Introduction to Python for Finance

INTRODUCTION TO PYTHON FOR FINANCE

Adina Howe
Instructor

datacamp

Why Python for Finance?

- Easy to Learn and Flexible
  - General purpose
  - Dynamic
  - High-level language
- Integrates with other languages
- Open source
  - Accessible to anyone

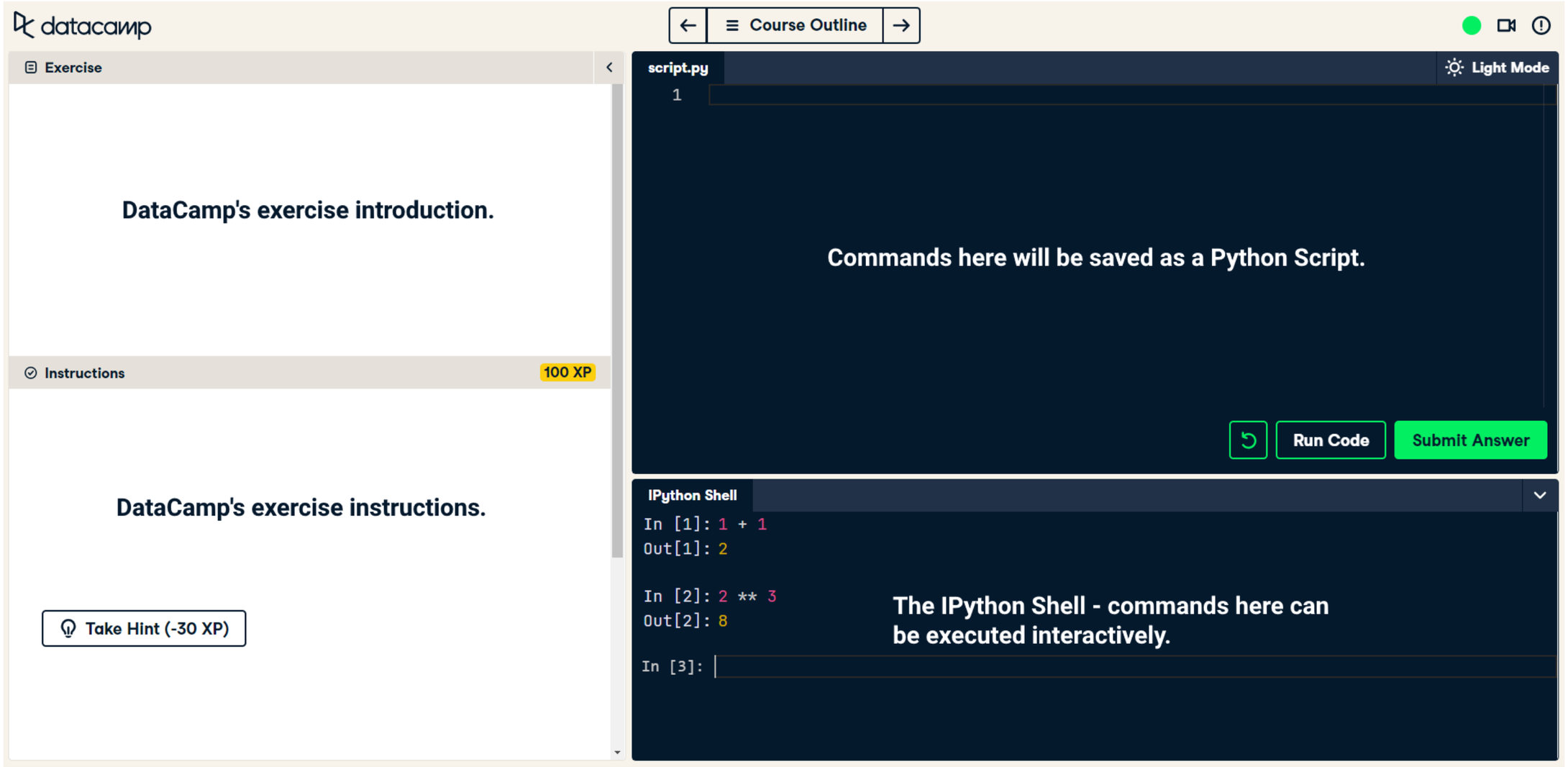
Python Shell

```
In [1]:
```

Calculations in IPython

```
In [1]: 1 + 1
```

```
2
```


datacamp
Course Outline
Exercise
DataCamp's exercise introduction.
Instructions
100 XP
DataCamp's exercise instructions.
Take Hint (-30 XP)
script.py
Light Mode
1
Commands here will be saved as a Python Script.
Run Code
Submit Answer
IPython Shell
In [1]: 1 + 1
Out[1]: 2
In [2]: 2 ** 3
Out[2]: 8
The IPython Shell - commands here can be executed interactively.
In [3]:

Common mathematical operators

Operator	Meaning
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulus (remainder of division)
**	Exponent

Common mathematical operators

```
In  [1]: 8 + 4
```

```
Out [1]: 12
```

```
In  [2]: 8 / 4
```

```
Out [2]: 2
```

Comments and variables

INTRODUCTION TO PYTHON FOR FINANCE

Name Surname
Instructor

datacamp

Any comments?

```
# Example, do not modify!
print(8 / 2 )
print(2**2)

# Put code below here
print(1.0 + 0.10)
```

Outputs in IPython vs. script.py

IPython Shell

```
In [1]: 1 + 1
```

```
Out[1]: 2
```

```
In [1]: print(1 + 1)
```

```
2
```

script.py

```
1 + 1
```

```
# No output
```

```
print(1 + 1)
```

```
<script.py> output:
    2
```

Variables

Variable names

- Names can be upper or lower case le ers, digits, and underscores
- Variables cannot start with a digit
- Some variable names are reservedin Python (e.g., class or type) and should be avoided

Variable example

```
# Correct
day_2 = 5


# Incorrect, variable name starts with a digit
2_day = 5
```

Using variables to evaluate stock trends

$$\text{Price to earning ratio} = \frac{\text{Market price}}{\text{Earnings per share}}$$

```
price = 200
earnings = 5
pe_ratio = price / earnings
print(pe_ratio)
```

```
40
```

Variable Data Types

INTRODUCTION TO PYTHON FOR FINANCE

Adina Howe
Instructor

datacamp

Python Data Types

Variable Types	Example
Strings	'hello world'
Integers	40
Floats	3.1417
Booleans	True or False

Variable Types

Variable Types	Example	Abbreviations
Strings	'Tuesday'	`str`
Integers	40	`int`
Floats	3.1417	`float`
Booleans	True or False	`bool`

What data type is a variable: type()

To identify the type, we can use the function `type()` :

```
type(variable_name)
```

```
pe_ratio = 40
print(type(pe_ratio))
```

```
<class 'int'>
```

Booleans

operators	descriptions
`==`	equal
`!=`	does not equal
`>`	greater than
`<`	less than

Boolean Example

```
print(1 == 1)
```

```
True
```

```
print(type(1 == 1))
```

```
<class 'bool'>
```

Variable manipulations

```
x = 5
print(x * 3)
```

```
15
```

```
y = 'stock'
print(y * 3)
```

```
'stockstockstock'
```

```
print(x + 3)
```

```
8
```

```
print(y + 3)
```

```
TypeError: must be str, not int
```

Changing variable types

```
pi = 3.14159
print(type(pi))
```

```
<class 'float'>
```

```
pi_string = str(pi)
print(type(pi_string))
```

```
<class 'str'>
```

```
print('I love to eat ' + pi_string + '!')
```

```
I love to eat 3.14159!
```

Lists in Python

INTRODUCTION TO PYTHON FOR FINANCE

Adina Howe
Instructor

datacamp

Lists - square brackets [ ]

```
months = ['January', 'February', 'March', 'April', 'May', 'June']
```

Python is zero-indexed

```
months = ['January', 'February', 'March', 'April', 'May', 'June']
Index:      0          1          2         3       4       5
```

Subset lists

```
months = ['January', 'February', 'March', 'April', 'May', 'June']
```

```
months[0]
```

```
'January'
```

```
months[2]
```

```
'March'
```

Negative indexing of lists

```
months = ['January', 'February', 'March', 'April', 'May', 'June']
```

```
months[-1]
```

```
'June'
```

```
months[-2]
```

```
'May'
```

Subsetting multiple list elements with slicing

Slicing syntax

```
# Includes the start and up to (but not including) the end
mylist[startAt:endBefore]
```

Example

```
months = ['January', 'February', 'March', 'April', 'May', 'June']
```

```
months[2:5]
```

```
['March', 'April', 'May']
```

```
months[-4:-1]
```

```
['March', 'April', 'May']
```

Extended slicing with lists

```
months = ['January', 'February', 'March', 'April', 'May', 'June']
```

```
months[3:]
```

```
['April', 'May', 'June']
```

```
months[:3]
```

```
['January', 'February', 'March']
```

Slicing with Steps

```
# Includes the start and up to (but not including) the end
mylist[startAt:endBefore:step]
```

```
months = ['January', 'February', 'March', 'April', 'May', 'June']
```

```
months[0:6:2]
```

```
['January', 'March', 'May']
```

```
months[0:6:3]
```

```
['January', 'April']
```

Lists in Lists

INTRODUCTION TO PYTHON FOR FINANCE

Adina Howe
Instructor

datacamp

Lists in Lists

- Lists can contain various data types, including lists themselves.
- Example: a nested list describing the month and its associated consumer price index

```
cpi = [['Jan', 'Feb', 'Mar'], [238.11, 237.81, 238.91]]
```

Subsetting Nested Lists

```
months = ['Jan', 'Feb', 'Mar']
print(months[1])
```

```
'Feb'
```

```
cpi = [['Jan', 'Feb', 'Mar'], [238.11, 237.81, 238.91]]
print(cpi[1])
```

```
[238.11, 237.81, 238.91]
```

More on Subsetting Nested Lists

How would one subset out a speci c price index?

```
cpi = [['Jan', 'Feb', 'Mar'], [238.11, 237.81, 238.91]]
print(cpi[1])
```

```
[238.11, 237.81, 238.91]
```

```
print(cpi[1][0])
```

```
238.11
```

Methods and functions

INTRODUCTION TO PYTHON FOR FINANCE

Adina Howe
Instructor

datacamp

Methods vs. Functions

Methods

- All methods are functions
- List methods are a subset of built-in functions in Python
- Used on an object
  - `prices.sort()`

Functions

- Not all functions are methods
- Requires an input of an object
  - `type(prices)`

List Methods - sort

- Lists have several built-in methods that can help retrieve and manipulate data
- Methods can be accessed as `list.method()`

`list.sort()` sorts list elements in ascending order

```
prices = [238.11, 237.81, 238.91]
prices.sort()
print(prices)
```

```
[237.81, 238.11, 238.91]
```

Adding to a list with append and extend

`list.append()` adds a single element to a list

```
months = ['January', 'February', 'March']
months.append('April')
print(months)
```

```
['January', 'February', 'March', 'April']
```

`list.extend()` adds each element to a list

```
months.extend(['May', 'June', 'July'])
print(months)
```

```
['January', 'February', 'March', 'April', 'May', 'June', 'July']
```

Useful list methods - index

`list.index(x)` returns the lowest index where the element x appears

```
months = ['January', 'February', 'March']
prices = [238.11, 237.81, 238.91]
```

```
months.index('February')
```

```
1
```

```
print(prices[1])
```

```
237.81
```

More functions ...

- `min(list)` : returns the smallest element
- `max(list)` : returns the largest element

Find the month with smallest CPI

```
months = ['January', 'February', 'March']
prices = [238.11, 237.81, 238.91]
```

```
# Identify min price
min_price = min(prices)
# Identify min price index
min_index = prices.index(min_price)
# Identify the month with min price
min_month = months[min_index]
print(min_month)
```

```
February
```

Arrays

INTRODUCTION TO PYTHON FOR FINANCE

Adina Howe
Instructor

datacamp

Installing packages

```
pip3 install package_name_here
```

```
pip3 install numpy
```

Importing packages

```
import numpy
```

NumPy and Arrays

```
import numpy
my_array = numpy.array([0, 1, 2, 3, 4])
print(my_array)
```

```
[0, 1, 2, 3, 4]
```

```
print(type(my_array))
```

```
<class 'numpy.ndarray'>
```

Using an alias

```
import package_name
package_name.function_name(...)
```

```
import numpy as np
my_array = np.array([0, 1, 2, 3, 4])
print(my_array)
```

```
[0, 1, 2, 3, 4]
```

Why use an array for financial analysis?

- Arrays can handle very large datasets e ciently
  - Computationally-memory e cient
  - Faster calculations and analysis than lists
  - Diverse functionality (many functions in Python packages)

What's the difference?

Numpy arrays

```
my_array = np.array([3, 'is', True])  True
print(my_array)
```

```
['3' 'is' 'True']
```

Lists

```
my_list = [3, 'is', True]  True
print(my_list)
```

```
[3, 'is', True]
```

Array operations

Arrays

```
import numpy as np

array_A = np.array([1, 2, 3])
array_B = np.array([4, 5, 6])

print(array_A + array_B)
```

```
[5 7 9]
```

Lists

```
list_A = [1, 2, 3]
list_B = [4, 5, 6]

print(list_A + list_B)
```

```
[1, 2, 3, 4, 5, 6]
```

Array indexing

```
import numpy as np

months_array = np.array(['Jan', 'Feb', 'March', 'Apr', 'May'])
print(months_array[3])
```

```
Apr
```

```
print(months_array[2:5])
```

```
['March' 'Apr' 'May']
```

Array slicing with steps

```
import numpy as np


months_array = np.array(['Jan', 'Feb', 'March', 'Apr', 'May'])
```

```
print(months_array[0:5:2])
```

```
['Jan' 'March' 'May']
```

Two Dimensional Arrays

INTRODUCTION TO PYTHON FOR FINANCE

Adina Howe
Instructor

datacamp

Two-dimensional arrays

```
import numpy as np

months = [1, 2, 3]
prices = [238.11, 237.81, 238.91]
```

```
cpi_array = np.array([months, prices])


print(cpi_array)
```

```
[[  1.    2.    3.  ]
 [238.11 237.81 238.91]]
```

Array Methods

```
print(cpi_array)
```

```
[[   1.     2.     3.  ]
 [ 238.11  237.81  238.91]]
```

`.shape` gives you dimensions of the array

```
print(cpi_array.shape)
```

```
(2, 3)
```

`.size` gives you total number of elements in the array

```
print(cpi_array.size)
```

```
6
```

Array Functions

```
import numpy as np

prices = [238.11, 237.81, 238.91]
prices_array = np.array(prices)
```

`np.mean()` calculates the mean of an input

```
print(np.mean(prices_array))
```

```
238.27666666666667
```

`np.std()` calculates the standard deviation of an input

```
print(np.std(prices_array))
```

```
0.46427960923946671
```

The `arange()` function

`numpy.arange()` creates an array with start, end, step

```
import numpy as np

months = np.arange(1, 13)
print(months)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

```
months_odd = np.arange(1, 13, 2)
print(months_odd)
```

```
[ 1  3  5  7  9 11]
```

The `transpose()` function

`numpy.transpose()` switches rows and columns of a numpy array

```
print(cpi_array)
```

```
[[   1.     2.     3.  ]
 [ 238.11  237.81  238.91]]
```

```
cpi_transposed = np.transpose(cpi_array)
```

```
print(cpi_transposed)
```

```
[[   1.    238.11]
 [   2.    237.81]
 [   3.    238.91]]
```

Array Indexing for 2D arrays

```
print(cpi_array)
```

```
[[   1.     2.     3.  ]
 [ 238.11  237.81  238.91]]
```

```
# row index 1, column index 2
cpi_array[1, 2]
```

```
238.91
```

```
# all row slice, third column
print(cpi_array[:, 2])
```

```
[   3.    238.91]
```

Using Arrays for Analyses

INTRODUCTION TO PYTHON FOR FINANCE

Adina Howe
Instructor

datacamp

Indexing Arrays

```
import numpy as np


months_array = np.array(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'])
indexing_array = np.array([1, 3, 5])
```

```
months_subset = months_array[indexing_array]
print(months_subset)
```

```
['Feb' 'Apr' 'Jun']
```

More on indexing arrays

```
import numpy as np

months_array = np.array(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'])

negative_index = np.array([-1, -2])

print(months_array[negative_index])
```

```
['Jun' 'May']
```

Boolean arrays

```
import numpy as np

months_array = np.array(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'])

boolean_array = np.array([True, True, True, False, False, False])
print(months_array[boolean_array])
```

```
['Jan' 'Feb' 'Mar']
```

More on Boolean arrays

```
prices_array = np.array([238.11, 237.81, 238.91])
# Create a Boolean array
boolean_array = (prices_array > 238)


print(boolean_array)
```

```
[ True False  True]
```

```
print(prices_array[boolean_array])
```

```
[ 238.11  238.91]
```

Visualization in Python

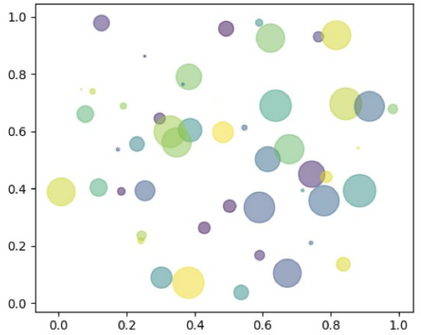
INTRODUCTION TO PYTHON FOR FINANCE

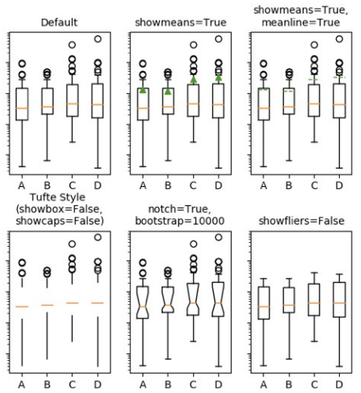
Adina Howe
Instructor

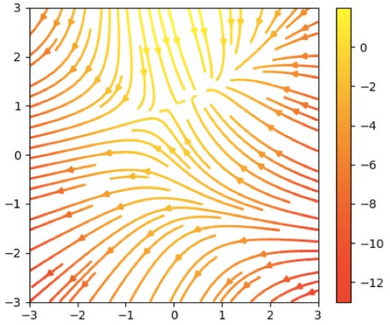
datacamp

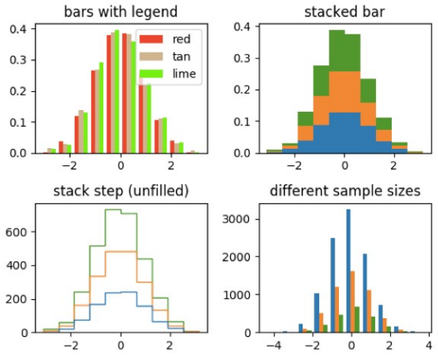
Matplotlib: A visualization package

See more of the Matplotlib gallery by clicking this link.









matplotlib.pyplot - diverse plotting functions

```
import matplotlib.pyplot as plt
```

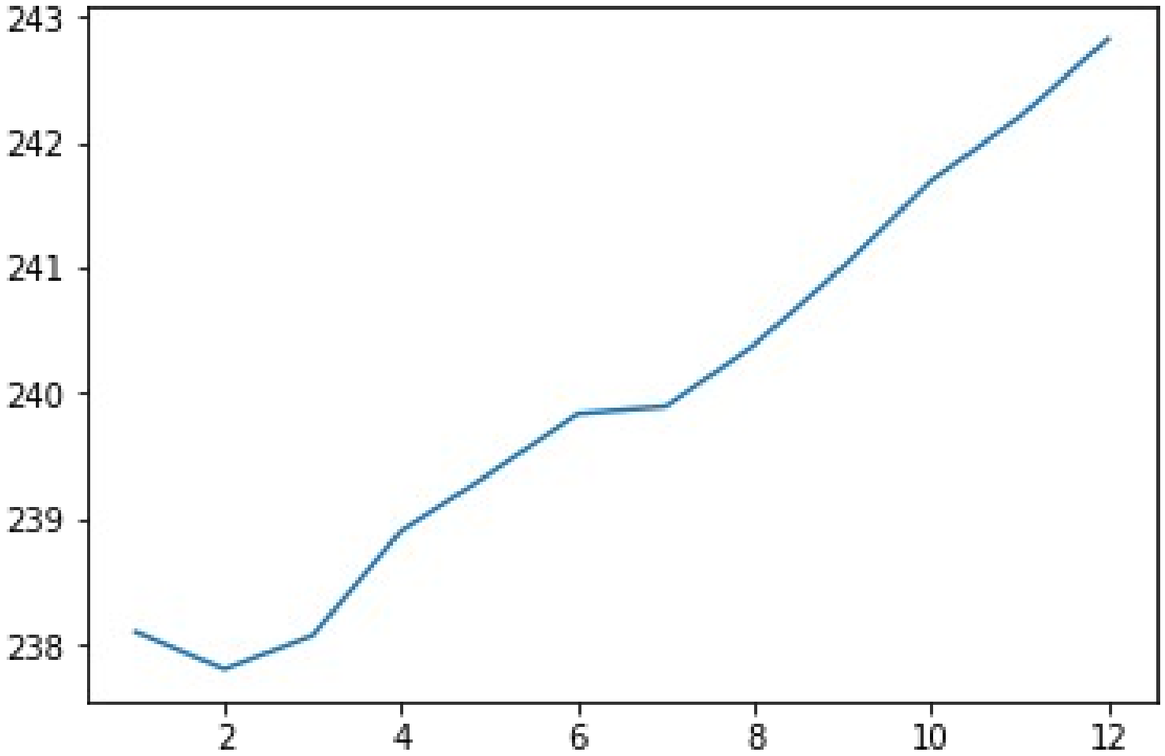
matplotlib.pyplot - diverse plotting functions

- `plt.plot()`
  - takes arguments that describe the data to be plo ed
- `plt.show()`
  - displays plot to screen

Plotting with pyplot

```
import matplotlib.pyplot as plt
plt.plot(months, prices)
plt.show()
```

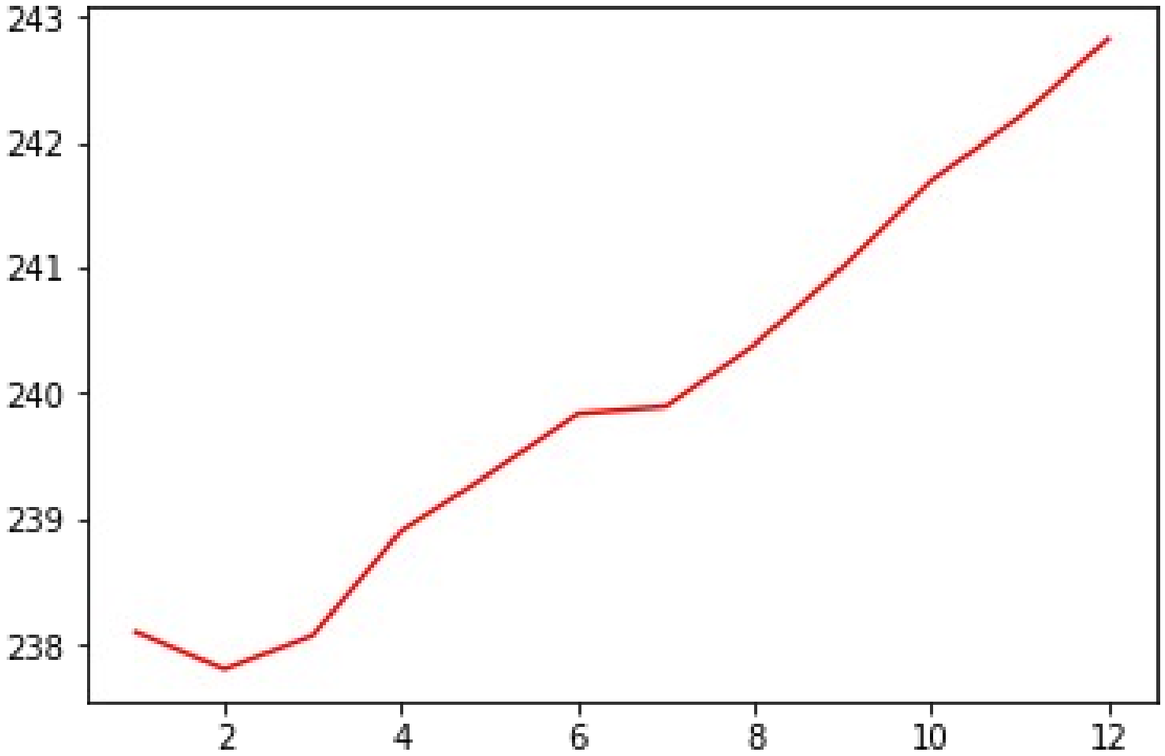
Plot result



Red solid line

```
import matplotlib.pyplot as plt
plt.plot(months, prices, color = 'red')
plt.show()
```

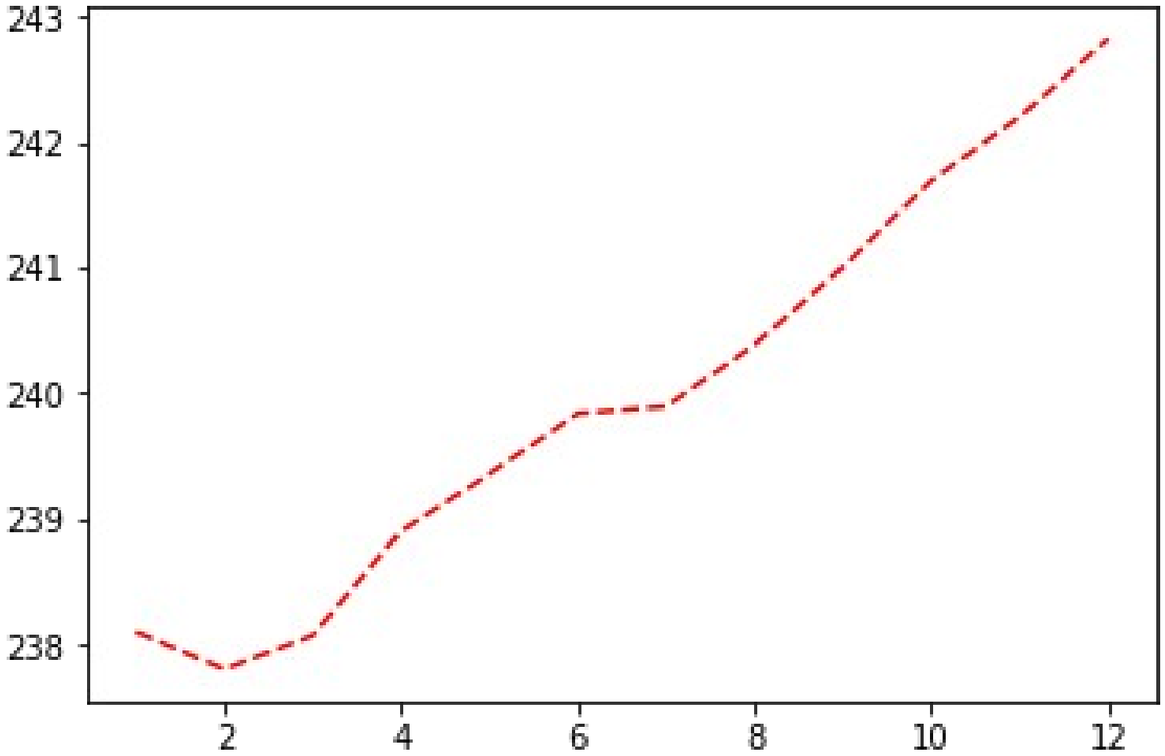
Plot result



Dashed line

```
import matplotlib.pyplot as plt
plt.plot(months, prices, color = 'red', linestyle = '--')
plt.show()
```

Plot result



Colors and linestyles

	color
'green'	green
'red'	red
'cyan'	cyan
'blue'	blue

	linestyle
'-'	solid line
'--'	dashed line
'-.'	dashed dot line
':'	do ed

More documentation on colors and lines can be found here.

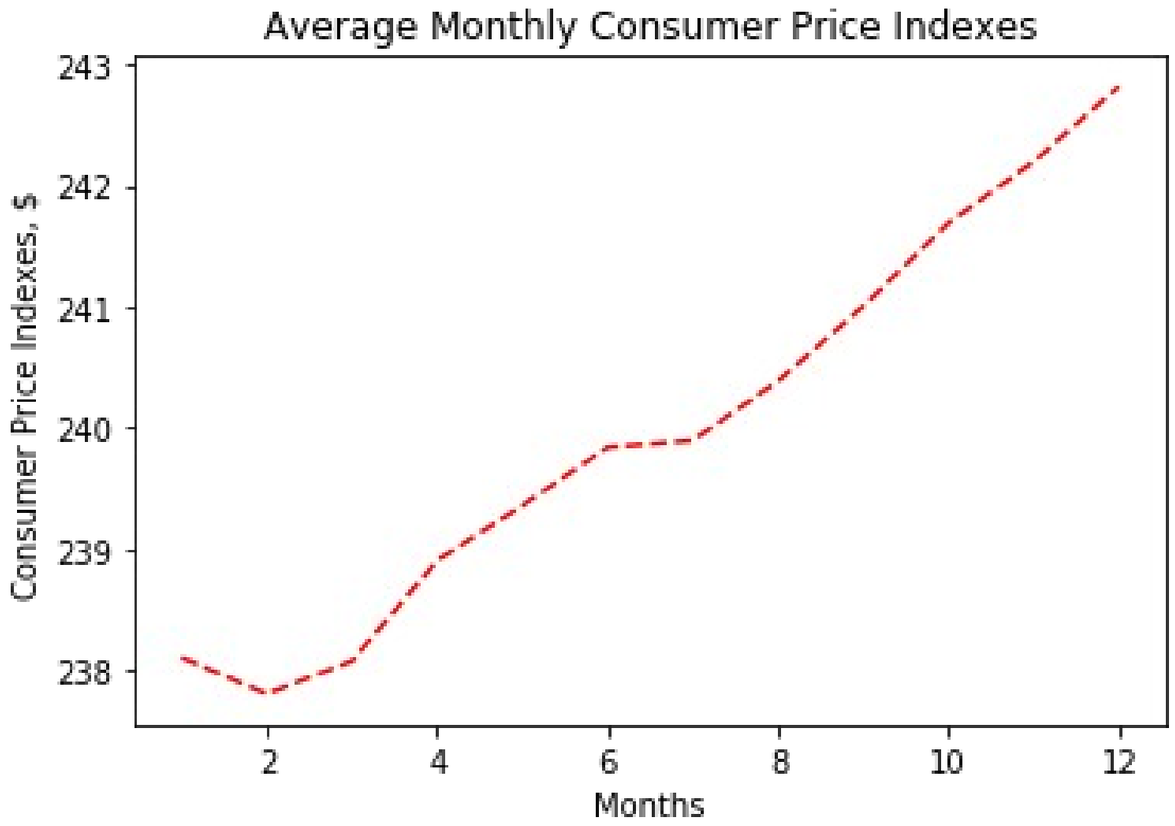
Adding Labels and Titles

```
import matplotlib.pyplot as plt
plt.plot(months, prices, color = 'red', linestyle = '--')

# Add labels
plt.xlabel('Months')
plt.ylabel('Consumer Price Indexes, $')
plt.title('Average Monthly Consumer Price Indexes')

# Show plot
plt.show()
```

Plot result


Average Monthly Consumer Price Indexes
Consumer Price Indexes, $
243
242
241
240
239
238
2
4
6
8
10
12
Months

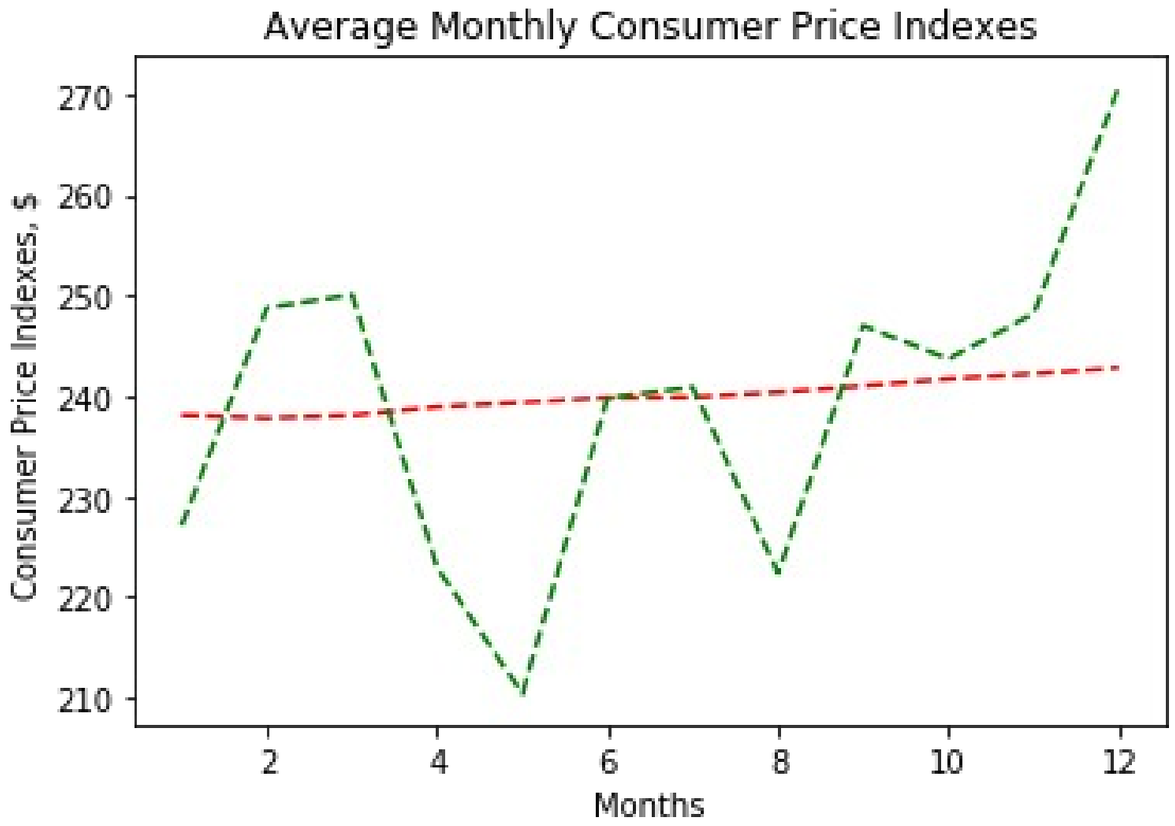
Adding additional lines

```
import matplotlib.pyplot as plt
plt.plot(months, prices, color = 'red', linestyle = '--')

# adding an additional line
plt.plot(months, prices_new, color = 'green', linestyle = '--')

plt.xlabel('Months')
plt.ylabel('Consumer Price Indexes, $')
plt.title('Average Monthly Consumer Price Indexes')
plt.show()
```

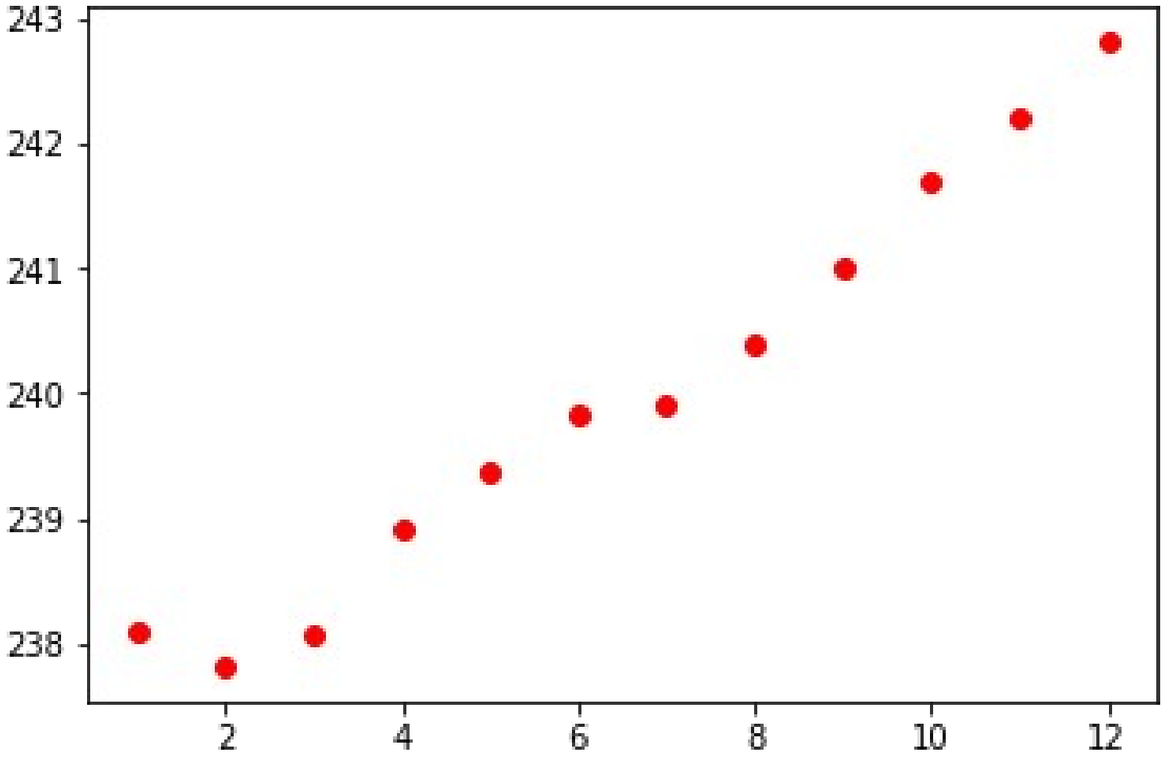
Plot result


Average Monthly Consumer Price Indexes
Consumer Price Indexes, $
270
260
250
240
230
220
210
2
4
6
8
10
12
Months

Scatterplots

```
import matplotlib.pyplot as plt
plt.scatter(x = months, y = prices, color = 'red')
plt.show()
```

Scatterplot result



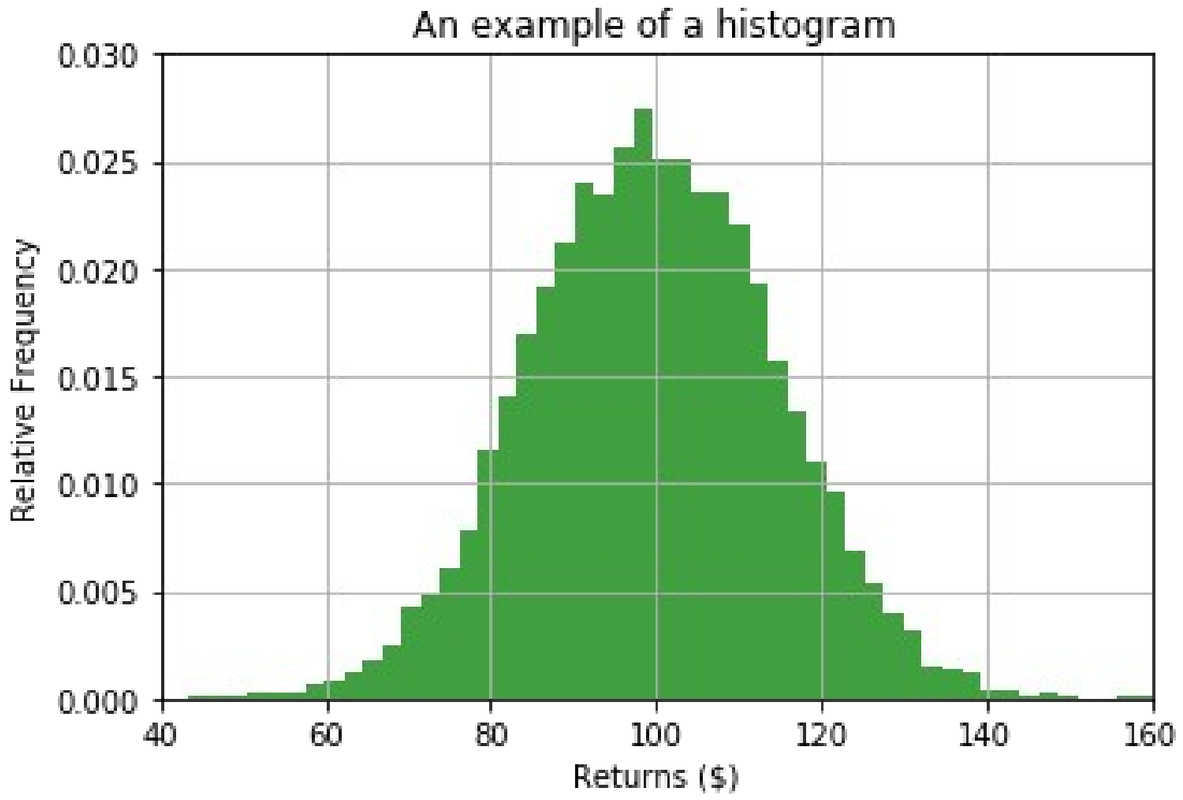
Histograms

INTRODUCTION TO PYTHON FOR FINANCE

Adina Howe
Instructor

datacamp

Why histograms for financial analysis?

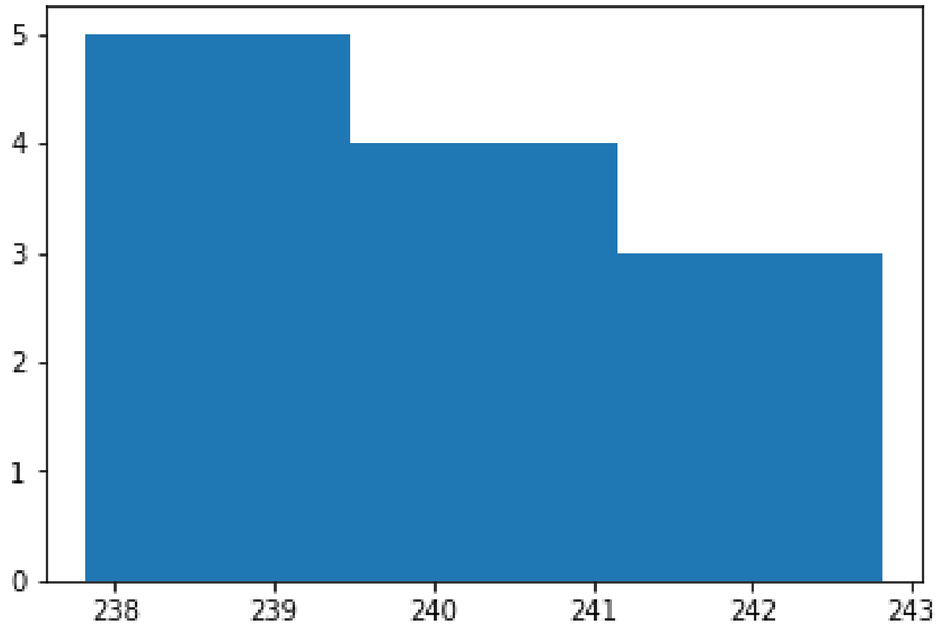


Histograms and Data

- Is your data skewed?
- Is your data centered around the average?
- Do you have any abnormal data points (outliers) in your data?

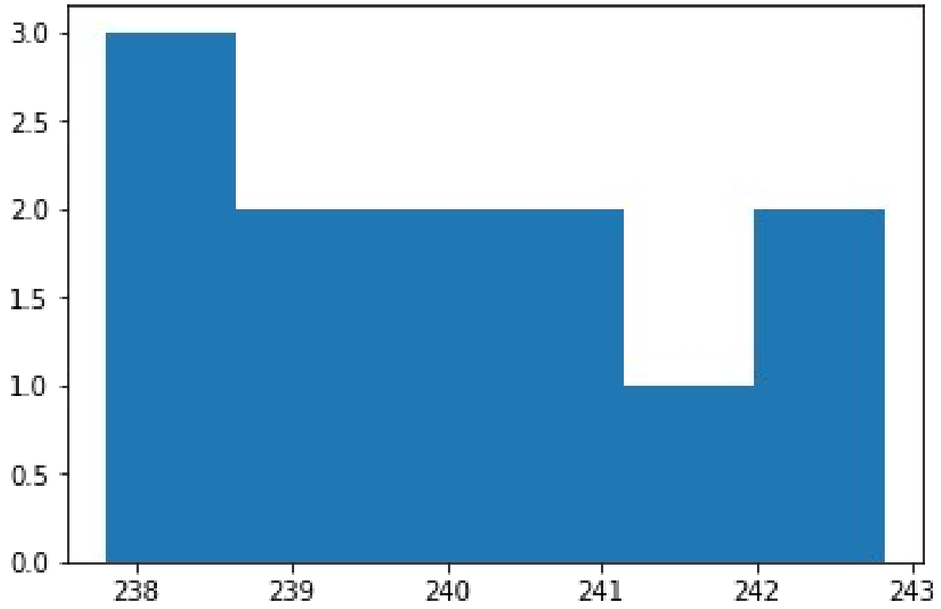
Histograms and matplotlib.pyplot

```
import matplotlib.pyplot as plt
plt.hist(x=prices, bins=3)
plt.show()
```



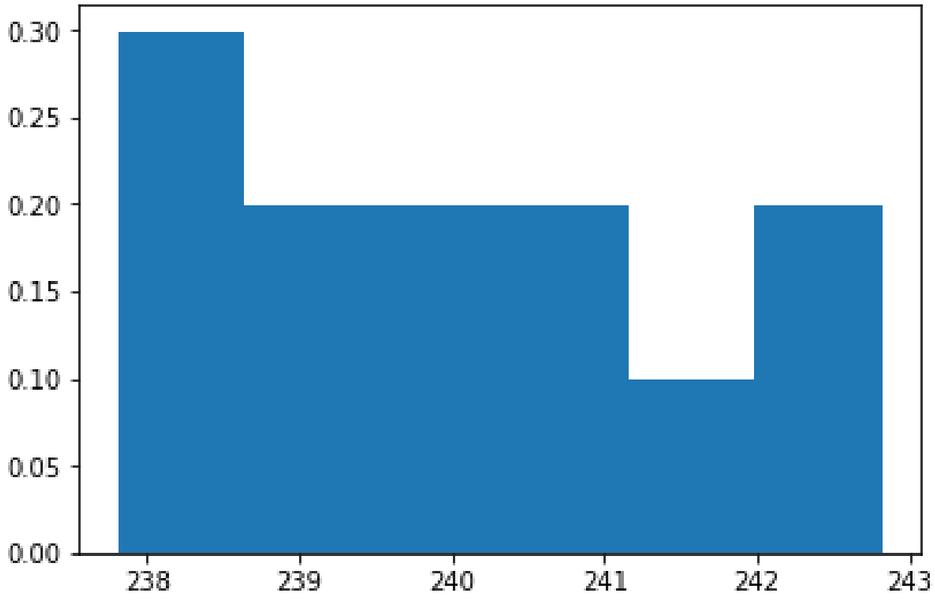
Changing the number of bins

```
import matplotlib.pyplot as plt
plt.hist(prices, bins=6)
plt.show()
```



Normalizing histogram data

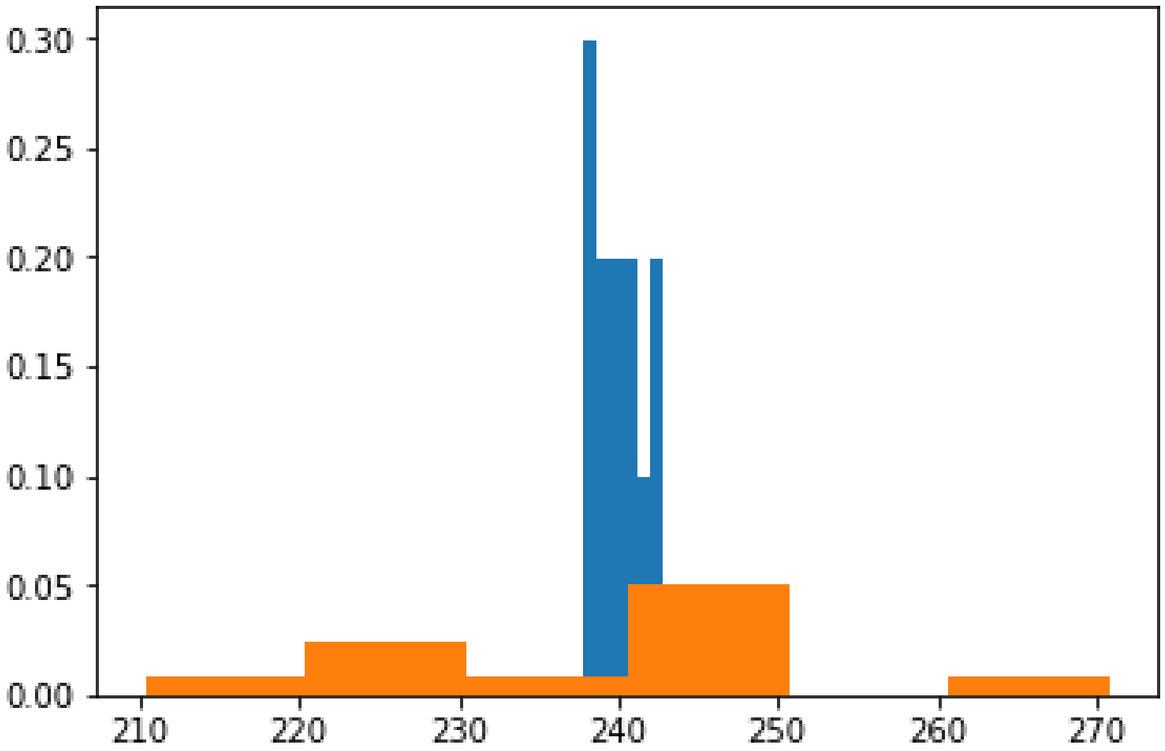
```
import matplotlib.pyplot as plt
plt.hist(prices, bins=6, normed=1)
plt.show()
```



Layering histograms on a plot

```
import matplotlib.pyplot as plt
plt.hist(x=prices, bins=6, normed=1)
plt.hist(x=prices_new, bins=6, normed=1)
plt.show()
```

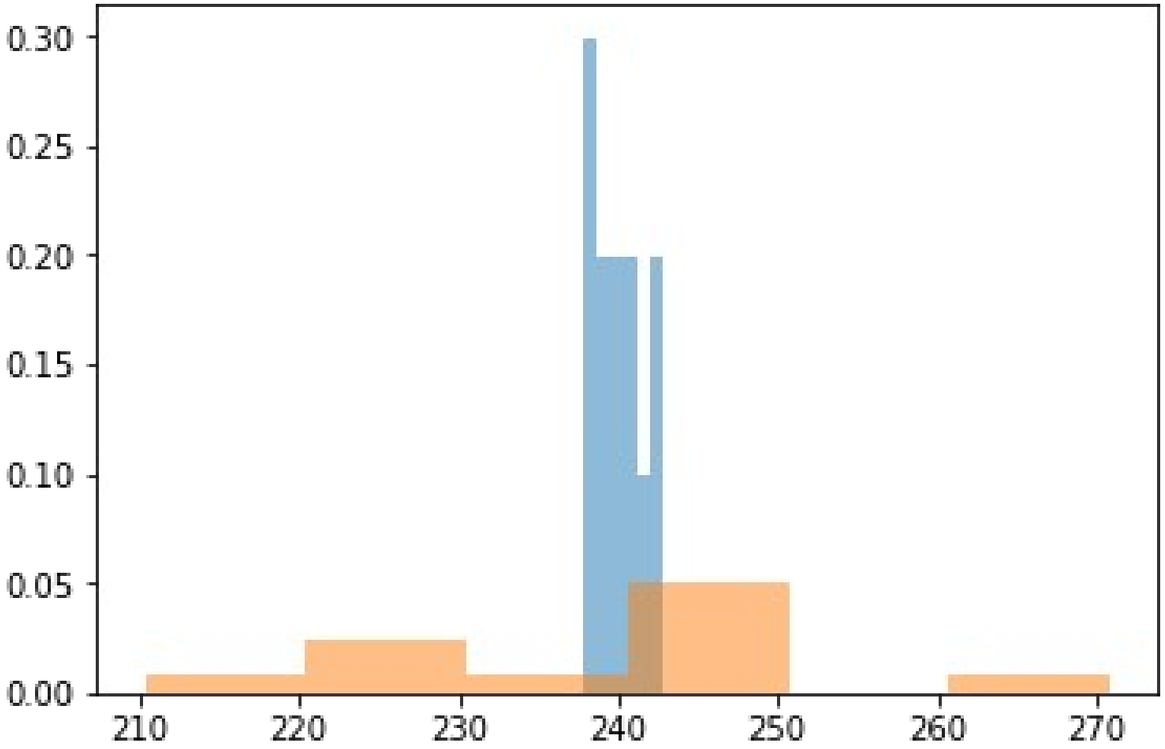
Histogram result



Alpha: Changing transparency of histograms

```
import matplotlib.pyplot as plt
plt.hist(x=prices, bins=6, normed=1, alpha=0.5)
plt.hist(x=prices_new, bins=6, normed=1, alpha=0.5)
plt.show()
```

Histogram result



Adding a legend

```
import matplotlib.pyplot as plt
plt.hist(x=prices, bins=6, normed=1, alpha=0.5, label="Prices 1")
plt.hist(x=prices_new, bins=6, normed=1, alpha=0.5, label="Prices New")
plt.legend()
plt.show()
```

Histogram result