DATA CLEANING

&

PREPROCESSING

in Machine Learning

Submitted By:
Janeeta Ishtiaq

Data Cleaning & Preprocessing in Machine Learning

Before training any machine learning model, data must be cleaned and transformed into a usable format. Raw data often contains **missing values, inconsistent formats, duplicates, outliers, and irrelevant information**. This article walks through important preprocessing techniques with Python examples.

1 Handling Missing Values

Missing values are very common in datasets (e.g., missing ages, salaries, or survey answers). If left untreated, they can bias results or break algorithms.

Common solutions include:

- **Dropping** missing rows or columns (only if data loss is small).
- **Filling** missing values with statistical measures like:
  - Mean (for continuous numerical data)
  - Median (when data is skewed)
  - Mode (for categorical data)

```
#1-Handling missing values
data1={
    "Name":["Janeeta","Ali","Sana","Hassan","Ahmed"],
    "Age":[22,np.nan,30,25,28],
    "Salary":[50000,40000,np.nan,80000,90000]
}
df1=pd.DataFrame(data1)
print(df1)
#Drop missing rows/columns:
# df.dropna(inplace=True)
# print(f"Drop missing rows/columns:\n {df}")

#Fill with mean/median/mode:
df1.fillna({"Age":int(df1["Age"].mean()),"Salary":int(df1["Salary"].median())},inplace=True)
print(f"Fill with mean/median/mode:\n {df1}")
```

Output:

```
      Name   Age   Salary
0  Janeeta  22.0  50000.0
1      Ali   NaN  40000.0
2     Sana  30.0      NaN
3   Hassan  25.0  80000.0
4    Ahmed  28.0  90000.0
Drop missing rows/columns:
      Name   Age   Salary
0  Janeeta  22.0  50000.0
3   Hassan  25.0  80000.0
4    Ahmed  28.0  90000.0
```

```
Fill with mean/median/mode:
      Name   Age   Salary
0  Janeeta  22.0  50000.0
1      Ali  26.0  40000.0
2     Sana  30.0  65000.0
3   Hassan  25.0  80000.0
4    Ahmed  28.0  90000.0
```

2. Data Type Conversion (Dates, Prices)

Datasets often have values stored in the wrong format:

- Dates stored as text (`"2025-02-01"`) instead of **datetime**.
- Prices stored with symbols (`"Rs500"`) instead of numbers.

Converting them to the right format ensures correct calculations, sorting, and analysis. For example:

- Converting `"Rs500"` → `500`
- Extracting day, month, year from `"2025-02-01"`

```python
data2={
    "Date":["2025-02-01","2025-03-21","2025-05-30","2025-07-01"],
    "Price":["Rs500","Rs800","Rs250","Rs1200"]
}
df2=pd.DataFrame(data2)
print(df2)
df2["Date"]=pd.to_datetime(df2["Date"])
df2["Price"] = df2["Price"].replace('Rs', '', regex=True).astype(int)
print(f"After Conversion:\n{df2}")

```

Output:

```
         Date   Price
0  2025-02-01   Rs500
1  2025-03-21   Rs800
2  2025-05-30   Rs250
3  2025-07-01  Rs1200
After Conversion:
        Date  Price
0 2025-02-01    500
1 2025-03-21    800
2 2025-05-30    250
3 2025-07-01   1200
```

3 Text Cleaning (HTML, Stopwords)

When working with textual data (reviews, comments, web pages), the raw text usually contains:

- **HTML tags** (`<p>`, `<b>`)
- **Special characters / punctuation** (`!!!`, `...`)
- **Stopwords** (common words like *is, the, an*) that don't carry useful meaning.

Cleaning involves:

1. Removing HTML tags
2. Removing special characters
3. Converting to lowercase
4. Removing stopwords

This process makes text **simpler and meaningful** for NLP tasks like sentiment analysis.

```
#Text Cleaning(HTML,StopWords)
text = "<p>Hello!!! This is <b>an example</b> sentence...</p>"
cleaned=BeautifulSoup(text,"html.parser").get_text()
cleaned=re.sub(r"[^a-zA-Z]"," ",cleaned)
cleaned=cleaned.lower()
stop_words = set(stopwords.words("english"))
cleaned = " ".join([word for word in cleaned.split() if word not in stop_words])
print(cleaned)
```

Output:

```
hello example sentence
```

4 Normalization & Standardization

Machine learning models perform better when all features are on the **same scale**.

- **Normalization**: Rescales values between `0` and `1`.
  Useful when features have very different ranges (e.g., income vs age).
- **Standardization**: Rescales values so they have **mean = 0** and **standard deviation = 1**.
  Useful for algorithms sensitive to distribution, like **SVM or K-Means**.

```
#Normalization & Standardization
data3=np.array([[150],[160],[170],[180],[190]])
df3=pd.DataFrame(data3,columns=["Heights"])
print(f"Original data:\n {df3}")
minmax_scaler = MinMaxScaler()
df3["Normalized"]=minmax_scaler.fit_transform(df3[["Heights"]])
standard_scaler = StandardScaler()
df3["standardized"]=standard_scaler.fit_transform(df3[["Heights"]])
print(f"After Normalization and Standaradization \n: {df3}")
```

Output:

```
Original data:
    Heights
0       150
1       160
2       170
3       180
4       190
After Normalization and Standaradization
:     Heights  Normalized  standardized
0       150        0.00     -1.414214
1       160        0.25     -0.707107
2       170        0.50      0.000000
3       180        0.75      0.707107
4       190        1.00      1.414214
```

5 Duplicates & Outliers Handling

- **Duplicates**: Sometimes the same rows appear multiple times in data. Removing duplicates avoids bias and redundancy.
- **Outliers**: Extremely high or low values (e.g., salary = 1,000,000 in a dataset of 40k–90k).
  - Can distort averages and affect models.
  - Common approaches:
    - Remove outliers
    - Replace them with mean/median values

```python
#Duplicates & Outliers Handling
data4={
    "Name":["Ali","Sara","Ali"],
    "Age":[22,30,22]
}
df4=pd.DataFrame(data4)
print(f"data with dupliacte rows:\n{df4}")
df4=df4.drop_duplicates(df4)
print (f"After handling duplicates :\n{df4}")

#Outliers
salary = [40000, 45000, 42000, 1000000]
q1, q3 = np.percentile(salary, [25, 75])
iqr = q3 - q1
lower = q1 - 1.5 * iqr
upper = q3 + 1.5 * iqr
outliers = [x for x in salary if x < lower or x > upper]
print(outliers)
salary_no_outliers = [x for x in salary if lower <= x <= upper]
mean_value = int(np.mean(salary_no_outliers))
salary_mean_replaced = [mean_value if (x < lower or x > upper) else x for x in salary]
print("Replaced with Mean:", salary_mean_replaced)
```

Output:

```
data with dupliacte rows:
   Name  Age
0   Ali   22
1  Sara   30
2   Ali   22
After handling duplicates :
   Name  Age
0   Ali   22
1  Sara   30
```

```
[1000000]
Replaced with Mean: [40000, 45000, 42000, 42333]
```

6 Encoding Categorical Data

Machine learning models work with **numbers**, not text.
Categorical variables (like *city = Lahore, Karachi*) must be converted:

- **Label Encoding**: Assigns each category a numeric value (e.g., *Lahore=0, Karachi=1*).
- **One-Hot Encoding**: Creates separate columns for each category (*City_Lahore, City_Karachi*).
  - Prevents models from assuming order in categories.

```
#Encoding Categorical Data
data5 = {"City": ["Lahore", "Karachi", "Lahore"]}
df5 = pd.DataFrame(data5)
#Label Encoding
le = LabelEncoder()
df5["City_encoded"] = le.fit_transform(df5["City"])
print(df5)
#One-Hot Encoding
df5 = pd.get_dummies(df5, columns=["City"])
print(df5)
```

Output:

```
      City  City_encoded
0   Lahore             1
1  Karachi             0
2   Lahore             1
   City_encoded  City_Karachi  City_Lahore
0             1         False         True
1             0          True        False
2             1         False         True
```

7 Feature Engineering

Feature engineering means **creating new features** from existing data to improve model learning.

Examples:

- **Word count**: Counting words in a review → helps in sentiment analysis.
- **Date parts**: Extracting year, month, weekday from a date → helps in time-series forecasting.

This step makes models smarter by adding **domain knowledge** into features.

```python
#Feature Engineering
#Word Count (Text Data)
df = pd.DataFrame({
    "Review": ["This product is good", "Worst experience"]
})

df["word_count"] = df["Review"].apply(lambda x: len(x.split()))
print(df)

#Date Parts (Date/Time Data)
df = pd.DataFrame({
    "Order_Date": pd.to_datetime(["2024-01-15", "2024-06-30"])
})
df["year"] = df["Order_Date"].dt.year
df["month"] = df["Order_Date"].dt.month
df["day_of_week"] = df["Order_Date"].dt.dayofweek
print(df)
```

Output:

```
                 Review  word_count
0  This product is good           4
1      Worst experience           2
  Order_Date  year  month  day_of_week
0 2024-01-15  2024      1            0
1 2024-06-30  2024      6            6
```

8 Merging Datasets

In real-world projects, data often comes from **multiple sources** (customers table, orders table, transactions table).

Merging datasets combines them into one complete dataset for analysis.
For example: joining **customer info** with **order history** using a common key (`CustomerID`).

```python
#Merge Datasets
customers = pd.DataFrame({
    "CustomerID": [1, 2],
    "Name": ["Ali", "Sana"]
})

orders = pd.DataFrame({
    "OrderID": [101, 102],
    "CustomerID": [1, 2],
    "Amount": [500, 300]
})

merged = pd.merge(orders, customers, on="CustomerID")
print(merged)
```

Output:

```
1 2024-06-30  2024       6           6
   OrderID  CustomerID  Amount  Name
0      101           1     500   Ali
1      102           2     300  Sana
```

Conclusion

Data cleaning & preprocessing are the **foundation of machine learning**.
We covered:

✅Handling missing values
✅ Data type conversion
✅ Text cleaning
✅ Normalization & standardization

✅ Removing duplicates & outliers
✅ Encoding categorical data
✅ Feature engineering
✅ Merging datasets

The cleaner the data → the better the features → the more accurate the model