Statistics and Data Analysis

Unit 01 – Lecture 02: Feature Scaling and Feature Engineering Basics

Tofik Ali

School of Computer Science, UPES Dehradun

February 14, 2026

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

Quick Links

Scaling Transforms Feature Engg. Demo Summary

Agenda

Learning Outcomes

- Explain why feature scaling is needed in many ML/analytics workflows

Learning Outcomes

- Explain why feature scaling is needed in many ML/analytics workflows
- Compute min–max normalization and z-score standardization

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

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

Why Scale Features?

- Different features can have very different units and ranges
- Distance-based methods (kNN, k-means) are dominated by large-scale features

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)

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

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)

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**

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

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)

Why Transform Features?

- Reduce skewness (e.g., income is often right-skewed)
- Stabilize variance and make patterns more linear

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

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)

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)

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)

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)

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`

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)

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`

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

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 × 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)

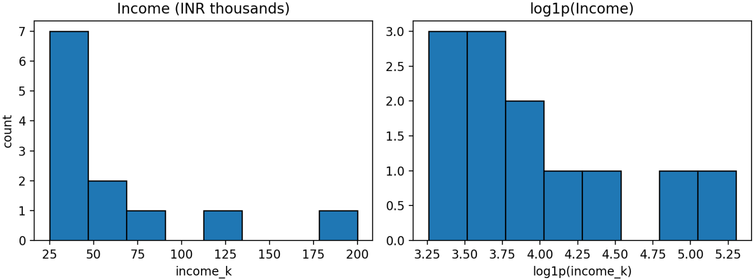
Run from the lecture folder:

```
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

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

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

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

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

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

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?