Statistics and Data Analysis
Unit 01 – Lecture 02 Notes
Feature Scaling and Feature Engineering Basics

Tofik Ali

February 17, 2026

What You Will Learn

In this lecture we focus on practical preprocessing steps that are used *before* modeling:

- scaling (normalization and standardization),
- transformations (especially for skewed variables),
- and basic feature engineering (creating helpful new variables).

1. Feature Scaling

1.1 Why scaling is needed

Many datasets contain features with different units and magnitudes, for example:

- attendance (%) typically ranges 0–100,
- study hours per week might range 0–25,
- income could range 20–200 (in thousands) or much more.

If we use distance (like Euclidean distance), the largest-scale feature can dominate. This is why scaling is important for:

- kNN (nearest neighbors),
- k-means clustering,
- many gradient-based optimization methods.

1.2 Min–max normalization

Min–max scaling maps a feature into $[0, 1]$:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Interpretation: $x' = 0$ means the minimum observed value; $x' = 1$ means the maximum observed value.

Exercise 1 (solution). If min=20, max=80, and $x = 50$:

$$x' = \frac{50 - 20}{80 - 20} = \frac{30}{60} = 0.5$$

1.3 Standardization (z-score)

Standardization uses mean and standard deviation:

$$z = \frac{x - \mu}{\sigma}$$

Interpretation: z is the number of standard deviations x is above/below the mean.

Exercise 2 (solution). If $\mu = 60$, $\sigma = 10$, and $x = 80$:

$$z = \frac{80 - 60}{10} = 2$$

So the score 80 is 2 standard deviations above the mean.

1.4 Important warning: data leakage

When we scale, we compute parameters (min/max or mean/std). These parameters must be estimated on the **training data only**. Then we use the same parameters to transform the test set.

If we use the whole dataset (including test) to compute scaling parameters, then test information "leaks" into training. That can make performance look better than it truly is.

Exercise 3 (solution).

- kNN: Yes (distance-based)
- k-means: Yes (distance-based)
- Linear regression (gradient descent): often yes (helps optimization)
- Decision tree: usually no (split rules are scale-invariant)

2. Transformations

2.1 Why transform?

Scaling changes the *units* or spread, but it does not necessarily fix skewness. For right-skewed features (like income), many values are small and a few values are extremely large.

Transformations can:

- reduce skewness,
- make relationships more linear,
- reduce the impact of extreme values.

2.2 Log transform

A common transform for positive values is:

$$x' = \log(1 + x)$$

We use $\log(1 + x)$ instead of $\log(x)$ to handle zero values safely.

Exercise 4 (solution). For right-skewed income, a reasonable first transform is $\log(1 + x)$.

3. Feature Engineering Basics

3.1 What is feature engineering?

Feature engineering means creating new variables (features) from raw columns. Good engineered features can:

- capture patterns more directly,
- improve model performance,
- and make interpretations clearer.

3.2 Common patterns

- **Encoding categorical variables**: one-hot encoding for nominal categories.
- **Buckets/bins**: convert continuous variables to categories (e.g., attendance low/medium/high).
- **Interactions**: multiply/add features when combined effect matters (effort = hours $\times$ attendance).
- **Ratios**: e.g., marks per hour, cost per unit.
- **Date parts**: month, weekday, time-of-day.
- **Text features**: word count, length, keyword flags (basic features).

Exercise 5 (example answers).

- `income_log1p`: reduces skew and outlier impact.
- `effort_index` = hours $\times$ attendance/100: single "effort" score.
- `join_month`: captures seasonal effects.

Exercise 6 (solution). If `program` has categories CSE/ECE/AIML, the one-hot columns can be:

`program_CSE`, `program_ECE`, `program_AIML`

Each row has exactly one 1 (its category) and 0 for the others.

4. Mini Demo (Python)

Run from the lecture folder:

```
python demo/scaling_feature_engineering_demo.py
```

The demo:

- scales three features (attendance, hours, income) and prints how distances change after scaling,
- engineers features (month, weekday, income log, effort index, backlog flag, one-hot program),
- saves `data/student_features_engineered.csv`,
- saves plots in `images/`.

References

- McKinney, W. *Python for Data Analysis.* O'Reilly, 2022.
- Kuhn, M., & Johnson, K. *Feature Engineering and Selection.* CRC Press, 2019.
- Montgomery, D. C., & Runger, G. C. *Applied Statistics and Probability for Engineers.* Wiley, 7th ed., 2020.

Appendix: Slide Deck Content (Reference)

The material below is a reference copy of the slide deck content. Exercise solutions are explained in the main notes where applicable.

Title Slide

https://github.com/tali7c/Statistics-and-Data-Analysis

Quick Links

Scaling Transforms Feature Engg. Demo Summary

Agenda

- Overview
- Feature Scaling
- Transformations
- Feature Engineering Basics
- Demo
- Summary

Learning Outcomes

- Explain why feature scaling is needed in many ML/analytics workflows
- Compute min–max normalization and z-score standardization
- Choose appropriate scaling/transformation for a given scenario
- Create basic engineered features (encoding, bins, interactions, date parts)

Why Scale Features?

- Different features can have very different units and ranges
- Distance-based methods (kNN, k-means) are dominated by large-scale features
- Optimization can be faster/more stable after scaling (gradient-based models)
- Scaling also helps compare feature importance in some linear models

Min–Max Normalization

Maps a feature to $[0, 1]$:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- Keeps relative ordering
- Sensitive to extreme outliers (min/max can be unstable)

Standardization (z-score)

Centers and scales using mean and standard deviation:

$$z = \frac{x - \mu}{\sigma}$$

- Produces mean 0 and std 1 (approximately)
- Often a good default for many algorithms

Important: Avoid Data Leakage

- Scaling parameters (min/max, mean/std) must be computed on **training data only**
- Then apply the same parameters to validation/test data
- Otherwise, test information leaks into training and results look unrealistically good

Exercise 1: Min–Max

Suppose a feature has min = 20, max = 80.
Compute the min–max normalized value for $x = 50$.

Solution 1

$$x' = \frac{50 - 20}{80 - 20} = \frac{30}{60} = 0.5$$

Exercise 2: z-score

A test score has mean $\mu = 60$ and standard deviation $\sigma = 10$.
Compute the z-score for $x = 80$ and interpret it.

Solution 2

$$z = \frac{80 - 60}{10} = 2$$

Interpretation: 80 is 2 standard deviations above the mean.

Exercise 3: Who Needs Scaling?

For each algorithm, answer: scaling important? (Yes/No)

1. kNN
2. k-means clustering
3. Linear regression (with gradient descent)
4. Decision tree

Solution 3

- kNN: Yes (distance-based)
- k-means: Yes (distance-based)
- Linear regression (GD): Often yes (helps optimization)
- Decision tree: Usually no (splits are scale-invariant)

Why Transform Features?

- Reduce skewness (e.g., income is often right-skewed)
- Stabilize variance and make patterns more linear
- Improve interpretability and model fit in some cases

Log Transform (Common for Right-Skew)

For positive values, a common transform is:

$$x' = \log(1 + x)$$

- Compresses large values and spreads small values
- Reduces the impact of extreme values

Exercise 4: Pick a Transform

Which is a reasonable first transform for a right-skewed income column?
(a) x (b) x^2 (c) $\log(1 + x)$

Solution 4

(c) $\log(1 + x)$ is a common choice for right-skewed positive features.

What is Feature Engineering?

Feature engineering means creating useful new variables from raw data to improve analysis/models.

- Encode categorical variables (one-hot)
- Create buckets/bins (low/medium/high)
- Interactions and ratios (effort = hours $\times$ attendance)
- Date parts (month, weekday)
- Simple text features (length, keywords)

Example (Student Dataset Columns)

Raw columns:

`program, city, join_date, attendance_pct, study_hours_week, cgpa, family_income_k, backlogs`

Possible engineered features:

- `join_month`, `join_weekday`
- `attendance_bucket` (low/medium/high)
- `income_log1p`
- `effort_index` = hours $\times$ attendance/100
- `has_backlog` (0/1)

Exercise 5: Design Features

You have: `attendance_pct, study_hours_week, family_income_k, join_date, program`.

Task: Propose **three** engineered features and explain why they might help.

Solution 5 (Examples)

- `income_log1p`: reduces right-skew and outlier impact
- `effort_index`: combines hours + attendance into one effort score
- `join_month`: captures seasonal patterns (admissions, attendance patterns)

Exercise 6: One-hot Encoding

Program has categories: `CSE`, `ECE`, `AIML`.
Task: Write the one-hot encoded columns for `program`.

Solution 6

One-hot columns:

- `program_CSE`
- `program_ECE`
- `program_AIML`

Each row has 1 for its category and 0 for the others.

Mini Demo (Python)

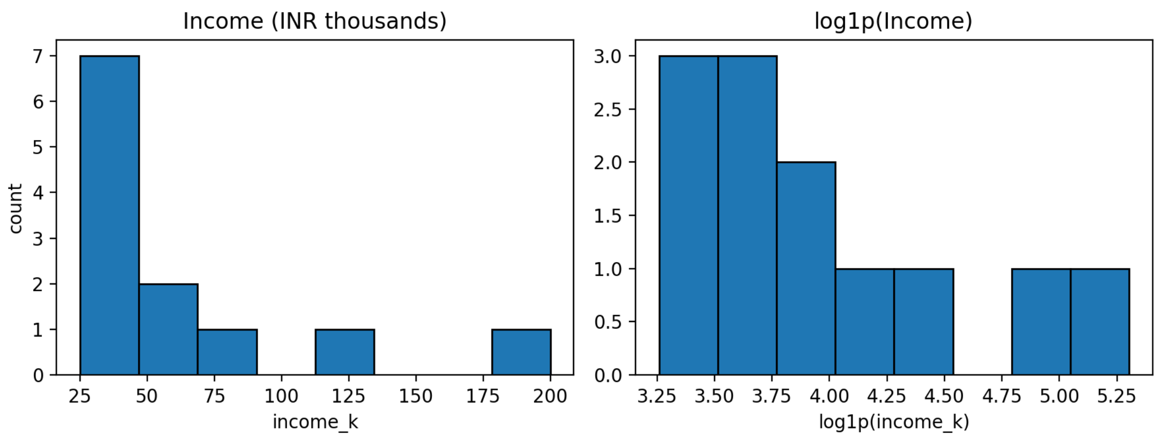
Run from the lecture folder:

```
python demo/scaling_feature_engineering_demo.py
```

Outputs:

- `data/student_features_engineered.csv`
- plots in `images/` (income vs log-income, hours vs CGPA)

Demo Output (Example Plot)



Summary

- Scaling makes features comparable and prevents domination by large-scale variables
- Min–max maps to $[0, 1]$; z-score centers and scales by mean/std
- Transformations like $\log(1 + x)$ help reduce skew and outlier influence
- Feature engineering creates informative variables (encoding, bins, interactions, dates)

Exit question: Why is scaling “fit on train only” and then applied to test?