Making the complex simple in data viz

Tania Vasilikioti
Data Scientist in Product Analytics, Babbel

PyConDE & PyData Berlin 2019

Why I’m here today

- Data Scientist @Babbel (language learning app)
- Communicate daily with technical and non technical people about our learners’ data, using graphics 👩‍🎓
- Often the picture is complex! 👩‍🎓👨‍🎓💤👨‍🎓👨‍🎓👩‍🎓💀👨‍🎓

Key challenge: How do make a chart that is complex in meaning yet concise and reusable in code?

Today's menu 

I. The (Layered) Grammar of Graphics
II. Implementations in Python
III. *The Grammar* as a mindset (aka DIY)

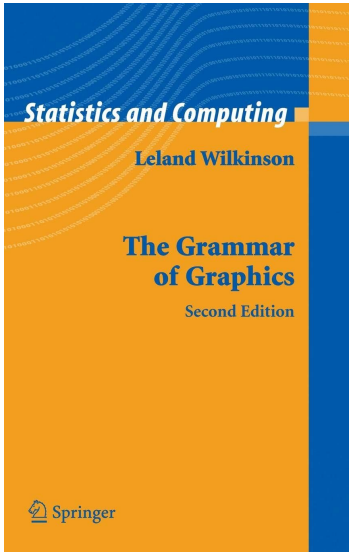
PART I: Grammar of Graphics primer

A Grammar of Graphics

"A **grammar of graphics** is a **tool** that enables us to concisely **describe** the **components** of a **graphic**" - H. Wickham

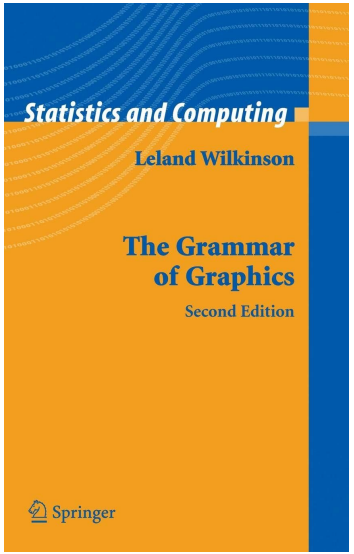
The Grammar of Graphics: origins

- The Grammar of Graphics, L. Wilkinson, 2006 (693 pages)
- Not a package!
- Rather academic work: “this book focuses on the deep structures involved in producing quantitative graphics from data.”



The Grammar of Graphics: origins

- The Grammar of Graphics, L. Wilkinson, 2006 (693 pages)
- Not a package!
- Rather academic work: “this book focuses on the deep structures involved in producing quantitative graphics from data.”



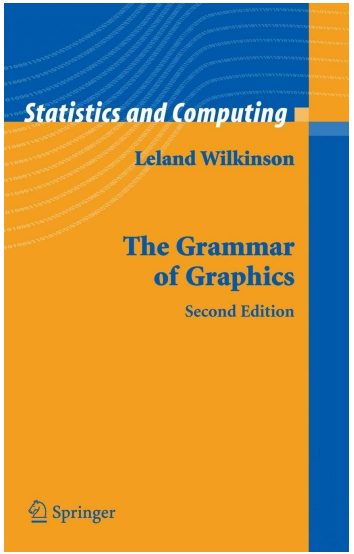
The components:

1. Data
2. Transformation
3. Element
4. Scale
5. Guide
6. Coordinates

The Grammar of Graphics: origins

- The Grammar of Graphics, L. Wilkinson, 2006 (693 pages)
- Not a package!
- Rather academic work: “this book focuses on the deep structures involved in producing quantitative graphics from data.”

Key takeaway: What is a graphic? What are the rules that underlie the production of various graphics?

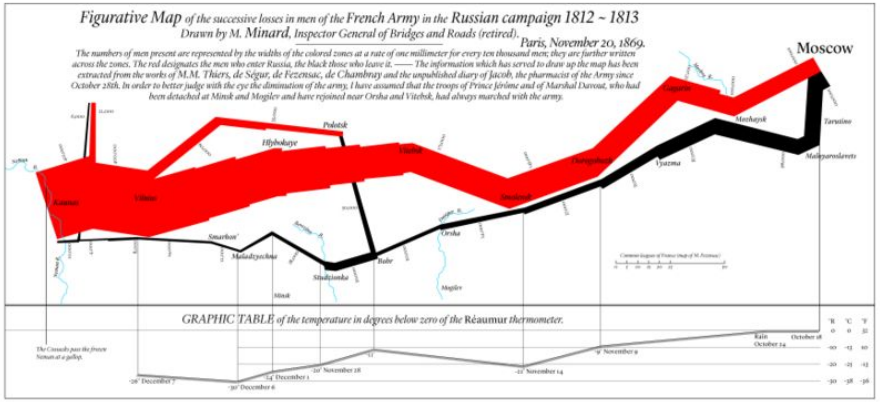


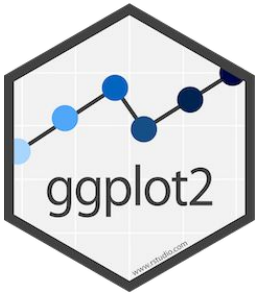
The components:

1. Data
2. Transformation
3. Element
4. Scale
5. Guide
6. Coordinates

...enter the *Layered* Grammar of Graphics

- Hadley Wickham, 2010 develops an open-source implementation of the Grammar of Graphics in R, `ggplot2`, as a graduate student at Iowa State U
- Accompanied by a paper, "A layered grammar of graphics" (27 pages)
- The paper explains:
  - Turning the theory into code
  - Refinements made to the elements
  - Practical examples

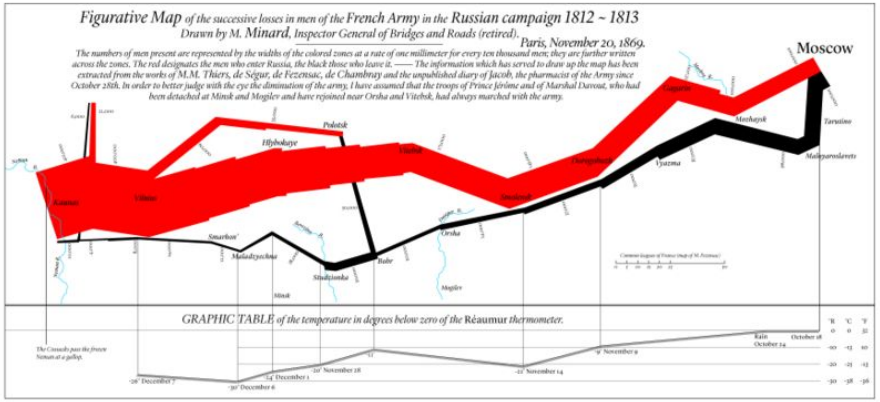


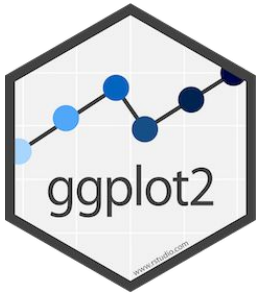


...enter the *Layered* Grammar of Graphics

- Hadley Wickham, 2010 develops an open-source implementation of the Grammar of Graphics in R, `ggplot2`, as a graduate student at Iowa State U
- Accompanied by a paper, "A layered grammar of graphics" (27 pages)
- The paper explains:
  - Turning the theory into code
  - Refinements made to the layers
  - Practical examples

Key takeaway: there is now a package that turns "The Grammar" into elegant code!



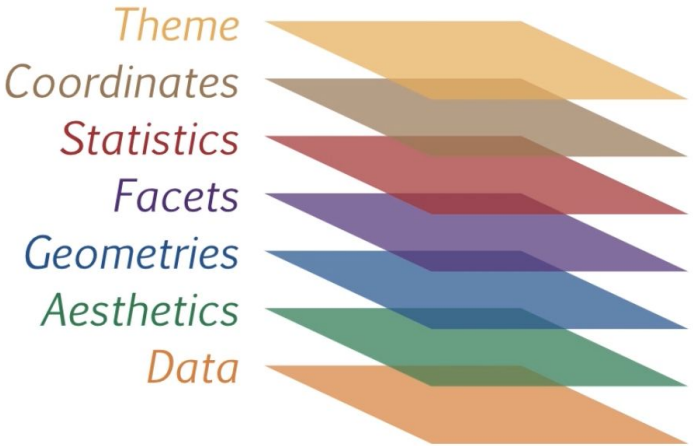


GRAMMAR OF GRAPHICS

The components of a layer

The components of a layer

- **Data**, **Aesthetics**, **Geometries** and **Statistics** (and ***Positions**) are the mandatory elements of a layer
- **Facets** and **Coordinates** (and ***Scales**) can belong to one or more layers
- **Themes** are usually global



*Positions are often thought of as a combination of statistical transformation and geometry

Ingredient 1: Data

Literally, the dataset being plotted

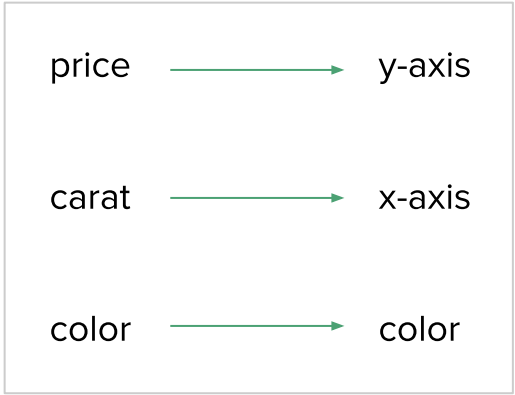
carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
..	...	...	...	...	...	...	...	...	...

Ingredient 2: Aesthetics

The dimensions being plotted.

Aesthetic mappings describe how variables in the data are mapped to visual properties (aesthetics) of the geometry we desire to plot.

They can be things like what goes on the x-axis, y-axis, size of the graphic element (e.g. dots), color or even line width.



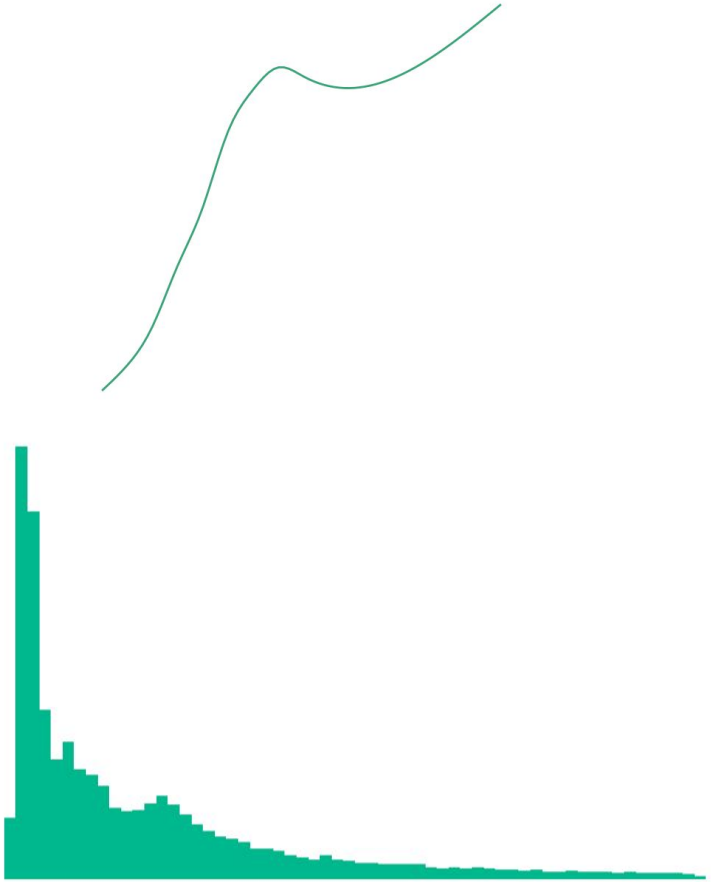
Example of mapping data variables to visual properties on the graphic.

Ingredient 3: Geometries

The type of plot that you create.

Geometric objects, control the type of plot that you create.

Examples of geometries are line plots, scatter plots, bar plots, violin plots, etc.



Ingredient 4: Statistical transformations

Perform a statistical transformation on the data

You can compute a cumulative distribution, a summary of stats, remove duplicates, a custom function or leave the data as it is.

Ingredient 4: Statistical transformations

Perform a statistical transformation on the data

You can compute a cumulative distribution, a summary of stats, remove duplicates, a custom function or leave the data as it is.

cut	price
Ideal	326
Premium	326
Premium	334
...	...

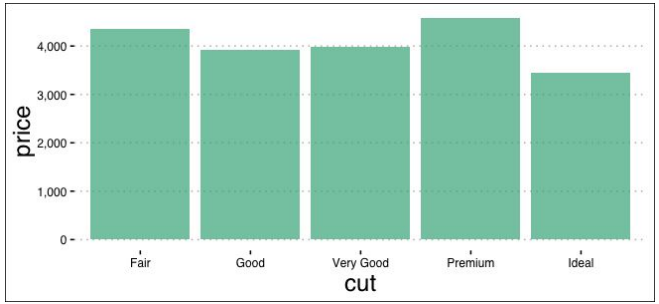
Ingredient 4: Statistical transformations

Perform a statistical transformation on the data

You can compute a cumulative distribution, a summary of stats, remove duplicates, a custom function or leave the data as it is.

cut	price
Ideal	326
Premium	326
Premium	334
...	...

Price per cut – mean



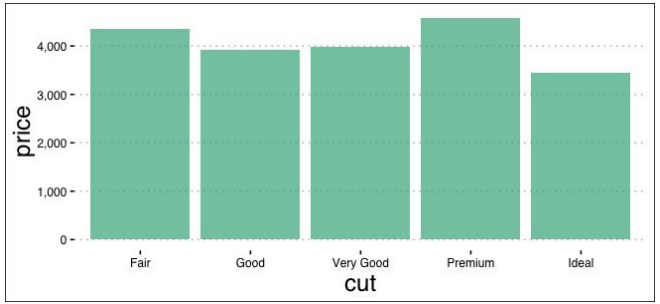
Ingredient 4: Statistical transformations

Perform a statistical transformation on the data

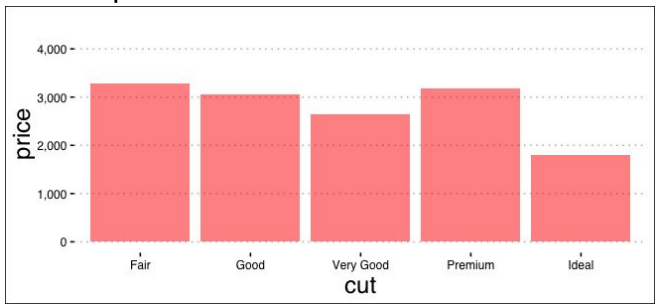
You can compute a cumulative distribution, a summary of stats, remove duplicates, a custom function or leave the data as it is.

cut	price
Ideal	326
Premium	326
Premium	334
...	...

Price per cut – mean



Price per cut – median



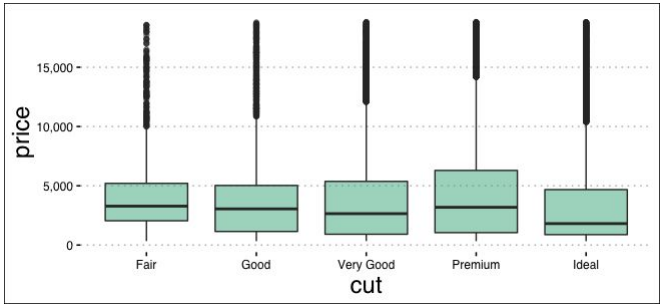
Ingredient 4: Statistical transformations

Perform a statistical transformation on the data

You can compute a cumulative distribution, a summary of stats, remove duplicates, a custom function or leave the data as it is.

cut	price
Ideal	326
Premium	326
Premium	334
...	...

Price per cut – boxplot



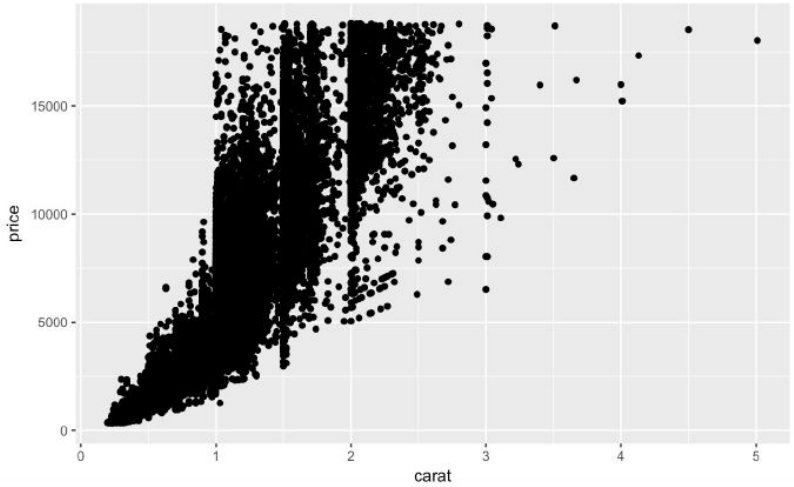
GRAMMAR OF GRAPHICS

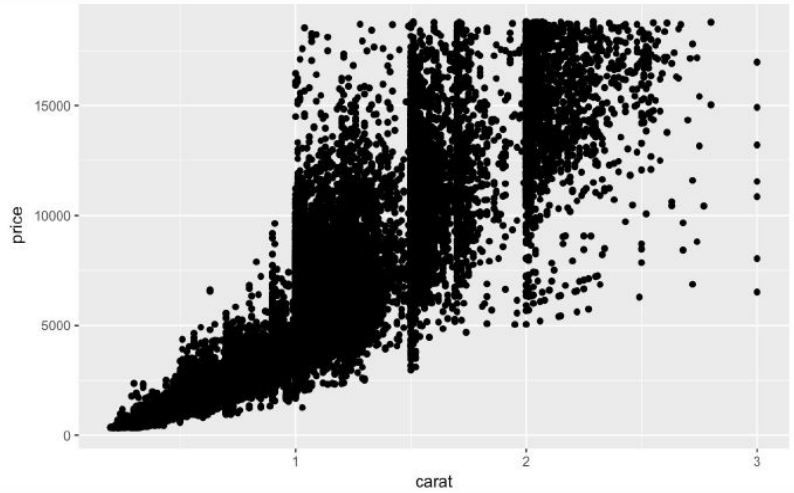
Additional components of a graphic

Ingredient 5: Scales

A scale controls the mapping from data to aesthetic attributes and so we need one scale for each aesthetic property used in a layer.

Scales are common across layers to ensure a consistent mapping from data to aesthetics.

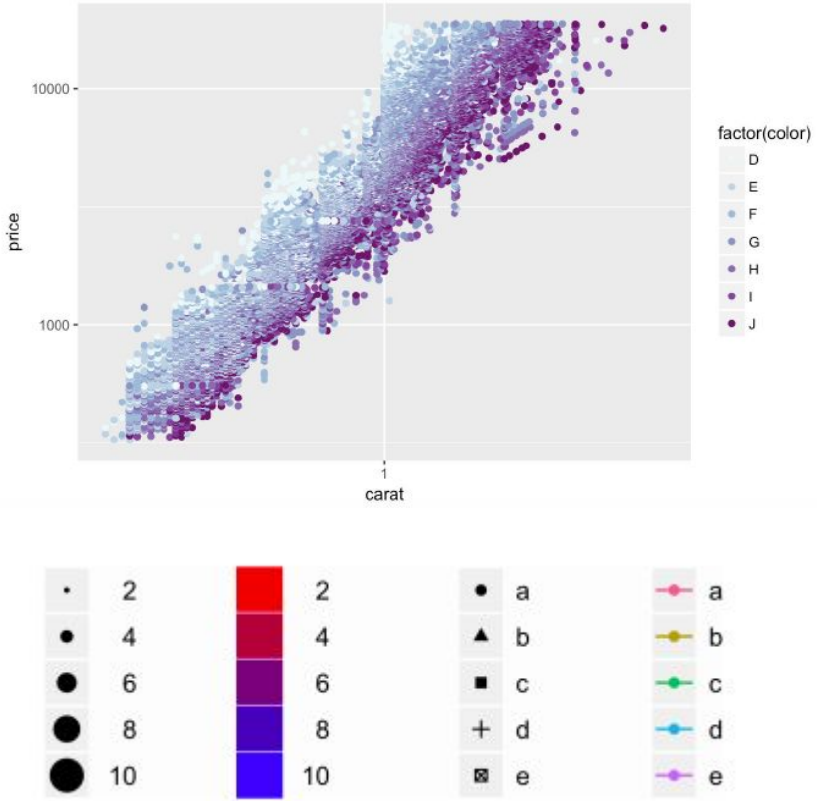




Ingredient 5: Scales

A scale controls the mapping from data to aesthetic attributes and so we need one scale for each aesthetic property used in a layer.

Scales are common across layers to ensure a consistent mapping from data to aesthetics.



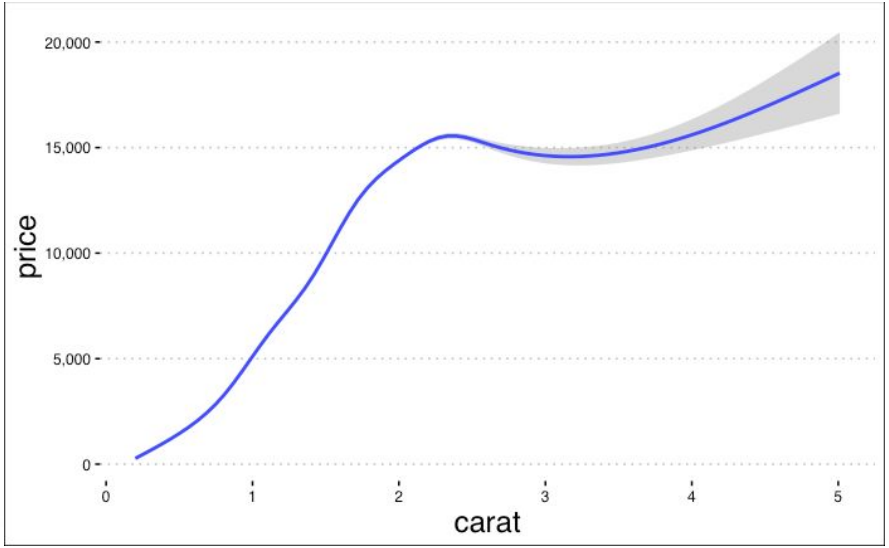
Ingredient 6: Facets 💎

In some circumstances we want to plot **relationships between set variables in multiple subsets of the data with the results appearing as panels in a larger figure**.

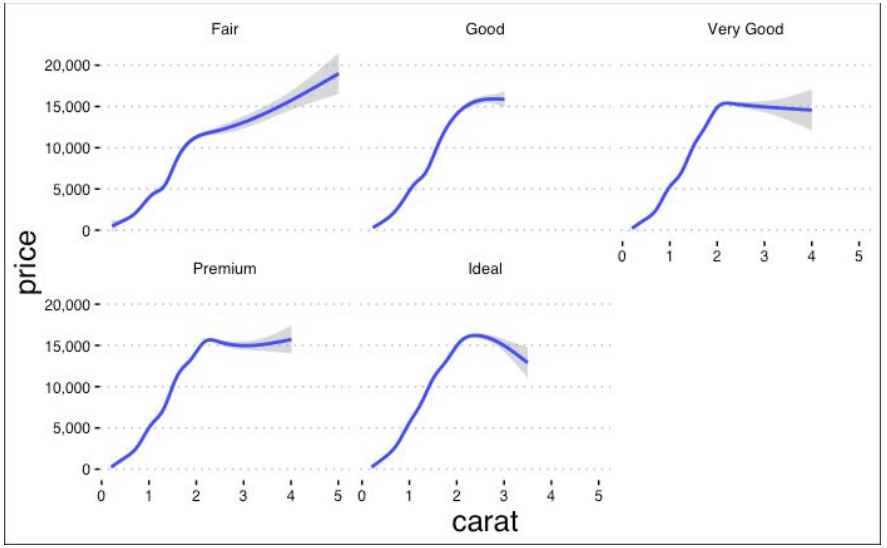
This is a known as a **facet plot**.

Faceting is defined by a categorical variable or variables.

Ingredient 6: Facets 💎



How does the price of a diamond increase as its size in carats increases? We can plot all the data points together...



...or we might get more insight by focusing on each diamond **cut** separately (Fair, Good, Very Good, etc.) through a **facet plot / grid.**

Ingredient 7: Coordinate system

A coordinate system maps the position of objects onto the plane of the plot.

The most typical ones are Cartesian or polar coordinates.

In contrast to *scales*, the coordinate system only changes the *depiction* of the data, not the data itself!

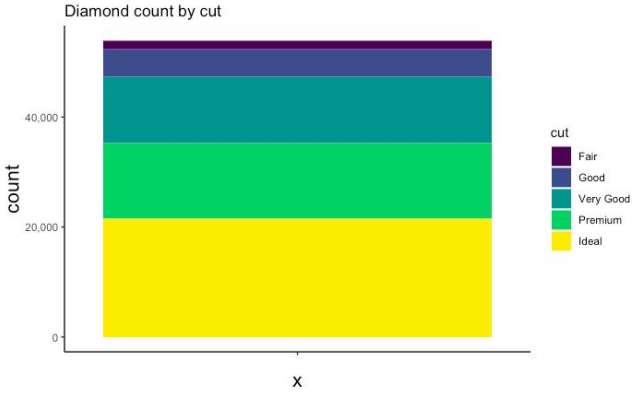
Ingredient 7: Coordinate system

A coordinate system maps the position of objects onto the plane of the plot.

The most typical ones are Cartesian or polar coordinates.

In contrast to *scales*, the coordinate system only changes the *depiction* of the data, not the data itself!

Cartesian coordinates



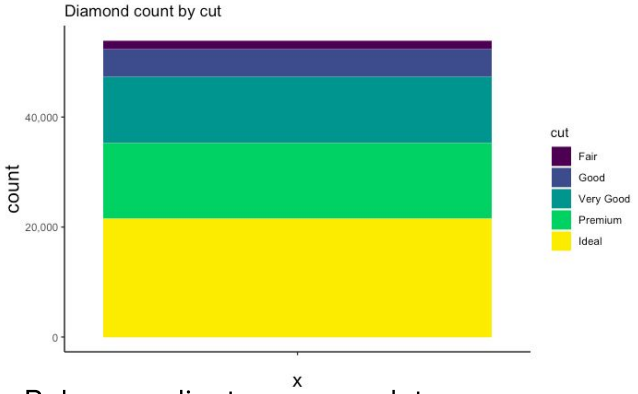
Ingredient 7: Coordinate system

A coordinate system maps the position of objects onto the plane of the plot.

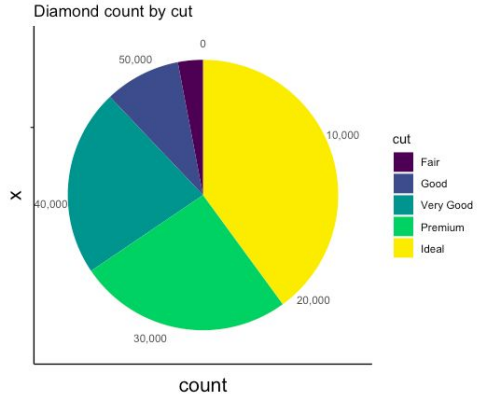
The most typical ones are Cartesian or polar coordinates.

In contrast to *scales*, the coordinate system only changes the *depiction* of the data, not the data itself!

Cartesian coordinates



Polar coordinates - same data



Ingredient 8: Theme

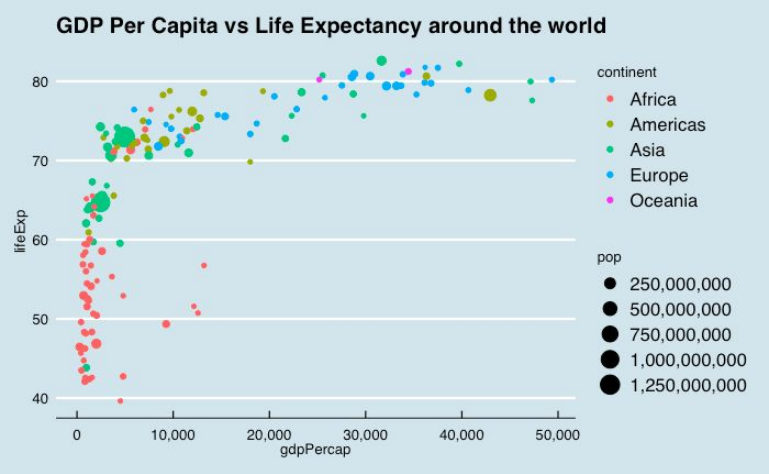
All the other elements of the plot that are not directly related to the data.

- Annotations
- Titles, labels
- Fonts, font sizes
- Other stylistic choices

Ingredient 8: Theme

All the other elements of the plot that are not directly related to the data.

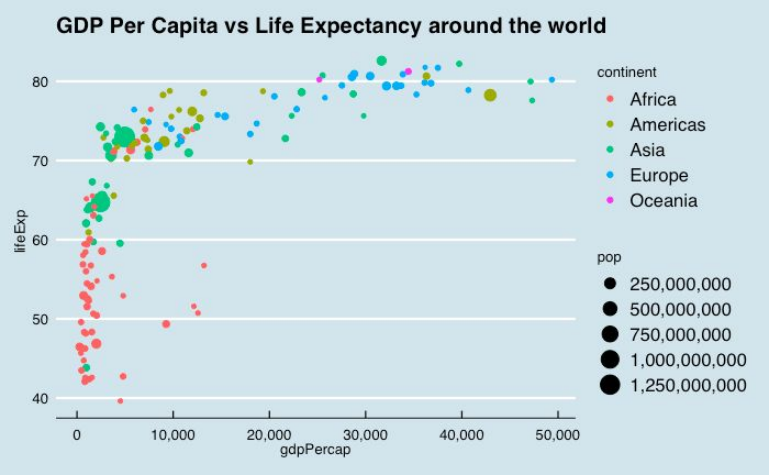
- Annotations
- Titles, labels
- Fonts, font sizes
- Other stylistic choices

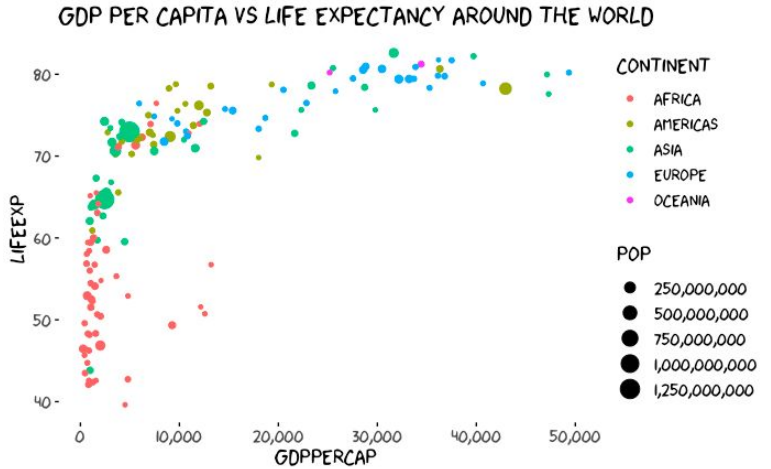


Ingredient 8: Theme

All the other elements of the plot that are not directly related to the data.

- Annotations
- Titles, labels
- Fonts, font sizes
- Other stylistic choices

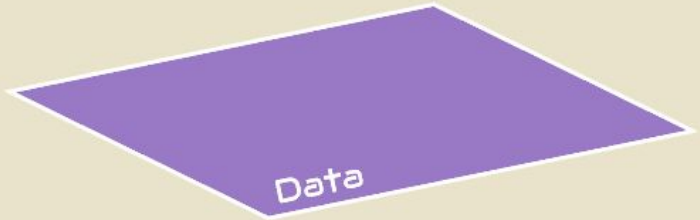




Grammar of Graphics

```r
#------------------------------ Visualising your boxplot -------------

# Before plotting if not installed install.package("ggplot2")
# Then activate ggplot2 package
library(ggplot2)

# Create new variable for plot only x and y axis. ('data' and 'aesthatics' layer)
plot <- ggplot(data=new.data, aes(x=Genre, y=Gross...US))

# Create new variable with geometries layer.
q <- plot + geom_jitter(aes(fill=Studio, size=Budget...mill.),
                        shape = 21, # this will shape a border around data points.
                        colour = "Black") + # with the border color of black.
  geom_boxplot(alpha=0.7, outlier.color = NA) # places the boxplot on the data points
                                              # and removes boxplot layer outliers.

# Change axis and title if needed.
q <- q +
  xlab("Genre") +
  ylab("Gross % US") +
  ggtitle("Domestic Gross % by Genre")

# Make your plot visually attractive and readable with the 'theme' function. (Theme layer)
q + theme(axis.title = element_text(colour = "Blue", size = 14),
          axis.text = element_text(size = 12),
          legend.title = element_text(size = 12),
          legend.text = element_text(size = 10),
          plot.title = element_text(size = 14, hjust = 0.5), # 'hjust' will center your text.
          panel.background = element_rect(fill = "#E6E3C5"))

```

xy, 3902, 29, 9,
4756, x, 72, 633,
647, 617, 827, 3,
1, 21, 45, tyu, 6,
987, 457, 283, 8,
4, 5, 671, 34, 67,
x, 981, hu, 89, 5



Awesome illustration made by https://medium.com/@TdeBeus

PART II: Implementations in Python

(*small side note on implementations🤫)

There are multiple other Python packages that are inspired by the Grammar of Graphics, most famously:

- Altair
- Bokeh

Today I will talk to you about the ones I use.

`plotnine`: `ggplot2` for Python

```
from plotnine import *
```

`plotnine` is an implementation of *layered grammar of graphics* in Python and it is based on ggplot2. The grammar allows users to compose plots by explicitly mapping data to the visual objects that make up the plot.

- Latest version: plotnine 0.6.0
- Documentation: (https://plotnine.readthedocs.io)
- Data visualization in Python like in R's ggplot2, Dr. Gregor Scheithauer (https://medium.com/@gscheithauer/data-visualization-in-python-like-in-rs-ggplot2-bc62f8debbf5)

`plotnine`: `ggplot2` for Python

`plotnine` is an implementation of *layered grammar of graphics* in Python and it is based on ggplot2. The grammar allows users to compose plots by explicitly mapping data to the visual objects that make up the plot.

- Latest version: plotnine 0.6.0
- Documentation: (https://plotnine.readthedocs.io)
- Data visualization in Python like in R's ggplot2, Dr. Gregor Scheithauer (https://medium.com/@gscheithauer/data-visualization-in-python-like-in-rs-ggplot2-bc62f8debbf5)

name	defense	attack	speed	type1	type2	is_legendary
Bulbasaur	49	49	45	grass	poison	not_legendary
Ivysaur	63	62	60	grass	poison	not_legendary
Venusaur	123	100	80	grass	poison	not_legendary
Squirtle	65	48	43	water	NaN	not_legendary
Wartortle	80	63	58	water	NaN	not_legendary
Blastoise	120	103	78	water	NaN	not_legendary
Caterpie	35	30	45	bug	NaN	not_legendary
Metapod	55	20	30	bug	NaN	not_legendary
Butterfree	50	45	70	bug	flying	not_legendary
Weedle	30	35	50	bug	poison	not_legendary

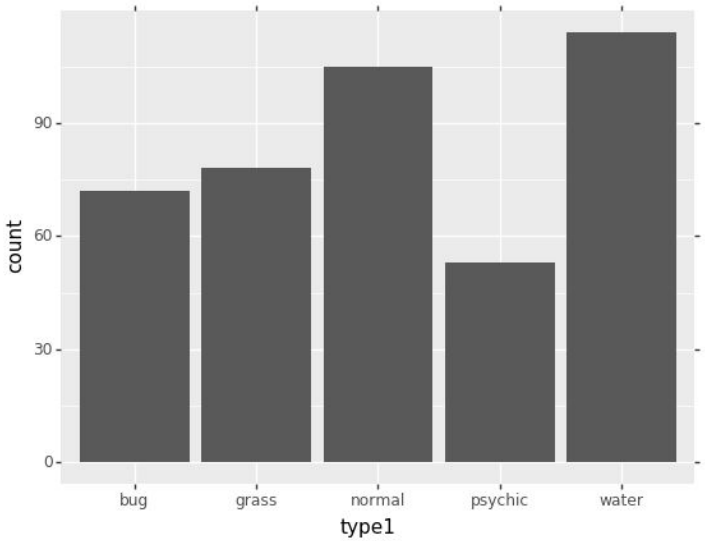
Pokemon dataset on Kaggle

`plotnine`: how it works

`plotnine` is an implementation of ggplot2 (R) in Python, and so it follows the same additive syntax, building the components of a graphic layer by layer. (Some of) the components are:

- Data

```
from plotnine import *

ggplot(data = pokemons)
```

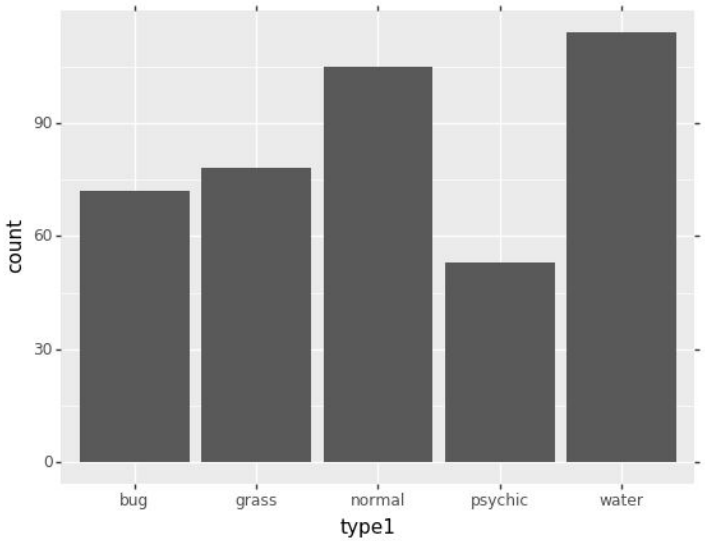


`plotnine`: how it works

`plotnine` is an implementation of ggplot2 (R) in Python, and so it follows the same additive syntax, building the components of a graphic layer by layer. (Some of) the components are:

- Data
- Geometries:
  - geom_ point()
  - geom_ bar()
  - geom_density()
  - geom_area()
  - geom_ histogram()
  - geom_boxplot()

```
ggplot(data = pokemons) +\
    geom_bar( ... )
```

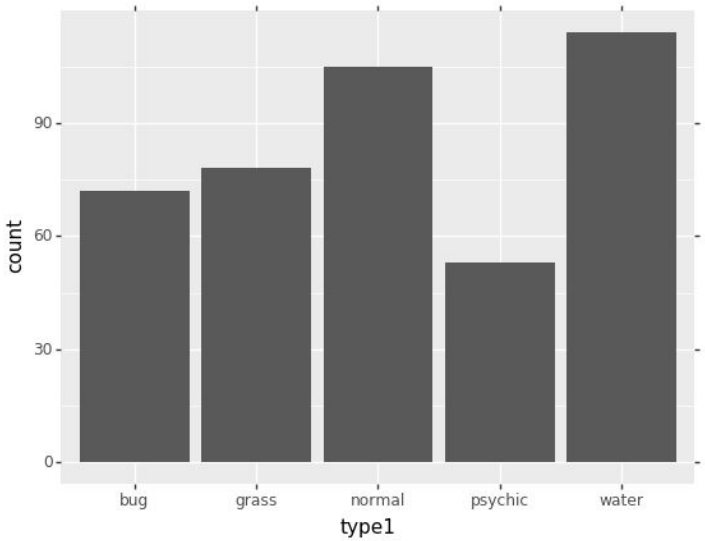


`plotnine`: how it works

`plotnine` is an implementation of ggplot2 (R) in Python, and so it follows the same additive syntax, building the components of a graphic layer by layer. (Some of) the components are:

- Data
- Geometries
- Aesthetics:
  - x,y,
  - Color
  - Fill
  - Shape
  - Linewidth

```
ggplot(data = pokemons) +\
    geom_bar(aes(x = 'type1'))
```

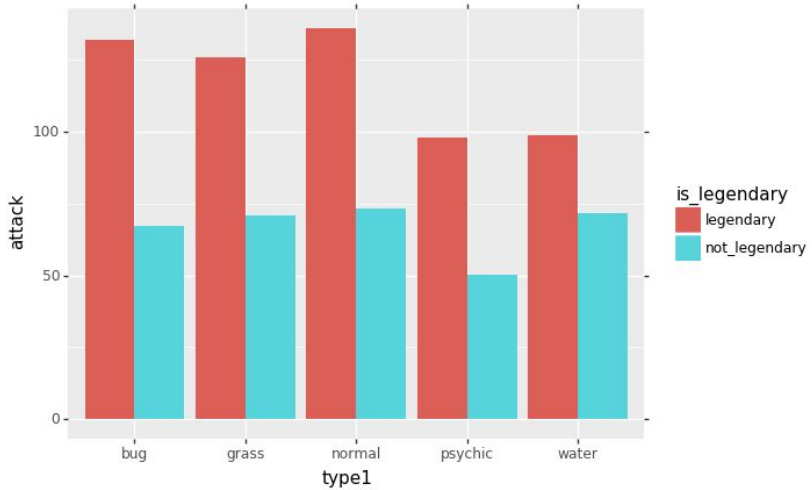


`plotnine`: how it works

`plotnine` is an implementation of ggplot2 (R) in Python, and so it follows the same additive syntax, building the components of a graphic layer by layer. (Some of) the components are:

- Data
- Geometries
- Aesthetics
- Statistical transformations:
  - stat_identity()
  - stat_count()
  - stat_bin()
  - stat_boxplot()

```
ggplot(data = pokemons) +\
    geom_col(aes(x = 'type1',
                 y = 'attack',
                 fill = 'is_legendary'),
             stat = 'summary',
             position = 'dodge')
```

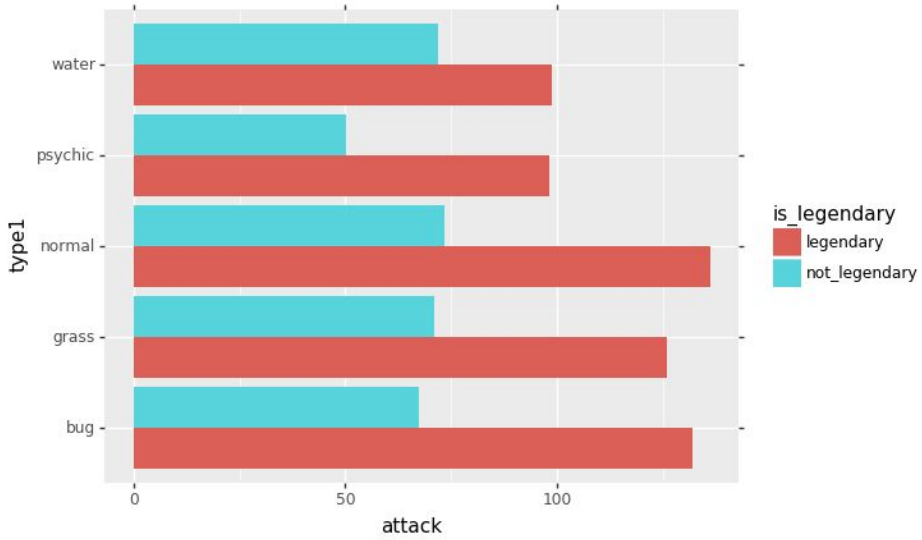


`plotnine`: how it works

`plotnine` is an implementation of ggplot2 (R) in Python, and so it follows the same additive syntax, building the components of a graphic layer by layer. (Some of) the components are:

- Data
- Geometries
- Aesthetics
- Statistical transformations
- Coordinates
  - coord_cartesian()
  - coord_flip()
  - NOT yet coord_polar() (See open issue)

```
ggplot(data = df) +\
    geom_col(aes(x = 'type1',
                 y = 'attack',
                 fill = 'is_legendary'),
             stat = 'summary',
             position = 'dodge') +\
    coord_flip()
```

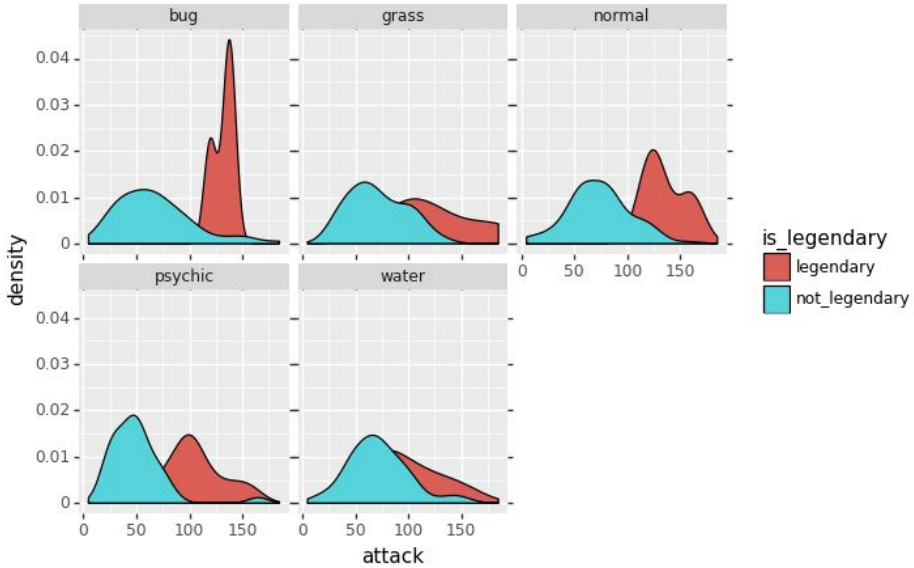


`plotnine`: how it works

`plotnine` is an implementation of ggplot2 (R) in Python, and so it follows the same additive syntax, building the components of a graphic layer by layer. (Some of) the components are:

- Data
- Geometries
- Aesthetics
- Statistical transformations
- Coordinates
- Facets:
  - facet_grid()
  - facet_wrap()

```
ggplot(data = df) +\
    geom_density(aes(x = 'attack',
                     fill = 'is_legendary')) +\
    facet_wrap('~ type1', scales = 'fixed')
```

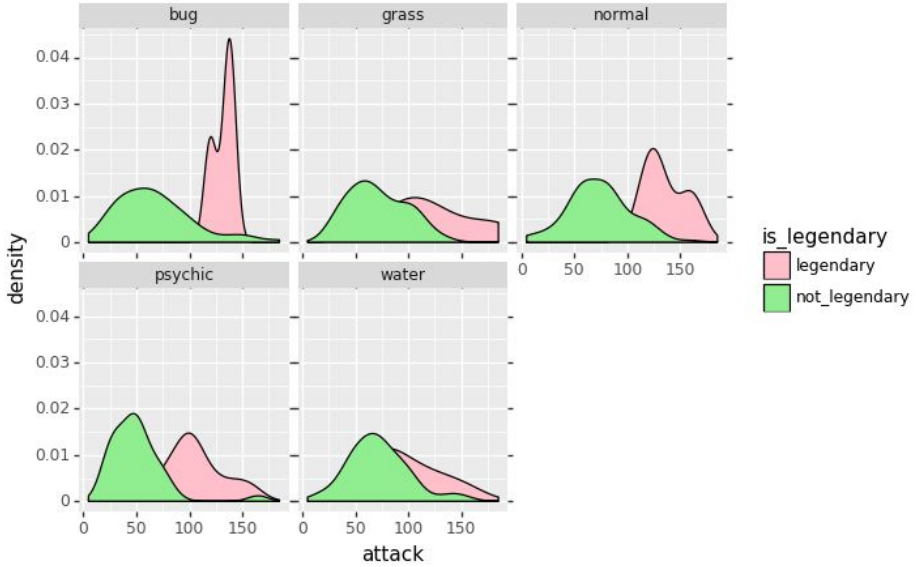


`plotnine`: how it works

`plotnine` is an implementation of ggplot2 (R) in Python, and so it follows the same additive syntax, building the components of a graphic layer by layer. (Some of) the components are:

- Data
- Geometries
- Aesthetics
- Statistical transformations
- Coordinates
- Facets
- Scales:
  - scale_x_continuous()
  - scale_x_log10()
  - scale_color_discrete()
  - ylim(), xlim()

```
ggplot(data = df) +\
    geom_density(aes(x = 'attack',
                     fill = 'is_legendary')) +\
    facet_wrap('~ type1', scales = 'fixed') +\
    scale_fill_manual(['pink', 'lightgreen'])
```

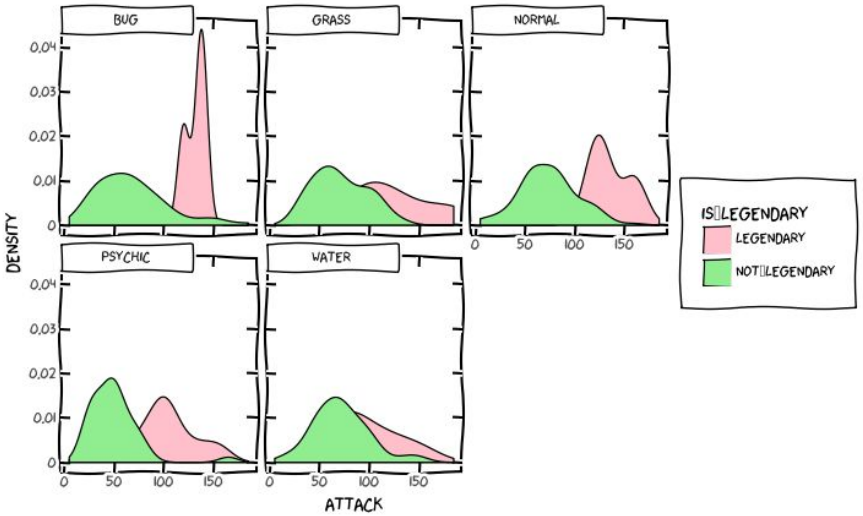


`plotnine`: how it works

`plotnine` is an implementation of ggplot2 (R) in Python, and so it follows the same additive syntax, building the components of a graphic layer by layer. (Some of) the components are:

- Data
- Geometries
- Aesthetics
- Statistical transformations
- Coordinates
- Facets
- Scales
- *Positions
- *Themes

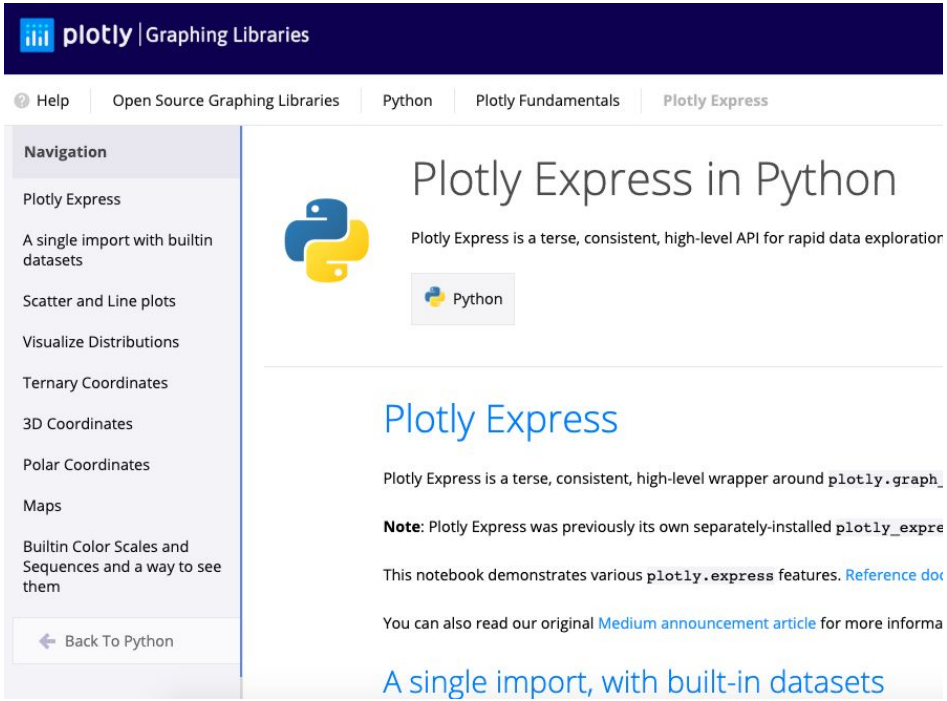
```
ggplot(data = df) +\
    geom_density(aes(x = 'attack',
                     fill = 'is_legendary')) +\
    facet_wrap('~ type1', scales = 'fixed') +\
    scale_fill_manual(['pink', 'lightgreen']) +\
    theme_xkcd()
```



`plotly_express`: interactive, layered GoG

`plotly_express` is a wrapper for Plotly.py that exposes a simple syntax for complex charts. Inspired by `seaborn` and `ggplot2`, it was specifically designed to have a terse, consistent and easy-to-learn API: with just a single import, you can make richly interactive plots in just a single function call, including faceting, maps, animations, and trendlines

- Latest version: plotly-express 0.4.1
- Documentation
- Introducing Plotly Express on Medium



`plotly_express`: how it works

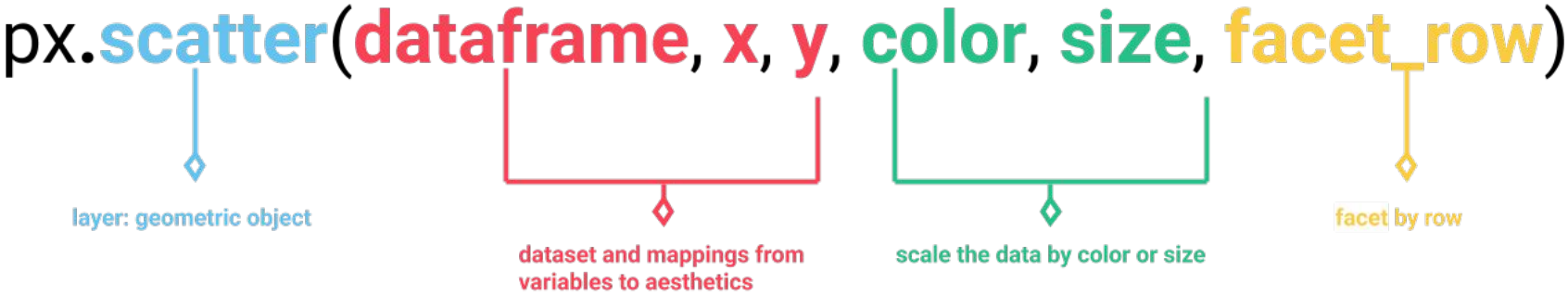
```
import plotly_express as px
```

- Inspired by the Layered Grammar of Graphics (H.Wickham)
- Takes in a tidy Pandas dataframe as input
- Allows for interactivity
- Focus on *conciseness*

`plotly_express`: how it works

```
import plotly_express as px
```

- Inspired by the Layered Grammar of Graphics (H.Wickham)
- Takes in a tidy Pandas dataframe as input
- Allows for interactivity
- Focus on *conciseness*



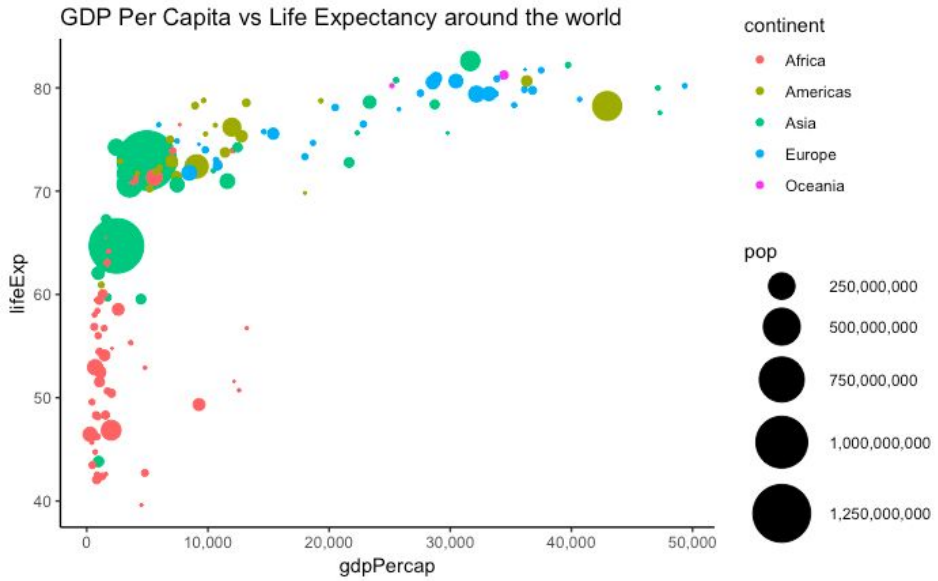
The Plotly Express syntax translates to the elements of the Layered Grammar of Graphics. Example of a scatterplot function. Source: Express graphics with Plotly, Steven Liu

`plotly_express`: how it works

Example with Gapminder data:

> Plotting life expectancy vs GDP in 2007

	country	continent	year	lifeExp	pop	gdpPercap
11	Afghanistan	Asia	2007	43.828	31889923	974.580338
23	Albania	Europe	2007	76.423	3600523	5937.029526
35	Algeria	Africa	2007	72.301	33333216	6223.367465
47	Angola	Africa	2007	42.731	12420476	4797.231267
59	Argentina	Americas	2007	75.320	40301927	12779.379640



`plotly_express`: how it works

The arguments plotly_express takes can be mapped also the the GoG elements:

```
px.scatter()
```

- Geometries:
  - px.scatter()
  - px.bar()
  - px.line()
  - px.histogram()
  - px.choropleth()
  - px.parallel_coordinates()

`plotly_express`: how it works

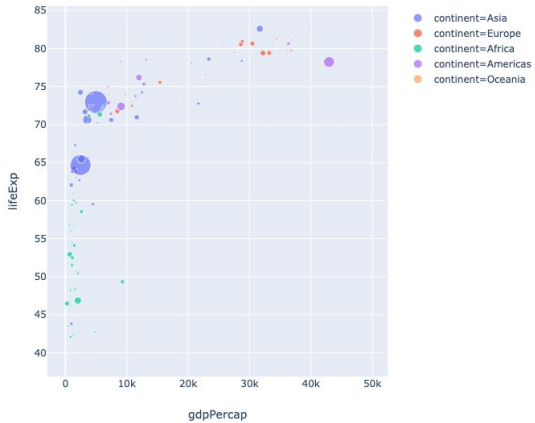
The arguments plotly_express takes can be mapped also the the GoG elements:

- Geometries
- Data
- Aesthetics:
  - x, y, color, size, linewidth, theta, etc

```
px.scatter(gapminder,
           x="gdpPercap",
           y="lifeExp",
           color='continent',
           size = 'pop')
```

data

mapping of data to graph ical properties



`plotly_express`: how it works

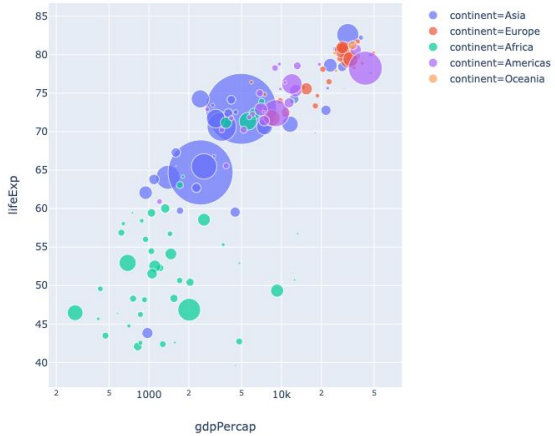
The arguments plotly_express takes can be mapped also the the GoG elements:

- Geometries
- Data
- Aesthetics
- Scales:
  - size_max
  - category_orders
  - color_discrete_sequence
  - color_continuous_scale
  - log_x, log_y, etc

```
px.scatter(gapminder,
           x="gdpPercap",
           y="lifeExp",
           color='continent',
           size = 'pop',
           size_max=60,
           log_x = True)
```

Limit on the scale of "size" aesthetic

Transforms the x into the logarithmic scale

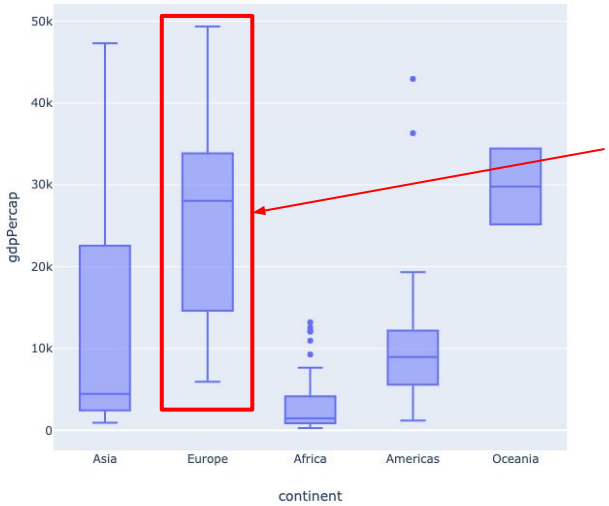


`plotly_express`: how it works

The arguments plotly_express takes can be mapped also the the GoG elements:

- Geometries
- Data
- Aesthetics
- Scales
- Statistical transformations:
  - Implicit in each kind of geometry
  - Otherwise need to be specified as new column (e.g. **bar_plot** with median instead of counts)

```
px.box(gapminder,
       x = 'continent',
       y = 'gdpPercap')
```



The statistical transformations for the median, quartiles and whiskers are automatically performed by specifying the geometry

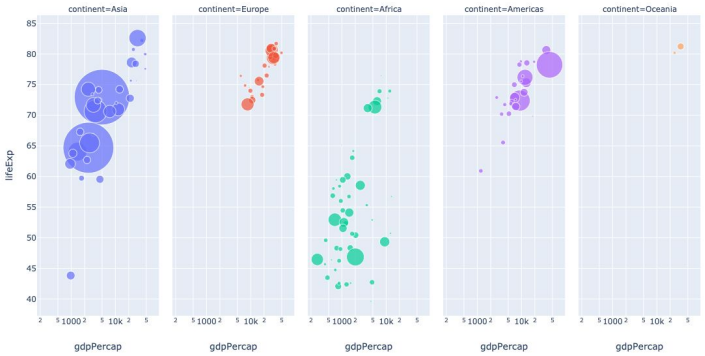
`plotly_express`: how it works

The arguments plotly_express takes can be mapped also the the GoG elements:

- Geometries
- Data
- Aesthetics
- Scales
- Statistical transformations
- Facets
  - facet_col
  - facet_row

```
px.scatter(gapminder,
           x="gdpPercap",
           y="lifeExp",
           color="continent",
           size="pop",
           size_max=60,
           facet_col="continent",
           log_x=True)
```

Which variable defines the facet subsetting



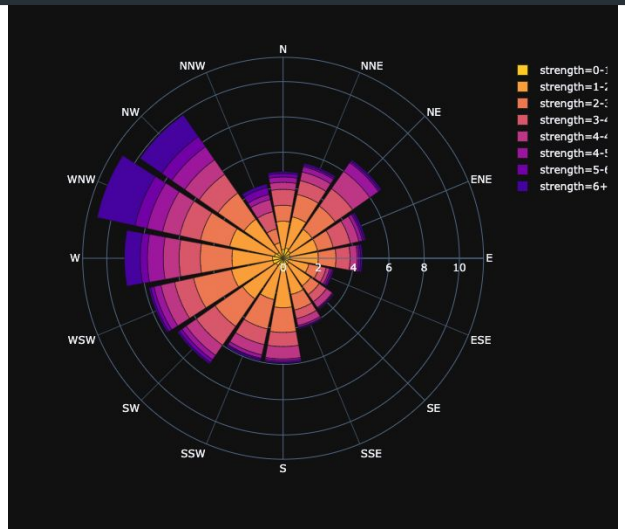
`plotly_express`: how it works

The arguments plotly_express takes can be mapped also the the GoG elements:

- Geometries
- Data
- Aesthetics
- Scales
- Statistical transformations
- Facets
- Coordinates
  - Inbuilt in the chart geometries and in the scalar transformations (e.g. log_x)

```
wind = px.data.wind()
fig = px.bar_polar(wind,
                   r="frequency",
                   theta="direction",
                   color="strength",
                   template="plotly_dark")
```

Implied in the chart geometry

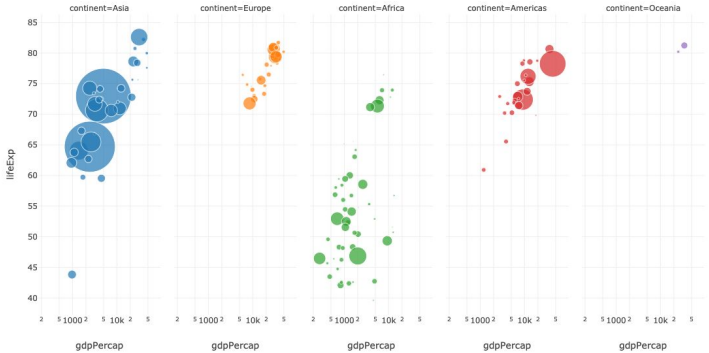


`plotly_express`: how it works

The arguments plotly_express takes can be mapped also the the GoG elements:

- Aesthetics
- Scales
- Geometries
- Statistical transformations
- Facets
- Coordinates
- Themes
  - Plotly express provides some limited inbuilt themes, inherited from Plotly

```
px.scatter(gapminder,
           x="gdpPercap",
           y="lifeExp",
           color='continent',
           size = 'pop',
           size_max=60,
           facet_col='continent',
           log_x = True,
           template='none')
```

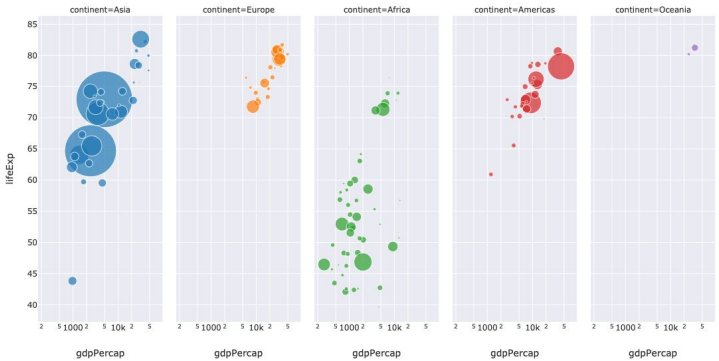


`plotly_express`: how it works

The arguments plotly_express takes can be mapped also the the GoG elements:

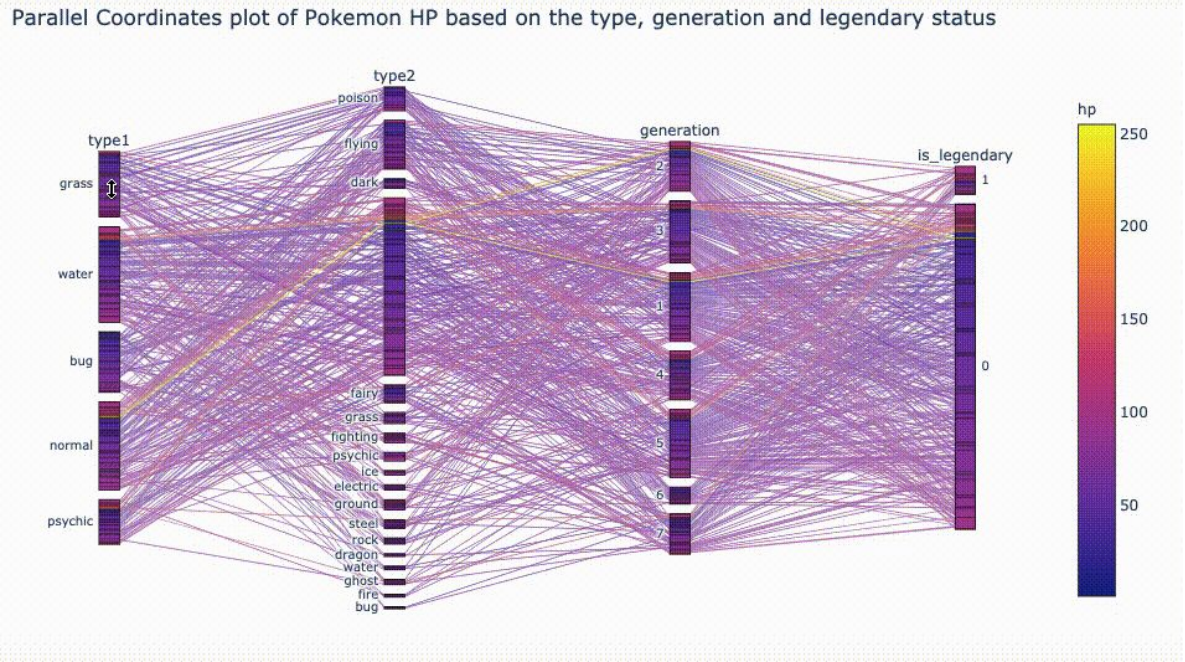
- Aesthetics
- Scales
- Geometries
- Statistical transformations
- Facets
- Coordinates
- Themes
  - Plotly express provides some limited inbuilt themes, inherited from Plotly

```
px.scatter(gapminder,
           x="gdpPercap",
           y="lifeExp",
           color='continent',
           size = 'pop',
           size_max=60,
           facet_col='continent',
           log_x = True,
           template='none')
```



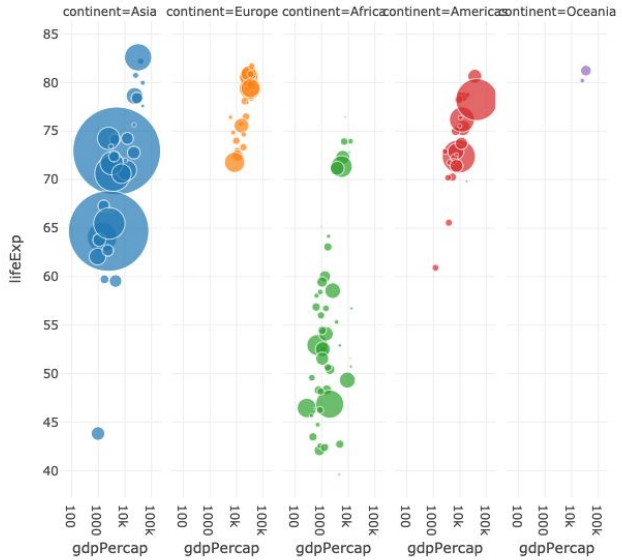
`plotly_express`: summary

Build complex interactive plots quickly by using the logic of the Grammar



PART III: *The Grammar* as a mindset (aka DIY 🛠️)

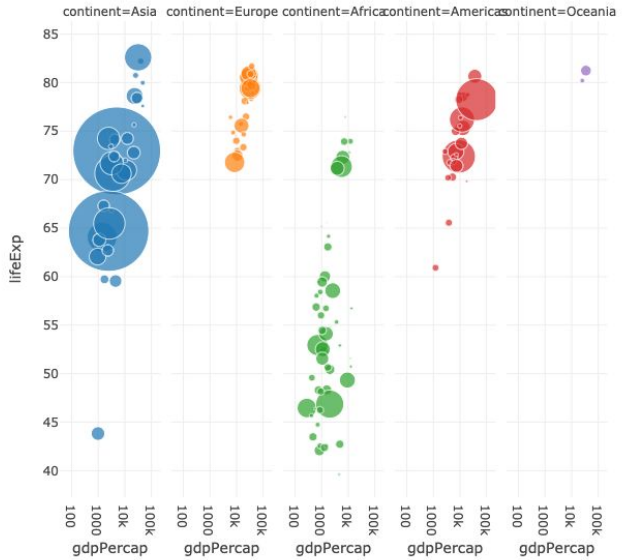
Challenge: Build the Gapminder chart in `matplotlib` using the Grammar of Graphics



Challenge: Build the Gapminder chart in `matplotlib` using the Grammar of Graphics

Approach:

1. Map the data to the aesthetics we want to depict
2. Define the geometry and the statistical transformations
3. Choose the scale and coordinate system
4. Add other “theme” elements



Challenge: Build the Gapminder chart in `matplotlib` using the Grammar of Graphics

Grammatical element	Our choice	Matplotlib class / parameter (maybe)
geometry	scatter plot	`matplotlib.pyplot.scatter()`

aesthetics
- x
- y
- size
- color
	- GDP per capita
	- Life expectancy
	- Population
	- Continent
		`...scatter(x =...,`
		`y =...,`
		`s =...,`
		`c =...)`

scale	- x in log10 transformation - size (population) needs to be scaled to the range of x, y	`matplotlib.pyplot.xscale('log')` *`Has to be done manually?`*

facets	split on 'continent'	`fig, ax = plt.subplots()`
		`for i, continent in enumerate(continents):`
		`ax[i] = ...`

Challenge: Build the Gapminder chart in `matplotlib` using the Grammar of Graphics

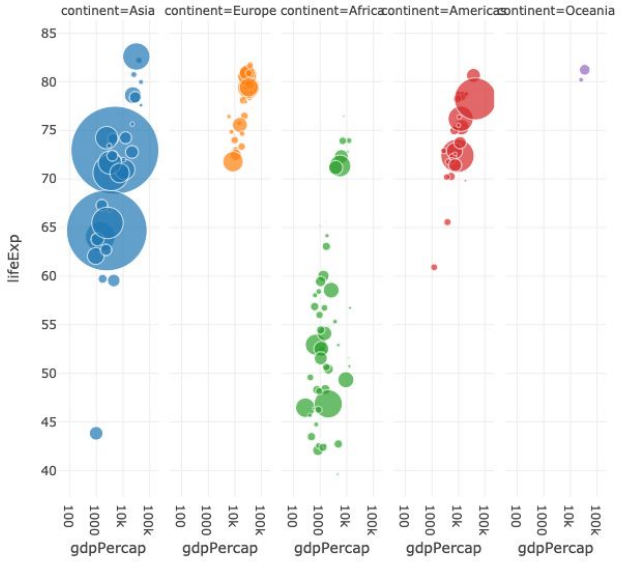
Grammatical element	Our choice	Matplotlib class / parameter (maybe)
…	…	…

facets	split on ‘continent’	`fig, ax = plt.subplots()`
		`for i, continent in enumerate(continents):`
		`  ax[i] = ...`

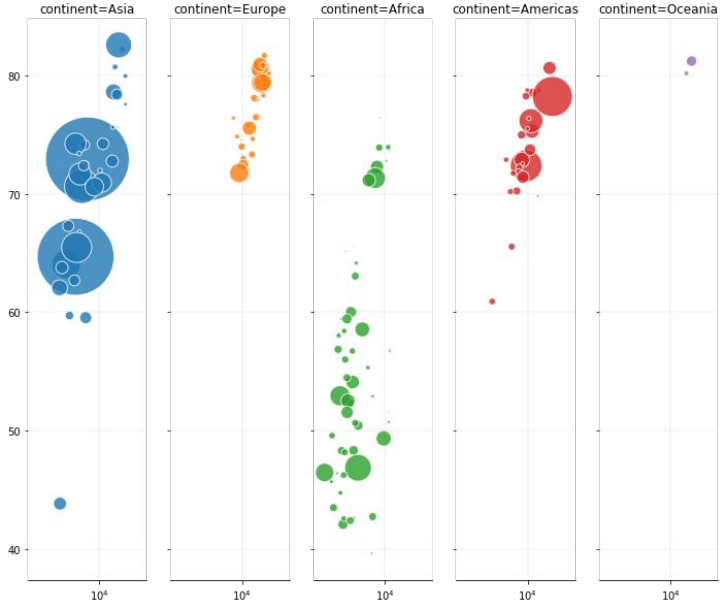
theme	- Seaborn default colormap
	- Bubbles have white edges
	- Some alpha
	- (other small changes)
		`pal = sns.color_palette()`
		`colors = pal.as_hex()`
		`...scatter(...,`
		`  color = colors[i],`
		`  alpha = 0.6,`
		`  edgecolor = 'white',`
		`  lw = 2`

Challenge: Build the Gapminder chart in `matplotlib` using the Grammar of Graphics

Made with Plotly Express



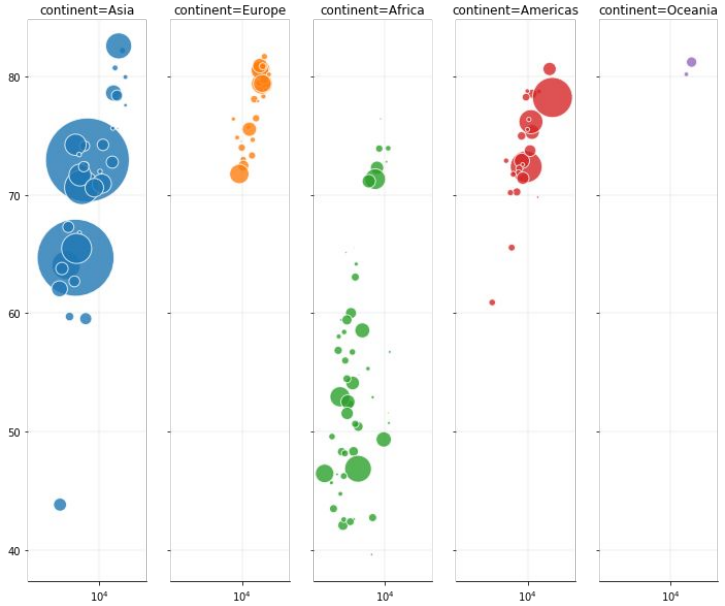
Made with matplotlib



Challenge: Build the Gapminder chart in `matplotlib` using the Grammar of Graphics

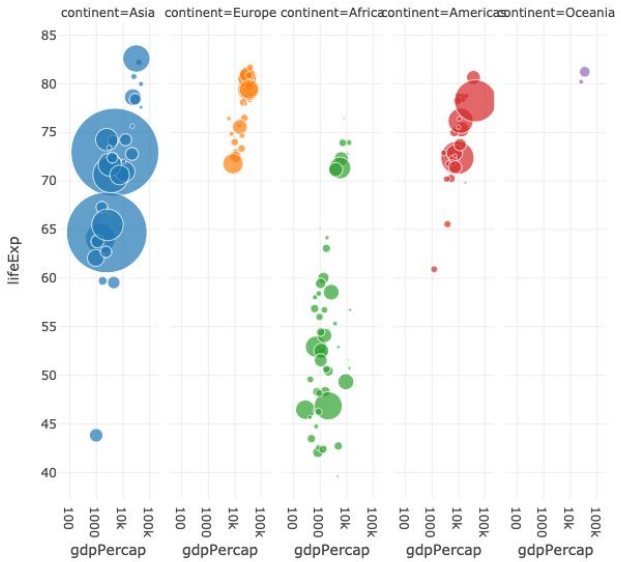
```
f, ax = plt.subplots(ncols=5,
                     sharex=True,
                     sharey=True,
                     figsize = (12,10))
pal = sns.color_palette()
colors = pal.as_hex()
for i, continent in enumerate(gapminder.continent.unique()):
    tmp = gapminder[gapminder.continent == continent]
    ax[i].scatter(x = tmp['gdpPercap'],
                  y = tmp['lifeExp'],
                  s = tmp['pop']/200000,
                  color = colors[i],
                  alpha = 0.8,
                  edgecolor = 'white',
                  lw = 1)
    ax[i].spines['top'].set_visible(False)
    ax[i].spines['left'].set_visible(False)
    ax[i].spines['right'].set_visible(False)
    ax[i].grid(color='grey',
               linestyle='-',
               linewidth=0.25,
               alpha=0.5)
    ax[i].set_title('continent={}'.format(continent))
    plt.xscale('log')
plt.show()
```

Made with matplotlib



Challenge: Build the Gapminder chart in `matplotlib` using the Grammar of Graphics

Made with Plotly Express




```
p = px.scatter(gapminder,
               x="gdpPercap",
               y="lifeExp",
               color='continent',
               size = 'pop',
               size_max=60,
               facet_col='continent',
               log_x = True,
               template='none')
p.update_layout(showlegend=False, template = 'none')
```

Thank you!

Twitter: @tvasi
Email: tvasilikioti@babbel.com