dplyr & ggplot2

In the week 2 live session we use the libraries **dplyr** and **ggplot2**. These are two of the most important libraries available in R and are widely used for data analysis; dplyr for data manipulation and ggplot2 for data visualization. Both packages are part of the tidyverse collection of packages, supported by RStudio. This document gives just a few brief examples of functions from these packages, so please follow the links to learn more.

```
library(dplyr)
library(ggplot2)
```

dplyr

Dplyr's official guide describes it as a 'grammar' of data manipulation, involving various 'verbs' such as *filter*, *mutate* and *summarize* which can be applied to data frames and tables. These operations can be chained together using the pipe-operator, *%>%*.

The dplyr library comes with the **iris** data set which you may have seen in other data science courses. Using the *glimpse* function we can see that this data set contains 150 observations with five variables; four numeric (type double) and one factor variable.

```
glimpse(iris)
```

```
## Observations: 150
## Variables: 5
## $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4,...
## $ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7,...
## $ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5,...
## $ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2,...
## $ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa...
```

We can use the *sample_n* function to view a random sample of n observations.

```
sample_n(iris, 10)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1           6.4         2.7          5.3         1.9  virginica
## 2           5.0         2.3          3.3         1.0 versicolor
## 3           6.9         3.1          5.1         2.3  virginica
## 4           7.2         3.0          5.8         1.6  virginica
## 5           4.6         3.4          1.4         0.3     setosa
## 6           5.6         3.0          4.5         1.5 versicolor
## 7           5.4         3.0          4.5         1.5 versicolor
## 8           5.0         3.6          1.4         0.2     setosa
## 9           6.1         3.0          4.9         1.8  virginica
## 10          5.7         2.8          4.5         1.3 versicolor
```

Say you want to keep only observations belonging to the *virginica* species with a petal length greater than 5. The *filter* function can be applied to variables using one or more conditional statements.

```
iris_filtered <- filter(iris, Species == 'virginica', Petal.Length > 5)
glimpse(iris_filtered)
```

```
## Observations: 41
## Variables: 5
## $ Sepal.Length <dbl> 6.3, 5.8, 7.1, 6.3, 6.5, 7.6, 7.3, 6.7, 7.2, 6.5, 6.4,...
## $ Sepal.Width  <dbl> 3.3, 2.7, 3.0, 2.9, 3.0, 3.0, 2.9, 2.5, 3.6, 3.2, 2.7,...
## $ Petal.Length <dbl> 6.0, 5.1, 5.9, 5.6, 5.8, 6.6, 6.3, 5.8, 6.1, 5.1, 5.3,...
## $ Petal.Width  <dbl> 2.5, 1.9, 2.1, 1.8, 2.2, 2.1, 1.8, 1.8, 2.5, 2.0, 1.9,...
## $ Species      <fct> virginica, virginica, virginica, virginica, virginica,...
```

Here the comma acts as an *and* operator, but you can use *or* or *xor* (exclusive or) operators, for example:

```
glimpse(filter(iris, Species == 'virginica' | Petal.Length > 5))
```

```
## Observations: 51
## Variables: 5
## $ Sepal.Length <dbl> 6.0, 6.3, 5.8, 7.1, 6.3, 6.5, 7.6, 4.9, 7.3, 6.7, 7.2,...
## $ Sepal.Width  <dbl> 2.7, 3.3, 2.7, 3.0, 2.9, 3.0, 3.0, 2.5, 2.9, 2.5, 3.6,...
## $ Petal.Length <dbl> 5.1, 6.0, 5.1, 5.9, 5.6, 5.8, 6.6, 4.5, 6.3, 5.8, 6.1,...
## $ Petal.Width  <dbl> 1.6, 2.5, 1.9, 2.1, 1.8, 2.2, 2.1, 1.7, 1.8, 1.8, 2.5,...
## $ Species      <fct> versicolor, virginica, virginica, virginica, virginica...
```

The *select* function can be used to create a new data frame from selected columns of an existing dataframe, for example, a dataframe that just contains the species and the sepal length can be created with:

```
iris_selected <- select(iris, Sepal.Length, Species)
glimpse(iris_selected)
```

```
## Observations: 150
## Variables: 2
## $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4,...
## $ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa...
```

This function can also be used to drop columns from a data frame, using a minus sign in front of the variables you want to drop.

```
iris_selected <- select(iris, -Sepal.Width, -Petal.Length, -Petal.Width)
glimpse(iris_selected)
```

```
## Observations: 150
## Variables: 2
## $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4,...
## $ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa...
```

The *mutate* function allows you to create a new column from existing columns. For example, if you wanted to create a new variable that was the sum of sepal length and sepal width, you could do the following:

```
iris_mutated <- mutate(iris, Sepal.Length.Width = Sepal.Length + Sepal.Width)
glimpse(iris_mutated)
```

```
## Observations: 150
## Variables: 6
## $ Sepal.Length       <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9...
## $ Sepal.Width        <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1...
## $ Petal.Length       <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5...
## $ Petal.Width        <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1...
## $ Species            <fct> setosa, setosa, setosa, setosa, setosa, setosa, ...
## $ Sepal.Length.Width <dbl> 8.6, 7.9, 7.9, 7.7, 8.6, 9.3, 8.0, 8.4, 7.3, 8.0...
```

A very useful dplyr function is *group_by*. This allows you to group the data frame by some variable in order to perform another operation. It is commonly used with *summarise*, a function that lets you perform an operation (such as mean, median, sd, count, min, max or quantile) on the grouped data (I use *summarise* with the British spelling to avoid confusing it with a different *summarize* function from the *Hmisc* library). The *ungroup* function removes the grouping.

As an example, to show the mean sepal length for each of the three species in the iris data set:

```
iris_grouped <- group_by(iris, Species)
summarise(iris_grouped, Mean.Sepal.Length = mean(Sepal.Length))
```

```
## # A tibble: 3 x 2
##   Species    Mean.Sepal.Length
##   <fct>                  <dbl>
## 1 setosa                  5.01
## 2 versicolor              5.94
## 3 virginica               6.59
```

Grouped data can also be used with mutate, for example we can create an additional variable with the species mean with:

```
glimpse(mutate(iris_grouped, Mean.Sepal.Length = mean(Sepal.Length)))
```

```
## Observations: 150
## Variables: 6
## Groups: Species [3]
## $ Sepal.Length      <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9,...
## $ Sepal.Width       <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1,...
## $ Petal.Length      <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5,...
## $ Petal.Width       <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1,...
## $ Species           <fct> setosa, setosa, setosa, setosa, setosa, setosa, s...
## $ Mean.Sepal.Length <dbl> 5.006, 5.006, 5.006, 5.006, 5.006, 5.006, 5.006, ...
```

A nice feature of dplyr is the pipe-operator %>% which takes the argument on the left of the pipe and applies the function on the right side of the pipe (the keyboard shortcut for typing this pipe operator is Ctrl/Cmd+Shift+M). Using the pipe operator can simplify your code if you are performing a number of operations; it is easier to write the functions in sequence than to nest them inside each other.

For example, beginning with the original iris data set, let's filter to keep only observations with a petal length between 1 and 5, then from those observations create a new column being the sum of sepal length and sepal width, and then find the average of this combined sepal-length-sepal-width measure for each species.

```
iris %>% filter(between(Petal.Length,1,5)) %>%
  mutate(Sepal.Length.Width = Sepal.Length + Sepal.Width) %>%
  group_by(Species) %>%
  summarise(Mean.Sepal.Length.Width = mean(Sepal.Length.Width)) %>%
  ungroup()
```

```
## # A tibble: 3 x 2
##   Species    Mean.Sepal.Length.Width
##   <fct>                        <dbl>
## 1 setosa                        8.43
## 2 versicolor                    8.71
## 3 virginica                     8.57
```

These are just basic examples. There is a good introduction to dplyr vignette, and numerous tutorials available online (for example.) See also RStudio's data wrangling cheat sheet and Chapter 5: Data transformation from the online textbook R for Data Science

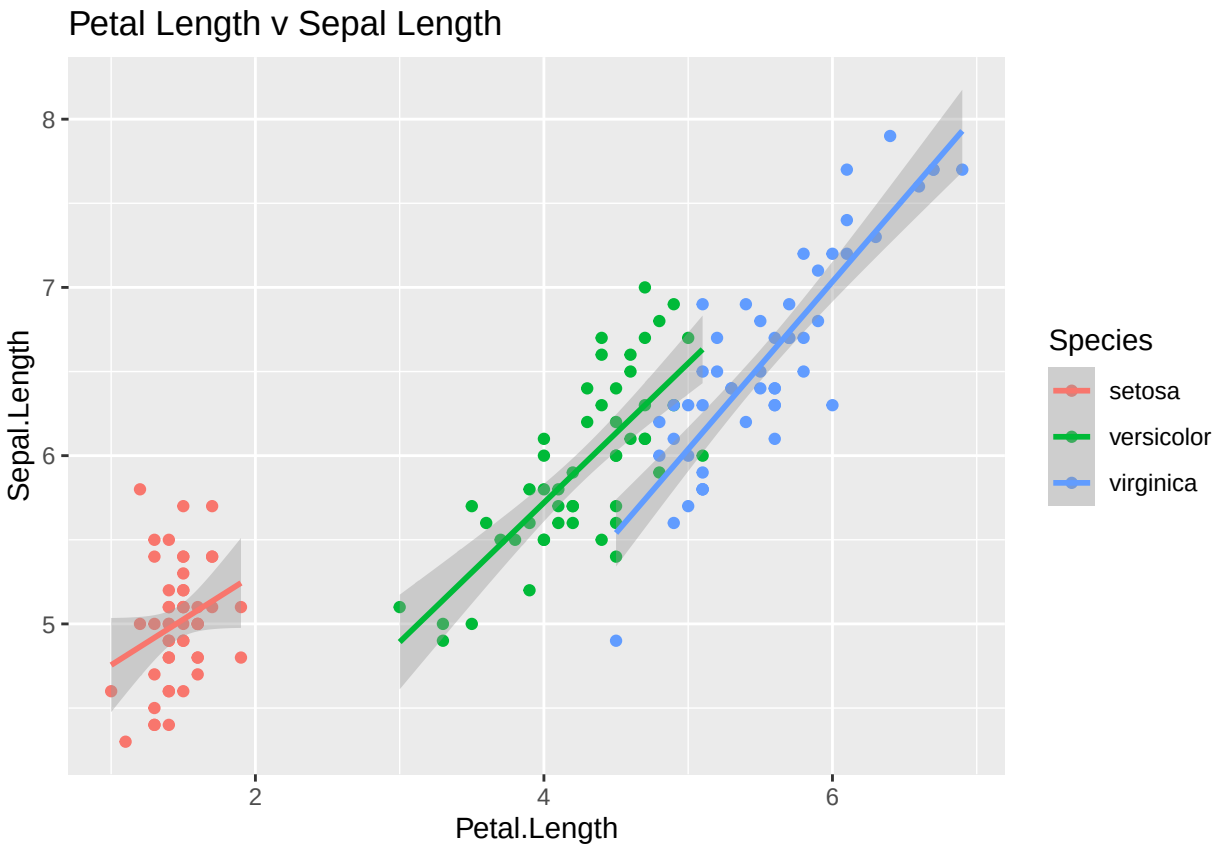
gpplot2

The standard plotting package in R is ggplot2, which gives you a wide range of options to visualize your data. ggplot2 has a syntax which starts with a ggplot function specifying the data and an *'aesthetic'* aes argument for the x variable, y variable and color/fill variable. You then literally add (+) features to your ggplot with *geom* objects for different types of graph features, e.g. geom_bar, geom_col, geom_line, geom_point. You can also define aesthetics inside geom objects.

To use the iris data set for example, we could create a scatterplot showing relationships between petal length and sepal length (with observations colored by species) by defining that aesthetic inside the ggplot function and then adding a **geom_point** geom. To also include linear regression lines through the three groups we can add a **geom_smooth** geom, specifying the model as 'lm' (the default is loess).

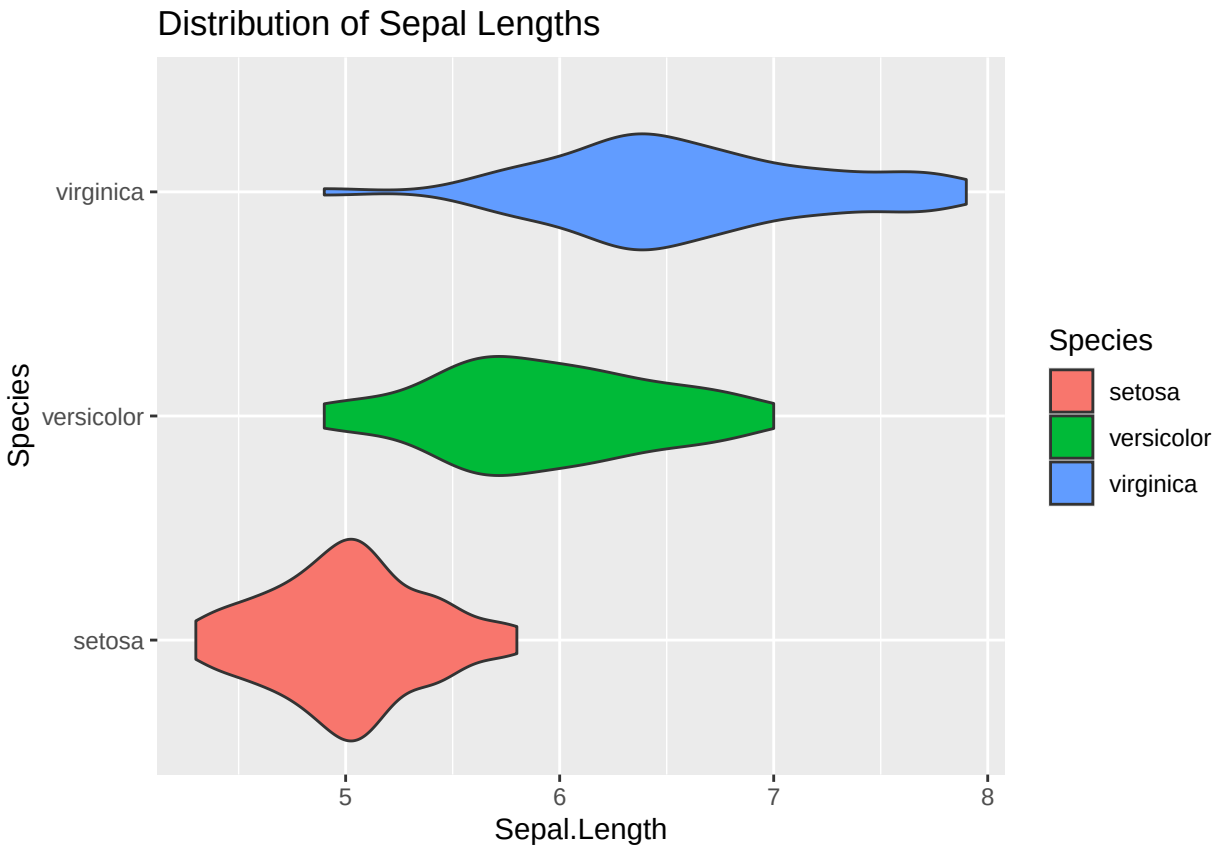
```
ggplot(iris, aes(x = Petal.Length, y=Sepal.Length, color=Species)) +
  geom_point() +
  geom_smooth(method = 'lm') +
  ggtitle('Petal Length v Sepal Length')
```

```
## `geom_smooth()` using formula 'y ~ x'
```



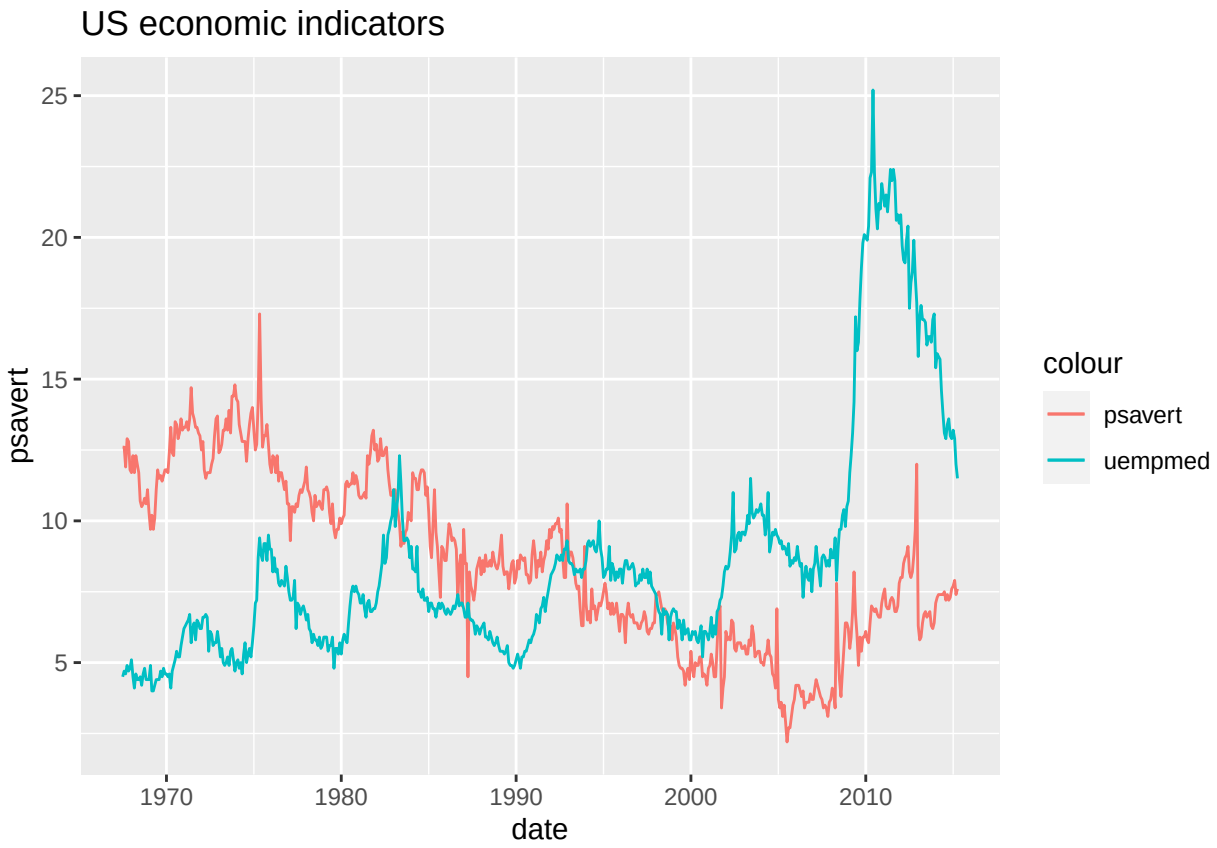
The x variable in your aesthetic can be a factor variable. The following uses **geom_violin** (an alternative to box plots or density plots) to show how the distribution of sepal length differs between the three species. The **coord_flip** option reorients graphs from vertical to horizontal, and can be added to geoms including box plots and histograms.

```
ggplot(iris, aes(x=Species, y=Sepal.Length, fill=Species)) +
  geom_violin() + coord_flip() +
  ggtitle('Distribution of Sepal Lengths')
```



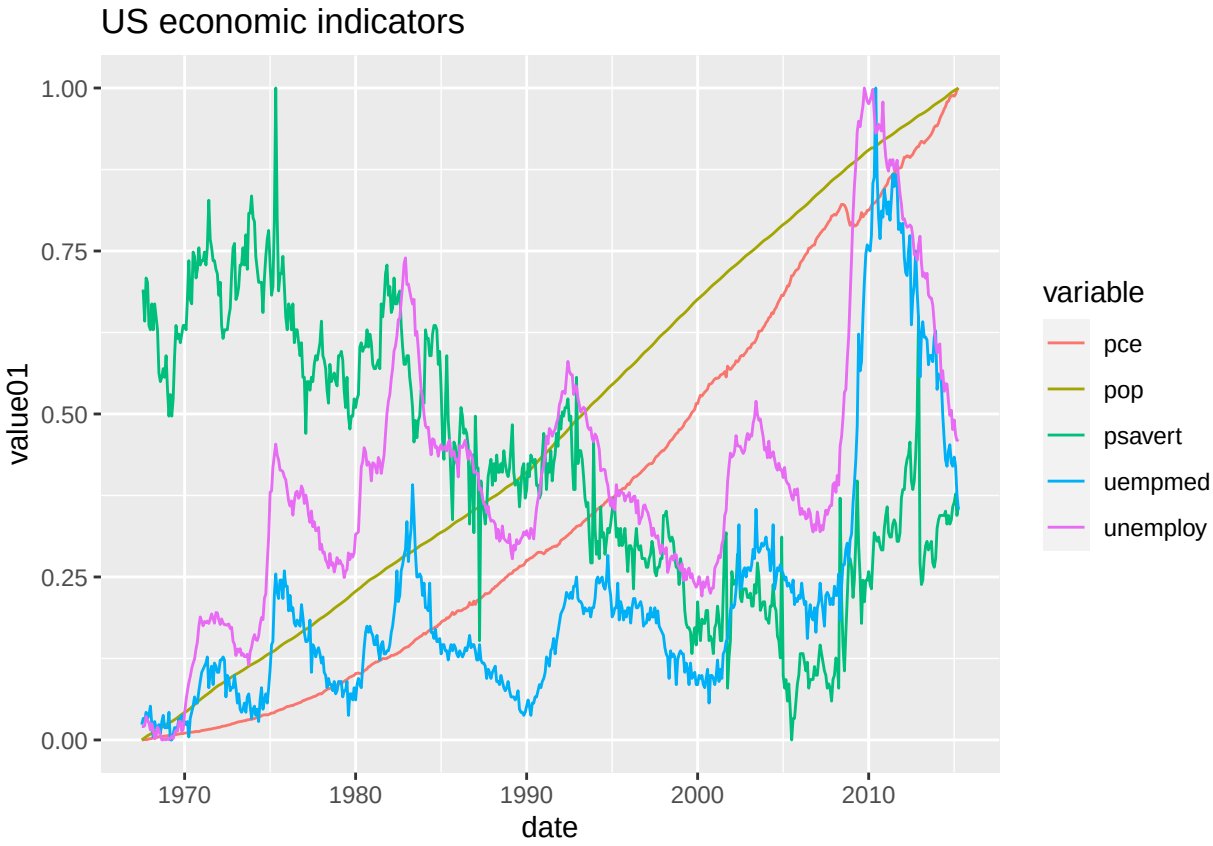
For plotting multivariate time series objects (for example the **economics** data set that comes with the ggplot2 library), the x axis (time index) can be specified in the ggplot aesthetic, while the individual series can be specified in the aesthetics of the geom_line objects (or geom_col if you prefer to mix lines and columns). Colors should specify the name of the variable as a string, otherwise a color scale will be used for the individual observations.

```
ggplot(economics, aes(x=date)) +
  geom_line(aes(y=psavert, color='psavert')) +
  geom_line(aes(y=uempmed, color='uempmed')) +
  ggtitle('US economic indicators')
```



To plot numerous series together on the same chart, code can be simplified by transforming your data from wide to long format. Another data set that comes with ggplot2 is **economics_long**, which is the same as **economics** but in 'long' format such that the observations are 'stacked' in key-value pairs (in this data set the key column is named *variable* and there are two value columns, the original *value* and the standardized *value01*).

```
ggplot(economics_long, aes(date, value01, colour = variable)) +
  geom_line() +
  ggtitle('US economic indicators')
```



For multivariate time series it can be convenient to transform your data from wide to long format. The *gather* function from the *tidyr* package (also part of the tidyverse) is a convenient way to switch between wide and long formats. For example

```
library(tidyr)
gather(economics, key= "variable", value= "value", -date)
```

```
## # A tibble: 2,870 x 3
##    date       variable value
##    <date>     <chr>    <dbl>
##  1 1967-07-01 pce       507.
##  2 1967-08-01 pce       510.
##  3 1967-09-01 pce       516.
##  4 1967-10-01 pce       512.
##  5 1967-11-01 pce       517.
##  6 1967-12-01 pce       525.
##  7 1968-01-01 pce       531.
##  8 1968-02-01 pce       534.
##  9 1968-03-01 pce       544.
## 10 1968-04-01 pce       544
## # ... with 2,860 more rows
```

The *melt* function works similarly.

The ggplot2 package offers a huge range of options for visualizing your data. Some useful ggplot2 resources are:

- R for Data Science Chapter 3: Data visualisation
- The official guide (includes cheatsheet)
- The R graph gallery
- The complete ggplot2 tutorial
- Using ggplot2 for functional time series