CT5102: Programming for Data Analytics

Week 7: Useful Functions in R

https://github.com/JimDuggan/CT5102

Dr. Jim Duggan,
Information Technology,
School of Engineering & Informatics

(1) Missing Values

- In most projects, data is likely to be incomplete, for reasons such as:
  - Missed questions
  - Faulty equipment (sensor data)
  - Improperly coded data
- In R, missing values are represented by the symbol NA
- Function is.na() tests for missing values
- Returns an object of the same size

```
> v<-c(1:3,NA,5)
> v
[1]  1  2  3 NA  5
> is.na(v)
[1] FALSE FALSE FALSE  TRUE FALSE
> v[!is.na(v)]
[1] 1 2 3 5
```

```
> v<-c(1:3,NA,5)
> v
[1]  1  2  3 NA  5
> sum(v)
[1] NA
> sum(v,na.rm = T)
[1] 11
```

Recode missing values

```
> v<-c(1:3,NA,5)
> v
[1]  1  2  3 NA  5
> v[is.na(v)]<- -1
> v
[1]  1  2  3 -1  5
```

```
> ages<-c(23,65,1111,20)
> ages
[1]   23   65 1111   20
> ages[ages>110]<-NA
> ages
[1] 23 65 NA 20
```

Dealing with data frames – na.omit()

```
> age<-c(18,27,31,18,26)
> q1<-c(4,2,1,4,5)
> q2<-c(1,NA,1,4,3)
> q3<-c(4,4,5,4,5)
> q4<-c(4,3,3,2,1)
> q5<-c(3,NA,1,3,5)
>
> df<-data.frame(age,q1,q2,q3,q4,q5)
> df
  age q1 q2 q3 q4 q5
1  18  4  1  4  4  3
2  27  2 NA  4  3 NA
3  31  1  1  5  3  1
4  18  4  4  4  2  3
5  26  5  3  5  1  5
```

```
> na.omit(df)
  age q1 q2 q3 q4 q5
1  18  4  1  4  4  3
3  31  1  1  5  3  1
4  18  4  4  4  2  3
5  26  5  3  5  1  5
```

Finding incomplete data
complete.cases(df)

```
> df
  age q1 q2 q3 q4 q5
1  18  4  1  4  4  3
2  27  2 NA  4  3 NA
3  31  1  1  5  3  1
4  18  4  4  4  2  3
5  26  5  3  5  1  5
> complete.cases(df)
[1]  TRUE FALSE  TRUE  TRUE  TRUE
> df[!complete.cases(df),]
  age q1 q2 q3 q4 q5
2  27  2 NA  4  3 NA
```

(2) pmin() and pmax()

Returns the (parallel) maxima and minima of the input values.

```
> v1<-1:10
> v2<-10:1
> v1
 [1]  1  2  3  4  5  6  7  8  9 10
> v2
 [1] 10  9  8  7  6  5  4  3  2  1
> pmax(v1,v2)
 [1] 10  9  8  7  6  6  7  8  9 10
> pmin(v1,v2)
 [1] 1 2 3 4 5 5 4 3 2 1
```

(3) Sorting

- Ordinary numerical sorting of a vector can be done with the sort() function

```
> v<-c(12,87,34,23,10)
> v
[1] 12 87 34 23 10
> sort(v)
[1] 10 12 23 34 87
> sort(v,decreasing = T)
[1] 87 34 23 12 10
>
```

Function order()

- Arranges the indices of the sorted values
- Can be used to sort data frames
- Default increasing order

```
> v
[1] 12 87 34 23 10
> order(v)
[1] 5 1 4 3 2
> order(v, decreasing = T)
[1] 2 3 4 1 5
```

Use with data frame

```
> df
  age   area
1  21 Dublin
2  34 Dublin
3  24 Galway
4  19 Galway
5  56 Dublin
> r<-order(df$age)
> r
[1] 4 1 3 2 5
> df[r,]
  age   area
4  19 Galway
1  21 Dublin
3  24 Galway
2  34 Dublin
5  56 Dublin
```

```
> df
  age   area
1  21 Dublin
2  34 Dublin
3  24 Galway
4  19 Galway
5  56 Dublin
> r<-order(df$area,decreasing = T)
> r
[1] 3 4 1 2 5
> df[r,]
  age   area
3  24 Galway
4  19 Galway
1  21 Dublin
2  34 Dublin
5  56 Dublin
```

Order by more than 1 column

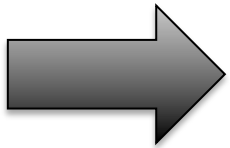
```
> df
  age   area
1  21 Dublin
2  34 Dublin
3  24 Galway
4  19 Galway
5  19 Dublin
> r<-order(df$age,df$area)
> r
[1] 5 4 1 3 2
```

```
> df[r,]
  age   area
5  19 Dublin
4  19 Galway
1  21 Dublin
3  24 Galway
2  34 Dublin
```

Challenge 7.1

- Sort the following data frame (on software product information) into descending order

```
> ds1.df
   Effort Duration  Size
1    16.7     23.0  6050
2    22.6     15.5  8363
3    32.2     14.0 13334
4     3.9      9.2  5942
5    17.3     13.5  3315
6    67.7     24.5 38988
7    10.1     15.2 38614
8    19.3     14.7 12762
9    10.6      7.7 13510
10   59.5     15.0 26500
```



```
   Effort Duration  Size
6    67.7     24.5 38988
7    10.1     15.2 38614
10   59.5     15.0 26500
9    10.6      7.7 13510
3    32.2     14.0 13334
8    19.3     14.7 12762
2    22.6     15.5  8363
1    16.7     23.0  6050
4     3.9      9.2  5942
5    17.3     13.5  3315
```

(4) Set Operations

Function	Description
union(x,y)	Union of the sets x and y
intersect(x,y)	Intersection of the sets x and y
setdiff(x,y)	Set difference between x and y, consisting of all elements of x that are not in y
setequal(x,y)	Test for set equality between x and y
c %in% y	Membership, testing whether c is an element of the set y

Examples

```
> x<-c(1,2,3)
> y<-c(3,4,5)
> x
[1] 1 2 3
> y
[1] 3 4 5
> union(x,y)
[1] 1 2 3 4 5
> intersect(x,y)
[1] 3
```

```
> setdiff(x,y)
[1] 1 2
> setdiff(y,x)
[1] 4 5
> setequal(x,y)
[1] FALSE
> 3 %in% x
[1] TRUE
> 3 %in% y
[1] TRUE
```

(5) get() function

- Return the value of a named object.
- A way to reference variables.

```
> x<-10
> y<-20
> z<-30
>
> vars<-c("x","y","z")
>
> for(v in vars){
+   o<-get(v)
+   cat(o," ")
+ }
10  20  30
```

ls() function

- ls() returns a vector of character strings giving the names of the objects in the specified environment.
- When invoked with no argument at the top level prompt, ls shows what data sets and functions a user has defined.
- When invoked with no argument inside a function, ls returns the names of the function's local variables: this is useful in conjunction with browser

```
> ls()
 [1] "age"            "ages"           "area"           "b1"
 [5] "b2"             "c1"             "d"              "date_now"
 [9] "days"           "df"             "getCurrentTime" "ind"
[13] "index"          "Jan1"           "now"            "o"
[17] "op"             "p"              "props"          "q1"
[21] "q2"             "q3"             "q4"             "q5"
> myvars<-ls(pattern="v")
> myvars
 [1] "v"    "v1"   "v2"   "v3"   "v4"   "v5"   "v6"   "v7"
```

Challenge 7.2

Based on the following vectors (v1 v2, v3), create a fourth vector (v4) that contains the variables in a form that can processed by the get() function. The fourth vector should be created automatically. Then write an sapply() function that will process the vector v4 and get the average of the three vectors.

```
> v1<-1:10
> v2<-11:20
> v3<-21:30
> v1
 [1]  1  2  3  4  5  6  7  8  9 10
> v2
 [1] 11 12 13 14 15 16 17 18 19 20
> v3
 [1] 21 22 23 24 25 26 27 28 29 30
```

(6) Date Values

- Typically entered into R as character strings and then translated into date variables that are stored numerically.
- Function as.Date() used to make the translation
- Syntax is as.Date(x,"*input_format*"), where x is the character data and *input_format* gives the appropriate format for reading the date.
- Supports date arithmetic

Date & Time Formats

Symbol	Meaning	Example
%d	Day as number (0-31)	01-31
%a %A	Abbreviated weekday Unabbreviated weekday	Mon Monday
%m	Month (00-12)	00-12
%b %B	Abbreviated month Unabbreviated month	Jan January
%y %Y	2-digit year 4-digit year	07 2007
%H	Decimal hour	00-23
%M	Decimal minute	00-59
%S (%OS)	Decimal second/ millsecond	00-61 (allowing for up to 2 leap seconds)

Example: days until end of S1

```
> now<-Sys.Date()
> now
[1] "2015-10-17"
> s1<-as.Date("2015-11-27","%Y-%m-%d")
> s1
[1] "2015-11-27"
> weeks<-difftime(s1, now, unit="week")
> weeks
Time difference of 5.857143 weeks
> days<-difftime(s1, now, unit="day")
> days
Time difference of 41 days
```

Finding new dates

- 37 Days from a given date?

```
> s1<-as.Date("2015-11-27","%Y-%m-%d")
> s1
[1] "2015-11-27"
> s1+37
[1] "2016-01-03"
> s1-37
[1] "2015-10-21"
```

Getting the system date and time

- Sys.Date()
- Sys.time()

```
> date_now<-Sys.Date()
> time_now<-Sys.time()
> date_now
[1] "2015-10-17"
> time_now
[1] "2015-10-17 18:11:23.890 IST"
```

Challenge 7.3

- Write a function that returns the following:

```
> getCurrentTime()
$Date
[1] "2015-10-17"

$Timestamp
[1] "2015-10-17 14:40:25.375 IST"
```

strptime(), dealing with times

```
> today <- strptime("2015-10-17 14:29:55.111", "%Y-%m-%d %H:%M:%OS")
> today
[1] "2015-10-17 14:29:55.111 IST"
> str(today)
 POSIXlt[1:1], format: "2015-10-17 14:29:55.111"
> tme<-unclass(today)
> str(tme)
List of 11
 $ sec   : num 55.1
 $ min   : int 29
 $ hour  : int 14
 $ mday  : int 17
 $ mon   : int 9
 $ year  : int 115
 $ wday  : int 6
 $ yday  : int 289
 $ isdst : int 1
 $ zone  : chr "IST"
 $ gmtoff: int NA
```

strptime() generates a POSIXlt object

- Class "POSIXlt" is a named list of vectors representing

Vector	Values	Vector	Values
sec	0-61: seconds	**wday**	0-6 day of week, starting on Sunday
min	0-59: minutes	**yday**	0-365: Day of year
hour	0-23: hours	**isdst**	Daylight saving time flag
mday	1-31: day of month	**zone**	Abbreviation of time zone
mon	0-11: months after first of year	**gmtoff**	Offset in seconds from GMT
year	Years since 1900		

Useful functions for POSIXlt object

```
> today
[1] "2015-10-17 14:29:55.111 IST"
> weekdays(today)
[1] "Saturday"
> months(today)
[1] "October"
> quarters(today)
[1] "Q4"
```

strftime() – Extracting information from POSIXlt object into string format

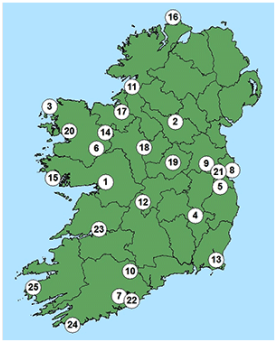
```
> today
[1] "2015-10-17 14:29:55.111 IST"
> strftime(today,"Year = %Y")
[1] "Year = 2015"
> strftime(today,"AM/PM = %p")
[1] "AM/PM = pm"
> strftime(today,"Week of year = %U")
[1] "Week of year = 41"
> strftime(today,"Year with Century = %Y")
[1] "Year with Century = 2015"
```

%A	Full weekday name in the current locale. (Also matches abbreviated name on input.)
%b	Abbreviated month name in the current locale. (Also matches full name on input.
%B	Full month name in the current locale. (Also matches abbreviated name on input.)
%c	Date and time. Locale-specific on output, "%a %b %e %H:%M:%S %Y" on input.
%C	Century (00–99): the integer part of the year divided by 100.
%d	Day of the month as decimal number (01–31).
%D	Date format such as %m/%d/%y: ISO C99 says it should be that exact format.
%e	Day of the month as decimal number (1–31), with a leading space for a single-digit number.
%F	Equivalent to %Y-%m-%d (the ISO 8601 date format).
%g	The last two digits of the week-based year (see %V). (Accepted but ignored on input.)
%G	The week-based year (see %V) as a decimal number. (Accepted but ignored on input.)
%h	Equivalent to %b.
%H	Hours as decimal number (00–23). As a special exception strings such as 24:00:00 are accepted for input, since ISO 8601 allows these.
%I	Hours as decimal number (01–12).
%j	Day of year as decimal number (001–366).
%m	Month as decimal number (01–12).
%M	Minute as decimal number (00–59).
%n	Newline on output, arbitrary whitespace on input.
%p	AM/PM indicator in the locale. Used in conjunction with %I and **not** with %H. An empty string in some locales (and the behaviour is undefined if used for input in such a locale). Some platforms accept %P for output, which uses a lower-case version: others will output P.
%r	The 12-hour clock time (using the locale's AM or PM). Only defined in some locales.

%R	Equivalent to %H:%M.
%S	Second as decimal number (00–61), allowing for up to two leap-seconds (but POSIX-compliant implementations will ignore leap seconds).
%t	Tab on output, arbitrary whitespace on input.
%T	Equivalent to %H:%M:%S.
%u	Weekday as a decimal number (1–7, Monday is 1).
%U	Week of the year as decimal number (00–53) using Sunday as the first day 1 of the week (and typically with the first Sunday of the year as day 1 of week 1). The US convention.
%V	Week of the year as decimal number (00–53) as defined in ISO 8601. If the week (starting on Monday) containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. (Accepted but ignored on input.)
%w	Weekday as decimal number (0–6, Sunday is 0).
%W	Week of the year as decimal number (00–53) using Monday as the first day of week (and typically with the first Monday of the year as day 1 of week 1). The UK convention.
%x	Date. Locale-specific on output, "%y/%m/%d" on input.

%X	Time. Locale-specific on output, "%H:%M:%S" on input.
%y	Year without century (00–99). On input, values 00 to 68 are prefixed by 20 and 69 to 99 by 19 – that is the behaviour specified by the 2004 and 2008 POSIX standards, but they do also say 'it is expected that in a future version the default century inferred from a 2-digit year will change'.
%Y	Year with century. Note that whereas there was no zero in the original Gregorian calendar, ISO 8601:2004 defines it to be valid (interpreted as 1BC): see http://en.wikipedia.org/wiki/0_(year). Note that the standards also say that years before 1582 in its calendar should only be used with agreement of the parties involved. For input, only years 0:9999 are accepted.
%z	Signed offset in hours and minutes from UTC, so -0800 is 8 hours behind UTC. Values up to +1400 are accepted as from **R** 3.1.1: previous versions only accepted up to +1200.
%Z	(Output only.) Time zone abbreviation as a character string (empty if not available). This may not be reliable when a time zone has changed abbreviations over the years.

(7) Time Series Object (ts)

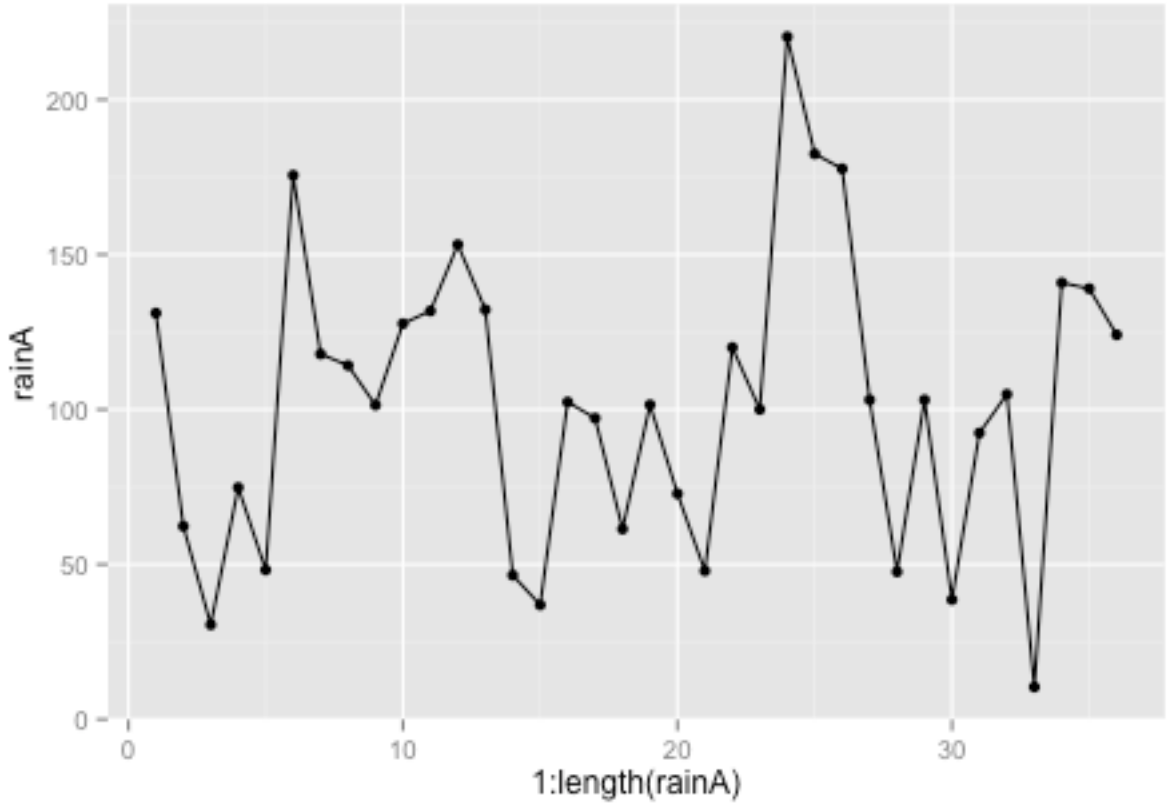


- The function ts is used to create time-series objects.
- These are vector or matrices with class of "ts" (and additional attributes) which represent data which has been sampled at equispaced points in time.
- In the matrix case, each column of the matrix data is assumed to contain a single (univariate) time series.

Monthly values for Athenry up to 17-oct-2015

Total rainfall in millimetres for Athenry

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Annual
2015	191.1	68.7	129.9	74.8	138.0	44.9	138.2	114.6	93.3	17.1			1010.6
2014	182.5	177.7	103.1	47.6	103.1	38.6	92.4	104.9	10.4	140.9	139.0	124.1	1264.3
2013	132.2	46.5	36.9	102.4	97.2	61.4	101.5	72.8	47.9	120.0	100.0	220.3	1139.1
2012	131.1	62.3	30.5	74.8	48.2	175.6	117.9	114.2	101.4	127.7	131.8	153.2	1268.7
mean	116.7	87.8	94.7	72.0	75.3	79.6	86.5	107.8	100.3	128.9	120.3	123.2	1192.9




```
# Montly rainfall in athenry from 2012-2014
rainA<-c(131.1,  62.3,   30.5,   74.8,   48.2,  175.6,
         117.9, 114.2,  101.4,  127.7,  131.8,  153.2,
         132.2,  46.5,   36.9,  102.4,   97.2,   61.4,
         101.5,  72.8,   47.9,  120.0,  100.0,  220.3,
         182.5, 177.7,  103.1,   47.6,  103.1,   38.6,
         92.4,  104.9,   10.4,  140.9,  139.0,  124.1)

qplot(x=1:length(rainA),y=rainA,geom=c("point","line"))
```

Create a ts object

- **data:** a vector or matrix of the observed time-series values. A data frame will be coerced to a numeric matrix via data.matrix.
- **start:** the time of the first observation. Either a single number or a vector of two integers, which specify a natural time unit and a (1-based) number of samples into the time unit. See the examples for the use of the second form.
- **end**: the time of the last observation, specified in the same way as start.
- **frequency:** the number of observations per unit of time.

```
> r.ts<-ts(rainA,start=c(2012,1),end=c(2014,12),frequency=12)
> r.ts
       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
2012 131.1  62.3  30.5  74.8  48.2 175.6 117.9 114.2 101.4 127.7 131.8 153.2
2013 132.2  46.5  36.9 102.4  97.2  61.4 101.5  72.8  47.9 120.0 100.0 220.3
2014 182.5 177.7 103.1  47.6 103.1  38.6  92.4 104.9  10.4 140.9 139.0 124.1
```

Using the window() function

- window is a generic function which extracts the subset of the object x observed between the times start and end.
- If a frequency is specified, the series is then re-sampled at the new frequency.
- The parameter deltat can be used to extract values at specified intervals

```
> window(r.ts,
+        start=c(2012,1),
+        end=c(2012,4))
       Jan   Feb   Mar   Apr
2012 131.1  62.3  30.5  74.8
```

```
> window(r.ts,deltat=1)
Time Series:
Start = 2012
End = 2014
Frequency = 1
[1] 131.1 132.2 182.5
```

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2012	131.1	62.3	30.5	74.8	48.2	175.6	117.9	114.2	101.4	127.7	131.8	153.2
2013	132.2	46.5	36.9	102.4	97.2	61.4	101.5	72.8	47.9	120.0	100.0	220.3
2014	182.5	177.7	103.1	47.6	103.1	38.6	92.4	104.9	10.4	140.9	139.0	124.1

Getting max and min values (library zoo for as.yearmon)

```
> r.ts
       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
2012 131.1  62.3  30.5  74.8  48.2 175.6 117.9 114.2 101.4 127.7 131.8 153.2
2013 132.2  46.5  36.9 102.4  97.2  61.4 101.5  72.8  47.9 120.0 100.0 220.3
2014 182.5 177.7 103.1  47.6 103.1  38.6  92.4 104.9  10.4 140.9 139.0 124.1
> as.yearmon(time(r.ts))
 [1] "Jan 2012" "Feb 2012" "Mar 2012" "Apr 2012" "May 2012" "Jun 2012" "Jul 2012"
 [8] "Aug 2012" "Sep 2012" "Oct 2012" "Nov 2012" "Dec 2012" "Jan 2013" "Feb 2013"
[15] "Mar 2013" "Apr 2013" "May 2013" "Jun 2013" "Jul 2013" "Aug 2013" "Sep 2013"
[22] "Oct 2013" "Nov 2013" "Dec 2013" "Jan 2014" "Feb 2014" "Mar 2014" "Apr 2014"
[29] "May 2014" "Jun 2014" "Jul 2014" "Aug 2014" "Sep 2014" "Oct 2014" "Nov 2014"
[36] "Dec 2014"
>
> as.yearmon(time(r.ts))[which.max(r.ts)]
[1] "Dec 2013"
>
> as.yearmon(time(r.ts))[which.min(r.ts)]
[1] "Sep 2014"
```

R Functions for ts analysis

R functions for time series analysis by Vito Ricci (vito_ricci@yahoo.com) R.0.5 26/11/04

R FUNCTIONS FOR TIME SERIES ANALYSIS

Here are some helpful R functions for time series analysis. They belong from *stats*, *tseries*, *ast* and *lmtest* packages and grouped by their goal.

INPUT

cycle(): gives the positions in the cycle of each observation (stats)
deltat(): returns the time interval between observations (stats)
end(): extracts and encodes the times the last observation were taken (stats)
frequency(): returns the number of samples per unit time (stats)
read.ts(): reads a time series file (tseries)
start(): extracts and encodes the times the first observation were taken (stats)
time(): creates the vector of times at which a time series was sampled (stats)
ts(): creates time-series objects (stats)
window(): is a generic function which extracts the subset of the object 'x' observed between the times 'start' and 'end'. If a frequency is specified, the series is then re-sampled at the new frequency (stats)

TS DECOMPOSITION

decompose(): decomposes a time series into seasonal, trend and irregular components using moving averages. Deals with additive or multiplicative seasonal component (stats)
filter(): linear filtering on a time series (stats)
*HoltWinters(): c*omputes Holt-Winters Filtering of a given time series (stats)
sfilter(): removes seasonal fluctuation using a simple moving average (ast)
spectrum(): estimates the spectral density of a time series (stats)
stl(): decomposes a time series into seasonal, trend and irregular components using 'loess' (stats)
tsr(): decomposes a time series into trend, seasonal and irregular. Deals with additive and multiplicative components (ast)

TESTS

adf.test(): computes the Augmented Dickey-Fuller test for the null that 'x' has a unit root (tseries)
Box.test(): computes the Box-Pierce or Ljung-Box test statistic for examining the null hypothesis of independence in a given time series (stats)
bds.test(): computes and prints the BDS test statistic for the null that 'x' is a series of i.i.d. random variables (tseries)
bptest(): performs the Breusch-Pagan test for heteroskedasticity of residuals (lmtest)
dwtest(): performs the Durbin-Watson test for autocorrelation of residuals (lmtest)
jarque.bera.test(): Jarque-Bera test for normality (tseries)
kpss.test(): computes KPSS test for stationarity (tseries)

STOCHASTIC MODELS

ar(): fits an autoregressive time series model to the data, by default selecting the complexity by AIC (stats)
arima(): fits an ARIMA model to a univariate time series (stats)
arima.sim(): simulate from an ARIMA model (stats)
arma(): fits an ARMA model to a univariate time series by conditional least squares (tseries)
garch(): fits a Generalized Autoregressive Conditional Heteroscedastic GARCH(p, q) time series model to the data by computing the maximum-likelihood estimates of the conditionally normal model (tseries)