Data Cleaning

Type	@datasciencebrain

Data cleaning is a crucial step in machine learning and data science workflows. The quality of the data significantly impacts model performance, and unclean data can lead to misleading insights and poor predictions. Data cleaning involves detecting, diagnosing, and correcting errors or inconsistencies in datasets.

Common Data Issues

1. **Missing Values**
2. **Duplicate Entries**
3. **Outliers**
4. **Inconsistent Formatting**
5. **Incorrect Data Types**
6. **String/Whitespace Issues**
7. **Erroneous or Irrelevant Data**
8. **Mismatched Data**
9. **Scaling and Normalization Issues**

Handling Missing Data

Missing data is a common problem that can arise due to various reasons such as human error, sensor failures, or system limitations.

Techniques to Handle Missing Data

1. **Remove Rows or Columns with Missing Values**
    - If a column has a high percentage of missing values (e.g., > 70%), it might be better to drop it.

- If only a few rows have missing values, they can be removed if the dataset is large.

```
df.dropna()  # Drop rows with missing values
df.dropna(axis=1)  # Drop columns with missing values
```

2. **Impute Missing Values**
    - **Mean/Median Imputation:** For numerical data, replace missing values with the mean or median.

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

    - **Mode Imputation:** For categorical data, replace missing values with the most frequent value.

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

    - **Interpolation:** Useful for time-series data where missing values can be estimated based on surrounding values.

    ```
    df.interpolate(method='linear')
    ```

    - **Using Predictive Models:** Use KNN or regression-based models to estimate missing values.

Best Practices

- Visualize missing data using heatmaps (`seaborn.heatmap(df.isnull(), cbar=False)`) to identify patterns.
- Avoid imputation methods that introduce bias into the dataset.
- Ensure that missing values are handled before applying ML algorithms.

Handling Duplicate Data

Duplicate data can occur due to multiple data sources, system glitches, or improper merging of datasets.

Detecting Duplicates

```
df.duplicated().sum()  # Count duplicate rows
```

Removing Duplicates

```
df.drop_duplicates(inplace=True)
```

Best Practices

- Always check for duplicates before performing analysis.
- Define a unique identifier column to prevent duplicate records.
- Use domain knowledge to determine whether near-duplicate records should be merged or removed.

Handling Outliers

Outliers are extreme values that can distort machine learning models.

Detecting Outliers

1. **Using Boxplots**

   ```
   import seaborn as sns
   sns.boxplot(x=df['column'])
   ```

2. **Using Z-Score**

   ```
   from scipy import stats
   df[(np.abs(stats.zscore(df['column'])) < 3)]
   ```

3. **Using IQR (Interquartile Range)**

```
Q1 = df['column'].quantile(0.25)
Q3 = df['column'].quantile(0.75)
IQR = Q3 - Q1
df_filtered = df[(df['column'] >= Q1 - 1.5*IQR) & (df['column'] <= Q3 + 1.5*IQR)]
```

Handling Outliers

- **Winsorization:** Replace extreme values with percentiles.
- **Transformation:** Log transformations (`np.log1p(df['column'])`) reduce skewness.
- **Capping:** Set upper and lower thresholds based on percentiles.
- **Remove Outliers:** Drop rows containing outliers if they do not provide valuable insights.

Best Practices

- Always check the impact of outliers before removing them.
- Use domain knowledge to determine whether an outlier is a valid data point.
- Log-transform skewed distributions before feeding data into models.

Fixing Inconsistent Formatting

Inconsistencies in data formatting can cause issues in processing and analysis.

Common Fixes

1. **Convert Data Types**

```
df['date'] = pd.to_datetime(df['date'])
df['column'] = df['column'].astype(int)
```

2. **Standardize Categorical Values**

```
df['category'] = df['category'].str.lower().str.strip()
```

3. **Fix Currency and Units**
    - Ensure numerical columns are in the same unit system.
    - Convert currencies using exchange rates before analysis.

Best Practices

- Set strict data type rules during data collection.
- Use automated pipelines to enforce data formatting standards.

Handling String and Whitespace Issues

String-based issues can lead to mismatches and incorrect analysis.

Common Fixes

1. **Trim Whitespace**

    ```
    df['column'] = df['column'].str.strip()
    ```

2. **Remove Special Characters**

    ```
    df['column'] = df['column'].str.replace(r'[^a-zA-Z0-9]', '', regex=True)
    ```

3. **Convert to Lowercase**

    ```
    df['column'] = df['column'].str.lower()
    ```

Best Practices

- Always clean textual data before analysis.
- Use regex to automate string cleaning tasks.

Handling Erroneous or Irrelevant Data

Incorrect or irrelevant data can mislead the model.

Detecting Erroneous Data

- Check for negative values where they are not possible (e.g., negative age).
- Identify logical inconsistencies (e.g., a person's birthdate after their death date).

Fixing Issues

- Remove incorrect entries manually or based on domain rules.
- Replace erroneous values with estimates using imputation.

Best Practices

- Validate data against predefined business rules.
- Use assertions to catch errors early in the pipeline.

Handling Mismatched Data

When merging multiple data sources, inconsistencies may arise.

Common Fixes

1. **Check for Mismatched Column Names**

```
df1.columns, df2.columns
```

2. **Resolve Data Mismatches Before Merging**

```
df1['column'] = df1['column'].astype(str)
df2['column'] = df2['column'].astype(str)
```

3. **Handle Missing Values After Merging**

```
df = pd.merge(df1, df2, on='common_column', how='outer')
```

Best Practices

- Always check for missing and mismatched values after merging.
- Ensure primary key uniqueness before merging datasets.

Scaling and Normalization

Numerical features should be transformed to the appropriate scale.

Methods

1. **Min-Max Scaling**

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[['column']] = scaler.fit_transform(df[['column']])
```

2. **Standardization (Z-score)**

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['column']] = scaler.fit_transform(df[['column']])
```

Best Practices

- Use Min-Max Scaling for neural networks.
- Use Standardization for distance-based models like SVM or KNN.

Tips and Tricks for Efficient Data Cleaning in Machine Learning

1. **Automate Cleaning Pipelines**
    - Use scripts or functions to automate repetitive cleaning tasks.
    - Libraries like `pandas-profiling` and `dataprep` can help with automated EDA.
2. **Understand the Data Before Cleaning**
    - Use `df.describe()`, `df.info()`, and visualizations to explore data before making any changes.

- Look for patterns in missing values instead of blindly filling them.

3. **Use Domain Knowledge**
    - Not all outliers are errors. Some might be important insights (e.g., a sudden stock price jump).
    - Validate missing data handling with real-world understanding.
4. **Avoid Over-Cleaning**
    - Removing too many outliers can lead to loss of valuable information.
    - Do not impute data unless necessary, as it can introduce bias.
5. **Check Data Types Early**
    - Mismatched data types cause hidden errors in processing. Convert them appropriately.
    - Ensure categorical values are stored as `category` types in pandas to save memory.
6. **Use Visualization for Better Insights**
    - Heatmaps (`sns.heatmap(df.isnull(), cbar=False)`) help in identifying missing patterns.
    - Boxplots and histograms help in detecting outliers.
7. **Fix Categorical Data Properly**
    - Convert categories into a consistent format (e.g., lowercase and trimmed).
    - Use encoding techniques (`OneHotEncoder` or `LabelEncoder`) where needed.
8. **Always Back Up the Original Data**
    - Keep a copy of the raw data before cleaning to avoid irreversible mistakes.
    - Use `.copy()` in pandas before modifying DataFrames.
9. **Use Assertions to Validate Cleaning Steps**

```
assert df.isnull().sum().sum() == 0  # Ensure no missing values
assert df.duplicated().sum() == 0  # Ensure no duplicate rows
```

- Helps to quickly validate that cleaning steps are correctly applied.

10. **Handle Date/Time Data Efficiently**

- Convert date columns using `pd.to_datetime()`.
- Extract useful features like `year`, `month`, or `day_of_week`.

1. **Be Careful with Data Leakage**

- Never use future data to fill missing values in a training set.
- Handle missing values separately for train and test datasets.

1. **Standardize Numeric Data for Better Model Performance**

- Scaling (`MinMaxScaler`, `StandardScaler`) ensures numerical stability in ML models.
- Use log transformation for skewed distributions.

1. **Use Feature Engineering to Enhance Data Quality**

- Instead of dropping missing data, create an indicator variable showing where data was missing.
- Combine multiple columns into meaningful new features.

1. **Monitor Memory Usage**

```
df.memory_usage(deep=True).sum() / (1024**2)  # Check DataFrame memory usage in MB
```

- Optimize large datasets by converting data types (`float32` instead of `float64`).

1. **Regularly Validate and Test Data Cleaning Pipelines**

- If using data pipelines, test them on new datasets to ensure robustness.
- Implement logging to track changes applied during cleaning.

By following these tips, you can ensure your data cleaning process is efficient, accurate, and scalable.