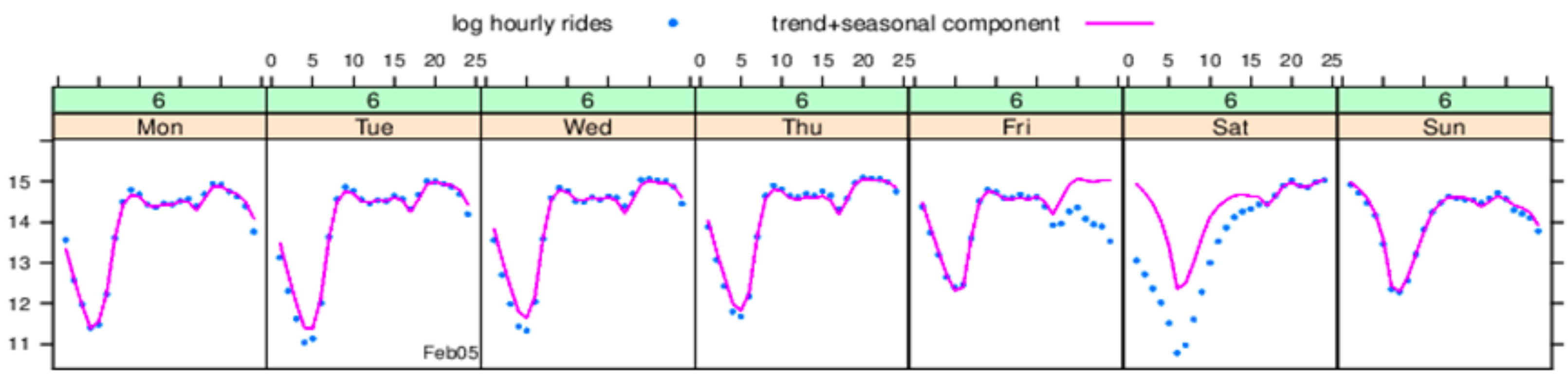
Tessera: Open Source Tools for Big Data Analysis in R

Ryan P. Hafen, Stephen F. Elston, Amanda M. White

useR! Conference
June 30, 2015

Deep Analysis of Large, Complex Data

- Goals of analysis:
  - Uncover interesting or previously unknown behavior
  - Develop new insights
  - Identify deviations from expected behavior
  - Confirm or reject hypotheses or suspicions
- Visualization is critical in this process



Deep Analysis of Large, Complex Data

- Data most often do not come with a manual for what to do
- If we already (think we) know the algorithm / model to apply and simply apply it to the data, we are not doing analysis, we are processing
- Deep analysis means
  - detailed, comprehensive analysis that does not lose important information in the data
  - learning from the data, not forcing our preconceptions on the data
  - being willing and able to use any of the 1000s of statistical, machine learning, and visualization methods as dictated by the data
  - trial and error, an iterative process of hypothesizing, fitting, validating, learning

Deep Analysis of Large, Complex Data

Any or all of the following:

- Large number of records
- Many variables
- Complex data structures not readily put into tabular form of cases by variables
- Intricate patterns and dependencies that require complex models and methods of analysis
- Does not conform to simple assumptions made by many algorithms

Tessera

Our goal in creating Tessera was to

- Enable users to
  - visually explore large datasets, and
  - fit statistical models
  - **with minimal lines of code**
- While providing
  - a familiar, interactive, desktop programming environment (R)
  - scalable access to the thousands of analytic methods of statistics, machine learning and visualization available in R
  - automatic management of the complicated tasks of distributed storage and computation required for big data

Tessera Fundamentals

- Users interact primarily with two R packages:
    - **datadr**: data analysis toolkit implementing the Divide & Recombine paradigm that allows data scientists to leverage parallel processing back-ends such as Hadoop and Spark
    - **Trelliscope**: visualization package that enables detailed but scalable visualization of large, complex data
- Open source
    - http://tessera.io
    - http://github.com/tesseradata

Tessera Fundamentals

Flexibility

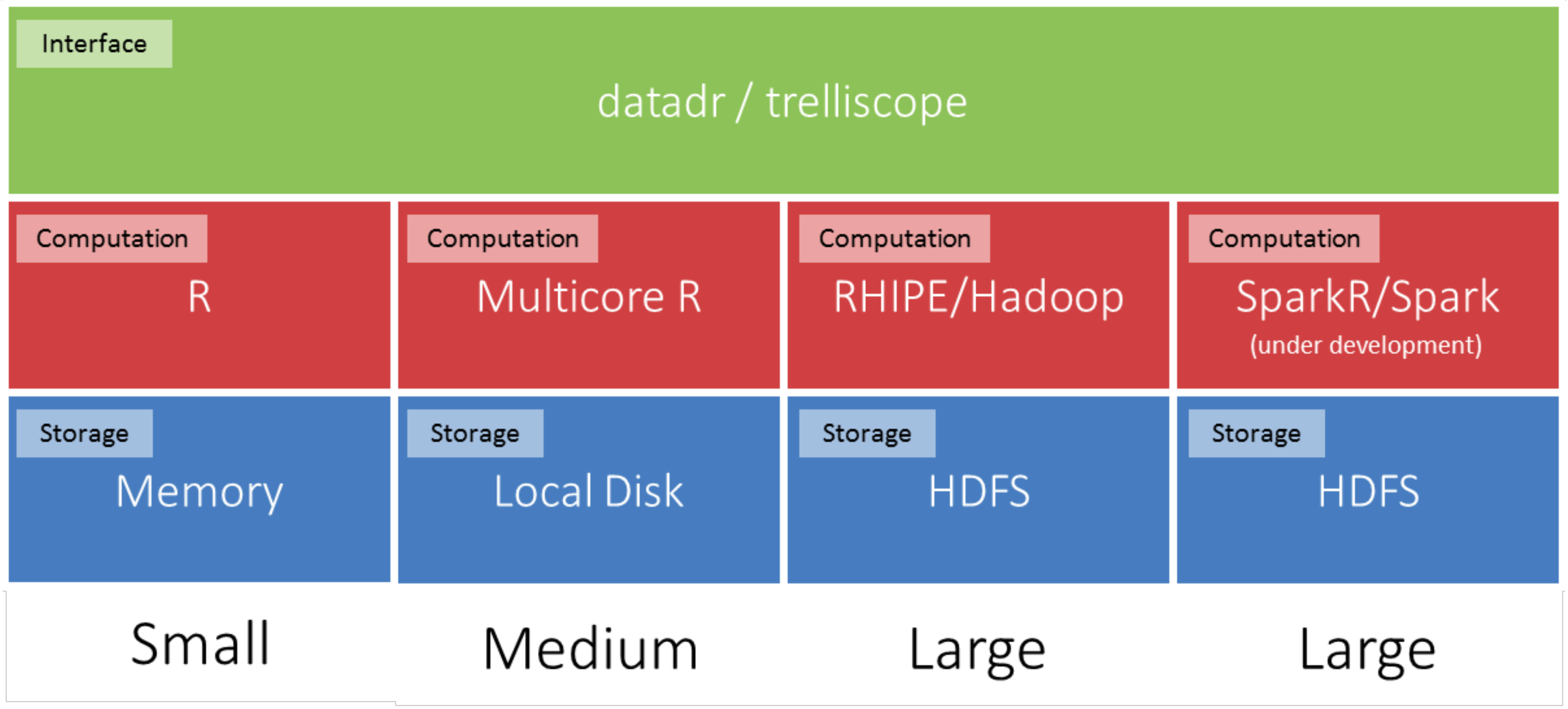
+

- Rapid development of code and models
- Excellent flexible statistical visualization capabilities
- Immense collection of statistical routines

Scalability

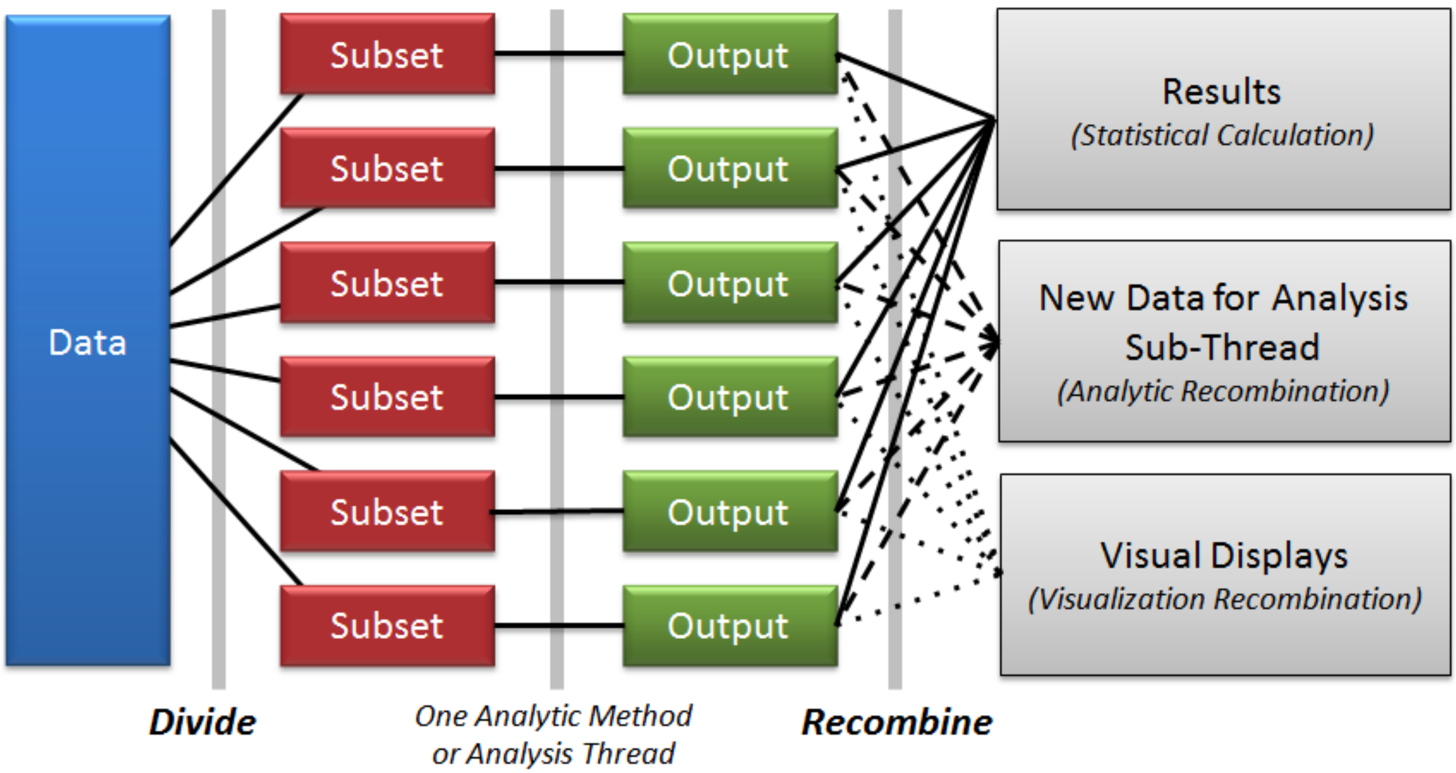
- Hides messy details of parallelization
- Takes care of partitioning, scheduling, fault tolerance, data management, and execution
- Parallel programming paradigm (MapReduce) makes sense for many statistical algorithms

Back End Agnostic



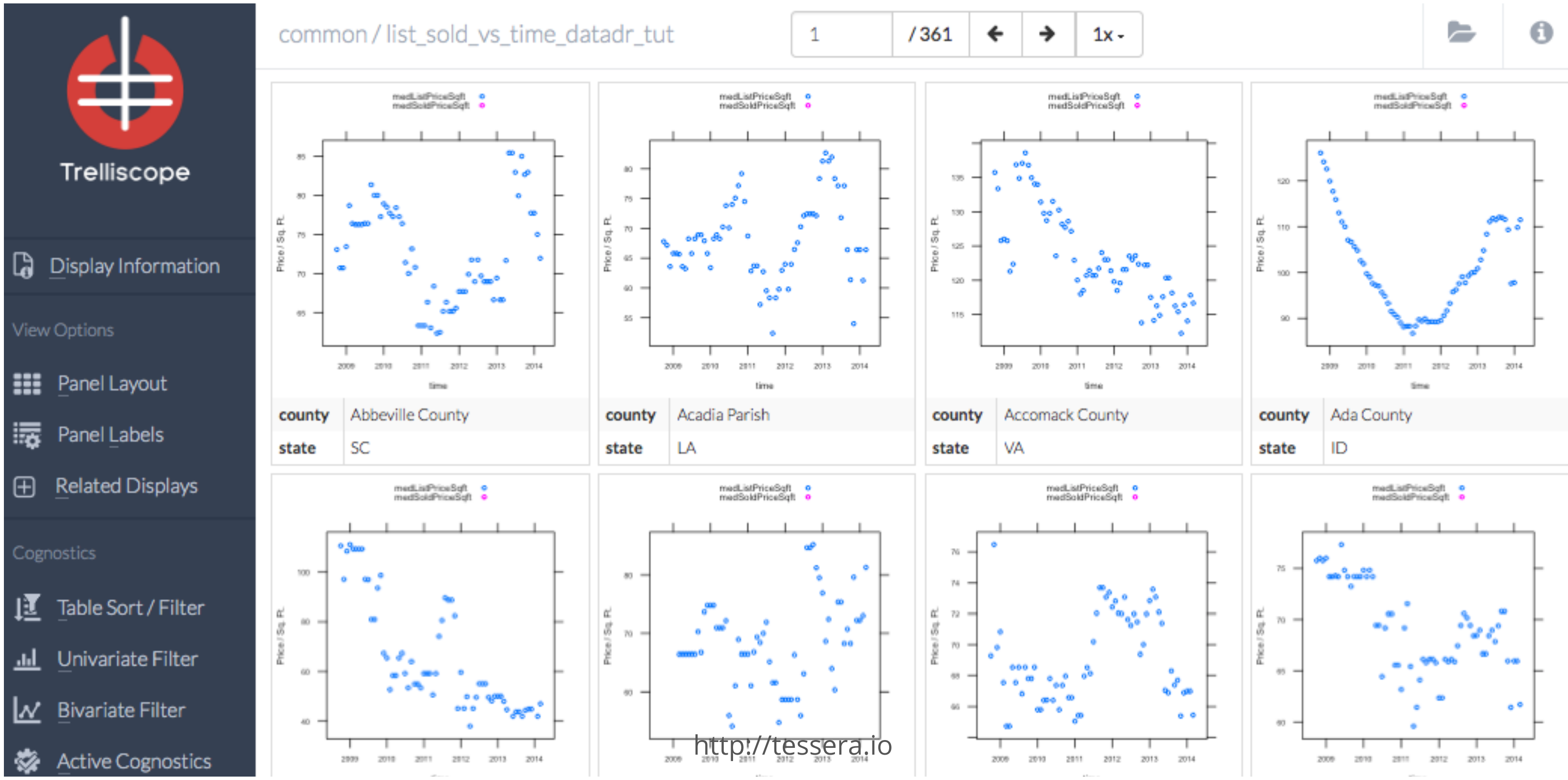
Interface stays the same regardless of back end

Tessera Fundamentals: datadr

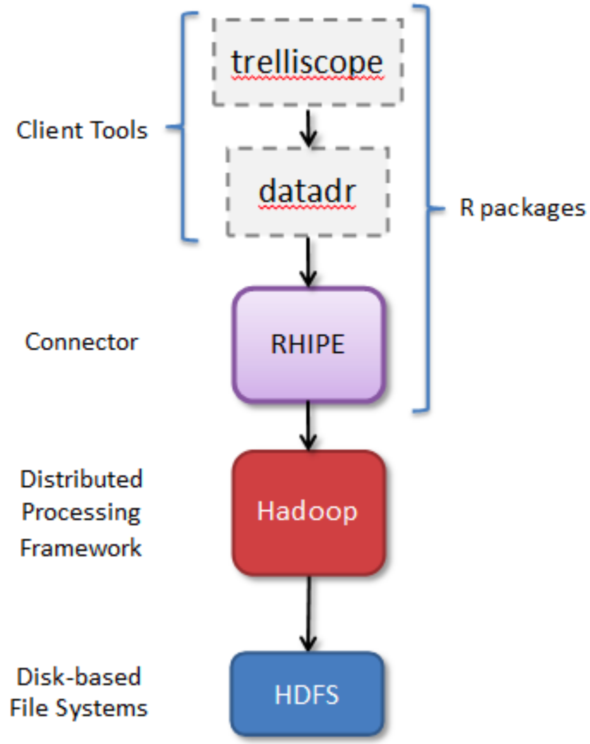


Tessera Fundamentals: Trelliscope

- Trelliscope: a viz tool that enables scalable, detailed visualization of large data
- Data is split into meaningful subsets, and a visualization method is applied to each subset
- The user can sort and filter plots based on "cognostics"-summary statistics of interest-to explore the data



The current Tessera distributed computing stack



- trelliscope: visualization of subsets of data, web interface powered by Shiny http://shiny.rstudio.com
- datadr: interface for divide and recombine operations
- RHIPE: The R and Hadoop Integrated Programming Environment
- Hadoop: Framework for managing data and computation distributed across multiple hardrives in a cluster
- HDFS: Hadoop Distributed File System

Introduction to `datadr`

Installing the Tessera packages

```
install.packages("devtools") # if not installed
library(devtools)
install_github("tesseradata/datadr")
install_github("tesseradata/trelliscope")
install_github("hafen/housingData") # demo data
```

Housing Data

- Housing sales and listing data in the United States
- Between 2008-10-01 and 2014-03-01
- Aggregated to the county level
- Zillow.com data provided by Quandl (https://www.quandl.com/c/housing)

Housing Data Variables

Variable	Description
fips	Federal Information Processing Standard a 5 digit count code
county	US county name
state	US state name
time	date (the data is aggregated monthly)
nSold	number sold this month
medListPriceSqft	median list price per square foot
medSoldPriceSqft	median sold price per square foot

`datadr` data representation

- Two main data types are
  - Distributed data frame (`ddf`):
    - A data frame that is split into chunks
    - Each chunk contains a subset of the rows of the data frame
    - Each subset may be distributed across the nodes of a cluster
  - Distributed data object (`ddo`):
    - Similar to distributed data frame
    - Except each chunk can be an object with any structure
    - Every distributed data frame is also a distributed data object
- Both `ddf` and `ddo` types use key/value pairs for their structure

Data storage

`datadr` data storage options:

- In memory
- Local disk
- HDFS
- Spark (coming soon)

Data ingest

```
# similar to read.table function:
my.data <- drRead.table(
    hdfsConn("/home/me/dir/datafile.txt",
        header=TRUE, sep="\t")
)

# similar to read.csv function:
my.data2 <- drRead.csv(
    localDiskConn("c:/my/local/data.csv"))

#convert in memory data.frame to ddf:
my.data3 <- ddf(some.data.frame)
```

You try it

```r
# Load necessary libraries
library(datadr)
library(trelliscope)
library(housingData)

# housing data frame is in the housingData package
housingDdf <- ddf(housing)
```

Division

- A common thing to do is to divide a dataset based on the value of one or more variables
- Another option is to divide data into random replicates
  - Use random replicates to estimate a GLM fit by applying GLM to each replicate subset and taking the mean coefficients
  - Random replicates can also be used for a bag of little bootstraps approach

Divide example

Divide the housing data set by the variables "county" and "state"

(This kind of data division is very similar to the functionality provided by the plyr package)

```
byCounty <- divide(housingDdf,
    by = c("county", "state"), update = TRUE)
```

Divide example

```
byCounty
```

```
##
## Distributed data frame backed by 'kvMemory' connection
##
##  attribute      | value
## ----------------+-----------------------------------------------------------
##  names          | fips(cha), time(Dat), nSold(num), and 2 more
##  nrow           | 224369
##  size (stored)  | 15.73 MB
##  size (object)  | 15.73 MB
##  # subsets      | 2883
##
## * Other attributes: getKeys(), splitSizeDistn(), splitRowDistn(), summary()
## * Conditioning variables: county, state
```

Exercise: try the `divide` function

Now try using the divide statement to divide on one or more variables

Possible solutions

```
byState <- divide(housing, by="state", update = TRUE)

byMonth <- divide(housing, by="time", update=TRUE)
```

Exploring the `ddf` data object

Data divisions can be accessed by index or by key name

```
byCounty[[1]]
```

```
## $key
## [1] "county=Abbeville County|state=SC"
##
## $value
##    fips       time nSold medListPriceSqft medSoldPriceSqft
## 1 45001 2008-10-01    NA         73.06226               NA
## 2 45001 2008-11-01    NA         70.71429               NA
## 3 45001 2008-12-01    NA         70.71429               NA
## 4 45001 2009-01-01    NA         73.43750               NA
## 5 45001 2009-02-01    NA         78.69565               NA
## ...
```

```
byCounty[["county=Benton County|state=WA"]]
```

Exploring the `ddf` data object

Partipants: try these functions on your own

- `summary(byCounty)`
- `names(byCounty)`
- `length(byCounty)`
- `getKeys(byCounty)`

Transformations

- The `addTransform` function applies a function to each key/value pair in a `ddf`
  - E.g. to calculate a summary statistic
- The transformation is not applied immediately, it is deferred until:
  - A function that kicks off a map/reduce job is called (e.g. `recombine`)
  - A subset of the data is requested (e.g. `byCounty[[1]]`)
  - `drPersist` function explicitly forces transformation computation

Transformation example

```r
# Function to calculate a linear model and extract
# the slope parameter
lmCoef <- function(x) {
  coef(lm(medListPriceSqft ~ time, data = x))[2]
}

# Best practice tip: test transformation
# function on one division
lmCoef(byCounty[[1]]$value)
```

```
##          time
## -0.0002323686
```

```r
# Apply the transform function to the ddf
byCountySlope <- addTransform(byCounty, lmCoef)
```

Transformation example

```
byCountySlope[[1]]
```

```
## $key
## [1] "county=Abbeville County|state=SC"
##
## $value
##          time
## -0.0002323686
```

Exercise: create a transformation function

- Try creating your own transformation function
- Hint: the input to your function will be one value from a key/value pair (e.g. `byCounty[[1]]$value`)

```
transformFn <- function(x) {
    ## you fill in here
}

# test:
transformFn(byCounty[[1]]$value)

# apply:
xformedData <- addTransform(byCounty, transformFn)
```

Possible solutions

```r
# example 1
totalSold <- function(x) {
   sum(x$nSold, na.rm=TRUE)
}
byCountySold <- addTransform(byCounty, totalSold)

# example 2
timeRange <- function(x) {
   range(x$time)
}
byCountyTime <- addTransform(byCounty, timeRange)
```

Recombination

- Combine transformation results together again
- Example

```
countySlopes <- recombine(byCountySlope,
    combine=combRbind)
```

```
head(countySlopes)
```

```
##                  county state           val
## time  Abbeville County    SC -0.0002323686
## time1    Acadia Parish    LA  0.0019518441
## time2  Accomack County    VA -0.0092717711
## time3       Ada County    ID -0.0030197554
## time4     Adair County    IA -0.0308381951
## time5     Adair County    KY  0.0034399585
```

Recombination options

`combine` parameter controls the form of the result

- `combine=combRbind`: `rbind` is used to combine results into `data.frame`, this is the most frequently used option
- `combine=combCollect`: results are collected into a list
- `combine=combDdo`: results are combined into a `ddo` object

Exercise: try the `recombine` function

- Apply `recombine` to the data with your custom transformation
- Hint: `combine=combRbind` is probably the simplest option

Exercise: `divide`

Divide two new datasets `geoCounty` and `wikiCounty` by county and state

```
# look at the data first
head(geoCounty)
head(wikiCounty)

# use divide function on each
```

Solution

```
geoByCounty <- divide(geoCounty,
    by=c("county", "state"))

wikiByCounty <- divide(wikiCounty,
    by=c("county", "state"))
```

Data operations: drJoin

Join together multiple data objects based on key

```
joinedData <- drJoin(housing=byCounty,
    slope=byCountySlope,
    geo=geoByCounty,
    wiki=wikiByCounty)
```

Distributed data objects vs distributed data frames

- In a `ddf` the value in each key/value is always a `data.frame`
- A `ddo` can accomodate values that are not `data.frame`s

```
class(joinedData)
```

```
## [1] "ddo"      "kvMemory"
```

Distributed data objects vs distributed data frames

```r
joinedData[[176]]
```

```
## $key
## [1] "county=Benton County|state=WA"
##
## $value
## $housing
##     fips       time nSold medListPriceSqft medSoldPriceSqft
## 1  53005 2008-10-01   137         106.6351         106.2179
## 2  53005 2008-11-01    80         106.9650               NA
## 3  53005 2008-11-01    NA               NA         105.2370
## 4  53005 2008-12-01    95         107.6642         105.6311
## 5  53005 2009-01-01    73         107.6868         105.8892
## 6  53005 2009-02-01    97         108.3566               NA
## 7  53005 2009-02-01    NA               NA         104.3273
## 8  53005 2009-03-01   125         107.1968         103.2748
## 9  53005 2009-04-01   147         107.7649         102.2363
## 10 53005 2009-05-01   192         108.6823               NA
```

Data operations: drFilter

Filter a `ddf` or `ddo` based on key and/or value

```
# Note that a few county/state combinations do
# not have housing sales data:
names(joinedData[[2884]]$value)
```

```
## [1] "geo"  "wiki"
```

```
# We want to filter those out those
joinedData <- drFilter(joinedData,
    function(v) {
        !is.null(v$housing)
    })
```

Other data operations

- `drSample`: returns a `ddo` containing a random sample (i.e. a specified fraction) of key/value pairs
- `drSubset`: applies a subsetting function to the rows of a `ddf`
- `drLapply`: applies a function to each subset and returns the results in a `ddo`

Exercise: `datadr` data operations

Apply one or more of these data operations to `joinedData` or a `ddo` or `ddf` you created

- `drJoin`
- `drFilter`
- `drSample`
- `drSubset`
- `drLapply`

Using Tessera with a Hadoop cluster

Differences from in memory computation:

- Data ingest: use `hdfsConn` to specify a file location to read in HDFS
- Each data object is stored in HDFS
  - Use `output` parameter in most functions to specify a location in HDFS to store data

```
housing <- drRead.csv(
    file=hdfsConn("/hdfs/data/location"),
    output=hdfsConn("/hdfs/data/second/location"))

byCounty <- divide(housing, by=c("state", "county"),
    output=hdfsConn("/hdfs/data/byCounty"))
```

Hadoop demo

Introduction to `trelliscope`

Trelliscope

- Divide and recombine visualization tool
- Based on Trellis display
- Apply a visualization method to each subset of a `ddf` or `ddo`
- Interactively sort and filter plots

Trelliscope panel function

- Define a function to apply to each subset that creates a plot
- Plots can be created using base R graphics, ggplot, lattice, rbokeh, conceptually any htmlwidget

```
# Plot medListPriceSqft and medSoldPriceSqft by time
timePanel <- function(x) {
   xyplot(medListPriceSqft + medSoldPriceSqft ~ time,
      data = x$housing, auto.key = TRUE,
      ylab = "Price / Sq. Ft.")
}
```

Trelliscope panel function

```
# test the panel function on one division
timePanel(joinedData[[176]]$value)
```

Visualization database (vdb)

- Trelliscope creates a directory with all the data to render the plots
- Can later re-launch the Trelliscope display without all the prior data analysis

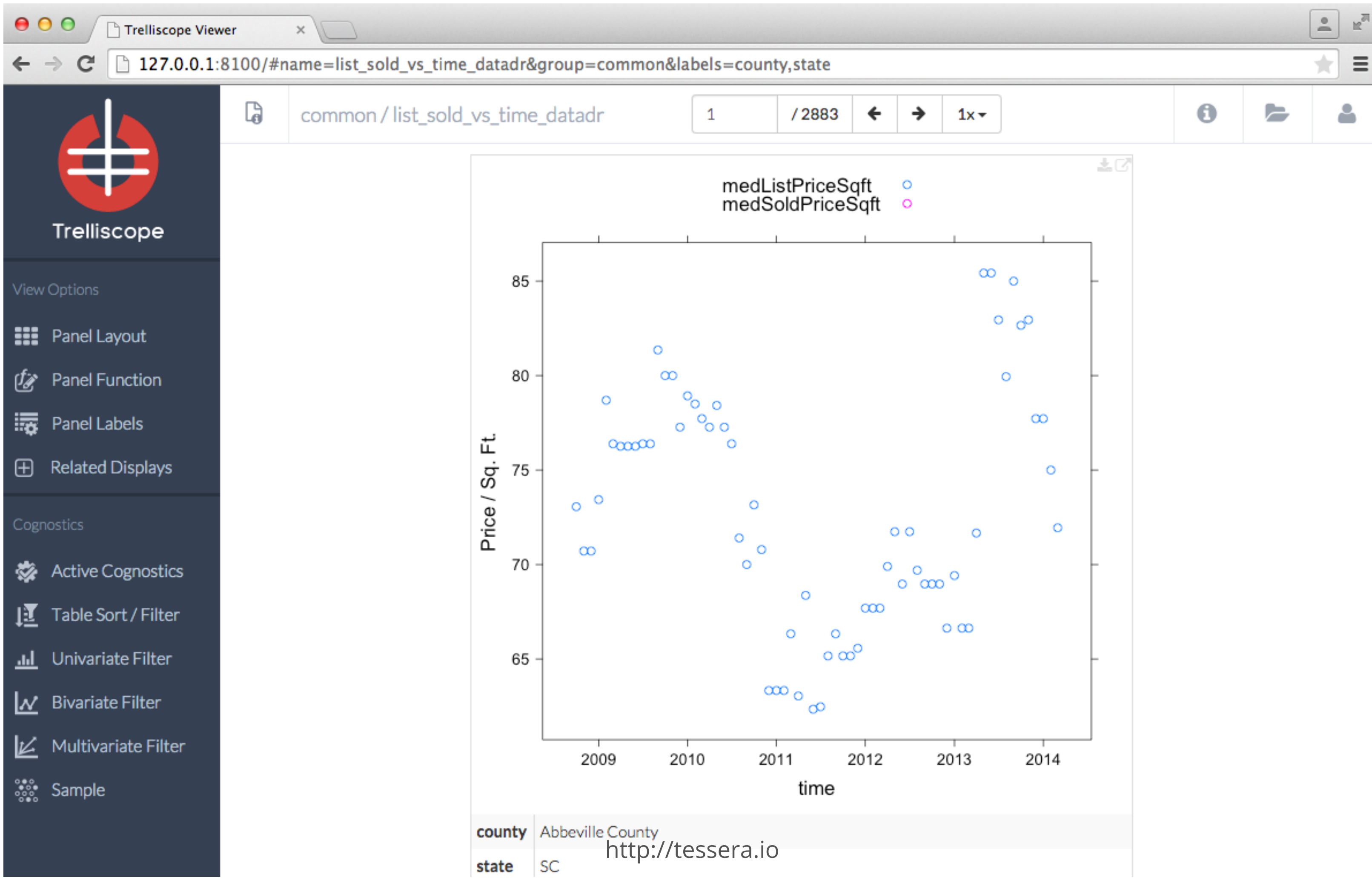
```
vdbConn("housing_vdb", autoYes=TRUE)
```

Creating a Trelliscope display

```
makeDisplay(joinedData,
   name = "list_sold_vs_time_datadr",
   desc = "List and sold price over time",
   panelFn = timePanel,
   width = 400, height = 400,
   lims = list(x = "same")
)
```

```
view()
```

Trelliscope demo



Exercise: create a panel function

```
newPanelFn <- function(x) {
   # fill in here
}

# test the panel function
timePanel(joinedData[[1]]$value)

vdbConn("housing_vdb", autoYes=TRUE)

makeDisplay(joinedData,
   name = "panel_test",
   desc = "Your test panel function",
   panelFn = newPaneFn)
```

Cognostics and display organization

- Cognostic:
  - a value or summary statistic
  - calculated on each subset
  - to help the user focus their attention on plots of interest
- Cognostics are used to sort and filter plots in Trelliscope
- Define a function to apply to each subset to calculate desired values
  - Return a list of named elements
  - Each list element is a single value (no vectors or complex data objects)

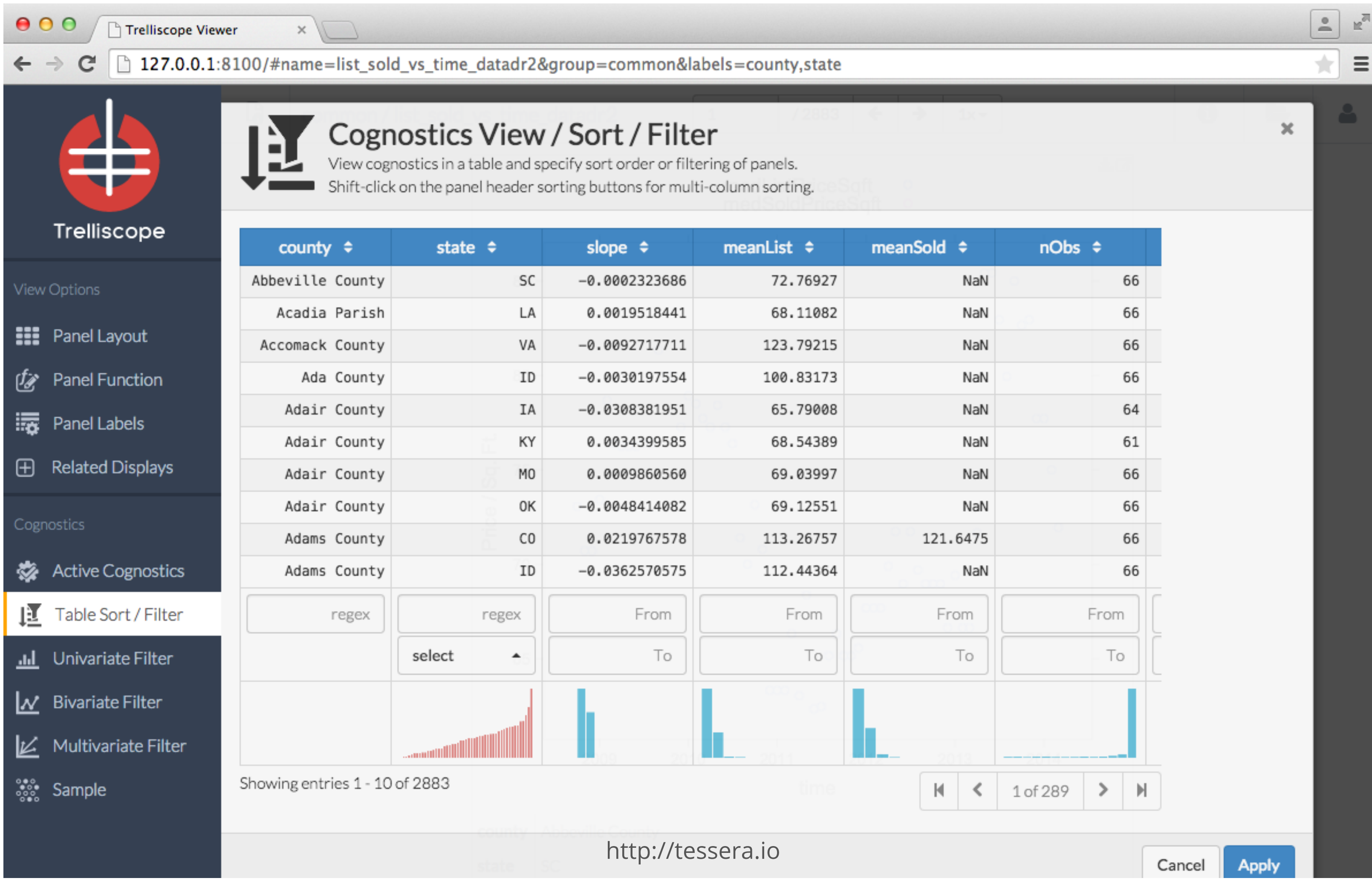
Cognostics function

```r
priceCog <- function(x) {
   st <- getSplitVar(x, "state")
   ct <- getSplitVar(x, "county")
   zillowString <- gsub(" ", "-", paste(ct, st))
   list(
      slope = cog(x$slope, desc = "list price slope"),
      meanList = cogMean(x$housing$medListPriceSqft),
      meanSold = cogMean(x$housing$medSoldPriceSqft),
      lat = cog(x$geo$lat, desc = "county latitude"),
      lon = cog(x$geo$lon, desc = "county longitude"),
      wikiHref = cogHref(x$wiki$href, desc="wiki link"),
      zillowHref = cogHref(
          sprintf("http://www.zillow.com/homes/%s_rb/",
              zillowString),
          desc="zillow link")
   )
}
```

Use the cognostics function in trelliscope

```
makeDisplay(joinedData,
   name = "list_sold_vs_time_datadr2",
   desc = "List and sold price with cognostics",
   panelFn = timePanel,
   cogFn = priceCog,
   width = 400, height = 400,
   lims = list(x = "same")
)
```

Trelliscope demo



Exercise: create a cognostics function

```
newCogFn <- function(x) {
#     list(
#         name1=cog(value1, desc="description")
#     )
}

# test the cognostics function
newCogFn(joinedData[[1]]$value)

makeDisplay(joinedData,
   name = "cognostics_test",
   desc = "Test panel and cognostics function",
   panelFn = newPaneFn,
   cogFn = newCogFn)
view()
```

Analysis of NYC Taxi Data Set

Data download

Data and solution for UseR! 2015: http://tessera.io/docs-UseR2015/

- Subset taxi data for New York City
- First seven days of 2013
- Ryan has posted some analysis of the NYC taxi data: http://hafen.github.io/taxi/#getting-the-data
- NYC taxi rates are complex: http://www.nyc.gov/html/tlc/html/passenger/taxicab_rate.shtml

Data Contents

Variable	Description
medallion	identifier of the individual taxi's license
hack_license	taxi driver license id
vendor_id	unique identifier of taxi owner
rate_code	code based on type of trip
store_and_fwd_flag	ignore
pickup_datetime	date and time of trip start
dropoff_datetime	date and time of trip end
passenger_count	number of riders
trip_time_in_secs	total time between pickup and dropoff
trip_distance	distance covered in miles

Data Contents

Variable	Description
pickup_longitude, pickup_latitude	coordinates for start of trip
dropoff_longitude, dropoff_latitude	coordinates for end of trip
payment_type	"CRD" - credit or debit card, "CSH" - cash
fare_amount	base fare amount
surcharge	additional change for peak hours, certain locations
mta_tax	local tax
tip_amount	tip paid, may be $0
tolls_amount	bridge and tunnel tolls
total_amount	total amount paid by passengers

Running a Local Cluster

- Local cluster can use multiple cores: each running an R process
- R process requires memory for the chunks being processed
- Buffer size limits number of chunks, but large chunks must be loaded into memory
  Hint: look at arguments of localDiskControl
  Hint: look at process size, ps -aux on Linux or Windows Task Manager
- Local calculations often disk-I/O bound
- Clusters with HDFS achieve greater I/O bandwidth

Challenge Questions

- Compute and plot quantiles of passenger_count, trip_time_in_secs, trip_distance, total_amount, tip_amount
- Compute summary statistics, such as mean toll or mean tip percent by distance category
  Hint - cut() on divisions of log distance
  Hint - divide the ddf by distance category
- Explore how tip percent changes with distance category, rate_code and perhaps hour of the day
  Hint - Think of some potentially useful cognostics