Assignment 2

ML as a Service:

Sales Revenue Prediction and Time-Series Forecasting for a Multi-State Retailer

Roy Hegde
Student ID: 24667610

Github Username	roynyne
Github Repos	Experiment Repository: https://github.com/roynyne/36120_AT2_experimentations API Repository: https://github.com/roynyne/sales-api
URLs	Backend: https://sales-api-backend-qw48.onrender.com Frontend: https://sales-api-frontend.onrender.com

36120 - Advanced Machine Learning Application
Master of Data Science and Innovation
University of Technology of Sydney

Table of Contents

1. Executive Summary

The goal of this project is to create machine learning models for a store that has locations in Wisconsin, Texas, and California. Predicting item-specific sales income at the shop level and projecting the overall sales for all stores over the following seven days were the objectives. The project's goal was to give the shop useful information for improving labour scheduling, marketing tactics, and inventory control.

Figure 1.1 Retail Store

Two models were developed:

- **Predictive Model:** Sales revenue for a certain item at a given store on a given day was predicted using XGBoost.
- **Forecasting Model:** For the following seven days, total sales across all stores were predicted using the time-series forecasting algorithm ARIMA.

The shop was able to obtain real-time sales forecasts when both models were made available as Render APIs. With Streamlit on the front end and FastAPI on the back end, the deployment was

made easier and the user experience was smooth. Because the models provide precise forecasts which are essential for labour allocation and inventory management—the company can save operating expenses. For the predictive test, XGBoost obtained a Root Mean Squared Error (RMSE) of 3.013 whereas ARIMA outperformed Prophet in terms of dependable performance for time-series forecasting.

2. Business Understanding

a. Business Use Cases

Keeping up with changing demand may be difficult for retail firms when it comes to personnel, marketing, and inventory management. This project's retailer, which has ten locations in three states, needs precise sales projections to handle the following:

- Inventory control: While stockouts result in lost sales and disgruntled customers, overstocking can raise holding expenses. To effectively manage inventory, the store has to be able to forecast demand for certain goods.
- Labour Planning: Inaccurate estimates might result in stores having too few employees during busy times or too many employees during calm times, which would be inefficient use of labour.
- Marketing and Promotions: Targeting promotions and running more efficient marketing campaigns are two ways to increase profitability when you know when and where demand will jump. to speak to:

In order to assist managers make data-driven choices, the machine learning models created for this project provide forecasts that specifically meet these business demands. The shop can enhance promotional activities, effectively schedule workers, and optimise inventory levels using precise demand estimates.

b. Key Objectives

The project has three main objectives:

- Create a Predictive Model: Based on past sales data, event information, and price, machine learning algorithms are used to anticipate sales income for certain goods at individual stores.
- Create a Model for Forecasting: To anticipate the entire sales income for all retailers over the next seven days, use a time-series forecasting model. Allocating resources and strategic planning will be aided by this paradigm.
- Models to be Deployed as APIs: In order to facilitate real-time engagement, both models will be implemented as APIs, enabling stakeholders to obtain forecasts whenever needed.

Stakeholders include:

- Store managers are in charge of making sure that personnel and inventory levels correspond to demand.
- The operations team allocates resources throughout retailers and manages supply chain logistics using projections.
- Marketing Department: Based on anticipated demand, forecasts are used to optimise promotional efforts and modify strategy.

By addressing these key objectives, the project empowers the retailer to enhance its operational efficiency and improve decision-making processes.

3. Data Understanding

The study made use of many datasets from different sources, each of which provided crucial data for model training:

- Sales Training Data: This dataset includes daily sales information for certain products sold in a number of different retailers. It serves as the main dataset for the prediction model and offers a detailed historical record of sales patterns. Each row includes the item ID, shop ID, and many identifiers such as department and category, and it also represents the daily sales of an item at a certain store.

- Calendar Data: Weekends, holidays, and other noteworthy events that can have an impact on sales trends are among the crucial date-related details that are captured in the calendar data. Seasonality and trends, which are critical for time-series forecasting models like ARIMA and Prophet, may be found in this data.

- Event Data: Promotions and vacations, for example, are known to have a big influence on customer behaviour. The event dataset contains details on a number of annual occasions, such Black Friday and Thanksgiving, that might have an impact on sales. The model may take these demand surges into account by integrating this data.

- Weekly Sales Data for goods: This dataset compiles pricing and sales information for individual goods on a weekly basis, giving users a general idea of how prices change over time. Given the importance of price in consumer decision-making, this information is critical to the predictive and forecasting models.

Every dataset has some restrictions. For instance, several item prices in the sales data were missing, and some holidays' event data was lacking. During the data preparation stage, these problems were resolved via preprocessing and data cleaning.

```
Sales Training Data:
                            id        item_id    dept_id   cat_id store_id  \
0  HOBBIES_1_001_CA_1_evaluation  HOBBIES_1_001  HOBBIES_1  HOBBIES     CA_1
1  HOBBIES_1_002_CA_1_evaluation  HOBBIES_1_002  HOBBIES_1  HOBBIES     CA_1
2  HOBBIES_1_003_CA_1_evaluation  HOBBIES_1_003  HOBBIES_1  HOBBIES     CA_1
3  HOBBIES_1_004_CA_1_evaluation  HOBBIES_1_004  HOBBIES_1  HOBBIES     CA_1
4  HOBBIES_1_005_CA_1_evaluation  HOBBIES_1_005  HOBBIES_1  HOBBIES     CA_1

  state_id  d_1  d_2  d_3  d_4  ...  d_1532  d_1533  d_1534  d_1535  d_1536  \
0       CA    0    0    0    0  ...       1       1       1       0       1
1       CA    0    0    0    0  ...       0       0       0       0       0
2       CA    0    0    0    0  ...       0       0       1       0       0
3       CA    0    0    0    0  ...       8       2       0       8       2
4       CA    0    0    0    0  ...       2       0       1       3       2

   d_1537  d_1538  d_1539  d_1540  d_1541
0       0       1       0       0       1
1       0       0       0       1       0
2       0       0       0       0       0
3       3       1       1       3       8
4       1       1       2       2       3

[5 rows x 1547 columns]
```

Figure 3.1 Sales Training Data

```
5]: print("\nCalendar Data:")
    print(calendar_df.head())

Calendar Data:
         date  wm_yr_wk    d
0  2011-01-29     11101  d_1
1  2011-01-30     11101  d_2
2  2011-01-31     11101  d_3
3  2011-02-01     11101  d_4
4  2011-02-02     11101  d_5
```

Figure 3.2 Calender Data

```
: print("\nCalendar Events Data:")
  print(calendar_events_df.head())
```

```
Calendar Events Data:
         date     event_name event_type
0  2011-02-06      SuperBowl   Sporting
1  2011-02-14  ValentinesDay   Cultural
2  2011-02-21  PresidentsDay   National
3  2011-03-09      LentStart  Religious
4  2011-03-16      LentWeek2  Religious
```

Figure 3.3 Calender Events Data

```
print("\nItems Weekly Sales:")
print(items_weekly_sales_df.head())
```

```
Items Weekly Sales:
  store_id        item_id  wm_yr_wk  sell_price
0     CA_1  HOBBIES_1_001     11325        9.58
1     CA_1  HOBBIES_1_001     11326        9.58
2     CA_1  HOBBIES_1_001     11327        8.26
3     CA_1  HOBBIES_1_001     11328        8.26
4     CA_1  HOBBIES_1_001     11329        8.26
```

Figure 3.4 Items Weekly Sales Data

4. Data Preparation

A number of crucial procedures were engaged in data preparation to guarantee that the datasets were appropriate for model training:

- Dataset Combination:
    - Reshaping Sales Data: The melt function was used to convert the sales data from wide to long format. This made it possible to show the sales of each item on a given day for each row.
    - Merging Calendar Data: Using the day reference (d), the reshaped sales data was combined with calendar data to provide attributes relating to time and date.
    - Incorporating Event Information: To account for the influence of promotions and holidays, calendar event data on the date column was combined to create special events.
    - Adding Price Data: In order to account for price variations that effect sales, item prices were finally merged depending on wm_yr_wk, item_id, and store_id as well.
- Data cleaning: Forward-filling techniques were used to fill in the missing data, particularly in the item price columns. The model was given complete training data by filling up the missing variables using historical prices. To enable the algorithm to discern between genuine events and ordinary sales periods, entries without event data were labelled as "No Event."
- Feature Engineering: In order to better capture the underlying patterns in the data, new features were produced from old ones via feature engineering, which was a crucial step in the data preparation process:
    - Date Features: To help the model understand how time affects sales patterns, the "date" column was divided into many aspects, such as the day, month, and weekday.
    - Label Encoding: Label encoding was applied to categorical variables, such as store_id, item_id, event_name, and event_type. This made it possible for machine learning algorithms to handle these category characteristics numerically, such as XGBoost and Ridge.

- Managing Outliers and Missing Data: To maintain the integrity of the dataset, outliers in the sales data were retained, while forward-fill methods were utilised to fill in the missing values in non-essential categories. Price data consistency was guaranteed by the forward-fill approach, and outliers offered vital information about atypical sales spikes—a critical component for models attempting to understand the impact of uncommon occurrences.

- Data Splitting: Training and testing sets of the data were created. Twenty percent of the data were set aside for testing and eighty percent were utilised for training the prediction model. To make sure the forecasting model was not trained on any future data, time-based validation was employed, whereby data from previous periods was utilised to predict future patterns.

```
: sales_train_with_prices.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 47107050 entries, 0 to 47107049
Data columns (total 16 columns):
 #   Column      Dtype
---  ------      -----
 0   id          object
 1   item_id     object
 2   dept_id     object
 3   cat_id      object
 4   store_id    object
 5   state_id    object
 6   d           object
 7   sales       int64
 8   date        datetime64[ns]
 9   wm_yr_wk    int64
 10  event_name  object
 11  event_type  object
 12  sell_price  float64
 13  day         int64
 14  month       int64
 15  weekday     int64
dtypes: datetime64[ns](1), float64(1), int64(5), object(9)
memory usage: 6.0+ GB
```

Figure 4.1 Calender Data

	Event Name	Encoded Value
0	Chanukah End	0
1	Christmas	1
2	Cinco De Mayo	2
3	ColumbusDay	3
4	Easter	4
5	Eid al-Fitr	5
6	EidAlAdha	6
7	Father's day	7
8	Halloween	8
9	IndependenceDay	9
10	LaborDay	10
11	LentStart	11
12	LentWeek2	12
13	MartinLutherKingDay	13
14	MemorialDay	14
15	Mother's day	15
16	NBAFinalsEnd	16
17	NBAFinalsStart	17
18	NewYear	18
19	No Event	19
20	OrthodoxChristmas	20
21	OrthodoxEaster	21
22	Pesach End	22
23	PresidentsDay	23
24	Purim End	24
25	Ramadan starts	25
26	StPatricksDay	26
27	SuperBowl	27
28	Thanksgiving	28
29	ValentinesDay	29
30	VeteransDay	30

Figure 4.2 Encoded Event Name Values

	Event Type	Encoded Value
0	Cultural	0
1	National	1
2	None	2
3	Religious	3
4	Sporting	4

Figure 4.3 Encoded Event Type Values

5. Modeling

a. Approach 1: Ridge Regression

The baseline model for item-specific sales prediction was Ridge Regression, a linear model that supplements ordinary least squares with L2 regularisation. Ridge is especially helpful in situations when the data exhibits multicollinearity, or strongly linked properties. Due to its ability to penalise high coefficients and hence avoid overfitting, the regularisation is a viable contender for initial model selection.

Why Ridge Regression?

Ridge Regression is a great place to start for predictive modelling because of its simplicity, particularly in retail settings where the relationship between sales and characteristics (such item price, shop location, or promotions) might not seem to be linear at first.

The dataset underwent preprocessing to provide significant characteristics prior to the fitting of the Ridge Regression model:

- Date Features: Day, month, and weekday variables were created from the 'date' field. Retail sales are cyclical and seasonal, as seen by the fact that they usually rise on weekends and fall in the middle of the week.
- Event Encoding: Event_name_encoded and Event_type_encoded are categorical characteristics that were label-encoded using holiday and event data, which is essential for retail forecasting. The model was able to take into consideration surges during important sales occasions like Black Friday or Christmas thanks to this modification.
- Store and Item Encoding: To give numeric values to category store and item IDs, store_id and item_id were both label-encoded. To capture trends unique to individual items in stores, these aspects are crucial.

b. Approach 2: XGBoost

The second strategy was XGBoost (Extreme Gradient Boosting), a potent and well-liked ensemble learning technique that excels at managing big datasets, feature interactions, and non-linear correlations. XGBoost's performance and versatility have made it a leading algorithm for structured/tabular data certain patterns.

Similar to Ridge Regression, XGBoost's performance was greatly enhanced via feature engineering:

- Price Normalisation: To assist the model train more efficiently, the sell_price feature was normalised to standardise the range of prices across various goods.
- Event Features: To capture the effect of holidays and special events on sales, two features were used: event_name_encoded and event_type_encoded. These characteristics enabled XGBoost to modify its forecasts according to the presence of promotions or the holiday season on a given day.
- Date data: To better capture cyclical sales trends, XGBoost benefited from extra date data such as the week of the year and whether the day was a weekday or a weekend.

```python
# Function to train XGBoost model
def train_xgboost(X_train, y_train):
    xgb_model = xgb.XGBRegressor(n_estimators=100, max_depth=10, learning_rate=0.1, random_state=42)
    xgb_model.fit(X_train, y_train)
    print("XGBoost model trained successfully.")
    return xgb_model
```

Figure 5.b.1 XGB Model Parameters

c. Approach 3: ARIMA (Forecasting Model)

ARIMA (Auto-Regressive Integrated Moving Average) was chosen as the main model for time-series forecasting. The retailer's sales data included both seasonality and observable patterns, which make this approach especially useful.

Model Selection and Tuning

Three critical parameters of the ARIMA model need to be carefully adjusted: p (number of lag observations), d (degree of differencing), and q (moving average window size). The model that performed the best after experimenting with several parameter combinations was ARIMA(5, 1, 0). These specifications showed shown in Figure 5.c.1:

- p=5: The model predicts the subsequent value based on five prior observations.
- d=1: To make the series stationary and account for trends over time, a single differencing was used.
- q=0: Since the moving average window wasn't needed for the sales data, it wasn't applied in this instance.

```python
# Function to train ARIMA model
def train_arima(sales_data):
    # Convert the aggregated sales data into a time series
    sales_series = sales_data.set_index('ds')['y']

    # Train ARIMA model (ARIMA(5, 1, 0) as an example, modify parameters as needed)
    arima_model = ARIMA(sales_series, order=(5, 1, 0))
    arima_result = arima_model.fit()

    # Save the ARIMA model using joblib
    #joblib.dump(arima_result, '/Users/bananavodka/Projects/at2_mla/at2_mla/models/f
    #print("ARIMA model saved successfully.")

    return arima_result
```

Figure 5.c.1 ARIMA Parameters

d. Approach 4: Prophet (Forecasting Model)

From Figure 6.2, prophet's ability to manage complicated or erratic sales patterns brought on by holidays and promotions is one of its distinctive qualities. Users may enter particular holidays or events that could affect sales into Prophet, and the program modifies its forecast appropriately. In this project, Prophet was fed special event data, including Thanksgiving and other holidays, so it could account for the corresponding surges in sales.

Though Prophet could identify these anomalous sales trends, its performance fell short of ARIMA in this particular instance. The retailer's sales data showed high weekly seasonality, which ARIMA handled more skilfully with simple differencing and autoregressive terms. This might be one explanation for the observed behaviour.

6. Evaluation

a. Evaluation Metrics

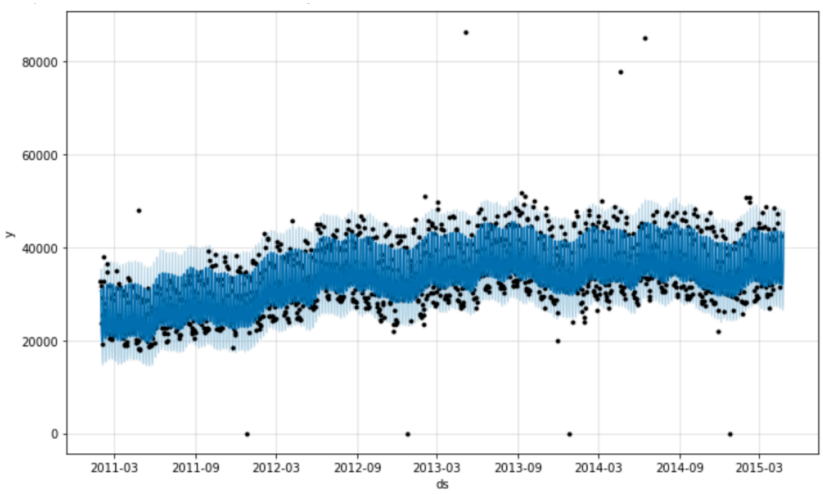
The main assessment statistic for the forecasting and predicting models was RMSE, or root mean squared error. The average discrepancy between the expected and actual sales numbers is measured by RMSE. It was especially crucial in this situation since even little fluctuations in sales forecasts can have a big impact on the retailer's operations, resulting in things like excess inventory or understaffing.

b. Results and Analysis

- Ridge Regression: This baseline model has an RMSE of 3.544.
- XGBoost: With an RMSE of 3.013, this prediction model was chosen as the final one because of its excellent results.
- ARIMA: Selected as the ultimate forecasting model due to its precise predictions and dependable national sales forecasting performance.

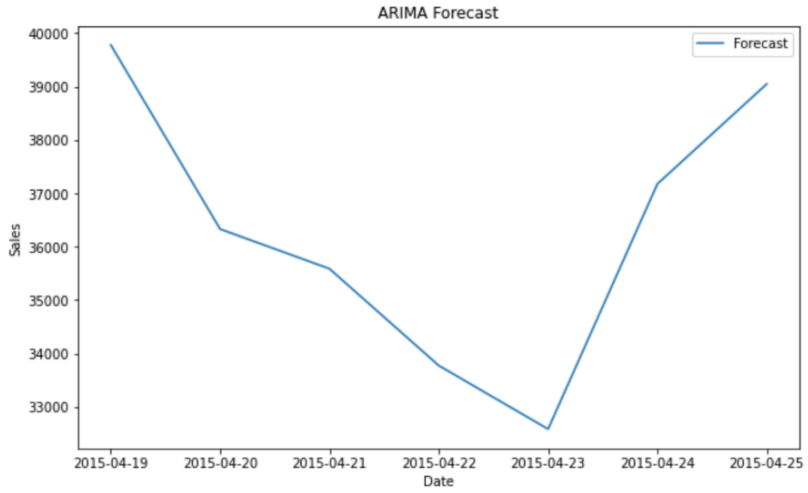
```
Datasets loaded successfully.
Data preprocessed successfully.
Features created successfully.
Data split into training and testing sets.
Ridge model trained successfully.
Test RMSE with Ridge Regression: 3.5442486407875036
XGBoost model trained successfully.
Test RMSE with XGBoost: 3.0138805175029355
```

Figure 6.1 XGB & Ridge Regression Results




```
Forecast generated and plotted successfully.
             ds          yhat    yhat_lower    yhat_upper
1541 2015-04-19  43406.839464  38388.593292  48039.169527
1542 2015-04-20  34755.405185  29633.130013  39444.067132
1543 2015-04-21  32354.027674  27404.505361  37183.561481
1544 2015-04-22  31946.385414  26593.682912  36263.521767
1545 2015-04-23  32017.665288  27450.433695  36859.524362
1546 2015-04-24  36124.125229  31358.820457  41040.959960
1547 2015-04-25  43152.995340  38530.039900  48021.927489
```

Figure 6.2 fbprophet Results




```
ARIMA Forecast generated successfully.
                 Date  Forecasted_Sales
2015-04-19 2015-04-19      39778.550864
2015-04-20 2015-04-20      36329.955542
2015-04-21 2015-04-21      35587.483480
2015-04-22 2015-04-22      33773.875764
2015-04-23 2015-04-23      32584.160338
2015-04-24 2015-04-24      37176.035216
2015-04-25 2015-04-25      39047.239653
```

Figure 6.3 ARIMA Results

c. Business Impact and Benefits

The models created have significant business value.

- XGBoost: The merchant may more effectively manage inventory and steer clear of stock-related problems by precisely projecting sales for individual goods at certain shops. As a result, there are lower overstocking expenses and fewer stockouts that result in missed revenues.
- ARIMA: The store can best allocate labour and manage resources by using the national sales prediction. The retailer can staff stores adequately and make sure supply chain operations are in line with predicted demand by having knowledge of future sales patterns.
- The models not only increase operational efficiency but also give the marketing team insightful information. The retailer may more efficiently arrange promotions and boost sales and customer happiness by pinpointing times of strong demand.

d. Data Privacy and Ethical Concerns

Anonymised data was utilised in this project to guarantee that no private client information was revealed. The models' primary focus on sales patterns and outside events reduced the likelihood of data privacy violations. To guarantee that all shops and areas were treated equally in the projections, care was taken to prevent bias in the models, especially in the event forecasting aspects.

7. Deployment

Two essential elements had to be put up in order for the predictive and forecasting models to be deployed:

- FastAPI Backend: For managing forecast and prediction queries.
- Streamlit Frontend: To offer an interactive user interface for accessing the models.

To provide scalability and accessibility, both services were deployed on Render after being containerised using Docker. An outline of the deployment structure may be seen below, along with thorough descriptions of each part.

Backend Deployment with FastAPI

FastAPI, a cutting-edge, asynchronous web framework perfect for managing real-time forecasts and predictions, was used in the development of the backend API. There are two primary uses for the API:

- Forecast sales by item with the XGBoost model.
- Using the ARIMA model, provide a seven-day national sales estimate.

The **main.py** file defines the API endpoints:

- Root Endpoint: Shows instructions on how to use the API and a welcome message.
- Healthcheck Endpoint: Attests to the models operational readiness.
- Prediction endpoint (/sales/stores/items/): Returns sales projections based on store ID, item ID, date, price, and event information at the.
- Forecasting Endpoint (/sales/national/): Based on the commencement date, offers a seven-day projection for national sales.

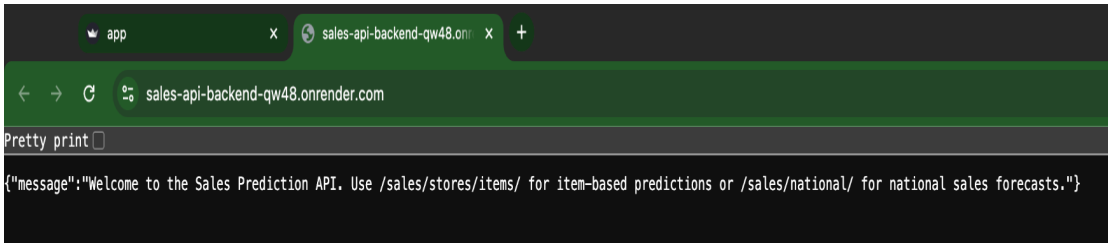


Figure 7.1 GET (“/”) root welcome message

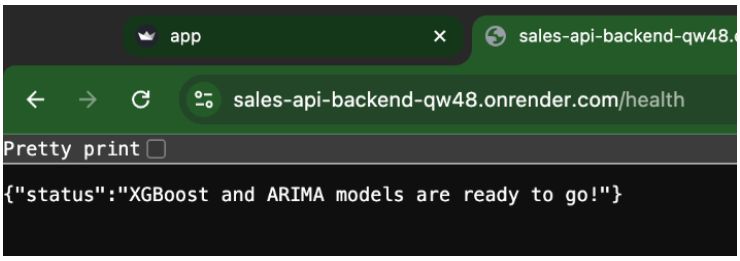


Figure 7.2 GET ("/health") health check

CORS (Cross-Origin Resource Sharing) was enabled in order to guarantee smooth communication between the frontend and backend services, enabling requests from the Streamlit interface hosted on Render.

```
12
13    # Set allowed origins for CORS (change to your frontend URL if needed)
14    origins = [
15        "http://localhost:8501",  # Streamlit frontend running locally
16        "https://sales-api-frontend.onrender.com",  # Replace with actual Render frontend URL
17    ]
18
19    # Add CORS middleware to the app to allow communication between frontend and backend
20    app.add_middleware(
21        CORSMiddleware,
22        allow_origins=origins,
23        allow_credentials=True,
24        allow_methods=["*"],
25        allow_headers=["*"],
26    )
27
```

Figure 7.3 Bridging both ends with CORS

Docker for Backend

Docker was used to containerise the FastAPI backend, and the Dockerfile specified the environment and required libraries to operate the API. The backend may be readily installed on Render or other cloud services, and it is accessible via port 8000.

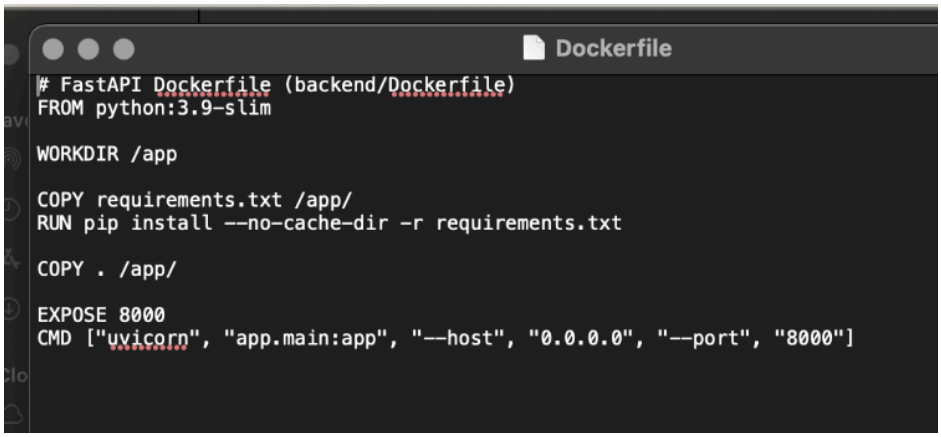


Figure 7.4 FastAPI/Backend Docker File

Frontend Deployment with Streamlit

Streamlit was used in the frontend's construction to provide an interactive user experience.

- Enter the pricing, event details using encoded integers from Figure 4.2 & Figure 4.3, store and item details, and predict store/item sales. The FastAPI backend receives these inputs from Streamlit and returns the anticipated sales.
- With the start date supplied by the user, forecast national sales over the next seven days. The FastAPI endpoint receives the request from the frontend and provides the predicted values in a table.

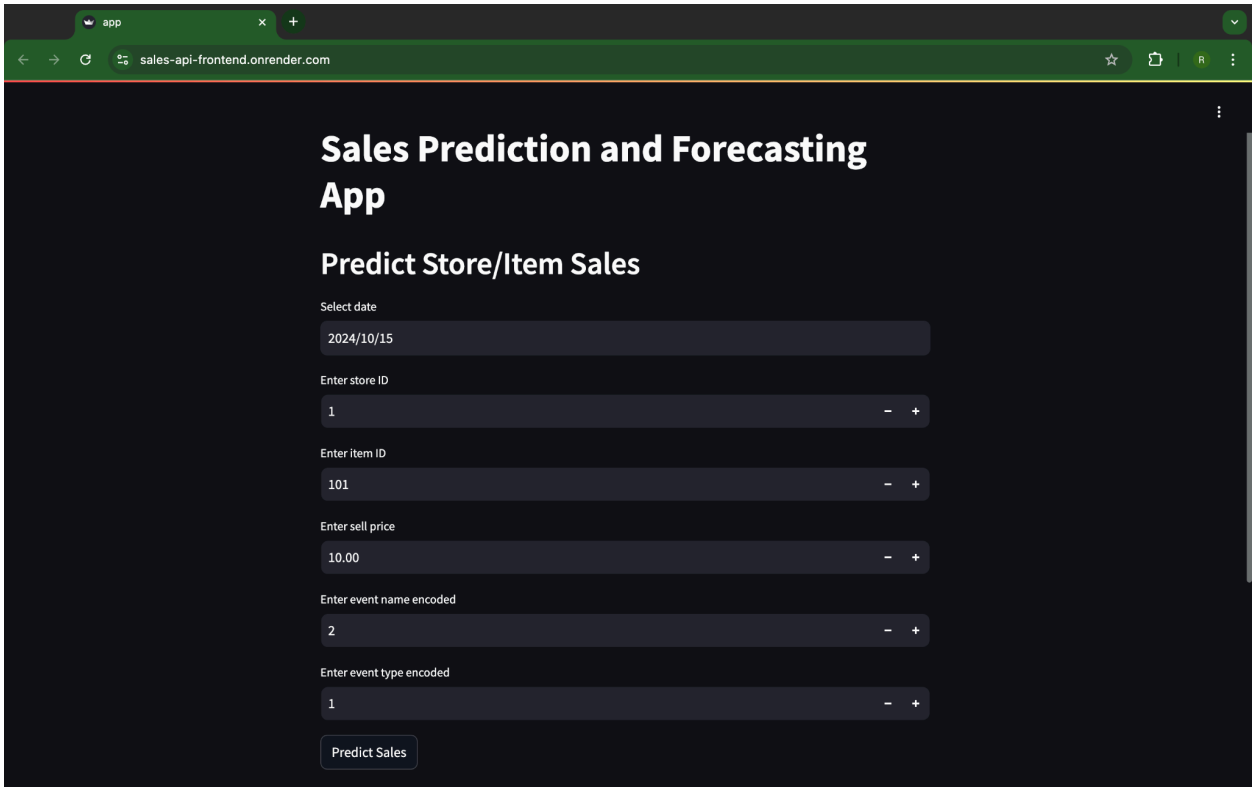


Figure 7.5 Streamlit Interface Input Screen

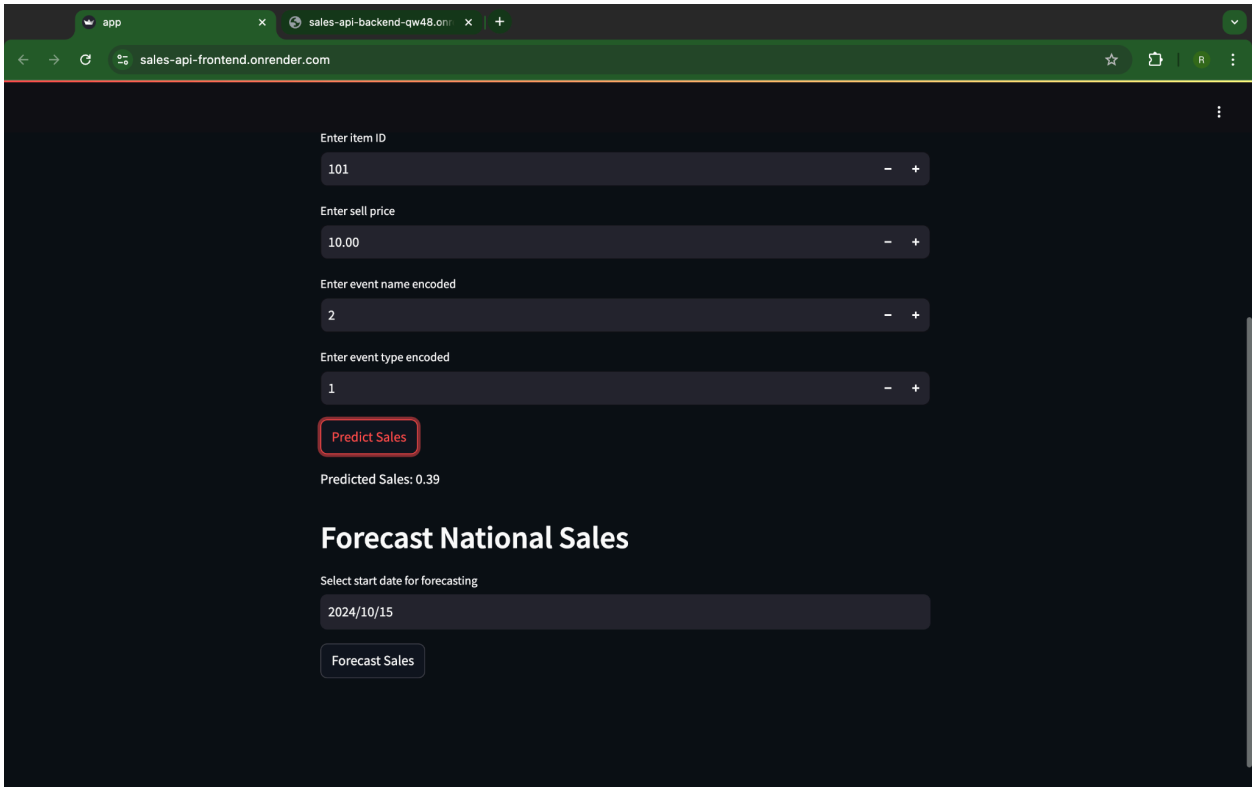


Figure 7.6 ("/sales/stores/items/") Results

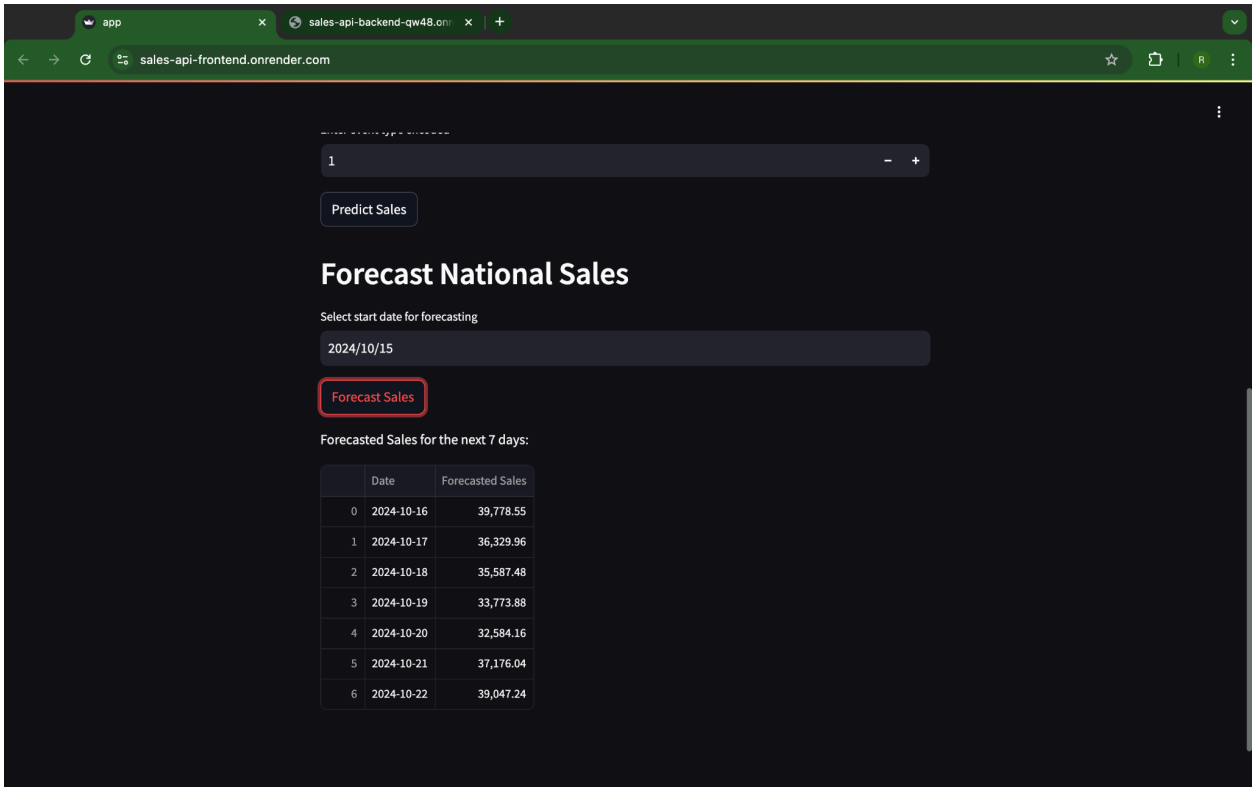


Figure 7.7 ("/sales/national/") Results

Docker for Frontend

Docker is used to containerise and deploy the Streamlit frontend, much like it is for the backend. The frontend uses HTTP requests to establish direct communication with the FastAPI backend while operating on port 8501.

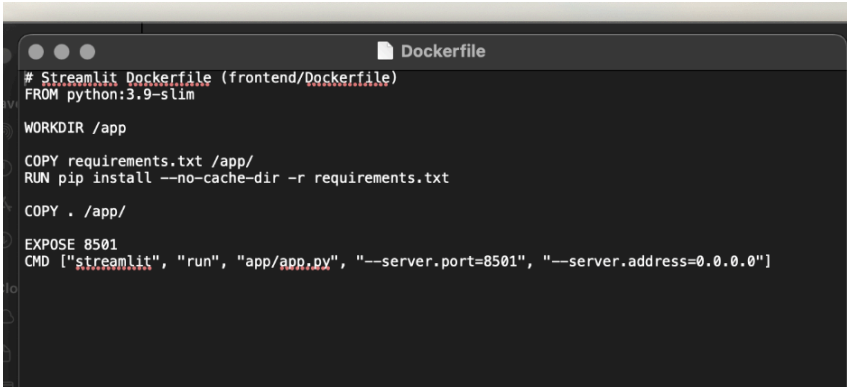


Figure 7.8 Streamlit/Frontend Docker File

Simultaneous Deployment on Render

The frontend (Streamlit) and backend (FastAPI) were both deployed on Render, guaranteeing smooth communication between the two services. With the frontend acting as the user interface and the backend performing model inference, Render's free tier makes it simple to set up and host both services.

Render Links:

- Backend API: https://sales-api-backend-qw48.onrender.com
- Frontend App: https://sales-api-frontend.onrender.com

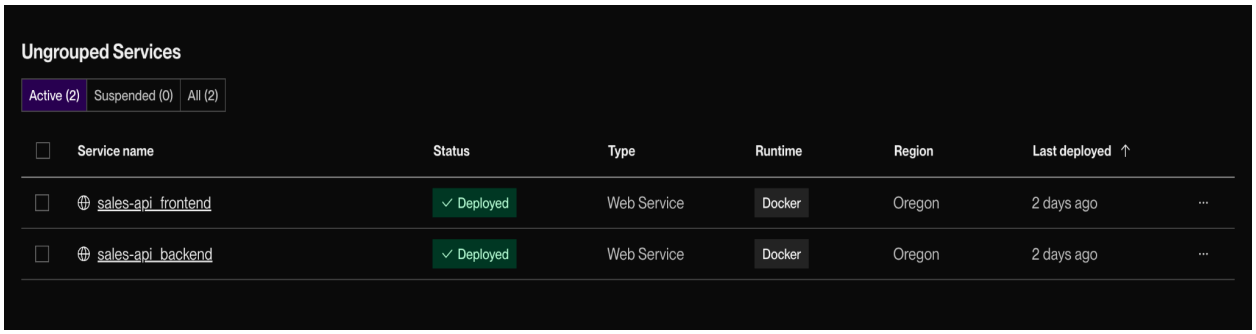


Figure 7.9 Render Web Services for Streamlit & FastAPI

8. Conclusion

Through the development and implementation of machine learning models for sales forecasting and prediction, this project effectively met its goals. Because of its accuracy and performance, the XGBoost model was picked for item-store-date forecasts, whereas the ARIMA model was chosen for time-series forecasting across all stores. The store may make better decisions about labour allocation, marketing tactics, and inventory management with the help of these actionable data from both models.

To further improve forecast accuracy, future studies may involve adding other factors, such as weather data. Maximising the advantages of these models might also include expanding the solution's reach to more locations and connecting it with the retailer's larger supply chain management system.