Fundamentals of Machine Learning
Python for Data Science – Part 2

Kien C Nguyen

11 July, 2020

Contents

1 Numpy

2 Matplotlib

3 Pandas

Numpy [1]

- Numpy is the core library for scientific computing in Python.
- It provides a high-performance multidimensional array object, and tools for working with these arrays.

```
import numpy as np
```

Numpy array [1]

- A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers.
- The number of dimensions is the rank of the array
- The shape of an array is a tuple of integers giving the size of the array along each dimension.

```
[2]  1 a = np.array([1, 2, 3])  # Create a rank 1 array
     2 print(type(a), a.shape, a[0], a[1], a[2])
     3 a[0] = 5                 # Change an element of the array
     4 print(a)
```

```
<class 'numpy.ndarray'> (3,) 1 2 3
[5 2 3]
```

```
[3]  1 b = np.array([[1,2,3],[4,5,6]]) # Create a rank 2 array
     2 print(b.shape)
     3 print(b)
```

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

Creating an array

```
[7]  1 a = np.zeros((2,2))  # Create an array of all zeros
     2 print(a)
```

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

```
[8]  1 b = np.ones((1,2))   # Create an array of all ones
     2 print(b)
```

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

```
[9]  1 c = np.full((2,2), 7) # Create a constant array
     2 print(c)
```

```
[[7 7]
 [7 7]]
```

```
[10]  1 d = np.eye(2)        # Create a 2x2 identity matrix
      2 print(d)
```

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

```
[11]  1 e = np.random.random((2,2)) # Create an array filled with random values
      2 print(e)
```

```
[[0.86679868 0.46266196]
 [0.87642044 0.00682859]]
```

Array indexing

```
[3]  1 b = np.array([[1,2,3],[4,5,6]]) # Create a rank 2 array
     2 print(b.shape)
     3 print(b)
```

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

```
[4]  1 # Array Indexing
     2 #Indexing a rank 2 array could be written in two different ways
     3 # (and there are many to come in the section below)
     4 print(b[0, 0], b[0, 1], b[1, 0])
     5 print(b[0][0], b[0][1], b[1][0])
```

```
1 2 4
1 2 4
```

```
[5]  1 c = np.array([[[1,2,3], [4,5,6]], [[7,8,9], [10, 11, 12]]]) # Create a rank 3 array
     2 print(c.shape)
     3 print(c)
```

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

 [[ 7  8  9]
  [10 11 12]]]
```

```
[6]  1 # Indexing a rank 3 array is pretty similar to that of rank 2 array
     2 print(c[0, 0, 0], c[0, 1, 0], c[1, 0, 1])
     3 print(c[0][0][0], c[0][1][0], c[1][0][1])
```

```
1 4 8
1 4 8
```

Array indexing

```
[12]  1 # Create the following rank 2 array with shape (3, 4)
      2 # [[ 1  2  3  4]
      3 #  [ 5  6  7  8]
      4 #  [ 9 10 11 12]]
      5 a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
      6
      7 # Use slicing to pull out the subarray consisting of the first 2 rows
      8 # and columns 1 and 2; b is the following array of shape (2, 2):
      9 # [[2 3]
     10 #  [6 7]]
     11 b = a[:2, 1:3]
     12 print(b)
```

```
[[2 3]
 [6 7]]
```

Array math

- numpy.add()
- numpy.subtract()
- numpy.multiply()
- numpy.divide()
- numpy.dot()
- numpy.sqrt()
- . . .

Array math

```
[25]  1 x = np.array([[1,2],[3,4]], dtype=np.float64)
      2 y = np.array([[5,6],[7,8]], dtype=np.float64)
      3
      4 # Elementwise sum; both produce the array
      5 print(x + y)
      6 print(np.add(x, y))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
```

```
[26]  1 # Elementwise difference; both produce the array
      2 print(x - y)
      3 print(np.subtract(x, y))
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

Array math

```
[27]  1 # Elementwise product; both produce the array
      2 print(x * y)
      3 print(np.multiply(x, y))
```

```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
```

```
[28]  1 # Elementwise division; both produce the array
      2 # [[ 0.2         0.33333333]
      3 #  [ 0.42857143  0.5       ]]
      4 print(x / y)
      5 print(np.divide(x, y))
```

```
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

```
[29]  1 # Elementwise square root; produces the array
      2 # [[ 1.          1.41421356]
      3 #  [ 1.73205081  2.        ]]
      4 print(np.sqrt(x))
```

```
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

Contents

1 Numpy

2 Matplotlib

3 Pandas

Why visualization

- Graphs help us understand data, especially qualitative aspects of data, possibly more quickly and easily
- Graphs are a powerful tool to summarize data
- Graphs help us identify patterns, spot outliers, detect corrupt data
- Graphs help us see the relationships among features, and between features and labels, therefore help us in model selection and feature selection.

Matplotlib

- Matplotlib is a popular plotting library for Python
- The matplotlib provides a context, one in which one or more plots can be drawn before the image is shown or saved to file. The context can be accessed via functions on pyplot. The context can be imported as follows:

```
import matplotlib import pyplot
```

- There is some convention to import this context and name it *plt*

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

Line plots

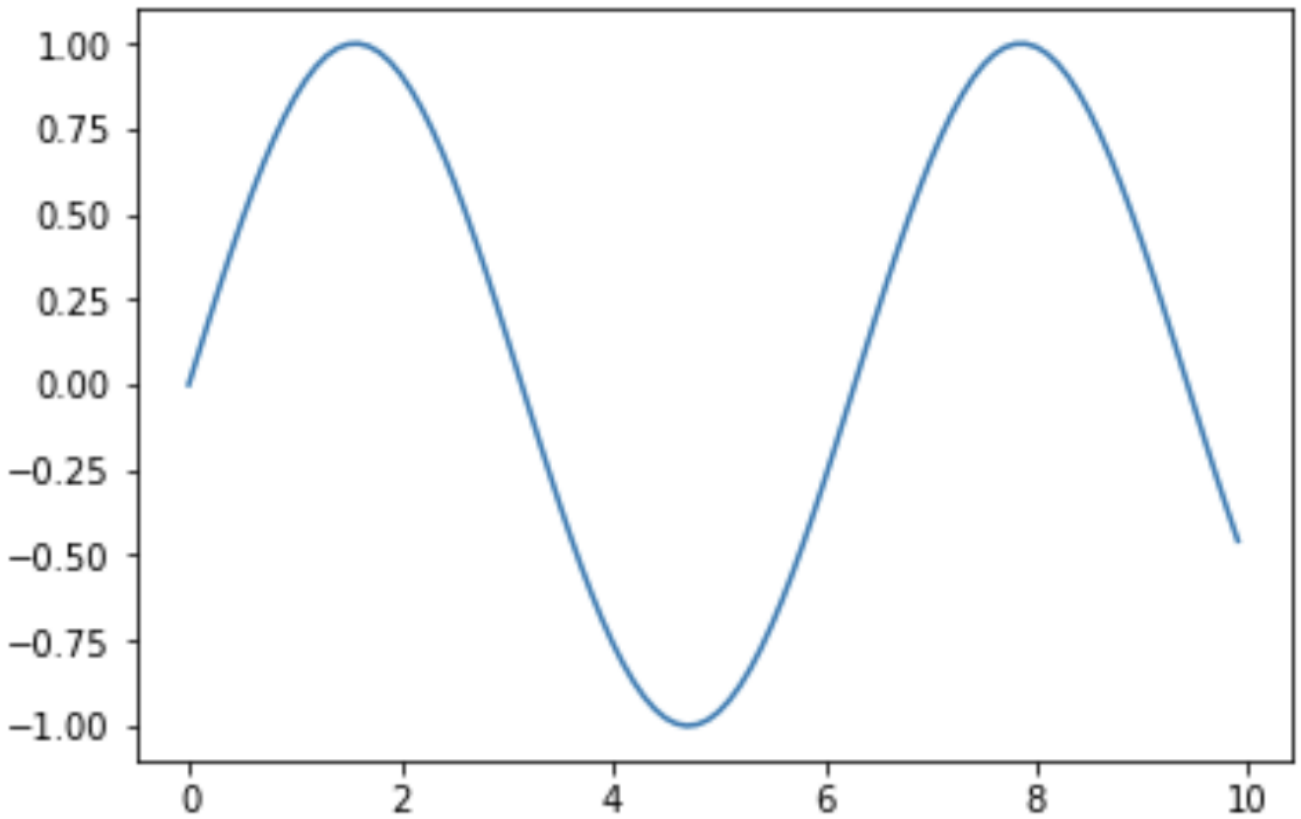
- A line plot is generally used to present observations collected at regular intervals.
- The x-axis represents the regular interval, such as time.
- The y-axis shows the observations, ordered by the x-axis and connected by a line.
- A line plot can be created by calling the plot() function and passing the x-axis data for the regular interval, and y-axis for the observations.

```
# create line plot
pyplot.plot(x, y)
```

Line plots – Plotting a sine

```
import numpy as np
import matplotlib.pyplot as plt
# consistent interval for x-axis
x = [x*0.1 for x in range(100)]
# function of x for y-axis
y = np.sin(x)
# create line plot
plt.plot(x, y)
# show line plot
plt.show()
```

Line plots – Plotting a sine



Bar Chart

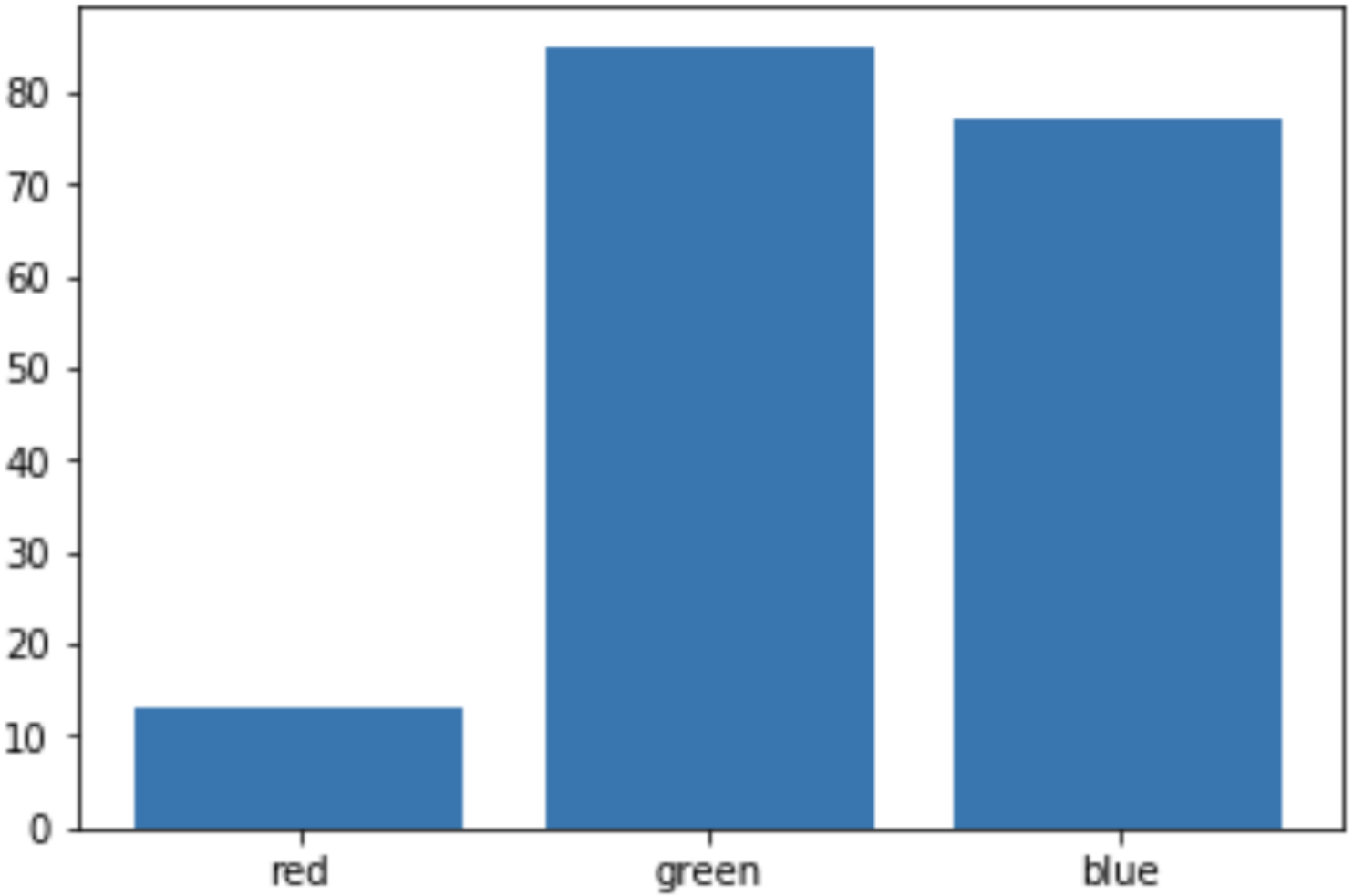
- A bar chart is generally used to present relative quantities for multiple categories.
- The x-axis represents the categories and are spaced evenly.
- The y-axis represents the quantity for each category and is drawn as a bar from the baseline to the appropriate level on the y-axis.
- A bar chart can be created by calling the bar() function and passing the category names for the x-axis and the quantities for the y-axis.
- To create a bar chart

```
# create bar chart
pyplot.bar(x, y)
```

Bar chart – Example

```
# example of a bar chart
from random import seed
from random import randint
from matplotlib import pyplot
# seed the random number generator
seed(1)
# names for categories
x = ['red', 'green', 'blue']
# quantities for each category
y = [randint(0, 100), randint(0, 100), randint(0, 100)]
# create bar chart
pyplot.bar(x, y)
# show line plot
pyplot.show()
```

Bar chart – Example



Histogram Plot

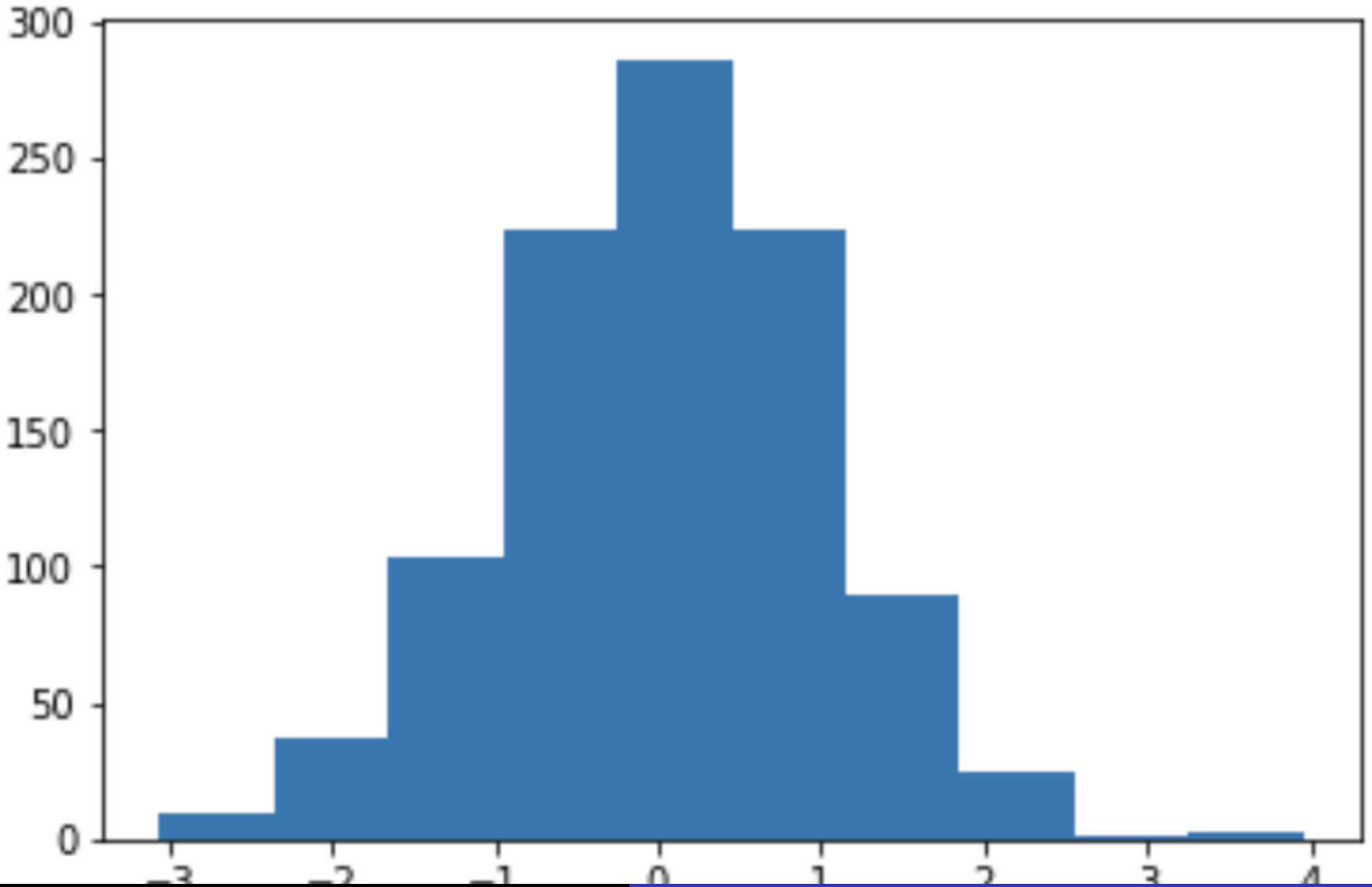
- A histogram plot is generally used to summarize the distribution of a data sample.
- The x-axis represents discrete bins or intervals for the observations.
- For example observations with values between 1 and 10 may be split into five bins, the values [1,2] would be allocated to the first bin, [3,4] would be allocated to the second bin, and so on.
- The y-axis represents the frequency or count of the number of observations in the dataset that belong to each bin.
- To create a Histogram Plot

```
# create histogram plot
pyplot.hist(x)
```

Histogram Plot – Example

```
# example of a histogram plot
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# random numbers drawn from a Gaussian distribution
x = randn(1000)
# create histogram plot
pyplot.hist(x)
# show line plot
pyplot.show()
```

Histogram Plot – Example



Box and Whisker Plot

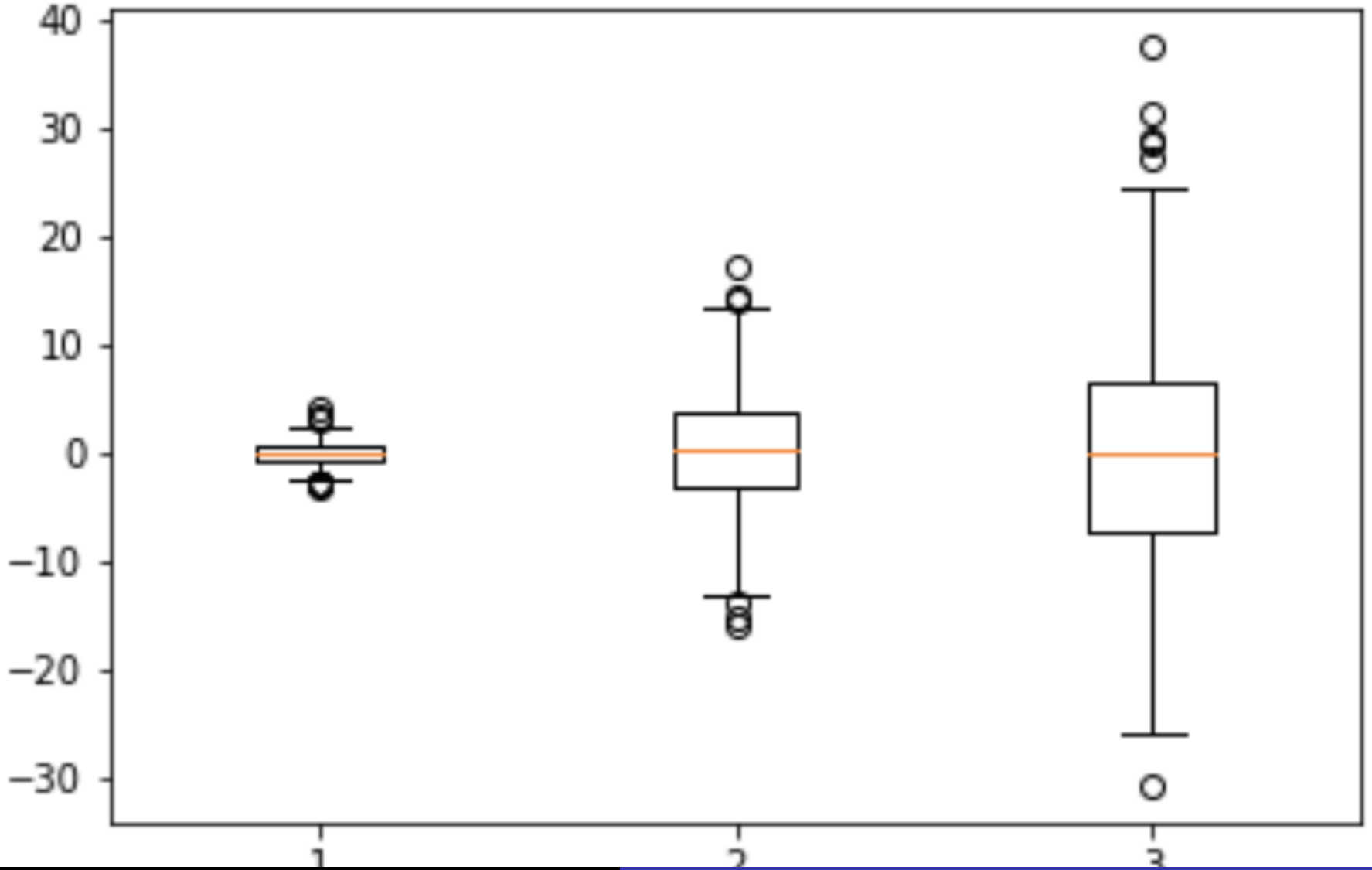
- A box and whisker plot, or boxplot for short, is generally used to summarize the distribution of a data sample.
- The x-axis is used to represent the data sample, where multiple boxplots can be drawn side by side on the x-axis if desired.
- The y-axis represents the observation values.
- A box is drawn to summarize the middle 50
- The median, or 50th percentile, is drawn with a line.
- A value called the interquartile range, or IQR, is calculated as 1.5 * the difference between the 75th and 25th percentiles.
- Lines called whiskers are drawn extending from both ends of the box with the length of the IQR to demonstrate the expected range of sensible values in the distribution.
- Observations outside the whiskers might be outliers and are drawn with small circles.
- To create a Box and Whisker Plot

```
pyplot.boxplot(x)
```

Box and Whisker Plot – Example

```
# example of a box and whisker plot
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# random numbers drawn from a Gaussian distribution
x = [randn(1000), 5 * randn(1000), 10 * randn(1000)]
# create box and whisker plot
pyplot.boxplot(x)
# show line plot
pyplot.show()
```

Box and Whisker Plot – Example



Scatter Plot

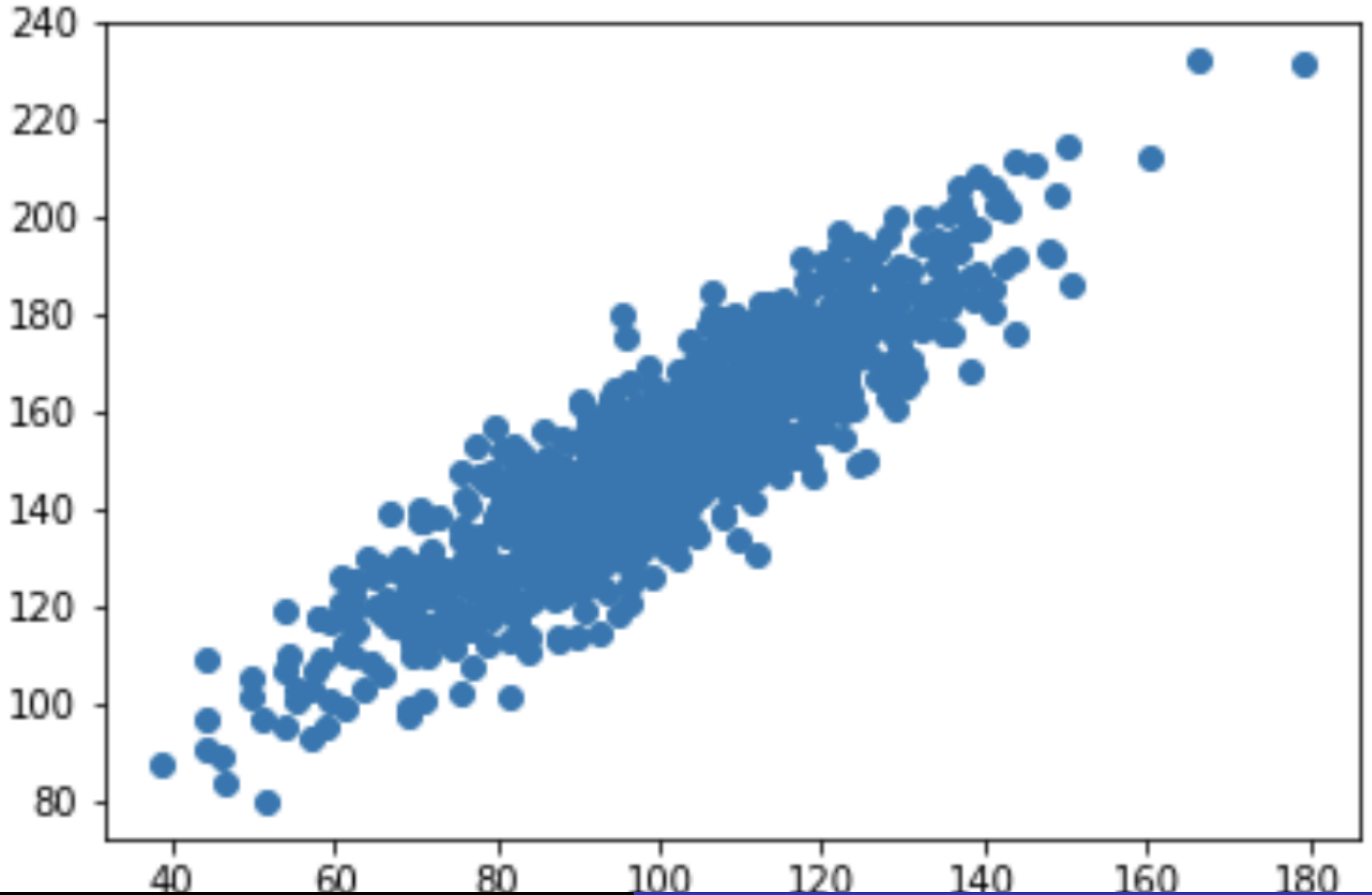
- A scatter plot (or 'scatterplot') is generally used to summarize the relationship between two paired data samples.
- Paired data samples means that two measures were recorded for a given observation, such as the weight and height of a person.
- The x-axis represents observation values for the first sample, and the y-axis represents the observation values for the second sample. Each point on the plot represents a single observation.
- To create a Scatter Plot

```
# create scatter plot
pyplot.scatter(x, y)
```

Scatter Plot – Example

```
# example of a scatter plot
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
seed(1)
# first variable
x = 20 * randn(1000) + 100
# second variable
y = x + (10 * randn(1000) + 50)
# create scatter plot
pyplot.scatter(x, y)
# show line plot
pyplot.show()
```

Scatter Plot – Example



Contents

1 Numpy

2 Matplotlib

3 Pandas

Creating a dataframe in Pandas [3]

- Importing pandas

```
import pandas as pd
```

- Create a simple DataFrame syntax: pd.DataFrame(column1 : value1, column2 : value2, column3 : value3)
- You can have anything as column names and anything as values.
- The only requirement is to have all value lists being of equal length (all are of length 3 in this example)

Creating a simple dataframe [3]

```
[52]  1 df00 = pd.DataFrame({'name':['Bob','Jen','Tim'],
      2                      'age':[20,30,40],
      3                      'hobby':['tennis', 'swimming', 'golf']})
      4
      5 df00
```

	name	age	hobby
0	Bob	20	tennis
1	Jen	30	swimming
2	Tim	40	golf

View the column names and index values [3]

- The index is one of the most important concepts in pandas.
- Each dataframe has only a single index which is always available as df.index and if you do not supply one (as we did not for this dataframe) a new one is made automatically.
- Indexes define how to access rows of the dataframe.
- The simplest index is the range index but there are more complex ones like interval index, datetime index and multi index.

View the column names and index values [3]

```
[53]  1 print(df00.columns)
      2 print(df00.index)
```

```
Index(['name', 'age', 'hobby'], dtype='object')
RangeIndex(start=0, stop=3, step=1)
```

View the column names and index values [3]

- We could select specific columns

```
[54]  1 df00[['name','hobby']]
```

	name	hobby
0	Bob	tennis
1	Jen	swimming
2	Tim	golf

View the column names and index values [3] - `.loc` vs `.iloc`

- We can always access any row in the dataframe using `.iloc[]` or `.loc[]`.
- `.loc` selection is based on the value of the index. For example if the index was categorical we could index via some category.
- `.iloc` selection is **always** based on integer positions. When using `.iloc` we are treating the dataframe as 2d-array with no special structure compared to the case of `.loc`

View the column names and index values [3]

```
[57]  1 df00.loc[0] #index based
```

```
name         Bob
age           20
hobby     tennis
Name: 0, dtype: object
```

```
[59]  1 df00.iloc[0] #relative position based indexing
```

```
name         Bob
age           20
hobby     tennis
Name: 0, dtype: object
```

```
[60]  1 df00.iloc[-1,:] #Use iloc to select the last row
```

```
name        Tim
age          40
hobby     golf
Name: 2, dtype: object
```

References

[1] https://colab.research.google.com/github/cs231n/cs231n.github.io/blob/master/python-colab.ipynb
[2] J. Unpingco, Python for Probability, Statistics, and Machine Learning (1st. ed.). Springer Publishing Company, Incorporated, 2016
[3] https://colab.research.google.com/drive/1a4sbKG7jOJGn4oeonQPA8XjJm7OYgcdX
[4] VEF Academy, Machine Learning, 2019
[5] VEF Academy, Data Analytics, 2020