UNIT 5_I

GitHub

PREPROCESSING | DATA CLEANING → MISSING VALUES

A. Data Cleaning

Data cleaning is the process of identifying and correcting errors, inconsistencies and removing any missing, duplicate or irrelevant data in the data to ensure it is accurate and complete. The objective is to address issues that can distort analysis or model performance.

- Raw data (log file, transactions, audio /video recordings, etc.) is often noisy, incomplete and inconsistent which can negatively impact the accuracy of model.
- The goal of data cleaning is to ensure that the data is accurate, consistent and free of errors.
- Clean datasets also important in EDA (Exploratory Data Analysis) which enhances the interpretability of data so that the right actions can be taken based on insights.

For example:

- **Handling missing values:** Using strategies like mean/mode imputation, deletion, or predictive models to fill in or remove missing data.
- **Removing duplicates:** Eliminating duplicate records to ensure each entry is unique and relevant.
- **Correcting inconsistent formats:** Standardizing formats (e.g., date formats, string cases) to maintain consistency.

a. Install Libraries

```
# After changing kernel, install directly
!conda install numpy pandas matplotlib seaborn scikit-learn tensorflow pytorch scikit-learn jupyter -y
```

b. Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

c. Load Dataset (Seaborn Library or Local Directory

```
df = sns.load_dataset('titanic')
```

```
# Alternative if loading from CSV
# df = pd.load_dataset('titanic.csv')

# Alternative if loading from JSON
# df = pd.json('titanic.json')

# Alternative if loading from Excel
# df = pd.read_excel('titanic.xlsx')

# Alternative if loading from SQL
# df = pd.read_sql('SELECT * FROM titanic_table', connection)
```

d. Display first 5 rows of the dataset

```
df.head()
```

Output:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

1.1. Exploring/Identifying Missing Values in the Dataset

a. Method 1: Missing Values Analysis

```
# df.isnull().sum()
df.isnull().sum().sort_values(ascending=False)
```

Output:

```
deck           688
age            177
embarked         2
embark_town      2
sex              0
pclass           0
survived         0
```

```
fare            0
parch           0
sibsp           0
class           0
adult_male      0
who             0
alive           0
alone           0
dtype: int64
```

b. Method 2: Percentage of missing values

```
# df.isnull().mean() * 100

# df.isnull().sum() / len(df) * 100

# (df.isnull().sum() / len(df) * 100).sort_values(ascending=False)

(df.isnull().sum() / len(df) *
100).round(2).sort_values(ascending=False)
```

Output:

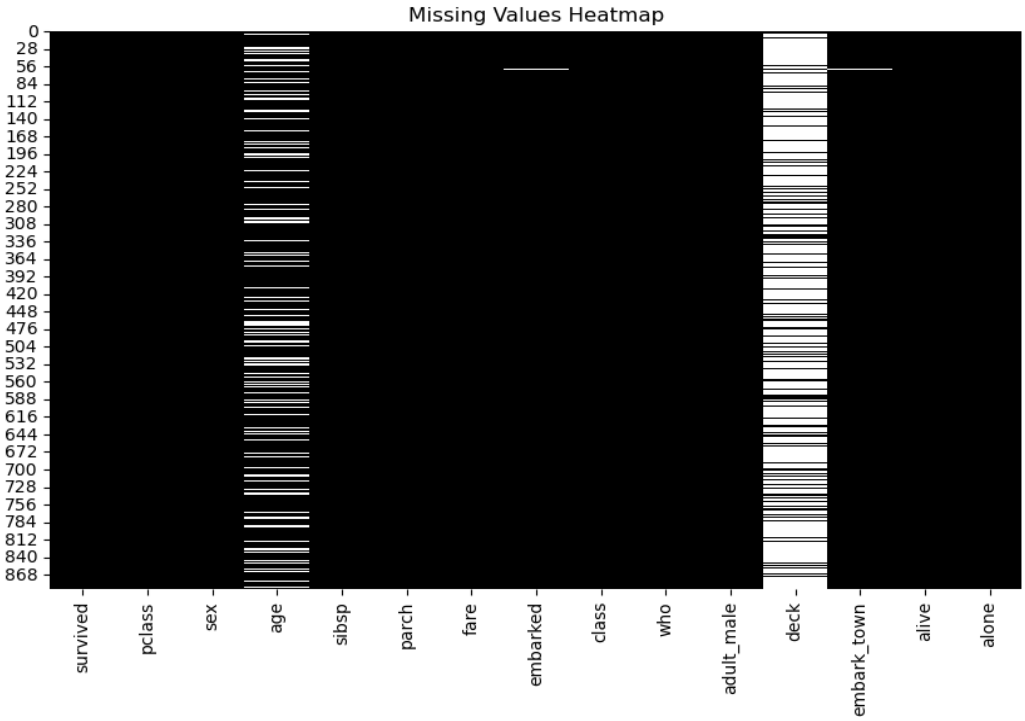
```
deck           77.22
age            19.87
embarked        0.22
embark_town     0.22
sex             0.00
pclass          0.00
survived        0.00
fare            0.00
parch           0.00
sibsp           0.00
class           0.00
adult_male      0.00
who             0.00
alive           0.00
```

```
alone           0.00
dtype: float64
```

c. Method 3: Visualizing Missing Values

```
from matplotlib.colors import ListedColormap
plt.figure(figsize=(10,6))
sns.heatmap(df.isnull(), cbar=False, cmap=ListedColormap(['black',
'white']))
plt.title('Missing Values Heatmap')
plt.show()
```

Output:



d. Method 4: DataFrame Info Missing Values Summary

```
df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
```

```
 1   pclass        891 non-null    int64
 2   sex           891 non-null    object
 3   age           714 non-null    float64
 4   sibsp         891 non-null    int64
 5   parch         891 non-null    int64
 6   fare          891 non-null    float64
 7   embarked      889 non-null    object
 8   class         891 non-null    category
 9   who           891 non-null    object
 10  adult_male    891 non-null    bool
 11  deck          203 non-null    category
 12  embark_town   889 non-null    object
 13  alive         891 non-null    object
 14  alone         891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

1.2. Other Names of Missing Values

- **NaN (Not a Number):** Used in programming libraries like Python (pandas) to represent missing or invalid numeric data.
- **Null:** Commonly used in database systems such as SQL to indicate no value.
- **Undefined:** A value that has not been assigned or defined.
- **Blank / Empty:** A field with no entered value.
- **Placeholder Values:** Default values (e.g., −1, 0, 999) used to represent missing data.
- **Sentinel Values:** Special placeholder values indicating specific conditions, often used for missing data.
- **Dummy Data:** Artificial or test data that does not represent real observations.
- **Missing Data:** A general term used in statistics and research for unavailable values.

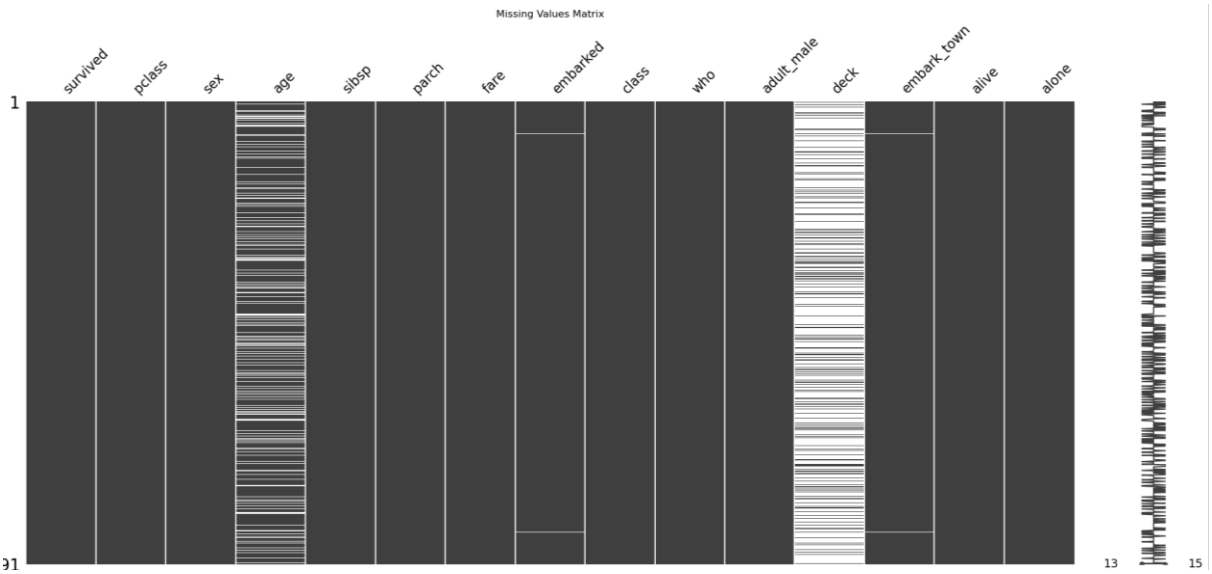
1.3. How to Identify Missing Values?

- To identify missing values, we use some techniques. These techniques are called **Missing Value Detection Techniques**. Some of these techniques are given below:
- **Visual Inspection:** Missing values are identified by visualizing the data.
- **Descriptive Statistics:** Missing values are identified by calculating the descriptive statistics of the data.

- **Missingno Library:** Missing values are identified by using the Missingno library.

```
import missingno as msno
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))  # Set figure size
msno.matrix(df)
plt.title('Missing Values Matrix')
plt.show()
```

Output:



1.4. Why Is Handling Missing Values So Important?

- **Deep Impact on Model Accuracy:** The presence of missing values reduces the accuracy of machine learning models and negatively affects their overall performance.
- **Questions on Data Quality:** Missing values weaken the quality of data, which can lead to misunderstandings and incorrect conclusions in our analysis and decision-making.
- **Model Training Time Increases:** Sometimes, due to missing values, the model training time increases, resulting in a waste of both time and computational resources.

1.5. Deciding Whether to Remove a Column with Missing Values

Having missing values is common in any dataset, but when deciding whether a column should be removed or not, this decision depends on several important factors:

- **Data Quantity:**

  If you have a very large amount of data and a specific column contains a very high percentage of missing values (for example, 70% or 80%), then removing that column may be a better option, because it becomes difficult to extract meaningful value from it.

- **Column Importance:**

If the column with missing values is very important for your analysis or model, then removing it would not be a good decision. In such cases, you can use different techniques to impute the missing values.

- **Nature of the Data:**

Sometimes, the presence of missing values itself conveys important information. For example, in a survey, if a question is left unanswered, it may indicate that the participant was uncomfortable answering it. In such cases, removing or blindly replacing the missing value may not be appropriate.

- **Model Sensitivity:**

Some machine learning models can handle missing values, while others are sensitive to them. If the chosen model is sensitive to missing values, then you must handle those missing values properly before training.

- **Type of Data:**

For numerical data, missing values can be replaced using the mean, median, or mode. For categorical data, missing values can be replaced using the mode or by assigning a specific category.

- **General Rule (Not Absolute):**

Generally, if more than 50% of the data in a column is missing, you should consider whether removing that column would be a better option. However, this is not a hard-and-fast rule. Every dataset is unique and has different requirements. Therefore, the decision on how to handle missing values should always be made based on the context of the dataset.

1.6. Detailed Methods for Handling Missing Values

- **Re-Collecting Data from the Original Data Source:**

If you have access to the original resource from where the data was collected, you can retrieve the missing values again from that source.

- **Imputing Data Using Mean, Median, or Mode:**

If the data is numerical, missing values are commonly replaced using the mean or median.
For categorical data, the mode is typically used to fill in missing values.

- **Using Forward or Backward Fill:**

In some datasets, there is a sequence of time or dates. In such datasets, a missing value in one row is replaced with the value from the previous or the next row.

- **Using KNN Imputation:**

This is an advanced technique in which a missing value is replaced with the average value of its neighboring data points. This method is available in libraries such as scikit-learn.

- **Using Deep Learning Techniques:**

Deep learning techniques, such as autoencoders, can also be helpful in handling missing values.

- **Simply Deleting the Data:**

If the number of missing values in your dataset is very small, you can delete the specific row or column containing those missing values.

- **What If I Don't Handle Missing Values?**

Then the model will definitely not work well—and that's not all, listen further!

If we ignore missing values, we may face many problems. Some of the issues that can arise are listed below:

- **Decrease in Model Accuracy:**

  The accuracy of machine learning models can decrease because the model does not receive complete information.

- **Incorrect Analysis:**

  Wrong results may be produced during data analysis, which can negatively affect decision-making.

- **Model Confusion:**

  Some models cannot handle missing values, causing the model to fail during training or make incorrect predictions.

- **Bias in the Model:**

  Missing values increase the risk of introducing bias into the model.

- **Incorrect Interpretation of Data:**

  Due to missing values, the available information may be incomplete or incorrect, leading to wrong interpretation of the data.

- **Storage Issues:**

  If missing values are not replaced, storage-related issues may occur because some systems cannot properly store missing values.

- **Data Integration Problems:**

  When integrating data from different sources, missing values can create serious integration issues.

- **Incorrect Feature Selection:**

  In the presence of missing values, some important features that should influence the model may be ignored.

- **Incorrect Experimental Results:**

In scientific or research projects, missing values can lead to incorrect experimental outcomes.

- **Stress and Extra Work:**

Data scientists may have to do extra work during analysis because missing values need to be identified and handled.

1.7. How to Perform Data Cleaning

The process begins by identifying issues like missing values, duplicates and outliers. Performing data cleaning involves a systematic process to identify and remove errors in a dataset. The following steps are essential to perform data cleaning:

- **Remove Unwanted Observations:** Eliminate duplicates, irrelevant entries or redundant data that add noise.
- **Fix Structural Errors:** Standardize data formats and variable types for consistency.
- **Manage Outliers:** Detect and handle extreme values that can skew results, either by removal or transformation.
- **Handle Missing Data:** Address gaps using imputation, deletion or advanced techniques to maintain accuracy and integrity.

1.8. Dealing/Impute Missing Values Using Pandas Library

- **Calculate median for age column**

```
df['age'].median()
```

Output:

```
28.0
```

- **Calculate the mean for age column, median is less affected by outliers as compared to mean**

```
df['age'].mean().round(2)
```

Output:

```
np.float64(29.7)
```

- **Mean or Median Imputation to fill age column missing values using pandas**

```
# df['age'].fillna(df['age'].mean(), inplace=True)  # Mean

df['age'].fillna(df['age'].median(), inplace=True)  # Median

# Missing Values Analysis
df.isnull().sum().sort_values(ascending=False)
```

Output:

```
deck           688
embarked         2
embark_town      2
age              0
survived         0
pclass           0
sex              0
fare             0
parch            0
sibsp            0
class            0
adult_male       0
who              0
alive            0
alone            0
dtype: int64
```

- **Drop or remove the duck column with high missing values**

```
df.drop('deck', axis=1, inplace=True) # axis=0 for rows,
axis=1 for columns, inplace=True to modify the original
dataframe

# Missing Values Analysis
df.isnull().sum().sort_values(ascending=False)
```

Output:

```
embark_town    2
embarked       2
sex            0
age            0
survived       0
pclass         0
parch          0
sibsp          0
class          0
```

```
fare           0
who            0
adult_male     0
alive          0
alone          0
dtype: int64
```

- **Calculate mod for embarked column**

```
df['embarked'].mode()[0]
```

Output:

```
'S'
```

- **Calculate mod for embark_town column**

```
df['embark_town'].mode()[0]
```

Output:

```
'Southampton'
```

- **Value counts for embarked columns**

```
df['embarked'].value_counts()
```

Output:

```
embarked
S    644
C    168
Q     77
Name: count, dtype: int64
```

- **embarked and embark_town are categorical columns with few missing values, we can use Mode Imputation to fill missing values**

```
# Mode Imputation to fill embarked missing values using pandas
df['embarked'].fillna(df['embarked'].mode()[0],
inplace=True)  # Mode Imputation

# Mode Imputation to fill embark_town missing values using
pandas
df['embark_town'].fillna(df['embark_town'].mode()[0],
inplace=True)  # Mode Imputation
```

```
# Missing Values Analysis
df.isnull().sum().sort_values(ascending=False)
```

Output:

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
embark_town    0
alive          0
alone          0
dtype: int64
```

- **Using dropna method to remove rows with missing values in embarked column**

```
dna = sns.load_dataset('titanic')
dna.dropna(subset=['embarked'], inplace=True)

# Missing Values Analysis
dna.isnull().sum().sort_values(ascending=False)
```

Output:

```
deck           688
age            177
survived         0
pclass           0
sex              0
sibsp            0
parch            0
fare             0
embarked         0
```

```
class            0
who              0
adult_male       0
embark_town      0
alive            0
alone            0
dtype: int64
```

See more ways...

1.9. Imputer age column missing values using forward fill method using pandas

Forward fill is a data imputation technique used to handle missing values, by replacing a missing value with the **most recent non-missing value** in rows that appears **before it** in the dataset.

```
# Load dataset
ff = sns.load_dataset('titanic')

# Forward fill missing values in 'age' column (safe way)
ff['age'] = ff['age'].ffill()

# Missing values analysis
ff.isnull().sum().sort_values(ascending=False)
```

Output:

```
deck           688
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
age              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
```

```
alive            0
alone            0
dtype: int64
```

1.10. Imputer age column missing values using backward fill method using pandas

Backward fill is a data imputation technique used to handle missing values, by replacing a missing value with the **next available non-missing value** in rows that appears **after it** in the dataset.

```
# Load dataset
bf = sns.load_dataset('titanic')

# Backward fill missing values in 'age' column (safe way)
bf['age'] = ff['age'].bfill()

# Missing values analysis
bf.isnull().sum().sort_values(ascending=False)
```

Output:

```
deck           688
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
age              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

1.11. Imputation of Missing Values Using scikit-learn Library | Link

- **importing libraries to check their versions**

```
import numpy
import scipy
import sklearn

print(numpy.__version__)
print(scipy.__version__)
print(sklearn.__version__)
```

- **Import SimpleImputer from sklearn.impute**

```
from sklearn.impute import SimpleImputer
```

There are four types of imputation for handling missing values:

1.11.1. Univariate Feature Imputation (SimpleImputer):

The **SimpleImputer** class provides basic strategies for imputing missing values.

Univariate imputation is a missing-value handling approach in which missing values are filled **using a single column's own information**. As described, the SimpleImputer replaces missing values with a **constant value or a statistical measure (mean, median, or most frequent)** calculated **from the same column where the missing values occur**, without using information from other features.

The **SimpleImputer** class also supports categorical data represented as string values or pandas categoricals when using the 'most_frequent' or 'constant'

```
# Load Dataset
si = sns.load_dataset('titanic')

# Impute age column missing values using SimpleImputer with
Median strategy
imputer = SimpleImputer(strategy='median')
si['age'] = imputer.fit_transform(si[['age']])

# Missing Values Analysis
si.isnull().sum().sort_values(ascending=False)
```

Output:

```
deck           688
embarked         2
```

```
embark_town      2
survived         0
pclass           0
sex              0
age              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

1.11.2. Multivariate Feature Imputation (IterativeImputer):

Multivariate feature imputation is an advanced missing-value handling technique in which the missing values of a feature are estimated **using information from other related features in the dataset**. Instead of treating each column independently, this approach models each feature with missing values as a **function of the remaining features**.

Using the IterativeImputer, each feature is taken one by one as the target variable, while the other features are used as predictors. A regression model is trained on the observed values and then used to predict the missing values. This process is performed **iteratively in multiple rounds**, refining the estimates at each step, until the final imputed values are produced.

```
# Load Dataset
ii = sns.load_dataset('titanic')

# Enable IterativeImputer from experimental module
from sklearn.experimental import enable_iterative_imputer

# Impute age column missing values using IterativeImputer with
Median strategy
from sklearn.impute import IterativeImputer
```

```
imputer = IterativeImputer()
# imputer = IterativeImputer(max_iter=10, random_state=0,
n_nearest_features=5)
ii['age'] = imputer.fit_transform(ii[['age']])

# Missing Values Analysis
ii.isnull().sum().sort_values(ascending=False)
```

Output:

```
deck           688
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
age              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

1.11.3. Nearest Neighbors Imputation (KNNImputer):

Nearest Neighbors Imputation is a method for handling missing values, in which each missing feature value is estimated using the values of the *k* nearest sample = rows (neighbors – data point) that have observed values for that feature.

Distances between samples = rows (data points) are computed using a Euclidean metric that supports missing values, and the imputed value is obtained by taking a uniform or distance-weighted average of the neighbors' feature values.

```
# Load Dataset
knni = sns.load_dataset('titanic')
```

```
# Impute age column missing values using KNNImputer
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=5, weights='uniform')
knni['age'] = imputer.fit_transform(knni[['age']])

# Missing Values Analysis
knni.isnull().sum().sort_values(ascending=False)
```

Output:

```
deck           688
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
age              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

1.11.4. Marking Imputed Values (SimpleImputer, MissingIndicator):

Marking imputed values means **adding extra columns that show whether a value was originally missing or not**.

The **MissingIndicator** transformer creates a **binary matrix**:

- 1 → value was **missing**
- 0 → value was **present**

This is useful because **after imputation**, the model cannot naturally know which values were originally missing. Sometimes, *the fact that a value was missing itself carries important information*.

Table 1.11.4.4. Marking Missing Values Example

Original Data

Age
25
NaN

→

After Imputation (Age filled)

Age
25
26

→

After adding MissingIndicator

Age	Age_missing
25	0
26	1

MissingIndicator adds binary features that indicate which values were originally missing, so this information is preserved after imputation.

- It only **marks missing positions**
- Imputation still fills the values

```
from sklearn.impute import MissingIndicator, SimpleImputer

# Load dataset
mi = sns.load_dataset('titanic')

# Create MissingIndicator object
indicator = MissingIndicator(features='missing-only')

# Find where values are missing
missing_mask = indicator.fit_transform(mi[['age']])
```

Output:

```
   array([[False],
   [False],
   [False],
   [False],
   [False],
   [False],
   ...
   [False],
   [False],
   [True],
   [False],
   [False]])
```

```
# Impute missing age values using median
imputer = SimpleImputer(strategy='median')
mi['age'] = imputer.fit_transform(mi[['age']])

# Check remaining missing values
mi.isnull().sum().sort_values(ascending=False)
```

Output:

```
deck           688
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
age              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

1.12. Conclusion

Missing Values – A Major Challenge but Also a Valuable Opportunity

Dealing with missing values is certainly a challenge for every data scientist, but it also provides an important learning opportunity. It helps us understand how to improve data quality and make our datasets more reliable. In the end, high-quality data leads to more accurate, intelligent insights and better-performing models.