Plotting

Content:

- Plotting line graphs
- Label Type and Graph Thickness
- x and y axis layout
- plt.scatter() and discrete data points (not calculated, whole numbers)
- plt.savefig() for saving to file
- Bar plots using demographic
- Data scale on axis

https://matplotlib.org/3.1.0/tutorials/introductory/pyplot.html

Matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

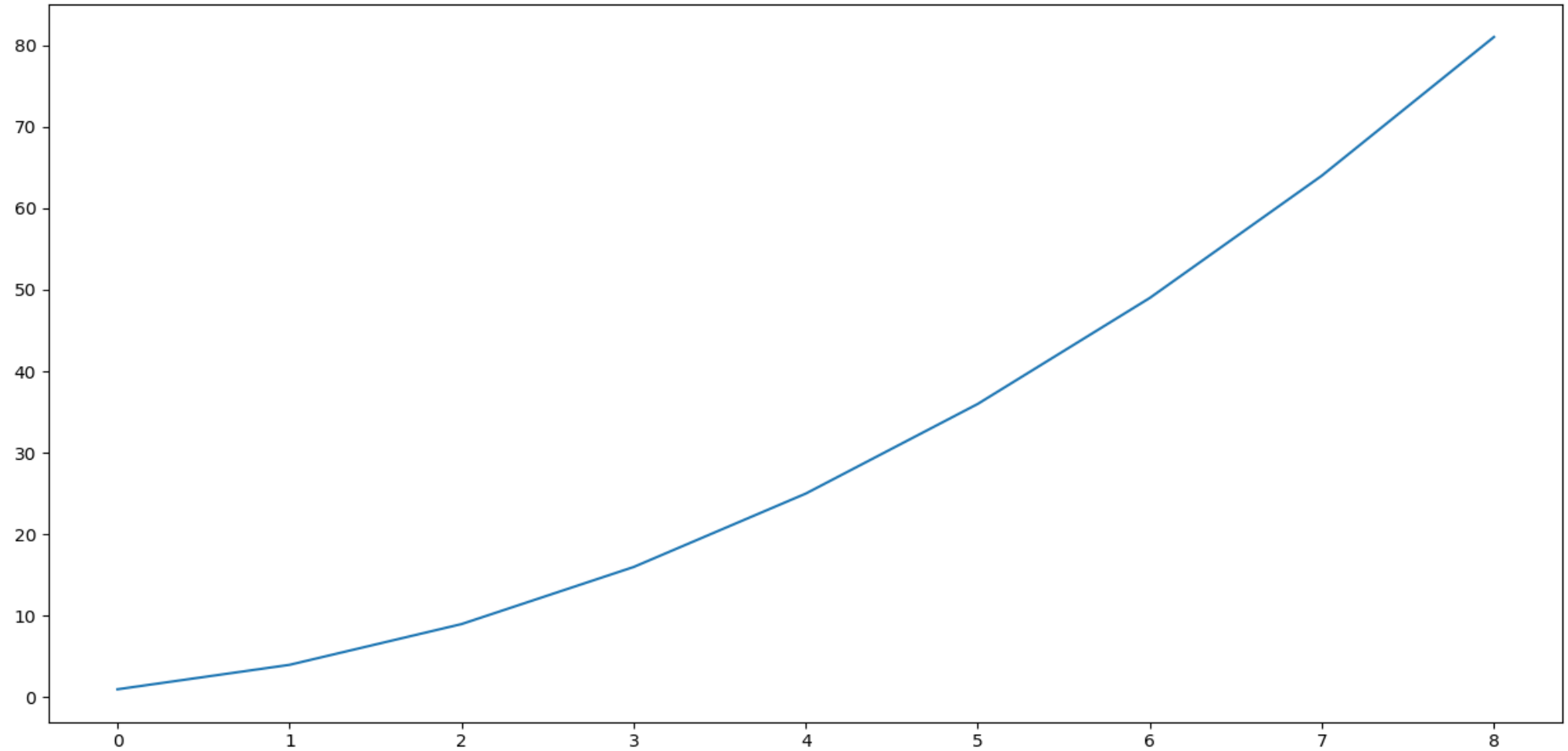
https://en.m.wikipedia.org/wiki/Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged.[3] SciPy makes use of Matplotlib.

https://www.techiediaries.com/python-matplotlib-tutorial/

Plotting line graphs

```
import matplotlib.pyplot as plt
squares = [x ** 2 for x in range(1,10)]
plt.plot(squares)
plt.show()
```


80
70
60
50
40
30
20
10
0
0
1
2
3
4
5
6
7
8

Label Type and Graph Thickness

Although the plot shown above illustrates that the numbers are increasing, the label type is too small and the line is too thin. Fortunately, matplotlib allows you to adjust every feature of a visualization.

The linewidth parameter of the plot method controls the thickness of the plotted line.

The title() method sets a title for the chart.

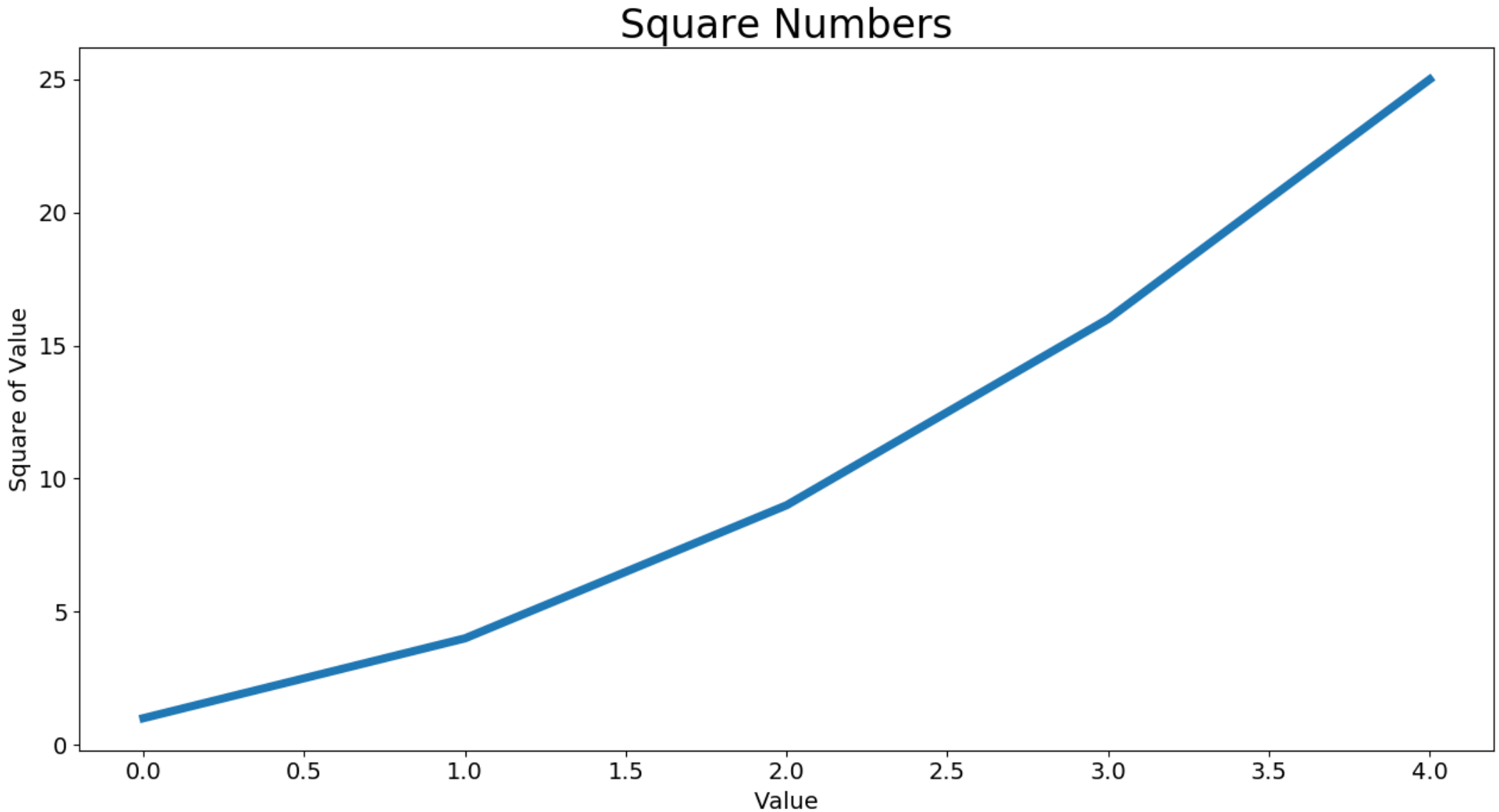
The fontsize parameters, which appear repeatedly throughout the code, control the size of the text on the chart.

The xlabel() and ylabel() functions allow you to set a title for each of the axes, and the function tick_params() styles the tick marks.

The arguments shown here affect the tick marks on both the x- and y-axes (axes='both') and set the font size of the tick mark labels to 14 (labelsize=14).

```
import matplotlib.pyplot as plt
plt.figure() #The figure() command here is optional because figure will be created by default
squares = [x ** 2 for x in range(1,6)]
print(squares)
plt.plot(squares, linewidth=5)
# Set chart title and label axes.
plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=14)
plt.ylabel("Square of Value", fontsize=14)
# Set size of tick labels.
plt.tick_params(axis='both', labelsize=14)
plt.show()
```

[1, 4, 9, 16, 25]


Square Numbers
Square of Value
Value
0
5
10
15
20
25
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

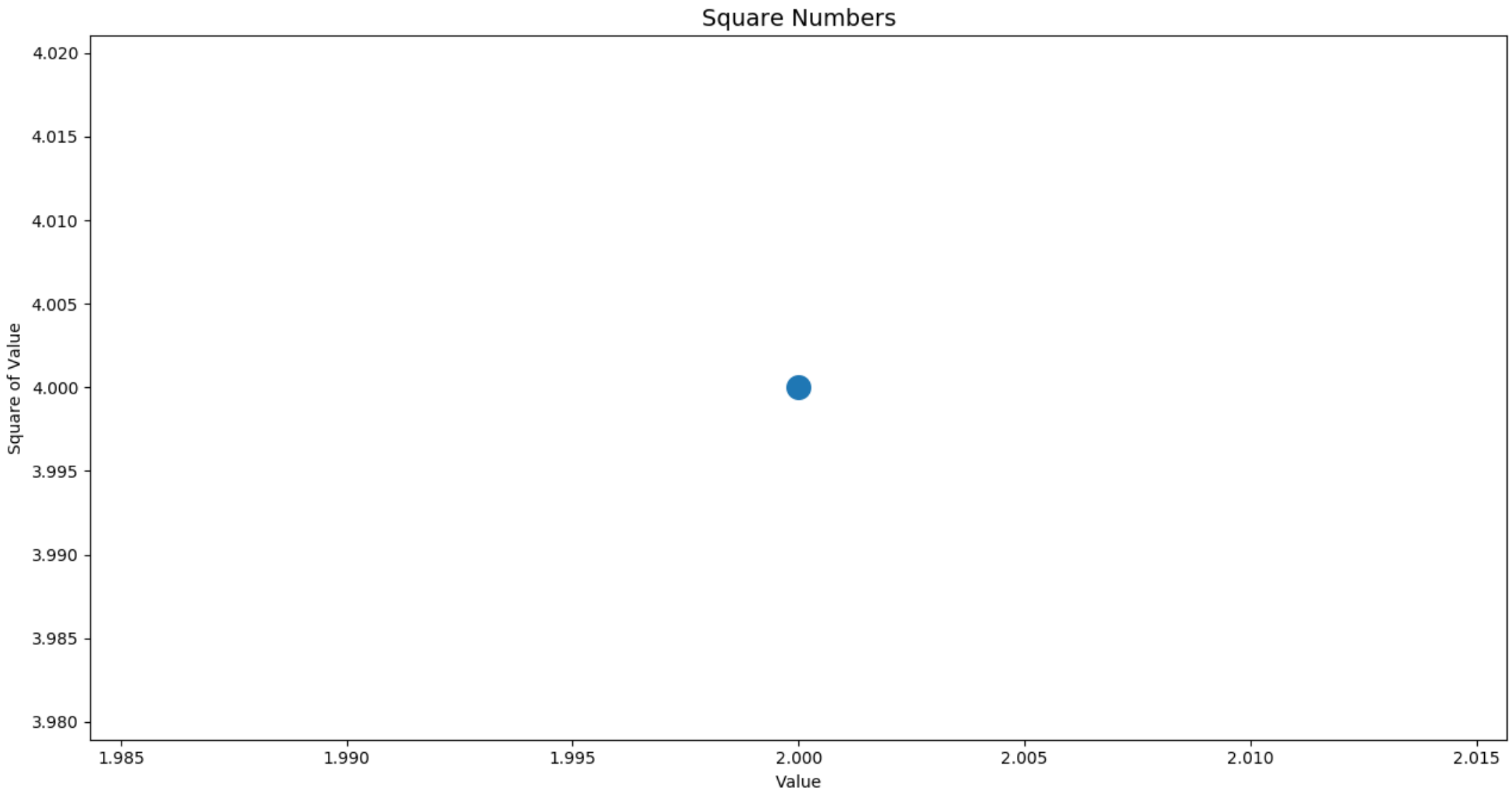
plt.scatter() and discrete data points (not calculated, whole numbers)

Discrete and Continuous Data.

A line plot is not necessarily the right visualization for discrete data point. Instead a plot of 'points', a scatter plot is more appropriate.

To plot a single point, use the scatter() function. Pass the single (x, y) values of the point of interest to scatter().

```
import matplotlib.pyplot as plt
plt.figure()
plt.scatter(2, 4, s=200) # s is the scalar value determines the area of the point.
# Set chart title and label axes.
plt.title("Square Numbers", fontsize=14)
plt.xlabel("Value", fontsize=10)
plt.ylabel("Square of Value", fontsize=10)
# Set size of tick labels.
plt.tick_params(axis='both', which='major', labelsize=10)
plt.show()
```


Square Numbers
Square of Value
4.020
4.015
4.010
4.005
4.000
3.995
3.990
3.985
3.980
1.985
1.990
1.995
2.000
2.005
2.010
2.015
Value

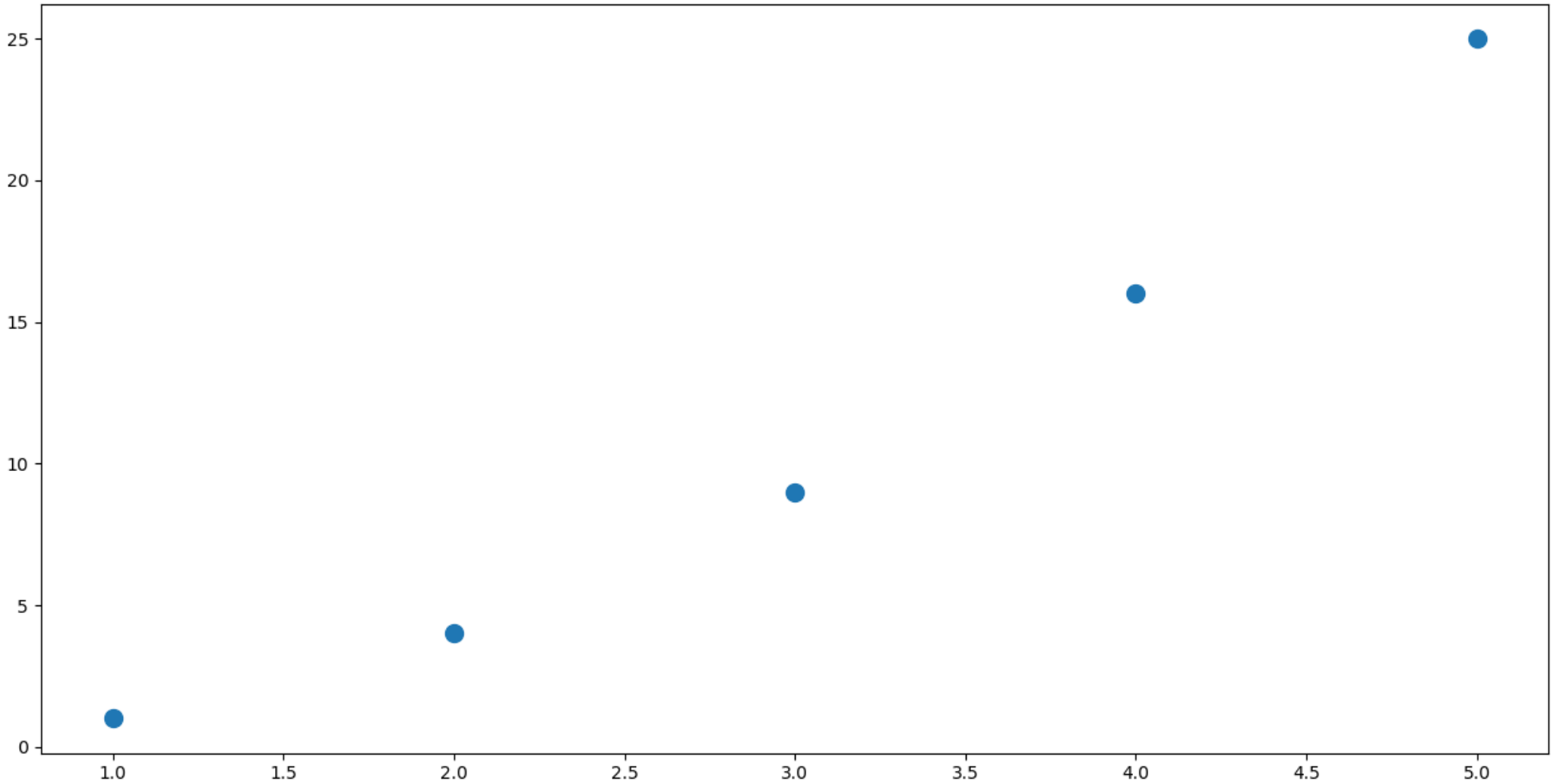
To plot a series of points, we can pass scatter() separate lists of x- and y-values. The x_values list contains the numbers to be squared, and y_values contains the square of each number. When these lists are passed to scatter(), matplotlib reads one value from each list as it plots each point. The points to be plotted are (1, 1), (2, 4), (3, 9), (4, 16), and (5, 25).

```
import matplotlib.pyplot as plt

plt.figure()

x_values = range(1, 6)

y_values = [x ** 2 for x in x_values]

print('Points to be plotted {}'.format(list(zip(x_values, y_values))))
```

The purpose of zip() is to map the similar index of multiple containers so that they can be used just using as single entity.

```
plt.scatter(x_values, y_values, s=100)

plt.show()
```

Points to be plotted [(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]


25
20
15
10
5
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

Removing Outlines from Data Points

matplotlib lets you color points individually in a scatter plot. The default— blue dots with a black outline—works well for plots with a few points. But when plotting many points, the black outlines can blend together. To remove the outlines around points, pass the argument edgecolor='none' when you call scatter()

Defining Custom Colors

To change the color of the points, pass c to scatter() with the name of a color to use, as shown here:

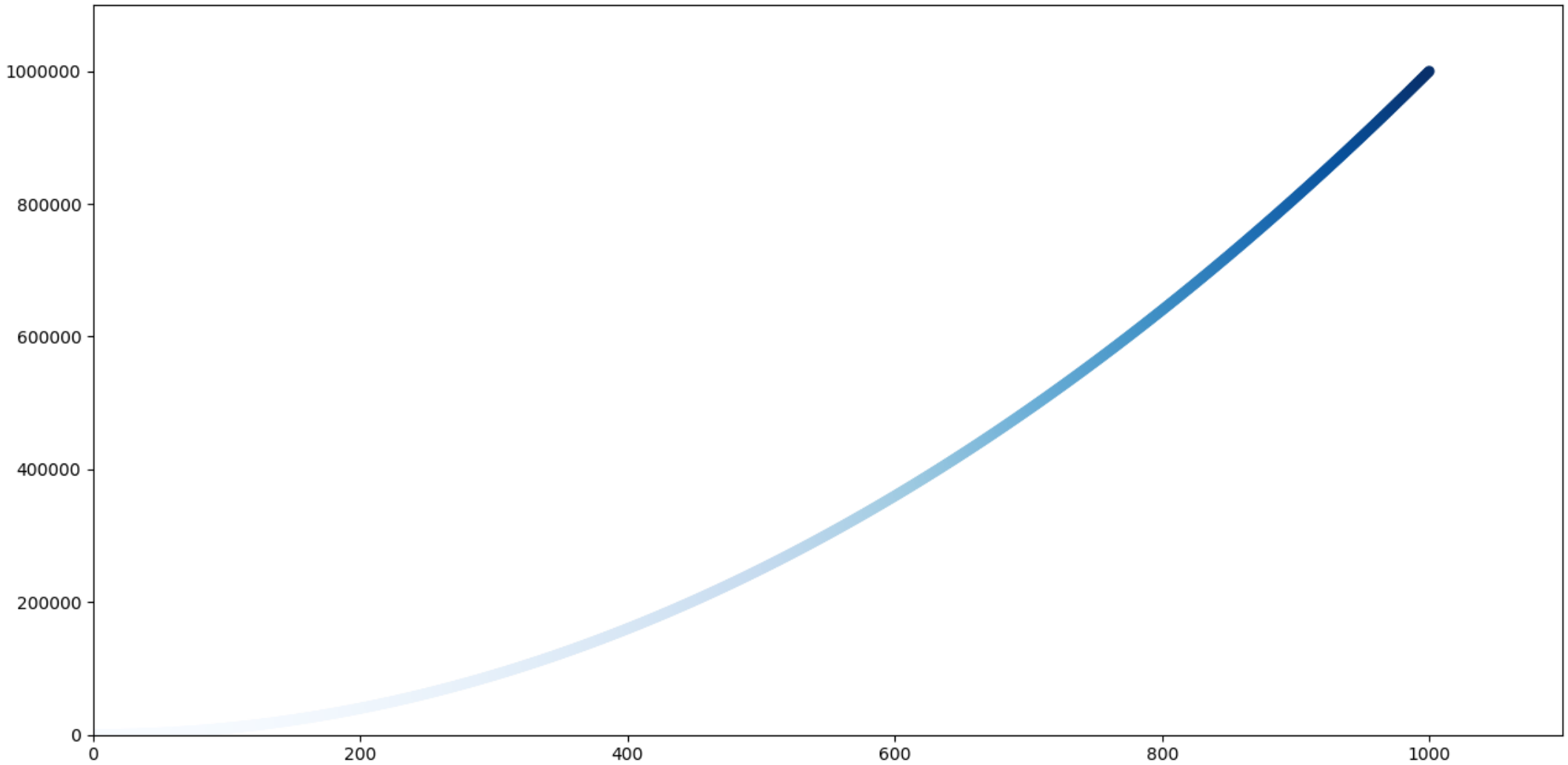
```
plt.scatter(x_values, y_values, c='red', edgecolor='none', s=40)
```

You can also define custom colors using the RGB color model. To define a color, pass the c argument a tuple with three decimal values (one each for red, green, and blue), using values between 0 and 1. For example, the following line would create a plot with light blue dots:

```
plt.scatter(x_values, y_values, c=(0, 0, 0.8), edgecolor='none', s=40)
```

Values closer to 0 produce darker colors, and values closer to 1 produce lighter colors.

We pass the list of y-values to c and then tell pyplot which colormap to use through the cmap argument. This code colors the points with lower y-values light blue and the points with larger y-values dark blue.


1000000
800000
600000
400000
200000
0
0
200
400
600
800
1000

Saving Plots

If you want your program to save the plot to a file, call plt.savefig():

```
plt.savefig('filename.png', bbox_inches='tight')
```

The first argument is a filename for the plot image, which will be saved in the same directory. The second argument trims extra whitespace from the plot. If you want the extra whitespace around the plot, you can omit this argument.

```
import matplotlib.pyplot as plt

x_values = range(1, 1001)
y_values = [x ** 2 for x in x_values]

plt.scatter(x_values, y_values, c=y_values, cmap=plt.cm.Blues, edgecolor='none', s=40)
plt.axis([0, 1100, 0, 1100000])

plt.savefig('squares_plot.png', bbox_inches='tight')
```

Bar Plots

We are going to have a look at the data about citizens in Copenhagen. http://data.kk.dk/dataset/befolkningen efter-ar-bydel-alder-og-statsborgerskab

In this section, we will mostly consider plotting histograms with bar plots. To do so, we have to first compute a bit of statistics. In the following there are computing how many people of a certain age lived in 2015 in Copenhagen.

Getting the data ready.

```
import kkdata
import matplotlib.pyplot as plt

neighbourhoods_in_2015_data = kkdata.STATISTICS[2015].keys()
print('neighbourhoods: ',neighbourhoods_in_2015_data)
print('age range of neighbourhood 1', kkdata.STATISTICS[2015][1].keys(),'\n')
print('20 year olds in hood 1', kkdata.STATISTICS[2015][1][20], '\n')
print('20 year old danes in hood 1 (danes=5100)',
kkdata.STATISTICS[2015][1][20][5100])
neighbourhoods = neighbourhoods_in_2015_data
```

neighbourhoods: dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 99])

age range of neighbourhood 1 dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,

53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 102, 103, 106])

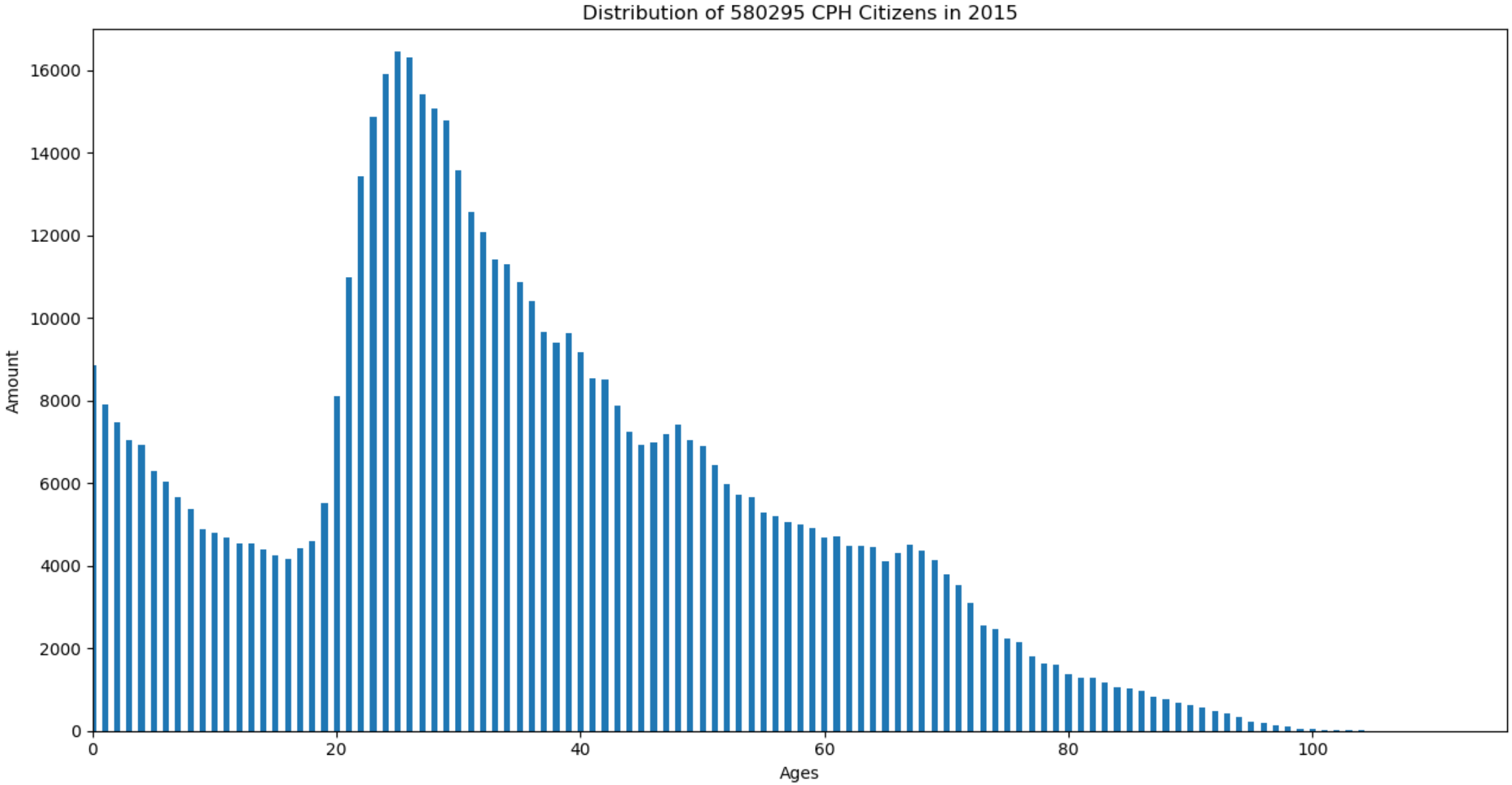
20 year olds in hood 1 {5100: 678, 5104: 3, 5106: 1, 5110: 21, 5120: 5, 5128: 1, 5130: 6, 5140: 2, 5150: 2, 5154: 6, 5158: 2, 5164: 1, 5170: 6, 5174: 3, 5180: 4, 5182: 1, 5244: 1, 5302: 1, 5314: 3, 5390: 8, 5432: 1, 5442: 1, 5448: 1, 5482: 1, 5486: 1, 5502: 1, 5704: 1, 5776: 1, 5778: 1}

20 year old danes in hood 1 (danes=5100) 678

```
stats = get_population_stat(2015)
ages = list(stats.keys())
no_citicens = list(stats.values())
```

```
def get_population_stat(year_of_interest=2015):
    neighbourhoods = kkdata.STATISTICS[year_of_interest].keys()
    age_range = set([])
    for n in neighbourhoods:
        age_range.update(kkdata.STATISTICS[year_of_interest][n].keys())

    no_citicens_per_age = {}

    for n in neighbourhoods:
        for age in age_range:
            if age in kkdata.STATISTICS[year_of_interest][n].keys():
                c_codes = set(kkdata.STATISTICS[year_of_interest][n][age].keys())
                for f_code in c_codes:
                    no_citicens_per_age.setdefault(age, 0)
                    no_citicens_per_age[age] +=
kkdata.STATISTICS[year_of_interest][n][age][f_code]

    return no_citicens_per_age
```

```
#plt.cla()
plt.bar(ages, no_citicens, width=0.5, align='center') # bar(x-vals, y-vals,
        bar width, align bar relative to x-val on x-axis) )
#plt.ticklabel_format(useOffset=False)
plt.axis([0, max(ages) + 10, 0, 17000]) #axis(x-min, x-max, y-min, y-max)
         title = 'Distribution of {} CPH Citizens in
         {}'.format(sum(no_citicens), 2015)
plt.title(title, fontsize=12)
plt.xlabel("Ages", fontsize=10)
plt.ylabel("Amount", fontsize=10)
plt.tick_params(axis='both', which='major', labelsize=10)
plt.show()
```


Distribution of 580295 CPH Citizens in 2015
Amount
Ages
0
2000
4000
6000
8000
10000
12000
14000
16000
0
20
40
60
80
100

2 data sets displayed together

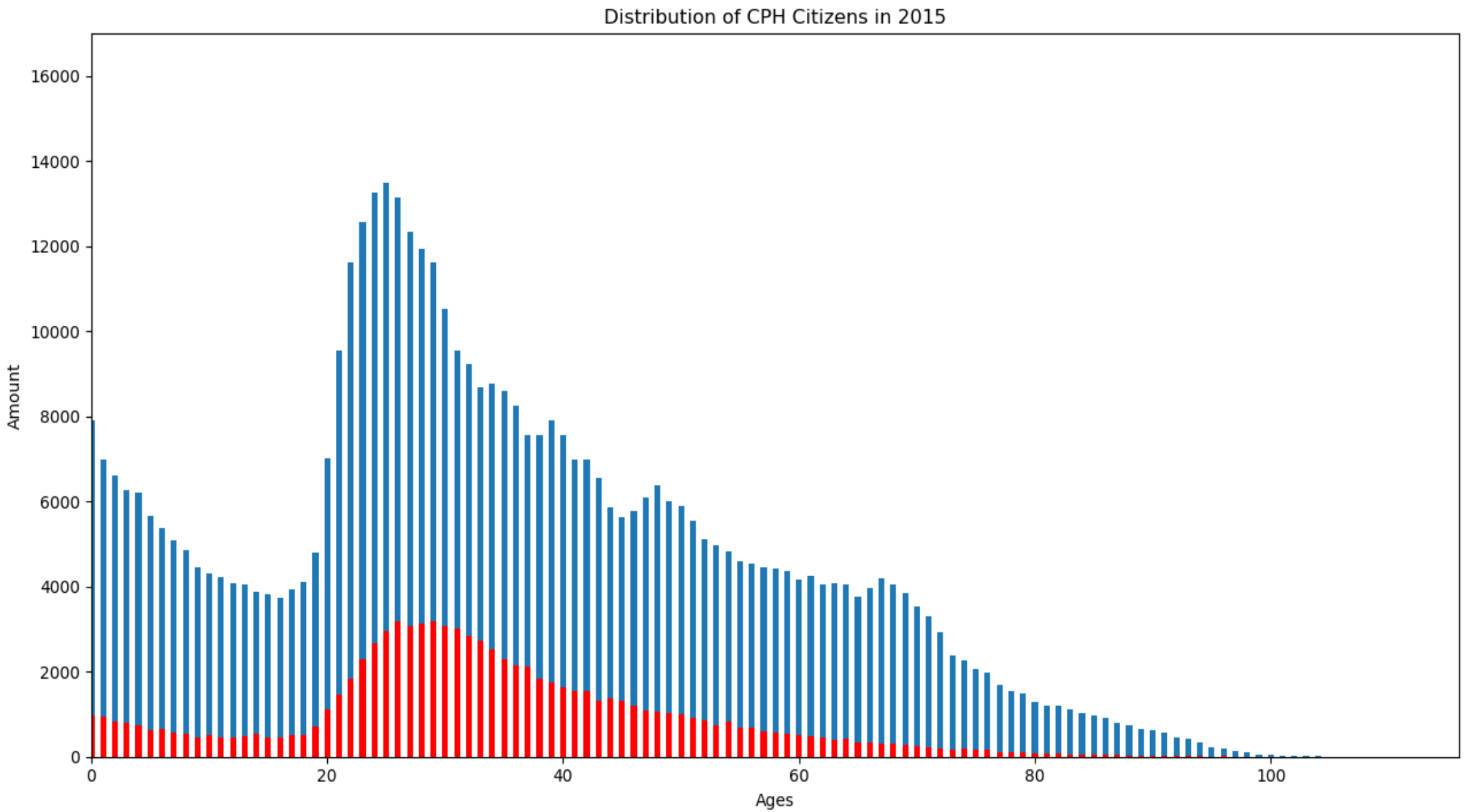
Now we are going to extend our function that computes the statistics, so that it can distinguish between Copenhagen citicens who are Danish nationals and those who are expats.

```
import kkdata
import matplotlib.pyplot as plt


def get_population_stat():
        age_range = set([])
    for n in neighbourhoods:
        age_range.update(kkdata.STATISTICS[2015][n].keys())
    no_danes_per_age = {}
    no_foreign_per_age = {}
```

```
for n in neighbourhoods:
        for age in age_range:
            if age in kkdata.STATISTICS[2015][n].keys():
                c_codes = set(kkdata.STATISTICS[2015][n][age].keys())
                if 5100 in c_codes:
                    no_danes_per_age.setdefault(age, 0)
                    no_danes_per_age[age] += kkdata.STATISTICS[2015][n][age][5100]
                    c_codes.remove(5100)
                for f_code in c_codes:
                    no_foreign_per_age.setdefault(age, 0)
                    no_foreign_per_age[age] += kkdata.STATISTICS[2015][n][age][f_code]

    return no_danes_per_age, no_foreign_per_age
# returning a comma separated set of elements creates a tuple
```

```
ages = list(danes_per_age.keys())
no_citicens = list(danes_per_age.values())
ages_f = list(foreigners_per_age.keys())
no_citicens_f = list(foreigners_per_age.values())
```

```
plt.bar(ages, no_citicens, width=0.5, linewidth=0, align='center')
plt.ticklabel_format(useOffset=False)
plt.axis([0, max(ages) + 10, 0, 17000])
title = 'Distribution of CPH Citizens in {}'.format(2015)
plt.title(title, fontsize=12)
plt.xlabel("Ages", fontsize=10)
plt.ylabel("Amount", fontsize=10)
plt.tick_params(axis='both', which='major', labelsize=10)

plt.bar(ages_f, no_citicens_f, width=0.5, linewidth=0, align='center', color='red')

fakes = list(range(10900, 300, -100))
print(len(ages), '\n\n', no_citicens, '\n\n', len(fakes))
#plt.bar(ages, fakes, width=0.5, linewidth=0, align='center', color='yellow')

plt.show()
```


Distribution of CPH Citizens in 2015
Amount
Ages
0
2000
4000
6000
8000
10000
12000
14000
16000
0
20
40
60
80
100

Assignment.

The following must be done:

1. A plot showing the maximum, minimum and average time of the simple recursive algorithm for solving the maze.

2. A plot showing maximum, minimum, and average iterations for the simple recursive algorithm for solving the maze.

You have 2 weeks for it and it needs to be reviewed with your review group before you review it with me.