In [1]:
```
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')
```

In [2]:
```
conn = sqlite3.connect('inventory.db')
df = pd.read_sql_query("select * from vendor_sales_summary",conn)
df.head()
```

Out[2]:

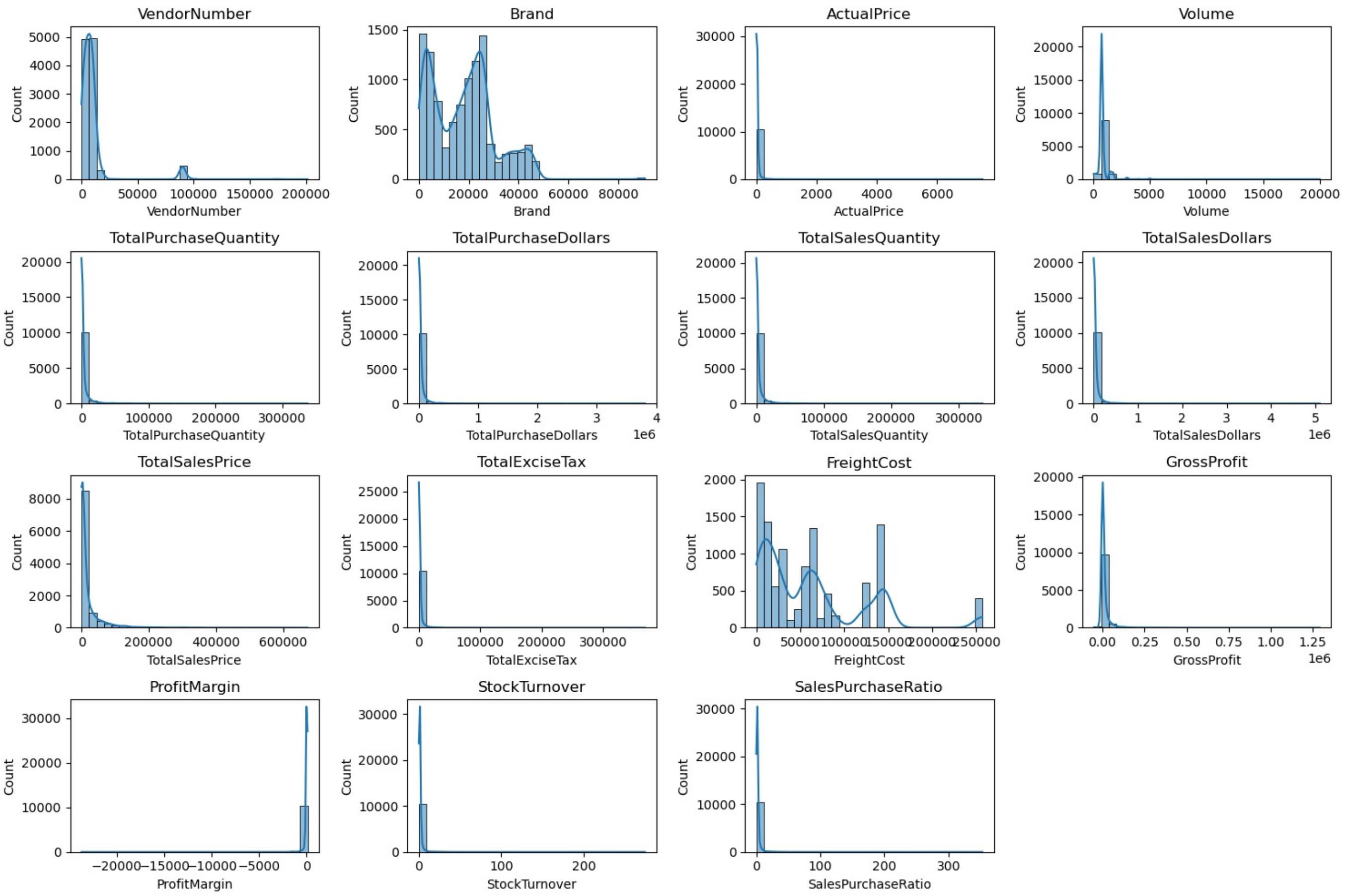
	VendorNumber	VendorName	Brand	Description	ActualPrice	Volume	TotalPurchaseQuantity	TotalPurchaseDollars	TotalSale
0	1128	BROWN-FORMAN CORP	1233	Jack Daniels No 7 Black	36.99	1750.0	145080	3811251.60	
1	4425	MARTIGNETTI COMPANIES	3405	Tito's Handmade Vodka	28.99	1750.0	164038	3804041.22	
2	17035	PERNOD RICARD USA	8068	Absolut 80 Proof	24.99	1750.0	187407	3418303.68	
3	3960	DIAGEO NORTH AMERICA INC	4261	Capt Morgan Spiced Rum	22.99	1750.0	201682	3261197.94	
4	3960	DIAGEO NORTH AMERICA INC	3545	Ketel One Vodka	29.99	1750.0	138109	3023206.01	

In [3]:
```
df.describe().T
```

Out[3]:

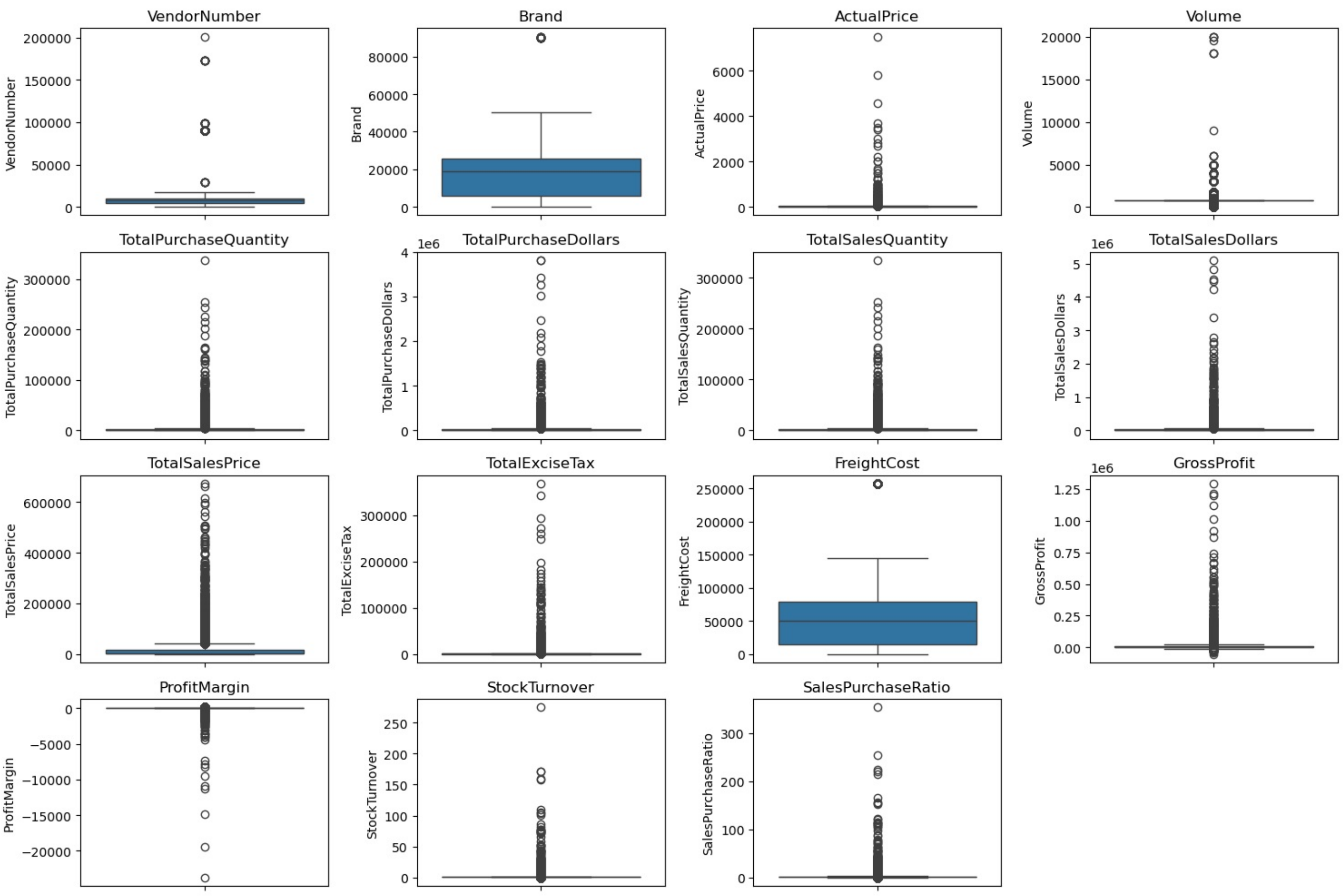
	count	mean	std	min	25%	50%	75%	max
VendorNumber	10692.0	1.065065e+04	18753.519148	2.00	3951.000000	7153.000000	9552.000000	2.013590e+05
Brand	10692.0	1.803923e+04	12662.187074	58.00	5793.500000	18761.500000	25514.250000	9.063100e+04
ActualPrice	10692.0	3.564367e+01	148.246016	0.49	10.990000	15.990000	28.990000	7.499990e+03
Volume	10692.0	8.473605e+02	664.309212	50.00	750.000000	750.000000	750.000000	2.000000e+04
TotalPurchaseQuantity	10692.0	3.140887e+03	11095.086769	1.00	36.000000	262.000000	1975.750000	3.376600e+05
TotalPurchaseDollars	10692.0	3.010669e+04	123067.799627	0.71	453.457500	3655.465000	20738.245000	3.811252e+06
TotalSalesQuantity	10692.0	3.077482e+03	10952.851391	0.00	33.000000	261.000000	1929.250000	3.349390e+05
TotalSalesDollars	10692.0	4.223907e+04	167655.265984	0.00	729.220000	5298.045000	28396.915000	5.101920e+06
TotalSalesPrice	10692.0	1.879378e+04	44952.773386	0.00	289.710000	2857.800000	16059.562500	6.728193e+05
TotalExciseTax	10692.0	1.774226e+03	10975.582240	0.00	4.800000	46.570000	418.650000	3.682428e+05
FreightCost	10692.0	6.143376e+04	60938.458032	0.09	14069.870000	50293.620000	79528.990000	2.570321e+05
GrossProfit	10692.0	1.213238e+04	46224.337964	-52002.78	52.920000	1399.640000	8660.200000	1.290668e+06
ProfitMargin	10692.0	-inf	NaN	-inf	13.324515	30.405457	39.956135	9.971666e+01
StockTurnover	10692.0	1.706793e+00	6.020460	0.00	0.807229	0.981529	1.039342	2.745000e+02
SalesPurchaseRatio	10692.0	2.504390e+00	8.459067	0.00	1.153729	1.436894	1.665449	3.529286e+02

In [4]:
```
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)
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(col)
plt.tight_layout()
plt.show()
```



In [5]:

```python
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 [ ]:

```
Negative & Zero Values:

Gross Profit: Minimum value is -52,002.78, indicating losses. Some products or transactions may be selling at a
Profit Margin: Has a minimum of-00, which suggests cases where revenue is zero or even lower than costs.
```

```
Total Sales Quantity & Sales Dollars: Minimum values are 0, meaning some products were purchased but never sold
stock
Outliers Indicated by High Standard Deviations:

Purchase & Actual Prices:
The max values (5,681.81 & 7,499.99) are significantly higher than the mean (24.39 & 35.64), indicating potenti
Freight Cost: Huge variation, from 0.09 to 257,032.07, suggests logistics inefficiencies or bulk shipments.
Stock Turnover: Ranges from 0 to 274.5, implying some products sell extremely fast while others remain in stock
```

In [6]:

```python
df = pd.read_sql_query("""SELECT *
FROM vendor_sales_summary
WHERE GrossProfit > 0
AND ProfitMargin > 0
AND TotalSalesQuantity > 0""",conn)
df
```

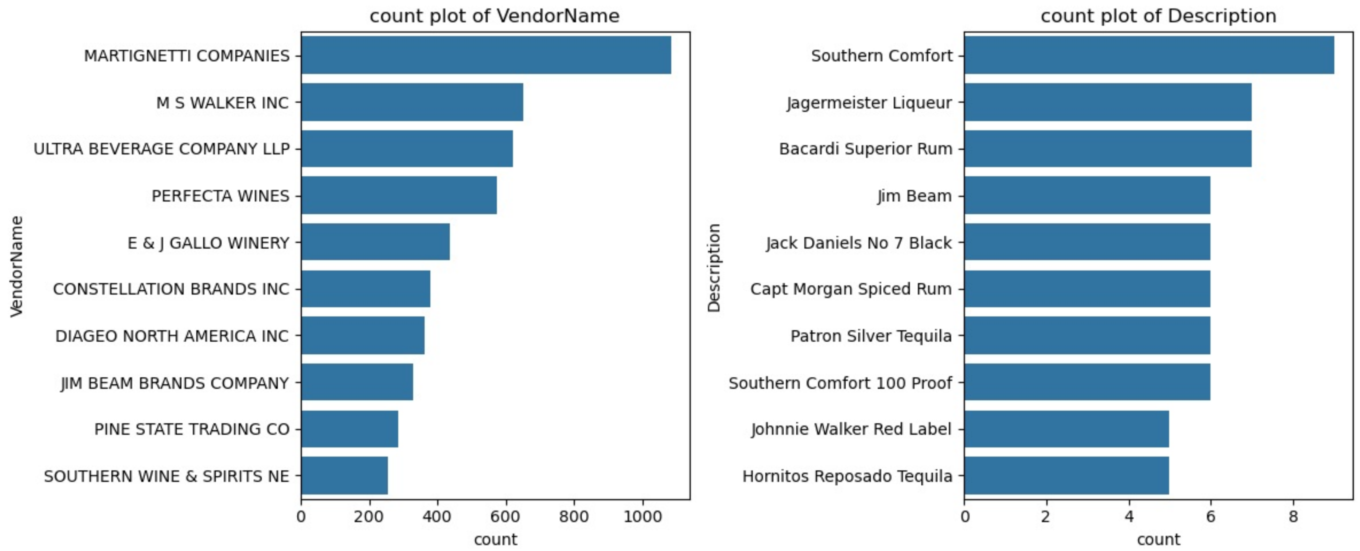
Out[6]:

	VendorNumber	VendorName	Brand	Description	ActualPrice	Volume	TotalPurchaseQuantity	TotalPurchaseDollars	Tota
0	1128	BROWN-FORMAN CORP	1233	Jack Daniels No 7 Black	36.99	1750.0	145080	3811251.60	
1	4425	MARTIGNETTI COMPANIES	3405	Tito's Handmade Vodka	28.99	1750.0	164038	3804041.22	
2	17035	PERNOD RICARD USA	8068	Absolut 80 Proof	24.99	1750.0	187407	3418303.68	
3	3960	DIAGEO NORTH AMERICA INC	4261	Capt Morgan Spiced Rum	22.99	1750.0	201682	3261197.94	
4	3960	DIAGEO NORTH AMERICA INC	3545	Ketel One Vodka	29.99	1750.0	138109	3023206.01	
...	...	...	...	...	...	...	...	...	
8559	9815	WINE GROUP INC	8527	Concannon Glen Ellen Wh Zin	4.99	750.0	2	2.64	
8560	8004	SAZERAC CO INC	5683	Dr McGillicuddy's Apple Pie	0.49	50.0	6	2.34	
8561	3924	HEAVEN HILL DISTILLERIES	9123	Deep Eddy Vodka	0.99	50.0	2	1.48	
8562	3960	DIAGEO NORTH AMERICA INC	6127	The Club Strawbry Margarita	1.99	200.0	1	1.47	
8563	7245	PROXIMO SPIRITS INC.	3065	Three Olives Grape Vodka	0.99	50.0	1	0.71	

8564 rows × 17 columns

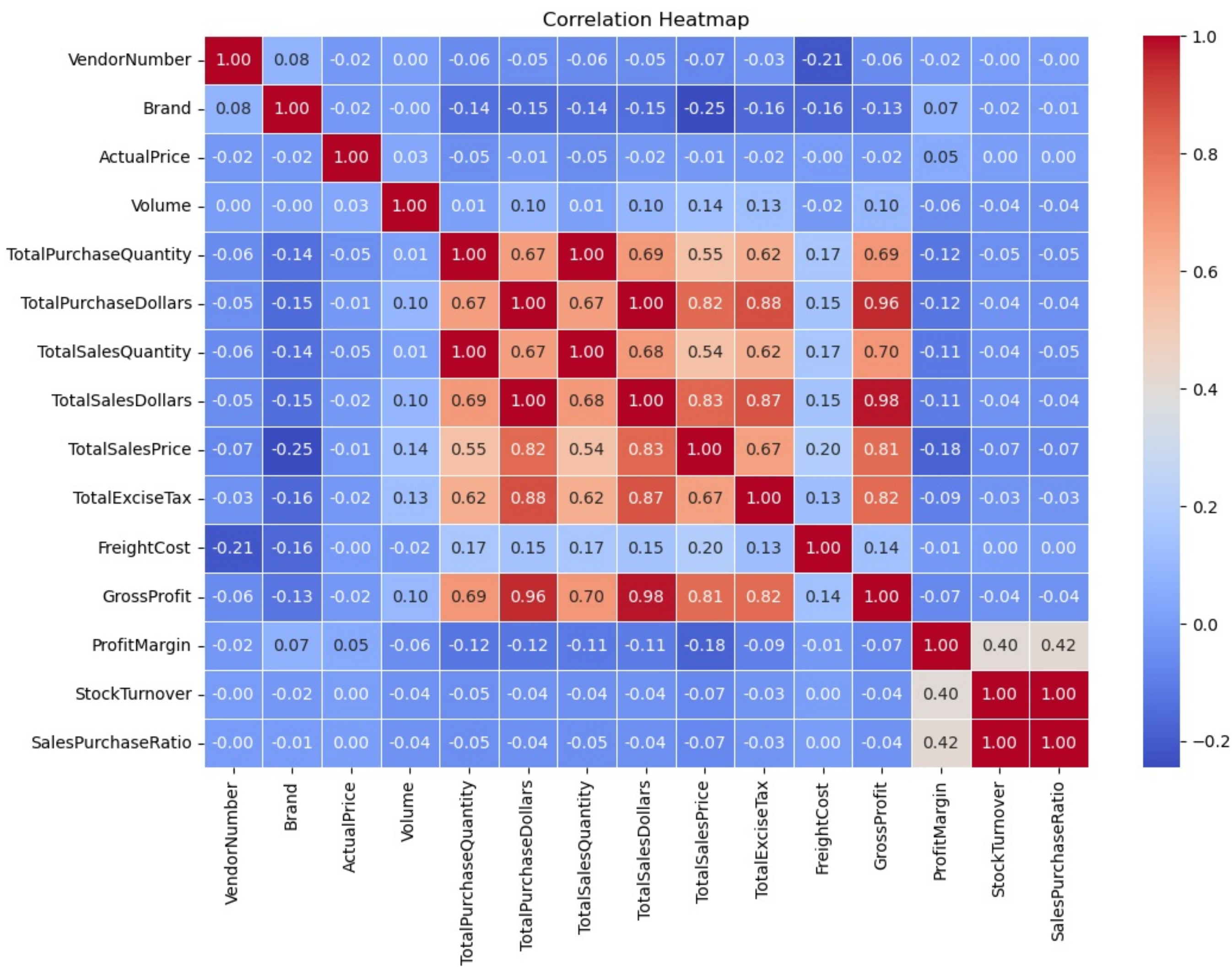
In [7]:

```python
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])
    plt.title(f"count plot of {col}")
plt.tight_layout()
plt.show()
```



In [8]:

```
plt.figure(figsize=(12,8))
correlation_matrix = df[numerical_cols].corr()
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```



In [ ]:

```
Correlation Insights
PurchasePrice has weak correlations with TotalSales Dollars (-0.012) and GrossProfit (-0.016), suggesting that 
Strong correlation between total purchase quantity and total sales quantity (0.999), confirming efficient inven
Negative correlation between profit margin & total sales price (-0.179) suggests that as sales plice increases,
Stock Turnover has weak negative correlations with both GrossProfit (-0.038) and ProfitMargin (-0.055), indicat
```

In [9]:

```
brand_performance = df.groupby('Description').agg({
    'TotalSalesDollars':'sum',
    'ProfitMargin':'mean'}).reset_index()
```

In [10]:

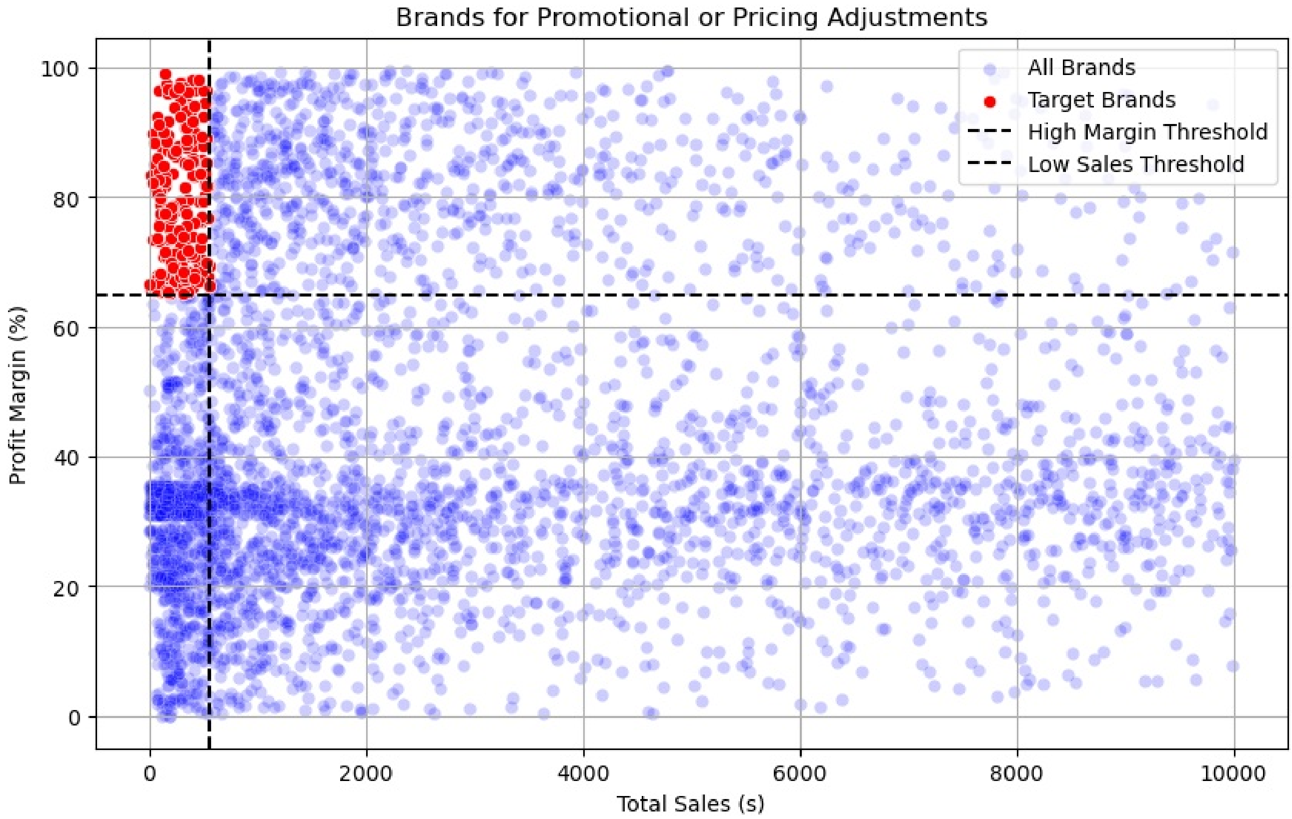
```
low_sales_threshold = brand_performance['TotalSalesDollars'].quantile(0.15)
high_margin_threshold = brand_performance['ProfitMargin'].quantile(0.85)
```

In [11]:
```
low_sales_threshold
```
Out[11]:
```
np.float64(560.299)
```
In [12]:
```
high_margin_threshold
```
Out[12]:
```
np.float64(64.97017552750113)
```
In [13]:
```
target_brands = brand_performance[
    (brand_performance['TotalSalesDollars'] <= low_sales_threshold) &
    (brand_performance['ProfitMargin'] >= high_margin_threshold)
     ]
print("Brands with low sales but Hight Proft Margins:")
display(target_brands.sort_values('TotalSalesDollars'))
```
```
Brands with low sales but Hight Proft Margins:
```

	Description	TotalSalesDollars	ProfitMargin
6199	Santa Rita Organic Svgn Bl	9.99	66.466466
2369	Debauchery Pnt Nr	11.58	65.975820
2070	Concannon Glen Ellen Wh Zin	15.95	83.448276
2188	Crown Royal Apple	27.86	89.806174
6237	Sauza Sprklg Wild Berry Marg	27.96	82.153076
...	...	...	...
5074	Nanbu Bijin Southern Beauty	535.68	76.747312
2271	Dad's Hat Rye Whiskey	538.89	81.851584
57	A Bichot Clos Marechaudes	539.94	67.740860
6245	Sbragia Home Ranch Merlot	549.75	66.444748
3326	Goulee Cos d'Estournel 10	558.87	69.434752

198 rows × 3 columns

In [14]:
```
brand_performance = brand_performance[brand_performance['TotalSalesDollars']<10000]
```
In [15]:
```
plt.figure(figsize=(10, 6))

sns.scatterplot(data=brand_performance, x='TotalSalesDollars', y='ProfitMargin', color="blue", label="All Brands
sns.scatterplot(data=target_brands, x='TotalSalesDollars', y='ProfitMargin', color="red", label="Target Brands"

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

plt.xlabel("Total Sales (s)")

plt.ylabel("Profit Margin (%)")

plt.title("Brands for Promotional or Pricing Adjustments")

plt.legend()

plt.grid(True)

plt.show()
```



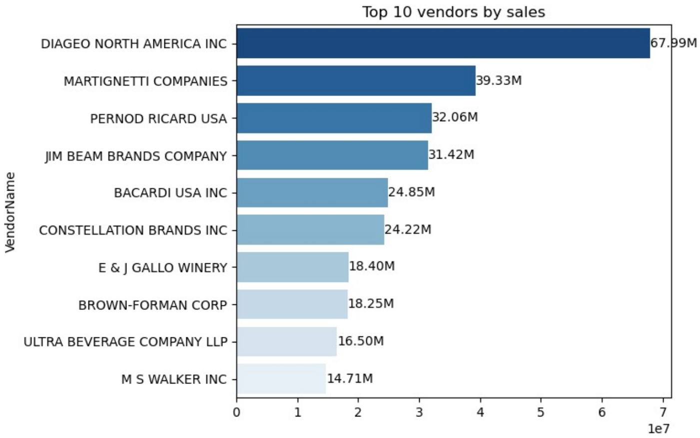
In [16]:
```
def format_dollars(value):
    if value >= 1_000_000:
        return f"{value / 1_000_000:.2f}M"
    elif value >= 1_000:
        return f"{value / 1_000:.2f}.K"
    else:
        return str(value)
```

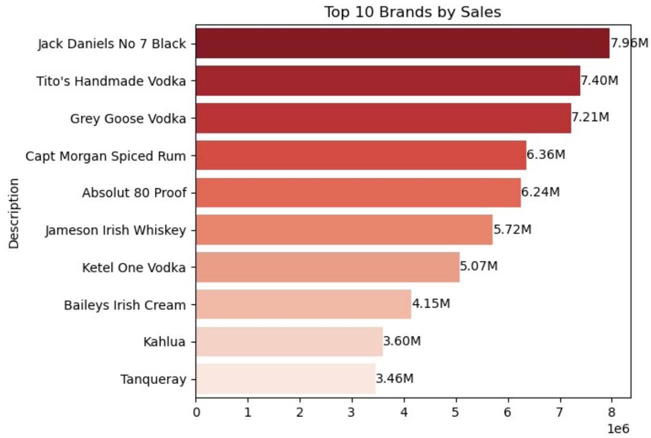
In [17]:
```
top_vendors  =df.groupby("VendorName")["TotalSalesDollars"].sum().nlargest(10)
top_brands  =df.groupby("Description")["TotalSalesDollars"].sum().nlargest(10)
```

In [18]:
```
top_brands.apply(lambda x : format_dollars(x))
```

Out[18]:
```
Description
Jack Daniels No 7 Black     7.96M
Tito's Handmade Vodka       7.40M
Grey Goose Vodka            7.21M
Capt Morgan Spiced Rum      6.36M
Absolut 80 Proof            6.24M
Jameson Irish Whiskey       5.72M
Ketel One Vodka             5.07M
Baileys Irish Cream         4.15M
Kahlua                      3.60M
Tanqueray                   3.46M
Name: TotalSalesDollars, dtype: object
```

In [19]:
```
plt.figure(figsize =(15,5))
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')

plt.subplot(1, 2, 2)

ax2 = sns.barplot(y=top_brands.index.astype(str), x=top_brands.values, palette="Reds_r")
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()
```





In [20]:
```
vendor_performance = df.groupby('VendorName').agg({
    'TotalPurchaseDollars' : 'sum',
    'GrossProfit' : 'sum',
    'TotalSalesDollars' : 'sum'
}).reset_index()
vendor_performance.shape
```

Out[20]:
```
(119, 4)
```

In [21]:
```
vendor_performance['PurchaseContribution%'] = vendor_performance['TotalPurchaseDollars']/ vendor_performance['T
```

In [22]:
```
vendor_performance = round(vendor_performance.sort_values('PurchaseContribution%',ascending = False),2)
```

In [23]:
```
top_vendors = vendor_performance.head(10)
top_vendors['TotalSalesDollars'] = top_vendors['TotalSalesDollars'].apply(format_dollars)
top_vendors['TotalPurchaseDollars'] = top_vendors['TotalPurchaseDollars'].apply(format_dollars)
top_vendors['GrossProfit'] = top_vendors['GrossProfit'].apply(format_dollars)
top_vendors
```

Out[23]:

	VendorName	TotalPurchaseDollars	GrossProfit	TotalSalesDollars	PurchaseContribution%
25	DIAGEO NORTH AMERICA INC	50.10M	17.89M	67.99M	0.16
57	MARTIGNETTI COMPANIES	25.50M	13.83M	39.33M	0.08
68	PERNOD RICARD USA	23.85M	8.21M	32.06M	0.08
46	JIM BEAM BRANDS COMPANY	23.49M	7.93M	31.42M	0.08
6	BACARDI USA INC	17.43M	7.42M	24.85M	0.06
20	CONSTELLATION BRANDS INC	15.27M	8.95M	24.22M	0.05
11	BROWN-FORMAN CORP	13.24M	5.01M	18.25M	0.04
30	E & J GALLO WINERY	12.07M	6.33M	18.40M	0.04
106	ULTRA BEVERAGE COMPANY LLP	11.17M	5.34M	16.50M	0.04
53	M S WALKER INC	9.76M	4.94M	14.71M	0.03

In [24]:
```
top_vendors['PurchaseContribution%'].sum()
```

Out[24]:
```
np.float64(0.6600000000000001)
```

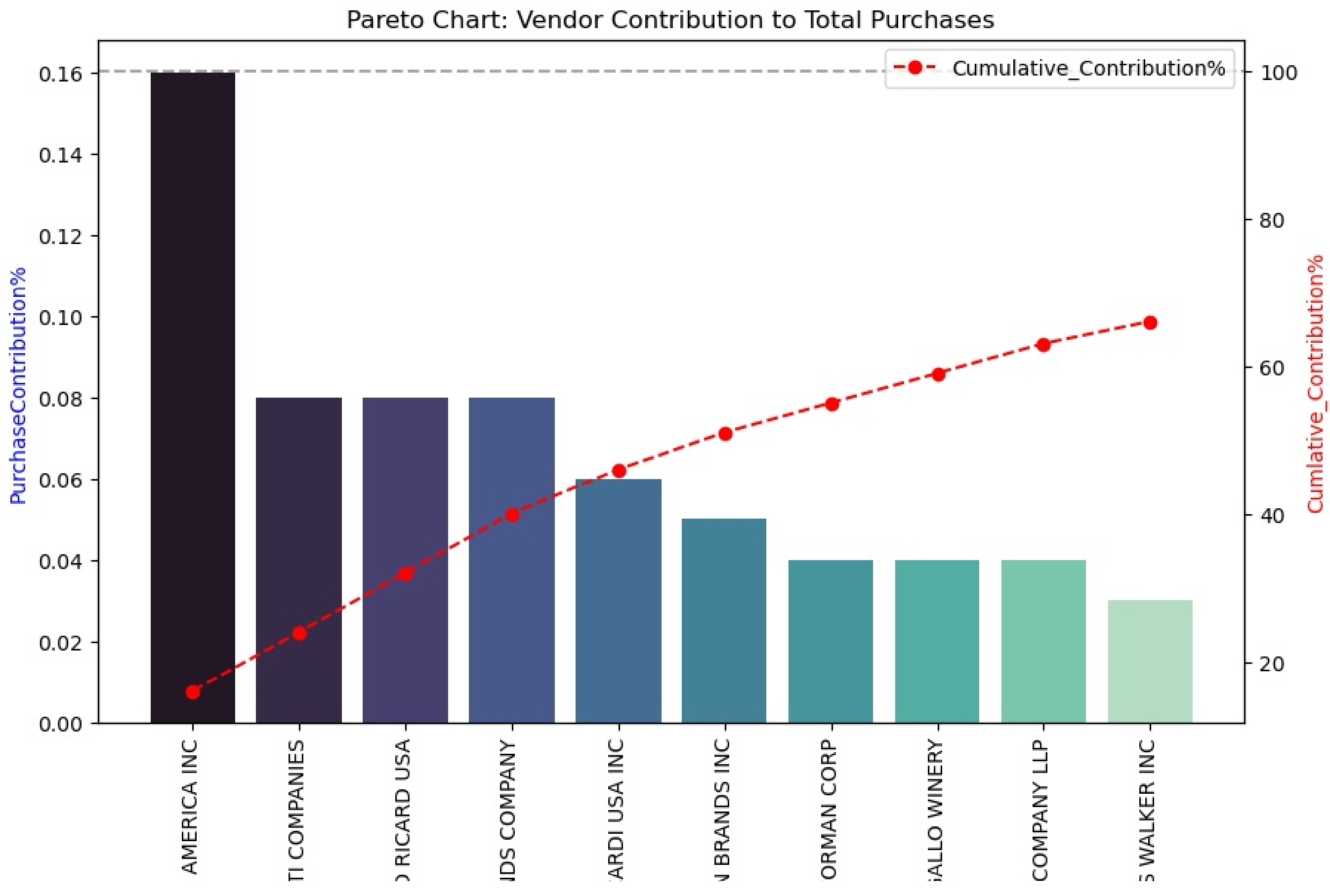
In [25]:
```
top_vendors['Cumlative_Contribution%'] = top_vendors['PurchaseContribution%'].cumsum()*100
top_vendors
```

Out[25]:

	VendorName	TotalPurchaseDollars	GrossProfit	TotalSalesDollars	PurchaseContribution%	Cumlative_Contribution%
25	DIAGEO NORTH AMERICA INC	50.10M	17.89M	67.99M	0.16	16.0
57	MARTIGNETTI COMPANIES	25.50M	13.83M	39.33M	0.08	24.0
68	PERNOD RICARD USA	23.85M	8.21M	32.06M	0.08	32.0
46	JIM BEAM BRANDS COMPANY	23.49M	7.93M	31.42M	0.08	40.0
6	BACARDI USA INC	17.43M	7.42M	24.85M	0.06	46.0
20	CONSTELLATION BRANDS INC	15.27M	8.95M	24.22M	0.05	51.0
11	BROWN-FORMAN CORP	13.24M	5.01M	18.25M	0.04	55.0
30	E & J GALLO WINERY	12.07M	6.33M	18.40M	0.04	59.0
106	ULTRA BEVERAGE COMPANY LLP	11.17M	5.34M	16.50M	0.04	63.0
53	M S WALKER INC	9.76M	4.94M	14.71M	0.03	66.0

In [26]:

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

sns.barplot(x=top_vendors['VendorName'], y=top_vendors['PurchaseContribution%'], palette="mako", ax=ax1)

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

ax2= ax1.twinx()
ax2.plot(top_vendors['VendorName'], top_vendors['Cumlative_Contribution%'], color='red', marker='o', linestyle=

ax1.set_xticklabels(top_vendors['VendorName'], rotation=90)
ax1.set_ylabel('PurchaseContribution%', color='blue')
ax2.set_ylabel('Cumlative_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()
```




DIAGEO NORTH
MARTIGNET
PERNOD
JIM BEAM BRAN
BAC
CONSTELLATION
BROWN-F
E & J G
ULTRA BEVERAGE
Vendors

In [27]:

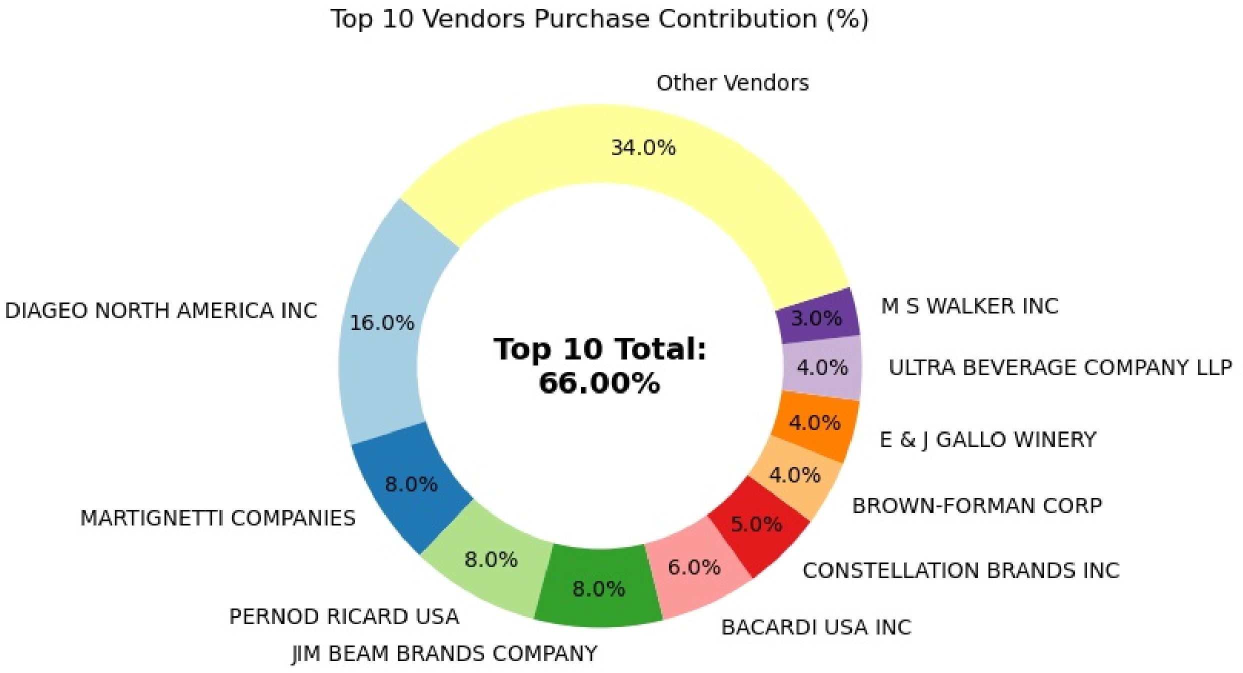
```
total = top_vendors['PurchaseContribution%'].sum() * 100
print(f"Total Purchase Contribution of top 10 vendors is {round(total, 2)} %")
```

```
Total Purchase Contribution of top 10 vendors is 66.0 %
```

In [28]:

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

# Convert PurchaseContribution% column to numeric
purchase_contributions = pd.to_numeric(top_vendors['PurchaseContribution%'], errors='coerce').fillna(0)
vendors = list(top_vendors['VendorName'].values)

# Convert from fraction to percentage
purchase_contributions = list(purchase_contributions * 100)

# Compute totals
total_contribution = float(np.sum(purchase_contributions))
remaining_contribution = float(100 - total_contribution)
```

```python
# Add 'Other Vendors'
vendors.append("Other Vendors")
purchase_contributions.append(remaining_contribution)

# Plot donut chart
fig, ax = plt.subplots(figsize=(8, 8))
wedges, texts, autotexts = ax.pie(
    purchase_contributions,
    labels=vendors,
    autopct='%1.1f%%',
    startangle=140,
    pctdistance=0.85,
    colors=plt.cm.Paired.colors
)

# Donut center
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig.gca().add_artist(centre_circle)

#  Correct f-string for formatted value
plt.text(
    0, 0,
    f"Top 10 Total:\n{total_contribution:.2f}%",
    fontsize=14,
    fontweight='bold',
    ha='center',
    va='center'
)

plt.title("Top 10 Vendors Purchase Contribution (%)")
plt.tight_layout()
plt.show()
```



In [29]:
```python
df['UnitPurchasePrice'] = df['TotalPurchaseDollars'] / df['TotalPurchaseQuantity']
```

In [30]:
```python
df["OrderSize"] = pd.qcut(df["TotalPurchaseQuantity"], q=3, labels=("Small", "Medium", "Large"))
```

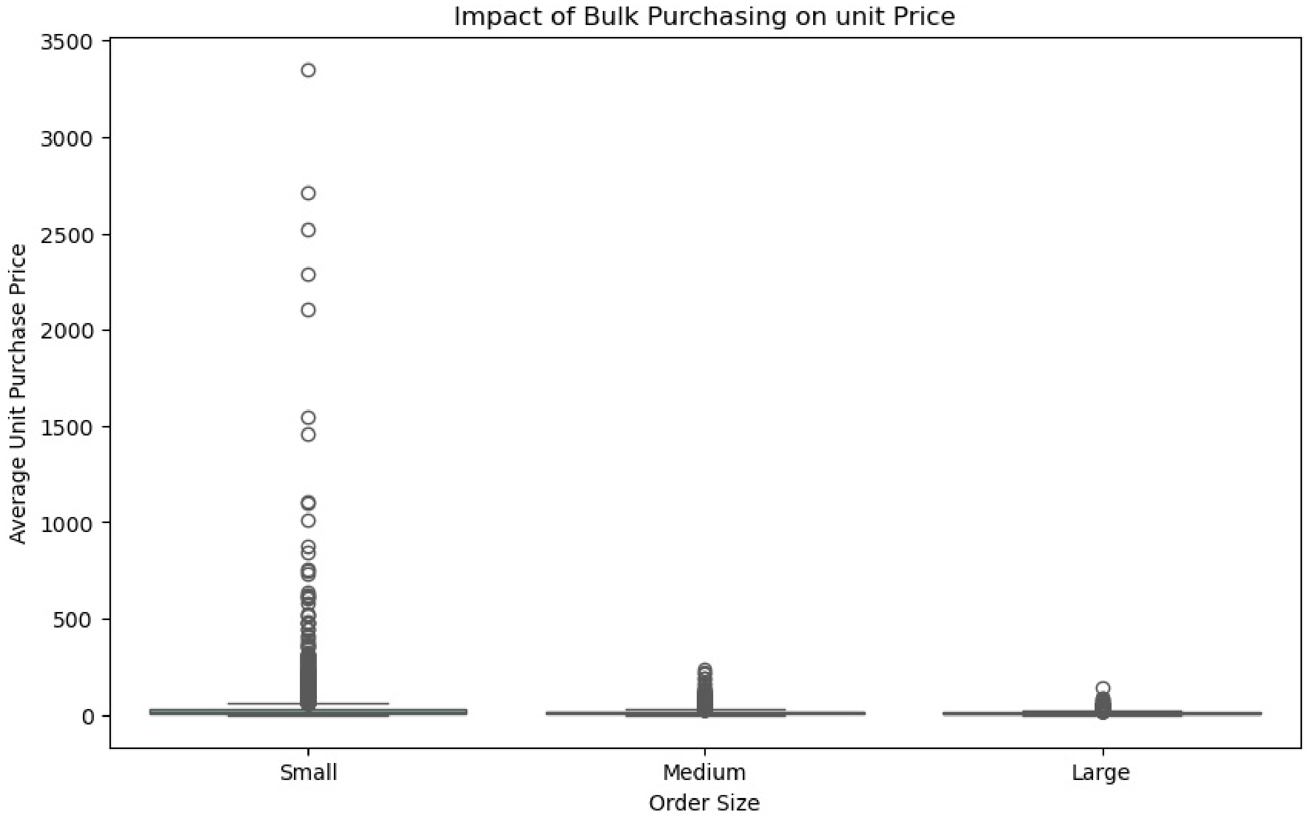
In [31]:
```python
df.groupby('OrderSize')[['UnitPurchasePrice']].mean()
```

Out[31]:

	UnitPurchasePrice
OrderSize	
Small	39.068186
Medium	15.486414
Large	10.777625

In [32]:
```python
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")
```

```python
plt.show()
```



In [33]:
```python
df[df['StockTurnover']<1].groupby('VendorName')[['StockTurnover']].mean().sort_values('StockTurnover',ascending
```

Out[33]:

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

In [34]:
```python
df["UnsoldInventoryValue"] = (df["TotalPurchaseQuantity"] - df["TotalSalesQuantity"]) * df["ActualPrice"]
print('Total Unsold Capital:', format_dollars(df["UnsoldInventoryValue"].sum()))
```

```
Total Unsold Capital: 3.71M
```

In [35]:
```python
inventory_value_per_vendor = df.groupby("VendorName")["UnsoldInventoryValue"].sum().reset_index()

inventory_value_per_vendor = inventory_value_per_vendor.sort_values(by="UnsoldInventoryValue", ascending=False)
inventory_value_per_vendor["UnsoldInventoryValue"] = inventory_value_per_vendor["UnsoldInventoryValue"].apply(f
inventory_value_per_vendor.head(10)
```

Out[35]:

	VendorName	UnsoldInventoryValue
25	DIAGEO NORTH AMERICA INC	984.23.K
46	JIM BEAM BRANDS COMPANY	761.21.K
68	PERNOD RICARD USA	647.40.K
116	WILLIAM GRANT & SONS INC	538.71.K
30	E & J GALLO WINERY	369.73.K
79	SAZERAC CO INC	273.37.K
11	BROWN-FORMAN CORP	247.06.K
20	CONSTELLATION BRANDS INC	227.37.K
61	MOET HENNESSY USA INC	197.64.K
54	MAJESTIC FINE WINES	180.73.K

In [36]:

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

In [37]:

```
top_vendors = df[df["TotalSalesDollars"] >= top_threshold]["ProfitMargin"].dropna()
low_vendors = df[df["TotalSalesDollars"] >= low_threshold]["ProfitMargin"].dropna()
```

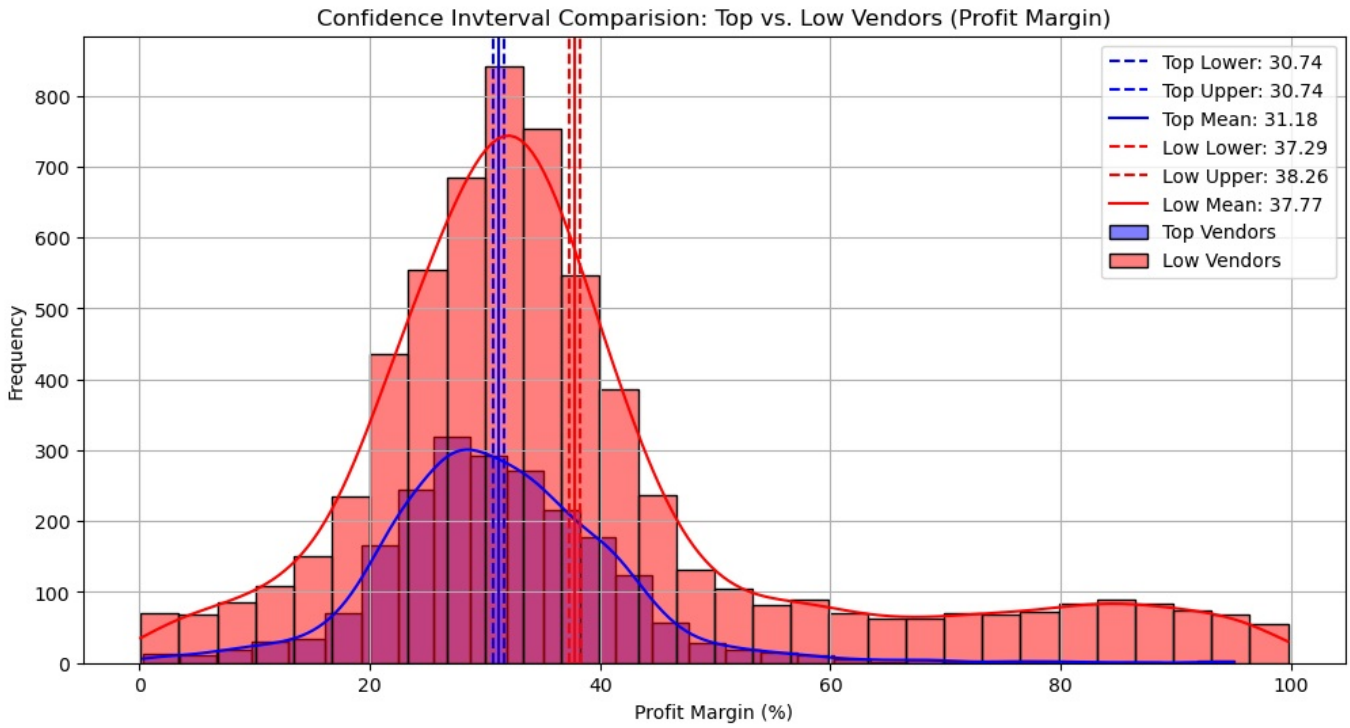
In [38]:

```
def confidence_interval(data, confidence=0.95):
    mean_val = np.mean(data)
    std_err = np.std(data, ddof=1) / np.sqrt(len(data))
    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
```

In [39]:

```
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:.2f}")
print(f"low Vendors 95% CI: ({low_lower:.2f}, {low_upper:.2f}), mean: {low_mean:.2f}")

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

sns.histplot(top_vendors, kde=True, color="blue", bins=30, alpha=0.5, label="Top Vendors")
plt.axvline(top_lower, color="blue", linestyle="--", label=f"Top Lower: {top_lower:.2f}")
plt.axvline(top_upper, color="blue", linestyle="--", label=f"Top Upper: {top_lower:.2f}")
plt.axvline(top_mean, color="blue", linestyle="-", label=f"Top Mean: {top_mean:.2f}")

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

plt.title("Confidence Invterval Comparision: Top vs. Low Vendors (Profit Margin)")
plt.xlabel("Profit Margin (%)")
plt.ylabel("Frequency")
plt.legend()
plt.grid(True)
plt.show()
```

```
Top Vendors 95% CI: (30.74, 31.61), mean: 31.18
low Vendors 95% CI: (37.29, 38.26), mean: 37.77
```



In [40]:
```
top_threshold = df["TotalSalesDollars"].quantile(0.75)
low_threshold = df["TotalSalesDollars"].quantile(0.25)

top_vendors = df[df["TotalSalesDollars"] >= top_threshold]["ProfitMargin"].dropna()
low_vendors = df[df["TotalSalesDollars"] >= low_threshold]["ProfitMargin"].dropna()

t_stat, p_value = ttest_ind(top_vendors, low_vendors, equal_var=False)

print(f"T-Statistic: {t_stat:.4f}, P-Value: {p_value:.4f}")
if p_value < 0.05:
    print("Reject He: There is a significant difference in profit margins between top and low_performing vendors
else:
    print("Fail to Reject He: No significant difference in profit margins.")
```

```
T-Statistic: -19.8217, P-Value: 0.0000
Reject He: There is a significant difference in profit margins between top and low_performing vendors.
```

In [43]:
```
pip install xgboost matplotlib scikit-learn pandas
```

```
Collecting xgboost
  Downloading xgboost-3.1.1-py3-none-win_amd64.whl.metadata (2.1 kB)
Requirement already satisfied: matplotlib in c:\users\shaik\anaconda3\lib\site-packages (3.10.0)
Requirement already satisfied: scikit-learn in c:\users\shaik\anaconda3\lib\site-packages (1.6.1)
Requirement already satisfied: pandas in c:\users\shaik\anaconda3\lib\site-packages (2.2.3)
Requirement already satisfied: numpy in c:\users\shaik\appdata\roaming\python\python313\site-packages (from xgbo
ost) (2.2.3)
Requirement already satisfied: scipy in c:\users\shaik\anaconda3\lib\site-packages (from xgboost) (1.15.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\shaik\anaconda3\lib\site-packages (from matplotlib)
(1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\shaik\anaconda3\lib\site-packages (from matplotlib) (0.1
1.0)
```

```
Requirement already satisfied: fonttools>=4.22.0 in c:\users\shaik\anaconda3\lib\site-packages (from matplotlib)
(4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\shaik\anaconda3\lib\site-packages (from matplotlib)
(1.4.8)
Requirement already satisfied: packaging>=20.0 in c:\users\shaik\appdata\roaming\python\python313\site-packages
(from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\shaik\anaconda3\lib\site-packages (from matplotlib) (11.1.0
)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\shaik\anaconda3\lib\site-packages (from matplotlib)
(3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\shaik\appdata\roaming\python\python313\site-pack
ages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: joblib>=1.2.0 in c:\users\shaik\anaconda3\lib\site-packages (from scikit-learn) (
1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\shaik\anaconda3\lib\site-packages (from scikit-l
earn) (3.5.0)
Requirement already satisfied: pytz>=2020.1 in c:\users\shaik\anaconda3\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\shaik\anaconda3\lib\site-packages (from pandas) (2025.
2)
Requirement already satisfied: six>=1.5 in c:\users\shaik\appdata\roaming\python\python313\site-packages (from p
ython-dateutil>=2.7->matplotlib) (1.17.0)
Downloading xgboost-3.1.1-py3-none-win_amd64.whl (72.0 MB)
   ---------------------------------------- 0.0/72.0 MB ? eta -:--:--
   ---------------------------------------- 0.0/72.0 MB ? eta -:--:--
   ---------------------------------------- 0.5/72.0 MB 2.1 MB/s eta 0:00:34
    --------------------------------------- 1.0/72.0 MB 2.4 MB/s eta 0:00:30
   - -------------------------------------- 1.8/72.0 MB 2.6 MB/s eta 0:00:27
   - -------------------------------------- 2.4/72.0 MB 2.7 MB/s eta 0:00:26
   - -------------------------------------- 3.1/72.0 MB 2.9 MB/s eta 0:00:25
   -- ------------------------------------- 3.9/72.0 MB 3.0 MB/s eta 0:00:23
   -- ------------------------------------- 4.7/72.0 MB 3.1 MB/s eta 0:00:22
   --- ------------------------------------ 5.5/72.0 MB 3.1 MB/s eta 0:00:22
   --- ------------------------------------ 6.3/72.0 MB 3.2 MB/s eta 0:00:21
   --- ------------------------------------ 6.8/72.0 MB 3.2 MB/s eta 0:00:21
   ---- ----------------------------------- 7.9/72.0 MB 3.3 MB/s eta 0:00:20
   ---- ----------------------------------- 8.7/72.0 MB 3.3 MB/s eta 0:00:19
   ----- ---------------------------------- 9.4/72.0 MB 3.4 MB/s eta 0:00:19
   ----- ---------------------------------- 10.2/72.0 MB 3.4 MB/s eta 0:00:19
   ------ --------------------------------- 11.0/72.0 MB 3.4 MB/s eta 0:00:18
   ------ --------------------------------- 11.8/72.0 MB 3.4 MB/s eta 0:00:18
   ------ --------------------------------- 12.6/72.0 MB 3.5 MB/s eta 0:00:18
   ------- -------------------------------- 13.4/72.0 MB 3.5 MB/s eta 0:00:17
   ------- -------------------------------- 14.2/72.0 MB 3.5 MB/s eta 0:00:17
   -------- ------------------------------- 14.9/72.0 MB 3.5 MB/s eta 0:00:17
   -------- ------------------------------- 15.7/72.0 MB 3.5 MB/s eta 0:00:17
   --------- ------------------------------ 16.5/72.0 MB 3.5 MB/s eta 0:00:16
   --------- ------------------------------ 17.3/72.0 MB 3.5 MB/s eta 0:00:16
   ---------- ----------------------------- 18.1/72.0 MB 3.5 MB/s eta 0:00:16
   ---------- ----------------------------- 18.9/72.0 MB 3.6 MB/s eta 0:00:15
   ---------- ----------------------------- 19.7/72.0 MB 3.5 MB/s eta 0:00:15
   ----------- ---------------------------- 20.4/72.0 MB 3.6 MB/s eta 0:00:15
   ----------- ---------------------------- 21.2/72.0 MB 3.6 MB/s eta 0:00:15
   ------------ --------------------------- 22.0/72.0 MB 3.6 MB/s eta 0:00:14
   ------------ --------------------------- 22.8/72.0 MB 3.6 MB/s eta 0:00:14
   ------------- -------------------------- 23.6/72.0 MB 3.6 MB/s eta 0:00:14
   ------------- -------------------------- 24.4/72.0 MB 3.6 MB/s eta 0:00:14
   ------------- -------------------------- 25.2/72.0 MB 3.6 MB/s eta 0:00:14
   -------------- ------------------------- 26.0/72.0 MB 3.6 MB/s eta 0:00:13
   -------------- ------------------------- 26.7/72.0 MB 3.6 MB/s eta 0:00:13
   --------------- ------------------------ 27.5/72.0 MB 3.6 MB/s eta 0:00:13
   --------------- ------------------------ 28.3/72.0 MB 3.6 MB/s eta 0:00:13
   ---------------- ----------------------- 29.1/72.0 MB 3.6 MB/s eta 0:00:12
   ---------------- ----------------------- 29.9/72.0 MB 3.6 MB/s eta 0:00:12
   ----------------- ---------------------- 30.7/72.0 MB 3.6 MB/s eta 0:00:12
   ----------------- ---------------------- 31.5/72.0 MB 3.6 MB/s eta 0:00:12
   ----------------- ---------------------- 32.2/72.0 MB 3.6 MB/s eta 0:00:12
   ------------------ --------------------- 33.0/72.0 MB 3.6 MB/s eta 0:00:11
   ------------------ --------------------- 33.8/72.0 MB 3.6 MB/s eta 0:00:11
   ------------------- -------------------- 34.6/72.0 MB 3.6 MB/s eta 0:00:11
   ------------------- -------------------- 35.4/72.0 MB 3.6 MB/s eta 0:00:11
   -------------------- ------------------- 36.2/72.0 MB 3.6 MB/s eta 0:00:10
   -------------------- ------------------- 37.2/72.0 MB 3.6 MB/s eta 0:00:10
   --------------------- ------------------ 38.0/72.0 MB 3.6 MB/s eta 0:00:10
   --------------------- ------------------ 38.8/72.0 MB 3.6 MB/s eta 0:00:10
   --------------------- ------------------ 39.6/72.0 MB 3.6 MB/s eta 0:00:09
   ---------------------- ----------------- 40.4/72.0 MB 3.6 MB/s eta 0:00:09
   ---------------------- ----------------- 41.2/72.0 MB 3.6 MB/s eta 0:00:09
   ----------------------- ---------------- 41.9/72.0 MB 3.6 MB/s eta 0:00:09
   ----------------------- ---------------- 42.7/72.0 MB 3.6 MB/s eta 0:00:09
   ------------------------ --------------- 43.5/72.0 MB 3.6 MB/s eta 0:00:08
   ------------------------ --------------- 44.3/72.0 MB 3.6 MB/s eta 0:00:08
   ------------------------- -------------- 45.1/72.0 MB 3.6 MB/s eta 0:00:08
   ------------------------- -------------- 45.9/72.0 MB 3.7 MB/s eta 0:00:08
```

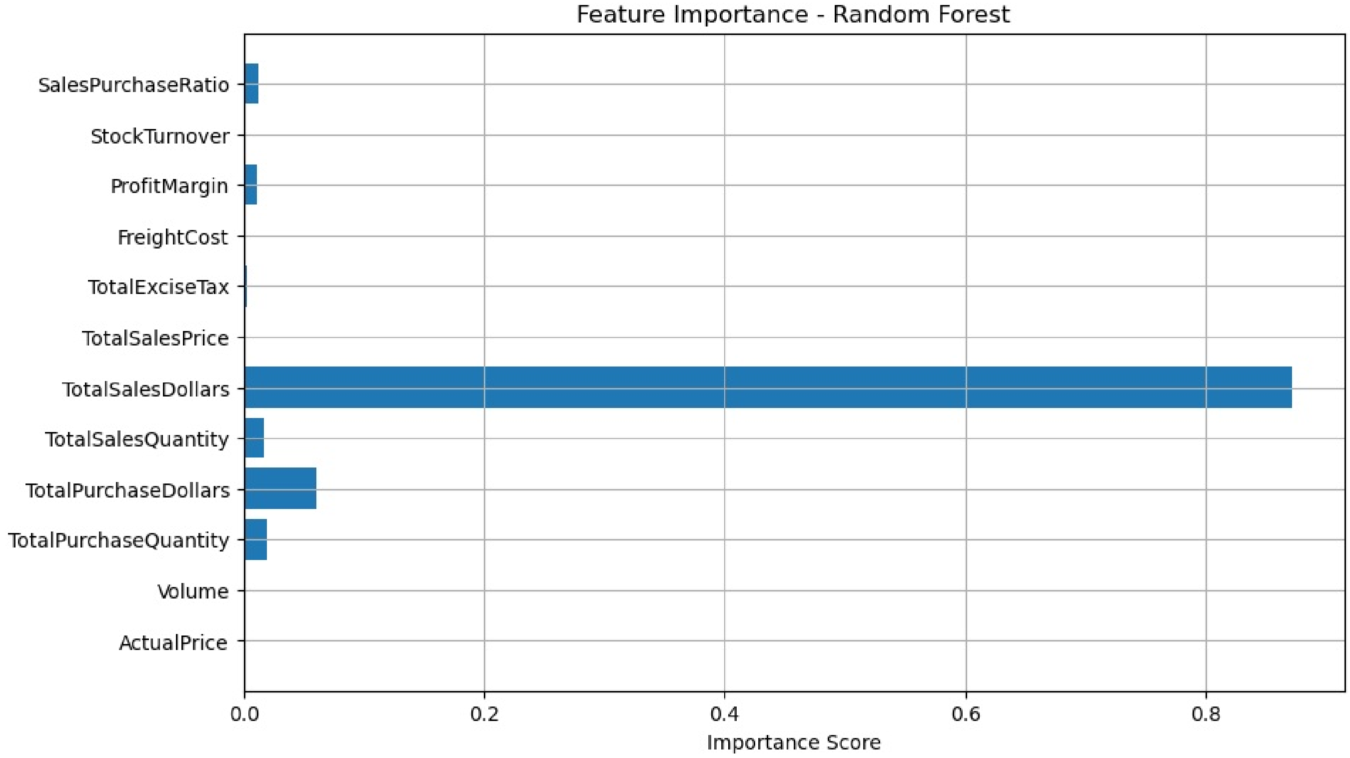
```
   ------------------------- -------------- 46.7/72.0 MB 3.7 MB/s eta 0:00:07
   -------------------------- ------------- 47.7/72.0 MB 3.7 MB/s eta 0:00:07
   -------------------------- ------------- 48.5/72.0 MB 3.7 MB/s eta 0:00:07
   --------------------------- ------------ 49.3/72.0 MB 3.7 MB/s eta 0:00:07
   --------------------------- ------------ 50.1/72.0 MB 3.7 MB/s eta 0:00:06
   ---------------------------- ----------- 50.9/72.0 MB 3.7 MB/s eta 0:00:06
   ---------------------------- ----------- 51.6/72.0 MB 3.7 MB/s eta 0:00:06
   ----------------------------- ---------- 52.4/72.0 MB 3.7 MB/s eta 0:00:06
   ----------------------------- ---------- 53.2/72.0 MB 3.7 MB/s eta 0:00:06
   ------------------------------ --------- 54.0/72.0 MB 3.7 MB/s eta 0:00:05
   ------------------------------ --------- 55.1/72.0 MB 3.7 MB/s eta 0:00:05
   ------------------------------- -------- 55.8/72.0 MB 3.7 MB/s eta 0:00:05
   ------------------------------- -------- 56.6/72.0 MB 3.7 MB/s eta 0:00:05
   ------------------------------- -------- 57.4/72.0 MB 3.7 MB/s eta 0:00:04
   -------------------------------- ------- 58.2/72.0 MB 3.7 MB/s eta 0:00:04
   -------------------------------- ------- 59.0/72.0 MB 3.7 MB/s eta 0:00:04
   --------------------------------- ------ 59.8/72.0 MB 3.7 MB/s eta 0:00:04
   --------------------------------- ------ 60.8/72.0 MB 3.7 MB/s eta 0:00:04
   ---------------------------------- ----- 61.6/72.0 MB 3.7 MB/s eta 0:00:03
   ---------------------------------- ----- 62.4/72.0 MB 3.7 MB/s eta 0:00:03
   ----------------------------------- ---- 63.2/72.0 MB 3.7 MB/s eta 0:00:03
   ----------------------------------- ---- 64.0/72.0 MB 3.7 MB/s eta 0:00:03
   ----------------------------------- ---- 64.7/72.0 MB 3.7 MB/s eta 0:00:02
   ------------------------------------ --- 65.5/72.0 MB 3.7 MB/s eta 0:00:02
   ------------------------------------ --- 66.3/72.0 MB 3.7 MB/s eta 0:00:02
   ------------------------------------- -- 67.1/72.0 MB 3.7 MB/s eta 0:00:02
   ------------------------------------- -- 68.2/72.0 MB 3.7 MB/s eta 0:00:02
   -------------------------------------- - 68.9/72.0 MB 3.7 MB/s eta 0:00:01
   -------------------------------------- - 69.7/72.0 MB 3.7 MB/s eta 0:00:01
   ---------------------------------------  70.5/72.0 MB 3.7 MB/s eta 0:00:01
   ---------------------------------------  71.3/72.0 MB 3.7 MB/s eta 0:00:01
   ---------------------------------------  71.8/72.0 MB 3.7 MB/s eta 0:00:01
   ---------------------------------------- 72.0/72.0 MB 3.7 MB/s eta 0:00:00
Installing collected packages: xgboost
Successfully installed xgboost-3.1.1
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 25.0.1 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
```

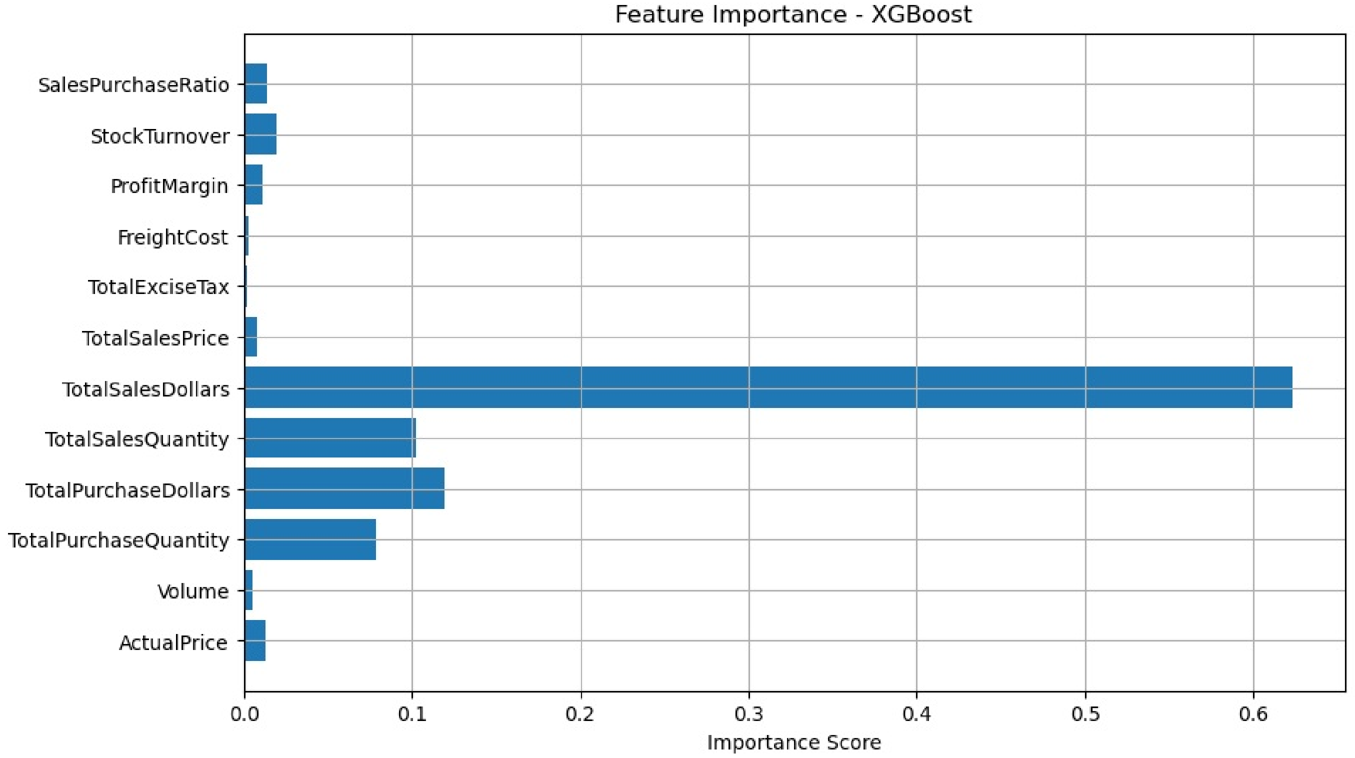
In [49]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error


# Features + Target
X = df[['ActualPrice','Volume','TotalPurchaseQuantity','TotalPurchaseDollars',
        'TotalSalesQuantity','TotalSalesDollars','TotalSalesPrice',
        'TotalExciseTax','FreightCost','ProfitMargin','StockTurnover',
        'SalesPurchaseRatio']]

y = df['GrossProfit']

# 2. TRAIN TEST SPLIT
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# 3. SCALING (for Linear Regression & XGBoost)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. MODELS
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=200, random_state=42),
    "XGBoost": XGBRegressor(n_estimators=300, learning_rate=0.05,
                            max_depth=5, subsample=0.9, colsample_bytree=0.9)
}

predictions = {}
scores = []

# Train & Predict
for name, model in models.items():
    if name == "Linear Regression" or name == "XGBoost":
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
```

```
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    predictions[name] = y_pred

    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    scores.append([name, r2, mae, rmse])

scores_df = pd.DataFrame(scores, columns=["Model", "R2 Score", "MAE", "RMSE"])
print("\n\n===== MODEL ACCURACY COMPARISON =====")
print(scores_df)


# GRAPH C: Feature Importance (Random Forest)
rf_model = models["Random Forest"]
plt.figure(figsize=(10,6))
plt.barh(X.columns, rf_model.feature_importances_)
plt.title("Feature Importance - Random Forest")
plt.xlabel("Importance Score")
plt.grid(True)
plt.show()

# GRAPH D: Feature Importance (XGBoost)
xgb_model = models["XGBoost"]
plt.figure(figsize=(10,6))
plt.barh(X.columns, xgb_model.feature_importances_)
plt.title("Feature Importance - XGBoost")
plt.xlabel("Importance Score")
plt.grid(True)
plt.show()
```

```
===== MODEL ACCURACY COMPARISON =====
               Model  R2 Score           MAE          RMSE
0  Linear Regression  1.000000  4.516691e-11  9.665176e-11
1      Random Forest  0.989374  6.522354e+02  4.675393e+03
2            XGBoost  0.977651  8.403499e+02  6.780527e+03
```

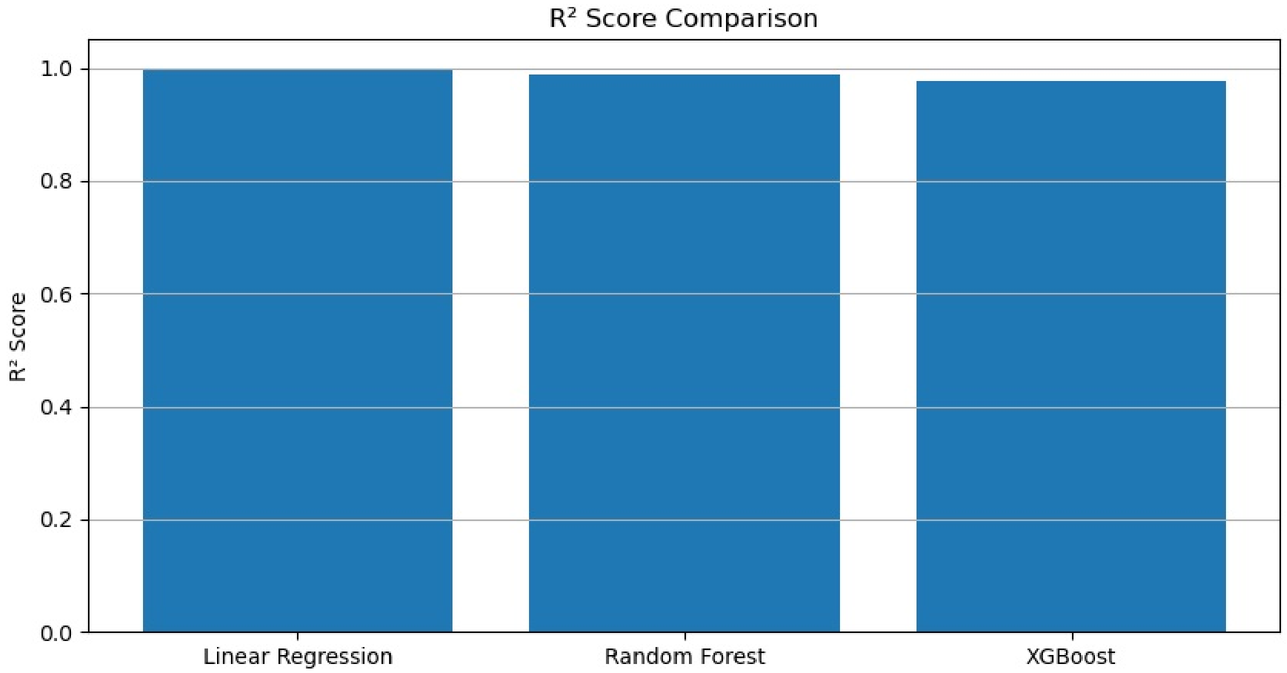


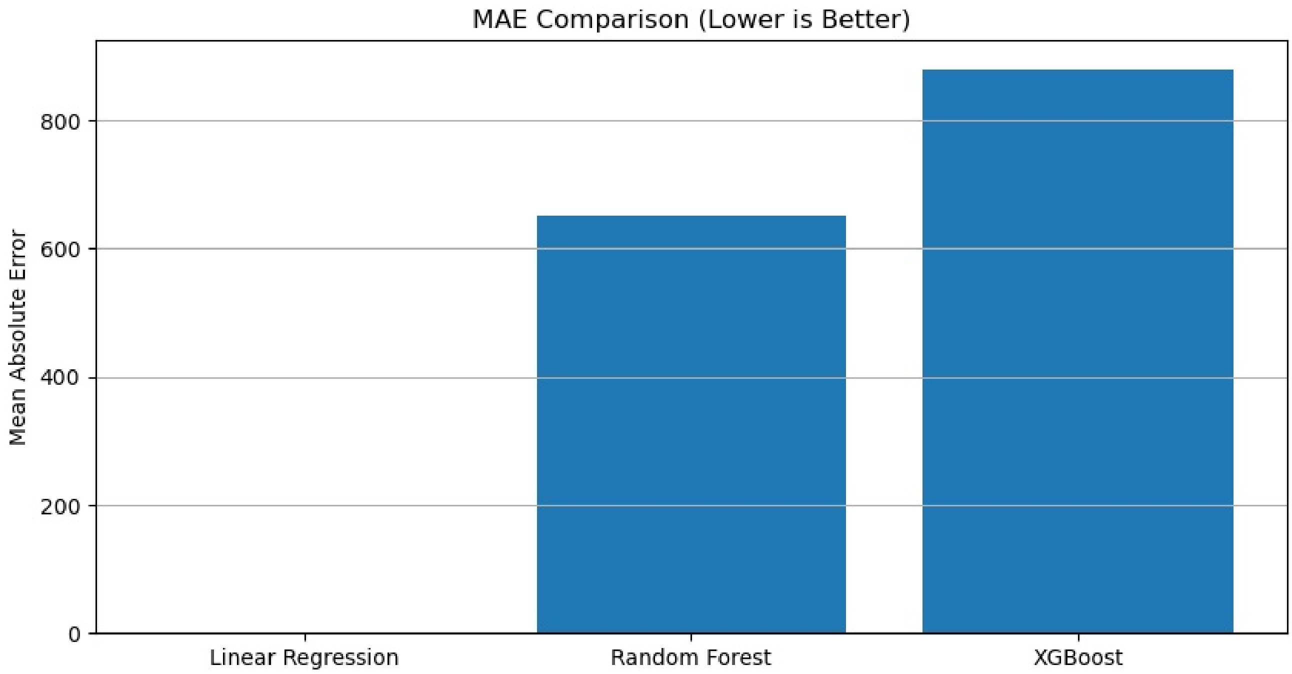


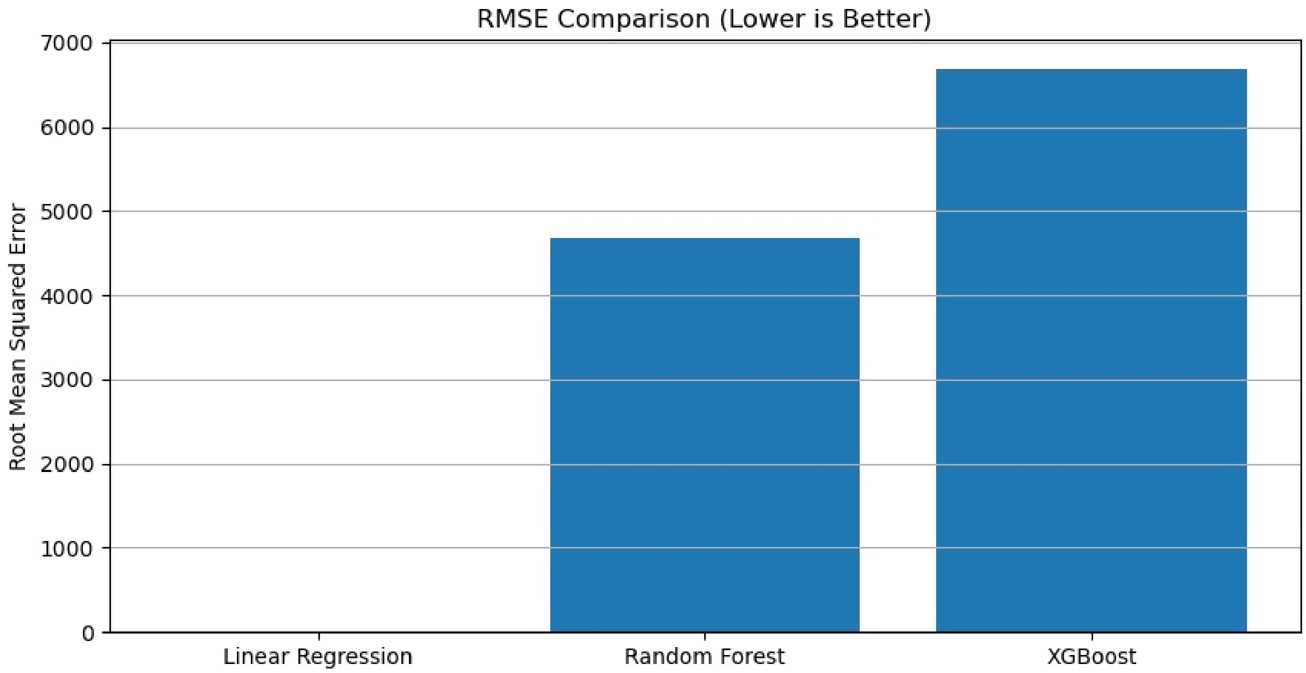
In [48]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor

X = df[['ActualPrice','Volume','TotalPurchaseQuantity','TotalPurchaseDollars',
        'TotalSalesQuantity','TotalSalesDollars','TotalSalesPrice',
        'TotalExciseTax','FreightCost','ProfitMargin','StockTurnover',
        'SalesPurchaseRatio']]

y = df['GrossProfit']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=200, random_state=42),
    "XGBoost": XGBRegressor(n_estimators=300, learning_rate=0.05, max_depth=5)
}

r2_scores = []
mae_scores = []
rmse_scores = []
model_names = []

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    model_names.append(name)
    r2_scores.append(r2_score(y_test, y_pred))
    mae_scores.append(mean_absolute_error(y_test, y_pred))
    rmse_scores.append(np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
print("\n===== MODEL ACCURACY RESULTS =====")
for i in range(len(models)):
    print(f"{model_names[i]} → R2: {r2_scores[i]},  MAE: {mae_scores[i]}, RMSE: {rmse_scores[i]}")

# GRAPH 1: R2 SCORE
plt.figure(figsize=(10,5))
plt.bar(model_names, r2_scores)
plt.title("R² Score Comparison")
plt.ylabel("R² Score")
plt.grid(axis='y')
plt.show()

# GRAPH 2: MAE
plt.figure(figsize=(10,5))
plt.bar(model_names, mae_scores)
plt.title("MAE Comparison (Lower is Better)")
plt.ylabel("Mean Absolute Error")
plt.grid(axis='y')
plt.show()

#GRAPH 3: RMSE
plt.figure(figsize=(10,5))
plt.bar(model_names, rmse_scores)
plt.title("RMSE Comparison (Lower is Better)")
plt.ylabel("Root Mean Squared Error")
plt.grid(axis='y')
plt.show()
```

```
===== MODEL ACCURACY RESULTS =====
Linear Regression →  R2: 1.0,  MAE: 3.8426246034115676e-11, RMSE: 7.004272499981352e-11
Random Forest →  R2: 0.9893739477200437,  MAE: 652.2354249854048, RMSE: 4675.393479606263
XGBoost →  R2: 0.9782272028226651,  MAE: 880.5424324083721, RMSE: 6692.514274353861
```





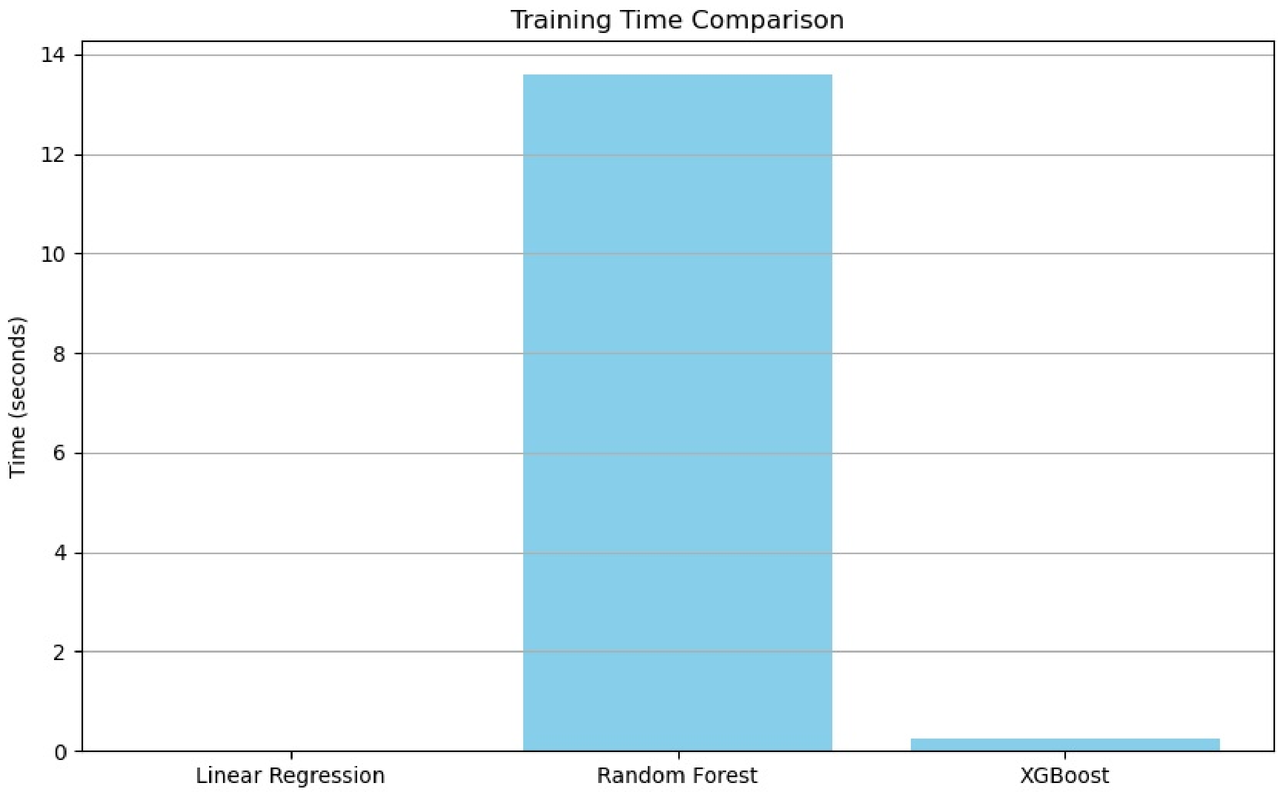


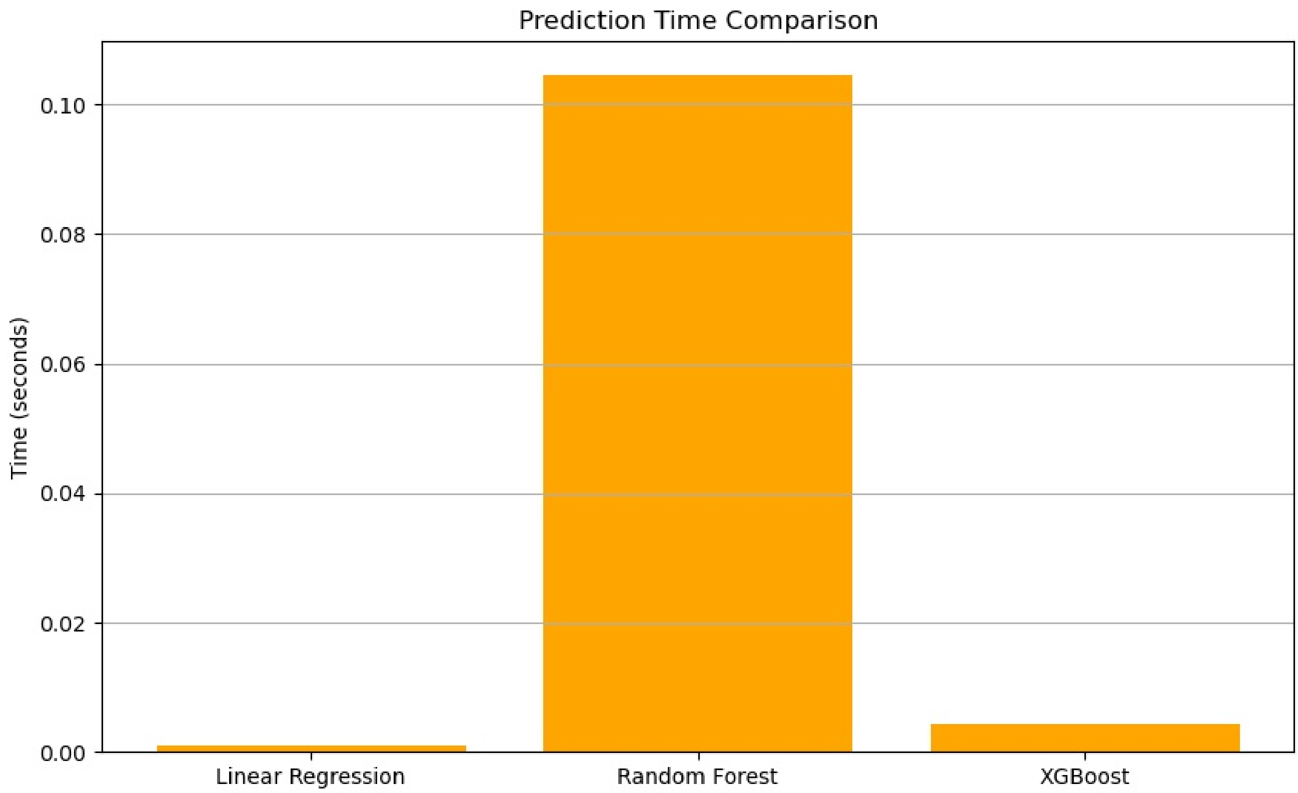
In [52]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
from sklearn.model_selection import train_test_split
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error


# Features
X = df[['ActualPrice','Volume','TotalPurchaseQuantity','TotalPurchaseDollars',
        'TotalSalesQuantity','TotalSalesDollars','TotalSalesPrice',
        'TotalExciseTax','FreightCost','ProfitMargin','StockTurnover',
        'SalesPurchaseRatio']]

# Target
y = df['GrossProfit']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Models
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=200, random_state=42),
    "XGBoost": XGBRegressor(n_estimators=300, learning_rate=0.05, max_depth=5)
}

train_times = []
predict_times = []
model_names = []

# Measure time for each model
for name, model in models.items():
    model_names.append(name)

    # Training time
    start_train = time.time()
    model.fit(X_train, y_train)
    end_train = time.time()

    train_time = end_train - start_train
    train_times.append(train_time)

    # Prediction time
    start_pred = time.time()
    model.predict(X_test)
    end_pred = time.time()

    pred_time = end_pred - start_pred
    predict_times.append(pred_time)

    print(f"{name}: Training Time = {train_time:.4f}s, Prediction Time = {pred_time:.4f}s")

# GRAPH 1: TRAINING TIME
plt.figure(figsize=(10,6))
plt.bar(model_names, train_times, color='skyblue')
plt.title("Training Time Comparison")
plt.ylabel("Time (seconds)")
plt.grid(axis="y")
plt.show()

#GRAPH 2: PREDICTION TIME
plt.figure(figsize=(10,6))
plt.bar(model_names, predict_times, color='orange')
plt.title("Prediction Time Comparison")
plt.ylabel("Time (seconds)")
plt.grid(axis="y")
plt.show()
```

```
Linear Regression: Training Time = 0.0101s, Prediction Time = 0.0010s
Random Forest: Training Time = 13.5814s, Prediction Time = 0.1045s
XGBoost: Training Time = 0.2413s, Prediction Time = 0.0043s
```


Training Time Comparison
14
12
10
8
6
4
2
0
Time (seconds)
Linear Regression
Random Forest
XGBoost


Prediction Time Comparison
0.10
0.08
0.06
0.04
0.02
0.00
Time (seconds)
Linear Regression
Random Forest
XGBoost

In [ ]: