Tidyverse

Dom Owens

29/10/2019

The **Tidyverse** is a collection of compatible R packages based around data frames, improving on base R and unifying the data analysis process.

```
# install.packages("tidyverse")
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

We cover

- `magrittr`
- `ggplot2`
- `dplyr`
- `tidyr`

For manipulating data and illustrating wider concepts.

Pipes

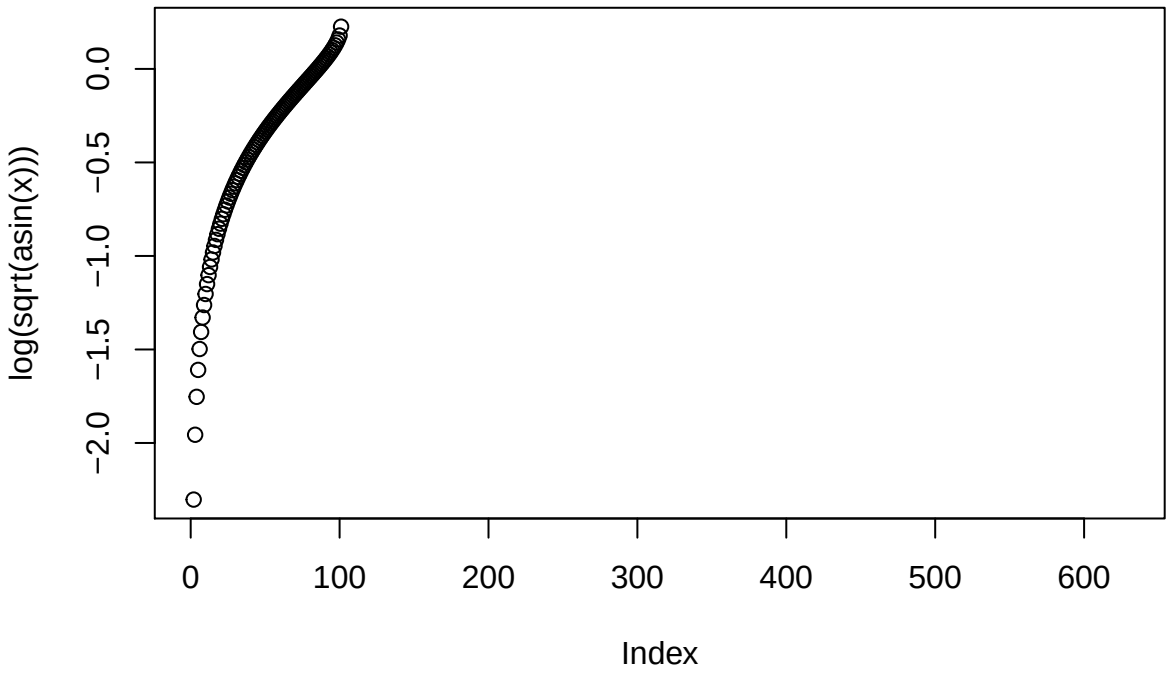
We can pass arguments to functions using pipes `%>%`, from the `magrittr` package (named for the artist Rene Magritte).

Why is piping useful? Consider plotting

$$\log(\sqrt{arcsin(x)})$$

```
x <- seq(0, 2*pi, by = 0.01) #interval [0,2*pi]
plot(log(sqrt(asin(x)))) #plot log sqrt arcsin x
```

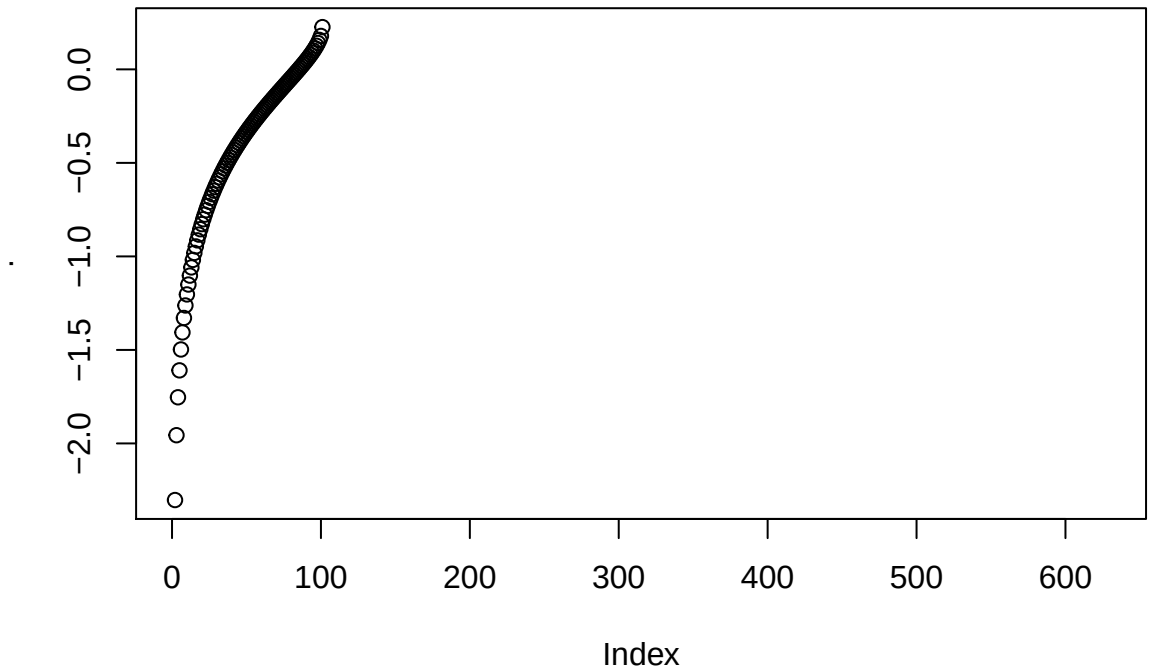
```
## Warning in asin(x): NaNs produced
```



We compare the syntax for piping:

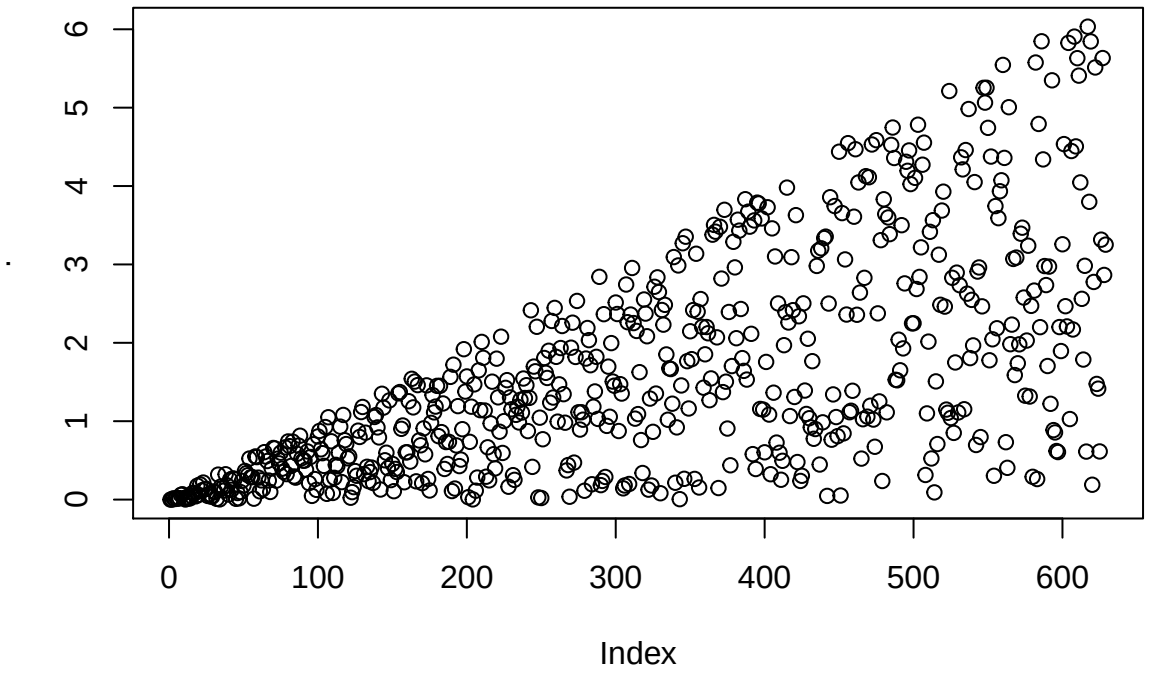
```
x %>% asin %>% sqrt %>% log %>% plot #plot log sqrt arcsin x
```

```
## Warning in asin(.): NaNs produced
```



This clarifies the location where arguments are being passed. `.` is used to assign the location which is being piped to; the default is the first argument.

```
y <- x %>% runif(1:length(x), 0, .) #pipe to final argument
y %>% plot #plot
```



These are particularly useful when manipulating data (see Lab sheet).

Other types of pipe include:

The **assignment pipe** %<>%

```
library(magrittr)
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

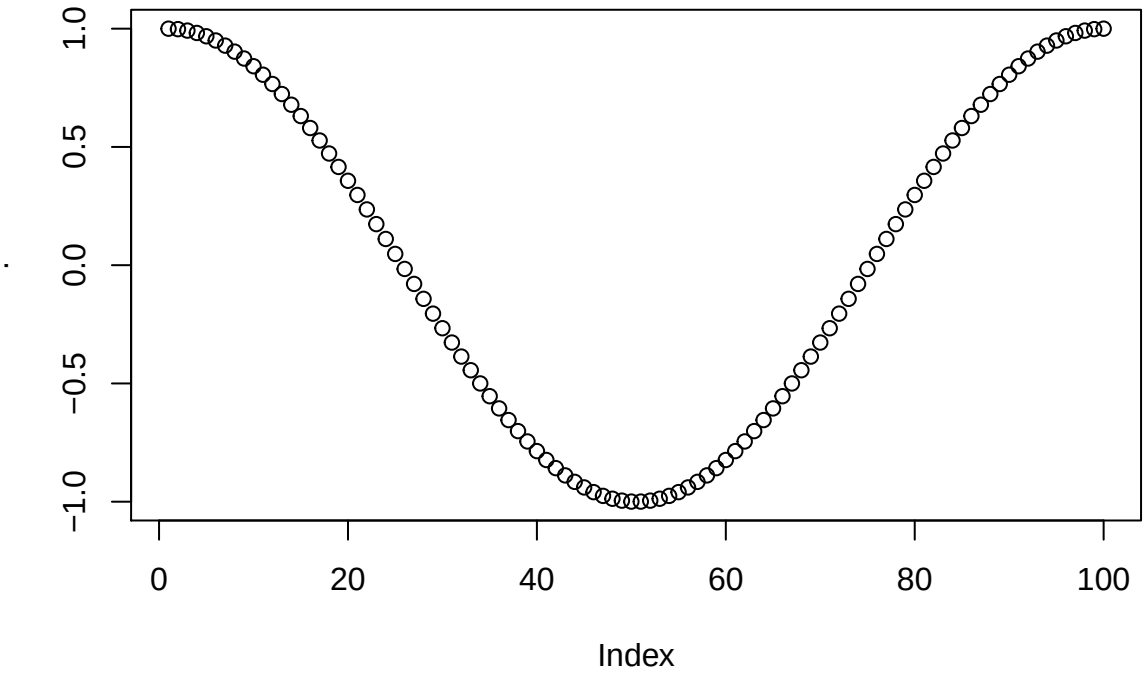
```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
x <- runif(10, 0, 10)
x %<>% sort
x
```

```
##  [1] 1.729708 2.057194 2.536604 2.979598 3.207516 3.215442 4.494873
##  [8] 6.522751 7.270604 7.915659
```

The **tee pipe** %T>%, which stores the left hand side of the line as opposed to the right hand side.

```
x <- 100 %>%
  seq(0, 2*pi, length.out = .) %>%
  cos %T>%
  plot
```




```
x[1:10]
```

```
##  [1] 1.0000000 0.9979867 0.9919548 0.9819287 0.9679487 0.9500711 0.9283679
##  [8] 0.9029265 0.8738494 0.8412535
```

ggplot2

We see a different approach to plotting in R, using the layered grammar of graphics.

We use the following additive notation:

```
#ggplot(data = <data.frame>) +
#  <geom_layer>(mapping = aes(<variables_map>))
```

See the lab sheet for demonstration.

Case study

Consider a poisson GAM

$$y \sim Pois(\mu(\mathbf{x}))$$
$$\log \mu(\mathbf{x}) = \beta_0 + f_1(x_1) + f_2(x_2)$$

These functions are non-linear and built using spline basis expansions.

```
##install.packages("qgam")
library(qgam)
```

```
## Loading required package: mgcv
```

```
## Loading required package: nlme
```

```
##
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
## This is mgcv 1.8-30. For overview type 'help("mgcv-package")'.
```

```
data(UKload) #load data
head(UKload)
```

```
##     NetDemand       wM   wM_s95       Posan      Dow      Trend
## 25      38353 6.046364 5.558800 0.001369941   samedi 1293879600
## 73      41192 2.803969 3.230582 0.004109824 dimanche 1293966000
## 121     43442 2.097259 1.858198 0.006849706    lundi 1294052400
## 169     50736 3.444187 2.310408 0.009589588    mardi 1294138800
## 217     50438 5.958674 4.724961 0.012329471 mercredi 1294225200
## 265     50064 4.124248 4.589470 0.015069353    jeudi 1294311600
##     NetDemand.48 Holy Year                Date
## 25         38353    1 2011 2011-01-01 12:00:00
## 73         38353    0 2011 2011-01-02 12:00:00
## 121        41192    0 2011 2011-01-03 12:00:00
## 169        43442    0 2011 2011-01-04 12:00:00
## 217        50736    0 2011 2011-01-05 12:00:00
## 265        50438    0 2011 2011-01-06 12:00:00
```

```
fitG <- gam(NetDemand ~ Dow + s(wM) + s(wM_s95) + s(Posan) +
                        s(NetDemand.48) + s(Trend, k = 6), data = UKload) # fit GAM to data
```

There are many reasons for using `ggplot`, including:

- The ability to add features to the `plot` function
- The ability to control element properties
- The ability to change the order of rendering

This is implemented for GAMs with the`mgcViz` package, which wraps the GAM object using `ggplot2`.

```
##install.packages("mgcViz")
##library(mgcViz)
##fitG_v <- getViz(fitG) #convert into visualisation object
```

dplyr

`ggplot2` asks for a dataframe as an argument, as do many modelling functions. Together, `dplyr` and `tidyr` allow us to combine data into this format.

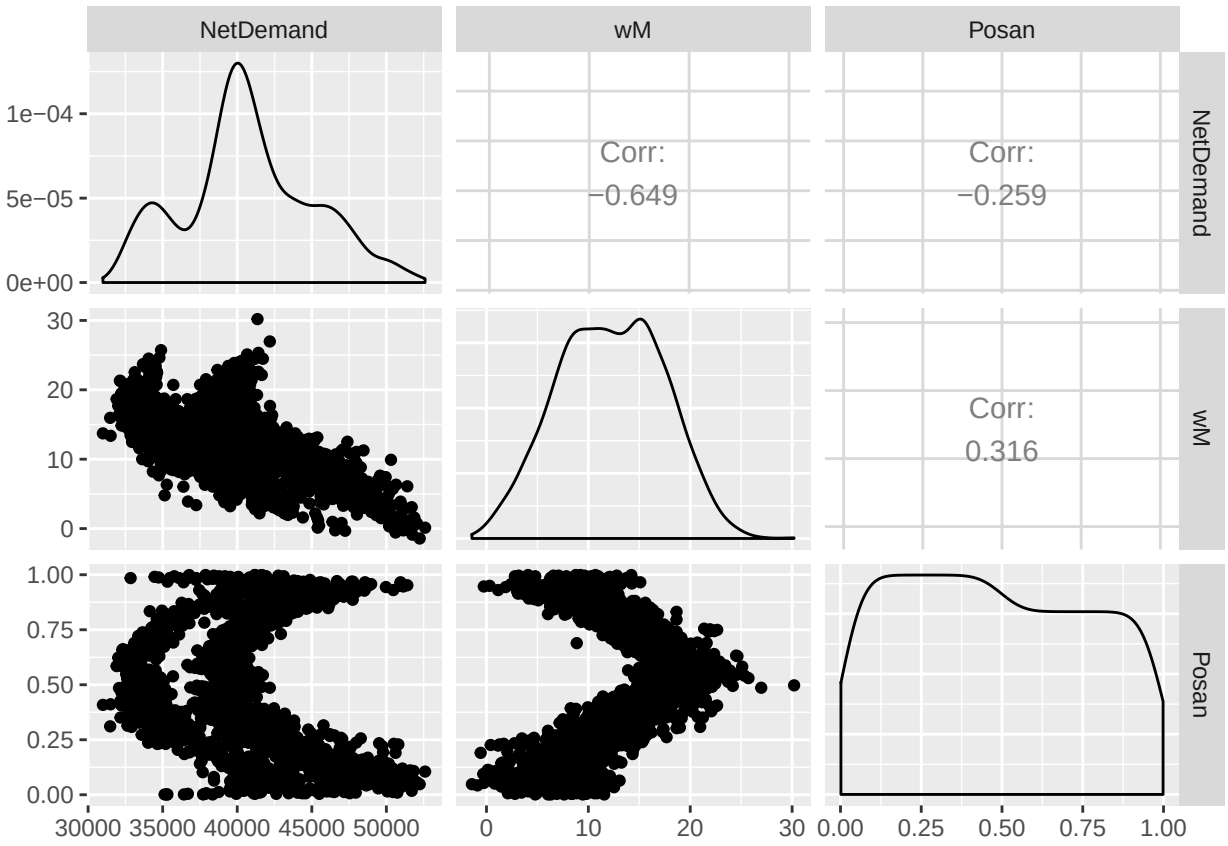
`select` is a good alternative to `subset` from base R.

```
library(GGally)
```

```
##
## Attaching package: 'GGally'
```

```
## The following object is masked from 'package:dplyr':
##
##     nasa
```

```
UKload %>% select(NetDemand, wM, Posan) %>%
           ggpairs() #plot pairs
```



Other useful functions include `mutate`, which can alter or add variables to a dataframe, and `summarise` which can provide summary statistics.

Reshaping with `tidyr` and `dplyr`

Data in a long format is needed for modelling and plotting. This is easily done with the `gather` function from `tidyr`. The opposite transformation is achieved by `spread`, outputting a tibble.

```
wideDat <- UKload %>% select(NetDemand, Date) %>% spread(key = Date, value = NetDemand)
head(wideDat[,1:10])
```

```
##   2011-01-01 12:00:00 2011-01-02 12:00:00 2011-01-03 12:00:00
## 1               38353               41192               43442
##   2011-01-04 12:00:00 2011-01-05 12:00:00 2011-01-06 12:00:00
## 1               50736               50438               50064
##   2011-01-07 12:00:00 2011-01-08 12:00:00 2011-01-09 12:00:00
## 1               51698               43988               43340
##   2011-01-10 12:00:00
## 1               50645
```

We might wish to merge mutliple data frames. This is possible using the `left_join` and `right_join` functions; see the lab sheet for a demonstration of these.

Combining into one `data.frame` rapidly expands the size of the file. We should consider compression techniques here *(SQL?)*.