R’s Grammar of Graphics codifies some standard patterns in plotting data. It will simplify your life—if you learn the way it thinks, and if you don’t step outside its scope.

Lecture: high-level concepts in ggplot
Practical: how to actually use it
rhetoric = grammar + style + reason / arrangement

The Visual Display of Quantitative Information
EDWARD R. TUFTE

R + ggplot2 Javascript + D3 Vega Lite and many many badly conceived libraries ...
First get Jupyter+Python+R up and running

You can run arbitrary R cells in the Jupyter notebook. If a line starts with `%%R`, that line is run in R. If a cell starts with `%%`, the entire cell is run in R. The `%%R` magic has some optional arguments, e.g. to set the size of the graphic that R might be asked to generate.

In [35]:

```R
%%R -w6 -h3 -uin -r120

ggplot(data=iris) +
geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species), size=3, alpha=stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species), method='lm') +
  scale_y_continuous(breaks=seq(0,10)) +
  facet_wrap(~Petal.size) +
  ggtitle('Iris features') +
  theme_bw()
```

Iris features

- **Species**: setosa, versicolor, virginica

- **Features**:
  - **Sepal.Length**
  - **Petal.Length**
You can run arbitrary R cells in the Jupyter notebook. If a line starts with `%R`, that line is run in R. If a cell starts with `%%%R`, the entire cell is run in R. The `%%%R` magic has some optional arguments, e.g. to set the size of the graphic that R might be asked to generate.

```r
In [35]:

   %R -w6 -h3 -uin -r120

ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Petal.Length, col=Species), size=3, alpha=
  stat_smooth(aes(x=Sepal.Length, y=Petal.Length, col=Species), method='lm')
  scale_y_continuous(breaks=seq(0,10)) +
  facet_wrap(~petal.size) +
  ggtitle('Iris features') +
  theme_bw()
```

**Iris features**

- **Species**
  - setosa
  - versicolor
  - virginica

- **Features**
  - Sepal.Length
  - Petal.Length
Data comes in data frames.

ggplot2 is only for this sort of data.

```
library(ggplot2)

ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Petal.Length))
```
The *aesthetic mapping* specifies which data columns should be mapped to which visual dimensions.

```r
ggplot(data=iris) +
  geom_point(aes(x=Sepal.Width, y=Sepal.Length, col=Species, shape=Species))
```

Species is a discrete (string) value, so the default scale is discrete.

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

```r
ggplot(data=iris) +
  geom_point(aes(x=Sepal.Width, y=Sepal.Length, size=Petal.Length*Petal.Width), alpha=.4)
```
Exercise. What is the aesthetic mapping?
The **aesthetic mapping** specifies which data columns should be mapped to which visual dimensions

- The entire range of data values is mapped onto the visual range, which can be configured with `scale_*`

```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))

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)) +
  scale_fill_gradient2(midpoint=14000, high='forestgreen', low='darkblue')
```
Color Brewer: sequential / diverging / qualitative scales, for discrete data
- The **aesthetic mapping** specifies which data columns should be mapped to which visual dimensions.
- The entire range of data values is mapped onto the visual range, which can be configured with `scale_*`

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

# Original ggplot code
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))

# Using scale_fill_gradient2
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))

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

**Error:** continuous values supplied to a discrete scale

```r
# Using scale_fill_brewer
ggplot(data=ukmap) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=as.numeric(id)), col='white', size=.1) +
  scale_fill_brewer(type='qual') +
  coord_fixed(ratio=1/cos(50*2*pi/360))
```
Examples of colour scales

Religious adherents as a percentage of all residents, 2010

Alex Egoshin
www.vividmaps.com
Data source: ARDA
DATASET: total column density of ozone above the southern hemisphere (Why Should Engineers and Scientists Be Worried About Color? Rogowitz and Trienish, 1998)
(a) rainbow palette
(b) brightness palette
(c) divergent hue palette
(d) combines (b) and (c)
- The **aesthetic mapping** specifies which data columns should be mapped to which visual dimensions.
- The entire range of data values is mapped onto the visual range, which can be configured with `scale_*`
  which is a common source of confusion.

```r
ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Sepal.Width, size=Petal.Length * Petal.Width / 10, col=Species)) +
  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)) +
  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)) +
  scale_size_area(max_size=3, limits=c(0,NA))
```

I really want to map data range [0, max.in.data] to the output range [0, 3pt]
- The **aesthetic mapping** specifies which data columns should be mapped to which visual dimensions.
- The entire range of data values is mapped onto the visual range, which can be configured with `scale_*`

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

# Plot both iris and the synthetic dataset
ggplot() +
  geom_point(data=iris, aes(x=Sepal.Length, y=Petal.Length, col=Species, shape=Species)) +
  geom_point(data=df, aes(x=Sepal.Length, y=Petal.Length, col='sim', shape='sim'))
```

**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.

![Plot of Iris dataset with synthetic data]
The **aesthetic mapping** specifies which data columns should be mapped to which visual dimensions.

The entire range of data values is mapped onto the visual range, which can be configured with `scale_*`.

---

### Example Code

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

# Plot both iris and the synthetic dataset
ggplot() +
  geom_point(data=iris, aes(x=Sepal.Length, y=Petal.Length, col=Species, shape=Species)) +
  geom_point(data=df, aes(x=Sepal.Length, y=Petal.Length, col='sim', shape='sim'))

ggplot(data=iris, aes(x=Sepal.Length, y=Petal.Length)) +
  geom_point(aes(col=Species, shape=Species)) +
  geom_point(data=df, aes(col='sim', shape='sim'))
```

---

**Syntactic sugar:**

Plot specs can be set in `ggplot()`, and they become defaults for the plot layers.
- The **aesthetic mapping** specifies which data columns should be mapped to which visual dimensions
- The entire range of data values is mapped onto the visual range, which can be configured with `scale_*`

```
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))

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

This produces an error when it’s training the scale.
(In other situations it might coerce `num` to string and show you a discrete scale with thousands of levels.)
The **aesthetic mapping** specifies which data columns should be mapped to which visual dimensions.

- The entire range of data values is mapped onto the visual range, which can be configured with `scale_*`

---

**How would you set up the y axis for this plot?**

**You can't!** If you use

```r
aes(y = precipitation) & aes(y = humidity)
```

`ggplot` will simply construct a single y scale for all the values.
Components of a chart

- **data**
  - age
  - income
  - lat, lng
- **stats**
- **transform**
- **aesthetic attributes**
  - colour, fill, alpha, thickness, size
- **geometrical object**
  - $x, y$
- **positioning**
• A geom is an object that is plotted, occupying part of the coordinate space
• A stat is a transformation of the data
• Each geom comes with a default stat (sometimes just stat='identity')
  Some stats come with a default aes

```
# Example

ggplot(data=iris) +
  geom_bar(aes(x=Sepal.Length, y=..count..), col='blue', fill='cornflowerblue', stat='bin', bins=37)

ggplot(data=iris) +
  geom_bar(aes(x=Sepal.Length), col='blue', fill='cornflowerblue')
```
- A geom is an object that is plotted, occupying part of the coordinate space
- A stat is a transformation of the data
- Each geom comes with a default stat (sometimes just stat='identity')
  Some stats come with a default aes

So you can easily change how the data is depicted

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

ggplot(data=iris) +
  geom_area(aes(x=Sepal.Length, y=..count..), stat='bin', bins=20)

ggplot(data=iris) +
  geom_line(aes(x=Sepal.Length, y=..count..), stat='bin', bins=20) +
  scale_y_continuous(limits=c(0,NA))

ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=..count..), stat='bin', bins=20) +
  scale_y_continuous(limits=c(0,NA))
```
A geom is an object that is plotted, occupying part of the coordinate space
A stat is a transformation of the data
Each geom comes with a default stat (sometimes just stat='identity')

Some stats come with a default aes

which is usually helpful, but sometimes bothersome —
If the stat stuff gets confusing, just give up and preprocess the data yourself.

```r
# Do my own version of stat_bin (group the data by Sepal.Length, and get counts)
df = as.data.table(iris)[, list(Sepal.Length=mean(Sepal.Length), count=.N), by=cut(Sepal.Length, breaks=20)]

ggplot(data=df) +
  geom_segment(aes(x=Sepal.Length, xend=Sepal.Length, y=0, yend=count), stat='bin', bins=20)
```

```r
ggplot(data=iris) +
  geom_segment(aes(x=Sepal.Length, xend=Sepal.Length, y=0, yend=..count..), stat='bin', bins=20)
```

Error: stat_bin() must not be used with a y aesthetic.

```r
# Do my own version of stat_bin (group the data by Sepal.Length, and get counts)
df = as.data.table(iris)[, list(Sepal.Length=mean(Sepal.Length), count=.N), by=cut(Sepal.Length, breaks=20)]

ggplot(data=df) +
  geom_segment(aes(x=Sepal.Length, xend=Sepal.Length, y=0, yend=count), arrow=arrow(length=unit(0.03, 'npc')))

ggplot(data=df) +
  geom_rect(aes(xmin=Sepal.Length-0.05, xmax=Sepal.Length+0.05, ymin=0, ymax=count))
```
A histogram is just a geom_bar with a stat_bin.

Think in terms of combining simple geoms and stats, and you’ll be able to create an endless variety of charts, without having to learn a taxonomy.

But... ggplot2 provides a confusing taxonomy of geoms and stats! Happily you don’t need to remember them, because they are mostly just groupings of simpler geoms, with sensible defaults for stat.

```r
geom_histogram = geom_bar with stat=count
stat_smooth = geom_ribbon + geom_line with stat=smooth
```

We often call graphics charts (from χάρτης or Latin charta, a leaf of paper or papyrus). There are pie charts, bar charts, line charts, and so on. This book shuns chart typologies. For one thing, charts are usually instances of much more general objects. Once we understand that a pie is a divided bar in polar coordinates, we can construct other polar graphics that are less well known. We will also come to realize why a histogram is not a bar chart and why many other graphics that look similar nevertheless have different grammars.

There is also a practical reason for shunning chart typology. If we endeavor to develop a charting instead of a graphing program, we will accomplish two things. First, we inevitably will offer fewer charts than people want. Second, our package will have no deep structure. Our computer program will be unnecessarily complex, because we will fail to reuse objects or routines that function similarly in different charts. And we will have no way to add new charts to our system without generating complex new code. Elegant design requires us to think about a theory of graphics, not charts.

A chart metaphor is especially popular in user interfaces. The typical interface for a charting program is a catalog of little icons of charts. This is easy to construct from information gathered in focus groups, surveys, competitive analysis, and user testing. Much more difficult is to understand what users intend to do with their data when making a graphic. Instead of taking this risk, most charting packages channel user requests into a rigid array of chart types. To atone for this lack of flexibility, they offer a kit of post-creation editing tools to return the image to what the user originally envisioned. They give the user an impression of having explored data rather than the experience.

Some Useful stats + geoms:

```r
# First plot
ggplot(data=iris, aes(x=Sepal.Length, y=Petal.Length, col=Species)) +
  geom_point(aes(shape=Species)) +
  stat_smooth(method='loess')

# Second plot
ggplot(data=iris, aes(x=Sepal.Length, y=Petal.Length, col=Species)) +
  geom_ribbon(aes(group=Species), stat='smooth', method='loess', size=.2, fill='grey75', col=NA) +
  geom_line(stat='smooth', method='loess') +
  geom_point(aes(shape=Species), size=1)
```
Some useful stats + geoms:

```r
# Boxplot
ggplot(data=iris) +
  geom_boxplot(aes(x=Species, y=Sepal.Length))

# Violin plot
ggplot(data=iris) +
  geom_violin(aes(x=Species, y=Sepal.Length))

# Area and line plots
ggplot(data=iris) +
  geom_area(aes(x=Sepal.Length, y=..density.., fill=Species), position='identity', stat='bin', alpha=0.4, bins=20) +
  geom_line(aes(x=Sepal.Length, y=..density.., col=Species), stat='bin', bins=20)
```
DATASET: Spatial navigation ability, measured in a computer game (Global determinants of navigation ability, Coutrot et al. 2017)

2.5 million subjects were shown a map with waypoints, then asked to visit the waypoints, then shoot a flare at their start point.

OP = Overall Performance (path duration, path length, shooting accuracy, combined using PCA)

CM = Conditional Modes (overall performance compared to global average)

Exercise: what are the geoms in this plot?
Components of a chart

- **data**: age, income, lat, lng
- **stats/transform**: smooth, bin, count
- **aesthetic attributes**: $x, y$, colour, fill, alpha, thickness, size
- **geometrical object**: point, line, polygon
- **positioning**
A facetted plot shows several panels, each containing a subset of the data

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

```
# Create two categorical (i.e. string) columns, by binning Sepal.Width and Sepal.Length into buckets
iris[, Sepal.Width.f := cut(Sepal.Width, 2, labels=c('narrow', 'broad'))]
iris[, Sepal.Length.f := cut(Sepal.Length, 3, labels=c('short', 'med', 'long'))]
```

```
ggplot(data=iris) +
  geom_point(aes(x=Petal.Width, y=Petal.Length, col=Species, shape=Species)) +
  facet_grid(Sepal.Width.f ~ Sepal.Length.f)
```
Some of the things you can’t do with facets

This is a type of faceting — but ggplot2 has only implemented 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).
Exercise: control which axes are shared between facets, and adjust the size of each facet according to how much y-range it spans.
**DATASET:** medical data for 10 patients was processed by 4 machine learning algorithms, and each algorithm was scored for prediction accuracy.

<table>
<thead>
<tr>
<th>patient ID</th>
<th>machine learning algorithm</th>
<th>accuracy score</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2</td>
<td>lasso</td>
<td>0.2282353</td>
</tr>
<tr>
<td>p3</td>
<td>owl</td>
<td>0.2797059</td>
</tr>
<tr>
<td>p3</td>
<td>crl</td>
<td>0.1970588</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

```
ggplot(data=scans) +
  geom_bar(aes(x=algorithm, fill=algorithm, y=accuracy), stat='identity') +
  facet_grid(~patient)
```

```
ggplot(data=scans) +
  geom_line(aes(x=algorithm, y=accuracy, group=patient, col=patient))
```
Each object in a geom has values for its $x$ and $y$ scales. The coordinate system says how $x$ and $y$ should be located on the display.

The displayed position of a geom can be tweaked, to deal with overlap.

John Snow, 1854
Each object in a geom has values for its $x$ and $y$ scales. The coordinate system says how $x$ and $y$ should be located on the display.

The displayed position of a geom can be tweaked, to deal with overlap.

**DATASET:** A survey of U.S. scholars (Morton and Price, 1989).
Surveyed were 5,385. Respondents numbered 3,835. Respondents answered the question “How often, if at all, do you think the peer review refereeing system for scholarly journals in your field is biased in favour of males?”

<table>
<thead>
<tr>
<th></th>
<th>rarely</th>
<th>infrequently</th>
<th>occasionally</th>
<th>frequently</th>
<th>not sure</th>
</tr>
</thead>
<tbody>
<tr>
<td>male respondents</td>
<td>851</td>
<td>426</td>
<td>284</td>
<td>199</td>
<td>1078</td>
</tr>
<tr>
<td>female respondents</td>
<td>80</td>
<td>110</td>
<td>170</td>
<td>319</td>
<td>319</td>
</tr>
</tbody>
</table>

```r
ggplot(data=survey) + geom_bar(aes(x=response, y=value, col=gender, fill=gender), stat='identity', position='identity', alpha=.4)
```
- Each object in a geom has values for its $x$ and $y$ scales
  The coordinate system says how $x$ and $y$ should be located on the display
- The displayed position of a geom can be tweaked, to deal with overlap

```r
# Example code

ggplot(data=survey) + geom_bar(aes(x=response, y=value, col=gender, fill=gender), stat='identity', position='identity', alpha=.4)

ggplot(data=survey) + geom_bar(aes(x=response, y=value, col=gender, fill=gender), stat='identity', position='dodge', alpha=.4)

ggplot(data=survey) + geom_bar(aes(x=response, y=value, col=gender, fill=gender), stat='identity', position='stack', alpha=.4)
```
Each object in a geom has values for its $x$ and $y$ scales. The coordinate system says how $x$ and $y$ should be located on the display.

The displayed position of a geom can be tweaked, to deal with overlap.

ggplot(data=survey) + geom_bar(aes(x=gender, y=value, col=gender, fill=response), stat='identity', position='stack') + scale_fill_brewer()

These are all stacked bar charts—just in different coordinate systems!
- Each object in a geom has values for its $x$ and $y$ scales. The coordinate system says how $x$ and $y$ should be located on the display.
- The displayed position of a geom can be tweaked, to deal with overlap.

```r
ggplot(data=iris, aes(x=Species, y=Sepal.Length)) + geom_violin() + geom_point(alpha=.6)

ggplot(data=iris, aes(x=Species, y=Sepal.Length)) + geom_violin() + geom_point(position=position_jitter(width=.1, height=0), col='cornflowerblue', alpha=.3)
```
Canadian designer Kamel Makhloifi’s pair of stark graphs visualize the human toll of the Iraq war. Each pixel represents a death.
https://www.flickr.com/photos/melkaone/5121285002/
Components of a chart

- **Data**: age, income, lat, lng
- **Stats**: smooth, bin, count
- **Transform**: linear, log, etc.
- **Aesthetic Attributes**: $x, y$, colour, fill, alpha, thickness, size
- **Geometrical Object**: point, line, polygon
- **Positioning**: dodge, jitter, stack
  - Cartesian, polar
  - Subplot

Legend:
- Satosa
- Versicolor
- Virginica
- 4, 8, 12
- Colour scale: 13950, 14000, 14050, 14100
Components of a chart

**data**

- `ggplot(data=...)`

**stats transform**

- `stat_smooth()`
- `geom_bar()` # default `stat_count`

**aesthetic attributes**

- `aes(x=..., fill=...)`
- `scale_fill_gradient2()`

**geometrical object**

- `geom_bar()` # default `geom_ribbon`
- `stat_smooth()` # default `geom_ribbon`

**positioning**

- `facet_wrap(~...)`
- `facet_grid(~...)`
- `position_dodge()`, `geom_bar(position='dodge')`
- `coord_fixed(ratio=0.4)`
- `coord_cartesian(xlim, ylim)`
- `coord_polar(theta='y')`

**guides**

endless tweaks to the visuals...
- Apply styling with theme(). Beautiful code = ugly plot, ugly code = beautiful plot.
- Modify the guide (i.e. ticks or legend) for a scale
  - `scale_y_continuous(breaks=..., labels=...)`
  - `scale_colour_discrete(..., guide=FALSE)`
  - `guides(colour = guide_legend(...), size=FALSE)`
- Name the guides
  - `labs(x="", title=...)`
  - `scale_colour_discrete(..., name=...)`

```r
ggplot(data=survey) +
  geom_bar(aes(x=response, y=value, col=gender, fill=gender), stat='identity', position='dodge', alpha=.4)

g <- ggplot(data=survey) +
  geom_bar(aes(x=response, y=value, col=gender, fill=gender), stat='identity', position='dodge', alpha=.4) +
  scale_y_continuous(breaks=c(0,250,500,750,1000)) +
  guides(colour = guide_legend(override.aes=list(alpha=1))) +
  labs(x="", y="", title="Number of responses") +
  theme_economist() +
  theme(plot.background = element_rect(color=NA, fill="transparent"),
        panel.background = element_rect(color=NA, fill="grey90"),
        legend.background = element_rect(color=NA, fill="transparent"),
        axis.text.x = element_text(angle=-45, hjust=0))
ggsave(g, file='~/winhome/Downloads/myplot.png', bg='transparent', width=3, height=3)
```