Swiss UMEF

UNIVERSITY OF APPLIED SCIENCES INSTITUTE

Big Data Analytics

04: In-Memory Analytics with Pandas. Data Cleaning and Preparation

Instructor: Oleh Tymchuk

#04: Agenda

- Introduction to Data Cleaning
- Handling Missing Data
- Detecting and Removing Duplicates
- Fixing Data Types and Formatting
- Practical cases
- Useful Links

Introduction to Data Cleaning

Why Data Cleaning is Important?

- Real-world data is often messy: missing values, duplicates, inconsistencies
- Data Cleaning is a process of detecting and correcting (or removing) errors, inconsistencies, and inaccuracies in datasets
- Clean data ensures data quality and reliability for accurate analysis
- Saves time and resources by preventing errors in downstream tasks
- A crucial step in any data-driven workflow

Common Data Issues

- Missing or null values
- Duplicates and inconsistencies
- Incorrect data types
- Outliers and incorrect formatting

Handling Missing Data

Identifying Missing Data

- Missing data refers to the absence of values in a dataset
- Represented as NaN (Not a Number) or None in Pandas
- Why Identify Missing Data?
  - Missing data can lead to incorrect analysis or modeling results.
  - Helps decide the appropriate strategy for handling it.
- How to Identify Missing Data

check for missing values:

```
df.isnull()
```

count missing values per column:

```
df.isnull().sum()
```

Strategies for Handling Missing Data

Remove Missing Data

```
df.dropna()            # Drop rows with any missing values
df.dropna(axis=1)      # Drop columns with any missing values
```

Use when missing data is minimal and doesn't affect analysis

Strategies for Handling Missing Data

Fill Missing Data

- Fill with a constant value

```
df.fillna(0)  # Fill with 0
```

- Fill with statistical measures

```
df.fillna(df.mean())  # Fill with column mean
df.fillna(df.median())  # Fill with column median
```

- Forward or backward fill

```
df.fillna(method='ffill')  # Forward fill
df.fillna(method='bfill')  # Backward fill
```

Strategies for Handling Missing Data

Estimate missing values using interpolation

```
df.interpolate()
```

Use machine learning models to predict missing values

Example: K-Nearest Neighbors (KNN) imputation

Best Practices

- Understand the reason for missing data (e.g., random or systematic)
- Choose a strategy based on the context and impact on analysis

Example. Mean

```
data = pd.Series([30, np.nan, 10, 40, np.nan, 20, np.nan])
```

```
filled_mean = data.fillna(data.mean())
```

- **How it works?**
  - The mean is calculated as:

$$\frac{30 + 10 + 40 + 20}{4} = 25$$

  - Every `NaN` is replaced with `25`.
  - *Result:**

```
0    30.0
1    25.0
2    10.0
3    40.0
4    25.0
5    20.0
6    25.0
dtype: float64
```

*When to use it?**

✅ If data is **evenly distributed** without extreme values (outliers).

✅ Example: **Filling missing test scores when calculating class averages.**

Example. Median

```python
data = pd.Series([30, np.nan, 10, 40, np.nan, 20, np.nan])
```

```python
filled_median = data.fillna(data.median())
```

- **How it works?**
  - The median is the middle value of the sorted list: `[10, 20, 30, 40]`.
  - Since we have **an even number of values**, the median is:

$$\frac{20+30}{2}=25$$

  - Every `NaN` is replaced with `25`.

Result:

```
0    30.0
1    25.0
2    10.0
3    40.0
4    25.0
5    20.0
6    25.0
dtype: float64
```

When to use it?

✅ If data has **outliers** (e.g., extreme high or low values).

✅ Example: **Filling missing student salaries in a dataset (since some may have very high salaries, affecting the mean).**

Example. Interpolate

```python
data = pd.Series([30, np.nan, 10, 40, np.nan, 20, np.nan])
```

```python
filled_interpolate = data.interpolate()
```

- **How it works?**
  - Missing values are **estimated based on nearby values.**
  - Calculation:
    1. Between 30 and 10:
$$\frac{30+10}{2} = 20.0$$
    2. Between 10 and 40:
$$10 + \frac{40-10}{2} = 25.0$$
    3. Between 40 and 20:
$$40 + \frac{20-40}{2} = 30.0$$

Result:

```
0    30.0
1    20.0
2    10.0
3    40.0
4    25.0
5    20.0
6    30.0
dtype: float64
```

When to use it?

✅ If data represents **a continuous process**, such as time series data.

✅ Example: **Filling missing temperature readings from a weather station.**

Comparison Table

Method	When to Use?	Example
`mean()` (average)	Data is evenly distributed, no outliers	Student test scores
`median()`	There are outliers, need a "central" value	Student salaries
`interpolate()`	Data changes smoothly over time	Temperature sensor data

Comparison Table

Method	How it works?	When to use?
`fillna(value)`	Replaces `NaN` with a fixed value	When `NaN` means missing data (e.g., warehouse stock)
`fillna(method='ffill')`	Copies the previous value	Time series, such as temperature or sales data
`fillna(method='bfill')`	Copies the next value	When future values are more reliable (e.g., expected payments)
`fillna(limit=n, method=...)`	Limits the number of copied values	When long gaps shouldn't be blindly filled
`dropna()`	Removes rows with `NaN`	When missing data is minimal and can be ignored

Detecting and Removing Duplicates

Identifying and Removing Duplicates

What are Duplicates?

- Duplicates are repeated rows in a dataset.
- They can occur due to data entry errors, merging datasets, or other reasons.

Why Remove Duplicates?

- Duplicates can skew analysis and lead to incorrect results.
- They waste storage and computational resources.

Detecting duplicates: `df.duplicated()`

Removing duplicates: `df.drop_duplicates()`

Fixing Data Types and Formatting

Handling Incorrect Data Types

What are Incorrect Data Types?

- Data stored in a format that doesn't match its intended type (e.g., numbers stored as strings, dates stored as text)

Why Fix Data Types?

- Ensures proper analysis and computation (e.g., arithmetic operations on numeric data)
- Enables use of specialized functions (e.g., date operations)

Common Data Type Issues

- Numeric data stored as strings (e.g., "123" instead of 123)
- Dates stored as strings (e.g., "2023-10-01" instead of datetime)
- Categorical data stored as strings or numbers

How to Fix Data Types

- Converting data types: .astype()
- Parsing dates: pd.to_datetime()
- Handling categorical data: .astype('category')

Fixing Inconsistent Data

What is Inconsistent Data?

- Data that doesn’t follow a standard format or convention (e.g., mixed cases, extra spaces, inconsistent units)

Why Fix Inconsistent Data?

- Ensures uniformity for accurate analysis and reporting
- Improves readability and usability of the dataset

Common Inconsistencies

- Text: Mixed cases ("New York" vs. "new york"), extra spaces
- Units: Inconsistent measurements (e.g., "kg" vs. "lbs")
- Categories: Different spellings or representations (e.g., "USA" vs. "U.S.A.")

How to Fix Inconsistent Data

- Standardizing text format (lowercase, trimming spaces)
- Correcting categorical inconsistencies (e.g., "USA" vs. "United States")
- Replacing incorrect values: .replace()

Data Cleaning Best Practices

- Always check the dataset before analysis
- Keep track of changes for reproducibility
- Validate results after cleaning

Practical cases

Useful Links

[Pandas Cheat Sheet]

[Working with missing data]

[Outlier Detection Techniques in Python]