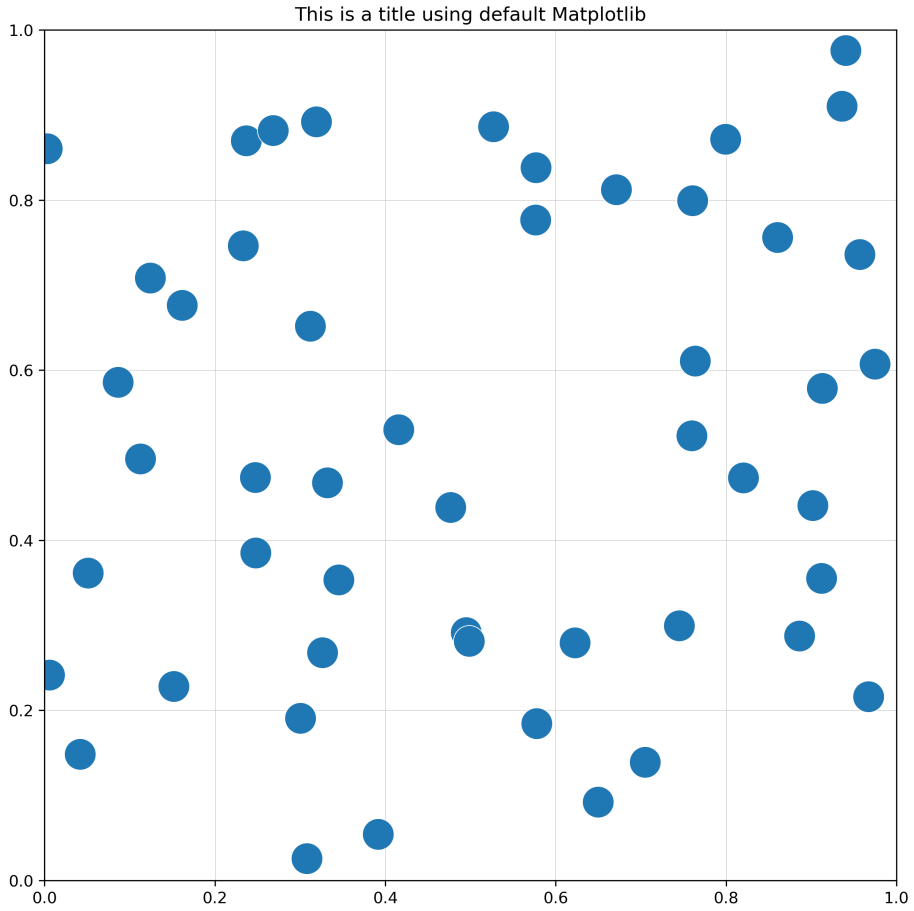
Matplotlib: Beyond the Default

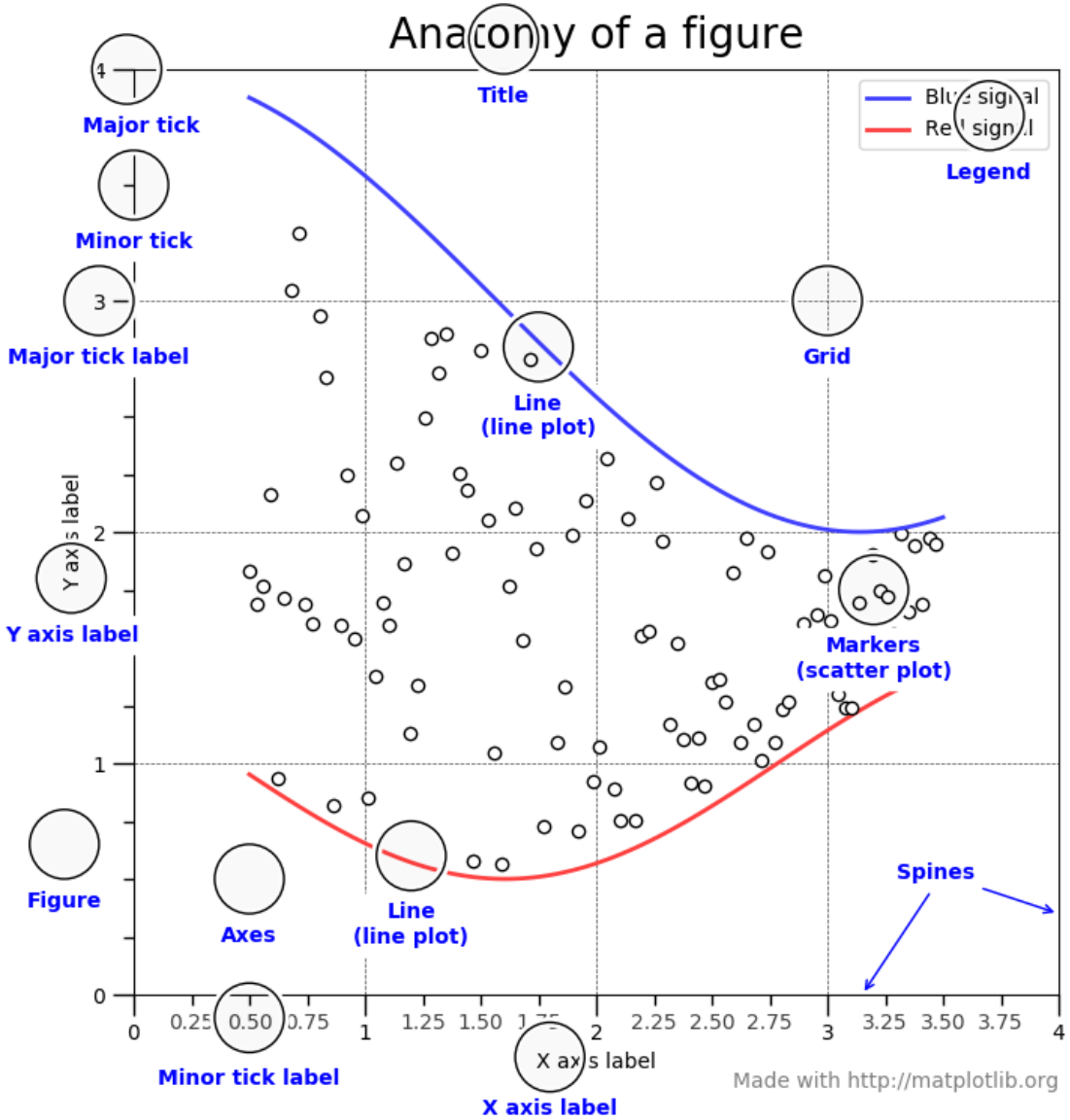
Good practice in data visualisation with Peter McKeever

Default Matplotlib

```
fig, ax = plt.subplots(figsize=(8,8))
fig.set_facecolor("white")

ax.grid(linewidth=.25, zorder=1)

ax.set_title("This is a title using default Matplotlib")

np.random.seed(402)
x = np.random.uniform(0,1,50)
y = np.random.uniform(0,1,50)

ax.scatter(x,y,
           edgecolors="white",
           linewidths=0.5,
           s=400,
           zorder=2)

ax.set_xlim(0,1)
ax.set_ylim(0,1)

plt.tight_layout()
plt.savefig("../plots/3_default_mpl", bbox_inches="tight", dpi=300)
plt.show()
```



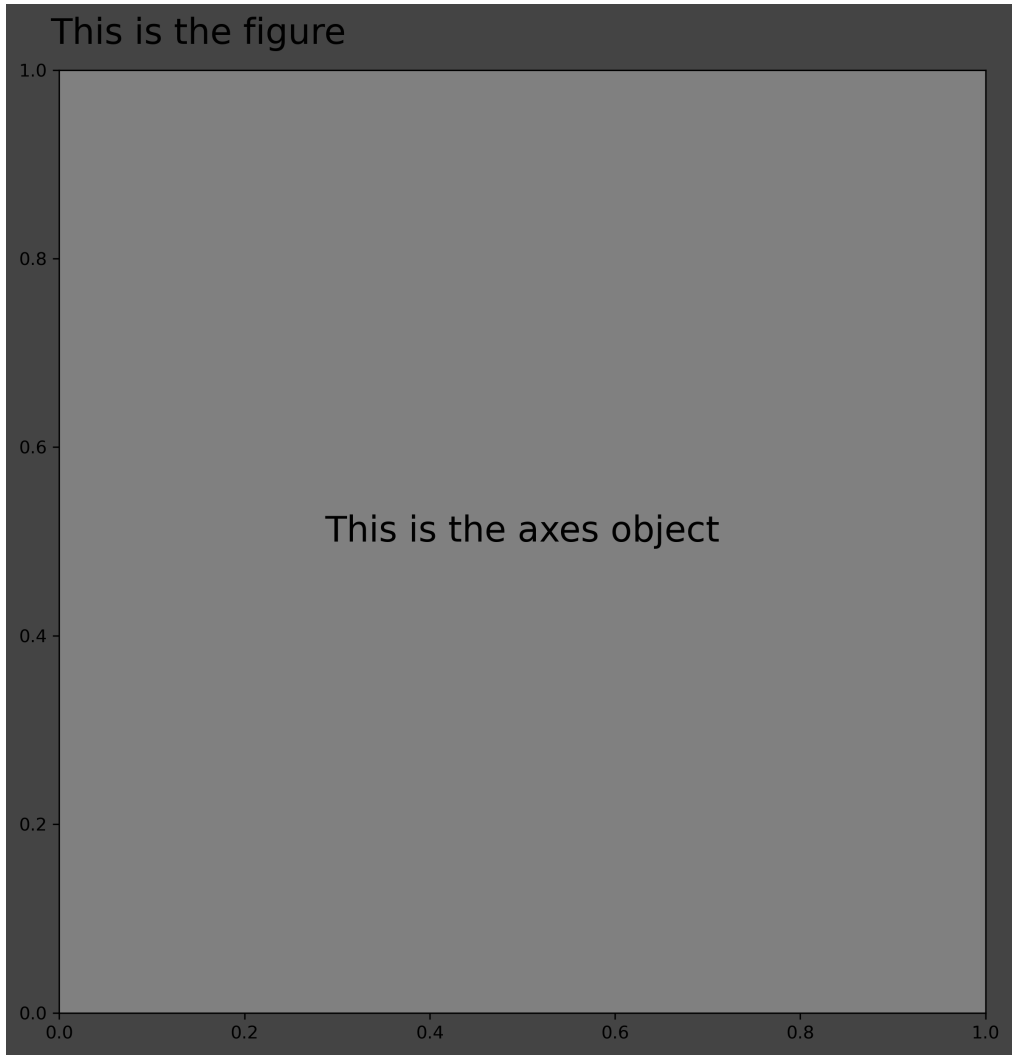
Elements of Matplotlib – What's What?



Source: https://matplotlib.org/3.1.1/gallery/showcase/anatomy.html

Elements of Matplotlib – Figures and Axes

```
fig, ax = plt.subplots(figsize=(8,8))

fig.set_facecolor("#444444")
ax.patch.set_facecolor("grey")

fig.text(0.05,1,"This is the figure", fontsize=20)
ax.text(0.5,0.5,"This is the axes object", ha="center", fontsize=20)

plt.tight_layout()
plt.savefig("../plots/figure", dpi=300, bbox_inches="tight")
plt.show()
```



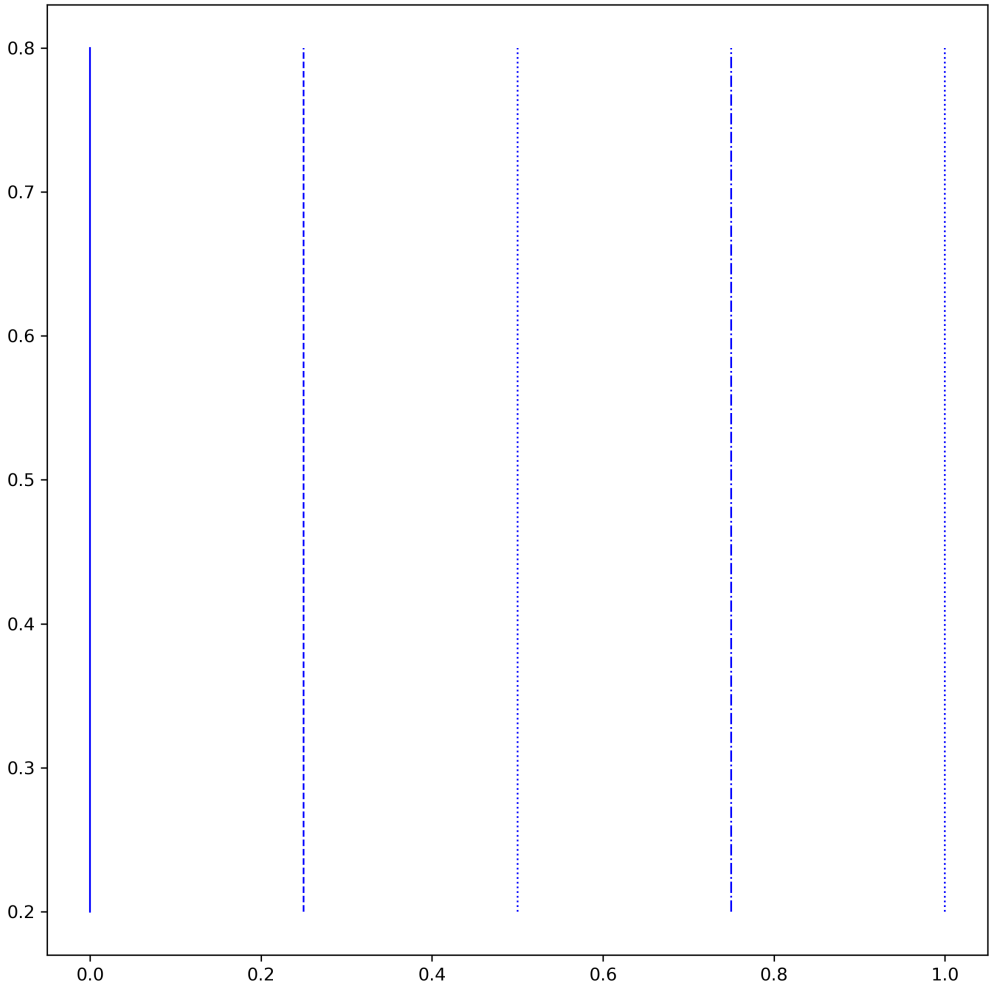
Elements of Matplotlib – Lines

We plot lines in matplotlib by using ax.plot

Lines can also take a linestyle (ls) and linewidth (lw) argument. The styles in this graphic are (from left to right)

"-", "--", "dotted", "-." ,":"

you can find more information on linestyles here: https://matplotlib.org/gallery/lines_bars_and_markers/linestyles.html?highlight=linestyle



Elements of Matplotlib – Lines

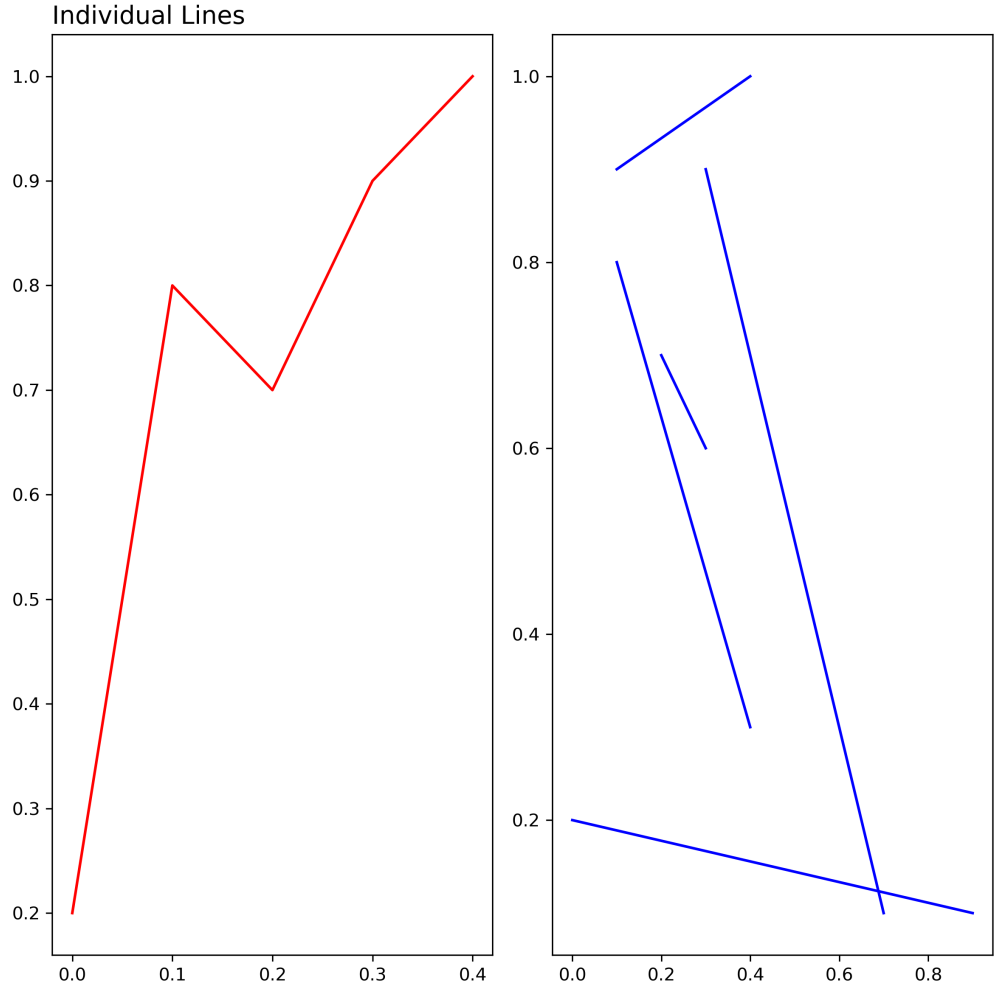
There are two main ways we can plot lines:

If we have a set of x,y coordinates we can plot them together with ax.plot(x,y), which will draw a connected series of lines:

```
x = [0,0.1,0.2,0.3,0.4]
y = [0.2,0.8,0.7,0.9,1]

ax.plot(x,y,color="red")
```

If we want to plot individual lines with starting and ending x and y values, we can loop through those values:

```
# using x and y from the above
ex = [0.9,0.4,0.3,0.7,0.1]
ey = [0.1,0.3,0.6,0.1,0.9]

for x,y,ex,ey in zip(x,y,ex,ey):
    ax1.plot([x,ex],[y,ey], color="blue")
```



Elements of Matplotlib – Markers

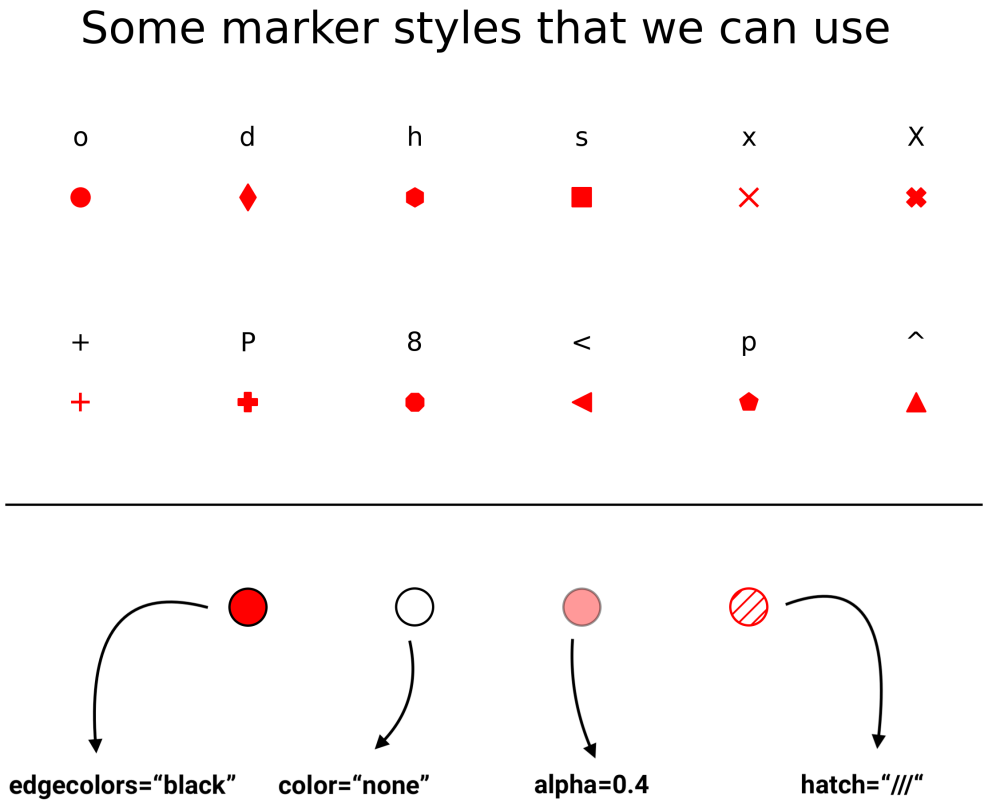
There are many markers we can use. We set these by using the marker argument:

ax.scatter(x, y, marker="o") would create a circular marker, for example.

Along with the shape, we can set the size of our markers with the "s" argument, we can set the transparency with "alpha" (which **must** be a value between 0 and 1 with 0 being invisible, and 1 being a solid colour).

We can also add an edgecolor with the "edgecolors" argument. We can also adjust the linestyle and linewidth of these as we have seen with lines.

More hatch styles can be found here https://matplotlib.org/2.0.2/examples/pylab_examples/hatch_demo.html



https://matplotlib.org/api/markers_api.html#module-matplotlib.markers

Elements of Matplotlib – Ticks and Spines

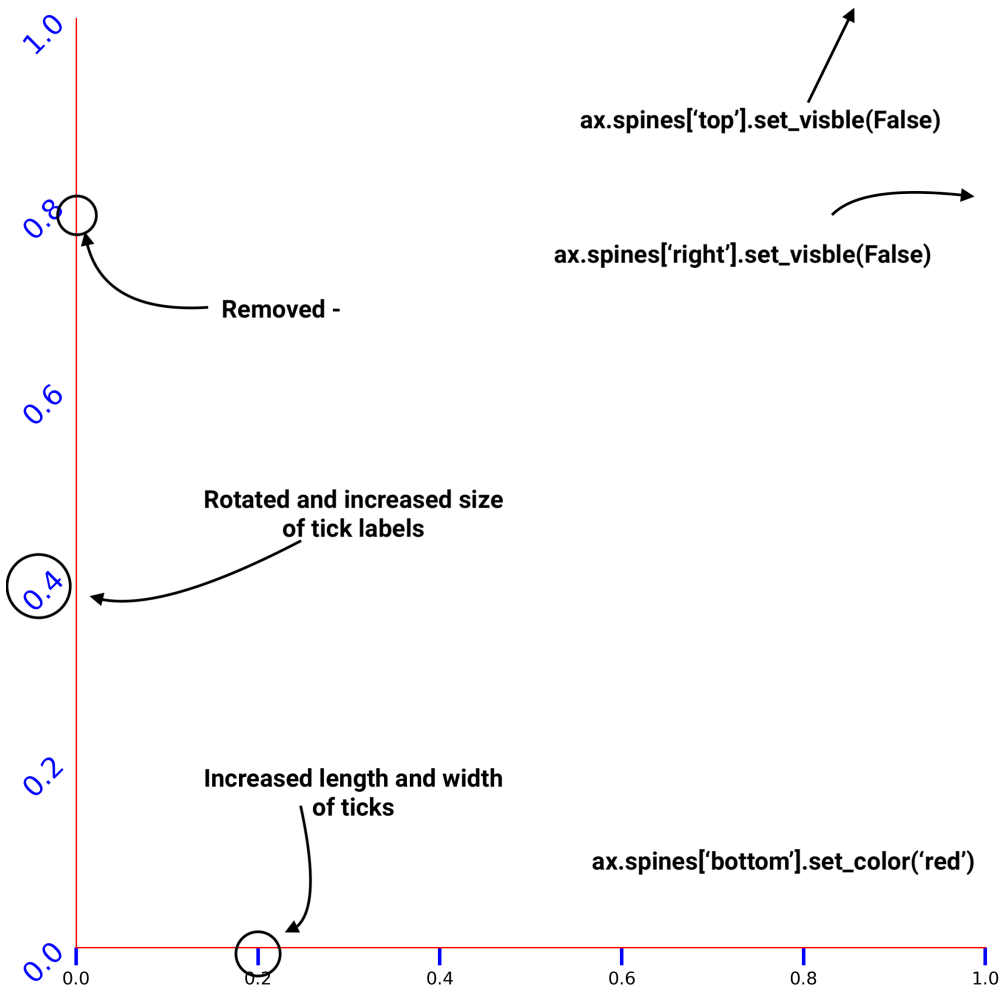
Spines can be accessed by using their location – top, right, bottom, left. For example, to hide the top spine, we use

ax.spines['top'].set_visible(False)

Ticks are accessed in a few ways. We can use ax.set_xticks to set the values we want, ax.set_xticklabels to change the labels on those values as well as the color, font, and fontsize. We can also use tick_params to handle most of that. From our example on the right:

```
ax.tick_params(axis="x",color="blue",length=10, width=2)

ax.tick_params(axis="y",length=0, labelcolor="blue",
labelsize=16, labelrotation=45)
```

We can also pass a `"which"` argument to adjust the major or minor ticks, or both.



https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.tick_params.html

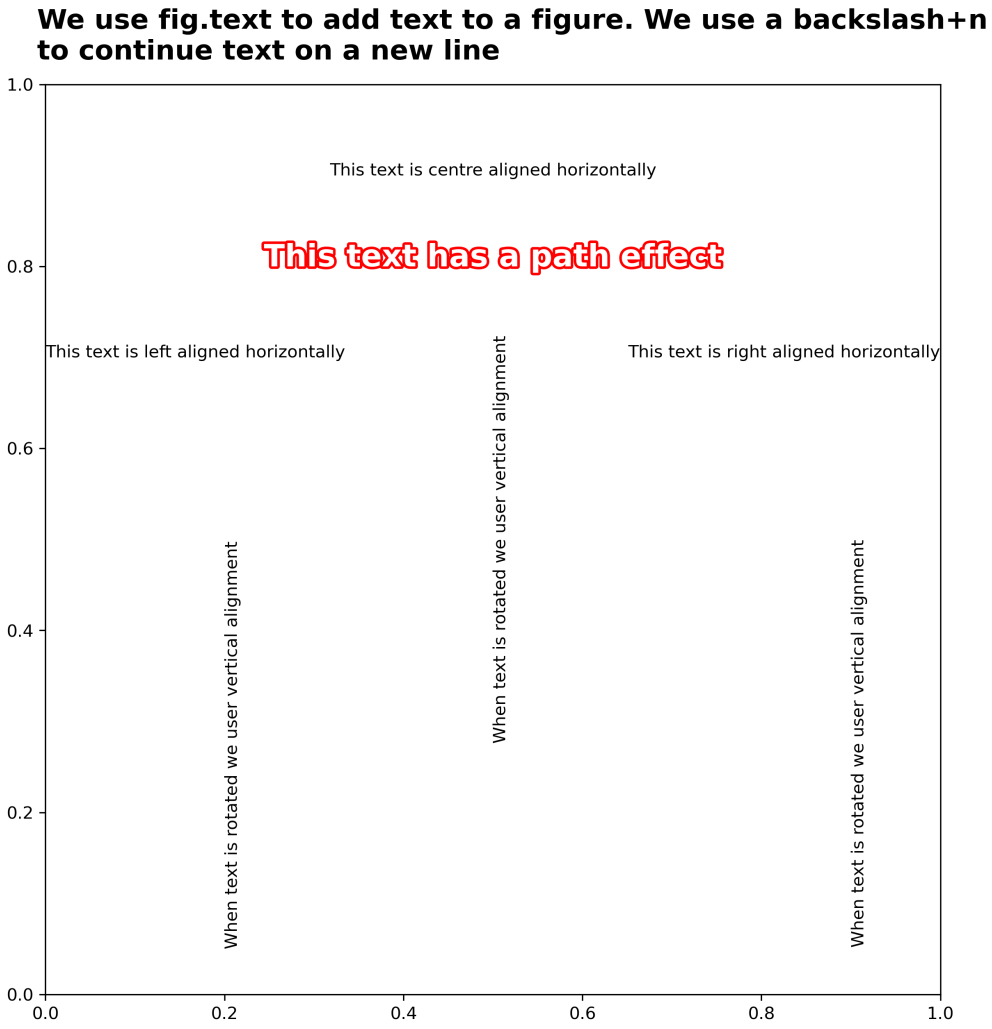
Elements of Matplotlib – Text

We use \n within a text string to continue text on a new line

We can align our text both horizontally and vertically. `ha` accepts `"center", "left", "right"`

`va` accepts `"center", "top", "bottom"`

We can use `fontfamily` to change our font

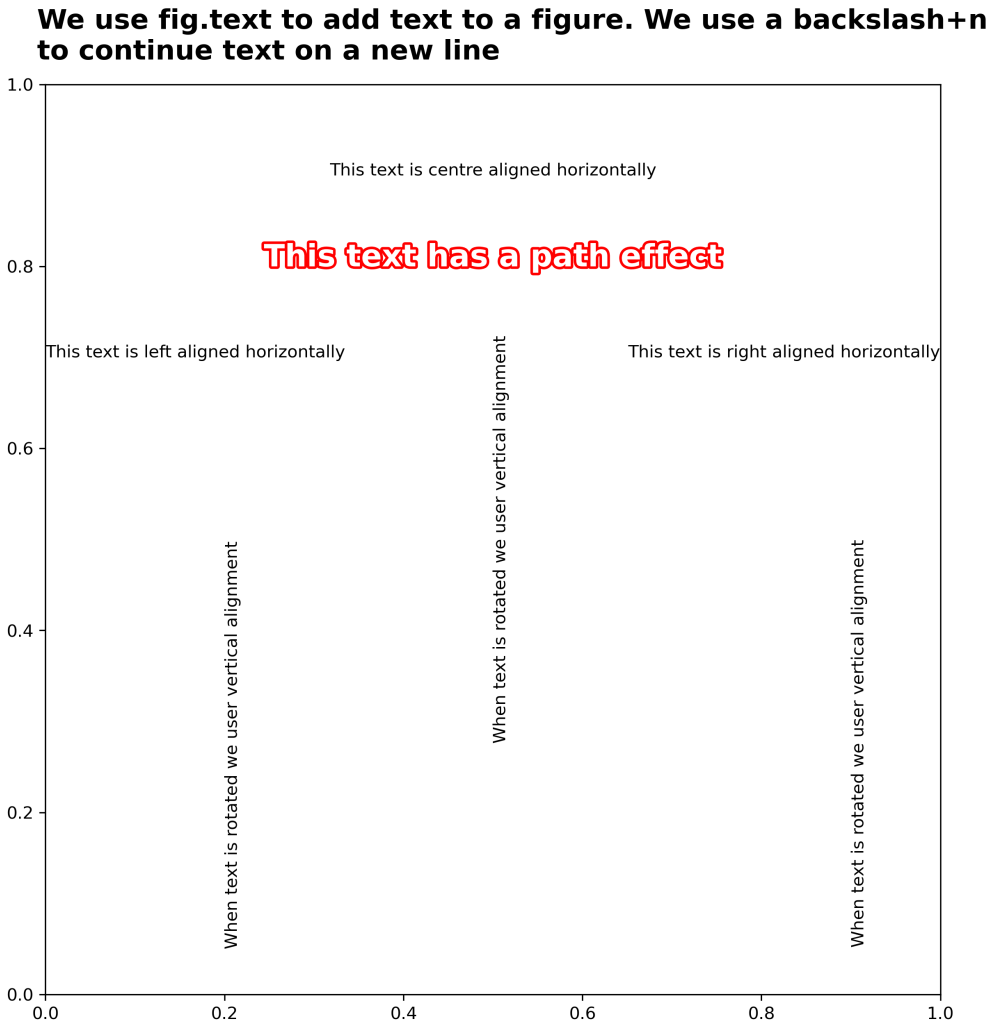


Elements of Matplotlib – Text

We can also set the **weight** and *style* of our fonts.

fontweight controls how heavy or light or text is. We can pass a range of options here, but the most common are light, medium, and bold

fontstyle controls the style of our text. We can pass italic, regular, or oblique

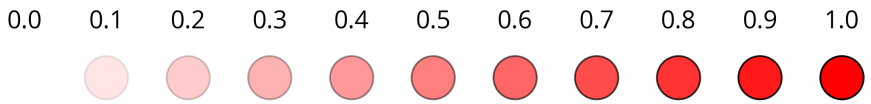


Elements of Matplotlib – Alpha

Alpha controls the transparency of our viz elements. It can be applied to any element in our visualisations.

Alpha values must be between 0 and 1, if outside of this range, you will receive an error when attempting to save certain file types (svg)

alpha values



Under the Hood: rcParams

- Currently over 300 params that can be set.
- We do not need to use all of these when customising our visualisations.
- Stylesheets exist that we can use, for example we can use a fivethirtyeight or seaborn style.
- We can adjust these by using

```
mpl.rcParams[<param_name>] = value
```

- We can revert to default matplotlib by using

```
mpl.rcParams.update(mpl.rcParamsDefault)
```

```
In [2]: mpl.rcParams

Out[2]: RcParams({'_internal.classic_mode': False,
                  'agg.path.chunksize': 0,
                  'animation.avconv_args': [],
                  'animation.avconv_path': 'avconv',
                  'animation.bitrate': -1,
                  'animation.codec': 'h264',
                  'animation.convert_args': [],
                  'animation.convert_path': 'convert',
                  'animation.embed_limit': 20.0,
                  'animation.ffmpeg_args': [],
                  'animation.ffmpeg_path': 'ffmpeg',
                  'animation.frame_format': 'png',
                  'animation.html': 'none',
                  'animation.html_args': [],
                  'animation.writer': 'ffmpeg',
                  'axes.autolimit_mode': 'data',
                  'axes.axisbelow': 'line',
                  'axes.edgecolor': 'black',
                  'axes.facecolor': 'white',
                  'axes.formatter.limits': [-5, 6],
                  'axes.formatter.min_exponent': 0,
                  'axes.formatter.offset_threshold': 4,
                  'axes.formatter.use_locale': False,
                  'axes.formatter.use_mathtext': False,
                  'axes.formatter.useoffset': True,
                  'axes.grid': False,
                  'axes.grid.axis': 'both',
                  'axes.grid.which': 'major',
                  'axes.labelcolor': 'black',
                  'axes.labelpad': 4.0,
                  'axes.labelsize': 'medium',
                  'axes.labelweight': 'normal',
                  'axes.linewidth': 0.8,
                  'axes.prop_cycle': cycler('color', ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b',
'#e377c2', '#7f7f7f', '#bcbd22', '#17becf']),
                  'axes.spines.bottom': True,
                  'axes.spines.left': True,
                  'axes.spines.right': True,
                  'axes.spines.top': True,
                  'axes.titlecolor': 'auto',
                  'axes.titlelocation': 'center',
                  'axes.titlepad': 6.0,
                  'axes.titlesize': 'large',
                  'axes.titleweight': 'normal',
                  'axes.titley': None,
                  'axes.unicode_minus': True,
                  'axes.xmargin': 0.05,
                  'axes.ymargin': 0.05,
```

Under the Hood: rcParams

The main params we are interested in are:

- axes.facecolor
- axes.labelcolor
- figure.facecolor
- font.family
- xtick.color
- xtick.labelsize
- ytick.color
- ytick.labelsize

Fonts **fonts** fonts *fonts*

```
import matplotlib.font_manager
from IPython.core.display import HTML

def make_html(fontname,typetest_string):

    return "<p><b>{font}:</b> <br/><span style='font-
family:{font}; font-
size: 20px;'>{typetest_string}</p><br>".format(typetes
t_string=typetest_string,font=fontname)

typetest_string = "The quick brown fox jumps over the
lazy dog”

code = "\n".join([make_html(font, typetest_string) for
 font in sorted(set([f.name for f in matplotlib.font_m
anager.fontManager.ttflist]))])


HTML("<div style='column-
count: 2;'>{}</div>".format(code))
```

American Typewriter:
The quick brown fox jumps over the lazy dog

Andale Mono:
The quick brown fox jumps over the lazy dog

Apple Braille:
The quick brown fox jumps over the lazy dog

Apple Chancery:
The quick brown fox jumps over the lazy dog

Apple SD Gothic Neo:
The quick brown fox jumps over the lazy dog

Apple Symbols:
The quick brown fox jumps over the lazy dog

AppleGothic:
The quick brown fox jumps over the lazy dog

AppleMyungjo:
The quick brown fox jumps over the lazy dog

Arial:
The quick brown fox jumps over the lazy dog

Arial Black:
The quick brown fox jumps over the lazy dog

Arial Hebrew:
The quick brown fox jumps over the lazy dog

Kulim Park:
The quick brown fox jumps over the lazy dog

Lao MN:
The quick brown fox jumps over the lazy dog

Lao Sangam MN:
The quick brown fox jumps over the lazy dog

Lucida Grande:
The quick brown fox jumps over the lazy dog

Luminari:
The quick brown fox jumps over the lazy dog

Luxi Mono:
The quick brown fox jumps over the lazy dog

Luxi Sans:
The quick brown fox jumps over the lazy dog

Luxi Serif:
The quick brown fox jumps over the lazy dog

Malayalam MN:
The quick brown fox jumps over the lazy dog

Malayalam Sangam MN:
The quick brown fox jumps over the lazy dog

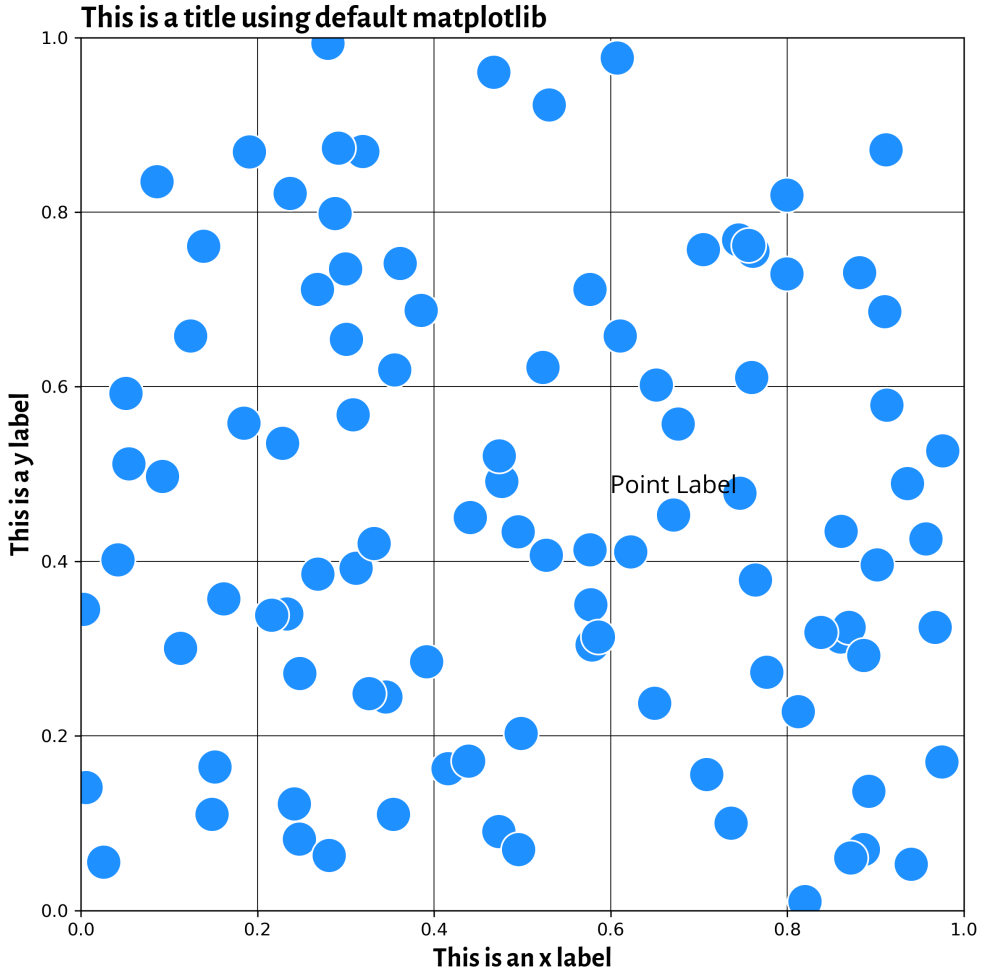
Marion:
The quick brown fox jumps over the lazy dog

Marker Felt:
The quick brown fox jumps over the lazy dog

Changing fonts

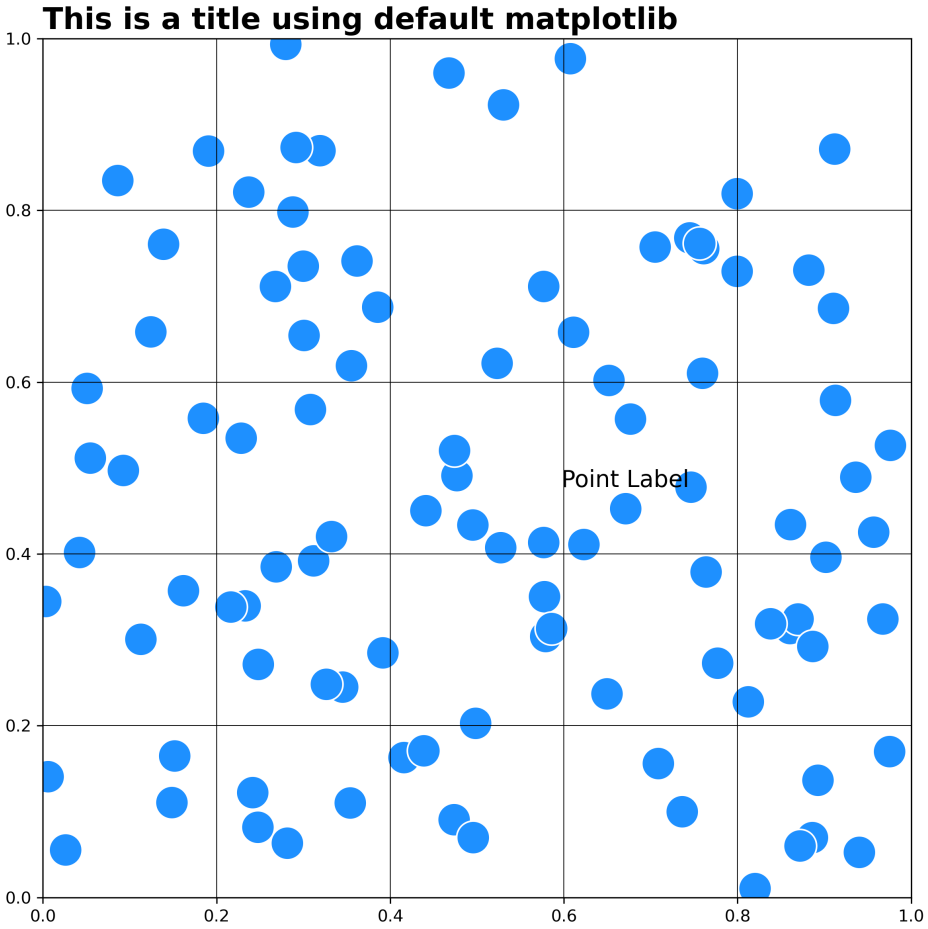
```
title_font = "Alegreya Sans"
main_font = "Open Sans"

fig, ax = plt.subplots(figsize=(8,8))
fig.set_facecolor("w")
ax.patch.set_facecolor("w")

np.random.seed(402)
x = np.random.uniform(0,1,100)
y = np.random.uniform(0,1,100)

ax.set_title("This is a title using default matplotlib",
             loc="left",
             fontweight="bold",
             fontsize=18,
             fontfamily=title_font)

ax.grid(lw=0.5, color="k", zorder=1)

ax.scatter(x,y,
           color="dodgerblue",
           edgecolors="w",
           linewidths=1,
           s=400, zorder=2)

ax.text(x[19],
        y[19]+0.025,
        "Point Label",
        ha="center",
        color="k", fontsize=14,
        fontweight="medium", zorder=3,
        fontfamily=main_font)
```

```
# continued …

ax.set_xlabel("This is an x label", fontweight="bold", fontfamily=title_font, size=16)
ax.set_ylabel("This is a y label", fontweight="bold", fontfamily=title_font, size=16)
ax.set_xlim(0,1)
ax.set_ylim(0,1)

plt.show()
```



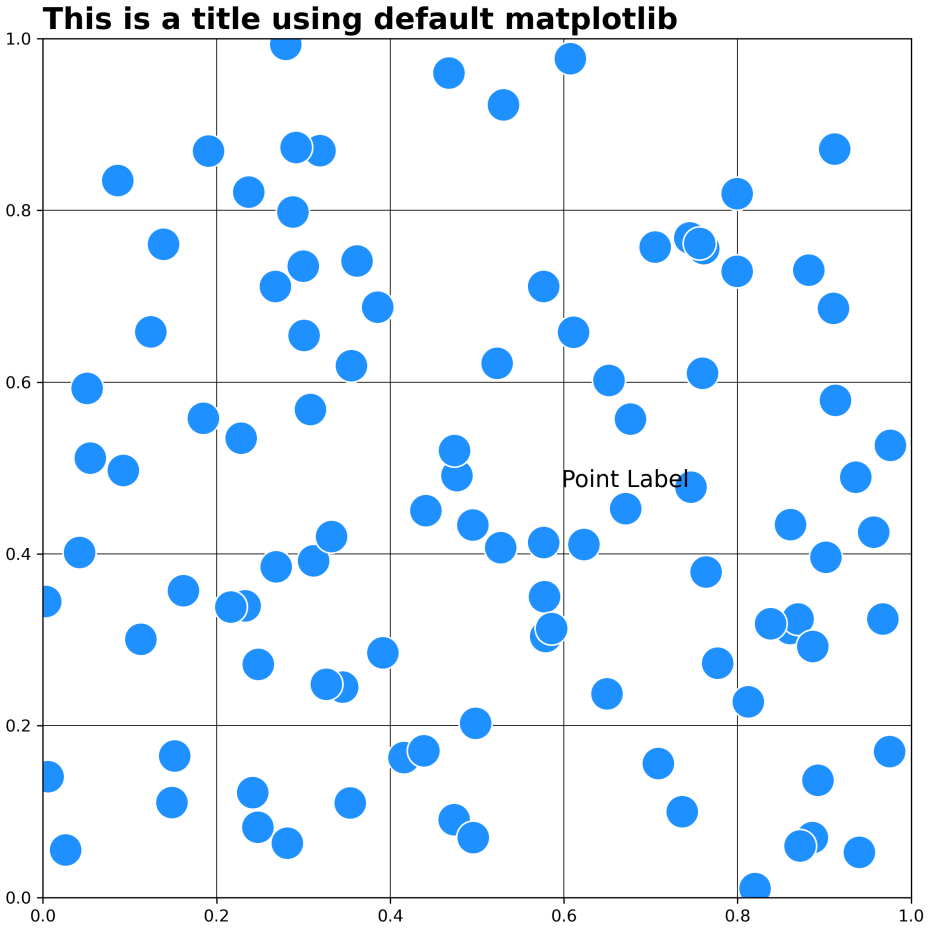
The folly of not using zorder

```
fig, ax = plt.subplots(figsize=(8,8))

np.random.seed(402)
x = np.random.uniform(0,1,100)
y = np.random.uniform(0,1,100)

ax.set_title("This is a title using a default matplotlib",
             loc="left", fontweight="bold",
             fontsize=18)

ax.grid(lw=0.5, color="k")

ax.scatter(x,y,
           color="dodgerblue",
           edgecolors="w",
           linewidths=1,
           s=400)

ax.text(x[19],y[19]+0.025,
        "Point Label",
        ha="center", color="k", fontsize=14,
        fontweight="medium")

ax.set_xlim(0,1)
ax.set_ylim(0,1)

plt.tight_layout()
plt.show()
```



The folly of not using zorder

```
fig, ax = plt.subplots(figsize=(8,8))

np.random.seed(402)
x = np.random.uniform(0,1,100)
y = np.random.uniform(0,1,100)

ax.set_title("This is a title using a default matplotlib",
             loc="left", fontweight="bold",
             fontsize=18)

ax.grid(lw=0.5, color="k", zorder=1)

ax.scatter(x,y,
           color="dodgerblue",
           edgecolors="w",
           linewidths=1,
           s=400, zorder=2)

ax.text(x[19],y[19]+0.025,
        "Point Label",
        ha="center", color="k", fontsize=14,
        fontweight="medium", zorder=3)

ax.set_xlim(0,1)
ax.set_ylim(0,1)

plt.tight_layout()
plt.show()
```



Layering objects with zorder

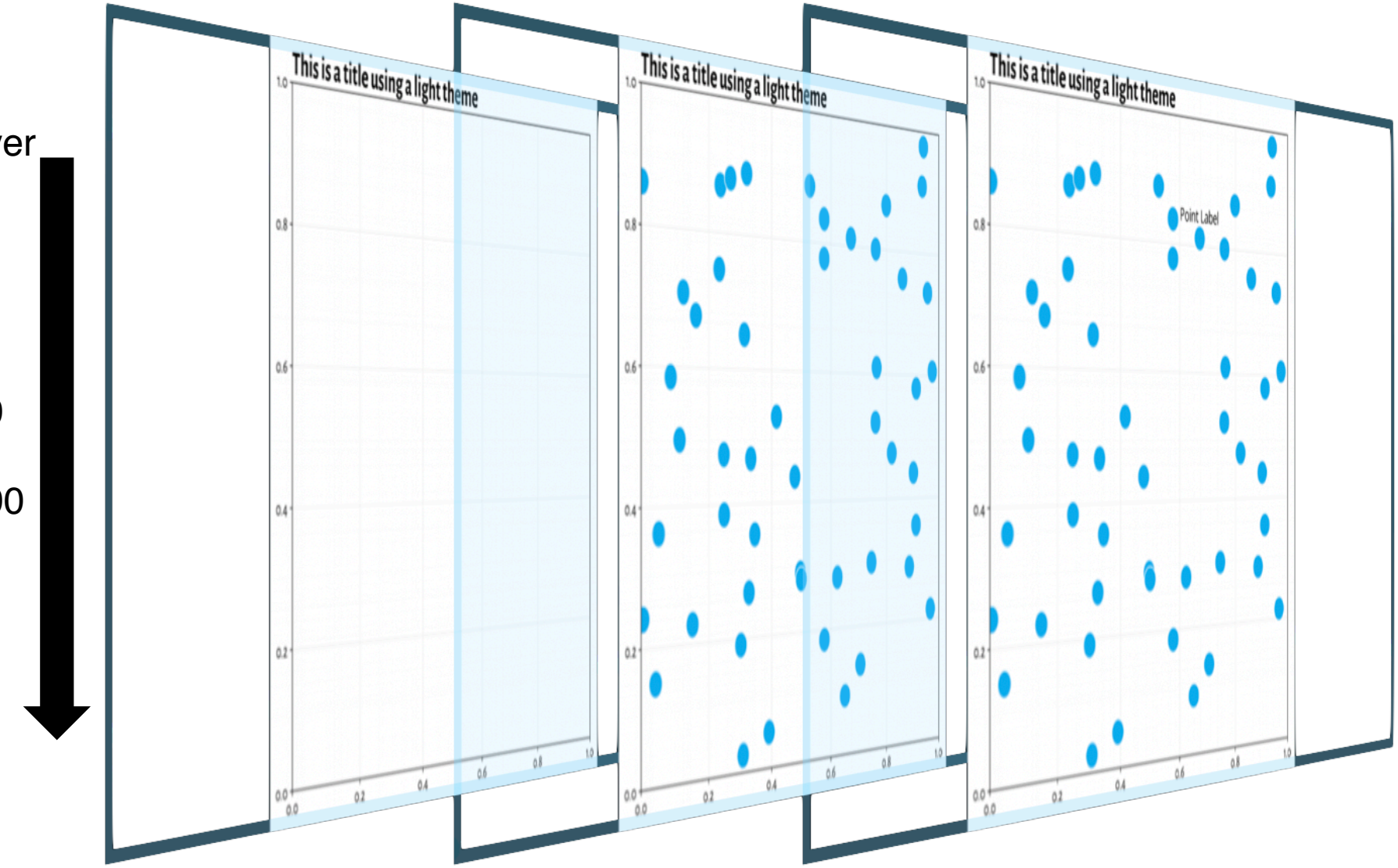
Bottom Layer
zorder = 1
zorder = 2
…
…

zorder = 10

zorder = 500

…
…

Top Layer



Saving Visualisations

When saving visualisations, always be mindful of how the margins will appear in your saved graphic. If we are working in a notebook and would like to save our graphic in a plots folder (outside of a notebooks folder) we could use the following. Note, I have also set the dpi to 300, this insures the quality of saved image is of a high enough standard:

```
plt.savefig("../plots/this-is-a-cool-viz",
bbox_inches="tight", dpi=300)
```

We can also save svg files by adding ".svg" to the end of our file name - this-is-a-cool-viz.svg

A real world example

A real world example: packages used

```
import pandas as pd
import numpy as np

import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.patheffects as path_effects

from PIL import Image
import requests
from io import BytesIO


from highlight_text import htext
```

A real world example: the data

```
df = pd.read_csv("../data/pl_goal_summary_data.csv")
df.head(20)
```

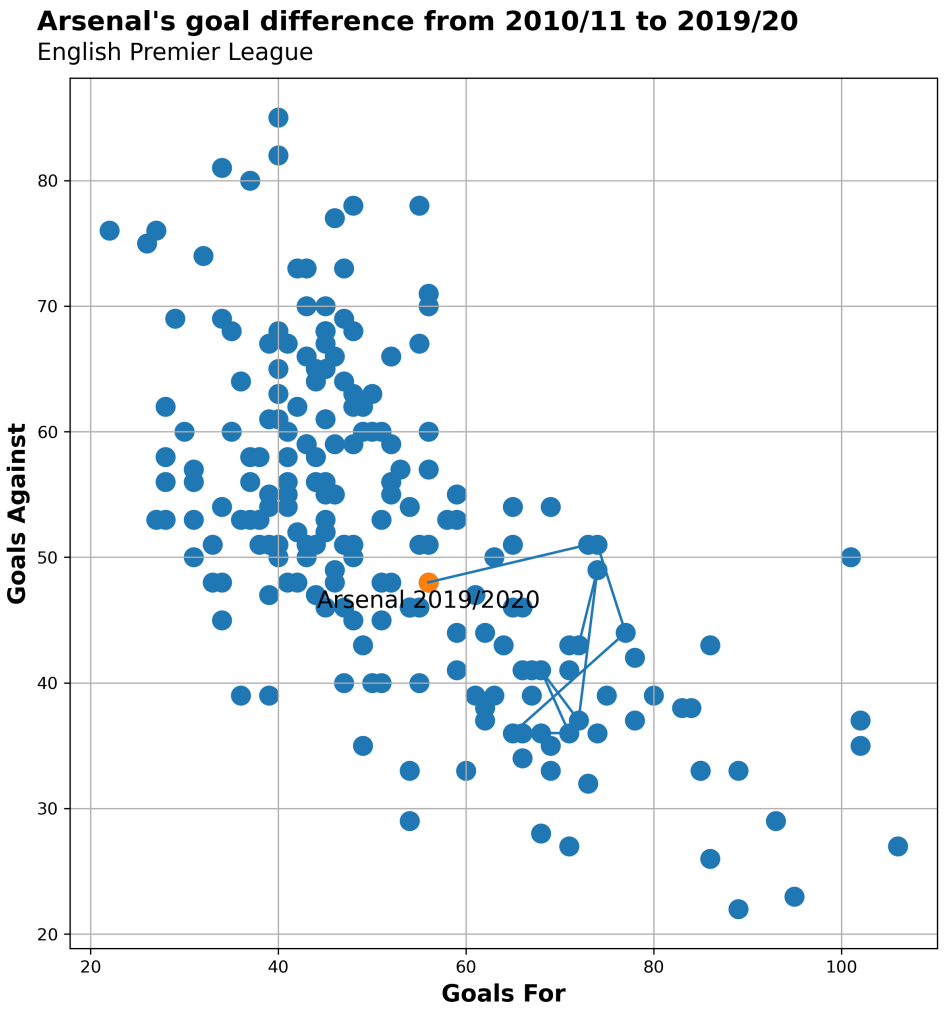
- A dataset showing team goals for and against per season in the Premier League from 2010/11 to 2019/20
- Dataset is from analysis of football-data.co.uk's freely available match level data
- The data can be found in the data folder of the GitHub repo that accompanies this presentation

https://github.com/petermckeeverPerform/friends-of-tracking-viz-lecture

season_id	team	goals_for	goals_against	goal_diff	team_season
2010	Arsenal	72.0	43.0	29.0	Arsenal 2010/2011
2010	Aston Villa	48.0	59.0	-11.0	Aston Villa 2010/2011
2010	Birmingham	37.0	58.0	-21.0	Birmingham 2010/2011
2010	Blackburn	46.0	59.0	-13.0	Blackburn 2010/2011
2010	Blackpool	55.0	78.0	-23.0	Blackpool 2010/2011
2010	Bolton	52.0	56.0	-4.0	Bolton 2010/2011
2010	Chelsea	69.0	33.0	36.0	Chelsea 2010/2011
2010	Everton	51.0	45.0	6.0	Everton 2010/2011
2010	Fulham	49.0	43.0	6.0	Fulham 2010/2011
2010	Liverpool	59.0	44.0	15.0	Liverpool 2010/2011
2010	Man City	60.0	33.0	27.0	Man City 2010/2011
2010	Man United	78.0	37.0	41.0	Man United 2010/2011
2010	Newcastle	56.0	57.0	-1.0	Newcastle 2010/2011
2010	Stoke	46.0	48.0	-2.0	Stoke 2010/2011
2010	Sunderland	45.0	56.0	-11.0	Sunderland 2010/2011
2010	Tottenham	55.0	46.0	9.0	Tottenham 2010/2011
2010	West Brom	56.0	71.0	-15.0	West Brom 2010/2011
2010	West Ham	43.0	70.0	-27.0	West Ham 2010/2011
2010	Wigan	40.0	61.0	-21.0	Wigan 2010/2011
2010	Wolves	46.0	66.0	-20.0	Wolves 2010/2011

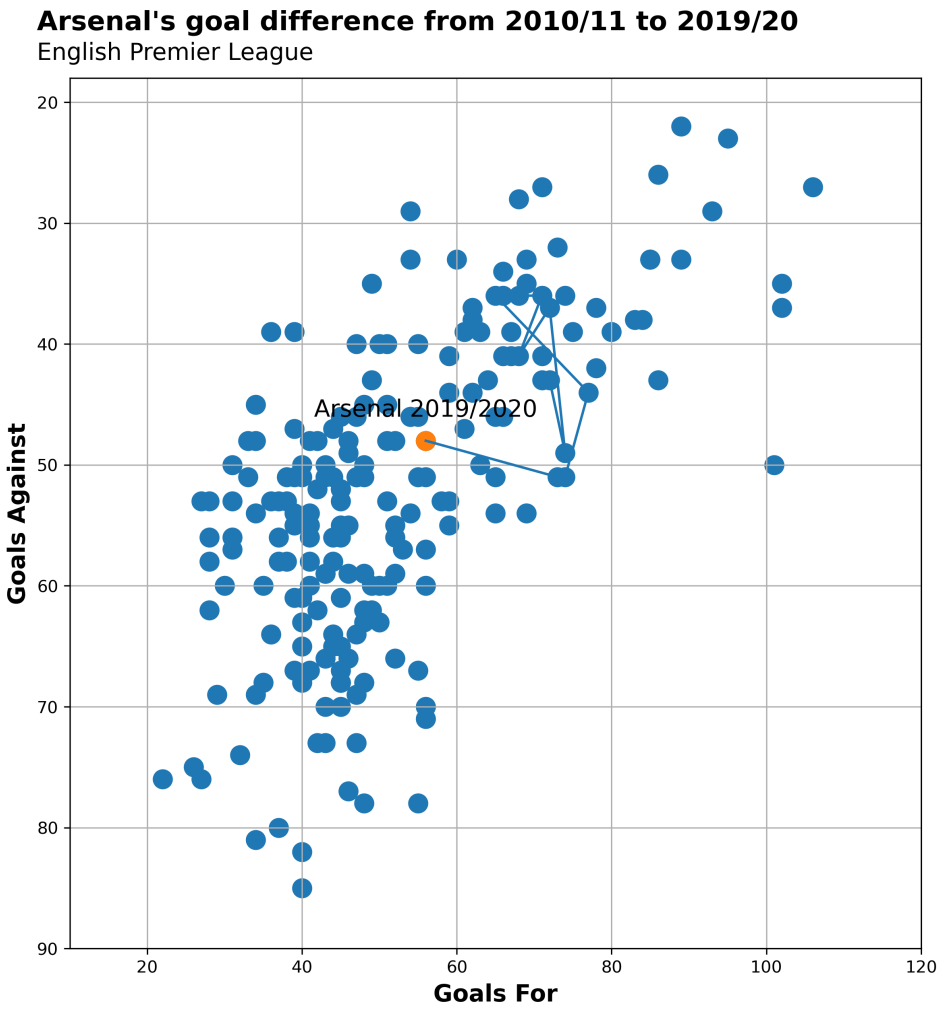
A real world example: the initial plot

```python
# selectiing team
team = "Arsenal"

# creating figure and axes
fig, ax = plt.subplots(figsize=(8,8))
fig.set_facecolor("w")

# adding a grid
ax.grid()

# getting our data to plot

x = df.goals_for.tolist()
y = df.goals_against.tolist()

ax.scatter(x,y,s=120)

# getting data for specified team
x = df[df.team == team].goals_for.tolist()
y = df[df.team == team].goals_against.tolist()
t = df[df.team == team].team_season.tolist()

# plotting a line to track team's change over seasons
ax.plot(x,y)

# plotting team's most recent season
ax.scatter(x[-1],y[-1],s=120)
t = ax.text(x[-1],y[-1]-2,t[-1],color='k',fontsize=14, ha="center")

# adding title and subtitle
fig.text(0.05,1,"{}'s goal difference from 2010/11 to 2019/20\n".format(team), fontsize=16, fontweight="bold")
fig.text(0.05,1,"English Premier League",fontweight="regular", fontsize=14)

# adding x and y labels
ax.set_xlabel("Goals For", fontweight="bold", fontsize=14)
ax.set_ylabel("Goals Against", fontweight="bold", fontsize= 14)

plt.tight_layout()
plt.show()
```



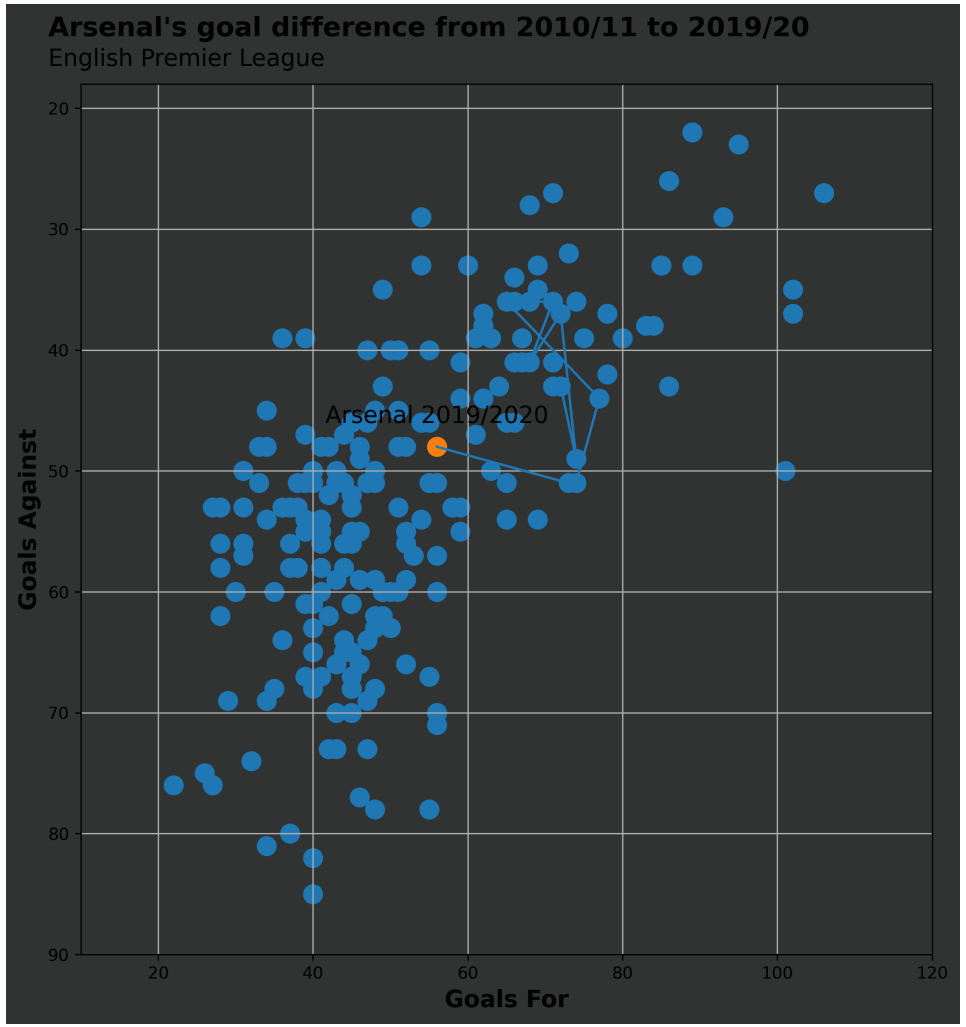
A real world example: reverse y and set lims

```python
# selecting team
team = "Arsenal"

# creating figure and axes
fig, ax = plt.subplots(figsize=(8,8))
fig.set_facecolor("w")

# adding a grid
ax.grid()

# getting our data to plot

x = df.goals_for.tolist()
y = df.goals_against.tolist()

ax.scatter(x,y,s=120)

# getting data for specified team
x = df[df.team == team].goals_for.tolist()
y = df[df.team == team].goals_against.tolist()
t = df[df.team == team].team_season.tolist()

# plotting a line to track team's change over seasons
ax.plot(x,y)

# plotting team's most recent season
ax.scatter(x[-1],y[-1],s=120)
t = ax.text(x[-1],y[-1]-2,t[-1],color='k',fontsize=14, ha="center")

# adding title and subtitle
fig.text(0.05,1,"{}'s goal difference from 2010/11 to 2019/20\n".format(team), fontsize=16, fontweight="bold")
fig.text(0.05,1,"English Premier League",fontweight="regular", fontsize=14)

# adding x and y labels
ax.set_xlabel("Goals For", fontweight="bold", fontsize=14)
ax.set_ylabel("Goals Against", fontweight="bold", fontsize= 14)

ax.set_ylim(90,18)
ax.set_xlim(10,120)

plt.tight_layout()
```



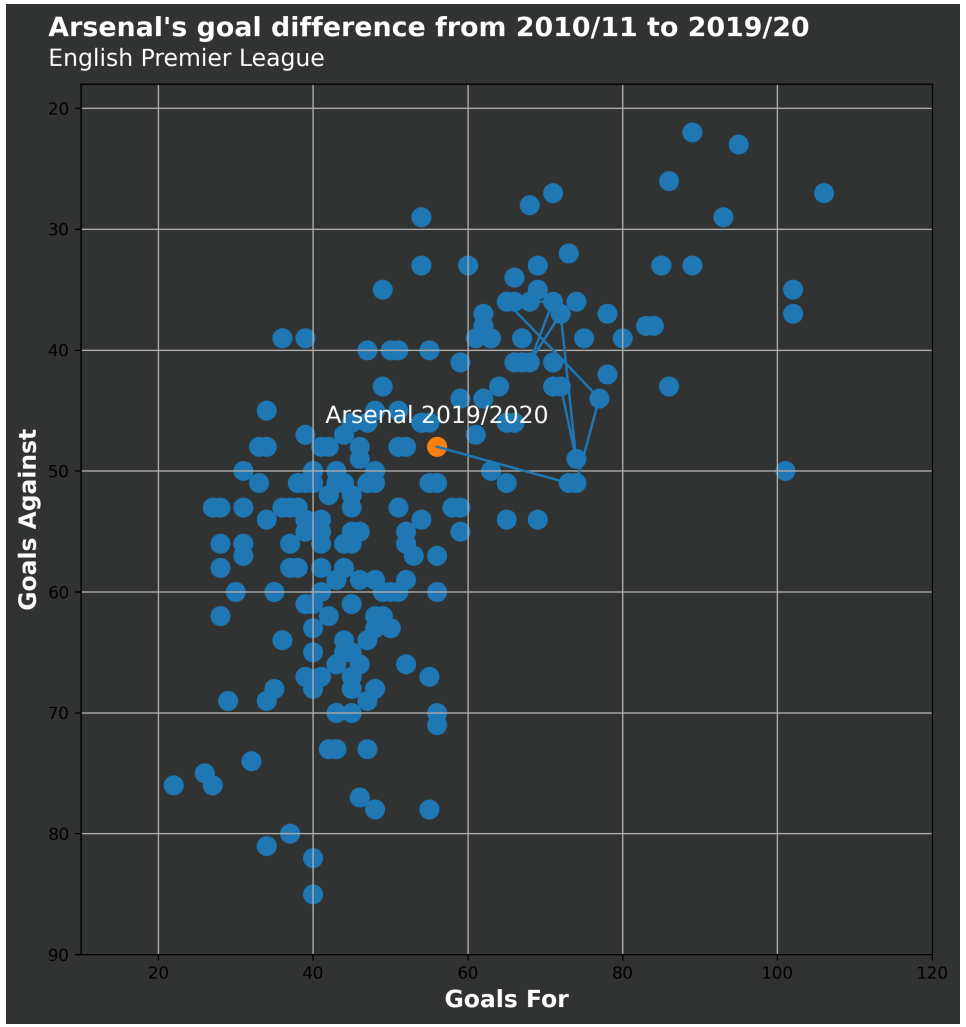
A real world example: background colour

```python
# selectiing team
team = "Arsenal"

background = "#313332"

# creating figure and axes
fig, ax = plt.subplots(figsize=(8,8))
fig.set_facecolor(background)
ax.patch.set_facecolor(background)

# adding a grid
ax.grid()

# getting our data to plot

x = df.goals_for.tolist()
y = df.goals_against.tolist()

ax.scatter(x,y,s=120)

# getting data for specified team
x = df[df.team == team].goals_for.tolist()
y = df[df.team == team].goals_against.tolist()
t = df[df.team == team].team_season.tolist()

# plotting a line to track team's change over seasons
ax.plot(x,y)

# plotting team's most recent season
ax.scatter(x[-1],y[-1],s=120)
t = ax.text(x[-1],y[-1]-2,t[-1],color='k',fontsize=14, ha="center")

# adding title and subtitle
fig.text(0.05,1,"{}'s goal difference from 2010/11 to 2019/20\n".format(team), fontsize=16, fontweight="bold")
fig.text(0.05,1,"English Premier League",fontweight="regular", fontsize=14)

# adding x and y labels
ax.set_xlabel("Goals For", fontweight="bold", fontsize=14)
ax.set_ylabel("Goals Against", fontweight="bold", fontsize= 14)

ax.set_ylim(90,18)
```



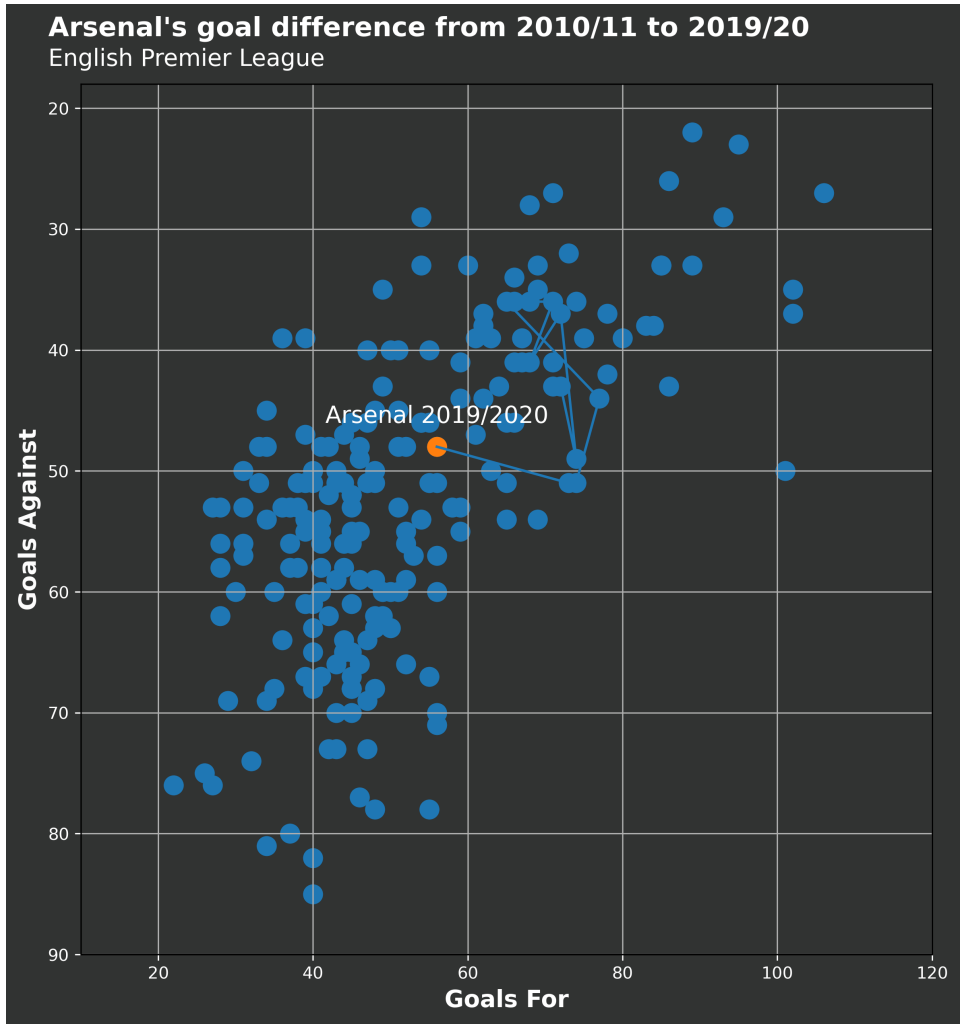
A real world example: text colour

```python
background = "#313332"
text_color = "w"


# creating figure and axes
fig, ax = plt.subplots(figsize=(8,8))
fig.set_facecolor(background)
ax.patch.set_facecolor(background)

# adding a grid
ax.grid()

# getting our data to plot

x = df.goals_for.tolist()
y = df.goals_against.tolist()

ax.scatter(x,y,s=120)

# getting data for specified team
x = df[df.team == team].goals_for.tolist()
y = df[df.team == team].goals_against.tolist()
t = df[df.team == team].team_season.tolist()

# plotting a line to track team's change over seasons
ax.plot(x,y)

# plotting team's most recent season
ax.scatter(x[-1],y[-1],s=120)
t = ax.text(x[-1],y[-1]-2,t[-1],color=text_color,fontsize=14, ha="center")

# adding title and subtitle
fig.text(0.05,1,"{}'s goal difference from 2010/11 to 2019/20\n".format(team),
         fontsize=16, fontweight="bold", color=text_color)
fig.text(0.05,1,"English Premier League",fontweight="regular", fontsize=14,
         color=text_color)

# adding x and y labels
ax.set_xlabel("Goals For", fontweight="bold", fontsize=14, color=text_color)
ax.set_ylabel("Goals Against", fontweight="bold", fontsize= 14,color=text_color)
```



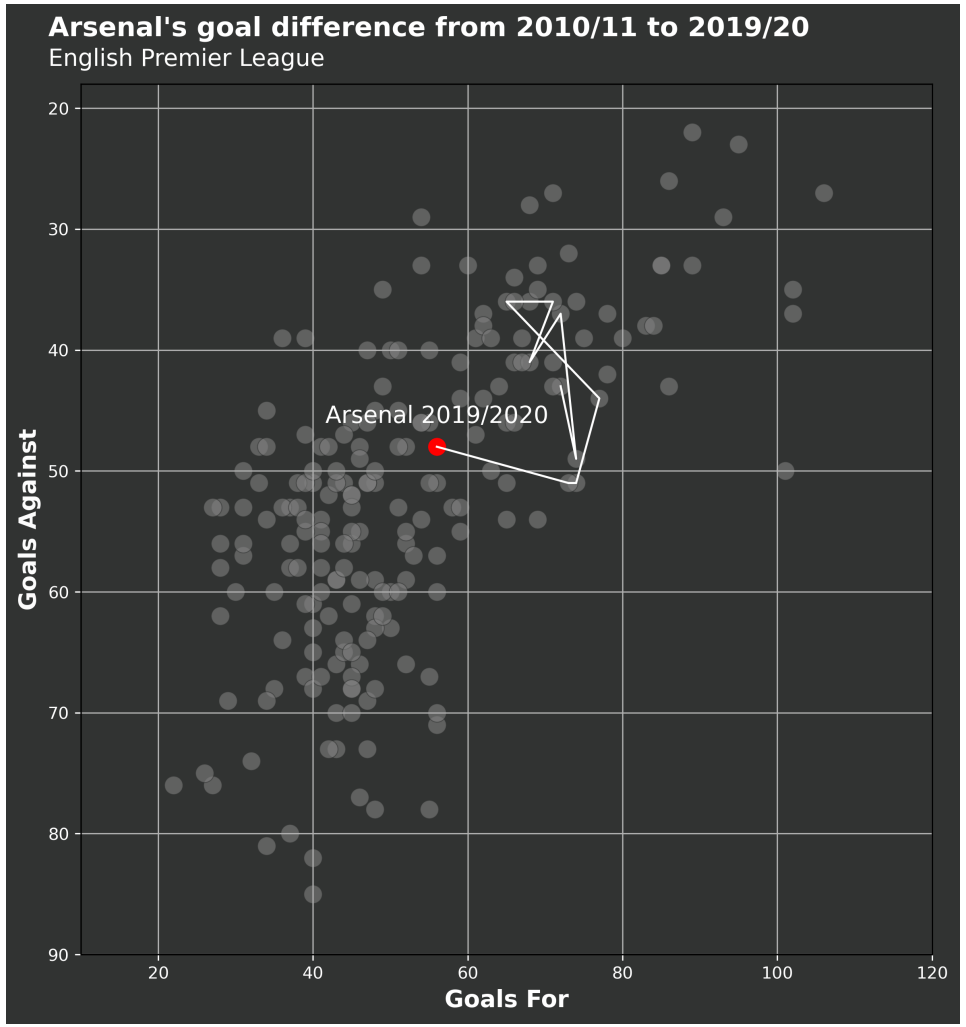
A real world example: tick colour

```python
# selectiing team
team = "Arsenal"

background = "#313332"
text_color = "w"
mpl.rcParams['xtick.color'] = text_color
mpl.rcParams['ytick.color'] = text_color


# creating figure and axes
fig, ax = plt.subplots(figsize=(8,8))
fig.set_facecolor(background)
ax.patch.set_facecolor(background)

# adding a grid
ax.grid()

# getting our data to plot

x = df.goals_for.tolist()
y = df.goals_against.tolist()

ax.scatter(x,y,s=120)

# getting data for specified team
x = df[df.team == team].goals_for.tolist()
y = df[df.team == team].goals_against.tolist()
t = df[df.team == team].team_season.tolist()

# plotting a line to track team's change over seasons
ax.plot(x,y)

# plotting team's most recent season
ax.scatter(x[-1],y[-1],s=120)
t = ax.text(x[-1],y[-1]-2,t[-1],color=text_color,fontsize=14, ha="center")

# adding title and subtitle
fig.text(0.05,1,"{}'s goal difference from 2010/11 to 2019/20\n".format(team),
         fontsize=16, fontweight="bold", color=text_color)
fig.text(0.05,1,"English Premier League",fontweight="regular", fontsize=14,
         color=text_color)
```



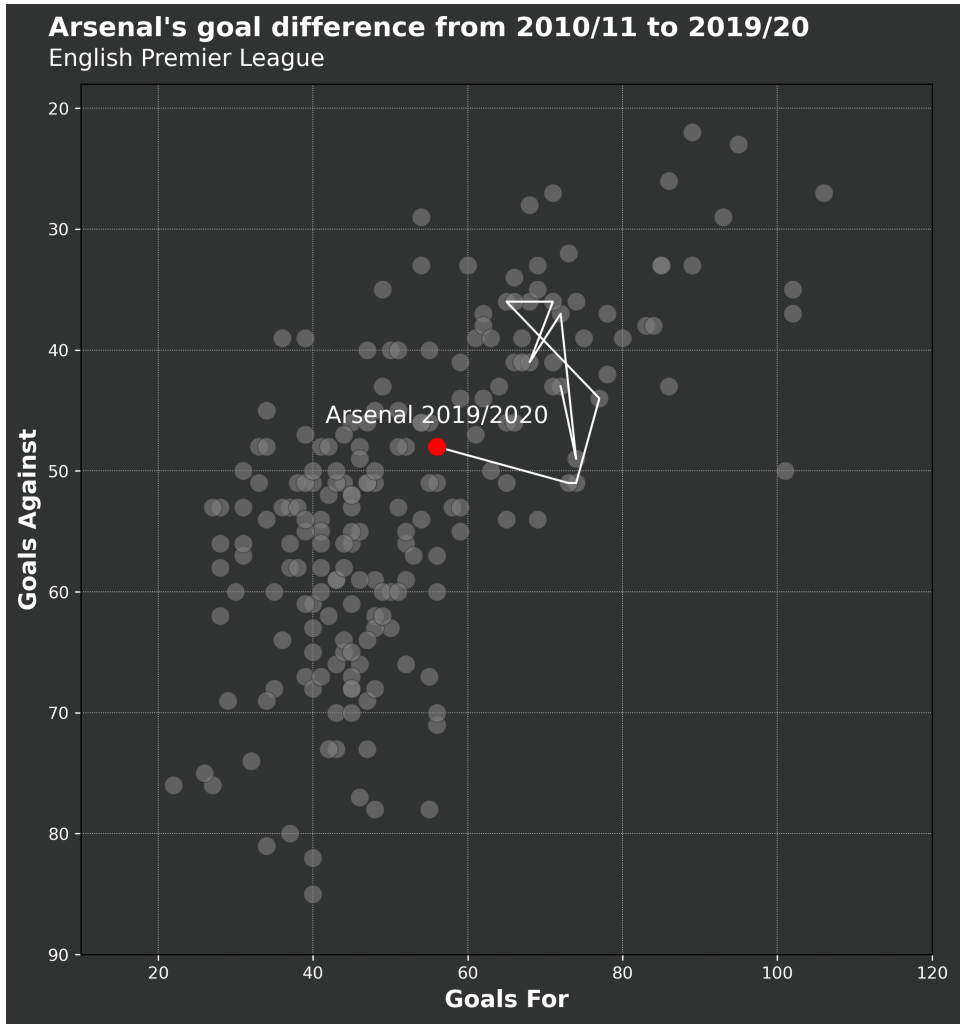
A real world example: scatter and line colours

```python
text_color = "w"
mpl.rcParams['xtick.color'] = text_color
mpl.rcParams['ytick.color'] = text_color

filler = "grey"
primary = "red"


# creating figure and axes
fig, ax = plt.subplots(figsize=(8,8))
fig.set_facecolor(background)
ax.patch.set_facecolor(background)

# adding a grid
ax.grid()

# getting our data to plot

x = df.goals_for.tolist()
y = df.goals_against.tolist()

ax.scatter(x,y,s=120,color=filler,edgecolors=background, alpha=0.6, lw=0.5)

# getting data for specified team
x = df[df.team == team].goals_for.tolist()
y = df[df.team == team].goals_against.tolist()
t = df[df.team == team].team_season.tolist()

# plotting a line to track team's change over seasons
ax.plot(x,y, color=text_color, lw=1.25)

# plotting team's most recent season
ax.scatter(x[-1],y[-1],s=120, color=primary, edgecolors=background, lw=.25,)
t = ax.text(x[-1],y[-1]-2,t[-1],color=text_color,fontsize=14, ha="center")

# adding title and subtitle
fig.text(0.05,1,"{}'s goal difference from 2010/11 to 2019/20\n".format(team),
         fontsize=16, fontweight="bold", color=text_color)
fig.text(0.05,1,"English Premier League",fontweight="regular", fontsize=14,
         color=text_color)

```



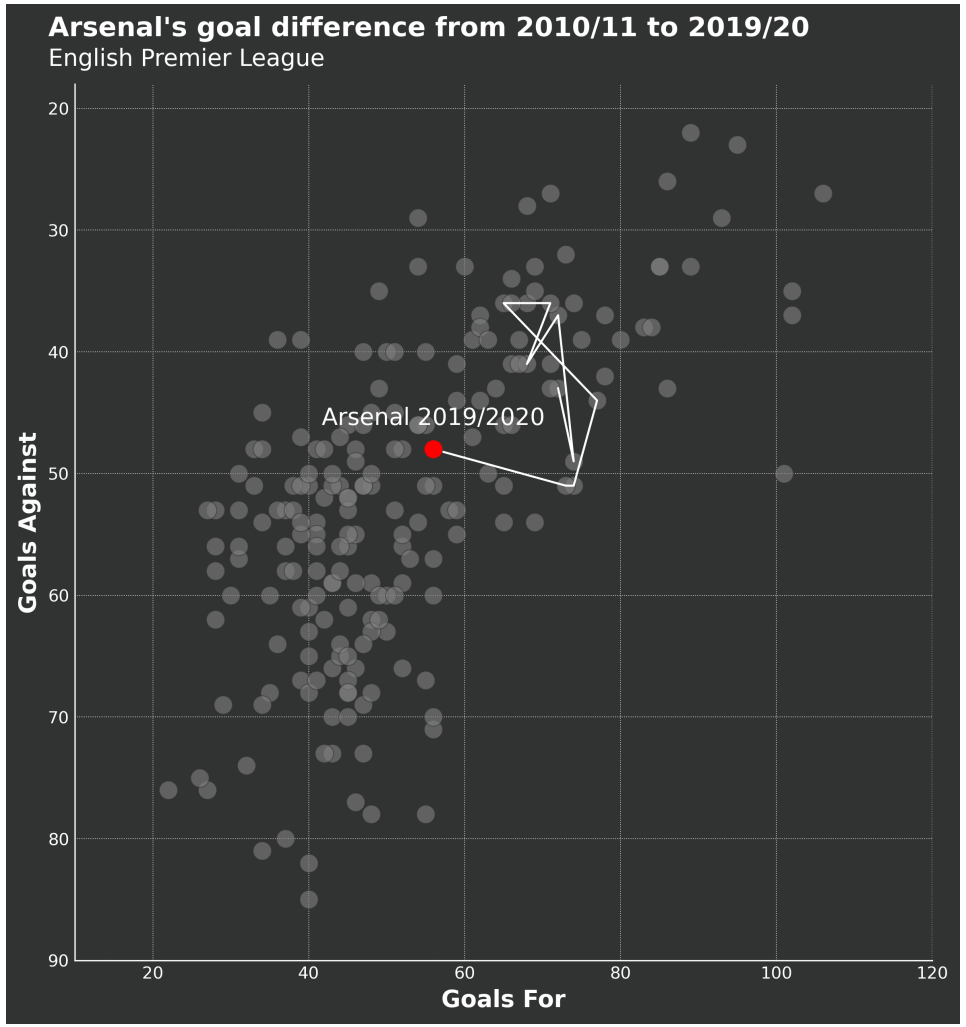
A real world example: grid and zorder

```python
primary = "red"


# creating figure and axes
fig, ax = plt.subplots(figsize=(8,8))
fig.set_facecolor(background)
ax.patch.set_facecolor(background)

# adding a grid
ax.grid(ls="dotted",lw="0.5",color="lightgrey", zorder=1)

# getting our data to plot

x = df.goals_for.tolist()
y = df.goals_against.tolist()

ax.scatter(x,y,s=120,color=filler,edgecolors=background, alpha=0.6, lw=0.5, zorder=2)

# getting data for specified team
x = df[df.team == team].goals_for.tolist()
y = df[df.team == team].goals_against.tolist()
t = df[df.team == team].team_season.tolist()

# plotting a line to track team's change over seasons
ax.plot(x,y, color=text_color, lw=1.25, zorder=3)

# plotting team's most recent season
ax.scatter(x[-1],y[-1],s=120, color=primary, edgecolors=background, lw=.25, zorder=4)
t = ax.text(x[-1],y[-1]-2,t[-1],color=text_color,fontsize=14, ha="center")

# adding title and subtitle
fig.text(0.05,1,"{}'s goal difference from 2010/11 to 2019/20\n".format(team),
         fontsize=16, fontweight="bold", color=text_color)
fig.text(0.05,1,"English Premier League",fontweight="regular", fontsize=14,
         color=text_color)

# adding x and y labels
ax.set_xlabel("Goals For", fontweight="bold", fontsize=14, color=text_color)
ax.set_ylabel("Goals Against", fontweight="bold", fontsize= 14,color=text_color)

ax.set_ylim(90,18)
```



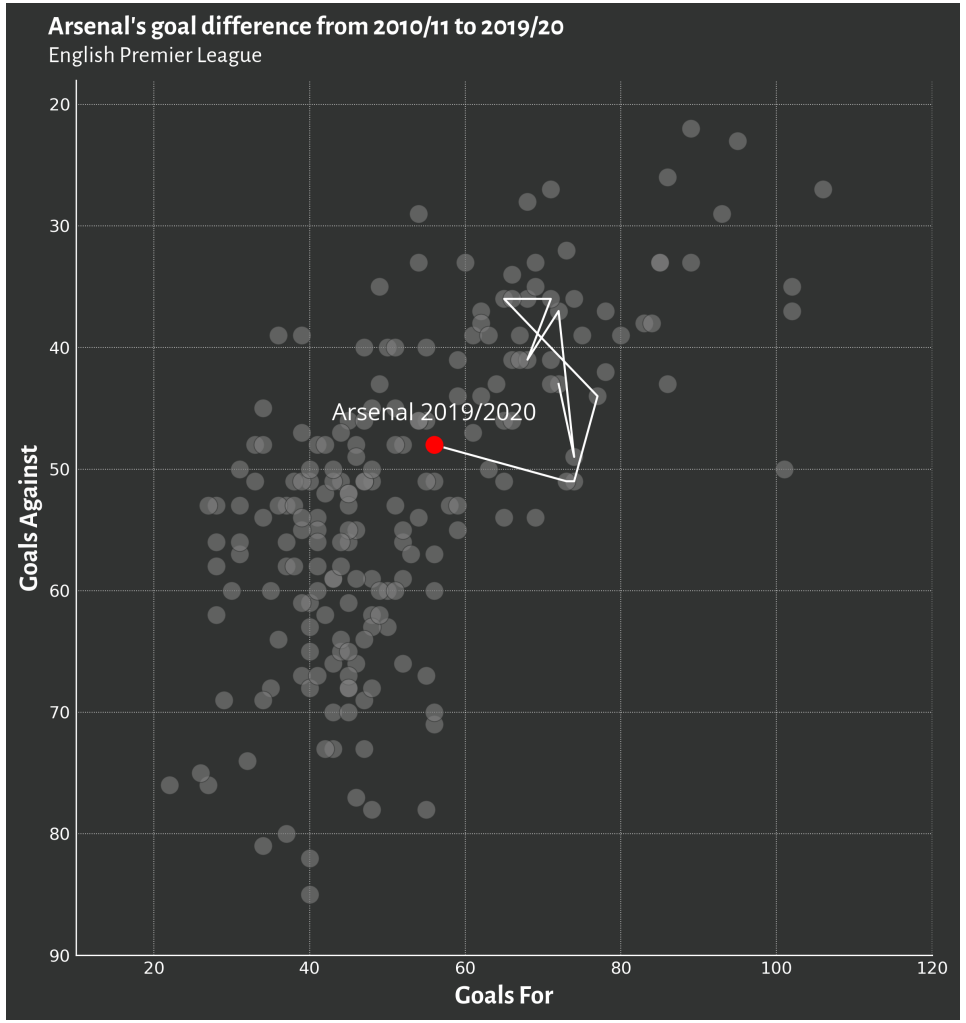
A real world example: spines and pips

```python
x = df[df.team == team].goals_for.tolist()
y = df[df.team == team].goals_against.tolist()
t = df[df.team == team].team_season.tolist()

# plotting a line to track team's change over seasons
ax.plot(x,y, color=text_color, lw=1.25, zorder=3)

# plotting team's most recent season
ax.scatter(x[-1],y[-1],s=120, color=primary, edgecolors=background, lw=.25, zorder=4)
t = ax.text(x[-1],y[-1]-2,t[-1],color=text_color,fontsize=14, ha="center")

# adding title and subtitle
fig.text(0.05,1,"{}'s goal difference from 2010/11 to 2019/20\n".format(team),
         fontsize=16, fontweight="bold", color=text_color)
fig.text(0.05,1,"English Premier League",fontweight="regular", fontsize=14,
         color=text_color)

# adding x and y labels
ax.set_xlabel("Goals For", fontweight="bold", fontsize=14, color=text_color)
ax.set_ylabel("Goals Against", fontweight="bold", fontsize= 14,color=text_color)

ax.set_ylim(90,18)
ax.set_xlim(10,120)

# remove pips
ax.tick_params(axis="both",length=0)

# remove top and right spines, color bottom and left
spines = ["top","right","bottom","left"]
for s in spines:
    if s in ["top","right"]:
        ax.spines[s].set_visible(False)
    else:
        ax.spines[s].set_color(text_color)



plt.tight_layout()
plt.show()
```



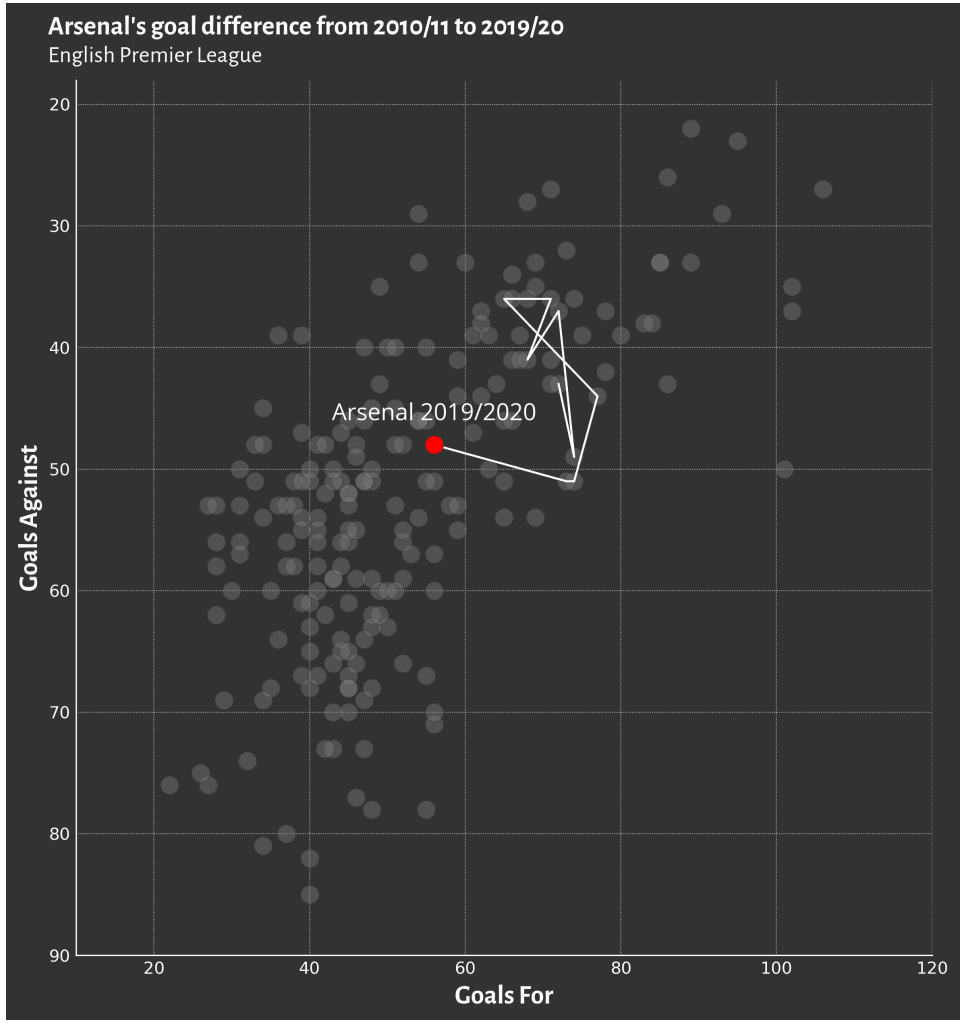
A real world example: fonts

```python
title_font = "Alegreya Sans"
body_font = "Open Sans"

# creating figure and axes
fig, ax = plt.subplots(figsize=(8,8))
fig.set_facecolor(background)
ax.patch.set_facecolor(background)

# adding a grid
ax.grid(ls="dotted",lw="0.5",color="lightgrey", zorder=1)

# getting our data to plot

x = df.goals_for.tolist()
y = df.goals_against.tolist()

ax.scatter(x,y,s=120,color=filler,edgecolors=background, alpha=0.6, lw=0.5, zorder=2)

# getting data for specified team
x = df[df.team == team].goals_for.tolist()
y = df[df.team == team].goals_against.tolist()
t = df[df.team == team].team_season.tolist()

# plotting a line to track team's change over seasons
ax.plot(x,y, color=text_color, lw=1.25, zorder=3)

# plotting team's most recent season
ax.scatter(x[-1],y[-1],s=120, color=primary, edgecolors=background, lw=.25, zorder=4)
t = ax.text(x[-1],y[-1]-2,t[-1],color=text_color,fontsize=14, ha="center", fontfamily=body_font)

# adding title and subtitle
fig.text(0.05,1,"{}'s goal difference from 2010/11 to 2019/20\n".format(team),
         fontsize=16, fontweight="bold", color=text_color, fontfamily=title_font)
fig.text(0.05,1,"English Premier League",fontweight="regular", fontsize=14,
         color=text_color, fontfamily=title_font)

# adding x and y labels
ax.set_xlabel("Goals For", fontweight="bold", fontsize=16, color=text_color, fontfamily=title_font)
ax.set_ylabel("Goals Against", fontweight="bold", fontsize= 16,color=text_color, fontfamily=title_font)
```



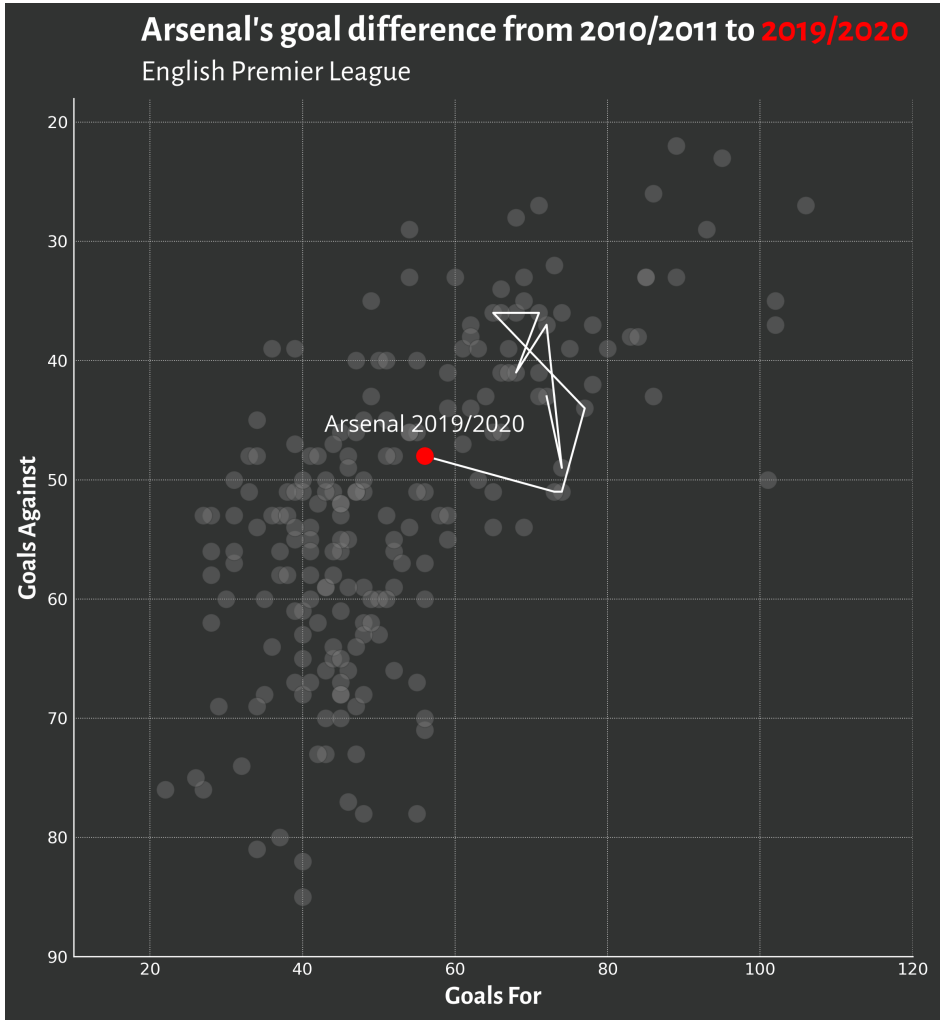
A real world example: toning down fillers

```python
# fonts
title_font = "Alegreya Sans"
body_font = "Open Sans"

# creating figure and axes
fig, ax = plt.subplots(figsize=(8,8))
fig.set_facecolor(background)
ax.patch.set_facecolor(background)

# adding a grid
ax.grid(ls="dotted",lw="0.5",color="lightgrey", zorder=1)

# getting our data to plot

x = df.goals_for.tolist()
y = df.goals_against.tolist()

ax.scatter(x,y,s=120,color=filler,edgecolors=background, alpha=0.4, lw=0.5, zorder=2)

# getting data for specified team
x = df[df.team == team].goals_for.tolist()
y = df[df.team == team].goals_against.tolist()
t = df[df.team == team].team_season.tolist()

# plotting a line to track team's change over seasons
ax.plot(x,y, color=text_color, lw=1.25, zorder=3)

# plotting team's most recent season
ax.scatter(x[-1],y[-1],s=120, color=primary, edgecolors=background, lw=.25, zorder=4)
t = ax.text(x[-1],y[-1]-2,t[-1],color=text_color,fontsize=14, ha="center", fontfamily=body_font)

# adding title and subtitle
fig.text(0.05,1,"{}'s goal difference from 2010/11 to 2019/20\n".format(team),
         fontsize=16, fontweight="bold", color=text_color, fontfamily=title_font)
fig.text(0.05,1,"English Premier League",fontweight="regular", fontsize=14,
         color=text_color, fontfamily=title_font)

# adding x and y labels
ax.set_xlabel("Goals For", fontweight="bold", fontsize=16, color=text_color, fontfamily=title_font)
ax.set_ylabel("Goals Against", fontweight="bold", fontsize= 16,color=text_color, fontfamily=title_font)

```



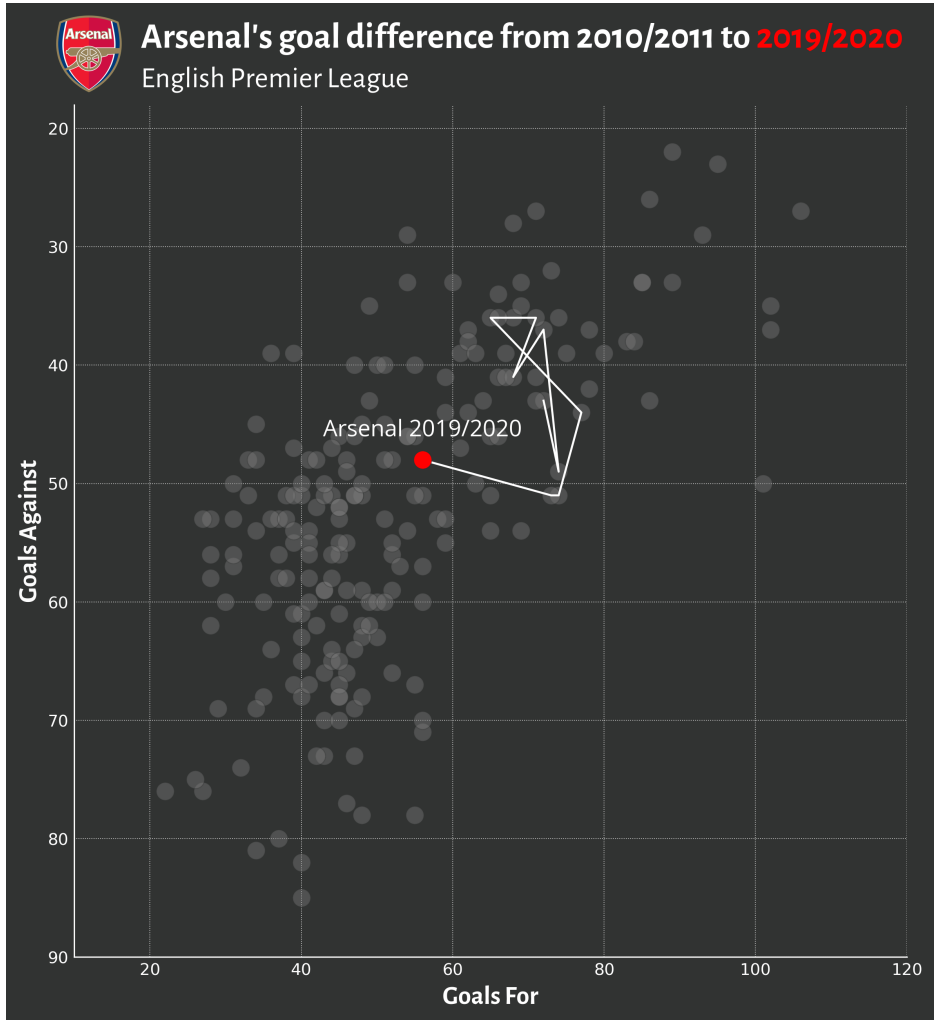
A real world example: adjust title

```python
fig.set_facecolor(background)
ax.patch.set_facecolor(background)

# adding a grid
ax.grid(ls="dotted",lw="0.5",color="lightgrey", zorder=1)

# getting our data to plot

x = df.goals_for.tolist()
y = df.goals_against.tolist()

ax.scatter(x,y,s=120,color=filler,edgecolors=background, alpha=0.4, lw=0.5, zorder=2)

# getting data for specified team
x = df[df.team == team].goals_for.tolist()
y = df[df.team == team].goals_against.tolist()
t = df[df.team == team].team_season.tolist()

# plotting a line to track team's change over seasons
ax.plot(x,y, color=text_color, lw=1.25, zorder=3)

# plotting team's most recent season
ax.scatter(x[-1],y[-1],s=120, color=primary, edgecolors=background, lw=.25, zorder=4)
t = ax.text(x[-1],y[-1]-2,t[-1],color=text_color,fontsize=14, ha="center", fontfamily=body_font)


# adding title and subtitle
ssn_start = df[df.team == team].season_id.iloc[0]
ssn_end = df[df.team == team].season_id.iloc[-1]

ssn_start = str(ssn_start)+"/"+str(ssn_start+1)
ssn_end = str(ssn_end)+"/"+str(ssn_end+1)

s = "{}'s goal difference from {} to <{}>\n"
htext.fig_htext(s.format(team,ssn_start,ssn_end),0.15,0.99,highlight_colors=[primary],
                highlight_weights=["bold"],string_weight="bold",fontsize=22,
                fontfamily=title_font,color=text_color)

fig.text(0.15,1,"English Premier League",fontweight="regular",
         fontsize=18,fontfamily=title_font, color=text_color)
```



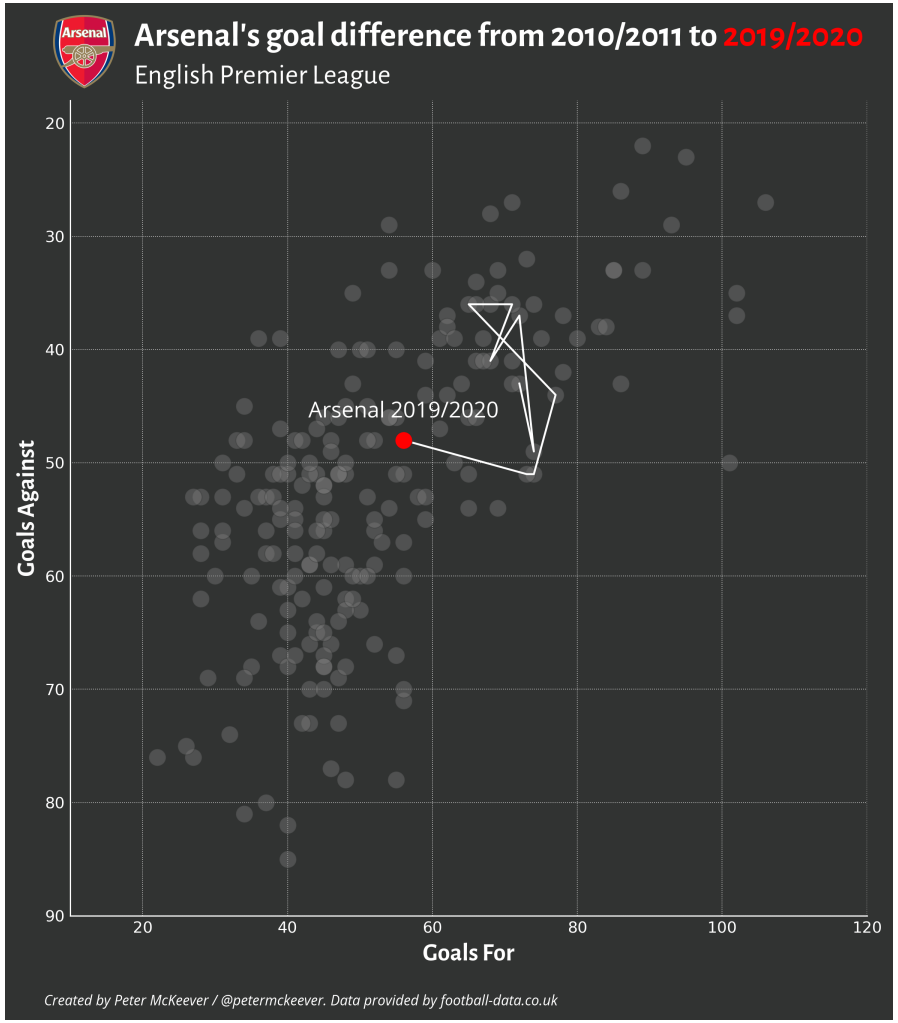
A real world example: logo

```python
ssn_start = str(ssn_start)+"/"+str(ssn_start+1)
ssn_end = str(ssn_end)+"/"+str(ssn_end+1)

s = "{}'s goal difference from {} to <{}>\n"
htext.fig_htext(s.format(team,ssn_start,ssn_end),0.15,0.99,highlight_colors=[primary],
                highlight_weights=["bold"],string_weight="bold",fontsize=22,
                fontfamily=title_font,color=text_color)

fig.text(0.15,1,"English Premier League",fontweight="regular",
         fontsize=18,fontfamily=title_font, color=text_color)

# adding x and y labels
ax.set_xlabel("Goals For", fontweight="bold", fontsize=16, color=text_color, fontfamily=title_font)
ax.set_ylabel("Goals Against", fontweight="bold", fontsize= 16,color=text_color, fontfamily=title_font)

ax.set_ylim(90,18)
ax.set_xlim(10,120)

# remove pips
ax.tick_params(axis="both",length=0)

# remove top and right spines, color bottom and left
spines = ["top","right","bottom","left"]
for s in spines:
    if s in ["top","right"]:
        ax.spines[s].set_visible(False)
    else:
        ax.spines[s].set_color(text_color)

# adding logo
ax2 = fig.add_axes([0.02,0.96,0.15,0.15])
ax2.axis("off")
url = "https://logos-world.net/wp-content/uploads/2020/05/Arsenal-Logo.png"
response = requests.get(url)
img = Image.open(BytesIO(response.content))
ax2.imshow(img)



plt.tight_layout()
```



A real world example: adding credits

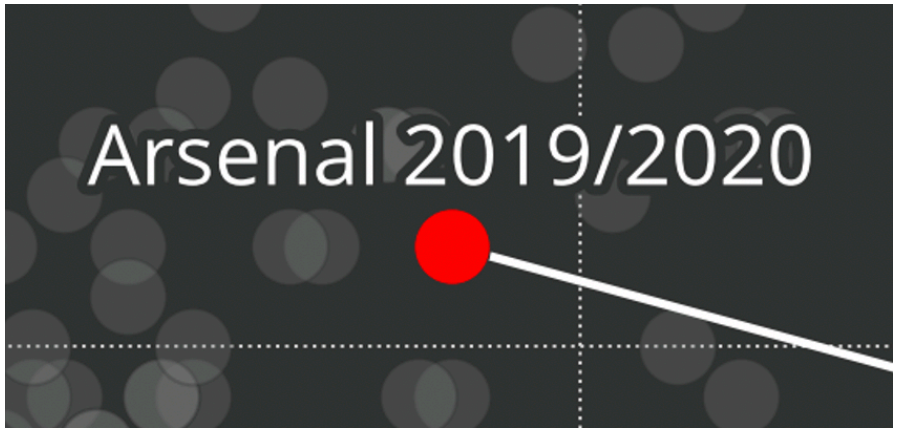
```python
s = "{}'s goal difference from {} to <{}>\n"
htext.fig_htext(s.format(team,ssn_start,ssn_end),0.15,0.99,highlight_colors=[primary],
                highlight_weights=["bold"],string_weight="bold",fontsize=22,
                fontfamily=title_font,color=text_color)

fig.text(0.15,1,"English Premier League",fontweight="regular",
         fontsize=18,fontfamily=title_font, color=text_color)

# adding x and y labels
ax.set_xlabel("Goals For", fontweight="bold", fontsize=16, color=text_color, fontfamily=title_font)
ax.set_ylabel("Goals Against", fontweight="bold", fontsize= 16,color=text_color, fontfamily=title_font)

ax.set_ylim(90,18)
ax.set_xlim(10,120)

# remove pips
ax.tick_params(axis="both",length=0)

# remove top and right spines, color bottom and left
spines = ["top","right","bottom","left"]
for s in spines:
    if s in ["top","right"]:
        ax.spines[s].set_visible(False)
    else:
        ax.spines[s].set_color(text_color)

# adding logo
ax2 = fig.add_axes([0.02,0.96,0.15,0.15])
ax2.axis("off")
url = "https://logos-world.net/wp-content/uploads/2020/05/Arsenal-Logo.png"
response = requests.get(url)
img = Image.open(BytesIO(response.content))
ax2.imshow(img)



fig.text(0.05, -0.025, "Created by Peter McKeever / @petermckeever. Data provided by football-data.co.uk",
         fontstyle="italic",fontsize=9, fontfamily=body_font, color=text_color)
```



A real world example: adding path effects

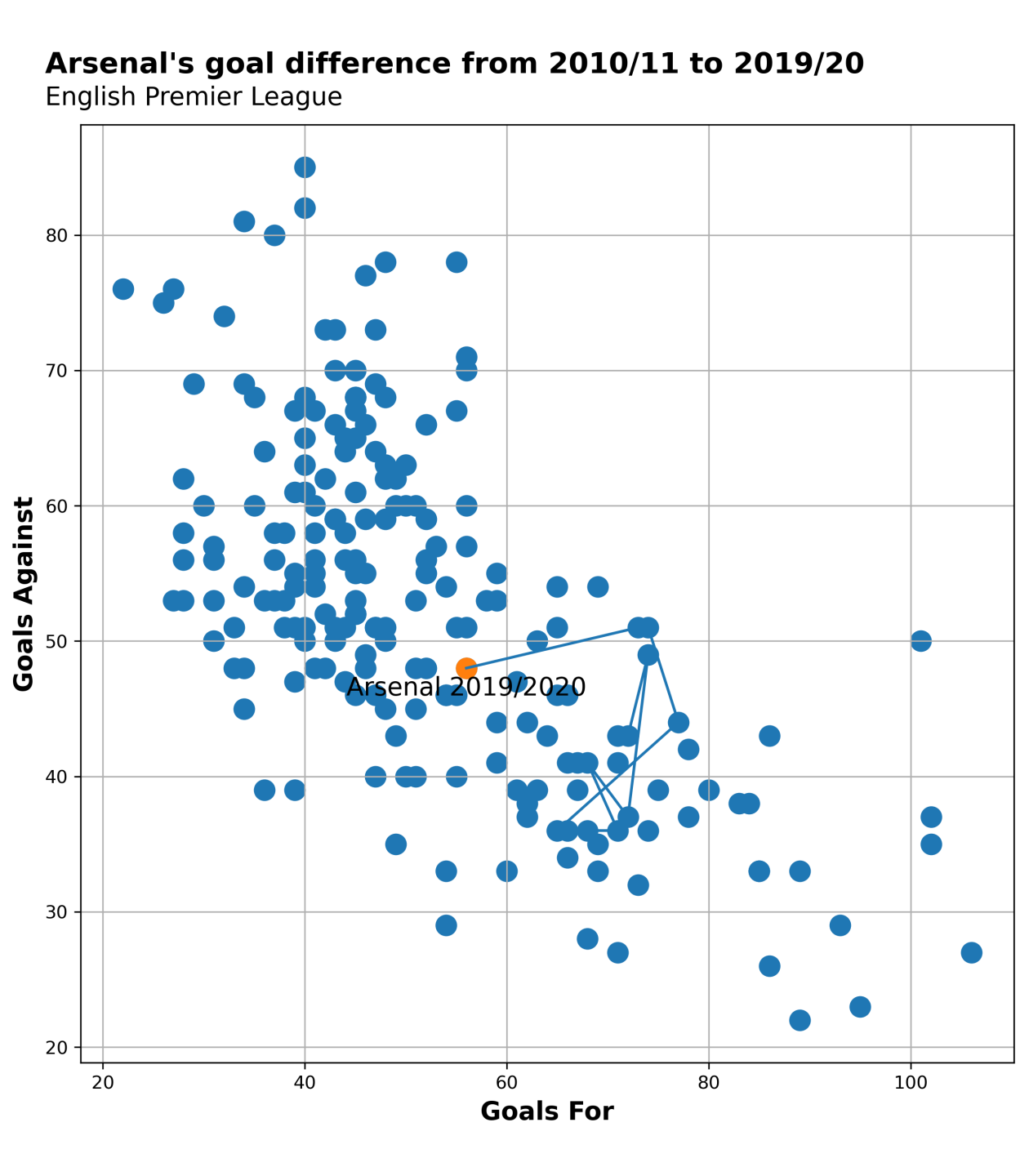
Note that we created our text label as a variable : t

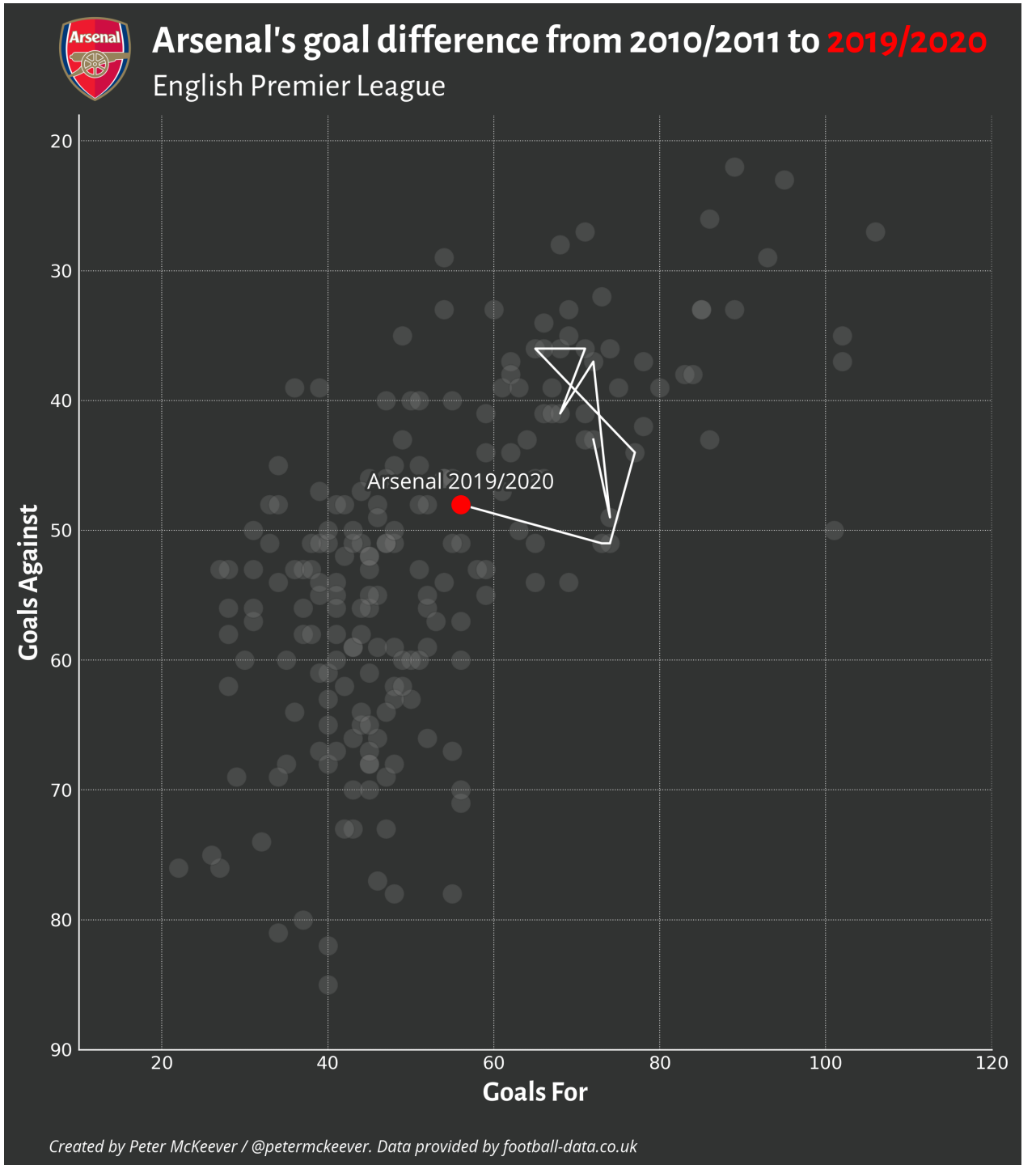
```
t = ax.text(x[-1], y[-1]-2, t[-1],
            color=text_color,
            fontsize=14, ha="center",
            fontfamily=body_font)
```



By setting that as a variable, we now have a Text object that we can modify. By using path_effects we can draw a border around the text to make it stand out from the background:

```
t.set_path_effects([path_effects.withStroke(linewidth=3,
                    foreground=background)])
```


Arsenal's goal difference from 2010/11 to 2019/20
English Premier League
Arsenal 2019/2020
Goals Against
Goals For
20
30
40
50
60
70
80
20
40
60
80
100


Arsenal's goal difference from 2010/2011 to 2019/2020
English Premier League
Arsenal 2019/2020
Goals Against
Goals For
20
30
40
50
60
70
80
90
20
40
60
80
100
120
Created by Peter McKeever / @petermckeever. Data provided by football-data.co.uk

An alternative: rcParams

Most of what we have covered has required us adding the same variables to various parts of our code:

```
background = "#313332"
text_color = "w"
title_font = "Alegreya Sans"
```

If we will reuse these elements, we can change the rcParams to use those as our defaults. This saves us time and makes our output more consistent.

```
mpl.rcParams['figure.facecolor'] = "#313332"
mpl.rcParams['axes.facecolor'] = "#313332"
mpl.rcParams['font.family'] = "Alegrey Sans"
```

An alternative: rcParams

Note that when we change rcParams, we change the default matplotlib until either the package is reimported or reloaded, or until we reset our rcParams.

A simple way of doing this is to create a function to handle the reset:

```
def reset_matplotlib():
    mpl.rcParams.update(mpl.rcParamsDefault)
```

By creating this, we can simply reset rcParams to their default settings by running

```
reset_matplotlib()
```

Matplotlib: Beyond the Default

Thank you