Econometrics

TA Session 1

Lucia Sauer

2025-09-25

TA Materials

All the material for this course — slides, code, and some LaTeX utilities for referencing assignments — will be hosted in the GitHub repository.

> git clone HHTPS

Install uv

> uv sync

Pull the repository and get the updated materials for each session.

Overview

- Matrix Algebra with Python
- Derivatives with vectors
- Histograms and KDE.

Why These Topics?

Matrix Algebra

- Compact notation for regression & multivariate models
- Solve systems efficiently with Python

Derivatives with Vectors

- Gradients for optimization & ML
- Key for regression, loss functions

Histograms & KDE

- Explore data shape: skewness, modality, outliers
- Nonparametric distribution estimation & model checks

System of equations in Matrix Form

Consider the following system of equations:

$$\begin{cases} 2x_1 + 3x_2 + 10x_3 = 5 \\ 4x_1 + x_2 + 12x_3 = 6 \\ 7x_1 + 2x_2 + x_3 = 10 \end{cases}$$

Exercises

- How can we write this system in matrix form?
- Solve it using Python.

Matrix Operations using Python

```python
import numpy as np

#Define matrix A and vector b
A = np.array([[2, 3, 10],
                [4 , 1, 12],
                [7 , 2, 1]])
b = np.array([5 , 6, 10])

#Transpose A
A_transpose = np.transpose(A)

#Inverse A
A_inverse = np.linalg.inv(A)

x = np.dot(A_inverse, b)
print('Solution: ', x)
```

```
Solution:  [1.21078431 0.74509804 0.03431373]
```

Matrix Operations using R

```r
# Define matrix A and vector b
A <- matrix(c(2, 3, 10,
              4, 1, 12,
              7, 2, 1),
            nrow = 3, byrow = TRUE)

b <- c(5, 6, 10)

# Transpose A
A_transpose <- t(A)

# Inverse A
A_inverse <- solve(A)

# Solve Ax = b
x <- solve(A, b)
print(x)
```

Derivatives of a vector

Gradient with respect to a vector

- In single-variable calculus:
  $\frac{d}{dx} f(x)$ is the slope of a scalar function of one variable.
- In multivariable calculus:
  If $f(z_1, z_2)$, we can group the variables into a vector:

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

then the derivative with respect to a vector is the **gradient**:

$$\nabla_z f(z) = \begin{bmatrix} \frac{\partial f}{\partial z_1} \\ \frac{\partial f}{\partial z_2} \end{bmatrix}$$

Take the partial derivative with respect to each coordinate, then stack them in a column vector.

Consider:

$$a = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad B = \begin{bmatrix} b_{12} & b_{12} \\ b_{22} & b_{22} \end{bmatrix}$$

Exercises

Verify, by showing all the relevant steps, that:

$$(i) \quad \frac{\partial z' a}{\partial a} = z$$

$$(ii) \quad \frac{\partial a' z}{\partial z} = a$$

$$(iii) \quad \frac{\partial z' B z}{\partial z} = 2Bz \quad \text{since B is symmetric}$$

Density Estimation - Introduction

Given some observations from some variable X, we would like to obtain an estimate of its density.

Parametric vs Nonparametric

- **Parametric approach**: we would choose a parametric density function (normal, exponential, uniform, etc) and use data to estimate the parameters of this density.

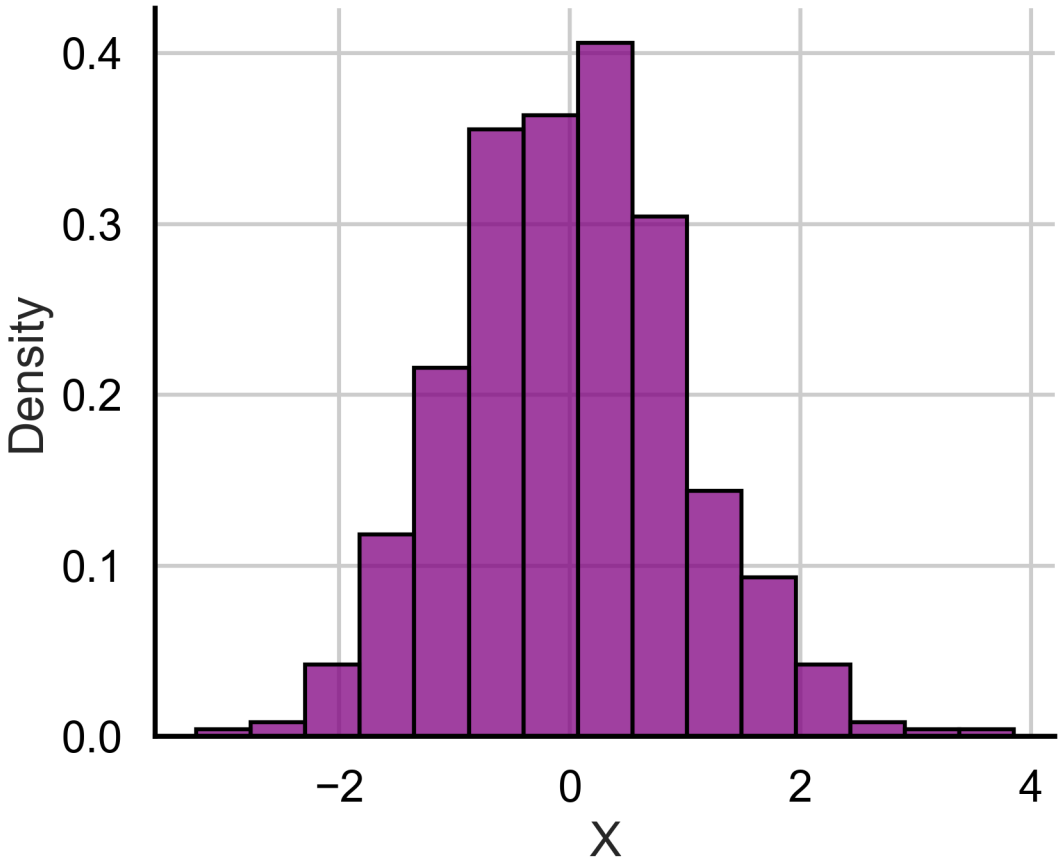
Problem: what if X doesn't follow a normal distribution?

- **Nonparametric approach** like histograms and KD estimate the distribution of a variable X without strong parametric assumptions.

Histogram

Key idea: A histogram is a graphical tool to visualize the distribution of a dataset.

- A **nonparametric method** to estimate the probability density function (PDF)
- Groups data into **intervals (bins)**
- Plots the **frequency** of data points in each bin



Concept of Density

Let's first differenciate between frequency histograms, and density histograms.

Bins	Frequency	Width	Density
0-5	2	5	$2/(12*5) = 0.033$
5-10	4	5	$4/(12*5) = 0.067$
10-15	5	5	$5/(12*5) = 0.083$
15-20	1	5	$1/(12*5) = 0.017$

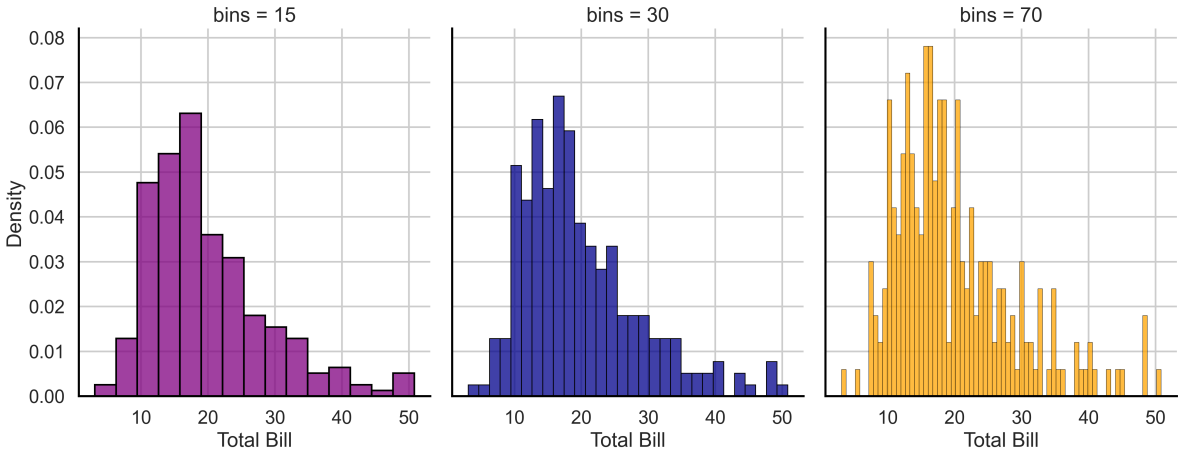
Mathematically, for bin width h and number of observations n:

$$\hat{f(x)} = \frac{1}{nh} \sum_{i=1}^{n} \mathbf{1}\{x_i \in \text{bin}(x)\}$$

Coding Example

Important parameters bins, stat, binwidth, binrange

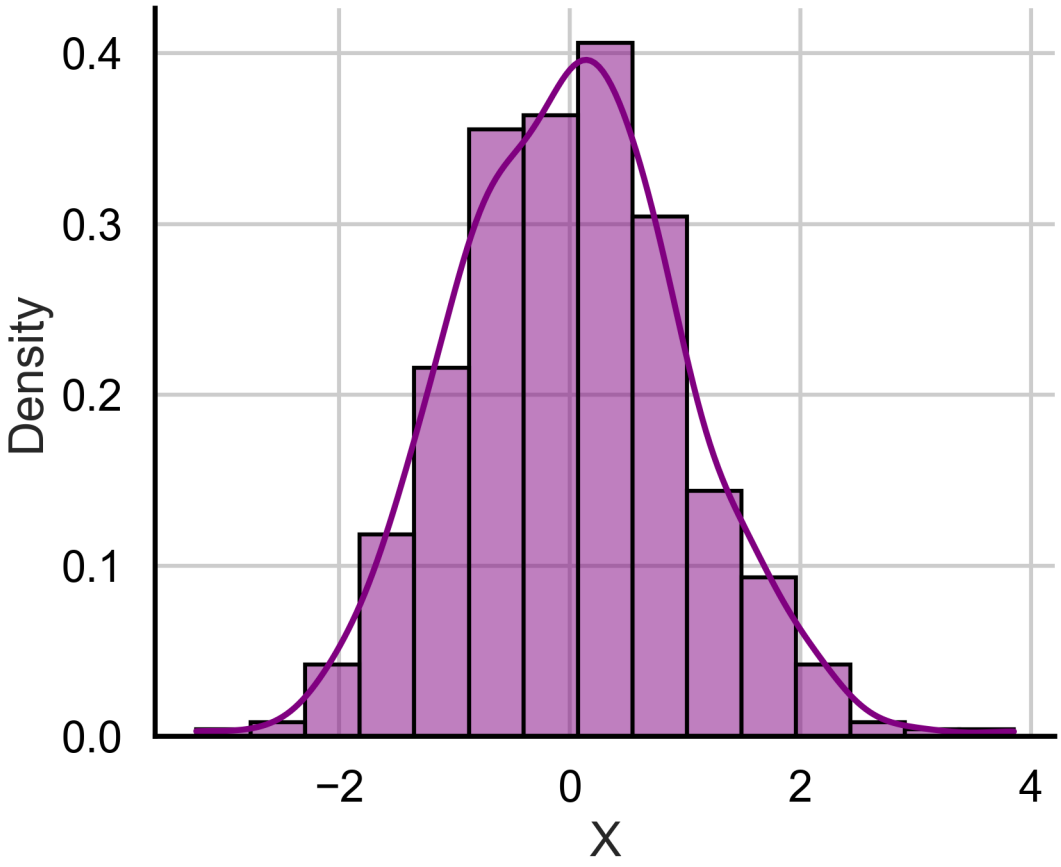
```
import seaborn as sns
x = sns.load_dataset("tips")["total_bill"]
bins = [15, 30, 70]
for bin in bins:
  sns.histplot(x,bins=bin, stat='density', binwidth=None, binrange=None)
```



Kernel Density Estimation

Key idea: Smooth, nonparametric estimate of the probability density function

- Instead of counting frequencies in bins, KDE places a smooth kernel function (usually Gaussian) centered at each data point, and then averages them.



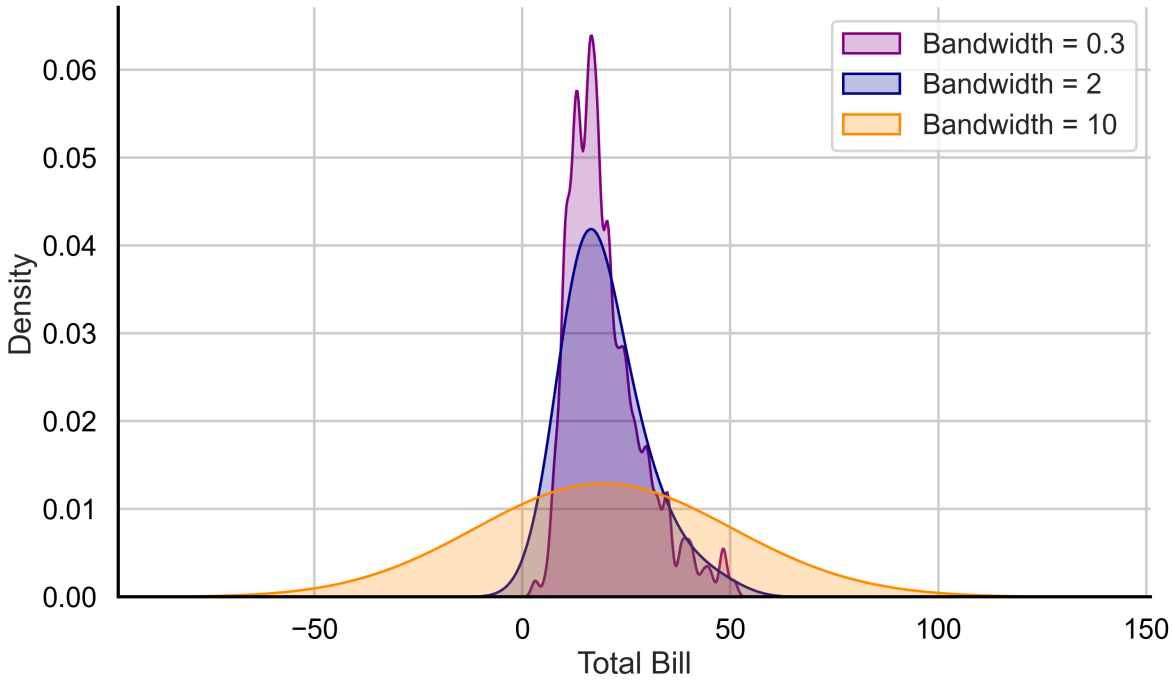
Mathematical formulation:

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

- N is the number of observations
- w is the bandwidth (controls how much the influence of each observation expands)
- and, K is the Kernel, a function that defines the shape and distribution of influence associated with each observation.

Coding Example

```
import seaborn as sns
x = sns.load_dataset('tips')['total_bill']
for bw in [0.3, 1, 2]:
    sns.kdeplot(x, bw_adjust=bw, label=f"Bandwidth = {bw}")
```



- Small h: very wiggly estimate (low bias, high variance).
- Large h: oversmoothed estimate (high bias, low variance).

Thanks!