

---

# STAT S400: Statistical Computing with R

## Assignment #2 – Constructing and Testing Your Own Functions

In this assignment you are introduced to the very basics of creating your own functions. This particular topic will be revisited in gradually increasing levels of complexity as the semester progresses.

### Four Suggestions

First, the code displayed and run in the following pages is contained in the script file **Assignment2.R**, which is located in the Assignment 2 folder on the course website. Save this script file to your course directory, for example,

```
z:\My Documents\STAT400\Assign2\Assignment2.R
```

and then, when you open R, set the working directory to

```
z:\My Documents\STAT400\Assign2
```

Then open **Assignment2.R** in R and run the code in this script file, command-by-command, to follow along with the discussions included in these notes. This will save you a *lot* of time.

Second, as you will come to notice, in each new assignment you will continue to be introduced to new R functions. For some of these a discussion of the functions and what they do are given. There are others that are simply used without any discussion. Whenever the latter occurs, you are encouraged to open the relevant documentation pages (using **help** or **?**) and find out more about the functions in case you find a need for them later.

Third, when going through this and later assignment notes, begin by first *browsing* through the whole document fairly quickly. Then make use of the script file to follow the R-related discussions and illustrations (you may choose to do this alongside the browsing if you wish). Next, take a look at the assignment questions/problems and go back to relevant sections of the notes (this may include the Exploratory Exercises section) and begin working on the assignment. Finally, if you feel like wandering or looking into particular sections of interest (and you have time) you can do so – as a rough cutoff for wandering time, allow yourself a maximum of 6 hours (total) for time spent on each assignment before moving on to the next assignment. There will be occasions later in the course where you may get an opportunity to revisit a particular topic.

Fourth, and this is very important. You are not expected to remember every single R function or capability you encounter as you go through this course. You will not be tested on your ability to memorize “things.” However, you will be expected to be able to know how to find out information on your own, at a gradually increasing rate.

### Relevant Resources for this Assignment

**The R Book:** Chapters 2, 7 and 8

**Naming Conventions in R:** Full article.

**An Introduction to R:** Chapter 8 and 10

**R Language Definition:** Chapters 1-3

**Your favorite elementary statistics text:** Relevant topics/chapters.

## Functions in R

Working in R is pretty much all about working with what are referred to as *objects*. These objects contain data in a variety of forms: in the form we commonly think of data as being – numbers and other observations from a study – or other forms we might not naturally think of as being data. So as to avoid confusion, in this course we will always refer to an object containing “data as we know it to be” either as a particular type of *data structure* (for example, a *vector*, *factor*, *data frame*, *list*, etc.) or simply *data*. We’ll get into more details about these structures as the course progresses.

One *class* of objects in R is the *function*. You have already seen some examples of these in `c`, `length`, `sum`, `sqrt`, `qt`, `round`, `cat`, and `paste`. It turns out that the *operators* “+”, “-”, “\*”, “/”, “^”, and “<-” are also functions, but these are used in a different way than the “usual” function.

In algebra a function  $f$  is defined to be a rule that takes a number  $x$  from a set  $X$  and assigns to it exactly one number  $y = f(x)$  in another set  $Y$ . In programming, the definition of a function allows for quite a bit of flexibility. Very loosely, in R, a function is a specialized object that takes in a collection of (other) objects – the *arguments* of the function – and then performs a collection of tasks on (the contents of) these objects.

So, making use of code from the previous assignment for some simple illustrations, in the statement

```
x <- c(74.3, 69.9, 65.7, 70.1, 56.8, 69.2, 73.2, 49.6, 73.4, 77.8)
```

the *combine function*, `c`, combines the included numbers into a *vector* of numbers and the *assignment operator*, `<-`, assigns this vector a name, `x`. So, the object `x` is a (numeric) vector. This object can then be treated as a *variable*, in the usual sense of the word. Simply put, a function needs a name, a list of arguments (the input), and instructions (the rules) on how to produce the output (in whatever form it may be). Then, when the function is *called* (used) the name is entered and the arguments are specified within parentheses. Here’s a simple example of preparing a function of your own.

## Preparing a User-defined Function

Recall the code used to compute the confidence interval for a population mean. A bit of statistics background on this method is appropriate before beginning.

**The  $t$ -interval for  $\mu$ :** Let  $X$  be a normally distributed random variable with mean  $\mu$  and standard deviation  $\sigma$ . Suppose the population standard deviation,  $\sigma$ , is *not known*. Let  $\bar{x}$  and  $s$  denote the sample mean and sample standard deviation, respectively, of a random sample of size  $n$  for  $X$ . Then, a  $100(1 - \alpha)\%$  confidence interval for  $\mu$  is given by

$$\bar{x} - t_{\alpha/2} s / \sqrt{n} \leq \mu \leq \bar{x} + t_{\alpha/2} s / \sqrt{n}, \quad (1)$$

where the notation  $t_{\alpha/2}$  denotes the *quantile* that satisfies the right-tailed probability  $P(t > t_{\alpha/2}) = \alpha/2$  using the  $t$  distribution with  $n - 1$  degrees of freedom.

The “formula” in the inequality 1 provides complete instructions on how to perform the relevant computations to produce the confidence interval. All that remains is to put these instructions in a form that the R interpreter can use – the code from Assignment 1 does exactly that. What we can do is to package this in a function that can be used again and again on any (legal) sample that it is given. The following is a development of a (very basic) function.

Begin by assigning the function a name – we’ll use the *period.separated* naming-convention for functions – and listing the arguments to be passed into the function. This is done using, for example,

```
t.confidence <- function(x, confidence, dec){ ...relevant instructions... }
```

Here, the “...relevant instructions...” within the *function body* are taken from Assignment 1. In these instructions `x` is the object name the function `t.confidence` will “recognize” as identifying the sample, `confidence` the confidence level, and `dec` the number of decimal places to which the answer is displayed.

So, the function is provided the sample and the confidence level and has to be instructed how to compute the desired interval. The code from Assignment 1 provides adequate *bare-bones* instructions, and the function can be defined as follows.

---

```

t.confidence <- function(x, confidence, dec)
#
# Argument details
#           x:  A numeric sample
#   confidence: A number between 0 and 1
#           dec: A non-negative integer
#
{  # Begin function body
    # Get the sample size
    n <- length(x)
    # Calculate the sample mean
    xBar <- sum(x)/n
    # Calculate the sample standard deviation
    s <- sqrt(sum((x - xBar)^2)/(n - 1))
    # Obtain the corresponding significance level
    alpha <- 1 - confidence
    # Obtain the upper-tail critical value
    critValue <- qt(p = alpha/2, df = n-1, lower.tail = FALSE)
    # Calculate the maximum error of estimate
    error <- critValue*s/sqrt(n)
    # Compute the interval bounds, rounded to "dec" decimal places
    bounds <- round(xBar + c(-error, error), dec)
    # Gather, format, and output the results
    cat(paste("\n", as.integer(100*confidence),
              "% confidence interval for the population mean: ",
              "[", bounds[1], ", ", bounds[2], "]\n\n", sep = ""))
  }  # End function body

```

You might notice that some small changes have been made to the Assignment 1 code, see if you can spot them and figure out if they change much in what the this code will compute and output. You can test the code placed in the body of the function by first beginning with bare-bones code as in Assignment 1 (assigning a sample to `x`, a confidence level to `confidence`, and a non-negative integer to `dec`) and then running *only* the code *within the braces* just as you did for Assignment 1. You should notice one very important fact in this code, there is no mention *inside the function* of what `x` contains. Also, pay close attention to the manner in which indentations are used in the these (and later) code – we’ll use four spaces for each level of indentations as appropriate.

Now load this function into the workspace by highlighting the function code and running it; see Figure 1 and see Figure 2 for what shows up on the console. Next, run

```

> ls()

[1] "t.confidence"

```

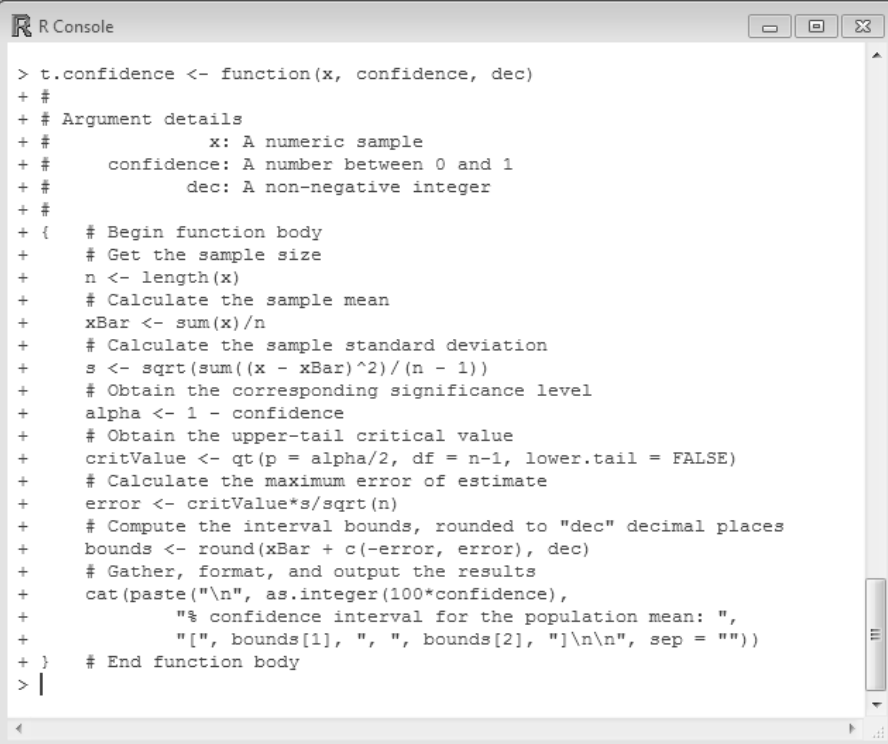
This informs us that the object `t.confidence` has been created and stored in the workspace.

```

t.confidence <- function(x, confidence, dec)
#
# Argument details
#
#   x: A numeric sample
#   confidence: A number between 0 and 1
#   dec: A non-negative integer
#
{
  # Begin function body
  # Get the sample size
  n <- length(x)
  # Calculate the sample mean
  xBar <- sum(x)/n
  # Calculate the sample standard deviation
  s <- sqrt(sum((x - xBar)^2)/(n - 1))
  # Obtain the corresponding significance level
  alpha <- 1 - confidence
  # Obtain the upper-tail critical value
  critValue <- qt(p = alpha/2, df = n-1, lower.tail = FALSE)
  # Calculate the maximum error of estimate
  error <- critValue*s/sqrt(n)
  # Compute the interval bounds, rounded to "dec" decimal places
  bounds <- round(xBar + c(-error, error), dec)
  # Gather, format, and output the results
  cat(paste("\n", as.integer(100*confidence),
            "% confidence interval for the population mean: ",
            "[", bounds[1], ", ", bounds[2], "]\n\n", sep = ""))
}
# End function body
#

```

Figure 1: To run the code for the function `t.confidence`, highlight all of the code for the function and then select “Run line or selection,” or use the Ctrl+R shortcut.



```

R Console
> t.confidence <- function(x, confidence, dec)
+ #
+ # Argument details
+ #
+ #   x: A numeric sample
+ #   confidence: A number between 0 and 1
+ #   dec: A non-negative integer
+ #
+ {
+   # Begin function body
+   # Get the sample size
+   n <- length(x)
+   # Calculate the sample mean
+   xBar <- sum(x)/n
+   # Calculate the sample standard deviation
+   s <- sqrt(sum((x - xBar)^2)/(n - 1))
+   # Obtain the corresponding significance level
+   alpha <- 1 - confidence
+   # Obtain the upper-tail critical value
+   critValue <- qt(p = alpha/2, df = n-1, lower.tail = FALSE)
+   # Calculate the maximum error of estimate
+   error <- critValue*s/sqrt(n)
+   # Compute the interval bounds, rounded to "dec" decimal places
+   bounds <- round(xBar + c(-error, error), dec)
+   # Gather, format, and output the results
+   cat(paste("\n", as.integer(100*confidence),
+             "% confidence interval for the population mean: ",
+             "[", bounds[1], ", ", bounds[2], "]\n\n", sep = ""))
+ }
+ # End function body
+
>

```

Figure 2: The results of running the code for `t.confidence`. A successful run is indicated by no error messages and the appearance of the prompt “>” at the end of the run. Again, note the appearance of “+” before each line after the first, indicating the continuation of one very long “command.”

Now consider testing this function on the same sample that was used in Assignment 1; but, here, let's give the sample a different name. Run

```
> mySample <- c(74.3, 69.9, 65.7, 70.1, 56.8,
+              69.2, 73.2, 49.6, 73.4, 77.8)
```

and then

```
> t.confidence(x = mySample, confidence = 0.95, dec = 1)
```

```
95% confidence interval for the population mean: [61.8, 74.2]
```

It is important to remember that it does not matter what the sample is named in the workspace, as long as this sample is correctly passed into the function the computations are performed.

It is a simple matter (in R) to crank out a new random sample from any distribution (for example a normal distribution with  $\mu = 75$  and  $\sigma = 15$ ). For example, another trial run of `t.confidence` can be performed using

```
> # Create a new sample
> # First set the seed for the random sample generator
> # (can use any positive integer)
> set.seed(seed = 13)
> # Generate and output the sample
> (bigFish <- rnorm(n = 73, mean = 75, sd = 15))
[1] 83.31490 70.79592 101.62745 77.80980 92.13789 81.23289
[7] 93.44260 78.55020 69.51926 91.57716 58.59609 81.92806
[13] 54.58523 47.15959 68.40217 72.09080 95.94647 76.50995
[19] 73.28342 85.53338 78.93814 102.54245 80.36104 59.31885
[25] 84.30276 77.24032 53.11025 44.59434 59.14563 64.07784
[31] 74.87684 87.71696 69.24763 67.10233 70.90161 65.91388
[37] 70.00699 71.37694 62.05837 62.29544 76.50511 98.85050
[43] 83.49742 99.21719 67.97025 64.10848 59.64991 45.93277
[49] 79.15721 96.12530 79.09694 86.33288 69.76472 66.80714
[55] 78.51543 70.53258 62.39286 87.39766 97.25537 85.49513
[61] 56.07639 79.47408 72.78289 61.66616 90.19599 61.19212
[67] 66.39158 92.25548 92.15737 71.40836 58.69797 74.07830
[73] 67.24954
> # Now use this sample in t.confidence
> t.confidence(x = bigFish, confidence = 0.99, dec = 2)
```

```
99% confidence interval for the population mean: [70.49, 79.02]
```

This gives us a *very basic* introduction to writing our own functions. As the semester develops we will look at ways in which we can add to the complexity (and robustness) of functions we create, and we will take a deeper look at built-in functions and how we can use them effectively.

Note that, for this course, when we want to name data structures such as `bigFish` we will use what is referred to as the *lowerCamelCase* naming-convention.

## A Brief Background Information Refresher

The code for the function `t.confidence` can be used as a (very preliminary) template for defining functions in R, and this template can be used for other basic functions for confidence intervals.

In typical elementary statistics courses the previously demonstrated  $t$ -interval and three other types of confidence intervals for single-population parameter estimates are encountered. In the following  $\alpha$ , is used to denote *significance level* and  $100(1 - \alpha)\%$  is used to denote the corresponding *confidence level*.

**$z$ -interval for  $\mu$ :** Let  $X$  be a random variable with mean  $\mu$  and standard deviation  $\sigma$ . Suppose the population standard deviation,  $\sigma$ , is known, and let  $\bar{x}$  be the sample mean of a random sample of size  $n$  for  $X$ . Then if  $X$  is normally distributed, or if  $n \geq 30$ , a  $100(1 - \alpha)\%$  confidence interval for  $\mu$  is given by

$$\bar{x} - z_{\alpha/2} \sigma / \sqrt{n} \leq \mu \leq \bar{x} + z_{\alpha/2} \sigma / \sqrt{n},$$

where the quantile  $z_{\alpha/2}$  satisfies  $P(z > z_{\alpha/2}) = \alpha/2$  using the standard normal distribution (commonly called the  $z$ -distribution).

**$z$ -interval for  $p$ :** Let  $p$  represent the true proportion of a population that satisfies a certain characteristic. For a random sample of size  $n$ , let  $x$  represent the number of individuals in the sample that have the characteristic in question (Note that in this case the underlying random variable  $X$  is binomially distributed with parameters  $p$  and  $n$ ), and denote the sample proportion by  $\hat{p} = x/n$ . If both  $n\hat{p} \geq 5$  and  $n(1 - \hat{p}) \geq 5$ , then a  $100(1 - \alpha)\%$  confidence interval for  $p$  is given by

$$\hat{p} - z_{\alpha/2} \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}} \leq p \leq \hat{p} + z_{\alpha/2} \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}},$$

where the quantile  $z_{\alpha/2}$  satisfies  $P(z > z_{\alpha/2}) = \alpha/2$  using the standard normal distribution.

**$\chi^2$ -interval for  $\sigma^2$ :** Let  $X$  be a random variable with variance  $\sigma^2$ . Suppose  $X$  is normally distributed, and let  $s^2$  be the sample variance of a random sample of size  $n$ . Then, a  $100(1 - \alpha)\%$  confidence interval for  $\sigma^2$  is given by

$$(n - 1) s^2 / \chi_{\alpha/2}^2 \leq \sigma^2 \leq (n - 1) s^2 / \chi_{1-\alpha/2}^2,$$

where the quantile  $\chi_{\alpha/2}^2$  satisfies  $P(\chi^2 > \chi_{\alpha/2}^2) = \alpha/2$  and  $\chi_{1-\alpha/2}^2$  satisfies  $P(\chi^2 > \chi_{1-\alpha/2}^2) = 1 - \alpha/2$  using the  $\chi^2$ -distribution (pronounced Chi-squared) with  $n - 1$  degrees of freedom.

Of interest at this point is how to calculate *critical values* (technical term is *quantile*) such as  $z_{\alpha/2}$ ,  $t_{\alpha/2}$ ,  $\chi_{1-\alpha/2}^2$ , and so on.

Note that some authors use open intervals to define confidence intervals. In these notes we will use closed intervals.

## Finding Critical Values for Confidence Intervals

Refer to your favorite elementary statistics text for further details on this matter. Run the following lines of code – you should get the output lines shown.

```
> qt(p = 0.05, df = 7, lower.tail = TRUE)
[1] -1.894579
> pt(q = -1.894579, df = 7, lower.tail = TRUE)
[1] 0.04999997
> qt(p = 0.05, df = 7, lower.tail = FALSE)
[1] 1.894579
> pt(q = 1.894579, df = 7, lower.tail = FALSE)
[1] 0.04999997
```

Now look up the documentation page for the  $t$ -distribution and find out more about what the functions `qt` and `pt` produce. Compare the results obtained above with values obtained from a  $t$ -distribution table (in the back of your favorite elementary statistics text).

Explore the use of analogous functions for the standard normal (Gaussian) and  $\chi^2$ -distributions. Perform trial runs for these functions, use  $\alpha = 0.05$ , and compare the results with values obtained from a standard normal and a  $\chi^2$ -distribution table.

## Taking Advantage of Built-in Functions

In Assignment 1 you should have looked for, and found functions that compute the mean, standard deviation, and variance of a sample. Now, instead of writing “from scratch” code to obtain the mean of a sample you can (and should) use, for example, the function call

```
> mean(bigFish)
[1] 74.75894
```

to calculate the mean of the sample `bigFish`. Similarly, you can use built-in functions to calculate the standard deviation, and variance of a sample. So, in the code for the function `t.confidence` you could very easily have used

```
xBar <- mean(x)
```

in place of

```
xBar <- sum(x)/n
```

and

```
s <- sd(x)
```

in place of

```
s <- sqrt(sum((x - xBar)^2)/(n - 1))
```

See the script file for the “tightened” code for `t.confidence`.

As we move through this course you will start taking more and more advantage of built-in functions in preparing your own code.

## Exploratory Exercises

In Assignment 1 you performed a key-word search for the word “operator.” Browse the documentation pages for *logical* and *relational* operators. Try using relational operators on some very simple relational statement examples (*simple statements*), then experiment with using logical operators to combine two or more simple relational statements (*compound statements*). For example,

```
> # Assign names to the numbers 5 and 7
> a <- 5; b <- 7
> # Test if a is larger than 5
> a > 5
[1] FALSE
> # Test if a is less than b
> a < b
[1] TRUE
```

```
> # Does 6 lie between a and b?
> (a < 6) & (6 < b)
[1] TRUE
```

The semi-colon “;” in the first line of code above, `a <- 5; b <- 7`, informs R to run “`a <- 5`”, and then run “`b <- 7`”. You may also choose to go further. For example,

```
> # Set the random number generator seed
> set.seed(seed = 5)
> # Get a small random sample of size 7, with replacement,
> # from the sequence 37, 38, ..., 93 and display it
> (smallSample <- sample(x = 37:93, size = 7, replace = TRUE))
[1] 48 76 89 53 42 76 67
> #
> # Test if any numbers in smallSample are between 45 and 53
> (45 < smallSample) & (smallSample < 53)
[1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE
> #
> # List those numbers in smallSample that lie between 45 and 53
> smallSample[(smallSample > 45) & (smallSample < 53)]
[1] 48
```

Keep these (relational and logical) operators in the back of your mind. We will soon start using R’s capabilities in the area of logical manipulations. Also useful in performing searches within data objects are the `which`, `all`, and `any` functions, and the `%in%` operator.

One useful operator, the *colon operator* “:,” and two useful functions, `set.seed` and `sample`, appear in the above code. The colon operator generates increasing (or decreasing) sequences of numbers, the difference between consecutive numbers being 1. Try running

```
1:10; 10:1; 1.25:5.5
```

The `sample` function generates a (psuedo-) random sample from a given sample space (passed into the function through the argument `x`), try running each of the following lines code several times.

```
sample(x = c("Yes", "No"), size = 10, replace = TRUE)
sample(x = 1:10, size = 10, replace = FALSE)
```

You will notice that it is very unlikely for two runs produce the same output. The `set.seed` function can be used to, in a sense, “fix” the random number generator that the `sample` function uses to generate a sample, thus “fixing” the outcome. Try running each of the following lines code several times.

```
set.seed(seed = 2); sample(x = c("Yes", "No"), size = 10, replace = TRUE)
set.seed(seed = 1005); sample(x = 1:10, size = 10, replace = FALSE)
```

Any positive integer can be used for the `seed`.

We will find that there are a variety of other ways in which R can be used to simulate “real” data. This capability proves very useful when data are needed (quickly) to test code that is being prepared for a statistical study.



## To be Submitted

You are to prepare two *well-documented*, and *clearly organized* functions for this assignment and then use your functions. Remember, you can compare your functions' output against hand- or TI-calculator computed intervals.

Call your script file, for example, “Hay-Jahans\_Assign2.R.” Once again, submit your assignment by email as an email attachment.

1. Prepare a function that will compute a  $100(1 - \alpha)\%$  confidence interval for a population proportion  $p$ . Hint: See p. 6 of these notes and then insert the *relevant computational instructions* in

```
p.confidence <- function(x, n, confidence, dec)
{   # Begin function body
:
...relevant instructions...
:
}   # End function body
```

and use the template illustrated on page 3.

2. Suppose an interval estimate of the true proportion of UAS students who have heard about R is desired. A random sample of 103 students reveals that 13 have heard about R. Provide code to use *your* `p.confidence` function to compute the 90% confidence interval for the true proportion of UAS students who have heard about R. Interpret the output.
3. Prepare a function that will compute a  $100(1 - \alpha)\%$  confidence interval for a population variance  $\sigma^2$ . Hint: Once again, see p. 6 of these notes and then insert the *relevant computational instructions* in

```
var.confidence <- function(x, confidence, dec)
{   # Begin function body
:
...relevant instructions...
:
}   # End function body
```

4. To obtain an estimate of the true variance of the weights (in lbs<sup>2</sup>.) of pink salmon that arrived at the Peterson Creek Salt-Chuck (near Juneau) in the 2012 season, UAS marine biology students used a random sample of 36 pink salmon obtained in that season. The data obtained are:

5.3, 4.0, 4.9, 6.4, 2.0, 4.8, 3.8, 1.7, 6.7, 9.6, 5.2, 9.4, 3.8, 2.4, 3.9, 4.8, 5.0, 3.8, 7.3, 4.5,  
1.8, 3.3, 4.0, 4.5, 4.8, 5.9, 7.3, 3.6, 3.4, 4.4, 6.2, 4.5, 5.3, 1.4, 5.7, 2.3

Provide code to use *your* `var.confidence` function to compute a 99% confidence interval for the true variance of the weights of pink salmon that arrived at the Peterson Creek Salt-Chuck in the 2012 season. Interpret the output.