Visualising data in R
Graduate skills class OU22

Practical:
Monday 19 February, 2-3pm, SW01.
OR Wednesday 21 February, 3-4pm, SW01.
The *Grammar of Graphics* codifies some standard patterns in plotting data. It doesn’t try to be exhaustive, nor is it a guide to good style.

ggplot2 will simplify your life — if you learn the way it thinks, and if you don’t step outside its scope.
OU22: visualizing data with R

Aims

The R language is widely used by data scientists. It is concise and expressive for handling data, and it has a powerful graphics library called ggplot. This course will show you how to get data into R, how to do simple data manipulation, and how to use ggplot for flexible and versatile data visualization. See examples at the R graph gallery.

Resources

- Invoking R+ggplot from Jupyter/Python [notebook]
- Running R+ggplot in Jupyter/Python [notebook]
- Slides (to appear)
- Practical (to appear)

Installation instructions (Ubuntu, Windows)

On Windows, first follow Microsoft's instructions to install the Windows Subsystem for Linux, with Ubuntu 16.04.

Installing Jupyter and Python.

Install git, Python, and Jupyter. Ubuntu 16.04 uses Python 3.5 for many of its system scripts, so it's not a good idea to upgrade Python systemwide, so it's best to install Jupyter on top of Python 3.5:

```
sudo apt install python3-pip python3-dev git
dsud pip3 install --upgrade pip
ndsud pip3 install jupyter
```

Install Python 3.6 and make it available as a programming environment within Jupyter:

```
sudo add-apt-repository ppa:jonathonf/python-3.6
```
Python notebook + R plots

First, follow the installation instructions for Jupyter, Python, and R/Jupyter.

Warning. This doesn’t work from Azure notebooks.

```python
In [1]: `%capture`  
   # Jupyter is very chatty. If you start a cell with `%capture`, it suppresses all output  
   # (including any helpful warning or error messages).

   # Load the rpy2 extension for Jupyter.
   # (There will now be an R kernel sitting alongside the Python kernel.)
%load_ext rpy2.ipython

   # You can run a one-line R command with %r. Here, I'm loading a plotting library.
   # It will be available for the rest of this Jupyter session.
%r
library(ggplot2)
%r
library(ggthemes)

In [5]:
import urllib
import pandas
import numpy as np

   # Load in a dataset
iris = pandas.read_csv(urllib.request.urlopen('https://teachingfiles.blob.core.windows.net/founds/iris.csv'))

   # Have a quick Look
iris.iloc[np.random.choice(len(iris), 3, replace=False)]

Out[5]:

<table>
<thead>
<tr>
<th></th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>53</td>
<td>5.5</td>
<td>2.3</td>
<td>4.0</td>
<td>versicolor</td>
</tr>
<tr>
<td>1</td>
<td>67</td>
<td>5.8</td>
<td>2.7</td>
<td>4.1</td>
<td>versicolor</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>5.1</td>
<td>3.8</td>
<td>1.9</td>
<td>setosa</td>
</tr>
</tbody>
</table>

We can inject a Python object into R using -i. It will be copied across, and available in R with the same name.

```r
In [12]:
%r -i iris

You can’t copy arbitrary variables across from R to Python. The rpy2 package knows about some standard simple datatypes (including pandas DataFrames), and it is able to translate them, but if you copy across something weird it will fail.

```python
class X: pass
x = X()
```
Visualizing data with R

In [2]: # Load some packages for plotting and data manipulation.
library(ggplot2)
library(ggthemes)
library(data.table)

In [4]: # Set the plotting size (in inches)
options(repr.plot.width=4, repr.plot.height=2.5)

In [5]: # Fetch a dataset in CSV format
iris <- fread('https://teachingfiles.blob.core.windows.net/founds/iris.csv')
# Print out some sample rows
iris[sample(nrow(iris),4)]

<table>
<thead>
<tr>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>3.4</td>
<td>1.6</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>6.5</td>
<td>3.0</td>
<td>5.5</td>
<td>1.8</td>
<td>virginica</td>
</tr>
<tr>
<td>5.0</td>
<td>3.5</td>
<td>1.3</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>6.7</td>
<td>2.5</td>
<td>5.8</td>
<td>1.8</td>
<td>virginica</td>
</tr>
</tbody>
</table>

In [12]: iris[, shape := cut(Sepal.Width/(Sepal.Width+Sepal.Length), 2, labels=c('narrow','broad'))]

ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species), size=1.3, alpha=.8) +
  stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species), method='lm', size=.2) +
  facet_wrap(~Species) +
  scale_y_continuous(breaks=seq(0,10)) +
  gtitle('Iris features') +
  theme_bw()

Warning message in qt((1 - level)/2, df):
"NAs introduced by coercion"
The anatomy of a ggplot

```r
# Fetch a dataset in CSV format
iris <- fread('https://teachingfiles.blob.core.windows.net/founds/iris.csv')
# Print out some sample rows
iris[sample(nrow(iris), 4)]

<table>
<thead>
<tr>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>3.4</td>
<td>1.6</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>6.5</td>
<td>3.0</td>
<td>5.5</td>
<td>1.8</td>
<td>virginica</td>
</tr>
<tr>
<td>5.0</td>
<td>3.5</td>
<td>1.3</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>6.7</td>
<td>2.5</td>
<td>5.8</td>
<td>1.8</td>
<td>virginica</td>
</tr>
</tbody>
</table>

iris[, shape := cut(Sepal.Width/(Sepal.Width+Sepal.Length), 2, labels=c('narrow', 'broad'))]

ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species), size=1.3, alpha=.8) +
  stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species), method='lm', size=.2) +
  facet_wrap(~shape) +
  scale_y_continuous(breaks=seq(0,10)) +
  ggtitle('Iris features') +
  theme_bw()

Warning message in qt((1 - level)/2, df): "NAs produced"
```

The anatomy of a ggplot
Data comes in data frames.

ggplot2 is only for this sort of data (though it has handy functions for *fortifying* some other data types into data frames).
<table>
<thead>
<tr>
<th>LSOA_id</th>
<th>piece</th>
<th>seq</th>
<th>lat</th>
<th>lng</th>
<th>deprivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>007D</td>
<td>1</td>
<td>1</td>
<td>52.0</td>
<td>1.1</td>
<td>0.4</td>
</tr>
<tr>
<td>007D</td>
<td>1</td>
<td>2</td>
<td>52.3</td>
<td>1.05</td>
<td>0.4</td>
</tr>
<tr>
<td>007D</td>
<td>1</td>
<td>3</td>
<td>52.1</td>
<td>1.3</td>
<td>0.4</td>
</tr>
<tr>
<td>007D</td>
<td>2</td>
<td>4</td>
<td>52.1</td>
<td>1.2</td>
<td>0.4</td>
</tr>
</tbody>
</table>
```r
ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species),
             size=1.3, alpha=.8) +
  stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species),
              method='lm', size=.2) +
  facet_wrap(~shape) +
  scale_y_continuous(breaks=seq(0,10)) +
  ggtitle('Iris features') +
  theme_bw()
```
Data is plotted as geoms. A geom occupies part of a coordinate space.

A *geom_bar* occupies an interval of space, and is specified by its two ends (often one is zero). Where the bar is placed is another matter...

<table>
<thead>
<tr>
<th>gender</th>
<th>response</th>
<th>value</th>
<th>bar</th>
<th>bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>female</td>
<td>rarely</td>
<td>0.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>female</td>
<td>infrequently</td>
<td>0.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>female</td>
<td>occasionally</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>female</td>
<td>frequently</td>
<td>0.32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>female</td>
<td>not sure</td>
<td>0.32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>male</td>
<td>rarely</td>
<td>0.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>male</td>
<td>infrequently</td>
<td>0.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>male</td>
<td>occasionally</td>
<td>0.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>male</td>
<td>frequently</td>
<td>0.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>male</td>
<td>not sure</td>
<td>0.38</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A `geom_bar` occupies an interval of space, and is specified by its two ends (often one is zero). Where the bar is placed is another matter...
The same data can be represented by many different geoms.

- **Point**
- **Line**
- **Area**
- **Interval**
- **Path**
ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species),
             size=1.3, alpha=.8) +
  stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species),
              method='lm', size=.2) +
  facet_wrap(~shape) +
  scale_y_continuous(breaks=seq(0,10)) +
  ggtitle('Iris features') +
  theme_bw()
Data: geom.

**Example Code**:

```r
ggplot(data=iris) + geom_histogram(aes(x=Sepal.Length, fill=Species), bins=20) + facet_wrap(~Species)
```

This code uses `stat=bin` by default, but you can always override it.

```r
ggplot(data=iris) + geom_bar(aes(x=Sepal.Length, fill=Species), stat='bin', bins=20) + facet_wrap(~Species)
```

This code uses `stat=identity` by default, but you can always override it.

```r
ggplot(data=iris) + geom_area(aes(x=Sepal.Length, y=..count.., fill=Species, col=Species), stat='bin', bins=20, alpha=.5) + facet_wrap(~Species)
```

*Note*: A column generated by the stat is added to the dataframe before passing it to `geom-*`. The new dataframe may have different columns.

Data: geom. aes. facet. coord. guide.
An `aes` is the map from data value to value on some aesthetic scale.

e.g. size, hue, intensity
```r
ggplot(data=iris) + geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species), size=1.3, alpha=.8) + stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species), method='lm', size=.2) + facet_wrap(~shape) + scale_y_continuous(breaks=seq(0,10)) + ggtitle('Iris features') + theme_bw()
```
Each geom understands a specific set of aes.

x, y, size, label, shape/pch, fill, colour, alpha, linetype/lt

http://ggplot2.tidyverse.org/reference/
```r
ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species), size=1.3, alpha=.8) +
  stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species, method='lm', size=.2)) +
  facet_wrap(~shape) +
  scale_y_continuous(breaks=seq(0,10)) +
  ggtitle('Iris features') +
  theme_bw()
```

In this plot, these attributes are constants rather than `aes`.
An aes is the map from data value to value on some aesthetic scale.

- The output scale can be customized with `scale_*`
- It maps the entire range of data values in the entire graphic.

```r
ukmap <- fread('https://teachingfiles.blob.core.windows.net/datasets/uk_poly.csv')

ggplot(data=ukmap) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=as.numeric(id)), col='white', size=.1) +
  coord_fixed(ratio=1/cos(50*2*pi/360))
```

```r
ggplot(data=ukmap) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=as.numeric(id)), col='white', size=.1) +
  scale_fill_gradient2(midpoint=14000, high='forestgreen', low='darkblue') +
  coord_fixed(ratio=1/cos(50*2*pi/360))
```
An aes is the map from data value to value on some aesthetic scale.

- The output scale can be customized with `scale_*`
- It maps the entire range of data values in the entire graphic.

```r
ggplot(data=iris) + geom_point(aes(x=Sepal.Length, y=Sepal.Width, size=Petal.Length * Petal.Width, col=Species), alpha=.8) + scale_size_area()
```

Hey, the points are occluding each other. I need to make them smaller.

```r
ggplot(data=iris) + geom_point(aes(x=Sepal.Length, y=Sepal.Width, size=Petal.Length * Petal.Width / 10, col=Species), alpha=.8) + scale_size_area()
```

Hold on, the points are exactly the same size as before. WTF?

The data range is different, but scale training has mapped it to precisely the same visual range as before!

```r
ggplot(data=iris) + geom_point(aes(x=Sepal.Length, y=Sepal.Width, size=Petal.Length * Petal.Width, col=Species), alpha=.8) + scale_size_area(max_size=3, limits=c(0,NA))
```

I want to map data range `[0, max.in.data]` to the output range `[0, 3pt]`
An aes is the map from data value to value on some aesthetic scale.

- The output scale can be customized with scale_*
- It maps the entire range of data values in the entire graphic.

```r
fit <- lm(Petal.Length ~ Sepal.Length, data=iris)
df <- copy(iris)
df[, Petal.Length := simulate(fit)]
df <- df[sample(nrow(iris), 60, replace=FALSE)]
ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species), size=1.3, alpha=.8) +
  geom_point(data=df, aes(x=Sepal.Length, y=Petal.Length, col='sim', pch='sim')) +
  scale_y_continuous(breaks=seq(0,10)) +
  ggtitle('Iris features') +
  theme_bw()
```

This geom is drawn from iris data.

For this geom, we override the dataset, and use df instead.
An aes is the map from data value to value on some aesthetic scale.

- The output scale can be customized with `scale_*`
- It maps the entire range of data values in the entire graphic.

```
fit <- lm(Petal.Length ~ Sepal.Length, data=iris)
df <- copy(iris)
df[, Petal.Length := simulate(fit)]
df <- df[sample(nrow(iris), 60, replace=FALSE)]

ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species), size=1.3, alpha=.8) +
  geom_point(data=df, aes(x=Sepal.Length, y=Petal.Length, col="sim", pch="sim")) +
  scale_y_continuous(breaks=seq(0,10)) +
  ggtitle('Iris features') +
  theme_bw()
```

Training phase:
- What data values do we see for `col`, across the entire plot?
  - "setosa" "versicolor" "virginica" "sim"

Scale phase:
- The data has 4 distinct string values, so we'll use a discrete colour scale by default
  - This is called the "blending trick"
An aes is the map from data value to value on some aesthetic scale.

- The output scale can be customized with scale_*
- It maps the entire range of data values in the entire graphic.

I have two nice plots. What if I combine them?

```r
ggplot() +
  geom_point(data=iris[Species!='setosa'], aes(x=Sepal.Length, y=Sepal.Width, col=Species))
```

```r
ggplot() +
  geom_point(data=iris[Species=='setosa'], aes(x=Sepal.Length, y=Sepal.Width, col= Petal.Length*Petal.Width))
```

This produces an error when it's training the scale. (In other situations it might coerce num->string and show you a discrete scale with thousands of levels.)
An aes is the map from data value to value on some aesthetic scale.

- The output scale can be customized with scale_*
- It maps the entire range of data values in the entire graphic.

How would you specify the y aes / scale for this plot?

You can't! If you use aes(y = precipitation) & aes(y = Humidity)
ggplot will simply construct a single y scale for all the values.
```r
ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species, pch=Species),
             size=1.3, alpha=.8) +
  stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species),
              method='lm', size=.2) +
  facet_wrap(~shape) +
  scale_y_continuous(breaks=seq(0,10)) +
  ggtitle('Iris features') +
  theme_bw()
```

Question. What scale am I customizing in this plot?

Answer. None! The `scale_y_continuous()` command can be used to customize the scale, e.g. by setting lower and upper ranges (and any data outside those ranges will be mapped to "missing").

But here I'm not controlling the map itself, only how it's displayed.
A faceted plot shows several panels, each containing a subset of the data.
A facetted plot shows several panels, each containing a subset of the data.

Some things you can’t do with facets:

The Grammar of Graphics lets you specify how facets are arranged — but ggplot2 is limited to simple rectangular arrangements.

This is a visual arrangement, not a data arrangement. It’s outside the scope of the Grammar of Graphics. (Hack around with gridExtra instead.)

The Grammar of Graphics doesn’t go this far (but it should).
The coordinate system says how x and y aesthetic values should be located on the display.

```r
ggplot(data=survey) + geom_bar(aes(x=response, y=value, fill=response, col=gender), stat='identity', scale_fill_discrete(guide=FALSE) + theme_grey(base_size=10) + theme(axis.text.x=element_text(angle=-45, hjust=0))
```

<table>
<thead>
<tr>
<th>gender</th>
<th>response</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>female</td>
<td>rarely</td>
<td>0.08</td>
</tr>
<tr>
<td>female</td>
<td>infrequently</td>
<td>0.11</td>
</tr>
<tr>
<td>female</td>
<td>occasionally</td>
<td>0.17</td>
</tr>
<tr>
<td>female</td>
<td>frequently</td>
<td>0.32</td>
</tr>
<tr>
<td>female</td>
<td>not sure</td>
<td>0.32</td>
</tr>
<tr>
<td>male</td>
<td>rarely</td>
<td>0.30</td>
</tr>
<tr>
<td>male</td>
<td>infrequently</td>
<td>0.15</td>
</tr>
<tr>
<td>male</td>
<td>occasionally</td>
<td>0.10</td>
</tr>
<tr>
<td>male</td>
<td>frequently</td>
<td>0.07</td>
</tr>
<tr>
<td>male</td>
<td>not sure</td>
<td>0.38</td>
</tr>
</tbody>
</table>

At each x, there are two rows in the table. 
- Two overlapping geoms
The coordinate system says how $x$ and $y$ aesthetic values should be located on the display.

ggplot(data=survey) +
  geom_bar(aes(x=response, y=value, fill=response, col=gender), stat='identity', position='stack') +
  scale_fill_discrete(guide=FALSE) +
  theme_grey(base_size=8) +
  theme(axis.text.x=element_text(angle=-45, hjust=0))

position_identity  position_stack  position_dodge
The coordinate system says how $x$ and $y$ aesthetic values should be located on the display.

These are both a stacked bar chart!

- use $x$ for radius
- use $y$ for angle

data. geom. aes. facet. **coord.** guide.
The Building Blocks of the Grammar of Graphics:
The Grammar of Graphics is a declarative language for building common data plots. It’s great for exploration. It’s not great for fine art, for animation, for interaction.

ggplot(data=...) 
geom_point(data=...) 
stat_smooth()  # transforms the data 
geom_bar()  # automatically applies stat_count

guide

scale_x_continuous(breaks=...) 
scale_fill_gradient2(guide=FALSE) 
theme()  gglabell()  ggsave()