Digital Egypt Pioneers

SALES FORECASTING

FINAL PROJECT REPORT

Contents

Sales Forecasting Model

1. Introduction

This project focuses on building and deploying a sales forecasting model that predicts product demand. The objective is to support data-driven decision-making for inventory management, demand planning, marketing strategies, and staffing.

2. Problem Definition

Retail and supply chain businesses often face issues in anticipating future product demand, leading to overstock or stockouts. This project aims to build a machine learning model that accurately forecasts sales based on historical data to solve this challenge.

3. Data Exploration

- **Source**: Historical sales data including fields such as Date, Product Code, Order Demand, Warehouse, Product Category, etc.
- **Cleaning**: Missing values were handled, and demand values were normalized.
- **Visualization**: Time-series trends, seasonal patterns, and demand distributions were analyzed.

(21128, 5)

Statistics

	Order_Demand
count	2.112700e+04
mean	1.453929e+04
std	6.273185e+04
min	1.000000e+00
25%	2.000000e+02
50%	2.000000e+03
75%	1.000000e+04
max	4.000000e+06

Info about the columns

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21128 entries, 0 to 21127
Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	Product_Code	21128 non-null	object
1	Warehouse	21128 non-null	object
2	Product_Category	21128 non-null	object
3	Date	21127 non-null	object
4	Order_Demand	21127 non-null	float64

dtypes: float64(1), object(4)
memory usage: 825.4+ KB
None

Unique Values

'Product_Code': 928 unique values
'Warehouse': 3 unique values
'Product_Category': 27 unique values
'Date': 334 unique values
'Order_Demand': 713 unique valu

Missing Values

Product_Code	0
Warehouse	0
Product_Category	0
Date	1
Order_Demand	1

dtype: int64

4. Feature Engineering

- Extracted time-based features: Year, Month
- Aggregated demand by product, warehouse, and category
- One-hot encoded categorical variables
- Created lag features and rolling averages to capture seasonality and trends

```
[ ]  # Add new features
     data['Year'] = data['Date'].dt.year
     data['Month'] = data['Date'].dt.month
     data['Day'] = data['Date'].dt.day
     data['Weekday'] = data['Date'].dt.weekday

[ ]  # Lag and Rolling Features
     for lag in [1, 7, 30]:
         data[f'Lag_{lag}'] = data['Order_Demand'].shift(lag)
     for window in [7, 30]:
         data[f'Rolling_{window}'] = data['Order_Demand'].rolling(window=window).mean()

[ ]  # External Factors (simulated for demo purposes)
     np.random.seed(42)
     data['Weather_Index'] = np.random.normal(loc=0.5, scale=0.1, size=len(data))  # 0 to 1 index
     promotions = np.random.choice([0, 1], size=len(data), p=[0.8, 0.2])  # 20% chance of promotion
     data['Promotion'] = promotions
     data['Econ_Indicator'] = np.random.normal(loc=0.6, scale=0.15, size=len(data))
```

5. Model Development

- **Model Used**: XGBoost Regressor
- **Training**: Model trained on features Year, Month
- **Evaluation**:
  - RMSE: calculated on test data
  - R^2 Score: calculated to evaluate explained variance
- **Actual value which is calculated**
  - RMSE: 6659709.63
  - R^2: 0.97
  - Success Rate: 64.29%
- **Prediction**: Model forecasts future demand for each product

```
last_date = data['Date'].max()
future_dates = pd.date_range(start=last_date + pd.offsets.MonthBegin(1), periods=6, freq='MS')
future_data = pd.DataFrame({
    'Year': future_dates.year,
    'Month': future_dates.month})
# Load the saved XGBoost model
model = joblib.load("xgb_demand_model.pkl")

future_data['Predicted_Order_Demand'] = model.predict(future_data[['Year', 'Month']])
print(future_data)
```

```
   Year  Month  Predicted_Order_Demand
0  2013      2            125495128.0
1  2013      3            125495128.0
2  2013      4            125495128.0
3  2013      5            125495128.0
4  2013      6            125495128.0
5  2013      7            125495128.0
```

```
# Train model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Predict
y_pred = model.predict(X_test)
# Evaluation
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
# Custom success rate
threshold = 0.1 * y_test.mean()
success_rate = (abs(y_test - y_pred) <= threshold).mean() * 100
print(f"MAE: {mae:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"R²: {r2:.2f}")
print(f"Custom Success Rate (Within {threshold:.2f}): {success_rate:.2f}%")
```

```
MAE: 6047063.66
RMSE: 7314846.08
R²: 0.96
Custom Success Rate (Within 6864146.92): 64.29%
```

6. Deployment (API / Cloud Model)

- **Platform**: Deployed on a cloud server using Flask/Dash
- **Real-Time Forecasts**: API endpoint allows users to query future predictions
- **User Interface**: Dashboard built with Plotly Dash to visualize trends and predictions

7. MLOps Pipeline Report

- **Experiment Tracking**: Used MLflow to log model parameters, metrics, and versions
- **Version Control**: Managed codebase with Git
- **Deployment Pipeline**:
  - Trained model serialized using joblib
  - Model deployed as RESTful API
- **Automated Training**: Jupyter Notebook and scripts prepared for periodic retraining

8. Monitoring Setup

- **Model Performance:**
  - RMSE and R^2 monitored over time using test datasets
  - Alerting thresholds can be defined for model degradation
- **Retraining Strategy**:
  - New data triggers retraining script
  - Periodic batch jobs scheduled (e.g., monthly)

9. Business Impact

- **Inventory Optimization**: Prevent overstock and stockouts by predicting demand
- **Improved Planning**: Better decisions for production, marketing, and staffing
- **Efficiency**: Reduced manual forecasting efforts, increased forecast reliability

10. Challenges Faced

- Handling inconsistent demand patterns
- Missing or unstructured historical data
- Choosing the optimal model and hyperparameters

11. Final Presentation Summary

- Delivered a stakeholder-friendly presentation
- Included visuals of prediction trends and heatmaps
- Live demo of the deployed dashboard with product-level filtering
- Showed how to interpret forecasts to take action in business planning

12. Future Improvements

- Integrate external data sources (e.g., macroeconomic indicators, competitor pricing)
- Try time-series-specific models like Facebook Prophet or ARIMA
- Deploy using scalable infrastructure like Docker and Kubernetes
- Add authentication and user roles to the dashboard for enterprise use

Deliverables

- **Deployed Model**: Available via web dashboard + REST API
- **MLOps Report**: Includes training pipeline, logging, retraining setup
- **Monitoring Setup**: Documentation on performance tracking and retraining
- **Final Project Report**: This document
- **Final Presentation**: Separate PowerPoint/Google Slides summarizing model usage and results