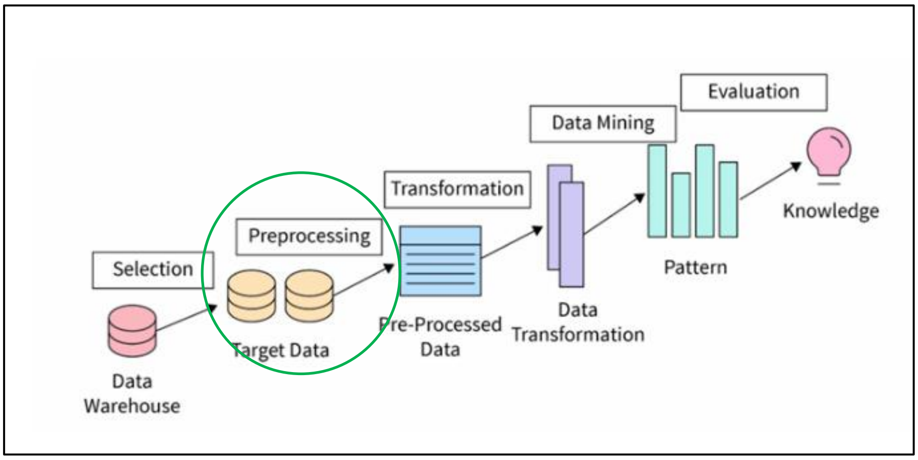
DMBI Experiment 2 – Data Preprocessing using Python

Name: Atharva Lotankar
Class: D15C; **Rnum:** 27
Batch B

Aim: To perform Data Preprocessing using Python.

Theory:

Data Preprocessing as a KDD Process
Data preprocessing is a crucial step in the Knowledge Discovery in Databases (KDD) process. It ensures that raw data, which is often incomplete, noisy, or inconsistent, is transformed into a usable form for further analysis and modeling. By cleaning, integrating, reducing, and transforming data, preprocessing lays the foundation for extracting meaningful patterns.



Importance of Data Preprocessing and Activities
High-quality preprocessing improves accuracy, efficiency, and reliability of data mining models. Key activities include data cleaning (handling missing values and noise), data integration (merging sources), data reduction (dimensionality reduction and sampling), and data transformation (normalization, aggregation, encoding). These steps collectively enhance model performance and interpretability.

Why Python for Data Preprocessing
Python is the preferred language for data preprocessing due to its simplicity, scalability, and extensive ecosystem. Libraries like Pandas, NumPy, and Scikit-learn provide efficient tools for handling, cleaning, and transforming data, making Python both versatile and widely adopted in data science workflows.

Algorithm / Code Snippets:

```
import pandas as pd

import numpy as np


# Load dataset

df = pd.read_csv('cosmetics_fraud_2025.csv')


# 1. Handling missing values:

# Fill numeric missing with mean, categorical with mode

df['Customer_Age'].fillna(df['Customer_Age'].mean(), inplace=True)

df['Customer_Loyalty_Tier'].fillna(df['Customer_Loyalty_Tier'].mode()[0], inplace=True)

df['Payment_Method'].fillna(df['Payment_Method'].mode()[0], inplace=True)


# 2. Remove duplicates

df.drop_duplicates(inplace=True)


# 3. Encode categorical variables using one-hot encoding

cat_cols = ['Customer_Loyalty_Tier', 'Location', 'Product_Category', 'Payment_Method', 'Device_Type']

df_encoded = pd.get_dummies(df, columns=cat_cols)


# 4. Fix datatypes

# Convert Transaction_Date to datetime

df_encoded['Transaction_Date'] = pd.to_datetime(df_encoded['Transaction_Date'])

# Convert Transaction_Time to datetime.time (parsing automatically)

df_encoded['Transaction_Time'] = pd.to_datetime(df_encoded['Transaction_Time']).dt.time


# 5. Handle outliers by capping with IQR method

for col in ['Customer_Age', 'Purchase_Amount']:

    Q1 = df_encoded[col].quantile(0.25)

    Q3 = df_encoded[col].quantile(0.75)

    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR

    upper_bound = Q3 + 1.5 * IQR

    df_encoded[col] = np.where(df_encoded[col] < lower_bound, lower_bound, df_encoded[col])

    df_encoded[col] = np.where(df_encoded[col] > upper_bound, upper_bound, df_encoded[col])
```

Code Implementation:

- The code begins by importing pandas and numpy libraries for data manipulation and numerical operations.
- The dataset is loaded into a pandas DataFrame using pd.read_csv().
- Missing values in numeric column Customer_Age are filled with the mean of that column to avoid losing data.
- Missing categorical values in Customer_Loyalty_Tier and Payment_Method are filled with the most frequent value (mode) from those columns.
- Duplicate rows, if any, are removed using drop_duplicates() to ensure uniqueness of data.
- Categorical columns such as loyalty tier, location, product category, payment method, and device type are converted into numeric form using one-hot encoding (pd.get_dummies()), which creates binary columns for each category.
- The Transaction_Date column is converted from string to a proper datetime format using pd.to_datetime() for easier date/time operations.
- The Transaction_Time column is converted to datetime.time objects to represent the time correctly.
- Outliers in numeric columns Customer_Age and Purchase_Amount are handled by capping values outside the Interquartile Range (IQR). Values below Q1−1.5×IQR$Q1-1.5\times IQR$ are set to the lower bound and above Q3+1.5×IQR$Q3+1.5\times IQR$ are set to the upper bound.
- Finally, after all preprocessing, the cleaned and transformed DataFrame is ready for analysis or modeling.

This sequence ensures the dataset is cleaned, formatted correctly, and numerically encoded for downstream tasks like machine learning or statistical analysis.

Inference:

- Python's pandas library efficiently handles core preprocessing tasks such as filling missing values with statistical imputations (mean/mode), removing duplicates, and encoding categorical variables with one-hot encoding, providing a straightforward and powerful workflow.
- Converting date and time columns into appropriate datetime formats enables easier temporal analysis and feature extraction, while outlier handling using the IQR method helps maintain model robustness by reducing the influence of extreme values.
- This structured step-by-step approach in Python ensures the raw dataset is transformed into a clean, consistent, and machine-learning-ready format, ultimately enhancing the performance and reliability of downstream analytics or predictive models.

Results:

Print final processed dataframe info and first 5 rows

```
print(df_encoded.info())
print(df_encoded.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2133 entries, 0 to 2132
Data columns (total 54 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Transaction_ID      2133 non-null   object
 1   Customer_ID         2133 non-null   object
 2   Transaction_Date    2133 non-null   datetime64[ns]
 3   Transaction_Time    2133 non-null   object
 4   Customer_Age        2133 non-null   float64
 5   Store_ID            2133 non-null   object
 6   Product_SKU         2133 non-null   object
 7   Purchase_Amount     2133 non-null   float64
 8   IP_Address          2133 non-null   object
 9   Fraud_Flag          2133 non-null   int64
 10  Footfall_Count      2133 non-null   int64
```

```
print(df_encoded.info())
print(df_encoded.head())
```

```
None
                         Transaction_ID                           Customer_ID  \
0  702bdd9b-9c93-41e3-9dbb-a849b2422080  119dca0b-8554-4b2d-9bec-e964eaf6af97
1  2e64c346-36bc-4acf-bc2b-8b0fdf46abc5  299df086-26c4-4708-b6d7-fcaeceb14637
2  29ad1278-70ce-421f-8d81-23816b39f4ac  dfa3d24d-b935-49a5-aa1d-7d57a44d8773
3  07dc4894-e0eb-48f1-99a7-1942b1973d9b  7a67e184-9369-49ee-aeac-18f5b51b230f
4  ae407054-5543-429c-918a-cdcc42ea9782  cf14730a-8f5a-453d-b527-39a278852b27

  Transaction_Date Transaction_Time  Customer_Age           Store_ID  \
0       2025-07-27         04:04:15     56.000000        FLAGSHIP-LA
1       2025-03-14         20:23:23     46.000000  BOUTIQUE-SHANGHAI
2       2025-02-20         12:36:02     32.000000        POPUP-TOKYO
3       2025-04-25         19:09:43     60.000000       BOUTIQUE-NYC
4       2025-04-17         14:23:23     41.684262       BOUTIQUE-NYC

        Product_SKU  Purchase_Amount       IP_Address  Fraud_Flag  ...  \
0   NEBULA-SERUM-07           158.24   239.249.58.237           0  ...
1  STELLAR-FOUND-03            86.03     84.49.227.90           0  ...
2    SOLAR-BLUSH-04           255.69     79.207.35.55           0  ...
3    GALAXIA-SET-08           282.76  176.194.167.253           0  ...
4     LUNAR-MASC-02           205.86    166.31.46.111           0  ...
```

```
   Payment_Method_Credit Card  Payment_Method_Debit Card  \
0                       False                      False
1                        True                      False
2                       False                      False
3                       False                      False
4                       False                      False

   Payment_Method_Gift Card  Payment_Method_Mobile Payment  \
0                     False                           True
1                     False                          False
2                      True                          False
3                      True                          False
4                      True                          False

   Device_Type_Desktop  Device_Type_Laptop  Device_Type_Mobile  \
0                 True               False               False
1                False               False               False
2                 True               False               False
```

Conclusion:

The data preprocessing experiment concludes that applying systematic Python techniques—handling missing values, removing duplicates, encoding categorical variables, fixing data types, and managing outliers—transforms raw data into a clean, consistent, and analysis-ready format. This preprocessing significantly improves the reliability and accuracy of subsequent data analysis or machine learning models.