MicroStrategy | 10

R Integration Pack User Guide

Version: 2.2

2.2, August 2016

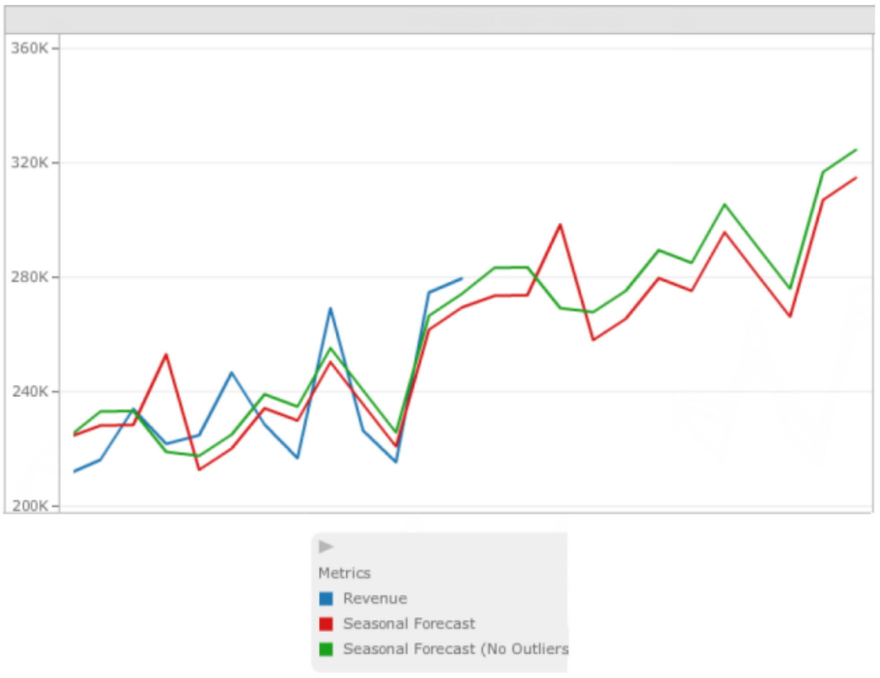
Contents

1

Overview of the R Integration Pack

The MicroStrategy R Integration Pack lets you easily integrate statistical computing and graphics environments into MicroStrategy by deploying R analytics as standard MicroStrategy metrics that implement R scripts. R is an open-source, industry-leading language and environment for statistical computing and graphics. A MicroStrategy metric is a calculation that represents a business measure or key performance indicator, in this case, the R analysis. Metrics can be used directly on MicroStrategy reports, documents, and dashboards.

For example, you can project a seasonal forecast of revenue, as shown in the following dashboard. The R analytics have been deployed as the metrics Seasonal Forecast and Seasonal Forecast (No Outliers).



Because the R Integration Pack deploys R analytics as MicroStrategy metrics, deployment and management is easy. Additionally, the R Integration Pack includes a set of common R script functions, pre-compiled and ready to integrate into any MicroStrategy project.

Depending on your role, you perform one or more the following tasks to integrate or use your MicroStrategy software with R:

- If you are a system administrator, install the R Integration Pack for R developers and MicroStrategy users.

  For detailed information on preparing your environments, and steps to install the R Integration Pack, see *Chapter 2, Installing and Configuring the R Integration Pack*.

- If you are a MicroStrategy analyst, use R analytics to analyze data. With the R Integration Pack installed and R analytics integrated with MicroStrategy, you can begin to analyze the R analytic statistical information in MicroStrategy by creating a metric. For detailed steps to create a metric that uses your R analytics, see *Chapter 3, Performing statistical analysis*.

- If you are an R developer, install the MicroStrategyR package, and develop your R analytics. An R analytic is a statistical analysis function that you write in R. To allow users to perform statistical analysis with an R analytic, you must implement the analytic as an R script, and integrate it with MicroStrategy using the deployR utility.

  For detailed steps and best practices to create R analytics for use in MicroStrategy, see *Chapter 4, Developing R Analytics for MicroStrategy*.

- The MicroStrategy R Integration Pack returns error and warning messages to help troubleshoot potential issues. Once you receive an error message you can search for it within *Chapter 5, Troubleshooting*, which provides a potential cause and resolution to the problem.

2

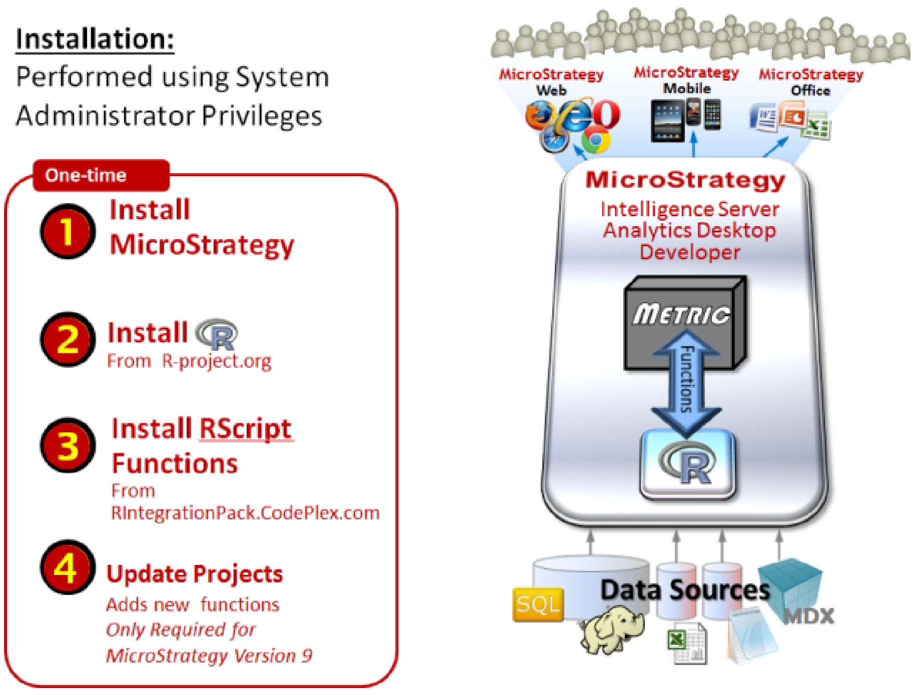
Installing and Configuring the R Integration Pack

To integrate R analytics with MicroStrategy offerings such as MicroStrategy Analytics Enterprise and MicroStrategy Desktop, you must install and configure the necessary R Integration Pack software.

The R Integration Pack consists of the following components:

- **RScript Functions**: A set of functions that are added to a MicroStrategy project to allow MicroStrategy metrics use your R analytics.
- **MicroStrategyR Package for R**: A package that contains an R-based utility called deployR, which prepares your R analytics for use in MicroStrategy metrics. The MicroStrategyR Package is only required on systems that are being used to develop your R analytics. For detailed steps to install the MicroStrategyR Package, see *Chapter 4, Developing R Analytics for MicroStrategy*.

The following figure provides a high-level summary of the tasks you perform to install the R Integration Pack.



Perform the following steps to install and configure the R Integration Pack environment:

1. *Installing MicroStrategy products, page 7*
2. *Installing R, page 9*
3. *Installing the RScript functions, page 10*
4. *Upgrading MicroStrategy Analytics Enterprise projects, page 13*

Installing MicroStrategy products

Before installing the R Integration Pack, you must install and configure the supported MicroStrategy products. You can integrate R analytics with the following MicroStrategy products:

- **MicroStrategy Analytics Enterprise**: MicroStrategy version 9.2.1 or later must be installed. If installing with MicroStrategy 9.x, MicroStrategy Architect is necessary for the one-time addition of the common R script functions.

  Additional requirements include:

  - To support deploying a new R analytic as a derived metric, you must have MicroStrategy OLAP Services to create derived metrics.

    A derived metric is based on the existing metrics in the report, document, or dashboard; therefore, the derived metric can only be used in that report, document, or dashboard.

  - To support deploying a new R analytic as a derived metric in a dashboard, MicroStrategy version 9.3.1 or later must be installed.

  - To create metrics using the R script functions, you must have the MicroStrategy Developer or MicroStrategy Web Designer privilege. Any user can view and analyze the results of metrics that employ the R script functions.

    As of MicroStrategy version 9.4.1, the name of MicroStrategy Desktop has been changed to MicroStrategy Developer.

- **MicroStrategy Desktop**: MicroStrategy Desktop version 9.4.1.3 or later must be installed. There are no MicroStrategy privileges required to create, view, or analyze metrics that employ R script functions.

- When migrating your MicroStrategy 9.x environment to a new major version (excludes hotfix and service packs), you must first uninstall the R Integration Pack. After the new MicroStrategy version is installed, re-install the R Integration Pack as described in *Installing the RScript functions, page 10*.

Creating centralized repositories for R files

To perform and display the statistical analysis you create using R, MicroStrategy products require access to the R script (`.R`) files, as well as any images made in R that are included in a MicroStrategy dashboard or document.

For the single-user MicroStrategy Desktop product, these files can be stored on the same machine as MicroStrategy Desktop.

For MicroStrategy Analytics Enterprise, creating centralized repositories for these files provides a single location that can be used across the enterprise. To create these repositories, see:

- *Creating a centralized repository for R script files , page 8*
- *Creating a repository for R images, page 8*

Creating a centralized repository for R script files

When performing analysis of your R statistics from MicroStrategy, a MicroStrategy metric must have access to the R script (`.R`) file. These R script files can be stored in the default R scripts folder that is created when installing the R Integration Pack. This provides a simple centralized repository for your R script files and allows for a quick configuration.

To provide a single network location where all R script files can be stored and accessed, MicroStrategy recommends that you save the R script files on a shared drive that all MicroStrategy products, including Intelligence Server and MicroStrategy Developer, can use and access. The shared drive can then be made available as a local path defined in the `_WorkingDir` parameter or the directory defined in the registry key `RScriptsFolder`.

Creating a repository for R images

To use images made in R in a MicroStrategy dashboard or document, you must save the images in a location accessible by the MicroStrategy platform.

MicroStrategy recommends that you save the image file on a web server that all your MicroStrategy products, including Intelligence Server, can use and access. When you add the image to a document or dashboard, use an HTTP reference to that accessible web server machine, such as `http://microstrategy/Test/myimage.jpg`.

If your system cannot support accessing a central web server using the HTTP syntax because of permissions or other access issues, you can save the image on a shared network drive or within the MicroStrategy product you are using. For information on storing images on shared network drives or within the MicroStrategy product, see the Document Creation Guide .

Installing R

Before installing the R Integration Pack, you must ensure that your environments have R and its dependencies installed. The following configurations are required when installing R for MicroStrategy environments:

- You must install R version 3.x or later. The proper version of R must be installed from http://CRAN.R-project.org:
    - Windows: When installing R, keep the default option to install both the 32-bit and 64-bit version of R.
    - Unix and Linux: The 64-bit version of R is required. You must also, ensure that `libR.so` is included in the R environment. You can define this support by running the following command for your R environment:

      ```
      ./configure --enable-R-shlib
      ```

      If an R 3.x installation is not available from http://CRAN.R-project.org for your version of Linux, third-party tools such as the Yellowdog Updated, Modified (Yum) and Extra Packages for Enterprise Linux (EPEL) can be used to install and update your R installation. Refer to your third-party R documentation available at http://www.r-project.org/ for information on installing and updating R.

- R and any dependencies must be installed on systems that will execute the R analytics deployed to MicroStrategy. These dependencies include any add-on packages that are required for a given R analytic. R and any dependencies must be installed on the following systems:
    - If you are using MicroStrategy Analytics Enterprise:
        - **MicroStrategy Intelligence Server host machine**, which supports R analytic execution for MicroStrategy Web, MicroStrategy Mobile, MicroStrategy Visual Insight, and MicroStrategy Office.

          Intelligence Server also supports R analytic execution with stand-alone metrics for MicroStrategy Developer clients that are connected in server (three-tier) mode.
        - **MicroStrategy Developer clients** that execute R analytics as derived metrics or in direct (two-tier) mode.

- If you are using MicroStrategy Desktop:
    - **MicroStrategy Desktop machine** that executes deployed R analytics. For MicroStrategy Desktop configurations, this is the only machine that must include the R dependencies.

Installing the RScript functions

The **R Script Functions** are a set of common functions and supporting files that are added to a MicroStrategy project, which allow MicroStrategy metrics to pass an R script and its associated inputs to R for execution. Metrics using these common R script functions also return the results from R for display and analysis in MicroStrategy reports, documents, and dashboards.

The RScript functions must be installed on the following systems:

- If you are using MicroStrategy Desktop, the machine that has MicroStrategy Desktop installed.
- If you are using MicroStrategy Analytics Enterprise:
    - The MicroStrategy Intelligence Server machine.
    - If your users need to execute R analytics as derived metrics in MicroStrategy Developer, or in direct (two-tier) mode, the MicroStrategy Developer machine.

The required installation files can be downloaded from the Downloads tab at http://RIntegrationPack.codeplex.com. Depending on your platform, use one of the following procedures to complete the installation:

- *Installing on Windows, page 10*
- *Installing on UNIX and Linux, page 11*

Installing on Windows

The steps below show you how to use the RScript Functions Installer to install the R script functions on a Windows environment for MicroStrategy Analytics Enterprise and MicroStrategy Desktop.

Prerequisite

- To successfully complete the installation, you must execute the installation using an account with administrative privileges or choose to execute the installation as an administrator.

To install the R script functions on Windows

1. Access the site http://RIntegrationPack.codeplex.com.
2. From the **Downloads** tab, locate and download the **RIntegrationPack.msi** file.

3 Double-click **RintegrationPack.msi** to execute the file and begin the installation process. The installation wizard is displayed.

4 Complete the steps provided in the installation wizard. By default, the installation location is:

- 32-bit Windows environments: `C:\Program Files\R Integration Pack`
- 64-bit Windows environments: `C:\Program Files (x86)\R Integration Pack`

The following files and folders are available after the installation:

- **ReadMe.txt**: Includes what's new information as well as important support and configuration information such as upgrade considerations.
- **Documentation**: The folder location of this documentation, in PDF format.
- **RScripts**: The default folder location for R scripts (`.R` files) deployed to MicroStrategy. The sample R script referenced later in this guide can be found in this folder.

5 When installation is complete, ensure that the R script functions are available for the environment as described below:

- **MicroStrategy Analytics Enterprise**:
  - For existing projects, the R script functions may need to be added to the projects' metadata or upgraded. For steps to add or upgrade R script functions for existing projects, see *Upgrading MicroStrategy Analytics Enterprise projects, page 13*.
  - For projects created after the R script functions are installed, the R script functions are automatically added to these newly created projects. The R script functions are added to the list of Data Mining Functions available when creating metrics, including the functions `RScript`, `RScriptU`, `RScriptAgg`, `RScriptAggU`, and `RScriptSimple`.
- **MicroStrategy Desktop**:
  - The R script functions are included with the standard installation of MicroStrategy Desktop. The functions `RScript`, `RScriptU`, `RScriptAgg`, `RScriptAggU`, and `RScriptSimple` are included in the list of Data Mining Functions, available when creating metrics.

Installing on UNIX and Linux

The steps below show you how to use the RScript Functions Installer to install the R script functions on a UNIX or Linux environment for MicroStrategy Analytics Enterprise.

To install the R script functions on UNIX or Linux

1 Access the site http://RIntegrationPack.codeplex.com.

2 From the **Downloads** tab, click one of the following options:

- **AIX Installer**: Downloads `RintegrationPack_AIX.tar.gz`, for installation on AIX.
- **Linux Installer**: Downloads `RintegrationPack_Linux.tar.gz`, for installation on Linux.
- **SunOS Installer**: Downloads `RintegrationPack_SunOS.tar.gz`, for installation on Solaris.

3 Extract the installation files, using the following command:

```
tar -zxvf RintegrationPack_OS.tar.gz
```

Where *`OS`* is the operating system that you are installing to.

4 Navigate to the new `RIntegrationPack_`*`OS`* folder, where *`OS`* is the operating system that you are installing to.

5 You must execute the RScript Functions Installer with the same user account that executed the MicroStrategy software installation for the system. This ensures that the R script function files are installed to the required MicroStrategy installation folder location.

You can complete the installation using an interface or through command line mode:

- To use the installation wizard interface, type and execute the following command:

  ```
  setup.sh
  ```

- To use the command line mode, type and execute the following command:

  ```
  setup.sh -console
  ```

6 Complete the steps provided in the installation wizard. By default, the installation location is `/var/opt/R_Integration_Pack`, or `$HOME/R_Integration_Pack` if you do not have write access to `/var/opt/R_Integration_Pack`. The following files and folders are available after completing the installation:

- **ReadMe.txt**: Includes what's new information as well as important support and configuration information such as upgrade considerations.
- **Documentation**: The folder location of this documentation, in PDF format.
- **RScripts**: The default folder location for R scripts (`.R` files) deployed to MicroStrategy. The sample R script referenced later can be found in this folder.
- **uninstall**: Includes the file `uninstall.sh`, which can be used to uninstall the R Integration Pack.

7 When the installation is complete, ensure that the R script functions are available for your environment as described below:

- **MicroStrategy Analytics Enterprise**:

- For existing projects, the R script functions may need to be added to the projects' metadata or upgraded. For steps to add or upgrade R script functions for existing projects, see *Upgrading MicroStrategy Analytics Enterprise projects, page 13*.
- For projects created after the R script functions are installed, the R script functions are automatically added to these newly created projects. The R script functions are added to the list of Data Mining Functions available when creating metrics, including the functions `RScript`, `RScriptU`, `RScriptAgg`, `RScriptAggU`, and `RScriptSimple`.

Upgrading MicroStrategy Analytics Enterprise projects

If you have MicroStrategy Analytics Enterprise projects that existed before the installation of the R script functions, you must upgrade the projects using the MicroStrategy Configuration Wizard to support the following scenarios:

- If this is the first installation of the R script functions with MicroStrategy 9.x, upgrading a project adds the R script functions to the project.
- If you upgraded a previous version of the R script functions, upgrading a project ensures that any changes to the R script functions since your previous installation are applied to the project. The `ReadMe.txt` file includes information on whether the R script functions were modified for the most recent release. If you are upgrading a version of the R script functions from more than one release prior, it is recommended to upgrade a project to ensure that any R script function changes are applied.

For MicroStrategy Desktop configurations, the R script functions are included with the standard installation of MicroStrategy Desktop.

The steps below show you how to add or upgrade the R script functions for existing MicroStrategy projects by upgrading the projects.

To add or upgrade R script functions for existing MicroStrategy Analytics Enterprise projects

1 If you are upgrading MicroStrategy projects on:

- Windows, then perform the following step:

  From the **Start** menu, point to **Programs**, then **MicroStrategy Tools**, and then choose **Configuration Wizard**. The Configuration Wizard opens.

- UNIX or Linux using the Configuration Wizard interface, then perform the following steps:

   a. From a UNIX or Linux console window, browse to `HOME_PATH`, where `HOME_PATH` is the directory that you specified as the home directory during your MicroStrategy installation.

   b. Browse to the folder `bin` and type `./mstrcfgwiz`, then press ENTER. The Configuration Wizard opens.

2. Select **Upgrade existing environment to MicroStrategy <*version*>**, and click **Next**. Options to upgrade your MicroStrategy environment are displayed.

3. Select **Intelligence Server Components**, and click **Next**. The MicroStrategy Authentication page opens.

4. Type the user name and password of a MicroStrategy system administrator, and click **Next**. The Select Components to Upgrade page opens.

5. For each project to add the R script functions to, select the **Re-execute the Project Logical Upgrade** check box.

6. For the Local Host Intelligence Server, select the **Re-execute Metadata Repository Upgrade** check box, and click **Next**. The Summary page opens.

7. Click **Finish** to begin the upgrade.

8. When the upgrade is complete, restart any MicroStrategy Intelligence Server that is connected to the project metadata that was upgraded. The R script functions are added to the list of Data Mining Functions available when creating metrics, including the functions `RScript`, `RScriptU`, `RScriptAgg`, `RScriptAggU`, and `RScriptSimple`.

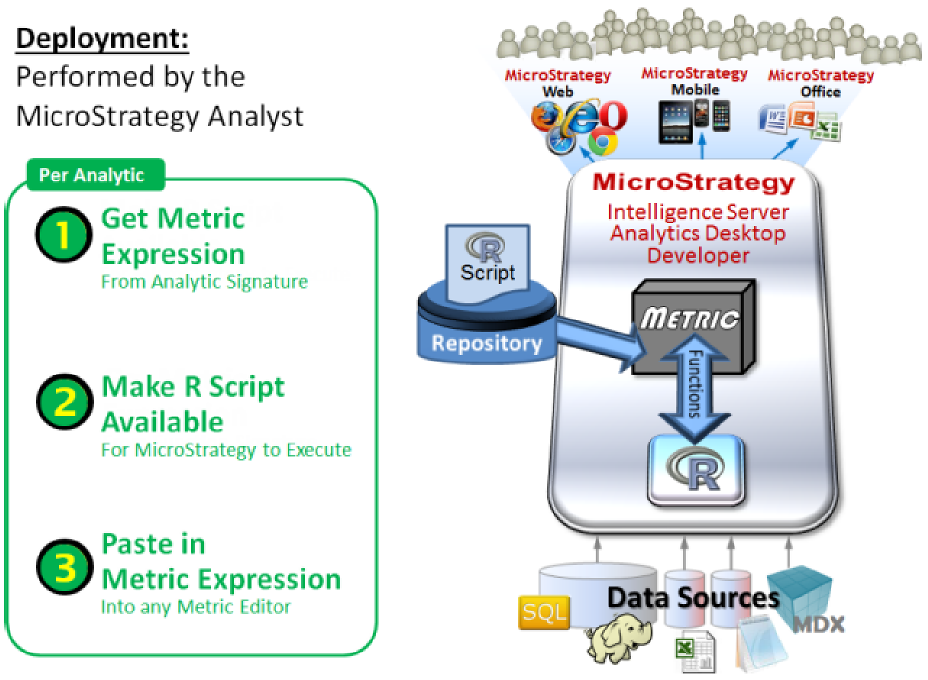
After you have installed the R script functions, the MicroStrategyR Package must be installed on systems that are being used to develop your R analytics. For detailed steps to install the MicroStrategyR Package, see *Chapter 4, Developing R Analytics for MicroStrategy*.

3

PERFORMING STATISTICAL ANALYSIS

Creating metrics with R analytics

With the R Integration Pack installed, you can begin to analyze the R analytic statistical information in MicroStrategy. The following diagram provides a high-level overview of the steps to use an R analytic in a MicroStrategy metric:



To begin performing R statistical analysis in MicroStrategy, complete the following steps:

1. *Retrieving the metric expression, page 16*
2. *Making the R script available, page 16*
3. *Including the R script in a metric, page 17*

Retrieving the metric expression

Once an R script that includes statistical analysis has been processed using the MicroStrategy deployR utility, it is ready to be integrated into your MicroStrategy environment.

Creating R scripts using R and processing them using the deployR utility is described in *Chapter 4, Developing R Analytics for MicroStrategy*.

To integrate the statistical analysis, you must retrieve the metric expression. You can open the `.R` file using a text editor. Within the file, search for the line that begins with one of the available R script functions, which includes `RScript`, `RScriptU`, `RScriptAgg`, `RScriptAggU`, and `RScriptSimple`.

For example, the `SeasonalForecasting.R` file included with the R Integration Pack includes the following metric expression:

```
RScript<_RScriptFile="SeasonalForecasting.R", _
InputNames="Target, Trend, Season", StringParam9="">
(Target, Trend, Season)
```

In addition to the `SeasonalForecasting.R` file included with the R Integration Pack, MicroStrategy provides example R script files that can be downloaded from the R script shelf. The metric expressions include comments which need to be removed manually using a text editor, or the example scripts can be processed using the deployR utility (see *Preparing your analytic for MicroStrategy: the deployR utility, page 38*).

This metric expression can then be used to help make the R script available (see *Making the R script available, page 16*) and finally begin to perform statistical analysis in MicroStrategy by creating a metric (see *Including the R script in a metric, page 17*).

Making the R script available

With an R script file and the associated metric expression (see *Retrieving the metric expression, page 16*), you are ready to make the R script available to your MicroStrategy environment. You can determine the required location for the R script based on the `_RScriptFile` parameter that is part of the metric expression:

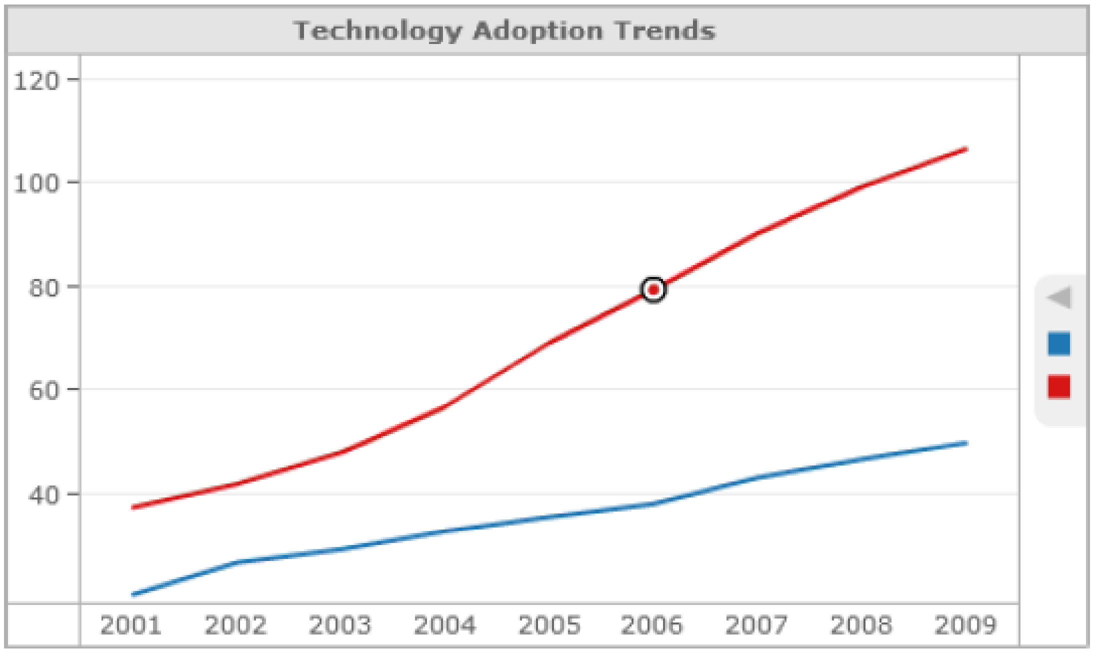
- If the `_RScriptFile` parameter includes only a file name, the R script must be stored in the default RScripts folder for the computer. This folder is created when the R Integration Pack is installed, and the default locations include:
  - 32-bit Windows environments: `C:\Program Files\R Integration Pack\RScripts`
  - 64-bit Windows environments: `C:\Program Files (x86)\R Integration Pack\RScripts`
  - UNIX and Linux environments: `/var/opt/R_Integration_Pack/RScripts`, or `$HOME/R_Integration_Pack/RScripts` if you do not have write access to `/var/opt/R_Integration_Pack/`.

- If the `_RScriptFile` parameter includes a full path and file name, the R script must be stored in the path specified.
- To provide a single location where all R scripts can be stored and accessed, see *Creating a centralized repository for R script files , page 8*.

Including the R script in a metric

You can begin to analyze the R analytic statistical information in MicroStrategy by creating a metric. A metric is a calculation that represents a business measure or key performance indicator, in this case, the R analysis. You can use metrics in reports, documents, and dashboards.

For example, trend analysis metrics have been created using R analytics. The metrics have been added to a visualization in a dashboard, as shown below:



The high-level steps for creating a metric and including an R script include:

1 Create a metric using your MicroStrategy product. R scripts can be integrated with all metrics available with your MicroStrategy products, such as standalone metrics that can be included in reports and dashboards or derived metrics that are created directly within a report or dashboard.

For detailed steps to create metrics, click the Help button or ? (question mark) in your MicroStrategy product to access the MicroStrategy documentation.

2 In the metric editor, paste the metric expression from your R script.

3 Map all of the inputs for the metric expression, which are included in parentheses and separated by commas at the end of the metric expression, to metrics. Each input must map to a single metric. To map an input, replace its name in the expression in the pane with a metric.

If you are mapping inputs for a derived metric, the metrics that you map to the inputs must all be included in the report or dashboard.

4 Make any additional changes to the metric and save your metric.

5 If you created a standalone metric, add the metric to a report, document, or dashboard.

6 Run the report, document, or dashboard to begin your R statistical analysis in MicroStrategy.

You can add images created using R scripts to a document to display the results of an analysis, help illustrate a trend, and so on. The image in the document is updated whenever the script to create the image is run again.

Reviewing and troubleshooting your R analysis

Along with analyzing the statistical data on your MicroStrategy report, document, or dashboard, you can also review the success or failure of the R script execution. Depending on whether the R script executed when executing the report, document, or dashboard that included the R analysis, you can do one of the following:

- Locate the `.Rdata` file in your working directory, open it, and explore the R environment that is saved when your function executes. The name of the `.Rdata` file is specified in the R script.
- If an error occurred or the metrics for your R script did not return any data, check the `DSSErrors.log` and `RScriptErrors.log` file for information on any errors detected. The developer of the R script determines what information is included in these log files, as described in *Implementing error handling, page 26*.
- When you receive an error message you can search for it within *Chapter 5, Troubleshooting*, which provides a potential cause and resolution to the problem.

Adding images made in R to documents and dashboards

You can add images created using R scripts (see *Creating images in R to add to MicroStrategy documents, page 24*) to a document or dashboard to display the results of an analysis, help illustrate a trend, and so on. The image in the document or dashboard is updated whenever the R script used to create the image is run again.

Contact your administrator to determine the location used to store images, which must be accessible by the MicroStrategy platform, as described in *Creating a repository for R images, page 8*. When including an R image in a document or dashboard, use this location as the path to the image. For additional information on including images in documents and dashboards, such as steps to add a dynamic image to a document, see the Document Creation Guide.

R analysis example

Below is the R script for the forecasting analytic. It is also available as part of the installation package in the default `RScripts` folder as a file called `SeasonalForecasting.R`. This example explains how to deploy this script with the MicroStrategy Tutorial project available with MicroStrategy Analytics Enterprise.

The header block for this R script includes the following metric expression information:

```
#Metric Expression:
RScript<_RScriptFile="SeasonalForecasting.R",
_InputNames="Target, Trend, Season", StringParam9="">
(Target, Trend, Season)
```

The inputs for the metric expression are included in parentheses and separated by commas at the end of the metric expression: `(Target, Trend, Season)`.

The following steps provide a brief overview of how to use the metric expression to deploy the example analytic to MicroStrategy Analytics Enterprise. The metric is placed in a report for analysis.

To deploy the sample R analytic

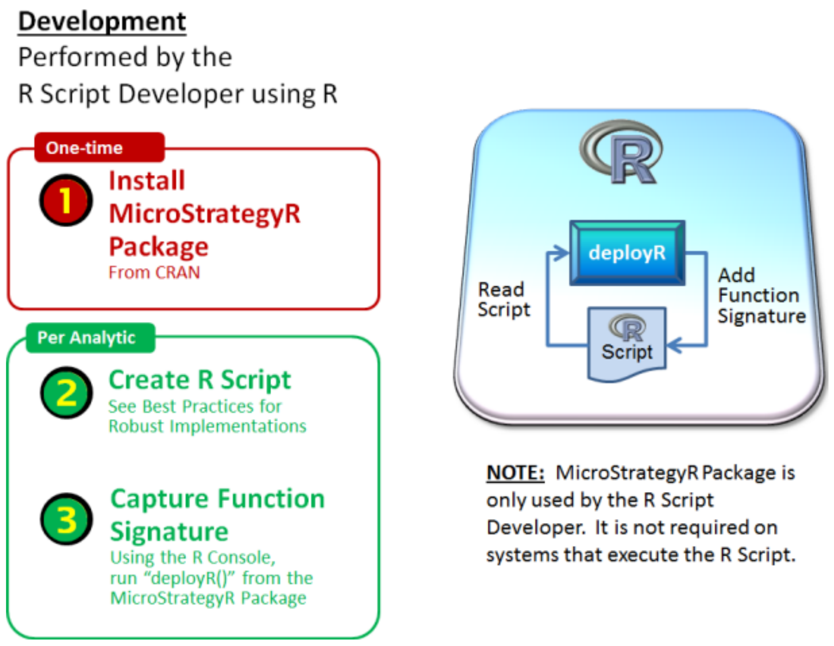
1. Copy the metric expression to the clipboard.
2. Open the MicroStrategy Tutorial project in either Developer or Web and run the report named `2 -- Monthly Revenue Forecast` located by default at `Tutorial\Public Objects\Reports\MicroStrategy Platform Capabilities\MicroStrategy Data Mining Services\Linear Regression\Monthly`.
3. When the report finishes executing, insert a new metric and give it a name such as `Forecast from R`.
4. Paste the metric expression from the clipboard into the Definition field of the new metric.
5. The inputs are listed at the end of the metric expression, in parentheses and separated by commas. Map the three inputs to MicroStrategy metrics using the following steps:
   a. Highlight **Target** in the expression and replace it with **Revenue** from the Report Objects.
   b. Highlight **Trend** in the expression and replace it with **Month Index** from the Report Objects.
   c. Highlight **Season** in the expression and replace it with **Month of Year** from the Report Objects.
6. Click **OK** to save the new metric and re-execute your report.

The values for your new metric generated by R match those from the Revenue Predictor (Monthly) metric, because they are both linear regression models trained with the same data as on this report.

4

DEVELOPING R ANALYTICS FOR MICROSTRATEGY

To allow users to perform statistical analysis in MicroStrategy using R analytics, the R analytics must first be implemented and deployed to MicroStrategy, as outlined in the steps below:


Development
Performed by the
R Script Developer using R
One-time
1 Install MicroStrategyR Package
From CRAN
Per Analytic
2 Create R Script
See Best Practices for Robust Implementations
3 Capture Function Signature
Using the R Console, run "deployR()" from the MicroStrategyR Package
deployR
Read Script
Add Function Signature
Script
NOTE: MicroStrategyR Package is only used by the R Script Developer. It is not required on systems that execute the R Script.

1 Install the MicroStrategyR package, described in *Installing the MicroStrategyR Package for R, page 22*.

2 Create the R scripts for your analytics, described in *Implementing the analytic in R for use in MicroStrategy, page 23*.

   The best practices to make your R script more robust are described in *Best practices: Making the R script robust, page 25*.

3. Capture an analytic's signature to the R script using the deployR utility and create a metric expression, as described in *Preparing your analytic for MicroStrategy: the deployR utility, page 38*.

For an example of deploying an R script as an R analytic in MicroStrategy, see *R analysis example, page 19*.

Installing the MicroStrategyR Package for R

The MicroStrategyR Package includes the R-based deployR utility, which is an interface used to prepare your R script for deployment to MicroStrategy.

The deployR utility adds a header comment block that reflects the analytic's signature to the R script. This information is used by MicroStrategy to pass data to R, execute the analytic, and then deploy the results.

You must install the MicroStrategyR Package on systems that are being used to develop your R analytics. R and any dependencies must be installed on systems that will execute the R analytics deployed to MicroStrategy. These dependencies include any add-on packages that are required for a given R analytic. The version requirements for your R environment are described in *Installing R, page 9*.

The high-level steps to install the MicroStrategyR package are as follows:

1. The MicroStrategyR Package can be installed using the Comprehensive R Archive Network (CRAN), available via the http://www.r-project.org home page. Alternatively, this package can be installed by typing this command into an R console:

   ```
   install.packages("MicroStrategyR")
   ```

   To be available to all users and applications, packages need to be installed into the default R library. To install the package into the default R library to provide availability to all users and applications, it is recommended that you use an account with administrative privileges and permissions to the default R library to install packages.

2. When the MicroStrategyR library is installed, load the library by typing the following command in the R Console:

   ```
   library(MicroStrategyR)
   ```

   This package depends on other R packages for graphics as well as the `GTK+` graphical environment. The absence of the `GTK+` graphical environment is common and can cause the following error messages:

   - Windows environments: Messages are displayed that mention missing `DLL` files. Click **OK** to dismiss any error messages encountered due to these missing requirements. When prompted, accept the installation of the `GTK+` graphical environment to complete the MicroStrategyR Package installation.

   - UNIX/Linux environments: The console displays errors related to other packages, such as the `RGtk2` package. You can download the most recent version of the GTK+ graphical environment for your operating system at sites such as http://ftp.acc.umu.se/pub/gnome/binaries. After downloaded, you must manually install the `GTK+` graphical environment.

3 When installed, you can launch the deployR utility by typing the following command in the R Console:

```
deployR()
```

After you have installed the components of the R Integration Pack, you can develop R analytics, described in *Implementing the analytic in R for use in MicroStrategy, page 23*.

Implementing the analytic in R for use in MicroStrategy

The R Integration Pack requires a general understanding of the MicroStrategy Business Intelligence environment, such as creating metrics and using them on reports.

To develop your R analytics, ensure that you or your organization's system administrator has installed the MicroStrategyR package on your development machine.

When implementing an analytic in R to use in MicroStrategy, review the following considerations:

- The inputs and outputs of the R analytic correspond to MicroStrategy metrics. MicroStrategy metrics represent values in a cell or column of data in a report, grid, dataset, and so on.
- The input and output variables can be either scalars (a single value) or vectors (one or more values). It is not possible to automatically pass tables, matrices, or data frames between MicroStrategy and R when using this approach. To mimic passing table-like structures between MicroStrategy and R, you can use the Repeated Parameter option for R analytics, which allows a varying number of inputs.
- In addition to passing values as metric arguments, a set of Boolean, numeric, and string parameters is available to pass into R scalar values that do not change from execution to execution. The script's working directory is also a special function parameter (see *Defining an R script's file location and working directory, page 23*).
- Graphs and plots can be saved to the file system to be included in MicroStrategy documents and dashboards, as described in *Creating images in R to add to MicroStrategy documents, page 24*.
- The R workspace can also be saved to the file system.

Defining an R script's file location and working directory

The R scripts for your R analytics store the R script file, error log, and other supporting files in the directories described below.

Location of the R script file

The R script file is located in the R script's directory, as specified in the `_RScriptFile` parameter for the R analytic's metric in MicroStrategy.

If the path or R script file name listed in the `_RScriptFile` parameter for R does not exist or is defined with a URL, the execution of the R script will fail. An R script failure can cause different error results in MicroStrategy, depending on whether the R script is used by a stand-alone metric or a derived metric, as described in *Implementing error handling, page 26*.

- If the `_RScriptFile` parameter does not contain a path, the directory defined in the `_WorkingDir` parameter for R is used.
- If the `_RScriptFile` parameter does not contain a path and an existing directory is not defined in the `_WorkingDir` parameter for R, the directory defined in the registry key `RScriptsFolder` is used.
- If the directory could not be identified using the locations listed above, the default `RScripts` folder defined during the R Integration Pack installation is used. This is typically *`InstallPath`*`\RScripts`.

Location of the R script working directory

The working directory stores an error log and supporting files in the following location:

- If a working directory is specified for R, the directory defined in the `_WorkingDir` parameter for R is used.
- If a working directory is not specified for R or the specified directory does not exist, the location of the R script file, as determined in *Location of the R script file, page 24* above, is used.
- If the directory could not be identified using the locations listed above, then the R Integration Pack installation folder is used.

Creating images in R to add to MicroStrategy documents

You can create images using R scripts, which can be included in MicroStrategy documents to display the result of an analysis, illustrate a trend, and so on. The image in the document is updated whenever the R script used to create the image is run again.

You can also create dynamic images, which allow different images to be displayed, based on an attribute or metric. For example, a document is paged by Region. You can display a graph made in R for that particular region in the document.

To create images in R to use in MicroStrategy

1. Define the R script's working directory to the location where images for MicroStrategy are stored. All images must be saved in a location accessible by the MicroStrategy platform, which can be defined by your system administrator. For guidelines on where to save images, see *Creating a repository for R images, page 8*.

2. Within the R script, include code to create an image in R and save the image in the working directory. For example, the `plot` command can be used to create images of statistical graphs. Refer to your third-party R documentation for information on how to create an R image.

   If you are creating dynamic images, save all of the images in the same directory. Give each image the name of its corresponding attribute element or metric value. For example, if you created an image you want to display when the value of the Region attribute is Northeast, you would name the image `Northeast`. If you have multiple dynamic images you want to display for an attribute element or metric value, ensure that the correct image is used by giving each dynamic image a different name that contains the attribute element or metric value. For information on how to name dynamic images, see the Document Creation Guide.

Best practices: Making the R script robust

The R Integration Pack automates the execution of an R script and the delivery of its results to MicroStrategy users. Ideally, if an R script works as expected when executed in the R console, the script should work properly when deployed to MicroStrategy. To avoid potential errors, you can make preparations to handle situations that can arise from differences in how data is supplied, how exceptions are handled, and even differences in the operating environment. For example, the deployed system can be different from the R script developer's system.

This section provides guidance to help make the R script robust enough to handle real world circumstances that can arise and to make troubleshooting as easy as possible. The following best practices should be applied to the R script before deploying it to MicroStrategy:

The following best practices are provided to help utilize R scripts with the MicroStrategy R Integration Pack. For general R coding principles and examples, refer to your third-party R documentation.

- *Implementing error handling, page 26*
- *Saving the R workspace , page 28*
- *Configuring dual execution modes, page 29*
- *Creating a data frame, page 30*
- *Using MicroStrategy names for R variables, page 31*
- *Installing required packages, page 31*

- *Upgrading R, page 34*
- *R environment with the R Integration Pack, page 34*
- *Combining all techniques for a robust script, page 35*

The simple R script for forecasting seasonal data, shown below, will be used to provide examples of how to implement these best practices to make your R scripts robust.

Simple R script

```
#Create a data frame from the input variables
df <- data.frame(cbind(Target, Trend, Season))
#Train model on all records with Target values
model <- lm(Target ~ Trend + factor(Season),data=df[!is.na(Target), ])
#Return predictions from the model
Forecast <- predict(model, newdata = df[, -1])
```

Implementing error handling

Errors or exceptions encountered while executing code enclosed within a `tryCatch` function are caught and returned to MicroStrategy in the R variable called `mstr.ErrMsg`. (This variable is a reserved name for this purpose).

If an error is returned, MicroStrategy creates an error log file with the message. If your script generates any output files, this error log file should appear in the same location. It is good practice to use a working directory specific to your script to keep its error log files separate from those from other scripts.

The error log file is called `RScriptErrors.log` and users can find it in the supporting directory for the R script (see *Defining an R script's file location and working directory, page 23*).

- If the R analytic is deployed as a stand-alone metric, the error causes the report or document execution to fail, resulting in the error message being displayed to the user.
- Errors in R analytics deployed as derived metrics or stand-alone metrics that are configured as smart metrics typically do not cause report or document execution to fail. (A smart metric calculates subtotals on the individual elements of a metric.) Instead, null results are displayed as if empty values were returned by the R analytic. Empty results can indicate that there was a problem executing the R analytic and that you should check the log file for more details.

For steps to define stand-alone metrics as smart metrics, also referred to as smart totals, as well as background information on smart metrics, refer to the Advanced Reporting Guide.

In both cases described above, a message is logged in the `DSSErrors.log` file, unless logging is disabled using the MicroStrategy Diagnostics Utility.

The `DSSErrors.log` is typically found at *`X:`*`\Program Files (x86)\Common Files\MicroStrategy\Log`, where *`X:`* is the drive where MicroStrategy is installed.

In addition to errors caught by the `tryCatch` function, other conditions can result in error logging, including:

- Errors during the processing of registry entries, for example, problems with the keys associated with the *`InstallPath`* and `RScriptsFolder`. These errors do not cause a report to fail.
- Errors loading the R library or initializing the R environment.
- Errors parsing the R script header block.
- Data type mismatches.
- Failure locating the R script specified by the `_RScriptFile` parameter for the R analytic's metric in MicroStrategy (see *Location of the R script file, page 24*). This error can occur if a bad path or file name is specified, or if the script is not found at the location specified by the `RScriptsFolder` registry key.

The example R script described in *R analysis example, page 19* includes a command to set the working directory. It is recommended that this line of code is moved to the body of the script, within a `tryCatch` function wrapper. This ensures that any potential failure of the `setwd()` command is caught.

The following example shows how to wrap the R Integration Pack forecasting example code in a `tryCatch` function to catch errors:

R script with tryCatch

```
#tryCatch for Exception Handling
mstr.ErrMsg <- tryCatch({
```

```
#Create a data frame from the input variables
df <- data.frame(cbind(Target, Trend, Season))
#Train model on all records with Target values
model <- lm(Target ~ Trend + factor(Season),data=df[!is.na(Target),
])
#Return predictions from the model
Forecast <- predict(model, newdata = df[, -1])
```

```
#Print success message when run from the console
try(print("Success!"))
#If we made it here, no errors were caught
mstr.ErrMsg <- ""
```

R script with tryCatch

`#Catch block to report an error`
`}, error = function(err) {`
` #Print error message if run from console`
` try(print(err))`
` #Return error message`
` return(err$message)`
`})`

Logging warning messages and errors from R

Warning messages related to R code inform users of potential issues that may be important, but are not severe enough to halt execution of the statistical analysis.

You can define your R scripts that are executed by MicroStrategy to log all warning messages to a log file. This provides a way to review warning messages from R that occurred when the R script was executed by MicroStrategy.

The following code can be added to the beginning of the try block of your R script to include both warning and error messages from R to the `RscriptErrors.log` file:

```
warning_file = file("RScriptErrors.log", open = "wt")
sink(warning_file, type = "message")
```

At the end of the try block of your R script, and before the line `mstr.ErrMsg <- ""`, add the following line of code:

```
sink(type = "message")
```

At the beginning of the error block, add the following line of code:

```
sink(type = "message")
```

When the R script is executed by MicroStrategy, warning messages and errors from R are logged to `RScripErrors.log` for review. For the cause and resolution to potential errors, see *Chapter 5, Troubleshooting*.

Saving the R workspace

R can save its workspace to the file system. This is helpful when you need to review the results of an analysis. When you execute a script from the R console, you can easily inspect the state of the workspace and its objects. It is helpful to capture the R workspace when MicroStrategy executes the script. The workspace is a valuable tool for the R developer to use when troubleshooting problems or verifying results.

An example of code inserted to save a workspace is shown below. The addition (highlighted with bold) to the R script shown below saves objects from the R workspace.

R script that includes saving the R work space

```
#Create a data frame from the input variables
df <- data.frame(cbind(Target, Trend, Season))
#Train model on all records with Target values
model <- lm(Target ~ Trend + factor(Season),data=df[!is.na(Target),
])
#Return predictions from the model
Forecast <- predict(model, newdata = df[, -1])
#Persist objects to file
save(list=c("df", "model", "Forecast"), file=paste(FileName,
".Rdata", sep = ""))
```

Configuring dual execution modes

When developing a new analytic in R, a common practice is to use an iterative process to ensure that the analytic works as expected. That process usually involves the R script taking inputs that are either created by the script or read from a data source, such as a file or a database. The results of the script are usually returned to the R console. But when the script is deployed to MicroStrategy, MicroStrategy provides the inputs to the R script and uses the R script outputs.

While it is possible to have the same analytic implemented in two different scripts, one for running from the console and one for execution by MicroStrategy, this approach requires maintaining two scripts. In this scenario, it can be difficult to keep the scripts synchronized as changes occur.

MicroStrategy provides the execution flag `mstr.ExFlag`, which exists only when MicroStrategy executes the R script. This means that you can use the existence of the flag to determine if MicroStrategy executed the R script. You can develop your script to use this flag and react to whether the script is run from the R console or executed by MicroStrategy.

The example shown below includes code inserted to adapt the R Integration Pack forecasting script example to generate its own data when it is not executed by MicroStrategy. The addition is highlighted with bold.

Code to configure dual execution modes

```
#Get data
#If this is executed by MicroStrategy
if(exists("mstr.ExFlag")) {
 #Create a data frame from the input variables
 df <- data.frame(cbind(Target, Trend, Season))
```

Code to configure dual execution modes

```
 #If InputNames is non-empty
 if(length(mstr.InputNames) > 0) {
  #Name these variables
  colnames(df) <- mstr.InputNames
 }
#If this is NOT via a MicroStrategy Report Execution
} else {
 #Set random number seed for consistency
 set.seed(42)
 #Set Trend variable for 48 months
 Trend <- seq(1:48)
 #Set Season variable for 4 years of 12 months
 Season <- rep(seq(1:12),4)
 #Set 3 years of linear but noisy values for the Target
 Target <- (seq(1:36)*(0.8+(0.4*runif(36,0,1))))
 #Add the forecast horizon
 Target <- append(Target, c(rep(NA, 12)))
 #Create a data frame from the input variables
 df <- data.frame(cbind(Target, Trend, Season))
 #Set the name for saving output
 FileName <- "SeasonalForecasting_console"
}
```

```
#Modeling
#Train model on all records with Target values
model <- lm(Target ~ Trend + factor(Season),data=df[!is.na(Target),
])
#Return predictions from the model
Forecast <- predict(model, newdata = df[, -1])
```

Creating a data frame

Since most analytics operate on a table of data, known as a data frame in R, it is often helpful to combine input variables in a data frame. In the seasonal forecast data example, the three input variables (target, trend, and season) are combined into a data frame. Since both execution flows use the same data frame object, you can compare the data used when MicroStrategy executes the script with the data used when the R console executes the script. While there are reasons to avoid using a data frame in either flow, such as if the data frame causes undesired side effects like performance problems, having the data in the same object for both flows allows for easier comparison.

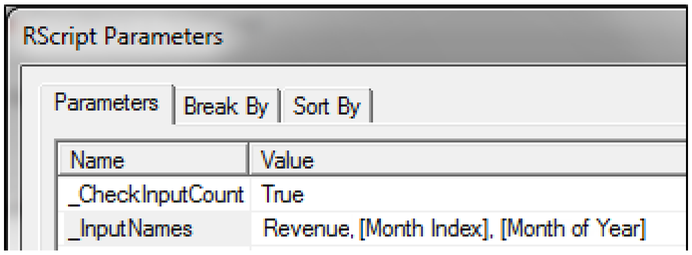
Using MicroStrategy names for R variables

The R Integration Pack passes data from MicroStrategy into R for execution by mapping MicroStrategy metrics to R variables. For the R environment to be able to use the MicroStrategy metric names in objects and graphics that R generates, the names associated with the inputs from MicroStrategy need to be passed to R.

The names of the metrics in MicroStrategy may not always match the names of the variables in R. For R analytics, R variables typically have generic names, as is the case with the R script example used in this section: Target, Trend, and Season. To forecast monthly revenue, the corresponding MicroStrategy metrics could be Revenue, Month Index, and Month of Year.

To provide this support, each R script function has an `_InputNames` function parameter to allow you to pass MicroStrategy names to R. When you create a metric to display the results of your R analytic in the MicroStrategy Metric Editor, you can define the `_InputNames` function. You can use the metric expression generated for your R analytic (see *Preparing your analytic for MicroStrategy: the deployR utility, page 38*) to retrieve the inputs for your R script. The inputs are located at the end of the metric expression between the final parentheses "(...)". You can then replace each input name with the associated MicroStrategy metric that provides its data, and copy the inputs to the `_InputNames` function parameter.

For this example, the metric expression would be "`RScript(Revenue, [Month Index], [Month of Year])`". Copy the metric names (between the parentheses) and paste them into the `_InputNames` function parameter to define the R analytic's metric parameter as `Revenue, [Month Index], [Month of Year]`. An example using MicroStrategy Developer's Metric Editor is shown below:



Name	Value
_CheckInputCount	True
_InputNames	Revenue, [Month Index], [Month of Year]

Installing required packages

R scripts commonly have dependencies on R packages, which are modules that provide additional functionality beyond the out-of-the-box features included with the standard R installation. R packages are installed from CRAN mirror repository sites. The machine executing the script must have access to the Internet to download any missing packages.

⚠ If an R 3.x installation is not available from http://CRAN.R-project.org for your version of Linux, third-party tools such as the Yellowdog Updated, Modified (Yum) and Extra Packages for Enterprise Linux (EPEL) can be used to install and update your R installation. Refer to your third-party R documentation available at http://www.r-project.org/ for information on installing and updating R using these tools.

A default CRAN mirror repository site can be established by modifying the following code, by replacing `CRANMirrorURL` with the desired CRAN mirror URL in the `RProfile` file located in the `/library/base/R` directory of the R installation:

```
options(repos= c(CRAN="CRANMirrorURL"))
```

For example, you can establish `http://cran.rstudio.com` as the default CRAN mirror repository as shown below:

```
options(repos= c(CRAN="http://cran.rstudio.com"))
```

Packages can also be installed manually using the R console, as shown below:

```
install.packages("RPackage")
```

⚠ Packages need to be installed into the default R library to be available to all users and applications. Administrative privileges are typically required to install into the default R library. Therefore, to install a package into the default R library to provide availability to all users and applications, it is recommended to use an account with administrative privileges and permissions to the default R library to install packages.

When you install packages manually, the R console will prompt you to identify a CRAN mirror repository to use. To avoid this prompt, specify a CRAN mirror repository as part of the manual installation from the R console, as shown below:

```
install.packages("RPackage",
repos="http://cran.rstudio.com")
```

The sample R script below avoids any required user intervention to install R packages by automatically checking for and installing R packages if necessary. This approach assumes that the machine executing the script has access to the Internet to download any missing packages. For environments that do not have access to the Internet, other workflows need to be used to add any required R packages to the environment.

⚠ Installing R packages can require administrative access to folders. The sample R script below includes error handling if the user employing the R script does not have the administrative access required to complete the package installation. In these cases, the sample R script below installs the package in a personal folder for the user that he has access to, or a new folder is created. This error handling for installing packages should be included in R scripts used in MicroStrategy to ensure that users without administrative access can employ the R scripts.

R script that checks for and installs missing packages

```
#Check to see if package(s) are installed, install if not and then
load

#pkgs is a vector of strings with length >= 1

CheckInstallPackages <- function(pkgs){

#For each pkg in pkgs (attempt to load each package one at a time):

 x <- lapply(pkgs, function(pkg){

  #Load the package if available,

  if(!do.call("require", list(pkg))) {

   #Silently attempt to install into the default library

   try(install.packages(pkg,
lib=.Library,repos="http://cran.rstudio.com"))

   #Now attempt to load the package, catch error if it wasn't
installed

   tryCatch(do.call("library", list(pkg)),

    #Catch if we're unable to install into the default library

    error = function(err) {

     #If non-interactive, install into this user's personal library

     if(!interactive()) {

      #Get the path to this user's personal library

      personalLibPath <- Sys.getenv("R_LIBS_USER")

      #If the personal library is not in the list of libraries

      if(is.na(match(personalLibPath, .libPaths()))) {

       #Then create the personal library

       dir.create(personalLibPath, recursive = TRUE)

       #And add the personal library to the list of libraries

       .libPaths(personalLibPath)

      }

      #Attempt to install the package into the personal library

      #If this fails, raise the error back to the report

      install.packages(pkg,
lib=personalLibPath, repos="http://cran.rstudio.com")

      #Finally, attempt to load the package

      do.call("library", list(pkg))

 }})}})
```

R script that checks for and installs missing packages
`}`

`#Load the PMML package`
`CheckInstallPackages(c("pmml"))`
`#Save the model as PMML`
`saveXML(pmml(model), paste(FileName,".xml", sep=""))`

Upgrading R

Several times a year, new releases of R become available from http://CRAN.R-project.org. The following steps are recommended to minimize the possibility of problems when upgrading to a new R release to use with the R Integration Pack.

If an R 3.x installation is not available from http://CRAN.R-project.org for your version of Linux, third-party tools such as the Yellowdog Updated, Modified (Yum) and Extra Packages for Enterprise Linux (EPEL) can be used to install and update your R installation. Refer to your third-party R documentation available at http://www.r-project.org/ for information on installing and updating R using these tools.

To upgrade your R environment

1 Uninstall the old version of R. Uninstalling R removes files from the initial installation, but not packages that have been installed or updated.

   See the third-party R documentation for steps to uninstall R for your machine configuration.

2 Install the new version of R from http://CRAN.R-project.org.

3 Copy any installed packages from the old installation library folder to the new installation library folder.

4 In the new R console, run the following command:

   ```
   update.packages(checkBuilt=TRUE, ask=FALSE)
   ```

5 Delete any files left from the old installation.

R environment with the R Integration Pack

The R Integration Pack uses local environments for each execution. However, when developing R scripts to be executed in the R Integration Pack environment, use the following rules to ensure proper and expected behavior:

- Always initialize R variables before use.

- Do not rely on a test of an R variable's existence to indicate the state of the script, since the variable could have been created by an entirely different script.

Combining all techniques for a robust script

The simple R script example, unmodified, is shown again below:

Simple R script

```
#Create a data frame from the input variables

df <- data.frame(cbind(Target, Trend, Season))

#Train model on all records with Target values

model <- lm(Target ~ Trend + factor(Season),data=df[!is.na(Target),
])

#Return predictions from the model

Forecast <- predict(model, newdata = df[, -1])
```

By using all of the recommendations above, the script shown below is more robust to handle errors, generates PMML, saves important objects from the R environment for future analysis, and can be executed using the R console or MicroStrategy:

Robust R script

```
#tryCatch for Exception Handling

mstr.ErrMsg <- tryCatch({
```

```
#Working Directory if executed by MicroStrategy

if(exists("mstr.WorkingDir")) setwd(mstr.WorkingDir)
```

```
#Check to see if package(s) are installed, install if not and then
load

#pkgs is a vector of strings with length >=1

CheckInstallPackages <- function(pkgs){

#For each pkg in pkgs (attempt to load each package one at a time):

 x <- lapply(pkgs, function(pkg){

  #Load the package if available,

  if(!do.call("require", list(pkg))) {

   #Silently attempt to install into the default library

   try(install.packages(pkg,
lib=.Library,repos="http://cran.rstudio.com"))

   #Now attempt to load the package, catch error if it wasn't
installed
```

Robust R script

```
   tryCatch(do.call("library", list(pkg)),
    #Catch if we're unable to install into the default library
    error = function(err) {
     #If non-interactive, install into this user's personal library
     if(!interactive()) {
      #Get the path to this user's personal library
      personalLibPath <- Sys.getenv("R_LIBS_USER")
      #If the personal library is not in the list of libraries
      if(is.na(match(personalLibPath, .libPaths()))) {
       #Then create the personal library
       dir.create(personalLibPath, recursive = TRUE)
       #And add the personal library to the list of libraries
       .libPaths(personalLibPath)
      }
      #Attempt to install the package into the personal library
      #If this fails, raise the error back to the report
      install.packages(pkg,
lib=personalLibPath, repos="http://cran.rstudio.com")
      #Finally, attempt to load the package
      do.call("library", list(pkg))
 }})}})
}
```

Robust R script

```
#Get data
#If this is executed by MicroStrategy
if(exists("mstr.ExFlag")) {
 #Create a data frame from the input variables
 df <- data.frame(cbind(Target, Trend, Season))
 #If InputNames is non-empty
 if(length(mstr.InputNames) > 0) {
  #Name these variables
  colnames(df) <- mstr.InputNames
 }
#If this is NOT via a MicroStrategy Report Execution
} else {
 #Set random number seed for consistency
 set.seed(42)
 #Set Trend variable for 48 months
 Trend <- seq(1:48)
 #Set Season variable for 4 years of 12 months
 Season <- rep(seq(1:12),4)
 #Set 3 years of linear but noisy values for the Target
 Target <- (seq(1:36)*(0.8+(0.4*runif(36,0,1))))
 #Add the forecast horizon
 Target <- append(Target, c(rep(NA, 12)))
 #Create a data frame from the input variables
 df <- data.frame(cbind(Target, Trend, Season))
 #Set the name for saving output
 FileName <- "SeasonalForecasting_console"
}
```

```
#Modeling
#Train model on all records with Target values
model <- lm(Target ~ Trend + factor(Season),data=df[!is.na(Target),
])
#Return predictions from the model
```

Robust R script

```
Forecast <- predict(model, newdata = df[, -1])
```

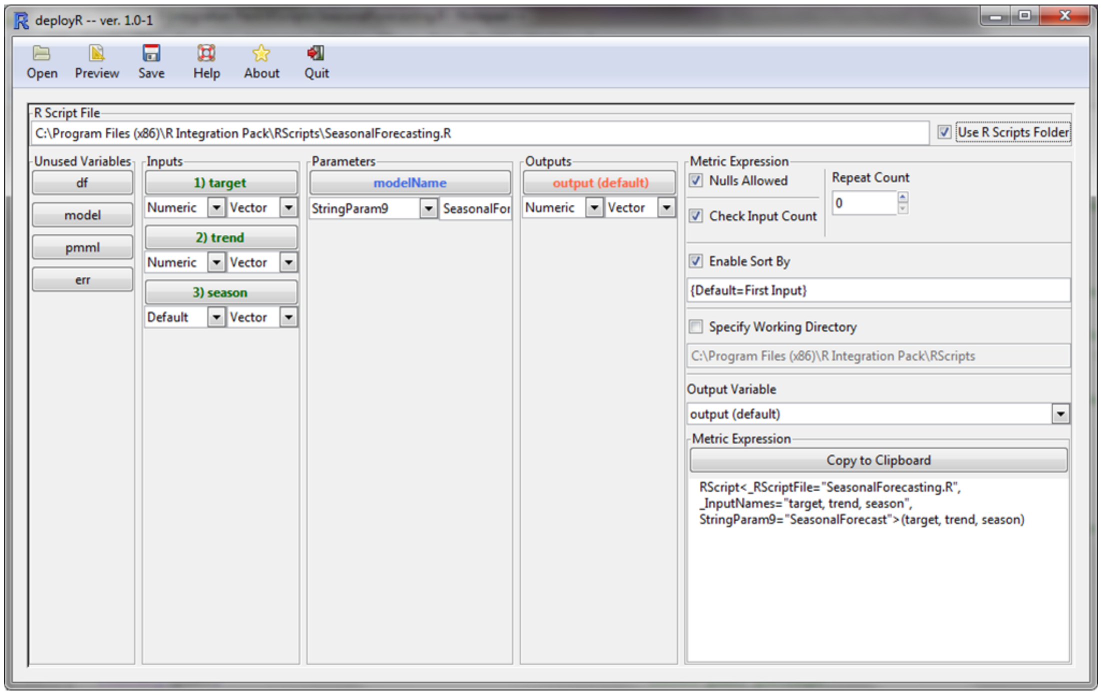
```
#If FileName is not an empty string
if(nchar(FileName)>0) {
 #Persist objects to file
 save(list=c("df", "model", "Forecast"), file=paste(FileName,
".Rdata", sep = ""))
 #Load the PMML package
 CheckInstallPackages(c("pmml"))
 #Save the model as PMML
 saveXML(pmml(model), paste(FileName,".xml", sep=""))
}
```

```
#Print completion message when run from the console
try(print("Success!"))
#If we made it here, no errors were caught
mstr.ErrMsg <- ""
#Catch block to report an error
}, error = function(err) {
 #Print error message if run from console
 try(print(err))
 #Return error message
 return(err$message)
})
```

Preparing your analytic for MicroStrategy: the deployR utility

After implementing an R analytic in an R script that is ready to be deployed to MicroStrategy, you must define how MicroStrategy interacts with the analytic by capturing the analytic's signature. The analytic's signature is a description of the number and nature of the inputs and outputs to the R script, along with any other information needed for MicroStrategy to execute the script properly.

The deployR utility is an R-based utility that analyzes an R script, captures its signature, and creates the metric expressions that execute the function within MicroStrategy. The following image shows the deployR utility.



R Script File: C:\Program Files (x86)\R Integration Pack\RScripts\SeasonalForecasting.R

Inputs: 1) target — Numeric, Vector; 2) trend — Numeric, Vector; 3) season — Default, Vector

Parameters: modelName — StringParam9, SeasonalFor

Outputs: output (default) — Numeric, Vector

Repeat Count: 0

Enable Sort By: {Default=First Input}

Specify Working Directory: C:\Program Files (x86)\R Integration Pack\RScripts

Output Variable: output (default)

```
RScript<_RScriptFile="SeasonalForecasting.R",
_InputNames="target, trend, season",
StringParam9="SeasonalForecast">(target, trend, season)
```

The standard workflow for using the deployR utility to capture the signature of your R analytic and deploy it to MicroStrategy follows.

To prepare R scripts using the deployR utility

1 From an R console, open the deployR utility using the following commands:

```
library(MicroStrategyR)
deployR()
```

The deployR utility opens.

2 Click the **Open** button at the upper left to open your R script.

After you open your R script, the deployR utility parses the script to identify all potential variables.

- If this script contains the MicroStrategy header block at the top, then the header information is used to configure the utility. Any unidentified variables are displayed in the Unused Variables column.
- If the script does not contain a MicroStrategy header block, the deployR utility attempts to determine whether a variable is an input or an output, based on the first occurrence of that variable in the script. If the variable's first occurrence assigns it a value, it is considered an output; otherwise, it is designated as an input.
- For new variables, the default Data Type is Numeric and the default Parameter Type is Vector.

3 Determine the path that is used for the R script using one of the following methods:

- To include the selected R script's file name and path in the metric expression, clear the **Use R Script Folder** check box. When executed, the R script file name and path you specify will be used. If the R script is not found, execution will fail and an error will be logged, if possible.

  If you clear this check box, the same folder path must exist or be accessible on the machines where this R script is provided to support execution in MicroStrategy. For more information, see *Creating a centralized repository for R script files , page 8*.

- To include only the file name of the R script in the metric expression, select the **Use R Script Folder** check box. When executed, the R Script Folder is searched for the specified script. If the R script is not found, execution will fail and an error will be logged, if possible. The default location for the R Script Folder is `RIntegrationPackInstallFolder\RScripts`. This location is controlled by the `HKLM\SOFTWARE\\MicroStrategy\R Integration Pack\RScriptsFolder` registry key.

4 Modify the definition of each variable as required to match the function's logic.

a Drag and drop variables to place them in the appropriate columns:

- **Unused Variables**: Variables that appear in the R script but are not passed between MicroStrategy and R as either inputs, outputs, or parameters. The order of unused variables does not affect the R script execution.
- **Input**: Data that is imported into R from MicroStrategy. The order of inputs, from top to bottom, determines the order of arguments passed in from MicroStrategy, from left to right.
- **Parameter**: Data that is passed as one of the various function parameters available for passing scalar values from MicroStrategy to R. These function parameters include Boolean parameters, numbers, and strings. Parameters are typically used for values that are changed infrequently or values that are not determined from other metrics.

  Use the **Parameter** drop-down list to specify which parameter to use. Each parameter can only be used for one variable. The order of parameters does not affect the R script execution.
- **Output**: Data that is returned from R to MicroStrategy. If there is more than one output, the first output is considered the default output. The order of any additional outputs does not affect the signature.

b Each variable must be configured appropriately, as described below:

a Set **Data Type** to one of the following options:

— **Numeric**: Indicates variables that contain numbers.

— **String**: Indicates variables that contain text.

— **Default**: Indicates that the data type defined by MicroStrategy should be used. This setting can be used for inputs only. It is useful for scripts that use categorical variables, such as Month. This type of categorical variable can be strings (such as Jan, Feb, Mar, and so on) or numbers.

   b. Set **Parameter Type** to one of the following options:

      — **Vector**: Indicates a variable that contains one or more values.

      — **Scalar**: Indicates a variable that contains only one value.

5. If one or more of the input variables form a repeated argument, define a value for **Repeat Count**.

   This option identifies an input that can vary in quantity; such variables are known as repeated arguments because they represent a varying number of variables. The Repeat Count value specifies how many of the input variables can be repeated, counting backwards from the last variable. These variables always occur at the end of the list of arguments. These variables appear in the Inputs column with an asterisk (*). Examples include:

   - A predictive analytical function supports one target variable Y (the dependent variable) and an indeterminate number of explanatory variables X (independent variables). Establish this configuration by defining Y as the first variable, defining X as the second variable, and defining a Repeat Count value of 1. The deployR utility recognizes that Y is the first argument passed into the function, followed by one or more X variables.
   - A predictive analytical function supports one target variable Y (the dependent variable) and an indeterminate number of explanatory, independent variable pairs, X1 and X2. X1 is a numeric identifier for an item and X2 is its text description. By defining Y as the first input, X1 as the second, X2 as the third, and a Repeat Count value of 2, the deployR utility recognizes that Y is the first argument and there can be one or more pairs of X1 and X2 variables passed into the R script.

   While defining a Repeat Count allows for additional metrics to be included as inputs passed back to R, the number of inputs provided in the metric expression generated by deployR is not affected by the value specified for Repeat Count. Additional metrics can be included in the metric expression when *Including the R script in a metric, page 17*.

6. You can define the metric that utilizes the R script using the **Metric Specification** section, which contains the following options:

   - **Nulls Allowed**: Controls whether records containing null values are to be passed in as inputs to your analytic:
     - By default this option is selected and null values are included in the analysis.
     - If this option is cleared, all records containing null values are eliminated from the analysis.
   - **Check Input Count**: Controls whether MicroStrategy ensures that the number of inputs to the metric matches exactly with the number of inputs specified in the function's signature:
     - By default, the option is selected. If it is selected and the number of inputs is different, a warning message is returned when using the R script in MicroStrategy.

    - If the option is cleared and the number of inputs is different, the script execution will attempt to proceed.

- **Enable Sort By**: Controls the sorting of records before the data is passed to R:
    - By default, the option is selected. If this option is selected, the first input must be a vector, since the default behavior sorts records in ascending order by the first input. To specify a particular sorting criterion, you can type the sort by value in the field below the check box.
    - If this option is cleared, the order of records passed into R is determined by MicroStrategy automatically.

- **Specify Working Directory**: Allows you to specify a working directory for your R scripts used in MicroStrategy, without affecting your R configuration:
    - By default, this option is cleared and the path provided for each R script is used.
    - To specify a working directory for MicroStrategy to search for R scripts, select the check box and specify a working directory in the field below the check box. MicroStrategy does not alter R's working directory, which is otherwise determined by R.

- **Output Variable**: Allows the user to control which variable is returned to MicroStrategy. The first output of an R script is selected by default. This output variable is not included in the metric expression.

7. To review the changes before saving, click **Preview**.
8. After you have configured the variables and specified the metric options, you can save the analytic's signature to the R script by clicking **Save**.
9. After saving the signature to your R script, the deployR utility provides the metric expression for your analytic. The Metric Expression pane at the bottom right of the dialog box shows the metric expression that defines how MicroStrategy interacts with your function. To complete the creation of a metric for your R analytic and test your R analytic:

    a. Click **Copy to Clipboard**, and then paste this metric expression into any MicroStrategy metric editor, including the metric editors available for MicroStrategy Developer, MicroStrategy Web, MicroStrategy Desktop, and MicroStrategy Visual Insight.

    b. Map each of the inputs of the expression, which are included between parentheses, to MicroStrategy metrics that provide the data for statistical analysis. An example of this modification is provided in *R analysis example, page 19*.

    c. If any errors are encountered when executing reports, documents, or dashboards that include the metric, refer to *Chapter 5, Troubleshooting* for potential error messages and their resolutions.

Modifying the R analytic deployment example

Below is the R script for the forecasting analytic described in *Combining all techniques for a robust script, page 35*. It is also available as part of the installation package in the default `RScripts` folder as a file called `SeasonalForecasting.R`.

The header block for this R script is as follows:

```
#MICROSTRATEGY_BEGIN
#
#RVAR target -input -numeric -vector
#RVAR trend -input -numeric -vector
#RVAR Season -input -vector
#
#RVAR FileName -parameter StringParam9
#
#RVAR Forecast -output -numeric -vector
#Metric Expression:
RScript<_RScriptFile="SeasonalForecasting.R",
_InputNames="Target, Trend, Season", StringParam9="">
(Target, Trend, Season)
#
#MICROSTRATEGY_END
```

This header includes the following features:

- Lines that begin with `#` are comments, and are ignored by R.
- MicroStrategy processes the contents between the `#MICROSTRATEGY_BEGIN` and `#MICROSTRATEGY_END` lines to understand the analytic's signature. You should not edit this block manually because doing so can cause errors.
- If you modify the R script or want to change anything else, open the R script using the deployR utility. The deployR utility automatically restores everything in the header, as well as looks for any new variables in the script. You can then use the deployR utility to make adjustments. Saving any changes replaces the existing header block with an updated one, and provides the updated metric expression as well.

The metric expression, highlighted with bold text above, is included in this header block and is ready to copy and paste into a metric definition. For steps to deploy this script with the MicroStrategy Tutorial project available with MicroStrategy Analytics Enterprise, see *R analysis example, page 19*.

5

TROUBLESHOOTING

The MicroStrategy R Integration Pack returns error and warning messages to help troubleshoot potential issues.

You can search for error messages within this guide, to find a potential cause and resolution to a problem. Be aware that error messages can contain names of variables, data types, file names, and other options that are specific to your environment. If you cannot find the error message by searching for the entire error message, try searching for one or several words in the error message.

- *Locating error messages, page 44*: Information on how to locate and review any error messages that are returned when using the R Integration Pack.
- *Troubleshooting your installation, page 45*: Troubleshooting steps for error messages that are potentially caused by problems with the installation or setup of the R Integration Pack or other supporting files or systems.
- *Troubleshooting the development of R scripts, page 46*: Troubleshooting steps for error messages that are potentially caused when you create your R script or process it with the deployR utility.
- *Troubleshooting R integration in MicroStrategy, page 50*: Troubleshooting steps for error messages that are potentially caused when R statistical analysis is performed in MicroStrategy by creating a metric that includes statistical analysis from an R script.

Locating error messages

When the R Integration Pack returns error messages, they are logged to files listed below:

- The `DSSErrors.log` file for your MicroStrategy installation. Contact your MicroStrategy administrator for information on where this log is stored and what information is included.
- The `RScriptErrors.log` file for your R Integration Pack installation. The developer of the R script determines where this log is stored and what information is included, as described in *Implementing error handling, page 26*.

Error messages can also be displayed as a replacement to the numeric results of a metric displayed in a report, document, or dashboard that uses an R script to integrate R statistical analysis into MicroStrategy.

⚠ If a metric displays empty results, this can indicate that an error was encountered. The error can be retrieved from the error logs listed above.

Troubleshooting your installation

The following table lists error messages that are potentially caused by problems with the installation or set up of the R Integration Pack or other supporting files or systems (see *Chapter 2, Installing and Configuring the R Integration Pack*). An administrator who installs and configures the R Integration Pack can review the resolutions listed below to troubleshoot any problems.

Error message	Cause	Resolution
RIntegrationPack required. This error message is displayed as the results of a metric in MicroStrategy.	The R Script functions of the R Integration Pack have not been installed on the computer.	Install the R Script functions on the computer, as described in *Installing the RScript functions, page 10*.
Load of R.DLL failed (error code=n).	The R libraries or dependent libraries are not available or accessible on the Windows environment.	Ensure that you have installed R to meet the requirements of the R Integration Pack, as described in *Installing R, page 9*.
Load of R library failed. An error message from dlerror (), such as: Cannot open libR.so: no such file or directory	The R libraries or dependent libraries are not available or accessible on the UNIX environment.	Ensure that you have installed R to meet the requirements of the R Integration Pack, as described in *Installing R, page 9*.
Initialization of R environment failed.	Your R installation cannot be initialized.	Ensure that you have installed R to meet the requirements of the R Integration Pack, as described in *Installing R, page 9*.
A newer version of this application is already installed on this computer. If you wish to install this version, please uninstall the newer version first. Click OK to exit the wizard. This error message is displayed when attempting to upgrade the R Integration Pack.	If you are upgrading an early version (1.000.002 and earlier) of the R script functions, the installation may fail.	If you encounter this problem while upgrading, you must first uninstall the existing version of the R script functions. Refer to your third-party Microsoft documentation for steps to remove programs from your system. Once the earlier version of the R script functions are removed, you can install the new version using the steps described in *Installing the RScript functions, page 10*.

Troubleshooting the development of R scripts

The following table lists error messages that are potentially caused when creating your R script or processing it with the deployR utility (see *Chapter 4, Developing R Analytics for MicroStrategy*). An R developer who creates R Analytics can review the resolutions listed below to troubleshoot any problems

Error message	Cause	Resolution
One of the defined RVAR variable names exceeds the maximum allowable length (250).	A variable has a name that exceeds 250 characters. This is often caused by making manual changes to the R script header block. This can also be caused by a variable in your R code that exceeds 250 characters for its name.	Use the deployR utility to update the names of your variables so that they meet the 250-character limit. If the names depend on information that is part of the R script code, make modifications to the R script code to reduce the length of variable names.
One of the defined RVAR parameter names exceeds the maximum allowable length (250).	A parameter has a name that exceeds 250 characters. This is often caused by making manual changes to the R script header block. This can also be caused by a parameter in your R code that exceeds 250 characters for its name.	Use the deployR utility to update the names of your parameters so that they meet the 250-character limit. If the names depend on information that is part of the R script code, make modifications to the R script code to reduce the length of names.
One of the defined RVAR options exceeds the maximum allowable length (250).	An option has a name that exceeds 250 characters. This is often caused by making manual changes to the R script header block.	Use the deployR utility to update the names of your options so that they meet the 250-character limit.
Missing 'MICROSTRATEGY_ BEGIN' marker.	The header block of the R script is incorrect. This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script. Once the R script has been updated using deployR, open the R script to ensure that it includes MICROSTRATEGY_BEGIN in the header block.
Missing 'MICROSTRATEGY_ END' marker.	The header block of the R script is incorrect. This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script. Once the R script has been updated using deployR, open the R script to ensure that it includes MICROSTRATEGY_END in the header block.
The variable name '*Variable*' is defined multiple times. Where *Variable* is the name of the variable.	The header block of the R script is incorrect. This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script, which ensures that variables are not defined multiple times.
The name '*Variable*' is	Your R code uses a reserved	Update your R code to ensure that

Error message	Cause	Resolution
reserved and cannot be used as a variable name. Where *Variable* is the name of the variable.	name for a variable.	variables do not use reserved names. Reserved names include: • `if` • `else` • `repeat` • `while` • `function` • `for` • `in` • `next` • `break` • `TRUE` • `FALSE` • `NULL` • `Inf` • `NaN` • `NA` • `NA_integer_` • `NA_real_` • `NA_complex` • `NA_character_and`
The variable name '*Variable*' begins with 'mstr.', which is reserved for internal use only. Where *Variable* is the name of the variable.	Variables beginning with `mstr.` are reserved to pass information between MicroStrategy and the R script. This can occur in your R code or within the R script header block if manual changes were made.	Update your R code to ensure that variables do not use the `mstr.` prefix. Use the deployR utility to process your R script, which ensures that variables do not use this reserved prefix.
The variable name '*Variable*' contains an invalid character. Where *Variable* is the name of the variable.	Your R code uses an invalid character or begins with an invalid character, or manual changes were done to the R script header block.	Update your R code to ensure that variables do not use invalid characters. Invalid characters include: / ; , : | \ { } [ ] + = - ! @ # $ % ^ & * ( ) ~ ? > < ' ` \ " Additionally, you cannot use any of

Error message	Cause	Resolution
		the following characters as the first character of a variable name: `_0123456789` Use the deployR utility to process your R script, which ensures that variables do not use invalid characters in their names.
'-*Option*' is an unsupported parameter data type. Where *Option* is the name of the data type for a variable.	This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script.
'*Parameter*' is not a valid function parameter name. Where *Parameter* is the name of the parameter.	This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script, which ensures that parameters use names of the following formats: • `NumericParam`*N* • `StringParam`*N* • `BooleanParam`*N* Where *N* is a digit from 1 to 9.
The function parameter name '*Parameter*' is defined multiple times. Where *Parameter* is the name of the parameter.	This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script.
Must specify data type (-num or -str) for all output (-o) variables.	This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script and select an appropriate data type for each output variable.
At least one input is required.	This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script. Your changes cannot be saved until you define at least one input.
At least one output is required.	This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script. Your changes cannot be saved until you define at least one output.

Error message	Cause	Resolution
Vector output requires at least one vector input.	All inputs are defined with a scalar data type, but at least one of the outputs uses a vector data type.	Use the deployR utility to process your R script. Your changes cannot be saved until at least one vector input is defined when there are vector outputs.
All scalar inputs and output. Must use RScriptSimple function.	This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script. When all inputs and outputs use the scalar data type, the `RScriptSimple` function is automatically used for the metric expression.
Vector inputs and output. Must use RScriptU or Rscript function.	This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script. A valid function is automatically used for the metric expression.
Vector inputs and scalar output. Must use RScriptAggU or RScriptAgg function.	This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script. A valid function is automatically used for the metric expression.
RScript and RScriptAgg require a vector for the first input.	This is often caused by not using the deployR utility to process the R script or by making manual changes to the R script.	Use the deployR utility to process your R script. A valid metric expression is automatically generated based on your R script.
R script execution error: *RErrorMessage* Where *RErrorMessage* is the error message returned from R.	An error was returned from the R script code.	Review the error message returned and review your R code. You can use the R console provided with your installation of R to perform troubleshooting on your R code.
R script execution error with no error message. Possible causes: execution error outside tryCatch() or syntax error.	The R code includes a syntax error or an execution error occurred that was not caught by tryCatch() error handling.	Review the error message returned and review your R code. You can use the R console provided with your installation of R to perform troubleshooting on your R code. If no errors are returned from the R console, review your R code for sections that are not included within a tryCatch().
Actual data type *RDataType* is not compatible with the expected data type *DataType* for output '*Output*'. Where: • *RDataType* is the data type retrieved from R.	The data type defined in the R code and the data type defined by using deployR are not compatible.	Determine the correct data type for the output: • If the incorrect data type is used in your R code, update your R code to use the correct data type for the output. • If the incorrect data type is used

Error message	Cause	Resolution
• *DataType* is the data type defined in the header block of the R script. • *Output* is the name of the output variable.		in the header block of your R script, use deployR to select the correct data type for the output.

Troubleshooting R integration in MicroStrategy

The following table lists error messages that are potentially caused when performing R statistical analysis in MicroStrategy by creating a metric that includes statistical analysis from an R script (see *Chapter 3, Performing statistical analysis*). A MicroStrategy user who creates metrics, reports, documents, and dashboards that include R statistical analysis can review the resolutions listed below to troubleshoot any problems

Error message	Cause	Resolution
Valid script repository must be specified if using relative R script filename.	The `_RScriptFile` parameter of the metric expression includes only an R script name and the R script repository for the computer could not be found.	Ensure that the R script is stored in the R script repository for the computer.
Missing R script file name.	The `_RScriptFile` parameter is empty.	Define the `_RScriptFile` parameter in the metric expression. You can retrieve the original metric expression from the R script, as described in *Retrieving the metric expression, page 16*. If the `_RScriptFile` parameter is also empty in the R script, contact the developer of the R script to determine its location.
R script file *'Script'* not found. Where *'Script'* is the name of the R script.	The R script could not be found in the specified location.	Ensure that the R script is stored in the location specified in the `_RScriptFile` parameter (see *Making the R script available, page 16*).
Error opening R script file *'Script'*. Where *'Script'* is the name of the R script.	The permissions for accessing the R script prevent opening the R script.	Ensure that you have the required permissions to access the folder where the R script is stored and the R script itself.
The output *Output* specified in the `_OutputVar` parameter is not defined.	The `_OutputVar` parameter has been modified when creating a metric.	Ensure that the `_OutputVar` parameter for the metric is defined with a valid value.

Error message	Cause	Resolution
Where *Output* is the name of the output variable.		
The actual number of inputs (*InputCount*) does not equal to the number of inputs specified within the R script (*InputRVARCount*). Where: • *InputCount* is the number of inputs used when executing the R script. • *InputRVARCount* is the number of inputs defined in the header block of the R script.	The `_CheckInputCount` parameter is defined as True and the number of metrics included in the metric expression is not valid for the inputs required for the R script.	Ensure that you have provided metrics for each input included in the metric expression. You can retrieve the original metric expression from the R script, as described in *Retrieving the metric expression, page 16*. If the correct number of metrics appears to be included but you continue to receive this error, contact the developer of the R script to determine the expected number of metrics that should be included in your metric expression.
The actual number of inputs (*InputCount*) is less than the number of inputs specified within the R script (*InputRVARCount*). Where: • *InputCount* is the number of inputs used when executing the R script. • *InputRVARCount* is the number of inputs defined in the header block of the R script.	The `_CheckInputCount` parameter is defined as True and the number of metrics included in the metric expression is less than the inputs required for the R script.	Ensure that you have provided metrics for each input included in the metric expression. You can retrieve the original metric expression from the R script, as described in *Retrieving the metric expression, page 16*.
The actual number of inputs (*InputCount*) is not consistent with the number of repeated inputs specified within the R script (*InputRVARCount*). Where: • *InputCount* is the number of inputs used when executing the R script. • *InputRVARCount* is the number of inputs defined in the header block of the R script.	The `_CheckInputCount` parameter is defined as True, at least one input variable can accept multiple inputs, and the number of metrics included in the metric expression does not match with the allowable number of inputs that can be passed back to R.	Ensure that you have provided metrics for each input included in the metric expression. Inputs for R can support multiple metrics as their input. Contact the developer of the R script to determine the expected number of metrics that should be included in your metric expression.
Actual data type *DataType* for input variable '*Input*' is not supported. Where:	The data type for the metric mapped to an input variable is not compatible with the input variables data type defined in R.	Ensure that you have provided the correct metric for the input variable specified in the error message and that the metric's data type is correct. If no changes

Error message	Cause	Resolution
• *DataType* is the data type for the metric mapped to an input variable. • *Input* is the name of the input variable. • *RDataType* is the data type retrieved from R.		to the metric are needed, contact the developer of the R script to determine if the input variable is using the correct data type.
Actual data type *DataType* for input variable *'Input'* is not compatible with the expected data type *RDataType*. Where: • *DataType* is the data type for the metric mapped to an input variable. • *Input* is the name of the input variable. • *RDataType* is the data type retrieved from R.	The data type for the metric uses a data type that is not supported for passing information back to R.	Modify the data type for the metric that is mapped to the input variable specified. Supported metric data types include: • Short • Integer • Float • Double • String • UTF8 string
The input variable *'Input'* contained nulls which were skipped due to null-processing settings. Failing report to avoid potential pairing-up problems.	The `_NullsAllowed` parameter for the metric is defined as False, and multiple metrics were mapped to input variables that contained null values.	You can ignore this warning message and allow null values to be passed back to R by defining the `_NullsAllowed` parameter for the metric as True.
Invalid `_InputNames` value: Nested brackets not allowed.	Nested brackets (bracket characters are `[ ]`) are included in the `_InputNames` parameter value, which is not a valid syntax. This can occur if you updated the metric's `_InputNames` parameter.	A single set of brackets is required if an input name includes a space or other special character. Update the metric expression for a metric to ensure that all nested brackets are removed. For example: • Incorrect syntax: `[_InputNames]="Input1, [[Input 2]], Input3"` • Correct syntax: `[_InputNames]="Input1, [Input 2], Input3"`
Invalid `_InputNames` value: Missing comma separator before opening bracket.	A comma is missing between the input variables included in the `_InputNames` parameter value. This can occur if you updated the metric's `_InputNames` parameter.	Update the metric expression for a metric to ensure that a single comma is used to separate each input variable. For example: • Incorrect syntax: `[_InputNames]="Input1 [Input 2], Input3"` • Correct syntax: `[_InputNames]="Input1,`

Error message	Cause	Resolution
		`[Input 2], Input3"`
Invalid `_InputNames` value: Missing opening bracket.	A closing bracket (`]`) does not have a matching opening bracket (`[`) within the `_InputNames` parameter value. This can occur if you updated the metric's `_InputNames` parameter.	Update the metric expression for a metric to ensure that matching opening and closing brackets are included. For example: • Incorrect syntax: `[_InputNames]="Input1, Input 2], Input3"` • Correct syntax: `[_InputNames]="Input1, [Input 2], Input3"`
Invalid `_InputNames` value: Zero-length names not allowed.	An input variable is missing from the `_InputNames` parameter value or an extra comma was included. This can occur if you updated the metric's `_InputNames` parameter.	Update the metric expression for a metric to ensure that each input variable is included and multiple commas are not included sequentially. For example: • Incorrect syntax: `[_InputNames]="Input1,, Input3"` • Correct syntax: `[_InputNames]="Input1, Input2, Input3"`
Invalid `_InputNames` value: Missing closing bracket.	An opening bracket (`[`) does not have a matching closing bracket (`]`) within the `_InputNames` parameter value. This can occur if you updated the metric's `_InputNames` parameter.	Update the metric expression for a metric to ensure that matching opening and closing brackets are included. For example: • Incorrect syntax: `[_InputNames]="Input1, [Input 2, Input3"` • Correct syntax: `[_InputNames]="Input1, [Input 2], Input3"`
Invalid `_InputNames` value: Number of names does not match the actual count.	The number of input variables included in the `_InputNames` parameter value is different than the R script. This can occur if you updated the metric's `_InputNames` parameter.	Ensure that the number of input variables specified is consistent with the specified R script. You can review the metric expression in the R script to confirm the number of input variables before any changes were done for the metric (see *Retrieving the metric expression, page 16*).

INDEX