CHAPTER-07

Data Cleaning and Preparation

JULY 26, 2024

PYTHON_FOR_DATA_ANALYSIS_2ND_EDITION

By Wes McKinney

DATA CLEANING AND PREPROCESSING

1. Handling Missing Data

Missing data is a common issue in datasets and can lead to biased or inaccurate results if not handled properly. Pandas provides various methods to handle missing data in both Series and DataFrames.

Checking for Missing Data

To check for missing data, use the isnull or notnull methods which return a boolean mask indicating where values are missing.

```
import pandas as pd

import numpy as np

# Creating a Series with missing values

string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])

print(string_data.isnull())
```

Dropping Missing Data

You can remove rows or columns with missing data using the dropna method.

```
# DataFrame with missing values

data = pd.DataFrame([[1., 6.5, 3.], [1., np.nan, np.nan],

                     [np.nan, np.nan, np.nan], [np.nan, 6.5, 3.]])

# Dropping rows with missing values

cleaned = data.dropna()

print(cleaned)


# Dropping columns with missing values

cleaned = data.dropna(axis=1)

print(cleaned)
```

2. Filling Missing Data

Filling in missing data can be achieved using the fillna method. This method allows filling with a specified value, filling forward or backward, or filling with a statistical measure such as the mean.

```
# Creating a DataFrame with NaN values

df = pd.DataFrame(np.random.randn(7, 3))

df.iloc[:4, 1] = np.nan

df.iloc[:2, 2] = np.nan


# Filling missing values with a specified value

filled = df.fillna(0)

print(filled)


# Forward filling missing values with a limit

filled = df.fillna(method='ffill', limit=2)

print(filled)


# Filling with the mean of the Series

data = pd.Series([1., np.nan, 3.5, np.nan, 7])

filled = data.fillna(data.mean())

print(filled)
```

3. Removing Duplicates

Removing duplicate rows can be important to ensure the quality of the data. The duplicated method identifies duplicates and drop_duplicates removes them.

```
# Creating a DataFrame with duplicate rows

data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],

                     'k2': [1, 1, 2, 3, 3, 4, 4]})

# Detecting duplicates
```

```
print(data.duplicated())


# Removing duplicates
data_cleaned = data.drop_duplicates()
print(data_cleaned)


# Specifying subset of columns for detecting duplicates
data['v1'] = range(7)
print(data.drop_duplicates(['k1']))


# Keeping the last occurrence of duplicates
print(data.drop_duplicates(['k1', 'k2'], keep='last'))
```

4. Detecting and Filtering Outliers

Outliers can significantly affect the results of data analysis. Pandas provides methods to detect and handle outliers.

Detecting Outliers

You can use boolean indexing to filter out outliers based on their deviation from the mean.

```
# Creating a DataFrame with normally distributed data
data = pd.DataFrame(np.random.randn(1000, 4))
# Detecting outliers in a column
col = data[2]
print(col[np.abs(col) > 3])
```

Capping Outliers

Capping outliers can be done by setting them to a specified range.

```
# Capping values outside a range
data[np.abs(data) > 3] = np.sign(data) * 3
print(data.describe())
```

5. Replacing Values

Replacing specific values in a Series or DataFrame can be done using the replace method.

```
# Creating a Series with sentinel values
data = pd.Series([1., -999., 2., -999., -1000., 3.])

# Replacing a single value
print(data.replace(-999, np.nan))

# Replacing multiple values
print(data.replace([-999, -1000], np.nan))

# Replacing with different values
print(data.replace([-999, -1000], [np.nan, 0]))

# Replacing using a dictionary
print(data.replace({-999: np.nan, -1000: 0}))
```