# dplyrXdf cheat sheet

Using dplyr with out-of-memory data in Microsoft R Server

## Overview

### dplyr

- Popular package for data wrangling
- Simplifies and streamlines working with data
- Built around concept of data transformation pipelines
- Part of the "Hadleyverse"

### xdf file format

- Format for data storage on disk
- Breaks the memory barrier
- Built to facilitate efficient, scalable, multithreaded/multiple-core processing
- Part of Microsoft R Server, previously known as Revolution R Enterprise

### dplyrXdf

- Extends dplyr framework to large, on-disk data sets
- Simplifies current interface to xdf functionality
- Handles the task of file management for the user
- Transparent to other xdf-aware functions

## Implementation

### Verbs

dplyr verbs are S3 generics, with methods provided for data frames, data tables, and so on. This means dplyr is *extensible*.

- Define methods for Microsoft R Server data source objects.
- All major single- and two-table verbs supported, as well as grouping.
- Leverage existing MRS functions for working with xdf files: for example `mutate` calls `rxDataStep` to create new columns.
- Evaluation is eager, not lazy (that is, behavior is like dplyr with data frames, as opposed to SQL databases).
- dplyrXdf verbs take xdf files as input and create xdf files as output, so they will scale to large data sets.

### File management

Define a new tbl_xdf class:

- Inherits from existing Microsoft R Server classes for representing on-disk data sources.
- Conceptually similar to SQL database tbls/RODBC/DBI objects: an R object that points to out-of-memory data.
- dplyrXdf verbs write to temporary, auto-generated files: you never have to supply/remember file names.
- Only the most recent output is kept: no obsolete files lying around.
- Non-tbl data is never modified: your original data is safe.

### Extensions

- New `persist` verb: save a tbl_xdf's data to a permanent location.
- New `factorise` (`-ze`) verb: convert columns to factors (with optional specified levels).
- New `doXdf` verb, a scalable version of `do`: data for each subset stays on disk, not read into memory as a data frame.
- Carry out multiple actions in a single verb with `.rxArgs` argument: key optimization for large data sets.
- Additional helper functions: `deleteXdfTbls` (clean up temporary data files), `as.data.frame` (method for xdf data sources).



Microsoft

# dplyrXdf examples

```
## in dplyr: data frames ###################################
library(dplyr)

nycflights13::flights %>%
    group_by(carrier) %>%
    summarise(meantime=mean(air_time, na.rm=TRUE),
              sumdist=sum(distance, na.rm=TRUE))


## in dplyrXdf: xdf files ##################################
library(dplyrXdf)

# from http://packages.revolutionanalytics.com/datasets/
flx <- RxXdfData("AirOnTime2012.xdf")
flx %>%
    group_by(UniqueCarrier) %>%
    summarise(meantime=mean(AirTime),
              sumdist=sum(Distance)) %>%
    as.data.frame


# one day from Criteo 1TB dataset:
# https://blogs.technet.microsoft.com/machinelearning/2015/04/01/now-available-
on-azure-ml-criteos-1tb-click-prediction-dataset/
day0 <- RxXdfData("day_0.xdf")

dim(day0)
#[1] 195841983        40    ## 19.5 million rows
file.size(day0@file)
#[1] 16239112599            ## 16 gigabytes

hash <- function(x, nBits)
{
    nMax <- 2^nBits
    nChars <- ceiling(nBits/4)
    factor(strtoi(substr(x, 1, nChars), base=16) %% nMax, levels=0:(nMax-1))
}

day0Hashed <- day0 %>%
    mutate(day0, .rxArgs=list(
        transformFunc=function(data) {
            lapply(data, .hash, nBits=5)
        },
        transformVars=paste0("V", 15:40),
        transformObjects=list(.hash=hash))
    ) %>%
    persist("day0Hashed.xdf")
```

Same syntax as in dplyr pipeline

Data stays on disk until/unless read in with as.data.frame

dplyrXdf, like Microsoft R Server, scales to arbitrary data set sizes

Use the MS R Server transformFunc feature to transform multiple columns

Save data to permanent location with persist verb

```
## with standard Microsoft R Server RevoScaleR functions ########

# from http://packages.revolutionanalytics.com/datasets/
clm <- RxXdfData("claims.xdf")

catVars <- grep("^Cat", names(clm), value=TRUE)

clmRx1 <- rxDataStep(clm, outFile="claimsRx1.xdf",
    rowSelection=Model_Year >= 2000 & Calendar_Year == 2007,
    varsToKeep=catVars,
    transforms=list(has_claim=Claim_Amount > 0))

clmRx2 <- rxFactors(clmRx1, outFile="claimsRx2.xdf",
    factorInfo=catVars)

# formula: has_claim ~ Cat1:Cat2:Cat3:...
clmRxSmryFm <- paste("has_claim ~", paste(catVars, collapse=":"))
clmRxSmry <- rxSummary(clmRxSmry,
    data=clmRx2, means=FALSE, useSparseCube=TRUE,
    summaryStats=c("Sum", "ValidObs"))$categorical[[1]]

file.remove("claimsRx1.xdf", "claimsRx2.xdf")
rm(clmRx1, clmRx2)


## with dplyrXdf ###################################################
clmSmry <- clm %>%
    filter(Model_Year >= 2000, Calendar_Year == 2007) %>%
    select(starts_with("Cat")) %>%
    mutate(has_claim=Claim_Amount > 0) %>%
    group_by_(.dots=catVars) %>%
    summarise(claims=sum(has_claim), policies=n())


## with dplyrXdf, using .rxArgs ####################################
clmSmry2 <- clm %>%
    filter(Model_Year >= 2000, Calendar_Year == 2007,
        .rxArgs=list(
            varsToKeep=catVars),
            transforms=list(has_claim=Claim_Amount > 0)
        )
    ) %>%
    group_by_(.dots=catVars) %>%
    summarise(claims=sum(has_claim), policies=n()) %>%
    persist("clmSmry2.xdf")
```

MRS functions are scalable, fast, and flexible, but interface can be hard to master

Manual cleanup when done

Verbs output to auto-generated files, clean up after themselves

Verbs will not overwrite non-tbl data sources; this pipeline uses the same xdf file as the previous one

Reduce I/O by packing multiple actions into one step; key consideration for big data

```
## do and doXdf: parallel processing of subsets of data ######

# from http://packages.revolutionanalytics.com/datasets/
cens <- RxXdfData("Census5PCT2000.xdf")

rxSetOption(NumCoresToUse=parallel::detectCores() - 1)
rxSetComputeContext("localpar")


# fit linear models using MS R Server rxLinMod function
censLinMods <- cens %>%
    select(statefip, hhincome, age, sex, educ99, hhwt) %>%
    group_by(statefip) %>%
    doXdf(model=rxLinMod(log(hhincome) ~ age + sex + educ99,
                         data=., fweights=hhwt))


# fit trees using rpart (could also use MRS rxDTree function)
library(rpart)

censTreeMods <- cens %>%
    select(statefip, hhincome, age, sex, educ99, hhwt) %>%
    group_by(statefip) %>%
    do(model=rpart(log(hhincome) ~ age + sex + educ99,
                   data=., weights=hhwt, cp=0.001))

# get predictions
censTreeMods_AZ <- filter(censTreeMods, statefip == "AZ")$model
censTreePreds_AZ <- cens %>%
    filter(state == "AZ") %>%
    transmute(cens,
        pred=predict(model, data.frame(age, sex, educ99)),
        .rxArgs=list(
            transformObjects=list(model=censTreeMods_AZ),
            transformPackages="rpart"
        )
    ) %>%
    persist("censTreePreds_AZ.xdf")
```

Like MRS, dplyrXdf can take advantage of multiple cores

Scalable version of do: data for each group stays on disk, rather than read into memory

For working with open-source R code, do is also available

Scoring open-source models in a scalable manner: predict inside a dplyrXdf mutate/ transmute

Package available for download at https://github.com/RevolutionAnalytics/dplyrXdf
Requires Microsoft R Server or Client

Microsoft