

```

import kagglehub
import os
import shutil

if False:
    # Download latest version of the dataset
    download_temp_path =
    kagglehub.dataset_download("sloozecareers/slooze-challenge")

    # User-specified destination path
    destination_path = "/content/drive/MyDrive/Colab
Notebooks/Slooze/datasets"

    # Create the destination directory if it does not exist
    os.makedirs(destination_path, exist_ok=True)

    # Copy the contents of the downloaded directory to the destination
    path
    # This ensures the dataset files are directly within the specified
    folder
    for item in os.listdir(download_temp_path):
        source_item_path = os.path.join(download_temp_path, item)
        destination_item_path = os.path.join(destination_path, item)

        if os.path.isdir(source_item_path):
            # Use copytree for directories. dirs_exist_ok=True allows
            merging if directory already exists.
            shutil.copytree(source_item_path, destination_item_path,
            dirs_exist_ok=True)
        else:
            # Use copy2 for files to preserve metadata
            shutil.copy2(source_item_path, destination_item_path)

    print(f"Dataset successfully downloaded and copied to:
{destination_path}")

```

START

Import Libraries

```

# Data manipulation
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

```

```

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Date handling
from datetime import datetime, timedelta

# Statistical analysis
from scipy import stats

# Machine Learning & Forecasting
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
from prophet import Prophet

# Kaggle dataset download
import kagglehub

# File system
import os

# Display settings
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)
plt.style.use('seaborn-v0_8-darkgrid')

print("✅ All libraries imported successfully!")
✅ All libraries imported successfully!

```

Download Dataset from Kaggle

```

import kagglehub

print("✅ Downloading Slooze Challenge dataset from Kaggle...\n")

# Download latest version
path = kagglehub.dataset_download("sloozecareers/slooze-challenge")

print("\n" + "="*60)
print("✅ Dataset downloaded successfully!")
print("="*60)
print(f"✅ Path to dataset files: {path}")

✅ Downloading Slooze Challenge dataset from Kaggle...

```

Using Colab cache for faster access to the 'slooze-challenge' dataset.

```
=====
[] Dataset downloaded successfully!
=====
```

```
[] Path to dataset files: /kaggle/input/slooze-challenge
```

List All Files in Dataset

```
# The files are actually in a subfolder
actual_path = os.path.join(path, 'slooze_challenge')

print("[] Checking actual file structure...\n")
print(f"Main path: {path}")
print(f"Subfolder path: {actual_path}\n")

# List files in the subfolder
print("[] Files in slooze_challenge folder:\n")

files = os.listdir(actual_path)
for file in sorted(files):
    file_path = os.path.join(actual_path, file)
    size_mb = os.path.getsize(file_path) / (1024 * 1024)
    print(f"  [] {file:<35} ({size_mb:>7.2f} MB)")

print("\n" + "="*60)

[] Checking actual file structure...

Main path: /kaggle/input/slooze-challenge
Subfolder path: /kaggle/input/slooze-challenge/slooze_challenge

[] Files in slooze_challenge folder:

  [] 2017PurchasePricesDec.csv          (  1.10 MB)
  [] BegInvFINAL12312016.csv           ( 18.41 MB)
  [] EndInvFINAL12312016.csv           ( 20.03 MB)
  [] InvoicePurchases12312016.csv       (  0.56 MB)
  [] PurchasesFINAL12312016.csv        (383.14 MB)
  [] SalesFINAL12312016.csv            (121.94 MB)
```

Define File Paths

```
# CORRECTED: Files are in the slooze_challenge subfolder
file_paths = {
```

```

    'purchase_prices': os.path.join(actual_path,
'2017PurchasePricesDec.csv'),
    'beg_inventory': os.path.join(actual_path,
'BegInvFINAL12312016.csv'),
    'end_inventory': os.path.join(actual_path,
'EndInvFINAL12312016.csv'),
    'invoice_purchases': os.path.join(actual_path,
'InvoicePurchases12312016.csv'),
    'purchases': os.path.join(actual_path,
'PurchasesFINAL12312016.csv'),
    'sales': os.path.join(actual_path, 'SalesFINAL12312016.csv')
}

print("□ File paths configured correctly!")
print("\n□ Files to load:")
for key, file_path in file_paths.items():
    exists = "□" if os.path.exists(file_path) else "□"
    print(f"    {exists} {key:<20}: {os.path.basename(file_path)}")

```

□ File paths configured correctly!

□ Files to load:

□ purchase_prices	: 2017PurchasePricesDec.csv
□ beg_inventory	: BegInvFINAL12312016.csv
□ end_inventory	: EndInvFINAL12312016.csv
□ invoice_purchases	: InvoicePurchases12312016.csv
□ purchases	: PurchasesFINAL12312016.csv
□ sales	: SalesFINAL12312016.csv

Load All CSV Files

```

print("\n□ Loading datasets... This may take 2-3 minutes for large
files.\n")
print("="*60)

# Dictionary to store all dataframes
data = {}

# Load each file with progress updates
for key, file_path in file_paths.items():
    print(f"Loading {key}...", end=" ")
    try:
        data[key] = pd.read_csv(file_path, low_memory=False)
        rows, cols = data[key].shape
        print(f"□ Loaded! {rows:,} rows × {cols} columns")
    except Exception as e:
        print(f"□ Error: {e}")

```

```

print("="*60)

# Check if all files loaded successfully
if len(data) == 6:
    print("✅ ALL FILES LOADED SUCCESSFULLY!")
else:
    print(f"⚠️ WARNING: Only {len(data)}/6 files loaded!")

print("="*60)

```

✅ Loading datasets... This may take 2-3 minutes for large files.

```

=====
Loading purchase_prices... ✅ Loaded! 12,261 rows × 9 columns
Loading beg_inventory... ✅ Loaded! 206,529 rows × 9 columns
Loading end_inventory... ✅ Loaded! 224,489 rows × 9 columns
Loading invoice_purchases... ✅ Loaded! 5,543 rows × 10 columns
Loading purchases... ✅ Loaded! 2,372,474 rows × 16 columns
Loading sales... ✅ Loaded! 1,048,575 rows × 14 columns
=====
✅ ALL FILES LOADED SUCCESSFULLY!
=====

```

Quick Overview of Each Dataset

```

print("\n✅ DATASET OVERVIEW\n")

for key, df in data.items():
    print("="*60)
    print(f"✅ {key.upper().replace('_', ' ')}")
    print("="*60)
    print(f"Shape: {df.shape[0]:,} rows × {df.shape[1]} columns")
    print(f"Memory: {df.memory_usage(deep=True).sum() / 1024**2:.2f} MB")
    print(f"\nColumns ({len(df.columns)}):")
    print(f"{list(df.columns)}")
    print(f"\n✅ First 3 rows:")
    display(df.head(3))
    print("\n")

```

✅ DATASET OVERVIEW

```

=====
✅ PURCHASE PRICES
=====

```

Shape: 12,261 rows × 9 columns
Memory: 3.43 MB

```
['Brand', 'Description', 'Price', 'Size', 'Volume', 'Classification',  
'PurchasePrice', 'VendorNumber', 'VendorName']
```

[illegible]

```

{"num_unique_values": 2, "samples": [{"BROWN-
FORMAN CORP", "semantic_type": "\"\"",
"description": "\"\""}], "type": "dataframe"}

```

□ BEG INVENTORY

```
Shape: 206,529 rows x 9 columns
Memory: 66.94 MB
```

```
Columns (9):  
['InventoryId', 'Store', 'City', 'Brand', 'Description', 'Size',  
'onHand', 'Price', 'startDate']
```

□ First 3 rows:

[illegible]


```

n      \ "semantic_type\": \ "\",\n          \ "description\": \ "\n
}\n    },\n    {\n        \ "column\": \ "Size\",\n        \ "properties\":
{\n        \ "dtype\": \ "category",\n        \ "num_unique_values\":
1,\n        \ "samples\": [\n            \ "750mL",\n        ],\n
\ "semantic_type\": \ "\",\n        \ "description\": \ "\n    }\n
n    },\n    {\n        \ "column\": \ "onHand",\n        \ "properties\":
{\n        \ "dtype\": \ "number",\n        \ "std\": 2,\n
\ "min\": 7,\n        \ "max\": 11,\n        \ "num_unique_values\": 2,\n
\ "samples\": [\n            7,\n        ],\n        \ "semantic_type\":
\ "\",\n        \ "description\": \ "\n    },\n    {\n
\ "column\": \ "Price",\n        \ "properties\": {\n        \ "dtype\":
\ "number",\n        \ "std\": 14.46835627614047,\n        \ "min\":
12.99,\n        \ "max\": 38.99,\n        \ "num_unique_values\": 3,\n
\ "samples\": [\n            12.99,\n        ],\n
\ "semantic_type\": \ "\",\n        \ "description\": \ "\n    }\n
n    },\n    {\n        \ "column\": \ "endDate",\n        \ "properties\":
{\n        \ "dtype\": \ "object",\n        \ "num_unique_values\": 1,\n
\ "samples\": [\n            \ "2016-12-31",\n        ],\n
\ "semantic_type\": \ "\",\n        \ "description\": \ "\n    }\n
n    }\n ]\n }","type":"dataframe"}

```

```

=====
□ INVOICE PURCHASES
=====

```

```

Shape: 5,543 rows × 10 columns
Memory: 1.73 MB

```

```

Columns (10):
['VendorNumber', 'VendorName', 'InvoiceDate', 'PONumber', 'PODate',
'PayDate', 'Quantity', 'Dollars', 'Freight', 'Approval']

```

```

□ First 3 rows:

```

```

{"repr_error":"Out of range float values are not JSON compliant:
nan","type":"dataframe"}

```

```

=====
□ PURCHASES
=====

```

```

Shape: 2,372,474 rows × 16 columns
Memory: 1278.97 MB

```

```

Columns (16):
['InventoryId', 'Store', 'Brand', 'Description', 'Size',
'VendorNumber', 'VendorName', 'PONumber', 'PODate', 'ReceivingDate',
'InvoiceDate', 'PayDate', 'PurchasePrice', 'Quantity', 'Dollars',
'Classification']

```

□ First 3 rows:

[illegible]

```

{"properties": {"dtype": "object",
  "num_unique_values": 2,
  "samples": ["2016-01-01", "2016-01-02"],
  "semantic_type": "",
  "description": ""},
  "column": "InvoiceDate",
  "properties": {"dtype": "object",
    "num_unique_values": 2,
    "samples": ["2016-01-07", "2016-01-04"],
    "semantic_type": "",
    "description": ""},
    "column": "PayDate",
    "properties": {"dtype": "object",
      "num_unique_values": 2,
      "samples": ["2016-02-21", "2016-02-16"],
      "semantic_type": "",
      "description": ""},
      "column": "PurchasePrice",
      "properties": {"dtype": "number",
        "std": 15.201662189817709,
        "min": 9.35,
        "max": 35.71,
        "num_unique_values": 3,
        "samples": [9.35, 35.71],
        "semantic_type": "",
        "description": ""},
        "column": "Quantity",
        "properties": {"dtype": "number",
          "std": 1,
          "min": 4,
          "max": 6,
          "num_unique_values": 3,
          "samples": [6, 4],
          "semantic_type": "",
          "description": ""},
          "column": "Dollars",
          "properties": {"dtype": "number",
            "std": 99.44157933178656,
            "min": 37.4,
            "max": 214.26,
            "num_unique_values": 3,
            "samples": [214.26, 37.4],
            "semantic_type": "",
            "description": ""},
            "column": "Classification",
            "properties": {"dtype": "number",
              "std": 0,
              "min": 1,
              "max": 1,
              "num_unique_values": 1,
              "samples": [1],
              "semantic_type": "",
              "description": ""}
        ],
        "type": "dataframe"}

```

```

=====
SALES
=====

```

```

Shape: 1,048,575 rows x 14 columns
Memory: 395.01 MB

```

```
Columns (14):
```

```

['InventoryId', 'Store', 'Brand', 'Description', 'Size',
'SalesQuantity', 'SalesDollars', 'SalesPrice', 'SalesDate', 'Volume',
'Classification', 'ExciseTax', 'VendorNo', 'VendorName']

```

```

First 3 rows:

```

```

{"summary":{"name":"Inventory","rows":3,"fields":{"column":"InventoryId","properties":{"dtype":"category","num_unique_values":1,"samples":["1_HARDERSFIELD_1004"],"semantic_type":"","description":"","column":"Store","properties":{"dtype":"number","std":0,"min":1,"max":1,"num_unique_values":1,"samples":["1"],"semantic_type":"","description":"","column":"Brand","properties":{"dtype":"number","std":0,"min":1004,"max":1004,"num_unique_values":1,"samples":["1004"],"semantic_type":"","description":"","column":"Description","properties":{"dtype":"category","num_unique_values":1,"samples":["Jim Beam w/2 Rocks Glasses"],"semantic_type":"","description":"","column":"Size","properties":{"dtype":"category","num_unique_values":1,"samples":["750mL"],"semantic_type":"","description":"","column":"SalesQuantity","properties":{"dtype":"number","std":0,"min":1,"max":2,"num_unique_values":2,"samples":["2"],"semantic_type":"","description":"","column":"SalesDollars","properties":{"dtype":"number","std":9.520505938936928,"min":16.49,"max":32.98,"num_unique_values":2,"samples":["32.98"],"semantic_type":"","description":"","column":"SalesPrice","properties":{"dtype":"number","std":0.0,"min":16.49,"max":16.49,"num_unique_values":1,"samples":["16.49"],"semantic_type":"","description":"","column":"SalesDate","properties":{"dtype":"object","num_unique_values":3,"samples":["1/1/2016"],"semantic_type":"","description":"","column":"Volume","properties":{"dtype":"number","std":0,"min":750,"max":750,"num_unique_values":1,"samples":["750"],"semantic_type":"","description":"","column":"Classification","properties":{"dtype":"number","std":

```

```

0,\n          \"min\": 1,\n          \"max\": 1,\n          \"num_unique_values\": 1,\n          \"samples\": [\n          1\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\"\n        },\n        {\n          \"column\": \"ExciseTax\",\n          \"properties\": {\n            \"dtype\": \"number\",\n            \"std\": 0.4503332099679081,\n            \"min\": 0.79,\n            \"max\": 1.57,\n            \"num_unique_values\": 2,\n            \"samples\": [\n            1.57\n            ],\n            \"semantic_type\": \"\",\n            \"description\": \"\"\n          },\n          {\n            \"column\": \"VendorNo\",\n            \"properties\": {\n              \"dtype\": \"number\",\n              \"std\": 0,\n              \"min\": 12546,\n              \"max\": 12546,\n              \"num_unique_values\": 1,\n              \"samples\": [\n              12546\n              ],\n              \"semantic_type\": \"\",\n              \"description\": \"\"\n            },\n            {\n              \"column\": \"VendorName\",\n              \"properties\": {\n                \"dtype\": \"category\",\n                \"num_unique_values\": 1,\n                \"samples\": [\n                \"JIM BEAM BRANDS COMPANY\"\n                ],\n                \"semantic_type\": \"\",\n                \"description\": \"\"\n              }\n            }\n          }\n        },\n        {\n          \"column\": \"VendorName\",\n          \"properties\": {\n            \"dtype\": \"category\",\n            \"num_unique_values\": 1,\n            \"samples\": [\n            \"JIM BEAM BRANDS COMPANY\"\n            ],\n            \"semantic_type\": \"\",\n            \"description\": \"\"\n          }\n        }\n      ],\n      \"type\": \"dataframe\"

```

Detailed Data Profiling

```

def profile_dataset(df, name):
    """
    Generate a detailed profile report for a dataset
    """
    print("="*70)
    print(f" DETAILED PROFILE: {name.upper()}")
    print("="*70)

    # Basic info
    print(f"\n1 BASIC INFO:")
    print(f"    Rows: {df.shape[0]:,}")
    print(f"    Columns: {df.shape[1]}")
    print(f"    Memory: {df.memory_usage(deep=True).sum() / 1024**2:.2f} MB")

    # Data types
    print(f"\n2 DATA TYPES:")
    print(df.dtypes.value_counts())

    # Missing values
    print(f"\n3 MISSING VALUES:")
    missing = df.isnull().sum()
    missing_pct = (missing / len(df)) * 100

```

```

missing_df = pd.DataFrame({
    'Missing Count': missing[missing > 0],
    'Percentage': missing_pct[missing > 0]
}).sort_values('Percentage', ascending=False)

if len(missing_df) > 0:
    print(missing_df)
else:
    print("    No missing values!")

# Duplicates
print(f"\n4 DUPLICATES:")
duplicates = df.duplicated().sum()
print(f"    Duplicate rows: {duplicates:,}
({(duplicates/len(df)*100):.2f}%)")

# Numeric columns summary
print(f"\n5 NUMERIC COLUMNS SUMMARY:")
numeric_cols = df.select_dtypes(include=[np.number]).columns
if len(numeric_cols) > 0:
    print(df[numeric_cols].describe().T)
else:
    print("    No numeric columns found")

print("\n" + "="*70 + "\n")

# Profile each dataset
for key, df in data.items():
    profile_dataset(df, key)

```

No DETAILED PROFILE: PURCHASE_PRICES

1 BASIC INFO:

Rows: 12,261
 Columns: 9
 Memory: 3.43 MB

2 DATA TYPES:

object 4
 int64 3
 float64 2
 Name: count, dtype: int64

3 MISSING VALUES:

	Missing Count	Percentage
Description	1	0.008156
Size	1	0.008156
Volume	1	0.008156

4 DUPLICATES:

Duplicate rows: 0 (0.00%)

5 NUMERIC COLUMNS SUMMARY:

	count	mean	std	min	25%
Brand	12261.0	17989.067123	12528.503464	58.0	5990.00
Price	12261.0	38.640240	206.151172	0.0	10.99
Classification	12261.0	1.708996	0.454244	1.0	1.00
PurchasePrice	12261.0	26.488220	156.182948	0.0	6.89
VendorNumber	12261.0	10814.861757	19007.682322	2.0	3960.00

	75%	max
Brand	25117.00	90631.00
Price	29.99	13999.90
Classification	2.00	2.00
PurchasePrice	20.13	11111.03
VendorNumber	9552.00	173357.00

DETAILED PROFILE: BEG_INVENTORY

1 BASIC INFO:

Rows: 206,529

Columns: 9

Memory: 66.94 MB

2 DATA TYPES:

object 5

int64 3

float64 1

Name: count, dtype: int64

3 MISSING VALUES:

No missing values!

4 DUPLICATES:

Duplicate rows: 0 (0.00%)

5 NUMERIC COLUMNS SUMMARY:

	count	mean	std	min	25%
--	-------	------	-----	-----	-----

50% \						
Store	206529.0	42.122457	23.191393	1.0	22.00	42.00
Brand	206529.0	13761.482320	13059.429355	58.0	3746.00	8010.00
onHand	206529.0	20.429455	31.467342	0.0	7.00	12.00
Price	206529.0	22.253910	70.178964	0.0	9.99	14.99

	75%	max
Store	64.00	79.0
Brand	22143.00	90090.0
onHand	21.00	1251.0
Price	21.99	13999.9

DETAILED PROFILE: END_INVENTORY

1 BASIC INFO:

Rows: 224,489

Columns: 9

Memory: 72.74 MB

2 DATA TYPES:

object 5

int64 3

float64 1

Name: count, dtype: int64

3 MISSING VALUES:

	Missing Count	Percentage
City	1284	0.571966

4 DUPLICATES:

Duplicate rows: 0 (0.00%)

5 NUMERIC COLUMNS SUMMARY:

	count	mean	std	min	25%	50%
\						
Store	224489.0	43.505740	23.326415	1.00	23.00	44.00
Brand	224489.0	14356.370513	13118.467851	58.00	3798.00	8259.00
onHand	224489.0	21.763988	37.233576	0.00	7.00	12.00
Price	224489.0	23.585583	79.202775	0.49	9.99	14.99

	75%	max
Store	66.00	81.0
Brand	23965.00	90631.0
onHand	22.00	3676.0
Price	23.49	13999.9

DETAILED PROFILE: INVOICE_PURCHASES

1 BASIC INFO:

Rows: 5,543
 Columns: 10
 Memory: 1.73 MB

2 DATA TYPES:

object 5
 int64 3
 float64 2
 Name: count, dtype: int64

3 MISSING VALUES:

	Missing	Count	Percentage
Approval		5169	93.252751

4 DUPLICATES:

Duplicate rows: 0 (0.00%)

5 NUMERIC COLUMNS SUMMARY:

	count	mean	std	min	25%
VendorNumber	5543.0	20662.752120	34582.158410	2.00	3089.00
PONumber	5543.0	10889.419087	1600.859969	8106.00	9503.50
Quantity	5543.0	6058.880931	14453.338164	1.00	83.00
Dollars	5543.0	58073.383642	140234.031377	4.14	967.81
Freight	5543.0	295.954301	713.585093	0.02	5.02

	75%	max
VendorNumber	10754.000	201359.00
PONumber	12275.500	13661.00
Quantity	5100.500	141660.00
Dollars	44587.175	1660435.88

Freight 229.660 8468.22

=====

▢ DETAILED PROFILE: PURCHASES

=====

1 BASIC INFO:

Rows: 2,372,474

Columns: 16

Memory: 1278.97 MB

2 DATA TYPES:

object 8

int64 6

float64 2

Name: count, dtype: int64

3 MISSING VALUES:

	Missing Count	Percentage
Size	3	0.000126

4 DUPLICATES:

Duplicate rows: 0 (0.00%)

5 NUMERIC COLUMNS SUMMARY:

	count	mean	std	min	25%
\					
Store	2372474.0	44.651328	23.512448	1.0	25.00
Brand	2372474.0	12418.641110	12557.278331	58.0	3639.00
VendorNumber	2372474.0	6886.435533	8066.693891	2.0	3252.00
PONumber	2372474.0	11040.936647	1565.340220	8106.0	9761.00
PurchasePrice	2372474.0	12.050050	17.945104	0.0	6.12
Quantity	2372474.0	14.155846	23.446162	1.0	6.00
Dollars	2372474.0	135.681472	281.664941	0.0	49.26
Classification	2372474.0	1.443520	0.496800	1.0	1.00

	50%	75%	max
Store	48.00	67.00	81.00
Brand	6523.00	18877.00	90631.00
VendorNumber	4425.00	9552.00	201359.00
PONumber	11103.00	12397.00	13661.00

PurchasePrice	9.22	14.49	5681.81
Quantity	10.00	12.00	3816.00
Dollars	83.93	140.52	50175.70
Classification	1.00	2.00	2.00

DETAILED PROFILE: SALES

1 BASIC INFO:

Rows: 1,048,575

Columns: 14

Memory: 395.01 MB

2 DATA TYPES:

int64 6

object 5

float64 3

Name: count, dtype: int64

3 MISSING VALUES:

No missing values!

4 DUPLICATES:

Duplicate rows: 0 (0.00%)

5 NUMERIC COLUMNS SUMMARY:

	count	mean	std	min	25%
\					
Store	1048575.0	40.080632	24.357388	1.00	15.00
Brand	1048575.0	12169.585281	12419.213625	58.00	3680.00
SalesQuantity	1048575.0	2.337619	3.511492	1.00	1.00
SalesDollars	1048575.0	31.604201	65.702486	0.49	10.99
SalesPrice	1048575.0	15.431622	14.049674	0.49	8.99
Volume	1048575.0	950.028027	714.270774	50.00	750.00
Classification	1048575.0	1.416550	0.492987	1.00	1.00
ExciseTax	1048575.0	1.326847	3.407898	0.01	0.16
VendorNo	1048575.0	6995.043206	8426.735714	2.00	3252.00
	50%	75%	max		

Store	39.00	64.00	79.00
Brand	6296.00	17954.00	90089.00
SalesQuantity	1.00	2.00	432.00
SalesDollars	17.99	31.99	13279.97
SalesPrice	12.99	18.99	4999.99
Volume	750.00	1500.00	20000.00
Classification	1.00	2.00	2.00
ExciseTax	0.68	1.57	378.52
VendorNo	4425.00	9552.00	173357.00

=====

Data Cleaning Function

```
def clean_dataset(df, name):
    """
    Clean a dataset by handling missing values and duplicates
    """
    print(f"🧹 Cleaning {name}...")

    original_shape = df.shape

    # 1. Remove completely empty rows
    df = df.dropna(how='all')

    # 2. Remove duplicate rows
    df = df.drop_duplicates()

    # 3. Handle date columns (convert to datetime)
    date_columns = [col for col in df.columns if 'date' in col.lower()
    or 'time' in col.lower()]
    for col in date_columns:
        try:
            df[col] = pd.to_datetime(df[col], errors='coerce')
            print(f"📅 Converted {col} to datetime")
        except:
            pass

    # 4. Strip whitespace from string columns
    string_cols = df.select_dtypes(include=['object']).columns
    for col in string_cols:
        if df[col].dtype == 'object':
            df[col] = df[col].str.strip()

    print(f"📊 Original: {original_shape[0]:,} rows")
    print(f"📊 After cleaning: {df.shape[0]:,} rows")
    print(f"📊 Removed: {original_shape[0] - df.shape[0]:,} rows")
```

```

    print(f"    Cleaning complete!\n")

    return df

# Clean all datasets
print("Cleaning ALL DATASETS\n")
print("="*60)

cleaned_data = {}
for key, df in data.items():
    cleaned_data[key] = clean_dataset(df.copy(), key)

print("="*60)
print("ALL DATASETS CLEANED!")

```

Cleaning ALL DATASETS

```

=====
Cleaning purchase_prices...
  Original: 12,261 rows
  After cleaning: 12,261 rows
  Removed: 0 rows
  Cleaning complete!

Cleaning beg_inventory...
  Converted startDate to datetime
  Original: 206,529 rows
  After cleaning: 206,529 rows
  Removed: 0 rows
  Cleaning complete!

Cleaning end_inventory...
  Converted endDate to datetime
  Original: 224,489 rows
  After cleaning: 224,489 rows
  Removed: 0 rows
  Cleaning complete!

Cleaning invoice_purchases...
  Converted InvoiceDate to datetime
  Converted PODate to datetime
  Converted PayDate to datetime
  Original: 5,543 rows
  After cleaning: 5,543 rows
  Removed: 0 rows
  Cleaning complete!

Cleaning purchases...
  Converted PODate to datetime
  Converted ReceivingDate to datetime

```

```

Cleaning sales...
  Converted SalesDate to datetime
  Original: 1,048,575 rows
  After cleaning: 1,048,575 rows
  Removed: 0 rows
  Cleaning complete!

```

Summary Statistics

CLEANED DATA SUMMARY

```
{
  "summary": {
    "name": "summary_df",
    "rows": 6,
    "fields": [
      {
        "column": "Dataset",
        "dtype": "string",
        "num_unique_values": 6,
        "samples": [
          "Purchase Prices",
          "Beg Inventory",
          "Sales"
        ],
        "semantic type": ""
      }
    ]
  }
}
```

```

"description\": \"\\"\n        }\n    },\n    {\n        \"column\":
\"Rows\", \n        \"properties\": {\n            \"dtype\": \"string\", \n
\"num_unique_values\": 6, \n            \"samples\": [\n
\"12,261\", \n            \"206,529\", \n            \"1,048,575\" \n
n            ], \n            \"semantic_type\": \"\", \n
\"description\": \"\\"\n        }\n    },\n    {\n        \"column\":
\"Columns\", \n        \"properties\": {\n            \"dtype\": \"number\", \n
n            \"std\": 3, \n            \"min\": 9, \n            \"max\": 16, \n
\"num_unique_values\": 4, \n            \"samples\": [\n            10, \n
14, \n            9 \n            ], \n            \"semantic_type\": \"\", \n
\"description\": \"\\"\n        }\n    },\n    {\n        \"column\":
\"Memory (MB)\", \n        \"properties\": {\n            \"dtype\":
\"string\", \n            \"num_unique_values\": 6, \n            \"samples\":
[\n            \"3.36\", \n            \"56.89\", \n            \"339.07\" \n
], \n            \"semantic_type\": \"\", \n            \"description\": \"\\"\n
}\n    }\n    ], \n    \"type\": \"dataframe\", \"variable_name\": \"summary_df\"}

```

=====
□ Phase 1 Complete! Ready for Analysis! □

PAHSE 2

Calculate Product-Level Revenue

Aggregating total sales revenue for each product across all stores and dates.

```

print("□ ABC ANALYSIS - Step 1: Calculate Product Revenue\n")
print("="*70)

# Use sales data
sales_df = cleaned_data['sales'].copy()

# Group by product (Brand + Description) and calculate total revenue
product_revenue = sales_df.groupby(['Brand', 'Description']).agg({
    'SalesDollars': 'sum',
    'SalesQuantity': 'sum',
    'InventoryId': 'count' # Number of transactions
}).reset_index()

# Rename columns for clarity
product_revenue.columns = ['Brand', 'Description', 'TotalRevenue',
    'TotalQuantitySold', 'TransactionCount']

# Sort by revenue (descending)
product_revenue = product_revenue.sort_values('TotalRevenue',
    ascending=False).reset_index(drop=True)

```

```

print(f" Total Unique Products: {len(product_revenue):,}")
print(f" Total Revenue: $
{product_revenue['TotalRevenue'].sum():,.2f}")
print(f" Total Units Sold:
{product_revenue['TotalQuantitySold'].sum():,.0f}")

print("\n Top 10 Products by Revenue:\n")
display(product_revenue.head(10))

print("="*70)

```

ABC ANALYSIS - Step 1: Calculate Product Revenue

```

=====
 Total Unique Products: 7,658
 Total Revenue: $33,139,375.29
 Total Units Sold: 2,451,169

```

Top 10 Products by Revenue:

```

{"summary":{"\n  \"name\": \"print(\\\"\\\"*70\\\")\", \"rows\": 10, \"\n
\"fields\": [\n    {\n      \"column\": \"Brand\", \"\n
\"properties\": {\n        \"dtype\": \"number\", \"std\":\n
1927, \"\n        \"min\": 1233, \"max\": 8068, \"\n
\"num_unique_values\": 10, \"samples\": [\n          1376, \"\n
3545, \"          3858\", \"          ], \"semantic_type\": \"\", \"\n
\"description\": \"\", \"\n        }, \"\n        \"column\":\n
\"Description\", \"\n        \"properties\": {\n          \"dtype\":\n
\"string\", \"\n          \"num_unique_values\": 9, \"samples\":\n
[\n            \"Bacardi Superior Rum Trav\", \"Ketel One\n
Vodka\", \"\n            \"Grey Goose Vodka\", \"\n          ], \"\n
\"semantic_type\": \"\", \"description\": \"\", \"\n
n      }, \"\n        \"column\": \"TotalRevenue\", \"\n
\"properties\": {\n          \"dtype\": \"number\", \"std\":\n
94677.04446377345, \"\n          \"min\": 164426.6, \"max\":\n
444810.74, \"\n          \"num_unique_values\": 10, \"samples\":\n
[\n            169922.38, \"          357759.17, \"          225014.22\", \"\n
          ], \"\n          \"semantic_type\": \"\", \"description\": \"\", \"\n
        }, \"\n        \"column\": \"TotalQuantitySold\", \"\n
\"properties\": {\n          \"dtype\": \"number\", \"std\":\n
4159, \"\n          \"min\": 4649, \"max\": 20226, \"\n
\"num_unique_values\": 10, \"samples\": [\n            7362, \"\n
11883, \"          9378\", \"          ], \"semantic_type\": \"\", \"\n
\"description\": \"\", \"\n        }, \"\n        \"column\":\n
\"TransactionCount\", \"\n        \"properties\": {\n          \"dtype\":\n
\"number\", \"\n          \"std\": 146, \"\n          \"min\": 1407, \"\n
\"max\": 1969, \"\n          \"num_unique_values\": 10, \"\n
\"samples\": [\n            1841, \"          1749, \"          1762\", \"\n

```

```
] , \n      \"semantic_type\": \"\", \n      \"description\": \"\" \n} \n    } \n  ] \n} \", \"type\": \"dataframe\"}
```

Calculate Cumulative Revenue Percentage

Computing what percentage of total revenue each product contributes cumulatively.

```
print("\n ABC ANALYSIS - Step 2: Calculate Cumulative Metrics\n")
print("="*70)
```

```
# Calculate percentage of total revenue
product_revenue['RevenuePercentage'] =
(product_revenue['TotalRevenue'] /
product_revenue['TotalRevenue'].sum()) * 100
```

```
# Calculate cumulative revenue percentage
product_revenue['CumulativeRevenuePercentage'] =
product_revenue['RevenuePercentage'].cumsum()
```

```
# Calculate cumulative product count percentage
product_revenue['CumulativeProductPercentage'] =
((product_revenue.index + 1) / len(product_revenue)) * 100
```

```
print("█ Cumulative metrics calculated!")
```

```
print("\n Sample with Cumulative Percentages:\n")
display(product_revenue[['Brand', 'Description', 'TotalRevenue',
                          'RevenuePercentage',
                          'CumulativeRevenuePercentage',
                          'CumulativeProductPercentage']].head(15))
```

```
print( "="*70)
```

ABC ANALYSIS - Step 2: Calculate Cumulative Metrics

□ Cumulative metrics calculated!

□ Sample with Cumulative Percentages:

```
{
  "summary": {
    "name": "print(\\\"=\\\"*70\\\")",
    "rows": 15,
    "fields": [
      {
        "column": "Brand",
        "dtype": "number",
        "std": 2089,
        "min": 1233,
        "max": 8680,
        "num unique values": 15,
        "samples": [
          2585
        ]
      }
    ]
  }
}
```

```

2757,\n          4261\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\"\n          },\n          {\n          \"column\":\n          \"Description\",\n          \"properties\": {\n          \"dtype\":\n          \"string\",\n          \"num_unique_values\": 14,\n          \"samples\":\n          [\n          \"Crown Royal\",\n          \"Kahlua\",\n          \"Capt\nMorgan Spiced Rum\"\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\"\n          },\n          {\n          \"column\":\n          \"TotalRevenue\",\n          \"properties\": {\n          \"dtype\":\n          \"number\",\n          \"std\": 94084.1565481131,\n          \"min\":\n          142887.85,\n          \"max\": 444810.74,\n          \"num_unique_values\": 15,\n          \"samples\": [\n          164426.6,\n          155678.09,\n          444810.74\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\"\n          },\n          {\n          \"column\": \"RevenuePercentage\",\n          \"properties\": {\n          \"dtype\": \"number\",\n          \"std\":\n          0.2839044361119974,\n          \"min\": 0.431172430830696,\n          \"max\": 1.3422423811779705,\n          \"num_unique_values\": 15,\n          \"samples\": [\n          0.49616686663860166,\n          0.46976772687376755,\n          1.3422423811779705\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\"\n          },\n          {\n          \"column\": \"CumulativeRevenuePercentage\",\n          \"properties\": {\n          \"dtype\": \"number\",\n          \"std\":\n          2.753770162998267,\n          \"min\": 1.3422423811779705,\n          \"max\": 10.259712563217725,\n          \"num_unique_values\": 15,\n          \"samples\": [\n          7.982265226340059,\n          8.941057047910332,\n          1.3422423811779705\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\"\n          },\n          {\n          \"column\": \"CumulativeProductPercentage\",\n          \"properties\": {\n          \"dtype\": \"number\",\n          \"std\":\n          0.05839822349176781,\n          \"min\": 0.013058239749281797,\n          \"max\": 0.19587359623922695,\n          \"num_unique_values\": 15,\n          \"samples\": [\n          0.13058239749281797,\n          0.15669887699138157,\n          0.013058239749281797\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\"\n          }\n          }\n          ],\n          \"type\": \"dataframe\"}

```

=====

Assign ABC Classification

Categorizing each product into A, B, or C class based on cumulative revenue contribution.

```

print("\n ABC ANALYSIS - Step 3: Assign ABC Categories\n")
print("="*70)

def assign_abc_category(cumulative_revenue_pct):
    """

```

```

    Assign ABC category based on cumulative revenue percentage
    A: Top products contributing to first 80% of revenue
    B: Products contributing to next 15% of revenue (80-95%)
    C: Products contributing to last 5% of revenue (95-100%)
    """
    if cumulative_revenue_pct <= 80:
        return 'A'
    elif cumulative_revenue_pct <= 95:
        return 'B'
    else:
        return 'C'

# Apply ABC classification
product_revenue['ABC_Category'] =
product_revenue['CumulativeRevenuePercentage'].apply(assign_abc_category)

print("\n ABC Categories assigned!")

# Summary statistics by category
abc_summary = product_revenue.groupby('ABC_Category').agg({
    'Brand': 'count', # Number of products
    'TotalRevenue': 'sum',
    'TotalQuantitySold': 'sum'
}).reset_index()

abc_summary.columns = ['ABC_Category', 'ProductCount', 'TotalRevenue',
'TotalUnitsSold']
abc_summary['RevenuePercentage'] = (abc_summary['TotalRevenue'] /
abc_summary['TotalRevenue'].sum()) * 100
abc_summary['ProductPercentage'] = (abc_summary['ProductCount'] /
abc_summary['ProductCount'].sum()) * 100

print("\n ABC CATEGORY SUMMARY:\n")
display(abc_summary)

print("\n" + "="*70)

```

ABC ANALYSIS - Step 3: Assign ABC Categories

```

=====
ABC Categories assigned!

```

```

ABC CATEGORY SUMMARY:

```

```

{"summary":{"\n  \"name\": \"abc_summary\", \n  \"rows\": 3, \n
\"fields\": [\n    {\n      \"column\": \"ABC_Category\", \n
\"properties\": {\n        \"dtype\": \"string\", \n
\"num_unique_values\": 3, \n        \"samples\": [\n          \"A\", \n

```

```

\\B\\",\n          \\C\\",\n          ],\n          \\semantic_type\\": \\",\n          \\description\\": \\",\n          },\n          {\n          \\column\\": \\ProductCount\\",\n          \\properties\\": {\n          \\dtype\\": \\number\\",\n          \\std\\": 1558,\n          \\min\\": 1502,\n          \\max\\": 4343,\n          \\num_unique_values\\": 3,\n          \\samples\\": [\n          1502,\n          1813,\n          4343\n          ],\n          \\semantic_type\\": \\",\n          \\description\\": \\",\n          },\n          {\n          \\column\\": \\TotalRevenue\\",\n          \\properties\\": {\n          \\dtype\\": \\number\\",\n          \\std\\": 13493382.076854398,\n          \\min\\": 1658184.1199999999,\n          \\max\\": 26509374.02,\n          \\num_unique_values\\": 3,\n          \\samples\\": [\n          26509374.02,\n          4971817.15,\n          1658184.1199999999\n          ],\n          \\semantic_type\\": \\",\n          \\description\\": \\",\n          },\n          {\n          \\column\\": \\TotalUnitsSold\\",\n          \\properties\\": {\n          \\dtype\\": \\number\\",\n          \\std\\": 927185,\n          \\min\\": 128884,\n          \\max\\": 1871415,\n          \\num_unique_values\\": 3,\n          \\samples\\": [\n          1871415,\n          450870,\n          128884\n          ],\n          \\semantic_type\\": \\",\n          \\description\\": \\",\n          },\n          {\n          \\column\\": \\RevenuePercentage\\",\n          \\properties\\": {\n          \\dtype\\": \\number\\",\n          \\std\\": 40.71706831759772,\n          \\min\\": 5.003667406187849,\n          \\max\\": 79.99358403113699,\n          \\num_unique_values\\": 3,\n          \\samples\\": [\n          79.99358403113699,\n          15.002748562675153,\n          5.003667406187849\n          ],\n          \\semantic_type\\": \\",\n          \\description\\": \\",\n          },\n          {\n          \\column\\": \\ProductPercentage\\",\n          \\properties\\": {\n          \\dtype\\": \\number\\",\n          \\std\\": 20.3480325543316,\n          \\min\\": 19.61347610342126,\n          \\max\\": 56.71193523113084,\n          \\num_unique_values\\": 3,\n          \\samples\\": [\n          19.61347610342126,\n          23.6745886654479,\n          56.71193523113084\n          ],\n          \\semantic_type\\": \\",\n          \\description\\": \\",\n          },\n          }\n          }\n          ]\n          n},"type":"dataframe","variable_name":"abc_summary"}

```

Visualize ABC Analysis - Pareto Chart

Creating a Pareto chart to visualize the 80/20 rule in action.

```

print("\n ABC ANALYSIS - Step 4: Create Pareto Chart\n")
print("="*70)

# Create Pareto Chart
fig = make_subplots(specs=[[{"secondary_y": True}]])

```

```

# Bar chart - Product count
fig.add_trace(
    go.Bar(
        x=product_revenue.index[:100], # First 100 products
        y=product_revenue['TotalRevenue'][:100],
        name='Revenue per Product',
        marker_color='steelblue'
    ),
    secondary_y=False
)

# Line chart - Cumulative percentage
fig.add_trace(
    go.Scatter(
        x=product_revenue.index[:100],
        y=product_revenue['CumulativeRevenuePercentage'][:100],
        name='Cumulative Revenue %',
        line=dict(color='red', width=3),
        mode='lines'
    ),
    secondary_y=True
)

# Add reference lines for ABC boundaries
fig.add_hline(y=80, line_dash="dash", line_color="green",
              annotation_text="A-B Boundary (80%)", secondary_y=True)
fig.add_hline(y=95, line_dash="dash", line_color="orange",
              annotation_text="B-C Boundary (95%)", secondary_y=True)

# Update layout
fig.update_layout(
    title='☐ ABC Analysis - Pareto Chart (Top 100 Products)',
    xaxis_title='Product Rank',
    height=600,
    hovermode='x unified',
    showlegend=True
)

fig.update_yaxes(title_text="Revenue ($)", secondary_y=False)
fig.update_yaxes(title_text="Cumulative Revenue (%)",
                 secondary_y=True, range=[0, 105])

fig.show()

print("☐ Pareto chart generated!")
print("="*70)

```

☐ ABC ANALYSIS - Step 4: Create Pareto Chart

```
=====
❑ Pareto chart generated!
=====
```

Visualize ABC Category Distribution

Creating pie charts and bar charts to show the distribution of products and revenue across ABC categories.

```
print("\n❑ ABC ANALYSIS - Step 5: Category Distribution Charts\n")
print("="*70)

# Create subplots: 2 pie charts side by side
fig = make_subplots(
    rows=1, cols=2,
    subplot_titles=('Product Count by ABC Category', 'Revenue
Distribution by ABC Category'),
    specs=[[{'type': 'pie'}, {'type': 'pie'}]]
)

# Pie chart 1: Product count
fig.add_trace(
    go.Pie(
        labels=abc_summary['ABC_Category'],
        values=abc_summary['ProductCount'],
        marker=dict(colors=['#2ecc71', '#f39c12', '#e74c3c']),
        textinfo='label+percent',
        name='Products'
    ),
    row=1, col=1
)

# Pie chart 2: Revenue
fig.add_trace(
    go.Pie(
        labels=abc_summary['ABC_Category'],
        values=abc_summary['TotalRevenue'],
        marker=dict(colors=['#2ecc71', '#f39c12', '#e74c3c']),
        textinfo='label+percent',
        name='Revenue'
    ),
    row=1, col=2
)

fig.update_layout(height=500, title_text="❑ ABC Category
Distribution")
```

```
fig.show()

print(" Distribution charts generated!")
print("="*70)
```

ABC ANALYSIS - Step 5: Category Distribution Charts

```
=====  
 Distribution charts generated!  
=====
```

Detailed Breakdown by Category

Showing sample products from each ABC category with their metrics.

```
print("\n ABC ANALYSIS - Step 6: Detailed Category Breakdown\n")
print("="*70)

for category in ['A', 'B', 'C']:
    category_data = product_revenue[product_revenue['ABC_Category'] ==
category]

    print(f"\n{'='*70}")
    print(f" CATEGORY {category}")
    print(f"{'='*70}")
    print(f"Total Products: {len(category_data):,}")
    print(f"Total Revenue: $
{category_data['TotalRevenue'].sum():,.2f}")
    print(f"Average Revenue per Product: $
{category_data['TotalRevenue'].mean():,.2f}")
    print(f"Revenue Range: ${category_data['TotalRevenue'].min():,.2f}
- ${category_data['TotalRevenue'].max():,.2f}")

    print(f"\n Top 5 Products in Category {category}:\n")
    display(category_data[['Brand', 'Description', 'TotalRevenue',
'TotalQuantitySold',
'RevenuePercentage',
'CumulativeRevenuePercentage']].head(5))

print("\n" + "="*70)
```

ABC ANALYSIS - Step 6: Detailed Category Breakdown

```
=====  

=====
```

=====

□ Top 5 Products in Category A:

```
{
  "summary": {
    "name": "print(\\\"\\\\\\\\\\\\\\\\\\\\\\\\ + \\\"\\\"=\\\"*70)\\\"\\\",\\n
    \"rows\": 5,\\n
    \"fields\": [\\n
      {\\n
        \"column\": \"Brand\\\",\\n
        \"dtype\": \"number\\\",\\n
        \"std\": 2489,\\n
        \"min\": 1233,\\n
        \"max\": 8068,\\n
        \"num_unique_values\": 5,\\n
        \"samples\": [\\n
          3545,\\n
          3405,\\n
          1233\\n
        ],\\n
        \"semantic_type\": \"\\\"\\\",\\n
        \"description\": \"\\\"\\\"\\n
      }\\n
    ],\\n
    {\\n
      \"column\": \"Description\\\",\\n
      \"dtype\": \"string\\\",\\n
      \"num_unique_values\": 5,\\n
      \"samples\": [\\n
        \"Ketel One Vodka\\\",\\n
        \"Tito's Handmade Vodka\\\",\\n
        \"Jack Daniels No 7 Black\\\"\\n
      ],\\n
      \"semantic_type\": \"\\\"\\\",\\n
      \"description\": \"\\\"\\\"\\n
    }\\n
  ],\\n
  {\\n
    \"column\": \"TotalRevenue\\\",\\n
    \"dtype\": \"number\\\",\\n
    \"std\": 67443.1306008423,\\n
    \"min\": 275162.97,\\n
    \"max\": 444810.74,\\n
    \"num_unique_values\": 5,\\n
    \"samples\": [\\n
      357759.17,\\n
      275162.97,\\n
      344712.22000000003\\n
    ],\\n
    \"semantic_type\": \"\\\"\\\",\\n
    \"description\": \"\\\"\\\"\\n
  }\\n
],\\n
{\\n
  \"column\": \"TotalQuantitySold\\\",\\n
  \"dtype\": \"number\\\",\\n
  \"std\": 4504,\\n
  \"min\": 9203,\\n
  \"max\": 20226,\\n
  \"num_unique_values\": 5,\\n
  \"samples\": [\\n
    11883,\\n
    9203,\\n
    9578\\n
  ],\\n
  \"semantic_type\": \"\\\"\\\",\\n
  \"description\": \"\\\"\\\"\\n
}
  ],\\n
  {\\n
    \"column\": \"RevenuePercentage\\\",\\n
    \"dtype\": \"number\\\",\\n
    \"std\": 0.20351358470294897,\\n
    \"min\": 0.8303203291916972,\\n
    \"max\": 1.3422423811779705,\\n
    \"num_unique_values\": 5,\\n
    \"samples\": [\\n
      1.0795591856191566,\\n
      0.8303203291916972,\\n
      1.0401892521613676\\n
    ],\\n
    \"semantic_type\": \"\\\"\\\",\\n
    \"description\": \"\\\"\\\"\\n
  }\\n
],\\n
  {\\n
    \"column\": \"CumulativeRevenuePercentage\\\",\\n
    \"dtype\": \"number\\\",\\n
    \"std\": 1.5125129471674137,\\n
    \"min\": 1.3422423811779705,\\n
    \"max\": 5.1617756672564,\\n
    \"num_unique_values\": 5,\\n
    \"samples\": [\\n
      2.421801566797127,\\n
      5.1617756672564,\\n
      3.4619908189584945\\n
    ],\\n
    \"semantic_type\": \"\\\"\\\",\\n
    \"description\": \"\\\"\\\"\\n
  }\\n
]
},\\n
\"type\": \"dataframe\"
}
```

CATEGORY B

Total Products: 1,813
Total Revenue: \$4,971,817.15
Average Revenue per Product: \$2,742.32
Revenue Range: \$1,341.86 - \$5,035.03

Top 5 Products in Category B:

```
{"summary":{"name": "print(\\\\"\\\\"\\\\" + \\\\"=\\\\"*70)\\",\n"rows": 5,\n"fields": {\n"column": "Brand",\n"properties": {\n"dtype": "number",\n"std": 14811,\n"min": 757,\n"max": 32796,\n"num_unique_values": 5,\n"samples": [\n759,\n32796,\n5334\n],\n"semantic_type": "\\\"",\n"description": "\\\"",\n},\n"column": "Description",\n"properties": {\n"dtype": "string",\n"num_unique_values": 5,\n"samples": [\n"BunnaHabhaIn 12 Yr Single",\n"Warre's Otima 10-Yr Tawny",\n"Kahlua Especial"\n],\n"semantic_type": "\\\"",\n"description": "\\\"",\n},\n"column": "TotalRevenue",\n"properties": {\n"dtype": "number",\n"std": 4.319459456922836,\n"min": 5023.49,\n"max": 5035.03,\n"num_unique_values": 5,\n"samples": [\n5027.04,\n5023.49,\n5026.95\n],\n"semantic_type": "\\\"",\n"description": "\\\"",\n},\n"column": "TotalQuantitySold",\n"properties": {\n"dtype": "number",\n"std": 69,\n"min": 96,\n"max": 278,\n"num_unique_values": 5,\n"samples": [\n96,\n251,\n205\n],\n"semantic_type": "\\\"",\n"description": "\\\"",\n},\n"column": "RevenuePercentage",\n"properties": {\n"dtype": "number",\n"std": 1.3034221131581148e-05,\n"min": 0.015158674404812535,\n"max": 0.015193497028652046,\n"num_unique_values": 5,\n"samples": [\n0.015169386737102851,\n0.015158674404812535,\n0.015169115156847605\n],\n"semantic_type": "\\\"",\n"description": "\\\"",\n},\n"column": "CumulativeRevenuePercentage",\n"properties": {\n"dtype": "number",\n"std": 0.023980216572086065,\n"min": 80.0087775281659,\n"max": 80.06944161680389,\n"num_unique_values": 5,\n"samples": [\n80.023946914903,\n80.06944161680389,\n80.03911603005984\n],\n
```

```
\n    \"semantic_type\": \"\", \n    \"description\": \"\" \n  }\n  ], \"type\": \"dataframe\"}
```

CATEGORY C

Total Products: 4,343
Total Revenue: \$1,658,184.12
Average Revenue per Product: \$381.81
Revenue Range: \$0.98 - \$1,339.93

Top 5 Products in Category C:

```
{\"summary\":{\"name\": \"print(\\\"\\\\\\\"\\\\\\\"\\\\\\\" + \\\"=\\\\\\\"*70)\\\", \n\"rows\": 5, \n\"fields\": [ \n  { \n    \"column\": \"Brand\", \n    \"properties\": { \n      \"dtype\": \"number\", \n      \"std\": 9161, \n      \"min\": 1065, \n      \"max\": 24498, \n      \"num_unique_values\": 5, \n      \"samples\": [ \n        20800, \n        24498, \n        15334 \n      ], \n      \"semantic_type\": \"\", \n      \"description\": \"\" \n    }, \n    { \n      \"column\": \"Description\", \n      \"properties\": { \n        \"dtype\": \"string\", \n        \"num_unique_values\": 5, \n        \"samples\": [ \n          \"Talamonti Moda Montepulciano\", \n          \"Cuvaion Chard\", \n          \"Brotte La Griveliere CDR\" \n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\" \n      }, \n      { \n        \"column\": \"TotalRevenue\", \n        \"properties\": { \n          \"dtype\": \"number\", \n          \"std\": 1.0176590784737405, \n          \"min\": 1337.5, \n          \"max\": 1339.93, \n          \"num_unique_values\": 5, \n          \"samples\": [ \n            1339.51, \n            1337.5, \n            1338.66 \n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\" \n        }, \n        { \n          \"column\": \"TotalQuantitySold\", \n          \"properties\": { \n            \"dtype\": \"number\", \n            \"std\": 37, \n            \"min\": 50, \n            \"max\": 149, \n            \"num_unique_values\": 5, \n            \"samples\": [ \n              149, \n              50, \n              134 \n            ], \n            \"semantic_type\": \"\", \n            \"description\": \"\" \n          }, \n          { \n            \"column\": \"RevenuePercentage\", \n            \"properties\": { \n              \"dtype\": \"number\", \n              \"std\": 3.0708456920753644e-06, \n              \"min\": 0.004035984348816613, \n              \"max\": 0.004043317015708294, \n              \"num_unique_values\": 5, \n              \"samples\": [ \n                0.004042049641183806, \n                0.004035984348816613, \n                0.0040394847165509135 \n              ], \n              \"semantic_type\": \"\", \n              \"description\": \"\" \n            }, \n            { \n              \"column\": \"CumulativeRevenuePercentage\", \n              \"properties\": { \n                \"dtype\": \"number\", \n                \"std\": 0.006385702976812503, \n                \"min\": 95.00037591082769, \n                \"max\": 95.0165308321355, \n                \"num_unique_values\": 5, \n
```

=====

```
=====
[ ] ABC Analysis saved to: /content/ABC_Analysis_Results.csv
[ ] ABC Summary saved to: /content/ABC_Summary.csv

[ ] Files saved successfully!
=====
```

```
print("\n ABC ANALYSIS - Step 8: Key Insights\n")
print("="*70)
```

```

# Calculate key metrics
total_products = len(product_revenue)
a_products = len(product_revenue[product_revenue['ABC_Category'] ==
'A'])
b_products = len(product_revenue[product_revenue['ABC_Category'] ==
'B'])
c_products = len(product_revenue[product_revenue['ABC_Category'] ==
'C'])

a_revenue_pct = abc_summary[abc_summary['ABC_Category'] == 'A']
['RevenuePercentage'].values[0]
b_revenue_pct = abc_summary[abc_summary['ABC_Category'] == 'B']
['RevenuePercentage'].values[0]
c_revenue_pct = abc_summary[abc_summary['ABC_Category'] == 'C']
['RevenuePercentage'].values[0]

print("\n KEY INSIGHTS:\n")

print(1"1 CATEGORY A (High Value Items):")
print(f"    • {a_products:,} products
({(a_products/total_products)*100:.1f}% of inventory)")
print(f"    • Generate ${abc_summary[abc_summary['ABC_Category']=='A']
['TotalRevenue'].values[0]:,.2f}")
print(f"    • Contribute {a_revenue_pct:.1f}% of total revenue")
print(f"    • RECOMMENDATION: Daily monitoring, never stock out,
priority reordering\n")

print(2"2 CATEGORY B (Medium Value Items):")
print(f"    • {b_products:,} products
({(b_products/total_products)*100:.1f}% of inventory)")
print(f"    • Generate ${abc_summary[abc_summary['ABC_Category']=='B']
['TotalRevenue'].values[0]:,.2f}")
print(f"    • Contribute {b_revenue_pct:.1f}% of total revenue")
print(f"    • RECOMMENDATION: Weekly monitoring, maintain safety
stock\n")

print(3"3 CATEGORY C (Low Value Items):")
print(f"    • {c_products:,} products
({(c_products/total_products)*100:.1f}% of inventory)")
print(f"    • Generate ${abc_summary[abc_summary['ABC_Category']=='C']
['TotalRevenue'].values[0]:,.2f}")
print(f"    • Contribute {c_revenue_pct:.1f}% of total revenue")
print(f"    • RECOMMENDATION: Monthly monitoring, consider
discontinuing slow movers\n")

print("="*70)
print("\n ABC ANALYSIS COMPLETE!")
print("="*70)

```

□ ABC ANALYSIS - Step 8: Key Insights

□ KEY INSIGHTS:

1 CATEGORY A (High Value Items):

- 1,502 products (19.6% of inventory)
- Generate \$26,509,374.02
- Contribute 80.0% of total revenue
- □ RECOMMENDATION: Daily monitoring, never stock out, priority reordering

2 CATEGORY B (Medium Value Items):

- 1,813 products (23.7% of inventory)
- Generate \$4,971,817.15
- Contribute 15.0% of total revenue
- □ RECOMMENDATION: Weekly monitoring, maintain safety stock

3 CATEGORY C (Low Value Items):

- 4,343 products (56.7% of inventory)
- Generate \$1,658,184.12
- Contribute 5.0% of total revenue
- □ RECOMMENDATION: Monthly monitoring, consider discontinuing slow movers

□ ABC ANALYSIS COMPLETE!

PHASE 2.2

DEMAND FORECASTING Converting sales data into a time series format with daily aggregations.

```
print("□ DEMAND FORECASTING - Step 1: Prepare Time Series Data\n")
print("="*70)
```

```
# Use sales data
```

```
sales_df = cleaned_data['sales'].copy()
```

```
# Ensure SalesDate is datetime
```

```
sales_df['SalesDate'] = pd.to_datetime(sales_df['SalesDate'])
```

```
# Aggregate daily sales across all products
```

```
daily_sales = sales_df.groupby('SalesDate').agg({
    'SalesDollars': 'sum',
    'SalesQuantity': 'sum'
})
```

```

}).reset_index()

# Sort by date
daily_sales =
daily_sales.sort_values('SalesDate').reset_index(drop=True)

# Create additional time features
daily_sales['Year'] = daily_sales['SalesDate'].dt.year
daily_sales['Month'] = daily_sales['SalesDate'].dt.month
daily_sales['DayOfWeek'] = daily_sales['SalesDate'].dt.dayofweek
daily_sales['DayName'] = daily_sales['SalesDate'].dt.day_name()
daily_sales['WeekOfYear'] =
daily_sales['SalesDate'].dt.isocalendar().week

print(f" Time series data prepared!")
print(f"    Date Range: {daily_sales['SalesDate'].min()} to
{daily_sales['SalesDate'].max()}")
print(f"    Total Days: {len(daily_sales)}")
print(f"    Total Revenue: ${daily_sales['SalesDollars'].sum():,.2f}")

print("\n First 10 days of data:\n")
display(daily_sales.head(10))

print("\n Basic Statistics:\n")
print(daily_sales[['SalesDollars', 'SalesQuantity']].describe())

print("="*70)

```

DEMAND FORECASTING - Step 1: Prepare Time Series Data

```

=====
 Time series data prepared!
    Date Range: 2016-01-01 00:00:00 to 2016-02-29 00:00:00
    Total Days: 60
    Total Revenue: $33,139,375.29

```

First 10 days of data:

```

{"summary":{"\n  \"name\": \"print(\\\"\\\"=\\\"\\\"*70)\\\",\\n  \"rows\": 10,\\n
\\\"fields\": [\\n    {\\n      \"column\": \"SalesDate\\\",\\n
\\\"properties\": {\\n        \"dtype\": \"date\\\",\\n        \"min\":
\\\"2016-01-01 00:00:00\\\",\\n        \"max\": \"2016-01-10 00:00:00\\\",\\n
\\\"num_unique_values\": 10,\\n        \"samples\": [\\n          \"2016-
01-09 00:00:00\\\",\\n          \"2016-01-02 00:00:00\\\",\\n
\\\"2016-01-06 00:00:00\\\"\\n        ],\\n        \"semantic_type\": \"\\\",\\n
\\n        \"description\": \"\\\"\\\"\\n      }\\n    },\\n    {\\n
\\\"column\": \"SalesDollars\\\",\\n      \"properties\": {\\n
\\\"dtype\": \"number\\\",\\n      \"std\": 360426.03202491975,\\n
\\\"min\": 496719.05,\\n      \"max\": 1428520.9,\\n
\\\"num_unique_values\": 10,\\n      \"samples\": [\\n

```

```

1428520.9,\n          1303610.64,\n          660150.82\n          ],\n  \"semantic_type\": \"\",\n  \"description\": \"\",\n  },\n  {\n    \"column\": \"SalesQuantity\",\n    \"properties\": {\n      \"dtype\": \"number\",\n      \"std\": 24141,\n      \"min\": 40831,\n      \"max\": 101690,\n      \"num_unique_values\": 10,\n      \"samples\": [\n        101690,\n        93114,\n        51358\n      ],\n      \"semantic_type\": \"\",\n      \"description\": \"\",\n    },\n    {\n      \"column\": \"Year\",\n      \"properties\": {\n        \"dtype\": \"int32\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          2016\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\",\n      },\n    },\n    {\n      \"column\": \"Month\",\n      \"properties\": {\n        \"dtype\": \"int32\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          1\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\",\n      },\n    },\n    {\n      \"column\": \"DayOfWeek\",\n      \"properties\": {\n        \"dtype\": \"int32\",\n        \"num_unique_values\": 7,\n        \"samples\": [\n          4\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\",\n      },\n    },\n    {\n      \"column\": \"DayName\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 7,\n        \"samples\": [\n          \"Friday\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\",\n      },\n    },\n    {\n      \"column\": \"WeekOfYear\",\n      \"properties\": {\n        \"dtype\": \"UInt32\",\n        \"num_unique_values\": 2,\n        \"samples\": [\n          1\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\",\n      },\n    }\n  ],\n  \"type\": \"dataframe\"}

```

Basic Statistics:

	SalesDollars	SalesQuantity
count	6.000000e+01	60.000000
mean	5.523229e+05	40852.816667
std	5.510477e+05	38472.384148
min	6.415285e+04	5425.000000
25%	9.491752e+04	7513.250000
50%	4.981435e+05	41086.000000
75%	7.878177e+05	59825.500000
max	2.721899e+06	180426.000000

Visualize Overall Sales Trends

Creating visualizations to understand historical sales patterns and seasonality.

```

print("\n DEMAND FORECASTING - Step 2: Visualize Sales Trends\n")
print("="*70)

# Create time series plot
fig = make_subplots(
    rows=2, cols=1,
    subplot_titles=('Daily Revenue Trend', 'Daily Units Sold Trend'),
    vertical_spacing=0.15
)

# Revenue trend
fig.add_trace(
    go.Scatter(
        x=daily_sales['SalesDate'],
        y=daily_sales['SalesDollars'],
        mode='lines',
        name='Daily Revenue',
        line=dict(color='steelblue', width=1)
    ),
    row=1, col=1
)

# Add 7-day moving average for revenue
daily_sales['Revenue_MA7'] =
daily_sales['SalesDollars'].rolling(window=7).mean()
fig.add_trace(
    go.Scatter(
        x=daily_sales['SalesDate'],
        y=daily_sales['Revenue_MA7'],
        mode='lines',
        name='7-Day Moving Average',
        line=dict(color='red', width=2)
    ),
    row=1, col=1
)

# Quantity trend
fig.add_trace(
    go.Scatter(
        x=daily_sales['SalesDate'],
        y=daily_sales['SalesQuantity'],
        mode='lines',
        name='Daily Units Sold',
        line=dict(color='green', width=1)
    ),
    row=2, col=1
)

# Add 7-day moving average for quantity
daily_sales['Quantity_MA7'] =

```

```

daily_sales['SalesQuantity'].rolling(window=7).mean()
fig.add_trace(
    go.Scatter(
        x=daily_sales['SalesDate'],
        y=daily_sales['Quantity_MA7'],
        mode='lines',
        name='7-Day Moving Average',
        line=dict(color='orange', width=2)
    ),
    row=2, col=1
)

fig.update_layout(
    height=700,
    title_text="📊 Sales Trends Over Time (2016)",
    showlegend=True,
    hovermode='x unified'
)

fig.update_xaxes(title_text="Date", row=2, col=1)
fig.update_yaxes(title_text="Revenue ($)", row=1, col=1)
fig.update_yaxes(title_text="Units Sold", row=2, col=1)

fig.show()

print("📊 Trend visualizations created!")
print("="*70)

```

📊 DEMAND FORECASTING - Step 2: Visualize Sales Trends

```

=====
📊 Trend visualizations created!
=====

```

Analyze Seasonality Patterns

Examining sales patterns by month, day of week, and week of year to identify seasonality.

```

print("\n📊 DEMAND FORECASTING - Step 3: Analyze Seasonality\n")
print("="*70)

# Monthly aggregation
monthly_sales = daily_sales.groupby('Month').agg({
    'SalesDollars': 'mean',
    'SalesQuantity': 'mean'
}).reset_index()

```

```

monthly_sales['MonthName'] = monthly_sales['Month'].apply(
    lambda x: datetime(2016, x, 1).strftime('%B')
)

# Day of week aggregation
dow_sales = daily_sales.groupby('DayName').agg({
    'SalesDollars': 'mean',
    'SalesQuantity': 'mean'
}).reindex(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']).reset_index()

# Create seasonality visualizations
fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=('Average Revenue by Month', 'Average Units by
Month',
                    'Average Revenue by Day of Week', 'Average Units
by Day of Week'),
    specs=[[{'type': 'bar'}, {'type': 'bar'}],
           [{'type': 'bar'}, {'type': 'bar'}]]
)

# Monthly revenue
fig.add_trace(
    go.Bar(x=monthly_sales['MonthName'],
y=monthly_sales['SalesDollars'],
        marker_color='steelblue', name='Revenue'),
    row=1, col=1
)

# Monthly quantity
fig.add_trace(
    go.Bar(x=monthly_sales['MonthName'],
y=monthly_sales['SalesQuantity'],
        marker_color='green', name='Units'),
    row=1, col=2
)

# Day of week revenue
fig.add_trace(
    go.Bar(x=dow_sales['DayName'], y=dow_sales['SalesDollars'],
        marker_color='coral', name='Revenue'),
    row=2, col=1
)

# Day of week quantity
fig.add_trace(
    go.Bar(x=dow_sales['DayName'], y=dow_sales['SalesQuantity'],
        marker_color='orange', name='Units'),

```

```

        row=2, col=2
    )

fig.update_layout(height=800, title_text="□ Seasonality Analysis",
showlegend=False)
fig.update_yaxes(title_text="Avg Revenue ($)", row=1, col=1)
fig.update_yaxes(title_text="Avg Units", row=1, col=2)
fig.update_yaxes(title_text="Avg Revenue ($)", row=2, col=1)
fig.update_yaxes(title_text="Avg Units", row=2, col=2)

fig.show()

print("□ Seasonality patterns analyzed!")

# Print insights
print("\n□ SEASONALITY INSIGHTS:\n")
print("□ Top 3 Months by Revenue:")
top_months = monthly_sales.nlargest(3, 'SalesDollars')[['MonthName',
'SalesDollars']]
for idx, row in top_months.iterrows():
    print(f"    {row['MonthName']}: ${row['SalesDollars']:, .2f}/day")

print("\n□ Top 3 Days by Revenue:")
top_days = dow_sales.nlargest(3, 'SalesDollars')[['DayName',
'SalesDollars']]
for idx, row in top_days.iterrows():
    print(f"    {row['DayName']}: ${row['SalesDollars']:, .2f}/day")

print("\n"*70)

```

□ DEMAND FORECASTING - Step 3: Analyze Seasonality

```
=====
```

□ Seasonality patterns analyzed!

□ SEASONALITY INSIGHTS:

□ Top 3 Months by Revenue:
 January: \$963,033.16/day
 February: \$113,287.84/day

□ Top 3 Days by Revenue:
 Friday: \$943,204.24/day
 Saturday: \$895,586.45/day
 Sunday: \$490,739.96/day

```
=====
```

Prepare Data for Top A-Class Products

Selecting top 5 A-class products for detailed demand forecasting.

```
print("\n DEMAND FORECASTING - Step 4: Select Top Products for  
Forecasting\n")  
print("="*70)  
  
# Get top 5 A-class products  
top_products = product_revenue[product_revenue['ABC_Category'] ==  
'A'].head(5)  
  
print("\n Top 5 Products Selected for Demand Forecasting:\n")  
display(top_products[['Brand', 'Description', 'TotalRevenue',  
'ABC_Category']])  
  
# Prepare individual product time series  
product_forecasts = {}  
  
for idx, product in top_products.iterrows():  
    brand = product['Brand']  
    description = product['Description']  
  
    # Filter sales for this product  
    product_sales = sales_df[  
        (sales_df['Brand'] == brand) &  
        (sales_df['Description'] == description)  
    ].copy()  
  
    # Daily aggregation  
    product_daily = product_sales.groupby('SalesDate').agg({  
        'SalesQuantity': 'sum',  
        'SalesDollars': 'sum'  
    }).reset_index()  
  
    # Ensure complete date range (fill missing dates with 0)  
    date_range = pd.date_range(  
        start=daily_sales['SalesDate'].min(),  
        end=daily_sales['SalesDate'].max(),  
        freq='D'  
    )  
  
    product_daily =  
    product_daily.set_index('SalesDate').reindex(date_range,  
        fill_value=0).reset_index()  
    product_daily.columns = ['SalesDate', 'SalesQuantity',  
        'SalesDollars']  
  
    product_forecasts[f"{brand}_{description}"] = product_daily
```

```

    print(f" Prepared data for: {description} (Brand {brand})")

print(f"\n {len(product_forecasts)} products ready for forecasting!")
print("="*70)

```

DEMAND FORECASTING - Step 4: Select Top Products for Forecasting

Top 5 Products Selected for Demand Forecasting:

```

{"summary":{"name": "print(\\\\"=\\\\"*70)\\", "rows": 5, "fields": [{"column": "Brand", "properties": {"dtype": "number", "std": 2489, "min": 1233, "max": 8068, "num_unique_values": 5, "samples": [3545, 3405, 1233]}, {"column": "Description", "properties": {"dtype": "string", "num_unique_values": 5, "samples": ["Ketel One Vodka", "Tito's Handmade Vodka", "Jack Daniels No 7 Black"]}, {"column": "TotalRevenue", "properties": {"dtype": "number", "std": 67443.1306008423, "min": 275162.97, "max": 444810.74, "num_unique_values": 5, "samples": [357759.17, 275162.97, 344712.220000000003]}, {"column": "ABC_Category", "properties": {"dtype": "category", "num_unique_values": 1, "samples": ["A"]}], "semantic_type": "\\\"", "description": "\\\"", "type": "dataframe"}

```

- Prepared data for: Capt Morgan Spiced Rum (Brand 4261)
- Prepared data for: Ketel One Vodka (Brand 3545)
- Prepared data for: Jack Daniels No 7 Black (Brand 1233)
- Prepared data for: Absolut 80 Proof (Brand 8068)
- Prepared data for: Tito's Handmade Vodka (Brand 3405)

5 products ready for forecasting!

NEXT STEPS: Prophet Forecasting Models

Build Prophet Models for Top Products

Using Facebook Prophet to forecast demand for the next 14 days for each top product.

```
print("\n DEMAND FORECASTING - Step 5: Build Prophet Models\n")
print("="*70)

# Store forecast results
forecast_results = {}

for product_key, product_data in product_forecasts.items():
    print(f"\n Forecasting for: {product_key}")
    print("-" * 70)

    # Prepare data for Prophet (requires 'ds' and 'y' columns)
    prophet_df = product_data[['SalesDate', 'SalesQuantity']].copy()
    prophet_df.columns = ['ds', 'y']

    # Initialize Prophet model with weekly seasonality
    model = Prophet(
        daily_seasonality=True,
        weekly_seasonality=True,
        yearly_seasonality=False, # Not enough data for yearly
        seasonality_mode='multiplicative',
        changepoint_prior_scale=0.05
    )

    # Fit the model
    print(" Training model...", end=" ")
    model.fit(prophet_df)
    print("")

    # Create future dataframe for next 14 days
    future = model.make_future_dataframe(periods=14)

    # Make predictions
    print(" Generating forecast...", end=" ")
    forecast = model.predict(future)
    print("")

    # Store results
    forecast_results[product_key] = {
        'model': model,
        'forecast': forecast,
        'historical': prophet_df
    }

    # Calculate accuracy metrics on historical data
    historical_predictions =
forecast[forecast['ds'].isin(prophet_df['ds'])]
    mae = mean_absolute_error(prophet_df['y'],
historical_predictions['yhat'])
```

```

    rmse = np.sqrt(mean_squared_error(prophet_df['y'],
historical_predictions['yhat']))

    print(f"    □ Model Performance:")
    print(f"        MAE: {mae:.2f} units/day")
    print(f"        RMSE: {rmse:.2f} units/day")

print("\n" + "="*70)
print(f"□ Forecasting complete for {len(forecast_results)} products!")
print("="*70)

```

□ DEMAND FORECASTING - Step 5: Build Prophet Models

□ Forecasting for: 4261_Capt Morgan Spiced Rum

```

-----
Training model... □
Generating forecast... □
□ Model Performance:
    MAE: 135.44 units/day
    RMSE: 185.91 units/day

```

□ Forecasting for: 3545_Ketel One Vodka

```

-----
Training model... □
Generating forecast... □
□ Model Performance:
    MAE: 18.97 units/day
    RMSE: 27.90 units/day

```

□ Forecasting for: 1233_Jack Daniels No 7 Black

```

-----
Training model... □
Generating forecast... □
□ Model Performance:
    MAE: 62.80 units/day
    RMSE: 85.82 units/day

```

□ Forecasting for: 8068_Absolut 80 Proof

```

-----
Training model... □
Generating forecast... □
□ Model Performance:
    MAE: 36.08 units/day
    RMSE: 48.82 units/day

```

□ Forecasting for: 3405_Tito's Handmade Vodka

```

-----

```

```
Training model... ☐
Generating forecast... ☐
☐ Model Performance:
    MAE: 13.35 units/day
    RMSE: 18.08 units/day
```

```
=====
☐ Forecasting complete for 5 products!
=====
```

Visualize Forecasts for Each Product

Creating interactive charts showing historical data, forecasts, and confidence intervals for each product.

```
print("\n DEMAND FORECASTING - Step 6: Visualize Forecasts\n")
print("="*70)

for product_key, results in forecast_results.items():

    forecast = results['forecast']
    historical = results['historical']

    # Extract product name for title
    product_name = product_key.split('_', 1)[1].replace('_', ' ')

    # Create plot
    fig = go.Figure()

    # Historical data
    fig.add_trace(go.Scatter(
        x=historical['ds'],
        y=historical['y'],
        mode='lines+markers',
        name='Historical Sales',
        line=dict(color='steelblue', width=2),
        marker=dict(size=4)
    ))

    # Forecast
    future_data = forecast[forecast['ds'] > historical['ds'].max()]
    fig.add_trace(go.Scatter(
        x=future_data['ds'],
        y=future_data['yhat'],
        mode='lines+markers',
        name='Forecast',
        line=dict(color='red', width=2, dash='dash'),
        marker=dict(size=6)
```

```

))

# Confidence interval
fig.add_trace(go.Scatter(
    x=future_data['ds'].tolist() + future_data['ds'].tolist()[::-1],
    y=future_data['yhat_upper'].tolist() +
future_data['yhat_lower'].tolist()[::-1],
    fill='toself',
    fillcolor='rgba(255, 0, 0, 0.2)',
    line=dict(color='rgba(255,255,255,0)'),
    name='Confidence Interval',
    showlegend=True
))

fig.update_layout(
    title=f'📅 Demand Forecast: {product_name}',
    xaxis_title='Date',
    yaxis_title='Units Sold',
    hovermode='x unified',
    height=500,
    showlegend=True
)

fig.show()

print(f"📅 Visualization created for: {product_name}\n")

print("="*70)

```

📅 DEMAND FORECASTING - Step 6: Visualize Forecasts

=====

📅 Visualization created for: Capt Morgan Spiced Rum

📅 Visualization created for: Ketel One Vodka

📅 Visualization created for: Jack Daniels No 7 Black

📅 Visualization created for: Absolut 80 Proof

📅 Visualization created for: Tito's Handmade Vodka

=====

Create Forecast Summary Table

Generating a summary table with forecasted quantities for the next 14 days for all products.

```
print("\n DEMAND FORECASTING - Step 7: Forecast Summary\n")
print("="*70)

# Create summary dataframe
forecast_summary = []

for product_key, results in forecast_results.items():
    forecast = results['forecast']
    product_name = product_key.split('_', 1)[1].replace('_', ' ')

    # Get next 14 days forecast
    future_forecast = forecast[forecast['ds'] > results['historical']
    ['ds'].max()].head(14)

    for _, row in future_forecast.iterrows():
        forecast_summary.append({
            'Product': product_name,
            'Date': row['ds'].strftime('%Y-%m-%d'),
            'Forecasted_Units': max(0, round(row['yhat'])), # Ensure
non-negative
            'Lower_Bound': max(0, round(row['yhat_lower'])),
            'Upper_Bound': max(0, round(row['yhat_upper']))
        })

summary_df = pd.DataFrame(forecast_summary)

print("\n 14-DAY FORECAST SUMMARY (First 20 rows):\n")
display(summary_df.head(20))

# Aggregate by product
print("\n TOTAL FORECASTED DEMAND (Next 14 Days):\n")
product_totals = summary_df.groupby('Product').agg({
    'Forecasted_Units': 'sum',
    'Lower_Bound': 'sum',
    'Upper_Bound': 'sum'
}).reset_index()

product_totals.columns = ['Product', 'Total_Forecasted_Units',
    'Total_Lower_Bound', 'Total_Upper_Bound']
display(product_totals)

print("="*70)
```

DEMAND FORECASTING - Step 7: Forecast Summary

=====

□ 14-DAY FORECAST SUMMARY (First 20 rows):

```
{"summary":{"\n  \"name\": \"print(\\\"\\\"=\\\"\\\"*70)\\\", \"rows\": 20,\n  \"fields\": [\n    {\n      \"column\": \"Product\", \n      \"properties\": {\n        \"dtype\": \"category\", \n        \"num_unique_values\": 2, \n        \"samples\": [\n          \"Ketel One Vodka\", \n          \"Capt Morgan Spiced Rum\" \n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\" \n      }, \n      \"column\": \"Date\", \n      \"properties\": {\n        \"dtype\": \"object\", \n        \"num_unique_values\": 14, \n        \"samples\": [\n          \"2016-03-10\", \n          \"2016-03-12\" \n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\" \n      }, \n      \"column\": \"Forecasted_Units\", \n      \"properties\": {\n        \"dtype\": \"number\", \n        \"std\": 22, \n        \"min\": 0, \n        \"max\": 70, \n        \"num_unique_values\": 9, \n        \"samples\": [\n          2 \n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\" \n      }, \n      \"column\": \"Lower_Bound\", \n      \"properties\": {\n        \"dtype\": \"number\", \n        \"std\": 0, \n        \"min\": 0, \n        \"max\": 0, \n        \"num_unique_values\": 1, \n        \"samples\": [\n          0 \n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\" \n      }, \n      \"column\": \"Upper_Bound\", \n      \"properties\": {\n        \"dtype\": \"number\", \n        \"std\": 124, \n        \"min\": 39, \n        \"max\": 542, \n        \"num_unique_values\": 20, \n        \"samples\": [\n          238 \n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\" \n      } \n    ] \n  }, \"type\": \"dataframe\"}
```

□ TOTAL FORECASTED DEMAND (Next 14 Days):

```
{"summary":{"\n  \"name\": \"product_totals\", \n  \"rows\": 5, \n  \"fields\": [\n    {\n      \"column\": \"Product\", \n      \"properties\": {\n        \"dtype\": \"string\", \n        \"num_unique_values\": 5, \n        \"samples\": [\n          \"Capt Morgan Spiced Rum\", \n          \"Tito's Handmade Vodka\", \n          \"Jack Daniels No 7 Black\" \n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\" \n      }, \n      \"column\": \"Total_Forecasted_Units\", \n      \"properties\": {\n        \"dtype\": \"number\", \n        \"std\": 292, \n        \"min\": 2, \n        \"max\": 607, \n        \"num_unique_values\": 5, \n        \"samples\": [\n          9, \n          266, \n          2 \n        ], \n        \"semantic_type\": \"\", \n        \"description\": \"\" \n      }, \n      \"column\": \"Total_Lower_Bound\", \n      \"properties\": {\n        \"dtype\": \"number\", \n        \"std\": \n
```

```
0,\n          \"min\": 0,\n          \"max\": 1,\n          \"num_unique_values\": 2,\n          \"samples\": [\n              0,\n              1\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\",\n          \"Total_Upper_Bound\": 1,\n          \"properties\": {\n              \"dtype\": \"number\",\n              \"std\": 3097,\n              \"min\": 1122,\n              \"max\": 8872,\n              \"num_unique_values\": 5,\n              \"samples\": [\n                  2563,\n                  3011\n              ],\n              \"semantic_type\": \"\",\n              \"description\": \"\"\n          }\n      ],\n      \"type\": \"dataframe\", \"variable_name\": \"product_totals\"}
```

Save Forecast Results

Saving all forecasts to CSV files for business use.

```
print("\n DEMAND FORECASTING - Step 8: Save Forecast Results\n")
print("="*70)

# Save detailed forecast
detailed_forecast_path = '/content/Demand_Forecast_Detailed.csv'
summary_df.to_csv(detailed_forecast_path, index=False)
print(f"Detailed forecast saved to: {detailed_forecast_path}")

# Save product totals
totals_path = '/content/Demand_Forecast_Summary.csv'
product_totals.to_csv(totals_path, index=False)
print(f"Forecast summary saved to: {totals_path}")

print("\n Files saved successfully!")
print("="*70)
```

DEMAND FORECASTING - Step 8: Save Forecast Results

```
=====
Detailed forecast saved to: /content/Demand_Forecast_Detailed.csv
Forecast summary saved to: /content/Demand_Forecast_Summary.csv

Files saved successfully!
=====
```

Business Recommendations from Forecasts

Generating actionable insights based on the forecasts.

```

print("\n DEMAND FORECASTING - Step 9: Business Recommendations\n")
print("="*70)

print("\n KEY FORECASTING INSIGHTS:\n")

for idx, row in product_totals.iterrows():
    product = row['Product']
    forecasted = row['Total_Forecasted_Units']
    lower = row['Total_Lower_Bound']
    upper = row['Total_Upper_Bound']

    print(f"{idx+1} {product}:")
    print(f"    Expected demand (14 days): {forecasted:,.0f} units")
    print(f"    Range: {lower:,.0f} - {upper:,.0f} units")
    print(f"    RECOMMENDATION: Order {upper:,.0f} units to avoid
stockouts")
    print(f"        (based on upper confidence bound)\n")

print("="*70)
print(" DEMAND FORECASTING COMPLETE!")
print("="*70)

```

DEMAND FORECASTING - Step 9: Business Recommendations

KEY FORECASTING INSIGHTS:

- 1 Absolut 80 Proof:**
 Expected demand (14 days): 607 units
 Range: 1 - 1,843 units
 RECOMMENDATION: Order 1,843 units to avoid stockouts
 (based on upper confidence bound)
- 2 Capt Morgan Spiced Rum:**
 Expected demand (14 days): 9 units
 Range: 0 - 2,563 units
 RECOMMENDATION: Order 2,563 units to avoid stockouts
 (based on upper confidence bound)
- 3 Jack Daniels No 7 Black:**
 Expected demand (14 days): 2 units
 Range: 0 - 1,122 units
 RECOMMENDATION: Order 1,122 units to avoid stockouts
 (based on upper confidence bound)
- 4 Ketel One Vodka:**
 Expected demand (14 days): 570 units
 Range: 0 - 8,872 units
 RECOMMENDATION: Order 8,872 units to avoid stockouts

(based on upper confidence bound)

5 Tito's Handmade Vodka:

Expected demand (14 days): 266 units

Range: 0 - 3,011 units

□ RECOMMENDATION: Order 3,011 units to avoid stockouts
(based on upper confidence bound)

=====

□ DEMAND FORECASTING COMPLETE!

=====

PHASE 2.3: REORDER POINT ANALYSIS

Step 1: Calculate Average Daily Demand (January Data)

Using January data (which is more representative) to calculate realistic daily demand for top products.

```
print("\n□ REORDER POINT ANALYSIS - Step 1: Calculate Daily Demand\n")
print("="*70)

# Use only January data for more reliable averages
sales_df = cleaned_data['sales'].copy()
jan_sales = sales_df[sales_df['SalesDate'].dt.month == 1]

# Calculate daily demand for top 5 products
top_products_list = [
    (4261, 'Capt Morgan Spiced Rum'),
    (3545, 'Ketel One Vodka'),
    (1233, 'Jack Daniels No 7 Black'),
    (8068, 'Absolut 80 Proof'),
    (3405, "Tito's Handmade Vodka")
]

demand_stats = []

for brand, description in top_products_list:
    # Filter for this product
    product_sales = jan_sales[
        (jan_sales['Brand'] == brand) &
        (jan_sales['Description'] == description)
    ]
```

```

# Daily aggregation
daily_demand = product_sales.groupby('SalesDate')
['SalesQuantity'].sum()

# Calculate statistics
avg_daily_demand = daily_demand.mean()
std_daily_demand = daily_demand.std()
max_daily_demand = daily_demand.max()
min_daily_demand = daily_demand.min()

demand_stats.append({
    'Brand': brand,
    'Product': description,
    'Avg_Daily_Demand': round(avg_daily_demand, 2),
    'Std_Daily_Demand': round(std_daily_demand, 2),
    'Max_Daily_Demand': int(max_daily_demand),
    'Min_Daily_Demand': int(min_daily_demand)
})

demand_df = pd.DataFrame(demand_stats)

print("\n DAILY DEMAND STATISTICS (Based on January 2016):\n")
display(demand_df)

print("\n\n Daily demand calculated from historical data!")
print("="*70)

```

REORDER POINT ANALYSIS - Step 1: Calculate Daily Demand

```
=====
\n DAILY DEMAND STATISTICS (Based on January 2016):
```

```

{"summary":{"\n  \"name\": \"demand_df\", \n  \"rows\": 5, \n
\n \"fields\": [\n    {\n      \"column\": \"Brand\", \n
\n \"properties\": {\n        \"dtype\": \"number\", \n        \"std\":
2489, \n        \"min\": 1233, \n        \"max\": 8068, \n
\n \"num_unique_values\": 5, \n        \"samples\": [\n          3545, \n
3405, \n          1233 \n        ], \n        \"semantic_type\": \"\", \n
\n \"description\": \"\", \n      }, \n    {\n      \"column\":
\n \"Product\", \n      \"properties\": {\n        \"dtype\": \"string\", \n
n        \"num_unique_values\": 5, \n        \"samples\": [\n
\n \"Ketel One Vodka\", \n        \"Tito's Handmade Vodka\", \n
\n \"Jack Daniels No 7 Black\" \n        ], \n        \"semantic_type\":
\n \"\", \n        \"description\": \"\", \n      }, \n    {\n
\n \"column\": \"Avg_Daily_Demand\", \n        \"properties\": {\n
\n \"dtype\": \"number\", \n        \"std\": 128.41535266470282, \n
\n \"min\": 268.65, \n        \"max\": 578.61, \n
\n \"num_unique_values\": 5, \n        \"samples\": [\n          359.9, \n

```

```

268.65,\n          273.39\n          ],\n          \"semantic_type\":\n          \"\", \n          \"description\": \"\", \n          } \n          }, \n          {\n          \"column\": \"Std_Daily_Demand\", \n          \"properties\": {\n          \"dtype\": \"number\", \n          \"std\": 77.3265282422533, \n          \"min\": 133.02, \n          \"max\": 325.75, \n          \"num_unique_values\": 5, \n          \"samples\": [\n          212.32, \n          169.15, \n          133.02\n          ], \n          \"semantic_type\":\n          \"\", \n          \"description\": \"\", \n          } \n          }, \n          {\n          \"column\": \"Max_Daily_Demand\", \n          \"properties\": {\n          \"dtype\": \"number\", \n          \"std\": 272, \n          \"min\": 566, \n          \"max\": 1227, \n          \"num_unique_values\": 5, \n          \"samples\": [\n          844, \n          672, \n          566\n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\", \n          } \n          }, \n          {\n          \"column\": \"Min_Daily_Demand\", \n          \"properties\": {\n          \"dtype\": \"number\", \n          \"std\": 38, \n          \"min\": 72, \n          \"max\": 172, \n          \"num_unique_values\": 5, \n          \"samples\": [\n          110, \n          72, \n          94\n          ], \n          \"semantic_type\": \"\", \n          \"description\": \"\", \n          } \n          } \n          ] \n          } \n          }, \n          {\"type\": \"dataframe\", \"variable_name\": \"demand_df\"}

```

☐ Daily demand calculated from historical data!

Calculate Lead Time from Purchase Data

Analyzing purchase records to find how long it takes suppliers to deliver products.

```

print("\n☐ REORDER POINT ANALYSIS - Step 2: Calculate Lead Time\n")
print("="*70)

# Use purchases data
purchases_df = cleaned_data['purchases'].copy()

# Calculate lead time (days between PO Date and Receiving Date)
purchases_df['LeadTime_Days'] = (
    purchases_df['ReceivingDate'] - purchases_df['PODate']
).dt.days

# Remove negative or zero lead times (data errors)
purchases_df = purchases_df[purchases_df['LeadTime_Days'] > 0]

# Calculate lead time for top products
lead_time_stats = []

for brand, description in top_products_list:
    # Filter for this product

```

```

product_purchases = purchases_df[purchases_df['Brand'] == brand]

if len(product_purchases) > 0:
    avg_lead_time = product_purchases['LeadTime_Days'].mean()
    max_lead_time = product_purchases['LeadTime_Days'].max()
    min_lead_time = product_purchases['LeadTime_Days'].min()
else:
    # Default values if no purchase data
    avg_lead_time = 7 # Assume 1 week default
    max_lead_time = 14
    min_lead_time = 3

lead_time_stats.append({
    'Brand': brand,
    'Product': description,
    'Avg_Lead_Time_Days': round(avg_lead_time, 1),
    'Max_Lead_Time_Days': int(max_lead_time),
    'Min_Lead_Time_Days': int(min_lead_time)
})

lead_time_df = pd.DataFrame(lead_time_stats)

print("\n LEAD TIME STATISTICS:\n")
display(lead_time_df)

print("\n\n Lead time calculated from purchase history!")
print("=*70)

```

REORDER POINT ANALYSIS - Step 2: Calculate Lead Time

LEAD TIME STATISTICS:

```

{"summary":{"name": "lead_time_df", "rows": 5, "fields": [{"column": "Brand", "properties": {"dtype": "number", "std": 2489, "min": 1233, "max": 8068, "num_unique_values": 5, "samples": [3545, 3405, 1233]}, {"column": "Product", "properties": {"dtype": "string", "std": 0.10954451150103309, "min": 7.3, "max": 7.6, "num_unique_values": 2}], "description": "Lead time statistics for different brands and products."}}

```

```

3,\n          \"samples\": [\n          7.3,\n          7.4,\n          7.6\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\",\n          }\n          },\n          {\n          \"column\":\n          \"Max_Lead_Time_Days\",\n          \"properties\": {\n          \"dtype\":\n          \"number\",\n          \"std\": 0,\n          \"min\": 14,\n          \"max\": 14,\n          \"num_unique_values\": 1,\n          \"samples\":\n          [\n          14\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\",\n          }\n          },\n          {\n          \"column\":\n          \"Min_Lead_Time_Days\",\n          \"properties\": {\n          \"dtype\":\n          \"number\",\n          \"std\": 0,\n          \"min\": 3,\n          \"max\": 3,\n          \"num_unique_values\": 1,\n          \"samples\":\n          [\n          3\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\",\n          }\n          }\n          ]\n          }\", \"type\": \"dataframe\", \"variable_name\": \"lead_time_df\"}

```

□ Lead time calculated from purchase history!

Calculate Safety Stock

Determining buffer inventory based on demand variability and desired service level.

```

print("\n□ REORDER POINT ANALYSIS - Step 3: Calculate Safety Stock\n")
print("="*70)

# Service level factor (Z-score)
# 95% service level = 1.65, 99% service level = 2.33
SERVICE_LEVEL = 0.95
Z_SCORE = 1.65 # 95% service level

print(f"Using {SERVICE_LEVEL*100:.0f}% service level (Z-score: {Z_SCORE})")
print("\n" + "-"*70)

# Merge demand and lead time data
rop_analysis = demand_df.merge(
    lead_time_df[['Brand', 'Avg_Lead_Time_Days',
    'Max_Lead_Time_Days']],
    on='Brand'
)

# Calculate safety stock
# Formula: Z × Std_Demand × √Lead_Time
rop_analysis['Safety_Stock'] = (
    Z_SCORE *
    rop_analysis['Std_Daily_Demand'] *
    np.sqrt(rop_analysis['Avg_Lead_Time_Days'])
)

```

```

).round(0).astype(int)

print("\n SAFETY STOCK CALCULATION:\n")
display(rop_analysis[['Product', 'Std_Daily_Demand',
'Avg_Lead_Time_Days', 'Safety_Stock']])

print("\n Safety stock calculated!")
print("="*70)

```

REORDER POINT ANALYSIS - Step 3: Calculate Safety Stock

Using 95% service level (Z-score: 1.65)

□ SAFETY STOCK CALCULATION:

```
{
  "summary": "{\n  \"name\": \"print(\\\"\\\"=\\\"\\\"*70)\\\",\\n  \"rows\": 5,\\n  \"fields\": [\n    {\n      \"column\": \"Product\\\",\\n      \"dtype\": \"string\\\",\\n      \"num_unique_values\": 5,\\n      \"samples\": [\n        \"Ketel One Vodka\\\",\\n        \"Tito's Handmade Vodka\\\",\\n        \"Jack Daniels No 7 Black\\\",\\n        \"semantic_type\": \"\\\",\\n        \"description\": \"\\\"\\\"\\n      }\\n    },\\n    {\n      \"column\": \"Std_Daily_Demand\\\",\\n      \"properties\": {\n        \"dtype\": \"number\\\",\\n        \"std\": 77.3265282422533,\\n        \"min\": 133.02,\\n        \"max\": 325.75,\\n        \"num_unique_values\": 5,\\n        \"samples\": [\n          212.32,\\n          169.15,\\n          133.02\\n        ],\\n        \"semantic_type\": \"\\\"\\\",\\n        \"description\": \"\\\"\\\"\\n      }\\n    },\\n    {\n      \"column\": \"Avg_Lead_Time_Days\\\",\\n      \"properties\": {\n        \"dtype\": \"number\\\",\\n        \"std\": 0.10954451150103309,\\n        \"min\": 7.3,\\n        \"max\": 7.6,\\n        \"num_unique_values\": 3,\\n        \"samples\": [\n          7.3,\\n          7.4,\\n          7.6\\n        ],\\n        \"semantic_type\": \"\\\"\\\",\\n        \"description\": \"\\\"\\\"\\n      }\\n    },\\n    {\n      \"column\": \"Safety_Stock\\\",\\n      \"properties\": {\n        \"dtype\": \"number\\\",\\n        \"std\": 341,\\n        \"min\": 597,\\n        \"max\": 1452,\\n        \"num_unique_values\": 5,\\n        \"samples\": [\n          953,\\n          769,\\n          597\\n        ],\\n        \"semantic_type\": \"\\\"\\\",\\n        \"description\": \"\\\"\\\"\\n      }\\n    }\\n  ]\\n}\", \"type\": \"dataframe\"}
```

□ Safety stock calculated!

Calculate Reorder Point

Computing the exact inventory level at which to place new orders.

```
print("\n REORDER POINT ANALYSIS - Step 4: Calculate Reorder Point\n")
print("="*70)

# Calculate Reorder Point
# ROP = (Average Daily Demand × Lead Time) + Safety Stock
rop_analysis['Reorder_Point'] = (
    (rop_analysis['Avg_Daily_Demand'] *
rop_analysis['Avg_Lead_Time_Days']) +
    rop_analysis['Safety_Stock']
).round(0).astype(int)

# Calculate for max lead time scenario (worst case)
rop_analysis['Reorder_Point_WorstCase'] = (
    (rop_analysis['Avg_Daily_Demand'] *
rop_analysis['Max_Lead_Time_Days']) +
    rop_analysis['Safety_Stock']
).round(0).astype(int)

print("\n REORDER POINT ANALYSIS RESULTS:\n")
display(rop_analysis[[
    'Product',
    'Avg_Daily_Demand',
    'Avg_Lead_Time_Days',
    'Safety_Stock',
    'Reorder_Point',
    'Reorder_Point_WorstCase'
]])

print("\n Reorder points calculated!")
print("="*70)
```

REORDER POINT ANALYSIS - Step 4: Calculate Reorder Point

=====

REORDER POINT ANALYSIS RESULTS:

```
{"summary": "{\n  \"name\": \"print(\\\"\\\"=\\\"\\\"*70)\\\",\\n  \"rows\": 5,\\n  \"fields\": [\n    {\n      \"column\": \"Product\",\\n      \"properties\": {\n        \"dtype\": \"string\",\\n        \"num_unique_values\": 5,\\n        \"samples\": [\n          \"Ketel One Vodka\",\\n          \"Tito's Handmade Vodka\",\\n          \"Jack Daniels No 7 Black\",\\n          ],\\n        \"semantic_type\": \"\",\\n        \"description\": \"\",\\n      }\\n    },\\n    {\n      \"column\":
```

```

{"Avg_Daily_Demand": {"properties": {"dtype": "number", "std": 128.41535266470282, "min": 268.65, "max": 578.61, "num_unique_values": 5, "samples": [359.9, 268.65, 273.39]}, "semantic_type": "", "description": ""}, {"Avg_Lead_Time_Days": {"properties": {"dtype": "number", "std": 0.10954451150103309, "min": 7.3, "max": 7.6, "num_unique_values": 3, "samples": [7.3, 7.4, 7.6]}, "semantic_type": "", "description": ""}, {"Safety_Stock": {"properties": {"dtype": "number", "std": 341, "min": 597, "max": 1452, "num_unique_values": 5, "samples": [953, 769, 597]}, "semantic_type": "", "description": ""}, {"Reorder_Point": {"properties": {"dtype": "number", "std": 1251, "min": 2620, "max": 5676, "num_unique_values": 5, "samples": [3616, 2811, 2620]}, "semantic_type": "", "description": ""}, {"Reorder_Point_WorstCase": {"properties": {"dtype": "number", "std": 2130, "min": 4424, "max": 9553, "num_unique_values": 5, "samples": [5992, 4530, 4424]}, "semantic_type": "", "description": ""}]}, {"type": "dataframe"}

```

☐ Reorder points calculated!

Visualize Reorder Point Components

Creating a stacked bar chart showing how reorder points are composed.

```

print("\n☐ REORDER POINT ANALYSIS - Step 5: Visualize Components\n")
print("=*70)

# Calculate base demand during lead time
rop_analysis['Base_Demand_During_Lead_Time'] = (
    rop_analysis['Avg_Daily_Demand'] *
    rop_analysis['Avg_Lead_Time_Days']
).round(0).astype(int)

# Create stacked bar chart
fig = go.Figure()

```

```

# Base demand during lead time
fig.add_trace(go.Bar(
    name='Demand During Lead Time',
    x=rop_analysis['Product'],
    y=rop_analysis['Base_Demand_During_Lead_Time'],
    marker_color='steelblue'
))

# Safety stock
fig.add_trace(go.Bar(
    name='Safety Stock (Buffer)',
    x=rop_analysis['Product'],
    y=rop_analysis['Safety_Stock'],
    marker_color='orange'
))

fig.update_layout(
    title='📦 Reorder Point Components by Product',
    xaxis_title='Product',
    yaxis_title='Units',
    barmode='stack',
    height=500,
    showlegend=True,
    hovermode='x unified'
)

fig.show()

print("📦 Visualization created!")
print("="*70)

```

📦 REORDER POINT ANALYSIS - Step 5: Visualize Components

=====

📦 Visualization created!

=====

Create Inventory Management Guidelines

Generating actionable recommendations for each product.

```

print("\n📦 REORDER POINT ANALYSIS - Step 6: Management Guidelines\n")
print("="*70)

print("\n📦 INVENTORY MANAGEMENT RECOMMENDATIONS:\n")

```

```

for idx, row in rop_analysis.iterrows():
    product = row['Product']
    avg_demand = row['Avg_Daily_Demand']
    lead_time = row['Avg_Lead_Time_Days']
    safety_stock = row['Safety_Stock']
    rop = row['Reorder_Point']
    rop_worst = row['Reorder_Point_WorstCase']

    print(f"\n{'='*70}")
    print(f"□ {product}")
    print(f"{'='*70}")
    print(f"Average Daily Demand: {avg_demand:.0f} units/day")
    print(f"Average Lead Time: {lead_time:.0f} days")
    print(f"Safety Stock: {safety_stock} units")
    print(f"\n□ REORDER POINT: {rop} units")
    print(f"    △ Worst Case ROP: {rop_worst} units (max lead time)")
    print(f"\n□ MANAGEMENT ACTIONS:")
    print(f"    1. When inventory drops to {rop} units → Place order immediately")
    print(f"    2. Maintain minimum {safety_stock} units as safety buffer")
    print(f"    3. Order quantity should cover {lead_time:.0f} days of demand")
    print(f"    4. Expected consumption during lead time: {int(avg_demand * lead_time)} units")

    print(f"\n{'='*70}")
    print(f"□ REORDER POINT ANALYSIS COMPLETE!")
    print(f"{'='*70}")

```

□ REORDER POINT ANALYSIS - Step 6: Management Guidelines

=====

□ INVENTORY MANAGEMENT RECOMMENDATIONS:

=====

□ Capt Morgan Spiced Rum

=====

Average Daily Demand: 579 units/day

Average Lead Time: 7 days

Safety Stock: 1452 units

□ REORDER POINT: 5676 units

△ Worst Case ROP: 9553 units (max lead time)

□ MANAGEMENT ACTIONS:

1. When inventory drops to 5676 units → Place order immediately
2. Maintain minimum 1452 units as safety buffer
3. Order quantity should cover 7 days of demand
4. Expected consumption during lead time: 4223 units

=====

☐ Ketel One Vodka

=====

Average Daily Demand: 360 units/day
Average Lead Time: 7 days
Safety Stock: 953 units

☐ REORDER POINT: 3616 units
 △ Worst Case ROP: 5992 units (max lead time)

☐ MANAGEMENT ACTIONS:
 1. When inventory drops to 3616 units → Place order immediately
 2. Maintain minimum 953 units as safety buffer
 3. Order quantity should cover 7 days of demand
 4. Expected consumption during lead time: 2663 units

=====

☐ Jack Daniels No 7 Black

=====

Average Daily Demand: 273 units/day
Average Lead Time: 7 days
Safety Stock: 597 units

☐ REORDER POINT: 2620 units
 △ Worst Case ROP: 4424 units (max lead time)

☐ MANAGEMENT ACTIONS:
 1. When inventory drops to 2620 units → Place order immediately
 2. Maintain minimum 597 units as safety buffer
 3. Order quantity should cover 7 days of demand
 4. Expected consumption during lead time: 2023 units

=====

☐ Absolut 80 Proof

=====

Average Daily Demand: 312 units/day
Average Lead Time: 7 days
Safety Stock: 672 units

☐ REORDER POINT: 2978 units
 △ Worst Case ROP: 5035 units (max lead time)

☐ MANAGEMENT ACTIONS:
 1. When inventory drops to 2978 units → Place order immediately
 2. Maintain minimum 672 units as safety buffer

3. Order quantity should cover 7 days of demand
4. Expected consumption during lead time: 2306 units

=====

□ Tito's Handmade Vodka

=====

Average Daily Demand: 269 units/day
Average Lead Time: 8 days
Safety Stock: 769 units

□ REORDER POINT: 2811 units

△ Worst Case ROP: 4530 units (max lead time)

□ MANAGEMENT ACTIONS:

1. When inventory drops to 2811 units → Place order immediately
2. Maintain minimum 769 units as safety buffer
3. Order quantity should cover 8 days of demand
4. Expected consumption during lead time: 2041 units

=====

□ REORDER POINT ANALYSIS COMPLETE!

=====

Save Reorder Point Analysis

Exporting the analysis to CSV for business use.

```
print("\n□ REORDER POINT ANALYSIS - Step 7: Save Results\n")
print("="*70)

# Save full analysis
rop_output_path = '/content/Reorder_Point_Analysis.csv'
rop_analysis.to_csv(rop_output_path, index=False)

print(f"□ Reorder Point Analysis saved to: {rop_output_path}")

# Create summary for quick reference
rop_summary = rop_analysis[[
    'Product',
    'Avg_Daily_Demand',
    'Avg_Lead_Time_Days',
    'Reorder_Point',
    'Safety_Stock'
]].copy()

rop_summary_path = '/content/Reorder_Point_Summary.csv'
rop_summary.to_csv(rop_summary_path, index=False)
```

```
print(f"📄 Quick Reference Summary saved to: {rop_summary_path}")

print("\n📄 Files saved successfully!")
print("="*70)
```

📄 REORDER POINT ANALYSIS - Step 7: Save Results

```
=====
📄 Reorder Point Analysis saved to: /content/Reorder_Point_Analysis.csv
📄 Quick Reference Summary saved to: /content/Reorder_Point_Summary.csv

📄 Files saved successfully!
=====
```

Compare Current vs. Recommended Inventory Levels

Checking if current inventory (from end inventory data) is above or below reorder points.

```
print("\n📄 REORDER POINT ANALYSIS - Step 8: Current Inventory Check\
n")
print("="*70)

# Get end inventory data
end_inv = cleaned_data['end_inventory'].copy()

# Check current inventory for top products
current_inventory = []

for brand, description in top_products_list:
    # Filter for this product across all stores
    product_inv = end_inv[end_inv['Brand'] == brand]

    if len(product_inv) > 0:
        total_on_hand = product_inv['onHand'].sum()
        stores_stocking = len(product_inv)
        avg_per_store = total_on_hand / stores_stocking if
stores_stocking > 0 else 0
    else:
        total_on_hand = 0
        stores_stocking = 0
        avg_per_store = 0

    current_inventory.append({
        'Product': description,
        'Total_OnHand': int(total_on_hand),
```

```

        'Stores_Stocking': stores_stocking,
        'Avg_Per_Store': round(avg_per_store, 1)
    })

current_inv_df = pd.DataFrame(current_inventory)

# Merge with reorder point analysis
inventory_status = rop_analysis[['Product', 'Reorder_Point',
'Safety_Stock']].merge(
    current_inv_df, on='Product'
)

# Calculate status
inventory_status['Status'] = inventory_status.apply(
    lambda row: ' Above ROP' if row['Total_OnHand'] >
row['Reorder_Point']
    else ' Below ROP - ORDER NOW!' if row['Total_OnHand'] <
row['Reorder_Point']
    else ' At ROP - Monitor Closely',
    axis=1
)

inventory_status['Units_Above_Below_ROP'] = (
    inventory_status['Total_OnHand'] -
inventory_status['Reorder_Point']
)

print("\n CURRENT INVENTORY STATUS vs REORDER POINTS:\n")
display(inventory_status)

print("\n ACTION ITEMS:")
below_rop = inventory_status[inventory_status['Units_Above_Below_ROP']
< 0]
if len(below_rop) > 0:
    print(f"\n URGENT: {len(below_rop)} product(s) below reorder
point!")
    for _, row in below_rop.iterrows():
        shortage = abs(row['Units_Above_Below_ROP'])
        print(f"    • {row['Product']}: {shortage} units below ROP -
Order immediately!")
else:
    print("\n All products are at or above reorder points!")

print("\n*70)

```

REORDER POINT ANALYSIS - Step 8: Current Inventory Check

=====

□ CURRENT INVENTORY STATUS vs REORDER POINTS:

```
{
  "summary": {
    "name": "inventory_status",
    "rows": 5,
    "fields": [
      {
        "column": "Product",
        "dtype": "string",
        "num_unique_values": 5,
        "samples": [
          "Ketel One Vodka",
          "Tito's Handmade Vodka",
          "Jack Daniels No 7 Black",
          ""
        ],
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "Reorder_Point",
        "dtype": "number",
        "std": 1251,
        "min": 2620,
        "max": 5676,
        "num_unique_values": 5,
        "samples": [
          3616,
          2811,
          2620
        ],
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "Safety_Stock",
        "dtype": "number",
        "std": 341,
        "min": 597,
        "max": 1452,
        "num_unique_values": 5,
        "samples": [
          953,
          769,
          597
        ],
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "Total_OnHand",
        "dtype": "number",
        "std": 1848,
        "min": 12268,
        "max": 16770,
        "num_unique_values": 5,
        "samples": [
          16770,
          12268,
          15047
        ],
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "Stores_Stocking",
        "dtype": "number",
        "std": 0,
        "min": 79,
        "max": 79,
        "num_unique_values": 1,
        "samples": [
          79
        ],
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "Avg_Per_Store",
        "dtype": "number",
        "std": 23.408759044426084,
        "min": 155.3,
        "max": 212.3,
        "num_unique_values": 4,
        "samples": [
          190.5
        ],
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "Status",
        "dtype": "category",
        "num_unique_values": 1,
        "samples": [
          "\ud83d\udfe2 Above ROP"
        ],
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "Units_Above_Below_ROP",
        "dtype": "number",
        "std": 1491,
        "min": 9457,
        "max": 13154,
        "num_unique_values": 5,
        "samples": [
          13154
        ],
        "semantic_type": "",
        "description": ""
      }
    ]
  },
  "type": "dataframe",
  "variable_name": "inventory_status"
}
```

□ ACTION ITEMS:

□ All products are at or above reorder points!

=====

PHASE 2.4: LEAD TIME ANALYSIS

Deep-dive into supplier efficiency, delivery consistency, and procurement optimization opportunities.

Vendor Performance Metrics

Analyze supplier delivery speeds and reliability

Prepare Lead Time Data

Calculate lead times with robust data validation

```
print("\n□ LEAD TIME ANALYSIS - Step 1: Prepare Lead Time Data\n")
print("="*70)

purchases_df = cleaned_data['purchases'].copy()

# Ensure datetime format
purchases_df['PODate'] = pd.to_datetime(purchases_df['PODate'])
purchases_df['ReceivingDate'] =
pd.to_datetime(purchases_df['ReceivingDate'])

# Calculate lead time
purchases_df['LeadTime_Days'] = (
    purchases_df['ReceivingDate'] - purchases_df['PODate']
).dt.days

# Remove invalid lead times (negative or zero)
valid_purchases = purchases_df[purchases_df['LeadTime_Days'] > 0]

print(f"□ Valid purchase records: {len(valid_purchases):,}")
print(f"□ Lead time range: {valid_purchases['LeadTime_Days'].min()} to
{valid_purchases['LeadTime_Days'].max()} days")
print(f"□ Average lead time:
{valid_purchases['LeadTime_Days'].mean():.1f} days")
print("="*70)
```

□ LEAD TIME ANALYSIS - Step 1: Prepare Lead Time Data

```
=====
□ Valid purchase records: 2,372,474
□ Lead time range: 3 to 14 days
□ Average lead time: 7.6 days
=====
```

Vendor Performance Metrics

Calculate average lead times and consistency scores by supplier

```
print("\n□ LEAD TIME ANALYSIS - Step 2: Vendor-Level Statistics\n")
print("="*70)

# Group by vendor (using VendorName if available, fallback to
VendorNumber)
groupby_cols = ['VendorNumber']
if 'VendorName' in valid_purchases.columns:
    groupby_cols.append('VendorName')

vendor_stats = valid_purchases.groupby(groupby_cols).agg(
    Avg_Lead_Time=('LeadTime_Days', 'mean'),
    Median_Lead_Time=('LeadTime_Days', 'median'),
    Std_Lead_Time=('LeadTime_Days', 'std'),
    Min_Lead_Time=('LeadTime_Days', 'min'),
    Max_Lead_Time=('LeadTime_Days', 'max'),
    Total_POs=('LeadTime_Days', 'count'),
    Total_Spend=('Dollars', 'sum'),
    Total_Quantity=('Quantity', 'sum')
).reset_index()

# Try to get vendor names from purchase data
if 'VendorName' in valid_purchases.columns:
    vendor_names = valid_purchases.groupby('VendorNumber')
['VendorName'].first().to_dict()
    vendor_stats['Vendor_Name'] =
vendor_stats['VendorNumber'].map(vendor_names)
else:
    # Create from InvoicePurchases if available
    try:
        invoice_df = cleaned_data['invoice_purchases']
        if 'VendorName' in invoice_df.columns:
            vendor_names = invoice_df.groupby('VendorNumber')
['VendorName'].first().to_dict()
            vendor_stats['Vendor_Name'] =
vendor_stats['VendorNumber'].map(vendor_names)
```

```

except:
    vendor_stats['Vendor_Name'] = 'Unknown'

# Clean up nulls and round
vendor_stats['Std_Lead_Time'] =
vendor_stats['Std_Lead_Time'].fillna(0).round(2)
vendor_stats['Avg_Lead_Time'] = vendor_stats['Avg_Lead_Time'].round(2)
vendor_stats['CV'] = (vendor_stats['Std_Lead_Time'] /
vendor_stats['Avg_Lead_Time']).round(2)

# FIX: Apply minimum PO threshold BEFORE classification
MIN_PO_THRESHOLD = 10
vendor_stats = vendor_stats[vendor_stats['Total_POs'] >=
MIN_PO_THRESHOLD] # Filter here!

print(f"Filtered to {len(vendor_stats)} vendors with
≥{MIN_PO_THRESHOLD} POs")

print(f"\nTOP 10 FASTEST VENDORS:")
display(vendor_stats.nsmallest(10, 'Avg_Lead_Time')[['VendorNumber',
'Vendor_Name', 'Avg_Lead_Time', 'Median_Lead_Time', 'Total_POs',
'CV']])

print(f"\nTOP 10 SLOWEST VENDORS:")
display(vendor_stats.nlargest(10, 'Avg_Lead_Time')[['VendorNumber',
'Vendor_Name', 'Avg_Lead_Time', 'Max_Lead_Time', 'Total_POs',
'Total_Spend']])
print("="*70)

```

LEAD TIME ANALYSIS - Step 2: Vendor-Level Statistics

```
=====
Filtered to 120 vendors with ≥10 POs
```

```
TOP 10 FASTEST VENDORS:
```

```

{"summary":{"\n  \"name\": \"print(\\\"\\\"=\\\"\\\"*70)\\\",\\n  \"rows\": 10,\\n
\"fields\": [\\n    {\\n      \"column\": \"VendorNumber\\\",\\n
\"properties\": {\\n        \"dtype\": \"number\\\",\\n        \"std\":
29456,\\n        \"min\": 1650,\\n        \"max\": 98450,\\n
\"num_unique_values\": 10,\\n        \"samples\": [\\n          98450,\\n
5083,\\n          8150\\n        ],\\n        \"semantic_type\": \"\\\",\\n
\"description\": \"\\\"\\n      }\\n    },\\n    {\\n      \"column\":
\"Vendor_Name\\\",\\n      \"properties\": {\\n        \"dtype\":
\"string\\\",\\n        \"num_unique_values\": 10,\\n        \"samples\":
[\\n          \"Serralles Usa LLC\\\",\\n          \"LOYAL DOG WINERY\\\",\\n
\"SEA HAGG DISTILLERY LLC\\\"\\n        ],\\n        \"semantic_type\":
\"\\\",\\n        \"description\": \"\\\"\\n      }\\n    },\\n    {\\n
\"column\": \"Avg_Lead_Time\\\",\\n      \"properties\": {\\n

```

```

\ "dtype\ ": \ "number\ ",\n          \ "std\ ": 0.6111410093696324,\n
\ "min\ ": 5.32,\n          \ "max\ ": 7.13,\n          \ "num_unique_values\ ":
10,\n          \ "samples\ ": [\n          7.12,\n          5.95,\n
7.02\n          ],\n          \ "semantic_type\ ": \ "\",\n
\ "description\ ": \ "\",\n          \ "column\ ":
\ "Median_Lead_Time\ ",\n          \ "properties\ ": {\n          \ "dtype\ ":
\ "number\ ",\n          \ "std\ ": 0.5400617248673217,\n          \ "min\ ":
5.5,\n          \ "max\ ": 7.0,\n          \ "num_unique_values\ ": 3,\n
\ "samples\ ": [\n          6.0,\n          5.5,\n          7.0\n
],\n          \ "semantic_type\ ": \ "\",\n          \ "description\ ": \ "\",\n
}\n          },\n          {\n          \ "column\ ": \ "Total_POs\ ",\n
\ "properties\ ": {\n          \ "dtype\ ": \ "number\ ",\n          \ "std\ ":
10294,\n          \ "min\ ": 20,\n          \ "max\ ": 33271,\n
\ "num_unique_values\ ": 10,\n          \ "samples\ ": [\n          1509,\n
20,\n          1183\n          ],\n          \ "semantic_type\ ": \ "\",\n
\ "description\ ": \ "\",\n          \ "column\ ":
\ "CV\ ",\n          \ "properties\ ": {\n          \ "dtype\ ": \ "number\ ",\n
\ "std\ ": 0.05087020520675907,\n          \ "min\ ": 0.28,\n
\ "max\ ": 0.41,\n          \ "num_unique_values\ ": 8,\n
\ "samples\ ": [\n          0.4,\n          0.33,\n          0.41\n
],\n          \ "semantic_type\ ": \ "\",\n          \ "description\ ": \ "\",\n
}\n          }\n          ]\n          }","type":"dataframe"}

```

□ TOP 10 SLOWEST VENDORS:

```

{"summary":{"\n \ "name\ ": \ "print(\\\\"=\\\\"*70)\",\n \ "rows\ ": 10,\n
\ "fields\ ": [\n          {\n          \ "column\ ": \ "VendorNumber\ ",\n
\ "properties\ ": {\n          \ "dtype\ ": \ "number\ ",\n          \ "std\ ":
58943,\n          \ "min\ ": 2,\n          \ "max\ ": 173357,\n
\ "num_unique_values\ ": 10,\n          \ "samples\ ": [\n          173357,\n
n          90059,\n          9165\n          ],\n
\ "semantic_type\ ": \ "\",\n          \ "description\ ": \ "\",\n
n          },\n          {\n          \ "column\ ": \ "Vendor_Name\ ",\n
\ "properties\ ": {\n          \ "dtype\ ": \ "string\ ",\n
\ "num_unique_values\ ": 10,\n          \ "samples\ ": [\n
\ "TAMWORTH DISTILLING\ ",\n          \ "BLACK COVE BEVERAGES\ ",\n
\ "ULTRA BEVERAGE COMPANY LLP\ "\n          ],\n          \ "semantic_type\ ":
\ "\",\n          \ "description\ ": \ "\",\n
n          },\n          {\n          \ "column\ ": \ "Avg_Lead_Time\ ",\n
\ "properties\ ": {\n          \ "dtype\ ": \ "number\ ",\n          \ "std\ ": 0.4699893615817274,\n
\ "min\ ": 8.24,\n          \ "max\ ": 9.62,\n          \ "num_unique_values\ ":
9,\n          \ "samples\ ": [\n          8.31,\n          9.32,\n
8.41\n          ],\n          \ "semantic_type\ ": \ "\",\n
\ "description\ ": \ "\",\n          \ "column\ ":
\ "Max_Lead_Time\ ",\n          \ "properties\ ": {\n          \ "dtype\ ":
\ "number\ ",\n          \ "std\ ": 0,\n          \ "min\ ": 11,\n
\ "max\ ": 14,\n          \ "num_unique_values\ ": 4,\n          \ "samples\ ":
[\n          13,\n          14,\n          11\n          ],\n
\ "semantic_type\ ": \ "\",\n          \ "description\ ": \ "\",\n
}\n          }\n          ]\n          }

```

```

n    },\n    {\n        \"column\": \"Total_POs\", \n        \"properties\": {\n            \"dtype\": \"number\", \n            \"std\": 26557, \n            \"min\": 13, \n            \"max\": 84034, \n            \"num_unique_values\": 9, \n            \"samples\": [\n                153, \n                44, \n                84034\n            ], \n            \"semantic_type\": \"\", \n            \"description\": \"\"\n        }, \n        {\n            \"column\": \"Total_Spend\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 4173447.9740632963, \n                \"min\": 2452.29, \n                \"max\": 13210613.93, \n                \"num_unique_values\": 10, \n                \"samples\": [\n                    41036.44, \n                    14465.06, \n                    13210613.93\n                ], \n                \"semantic_type\": \"\", \n                \"description\": \"\"\n            }\n        }\n    ], \n    \"type\": \"dataframe\"}

```

=====

Intelligent Risk Classification

Classify vendors using median-based thresholds (data-driven)

```

print("\n LEAD TIME ANALYSIS - Step 3: Vendor Risk Classification\n")
print("="*70)

# Use median thresholds for data-driven classification
AVG_THRESHOLD = vendor_stats['Avg_Lead_Time'].median()
STD_THRESHOLD = vendor_stats['Std_Lead_Time'].median()

print(f" Classification Thresholds (Median-based):")
print(f"     Average Lead Time Threshold: {AVG_THRESHOLD:.1f} days")
print(f"     Variability Threshold: {STD_THRESHOLD:.1f} days")

def classify_vendor(row):
    """Classify based on both speed and consistency"""
    if row['Avg_Lead_Time'] <= AVG_THRESHOLD and row['Std_Lead_Time']
    <= STD_THRESHOLD:
        return ' Premium (Fast & Reliable)'
    elif row['Avg_Lead_Time'] > AVG_THRESHOLD and row['Std_Lead_Time']
    > STD_THRESHOLD:
        return ' High Risk (Slow & Unpredictable)'
    elif row['Avg_Lead_Time'] <= AVG_THRESHOLD and
    row['Std_Lead_Time'] > STD_THRESHOLD:
        return ' Fast but Variable'
    else:
        return ' Slow but Steady'

vendor_stats['Vendor_Risk'] = vendor_stats.apply(classify_vendor,
axis=1)

```

```
# Now safe to display with Vendor_Risk
display_cols = ['VendorNumber', 'Vendor_Name', 'Avg_Lead_Time',
'Total_POs', 'Vendor_Risk']
print(f"\n Sample classified vendors:")
display(vendor_stats[display_cols].head(10))

# Summary table
risk_summary =
vendor_stats['Vendor_Risk'].value_counts().reset_index()
risk_summary.columns = ['Risk_Category', 'Vendor_Count']
risk_summary['Percentage'] = (risk_summary['Vendor_Count'] /
risk_summary['Vendor_Count'].sum() * 100).round(1)

print(f"\n VENDOR RISK DISTRIBUTION:\n")
display(risk_summary)
print("="*70)
```

LEAD TIME ANALYSIS - Step 3: Vendor Risk Classification

Classification Thresholds (Median-based):

Average Lead Time Threshold: 7.7 days

Variability Threshold: 2.2 days

Sample classified vendors:

```
{
  "summary": {
    "name": "print(\\\"\\\"\\\"\\\"*70\\\"\\\",\\n \\\"rows\\\": 10,\\n \\\"fields\\\": [\\n \\\"column\\\": \\\"VendorNumber\\\",\\n \\\"properties\\\": {\\n \\\"dtype\\\": \\\"number\\\",\\n \\\"std\\\": 313,\\n \\\"min\\\": 2,\\n \\\"max\\\": 1003,\\n \\\"num_unique_values\\\": 10,\\n \\\"samples\\\": [\\n 660,\\n 60,\\n 480\\n ],\\n \\\"semantic_type\\\": \\\"\\\",\\n \\\"description\\\": \\\"\\\"\\n }\\n },\\n {\\n \\\"column\\\": \\\"Vendor_Name\\\",\\n \\\"properties\\\": {\\n \\\"dtype\\\": \\\"string\\\",\\n \\\"num_unique_values\\\": 10,\\n \\\"samples\\\": [\\n \\\"SAZERAC NORTH AMERICA INC.\\\",\\n \\\"ADAMB IMPORTS INTL INC\\\",\\n \\\"BACARDI USA INC\\\"\\n ],\\n \\\"semantic_type\\\": \\\"\\\",\\n \\\"description\\\": \\\"\\\"\\n }\\n },\\n {\\n \\\"column\\\": \\\"Avg_Lead_Time\\\",\\n \\\"properties\\\": {\\n \\\"dtype\\\": \\\"number\\\",\\n \\\"std\\\": 0.6485162552575942,\\n \\\"min\\\": 7.44,\\n \\\"max\\\": 9.62,\\n \\\"num_unique_values\\\": 10,\\n \\\"samples\\\": [\\n 7.51,\\n 7.61,\\n 7.7\\n ],\\n \\\"semantic_type\\\": \\\"\\\",\\n \\\"description\\\": \\\"\\\"\\n }\\n },\\n {\\n \\\"column\\\": \\\"Total_POs\\\",\\n \\\"properties\\\": {\\n \\\"dtype\\\": \\\"number\\\",\\n \\\"std\\\": 91846,\\n \\\"min\\\": 13,\\n \\\"max\\\": 91846,\\n \\\"num_unique_values\\\": 10,\\n \\\"samples\\\": [\\n 54331,\\n 626,\\n 91846\\n ],\\n \\\"semantic_type\\\": \\\"\\\",\\n \\\"description\\\": \\\"\\\"\\n }\\n }
}
```

```
}\\n    },\\n    {\\n        \\\"column\\\": \\\"Vendor_Risk\\\",\\n
\\\"properties\\\": {\\n            \\\"dtype\\\": \\\"category\\\",\\n
\\\"num_unique_values\\\": 4,\\n            \\\"samples\\\": [\\n                \\\"\\u2b50
Premium (Fast & Reliable)\\\",\\n                \\\"\\ud83d\\udd34 High Risk
(Slow & Unpredictable)\\\",\\n                \\\"\\ud83d\\udfe2 Slow but
Steady\\\"\\n            ],\\n            \\\"semantic_type\\\": \\\"\\\",\\n
\\\"description\\\": \\\"\\\"\\n        }\\n    }\\n ]\\n}\\\", \"type\": \"dataframe\"}
```

□ VENDOR RISK DISTRIBUTION:

```
{\"summary\":{\"\\n    \\\"name\\\": \\\"risk_summary\\\",\\n    \\\"rows\\\": 4,\\n
\\\"fields\\\": [\\n        {\\n            \\\"column\\\": \\\"Risk_Category\\\",\\n
\\\"properties\\\": {\\n                \\\"dtype\\\": \\\"string\\\",\\n
\\\"num_unique_values\\\": 4,\\n                \\\"samples\\\": [\\n                    \\\"\\
ud83d\\udfe1 Fast but Variable\\\",\\n                    \\\"\\ud83d\\udd34 High
Risk (Slow & Unpredictable)\\\",\\n                    \\\"\\ud83d\\udfe2 Slow but
Steady\\\"\\n                ],\\n                \\\"semantic_type\\\": \\\"\\\",\\n
\\\"description\\\": \\\"\\\"\\n            }\\n        },\\n        {\\n            \\\"column\\\":
\\\"Vendor_Count\\\",\\n            \\\"properties\\\": {\\n                \\\"dtype\\\":
\\\"number\\\",\\n                \\\"std\\\": 5,\\n                \\\"min\\\": 25,\\n
\\\"max\\\": 35,\\n                \\\"num_unique_values\\\": 4,\\n                \\\"samples\\\":
[\\n                    34,\\n                    25,\\n                    35\\n                ],\\n
\\\"semantic_type\\\": \\\"\\\",\\n                \\\"description\\\": \\\"\\\"\\n            }\\
n        },\\n        {\\n            \\\"column\\\": \\\"Percentage\\\",\\n
\\\"properties\\\": {\\n                \\\"dtype\\\": \\\"number\\\",\\n                \\\"std\\\":
4.361192497471305,\\n                \\\"min\\\": 20.8,\\n                \\\"max\\\": 29.2,\\n
\\\"num_unique_values\\\": 4,\\n                \\\"samples\\\": [\\n                    28.3,\\n
20.8,\\n                    29.2\\n                ],\\n                \\\"semantic_type\\\": \\\"\\\",\\n
\\\"description\\\": \\\"\\\"\\n            }\\n        }\\n    ]\\
n}\\\", \"type\": \"dataframe\", \"variable_name\": \"risk_summary\"}
```

Advanced Visualizations

4-panel comprehensive dashboard

```
print("\\n□ LEAD TIME ANALYSIS - Step 4: Visualization Dashboard\\n")
print("="*70)
print("Generating 4-panel visualization...\\n")

fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=('Vendor Risk Scatter (Size = Spend)',
                    'Lead Time Distribution',
                    'Financial Impact by Risk Tier',
```

```

        'Top 15 Vendors Lead Time Spread'),
    specs=[[{'type': 'scatter'}, {'type': 'histogram'}],
           [{'type': 'bar'}, {'type': 'box'}]]
)

# 1. Scatter plot: Speed vs Reliability (ChatGPT style + my size
# feature)
colors = {'Premium (Fast & Reliable)': '#2ecc71',
          'Slow but Steady': '#3498db',
          'Fast but Variable': '#f1c40f',
          'High Risk (Slow & Unpredictable)': '#e74c3c'}

for risk_class in vendor_stats['Vendor_Risk'].unique():
    mask = vendor_stats['Vendor_Risk'] == risk_class
    subset = vendor_stats[mask]

    fig.add_trace(
        go.Scatter(
            x=subset['Avg_Lead_Time'],
            y=subset['Std_Lead_Time'],
            mode='markers',
            name=risk_class,
            marker=dict(
                size=np.log(subset['Total_Spend'] + 1) * 3, # Size by
                color=colors.get(risk_class, 'gray'),
                opacity=0.7,
                line=dict(width=1, color='black')
            ),
            text=subset['VendorNumber'],
            hovertemplate='<b>Vendor:</b> %{text}<br>' +
                          '<b>Avg Lead:</b> %{x:.1f} days<br>' +
                          '<b>Std Dev:</b> %{y:.1f} days<br>' +
                          '<b>Spend:</b> $%
{marker.size:.0f}<extra></extra>'
        ),
        row=1, col=1
    )

# 2. Histogram: Lead time distribution (My contribution)
fig.add_trace(
    go.Histogram(
        x=valid_purchases['LeadTime_Days'],
        nbinsx=30,
        marker_color='steelblue',
        opacity=0.7,
        name='Lead Time Dist'
    ),
    row=1, col=2
)

```

```

# 3. Financial Impact by Risk Tier (My contribution)
financial_impact = vendor_stats.groupby('Vendor_Risk').agg({
    'Total_Spend': 'sum',
    'Total_POs': 'sum'
}).reset_index()

fig.add_trace(
    go.Bar(
        x=financial_impact['Vendor_Risk'],
        y=financial_impact['Total_Spend'],
        marker_color=[colors.get(x, 'gray') for x in
financial_impact['Vendor_Risk']],
        name='Total Spend',
        text=financial_impact['Total_Spend'].apply(lambda x: f'$
{x:,.0f}'),
        textposition='auto'
    ),
    row=2, col=1
)

# 4. Box plots for top 15 vendors by spend (My contribution)
top_15_vendors = vendor_stats.nlargest(15, 'Total_Spend')
['VendorNumber'].tolist()

for i, vendor in enumerate(top_15_vendors[:5]): # Show top 5 for
clarity
    vendor_data = valid_purchases[valid_purchases['VendorNumber'] ==
vendor]['LeadTime_Days']
    vendor_spend = vendor_stats[vendor_stats['VendorNumber'] ==
vendor]['Total_Spend'].iloc[0]

    fig.add_trace(
        go.Box(
            y=vendor_data,
            name=f"Vendor {vendor}",
            boxpoints='outliers',
            hovertemplate=f"<b>Vendor {vendor}</b><br>Spend: $
{vendor_spend:,.0f}<br>Days: %{y}<extra></extra>"
        ),
        row=2, col=2
    )

fig.update_layout(
    height=900,
    title_text="☐ Lead Time Analysis Dashboard - Vendor Performance
Overview",
    showlegend=True,
    hovermode='closest'
)

```

```

fig.update_xaxes(title_text="Average Lead Time (Days)", row=1, col=1)
fig.update_yaxes(title_text="Variability (Std Dev)", row=1, col=1)
fig.update_xaxes(title_text="Lead Time (Days)", row=1, col=2)
fig.update_yaxes(title_text="Frequency", row=1, col=2)
fig.update_xaxes(title_text="Vendor Risk Category", row=2, col=1)
fig.update_yaxes(title_text="Total Spend ($)", row=2, col=1)
fig.update_yaxes(title_text="Lead Time (Days)", row=2, col=2)

fig.show()
print("📊 Dashboard generated!")
print("="*70)

```

📊 LEAD TIME ANALYSIS - Step 4: Visualization Dashboard

```

=====
Generating 4-panel visualization...

```

```

📊 Dashboard generated!
=====

```

A-Class Product Supplier Review

Critical integration with ABC Analysis

```

print("\n📊 LEAD TIME ANALYSIS - Step 5: A-Class Product Supplier
Review + Risk Analysis\n")
print("="*70)

# Top brands from ABC Analysis
top_brands = [4261, 3545, 1233, 8068, 3405]
brand_names = {
    4261: 'Capt Morgan Spiced Rum',
    3545: 'Ketel One Vodka',
    1233: 'Jack Daniels No 7 Black',
    8068: 'Absolut 80 Proof',
    3405: 'Tito's Handmade Vodka'
}

aclass_purchases =
valid_purchases[valid_purchases['Brand'].isin(top_brands)]

if len(aclass_purchases) > 0:
    aclass_vendor_stats = aclass_purchases.groupby(
        ['Brand', 'VendorNumber']
    ).agg(

```

```

        Avg_Lead_Time=('LeadTime_Days', 'mean'),
        Std_Lead_Time=('LeadTime_Days', 'std'),
        Total_POs=('LeadTime_Days', 'count'),
        Avg_Cost=('Dollars', 'mean')
    ).reset_index()

    aclass_vendor_stats['Std_Lead_Time'] =
aclass_vendor_stats['Std_Lead_Time'].fillna(0).round(2)
    aclass_vendor_stats['Avg_Lead_Time'] =
aclass_vendor_stats['Avg_Lead_Time'].round(2)

    # Add product names
    aclass_vendor_stats['Product'] =
aclass_vendor_stats['Brand'].map(brand_names)

    # Add risk classification from previous step
    aclass_vendor_stats = aclass_vendor_stats.merge(
        vendor_stats[['VendorNumber', 'Vendor_Risk', 'Vendor_Name']],
        on='VendorNumber',
        how='left'
    )

    print("\n A-CLASS PRODUCT SUPPLIER PERFORMANCE:\n")
    display_cols = ['Product', 'VendorNumber', 'Vendor_Name',
'Avg_Lead_Time', 'Std_Lead_Time', 'Vendor_Risk', 'Total_POs']
    display(aclass_vendor_stats[display_cols].sort_values(['Product',
'Avg_Lead_Time']))

    # Critical risk alert
    high_risk_acl =
aclass_vendor_stats[aclass_vendor_stats['Vendor_Risk'].str.contains('H
igh Risk', na=False)]
    if len(high_risk_acl) > 0:
        print("\n CRITICAL ALERT: High-risk vendors supplying A-Class
products!")
        display(high_risk_acl[['Product', 'VendorNumber',
'Vendor_Name', 'Avg_Lead_Time', 'Vendor_Risk']])

    # 5.1: Find Premium alternatives
    print("\n FINDING PREMIUM VENDOR ALTERNATIVES FOR A-CLASS
PRODUCTS\n")
    premium_vendors =
vendor_stats[vendor_stats['Vendor_Risk'].str.contains('Premium',
na=False)]

    for brand_id, brand_name in brand_names.items():
        current_info =
aclass_vendor_stats[aclass_vendor_stats['Brand'] == brand_id]
        if len(current_info) == 0: continue

```

```

current_vendor = current_info['VendorNumber'].iloc[0]
current_risk = current_info['Vendor_Risk'].iloc[0]

brand_purchases = valid_purchases[valid_purchases['Brand'] ==
brand_id]
brand_vendors = brand_purchases.groupby('VendorNumber').agg({
    'LeadTime_Days': 'mean',
    'Dollars': 'sum'
}).reset_index()

brand_vendors = brand_vendors.merge(
    vendor_stats[['VendorNumber', 'Vendor_Risk']],
    on='VendorNumber'
)

premium_alternatives =
brand_vendors[brand_vendors['Vendor_Risk'].str.contains('Premium',
na=False)]

if len(premium_alternatives) > 0 and 'Premium' not in
str(current_risk):
    print(f"△ {brand_name}:")
    print(f"    Current: Vendor {current_vendor}
({current_risk})")
    print(f"    Alternative Premium vendors available:")
    for _, alt in premium_alternatives.iterrows():
        print(f"        → Vendor {alt['VendorNumber']}:
{alt['LeadTime_Days']:.1f} days, ${alt['Dollars']:.0f} volume")
    elif 'Premium' not in str(current_risk):
        print(f"□ {brand_name}: NO PREMIUM ALTERNATIVES FOUND -
Dual-source immediately!")

# 5.2: Concentration risk
print("\n△ VENDOR CONCENTRATION RISK ANALYSIS\n")
vendor_aiclass_count =
aclass_vendor_stats['VendorNumber'].value_counts()
multi_product_vendors = vendor_aiclass_count[vendor_aiclass_count >
1]

if len(multi_product_vendors) > 0:
    print("□ Vendors supplying MULTIPLE A-Class products
(Concentration Risk):")
    for vendor, count in multi_product_vendors.items():
        products =
aclass_vendor_stats[aclass_vendor_stats['VendorNumber'] == vendor]
['Product'].tolist()
        print(f"    • Vendor {vendor}: Supplies {count} products
({'', '.join(products)}")
        print(f"    Risk: If this vendor fails, you lose
{count}/5 top products\n")

```

```

else:
    print(" No purchase records found for A-Class products in this
period")

print("="*70)

```

□ LEAD TIME ANALYSIS - Step 5: A-Class Product Supplier Review + Risk Analysis

□ A-CLASS PRODUCT SUPPLIER PERFORMANCE:

```

{"summary":{"\n  \"name\": \"print(\\\"\\\"=\\\"\\\"*70)\\\",\\n  \"rows\": 5,\\n
\"fields\": [\n    {\n      \"column\": \"Product\\\",\\n
\"properties\": {\n        \"dtype\": \"string\\\",\\n
\"num_unique_values\": 5,\\n        \"samples\": [\n          \"Capt
Morgan Spiced Rum\\\",\\n          \"Tito's Handmade Vodka\\\",\\n
\"Jack Daniels No 7 Black\\\"\\n          ],\\n        \"semantic_type\":
\"\\\",\\n        \"description\": \"\\\"\\\"\\n        }\\n      },\\n    {\n
\"column\": \"VendorNumber\\\",\\n    \"properties\": {\n
\"dtype\": \"number\\\",\\n    \"std\": 6250,\\n    \"min\":
1128,\\n    \"max\": 17035,\\n    \"num_unique_values\": 4,\\n
\"samples\": [\n      3960,\\n      4425,\\n      17035\\n
],\\n    \"semantic_type\": \"\\\",\\n    \"description\": \"\\\"\\\"\\n
} \\n    },\\n    {\n      \"column\": \"Vendor_Name\\\",\\n
\"properties\": {\n        \"dtype\": \"string\\\",\\n
\"num_unique_values\": 4,\\n        \"samples\": [\n          \"DIAGEO
NORTH AMERICA INC\\\",\\n          \"MARTIGNETTI COMPANIES\\\",\\n
\"PERNOD RICARD USA\\\"\\n          ],\\n        \"semantic_type\": \"\\\",\\n
\"description\": \"\\\"\\\"\\n        } \\n    },\\n    {\n      \"column\":
\"Avg_Lead_Time\\\",\\n    \"properties\": {\n      \"dtype\":
\"number\\\",\\n      \"std\": 0.08786353054595505,\\n      \"min\":
7.34,\\n      \"max\": 7.56,\\n      \"num_unique_values\": 5,\\n
\"samples\": [\n        7.34,\\n        7.56,\\n        7.44\\n
],\\n      \"semantic_type\": \"\\\",\\n      \"description\": \"\\\"\\\"\\n
} \\n    },\\n    {\n      \"column\": \"Std_Lead_Time\\\",\\n
\"properties\": {\n        \"dtype\": \"number\\\",\\n        \"std\":
0.02302172886644268,\\n        \"min\": 2.16,\\n        \"max\": 2.22,\\n
\"num_unique_values\": 5,\\n        \"samples\": [\n          2.21,\\n
2.16,\\n          2.22\\n          ],\\n        \"semantic_type\": \"\\\",\\n
\"description\": \"\\\"\\\"\\n        } \\n    },\\n    {\n      \"column\":
\"Vendor_Risk\\\",\\n    \"properties\": {\n      \"dtype\":
\"category\\\",\\n      \"num_unique_values\": 2,\\n      \"samples\":
[\n        \"\\\"ud83d\\udfe2 Slow but Steady\\\",\\n        \"\\\"
ud83d\\udfe1 Fast but Variable\\\"\\n        ],\\n      \"semantic_type\":
\"\\\",\\n      \"description\": \"\\\"\\\"\\n      } \\n    },\\n    {\n
      \"column\": \"Total_POs\\\",\\n
\"properties\": {\n        \"dtype\": \"number\\\",\\n        \"std\":

```

```
396,\n          \"min\": 6326,\n          \"max\": 7359,\n          \"num_unique_values\": 5,\n          \"samples\": [\n          6774,\n          6506\n        ],\n          \"semantic_type\": \"\",\n          \"description\": \"\"\n      }\n      }\n      ]\n    }\", \"type\": \"dataframe\"}
```

□ FINDING PREMIUM VENDOR ALTERNATIVES FOR A-CLASS PRODUCTS

□ Capt Morgan Spiced Rum: NO PREMIUM ALTERNATIVES FOUND - Dual-source immediately!

□ Ketel One Vodka: NO PREMIUM ALTERNATIVES FOUND - Dual-source immediately!

□ Jack Daniels No 7 Black: NO PREMIUM ALTERNATIVES FOUND - Dual-source immediately!

□ Absolut 80 Proof: NO PREMIUM ALTERNATIVES FOUND - Dual-source immediately!

□ Tito's Handmade Vodka: NO PREMIUM ALTERNATIVES FOUND - Dual-source immediately!

△ VENDOR CONCENTRATION RISK ANALYSIS

□ Vendors supplying MULTIPLE A-Class products (Concentration Risk):

- Vendor 3960: Supplies 2 products (Ketel One Vodka, Capt Morgan Spiced Rum)

Risk: If this vendor fails, you lose 2/5 top products

=====

Financial Impact Analysis

Quantify procurement risk by dollar exposure

```
print("\n□ LEAD TIME ANALYSIS - Step 6: Financial Impact by Risk Tier\n")
print("="*70)
```

```
financial_summary = vendor_stats.groupby('Vendor_Risk').agg({
    'Total_Spend': ['sum', 'mean', 'count'],
    'Total_POs': 'sum',
    'Avg_Lead_Time': 'mean',
    'CV': 'mean'
}).round(2)
```

```
financial_summary.columns = ['Total_Spend', 'Avg_Spend_Per_Vendor',
    'Vendor_Count', 'Total_POs', 'Avg_Lead_Days', 'Avg_CV']
financial_summary['Spend_Percentage'] =
    (financial_summary['Total_Spend'] /
    financial_summary['Total_Spend'].sum() * 100).round(1)
```

```

# Reorder for display hierarchy
tier_order = ['Premium (Fast & Reliable)', 'Slow but Steady', 'Fast but Variable', 'High Risk (Slow & Unpredictable)']
financial_summary = financial_summary.reindex([x for x in tier_order if x in financial_summary.index])

print("\n FINANCIAL EXPOSURE BY SUPPLIER TIER:\n")
display(financial_summary)

# Calculate risk exposure
total_spend = financial_summary['Total_Spend'].sum()
risky_spend = financial_summary.loc['High Risk (Slow & Unpredictable)', 'Total_Spend'] if 'High Risk (Slow & Unpredictable)' in financial_summary.index else 0
variable_spend = financial_summary.loc['Fast but Variable', 'Total_Spend'] if 'Fast but Variable' in financial_summary.index else 0

HIGH_SPEND_THRESHOLD = vendor_stats['Total_Spend'].quantile(0.90) # Top 10% spenders

high_spend_slow = vendor_stats[
    (vendor_stats['Total_Spend'] > HIGH_SPEND_THRESHOLD) &
    (vendor_stats['Avg_Lead_Time'] > AVG_THRESHOLD)
]

if len(high_spend_slow) > 0:
    print(f"\n CRITICAL: HIGH-SPEND SLOW VENDORS (Top 10% spenders slower than median)")
    print(f"    These vendors cost you speed AND money:\n")
    display(high_spend_slow[['VendorNumber', 'Vendor_Name', 'Total_Spend', 'Total_P0s', 'Avg_Lead_Time', 'Vendor_Risk']].sort_values('Total_Spend', ascending=False))

    total_at_risk = high_spend_slow['Total_Spend'].sum()
    print(f"\n Total spend with slow high-volume vendors: ${total_at_risk:,.2f}")
    print(f"    Action: Renegotiate SLAs with penalty clauses for late delivery")

print(f"\n KEY METRICS:")
print(f"    • Total Procurement Exposure: ${total_spend:,.2f}")
print(f"    • High-Risk Vendor Exposure: ${risky_spend:,.2f} ({risky_spend/total_spend*100:.1f}%)")
print(f"    • Variable Vendor Exposure: ${variable_spend:,.2f} ({variable_spend/total_spend*100:.1f}%)")
print(f"    • Reliable Vendor Coverage: {(total_spend-risky_spend-variable_spend)/total_spend*100:.1f}%")

```

```
print("="*70)
```

□ LEAD TIME ANALYSIS - Step 6: Financial Impact by Risk Tier

□ FINANCIAL EXPOSURE BY SUPPLIER TIER:

```
{
  "summary": {
    "name": "financial_summary",
    "rows": 4,
    "fields": [
      {
        "column": "Vendor_Risk",
        "properties": {
          "dtype": "string",
          "num_unique_values": 4,
          "samples": [
            "\ud83d\udfe2 Slow but Steady",
            "\ud83d\udd34 High Risk (Slow & Unpredictable)",
            "\u2b50 Premium (Fast & Reliable)"
          ],
          "semantic_type": "",
          "description": ""
        },
        "column": "Total_Spend",
        "properties": {
          "dtype": "number",
          "std": 61331963.961250804,
          "min": 23467173.12,
          "max": 164945956.03,
          "num_unique_values": 4,
          "samples": [
            82905291.92,
            23467173.12,
            50580379.19,
            1945399.2
          ],
          "semantic_type": "",
          "description": ""
        },
        "column": "Avg_Spend_Per_Vendor",
        "properties": {
          "dtype": "number",
          "std": 1662192.9855604942,
          "min": 938686.92,
          "max": 4851351.65,
          "num_unique_values": 4,
          "samples": [
            2368722.63,
            938686.92,
            1945399.2,
            1125973
          ],
          "semantic_type": "",
          "description": ""
        },
        "column": "Vendor_Count",
        "properties": {
          "dtype": "number",
          "std": 5,
          "min": 25,
          "max": 35,
          "num_unique_values": 4,
          "samples": [
            25,
            26,
            35,
            26
          ],
          "semantic_type": "",
          "description": ""
        },
        "column": "Total_POs",
        "properties": {
          "dtype": "number",
          "std": 418605,
          "min": 131803,
          "max": 1125973,
          "num_unique_values": 4,
          "samples": [
            672199,
            131803,
            442482,
            131803
          ],
          "semantic_type": "",
          "description": ""
        },
        "column": "Avg_Lead_Days",
        "properties": {
          "dtype": "number",
          "std": 0.3807886552931954,
          "min": 7.33,
          "max": 8.08,
          "num_unique_values": 4,
          "samples": [
            8.08,
            7.93,
            7.38,
            7.33
          ],
          "semantic_type": "",
          "description": ""
        },
        "column": "Avg_CV",
        "properties": {
          "dtype": "number",
          "std": 7.33,
          "min": 7.33,
          "max": 7.33,
          "num_unique_values": 4,
          "samples": [
            7.33,
            7.33,
            7.33,
            7.33
          ],
          "semantic_type": "",
          "description": ""
        }
      ]
    }
  }
}
```

```

0.02886751345948129,\n          \"min\": 0.25,\n          \"max\": 0.32,\n          \"num_unique_values\": 4,\n          \"samples\": [\n              0.25,\n              0.29,\n              0.28\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\",\n          \"column\": \"Spend_Percentage\",\n          \"properties\": {\n              \"dtype\": \"number\",\n              \"std\": 19.03382953235283,\n              \"min\": 7.3,\n              \"max\": 51.2,\n              \"num_unique_values\": 4,\n              \"samples\": [\n                  25.8,\n                  7.3,\n                  15.7\n              ],\n              \"semantic_type\": \"\",\n              \"description\": \"\"\n          }\n      ],\n      \"type\": \"dataframe\", \"variable_name\": \"financial_summary\"}

```

❑ CRITICAL: HIGH-SPEND SLOW VENDORS (Top 10% spenders slower than median)

These vendors cost you speed AND money:

```

{"summary": "{\n    \"name\": \"print(\\\"\\\"=\\\"\\\"*70)\\\", \"rows\": 5,\n    \"fields\": [\n        {\n            \"column\": \"VendorNumber\",\n            \"properties\": {\n                \"dtype\": \"number\",\n                \"std\": 4238,\n                \"min\": 480,\n                \"max\": 9552,\n                \"num_unique_values\": 5,\n                \"samples\": [\n                    480,\n                    9552,\n                    1392\n                ],\n                \"semantic_type\": \"\",\n                \"description\": \"\",\n                \"column\": \"Vendor_Name\",\n                \"properties\": {\n                    \"dtype\": \"string\",\n                    \"num_unique_values\": 5,\n                    \"samples\": [\n                        \"BACARDI USA INC\",\n                        \"M S WALKER INC\",\n                        \"CONSTELLATION BRANDS INC\"\n                    ],\n                    \"semantic_type\": \"\",\n                    \"description\": \"\"\n                },\n                \"column\": \"Total_Spend\",\n                \"properties\": {\n                    \"dtype\": \"number\",\n                    \"std\": 6548379.684539815,\n                    \"min\": 10935817.3,\n                    \"max\": 27861690.02,\n                    \"num_unique_values\": 5,\n                    \"samples\": [\n                        17624378.72,\n                        10935817.3,\n                        15573917.9\n                    ],\n                    \"semantic_type\": \"\",\n                    \"description\": \"\"\n                },\n                {\n                    \"column\": \"Total_POs\",\n                    \"properties\": {\n                        \"dtype\": \"number\",\n                        \"std\": 47805,\n                        \"min\": 84034,\n                        \"max\": 185574,\n                        \"num_unique_values\": 5,\n                        \"samples\": [\n                            91846,\n                            112792,\n                            185574\n                        ],\n                        \"semantic_type\": \"\",\n                        \"description\": \"\"\n                    },\n                    {\n                        \"column\": \"Avg_Lead_Time\",\n                        \"properties\": {\n                            \"dtype\": \"number\",\n                            \"std\": 0.2904307146291521,\n                            \"min\": 7.7,\n                            \"max\": 8.41,\n                            \"num_unique_values\": 5,\n                            \"samples\": [\n                                7.7,\n                                7.91,\n                                7.74\n                            ],\n                            \"semantic_type\": \"\",\n                            \"description\": \"\"\n                        },\n                        {\n                            \"column\": \"Vendor_Risk\",\n                            \"properties\": {\n                                \"dtype\": \"category\",\n                                \"num_unique_values\": 2,\n                                \"samples\": [\n                                    \"Low\",\n                                    \"High\"\n                                ]\n                            }\n                        }\n                    ]\n                }\n            }\n        }\n    ]\n}"

```

```
[\\n          \\\"\\ud83d\\udd34 High Risk (Slow & Unpredictable)\\\",\\n
\\\"\\ud83d\\udfe2 Slow but Steady\\\"\\n          ],\\n
\\\"semantic_type\\\": \\\"\\\",\\n          \\\"description\\\": \\\"\\\"\\n          }\\n
n      ]\\n}\\\", \"type\": \"dataframe\"}
```

□ Total spend with slow high-volume vendors: \$85,206,417.87
 □ Action: Renegotiate SLAs with penalty clauses for late delivery

□ KEY METRICS:

- Total Procurement Exposure: \$321,898,800.26
- High-Risk Vendor Exposure: \$23,467,173.12 (7.3%)
- Variable Vendor Exposure: \$164,945,956.03 (51.2%)
- Reliable Vendor Coverage: 41.5%

Strategic Recommendations

Actionable procurement strategy

```
print("\\n□ LEAD TIME ANALYSIS - Step 7: Strategic Recommendations\\n")
print("="*70)

# Identify specific problematic vendors
high_spend_slow = vendor_stats[
    (vendor_stats['Total_Spend'] >
 vendor_stats['Total_Spend'].quantile(0.75)) &
    (vendor_stats['Avg_Lead_Time'] > AVG_THRESHOLD)
]

high_variability_critical = vendor_stats[
    (vendor_stats['CV'] > 0.5) &
    (vendor_stats['Total_POs'] > 20)
]

print("□ STRATEGIC INSIGHTS:\\n")

print(11 SUPPLIER PORTFOLIO HEALTH:")
print(f"    •
{len(vendor_stats[vendor_stats['Vendor_Risk'].str.contains('Premium')])} Premium vendors
({len(vendor_stats[vendor_stats['Vendor_Risk'].str.contains('Premium')])}/len(vendor_stats)*100:.0f}%)")
print(f"    •
{len(vendor_stats[vendor_stats['Vendor_Risk'].str.contains('High Risk')])} High-risk vendors")
print(f"    • Average lead time:
{vendor_stats['Avg_Lead_Time'].mean():.1f} days")
```

```

if len(high_spend_slow) > 0:
    print(f"\n2 HIGH-PRIORITY NEGOTIATIONS:")
    print(f"    • {len(high_spend_slow)} high-spend vendors are slower than median")
    print(f"    • Combined spend: $ {high_spend_slow['Total_Spend'].sum():,.2f}")
    print(f"    • Action: Renegotiate SLAs or find alternative sources")

if len(high_variability_critical) > 0:
    print(f"\n3 CONSISTENCY ISSUES:")
    print(f"    • {len(high_variability_critical)} vendors show high variability (>50% CV) with significant order volume")
    print(f"    • Action: Implement vendor-managed inventory or increase safety stock")

print(f"\n4 A-CLASS PRODUCT STRATEGY:")
if 'aclass_vendor_stats' in locals() and len(aiclass_vendor_stats) > 0:
    # Check actual risk categories
    premium_count = len(aiclass_vendor_stats[aclass_vendor_stats['Vendor_Risk'].str.contains('Premium', na=False)])
    variable_count = len(aiclass_vendor_stats[aclass_vendor_stats['Vendor_Risk'].str.contains('Variable', na=False)])
    high_risk_count = len(aiclass_vendor_stats[aclass_vendor_stats['Vendor_Risk'].str.contains('High Risk', na=False)])
    slow_steady_count = len(aiclass_vendor_stats[aclass_vendor_stats['Vendor_Risk'].str.contains('Slow but Steady', na=False)])

    print(f"    • □ Premium Vendors: {premium_count}/5 products")
    print(f"    • □ Fast but Variable: {variable_count}/5 products ▲")
    print(f"    • □ Slow but Steady: {slow_steady_count}/5 products")
    print(f"    • □ High Risk: {high_risk_count}/5 products")

    if premium_count == 0:
        print(f"\n    □ CRITICAL GAP: No A-Class products use Premium vendors!")
        print(f"        All top revenue products are with variable or slow suppliers.")
        print(f"        Action: Immediate supplier development/negotiation required.")
        elif variable_count > 0:
            print(f"\n    ▲ WARNING: {variable_count} A-Class products use variable suppliers")
            print(f"        Risk of stockouts during demand spikes.")
    else:
        print(f"    • Review purchase data linkage for A-Class products")

```

```

    print("    • Review purchase data linkage for A-Class products")

print(f"\n5 IMMEDIATE ACTIONS:")
print(f"    • Consolidate {vendor_stats.nlargest(3, 'Total_POs')}
['Total_POs'].sum()} orders from top 3 vendors to negotiate volume
discounts")
print(f"    • Set safety stock multiplier to 1.5x for 'Fast but
Variable' vendors")
print(f"    • Negotiate penalty clauses with vendors having CV > 0.8")
print("="*70)

```

□ LEAD TIME ANALYSIS - Step 7: Strategic Recommendations

□ STRATEGIC INSIGHTS:

1 SUPPLIER PORTFOLIO HEALTH:

- 26 Premium vendors (22%)
- 25 High-risk vendors
- Average lead time: 7.7 days

2 HIGH-PRIORITY NEGOTIATIONS:

- 7 high-spend vendors are slower than median
- Combined spend: \$94,833,640.94
- Action: Renegotiate SLAs or find alternative sources

4 A-CLASS PRODUCT STRATEGY:

- □ Premium Vendors: 0/5 products
- □ Fast but Variable: 4/5 products ▲
- □ Slow but Steady: 1/5 products
- □ High Risk: 0/5 products

□ CRITICAL GAP: No A-Class products use Premium vendors!

All top revenue products are with variable or slow suppliers.
Action: Immediate supplier development/negotiation required.

5 IMMEDIATE ACTIONS:

- Consolidate 618732 orders from top 3 vendors to negotiate volume discounts
- Set safety stock multiplier to 1.5x for 'Fast but Variable' vendors
- Negotiate penalty clauses with vendors having CV > 0.8

Export Results

Save analysis for downstream use

```

print("\n LEAD TIME ANALYSIS - Step 8: Save Results\n")
print("="*70)

# Save complete vendor scorecard
vendor_file = '/content/Vendor_Performance_Scorecard.csv'
vendor_stats.to_csv(vendor_file, index=False)
print(f" Vendor Scorecard: {vendor_file}")

# Save A-Class vendor analysis
if 'aclass_vendor_stats' in locals() and len(aclass_vendor_stats) > 0:
    aclass_file = '/content/AClass_Vendor_Analysis.csv'
    aclass_vendor_stats.to_csv(aclass_file, index=False)
    print(f" A-Class Analysis: {aclass_file}")

# Save raw lead time data for audit
raw_file = '/content/LeadTime_Raw_Data.csv'
valid_purchases[['VendorNumber', 'Brand', 'LeadTime_Days', 'Dollars',
'Quantity', 'PODate', 'ReceivingDate']].to_csv(raw_file, index=False)
print(f" Raw Data: {raw_file}")

print("\n Files ready for procurement team review!")
print("="*70)
print("\n PHASE 2.4 COMPLETE - Lead Time Analysis Finished!")
print("="*70)

```

LEAD TIME ANALYSIS - Step 8: Save Results

```

=====
 Vendor Scorecard: /content/Vendor_Performance_Scorecard.csv
 A-Class Analysis: /content/AClass_Vendor_Analysis.csv
 Raw Data: /content/LeadTime_Raw_Data.csv

```

Files ready for procurement team review!

```

=====
 PHASE 2.4 COMPLETE - Lead Time Analysis Finished!
=====

```