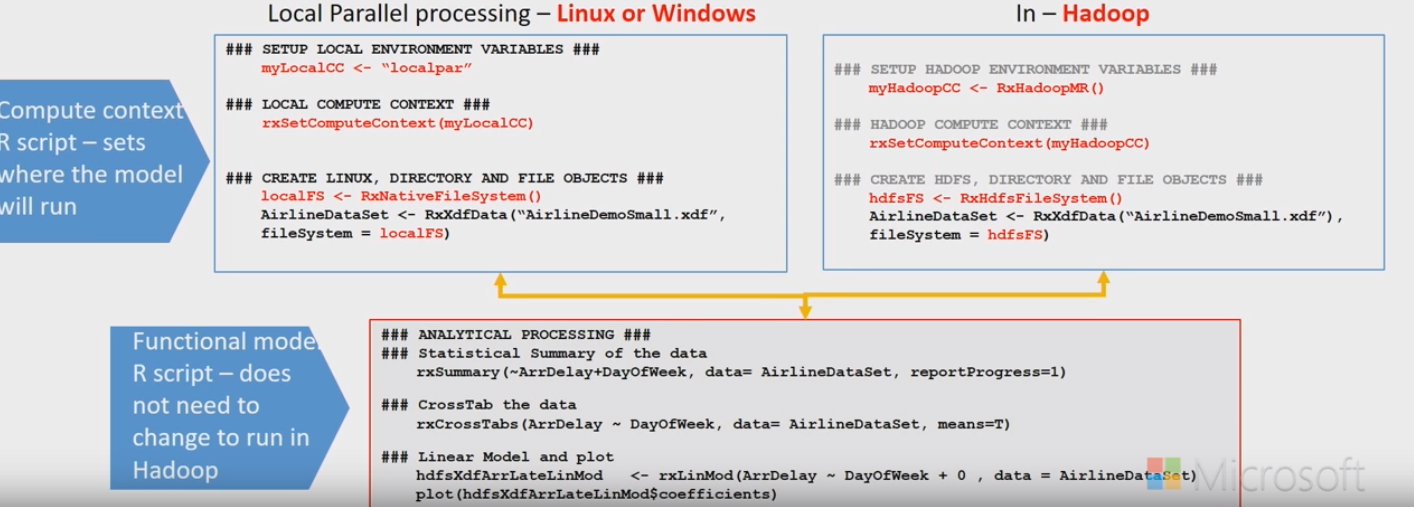
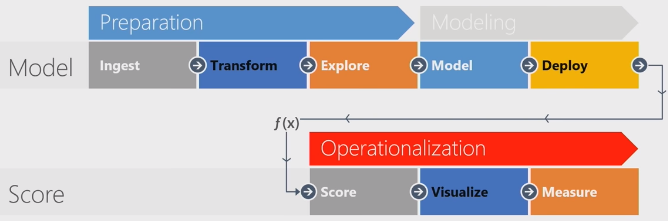
* Traditionally, data in R has to be loaded as a **data frame** = a memory object 🡺 amount of data we can load into R is limited to the amount of memory available on a machine.
* This is a serious challenge for enterprises that deal w/ large data sets.
* Even when enough memory is available to load data, large data sets can quickly run into performance issues (especially true when modeling and analytics is involved)
* Lack of commercial support has also been a challenge for enterprises wanting to adopt R.
* Ability to run R in a PROD environment has been another major hurdle to get R's foot in the door of industry
* **Microsoft R Server** = can have peace of mind that R code is going to run w/ data of ANY size.
* R Server's algorithms are **parallelized** = giving ability to run R code in an efficient + scalable fashion.
* R Server allows deployment of code in a flexible way inside a PROD environments w/ minimal changes made to the overall code structure.
* 2 different flavors.
* Microsoft R Open = community version 🡪 run open-source R code in a more efficient manner using multithreading + changes made to the math kernel library.
* Microsoft R = commercial product 🡪 can write code that runs inside of PROD environments such as Hadoop, Linux, or Teradata + can do in-database analytics using SQL Server R services w/ **RevoScaleR** package (escape R's traditional memory limits for size of data)
* Can also write a code for predictive models + analytics that is scalable b/c *RevoScaleR's algorithms are parallel*.
* R code can distribute to multiple cores inside a single machine or multiple nodes in a cluster, thereby scalability.
* W/ very little changes to overall code structure, can take R code + deploy it inside of PROD environments such as Hadoop, Spark, or SQL server.

**Overview of RevoScaleR**

* **RevoScaleR** stores datasets on the disk/hard drive + loads it only *in chunks at a time* (each chunk is a certain # of rows) for processing.
* Once the chunk is processed, it then loads the next chunk of the data.
* By default, chunk size = 500K rows 🡪 can change to a lower # when dealing w/ *wider* datasets (lots of columns) + a larger # when dealing w/ *longer* data sets (few columns).
* Data in RevoScaleR is *external* (stored on disk) + inherently distributed (process it chunk-wise).
* This means we’re no longer bound by memory when dealing w/ data 🡪 data can be as large as we have space on the hard-disk to store it.
* Since at every point in time we only load 1 chunk of data as a memory object (R list object), we never overexert system’s memory (All happening behind the scene w/ minimal input from user)
* No such thing as a free lunch 🡪 cost to pay when working w/ distributed data:
* Most open-source R algorithms for data processing + analysis (including most 3rd-party packages) rely on the *whole* dataset to be loaded into an R session as a data.frame object = *no longer work directly w/ distributed data.*
* But most data-processing steps (cleaning, creating new/modifying existing cols) can still indirectly (+ relatively easily) be used by RevoScaleR to process distributed data so we can still leverage a great deal of R code
* On the other hand, some data processing steps (merging or sorting data) + most analysis + statistical algorithms (lm()) have RevoScaleR counterparts which mirror the way they work but work on a *distributed data set* in addition to a data frame.
* Ex: RevoScaleR has **rxLinMod()** which replicates lm()
* rxLinMod() is a *distributed* algorithm = it runs *both* on a data frame (far outperforms lm if the data frame in question is large) + on a distributed dataset.
* Using RevoScaleR 🡪 can both leverage *existing* R functionality (offered by R’s rich set of 3rd-party packages) + use what RevoScaleR offers through its own set of distributed functions.
* 1 last advantage = RevoScaleR’s distributed functions offer **code portability**:
* B/c open-source R’s analytics functions are generally not parallel, using these algorithms in an inherently distributed environment can be a challenge.
* Ex: Deploying code to Hadoop means having to rewrite it as **mappers** + **reducers** that Hadoop understands, which can be daunting.
* Inherently parallel data processing + analysis functions in RevoScaleR make them ideal for porting code from **MRS** running on a single machine to **MRS** on a Hadoop cluster or other inherently distributed environments.
* Summary
* When data is large but still small enough to fit in the memory as a data frame, can still use RevoScaleR’s parallel algorithms to run models much faster than open-source counterparts
* When data is too large to fit in available memory, can work directly w/ data on disk (such as flat files) or convert data to an external + distributed format called **XDF**.
* RevoScaleR’s data-processing + analysis functions work w/ such data in addition to a data frame.
* When data is saved in a distributed environment such as HDFS or SQL Server (often the case in PROD), w/ some minor adjustments, can deploy code in such environments, reducing the hurdle of going from DEV to PROD.
* **RevoScaleR** reads data from a variety of sources + can prep data, run descriptive statistics + statistical tests
* Can take a sample from our data + w/ that sample, have a data frame that can be used by all of R's packages for performing various analysis.
* Bread + butter of RevoScaleR = the **analytics algorithms** = regression + classification algorithms such as linear models, logistic regression, decision trees, ensemble models such as random forests, + k-means algorithms for building clusters.
* All these algorithms have counterparts in open source R functions but strength of MSR algorithms = they’re **parallel** = makes it so we can run algorithms on very large data sets in a scalable fashion
* In addition to being parallel + scalable, these algorithms can also run inside of PROD environments
* Ex: What’s happening inside of Hadoop:



* Left = Code to point to some data on a local Linux or Windows machine
* Bottom = Take data + run various statistical summaries on it, run some cross tabulations, + finally perform some kind of modeling task w/ the data.
* Right = Similar to left but instead of pointing to data sitting on a local machine, is pointing to data sitting on **HDFS** (inside of a Hadoop cluster)
* Once you change the code to pointing to data on HDFS, *the rest of the code is the same.*
* This sort of code portability in RevoScaleR is very important in order to be able to run R code inside of PROD environments without having to completely change the structure of the code.
* Typical Analytics Lifecycle:



* Any analytics lifecycle begins w/ data + 1st thing to do = ingest it.
* Once ingested, have to ask what we need to do to prep data for analysis (various transformations + explorations)
* Once data is ready, move to the modeling phase w/ different models being interested in comparing performance of each by deploying models on some **out-of-sample data**
* Once settled on a model, move onto operationalization phase = interested in scoring data sitting in a PROD server or in rebuilding our model on the large dataset sitting on the PROD server
* Either way, at the end of the day 🡪 look at various measurements + somehow visualize the results.
* More often than not, loop back + go back to square one b/c the way data is being used in production can often guide us + inform us on how we can do a better job of ingesting, transforming, + getting the data ready for the analysis, or which models seem to work better + which models don't.
* *This is a continuous lifecycle.*
* 3 main benefits of RevoScaleR package.
* 1) Even w/ enough memory to load data as a data frame into an R session, can use RevoScaleR's parallel algorithms to run analytics on that data much faster than w/ open source counterparts
* 2) If data is too large to fit in available memory, can still use RevoScaleR algorithms just as before by simply pointing to data sitting on disks.
* RevoScaleR operates by loading data into R session as a data frame but only a chunk at a time (default = 500K rows but can be changed)
* By doing so RevoScaleR can simply load data 1 chunk at a time, process it, move on to the next chunk, + keep doing this until all data has been handled.
* The fact that its algorithms are parallel is what makes this possible.
* 3) Can take code + deploy it inside the PROD environment (Hadoop cluster or SQL Server database) w/ very little changes made to the code structure.
* **Microsoft R Client =** lightweight version of MRS (not meant to be use as a PROD environment for the MRS on a single machine
* a separate installation of R (separate from CRAN) that points to the Microsoft R Client installation
* Must point to the R Client in R Studio 🡪 Tools, Options, R Tools Advanced 🡪 point to C:Program Files/MicrosoftRClient/ R\_SERVER (default directory)