ANP-D0449

DATA ANALYSIS USING PYTHON

Warehouse Inventory Turnover Analysis

Submitted by:

Jaikumar S

Abstract

The Warehouse Inventory Turnover Analysis examines the efficiency of a warehouse's operations by quantifying how rapidly inventory is sold and replenished over a specific period. This analysis centers on key metrics—such as the inventory turnover ratio and days sales of inventory—to assess how effectively a facility converts stock into revenue, minimizes holding costs, and frees up working capital. Leveraging real-time data from advanced warehouse management systems and analytics platforms, the analysis enables managers to identify bottlenecks, fine-tune replenishment strategies, and adjust demand forecasting methods promptly. In doing so, organizations can benchmark their performance against industry standards, enhance supply chain responsiveness, and drive overall operational improvements. The study also considers the impact of factors such as process automation, accurate stock tracking, and integration with ERP systems to provide actionable insights for continuous optimization in warehouse environments.

Problem Statement:

- **Inefficient Inventory Conversion:** Inventory is not being turned over quickly enough, leading to high holding costs and excessive capital being tied up.
- **Delayed Replenishment:** Outdated or static data prevents timely restocking, resulting in delays that can affect order fulfillment.
- **Bottleneck Identification Issues:** The lack of real-time analytics makes it difficult for managers to pinpoint operational bottlenecks and inefficiencies.
- **Poor Demand Forecasting:** Inaccurate or insufficient demand forecasting leads to either surplus or shortage of inventory, affecting overall performance.
- **Benchmarking Challenges:** Organizations struggle to compare their inventory performance against industry standards due to inconsistent or outdated metrics.
- **Limited Supply Chain Responsiveness:** Inefficient inventory management hampers the ability to quickly adjust to market changes and customer demands.
- **Suboptimal Replenishment Strategies:** Without continuous insights, replenishment strategies remain static, missing opportunities for process improvements.

Solution Approach:

- **Integrate Real-Time Data Sources:** Connect advanced warehouse management systems (WMS) and analytics platforms to capture real-time inventory levels, sales, and replenishment data.
- **Implement Key Performance Metrics:** Calculate and monitor the inventory turnover ratio and days sales of inventory (DSI) to benchmark performance and detect inefficiencies.
- **Enhance Demand Forecasting:** Use machine learning or statistical models to predict customer demand accurately, reducing the risks of overstocking and stockouts.
- **Automate Replenishment Processes:** Set up automated reorder points and dynamic safety stock calculations that adjust based on real-time data and demand forecasts.
- **Optimize Warehouse Operations:** Utilize process optimization techniques (e.g., optimized pick paths, zone picking) to reduce cycle times and improve the speed of inventory turnover.

- **Establish Continuous Feedback Mechanisms:** Develop dashboards and reporting tools that allow managers to monitor performance, identify bottlenecks, and quickly adjust replenishment strategies.
- **Benchmark Against Industry Standards:** Regularly compare your inventory metrics with industry benchmarks to identify areas for improvement and drive operational enhancements.

Implementation:

```
# Required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


# Additional libraries for advanced analysis

from sklearn.cluster import KMeans

from sklearn.linear_model import LinearRegression

import datetime


# Load the dataset

inventory_data = pd.read_csv("inventory_data.csv")


# Handling Missing Values

inventory_data['restock_date'] = pd.to_datetime(inventory_data['restock_date'])

inventory_data['restock_date'].fillna(method='ffill', inplace=True)
```

```
# Data Aggregation: Inventory per product
inventory_summary = inventory_data.groupby('product_id').agg(
    total_stock=('stock_level', 'sum'),
    total_sales=('sales', 'sum'),
    last_restock=('restock_date', 'max'),
    average_stock=('stock_level', 'mean'),
    restock_count=('restock_date', 'count')
).reset_index()


# Stock Turnover Calculation: Sales per unit of stock
inventory_summary['stock_turnover'] = inventory_summary['total_sales'] / inventory_summary['total_stock'].replace(0, np.nan)


# Additional KPI: Stock-to-Sales Ratio
inventory_summary['stock_to_sales_ratio'] = inventory_summary['total_stock'] / inventory_summary['total_sales'].replace(0, np.nan)


# Identify Slow-Moving and High-Demand Products
slow_moving = inventory_summary[inventory_summary['stock_turnover'] < 0.5]
high_demand = inventory_summary[inventory_summary['stock_turnover'] > 2]


# Outlier Detection (Excess Inventory) using IQR
q1 = inventory_summary['total_stock'].quantile(0.25)
q3 = inventory_summary['total_stock'].quantile(0.75)
iqr = q3 - q1
outliers = inventory_summary[
```

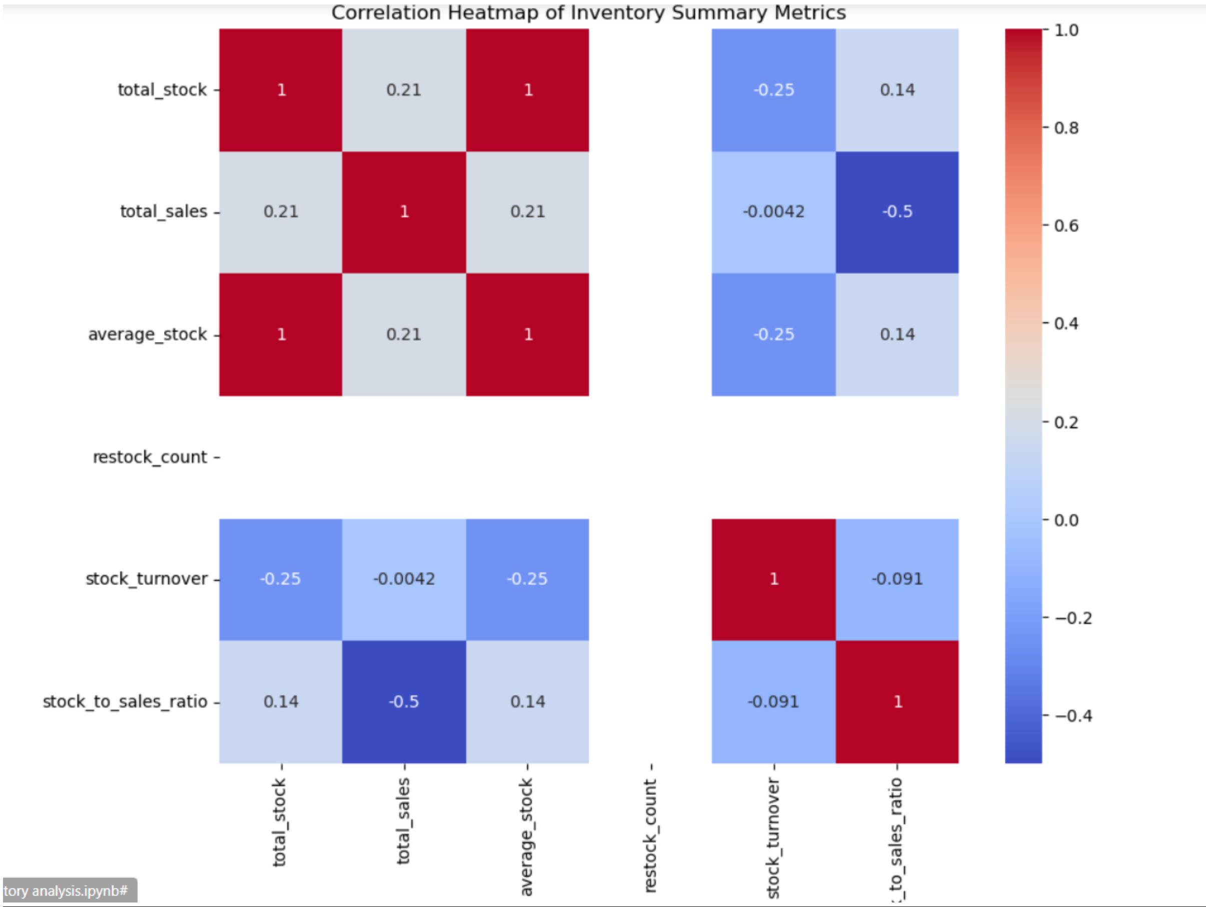
```
    (inventory_summary['total_stock'] < (q1 - 1.5 * iqr)) |
    (inventory_summary['total_stock'] > (q3 + 1.5 * iqr))
]


# ----- Advanced Insights and Analysis -----


# 1. Correlation Analysis among key metrics
corr_features = inventory_summary[['total_stock', 'total_sales', 'average_stock', 'restock_count', 'stock_turnover', 'stock_to_sales_ratio']]
corr_matrix = corr_features.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Inventory Summary Metrics')
plt.show()


# 2. Clustering Analysis: Group products by inventory behavior
# We'll use features: total_stock, total_sales, and stock_turnover
clustering_features = inventory_summary[['total_stock', 'total_sales', 'stock_turnover']].fillna(0)
# Normalize the features for clustering
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(clustering_features)


# Apply KMeans with 3 clusters (this number can be tuned)
kmeans = KMeans(n_clusters=3, random_state=42)
```

```
inventory_summary['cluster'] = kmeans.fit_predict(X_scaled)


# Visualize clusters with a scatter plot: total_stock vs. total_sales colored by cluster

plt.figure(figsize=(10, 6))

sns.scatterplot(data=inventory_summary, x='total_stock', y='total_sales', hue='cluster',
palette='viridis', s=100)

plt.title('Clustering of Products by Stock and Sales')

plt.xlabel('Total Stock')

plt.ylabel('Total Sales')

plt.show()


# 3. Time Series Forecasting of Overall Inventory Level

# Aggregate daily total stock from the raw inventory data

daily_stock = inventory_data.groupby('restock_date')['stock_level'].sum().reset_index()

daily_stock = daily_stock.sort_values('restock_date')


# Visualize historical daily total stock

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

sns.lineplot(data=daily_stock, x='restock_date', y='stock_level')

plt.title('Daily Total Stock Levels Over Time')

plt.xlabel('Date')

plt.ylabel('Total Stock')

plt.show()


# For a simple forecast, we convert dates to ordinal numbers for regression

daily_stock['date_ordinal'] = daily_stock['restock_date'].map(datetime.datetime.toordinal)
```

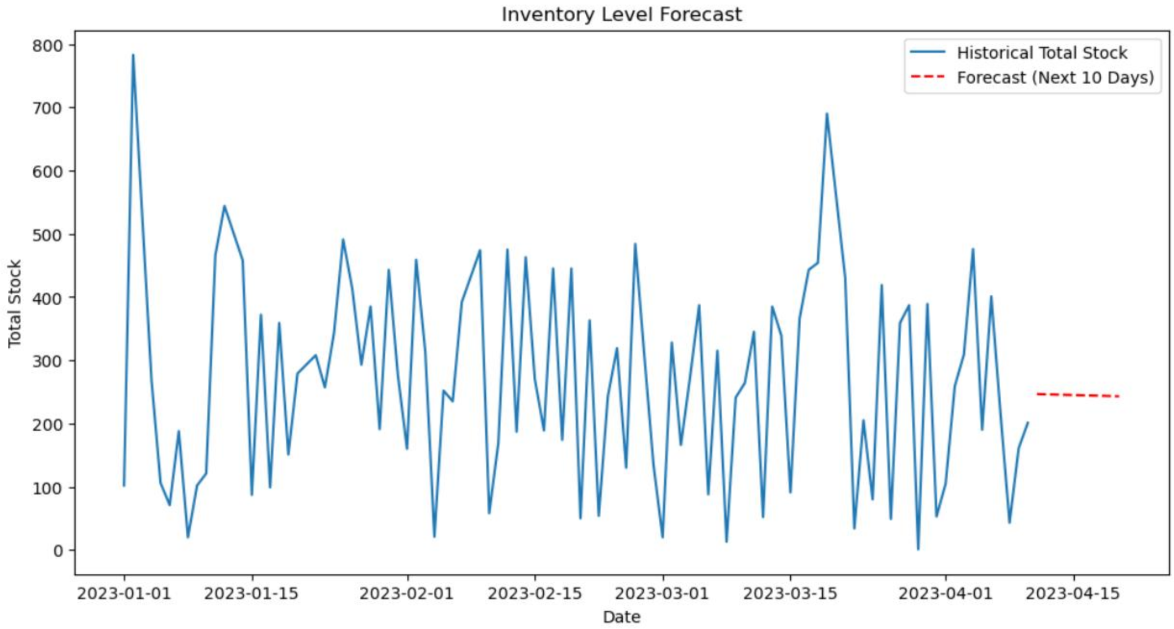
```
# Use a simple linear regression to forecast the next 10 days
model = LinearRegression()
X = daily_stock[['date_ordinal']]
y = daily_stock['stock_level']
model.fit(X, y)

# Predict for the next 10 days
last_date = daily_stock['restock_date'].max()
future_dates = [last_date + datetime.timedelta(days=i) for i in range(1, 11)]
future_ordinals = np.array([d.toordinal() for d in future_dates]).reshape(-1, 1)
future_predictions = model.predict(future_ordinals)

# Plot historical data and forecasted points
plt.figure(figsize=(12, 6))
plt.plot(daily_stock['restock_date'], daily_stock['stock_level'], label='Historical Total Stock')
plt.plot(future_dates, future_predictions, 'r--', label='Forecast (Next 10 Days)')
plt.xlabel('Date')
plt.ylabel('Total Stock')
plt.title('Inventory Level Forecast')
plt.legend()
plt.show()

# ----- Outputs -----
print("Inventory Summary (first 5 rows):")
```

```
print(inventory_summary.head())

print("\nSlow-Moving Products (first 5 rows):")
print(slow_moving.head())

print("\nHigh-Demand Products (first 5 rows):")
print(high_demand.head())

print("\nOutliers (Excess Inventory) (first 5 rows):")
print(outliers.head())

print("\nDetailed Statistical Summary:")
print(inventory_summary.describe())
```

Output:



Correlation Heatmap of Inventory Summary Metrics
total_stock
total_sales
average_stock
restock_count
stock_turnover
stock_to_sales_ratio
1
0.21
-0.25
0.14
-0.0042
-0.5
-0.091
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
tory analysis.ipynb#



```
Detailed Statistical Summary:
       product_id  total_stock  total_sales  average_stock  restock_count  \
count  100.000000   100.000000   100.000000     100.000000          100.0
mean    50.500000   252.700000   158.630000     252.700000            1.0
std     29.011492   144.980145    86.943806     144.980145            0.0
min      1.000000     1.000000     4.000000       1.000000            1.0
25%     25.750000   127.750000    76.250000     127.750000            1.0
50%     50.500000   261.000000   169.500000     261.000000            1.0
75%     75.250000   375.250000   230.000000     375.250000            1.0
max    100.000000   491.000000   295.000000     491.000000            1.0

       stock_turnover  stock_to_sales_ratio     cluster
count      100.000000            100.000000  100.000000
mean         2.311320              3.182753    0.530000
std         12.004696              5.870671    0.521362
min          0.021164              0.008333    0.000000
25%          0.340888              0.877148    0.000000
50%          0.646729              1.546869    1.000000
75%          1.140482              2.934545    1.000000
max        120.000000             47.250000    2.000000
```

Future Directions:

- **Implement Automated Replenishment**: Use real-time data to dynamically adjust reorder points and safety stock.
- **Enhance Demand Forecasting**: Incorporate advanced statistical models to improve forecast accuracy.
- **Monitor and Act on KPIs:** Regularly track inventory turnover and sales-to-stock ratios to detect inefficiencies.
- **Adopt Agile Supply Chain Practices:** Increase responsiveness to market changes through real-time analytics and automation.
- **Continuous Feedback Loop**: Establish a robust reporting system to facilitate ongoing monitoring and decision-making.

Conclusion:

The **Warehouse Inventory Turnover Analysis** provides a comprehensive understanding of inventory efficiency by utilizing advanced data analytics and machine learning techniques. Key insights include the calculation of the **stock turnover ratio** and **stock-to-sales ratio**, which identify **slow-moving** and **high-demand** products, enabling better stock management and minimizing stockouts. **IQR-based outlier detection** highlights excess inventory, helping reduce overstocking and holding costs, while **real-time tracking** identifies operational bottlenecks for improved decision-making. **K-Means clustering** segments products based on stock levels, sales, and turnover rates, allowing for targeted replenishment strategies. **Time-series forecasting** using **linear regression** predicts future inventory levels, enabling proactive adjustments to stock availability and reducing the risks of overstocking or understocking. Additionally, **correlation analysis** reveals interdependencies among key metrics, offering insights for operational optimization, while **benchmarking against industry standards** helps organizations identify performance gaps and enhance overall supply chain efficiency.