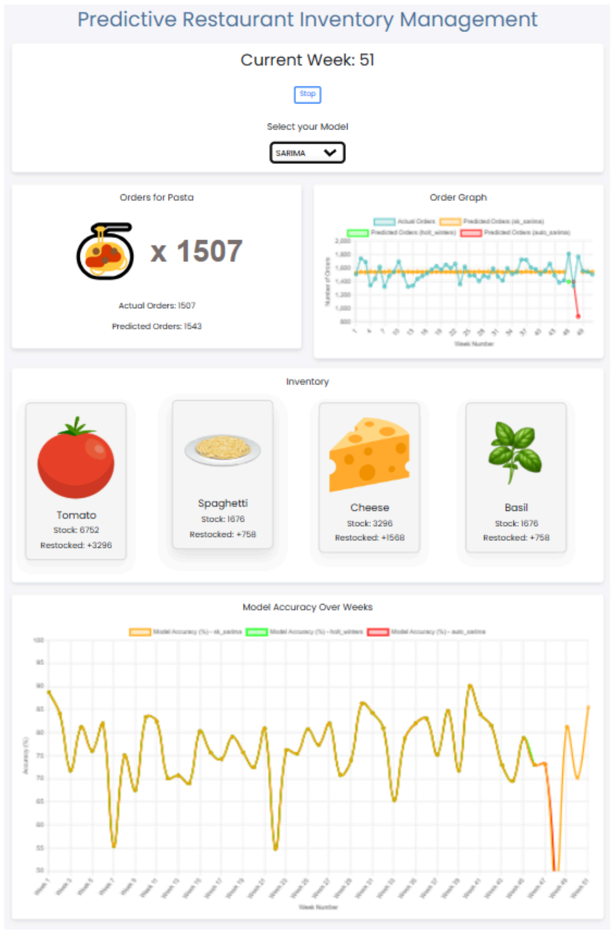
Predictive Restaurant Inventory Management

Current Week: 51

Select your Model: SARIMA

Orders for Pasta

x 1507

Actual Orders: 1507

Predicted Orders: 1543

Inventory

Tomato
Stock: 6752
Restocked: +3296

Spaghetti
Stock: 1676
Restocked: +758

Cheese
Stock: 3296
Restocked: +1568

Basil
Stock: 1676
Restocked: +758

Stop
Order Graph
Actual Orders
Predicted Orders (sk_sarima)
Predicted Orders (holt_winters)
Predicted Orders (auto_sarima)
Number of Orders
Week Number
Model Accuracy Over Weeks
Model Accuracy (%) - sk_sarima
Model Accuracy (%) - holt_winters
Model Accuracy (%) - auto_sarima
Accuracy (%)
Week Number

Predictive Restaurant Inventory Management System (PRIMS)

Team 7

Akash Vijaykumar Kotekar - 124106044
Lakshya Suri - 124101111
Mamtha Mahanthesh Vathar - 124106172
Mandar Deepak Bagwe - 123111875
Meghan Vinayak Mane - 124100669
Moras Kashyap - 124103079
Ramya Thimmappa Gowda - 124106243

Github Link :- https://github.com/tramya16/predictive-restaurant-inventory.git

Table of Contents

Table of Figures

Introduction

Overview

The idea for Predictive Restaurant Inventory Management System stems from a show called “The Bear”. Managing a kitchen at a restaurant can be really hectic and the last thing anyone would want is to run out of ingredients! Or throw away spoiled ingredients, like fruits and vegetables, because of their low shelf life.

However, if the future orders are predicted, based on the history of orders, then the optimal amount of ingredients needed for the upcoming time period (a week or a couple of days) can be estimated. With the help of this estimate, the food inventory of a Restaurant would never be understocked or overstocked. Not only would it save cost, but also reduce wastage and prevent stockouts. If the ordering of raw ingredients from the suppliers can be automated, it’ll save time as well.

The Predictive Restaurant Inventory Management System, aka PRIMS, would do just that. We are going to achieve this by implementing a machine learning model that will be trained using a historical restaurant dataset and run periodically to make predictions. According to the predicted number of orders (e.g., number of pasta dishes), the system would automatically place orders for the required number of ingredients (e.g., pasta noodles, cheese, tomatoes, garlic, etc.) by taking the current inventory levels into account. If there aren’t enough ingredients, more would be ordered. After ordering, the inventory would be updated to reflect the new values.

Furthermore, the updated inventory values and the current predictions, along with the existing data, would be used in the next iteration, i.e., when more orders are placed and the machine learning model runs. This cycle will keep on repeating unless stopped externally.

Finally, a UI would display real-time statuses like inventory level, predicted demand, current orders, and model performance.

Goals

1. **Demand Forecasting:**
- Use historical order data to predict future demand for menu items.
- Implement a time-series machine learning model like ARIMA, SARIMA, Prophet, Holt-Winters, etc.
2. **Inventory Optimization:**
- Design an algorithm to calculate the optimal inventory levels based on predicted demand and current stock.
3. **Automated Ordering System:**
- Automate supplier order placements based on predicted shortages.
4. **Real-Time Processing:**
- Implement a method to simulate real-time orders that match the historic dataset's trend.
- Develop a system to monitor and update inventory levels in real-time as ingredients are used and restocked
5. **User-Friendly Interface:**
- Build a dashboard/UI to display:
    - Current inventory levels
    - Predicted demand per dish/item
    - Model performance over time
    - Actual orders vs predicted orders

Requirement Specification

The proposed system aims to streamline restaurant inventory management by leveraging advanced technologies and automation. By integrating data management, machine learning, and intuitive interfaces, the solution seeks to address key challenges such as demand forecasting, inventory optimization, and reducing expenses. Functional requirements ensure efficient handling of core activities like data storage, demand prediction, real-time order simulation and automation of inventory restocking. Non-functional requirements emphasize scalability, reliability, and usability to ensure the system performs efficiently and meets user expectations. Technical requirements outline the framework and tools necessary for the development and deployment of the solution, ensuring alignment with modern standards and best practices.

Functional Requirements

1. **Data Management**
   - Store historical order data, inventory records, recipe details, and ingredient information in a centralized database.
2. **Machine Learning Integration**
   - Use trained model's outputs to drive inventory calculations.
   - Facilitate tracking and evaluation of the model's prediction performance over time.
3. **Inventory Logic**
   - Set minimum inventory levels for each ingredient to meet the next week's predicted demand based on the machine learning model's output.
   - Start the process - week 1 - with a full inventory.
   - Set a threshold value of ingredient units to be added to the predicted amount for restocking to account for errors in the prediction. The value should be the standard deviation of the number of orders per week.
4. **Automation**
   - Adjust inventory levels and based on predictions for the upcoming week's demand.
   - Generate real-time orders that match the historical data patterns. Implement the simulation process using the method 'blocked bootstrap'.

Non - Functional Requirements

1. **Scalability**
    - Support scaling to handle data from multiple restaurant branches.
    - Accommodate increased menu complexity and diverse ingredient types.
2. **Reliability**
    - Ensure high uptime and real-time inventory tracking.
    - Allow for dynamic machine learning model switching.
3. **Performance**
    - Forecast demand with minimal latency to enable timely decision-making.
    - Enable real-time inventory updates for seamless operations.
    - User-interface to be updated dynamically to display real-time order simulation, predictions, and model performance.
4. **Usability**
    - Provide an intuitive and visually appealing user-interface.
    - Enable process initialization and model switching via the user-interface.

Technical Requirements

1. **Technologies**
    - **Backend**: Flask (a Python package) for API development and XAMPP for a web server.
    - **Frontend**: HTML, CSS and JavaScript for an interactive UI and dashboard.
    - **Database**: MySQL for relational data and real-time inventory updates.
    - **Machine Learning**: Python.
    - **Deployment and Version Control**: Github

Design & Implementation

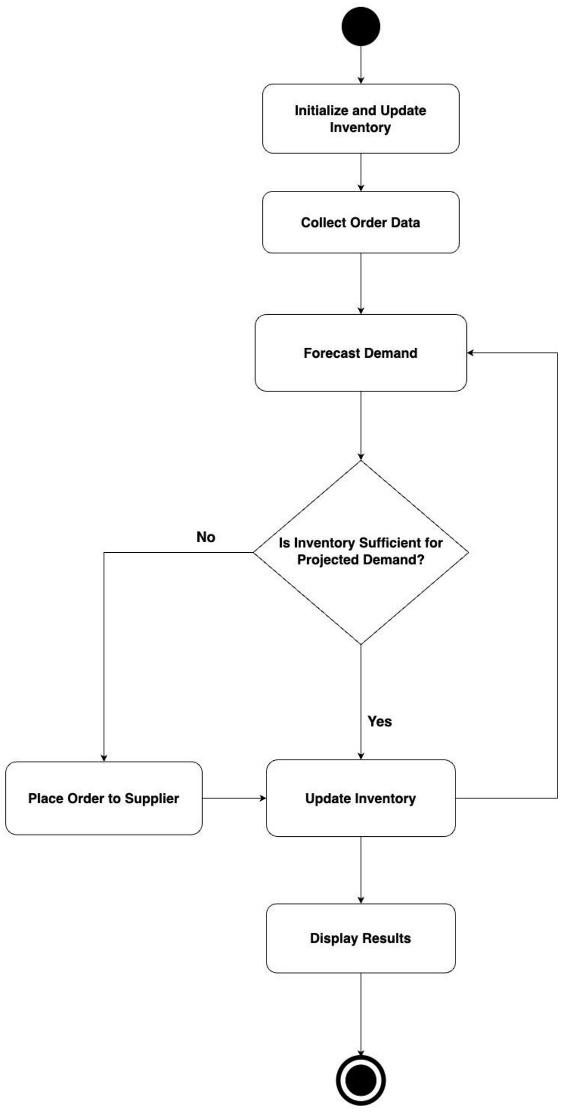

Fig. 1 : Activity Diagram

PRIMS employs a modular design to facilitate scalability, robustness, and ease of operation. The primary components include inventory management, data collection, demand forecasting, decision-making, and user interaction modules. These components work in cohesion to create a seamless and efficient inventory management cycle.

1. **Inventory Initialization and Management**:
   The system begins by initializing the inventory to a maximum level. All the ingredients are fully stocked. It serves as the foundation for demand forecasting and subsequent decision-making.
2. **Data Collection and Processing**:
   A data acquisition pipeline gathers historical and real-time order data which is simulated to represent the historical data's trend. Historical data is used for training the machine learning model, while the simulated orders represent real-world orders. Simulated orders are also used to calculate the model performance. Pre-processing historical data ensures data quality, handles inconsistencies, and prepares it for model building.
3. **Demand Forecasting**:
   The forecasting module is based on exploratory analysis of the historical dataset. After identifying the trend and seasonality of the data, 2 SARIMA (Seasonal Auto Regressive and Moving Average) and 1 Holt-Winters Exponential Smoothing machine learning models are applied to it. The trained models are used by the forecasting class to predict upcoming week's orders.
4. **Inventory Adjustment and Updates**:
   The inventory update module compares forecasted demand against current inventory levels to determine stock sufficiency. A predefined ingredient threshold acts as a buffer to the predicted inventory levels. This buffer is meant to account for errors in the model performance because no model can be 100% accurate; there's always randomness. The buffer value is set to be one standard deviation of orders per week. If inventory is predicted to be insufficient, the system triggers replenishment. The module also uses the simulated real-world orders to deplete the inventory.
5. **Real-time Order Simulation**:
   The order simulation module uses a method called blocked bootstrap to simulate real-world orders. This method accounts for randomness of the real-world as well as patterns in the existing restaurant dataset (i.e., historical dataset). The dataset already has a lot of randomness and is therefore used to generate real-word orders as well. Random weekly blocks are chosen from the dataset - to preserve the weekly seasonality - and a value is randomly chosen from that block to simulate total number of orders for a day. The simulated orders are used to deplete the inventory as well as to evaluate the model performance over time.
6. **User Interface**:
   A comprehensive and a user-friendly dashboard provides information about the current state of the system. It dynamically fetches updated values to show the real-time process to users. The UI shows the current week number, current inventory levels (ingredient counts), actual orders, predicted orders, restocked ingredient units, and a drop-down menu to choose amongst the three trained

models. Furthermore, it also displays graphs depicting predicted vs actual orders, and the model performance over time.

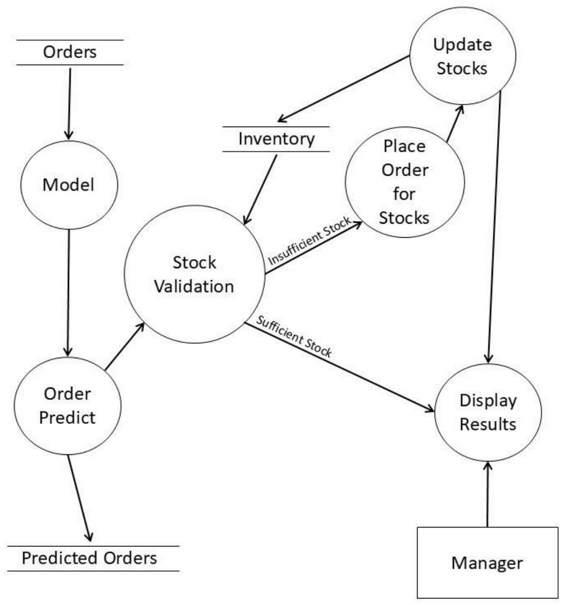

Fig. 2 : Data Flow Diagram

Machine Learning Model Overview

This section briefly explains about the machine learning models chosen for this project and the criteria used for selecting the optimal model.

Machine learning is statistical learning or to be more precise, it is a method of learning patterns from data using statistical learning. The following steps summarize the process of applying a machine learning model to a dataset:

1. **Identifying patterns**: The dataset is analysed to identify patterns between variables, i.e., columns. For example, in this project, the analysis focused on how the daily orders changed over time, where the Y-axis represents the daily order numbers and the X-axis represents the time. This is also called exploratory analysis.
2. **Choosing a statistical model**: Once a general pattern is identified, a statistical equation is applied to describe it. A simple example of such an equation is a straight line, where the parameters - the slope and the intercept - are estimated using the available data.
3. **Make predictions:** After estimating the parameters of the model/equation, it can be used to predict values for unseen data. In this project, the model predicts the number of daily orders for future dates based on historical data.

In short, machine learning is about using the available data to discover patterns, fit statistical models to it, and use them to make predictions!

The models where we want to find the behaviour of a variable over time are called Time Series models.

Model Selection Criteria

Before describing the models selected for this project, a brief introduction to the selection criteria is essential. Time series forecasting - predicting the behaviour of one variable with respect to time - generally requires identifying “trend” and “seasonality” in the data before applying a model.

Trend: A trend refers to the long-term direction in the data, showing whether the values are increasing, decreasing, or remaining stable over time. For example, a restaurant’s daily orders might show a gradual increase over months due to growing popularity.

Seasonality: Seasonality refers to repeating patterns or fluctuations in the data that occur at regular intervals, such as daily, weekly, or yearly. For instance, higher orders might occur on weekends compared to weekdays, reflecting a weekly seasonality.

Trend and seasonality can be identified in different ways. The methods chosen in this project are:

1. Plotting the number of daily orders vs date using the restaurant dataset:

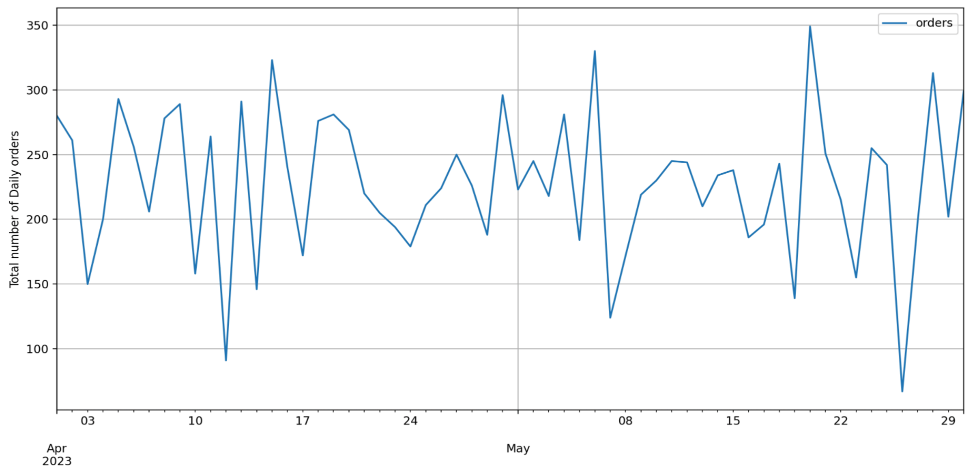

Fig. 3 : Graph of daily orders

   The above graph is for two months worth of data. It shows that there are weekly spikes, on average. Hence, the seasonality can be thought of as 7 days - pattern repeats after every 7 days (Higher order on weekends, for example).

   The trend seems to be stationary, i.e., there is no overall increase or decrease in the number of daily orders. The spikes are around a constant average.

   Note: this graph represents a subset of the data. Entire data was analysed and the same pattern was observed.

2. ADF test: The Augmented Dickey-Fuller test is a statistical test used to check if a time series data is stationary or not. A time series is said to be stationary if its statistical properties like mean and variance do not change over time. This means that its **trend** is stationary. ADF test uses the null hypothesis and p-value to test if a time series data is stationary or not. If the p-value (probability) of this test is less than 0.05 then one can reject the null hypothesis which states that the time series is stationary. This means that there is 95% certainty that the data is stationary and only a 5% chance that it isn't.

   The ADF test's p-value for this project's dataset is 3.531 x 10^{-7}. The value is significantly lower than 0.05.

3. Seasonal decomposition: A final statistical tool (available as a Python package) is also used to get a good visual idea about stationarity and trend. Seasonal decomposition splits a series into individual components - trend, seasonality, and residuals (randomness that doesn't follow a pattern). It plots a single graph using

the individual split components. As per the below mentioned graph, the trend of the data is stationary with strong seasonality and noise (randomness).

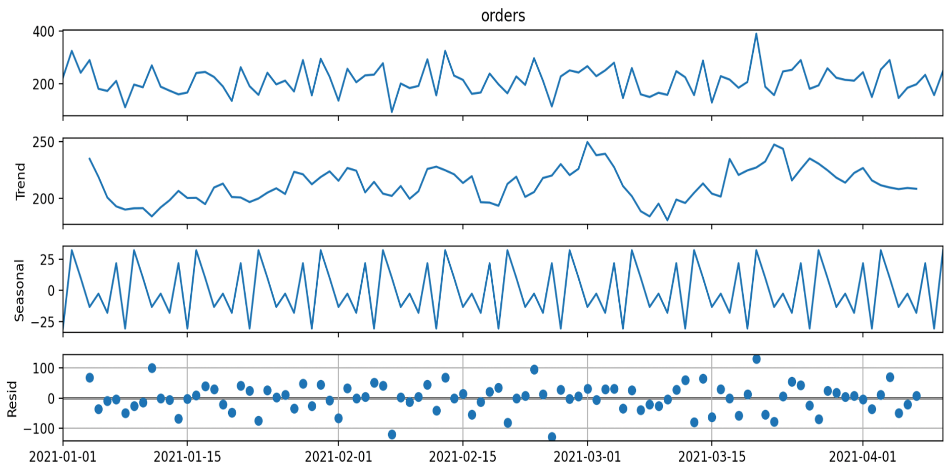

Fig. 4 : Analyzing seasonality and trend in data

Shortlisted Models

Based on the exploratory data analysis and model selection criteria, a time series machine learning model which can efficiently handle seasonal data would be the best for predictions in this case.

It narrowed down the search to two models - SARIMA and Holt-Winters Exponential Smoothing.

SARIMA

SARIMA is an acronym that stands for Seasonal Auto-Regressive Integrated Moving Average mode. The following describes each component of the model:

1. Seasonal: It accounts for repeating patterns in the data, i.e., seasonality. It can be daily, yearly, monthly, etc.
2. Auto-Regressive: The AR component models the relationship between the current data point and its past values, i.e., how correlated the data is with itself over time.
3. Integrated: It makes the data stationary by removing any upward or downward trends.
4. Moving Average: It models the dependency between the current data point and past prediction errors. It helps capture short-term noise in the data.

Each of the above-mentioned components represents an integer value. For example, the integrated value for this model will be 0 for this project because the restaurant dataset is stationary, as per the analysis mentioned in the previous section. A 0 value means that there is no need to make the data stationary. Stationarity is desired because it makes the predictions easier and more accurate. Furthermore, the seasonality component of the model is 7 in this case. This is because there are weekly order spikes in the dataset - the demand is slightly higher over the weekends. 7 represents seven days.

Deciding the AR and MA values is trickier. A good starting point for these values are the Auto-Correlation and Partial Auto-Correlation plots (ACF and PACF).

The PACF plot helps determine the value of the Auto Regressive component of the model, while the ACF helps determine the Moving Average component:

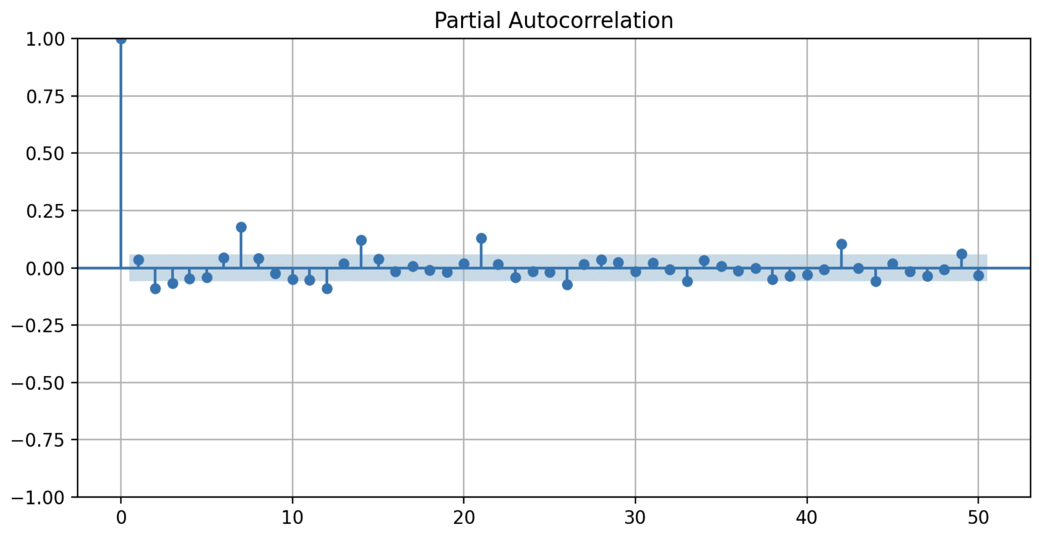

Fig. 5 : Partial Auto Correlation plot

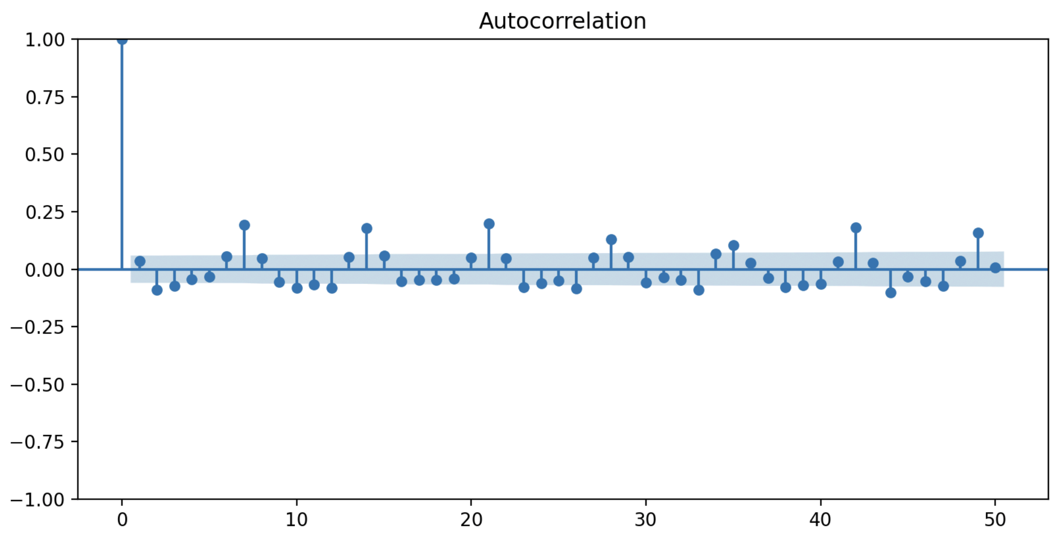

Fig. 6 : Auto Correlation plot

The number bulbs that reach outside of the blue shaded area represent the number of past values that are correlated to the current value in the data. Only a few initial spikes have to be taken into account. Afterwards, the occasional spikes are due to the repeating trend (seasonality) in the data. As per the above two graphs, around 2 to 3 bulbs (excluding the first tallest one) initial bulbs are outside the shaded area or slightly touching it. Hence, as per these graphs, the optimal MA and AR values would be between 0 to 3.

Finally, the ACF and PACF graphs only provide a rough idea of the model parameters. The best way is to try all the combinations of the 4 component values, within a specified range for each component, and choose the combination that gives the least error (error criteria is explained in the next section). For example, as per the initial analysis of the data, the best MA and AR component of the SARIMA model would be an integer between 0 to 3, the best Integrated component value is 0, and the optimal Seasonality value is 7.

Note: Two different SARIMA models have been trained using the data. Both of them are the same models but their performance measurement criteria, i.e., the error criteria is different. This will be explained in the next section.

The parameters of the best SARIMA models based on different error evaluation criteria:

S = 7, AR = 2, I = 0, MA = 1 (root mean squared error criteria)

S = 7, AR = 0, I = 0, MA = 0 (Akaike information criterion (AIC))

The first model is named as “sk_sarima” and the second as “auto_sarima” in this project.

Holt-Winters Exponential Smoothing

Holt-Winters Exponential Smoothing is another time series forecasting model which is used for predictions when the data has **trend** and **seasonality**. Holt-Winters model uses three components to smooth the data and identify patterns:

1. Level (base value): This is the baseline value of the data at current time. It captures where the data is “on average” right now.
2. Trend: As explained above, trend represents the direction of the data over time - upward, downward, or stationary.
3. Seasonality: As explained in the previous section, seasonality is the recurring pattern(s) in the data.

There are two main types of this model:

1. Additive: Used then the seasonality of the data has a constant magnitude over time. For example, the weekend restaurant orders increase by 100 units on average, regardless of the overall growth.
2. Multiplicative: Used when the seasonality increases or decreases in magnitude over time. For example, during winter break the weekend restaurant order values increase by 200 units rather than 100 units on average.

In this project the **additive** version of this model has been implemented because the magnitude of the seasonality in the data stays constant over time. The model has been evaluated using the **root mean squared error** criteria.

Model Evaluation Criteria

The three models (2 SARIMA and 1 Holt-Winters) described above have been evaluated using two different methods - RMSE and AIC. Based on the values of these two functions, the best one - “sk_sarima” - has been selected.

Root Mean Squared Error

A model is “fit” or applied to an existing dataset or a part of it called the training dataset. Using the training dataset’s values the model gets trained and gives its estimated (predicted) values of the outcome variable - daily orders in this case. These estimated values are compared with the actual observed values to assess how big the difference is. Root mean squared error or RMSE for short is a metric to measure how well a model’s predictions match the actual values.

It’s formula is: $\sqrt{\frac{\sum_{i=1}^{n} (yi - yi\hat{})^2}{n}}$

Where yi is the observed ith value and yi^ is the predicted ith value, given that there are total n data points, i.e., rows in the dataset. The difference between the observed and predicted value is squared so that the negative and positive values don’t cancel each other out while calculating the average of the differences. The average is then squared to flatten out the squared differences.

Akaike Information Criterion

AIC is a measure used to compare different models and determine which ones best fits the data while avoiding overfitting. A lower AIC score indicates a better model.

When building machine learning models, adding more parameters to a model (increasing the complexity) makes the model fit to the training dataset really well. However, it doesn’t perform well on unseen data. This is because the model was trained “too well” and it cannot handle values that it hasn’t seen before. Hence, AIC penalizes models with too many parameters.

Formula for AIC is: 2k - 2ln(L)

Where k is the number of parameters in the model, L is the likelihood (probability) function of the model, and ln is log to the base e.

AIC metric has been used as another criteria in choosing the best SARIMA model. As a result, two SARIMA models have been built as part of this project - one according to the best RMSE value and the other as per the best AIC value.

Verdict

All three models perform approximately the same with minor differences. The best fitted model came out to be sk_sarima with the lowest RMSE value:

RMSE values:

Sk_sarima - 56.757423

Auto_sarima - 70.811668

Holt_winters - 89.596655

Auto_sarima performs the best as per the AIC score, however, as per the graph of actual order values vs predicted values against time mentioned below, sk_sarima model's predictions seem slightly better. This is because they match the high order value peaks more than the auto_sarima model and don't match lowest order values too well, as compared to the auto_sarima model. For a restaurant's use case, this is more desirable because having slightly excess ingredients in the inventory is better than having an insufficient amount.

The Holt-Winters model performs slightly worse than the other two as its RMSE is the highest.

In conclusion, with sk_sarima giving the best performance, the other two models are very similar. As a result, the project's user-interface offers an option to select any model among the three for predictions, with the default being sk_sarima.

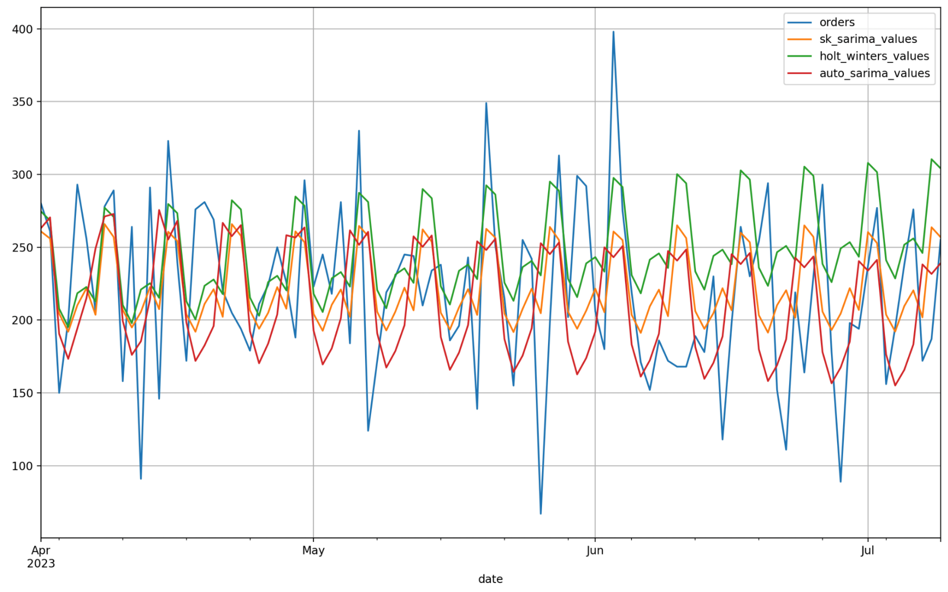

Fig. 7 : Combined Prediction Plots of models

User Guide

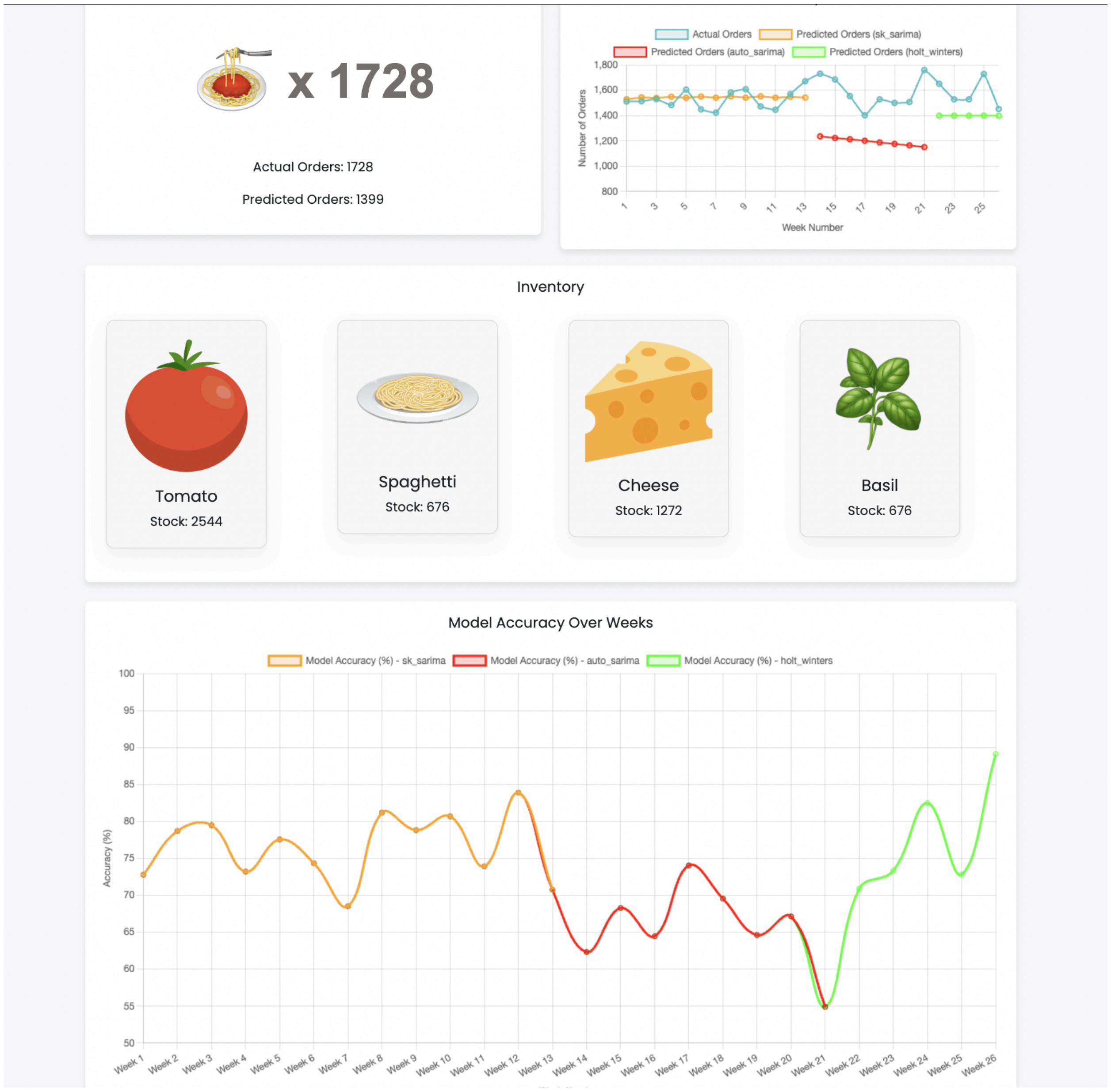
x 1728
Actual Orders: 1728
Predicted Orders: 1399
Actual Orders
Predicted Orders (sk_sarima)
Predicted Orders (auto_sarima)
Predicted Orders (holt_winters)
Number of Orders
Week Number
Inventory
Tomato
Stock: 2544
Spaghetti
Stock: 676
Cheese
Stock: 1272
Basil
Stock: 676
Model Accuracy Over Weeks
Model Accuracy (%) - sk_sarima
Model Accuracy (%) - auto_sarima
Model Accuracy (%) - holt_winters
Accuracy (%)

Fig. 8 : System UI

This guide provides a step-by-step walkthrough to set up and run the Predictive Restaurant Inventory Management System (PRIMS). It assumes the prerequisites are met and outlines the folder structure, common issues, and troubleshooting tips.

Prerequisites

To ensure the system runs smoothly, the following prerequisites must be installed on your machine:

1. **Python 3.x**:
   - Download and install the latest version from python.org.
   - Verify the installation by running `python --version` or `python3 --version`.
2. **Git**:
   - Install Git from git-scm.com.
   - Verify by running `git --version`.

Steps to Set Up PRIMS

Step 1: Clone the Repository

Clone the repository to your local machine using Git:
bash
Copy code

```
git clone
https://github.com/tramya16/predictive-restaurant-inventory.git
cd predictive-restaurant-inventory
```

Step 2: Set Up a Virtual Environment (Optional but Recommended)

A virtual environment helps manage dependencies and isolate your project environment.

On Windows:

```
python -m venv .venv
.venv\Scripts\activate
```

On macOS/Linux:

```
python3 -m venv .venv
source .venv/bin/activate
```

Step 3: Install Dependencies

Install the required Python packages listed in `requirements.txt`:

Using `pip` (on Windows):

```
pip install -r requirements.txt
```

Using `pip3` (on macOS/Linux):

```
pip3 install -r requirements.txt
```

Step 4: Adding models and historical data

1. Add machine learning models:
    - Go to https://drive.google.com/drive/folders/1L-UlaPMUdl_u0AEjKQIcE9TReMveopUS and download all the '.pkl' files. These are the trained models.
    - Place the '.pkl' files into the app/models/ directory of this project.
2. Add the historical dataset:
    - Go to https://drive.google.com/drive/folders/1L-UlaPMUdl_u0AEjKQIcE9TReMveopUS and download the 'historical_data.csv' file.
    - Place the file in the home directory of your system. For example - /Users/lakshya in MacOS, C:\Users\lakshya in Windows, and /home/lakshya in Linux.

Step 5: Run the Flask Application

Start the Flask application to launch the PRIMS dashboard:

On Windows:

```
python app/app.py
```

On macOS/Linux:

```
python3 app/app.py
```

Open your browser and visit:
http://127.0.0.1:5000/

Folder Structure

The project is organized as follows:

```
predictive-restaurant-inventory/

├── app/
│   ├── app.py            # Main Flask application
│   ├── static/           # Static files (CSS, JS, images)
│   └── templates/        # HTML templates
├── documentation/
│   └── images/           # Folder for documentation images (e.g., for
reports or presentations)
├── .venv/                # Virtual environment (optional)
├── requirements.txt      # Python dependencies
└── README.md             # General project instructions
```

Troubleshooting

Flask Command Not Found:
If the `flask` command is unavailable, use the Python command instead:
```
python app/app.py
```

Virtual Environment Not Activating:

Ensure the correct activation command is used for your operating system.If issues persist, check the Python installation path or ensure `venv` is installed with Python.

Missing Dependencies:
If errors arise due to missing packages, re-run:
```
pip install -r requirements.txt
```

Notes

- Always activate the virtual environment before running or modifying the application.
- The `.venv/` directory is excluded from version control via `.gitignore` to prevent unnecessary file tracking.
- Regularly update the `requirements.txt` file if new dependencies are added.

Contributions

Akash Vijaykumar Kotekar (124106044)

Throughout the development of the Predictive Restaurant Inventory Management System (PRIMS), Akash Vijaykumar Kotekar made valuable contributions, focusing on user interface design, dataset creation, and optimization. His efforts were pivotal in ensuring the system was both functional and user-friendly.

Akash collaborated on designing and implementing the application's user interface (UI) using Flask. He prioritized creating an intuitive and visually appealing interface that catered to the needs of end users, particularly restaurant staff. His work ensured that the UI not only displayed information effectively but also facilitated seamless interaction with the system's various functionalities.

In addition to his contributions to the UI, Akash worked extensively on developing and optimizing the code for dataset creation. He focused on streamlining data collection and preprocessing, ensuring the dataset included all relevant features necessary for accurate predictions. His efforts enhanced the efficiency of the dataset pipeline, enabling the system to generate reliable insights based on historical and real-time data.

Akash's ability to bridge the gap between technical functionality and user-centric design was a key factor in the success of PRIMS. His expertise in Flask development and data optimization played an essential role in delivering a system that was both practical and accessible, making his contributions integral to the project.

Lakshya Suri (124101111)

As part of the PRIMS project, Lakshya's contributions were primarily directed towards formulating the general design, machine learning model building, and diagnostics. As mentioned in the abstract, the inspiration for the project came from the show Bear.

Before applying any statistical model to a dataset, an exploratory analysis was performed to examine the relationships between different variables (columns) of the dataset. This analysis can be conducted in several ways. For example, Lakshya checked how the daily number of orders behaved over time by plotting a graph. Additionally, he examined other factors such as: whether the average weekly order remained consistent throughout the dataset, the presence of any repeating patterns, and whether today's order value depended on past values.

Identifying these characteristics helped narrow down the vast number of machine learning models that could be applied to the dataset. After analyzing the dataset and gaining a general understanding, Lakshya's focus shifted to studying various time-series machine learning models. In a nutshell, time-series models are mathematical equations used to predict outcomes — in this case, weekly orders — over time.

Finally, three models were selected and trained using the historical daily orders dataset. The model-building code was designed to automatically select the best-performing model based on the lowest error value, measured using the root mean squared error (RMSE). After the model was trained and tested, predictions were generated.

A predictor script was developed to load the trained model (as the model is not trained at runtime) and use it to make predictions for a specified date range. The predictor script is parameter-driven and can select any of the three models. Additionally, the predictor class calculates the model's accuracy and average error value. Both metrics are stored in the database and displayed as a graph on the user interface. These metrics can also be used to retrain the model.

Mamtha Mahanthesh Vathar (124106172)

Mamtha made significant contributions to the development and management of the database system for the Predictive Restaurant Inventory Management System (PRIMS), focusing on creating a robust, efficient, and scalable solution.

Mamtha collaborated on designing and optimizing SQL queries to handle extensive datasets effectively. Her work ensured that data retrieval, updates, and manipulations were performed efficiently, supporting critical functions like calculating inventory restocking levels and maintaining system accuracy.

As part of the project's initial setup, Mamtha played a key role in migrating data from CSV files into the database. This laid the groundwork for subsequent processing and enabled the system to operate seamlessly. She also implemented mechanisms to dynamically update the database with real-time values, ensuring the data remained accurate and reflective of current conditions.

Mamtha worked extensively on integrating the database with Python scripts to enhance the system's functionality. By developing custom-built functions, she enabled smooth data operations such as reading, updating, and inserting data. Her work ensured the seamless integration of calculated results and model outputs into Flask applications, supporting a highly efficient back-end system.

In later stages, Mamtha extended her contributions to the user interface, connecting back-end processes to the front-end. Her efforts ensured that the outputs of predictive models were presented clearly and intuitively, delivering a user-friendly experience and contributing significantly to the system's success.

Mandar Deepak Bagwe (123111875)

Mandar played a critical role in designing, developing, and managing the database system for the Predictive Restaurant Inventory Management System (PRIMS). Collaborating closely with the team, he focused on ensuring that the database was robust, efficient, and seamlessly integrated with the application.

Mandar was instrumental in crafting efficient SQL queries to handle large datasets, enabling optimal performance. These queries facilitated critical operations such as retrieving, updating, and manipulating data to support essential calculations, including determining inventory restocking levels.

As part of the foundational setup, Mandar oversaw the migration of data from CSV files into the database, establishing a solid basis for further processing. He implemented dynamic database update mechanisms to ensure that the system remained current with real-time values generated during operation.

To streamline integration between the application and the database, Mandar embedded database connections into Python scripts. He developed custom functions to enable seamless reading, updating, and inserting of data, ensuring a reliable and efficient flow of information. These functions were also used to integrate calculated results and model outputs into Flask applications, enhancing the efficiency and functionality of the back-end system.

Mandar contributed to extending the database functionality to the user interface by connecting back-end logic to the front-end. His efforts ensured that the model's outputs were accurately displayed in an intuitive and user-friendly manner, enriching the overall user experience.

Meghan Vinayak Mane (124100669)

Throughout the development of the Predictive Restaurant Inventory Management System (PRIMS), Meghan Mane played a pivotal role in shaping the project. His contributions encompassed key areas, including planning, implementation, experimentation, and visualization.

Meghan actively participated in finalizing the project scope and defining its structural framework. He worked closely with the team to ensure that all components—forecasting, inventory management, and visualization—were logically connected and aligned with the project's objectives.

A significant area of his contribution involved researching and experimenting with machine learning models for demand forecasting. He implemented and tested the Long Short-Term Memory (LSTM) algorithm, focusing on preparing and preprocessing the dataset as well as hyperparameter tuning, including adjustments to learning rates and network configurations. While the LSTM model did not achieve the desired accuracy and was excluded from the final implementation, his experimentation provided valuable insights that informed other aspects of the project.

Meghan also played a key role in creating a high-quality dataset, which was critical to the project's success. He identified relevant data sources, ensured the dataset included essential features such as historical order patterns and seasonality, and structured it to support accurate model training and validation.

Additionally, he contributed to crafting the implementation strategy for the project, working with the team to outline steps for integrating predictive models into inventory workflows and automating processes like real-time inventory updates and order placements.

Moras Kashyap (124103079)

Moras Kashyap made significant contributions that were essential to the project’s progress and success. His work spanned critical areas such as planning, modeling, dataset preparation, and system implementation.

Moras was instrumental in defining the scope of the project and establishing its structural framework. Collaborating closely with the team, he ensured that the system’s core components—including forecasting, inventory management, and visualization—were seamlessly integrated and aligned with the project’s objectives.

A major focus of his efforts was researching and experimenting with statistical models for demand forecasting. He implemented and evaluated SARIMA and ARIMA models, emphasizing their suitability for time-series data. Moras worked extensively on preprocessing the dataset to ensure compatibility with these models and fine-tuned parameters such as seasonality, differencing, and autoregression to optimize their performance. His analysis played a critical role in refining the forecasting process, even though some models were excluded in favor of more accurate alternatives.

In addition to modeling, Moras played a vital role in developing the project’s dataset. He identified relevant data points, including historical order trends, seasonal variations, and external factors like promotions and holidays. His efforts in structuring and validating the dataset ensured it was robust and capable of supporting reliable predictions.

Moras also contributed to the project’s implementation strategy. He collaborated with the team to design workflows that integrated predictive models into inventory management processes. This included automating key operations, such as real-time stock updates and initiating replenishment orders based on forecasted demand.

Through his expertise in statistical modeling, data handling, and strategic planning, Moras Kashyap made indispensable contributions to PRIMS, helping to create a system that was both accurate and efficient in addressing the needs of restaurant inventory management.

Ramya Thimmappa Gowda (124106243)

Throughout the development of the Predictive Restaurant Inventory Management System (PRIMS), Ramya played an instrumental role in designing the user interface and fostering collaboration within the team. Her contributions established a strong foundation for the project and ensured its components were cohesive and user-friendly.

Ramya focused on developing and refining the user interface (UI), ensuring it was intuitive and aligned with project objectives. She designed a user-friendly interface, emphasizing functionality and ease of use for the restaurant staff. By building the initial application with a mocked process and basic UI, she provided a foundation for further development and integration, accelerating the team's progress.

To support backend development, Ramya set up skeleton code, establishing a clear and scalable structure for the project. This framework maintained consistency in coding practices and prevented development bottlenecks. Additionally, she managed the Git repository, centralizing the project codebase and streamlining collaboration. Her work ensured smooth version control and allowed the team to work efficiently without conflicts.

In the planning phase, Ramya contributed to creating activity and data flow diagrams, providing a blueprint for the team to follow. These diagrams clarified the system's functional flow and ensured a structured development process. She also worked closely with the team to integrate UI components with backend logic, ensuring smooth functionality and coherence across the application.

Beyond her technical contributions, Ramya facilitated team collaboration by fostering open communication and problem-solving. She provided valuable feedback on design decisions and helped the team stay aligned with project objectives.

Through her expertise in UI design, backend structuring, and collaborative development, Ramya significantly contributed to the success of PRIMS. Her efforts ensured the system was functional, user-friendly, and aligned with the project's goals.