UNIT 5_II

GitHub

PREPROCESSING | DATA CLEANING → DATA INCONSISTENCIES/ ANOMALIES

1. Data Inconsistency

Data inconsistencies are errors or contradictions in a dataset, where the same data is represented in different, conflicting, mismatched, or illogical ways across a dataset. In machine learning, inconsistent data can **mislead the model**, reduce accuracy, and increase preprocessing complexity.

In **data cleaning**, handling data inconsistencies means identifying and correcting these mismatches so the data becomes **accurate, uniform, and reliable** for training machine learning models.

1.1. Data Inconsistency by Data Type / Format

1.1.1. Numerical Data

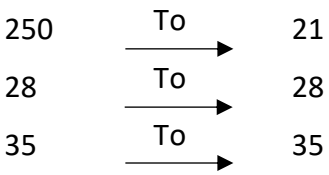
- **Example:** Age recorded as 25 (year) in one row and 250 (months) in another for the same person. If Age is greater than 100 consider as months.
- **Issues:** Outliers, wrong units, conflicting measurements.
- **Cleaning Approach:** Apply validation rules, detect outliers, cross-check with reference ranges.

```
# Convert unrealistic age values (>100) from months to years
# Values greater than 100 are assumed to be recorded in months
df['Age'] = df['Age'].apply(
    lambda x: round(x / 12) if x > 100 else x
)

# Preview corrected Age values
df['Age'].head()
```

Output:

Age		Age
25	To →	25
30	To →	30



1.1.2. Categorical / Text Data

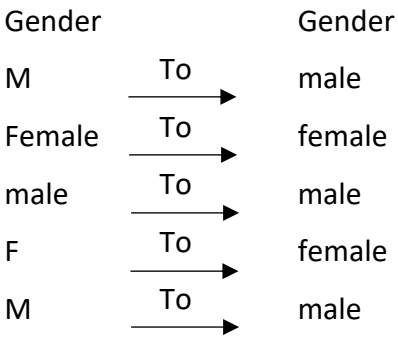
- **Example:** Gender recorded as M, Male, male, or m in different rows.
- **Issues:** Different spellings, synonyms, inconsistent abbreviations.
- **Cleaning Approach:** Standardization, mapping to a canonical form, case normalization.

```
# Convert Gender values to lowercase for consistency
df['Gender'] = df['Gender'].str.lower()

# Replace abbreviated gender labels with full descriptive words
df['Gender'] = df['Gender'].replace({
    'm': 'male',
    'f': 'female'
})

# Preview standardized Gender values
df['Gender'].head()
```

Output:



1.1.3. Date / Time Data

- **Example:** 01/02/2025 in one row, 2025-02-01 in another (DD/MM/YYYY vs YYYY-MM-DD).
- **Issues:** Different formats, wrong time zones, impossible dates (like 30 Feb).
- **Cleaning Approach:** Convert to a unified format (ISO standard), detect invalid dates.

```
# Function to parse dates in multiple formats and standardize to
dd/mm/yyyy
def parse_date_flexible(date_str):
    # List of possible input date formats
    formats = [
        '%Y-%m-%d', '%d/%m/%Y', '%m/%d/%Y', '%d-%m-%Y',
   '%Y/%m/%d'
    ]

    # Try each format until one succeeds
    for fmt in formats:
        try:
            # Parse string into datetime object
            date_obj = datetime.strptime(str(date_str), fmt)
            # Return standardized date format
            return date_obj.strftime('%d/%m/%Y')
        except:
            pass  # continue trying other formats

    return None  # return None if no format matches

# Apply flexible date parsing to DateOfBirth column
df['DateOfBirth'] = df['DateOfBirth'].apply(parse_date_flexible)

# Preview cleaned dates
df['DateOfBirth'].head()
```

Output:

DateOfBirth		DateOfBirth
1/15/1998	to →	15/1/1998
5/20/1993	to →	20/5/1993
3/10/1995	to →	3/10/1995
7/25/1995	to →	25/7/1995
11/30/1988	to →	30/11/1998

1.1.4. Boolean / Binary Data

- **Example:** True/False vs 1/0 vs Yes/No in the same column.
- **Issues:** Mixed representations can confuse algorithms.
- **Cleaning Approach:** Map all values to a single consistent representation.

```
# Define accepted truthy and falsy string values
TRUE_WORDS = {'true', 'yes', 'y', '1', 'on'}
FALSE_WORDS = {'false', 'no', 'n', '0', 'off'}

# Normalize boolean-like values to binary (1/0); invalid values
become NA
def normalize_boolean(s):
    s = s.astype(str).str.strip().str.lower()
     # standardize text format
    return s.map(
        lambda x: 1 if x in TRUE_WORDS else
                0 if x in FALSE_WORDS else pd.NA
    )

# Apply normalization to the IsActive column
normalize_boolean(df['IsActive']).head()
```

Output:

IsActive		IsActive
TRUE	To →	1
1	To →	1
Yes	To →	1
TURE	To →	1
FALSE	To →	0

1.1.5. Multi-format / Mixed-type Columns

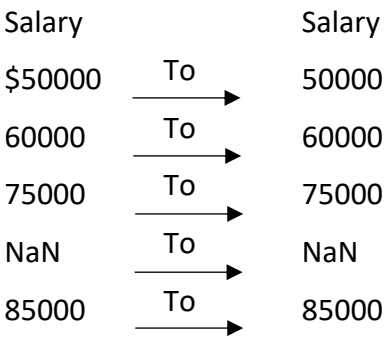
- **Example:** Column Price having values "$100", 100, USD 100.
- **Issues:** Numeric algorithms can't directly process strings.
- **Cleaning Approach:** Strip non-numeric symbols, convert to float or integer.

```
# Clean Salary column: remove currency symbols and standardize
format
df['Salary'] = (
    df['Salary']
    .astype(str)                # ensure string type
    .str.replace(r'[$,]|USD', '', regex=True)
# remove $, commas, USD
    .str.strip()         # remove extra spaces
)

# Convert cleaned values to numeric; invalid entries become NaN
df['Salary'] = pd.to_numeric(df['Salary'], errors='coerce')

df['Salary'].head()
```

Output:



1.1.6. Missing / Null Values

- **Example:** One system records missing salary as NaN, another as 0, and another as Unknown.
- **Issues:** Can create misinterpretation during analysis.
- **Cleaning Approach:** Replace with a consistent placeholder, impute, or remove rows.

```
from sklearn.impute import SimpleImputer

# Standardize missing values
df['Salary'] = (
    df['Salary']
    .astype(str)
    .str.strip()
```

```
    .replace(['Unknown', 'unknown', '', 'None', 'null', 'nan',
'NaN'], np.nan)
)

# Convert to numeric
df['Salary'] = pd.to_numeric(df['Salary'], errors='coerce')

# Treat 0 as missing
df.loc[df['Salary'] == 0, 'Salary'] = np.nan

# Median Imputation
imputer = SimpleImputer(strategy='median')
df['Salary'] = imputer.fit_transform(df[['Salary']]).round(0)

df['Salary'].head()
```

Output:

Salary		Salary
$50000	To →	50000.0
60000	To →	60000.0
75000	To →	75000.0
NaN	To →	76500.0
85000	To →	85000.0

1.1.7. Duplicated / Redundant Records

- **Example:** Two entries for the same customer with slight differences in spelling or address.
- **Issues:** Biases statistical distributions, misleads ML models.
- **Cleaning Approach:** Deduplication using fuzzy matching, key-based merging.

```
# Check whether duplicate rows exist in the dataset (True meaning
duplicates exist)
print(df.duplicated().any())

# If duplicates are found, remove them
if df.duplicated().any():
```

```
    # Assign back to apply the change
    df = df.drop_duplicates()

# Verify that duplicates have been removed (False meaning no duplicates)
df.duplicated().any()
```

Output:

```
True
False
```

1.1.8. Logical Conflicts

- **Example:** A person's Date of Birth is 2010-05-10 but Age column shows 30.
- **Issues:** Internal inconsistencies can break model assumptions.
- **Cleaning Approach:** Recalculate dependent fields or flag errors for manual review.

```
from datetime import datetime
import pandas as pd
import numpy as np

current_year = datetime.now().year

def clean_dob(dob):
    try:
        dob_str = str(dob).strip()
        # Convert numeric ages to DOB
        if dob_str.replace('.', '').isdigit():
            return f"01/01/{current_year - int(float(dob_str))}"
    except:
        pass
    # Parse known date formats
    dt = pd.to_datetime(dob, errors='coerce', dayfirst=True)
    return dt.strftime('%d/%m/%Y') if pd.notna(dt) else None

# Apply
df['RegistrationDate'] = df['RegistrationDate'].apply(clean_dob)
```

```
df['RegistrationDate'].head()
```

Output:

DateOfBirth		DateOfBirth
2/20/2023	to →	2/20/2023
26	to →	01/01/2000
4/10/2023	to →	04/10/2023
5/15/2023	to →	15/05/2023
25	to →	01/01/2001

1.1.9. Naming Conversion

- U.S.A, USA, United State, United State of America

```
import country_converter as coco
import warnings

cc = coco.CountryConverter()

# Pre-clean the country column
df['Country_cleaned'] = (
    df['Country']
    .fillna('')  # Handle NaN values
    .astype(str)
    .str.strip()
    .str.replace('U.S.A', 'USA', regex=False)
    .str.replace('United State', 'United States', regex=False)
    .str.replace('U.K', 'UK', regex=False)
)
# Convert with suppressed warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

    # Convert country names
    converted = cc.convert(
        names=df['Country_cleaned'].tolist(), #Convert to list
        to='name_short',
```

```
        not_found='UNKNOWN'
    )

# Create new column from converted results
df['Country'] = converted

# Replace failed conversions with NA
df.loc[df['Country'] == 'UNKNOWN', 'Country'] = pd.NA
df.loc[df['Country'] == 'not found', 'Country'] = pd.NA

# Convert to lowercase (only for non-NA values)
df['Country'] = df['Country'].str.lower()

# Drop temporary column
df.drop('Country_cleaned', axis=1, inplace=True)

df['Country'].head(20)
```

Output:

Country		Country
U.S.A	to →	united states
United States	to →	united states
Use	to →	united states
Germany	to →	germany
United Kingdom	to →	united kingdom

1.1.10. Typographical Mistake

- James Taylor, james taylor

```
# Standardize Category column to lowercase
df['Category'] = df['Category'].str.strip().str.lower()

df['Category'].head()
```

Output:

Category		Category
Premium	to	Premium
PREMIUM	to	Premium
Gold	to	Gold
Gold	to	Gold
Silver	to	Silver

1.1.11. Contradictory Data

- Son age < father age (truth)
- Son age > father age (not truth remove this contradictory inconsistency)

```
from datetime import datetime

# Parse dates
df['temp_second'] = pd.to_datetime(df['SecondLastPerchase'],
format='mixed', dayfirst=True, errors='coerce')
df['temp_last'] = pd.to_datetime(df['LastPerchase'],
format='mixed', dayfirst=True, errors='coerce')

# Find rows to swap
needs_swap = df['temp_last'] < df['temp_second']

# Swap the original string columns
df.loc[needs_swap, ['SecondLastPerchase', 'LastPerchase']] = \
    df.loc[needs_swap, ['LastPerchase',
'SecondLastPerchase']].values

# Clean up
df.drop(['temp_second', 'temp_last'], axis=1, inplace=True)

df[['SecondLastPerchase', 'LastPerchase']].head()
```

Output:

SecondLast Perchase	LastPerchase		SecondLastPerchase	LastPerchase
15/02/2024	15/02/2023	to	15/02/2023	15/02/2024

SecondLast Perchase	LastPerchase		SecondLastPerchase	LastPerchase
3/1/2024	3/1/2026	to	3/1/2024	3/1/2026
30/04/2024	4/10/2025	to	30/04/2024	4/10/2025
5/20/24	5/15/2023	to	5/15/2023	5/20/202024
07/10/2024	11/4/2023	to	11/4/2023	7/10/2024