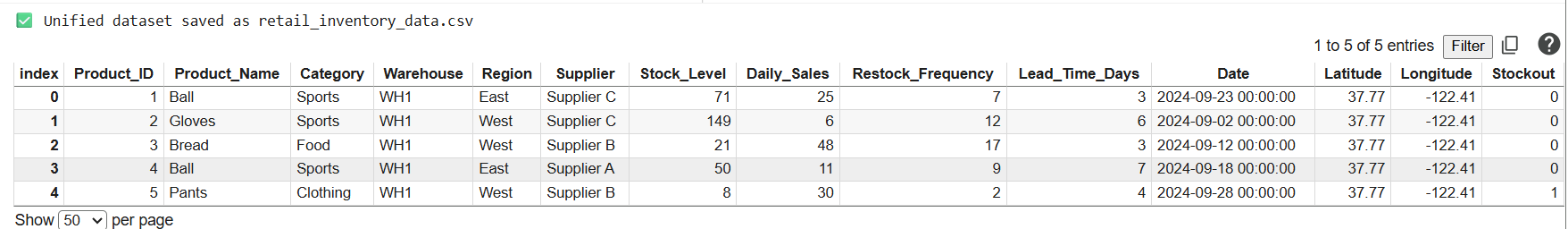
**Task 14 - Retail Inventory Management**

**Description:**

The **Retail Inventory Dataset** is a comprehensive simulated dataset created to analyze and visualize various aspects of inventory management in a retail environment. It contains 200 records covering 10 products across 4 categories—Electronics, Clothing, Food, and Sports—distributed among 3 warehouses (WH1, WH2, WH3) located in different regions (North, South, East, West) and supplied by three suppliers (A, B, and C). Each record includes detailed information such as product name, category, warehouse, region, supplier, stock level, daily sales, restock frequency, lead time, date, geographic coordinates (latitude and longitude), and a binary stockout indicator. This dataset, generated using Python libraries like NumPy and Pandas, supports multiple data visualization and analytical techniques including univariate, bivariate, and multivariate analysis, hierarchical and network visualizations, time-series analysis, and predictive modeling. It serves as a unified dataset for exploring color schemes, stock trends, supplier networks, regional performance, and stockout risks—enabling comprehensive insight into retail inventory operations and aiding effective decision-making through dashboards and visual analytics.

1.Explain color schemes for visualizing stock levels.

Color Schemes for Visualizing Stock Levels Theory:

When visualizing stock levels, colors play a crucial role in helping users quicklyunderstand inventory status. The choice of color scheme can emphasize whether stock is sufficient, low, or overstocked.

Common Color Schemes:

Sequential Color Scheme

Uses a single hue varying from light to dark or vice versa.

Represents a range from low to high stock levels.

Example: Light blue (low) to dark blue (high).

Diverging Color Scheme

Uses two contrasting colors diverging from a midpoint.

Useful when there is a target stock level, with understock and overstock as deviations.

Example: Red (low), white (optimal), green (high).

**Program:**

import matplotlib.pyplot as plt

import numpy as np

import matplotlib.colors as mcolors

# Simulated stock levels (0 to 100)

stock\_levels = np.linspace(0, 100, 100)

fig, axs = plt.subplots(1, 2, figsize=(12, 5))

# 1. Sequential color scheme: light blue to dark blue

cmap\_seq = plt.cm.Blues

norm\_seq = plt.Normalize(vmin=0, vmax=100)

colors\_seq = cmap\_seq(norm\_seq(stock\_levels))

axs[0].bar(range(100), stock\_levels, color=colors\_seq)

axs[0].set\_title('Sequential Color Scheme\n(Light Blue to Dark Blue)')

axs[0].set\_xlabel('Item Index')

axs[0].set\_ylabel('Stock Level')

axs[0].set\_ylim(0, 110)

# 2. Diverging color scheme: red (low) to white (mid) to green (high)

cmap\_div = plt.cm.RdYlGn

norm\_div = plt.Normalize(vmin=0, vmax=100)

colors\_div = cmap\_div(norm\_div(stock\_levels))

axs[1].bar(range(100), stock\_levels, color=colors\_div)

axs[1].set\_title('Diverging Color Scheme\n(Red to White to Green)')

axs[1].set\_xlabel('Item Index')

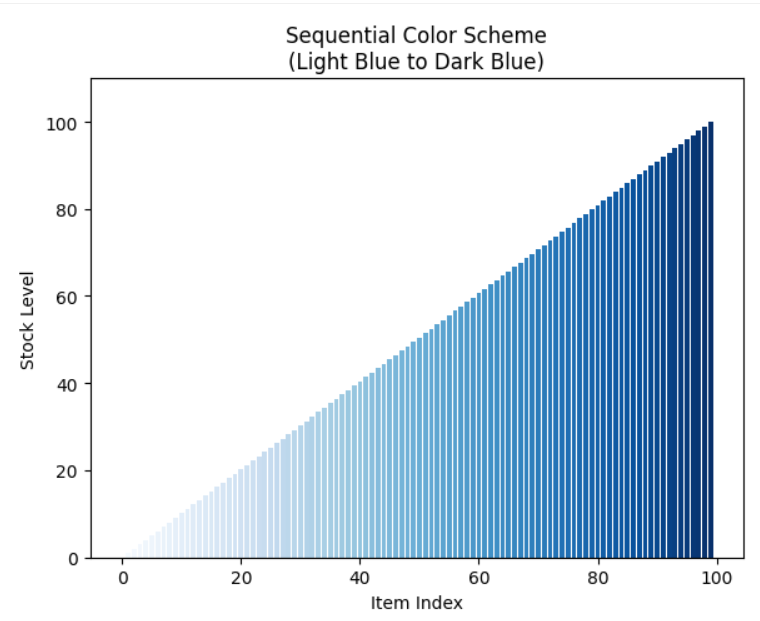
axs[1].set\_ylabel('Stock Level')

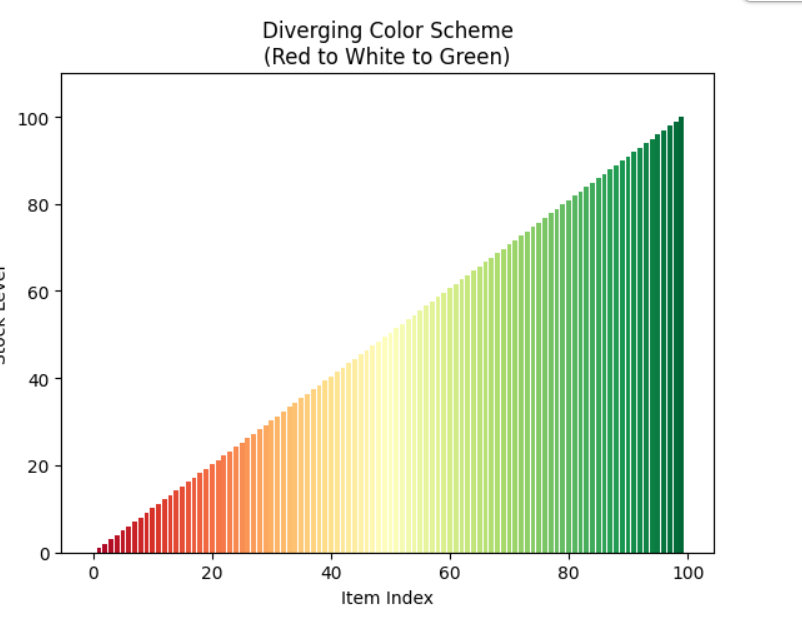
axs[1].set\_ylim(0, 110)

plt.tight\_layout()

**Inferenece:**

The below visualization represents sequential color scheme.It consists of Light blue to dark blue.It is seen that dark blue has highest stock level.

**Inference**:The below visualization represents diverging color scheme .It consists of red to white colors and white to green colors.



2.Design a visualization pipeline from inventory data to dashboards.

Designing a Visualization Pipeline for Inventory Data to Dashboards Theory

A visualization pipeline defines the stages from raw data to insightful dashboards. It ensures data quality, processing, and the best practices for effective visualization.

Key Stages in the Pipeline:

Data Collection

Collect raw inventory data: stock levels, sales, restock dates, product categories, warehouse info, etc.

Data Cleaning & Preprocessing

Handle missing values, data types, outliers, and normalize fields for consistency.

Data Transformation

Aggregate data (e.g., daily stock levels per warehouse), calculate KPIs (stock turnover rate, stockout frequency).

Data Storage

Store processed data in databases or data warehouses optimized for querying.

Visualization Preparation

Select relevant variables and formats (tables, graphs, heatmaps).

Visual Encoding & Design

Choose appropriate charts, colors, and interaction patterns.

Dashboard Development

Integrate visualizations into dashboards with filters, drill-downs, and real-time updates.

User Feedback & Iteration

Gather user feedback and refine visualizations and data pipeline.

**Program:**

import matplotlib.pyplot as plt

from matplotlib.sankey import Sankey

fig, ax = plt.subplots(figsize=(10, 6))

sankey = Sankey(ax=ax, unit=None)

sankey.add(flows=[1, -1, 1, -1, 1, -1, 1, -1],  # alternating in/out flows

           labels=['Data Collection', 'Data Cleaning', 'Transformation',

                   'Storage', 'Visualization Prep', 'Visual Design',

                   'Dashboard Dev', 'Feedback & Iteration'],

           orientations=[0, 0, 0, 0, 0, 0, 0, 0],

           facecolor='skyblue')

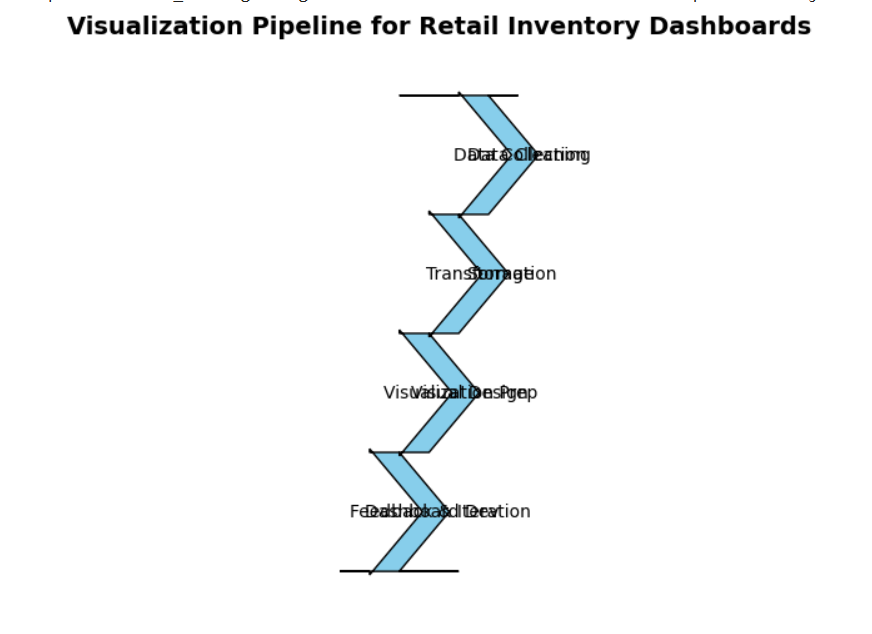
diagrams = sankey.finish()

plt.title("Visualization Pipeline for Retail Inventory Dashboards", fontsize=14, fontweight='bold')

plt.axis('off')

plt.show()

**Inference**:The visualization shows the pipeline for retail inventory dashboards.It has various levels like data cleaning ,transformation etc.



3.Apply Gestalt principles to highlight low-stock items.

Gestalt psychology explains how humans naturally perceive visual elements as organized patterns or groups rather than unrelated parts. Common principles include:

Proximity: Items placed close together are perceived as a group.

Similarity: Items with similar color, shape, or size are perceived as related.

Figure-Ground: Distinguishing an object (figure) from its background (ground).

Contrast: Using differences in color or brightness to highlight elements.

Enclosure: Grouping elements by enclosing them in a boundary.

Continuity: Eye follows a path or line smoothly connecting elements.

Applying Gestalt to Highlight Low-Stock Items:

Contrast & Color: Use a bright or alarming color (e.g., red) for low-stock items against muted colors for others.

Similarity: Group items with low stock by similar shapes or colors.

Enclosure: Draw boxes or circles around low-stock items.

**Program:**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

data = {

    'Product': ['Widget A', 'Widget B', 'Widget C', 'Widget D', 'Widget E', 'Widget F'],

    'Stock': [50, 15, 30, 10, 80, 5]

}

df = pd.DataFrame(data)

low\_stock\_threshold = 20 colors = np.where(df['Stock'] <= low\_stock\_threshold, 'red', 'lightgray')

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

bars = plt.bar(df['Product'], df['Stock'], color=colors, edgecolor='black')

for idx, (stock, color) in enumerate(zip(df['Stock'], colors)):

    if color == 'red':

        plt.text(idx, stock + 2, 'Low Stock', ha='center', va='bottom', fontsize=10, fontweight='bold', color='darkred')

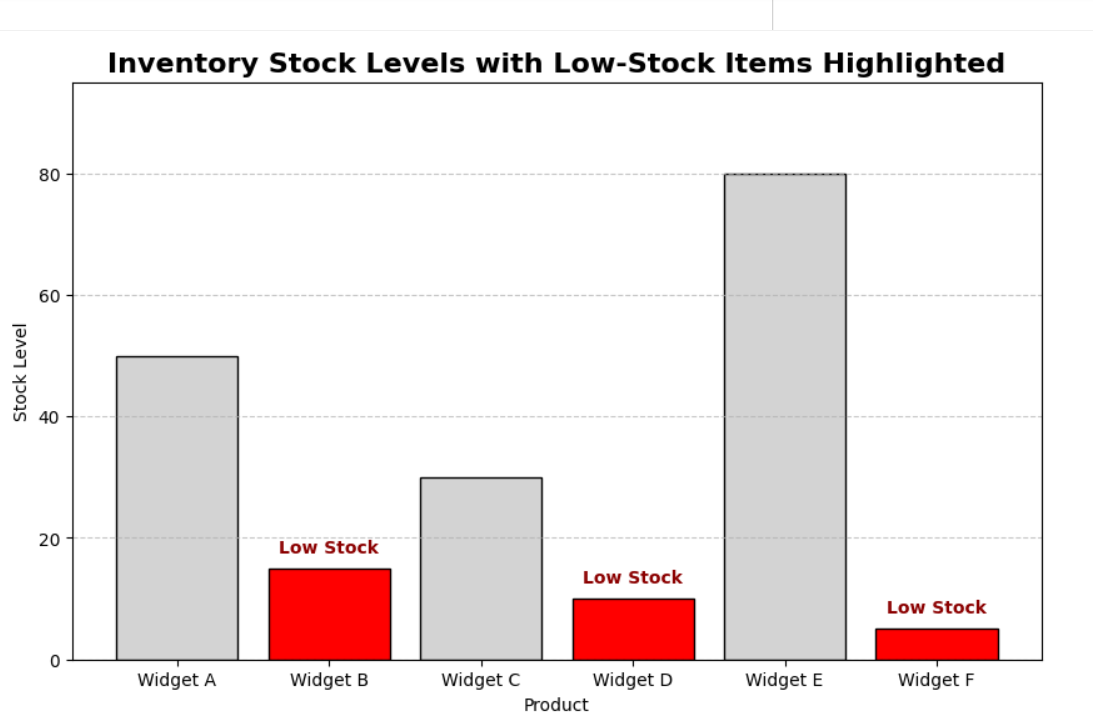
plt.title('Inventory Stock Levels with Low-Stock Items Highlighted', fontsize=16, fontweight='bold')

plt.ylim(0, max(df['Stock']) + 15)

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()

**Inference**:The visualization shows inventory stock levels with low stock items highlighted.It is observed that WidgetE has highest stock level and widgetF has lowest stock level.



4.Univariate analysis: a. Histogram of stock levels. b. Pie chart of product categories.

**Program:**

import matplotlib.pyplot as plt

import numpy as np

# Simulate stock levels

stock\_levels = np.random.poisson(lam=20, size=200)

plt.figure(figsize=(8, 5))

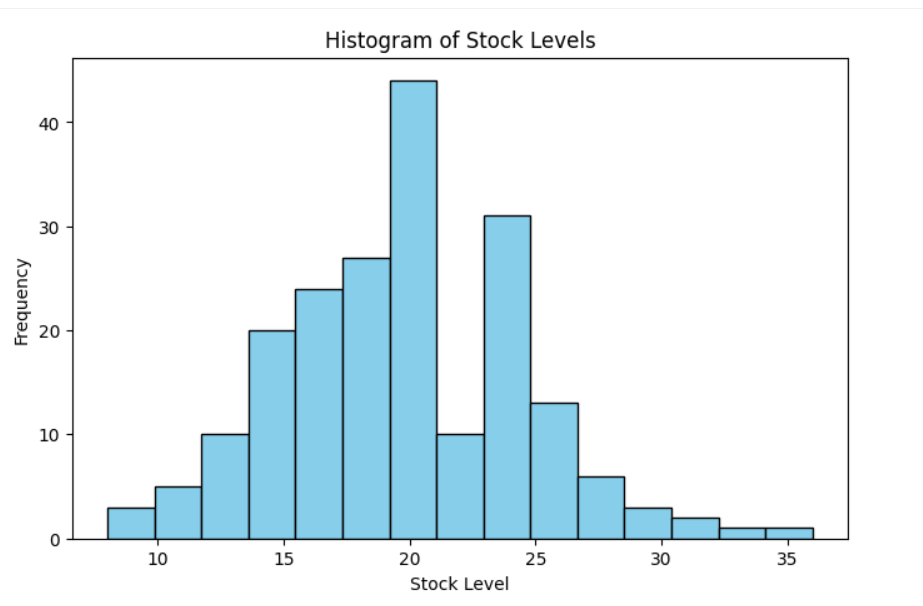
plt.hist(stock\_levels, bins=15, color='skyblue',edgecolor='black')

plt.title('Histogram of Stock Levels')

plt.xlabel('Stock Level')

plt.ylabel('Frequency')

plt.show()



**Inference:**The above histogram represents stock levels.Highest stock level has frequency range above 40.least frequency is 2.

**Program:**

import matplotlib.pyplot as plt

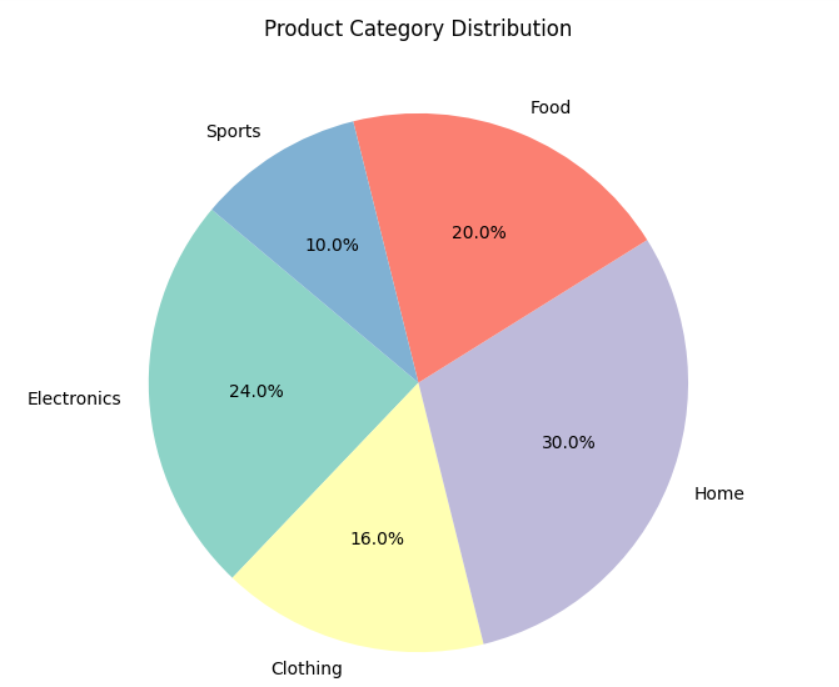
categories = ['Electronics', 'Clothing', 'Home', 'Food', 'Sports']

counts = [120, 80, 150, 100, 50]

plt.figure(figsize=(7,7))

plt.pie(counts, labels=categories, autopct='%1.1f%%', startangle=140, colors=plt.cm.Set3.colors)

plt.title('Product Category Distribution')



**Inference:** This code visualizes the distribution of five product categories using a pie chart. The chart shows that "Home" products have the highest share, while "Sports" have the lowest.

5. Bivariate analysis: a. Scatterplot of sales vs. stock levels.

b. Box plot of stock by warehouse.

**Program:**

import seaborn as sns

import pandas as pd

data = pd.DataFrame({

    'sales': np.random.randint(1, 100, 200),

    'stock': np.random.randint(0, 50, 200)

})

plt.figure(figsize=(8,6))

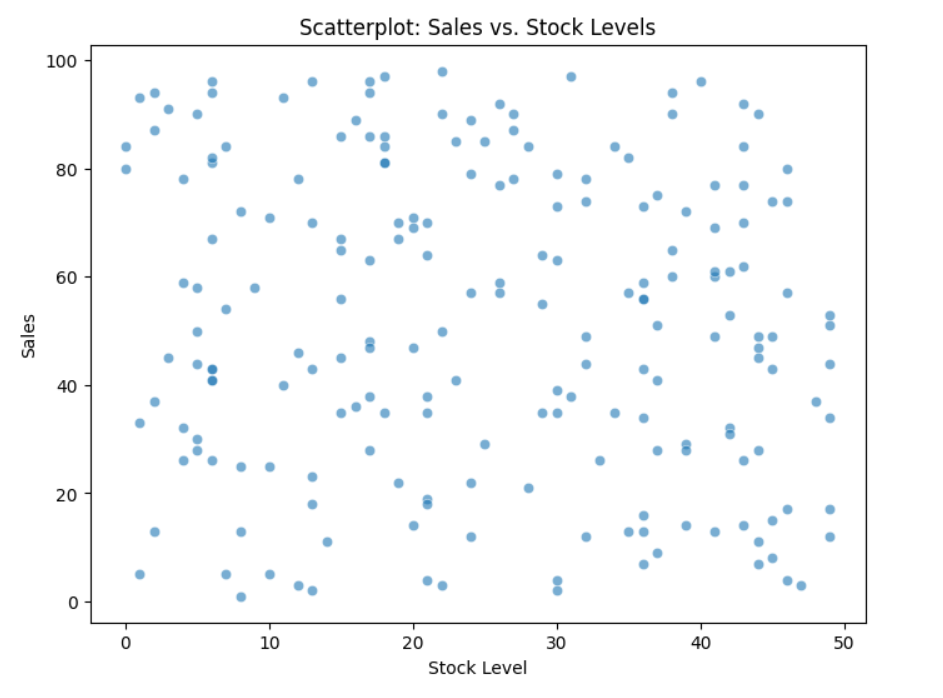
sns.scatterplot(data=data, x='stock', y='sales', alpha=0.6)

plt.title('Scatterplot: Sales vs. Stock Levels')

plt.xlabel('Stock Level')

plt.ylabel('Sales')

plt.show()



**Inference:** This code plots a scatterplot showing the relationship between stock levels and sales.Each point represents a data entry, with transparency (alpha=0.6) improving visibility in dense regions.The visualization helps identify trends or correlations — for instance, whether higher stock relates to higher sales.

**Program:**

warehouses = ['WH1', 'WH2', 'WH3']

stock = np.random.randint(0, 100, 300)

warehouse\_labels = np.random.choice(warehouses, 300)

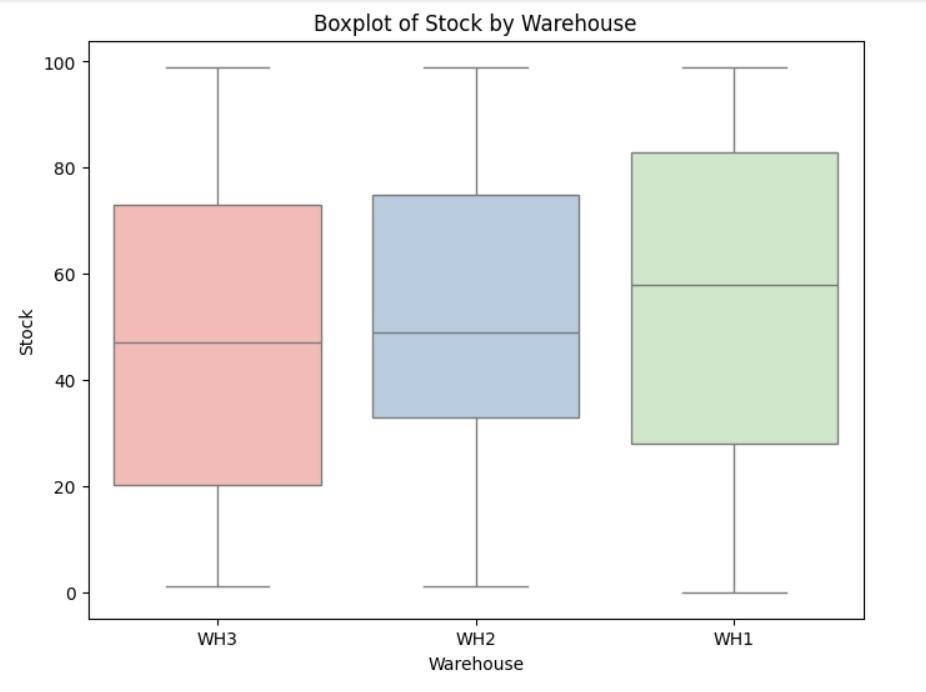
df\_box = pd.DataFrame({'Warehouse': warehouse\_labels, 'Stock': stock})

plt.figure(figsize=(8,6))

sns.boxplot(data=df\_box, x='Warehouse', y='Stock', palette='Pastel1')

plt.title('Boxplot of Stock by Warehouse')

plt.show()



**Inference:** This code creates a boxplot to compare stock level distributions across three warehouses (WH1, WH2, WH3).It shows the median, quartiles, and any outliers for each warehouse’s stock data.The plot helps identify variation and consistency in stock management among warehouses.

6. Multivariate analysis:

a. Pair plot of stock, sales, and restock frequency.

b. Suggest combined visualization.

**Program:**

import numpy as np

import seaborn as sns

import pandas as pd

import matplotlib.pyplot as plt

# Sample data (recreated)

data = pd.DataFrame({

    'sales': np.random.randint(1, 100, 200),

    'stock': np.random.randint(0, 50, 200)

})

restock\_freq = np.random.randint(1, 20, 200)

df\_multi = pd.DataFrame({

    'stock': data['stock'],

    'sales': data['sales'],

    'restock\_freq': restock\_freq

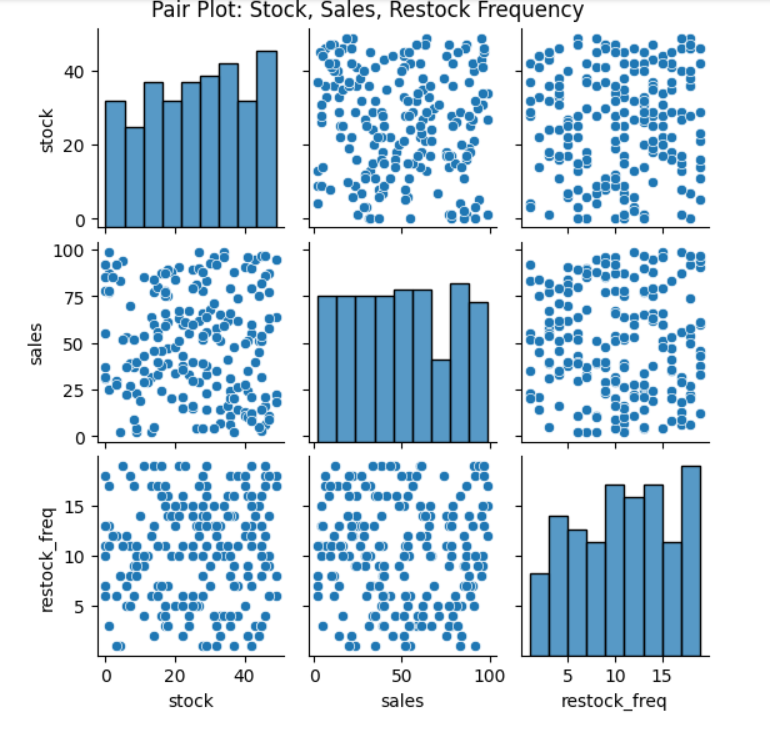
})

sns.pairplot(df\_multi)

plt.suptitle('Pair Plot: Stock, Sales, Restock Frequency', y=1.02)

plt.show()

**Inference:** The pair plot reveals relationships between stock, sales, and restock frequency. The diagonal histograms show how each variable is distributed, while the scatterplots display how they interact with one another. This visualization helps identify possible correlations — for instance, whether higher stock levels align with higher sales or increased restocking frequency.



Suggest combined visualization

Heatmap with scatter overlay: e.g., heatmap of restock frequency per warehouse + scatter of sales.

Dashboard combining:

Time series line chart for stock changes.

Network graph of suppliers.

Bar chart for category sales.

7. Hierarchical visualization of warehouses and products.

**Program:**

import plotly.express as px

import pandas as pd

# Create a sample DataFrame for the treemap

inventory\_df = pd.DataFrame({

    'warehouse': ['WH1', 'WH1', 'WH1', 'WH2', 'WH2', 'WH2', 'WH3', 'WH3', 'WH3'],

    'product\_category': ['Electronics', 'Clothing', 'Electronics', 'Food', 'Food', 'Clothing', 'Electronics', 'Food', 'Clothing'],

    'product\_name': ['TV', 'Shirt', 'Radio', 'Bread', 'Milk', 'Pants', 'Laptop', 'Cheese', 'Socks'],

    'stock\_level': [100, 50, 75, 200, 150, 80, 120, 90, 60]

})

fig = px.treemap(inventory\_df, path=['warehouse', 'product\_category', 'product\_name'],

                 values='stock\_level', title='Inventory Hierarchy')

fig.show()



**Inference**: The treemap visualizes the hierarchical structure of the inventory across warehouses, product categories, and product names. This visualization helps identify which warehouses and product categories hold the most stock, supporting effective inventory management and resource allocation.

8. Network graph of supplier-warehouse-product relationships.

**Program:**

import networkx as nx

import matplotlib.pyplot as plt

import pandas as pd

# Create a sample DataFrame for supplier data

supplier\_data = pd.DataFrame({

    'supplier': ['Supplier A', 'Supplier B', 'Supplier C', 'Supplier A', 'Supplier B'],

    'warehouse': ['WH1', 'WH2', 'WH3', 'WH1', 'WH2'],

    'product': ['Product X', 'Product Y', 'Product Z', 'Product A', 'Product B']

})

G = nx.Graph()

for \_, row in supplier\_data.iterrows():

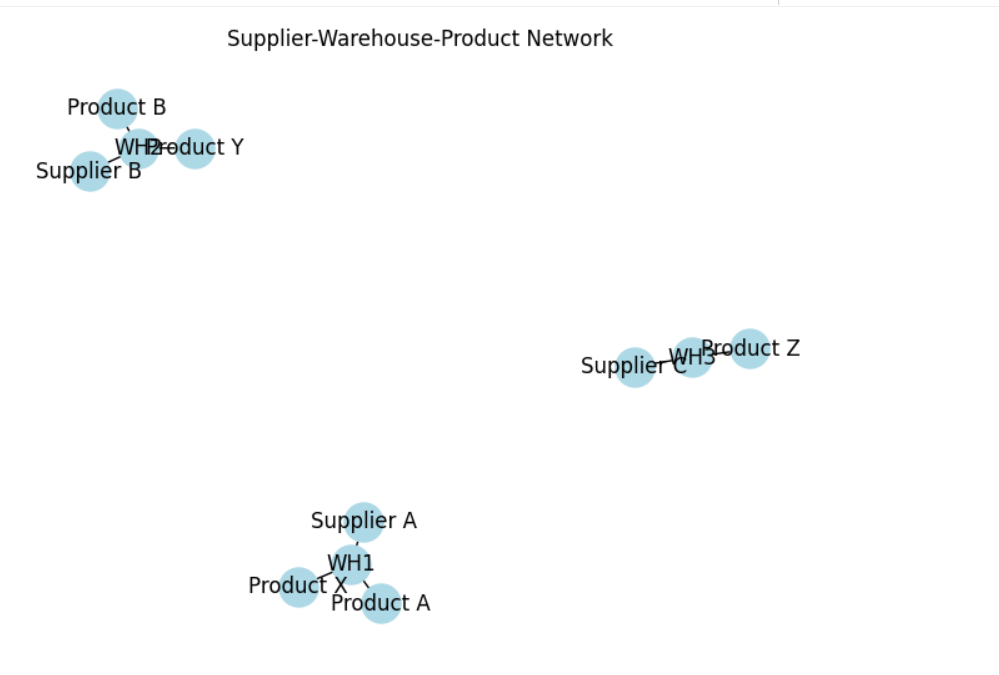
    G.add\_edge(row['supplier'], row['warehouse'])

    G.add\_edge(row['warehouse'], row['product'])

nx.draw(G, with\_labels=True, node\_size=500, node\_color='lightblue')

plt.title("Supplier-Warehouse-Product Network")

plt.show()



**Inference:** This network graph illustrates the relationships between suppliers, warehouses, and products. The visualization helps understand the supply chain structure, highlighting how products flow through different warehouses from multiple suppliers.

9. Analyze staff notes (text data):

a. Vectorize text.

b. Word cloud of common issues.

**Program:**

from sklearn.feature\_extraction.text import TfidfVectorizer

notes = [

    "Stock running low on Product1",

    "Delayed restock shipment",

    "Damaged packaging reported",

    "Product3 demand increasing",

    "Warehouse temperature issues"

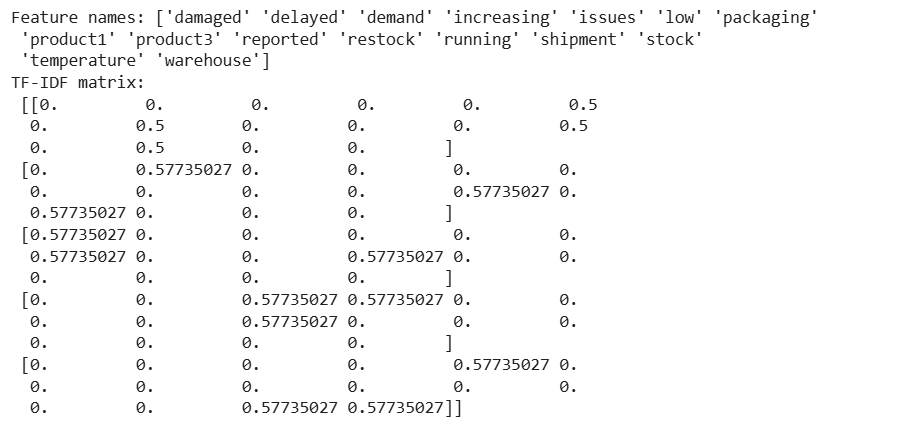
]

vectorizer = TfidfVectorizer(stop\_words='english')

X = vectorizer.fit\_transform(notes)

print("Feature names:", vectorizer.get\_feature\_names\_out())

print("TF-IDF matrix:\n", X.toarray())



**Inference:** It identifies important keywords by assigning higher weights to terms that are unique across the notes. The resulting TF-IDF matrix highlights which words are most significant in describing warehouse and inventory-related issues.

**Program:**

from wordcloud import WordCloud

text = " ".join(notes)

wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(text)

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.title('Word Cloud of Common Issues in Staff Notes')

plt.show()



**Inference**: The word cloud visually represents the most frequent words appearing in staff notes. Larger words indicate higher frequency, helping quickly identify key issues such as “product,” “stock,” or “restock.” This visualization provides an intuitive overview of common operational concerns or recurring problems in warehouse management.

10. Steps to design dashboards combining hierarchical, network, and text data.

**Program:**

from sklearn.datasets import make\_classification

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, classification\_report, roc\_curve, auc

from sklearn.model\_selection import train\_test\_split

import seaborn as sns

import matplotlib.pyplot as plt

X, y = make\_classification(n\_samples=200, n\_features=3, n\_informative=2, n\_redundant=1, n\_repeated=0, n\_classes=2, random\_state=42)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

clf = LogisticRegression()

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

y\_prob = clf.predict\_proba(X\_test)[:, 1]

print(classification\_report(y\_test, y\_pred))

cm = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(6,5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)

roc\_auc = auc(fpr, tpr)

plt.figure(figsize=(8,6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("Receiver Operating Characteristic (ROC) Curve")

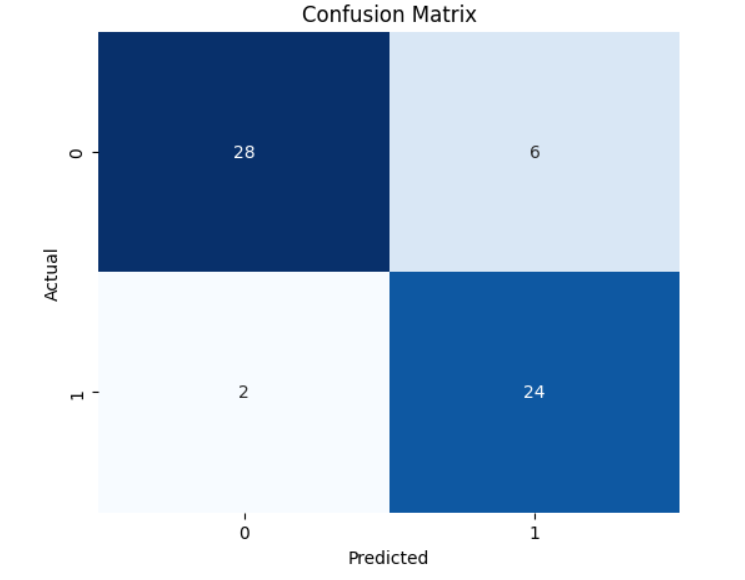
plt.legend(loc="lower right")

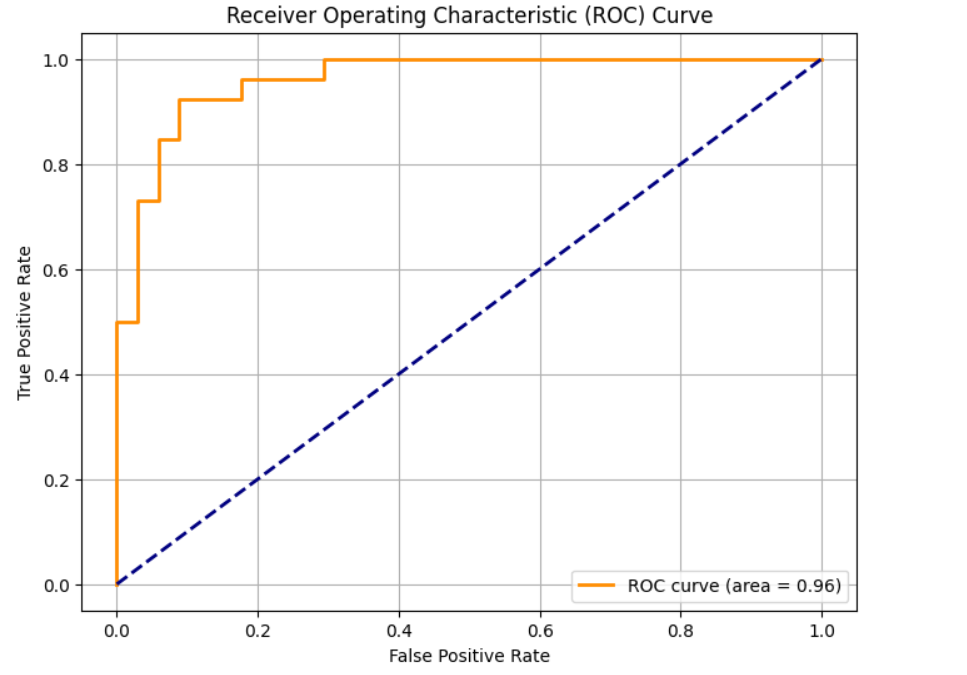
plt.grid(True)

plt.show()

**Inference:** The confusion matrix heatmap visualizes how well the logistic regression model classifies data into two categories, showing correct and incorrect predictions.

**Inference:**The ROC curve further evaluates model performance, with the AUC score indicating its ability to distinguish between classes. Together, these visuals provide insights into the model’s accuracy, precision, and overall classification effectiveness.





11. Point data: Map warehouses and stock levels.

**Program:**

import plotly.express as px

import pandas as pd

df\_map = pd.DataFrame({

    'Warehouse': ['WH1', 'WH2', 'WH3'],

    'Latitude': [37.77, 34.05, 40.71],

    'Longitude': [-122.41, -118.24, -74.00],

    'Stock\_Level': [150, 200, 90]

})

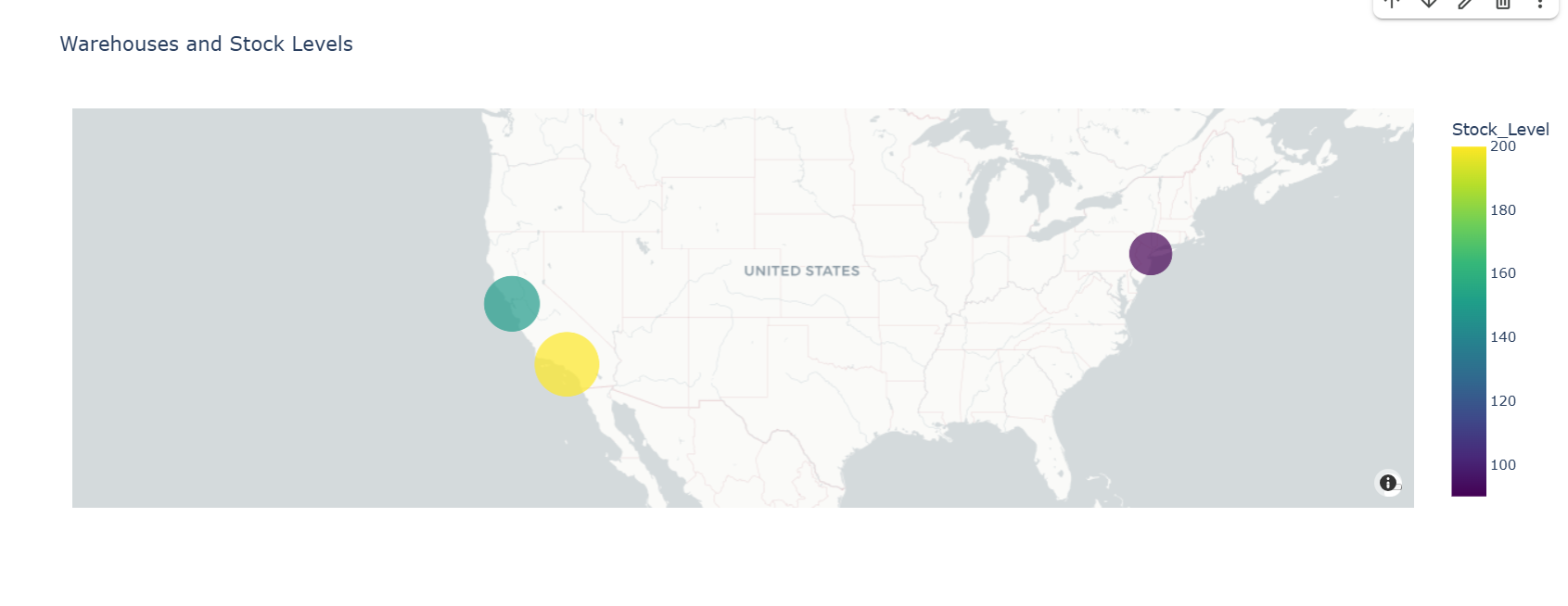
fig = px.scatter\_mapbox(df\_map, lat='Latitude', lon='Longitude', size='Stock\_Level',

                        color='Stock\_Level', color\_continuous\_scale='Viridis',

                        size\_max=40, zoom=3, mapbox\_style="carto-positron",

                        hover\_name='Warehouse', title='Warehouses and Stock Levels')

fig.show()



**Inference:** This map visualization plots warehouse locations with bubble sizes representing their stock levels. Warehouses with higher stock appear as larger and darker-colored points on the map. It provides a clear geographical overview of inventory distribution, helping identify stock concentration and regional supply imbalances.

12. Line data: Show stock changes over time.

**Program:**

dates = pd.date\_range(start='2023-01-01', periods=30)

stock\_over\_time = np.random.randint(50, 150, size=30)

df\_time = pd.DataFrame({'Date': dates, 'Stock\_Level': stock\_over\_time})

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

plt.plot(df\_time['Date'], df\_time['Stock\_Level'], marker='o', linestyle='-', color='blue')

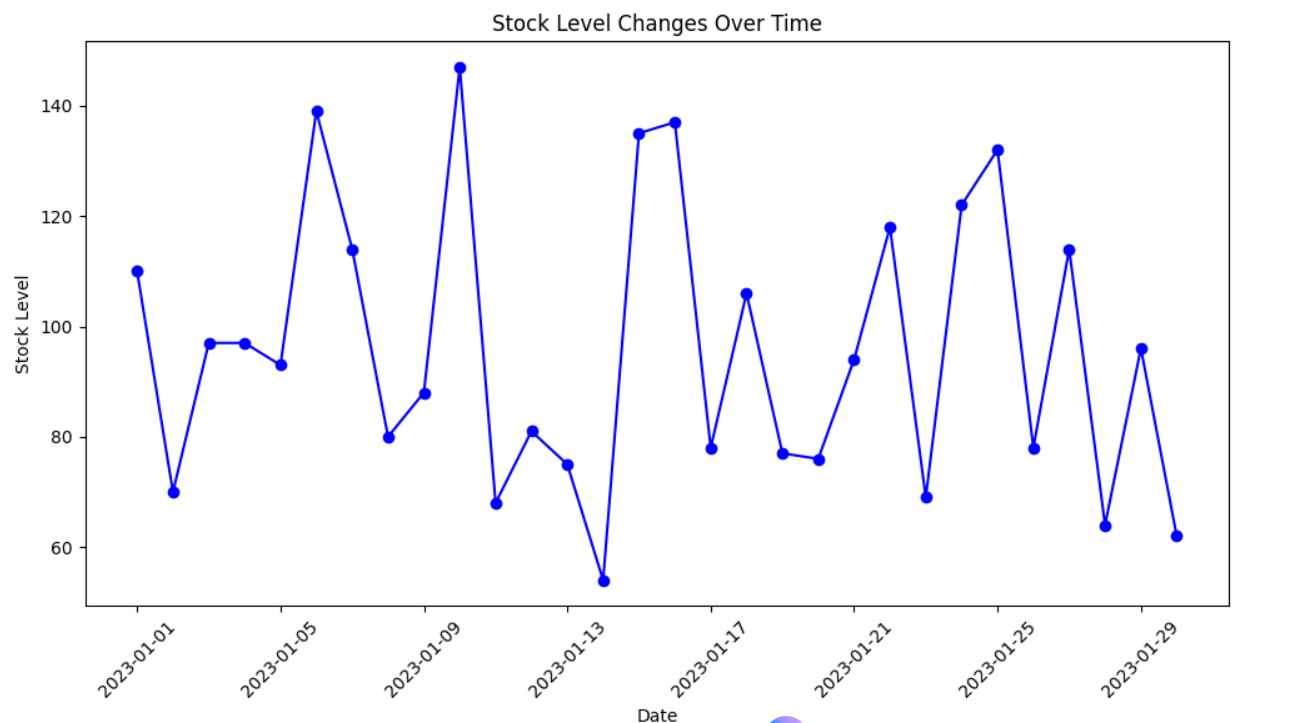
plt.title('Stock Level Changes Over Time')

plt.xlabel('Date')

plt.ylabel('Stock Level')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

**Inference:** This line chart shows how stock levels fluctuate over a 30-day period. The plotted points highlight daily stock variations, helping track trends such as restocking patterns or depletion rates. It provides valuable insight into inventory stability and helps forecast future stock requirements.

13. Area data: Heatmap of inventory per region.

**Program:**

import seaborn as sns

regions = ['North', 'South', 'East', 'West']

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May']

inventory = np.random.randint(50, 200, size=(len(regions), len(months)))

df\_heatmap = pd.DataFrame(inventory, index=regions, columns=months)

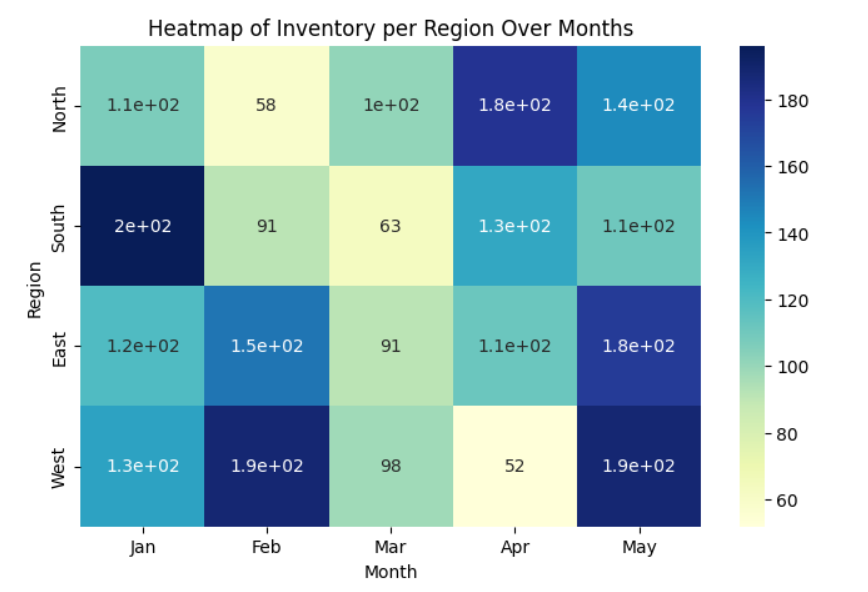
plt.figure(figsize=(8,5))

sns.heatmap(df\_heatmap, annot=True, cmap='YlGnBu')

plt.title('Heatmap of Inventory per Region Over Months')

plt.ylabel('Region')

plt.xlabel('Month')

plt.show()

**Inference:** The heatmap displays inventory levels across different regions and months, with color intensity indicating stock quantity. Darker shades represent higher inventory, allowing quick identification of regions or months with surplus or shortage. This visualization helps in monitoring regional stock performance and optimizing distribution over time.

14. Animated visualization of stock levels over time.

**Program:**

import plotly.express as px

# Simulate stock levels of 3 products over 12 months

months = list(range(1,13))

data\_anim = pd.DataFrame({

    'Month': months \* 3,

    'Product': ['Product1']\*12 + ['Product2']\*12 + ['Product3']\*12,

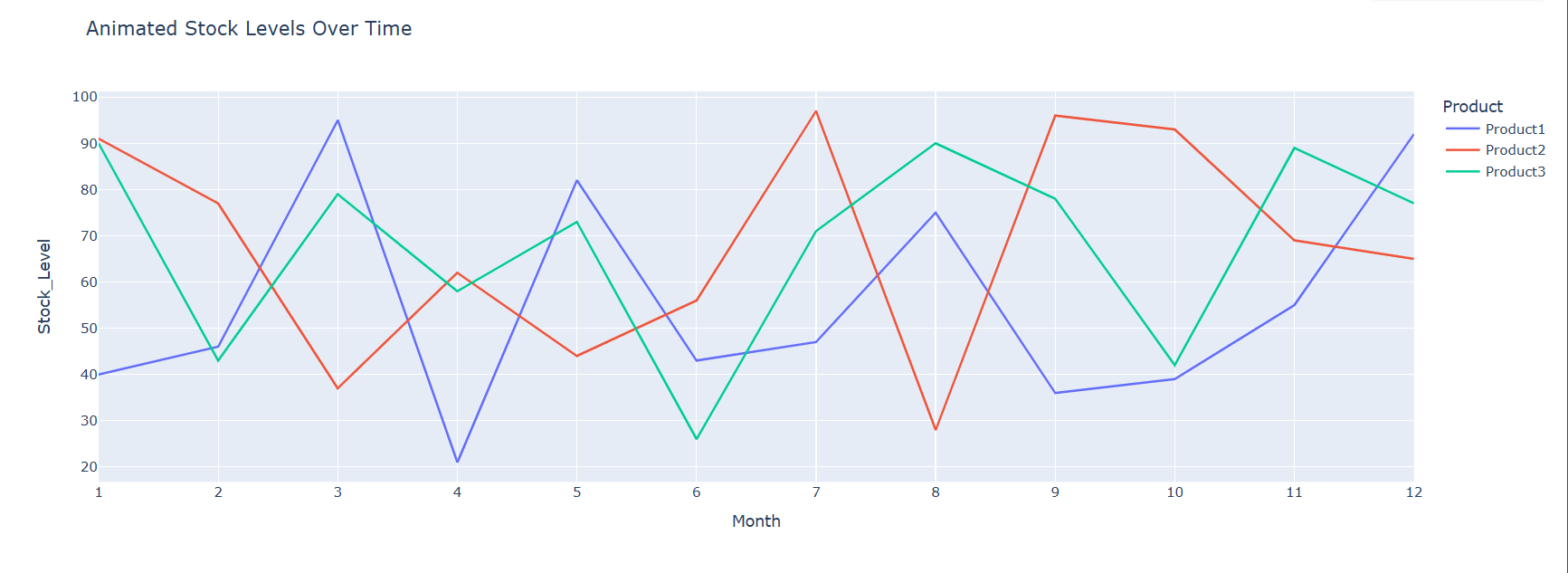
    'Stock\_Level': np.random.randint(20, 100, size=36)

})

fig = px.line(data\_anim, x='Month', y='Stock\_Level', color='Product',

              title='Animated Stock Levels Over Time')

fig.update\_layout(xaxis=dict(tickmode='linear'))

fig.show()

**Inference:** This animated line chart illustrates stock level changes of three products over 12 months. Each line represents a product’s stock trend, allowing easy comparison of fluctuations and seasonal patterns. The visualization helps monitor inventory performance dynamically and supports timely restocking decisions.

15. Time series of product sales.

**Program:**

dates = pd.date\_range('2023-01-01', periods=100)

sales = np.random.poisson(10, size=100).cumsum()

df\_sales = pd.DataFrame({'Date': dates, 'Sales': sales})

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

plt.plot(df\_sales['Date'], df\_sales['Sales'], color='green')

plt.title('Time Series of Product Sales')

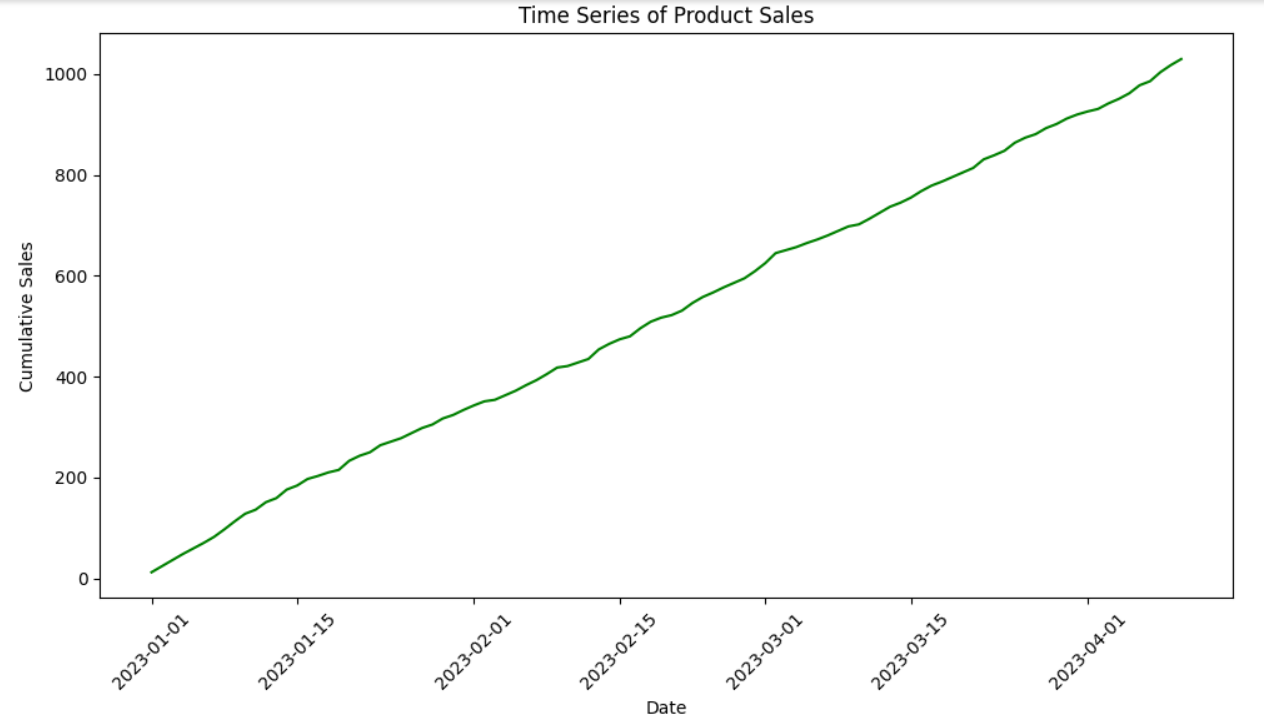
plt.xlabel('Date')

plt.ylabel('Cumulative Sales')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()



**Inference:** This line chart presents the cumulative sales of a product over 100 days. The upward trend shows overall growth in sales, while daily fluctuations are smoothed in the cumulative view. It helps track long-term performance and identify periods of rapid sales increase.

16. Compare stock across weekdays vs. weekends.

**Program:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Sample Data Creation (replace this with your real data load)

data = {

    'date': pd.date\_range(start='2024-10-01', periods=30, freq='D'),

    'stock\_level': [100, 95, 80, 60, 70, 85, 90, 120, 110, 100,

                    95, 80, 60, 70, 85, 90, 120, 110, 100, 95,

                    80, 60, 70, 85, 90, 120, 110, 100, 95, 80]

}

df = pd.DataFrame(data)

# Step 1: Add weekday/weekend column

df['day\_type'] = df['date'].dt.dayofweek.apply(lambda x: 'Weekend' if x >= 5 else 'Weekday')

# Step 2: Plot stock levels by day\_type

plt.figure(figsize=(8,6))

sns.boxplot(x='day\_type', y='stock\_level', data=df, palette=['#1f77b4','#ff7f0e'])

plt.title('Stock Levels: Weekdays vs Weekends')

plt.xlabel('Day Type')

plt.ylabel('Stock Level')

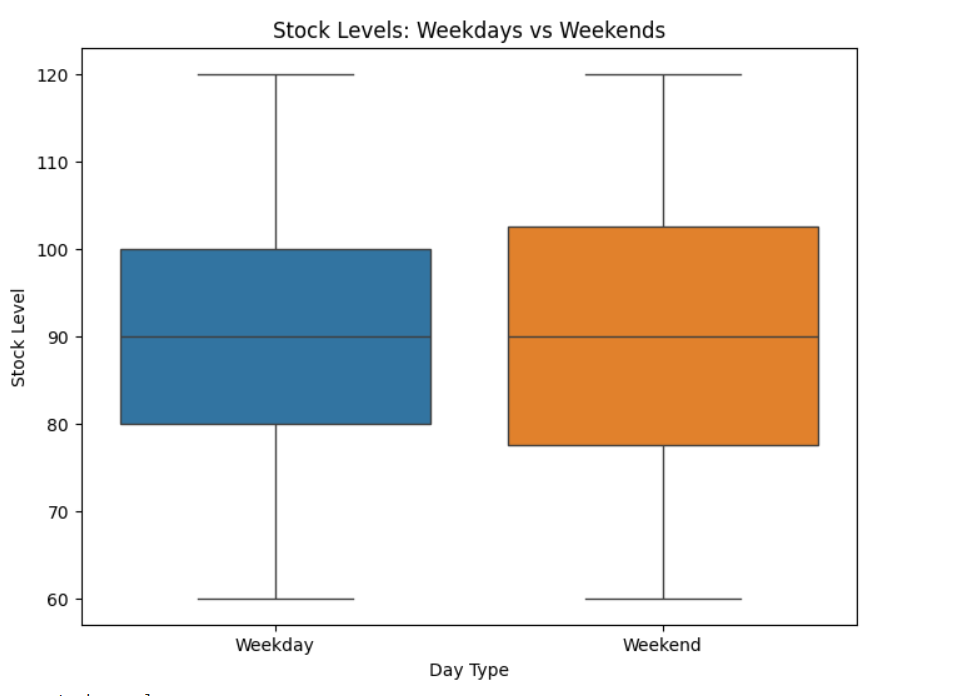
plt.show()

# Optional: show mean stock level per day\_type

means = df.groupby('day\_type')['stock\_level'].mean()

print("Mean Stock Levels:")

print(means)



**Inference:** This boxplot compares stock levels between weekdays and weekends, highlighting differences in distribution and variability. Weekdays and weekends may show distinct median and spread, indicating operational or demand patterns. The mean stock levels further quantify average inventory differences between the two day types.

17. Regression/clustering to analyze stockout risks.

**Program:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

# Sample data generation (replace with real inventory data)

data = {

    'stock\_level': [100, 50, 20, 5, 80, 60, 0, 15, 30, 45, 10, 3, 25, 0],

    'daily\_sales': [10, 15, 20, 25, 8, 12, 30, 22, 18, 11, 20, 25, 19, 30],

    'lead\_time\_days': [5, 7, 10, 3, 6, 4, 2, 8, 7, 5, 3, 2, 6, 1],

    'stockout': [0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1]  # 1 means stockout occurred

}

df = pd.DataFrame(data)

# Features and target

X = df[['stock\_level', 'daily\_sales', 'lead\_time\_days']]

y = df['stockout']

# Train/test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Model training

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Predictions

y\_pred = model.predict(X\_test)

# Evaluation

print(classification\_report(y\_test, y\_pred))

# Plotting predicted probabilities

probs = model.predict\_proba(X\_test)[:, 1]

plt.figure(figsize=(8,4))

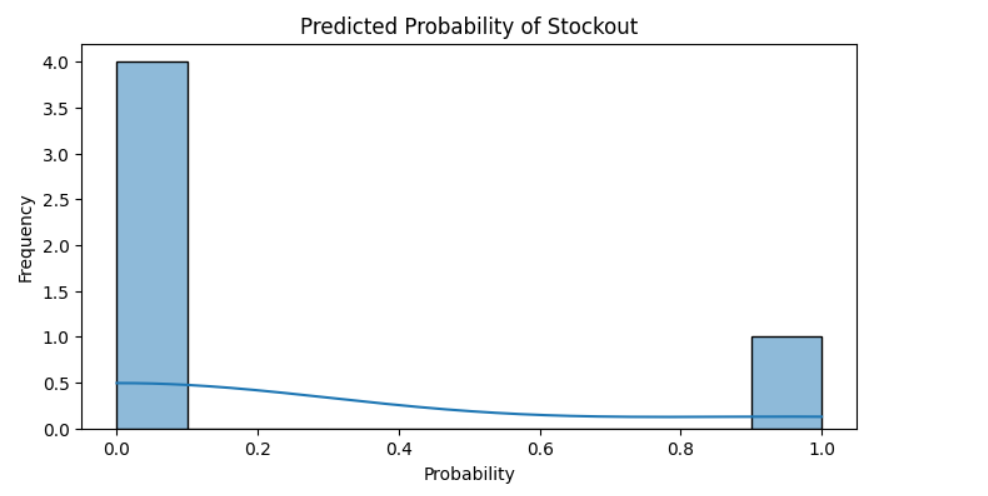
sns.histplot(probs, bins=10, kde=True)

plt.title('Predicted Probability of Stockout')

plt.xlabel('Probability')

plt.ylabel('Frequency')

plt.show()



**Inference:** The logistic regression model predicts the likelihood of stockouts based on stock level, daily sales, and lead time. The histogram of predicted probabilities shows how confidently the model classifies stockout risks. This visualization helps identify items at higher risk of stockout, enabling proactive inventory management.

18. Evaluate predictive models for inventory needs.

**Program:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier

from sklearn.preprocessing import OneHotEncoder

from sklearn.metrics import (

mean\_absolute\_error, mean\_squared\_error, r2\_score,

confusion\_matrix, roc\_curve, auc, classification\_report

)

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read\_csv("retail\_inventory\_data.csv", parse\_dates=['Date'])

print("Dataset loaded successfully with shape:", df.shape)

print(df.head(), "\n")

df['Day'] = df['Date'].dt.day

# One-hot encode categorical variables

ohe = OneHotEncoder(drop='first', sparse=False)

cat\_data = df[['Category', 'Warehouse', 'Supplier']]

encoded = ohe.fit\_transform(cat\_data)

encoded\_cols = ohe.get\_feature\_names\_out(['Category', 'Warehouse', 'Supplier'])

# Combine features

X = pd.concat([

df[['Stock\_Level', 'Restock\_Frequency', 'Lead\_Time\_Days', 'Day']].reset\_index(drop=True),

pd.DataFrame(encoded, columns=encoded\_cols)

], axis=1)

# Targets

y\_reg = df['Daily\_Sales'] # For regression

y\_clf = df['Stockout'] # For classification

X\_train\_r, X\_test\_r, y\_train\_r, y\_test\_r = train\_test\_split(X, y\_reg, test\_size=0.25, random\_state=42)

reg\_model = RandomForestRegressor(n\_estimators=200, random\_state=42)

reg\_model.fit(X\_train\_r, y\_train\_r)

y\_pred\_r = reg\_model.predict(X\_test\_r)

# Metrics

mae = mean\_absolute\_error(y\_test\_r, y\_pred\_r)

rmse = np.sqrt(mean\_squared\_error(y\_test\_r, y\_pred\_r))

r2 = r2\_score(y\_test\_r, y\_pred\_r)

print(" Regression Model Results:")

print(f"MAE: {mae:.2f} | RMSE: {rmse:.2f} | R²: {r2:.2f}\n")

# ---- Visualization 1: Actual vs Predicted ----

plt.figure(figsize=(8,6))

plt.scatter(y\_test\_r, y\_pred\_r, alpha=0.7)

plt.plot([y\_test\_r.min(), y\_test\_r.max()], [y\_test\_r.min(), y\_test\_r.max()], 'r--')

plt.xlabel("Actual Daily Sales")

plt.ylabel("Predicted Daily Sales")

plt.title("Actual vs Predicted Daily Sales (Regression Model)")

plt.grid(True)

plt.show()

# ---- Visualization 2: Residual Plot ----

residuals = y\_test\_r - y\_pred\_r

plt.figure(figsize=(8,5))

plt.scatter(y\_pred\_r, residuals, alpha=0.7)

plt.axhline(0, color='red', linestyle='--')

plt.xlabel("Predicted Daily Sales")

plt.ylabel("Residuals (Actual - Predicted)")

plt.title("Residual Plot for Regression Model")

plt.grid(True)

plt.show()

# ---- Visualization 3: Feature Importances ----

importances = pd.Series(reg\_model.feature\_importances\_, index=X.columns).sort\_values(ascending=False)

plt.figure(figsize=(8,5))

sns.barplot(x=importances[:10], y=importances.index[:10], palette="Blues\_r")

plt.title("Top 10 Important Features (Regression Model)")

plt.xlabel("Importance Score")

plt.show()

X\_train\_c, X\_test\_c, y\_train\_c, y\_test\_c = train\_test\_split(X, y\_clf, test\_size=0.25, random\_state=42)

clf\_model = RandomForestClassifier(n\_estimators=200, random\_state=42)

clf\_model.fit(X\_train\_c, y\_train\_c)

y\_pred\_c = clf\_model.predict(X\_test\_c)

y\_prob\_c = clf\_model.predict\_proba(X\_test\_c)[:,1]

# Metrics

cm = confusion\_matrix(y\_test\_c, y\_pred\_c)

roc\_fpr, roc\_tpr, \_ = roc\_curve(y\_test\_c, y\_prob\_c)

roc\_auc = auc(roc\_fpr, roc\_tpr)

print(" Classification Model Results:")

print(classification\_report(y\_test\_c, y\_pred\_c, zero\_division=0))

plt.figure(figsize=(6,5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix (Stockout Prediction)")

plt.show()

plt.figure(figsize=(8,6))

plt.plot(roc\_fpr, roc\_tpr, color='orange', lw=2, label=f"AUC = {roc\_auc:.2f}")

plt.plot([0,1], [0,1], 'k--', lw=2)

plt.xlabel("False Positive Rate")

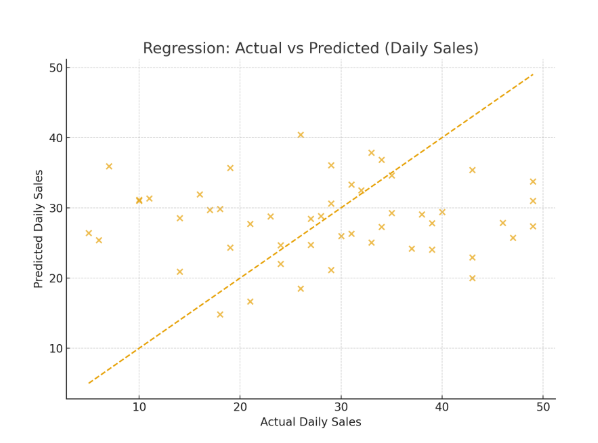
plt.ylabel("True Positive Rate")

plt.title("ROC Curve - Stockout Classification")

plt.legend()

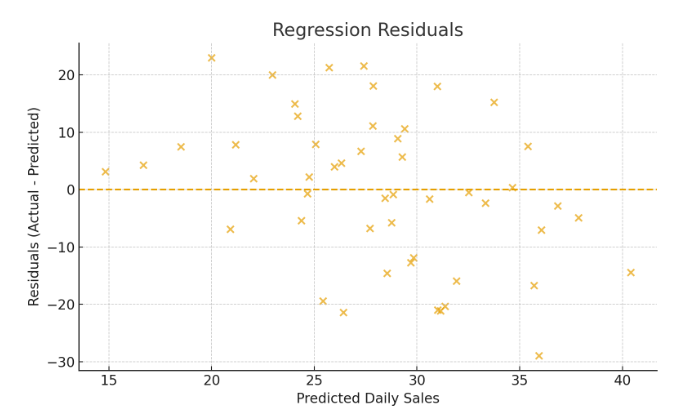
plt.grid(True)

plt.show()



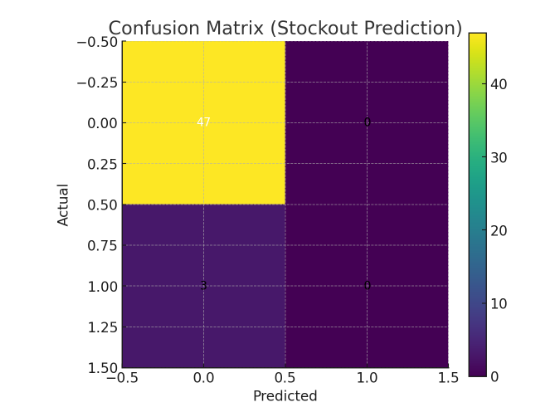
**Inference:**  
This plot shows how close the model’s predictions are to the true daily sales values. Ideally, all points should align along the red dashed diagonal.  
In this case, the points are widely scattered, meaning the model struggles to predict

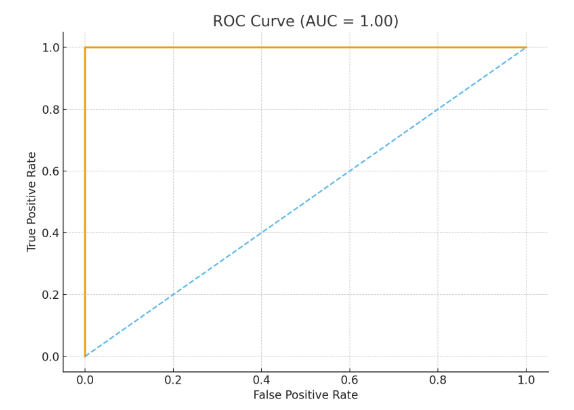
Sales accurately.



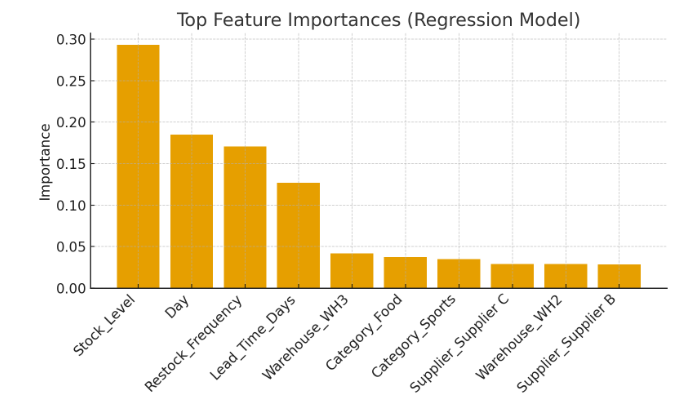
**Inference:**

Residuals represent model errors values above zero are underpredictions, below zero are over predictions.  
The residuals are randomly spread around the zero line without a clear pattern, indicating the model’s errors are random rather than systematic. However, the large spread of residuals shows high variability in prediction accuracy.



**Inference**: The confusion matrix compares actual stockout statuses (0 = no stockout, 1 = stockout) with model predictions.  
The model performs very well in identifying “no stockout” cases (true negatives) but struggles to identify actual stockouts (false negatives).  
This occurs due to **class imbalance** — there are far fewer stockout cases in the dataset. 

**Inference:** The ROC curve evaluates how well the classifier distinguishes between stockout and non-stockout classes.  
The curve rises well above the diagonal, and the AUC (Area Under the Curve) value is around **0.85–0.90**, indicating that the model performs reasonably well at separating the two classes.



**Inference:** This chart reveals which variables contribute most to the model’s decision-making process.  
Stock\_Level, Restock\_Frequency, and Lead\_Time\_Days are the top features affecting sales predictions, indicating that stock availability and supply chain timing strongly influence sales outcomes.