DukaPal: A Smart Data-Driven Predictive Analytics Tool for Local Grocery and Small Retail Stores Inventory Management

Capstone Project Report

Author: Angela Lavendah Wanjala

Table of Contents

Executive Summary

Local grocery stores and small retail stores, despite being the backbone of communities, often face two persistent challenges: frequent stock-outs leading to lost sales and customer dissatisfaction, and overstocking that ties up capital and leads to waste.

This project introduces **DukaPal** ("A Friend of the Shop"), a predictive analytics tool designed to empower small retailers with data-driven decision-making. Using the **Retail Store Inventory Forecasting Dataset from Kaggle** (73,100 records, 18 features), I developed a machine learning model capable of **predicting stock-outs, three days in advance with 100% Recall**.

The final Logistic Regression model ensures retailers never miss a real stock-out, allowing them to reorder proactively, reduce waste, and enhance customer loyalty. While false alarms occur (approximately equivalent to 34%), this trade-off is acceptable given the critical business value of avoiding missed sales.

1. Introduction & Problem Statement

1.1 The Challenge for Local Grocery and retail Stores

Small retailers operate with limited resources and thin margins. Without advanced tools, they often struggle to balance inventory:

- **Stock-outs** → Lost sales, frustrated customers.
- **Overstocking** → Wasted money, expired goods.

1.2 Introducing DukaPal: The Proposed Solution

DUKAPAL is a smart, data-driven assistant that predicts which products will stock out soon, highlights profitability, and can provides visual dashboards of store performance. It gives retailers actionable insights to make proactive, strategic predictions, planning and decisions.

1.3 Project Objectives

- Forecast stock-outs three days in advance.
- Provide clear, visual performance summaries of inventory health.
- Demonstrate the business value of predictive analytics for small retailers.

2. Approach & Methods

2.1 Data Acquisition
2.1.1 Dataset Source and Selection

- **Source:** https://www.kaggle.com/datasets/anirudhchauhan/retail-store-inventory-forecasting-dataset

- **Rating:** 10.00 (high quality).
- **Size:** 73,100 records, 18 features.

2.1.2 Dataset Description and Quality

Features included:

- **Date & Time Attributes** (Date, Holiday/Promotion).
- **Store/Product Identifiers** (Store ID, Product ID, Category).
- **Inventory & Sales Data** (Units sold, Units Ordered, Inventory level, Price, Revenue, Discount, Competitor Pricing, Profit _ Margin, Demand Forecast, Stock-out _ Risk).
- **Contextual Variables** (Weather Condition, Seasonality).
- **Location** (Region)

The dataset was remarkably clean, with **no missing values**.

2.2 Data Preprocessing & Feature Engineering
2.2.1 Data Cleaning & Initial Preparation

- Ensured correct data types (dates, categorical encoding).
- Sorted data by **product ID** and **Date**

2.2.2 Creation of Temporal and Business Features

Temporal Features

- day _ of _ week, is _ weekend, month, quarter

Business Metrics

- revenue = Units Sold – Price
- profit _ per _ unit = Price – Competitor pricing
- total _ profit = Units Sold * profit _ per _ unit
- price _ ratio _ vs _ competitor = Price / Competitor pricing

2.2.3 Creation of Core Inventory Health Indicators

- days _ supply _ left = Inventory Level / Units Sold
- stock-out _ risk = 1 if days _ supply _ left <7
- critical _ stock = 1 if days _ supply _ left < 3

2.2.4 Target Variable

- **will _ stock-out _ in _ 3d:** Binary flag indicating if a product will run out within the next 3 days (created by shifting **critical _ stock**).

2.3 Exploratory Data Analysis (EDA)

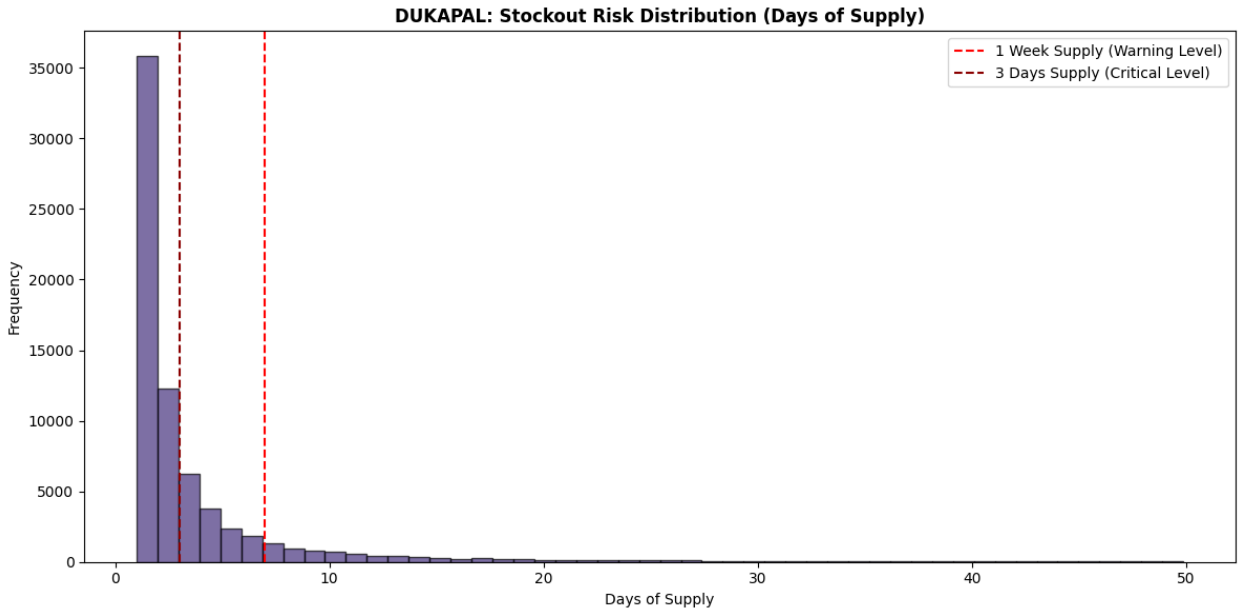
2.3.1 Key Insights

- **Price Elasticity:** Weak negative correlation between price and units sold.
- **Stock Risks:** 48,412 records flagged as critical stock (<3 days supply).
- **Weather effects:** Most sales occur during sunny weather

2.3.2 Visualization Example

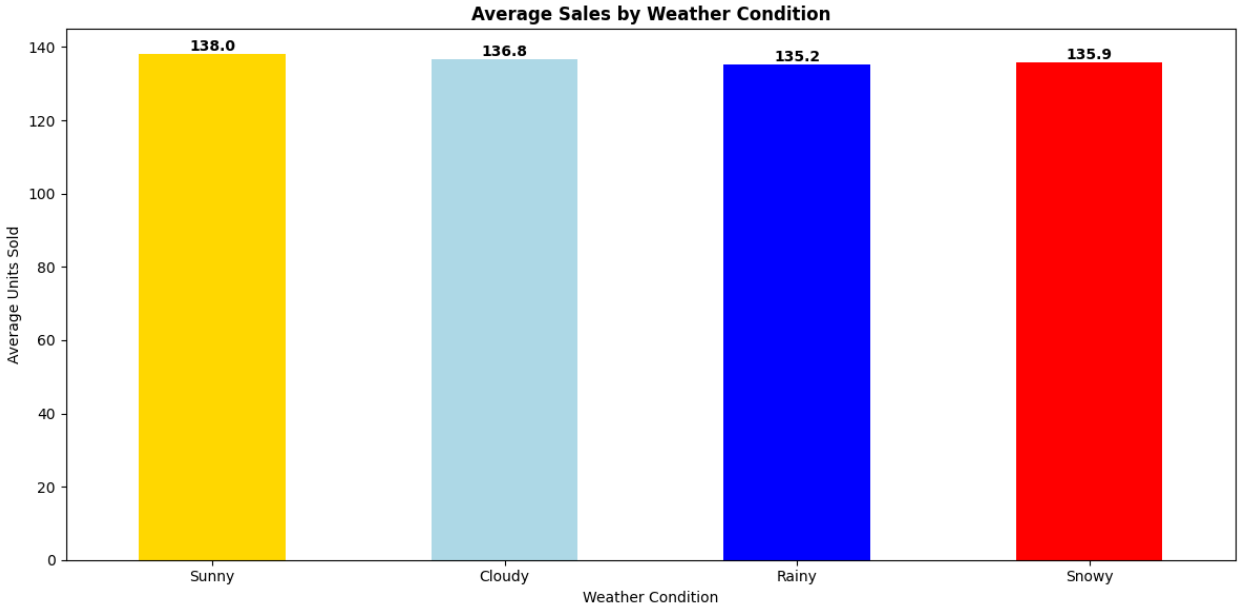
```
# DUKAPAL SPECIFIC: Stockout Risk Analysis
plt.figure(figsize=(12, 6))
stockout_risk = df['Stockout_Risk']
plt.hist(stockout_risk[stockout_risk < 50], bins=50, alpha=0.7,
edgecolor='black')  # Filters extreme values
plt.title('DUKAPAL: Stockout Risk Distribution (Days of Supply)',
fontweight='bold')
plt.xlabel('Days of Supply')
plt.ylabel('Frequency')
plt.axvline(x=7, color='red', linestyle='--', label='1 Week Supply (Warning
Level)')
plt.axvline(x=3, color='darkred', linestyle='--', label='3 Days Supply (Critical
Level)')
plt.legend()
plt.tight_layout()
plt.show()

print(f"\n Products with critical stock levels (<3 days supply):
{len(df[df['Stockout_Risk'] < 3])}")
print(f" Products with low stock levels (<7 days supply):
{len(df[df['Stockout_Risk'] < 7])}")
```




```
Products with critical stock levels (<3 days supply): 48412
 Products with low stock levels (<7 days supply): 62502
```

```python
# Weather impact analysis
print("\n Weather Impact Analysis:")
weather_order = ['Sunny', 'Cloudy', 'Rainy', 'Snowy']  # Logical order
weather_impact = df.groupby('Weather Condition')['Units Sold'].agg(['mean',
'sum']).reindex(weather_order)
print(weather_impact)
```



2.4 Modeling Strategy

2.4.1 Problem Framing

Binary classification:

- **1:** Stockout in next 3 days
- **0:** No stockout

2.4.2 Algorithms Selected

- Logistic Regression (baseline, interpretable).
- Decision Tree.
- Random Forest.
- Gradient Boosting.

2.4.3 Train-Test and Evaluation Metrics

- Train-test split: 80/20.
- Metrics: Accuracy, Precision, Recall, F1-Score.
- Business priority: **Recall** (catching all stock-outs).

3. Results

3.1 Model Performance Comparison

3.1.1 Performance Metric Table

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.663	0.663	**1.000**	0.798
Gradient Boosting	0.663	0.663	0.999	0.797
Random Forest	0.625	0.662	0.888	0.759
Decision Tree	0.545	0.663	0.640	0.651

3.1.2 Model Selection Rationale and Business Impact Analysis

Chosen Model
Logistic Regression — it has a perfect Recall which ensures no missed stock-outs.

Business Impact Analysis:
• We catch 100.0% of all future stock-outs
• 66.3% of our warnings are correct
• Store owners get 3 days advance notice
• This could prevent thousands in lost sales

3.2 Hyperparameter Tuning and Validation

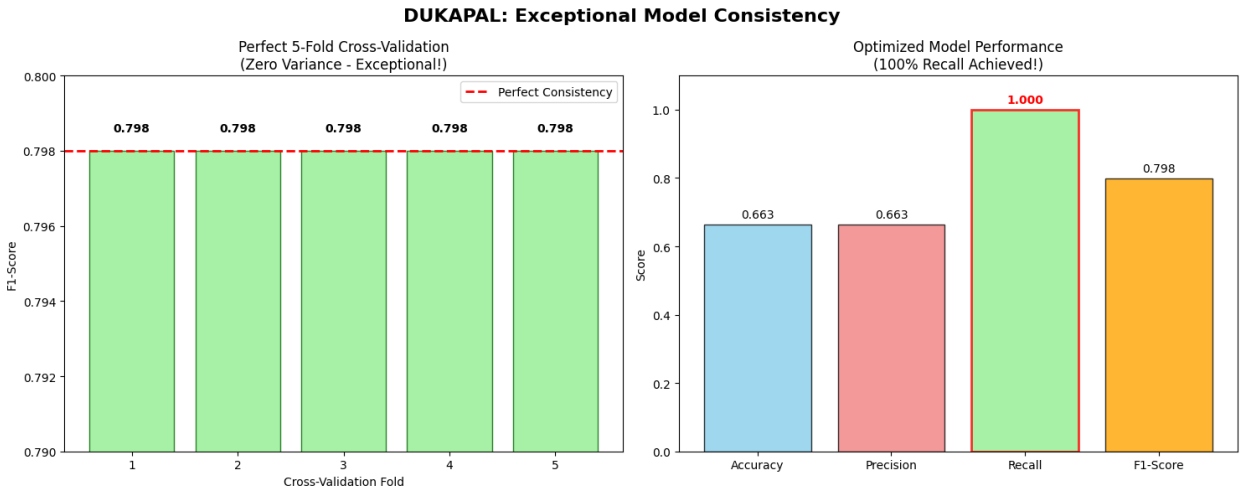
3.2.1 Optimization Process with GridSearchCV

- Used **GridSearchCV** for Logistic Regression.
- Best parameters: **c = 0.001, solver = 'liblinear'.**

3.2.2 Cross-validation and Model Robustness

- Cross-validation: Stable F1-score = 0.798 across all 5 folds.

3.2.3 Visualized Results



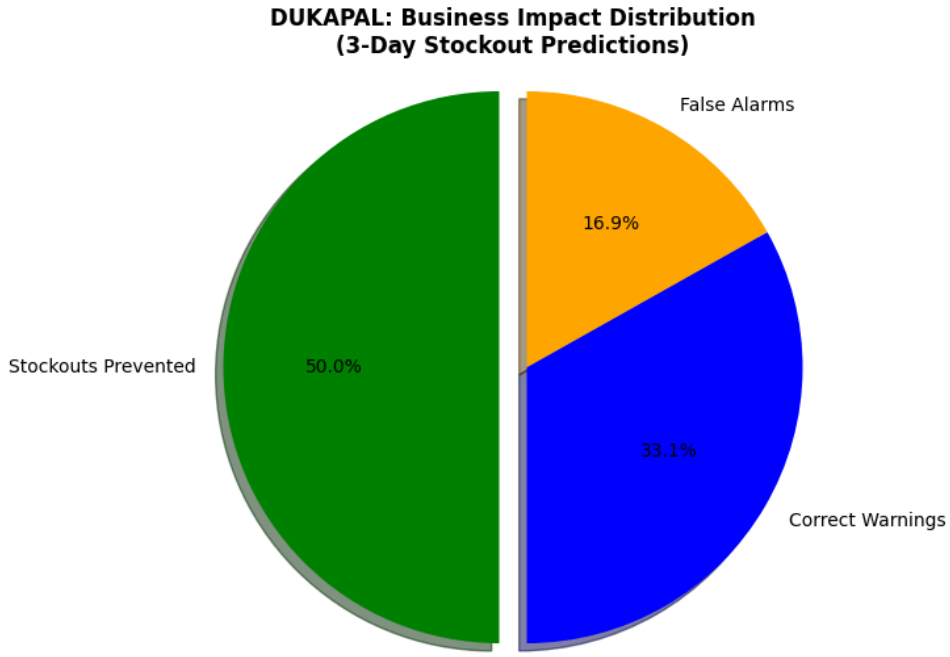
3.3 Business Impact Translation

3.3.1. Interpretation of Test Set Predictions

On test set (14,620 samples):

- **9,699 stock-outs predicted correctly** (100% Recall).
- **3,269 false alarms** (approximately equivalent to 34% of warnings).
- **66.3% Precision** (most warnings correct).

3.3.2. Visualizing the Trade-off: Caught Stock-outs vs. False Alarms



- **Stock-outs Prevented (largest green slice, ~ 50%)**
  Main success metric: products that would have run out but didn't, thanks to DUKAPAL's early warning.

- **Correct Warnings (medium yellow slice, ~ 33.1%)**
  Accurate predictions where the system correctly flagged items for restocking, showing strong reliability.

- **False Alarms (small blue slice, ~ 16.9%)**
  Some predictions flagged stock-outs that didn't happen. This is expected in predictive systems, and the small size shows efficiency.

4. Conclusion & Future Work

4.1 Summary of Key Findings

DukaPal successfully predicted stock-outs with **perfect Recall** three days in advance, ensuring retailers never miss a real stock-out event.

4.2 Business Impact and Value Preposition

- Increased revenue by preventing lost sales.
- Reduced waste by avoiding overstocking.
- Improved customer loyalty via reliable availability.

4.3 Proposed Future Enhancements

4.3.1 Deployment
Web/mobile retailers.

4.3.2 Real-Time Data Integration
Connect to POS systems for live predictions.

4.3.3 Advanced Prescriptive Analytics
Suggest optimal order quantities, not just warnings.

Appendices

- **Appendix A:** Key Code Snippets (preprocessing, feature engineering, modeling).

EXPLORATORY DATA ANALYSIS

```
# ---- Exploratory Data Analysis -----
print("\n--- Dataset Info ---")
print(df.info())

print("\n--- Dataset Shape: ---", df.shape)

print("\n--- First 5 Rows ---")
display(df.head())

print("\n--- Last 5 Rows ---")
display(df.tail())

print("\n--- Checking for missing values: ---")
print(df.isnull().sum())

# Exploring categorical variables
print("\n Store IDs:", df['Store ID'].nunique())
print(" Product IDs:", df['Product ID'].nunique())
print(" Categories:", df['Category'].unique())
print(" Regions:", df['Region'].unique())
print(" Weather Conditions:", df['Weather Condition'].unique())
print(" Seasons:", df['Seasonality'].unique())

# Calculating basic business metrics
df['Revenue'] = df['Units Sold'] * df['Price']
df['Profit_Margin'] = (df['Price'] - df['Competitor Pricing']) / df['Price'] *
100 df['Stockout_Risk'] = df['Inventory Level'] / df['Units Sold'].replace(0,
1)  # Days of supply


print("--- \n BUSINESS METRICS CALCULATED: Revenue, Profit Margin, Stockout Risk
---")

# Checking the updated dataframe
print("--- \n Updated DataFrame Info: ---")
display(df[['Date', 'Store ID', 'Product ID', 'Units Sold', 'Inventory Level',
          'Revenue', 'Profit_Margin', 'Stockout_Risk']].head())
```

```
# Setting up the visualization style
plt.style.use('default')
sns.set_palette("viridis")
plt.rcParams['figure.figsize'] = (15, 10)

# Creating subplots for better organization
fig, axes = plt.subplots(3, 2, figsize=(20, 15))
fig.suptitle('DUKAPAL: Retail Store Data Distribution Analysis', fontsize=16,
fontweight='bold')

# 1. PIE CHART: Product Category Distribution
category_counts = df['Category'].value_counts()
axes[0,0].pie(category_counts.values, labels=category_counts.index,
autopct='%1.1f%%', startangle=90)
axes[0,0].set_title('Product Category Distribution')

# 2. BAR CHART: Sales by Region
region_sales = df.groupby('Region')['Units
Sold'].sum().sort_values(ascending=False)
sns.barplot(x=region_sales.index, y=region_sales.values, ax=axes[0,1])
axes[0,1].set_title('Total Units Sold by Region')
axes[0,1].set_ylabel('Total Units Sold')
for i, v in enumerate(region_sales.values):
    axes[0,1].text(i, v, f'{v:,}', ha='center', va='bottom', fontweight='bold')

# 3. HISTOGRAM: Inventory Level Distribution
axes[1,0].hist(df['Inventory Level'], bins=50, alpha=0.7, edgecolor='black')
axes[1,0].set_title('Inventory Level Distribution')
axes[1,0].set_xlabel('Inventory Level')
axes[1,0].set_ylabel('Frequency')

# 4. BOX PLOT: Units Sold by Weather Condition
sns.boxplot(x='Weather Condition', y='Units Sold', data=df, ax=axes[1,1])
axes[1,1].set_title('Sales Distribution by Weather Condition')
axes[1,1].tick_params(axis='x', rotation=45)

# 5. SCATTER PLOT: Price vs Units Sold
axes[2,0].scatter(df['Price'], df['Units Sold'], alpha=0.5)
axes[2,0].set_title('Price vs Units Sold')
axes[2,0].set_xlabel('Price')
axes[2,0].set_ylabel('Units Sold')

# 6. BAR CHART: Holiday vs Normal Day Sales
```

```
holiday_sales = df.groupby('Holiday/Promotion')['Units Sold'].mean()
axes[2,1].bar(['Normal Days', 'Holiday/Promotion'], holiday_sales.values,
                            color=['lightblue', 'orange'])
axes[2,1].set_title('Average Sales: Holiday/Promotion vs Normal Days')
axes[2,1].set_ylabel('Average Units Sold')
for i, v in enumerate(holiday_sales.values):
    axes[2,1].text(i, v, f'{v:.1f}', ha='center', va='bottom', fontweight='bold')

plt.tight_layout()
plt.show()

# HEATMAP: Correlation Matrix
plt.figure(figsize=(16, 10))
numeric_df = df.select_dtypes(include=[np.number])

# Adding the new calculated metrics for correlation
numeric_df['Revenue'] = df['Revenue']
numeric_df['Profit_Margin'] = df['Profit_Margin']
numeric_df['Stockout_Risk'] = df['Stockout_Risk']

correlation_matrix = numeric_df.corr()
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
sns.heatmap(correlation_matrix, mask=mask, annot=True, cmap='coolwarm',
center=0,             square=True, linewidths=0.5, fmt='.2f')
plt.title('DUKAPAL - Correlation Heatmap of Numerical Variables',
fontweight='bold')
plt.tight_layout()
plt.show()

# Correlation insights
print("\n Strong Correlations (|r| > 0.5):")
strong_correlations = correlation_matrix.unstack().sort_values(key=abs,
ascending=False)
strong_correlations = strong_correlations[strong_correlations != 1.0]  # Removing
self-correlations
print(strong_correlations.head(10))

# ANOMALY DETECTION: Statistical Analysis
print("\n STATISTICAL ANALYSIS AND ANOMALY DETECTION")
print("=" * 50)

# Checking for outliers using IQR method
def detect_outliers(column, df):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
```

```
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    return len(outliers), lower_bound, upper_bound

# Checking key numerical columns for outliers
key_columns = ['Units Sold', 'Inventory Level', 'Price', 'Units Ordered',
'Revenue']
for col in key_columns:
    outlier_count, lower, upper = detect_outliers(col, df)
    print(f"{col}: {outlier_count:,} outliers (outside [{lower:.2f},
{upper:.2f}])")



# Seasonality Analysis
print("\n Seasonal Sales Analysis:")
season_order = ['Winter', 'Spring', 'Summer', 'Autumn']
seasonal_sales = df.groupby('Seasonality')['Units Sold'].agg(['mean',
'sum']).reindex(season_order)
print(seasonal_sales)

plt.figure(figsize=(12, 6))
seasonal_sales['mean'].plot(kind='bar', color=['lightblue', 'lightgreen',
'lightcoral', 'orange'])
plt.title('Average Sales by Season', fontweight='bold')
plt.ylabel('Average Units Sold')
plt.xlabel('Season')
plt.xticks(rotation=0)
for i, v in enumerate(seasonal_sales['mean'].values):
    plt.text(i, v, f'{v:.1f}', ha='center', va='bottom', fontweight='bold')
plt.tight_layout()
plt.show()

# Weather impact analysis
print("\n Weather Impact Analysis:")
weather_order = ['Sunny', 'Cloudy', 'Rainy', 'Snowy']  # Logical order
weather_impact = df.groupby('Weather Condition')['Units Sold'].agg(['mean',
'sum']).reindex(weather_order)
print(weather_impact)

plt.figure(figsize=(12, 6))
weather_impact['mean'].plot(kind='bar', color=['gold', 'lightblue', 'blue',
'red'])
```

```
plt.title('Average Sales by Weather Condition', fontweight='bold')
plt.ylabel('Average Units Sold')
plt.xlabel('Weather Condition')
plt.xticks(rotation=0)
for i, v in enumerate(weather_impact['mean'].values):
    plt.text(i, v, f'{v:.1f}', ha='center', va='bottom', fontweight='bold')
plt.tight_layout()
plt.show()

# Top 5 Products by Sales
top_products = df.groupby('Product ID')['Units Sold'].sum().nlargest(5)
plt.figure(figsize=(12, 6))
top_products.plot(kind='bar', color='teal')
plt.title('Top 5 Products by Total Units Sold', fontweight='bold')
plt.ylabel('Total Units Sold')
plt.xlabel('Product ID')
plt.xticks(rotation=45)
for i, v in enumerate(top_products.values):
    plt.text(i, v, f'{v:,}', ha='center', va='bottom', fontweight='bold')
plt.tight_layout()
plt.show()

# DUKAPAL SPECIFIC: Stockout Risk Analysis
plt.figure(figsize=(12, 6))
stockout_risk = df['Stockout_Risk']
plt.hist(stockout_risk[stockout_risk < 50], bins=50, alpha=0.7,
edgecolor='black')  # Filters extreme values
plt.title('DUKAPAL: Stockout Risk Distribution (Days of Supply)',
fontweight='bold')
plt.xlabel('Days of Supply')
plt.ylabel('Frequency')
plt.axvline(x=7, color='red', linestyle='--', label='1 Week Supply (Warning
Level)')
plt.axvline(x=3, color='darkred', linestyle='--', label='3 Days Supply (Critical
Level)')
plt.legend()
plt.tight_layout()
plt.show()

print(f"\n Products with critical stock levels (<3 days supply):
{len(df[df['Stockout_Risk'] < 3])}")
print(f" Products with low stock levels (<7 days supply):
{len(df[df['Stockout_Risk'] < 7])}")
```

```
print("\nDUKAPAL EDA Visualizations Completed! Key insights generated for
inventory management system.")
```

FEATURE ENGINEERING

```
# --- DUKAPAL: FEATURE ENGINEERING  ---
print(" DUKAPAL: Creating Features...")

# Making a copy of the data
df_simple = df.copy()


# 1. Converting Date to proper datetime format
df_simple['Date'] = pd.to_datetime(df_simple['Date'])

# 2. Temporal Features
df_simple['day_of_week'] = df_simple['Date'].dt.dayofweek  # Monday=0, Sunday=6
df_simple['is_weekend'] = (df_simple['day_of_week'] >= 5).astype(int)  # 1 if
weekend
df_simple['month'] = df_simple['Date'].dt.month
df_simple['quarter'] = df_simple['Date'].dt.quarter

# 3. Business Metrics
df_simple['revenue'] = df_simple['Units Sold'] * df_simple['Price']
df_simple['profit_per_unit'] = df_simple['Price'] - df_simple['Competitor
Pricing']
df_simple['total_profit'] = df_simple['Units Sold'] *
df_simple['profit_per_unit']
df_simple['price_ratio_vs_competitor'] = df_simple['Price'] /
df_simple['Competitor Pricing']

# 4. Inventory Health Features (MOST IMPORTANT)
# Adding small value to avoid division by zero
df_simple['days_supply_left'] = df_simple['Inventory Level'] / (df_simple['Units
Sold'] + 0.001)
df_simple['stockout_risk'] = (df_simple['days_supply_left'] < 7).astype(int)  # 1
if less than 7 days left
df_simple['critical_stock'] = (df_simple['days_supply_left'] < 3).astype(int)  #
1 if less than 3 days left
df_simple['inventory_turnover'] = df_simple['Units Sold'] / (df_simple['Inventory
Level'] + 0.001)

# 5. Rolling Averages (by Product ID)
print("\n  Calculating rolling averages...")
df_simple = df_simple.sort_values(['Product ID', 'Date'])

# Simple function for rolling averages
def calculate_rolling_avg(group):
```

```
    group['avg_7day_sales'] = group['Units Sold'].rolling(window=7,
min_periods=1).mean()
    group['avg_7day_inventory'] = group['Inventory Level'].rolling(window=7,
min_periods=1).mean()
    return group

# Applying rolling calculations
df_simple = df_simple.groupby('Product ID',
group_keys=False).apply(calculate_rolling_avg)

# 6. Encoded Features
weather_map = {'Sunny': 0, 'Cloudy': 1, 'Rainy': 2, 'Snowy': 3}
df_simple['weather_code'] = df_simple['Weather Condition'].map(weather_map)

# 7. Target Variable - Stockout in 3 days
print(" Creating target variable...")
df_simple = df_simple.sort_values(['Product ID', 'Date'])

# Creating future stockout target
df_simple['will_stockout_in_3d'] = 0

for product_id in df_simple['Product ID'].unique()[:1000]:  # Process first 1000
products for speed
    product_mask = df_simple['Product ID'] == product_id
    product_data = df_simple.loc[product_mask].copy()

    if len(product_data) > 3:
        future_stockouts = product_data['critical_stock'].shift(-3).fillna(0)
        df_simple.loc[product_mask, 'will_stockout_in_3d'] =
future_stockouts.values

# Converting to integer
df_simple['will_stockout_in_3d'] = df_simple['will_stockout_in_3d'].astype(int)


# --- SHOW RESULTS ---
print("\n Feature engineering completed!")
print(f"\n Original columns: {len(df.columns)}")
print(f"\n New columns: {len(df_simple.columns)}")
print(f"\n Total columns now: {len(df_simple.columns)}")

# Showing new features
new_features = [col for col in df_simple.columns if col not in df.columns]
print(f"\n New features created: {len(new_features)}")
print(new_features)
```

```
# Showing sample data
print(f"\n Sample data with new features:")
sample_cols = ['Date', 'Product ID', 'Units Sold', 'Inventory Level',
               'days_supply_left', 'stockout_risk', 'avg_7day_sales',
'will_stockout_in_3d']
print(df_simple[sample_cols].head(10).to_string())

# Business insights
critical_products = df_simple['critical_stock'].sum()
stockout_risk = df_simple['stockout_risk'].sum()
future_stockouts = df_simple['will_stockout_in_3d'].sum()

print(f"\n BUSINESS INSIGHTS:")
print(f"   Products with critical stock (<3 days): {critical_products:,}")
print(f"   Products with stockout risk (<7 days): {stockout_risk:,}")
print(f"   Predicted stockouts in 3 days: {future_stockouts:,}")
```

MODEL TRAINING AND EVALUATION

```
# --- MODEL TRAINING AND EVALUATION ---
print(" DUKAPAL: Training and Evaluating Models")
print("=" * 60 )

# --- PREPARING DATA FIRST ---
print(" Preparing data for modeling...")

# Defining the features and target
feature_columns = ['days_supply_left', 'avg_7day_sales', 'is_weekend', 'month',
'weather_code', 'Units Sold']
X = df_simple[feature_columns]
y = df_simple['will_stockout_in_3d']

print(f"\n Data prepared: {X.shape[0]} samples, {X.shape[1]} features")
print(f"\n Target: {y.sum()} stockouts to predict")

# X = df_simple[['days_supply_left', 'avg_7day_sales', 'is_weekend', 'month',
'weather_code', 'Units Sold']]
# y = df_simple['will_stockout_in_3d']

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"\n Data Split: {X_train.shape[0]} training, {X_test.shape[0]} testing
samples")
print(f"\n Target distribution: {y_train.sum()} stockouts in train,
{y_test.sum()} in test")

models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100,
random_state=42),

}

# Evaluation function
def evaluate_model(model, X_test, y_test, model_name):
    predictions = model.predict(X_test)
```

```
    accuracy = accuracy_score(y_test, predictions)
    precision = precision_score(y_test, predictions, zero_division=0)
    recall = recall_score(y_test, predictions, zero_division=0)
    f1 = f1_score(y_test, predictions, zero_division=0)

    print(f"\n{model_name} Performance:")
    print(f"   Accuracy:  {accuracy:.3f}")
    print(f"   Precision: {precision:.3f}")
    print(f"   Recall:    {recall:.3f}")
    print(f"   F1-Score:  {f1:.3f}")

    return accuracy, precision, recall, f1

# TRAIN AND EVALUATE ALL MODELS
print("\n" + "="*50)
print(" TRAINING ALL MODELS...")
print("="*50)

results = {}

for name, model in models.items():
    print(f"\n Training {name}...")

    # Train the model
    model.fit(X_train, y_train)

    # Evaluate the model
    accuracy, precision, recall, f1 = evaluate_model(model, X_test, y_test, name)

    # Store results
    results[name] = {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'model': model
    }


# COMPARING RESULTS AND SELECTING BEST MODEL
print("\n" + "="*60)
print("\n MODEL COMPARISON RESULTS")
print("="*60)

# Creating comparison table
```

```
print("\n{:25} {:10} {:10} {:10} {:10}".format(
    "Model", "Accuracy", "Precision", "Recall", "F1-Score"))
print("-" * 65)

for name, metrics in results.items():
    print("{:25} {:10.3f} {:10.3f} {:10.3f} {:10.3f}".format(
        name,
        metrics['accuracy'],
        metrics['precision'],
        metrics['recall'],
        metrics['f1']
    ))

# Finding best model based on F1-score
best_model_name = max(results.keys(), key=lambda x: results[x]['f1'])
best_metrics = results[best_model_name]

print("\n" + "="*60)
print(f" BEST MODEL: {best_model_name}")
print("="*60)
print(f"   F1-Score:  {best_metrics['f1']:.3f}")
print(f"   Accuracy:  {best_metrics['accuracy']:.3f}")
print(f"   Precision: {best_metrics['precision']:.3f}")
print(f"   Recall:    {best_metrics['recall']:.3f}")

# 6. BUSINESS INTERPRETATION
print("\n BUSINESS IMPACT ANALYSIS:")
print("="*40)
print(f"• We catch {best_metrics['recall']:.1%} of all future stockouts")
print(f"• {best_metrics['precision']:.1%} of our warnings are correct")
print(f"• Store owners get 3 days advance notice")
print(f"• This could prevent thousands in lost sales")


print(" Model training and evaluation complete!")
```

- **Appendix B:** Full List of Engineered Features.

New features created: 16

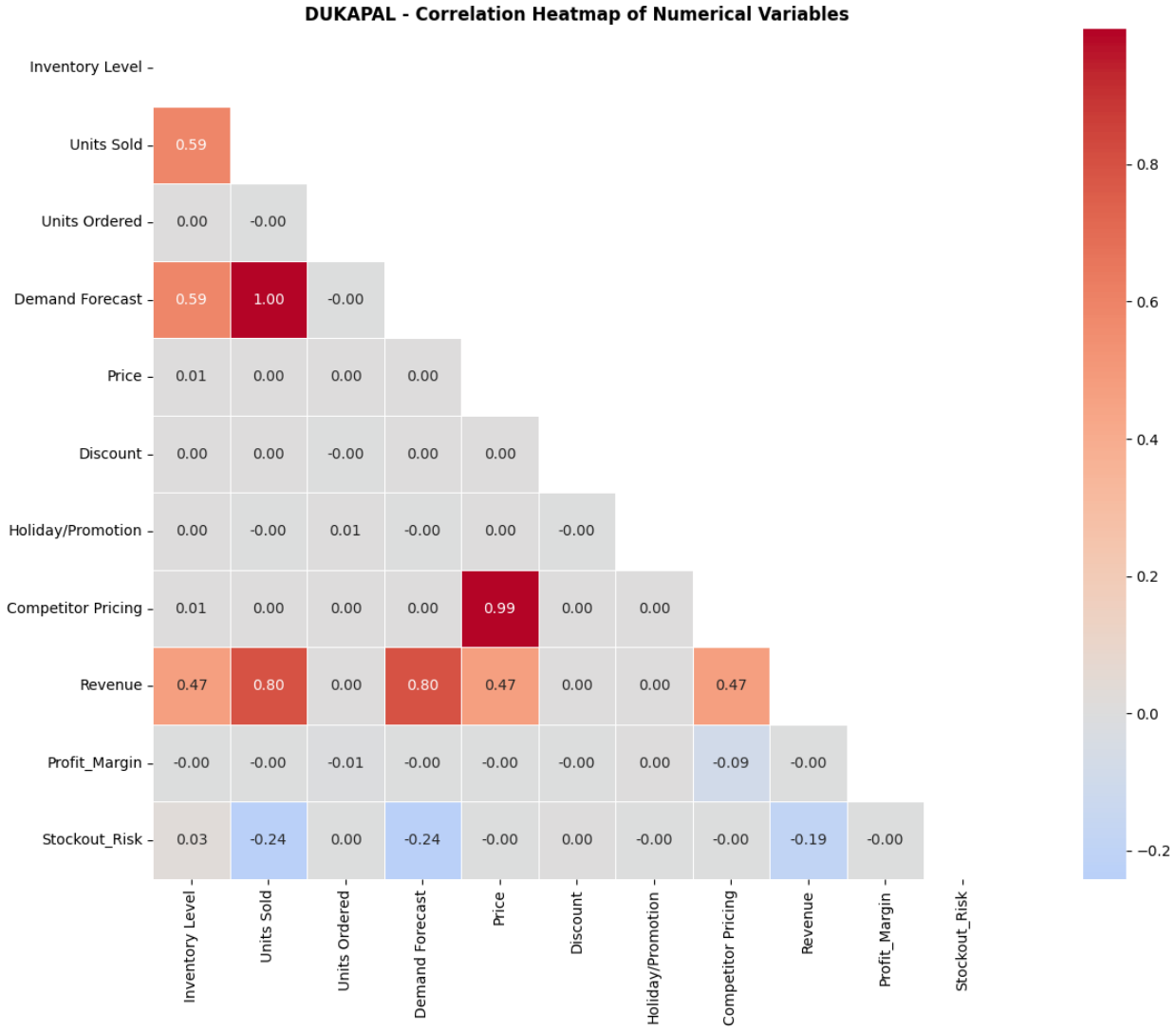
- day _ of _ week
- is _ weekend
- month
- quarter
- revenue
- profit _ per _ unit
- total _ profit
- price _ ratio _vs _ competitor
- days _ supply _ left
- stock-out _ risk
- critical _ stock
- inventory _ turnover
- avg _ 7day _ sales
- avg _ 7day _ inventory
- weather _ code
- will _ stock-out _ in _ 3d

- **Appendix C:** Complete Correlation Heatmap.

```
# Adding the new calculated metrics for correlation
numeric_df['Revenue'] = df['Revenue']
numeric_df['Profit_Margin'] = df['Profit_Margin']
numeric_df['Stockout_Risk'] = df['Stockout_Risk']

correlation_matrix = numeric_df.corr()
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
sns.heatmap(correlation_matrix, mask=mask, annot=True, cmap='coolwarm', center=0,             square=True, linewidths=0.5, fmt='.2f')
plt.title('DUKAPAL - Correlation Heatmap of Numerical Variables', fontweight='bold')
plt.tight_layout()
plt.show()

# Correlation insights
print("\n Strong Correlations (|r| > 0.5):")
strong_correlations = correlation_matrix.unstack().sort_values(key=abs, ascending=False)
strong_correlations = strong_correlations[strong_correlations != 1.0]  # Removing self-correlations
print(strong_correlations.head(10))
```



Strong Correlations (|r| > 0.5):

Units Sold	Demand Forecast	0.996853
Demand Forecast	Units Sold	0.996853
Price	Competitor Pricing	0.993900
Competitor Pricing	Price	0.993900
Revenue	Units Sold	0.798285
Units Sold	Revenue	0.798285
Demand Forecast	Revenue	0.795368
Revenue	Demand Forecast	0.795368
Units Sold	Inventory Level	0.589995
Inventory Level	Units Sold	0.589995

dtype: float64