Data Cleaning in Python

Instructor: Dr. Pornpat Sirithumgul

Data Cleaning

- **Cleaning your data** is a process of removing errors, outliers, and inconsistencies and ensuring that all of your data is in a format that is appropriate for your analysis.
- Data that contains many errors or that hasn't gone through a data-cleaning process is referred to as **dirty data**.

Dirty Data [Example]

Messy_data

ID	Name	Age	Height	Weight	City	Date_of_Birth	Income	Employment_Status
1	John Doe	29	170		New York	1/2/95	50000	Employed
2		34		70.5	Bangkok	7/25/90	NaN	Unemployed
3	Jane Doe	27	168	55		27th August 1997	40000	Employed
4	Jim Smith	29.5	171	60.2	Chicago	12/1/94	55000	Employed
5	Ana Bell	24	169	NaN	Los Angeles	24-Aug-00	45000	Self-Employed
5	Ana Bell	24	169		Los Angeles	24-Aug-00	45000	Self-Employed
6	Chris O'Neil	25	167	70	Boston	NaN	48000	Unemployed
7	Mike Brown		172	68	Miami	1/1/98	52000	
8	Susan Clark	28		55	Paris	2/1/96	49000	Employed
9	Oliver Twist	35	175	75	London	01-01-1989	9999999999	Self-Employed

Key Issues

- Missing Values (e.g., blank, NaN)
- Outliers (e.g., too high salary income)
- Duplicates (e.g., duplicated rows)
- Inconsistencies (e.g., date of birth)

Python Snippet to Detect Missing Values

```
import pandas as pd
import numpy as np

# Define the dataset
data = {
    "ID": [1, 2, 3, 4, 5, 5, 6, 7, 8],
    "Name": ["John Doe", np.nan, "Jane Doe", "Jim Smith", "Ana Bell", "Ana Bell", "Chris O'Neil", "Mike Brown", "Susan Clark"],
    "Age": [29, 34, 27, 29.5, 24, 24, 25, np.nan, 28],
    "Height": [170, np.nan, 168, 171, 169, 169, 167, 172, np.nan],
    "Weight": [np.nan, 70.5, 55, 60.2, np.nan, np.nan, 70, 68, 55],
    "City": ["New York", "Bangkok", np.nan, "Chicago", "Los Angeles", "Los Angeles", "Boston", "Miami", "Paris"],
    "Date_of_Birth": ["01-02-1995", "1990-07-25", "27th August 1997", "12/01/1994", "24-August-2000", "24-August-2000", np.nan, "01-01-1998", "1996/02/01"]
    "Income": [50000, np.nan, 40000, 55000.00, 45000, 45000, 48000, 52000, 49000],
    "Employment_Status": ["Employed", "Unemployed", "Employed", "Employed", "Self-Employed", "Self-Employed", "Unemployed", np.nan, "Employed"]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Detect missing values (NaN and blanks)
missing_summary = df.isnull().sum()
print("Missing values per column:\n", missing_summary)

# Show rows with missing values
rows_with_missing = df[df.isnull().any(axis=1)]
print("\nRows with missing values:\n", rows_with_missing)
```

See Python Code

Deciding to Eliminate or Impute Missing Values

- **Eliminate entire entry:** If missing values are less than 5% of the dataset, drop the rows.
- **Impute missing values:** If missing value exceed 5%, use:
  - Mean for numerical data.
  - Median for ordinal data.
  - Mode for nominal data.

Python Snippet to Eliminate Missing Values

```
import pandas as pd
import numpy as np

# Create a dataset with 3% missing values
data = {
    "ID": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    "Name": ["John Doe", np.nan, "Jane Doe", "Jim Smith", "Ana Bell", "Chris O'Neil", "Mike Brown", "Susan Clark", "Oliver Twist", "Emily Rose"],
    "Age": [29, 34, 27, 29.5, 24, 25, np.nan, 28, 35, 30],
    "Height": [170, np.nan, 168, 171, 169, 167, 172, np.nan, 175, 165],
    "Weight": [np.nan, 70.5, 55, 60.2, np.nan, 70, 68, 55, 75, 60],
    "City": ["New York", "Bangkok", "Paris", "Chicago", "Los Angeles", "Boston", np.nan, "Miami", "London", "San Francisco"],
    "Income": [50000, np.nan, 40000, 55000, 45000, 48000, 52000, 49000, 9999999999, 60000],
    "Employment_Status": ["Employed", "Unemployed", "Employed", "Employed", "Self-Employed", "Unemployed", "Employed", "Employed", "Self-Employed", np.nan]
}

# Calculate total values and 3% of it for missing data validation
total_values = sum(len(col) for col in data.values())
missing_values = sum(pd.isnull(val).sum() for val in data.values())
missing_percentage = (missing_values / total_values) * 100
print(f"Total values: {total_values}, Missing values: {missing_values}, Missing Percentage: {missing_percentage:.2f}%")

# Load the data into a DataFrame
df = pd.DataFrame(data)

# Display the initial DataFrame
print("Original DataFrame:\n", df)

# Drop rows with missing values (NaN or blanks)
df_cleaned = df.dropna()

# Display the cleaned DataFrame
print("\nDataFrame after eliminating rows with missing values:\n", df_cleaned)
```

See Python Code

Python Snippet to Impute Missing Values

Dataset

ID	Age	Shirt_Size	Gender
1	25	M	Male
2	30	L	Female
3	NaN	S	Female
4	35	NaN	Male
5	28	XL	NaN
6	40	M	Male
7	22	XS	Female
8	NaN	NaN	Female

Python Snippet to Impute Missing Values

```
import pandas as pd
import numpy as np

# Create the dataset
data = {
    "ID": [1, 2, 3, 4, 5, 6, 7, 8],
    "Age": [25, 30, np.nan, 35, 28, 40, 22, np.nan],  # Numerical data
    "Shirt_Size": ["M", "L", "S", np.nan, "XL", "M", "XS", np.nan],  # Ordinal data
    "Gender": ["Male", "Female", "Female", "Male", np.nan, "Male", "Female", "Female"]  # Nominal data
}

# Load the data into a DataFrame
df = pd.DataFrame(data)

# Define the order for ordinal data
shirt_size_order = {"XS": 1, "S": 2, "M": 3, "L": 4, "XL": 5}

# Map Shirt_Size to numerical values for median calculation
df['Shirt_Size_Encoded'] = df['Shirt_Size'].map(shirt_size_order)

# Impute missing values
# Impute Age (numerical) with mean
df['Age'] = df['Age'].fillna(df['Age'].mean())

# Impute Shirt_Size (ordinal) with median
median_shirt_size = df['Shirt_Size_Encoded'].median()
df['Shirt_Size_Encoded'] = df['Shirt_Size_Encoded'].fillna(median_shirt_size)

# Reverse map Shirt_Size_Encoded to original categories
reverse_shirt_size_order = {v: k for k, v in shirt_size_order.items()}
df['Shirt_Size'] = df['Shirt_Size_Encoded'].map(reverse_shirt_size_order)

# Impute Gender (nominal) with mode
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])

# Drop the encoded column
df = df.drop(columns=['Shirt_Size_Encoded'])

# Display the cleaned DataFrame
print("DataFrame after imputing missing values:\n", df)
```

See Python Code

Python Snippet to Impute Missing Values

```
DataFrame after imputing missing values:
   ID   Age Shirt_Size  Gender
0   1  25.0          M    Male
1   2  30.0          L  Female
2   3  30.0          S  Female
3   4  35.0          M    Male
4   5  28.0         XL  Female
5   6  40.0          M    Male
6   7  22.0         XS  Female
7   8  30.0          M  Female
```

Outliers

Dataset 1 (One-Variable Data)

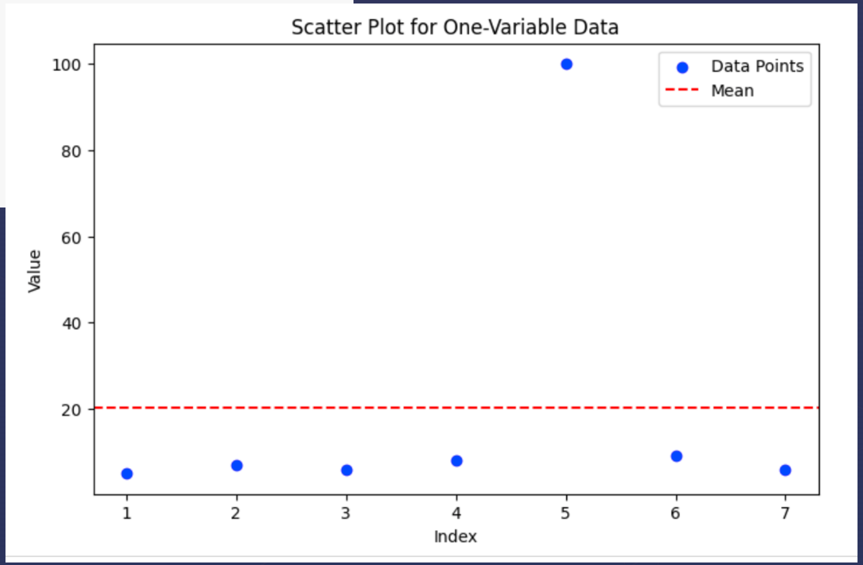
Index	Value
1	5
2	7
3	6
4	8
5	100
6	9
7	6

Using Scatter Plots to Detect Outliers

```python
import pandas as pd
import matplotlib.pyplot as plt

# Create one-variable dataset
data1 = {"Index": [1, 2, 3, 4, 5, 6, 7], "Value": [5, 7, 6, 8, 100, 9, 6]}
df1 = pd.DataFrame(data1)

# Scatter plot for one-variable data
plt.figure(figsize=(8, 5))
plt.scatter(df1['Index'], df1['Value'], color='blue', label='Data Points')
plt.axhline(y=df1['Value'].mean(), color='red', linestyle='--', label='Mean')
plt.title("Scatter Plot for One-Variable Data")
plt.xlabel("Index")
plt.ylabel("Value")
plt.legend()
plt.show()
```



See Python Code

Outliers

Dataset 2 (Two-Variable Data)

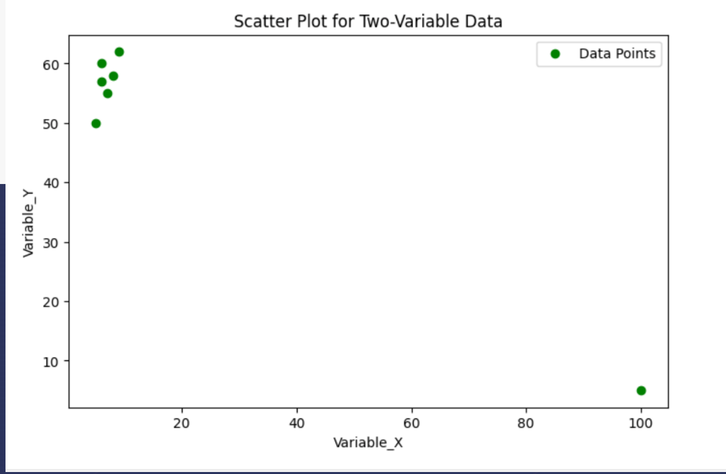
Index	Variable_X	Variable_Y
1	5	50
2	7	55
3	6	60
4	8	58
5	100	5
6	9	62
7	6	57

Using Scatter Plots to Detect Outliers

```
import pandas as pd
import matplotlib.pyplot as plt

# Create two-variable dataset
data2 = {
    "Index": [1, 2, 3, 4, 5, 6, 7],
    "Variable_X": [5, 7, 6, 8, 100, 9, 6],
    "Variable_Y": [50, 55, 60, 58, 5, 62, 57]
}
df2 = pd.DataFrame(data2)

# Scatter plot for two-variable data
plt.figure(figsize=(8, 5))
plt.scatter(df2['Variable_X'], df2['Variable_Y'], color='green', label='Data Points')
plt.title("Scatter Plot for Two-Variable Data")
plt.xlabel("Variable_X")
plt.ylabel("Variable_Y")
plt.legend()
plt.show()
```



See Python Code

Using Z-Scores to Detect Outliers

```
import numpy as np

# Sample weight data with outliers
weights = [60, 62, 65, 68, 70, 72, 75, 100, 78, 80, 82, 85, 150, 88, 90, 92]

# Calculate the z-scores
z_scores = np.abs((np.array(weights) - np.mean(weights)) / np.std(weights))

# Set a threshold for outlier detection (e.g., z-score > 3)
threshold = 3

# Filter out outliers
filtered_weights = [w for w, z in zip(weights, z_scores) if z <= threshold]

print("Original weights:", weights)
print("Filtered weights:", filtered_weights)

print("Outliers are:", list(set(weights) - set(filtered_weights)))
```

```
Original weights: [60, 62, 65, 68, 70, 72, 75, 100, 78, 80, 82, 85, 150, 88, 90, 92]
Filtered weights: [60, 62, 65, 68, 70, 72, 75, 100, 78, 80, 82, 85, 88, 90, 92]
Outliers are: [150]
```

See [Python Code]

Using Interquartile Range (IQR) to Detect Outliers

Generated code may be subject to a license |

```python
import numpy as np

# Sample weight data with outliers
weights = np.array([60, 62, 65, 68, 70, 72, 75, 78, 80, 82, 85, 100, 50, 120])

# Calculate Q1 and Q3
q1 = np.percentile(weights, 25)
q3 = np.percentile(weights, 75)

# Calculate IQR
iqr = q3 - q1

# Calculate outlier boundaries
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr

# Identify outliers
outliers = weights[(weights < lower_bound) | (weights > upper_bound)]

# Print the results
print("Q1:", q1)
print("Q3:", q3)
print("IQR:", iqr)
print("Lower Bound:", lower_bound)
print("Upper Bound:", upper_bound)
print("Outliers:", outliers)
```

```
Q1: 65.75
Q3: 81.5
IQR: 15.75
Lower Bound: 42.125
Upper Bound: 105.125
Outliers: [120]
```

See [Python Code]

Remove Duplicates

```python
import pandas as pd

# Sample data with duplicates
data = {'col1': [1, 2, 2, 3, 4, 4, 5],
        'col2': ['A', 'B', 'B', 'C', 'D', 'D', 'E']}
df = pd.DataFrame(data)

# Display original DataFrame
print("Original DataFrame:")
print(df)

# Remove duplicates based on all columns
df_no_duplicates = df.drop_duplicates()

# Display DataFrame after removing duplicates
print("\nDataFrame after removing duplicates:")
print(df_no_duplicates)
```

```
Original DataFrame:
   col1 col2
0     1    A
1     2    B
2     2    B
3     3    C
4     4    D
5     4    D
6     5    E

DataFrame after removing duplicates:
   col1 col2
0     1    A
1     2    B
3     3    C
4     4    D
6     5    E
```

See [Python Code]

Address Inconsistencies

```python
import pandas as pd

# Sample inconsistent data
data = {'Gender': ['Male', 'M', 'Female', 'F', 'male', 'FEMALE', 'm', 'f']}
df = pd.DataFrame(data)

# Value mapping for gender
gender_mapping = {
    'Male': 'Male',
    'M': 'Male',
    'm': 'Male',
    'Female': 'Female',
    'F': 'Female',
    'f': 'Female',
    'FEMALE': 'Female',
    'male': 'Male'
}

# Apply mapping to the 'Gender' column
df['Gender'] = df['Gender'].map(gender_mapping)

print(df)
```

```
   Gender
0    Male
1    Male
2  Female
3  Female
4    Male
5  Female
6    Male
7  Female
```

See Python Code

Q & A