CC 3.0

FetchHub

Krish Pillai ,Nnisarg Gada, Yash Shah

Solution

Harnessing Fetch.ai's uAgents, we've created a decentralized network for seamless data exchange, focusing on critical attributes like restocking quantity. By introducing advanced neural network models, we empower Aryan's business with predictive analytics, optimizing inventory management and maximizing profitability. This innovative integration ensures a dynamic and proactive approach to business operations, positioning Aryan's story at the forefront of efficiency and competitiveness.

Technologies Used

- ## Fetch.ai uagents

  Facilitates efficient collaboration and real-time data sharing between different components, enhancing inventory tracking and overall supply chain management.

- ## Tensorflow

  Employs TensorFlow for developing and training deep learning models, particularly neural networks, to analyze and predict trends in inventory, optimizing decision-making.

- ## Fastapi

  Utilizes FastAPI to develop a robust API for seamless integration between different components, enabling efficient data communication and retrieval.

- ## Scikit learn

  Leverages Scikit-learn for various machine learning tasks, such as preprocessing data, training models, and evaluating their performance, contributing to accurate predictions and analysis.

- ## Pandas & numpy

  pandas and numpy are used for efficient data handling, preprocessing, and manipulation, enabling seamless integration of tabular data

- ## Astro.js

  Astro.js prioritizes content-driven websites, offering server-first simplicity and versatile UI framework support for a developer-friendly, high-performance experience.

Phases of FetchHub

1

uagents communication

Employing Fetch.ai uAgents, our project establishes decentralized communication protocols, fostering efficient data exchange and coordination among autonomous agents.

2

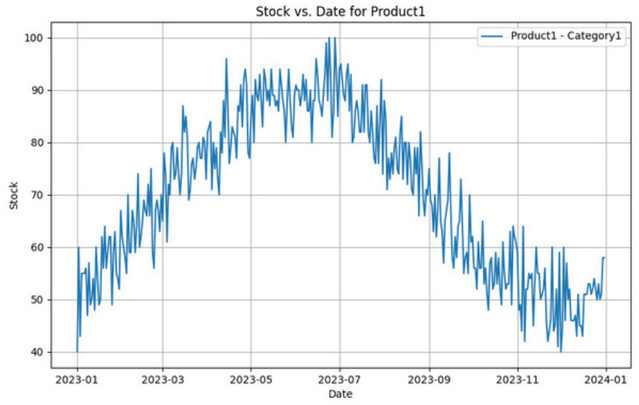
deep learning

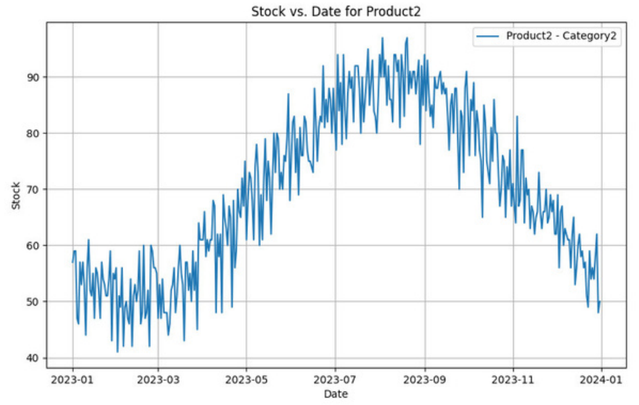
Leveraging advanced deep learning models, our project utilizes TensorFlow and scikit-learn for predictive analytics, enabling accurate forecasts for inventory management and business optimization.

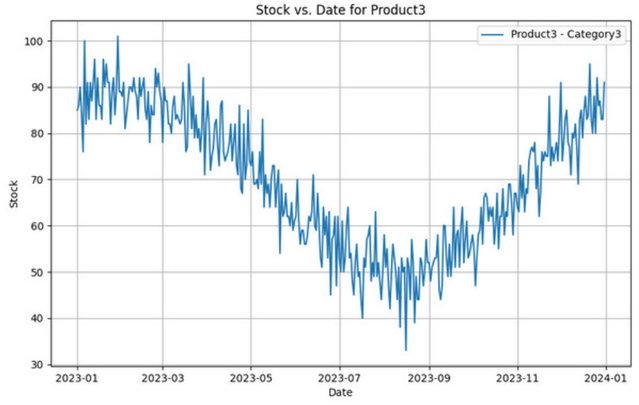
3

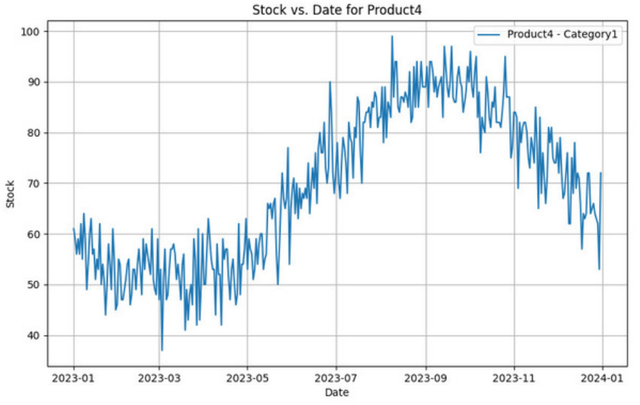
frontend

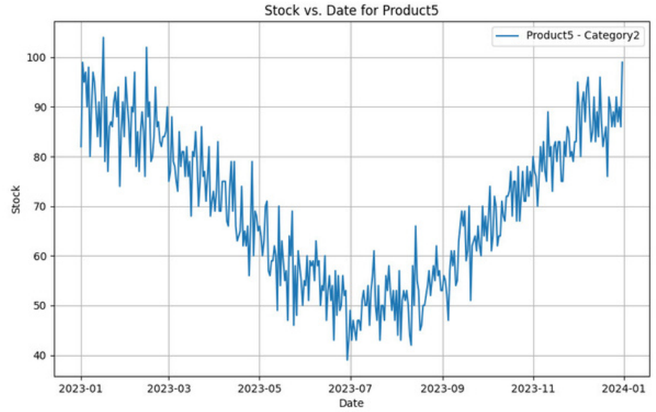
Leveraging advanced deep learning models, our project utilizes TensorFlow and scikit-learn for predictive analytics, enabling accurate forecasts for inventory management and business optimization.


Stock vs. Date for Product1
Product1 - Category1
Stock
Date
100
90
80
70
60
50
40
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01


Stock vs. Date for Product2
Product2 - Category2
Stock
Date
90
80
70
60
50
40
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01


Stock vs. Date for Product3
Product3 - Category3
Stock
Date
100
90
80
70
60
50
40
30
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01


Stock vs. Date for Product4
Product4 - Category1
Stock
Date
100
90
80
70
60
50
40
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01


Stock vs. Date for Product5
Product5 - Category2
Stock
Date
100
90
80
70
60
50
40
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01

Synthesizing the dataset

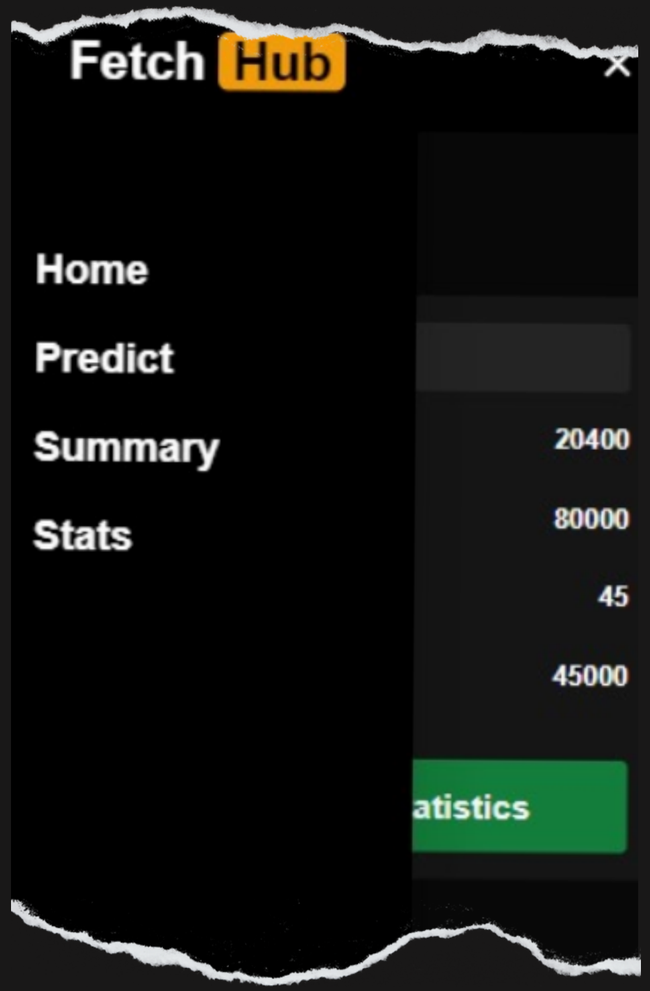
- Used sine waves to emulate sales and added offsets for different products
- Increased noise during the summertime to make realistic sales data
- Stock levels replenish if they are below threshold for 2 consecutive days

```
for date in date_range:
    for product in products:
        category = categories[product]

        # Generate periodic demand increase during the summer with an offset
        demand_increase = 20 * np.sin(2 * np.pi * date.timetuple().tm_yday / 365 + offsets[product])

        # Generate stock reduction during high demand periods
        base_stock = 70
        stock_noise = base_stock - demand_increase + np.random.normal(0, 5)

        # If stock goes below 0, set it to 0
        stock = max(0, int(stock_noise))

        # If stock is 0 for two consecutive days, restock on the next day
        if stock == 0:
            consecutive_zero_days = data[-2][0] if len(data) >= 2 and data[-1][2] == 0 else 0
            if consecutive_zero_days == date - timedelta(days=1):
                stock = restock_thresholds[product]  # Restock threshold

        # Calculate today's restock amount
        if len(data) > 0 and data[-1][0] == date - timedelta(days=1):
            yesterday_stock = data[-1][2]
```

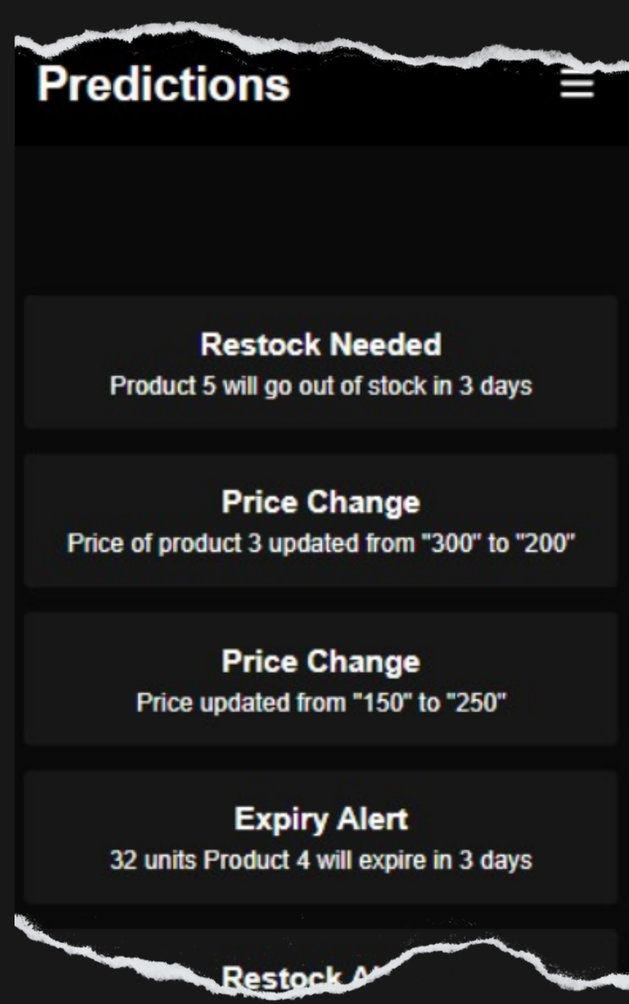
```
        # Calculate today's restock amount
        if len(data) > 0 and data[-1][0] == date - timedelta(days=1):
            yesterday_stock = data[-1][2]
            restock_amount = max(0, stock - yesterday_stock)
        else:
            restock_amount = 0

        # Restock only when the stock is significantly close to the restock threshold
        if stock < 0.2 * restock_thresholds[product]:
            restock_amount = max(restock_amount, np.random.randint(5, 20))

        # Calculate today's price based on yesterday's price and dynamic pricing
        if len(data) > 0 and data[-1][0] == date - timedelta(days=1):
            yesterday_price = data[-1][5]
            # Change price by 5% every 7 to 15 days
            if np.random.rand() < 0.1:  # 10% chance for a price change
                price = yesterday_price * np.random.uniform(0.95, 1.05)
            else:
                price = yesterday_price
        else:
            price = base_prices[product]

        data.append([date, product, stock, restock_thresholds[product], restock_amount, price, category, shelf_lives[product]])
```

PROJECT WORKING

Home
page

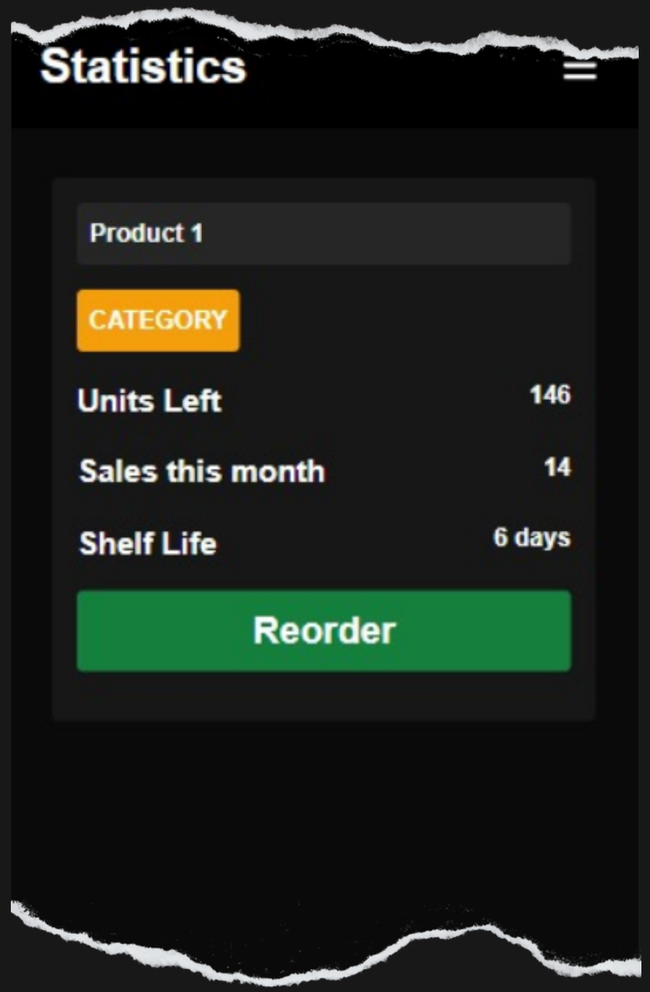


Menu
page

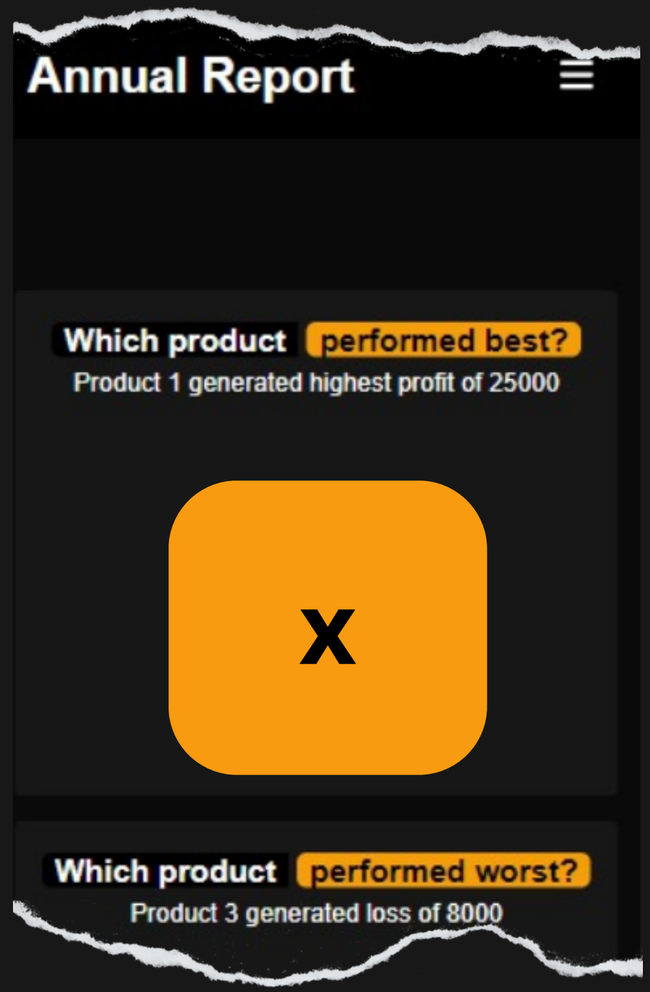


Predictions
page

PROJECT WORKING



Statistics
page



Summary
page

ThankYou