Data Cleaning for Machine Learning in Python

1. Remove Duplicate Data

Duplicate rows can skew your analysis.
Use drop_duplicates() to remove them.

```
import pandas as pd

# Sample data with duplicates
data = {'Name': ['Alice', 'Bob', 'Alice', 'Charlie'], 'Age': [25, 30, 25, 35]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

print("============"*2)
# Remove duplicates
df_cleaned = df.drop_duplicates()
print("\033[32mCleaned DataFrame\033[0m")
display(df_cleaned)
```

```
Original DataFrame
```

	Name	Age
0	Alice	25
1	Bob	30
2	Alice	25
3	Charlie	35

```
====================
Cleaned DataFrame
```

	Name	Age
0	Alice	25
1	Bob	30
3	Charlie	35

Prepared by: Syed Afroz Ali
Connect with me on LinkedIn
https://www.linkedin.com/in/syed-afroz-70939914/

2. Handle Missing Values

Missing data can be handled by replacing, removing, or interpolating.

```
# Sample data with missing values
data = {'Name': ['Alice', 'Bob', None, 'Charlie'], 'Age': [25, None, 30, 35]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Replace missing values with a placeholder
df_filled = df.fillna({'Name': 'Rajesh', 'Age': df['Age'].mean()})
print("\033[32mCleaned DataFrame\033[0m")
display(df_filled)
```

Original DataFrame

	Name	Age
0	Alice	25.0
1	Bob	NaN
2	None	30.0
3	Charlie	35.0

Cleaned DataFrame

	Name	Age
0	Alice	25.0
1	Bob	30.0
2	Rajesh	30.0
3	Charlie	35.0

3. Correct Data Types

Ensure data is in the correct format, such as converting strings to datetime.

```
# Sample data with incorrect data types
data = {'Date': ['2023-01-01', '2023-02-01', '2023-03-01'], 'Sales': [100, 150, 200]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Convert 'Date' to datetime
df['Date'] = pd.to_datetime(df['Date'])
print("\033[32mCleaned DataFrame\033[0m")
display(df.dtypes)
```

Original DataFrame

	Date	Sales
0	2023-01-01	100
1	2023-02-01	150
2	2023-03-01	200

Cleaned DataFrame

	0
Date	datetime64[ns]
Sales	int64

dtype: object

4. Standardize Data

Standardize text cases or units of measurement.

```
# Sample data with inconsistent text cases
data = {'Name': ['alice', 'BOB', 'Charlie'], 'Age': [25, 30, 35]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Standardize text to title case
df['Name'] = df['Name'].str.title()
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Name	Age
0	alice	25
1	BOB	30
2	Charlie	35

Cleaned DataFrame

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35

5. Filter Outliers

Identify and address extreme values that may distort analysis.

```
# Sample data with outliers
data = {'Values': [10, 12, 1000, 13]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Calculate IQR
Q1 = df['Values'].quantile(0.25)
Q3 = df['Values'].quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers
df_cleaned = df[(df['Values'] >= lower_bound) & (df['Values'] <=
upper_bound)]

print("\033[32mCleaned DataFrame\033[0m")
display(df_cleaned)
```

Original DataFrame

	Values
0	10
1	12
2	1000
3	13

Cleaned DataFrame

	Values
0	10
1	12
3	13

6. Normalize Data

Scale numerical data to ensure features contribute equally in models.

```
# Sample data for normalization
data = {'Feature1': [100, 200, 300, 500], 'Feature2': [0.5, 0.2, 0.3,.9]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Normalize data using Min-Max scaling
df_normalized = (df - df.min()) / (df.max() - df.min())
print("\033[32mCleaned DataFrame\033[0m")
display(df_normalized)
```

Original DataFrame

	Feature1	Feature2
0	100	0.5
1	200	0.2
2	300	0.3
3	500	0.9

Cleaned DataFrame

	Feature1	Feature2
0	0.00	0.428571
1	0.25	0.000000
2	0.50	0.142857
3	1.00	1.000000

7. Remove Irrelevant Data

Drop columns or rows that do not contribute to the analysis.

```
# Sample data with irrelevant columns
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35], 'Unnecessary': [1, 2, 3]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Drop the 'Unnecessary' column
df_cleaned = df.drop(columns=['Unnecessary'])
print("\033[32mCleaned DataFrame\033[0m")
display(df_cleaned)
```

Original DataFrame

	Name	Age	Unnecessary
0	Alice	25	1
1	Bob	30	2
2	Charlie	35	3

Cleaned DataFrame

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35

8. Validate Data Integrity

Check for inconsistencies or errors, such as typos or incorrect data entries.

```
# Sample data with potential typos
data = {'Name': ['Alice', 'Bob', 'Charly'], 'Age': [25, 30, 35]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Check for typos in 'Name'
valid_names = ['Alice', 'Bob', 'Charlie']
df_cleaned = df[df['Name'].isin(valid_names)]
print("\033[32mCleaned DataFrame\033[0m")
display(df_cleaned)
```

Original DataFrame

	Name	Age
0	Alice	25
1	Bob	30
2	Charly	35

Cleaned DataFrame

	Name	Age
0	Alice	25
1	Bob	30

9. Document Cleaning Steps

Keep a record of all changes made during cleaning for transparency and reproducibility.

```
import pandas as pd
import numpy as np

# Sample data
data = {'Name': ['Alice', 'Bob', 'Charlie', 'Alice'], 'Age': [25, 30, 35,
np.nan], 'Salary': [50000, 60000, 75000, 50000]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Document cleaning steps in a list
cleaning_steps = []

# Step 1: Remove duplicates
df = df.drop_duplicates()
cleaning_steps.append("Removed duplicate rows")

# Step 2: Fill missing values
df['Age'] = df['Age'].fillna(df['Age'].mean())
cleaning_steps.append("Filled missing age values with mean")

# Step 3: Standardize text data (e.g., convert names to uppercase)
df['Name'] = df['Name'].str.upper()
cleaning_steps.append("Standardized names to uppercase")

# Step 4: Remove irrelevant columns (e.g., 'Salary' is not needed)
df = df.drop(columns=['Salary'])
cleaning_steps.append("Removed irrelevant column 'Salary'")

# Step 5: Correct data types (e.g., ensure 'Age' is integer)
df['Age'] = df['Age'].astype(int)
cleaning_steps.append("Converted 'Age' column to integer")
```

```
# Step 6: Filter outliers in 'Age' (e.g., remove ages > 100 or < 0)
df = df[(df['Age'] > 0) & (df['Age'] < 100)]
cleaning_steps.append("Filtered outliers in 'Age' column")

# Step 7: Validate data integrity
if df.isnull().sum().sum() == 0:
    cleaning_steps.append("Validated data integrity: No missing values")
else:
    cleaning_steps.append("Data integrity check failed: Missing values detected")

# Step 8: Reset index after cleaning
df = df.reset_index(drop=True)
cleaning_steps.append("Reset DataFrame index")

print("\033[1mData Cleaning Steps:\033[0m")  # Print in bold
for step in cleaning_steps:
    print(f"- {step}")

print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

	Name	Age	Salary
0	Alice	25.0	50000
1	Bob	30.0	60000
2	Charlie	35.0	75000
3	Alice	NaN	50000

```
Data Cleaning Steps:
- Removed duplicate rows
- Filled missing age values with mean
- Standardized names to uppercase
- Removed irrelevant column 'Salary'
- Converted 'Age' column to integer
- Filtered outliers in 'Age' column
- Validated data integrity: No missing values
- Reset DataFrame index
Cleaned DataFrame
```

	Name	Age
0	ALICE	25
1	BOB	30
2	CHARLIE	35
3	ALICE	30

10. Remove Extra Spaces

Trim leading, trailing, and extra spaces in text data.

```
# Sample data with extra spaces
data = {'Name': [' Alice ', 'Bob  ', '  Charlie']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Remove extra spaces
df['Name'] = df['Name'].str.strip()
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Name
0	Alice
1	Bob
2	Charlie

Cleaned DataFrame

	Name
0	Alice
1	Bob
2	Charlie

11. Handle Inconsistent Categorical Data

Standardize categorical values (e.g., "Male" vs. "M").

```
# Sample data with inconsistent categories
data = {'Gender': ['Male', 'M', 'Female', 'F',
'male']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Standardize categories
df['Gender'] = df['Gender'].replace({'M': 'Male',
'F': 'Female', 'male': 'Male'})
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Gender
0	Male
1	M
2	Female
3	F
4	male

Cleaned DataFrame

	Gender
0	Male
1	Male
2	Female
3	Female
4	Male

Prepared by: Syed Afroz Ali
Connect with me on LinkedIn
https://www.linkedin.com/in/syed-afroz-70939914/

12. Split Columns

Split a column into multiple columns (e.g., splitting "Full Name" into "First Name" and "Last Name").

```
# Sample data with full names
data = {'Full Name': ['Alice Smith', 'Bob Johnson', 'Charlie Brown']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Split into first and last names
df[['First Name', 'Last Name']] = df['Full Name'].str.split(' ', expand=True)
print("\033[32mCleaned DataFrame\033[0m")
display(df
```

Original DataFrame

	Full Name
0	Alice Smith
1	Bob Johnson
2	Charlie Brown

Cleaned DataFrame

	Full Name	First Name	Last Name
0	Alice Smith	Alice	Smith
1	Bob Johnson	Bob	Johnson
2	Charlie Brown	Charlie	Brown

13. Merge Columns

Combine multiple columns into one (e.g., combining "Year", "Month", and "Day" into a single "Date" column).

```
# Sample data with separate date components
data = {'Year': [2023, 2023], 'Month': [1, 2], 'Day': [1, 15]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)


# Combine into a single date column
df['Date'] = pd.to_datetime(df[['Year', 'Month', 'Day']])
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

```
Original DataFrame
```

	Year	Month	Day
0	2023	1	1
1	2023	2	15

```
Cleaned DataFrame
```

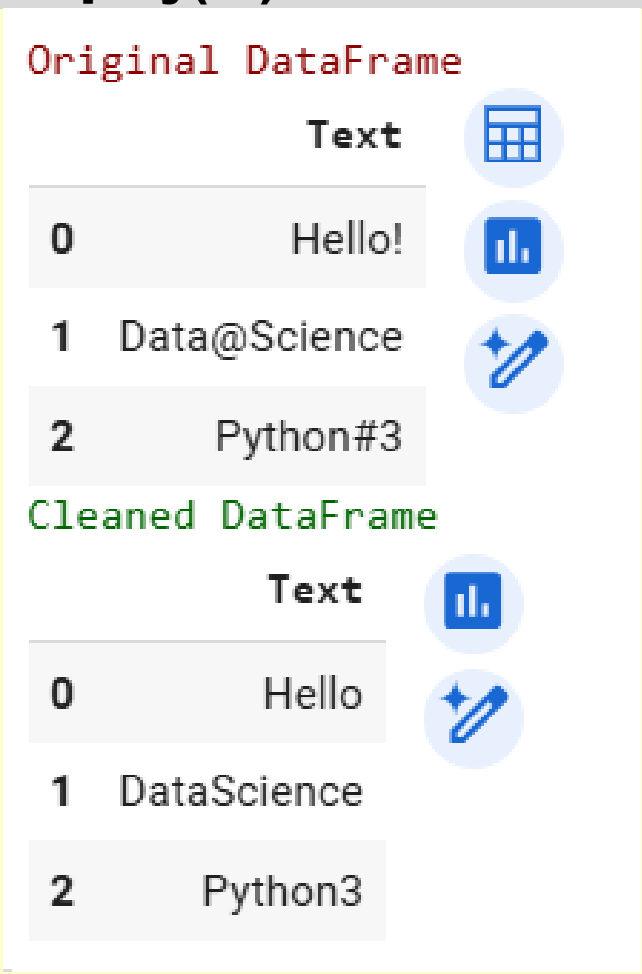
	Year	Month	Day	Date
0	2023	1	1	2023-01-01
1	2023	2	15	2023-02-15

14. Remove Special Characters

Clean text data by removing special characters or symbols.

```
# Sample data with special characters
data = {'Text': ['Hello!', 'Data@Science', 'Python#3']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Remove special characters
df['Text'] = df['Text'].str.replace(r'[^\w\s]', '',
regex=True)
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```



15. Convert Text to Lowercase or Uppercase

Ensure consistent text casing.

```
# Sample data with inconsistent casing
data = {'Text': ['Hello', 'WORLD', 'Python']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Convert to lowercase
df['Text'] = df['Text'].str.lower()
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Text
0	Hello
1	WORLD
2	Python

Cleaned DataFrame

	Text
0	hello
1	world
2	python

16. Remove Unwanted Rows

Filter out rows based on specific conditions.

```
# Sample data with unwanted rows
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, -30, 35]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Remove rows where Age is negative
df = df[df['Age'] >= 0]
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Name	Age
0	Alice	25
1	Bob	-30
2	Charlie	35

Cleaned DataFrame

	Name	Age
0	Alice	25
2	Charlie	35

17. Encode Categorical Variables

Convert categorical data into numerical format for machine learning.

```
# Sample data with categorical variables
data = {'Color': ['Red', 'Blue', 'Green', 'Blue']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# One-hot encoding
df_encoded = pd.get_dummies(df, columns=['Color'])
print("\033[32mCleaned DataFrame\033[0m")
display(df_encoded)
```

```
Original DataFrame
```

	Color
0	Red
1	Blue
2	Green
3	Blue

```
Cleaned DataFrame
```

	Color_Blue	Color_Green	Color_Red
0	False	False	True
1	True	False	False
2	False	True	False
3	True	False	False

18. Check for Data Consistency

Ensure data follows logical rules (e.g., age cannot be negative).

```
# Sample data with inconsistent values
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, -30, 35]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Check for invalid ages
invalid_ages = df[df['Age'] < 0]
print("\033[32mCleaned DataFrame\033[0m")
display("Invalid Ages:", invalid_ages)
```

Original DataFrame

	Name	Age
0	Alice	25
1	Bob	-30
2	Charlie	35

Cleaned DataFrame

'Invalid Ages:'

	Name	Age
1	Bob	-30

19. Aggregate Data

Summarize data by grouping and aggregating (e.g., calculating average sales by region).

```
# Sample data for aggregation
data = {'Region': ['North', 'South', 'North', 'South'],
'Sales': [100, 150, 200, 250]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Aggregate sales by region
df_aggregated =
df.groupby('Region')['Sales'].mean().reset_index()
print("\033[32mCleaned DataFrame\033[0m")
display(df_aggregated)
```

Original DataFrame

	Region	Sales
0	North	100
1	South	150
2	North	200
3	South	250

Cleaned DataFrame

	Region	Sales
0	North	150.0
1	South	200.0

20. Handle Mixed Data Types in a Column

Ensure a column contains only one data type.

```
# Sample data with mixed types
data = {'Values': [10, 'King', 30, '40']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Convert all values to integers
df['Values'] = pd.to_numeric(df['Values'],
errors='coerce')
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Values
0	10
1	King
2	30
3	40

Cleaned DataFrame

	Values
0	10.0
1	NaN
2	30.0
3	40.0

21. Detect and Remove Corrupted Data

Identify and remove rows with corrupted or nonsensical data.

```
# Sample data with corrupted entries
data = {'Name': ['Alice', 'Bob', 'Charlie', '###'], 'Age': [25, 30, 35, -999]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Remove rows with corrupted data
df = df[~df['Name'].str.contains(r'[^a-zA-Z\s]', regex=True)]
df = df[df['Age'] != -999]
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35
3	###	-999

Cleaned DataFrame

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35

22. Reshape Data (Pivot/Unpivot)

Convert data between wide and long formats.

```
# Sample data in wide format
data = {'Year': [2022, 2023], 'Sales_Q1': [100, 150],
'Sales_Q2': [200, 250]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Unpivot to long format
df_long = df.melt(id_vars=['Year'],
var_name='Quarter', value_name='Sales')
print("\033[32mCleaned DataFrame\033[0m")
display(df_long)
```

Original DataFrame

	Year	Sales_Q1	Sales_Q2
0	2022	100	200
1	2023	150	250

Cleaned DataFrame

	Year	Quarter	Sales
0	2022	Sales_Q1	100
1	2023	Sales_Q1	150
2	2022	Sales_Q2	200
3	2023	Sales_Q2	250

23. Check for Data Skewness

Identify and address skewed data distributions.

```
import pandas as pd
import numpy as np
# Sample data with skewed distribution
data = {'Values': [1, 2, 3, 1000, 4, 5]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Check skewness
skewness = df['Values'].skew()
print("Skewness:", skewness)

# Apply log transformation to reduce skewness
df['Values'] = df['Values'].apply(lambda x: np.log(x) if x > 0 else 0)
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Values
0	1
1	2
2	3
3	1000
4	4
5	5

Skewness: 2.4494232090208956

Cleaned DataFrame

	Values
0	0.000000
1	0.693147
2	1.098612
3	6.907755
4	1.386294
5	1.609438

24. Handle Time Zones in Datetime Data

Ensure datetime data is in the correct time zone.

```
# Sample data with datetime
data = {'Timestamp': ['2023-01-01 12:00:00', '2023-01-02 12:00:00']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Convert to datetime and set timezone
df['Timestamp'] = pd.to_datetime(df['Timestamp']).dt.tz_localize('UTC')
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Timestamp
0	2023-01-01 12:00:00
1	2023-01-02 12:00:00

Cleaned DataFrame

	Timestamp
0	2023-01-01 12:00:00+00:00
1	2023-01-02 12:00:00+00:00

25. Handle Imbalanced Data

Address class imbalance in categorical data for machine learning.

```
# Sample data with imbalanced classes
data = {'Class': ['A', 'A', 'A', 'B', 'B', 'A', 'A', 'A', 'A', 'A']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)
# Check class distribution
print(df['Class'].value_counts())

# Use oversampling to balance classes
from sklearn.utils import resample
df_majority = df[df['Class'] == 'A']
df_minority = df[df['Class'] == 'B']

df_minority_upsampled = resample(df_minority, replace=True,
n_samples=len(df_majority), random_state=42)
df_balanced = pd.concat([df_majority, df_minority_upsampled])
print("\033[32mCleaned DataFrame\033[0m")
display(df_balanced['Class'].value_counts())
```

```
Original DataFrame
```

	Class
0	A
1	A
2	A
3	B
4	B
5	A
6	A
7	A
8	A
9	A

```
Class
A    8
B    2
Name: count, dtype: int64
Cleaned DataFrame
```

Class	count
A	8
B	8

26. Detect and Remove Multicollinearity

Identify and remove highly correlated features.

```
# Sample data with correlated features
data = {'Feature1': [1, 2, 3, 4], 'Feature2': [2, 4, 6, 8], 'Feature3': [3, 6, 9, 12]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Calculate correlation matrix
corr_matrix = df.corr()

# Remove highly correlated features
import numpy as np
upper_tri = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.95)]
df_cleaned = df.drop(columns=to_drop)
print("\033[32mCleaned DataFrame\033[0m")
display(df_cleaned)
```

Original DataFrame

	Feature1	Feature2	Feature3
0	1	2	3
1	2	4	6
2	3	6	9
3	4	8	12

Cleaned DataFrame

	Feature1
0	1
1	2
2	3
3	4

27. Handle Noisy Data

Smooth out noisy data using techniques like moving averages.

```
# Sample noisy data
data = {'Values': [10, 12, 15, 100, 14, 13, 16]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Apply moving average smoothing
window_size = 2
df['Smoothed'] =
df['Values'].rolling(window=window_size).mean()
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Values
0	10
1	12
2	15
3	100
4	14
5	13
6	16

Cleaned DataFrame

	Values	Smoothed
0	10	NaN
1	12	11.0
2	15	13.5
3	100	57.5
4	14	57.0
5	13	13.5
6	16	14.5

28. Detect and Handle Data Leakage

Ensure no future information is used in training data.

```
# Sample data with potential leakage
data = {'Date': ['2023-01-01', '2023-01-02', '2023-01-03'], 'Sales': [100, 150, 200], 'Future_Info': [150, 200, 250]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Remove future information
df_cleaned = df.drop(columns=['Future_Info'])
print("\033[32mCleaned DataFrame\033[0m")
display(df_cleaned)
```

```
Original DataFrame
```

	Date	Sales	Future_Info
0	2023-01-01	100	150
1	2023-01-02	150	200
2	2023-01-03	200	250

```
Cleaned DataFrame
```

	Date	Sales
0	2023-01-01	100
1	2023-01-02	150
2	2023-01-03	200

29. Handle Hierarchical Data

Flatten nested or hierarchical data structures.

```
# Sample hierarchical data
data = {'Name': ['Alice', 'Bob'], 'Details': [{'Age': 25,
'City': 'New York'}, {'Age': 30, 'City': 'Los Angeles'}]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Flatten nested data
df_flattened = pd.json_normalize(df['Details'])
df_cleaned = pd.concat([df.drop(columns=['Details']),
df_flattened], axis=1)
print("\033[32mCleaned DataFrame\033[0m")
display(df_cleaned)
```

Original DataFrame

	Name	Details
0	Alice	{'Age': 25, 'City': 'New York'}
1	Bob	{'Age': 30, 'City': 'Los Angeles'}

Cleaned DataFrame

	Name	Age	City
0	Alice	25	New York
1	Bob	30	Los Angeles

30. Detect and Handle Data Drift

Identify changes in data distribution over time.

```
# Sample data with drift
data = {'Date': pd.date_range(start='2023-01-01',
periods=10), 'Values': [10, 12, 15, 100, 14, 13, 16,
200, 18, 20]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Detect drift using statistical tests
from scipy.stats import ks_2samp
train = df[df['Date'] < '2023-01-05']['Values']
test = df[df['Date'] >= '2023-01-05']['Values']
stat, p_value = ks_2samp(train, test)
print("P-value for drift detection:", p_value)
```

```
Original DataFrame
```

	Date	Values
0	2023-01-01	10
1	2023-01-02	12
2	2023-01-03	15
3	2023-01-04	100
4	2023-01-05	14
5	2023-01-06	13
6	2023-01-07	16
7	2023-01-08	200
8	2023-01-09	18
9	2023-01-10	20

```
P-value for drift detection: 0.5523809523809524
```

31. Handle High-Cardinality Categorical Data

Reduce the number of unique categories in high-cardinality features.

```
# Sample data with high-cardinality categories
data = {'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix', 'Philadelphia', 'San Antonio', 'San Diego', 'Dallas', 'San Jose'] * 10}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Group rare categories into 'Other'
threshold = 5
counts = df['City'].value_counts()
df['City'] = df['City'].apply(lambda x: x if counts[x] >= threshold else 'Other')
display(df['City'].value_counts())
```

Original DataFrame

	city
0	New York
1	Los Angeles
2	Chicago
3	Houston
4	Phoenix
...	...
95	Philadelphia
96	San Antonio
97	San Diego
98	Dallas
99	San Jose

100 rows × 1 columns

city	count
New York	10
Los Angeles	10
Chicago	10
Houston	10
Phoenix	10
Philadelphia	10
San Antonio	10
San Diego	10
Dallas	10
San Jose	10

32. Handle Time-Based Data

Resample or aggregate time-series data.

```
# Sample time-series data
data = {'Date': pd.date_range(start='2023-01-01',
periods=10, freq='D'), 'Sales': [10, 15, 20, 25, 30, 35, 40, 45,
50, 55]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Resample to weekly data
df_resampled = df.resample('W', on='Date').sum()
print("\033[32mCleaned DataFrame\033[0m")
display(df_resampled)
```

Original DataFrame

	Date	Sales
0	2023-01-01	10
1	2023-01-02	15
2	2023-01-03	20
3	2023-01-04	25
4	2023-01-05	30
5	2023-01-06	35
6	2023-01-07	40
7	2023-01-08	45
8	2023-01-09	50
9	2023-01-10	55

Cleaned DataFrame

Date	Sales
2023-01-01	10
2023-01-08	210
2023-01-15	105

33. Detect and Handle Data Anomalies

Use statistical or machine learning methods to detect anomalies.

```
# Sample data with anomalies
data = {'Values': [10, 12, 15, 100, 14, 13, 16]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Detect anomalies using Z-score
from scipy.stats import zscore
df['Z-score'] = zscore(df['Values'])
df['Anomaly'] = df['Z-score'].abs() > 2
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Values
0	10
1	12
2	15
3	100
4	14
5	13
6	16

Cleaned DataFrame

	Values	Z-score	Anomaly
0	10	-0.517225	False
1	12	-0.451396	False
2	15	-0.352653	False
3	100	2.445063	True
4	14	-0.385568	False
5	13	-0.418482	False
6	16	-0.319739	False

34. Handle Text Data

Clean and preprocess text data for NLP tasks.

```
# Sample text data
data = {'Text': ['Hello, world!', 'This is a test.', 'Data cleaning is important.']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Remove punctuation and convert to lowercase
import string
df['Cleaned_Text'] = df['Text'].str.translate(str.maketrans('', '', string.punctuation)).str.lower()
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Text
0	Hello, world!
1	This is a test.
2	Data cleaning is important.

Cleaned DataFrame

	Text	Cleaned_Text
0	Hello, world!	hello world
1	This is a test.	this is a test
2	Data cleaning is important.	data cleaning is important

35. Handle Geospatial Data

Clean and preprocess geospatial data.

```
# Sample geospatial data
data = {'City': ['New York', 'Los Angeles', 'Chicago'],
'Latitude': [40.7128, 34.0522, 41.8781], 'Longitude': [-
74.0060, -118.2437, -87.6298]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Validate coordinates
df = df[(df['Latitude'].between(-90, 90)) &
(df['Longitude'].between(-180, 180))]
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	City	Latitude	Longitude
0	New York	40.7128	-74.0060
1	Los Angeles	34.0522	-118.2437
2	Chicago	41.8781	-87.6298

Cleaned DataFrame

	City	Latitude	Longitude
0	New York	40.7128	-74.0060
1	Los Angeles	34.0522	-118.2437
2	Chicago	41.8781	-87.6298

36. Handle Sparse Data

Remove or impute sparse features.

```
# Sample sparse data
data = {'Feature1': [1, 0, 0, 0], 'Feature2': [0, 0, 0, 0],
'Feature3': [1, 1, 1, 1]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Remove sparse features
df = df.loc[:, (df != 0).any(axis=0)]
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Feature1	Feature2	Feature3
0	1	0	1
1	0	0	1
2	0	0	1
3	0	0	1

Cleaned DataFrame

	Feature1	Feature3
0	1	1
1	0	1
2	0	1
3	0	1

37. Handle Data with Mixed Granularity

Aggregate or disaggregate data to a consistent granularity.

```
# Sample data with mixed granularity
data = {'Date': ['2023-01-01', '2023-01-01', '2023-01-02'], 'Sales': [100, 150, 200]}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Aggregate to daily sales
df_aggregated = df.groupby('Date')['Sales'].sum().reset_index()
print("\033[32mCleaned DataFrame\033[0m")
display(df_aggregated)
```

Original DataFrame

	Date	Sales
0	2023-01-01	100
1	2023-01-01	150
2	2023-01-02	200

Cleaned DataFrame

	Date	Sales
0	2023-01-01	250
1	2023-01-02	200

38. Handle Data with Encoding Issues

Fix encoding problems in text data.

```
# Sample data with encoding issues
data = {'Text': [b'Hello, world!', b'This is a test.']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Decode bytes to strings
df['Text'] = df['Text'].str.decode('utf-8')
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

39. Handle Data with Inconsistent Units

Convert data to consistent units.

```
# Sample data with inconsistent units
data = {'Weight': ['10kg', '20kg', '15000g']}
df = pd.DataFrame(data)
print("\033[31mOriginal DataFrame\033[0m")
display(df)

# Convert all weights to kilograms
df['Weight'] = df['Weight'].apply(lambda x: float(x[:-2]) if 'kg' in x else float(x[:-1]) / 1000)
print("\033[32mCleaned DataFrame\033[0m")
display(df)
```

Original DataFrame

	Weight
0	10kg
1	20kg
2	15000g

Cleaned DataFrame

	Weight
0	10.0
1	20.0
2	15.0

Prepared by: Syed Afroz Ali

Connect with me on LinkedIn

https://www.linkedin.com/in/syed-afroz-70939914/