1. Create Maps With R Geospatial Classes and Graphics Tools (*Making Maps*)
2. Read and write ESRI Shape Files (*ESRI*)
3. Display T Spatial Objects with Google Maps and Google Earth (*KML*)
4. Read and Display Data from GPS Devices Using R (*GPX*)
5. Overlay Points on Satellite Image / Extract Pixel Values (*Raster*)

# Read and Display Data from GPS Devices Using R
# Part 5

# `readGPS` function

Read, translate, display tracks and waypoints stored in GPS Exchange (GPX) format

- Data collected using GPS receivers are stored in a myriad of formats; scientists wishing to use GPS data must be prepared to read any of these formats in order to import their GPS data into a scientific software package.

# `readGPS` function

- This section demonstrates use of the `readGPS` function, included in the `R` **maptools** package.
- The `readGPS()` function is an R interface to the *GPSBabel* utility, which provides file conversion and data manipulation tools for many popular GPS data formats.
- For today, we selected the GPS Exchange (GPX) format, GPS platform-independent and widely-used by Web-aware programs to exchange geospatial data.

# Demonstration:

The demonstration has three parts:

1. Read the input files into R Data Frames
2. Plot the data points
3. Write a subset of the input data to CSV output files

Let's look at the R script, section by section. This example code contains two functions:

1. `TheDriver()`, which manages execution and generates data plots,
2. `ConvertGPXFiles()`, which extracts and filters GPS data & writes .CSV files.

# Demonstration:

- Let's look at the R source code (GPX folder).
- Program documentation: This describes the two functions in the package.

# Demonstration:

- This demonstration reads GPX-format files containing, respectively, waypoints and tracks collected with a GPS reciever, into an R Data Frame.
- After this, it extracts selected columns from each incoming data frame (date/time/latitude/longitude/altitude) into a new and separate data frame.
- The new data frame is then written to a comma-separated-value (CSV) file.

# Demonstration:

**Functions:**

- TheDriver() - Manages execution of demonstration function
- ConvertGPXFiles() - Reads input .GPX- format file, writes a subset to an output .CSV file.

Remarks:

- debug() and browser() statements are incorporated to allow the analysis to 'single-step' through the example:
- debug() : Sets up single-line execution of the function.
- browser(): Interrupts execution of function, allows inspection of `R` variable values.

Convert a file containing Waypoints

```
ConvertGPXFiles("SampleWaypointsOnlyNoOutliers.gpx",
   "SampleWaypointsOut.csv","w")
#
# Plot the waypoints
#
 plot(dfWayptForPlotting@coords[,1:2],type="p",
     xlab="longitude",ylab="latitude")
 title("Waypoints from GPX-format GPS file")
```

Next, call ConvertGPXFiles() for two different input files:
1- extract waypoints(single locations)
2- extract tracks (linear featues defined by a series of points):

```
# Convert a file containing Tracks
ConvertGPXFiles("SampleTracksOnly.gpx",
 "SampleTracksOut.csv","t")

 #t for tracks
```

Next create a plot using the tracks data, then add ('overlay') the waypoints:

# Plot the track

- With the default form , plot axes do not display.
- To plot x and y axes, 'deconstruct' the x and y coordinates: 'embedded' in the `SpatialLinesDataFrame`.

# outputs

```
....
 plot(sldfTracksForPlotting@lines[[1]]@Lines[[1]]@coords[
 sldfTracksForPlotting@lines[[1]]@Lines[[1]]@coords[,2],
 type="l",xlab="longitude",ylab="latitude",add=TRUE)
 points(dfWayptForPlotting@coords[,1:2],type="p",col="red
 title("GPX/GPS Tracks (black) | Waypoints (red)")
....
```

# ConvertGPXFiles():

i. extraction of GPS data stored in the standard GPX format into an R data frame, using the R readGPS function;

ii. extraction of key fields (Date/Time, Latitude/Longitude, Elevation) from the initial data frame into a new data frame.

# function arguments

- inGPSFile (string): Input GPX file name
- outConvertFile (string): Output CSV file name
- FileType (string): Flag indicates input
- file data type: "w" for waypoints, "t" for tracks.

```
ConvertGPXFiles <- function(inGPSFile,
   outConvertFile,FileType)
{
```

Two processing sections,based on file type: first the section for waypoints, then the section for tracks

```
if (FileType == "w")
{
..
}

if (FileType == "t")
{
..
}
```

- Read the GPX-format waypoints into a Data Frame
- Note here: readGPS converts waypoints and tracks into data frames with different column layouts.
- Columns are labeled V1 - Vn)

```
gRawWaypt = readGPS("gpx",inGPSFile,"w")
```

# Get number of observations (waypoints)

`gRawWaypoint` data frame contains the attributes that we desire in the following columns:

- V3: Observation Date (factor)
- V4: Observation Time (factor)
- V8: Descruptive Label (string)
- V10: Latitude (numeric)
- V11: Longitude (numeric)
- V21: Elevation (Factor, includes M in last character position)

- Lets extract these columns, convert Date, Time to strings
- Convert elevation to numeric format,
- Construct a new data frame containing only the columns of interest.

```
 nObs = length(gRawWaypt[,"V3"])
 sDate = as.character(gRawWaypt[,"V3"])
 sTime = as.character(gRawWaypt[,"V4"])
 sLabel = as.character(gRawWaypt[,"V9"])
 fLat = as.numeric(gRawWaypt[,"V10"])
 fLong = as.numeric(gRawWaypt[,"V11"])
#
 bounds = c(range(fLong),range(fLat))
 dim(bounds) = c(2,2)
```

Elevation is a factor with the letter 'M' appended. Remove this, and convert the elevation to numeric

```
fAlt <- as.numeric(substring
 (as.character(gRawWaypt$V21),1,
 (nchar(as.character(gRawWaypt$V21))-1)))
```

Output data frames - One 'standard' DF for file output, one 'SpatialPointsDataFrame' for plotting.

```
dfWaypoints <<- as.data.frame(cbind(sLabel,sDate,
      sTime,fLat,fLong,fAlt))
write.csv(dfWaypoints,outConvertFile)

LatLongCoords = SpatialPoints(cbind(fLong,fLat),
  proj4string = CRS("+proj=longlat"))

dfWayptForPlotting <<- SpatialPointsDataFrame
 (LatLongCoords,
 bbox=as.matrix(bounds),
 dfWaypoints[1])
...
```

Here is the section for tracks:

```
 else if (FileType == "t")
 {
 gRawTracks = readGPS("gpx",inGPSFile,"t")
..
```

A GPX file can include multiple tracks, known as track sequences, each of which contains the vertices for a single track (line). *However, the current version of `readGPS()` combines all points in all track sequences into a single track. (This may change in the future).*

- Here we will convert these track points into a `SpatialLinesDataFrame` (with a single SpatialLines object in one 'row' and a single attribute - "Track 1" - attached to the track (and stored in a DataFrame).
- The data frame generated by readGPS for Tracks has a different layout:
  V3: Latitude (numeric)
  V4: Longitude (numeric)
  V14: Elevation (Factor, includes 'M' in last character position)

We use two R geospatial data structures, provided by the **sp** package:

- `SpatialPointsDataFrame` for waypoints,
- `SpatialLinesDataFrame` for tracks.

**SpatialPointsDataFrame:**
One attribute (altitude) per point.

```
dfTrackPoints <<- SpatialPointsDataFrame
  (LatLongCoords,data.frame(fAlt))
```

Create SpatialLinesDF from the SpatialPointsDF.
First, SpatialLines object:

```
slTrackLine = Lines(list
   (Line(dfTrackPoints@coords)))
slTrackLine@ID = "Track_One"
SLPath = SpatialLines(list(slTrackLine),
 proj4string = CRS("+proj=longlat"))
```

Finally, the `SpatialLinesDataFrame`:

- Create a global variable using the "¡¡-" operator
- Plot this in TheDriver().
- Write the original track points to a CSV file.

```
 sldfTracksForPlotting <<- SpatialLinesDataFrame(SLPath,T

 write.csv(dfTrackPoints,outConvertFile)
 print(sprintf("done - tracks"))
 }
}
```

Here is the R command sequence that loads and runs the program:

```
> source("ExtractWaypoints.r")
> TheDriver()
```

Here are the plots. Note that two 'outlier' waypoints are omitted from the right-side plot as they do not fall within the tracks.