Fundamentals of Social Data Science: Day 1 - Distributions

Bernie Hogan

Associate Professor and Senior Research Fellow

Learning goals for today

- Course Overview
- Distinguish a mathematical distribution from an empirical one
- Comparing mathematical distributions to empirical ones
- Creating a Python environment to work in
- Basic Series operations
- Overview of week 1 project
- Assigning groups

Course Overview

- The goals for this course are readiness in Python programming and related skills for readiness in data analysis for machine learning and advanced quantitative approaches.
- **Week 1**. Basic data manipulation.
- **Week 2**. Working with external data: files, repos, APIs.
- **Week 3**. Understanding distance as a concept, working on servers
- **Week 4**. Basics of classification and embeddings.
- ***Note that this is somewhat different to the syllabus as provided.***

Summative assessment

- This course will be assessed as a part of the Introduction to Data Science and Machine Learning in Python take home exam which will run in Week 10 of the term.
- This portion of the course will involve drafting around 1000-1200 words on a data analysis task. The specific parameters of the task will be revealed in the course paper.
    - It will follow a similar logic as the exercises in the Lab. It will involve a specific site where we can choose what data to collect and to make an observation about communication related to that site.
    - You will be evaluated on the strength of the claims, the visualizations, the judicious use of any statistics, and the clarity of the code.

DIKW

From the Information Visualisation (INFOVIS) literature is a schema called DIKW standing for Data -> Information -> Knowledge -> Wisdom (Rowley, 2007).

- **Data**: measurements of phenomena,
- **Information**: Signals from that data / “differences that make a difference”,
- **Knowledge**: Information situated in or understood in context,
- **Wisdom**: Information outside of or in an analogous context.

It’s not bad but it’s certainly not perfect. Do we start with data? Does all knowledge need to go through this schema to become wisdom? What of things we learn from experience rather than through testing?

(PO)DIKW

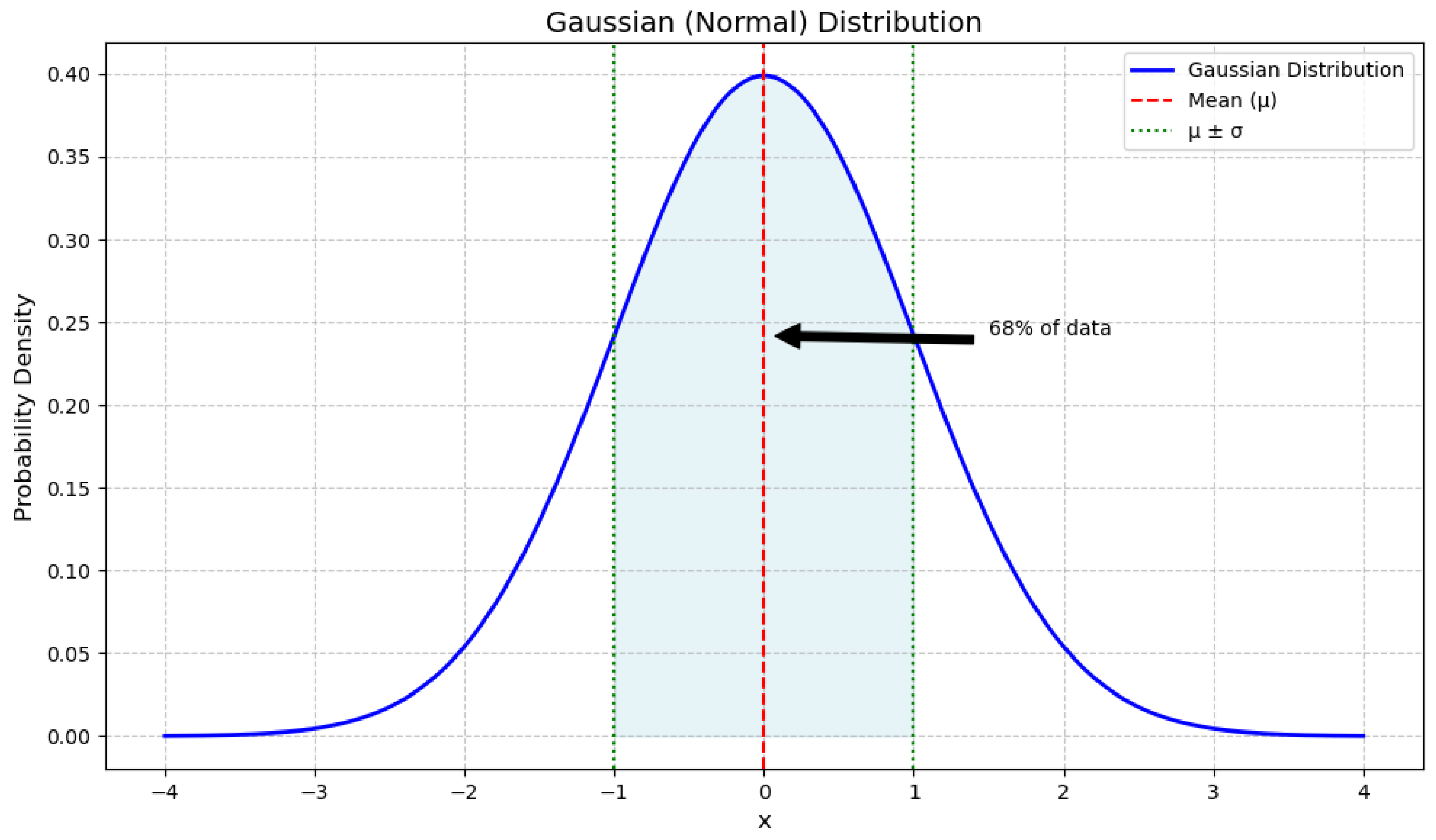
We might modify this schema and start not from data but from ‘the world’ (not the earth, but the known or knowable Universe). The world simply exists. To interact with the world we take in information, encode it, and react to it. So prior to data we should think of:

- **Phenomena:** What is it we think we are testing, observing, expecting from the world? How are we using our intuitions, prior academic literature, and analytical tools to perceive (or collect) phenomena?
- **Operationalisation**: How can we encode and thus measure or compare different phenomena.
    - E.g., Is every friendship the same? How would we know what to measure for a friendship? A Facebook friendship? “Mutuals” + real name? Verbally stated “is my friend”, observed through proximity at an event?
- We will return to this in later in the week when we operationalize “helpfulness” on Stack Exchange

Mathematical distributions: f(x)

- A mathematical distribution is an abstraction that is based on a specific formula that generates probabilities.
- A uniform distribution means that each number is equally likely.
- A normal distribution means that numbers near the mean are more likely.
- A scale-free distribution means that numbers are extremely skewed.
- These are three common ideal types, but there are a huge number of both different distributions and variations in shape.

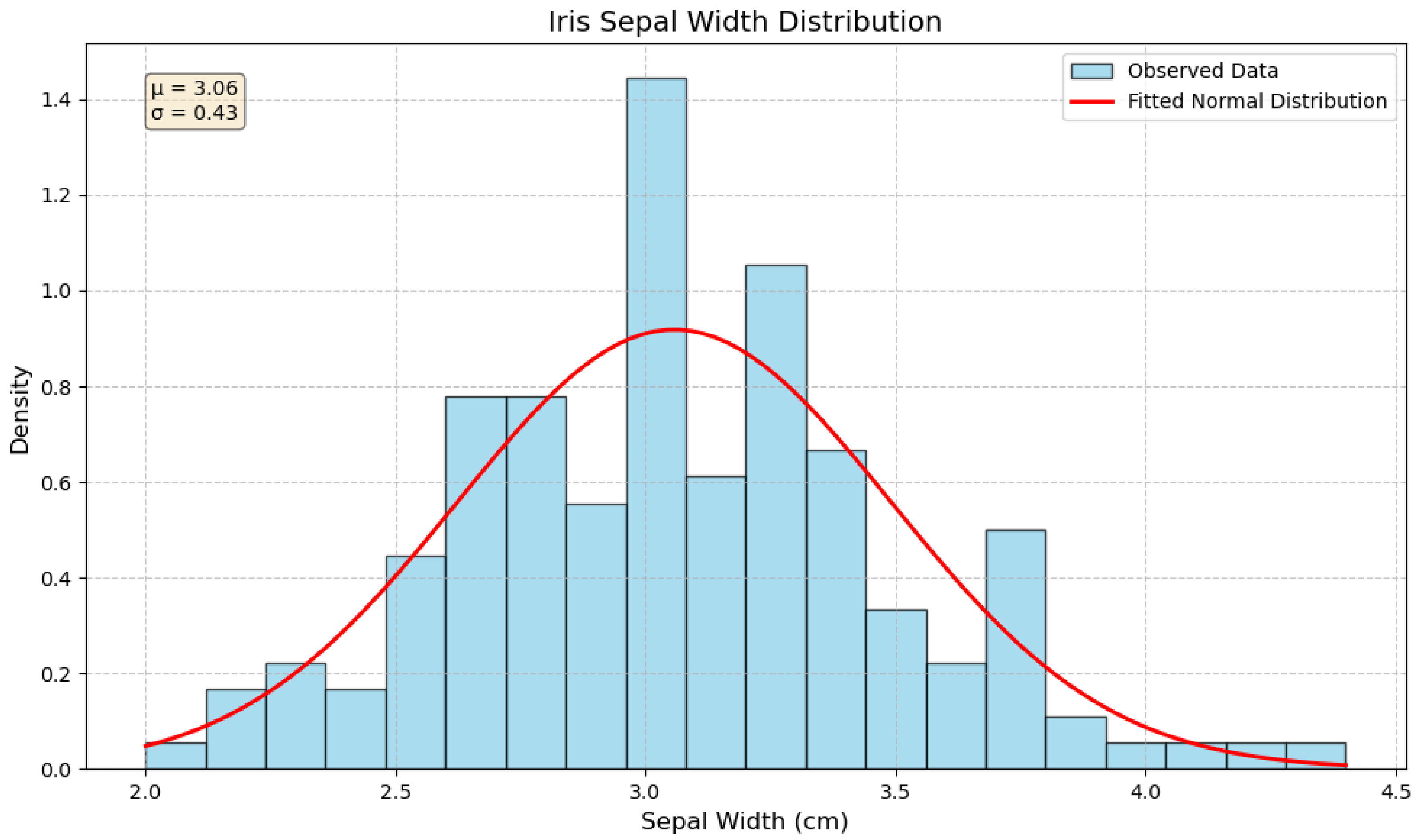
A normal (Gaussian) curve



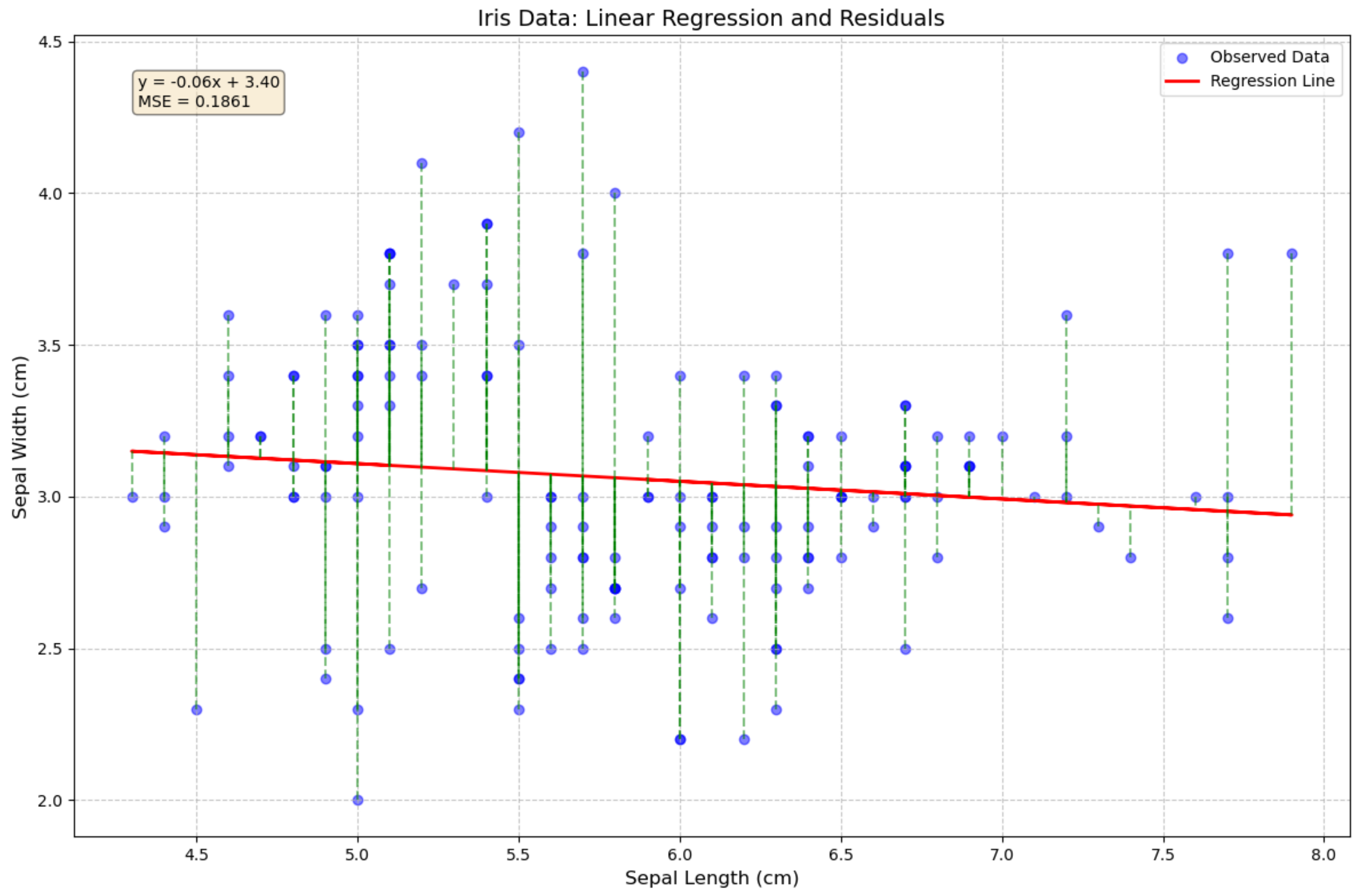
Observed distributions

- An observed distribution or an empirical distribution is a collection of observations or measurements.
- They may or may not conform to an expected mathematical distribution.
- The extent to which information about one distribution can inform another is generally of interest to us. Can income predict health? Can place of birth predict life satisfaction? This is correlating two empirical distributions, but we can also compare to mathematical distributions, and often do this in our calculations.

Comparing an observed distribution to a mathematical one



Comparing observed data to a function: Loss



How do we handle empirical distributions

- Empirical distributions are stored as data. They are typically stored in tables since these structures allow us to align multiple measurements on the same object: one row per case, many columns for many measurements.
- This can be done in Python, Excel, R, etc…
- What are the advantages to working on distributions in:
  - Excel or Google Sheets?
  - Python or R?

Python’s Data Package: Pandas

- Pandas is the de facto leader in managing data in Python. A single column can be addressed as a Series, a single table as a DataFrame.
- Most data manipulation operations you might expect are available, but the syntax can be tricky. Resources are available, so I only cover some brief key points here.

Why create a Python environment

- Many tasks involve the use of external packages or programs.
- These often have version collisions. For example, consider “Kohya”, the program used to train AI image models. [see browser]
- Building a Python environment ensures that your code remains suitable for a specific Python version and is updated in tandem.
- What about ‘too many versions?’?
- Why venv and not conda or UV?

Getting data from a Series I: By index and position

A `Series` always has an index. It is automatically generated but you can alter it to be meaningful in someway. A `RangeIndex` is just the numbers *0…n*. You can access each element in a Series:

- **By label**: If the index contains days of the week, `ser["Monday"]` will return the value with the `"Monday"` index.
- **By position**: You can request the element in the n^{th} position as a `Series` always has an order (but you can reorder it).
- **Iteratively**:
  - `for value in series_obj: print(value)`
  - `for index in series_obj.index: print(index)`
  - `for pair in series_obj.items(): print(pair)`

Getting data from a Series II: Masks and filters

To mask is to return a 'view' of the series. It's not a copy so this might cause trouble if you try to edit it. Yet, it is useful for getting subsets for analysis. You can use Boolean operators on a Series:

- Or: |, And: &, Greater than: <, Less than: >, equal to: ==, not equals: !=
- It will return a new series with `True` or `False` based on the condition.

```
ser_obj = pd.Series([30,10,35,21])
ser_obj > 30 >>> False, False, True, False
```

- Then when you index by this new series, it only returns the `True` elements.

```
ser_obj[ser_obj > 30] >>> [35, 21]
```

To add multiple conditions, remember to put each one (in parentheses):

```
ser_obj[(ser_obj > 30) | (ser_obj < 20)] >>> False, True, True,
False
```

Getting data from a Series III: As aggregate and summary

We can use operations to summarise a `Series`. `Series`-specific operations are *methods*, general operations are *functions*. For example:

- **As a method**: `series.mean()` will report the mean value for a Series of numbers.
- **As a function**: `len(<Series>)` will report the number of entries in the Series.

A common method is `value_counts()`. This will summarise the values. They become the index and a count of how many becomes the new value. This is like 'tabulate' in R or 'freq' in SPSS.

```
ser_obj = pd.Series([1,1,2,1,2,1,3,4,2,1,3,1,2])
ser_obj.value_counts() >>>
```

```
1    6
2    4
3    2
4    1
dtype: int64
```

Changing a Series IV: Using map(lambda)

A `lambda` function is one that does not have a name. It is an ad hoc function. It is not so useful on its own, but inside a `map` statement it is powerful: you can do the same operation for every element in a `Series`.

- `demo_lam = lambda x: x*2`
- `demo_lam(10) >>> 20`

- Inside a `map` statement `x` (just by convention) stands for each element:
  - `ser_obj = pd.Series([5,6,10])`
  - `ser_obj.map(lambda x: x*2) >>> 10, 12, 20`
- `lambda` works great here since we did not want to create a dictionary for all possible cases. If you want to do many things to each element of a series, write a function and then call that instead, or within `lambda` if it has multiple arguments.

Widening out: How to Code / When to Code

- Fixed costs versus marginal costs:
- "The reason that the rich were so rich, Vimes reasoned, was because they managed to spend less money. Take boots, for example. … A really good pair of leather boots cost fifty dollars. But an affordable pair of boots, which were sort of OK for a season or two and then leaked like hell when the cardboard gave out, cost about ten dollars. … But the thing was that good boots lasted for years and years. A man who could afford fifty dollars had a pair of boots that'd still be keeping his feet dry in ten years' time, while a poor man who could only afford cheap boots would have spent a hundred dollars on boots in the same time and would still have wet feet." – Terry Pratchett
- Impoverished code, either for lack of time or expertise can lead to more work over time.
- In this case, the upfront costs of 50 dollars allowed for more boots-wearing-per-year, whereas the upfront cost of 10 dollars led to repeated purchases yearly.

FREE code as practice and paradigm

To think of how to approach coding so that you make good use of your time, consider the FREE mnemonic:

- **Functioning**: With the expected input does the program give the expected output. Get this right first or the rest do not matter.
- **Robust**: Can the program still manage with unexpected inputs?
- **Elegant**: Does the program minimise repetition and have an easy-to-understand structure?
- **Efficient**: Does the program run using algorithms and technologies that reduce computation time?

This is in addition to the notion of free and open-source software (FOSS), which has been an essential part of the development of modern programming and science.

Today’s Lab: Observing Distributions

- Today, you are encouraged to open the Movies Stack Exchange data provided on canvas and to explore distributions.
- We are interested in whether you are able to manage:
  - Variable types (when are they numeric?)
  - Masking (can you create a series from a subset of the data?)
  - Empirical testing (can do determine whether the data is significantly different from an ideal distribution).
    - Is there a uniform distribution in this data? That might be hard to find.

Labs and Groups

- Individuals are assigned into Groups. These groups broadly follow colleges allowing people to work in groups together.
- Week 1.1-1.3: Individual assignments
  - Week 1.1 and 1.2 are not graded, but shared on next day.
- Week 2.1-3.1: Group assignment on Wikipedia
  - Weeks 2.1-2.3 are not graded but shared on next day
- Week 3.2-4.3: Group assignment on Reddit data
  - Weeks 3.2-4.1 are not graded but shared. 4.2 is dedicated to effort on the assignment.

Wednesday’s Class

- For the next lecture we will be looking at comparisons between variables. Much of this will be covered in class and in the virtual walkthrough. However, the most useful readings might be:
    - FSStDS: Chapter 3 (DataFrame),
    - Chapter 9.5-end of chapter.
- Again, walkthrough will be released by 9am on morning of (if not sooner).