```
import pandas as pd
df=pd.read_csv('/content/Sample - Superstore.csv',encoding='latin-1')


print("\nFirst 5 rows:")
print(df.head())
```

```
First 5 rows:
   Row ID        Order ID  Order Date   Ship Date       Ship Mode Customer ID  \
0       1  CA-2016-152156   11/8/2016  11/11/2016    Second Class    CG-12520
1       2  CA-2016-152156   11/8/2016  11/11/2016    Second Class    CG-12520
2       3  CA-2016-138688   6/12/2016   6/16/2016    Second Class    DV-13045
3       4  US-2015-108966  10/11/2015  10/18/2015  Standard Class    SO-20335
4       5  US-2015-108966  10/11/2015  10/18/2015  Standard Class    SO-20335

     Customer Name    Segment        Country             City  ...  \
0      Claire Gute   Consumer  United States        Henderson  ...
1      Claire Gute   Consumer  United States        Henderson  ...
2  Darrin Van Huff  Corporate  United States      Los Angeles  ...
3   Sean O'Donnell   Consumer  United States  Fort Lauderdale  ...
4   Sean O'Donnell   Consumer  United States  Fort Lauderdale  ...

  Postal Code  Region       Product ID         Category Sub-Category  \
0       42420   South  FUR-BO-10001798        Furniture    Bookcases
1       42420   South  FUR-CH-10000454        Furniture       Chairs
2       90036    West  OFF-LA-10000240  Office Supplies       Labels
3       33311   South  FUR-TA-10000577        Furniture       Tables
4       33311   South  OFF-ST-10000760  Office Supplies      Storage

                                     Product Name     Sales  Quantity  \
0                  Bush Somerset Collection Bookcase  261.9600         2
1  Hon Deluxe Fabric Upholstered Stacking Chairs,...  731.9400         3
2  Self-Adhesive Address Labels for Typewriters b...   14.6200         2
3      Bretford CR4500 Series Slim Rectangular Table  957.5775         5
4                    Eldon Fold 'N Roll Cart System   22.3680         2

   Discount    Profit
0      0.00   41.9136
1      0.00  219.5820
2      0.00    6.8714
3      0.45 -383.0310
4      0.20    2.5164

[5 rows x 21 columns]
```

```
print("\nDataset Info:")
print(df.info())

print("\nBasic Statistics:")
print(df.describe())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9994 non-null   int64
 1   Order ID       9994 non-null   object
 2   Order Date     9994 non-null   object
 3   Ship Date      9994 non-null   object
 4   Ship Mode      9994 non-null   object
 5   Customer ID    9994 non-null   object
 6   Customer Name  9994 non-null   object
```

```
 7   Segment        9994 non-null   object
 8   Country        9994 non-null   object
 9   City           9994 non-null   object
 10  State          9994 non-null   object
 11  Postal Code    9994 non-null   int64
 12  Region         9994 non-null   object
 13  Product ID     9994 non-null   object
 14  Category       9994 non-null   object
 15  Sub-Category   9994 non-null   object
 16  Product Name   9994 non-null   object
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
dtypes: float64(3), int64(3), object(15)
memory usage: 1.6+ MB
None

Basic Statistics:
            Row ID   Postal Code         Sales     Quantity     Discount  \
count  9994.000000   9994.000000   9994.000000  9994.000000  9994.000000
mean   4997.500000  55190.379428    229.858001     3.789574     0.156203
std    2885.163629  32063.693350    623.245101     2.225110     0.206452
min       1.000000   1040.000000      0.444000     1.000000     0.000000
25%    2499.250000  23223.000000     17.280000     2.000000     0.000000
50%    4997.500000  56430.500000     54.490000     3.000000     0.200000
75%    7495.750000  90008.000000    209.940000     5.000000     0.200000
max    9994.000000  99301.000000  22638.480000    14.000000     0.800000

            Profit
count  9994.000000
mean     28.656896
std     234.260108
min   -6599.978000
25%       1.728750
50%       8.666500
75%      29.364000
max    8399.976000
```

```
#Checking for missing values
print("Missing Values:")
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0])
```

```
Missing Values:
Series([], dtype: int64)
```

```
# Check for duplicates
duplicate_count = df.duplicated().sum()
print(f"\nDuplicate Rows: {duplicate_count}")
```

```

Duplicate Rows: 0
```

```
# Check data types
print("\nData Types:")
print(df.dtypes)
```

```

Data Types:
Row ID             int64
Order ID          object
Order Date        object
Ship Date         object
Ship Mode         object
Customer ID       object
Customer Name     object
```

```
Segment           object
Country           object
City              object
State             object
Postal Code        int64
Region            object
Product ID        object
Category          object
Sub-Category      object
Product Name      object
Sales            float64
Quantity           int64
Discount         float64
Profit           float64
dtype: object
```

```
# Check for inconsistent formats in categorical columns
print("\nUnique values in categorical columns:")
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    unique_count = df[col].nunique()
    if unique_count < 20:  # Only show if less than 20 unique values
        print(f"\n{col} ({unique_count} unique values):")
        print(df[col].value_counts().head(10))
```

```
Unique values in categorical columns:

Ship Mode (4 unique values):
Ship Mode
Standard Class    5968
Second Class      1945
First Class       1538
Same Day           543
Name: count, dtype: int64

Segment (3 unique values):
Segment
Consumer       5191
Corporate      3020
Home Office    1783
Name: count, dtype: int64

Country (1 unique values):
Country
United States    9994
Name: count, dtype: int64

Region (4 unique values):
Region
West       3203
East       2848
Central    2323
South      1620
Name: count, dtype: int64

Category (3 unique values):
Category
Office Supplies    6026
Furniture          2121
Technology         1847
Name: count, dtype: int64

Sub-Category (17 unique values):
Sub-Category
Binders        1523
Paper          1370
Furnishings     957
Phones          889
Storage         846
```

```
Art             796
Accessories     775
Chairs          617
Appliances      466
Labels          364
Name: count, dtype: int64
```

```
print("STARTING DATA CLEANING PROCESS")
# Create a copy for cleaning
df_cleaned = df.copy()
cleaning_summary = []

# 5.1: Clean Column Names
print("\n1. Cleaning Column Names...")
original_columns = df_cleaned.columns.tolist()

# Clean column names: lowercase, replace spaces with underscores, remove special characters
df_cleaned.columns = df_cleaned.columns.str.lower().str.replace(' ', '_').str.replace('-', '_')
df_cleaned.columns = df_cleaned.columns.str.replace('[^a-zA-Z0-9_]', '', regex=True)

new_columns = df_cleaned.columns.tolist()
cleaning_summary.append(f"Column names standardized: {len(original_columns)} columns renamed")
print(f"✓ Columns renamed from {original_columns} to {new_columns}")
```

```
STARTING DATA CLEANING PROCESS

1. Cleaning Column Names...
✓ Columns renamed from ['Row ID', 'Order ID', 'Order Date', 'Ship Date', 'Ship Mode', 'Customer ID', 'Customer Nam
```

```
# 5.2: Handle Missing Values
print("\n2. Handling Missing Values...")
initial_missing = df_cleaned.isnull().sum().sum()

# Strategy for different types of missing values
for col in df_cleaned.columns:
    missing_count = df_cleaned[col].isnull().sum()
    if missing_count > 0:
        print(f"  - {col}: {missing_count} missing values")

        # For numerical columns: fill with median
        if df_cleaned[col].dtype in ['int64', 'float64']:
            median_val = df_cleaned[col].median()
            df_cleaned[col].fillna(median_val, inplace=True)
            cleaning_summary.append(f"Filled {missing_count} missing values in {col} with median ({median_val})")

        # For categorical columns: fill with mode or 'Unknown'
        else:
            if df_cleaned[col].mode().empty:
                df_cleaned[col].fillna('Unknown', inplace=True)
                cleaning_summary.append(f"Filled {missing_count} missing values in {col} with 'Unknown'")
            else:
                mode_val = df_cleaned[col].mode()[0]
                df_cleaned[col].fillna(mode_val, inplace=True)
                cleaning_summary.append(f"Filled {missing_count} missing values in {col} with mode ({mode_val})")

final_missing = df_cleaned.isnull().sum().sum()
print(f"✓ Missing values reduced from {initial_missing} to {final_missing}")
```

```
2. Handling Missing Values...
✓ Missing values reduced from 0 to 0
```

```
# 5.3: Remove Duplicates
print("\n3. Removing Duplicates...")
```

```
initial_rows = len(df_cleaned)
df_cleaned = df_cleaned.drop_duplicates()
final_rows = len(df_cleaned)
duplicates_removed = initial_rows - final_rows
cleaning_summary.append(f"Removed {duplicates_removed} duplicate rows")
print(f"✓ Removed {duplicates_removed} duplicate rows. Dataset now has {final_rows} rows")
```

```
3. Removing Duplicates...
✓ Removed 0 duplicate rows. Dataset now has 9994 rows
```

```
# 5.4: Standardize Text Values
print("\n4. Standardizing Text Values...")

# Find text columns that might need standardization
text_columns = df_cleaned.select_dtypes(include=['object']).columns

for col in text_columns:
    # Remove leading/trailing whitespace
    original_values = df_cleaned[col].nunique()
    df_cleaned[col] = df_cleaned[col].astype(str).str.strip()

    # Standardize case for certain columns (like categories, regions, etc.)
    if any(keyword in col.lower() for keyword in ['category', 'region', 'segment', 'ship_mode']):
        df_cleaned[col] = df_cleaned[col].str.title()

    new_values = df_cleaned[col].nunique()
    if original_values != new_values:
        cleaning_summary.append(f"Standardized text in {col}: {original_values} → {new_values} unique values")
        print(f"  ✓ {col}: {original_values} → {new_values} unique values")
```

```
4. Standardizing Text Values...
```

```
# 5.5: Fix Date Formats
print("\n5. Fixing Date Formats...")

# Identify potential date columns
date_columns = [col for col in df_cleaned.columns if 'date' in col.lower()]

for col in date_columns:
    try:
        original_dtype = df_cleaned[col].dtype
        df_cleaned[col] = pd.to_datetime(df_cleaned[col], errors='coerce')
        if df_cleaned[col].dtype != original_dtype:
            cleaning_summary.append(f"Converted {col} to datetime format")
            print(f"  ✓ {col}: {original_dtype} → datetime64")
    except Exception as e:
        print(f"  ⚠ Could not convert {col} to datetime: {str(e)}")
```

```
5. Fixing Date Formats...
  ✓ order_date: object → datetime64
  ✓ ship_date: object → datetime64
```

```
# 5.6: Fix Data Types
print("\n6. Optimizing Data Types...")

# Optimize numerical columns
for col in df_cleaned.columns:
    if df_cleaned[col].dtype == 'object':
        # Try to convert to numeric if possible
        numeric_series = pd.to_numeric(df_cleaned[col], errors='coerce')
        if not numeric_series.isnull().all():
```

```
            # Check if it's integer-like
            if numeric_series.dropna().apply(lambda x: x.is_integer()).all():
                df_cleaned[col] = numeric_series.astype('Int64')  # Nullable integer
                cleaning_summary.append(f"Converted {col} to integer type")
                print(f"  ✓ {col}: object → Int64")
            else:
                df_cleaned[col] = numeric_series
                cleaning_summary.append(f"Converted {col} to float type")
                print(f"  ✓ {col}: object → float64")
```

```
    6. Optimizing Data Types...
```

```
print("DATA VALIDATION")
print("Final Dataset Info:")
print(df_cleaned.info())

print(f"\nFinal Shape: {df_cleaned.shape}")
print(f"Missing Values: {df_cleaned.isnull().sum().sum()}")
print(f"Duplicate Rows: {df_cleaned.duplicated().sum()}")
```

```
DATA VALIDATION
Final Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   row_id         9994 non-null   int64
 1   order_id       9994 non-null   object
 2   order_date     9994 non-null   datetime64[ns]
 3   ship_date      9994 non-null   datetime64[ns]
 4   ship_mode      9994 non-null   object
 5   customer_id    9994 non-null   object
 6   customer_name  9994 non-null   object
 7   segment        9994 non-null   object
 8   country        9994 non-null   object
 9   city           9994 non-null   object
 10  state          9994 non-null   object
 11  postal_code    9994 non-null   int64
 12  region         9994 non-null   object
 13  product_id     9994 non-null   object
 14  category       9994 non-null   object
 15  sub_category   9994 non-null   object
 16  product_name   9994 non-null   object
 17  sales          9994 non-null   float64
 18  quantity       9994 non-null   int64
 19  discount       9994 non-null   float64
 20  profit         9994 non-null   float64
dtypes: datetime64[ns](2), float64(3), int64(3), object(13)
memory usage: 1.6+ MB
None

Final Shape: (9994, 21)
Missing Values: 0
Duplicate Rows: 0
```

```
import numpy as np
# Check for outliers in numerical columns
numerical_cols = df_cleaned.select_dtypes(include=[np.number]).columns
print(f"\nNumerical Columns: {list(numerical_cols)}")
```

```
Numerical Columns: ['row_id', 'postal_code', 'sales', 'quantity', 'discount', 'profit']
```

```
# Step 7: Save Cleaned Dataset
```

```
print("SAVING CLEANED DATASET")

# Save to CSV
df_cleaned.to_csv('superstore_cleaned.csv', index=False)
print("✓ Cleaned dataset saved as 'superstore_cleaned.csv'")
```

```
SAVING CLEANED DATASET
✓ Cleaned dataset saved as 'superstore_cleaned.csv'
```

```
# Step 8: Generate Cleaning Summary Report
print("\n" + "="*50)
print("DATA CLEANING SUMMARY REPORT")
print("="*50)

import datetime

summary_report = f"""
SUPERSTORE DATASET CLEANING SUMMARY
Generated on: {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}

ORIGINAL DATASET:
- Shape: {df.shape}
- Missing Values: {df.isnull().sum().sum()}
- Duplicate Rows: {df.duplicated().sum()}

CLEANED DATASET:
- Shape: {df_cleaned.shape}
- Missing Values: {df_cleaned.isnull().sum().sum()}
- Duplicate Rows: {df_cleaned.duplicated().sum()}

CLEANING ACTIONS PERFORMED:
"""

for i, action in enumerate(cleaning_summary, 1):
    summary_report += f"{i}. {action}\n"

summary_report += f"""
DATA QUALITY IMPROVEMENTS:
- Rows processed: {df.shape[0]:,}
- Final rows: {df_cleaned.shape[0]:,}
- Data quality score: {((df_cleaned.shape[0] - df_cleaned.isnull().sum().sum()) / (df_cleaned.shape[0] * df_cleaned.sh

COLUMN SUMMARY:
"""

for col in df_cleaned.columns:
    col_info = f"- {col}: {df_cleaned[col].dtype}"
    if df_cleaned[col].dtype == 'object':
        col_info += f" ({df_cleaned[col].nunique()} unique values)"
    summary_report += col_info + "\n"

print(summary_report)

# Save summary report
with open('cleaning_summary_report.txt', 'w') as f:
    f.write(summary_report)
print("\n✓ Cleaning summary saved as 'cleaning_summary_report.txt'")
```

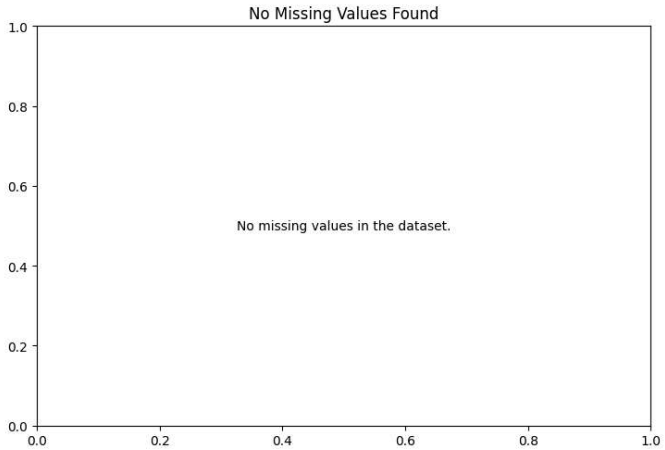
```

==================================================
DATA CLEANING SUMMARY REPORT
==================================================

SUPERSTORE DATASET CLEANING SUMMARY
Generated on: 2025-06-25 06:29:09
```

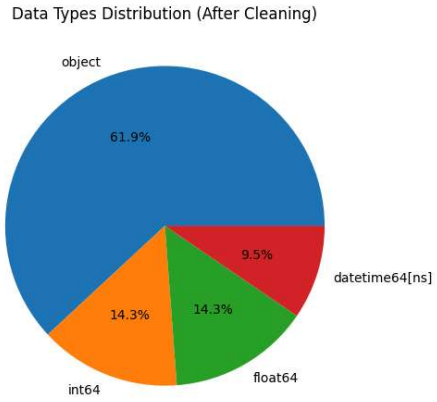
```
ORIGINAL DATASET:
- Shape: (9994, 21)
- Missing Values: 0
- Duplicate Rows: 0

CLEANED DATASET:
- Shape: (9994, 21)
- Missing Values: 0
- Duplicate Rows: 0

CLEANING ACTIONS PERFORMED:
1. Column names standardized: 21 columns renamed
2. Removed 0 duplicate rows
3. Converted order_date to datetime format
4. Converted ship_date to datetime format

DATA QUALITY IMPROVEMENTS:
- Rows processed: 9,994
- Final rows: 9,994
- Data quality score: 4.8%

COLUMN SUMMARY:
- row_id: int64
- order_id: object (5009 unique values)
- order_date: datetime64[ns]
- ship_date: datetime64[ns]
- ship_mode: object (4 unique values)
- customer_id: object (793 unique values)
- customer_name: object (793 unique values)
- segment: object (3 unique values)
- country: object (1 unique values)
- city: object (531 unique values)
- state: object (49 unique values)
- postal_code: int64
- region: object (4 unique values)
- product_id: object (1862 unique values)
- category: object (3 unique values)
- sub_category: object (17 unique values)
- product_name: object (1850 unique values)
- sales: float64
- quantity: int64
- discount: float64
- profit: float64


✓ Cleaning summary saved as 'cleaning_summary_report.txt'
```

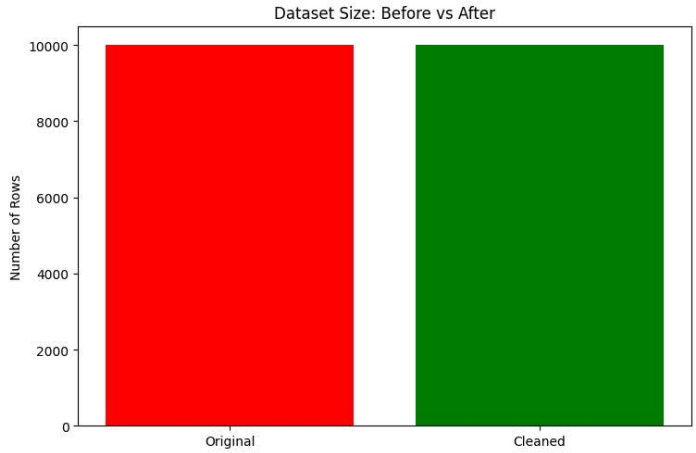
```
# Step 9: Data Quality Visualization
print("\n" + "="*50)
print("GENERATING DATA QUALITY VISUALIZATIONS")
print("="*50)

import matplotlib.pyplot as plt
import seaborn as sns

# Create visualizations to show data quality improvements
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# 1. Missing values comparison
missing_before = df.isnull().sum()
missing_after = df_cleaned.isnull().sum()
comparison_df = pd.DataFrame({
    'Before': missing_before,
    'After': missing_after
}).fillna(0)

# Only plot if there are missing values in at least one of the dataframes
if comparison_df.sum(axis=1).sum() > 0:
    comparison_df[comparison_df.sum(axis=1) > 0].plot(kind='bar', ax=axes[0,0])
```

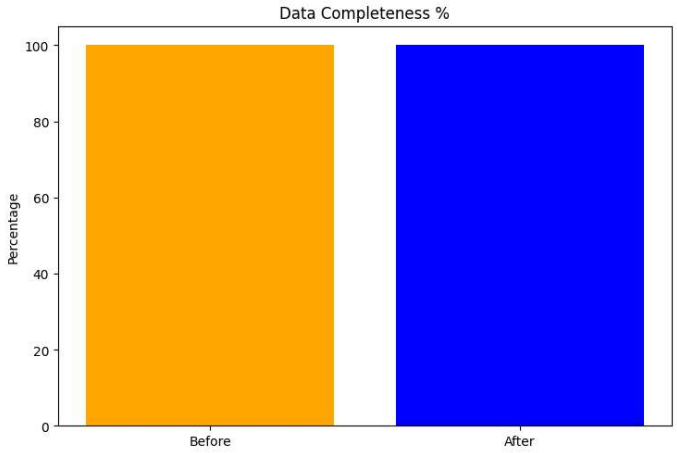
```
        axes[0,0].set_title('Missing Values: Before vs After')
        axes[0,0].set_ylabel('Count')
        axes[0,0].tick_params(axis='x', rotation=45)
    else:
        axes[0,0].set_title('No Missing Values Found')
        axes[0,0].text(0.5, 0.5, 'No missing values in the dataset.', horizontalalignment='center', verticalalignment='cent


    # 2. Data types distribution
    dtype_counts = df_cleaned.dtypes.value_counts()
    axes[0,1].pie(dtype_counts.values, labels=dtype_counts.index, autopct='%1.1f%%')
    axes[0,1].set_title('Data Types Distribution (After Cleaning)')

    # 3. Dataset size comparison
    sizes = ['Original', 'Cleaned']
    row_counts = [df.shape[0], df_cleaned.shape[0]]
    axes[1,0].bar(sizes, row_counts, color=['red', 'green'])
    axes[1,0].set_title('Dataset Size: Before vs After')
    axes[1,0].set_ylabel('Number of Rows')

    # 4. Data quality score
    quality_metrics = ['Complete Data', 'Missing Data']
    before_complete = ((df.shape[0] * df.shape[1]) - df.isnull().sum().sum()) / (df.shape[0] * df.shape[1]) * 100
    after_complete = ((df_cleaned.shape[0] * df_cleaned.shape[1]) - df_cleaned.isnull().sum().sum()) / (df_cleaned.shape[0]

    x = np.arange(len(['Before', 'After']))
    axes[1,1].bar(x, [before_complete, after_complete], color=['orange', 'blue'])
    axes[1,1].set_title('Data Completeness %')
    axes[1,1].set_ylabel('Percentage')
    axes[1,1].set_xticks(x)
    axes[1,1].set_xticklabels(['Before', 'After'])

    plt.tight_layout()
    plt.savefig('data_cleaning_visualization.png', dpi=300, bbox_inches='tight')
    plt.show()

    print("✓ Data quality visualizations generated and saved")

    print("\n" + "="*50)
    print("DATA CLEANING COMPLETED SUCCESSFULLY!")
    print("="*50)
    print("\nFiles generated:")
    print("1. superstore_cleaned.csv - Your cleaned dataset")
    print("2. cleaning_summary_report.txt - Detailed cleaning report")
    print("3. data_cleaning_visualization.png - Quality improvement charts")
    print("\nYou can now use the cleaned dataset for further analysis!")
```

```
==================================================
GENERATING DATA QUALITY VISUALIZATIONS
==================================================
```










```
✓ Data quality visualizations generated and saved

==================================================
DATA CLEANING COMPLETED SUCCESSFULLY!
==================================================

Files generated:
1. superstore_cleaned.csv - Your cleaned dataset
2. cleaning_summary_report.txt - Detailed cleaning report
3. data_cleaning_visualization.png - Quality improvement charts

You can now use the cleaned dataset for further analysis!
```