GradingData[[1]]$ChoiceID)
})
potKey <- potKey[!is.na(potKey)]
ectKey <- sapply(ectSQ,function(sq) {
  if (is.null(sq$GradingData) || length(sq$GradingData)==0L) return (NA)
  as.numeric(sq$GradingData[[1]]$ChoiceID)
})
ectKey <- ectKey[!is.na(ectKey)]
postBKey <- sapply(postBSQ,function(sq) {
  if (is.null(sq$GradingData) || length(sq$GradingData)==0L) return (NA)
  as.numeric(sq$GradingData[[1]]$ChoiceID)
})
postBKey <- postBKey[!is.na(postBKey)]
```

```{r KeyAndFormCode}
keyID <- "1fagyGKc30Fx2ORwcWoULJ_dYZWqwPwJWCc-xo1pfV2o"
physics.key <- read_sheet(keyID,"Physics")
IMI.scales <- read_sheet(keyID,"IMI")
PA.rev <- read_sheet(keyID,"PA")
```

I have key information from two sources, Qualtrics and external file. Check to see if they are the same.

```{r keyCheck}
formA.keys <- filter(physics.key,Form=="A") %>% select(ID,Key)
```

```
formA.keys$prekey <- preKey[formA.keys$ID]
formA.keys$potKey <- potKey[formA.keys$ID]
formA.keys$ectKey <- ectKey[formA.keys$ID]
formA.keys
```

No key mismatches, but column labeling issue. For some reason, `A-FQ10` is called `AFQ10` in pretest.

```{r getChoices}
choicesToFactor <- function (sq, col) {
  levels <- as.numeric(names(sq$Choices))
  labels <- sapply(sq$Choices, function(c) c$Display)
  factor(col,levels,labels)
}
```

Load The Qualtrics Data

Exclude metadata fields, fields containing PID and "DO" fields which give presentation ordering.

```{r LoadPrePost}
pretest <- fetch_survey(qualtricsIDs["PRE"],force=TRUE,
                        label=FALSE,convert=FALSE) %>%
  filter(ID %in% ValidIDs) %>%
  select(!(StartDate:IPAddress)) %>%
  select(!(ResponseId:UserLanguage)) %>%
  select(!(First:Q80)) %>%
  select(!contains("DO"))
names(pretest)[5] <- "StudyID"
pretest <- filter(pretest,!is.na(StudyID))

ecttest <- fetch_survey(qualtricsIDs["ECT"],force=TRUE,
                        label=FALSE, convert=FALSE) %>%
  filter(UserID1 %in% ValidIDs) %>%
  select(!(StartDate:IPAddress)) %>%
  select(!(ResponseId:UserLanguage)) %>%
  select(!contains("DO"))
names(ecttest)[5] <- "StudyID"
ecttest <- filter(ecttest,!is.na(StudyID))

pottest <- fetch_survey(qualtricsIDs["POT"], force=TRUE,
                        label=FALSE, convert=FALSE) %>%
  filter(UserID1 %in% ValidIDs) %>%
  select(!(StartDate:IPAddress)) %>%
  select(!(ResponseId:UserLanguage)) %>%
  select(!contains("DO"))
names(pottest)[5] <- "StudyID"
pottest <- filter(pottest,!is.na(StudyID))
```

Clean duplicates.

```{r dupesbytest}
dupes.pre <- pretest$StudyID[duplicated(pretest$StudyID)]
"Pretest:"
dupes.pre
"ECT:"
dupes.ect <- ecttest$StudyID[duplicated(ecttest$StudyID)]
dupes.ect
"POT:"
dupes.pot <- pottest$StudyID[duplicated(pottest$StudyID)]
dupes.pot
```

POT

```{r POTdupe}
pottest %>% filter(StudyID%in%dupes.pot)
```
```{r cleanPOT}
pottest <- filter(pottest,!(StudyID %in% dupes.pot) | Finished)
dupes.pot1 <- pottest$StudyID[duplicated(pottest$StudyID)]
pottest %>% filter(StudyID%in%dupes.pot1)
```

For A2079 and D3654 take the row with non-missing PA1
For F1562 Flip a coin.

```{r dropDups}
pottest <- filter(pottest,!(StudyID %in% c("A2079","D3654")) | !is.na(PA1))
dupes.pot2 <- pottest$StudyID[duplicated(pottest$StudyID)]
for (dupID in dupes.pot2) {
  dupindex <- which(pottest$StudyID==dupID)
  dropindix <- sample(dupindex,length(dupindex)-1,replace=FALSE)
  pottest <- pottest[-dropindix,]
}
anyDuplicated(pottest$StudyID)
```

ECT

```{r dupECT}

filter(ecttest, StudyID %in% dupes.ect) %>% arrange(StudyID)

```
Select the ones which are finished.
```{r cleanECT}
ecttest <- filter(ecttest,!(StudyID %in% dupes.ect) | Finished)
ecttest$StudyID[duplicated(ecttest$StudyID)]
```

PRE

```{r dupPRE}

filter(pretest, StudyID %in% dupes.pre) %>%
```

```
  arrange(StudyID) %>% select(Progress, Finished, StudyID)

```



```{r cleanPRE}
pretest <- filter(pretest,!(StudyID %in% dupes.pre) | Finished)
dupes.pre1 <- pretest$StudyID[duplicated(pretest$StudyID)]
filter(pretest,StudyID %in% dupes.pre1)
```
Still one ambiguous case. Take the one with the longest duration.

```{r preDuration}
for (dupID in dupes.pre1) {
  dupindex <- which(pretest$StudyID==dupID)
  dur <- pretest[dupindex,"Duration (in seconds)",drop=TRUE]
  dropindix <- dupindex[dur < max(dur)]
  pretest <- pretest[-dropindix,]
}
anyDuplicated(pretest$StudyID)
```

Covert to factors and add labels.

Demographics

Only a few students checked Nonbinary or Other, so put them together. Also, call prefer not to say as NA.

A couple of students put XX years old, instead of their age, so fix.

```{r demofix}
pretest$Sex <- choicesToFactor(preSQ$Sex,pretest$Sex)
pretest$Gender <- case_match(pretest$Sex,
                             "Male"~"Male",
                             "Female"~"Female",
                             c("Other","Nonbinary") ~ "Other",
                             .default=NA) %>%
  factor(levels=c("Male","Female","Other"))
age <- pretest$Age
age[!is.na(age)] <- regmatches(age[!is.na(age)],
                       regexpr("[[:digit:]]*",
                               age[!is.na(age)]))
pretest$Age <- as.numeric(age)
pretest$Gaming <- choicesToFactor(preSQ$Gaming,pretest$Gaming)
pretest$Physics <- choicesToFactor(preSQ$Physics,pretest$Physics)
```

Ethnicity. Collapse a couple of categories.
```{r ethnicityOther}
pretest$Ethnicity_9_TEXT[!is.na(pretest$Ethnicity_9)]
```
Qualtrics "helpfully" codes this value as 1/NA. Fix to true logicals.

```{r Logical}
ethids <- grep("Ethnicity_",names(pretest))
for (eth in ethids) {
  name <- names(pretest)[eth]
```

```
  if (any(grepl("TEXT",name))) {
    names(pretest)[eth] <- "Other_TEXT"
  } else {
    names(pretest)[eth] <- ethnames[name]
    pretest[[eth]] <- !is.na(pretest[[eth,drop=TRUE]])
  }
}
pretest[,ethids]
```

```{r collapseEthnicity}
ethids1 <- ethids[-length(ethids)] ## Remove Other_TEXT
ethcols <- as.matrix(as.data.frame(pretest[,ethids1]))
ethnicity <- sapply(1L:nrow(pretest), function(irow) {
    paste(ethnames[ethcols[irow,]], collapse=",")
})
unique(ethnicity)
```

To get a single factor, collapse any combination into "mixed"

```{r recodeEth}
ethnicity[grep(",",ethnicity)] <- "Mixed"
is.na(ethnicity) <- pretest$`Prefer not to say`
ethnicity <- as.factor(ethnicity)
summary(ethnicity)
pretest$Ethnicity <- ethnicity
```

Physics Questions

Need to fix naming issue with Column A-FQ10. Also A-NQ1 and A-FQ4 metadata missing.

```{r Pretest Glitches}
names(pretest)[names(pretest)=="AFQ10"] <- "A-FQ10"
names(preKey)[names(preKey)=="AFQ10"] <- "A-FQ10"
names(preSQ)[names(preSQ)=="AFQ10"] <- "A-FQ10"
preSQ["A-NQ1"] <- potSQ["A-NQ1"]
preSQ["A-FQ4"] <- potSQ["A-FQ4"]
```

```{r encodeAndScore}
for (q in grep("A-",names(pretest),value=TRUE)) {
  vals <- pretest[[q]]
  pretest[[paste(q,"scored",sep="_")]] <- as.numeric(vals==preKey[q])
  pretest[[q]] <- choicesToFactor(preSQ[[q]],vals)
}
for (q in grep("A-",names(ecttest),value=TRUE)) {
  vals <- ecttest[[q]]
  ecttest[[paste(q,"scored",sep="_")]] <- as.numeric(vals==ectKey[q])
  ecttest[[q]] <- choicesToFactor(preSQ[[q]],vals)
}
for (q in grep("A-",names(pottest),value=TRUE)) {
  vals <- pottest[[q]]
  pottest[[paste(q,"scored",sep="_")]] <- as.numeric(vals==preKey[q])
```

```
  pottest[[q]] <- choicesToFactor(preSQ[[q]],vals)
}
pretest
```

```{r subscales}
NearECTcols <- paste(physics.key$ID[physics.key$Form=="A" &
                                    physics.key$`Near/Far`=="Near" &
                                    physics.key$`HL Concept`=="EcT"],
                     "scored",sep="_")
FarECTcols <- paste(physics.key$ID[physics.key$Form=="A" &
                                   physics.key$`Near/Far`=="Far" &
                                   physics.key$`HL Concept`=="EcT"],
                    "scored",sep="_")
NearPOTcols <- paste(physics.key$ID[physics.key$Form=="A" &
                                    physics.key$`Near/Far`=="Near" &
                                    physics.key$`HL Concept`=="PoT"],
                     "scored",sep="_")
FarPOTcols <- paste(physics.key$ID[physics.key$Form=="A" &
                                   physics.key$`Near/Far`=="Far" &
                                   physics.key$`HL Concept`=="PoT"],
                    "scored",sep="_")
```


```{r subscores}
pretest %>%
  mutate(NearECT=rowSums(pretest[,NearECTcols],na.rm=TRUE),
         FarECT=rowSums(pretest[,FarECTcols],na.rm=TRUE),
         NearPOT=rowSums(pretest[,NearPOTcols],na.rm=TRUE),
         FarPOT=rowSums(pretest[,FarPOTcols],na.rm=TRUE)) %>%
  mutate(Near=NearECT+NearPOT,Far=FarECT+FarPOT,
         ECT=NearECT+FarECT, POT=NearPOT+FarPOT,
         PhysicsScore=NearECT+FarECT+NearPOT+FarPOT) ->
  pretest
```

```{r subscoresPOST}
pottest %>%
  mutate(NearPOTpost=rowSums(pretest[,NearPOTcols],na.rm=TRUE),
         FarPOTpost=rowSums(pretest[,FarPOTcols],na.rm=TRUE)) %>%
  mutate(POTpost = NearPOTpost+FarPOTpost) ->
  pottest
ecttest %>%
  mutate(NearECTpost=rowSums(ecttest[,NearECTcols],na.rm=TRUE),
         FarECTpost=rowSums(ecttest[,FarECTcols],na.rm=TRUE)) %>%
  mutate(ECTpost = NearECTpost+FarECTpost) ->
  ecttest
```

IMI Questions

IMI questions are only in the posttest.

```{r revcode}
for (col in IMI.scales$ID[IMI.scales$Reverse]) {
  pottest[[col]] <- 8-pottest[[col]]
  ecttest[[col]] <- 8-ecttest[[col]]
```

```
}
```

```{r IMIsubscales}
pottest$IMI <- rowSums(pottest[,unique(IMI.scales$ID)],na.rm=TRUE)
ecttest$IMI <- rowSums(ecttest[,unique(IMI.scales$ID)],na.rm=TRUE)

for (scale in unique(IMI.scales$Scale)) {
  pottest[[paste("IMI",scale,sep="_")]] <-
    rowSums(pottest[,IMI.scales$ID[IMI.scales$Scale==scale]],na.rm=TRUE)
  ecttest[[paste("IMI",scale,sep="_")]] <-
    rowSums(ecttest[,IMI.scales$ID[IMI.scales$Scale==scale]],na.rm=TRUE)
}


```


PA questions

Again, only in ECT and POT

UGH! Qualtrics was not consistent in the numeric values for these columns. Some were 1--5, some were 11--15, and one was 8--12. WTF. I went back and fixed the coding in Qualtrics.

```{r PArevcode}
for(paq in PA.rev$ID[PA.rev$Reversed]) {
  ecttest[[paq]] <- 5-ecttest[[paq]]
  pottest[[paq]] <- 5-pottest[[paq]]
}
ecttest$PA <- rowSums(ecttest[,PA.rev$ID],na.rm=TRUE)
pottest$PA <- rowSums(pottest[,PA.rev$ID],na.rm=TRUE)
pottest
ecttest
```


Do the join

```{r join}
FSUSFall2022 <- select(PPIDs,!Number) %>%
  full_join(pretest,by="StudyID",suffix=c("",".pre")) %>%
  full_join(ecttest,by="StudyID",suffix=c("",".ect")) %>%
  full_join(pottest,by="StudyID",suffix=c("",".pot"))
```

Check duplicates

```{r dupes}
dupes <- FSUSFall2022$StudyID[duplicated(FSUSFall2022$StudyID)]
dupes
```

Yay! Screening on Finished fixed the duplication problem.

Output the data

```{r output}
outID <- "1DJf7Iidg-GvEXHkrfNPwGTB7VerLSTAUohzZYQOov9E"
write_sheet(FSUSFall2022,outID, sheet="Data")
write_sheet(data.frame(StudyID=dupes),outID,"Duplicate IDs")
write_csv(FSUSFall2022,"data/PPIESFall2022PrePost.csv")
```

Generate cleaning script for the log data.

```{r invalid}
Xids <- c(InvIDs$ID1,InvIDs$ID2)
write_sheet(data.frame(Xids=Xids),outID,"DeleteThese")
```

This is now used to clean the log files.

Merge data from Log Files

BNScores
BNScores is keyed by uid, so we are ready to go. Kill the first two columns, which are unneeded.

```{r BNScores}
BNscores <- read_csv("https://pluto.coe.fsu.edu/Proc4/dongle/data/stats-BigStudy.csv")
BNscores <- select(BNscores,!any_of(c("...1","app")))
```

```{r writeBNScores}
write_sheet(BNscores, outID, "BNscores")
write_csv(BNscores,"data/PPIESFall2022BN.csv")
```

Observables

observables has (uid,Context) as key, need to pivot winder. Timestamp isn't really an
observable, but keep it anyway.

```{r cleanObs}
observables <-
read_csv("https://pluto.coe.fsu.edu/Proc4/dongle/data/ppObs.BigStudy.csv")
obsnames <- names(observables)[-(1:3)]
## Drop useless column and sort by timestamp
observables %>% select(!`...1`) %>% arrange(timestamp) -> observables
```
Add Quits

```{r add quits}
observables$TrophyLevel[is.na(observables$TrophyLevel)] <-

ifelse(is.na(observables$NumberAttempts[is.na(observables$TrophyLevel)]),NA,"quit")
observables$TrophyLevel <- ordered(observables$TrophyLevel,
                                   c("quit","silver","gold"))
```

Multiple Attempts
Problem. What to do with multiple attempts. Two choices, take First, and Take

Last

```{r pivotObs}
obsFirst <-
  pivot_wider(observables, id_cols=uid,
              names_from = context,
              values_from = all_of(obsnames),
              values_fn = function(x) first(x,na_rm=TRUE))
obsLast <-
  pivot_wider(observables, id_cols=uid,
              names_from = context,
              values_from = all_of(obsnames),
              values_fn = function(x) last(x,na_rm=TRUE))
obsMax <-
  pivot_wider(observables, id_cols=uid,
              names_from=context,
              values_from=TrophyLevel,
              values_fn = function(x) max(x,na.rm=FALSE))
```

A lot of N/A observables, so we have total NA columns. Drop these.

```{r dropAllNA}
obsFirst<- obsFirst[,sapply(obsFirst,function(c) !all(is.na(c)))]
obsFirst <-
  obsFirst[,sapply(obsFirst,function(c) !is.numeric(c) || sum(c,na.rm=TRUE) != 0)]
obsLast<- obsLast[,sapply(obsLast,function(c) !all(is.na(c)))]
obsLast<-
  obsLast[,sapply(obsLast,function(c) !is.numeric(c) || sum(c,na.rm=TRUE) != 0)]
obsLast
```

Level names

We currently don't need this, but might later.

```{r levelNames}
levels <-
gsub("TrophyLevel_(.*)","\\1",grep("TrophyLevel_",names(obsFirst),value=TRUE))
levels
```

Add Trophy Counts

```{r TrophyCounting}
obsMax %>% select(!uid) %>% as.matrix() %>% apply(1, function(row)
  table(ordered(row, c("quit","silver","gold")))) %>% t()  -> tMax
obsMax <- cbind(obsMax,tMax)
obsFirst %>% select(ends_with("Trophies")) %>% as.matrix() %>% apply(1,
function(row)
  table(ordered(row, c("quit","silver","gold")))) %>% t()  -> tFirst
obsFirst <- cbind(obsFirst,tFirst)
obsLast %>% select(ends_with("Trophies")) %>% as.matrix() %>% apply(1,
function(row)
  table(ordered(row, c("quit","silver","gold")))) %>% t()  -> tLast
obsLast <- cbind(obsLast,tLast)
```

Write it out
```{r writeObs}
write_sheet(obsFirst,outID,sheet="ObsFirst")
write_sheet(obsLast,outID,sheet="ObsLast")
write_sheet(obsMax,outID,sheet="ObsMax")
write_csv(obsFirst,"data/PPIESFall2022obsFirst.csv")
write_csv(obsLast,"data/PPIESFall2022obsLast.csv")
write_csv(obsMax,"data/PPIESFall2022obsMax.csv")
```

Learning Supports

Key is (uid,context,onWhat). Value is LS_duration. Can summarize with sum.

```{r cleanLS}
learningSupports <-
read_csv("https://pluto.coe.fsu.edu/Proc4/dongle/data/ppLS.BigStudy.csv")
learningSupports <-
  mutate(learningSupports,duration=LS_duration) %>%
  select(!any_of(c("...1","appid","LS_duration"))) %>%
  filter(!is.na(onWhat))
```

```{r lsPivot}
lsWide <-
  pivot_wider(learningSupports,
              id_cols=uid,
              names_from=c(context,onWhat),
              values_from=c(timestamp,duration),
              names_glue="{context}_{onWhat}_{.value}",
              values_fn=list(timestamp=~min(.x, na.rm=TRUE),
                             duration=~ sum(.x,na.rm=TRUE)))
lsWide
```

Kill combinations that never occur.
```{r rmNAs}
all(sapply(lsWide,function(c) !all(is.na(c))))

```

```{r writeLS}
write_sheet(lsWide,outID,"LearningSupport")
write_csv(lsWide,"data/PPIESFall2022ls.csv")
```

Join and Write

```{r BigJoin}
FSUSFall2022BigDaddy <- select(PPIDs,!Number) %>%
  full_join(pretest,by="StudyID",suffix=c("",".pre")) %>%
  full_join(ecttest,by="StudyID",suffix=c("",".ect")) %>%
  full_join(pottest,by="StudyID",suffix=c("",".pot")) %>%
  left_join(BNscores,by=join_by(StudyID==uid),suffix=c("",".bn")) %>%
  left_join(obsFirst,by=join_by(StudyID==uid),suffix=c("",".first")) %>%
  left_join(obsLast,by=join_by(StudyID==uid),suffix=c("",".last")) %>%
  left_join(obsMax,by=join_by(StudyID==uid),suffix=c("",".max")) %>%
```

```
  left_join(lsWide,by=join_by(StudyID==uid),suffix=c("",".ls"))
```

```{r writeBigDaddy}
write_sheet(FSUSFall2022BigDaddy,outID,"BigDaddy")
write_csv(FSUSFall2022BigDaddy,"data/PPIESFall2022Full.csv")
```

SPSS Outpt

```{r savOutput}
names(FSUSFall2022BigDaddy) <- gsub("-","_",names(FSUSFall2022BigDaddy)) %>%
  gsub(" (in seconds)","",.,fixed=TRUE) %>%
  gsub(" ","",.) %>% gsub("(enter)","",.,fixed=TRUE) %>%
  gsub("Yippie!","Yippie",.,fixed=TRUE)
haven::write_sav(FSUSFall2022BigDaddy,"FSUSFall2022BigDaddy.sav")
```

Metadata Export

```{r pretestMetadata}
prechoices <- sapply(preSQ, function(sq) {
  if (length(sq$Choices) >0L) {
    choices <- sapply(sq$Choices,function(ch) ch$Display)
    c(choices,rep(NA_character_,8-length(choices)))
  } else {
    rep(NA_character_,8)
  }
})
prechoices <- t(prechoices)
colnames(prechoices) <- paste(1:ncol(prechoices))
```

```{r pretestMeta2}
premeta <- data.frame(
  name=names(preSQ),
  qid=sapply(preSQ,function(sq) sq$QuestionID),
  qtag=sapply(preSQ,function(sq) sq$DataExportTag),
  text=sapply(preSQ,function(sq) sq$QuestionText),
  description=sapply(preSQ,function(sq) {
    if (is.null(sq$QuestionDescription))
      ""
    else
      sq$QuestionDescription
  }),
  choices=prechoices
)
premeta
```

ECT

IMI meta-data is completely different from the rest, so needs special handling.

```{r ecttestMetadata}
ectchoices <- sapply(ectSQ[names(ectSQ)!="IMI"], function(sq) {
  if (length(sq$Choices) >0L) {
```

```
    choices <- sapply(sq$Choices,function(ch) ch$Display)
    c(choices,rep(NA_character_,8-length(choices)))
  } else {
    rep(NA_character_,8)
  }
})
ectchoices <- t(ectchoices)
colnames(ectchoices) <- paste(1:ncol(ectchoices))

```


```{r ecttestMeta2}
ectmeta <- data.frame(
  name=paste(names(ectSQ[names(ectSQ)!="IMI"]),"ect",sep="."),
  qid=sapply(ectSQ[names(ectSQ)!="IMI"],function(sq) sq$QuestionID),
  qtag=sapply(ectSQ[names(ectSQ)!="IMI"],function(sq) sq$DataExportTag),
  text=sapply(ectSQ[names(ectSQ)!="IMI"],function(sq) sq$QuestionText),
  description=sapply(ectSQ[names(ectSQ)!="IMI"],function(sq) {
    if (is.null(sq$QuestionDescription))
      ""
    else
      sq$QuestionDescription
  }),
  choices=ectchoices
)


ectmeta
```


```{r IMImeta}
IMISQ <- ectSQ[["IMI"]]
IMI.options <- c(sapply(IMISQ$Answers,function(ch) ch$Display),"8"=NA_character_)

imimeta <- data.frame(
  name=names(IMI.stems),
  qid=IMISQ$QuestionID,
  qtag=IMISQ$DataExportTag,
  text=IMI.stems,
  description=IMI.stems,
  choices=t(replicate(length(IMI.stems),IMI.options))
)
imimeta

```

POT metadata

```{r pottestMetadata}
potchoices <- sapply(potSQ[names(potSQ)!="IMI"], function(sq) {
  if (length(sq$Choices) >0L) {
    choices <- sapply(sq$Choices,function(ch) ch$Display)
    c(choices,rep(NA_character_,8-length(choices)))
  } else {
    rep(NA_character_,8)
  }
```

```
})
potchoices <- t(potchoices)
colnames(ectchoices) <- paste(1:ncol(ectchoices))

```


```{r pottestMeta2}
potmeta <- data.frame(
  name=paste(names(potSQ[names(potSQ)!="IMI"]),"pot",sep="."),
  qid=sapply(potSQ[names(potSQ)!="IMI"],function(sq) sq$QuestionID),
  qtag=sapply(potSQ[names(potSQ)!="IMI"],function(sq) sq$DataExportTag),
  text=sapply(potSQ[names(potSQ)!="IMI"],function(sq) sq$QuestionText),
  description=sapply(potSQ[names(potSQ)!="IMI"],function(sq) {
    if (is.null(sq$QuestionDescription))
      ""
    else
      sq$QuestionDescription
  }),
  choices=potchoices
)


potmeta
```
Form B metadata

```{r postBMetadata}
bchoices <- sapply(postBSQ[names(postBSQ)!="IMI"], function(sq) {
  if (length(sq$Choices) >0L) {
    choices <- sapply(sq$Choices,function(ch) ch$Display)
    c(choices,rep(NA_character_,8-length(choices)))
  } else {
    rep(NA_character_,8)
  }
})
bchoices <- t(bchoices)
colnames(bchoices) <- paste(1:ncol(bchoices))

```


```{r postBMeta2}
bmeta <- data.frame(
  name=paste(names(postBSQ[names(postBSQ)!="IMI"]),"pot",sep="."),
  qid=sapply(postBSQ[names(postBSQ)!="IMI"],function(sq) sq$QuestionID),
  qtag=sapply(postBSQ[names(postBSQ)!="IMI"],function(sq) sq$DataExportTag),
  text=sapply(postBSQ[names(postBSQ)!="IMI"],function(sq) sq$QuestionText),
  description=sapply(postBSQ[names(postBSQ)!="IMI"],function(sq) {
    if (is.null(sq$QuestionDescription))
      ""
    else
      sq$QuestionDescription
  }),
  choices=bchoices
)
```

```
bmeta
```

Write it out

```{r writemeta}
write_sheet(rbind(premeta,ectmeta,imimeta,potmeta),outID,"Metadata")

```