DATA PREPARATION
FOR MACHINE LEARNING

From raw, messy data to model-ready features

SELECT | **CLEAN** | **TRANSFORM**

Dr. D Bhanu Prakash, Assistant Professor, Department of Mathematics and Computer Science

Reference: Jason Brownlee — Data Preparation for Machine Learning (Machine Learning Mastery)

Today's Roadmap

Three stages, one real dataset, hands-on the whole way

1

Data Preparation

The big picture — a 3-step framework and the golden rule: avoid data leakage.

2

Data Cleaning

Missing values, outliers, duplicates, and uninformative columns — found and fixed.

3

Data Transformation

Scaling, encoding, discretization, and power transforms that reshape features for learning.

Why Bother?

80%

of a data scientist's time is typically spent preparing data — not tuning models.

GIGO

Garbage In, Garbage Out

A powerful algorithm trained on messy, biased, or badly-scaled data will confidently produce garbage predictions. No amount of model sophistication compensates for poor data preparation.

Brownlee's 3-Step Framework

Preparation is a pipeline, not a checklist — order matters

1

Data Selection

Which rows and columns are actually relevant?

→

2

Data Cleaning

Is the data correct and complete?

→

3

Data Transformation

Is it in the right shape and scale for the algorithm?

Raw Data → [SELECT] → [CLEAN] → [TRANSFORM] → Model-Ready Data

The Golden Rule: Avoid Data Leakage

Any statistic used to clean or transform data — a mean, a min/max, a most-frequent category — must be learned ONLY from the training set, then applied to validation/test data. Never compute it on the full dataset before splitting.

✗ WRONG

1. Combine train + test into one dataframe
2. Fit scaler / imputer on the WHOLE dataset
3. Split into train / test
4. Train model

Test statistics leaked into training — accuracy looks better than it truly is.

✓ RIGHT

1. Split into train / test FIRST
2. Fit scaler / imputer on TRAIN only
3. Transform train and test using those fitted stats
4. Train model on transformed train set

scikit-learn's Pipeline does this automatically.

Our Lab Specimen: Pima Indians Diabetes

A small, real, delightfully messy medical dataset (courtesy of Jason Brownlee's dataset archive)

768
patient rows

8
numeric features

1
target: diabetes class

8 Features

Pregnancies • Plasma Glucose • Blood Pressure

Skin Fold Thickness • Insulin • BMI

Diabetes Pedigree Function • Age

? The Trap

Several columns use 0 to represent a MISSING value — a blood pressure of 0 is medically impossible. Pandas' isnull() won't catch it. Can you spot which columns before Section 2?

PART 1

Data Cleaning

Fixing what's wrong before we ask a model to learn from it

Missing Values: The Silent Problem

df.isnull().sum() reports ZERO missing values here. Look closer.

Column	Minimum value	Biologically plausible?
plas — Plasma glucose	0	✗ No living patient has 0 glucose
pres — Blood pressure	0	✗ Impossible
skin — Skin fold thickness	0	✗ Impossible
insu — Insulin	0	✗ Impossible
mass — BMI	0	✗ Impossible
preg — Pregnancies	0	✓ Yes, genuinely zero
age — Age	21	✓ Yes

Three Strategies for Missing Data

Listwise Deletion

WHAT

Remove any row with a missing value.

WHEN

Best when missingness is rare (<5%) and random.

RISK

Dropping our data loses 376 of 768 rows (49%)!

Column Deletion

WHAT

Drop an entire column.

WHEN

Best when a column is mostly missing (e.g. >50%).

RISK

Loses a potentially predictive feature entirely.

Imputation

WHAT

Fill with a plausible estimate (mean, median, KNN...).

WHEN

Best for moderate missingness — our approach today.

RISK

Introduces some artificial certainty into the data.

Imputation in Practice

SimpleImputer (mean / median)

Replace every missing value with one global statistic for that column.

- Mean — use on roughly symmetric distributions
- Median — use on skewed data / outliers (more robust)

KNNImputer (context-aware)

Look at the k most similar rows (by other features) and impute using their average.

More realistic estimates than a single global number — at a higher compute cost.

How to choose?

Check skewness first:

skew(insu) ≈ 2.27

skew(mass) ≈ 0.60

Values far from 0 → skewed → prefer median (or KNN) imputation over the mean, which is pulled toward the long tail.

⚠️ *Fit on training data only — see the golden rule.*

Detecting Outliers

An outlier is an observation abnormally distant from the rest — real, or an error?

Std-Dev Rule

mean ± 3σ

Assumes roughly Normal data. Simple, fast, one column at a time.

IQR Rule

Q1 − 1.5·IQR to Q3 + 1.5·IQR

Robust, no normality assumption — the same rule your box-plot whiskers use.

Isolation Forest

multivariate, model-based

Detects outliers using ALL features jointly, not one column at a time.

! *Never auto-delete outliers without inspecting them. In medical data, an 'outlier' patient may be exactly the case your model most needs to learn from.*

Duplicates & Uninformative Columns

Duplicate Rows

Duplicate rows often come from pipeline bugs — a double form submission, a re-run ETL job. They silently give some observations extra 'voting power' during training, skewing what the model learns as typical.

```
df.duplicated().sum()
df.drop_duplicates()
```

Zero / Low-Variance Columns

A column with the same value in every row carries zero information — it can never help distinguish outcomes. Wide, sensor- or log-derived datasets are especially prone to this.

```
VarianceThreshold(threshold=0.0)
.fit(X)
```

✅ Cleaning Checkpoint

Problem	Technique	Result
Disguised missing (zeros)	Domain-knowledge replace → NaN	5 columns flagged
Missing values	Median / KNN imputation	0 missing values remain
Outliers	IQR + Isolation Forest	Identified & inspected, not blindly deleted
Duplicate rows	drop_duplicates()	Removed
Zero-variance columns	VarianceThreshold	None found in this dataset

df_imputed is now our CLEAN dataset — next, we reshape it for learning algorithms.

PART 2

Data Transformation

Reshaping clean features into the form learning algorithms need

Feature Scaling

preg ranges 0–17, insu ranges up to ~850 — distance & gradient-based models are sensitive to that gap

Technique	Formula	Resulting range	Outlier-sensitive?
Normalization (MinMaxScaler)	(x − min) / (max − min)	exactly [0, 1]	Yes — one extreme value stretches the whole scale
Standardization (StandardScaler)	(x − mean) / std	mean 0, std 1, unbounded	Somewhat
Robust Scaling (RobustScaler)	(x − median) / IQR	centered on median, unbounded	No — built for this

Tree-based models (Decision Trees, Random Forests, Gradient Boosting) split on per-feature thresholds — they don't need scaling at all. Scaling changes the numbers, never the shape of the distribution: a skewed feature stays skewed after scaling.

Encoding Categorical Variables

Example: turning BMI into a clinical category

Ordinal Encoding

Maps categories to integers 0, 1, 2, ...
Use ONLY when categories have a genuine order.

Category	Code
Underweight	0
Normal	1
Overweight	2
Obese	3

One-Hot Encoding

Creates one binary column per category.
Use for NOMINAL (unordered) categories.

Category	Under	Normal	Over	Obese
Normal	0	1	0	0
Obese	0	0	0	1

⚠️ *Never one-hot-encode with plain integers for unordered categories — the model wrongly assumes Blue > Red.*

Discretization & Power Transforms

Discretization (Binning)

Converts a continuous variable into discrete buckets — helps simple models capture non-linear jumps and aids interpretation.

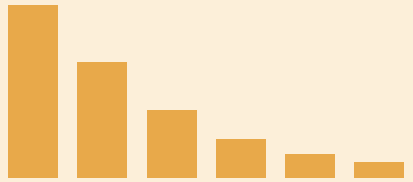
`quantile`	roughly equal COUNT per bin
`uniform`	roughly equal WIDTH per bin
`kmeans`	natural groupings / similarity

Power Transforms

Many models assume roughly Normal (bell-curve) features. Power transforms pull in a long tail to restore symmetry.

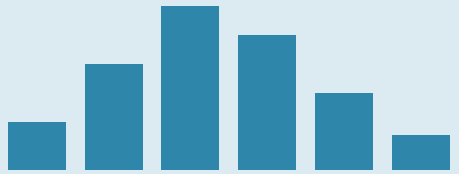
skew = 2.27

insu
(before)



skew ≈ 0.1

insu
(Yeo-Johnson)



Box-Cox needs strictly positive data; Yeo-Johnson also handles zero/negative values.

Bonus: Dimensionality Reduction with PCA

Principal Component Analysis re-expresses correlated features using a smaller number of synthetic 'components' that capture most of the original variance.

Why use it?

- Speeds up training
- Reduces overfitting risk
- Enables 2D visualization of high-dimensional data

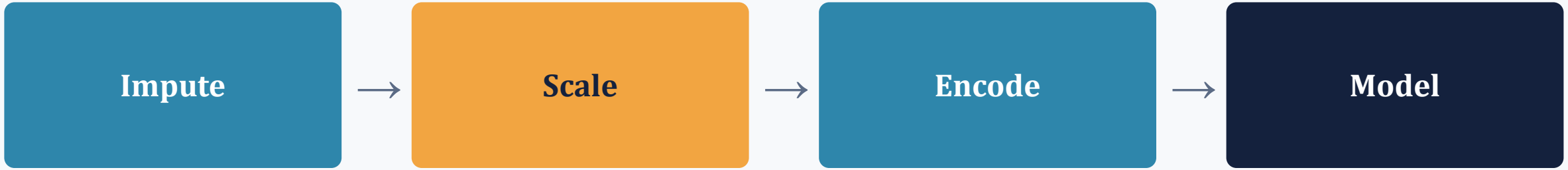
⚠️ *Always standardize features BEFORE PCA — otherwise the largest-range feature dominates every component.*

8 features → 2 components

PC1 + PC2 together explain roughly the majority of total variance in this dataset — enough to see clear structure in a 2D scatter plot colored by diabetes class.

Assembling a Leak-Free Pipeline

scikit-learn's Pipeline + ColumnTransformer bundle every step into one fittable object

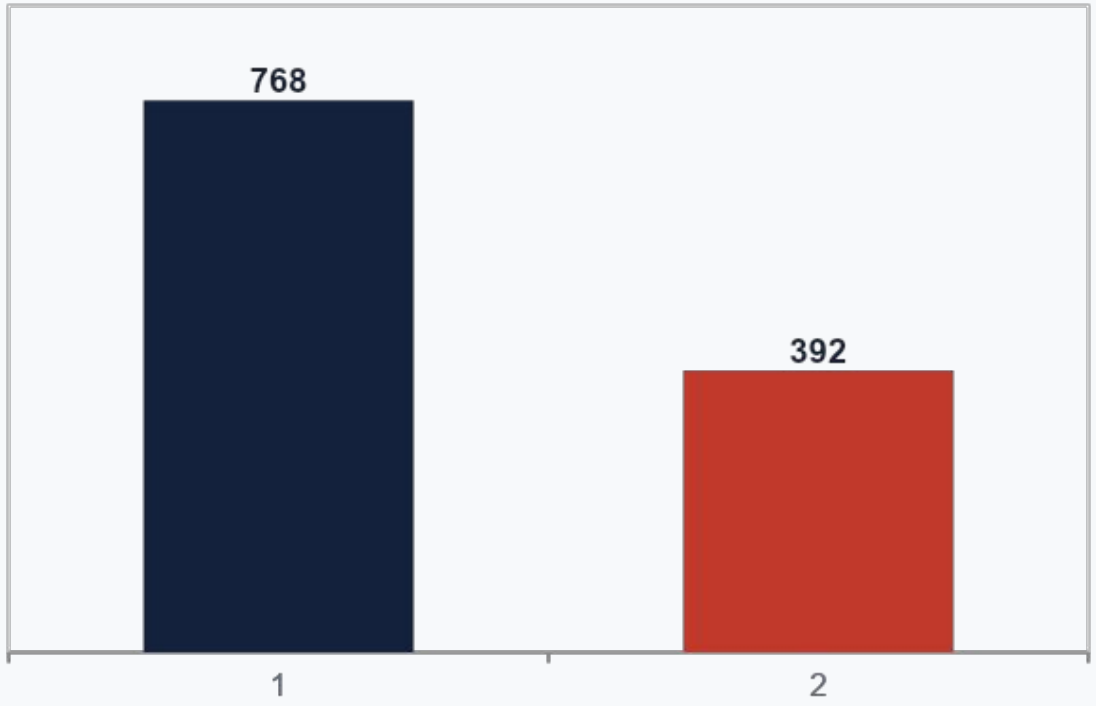


Fit .fit() ONLY on training data → the same fitted transformers apply to test / new data via .transform() / .predict()

```
Pipeline([
  ('impute', SimpleImputer(strategy='median')),
  ('scale', StandardScaler()),
  ('model', LogisticRegression())
])
```

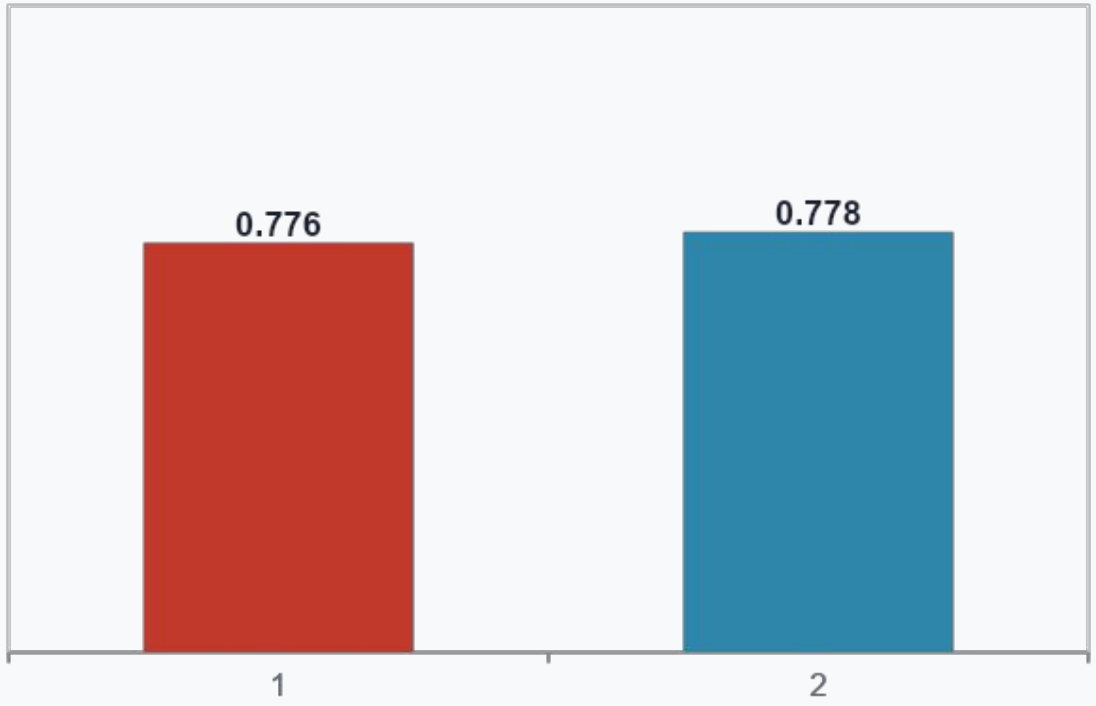
Does It Actually Work? The Numbers.

Naive listwise deletion loses HALF the dataset



376 rows (49%) gone — a steep price for the naive approach.

Cross-validated accuracy: naive fill vs. proper cleaning



A modest, honest gain here — but on larger, messier data and scale-sensitive models, the gap widens dramatically.

Cheat Sheet

Stage	Problem	Tool(s)
Cleaning	Missing values	replace(), SimpleImputer, KNNImputer
Cleaning	Outliers	IQR rule, z-score rule, IsolationForest
Cleaning	Duplicates	duplicated(), drop_duplicates()
Cleaning	Uninformative columns	VarianceThreshold
Transform	Different scales	MinMaxScaler, StandardScaler, RobustScaler
Transform	Categorical variables	OrdinalEncoder, OneHotEncoder
Transform	Continuous → categorical	KBinsDiscretizer
Transform	Skewed distributions	PowerTransformer (Box-Cox / Yeo-Johnson)
Assembly	Avoiding leakage	Pipeline, ColumnTransformer

In-Class / Homework Exercises

Open the companion notebook and try these before next lecture

1. Count outliers in `skin` before and after median imputation — does imputation hide or create outliers?

2. Compare three pipelines: drop rows / mean impute / KNN impute. Which wins on cross-validated accuracy, and why?

3. Train a Decision Tree on scaled vs. unscaled features. Explain why accuracy barely moves — unlike Logistic Regression.

4. Engineer a `glucose_risk` category from `plas` using real clinical thresholds, one-hot encode it, and add it to the pipeline.

5. Try Box-Cox instead of Yeo-Johnson on `insu`. Read the resulting error — explain in your own words why it fails here.

References & Thank You

- Brownlee, J. — Data Preparation for Machine Learning. Machine Learning Mastery.
- Brownlee, J. — Pima Indians Diabetes Dataset. github.com/jbrownlee/Datasets
- Kuhn, M. & Johnson, K. — Feature Engineering and Selection: A Practical Approach for Predictive Models.
- scikit-learn User Guide — Preprocessing data & Imputation of missing values (scikit-learn.org).

Remember: 80% of the work is preparation. Do it well, and the model has a fighting chance.