Vendor Performance Data Analytics Project Report

1. Introduction

This project focuses on analysing vendor and brand performance in the retail and wholesale industry to optimize profitability and operational efficiency. Effective inventory and sales management are critical to achieving these goals. The business problem addressed includes identifying underperforming brands, determining top vendors contributing to sales and profit, assessing the impact of bulk purchasing on unit costs, evaluating inventory turnover, and investigating profitability differences between high- and low-performing vendors. These insights aim to enable data-driven decisions for vendor management, pricing, promotional strategies, and inventory optimization.

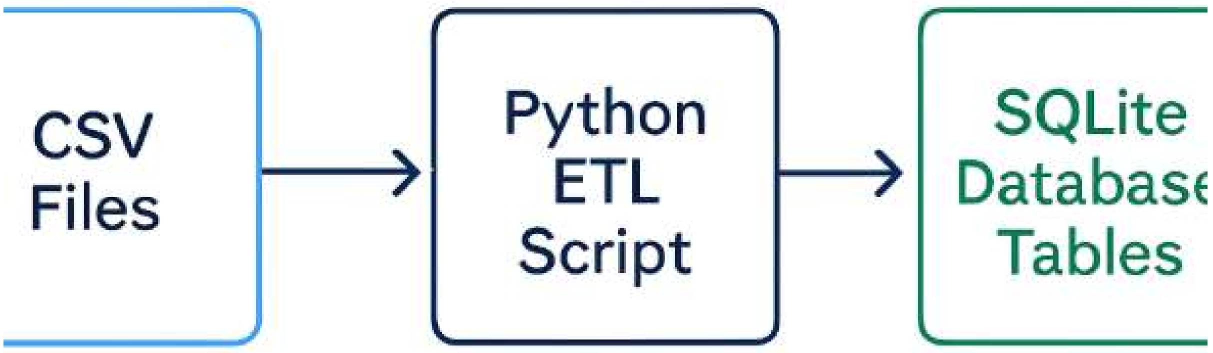
2. Data Pipeline and ETL Process

2.1 Data Ingestion

Raw data from multiple CSV files, including purchases, purchase prices, vendor invoices, and sales, was ingested into a centralized SQLite database (inventory.db). This automated ETL process was developed using Python with libraries such as pandas and SQLAlchemy. The pipeline iterates over CSV files, loads each into pandas DataFrames, and writes the data into corresponding SQLite tables, replacing existing data as needed. Detailed logging captures ingestion events, timestamps, and durations for monitoring and troubleshooting.

Code snippet:

```
def load_raw_data():
    for file in os.listdir('data'):
        if file.endswith('.csv'):
            df = pd.read_csv(f'data/{file}')
            logging.info(f'Ingesting {file} into database')
            df.to_sql(file[:-4], con=engine, if_exists='replace', index=False)
    logging.info('Ingestion complete')
```



2.2 Data Cleaning and Integration

Data was integrated by joining key tables (purchases, purchase_prices, vendor_invoice, sales) within SQLite using SQL queries and Common Table Expressions (CTEs). The aggregated summary table, vendor_sales_summary, combines purchase quantities and dollars, sales metrics, freight costs, and pricing details at vendor-brand granularity.

Post-extraction cleaning in Python included:

- Converting relevant columns to correct data types (e.g., floats)
- Filling missing numeric values with zeros
- Stripping whitespace from text fields (VendorName, Description)
- Calculating metrics: Gross Profit, Profit Margin (%), Stock Turnover Ratio, Sales-to-Purchase Ratio

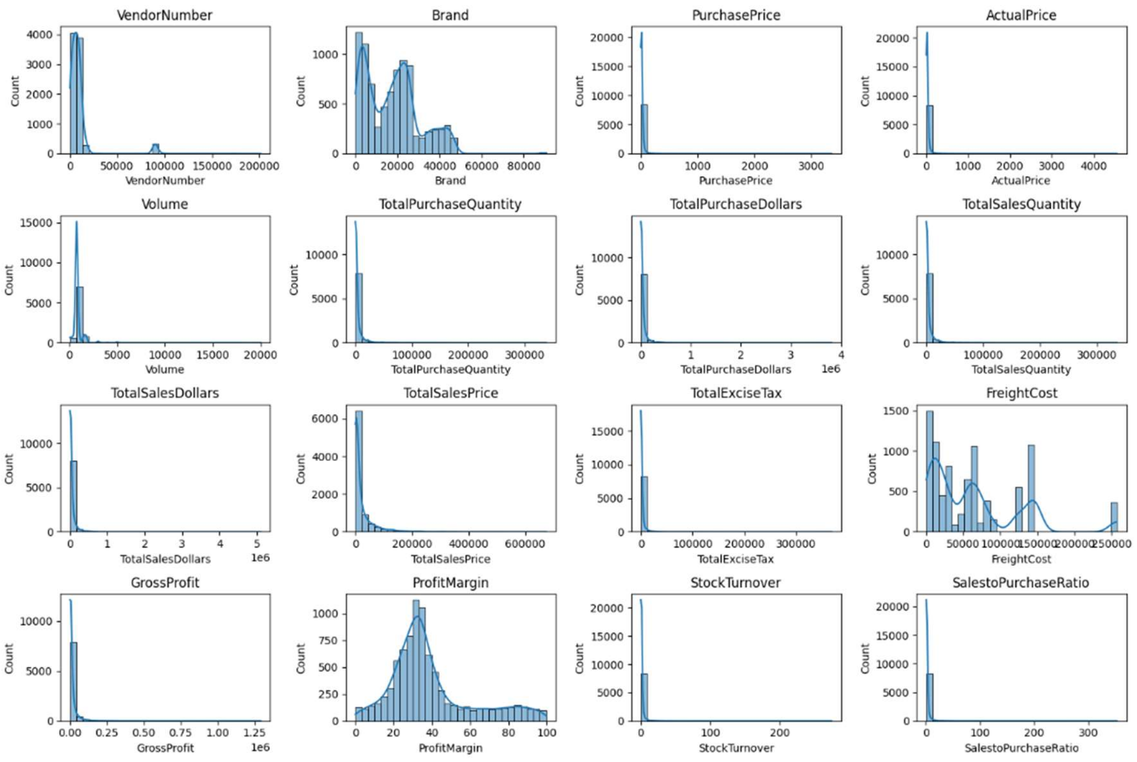
This cleansed analytical table supports efficient querying and reporting.

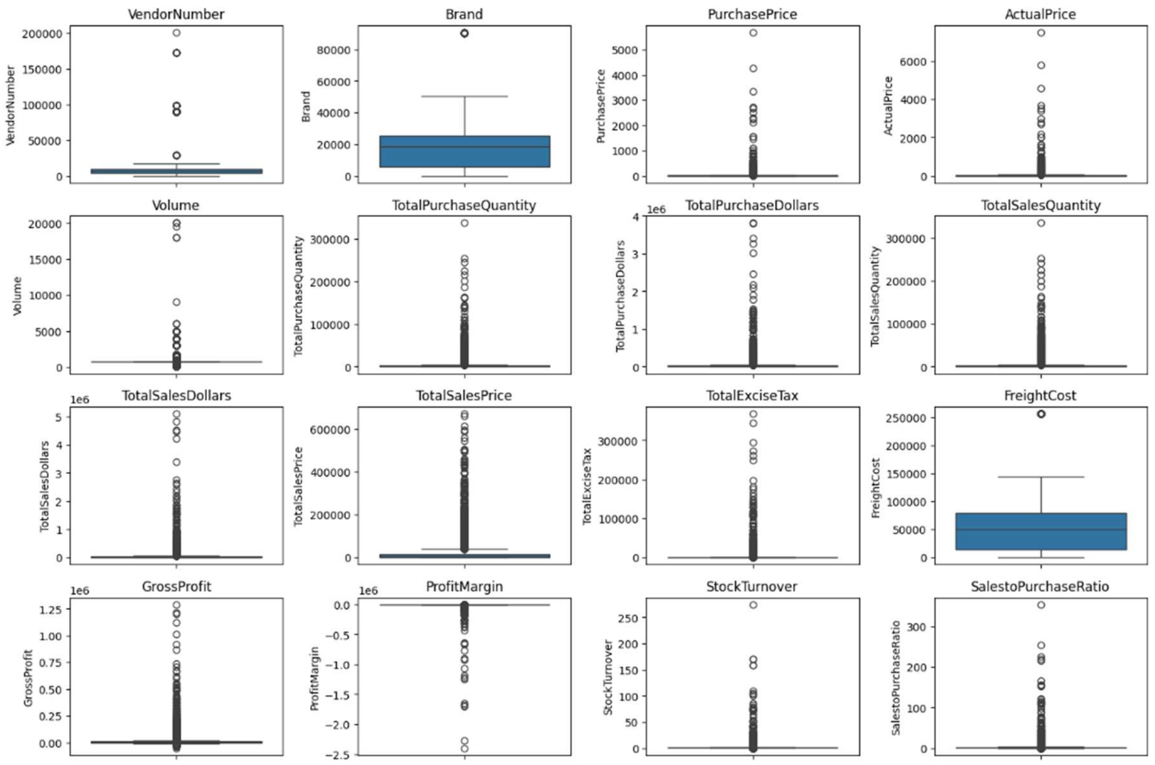
3. Exploratory Data Analysis (EDA)

3.1 Statistical Summaries & Outlier Detection

Exploratory analysis showed:

- Presence of negative and zero values in profit and sales indicators signaling loss-making or slow-moving products.
- Distributions indicated skewness and outliers especially in prices and freight costs.
- Filtering criteria (GrossProfit > 0, ProfitMargin > 0, TotalSalesQuantity > 0) were applied to focus on meaningful performance data.

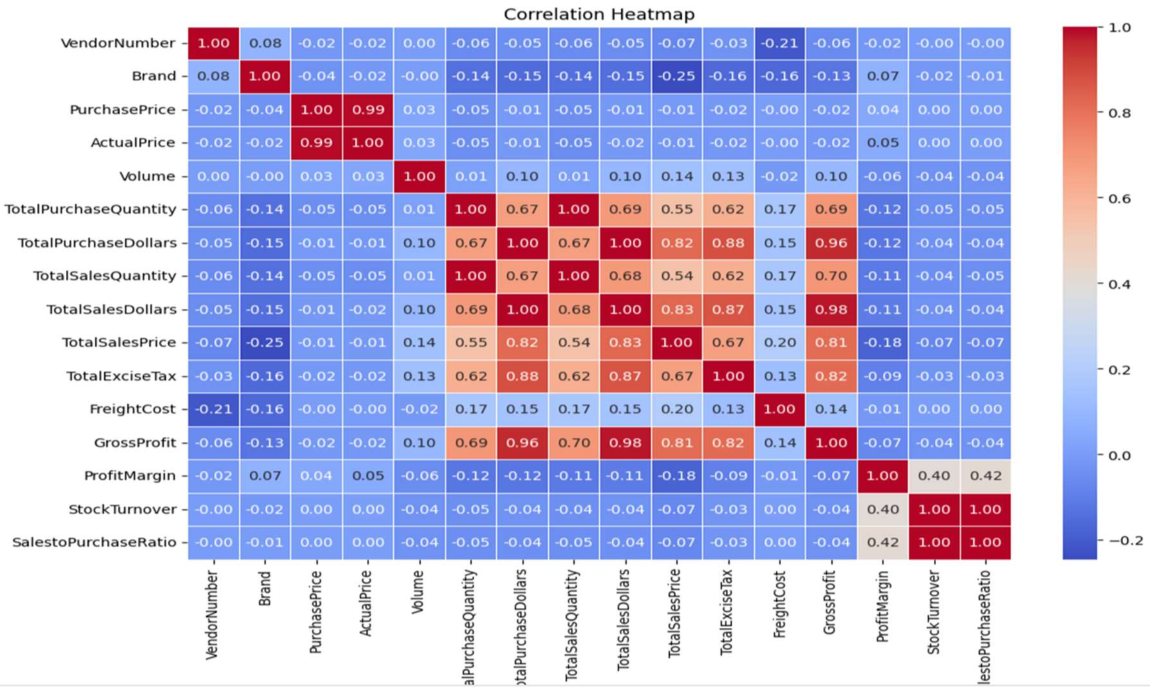




3.2 Correlation and Categorical Analysis

Correlation analysis revealed:

- Strong positive correlation between purchase and sales quantities.
- Weak or negative correlations between purchase prices and profit/sales, indicating price alone is not the driver of revenue or margin.
- Vendor and brand frequency counts identified top-performing entities in terms of transaction volume.



	VendorNumber	Brand	PurchasePrice	ActualPrice	Volume	TotalPurchaseQuantity	TotalPurchaseDollars	TotalSalesQuantity	TotalSalesDollars	TotalSalesPrice	TotalExciseTax	FreightCost	GrossProfit	ProfitMargin	StockTurnover	SalestoPurchaseRatio
VendorNumber	1.00	0.08	-0.02	-0.02	0.00	-0.06	-0.05	-0.06	-0.05	-0.07	-0.03	-0.21	-0.06	-0.02	-0.00	-0.00
Brand	0.08	1.00	-0.04	-0.02	-0.00	-0.14	-0.15	-0.14	-0.15	-0.25	-0.16	-0.16	-0.13	0.07	-0.02	-0.01
PurchasePrice	-0.02	-0.04	1.00	0.99	0.03	-0.05	-0.01	-0.05	-0.01	-0.01	-0.02	-0.00	-0.02	0.04	0.00	0.00
ActualPrice	-0.02	-0.02	0.99	1.00	0.03	-0.05	-0.01	-0.05	-0.02	-0.01	-0.02	-0.00	-0.02	0.05	0.00	0.00
Volume	0.00	-0.00	0.03	0.03	1.00	0.01	0.10	0.01	0.10	0.14	0.13	-0.02	0.10	-0.06	-0.04	-0.04
TotalPurchaseQuantity	-0.06	-0.14	-0.05	-0.05	0.01	1.00	0.67	1.00	0.69	0.55	0.62	0.17	0.69	-0.12	-0.05	-0.05
TotalPurchaseDollars	-0.05	-0.15	-0.01	-0.01	0.10	0.67	1.00	0.67	1.00	0.82	0.88	0.15	0.96	-0.12	-0.04	-0.04
TotalSalesQuantity	-0.06	-0.14	-0.05	-0.05	0.01	1.00	0.67	1.00	0.68	0.54	0.62	0.17	0.70	-0.11	-0.04	-0.05
TotalSalesDollars	-0.05	-0.15	-0.01	-0.02	0.10	0.69	1.00	0.68	1.00	0.83	0.87	0.15	0.98	-0.11	-0.04	-0.04
TotalSalesPrice	-0.07	-0.25	-0.01	-0.01	0.14	0.55	0.82	0.54	0.83	1.00	0.67	0.20	0.81	-0.18	-0.07	-0.07
TotalExciseTax	-0.03	-0.16	-0.02	-0.02	0.13	0.62	0.88	0.62	0.87	0.67	1.00	0.13	0.82	-0.09	-0.03	-0.03
FreightCost	-0.21	-0.16	-0.00	-0.00	-0.02	0.17	0.15	0.17	0.15	0.20	0.13	1.00	0.14	-0.01	0.00	0.00
GrossProfit	-0.06	-0.13	-0.02	-0.02	0.10	0.69	0.96	0.70	0.98	0.81	0.82	0.14	1.00	-0.07	-0.04	-0.04
ProfitMargin	-0.02	0.07	0.04	0.05	-0.06	-0.12	-0.12	-0.11	-0.11	-0.18	-0.09	-0.01	-0.07	1.00	0.40	0.42
StockTurnover	-0.00	-0.02	0.00	0.00	-0.04	-0.05	-0.04	-0.04	-0.04	-0.07	-0.03	0.00	-0.04	0.40	1.00	1.00
SalestoPurchaseRatio	-0.00	-0.01	0.00	0.00	-0.04	-0.05	-0.04	-0.05	-0.04	-0.07	-0.03	0.00	-0.04	0.42	1.00	1.00

Correlation Insights

1. PurchasePrice has weak correlation with TotalSalesDollars (-0.012) and GrossProfit(-0.016), suggesting that price variations do not significantly impact sales revenue or profit.

2. Strong correlation between total purchase quantity and total sales quantity (0.999), confirming efficient inventory turnover.

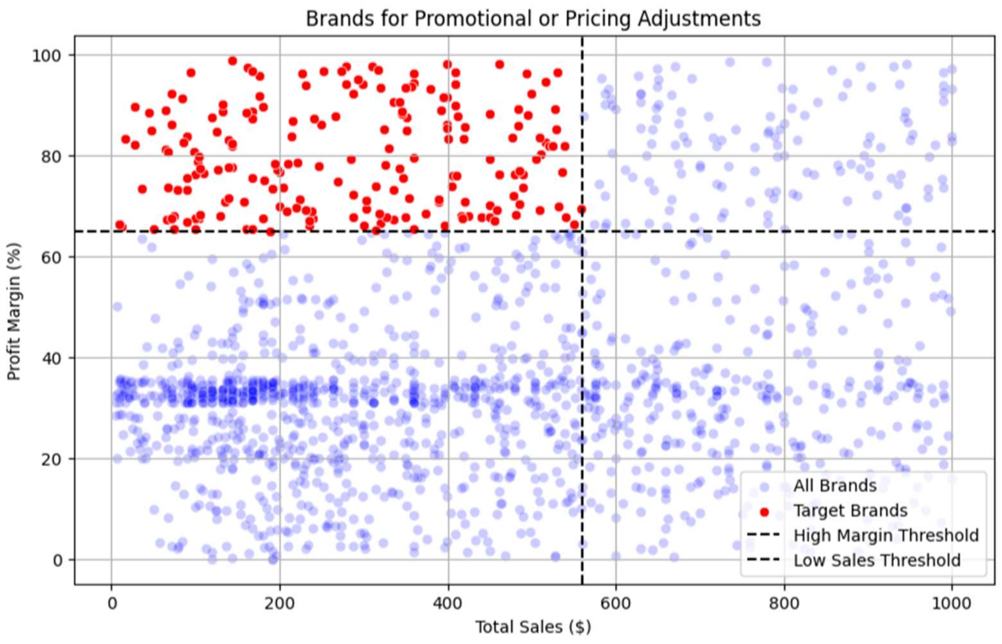
3. Negative correlation between profit margin & total sales price (-0.179) suggests that as sales price increases, margins decrease, possibly due to competitive pricing pressure.

4. StockTurnover has weak negative correlations with both GrossProfit (-0.038) and ProfitMargin(-0.055), indicating that faster turnover does not necessarily result in higher profitability.

4. Vendor Performance Analysis

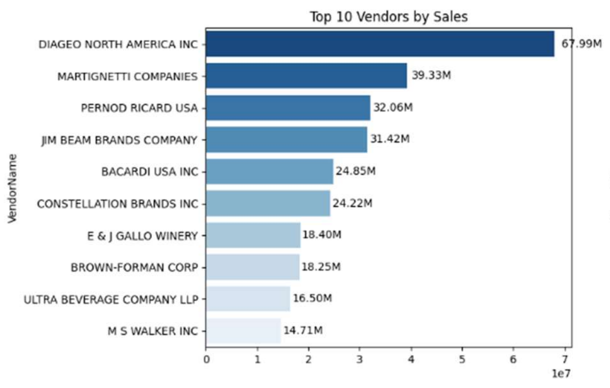
4.1 Underperforming Brands Identification

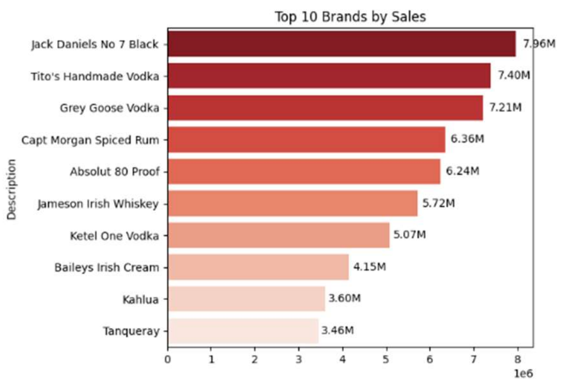
Scatterplots of profit margin versus total sales highlighted brands with low sales but high profit margins. These brands are candidates for promotional activities or price adjustments to increase revenue without sacrificing profitability.

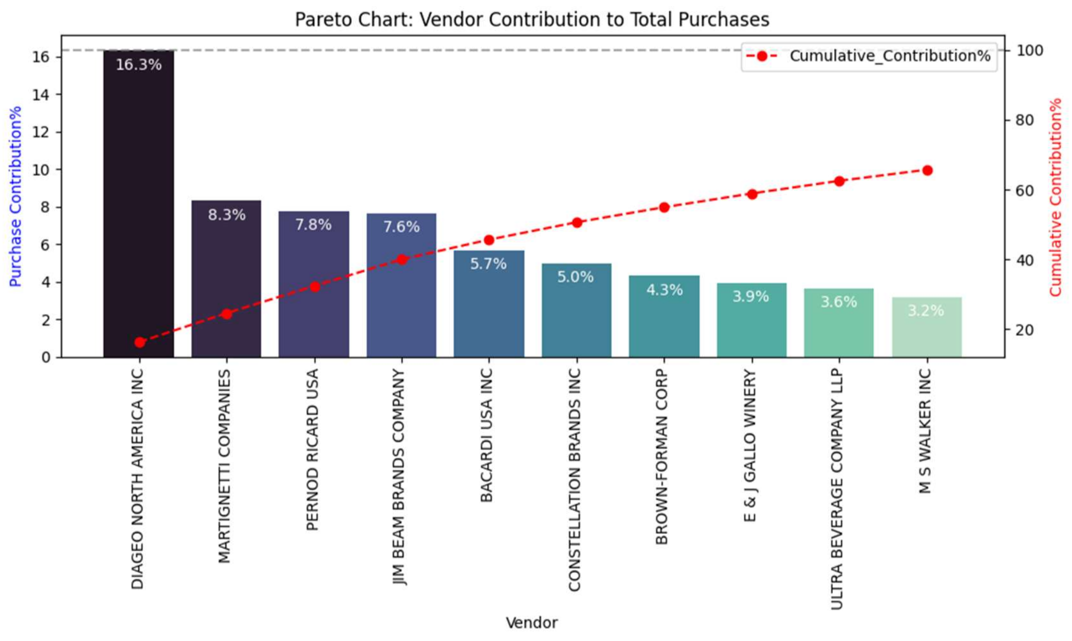


4.2 Top Vendors and Brands by Sales

Bar charts and Pareto analysis illustrated that the top 10 vendors contribute over 70% of total purchase dollars, showing supply concentration and vendor dependency.

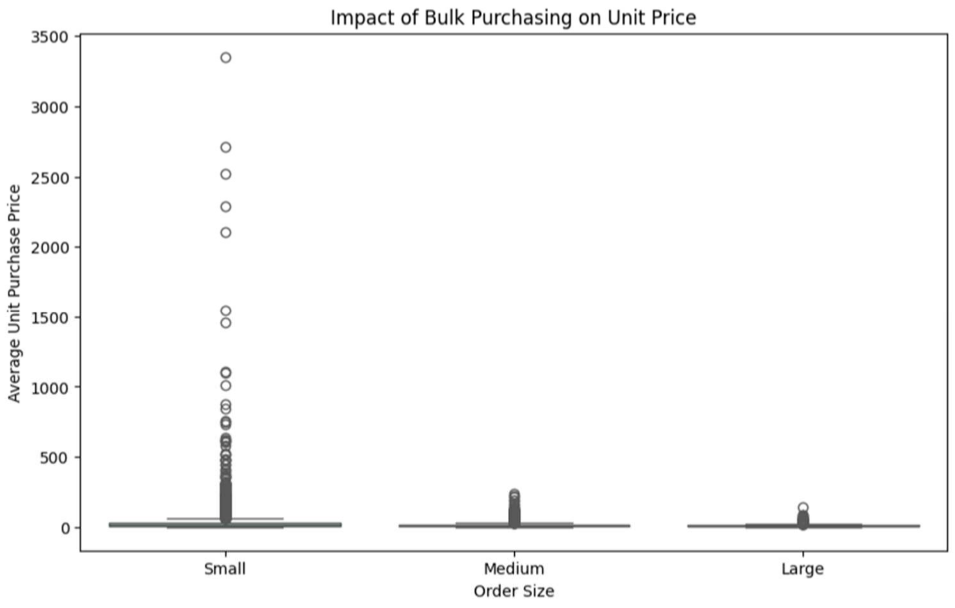






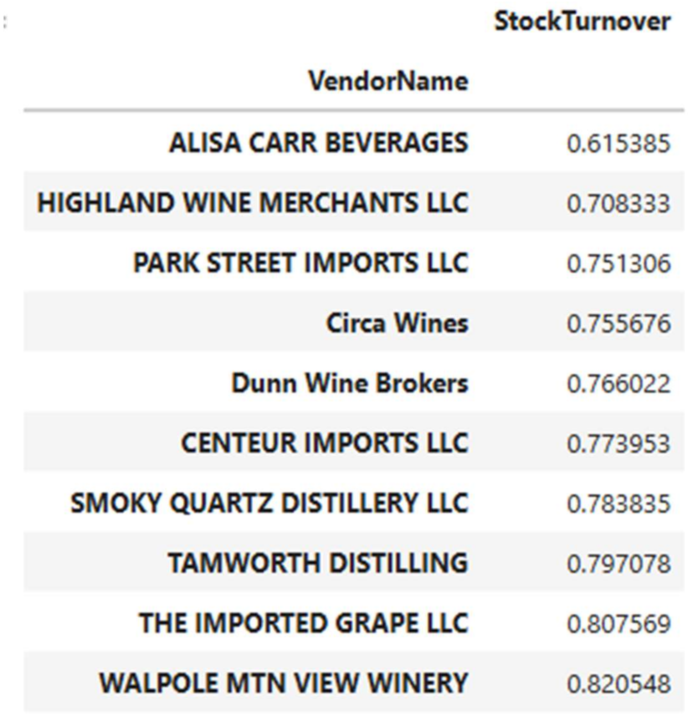
4.3 Bulk Purchasing Impact on Cost

Analysis of order size categories (small, medium, large) against unit purchase price demonstrated that large orders realize up to 72% reduction in unit cost, emphasizing the cost-saving benefits of bulk purchasing.



4.4 Inventory Turnover and Locked Capital

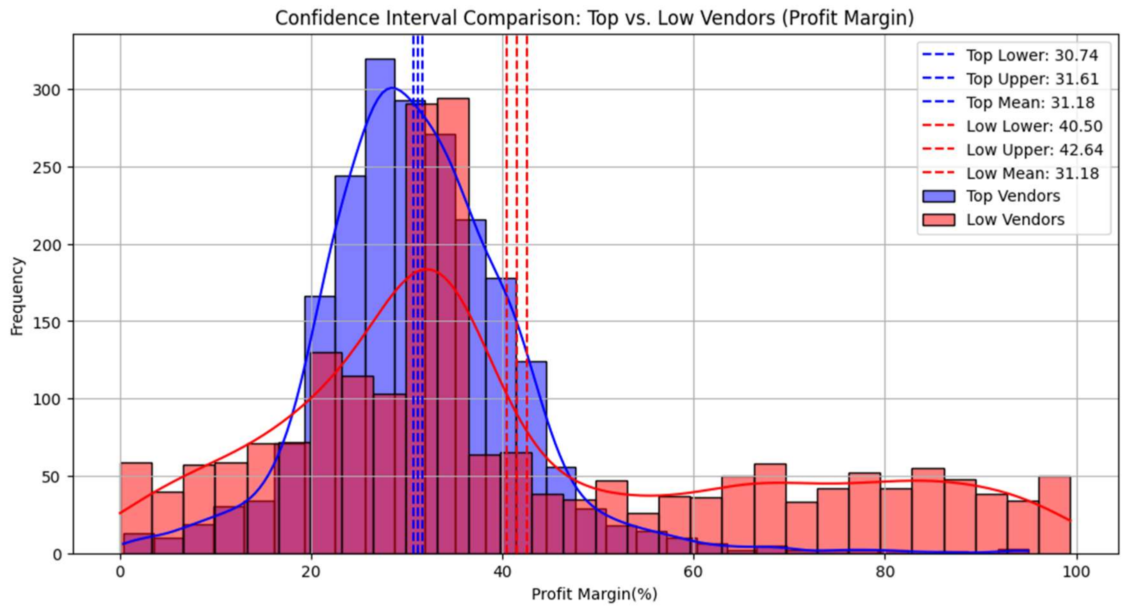
Turnover analysis identified vendors and brands with stock turnover ratios below 1, indicating slow-moving inventory. Calculations of unsold inventory value highlighted where capital is constrained by excess stock.



VendorName	StockTurnover
ALISA CARR BEVERAGES	0.615385
HIGHLAND WINE MERCHANTS LLC	0.708333
PARK STREET IMPORTS LLC	0.751306
Circa Wines	0.755676
Dunn Wine Brokers	0.766022
CENTEUR IMPORTS LLC	0.773953
SMOKY QUARTZ DISTILLERY LLC	0.783835
TAMWORTH DISTILLING	0.797078
THE IMPORTED GRAPE LLC	0.807569
WALPOLE MTN VIEW WINERY	0.820548

4.5 Profitability Differences & Statistical Testing

Confidence intervals for profit margins showed that low-sales vendors often have higher margins than high-sales vendors. A two-sample t-test confirmed statistically significant differences in these profit margins, suggesting distinct management strategies.



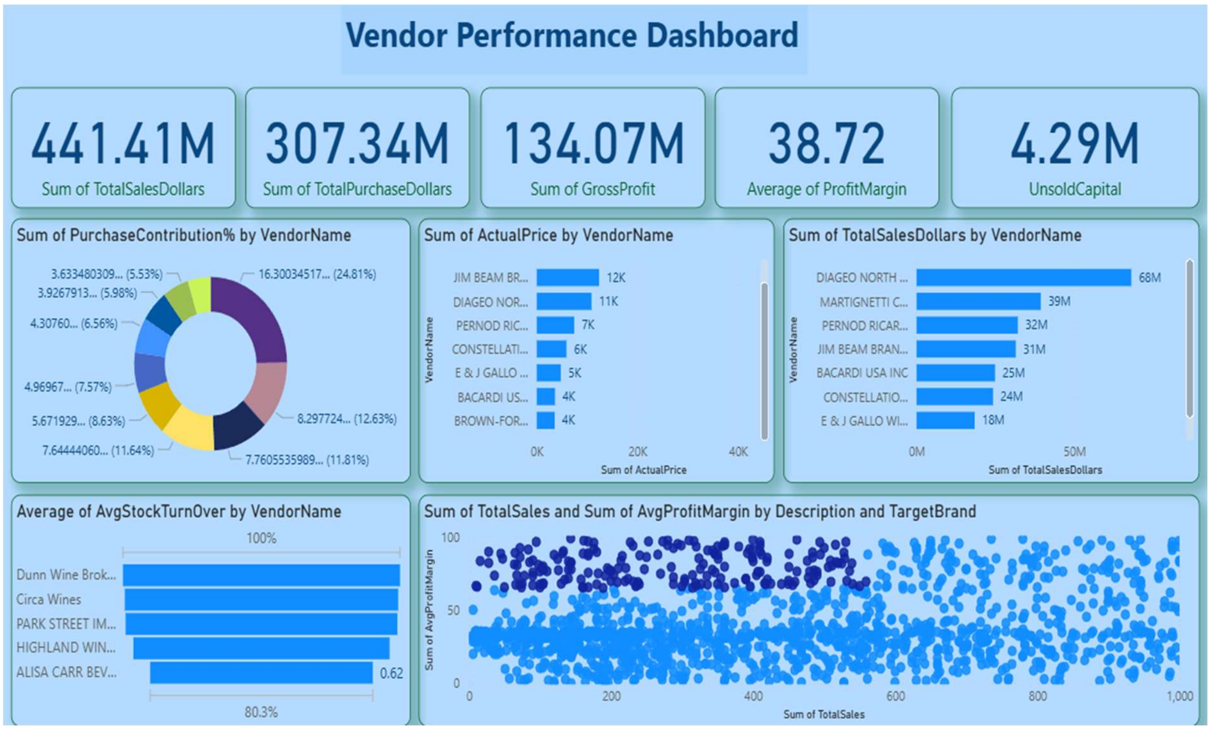
1. The confidence interval for low_performing vendors (40.48% to 42.62%) is significantly higher than that of top-performing vendors (30.74% to 31.61%).
2. This suggests that vendors with lower sales tend to maintian higher profit margins, potentially due to premium pricing or lower operational costs.
3. For High-Performing Vendors: If they aim to improve profitability, the could explore selective price adjustments, cost optimization, or bundling strategies.
4. For low-Performing Vendors: Despite higher margins, their low sales volumne might indicate a need for better marketing, competitive pricing, or improved distribution strategies.

5. Power BI Visualization and Supporting Tables

An interactive Power BI dashboard consolidates key insights, featuring visuals like KPIs, vendor and brand performance charts, inventory turnover heatmaps, and purchase contribution analyses.

Supporting analytical tables powering the dashboard include:

- BrandPerformance: Aggregates sales and profit metrics at the brand level.
- LowTurnVendor: Lists vendors with low stock turnover.
- PurchaseContribution: Details vendor purchase share and cumulative contribution for Pareto analysis.



6. Conclusion and Recommendations

This project provides a robust data-driven framework for vendor management and inventory optimization, revealing key insights into vendor contribution, brand profitability, purchasing strategies, and inventory efficiency.

Recommendations:

- Promote or price-adjust underperforming but profitable brands.
- Leverage top vendors for negotiation while mitigating dependency risks.
- Encourage bulk purchases to reduce unit costs.
- Improve inventory turnover by managing slow-moving stock.
- Tailor vendor management based on performance segments.

7. Technical Appendix

- ETL ingestion Python script

```
import pandas as pd
import os
from sqlalchemy import create_engine
import logging
import time

# Make sure 'logs' directory exists
os.makedirs("logs", exist_ok=True)
```

```
logging.basicConfig(
    filename="logs/ingestion_db.log",
    level= logging.DEBUG,
    format = "%(asctime)s - %(levelname)s - %(message)s",
    filemode="a"
)

engine = create_engine('sqlite:///inventory.db')

def ingest_db(df, table_name, engine):
    '''this funtion will ingest the dataframe into databse table'''
    df.to_sql(table_name, con = engine, if_exists = 'replace', index = False)



def load_raw_data():
    '''this function will load the CSVs as dataframe and ingest into db'''
    start = time.time()
    for file in os.listdir('data'):
        if '.csv' in file:
            df = pd.read_csv('data/' + file)
            logging.info(f'Ingesting {file} in db')
            ingest_db(df, file[:-4], engine)
    end = time.time()
    total_time = (end-start)/60
    logging.info('---------------Ingestion Complete----------------------')

    logging.info(f'Total Time Taken: {total_time} minutes')



if __name__ == '__main__':
    load_raw_data()
```

- SQL query for vendor summary creation

```
        import pandas as pd
import sqlite3
import logging
from ingestion_db import ingest_db
```

```
logging.basicConfig(
    filename="logs/get_vendor_summary.log",
    level=logging.DEBUG,
    format="%(asctime)s - %(levelname)s - %(message)s",
    filemode="a"
)

def create_vendor_summary(conn):
    '''This function merges tables to get overall vendor summary and adds new calculated columns.'''
    vendor_sales_summary = pd.read_sql_query('''
    WITH FreightSummary AS (
        SELECT
            VendorNumber,
            SUM(Freight) AS FreightCost
        FROM vendor_invoice
        GROUP BY VendorNumber
    ),

    PurchaseSummary AS (
        SELECT
            p.VendorNumber,
            p.VendorName,
            p.Brand,
            p.Description,
            p.PurchasePrice,
            pp.Volume,
            pp.Price AS ActualPrice,
            SUM(p.Quantity) AS TotalPurchaseQuantity,
            SUM(p.Dollars) AS TotalPurchaseDollars
        FROM purchases p
        JOIN purchase_prices pp
            ON p.Brand = pp.Brand
        WHERE p.PurchasePrice > 0
        GROUP BY p.VendorNumber, p.VendorName, p.Brand, p.Description, p.PurchasePrice, pp.Price,
pp.Volume
    ),

    SalesSummary AS (
        SELECT
```

```
            VendorNo,
            Brand,
            SUM(SalesDollars) AS TotalSalesDollars,
            SUM(SalesPrice) AS TotalSalesPrice,
            SUM(SalesQuantity) AS TotalSalesQuantity,
            SUM(ExciseTax) AS TotalExciseTax
        FROM sales
        GROUP BY VendorNo, Brand
    )

    SELECT
        ps.VendorNumber,
        ps.VendorName,
        ps.Brand,
        ps.Description,
        ps.PurchasePrice,
        ps.ActualPrice,
        ps.Volume,
        ps.TotalPurchaseQuantity,
        ps.TotalPurchaseDollars,
        ss.TotalSalesQuantity,
        ss.TotalSalesDollars,
        ss.TotalSalesPrice,
        ss.TotalExciseTax,
        fs.FreightCost
    FROM PurchaseSummary ps
    LEFT JOIN SalesSummary ss
        ON ps.VendorNumber = ss.VendorNo
        AND ps.Brand = ss.Brand
    LEFT JOIN FreightSummary fs
        ON ps.VendorNumber = fs.VendorNumber
    ORDER BY ps.TotalPurchaseDollars DESC
    ''', conn)

    return vendor_sales_summary

def clean_data(df):
    '''this function will clean the data'''
    #changing datatype to float
    df['Volume'] = df['Volumne'].astype('float')

    #filling missing value with 0
    df.fillna(0, inplace = True)

    #removing spaces from categorical columns
    df['VendorName'] = df['VendorName'].str.strip()
    df['Description'] = df['Description'].str.strip()

    #creating new columns for better analysis
    vendor_sales_summary['GrossProfit'] = vendor_sales_summary['TotalSalesDollars'] -
vendor_sales_summary['TotalPurchaseDollars']
    vendor_sales_summary['ProfitMargin'] = (vendor_sales_summary['GrossProfit']/
vendor_sales_summary['TotalSalesDollars'])*100
    vendor_sales_summary['StockTurnover'] =
vendor_sales_summary['TotalSalesQuantity']/vendor_sales_summary['TotalPurchaseQuantity']
    vendor_sales_summary['SalestoPurchaseRatio'] =
vendor_sales_summary['TotalSalesDollars']/vendor_sales_summary['TotalPurchaseDollars']

    return df

if __name__ == '__main__':
    #creating database connection
    conn = sqlite3.connect('inventory.db')

    logging.info('Creating Vendor Summary Table......')
    summary_df = create_vendor_summary(conn)
    logging.info(summary_df.head())

    logging.info('Cleaning Data.....')
    clean_df = clean_data(summary_df)
    logging.info(clean_df.head())

    logging.info('Ingesting data.....')
    ingest_db(clean_df,'vendor_sales_summary', conn)
    logging.info('Completed')
```

- Python data cleaning and feature engineering functions

```
import pandas as pd
import sqlite3
import logging
from ingestion_db import ingest_db

logging.basicConfig(
    filename="logs/get_vendor_summary.log",
    level=logging.DEBUG,
    format="%(asctime)s - %(levelname)s - %(message)s",
    filemode="a"
)

def create_vendor_summary(conn):
    '''This function merges tables to get overall vendor summary and adds new calculated columns.'''
    vendor_sales_summary = pd.read_sql_query('''
    WITH FreightSummary AS (
        SELECT
            VendorNumber,
            SUM(Freight) AS FreightCost
        FROM vendor_invoice
        GROUP BY VendorNumber
    ),

    PurchaseSummary AS (
        SELECT
            p.VendorNumber,
            p.VendorName,
            p.Brand,
            p.Description,
            p.PurchasePrice,
            pp.Volume,
            pp.Price AS ActualPrice,
            SUM(p.Quantity) AS TotalPurchaseQuantity,
            SUM(p.Dollars) AS TotalPurchaseDollars
        FROM purchases p
        JOIN purchase_prices pp
        ON p.Brand = pp.Brand
      WHERE p.PurchasePrice > 0
      GROUP BY p.VendorNumber, p.VendorName, p.Brand, p.Description, p.PurchasePrice, pp.Price,
pp.Volume
   ),

   SalesSummary AS (
      SELECT
        VendorNo,
        Brand,
        SUM(SalesDollars) AS TotalSalesDollars,
        SUM(SalesPrice) AS TotalSalesPrice,
        SUM(SalesQuantity) AS TotalSalesQuantity,
        SUM(ExciseTax) AS TotalExciseTax
      FROM sales
      GROUP BY VendorNo, Brand
   )

   SELECT
      ps.VendorNumber,
      ps.VendorName,
      ps.Brand,
      ps.Description,
      ps.PurchasePrice,
      ps.ActualPrice,
      ps.Volume,
      ps.TotalPurchaseQuantity,
      ps.TotalPurchaseDollars,
      ss.TotalSalesQuantity,
      ss.TotalSalesDollars,
      ss.TotalSalesPrice,
      ss.TotalExciseTax,
      fs.FreightCost
   FROM PurchaseSummary ps
   LEFT JOIN SalesSummary ss
      ON ps.VendorNumber = ss.VendorNo
      AND ps.Brand = ss.Brand
   LEFT JOIN FreightSummary fs
      ON ps.VendorNumber = fs.VendorNumber
```

```
    ORDER BY ps.TotalPurchaseDollars DESC
    ''', conn)

    return vendor_sales_summary

def clean_data(df):
    '''this function will clean the data'''
    #changing datatype to float
    df['Volume'] = df['Volumne'].astype('float')

    #filling missing value with 0
    df.fillna(0, inplace = True)

    #removing spaces from categorical columns
    df['VendorName'] = df['VendorName'].str.strip()
    df['Description'] = df['Description'].str.strip()

    #creating new columns for better analysis
    vendor_sales_summary['GrossProfit'] = vendor_sales_summary['TotalSalesDollars'] - 
vendor_sales_summary['TotalPurchaseDollars']
    vendor_sales_summary['ProfitMargin'] = (vendor_sales_summary['GrossProfit']/
vendor_sales_summary['TotalSalesDollars'])*100
    vendor_sales_summary['StockTurnover'] = 
vendor_sales_summary['TotalSalesQuantity']/vendor_sales_summary['TotalPurchaseQuantity']
    vendor_sales_summary['SalestoPurchaseRatio'] = 
vendor_sales_summary['TotalSalesDollars']/vendor_sales_summary['TotalPurchaseDollars']

    return df

if __name__ == '__main__':
    #creating database connection
    conn = sqlite3.connect('inventory.db')

    logging.info('Creating Vendor Summary Table......')
    summary_df = create_vendor_summary(conn)
    logging.info(summary_df.head())

    logging.info('Cleaning Data.....')
    clean_df = clean_data(summary_df)
    logging.info(clean_df.head())
    logging.info('Ingesting data.....')
    ingest_db(clean_df,'vendor_sales_summary', conn)
    logging.info('Completed')
```

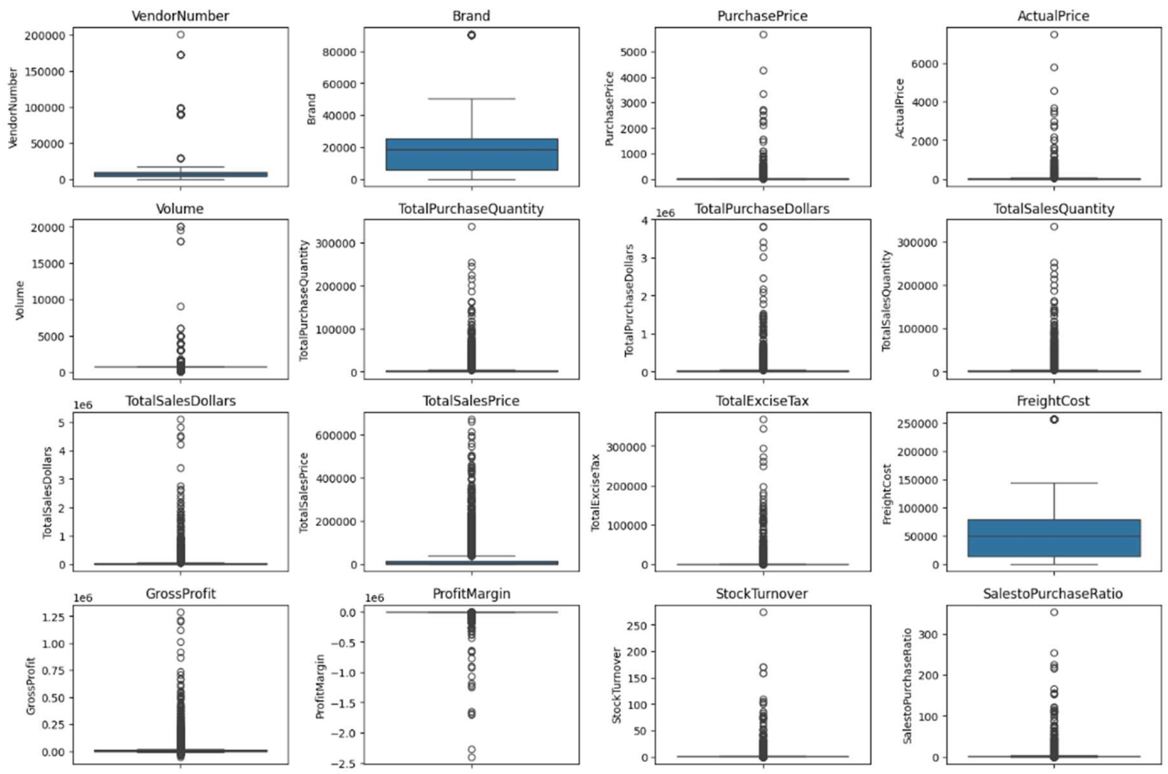
- Profitability statistical testing code

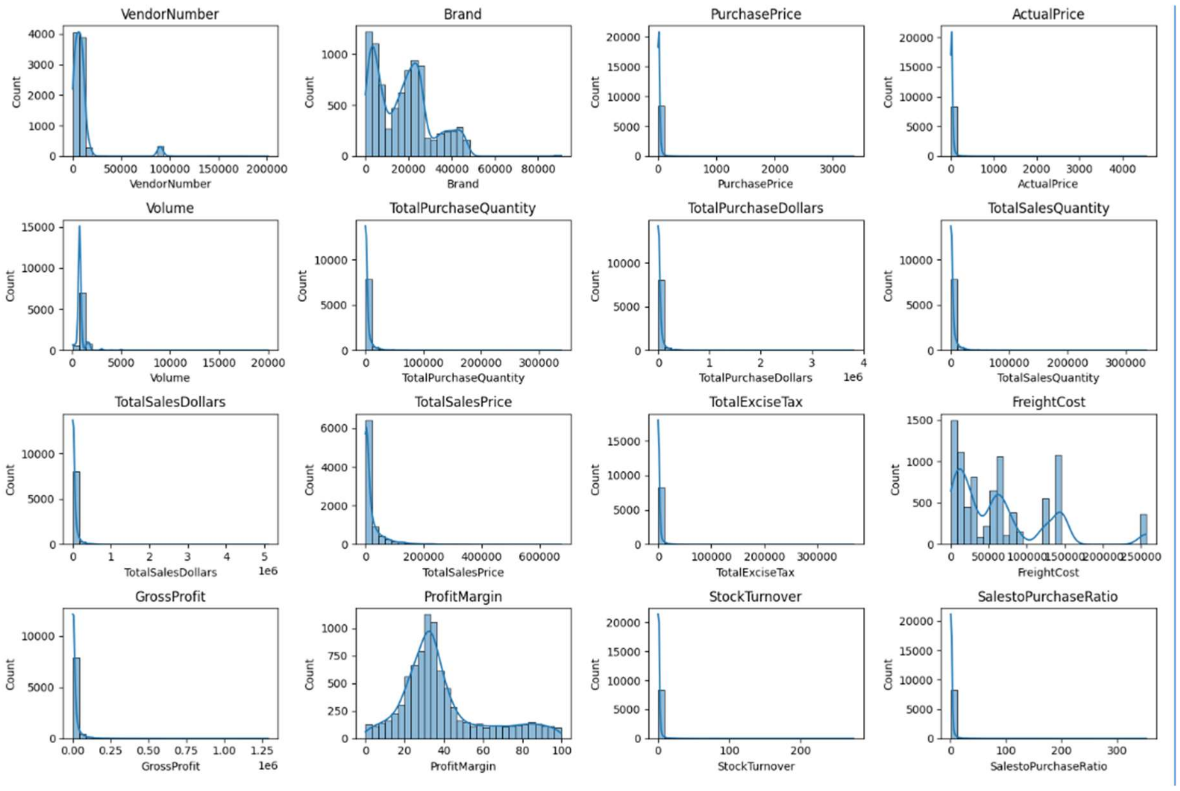
```
import pandas as pd
import sqlite3
import logging
from ingestion_db import ingest_db

logging.basicConfig(
    filename="logs/get_vendor_summary.log",
    level=logging.DEBUG,
    format="%(asctime)s - %(levelname)s - %(message)s",
    filemode="a"
)

def create_vendor_summary(conn):
    '''This function merges tables to get overall vendor summary and adds new calculated columns.'''
    vendor_sales_summary = pd.read_sql_query('''
    WITH FreightSummary AS (
        SELECT
            VendorNumber,
            SUM(Freight) AS FreightCost
        FROM vendor_invoice
        GROUP BY VendorNumber
    ),

    PurchaseSummary AS (
        SELECT
            p.VendorNumber,
            p.VendorName,
            p.Brand,
            p.Description,
            p.PurchasePrice,
            pp.Volume,
            pp.Price AS ActualPrice,
            SUM(p.Quantity) AS TotalPurchaseQuantity,
            SUM(p.Dollars) AS TotalPurchaseDollars
        FROM purchases p
        JOIN purchase_prices pp
            ON p.Brand = pp.Brand
        WHERE p.PurchasePrice > 0
        GROUP BY p.VendorNumber, p.VendorName, p.Brand, p.Description, p.PurchasePrice, pp.Price,
pp.Volume
    ),

    SalesSummary AS (
        SELECT
            VendorNo,
            Brand,
            SUM(SalesDollars) AS TotalSalesDollars,
            SUM(SalesPrice) AS TotalSalesPrice,
            SUM(SalesQuantity) AS TotalSalesQuantity,
            SUM(ExciseTax) AS TotalExciseTax
        FROM sales
        GROUP BY VendorNo, Brand
    )

    SELECT
        ps.VendorNumber,
        ps.VendorName,
        ps.Brand,
        ps.Description,
        ps.PurchasePrice,
        ps.ActualPrice,
        ps.Volume,
        ps.TotalPurchaseQuantity,
        ps.TotalPurchaseDollars,
        ss.TotalSalesQuantity,
        ss.TotalSalesDollars,
        ss.TotalSalesPrice,
        ss.TotalExciseTax,
        fs.FreightCost
    FROM PurchaseSummary ps
```

```
    LEFT JOIN SalesSummary ss
        ON ps.VendorNumber = ss.VendorNo
        AND ps.Brand = ss.Brand
    LEFT JOIN FreightSummary fs
        ON ps.VendorNumber = fs.VendorNumber
    ORDER BY ps.TotalPurchaseDollars DESC
    ''', conn)

    return vendor_sales_summary

def clean_data(df):
    '''this function will clean the data'''
    #changing datatype to float
    df['Volume'] = df['Volumne'].astype('float')

    #filling missing value with 0
    df.fillna(0, inplace = True)

    #removing spaces from categorical columns
    df['VendorName'] = df['VendorName'].str.strip()
    df['Description'] = df['Description'].str.strip()

    #creating new columns for better analysis
    vendor_sales_summary['GrossProfit'] = vendor_sales_summary['TotalSalesDollars'] -
vendor_sales_summary['TotalPurchaseDollars']
    vendor_sales_summary['ProfitMargin'] = (vendor_sales_summary['GrossProfit']/
vendor_sales_summary['TotalSalesDollars'])*100
    vendor_sales_summary['StockTurnover'] =
vendor_sales_summary['TotalSalesQuantity']/vendor_sales_summary['TotalPurchaseQuantity']
    vendor_sales_summary['SalestoPurchaseRatio'] =
vendor_sales_summary['TotalSalesDollars']/vendor_sales_summary['TotalPurchaseDollars']

    return df

if __name__ == '__main__':
    #creating database connection
    conn = sqlite3.connect('inventory.db')

    logging.info('Creating Vendor Summary Table......')
    summary_df = create_vendor_summary(conn)
```

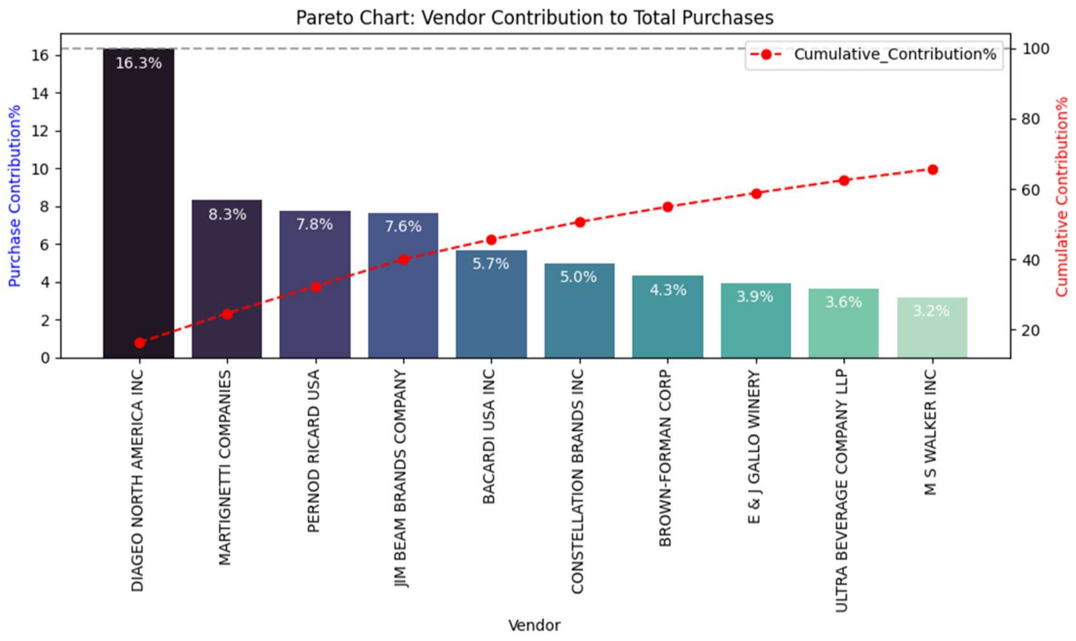
```
logging.info(summary_df.head())

logging.info('Cleaning Data.....')
clean_df = clean_data(summary_df)
logging.info(clean_df.head())

logging.info('Ingesting data.....')
ingest_db(clean_df,'vendor_sales_summary', conn)
logging.info('Completed')
```

8. Visual Appendix

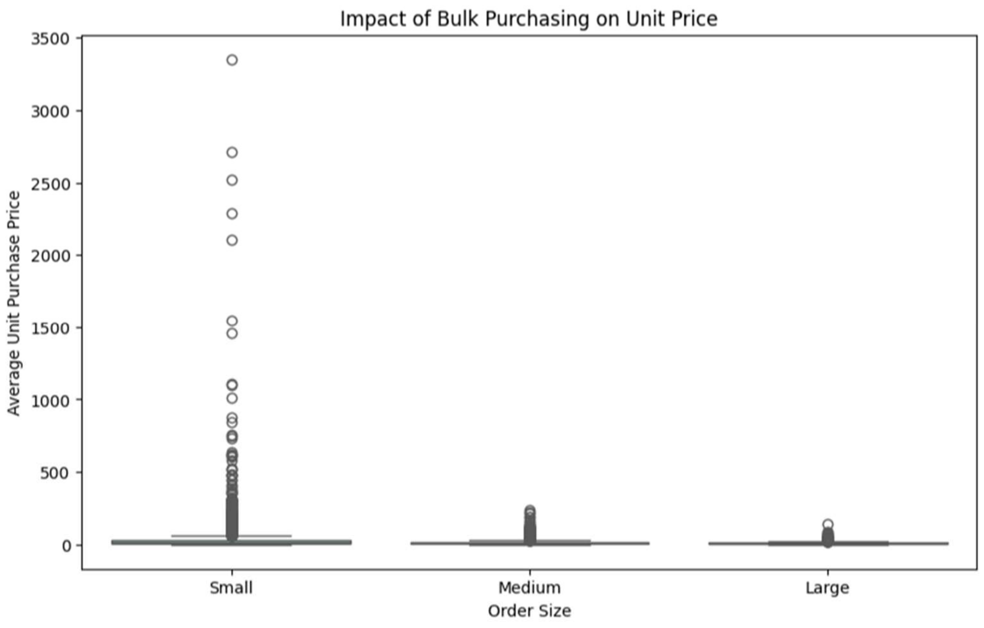




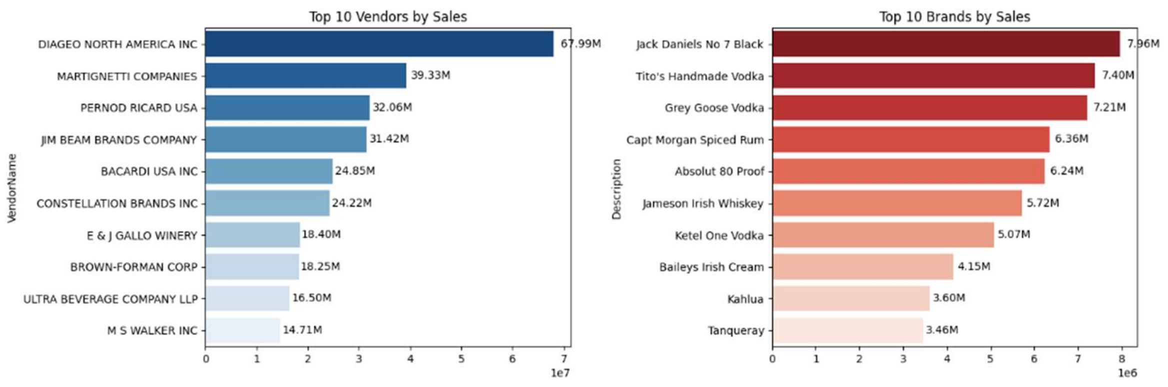
- Figure 1: Distribution histograms of numerical columns



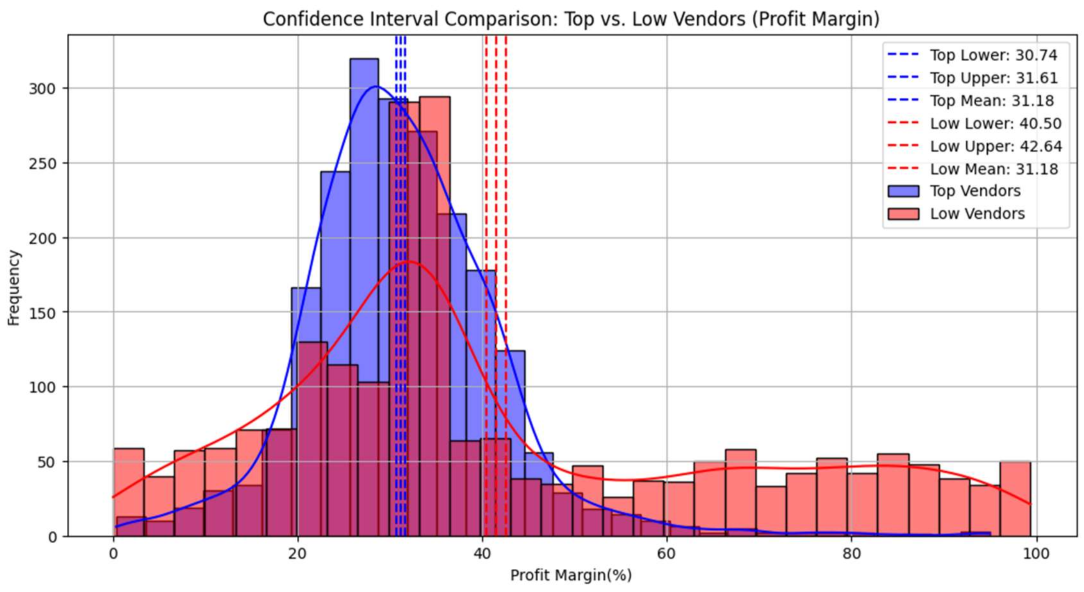
- Figure 2: Vendor sales Pareto chart



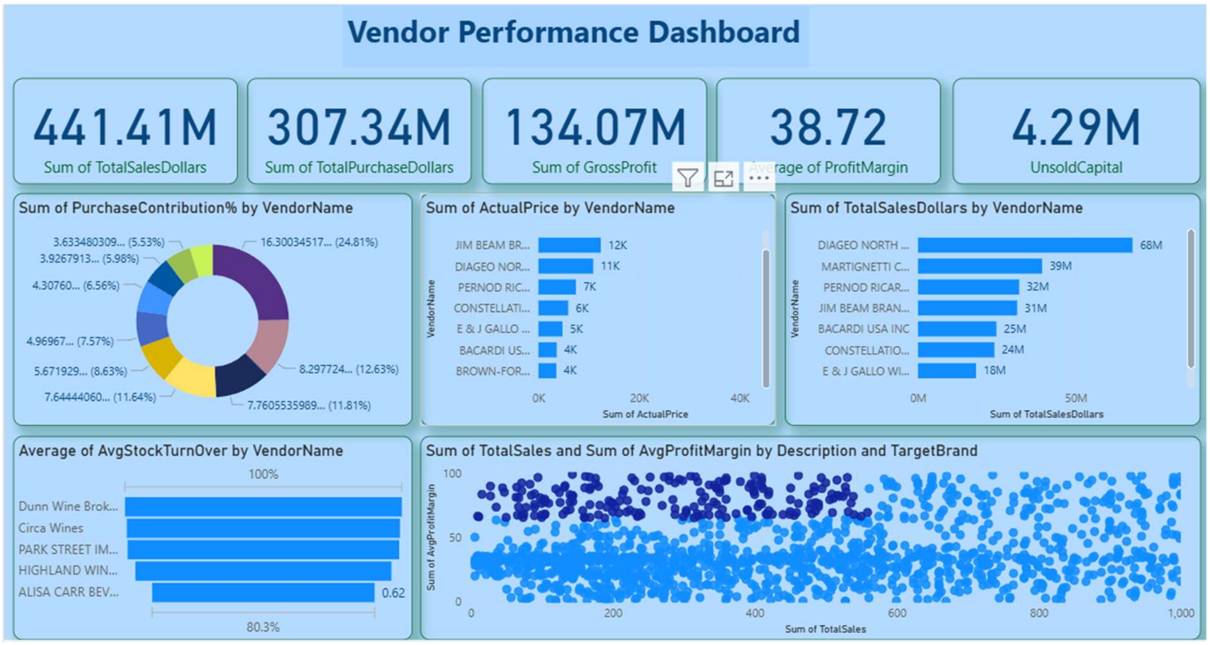
- Figure 3: Bulk purchase impact boxplot



- Figure 4: Inventory turnover bar chart



- Figure 5: Profitability confidence interval histograms



- Figure 6: Power BI dashboard screenshot