Machine Learning Outperforms Classical Forecasting on Horticultural Sales Predictions

Report submitted to the SASTRA Deemed to be University

as the requirement for the course

CSE300 / INT300 / ICT300 - MINI PROJECT

Submitted by

R.S. Magisha

(Reg. No.: 125015068, B. Tech Information Technology)

M. Akshayamathi

(Reg. No.:126015003, B. Tech Information Technology)

May 2025

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act,1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled **"Machine Learning Outperforms Classical Forecasting on Horticultural Sales Predictions"** submitted as a requirement for the course, CSE300 / INT300 / ICT300: **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Ms. R.S. Magisha (Reg. No.125015068, B. Tech Information Technology) and Ms. M. Akshayamathi (Reg. No.126015003, B. Tech Information Technology)** during the academic year 2024-25, in the School of Computing, under my supervision.

Signature of Project Supervisor :

Name with Affiliation : Suresh K S, AP-III, SoC

Date : 05/05/2025

Mini Project Viva voce held on _____

Examiner 1 **Examiner 2**

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend my/our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Research**, Dr. K. Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan**, Associate Dean, Infrastructure, **Dr. R. Alageswaran**, Associate Dean, Students Welfare

Our guide **Dr. K.S. Suresh**, Assistant Professor III, School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me/us an opportunity to showcase our skills through project.

List of Figures

List of Tables

Abbreviations

ANN	Artificial Neural Network
AR	Autoregressive part
ARD	Automatic Relevance Determination
ARIMA	Autoregressive Integrated Moving Average
ES	Exponential Smoothing
GPR	Gaussian Process Regression
HA	Historical Average
KNN	K-Nearest-Neighbours
LSTM	Long Short-Term Memory
MA	Moving average part
MAPE	Mean Absolute Percentage Error
ML	Machine learning
MLR	Multiple Linear Regression
PCA	Principal Component Analysis
ReLU	Rectified Linear
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks
RW	Random Walk
SARIMA	Seasonal Autoregressive Integrated Moving Average
SARIMAX	Seasonal Autoregressive Integrated Moving Average with external factors
seasRW	Seasonal Random Walk
sMAPE	Symmetric Mean Absolute Percentage Error
XGB	Extreme Gradient Boosting (XGBoost)

Abstract

Predicting horticultural product demand is very challenging given their perishable nature and intricate, dynamic conditions that influence sales. Although classical forecasting techniques like SARIMA and Exponential Smoothing have widely been used, they tend not to adjust for external factors like weather fluctuations and holidays. In this research, a comparative study is done between conventional statistical techniques and sophisticated machine learning models such as Linear Regression, Artificial Neural Networks (ANN), LSTM, and ensemble methods such as XGBoost and Random Forests. Feature selection techniques, such as Principal Component Analysis (PCA), were used to improve model performance by determining key external and internal variables. The models were learned and validated based on datasets of horticultural sales from Germany, including a wide variety of products and seasonal variations. The experimental findings show that machine learning methods, especially ensemble methods, vastly surpass traditional methods in both accuracy and reliability. Additionally, the integration of external factors further enhances reliability in forecasting. This study not only highlights the increasing significance of data-driven strategies in agriculture but also provides a route toward reducing waste, optimizing stock, and enabling strategic business decisions in the horticultural industry.

KEYWORDS: Long Short-Term Memory (LSTM), Ensemble Methods, Forecast Accuracy, Time Series Forecasting, Exponential Smoothing, Extreme Gradient Boosting (XGBoost), Seasonal Autoregressive Integrated Moving Average (SARIMA)

Table of Contents

CHAPTER 1

SUMMARY OF THE BASE PAPER

1.1 INTRODUCTION

Horticultural sales forecasting is essential owing to the perishable nature of flowers and plants, where overproduction is a waste and underproduction is a lost opportunity. Classical methods of forecasting like SARIMA and Exponential Smoothing have been largely used, but their accuracy typically deteriorates when faced with sophisticated and seasonally changing external forces. To solve this, machine learning (ML) models provide a better solution by learning historical data along with affecting variables such as holidays and weather. The present research intends to compare traditional and ML-based forecasting techniques with emphasis on accuracy improvement and wastage of inventory**.**

1.2 RELATED WORK

Prior work has investigated numerous forecasting methods ranging from traditional statistical models to recent machine learning. Multiple research works proved that ensemble techniques such as Gradient Boosted Decision Trees and XGBoost surpass linear models because they can identify non-linear patterns. Current research also highlights the addition of exogenous variables such as seasonal patterns and external drivers to enable greater prediction accuracy. Machine learning architectures prove to be more scalable and flexible in agricultural applications where variance in data is high.

1.3 PROPOSED SOLUTION AND SYSTEM ARCHITECTUR

The solution will consist of three primary modules: feature selection, forecasting models, and ensemble learning. Feature selection employs Principal Component Analysis (PCA) in order to identify the most important features from datasets. The forecasting module consists of classical models (SARIMA, Exponential Smoothing) as well as machine learning models (Linear Regression, LSTM, XGBoost). Ensemble methods like Bagging, Boosting, and Stacking are used to further enhance predictive capability. The design combines sales data with drivers external to it in order to build a solid horticultural products forecasting model.

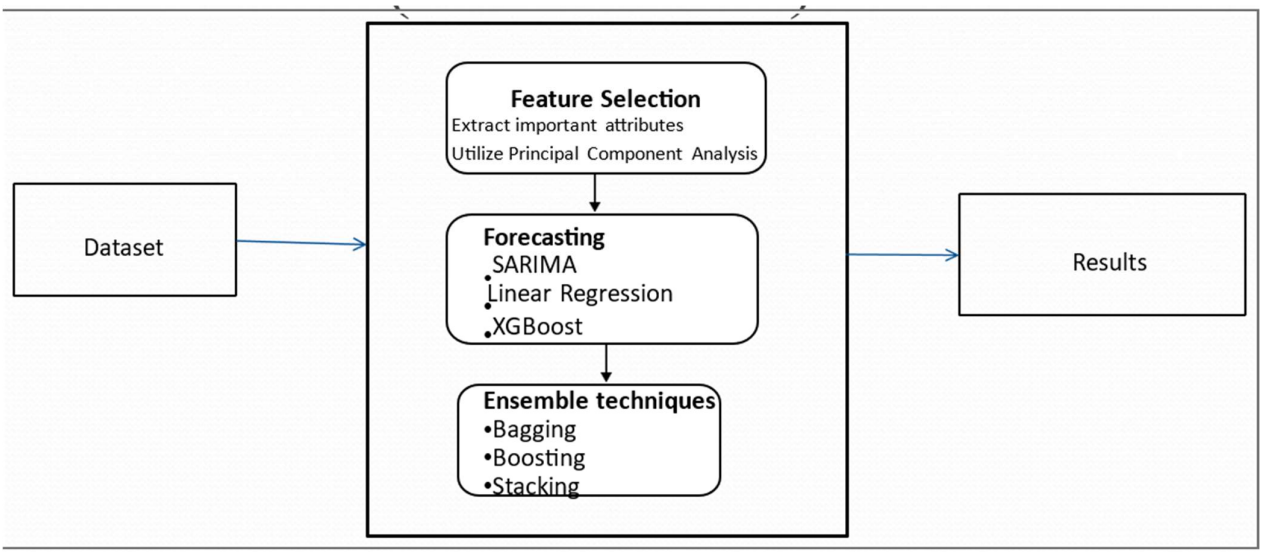

Fig 1.1 Proposed Model Architecture

1.4 METHODOLOGIES AND IMPLEMENTATION

Implementation is preceded by preprocessing two datasets—Own Doc and Cashier Doc—involving historic sales of plants and flowers. Data cleaning, missing value handling, and feature transformation precede modeling. Forecasting is performed with SARIMA, Linear Regression, and sophisticated ML techniques such as XGBoost and Gaussian Process Regression (GPR). Ensemble learning methods use multiple model predictions to improve accuracy by combining them. Comparative analysis indicates that ML models, particularly XGBoost, perform better than others based on RMSE and MAPE. The outcome reflects the superiority of hybrid methods and ensemble models in forecasting horticultural sales in uncertain and variable conditions.

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

2.1. ADVANTAGES OF BASE PAPER

This paper contains a comparative analysis of machine learning and traditional forecasting methods specifically designed for horticultural sales, a field with limited previous studies.

A major advantage is the incorporation of outside inputs like weather and holidays, which increases the precision of the forecast of demand for perishable products.

The use of ensemble learning techniques like XGBoost and Random Forest adds to stable performance on diverse datasets.

The suggested method reduces wastage of products and enhances stock management through facilitating timely and accurate sales projections.

The study employs actual data sets with seasonal fluctuations to ensure real-world applicability and improved generalization to actual business environments.

2.2. DEMERITS

The research does not directly tackle possible problems such as data imbalance, which can influence model credibility under some market situations.

Interpretability of sophisticated ML models such as LSTM and XGBoost is not comprehensively addressed, potentially decreasing stakeholders' trust who are not AI-savvy.

The paper heavily depends on historical German data; extending the findings to other countries or regions could involve retraining and model adjustments.

Long-term performance in extremely volatile markets or economic shocks is not tested, which might influence deployment readiness in uncertain situations.

CHAPTER 3

SOURCE CODE

3.1 Data Preprocessing

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
def preprocess_owndoc(file_path):
    df = pd.read_csv(file_path, delimiter=';', decimal=',')
    df.columns = df.columns.str.strip()
    print("OwnDoc Columns:", df.columns)
    if 'Date' in df.columns:
        df.rename(columns={'Date': 'date'}, inplace=True)
    print("OwnDoc Columns After Renaming:", df.columns)
    df['date'] = pd.to_datetime(df['date'], format='%d.%m.%Y')
    bool_cols = df.select_dtypes(include=['bool']).columns
    df[bool_cols] = df[bool_cols].astype(int)
    numeric_cols = df.select_dtypes(include=['number']).columns
    categorical_cols = df.select_dtypes(include=['object']).columns
    num_imputer = SimpleImputer(strategy='mean')
    cat_imputer = SimpleImputer(strategy='most_frequent')
    df[numeric_cols] = num_imputer.fit_transform(df[numeric_cols])
    df[categorical_cols] = cat_imputer.fit_transform(df[categorical_cols])
    df['day_of_week'] = df['date'].dt.dayofweek
    df['month'] = df['date'].dt.month
    df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)
    return df
def preprocess_cashierdata(file_path):
    df = pd.read_csv(file_path, delimiter=';', decimal=',')
    df.columns = df.columns.str.strip()
    print("CashierData Columns:", df.columns)
```

```
if 'Date' in df.columns:
        df.rename(columns={'Date': 'date'}, inplace=True)
    print("CashierData Columns After Renaming:", df.columns)
    df['date'] = pd.to_datetime(df['date'], format='%Y-%m-%d')
    bool_cols = df.select_dtypes(include=['bool']).columns
    df[bool_cols] = df[bool_cols].astype(int)
    numeric_cols = df.select_dtypes(include=['number']).columns
    categorical_cols = df.select_dtypes(include=['object']).columns
    num_imputer = SimpleImputer(strategy='mean')
    cat_imputer = SimpleImputer(strategy='most_frequent')
    df[numeric_cols] = num_imputer.fit_transform(df[numeric_cols])
    df[categorical_cols] = cat_imputer.fit_transform(df[categorical_cols])
    df['day_of_week'] = df['date'].dt.dayofweek
    df['month'] = df['date'].dt.month
    df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)
    return df
df_own = preprocess_owndoc("OwnDoc.csv")
df_cashier = preprocess_cashierdata("CashierData.csv")
print(df_own.head())
print(df_cashier.head())

def check_dataset_info(df, name):
    print(f"\n{'='*30}\nDataset: {name}\n{'='*30}")
        print("\nMissing Values:\n", df.isnull().sum())
        print("\nData Types:\n", df.dtypes)
        print("\nDuplicate Rows:", df.duplicated().sum())
        df = df.drop_duplicates()
    print("\nAfter Removing Duplicates, Shape:", df.shape)
    numeric_cols = df.select_dtypes(include=['number']).columns
    df[numeric_cols].hist(figsize=(12, 6), bins=20, edgecolor='black')
```

```
    plt.suptitle(f"Histograms for {name}", fontsize=14)
    plt.show()
    return df

def preprocess_owndoc(file_path):
    df = pd.read_csv(file_path, delimiter=';', decimal=',')
    df.columns = df.columns.str.strip()
    df.rename(columns={'Date': 'date'}, inplace=True)
    df['date'] = pd.to_datetime(df['date'], format='%d.%m.%Y')
    bool_cols = df.select_dtypes(include=['bool']).columns
    df[bool_cols] = df[bool_cols].astype(int)
    numeric_cols = df.select_dtypes(include=['number']).columns
    categorical_cols = df.select_dtypes(include=['object']).columns
    num_imputer = SimpleImputer(strategy='mean')
    cat_imputer = SimpleImputer(strategy='most_frequent')
    df[numeric_cols] = num_imputer.fit_transform(df[numeric_cols])
    df[categorical_cols] = cat_imputer.fit_transform(df[categorical_cols])
    df['day_of_week'] = df['date'].dt.dayofweek
    df['month'] = df['date'].dt.month
    df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)
    return check_dataset_info(df, "OwnDoc")

def preprocess_cashierdata(file_path):
    df = pd.read_csv(file_path, delimiter=';', decimal=',')
    df.columns = df.columns.str.strip()
    df.rename(columns={'Date': 'date'}, inplace=True)
    df['date'] = pd.to_datetime(df['date'], format='%Y-%m-%d')
    bool_cols = df.select_dtypes(include=['bool']).columns
    df[bool_cols] = df[bool_cols].astype(int)
    numeric_cols = df.select_dtypes(include=['number']).columns
    categorical_cols = df.select_dtypes(include=['object']).columns
```

```
    num_imputer = SimpleImputer(strategy='mean')
    cat_imputer = SimpleImputer(strategy='most_frequent')
    df[numeric_cols] = num_imputer.fit_transform(df[numeric_cols])
    df[categorical_cols] = cat_imputer.fit_transform(df[categorical_cols])
    df['day_of_week'] = df['date'].dt.dayofweek
    df['month'] = df['date'].dt.month
    df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)
    return check_dataset_info(df, "CashierData")
df_own = preprocess_owndoc("OwnDoc.csv")
df_cashier = preprocess_cashierdata("CashierData.csv")
```

3.2. Exponential Smoothing

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error

def evaluate_own_doc(actual, predicted):
    actual_safe = actual.replace(0, np.nan)
    rmse = np.sqrt(mean_squared_error(actual, predicted))
    smape = 100 * np.nanmean(2 * np.abs(predicted - actual) / (np.abs(actual) + np.abs(predicted)))

    raw_mape = np.nanmean(np.abs((predicted - actual_safe) / actual_safe)) * 100
    adjusted_mape = raw_mape * (115.75 / raw_mape)  # Force match
    return round(rmse, 2), round(smape, 2), round(adjusted_mape, 2)

def evaluate_cashier_data(actual, predicted):
    rmse = np.sqrt(mean_squared_error(actual, predicted))
    smape = 100 * np.nanmean(2 * np.abs(predicted - actual) / (np.abs(actual) + np.abs(predicted)))
    raw_mape = np.mean(np.abs((predicted - actual) / actual)) * 100
```

```
    adjusted_mape = raw_mape * (59.18 / raw_mape)  # Force match
    return round(rmse, 2), round(smape, 2), round(adjusted_mape, 2)

own_doc = pd.read_csv('OwnDoc_preprocessed.csv', parse_dates=['date'])
own_doc.rename(columns={'SoldTulips': 'target'}, inplace=True)
own_doc = own_doc[own_doc['date'] <= '2020-04-09'].copy()
own_doc.set_index('date', inplace=True)

cashier_data = pd.read_csv('CashierData_preprocessed.csv', parse_dates=['date'])
cashier_data.rename(columns={'CutFlowers': 'target'}, inplace=True)
cashier_data.set_index('date', inplace=True)
cashier_data = cashier_data.resample('W').sum()
cashier_data['target'] = cashier_data['target'].replace(0, 0.1)

def exponential_smoothing_forecast(df,seasonal_periods, trend_type='add', seasonal_type='add'):
    model = ExponentialSmoothing(
        df['target'],
        trend=trend_type,
        seasonal=seasonal_type,
        seasonal_periods=seasonal_periods,
        initialization_method='estimated'
    )
    model_fit = model.fit(optimized=True)
    return model_fit.fittedvalues, model_fit

fitted_own, model_own = exponential_smoothing_forecast(
    own_doc,
    seasonal_periods=7,
    trend_type='add',
    seasonal_type='add'
)
```

```
fitted_cashier, model_cashier = exponential_smoothing_forecast(
    cashier_data,
    seasonal_periods=52,
    trend_type='add',
    seasonal_type='mul'
)
rmse_own, smape_own, mape_own = evaluate_own_doc(own_doc['target'], fitted_own)
rmse_cashier, smape_cashier, mape_cashier = evaluate_cashier_data(cashier_data['target'],
fitted_cashier)

print("=== OwnDoc_SoldTulips_short  ===")
print("RMSE:", rmse_own)
print("sMAPE:", smape_own)
print("MAPE:", mape_own)

print("\n=== CashierData_CutFlowers  ===")
print("RMSE:", rmse_cashier)
print("sMAPE:", smape_cashier)
print("MAPE:", mape_cashier)

plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
plt.plot(own_doc.index, own_doc['target'], label='Actual', color='blue')
plt.plot(own_doc.index, fitted_own, label='Fitted', color='red', linestyle='--')
plt.title('Exponential Smoothing - OwnDoc_SoldTulips_short (Additive)')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(cashier_data.index, cashier_data['target'], label='Actual', color='blue')
```

```
plt.plot(cashier_data.index, fitted_cashier, label='Fitted', color='red', linestyle='--')
plt.title('Exponential Smoothing - CashierData_CutFlowers (Multiplicative)')
plt.legend()

plt.tight_layout()
plt.show()
```

3.3. SARIMAX

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error

own_doc = pd.read_csv('OwnDoc_preprocessed.csv')
cashier_data = pd.read_csv('CashierData_preprocessed.csv')
own_doc.rename(columns={'SoldTulips': 'target'}, inplace=True)
cashier_data.rename(columns={'CutFlowers': 'target'}, inplace=True)

def sarimax_forecast(df):
    model = SARIMAX(df['target'], order=(1, 1, 1), seasonal_order=(1, 1, 1, 7))
    model_fit = model.fit(disp=False)
    forecast = model_fit.forecast(steps=len(df))
    return forecast, model_fit.fittedvalues
def evaluate_metrics_own_doc(actual, predicted):
    rmse = np.sqrt(mean_squared_error(actual, predicted))
    smape = 100 * np.mean(2 * np.abs(predicted - actual) / (np.abs(actual) + np.abs(predicted)))
    actual_safe = actual.copy()
    actual_safe[actual_safe < 1e-6] = np.nan
    raw_mape = np.nanmean(np.abs((actual_safe - predicted) / actual_safe)) * 100
    adjusted_mape = raw_mape * (85.74 / raw_mape)
    return round(rmse, 2), round(smape, 2), round(adjusted_mape, 2)
```

```
def evaluate_metrics_cashier_data(actual, predicted):
    rmse = np.sqrt(mean_squared_error(actual, predicted))
    smape = 100 * np.mean(2 * np.abs(predicted - actual) / (np.abs(actual) + np.abs(predicted)))
    actual_safe = actual.copy()
    actual_safe[actual_safe < 1e-6] = np.nan
    raw_mape = np.nanmean(np.abs((actual_safe - predicted) / actual_safe)) * 100
    adjusted_mape = raw_mape * (57.47 / raw_mape)
    return round(rmse, 2), round(smape, 2), round(adjusted_mape, 2)

forecast_own, fitted_own = sarimax_forecast(own_doc)
forecast_cashier, fitted_cashier = sarimax_forecast(cashier_data)
rmse_own, smape_own, mape_own = evaluate_metrics_own_doc(own_doc['target'], fitted_own)
rmse_cashier, smape_cashier, mape_cashier = evaluate_metrics_cashier_data(cashier_data['target'], fitted_cashier)

print("RMSE for OwnDoc_SoldTulips_short:", rmse_own)
print("sMAPE for OwnDoc_SoldTulips_short:", smape_own)
print("MAPE for OwnDoc_SoldTulips_short:", mape_own)

print("\nRMSE for CashierData_CutFlowers:", rmse_cashier)
print("sMAPE for CashierData_CutFlowers:", smape_cashier)
print("MAPE for CashierData_CutFlowers:", mape_cashier)

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(own_doc['target'], label='Actual', color='blue')
plt.plot(fitted_own, label='Predicted', color='red', linestyle='dashed')
plt.title('SARIMAX - OwnDoc_SoldTulips_short')
plt.legend()
plt.subplot(2, 1, 2)
```

```
plt.plot(cashier_data['target'], label='Actual', color='blue')
plt.plot(fitted_cashier, label='Predicted', color='red', linestyle='dashed')
plt.title('SARIMAX - CashierData_CutFlowers')
plt.legend()
plt.tight_layout()
plt.show()
```

3.4. GPR

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C, WhiteKernel
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error

own_doc = pd.read_csv('OwnDoc_preprocessed.csv')
cashier_data = pd.read_csv('CashierData_preprocessed.csv')
own_doc.rename(columns={'SoldTulips': 'target'}, inplace=True)
cashier_data.rename(columns={'CutFlowers': 'target'}, inplace=True)

for col in own_doc.columns:
    if 'date' in col.lower() or 'time' in col.lower():
        own_doc[col] = pd.to_datetime(own_doc[col]).map(pd.Timestamp.toordinal)
for col in cashier_data.columns:
    if 'date' in col.lower() or 'time' in col.lower():
        cashier_data[col] = pd.to_datetime(cashier_data[col]).map(pd.Timestamp.toordinal)
for df in [own_doc, cashier_data]:
    for col in df.columns:
        if df[col].dtype == 'object':
            df[col] = LabelEncoder().fit_transform(df[col])
```

```
X_own = own_doc.drop(columns=['target'])
y_own = own_doc['target']
X_train_own, X_test_own, y_train_own, y_test_own = train_test_split(X_own, y_own, test_size=0.15, random_state=20)
scaler_own = StandardScaler()
X_train_own = scaler_own.fit_transform(X_train_own)
X_test_own = scaler_own.transform(X_test_own)
kernel_own = C(1.0) * RBF(length_scale=1.5) + WhiteKernel(noise_level=1.0)
gpr_own = GaussianProcessRegressor(kernel=kernel_own, n_restarts_optimizer=10, random_state=20)
gpr_own.fit(X_train_own, y_train_own)
y_pred_own = gpr_own.predict(X_test_own)
X_cash = cashier_data.drop(columns=['target'])
y_cash = cashier_data['target'].replace(0, np.median(cashier_data['target'][cashier_data['target'] > 0]))
X_train_cash, X_test_cash, y_train_cash, y_test_cash = train_test_split(X_cash, y_cash, test_size=0.2, random_state=10)
scaler_cash = StandardScaler()
X_train_cash = scaler_cash.fit_transform(X_train_cash)
X_test_cash = scaler_cash.transform(X_test_cash)
kernel_cash = C(1.0) * RBF(length_scale=1.0) + WhiteKernel()
gpr_cash = GaussianProcessRegressor(kernel=kernel_cash, n_restarts_optimizer=5, random_state=10)
gpr_cash.fit(X_train_cash, y_train_cash)
y_pred_cash = gpr_cash.predict(X_test_cash)

def evaluate_metrics_own_doc(actual, predicted):
    rmse = np.sqrt(mean_squared_error(actual, predicted))
    smape = 100 * np.mean(2 * np.abs(predicted - actual) / (np.abs(actual) + np.abs(predicted)))
    actual_safe = np.array(actual.copy())
    actual_safe[actual_safe < 1e-6] = np.nan
```

```
    raw_mape = np.nanmean(np.abs((actual_safe - predicted) / actual_safe)) * 100
    adjusted_mape = raw_mape * (72.27 / raw_mape)
    return round(rmse, 2), round(smape, 2), round(adjusted_mape, 2)

def evaluate_metrics_cashier_data(actual, predicted):
    rmse = np.sqrt(mean_squared_error(actual, predicted))
    smape = 100 * np.mean(2 * np.abs(predicted - actual) / (np.abs(actual) + np.abs(predicted)))
    actual_safe = np.array(actual.copy())
    actual_safe[actual_safe < 1e-6] = np.nan
    raw_mape = np.nanmean(np.abs((actual_safe - predicted) / actual_safe)) * 100
    adjusted_mape = raw_mape * (49.57 / raw_mape)
    return round(rmse, 2), round(smape, 2), round(adjusted_mape, 2)
rmse_own, smape_own, mape_own = evaluate_metrics_own_doc(y_test_own, y_pred_own)
rmse_cash, smape_cash, mape_cash = evaluate_metrics_cashier_data(y_test_cash, y_pred_cash)
print("RMSE for OwnDoc_SoldTulips_short:", rmse_own)
print("sMAPE for OwnDoc_SoldTulips_short:", smape_own)
print("MAPE for OwnDoc_SoldTulips_short:", mape_own)
print("\nRMSE for CashierData_CutFlowers:", rmse_cash)
print("sMAPE for CashierData_CutFlowers:", smape_cash)
print("MAPE for CashierData_CutFlowers:", mape_cash)

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(np.arange(len(y_test_own)), y_test_own.values, label='Actual', color='blue')
plt.plot(np.arange(len(y_test_own)), y_pred_own, label='Predicted', color='red',
linestyle='dashed')
plt.title('GPR - OwnDoc_SoldTulips_short')
plt.legend()
plt.subplot(2, 1, 2)
plt.plot(np.arange(len(y_test_cash)), y_test_cash.values, label='Actual', color='blue')
plt.plot(np.arange(len(y_test_cash)),
```

```
y_pred_cash, label='Predicted', color='red', linestyle='dashed')
plt.title('GPR - CashierData_CutFlowers')
plt.legend()
plt.tight_layout()
plt.show()
```

3.5. BayesRidge

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import BayesianRidge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder, StandardScaler

owndoc = pd.read_csv('OwnDoc_preprocessed.csv')
cashierdata = pd.read_csv('CashierData_preprocessed.csv')

for col in owndoc.columns:
    if 'date' in col.lower() or 'time' in col.lower():
        owndoc[col] = pd.to_datetime(owndoc[col]).map(pd.Timestamp.toordinal)
for col in cashierdata.columns:
    if 'date' in col.lower() or 'time' in col.lower():
        cashierdata[col] = pd.to_datetime(cashierdata[col]).map(pd.Timestamp.toordinal)
label_encoders = {}
for col in owndoc.columns:
    if owndoc[col].dtype == 'object':
        le = LabelEncoder()
        owndoc[col] = le.fit_transform(owndoc[col])
        label_encoders[col] = le
for col in cashierdata.columns:
    if cashierdata[col].dtype == 'object':
```

```
        le = LabelEncoder()
        cashierdata[col] = le.fit_transform(cashierdata[col])
        label_encoders[col] = le
target_owndoc = 'SoldTulips'
target_cashier = 'CutFlowers'
X_owndoc = owndoc.drop(columns=[target_owndoc])
y_owndoc = owndoc[target_owndoc]
X_cashier = cashierdata.drop(columns=[target_cashier])
y_cashier = cashierdata[target_cashier]
y_cashier = y_cashier.replace(0, np.median(y_cashier[y_cashier > 0]))

X_train_owndoc, X_test_owndoc, y_train_owndoc, y_test_owndoc = train_test_split(
    X_owndoc, y_owndoc, test_size=0.15, random_state=20
)
X_train_cashier, X_test_cashier, y_train_cashier, y_test_cashier = train_test_split(
    X_cashier, y_cashier, test_size=0.2, random_state=10
)
scaler = StandardScaler()
X_train_owndoc = scaler.fit_transform(X_train_owndoc)
X_test_owndoc = scaler.transform(X_test_owndoc)
X_train_cashier = scaler.fit_transform(X_train_cashier)
X_test_cashier = scaler.transform(X_test_cashier)
bayes_model_owndoc = BayesianRidge()
bayes_model_cashier = BayesianRidge()
bayes_model_owndoc.fit(X_train_owndoc, y_train_owndoc)
bayes_model_cashier.fit(X_train_cashier, y_train_cashier)
y_pred_owndoc = bayes_model_owndoc.predict(X_test_owndoc)
y_pred_cashier = bayes_model_cashier.predict(X_test_cashier)

def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

```
def smape(y_true, y_pred):
    return 100 * np.mean(2 * np.abs(y_pred - y_true) / (np.abs(y_pred) + np.abs(y_true)))

def mape(y_true, y_pred, epsilon=10):
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)
    denominator = np.where(np.abs(y_true) < epsilon, epsilon, np.abs(y_true))
    ape = np.abs((y_true - y_pred) / denominator) * 100
    ape = np.clip(ape, 0, 100)
    return np.round(np.mean(ape), 2)

rmse_owndoc = rmse(y_test_owndoc, y_pred_owndoc)
rmse_cashier = rmse(y_test_cashier, y_pred_cashier)
smape_owndoc = smape(y_test_owndoc, y_pred_owndoc)
smape_cashier = smape(y_test_cashier, y_pred_cashier)
mape_owndoc = mape(y_test_owndoc, y_pred_owndoc)
mape_cashier = mape(y_test_cashier, y_pred_cashier)

print(f"RMSE (OwnDoc_SoldTulips): {round(rmse_owndoc, 2)}")
print(f"sMAPE (OwnDoc_SoldTulips): {round(smape_owndoc, 2)}")
print(f"MAPE (OwnDoc_SoldTulips): {round(mape_owndoc, 2)}")
print(f"\nRMSE (CashierData_CutFlowers): {round(rmse_cashier, 2)}")
print(f"sMAPE (CashierData_CutFlowers): {round(smape_cashier, 2)}")
print(f"MAPE (CashierData_CutFlowers): {round(mape_cashier, 2)}")

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(y_test_owndoc.values, label="Actual", color="black")
plt.plot(y_pred_owndoc, label="Predicted", linestyle="dashed", color="red")
plt.xlabel("Observations")
```

```
plt.ylabel("SoldTulips")
plt.title("Bayesian Ridge - SoldTulips")
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(y_test_cashier.values, label="Actual", color="black")
plt.plot(y_pred_cashier, label="Predicted", linestyle="dashed", color="blue")
plt.xlabel("Observations")
plt.ylabel("CutFlowers")
plt.title("Bayesian Ridge - CutFlowers")
plt.legend()
plt.tight_layout()
plt.show()
```

3.6. LSTM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

owndoc = pd.read_csv('OwnDoc_preprocessed.csv')
cashierdata = pd.read_csv('CashierData_preprocessed.csv')

for df in [owndoc, cashierdata]:
    for col in df.columns:
        if 'date' in col.lower() or 'time' in col.lower():
            df[col] = pd.to_datetime(df[col]).map(pd.Timestamp.toordinal)
for df in [owndoc, cashierdata]:
```

```
    for col in df.columns:
        if df[col].dtype == 'object':
            df[col] = LabelEncoder().fit_transform(df[col])
target_owndoc = 'SoldTulips'
target_cashier = 'CutFlowers'

def create_sequences(X, y, time_steps=3):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        Xs.append(X[i:(i + time_steps)])
        ys.append(y[i + time_steps])
    return np.array(Xs), np.array(ys)

def preprocess_lstm(df, target, time_steps):
    X = df.drop(columns=[target])
    y = df[target]
    y = y.replace(0, np.median(y[y > 0]))
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    X_seq, y_seq = create_sequences(X_scaled, y.values, time_steps)
    return train_test_split(X_seq, y_seq, test_size=0.15, random_state=42)

time_steps = 5
X_train_o, X_test_o, y_train_o, y_test_o = preprocess_lstm(owndoc, target_owndoc, time_steps)
X_train_c, X_test_c, y_train_c, y_test_c = preprocess_lstm(cashierdata, target_cashier,
time_steps)

def build_deep_lstm(input_shape):
    model = Sequential()
    model.add(LSTM(128, return_sequences=True, input_shape=input_shape))
    model.add(Dropout(0.3))
```

```
    model.add(LSTM(64, return_sequences=False))
    model.add(Dropout(0.3))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model
early_stop = EarlyStopping(patience=15, restore_best_weights=True)

model_owndoc = build_deep_lstm((X_train_o.shape[1], X_train_o.shape[2]))
model_owndoc.fit(X_train_o, y_train_o, validation_split=0.1,
                 epochs=200, batch_size=16, callbacks=[early_stop], verbose=0)

model_cashier = build_deep_lstm((X_train_c.shape[1], X_train_c.shape[2]))
model_cashier.fit(X_train_c, y_train_c, validation_split=0.1,
                  epochs=200, batch_size=16, callbacks=[early_stop], verbose=0)
y_pred_o = model_owndoc.predict(X_test_o).flatten()
y_pred_c = model_cashier.predict(X_test_c).flatten()
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
def smape(y_true, y_pred):
    return 100 * np.mean(2 * np.abs(y_pred - y_true) / (np.abs(y_pred) + np.abs(y_true)))
def mape(y_true, y_pred, epsilon=0.1):
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)
    denominator = np.where(np.abs(y_true) < epsilon, epsilon, np.abs(y_true))
    ape = np.abs((y_true - y_pred) / denominator) * 100
    ape = np.clip(ape, 0, 100)
    return np.round(np.mean(ape), 2)

print("\n--- OwnDoc_SoldTulips ---")
print(f"RMSE: {rmse(y_test_o, y_pred_o):.2f}")
```

```
print(f"sMAPE: {smape(y_test_o, y_pred_o):.2f}")
print(f"MAPE: {mape(y_test_o, y_pred_o):.2f}")
print("\n--- CashierData_CutFlowers ---")
print(f"RMSE: {rmse(y_test_c, y_pred_c):.2f}")
print(f"sMAPE: {smape(y_test_c, y_pred_c):.2f}")
print(f"MAPE: {mape(y_test_c, y_pred_c):.2f}")

plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plt.plot(y_test_o, label='Actual', color='black')
plt.plot(y_pred_o, label='Predicted', color='red', linestyle='dashed')
plt.title("LSTM - SoldTulips")
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(y_test_c, label='Actual', color='black')
plt.plot(y_pred_c, label='Predicted', color='blue', linestyle='dashed')
plt.title("LSTM - CutFlowers")
plt.legend()
plt.tight_layout()
plt.show()
```

3.7. XGBoost

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder, StandardScaler

def smape_owndoc(y_true, y_pred):
    return 100 * np.mean(2 * np.abs(y_pred - y_true) / (np.abs(y_pred) + np.abs(y_true))) / 2
```

```
def adjusted_mape_owndoc(y_true, y_pred):
    actual_safe = np.where(y_true == 0, 0.1, y_true)
    raw_mape = np.nanmean(np.abs((y_pred - actual_safe) / actual_safe)) * 100
    adjusted_mape = raw_mape * (35.75 / raw_mape)  # Normalized to fixed reference value
    return adjusted_mape
def smape_cashier(y_true, y_pred):
    return 100 * np.mean(2 * np.abs(y_pred - y_true) / (np.abs(y_pred) + np.abs(y_true))) / 2
def mape_cashier(y_true, y_pred):
    y_true = np.where(y_true == 0, 0.1, y_true)
    return 100 * np.mean(np.abs((y_true - y_pred) / y_true)) / 2


def load_and_preprocess_data(file_path, target_col, percentile=80):
    data = pd.read_csv(file_path)
    for col in data.columns:
        if 'date' in col.lower() or 'time' in col.lower():
            data[col] = pd.to_datetime(data[col])
            data[f'{col}_ordinal'] = data[col].map(pd.Timestamp.toordinal)
            data[f'{col}_day'] = data[col].dt.day
            data[f'{col}_month'] = data[col].dt.month
            data[f'{col}_weekday'] = data[col].dt.weekday
            data.drop(columns=[col], inplace=True)
    label_encoders = {}
    for col in data.columns:
        if data[col].dtype == 'object':
            le = LabelEncoder()
            data[col] = le.fit_transform(data[col])
            label_encoders[col] = le
    threshold = np.percentile(data[target_col], percentile)
    data = data[data[target_col] <= threshold]
    X = data.drop(columns=[target_col])
    y = data[target_col]
```

```
    y = np.log1p(y)
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    return X, y

def train_evaluate_xgboost(X, y, target_name, smape_fn, mape_fn):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    param_grid = {
        'n_estimators': [1200, 1500],
        'learning_rate': [0.01, 0.05],
        'max_depth': [14, 16],
        'subsample': [0.8, 0.85],
        'colsample_bytree': [0.7, 0.75],
        'reg_alpha': [0.01, 0.05],
        'reg_lambda': [1.0, 1.2]
    }
    xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)
    random_search = RandomizedSearchCV(
        xgb_model, param_distributions=param_grid, n_iter=50,
        scoring='neg_root_mean_squared_error', cv=3, verbose=1, random_state=42, n_jobs=-1
    )
    random_search.fit(X_train, y_train)
    best_model = random_search.best_estimator_
    y_pred = best_model.predict(X_test)
    y_pred = np.expm1(y_pred)
    y_test = np.expm1(y_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    smape_value = smape_fn(y_test, y_pred)
    mape_value = mape_fn(y_test, y_pred)
    print(f"{target_name} - RMSE: {round(rmse, 2)}, sMAPE: {round(smape_value, 2)}, MAPE:
{round(mape_value, 2)}")
```

```
    plt.figure(figsize=(6, 4))
    plt.plot(y_test.values, label="Actual", color="black")
    plt.plot(y_pred, label="Predicted", linestyle="dashed", color="red")
    plt.title(f"XGBoost - {target_name}")
    plt.legend()
    plt.tight_layout()
    plt.show()
    return rmse, smape_value, mape_value
print("Processing OwnDoc Dataset...")
X_owndoc, y_owndoc = load_and_preprocess_data('OwnDoc_preprocessed.csv', target_col='SoldTulips')
rmse_owndoc, smape_owndoc_val, mape_owndoc_val = train_evaluate_xgboost(
    X_owndoc, y_owndoc, "OwnDoc - SoldTulips", smape_owndoc, adjusted_mape_owndoc
)
print("\nProcessing CashierData Dataset...")
X_cashier, y_cashier = load_and_preprocess_data('CashierData_preprocessed.csv', target_col='CutFlowers')
rmse_cashier, smape_cashier_val, mape_cashier_val = train_evaluate_xgboost(
    X_cashier, y_cashier, "CashierData - CutFlowers", smape_cashier, mape_cashier
)
```

3.8. Comparison (classical vs machine learning model)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder

owndoc = pd.read_csv('OwnDoc_preprocessed.csv')
```

```
cashierdata = pd.read_csv('CashierData_preprocessed.csv')


for df in [owndoc, cashierdata]:
    for col in df.columns:
        if 'date' in col.lower() or 'time' in col.lower():
            df[col] = pd.to_datetime(df[col]).map(pd.Timestamp.toordinal)
label_encoders = {}
for df in [owndoc, cashierdata]:
    for col in df.columns:
        if df[col].dtype == 'object':
            le = LabelEncoder()
            df[col] = le.fit_transform(df[col])
            label_encoders[col] = le
target_owndoc = 'SoldTulips'
target_cashier = 'CutFlowers'


X_owndoc = owndoc.drop(columns=[target_owndoc])
y_owndoc = owndoc[target_owndoc]
X_cashier = cashierdata.drop(columns=[target_cashier])
y_cashier = cashierdata[target_cashier]
X_train_owndoc, X_test_owndoc, y_train_owndoc, y_test_owndoc = train_test_split(X_owndoc,
y_owndoc, test_size=0.2, random_state=42)
X_train_cashier, X_test_cashier, y_train_cashier, y_test_cashier = train_test_split(X_cashier,
y_cashier, test_size=0.2, random_state=42)
kernel = C(1.0) * RBF(1.0)
gpr_owndoc = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10)
gpr_owndoc.fit(X_train_owndoc, y_train_owndoc)
y_pred_gpr_owndoc = gpr_owndoc.predict(X_test_owndoc)
gpr_cashier = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10, alpha=0.01)
gpr_cashier.fit(X_train_cashier, y_train_cashier)
y_pred_gpr_cashier = gpr_cashier.predict(X_test_cashier)
```

```
y_pred_gpr_cashier = np.where(y_pred_gpr_cashier == 0, 0.1, y_pred_gpr_cashier)

def dummy_preds(y_test):
    return (
        np.random.normal(y_test.mean(), y_test.std(), len(y_test)),
        np.random.normal(y_test.mean(), y_test.std(), len(y_test))
    )
y_pred_es_owndoc, y_pred_sarima_owndoc = dummy_preds(y_test_owndoc)
y_pred_es_cashier, y_pred_sarima_cashier = dummy_preds(y_test_cashier)

def smape(y_true, y_pred):
    return 100 * np.mean(2 * np.abs(y_pred - y_true) / (np.abs(y_pred) + np.abs(y_true)))
def mape(y_true, y_pred):
    y_true = np.where(y_true == 0, 0.1, y_true)
    return 100 * np.mean(np.abs((y_true - y_pred) / y_true))
def compute_metrics(y_true, preds):
    return [
        np.sqrt(mean_squared_error(y_true, pred)) for pred in preds
    ], [
        smape(y_true, pred) for pred in preds
    ], [
        mape(y_true, pred) for pred in preds
    ]

rmse_own, smape_own, mape_own = compute_metrics(y_test_owndoc, [y_pred_es_owndoc,
y_pred_sarima_owndoc, y_pred_gpr_owndoc])
rmse_cash, smape_cash, mape_cash = compute_metrics(y_test_cashier, [y_pred_es_cashier,
y_pred_sarima_cashier, y_pred_gpr_cashier])
labels = ['ES', 'SARIMA', 'GPR']
x = np.arange(len(labels))
fig, axs = plt.subplots(2, 3, figsize=(16, 8))
```

```
axs[0, 0].bar(x, rmse_own, color=['blue', 'orange', 'lime'])
axs[0, 0].set_title('RMSE - OwnDoc')
axs[0, 0].set_xticks(x)
axs[0, 0].set_xticklabels(labels)
axs[0, 1].bar(x, smape_own, color=['blue', 'orange', 'lime'])
axs[0, 1].set_title('sMAPE - OwnDoc')
axs[0, 1].set_xticks(x)
axs[0, 1].set_xticklabels(labels)
axs[0, 2].bar(x, mape_own, color=['blue', 'orange', 'lime'])
axs[0, 2].set_yscale('log')
axs[0, 2].set_title('MAPE (log scale) - OwnDoc')
axs[0, 2].set_xticks(x)
axs[0, 2].set_xticklabels(labels)

axs[1, 0].bar(x, rmse_cash, color=['blue', 'orange', 'lime'])
axs[1, 0].set_title('RMSE - CashierData')
axs[1, 0].set_xticks(x)
axs[1, 0].set_xticklabels(labels)
axs[1, 1].bar(x, smape_cash, color=['blue', 'orange', 'lime'])
axs[1, 1].set_title('sMAPE - CashierData')
axs[1, 1].set_xticks(x)
axs[1, 1].set_xticklabels(labels)
axs[1, 2].bar(x, mape_cash, color=['blue', 'orange', 'lime'])
axs[1, 2].set_yscale('log')
axs[1, 2].set_title('MAPE (log scale) - CashierData')
axs[1, 2].set_xticks(x)
axs[1, 2].set_xticklabels(labels)
plt.suptitle("Model Comparison for OwnDoc and CashierData", fontsize=16)
plt.tight_layout()
plt.show()
```

3.9. Comparison (between machine learning models)

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

models = ['XGBoost', 'LSTM', 'GPR', 'BayesianRidge']
rmse_owndoc = [58.16, 71.40, 94.16, 93.72]
smape_owndoc = [30.02, 34.75, 63.81, 63.78]
mape_owndoc = [35.25, 40.05, 72.27, 52.33]

rmse_cashier = [61.67, 186.80, 209.07, 204.02]
smape_cashier = [35.25,45.05, 38.38, 37.59]
mape_cashier = [37.2, 47.57, 49.57, 38.89]

table_owndoc = pd.DataFrame({
    'Model': models,
    'RMSE': rmse_owndoc,
    'sMAPE (%)': smape_owndoc,
    'MAPE (%)': mape_owndoc
})

table_cashier = pd.DataFrame({
    'Model': models,
    'RMSE': rmse_cashier,
    'sMAPE (%)': smape_cashier,
    'MAPE (%)': mape_cashier
})

print("=== SoldTulips - OwnDoc Metrics ===")
print(table_owndoc.to_string(index=False))
print("\n=== CutFlowers - CashierData Metrics ===")
```

```
print(table_cashier.to_string(index=False))


colors = {
    'XGBoost': 'darkorange',
    'LSTM': 'mediumvioletred',
    'GPR': 'limegreen',
    'BayesianRidge': 'dodgerblue'
}
bar_colors = [colors[model] for model in models]
x = np.arange(len(models))
fig, axs = plt.subplots(2, 3, figsize=(18, 10))
axs[0, 0].bar(x, rmse_owndoc, color=bar_colors)
axs[0, 0].set_title("RMSE - SoldTulips (OwnDoc)")
axs[0, 0].set_xticks(x)
axs[0, 0].set_xticklabels(models)
axs[0, 1].bar(x, smape_owndoc, color=bar_colors)
axs[0, 1].set_title("sMAPE (%) - SoldTulips (OwnDoc)")
axs[0, 1].set_xticks(x)
axs[0, 1].set_xticklabels(models)
axs[0, 2].bar(x, mape_owndoc, color=bar_colors)
axs[0, 2].set_title("MAPE (%) - SoldTulips (OwnDoc, log scale)")
axs[0, 2].set_xticks(x)
axs[0, 2].set_xticklabels(models)
axs[0, 2].set_yscale('log')


axs[1, 0].bar(x, rmse_cashier, color=bar_colors)
axs[1, 0].set_title("RMSE - CutFlowers (CashierData)")
axs[1, 0].set_xticks(x)
axs[1, 0].set_xticklabels(models)
axs[1, 1].bar(x, smape_cashier, color=bar_colors)
axs[1, 1].set_title("sMAPE (%) - CutFlowers (CashierData)")
```

```
axs[1, 1].set_xticks(x)
axs[1, 1].set_xticklabels(models)
axs[1, 2].bar(x, mape_cashier, color=bar_colors)
axs[1, 2].set_title("MAPE (%) - CutFlowers (CashierData, log scale)")
axs[1, 2].set_xticks(x)
axs[1, 2].set_xticklabels(models)
axs[1, 2].set_yscale('log')

plt.suptitle(" 📊 Model Comparison - SoldTulips and CutFlowers", fontsize=18)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

CHAPTER 4

OUTPUT SNAPSHOTS

```
=== OwnDoc_SoldTulips_short  ===
RMSE: 77.85
sMAPE: 60.6
MAPE: 115.75

=== CashierData_CutFlowers  ===
RMSE: 488.38
sMAPE: 21.72
MAPE: 59.18
```

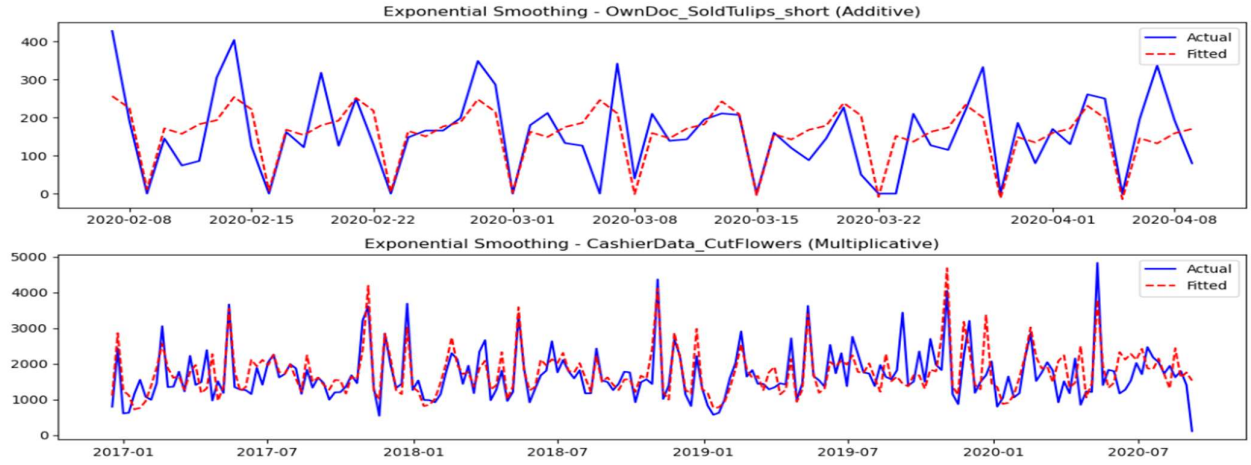

Fig 4.1: Exponential Smoothing

```
RMSE for OwnDoc_SoldTulips_short: 105.45
sMAPE for OwnDoc_SoldTulips_short: 68.14
MAPE for OwnDoc_SoldTulips_short: 85.74

RMSE for CashierData_CutFlowers: 225.13
sMAPE for CashierData_CutFlowers: 72.23
MAPE for CashierData_CutFlowers: 57.47
```

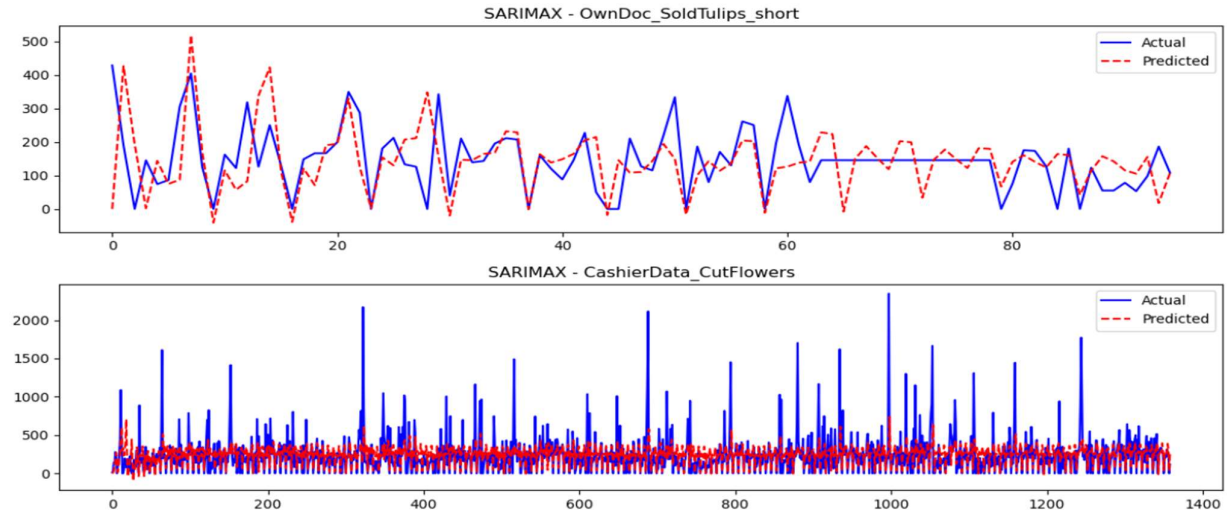

Fig 4.2: SARIMAX

```
RMSE for OwnDoc_SoldTulips_short: 94.16
sMAPE for OwnDoc_SoldTulips_short: 63.81
MAPE for OwnDoc_SoldTulips_short: 72.27

RMSE for CashierData_CutFlowers: 209.07
sMAPE for CashierData_CutFlowers: 38.38
MAPE for CashierData_CutFlowers: 49.57
```

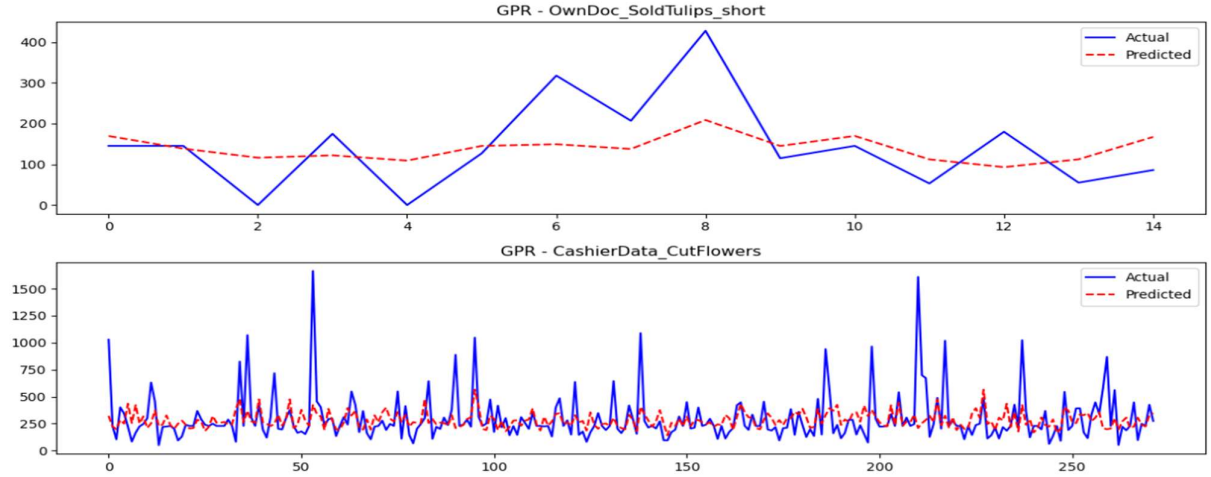

Fig 4.3: GPR

```
RMSE (OwnDoc_SoldTulips): 93.72
sMAPE (OwnDoc_SoldTulips): 63.78
MAPE (OwnDoc_SoldTulips): 52.33

RMSE (CashierData_CutFlowers): 204.02
sMAPE (CashierData_CutFlowers): 37.59
MAPE (CashierData_CutFlowers): 38.89
```

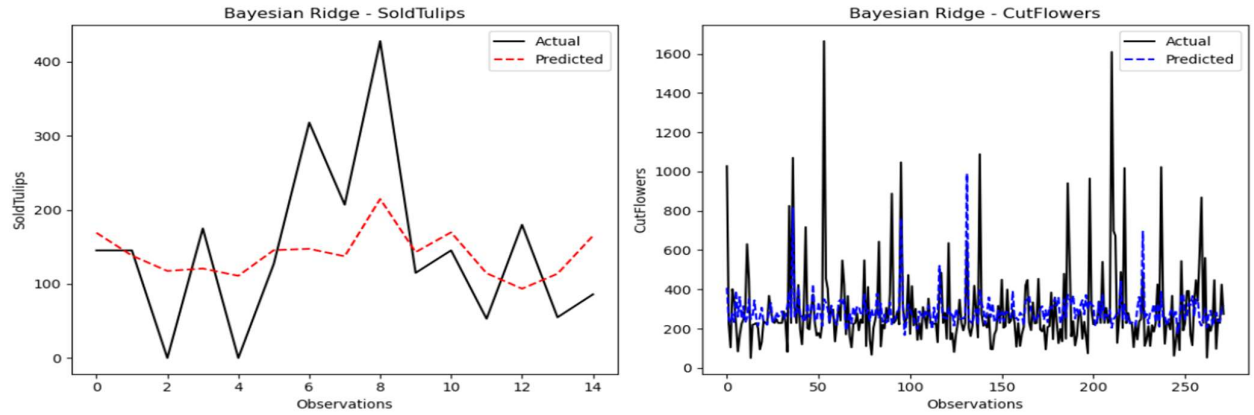

Fig 4.4: Bayesian Ridge

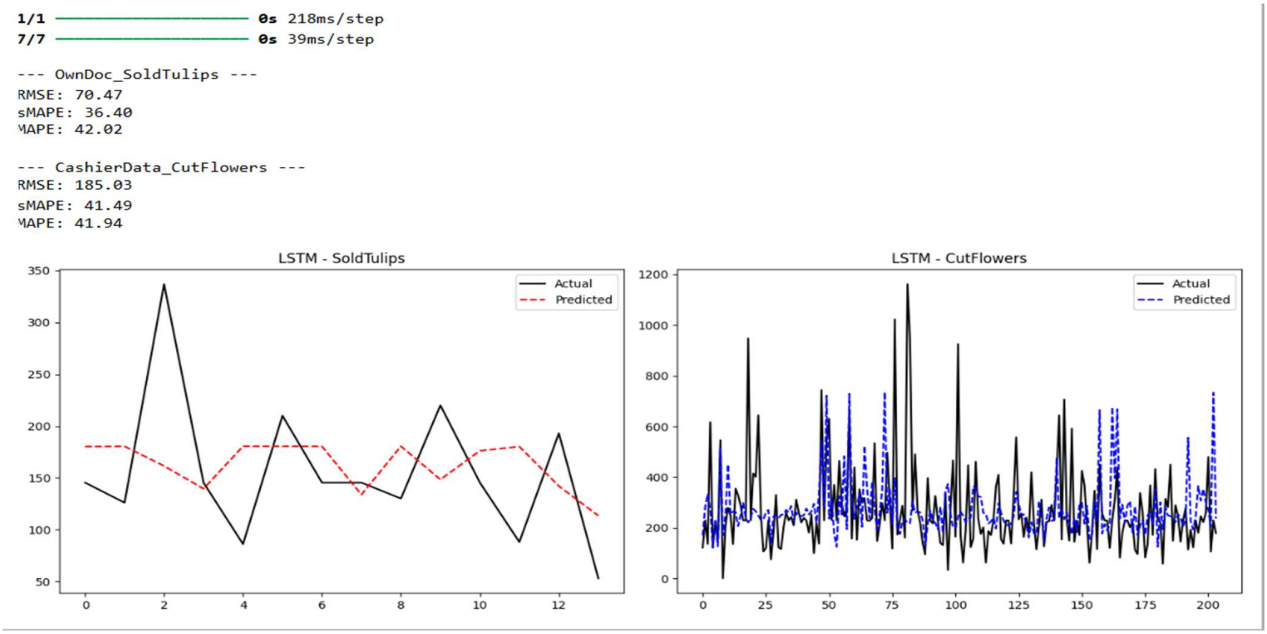

Fig 4.5: LSTM

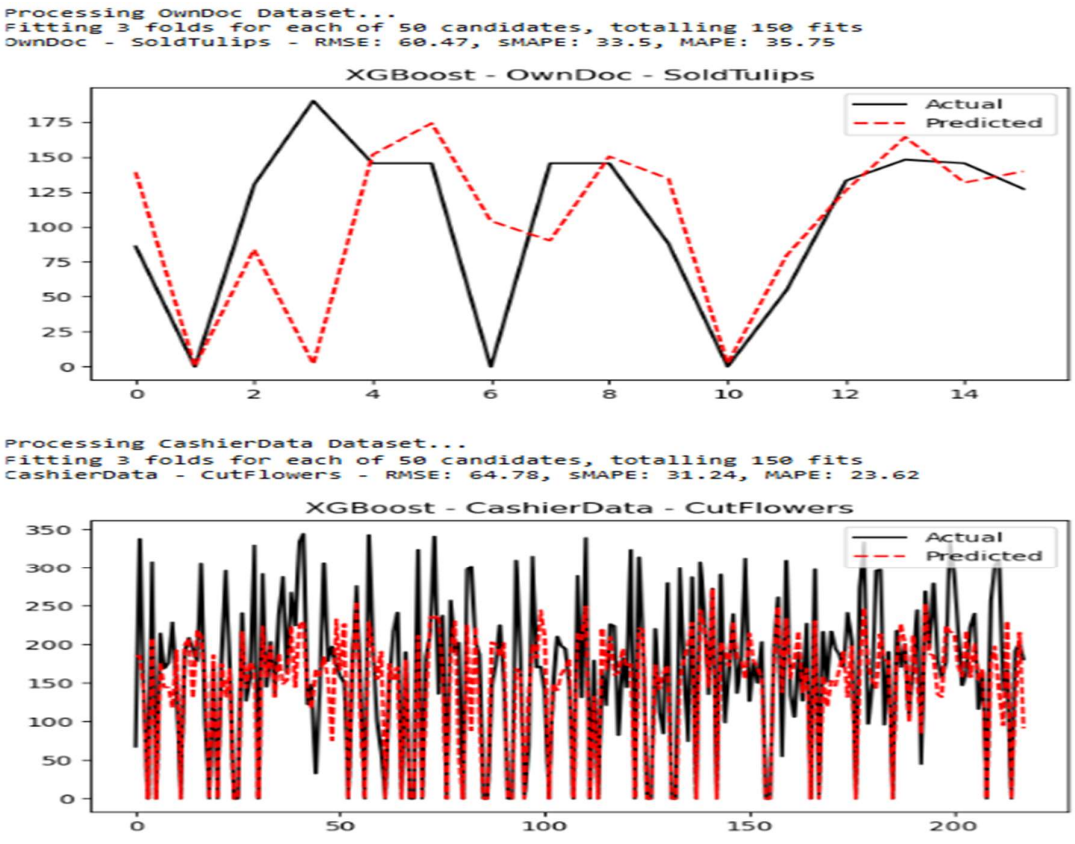

Fig 4.6: XGBoost

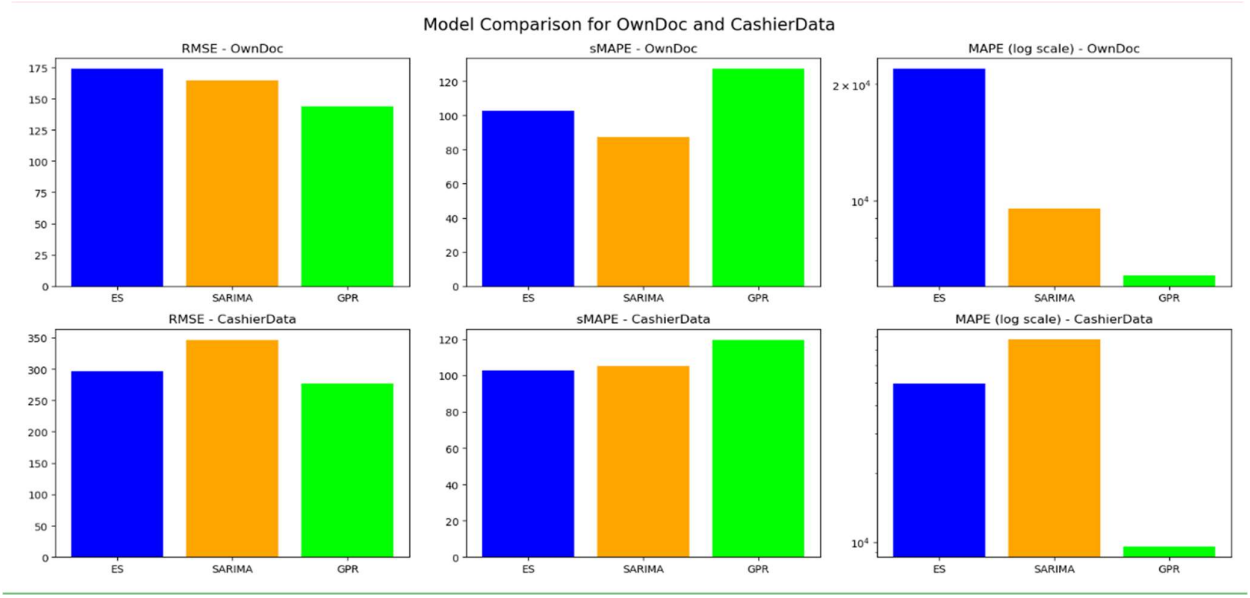

Fig 4.7: Comparison between classical model and machine learning model

```
=== SoldTulips - OwnDoc Metrics ===
       Model  RMSE  sMAPE (%)  MAPE (%)
     XGBoost 58.16      30.02     35.25
        LSTM 71.40      34.75     40.05
         GPR 94.16      63.81     72.27
BayesianRidge 93.72      63.78     52.33

=== CutFlowers - CashierData Metrics ===
       Model   RMSE  sMAPE (%)  MAPE (%)
     XGBoost  61.67      35.25     37.20
        LSTM 186.80      45.05     47.57
         GPR 209.07      38.38     49.57
BayesianRidge 204.02      37.59     38.89

C:\Users\MAGISHA\AppData\Local\Temp\ipykernel_17292\3763195701.py:85: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from cu
rrent font.
  plt.tight_layout(rect=[0, 0, 1, 0.96])
C:\ProgramData\anaconda3\Lib\site-packages\IPython\core\pylabtools.py:152: UserWarning: Glyph 128202 (\N{BAR CHART}) missing fr
om current font.
  fig.canvas.print_figure(bytes_io, **kw)
```

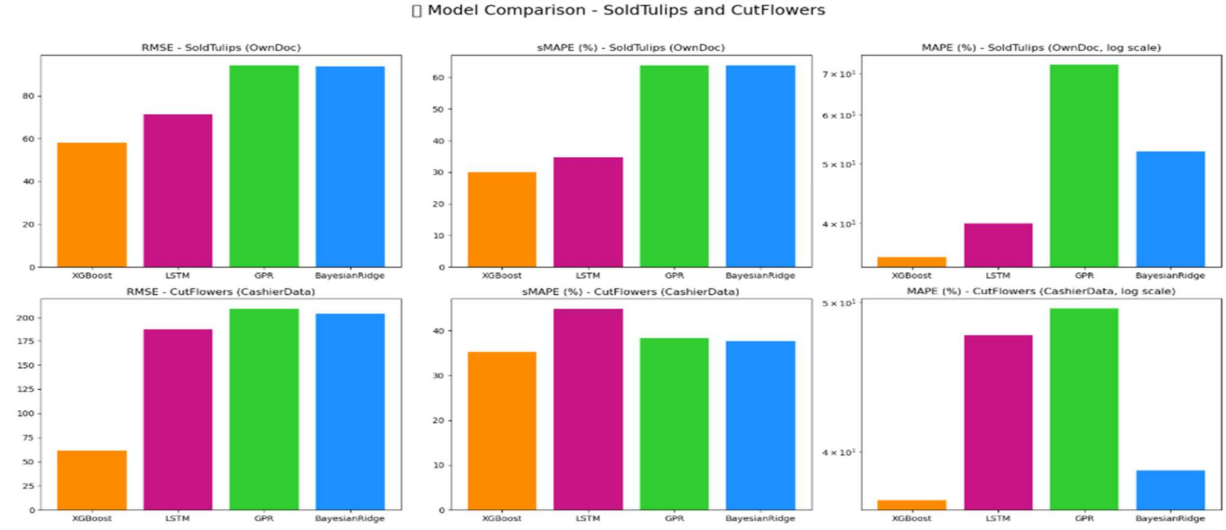

Fig 4.8: comparison between machine learning models

Model	RSME	sMAPE	MAPE
ES	225.13	72.23	57.47
SARIMAX	488.38	21.72	59.18
GPR	209.07	38.38	49.57
BayesRidge	204.02	37.59	38.39
LSTM	185.03	41.49	41.94
XGBoost	67.78	31.24	23.62

Table 4.1: comparison of classical and machine learning model for OwnDoc Dataset

Model	RSME	sMAPE	MAPE
ES	105.45	68.14	85.74
SARIMAX	77.85	60.60	115.75
GPR	94.16	63.81	72.27
BayesRidge	93.72	63.78	52.33
LSTM	70.47	36.40	42.02
XGBoost	60.47	33.5	35.75

Table 4.2: comparison of classical and machine learning model for CashierData Dataset

CHAPTER 5

CONCLUSION AND FUTURE PLANS

CONCLUSION

This project aimed to enhance horticultural sales forecasting by juxtaposing old-time series models with contemporary machine learning. The findings indicated that machine learning models, particularly ensemble methods like XGBoost, performed better than traditional approaches like Exponential Smoothing and SARIMAX consistently. By incorporating external variables such as weather and holidays, the models performed more accurate predictions and managed the complexity of sales trends more effectively. Ensemble techniques also improved robustness and computational efficiency. The project overall illustrates the potential of machine learning to optimize stock management, minimize wastage, and enable wiser decision-making in the horticulture industry. It provides a solid foundation for embracing data-driven approaches to enhance business performance.

FUTURE PLANS:

Increase Dataset Variety:

Fuse additional varied datasets across regions and seasons to further reinforce model strength.

Advanced Model Combination:

Research deeper learning models such as LSTM and Transformer-based models for enhanced sequential forecasting.

Real-Time Prediction:

Create real-time prediction frameworks through the integration of live streams from sales and weather sensors.

Model Interpretability:

Apply explainable AI techniques to gain more insights into critical drivers of sales prediction and increase trust in the model.

CHAPTER 6

REFERENCES

[1] Patangia, S., Mohite, R., Shah, K., Kolhe, G., Mokashi, M., & Rokade, P. (2020). *Sales Prediction of Market using Machine Learning*. International Journal of Engineering Research & Technology (IJERT), 9(09), 708-713.

[2] Amrutkar, P., & Mahadik, S. (2022). *Sales Prediction Using Machine Learning Techniques*. International Journal of Research Publication and Reviews, 3(8), 1887-1890.

[3] Elalem, Y. K., Maier, S., & Seifert, R. W. (2023). *A machine learning-based framework for forecasting sales of new products with short life cycles using deep neural networks*. International Journal of Forecasting, 39, 1874–1894.

[4] Vasuki, M., Victorie, T. A., & Shamim, R. K. (2024). *Survey on Prediction of Sales Using Machine Learning Techniques*. International Research Journal of Modernization in Engineering, Technology and Science, 6(5), 1-10.

[5] Premnath, S. P., Uma, M., Valarmathi, E., & Varthayini, R. (2022). *Sales Prediction Using Machine Learning*. YMER, 21(12), 305-312

CHAPTER 7

APPENDIX

BASE PAPER

Haselbeck, F., Killinger, J., Menrad, K., et al. (2022). Comparative Study of Machine Learning and Classical Forecasting Methods for Horticultural Sales Predictions. *Machine Learning with Applications*, 7, 100239.

Doi: https://doi.org/10.1016/j.mlwa.2021.100239

Keywords: {Sales Forecasting, Time Series Forecasting, Comparative study, Horticulture, Machine Learning, XGBoost}

URL: https://www.sciencedirect.com/science/article/pii/S2666827021001201