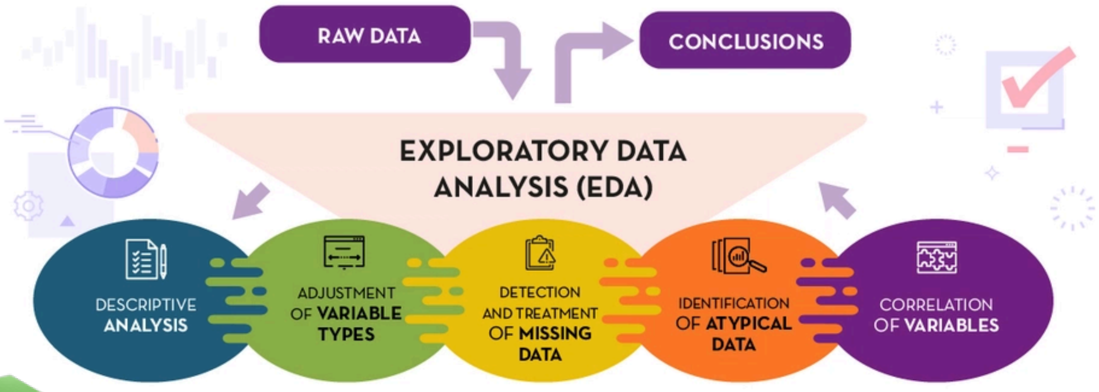


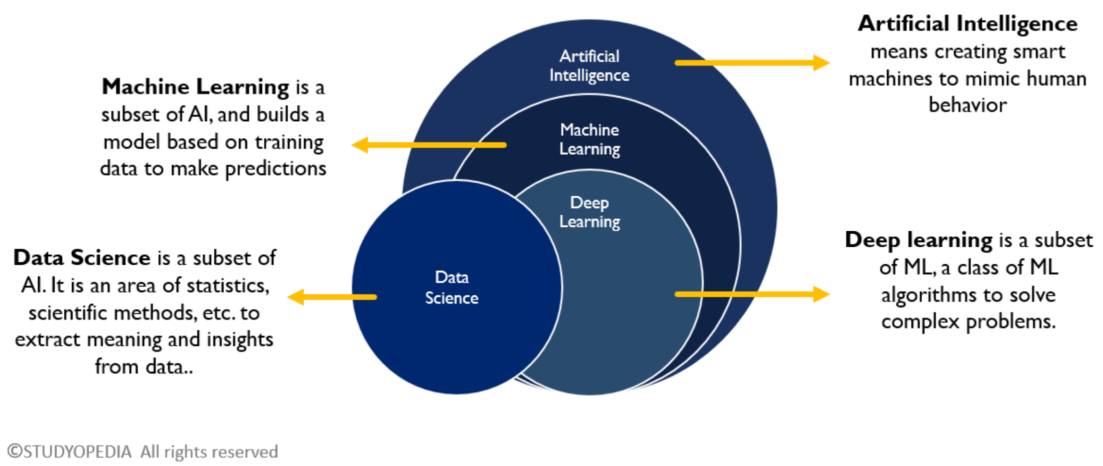
MOOC 1 — Exploratory Data Analysis for Machine Learning

Original Notion Document : https://www.notion.so/MOOC-1-Exploratory-Data-Analysis-for-Machine-Learning-27cec520737c8017b4a1ec2b8ee23499?source=copy_link

Quick Table of Contents

▼ Detailed Table of Contents

Module 1: Exploratory Data Analysis for Machine Learning – Study Notes

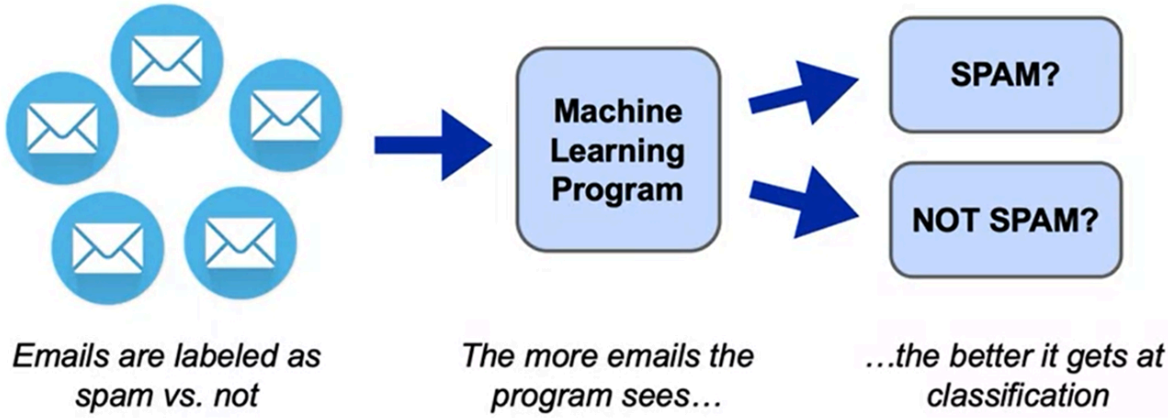


1. Artificial Intelligence (AI)

- **Concept:** Programs simulating intelligent behavior: sensing, reasoning, acting, adapting.
- **Details:** Not all AI learns from data; some are rule-based.
- **Examples:** Virtual assistants (Siri, Google Assistant), chatbots, rule-based systems.

- **Notes:** Subsets include ML and DL; modern AI often relies on learning from data.
- **Tip:** AI = “machines performing human-like intelligence” (Turing Test).

2. Machine Learning (ML)



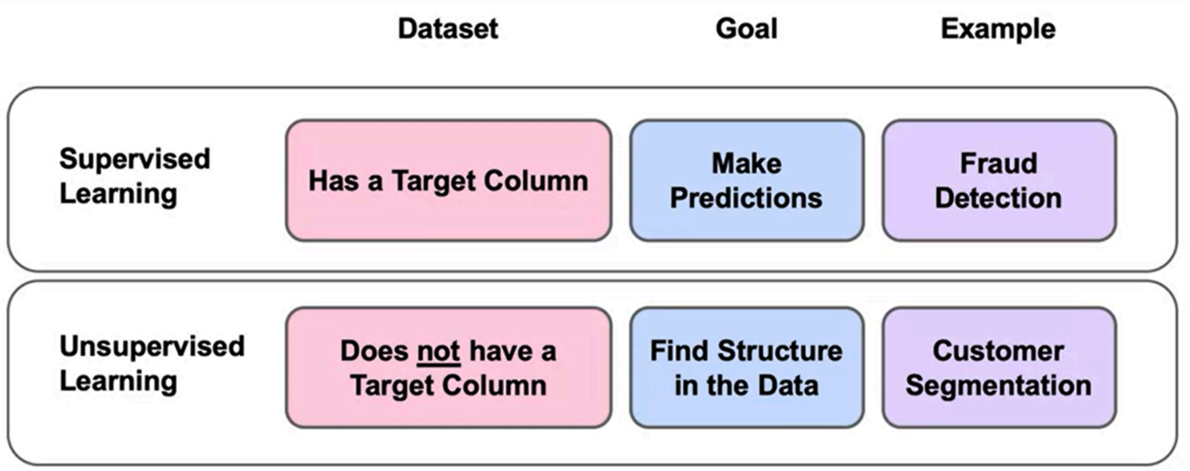
2.1 Concept

- Programs learn patterns from data without explicit programming.
- More data → better learning, eventually plateaus.
- Subset of AI, includes classical ML & DL.

2.2 Key Terms

- **Feature:** Input variable used to predict target (e.g., sepal/petal length, transaction amount, pixel values).
- **Target / Label:** Output to predict (e.g., species, spam/not spam, fraud/not fraud).

2.3 Types of ML

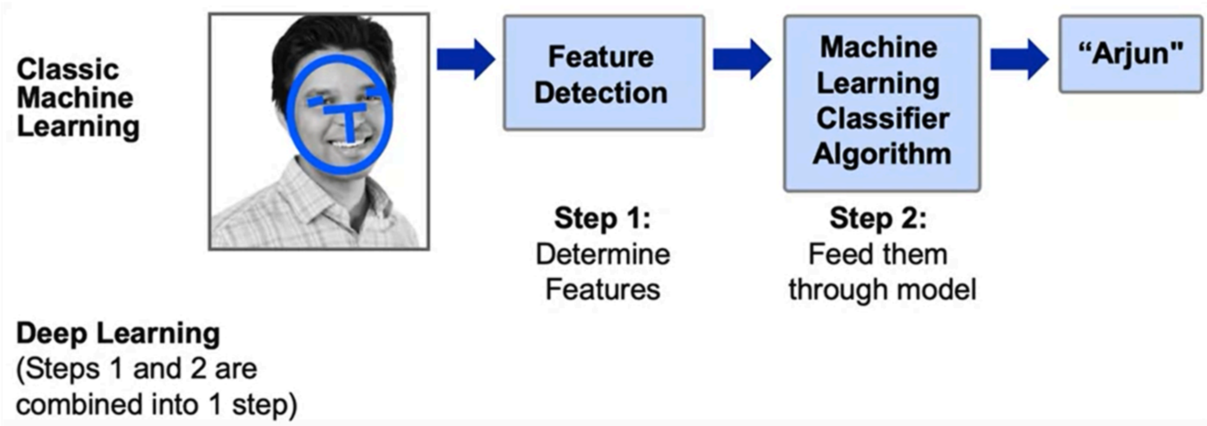


- **Supervised Learning:** Labeled data → predict target.
  - Examples: spam detection, fraud detection, iris classification.
  - Goal: predict labels for new data.
  - Tip: Like teaching the machine with exercises that have answers.
- **Unsupervised Learning:** No labels → find hidden structure.
  - Examples: customer segmentation, clustering.
  - Goal: identify patterns or natural groupings.
  - Tip: Like exploring data to find natural clusters.

2.4 Feature Engineering & Limitations

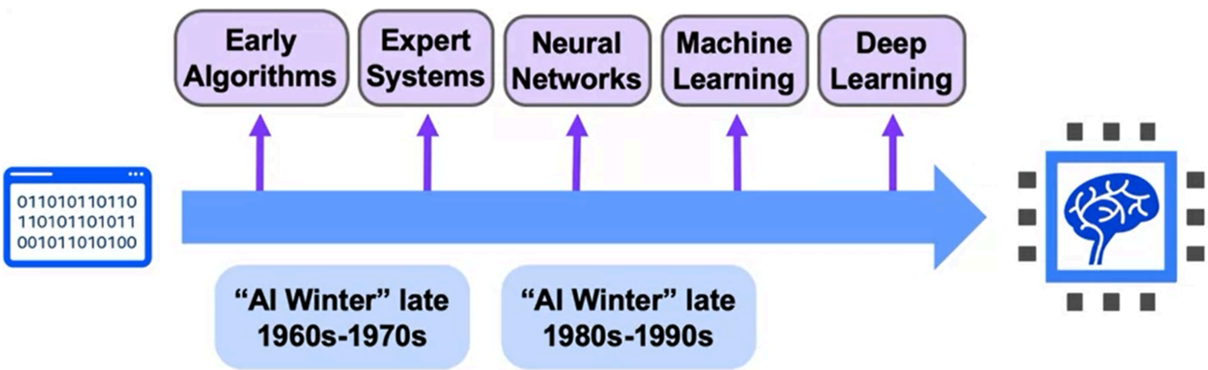
- Structured data: easier feature selection → classical ML works well.
- Unstructured data (images, audio, text): feature selection difficult → DL preferred.
- Example: Image 256×256 pixels → 65,000 features → classical ML loses spatial relationships.

3. Deep Learning (DL)



- Classical ML struggles with images:
  - Pixels as features → very high dimensionality.
  - Loses spatial relationships.
  - Requires manual feature engineering → difficult.
- **DL Advantages:**
  - Uses deep neural networks.
  - Learns features automatically.
  - Preserves spatial relationships.
- **Layer Workflow:** edges → shapes → high-level objects → prediction.

4. History of AI



4.1 Foundations & Early Excitement (1950s–1990s)

- **1950:** Alan Turing → Turing Test (can machines imitate humans?).
- **1956:** Dartmouth Conference → AI coined; goal: simulate intelligent behavior.

- **1957:** Rosenblatt → Perceptron → first neural network, learned from data.
- **1959:** Arthur Samuel → Checkers program → learns from past moves; popularized Machine Learning.

4.2 First AI Winter (1960s–1970s)

- Machine translation fails (Russian↔English) → low ROI → US government scrutiny.
- **1969:** Minsky → limitations of perceptron.
- **1973:** Lighthill report → research fell short → major funding cuts.

4.3 1980s – Expert Systems Boom

- Rule-based systems mimicking human experts.
- Ran on mainframes using LISP.
- Applied in business for decision-making → AI practically useful.

4.4 Neural Network Progress

- Geoffrey Hinton → Backpropagation algorithm → multi-layer networks can learn.
- Theoretical breakthroughs → potential for deep learning.

4.5 Second AI Winter (late 1980s–1990s)

- Expert systems unable to learn; brittle with unusual inputs.
- Neural networks couldn't scale; backpropagation had problems with large datasets.
- Investment decline as AI hype subsided.

5. Modern AI

- **Growth Areas:** Computer Vision (self-driving cars, medical imaging), NLP (translation, sentiment, text generation).
- **Why This Era is Different:** Larger datasets (ImageNet), faster & cheaper computing, neural networks & deep learning → practical applications.

5.1 Industry Applications

- **Healthcare:** Diagnosis, drug discovery, sensory aids.
- **Industrial:** Automation, predictive maintenance, agriculture.
- **Finance:** Trading, fraud detection, risk management.
- **Energy:** Smart grids, resource extraction, conservation.
- **Government:** Defense, citizen services, smart cities.
- **Transportation:** Autonomous vehicles, logistics, drones.
- **Education & Entertainment:** Personalized learning, gaming, media.

6. AI in Daily Life

6.1 Transportation

- Route optimization: Google Maps, Waze predict traffic, fastest routes.
- Ride-sharing: Uber, Lyft adjust pricing in real-time (supply & demand).

6.2 Social Media

- Content personalization: posts, groups, targeted ads.
- Sentiment analysis: reviews or content mood.
- Image recognition: face/object tagging and sharing.

6.3 Voice & NLP

- Virtual assistants: Siri, Alexa understand commands.
- NLP: translation, sentiment detection, task automation.

6.4 Computer Vision & Object Detection

- Self-driving cars: live detection of objects, pedestrians, road conditions.
- Security: detect abandoned baggage in real time.
- Image classification: DL can outperform humans.

7. Machine Learning Workflow

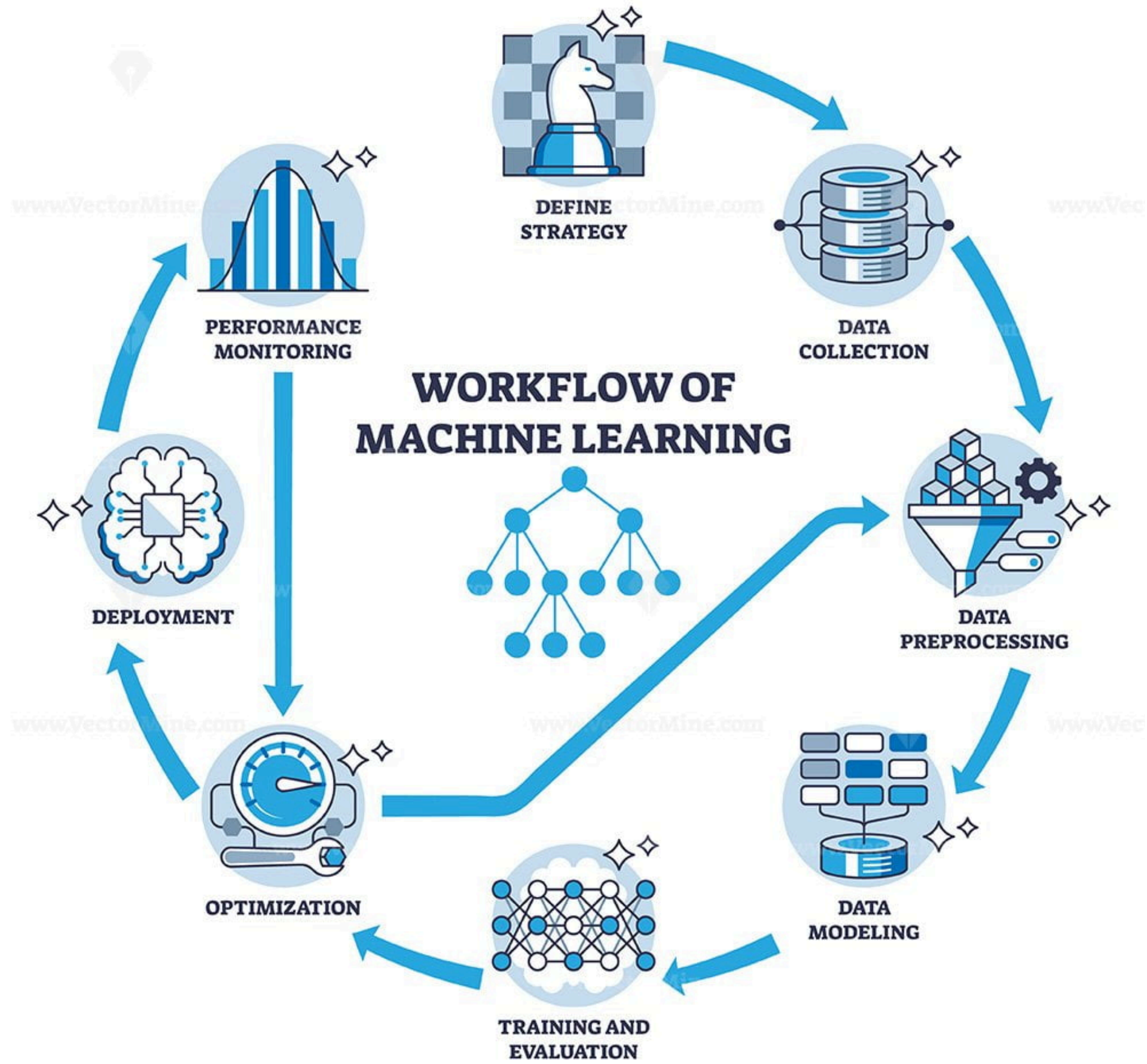
7.1 Prerequisites and Tools

- Python libraries: NumPy, Pandas, Matplotlib, Seaborn, Scikit-Learn, TensorFlow, Keras.

- Environment: Jupyter Notebook / iPython.
- Math background: basic statistics, probability, Bayes’ rule, linear algebra.

7.2 Typical Workflow

1. **Problem Statement:** Define the problem (e.g., classify dog breeds).
2. **Data Collection:** Gather labeled data (variety of angles, lighting, etc.).
3. **Data Exploration & Preprocessing:**
    - Clean data, handle missing values.
    - Analyze distributions, correlations, heatmaps.
    - Convert data into suitable formats (e.g., pixel arrays).
4. **Modeling:**
    - Build baseline models.
    - Train on training data.
5. **Validation:**
    - Evaluate using holdout/test set.
    - Measure accuracy or other metrics.
6. **Decision-making & Deployment:**
    - Communicate results, deploy model.



7.3 Basic ML Vocabulary

- **Target variable:** Value to predict (e.g., iris species).
- **Features / Explanatory variables:** Columns used to predict target (e.g., petal/sepal length & width).
- **Example / Observation:** Single row in dataset.
- **Label:** Value of target variable for one example (e.g., “versicolor”).

Module 2: Data Retrieval & Cleaning

2.1 Retrieving Data

Key points

- Common data sources:
  - SQL databases: MySQL, PostgreSQL, SQLite
  - NoSQL databases: MongoDB, Redis, Cassandra
  - APIs: REST, JSON-based services
  - Cloud data sources: AWS, GCP, Azure
- Common formats:
  - CSV or TSV: tabular data with commas or tabs
  - JSON: key–value format, similar to a Python dict

Overview table

Type	Format	Python library	Typical use case
CSV	Comma separated tabular data	pandas → `pd.read` _csv()	General tabular datasets
JSON	Key–value objects and arrays	pandas → `pd.read` _json()	APIs and NoSQL exports
SQL	Relational tables queried with SQL	sqlite3 , pandas → `pd.read` _sql()	MySQL, PostgreSQL, SQLite
NoSQL	Document or key–value stores	pymongo (MongoDB), drivers per DB	MongoDB, Redis, Cassandra

2.2 Data Cleaning

Key points

- Garbage in = Garbage out. Dirty data leads to unreliable models.
- Main issues:
  - Missing values
  - Duplicate data
  - Outliers
  - Unnecessary or irrelevant data

2.2.1 Missing values

Options

- Remove: drop rows with NaN, but you lose data
- Impute: replace with mean or median, adds uncertainty
- Mask: assign a category for missing values

Syntax

```
# Remove
df.dropna()

# Impute with mean
df.fillna(df.mean())

# Impute with median
df.fillna(df.median())

# Mask as a category
df.fillna("Missing")
```

2.2.2 Duplicate data

Key points

- Duplicates bias metrics and can leak across train and test splits.
- Causes include rerun pipelines and multiple manual entries.
- Solutions: drop all duplicates or enforce uniqueness by specific column sets.

Syntax

```
# Drop all duplicate rows
df.drop_duplicates()

# Enforce uniqueness by multiple columns
df.drop_duplicates(subset=["FirstName", "LastName"])

# Enforce uniqueness by a single column
df.drop_duplicates(subset=["brand"])
```

2.2.3 Filtering and dropping

Key points

- Remove irrelevant data and keep only what’s needed.
- Filter by name, regex, or substring.

Syntax

```
# Select columns by explicit names
df.filter(items=['one', 'three'])

# Select columns ending with 'e'
df.filter(regex='e$', axis=1)

# Select rows containing substring 'bbi' in the index
df.filter(like='bbi', axis=0)

# Drop columns B and C
df.drop(['B', 'C'], axis=1)
```

2.2.4 Outliers

Key points

- Outliers are far from most observations and strongly affect many models.
- Detection methods: plots, IQR, Z-score, residuals.
- Handling: remove, impute, transform, or use robust models.

Syntax

```
# IQR method
Q1 = df['col'].quantile(0.25)
Q3 = df['col'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['col'] < (Q1 - 1.5 * IQR)) | (df['col'] > (Q3 + 1.5 * IQR))]
```

```
# Z-score method
import numpy as np
```

```
z_scores = np.abs((df['col'] - df['col'].mean()) / df['col'].std())
outliers = df[z_scores > 3]
```

Module 3: EDA and Feature Engineering — Notes with Code Examples

3.1 What is EDA?

- Exploratory Data Analysis summarizes the main characteristics of a dataset using descriptive statistics and visualizations.
- Goal: understand structure, quality, and patterns before modeling.

```
import pandas as pd

# Example: load a CSV (replace with your dataset)
# csv_url = "https://.../your_data.csv"
# df =
```

3.2 Why is EDA useful?

- Get an initial feel for the data
- Detect cleaning and preprocessing needs
- Spot errors, missing values, inconsistencies
- Reveal patterns, trends, and relationships

```
# Structural overview
print(
```

3.3 EDA techniques

- Descriptive stats: mean, median, min, max, variance, correlations

```
# Average Age by Sex (use columns that exist in your dataset)
print(df.groupby("Sex")["Age"].mean())
```

```
# Correlation matrix (numeric only)
print(df.corr(numeric_only=True))
```

- Visualization: histograms, scatter plots, box plots, pair plots

```
import matplotlib.pyplot as plt
import seaborn as sns

# Histogram of Age
_ = df["Age"].hist(bins=30)
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.title("Distribution of Age")
```

3.4 Sampling from DataFrames

- Reasons: speed up experimentation, balance rare outcomes, quick tests on subsets

```
# Random sample of 100 rows
sample_df = df.sample(100, random_state=42)
print(sample_df.head())
```

3.5 Feature Scaling

Why scaling?

- Different features can have very different ranges. For example, Age ∈ [0, 100] vs Income ∈ [0, 100000].
- Distance- and gradient-based algorithms such as KNN, SVM, Logistic Regression, and Neural Networks are scale-sensitive. Unscaled large-range features can dominate the optimization.
- Scaling puts features on comparable ranges to stabilize training and improve model performance.

Scaling methods

1) Standard scaling (Z-score normalization)

- Formula: $ z = frac{x - mu}{sigma} $

- Effect: centers features to mean 0 and standard deviation 1
- Notes: often a good default for many linear models and neural networks

```
from sklearn.preprocessing import StandardScaler
X_std = StandardScaler().fit_transform(X)
```

2) Min–Max scaling

- Formula: $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$ √x Phương trình mới
- Effect: maps features to [0, 1]
- Notes: useful when models expect bounded inputs; sensitive to outliers

```
from sklearn.preprocessing import MinMaxScaler
X_mm = MinMaxScaler().fit_transform(X)
```

3) Robust scaling

- Formula (per feature): $x' = \frac{x - \mathrm{median}(x)}{\mathrm{IQR}(x)}$, where $\mathrm{IQR} = Q_{75} - Q_{25}$
- Effect: centers by the median and scales by IQR
- Notes: less sensitive to outliers than Standard or Min–Max

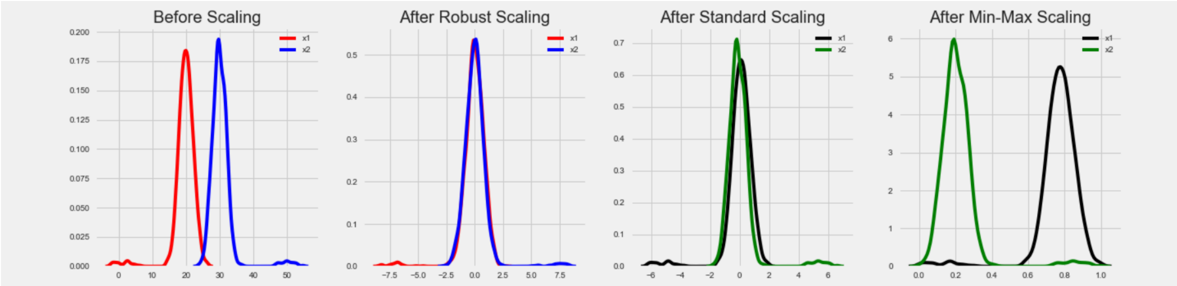
```
from sklearn.preprocessing import RobustScaler
X_robust = RobustScaler().fit_transform(X)
```

When to use which?

- If features have outliers: prefer RobustScaler
- If a bounded [0, 1] range helps or a model expects it: MinMaxScaler
- Otherwise, as a default starting point: StandardScaler

Practical tips

- Fit scalers on the training set only, then transform validation and test sets with the fitted scaler to avoid leakage
- Persist the fitted scaler for production to apply the exact same transformation

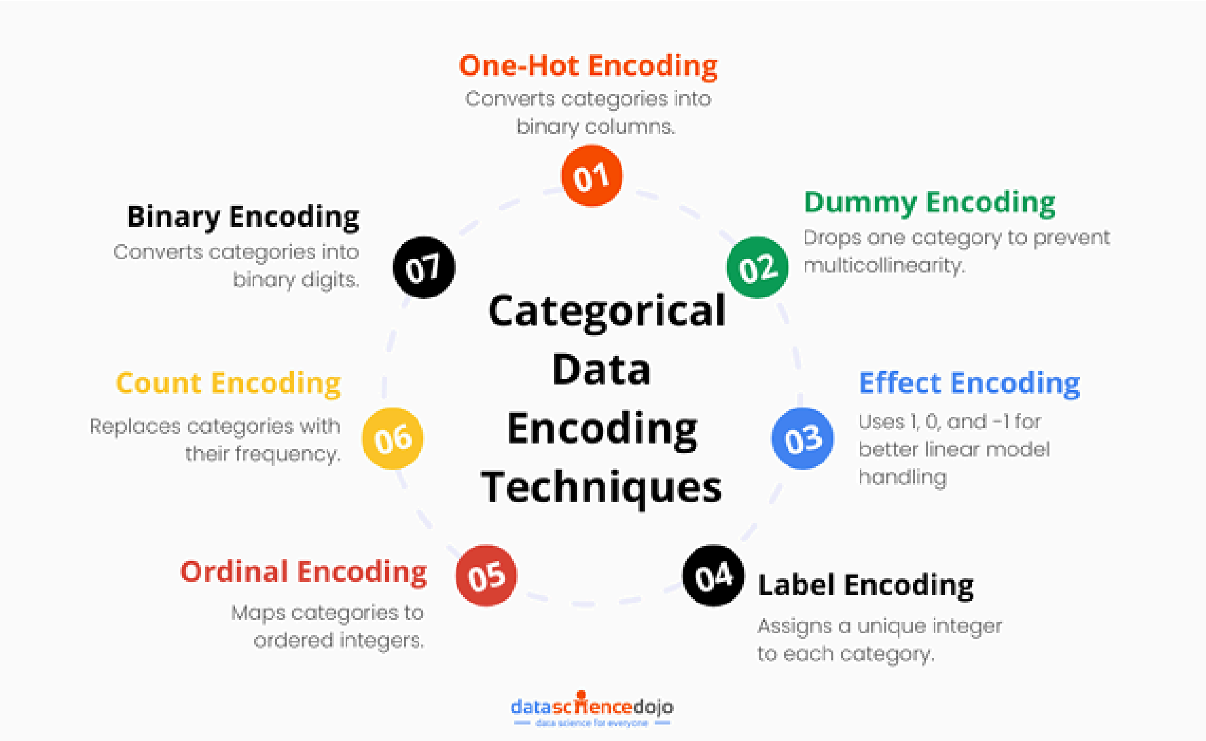


3.6 Encoding Categorical Variables

- Goal: transform categorical variables into numeric representations that models can learn from.

3.6.1 Common methods

- Binary encoding: two states → 0/1. Useful for flags and booleans.
- One-hot encoding: each category becomes a 0/1 column. Avoids imposing fake order.
- Dummy encoding: like one-hot but drop one reference column to avoid multicollinearity with an intercept.
- Ordinal encoding: map a true order Low < Medium < High → 1/2/3.
- Label encoding: integers 0..K-1 for categories. Prefer for tree-based models or truly ordered categories.



3.6.2 When to use which

- No natural order and using linear, KNN, SVM, or neural nets → One-hot or Dummy.
- Natural order present → Ordinal.
- High-cardinality features → consider Target encoding, Hashing encoding, or grouping rare levels.
- Tree models (GBDT/Random Forest) can work with Label/Ordinal, but beware introducing artificial order.

3.6.3 Code with pandas

```
import pandas as pd

# One-hot (keep all columns)
df_oh = pd.get_dummies(df, columns=["category"], dtype=int)

# Dummy (drop reference column to avoid multicollinearity)
df_dummy = pd.get_dummies(df, columns=["category"], drop_first=True, dt
ype=int)
```

3.6.4 Code with scikit-learn

```
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder

# One-hot encoder (robust to unseen categories at inference)
ohe = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
X_oh =
```

3.6.5 Practical tips

- Build a pipeline: fit encoders on the training set, then transform validation and test to avoid leakage.
- Persist fitted encoders with the model for production.
- Combine rare categories into "Other" to reduce one-hot width and stabilize estimates.
- Guard against unseen categories at inference: use handle_unknown="ignore" for OneHotEncoder or map to "Other" before encoding.
- Purpose: correct skewness, reduce outlier impact, linearize relationships, and help optimization converge
- Log transformation
    - Use for right-skewed variables such as income; use log1p when zeros are present

```
import numpy as np
import matplotlib.pyplot as plt

x = np.random.exponential(scale=2, size=1000)
x_log = np.log1p(x)

plt.figure(figsize=(12,4))
plt.subplot(1,2,1); plt.hist(x, bins=30, color="skyblue"); plt.title("Original (Ri
ght-Skewed)")
plt.subplot(1,2,2); plt.hist(x_log, bins=30, color="orange"); plt.title("Log-Tra
nsformed")
```

- Polynomial features
    - Use when relationships are non-linear; beware of overfitting at high degrees

```
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
import pandas as pd

X = np.arange(6).reshape(-1, 1)
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly =
```

- Feature engineering improves model accuracy by creating and transforming variables
- Common: logs and polynomials for skewness and nonlinearity. Encode categoricals. Normalize for comparability
- Principles: understand variable types and model assumptions. Apply consistently to train and test

---

Module 4: Estimation & Inference + Hypothesis Testing — Summary Notes

4.1 Estimation

- **Sample mean:** $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$
- **Sample variance:** $S^2 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n-1}$
- **Sample standard deviation:** $S = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{n-1}}$

4.2 Inference

- **Standard Error (SE):** $\mathrm{SE} = \frac{S}{\sqrt{n}}$
- **t-CI for mean:** $\bar{X} \pm t_{\alpha/2,\,n-1} \cdot \mathrm{SE}$

4.3 Parametric vs Non-parametric

- **Parametric:** assumes a distribution with a finite number of parameters. Example: Normal (μ, σ^2)
- **Non-parametric:** fewer assumptions, not distribution-based. Examples: histogram, empirical CDF, KDE

4.4 Maximum Likelihood Estimation (MLE)

- **Likelihood:** $L(\theta \mid x) = \prod_{i=1}^{n} f(x_i \mid \theta)$
- **Log-likelihood:** $\ell(\theta) = \sum_{i=1}^{n} \ln f(x_i \mid \theta)$
- **MLE estimator:** $\hat{\theta}_{\mathrm{MLE}} = \arg\max_{\theta} \ \ell(\theta)$

4.5 Common Distributions

- Uniform on $[a, b]$ with $a < b$ and density:

$$f(x) = \frac{1}{b-a}, \quad a \le x \le b$$

- Normal with mean μ và variance $ `\sigma^2` $:

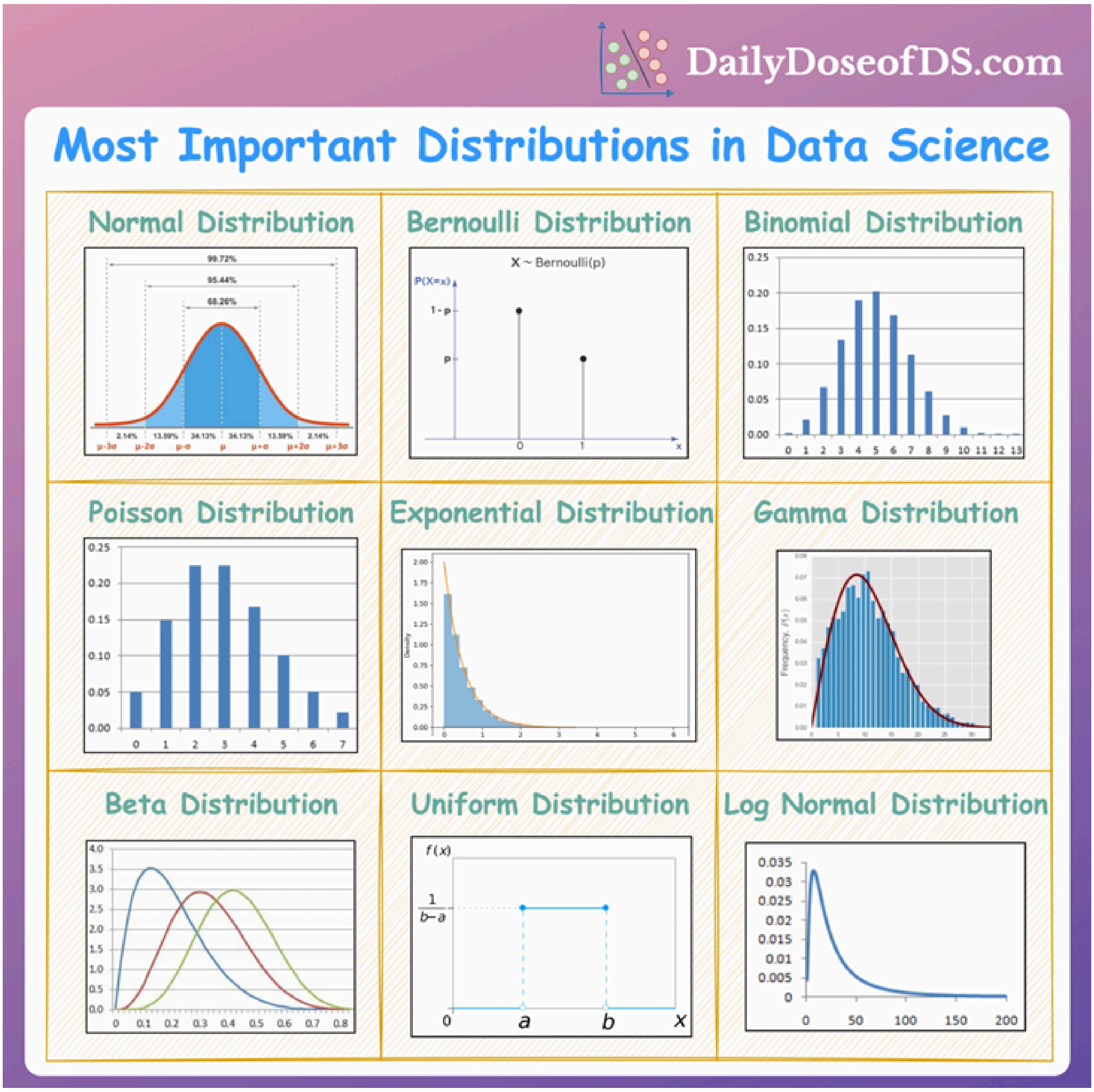
$$f(x) = \frac{1}{\sqrt{2\pi\,\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

- Exponential với tốc độ $lambda>0$:

$$f(x) = \lambda\, e^{-\lambda x}, \quad x \ge 0$$

- Poisson với tốc độ $lambda>0$:

$$\mathbb{P}(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad k = 0, 1, 2, \ldots$$



4.6 Frequentist vs Bayesian

- **Frequentist:** Probability = long-run frequency under repeated experiments
- **Bayesian:** Parameters are random variables. Bayes: $P(\theta \mid D) = \dfrac{P(D \mid \theta)\, P(\theta)}{P(D)}$

4.7 Hypothesis Testing

4.7.1 Concepts

- H_0 (null): baseline hypothesis, no effect
- H_1 (alternative): opposing hypothesis, effect exists

4.7.2 Decision rule

- From a test statistic define the **rejection region** for H_0 and the **acceptance region**

4.7.3 Bayesian view

- $P(H_i \mid D) = \dfrac{P(D \mid H_i)\, P(H_i)}{P(D)}$

4.7.4 Likelihood ratio

- $\Lambda(x) = \dfrac{P(\text{data} \mid H_1)}{P(\text{data} \mid H_0)}$

4.7.5 Type I and II errors

- **Type I (α):** reject H_0 when H_0 is true
- **Type II (β):** fail to reject H_0 when H_0 is false
- **Power:** $1 - \beta$

4.7.6 Significance level and p-value

- Choose α commonly 0.05 or 0.01
- Rule: reject H_0 if $p < \alpha$

4.7.7 Coin toss example

- H_0: fair coin $\Rightarrow X \sim \text{Binomial}(n = 10,\, p = 0.5)$
- Observe 3 heads. $p\text{-value} = P(X \le 3) \approx 0.17 > 0.05 \Rightarrow$ **do not reject** H_0

4.7.8 F-statistic (linear regression)

- H_0: all coefficients $\beta = 0$ (cụ thể: $beta_1=beta_2=cdots=beta_p=0$). Reject if F-test p-value is small enough

4.7.9 Multiple testing — Bonferroni

- Adjust significance level: $\alpha' = \dfrac{\alpha}{m}$

4.7.10 Correlation vs Causation

- Correlation $\neq$ causation

4.7.11 Code examples (Python)

- CI for the mean (use t-distribution when σ is unknown):

```
import numpy as np
from scipy import stats

# Example sample
x = np.array([12.1, 11.7, 12.4, 11.9, 12.3, 12.0, 11.8, 12.2])

n = len(x)
mean = x.mean()
s = x.std(ddof=1)
alpha = 0.05
se = s / np.sqrt(n)

# t critical with df = n-1
t_crit = stats.t.ppf(1 - alpha/2, df=n-1)
ci = (mean - t_crit*se, mean + t_crit*se)
mean, s, ci
```

- One-sample test (H0: μ = μ0) using t-test:

```
mu0 = 12.0
# two-sided t-test
res = stats.ttest_1samp(x, popmean=mu0)
res.statistic, res.pvalue
```

- Coin p-value example (n=10, k=3, H0: p=0.5, two-sided or one-sided):

```
from scipy.stats import binom
n, k, p0 = 10, 3, 0.5
# one-sided p-value: P(X <= k)
p_left = binom.cdf(k, n, p0)
# one-sided p-value: P(X >= k)
p_right = 1 - binom.cdf(k-1, n, p0)
# two-sided p-value (simple approach: double the more "extreme" side nea
```

```
r the mean)
p_two_sided = 2 * min(p_left, p_right)
p_left, p_right, p_two_sided
```

4.7.12 Frequentist vs Bayesian — Quick summary

Aspect	Frequentist	Bayesian
Meaning of probability	Long-run frequency of events	Degree of belief, prior-informed
Parameters	Fixed but unknown	Random variables with a prior distribution
Inference	Based on sampling distribution of statistics	Prior + likelihood → posterior
Intervals	CI: procedure yields 1−α coverage in repeated samples	Credible interval: probability the parameter lies in the interval
Decision	p-values, significance level α, hypothesis tests	Maximize or integrate over posterior, Bayes factor

4.7.13 Formulas

$$\mathrm{SE} = \frac{S}{\sqrt{n}} \qquad \mathrm{CI}_\mu : \\ \bar{X} \pm t_{\alpha/2,\, n-1} \cdot \frac{S}{\sqrt{n}}$$

$$\Lambda(x) = \frac{\mathcal{L}(H_1)}{\mathcal{L}(H_0)} \qquad \text{Bonferroni: } \alpha' = \frac{\alpha}{m}$$

- $X \to Y$
- $Y \to X$
- Confounders affect both
- Spurious correlation