Sales Forecasting Model Report

Introduction

In this report, we'll walk you through the steps taken to build a machine learning model that predicts sales revenue for Amaron battery products. The goal is to forecast how much revenue we can expect based on different factors, including sales date, product type, quantity sold, and other features.

To achieve this, we've used a Random Forest Regressor model. We'll dive into the data, preprocessing steps, model building, and evaluation, as well as the results. You'll get a clear overview of how the model works and its performance on test data.

Data Overview

We started with a dataset named `2025_01_22_amaron_sales_sample_data.xlsx`. This dataset contains several columns with both categorical and numerical data related to sales. The important columns include:

Sales Date: The date the sale was made.
Product Type: The type of product sold (e.g., battery type).
Quantity Sold:*How many units were sold.
-Revenue: The revenue generated from the sale (our target variable).

We began by examining the data and cleaning it, as some rows had missing values, and we removed those to make sure the model is trained on complete information.

Data Preprocessing

Preprocessing the data was a key step in making sure the model works correctly. We performed the following steps:

1. Extracting Date Features:
The `sale_date` column was converted to a datetime format, and we extracted features such as the **year, **month**, and **day**. This helps the model understand patterns based on the time of sale.**

2. Handling Missing Values:
We noticed some missing values in the dataset, which could impact the model's performance. Therefore, we decided to drop any rows with missing values.

3. Feature Transformation:
Categorical Data: We used One-Hot Encoding to convert non-numerical features (like `product_type`) into numerical values so the model can understand them.
Numerical Data:For columns that had numerical values, we used Standard Scaling to ensure all

features are on the same scale. This helps the model learn more effectively.

These steps made sure that the dataset is clean and ready for the model to learn from.

Building the Model

We used a Random Forest Regressor model for this task. Random Forests are a great choice for regression problems because they can handle complex relationships in the data and are robust to overfitting.

The model was set up using a Pipeline. A pipeline is a tool that automates the preprocessing and training process so we don't have to manually handle each step. Here's how the pipeline works:

1. Preprocessing: The data goes through transformations like One-Hot Encoding for categorical columns and Standard Scaling for numerical columns.
2. Training:*Once the data is ready, the model is trained on it, and it learns the relationship between the features and the target variable (revenue).

We used 150 estimators in the Random Forest to make sure the model has enough "trees" to make accurate predictions.

Code for Preprocessing and Model Building

Here’s the code we used for preprocessing and building the model. It may look like a lot, but don’t worry—it’s just a systematic way of handling the data and training the model.

```python
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib

# Load the data
file_path = "2025_01_22_amaron_sales_sample_data.xlsx" # Replace with your file path
data = pd.read_excel(file_path)

# Inspect the data
print("Dataset Overview:")
print(data.info())
print("
First 5 Rows:")
print(data.head())
```

```
# Step 1:

Preprocessing
# Convert 'sale_date' to datetime and extract features
data['sale_date'] = pd.to_datetime(data['sale_date'], errors='coerce') # Handle parsing errors
data['year'] = data['sale_date'].dt.year
data['month'] = data['sale_date'].dt.month
data['day'] = data['sale_date'].dt.day

# Drop 'sale_id' and 'sale_date' as they are not useful for prediction
data = data.drop(columns=['sale_id', 'sale_date'])

# Check for missing values and handle them
if data.isnull().sum().any():
 print("
Missing Values Detected:")
 print(data.isnull().sum())
 data = data.dropna() # Drop rows with missing values for simplicity

# Separate features and target
X = data.drop(columns=['revenue'])
y = data['revenue']

# Identify categorical and numerical columns
categorical_columns = [col for col in X.columns if X[col].dtype == 'object']
numerical_columns = [col for col in X.columns if X[col].dtype != 'object']

# Step 2:

Create a preprocessing pipeline
categorical_transformer = OneHotEncoder(handle_unknown='ignore', sparse_output=False) # Set
sparse=False for better readability
numerical_transformer = StandardScaler()

preprocessor = ColumnTransformer(
 transformers=[('num', numerical_transformer, numerical_columns), ('cat', categorical_transformer,
categorical_columns)]
)

# Step 3:

 Build the full pipeline with a RandomForestRegressor
model = Pipeline(steps=[('preprocessor', preprocessor), ('regressor',
RandomForestRegressor(n_estimators=150, random_state=42))])

# Step 4:
```

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

# Step 5:

 Train the model
model.fit(X_train, y_train)

# Step 6:

Evaluate the model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("
Model Evaluation:")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R^2 Score: {r2:.2f}")

# Save the model for future use
joblib.dump(model, "sales_forecast_model.pkl")
print("
Model saved as 'sales_forecast_model.pkl'")
```

Training and Evaluation

Once the model was trained, we tested it with a separate set of data (the test set) to see how well it performs on unseen data. The results were promising:

Mean Squared Error (MSE): A lower MSE indicates that our model's predictions were pretty close to the actual revenue.
R-Squared (R^2): This tells us how well the model captures the variance in the data. The closer to 1, the better the model explains the data.

The model's performance indicates it can make reasonable predictions on the revenue based on input features.

Model Saving and Deployment

To make sure we can use this model in the future, we saved it using joblib as a `.pkl` file. This allows us to easily load the model and use it on new data without retraining it from scratch.

The saved model file is named sales_forecast_model.pkl.

Conclusion and Recommendations

The Random Forest model has provided us with a solid baseline for predicting sales revenue. The model's performance is quite good, and there are still areas for improvement.

Next Steps:

1. Model Tuning: We can fine-tune the hyperparameters of the Random Forest model to improve performance.

2. Other Models:It might be worth testing other models like Gradient Boosting to see if they can provide even better results.

3. Data Enrichment: Adding more features such as promotions or weather data could potentially improve predictions further.

In the future, this model can be deployed for real-time sales forecasting, helping to make smarter, data-driven business decisions.