NAME: DISHANT H SHAH **ROLL: 16** **CLASS: BE-CMPN/B**
SUBJECT: SMA **EXPERIMENT: 3**

AIM : Data Cleaning and Storage- Preprocess, filter and store social media data for business (Using Python, MongoDB, R, etc).

Theory:

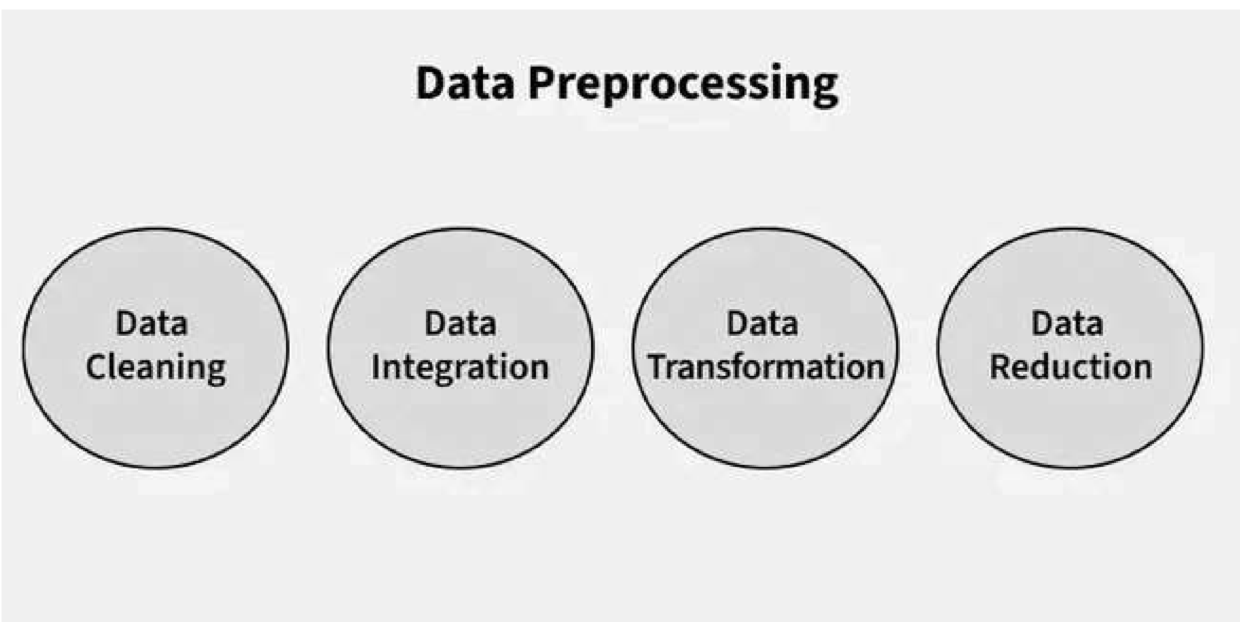
What is preprocessing?

Data preprocessing is the process of preparing raw data for analysis by cleaning and transforming it into a usable format. In data mining it refers to preparing raw data for mining by performing tasks like cleaning, transforming, and organizing it into a format suitable for mining algorithms.

- Goal is to improve the quality of the data.
- Helps in handling missing values, removing duplicates, and normalizing data.
- Ensures the accuracy and consistency of the dataset.

Steps in Data Preprocessing

Some key steps in data preprocessing are Data Cleaning, Data Integration, Data Transformation, and Data Reduction.



1. Data Cleaning: It is the process of identifying and correcting errors or inconsistencies in the dataset. It involves handling missing values, removing duplicates, and correcting incorrect or outlier data to ensure the dataset is accurate and reliable. Clean data is essential for effective analysis, as it improves the quality of results and enhances the performance of data models.

- **Missing Values:** This occur when data is absent from a dataset. You can either ignore the rows with missing data or fill the gaps manually, with the attribute mean, or by using the most probable value. This ensures the dataset remains accurate and complete for analysis.

- **Noisy Data:** It refers to irrelevant or incorrect data that is difficult for machines to interpret, often caused by errors in data collection or entry. It can be handled in several ways:
  - **Binning Method:** The data is sorted into equal segments, and each segment is smoothed by replacing values with the mean or boundary values.
  - **Regression:** Data can be smoothed by fitting it to a regression function, either linear or multiple, to predict values.
  - **Clustering:** This method groups similar data points together, with outliers either being undetected or falling outside the clusters. These techniques help remove noise and improve data quality.
- **Removing Duplicates:** It involves identifying and eliminating repeated data entries to ensure accuracy and consistency in the dataset. This process prevents errors and ensures reliable analysis by keeping only unique records.

2. Data Integration: It involves merging data from various sources into a single, unified dataset. It can be challenging due to differences in data formats, structures, and meanings. Techniques like record linkage and data fusion help in combining data efficiently, ensuring consistency and accuracy.

- **Record Linkage** is the process of identifying and matching records from different datasets that refer to the same entity, even if they are represented differently. It helps in combining data from various sources by finding corresponding records based on common identifiers or attributes.
- **Data Fusion** involves combining data from multiple sources to create a more comprehensive and accurate dataset. It integrates information that may be inconsistent or incomplete from different sources, ensuring a unified and richer dataset for analysis.

3. Data Transformation: It involves converting data into a format suitable for analysis. Common techniques include normalization, which scales data to a common range; standardization, which adjusts data to have zero mean and unit variance; and discretization, which converts continuous data into discrete categories. These techniques help prepare the data for more accurate analysis.

- **Data Normalization**: The process of scaling data to a common range to ensure consistency across variables.
- **Discretization**: Converting continuous data into discrete categories for easier analysis.
- **Data Aggregation**: Combining multiple data points into a summary form, such as averages or totals, to simplify analysis.
- **Concept Hierarchy Generation**: Organizing data into a hierarchy of concepts to provide a higher-level view for better understanding and analysis.

4. Data Reduction: It reduces the dataset's size while maintaining key information. This can be done through feature selection, which chooses the most relevant features, and feature extraction, which transforms the data into a lower-dimensional space while preserving important details. It uses various reduction techniques such as,

- **Dimensionality Reduction (e.g., Principal Component Analysis)**: A technique that reduces the number of variables in a dataset while retaining its essential information.
- **Numerosity Reduction**: Reducing the number of data points by methods like sampling to simplify the dataset without losing critical patterns.
- **Data Compression**: Reducing the size of data by encoding it in a more compact form, making it easier to store and process.

Uses of Data Preprocessing ?

Data preprocessing is utilized across various fields to ensure that raw data is transformed into a usable format for analysis and decision-making. Here are some key areas where data preprocessing is applied:

1. Data Warehousing: In data warehousing, preprocessing is essential for cleaning, integrating, and structuring data before it is stored in a centralized repository. This ensures the data is consistent and reliable for future queries and reporting.

2. Data Mining: Data preprocessing in data mining involves cleaning and transforming raw data to make it suitable for analysis. This step is crucial for identifying patterns and extracting insights from large datasets.

3. Machine Learning: In machine learning, preprocessing prepares raw data for model training. This includes handling missing values, normalizing features, encoding categorical variables, and splitting datasets into training and testing sets to improve model performance and accuracy.

4. Data Science: Data preprocessing is a fundamental step in data science projects, ensuring that the data used for analysis or building predictive models is clean, structured, and relevant. It enhances the overall quality of insights derived from the data.

5. Web Mining: In web mining, preprocessing helps analyze web usage logs to extract meaningful user behavior patterns. This can inform marketing strategies and improve user experience through personalized recommendations.

6. Business Intelligence (BI): Preprocessing supports BI by organizing and cleaning data to create dashboards and reports that provide actionable insights for decision-makers.

7. Deep Learning Purpose: Similar to machine learning, deep learning applications require preprocessing to normalize or enhance features of the input data, optimizing model training processes.

Explain 10 functions with syntax for preprocessing

1. `pd.to_datetime()`

Converts a column to `datetime` type.

```
df['date_column'] = pd.to_datetime(df['date_column'], errors='coerce')
```

- `errors='coerce'` will set invalid dates to `NaT`.

2. `fillna()`

Fills missing values with a specified value or method.

```
df['column_name'] = df['column_name'].fillna(value=0)  # Replace NaN with 0
df['column_name'] = df['column_name'].fillna(method='ffill')  # Forward fill
```

3. `dropna()`

Drops rows or columns with missing values.

```
df = df.dropna(subset=['column_name'])  # Drop rows with NaN in specified column
df = df.dropna(axis=1)  # Drop columns with any NaN values
```

4. `astype()`

Changes the data type of a column.

```
df['column_name'] = df['column_name'].astype('int')  # Convert to integer
df['column_name'] = df['column_name'].astype('float')  # Convert to float
```

5. `str.replace()`

Replaces substrings in string columns.

```
df['column_name'] = df['column_name'].str.replace('old_string', 'new_string')
```

6. `drop_duplicates()`

Removes duplicate rows based on specified columns.

```
df = df.drop_duplicates(subset=['column_name'], keep='first')  # Keep first occurrence
```

7. `scaler.fit_transform()` (from `sklearn.preprocessing`)

Scales data (e.g., StandardScaler for normalization).

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df[['column_name']])
```

8. `LabelEncoder()` (from `sklearn.preprocessing`)

Encodes categorical labels into numeric values.

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df['encoded_column'] = encoder.fit_transform(df['category_column'])
```

9. `get_dummies()`

Converts categorical variable(s) into dummy/indicator variables.

```
df_dummies = pd.get_dummies(df['category_column'], drop_first=True)  # Avoid multicollinearity
```

10. `apply()`

Applies a function along an axis (rows or columns) of a DataFrame.

```
df['new_column'] = df['column_name'].apply(lambda x: x*2)  # Apply function to column
```

Importing dataset

```
import pandas as pd
df = pd.read_csv('/dirty_cafe_sales.csv')

df.head()
```

	Transaction ID	Item	Quantity	Price Per Unit	Total Spent	Payment Method	Location	Transaction Date
0	TXN_1961373	Coffee	2	2.0	4.0	Credit Card	Takeaway	2023-09-08
1	TXN_4977031	Cake	4	3.0	12.0	Cash	In-store	2023-05-16
2	TXN_4271903	Cookie	4	1.0	ERROR	Credit Card	In-store	2023-07-19
3	TXN_7034554	Salad	2	5.0	10.0	UNKNOWN	UNKNOWN	2023-04-27
4	TXN_3160411	Coffee	2	2.0	4.0	Digital Wallet	In-store	2023-06-11

Next steps: Generate code with df | View recommended plots | New interactive sheet

Cleaning

```
df.isna().sum()
```

	0
Transaction ID	0
Item	333
Quantity	138
Price Per Unit	179
Total Spent	173
Payment Method	2579
Location	3265
Transaction Date	159

dtype: int64

```
# 1. Replace invalid values in 'Item' column with mode
import numpy as np
print("\nMissing values in 'Item' before cleaning:")
print(df['Item'].isnull().sum())
print(df['Item'].unique())
df['Item'] = df['Item'].replace(["ERROR","UNKNOWN",np.nan], df['Item'].mode()[0])
print("Missing values in 'Item' after cleaning:")
print(df['Item'].isnull().sum())
```

```
Missing values in 'Item' before cleaning:
333
['Coffee' 'Cake' 'Cookie' 'Salad' 'Smoothie' 'UNKNOWN' 'Sandwich' nan
 'ERROR' 'Juice' 'Tea']
Missing values in 'Item' after cleaning:
0
```

```
# 2. Replace 'UNKNOWN' in 'Quantity' column with median and convert to numeric
print("\nUnique values in 'Quantity' before cleaning:")
print(df['Quantity'].unique())
df['Quantity'] = pd.to_numeric(df['Quantity'], errors='coerce')
df['Quantity'].fillna(df['Quantity'].median(), inplace=True)
print("Unique values in 'Quantity' after cleaning:")
print(df['Quantity'].unique())
```

```
Unique values in 'Quantity' before cleaning:
['2' '4' '5' '3' '1' 'ERROR' 'UNKNOWN' nan]
```

```
Unique values in 'Quantity' after cleaning:
[2. 4. 5. 3. 1.]
<ipython-input-89-237b948dd6d3>:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df['Quantity'].fillna(df['Quantity'].median(), inplace=True)
```

```
# 3. Fill missing values in 'Price Per Unit' with the mean
print("Mean of 'Price Per Unit' before imputation:", pd.to_numeric(df['Price Per Unit'], errors='coerce').mean())
df['Price Per Unit'] = pd.to_numeric(df['Price Per Unit'], errors='coerce')
df['Price Per Unit'].fillna(df['Price Per Unit'].mean(), inplace=True)
print("Mean of 'Price Per Unit' after imputation:", df['Price Per Unit'].mean())
```

```
Mean of 'Price Per Unit' before imputation: 2.949984155487483
Mean of 'Price Per Unit' after imputation: 2.949984155487483
<ipython-input-90-34493b1f526a>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df['Price Per Unit'].fillna(df['Price Per Unit'].mean(), inplace=True)
```

```
# 4. Recalculate 'Total Spent' using Quantity * Price Per Unit
df['Total Spent'] = df['Quantity'] * df['Price Per Unit']
```

```
# 5. Fill missing values in 'Payment Method' with most frequent value
print("\nMost frequent 'Payment Method':", df['Payment Method'].mode()[0])
print("before cleaning",df['Payment Method'].unique())
# df['Payment Method'].fillna(df['Payment Method'].mode()[0], inplace=True)
df['Payment Method'] = df['Payment Method'].replace(["ERROR","UNKNOWN",np.nan], df['Payment Method'].mode()[0])
print("after cleaning",df['Payment Method'].unique())
```

```

Most frequent 'Payment Method': Digital Wallet
before cleaning ['Credit Card' 'Cash' 'UNKNOWN' 'Digital Wallet' 'ERROR' nan]
after cleaning ['Credit Card' 'Cash' 'Digital Wallet']
```

```
# 6. Fill missing values in 'Location' with most frequent value
print("\nMost frequent 'Location':", df['Location'].mode()[0])
print("before cleaning",df['Location'].unique())
df['Location'] = df['Location'].replace(["ERROR","UNKNOWN",np.nan], df['Location'].mode()[0])
print("After cleaning",df['Location'].unique())
```

```

Most frequent 'Location': Takeaway
before cleaning ['Takeaway' 'In-store' 'UNKNOWN' nan 'ERROR']
After cleaning ['Takeaway' 'In-store']
```

```
# 7. Convert 'Transaction Date' to datetime and handle errors
print("\nInvalid dates in 'Transaction Date' before cleaning:")
print(df[~df['Transaction Date'].str.match(r'\d{4}-\d{2}-\d{2}', na=False)])

df['Transaction Date'] = pd.to_datetime(df['Transaction Date'], errors='coerce')

df = df.dropna(subset=['Transaction Date'])

print("Invalid dates in 'Transaction Date' after cleaning:")
print(df[df['Transaction Date'].isnull()])
```

```

Invalid dates in 'Transaction Date' before cleaning:
     Transaction ID      Item  Quantity  Price Per Unit  Total Spent  \
11      TXN_3051279  Sandwich       2.0             4.0          8.0
29      TXN_7640952      Cake       4.0             3.0         12.0
33      TXN_7710508     Juice       5.0             1.0          5.0
77      TXN_2091733     Salad       1.0             5.0          5.0
103     TXN_7028009      Cake       4.0             3.0         12.0
...             ...       ...       ...             ...          ...
9933    TXN_9460419      Cake       1.0             3.0          3.0
9937    TXN_8253472      Cake       1.0             3.0          3.0
9949    TXN_3130865     Juice       3.0             3.0          9.0
```

```
9983    TXN_9226047  Smoothie       3.0             4.0         12.0
9988    TXN_9594133      Cake       5.0             3.0         15.0

      Payment Method  Location Transaction Date
11       Credit Card  Takeaway            ERROR
29    Digital Wallet  Takeaway            ERROR
33              Cash  Takeaway            ERROR
77    Digital Wallet  In-store              NaN
103   Digital Wallet  Takeaway            ERROR
...              ...       ...              ...
9933  Digital Wallet  Takeaway          UNKNOWN
9937  Digital Wallet  Takeaway          UNKNOWN
9949  Digital Wallet  In-store          UNKNOWN
9983            Cash  Takeaway          UNKNOWN
9988  Digital Wallet  Takeaway              NaN

[460 rows x 8 columns]
Invalid dates in 'Transaction Date' after cleaning:
Empty DataFrame
Columns: [Transaction ID, Item, Quantity, Price Per Unit, Total Spent, Payment Method, Location, Transaction Date]
Index: []
```

```
# 8. Drop rows where 'Transaction ID' is missing (shouldn't happen but good practice)
df.dropna(subset=['Transaction ID'], inplace=True)
```

```
<ipython-input-95-3f7b92615444>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  df.dropna(subset=['Transaction ID'], inplace=True)
```

```
# 9. Standardize column names (convert to lowercase and replace spaces with underscores)
df.columns = df.columns.str.lower().str.replace(" ", "_")
```

```
df.isna().sum()
```

	0
transaction_id	0
item	0
quantity	0
price_per_unit	0
total_spent	0
payment_method	0
location	0
transaction_date	0

dtype: int64

CONCLUSION

Data preprocessing is a crucial step in data analysis that enhances data quality by cleaning, transforming, and organizing raw data for better insights. It involves handling missing values, removing duplicates, and normalizing data to ensure consistency and accuracy. Techniques like data integration, transformation, and reduction streamline large datasets, making them more manageable for analysis. Various preprocessing functions in Python, such as `fillna()`, `dropna()`, and `LabelEncoder()`, help in structuring data effectively. Preprocessed data improves machine learning model performance by reducing noise and ensuring meaningful feature representation. It is widely used in fields like data mining, business intelligence, and deep learning to derive valuable insights. Effective preprocessing ensures that data-driven decisions are accurate, reliable, and efficient.