Inventory Sales Forecasting Model Documentation

Introduction

This document provides a step-by-step explanation of how the sales forecasting model is built, trained, and saved for future use. This guide is intended for beginners who want to understand the process in detail.

---

1. Importing Required Libraries

```
import pandas as pd

import numpy as np

import xgboost as xgb

import pickle

from sklearn.preprocessing import LabelEncoder
```

Explanation:

- pandas → Used for handling and processing datasets.
- numpy → Provides numerical operations.
- xgboost → Machine learning library for training the forecasting model.
- pickle → Helps save and load Python objects (encoders and data transformations).
- LabelEncoder → Converts categorical variables (e.g., product categories) into numerical values.

---

2. Loading the Dataset

```
file_path = "superstore_final_dataset.csv"

df = pd.read_csv(file_path, encoding="ISO-8859-1")
```

Explanation:

- Reads the dataset from a CSV file.
- ISO-8859-1 encoding is used to ensure special characters are properly read.

---

3. Processing Dates

```
df['Order_Date'] = pd.to_datetime(df['Order_Date'], dayfirst=True, errors='coerce')

df.dropna(subset=['Order_Date'], inplace=True)

df['Year-Month'] = df['Order_Date'].dt.to_period('M')
```

Explanation:

- Converts the Order_Date column to a proper date format.
- Drops any rows where the date is invalid.
- Extracts only the **Year and Month** to create a Year-Month column.

---

4. Aggregating Sales Data

```
sales_data = df.groupby(['Year-Month', 'Category',
'Sub_Category']).size().reset_index(name='Sales_Count')

sales_data['Year-Month'] = sales_data['Year-Month'].astype(str)

sales_data['Year-Month'] = pd.to_datetime(sales_data['Year-Month'])
```

Explanation:

- Groups sales by Year-Month, Category, and Sub_Category.
- Counts the number of sales in each category and sub-category per month.
- Converts Year-Month into a standard datetime format.

---

5. Encoding Categorical Variables

```
le_category = LabelEncoder()

le_sub_category = LabelEncoder()

sales_data['Category_Encoded'] = le_category.fit_transform(sales_data['Category'])

sales_data['Sub_Category_Encoded'] =
le_sub_category.fit_transform(sales_data['Sub_Category'])
```

Explanation:

- Converts text-based Category and Sub_Category into numbers using **Label Encoding**.
- Example: Furniture → 0, Technology → 1, Office Supplies → 2.

---

6. Creating Lag Features (Past Sales Trends)

```
sales_data.sort_values(by=['Year-Month'], inplace=True)

sales_data['Lag_1'] = sales_data.groupby(['Category_Encoded', 'Sub_Category_Encoded'])['Sales_Count'].shift(1)

sales_data['Lag_2'] = sales_data.groupby(['Category_Encoded', 'Sub_Category_Encoded'])['Sales_Count'].shift(2)

sales_data['Rolling_Mean_3'] = sales_data.groupby(['Category_Encoded', 'Sub_Category_Encoded'])['Sales_Count'].transform(lambda x: x.rolling(3, min_periods=1).mean())
```

Explanation:

- **Lag Features:** Adds past sales data to predict future sales.
- Lag_1 → Sales count from the previous month.
- Lag_2 → Sales count from two months ago.
- Rolling_Mean_3 → Average sales over the last 3 months.

---

7. Extracting Month and Year as Features

```
sales_data['Month'] = sales_data['Year-Month'].dt.month

sales_data['Year'] = sales_data['Year-Month'].dt.year
```

Explanation:

- Extracts Month and Year as separate features to help the model understand seasonality.

---

8. Dropping Missing Values

```
sales_data.dropna(inplace=True)
```

Explanation:

- Removes any remaining missing values (e.g., due to lag feature calculations).

---

9. Defining Features and Target Variable

```
FEATURES = ['Category_Encoded', 'Sub_Category_Encoded', 'Lag_1', 'Lag_2', 'Rolling_Mean_3', 'Month', 'Year']

X = sales_data[FEATURES]

y = sales_data['Sales_Count']
```

Explanation:

- Defines the **input features** (X) used for prediction.
- Defines the **target variable** (y) → The actual number of sales.

---

10. Training the XGBoost Model

```
model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=200, learning_rate=0.1, max_depth=5)

model.fit(X, y)
```

Explanation:

- **XGBoost Regressor** is used for forecasting sales.
- n_estimators=200 → Uses 200 trees in the model.
- learning_rate=0.1 → Determines how fast the model learns.
- max_depth=5 → Limits the depth of trees to prevent overfitting.

---

11. Saving the Model

```
model.save_model("sales_forecast_model.json")
```

Explanation:

- Saves the trained model as a .json file for future predictions.

---

12. Saving Encoders for Future Use

```
with open("label_encoders.pkl", "wb") as f:
   pickle.dump((le_category, le_sub_category, sales_data), f)
```

Explanation:

- Saves the **Label Encoders** and sales_data using pickle.
- Allows easy loading of encoded values when making predictions.

---

13. Final Confirmation

```
print("Model and encoders saved successfully!")
```

Explanation:

- Displays a success message to confirm everything is saved correctly.

Conclusion

This script:

1. Loads and processes sales data.
2. Converts categorical variables into numerical values.
3. Creates features based on past trends.
4. Trains an XGBoost model to forecast future sales.
5. Saves the trained model and encoders for later use.