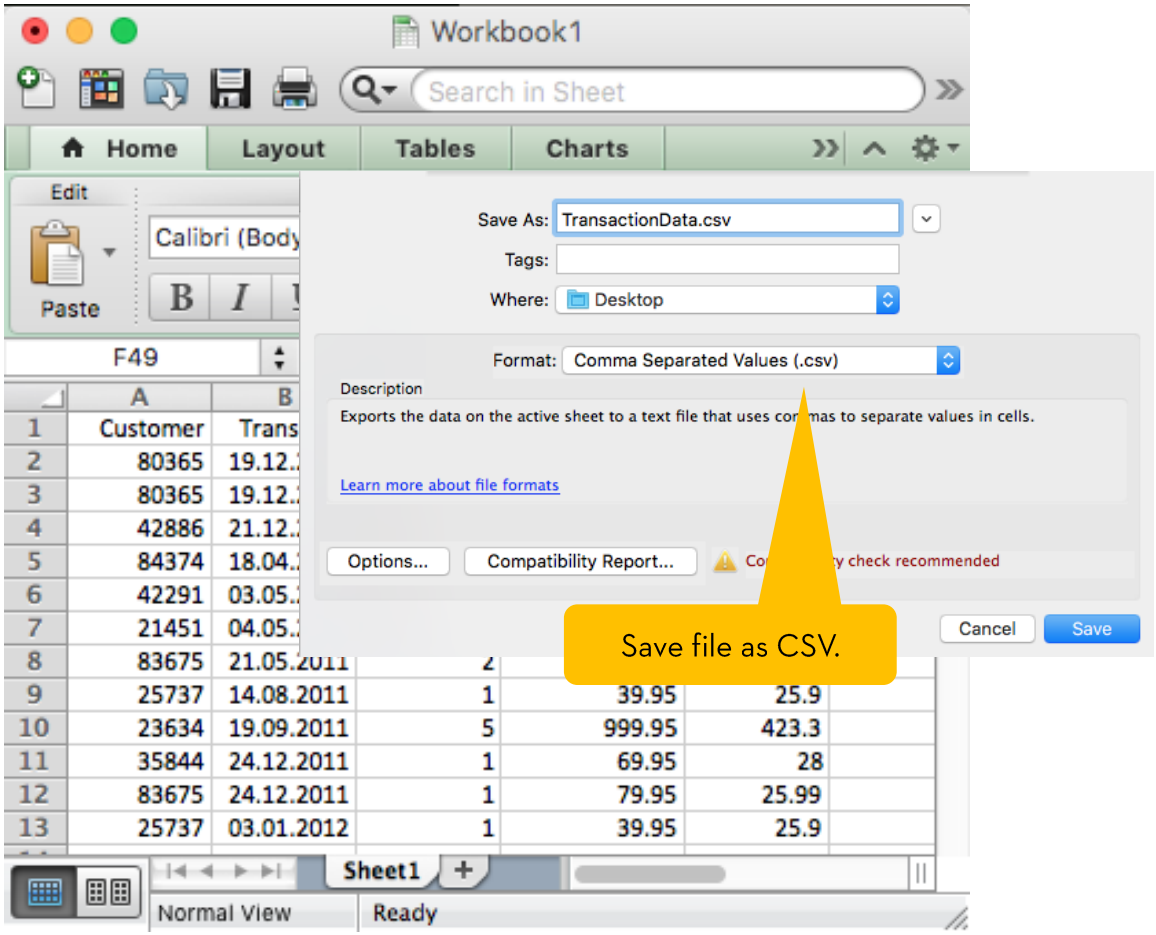
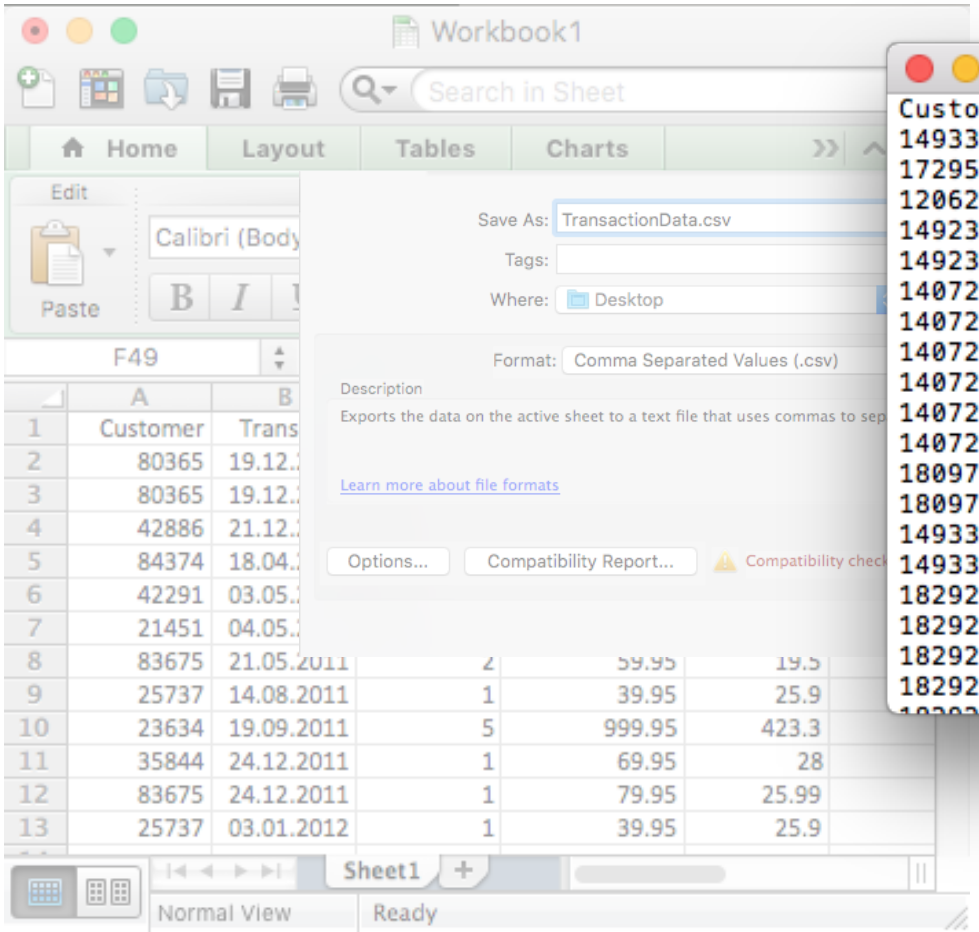
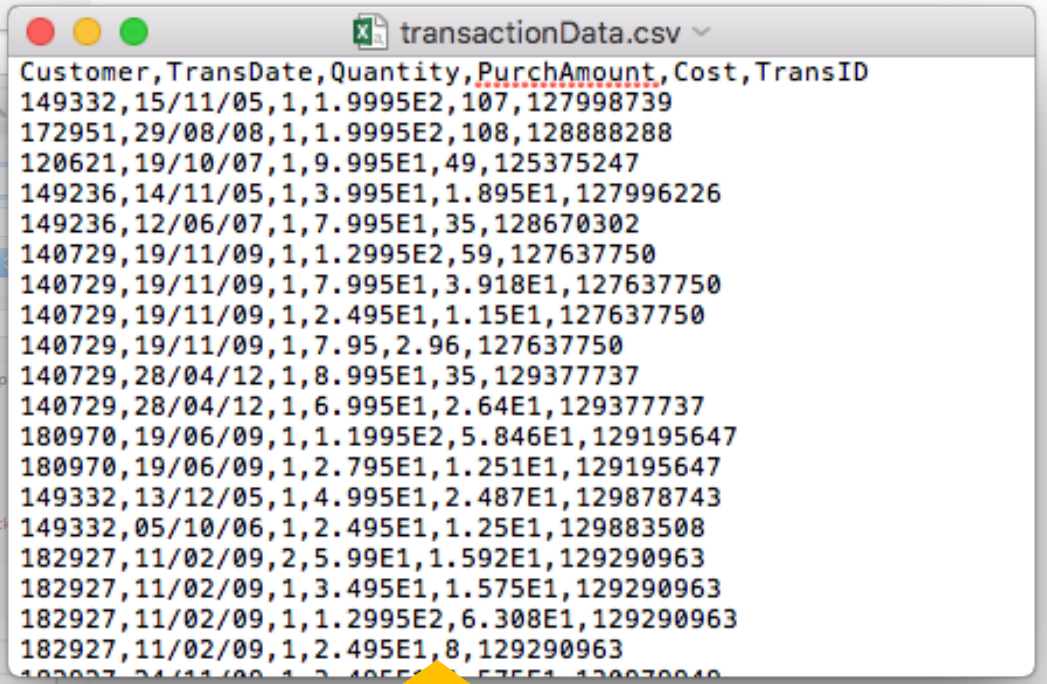
Read and write data

CSV Step 1:
Save spreadsheet as CSV



CSV Step 1:
Save spreadsheet as CSV





Values are separated by commas (but you also may see semicolons ";")

CSV Step 2:
How to load data into Python from a CSV

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15/11/05	1	199.95	107.00	127998739
172951	29/08/08	1	199.95	108.00	128888288
120621	19/10/07	1	99.95	49.00	125375247
149236	14/11/05	1	39.95	18.95	127996226
149236	12/06/07	1	79.95	35.00	128670302
...	...	...	...	...	...

```
pd.read_csv(input, sep=",", ...)
```

We have already imported the `pandas` library.

Name of CSV file.

Python Basics: A function performs a specific action and is controlled through parameters

Function reads in a CSV file.

`sep` specifies the character used to separate the values in the CSV.

```
pd.read_csv( filepath_or_buffer, sep=",",
             header="infer", names=None, ...)
```

`filepath_or_buffer` does not have a **default.**

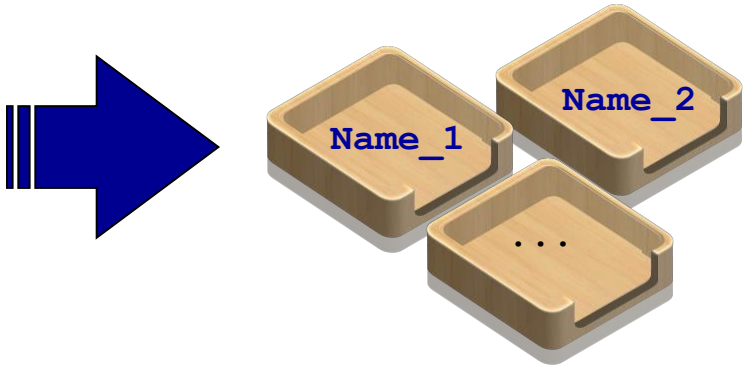
`sep` has a **default.**

Differentiate two types of function arguments:

- No default: if parameter is not specified, error is returned.
- Default: if parameter is not specified, default is used.

CSV Step 3:
Make data available for use

Customer	TransDate	Quantity	Purch Amount	Cost	TransID
149332	15/11/05	1	199.95	107.00	127998739
172951	29/08/08	1	199.95	108.00	128888288
120621	19/10/07	1	99.95	49.00	125375247
149236	14/11/05	1	39.95	18.95	127996226
149236	12/06/07	1	79.95	35.00	128670302
...	...	...	...	...	...



Name of new variable.

Use equal sign to store object to **variableName**.

```
variableName = (...)
```

Python Basics: Specifying your working directory

- To find the current working directory use:

```
os.getcwd()
```

The "os"- module is responsible for Mac **and** Windows. Review lecture 1 to see how to load a module.

- To set a new working directory use:

Version 1 - Mac:

```
os.chdir("~/path/to/my/
          directory/")
```

Plug in your intended path here.

Version 2 - Windows:

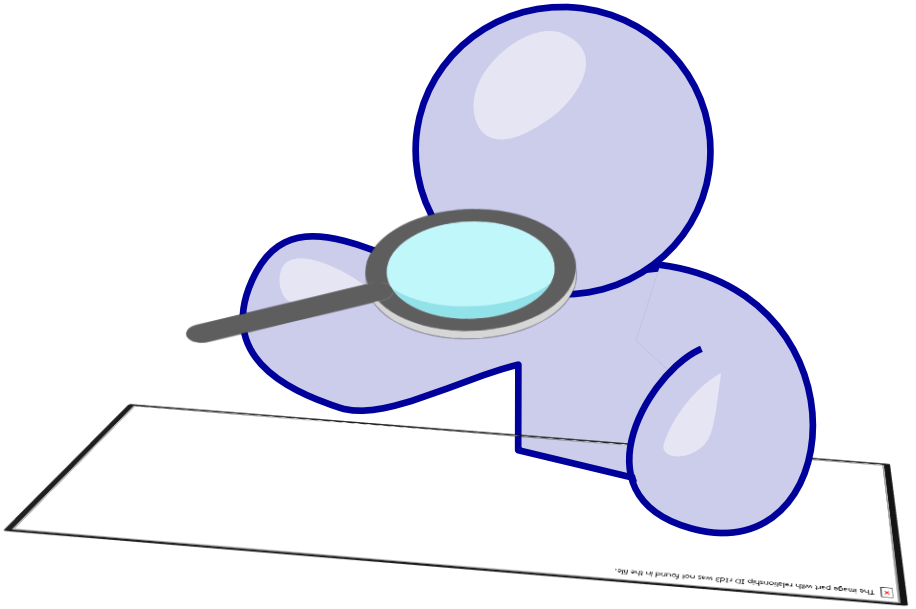
```
os.chdir("C:\Users\Desktop\
          MyR-Folder")
```

Basic investigation of your data

Observe and explore your data
3 options to make sure the data loaded correctly

Many mistakes can be made when loading data. Here is a good place to start:

1. Look at the data.
2. Look at the individual variables.
3. Look at summary statistics.



1) Look at your data

	Customer	TransDate	Quantity	PurchAmount	Cost	TransID
1	149332	15.11.2005	1	199.95	107.00	27998739
2	172951	29.08.2008	1	199.95	108.00	128888288
3	120621	19.10.2007	1	99.95	49.00	125375247
4	149236	14.11.2005	1	39.95	18.95	127996226
5	149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...	...
223187	199997	17.09.2012	1	29.95	13.80	132481149
223188	199997	17.09.2012	1	29.95	13.80	132481149
223189	199998	17.09.2012	1	29.95	13.80	132481154
223190	199999	17.09.2012	1	179.95	109.99	132481165
223191	199542	17.09.2012	1	39.95	10.50	131973368

[223191 rows x 5 columns]

myData

1) Look at your data

	Customer	TransDate	Quantity	PurchAmount	Cost	TransID
1	149332	15.11.2005	1	199.95	107.00	27998739
2	172951	29.08.2008	1	199.95	108.00	128888288
3	120621	19.10.2007	1	99.95	49.00	125375247

Look at the first observations with the head()-function

```
myData.head(n=3)
```

	Customer	TransDate	Quantity	PurchAmount	Cost	TransID
223189	199998	17.09.2012	1	29.95	13.80	132481154
223190	199999	17.09.2012	1	179.95	109.99	132481165
223191	199542	17.09.2012	1	39.95	10.50	131973368

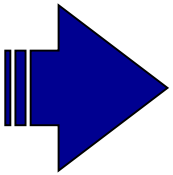
Do the same for the last observations with the tail()-function:

```
myData.tail(n=3)
```

Checking your data
2) Look at individual variables

Before processing any data, you always have to ensure, that your data is formatted properly and that the right data types are assigned to your variables. This will save a lot of time and you can avoid common mistakes.

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15/11/05	1	199.95	107.00	127998739
172951	29/08/08	1	199.95	108.00	128888288
120621	19/10/07	1	99.95	49.00	125375247
149236	14/11/05	1	39.95	18.95	127996226
149236	12/06/07	1	79.95	35.00	128670302
...	...	...	...	...	...



```
myDataCustomer          int64
TransDate               object
Quantity                int64
PurchAmount             float64
Cost                    float64.
dtype: object
```

```
mydata.dtypes
```

Check if the type of the variables is correct.

Sidenote: Built-in Data types in Python

Python distinguishes between several data types - the most common are:

Data type		Description	Sign	Example
Logical		Variable is a logical value which can either be *True* or *False*.	bool	*True, False*
Numeric	**integer**	Variable is a number which can be written without a fractional component (whole-number).	int	*-3, 0, 1, 2, 3,...*
	float	Variable is a computational approximation of any real-valued number.	float	*-2.6, 1.0, 1.1, 1.329*
Text	**string**	Variable is interpreted as "text".	str	*"a", "Z", "Hello", "Anna"*
Categorial	**pandas. categorial**	Variable, which can take on only a limited and usually fixed, number of possible values on a nominal scale.	category	*pd.Series(["a","b","c"], dtype="category")*
Dates and time	**datetime**	Variable is a data or time and special functionalities for manipulation are provided.	date / time	*d = datetime.datetime (2009, 10, 5)*

Sidenote: Module “datetime”

A **module** is a single file which can be imported in Python. A **package** refers to a collection of modules.

If working with dates and times, many mistakes occur when dates and times are not identified and/or formatted properly (especially wrt international settings).

”with respect to”

The **“datetime”** module makes it easier to work with dates and times:

- Identify and parse time,
- Extract and modify years, months, days, hours, ...
- Perform accurate math with date-times.

Format the data

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15/11/05	1	199.95	107.00	127998739
172951	29/08/08	1	199.95	108.00	128888288
120621	19/10/07	1	99.95	49.00	125375247
149236	14/11/05	1	39.95	18.95	127996226
149236	12/06/07	1	79.95	35.00	128670302
...	...	...	...	...	...

Object.

Customer	TransDate	...
149332	2005-11-15	...
172951	2008-08-29	...
120621	2007-10-19	...
149236	2005-11-14	...
149236	2007-06-12	...
...	...	...

Recognized as date.

Pandas object to modify.

Column to modify.

```
myData["TransDate"]=
        pd.to_datetime(mydata["TransDate"],
                       format="%d.%m.%Y",
                       utc=True, dayfirst=True)
```

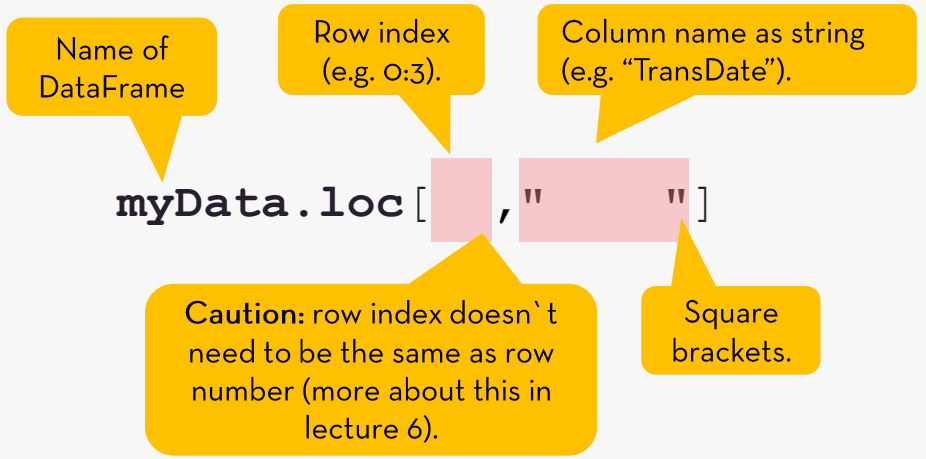
Pandas recognizes often used separators as "-" and "." automatically, but it is safer to specify the format explicitly.

Function is part of the pandas library.

Ensure the right timezone.

Ensure correct month and day ordering.

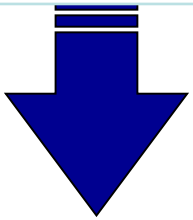
Sidenote: General command structure for addressing columns in a Pandas DataFrame



Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Checking your data
3) Look at summary statistics

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...



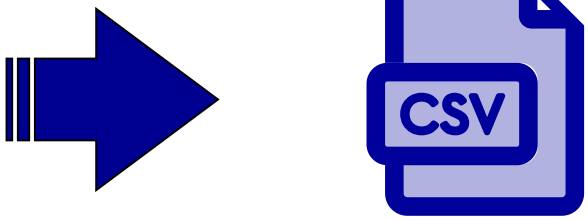
```
myData.describe()
```

Are the summary statistics as you expect them to be?

```
          Customer       Quantity    PurchAmount           Cost
count  223191.000000  223191.000000  223191.000000  223191.000000
mean   148366.708384       1.037009      84.164615      39.013295
std     28657.866956       0.336899     105.864308      57.145100
min    100001.000000       1.000000       0.000000       0.000000
25%    123563.000000       1.000000      34.950000      14.030000
50%    148635.000000       1.000000      59.950000      24.000000
75%    172467.000000       1.000000      99.950000      45.000000
max    199999.000000      70.000000    5000.000000    3100.000000
```

Write data as CSV

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...



Object to save.

Location and name of output CSV file.

```
myData.to_csv(path_or_buf="file.csv",
              sep="," ,...)
```

Value separation.

Sidenote: Remove objects from your workspace

When you are finished with an object it is good practice (but not obligatory) to remove it from your workspace. Thus, you save storage and keep your programming environment clean:

```
del DataFrame
```

`del` removes files from your environment.

Exercise
Basic investigation of your data

1. Look at the data that you loaded in the previous exercise.
2. Use pandas' `to_datetime()`-function to format the `TransDate` column.
3. Use `describe()` to see if the change was made correctly. (How can you tell?)
4. Save the DataFrame object to a csv-file with the name `transactions_students_backup.csv`.

Data manipulation & selecting rows

Data manipulation is the most intensive part of a project

- Data manipulation usually takes 80% of the time of your entire project.
- Avoid inefficient code:
  - Will take long to execute.
  - Is more likely to have bugs or mistakes.

Three basic data operations

- Select

Customer	TransDate
149332	2005-11-15
172951	2008-08-29

Select rows where TransDate < 2007

Customer	TransDate
149332	2005-11-15

- Aggregate

Customer	Quantity
149332	1
172951	1
149332	1

Sum up the „Quantity“ for each customer

Customer	Quantity
149332	2
172951	1

- Merge

Customer	TransDate
149332	2005-11-15
172951	2008-08-29

+

Customer	Quantity
149332	1
172951	1

Customer	TransDate	Quantity
149332	2005-11-15	1
172951	2008-08-29	1

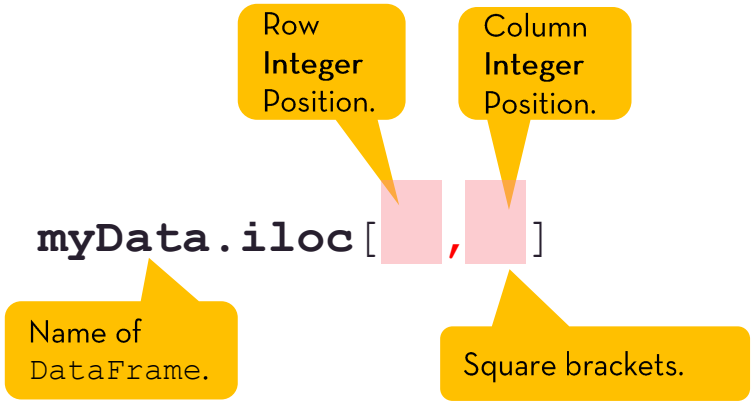
By selecting data from our dataset we can answer the following questions

- Which customers joined in 2015?
- Which customers spent the most on a single transaction?
- Which transactions had a purchase amount greater than 100?

General command structure for adressing rows and columns in a pandas DataFrame (1/2)

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

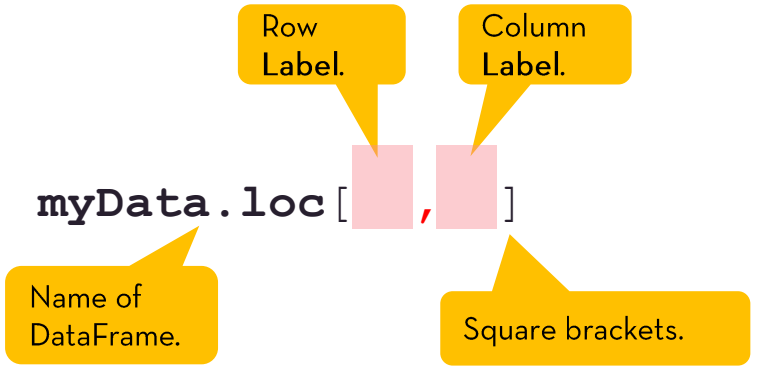
Option 1: Integer based selection with `iloc`



General command structure for addressing rows and columns in a pandas DataFrame (2/2)

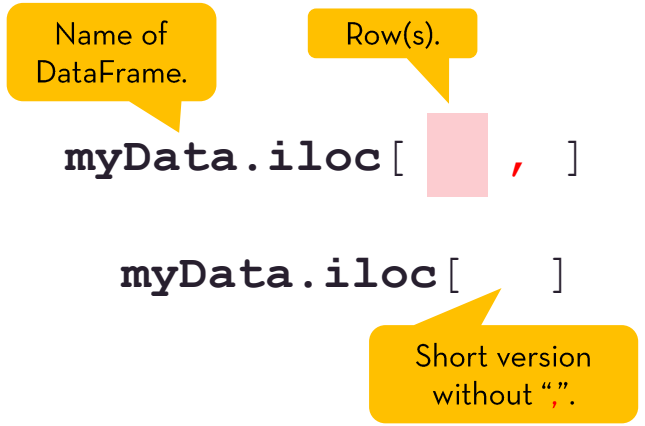
Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Option 2: Label-based selection with `loc`



Enter your selection commands in the row place-holder

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

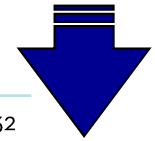


There are multiple ways of selecting rows

1. Selecting rows by row numbers.
2. Selecting rows by conditions.

Selecting rows by row numbers
Select the first row

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...



Select the first row

Returns a Series.

Customer	149332
TransDate	15.11.2005
Quantity	1
PurchAmount	199.95
Cost	107.00
TransID	127998739

Row number(s) to be selected.

```
myData.iloc [0, ]
```

Returns a DataFrame

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739

```
myData.iloc [[0], ]
```

Row number(s) to be selected.

Sidenote: Selecting does not make changes to the original data table

Select first row.

```
myData.iloc[[0],]
```

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739

The output of select operations need to be stored via "=".

myData

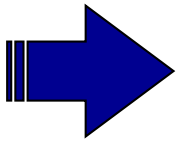
Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	12.06.2007	1	79.95	35.00	128670302

myData was not changed.

Selecting rows by row numbers
Select the first 3 rows

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Select the first 3 rows



Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247

```
myData.iloc[0:2, ]
```

Row numbers to be selected.
":" generates a sequence for slicing.

Sidenote: “0”-based indexing in Python

- Python uses 0-based indexing, i.e. the first index is 0.

Python Basics: Use the colon operator (:) for selection procedures

" : " generates a regular sequence:

```
> myData.iloc[0:5]
```

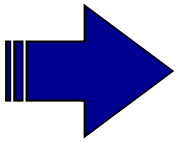
	Customer	TransDate	Quantity	PurchAmount	Cost	TransID
0	149332	15.11.2005	1	199.95	107.00	127998739
1	172951	29.08.2008	1	199.95	108.00	128888288
2	120621	19.10.2007	1	99.95	49.00	125375247
3	149236	14.11.2005	1	39.95	18.95	128670302
4	149236	12.06.2007	1	79.95	35.00	128670302

```
> myData.iloc[0:0]
Empty DataFrame
```

Selecting rows by row numbers
Select the first 3 and the 5th row

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Select the first 3 and the 5th row



Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	12.06.2007	1	79.95	35.00	128670302

```
myData.iloc[[0,1,2,4], ]
```

Combine integers in a **vector**.

Python Basics: Advanced slicing of lists

Reduced syntax:

- Select all elements **beginning with index start**:

  array[start :] myData.iloc[5 :]

- Select all elements from **index 0 to index stop**:

  array[: stop] myData.iloc[: 5]

- Select every index from start to stop which is a multiple of step:

  array[: : step] myData.iloc[: : 2]

Index	Customer	TransDate	Quantity	PurchAmount	Cost	TransID
2	120621	19.10.2007	1	99.95	49.00	125375247
4	149236	12.06.2007	1	79.95	35.00	128670302

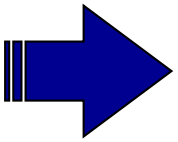
Select every second row from from start=2 to stop=6

```
myData.iloc[2:6:2, ]
```

Selecting rows by row numbers
Select the last row

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...
199542	17.09.2012	1	39.95	10.50	131973368

Select the last row



Customer	TransDate	Quantity	PurchAmount	Cost	TransID
199542	17.09.2012	1	39.95	10.50	131973368

Brackets are needed if representation of the output as Data.Frame is desired.

```
myData.iloc[[len(myData)-1], ]
myData.tail(1)
```

len() gives the number of rows. To obtain the last one we have to correct our index by- 1 (reason: zero-based indexing).

Python Basics: Understand the dimensions of your `Data.Frame`

Important functions to determine dimensions:

- Number of rows/columns:

```
myData.shape[0]
myData.shape[1]
```

- Length of a vector:

```
len([2,3,5])
```

- Length of string:

```
len("hello")
```

Sidenote: How to sort your DataFrame

- To sort your DataFrame according to transaction dates (increasing) use:

```
myData.sort(["TransDate"])
```

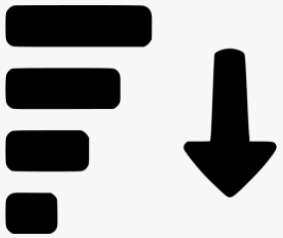
- Order first according to transaction dates and then according to customers:

```
myData.sort(["TransDate", "Customer"])
```

- Order decreasing:

```
myData.sort(["TransDate", "Customer"],
                        ascending=[0,0])
```

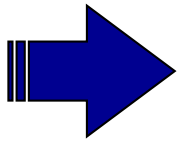
Specify decreasing order explicitly
(by default: 1 = increasing).



Selecting rows by condition
Identify transactions greater than $100

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Select transactions with value greater than 100



Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
...	...	...	...	...	...

```
myData.loc[myData["PurchAmount"]>100, ]
```

Select all transactions > $100.

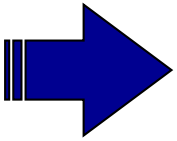
Python Basics: Logical Operators

Sign	Description	Example
<	less than	a < 0
<=	less than or equal than	a <= 3
>	greater than	a > 0
>=	greater than or equal than	a >= 3
==	equal to	a == 0
!=	not equal to	!= 0
not	logical negotiation (NOT)	not x
&	logical AND	x & y
|	logical OR	x | y

Selecting rows by condition
Select the transactions of a single customer

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Select transactions where customer is 149332



Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
...	...	...	...	...	...

Note: Variable "Customer" is of type integer.

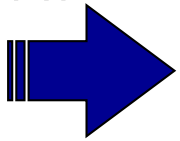
```
myData.loc[myData["Customer"]==149332, ]
```

Selects all observations where **Customer** is equal to **149332**.

Selecting rows by condition
Select the transactions of multiple customers

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Select transactions where customer is 149332 or 172951



Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
...	...	...	...	...	...
172951	29.08.2008	1	199.95	108.00	12888288
...	...	...	...	...	...

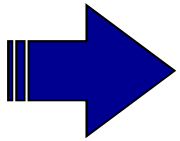
```
myData.loc[myData["Customer"].isin([149332, 172951]), ]
```

Selects all observations where **Customer** is either **149332** or **172951**.

Tilde operator (~) precedes an index vector to negate the condition

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Select transactions where customer is NOT 149332 or 172951



Customer	TransDate	Quantity	PurchAmount	Cost	TransID
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

```
myData.loc[~ myData["Customer"].isin([149332, 172951]), ]
```

"Not in"

Combining conditions

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...
140729	28.04.2012	1	89.95	35.00	129377737
...	...	...	...	...	...

Select transactions where

1) ***date after 24.10.2010***
2) ***PurchAmount greater than 70***

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
140729	28.04.2012	1	89.95	35.00	129377737
...	...	...	...	...	...

```
myData.loc[(myData["TransDate"]>
           pd.to_datetime("2010-12-24")) &
          (myData["PurchAmount"]>70), ]
```

Create a `date`.

Combine multiple conditions.

Round brackets.

Exercise
Selecting rows

1. Select rows 10 to 20.
2. Select all purchases from 2010.
3. Select purchases made on 19.12.2010 or on 21.12.2010.
4. Select all purchases with quantities between 2 and 5.
5. Select all purchases with purchase amount greater than 100 which were made from 01.01.2009 onwards.

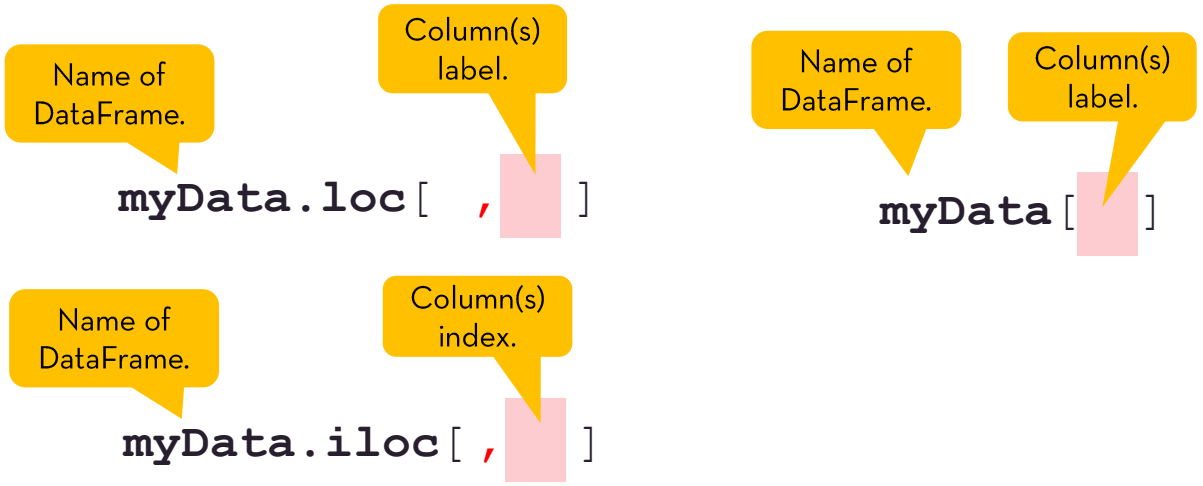
Selecting columns

There are multiple ways of selecting columns

1. Select a single column
2. Select multiple columns
3. Combine operations to select by rows and columns

Enter your selection commands in the column placeholder

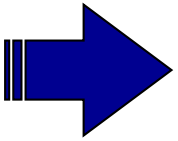
Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...



Select a single column by column name / number (1/2)

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Select column "TransDate"



TransDate
15.11.2005
29.08.2008
19.10.2007
14.11.2005
12.06.2007
...

Returns a Data Frame

```
[223191 rows x 1 columns]
```

DataFrame name. All row.s Column name.

```
myData.loc[ : , "TransDate"]
```

```
myData.iloc[ : , 1]
```

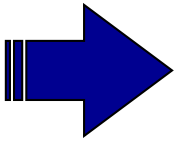
Select the second column (corresponding index=1).

```
myData ["TransDate"]
```

Select a single column by column name / number (2/2)

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Select column "TransDate" and return it as numpy array



```
array(['2005-11-
15T00:00:00.000000000',
'2008-08-
29T00:00:00.000000000',
'2007-10-
19T00:00:00.000000000', ...,
dtype='datetime64[ns]')
```

Returns a numpy array.

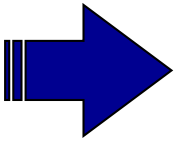
Turn the output into array.

```
myData.loc[ : , "TransDate"].values

myData.iloc[ : , 1].values

myData ["TransDate"].values
```

Select multiple columns by column name / number

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Select columns "Customer, "TransDate" and "Purch-Amount"



Customer	TransDate	PurchAmount
149332	15.11.2005	199.95
172951	29.08.2008	199.95
120621	19.10.2007	99.95
149236	14.11.2005	39.95
149236	12.06.2007	79.95
...	...	...

List of column vectors.

Column names.

```
myData.loc[ : , ["Customer", "TransDate", "PurchAmount"]]
```

```
myData.iloc[ : , [0,1,3]]
```

Column indexes.

```
myData[["Customer", "TransDate", "PurchAmount"]]
```

Python Basics: Find out the column names

- Get all columns names from your DataFrame:

```
myData.columns.values
```

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Sidenote: `Set_index` sets columns as row labels and enables advanced selection procedures

Row labels can be helpful for advanced selection procedures.

- Set an index:

```
myData_Cust=myData.set_index(['Customer'])
```

Set "`Customer`" as index.

- Label-based index selection with `loc( ):`

```
myData_Cust.loc[149332]
```

Return all entries for "Customer=149332".

Customer	TransDate	Quantity	...
149332	15.11.2005	1	.....
149332	13.12.2005	1	
149332	10.05.2006	1	

- Caution: `myData_Cust.iloc[149332]` returns the integer row count, i.e. the 149332nd row in the DataFrame.

Combine operations to select by row and column

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

Select column "TransDate" and "Cost" for entries where "PurchAmount" greater than 100

TransDate	Cost
15.11.2005	107.00
29.08.2008	108.00
...	...

Filtering condition for rows.

Select columns.

```
myData.loc[ myData["PurchAmount"]>100 , ["TransDate","Cost"]]

myData[["TransDate","Cost"]].loc[ myData["PurchAmount"]>100 ,]
```

Select columns first.

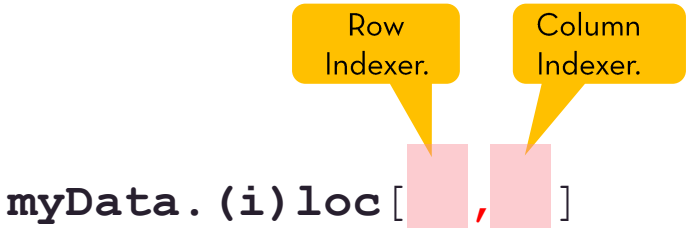
Exercise
Selecting columns

1. First select rows 1 to 100.
2. From this, select the following columns: `Customer`, `TransDate`, and `Quantity`.
3. Select the columns `Customer` and `Cost` from all rows with a purchase amount between 100 and 200.

Summary for adressing rows and columns in a pandas DataFrame

- Comparison of valid inputs for `iloc` and `loc`:

Option 1: loc (Label based)		Option 2: Iloc (Integer position based)	
Single label	*5 or 'a'*	Integer	*5*
list or array of labels	*['a', 'b', 'c']*	list or array of integers	*[4, 3, 0]*
slice object with labels	*'a':'f'*	slice object with ints	*1:7*
boolean array	*['True', 'True', 'False']*		



Appending and updating rows and columns

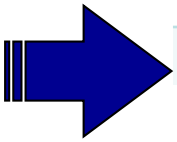
Appending rows into DataFrame

myData

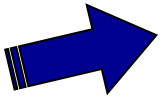
Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...

New row

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
199332	18.11.2011	1	99.95	47.00	197923739



Append the row to the DataFrame



Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
199332	18.11.2011	1	99.95	47.00	197923739
...	...	...	...	...	...

Make sure the formats are correctly set!

```
myData.append(pd.DataFrame([[199332, pd.to_datetime("2011-11-18"),
                             1,99.95,47.00, 197923739]],
                           columns=myData.columns.values),
              ignore_index=True)
```

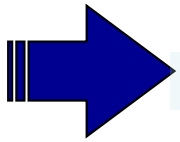
New row.

Ensures that the row index is automatically increased by one.

Adding columns to a DataFrame

Customer	TransDate	Quantity	PurchAmount	Cost
149332	15.11.2005	1	199.95	107.00
172951	29.08.2008	1	199.95	108.00
120621	19.10.2007	1	99.95	49.00
149236	14.11.2005	1	39.95	18.95
149236	12.06.2007	1	79.95	35.00
...	...	...	...	...

Add column "NewCol"



Customer	TransDate	Quantity	PurchAmount	Cost	NewCol
149332	15.11.2005	1	199.95	107.00	107
172951	29.08.2008	1	199.95	108.00	108
120621	19.10.2007	1	99.95	49.00	49
149236	14.11.2005	1	39.95	18.95	18
149236	12.06.2007	1	79.95	35.00	35
...	...	...	...	...	...

New column.

```
myData["NewCol"] = myData["Cost"].round()
```

Specify column name.

Round "Cost"-variable to a whole number.

Python Basics: By default DataFrame changes are made by reference

This can result in a messy situation:

```
myData.head(3)
```

myData before adding a new column.

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247

```
myData_modified=myData
myData_modified["NewCol2"]=
                myData_modified["Cost"].round()
myData.head(3)
```

Duplicate **myData**.

Add a new column to **myData_modified**.

Customer	TransDate	Quantity	PurchAmount	Cost	TransID	NewCol2
149332	15.11.2005	1	199.95	107.00	127998739	107.0
172951	29.08.2008	1	199.95	108.00	128888288	108.0
120621	19.10.2007	1	99.95	49.00	125375247	49.0

Changes are made to **myData** too.

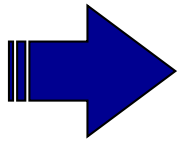
Use `copy()` to avoid this:

```
myData_modified = myData.copy()
```

Remove a single column from a DataFrame

Customer	TransDate	Quantity	PurchAmount	Cost
149332	15.11.2005	1	199.95	107.00
172951	29.08.2008	1	199.95	108.00
120621	19.10.2007	1	99.95	49.00
149236	14.11.2005	1	39.95	18.95
149236	12.06.2007	1	79.95	35.00
...	...	...	...	...

Remove column "Quantity"



Customer	TransDate	PurchAmount	Cost
149332	15.11.2005	199.95	107.00
172951	29.08.2008	199.95	108.00
120621	19.10.2007	99.95	49.00
149236	14.11.2005	39.95	18.95
149236	12.06.2007	79.95	35.00
...	...	...	...

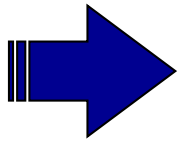
```
del myData["Quantity"]
```

Changes made by reference!!

Remove multiple columns at the same time

Customer	TransDate	Quantity	PurchAmount	Cost
149332	15.11.2005	1	199.95	107.00
172951	29.08.2008	1	199.95	108.00
120621	19.10.2007	1	99.95	49.00
149236	14.11.2005	1	39.95	18.95
149236	12.06.2007	1	79.95	35.00
...	...	...	...	...

Remove columns "Quantity" and "Cost"



Customer	TransDate	PurchAmount
149332	15.11.2005	199.95
172951	29.08.2008	199.95
120621	19.10.2007	99.95
149236	14.11.2005	39.95
149236	12.06.2007	79.95
...	...	...

Column-wise deletion. **`axis = 0`** would delete rows labelled "Quantity" & "Cost"

```
myData.drop(["Quantity", "Cost"], axis=1)
```

List of columns to be deleted.

Python Basics: Change the names of a variable in a DataFrame

- To change names of a DataFrame variable:

Old name.

```
myData.rename(columns = {"PurchAmount":"TransValue"},
                        Inplace=True)
```

New name.

Customer	TransDate	Quantity	PurchAmount	Cost
149332	15.11.2005	1	199.95	107.00
172951	29.08.2008	1	199.95	108.00
120621	19.10.2007	1	99.95	49.00
149236	14.11.2005	1	39.95	18.95
149236	12.06.2007	1	79.95	35.00
...	...	...	...	...

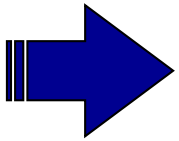
Change variable name "Purch-Amount"

Customer	TransDate	Quantity	TransValue	Cost
149332	15.11.2005	1	199.95	107.00
172951	29.08.2008	1	199.95	108.00
120621	19.10.2007	1	99.95	49.00
149236	14.11.2005	1	39.95	18.95
149236	12.06.2007	1	79.95	35.00
...	...	...	...	...

Remove rows from a DataFrame

Customer	TransDate	Quantity	PurchAmount	Cost
149332	15.11.2005	1	199.95	107.00
172951	29.08.2008	1	199.95	108.00
120621	19.10.2007	1	99.95	49.00
149236	14.11.2005	1	39.95	18.95
149236	12.06.2007	1	79.95	35.00
...	...	...	...	...

Remove the first three rows



Customer	TransDate	Quantity	PurchAmount	Cost
149236	14.11.2005	1	39.95	18.95
149236	12.06.2007	1	79.95	35.00
...	...	...	...	...

```
myData.drop(myData.index[:3], inplace=True)
```

Deletes the first three rows in the DataFrame.

No extra assignment necessary.

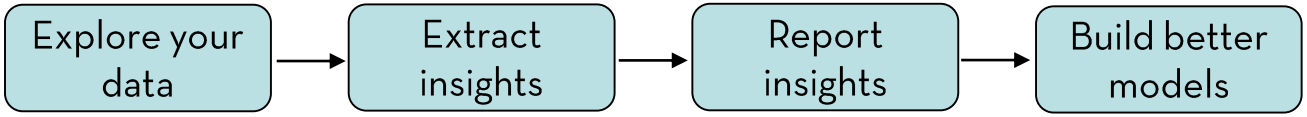
Exercise
Appending and updating rows and columns

1. Create a new column calculating the difference between `PurchAmount` and `Cost`. Call it `Profit`.
2. Rename `Profit` to `ProfitChange`.
3. Delete `ProfitChange` again.
4. Create a copy of `myData` and name it "`mydata_toppurchases`". In the new `DataFrame` delete all transactions with a `PurchAmount` smaller than 100.

Generate plots with matplotlib

Data visualization (in Python)

- Data visualization is an essential part in the data analysis process:



- Python's plotting modules and packages enable customized graphs and more:

Module/Package	Logo	Description
Matplotlib	matplotlib	- Open Source plotting library - Interactive plotting - Syntax fimiliar to Matlab
Ipython	IPython Interactive Computing	- "pylab" mode: designed for interactive plotting with matplotlib
Plotly	plotly	- Collaborative browser-based plotting and analytics platform
ggplot	ggplot	- Based on R's ggplot2 - „Grammar of Graphics": build your plot from various layers
Seaborn		- Visualization library based on matplotlib with simple functions - Provides good default values and integration with Pandas

https://wiki.python.org/moin/NumericAndScientific/Plotting

Data visualization (in Python)
Why use matplotlib and the pylab interface

- The matplotlib library:
  - Provides an easy but flexible Python 2D plotting library which visualizes figures in an interactive environment across platforms.
- The corresponding **pylab** interface:

Import the interface with:
```
import matplotlib.pyplot as plt
```

  - Collection of command style functions that offer plotting in a Matlab-like interface.
  - Each `pyplot` function makes some changes to a figure, e.g. creates a figure, adds some lines, labels the axes of the plot, etc.
  - Keeps track of the current figure and plotting area when adding functions or features to the plot.

Good plots have 3 characteristics

Plots should be:

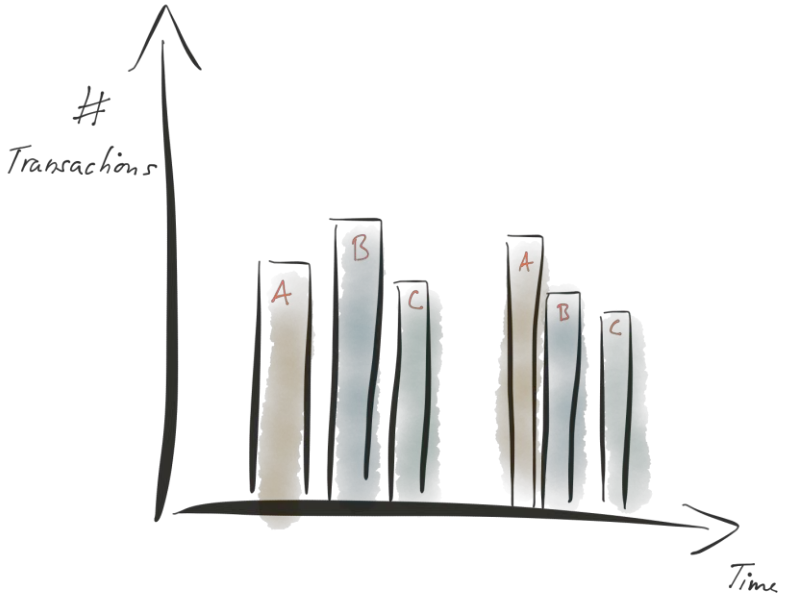
- Informative.
- Easy to understand.
- Visually appealing.

How to plot
Steps

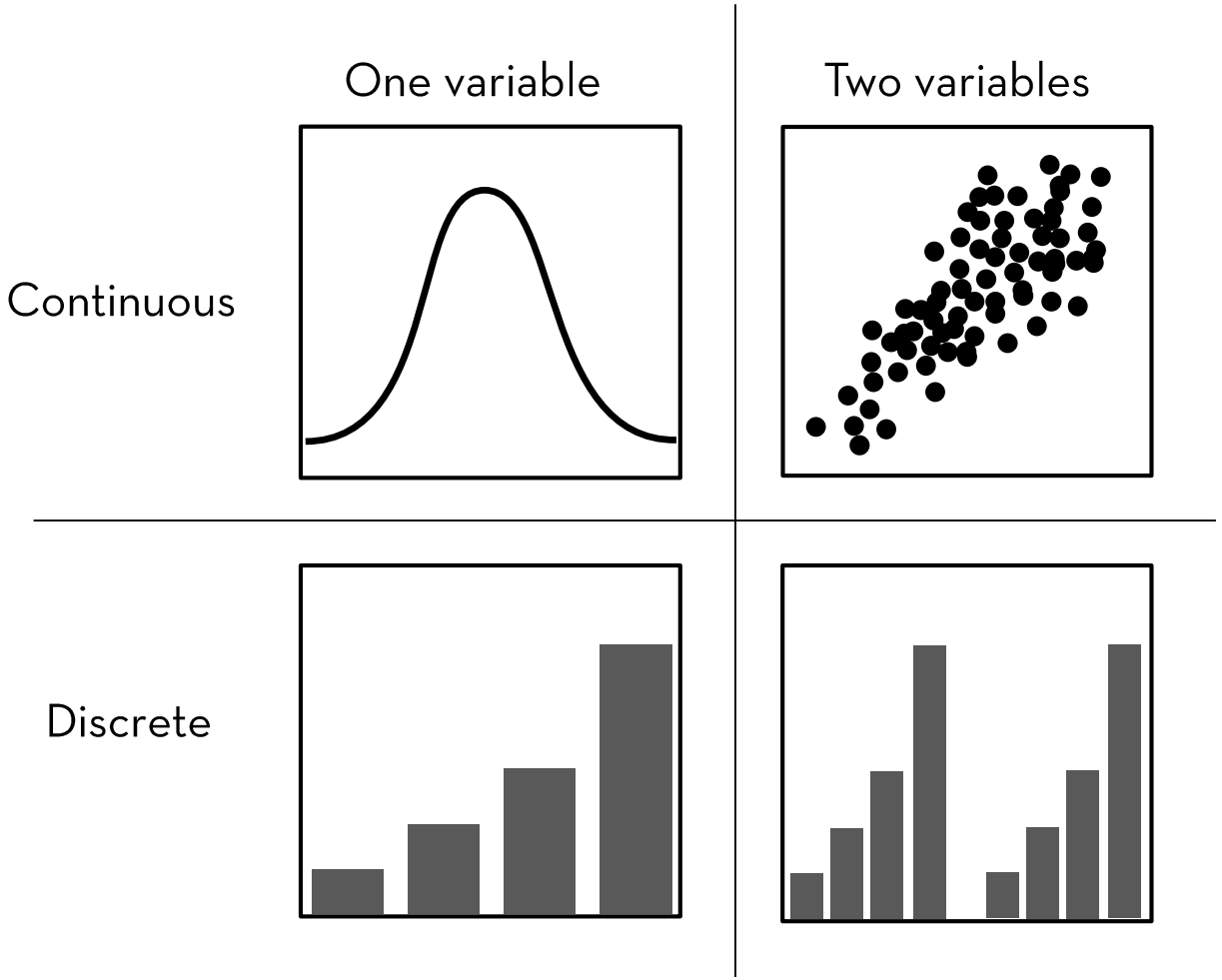
1. Choose a plot type.
2. Find the appropriate matplotlib function.
3. Transform data.
4. Create the plot.
5. Improve aesthetic features of the plot.
6. Save plot.

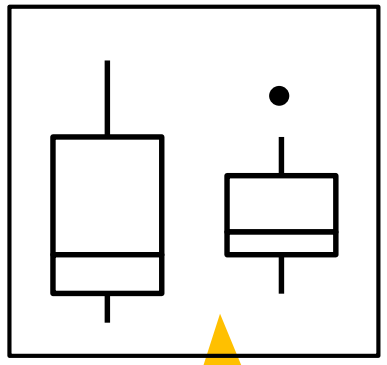
Step 1: Choose the plot type
Decide the best way to convey the information

- What do you want to show?
  - A single variable.
  - The relationship between multiple variables.
- Is your data continuous or discrete?



Different combinations of variables can be portrayed with different plot types

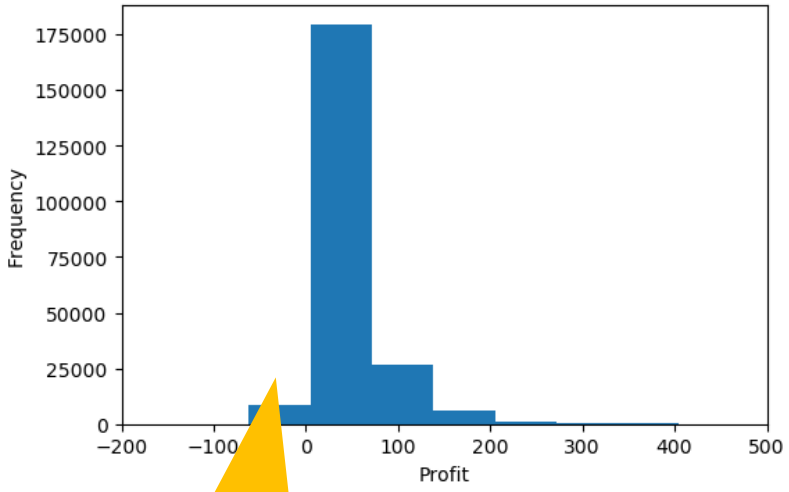




Discrete-continuous.

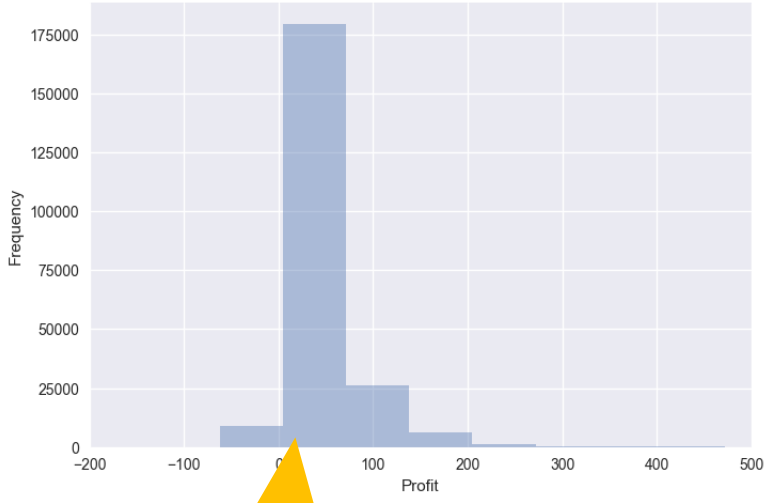
Step 2: Find the function – Pyplot and Seaborn (both Matplotlib-based) are the most used plotting tools

Pyplot



`Pyplot` is a collection of command style functions. Each pyplot function makes some change to a figure: e.g., creates a figure, adds a title, labels the axes, etc.

Seaborn

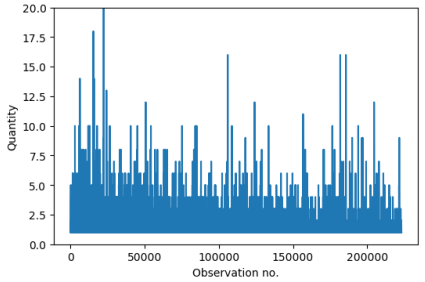


`Seaborn` provides easy functions and good defaults, such as the background and axis styles. It can be extended with `pyplot` functionality.

Step 2: Find the function – Matplotlib with the pyplot interface is the most used plotting tool in Python

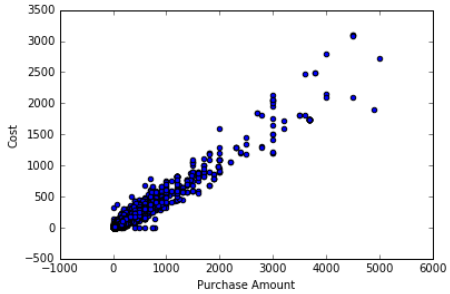
Regular plots (lines, density)

```
plt.plot(
    x, y, . . .)
```



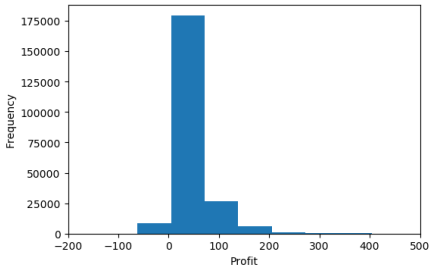
Scatterplot

```
plt.scatter(
    x, . . .)
```



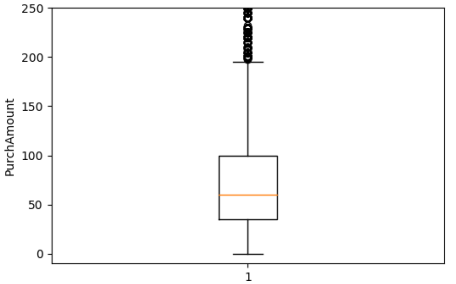
Histogram

```
plt.hist(x, . . .)
```



Boxplot

```
plt.boxplot(
    x, . . .)
```



`pyplot` is a very popular interface that facilitates making plots by deconstructing them into layers.

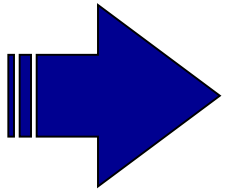
... there's more!

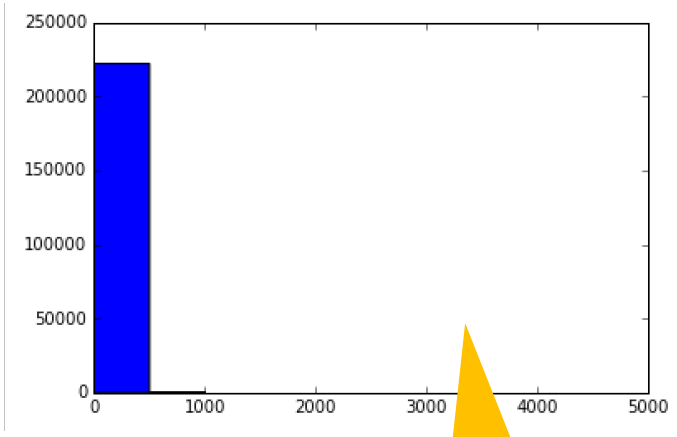
Step 3: Transforming data
Some graphs might require transformed data input

- It is quite rare that you can plot your data right away, i.e. certain **plots have requirements** on how the data should look like .
- In most cases it is **necessary to transform** your data before plotting it.
- Examples:
  - Transform times and dates for aggregation of month or years.
  - Group data for better overview.
  - Logarithmic transformations for nicer distributions.

Step 4: Create the plot (1/2)

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...





A meaningful interpretation is hardly possible.

Multilayer principle:

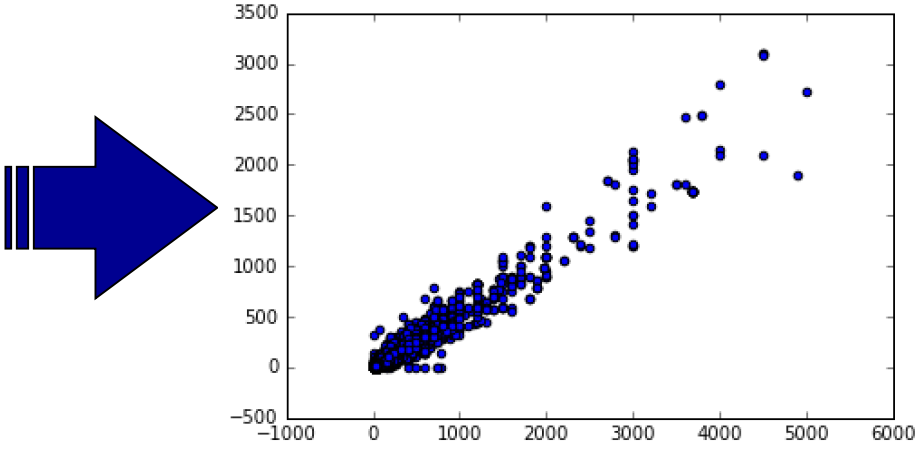
1) Determine how and what to draw.

```
plt.hist(x=myData["PurchAmount"], bin=10)
plt.show()
```

2) Command to actually plot the figure.

Step 4: Create the plot (2/2)

Customer	TransDate	Quantity	PurchAmount	Cost	TransID
149332	15.11.2005	1	199.95	107.00	127998739
172951	29.08.2008	1	199.95	108.00	128888288
120621	19.10.2007	1	99.95	49.00	125375247
149236	14.11.2005	1	39.95	18.95	127996226
149236	12.06.2007	1	79.95	35.00	128670302
...	...	...	...	...	...



x variable

y variable

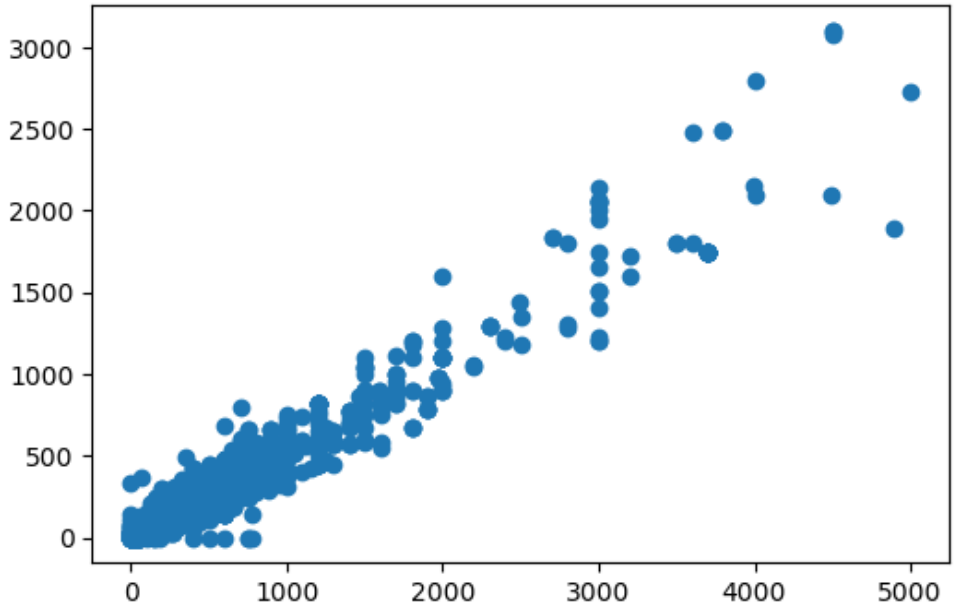
```
plt.scatter(x=myData["PurchAmount"], y=myData["Cost"])
```

Exercise
Generate plots with Matplotlib

1. Create a boxplot for the variable `PurchAmount` (x) .
2. Create a scatter plot for the variables `Quantity` (x) and `PurchAmount` (y). Is there any correlation?

Formatting plots

Step 5: Improve aesthetic feature of the plot
The standard plot output

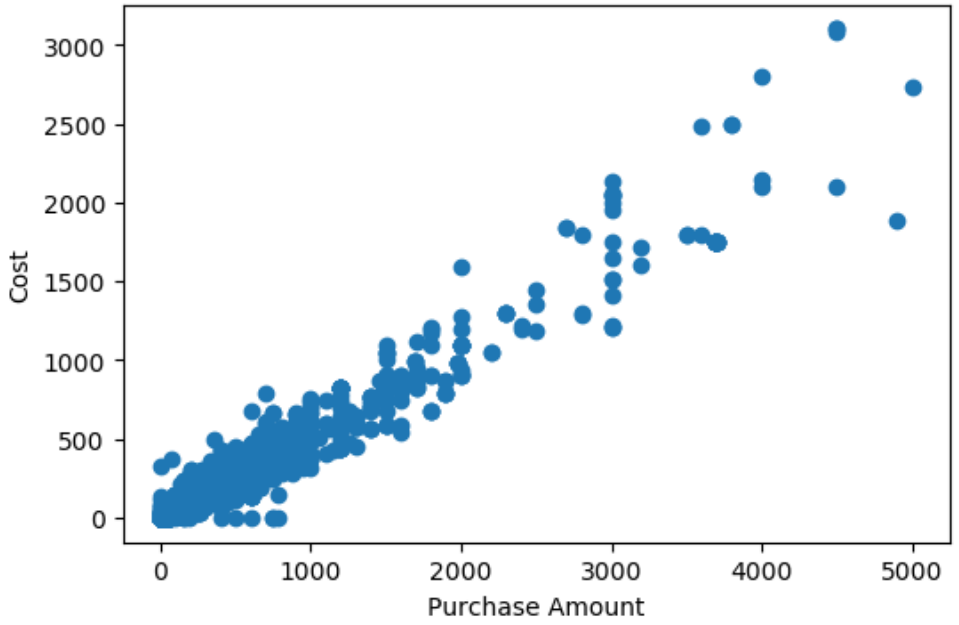


We switch to `plt.plot()`, since `plt.scatter()` offers less options.

```
plt.plot(myData["PurchAmount"], myData["Cost"])
plt.show()
```

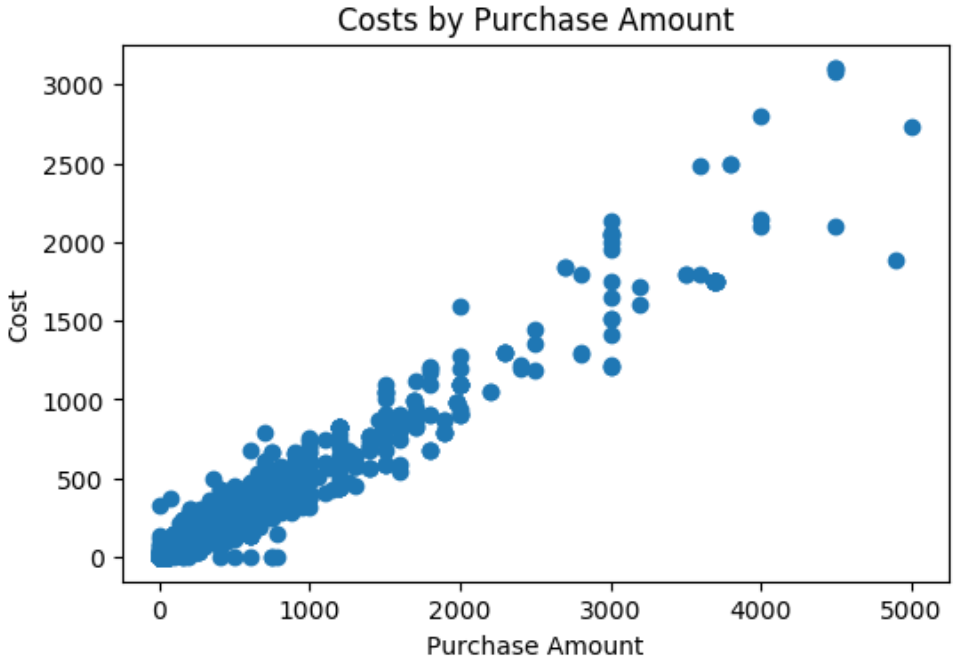
Do not explicitly specify "x=" and "y=".

Change the axis labels




```
plt.plot(x=myData["PurchAmount"], y=myData["Cost"])
plt.xlabel("Purchase Amount")
plt.ylabel("Cost")
plt.show()
```

Add a fitting and descriptive title

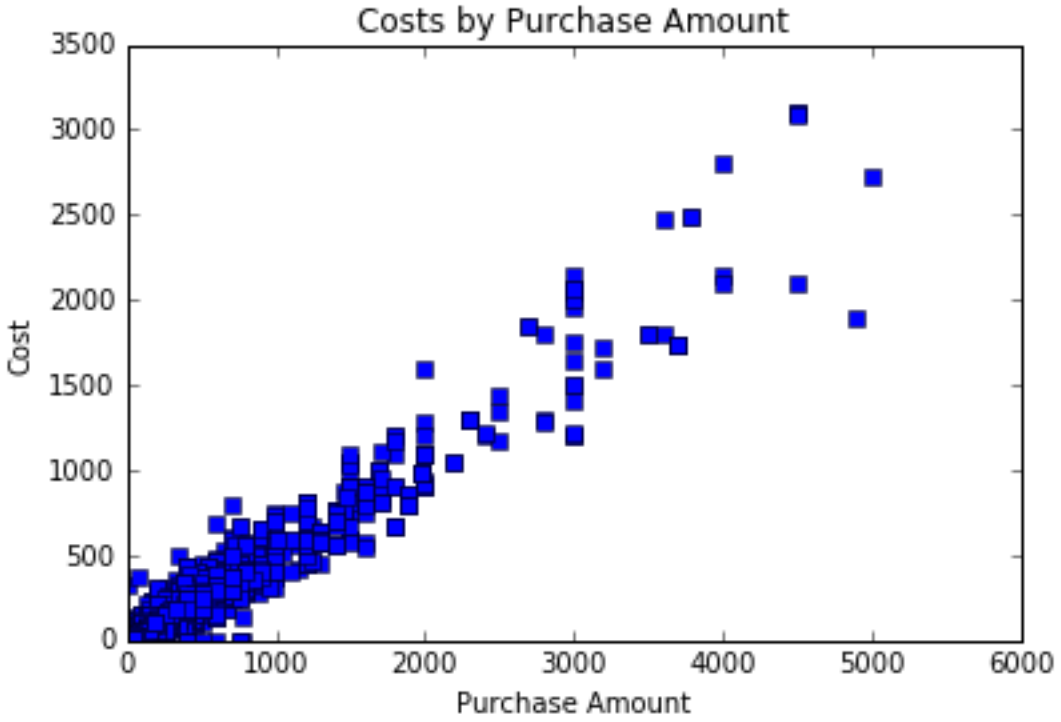



```
...
plt.title("Costs by Purchase Amount")
plt.show()
```

Adjust the axes limits


```
...
plt.xlim(-100,6000)
plt.ylim(-100,3500)
plt.show()
```

Change the marker type to squares



Overview over format strings to control the marker:

Character	Description
's'	square marker
'p'	Pentagon marker
'o'	circle marker
'v'	triangle marker
'x'	x marker

...

```
plt.plot(myData["PurchAmount"], myData["Cost"], "s")
plt.show()
```

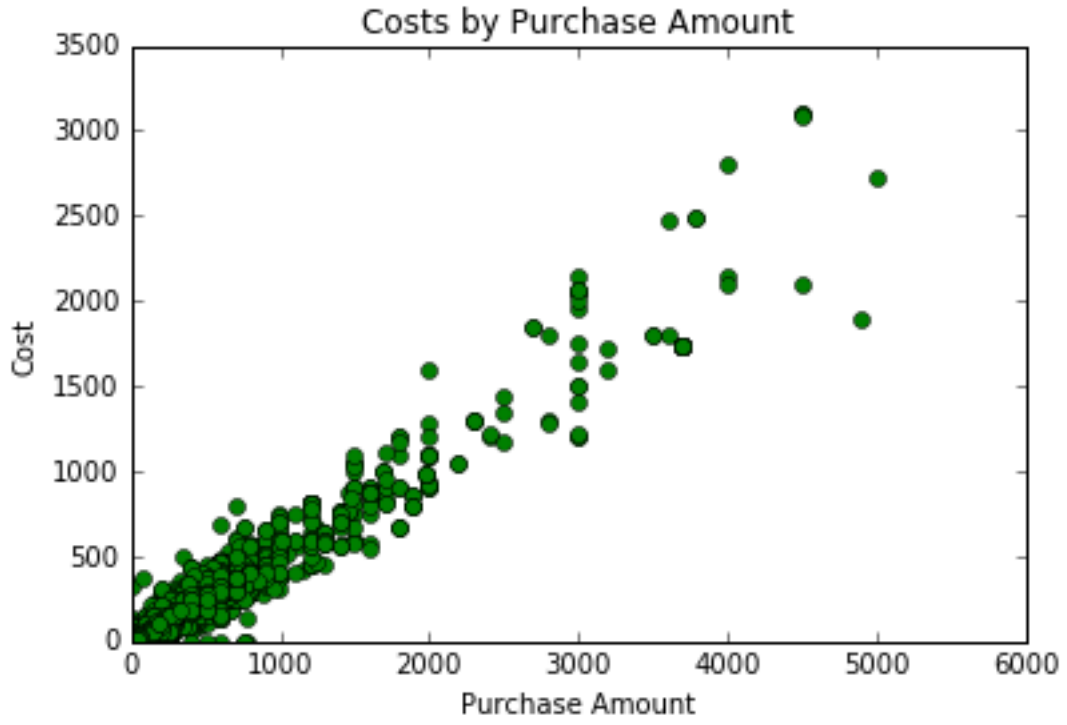
plt.scatter does not work here!

… or to 'x'-s

…

```
plt.plot(myData["PurchAmount"], myData["Cost"], "x")
plt.show()
```

Choose a nice color



...

```
1) plt.plot(myData["PurchAmount"], myData["Cost"], "go")
   plt.show()
```

You can pass the color (g) and the marker (s) in one argument

...

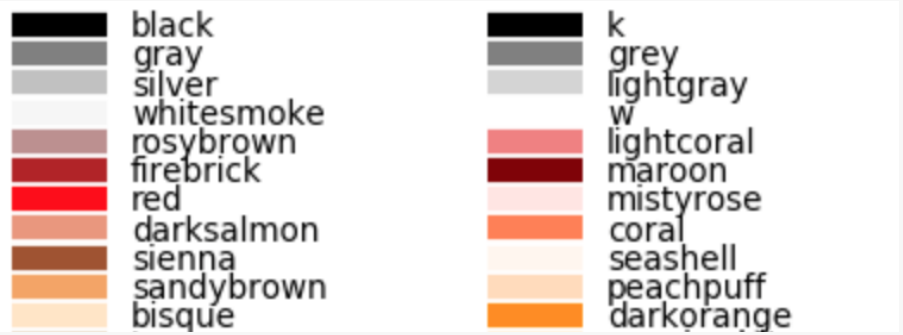
```
2) plt.plot(myData["PurchAmount"], myData["Cost"], "o", color="green")
   plt.show()
```

Alternatively, specify the color in a separate argument.

Sidenote: `Matplotlib`'s color repertoire is huge

- `Matplotlib` colors can be referenced as strings in certain functions:
- Overview over format characters to control the color:

Char-acter	Description
`'b'`	blue
`'g'`	green
`'r'`	red
`'c'`	cyan
...	and many more

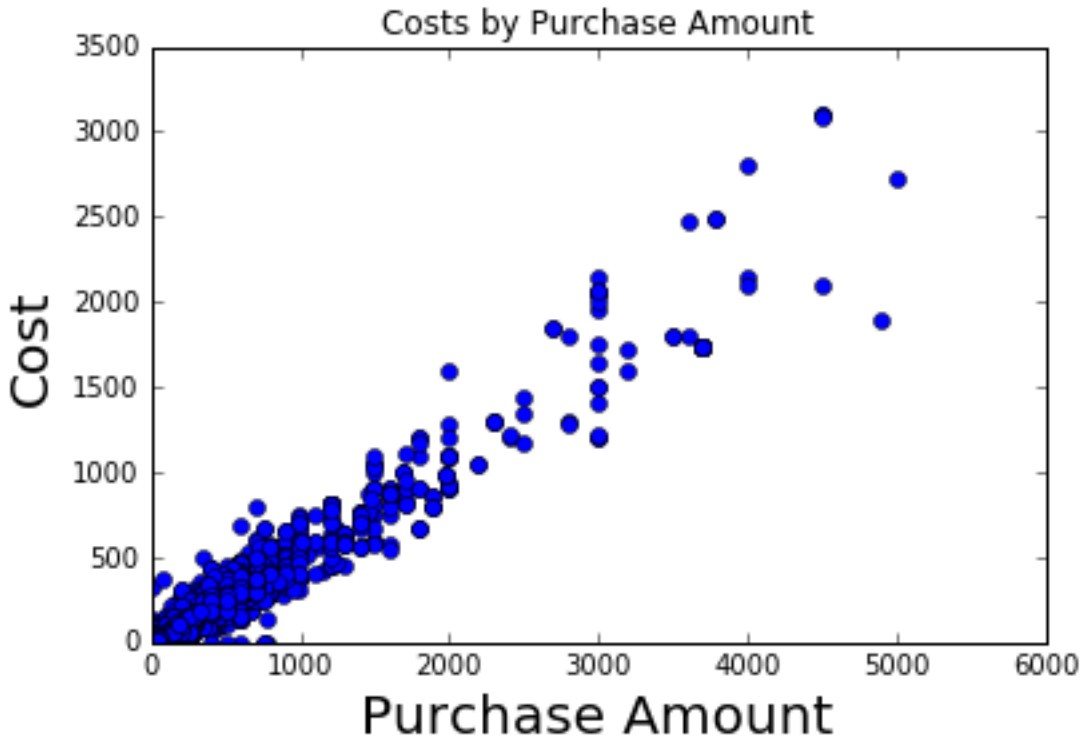


- See https://stackoverflow.com/questions/22408237/named-colors-in-matplotlib

Change the size of the points

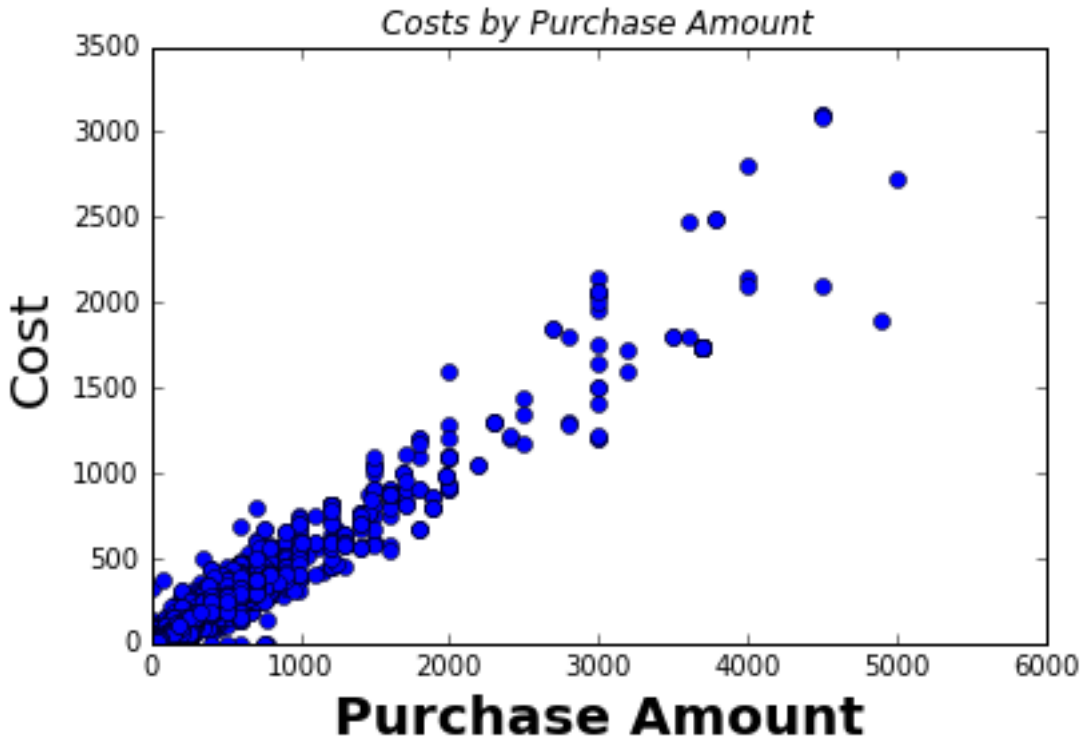

```
plt.plot(myData["PurchAmount"], myData["Cost"],"o", markersize=10)
plt.show()
```

… and the text size




```
plt.plot(myData["PurchAmount"], myData["Cost"], "o")
plt.xlabel("Purchase Amount", size=20)
plt.ylabel("Cost", size=20)
```

We can make the title italic and the axis labels bold too



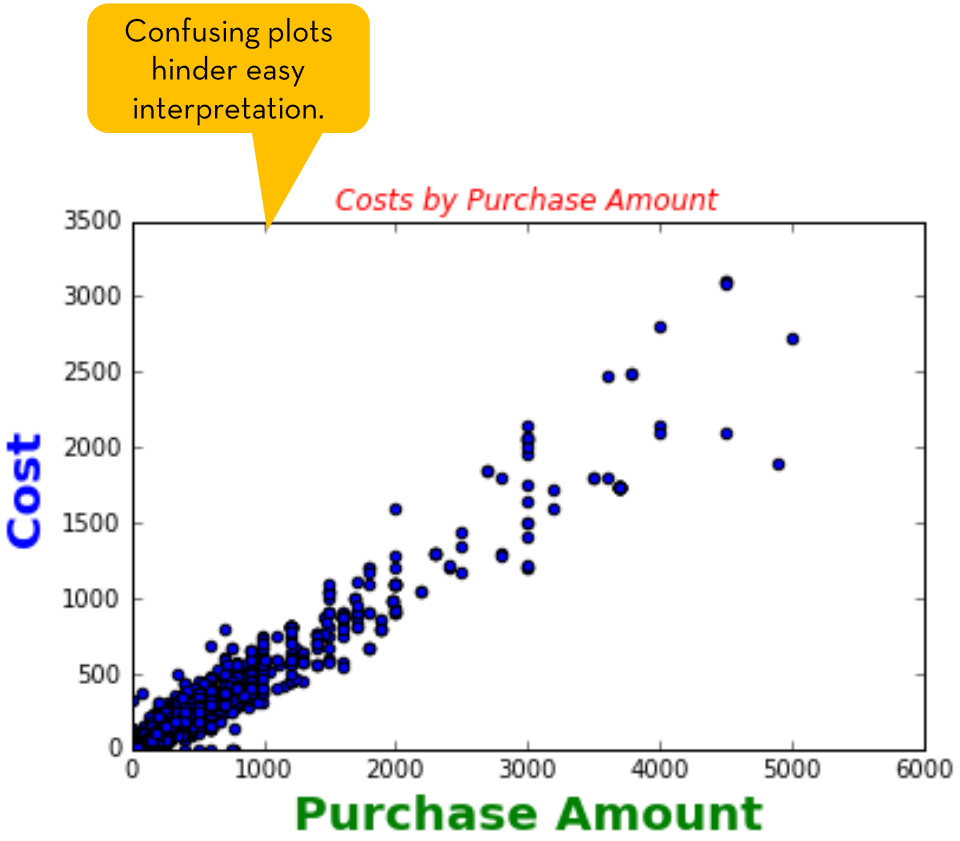

```
plt.plot(myData["PurchAmount"], myData["Cost"], "o")
plt.xlabel("Purchase Amount", size=20, fontweight='bold')
plt.title("Costs by Purchase Amount", style='italic')
```

Remove the box

Pylab combines the pyplot with the numpy functionality

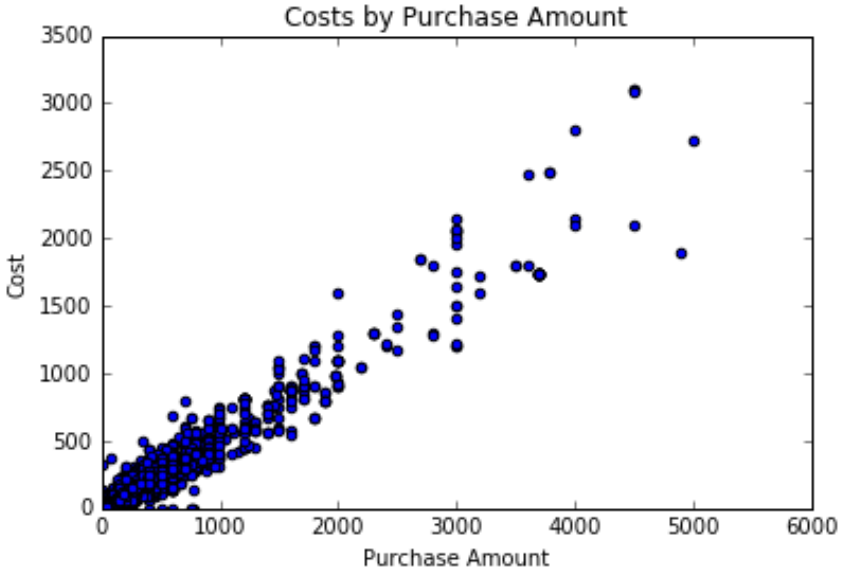
```
from pylab import *
axes(frameon = 0)
plt.plot(myData["PurchAmount"], myData["Cost"], "o")
...
```

Why not include everything?




```
plt.scatter(
        x=myData["PurchAmount"],
        y=myData["Cost"])
plt.xlabel("Purchase Amount",
        size=20,fontweight="bold",
        color="green")
plt.ylabel("Cost", size=20,
        fontweight="bold",
        color="blue")
plt.title("Costs by Purchase
        Amount", style="italic",
        color="red")
plt.xlim(0,6000)
plt.ylim(0,3500)
plt.show()
```

But simple, coordinated plots are nicer!



Even if you **can** change everything, that does not mean that you **should**: Less is sometimes more.

```
plt.scatter(
        x=myData["PurchAmount"],
        y=myData["Cost"])
plt.xlabel("Purchase Amount")
plt.ylabel("Cost")
plt.title("Costs by Purchase Amount")
plt.xlim(0,6000)
plt.ylim(0,3500)
plt.show()
```

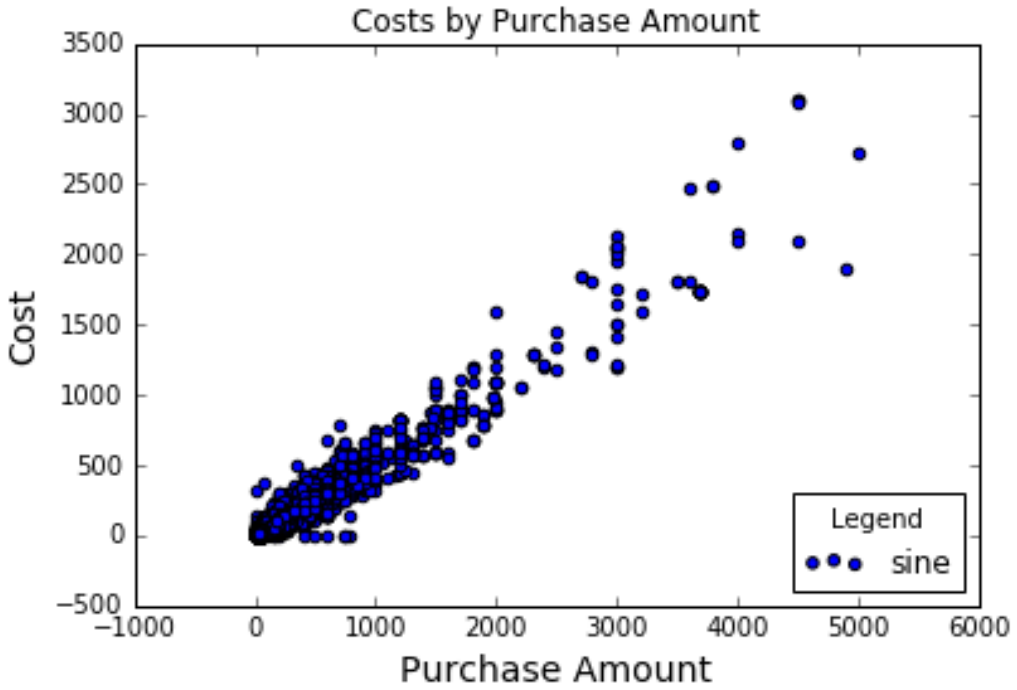
The simple plot requires less code which saves you time.

Exercise
Formatting plots

1. Format your earlier scatter plots in an appealing way .
    - Choose appropriate limits for the axes.
    - Consider changing colors, sizes and fonts for data points and text labels.

Adding further features

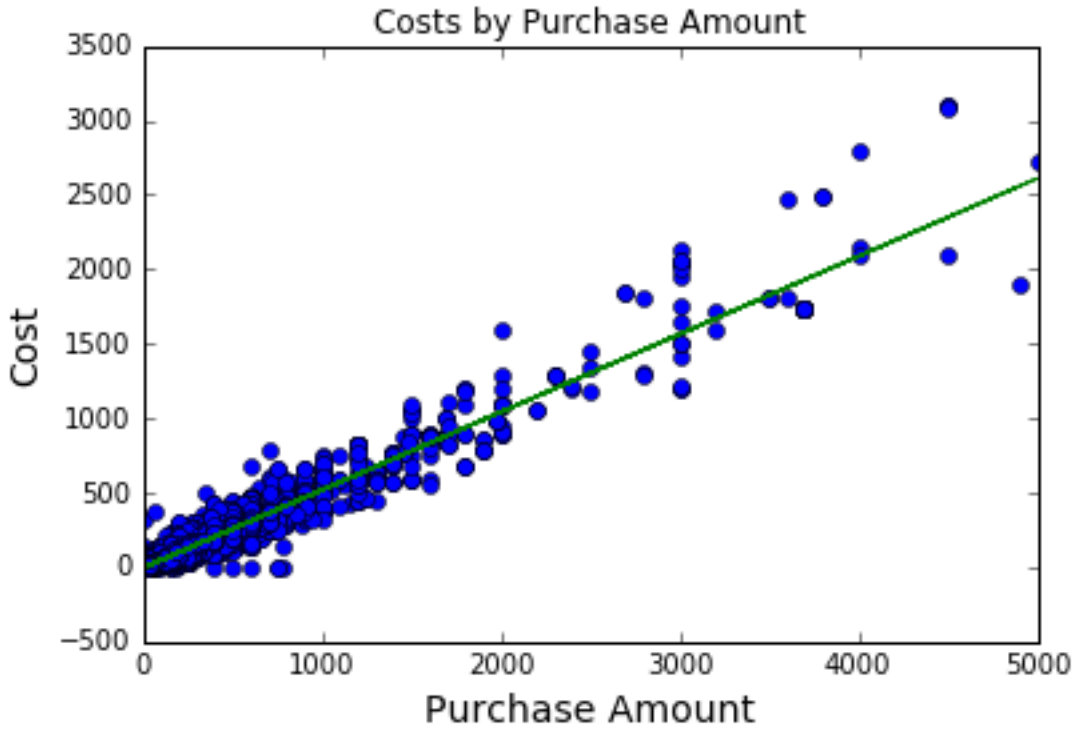
Add a legend to make your plot self-explanatory




```
plt.scatter( x=myData["PurchAmount"], y=myData["Cost"],
                                    label='sine')
plt.legend(loc="lower right", fontsize=12, title="Legend")
                                    plt.show()
```

Indicates the position of the legend in the plot.

Lines can be added for extra information



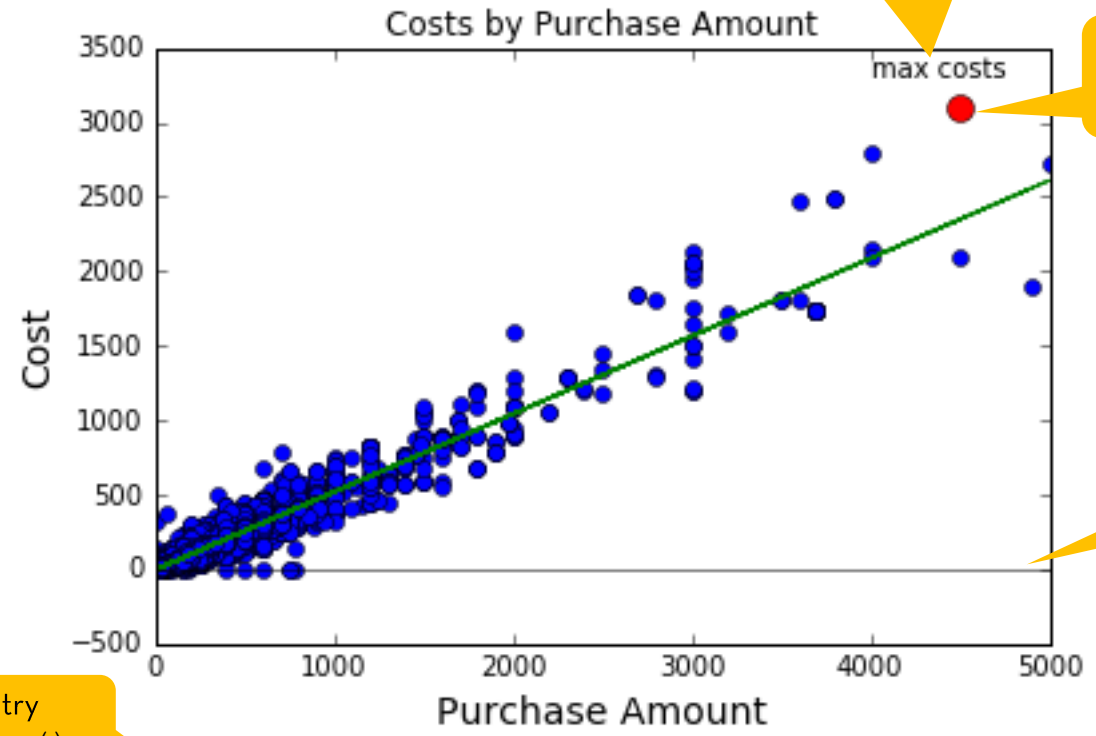
Fitting the linear regression line yields two values:
1) Slope,
2) Intercept.

```
plt.plot(myData["PurchAmount"], myData["Cost"], "o")

fit = np.polyfit(x=myData["PurchAmount"], y=myData["Cost"], deg=1)
plt.plot(myData["PurchAmount"], fit[0] * myData["PurchAmount"] +
                    fit[1],"-", color="green")
                    plt.show()
```

Specify the linetype.

Draw the regression line:
y= m*x + b.

Additional graphical elements can be added in the same fashion



`text()` to add text.

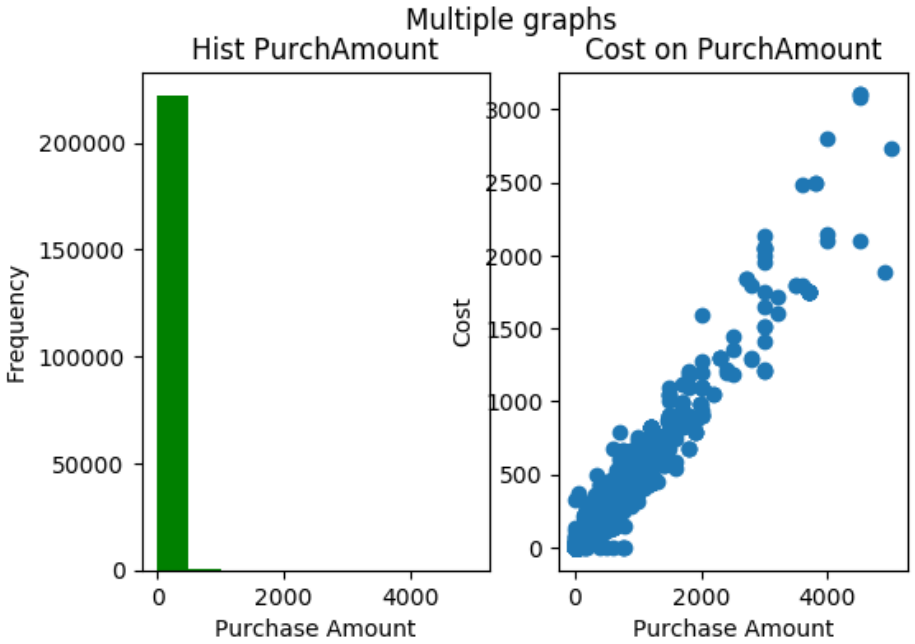
`plot.plot()` to add additional points in different colors.

`axhline()` to add horizontal lines.

Also try `axvline()`.

```
plt.axhline(y=0, xmin=min(myData["PurchAmount"]),
xmax=max(myData["PurchAmount"]), linewidth=0.5, color = 'k')
text(4000,3300, "max costs")
plt.plot(4500, 3100, "o", color="red", markersize=10)
```

You might want to plot multiple graphs on one image



Rows.

Columns.

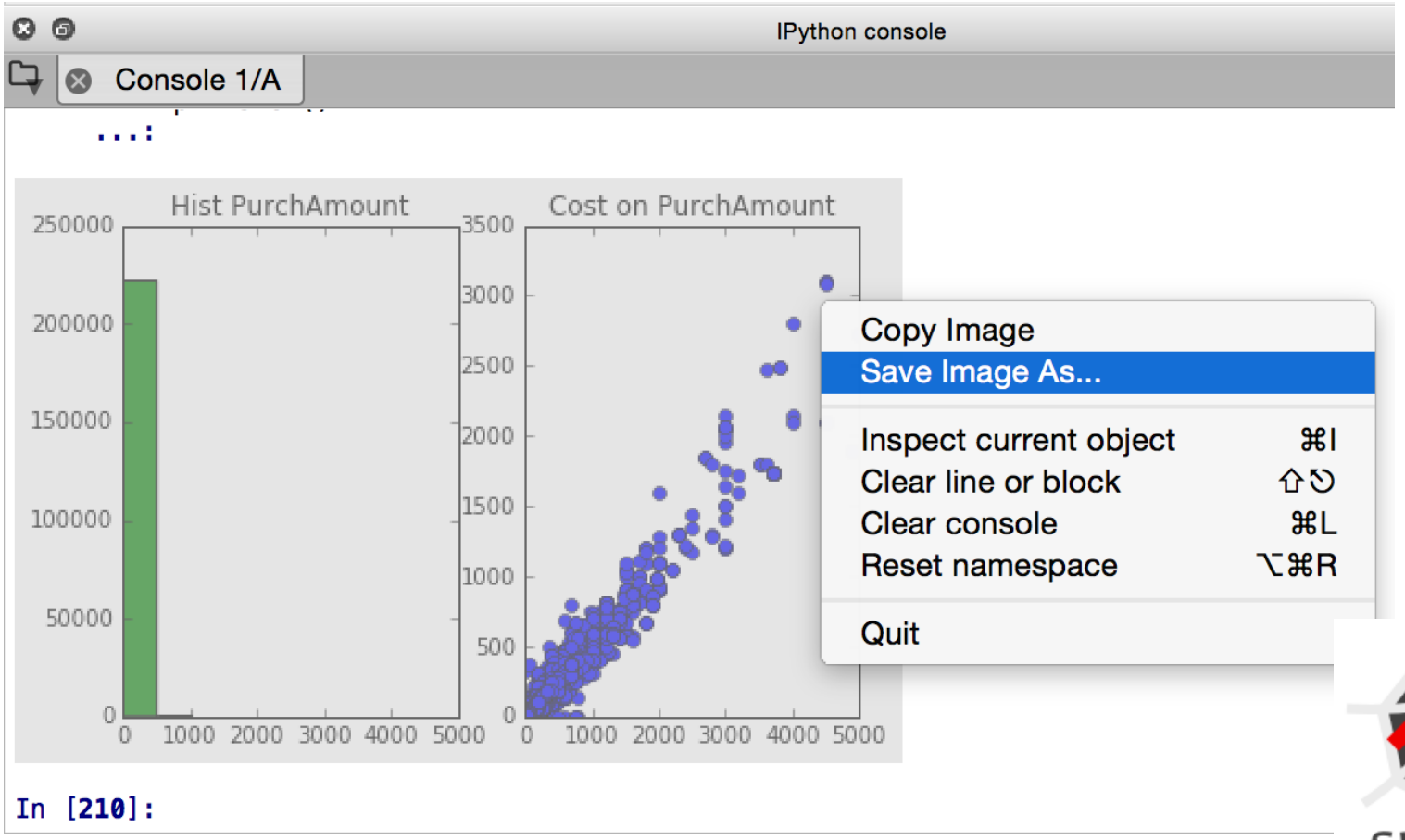
Plot 1.

```
plt.subplot(1,2,1)
plt.hist(myData["PurchAmount"],
         color="green")
plt.title("Hist PurchAmount")
plt.xlabel("Purchase Amount")
plt.ylabel("Frequency")

plt.subplot(1,2,2)
plt.plot(myData["PurchAmount"],
         myData["Cost"], "o")
plt.title("Cost on PurchAmount")
...
plt.suptitle("Multiple graphs")
plt.show()
```

Plot 2.

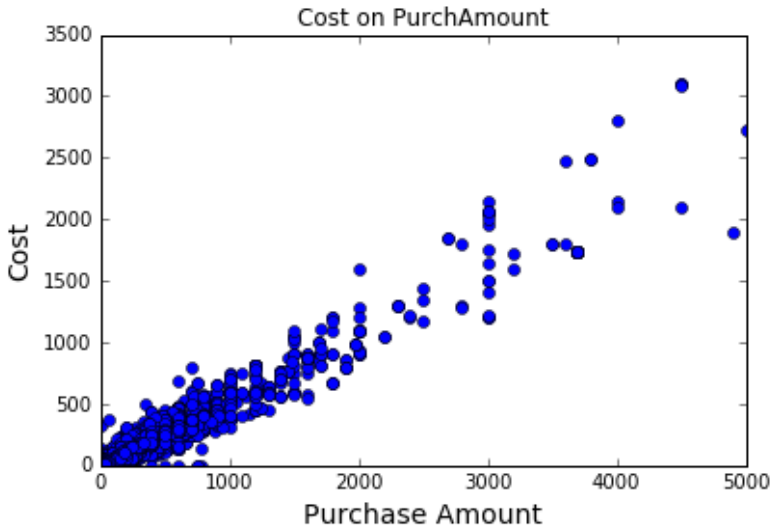
Common title.

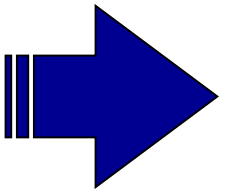
Step 6: Saving plots
Point-and-click method in Spyder



spyder

Saving plots
Using the command line





Caution: Do not show the plot with plt.show(): The picture will be stored blanc if you do

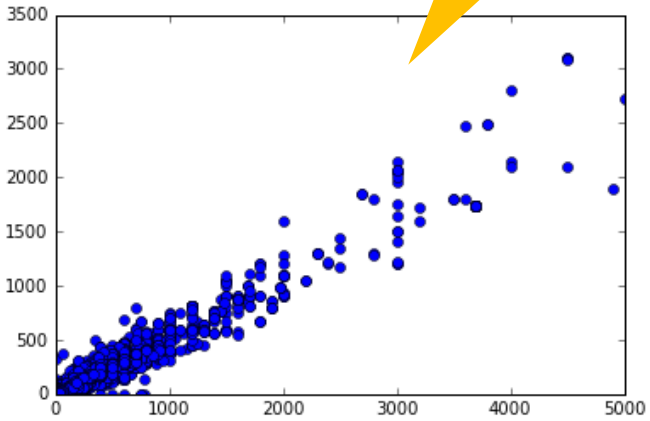
```
plt.plot(myData["PurchAmount"],myData["Cost"], "o")
plt.title("Cost on PurchAmount")
plt.savefig("Output.png")
```

Save the figure.

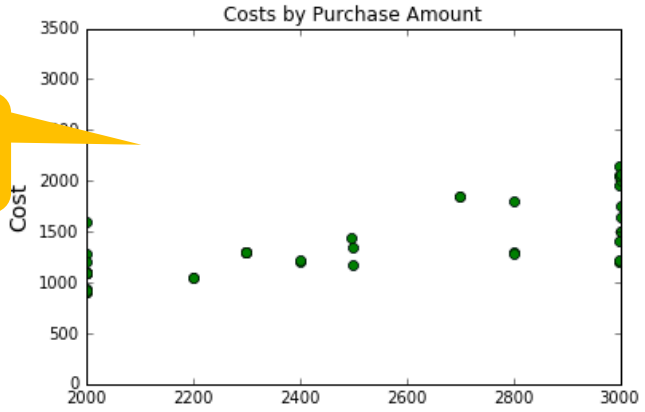
Use .jpg or .png as data formats.

Be careful: Saving the plot with the wrong specifications can ruin your hard work!

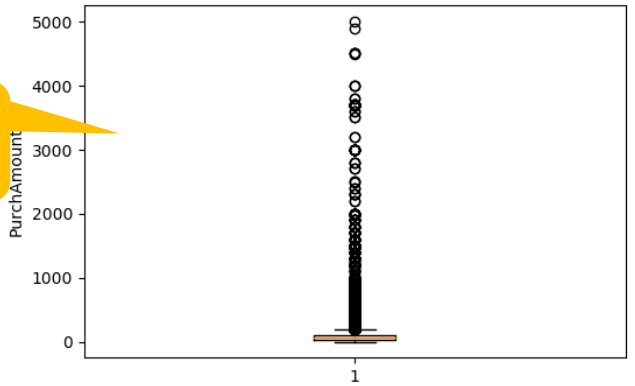
Axes labels and title are missing or not well specified.



Incorrect x-limits showing the wrong extract.



Width and/or height not appropriate.



Advice 1: The point-and-click method helps avoid this.

Advice 2: Always save and comment your code, so that you can modify simple changes with few effort.

Exercise
Adding further features

1. Do the following with the plot „Cost by Purchase Amount“:
   - Add a legend to your second plot.
   - Add a line horizontal to your plot to indicate the costs > 1000.
   - Export your final plot as a pdf using the console.
   - Export your final plot as a png using the Spyder Interface. Open the file.
2. Plot the histogram of Purchase Amount and the „Cost by Purchase Amount“ plot in the same image.