Data Science Workflows Using R and Spark

E. James (Jim) Harner

Department of Statistics
Department of Management Information Systems
West Virginia University

January 23, 2019

Outline

The Components of this Tutorial

This tutorial covers data science workflows using R as both an analysis and graphics engine and as an interface to databases, Hadoop, Spark, etc.

The following are the required components of this tutorial:

- the The MIST 493A rspark-tutorial-overview slides (these slides)
- the rspark-tutorial notebook content
- the rspark development environment
- the rspark-docker images
- the rspark-vagrant scripts

These components along with supporting technologies should be installed on your laptop prior to the tutorial, if possible. Directions for doing this are given in the next group of slides.

The R Computational Environment

The primary language used in this tutorial is R, but a working knowledge of bash scripts and SQL is useful. Optionally, you can download and install R if you have not already done so. Go to: https://www.r-project.org

Likewise, you should install the RStudio IDE, which is found here: https://www.rstudio.com

Note: Technically, you do not need to download R and RStudio to your laptop except to execute these slides. Both R and RStudio are installed in one of the `rspark` Docker containers, which is where the tutorial code actually runs.

Version Control Using Git

Most of the software you need will require cloning repositories from GitHub. `git` is preinstalled on macOS, but I recommend updating it. Go to: https://git-scm.com/downloads and download/install the latest release.

If you use Windows, I recommend you install another version.Go to: https://gitforwindows.org and download/install the latest release. `gitforwindows` also provides `bash`, i.e., a command-line shell, which you will need.

macOS is UNIX based and therefore has `bash` built-in. Simply launch Terminal.app from the Utilities directory in the Application directory.

The Tutorial Beamer Slides

The tutorial slides provide an overview of the tutorial content. The GitHub repo for the slides can be found here:
https://github.com/jharner/MISTrspark-tutorial

Go to the site and click on the green button to clone the repo to your computer. Alternatively, clone this repo by issuing the following command from a terminal running `bash`:

```
git clone
https://github.com/jharner/MISTrspark-tutorial.git
```

These slides are not the main source of content for this tutorial
The R markdown notebook documents in the next slide provide the detailed, executable content.

The Tutorial Notebook Content

The interactive, executable content of this tutorial is available in a GitHub repo called `rspark-tutorial`. The repo can be found here: https://github.com/jharner/rspark-tutorial

As before, go to the site and click on the green button to clone the repo to your computer. You can also clone this repo with the following command:

```
git clone
https://github.com/jharner/rspark-tutorial.git
```

The `rspark-tutorial` consists of executable R markdown documents organized into modules containing sections. These tutorial documents are executed within `rspark`.

The rspark Computational Environment

This `rspark-tutorial` local repo can then be imported into the computational environment used in this tutorial, which is called rspark. This is the development environment.

The `rspark` computational environment is available in several GitHub repos depending on whether you want to built the environment from scratch or you you want to download pre-built images. Assuming the latter, go to the rspark-docker repo here: https://github.com/jharner/rspark-docker.

Go to the site and click on the green button to clone the repo to your computer. Alternatively, clone this repo by issuing the following command from a terminal:

```
git clone
https://github.com/jharner/rspark-docker.git.
```

Installing Docker

In order to execute the content in the `rspark-tutorial`, the `rspark` computational environment must be run as a web application. Follow the directions in the rspark-docker to launch `rspark`.

In order to run `rspark`, you need to install Docker. macOS supports virtualization directly. Simply install Docker for Mac.

For Windows you must install Hyper-V first. Then install Docker for Windows.

Note: Hyper-V requires Windows 10 Enterprise or Professional. If you have Windows 10 Education, Home, or Mobile, or earlier versions of Windows, you will not be able to run `rspark`. In this case you will be able to view the tutorial content, but not interact with it.

Running rspark

In order to execute the content in the `rspark-tutorial`, the `rspark` computational environment must be run as a web application. Follow the directions in the rspark-docker to launch `rspark`.

Using the UNIX command `cd`, go the directory (folder) containing `rspark-docker`. Do something like this from the terminal:

```
cd rspark-docker
chmod 755 start.sh
./start.sh
```

The second line makes the script executable if it is not already. The third line executes the script. Running the script will take awhile, particularly the first time. Leave the program running in the terminal—do not use Control-C or quit the application.

Connecting to rspark

Open a browser in Chrome (or Firefox or Safari) and type the url as: `localhost:8787` and sign into RStudio:

Username: rstudio
Password: rstudio

When finished, quit the current R session (red button). Then return to your local terminal and type: Control-C to stop the containers, which will return the prompt.

To start the containers again, assuming you are in the `rspark-docker` directory, execute the following commands again and repeat the login:

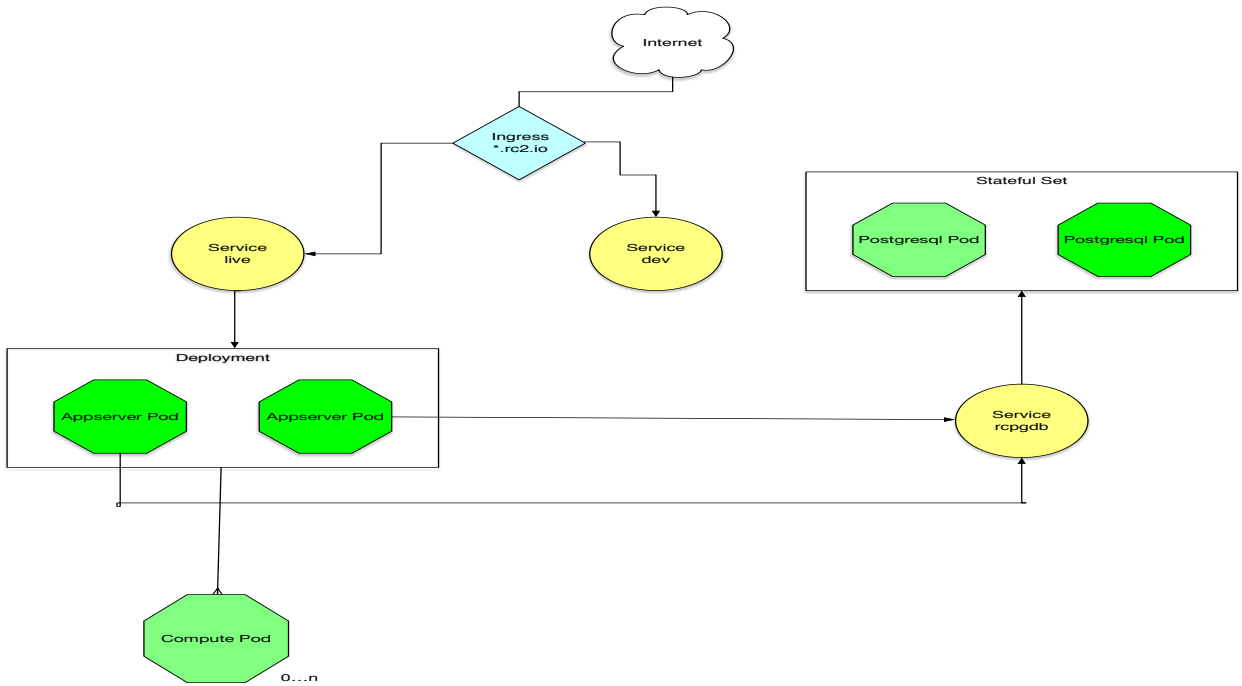
./start.sh

Scaling rspark with Kubernetes

Containers are self-contained execution environments based on Linux. Containers run on the same Linux kernel and thus share resources, which makes them far more efficient than virtual machines. Typically a container image is constructed for each application, but applications that are tightly coupled are often put in the same container.

Kubernetes orchestrates containerized applications, i.e., it is used to create, deploy, and manage containers in operational units called pods. Pods are of three types: deployments (persistent), stateful sets (databases, hdfs, etc.), and jobs (transient).

A possible rspark Kubernetes Cluster



Importing rspark-tutorial into rspark

Once `rspark` is running, import the `rspark-tutorial` by clicking on the `Files` tab in RStudio server. The `rspark-tutorial` directory must be zipped before being imported. Click on the `Upload` option under the `Files` tab and navigate to the `rspark-tutorial.zip` file and upload it.

You can now execute the R markdown documents, i.e., those with the `.Rmd` suffix. These files can be executed interactively as notebooks or knitted to html, pdf, or Word documents.

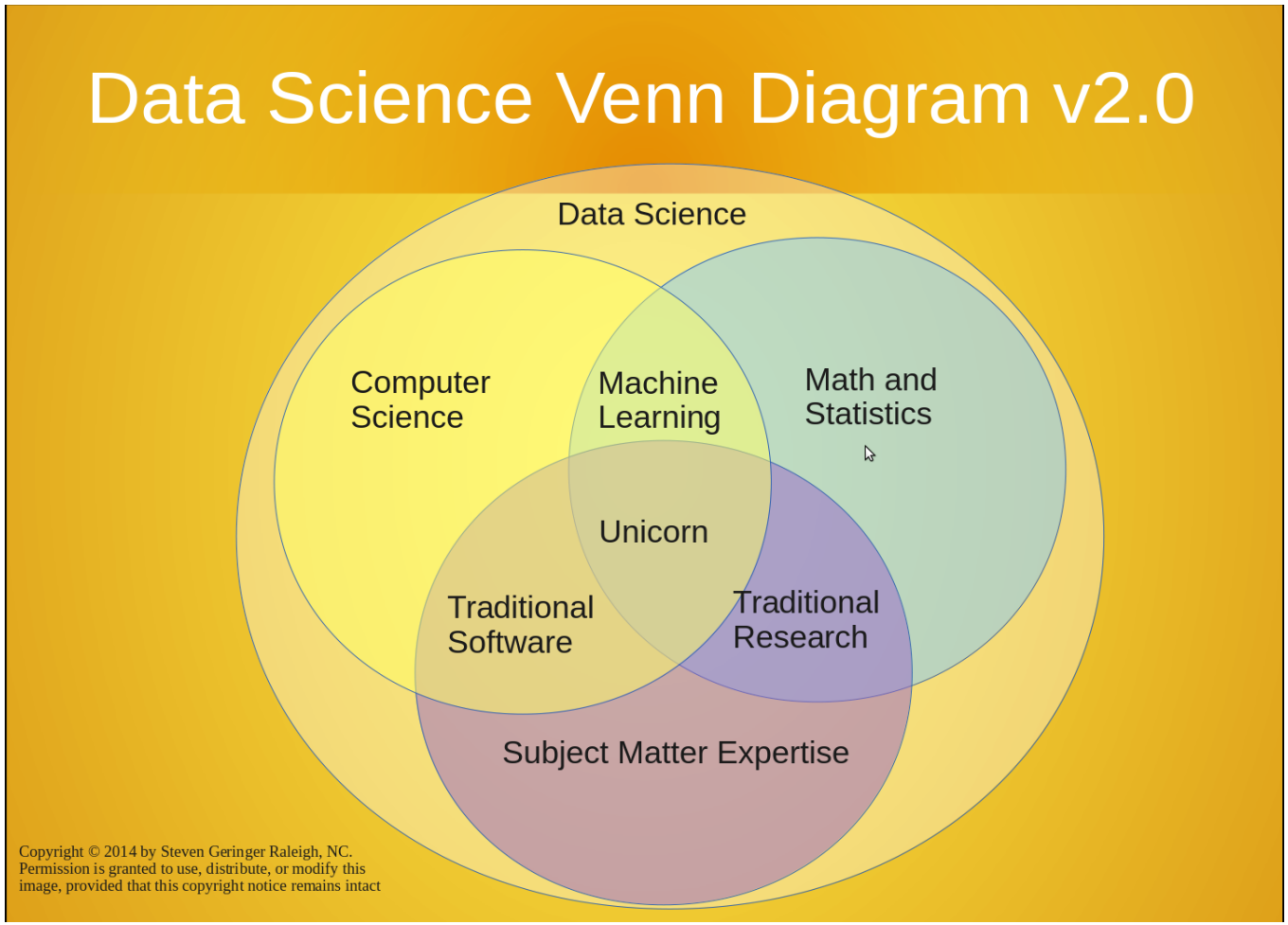
Running the Tutorial Slides

The slides cannot be executed in `rspark`'s `rstudio` docker container since the full publishing capability of RStudio is not supported. In particular the LaTeXBeamer package (and other components of LaTeX) was not installed to keep the container size reasonable.

As a result, the `MISTrspark-tutorial` repo must be executed within a local version of RStudio.

What is Data Science?

Data science combines elements of statistics and computer science to develop methodologies to analyze large, complex data and streaming data within various subject-matter areas.



rspark-tutorial

The slides that follow provide brief summaries of the `rspark-tutorial` content. Each slide provides a summary and a link to the actual section of `rspark-tutorial`. Collectively, these slides make up seven modules: fundamentals, data sources, data transformations, hadoop, spark, supervised learning, and unsupervised learning.

The slides have module names at the top with a small circle for each section in the module. The actual link provides access to the actual content in GitHub. Ideally, the content should be created dynamically within the `rspark` container environment.

Data Science Workflows

This tutorial spans the entire data science workflow or process.

The notes for this section are here.

RStudio Server

RStudio is a powerful integrated development environment (IDE) primarily used for developing code for R projects, including R packages. RStudio supports the integration of R, Python, bash, and SQL code chunks into Rmarkdown documents (and notebooks), among other languages.

The notes for this section are here.

Linux

Data science requires underlying tools and the most basic of these is the operating system (OS). Linux is most commonly used since it is open source and has advanced features, e.g., its kernel and file system, which make handling big data feasible.

The notes for this section are here.

ML Basics

An *algorithm* is a procedure specifying a set of steps to accomplish a task.

Efficient algorithms that work sequentially or in parallel are the basis of pipelines to prepare, process, and analyze data.

The notes for this section are here.

Plain Text

Plain text files are the simplest way to store data. Generally, the data is stored in rows representing observations (or records) with columns representing variables (or fields). The beginning of the file may contain **metadata**, i.e., information about the data. It is sometimes called the **header**, which may represent the variable names.

The notes for this section are here.

JSON

JavaScript Object Notation (JSON) is a text format for the serialization of structured data. The design of JSON is a simple and concise text-based format, particularly when compared to XML.

The notes for this section are here.

Spreadsheets

Spreadsheets are widely used as a storage option. Microsoft Excel is commonly used spreadsheet software. The ‘readxl‘ package is part of the ‘tidyverse‘. It is used for importing tabular Excel files (‘.xls‘ and ‘.xlsx‘) into R as tibbles.

The notes for this section are here.

Databases

This section introduces relational data base management systems and NoSQL databases. The relational model is essential for multi-user transactional data, but it does not scale for big data. NoSQL databases are often distributed across a cluster.

The notes for this section are here.

Web Services

Web Services is a recently introduced phrase. The *Web* and the *HyperText Transfer Protocol (HTTP)* that underlies the communication of data on the Web have become a vital part of our information network and day-to-day environment. Thus, being able to access various forms of data using HTTP is an important facility in a general programming language.

The notes for this section are here.

Data Manipulation

This section explores the main functions in ‘dplyr‘ which Hadley Wickham describes as a *grammar of data manipulation*—the counterpoint to his *grammar of graphics* in `ggplot2`

The `dplyr` package is part of the ‘tidyverse‘. It provides a grammar of data manipulation using a set of verbs for transforming tibbles (or data frames) in R or across various backend data sources.

The notes for this section are here.

dplyr Backends

A powerful feature of 'dplyr' is its ability to operate on various backends, including databases and Spark among others.

`dplyr` allows you to use the same verbs in a remote database as you would in R. It takes care of generating SQL for you so that you can avoid learning it.

The notes for this section are here.

Data Cleaning

In order for `dplyr` to work the data must be `tidy`, i.e., it must be structured as a data frame with certain characteristics.

Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is `messy or tidy` depending on how `rows, columns and tables are matched up with observations, variables and types`.

The notes for this section are here.

Hadoop

Hadoop is an open-source framework for large-scale data storage and distributed computing, built on the MapReduce model.

It is a general framework, applicable to a variety of domains and programming languages. One use case is to drive large R jobs.

The notes for this section are here.

HDFS

When data sets exceed the storage capacity of a single physical machine, we can spread them across a number of machines. Filesystems that manage the storage across a network of machines are called *distributed filesystems*. Since they are network based, the complications of network programming arise, e.g., experiencing node failures without data loss. Thus, distributed filesystems are more complex than regular disk filesystems.

HDFS is Hadoop's main filesystem, but Hadoop has a general-purpose filesystem abstraction. Thus, Hadoop integrates with other storage systems, e.g., the local filesystem and Amazon S3.

The notes for this section are here.

Spark Basics

Spark is a general-purpose cluster computing system, which: has high-level APIs in Java, Scala, Python and R; supports multi-step data pipelines using directed acyclic graphs (DAGs); supports in-memory data sharing across DAGs allowing different jobs to work with the same data.

Spark provides a unified framework to manage big data processing with a variety of data sets that are diverse in nature, e.g., text data, graph data, etc., as well as the source of data (batch vs. real-time streaming data).

The notes for this section are here.

Spark dplyr

`dplyr` is an R package for performing operations on structured data. The data is always a table-like structure, i.e., an R `data.frame` or tibble, a SQL data table, or a Spark DataFrame among others. Ideally, the structure should be in tidy form, i.e., each row is an observation and each column is a variable. Tidy data matches its semantics with how it is stored.

Besides providing functions for manipulating data frames in R, `dplyr` forms an interface for manipulating DataFrames directly in Spark using R. The user can performs operations on Spark DataFrames such as selecting, filtering, and aggregating.

The notes for this section are here.

Spark DataFrame SQL

In the last section, `dplyr` verbs were used to manipulate a Spark DataFrame. However, we often have multiple related Spark tables which we need to combine prior to performing data manipulations. This section shows you how to join two Spark DataFrames using `dplyr`.

The notes for this section are here.

Spark Apply

`sparklyr` provides support to run arbitrary R code at scale within Spark through the function `spark_apply`. Thus, most of R's functionality can be distributed across an R cluster. Apache Spark, even with Spark Packages, has a limited range of functions available.

The notes for this section are here.

Linear Regression

Linear regression models the linear relationship between an outcome variable (dependent or response variable) and one or more explanatory variables (predictors, independent variables, or features). Both the outcome and predictor variables are numeric. *Linearity* is an assumption that should be checked. In some cases it is difficult to assume linearity except locally.

The notes for this section are here.

Regularization Basics

The loss function, i.e., RSS for regression, can be generalized, e.g., by regularization. We now consider fitting all p predictors by constraining, (regularizing) the coefficients, i.e., shrinking the coefficients towards 0. This can often greatly reduce the coefficient variances without appreciably increasing the bias.

The notes for this section are here.

Linear Regression Example

This section illustrates the process of building a concrete compressive strength regression model using Spark.

The notes for this section are here.

Logistic Regression

In many situations the outcome variable is one of k levels or groups. In this section we explore the case in which $k = 2$, i.e., logistic regression.

We illustrate logistic regression modeling using the Wine Quality Data Set from the UCI Machine Learning Repository. We analyze the red variant of the Portuguese "Vinho Verde" wine.

The notes for this section are here.

Regularized Logistic Regression

We revisit the Wine Quality data set using regularization with Spark models.

The notes for this section are here.

Tree-based Models

This section compares 6 classification models, including tree-based models, using Spark.

The notes for this section are here.

Principal Component Analysis

Principal Component Analysis (PCA) is used to determine the structure of a multivariate data set composed of numerical variables. Specifically, the most important purposes of PCA are: to reduce the dimensionality of variable space; to find the linear combinations of the original variables which account for most of the variation in the multivariate system.

The notes for this section are here.

K-means Clustering

The objective of k-means is to group or cluster similar objects together. For example, suppose you have users and you know the age, gender, income, and household size for each user. We then want to segment or cluster the data. You may or may not have an *a priori* notion concerning the number of groups k.

The notes for this section are here.