Tutorial for Chapter 5: The PRTIII Test

Tara L. Valladares & Moritz Heene

July 22, 2020

# Preliminary steps before you can run the PRTIII analysis

1. Create a folder called “PRTIII\_analysis” on your desktop, for example. Download the data sets ARMChapter5\_PRTIII.csv and ARMChapter5.csv, and the R code Chapter\_5\_PRTIII\_eRm.R from the website and save both in *that* folder. This folder will serve as your working directory containing all files you need to conduct the analysis and to store optional output (i.e., code, data, and figures). If the R code and the .csv files are not in the same folder, you will not be able to load the data using the code below.
2. Open the file Chapter\_5\_PRTIII\_eRm.R in RStudio by clicking on the file. This will open the file in RStudio.
3. Go to “Session” “Set Working Directory” “To Source File Location”. This defines the folder you named above as your working directory in which you are currently working and where R expects all data sets to be.

You are now ready to run the Rasch analysis of the PRTIII data used in ARM4 Chapter 5. We are going to compare the person and item measures from the PRTIII to that of the BLOT. Please use the following instructions and explanations of the R code.

# Rasch Model Invariance Analysis: Data Preparation

First, we will load the necessary packages we will need for this analysis. If you have not yet, you will need to install the packages using the install.packages() function below. This will only need to be done once. Then, we will load the packages into R using library(). This will need to be done each time you open R.

# Installing the required packages for this analysis.   
# You will only need to do this once.  
# Uncomment the code below to run the install.packages function.  
# install.packages(c("eRm", "dplyr", "ggplot2", "pairwise", "TAM", "psych"))  
  
# Load the packages required for the analysis  
library(eRm)  
library(dplyr)  
library(ggplot2)

Now we will load the datasets we need. The first file is called ARMChapter5\_PRTII.csv, it has no header (header = FALSE) and a semi-colon is used as the field separator (sep = ";").

# Reading in the comma-seperated data set  
df <- read.csv("ARMChapter5\_PRTIII.csv", header = FALSE, sep = ";")

Let’s remind ourselves about the Piagetian Reasoning Task - the PRTIII. It consists of 13 short-answer items administered to 150 persons; the same high school students who took the BLOT test in the Chapter 4 data set. The items were scored dichtomously as 0 or 1.

We will need to preprocess and format our data. The following lines of code define column names of the data frame and then create an object called “prt\_items” containing the PRTIII items we need for the Rasch analysis.

# Defining the names of the columns  
colnames(df)[2:14] <- paste("PRTIII", 1:13, sep="\_")   
# Selecting only the PRTIII items  
prt\_items <- dplyr::select(df, PRTIII\_1:PRTIII\_13)   
# Looking at the first five rows of the PRTIII dataframe  
head(prt\_items)

## PRTIII\_1 PRTIII\_2 PRTIII\_3 PRTIII\_4 PRTIII\_5 PRTIII\_6 PRTIII\_7 PRTIII\_8  
## 1 1 0 0 1 1 0 1 1  
## 2 1 1 1 1 1 1 1 1  
## 3 1 0 1 1 1 0 1 1  
## 4 1 0 1 1 1 0 1 1  
## 5 1 1 1 0 0 1 0 1  
## 6 1 0 0 1 0 0 1 1  
## PRTIII\_9 PRTIII\_10 PRTIII\_11 PRTIII\_12 PRTIII\_13  
## 1 1 1 1 1 0  
## 2 0 1 1 1 1  
## 3 1 0 0 1 1  
## 4 1 1 1 1 0  
## 5 1 1 1 1 1  
## 6 0 0 0 0 0

We will also load the BLOT dataset that we used in *Tutorial for Chapter 5: BLOT Ivariance Rasch Analysis with the eRm package*. We’ll use the same code as above, but this time for the ARMChapter5.csv.

# Reading in the comma-seperated data set for the BLOT data  
df2 <- read.csv("ARMChapter5.csv", header = FALSE, sep = ";")   
# Defining the name of the first column of the data frame called "df2"  
colnames(df2)[1] <- c("group")   
colnames(df2)[2] <- c("gender")   
# Defining the names of the columns referring to the items  
colnames(df2)[3:37] <- paste("Blot", (1:35), sep="\_")   
# Selecting only the BLOT items  
blot\_items <- dplyr::select(df2, Blot\_1:Blot\_35)   
# Looking at the first five rows of the blot dataframe  
head(blot\_items)

## Blot\_1 Blot\_2 Blot\_3 Blot\_4 Blot\_5 Blot\_6 Blot\_7 Blot\_8 Blot\_9 Blot\_10  
## 1 1 1 1 1 1 1 1 1 1 1  
## 2 1 1 1 1 1 1 1 1 1 1  
## 3 1 1 0 1 0 1 1 1 1 1  
## 4 1 1 1 1 1 1 1 1 1 1  
## 5 1 1 1 1 1 1 1 1 1 1  
## 6 1 1 1 1 1 1 1 1 1 1  
## Blot\_11 Blot\_12 Blot\_13 Blot\_14 Blot\_15 Blot\_16 Blot\_17 Blot\_18 Blot\_19  
## 1 0 1 1 0 1 0 1 1 0  
## 2 1 1 1 1 1 1 1 1 1  
## 3 1 1 1 1 0 1 1 1 1  
## 4 1 1 1 1 1 1 1 1 1  
## 5 1 1 0 1 1 1 1 1 1  
## 6 1 1 1 0 1 1 1 1 0  
## Blot\_20 Blot\_21 Blot\_22 Blot\_23 Blot\_24 Blot\_25 Blot\_26 Blot\_27 Blot\_28  
## 1 1 0 1 1 1 1 1 1 1  
## 2 1 1 1 1 1 1 1 1 0  
## 3 1 0 1 1 1 1 1 1 0  
## 4 1 1 0 1 1 1 1 1 1  
## 5 1 0 1 1 1 1 1 1 1  
## 6 1 0 1 1 1 1 1 1 1  
## Blot\_29 Blot\_30 Blot\_31 Blot\_32 Blot\_33 Blot\_34 Blot\_35  
## 1 1 0 1 1 1 1 1  
## 2 1 1 1 1 1 1 1  
## 3 1 0 1 1 1 1 1  
## 4 1 1 1 1 1 1 1  
## 5 1 1 1 1 1 1 1  
## 6 1 1 1 1 1 1 1

We are now ready to run the actual Rasch analyses. In accordance with the corresponding Winsteps analysis, we will

* Fit a Rasch model for the PRTIII
* Examine item fit statistics for the PRTIII
* Examine fit plots for the PRTIII
* Compare the person estimates from the PRTIII and the BLOT test

# The PRTIII Rasch Model Analysis

We start by estimating the Rasch model parameters for the PRTIII and storing them in an object called fit\_rasch\_prt. The following line uses the *eRm* package to run a basic Rasch model analysis of the data on the object prt\_items and save the results in an object called fit\_rasch\_prt.

# Estimates Rasch model parameters  
fit\_rasch\_prt <- RM(prt\_items)

Let’s first take a look at the item parameters for the PRTIII. Similar to before, we’ll first have to estimate the person parameters using the person.paramter function.

# Estimating person parameters for each observed raw score.   
pparameters\_prt <- eRm::person.parameter(fit\_rasch\_prt)

In a second step, we reuse the previously estimated person parameters to estimate item fit statistics. We will look at it using the print function.

# Estimate item parameters  
item\_fit\_prt <- eRm::itemfit(pparameters\_prt)   
print(item\_fit\_prt)

##   
## Itemfit Statistics:   
## Chisq df p-value Outfit MSQ Infit MSQ Outfit t Infit t Discrim  
## PRTIII\_1 119.444 142 0.916 0.835 1.074 -0.177 0.505 0.195  
## PRTIII\_2 68.516 142 1.000 0.479 0.725 -0.397 -0.926 0.321  
## PRTIII\_3 145.301 142 0.408 1.016 1.117 0.147 1.088 0.402  
## PRTIII\_4 133.547 142 0.682 0.934 0.915 -0.356 -0.867 0.533  
## PRTIII\_5 123.965 142 0.860 0.867 0.901 -0.799 -1.012 0.548  
## PRTIII\_6 165.880 142 0.083 1.160 1.116 0.662 1.064 0.381  
## PRTIII\_7 104.243 142 0.993 0.729 0.804 -1.717 -2.124 0.610  
## PRTIII\_8 287.893 142 0.000 2.013 0.977 2.001 -0.124 0.230  
## PRTIII\_9 168.523 142 0.064 1.178 1.002 0.851 0.058 0.452  
## PRTIII\_10 119.124 142 0.919 0.833 0.966 -0.981 -0.325 0.547  
## PRTIII\_11 99.356 142 0.997 0.695 0.823 -1.407 -1.757 0.606  
## PRTIII\_12 115.707 142 0.948 0.809 0.938 -1.199 -0.611 0.565  
## PRTIII\_13 75.886 142 1.000 0.531 0.671 -0.714 -1.811 0.446

Notice that the *eRm* item fit output somewhat deviates from Winsteps’ item fit report. While they both report the unstandardized and standardized MNSQ infit and outfit statistics, *eRm* also provides chi-square based item fit statistics. The column Discrim shows the corrected item-raw score correlations correcting for item overlap as well as for the drop in score reliability when an item is removed from the scale to calculate item-total correlations (see Cureton, 1966, for details).

The outfit mean-square for item 8 is 2.01, which is pretty big. The expected value of a mean-square is 1.0. Values above 2.0 degrade measurement. The infit MSQ is much more reasonable at 0.977. We can also sort the itemfit statistics by the chi-squared *p*-value. This will print worse fitting items at the top.

print(item\_fit\_prt, sort\_by = "p")

##   
## Itemfit Statistics:   
## Chisq df p-value Outfit MSQ Infit MSQ Outfit t Infit t Discrim  
## PRTIII\_8 287.893 142 0.000 2.013 0.977 2.001 -0.124 0.230  
## PRTIII\_9 168.523 142 0.064 1.178 1.002 0.851 0.058 0.452  
## PRTIII\_6 165.880 142 0.083 1.160 1.116 0.662 1.064 0.381  
## PRTIII\_3 145.301 142 0.408 1.016 1.117 0.147 1.088 0.402  
## PRTIII\_4 133.547 142 0.682 0.934 0.915 -0.356 -0.867 0.533  
## PRTIII\_5 123.965 142 0.860 0.867 0.901 -0.799 -1.012 0.548  
## PRTIII\_1 119.444 142 0.916 0.835 1.074 -0.177 0.505 0.195  
## PRTIII\_10 119.124 142 0.919 0.833 0.966 -0.981 -0.325 0.547  
## PRTIII\_12 115.707 142 0.948 0.809 0.938 -1.199 -0.611 0.565  
## PRTIII\_7 104.243 142 0.993 0.729 0.804 -1.717 -2.124 0.610  
## PRTIII\_11 99.356 142 0.997 0.695 0.823 -1.407 -1.757 0.606  
## PRTIII\_2 68.516 142 1.000 0.479 0.725 -0.397 -0.926 0.321  
## PRTIII\_13 75.886 142 1.000 0.531 0.671 -0.714 -1.811 0.446

None of the other items are particularly alarming like Item 8 is, whether we look at Outfit or Infit. If you’d like to play around with the sorting function, you can sort specifically by Outfit or Infit by using sort\_by = "outfit\_MSQ" or sort\_by = "infit\_MSQ" in the print function. In any case, we should do some investigating to see what is going on with Item 8.

We are going to examine the residuals for each person’s responses to each item. Residuals are the difference between what the model predicts and what actually occurs in the data. A good fitting model will generally have smaller residuals. A large residual indicates an unexpected response - there is a big difference between what the person did and what the model predicted. Large standardized residuals contribute to large outfit mean-square fit statistics.

The standardized residuals were already calculated when we calculated item fit. We just need to call them using the $ operator. We’ll use the head function to just look at the first five rows.

# Print the first five rows of standardized residuals  
head(item\_fit\_prt$st.res)

## PRTIII\_1 PRTIII\_2 PRTIII\_3 PRTIII\_4 PRTIII\_5 PRTIII\_6 PRTIII\_7  
## P1 0.12187574 -0.3447472 -1.2223860 0.4089461 0.4598525 -1.1420029 0.3782032  
## P2 0.03697966 0.8801261 0.2482203 0.1240828 0.1395289 0.2656920 0.1147548  
## P3 0.12187574 -0.3447472 0.8180722 0.4089461 0.4598525 -1.1420029 0.3782032  
## P4 0.09087063 -0.4623751 0.6099552 0.3049105 0.3428663 -1.5316549 0.2819886  
## P5 0.09087063 2.1627464 0.6099552 -3.2796508 -2.9165884 0.6528886 -3.5462432  
## P6 0.40050377 -0.1049087 -0.3719795 1.3438644 -0.6617471 -0.3475184 1.2428381  
## PRTIII\_8 PRTIII\_9 PRTIII\_10 PRTIII\_11 PRTIII\_12 PRTIII\_13  
## P1 0.14418975 0.7178568 0.3708728 0.7657327 0.4509298 -0.5097503  
## P2 0.04375019 -4.5910979 0.1125306 0.2323394 0.1368216 0.5952346  
## P3 0.14418975 0.7178568 -2.6963419 -1.3059388 0.4509298 1.9617449  
## P4 0.10750797 0.5352345 0.2765230 0.5709308 0.3362136 -0.6836773  
## P5 0.10750797 0.5352345 0.2765230 0.5709308 0.3362136 1.4626783  
## P6 0.47383130 -0.4239092 -0.8205133 -0.3974051 -0.6748414 -0.1551201

As you can see, the standardized residuals are outputted in a matrix where columns are items and rows are persons. It’s not easy to see which residuals are high, though from this format. So we are going to transform our data into long format and then sort it based on the size of the residual.

# Reshape data into long format  
prt\_res<-reshape2::melt(item\_fit\_prt$st.res)  
# add new column names  
colnames(prt\_res) <- c("person", "item", "st.res")  
# short the data, and see the top 10 rows  
prt\_res[order(prt\_res$st.res),][1:10,]

## person item st.res  
## 1054 P54 PRTIII\_8 -13.456005  
## 1146 P2 PRTIII\_9 -4.591098  
## 1029 P29 PRTIII\_8 -4.271502  
## 1081 P82 PRTIII\_8 -4.271502  
## 85 P87 PRTIII\_1 -4.034655  
## 96 P99 PRTIII\_1 -4.034655  
## 863 P5 PRTIII\_7 -3.546243  
## 434 P5 PRTIII\_4 -3.279651  
## 451 P23 PRTIII\_4 -3.279651  
## 453 P25 PRTIII\_4 -3.279651

We can see that the response with the highest residual (the most unexpected or erratic result) is by person 54 to item 8 of the PRTIII. Let’s see what they responded and the difficulty of the item. To see the person’s response, we’ll look at the raw data (prt\_items). Remember that our rows are persons and our columns are items, so we’ll use the brackets ([]) to subset the 54th row and the 8th column. Our person difficulties are stored in pparameters\_prt under thetapar. “Theta” is the mathematical model term for person abilities, which is why we’re calling thetapar (theta parameters). Our item difficulties are in the model object we saved before (fit\_rasch\_prt). “Eta” is the mathematical model term for item difficulties, which is why we’re calling etapar (“eta parameters”) from the fit\_rasch\_prt object.

# Person 54's response to item 8  
prt\_items[54,8]

## [1] 0

# Person 54's ability score  
pparameters\_prt$thetapar$NAgroup1["P54"]

## P54   
## 2.505028

# Difficulty of the items  
fit\_rasch\_prt$etapar

## PRTIII\_2 PRTIII\_3 PRTIII\_4 PRTIII\_5 PRTIII\_6 PRTIII\_7 PRTIII\_8   
## 3.3093158 0.7778184 -0.6089159 -0.3742712 0.9138605 -0.7652196 -2.6938226   
## PRTIII\_9 PRTIII\_10 PRTIII\_11 PRTIII\_12 PRTIII\_13   
## 0.5164575 -0.8043643 0.6455834 -0.4134595 2.5270964

Item 8 was really easy; it had a difficulty of -2.6938226.

Person 54’s response was a wrong answer, 0, which is strange because they had a high ability score of 2.5050283. Since the item was so easy for person 54, a correct answer, 1, was expected by the Rasch model. That’s why the residual is so big.

Is 54’s response of “0” believable? What should we do? Qualitatively, we should:

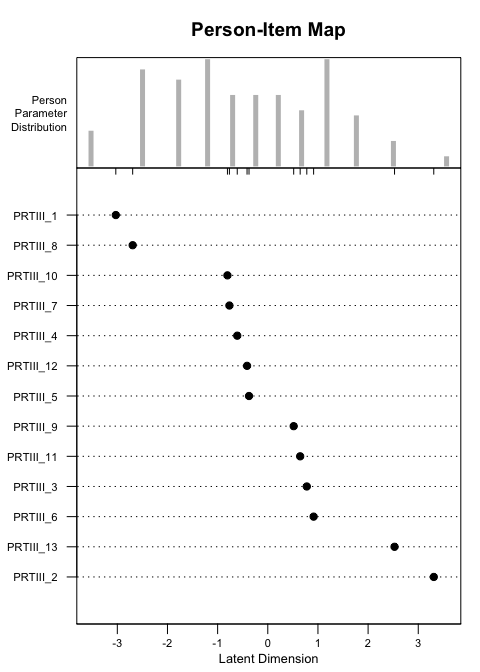
1. Go back to this child’s answer sheet to see if the answer to Item 8 was scored correctly by the investigator (Was the answer marked wrong when it was really correct?) and/or,
2. Check that the data point in the file corresponds with the mark on the answer sheet (Was the mark mis-keyed into the data file?)

If we can’t do that, or we have good reasons to suspect that this data point is not reliable, we could code that response as missing data. We will not do that now. But if we wanted to do that, we could run the following code: prt\_items[54,8] <- NA. It is important to note that *there is no undue button in R*. If you want to change your data like this, it may be prudent to create a copy of the data and then change the missing response in the new object (like, prt\_items\_fixed <- prt\_items and then prt\_items\_fixed[54,8] <- NA). Changing your object in R will not change the original data in the .csv file, though, so you will have a backup there. For now, we will move on with the analysis without changing the data.

## Plots

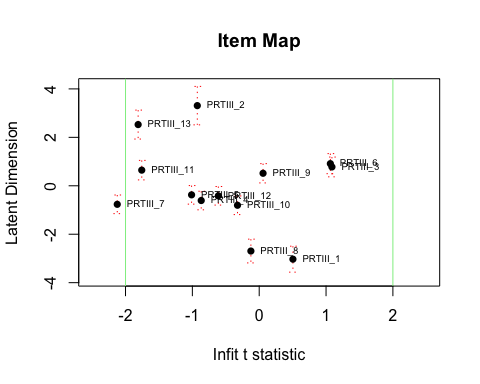
Now let’s take a look at some plots. First we’ll look at the Wright map which plots the difficulties of the items against the distribution of the person abilities. These graphs are also referred to as person-item maps. The argument sort=TRUE orders the items in increasing difficulty to make the graph easier to read. We can use the plot to see how well the item difficulty distribution matches our person ability distribution. Our map below looks pretty good, though we can see that most of our participants fall on the lower end of the scale. Another item with the difficulty of -2 could be helpful to add if we made a new version of the scale.

# Wright map  
plotPImap(fit\_rasch\_prt, irug = TRUE, sort = TRUE)



Now let’s look at the pathway map. This is similar to the bubble maps in ARMsteps. The pathway map displays the location of each item against its infit *t*-statistic. This graph lets us examine how well items fit the Rasch model at different values of the latent dimension. Items are more difficult as they move higher on the y-axis, and they are less predictable as they move to the right on the x-axis.

# Pathway map with  
# 95% confidence intervals for the item parameters  
plotPWmap(fit\_rasch\_prt, itemCI = list())



Notice that the *eRm* version of the pathway map also prints 95% confidence intervals for the item parameters. The red dotted lines represent the confidence intervals for the item difficulty estimation. The broader the bands, the less precise our estimate of the item difficulty is.

The Rasch model expects all of our items to have an infit *t*-statistic of zero, but this is an impossible standard for the whole test. The green vertical lines at -2 and +2 roughly cut-off items that are not behaving as the model expects them to. Items outside of this cut-off are likely to be misfitting.

We can see that most of our items lie within the infit bounds, which is good. Item 7 has some suspicious infit, but our other items look pretty good.

## Comparing the BLOT scores to the PRTIII scores

Now let’s see how the PRTIII scores compare to the BLOT scores. Remember that the same 150 students took both tests, so this is a meaningful comparison. First, we’re going to run a Rasch model for the BLOT items. These are the same functions that we used above. For more information about the fit of the BLOT test, see the Chapter 5: BLOT Invariance tutorial.

# Estimate Rasch model parameters for the BLOT items  
fit\_rasch\_blot <- RM(blot\_items)   
# Estimate person fit to get the person ability scores  
pparameters\_blot <- person.parameter(fit\_rasch\_blot)

First, we’ll need to combine our person parameters from the BLOT test and the PRTIII test into one data frame. This format will help us keep our data organized and is necessary for the plotting we will do below. This code is a bit complicated - we are using the merge and the reduce functions to combine four data frames that contain the person parameters and standard errors of the person parameters from both tests. We can’t just bind the vectors together as they are different lengths - the standard errors are not estimated for students who had full zero or full correct responses. Don’t worry too much about understanding this code; with practice R will come much easier.

# Create a data frame of ability scores and standard errors of the estimate  
combined\_data <- Reduce(function(x, y) merge(x, y, all=TRUE),   
 list(  
 data.frame(id = names(coef(pparameters\_prt, extrapolated = TRUE)),  
 theta\_prt = coef(pparameters\_prt, extrapolated = TRUE),  
 row.names=NULL),  
   
 data.frame(id=names(pparameters\_prt$se.theta$NAgroup1),  
 se\_prt= pparameters\_prt$se.theta$NAgroup1,   
 row.names=NULL),  
   
 data.frame(id=names(coef(pparameters\_blot, extrapolated = TRUE)),  
 theta\_blot = coef(pparameters\_blot, extrapolated = TRUE),   
 row.names=NULL),  
   
 data.frame(id=names(pparameters\_blot$se.theta$NAgroup1),  
 se\_blot = pparameters\_blot$se.theta$NAgroup1,   
 row.names=NULL)  
 ))

Now let’s take a look at a table of the person abilities from both tests next to each other. We’ll look at the first few rows using the head function.

head(combined\_data[c("id","theta\_blot", "theta\_prt")])

## id theta\_blot theta\_prt  
## 1 P1 1.8294467 1.1794277  
## 2 P10 3.1688120 4.7009431  
## 3 P100 0.6217320 -1.7768052  
## 4 P101 0.9185662 -0.2402373  
## 5 P102 0.7674638 -1.2000147  
## 6 P103 0.3416322 -1.7768052

Even if we printed the whole data frame out, it would be hard to tell how close the BLOT and PRTIII estimates were to each other. Not only would this be a lot of data, but the two scores might be on slightly different scales. We can solve the first problem by making a graph. The second problem we can solve by adjusting one of the scores to match the scale of the other. First we will calculate the mean of the PRTIII scores and the mean of the BLOT scores. Then, we will add the difference between the BLOT and PRTIII to the PRTIII data (to put in on the BLOT scale) and the difference between the PRTIII and the BLOT to the BLOT data (to put in on the BLOT scale). These will be saved in new columns, theta\_blot\_adjusted and theta\_prt\_adjusted.

# Calculate the average of the scores  
(mean\_prt <- mean(combined\_data$theta\_prt, na.rm = TRUE))

## [1] -0.6084111

(mean\_blot <- mean(combined\_data$theta\_blot, na.rm = TRUE))

## [1] 1.624069

# Adjust the PRTIII person measures by incorporating the mean difference  
combined\_data$theta\_prt\_adjusted <- combined\_data$theta\_prt + (mean\_blot - mean\_prt)  
# Adjust the BLOT person measures by incorporating the opposite mean difference  
combined\_data$theta\_blot\_adjusted <- combined\_data$theta\_blot + (mean\_prt - mean\_blot)

The mean of the PRTIII scores is -0.61 and the mean of the BLOT scores is 1.62. Let’s start by looking at the BLOT data and the PRTIII data that is on the BLOT scale. We’ll calculate a kind of confidence interval so we can get an idea of if the difference between the BLOT and adjusted PRTIII data is of practical significance or not. Don’t worry too much about the code and calculations here.

# CI LOWER - BLOT scale  
combined\_data$blot\_CI\_low <- (combined\_data$theta\_blot+combined\_data$theta\_prt\_adjusted)/2 - sqrt((combined\_data$se\_prt)^2 + (combined\_data$se\_blot)^2)  
  
# CI UPPER - BLOT scale  
combined\_data$blot\_CI\_high <- (combined\_data$theta\_blot+combined\_data$theta\_prt\_adjusted)/2 + sqrt((combined\_data$se\_prt)^2 + (combined\_data$se\_blot)^2)  
  
# plot - BLOT scale  
ggplot(combined\_data) +   
 # Label the axes  
 xlab("BLOT Estimates") + ylab("PRTIII Adjusted Estimates") +  
 ggtitle("Common person linking BLOT and PRTIII") +  
 # plot scatter plot  
 geom\_point(aes(x=theta\_blot, y=theta\_prt\_adjusted),color="blue") +  
 # add in lower CI based on a second degree polynomial  
 geom\_smooth(aes(x=blot\_CI\_low, y=blot\_CI\_high),color="black",  
 fullrange =TRUE, se=FALSE,  
 method="lm", formula = y ~ poly(x, 2)) +  
 # add in upper CI based on a second degree polynomial  
 geom\_smooth(aes(x=blot\_CI\_high, y=blot\_CI\_low),color="black",  
 fullrange =TRUE, se=FALSE,  
 method="lm", formula = y ~ poly(x, 2)  
 )

## Warning: Removed 10 rows containing non-finite values (stat\_smooth).  
  
## Warning: Removed 10 rows containing non-finite values (stat\_smooth).

