LECTURE 4

Relationships Between Variables and Correlation

Principles and Methods of Data Analysis in the Energy Industry

Sources: Think Stats, Chapter 7 (“Relationships Between Variables”) • Elements of Data Science, Chapter 9 (“Relationships”)

Learning objectives

- Build a scatter plot of two energy variables and recognize when overplotting is hiding the true shape of the relationship.
- Fix overplotting using transparency (alpha), smaller markers, and jittering.
- Compare a distribution across many categories at once using violin plots and box plots.
- Compute and interpret covariance, standardized (z-)scores, and Pearson's correlation coefficient.
- Explain why Pearson's r can be misleading for nonlinear relationships or in the presence of outliers, and when Spearman's rank correlation is the safer choice.
- Distinguish correlation from causation, and recognize a spurious correlation created by a shared on/off pattern in two signals.
- Fit and interpret a simple linear regression with scipy.stats.linregress, and explain why a strong correlation does not automatically mean a practically important slope.

Running example. All of the worked numbers in these notes come from the same GECAD photovoltaic dataset used in Lectures 1–3 and Labs 1–2: 5-minute readings of solar irradiance (IntSolIrr), module temperature (TmpMdul C), AC power (Pac) and grid frequency (Fac). Single-day figures use the 15-01-2013 sheet; multi-day figures combine all 25 available January day-sheets into one table of 6,350 rows.

Part A — Scatter Plots and Overplotting

Why look at relationships at all

Lectures 1–3 treated every column on its own: a histogram of solar irradiance, a PMF of module temperature, a CDF of AC power. That view answers questions about a single sensor, but an engineer rarely cares about one sensor in isolation. The more useful questions are almost always about how two signals move together: does AC power track solar irradiance the way the inverter's datasheet predicts? Does module temperature rise with irradiance, and by how much? Does the inverter's output frequency genuinely depend on how much power it is producing, or is that just an illusion created by the data?

A scatter plot is the simplest tool for a first look at a relationship between two numeric variables: one point per observation, one variable on each axis (Think Stats, Ch. 7; Elements of Data Science, Ch. 9).

A first scatter plot: Pac against IntSolIrr

```
import matplotlib.pyplot as plt

plt.plot(df['IntSolIrr'], df['Pac'], 'o', markersize=3, alpha=0.5)
plt.xlabel('Solar irradiance, W/m^2')
plt.ylabel('AC power, W')
plt.title('Pac vs. IntSolIrr -- 15-01-2013')
```

Each of the 288 five-minute samples from one day becomes one point. `plt.plot(..., 'o')` draws markers with no connecting line, which is the right choice whenever the order of the x-values (here, time) is not the axis being plotted. Even from a single day's data the pattern is unmistakable: AC power rises almost in lock-step with solar irradiance — exactly what a PV inverter's operating principle predicts.

The overplotting problem

One day of data (n = 288) plots cleanly. Combine all 25 available January sheets, however, and the same scatter plot has 6,350 points instead of 288. With solid, full-size markers, the dense region — low irradiance, low power, night-time and heavily overcast readings — collapses into a single black blob. That is exactly the information a scatter plot exists to reveal: how densely the points are packed, and whether there is structure hiding inside that dense region. This problem, called *overplotting*, gets worse with every additional day, sensor, or turbine added to the dataset. Elements of Data Science, Ch. 9 walks through the identical progression using survey data on height and weight; the fixes below are the same three fixes, applied here to sensor data.

Fix 1 — transparency (alpha)

```
plt.plot(all_days['IntSolIrr'], all_days['Pac'], 'o', alpha=0.03)
```

`alpha` sets each marker's opacity, from 0 (invisible) to 1 (fully solid). With thousands of overlapping points, a small alpha (0.03 here) means a spot only looks dark where MANY points genuinely overlap. The dense night-time / low-power region reappears as a visibly

darker patch rather than a flat mass — you can see that it is dense, and roughly how dense. A practical starting point is alpha ≈ 1/√n of the point count, adjusted by eye from there.

Fix 2 — smaller markers

```
plt.plot(all_days['IntSolIrr'], all_days['Pac'], 'o', markersize=0.8, alpha=0.1)
```

Smaller markers overlap less to begin with, so less transparency is needed to reveal structure — the two fixes are normally combined. Watch for the failure mode in the opposite direction: markers so small (and alpha so low) that sparse but important points, such as a rare fault reading, disappear completely. There is no single correct setting; the choice trades off visibility of the dense region against visibility of rare points, and that trade-off should match the purpose of the plot.

Fix 3 — jittering

```
import numpy as np

jittered_hour = hour_of_day + np.random.normal(0, 0.15, n)
plt.plot(jittered_hour, df['Pac'], 'o', markersize=2, alpha=0.1)
```

Some variables are recorded coarsely — the hour of day, or a sensor that rounds to whole degrees — so many real observations land on EXACTLY the same x (or y) value and stack into a vertical (or horizontal) line. Jittering adds a small amount of random noise, `np.random.normal(0, small_sd, n)`, purely for display, spreading the stack sideways so the eye can judge density again. Jittering never changes the underlying data or any calculation on it — only the copy of the data used for the plot.

Part B — Violin and Box Plots

Comparing a distribution across many groups

Suppose the question is how AC power output is distributed for EACH hour of the day — 24 separate groups. Overlaying 24 histograms on one axis is unreadable, and 24 separate small histograms are hard to compare side by side. What is usually wanted is a compact summary of each group's distribution, lined up so the groups can be compared directly. Violin plots and box plots both do exactly this: each takes a categorical grouping variable (here, hour of day, 0–23) on one axis and a numeric variable (Pac) on the other.

Violin plots

```
import seaborn as sns

sns.violinplot(x='hour', y='Pac', data=all_days, inner=None)
```

A violin plot draws a mirrored, smoothed density estimate — the same kind of kernel density estimate (KDE) introduced for a single variable in Lecture 3 — for each group, so its width at any height shows how common that value is within the group. Reading the hourly Pac violins built from all 25 January sheets: the violin for hour 3 is a thin sliver pinned at 0 W, because the panel is idle at night; the violin for hour 12 is wide and spread from 0 W up past 150 W, reflecting the full range of possible cloud conditions at midday. `inner=None` removes the default inner box/quartile marks when only the overall shape is wanted; leave it in when the quartiles should also be visible at a glance.

Box plots

```
sns.boxplot(x='hour', y='Pac', data=all_days, whis=10)
```

Each box shows the median (the line inside the box), the interquartile range or IQR (the box itself), and whiskers extending to the most extreme non-outlier points; points beyond the whiskers are drawn individually as outliers. The `whis` parameter controls how far the whiskers reach before a point counts as an outlier (the default is 1.5 × IQR). A busy dataset with many legitimate extreme values — a solar plant on a rare, unusually clear day, for instance — often needs a larger `whis` (e.g. 10) so that genuine variation is not flagged as noise. A box plot is more compact than a violin plot but hides multi-peaked (bimodal) shapes, so the two are often used together. For a right-skewed variable such as Pac, plotting the y-axis on a log scale (`plt.yscale('log')`) frequently makes the boxes easier to compare.

Worked example: Pac by hour of day, n = 6,350

- Data: every 5-minute reading from all 25 available day-sheets, combined and tagged with its hour of day (0–23).
- Hours 0–5 and 20–23 (night): Pac is essentially a spike at 0 W — the violin is a thin sliver, and the box collapses to a line.

- Hours 10–14 (midday): Pac spans the widest range, from 0 W on a heavily overcast day up to the daily maximum on a clear one. The box's IQR sits high, but with whis = 10 the whisker still reaches down toward 0.
- The transition hours (7–9 and 15–17, sunrise and sunset) show the fastest-growing and fastest-shrinking spread — exactly what is expected physically.

Takeaway: violin and box plots let you see a whole day's generation profile — not just its average — in a single figure, which is exactly what a 24-panel grid of histograms could not give at a glance.

Part C — Quantifying Correlation

Covariance

Covariance measures whether two variables tend to be above their own means at the same time, or not:

$Cov(X, Y) = mean[(x_i - mean(x)) \times (y_i - mean(y))]$

- Positive covariance: when irradiance is above its average, power tends to be above its average too.
- Negative covariance: when one variable is above its average, the other tends to be BELOW its average — for example, module efficiency tends to dip when module temperature is unusually high.

The problem with covariance as a measure of strength is that its units are the PRODUCT of the two variables' units — here, $W/m^2 \times W$ — so its raw size says nothing about strength on its own. A covariance of 5,000 cannot be compared to a covariance of 12 without knowing the units involved.

Standard scores (z-scores)

```
z_irr = (df['IntSolIrr'] - df['IntSolIrr'].mean()) / df['IntSolIrr'].std()
z_pac = (df['Pac'] - df['Pac'].mean()) / df['Pac'].std()
```

A z-score (standard score) expresses how many standard deviations a value lies above or below its own mean: z = (x − mean) / std. After standardizing, EVERY variable has mean 0 and standard deviation 1, so W/m^2, watts, hertz and degrees Celsius all become directly comparable, unit-free numbers. Standardizing is the trick that turns covariance into an interpretable number: computing the covariance of the standardized variables gives Pearson's correlation coefficient.

Pearson's correlation coefficient

```
df[['IntSolIrr', 'TmpMdul C', 'Pac', 'Fac']].corr()

import numpy as np
np.corrcoef(df['IntSolIrr'], df['Pac'])[0, 1]
```

r = Cov(X, Y) / (std(X) × std(Y)) — the covariance of the standardized variables. r = +1 is a perfect increasing straight-line relationship; r = −1 is a perfect decreasing straight line; r = 0 means no LINEAR relationship, which is not necessarily “no relationship at all” (see the limitation below). As a rough guide, |r| above about 0.7 is usually called “strong,” 0.3–0.7 “moderate,” and below 0.3 “weak” — but the scatter plot should always be checked too, since a single number can hide a great deal.

Worked example: correlation matrix, 15-01-2013 (n = 288)

	IntSolIrr	TmpMdul C	Pac	Fac
IntSolIrr	1.000	0.883	0.997	0.721

	IntSolIrr	TmpMdul C	Pac	Fac
TmpMdul C	0.883	1.000	0.879	0.718
Pac	0.997	0.879	1.000	0.678
Fac	0.721	0.718	0.678	1.000

- Pac and IntSolIrr are almost perfectly linearly related (r = 0.997) — exactly what a solar inverter's operating principle predicts.
- TmpMdul C rises with IntSolIrr too (r = 0.883), but less tightly, because module temperature depends on more than sunlight alone (ambient conditions, thermal mass, cooling).
- Fac (grid frequency) correlates with everything at r ≈ 0.68–0.72 — a suspiciously similar value across very different physical quantities. Part C closes by showing why that number is misleading.

Limitation 1 — nonlinear relationships

```
# Simulated turbine: power is proportional to wind speed cubed
wind_speed = np.linspace(3, 14, 200)   # m/s, cut-in to rated
power = 0.5 * wind_speed**3 + noise
np.corrcoef(wind_speed, power)[0, 1]   # -> 0.93 (looks strong)

# But look closer: below ~5 m/s the turbine barely turns, and
# above ~12 m/s the curve flattens (rated power) -- neither
# region is captured by a single straight-line slope.
```

A wind turbine's power curve, $P \propto v^3$, is strongly and deterministically related to wind speed — but it is a CURVE, not a line. Pearson's r can still come out high on a smooth, monotonically increasing curve like this one, but it systematically understates the relationship near the cut-in and rated-power plateaus, where power barely changes even though wind speed does. The textbook worst case makes the point starkly: a symmetric parabola $y = x^2$ + noise gives Pearson's r ≈ 0 even though y is completely determined by x. Correlation measures LINEAR association only, and can miss a real relationship entirely.

Limitation 2 — outliers

Because Pearson's r is built from means and standard deviations, it is sensitive to exactly the same kind of extreme values that distort a mean (Lecture 2). A single misrecorded sensor spike — a momentary voltage transient logged as an implausibly large Pac value, for instance — can swing a correlation coefficient noticeably, even though it says nothing about the true relationship between the variables. The practical defense is to always look at the scatter plot alongside the number, and to consider computing r with and without suspected outliers to see how much it moves. This is also where a threshold-based cleaning step, like the sensor-noise threshold used for daylight detection in Lab 1, protects a correlation analysis, not only a mean.

Spearman's rank correlation

```
from scipy.stats import spearmanr

rho, p_value = spearmanr(df['IntSolIrr'], df['Pac'])
```

Spearman's rho replaces each value with its RANK within its own variable, then computes Pearson's r on the ranks. Because it only cares about order, Spearman's rho is robust to outliers — an extreme value is still just the highest rank, however extreme — and it correctly detects any MONOTONIC relationship, not only a linear one. It is the more honest choice whenever the true relationship is suspected to curve (like a turbine power curve) or the data contains occasional bad readings. A useful rule of thumb is to compute both: if Pearson's r and Spearman's rho are close, the relationship is close to linear; if they differ substantially, that gap is itself informative.

> *On the single day 15-01-2013: IntSolIrr vs. Fac gives Pearson r = 0.721 but Spearman ρ = 0.90 — a large gap that is a warning sign, investigated next.*

Correlation and causation: a real, cautionary example

Fac (grid frequency) correlated with IntSolIrr at r = 0.72 in the worked example above — strong enough to tempt a claim that sunlight somehow affects grid frequency. Looking closer at WHY reveals the trap: the inverter reports Fac = 0 exactly whenever it is switched off (no sunlight → no generation → idle), and a near-constant ≈50.00 Hz whenever it is switched on. Restricting the comparison to the moments the inverter is actually online (Fac > 0) — across all 288 × 25 = 6,350 readings, 2,137 of them with Fac > 0 — gives:

> **Pearson r(IntSolIrr, Fac | Fac > 0) = −0.02. Essentially zero.**

The original correlation of 0.72 was never about physics. It was a SHARED ON/OFF PATTERN: both variables happen to be “low” at night and “high” in daytime for entirely unrelated reasons — irradiance because of the sun, Fac because the inverter is running at all. This is a classic confound, and it is a useful habit to check whenever two signals that share an operating schedule (on/off, weekday/weekend, seasonal) show a suspiciously strong correlation.

Part D — Simple Linear Regression

scipy.stats.linregress

```
from scipy.stats import linregress

result = linregress(df['IntSolIrr'], df['Pac'])
result.slope, result.intercept, result.rvalue,
result.pvalue, result.stderr
```

- linregress fits y = slope × x + intercept by least squares, minimizing the sum of squared vertical distances from each point to the line.
- rvalue is exactly Pearson's r for the two variables — regression and correlation are two views of the same linear relationship.
- rvalue**2 (“R-squared”) is the fraction of the variance in y that is explained by a linear function of x.
- pvalue tests the null hypothesis that the true slope is zero — useful, but with thousands of sensor readings, even a tiny, practically unimportant slope will often be “statistically significant.”

Worked example: Pac ~ IntSolIrr, across all 25 days (n = 6,350)

Subset	slope (W per W/m^2)	intercept (W)	r	r^2
All readings	0.248	−0.37	0.870	0.757
Daylight only (IntSolIrr > 1)	0.250	−1.33	0.828	0.686
Module temp. ~ irradiance (daylight)	0.021 °C per W/m^2	16.78	0.705	0.497

- About 0.25 W of AC output for every additional W/m^2 of irradiance, whether or not the (mostly-zero) night readings are included — the relationship is dominated by the daytime data either way.
- r^2 ≈ 0.69–0.76: roughly three-quarters of the variation in output power is explained by irradiance alone across a whole month of mixed weather; the remainder comes from cloud transients, temperature derating, and inverter behavior.
- Module temperature climbs more gently with irradiance (r^2 ≈ 0.50) — a real, moderate, but far less tight relationship than power itself.

Correlation strength is not the same as practical importance

Dataset	correlation r	slope	effect over 30 years
Site A — panel degradation vs. age	0.758	0.019 %/yr	0.56 % lost over 30 yr
Site B — panel degradation vs. age	0.478	0.176 %/yr	5.29 % lost over 30 yr

This pattern — adapted from Think Stats' two “fake dataset” example, reframed here for panel-degradation monitoring — is a real trap: Site A has the STRONGER correlation, but Site B has the more important, more expensive practical effect. The reason is that correlation reflects how TIGHTLY the points cluster around a line, while the regression slope reflects how STEEP that line is: two independent properties of the same scatter plot. A noisier relationship (lower r, more scatter around the line) can still carry a much larger, more consequential slope than a very tight, clean-looking one. The practical rule is to always report both numbers — correlation and slope — and never r alone.

Wrap-up

- Scatter plots reveal a relationship; alpha, markersize, and jittering keep them readable at real-world data volumes.
- Violin and box plots compress many groups' distributions into one comparable figure.
- Covariance, z-scores, and Pearson's r turn “looks related” into a single, comparable number — but one that only captures LINEAR association and is sensitive to outliers.
- Spearman's rank correlation, a look at the scatter plot, and healthy suspicion of shared on/off patterns all guard against being misled.
- Simple linear regression fits and quantifies that line — and correlation strength is not the same question as whether the slope matters in practice.

Coming up in Lecture 5: from describing relationships to estimation and hypothesis testing — how confident can we be in a correlation or a slope computed from a sample of readings?