

Laboratory Work No. 4

Modelling the Distribution of Inverter Sensor Data with the Normal Distribution

Related lecture: Lecture 3 — Probability Mass Functions, Cumulative Distribution Functions, Modeling Distributions, and Probability Density Functions

Dataset: the same GECAD Photovoltaic (PV) system file used in Labs 1–3, PV2013January.xlsx — same student, same assigned date as in the previous labs.

Purpose of the work

To practice reading a real sensor dataset into a pandas DataFrame and summarizing a variable with its mean and standard deviation; to use those two numbers to build a modelled normal distribution; to compare that model against the real data visually, by overlaying the modelled curve on a histogram of the actual readings; and to see how well — or how badly — a normal model fits a variable whose true shape includes a large spike of zero values, both before and after that spike is filtered out.

Background

1. The three variables for this lab. As in Labs 2 and 3, every sheet of the workbook records three columns that each tell a different part of the story of how the inverter behaves over a day:

Column	Meaning	Why it's interesting
F — TmpMdul C	Module (panel) temperature, °C	Varies continuously all day and night — a genuine physical quantity with no “off” state.
I — Fac	Frequency of the AC current the inverter generates, Hz	Is exactly 0 whenever the inverter is idle (no sunlight); only takes a real value (≈ 50 Hz) while actively feeding the grid.
P — Pac	AC power generated, W	Same pattern as Fac: 0 at night, a real distribution of values during the day.

2. Mean and standard deviation as model parameters (Lecture 3 / Think Stats Ch. 2–3).

The normal (Gaussian) distribution is fully determined by just two numbers: its mean μ and its standard deviation σ . “Modelling” a variable with a normal distribution means computing the SAMPLE mean and standard deviation from the real data, and then treating those two numbers as the parameters of an idealized bell curve $N(\mu, \sigma)$:

```
mu = data.mean()
sigma = data.std()
```

This is the same estimation idea used throughout the course: the sample statistics stand in for the unknown true parameters of whatever process generated the data.

3. Plotting the modelled normal PDF (Lecture 3).

Once μ and σ are known,

`scipy.stats.norm.pdf(x, mu, sigma)` computes the height of the normal probability density function at any x . Plotting that function over a range of x values draws the smooth theoretical bell curve:

```
from scipy.stats import norm
import numpy as np

xs = np.linspace(data.min(), data.max(), 300)
plt.plot(xs, norm.pdf(xs, mu, sigma))
```

4. Overlaying the model on the real data. A PDF curve and a plain count-histogram are not directly comparable, because a histogram's bar heights depend on the chosen bin width and the sample size. The fix is to normalize the histogram into a *density* histogram — one whose total area is 1, exactly like a PDF — using `plt.hist(data, bins=..., density=True)`. With both drawn on the same axes, the modelled curve and the real data become visually comparable: a good fit means the curve closely traces the top of the bars; a poor fit means the curve and the bars clearly disagree in shape.

5. Filtering out zeros. As in Lab 3, a Boolean condition filters a Series exactly as before: `nonzero = series[series > 0]`. For Fac and Pac, the large spike of exact zeros represents a different regime (“inverter off”) rather than genuine variability of the quantity itself, and a normal distribution — a single smooth, symmetric bell curve — is a poor model for data that is really a mixture of “off” (a spike at 0) and “on” (some spread of positive values). Re-fitting and re-plotting the model on the filtered, “on-only” data isolates the second regime and lets the normal model be judged on the variability it is actually meant to describe.

Task

Use the same data (variant) and the same CSV-reading approach as in Lab 1. For each of the three columns F (TmpMdul C), I (Fac), and P (Pac):

- 2.1 — Find the mean value and the standard deviation value of the column.
- 2.2 — Model a normal distribution using the calculated mean and standard deviation as its parameters, and plot the modelled distribution (its PDF).
- 2.3 — Redo the graph so that the real data (as a density histogram) is displayed together with the modelled normal curve on the same axes.
- 2.4 — Repeat steps 2.1–2.3 with the filtered data (zero values removed, where filtering is appropriate — see the Background section above for how to decide).

Place your code and all resulting graphs (modelled-vs-real overlays, raw and — where applicable — filtered, for all three columns) into your report.

Optional / bonus tasks

- For each column, compute the skewness of the real data (e.g. with `scipy.stats.skew`) and relate its sign and size to how well the normal overlay fits visually.
- Overlay the raw and filtered normal models for Fac (or Pac) on the SAME axes, rescaled so both are visible, and comment on how differently they are centered and spread.
- For TmpMdul C, try modelling the data with a different distribution shape (e.g. a mixture of two normals, or simply two separate histograms split by night/day) and compare, visually, how much better or worse the fit is than the single normal model.
- Use a goodness-of-fit diagnostic such as a Q-Q plot (`scipy.stats.probplot`) for one column, and explain in one or two sentences what deviation from the diagonal line reveals that the histogram overlay does not.

Report requirements

Your submitted report must contain:

- the topic and purpose of the work;
- your variant number and assigned date (as in the previous labs);
- your program code for all of steps 2.1–2.4, for all three columns;
- the printed mean and standard deviation (raw and, where filtering was applied, filtered) for all three columns;
- all resulting overlay graphs — raw for every column, plus filtered for any column where zeros were removed;
- a short conclusion: for which columns the normal model was a good fit and for which it was not, and why, based on what the overlay plots show.

Worked example (teacher's variant)

Date used: **15-01-2013** — the same date and the same `data_pv_15-01-2013.csv` file as the Lab 1–3 worked examples.

Column F — TmpMdul C (module temperature)

Quick check first: `(df['TmpMdul C'] == 0).sum()` returns **0** — there are no zero readings at all, so step 2.4 (filtering) does not apply to this column; the raw model is the only model needed.

```
from scipy.stats import norm

tmod = df['TmpMdul C']
mu, sigma = tmod.mean(), tmod.std()           # step 2.1
print(mu, sigma)

xs = np.linspace(tmod.min(), tmod.max(), 300)
plt.hist(tmod, bins=30, density=True)         # step 2.3: real data
plt.plot(xs, norm.pdf(xs, mu, sigma))         # step 2.2: modelled normal
```

Statistic	Value
Mean (μ)	18.24 °C
Standard deviation (σ)	2.40 °C

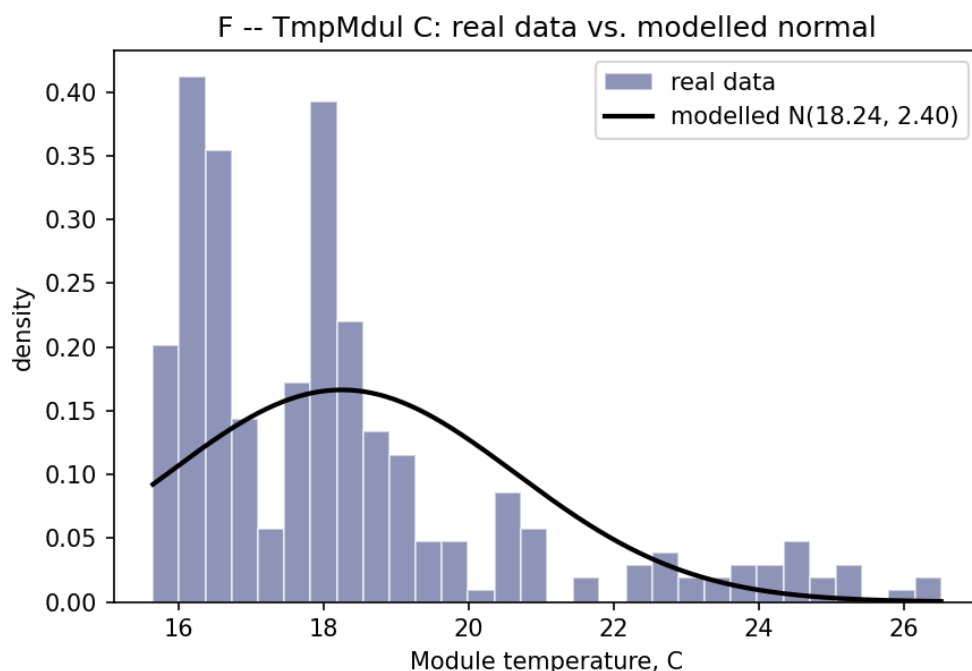


Figure 1. TmpMdul C: real data (density histogram) vs. modelled $N(18.24, 2.40)$. The real distribution is visibly bimodal — two separate humps rather than one smooth peak — so the single bell curve only loosely tracks the overall spread and misses the two-peaked shape entirely.

Column I — Fac (AC frequency)

Quick check: `(df['Fac'] == 0).sum()` returns **181** out of 288 readings (63%) — the inverter is idle most of the night and early morning, exactly as found in Labs 2 and 3. A normal model fitted to this raw mixture of “off” and “on” values is a particularly poor idea, as the plot below shows, so step 2.4 calls for filtering before the model is judged fairly.

```
fac = df['Fac']
mu_raw, sigma_raw = fac.mean(), fac.std()      # step 2.1, raw
print(mu_raw, sigma_raw)

fac_nz = fac[fac > 0]
mu_f, sigma_f = fac_nz.mean(), fac_nz.std()    # step 2.4: filter idle zeros
print(mu_f, sigma_f)                          # step 2.1, filtered
```

	Raw (n = 288)	Filtered, Fac > 0 (n = 107)
Mean (μ)	18.57 Hz	49.99 Hz
Standard deviation (σ)	24.20 Hz	0.02 Hz

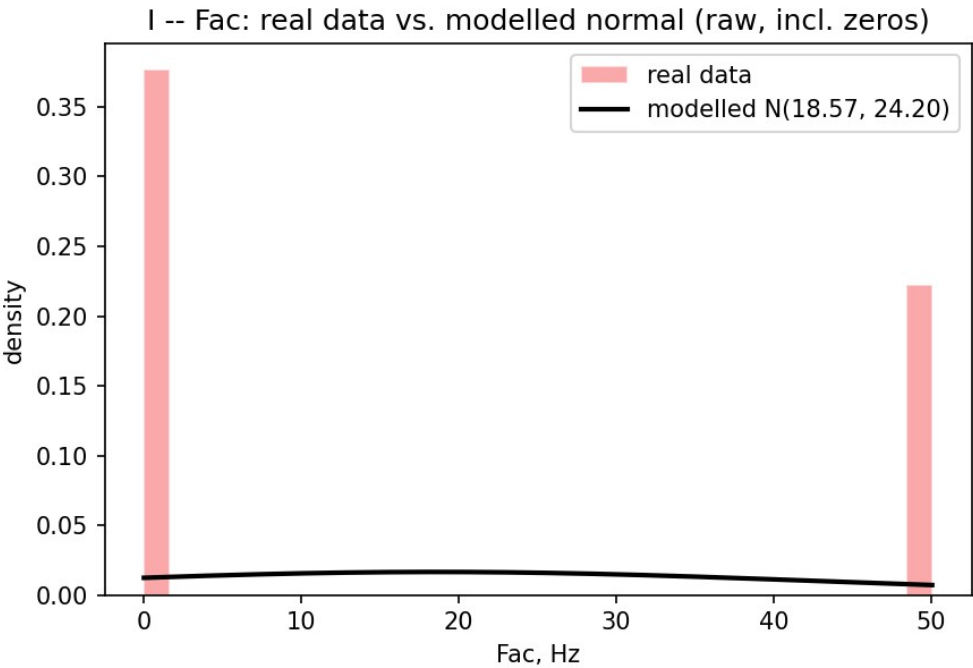


Figure 2. Fac: real data vs. modelled normal, RAW data (incl. zeros). The real data is two narrow spikes — at 0 and at ≈ 50 Hz — with almost nothing in between; the modelled bell curve, centered at ≈ 18.6 Hz with a huge spread, sits almost flat across the whole range and resembles the real data in no way at all. This is the clearest possible illustration of a normal model applied to the wrong kind of data.

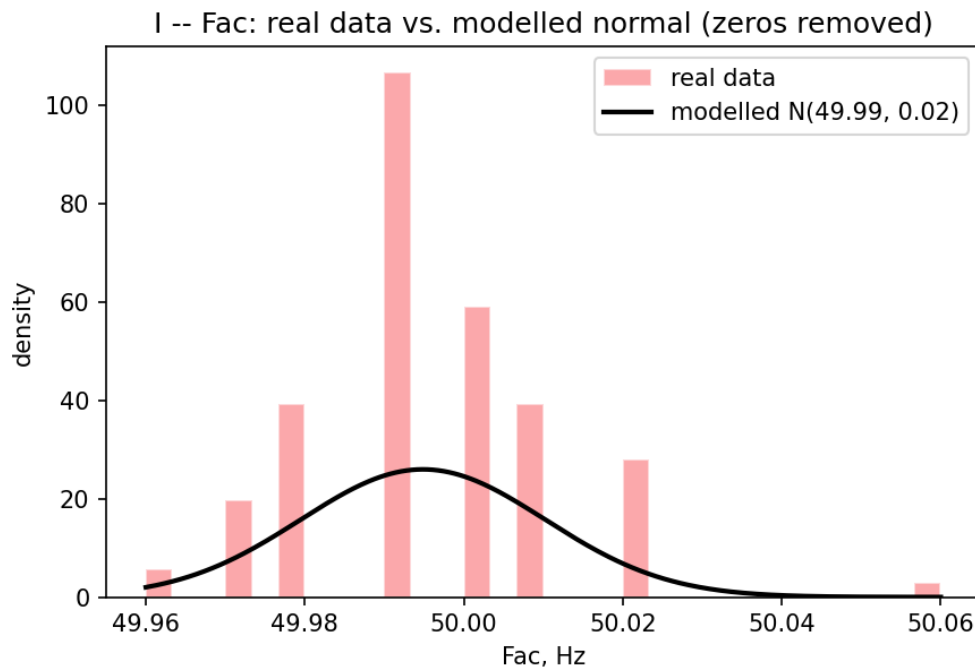


Figure 3. Fac: real data vs. modelled normal, zeros removed. Restricted to the “on” regime, the real data clusters tightly around 50 Hz in a few narrow bars (the frequency is digitally regulated, so it only takes a handful of distinct values); the modelled curve is far wider than the real spread, overstating how much the frequency actually varies while the inverter is running.

Column P — Pac (AC power)

Quick check: `(df['Pac'] == 0).sum()` returns **191** out of 288 (66%) — the same idle-at-night pattern as Fac, so the same filtering decision applies.

```
pac = df['Pac']
mu_raw, sigma_raw = pac.mean(), pac.std()      # step 2.1, raw
print(mu_raw, sigma_raw)

pac_nz = pac[pac > 0]                          # step 2.4
mu_p, sigma_p = pac_nz.mean(), pac_nz.std()    # step 2.1, filtered
print(mu_p, sigma_p)
```

	Raw (n = 288)	Filtered, Pac > 0 (n = 97)
Mean (μ)	15.65 W	46.48 W
Standard deviation (σ)	30.07 W	35.44 W

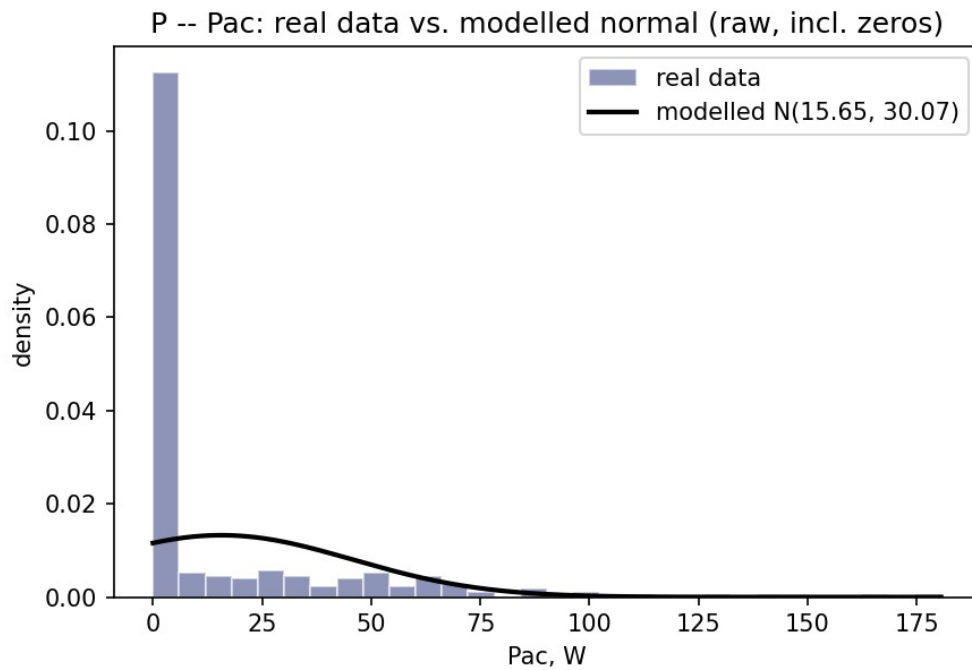


Figure 4. Pac: real data vs. modelled normal, RAW data (incl. zeros). Like Fac, the huge spike at 0 drags the modelled curve into a wide, nearly flat shape that bears little resemblance to the real, tightly-spiked distribution.

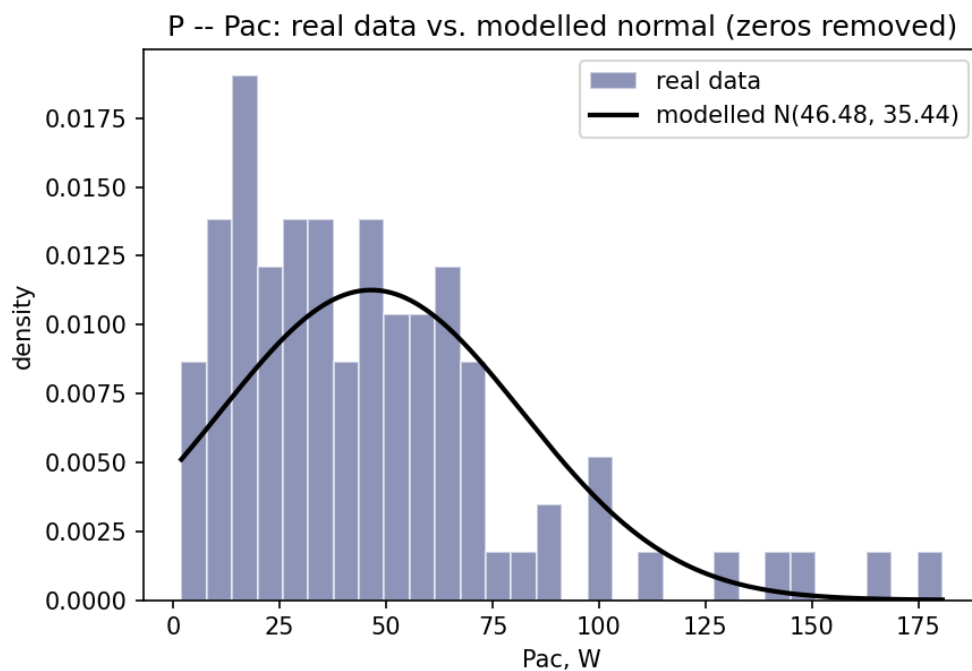


Figure 5. Pac: real data vs. modelled normal, zeros removed. The filtered data is still visibly right-skewed -- many moderate readings and a long tail of large ones -- so even the better-behaved filtered model is only a rough approximation: a true normal distribution would be symmetric around its mean and would, in principle, allow negative power, which is physically impossible here.

Summary of decisions

Column	Zeros filtered?	Raw fit quality	Filtered fit quality
F — TmpMdul C	No (no zeros present)	Poor — real data is bimodal	n/a — same as raw

Column	Zeros filtered?	Raw fit quality	Filtered fit quality
I — Fac	Yes (181/288, 63%)	Very poor — two spikes vs. one flat curve	Rough — real data narrower and more granular than the model
P — Pac	Yes (191/288, 66%)	Very poor — zero spike vs. wide flat curve	Moderate — model misses the right-skew

Interpretation

Fitting a normal distribution means trusting that two numbers — the mean and the standard deviation — capture everything important about a variable's shape. This worked example shows that trust is often misplaced. TmpMdul C, despite having no zero-filtering issue at all, turned out to be visibly bimodal (it clusters around two different temperature bands rather than one central value), so even its “clean” raw data is not well described by a single bell curve. Fac and Pac are more dramatic failures in their raw form: because roughly two-thirds of their readings are exactly 0 (the inverter idle overnight) and the rest are a real, positive-valued distribution, the raw data is really a MIXTURE of two regimes, not a single smooth process — and a normal model, built blindly from the raw mean and standard deviation, ends up as a wide, nearly flat curve that resembles neither regime. Filtering out the zeros removes the mixture and lets the model describe only the “on” regime, which helps considerably for both columns, but even then Fac's real data is narrower and more granular (it is digitally regulated near 50 Hz) than a smooth bell curve suggests, and Pac's real data stays right-skewed with a long tail toward higher power — a shape a symmetric normal distribution can never fully capture. The general lesson: always look at the overlay, not just the two fitted numbers, before trusting a normal model.

Control questions

- Why does fitting a normal distribution to the RAW Fac or Pac data produce a model that barely resembles the real histogram at all?
- What specific feature of a variable's real distribution (visible in a histogram) is a strong warning sign that a single normal model will fit poorly?
- Why is `plt.hist(..., density=True)` necessary when comparing a histogram to a modelled PDF, rather than just using the default histogram?
- TmpMdul C had no zeros to filter, yet the normal model still fit it only loosely. What does this tell you about the relationship between “has no zero-spike problem” and “is well-modelled by a normal distribution”?
- A normal distribution technically assigns positive probability to negative values. Why is this a conceptual problem for modelling Pac, and does it matter much in practice given the real range of the data?
- If a future column's real data turned out to be perfectly symmetric and single-peaked (unimodal), would you expect its mean-and-std normal model to fit well? What property of the real data would you still want to check before trusting that model?

Reminder: use your own assigned date's sheet (as in the previous labs) rather than 15-01-2013 when producing your own report — your means, standard deviations, and the resulting model fits will differ from the numbers shown here, and that is expected.