# Enabling Technologies for Data Science

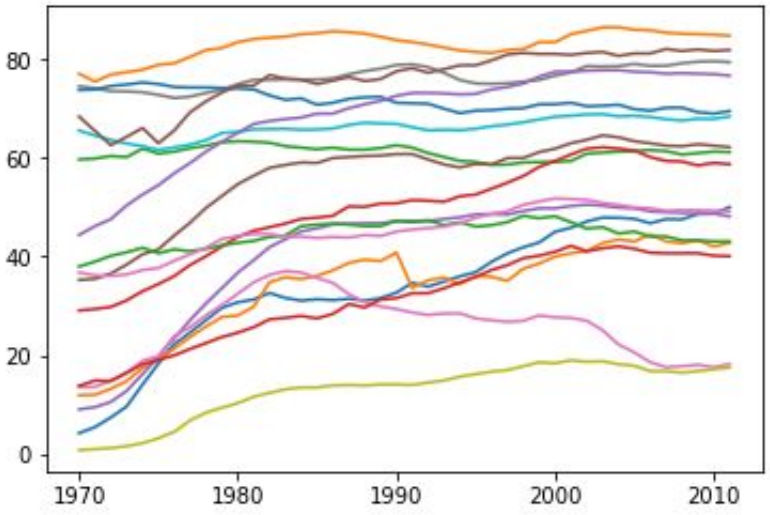
## Lesson 7 – Plotting in Python

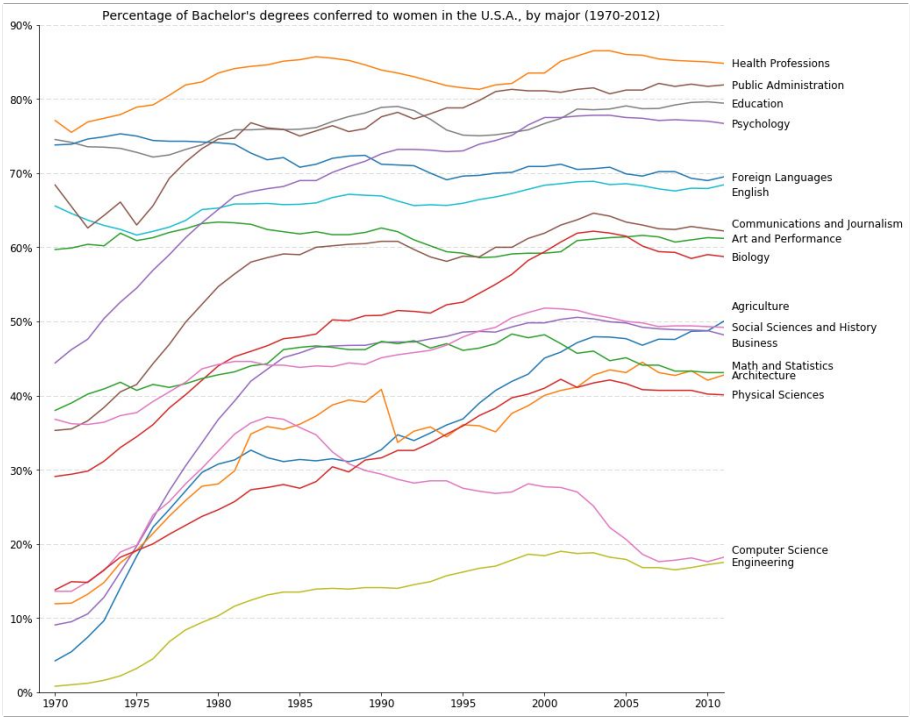

TINKERCADEMY

An Infocomm Club Appointed Vendor

# Plotting with Matplotlib

We’re going to learn how to take the default matplotlib output and customise it:



# Plotting with Matplotlib

As usual, import the libraries that we need:

```
import matplotlib.pyplot as plt
import pandas as pd
```

download the data set from:
http://www.randalolson.com/wp-content/uploads/percent-bachelors-degrees-women-usa.csv

```
# Read the data into a pandas DataFrame.
df = pd.read_csv("filename.csv")
```

or alternatively just read the data straight from the web:

```
gender_degree_data = pd.read_csv("http://www.randalolson.com/wp-content/uploads/percent-bachelors-degrees-women-usa.csv")
```

# Plotting with Matplotlib

Print out the first 5 rows with df.head(). Note that all the different majors are located in df.columns.

What is the current row index? How can we change this?

Take a look at the documentation for the read_csv method and see if you can figure out how to update the row index:

https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

# Plotting with Matplotlib

Let's first plot the data to see how it looks. We're going to use some new methods:

```
plt.plot(x, y, lw=1.5)
# lw sets the line width in points
# x,y are the x and y coords of your data points

plt.savefig("filename.png")
# this will overwrite files without prompting!
```

# Plotting with Matplotlib

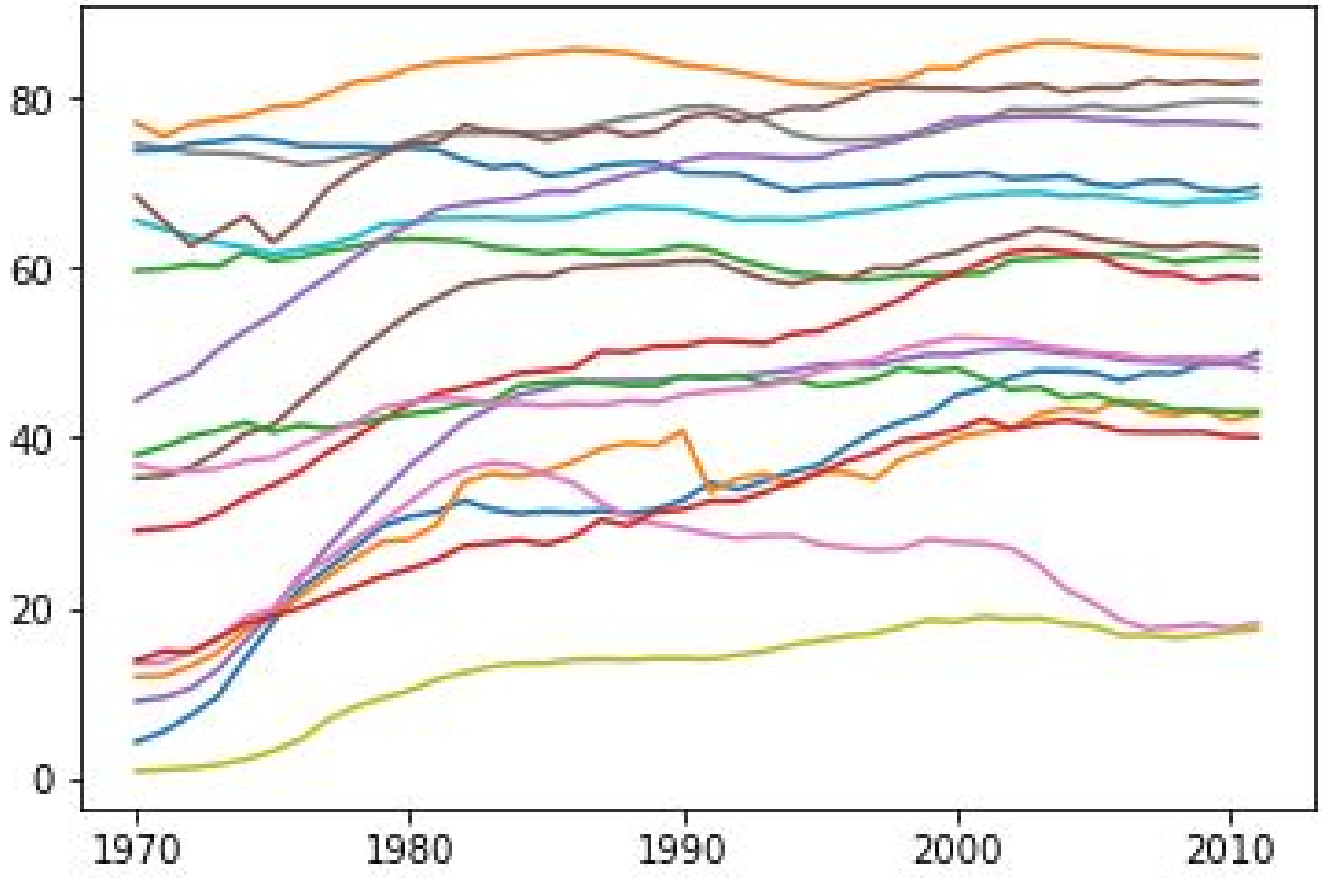
Let's plot the data to see how it looks:

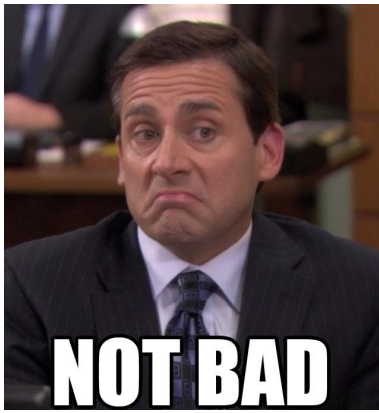
```
x = df.index.values  #the x axis values never change
for major in df.columns:
    y = df[major].values #the y axis values do change
    plt.plot(x, y, lw=1.5) #lw sets the line width in points

plt.savefig("percent-bachelors-degrees-women-usa1.png",
bbox_inches="tight")
#Note that this will overwrite files without prompting!
#bbox_inches="tight" removes all the extra whitespace on the
edges of your plot.
```

# Plotting with Matplotlib

So far, we have this:



# Improvements

But we can definitely improve it:

**What problems does the plot have currently?**

# Improvements

The plot is currently missing:

- line labels
- axis labels
- a title

But we can customise the plot to add these.

# Improvements – Line Labels

To add line labels, we're going to use plt.text:

```
plt.text(x_pos, y_pos, "text", fontsize=12)
# there are also many other keyword arguments,
#  such as fontsize or color.
```

Try using this to add line labels onto your plot. You'll need to figure out what you want to plot and where to plot it. Let's aim to plot our line labels on the righthand side of the plot next to the end of each line of data.

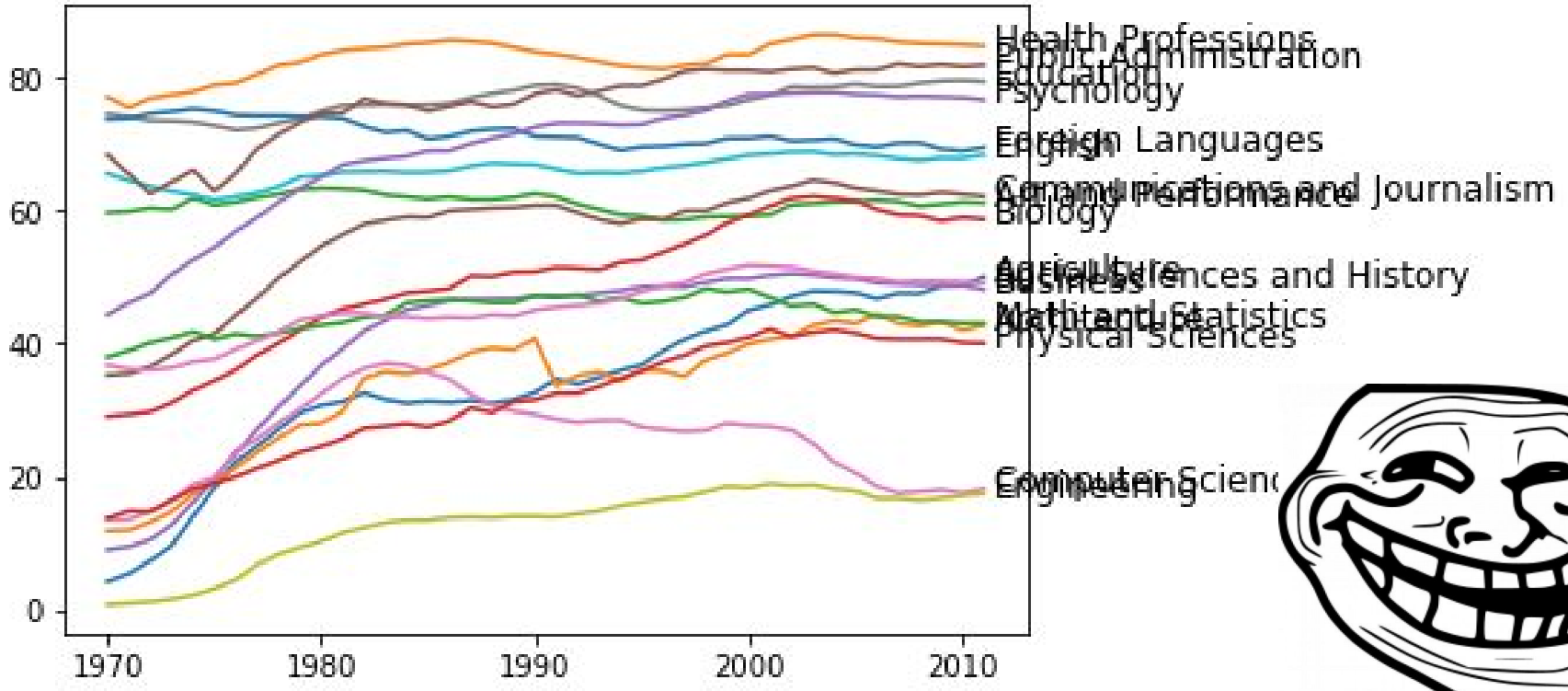
# Improvements – Line Labels

To add text to your line labels, add a plt.text inside your for loop:

```
y_pos = df[major].values[-1] - 0.5
plt.text(2011.5, y_pos, major, fontsize=12)
# we set  y_pos equal to the last value of each line
# since we want our labels to be right next to the ending point of each line
```

# Improvements – Line Labels

Looks perfect.



# Improvements – Line Labels

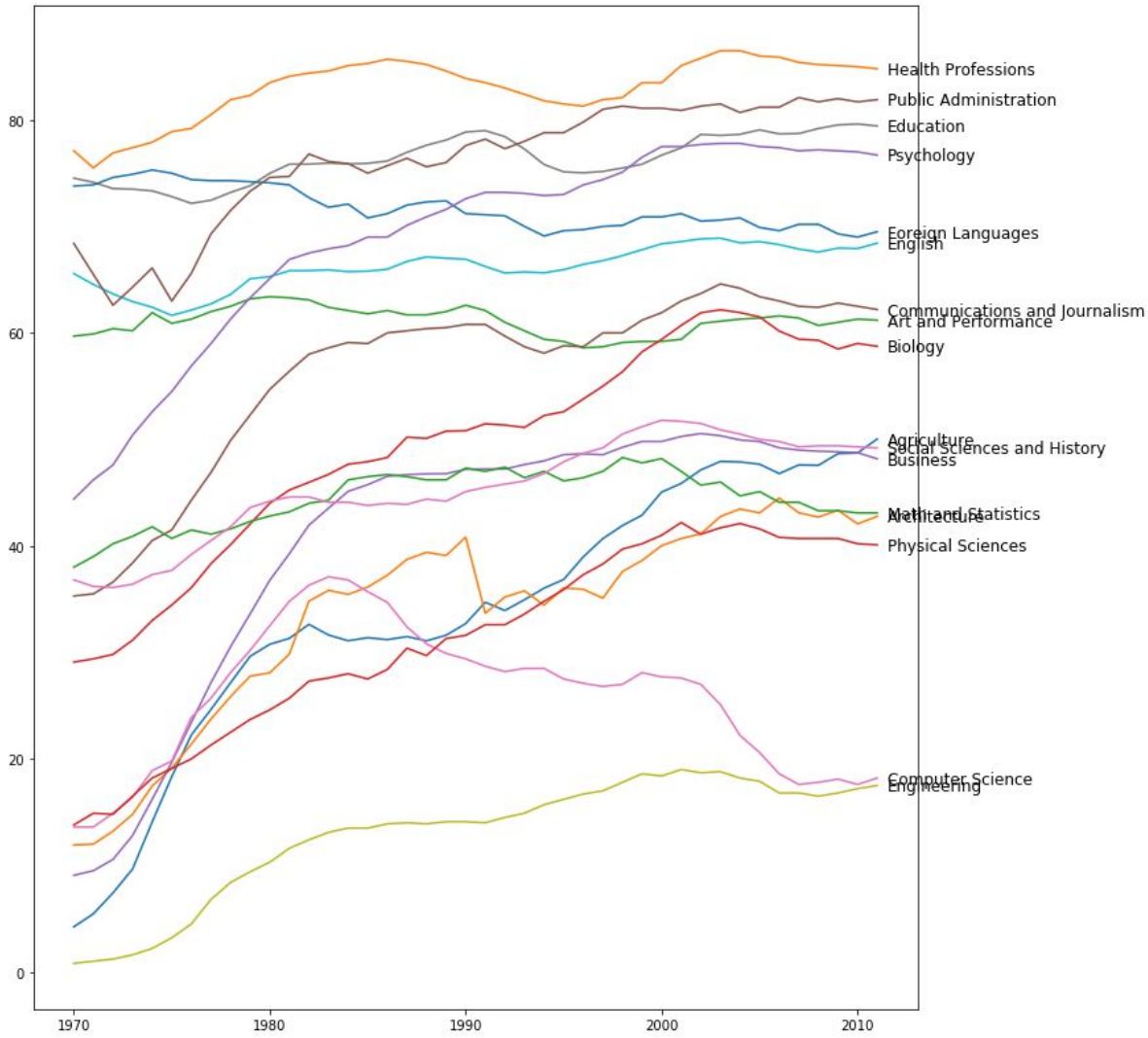
Let's make the graph a bit taller so that the line labels aren't overlapping as much:

```
plt.figure(figsize=(14, 14))
```

Make sure you add this line *__before__* you print your plot!

Feel free to try out different sizes!

# Improvements – Line Labels

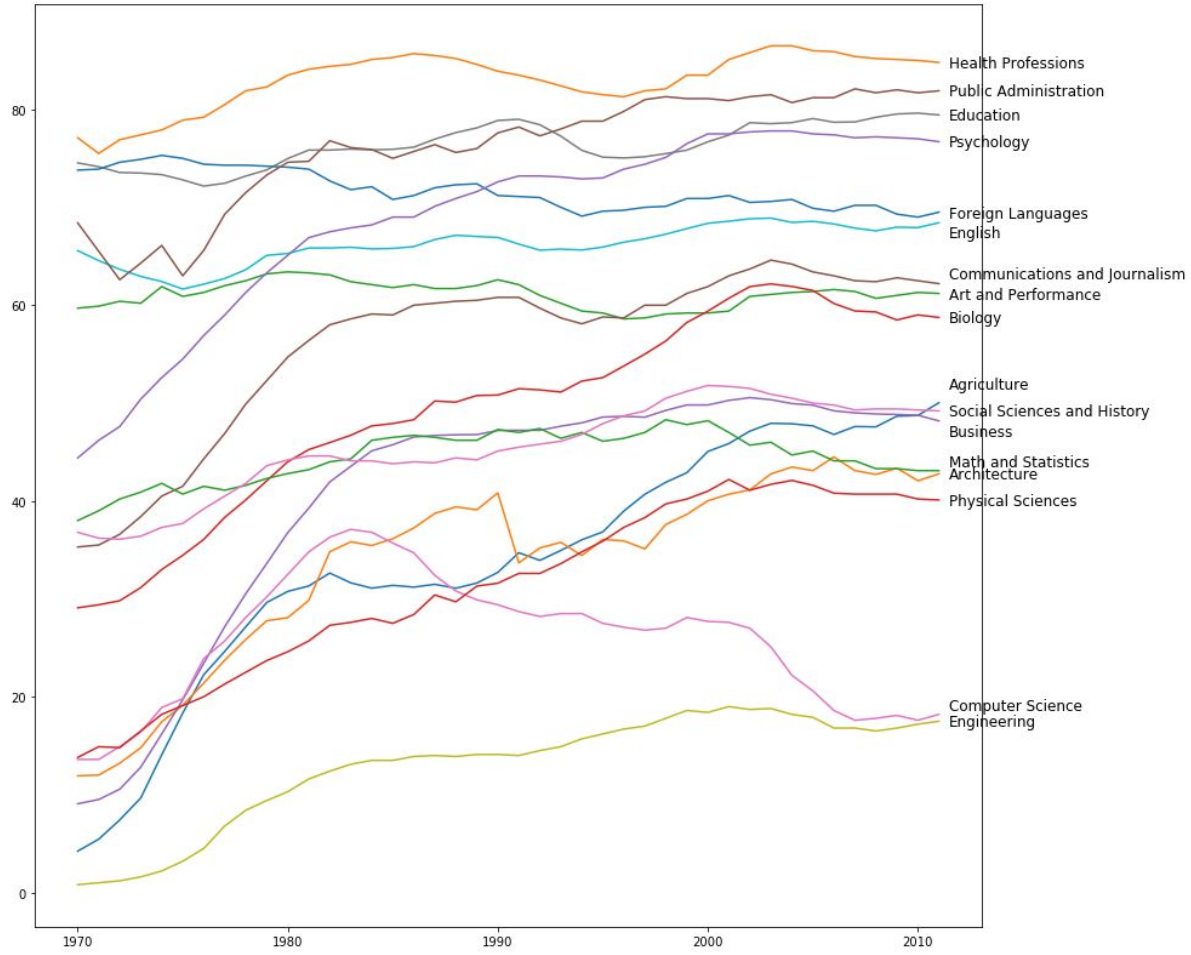


# Improvements – Line Labels

That's much better but there's still some overlap. Sadly we have to resort to manual adjustments :(

```
if major == "English":
    y_pos -= 1
elif major == "Communications and Journalism":
    y_pos += 1
elif major == "Agriculture":
    y_pos += 2
elif major == "Business":
    y_pos -= 1
elif major == "Math and Statistics":
    y_pos += 1
elif major == "Computer Science":
    y_pos += 1
```

# Improvements – Line Labels




Better, but still have a frame line in the way

# Improvements - Remove Frame Lines

Finally, let's remove the frame lines around the plot that are interfering with our line labels. To edit the frame lines (called spines in matplotlib), we first create a subplot, which will allow us to edit the spines:

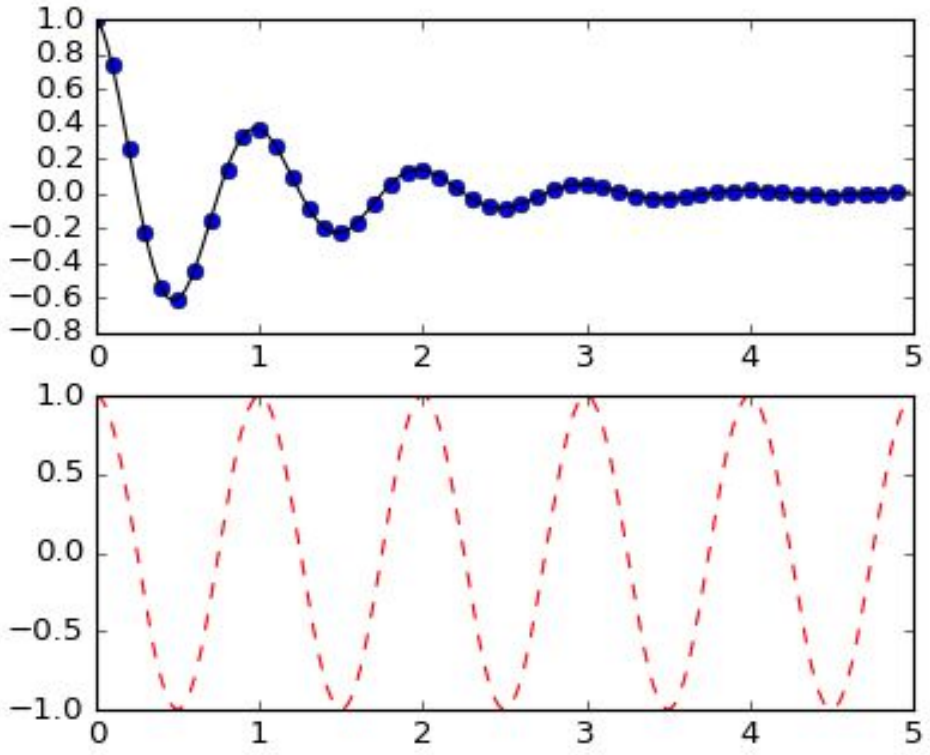
```
ax = plt.subplot(1,1,1)
```

subplot() arguments are:

numrows, numcols, figurenum

# Sidenote on subplots

For example:

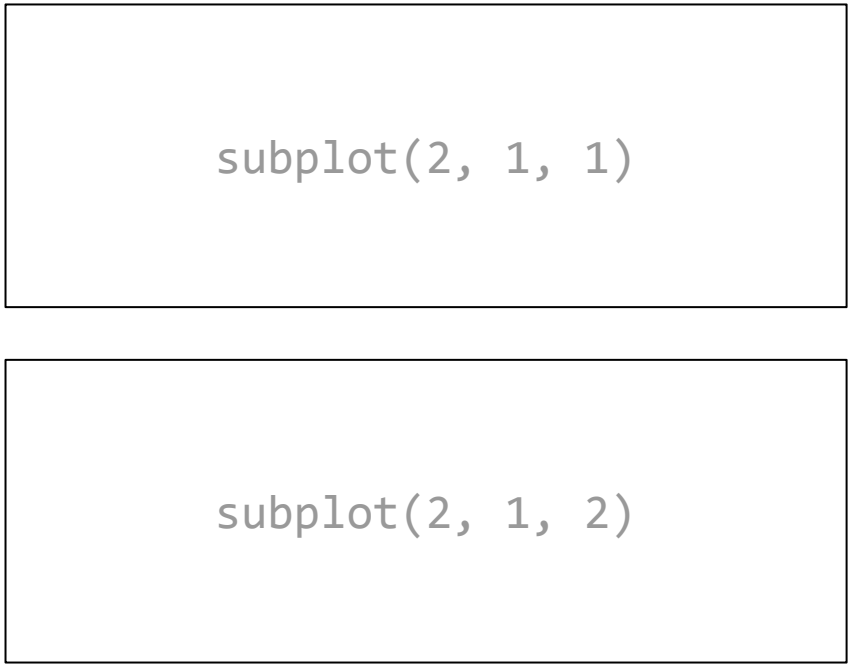



`numrows = 2, numcols = 1, figure = 1 or 2`

`So we use: subplot(2,1,1) or subplot(2,1,2)`

# Sidenote on subplots

For example:

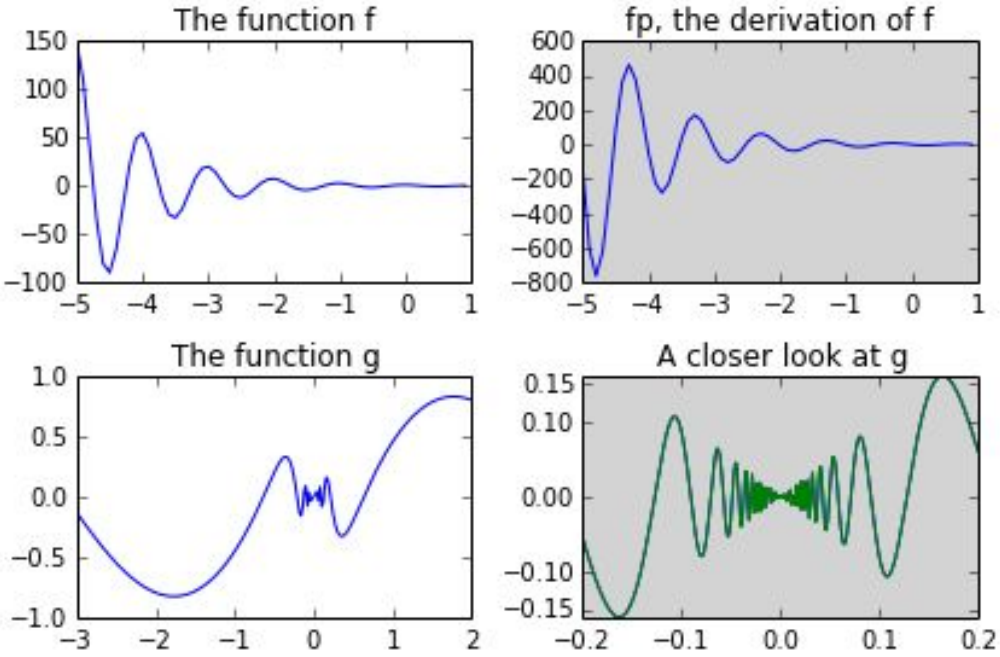



`numrows = 2, numcols = 1, figure = 1 or 2`

`So we use: subplot(2,1,1) or subplot(2,1,2)`

# Sidenote on subplots

For example:




`numrows = 2, numcols = 2, figure = 1, 2, 3, or 4`

`So we use: subplot(2,2,1) or subplot(2,2,2)`

`or subplot(2,2,3) or subplot(2,2,4)`

# Sidenote on subplots

For example:

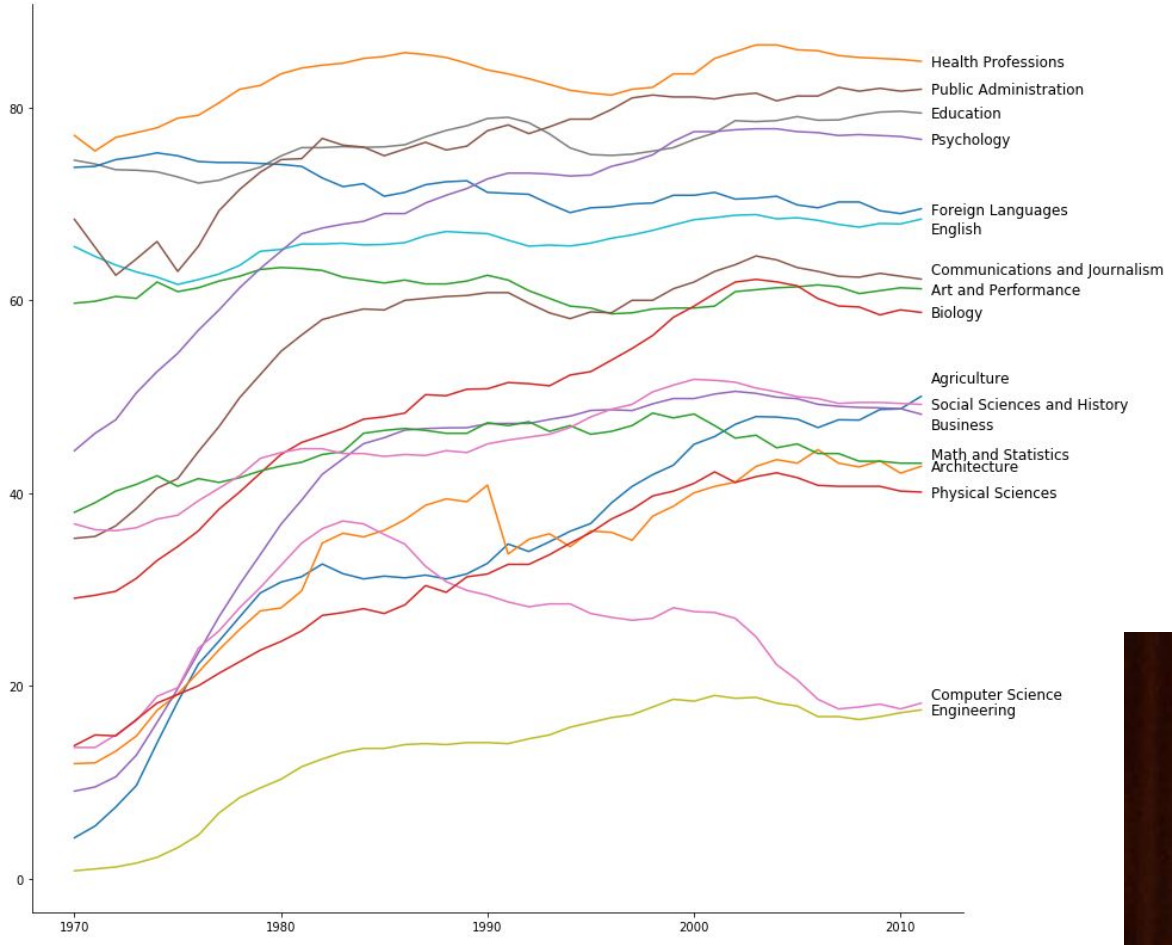

```
numrows = 2, numcols = 2, figure = 1, 2, 3, or 4
So we use: subplot(2,2,1) or subplot(2,2,2)
or subplot(2,2,3) or subplot(2,2,4)
```

# Improvements - Remove Frame Lines

Continuing with our line removal: we first create a subplot, then edit the spines:

```
ax = plt.subplot(1,1,1)
ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
#ax.spines["bottom"].set_visible(False)
#ax.spines["left"].set_visible(False)
```

# Improvements – Line Labels





Much better!

# Improvements – Frame Lines

There are a number of other changes we can make to the frame lines if needed:

- add/remove tick marks
- change tick labels
- change tick frequency
- change x and y axis limits

The next several slides give examples of these and show the results.

# Improvements – Tick Marks

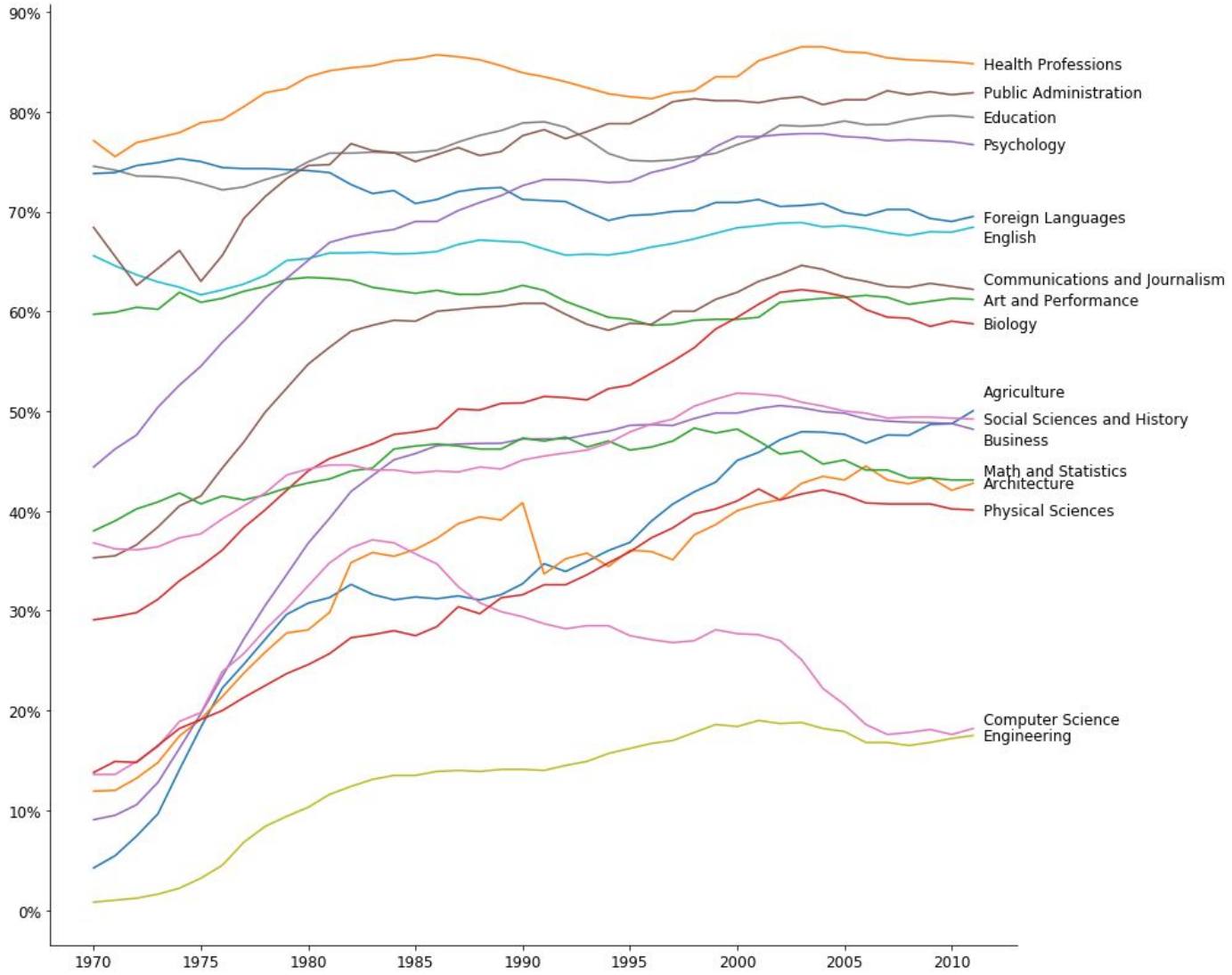
Changing the frequency of tick marks, and the fontsize of the tick labels:

```
plt.yticks(range(0, 91, 10), fontsize=12)
plt.xticks(range(df.index[0],df.index[-1]+1, 5), fontsize=12)
```

Optional: if we want to get fancy, we can add in % signs to the y-axis tick mark labels:

```
plt.yticks(range(0, 91, 10), [str(x) + "%" for x in range(0, 91, 10)], fontsize=12)
plt.xticks(range(df.index[0],df.index[-1]+1, 5), fontsize=12)
```

# Improvements - Tick Marks



# Improvements - Tick Marks

Changing the x and y axis limits (not really needed with this example data, but just to show how):

```
plt.ylim(0, 90)
plt.xlim(1969, 2011)
```

- Why start at 1969 when our data starts at 1970? We want a little bit of whitespace to the start of the x-axis to prevent the lines from crowding the y-axis (try out both ways to see the difference!)

# Improvements – Plot Tick Lines

Let's get fancy: because we have such a tall plot in this example, let's provide very light horizontal tick lines to help gauge the line values.

To do this, let's plot custom horizontal lines every 10%. Remember the arguments of plt.plot are:

```
plt.plot([array of x values],
         [array of y values],
         "line style string",
         plus keyword args like lw)
```

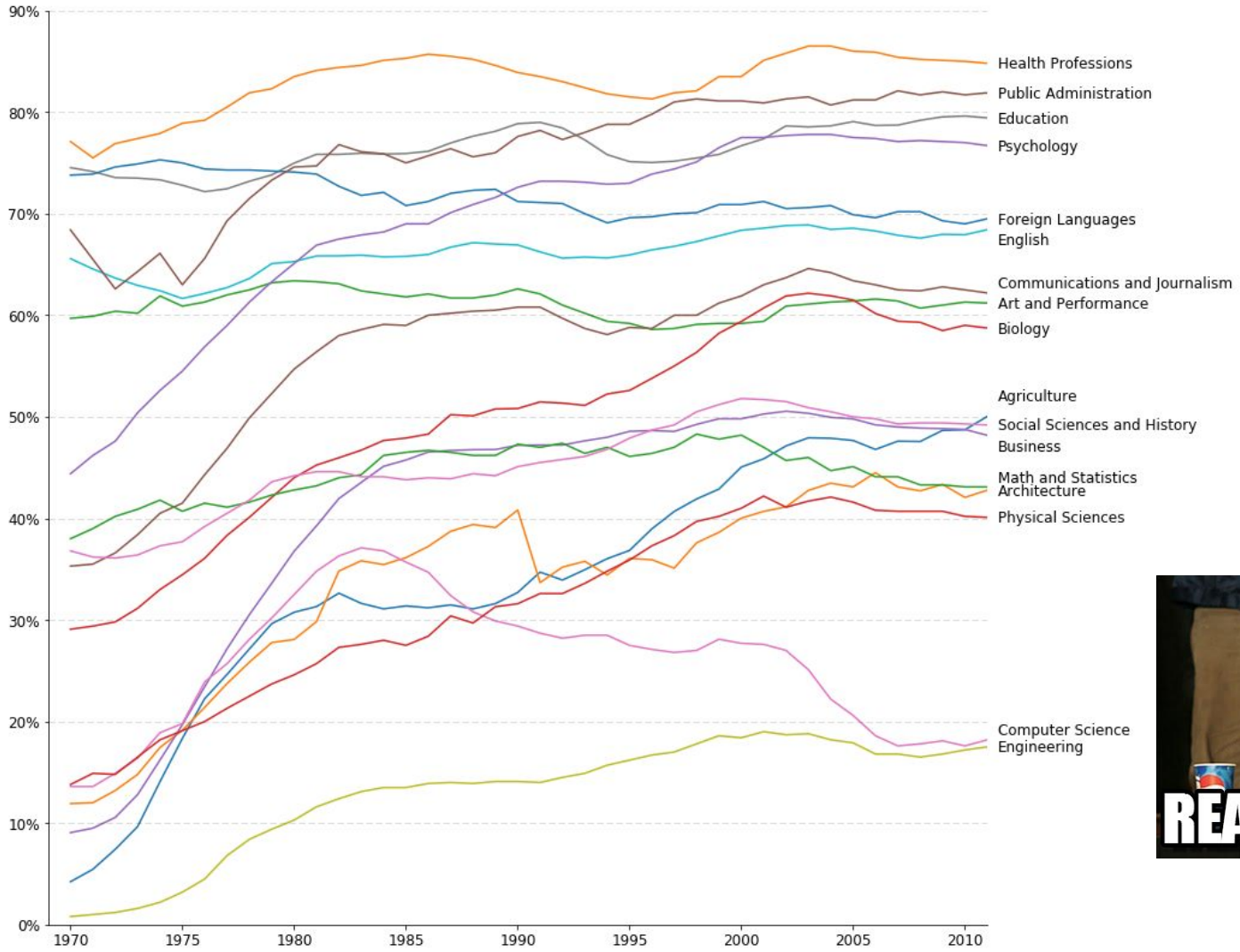
**Line style string specifies how to draw the line (dotted, dashed, etc.)**

# Improvements – Plot Tick Lines

Let's plot custom horizontal lines every 10%:

```
for y in range(10, 91, 10):
    plt.plot(range(1969, 2012),
        [y] * len(range(1969, 2012)),
        "--",
        lw=0.5,
        color="black",
        alpha=0.01)
```

# Improvements – Plot Tick Lines

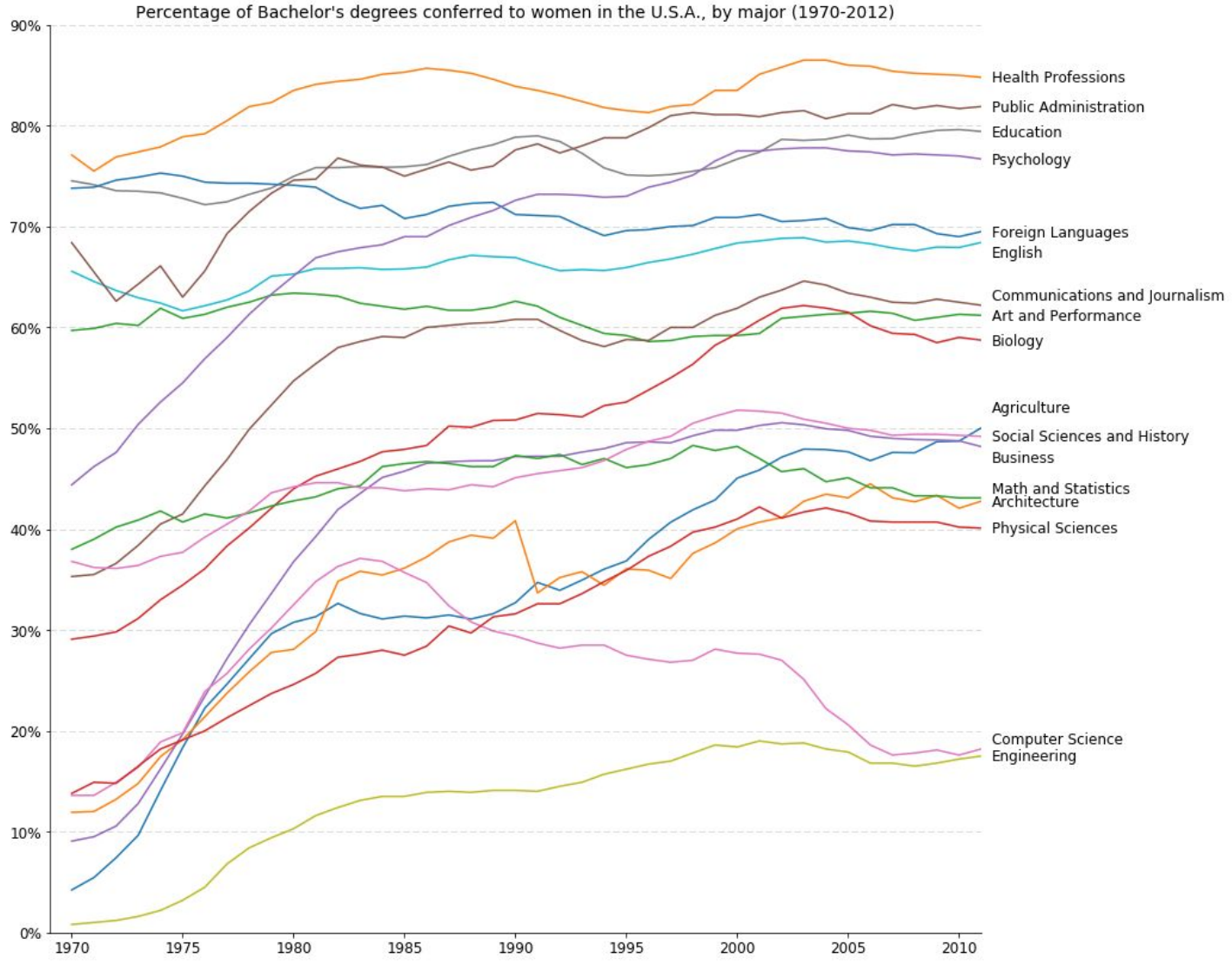



# Improvements – Plot Title

One more thing: let's add a title:

```
plt.title("Percentage of Bachelor's degrees
    conferred to women in the U.S.A., by major (1970-2012)",
    fontsize=14)
```

# Improvements - Plot Title





# Customising the stylesheet

We can customise the style of matplotlib using preset stylesheets:

```
plt.style.use('stylenamehere')
```

To see the list of available styles, just run:

```
print(plt.style.available)
```

For example:

```
plt.style.use('seaborn')
```

```
plt.style.use('fivethirtyeight')
```

# Plotting Exercises

**Try it out:**

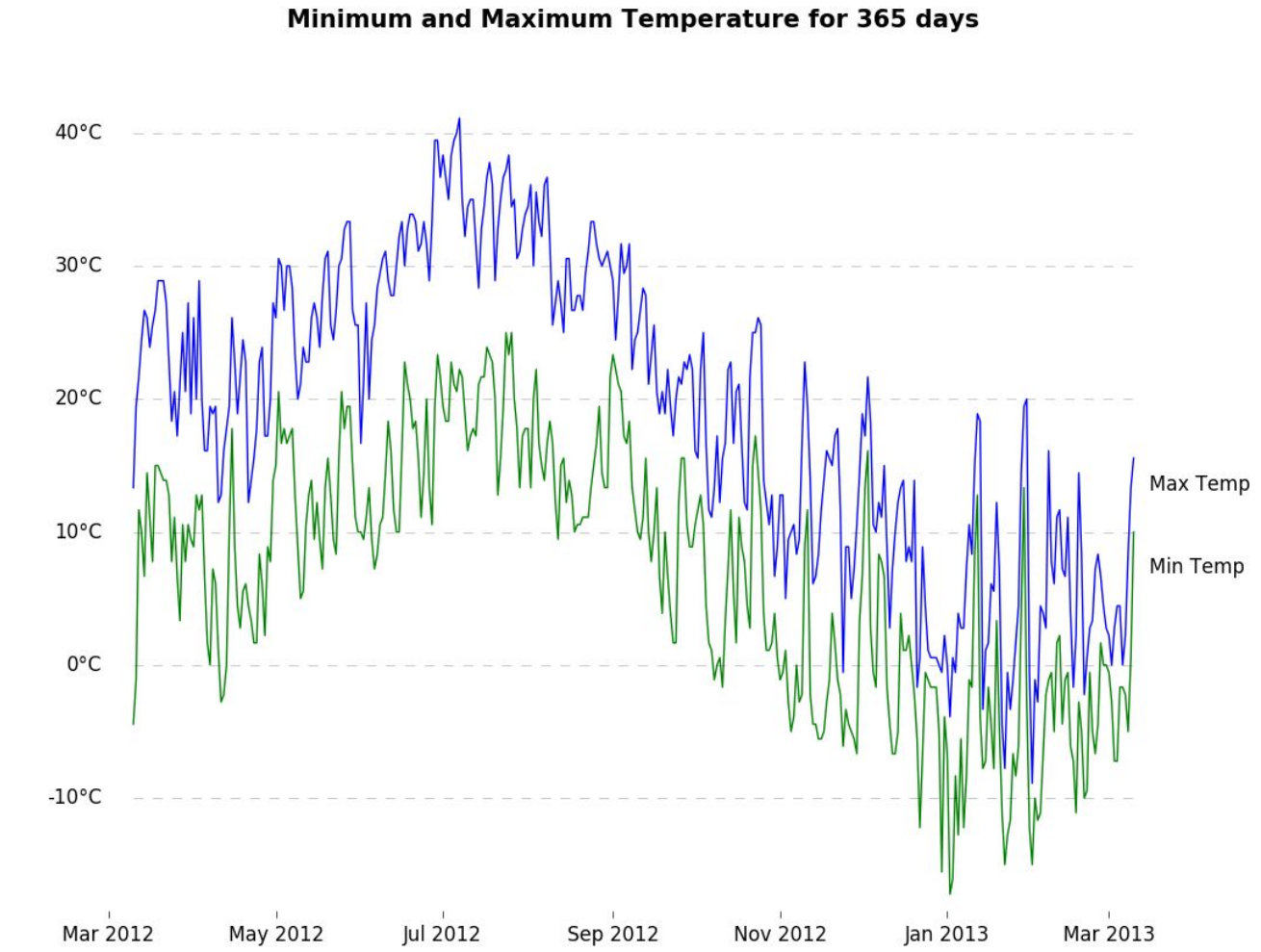
Using the data from weather_year.csv, plot lines that show the min and max temperature over the course of the data.

X-axis will be Time (in days)

Y-axis will be Temperature

Make sure you customise the plot to make it look nicer than the default settings.

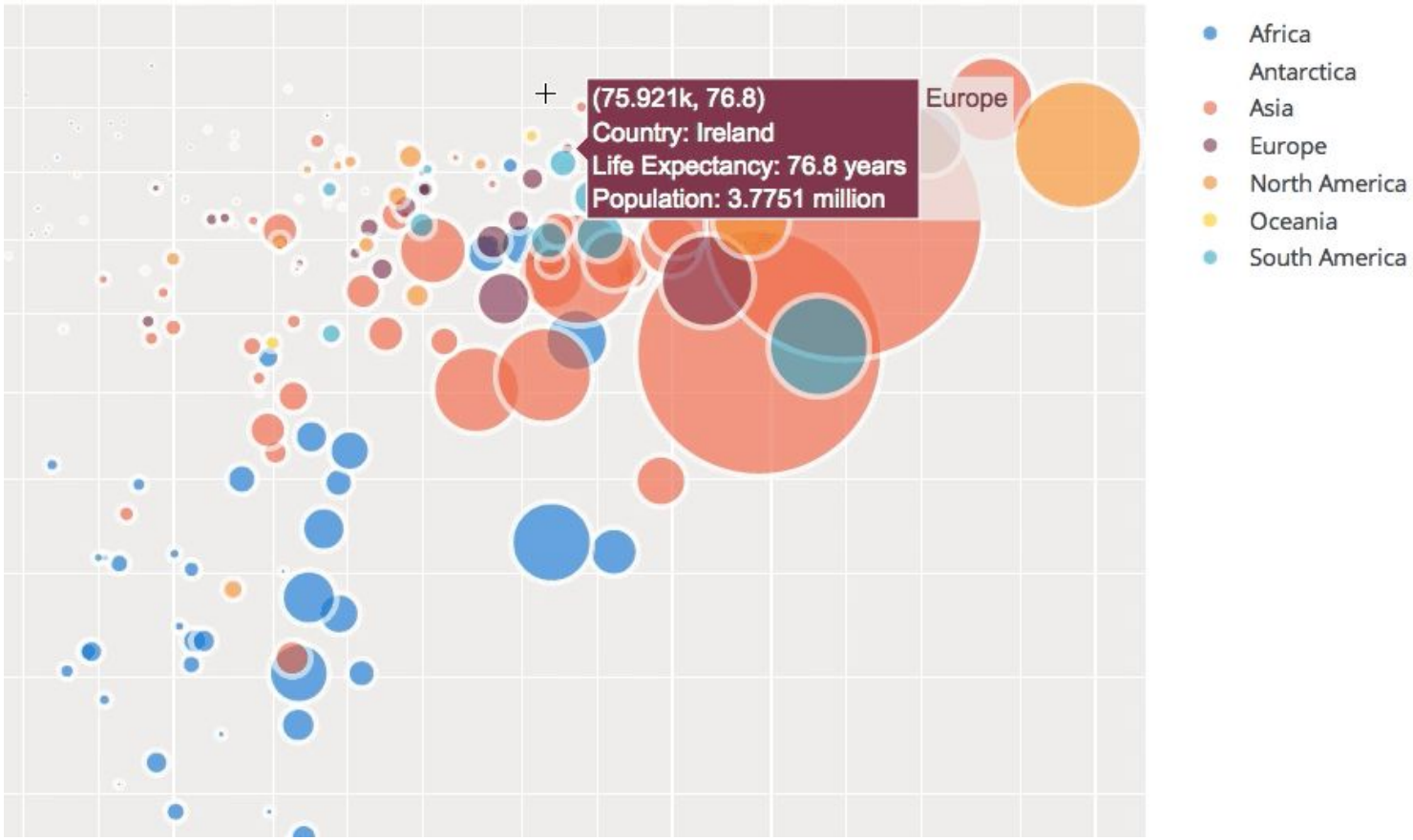
# Example Output




Data source:
Author: Mike Gonsalves
Note: Mention if any data is missing, estimated, etc.

# Plotting with plot.ly

Life expectancy vs GNP from MySQL world database (bubble chart)



# Plotting with plot.ly

Plot.ly is a data visualization toolbox that's compatible with IDLE or Jupyter notebooks.

It has an offline mode that allows you to save files locally as html files (if using IDLE) or inside ipython notebooks (if using Jupyter).

Every plotly graph is a JSON object.

# Plotting with plot.ly

Offline use in IDLE:

```
import plotly
print(plotly.__version__) # version >1.9.4 required for offline plots
plotly.offline.plot({
"data": [plotly.graph_objs.Scatter(x=[1, 2, 3, 4], y=[4, 1, 3, 7])],
"layout": plotly.graph_objs.Layout(title="hello world")
})
```

**This is how we plot offline using IDLE (meaning we don't upload the plot to our plotly account)**

# Plotting with plot.ly

Offline use in Jupyter Notebook:

```
import plotly
print(plotly.__version__) # version 1.9.4 required for offline plots
plotly.offline.init_notebook_mode() # run at the start of every notebook
plotly.offline.iplot({
"data": [{"x": [1, 2, 3],"y": [4, 2, 5]}],
"layout": {"title": "hello world"}
})
```

**This is how we plot offline in Jupyter notebook**

# Plotting with plot.ly

plotly charts are described *declaratively* with objects in plotly.graph_objs and dict

**Every aspect of a plotly chart** (the colors, the grids, the data, and so on) **has a corresponding key-value attribute** in these objects.

# plot.ly examples

**See the .ipynb file sent via Slack for an introduction to plot.ly**