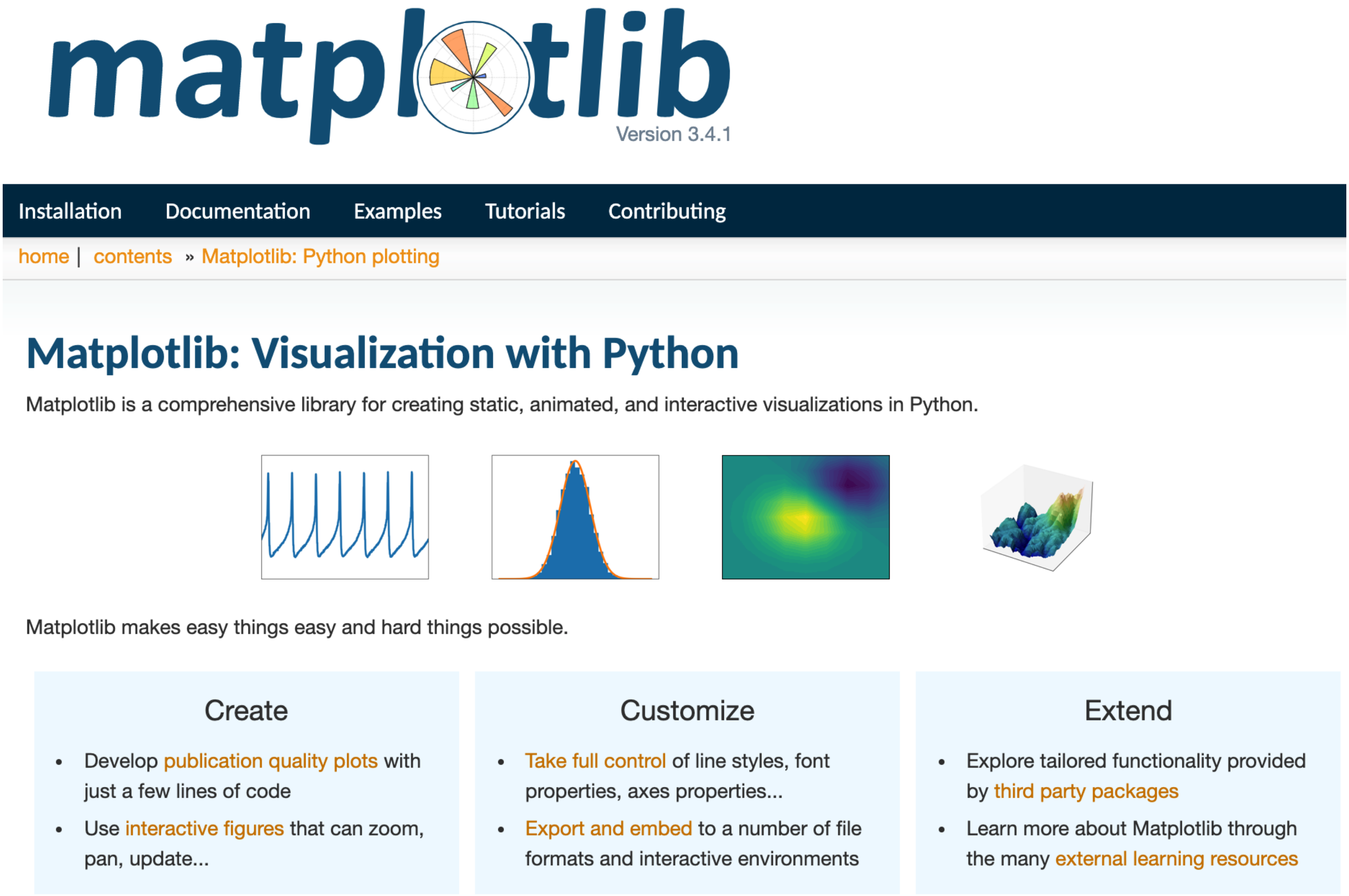
Table of Contents

Matplotlib module

https://matplotlib.org/stable/index.html (https://matplotlib.org/stable/index.html)



Matplotlib: Visualization with Python

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Matplotlib makes easy things easy and hard things possible.

Create

- Develop publication quality plots with just a few lines of code
- Use interactive figures that can zoom, pan, update...

Customize

- Take full control of line styles, font properties, axes properties...
- Export and embed to a number of file formats and interactive environments

Extend

- Explore tailored functionality provided by third party packages
- Learn more about Matplotlib through the many external learning resources

Check the gallery for getting a glimpse of what you can do with matplotlib!

https://matplotlib.org/stable/gallery/index.html (https://matplotlib.org/stable/gallery/index.html)

Need to **import the module** and define some convenient abbreviations

In [1]:

```
import matplotlib.pyplot as plt  # import the module

import matplotlib as mpl
mpl.rcParams['figure.dpi']= 130  # set the resolution to x dpi
```

Let's read a **CSV file** with financial data. We want to inspect the behaviors of a few indicators, such as the closing value, the opening value, and the trade volume.

In [2]:

```
f = open('AAPL.csv')
```

Let's deal with the file *without* using the methods from the `csv` module (just for the sake of playing a bit with string methods!)

The first line of the file, the header, contains the names/descriptions of the fields (the columns of the dataset)

In [3]:

```
header = f.readline()
```

In [4]:

```
header
```

Out[4]:

```
'Date,Open,High,Low,Close,Adj Close,Volume\n'
```

It's a *comma* separated values file, therefore, we can extract the individual fields by splitting on `','`

In [5]:

```
fieldnames = header.split(',')
```

In [6]:

```
fieldnames
```

Out[6]:

```
['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume\n']
```

Not exacty what we'd like to have: there's a `\n` character that we don't want.

Moreover, what about the potential presence of white spaces?

In [7]:

```
f.seek(0)
header = f.readline()
fieldnames = header[:-1].split(',')
fieldnames
```

Out[7]:

```
['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']
```

We can read now the file record by record and store the values of the columns `Date, Open, Close, Volume` in different lists (note that we will exploit the knowledge of what content is reported in a column).

Apart from `Date` , all values (that are read as strings) are numeric, such that are casted as floats.

In [8]:

```
open_val = []
close_val = []
volume = []
dates = []

for record in f:
        fields = record.split(',')

        #print(fields)

        dates.append(fields[0])
        open_val.append( float(fields[1]) )
        close_val.append( float(fields[4]) )
        volume.append( float(fields[6]) )
```

In [9]:

```
#open_val
```

How many records in total?

In [10]:

```
len(close_val)
```

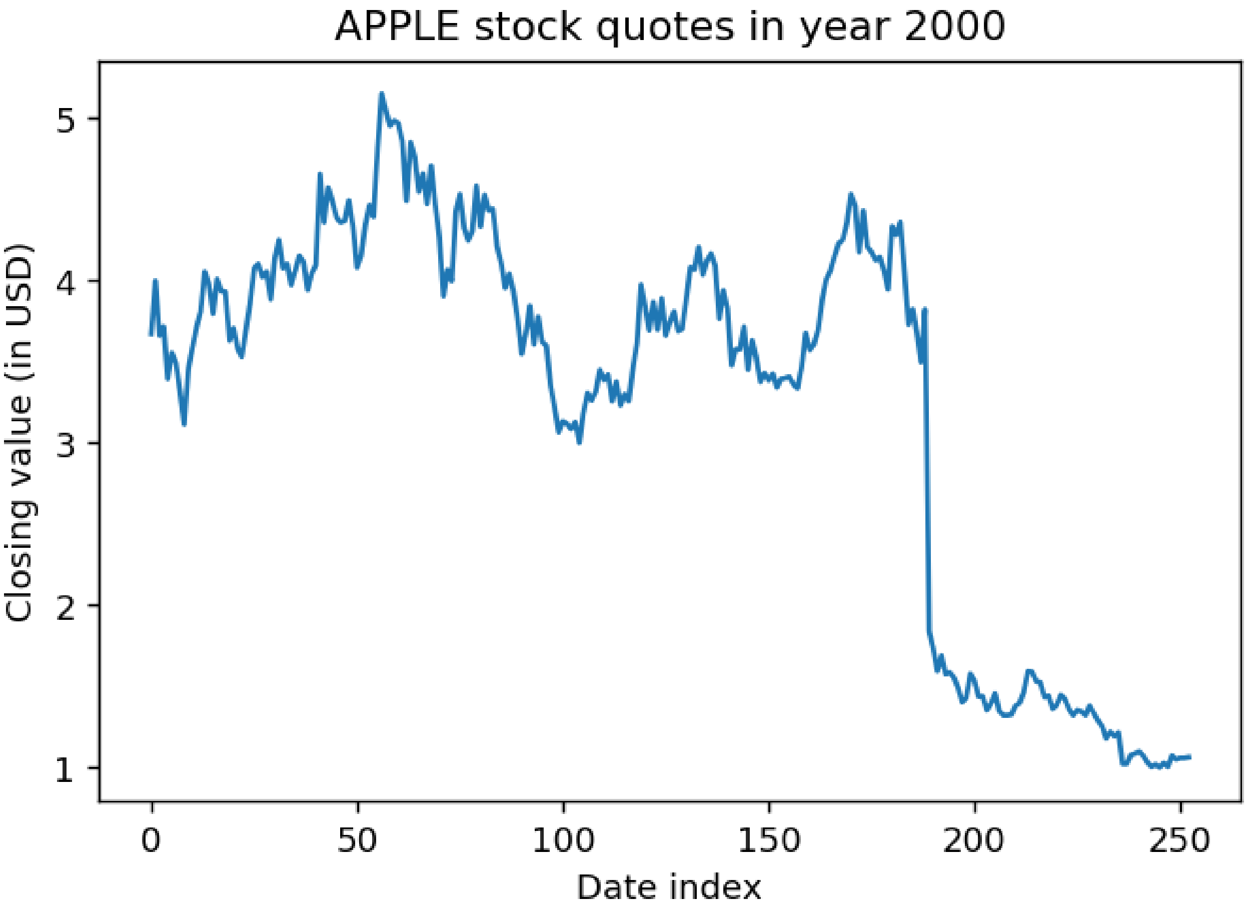
Out[10]:

```
253
```

How do we check what has happened for the Aplle shares? How do we inspect the data visually? Let's make **data plots!**

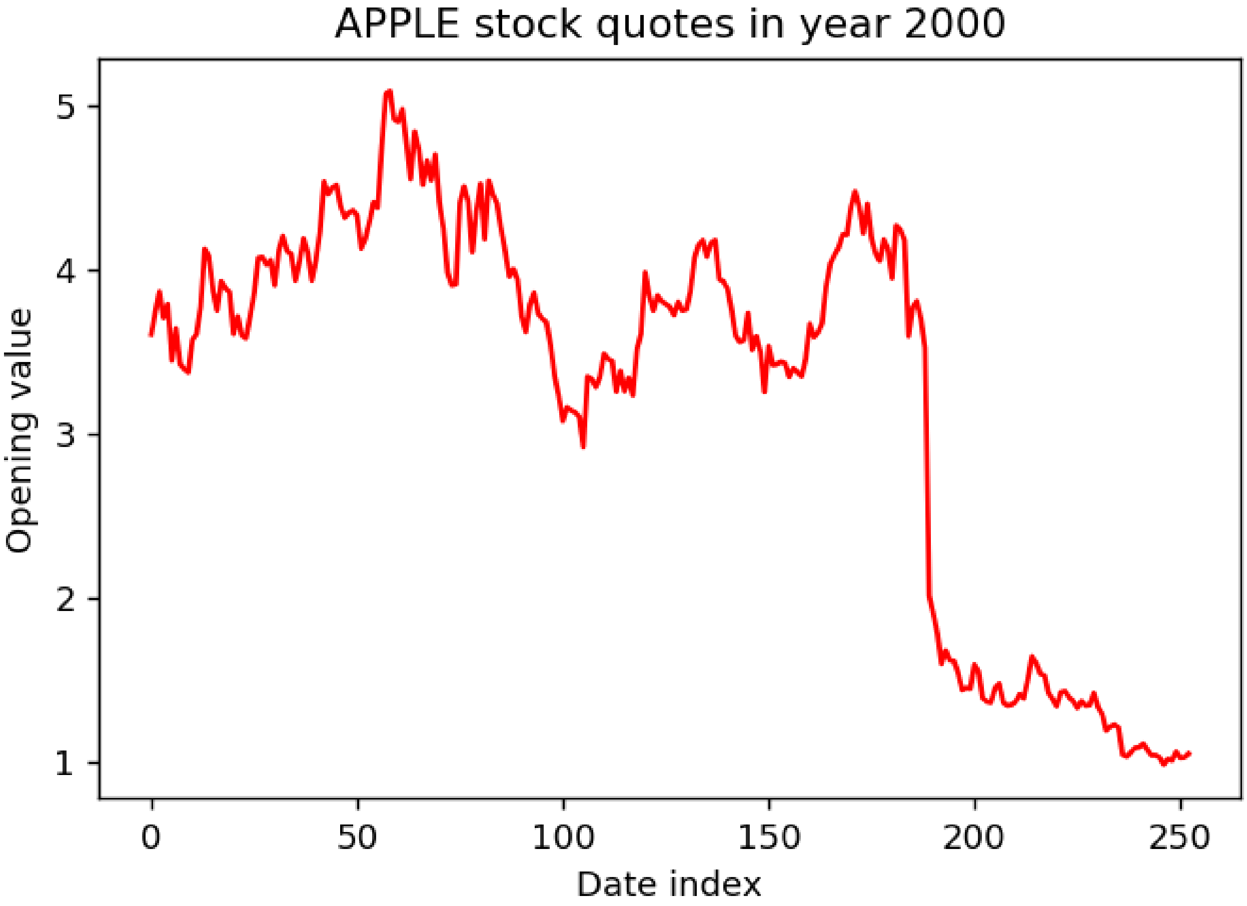
In [11]:

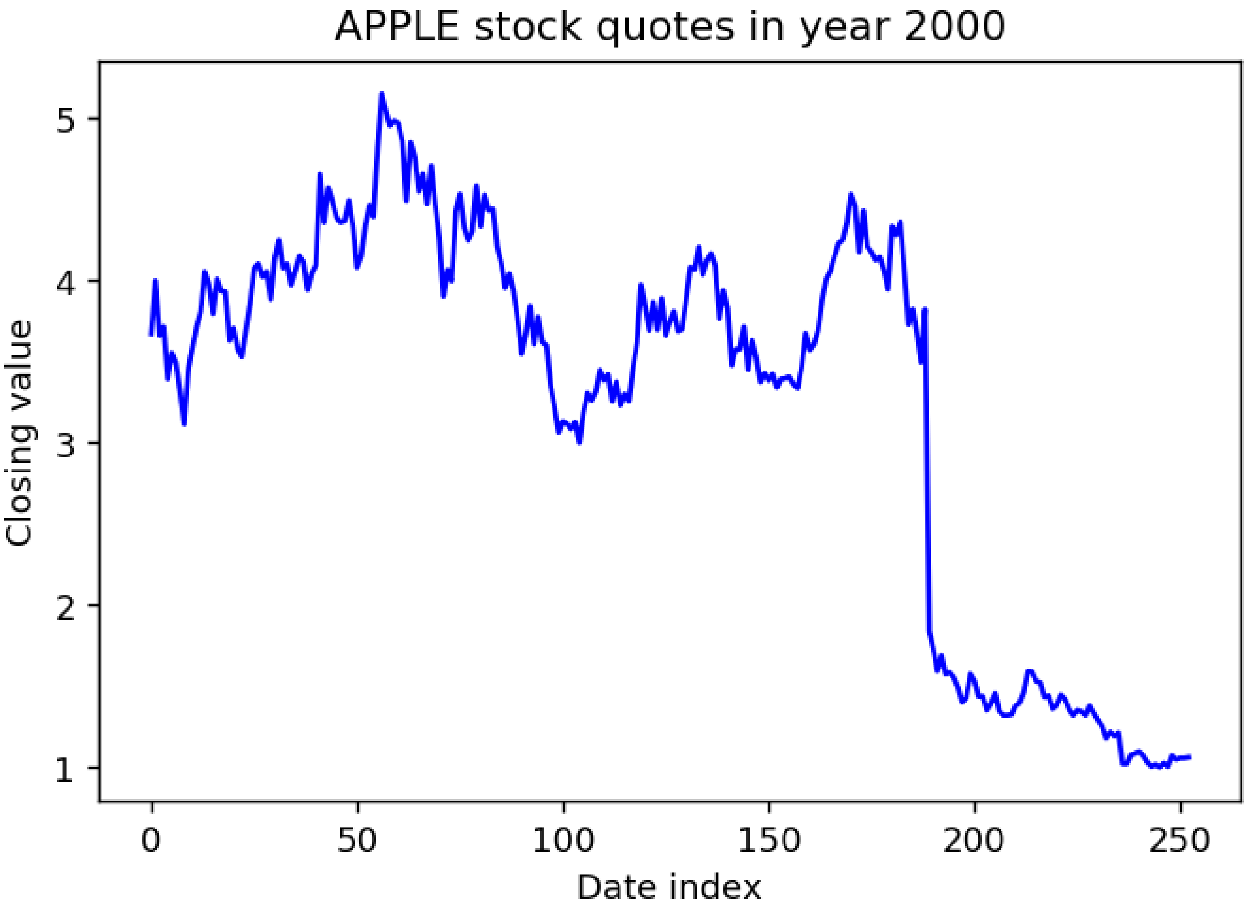
```
plt.title("APPLE stock quotes in year 2000")  # the title to give to the figure

plt.xlabel("Date index")  # x-axis label

plt.ylabel("Closing value (in USD)")  # y-axis label

plt.plot(close_val)

plt.show()
```

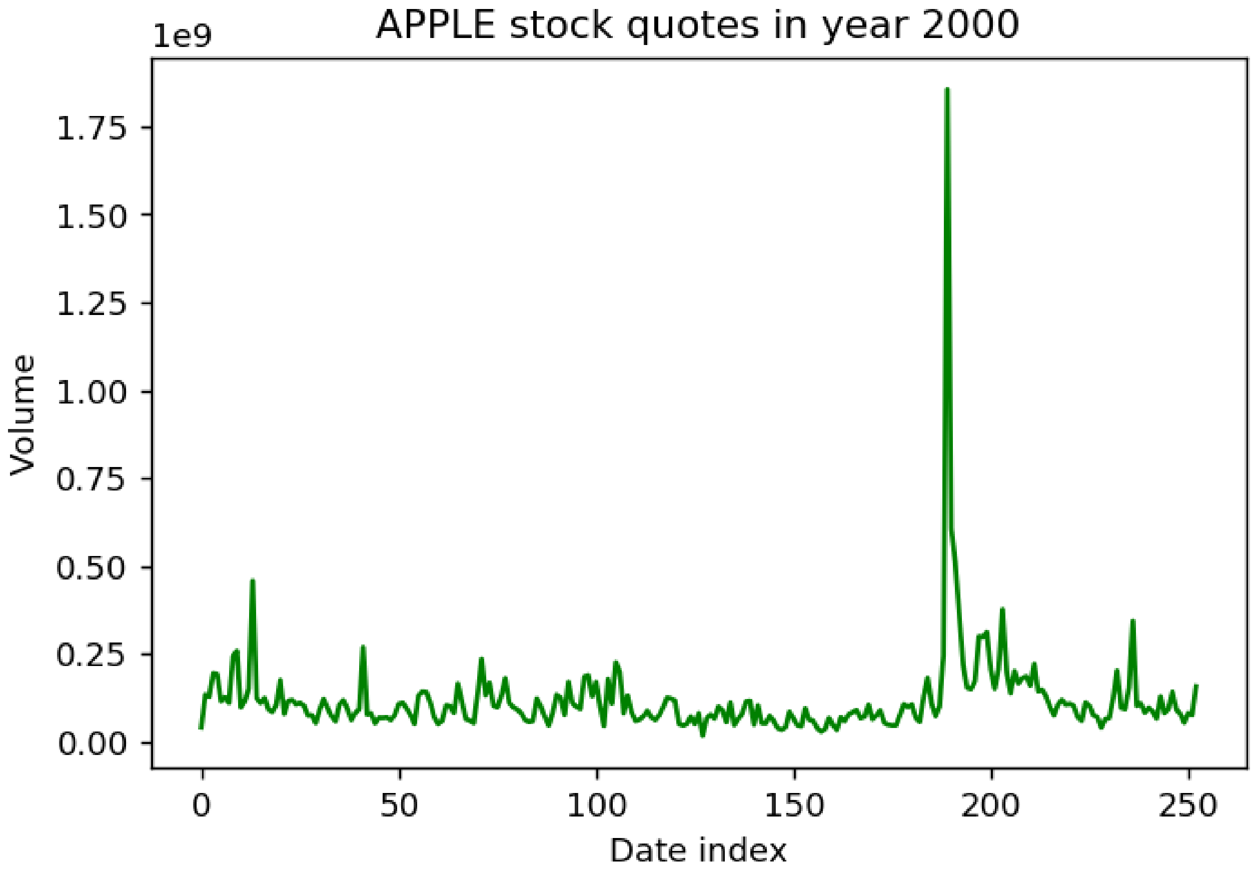

APPLE stock quotes in year 2000
Closing value (in USD)
1
2
3
4
5
0
50
100
150
200
250
Date index

In [12]:

```
for p in [open_val, close_val, volume]:

    plt.title("APPLE stock quotes in year 2000")  # the title to give to the figure

    plt.xlabel("Date index")  # x-axis label

    if p == open_val:
        ylabel = "Opening value"
        color = 'red'

    elif p == close_val:
        ylabel = "Closing value"
        color = 'blue'

    else:
        ylabel = "Volume"
        color = 'green'

    plt.ylabel(ylabel)  # y-axis label

    plt.plot(p, color = color)

    plt.show()
```


APPLE stock quotes in year 2000
Opening value
5
4
3
2
1
0
50
100
150
200
250
Date index

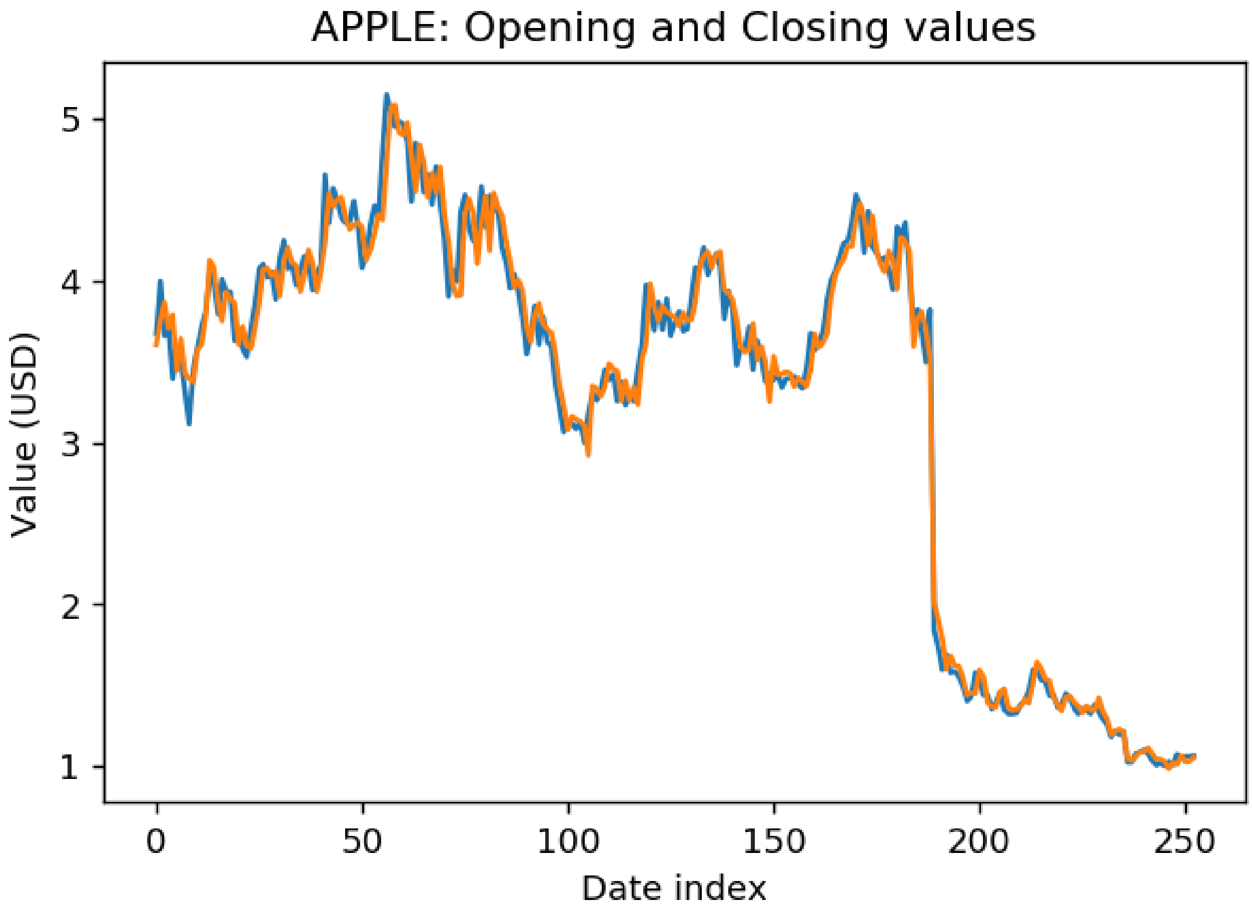

APPLE stock quotes in year 2000
Closing value
1
2
3
4
5
0
50
100
150
200
250
Date index



Closing and opening values look similar, can we compare them in the same data plot? $\to$ **Multiple plots**

In [13]:

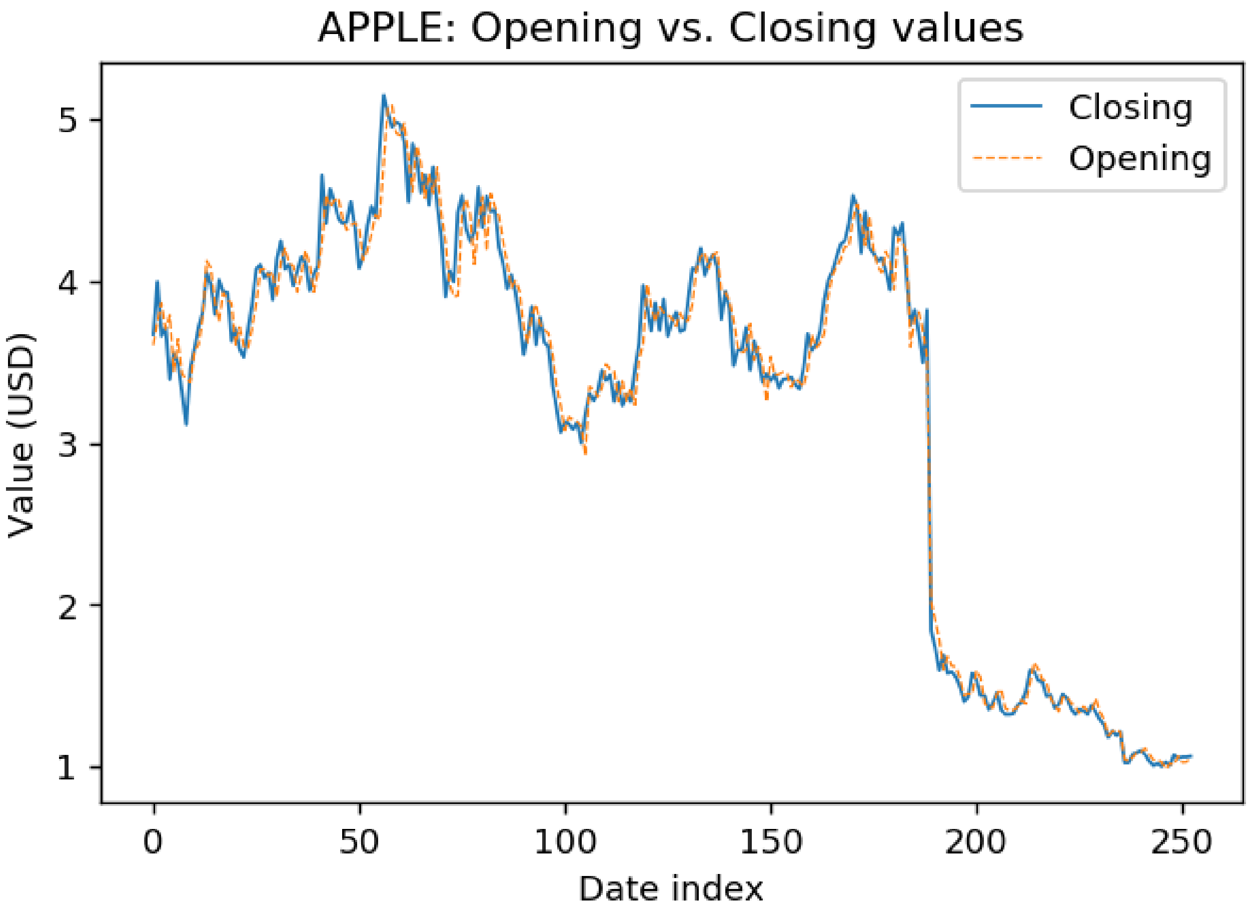
```
plt.title("APPLE: Opening and Closing values")

plt.xlabel("Date index")

plt.ylabel("Value (USD)")

plt.plot(close_val)

plt.plot(open_val)

plt.show()
```



Can we improve readability by controlling **style / width of lines** and adding a **legend?**

In [14]:

```
plt.title("APPLE: Opening vs. Closing values")

plt.xlabel("Date index")

plt.ylabel("Value (USD)")

plt.plot(close_val, linewidth=1, label = 'Closing')

plt.plot(open_val, linewidth=0.7, linestyle='dashed', label = 'Opening')

plt.legend()

plt.show()
```



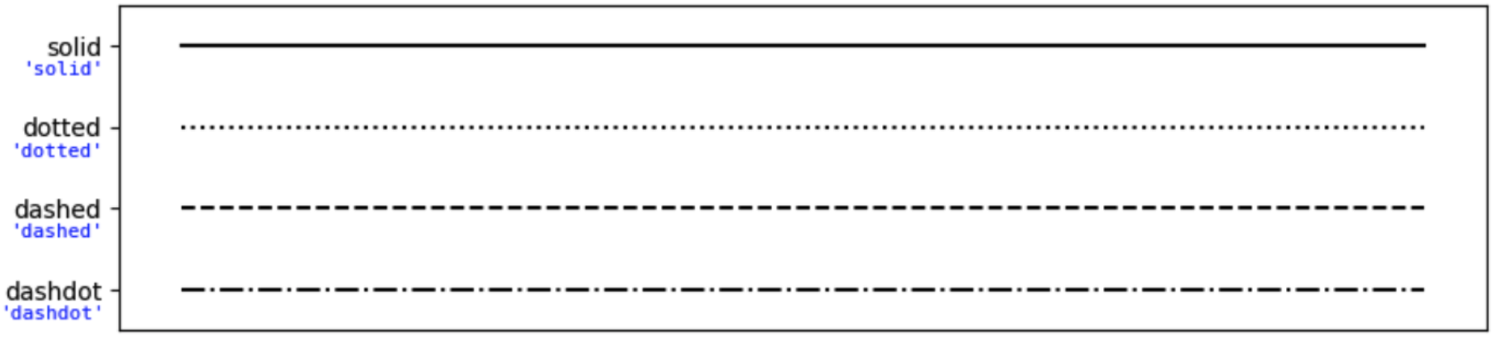
Different linestyles for the lines:

Solid: '_'

Dotted: '.'

Dashed: '--'

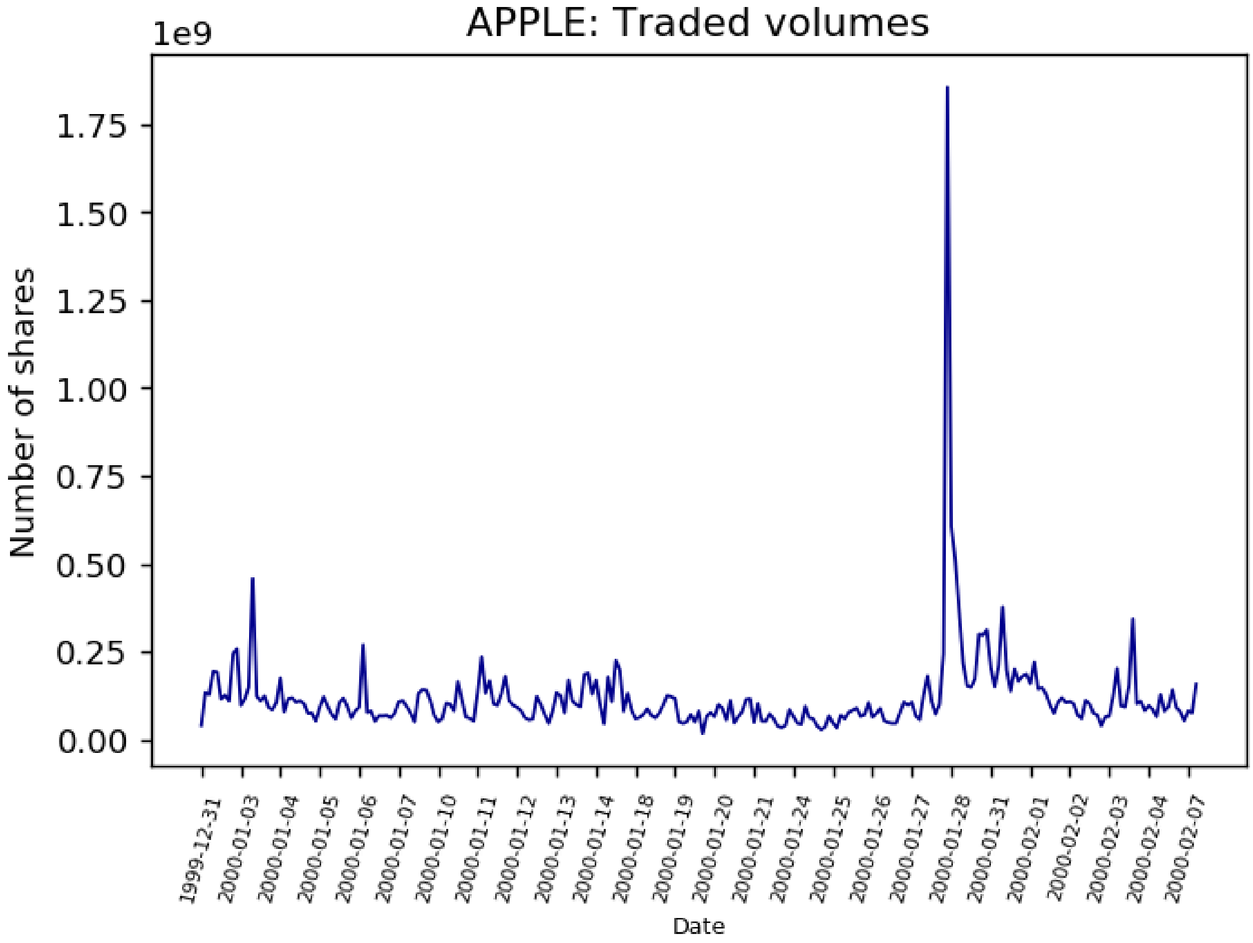
Dashdot: '_.'



What about the **dates** on the x axis? Can we write them in a nice way? $\to$ **Control ticks/labels on the axes**

In [15]:

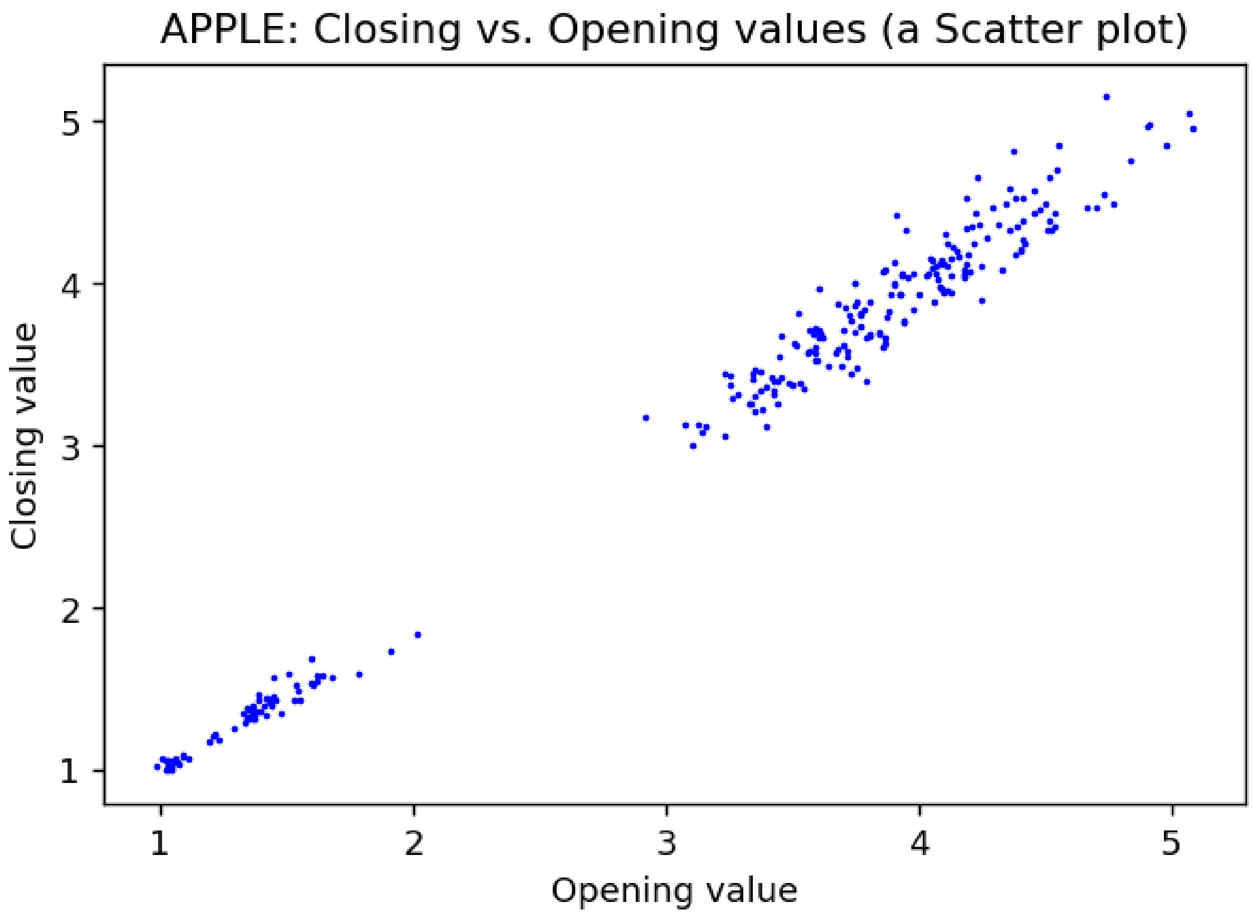
```
plt.title("APPLE: Traded volumes")

plt.xlabel("Date", fontsize=7)

plt.ylabel("Number of shares")

n_values = len(volume)
plt.xticks(range(0, n_values, 10), dates,
           fontsize=6, rotation=75)

plt.plot(volume, linewidth=1, color = 'darkblue')


plt.show()
```



So far we have only plotted one time series of data. What about correlation between different series of paired data? $\to$ **(x,y) data plots** showing y vs. x $\to$ **scatter plots**

In [16]:

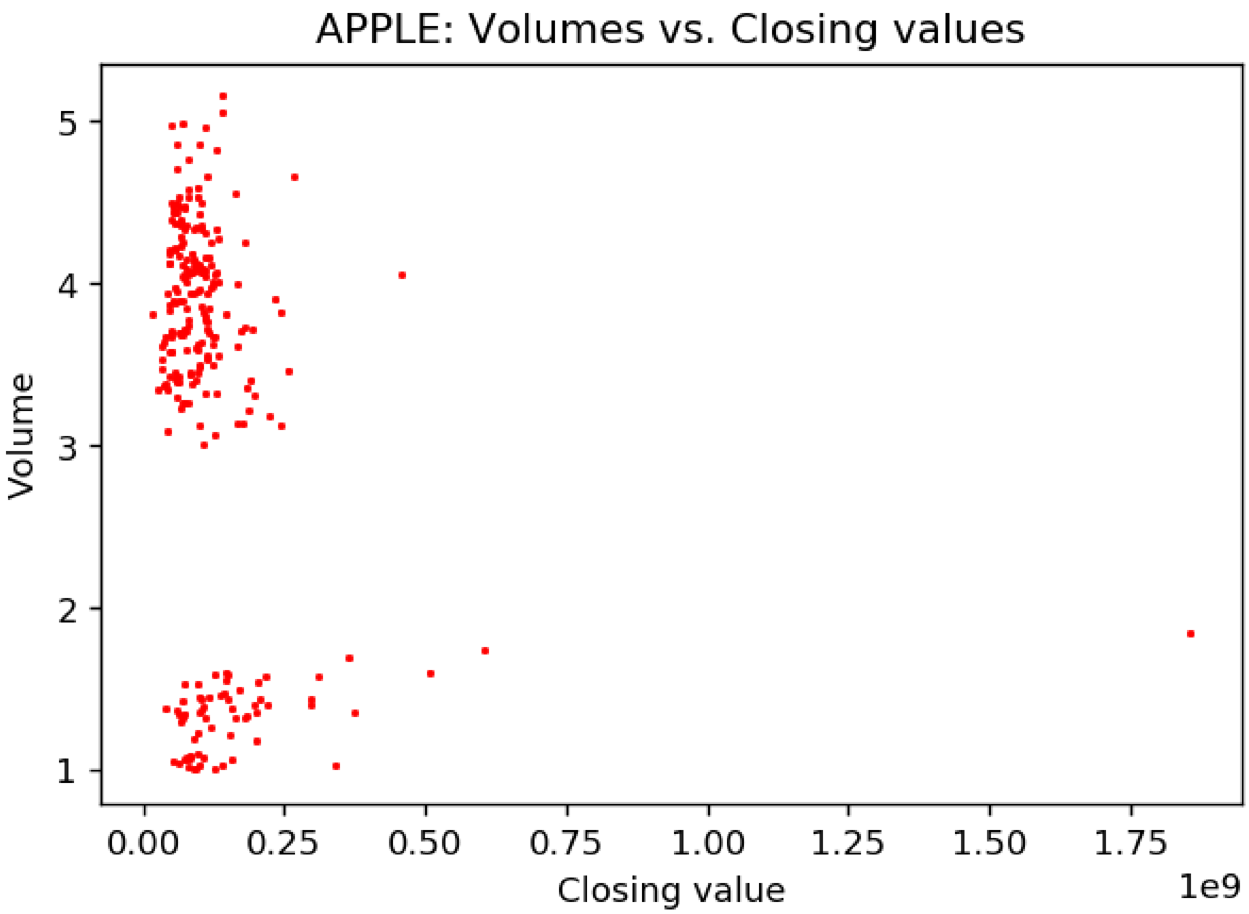
```
plt.title("APPLE: Closing vs. Opening values (a Scatter plot)")

plt.xlabel("Opening value")

plt.ylabel("Closing value")

plt.scatter(open_val, close_val, marker='o', s=1, color='blue')

plt.show()
```



Volumes vs. Closing values, and using a different marker for the points.

In [17]:

```
plt.title("APPLE: Volumes vs. Closing values")

plt.xlabel("Closing value")

plt.ylabel("Volume")

plt.scatter(volume, close_val, marker='s', s=1, color='red')

plt.show()
```



How many **different markers** can we use? A lot! A the named parameter `s` defines the size of the marker. Try them out to see the effect!

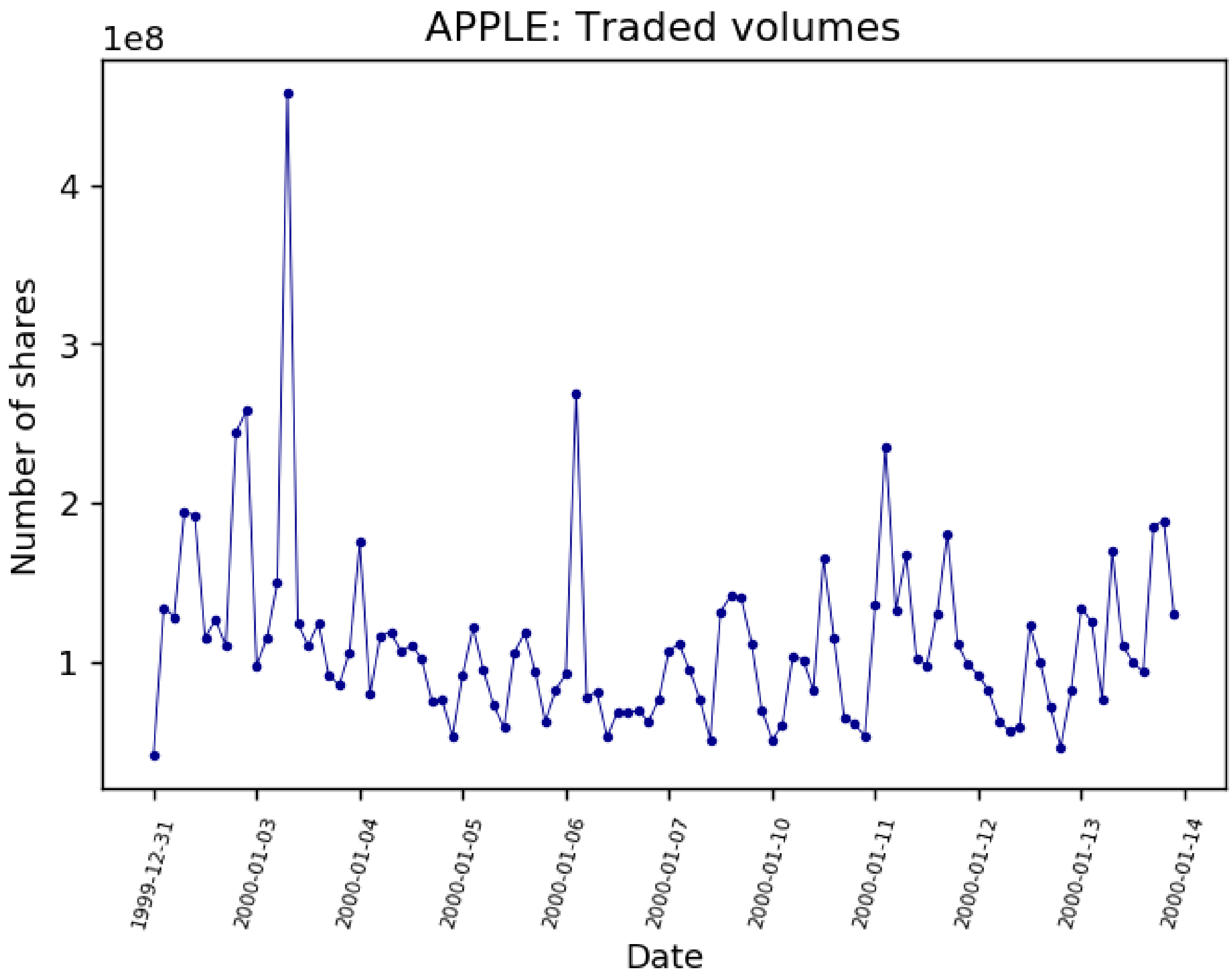
In [18]:

```
# Many different markers are available for the shape of the data points
#    =============    ===============================
#    character        description
#    =============    ===============================
#    ``'.'``          point marker
#    ``','``          pixel marker
#    ``'o'``          circle marker
#    ``'v'``          triangle_down marker
#    ``'^'``          triangle_up marker
#    ``'<'``          triangle_left marker
#    ``'>'``          triangle_right marker
#    ``'1'``          tri_down marker
#    ``'2'``          tri_up marker
#    ``'3'``          tri_left marker
#    ``'4'``          tri_right marker
#    ``'s'``          square marker
#    ``'p'``          pentagon marker
#    ``'*'``          star marker
#    ``'h'``          hexagon1 marker
#    ``'H'``          hexagon2 marker
#    ``'+'``          plus marker
#    ``'x'``          x marker
#    ``'D'``          diamond marker
#    ``'d'``          thin_diamond marker
#    ``'|'``          vline marker
#    ``'_'``          hline marker
#    =============    ===============================
```

Plots with **lines and point markers?**

In [19]:

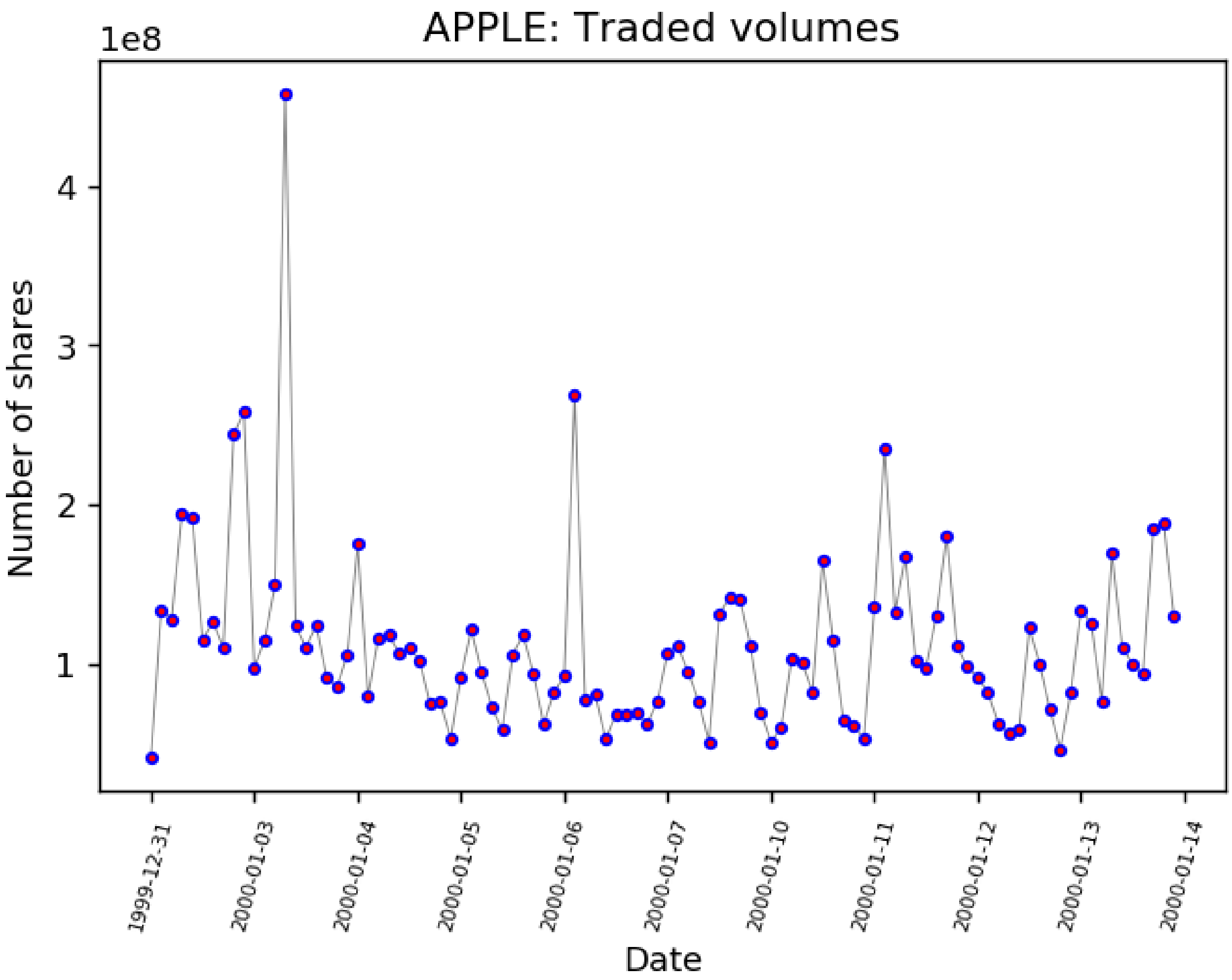
```
plt.title("APPLE: Traded volumes")

plt.xlabel("Date")

plt.ylabel("Number of shares")

n_values = len(volume)
plt.xticks(range(0, n_values, 10), dates,
           fontsize=6, rotation=75)

plt.plot(volume[0:100],
         marker = 'o', markersize=2,
         linewidth=0.5, color = 'darkblue')


plt.show()
```



We can also change the **colors of the markers (borders and inside).**

In [20]:

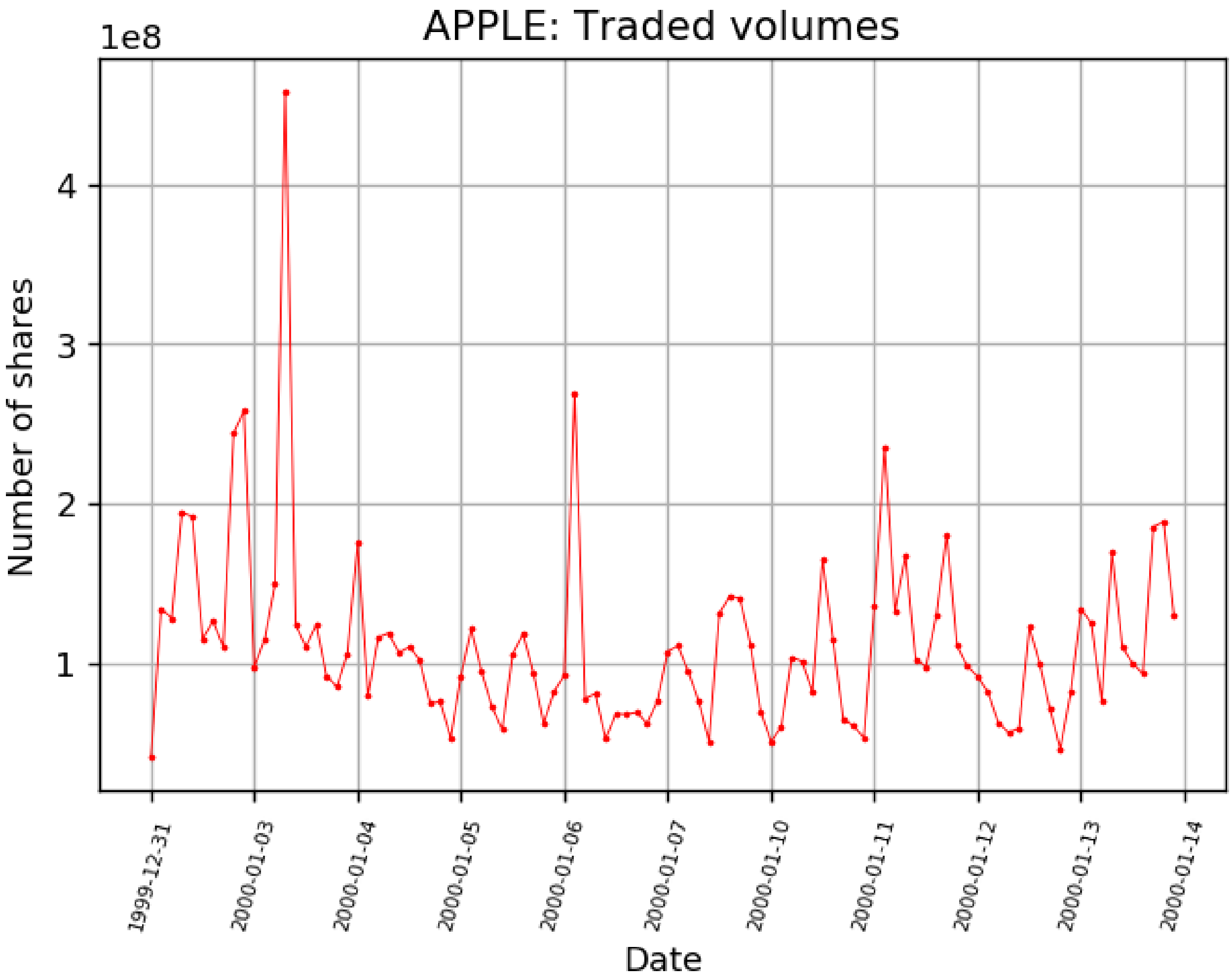
```
plt.title("APPLE: Traded volumes")

plt.xlabel("Date")

plt.ylabel("Number of shares")

n_values = len(volume)
plt.xticks(range(0, n_values, 10), dates,
           fontsize=6, rotation=75)

plt.plot(volume[0:100],
         marker = 'o', markersize=3,
         markerfacecolor='r', markeredgecolor='blue',
         linewidth=0.5, color = 'gray')


plt.show()
```



Can we add a **grid** to read the values more clearly?

In [21]:

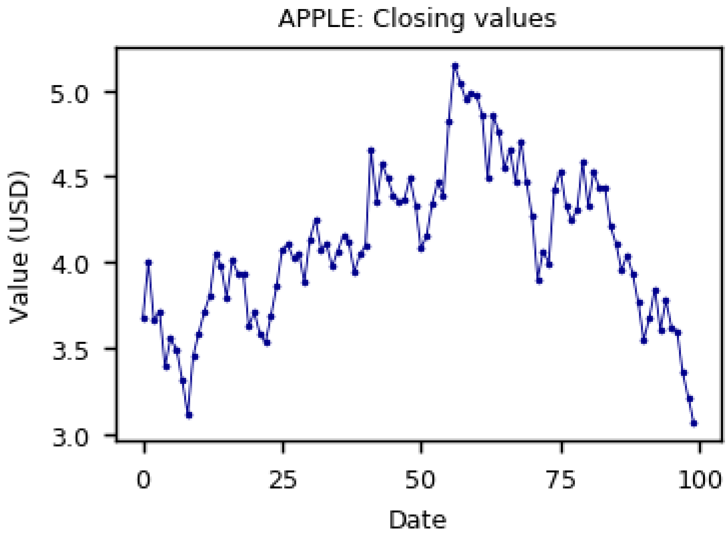
```
plt.title("APPLE: Traded volumes")

plt.xlabel("Date")

plt.ylabel("Number of shares")

n_values = len(volume)
plt.xticks(range(0, n_values, 10), dates,
           fontsize=6, rotation=75)

plt.plot(volume[0:100],
         marker = 'o', markersize=1,
         linewidth=0.5, color = 'red')

plt.grid()

plt.show()
```



What about **resizing** the plots and changing the font size of labels, etc.?

In [22]:

```
plt.figure ( figsize = (3,2))

scaled_fontsize = 7

plt.title("APPLE: Closing values", fontsize=scaled_fontsize)

plt.xlabel("Date", fontsize=scaled_fontsize)

plt.ylabel("Value (USD)", fontsize=scaled_fontsize)

plt.plot(close_val[0:100],
         marker = 'o', markersize=1,
         linewidth=0.5, color = 'darkblue')

plt.xticks(fontsize=scaled_fontsize)

plt.yticks(fontsize=scaled_fontsize)


plt.show()
```



The cells belwoe haven't been presented during the class!

They are there to let you give it a look (for the project), in any case we'll go through them during the next lecture.

Let's read another dataset, reporting **data about mall customers,** and let's use this time `csv` methods (it will be easier than before!)

In [23]:

```
import csv

f = open('Mall_Customers.csv')

f_csv = csv.reader(f)
```

In [24]:

```
fieldnames = next(f_csv)
print(fieldnames)

print(len(fieldnames))
```

```
['CustomerID', 'Gender', 'Age', 'Annual Income (k$)', 'Spending Scor
e (1-100)']
5
```

In [25]:

```
gender = []
age = []
income  = []
score = []
customer_id = []

for record in f_csv:
        customer_id.append(record[0])

        gender.append(record[1].strip()) # .strip() removes all extra white spac
es!

        age.append( int(record[2]) )
        income.append( int(record[3]) )
        score.append( int(record[4]) )
```

In [26]:

```
#age
```

In [27]:

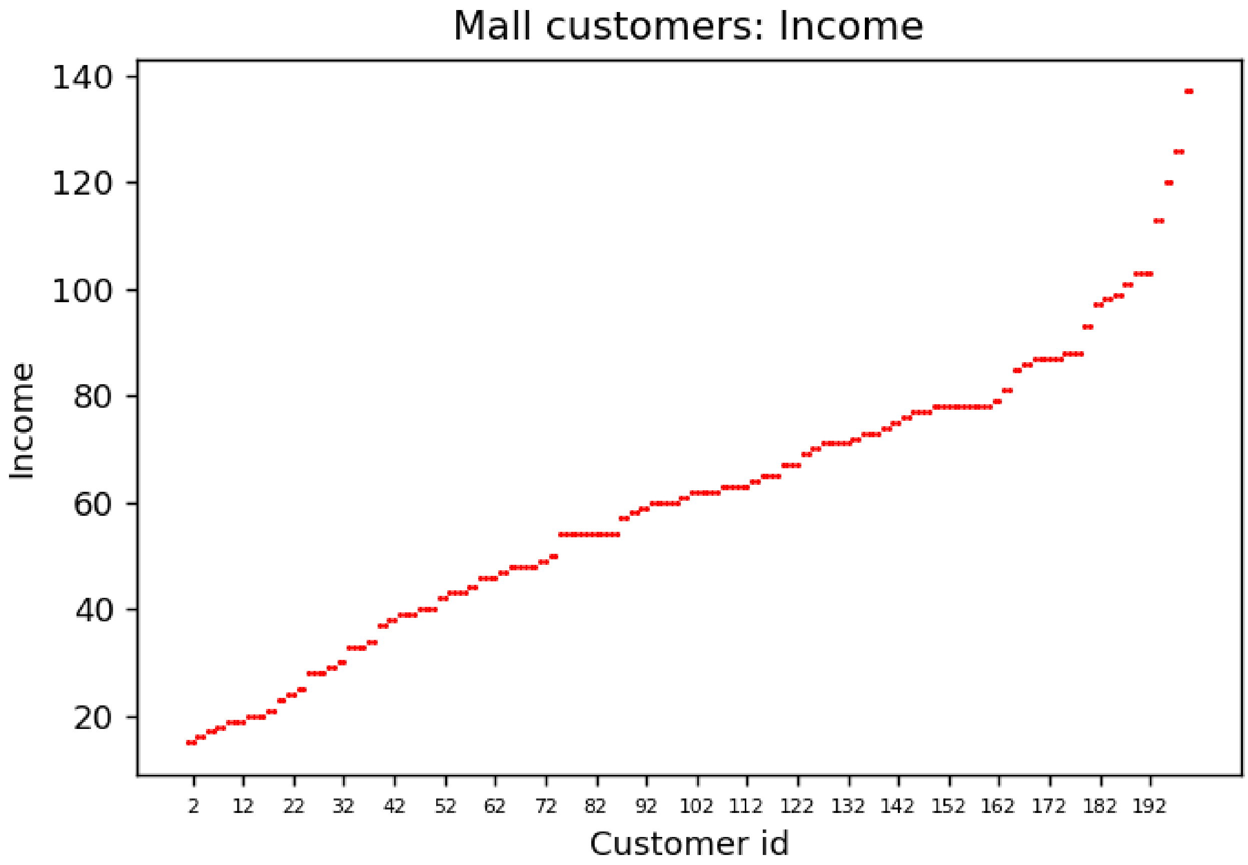
```
#gender
```

This is **not a time series** of data!

What about the distribution of the values of income?

In [28]:

```
plt.title("Mall customers: Income")

plt.xlabel("Customer id")

plt.ylabel("Income")

plt.scatter(customer_id, income, marker='o', s=0.5, color='red')

plt.xticks(customer_id[1::10], fontsize=6)

plt.show()
```

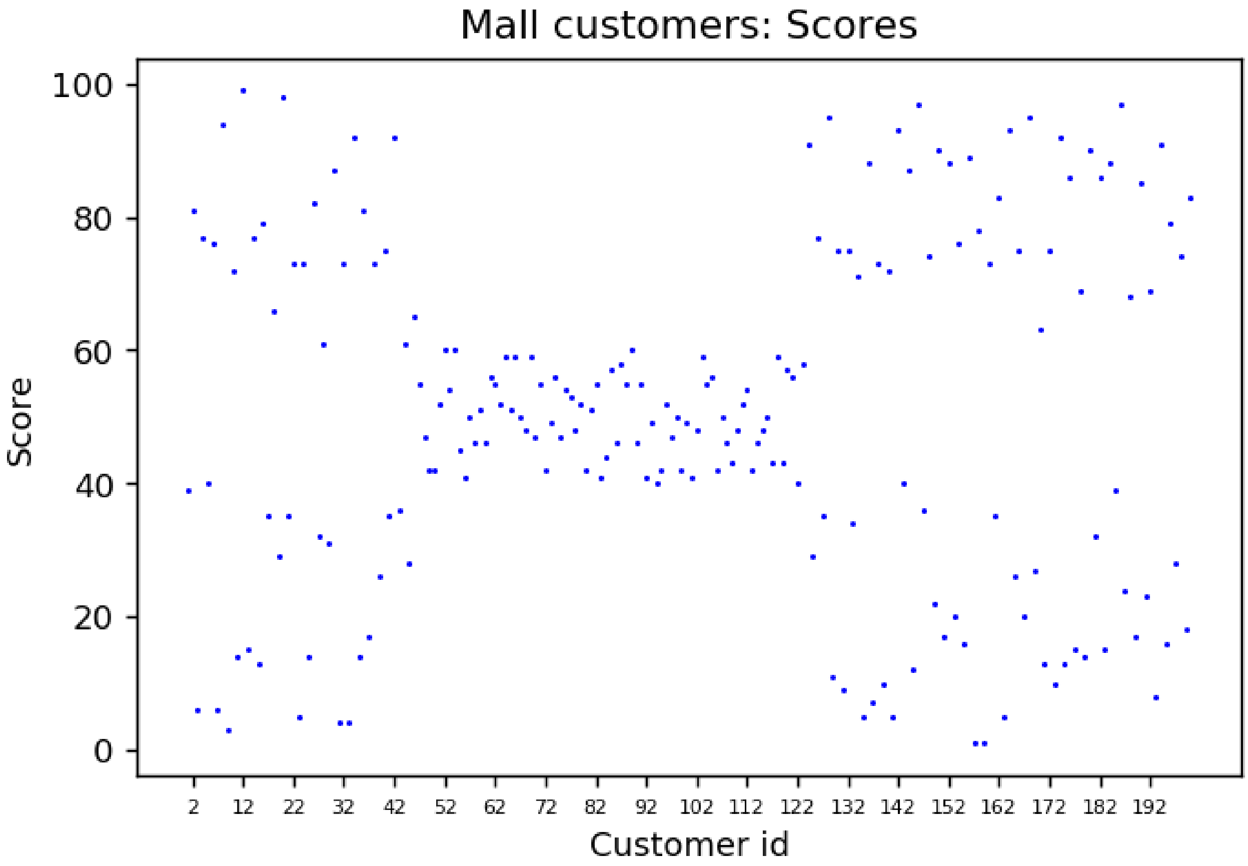


It looks like the income grows with the customer id! (values have been sorted vs. the income)

What about the shopping scores?

In [29]:

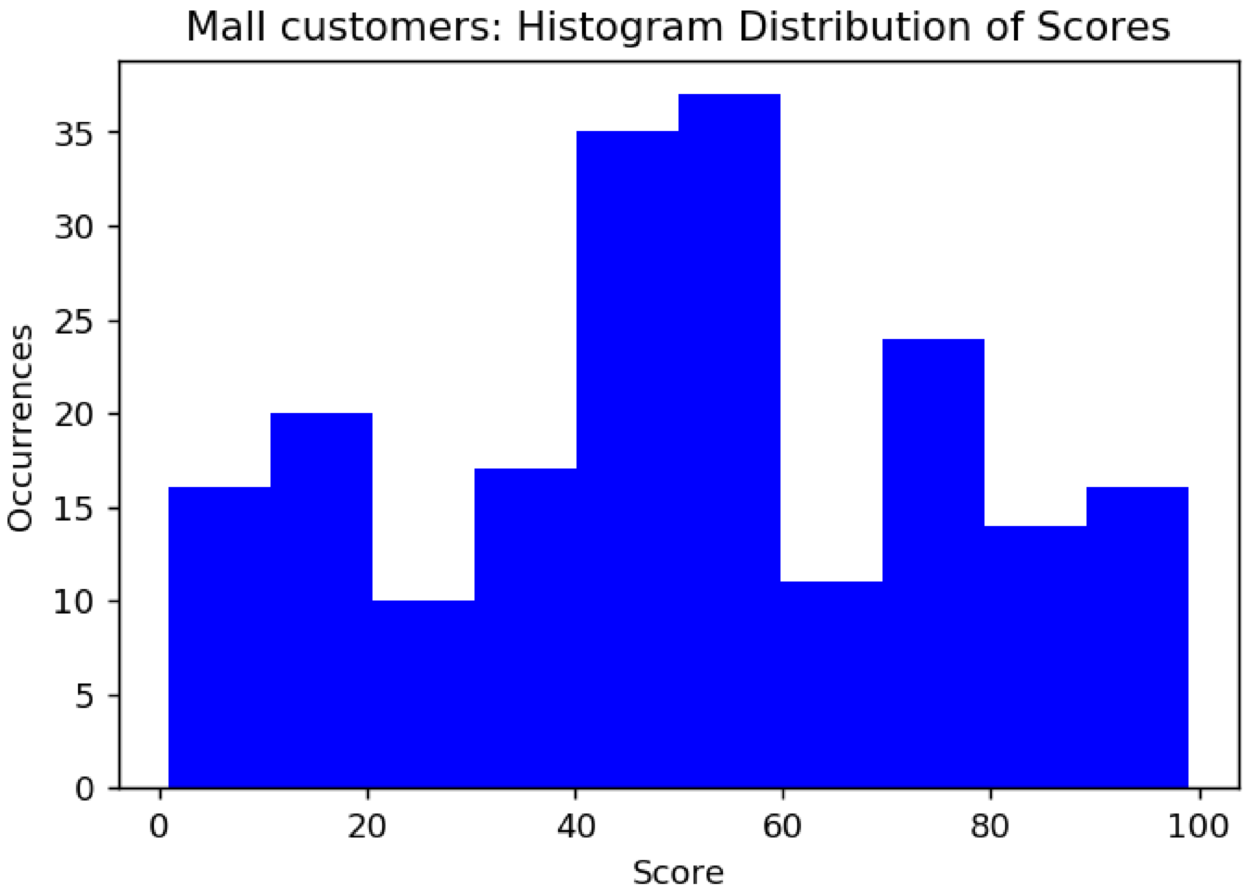
```
plt.title("Mall customers: Scores")

plt.xlabel("Customer id")

plt.ylabel("Score")

plt.scatter(customer_id, score, marker='o', s=0.5, color='blue')

plt.xticks(customer_id[1::10], fontsize=6)

plt.show()
```



Aggregating the data to check what the **distribution** of the scores look like in the customer population can be useful $\to$ **Histogram**

In [30]:

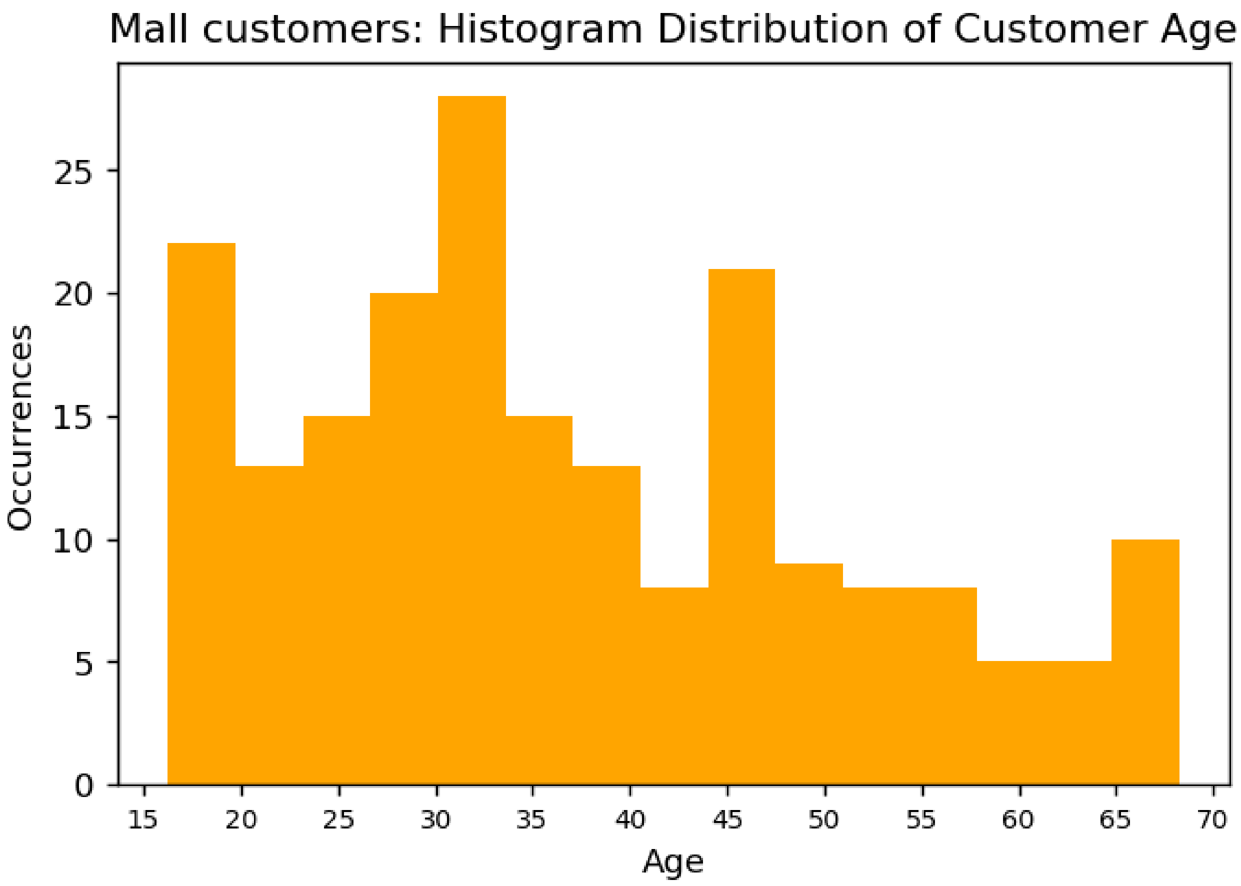
```
plt.title("Mall customers: Histogram Distribution of Scores")

plt.xlabel("Score")

plt.ylabel("Occurrences")

plt.hist(score, bins=10, color='blue')

#plt.xticks(customer_id[1::10], fontsize=6)

plt.show()
```



What about the distribution of the age of the customers?

In [31]:

```
plt.title("Mall customers: Histogram Distribution of Customer Age")

plt.xlabel("Age")

plt.ylabel("Occurrences")

plt.hist(age, bins=15, align='left',
         color='orange')

plt.xticks(range(15, 75, 5), fontsize=8)

plt.show()
```



What about the distribution of male vs. female?

In this case we have two values: number of male and number of female customers.

We need to count them from the `gender` list!

In [32]:

```
n_male = gender.count('Male')

n_female = gender.count('Female')

n_male, n_female
```

Out[32]:

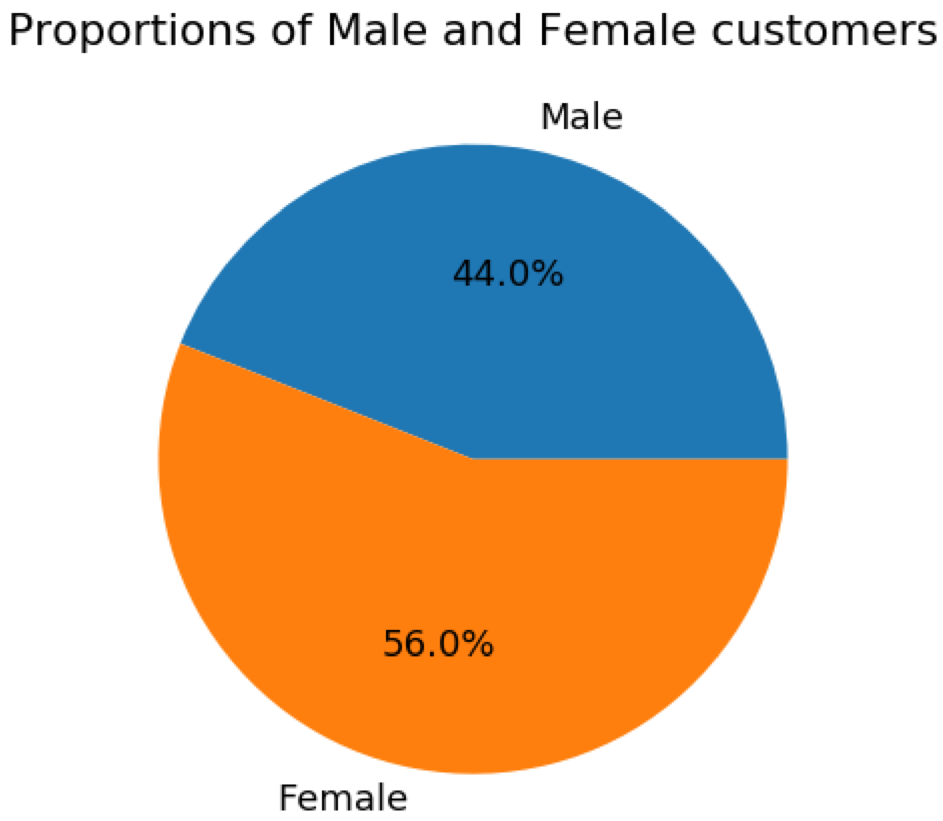
```
(88, 112)
```

How do we effectively show these proportions? (a histogram is not really appropriate for showing proportions)

- **Pie chart**
- **Bar chart**

In [33]:

```
plt.title("Proportions of Male and Female customers")

plt.pie([n_male, n_female],
        labels = ['Male', 'Female'],
        autopct="%.1f%%")  # this says to use one decimal digit and to use percentages

plt.show()
```

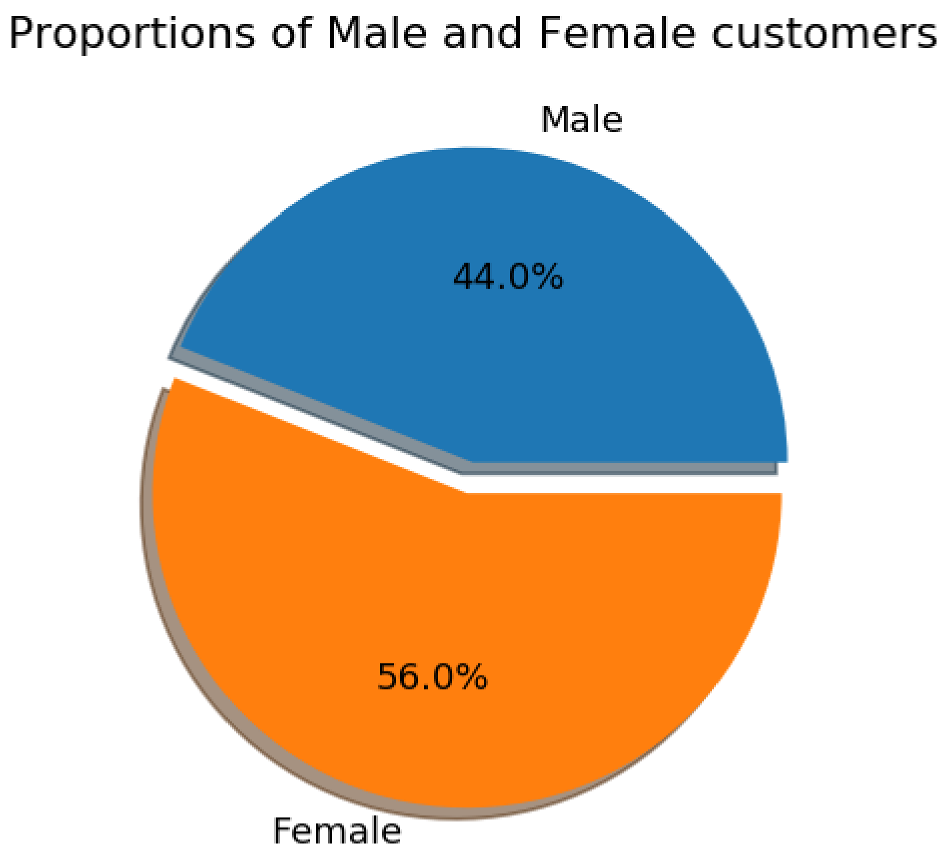


Some beautifying of the pie chart.

`shadow` adds a shadowing, while `explode` is tuple, where if the i-th value is greater than 0, it indcates that the i-th slice of the pie will be detached from the rest of the pie (at a distance proportional to the indicated value).

In [34]:

```
plt.title("Proportions of Male and Female customers")

plt.pie([n_male, n_female],
        labels = ['Male', 'Female'], shadow=True, explode = (0, 0.1),
        autopct="%.1f%%")

plt.show()
```



Can we make a pie chart for the customer age?

Yes, but we need to define age ranges, group / count the data accordingly, and store/define explanatory labels.

Let's consider 10-year ranges, starting from the minimal age up to the maximal age in the dataset.

We'll use a dictionary data structure.

In [35]:

```
import math

min_age = min(age)
max_age = max(age)

age_interval = 10

age_ranges = (max_age - min_age) / age_interval

age_ranges = math.ceil(age_ranges)
print(min_age, max_age, age_ranges)
```

```
18 70 6
```

In [36]:

```
age_dict = {}

for r in range(age_ranges):
        range_min = min_age + r * age_interval
        range_max = range_min + age_interval - 1
        range_str = str(range_min) + '-' + str(range_max)
        age_dict[ range_str ] = [range_min, range_max, 0]
```

In [37]:

```
age_dict
```

Out[37]:

```
{'18-27': [18, 27, 0],
 '28-37': [28, 37, 0],
 '38-47': [38, 47, 0],
 '48-57': [48, 57, 0],
 '58-67': [58, 67, 0],
 '68-77': [68, 77, 0]}
```

In [38]:

```
for v in age:
    for r in age_dict:
        if v >= age_dict[r][0] and v <= age_dict[r][1]:
            age_dict[r][2] += 1
            break
```

In [39]:

```
age_dict
```

Out[39]:

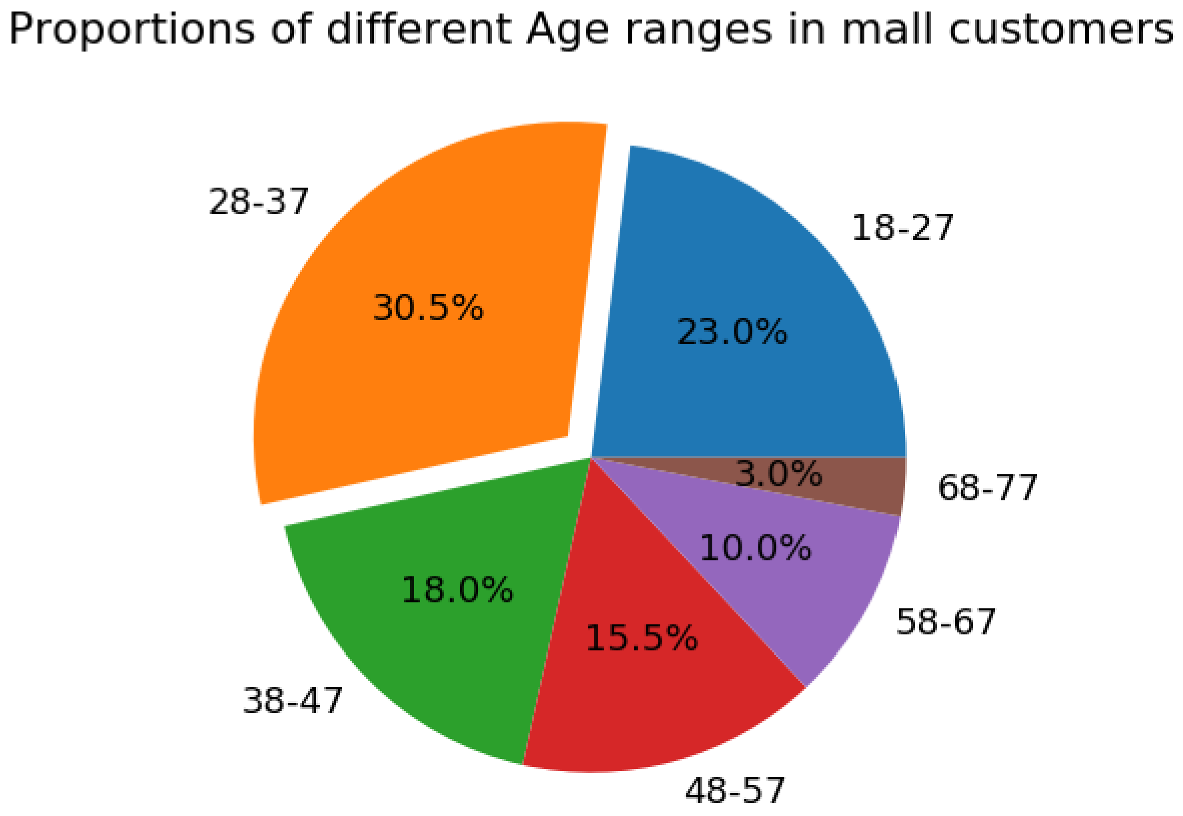
```
{'18-27': [18, 27, 46],
 '28-37': [28, 37, 61],
 '38-47': [38, 47, 36],
 '48-57': [48, 57, 31],
 '58-67': [58, 67, 20],
 '68-77': [68, 77, 6]}
```

Now we can show the counts in the selected ranges of age as proportions using a pie chart.

Counts are stored in the list `age_counts`

In [40]:

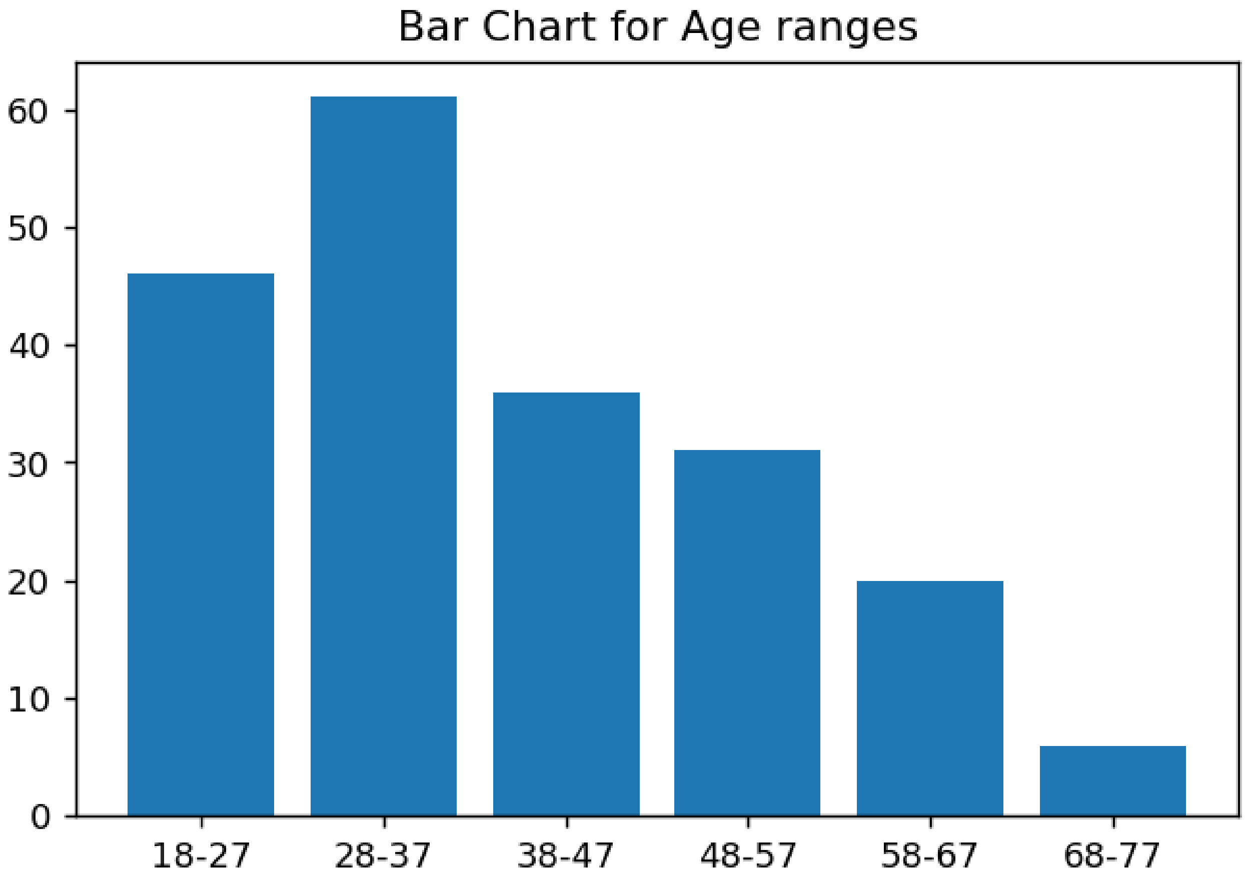
```
plt.title("Proportions of different Age ranges in mall customers")

age_counts = []
for r in age_dict.values():
    age_counts.append(r[2])

# this is to identify the slice with the largest proportion to explode it
max_range = -1
max_range_idx = -1
for i,r in enumerate(age_dict.values()):
    if r[2] > max_range:
        max_range = r[2]
        max_range_idx = i

explode_flag = [0] * len(age_dict)
explode_flag[max_range_idx] = 0.1


plt.pie(age_counts,
        labels = list(age_dict.keys()),
        #shadow=True,
        explode = explode_flag,
        autopct="%.1f%%")

plt.show()
```



We can display the same data using a **bar chart.**

In [41]:

```
plt.title("Bar Chart for Age ranges")

plt.bar(range(1, len(age_counts)+1),          # position of the bars
        age_counts,                           # value/height of the bars
        tick_label = list(age_dict.keys()))   # what to display at the x ticks

plt.show()
```



In [ ]:

```

```