

# Data Cleaning & Preprocessing

## 2.1 Formatting Column Names

```
# Membersihkan nama kolom
print(f'Original column names: {df.columns.tolist()}')

def clean_column_names(df):
    """
    Membersihkan nama kolom dari spasi dan mengubah menjadi lowercase
    """
    df.columns = df.columns.str.lower().str.replace(' ', '')
    return df

df = clean_column_names(df)
print(f'Cleaned column names: {df.columns.tolist()}')
```

### Penjelasan:

- `str.lower()` : Mengubah semua nama kolom menjadi huruf kecil
- `str.replace(' ', '')` : Menghapus spasi dalam nama kolom
- **Tujuan:** Standardisasi nama kolom agar konsisten dan mudah diakses

### output

```
original column names: ['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate', 'Un:
cleaned column names: ['invoiceno', 'stockcode', 'description', 'quantity', 'invoicedate', 'unit
```

## 2.2 Check Missing Value

```
# Menghitung missing value
missing_value = df.isnull().sum()
missing_percentage = (df.isnull().sum()/len(df)) * 100

# Membuat DataFrame untuk analisis missing value
missing_df = pd.DataFrame({
    'Kolom': missing_value.index,
    'Jumlah Missing Value': missing_value.values,
    'Persentase Missing Value': (missing_value.values / len(df)) * 100
})

# Filter dan sort kolom yang memiliki missing value
missing_df = missing_df[missing_df['Jumlah Missing Value'] > 0].sort_values('Jumlah Missing Value')

print(missing_df)
```

### Penjelasan:

- `df.isnull().sum()` : Menghitung jumlah nilai null per kolom
- Membuat DataFrame terstruktur untuk memvisualisasikan missing value
- **Output:** Tabel yang menampilkan kolom dengan missing value, jumlah, dan persentasenya

### output persentase missing value

	Kolom	Jumlah Missing Value	Persentase Missing Value
6	customerid	135080	24.926694
2	description	1454	0.268311

## 2.3 Handle Missing Values

### 2.3.1 Handle Missing Value pada CustomerID

```
if 'customerid' in df.columns:
    # Mengecek pola missing value
    print(f'Jumlah missing value di kolom customerid: {df["customerid"].isnull().sum():,}')

    # Hapus baris tanpa customerid (untuk analisis pelanggan)
    df_with_customerid = df.dropna(subset=['customerid']).copy()

    # Mengisi missing value dengan 'unknown' (untuk analisis produk)
    df['customerid'] = df['customerid'].fillna('unknown')
```

#### Strategi:

- **Dual approach:** Membuat 2 versi dataset
  - `df_with_customerid` : Untuk analisis yang berfokus pada pelanggan
  - `df` : Untuk analisis produk (customerid diisi 'unknown')

## output(untuk analisis produk dan analisis pelanggan)

jumlah missing value di kolom customerid: 135,080

data kolom customerid untuk analisis produk:

```
0          17850.0
1          17850.0
2          17850.0
3          17850.0
4          17850.0
...
541904     12680.0
541905     12680.0
541906     12680.0
541907     12680.0
541908     12680.0
```

Name: customerid, Length: 541909, dtype: object

data kolom customerid untuk analisis pelanggan:

```
0          17850.0
1          17850.0
2          17850.0
3          17850.0
4          17850.0
...
541904     12680.0
541905     12680.0
541906     12680.0
541907     12680.0
541908     12680.0
```

Name: customerid, Length: 406829, dtype: float64

### 2.3.2 Handle Missing Value pada Description

```
if 'description' in df.columns:
    print(f'Jumlah missing value di kolom description: {df["description"].isnull().sum():,}')

    # Mengisi missing value dengan 'no description'
    df['description'] = df['description'].fillna('no description')

    print(f'Jumlah missing value setelah handling: {df["description"].isnull().sum():,}')
```

## Strategi:

- Mengisi dengan placeholder 'no description' karena kolom ini penting untuk analisis produk

## output kolom deskripsi

jumlah missing value di kolom description: 1,454

jumlah missing value di kolom description setelah handling: 0

## 2.3.3 Handle Missing Value pada Kolom Numerik

```
# Identifikasi kolom numerik
numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Mengisi missing value berdasarkan distribusi data
for col in numeric_columns:
    missing_count = df[col].isnull().sum()

    if missing_count > 0:
        # Menghitung skewness
        skewness = df[col].skew()

        if abs(skewness) > 0.5:
            # Distribusi tidak normal → gunakan Median
            fill_value = df[col].median()
            strategy = 'Median'
        else:
            # Distribusi normal → gunakan Mean
            fill_value = df[col].mean()
            strategy = 'Mean'

    # Mengisi missing value
    df[col] = df[col].fillna(fill_value)
    print(f"Kolom '{col}': {missing_count} missing value diisi dengan {strategy} (skewness:"))
```

## Strategi Statistik:

- **Skewness > 0.5**: Data tidak normal → gunakan **Median** (lebih robust terhadap outliers)
- **Skewness ≤ 0.5**: Data normal → gunakan **Mean**

## output kolom numerik

```
['quantity', 'unitprice']  
Kolom quantity tidak memiliki missing value  
Kolom unitprice tidak memiliki missing value
```

## 2.3.4 Handle Missing Value pada Kolom Kategorikal

```
# Identifikasi kolom kategorikal  
categorical_columns = df.select_dtypes(include=['object']).columns.tolist()  
  
# Mengisi missing value dengan modus (nilai terbanyak)  
for col in categorical_columns:  
    missing_count_categoric = df[col].isnull().sum()  
  
    if missing_count_categoric > 0:  
        mode_value = df[col].mode()[0]  
        df[col] = df[col].fillna(mode_value)  
        print(f'Kolom {col}: {missing_count_categoric} missing value diisi dengan modus {mode_value}')
```

### Strategi:

- Gunakan **Modus** (nilai yang paling sering muncul) untuk data kategorikal

## output kolom kategorik

```
['invoiceno', 'stockcode', 'description', 'invoicedate', 'customerid', 'country']  
Kolom invoiceno tidak memiliki missing value  
Kolom stockcode tidak memiliki missing value  
Kolom description tidak memiliki missing value  
Kolom invoicedate tidak memiliki missing value  
Kolom customerid tidak memiliki missing value  
Kolom country tidak memiliki missing value
```

## 2.4 Menghandle Duplicate

```
# Mengecek duplikat
duplicates = df.duplicated().sum()
print(f'Jumlah baris duplikat: {duplicates:,}')

# Melihat contoh duplikat
if duplicates:
    print('Contoh baris duplikat:')
    print(df[df.duplicated(keep=False)].head(10))

# Menghapus duplikat
df_before = len(df)
df = df.drop_duplicates()
df_after = len(df)

print(f'Jumlah data sebelum drop duplicate: {df_before}')
print(f'Jumlah data setelah drop duplicate: {df_after}')
```

### Penjelasan:

- `duplicated()` : Mendeteksi baris yang identik
- `keep=False` : Menampilkan semua duplikat (termasuk yang pertama)
- `drop_duplicates()` : Menghapus baris duplikat, hanya menyimpan yang pertama

## output contoh data yang memiliki nilai duplikat

contoh baris duplikat :

	invoiceno	stockcode	description	quantity	\
485	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	
489	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	
494	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	
517	536409	21866	UNION JACK FLAG LUGGAGE TAG	1	
521	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	
527	536409	22866	HAND WARMER SCOTTY DOG DESIGN	1	
537	536409	22900	SET 2 TEA TOWELS I LOVE LONDON	1	
539	536409	22111	SCOTTIE DOG HOT WATER BOTTLE	1	
548	536412	22327	ROUND SNACK BOXES SET OF 4 SKULLS	1	
555	536412	22327	ROUND SNACK BOXES SET OF 4 SKULLS	1	

	invoicedate	unitprice	customerid	country
485	12/1/2010 11:45	4.95	17908.0	United Kingdom
489	12/1/2010 11:45	2.10	17908.0	United Kingdom
494	12/1/2010 11:45	1.25	17908.0	United Kingdom
517	12/1/2010 11:45	1.25	17908.0	United Kingdom
521	12/1/2010 11:45	2.95	17908.0	United Kingdom
527	12/1/2010 11:45	2.10	17908.0	United Kingdom
537	12/1/2010 11:45	2.95	17908.0	United Kingdom
539	12/1/2010 11:45	4.95	17908.0	United Kingdom
548	12/1/2010 11:49	2.95	17920.0	United Kingdom
555	12/1/2010 11:49	2.95	17920.0	United Kingdom

jumlah data sebelum drop duplicate: 541909

jumlah data setelah drop duplicate: 536641



## 2.5 Cleaning Data Values (Text Columns)

```
def clean_text_columns(df):  
    """  
    Membersihkan nilai dalam kolom text:  
    - Menghapus spasi berlebih  
    - Mengubah semua huruf menjadi lowercase  
    - Strip whitespace  
    """  
  
    text_columns = df.select_dtypes(include=['object']).columns  
  
    for col in text_columns:  
        # Mengubah huruf menjadi lowercase dan menghapus spasi berlebih  
        df[col] = df[col].astype(str).str.strip().str.lower()  
        # Ganti multiple spaces dengan single space  
        df[col] = df[col].str.replace(r'\s+', ' ', regex=True)  
  
    return df  
  
df = clean_text_columns(df)  
print(df.head())
```

### Tujuan:

- **Standardisasi teks:** lowercase, hapus spasi berlebih
- **Konsistensi data:** memudahkan analisis dan filtering

## output membersihkan nilai dalam kolom

	invoiceno	stockcode	description	quantity	\
0	536365	85123a	whitehangingheartt-lightholder	6	
1	536365	71053	whitemetallantern	6	
2	536365	84406b	creamcupidheartscoathanger	8	
3	536365	84029g	knittedunionflaghotwaterbottle	6	
4	536365	84029e	redwoollyhottiewwhiteheart.	6	

	invoicedate	unitprice	customerid	country
0	12/1/20108:26	2.55	17850.0	unitedkingdom
1	12/1/20108:26	3.39	17850.0	unitedkingdom
2	12/1/20108:26	2.75	17850.0	unitedkingdom
3	12/1/20108:26	3.39	17850.0	unitedkingdom
4	12/1/20108:26	3.39	17850.0	unitedkingdom

## 2.6 Adjust Data Type

### 2.6.1 Konversi InvoiceDate ke Datetime

```
if 'invoicedate' in df.columns:
    df['invoicedate'] = pd.to_datetime(df['invoicedate'], format='%m/%d/%Y %H:%M')
    print(df['invoicedate'].dtype)
```

## output mengecek tipe data sebelum diubah

```
tipe data setiap kolom sebelum diubah :
invoiceno      object
stockcode      object
description     object
quantity       int64
invoicedate    object
unitprice      float64
customerid     object
country        object
dtype: object
```

## 2.6.3 Konversi InvoiceDate ke datetime

```
# mengubah tipe data 'invoicedate' menjadi datetime
if 'invoicedate' in df.columns:
    df['invoicedate'] = pd.to_datetime(df['invoicedate'], format = '%m/%d/%Y%H:%M')
# mengecek tipe data 'invoicedate'
print(df['invoicedate'].dtype)
```

## tipe data InvoiceDate setelah diubah

```
datetime64[ns]
```

## 2.6.3 Konversi CustomerID ke String

```
if 'customerid' in df.columns:
    df['customerid'] = df['customerid'].astype(str)
print(df['customerid'].dtype)
```

```
object
```

## 2.6.4 Memastikan Kolom Numerik Benar

```
numerical_mapping = {
    'quantity': int,
    'unitprice': float
}

for col, dtype in numerical_mapping.items():
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
        if dtype == int:
            df[col] = df[col].astype(int)
        print(f'Kolom {col} sudah diubah menjadi {dtype}')

# Memverifikasi tipe data
print("Tipe data setiap kolom:\n", df.dtypes)
```

kolom quantity sudah diubah menjadi `<class 'int'>`  
kolom unitprice sudah diubah menjadi `<class 'float'>`

### Penjelasan:

- `pd.to_numeric()` : Mengkonversi ke numerik, nilai invalid menjadi NaN
- `errors='coerce'` : Mengubah nilai yang tidak valid menjadi NaN

## 2.7 Handling Outliers

### 2.7.1 Fungsi Deteksi Outliers dengan Metode IQR

```
def detect_outliers(df, column):  
    """  
    Mendeteksi outliers menggunakan metode IQR (Interquartile Range)  
    """  
    Q1 = df[column].quantile(0.25) # Kuartil 1 (25%)  
    Q3 = df[column].quantile(0.75) # Kuartil 3 (75%)  
  
    IQR = Q3 - Q1 # Interquartile Range  
  
    lower_bound = Q1 - 1.5 * IQR  
    upper_bound = Q3 + 1.5 * IQR  
  
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]  
  
    return outliers, lower_bound, upper_bound
```

#### Metode IQR:

- **Q1**: Persentil ke-25
- **Q3**: Persentil ke-75
- **IQR**:  $Q3 - Q1$
- **Outlier**: Nilai  $< Q1 - 1.5 \times IQR$  atau  $> Q3 + 1.5 \times IQR$

## 2.7.2 Mengecek Outliers untuk Setiap Kolom Numerik

```
numerical_cols = ['quantity', 'unitprice']

for col in numerical_cols:
    if col in df.columns:
        print(f'Outliers for {col}:')

        outliers, lower_bound, upper_bound = detect_outliers(df, col)
        print(f'Jumlah outliers: {len(outliers)}')
        print(f'Persentase outliers: {len(outliers) / len(df) * 100:.2f}%')
        print(f'Lower bound: {lower_bound:.2f}')
        print(f'Upper bound: {upper_bound:.2f}')
        print(f'Min value: {df[col].min():.2f}')
        print(f'Max value: {df[col].max():.2f}')
```

```
Outliers for quantity:
jumlah outliers: 58501
persentase outliers: 10.90%
lower bound: -12.50
upper bound: 23.50
min value: -80995.00
max value: 80995.00
```

```
Outliers for invoicedate:
jumlah outliers: 0
persentase outliers: 0.00%
lower bound: .2f
upper bound: .2f
min value: .2f
max value: .2f
```

```
Outliers for unitprice:
jumlah outliers: 39450
persentase outliers: 7.35%
lower bound: -3.07
upper bound: 8.45
min value: -11062.06
max value: 38970.00
```

## 2.7.3 Visualisasi Sebelum Handling Outliers

```
# Analisis outliers untuk setiap kolom numerik
numerical_cols = ['quantity', 'unitprice']

for col in numerical_cols:
    if col in df.columns:

        outliers, lower, upper = detect_outliers_iqr(df, col)

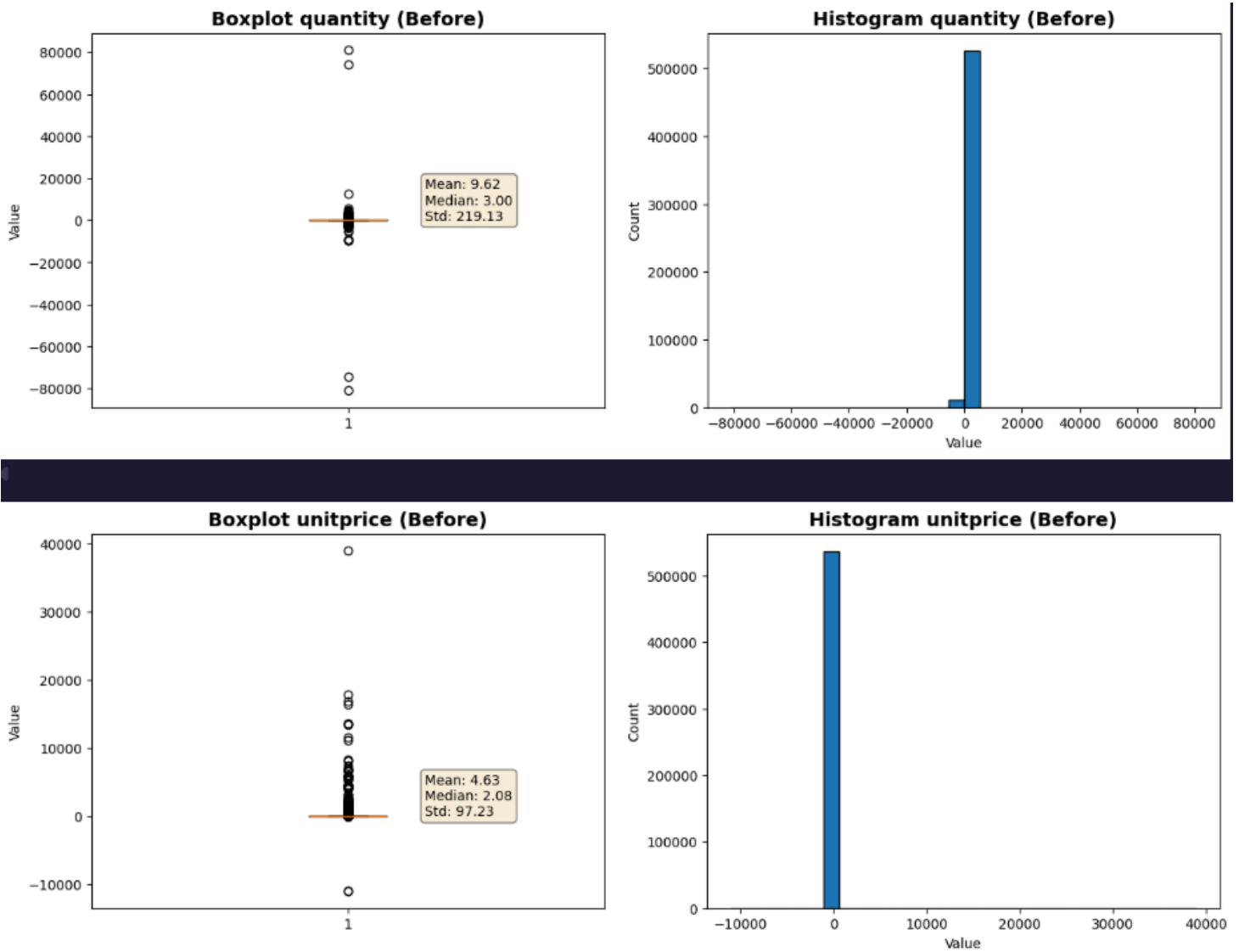
        print(f"Jumlah outliers: {len(outliers):,}")
        print(f"Persentase outliers: {(len(outliers)/len(df)*100):.2f}%")
        print(f"Lower bound: {lower:.2f}")
        print(f"Upper bound: {upper:.2f}")
        print(f"Min value: {df[col].min():.2f}")
        print(f"Max value: {df[col].max():.2f}")

# Visualisasi boxplot
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Before
axes[0].boxplot(df[col].dropna())
axes[0].set_title(f'Boxplot {col} (Before)', fontsize=14, fontweight='bold')
axes[0].set_ylabel('Value')

# histogram before
axes[1].hist(df[col].dropna(), bins=30, edgecolor='black')
axes[1].set_title(f'Histogram {col} (Before)', fontsize=14, fontweight='bold')
axes[1].set_xlabel('Value')
axes[1].set_ylabel('Count')

# Statistik
stats_text = f"Mean: {df[col].mean():.2f}\nMedian: {df[col].median():.2f}\nStd: {df[col].std():.2f}"
axes[0].text(1.15, df[col].median(), stats_text, fontsize=10, bbox=dict(boxstyle='round'))
```



## 2.7.4 Menghapus Outliers

```
for col in numerical_cols:
    if col in df.columns:
        outliers, lower, upper = detect_outliers_iqr(df, col)
        df[col] = df[col].clip(lower=lower, upper=upper)
        print(f"✓ Capping outliers untuk {col}")
```

### Strategi:

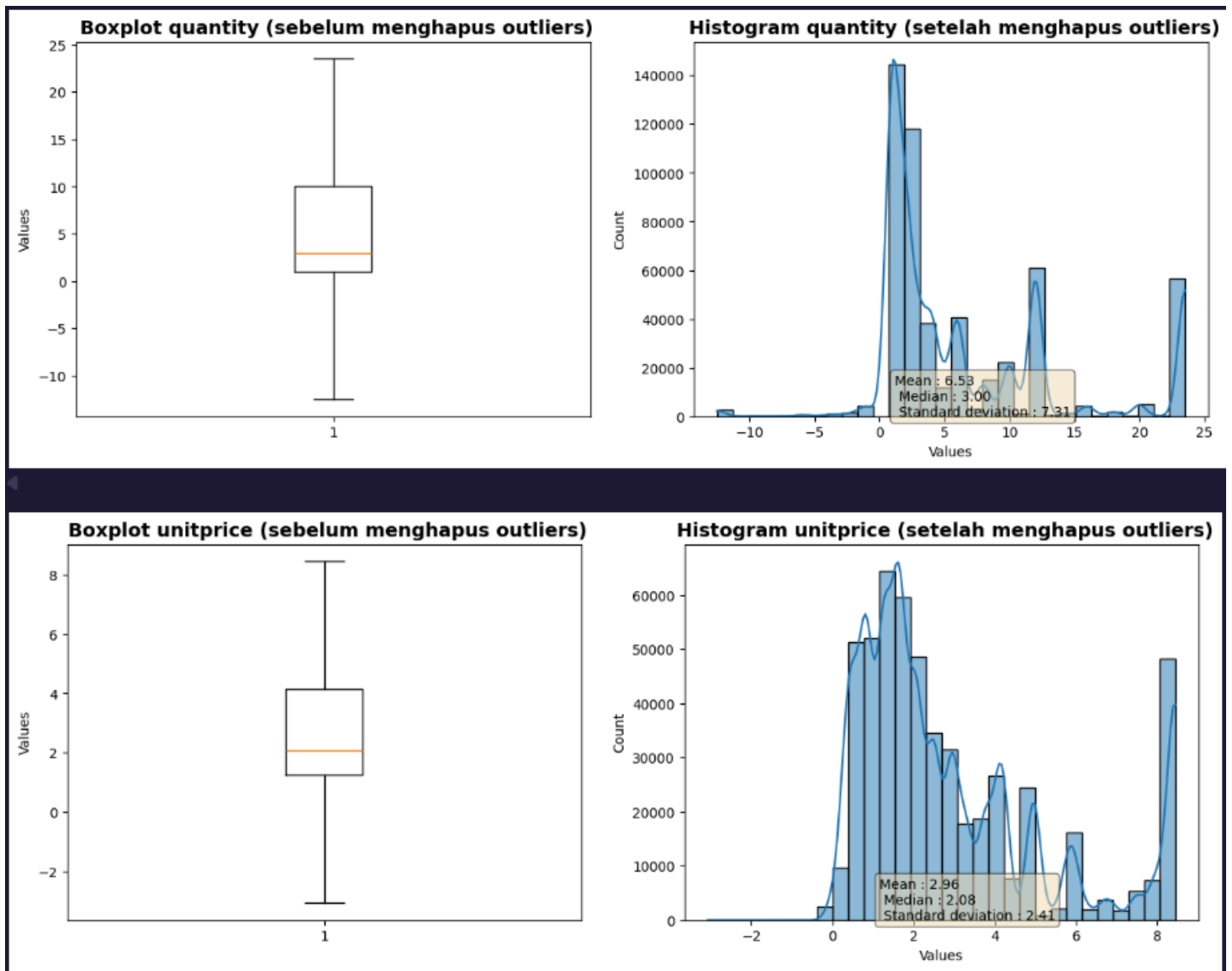
- Menghapus nilai negatif atau nol karena tidak masuk akal dalam konteks bisnis



## 2.7.5 Visualisasi Setelah Handling Outliers

```
# Boxplot setelah handling
axes[1].boxplot(df[col].dropna())
axes[1].set_title(f'Boxplot {col} (setelah menghapus outliers)', fontsize=14, fontweight='bold')
axes[1].set_ylabel('Values')

# Statistik setelah handling
stats_after = f'Mean: {df[col].mean():.2f}\nMedian: {df[col].median():.2f}\nStd Dev: {df[col].std():.2f}'
axes[1].text(1.15, df[col].median(), stats_after, fontsize=10,
            bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))
```



## 2.8 Create New Features (Feature Engineering)

### 2.8.1 Membuat Total Price

```
if 'quantity' in df.columns and 'unitprice' in df.columns:
    df['total_price'] = df['quantity'] * df['unitprice']
    print('Kolom total_price berhasil dibuat')
    print(df['total_price'].head())
```

#### Tujuan:

- Menghitung total harga per transaksi untuk analisis revenue

### 2.8.2 Extract Fitur dari InvoiceDate

```
if 'invoicedate' in df.columns:
    df['year'] = df['invoicedate'].dt.year
    df['month'] = df['invoicedate'].dt.month
    df['day'] = df['invoicedate'].dt.day
    df['dayofweek'] = df['invoicedate'].dt.dayofweek # 0=Monday, 6=Sunday
    df['hour'] = df['invoicedate'].dt.hour
    df['weekofyear'] = df['invoicedate'].dt.isocalendar().week
    df['quarter'] = df['invoicedate'].dt.quarter

    # Nama hari dan bulan
    df['dayname'] = df['invoicedate'].dt.day_name()
    df['monthname'] = df['invoicedate'].dt.month_name()
```

#### Fitur yang Dibuat:

- **Temporal features:** year, month, day, hour, quarter
- **Cyclical features:** dayofweek, weekofyear
- **Categorical:** dayname, monthname

#### Manfaat:

- Analisis tren temporal
- Identifikasi pola musiman

- Analisis peak hours/days

## 2.8.3 Kategorisasi Produk dari Deskripsi

```
if 'description' in df.columns:

    def categorize_product(description):
        desc = str(description).lower()

        if any(word in desc for word in ['bag', 'holder', 'storage']):
            return 'bag&storage'
        elif any(word in desc for word in ['light', 'lamp', 'candle']):
            return 'light'
        elif any(word in desc for word in ['paper', 'card', 'notebook']):
            return 'stationary'
        elif any(word in desc for word in ['kitchen', 'cook', 'food']):
            return 'kitchenware'
        elif any(word in desc for word in ['decoration', 'decor', 'ornament']):
            return 'decoration'
        elif any(word in desc for word in ['gift', 'present']):
            return 'gift'
        else:
            return 'others'

    df['category'] = df['description'].apply(categorize_product)
    print(df.columns.tolist())
```

### Tujuan:

- Membuat kategori produk dari deskripsi text
- Memudahkan analisis per kategori produk
- Menggunakan **keyword matching** untuk klasifikasi

## 2.9 Data Quality Report

```
print('=== Data Quality Report After Cleaning ===\n')

# Dimensi data
print(f'Jumlah baris: {df.shape[0]:,}')
print(f'Jumlah kolom: {df.shape[1]}\n')

# Missing values
print('Missing Values:')
print(f'Jumlah missing value: {df.isnull().sum().sum()}\n')

# Duplikat
print('Duplikat:')
print(f'Jumlah duplikat: {df.duplicated().sum()}\n')

# Tipe data
print('Tipe Data:')
for dtype in df.dtypes.unique():
    cols = df.select_dtypes(include=[dtype]).columns.tolist()
    print(f'{dtype}: {len(cols)} kolom')
```

### Output:

- Ringkasan kualitas data setelah cleaning
- Verifikasi bahwa tidak ada missing value atau duplikat
- Distribusi tipe data

## 2.10 Menyimpan Data Cleaned

```
# Menyimpan data yang sudah dibersihkan
df.to_csv('../data/cleaned/ecommerce_cleaned.csv', index=False)
```

### Penjelasan:

- `index=False` : Tidak menyimpan index DataFrame sebagai kolom
- Data siap untuk tahap analisis selanjutnya

