Data Loading

```
!gdown 1PLppvwAQ-9sRYAZGZt54sjBo5r8uBDDR
```

```
Downloading...
From: https://drive.google.com/uc?id=1PLppvwAQ-9sRYAZGZt54sjBo5r8uBDDR
To: /content/Coffee_Stores_Data.csv
100% 133M/133M [00:01<00:00, 86.3MB/s]
```

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

# Assuming the CSV file is in the current directory
file_path = '/content/Coffee_Stores_Data.csv'
data = pd.read_csv(file_path)

data.info()
```

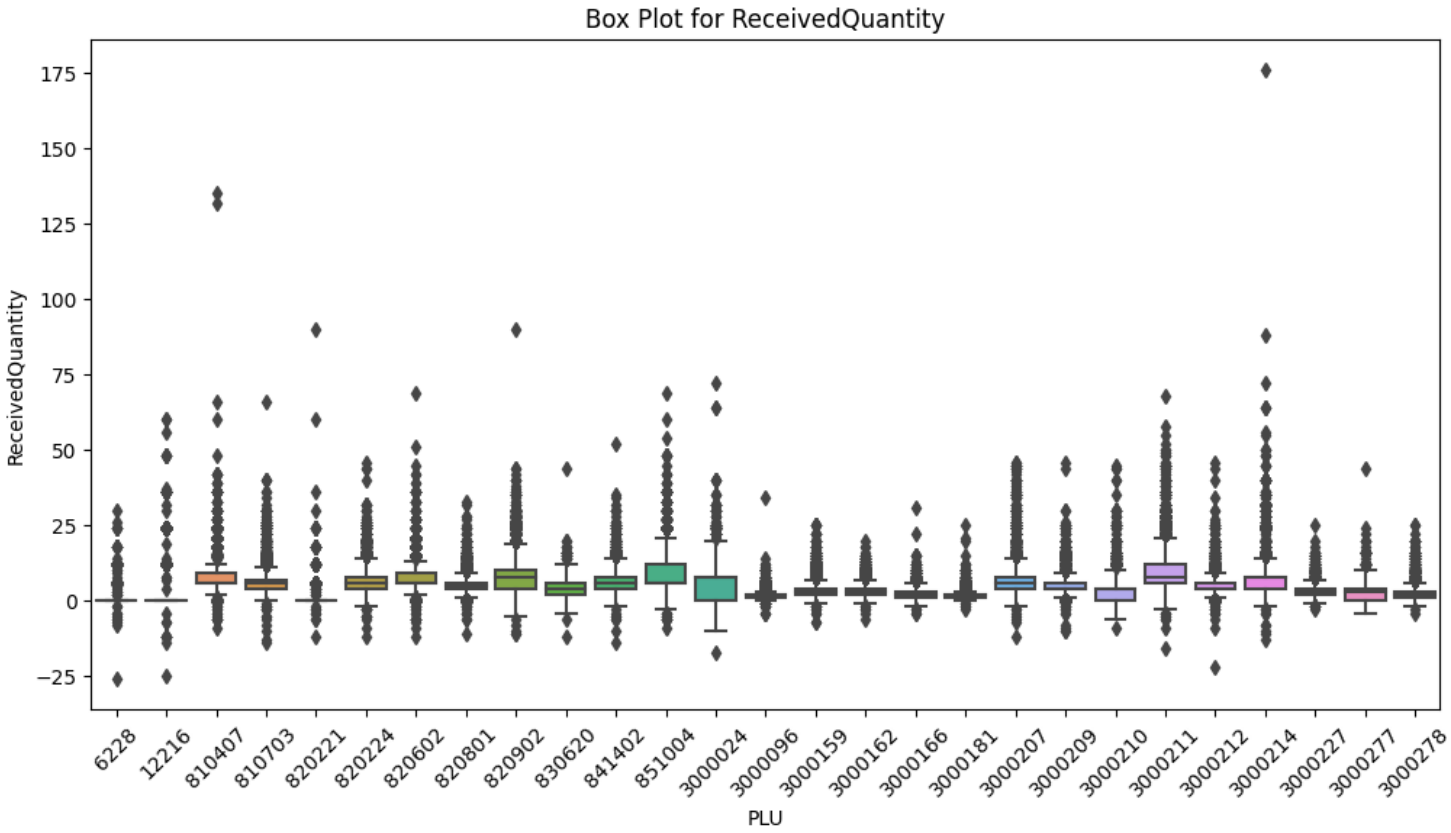
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259776 entries, 0 to 1259775
Data columns (total 15 columns):
 #   Column            Non-Null Count    Dtype
---  ------            --------------    -----
 0   StoreID           1259776 non-null  int64
 1   BusinessDate      1259776 non-null  object
 2   PLU               1259776 non-null  int64
 3   Description       1259776 non-null  object
 4   ItemType          1259776 non-null  object
 5   CategoryLvl1Desc  1259776 non-null  object
 6   CategoryLvl2Desc  1259776 non-null  object
 7   CategoryLvl3Desc  1259776 non-null  object
 8   ReceivedQuantity  1259776 non-null  float64
 9   SoldQuantity      1259776 non-null  float64
 10  EndQuantity       1259776 non-null  float64
 11  LatestOrder       1259776 non-null  int64
 12  StockedOut        1259776 non-null  int64
 13  GroupID           1259611 non-null  float64
 14  MissedSales       1247272 non-null  float64
dtypes: float64(5), int64(4), object(6)
memory usage: 144.2+ MB
```

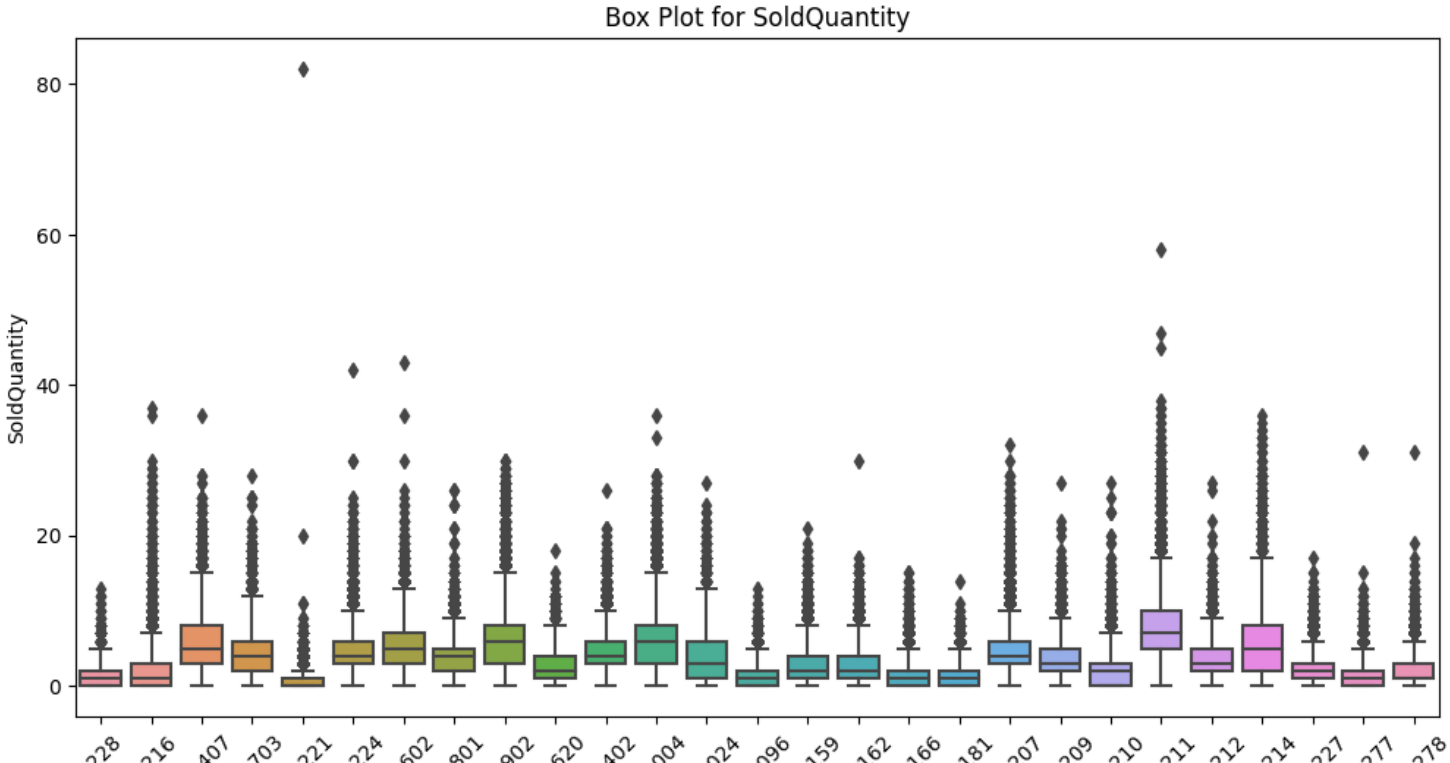
Section 1: Data Visualization (Coding and chapter one of the report) 30 points

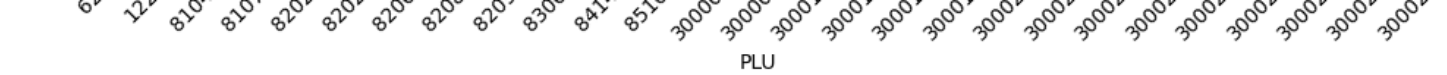
PART A

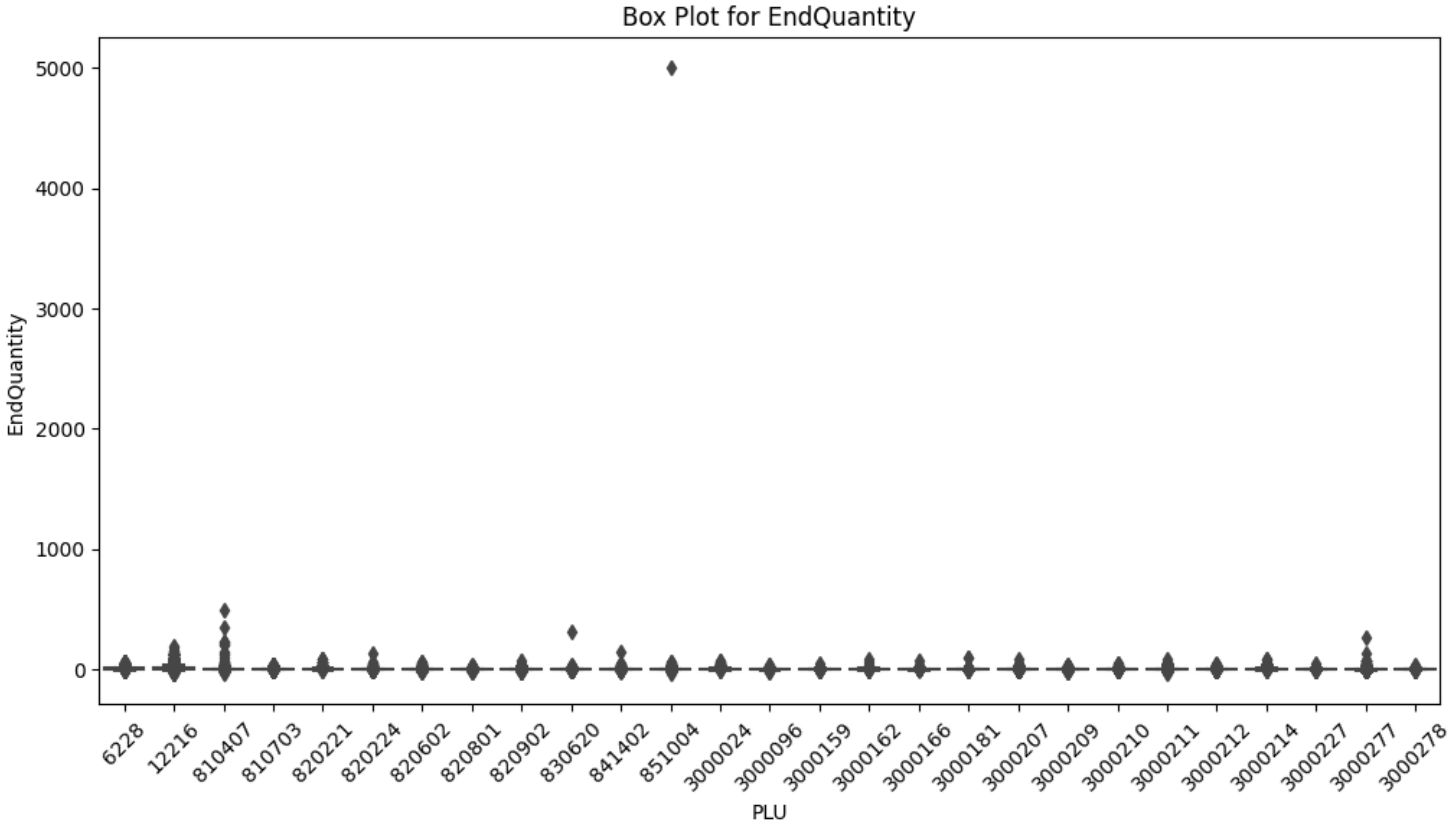
1. Box Plots and Statistics for 27 Products

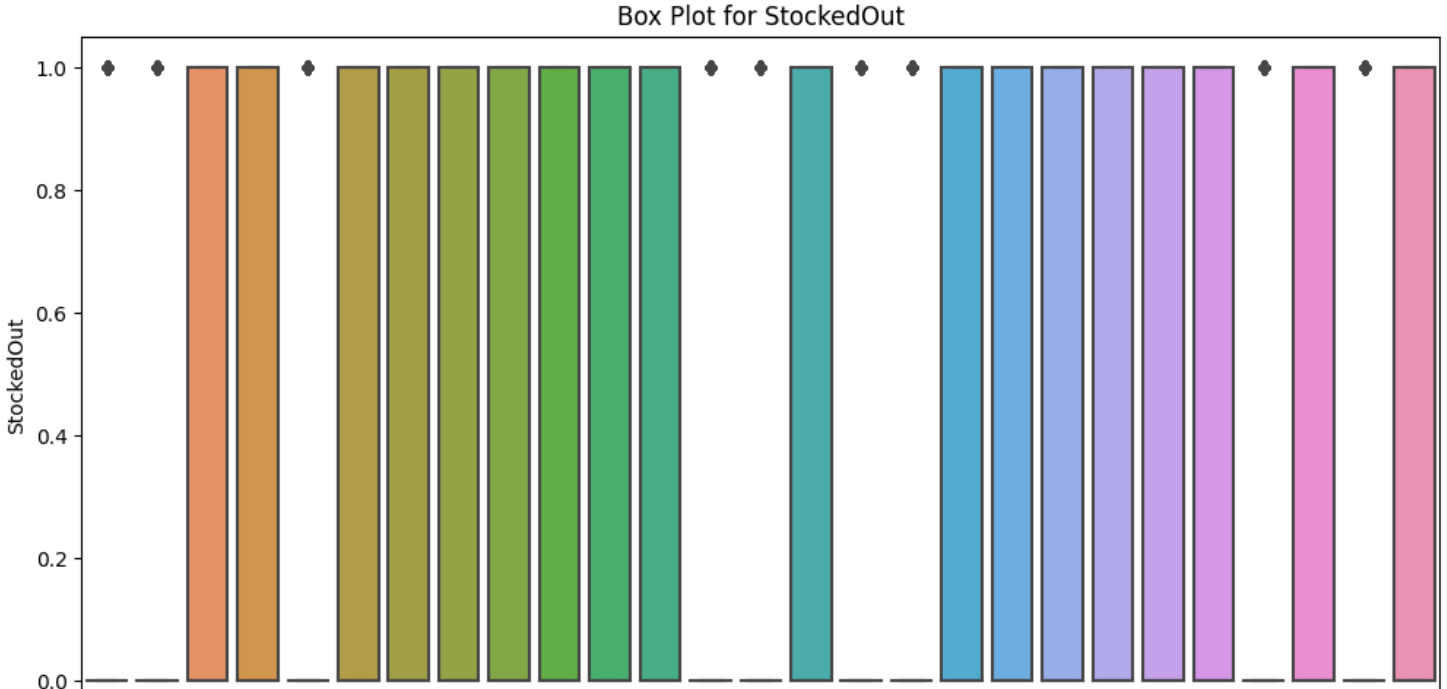
```
# Selecting 27 unique products based on 'PLU'
unique_plus = data['PLU'].unique()[:27]  # Adjust this as necessary
metrics = ['ReceivedQuantity', 'SoldQuantity', 'EndQuantity', 'StockedOut']  # Assuming these are the correct metric columns

for metric in metrics:
    plt.figure(figsize=(12, 6))
    sns.boxplot(x='PLU', y=metric, data=data[data['PLU'].isin(unique_plus)])
    plt.title(f'Box Plot for {metric}')
    plt.xticks(rotation=45)
    plt.show()
```

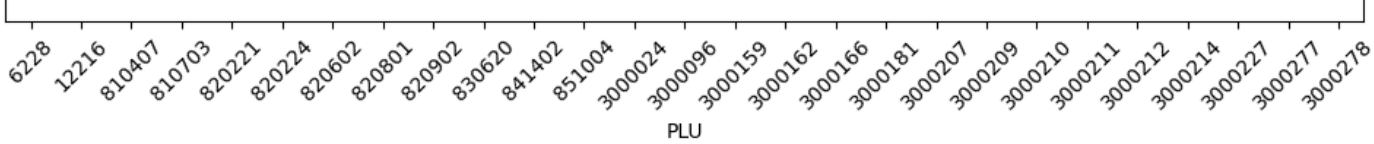

Box Plot for ReceivedQuantity
ReceivedQuantity
175
150
125
100
75
50
25
0
−25
6228
12216
810407
810703
820221
820224
820602
820801
820902
830620
841402
851004
3000024
3000096
3000159
3000162
3000166
3000181
3000207
3000209
3000210
3000211
3000212
3000214
3000227
3000277
3000278
PLU


Box Plot for SoldQuantity
SoldQuantity
80
60
40
20
0


PLU


Box Plot for EndQuantity
EndQuantity
5000
4000
3000
2000
1000
0
6228
12216
810407
810703
820221
820224
820602
820801
820902
830620
841402
851004
3000024
3000096
3000159
3000162
3000166
3000181
3000207
3000209
3000210
3000211
3000212
3000214
3000227
3000277
3000278
PLU


Box Plot for StockedOut
StockedOut
1.0
0.8
0.6
0.4
0.2
0.0



unique_plus

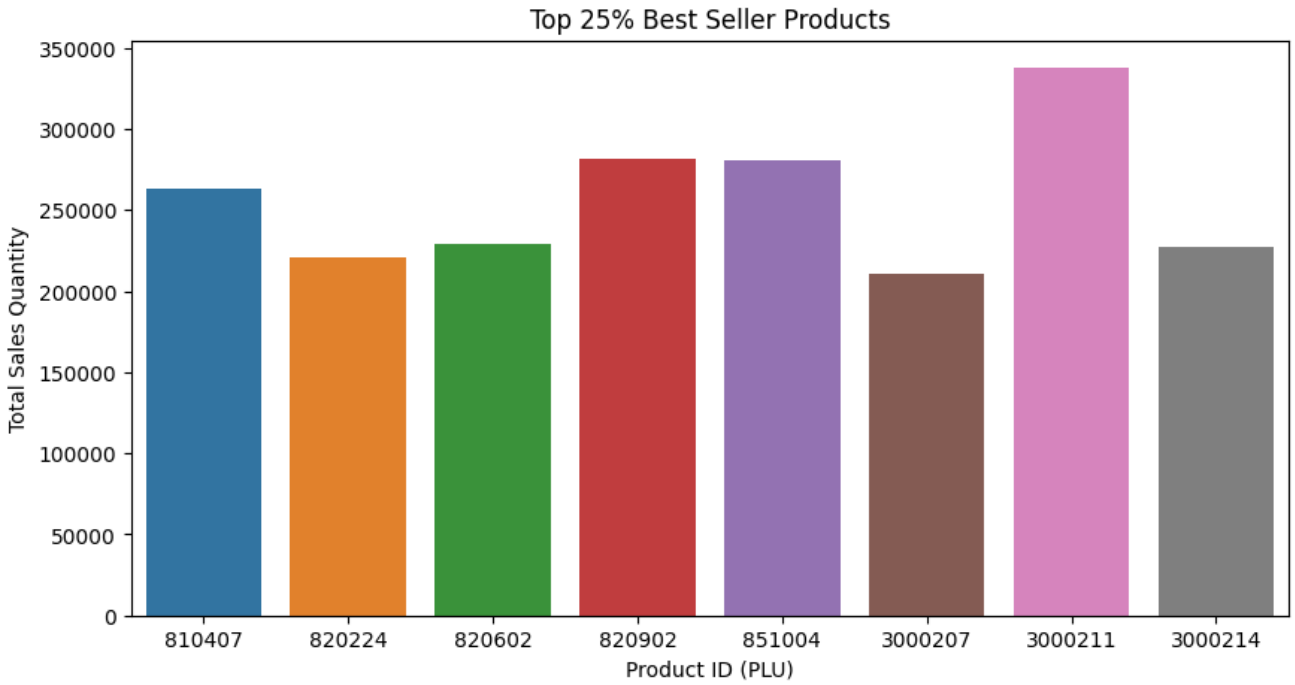
```
array([3000227,  830620,    6228,   12216, 3000277, 3000278,  851004,
        820602,  810407,  820801,  820902,  820224,  810703, 3000210,
       3000166, 3000096,  820221, 3000159, 3000162, 3000181, 3000207,
       3000209, 3000211, 3000212, 3000214, 3000024,  841402])
```

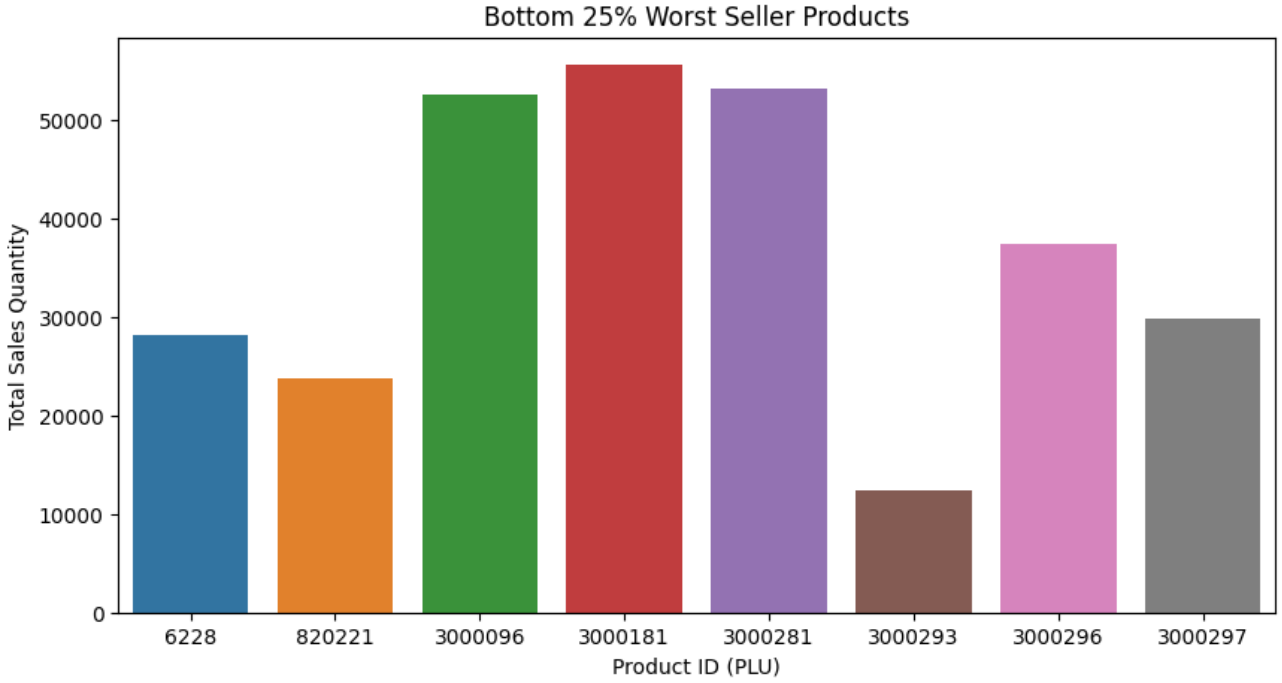
metrics

```
['ReceivedQuantity', 'SoldQuantity', 'EndQuantity', 'StockedOut']
```

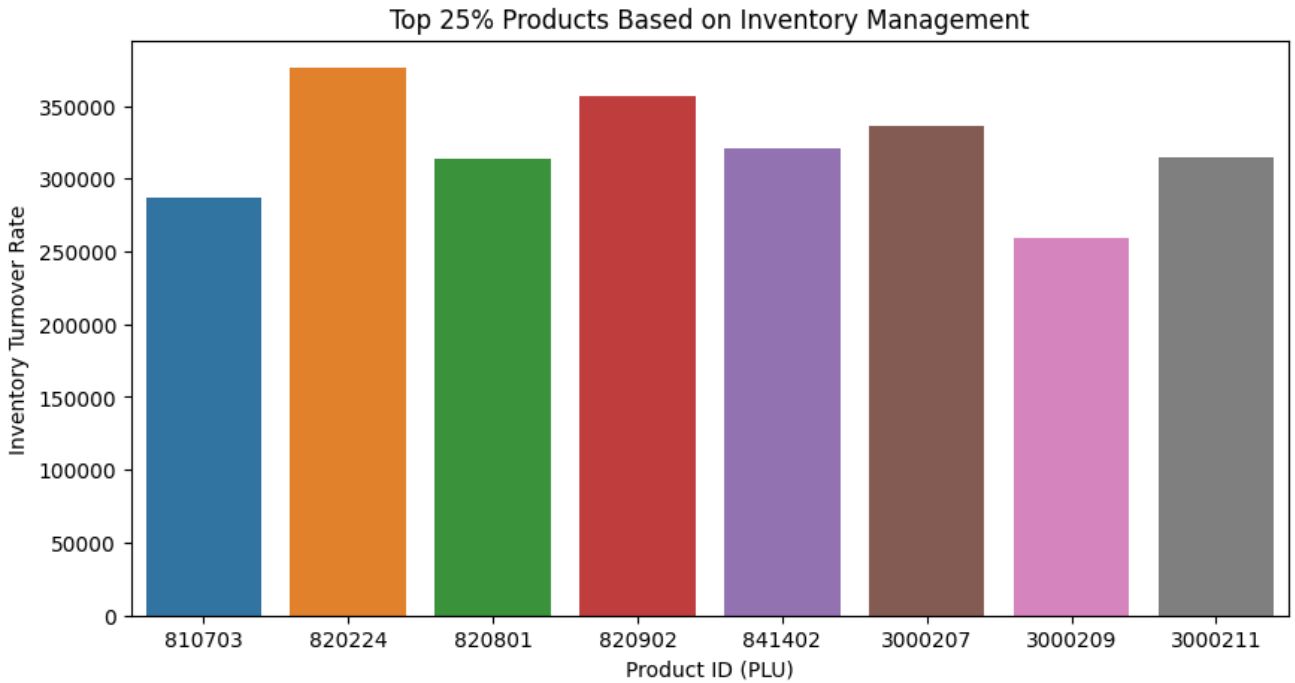
2. Best Seller and Worst Seller Products Analysis

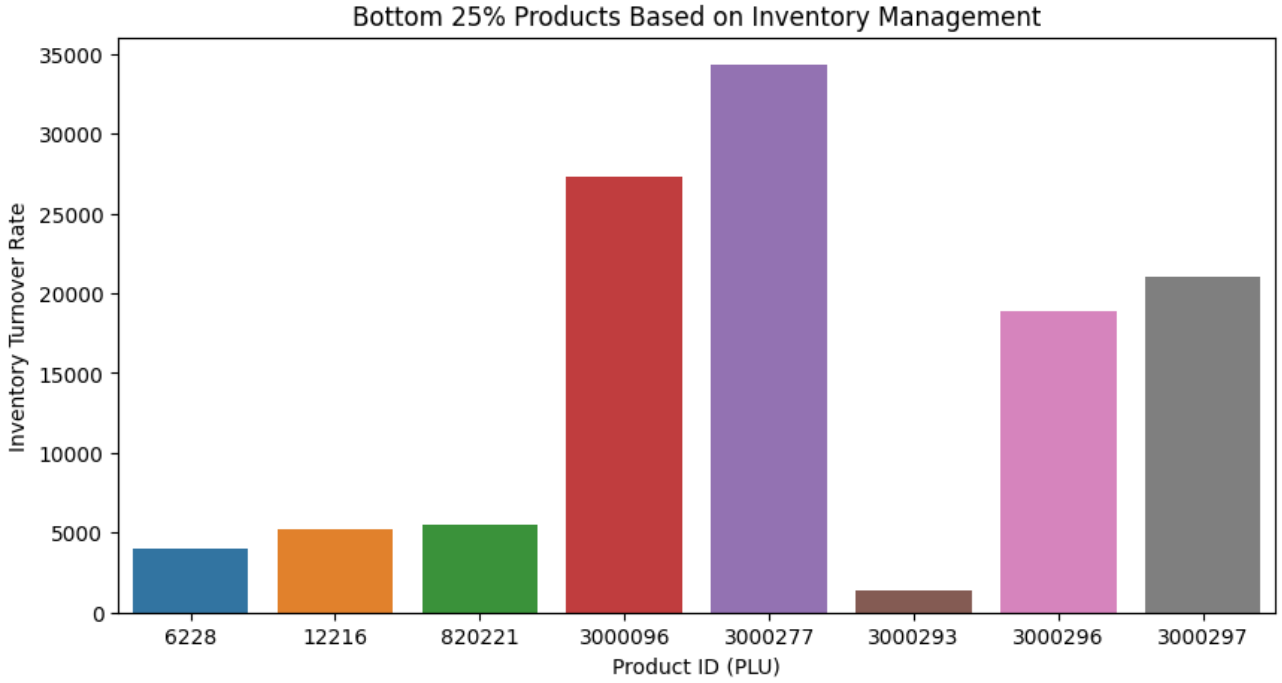
```
# Calculate total sales for each product
total_sales = data.groupby('PLU')['SoldQuantity'].sum()
top_25_percent = total_sales.quantile(0.75)
bottom_25_percent = total_sales.quantile(0.25)

# Top 25% products
top_products = total_sales[total_sales >= top_25_percent].sort_values(ascending=False)
plt.figure(figsize=(10, 5))
sns.barplot(x=top_products.index, y=top_products.values)
plt.title('Top 25% Best Seller Products')
plt.xlabel('Product ID (PLU)')
plt.ylabel('Total Sales Quantity')
plt.show()

# Bottom 25% products
bottom_products = total_sales[total_sales <= bottom_25_percent].sort_values()
plt.figure(figsize=(10, 5))
sns.barplot(x=bottom_products.index, y=bottom_products.values)
plt.title('Bottom 25% Worst Seller Products')
plt.xlabel('Product ID (PLU)')
plt.ylabel('Total Sales Quantity')
plt.show()
```


Top 25% Best Seller Products
Total Sales Quantity
350000
300000
250000
200000
150000
100000
50000
0
810407
820224
820602
820902
851004
3000207
3000211
3000214
Product ID (PLU)


Bottom 25% Worst Seller Products
Total Sales Quantity
50000
40000
30000
20000
10000
0
6228
820221
3000096
3000181
3000281
3000293
3000296
3000297
Product ID (PLU)

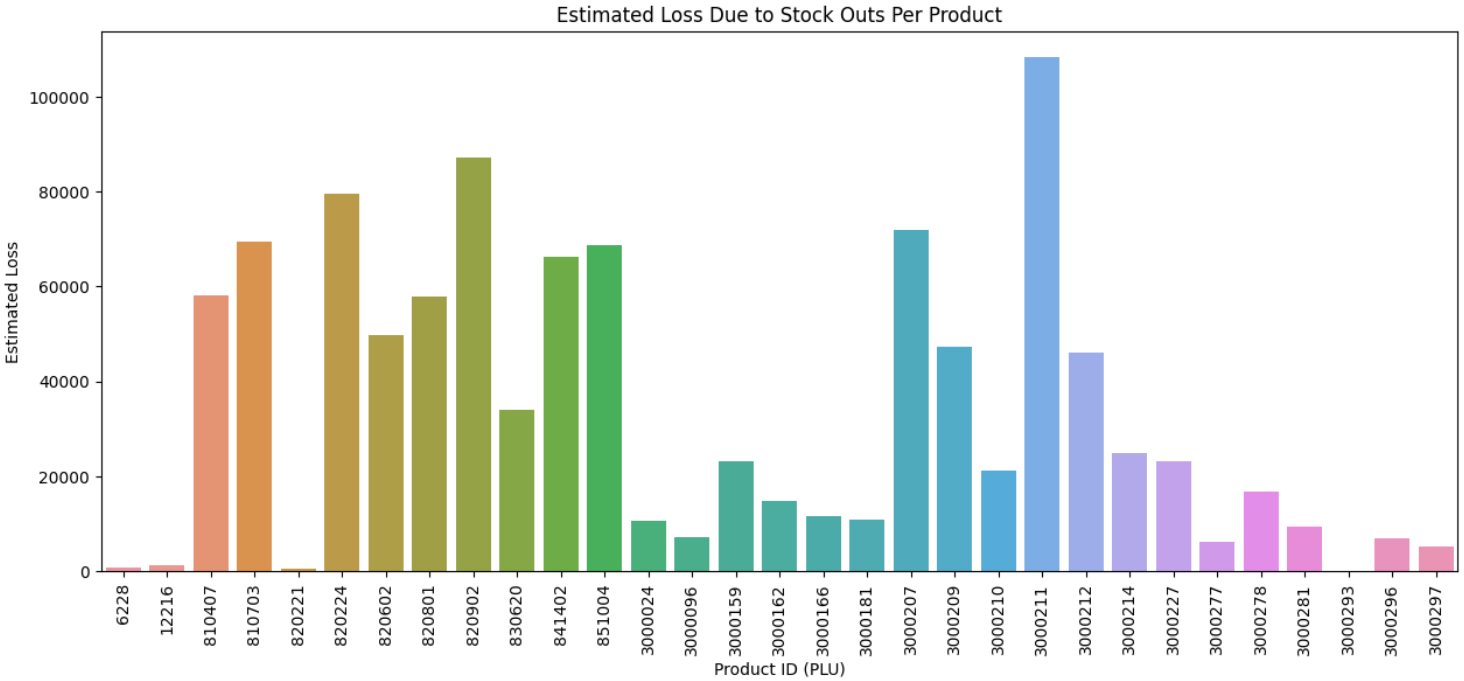
```
# Assuming 'EndQuantity' is the ending inventory
avg_inventory = data.groupby('PLU')['EndQuantity'].mean()
inventory_turnover = total_sales / avg_inventory

top_25_percent_inventory = inventory_turnover.quantile(0.75)
bottom_25_percent_inventory = inventory_turnover.quantile(0.25)

# Top 25% products based on inventory turnover
top_inventory_products = inventory_turnover[inventory_turnover >= top_25_percent_inventory].sort_values(ascending=False)
plt.figure(figsize=(10, 5))
sns.barplot(x=top_inventory_products.index, y=top_inventory_products.values)
plt.title('Top 25% Products Based on Inventory Management')
plt.xlabel('Product ID (PLU)')
plt.ylabel('Inventory Turnover Rate')
plt.show()

# Bottom 25% products based on inventory turnover
bottom_inventory_products = inventory_turnover[inventory_turnover <= bottom_25_percent_inventory].sort_values()
plt.figure(figsize=(10, 5))
sns.barplot(x=bottom_inventory_products.index, y=bottom_inventory_products.values)
plt.title('Bottom 25% Products Based on Inventory Management')
plt.xlabel('Product ID (PLU)')
plt.ylabel('Inventory Turnover Rate')
plt.show()
```





Task 4: Identify Stock Outs and Estimate Loss of Sales

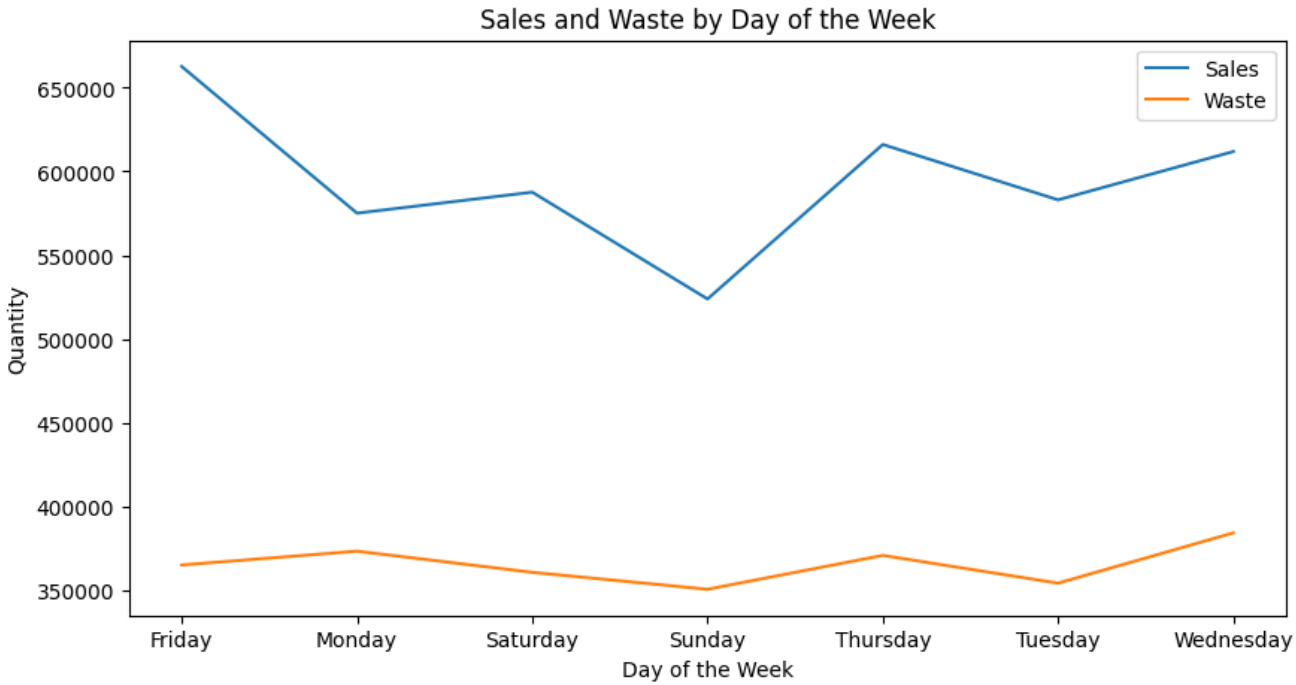
```
import pandas as pd


# Calculate the average sales per product
avg_sales_per_product = data.groupby('PLU')['SoldQuantity'].mean()

# Function to estimate loss of sales
def estimate_loss(row, avg_sales):
    if row['StockedOut'] == 1:  # Assuming 'StockedOut' is a binary column
        # Estimate loss as 75% of average sales of the product
        return avg_sales.get(row['PLU'], 0) * 0.75
    else:
        return 0

# Apply the function to estimate sales loss
data['EstimatedLoss'] = data.apply(lambda row: estimate_loss(row, avg_sales_per_product), axis=1)



# Plotting the estimated loss per product
plt.figure(figsize=(15, 6))
sns.barplot(x="PLU", y="EstimatedLoss", data=data.groupby('PLU')['EstimatedLoss'].sum().reset_index())
plt.title('Estimated Loss Due to Stock Outs Per Product')
plt.xlabel('Product ID (PLU)')
plt.ylabel('Estimated Loss')
plt.xticks(rotation=90)
plt.show()
```



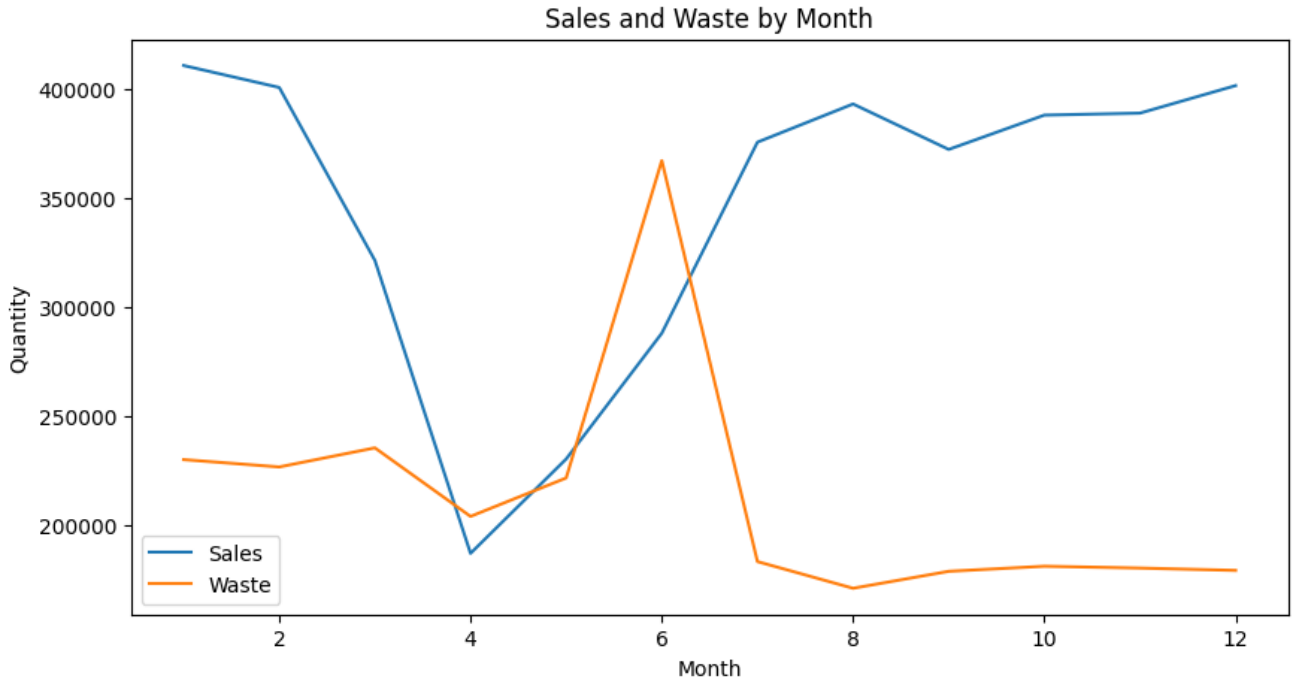
Task 5: Graphical Representation of Sales and Inventory Waste

```
# Feature Engineering
data['BusinessDate'] = pd.to_datetime(data['BusinessDate'])
data['Weekday'] = data['BusinessDate'].dt.day_name()
data['IsWeekend'] = data['Weekday'].isin(['Saturday', 'Sunday']).astype(int)
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Sales by day of the week
weekday_sales = data.groupby('Weekday')['SoldQuantity'].sum()

# Inventory waste by day of the week
weekday_waste = data.groupby('Weekday')['EndQuantity'].sum()  # Assuming 'EndQuantity' is a measure of waste

plt.figure(figsize=(10, 5))
sns.lineplot(data=weekday_sales, label='Sales')
sns.lineplot(data=weekday_waste, label='Waste')
plt.title('Sales and Waste by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Quantity')
plt.legend()
plt.show()
```



```
# Sales and waste by month
data['Month'] = data['BusinessDate'].dt.month
monthly_sales = data.groupby('Month')['SoldQuantity'].sum()
monthly_waste = data.groupby('Month')['EndQuantity'].sum()

plt.figure(figsize=(10, 5))
sns.lineplot(data=monthly_sales, label='Sales')
sns.lineplot(data=monthly_waste, label='Waste')
plt.title('Sales and Waste by Month')
plt.xlabel('Month')
plt.ylabel('Quantity')
plt.legend()
plt.show()
```

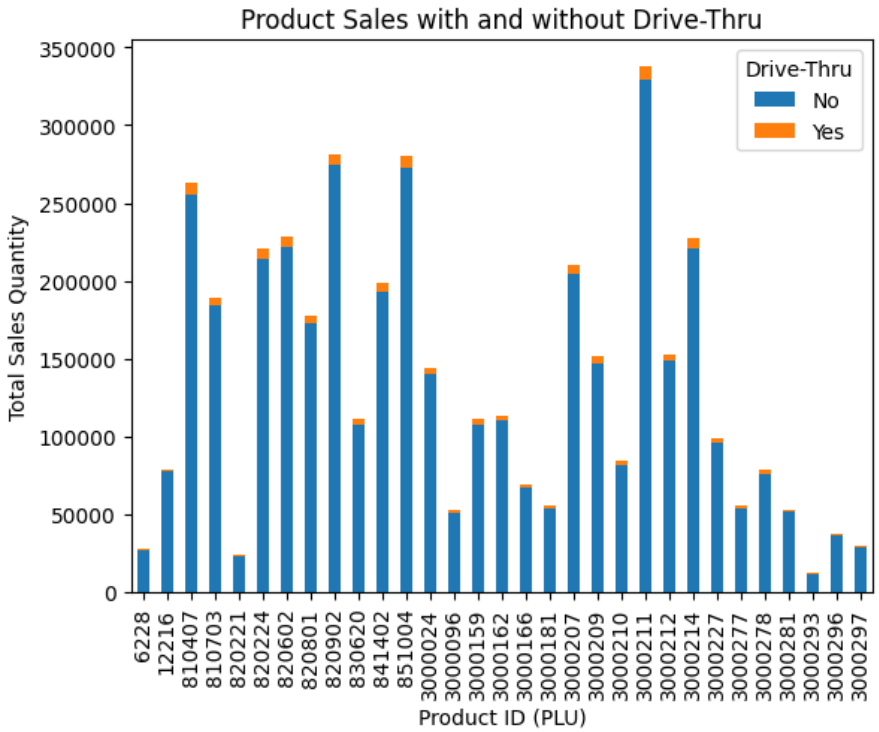


```
'''
# Sales by temperature and weather condition
# Assuming 'Temperature' and 'WeatherCondition' are in your dataset
sns.scatterplot(x='Temperature', y='SoldQuantity', hue='WeatherCondition', data=data)
plt.title('Sales by Temperature and Weather Condition')
plt.xlabel('Temperature')
plt.ylabel('Sales Quantity')
plt.show()
'''
```

```
'\n# Sales by temperature and weather condition\n# Assuming 'Temperature' and 'WeatherCondition' are in your dataset\nsns.scatte
rplot(x='Temperature', y='SoldQuantity', hue='WeatherCondition', data=data)\nplt.title('Sales by Temperature and Weather Conditi
on')\nplt.xlabel('Temperature')\nplt.ylabel('Sales Quantity')\nplt.show()\n'
```

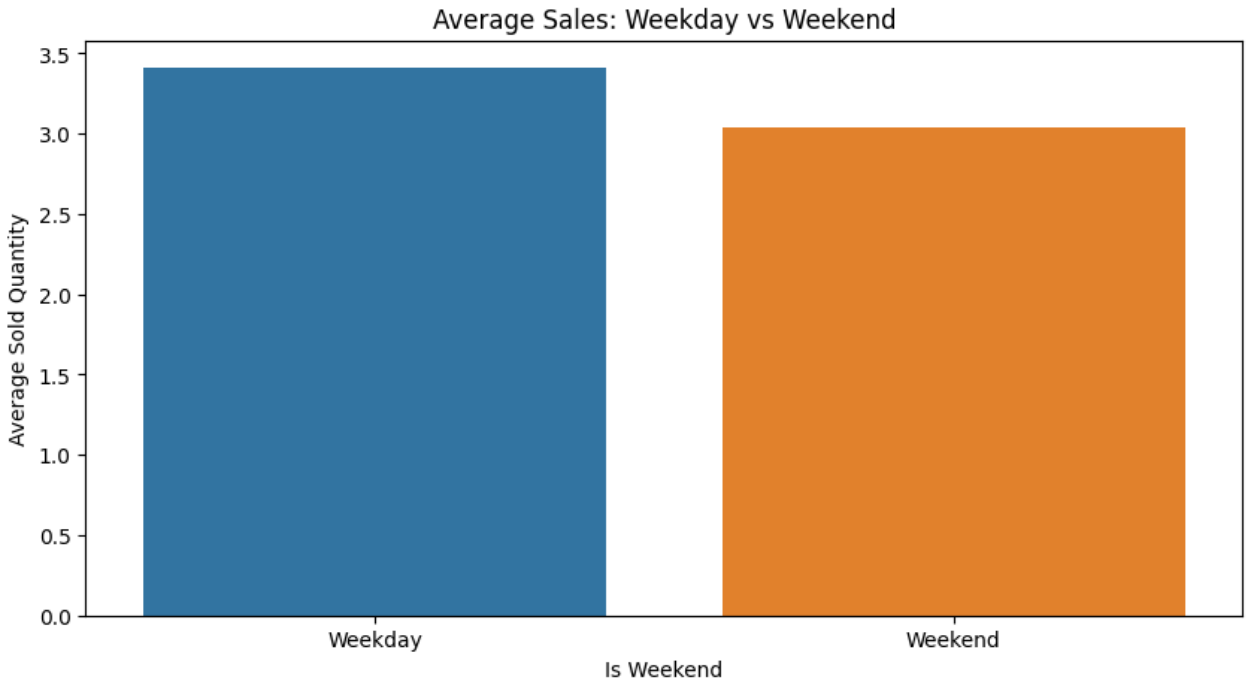
Task 6: Impact of Drive-Thru Feature on Sales

```
drive_thru_store_ids = [18, 117, 332, 132]  # Example store IDs with a drive-thru

# Create a new column 'HasDriveThru' in the DataFrame
data['HasDriveThru'] = data['StoreID'].isin(drive_thru_store_ids).astype(int)

# Now you can perform the analysis
# Group by PLU and 'HasDriveThru', then sum 'SoldQuantity'
drive_thru_sales = data.groupby(['PLU', 'HasDriveThru'])['SoldQuantity'].sum().unstack()

# Plotting the results
import matplotlib.pyplot as plt

drive_thru_sales.plot(kind='bar', stacked=True)
plt.title('Product Sales with and without Drive-Thru')
plt.xlabel('Product ID (PLU)')
plt.ylabel('Total Sales Quantity')
plt.legend(title='Drive-Thru', labels=['No', 'Yes'])
plt.show()
```



Task 7: Investigate the Impact of Weekday/Weekends

```
# Convert 'BusinessDate' to datetime and extract the day of the week
data['BusinessDate'] = pd.to_datetime(data['BusinessDate'])
data['DayOfWeek'] = data['BusinessDate'].dt.dayofweek  # Monday=0, Sunday=6
data['IsWeekend'] = data['DayOfWeek'].isin([5, 6]).astype(int)  # Mark weekends

# Analysis: Compare sales on weekdays vs weekends
weekday_sales = data.groupby('IsWeekend')['SoldQuantity'].mean()

# Plotting
plt.figure(figsize=(10, 5))
sns.barplot(x=weekday_sales.index, y=weekday_sales.values)
plt.title('Average Sales: Weekday vs Weekend')
plt.xlabel('Is Weekend')
plt.ylabel('Average Sold Quantity')
plt.xticks(ticks=[0, 1], labels=['Weekday', 'Weekend'])
plt.show()
```

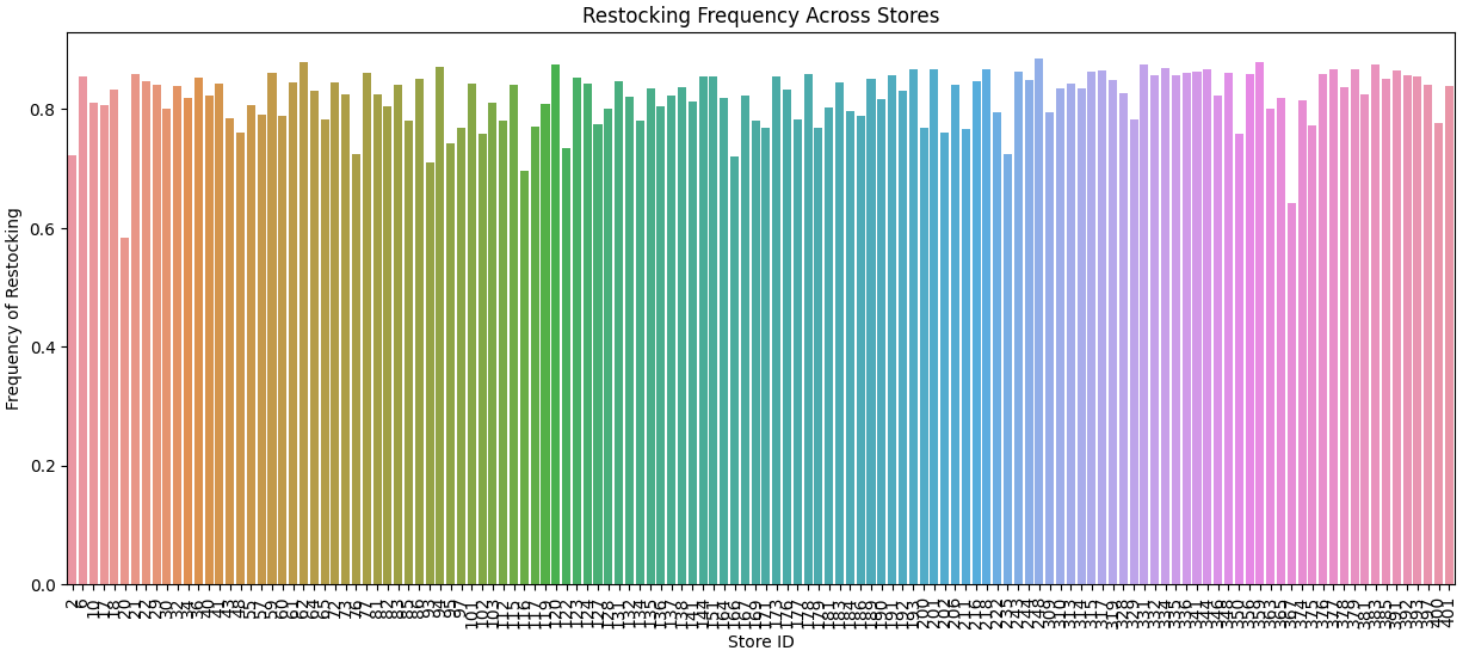



```
weekday_sales
```

```
IsWeekend
0    3.410453
1    3.038013
Name: SoldQuantity, dtype: float64
```

8. Identifying Stocking Patterns Across Multiple Stores

```
# Identify restocking days
data['IsRestocked'] = data['ReceivedQuantity'] > 0

# Restocking frequency analysis
restocking_frequency = data.groupby('StoreID')['IsRestocked'].mean()

# Plotting
plt.figure(figsize=(15, 6))
sns.barplot(x=restocking_frequency.index, y=restocking_frequency.values)
plt.title('Restocking Frequency Across Stores')
plt.xlabel('Store ID')
plt.ylabel('Frequency of Restocking')
plt.xticks(rotation=90)
plt.show()
```



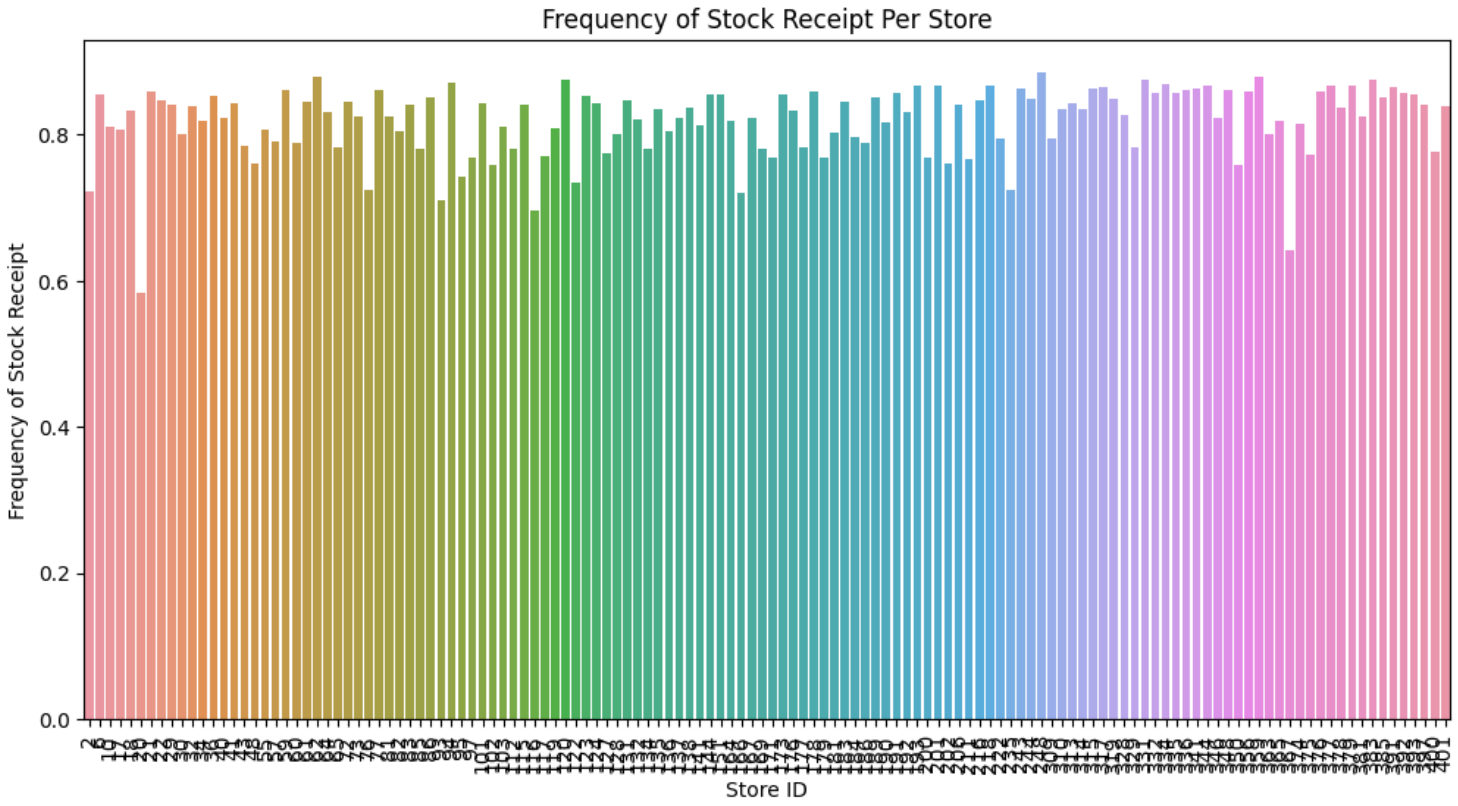

```
print(restocking_frequency)
```

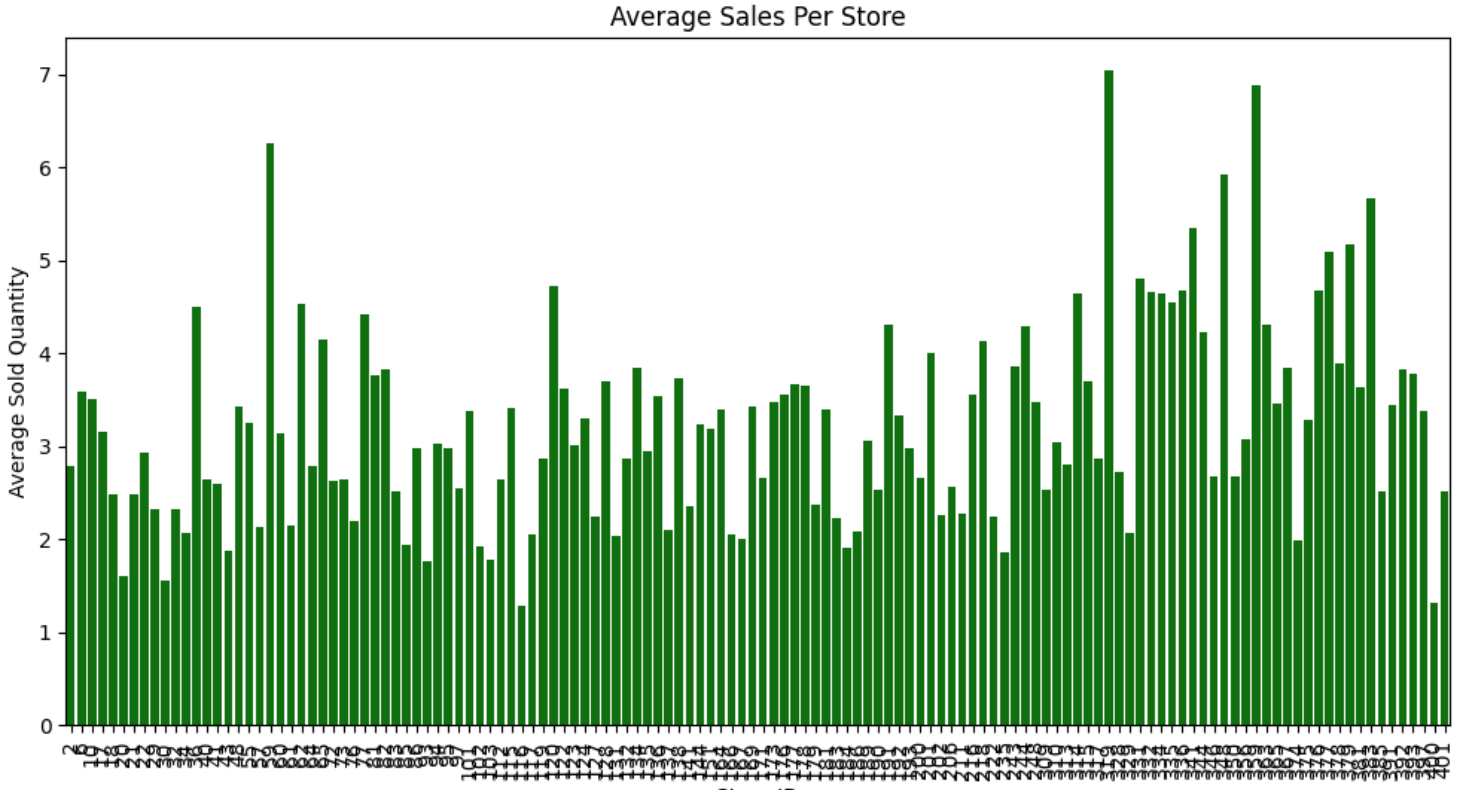
```
StoreID
2      0.722366
6      0.855063
```

```
10     0.810733
17     0.806448
18     0.833155
         ...
392    0.857459
393    0.855513
397    0.840574
400    0.775758
401    0.839815
Name: IsRestocked, Length: 132, dtype: float64
```

Task 9: Drawing Conclusions and Recommendations

```
# Convert 'BusinessDate' to datetime
data['BusinessDate'] = pd.to_datetime(data['BusinessDate'])

# Adding a column to identify if stock was received
data['StockReceived'] = data['ReceivedQuantity'] > 0

# Analyzing the frequency of receiving stock per store
restock_freq = data.groupby('StoreID')['StockReceived'].mean()

# Analyzing average sold quantity per store
avg_sales = data.groupby('StoreID')['SoldQuantity'].mean()

# Plotting the frequency of stock receipt per store
plt.figure(figsize=(12, 6))
sns.barplot(x=restock_freq.index, y=restock_freq.values)
plt.title('Frequency of Stock Receipt Per Store')
plt.xlabel('Store ID')
plt.ylabel('Frequency of Stock Receipt')
plt.xticks(rotation=90)
plt.show()

# Plotting the average sales per store
plt.figure(figsize=(12, 6))
sns.barplot(x=avg_sales.index, y=avg_sales.values, color='green')
plt.title('Average Sales Per Store')
plt.xlabel('Store ID')
plt.ylabel('Average Sold Quantity')
plt.xticks(rotation=90)
plt.show()
```


Frequency of Stock Receipt Per Store
Frequency of Stock Receipt
0.0
0.2
0.4
0.6
0.8
Store ID


Average Sales Per Store
Average Sold Quantity
0
1
2
3
4
5
6
7

Store ID

Part B

[ ] ↳ *7 cells hidden*

Section 2

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
import tensorflow as tf
from tensorflow.keras import layers

# Load the data
file_path = '/content/Coffee_Stores_Data.csv'
data = pd.read_csv(file_path)

# Assuming you want to generate synthetic 'SoldQuantity' data
features_to_use = ['PLU', 'ReceivedQuantity', 'EndQuantity', 'LatestOrder', 'StockedOut']  # example features
target_feature = 'SoldQuantity'

# Extract features and target
X = data[features_to_use]
y = data[target_feature]

# Preprocess the data
# One-hot encoding for categorical variables and scaling for numerical variables
preprocessor = ColumnTransformer(
    transformers=[
        ('num', MinMaxScaler(), features_to_use),
        # Add ('cat', OneHotEncoder(), ['categorical_feature']) for categorical features
    ])

X_processed = preprocessor.fit_transform(X)
y_processed = MinMaxScaler().fit_transform(y.values.reshape(-1, 1))


def build_generator():
    model = tf.keras.Sequential()
    model.add(layers.Dense(units=128, input_dim=100))  # Input dimension based on noise
    model.add(layers.LeakyReLU(alpha=0.01))
    model.add(layers.Dense(units=y_processed.shape[1], activation='linear'))  # Adjust the output unit to match target
    return model

def build_discriminator():
    model = tf.keras.Sequential()
    model.add(layers.Dense(units=128, input_dim=y_processed.shape[1]))  # Input dimension based on the target
    model.add(layers.LeakyReLU(alpha=0.01))
    model.add(layers.Dense(units=1, activation='sigmoid'))
    return model
```

```
generator = build_generator()
# Build and compile the discriminator
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(), metrics=['accuracy'])
# The discriminator is not trainable when training the generator within the GAN model
discriminator.trainable = False



def build_gan(generator, discriminator):
    model = tf.keras.Sequential()
    model.add(generator)
    model.add(discriminator)
    return model

# Build and compile the GAN
gan = build_gan(generator, discriminator)
gan.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam())
# Other GAN building and compiling steps remain the same

import numpy as np

def train_gan(generator, discriminator, gan, X, epochs, batch_size):
    half_batch = batch_size // 2
    for epoch in range(epochs):
        print('Epoch {}/{}'.format(epoch + 1, epochs))
        # ---------------------
        # Train Discriminator
        # ---------------------
        # Select a random half batch of real data
        idx = np.random.randint(0, X.shape[0], half_batch)
        real_data = X[idx]

        # Generate fake data
        noise = np.random.normal(0, 1, (half_batch, 100))
        fake_data = generator.predict(noise)

        # Train the discriminator
        d_loss_real = discriminator.train_on_batch(real_data, np.ones((half_batch, 1)))
        d_loss_fake = discriminator.train_on_batch(fake_data, np.zeros((half_batch, 1)))
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # ---------------------
```

```
        # Train Generator
        # ---------------------
        noise = np.random.normal(0, 1, (batch_size, 100))
        valid_y = np.array([1] * batch_size)
        g_loss = gan.train_on_batch(noise, valid_y)

# Example training call (adjust epochs and batch_size according to your needs)
train_gan(generator, discriminator, gan, y_processed, epochs=100, batch_size=32)
```

```
Epoch 1/100
1/1 [==============================] - 0s 225ms/step
Epoch 2/100
1/1 [==============================] - 0s 23ms/step
Epoch 3/100
1/1 [==============================] - 0s 23ms/step
Epoch 4/100
1/1 [==============================] - 0s 25ms/step
Epoch 5/100
1/1 [==============================] - 0s 22ms/step
Epoch 6/100
1/1 [==============================] - 0s 25ms/step
Epoch 7/100
1/1 [==============================] - 0s 26ms/step
Epoch 8/100
1/1 [==============================] - 0s 35ms/step
Epoch 9/100
1/1 [==============================] - 0s 27ms/step
Epoch 10/100
1/1 [==============================] - 0s 27ms/step
Epoch 11/100
1/1 [==============================] - 0s 26ms/step
Epoch 12/100
1/1 [==============================] - 0s 26ms/step
Epoch 13/100
1/1 [==============================] - 0s 22ms/step
Epoch 14/100
1/1 [==============================] - 0s 25ms/step
Epoch 15/100
1/1 [==============================] - 0s 27ms/step
Epoch 16/100
1/1 [==============================] - 0s 28ms/step
Epoch 17/100
1/1 [==============================] - 0s 30ms/step
Epoch 18/100
1/1 [==============================] - 0s 27ms/step
Epoch 19/100
1/1 [==============================] - 0s 28ms/step
Epoch 20/100
1/1 [==============================] - 0s 27ms/step
Epoch 21/100
1/1 [==============================] - 0s 24ms/step
Epoch 22/100
1/1 [==============================] - 0s 24ms/step
Epoch 23/100
1/1 [==============================] - 0s 26ms/step
Epoch 24/100
1/1 [==============================] - 0s 24ms/step
```

```
Epoch 25/100
1/1 [==============================] - 0s 23ms/step
Epoch 26/100
1/1 [==============================] - 0s 27ms/step
Epoch 27/100
1/1 [==============================] - 0s 28ms/step
Epoch 28/100
1/1 [==============================] - 0s 23ms/step
Epoch 29/100
1/1 [==============================] - 0s 23ms/step
```

```
# Generate synthetic data
def generate_synthetic_data(generator, number_of_samples):
    noise = tf.random.normal([number_of_samples, 100])
    synthetic_data = generator.predict(noise)
    return synthetic_data

# Generate synthetic data using the trained generator
synthetic_data = generate_synthetic_data(generator, 100)
```

```
4/4 [==============================] - 0s 3ms/step
```

```
synthetic_data
```

```
array([[ 0.20147592],
       [ 0.6652994 ],
       [ 0.1240507 ],
       [-0.12002311],
       [-0.588014  ],
       [ 0.01806863],
       [-0.14651367],
       [ 0.28712457],
       [ 0.66215605],
       [-0.04439475],
       [ 0.42722663],
       [ 0.03107124],
       [ 0.77178144],
       [ 0.2280587 ],
       [ 0.10480224],
       [ 0.28962642],
       [-0.29719165],
       [ 0.1292921 ],
       [ 0.10802691],
       [ 0.12236952],
       [ 0.1749058 ],
       [ 0.28470114],
       [-0.4656247 ],
       [ 0.04638933],
       [-0.01456238],
       [ 0.27590418],
       [ 0.23706186],
       [ 0.07489609],
       [ 0.3304588 ],
       [ 0.32669562],
       [-0.37442994],
```

```
[ 0.15425205],
[ 0.4787405 ],
[-0.49970317],
[-0.20491123],
[ 0.6300963 ],
[-0.32671392],
[ 0.18759996],
[ 0.29153612],
[-0.6523856 ],
[-0.66838276],
[ 0.17988521],
[-0.48042238],
[-0.10386214],
[ 0.07031452],
[ 0.47996932],
[-0.08043755],
[ 0.49256957],
[ 0.23664975],
[ 0.116418  ],
[-0.21424586],
[-0.33269948],
[-0.00835059],
[ 0.06213003],
[ 1.0274593 ],
[ 0.28222936],
[-0.24673167],
[ 0.23614001],
```

Task 2

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer


# Feature Engineering
data['BusinessDate'] = pd.to_datetime(data['BusinessDate'])
data['Weekday'] = data['BusinessDate'].dt.day_name()
data['IsWeekend'] = data['Weekday'].isin(['Saturday', 'Sunday']).astype(int)
# Add more features like weather, temperature, etc.

# Select relevant features for prediction
features = ['PLU', 'ReceivedQuantity', 'EndQuantity', 'LatestOrder', 'StockedOut', 'Weekday', 'IsWeekend']  # Add your additio

# Define target variable
target = 'SoldQuantity'

# Preprocess the features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', MinMaxScaler(), ['ReceivedQuantity', 'EndQuantity', 'LatestOrder']),  # Numerical features
        ('cat', OneHotEncoder(), ['Weekday', 'IsWeekend'])  # Categorical features
    ])

X_processed = preprocessor.fit_transform(data[features])
y = data[target]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_processed, y, test_size=0.2, random_state=42)



from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Linear Regression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Predict and evaluate
y_pred = lin_reg.predict(X_test)
print(f"Linear Regression RMSE: {mean_squared_error(y_test, y_pred, squared=False)}")
```

```
Linear Regression RMSE: 1.9744577110904487
```

Task 4

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import xgboost as xgb
from sklearn.metrics import classification_report, confusion_matrix

# Random Forest with reduced complexity
rf = RandomForestRegressor(n_estimators=100, max_depth=10, n_jobs=-1)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)

# Gradient Boosting with reduced complexity
gb = GradientBoostingRegressor(n_estimators=100, max_depth=5)
gb.fit(X_train, y_train)
gb_pred = gb.predict(X_test)

# XGBoost with adjusted parameters
xg_reg = xgb.XGBRegressor(n_estimators=100, max_depth=5, learning_rate=0.1)
xg_reg.fit(X_train, y_train)
xg_pred = xg_reg.predict(X_test)

# Evaluate models (example for RMSE)
from sklearn.metrics import mean_squared_error

print("Random Forest RMSE:", mean_squared_error(y_test, rf_pred, squared=False))
print("Gradient Boosting RMSE:", mean_squared_error(y_test, gb_pred, squared=False))
print("XGBoost RMSE:", mean_squared_error(y_test, xg_pred, squared=False))
```

```
Random Forest RMSE: 1.7056931660917694
Gradient Boosting RMSE: 1.7059640190611112
XGBoost RMSE: 1.7078109147560363
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Function to evaluate regression models
def evaluate_model(name, true_values, predicted_values):
    rmse = mean_squared_error(true_values, predicted_values, squared=False)
    mae = mean_absolute_error(true_values, predicted_values)
    r2 = r2_score(true_values, predicted_values)
    print(f"Model: {name}")
    print(f"RMSE: {rmse}")
    print(f"MAE: {mae}")
    print(f"R-squared: {r2}\n")
    return rmse, mae, r2

# Evaluate Random Forest
rf_rmse, rf_mae, rf_r2 = evaluate_model("Random Forest", y_test, rf_pred)

# Evaluate Gradient Boosting
gb_rmse, gb_mae, gb_r2 = evaluate_model("Gradient Boosting", y_test, gb_pred)

# Evaluate XGBoost
xg_rmse, xg_mae, xg_r2 = evaluate_model("XGBoost", y_test, xg_pred)
```

```
Model: Random Forest
RMSE: 1.7056931660917694
```