HITACHI
Inspire the Next

R on Pentaho Data Integration (PDI)

HITACHI
Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes

Contents

Overview 1
Before You Begin 1
Prerequisites 1
Integrating R and PDI 2
What is R? 2
What is RStudio? 2
Step by Step Installation 3
Verifying Your Installation 4
Getting Data Out 5
Returning a `data.frame` (Default) 5
Returning Data as Text 6
Getting Data In 6
Why R? 10
Why Not Use R By Itself? 10
Troubleshooting, Tips, and Tricks 11
Use RStudio 11
Testing Return Values 11
Problems with Output Strings 12
Testing Input Data 12
Related Information 13
Finalization Checklist 13

This page intentionally left blank.

Overview

This document covers some best practices on integrating R with Pentaho Data Integration (PDI). In it, you will learn how to install and use R with PDI and why you would want to use this setup.

Our intended audience includes data analysts, data scientists, and PDI users who need to use the variety of statistical and machine learning tools available in the R environment.

The intention of this document is to speak about topics generally; however, these are the specific versions covered here:

Software	Version(s)
Pentaho	6.x, 7.x, 8.0

The Components Reference in Pentaho Documentation has a complete list of supported software and hardware.

Before You Begin

Before beginning, use the following information to prepare for the procedures described in the main section of the document.

Prerequisites

This document assumes that you have knowledge about Pentaho and programming, and that you have already installed Pentaho.

Use Case: Interactive Statistical Analysis of a Sampled Data Set

"As a PDI user, I want to sample a subset of a larger data set being prepared in PDI to take advantage of R's machine learning libraries."

*R contains many libraries that support machine learning. Unfortunately, as many authors have pointed out, R does not have support for doing data interactive analysis at big data scale. Many users of PDI are used to dealing with data at scale, but may need to interface with another team who are expert in statistical analysis and machine learning. In this case, the R plugin can be used to sample the data to an R data frame for initial export. At this point, various machine learning packages in R can be used. Some of these include PARTY, for recursively **PART**itioning tree based classification models of various types, and CARET (**C**lassification **A**nd **Re**gression **T**raining), which allows for model tuning.*

Once the most promising models have been selected, they can be further tested against the larger by re-sampling the data in with new values for the random seed.

Integrating R and PDI

R is a GNU project available at no cost under the GNU General Public License. It is both a language and environment that is used for statistical and graphical work, and it can run on many platforms.

You can find details on these topics in the following sections:

- What is R?
- What is RStudio?
- Step by Step Installation
- Verifying Your Installation
- Getting Data Out
- Getting Data In

What is R?

You can use R to handle, arrange, and analyze data, and you can significantly extend R's capabilities using packages. The R distribution includes several packages such as graphics, statistics, utilities, and you can also create your own packages or use others' offerings from around the web.

The interface for R, alone, is text-only, so you may choose to use a GUI like RStudio for using R.

What is RStudio?

RStudio is an integrated development environment (IDE), a convenient GUI for editing and executing the R code. RStudio includes capabilities for debugging and plotting, as well as other features that will allow you to develop your own packages. In addition, RStudio provides better integration of R's useful help system, built-in support for editing and testing R notebooks and scripts. You also have access to an integrated R console, so you gain these features without losing any of the functionality that you had when running R directly.

We recommend that you test everything in RStudio before passing it to PDI, to make sure it is all working as you expect.

RStudio can run by itself from the desktop, or can be run in a web browser (RStudio Server version). Since RStudio depends on R, you first will need to install R. We provide more detailed instructions in the Step by Step Installation.

Step by Step Installation

Here is a walkthrough of the steps needed to integrate R and PDI.

1. Verify that the R Script Executor plugin is installed. It should have been installed as part of Pentaho Enterprise Edition.
   a. Check for Spoon's **R Script Executor** in the **design tab** under **Statistics**, or
   b. Search the **design tab** for **R Script Executor**.
2. Install R for your platform, if you have not already installed it. Choose the correct version and platform for your operating system (Windows, Mac, or Linux) and Java installation. (At a command prompt, run `java -version` to find out if your installation is 64-bit. If it does not say, it is 32-bit.)
3. Once R is installed, install RStudio.
4. On Windows, add the location of your R installation's bin platform directory to your path. (Not the bin directory, but the one underneath that has `rcmd.exe` and `r.dll`.) It will typically be `c:\Program Files\R\R-<version>\bin\x64` for 64-bit Windows or `c:\Program Files\R\R-<version>\bin\i386` for 32-bit Windows. Note also that on 64-bit Windows, the platform directory you choose should also match the platform of your JVM.

We recommend adding the location of the RStudio bin directory to the path as well, so that you can type `rstudio` at the command line. You can skip this if you do not use the command prompt often.

5. Run R, then from inside the R shell, run the following command:

```
install.packages('rJava')
```

6. Set two additional system environment variables:
   a. Set `R_HOME` to the root of your R installation, for example: `c:\Program Files\R\R-<version>` on Windows.
   b. Set an environment variable `R_LIBS_USER` to the directory where your user libraries are installed (it will be a directory with a folder called `rJava` in it). You can get this directory in R with this command:

```
sys.getenv("R_LIBS_USER")
```

7. The installation may now be complete, but the wiki recommends this additional step:
   a. Stop Spoon, if it is running.
   b. Find `%R_LIBS_USER%/rJava/jri/x64/jri.dll` (or `i386`, depending on your Java version).
   c. If on Windows, copy `jri.dll` to `[Pentaho directory]/client-tools/data-integration/libswt/win64`.
   d. If on Ubuntu/Linux, copy `jri.dll` to `[Pentaho directory]/client-tools/data-integration/libswt/linux/[x86_64]/`.

Verifying Your Installation

To verify your installation:

1. Open a new transformation in PDI.
2. Drag an **R Script Executor** step onto the canvas.
3. Double-click the step and select the middle tab, **R Script**. You will see some comments at the top of the window:

```
# The main output is expected to be a data frame, unless "Output
# from script is text" is checked. So, to output a data frame the
# last statement in the script should be the name of the frame.
# In the case that the output is text (as would be seen on the
# R console), the last statement should be a "print" statement in
# order to print the object required.
```

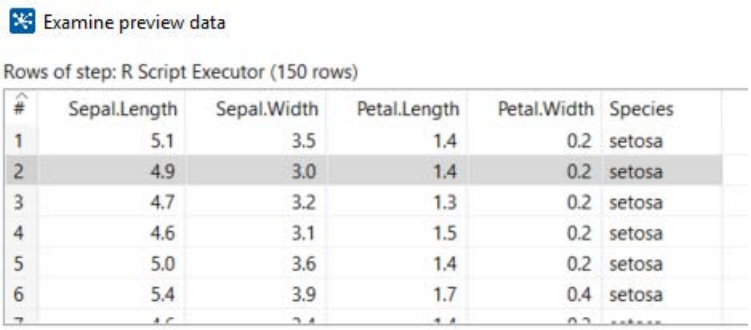
   You will often want to return a DataFrame instead of printing text. Try this by returning one of R's built-in datasets. Beneath the comment above, enter this code:

```
library(datasets)
iris
```

4. The first line lets you reference any of the built-in datasets by name, which you do in the next line, returning one of the datasets from your script. (Among data scientists, Fisher's Iris Flower Data Set is a sort of "Hello world" of datasets).

Note that you do not need to write `return (iris)` or the like as you would in some languages. R, like Python and Ruby, follows the convention that by default, the return value is the value of the last expression evaluated. In this case, that is the dataset `iris`.

5. Once you have entered this code in the **R Script** tab, click the **Test Script** button on this tab.
6. Click **Yes** if you're sure you want to run the step and you should see data like this:



#	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Figure 1: Preview Data

Getting Data Out

Now that you have verified your R configuration, set it up so that you can use the data in subsequent steps.

Returning a data.frame (Default)

Expose the columns as output fields:

1. Click the **Output Fields** tab.
2. Click **Get Fields**.
3. You may see a warning about overwriting the current output values. Click **Yes** and you should see the following:

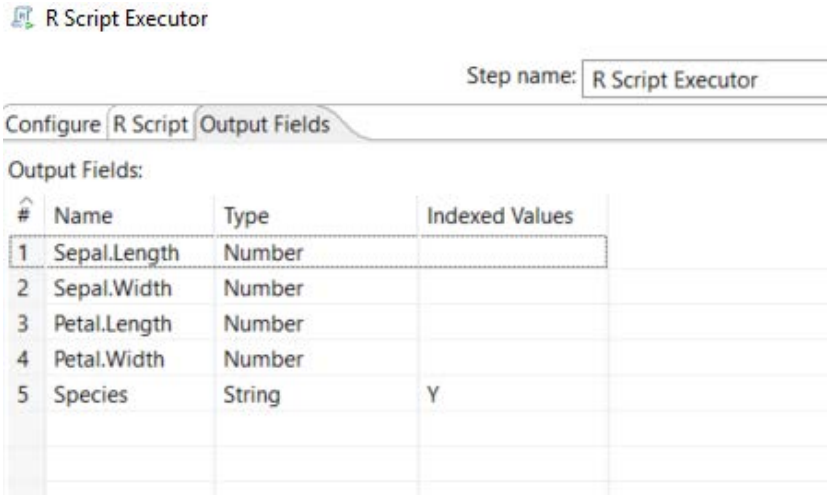


Figure 2: Output Fields

*As you can see, if you return a `data.frame` object from your script correctly, the **Output Fields** tab will expose each column of the frame as a field, and the rows will be available to subsequent steps.*

Returning Data as Text

Returning a `data.frame` object is the most common use case, and as you saw in the previous section, each of the columns of the `data.frame` can then be set to other steps as a field. However, another option for returning data from an R script is to return the data as text.

1. Go to the **Configure** tab in the step.
2. Select the **Output from Script is Text** option:

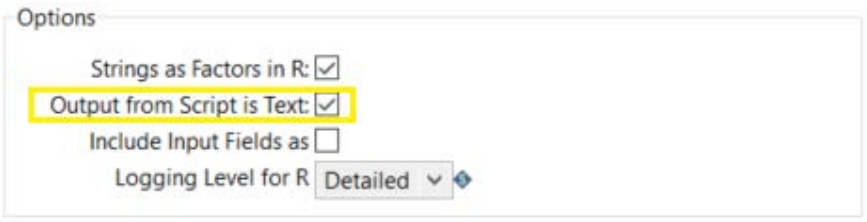


Figure 3: Output From Script is Text

The idea is to send the text you want to the console using one of the following R functions:

- `print` (See Troubleshooting for a caution on using `print`.)
- writeLines
- cat

One undocumented feature of the plugin is that multiple lines sent to the console will be concatenated into a single `R-Console` output variable. This means that, with a little extra work, you can return multiple text strings and then split them using a Split Fields step.

The following function will write a string followed by a semicolon, which can then be used as a delimiter in a Split Fields step.

```
writeField <- function(x) {
  cat(paste(x, ";"));
}
```

Getting Data In

You have seen how to get data out of the **R Script Executor** step. Let's look at the case where you have some existing data, and you want to submit it to the R Script Executor step as input.

You can use the tools you are familiar with in PDI to load and filter the data, and then rely on R to do a quick statistical analysis of the data.

For example, you want to examine your sample sales data and get quantile data for price, quantity ordered, total sales, and so forth. You can take advantage of your knowledge of PDI steps to do some of the data loading and filtering, as well as the output. You can use R for what it is best at – doing a statistical analysis of a data set.

Figure 4 shows how your transformation might look:

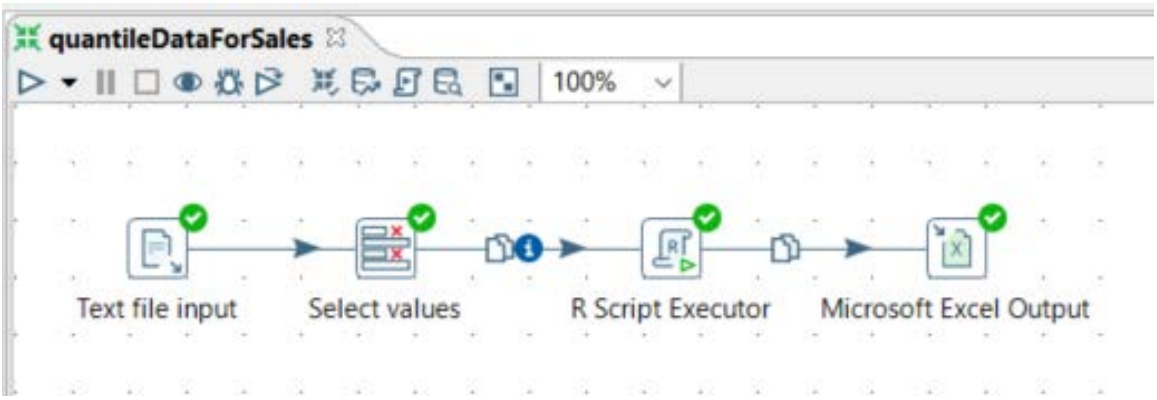


Figure 4: Sample Transformation

1. Use a **Text file input** step to load the `sales_data.csv` sample that ships with PDI at the following locations under the PDI installed directory:
2. Change the delimiter to a comma `,`
3. Do a **Get Fields**, and you will be set up to begin your transformation.
4. In the second step of the transformation, **Select values**, filter out the non-numeric fields and other fields you are not interested in:

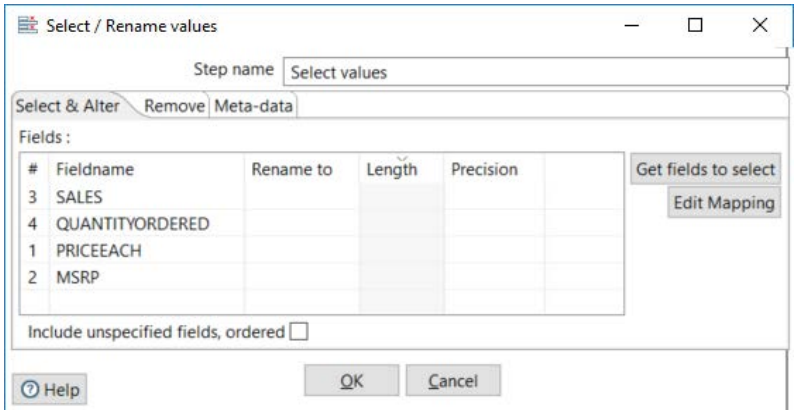


Figure 5: Select & Alter

You could do this selection in R as well, but doing it in PDI makes it clear at a glance what you are doing, which leaves your R code uncluttered by indices into columns that otherwise might be difficult to understand.

5. To input these four fields into R, configure your R step as follows:
   a. Set **Number of Rows to Process** to **All**, because you want to do a statistical analysis across all the rows of data, and the sample size of the dataset is not too big. (For

datasets too large to load into a single `data.frame`, you could use the **Reservoir Sampling** feature to select a random sample of data.)

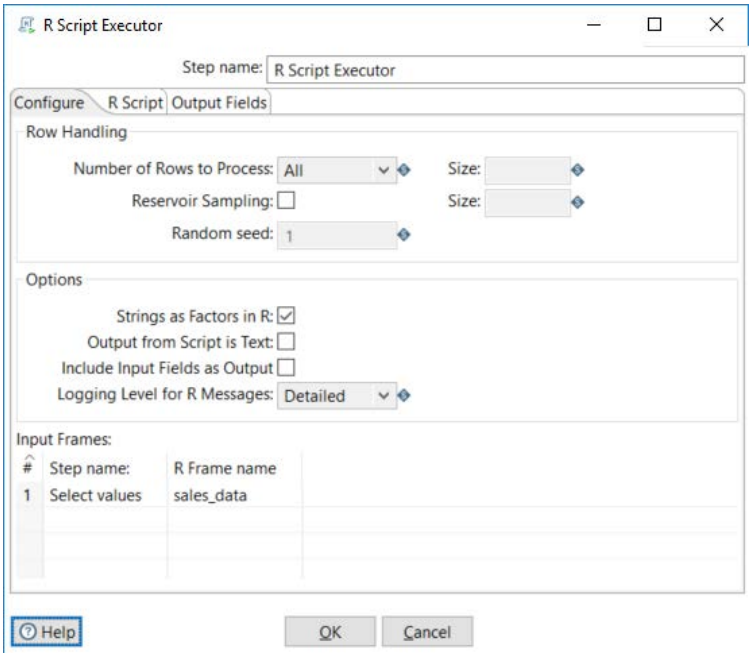


Figure 6: Configure Tab

b. Next, set a name for the `data.frame` object in R. You want the rows from the linked **Select Values** step; name those rows `sales_data` as a `data.frame` variable in R.
c. Then, on the **R Script** tab, enter these lines of code:

```
quants = as.data.frame(apply(sales_data, 2, quantile))
cbind(QUANTILE = row.names(quants), quants)
```

In the first line, use the `apply` function to run the `quantile` function against the numerical columns of the `sales_data` `data.frame`. The 2 tells the `apply` method to apply it to each of the rows of the `data.frame`. You cast the result to a `data.frame`, since it would otherwise be returned as a matrix.

In the second line, use the `row.names` function to prepend a column named `QUANTILE` to the `data.frame`, which is returned from the script as the result of `cbind`. This allows you to see the `QUANTILE` each row belongs to when the rows are set to later steps.

d. With those lines of code in place, click the **Output Fields** tab.

e. Click the **Get Fields** button and you should see the following fields:

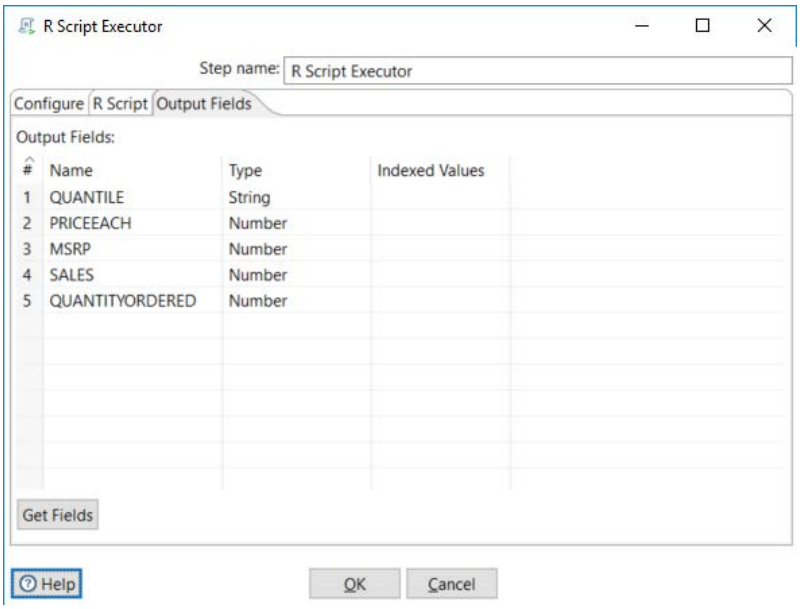


Figure 7: Get Fields

f. Use a **Microsoft Excel Output** step to save the fields to an Excel spreadsheet, by clicking **Get Fields** to get the values from the R step, and entering a filename.

g. You should get these results:

Table 1: Results

QUANTILE	PRICEEACH	MSRP	SALES	QUANTITYORDERED
0%	26.88	33.00	482.13	6.00
25%	68.86	68.00	2,203.43	27.00
50%	95.70	99.00	3,184.80	35.00
75%	100.00	124.00	4,508.00	43.00
100%	100.00	214.00	14,082.80	97.00

Why R?

We dove into the how of integrating R with PDI without tackling the related questions:

- Why should you think about integrating R and PDI?
- What problems does it solve that you won't be able to solve using PDI alone?

If you have a well-defined data warehouse architecture and you are using PDI to organize and prepare existing data for loading into your data warehouse, it may be that you do not need R at all. What is often true when evaluating business software is true here as well: If you don't know why you need it, maybe you don't.

However, there are some cases where R might be a good fit:

- You want to take the data you have wrangled using PDI and run some kind of ad hoc statistical analysis on it. This will often be the case if you have data scientists or statisticians on staff. For many such users, R is likely to be their favorite go-to tool.
- You want to take advantage of any of the over 11,000 software packages that work in the R environment. For example, perhaps you would like to output histograms or other plots inline in your PDI jobs, rather than using Pentaho Reporting tools at a later step in the process. Or, perhaps you want to take advantage of R's many tools for data partitioning and machine learning.

Why Not Use R By Itself?

Taking a slightly different approach to the question, if R has all this statistical goodness baked right in, why not just use R and not bother integrating it with PDI? There are three good reasons, relating to PDI's extensibility, simplicity, and scalability.

First, PDI's plugin system offers **extensibility using Java**, the most widely used general purpose business programming language. In contrast, though R has a great community and many fine packages, even R's biggest fans do not argue that R is easy to use, and no one claims R is a general purpose programming language.

R is a specialized, expert language. Its syntax and idioms are not easily accessible, even to developers trained in other languages. In contrast, PDI is not a language at all, but a **simple graphical tool**. It is easy for business analysts who are not developers to learn the basics of PDI in a few days and begin to do productive work. For most beginners, it is easier to read a PDI transformation and interpret what it does than it would be to read and interpret an R script.

Finally, **R is not designed for large datasets**, except through tricks like connecting to Spark. PDI also supports using Spark through its Adaptive Execution Layer (AEL), but the asynchronous, row-by-row processing model of PDI means it scales much better for medium to large datasets even without using Spark.

Troubleshooting, Tips, and Tricks

If you encounter trouble running your R scripts in the PDI plugin, the first thing to do is verify that R and PDI can communicate correctly as described in the Verifying Your Installation section.

Assuming your installation is correct, the next most common cause of problems using the R plugin is simple script errors.

Use RStudio

We recommend always having RStudio open when you are working in PDI. This will allow you to debug any script errors that might otherwise be very difficult to troubleshoot in PDI. You can load or paste your script into an R script window, and click Source to run the script, or paste a smaller number of lines directly into the console.

Testing Return Values

One of the more common errors is simply getting the script return value wrong. If you are returning a `data.frame`, for example, it is possible you have returned an expression that does not evaluate to a `data.frame` as you would expect.

This mistake is less common in R than when using the Python Pandas plugin, since the return values from querying an R `data.frame` for a specific column or row are more likely also to be a `data.frame`, whereas in Pandas it is possible to return a `Series` object.

However, if you are expecting a `data.frame` as a return value, you can easily test its value by running the script in RStudio. Taking a known good case, the iris dataset from our previous example, you can verify that it will work.

```
library(datasets)

iris
```

There are at least two options for verifying that you have indeed loaded a `data.frame`. While you are still in the console:

```
# What are we dealing with?

class(iris)

[1] "data.frame"
```

Or:

```
# Alternatively:

is.data.frame(iris)

[1] TRUE
```

Problems with Output Strings

You may have noticed the default comment that appears when you open the R Script tag of the R Script Executor plugin. It reads, in part, `In the case that the output is text (as would be seen on the R console), the last statement should be a "print" statement in order to print the object required`. However, you may also notice that if you do use `print`, you will see this in R style, with a `[1]` prepended to the string as it appears in the console:

```
> print("Hello")
[1] "Hello"
```

To avoid this, you can use `writeLines`, which will also append a new line for you, or `cat`, which will not.

Testing Input Data

It can be more challenging to verify input data than output data, since with input data, you may need to set up test versions of one or more `data.frames` to debug your whole script. There are a few approaches you might use here.

- You might temporarily divert the rows coming into the R plugin to a **CSV Output** step, and use R's `read.csv` or `read.table` to load the data back into R.
- You might try a sort of unit testing approach by testing various parts of your script individually.
- Particularly if the input `data.frames` are small and simple (at least as far as what R is doing with them), you could set up correctly named variables at the top of your script and run a test that way.

Related Information

Here are some links to information that you may find helpful while using this best practices document:

- Iris Flower Data Set
- Pentaho
  - Components Reference
  - Downloads
  - R Script Executor
  - R Project for Statistical Computing
- RStudio
  - RStudio Products
  - RStudio Online Learning Page
- What are the Best Machine Learning Packages in R? (Blog)

Once you have learned the basics. R Studio has an excellent integrated help system. In the console, you can learn more about any object or function by typing ? plus the object name. For example, try the following query:

```
?data.frame
?colnames
?install.packages.
```

Note too that tab completion is available for partial names.

Finalization Checklist

This checklist is designed to be added to any implemented project that uses this collection of best practices, to verify that all items have been considered and reviews have been performed.

Name of the Project:_____

Date of the Review:_____

Name of the Reviewer:_____

Item	Response	Comments
Did you install R?	YES_____ NO_____	
Did you add the location of your R installation's bin platform directory to your path?	YES_____ NO_____	
Did you set your `R_HOME` and `R_LIBS_USER` system environment variables?	YES_____ NO_____	
Did you verify your installation?	YES_____ NO_____	