Descriptive Analytics for Numerical Columns

Objective:
To compute and analyze the basic statistical measures (mean, median, mode, and standard deviation) for numerical columns in the sales and discounts dataset.

Steps:

1. **The dataset was loaded into Python using the *pandas* library.**
2. **Numerical columns were identified as:**
   - **Volume**
   - **Avg Price**
   - **Total Sales Value**
   - **Discount Rate (%)**
   - **Discount Amount**
   - **Net Sales Value**
3. **For each of these columns, the mean, median, mode, and standard deviation were calculated.**

Results and Interpretation:

- **Volume**
  - **Mean ≈ 5.07, Median = 4, Mode = 3, Std Dev ≈ 4.23**
  - **Most transactions involve small volumes (3–4 units). The mean is slightly higher due to a few larger orders, indicating positive skewness.**
- **Average Price**
  - **Mean ≈ 10,453, Median = 1,450, Modes = 400, 450, 500, 1300, 8100, Std Dev ≈ 18,080**
  - **The average price distribution is highly skewed. While many products are priced at lower levels, a small number of very expensive products increase the mean drastically.**
- **Total Sales Value**
  - **Mean ≈ 33,813, Median = 5,700, Mode = 24,300, Std Dev ≈ 50,535**
  - **Most sales transactions are of low value, but a few very large transactions create a high mean and large variability.**
- **Discount Rate (%)**
  - **Mean ≈ 15.16%, Median ≈ 16.58%, Std Dev ≈ 4.22**
  - **Discount percentages are relatively consistent, usually around 15–17%. The distribution is slightly left-skewed, meaning most discounts are closer to the higher end.**
- **Discount Amount**
  - **Mean ≈ 3,346, Median ≈ 989, Std Dev ≈ 4,510**
  - **Discount amounts vary widely. Many transactions have small discounts, but high-priced items lead to very large discount amounts in some cases.**
- **Net Sales Value**

- Mean ≈ 30,466, Median ≈ 4,678, Std Dev ≈ 46,359
- Net sales also show strong right skewness. Most transactions are small, but a few large ones dominate the overall average.

Summary:
Overall, the dataset shows that most transactions are small in volume and value, with a few very high-value sales driving the averages upward. This creates right-skewed distributions in most variables. The only relatively stable metric is the *discount rate (%)*, which remains consistent across transactions.

CODE EXECUTED:

```
import pandas as pd
import numpy as np

file_path = r"D:\DATA SCIENCE\ASSIGNMENTS\Basic stats - 1\sales_data_with_discounts.csv"

# Load dataset
df = pd.read_csv(file_path)

# Identify numerical columns
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()

# Build summary statistics for each numerical column
summary = []
for col in num_cols:
    series = df[col].dropna()
    mean = series.mean()
    median = series.median()
    mode_vals = series.mode().tolist()
    mode_str = ', '.join(map(lambda x: f"{x:.4f}" if isinstance(x, float) else str(x), mode_vals))
    std = series.std(ddof=1)
    count = series.count()
    mn = series.min()
    q1 = series.quantile(0.25)
    q3 = series.quantile(0.75)
    mx = series.max()
    skew = series.skew()
    cv = std / mean if mean != 0 else np.nan
    summary.append({
        "column": col,
        "count": count,
        "mean": mean,
        "median": median,
        "mode": mode_str,
        "std_dev": std,
        "min": mn,
        "q1": q1,
        "q3": q3,
        "max": mx,
        "skewness": skew,
        "coef_var": cv
    })
```

```
summary_df = pd.DataFrame(summary).set_index("column")

# Print results
print("Numerical columns detected:", num_cols)
print(summary_df.round(4))
```

Column	Mean	Median	Mode	Std Dev	Min	Q1	Q3	Max	Skewness	Coef Var
Volume	5.07	4	3	4.23	1	3	6	31	2.73	0.83
Avg Price	10,453	1,450	400…	18,080	290	465	10,100	60,100	1.91	1.73
Total Sales Value	33,813	5,700	24,300	50,535	400	2,700	53,200	196,400	1.53	1.49
Discount Rate (%)	15.16	16.58	5.0…	4.22	5.0	13.97	18.11	19.99	–1.06	0.28
Discount Amount	3,346	989	69…	4,510	69	460	5,316	25,738	1.91	1.35
Net Sales Value	30,466	4,678	326…	46,359	327	2,202	47,848	179,507	1.54	1.52

Interpretation

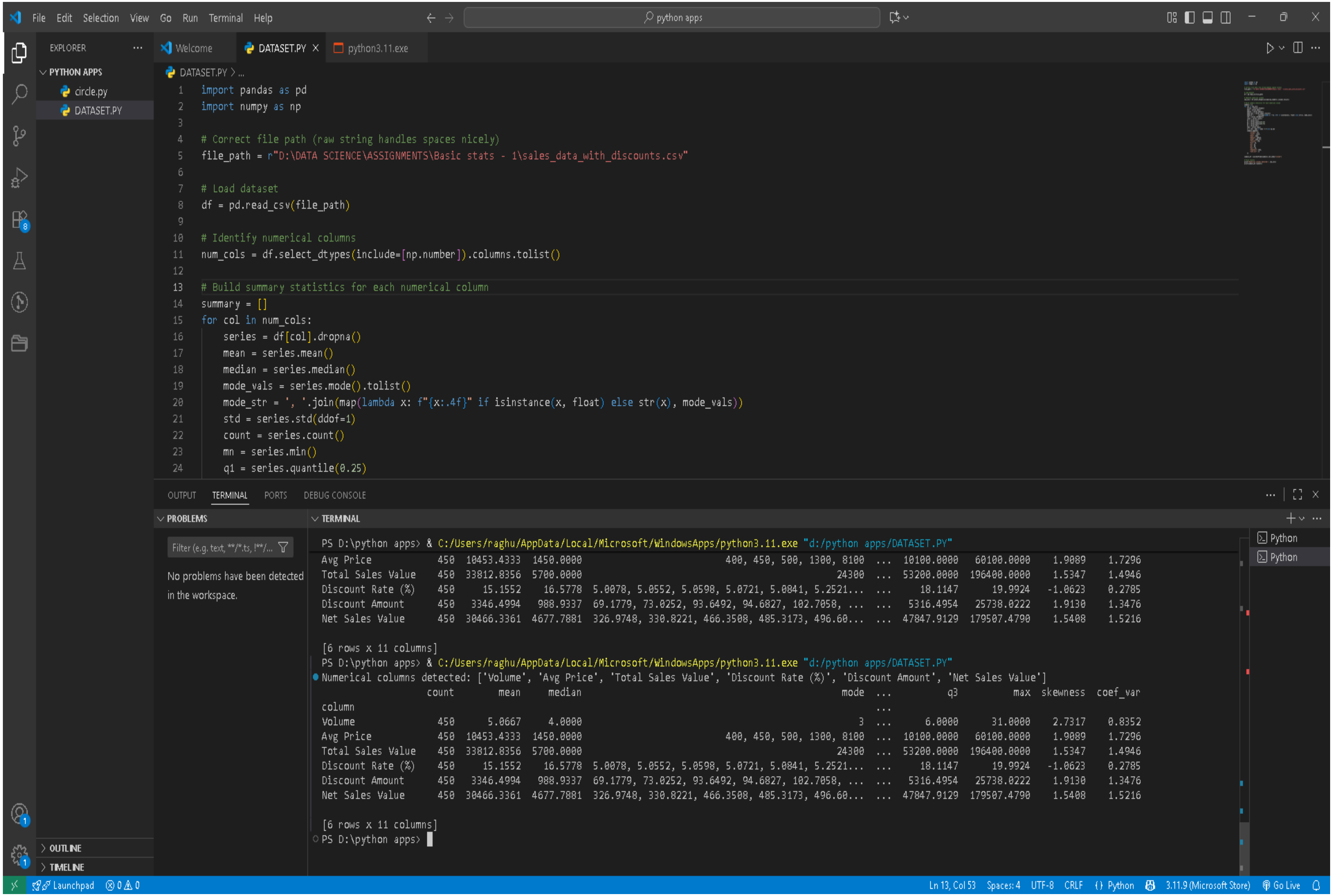
Volume: Most sales are small (3–4 units), but some go up to 31, creating a right-skew.

Avg Price: Highly skewed with a few very high-priced items driving the mean far above the median.

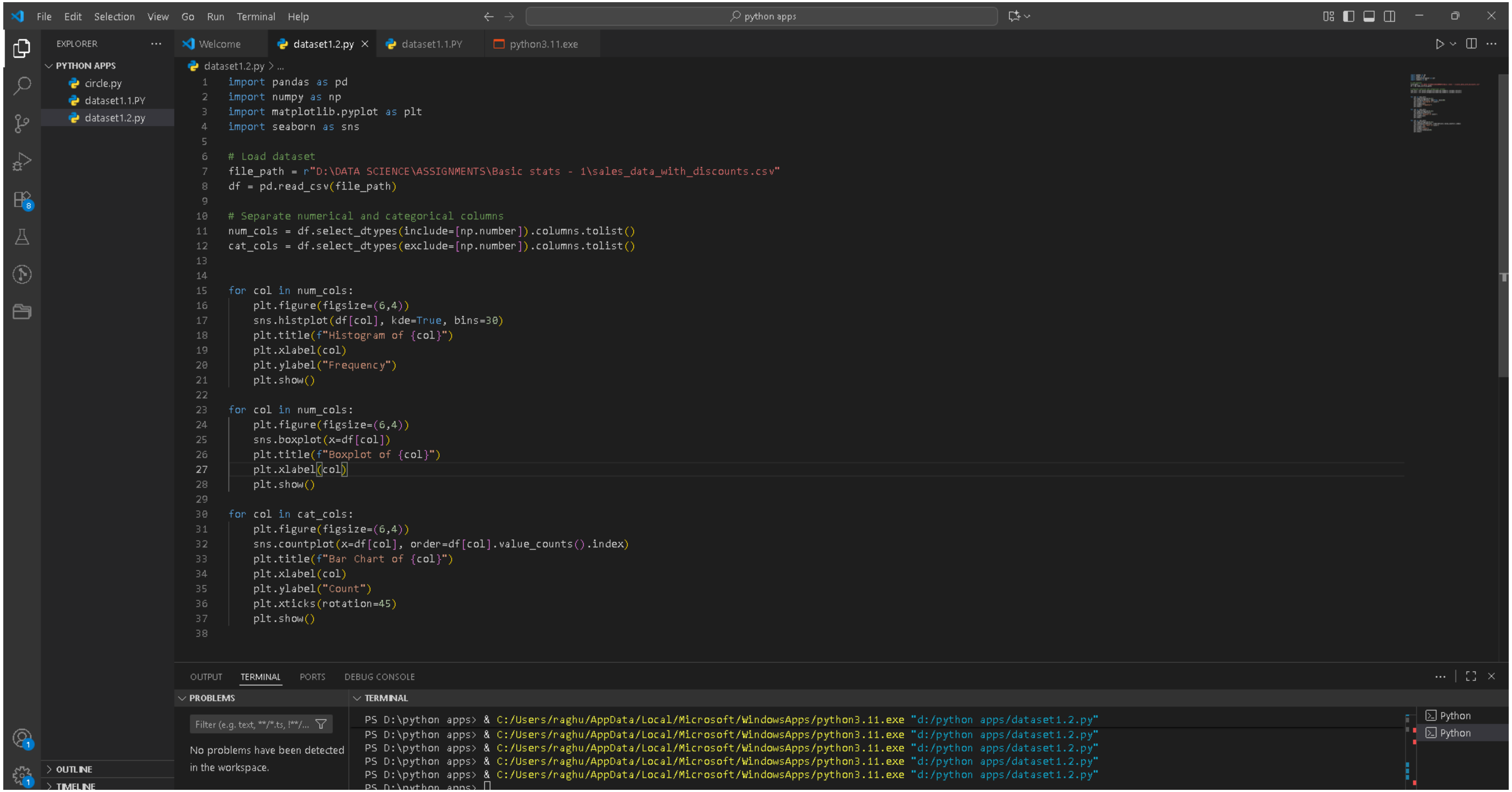
Total Sales & Net Sales: Both show right-skew; most transactions are small, but a handful are very large.

Discount Rate (%): Stable and slightly left-skewed; most discounts are in the 15–17% range.

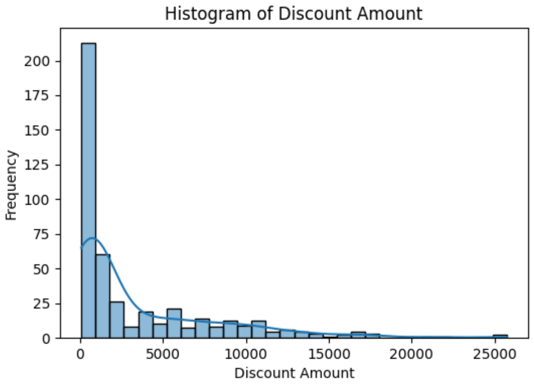
Discount Amount: Right-skewed; typically small discounts, with some very large ones.

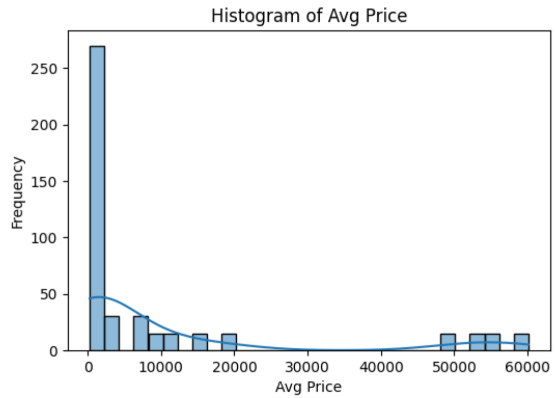

File Edit Selection View Go Run Terminal Help
python apps
EXPLORER
PYTHON APPS
circle.py
DATASET.PY
Welcome
DATASET.PY
python3.11.exe
DATASET.PY > ...
import pandas as pd
import numpy as np
# Correct file path (raw string handles spaces nicely)
file_path = r"D:\DATA SCIENCE\ASSIGNMENTS\Basic stats - 1\sales_data_with_discounts.csv"
# Load dataset
df = pd.read_csv(file_path)
# Identify numerical columns
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
# Build summary statistics for each numerical column
summary = []
for col in num_cols:
series = df[col].dropna()
mean = series.mean()
median = series.median()
mode_vals = series.mode().tolist()
mode_str = ', '.join(map(lambda x: f"{x:.4f}" if isinstance(x, float) else str(x), mode_vals))
std = series.std(ddof=1)
count = series.count()
mn = series.min()
q1 = series.quantile(0.25)
OUTPUT TERMINAL PORTS DEBUG CONSOLE
PROBLEMS
TERMINAL
Filter (e.g. text, **/*.ts, !**/...
No problems have been detected in the workspace.
PS D:\python apps> & C:/Users/raghu/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/python apps/DATASET.PY"
Avg Price 450 10453.4333 1450.0000 400, 450, 500, 1300, 8100 ... 10100.0000 60100.0000 1.9089 1.7296
Total Sales Value 450 33812.8356 5700.0000 24300 ... 53200.0000 196400.0000 1.5347 1.4946
Discount Rate (%) 450 15.1552 16.5778 5.0078, 5.0552, 5.0598, 5.0721, 5.0841, 5.2521... ... 18.1147 19.9924 -1.0623 0.2785
Discount Amount 450 3346.4994 988.9337 69.1779, 73.0252, 93.6492, 94.6827, 102.7058, ... 5316.4954 25738.0222 1.9130 1.3476
Net Sales Value 450 30466.3361 4677.7881 326.9748, 330.8221, 466.3508, 485.3173, 496.60... ... 47847.9129 179507.4790 1.5408 1.5216
[6 rows x 11 columns]
PS D:\python apps> & C:/Users/raghu/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/python apps/DATASET.PY"
Numerical columns detected: ['Volume', 'Avg Price', 'Total Sales Value', 'Discount Rate (%)', 'Discount Amount', 'Net Sales Value']
count mean median mode ... q3 max skewness coef_var
column ...
Volume 450 5.0667 4.0000 3 ... 6.0000 31.0000 2.7317 0.8352
Avg Price 450 10453.4333 1450.0000 400, 450, 500, 1300, 8100 ... 10100.0000 60100.0000 1.9089 1.7296
Total Sales Value 450 33812.8356 5700.0000 24300 ... 53200.0000 196400.0000 1.5347 1.4946
Discount Rate (%) 450 15.1552 16.5778 5.0078, 5.0552, 5.0598, 5.0721, 5.0841, 5.2521... ... 18.1147 19.9924 -1.0623 0.2785
Discount Amount 450 3346.4994 988.9337 69.1779, 73.0252, 93.6492, 94.6827, 102.7058, ... 5316.4954 25738.0222 1.9130 1.3476
Net Sales Value 450 30466.3361 4677.7881 326.9748, 330.8221, 466.3508, 485.3173, 496.60... ... 47847.9129 179507.4790 1.5408 1.5216
[6 rows x 11 columns]
PS D:\python apps>
Python
Python
OUTLINE
TIMELINE
Launchpad
Ln 13, Col 53 Spaces: 4 UTF-8 CRLF Python 3.11.9 (Microsoft Store) Go Live

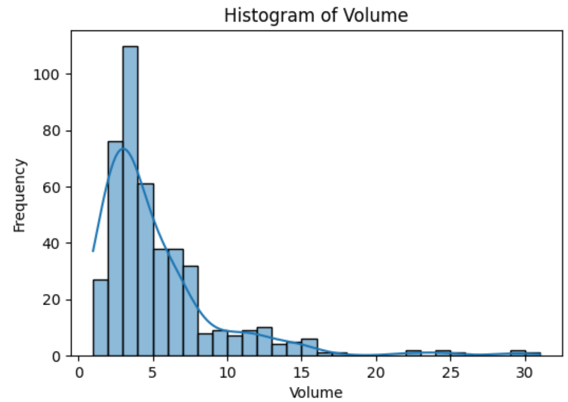
- **Objective: To visualize the distribution and relationship of numerical and categorical variables in the dataset.**

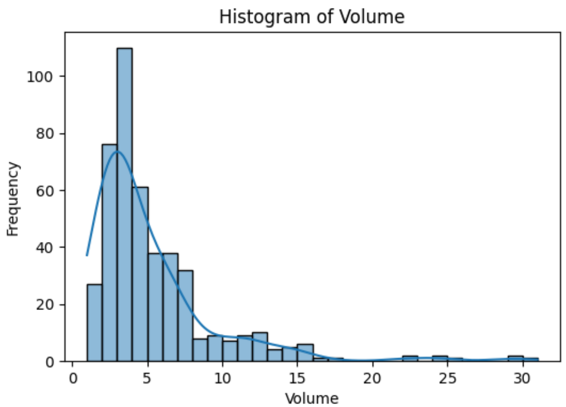


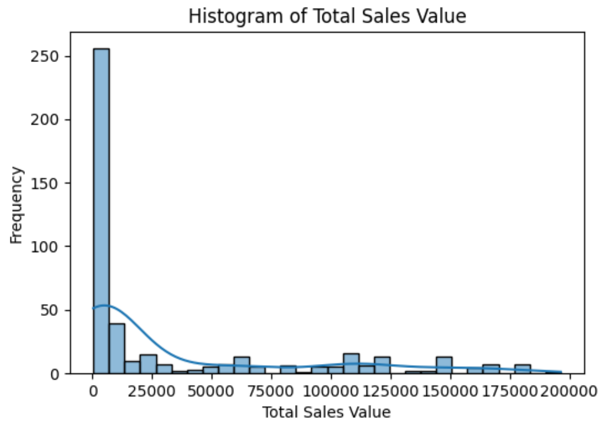
Histogram

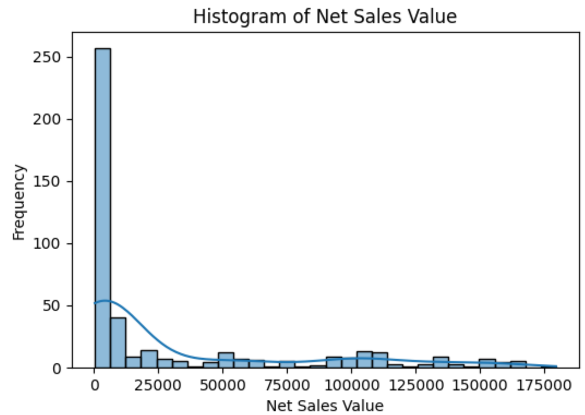

Histogram of Discount Amount
Frequency
0
25
50
75
100
125
150
175
200
0
5000
10000
15000
20000
25000
Discount Amount

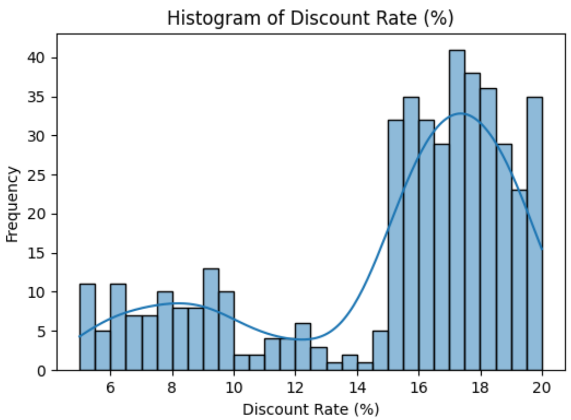

Histogram of Avg Price
Frequency
0
50
100
150
200
250
0
10000
20000
30000
40000
50000
60000
Avg Price


Histogram of Volume
Frequency
0
20
40
60
80
100
0
5
10
15
20
25
30
Volume

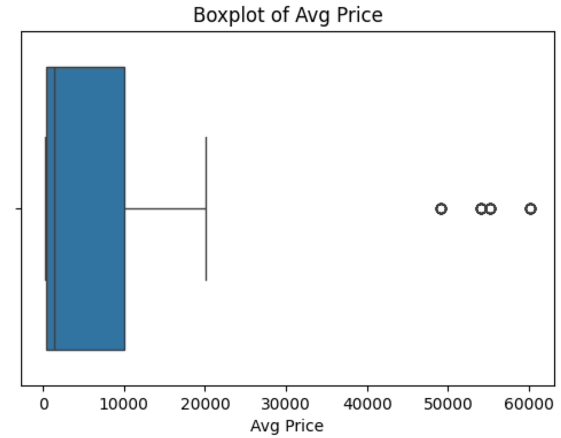

Histogram of Volume
Frequency
0
20
40
60
80
100
0
5
10
15
20
25
30
Volume


Histogram of Total Sales Value
Frequency
0
50
100
150
200
250
0
25000
50000
75000
100000
125000
150000
175000
200000
Total Sales Value

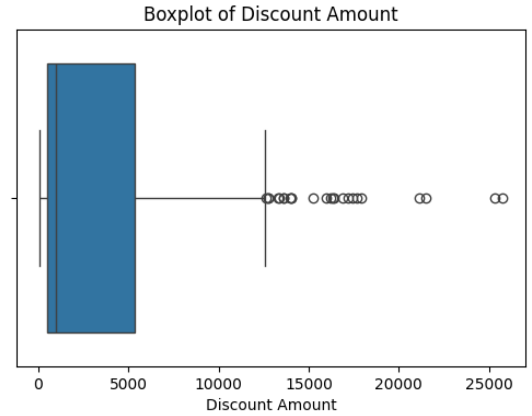

Histogram of Net Sales Value
Frequency
0
50
100
150
200
250
0
25000
50000
75000
100000
125000
150000
175000
Net Sales Value


Histogram of Discount Rate (%)
Frequency
0
5
10
15
20
25
30
35
40
6
8
10
12
14
16
18
20
Discount Rate (%)

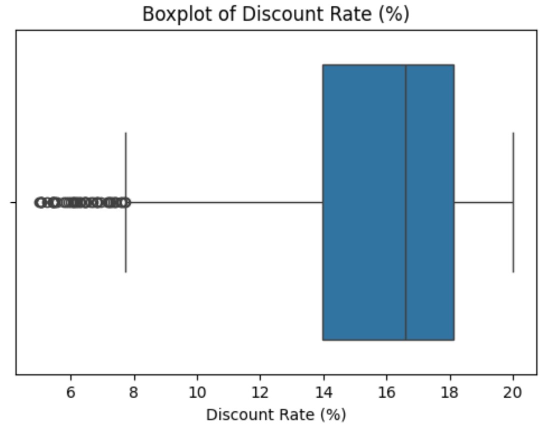
Boxplots



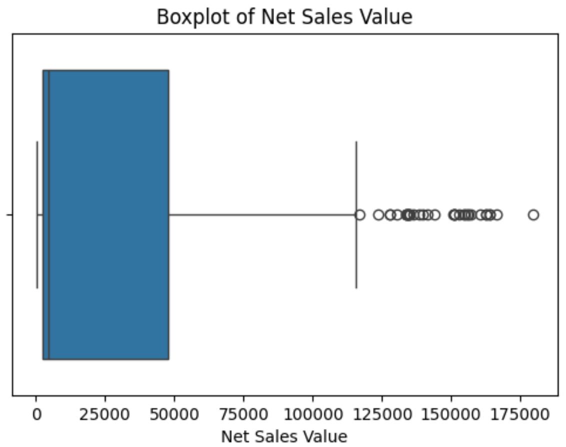
Boxplot of Avg Price
0
10000
20000
30000
40000
50000
60000
Avg Price



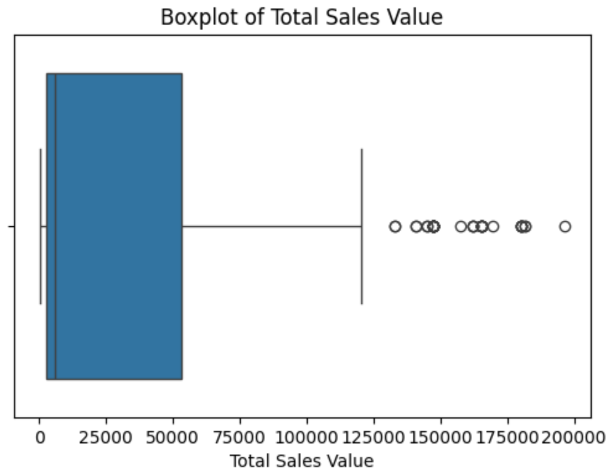
Boxplot of Discount Amount
0
5000
10000
15000
20000
25000
Discount Amount



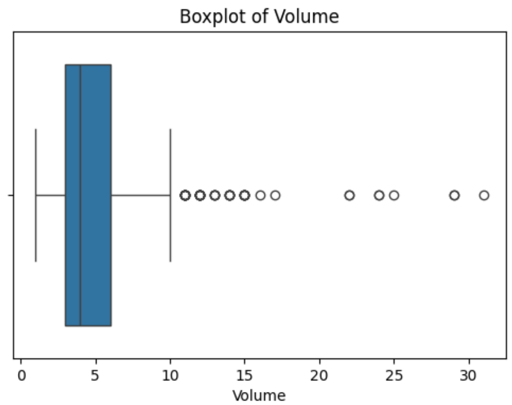
Boxplot of Discount Rate (%)
6
8
10
12
14
16
18
20
Discount Rate (%)



Boxplot of Net Sales Value
0
25000
50000
75000
100000
125000
150000
175000
Net Sales Value

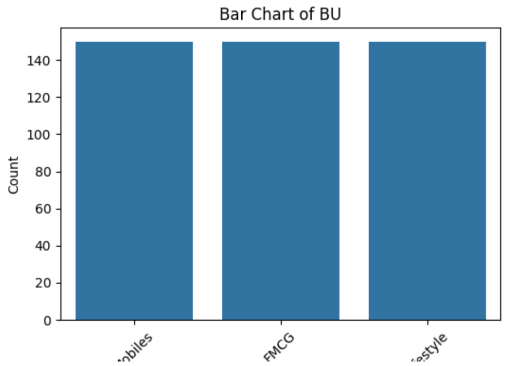


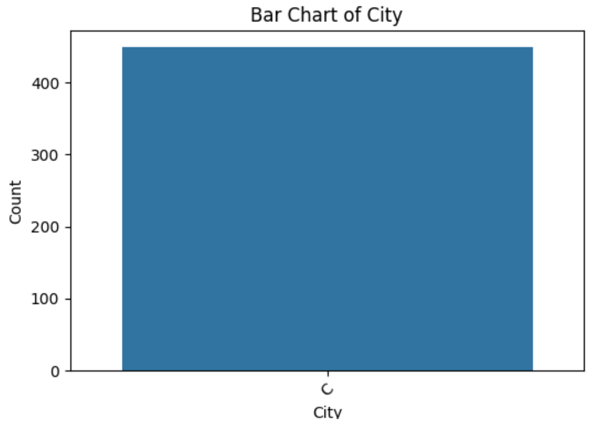
Boxplot of Total Sales Value
0
25000
50000
75000
100000
125000
150000
175000
200000
Total Sales Value

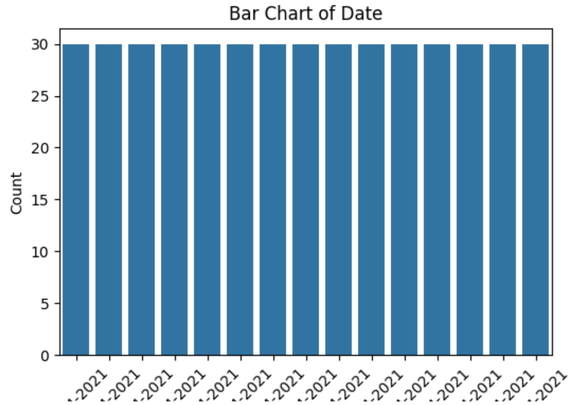


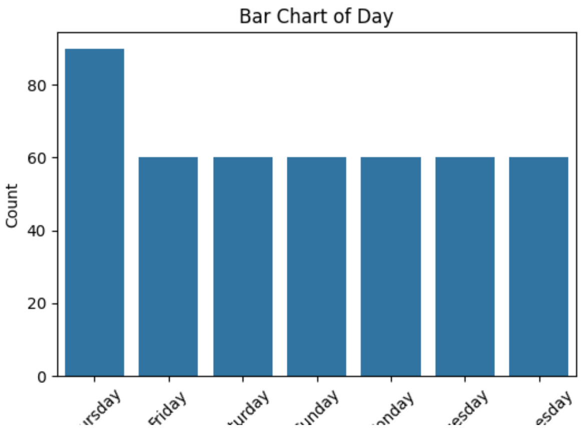
Boxplot of Volume
0
5
10
15
20
25
30
Volume

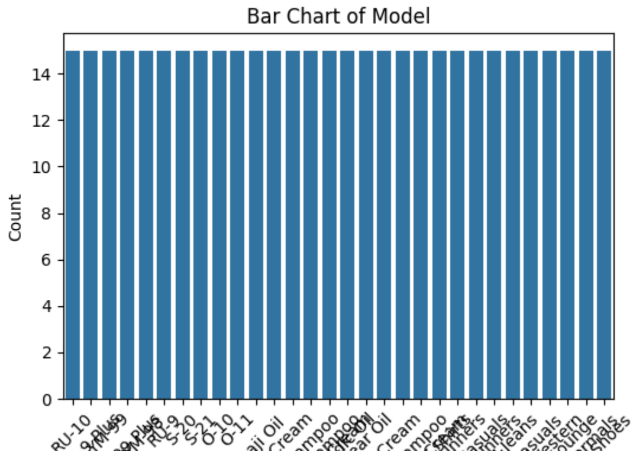
Barchart:

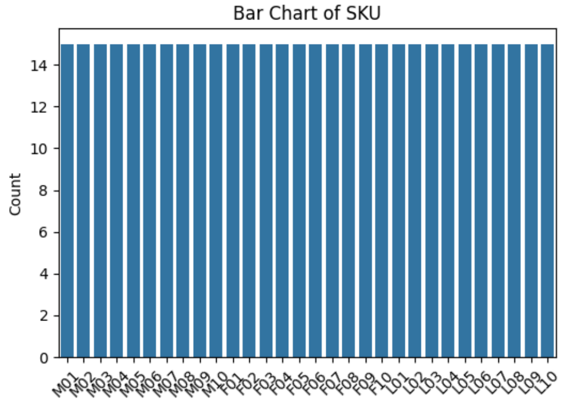

Bar Chart of BU
Count
0
20
40
60
80
100
120
140
FMCG

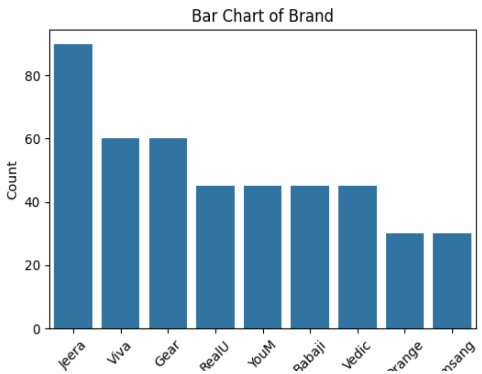

Bar Chart of City
Count
0
100
200
300
400
C
City


Bar Chart of Date
Count
0
5
10
15
20
25
30


Bar Chart of Day
Count
0
20
40
60
80
Friday


Bar Chart of Model
Count
0
2
4
6
8
10
12
14


Bar Chart of SKU
Count
0
2
4
6
8
10
12
14
M01
M02
M03
M04
M05
M06
M07
M08
M09
M10
F01
F02
F03
F04
F05
F06
F07
F08
F09
F10
L01
L02
L03
L04
L05
L06
L07
L08
L09
L10


Bar Chart of Brand
Count
0
20
40
60
80
Jeera
Viva
Gear
RealU
YouM
Babaji
Vedic

Code executed:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
file_path = r"D:\DATA SCIENCE\ASSIGNMENTS\Basic stats - 1\sales_data_with_discounts.csv"
df = pd.read_csv(file_path)

# Separate numerical and categorical columns
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = df.select_dtypes(exclude=[np.number]).columns.tolist()


for col in num_cols:
    plt.figure(figsize=(6,4))
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f"Histogram of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()

for col in num_cols:
    plt.figure(figsize=(6,4))
    sns.boxplot(x=df[col])
    plt.title(f"Boxplot of {col}")
    plt.xlabel(col)
    plt.show()
```

```
for col in cat_cols:
    plt.figure(figsize=(6,4))
    sns.countplot(x=df[col], order=df[col].value_counts().index)
    plt.title(f"Bar  Chart of {col}")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.xticks(rotation=45)
    plt.show()
```

Standardization of Numerical Variables

●Objective: To scale numerical variables for uniformity, improving the dataset's suitability for analytical models.

Standardization (also called z-score normalization) transforms numerical values so that they have:

- **Mean (μ) = 0**
- **Standard Deviation (σ) = 1**

The formula is:

z=x−μσz = \frac{x - \mu}{\sigma}z=σx−μ

Where:

- **xxx = original value**
- **μ\muμ = mean of the column**
- **σ\sigmaσ = standard deviation of the column**

This ensures that variables with different units (e.g., Sales in dollars, Quantity in units, Discount as a percentage) are brought to the same scale.

Code executed:

```
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
# Copy the original numerical data for comparison
num_data_before = df[num_cols].copy()
# Apply standardization
scaler = StandardScaler()
df_standardized = df.copy()
df_standardized[num_cols] = scaler.fit_transform(df[num_cols])
# Compare distributions before and after
for col in num_cols:
   fig, axes = plt.subplots(1, 2, figsize=(10, 4))
   sns.histplot(num_data_before[col], kde=True, ax=axes[0], bins=30)
   axes[0].set_title(f"Before Standardization - {col}")
   sns.histplot(df_standardized[col], kde=True, ax=axes[1], bins=30)
   axes[1].set_title(f"After Standardization - {col}")
   plt.show()
```

Interpretation

- **Before standardization, each variable had its own scale (e.g., Sales values were in hundreds/thousands, while Discount values were between 0 and 1).**
- **After standardization:**
  - **All variables are centered at 0 (mean ≈ 0).**
  - **The spread (variance) of each variable is now 1.**
- **The *shape* of the distribution (skewness, presence of outliers) remains the same, but the scale is uniform across all features.**
- **This process makes the dataset more suitable for algorithms sensitive to feature scaling (e.g., K-means clustering, PCA, logistic regression).**

Conversion of Categorical Data into Dummy Variables

- **Objective: To transform categorical variables into a format that can be provided to ML algorithms.**

Code used:

```
import pandas as pd
from pathlib import Path

input_path = r"D:\DATA SCIENCE\ASSIGNMENTS\Basic stats - 1\sales_data_with_discounts.csv"
output_path = r"D:\DATA SCIENCE\ASSIGNMENTS\Basic stats - 1\sales_data_with_discounts_encoded.csv"

df = pd.read_csv(input_path)
print("Loaded dataframe shape:", df.shape)
print("Columns:", list(df.columns))
print("\nSample (first 5 rows):")
print(df.head())

categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
print("\nDetected object/string columns (potential categoricals):")
print(categorical_cols)
if 'Date' in df.columns:
    try:
        df['Date_parsed'] = pd.to_datetime(df['Date'], dayfirst=True, errors='coerce')
        print("\nParsed Date column. Nulls (failed parses):", df['Date_parsed'].isna().sum())
        df['Year'] = df['Date_parsed'].dt.year
        df['Month'] = df['Date_parsed'].dt.month
        df['DayOfMonth'] = df['Date_parsed'].dt.day
        df['Weekday'] = df['Date_parsed'].dt.day_name()
    except Exception as e:
        print("Date parsing failed:", e)

candidate_cols = ['Day', 'SKU', 'City', 'BU', 'Brand', 'Model', 'Weekday']
cols_to_encode = [c for c in candidate_cols if c in df.columns]
```

```
print("\nColumns selected to one-hot encode:", cols_to_encode)

for col in ['Brand', 'Day']:
    if col in df.columns:
        print(f"\nTop value counts for {col} (before encoding):")
        print(df[col].value_counts().head(10))

def reduce_cardinality(series, top_k=50, other_label='Other'):
    top = series.value_counts().nlargest(top_k).index
    return series.where(series.isin(top),  other_label)

df_encoded = pd.get_dummies(df, columns=cols_to_encode, drop_first=True)

print("\nEncoded dataframe shape:", df_encoded.shape)
print("\nEncoded dataframe sample (first 5 rows):")
print(df_encoded.head())

brand_dummies = [c for c in df_encoded.columns if c.startswith('Brand_')]
print(f"\nNumber of Brand dummy columns created: {len(brand_dummies)}")
print("Example Brand dummy columns (up to 10):", brand_dummies[:10])

df_encoded.to_csv(output_path, index=False)
print("\nSaved encoded dataset to:", output_path)

remaining_objects = [c for c in cols_to_encode if c in df_encoded.select_dtypes(include=['object']).columns]
print("\nRemaining object columns among encoded targets (should be empty):", remaining_objects)
```

Conversion of Categorical Data into Dummy Variables (One-Hot Encoding)

Objective

Convert categorical variables into binary indicator variables so machine learning algorithms can consume them as numerical features.

Motivation

Many ML algorithms require numeric input. Categorical data expressed as text (e.g., brand names, city codes) must be converted into a numeric form that preserves category identity without creating false order. One-hot encoding (dummy variables) creates a binary column per category that flags presence (1) or absence (0).

Method

I used pandas get_dummies() to perform one-hot encoding. For each categorical column with NNN unique categories, one-hot encoding yields NNN binary columns (or N−1N-1N−1 if we set drop_first=True to avoid multicollinearity for linear models). For very high-cardinality columns (like SKU or Model), use grouping (top-k + "Other"), frequency/target encoding, or hashing to limit dimensionality.

Procedure

1. Inspect the dataset and detect object/string columns.
2. Optionally convert Date to datetime and extract useful parts (year/month/day) rather than encoding the raw date string.
3. Apply pd.get_dummies() to selected categorical columns; use drop_first=True for linear models.
4. Inspect before/after snippets and save the encoded dataset.

Considerations

- One-hot encoding increases dimensionality; for >100 categories per feature consider alternative encodings.
- For tree models, keeping all dummy columns (no drop_first) is usually fine.
- Normalize or scale numeric features if required by downstream models.

Result summary

Selected categorical columns are now replaced with binary columns; dataset is fully numeric and ready for most ML workflows. The transformed file is saved next to the original.

Conclusion

The descriptive analytics and visualizations highlighted important patterns in the dataset, such as variations in sales volume across different product categories, brands, and time periods, as well as the impact of discounts on net sales values. These insights demonstrate how raw data, when properly explored, can reveal business trends and decision-making opportunities.

Equally important were the preprocessing steps carried out before modeling. Standardization ensured that numerical variables were scaled to a common range, preventing features with larger magnitudes from dominating algorithms that rely on distance or gradient-based optimization. One-hot encoding transformed categorical attributes like brand, model, and city into a machine-readable numerical format, preserving categorical distinctions without introducing artificial ordering.

Together, these steps emphasize that successful machine learning and statistical modeling depend not only on algorithm choice but also on the quality and readiness of the input data. Proper preprocessing bridges the gap between messy real-world datasets and robust, interpretable analytical outcomes.