How Green Was My Valley

Joe Conway
joe.conway@crunchydata.com
mail@joeconway.com

Crunchy Data

October 17, 2019

Agenda

Spatial Analytics

- Overview
- Ingesting Data
- Analytics

Background

Inspiration for talk - MODIS Normalized Difference Vegetation Index (NDVI)

- Vegetation Indices
  - Derived from typical spectral reflectance signatures of leaves
  - Reflected energy in visible spectrum very low
  - Near-infrared radiation (NIR) scattered with very little absorption
  - Therefore contrast between visible and NIR is sensitive measure of vegetation density
- NDVI
  - Normalized transform of NIR to Red reflectance ratio
  - NDVI = (rNIR - rRed) / (rNIR + rRed)

https://vip.arizona.edu/documents/MODIS/MODIS_VI_UsersGuide_01_2012.pdf

Background

- MOD13A1
  - global, spatial raster data
  - provided about every 16 days
  - 500-meter spatial resolution (250 m and 1 km also available)
- Used for global monitoring of vegetation conditions
- Applications may include:
  - global biogeochemical and hydrologic modeling
  - agricultural monitoring and forecasting
  - land-use planning
  - land cover characterization
  - land cover change detection
- Also covered:
  - Administrative shape data
  - Geocoded data

`https://lpdaac.usgs.gov/dataset_discovery/modis/modis_products_table/mod13a1`

CRUNCHY Enterprise PostgreSQL

Components

- PostgreSQL
  - The world's most advanced open source database.
- PostGIS
  - A spatial database extender for PostgreSQL. Adds support for geographic objects allowing location queries to be run in SQL.
- R
  - An open source language and environment for statistical computing and graphics.
- PL/R
  - R Procedural Language Handler for PostgreSQL. Enables user-defined SQL functions to be written in the R language. Actively developed since early 2003.

https://www.r-project.org
http://www.postgis.net
http://www.joeconway.com/plr

PostgreSQL and R Setup

- Install desired versions of PostgreSQL, PostGIS, R, and PL/R
- Create the database and install Postgres extensions
- Install variety of required/interesting R packages
- Be sure to install R packages as `root` or `postgres`

PostgreSQL Setup

```
createdb gis
psql gis

gis=# create extension postgis;
gis=# create extension plr;
```

R Setup

```
R CMD javareconf
R

[...]

Type 'q()' to quit R.

x <- c("stringi", "raster", "sp", "spatial", "rgdal", "rgeos",
       "maptools", "dplyr", "tidyr", "tmap", "ggmap", "OpenStreetMap",
       "cairoDevice", "RGtk2", "wkb", "rasterVis")
install.packages(x)
```

- Use `getData()` (`rgdal` package) to get shape layer for an administrative area
  - `GADM`: database of global administrative boundaries
    → `http://www.gadm.org`
- Create a PostGIS table and store it there
- Note - `getData()` also supports:
  - `worldclim`: database of global interpolated climate data
  - `SRTM`: Shuttle Radar Topography Mission (SRTM) data
  - `alt`: altitude (elevation) aggregated from SRTM 90m resolution data
  - `countries`: polygons for all countries
  - References
    → `http://www.worldclim.org`
    → `http://srtm.csi.cgiar.org`
    → `http://diva-gis.org/gdata`

Create USA Counties Layer Table

```
CREATE OR REPLACE FUNCTION create_admin_layer(country text, level int, tablename text)
RETURNS text AS $$
  library(raster); library(rgdal)

  # set up database connections
  dsn="PG:user='postgres' dbname='gis' host='localhost'"
  conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")

  # download the requested administrative shapes and create the Postgres table
  shapes = getData('GADM', country = country, level = level)
  writeOGR(shapes, dsn, tablename, "PostgreSQL")
  sql.str <- sprintf("CREATE INDEX %s_idx1 ON %s(name_1,name_2)", tablename, tablename)
  dbGetQuery(conn, sql.str)
  return("OK")
$$ LANGUAGE plr;

DROP TABLE IF EXISTS counties;
SELECT create_admin_layer(country := 'USA', level := 2, tablename := 'counties');
```

Extract and Plot San Diego County

```
-- Note: might need to run "Xvfb :1 -screen 0 1024x768x24"
CREATE OR REPLACE FUNCTION plot_admin_layer(layername text, name1 text, name2 text)
RETURNS bytea AS $$
  library(rgeos); library(cairoDevice); library(RGtk2)

  pixmap <- gdkPixmapNew(w=1024, h=768, depth=24); asCairoDevice(pixmap)

  proj_str <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
  conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")
  sql.str <- sprintf("SELECT st_astext(wkb_geometry) AS geom
                      FROM %s WHERE name_1 = '%s' AND name_2 = '%s'", layername, name1, name2)
  plot(readWKT(dbGetQuery(conn, sql.str), p4s=proj_str))

  plot_pixbuf <- gdkPixbufGetFromDrawable(NULL, pixmap, pixmap$getColormap(),
                                          0, 0, 0, 0, 500, 500)
  buffer <- gdkPixbufSaveToBufferv(plot_pixbuf, "png", character(0), character(0))$buffer
  return(buffer)
$$ LANGUAGE 'plr' STRICT;
```

San Diego County Administrative Boundaries

```
SELECT plr_get_raw(plot_admin_layer('counties', 'California', 'San Diego'));
```

San Diego County Administrative Boundaries - Reprojected

```
p4s <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
mp4s <- "+proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997 +b=6370997 +units=m +no_defs"
plot(spTransform(readWKT(dbGetQuery(conn, sql.str), p4s=p4s), CRS(mp4s)))
```

Extract and Plot San Diego County - alt method #1

```
CREATE OR REPLACE FUNCTION plot_admin_layer_shp(layername text, name1 text, name2 text)
RETURNS bytea AS $$
  library(rgeos); library(cairoDevice); library(RGtk2)
  pixmap <- gdkPixmapNew(w=1024, h=768, depth=24); asCairoDevice(pixmap)

  dsn <- "/home/postgres"
  shapes <- readOGR(dsn=dsn, layername, stringsAsFactors = FALSE)
  plot(shapes[(shapes$NAME_1 %in% c(name1)) & (shapes$NAME_2 %in% c(name2)), ])

  plot_pixbuf <- gdkPixbufGetFromDrawable(NULL, pixmap, pixmap$getColormap(),
                                          0, 0, 0, 0, 500, 500)
  buffer <- gdkPixbufSaveToBufferv(plot_pixbuf, "png", character(0), character(0))$buffer
  return(buffer)
$$ LANGUAGE 'plr' STRICT;
```

Extract and Plot San Diego County - alt method #2

```
CREATE OR REPLACE FUNCTION plot_admin_layer_ogr(layername text, name1 text, name2 text)
RETURNS bytea AS $$
  library(rgeos); library(cairoDevice); library(RGtk2)
  pixmap <- gdkPixmapNew(w=1024, h=768, depth=24); asCairoDevice(pixmap)

  dsn <- "PG:user='postgres' dbname='gis' host='localhost'"
  shapes <- readOGR(dsn=dsn, layername, stringsAsFactors = FALSE)
  plot(shapes[(shapes$name_1 %in% c(name1)) & (shapes$name_2 %in% c(name2)), ])

  plot_pixbuf <- gdkPixbufGetFromDrawable(NULL, pixmap, pixmap$getColormap(),
                                          0, 0, 0, 0, 500, 500)
  buffer <- gdkPixbufSaveToBufferv(plot_pixbuf, "png", character(0), character(0))$buffer
  return(buffer)
$$ LANGUAGE 'plr' STRICT;
```

Performance Comparison of 3 Methods

```
SELECT count(plot_admin_layer('counties', 'California', 'San Diego'));
--Time: 83.189 ms

SELECT count(plot_admin_layer_shp('counties', 'California', 'San Diego'));
--Time: 1754.074 ms

SELECT count(plot_admin_layer_ogr('counties', 'California', 'San Diego'));
--Time: 9989.635 ms
```

Geocoding

- Transforming postal address description to spatial representation
- Process
  - Create list of addresses for Points of Interest (PoI)
  - Use R library ggmap to convert to coordinates
    - Uses either Google Maps API or Data Science Tool Kit
  - Add POI names
  - Set Coordinate Reference System (CRS)
  - Create PostGIS layer table using OGR

Geocoding

```
CREATE OR REPLACE FUNCTION create_poi_layer
  (addresses text[], poinames text[], layername text)
RETURNS text AS $$
  library(ggmap); library(rgdal)

  # geocode using the Google Maps API
  poilayer <- geocode(addresses)

  # add the airport names
  poilayer$name <- poinames

  # convert to SpatialDataFrame
  coordinates(poilayer) <- ~ lon + lat

  [...]
```

Geocoding

```
  [...]

  # specify the CRS
  proj4string(poilayer) <- CRS("+proj=longlat +datum=WGS84")

  # create the Postgres table for this layer
  dsn="PG:user='postgres' dbname='gis' host='localhost'"
  writeOGR(poilayer, dsn, layername, "PostgreSQL")

  # add indexes
  conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")
  sql.str <- sprintf("CREATE INDEX %s_name ON %s(name)", layername, layername)
  dbGetQuery(conn, sql.str)

  return("OK")
$$ LANGUAGE plr;
```

Geocoding

```
DROP TABLE IF EXISTS airports;
SELECT create_poi_layer
(
  addresses := ARRAY['1960 Joe Crosson Dr, El Cajon, CA 92020',
                     '1424 Continental St, San Diego, CA 92154',
                     '3225 N Harbor Dr, San Diego, CA 92101'],
  poinames  := ARRAY['Gillespie Field',
                     'Brown Field',
                     'San Diego Intl'],
  layername := 'airports'
);
```

Plotting Points of Interest

```
# Note: this is pure R code
plot_poi <- function(plotfile, aoilayer, aoiname1, aoiname2, poilayer) {
  library(sp); library(raster); library(rgeos); library(rgdal)
  library(RPostgreSQL)
  Sys.setenv("DISPLAY"=":0.0")

  # required in R, noop in PL/R
  conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")

  # plot to file if destination filename given
  if (plotfile != "") png(plotfile, width = 1024, height = 768)

  [...]
```

Plotting Points of Interest

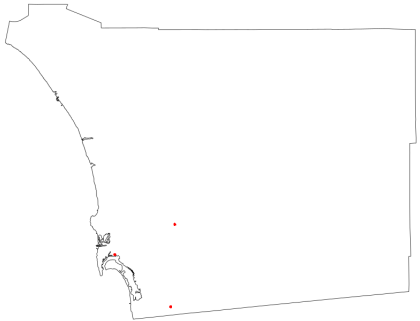
```
[...]

# retrieve boundaries for requested admin area
p4str <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
sql.str <- sprintf("SELECT st_astext(wkb_geometry) AS geom
                    FROM %s WHERE name_1 = '%s' and name_2 = '%s'",
                    aoilayer, aoiname1, aoiname2)
sdc <- readWKT(dbGetQuery(conn, sql.str), p4s=p4str)

# add county boundaries to plot
plot(sdc)

[...]
```

Plotting Points of Interest

```
  [...]

  # retrieve points of interest
  dsn <- "PG:user='postgres' dbname='gis' host='localhost'"
  poi <- readOGR(dsn=dsn, poilayer, stringsAsFactors = FALSE)

  # add poi to plot
  plot(poi, col = "red", pch = 16, add = TRUE)

  if (plotfile != "") dev.off()
}
```

Plotting Points of Interest

```
plot_poi("/tmp/airports1.png", "airports", "counties", "California", "San Diego")
```

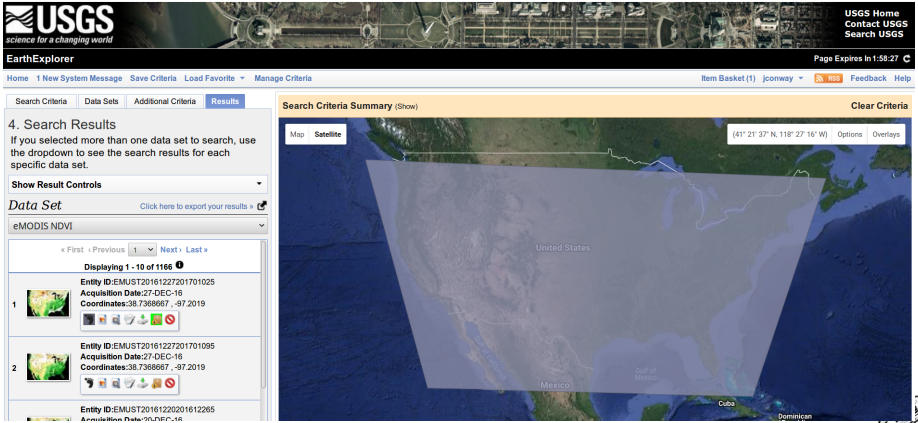


NDVI Download

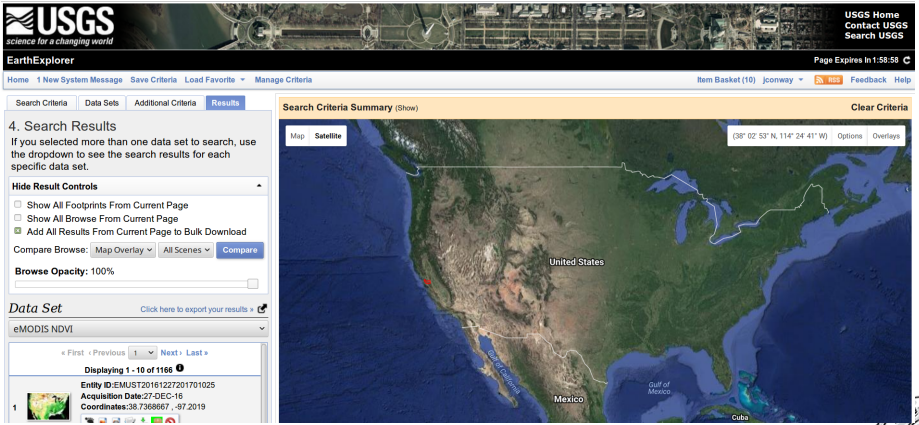
- Get USGS login
- Set search criteria
  →Predefined Area→Add Shape→State→CA→County→Alameda County
  →Date Range→From→To
  →Result Options→Number of records to return→25,000
  →Data Sets→Vegetation Monitoring→eMODIS NDVI
  →Additional Criteria→Platform→TERRA
- Select results
  →Results→Select Results for bulk download (see screenshot) →View Item Basket→Expand eMODIS NDVI→Modify Options For All Scenes→NDVI 500 M →Proceed To Checkout→104.2 GB→Submit
- Download selected results
  →Download/install BDA (bulk download application)

https://earthexplorer.usgs.gov/

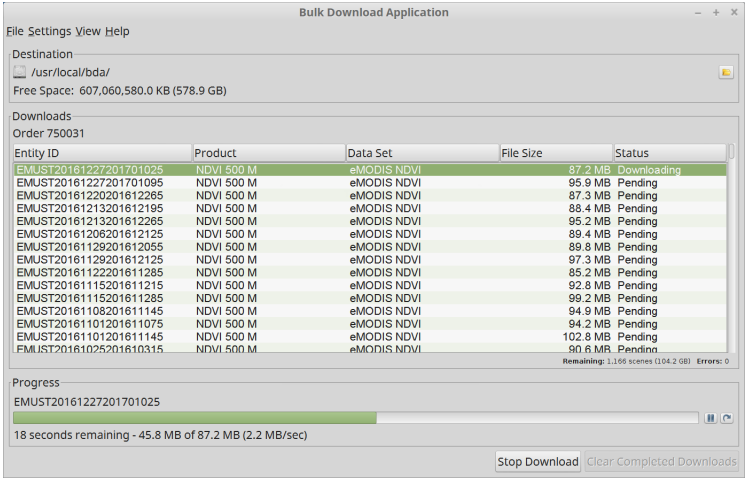
USGS - Footprint



USGS - Adding to Basket



USGS - Bulk Download Application



Preprocessing

- Review downloaded files, eliminate anomolies (9 out of 1166)
- Run preprocessing function
  - Load area of interest (AoI) administrative boundaries
  - Reproject AoI to same projection as MOD13A1 raster data
  - Loop through the downloaded files
    - Unzip File of Interest (FoI)
    - Load NDVI, QA, and Acquisition rasters
    - Crop rasters to AoI (trim matrix to AoI extent)
    - Mask rasters to AoI (mark cells outside AoI as NA)
    - Use QA raster to mark cells with questionable data in NDVI raster as NA
    - Use Acquisition raster to calculate mean date (YYYY.pppp) of NDVI data
  - Capture NDVI rasters and Dates

Preprocessing Function

```
# loop through all the MODIS files and preprocess
CREATE OR REPLACE FUNCTION process_raw_ndvi
  (mywd text, layername text, name1 text, name2 text)
RETURNS bytea AS $$

  [...]

  return(data.frame(ndvi_i, ndvi_dt))
$$ LANGUAGE 'plr' STRICT;
```

Preprocessing Function

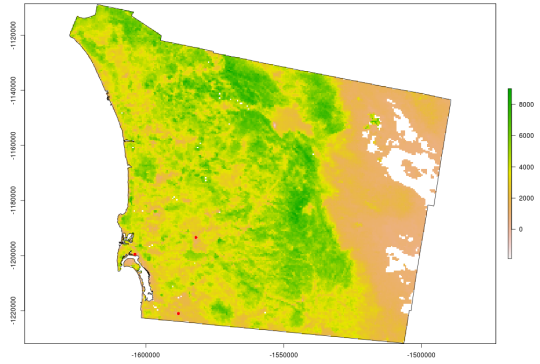
```
CREATE TABLE robjects(objname text primary key, obj bytea);

INSERT INTO robjects
SELECT 'ndvi_dt', process_raw_ndvi('/usr/local/bda/modis',
                                   'counties',
                                   'California',
                                   'San Diego');
--Time: 8010273.572 ms
```

Plotting NDVI

```
# Note: this is pure R code
plot_poi <- function(rastfile, bricklayer, plotfile, aoilayer, aoiname1, aoiname2, poilayer) {
  [...]
  # CRS of the MODIS raster
  mp4str <- "+proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997 +b=6370997 +units=m +no_defs"
  [...]
  # reproject the boundary to the CRS of the MODIS raster
  sdc <- spTransform(sdc, CRS(mp4str))
  [...]
  # read and plot requested raster layer
  ndvi <- brick(rastfile)
  ndvi_layer <- ndvi[[bricklayer]]
  plot(ndvi_layer, add = TRUE)
  [...]
  # reproject the poi to the CRS of the MODIS raster
  poi <- spTransform(poi, CRS(mp4str))
  [...]
}
```

Plotting NDVI

```
plot_poi("/usr/local/bda/modis/ndvi_cooked/ndvi.tif", 42, "/tmp/airports2.png",
        "airports", "counties", "California", "San Diego")
```



Decoding Raster Layer Dates

```
CREATE OR REPLACE FUNCTION get_robj(robjname text) RETURNS bytea AS $$
  SELECT r.obj FROM robjects r WHERE r.objname = robjname
$$ LANGUAGE sql STRICT;
```

Aside - PL/R Modules

- Support for auto-loading R code during interpreter initialization
- Uses special table, `plr_modules`, containing R code to be evaluated
- If table exists, modules defined are fetched and loaded on session start
- `modseq` is used to control the order of installation
- `plr_modules` must be readable by all, but please control writes

Aside - PL/R Modules

```
CREATE TABLE plr_modules (
  modseq int4,
  modsrc text
);

INSERT INTO plr_modules
  VALUES (0, '
ndvi_d2ts <-function(ndvi_d) {
  fnyr <- as.integer(ndvi_d)
  fnfr <- ndvi_d - fnyr
  daysinyr <- 365 + (fnyr %% 4 == 0) - (fnyr %% 100 == 0) + (fnyr %% 400 == 0)
  fnts <- paste(as.character(fnyr), as.character(round(fnfr * (daysinyr))))
  as.Date(fnts, "%Y %j")
}');
```

Decoding Raster Layer Dates

```
CREATE OR REPLACE FUNCTION ndvi_dates(robj bytea, OUT lyr_fn text, OUT lyr_date date)
RETURNS setof RECORD AS $$
  return(data.frame(robj$ndvi_i, format(ndvi_d2ts(robj$ndvi_dt))))
$$ LANGUAGE 'plr' STRICT;

SELECT lyr_fn, lyr_date FROM ndvi_dates(get_robj('ndvi_dt')) LIMIT 3;
                          lyr_fn                           |  lyr_date
-----------------------------------------------------------+------------
 US_eMTH_NDVI.2000.056-062.HKM.COMPRES.005.2009221013507.zip | 2000-02-29
 US_eMTH_NDVI.2000.063-069.HKM.COMPRES.005.2009220192553.zip | 2000-03-09
 US_eMTH_NDVI.2000.070-076.HKM.COMPRES.005.2009220130110.zip | 2000-03-11
(3 rows)

Time: 21.937 ms
```

NDVI - Average over Time

```
CREATE OR REPLACE FUNCTION get_ndvi_mean
  (rastfile text, OUT lyr_date date, OUT lyr_mean float8)
RETURNS setof RECORD AS $$
  library(cairoDevice); library(RGtk2); library(sp); library(raster)

  ndvi <- brick(rastfile)
  ndvi_mn <- cellStats(ndvi, stat = 'mean')

  conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")
  sql.str <- "SELECT lyr_date FROM ndvi_dates(get_robj('ndvi_dt'))"
  ndvi_dt <- dbGetQuery(conn, sql.str)

  return(data.frame(ndvi_dt, ndvi_mn))
$$ LANGUAGE 'plr' STRICT;
```

NDVI - Average over Time

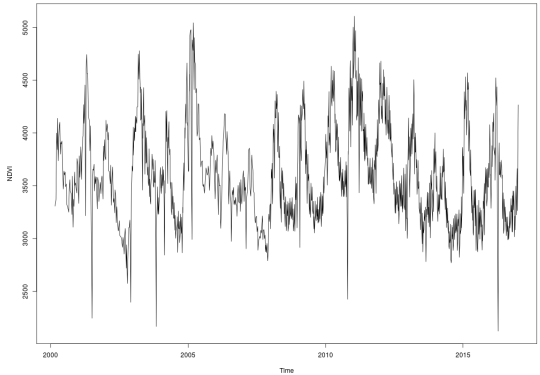
```
SELECT lyr_date, lyr_mean
FROM get_ndvi_mean('/usr/local/bda/modis/ndvi_cooked/ndvi.tif') LIMIT 3;
  lyr_date  |     lyr_mean
------------+------------------
 2000-02-29 | 3307.52445819025
 2000-03-09 | 3358.51562339221
 2000-03-11 | 3372.62659753491
(3 rows)
```

NDVI - Average over Time

```
CREATE OR REPLACE FUNCTION plot_ndvi_trend(rastfile text)
RETURNS bytea AS $$
  library(cairoDevice); library(RGtk2); library(sp); library(raster)
  pixmap <- gdkPixmapNew(w=1024, h=768, depth=24); asCairoDevice(pixmap)

  ndvi <- brick(rastfile)
  conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")
  sql.str <- "SELECT lyr_date FROM ndvi_dates(get_robj('ndvi_dt'))"
  ndvi_dt <- dbGetQuery(conn, sql.str)
  ndvi_mn <- cellStats(ndvi, stat = 'mean')
  idx <- order(ndvi_dt)
  Data <- data.frame(ndvi_dt$lyr_date[idx], ndvi_mn[idx])
  plot(Data, type ="l", xlab = "Time", ylab = "NDVI")

  plot_pixbuf <- gdkPixbufGetFromDrawable(NULL, pixmap, pixmap$getColormap(),
                                          0, 0, 0, 0, 500, 500)
  gdkPixbufSaveToBufferv(plot_pixbuf, "png", character(0), character(0))$buffer
$$ LANGUAGE 'plr' STRICT;
```

NDVI - Average over Time

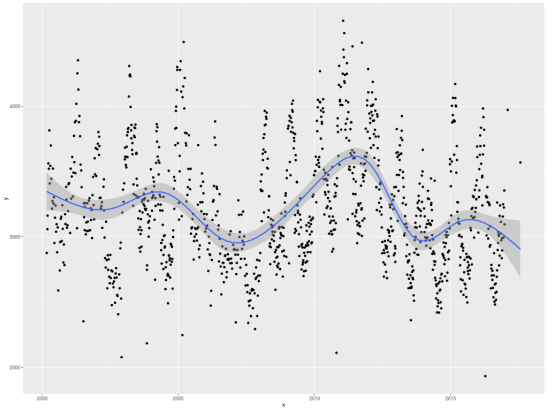
```
SELECT plr_get_raw(plot_ndvi_trend('/usr/local/bda/modis/ndvi_cooked/ndvi.tif'));
```



NDVI - Average over Time with ggplot

```
library(sp); library(raster); library(ggplot2); library(RPostgreSQL)
Sys.setenv("DISPLAY"=":0.0")

ndvi <- brick('/usr/local/bda/modis/ndvi_cooked/ndvi.tif')
conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")
sql.str <- "SELECT lyr_date FROM ndvi_dates(get_robj('ndvi_dt'))"
ndvi_dt <- dbGetQuery(conn, sql.str)
ndvi_mn <- cellStats(ndvi, stat = 'mean')
idx <- order(ndvi_dt)
Data <- data.frame(ndvi_dt$lyr_date[idx], ndvi_mn[idx])

png("/tmp/ndvi_trend_gg.png", width = 1024, height = 768)
ggplot(Data, aes(x,y)) + geom_point() + geom_smooth()
dev.off()
```

NDVI - Average over Time with ggplot



NDVI - Visualize Average by Month

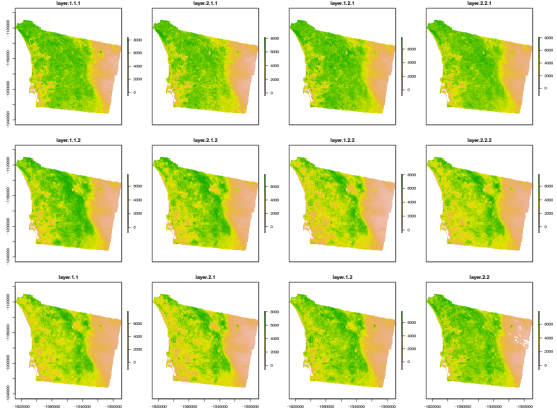
```
plot_ndvi_year <- function(rastfile, layeryr, plotfile) {
  library(sp); library(raster); library(RPostgreSQL)
  Sys.setenv("DISPLAY"=":0.0")

  # required in R, noop in PL/R
  conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")

  # plot to file if destination filename given
  if (plotfile != "") png(plotfile, width = 1024, height = 768)

  conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")
  sql.str <- "SELECT lyr_date FROM ndvi_dates(get_robj('ndvi_dt'))"
  ndvi_dt <- dbGetQuery(conn, sql.str)

  [...]
```

NDVI - Visualize Average by Month

```
  [...]

  # read and plot requested raster layer
  ndvi <- brick(rastfile)
  ndvi_rast <- raster()
  for (i in 1:12) {
    fmt_str <- paste(layeryr, sprintf("%02d", i), sep = "-")
    ndvi_layer <- mean(ndvi[[which(format(ndvi_dt, "%Y-%m") == fmt_str)]], na.rm = TRUE)
    ndvi_rast <- addLayer(ndvi_rast, ndvi_layer)
  }
  plot(ndvi_rast)

  if (plotfile != "") dev.off()
}
```

NDVI - Visualize Average by Month

```
plot_ndvi_year("/usr/local/bda/modis/ndvi_cooked/ndvi.tif", 2011, "/tmp/ndvi-2011.png")
```

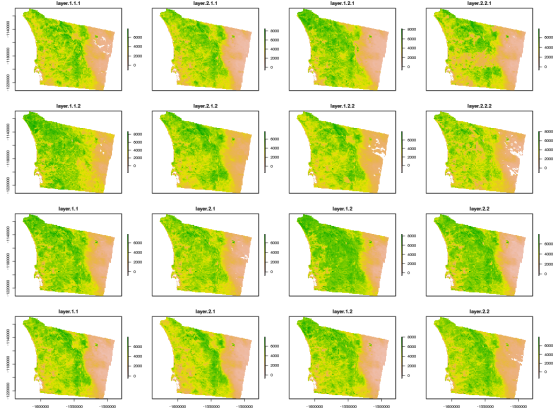


NDVI - Visualize Same Month Average by Year

```
plot_ndvi_month <- function(rastfile, layermonth, firstyr, lastyr, plotfile) {
  [...]

  for (i in firstyr:lastyr) {
    fmt_str <- paste(i, sprintf("%02d", layermonth), sep = "-")
    ndvi_layer <- mean(ndvi[[which(format(ndvi_dt, "%Y-%m") == fmt_str)]], na.rm = TRUE)
    ndvi_rast <- addLayer(ndvi_rast, ndvi_layer)
  }
  plot(ndvi_rast)

  [...]
```

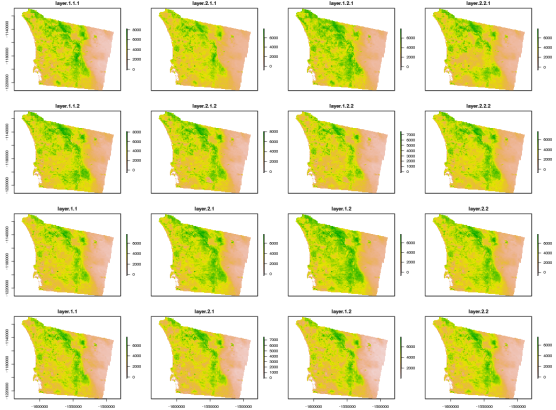
NDVI - Visualize Same Month Average by Year

```
plot_ndvi_year("/usr/local/bda/modis/ndvi_cooked/ndvi.tif", 1, 2001, 2016, "/tmp/ndvi-1.png")
```



NDVI - Visualize Same Month Average by Year

```
plot_ndvi_year("/usr/local/bda/modis/ndvi_cooked/ndvi.tif", 8, 2001, 2016, "/tmp/ndvi-8.png")
```



Questions?

Thank You!
mail@joeconway.com

PL/R Advantages

- Leverage people's knowledge and skills
  - statistics/math, database, web are distinct specialties
- Leverage hardware
  - server better able to handle analysis of large datasets
- Processing/bandwidth efficiency
  - why send large datasets across the network?
- Consistency of analysis
  - ensure analysis done consistently once vetted
- Abstraction of complexity
  - keep system understandable and maintainable
- Leverage R
  - rich core functionality and huge ecosystem

PL/R Disadvantages

- PostgreSQL user
  - Slower than standard SQL aggregates and PostgreSQL functions for simple cases
  - New language to learn
- R user
  - Debugging more challenging than working directly in R
  - Less flexible for ad hoc analysis
  - New language to learn

Creating PL/R Functions

- Standard R functions

```
func_name <- function(myarg1 [,myarg2...]) {
  function body referencing myarg1 [, myarg2 ...]
}
```

- PL/R

```
CREATE OR REPLACE FUNCTION func_name(arg-type1 [, arg-type2 ...])
RETURNS return-type AS $$
    function body referencing arg1 [, arg2 ...]
$$ LANGUAGE 'plr';

CREATE OR REPLACE FUNCTION func_name(myarg1 arg-type1
                                     [, myarg2 arg-type2 ...])
RETURNS return-type AS $$
    function body referencing myarg1 [, myarg2 ...]
$$ LANGUAGE 'plr';
```

RPostgreSQL Compatibility

- Allows prototyping using R, move to PL/R for production
- Queries performed in current database
- Driver/connection parameters ignored; dbDriver, dbConnect, dbDisconnect, and dbUnloadDriver are no-ops

```
dbDriver(character dvr_name)
dbConnect(DBIDriver drv, character user, character password,
          character host, character dbname, character port,
          character tty, character options)
dbSendQuery(DBIConnection conn, character sql)
fetch(DBIResult rs, integer num_rows)
dbClearResult (DBIResult rs)
dbGetQuery(DBIConnection conn, character sql)
dbReadTable(DBIConnection conn, character name)
dbDisconnect(DBIConnection conn)
dbUnloadDriver(DBIDriver drv)
```

RPostgreSQL Compatibility Example

- PostgreSQL access from R

```
get_ndvi_mean<-function(rastfile) {
  library(cairoDevice); library(RGtk2); library(sp); library(raster); require(RPostgreSQL)

  ndvi <- brick(rastfile)
  ndvi_mn <- cellStats(ndvi, stat = 'mean')

  conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")
  sql.str <- "SELECT lyr_date FROM ndvi_dates(get_robj('ndvi_dt'))"
  ndvi_dt <- dbGetQuery(conn, sql.str)

  return(data.frame(ndvi_dt, ndvi_mn))
}
```

RPostgreSQL Compatibility Example

- Same function from PL/R

```
CREATE OR REPLACE FUNCTION get_ndvi_mean
  (rastfile text, OUT lyr_date date, OUT lyr_mean float8)
RETURNS setof RECORD AS $$
  library(cairoDevice); library(RGtk2); library(sp); library(raster)

  ndvi <- brick(rastfile)
  ndvi_mn <- cellStats(ndvi, stat = 'mean')

  conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")
  sql.str <- "SELECT lyr_date FROM ndvi_dates(get_robj('ndvi_dt'))"
  ndvi_dt <- dbGetQuery(conn, sql.str)

  return(data.frame(ndvi_dt, ndvi_mn))
$$ LANGUAGE 'plr' STRICT;
```

Aside - Example Use of Raw Image from SQL

- Need screen buffer on typical server:

```
Xvfb :1 -screen 0 1024x768x24
export DISPLAY=:1.0
```

- Calling it from PHP

```
<?php
$dbconn = pg_connect("...");
$rs = pg_query( $dbconn,
      "SELECT plr_get_raw(plot_ndvi_trend('/usr/local/bda/modis/ndvi_cooked/ndvi.tif'))");
$hexpic = pg_fetch_array($rs);
$cleandata = pg_unescape_bytea($hexpic[0]);

header("Content-Type: image/png");
header("Last-Modified: " .
       date("r", filectime($_SERVER['SCRIPT_FILENAME'])));
header("Content-Length: " . strlen($cleandata));
echo $cleandata;
?>
```

Preprocessing Function

```
# loop through all the MODIS files and preprocess
CREATE OR REPLACE FUNCTION process_raw_ndvi
  (mywd text, layername text, name1 text, name2 text)
RETURNS bytea AS $$
  library(raster)
  library(rgdal)
  library(rgeos)

  # initialize
  ingest_dn <- paste(mywd, "/ndvi_raw/", sep = "")
  export_dn <- paste(mywd, "/ndvi_cooked/", sep = "")
  dir.create(export_dn, showWarnings = FALSE)
  ndvi_rast <- raster()
  ndvi_dt <- numeric()
  ndvi_i <- character()

  [...]
```

Preprocessing Function

```
[...]

# retrieve boundaries for requested admin area of interest
p4str <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
conn <- dbConnect(dbDriver("PostgreSQL"), user="postgres", dbname="gis", host="localhost")
sql.str <- sprintf("SELECT st_astext(wkb_geometry) AS geom
                    FROM %s WHERE name_1 = '%s' and name_2 = '%s'",
                    layername, name1, name2)
aoi <- readWKT(dbGetQuery(conn, sql.str), p4s=p4str)

# reproject the boundary to the CRS of the MODIS raster
mp4str <- "+proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997 +b=6370997 +units=m +no_defs"
aoi <- spTransform(aoi, CRS(mp4str))

[...]
```

Aside - Zip File Naming

- Example:
  `US_eMTH_NDVI.2016.363-011.HKM.COMPRES.005.2016016065930.zip`
  - Most important part for our purposes is `2016.363-011`
  - Represents dataset year, first DOY, last DOY
  - When first DOY > last DOY, year correlates with first DOY
  - When first DOY < last DOY, year correlates with both
- Interpretation of `2016.363-011`
  - First acquisition = 2015, DOY 363
  - Last acquisition = 2016, DOY 011

Preprocessing Function

```
[...]

# loop through all of the MODIS raster files
for (i in dir(path=ingest_dn, pattern="[.]zip$")) {

  # get a list and unzip the files in this archive
  fnoi <- unzip(paste(ingest_dn, i, sep=""), list=TRUE)
  fnparts <- unlist(strsplit(i,'[.]'))
  fnyr <- as.numeric(fnparts[2])
  fndoy <- as.numeric(unlist(strsplit(fnparts[3], "-")))
  unzip(paste(ingest_dn, i, sep=""), exdir = ingest_dn)

  [... loop continues ...]
```

Preprocessing Function

```
[... loop continues ...]

# map the geotiff files to rasterbricks and crop to region of interest
ndvi <- paste(ingest_dn, fnoi$Name[grep("VI_NDVI.*tif", fnoi$Name)], sep="")
r <- brick(ndvi)
r <- crop(r, extent(aoi))

qual <- paste(ingest_dn, fnoi$Name[grep("VI_QUAL.*tif", fnoi$Name)], sep="")
rq <- brick(qual)
rq <- crop(rq, extent(aoi))

acqu <- paste(ingest_dn, fnoi$Name[grep("VI_ACQI.*tif", fnoi$Name)], sep="")
ra <- brick(acqu)
ra <- crop(ra, extent(aoi))

[... loop continues ...]
```

Preprocessing Function

```
[... loop continues ...]

# mask cells to region of interest
r <- mask(r, aoi)
rq <- mask(rq, aoi)
ra <- mask(ra, aoi)

# mark cells with questionable data as NA
r[[1]][(rq[[1]] != 0)] <- NA

[... loop continues ...]
```

Aside - Acquistion File

- Raster layer of 16-bit integers where:
  - Each cell represents the data selected for the corresponding NDVI cell
  - Most significant portion of integer represents day of year (DOY)
  - Least significant portion represents acquisition number on that day
  - DOY values can be from one to three digits
  - Acquisition number (`acq_num`) will be two digits
  - Cell value = `DOY * 100 + acq_num`
- Examples:
  - 202: DOY 2, second acquisition of the day
  - 34211: DOY 342, eleventh acquisition of the day

Preprocessing Function

```
[... loop continues ...]

# create raster layer with YYYY.fff for each cell
if (fndoy[1] > fndoy[2]) {
  pfnyr <- fnyr - 1
  len_fnyr1 <- 365 + (pfnyr %% 4 == 0) - (pfnyr %% 100 == 0) + (pfnyr %% 400 == 0)
  len_fnyr2 <- 365 + ( fnyr %% 4 == 0) - ( fnyr %% 100 == 0) + ( fnyr %% 400 == 0)

  sel1 = trunc(ra[[1]] / 100) >= fndoy[1]
  sel2 = trunc(ra[[1]] / 100)  < fndoy[1]

  ra[[1]][sel1] <- pfnyr + (trunc(ra[[1]][sel1] / 100) / len_fnyr1)
  ra[[1]][sel2] <-  fnyr + (trunc(ra[[1]][sel2] / 100) / len_fnyr2)
} else {
  len_fnyr <- 365 + (fnyr %% 4 == 0) - (fnyr %% 100 == 0) + (fnyr %% 400 == 0)
  ra[[1]] <- fnyr + (trunc(ra[[1]]/100) / len_fnyr)
}

[... loop continues ...]
```

Preprocessing Function

```
  [... loop continues ...]

  # get mean point in time for represented area
  ra_mean <- mean(ra[[1]][], na.rm = TRUE)

  ndvi_rast <- addLayer(ndvi_rast, r)
  ndvi_dt <- c(ndvi_dt, ra_mean)
  ndvi_i <- c(ndvi_i, i)

  # clean up unzipped files
  file.remove(paste(ingest_dn, fnoi$Name, sep=""))

  # end of preprocessing loop
}

[...]
```

Preprocessing Function

```
  [...]

  ndviout <- paste(export_dn, "ndvi.tif", sep="")
  writeRaster(ndvi_rast, ndviout, overwrite=TRUE)

  return(data.frame(ndvi_i, ndvi_dt))
$$ LANGUAGE 'plr' STRICT;
```