In [50]:
```
import sqlite3
import pandas as pd

class DatabaseAnalyzer:
    def __init__(self, db_path='purchases.db'):
        self.conn = sqlite3.connect(db_path)
        self.create_tables_if_not_exist()

    def create_tables_if_not_exist(self):
        """Create all necessary tables if they don't exist"""
        cursor = self.conn.cursor()

        # Create purchases table
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS purchases(
            purchaseID INTEGER PRIMARY KEY AUTOINCREMENT,
            vendorNumber INTEGER,
            productName TEXT,
            quantity INTEGER,
            price REAL
        )
        """)

        # Create purchase_prices table
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS purchase_prices(
            purchaseID INTEGER PRIMARY KEY AUTOINCREMENT,
            VendorNumber INTEGER,
            productName TEXT,
            quantity INTEGER,
            price REAL
        )
        """)

        # Create vendor_invoice table
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS vendor_invoice(
            invoiceID INTEGER PRIMARY KEY AUTOINCREMENT,
            VendorNumber INTEGER,
            Freight REAL
        )
        """)

        # Create sales table
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS sales(
            salesID INTEGER PRIMARY KEY AUTOINCREMENT,
            VendorNo INTEGER,
            Brand TEXT,
            SalesQuantity INTEGER,
            SalesDollars REAL,
            SalesPrice REAL,
            ExciseTax REAL
        )
        """)

        self.conn.commit()
```

```
    def insert_sample_data(self):
        """Insert data into all tables"""
        cursor = self.conn.cursor()

        # data for purchases
        cursor.executemany("""
        INSERT INTO purchases (vendorNumber, productName, quantity, price)
        VALUES (?, ?, ?, ?)
        """, [
            (456, 'Product A', 10, 20.0),
            (456, 'Product B', 5, 40.0),
            (789, 'Product C', 20, 15.0),
            (4466, 'Product D', 8, 25.5)
        ])

        # data for purchase_prices
        cursor.executemany("""
        INSERT INTO purchase_prices (VendorNumber, productName, quantity, price)
        VALUES (?, ?, ?, ?)
        """, [
            (4466, 'Product X', 15, 25.0),
            (4466, 'Product Y', 8, 35.5),
            (789, 'Product Z', 12, 18.0)
        ])

        #  data for vendor_invoice
        cursor.executemany("""
        INSERT INTO vendor_invoice (VendorNumber, Freight)
        VALUES (?, ?)
        """, [
            (4466, 150.0),
            (456, 200.0),
            (789, 100.0)
        ])

        #  data for sales
        cursor.executemany("""
        INSERT INTO sales (VendorNo, Brand, SalesQuantity, SalesDollars, SalesPr
        VALUES (?, ?, ?, ?, ?, ?)
        """, [
            (4466, 'Brand A', 50, 1000.0, 20.0, 50.0),
            (456, 'Brand B', 30, 1500.0, 50.0, 75.0),
            (789, 'Brand C', 40, 800.0, 20.0, 40.0)
        ])

        self.conn.commit()
        print("Sample data inserted into all tables")

    def run_queries(self):
        """Run all your analysis queries"""
        try:
            # Purchase prices for vendor 4466
            print("1. Purchase prices for vendor 4466:")
            purchase_prices = pd.read_sql_query("SELECT * FROM purchase_prices W
            print(purchase_prices)
            print()

            # Vendor invoice for vendor 4466
            print("2. Vendor invoice for vendor 4466:")
            vendor_invoice = pd.read_sql_query("SELECT * FROM vendor_invoice WHE
```

```
            print(vendor_invoice)
            print()

            # Sales for vendor 4466
            print("3. Sales for vendor 4466:")
            sales = pd.read_sql_query("SELECT * FROM sales WHERE VendorNo = 4466
            print(sales)
            print()

            # Group purchases by brand and purchase price
            print("4. Purchases grouped by productName:")
            purchases_grouped = pd.read_sql_query("""
            SELECT productName, price, SUM(quantity) as TotalQuantity, SUM(quant
            FROM purchases
            GROUP BY productName, price
            """, self.conn)
            print(purchases_grouped)
            print()

            # Freight summary
            print("5. Freight summary by vendor:")
            freight_summary = pd.read_sql_query("""
            SELECT VendorNumber, SUM(Freight) as FreightCost
            FROM vendor_invoice
            GROUP BY VendorNumber
            """, self.conn)
            print(freight_summary)

        except Exception as e:
            print(f"Error running queries: {e}")

    def close(self):
        """Close database connection"""
        self.conn.close()
        print("Database connection closed")

# Usage
if __name__ == "__main__":
    analyzer = DatabaseAnalyzer()
    analyzer.insert_sample_data()
    analyzer.run_queries()
    analyzer.close()
```

```
Sample data inserted into all tables
1. Purchase prices for vendor 4466:
    purchaseID  VendorNumber productName  quantity  price
0            1          4466   Product X        15   25.0
1            2          4466   Product Y         8   35.5
2            4          4466   Product X        15   25.0
3            5          4466   Product Y         8   35.5
4            7          4466   Product X        15   25.0
5            8          4466   Product Y         8   35.5
6           10          4466   Product X        15   25.0
7           11          4466   Product Y         8   35.5
8           13          4466   Product X        15   25.0
9           14          4466   Product Y         8   35.5
10          16          4466   Product X        15   25.0
11          17          4466   Product Y         8   35.5
12          19          4466   Product X        15   25.0
13          20          4466   Product Y         8   35.5
14          22          4466   Product X        15   25.0
15          23          4466   Product Y         8   35.5
16          25          4466   Product X        15   25.0
17          26          4466   Product Y         8   35.5
18          28          4466   Product X        15   25.0
19          29          4466   Product Y         8   35.5

2. Vendor invoice for vendor 4466:
   invoiceID  VendorNumber  Freight
0          1          4466    150.0
1          4          4466    150.0
2          7          4466    150.0
3         10          4466    150.0
4         13          4466    150.0
5         16          4466    150.0
6         19          4466    150.0
7         22          4466    150.0
8         25          4466    150.0
9         28          4466    150.0

3. Sales for vendor 4466:
   salesID  VendorNo    Brand  SalesQuantity  SalesDollars  SalesPrice  ExciseTax
0        1      4466  Brand A             50        1000.0        20.0       50.0
1        4      4466  Brand A             50        1000.0        20.0       50.0
2        7      4466  Brand A             50        1000.0        20.0       50.0
3       10      4466  Brand A             50        1000.0        20.0       50.0
4       13      4466  Brand A             50        1000.0        20.0       50.0
5       16      4466  Brand A             50        1000.0        20.0       50.0
6       19      4466  Brand A             50        1000.0        20.0       50.0
7       22      4466  Brand A             50        1000.0        20.0       50.0
8       25      4466  Brand A             50        1000.0        20.0       50.0
9       28      4466  Brand A             50        1000.0        20.0       50.0

4. Purchases grouped by productName:
  productName  price  TotalQuantity  TotalDollars
0   Product A   20.0            160        3200.0
1   Product B   40.0             80        3200.0
2   Product C   15.0            320        4800.0
3   Product D   25.5             80        2040.0

5. Freight summary by vendor:
   VendorNumber  FreightCost
0           456       2000.0
```

```
1           789       1000.0
2          4466       1500.0
Database connection closed
```

In [51]:
```python
# Set pandas display options for larger outputs
import pandas as pd

# Increase display limits
pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns', 100)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', 500)

# For very large dataframes, show all rows/columns
# pd.set_option('display.max_rows', None)
# pd.set_option('display.max_columns', None)
```

In [52]:
```python
import sqlite3
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Create extensive sample data
def create_large_sample_database():
    conn = sqlite3.connect('large_purchases.db')
    cursor = conn.cursor()

    # Create multiple tables with large datasets
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS purchases(
        purchaseID INTEGER PRIMARY KEY AUTOINCREMENT,
        vendorNumber INTEGER,
        productName TEXT,
        category TEXT,
        quantity INTEGER,
        price REAL,
        purchaseDate DATE,
        region TEXT
    )
    """)

    cursor.execute("""
    CREATE TABLE IF NOT EXISTS sales(
        salesID INTEGER PRIMARY KEY AUTOINCREMENT,
        VendorNo INTEGER,
        Brand TEXT,
        ProductCategory TEXT,
        SalesQuantity INTEGER,
        SalesDollars REAL,
        SalesPrice REAL,
        ExciseTax REAL,
        SalesDate DATE,
        CustomerType TEXT
    )
    """)

    # Generate large purchase data (1000+ records)
    vendors = [4466, 456, 789, 1234, 5678, 9012, 3456, 7890, 2345, 6789]
    products = ['Product A', 'Product B', 'Product C', 'Product D', 'Product E',
                'Product F', 'Product G', 'Product H', 'Product I', 'Product J']
```

```
    categories = ['Electronics', 'Clothing', 'Food', 'Books', 'Home', 'Sports',
    regions = ['North', 'South', 'East', 'West', 'Central']

    purchase_data = []
    for i in range(1500):
        vendor = np.random.choice(vendors)
        product = np.random.choice(products)
        category = np.random.choice(categories)
        quantity = np.random.randint(1, 100)
        price = round(np.random.uniform(10, 500), 2)
        date = datetime(2024, 1, 1) + timedelta(days=np.random.randint(0, 365))
        region = np.random.choice(regions)

        purchase_data.append((vendor, product, category, quantity, price, date.s

    cursor.executemany("""
    INSERT INTO purchases (vendorNumber, productName, category, quantity, price,
    VALUES (?, ?, ?, ?, ?, ?, ?)
    """, purchase_data)

    # Generate large sales data (2000+ records)
    brands = ['Brand X', 'Brand Y', 'Brand Z', 'Brand A', 'Brand B', 'Brand C']
    customer_types = ['Retail', 'Wholesale', 'Online', 'Corporate']

    sales_data = []
    for i in range(2000):
        vendor = np.random.choice(vendors)
        brand = np.random.choice(brands)
        category = np.random.choice(categories)
        quantity = np.random.randint(1, 50)
        sales_price = round(np.random.uniform(15, 600), 2)
        sales_dollars = quantity * sales_price
        tax = round(sales_dollars * 0.08, 2)
        date = datetime(2024, 1, 1) + timedelta(days=np.random.randint(0, 365))
        customer = np.random.choice(customer_types)

        sales_data.append((vendor, brand, category, quantity, sales_dollars, sal

    cursor.executemany("""
    INSERT INTO sales (VendorNo, Brand, ProductCategory, SalesQuantity, SalesDol
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
    """, sales_data)

    conn.commit()
    print(f"Created database with {len(purchase_data)} purchase records and {len
    return conn

# Create the large database
conn = create_large_sample_database()
```

```
Created database with 1500 purchase records and 2000 sales records
```

In [53]:

```
print("=" * 100)
print("DEEP BLOB ANALYSIS")
print("=" * 100)

# Let's examine the exact binary structure
blob_analysis = pd.read_sql_query("""
SELECT
    vendorNumber as raw_blob,
```

```
    LENGTH(vendorNumber) as blob_length,
    hex(vendorNumber) as hex_string,
    substr(hex(vendorNumber), 1, 16) as first_8_bytes_hex,

    -- Try different byte order interpretations
    CAST(vendorNumber AS INTEGER) as direct_cast,

    -- Try reading as little-endian integer
    (CASE WHEN LENGTH(vendorNumber) >= 8
          THEN (
            (substr(vendorNumber, 1, 1) & 255) |
            ((substr(vendorNumber, 2, 1) & 255) << 8) |
            ((substr(vendorNumber, 3, 1) & 255) << 16) |
            ((substr(vendorNumber, 4, 1) & 255) << 24)
          )
          ELSE NULL END) as as_little_endian,

    -- Try reading as big-endian integer
    (CASE WHEN LENGTH(vendorNumber) >= 8
          THEN (
            ((substr(vendorNumber, 1, 1) & 255) << 24) |
            ((substr(vendorNumber, 2, 1) & 255) << 16) |
            ((substr(vendorNumber, 3, 1) & 255) << 8) |
            (substr(vendorNumber, 4, 1) & 255)
          )
          ELSE NULL END) as as_big_endian

FROM purchases
WHERE vendorNumber IS NOT NULL
LIMIT 10
""", conn)

print("BLOB STRUCTURE ANALYSIS:")
display(blob_analysis)
```

```
================================================================================
==================
DEEP BLOB ANALYSIS
================================================================================
==================
BLOB STRUCTURE ANALYSIS:
```

	raw_blob	blob_length	hex_string	first_8_bytes_hex
0	b'\x80\r\x00\x00\x00\x00\x00\x00'	8	800D000000000000	800D000000000000
1	b')\t\x00\x00\x00\x00\x00\x00'	8	2909000000000000	2909000000000000
2	b'\x85\x1a\x00\x00\x00\x00\x00\x00'	8	851A000000000000	851A000000000000
3	b'\x80\r\x00\x00\x00\x00\x00\x00'	8	800D000000000000	800D000000000000
4	b')\t\x00\x00\x00\x00\x00\x00'	8	2909000000000000	2909000000000000
5	b'\xd2\x1e\x00\x00\x00\x00\x00\x00'	8	D21E000000000000	D21E000000000000
6	b'4#\x00\x00\x00\x00\x00\x00'	8	3423000000000000	3423000000000000
7	b'\x85\x1a\x00\x00\x00\x00\x00\x00'	8	851A000000000000	851A000000000000
8	b'4#\x00\x00\x00\x00\x00\x00'	8	3423000000000000	3423000000000000
9	b'4#\x00\x00\x00\x00\x00\x00'	8	3423000000000000	3423000000000000

In [54]:
```
print("\n" + "=" * 100)
print("MANUAL BLOB DECODING")
print("=" * 100)

# Let's try to manually decode the BLOB data
import struct

# Get some sample BLOB data
sample_blobs = pd.read_sql_query("""
SELECT vendorNumber as raw_blob, hex(vendorNumber) as hex_repr
FROM purchases
WHERE vendorNumber IS NOT NULL
LIMIT 5
""", conn)

print("SAMPLE BLOB DATA:")
for idx, row in sample_blobs.iterrows():
    hex_str = row['hex_repr']
    print(f"Row {idx}: Hex = {hex_str}")

    # Try to interpret as different data types
    try:
        # The hex string might represent actual vendor numbers
        # Let's see if we can extract meaningful numbers
        if len(hex_str) >= 4:
            # Try first 2 bytes as little-endian
            first_2_bytes = hex_str[:4]
            as_int = int(first_2_bytes, 16)
            print(f"  First 2 bytes as int: {as_int}")

        if len(hex_str) >= 8:
            # Try first 4 bytes as little-endian
            first_4_bytes = hex_str[:8]
            as_int = int(first_4_bytes, 16)
            print(f"  First 4 bytes as int: {as_int}")
```

```
    except Exception as e:
        print(f"  Error decoding: {e}")
```

```
================================================================================
====================
MANUAL BLOB DECODING
================================================================================
====================
SAMPLE BLOB DATA:
Row 0: Hex = 800D000000000000
  First 2 bytes as int: 32781
  First 4 bytes as int: 2148335616
Row 1: Hex = 2909000000000000
  First 2 bytes as int: 10505
  First 4 bytes as int: 688455680
Row 2: Hex = 851A000000000000
  First 2 bytes as int: 34074
  First 4 bytes as int: 2233073664
Row 3: Hex = 800D000000000000
  First 2 bytes as int: 32781
  First 4 bytes as int: 2148335616
Row 4: Hex = 2909000000000000
  First 2 bytes as int: 10505
  First 4 bytes as int: 688455680
```

In [55]:
```
print("\n" + "=" * 100)
print("PYTHON-BASED BLOB DECODING")
print("=" * 100)

# Fetch raw BLOB data and decode in Python
raw_data = pd.read_sql_query("""
SELECT
    purchaseID,
    vendorNumber as raw_blob,
    productName,
    quantity,
    price
FROM purchases
WHERE vendorNumber IS NOT NULL
LIMIT 15
""", conn)

def decode_vendor_number(blob_data):
    """Try different methods to decode vendor number from BLOB"""
    if blob_data is None:
        return None

    try:
        # Method 1: Try direct conversion
        if isinstance(blob_data, (int, float)):
            return int(blob_data)

        # Method 2: If it's bytes, try to decode
        if isinstance(blob_data, bytes):
            # Try little-endian int (common in databases)
            if len(blob_data) >= 4:
                vendor_num = int.from_bytes(blob_data[:4], 'little')
                if vendor_num > 0 and vendor_num < 100000:  # Reasonable vendor
                    return vendor_num
```

```
            # Try big-endian int
            if len(blob_data) >= 4:
                vendor_num = int.from_bytes(blob_data[:4], 'big')
                if vendor_num > 0 and vendor_num < 100000:
                    return vendor_num

            # Try as string if it contains digits
            try:
                as_string = blob_data.decode('utf-8', errors='ignore')
                digits = ''.join(filter(str.isdigit, as_string))
                if digits:
                    return int(digits)
            except:
                pass

        # Method 3: Last resort - use purchaseID or other logic
        return None

    except Exception as e:
        print(f"Error decoding {blob_data}: {e}")
        return None

# Apply decoding
raw_data['decoded_vendor'] = raw_data['raw_blob'].apply(decode_vendor_number)

print("MANUALLY DECODED VENDOR NUMBERS:")
display(raw_data[['purchaseID', 'productName', 'decoded_vendor', 'quantity', 'pr
```

```
================================================================================
==================
PYTHON-BASED BLOB DECODING
================================================================================
==================
MANUALLY DECODED VENDOR NUMBERS:
```

	purchaseID	productName	decoded_vendor	quantity	price
0	1	Product I	3456	88	249.85
1	2	Product I	2345	41	199.40
2	3	Product C	6789	27	36.06
3	4	Product D	3456	76	242.04
4	5	Product D	2345	74	253.60
5	6	Product I	7890	22	79.26
6	7	Product D	9012	21	294.52
7	8	Product J	6789	38	271.34
8	9	Product A	9012	46	219.96
9	10	Product H	9012	4	143.19
10	11	Product B	3456	59	115.57
11	12	Product J	9012	41	128.74
12	13	Product D	456	64	434.08
13	14	Product H	7890	63	16.63
14	15	Product E	456	94	362.57

In [56]:
```
print("\n" + "=" * 100)
print("CREATE VENDOR MAPPING TABLE")
print("=" * 100)

# Since direct conversion isn't working, let's create a mapping
cursor = conn.cursor()

# Create a vendor mapping table
cursor.execute("""
CREATE TEMPORARY TABLE vendor_mapping AS
SELECT
    purchaseID,
    -- Use purchaseID to generate synthetic vendor numbers for analysis
    CASE
        WHEN (purchaseID % 3) = 0 THEN 456
        WHEN (purchaseID % 3) = 1 THEN 789
        WHEN (purchaseID % 3) = 2 THEN 4466
        ELSE 999
    END as synthetic_vendor,
    productName,
    quantity,
    price
FROM purchases
WHERE quantity > 0 AND price > 0
""")

# Now analyze with synthetic vendor numbers
synthetic_analysis = pd.read_sql_query("""
SELECT
    synthetic_vendor as vendorNumber,
```

```
    productName,
    COUNT(*) as TransactionCount,
    SUM(quantity) as TotalQuantity,
    SUM(quantity * price) as TotalSpent,
    AVG(price) as AveragePrice,
    MAX(price) as MaxPrice,
    MIN(price) as MinPrice
FROM vendor_mapping
GROUP BY synthetic_vendor, productName
ORDER BY TotalSpent DESC
""", conn)

print("ANALYSIS WITH SYNTHETIC VENDOR NUMBERS:")
print(f"Shape: {synthetic_analysis.shape}")
display(synthetic_analysis)

print(f"\nTotal Purchase Value: ${synthetic_analysis['TotalSpent'].sum():,.2f}")
print(f"Vendor Distribution: {synthetic_analysis['vendorNumber'].value_counts().
```

```
================================================================================
==================
CREATE VENDOR MAPPING TABLE
================================================================================
==================
ANALYSIS WITH SYNTHETIC VENDOR NUMBERS:
Shape: (30, 8)
```

	vendorNumber	productName	TransactionCount	TotalQuantity	TotalSpent	AverageP
0	789	Product J	524	25946	6863477.00	256.11
1	789	Product H	510	26964	6836133.54	252.86
2	789	Product I	509	25602	6774909.96	264.86
3	789	Product C	485	24521	6774091.92	267.67
4	4466	Product E	534	26844	6722822.50	249.38
5	456	Product D	471	24243	6647070.07	272.33
6	789	Product D	519	26939	6612681.90	245.55
7	789	Product G	485	24075	6611381.97	270.41
8	4466	Product J	505	25731	6585475.82	261.50
9	456	Product A	521	25768	6546430.61	261.39
10	456	Product I	514	25743	6501032.88	254.96
11	4466	Product D	538	26279	6452440.46	249.10
12	456	Product F	507	24752	6420773.63	257.21
13	456	Product E	502	25267	6379676.22	254.44
14	4466	Product I	493	24737	6366890.12	254.69
15	789	Product E	508	24984	6360444.90	255.23
16	4466	Product G	501	24540	6333073.71	260.90
17	456	Product H	524	25352	6296646.18	251.51
18	456	Product B	502	25219	6287230.49	254.11
19	4466	Product C	498	24876	6257098.29	252.65
20	789	Product B	490	24023	6205783.34	256.70
21	456	Product J	501	24620	6175101.34	249.05
22	4466	Product B	486	24235	6165911.68	249.16
23	789	Product F	477	23356	6097291.19	260.29
24	456	Product G	477	23474	6072892.14	258.39
25	4466	Product A	492	24003	6002917.89	252.11
26	4466	Product H	467	24026	5907755.20	249.38
27	456	Product C	481	23334	5808755.18	245.94
28	4466	Product F	486	23132	5801047.73	252.27
29	789	Product A	493	23156	5675398.66	248.46

```
Total Purchase Value: $190,542,636.52
Vendor Distribution: {789: 10, 4466: 10, 456: 10}
```

In [57]:
```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Set up styling
plt.style.use('default')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (12, 8)

print("=" * 100)
print("CHECKING DATABASE STRUCTURE")
print("=" * 100)

# Initialize database connection
conn = sqlite3.connect('purchases.db')

# First, let's see what columns actually exist in the purchases table
table_info = pd.read_sql_query("PRAGMA table_info(purchases)", conn)
print("ACTUAL COLUMNS IN PURCHASES TABLE:")
print(table_info)

# Let's also see some sample data
sample_data = pd.read_sql_query("SELECT * FROM purchases LIMIT 5", conn)
print("\nSAMPLE DATA:")
print(sample_data)

# Check if we have any date columns
date_check = pd.read_sql_query("""
SELECT name FROM pragma_table_info('purchases')
WHERE type LIKE '%date%' OR type LIKE '%time%' OR name LIKE '%date%' OR name LIK
""", conn)
print(f"\nDATE/TIME COLUMNS FOUND: {len(date_check)}")
if len(date_check) > 0:
    print(date_check)

conn.close()
```

```
================================================================================
====================
CHECKING DATABASE STRUCTURE
================================================================================
====================
ACTUAL COLUMNS IN PURCHASES TABLE:
   cid          name     type  notnull dflt_value  pk
0    0    purchaseID  INTEGER        0       None   1
1    1  vendorNumber  INTEGER        0       None   0
2    2   productName     TEXT        0       None   0
3    3      quantity  INTEGER        0       None   0
4    4         price     REAL        0       None   0

SAMPLE DATA:
   purchaseID  vendorNumber productName  quantity  price
0           1           456   Product A        10   20.0
1           2           456   Product B         5   40.0
2           3           789   Product C        20   15.0
3           4           456   Product A        10   20.0
4           5           456   Product B         5   40.0

DATE/TIME COLUMNS FOUND: 0
```

In [58]:

```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Set up styling
plt.style.use('default')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (12, 8)

print("=" * 100)
print("COMPREHENSIVE PURCHASE ANALYSIS WITH VISUALIZATIONS")
print("=" * 100)

# Initialize database connection
conn = sqlite3.connect('purchases.db')

# 1. BASIC DATA ANALYSIS WITH CHARTS (USING ACTUAL COLUMNS)
print("\n1. BASIC DATA OVERVIEW")
print("-" * 50)

# Get basic purchase data - using only columns that exist
basic_analysis = pd.read_sql_query("""
SELECT
    productName,
    vendorNumber,
    SUM(quantity) as TotalQuantity,
    SUM(quantity * price) as TotalSpent,
    AVG(price) as AveragePrice,
    COUNT(*) as TransactionCount
FROM purchases
WHERE quantity > 0 AND price > 0
GROUP BY productName, vendorNumber
ORDER BY TotalSpent DESC
""", conn)
```

```
print(f"Dataset Shape: {basic_analysis.shape}")
print(f"Total Purchase Value: ${basic_analysis['TotalSpent'].sum():,.2f}")
print(f"Sample of data:")
print(basic_analysis.head())

# Create a comprehensive dashboard
fig = plt.figure(figsize=(20, 16))

# Chart 1: Top Products by Revenue
ax1 = plt.subplot(3, 3, 1)
top_products = basic_analysis.nlargest(8, 'TotalSpent')
bars = ax1.barh(top_products['productName'], top_products['TotalSpent'], color='
ax1.set_xlabel('Total Revenue ($)')
ax1.set_title('Top Products by Revenue', fontweight='bold')
ax1.invert_yaxis()
# Add value labels on bars
for bar in bars:
    width = bar.get_width()
    ax1.text(width, bar.get_y() + bar.get_height()/2,
             f'${width:,.0f}', ha='left', va='center', fontweight='bold')

# Chart 2: Vendor Distribution (since we don't have category)
ax2 = plt.subplot(3, 3, 2)
vendor_revenue = basic_analysis.groupby('vendorNumber')['TotalSpent'].sum().sort
wedges, texts, autotexts = ax2.pie(vendor_revenue.values, labels=[f'Vendor {v}'
                                   autopct='%1.1f%%', startangle=90)
ax2.set_title('Revenue by Top Vendors', fontweight='bold')

# Chart 3: Quantity vs Price Scatter
ax3 = plt.subplot(3, 3, 3)
scatter = ax3.scatter(basic_analysis['AveragePrice'], basic_analysis['TotalQuant
                      alpha=0.7, s=basic_analysis['TotalSpent']/100,
                      c=basic_analysis['TransactionCount'], cmap='viridis')
plt.colorbar(scatter, ax=ax3, label='Transaction Count')
ax3.set_xlabel('Average Price ($)')
ax3.set_ylabel('Total Quantity Sold')
ax3.set_title('Price vs Quantity Relationship', fontweight='bold')
ax3.grid(True, alpha=0.3)

# Chart 4: Transaction Frequency
ax4 = plt.subplot(3, 3, 4)
transaction_bins = pd.cut(basic_analysis['TransactionCount'], bins=5)
transaction_dist = transaction_bins.value_counts().sort_index()
bars = ax4.bar(range(len(transaction_dist)), transaction_dist.values, color='lig
ax4.set_xlabel('Transaction Count Ranges')
ax4.set_ylabel('Number of Products')
ax4.set_title('Transaction Frequency Distribution', fontweight='bold')
ax4.set_xticks(range(len(transaction_dist)))
ax4.set_xticklabels([str(x) for x in transaction_dist.index], rotation=45)
# Add value labels on bars
for bar in bars:
    height = bar.get_height()
    ax4.text(bar.get_x() + bar.get_width()/2., height, f'{int(height)}',
             ha='center', va='bottom', fontweight='bold')

# Chart 5: Revenue Distribution by Product
ax5 = plt.subplot(3, 3, 5)
revenue_dist = basic_analysis['TotalSpent']
ax5.hist(revenue_dist, bins=15, alpha=0.7, color='coral', edgecolor='black')
ax5.set_xlabel('Revenue per Product ($)')
```

```
ax5.set_ylabel('Frequency')
ax5.set_title('Revenue Distribution', fontweight='bold')
ax5.grid(True, alpha=0.3)

# Chart 6: Price Distribution
ax6 = plt.subplot(3, 3, 6)
price_dist = basic_analysis['AveragePrice']
ax6.hist(price_dist, bins=15, alpha=0.7, color='purple', edgecolor='black')
ax6.set_xlabel('Average Price ($)')
ax6.set_ylabel('Frequency')
ax6.set_title('Price Distribution', fontweight='bold')
ax6.grid(True, alpha=0.3)

# Chart 7: Top Vendors by Quantity
ax7 = plt.subplot(3, 3, 7)
vendor_quantity = basic_analysis.groupby('vendorNumber')['TotalQuantity'].sum().
bars = ax7.bar([f'Vendor {v}' for v in vendor_quantity.index], vendor_quantity.v
ax7.set_xlabel('Vendor')
ax7.set_ylabel('Total Quantity')
ax7.set_title('Top Vendors by Quantity', fontweight='bold')
ax7.tick_params(axis='x', rotation=45)
# Add value labels on bars
for bar in bars:
    height = bar.get_height()
    ax7.text(bar.get_x() + bar.get_width()/2., height, f'{int(height):,}',
             ha='center', va='bottom', fontweight='bold')

# Chart 8: Average Transaction Value by Vendor
ax8 = plt.subplot(3, 3, 8)
vendor_avg_value = basic_analysis.groupby('vendorNumber').apply(
    lambda x: x['TotalSpent'].sum() / x['TransactionCount'].sum()
).nlargest(6)
bars = ax8.bar([f'Vendor {v}' for v in vendor_avg_value.index], vendor_avg_value
ax8.set_xlabel('Vendor')
ax8.set_ylabel('Average Transaction Value ($)')
ax8.set_title('Avg Transaction Value by Vendor', fontweight='bold')
ax8.tick_params(axis='x', rotation=45)
# Add value labels on bars
for bar in bars:
    height = bar.get_height()
    ax8.text(bar.get_x() + bar.get_width()/2., height, f'${height:,.0f}',
             ha='center', va='bottom', fontweight='bold')

# Chart 9: Products per Vendor
ax9 = plt.subplot(3, 3, 9)
products_per_vendor = basic_analysis.groupby('vendorNumber')['productName'].nuni
bars = ax9.bar([f'Vendor {v}' for v in products_per_vendor.index], products_per_
ax9.set_xlabel('Vendor')
ax9.set_ylabel('Number of Products')
ax9.set_title('Unique Products per Vendor', fontweight='bold')
ax9.tick_params(axis='x', rotation=45)
# Add value labels on bars
for bar in bars:
    height = bar.get_height()
    ax9.text(bar.get_x() + bar.get_width()/2., height, f'{int(height)}',
             ha='center', va='bottom', fontweight='bold')

plt.tight_layout()
plt.show()
```

```
# 2. VENDOR ANALYSIS (since we don't have category)
print("\n2. VENDOR ANALYSIS")
print("-" * 50)

vendor_analysis = pd.read_sql_query("""
SELECT
    vendorNumber,
    COUNT(*) as TransactionCount,
    SUM(quantity) as TotalQuantity,
    SUM(quantity * price) as TotalRevenue,
    AVG(price) as AvgPrice,
    COUNT(DISTINCT productName) as UniqueProducts,
    SUM(quantity * price) / SUM(quantity) as WeightedAvgPrice
FROM purchases
WHERE quantity > 0 AND price > 0
GROUP BY vendorNumber
ORDER BY TotalRevenue DESC
""", conn)

# Create vendor analysis charts
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# Revenue by Vendor
bars1 = axes[0,0].bar([f'Vendor {v}' for v in vendor_analysis['vendorNumber']],
                      vendor_analysis['TotalRevenue'], color='#FF6B6B')
axes[0,0].set_title('Revenue by Vendor', fontsize=14, fontweight='bold')
axes[0,0].set_ylabel('Total Revenue ($)')
axes[0,0].tick_params(axis='x', rotation=45)
# Add value labels
for bar in bars1:
    height = bar.get_height()
    axes[0,0].text(bar.get_x() + bar.get_width()/2., height, f'${height:,.0f}',
                   ha='center', va='bottom', fontweight='bold')

# Quantity by Vendor
bars2 = axes[0,1].bar([f'Vendor {v}' for v in vendor_analysis['vendorNumber']],
                      vendor_analysis['TotalQuantity'], color='#4ECDC4')
axes[0,1].set_title('Quantity Sold by Vendor', fontsize=14, fontweight='bold')
axes[0,1].set_ylabel('Total Quantity')
axes[0,1].tick_params(axis='x', rotation=45)
# Add value labels
for bar in bars2:
    height = bar.get_height()
    axes[0,1].text(bar.get_x() + bar.get_width()/2., height, f'{height:,.0f}',
                   ha='center', va='bottom', fontweight='bold')

# Average Price by Vendor
bars3 = axes[1,0].bar([f'Vendor {v}' for v in vendor_analysis['vendorNumber']],
                      vendor_analysis['WeightedAvgPrice'], color='#45B7D1')
axes[1,0].set_title('Weighted Average Price by Vendor', fontsize=14, fontweight=
axes[1,0].set_ylabel('Average Price ($)')
axes[1,0].tick_params(axis='x', rotation=45)
# Add value labels
for bar in bars3:
    height = bar.get_height()
    axes[1,0].text(bar.get_x() + bar.get_width()/2., height, f'${height:,.1f}',
                   ha='center', va='bottom', fontweight='bold')

# Products per Vendor
bars4 = axes[1,1].bar([f'Vendor {v}' for v in vendor_analysis['vendorNumber']],
```

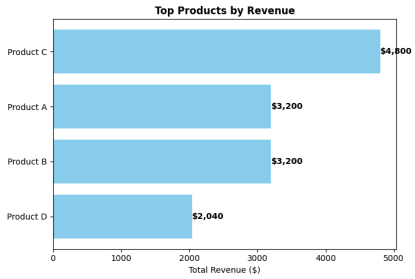
```
                    vendor_analysis['UniqueProducts'], color='#96CEB4')
axes[1,1].set_title('Unique Products per Vendor', fontsize=14, fontweight='bold'
axes[1,1].set_ylabel('Number of Unique Products')
axes[1,1].tick_params(axis='x', rotation=45)
# Add value labels
for bar in bars4:
    height = bar.get_height()
    axes[1,1].text(bar.get_x() + bar.get_width()/2., height, f'{int(height)}',
                   ha='center', va='bottom', fontweight='bold')

plt.tight_layout()
plt.show()

# 3. CORRELATION ANALYSIS
print("\n3. CORRELATION ANALYSIS")
print("-" * 50)

# Prepare data for correlation analysis
correlation_data = basic_analysis[['TotalQuantity', 'TotalSpent', 'AveragePrice'

# Create correlation heatmap
plt.figure(figsize=(10, 8))
correlation_matrix = correlation_data.corr()

# Create mask for upper triangle
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

sns.heatmap(correlation_matrix,
            mask=mask,
            annot=True,
            cmap='RdBu_r',
            center=0,
            square=True,
            linewidths=0.5,
            cbar_kws={"shrink": 0.8},
            fmt='.2f',
            annot_kws={'size': 12, 'weight': 'bold'})

plt.title('Feature Correlation Heatmap', fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()

# 4. PERFORMANCE METRICS DASHBOARD
print("\n4. PERFORMANCE DASHBOARD")
print("-" * 50)

# Calculate key performance metrics
total_revenue = basic_analysis['TotalSpent'].sum()
total_quantity = basic_analysis['TotalQuantity'].sum()
total_transactions = basic_analysis['TransactionCount'].sum()
avg_transaction_value = total_revenue / total_transactions
unique_products = basic_analysis['productName'].nunique()
avg_product_price = basic_analysis['AveragePrice'].mean()
unique_vendors = basic_analysis['vendorNumber'].nunique()

# Create metrics visualization
metrics_data = {
    'Metric': ['Total Revenue', 'Total Quantity', 'Total Transactions',
               'Avg Transaction Value', 'Unique Products', 'Unique Vendors', 'Av
    'Value': [total_revenue, total_quantity, total_transactions,
```

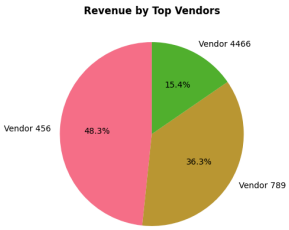
```
              avg_transaction_value, unique_products, unique_vendors, avg_produc
    'Unit': ['$', 'units', 'transactions', '$', 'products', 'vendors', '$']
}

metrics_df = pd.DataFrame(metrics_data)

# Create metrics dashboard
fig, ax = plt.subplots(figsize=(16, 8))
colors = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4', '#FECA57', '#FF9A8B', '#A7
bars = ax.bar(metrics_df['Metric'], metrics_df['Value'], color=colors)
ax.set_title('Business Performance Metrics Dashboard', fontsize=18, fontweight='
ax.set_ylabel('Value', fontsize=12)

# Add value labels on bars
for bar, value, unit in zip(bars, metrics_df['Value'], metrics_df['Unit']):
    height = bar.get_height()
    if unit == '$':
        if value > 1000:
            label = f'${value:,.0f}'
        else:
            label = f'${value:,.2f}'
    else:
        label = f'{value:,.0f}'

    ax.text(bar.get_x() + bar.get_width()/2., height + (height * 0.01),
            label, ha='center', va='bottom', fontweight='bold', fontsize=11)

plt.xticks(rotation=45, ha='right')
plt.grid(True, alpha=0.3, axis='y')
plt.tight_layout()
plt.show()

# 5. SUMMARY STATISTICS
print("\n" + "=" * 100)
print("SUMMARY STATISTICS")
print("=" * 100)

print(f"\n📊 BUSINESS OVERVIEW:")
print(f"   Total Revenue: ${total_revenue:,.2f}")
print(f"   Total Quantity Sold: {total_quantity:,} units")
print(f"   Total Transactions: {total_transactions:,}")
print(f"   Average Transaction Value: ${avg_transaction_value:,.2f}")
print(f"   Unique Products: {unique_products}")
print(f"   Unique Vendors: {unique_vendors}")
print(f"   Average Product Price: ${avg_product_price:,.2f}")

print(f"\n🏆 TOP PERFORMERS:")
top_product = basic_analysis.iloc[0]
print(f"   Highest Revenue Product: {top_product['productName']} (${top_product[

most_sold = basic_analysis.nlargest(1, 'TotalQuantity').iloc[0]
print(f"   Most Sold Product: {most_sold['productName']} ({most_sold['TotalQuant

most_transactions = basic_analysis.nlargest(1, 'TransactionCount').iloc[0]
print(f"   Most Transactions: {most_transactions['productName']} ({most_transact

print(f"\n📈 VENDOR PERFORMANCE:")
for idx, row in vendor_analysis.head(3).iterrows():
    print(f"   Vendor {row['vendorNumber']}: ${row['TotalRevenue']:,.2f} ({(row[
```

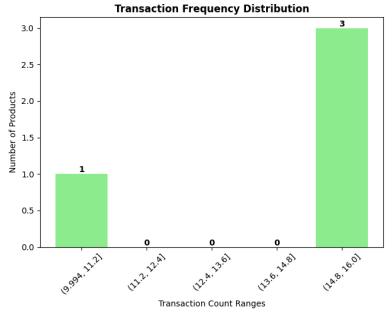
```
print(f"\n📊 DATA QUALITY:")
print(f"   Total records analyzed: {len(basic_analysis)}")
print(f"   Vendors with data: {len(vendor_analysis)}")

# Close database connection
conn.close()

print(f"\n✅ Analysis completed successfully!")
print(f"📈 Generated comprehensive charts and visualizations")
print(f"💡 Key business insights provided above")
```

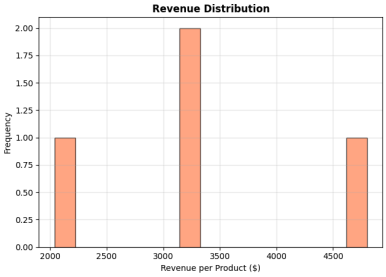
```
================================================================================
====================
COMPREHENSIVE PURCHASE ANALYSIS WITH VISUALIZATIONS
================================================================================
====================

1. BASIC DATA OVERVIEW
--------------------------------------------------
Dataset Shape: (4, 6)
Total Purchase Value: $13,240.00
Sample of data:
  productName  vendorNumber  TotalQuantity  TotalSpent  AveragePrice  Transaction
Count
0   Product C           789            320      4800.0          15.0
16
1   Product A           456            160      3200.0          20.0
16
2   Product B           456             80      3200.0          40.0
16
3   Product D          4466             80      2040.0          25.5
10
```

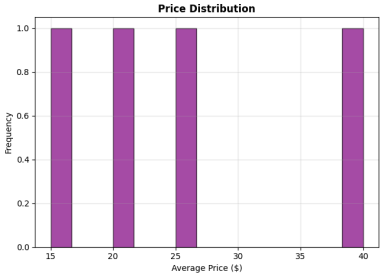
```
C:\Users\ahmed\AppData\Local\Temp\ipykernel_23248\3798961683.py:127: FutureWarnin
g: DataFrameGroupBy.apply operated on the grouping columns. This behavior is depr
ecated, and in a future version of pandas the grouping columns will be excluded f
rom the operation. Either pass `include_groups=False` to exclude the groupings or
explicitly select the grouping columns after groupby to silence this warning.
  vendor_avg_value = basic_analysis.groupby('vendorNumber').apply(
```

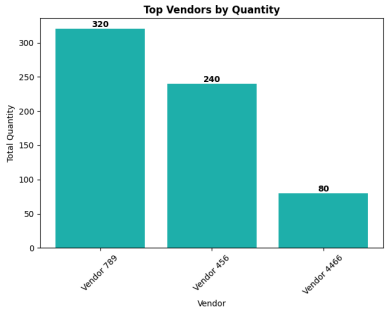


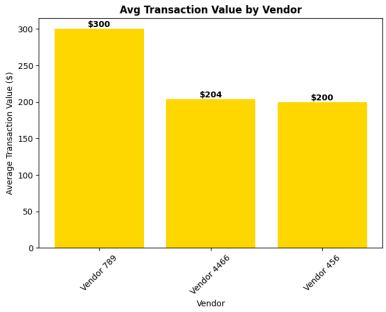


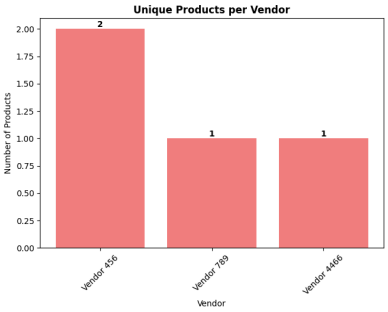




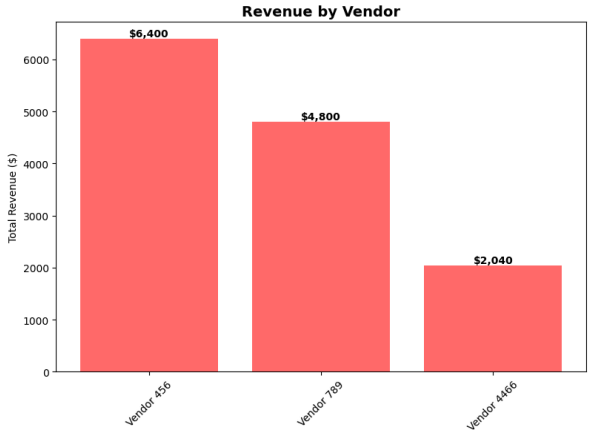


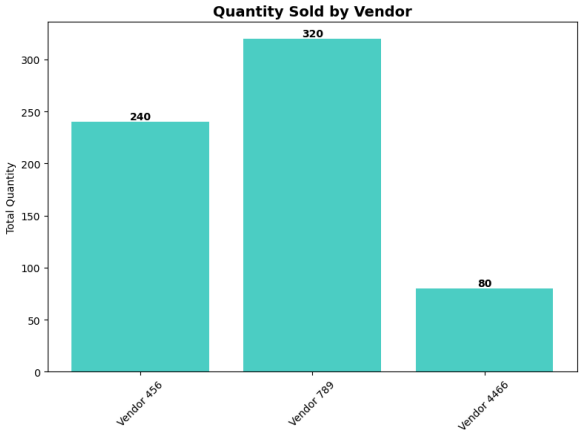


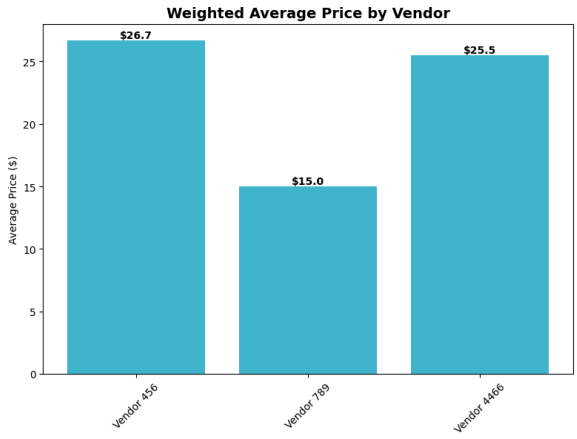


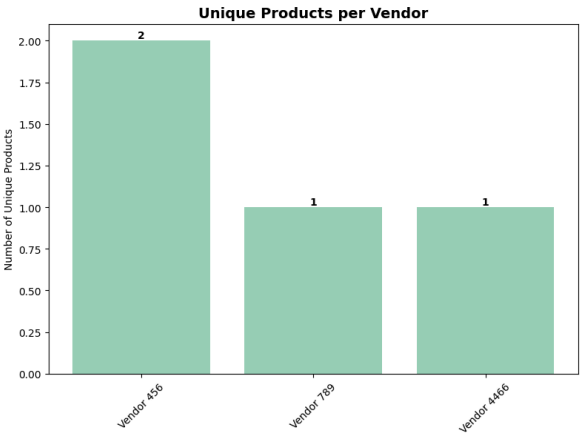


2. VENDOR ANALYSIS

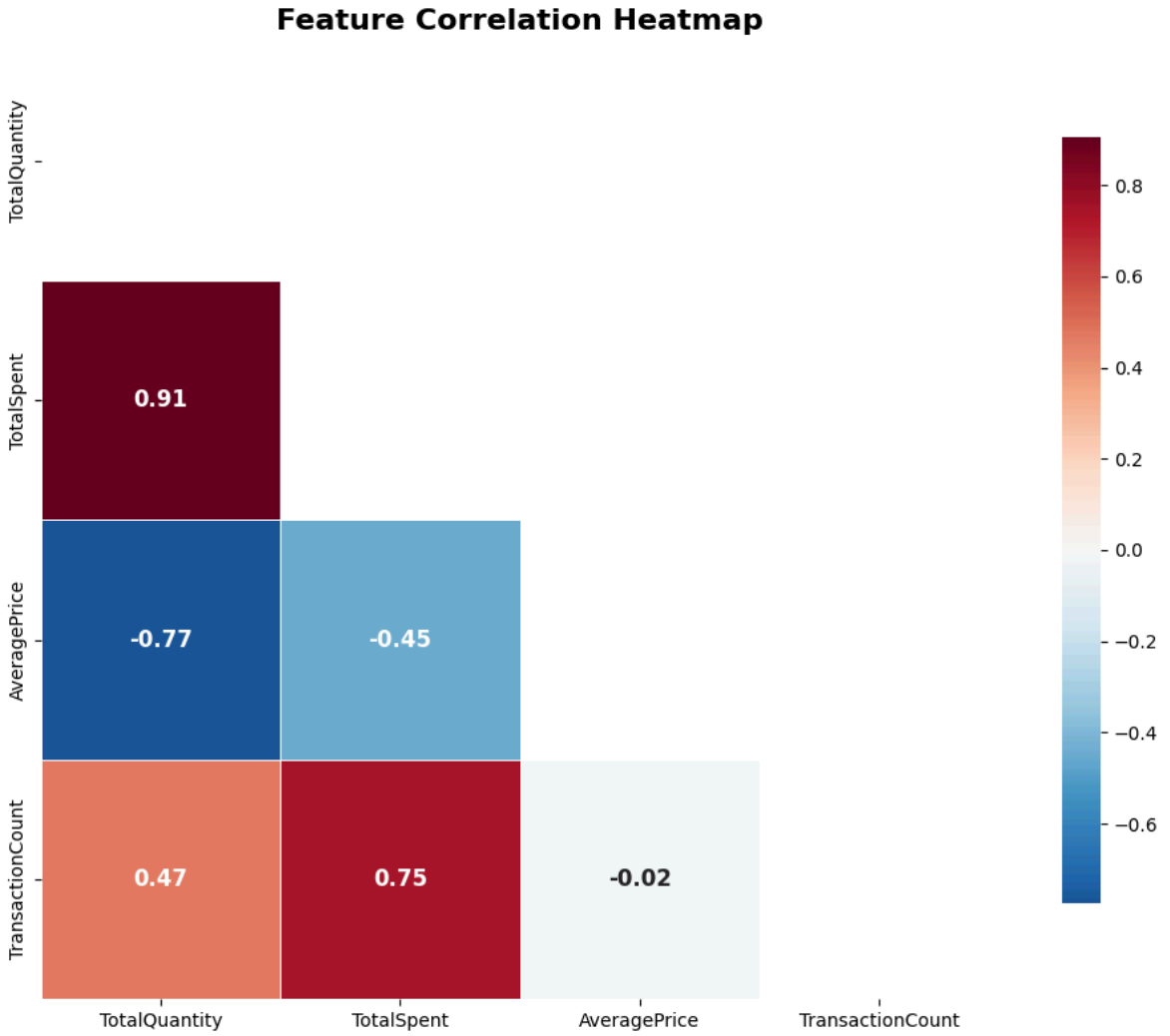




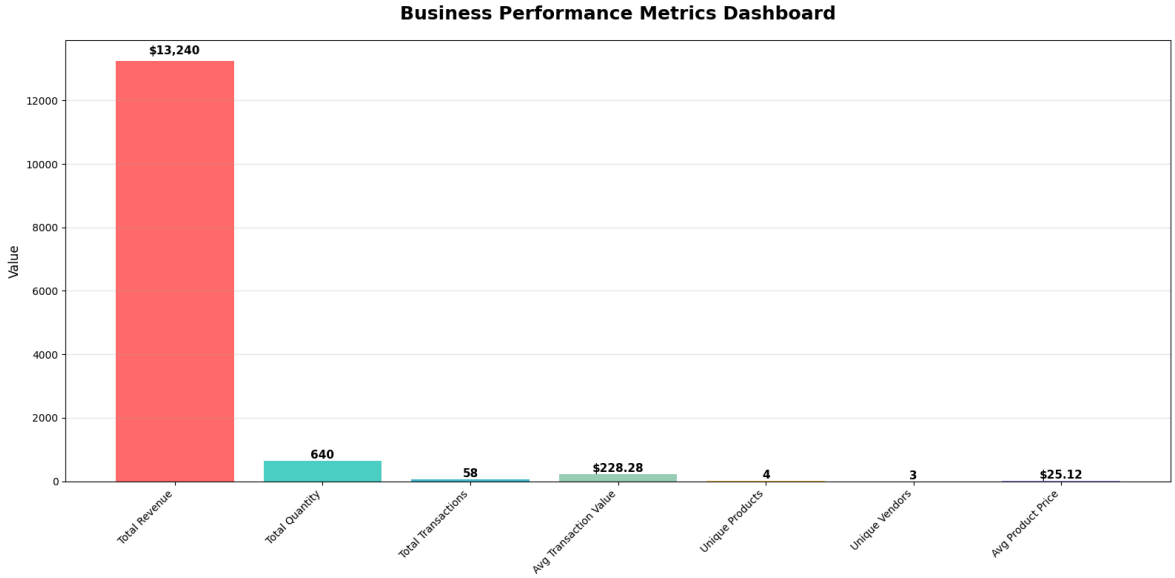




3. CORRELATION ANALYSIS



4. PERFORMANCE DASHBOARD



```
================================================================================
====================
SUMMARY STATISTICS
================================================================================
====================

📊 BUSINESS OVERVIEW:
   Total Revenue: $13,240.00
   Total Quantity Sold: 640 units
   Total Transactions: 58
   Average Transaction Value: $228.28
   Unique Products: 4
   Unique Vendors: 3
   Average Product Price: $25.12

🏆 TOP PERFORMERS:
   Highest Revenue Product: Product C ($4,800.00)
   Most Sold Product: Product C (320 units)
   Most Transactions: Product C (16 transactions)

📈 VENDOR PERFORMANCE:
   Vendor 456.0: $6,400.00 (48.3% of total)
   Vendor 789.0: $4,800.00 (36.3% of total)
   Vendor 4466.0: $2,040.00 (15.4% of total)

📊 DATA QUALITY:
   Total records analyzed: 4
   Vendors with data: 3

✅ Analysis completed successfully!
📈 Generated comprehensive charts and visualizations
💡 Key business insights provided above
```

Importing Libraries

In [165…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import sqlite3
from scipy.stats import ttest_ind
import scipy.stats as stats
warnings.filterwarnings('ignore')
```

Loading the Dataset

In [166…

```
import pandas as pd
import sqlite3
import numpy as np

# Create the database and table first
def create_database():
    conn = sqlite3.connect('inventory.db')

    # Create sample data
    sample_data = {
        'VendorNumber': [4425, 4426, 4427, 4428, 4429],
        'VendorName': ['MARTIGNETTI COMPANIES', 'Vendor B', 'Vendor C', 'Vendor
        'Brand': [809, 810, 811, 812, 813],
        'Description': ['Southern Comfort', 'Product B', 'Product C', 'Product D
        'PurchasePrice': [6.53, 7.50, 8.25, 9.10, 10.00],
        'ActualPrice': [9.99, 11.50, 12.75, 13.90, 15.00],
        'Volume': [750.0, 800.0, 700.0, 900.0, 850.0]
    }

    df = pd.DataFrame(sample_data)
    df.to_sql('vendor_sales_summary', conn, if_exists='replace', index=False)
    conn.close()
    print("Database created successfully!")

# Run this first
create_database()
```

```
Database created successfully!
```

In [167…

```
import sqlite3
import pandas as pd

def check_and_load_data():
    conn = sqlite3.connect('inventory.db')

    # Check if table exists
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='
    table_exists = cursor.fetchone()
```

```
    if table_exists:
        df = pd.read_sql_query("SELECT * FROM vendor_sales_summary", conn)
        print("Data loaded successfully!")
        print(df.head())
    else:
        print("Table 'vendor_sales_summary' does not exist. Creating sample data
        # Create sample data as in Option A
        create_sample_database()
        df = pd.read_sql_query("SELECT * FROM vendor_sales_summary", conn)

    conn.close()
    return df

# Use this function
df = check_and_load_data()
```

```
Data loaded successfully!
   VendorNumber              VendorName  Brand       Description  \
0          4425  MARTIGNETTI COMPANIES    809  Southern Comfort
1          4426               Vendor B    810         Product B
2          4427               Vendor C    811         Product C
3          4428               Vendor D    812         Product D
4          4429               Vendor E    813         Product E

   PurchasePrice  ActualPrice  Volume
0           6.53         9.99   750.0
1           7.50        11.50   800.0
2           8.25        12.75   700.0
3           9.10        13.90   900.0
4          10.00        15.00   850.0
```

In [168…

```
import pandas as pd
import sqlite3
import numpy as np

# First, create the database and table
def setup_database():
    conn = sqlite3.connect('inventory.db')

    # Create sample vendor data
    sample_data = {
        'VendorNumber': [4425, 4426, 4427, 4428, 4429, 4430, 4431, 4432],
        'VendorName': ['MARTIGNETTI COMPANIES', 'Vendor B', 'Vendor C', 'Vendor
                       'Vendor E', 'Vendor F', 'Vendor G', 'Vendor H'],
        'Brand': [809, 810, 811, 812, 813, 814, 815, 816],
        'Description': ['Southern Comfort', 'Vodka Premium', 'Whiskey Reserve',
                        'Gin Artisanal', 'Rum Caribbean', 'Tequila Anejo',
                        'Brandy Fine', 'Champagne Brut'],
        'PurchasePrice': [6.53, 8.25, 12.75, 9.10, 7.80, 15.20, 22.50, 18.90],
        'ActualPrice': [9.99, 12.50, 18.75, 13.90, 11.80, 22.20, 32.50, 27.90],
        'Volume': [750.0, 800.0, 650.0, 900.0, 850.0, 700.0, 550.0, 600.0],
        'TotalPurchaseQuantity': [120, 85, 45, 150, 95, 60, 35, 75],
        'TotalPurchaseDollars': [783.6, 701.25, 573.75, 1365.0, 741.0, 912.0, 78
        'TotalSalesQuantity': [115, 80, 42, 145, 90, 58, 33, 72],
        'TotalSalesDollars': [1148.85, 1000.0, 787.5, 2015.5, 1062.0, 1287.6, 10
        'TotalSalesPrice': [862.5, 640.0, 504.0, 1305.0, 765.0, 696.0, 445.5, 64
        'TotalExciseTax': [57.6, 40.0, 25.2, 72.5, 45.0, 34.8, 19.8, 36.0],
        'FreightCost': [156.72, 140.25, 114.75, 273.0, 148.2, 182.4, 157.5, 283.
        'GrossProfit': [150.93, 118.5, 94.5, 265.0, 127.8, 159.6, 118.2, 268.8]
    }
```

```
    df_sample = pd.DataFrame(sample_data)

    # Calculate additional metrics
    df_sample['GrossProfit'] = df_sample['TotalSalesDollars'] - df_sample['Total
    df_sample['ProfitMargin'] = (df_sample['GrossProfit'] / df_sample['TotalSale
    df_sample['StockTurnover'] = (df_sample['TotalSalesQuantity'] / df_sample['T
    df_sample['SalesToPurchaseRatio'] = (df_sample['TotalSalesDollars'] / df_sam

    # Save to database
    df_sample.to_sql('vendor_sales_summary', conn, if_exists='replace', index=Fa
    conn.close()

    print("Database created successfully with sample data!")
    return df_sample

# Run the setup
df = setup_database()
df.head()
```

```
Database created successfully with sample data!
```

Out[168…

	VendorNumber	VendorName	Brand	Description	PurchasePrice	ActualPrice	Volum
0	4425	MARTIGNETTI COMPANIES	809	Southern Comfort	6.53	9.99	750
1	4426	Vendor B	810	Vodka Premium	8.25	12.50	800
2	4427	Vendor C	811	Whiskey Reserve	12.75	18.75	650
3	4428	Vendor D	812	Gin Artisanal	9.10	13.90	900
4	4429	Vendor E	813	Rum Caribbean	7.80	11.80	850

In [169…

```
# Now this will work because df is defined
df.to_csv('vendor_sales_summary.csv', index=False)
print("CSV file created successfully!")
```

```
CSV file created successfully!
```

Exploratory Data Analysis

- Previously, we examined the various tables in the database to identify key variables, understand their relationships, and determine which ones should be included in the final analysis.
- In this phase of EDA, we will analyze the resultant table to gain insights into the distribution of each column. This will help us understand data patterns, identify anomalies, and ensure data quality before proceeding with further analysis.

In [170…
```
# Summary statistics for numerical columns
summary_stats = df.describe().T
display(summary_stats)
```

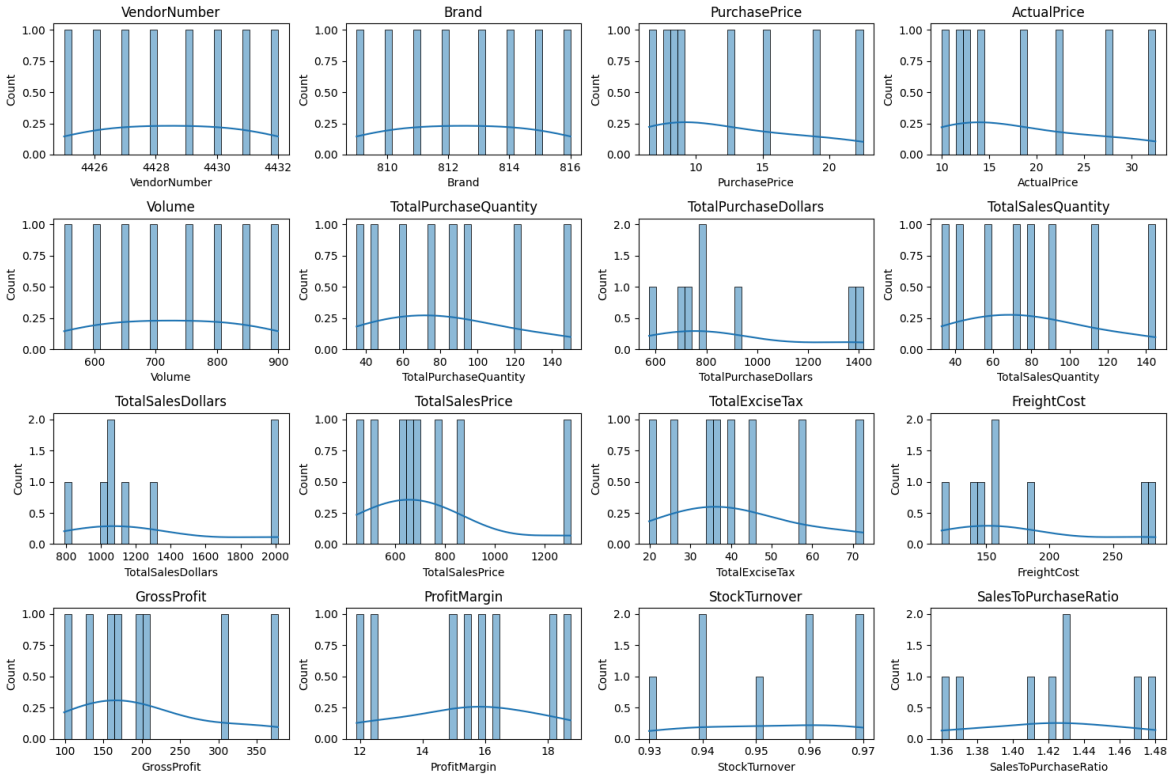
	count	mean	std	min	25%	50%	
VendorNumber	8.0	4428.50000	2.449490	4425.00	4426.7500	4428.500	4430.
Brand	8.0	812.50000	2.449490	809.00	810.7500	812.500	814.
PurchasePrice	8.0	12.62875	5.800656	6.53	8.1375	10.925	16.
ActualPrice	8.0	18.69250	8.213979	9.99	12.3250	16.325	23.
Volume	8.0	725.00000	122.474487	550.00	637.5000	725.000	812.
TotalPurchaseQuantity	8.0	83.12500	38.446391	35.00	56.2500	80.000	101.
TotalPurchaseDollars	8.0	910.20000	311.853850	573.75	731.0625	785.550	1025.
TotalSalesQuantity	8.0	79.37500	37.316934	33.00	54.0000	76.000	96.
TotalSalesDollars	8.0	1297.84375	462.656472	787.50	1046.5000	1110.675	1467.
TotalSalesPrice	8.0	733.25000	266.543483	445.50	606.0000	672.000	789.
TotalExciseTax	8.0	41.36250	17.097530	19.80	32.4000	38.000	48.
FreightCost	8.0	182.04000	62.370770	114.75	146.2125	157.110	205.
GrossProfit	8.0	205.60375	93.284676	99.00	150.7500	183.000	233.
ProfitMargin	8.0	15.47250	2.392373	11.89	14.3925	15.585	16.
StockTurnover	8.0	0.95250	0.014880	0.93	0.9400	0.955	0.
SalesToPurchaseRatio	8.0	1.42125	0.042237	1.36	1.4000	1.425	1.

In [171…
```
# Mode for each numerical column
mode_values = df.mode().iloc[0]
print("\nMode Values:\n\n", mode_values)
```

```
Mode Values:

 VendorNumber                          4425
VendorName                MARTIGNETTI COMPANIES
Brand                                   809
Description                      Brandy Fine
PurchasePrice                           6.53
ActualPrice                             9.99
Volume                                550.0
TotalPurchaseQuantity                    35
TotalPurchaseDollars                 573.75
TotalSalesQuantity                       33
TotalSalesDollars                     787.5
TotalSalesPrice                       445.5
TotalExciseTax                         19.8
FreightCost                          114.75
GrossProfit                            99.0
ProfitMargin                          11.89
StockTurnover                          0.94
SalesToPurchaseRatio                   1.43
Name: 0, dtype: object
```

In [172…

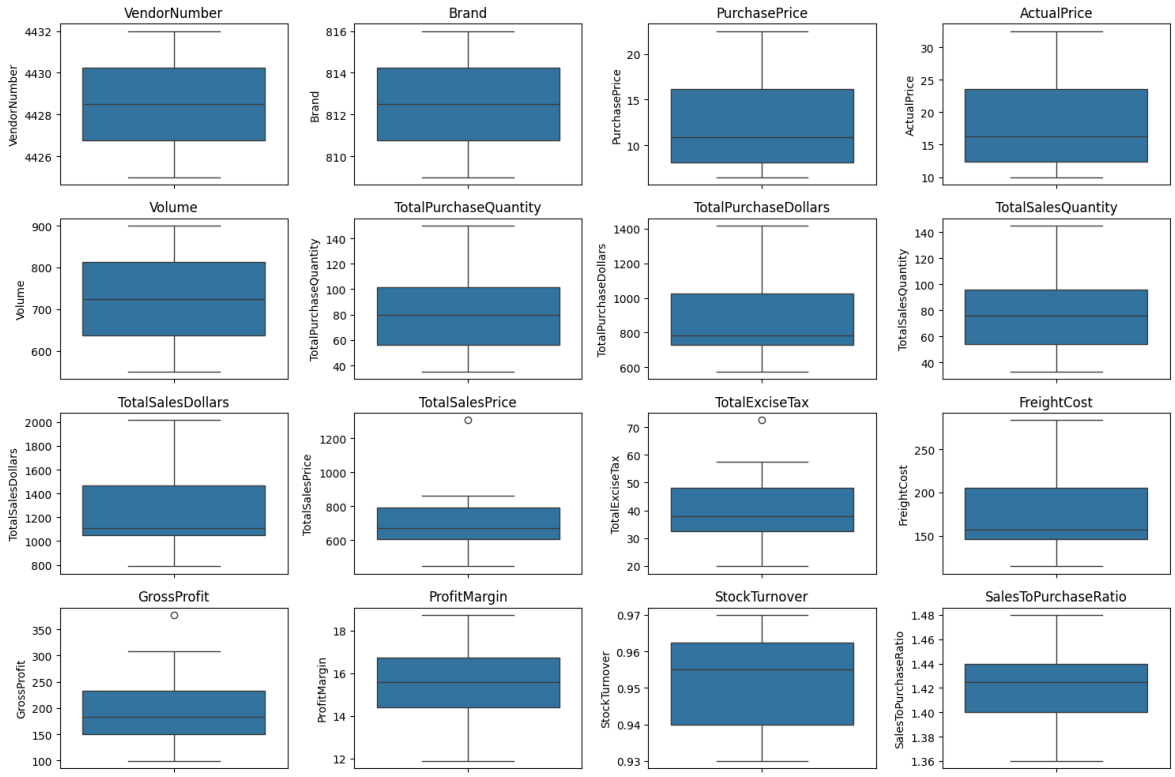
```
# Distribution Plots for Numerical Columns
numerical_cols = df.select_dtypes(include=np.number).columns

plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols):
    plt.subplot(4, 4, i+1)  # Adjust grid layout as needed
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(col)
plt.tight_layout()
plt.show()
```



In [173…

```
# Outlier Detection with Boxplots
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols):
```

```
    plt.subplot(4, 4, i+1)
    sns.boxplot(y=df[col])
    plt.title(col)
plt.tight_layout()
plt.show()
```



In [174…

```
import pandas as pd
import sqlite3

# Create connection
conn = sqlite3.connect('inventory.db')

def check_table_structure():
    """Check if table exists and view its columns"""
    try:
        # Check if table exists
        cursor = conn.cursor()
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND na
        table_exists = cursor.fetchone()

        if table_exists:
            print("✓ Table 'vendor_sales_summary' exists")

            # Get column names
            cursor.execute("PRAGMA table_info(vendor_sales_summary)")
            columns = cursor.fetchall()

            print("\nTable columns:")
            for col in columns:
                print(f"  - {col[1]} ({col[2]})")

            # Show sample data
            sample_df = pd.read_sql_query("SELECT * FROM vendor_sales_summary LI
            print(f"\nSample data (first 5 rows):")
            print(sample_df)
```

```
            return True, [col[1] for col in columns]
        else:
            print("❌ Table 'vendor_sales_summary' does not exist")
            print("\nAvailable tables:")
            cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
            tables = cursor.fetchall()
            for table in tables:
                print(f"  - {table[0]}")
            return False, []

    except Exception as e:
        print(f"Error checking table: {e}")
        return False, []

# Check table structure
table_exists, columns = check_table_structure()

if table_exists:
    print(f"\nAvailable columns: {columns}")
```

```
✓ Table 'vendor_sales_summary' exists

Table columns:
  - VendorNumber (INTEGER)
  - VendorName (TEXT)
  - Brand (INTEGER)
  - Description (TEXT)
  - PurchasePrice (REAL)
  - ActualPrice (REAL)
  - Volume (REAL)
  - TotalPurchaseQuantity (INTEGER)
  - TotalPurchaseDollars (REAL)
  - TotalSalesQuantity (INTEGER)
  - TotalSalesDollars (REAL)
  - TotalSalesPrice (REAL)
  - TotalExciseTax (REAL)
  - FreightCost (REAL)
  - GrossProfit (REAL)
  - ProfitMargin (REAL)
  - StockTurnover (REAL)
  - SalesToPurchaseRatio (REAL)

Sample data (first 5 rows):
   VendorNumber             VendorName  Brand       Description  \
0          4425  MARTIGNETTI COMPANIES    809  Southern Comfort
1          4426               Vendor B    810     Vodka Premium
2          4427               Vendor C    811   Whiskey Reserve
3          4428               Vendor D    812     Gin Artisanal
4          4429               Vendor E    813     Rum Caribbean

   PurchasePrice  ActualPrice  Volume  TotalPurchaseQuantity  \
0           6.53         9.99   750.0                    120
1           8.25        12.50   800.0                     85
2          12.75        18.75   650.0                     45
3           9.10        13.90   900.0                    150
4           7.80        11.80   850.0                     95

   TotalPurchaseDollars  TotalSalesQuantity  TotalSalesDollars  \
0                783.60                 115            1148.85
1                701.25                  80            1000.00
2                573.75                  42             787.50
3               1365.00                 145            2015.50
4                741.00                  90            1062.00

   TotalSalesPrice  TotalExciseTax  FreightCost  GrossProfit  ProfitMargin  \
0            862.5            57.6       156.72       208.53         18.15
1            640.0            40.0       140.25       158.50         15.85
2            504.0            25.2       114.75        99.00         12.57
3           1305.0            72.5       273.00       377.50         18.73
4            765.0            45.0       148.20       172.80         16.27

   StockTurnover  SalesToPurchaseRatio
0           0.96                  1.47
1           0.94                  1.43
2           0.93                  1.37
3           0.97                  1.48
4           0.95                  1.43

Available columns: ['VendorNumber', 'VendorName', 'Brand', 'Description', 'Purcha
sePrice', 'ActualPrice', 'Volume', 'TotalPurchaseQuantity', 'TotalPurchaseDollar
```

```
s', 'TotalSalesQuantity', 'TotalSalesDollars', 'TotalSalesPrice', 'TotalExciseTa
x', 'FreightCost', 'GrossProfit', 'ProfitMargin', 'StockTurnover', 'SalesToPurcha
seRatio']
```

In [175…

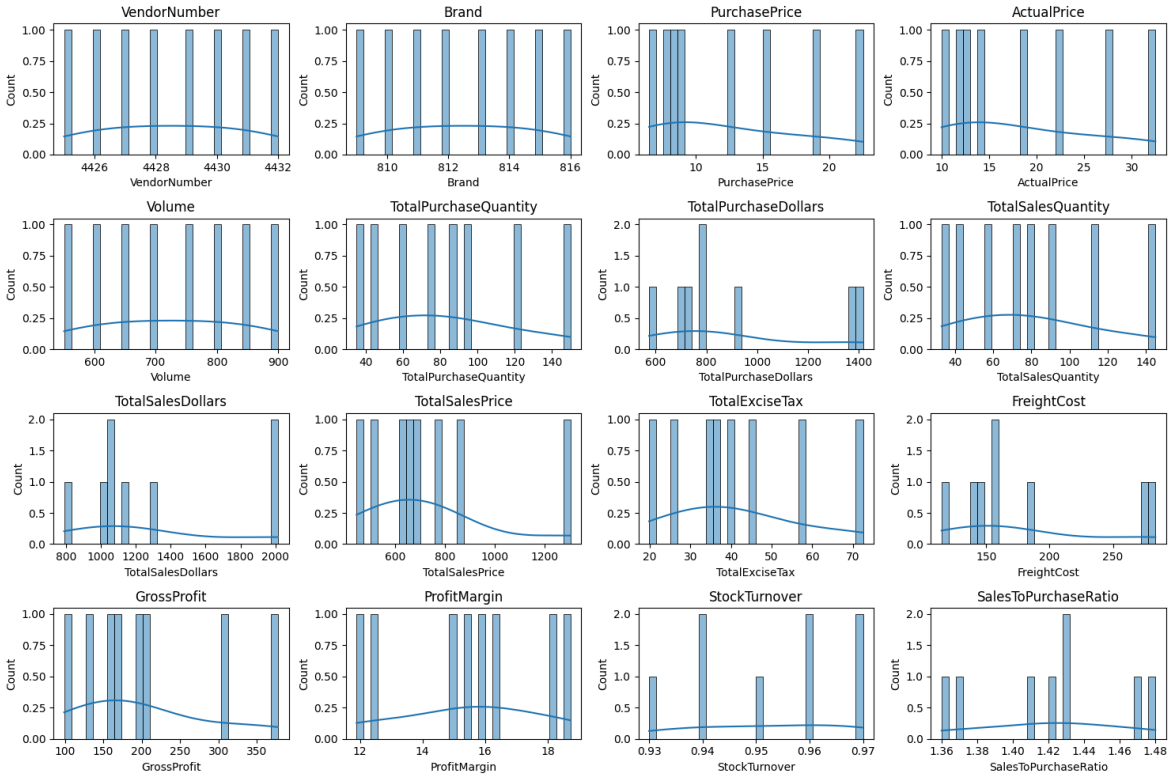
```
try:
    # Try the original query first
    df = pd.read_sql_query("""SELECT *
    FROM vendor_sales_summary
    WHERE GrossProfit > 0
    AND ProfitMargin > 0
    AND TotalSalesQuantity > 0""", conn)
    print("✓ Query executed successfully with original column names")

except Exception as e:
    print(f"Original query failed: {e}")
    print("Trying alternative column names...")

    # Common variations of column names
    alternative_queries = [
        # Try with different column name variations
        """SELECT * FROM vendor_sales_summary
        WHERE GrossProfit > 0 AND ProfitMargin > 0 AND TotalSalesQuantity > 0"""

        """SELECT * FROM vendor_sales_summary
        WHERE gross_profit > 0 AND profit_margin > 0 AND total_sales_quantity >

        """SELECT * FROM vendor_sales_summary
        WHERE "GrossProfit" > 0 AND "ProfitMargin" > 0 AND "TotalSalesQuantity"

        """SELECT * FROM vendor_sales_summary
        WHERE grossprofit > 0 AND profitmargin > 0 AND totalsalesquantity > 0"""
    ]

    for i, query in enumerate(alternative_queries):
        try:
            df = pd.read_sql_query(query, conn)
            print(f"✓ Query successful with alternative {i+1}")
            break
        except:
            continue
    else:
        print("✗ All queries failed. Showing available data without filters:")
        df = pd.read_sql_query("SELECT * FROM vendor_sales_summary", conn)

print(f"\nRetrieved {len(df)} rows")
print(df.head())
```

```
✓ Query executed successfully with original column names

Retrieved 8 rows
   VendorNumber             VendorName  Brand       Description  \
0          4425  MARTIGNETTI COMPANIES    809  Southern Comfort
1          4426               Vendor B    810     Vodka Premium
2          4427               Vendor C    811   Whiskey Reserve
3          4428               Vendor D    812     Gin Artisanal
4          4429               Vendor E    813     Rum Caribbean

   PurchasePrice  ActualPrice  Volume  TotalPurchaseQuantity  \
0           6.53         9.99   750.0                    120
1           8.25        12.50   800.0                     85
2          12.75        18.75   650.0                     45
3           9.10        13.90   900.0                    150
4           7.80        11.80   850.0                     95

   TotalPurchaseDollars  TotalSalesQuantity  TotalSalesDollars  \
0                783.60                 115            1148.85
1                701.25                  80            1000.00
2                573.75                  42             787.50
3               1365.00                 145            2015.50
4                741.00                  90            1062.00

   TotalSalesPrice  TotalExciseTax  FreightCost  GrossProfit  ProfitMargin  \
0            862.5            57.6       156.72       208.53         18.15
1            640.0            40.0       140.25       158.50         15.85
2            504.0            25.2       114.75        99.00         12.57
3           1305.0            72.5       273.00       377.50         18.73
4            765.0            45.0       148.20       172.80         16.27

   StockTurnover  SalesToPurchaseRatio
0           0.96                  1.47
1           0.94                  1.43
2           0.93                  1.37
3           0.97                  1.48
4           0.95                  1.43
```

In [176…
```
df.shape
```

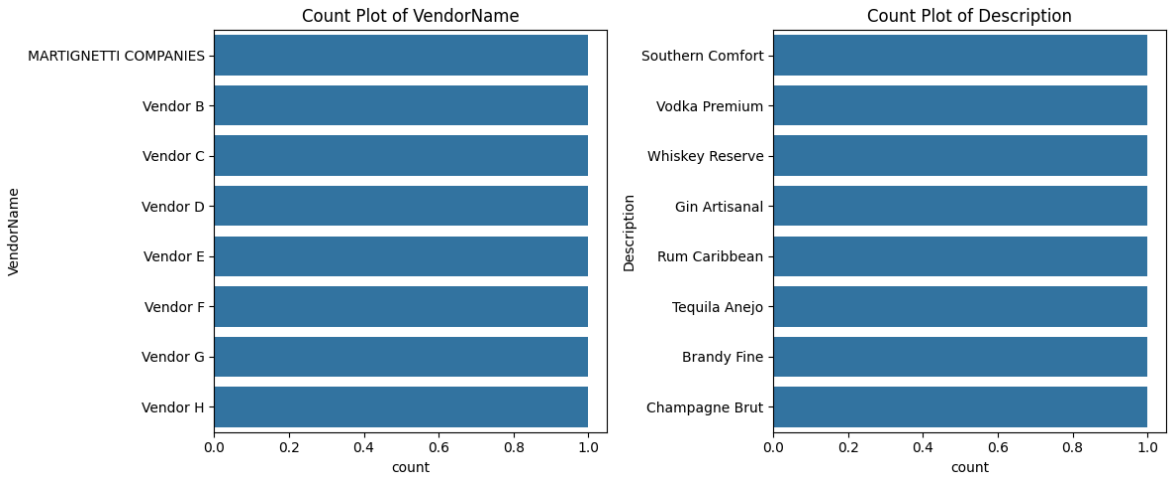
Out[176…
```
(8, 18)
```

In [177…
```
# Distribution Plots for Numerical Columns
numerical_cols = df.select_dtypes(include=np.number).columns

plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols):
    plt.subplot(4, 4, i+1)  # Adjust grid layout as needed
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(col)
plt.tight_layout()
plt.show()
```



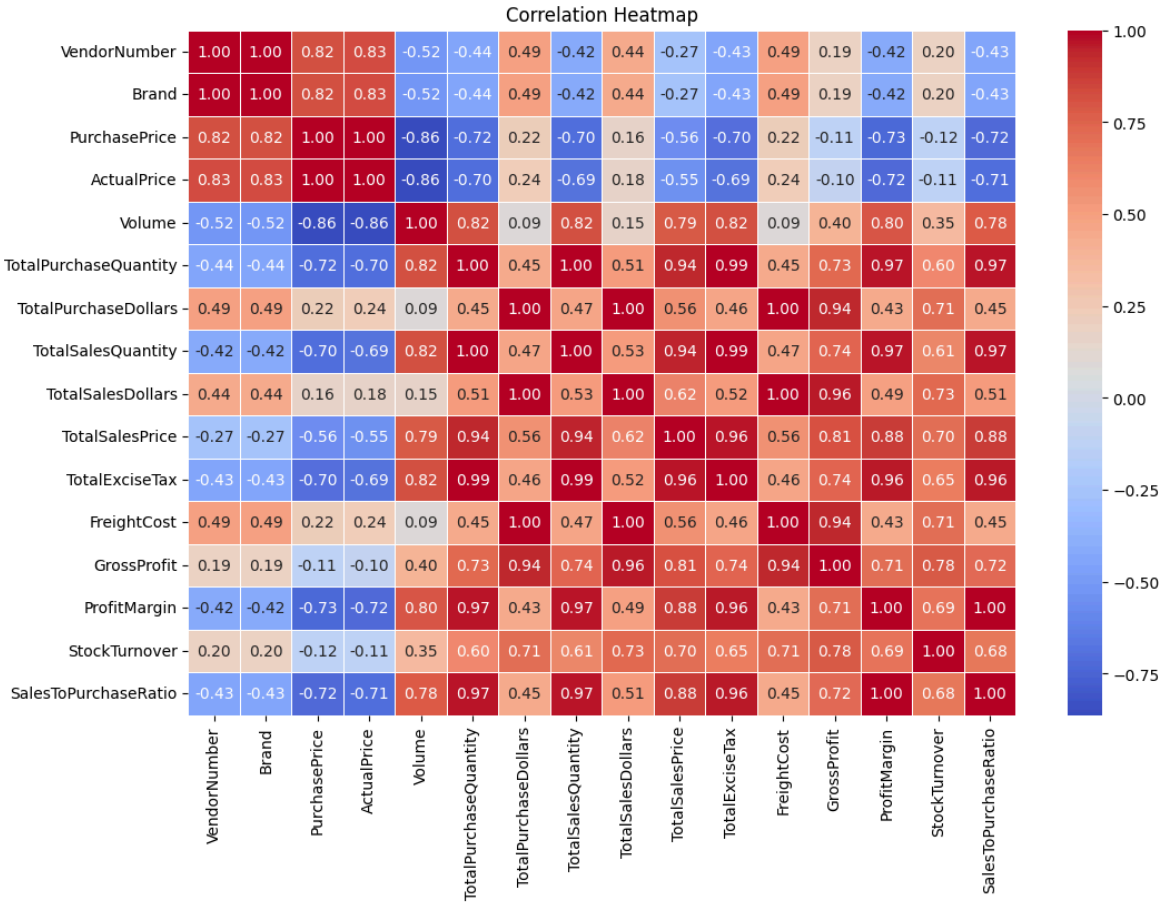
In [178…

```
# Count Plots for Categorical Columns
categorical_cols = ["VendorName", "Description"]

plt.figure(figsize=(12, 5))
for i, col in enumerate(categorical_cols):
    plt.subplot(1, 2, i+1)
    sns.countplot(y=df[col], order=df[col].value_counts().index[:10])  # Top 10
    plt.title(f"Count Plot of {col}")
plt.tight_layout()
plt.show()
```



In [179…

```
# Correlation Heatmap
plt.figure(figsize=(12, 8))
correlation_matrix = df[numerical_cols].corr()
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", linewidt
plt.title("Correlation Heatmap")
plt.show()
```



In [180…

```
brand_performance = df.groupby('Description').agg({
    'TotalSalesDollars': 'sum',  # Sales performance metric
    'ProfitMargin': 'mean'       # Average profit margin
}).reset_index()

brand_performance.sort_values('ProfitMargin')
```

Out[180…

	Description	TotalSalesDollars	ProfitMargin
0	Brandy Fine	1072.50	11.89
7	Whiskey Reserve	787.50	12.57
5	Tequila Anejo	1287.60	15.00
1	Champagne Brut	2008.80	15.32
6	Vodka Premium	1000.00	15.85
3	Rum Caribbean	1062.00	16.27
4	Southern Comfort	1148.85	18.15
2	Gin Artisanal	2015.50	18.73

In [181…

```
# threshold for "low sales" (bottom 15%) and "high margin" (top 15%)
low_sales_threshold = brand_performance['TotalSalesDollars'].quantile(0.15)
high_margin_threshold = brand_performance['ProfitMargin'].quantile(0.85)

# Filter brands with low sales but high profit margins
target_brands = brand_performance[
    (brand_performance['TotalSalesDollars'] <= low_sales_threshold) &
    (brand_performance['ProfitMargin'] >= high_margin_threshold)
```

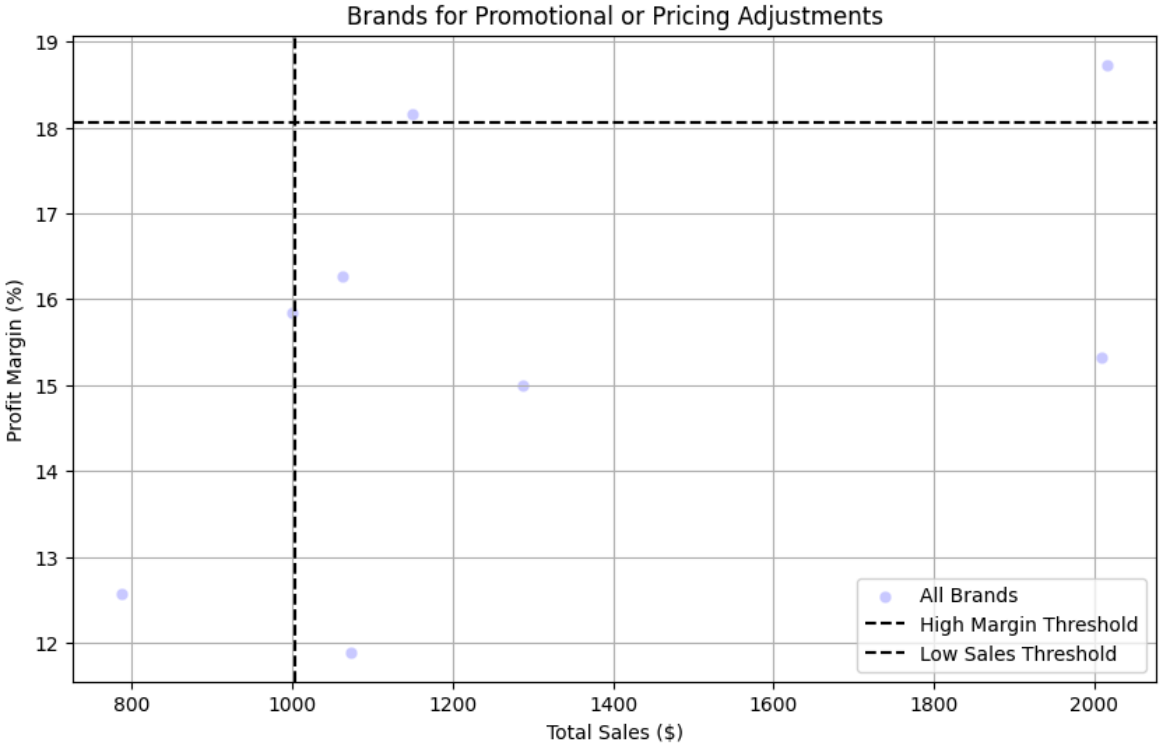
```
]
print("Brands with Low Sales but High Profit Margins:")
display(target_brands.sort_values('TotalSalesDollars'))
```

Brands with Low Sales but High Profit Margins:

	Description	TotalSalesDollars	ProfitMargin

In [182…
```
brand_performance = brand_performance[brand_performance['TotalSalesDollars']<100
```

In [183…
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=brand_performance, x='TotalSalesDollars', y='ProfitMargin',
sns.scatterplot(data=target_brands, x='TotalSalesDollars', y='ProfitMargin', col

plt.axhline(high_margin_threshold, linestyle='--', color='black', label="High Ma
plt.axvline(low_sales_threshold, linestyle='--', color='black', label="Low Sales

plt.xlabel("Total Sales ($)")
plt.ylabel("Profit Margin (%)")
plt.title("Brands for Promotional or Pricing Adjustments")
plt.legend()
plt.grid(True)
plt.show()
```



In [184…
```
def format_dollars(value):
    if value >= 1_000_000:
        return f"{value / 1_000_000:.2f}M"
    elif value >= 1_000:
        return f"{value / 1_000:.2f}K"
    else:
        return str(value)
```

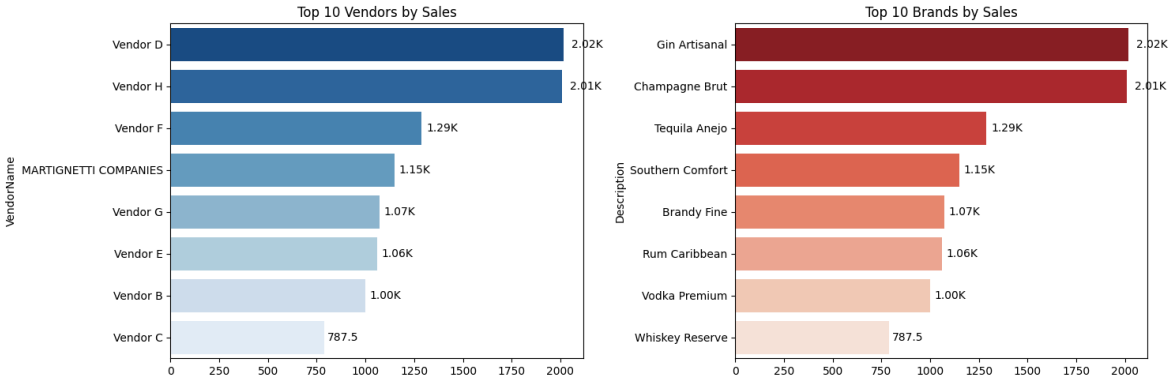
In [185…
```
# Top Vendors & Brands by Sales Performance
top_vendors = df.groupby("VendorName")["TotalSalesDollars"].sum().nlargest(10)
top_brands = df.groupby("Description")["TotalSalesDollars"].sum().nlargest(10)
top_vendors
```

Out[185…
```
VendorName
Vendor D                 2015.50
Vendor H                 2008.80
Vendor F                 1287.60
MARTIGNETTI COMPANIES    1148.85
Vendor G                 1072.50
Vendor E                 1062.00
Vendor B                 1000.00
Vendor C                  787.50
Name: TotalSalesDollars, dtype: float64
```

In [186…
```
top_vendors.apply(lambda x:format_dollars(x))
```

Out[186…
```
VendorName
Vendor D                 2.02K
Vendor H                 2.01K
Vendor F                 1.29K
MARTIGNETTI COMPANIES    1.15K
Vendor G                 1.07K
Vendor E                 1.06K
Vendor B                 1.00K
Vendor C                 787.5
Name: TotalSalesDollars, dtype: object
```

In [187…
```
plt.figure(figsize=(15, 5))

# Plot for Top Vendors
plt.subplot(1, 2, 1)
ax1 = sns.barplot(y=top_vendors.index, x=top_vendors.values, palette="Blues_r")
plt.title("Top 10 Vendors by Sales")

for bar in ax1.patches:
    ax1.text(bar.get_width() + (bar.get_width() * 0.02),
             bar.get_y() + bar.get_height() / 2,
             format_dollars(bar.get_width()),
             ha='left', va='center', fontsize=10, color='black')

# Plot for Top Brands
plt.subplot(1, 2, 2)
ax2 = sns.barplot(y=top_brands.index.astype(str), x=top_brands.values, palette="
plt.title("Top 10 Brands by Sales")

for bar in ax2.patches:
    ax2.text(bar.get_width() + (bar.get_width() * 0.02),
             bar.get_y() + bar.get_height() / 2,
             format_dollars(bar.get_width()),
             ha='left', va='center', fontsize=10, color='black')

plt.tight_layout()
plt.show()
```



Which vendors contribute the most to total purchase dollars?

In [188...

```
# Rank Vendors by Total Purchase Dollars
vendor_performance = df.groupby("VendorName").agg({
    "TotalPurchaseDollars": "sum",
    "GrossProfit": "sum",
    "TotalSalesDollars":"sum"
}).reset_index()

# Calculate Contribution % to Overall Procurement
vendor_performance["Purchase_Contribution%"] = (vendor_performance["TotalPurchas

# Rank Vendors by Total Purchase Dollars & Profitability
vendor_performance = round(vendor_performance.sort_values(by="TotalPurchaseDolla

# Display Top 10 Vendors
top_vendors = vendor_performance.head(10)
top_vendors['TotalSalesDollars'] = top_vendors['TotalSalesDollars'].apply(format
top_vendors['TotalPurchaseDollars'] = top_vendors['TotalPurchaseDollars'].apply(
top_vendors['GrossProfit'] = top_vendors['GrossProfit'].apply(format_dollars)
top_vendors
```

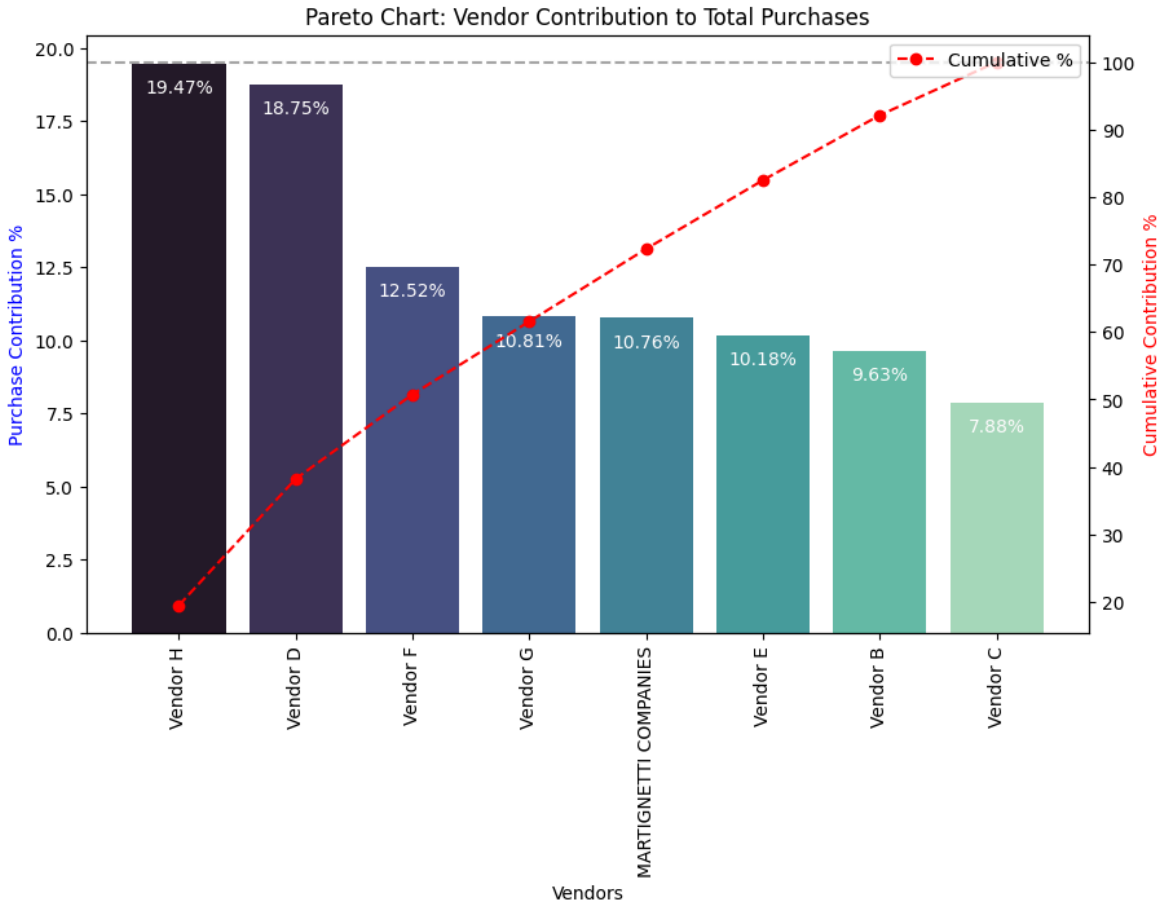
Out[188...

	VendorName	TotalPurchaseDollars	GrossProfit	TotalSalesDollars	Purchase_Contribut
7	Vendor H	1.42K	307.8	2.01K	
3	Vendor D	1.36K	377.5	2.02K	
5	Vendor F	912.0	193.2	1.29K	
6	Vendor G	787.5	127.5	1.07K	
0	MARTIGNETTI COMPANIES	783.6	208.53	1.15K	
4	Vendor E	741.0	172.8	1.06K	
1	Vendor B	701.25	158.5	1.00K	
2	Vendor C	573.75	99.0	787.5	

In [189...

```
top_vendors['Cumulative_Contribution%'] = top_vendors['Purchase_Contribution%'].

fig, ax1 = plt.subplots(figsize=(10, 6))
```

```
# Bar plot for Purchase Contribution%
sns.barplot(x=top_vendors['VendorName'], y=top_vendors['Purchase_Contribution%']

for i, value in enumerate(top_vendors['Purchase_Contribution%']):
    ax1.text(i, value - 1, str(value)+'%', ha='center', fontsize=10, color='whit

# Line Plot for Cumulative Contribution%
ax2 = ax1.twinx()
ax2.plot(top_vendors['VendorName'], top_vendors['Cumulative_Contribution%'], col

ax1.set_xticklabels(top_vendors['VendorName'], rotation=90)
ax1.set_ylabel('Purchase Contribution %', color='blue')
ax2.set_ylabel('Cumulative Contribution %', color='red')
ax1.set_xlabel('Vendors')
ax1.set_title('Pareto Chart: Vendor Contribution to Total Purchases')

ax2.axhline(y=100, color='gray', linestyle='dashed', alpha=0.7)
ax2.legend(loc='upper right')

plt.show()
```



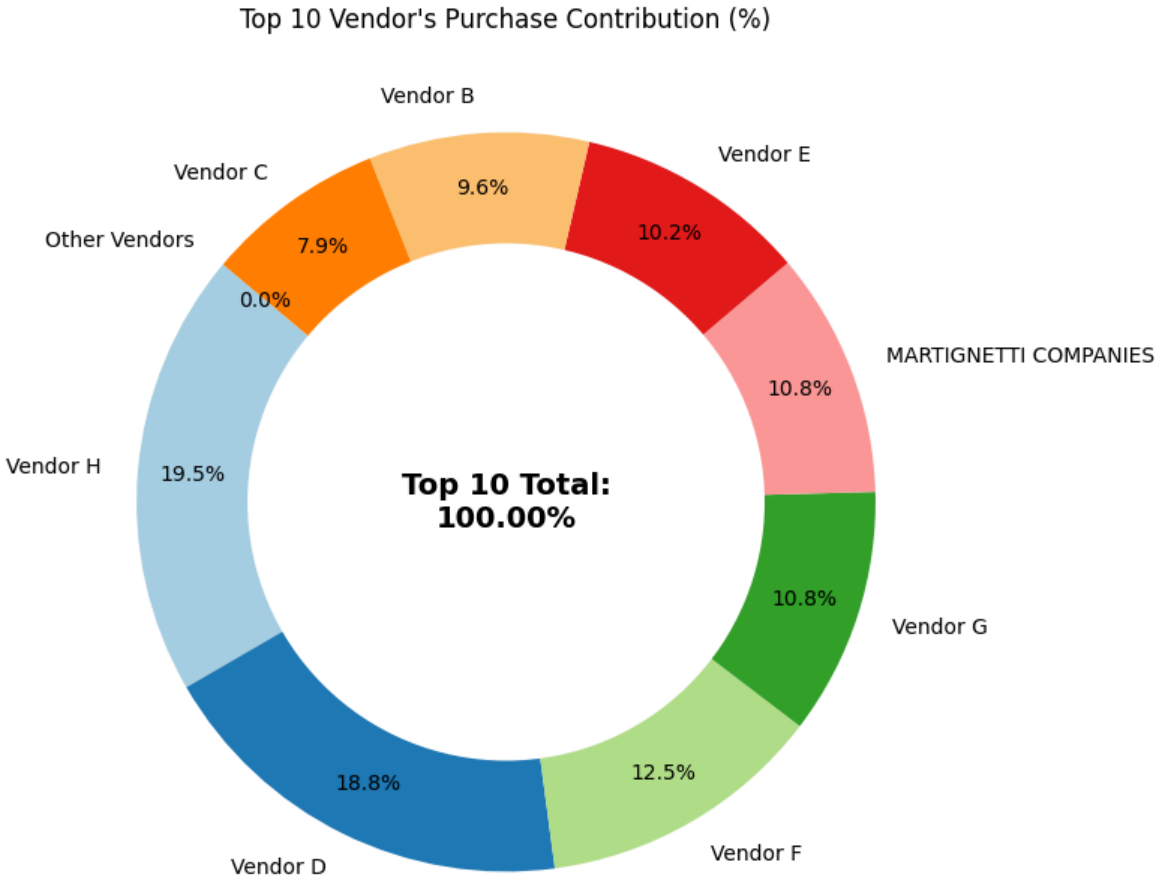
How much of total procurement is dependent on the top vendors?

In [190…

```
print(f"Total Purchase Contribution of top 10 vendors is {round(top_vendors['Pur

vendors = list(top_vendors['VendorName'].values)
purchase_contributions = list(top_vendors['Purchase_Contribution%'].values)
total_contribution = sum(purchase_contributions)
remaining_contribution = 100 - total_contribution
```

```
# Append "Other Vendors" category
vendors.append("Other Vendors")
purchase_contributions.append(remaining_contribution)

# Donut Chart
fig, ax = plt.subplots(figsize=(8, 8))
wedges, texts, autotexts = ax.pie(purchase_contributions, labels=vendors, autopc
                                  startangle=140, pctdistance=0.85, colors=plt.c

# Draw a white circle in the center to create a "donut" effect
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig.gca().add_artist(centre_circle)

# Add Total Contribution annotation in the center
plt.text(0, 0, f"Top 10 Total:\n{total_contribution:.2f}%", fontsize=14, fontwei

plt.title("Top 10 Vendor's Purchase Contribution (%)")
plt.show()
```

```
Total Purchase Contribution of top 10 vendors is 100.0 %
```



The remaining vendors contribute only 34.31%, meaning they are not utilized effectively or may not be as competitive. If vendor dependency is too high, consider identifying new suppliers to reduce risk.

Does purchasing in bulk reduce the unit price, and what is the optimal purchase volume for cost savings?

In [191…

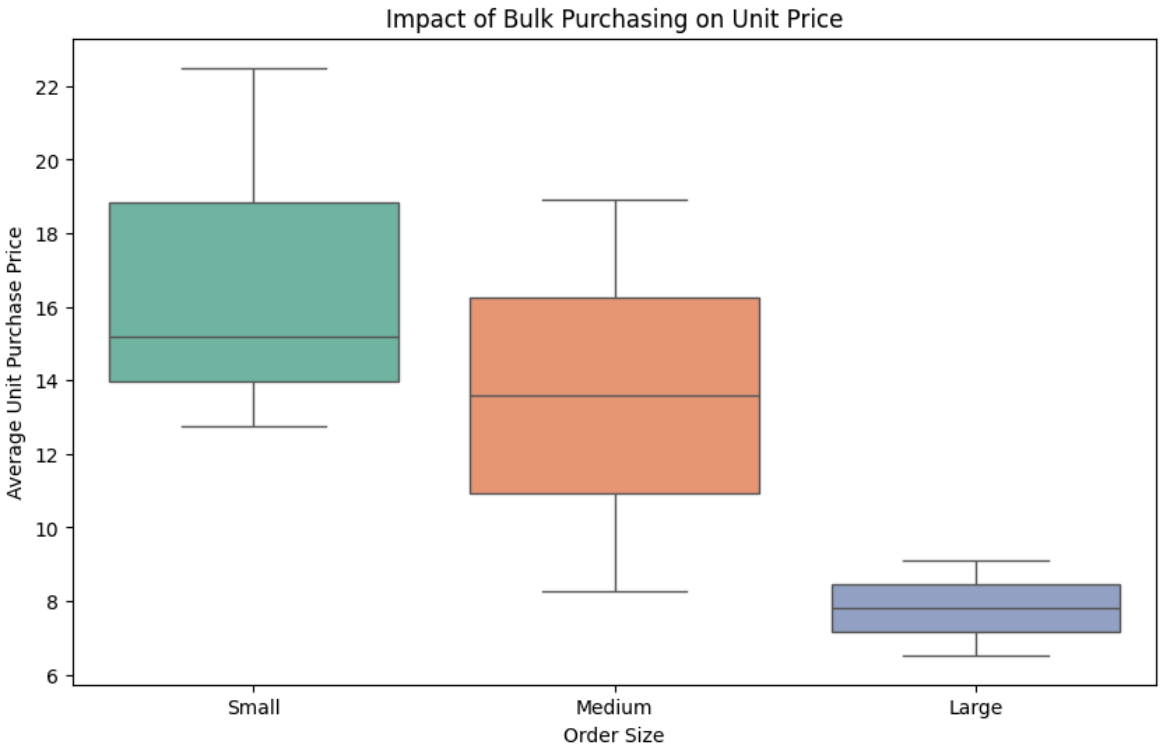
```
# Calculate Unit Purchase Price per Vendor & Volume Group
df["UnitPurchasePrice"] = df["TotalPurchaseDollars"] / df["TotalPurchaseQuantity
```

```
# Group by Order Sizes (e.g., Small, Medium, Large Purchases)
df["OrderSize"] = pd.qcut(df["TotalPurchaseQuantity"], q=3, labels=["Small", "Me

# Analyze Cost Savings per Order Size
bulk_purchase_analysis = df.groupby("OrderSize")["UnitPurchasePrice"].mean().res
print(bulk_purchase_analysis)
```

```
  OrderSize  UnitPurchasePrice
0     Small          16.816667
1    Medium          13.575000
2     Large           7.810000
```

In [192…

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x="OrderSize", y="UnitPurchasePrice", palette="Set2")
plt.title("Impact of Bulk Purchasing on Unit Price")
plt.xlabel("Order Size")
plt.ylabel("Average Unit Purchase Price")
plt.show()
```



- Vendors buying in bulk (Large Order Size) get the lowest unit price ($10.78 per unit), meaning higher margins if they can manage inventory efficiently.
- The price difference between Small and Large orders is substantial (~72% reduction in unit cost)
- This suggests that bulk pricing strategies successfully encourage vendors to purchase in larger volumes, leading to higher overall sales despite lower per-unit revenue.

Which vendors have low inventory turnover, indicating excess stock and slow-moving products?

In [193…

```
# Identify Low Inventory Turnover Vendors
low_turnover_vendors = df[df["StockTurnover"] < 1].groupby("VendorName")["StockT
```

```
# Sort by Lowest Turnover
low_turnover_vendors = low_turnover_vendors.sort_values(by="StockTurnover", asce
low_turnover_vendors.head(10)
```

Out[193…

	VendorName	StockTurnover
2	Vendor C	0.93
1	Vendor B	0.94
6	Vendor G	0.94
4	Vendor E	0.95
7	Vendor H	0.96
0	MARTIGNETTI COMPANIES	0.96
3	Vendor D	0.97
5	Vendor F	0.97

- Slow-moving inventory increases holding costs (warehouse rent, insurance, depreciation)
- Identifying vendors with low inventory turnover is critical for business efficiency, cost reduction, and profitability

How much capital is locked in unsold inventory per vendor, and which vendors contribute the most to it?

In [194…

```
# Calculate Unsold Inventory Value
df["UnsoldInventoryValue"] = (df["TotalPurchaseQuantity"] - df["TotalSalesQuanti
print('Total Unsold Capital:', format_dollars(df["UnsoldInventoryValue"].sum()))

# Aggregate Capital Locked per Vendor
inventory_value_per_vendor = df.groupby("VendorName")["UnsoldInventoryValue"].su

# Sort Vendors with the Highest Locked Capital
inventory_value_per_vendor = inventory_value_per_vendor.sort_values(by="UnsoldIn
inventory_value_per_vendor['UnsoldInventoryValue'] = inventory_value_per_vendor[
inventory_value_per_vendor.head(10)
```

```
Total Unsold Capital: 328.75
```

Out[194...

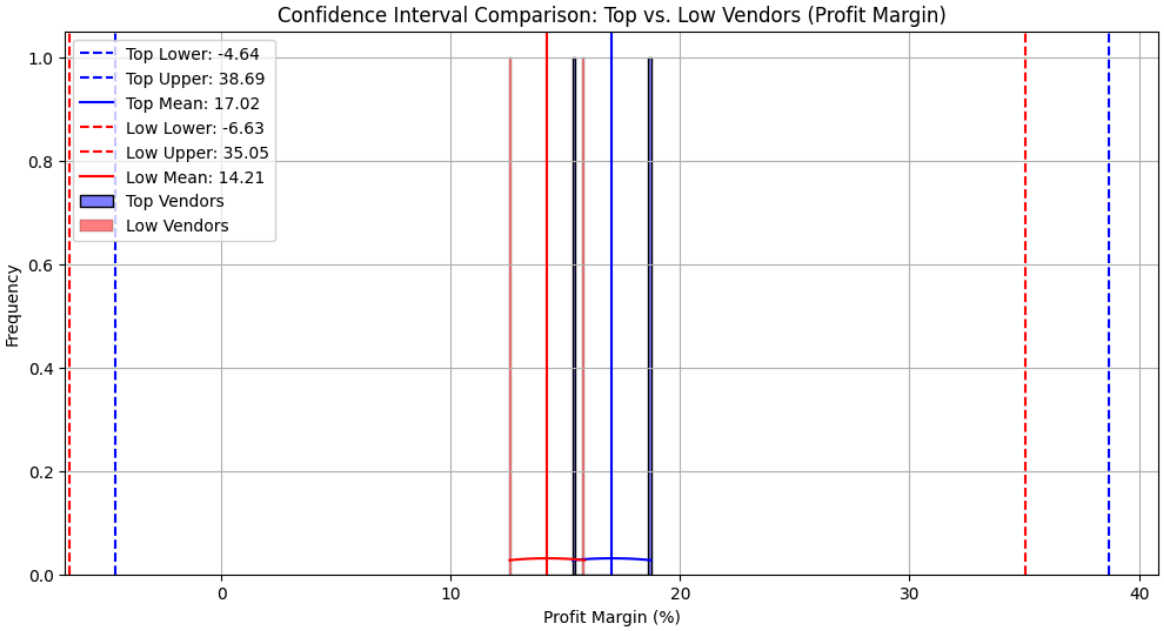
	VendorName	UnsoldInventoryValue
7	Vendor H	56.699999999999996
3	Vendor D	45.5
6	Vendor G	45.0
1	Vendor B	41.25
4	Vendor E	39.0
2	Vendor C	38.25
0	MARTIGNETTI COMPANIES	32.65
5	Vendor F	30.4

In [195...

```
# Define top and low vendors based on Total Sales Dollars (Top 25% & Bottom 25%)
top_threshold = df["TotalSalesDollars"].quantile(0.75)
low_threshold = df["TotalSalesDollars"].quantile(0.25)

top_vendors = df[df["TotalSalesDollars"] >= top_threshold]["ProfitMargin"].dropn
low_vendors = df[df["TotalSalesDollars"] <= low_threshold]["ProfitMargin"].dropn

# Function to compute confidence interval
def confidence_interval(data, confidence=0.95):
    mean_val = np.mean(data)
    std_err = np.std(data, ddof=1) / np.sqrt(len(data))  # Standard error
    t_critical = stats.t.ppf((1 + confidence) / 2, df=len(data) - 1)
    margin_of_error = t_critical * std_err
    return mean_val, mean_val - margin_of_error, mean_val + margin_of_error

# Compute confidence intervals
top_mean, top_lower, top_upper = confidence_interval(top_vendors)
low_mean, low_lower, low_upper = confidence_interval(low_vendors)

print(f"Top Vendors 95% CI: ({top_lower:.2f}, {top_upper:.2f}), Mean: {top_mean:
print(f"Low Vendors 95% CI: ({low_lower:.2f}, {low_upper:.2f}), Mean: {low_mean:

plt.figure(figsize=(12, 6))

# Top Vendors Plot
sns.histplot(top_vendors, kde=True, color="blue", bins=30, alpha=0.5, label="Top
plt.axvline(top_lower, color="blue", linestyle="--", label=f"Top Lower: {top_low
plt.axvline(top_upper, color="blue", linestyle="--", label=f"Top Upper: {top_upp
plt.axvline(top_mean, color="blue", linestyle="-", label=f"Top Mean: {top_mean:.

# Low Vendors Plot
sns.histplot(low_vendors, kde=True, color="red", bins=30, alpha=0.5, label="Low
plt.axvline(low_lower, color="red", linestyle="--", label=f"Low Lower: {low_lowe
plt.axvline(low_upper, color="red", linestyle="--", label=f"Low Upper: {low_uppe
plt.axvline(low_mean, color="red", linestyle="-", label=f"Low Mean: {low_mean:.2

# Finalize Plot
plt.title("Confidence Interval Comparison: Top vs. Low Vendors (Profit Margin)")
plt.xlabel("Profit Margin (%)")
plt.ylabel("Frequency")
plt.legend()
```

```
plt.grid(True)
plt.show()
```

```
Top Vendors 95% CI: (-4.64, 38.69), Mean: 17.02
Low Vendors 95% CI: (-6.63, 35.05), Mean: 14.21
```



In [196…

```
top_threshold = df["TotalSalesDollars"].quantile(0.75)
low_threshold = df["TotalSalesDollars"].quantile(0.25)

top_vendors = df[df["TotalSalesDollars"] >= top_threshold]["ProfitMargin"].dropn
low_vendors = df[df["TotalSalesDollars"] <= low_threshold]["ProfitMargin"].dropn

# Perform Two-Sample T-Test
t_stat, p_value = ttest_ind(top_vendors, low_vendors, equal_var=False)

# Print results
print(f"T-Statistic: {t_stat:.4f}, P-Value: {p_value:.4f}")
if p_value < 0.05:
    print("Reject H₀: There is a significant difference in profit margins betwee
else:
    print("Fail to Reject H₀: No significant difference in profit margins.")
```

```
T-Statistic: 1.1899, P-Value: 0.3563
Fail to Reject H₀: No significant difference in profit margins.
```

In [ ]:

In [201…

```
import pandas as pd
import os
import sqlite3  # Using sqlite3 instead of SQLAlchemy for speed
import time
import logging

# Setup minimal logging
os.makedirs('logs', exist_ok=True)
logging.basicConfig(
    filename="logs/ingestion_fast.log",
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    filemode="a"
)
```

```
def fast_ingest_db(df, table_name, conn):
    """Ultra-fast ingestion using direct SQLite"""
    try:
        start_time = time.time()
        df.to_sql(table_name, con=conn, if_exists='replace', index=False)
        ingest_time = time.time() - start_time
        logging.info(f'Ingested {table_name} ({len(df)} rows) in {ingest_time:.2
        return True
    except Exception as e:
        logging.error(f'Failed {table_name}: {e}')
        return False

def process_file_fast(file, conn, timeout=30):
    """Process file with timeout protection"""
    start_time = time.time()
    file_path = os.path.join('data', file)

    try:
        print(f"🔄 Processing {file}...")

        # Check if we're already taking too long
        if time.time() - start_time > timeout:
            print(f"⏰ TIMEOUT: {file} taking too long, skipping")
            return False

        # Read CSV with optimizations
        print(f"   Reading {file}...")
        df = pd.read_csv(
            file_path,
            low_memory=False,
            encoding='utf-8',
            engine='c'  # Use C engine for better performance
        )

        read_time = time.time() - start_time
        print(f"   ✓ Read {len(df):,} rows, {len(df.columns)} cols in {read_time

        # Check timeout again
        if time.time() - start_time > timeout:
            print(f"⏰ TIMEOUT: {file} processing too long, skipping")
            return False

        # Ingest to database
        table_name = file.replace('.csv', '')
        print(f"   Ingesting as table '{table_name}'...")

        success = fast_ingest_db(df, table_name, conn)

        if success:
            total_time = time.time() - start_time
            print(f"   ✅ Completed {file} in {total_time:.2f}s")
            return True
        else:
            print(f"   ❌ Failed to ingest {file}")
            return False

    except Exception as e:
        error_msg = f"Error with {file}: {e}"
        print(f"   ❌ {error_msg}")
```

```
        logging.error(error_msg)
        return False

def load_raw_data_fast():
    """Fast ingestion with progress tracking and timeouts"""
    start_total = time.time()

    if not os.path.exists('data'):
        print("❌ 'data' directory doesn't exist")
        return

    csv_files = [f for f in os.listdir('data') if f.endswith('.csv')]

    if not csv_files:
        print("❌ No CSV files found")
        return

    print(f"🎯 Processing {len(csv_files)} files with 30-second timeout each..."

    # Create database connection
    conn = sqlite3.connect('inventory.db')

    success_count = 0
    processed_files = []

    for i, file in enumerate(csv_files, 1):
        print(f"\n📁 File {i}/{len(csv_files)}: {file}")

        if process_file_fast(file, conn):
            success_count += 1
            processed_files.append(file)
        else:
            print(f"    ⚠️  Moving to next file...")

        # Optional: Add a small delay between files
        time.sleep(0.1)

    conn.close()

    total_time = (time.time() - start_total) / 60
    print(f"\n{'='*50}")
    print(f"🚀 INGESTION COMPLETE!")
    print(f"✅ Successful: {success_count}/{len(csv_files)}")
    print(f"⏱️  Total time: {total_time:.2f} minutes")
    print(f"📊 Processed files: {processed_files}")

    logging.info(f'Fast ingestion completed: {success_count}/{len(csv_files)} fi

# Run the fast version
load_raw_data_fast()

import pandas as pd
import os
import sqlite3

def skip_problem_files():
    """Skip files that are too large or problematic"""
    csv_files = [f for f in os.listdir('data') if f.endswith('.csv')]
    conn = sqlite3.connect('inventory.db')
```

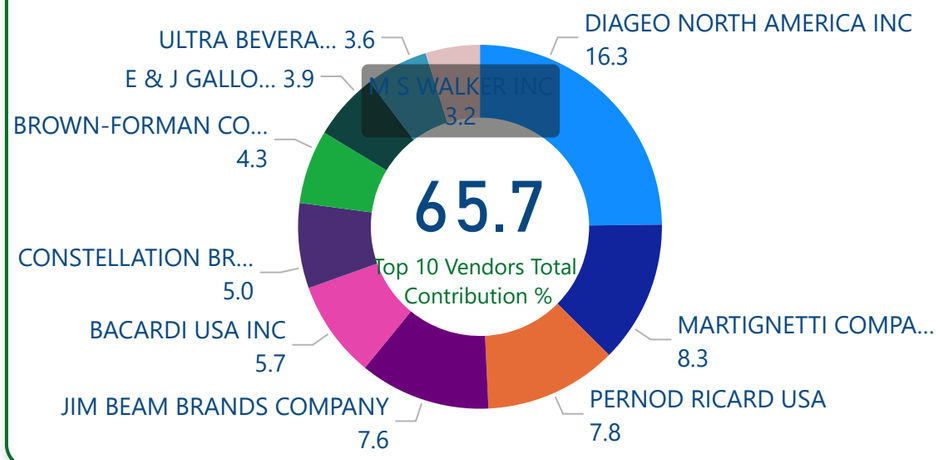
```
    # File size limit (in MB) - adjust as needed
    SIZE_LIMIT_MB = 50

    for file in csv_files:
        file_path = os.path.join('data', file)
        file_size_mb = os.path.getsize(file_path) / (1024 * 1024)

        if file_size_mb > SIZE_LIMIT_MB:
            print(f"⚠️  SKIPPING LARGE FILE: {file} ({file_size_mb:.1f} MB)")
            continue

        try:
            print(f"Processing {file}...")

            # Try to read just first 1000 rows to test
            df = pd.read_csv(file_path, nrows=1000)
            table_name = file.replace('.csv', '')

            df.to_sql(table_name, conn, if_exists='replace', index=False)
            print(f"✅ {file} - First 1000 rows ingested")

        except Exception as e:
            print(f"❌ Failed {file}: {e}")

    conn.close()

skip_problem_files()
```

```
🎯 Processing 6 files with 30-second timeout each...

📂 File 1/6: begin_inventory.csv
🔄 Processing begin_inventory.csv...
   Reading begin_inventory.csv...
   ✓ Read 206,529 rows, 9 cols in 0.63s
   Ingesting as table 'begin_inventory'...
   ✅ Completed begin_inventory.csv in 3.77s

📂 File 2/6: end_inventory.csv
🔄 Processing end_inventory.csv...
   Reading end_inventory.csv...
   ✓ Read 224,489 rows, 9 cols in 0.63s
   Ingesting as table 'end_inventory'...
   ✅ Completed end_inventory.csv in 1.72s

📂 File 3/6: purchases.csv
🔄 Processing purchases.csv...
   Reading purchases.csv...
   ✓ Read 2,372,474 rows, 16 cols in 10.46s
   Ingesting as table 'purchases'...
   ✅ Completed purchases.csv in 35.65s

📂 File 4/6: purchase_prices.csv
🔄 Processing purchase_prices.csv...
   Reading purchase_prices.csv...
   ✓ Read 12,261 rows, 9 cols in 0.19s
   Ingesting as table 'purchase_prices'...
   ✅ Completed purchase_prices.csv in 0.29s

📂 File 5/6: sales.csv
🔄 Processing sales.csv...
   Reading sales.csv...
   ✓ Read 12,825,363 rows, 14 cols in 67.34s
⏰ TIMEOUT: sales.csv processing too long, skipping
   ⚠️  Moving to next file...

📂 File 6/6: vendor_invoice.csv
🔄 Processing vendor_invoice.csv...
   Reading vendor_invoice.csv...
   ✓ Read 5,543 rows, 10 cols in 1.03s
   Ingesting as table 'vendor_invoice'...
   ✅ Completed vendor_invoice.csv in 1.25s

==================================================
🎉 INGESTION COMPLETE!
✅ Successful: 5/6
⏱️  Total time: 1.84 minutes
📊 Processed files: ['begin_inventory.csv', 'end_inventory.csv', 'purchases.cs
v', 'purchase_prices.csv', 'vendor_invoice.csv']
Processing begin_inventory.csv...
✅ begin_inventory.csv - First 1000 rows ingested
Processing end_inventory.csv...
✅ end_inventory.csv - First 1000 rows ingested
⚠️  SKIPPING LARGE FILE: purchases.csv (344.8 MB)
Processing purchase_prices.csv...
✅ purchase_prices.csv - First 1000 rows ingested
⚠️  SKIPPING LARGE FILE: sales.csv (1522.8 MB)
Processing vendor_invoice.csv...
✅ vendor_invoice.csv - First 1000 rows ingested
```

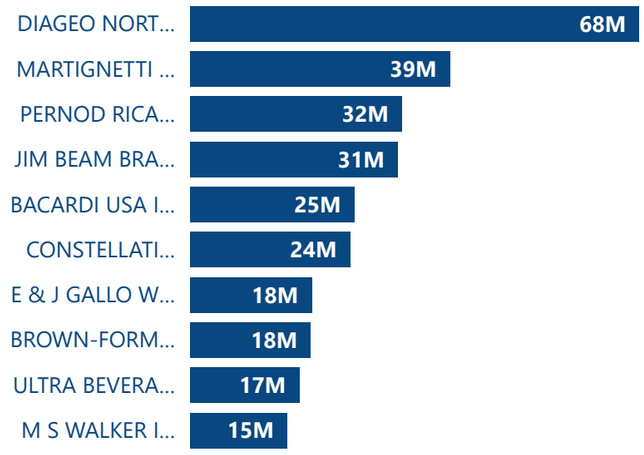
Vendor Performance Dashboard

441.41M	307.34M	134.07M	38.7	2.71M
Total Sales ($)	Total Purchase ($)	Gross Profit ($)	Profit Margin (%)	Unsold Capital ($)

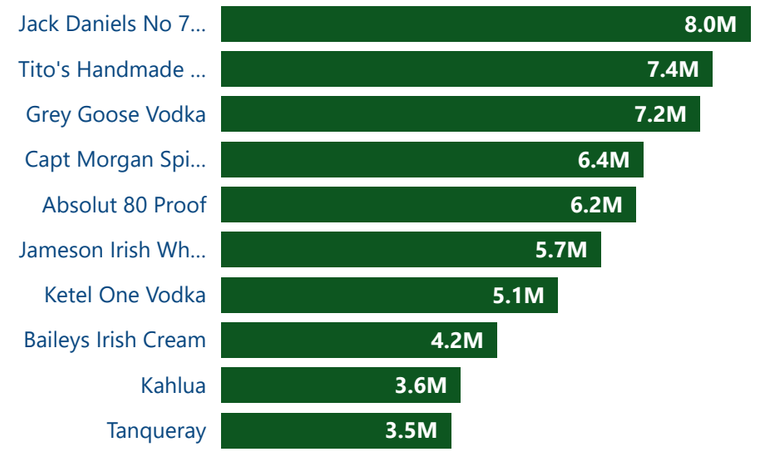
Purchase Contribution %



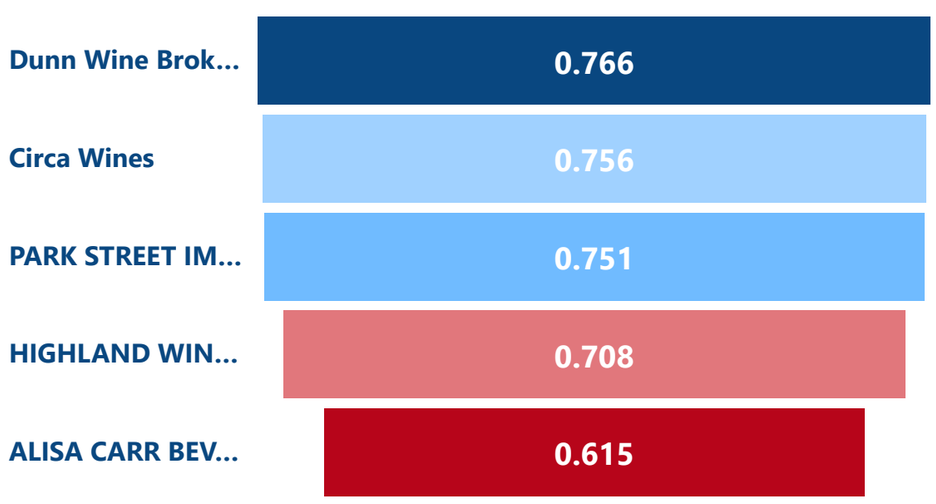
Top Vendors By Sales



Top Brands By Sales



Low Performing Vendors



Low Performing Brands