**Phase 2**

**DATA CLEANING**

* Sangini Pravin Phansekar

**Data cleaning** (or data cleansing) is the process of identifying, correcting, or removing inaccurate, incomplete, inconsistent, or irrelevant data from a dataset. The goal is to improve data quality and make it more accurate, consistent, and usable for analysis, decision-making, or machine learning.

**Why is Data Cleaning Important?**

1. **Improved Accuracy**: Dirty data can lead to incorrect conclusions. Cleaning ensures that the data reflects reality.
2. **Reliable Analysis**: Good data quality leads to better insights, predictions, and decisions.
3. **Efficiency**: Clean data saves time during analysis and prevents errors from propagating in downstream processes.
4. **Consistency**: Ensures that data from different sources is standardized and can be compared or combined.

**Step1:**

**Handling Missing Data**

• Null Values: Missing values (or nulls) in a dataset can distort analysis and lead to incorrect conclusions.

You must decide how to handle these:

**1. Remove Rows/Columns**: Use this when the amount of missing data is small and removing rows/columns won’t significantly affect the analysis.

2. **Imputation (Filling Data)**: Use this when missing data is substantial and you want to maintain data completeness without removing valuable rows/columns.

3. **Forward/Backward Fill & Interpolation**: Use these for time-series or sequential data where the values might logically follow from their neighbors.

**Step2:**

**Remove Duplicates**

1. duplicated(): Identifies which rows are duplicates, returning a boolean Series (True for duplicates).

2. drop\_duplicates(): Removes duplicate rows from the DataFrame.

3. reset\_index(drop=True): Resets the DataFrame index after dropping rows to avoid having gaps in the index.

**Step 3:**

**Fixed the Data types**

1. Data types (numeric, text, date, etc.) should be properly assigned to ensure correct analysis and

operations.

2. Example: Ensure that the "Date" field is in a date format rather than a string format.

**Step 4:**

**Standardize the Data**

• Consistency in Units: Ensure that measurements are consistent across your dataset (e.g., currency should be in the same unit, date format should be standardized).

• Scaling: For certain types of analysis, you may need to normalize numerical values (e.g., scaling data to fit between 0 and 1 for machine learning models).

**Step 5:**

**Remove Outliers**

1. **Visualize the Data**: Use boxplots and scatterplots to identify outliers visually.
2. **Identify Outliers**:
   * Use **Z-score** to identify points that are too far from the mean.
   * Use **IQR** to detect points outside the interquartile range.
3. **Remove or Replace Outliers**:
   * You can **remove** outliers by filtering them out.
   * Or you can **replace** them with a relevant value (mean/median).
4. **Recheck Data**: Ensure that the outliers are removed or handled properly.
5. **Save Cleaned Data**: Save the processed dataset for further analysis.

**Working on Snapdeal data for data cleaning**

**#importing all important libraries**

import pandas as pd # pandas library to work with dataframes

import numpy as np # numpy library to work with numerical or array functions

import matplotlib.pyplot as plt # matplotib is visualization library

import seaborn as sns # seaborn visualization library

%matplotlib inline

# load data

file\_path = r"D:\WebNeuralInfotech\Data Cleaning\snapdeal\_products.csv"

df = pd.read\_csv(file\_path)

**# Check for data types and any null values**

print(df.info())

Output :

A screenshot of a computer program

Description automatically generated

# Check for the first few rows

df.head()

A screenshot of a graph

Description automatically generated

df.describe()

A screenshot of a table

Description automatically generated

#checking Missing values & handling it

print(df.isnull().sum())

A screenshot of a computer screen

Description automatically generated

# Remove the percentage sign and convert to float

df['Rating'] = df['Rating'].replace('%', '', regex=True).astype(float)

# Optional: If you want to make sure the values are within a valid range (e.g., 0-100)

df['Rating'] = df['Rating'].clip(0, 100)

# Check the cleaned 'Rating' column

print(df['Rating'].head())

A screenshot of a computer

Description automatically generated

# Drop columns with too many missing values (threshold can be set based on your needs)

df = df.dropna(axis=1, thresh=len(df)\*0.5) # Drop columns with less than 50% non-null values

# Fill missing numerical columns with the mean or median

df['Rating'] = df['Rating'].fillna(df['Rating'].mean()) # Replace missing ratings with mean

# Fill missing categorical columns with the mode

df['Category'] = df['Category'].fillna(df['Category'].mode()[0]) # Replace missing categories with the most frequent value

# Drop rows where 'Rating' is missing (if applicable)

df = df.dropna(subset=['Rating'])

# Check if there are any missing values left

print(df.isnull().sum())

A white background with black text

Description automatically generated

#Removing duplicates

df = df.drop\_duplicates()

#converting Data types

df['Stock Status'] = df['Stock Status'].astype('category')

# Step 1: Convert the Price column to strings to use .str methods (handle NaN properly)

df['Price'] = df['Price'].astype(str)

# Step 2: Remove unwanted characters ('Rs.') and commas

df['Price'] = df['Price'].str.replace('Rs.', '', regex=True).str.replace(',', '', regex=True)

# Step 3: Replace empty strings with NaN

df['Price'] = df['Price'].replace('nan', np.nan) # Ensure 'nan' string is treated as NaN

# Step 4: Convert the column to numeric (integer values for valid numbers, NaN for invalid)

df['Price'] = pd.to\_numeric(df['Price'], errors='coerce', downcast='integer')

A screenshot of a product list

Description automatically generated

df['Stock Status'] = df['Stock Status'].replace({'Out of Stock': 0, 'In Stock': 1})

A screenshot of a product list

Description automatically generated

plt.figure(figsize=(8, 5))

sns.histplot(df['Rating'], kde=True, bins=10, color='blue')

plt.title('Distribution of Product Ratings')

plt.xlabel('Rating')

plt.ylabel('Frequency')

A graph with a line going up

Description automatically generated

# Boxplot to show rating distribution and outliers

plt.figure(figsize=(8, 5))

sns.boxplot(x=df['Rating'], color='orange')

plt.title('Boxplot of Product Ratings')

plt.show()

A box plot of product ratings

Description automatically generated

# For numerical columns 'Rating' filter extreme outliers (e.g., > 100 or < 0)

df = df[df['Rating'] <= 100]

# Or use the IQR method to filter out outliers

Q1 = df['Rating'].quantile(0.25)

Q3 = df['Rating'].quantile(0.75)

IQR = Q3 - Q1

df = df[(df['Rating'] >= (Q1 - 1.5 \* IQR)) & (df['Rating'] <= (Q3 + 1.5 \* IQR))]

# Boxplot to show rating distribution and outliers

plt.figure(figsize=(8, 5))

sns.boxplot(x=df['Rating'], color='orange')

plt.title('Boxplot of Product Ratings')

plt.show()

A bar chart with a bar and a rating

Description automatically generated with medium confidence