Week 10 – Session 1

DW 10.009 – Introduction to Python Programming

Week 10 Breakdown

- Session 1: Introduction to Data Science
  - Introduction to Numpy
  - Core ideas about data science
  - Data Manipulation and Visualization
- Session 2: Introduction to regression
  - Key parameters for regression
  - Linear regression
  - Multiple linear regression
- Session 3: About classification
  - Key parameters for classification
  - K-NN Classification

Week 10 Breakdown

- Session 1: Introduction to Data Science
  - Introduction to Numpy
  - Core ideas about data science
  - Data Manipulation and Visualization

- Session 2: Introduction to regression
  - Key parameters for regression
  - Linear regression
  - Multiple linear regression
- Session 3: About classification
  - Key parameters for classification
  - K-NN Classification

About the Numpy library

A quick tour of the key concepts and functions

Numpy vs lists

- **Numpy arrays** are objects from the **Numpy** library.
    - Typically used to describe **matrices** and **vectors**.
- The look very similar to **nested lists** of lists, which we have used earlier for representing matrices.

```python
import numpy as np

# Create a matrix as a nested list of lists
matrix1 = [[0,1], [2,3]]
print(type(matrix1))
print(matrix1)

# Create a matrix as a numpy array
matrix2 = np.array([[0,1],[2,3]])
print(type(matrix2))
print(matrix2)
```

```
<class 'list'>
[[0, 1], [2, 3]]
<class 'numpy.ndarray'>
[[0 1]
 [2 3]]
```

Numpy vs lists

- **Numpy arrays** offer more tools for matrix computation.
- For starters, Numpy can **identify oddly-shaped matrices** from the start.
- Many methods from **lists** can be reused, but the numpy library offers more functions/methods.

```python
# Create a matrix as a nested list of lists
matrix3 = [[0,1], [2,3,4]]
print(matrix3)

# Create a matrix as a numpy array
matrix4 = np.array([[0,1], [2,3,4]])
print(matrix4)
```

```
[[0, 1], [2, 3, 4]]
[list([0, 1]) list([2, 3, 4])]
```

Numpy generation: array, zeros, ones

- The Numpy arrays can be generated in multiple ways
- We can pass a nested list of lists detailing the **matrix elements**.
- Or we can generate matrices by specifying **dimensions** and fill it with **zeros** or **ones**.

```
# Create an array with a nested list of lists
matrix1 = np.array([[0,1],[2,3]])
print(matrix1)
```

```
[[0 1]
 [2 3]]
```

```
# Create a matrix full of zeros
matrix2 = np.zeros([4,2])
print(matrix2)
```

```
[[0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]]
```

```
# Create a matrix full of ones
matrix3 = np.ones([2,3])
print(matrix3)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

Numpy generation 2: eye ,linspace, arange

- It is also possible to create square identity matrices with **eye()**
- Or use linspace and a range functions to create vectors with regularly spaced elements.

```python
# Create an identity square matrix
matrix4 = np.eye(3)
print(matrix4)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```python
# - Create a linearly spaced 1D vector
# Linspace(a,b,c) creates a vector with c
# regularly spaced elements, with a as the
# first element and b as the last element
matrix5 = np.linspace(0,10,5)
print(matrix5)
```

```
[ 0.   2.5  5.   7.5 10. ]
```

```python
# - Create a linearly spaced 1D vector
# Arange(a,b,c) creates a vector, with a as the
# first element and c as the spacing between elements.
# b is the last element, but is never included.
matrix6 = np.arange(0,10,2)
print(matrix6)
```

```
[0 2 4 6 8]
```

Some key attributes of Numpy: shape and size

- The Numpy arrays are custom objects with some **attributes.**
- Some interesting attributes are
  - **Shape:** contains a tuple, with the number of rows and columns of the matrix.
  - **Size:** an integer containing the number of elements in the numpy array.

```
# Create a numpy array
matrix = np.array([[0,1,2],[3,4,5]])
print(matrix)

# Number of elements in array
print(matrix.size)

# Matrix dimensions
print(matrix.shape)

# Number of rows
print(matrix.shape[0])

# Number of columns
print(matrix.shape[1])
```

```
[[0 1 2]
 [3 4 5]]
6
(2, 3)
2
3
```

Basic operators: addition and multiplication

- Interestingly, the basic operations on matrices have been implemented in Numpy.
- For instance, the **addition +** works as expected.

```
# Some matrices
matrix1 = np.array([[0,1],[1,0]])
print(matrix1)
matrix2 = np.array([[0,2],[1,-1]])
print(matrix2)
```

```
[[0 1]
 [1 0]]
[[ 0  2]
 [ 1 -1]]
```

```
# Matrix addition
matrix3 = matrix1 + matrix2
print(matrix3)
```

```
[[ 0  3]
 [ 2 -1]]
```

Basic operators: addition and multiplication

- Interestingly, the basic operations on matrices have been implemented in Numpy.
- The **multiplication** * however is an **element-wise** multiplication by default.
- The standard matrix multiplication uses the **dot()** function.

```
# Some matrices
matrix1 = np.array([[0,1],[1,0]])
print(matrix1)
matrix2 = np.array([[0,2],[1,-1]])
print(matrix2)
```

```
[[0 1]
 [1 0]]
[[ 0  2]
 [ 1 -1]]
```

```
# Matrix addition
matrix3 = matrix1 + matrix2
print(matrix3)
```

```
[[ 0  3]
 [ 2 -1]]
```

```
# Element-wise multiplication
matrix4 = matrix1*matrix2
print(matrix4)
```

```
[[0 2]
 [1 0]]
```

```
# "Normal" matrix multiplication
matrix5 = np.dot(matrix1, matrix2)
print(matrix5)
```

```
[[ 1 -1]
 [ 0  2]]
```

Indexing

- As before with nested lists, we can perform **indexing** with **successive brackets**.
- However, we can also use the **bracket** and **comma notation** on Numpy arrays.
  - (and it is often much prefered)

```
# Some matrices
matrix1 = [[0,1],[1,0]]
print(matrix1)
# Element selection in nested lists
element1 = matrix1[0][1]
print(element1)
element1 = matrix1[0,1] # Does not work on lists
print(element1)
```

```
[[0, 1], [1, 0]]
1

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-42-9867793e052b> in <module>
      5 element1 = matrix1[0][1]
      6 print(element1)
----> 7 element1 = matrix1[0,1] # Does not work on lists
      8 print(element1)

TypeError: list indices must be integers or slices, not tuple
```

```
# Some matrix
matrix2 = np.array([[0,2],[1,-1]])
print(matrix2)
# Element selection in numpy arrays
element2 = matrix2[0][1]
print(element2)
element2 = matrix2[0,1] # Does not work on lists
print(element2)
```

```
[[ 0  2]
 [ 1 -1]]
2
2
```

Slicing

- We can also perform **slicing**, as before.
- We can also use the **:** symbol to **select all the elements along a dimension** of the matrix
  - (whole row/column)

```python
# A matrix
matrix = np.array([[0,2],[1,-1]])
print(matrix)
```

```
[[ 0  2]
 [ 1 -1]]
```

```python
# Getting a whole line
index = 1
elements = matrix[index,:]
print(elements)
```

```
[ 1 -1]
```

```python
# Getting a whole column
index = 1
elements = matrix[:,index]
print(elements)
```

```
[ 2 -1]
```

Reshaping into 1D arrays, a.k.a. flattening.

- Sometimes, we might be interested to reshape our matrix into a 1D vector.
- We can do so with the **reshape()** function.
- This operation is typically known as **flattening** a matrix

```
# A matrix
matrix = np.array([[0,1,2], [3,4,5]])
print(matrix)
```

```
[[0 1 2]
 [3 4 5]]
```

```
# Reshaping into a 2D (1 x N) array
matrix2 = np.reshape(matrix, [1,6])
# Or equivalently
matrix2 = np.reshape(matrix, [1,matrix.size])
print(matrix2)
```

```
[[0 1 2 3 4 5]]
```

Reshaping and transposing

- It can also be used to reshape a matrix from a given shape to another one.
- For instance **reshape** a (2,3) matrix into a (3,2) matrix.
- **Important: this is not a transposition operation!**
  - Transposition = **transpose()**

```python
# A matrix
matrix = np.array([[0,1,2], [3,4,5]])
print(matrix)
# Reshaping into another format
matrix3 = np.reshape(matrix, [3,2])
print(matrix3)
```

```
[[0 1 2]
 [3 4 5]]
[[0 1]
 [2 3]
 [4 5]]
```

```python
# Reshaping into another format
matrix4 = np.transpose(matrix)
print(matrix4)
```

```
[[0 3]
 [1 4]
 [2 5]]
```

Eigenvalues and eigenvectors

- Numpy also has lots of functions implementing linear algebra operations, such as
  - Finding **eigenvalues** and **eigenvectors**

```
# A matrix
matrix = np.array([[1,1],[0,-1]])
print(matrix)
```

```
[[ 1  1]
 [ 0 -1]]
```

```
# Find the eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(matrix)
```

```
# Array of eigenvalues
print(eigenvalues)
```

```
[ 1. -1.]
```

```
# Array of eigenvectors
print(eigenvectors)
```

```
[[ 1.         -0.4472136 ]
 [ 0.          0.89442719]]
```

```
# First eigenvector
print(eigenvectors[0])
```

```
[ 1.         -0.4472136]
```

```
# Second eigenvector
print(eigenvectors[1])
```

```
[0.         0.89442719]
```

Linear system solving

- Numpy also has lots of functions implementing linear algebra operations, such as
  - **Solving a linear system of equations**

```
# --- Attempting to solve the linear system below
# 3 x[0] + x[1] = 9
# x[0] + 2 x[1] = 8
# It can be described as Ax = B, with A and B as
A = np.array([[3,1], [1,2]])
B = np.array([9,8])

# Solving linear system
x = np.linalg.solve(A, B)

# Solutions
print(x)
# x[0] solution
print(x[0])
# x[1] solution
print(x[1])
```

```
[2. 3.]
2.0
3.0
```

Min, Max, Mean, Median, Percentile

- It has also functions for finding the
  - **Minimal** value,
  - **Maximal** value,
  - **Mean** value,
  - **Median** values,
  - Etc.
- For any given array, containing random variables realizations.
- Here, we create an array of realizations for a **uniform(0,1) random variable,** with the **random.random()** function.

```python
# Create a nested list of uniformly distributed
# random values between 0 and 1
nested_list = [random.random() for i in range(5)]

# Make it a numpy array
matrix = np.array([nested_list])
print(matrix)
```

```
[[0.22004288 0.53493323 0.73236796 0.37417799 0.6502602 ]]
```

```python
# Create a nested list of uniformly distributed
# random values between 0 and 1
nested_list = [random.random() for i in range(10000)]

# Make it a numpy array
matrix = np.array([nested_list])
print(matrix.shape)
```

Min, Max, Mean, Median, Percentile

- It has also functions for finding the
  - **Minimal** value,
  - **Maximal** value,
  - **Mean** value,
  - **Median** values,
  - Etc.
- For any given array, containing random variables realizations.
- Here, we create an array of realizations for a **uniform(0,1) random variable**, with the **random.random()** function.

```python
# Minimal value
print(np.min(matrix))
```

```
5.052808826566668e-06
```

```python
# Maximal value
print(np.max(matrix))
```

```
0.999945892478096
```

```python
# Mean value
print(np.mean(matrix))
```

```
0.5024344386819765
```

```python
# Median value
print(np.median(matrix))
```

```
0.5042077048148175
```

```python
# n%-percentile value: value of the element,
# which is greater than n% of the samples in matrix
n = 25
print(np.percentile(matrix, n))
```

```
0.2509906081803283
```

Conclusion: Numpy

- Numpy is a powerful library for matrix and vector calculations, typically used in data science.
- More info about its possibilities, here: https://www.numpy.org/devdocs/user/quickstart.html
- Also has some math functions: np.cos, np.sin, np.log, etc.
- Also has some random functions: np.random.randint
- More linear algebra functions: rank of a matrix, determinant of a matrix, inversion of a matrix, diagonalization of a matrix, etc.

A quick introduction to data science

We will see more about these typical problems in Sessions 2 and 3.

A quick introduction to data science

- Data science has been recently trending, with many keyswords…
- But what is the core idea behind this data science concept?

A quick introduction to data science

- Data science has been recently trending, with many keyswords…
- But what is the core idea behind this data science concept?

- **Core ideas**
  - **make sense from data**
  - **and learn information from it.**

Core idea behind data science: find the missing function, based on available data

- What we have done in Programming so far was to design functions,
  - which would do **specific operations**
  - and return **outputs**
  - for any **input** we could give it

Input (given) → Function (Created) → Output (Obtained)

Core idea behind data science: find the missing function, based on available data

- What we have done in Programming so far was to design functions,
  - which would do **specific operations**
  - and return **outputs**
  - for any **input** we could give it

Input (given) → Function (Created) → Output (Obtained)

- But sometimes, we can encounter problems where
  - we can easily find **inputs** and **expected** outputs,
  - but the **function** to be coded is **not simple** to figure out.

Input (available) → Function (?) → Output (available)

Core idea behind data science: find the missing function, based on available data

- What we have done in Programming so far was to design functions,
  - which would do **specific operations**
  - and return **outputs**
  - for any **input** we could give it

Input (given) → Function (Created) → Output (Obtained)

- But sometimes, we can encounter problems where
  - we can easily find **inputs** and **expected** outputs,
  - but the **function** to be coded is **not simple** to figure out.
- **Idea: What if the computer could learn the function on its own?**

Input (available) → Function (?) → Output (available)

A typical example

- For instance, if I were to give you this table of values…

Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25
7	49
8	64
9	81
10	100

A typical example

- For instance, if I were to give you this table of values…
- And then ask you to guess the expected output for the value **6**…

Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25
6	**?**
7	49
8	64
9	81
10	100

A typical example

- For instance, if I were to give you this table of values…
- And then ask you to guess the expected output for the value **6**…
- You would probably guess, it is **36**.

Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25
6	**?**
7	49
8	64
9	81
10	100

A typical example

- For instance, if I were to give you this table of values…
- And then ask you to guess the expected output for the value **6**…
- You would probably guess, it is **36**.
- Because you guessed, that the missing function $y = f(x)$, was $f(x) = x^2$.
- And $f(6) = 36$.

Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25
6	**?**
7	49
8	64
9	81
10	100

Record/Experience, features/inputs and labels/outputs

- What just happened?

Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25
7	49
8	64
9	81
10	100
6	**?**

Record/Experience, features/inputs and labels/outputs

- What just happened?
- You used your previous **experience/record**

Experience/Record

Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25
7	49
8	64
9	81
10	100
6	**?**

Record/Experience, features/inputs and labels/outputs

- What just happened?
- You used your previous **experience/record**
- To "guess" what might be the **relationship/function f**
  - Between your **inputs/features x**
  - And their respective **outputs/labels y**

Experience/Record

Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25
7	49
8	64
9	81
10	100
6	**?**

Input x (available) → Function f (?) → Output y = f(x) (available)

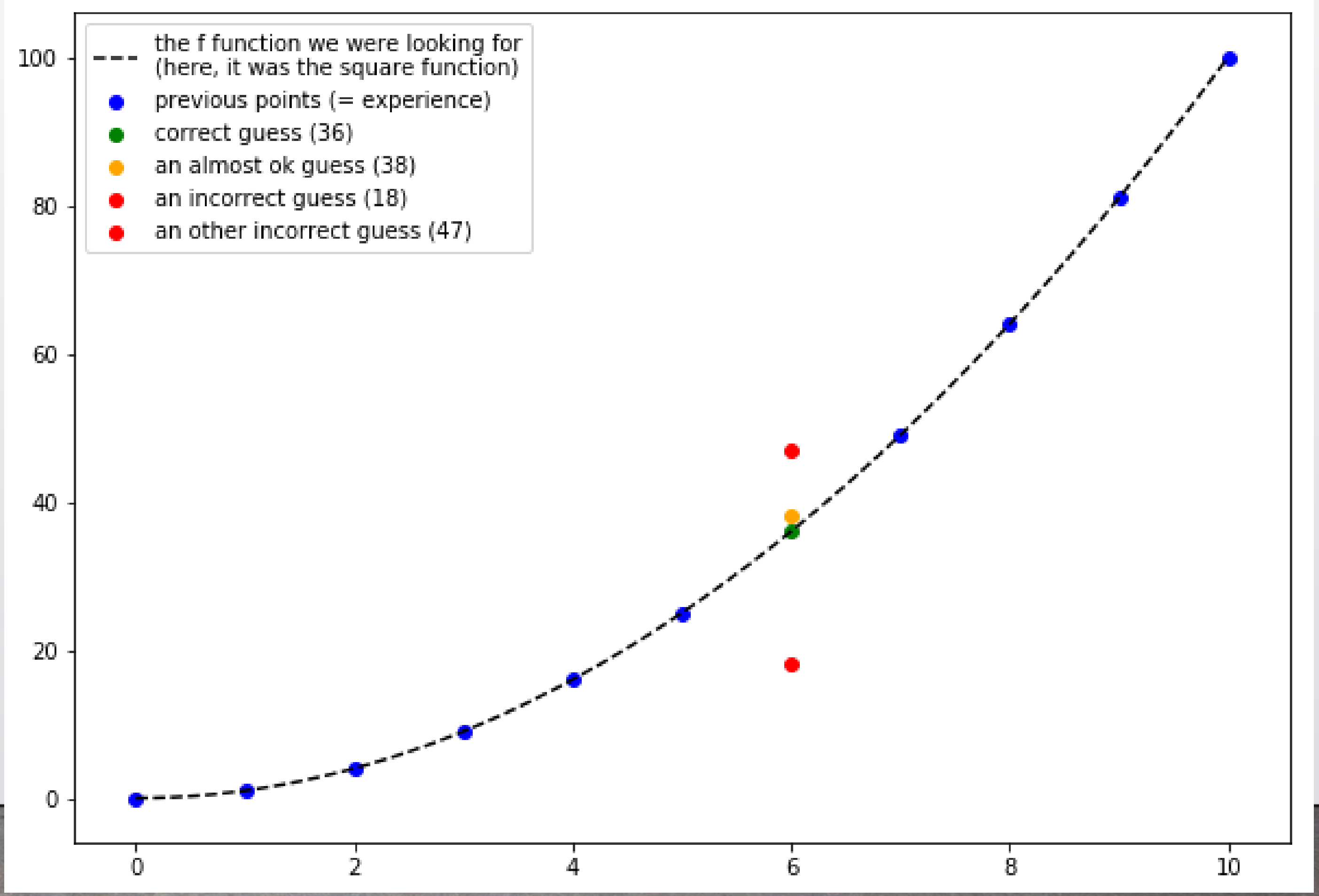
About regression

- That is a very common problem in data science, called **regression**.

About regression

```
x = [0, 1, 2, 3, 4, 5, 7, 8, 9, 10]
y = [0, 1, 4, 9, 16, 25, 49, 64, 81, 100]
```

- That is a very common problem in data science, called **regression**.
- Mathematically speaking, it consists of **finding the curve** that covers the points **(x,y)** you have in your **record/experience**.
- So that you could later **predict** the **outputs** of **unseen input values**.



Typical problems in regression

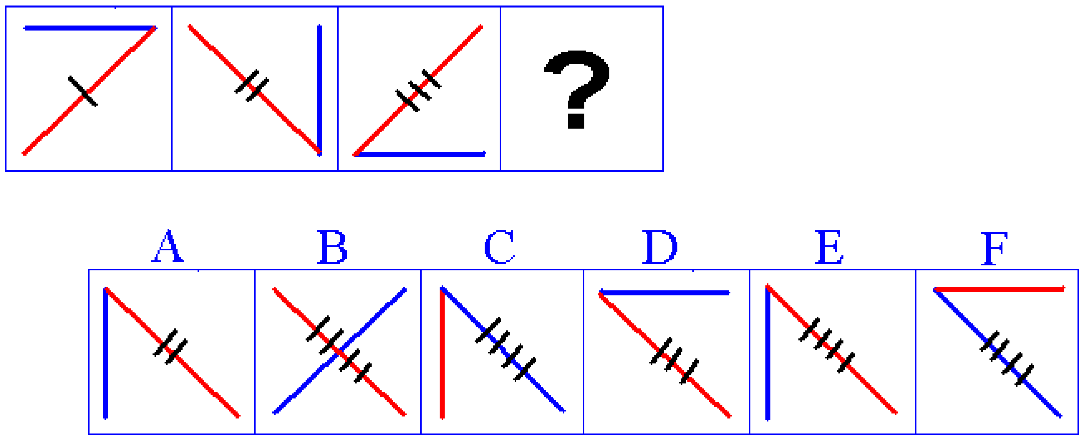
- Typically, our squares example

Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25
7	49
8	64
9	81
10	100
6	**?**

Typical problems in regression

Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25

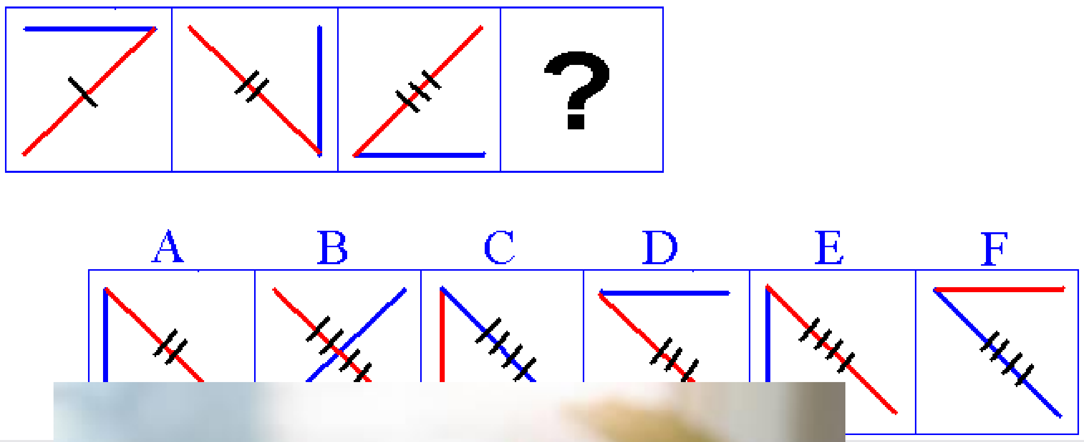
- Typically, our squares example
- The IQ tests: « guess the element that comes next in the sequence »



Typical problems in regression

Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25

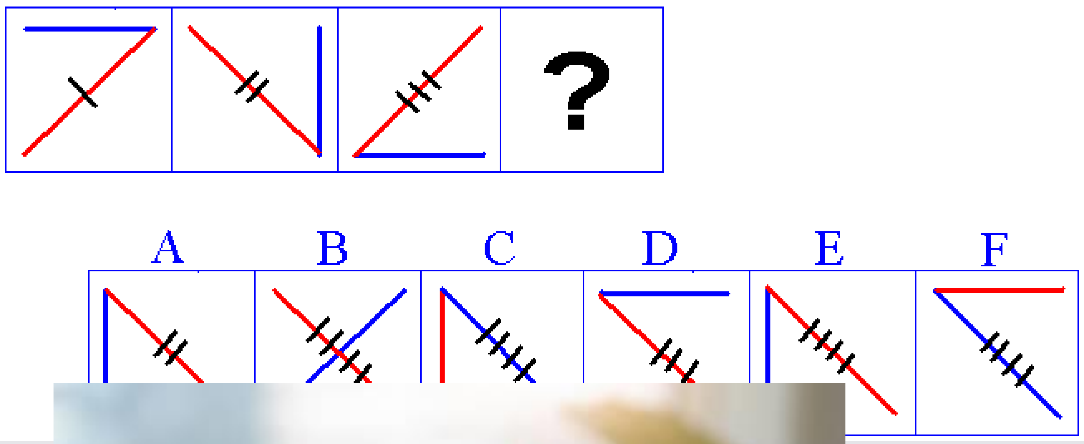
- Typically, our squares example
- The IQ tests: « guess the element that comes next in the sequence »
- Exemple with appartment selling
  - Guessing the selling price of an appartment based on its size and your previous sales.



Typical problems in regression

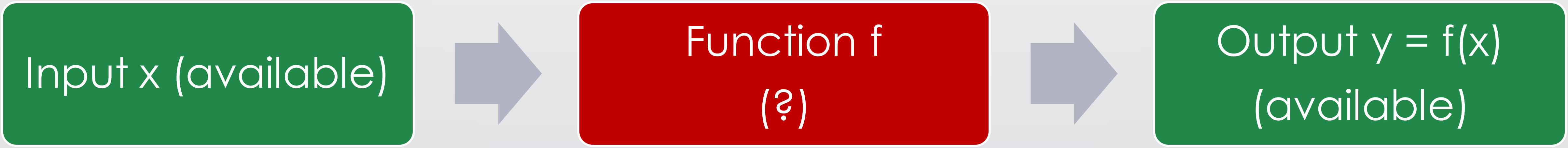
Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25

- Typically, our squares example
- The IQ tests: « guess the element that comes next in the sequence »
- Exemple with appartment selling
  - Guessing the selling price of an appartment based on its size and your previous sales.
- Etc.



Regression and classification

- Regression problems are very common in real-life.
- Other very common problems are classification ones.



Regression and classification

- Regression problems are very common in real-life.
- Other very common problems are classification ones.
- Typically used in computer vision.

Input x (available) → Function f (?) → Output y = f(x) (available)

Very easy for a human…

It's a cat!

Regression and classification

- Regression problems are very common in real-life.
- Other very common problems are classification ones.
- Typically used in computer vision.

Input x (available) → Function f (?) → Output y = f(x) (available)

Very easy for a human…
How would we do it with a computer?

It's a cat!

Classification are very common in computer vision

- Computer vision problems:
  - Image recognition: given a picture, tell me what it is (cats/dogs, name of the person)

It’s a cat!

Classification are very common in computer vision

- Computer vision problems:
  - Image recognition: given a picture, tell me what it is (cats/dogs, name of the person)
  - Image classification: given a CT scan, tell me if there is a cancer/not cancer

That's cancerous

Classification are very common in computer vision

- Computer vision problems:
  - Image recognition: given a picture, tell me what it is (cats/dogs, name of the person)
  - Image classification: given a CT scan, tell me if there is a cancer/no cancer
  - Image recognition + segmentation:
    - find if there is a pedestrian in the picture
    - and if so, where he/she is,
    - And what its movement is.

Recap for data science

- Many real-life problems
  - will fall in either the regression or the classification category
  - and can be addressed with data science based approaches.

- We will cover a typical regression problem on Session 2.
- And a typical classification problem on Session 3.

Recap for data science

- Many real-life problems
  - will fall in either the regression or the classification category
  - and can be addressed with data science based approaches.

- We will cover a typical regression problem on Session 2.
- And a typical classification problem on Session 3.

- But before we do so, a quick word about **data visualization.**

About data visualization

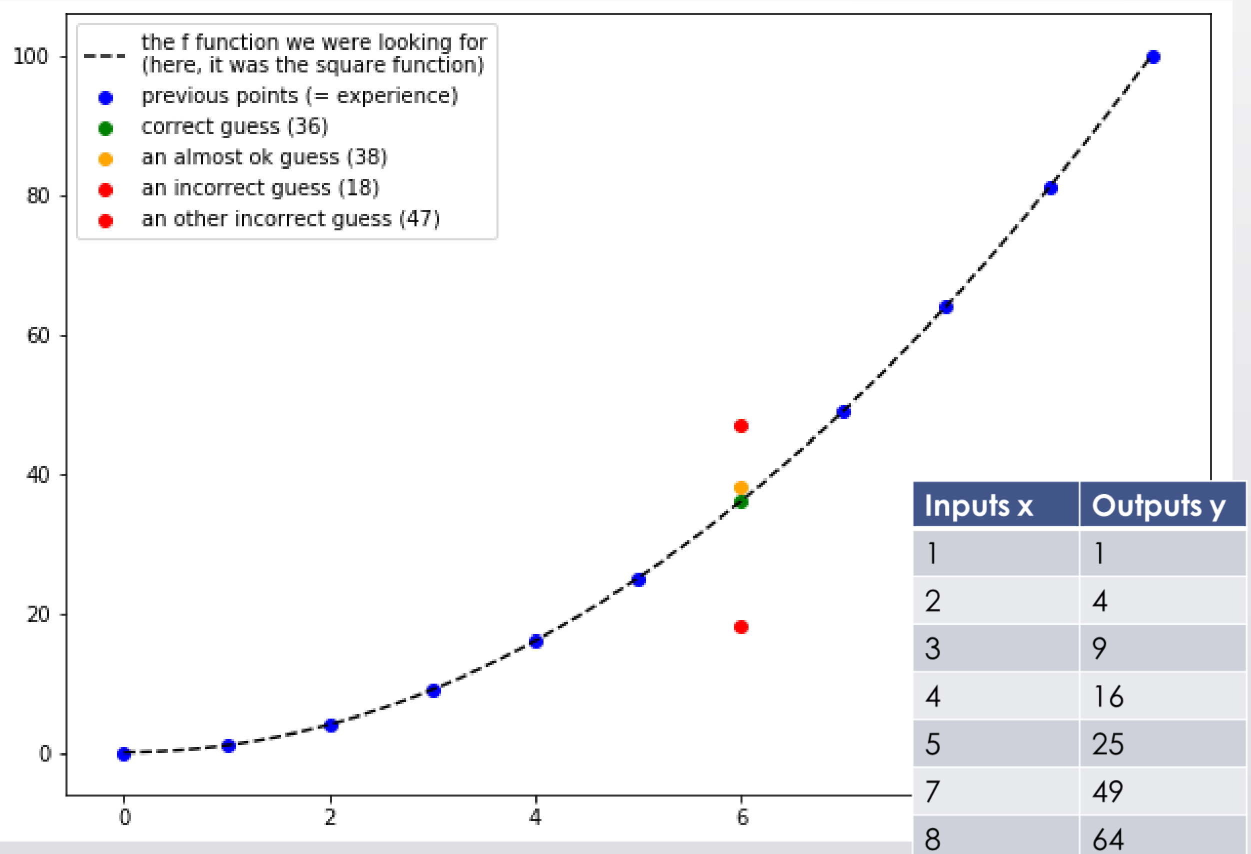
Looking for ideas, by exploring your data.

Seeking inspiration by visualizing data

- Often, in data science problems, we like to plot the data to see what the problem looks like.

Seeking inspiration by visualizing data

- Often, in data science problems, we like to plot the data to see what the problem looks like.
- For instance, we could have plotted the points of the square problem discussed earlier…
- And recognized that the missing function we were looking for was most likely a quadratic function of some sort.



Inputs x	Outputs y
1	1
2	4
3	9
4	16
5	25
7	49
8	64
9	81
10	100
6	**?**

Seeking inspiration by visualizing data

- Let us say I have a table of previous appartment sales…
- Containing a list of appartments I have sold in the past, with their size (in sqm) and their price (in $)

Size (in sqm)	Price (in $)
57	544000
76	760000
92	947000
101	1006000
101	1049000
106	1037000
…	…

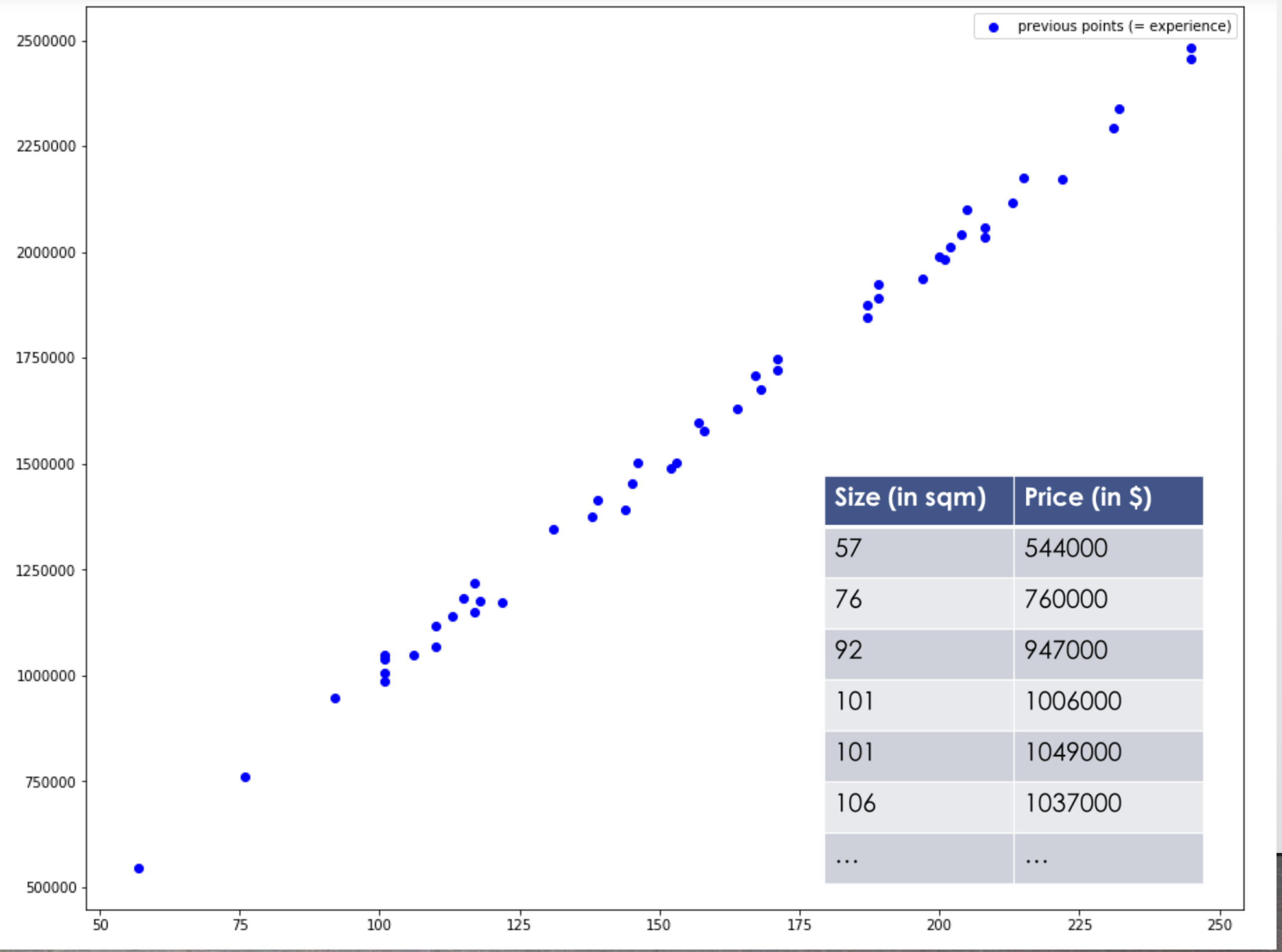
Seeking inspiration by visualizing data

- Let us say I have a table of previous appartment sales…
- Containing a list of appartments I have sold in the past, with their size (in sqm) and their price (in $).
- **And friend comes and asks for an estimation of the price of its 125sqm appartment. What is your answer?**

Size (in sqm)	Price (in $)
57	544000
76	760000
92	947000
101	1006000
101	1049000
106	1037000
…	…

Seeking inspiration by visualizing data

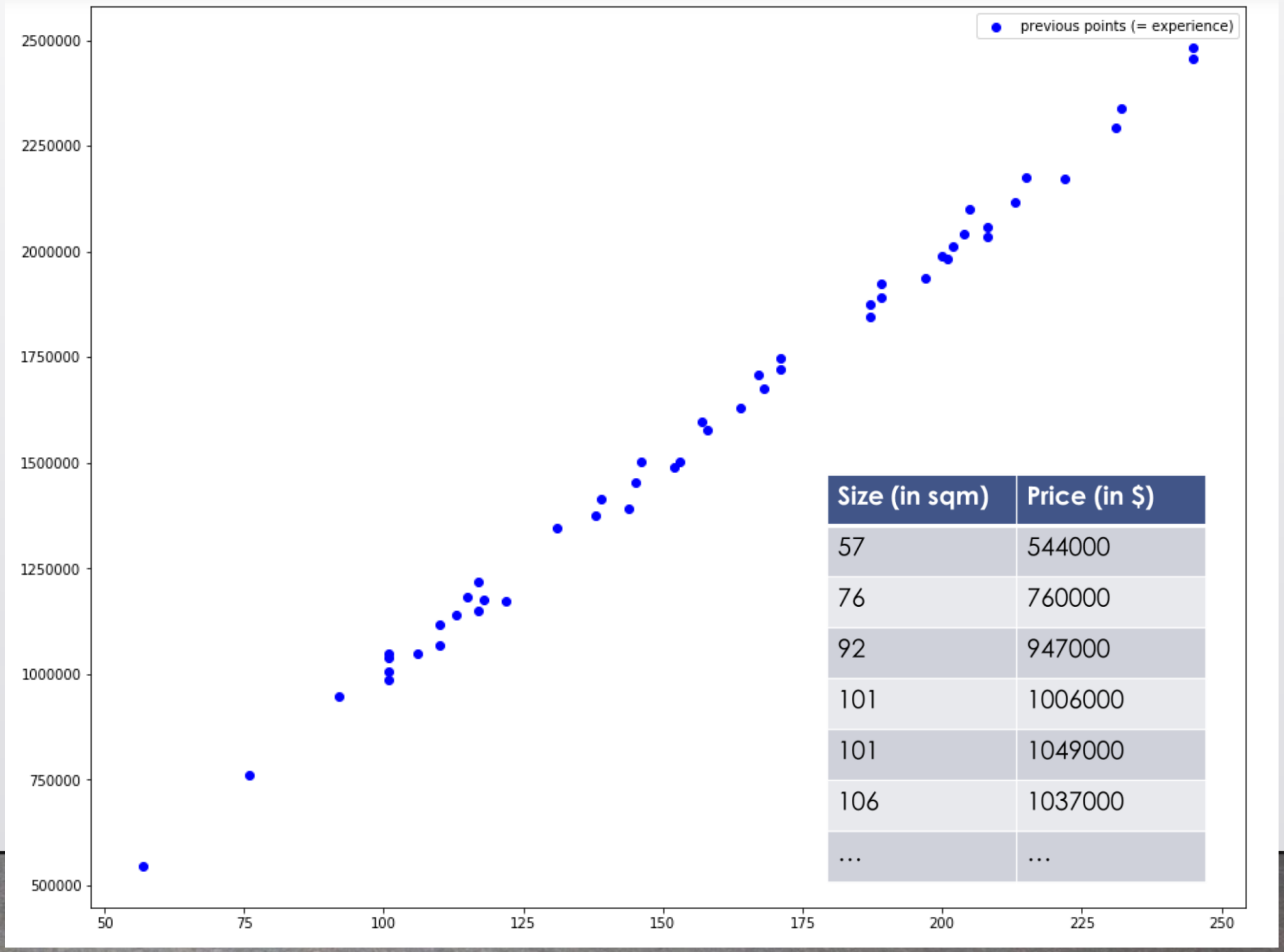
- Let us say I have a table of previous appartment sales…
- Containing a list of appartments I have sold in the past, with their size (in sqm) and their price (in $)
- **And friend comes and asks for an estimation of the price of its 125sqm appartment. What is your answer?**



Size (in sqm)	Price (in $)
57	544000
76	760000
92	947000
101	1006000
101	1049000
106	1037000
…	…

Scatter plots

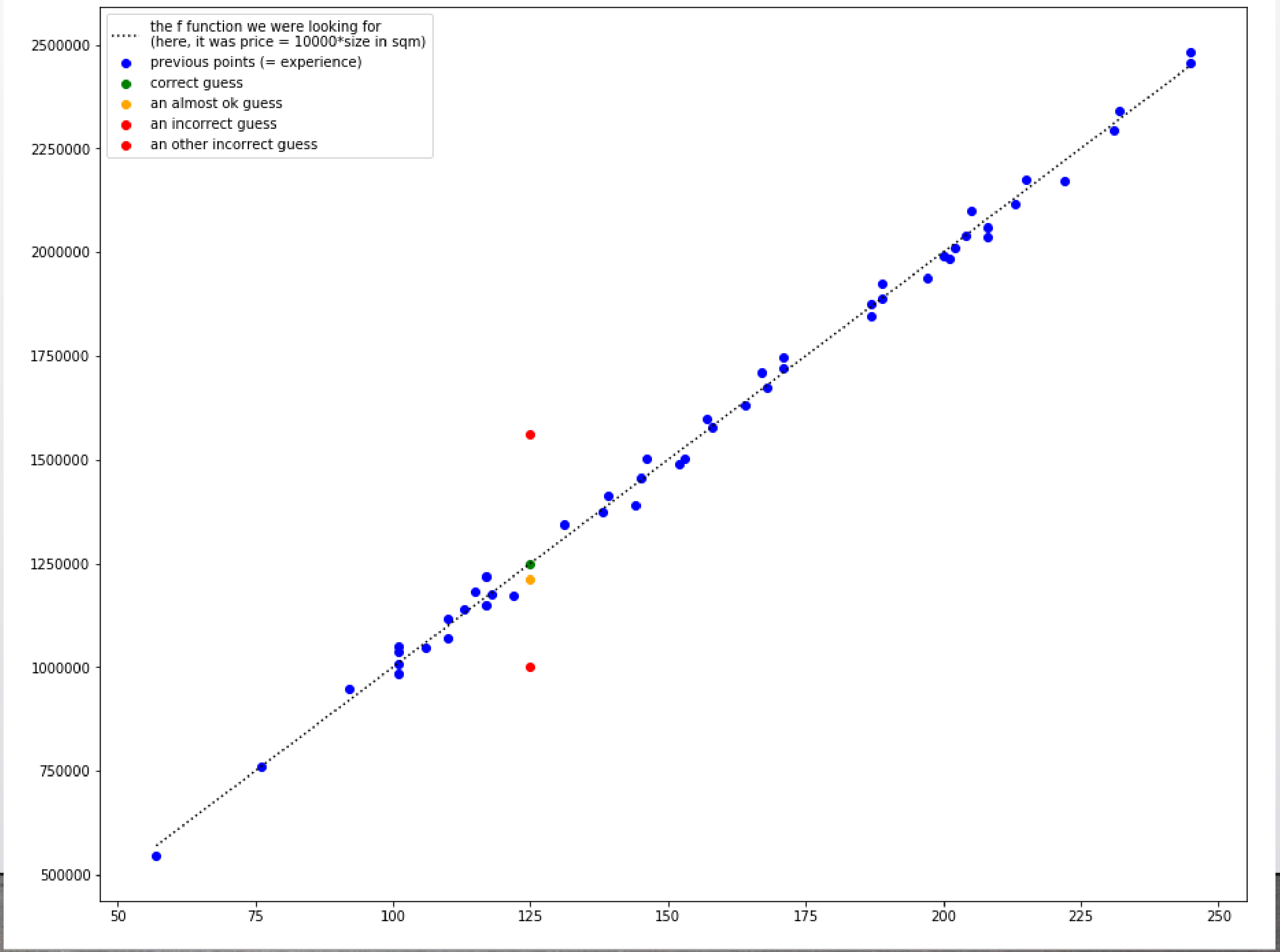
- Scatter plots are often useful.
- They give rough insights as to what the function linking my inputs and outputs might be.



Size (in sqm)	Price (in $)
57	544000
76	760000
92	947000
101	1006000
101	1049000
106	1037000
…	…

Scatter plots

- Scatter plots are often useful.
- They give rough insights as to what the function linking my inputs and outputs might be.



Using data descriptors

- Data descriptors are also very useful.
- They give a numerical recap of key values of the data (its min, max, mean, median, etc.)
- Data objects often have built-in descriptors, but their name may vary.

```python
# part b
bunchobject = datasets.load_breast_cancer()
print(bunchobject.DESCR)
print(bunchobject.feature_names)
print(bunchobject.target_names)
print(bunchobject.data.shape)
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
```

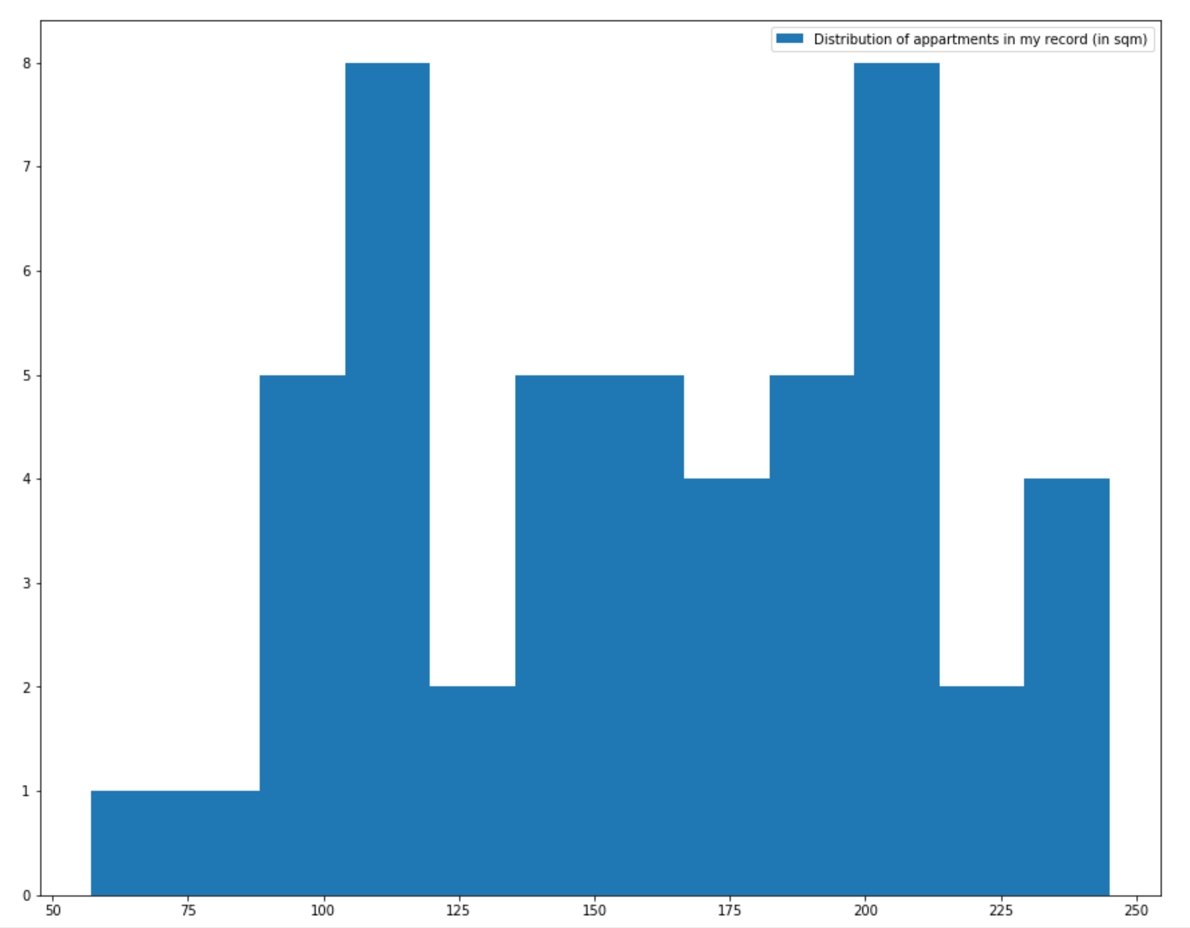
Seeking inspiration by visualizing data

- Let us say I have a table of previous appartment sales…
- Containing a list of appartments I have sold in the past, with their size (in sqm) and their price (in $).
- **And friend comes and asks for an estimation of the price of its 125sqm appartment. What is your answer?**

Size (in sqm)	Price (in $)
57	544000
76	760000
92	947000
101	1006000
101	1049000
106	1037000
…	…

Histogram plots

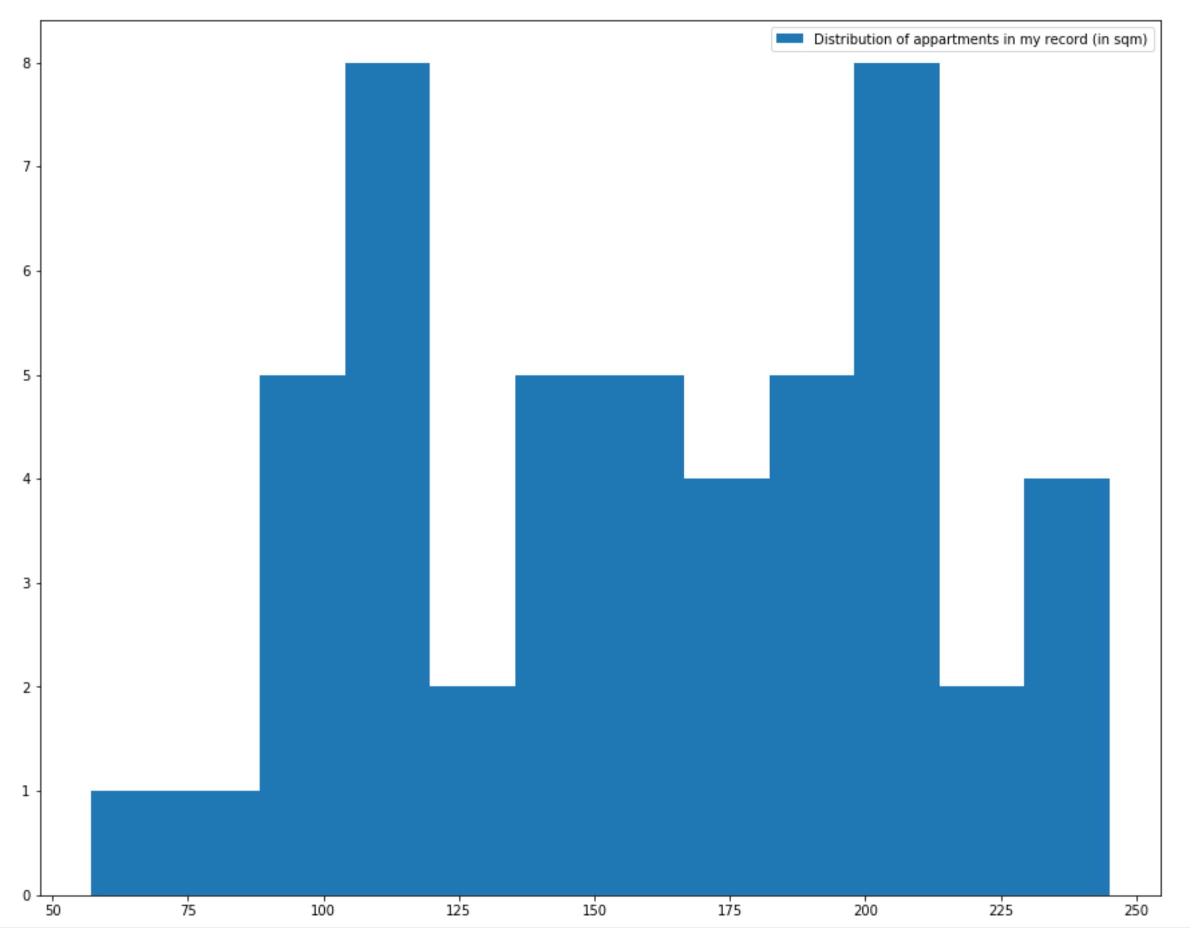
- Histograms are also useful, in order to visualize the distribution of the appartments you have in your record.



Histogram plots

- Histograms are also useful, in order to visualize the distribution of the appartments you have in your record.
- Price for a **110sqm** appartment?

→ Confident, I have seen a lot of those



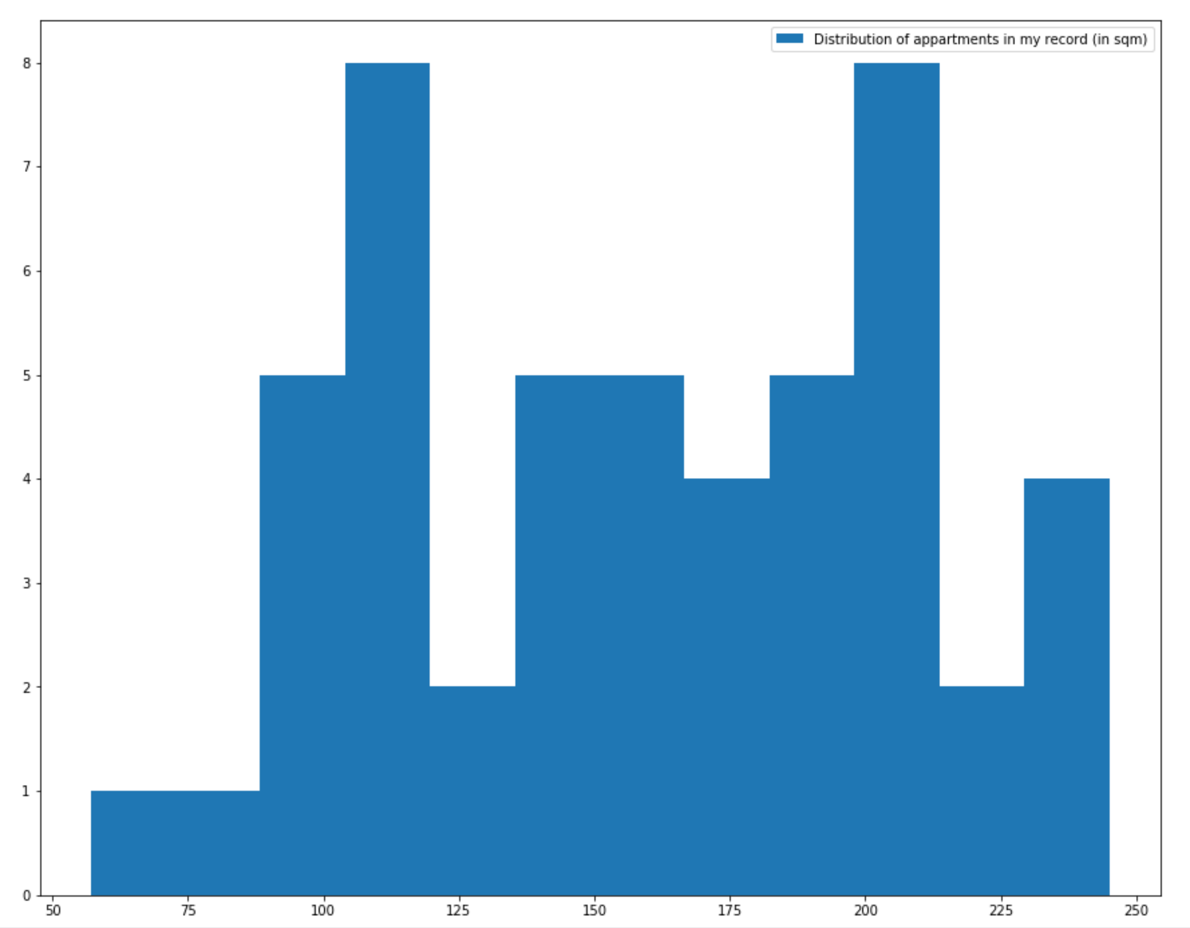
Histogram plots

- Histograms are also useful, in order to visualize the distribution of the appartments you have in your record.
- Price for a **110sqm** appartment?

→ Confident, I have seen a lot of those

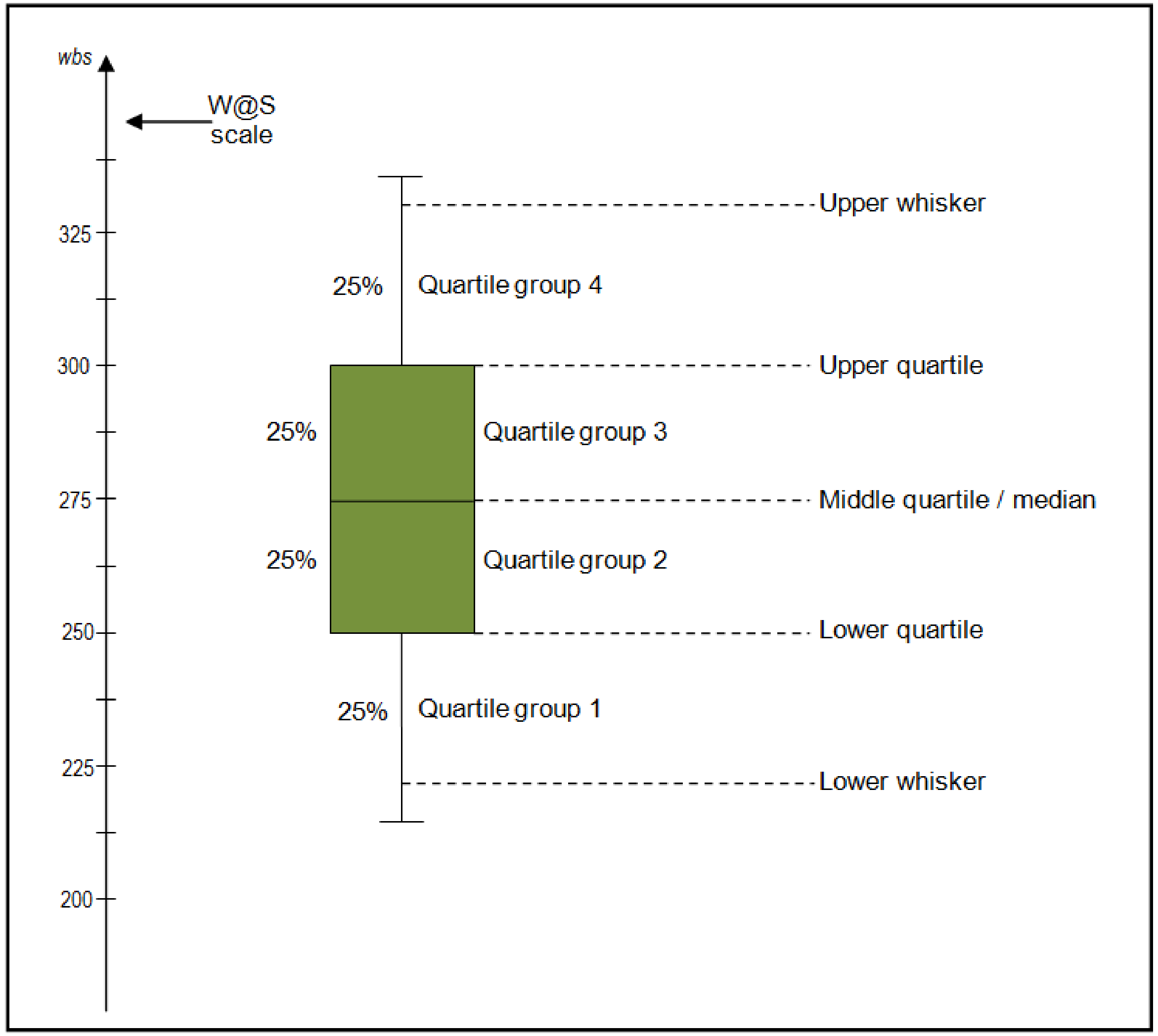
- Price for a **500sqm** appartment?

→ Have never seen those before…



Boxplots

- Boxplots can also be used to get information about the data distribution.
- They more or less do the same thing as the histogram plots and five-number summaries.



Let us practice a bit

Problem set 10 – Q1, Q2 & Q3 (confusion matrix, 5-number summary and normalization)

Q1: Confusion matrix and precision metrics

- In this activity, let us assume we have designed a computer vision AI, attempting to recognize images of birds/cats.
- We have two lists.
  - The first one, named actual, contains what really is in the image (bird or cat)
  - The second one, named predicted, contains what our AI identified in the images.
- Step 1: define a confusion matrix, listing the number of right guesses and mistakes, as described in Q1.
- Step 2: define some key precision metrics (recall, accuracy, false positive rate) for our AI.

Let us practice a bit

Problem set 10 – Q1, Q2 & Q3 (confusion matrix, 5-number summary and normalization)

Q2: Five-number summary

- The **five-number summary**, is an informative function about data, listing
  - The **minimal** value in a given array
  - The **maximal** value in a given array
  - The **median** value in a given array
  - The **first quarter percentile** value in a given array
  - The **third quarter percentile** value in a given array

For Q2: Min, Max, Mean, Median, Percentile

- Numpy has functions for finding the
  - **Minimal** value,
  - **Maximal** value,
  - **Mean** value,
  - **Median** values,
  - Etc.
- For any given array, containing data.

```python
# Minimal value
print(np.min(matrix))
```

```
5.052808826566668e-06
```

```python
# Maximal value
print(np.max(matrix))
```

```
0.999945892478096
```

```python
# Mean value
print(np.mean(matrix))
```

```
0.5024344386819765
```

```python
# Median value
print(np.median(matrix))
```

```
0.5042077048148175
```

```python
# n%-percentile value: value of the element,
# which is greater than n% of the samples in matrix
n = 25
print(np.percentile(matrix, n))
```

```
0.2509906081803283
```

Let us practice a bit

Problem set 10 – Q1, Q2 & Q3 (confusion matrix, 5-number summary and normalization)

Q3: data normalization

- Data normalization is a typical operation in Machine Learning.
  - It re-scales the data, so that the minimal value in the data will become 0.
  - And the maximal value will become 1.

Input x1	Input x2
1	10
2	6
3	2
4	4
5	0

Input x1 (normalized)	Input x2 (normalized)
0	1
0.25	0.6
0.5	0.2
0.75	0.4
1	0

Q3: data normalization

- Data normalization is a typical operation in Machine Learning.
    - It re-scales the data, so that the minimal value in the data will become 0.
    - And the maximal value will become 1.
- Q3: write a function that receives a data array, and normalize the columns of the array one-by-one.

Input x1	Input x2
1	10
2	6
3	2
4	4
5	0

Input x1 (normalized)	Input x2 (normalized)
0	1
0.25	0.6
0.5	0.2
0.75	0.4
1	0

Conclusion

- Data manipulation and visualization is the first thing to do when encountering a data science problem.
- It usually gives us good insights
  - as to what the data consists of,
  - how the data is distributed,
  - and sometimes, even gives us a clear linear/polynomial trend that we can reuse!
- On the next session, we will discuss linear/polynomial regression and classification problems.