Data preprocessing is a critical step in the data analysis process. It involves cleaning and transforming raw data into a format that can be easily analyzed and modeled. This includes handling missing values, dealing with outliers, normalizing and scaling data, encoding categorical variables, and other tasks that improve data quality and enhance the accuracy of analytical models.

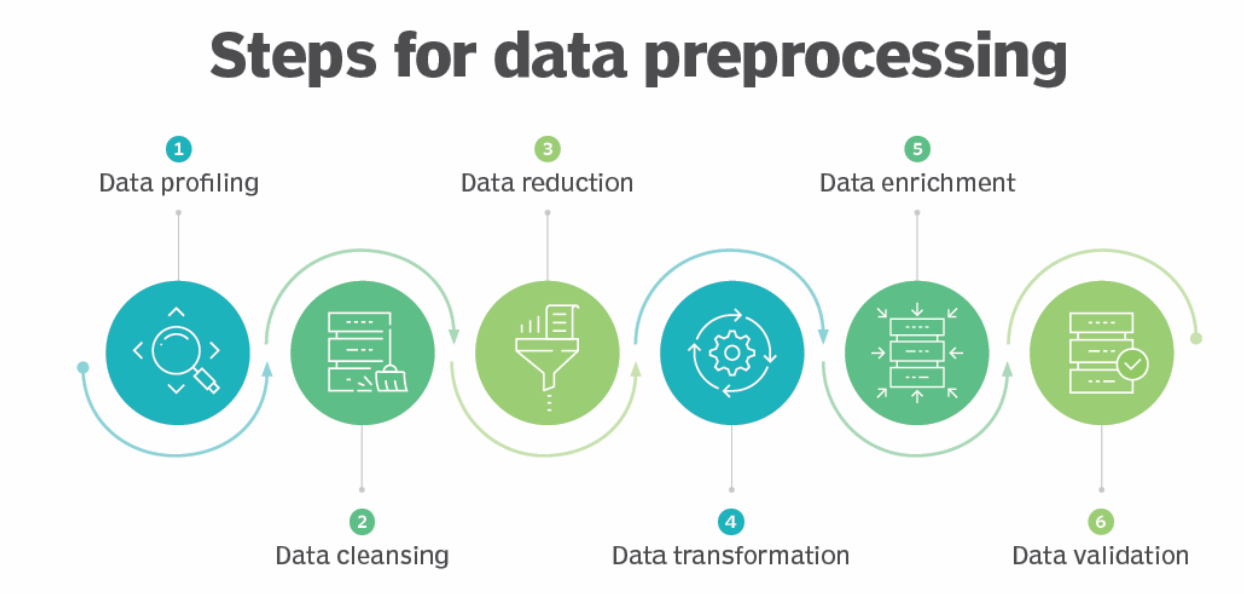

Image Credit: Shehmir Javaid

1. **Handling Missing Values**
   - Use Case: To address missing data in a dataset to improve model accuracy.
   - Steps:
     i. Identify missing values.
     ii. Decide whether to fill or drop them.
     iii. Implement the chosen method.
   - Python Code:

```
import pandas as pd
data = pd.read_csv("dataset.csv")
# Fill missing values
data.fillna(value, inplace=True)
# Or drop missing values
data.dropna(inplace=True)
```

2. **Handling Outliers**
   - Use Case: To minimize the impact of outliers on model performance.
   - Steps:
     i. Identify outliers.
     ii. Decide whether to transform, replace, or remove them.

iii. Implement the chosen method.

- Python Code:

```
import pandas as pd
data = pd.read_csv("dataset.csv")
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1
data = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 *
IQR))).any(axis=1)]
```

3. **Normalization**

- Use Case: To scale numerical data between a specific range, usually 0 to 1.
- Steps:
  i. Choose the range for scaling.
  ii. Apply the normalization formula.
- Python Code:

```
from sklearn.preprocessing import MinMaxScaler
data = pd.read_csv("dataset.csv")
scaler = MinMaxScaler(feature_range=(0, 1))
data_normalized = scaler.fit_transform(data)
```

4. **Standardization**

- Use Case: To bring data to a common scale without distorting differences in ranges.
- Steps:
  i. Calculate mean and standard deviation.
  ii. Apply the standardization formula.
- Python Code:

```
from sklearn.preprocessing import StandardScaler
data = pd.read_csv("dataset.csv")
scaler = StandardScaler()
data_standardized = scaler.fit_transform(data)
```

5. **Encoding Categorical Variables**

- Use Case: To convert categorical variables into numerical format.
- Steps:
  i. Identify categorical variables.
  ii. Decide on the type of encoding (one-hot or label).
  iii. Apply the chosen encoding method.
- Python Code:

```
from sklearn.preprocessing import LabelEncoder
data = pd.read_csv("dataset.csv")
```

```
label_encoder = LabelEncoder()
data['column_name'] =
label_encoder.fit_transform(data['column_name'])
```

6. **Handling Imbalanced Data**

   - Use Case: To address imbalanced classes in the dataset for better model performance.
   - Steps:
     i. Identify the class distribution.
     ii. Decide on the method to balance classes (oversampling, undersampling, SMOTE).
     iii. Implement the chosen method.
   - Python Code:

```
from imblearn.over_sampling import SMOTE
data = pd.read_csv("dataset.csv")
smote = SMOTE(sampling_strategy='auto')
X_res, y_res = smote.fit_resample(data.drop('target', axis=1),
data['target'])
```

7. **Feature Engineering**

   - Use Case: To create new features that better represent the data.
   - Steps:
     i. Understand the domain and existing features.
     ii. Create new features based on existing data.
     iii. Validate the new features.
   - Python Code:

```
import pandas as pd
data = pd.read_csv("dataset.csv")
data['new_feature'] = data['existing_feature1'] *
data['existing_feature2']
```

8. **Data Cleaning**

   - Use Case: To clean the data from any irrelevant or incorrect values.
   - Steps:
     i. Identify irrelevant and incorrect values.
     ii. Decide on the method to clean the data (replace, drop, or fill).
     iii. Implement the chosen method.
   - Python Code:

```
import pandas as pd
data = pd.read_csv("dataset.csv")
data.drop_duplicates(inplace=True)
```

```
data['column_name'].replace(wrong_value, correct_value,
inplace=True)
```

9. **Data Transformation**

   - Use Case: To transform data to better fit the model's assumptions.
   - Steps:
     i. Identify the necessary transformation.
     ii. Apply the transformation.
   - Python Code:

```
import pandas as pd
data = pd.read_csv("dataset.csv")
data['log_transformed'] = np.log(data['column_name'])
```

10. **Handling Time-Series Data**

    - Use Case: To preprocess time-series data for time-dependent analysis.
    - Steps:
      i. Convert the time variable to datetime format.
      ii. Set the time variable as the index.
      iii. Handle missing time points and duplicate entries.
    - Python Code:

```
import pandas as pd
data = pd.read_csv("dataset.csv")
data['time'] = pd.to_datetime(data['time'])
data.set_index('time', inplace=True)
data = data.asfreq('D')
```

Credit: Bytes of Intelligence