A. Motivation

During the recent RStudio Conference, an attendee asked the panel about the lack of support provided by the tidyverse in relation to time series data. As someone who has spent the majority of their career on time series problems, this was somewhat surprising because R already has a great suite of tools for visualizing, manipulating, and modeling time series data. I can understand the desire for a ‘tidyverse approved’ tool for time series analysis, but it seemed like perhaps the issue was a lack of familiarity with the available toolage. Therefore, I wanted to put together a list of the packages and tools that I use most frequently in my work. For those unfamiliar with time series analysis, this could a good place to start investigating Rs current capabilities.

B. Background

Time series data refers to a sequence of measurements that are made over time at regular or irregular intervals with each observation being a single dimension. An example of low dimensional time series is daily wind temperature from 01/01/2001 through 12/31/2005. High dimensional time series is characterized by a larger number of observations, so an example could be the daily wind temperature from 01/01/1980 through 12/31/2010. In either case, the goal of the analysis could lead one to perform regression, clustering, forecasting, or even classification.

C. Data For Examples

To run the code in this post, you will need to access the following data through the unix terminal. It will download a csv file from the City of Chicago website that contains  
information on reported incidents of crime that occurred in the city of Chicago from 2001 to present.

$ wget –no-check-certificate –progress=dot https://data.cityofchicago.org/api/views/ijzp-q8t2/rows.csv?accessType=DOWNLOAD > chicago\_crime\_data.csv

Import the data into R and get the aggregate number of reported incidents of theft by day.

library(data.table)

dat = fread("chicago\_crime\_data.csv")

colnames(dat) = gsub(" ", "\_", tolower(colnames(dat)))

dat[, date2 := as.Date(date, format="%m/%d/%Y")]

mydat = dat[primary\_type=="THEFT", .N, by=date2][order(date2)]

mydat[1:6]

D. Data Representation

The first set of packages that one should be aware of is related to data storage. One could use data frames, tibbles, or data tables, but there are already a number of data structures that are optimized for representing time series data. The fundamental time series object is “ts”. However, the “ts” class has a number of limitations, and so it is usually best to work with the extensible time series (“xts”) obect.

D1. xts

The xts package offers a number of great tools for data manipulation and aggregation. At it’s core is the xts object, which is essentially a matrix object that can represent time series data at different time increments. Xts is a subclass of the zoo object, and that provides it with a lot of functionality.

Here are some functions in xts that are worth investigating:

library(xts)

# create a xts object

mydat2 = as.xts(mydat)

mydat2

plot.xts(mydat2)

# filter by date

mydat2["2015"] ## 2015

mydat2["201501"] ## Jan 2015

mydat2["20150101/20150105"] ## Jan 01 to Jan 05 2015

# replace all valuues from Aug 25 onwards with 0

mydat2["20170825/"] <- 0

mydat2["20170821/"]

# get the last one month

last(mydat2, "1 month")

# get stats by time frame

apply.monthly(mydat2, sum)

apply.monthly(mydat2, quantile)

period.apply(mydat2, endpoints(mydat2,on='months'), sum)

period.apply(mydat2, endpoints(mydat2,on='months'), quantile)

E. Dates

R has a maddening array of date and time classes. Be it yearmon, POSIXct, POSIXlt, chron, or something else, each has specific strengths and weaknesses. In general, I find myself using the lubridate package as it simplifies many of the complexities associated with date-times in R.

E1. lubridate

The lubridate package provides a lot of functionality for parsing and formatting  
dates, comparing different times, extracting the components of a date-time, and so  
forth.

library(lubridate)

ymd("2010-01-01")

mdy("01-01-2010")

ymd\_h("2010-01-01 10")

ymd\_hm("2010-01-01 10:02")

ymd\_hms("2010-01-01 10:02:30")

F. Time Series Regression

Distributed lag models (error correction models) are a core component of doing time series analysis. They are many instances where we want to regress an outcome variable at the current time against values of various regressors at current and previous times. dynlm and ardl (wrapper for dynlm) are solid for this type of analysis. Another common task when working with distributed lag models involves using dynamic simulations to understand estimated outcomes in different scenarios. dynsim provides a coherent solution for simulation and visualization of those estimated values of the target variable.

F1. dynlm / ardl

Here is a brief example of how dynlm can be utilized. In what follows, I have created a new variable and lagged it by one day. So the model attempts to regress incidents or reported theft based on the weather from the previous day.

library(dynlm)

mydat = dat[primary\_type=="THEFT", .N, by=date2][order(date2)]

mydat[, weather := sample(c(20:90), dim(mydat), replace=TRUE)]

mydat[, weather\_lag := shift(weather, 1, type = 'lag')]

mod = dynlm(N ~ L(weather), data = mydat2)

summary(mod)

F2. dynsim

Here is a brief example of how dynlm can be utilized. In what follows, I have created a new variable and lagged it by one day. I’ve used the dynsim to product two dynamic simulations and plotted them.

library(dynsim)

mydat3 = mydat[1:10000]

mod = lm(N ~ weather\_lag, data = mydat3)

Scen1 <- data.frame(weather\_lag = min(mydat2$weather\_lag, na.rm=T))

Scen2 <- data.frame(weather\_lag = max(mydat2$weather\_lag, na.rm=T))

ScenComb <- list(Scen1, Scen2)

Sim1 <- dynsim(obj = mod, ldv = 'weather\_lag', scen = ScenComb, n = 20)

dynsimGG(Sim1)

D. Forecasting

D1. forecast

The forecast package is the most used package in R for time series forecasting. It contains functions for performing decomposition and forecasting with exponential smoothing, arima, moving average models, and so forth. For aggregated data that is fairly high dimensional, one of the techniques present in this package should provide an adequate forecasting model given that the assumptions hold.

Here is a quick example of how to use the auto.arima function in R. In general, automatic forecasting tools should be used with caution, but it is a good place to explore time series data.

library(forecast)

mydat = dat[primary\_type=="THEFT", .N, by=date2][order(date2)]

fit = auto.arima(mydat[,.(N)])

pred = forecast(fit, 200)

plot(pred)

D2. smooth

The smooth package provides functions to perform even more variations of exponential smoothing, moving average models, and various seasonal arima techniques. The smooth and forecast package are usually more than adequate for most forecasting problems that pertain to high dimensional data.

Here is a basic example that uses the automatic complex exponential smoothing function:

library(smooth)

mydat = dat[primary\_type=="THEFT", .N, by=date2][order(date2)]

fit = auto.ces(mydat[,N])

pred = forecast(fit, 200)

plot(pred)

So for those of you getting introduced to the R programming language, these are a list extremely useful packages for time series analysis that you will want to get some exposure to.