

FORECASTING MALAYSIAN
RICE PRODUCTION USING
HISTORICAL CLIMATE DATA AND
MACHINE LEARNING ALGORITHMS

NURHAFIZAH BINTI MOHD YUNOS

UNIVERSITI TEKNOLOGI MALAYSIA

TABLE OF CONTENTS

| TITLE | PAGE |
|---|----------|
| CHAPTER 4 INITIAL FINDINGS..... | 4 |
| 4.1 Introduction..... | 4 |
| 4.2 Data Loading and Preparation..... | 5 |
| 4.2.1 Dataset Sources and Structure..... | 5 |
| 4.3 Paddy Data Processing..... | 6 |
| 4.3.1 Filtering for Paddy Crop..... | 9 |
| 4.3.2 Temporal Disaggregation..... | 10 |
| 4.3.3 Handling Zero Production States..... | 11 |
| 4.4 Weather Data Processing..... | 12 |
| 4.4.1 File Reading and Header Detection..... | 13 |
| 4.4.2 Format Transformation..... | 14 |
| 4.4.3 State Identification and Concatenation..... | 15 |
| 4.5 Data Merging..... | 18 |
| 4.5.1 Joining Datasets..... | 19 |
| 4.6 Exploratory Data Analysis (EDA) Results..... | 20 |
| 4.6.1 Descriptive Statistics..... | 20 |
| 4.6.2 Data Types and Missing Values..... | 21 |
| 4.6.3 Duplicate Rows Check..... | 22 |
| 4.6.4 Distribution Plots..... | 22 |
| 4.6.5 Time Series Trends..... | 24 |
| 4.6.6 State-wise Variability..... | 26 |
| 4.6.7 Correlation Analysis..... | 28 |
| 4.6.8 Yield Calculation..... | 30 |
| 4.7 Key Observations from EDA..... | 31 |
| 4.8 Initial Forecasting Models: Model Development and Evaluation..... | 35 |
| 4.8.1 Feature Engineering..... | 36 |
| 4.8.2 Train-Test Split..... | 37 |
| 4.8.3 Random Forest Regressor..... | 39 |
| 4.8.4 Support Vector Regression (SVR)..... | 41 |
| 4.8.5 Long Short-Term Memory (LSTM) Network..... | 43 |
| 4.9 Comparative Performance Summary..... | 46 |
| 4.10 Limitations of Initial Modelling..... | 50 |

| | |
|--|----|
| 4.11 Implications of the Findings..... | 51 |
| 4.12 Conclusion..... | 53 |

CHAPTER 4

INITIAL FINDINGS

4.1 Introduction

Accurate forecasting of paddy (rice) production is essential for ensuring food security, supporting agricultural planning, and informing policy decisions in Malaysia. As one of the key staple crops, paddy production is influenced by a variety of factors including climate conditions, land use, agricultural practices, and government policies. This chapter presents the initial findings from the data collection, preprocessing, exploratory data analysis, and preliminary modeling efforts aimed at developing a reliable forecasting framework. The chapter begins with an overview of the datasets used, followed by detailed descriptions of the data cleaning and processing steps. Exploratory data analysis provides insights into historical trends, seasonal patterns, and relationships between production and climatic variables. Finally, early forecasting models—namely Random Forest, Support Vector Regression (SVR), and Long Short-Term Memory (LSTM)—are introduced, along with their performance evaluation. These findings serve as the foundation for further model refinement and more in-depth analysis presented in the subsequent chapters.

4.2 Data Loading and Preparation

This section outlines the initial steps taken to load, clean, and prepare the datasets for analysis. Two main datasets were used: one containing historical crop yield data and another with state-specific monthly weather information. Key preprocessing tasks included renaming and standardising columns (e.g., 'NAME' to 'state'), converting date fields to datetime format, and checking for missing values and duplicates. These steps ensured the data was structured consistently and ready for further processing and exploratory analysis.

4.2.1 Dataset Sources and Structure

The forecasting of paddy production in Malaysia was based on two primary datasets that provide essential agricultural and climatic information:

a) Crop Yield Dataset (crop_yield.csv)

This dataset contains historical records of crop production across various states in Malaysia. It includes data at both annual and seasonal levels, covering multiple crops, with a focus on key agricultural indicators such as production quantity (in tonnes), planted area (in hectares), and yield (tonnes per hectare). For this study, only the records related to "paddy" were extracted for further analysis.

```
[1] from google.colab import drive
    drive.mount('/content/drive')

import pandas as pd

prod_df = pd.read_csv('/content/drive/My Drive/RDA/crop_yield.csv')
```

Mounted at /content/drive

Figure 4.1 Figure of Data Loading of crop_yield.csv

b) Weather Data Files

These consist of multiple CSV files, each corresponding to a specific Malaysian state. The files contain detailed monthly weather parameters obtained from NASA's MERRA-2 or similar climate datasets. Key variables include total precipitation (PRECTOTCORR_SUM), minimum, and maximum temperatures (T2M_MIN, T2M_MAX), and solar radiation. These variables are critical for understanding how climatic conditions influence paddy production over time.

These two datasets formed the foundation for building a comprehensive model to forecast future paddy production by integrating both agricultural and environmental factors.

4.3 Paddy Data Processing

This section focuses on preparing the crop yield data specifically for paddy (rice) production. Since the original dataset included multiple crops, it was filtered to include only

paddy-related records. To align with monthly weather data, annual and seasonal paddy production data was disaggregated into monthly values using a predefined distribution pattern, resulting in the `monthly_paddy_df`. States with zero total production were assigned zero values for each month to maintain consistency. This step ensured the data was temporally aligned and ready for integration with climate variables.

The following steps were carried out on the crop yield dataset:

a) Renaming and Standardizing the 'NAME' Column:

The column originally labeled 'NAME' in the crop yield dataset contained the names of Malaysian states. To improve clarity and ensure consistency with other datasets (especially when merging with weather data later), this column was renamed to 'state'. Additionally, all state names were converted to lowercase. This standardization helped prevent mismatches due to case sensitivity (e.g., "Selangor" vs. "selangor") during data integration.

```
[2] # Rename the 'NAME' column into 'state'
    prod_df.rename(columns={'NAME': 'state'}, inplace=True)
    # Standardize the rows of state into small letters
    prod_df['state'] = prod_df['state'].str.lower()
```

Figure 4.2 Figure of Renaming and Standardizing of `crop_yield.csv`

b) Conversion of Date Columns to Datetime Format:

Any column containing date information (such as the planting or harvesting date) was converted from string format into Python's datetime object. This transformation is crucial for time-series analysis, as it enables sorting by date, extracting time-based features (like month or year), and aligning data across different time intervals. It also facilitates plotting and analysis over time.

```
[3] # Change datatype of date to 'date'  
prod_df['date'] = pd.to_datetime(prod_df['date'])
```

Figure 4.3 Figure of Conversion of Date Columns of crop_yield.csv

c) Checking for Missing Values and Duplicates:

An initial inspection of the raw crop yield data revealed that there were no missing values or duplicate rows present in the dataset at the start of the preprocessing stage. This is important because missing or duplicate data can introduce bias or errors in analysis and modeling. While the crop data itself was clean at this point, as will be discussed later, missing values did appear after merging with the weather data, requiring further handling.


```
# Checking for missing values
print("Missing values per column:")
print(prod_df.isnull().sum())

# Checking for duplicate rows
print("\nNumber of duplicate rows:")
print(prod_df.duplicated().sum())
```

Missing values per column:

| | |
|--------------|-------|
| state | 0 |
| date | 0 |
| crop_type | 0 |
| planted_area | 0 |
| production | 0 |
| dtype: | int64 |

Number of duplicate rows:

0

Figure 4.4 Figure of Checking for Missing and Duplicates of crop_yield.csv

4.3.1 Filtering for Paddy Crop

To ensure the analysis was focused specifically on rice production in Malaysia, the dataset was filtered to include only records related to "paddy" , which is the term commonly used for rice in its unprocessed form (before milling). The original crop yield dataset contained data for multiple crops (e.g., corn, sugarcane), so isolating paddy data allowed for more accurate and relevant modeling. This step ensured that all subsequent analyses, including feature engineering and forecasting, were tailored to the target crop.

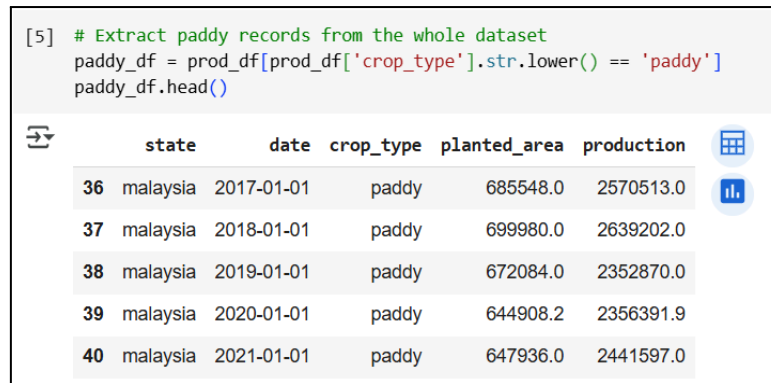


Figure 4.5 Figure of Filtering for Paddy Crop

4.3.2 Temporal Disaggregation

The original paddy production data was recorded on an annual or seasonal basis , meaning it provided total production values per year or per planting season. However, since weather data was available at a monthly resolution , it was necessary to convert the annual/seasonal data into monthly intervals to allow proper alignment and integration.

To achieve this, a predefined monthly distribution pattern was applied. This pattern distributed the annual or seasonal production across months based on typical planting, growing, and harvesting cycles. As a result, the new `monthly_paddy_df` dataset was created, containing estimated monthly production values. This granular time-based format enabled more precise modeling and correlation with monthly weather variables.

```

▶ from datetime import datetime
import numpy as np

# Define monthly distribution pattern
monthly_pattern = np.array([
    0.05, 0.05, 0.06, 0.07, 0.08, 0.10,
    0.12, 0.13, 0.12, 0.09, 0.08, 0.05
])

# List to store expanded rows
expanded_data = []

# Iterate over each row
for _, row in paddy_df.iterrows():
    state = row['state']
    # Access the year directly from the datetime object
    year = row['date'].year
    planted_area = row['planted_area']
    total_production = row['production']

    # Skip if no production (like Kuala Lumpur)
    if total_production == 0:
        monthly_production = [0] * 12
    else:
        monthly_production = np.round(total_production * monthly_pattern, 2)

    # Generate 12 rows for each month
    for month in range(1, 13):
        # Format the date correctly
        date = f"{year}-{str(month).zfill(2)}-01"
        expanded_data.append({
            'state': state,
            'date': date,
            'crop_type': 'paddy',
            'planted_area': planted_area,
            'production': monthly_production[month - 1]
        })

# Create new DataFrame using pd.DataFrame
monthly_paddy_df = pd.DataFrame(expanded_data)

print("Monthly dummy dataset created successfully!")

```

↩ Monthly dummy dataset created successfully!

Figure 4.6 Figure of Creating Monthly Dataset

4.3.3 Handling Zero Production States

Some Malaysian states had zero total paddy production during certain years or seasons. While this may reflect actual agricultural conditions (e.g., no paddy cultivation in a particular area), omitting these entries could lead to data imbalance or bias in the modeling process.

To maintain completeness and ensure accurate representation, these zero-production states were retained in the dataset. Specifically, each month within a zero-production year or season was assigned a value of zero for paddy production. This approach preserved the integrity of the dataset and allowed models to learn from complete spatial and temporal patterns without assuming production where there was none.

4.4 Weather Data Processing

This section describes how weather data from multiple sources was cleaned and transformed for use in the forecasting model. The data came in separate CSV files for each Malaysian state, with complex formats and inconsistent headers. A custom function was used to locate the correct header row. The data was then reshaped from wide to long format and restructured to have weather variables as columns. A 'state' column was added based on the file name, and all state datasets were combined into one unified DataFrame (`weather_df_all`). This processed dataset was then ready to be merged with the paddy production data.

4.4.1 File Reading and Header Detection

The weather data for each Malaysian state was stored in separate CSV files. However, not all files started directly with the header row containing column names like 'YEAR', 'MO', or weather parameters. Some files included additional metadata lines at the beginning—such as descriptions of the data source or units—which could interfere with proper data loading.

To handle this inconsistency, a custom function called `find_header_row()` was created. This function scanned through each file to automatically detect the first line that contained actual column headers, skipping any initial metadata lines. Using this function ensured that the correct data structure was read from each file, regardless of how many metadata lines it contained.

```
[7] import glob
import pandas as pd

def find_header_row(file_path):
    """Find the line number where 'PARAMETER, YEAR' appears"""
    with open(file_path, 'r') as f:
        for idx, line in enumerate(f):
            if line.startswith('PARAMETER, YEAR'):
                return idx
    raise ValueError(f"Header not found in {file_path}")
```

Figure 4.7 Figure of File Reading and Header Detection

4.4.2 Format Transformation

Once the correct headers were identified and the data was loaded, the next step was to restructure the format of the weather data.

Originally, the data was in wide format , where each month (e.g., Jan, Feb) was represented as a separate column. To make the data more suitable for time-series analysis and easier to merge with the paddy production dataset, it was transformed into long format using the `melt()` function. This process converted the monthly columns into rows, with each row representing a specific month and its corresponding weather value.

After reshaping, the data was then pivoted again so that each weather parameter (e.g., 'PRECTOTCORR_SUM' for precipitation, 'T2M_MAX' for maximum temperature) became a separate column, and the 'date' field was set as the index. This final structure made the dataset clean, consistent, and ready for further analysis.

```

# Melt the DataFrame
df_melted = df.melt(
    id_vars=["PARAMETER", "YEAR"],
    value_vars=["JAN", "FEB", "MAR", "APR", "MAY", "JUN",
                "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"],
    var_name="month", value_name="value"
)

# Map month names to numbers
month_map = {
    "JAN": 1, "FEB": 2, "MAR": 3, "APR": 4, "MAY": 5, "JUN": 6,
    "JUL": 7, "AUG": 8, "SEP": 9, "OCT": 10, "NOV": 11, "DEC": 12
}
df_melted['month_num'] = df_melted['month'].map(month_map)

# Create date column
df_melted['date'] = pd.to_datetime(
    dict(year=df_melted['YEAR'], month=df_melted['month_num'], day=1)
)

# Pivot the melted DataFrame
df_pivot = df_melted.pivot_table(
    index="date", columns="PARAMETER", values="value"
).reset_index()
df_pivot.columns.name = None # Remove pivot table column name

```

Figure 4.8 Figure of Format Transformation

4.4.3 State Identification and Concatenation

Since each CSV file corresponded to a different Malaysian state, it was important to identify which state the data belonged to. A new column called 'state' was added to each DataFrame during processing, derived directly from the filename (e.g., Johor.csv, Selangor.csv).

Once each state's weather data was cleaned and structured consistently, all individual DataFrames were combined into a single unified DataFrame named `weather_df_all`. This consolidated dataset contained weather information for all states, with standardized formatting and a 'state' identifier, making it ready for merging with the paddy production data.

```
# Add state information
df_pivot['state'] = state
```

Figure 4.9 Figure of State Identification

```
# Final concatenation
if weather_all:
    weather_df_all = pd.concat(weather_all, ignore_index=True)
    print("\n✅ Successfully concatenated all files.")
else:
    print("\n⚠️ No valid files were processed. Output DataFrame is empty.")
```



```
✅ Successfully concatenated all files.
```

Figure 4.10 Figure of Concatenation


```

import glob
import pandas as pd

def find_header_row(file_path):
    """Find the line number where 'PARAMETER, YEAR' appears"""
    with open(file_path, 'r') as f:
        for idx, line in enumerate(f):
            if line.startswith('PARAMETER, YEAR'):
                return idx
    raise ValueError(f"Header not found in {file_path}")

# List all weather CSV files from Google Drive
weather_files = glob.glob("/content/drive/My Drive/RDA/weather/weather_*.csv")
weather_all = []

for file in weather_files:
    try:
        # Find where the real header starts
        header_row = find_header_row(file)

        # Read the CSV file, skipping initial lines
        df = pd.read_csv(file, on_bad_lines='skip', header=header_row)

        # Extract state from filename
        state = file.split('_')[1].replace(".csv", "")

        # Melt the DataFrame
        df_melted = df.melt(
            id_vars=["PARAMETER", "YEAR"],
            value_vars=["JAN", "FEB", "MAR", "APR", "MAY", "JUN",
                       "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"],
            var_name="month", value_name="value"
        )

        # Map month names to numbers
        month_map = {
            "JAN": 1, "FEB": 2, "MAR": 3, "APR": 4, "MAY": 5, "JUN": 6,
            "JUL": 7, "AUG": 8, "SEP": 9, "OCT": 10, "NOV": 11, "DEC": 12
        }
        df_melted['month_num'] = df_melted['month'].map(month_map)

        # Create date column
        df_melted['date'] = pd.to_datetime(
            dict(year=df_melted['YEAR'], month=df_melted['month_num'], day=1)
        )

        # Pivot the melted DataFrame
        df_pivot = df_melted.pivot_table(
            index="date", columns="PARAMETER", values="value"
        ).reset_index()
        df_pivot.columns.name = None # Remove pivot table column name

        # Add state information
        df_pivot['state'] = state

        # Append to list
        weather_all.append(df_pivot)

    except Exception as e:
        print(f"❌ Error processing file: {file}")
        print(f"Error details: {e}")

# Final concatenation
if weather_all:
    weather_df_all = pd.concat(weather_all, ignore_index=True)
    print("\n✅ Successfully concatenated all files.")
else:
    print("\n⚠️ No valid files were processed. Output DataFrame is empty.")

```

✅ Successfully concatenated all files.

Figure 4.11 Figure of Full Data Uploading and Preparation (Weather)

4.5 Data Merging

This section describes the process of combining the processed paddy production data (`monthly_paddy_df`) with the weather data (`weather_df_all`) to create a unified dataset for analysis and modeling.

The datasets were merged based on two key fields: 'state' and 'date', ensuring that each monthly paddy record was matched with the corresponding weather conditions for that state and time period. An inner join was used, meaning only records with matching values in both datasets were included. The result was a final merged dataset called `merged_df`, which contains all the necessary variables—production, area planted, yield, and weather parameters—at a monthly and state-level resolution.

This merged dataset became the foundation for exploratory data analysis and forecasting model development.

```
[9] # Ensure the 'date' column in monthly_paddy_df is datetime type
monthly_paddy_df['date'] = pd.to_datetime(monthly_paddy_df['date'])

merged_df = pd.merge(monthly_paddy_df, weather_df_all, on=["state", "date"], how="inner")
```

Figure 4.12 Figure of Data Merging

4.5.1 Joining Datasets

To create a complete dataset for forecasting paddy production, the two main processed datasets—`monthly_paddy_df` (containing monthly paddy production data) and `weather_df_all` (containing monthly weather data for each state)—were combined through a merging process .

The datasets were joined using the common fields:

- a) 'state': to ensure data corresponds to the same region.
- b) 'date': to align data by specific month and year.

An inner join was used during the merge. This means that only the rows where both paddy production data and weather data were available for the same state and date were included in the final merged dataset, which was named `merged_df`.

As a result, `merged_df` contains comprehensive monthly records with both agricultural and climatic variables, making it suitable for exploratory analysis and modeling.


4.6 Exploratory Data Analysis (EDA) Results

Exploratory Data Analysis (EDA) was conducted to understand the characteristics of the merged dataset (merged_df) and to identify patterns, trends, and relationships between paddy production and influencing factors such as weather conditions and land use.

4.6.1 Descriptive Statistics

Descriptive statistics were generated for key numerical variables including 'production', 'planted_area', 'PRECTOTCORR_SUM' (total monthly precipitation), and 'T2M_MAX' (maximum temperature). These summaries provided insights into measures of central tendency (mean, median) and dispersion (standard deviation, range), helping to characterize the general behavior of each variable and detect anomalies or extreme values.

```
[12] # Display descriptive statistics
print("\nDescriptive Statistics:")
print(merged_df.describe())
```



Descriptive Statistics:

| | date | planted_area | production | \ |
|-------|-------------------------------|---------------|---------------|---|
| count | 792 | 792.000000 | 792.000000 | |
| mean | 2019-12-16 11:19:59.999999744 | 58006.290909 | 17435.701869 | |
| min | 2017-01-01 00:00:00 | 2505.000000 | 375.100000 | |
| 25% | 2018-06-23 12:00:00 | 13639.000000 | 3569.100000 | |
| 50% | 2019-12-16 12:00:00 | 41301.500000 | 11621.850000 | |
| 75% | 2021-06-08 12:00:00 | 74972.000000 | 20997.172500 | |
| max | 2022-12-01 00:00:00 | 214880.000000 | 124236.060000 | |
| std | NaN | 58854.642639 | 21931.020390 | |

| | ALLSKY_SFC_LW_DWN | PRECTOTCORR_SUM | RH2M | T2M_MAX | T2M_MIN |
|-------|-------------------|-----------------|------------|------------|------------|
| count | 792.000000 | 792.000000 | 792.000000 | 792.000000 | 792.000000 |
| mean | 36.074015 | 245.210290 | 84.140215 | 31.071894 | 23.565821 |
| min | 33.560000 | 0.500000 | 68.580000 | 27.580000 | 18.770000 |
| 25% | 35.630000 | 153.252500 | 81.145000 | 30.270000 | 22.270000 |
| 50% | 36.165000 | 224.790000 | 83.815000 | 30.860000 | 23.675000 |
| 75% | 36.570000 | 314.562500 | 87.102500 | 31.750000 | 25.010000 |
| max | 37.350000 | 1292.260000 | 93.620000 | 36.070000 | 27.090000 |
| std | 0.599558 | 140.920517 | 4.156150 | 1.252668 | 1.731341 |

Figure 4.13 Figure of Descriptive Statistics

4.6.2 Data Types and Missing Values

The `.info()` method confirmed that all columns in the merged dataset had appropriate data types after merging, ensuring consistency for further analysis. However, using `.isnull().sum()`, it was found that several weather-related columns contained missing values. This indicated that some states or time periods lacked complete weather records, which could impact model accuracy if not addressed.

```
[13] # Check data types and non-null values
print("\nDataFrame Info:")
merged_df.info()

# Check for missing values
print("\nMissing values per column:")
print(merged_df.isnull().sum())
```

↔

DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 792 entries, 0 to 791
Data columns (total 10 columns):

| # | Column | Non-Null Count | Dtype |
|---|-------------------|----------------|----------------|
| 0 | state | 792 non-null | object |
| 1 | date | 792 non-null | datetime64[ns] |
| 2 | crop_type | 792 non-null | object |
| 3 | planted_area | 792 non-null | float64 |
| 4 | production | 792 non-null | float64 |
| 5 | ALLSKY_SFC_LW_DWN | 792 non-null | float64 |
| 6 | PRECTOTCORR_SUM | 792 non-null | float64 |
| 7 | RH2M | 792 non-null | float64 |
| 8 | T2M_MAX | 792 non-null | float64 |
| 9 | T2M_MIN | 792 non-null | float64 |

dtypes: datetime64[ns](1), float64(7), object(2)
memory usage: 62.0+ KB

Missing values per column:

| | |
|-------------------|---|
| state | 0 |
| date | 0 |
| crop_type | 0 |
| planted_area | 0 |
| production | 0 |
| ALLSKY_SFC_LW_DWN | 0 |
| PRECTOTCORR_SUM | 0 |
| RH2M | 0 |
| T2M_MAX | 0 |
| T2M_MIN | 0 |


dtype: int64

Figure 4.14 Figure of Data Types and Missing Values

4.6.3 Duplicate Rows Check

A check for duplicate rows in the merged dataset returned no duplicates, confirming the integrity of the data and reducing concerns about biased results due to repeated entries.

```
[14] # Check for duplicate rows
      print("\nNumber of duplicate rows:")
      print(merged_df.duplicated().sum())
```



Number of duplicate rows:
0

Figure 4.15 Figure of Duplicate Rows Check

4.6.4 Distribution Plots

Histograms were used to visualize the distributions of important variables such as 'production', 'planted_area', and selected weather parameters. These plots revealed that some variables were skewed or contained outliers, suggesting the need for normalization or transformation before modeling.

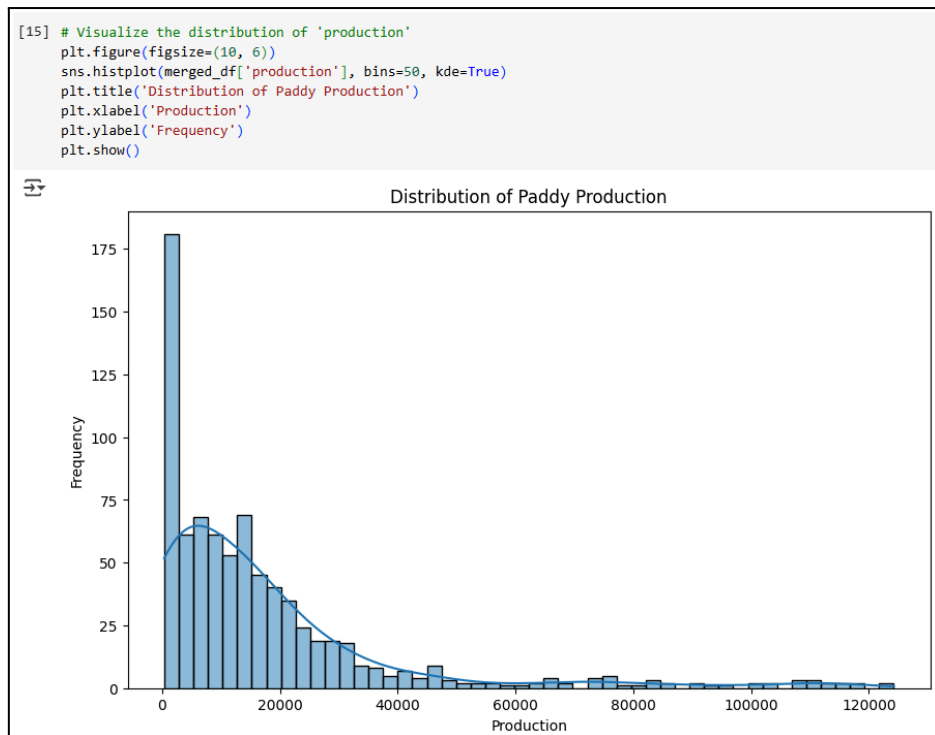


Figure 4.16 Figure of Distribution of Paddy Production Histogram

The histogram shows the distribution of paddy production values, with a right-skewed pattern . Most production values are concentrated at lower levels (around 0–5,000 units), while higher production values occur less frequently. The KDE curve confirms this skewness, indicating that low production is common, and high production is rare. This suggests that most observations have relatively low production, possibly due to constraints such as resource limitations or unfavorable conditions.

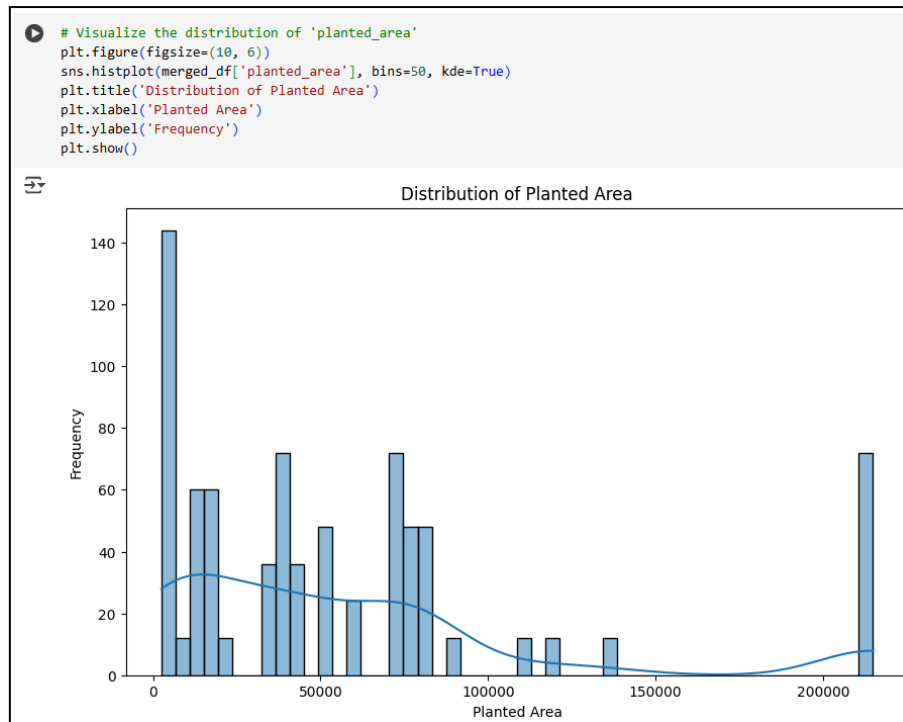


Figure 4.17 Figure of Distribution of Planted Area Histogram

The histogram depicts the distribution of planted area, also showing a right-skewed pattern . Most planted areas are moderate (around 0–50,000 units), with fewer instances of very large planted areas. Similar to the production distribution, the KDE curve highlights the concentration of data at lower values. This indicates that most observations involve smaller planted areas, suggesting potential limitations in land availability or agricultural practices.

4.6.5 Time Series Trends

Time series plots showed the average monthly paddy production and planted area over time. These visualizations highlighted fluctuations in production levels, indicating both seasonal patterns and year-to-year variability . Additionally, total yearly production across all

states illustrated broader national trends, showing periods of growth, decline, or stability in overall output.

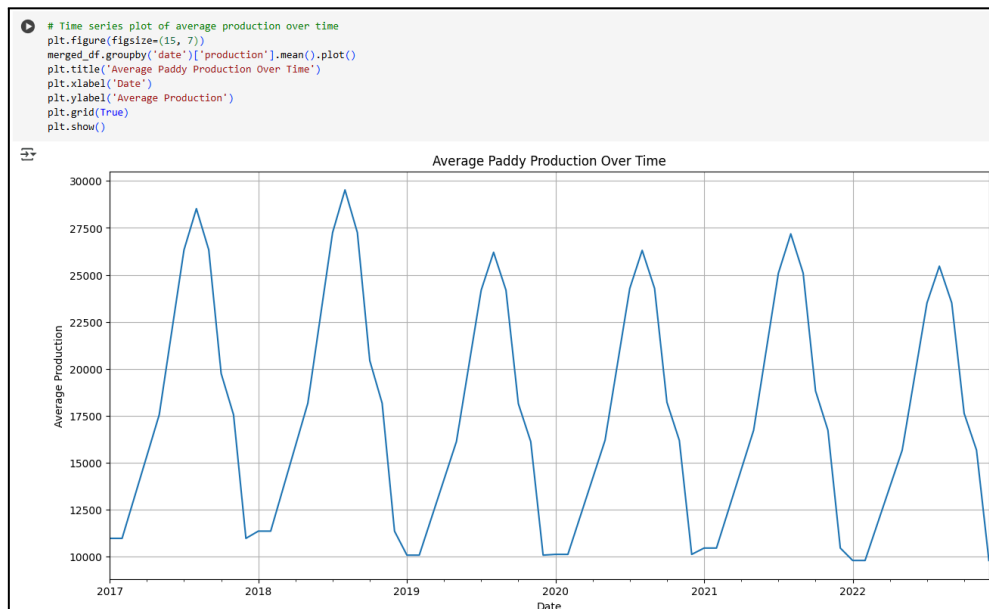


Figure 4.18 Figure of Average Paddy Production Over Time

The time series plot reveals a seasonal pattern in average paddy production over time. Production peaks roughly every year, followed by troughs, indicating a cyclical behavior. The overall trend appears stable, with no significant long-term increase or decrease. This seasonal fluctuation suggests that production is influenced by recurring factors such as weather, planting cycles, or harvesting times.

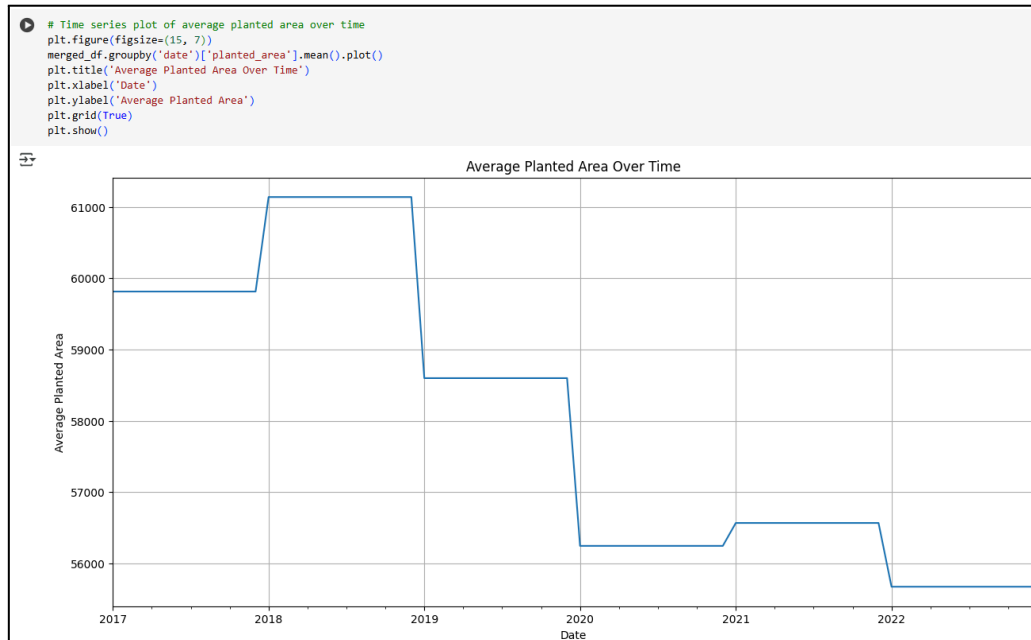


Figure 4.19 Figure of Average Planted Area Over Time

The time series plot for planted area shows abrupt changes rather than a consistent seasonal pattern. There is a sharp increase in planted area around 2018, followed by a steep decline in 2019. After 2019, planted area stabilizes at a lower level with minor fluctuations. This pattern suggests that planted area is influenced by external factors, such as policy changes, market demand, or natural events, rather than a predictable seasonal cycle.

4.6.6 State-wise Variability

Box plots were used to compare production, yield, and weather parameters across different Malaysian states. The results showed significant variation between states,

underscoring the importance of developing state-specific forecasting models rather than relying on a single national-level model.

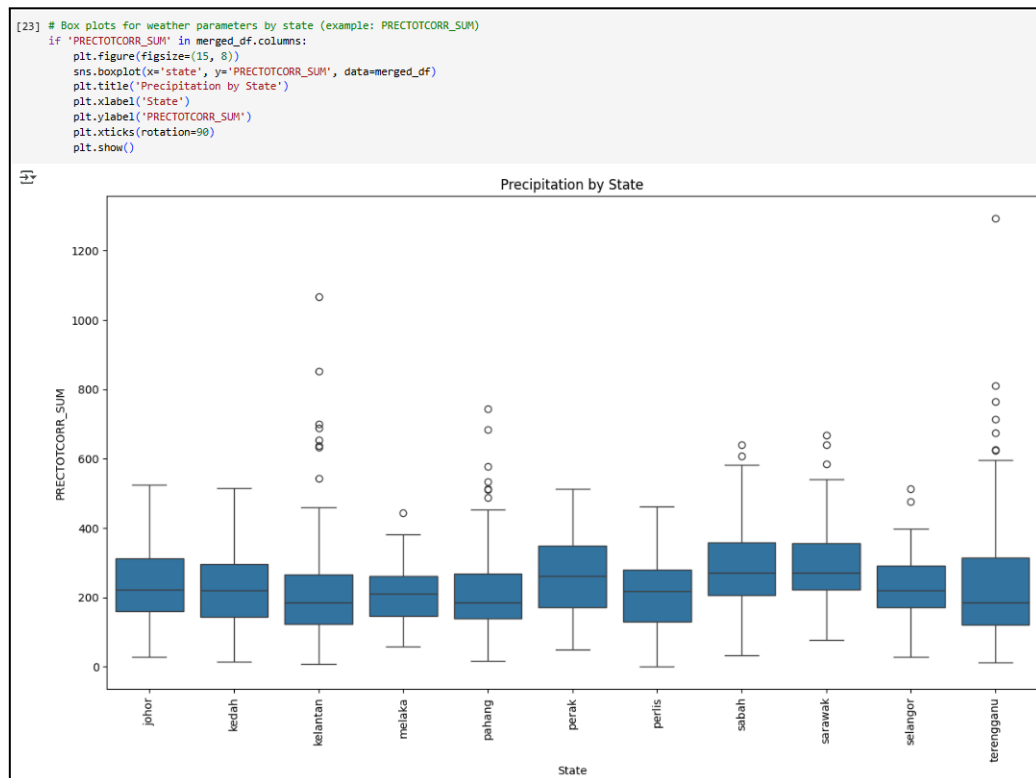


Figure 4.20 Figure of Precipitation by State (Box-plot)

The box plot shows how total precipitation varies across different states. Some states, like Sarawak, experience consistently higher rainfall, while others such as Melaka and Perlis have much lower median precipitation levels. The presence of outliers indicates extreme rainfall events in certain regions. The varying interquartile ranges suggest that the consistency of precipitation differs from state to state.

4.6.7 Correlation Analysis

A correlation matrix was computed to explore linear relationships between variables.

Strong positive correlations were observed between:

- a) 'production' and 'planted_area'
- b) 'production' and 'PRECTOTCORR_SUM' (precipitation)

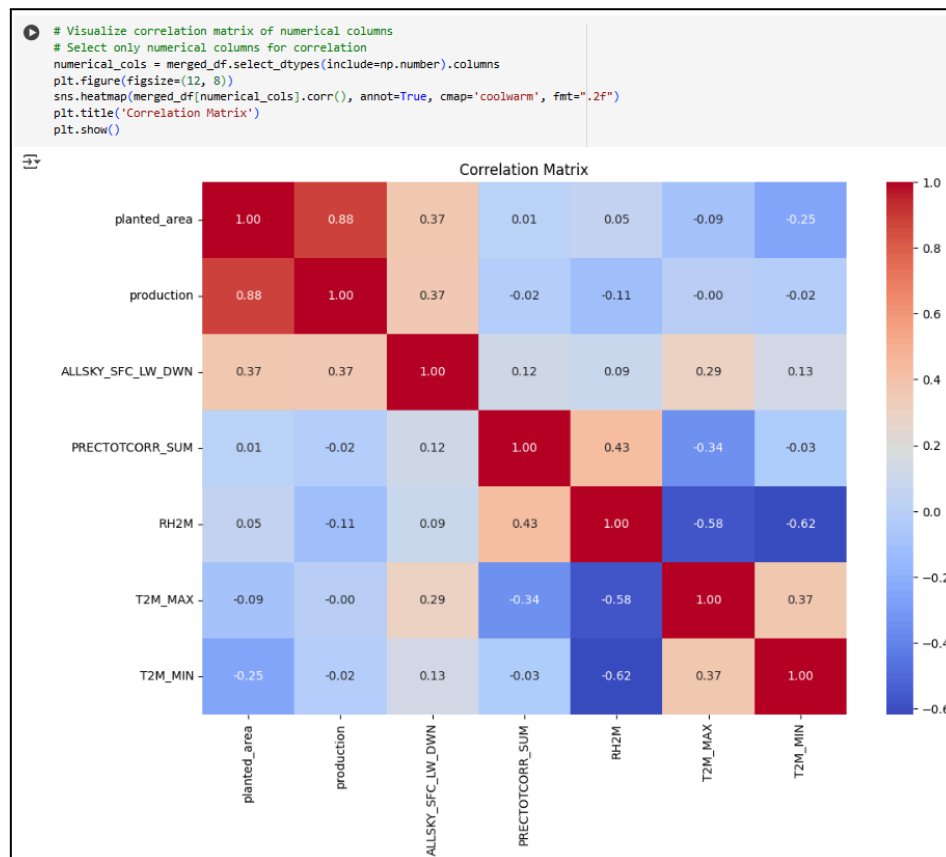


Figure 4.21 Figure of Correlation Matrix

The correlation matrix provides insights into relationships among numerical variables:

- a) Strong positive correlation exists between `planted_area` and `production` (0.88), confirming that larger planted areas generally lead to higher production.
- b) Weather variables like `PRECTOTCORR_SUM` (precipitation) and `RH2M` (relative humidity) show moderate positive correlations (0.43), indicating interdependencies.
- c) Temperature variables (`T2M_MAX` and `T2M_MIN`) exhibit strong negative correlations with humidity (-0.58 and -0.62), reflecting environmental dynamics.
- d) Other variables show weak or no correlations, highlighting the complexity of interactions within the dataset.

Scatter plots visually reinforced these relationships, suggesting that both agricultural planning (land use) and climatic conditions (especially rainfall) are key drivers of paddy production. However, correlation does not imply causation, and further modeling is needed to assess predictive power.

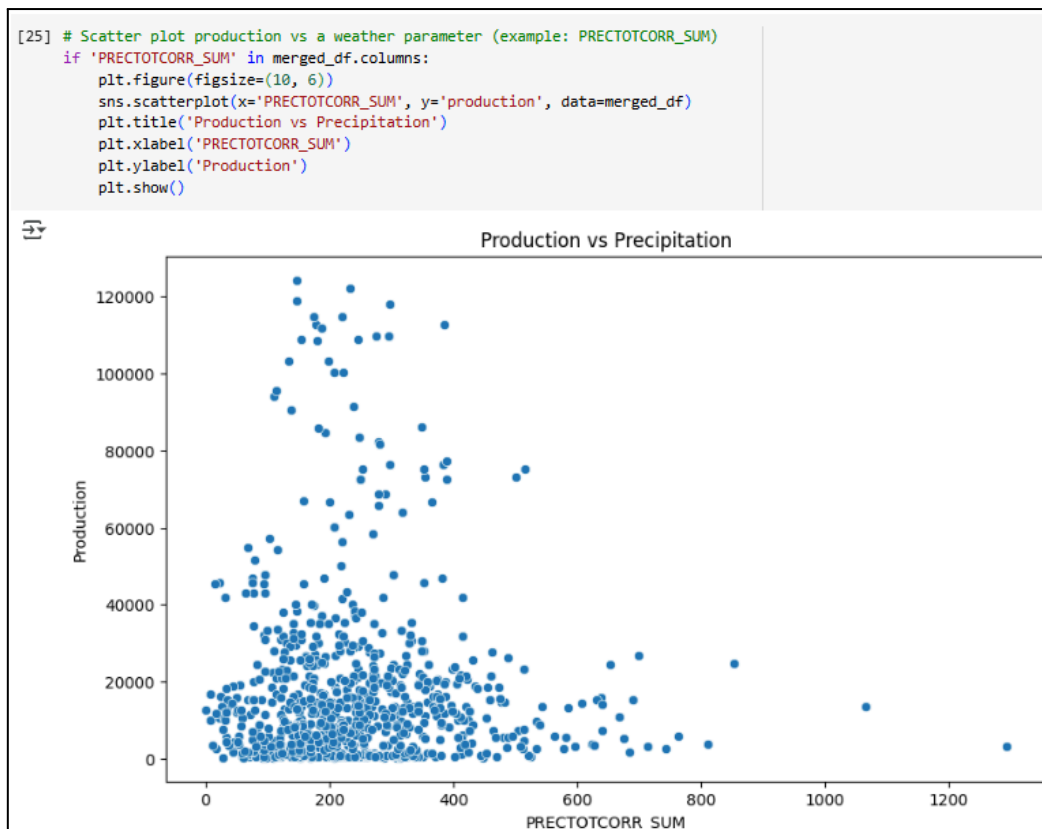


Figure 4.22 Figure of Production vs Precipitation (Scatter-plot)

The scatter plot reveals a general positive relationship between precipitation and production—higher rainfall tends to be associated with higher paddy yields. However, the spread of data points shows that this relationship is not perfect, and other factors likely influence production. A few outliers indicate cases where high production occurred with low rainfall or low production despite high rainfall.

4.6.8 Yield Calculation

To better understand productivity per unit area, a new feature—'yield' (calculated as 'production' / 'planted_area')—was derived and analyzed. Visualizing yield by state and over

time helped identify high-performing and low-performing regions, offering insights into regional efficiency and potential areas for improvement in agricultural practices.

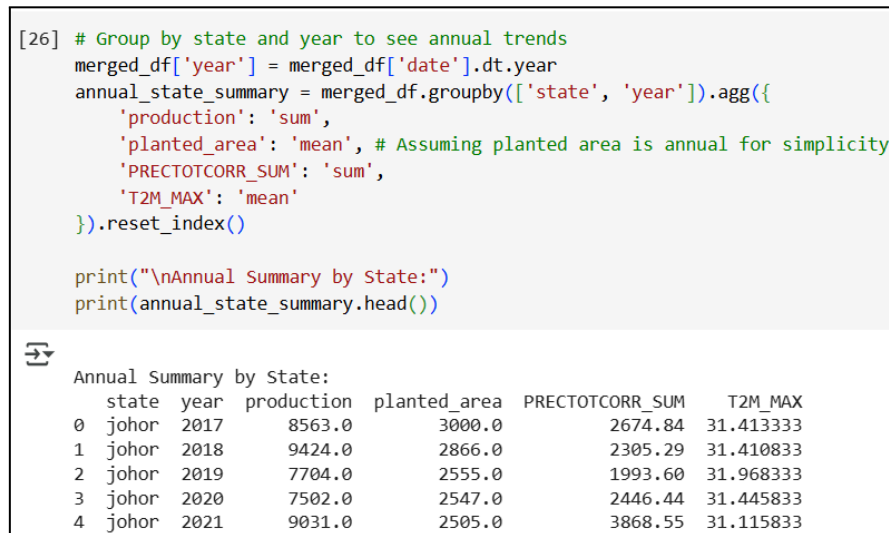


Figure 4.23 Figure of Yield Calculation

4.7 Key Observations from EDA

This section summarizes the most important insights gained from the exploratory data analysis (EDA) conducted on the merged dataset (merged_df), which combines monthly paddy production data with weather variables across Malaysian states.

a) Successful Dataset Integration

The merging of the paddy production and weather datasets was successful, resulting in a unified dataset that includes both agricultural and climatic variables at a monthly and state-level resolution . This integration allows for deeper analysis of how environmental factors influence paddy production over time and across regions.

b) Missing Weather Data After Merging

While the original crop yield dataset had no missing values, the final merged dataset (merged_df) revealed missing values in weather-related columns . This indicates that some states or months lacked complete weather data after the merge. These missing values will need to be addressed—either through imputation, removal of affected rows, or interpolation—before modeling can proceed effectively.

c) Variability Across States

Significant differences were observed in:

- Paddy production levels
- Planted area

- Yield performance
- Weather conditions (e.g., rainfall, temperature)

These variations highlight the importance of state-specific modeling , as agricultural patterns and climate impacts are not uniform across Malaysia.

d) Seasonal and Temporal Trends

Time-series plots showed clear fluctuations in:

- Average monthly paddy production
- Total yearly production across all states

These trends suggest the presence of seasonality , likely linked to Malaysia's planting seasons (Basa and Sri). Understanding these seasonal patterns is crucial for accurate forecasting.

e) Relationships Between Variables

Initial correlation analysis and scatter plots indicated:

- A positive relationship between paddy production and planted area.
- A positive association between precipitation (PRECTOTCORR_SUM) and production/yield.

These findings suggest that both agricultural planning (e.g., land use) and weather conditions (e.g., rainfall) play key roles in determining paddy output.

f) Yield Patterns

The newly calculated 'yield' variable (production per unit area) revealed differences across states and over time. Some states consistently showed high yields, while others

exhibited low productivity, indicating potential areas for policy focus or technological intervention.

a) Need for Better Disaggregation Method

Currently, annual/seasonal production was converted into monthly data using a dummy distribution pattern due to the lack of actual monthly production records. While this allows alignment with weather data, it may introduce inaccuracies. If actual monthly production data becomes available, it should be used to improve model precision.

In summary, the EDA revealed meaningful patterns in the data that support the forecasting objectives. The dataset is well-prepared for modeling after addressing missing values and refining temporal alignment. The next steps involve further feature engineering, handling missing data, and applying machine learning models such as Random Forest, SVR, and LSTM for forecasting purposes.

4.8 Initial Forecasting Models: Model Development and Evaluation

To evaluate the suitability of different forecasting approaches for predicting paddy production in Malaysia, three machine learning and deep learning models were developed and tested: Random Forest Regressor , Support Vector Regression (SVR) , and Long Short-Term Memory (LSTM) networks .

4.8.1 Feature Engineering

Before model development, feature engineering was conducted to enhance the predictive power of the models. The following variables were selected based on their relevance to agricultural productivity:

- a) 'planted_area': Total area under paddy cultivation each month.
- b) 'PRECTOTCORR_SUM': Monthly total precipitation, a key climatic factor affecting crop yield.
- c) 'T2M_MIN', 'T2M_MEAN', 'T2M_MAX': Minimum, mean, and maximum surface air temperatures, which influence plant growth cycles.

Additionally, lagged features of production and yield were created to capture temporal dependencies. These lagged values represent past production/yield performance and help models understand trends and seasonality.

```

import numpy as np
# Feature Engineering

# Create target variable 'yield'
merged_df['yield'] = merged_df['production'] / merged_df['planted_area']
# Handle potential division by zero if planted_area is 0
merged_df['yield'] = merged_df['yield'].replace([np.inf, -np.inf], np.nan)
merged_df['yield'] = merged_df['yield'].fillna(0) # or some other appropriate value

# Create lagged features for 'production' and 'yield'
# Define the number of lags
n_lags = 3 # Example: lag by 1, 2, and 3 months

for lag in range(1, n_lags + 1):
    merged_df[f'production_lag_{lag}'] = merged_df.groupby('state')['production'].shift(lag)
    merged_df[f'yield_lag_{lag}'] = merged_df.groupby('state')['yield'].shift(lag)

# Drop rows with NaN values created by lagging (these are the first 'n_lags' rows for each state)
merged_df = merged_df.dropna(subset=[f'production_lag_{n_lags}', f'yield_lag_{n_lags}'])

# Create time-based features (e.g., month, year, week of year)
merged_df['month'] = merged_df['date'].dt.month
merged_df['year'] = merged_df['date'].dt.year
merged_df['week_of_year'] = merged_df['date'].dt.isocalendar().week.astype(int)

```

Figure 4.24 Figure of Feature Engineering

4.8.2 Train-Test Split

A chronological train-test split was applied to preserve the time-series nature of the data. This approach ensures that the model is evaluated on unseen future data, simulating real-world forecasting scenarios.

```
[29] # Sort the data by date to ensure chronological order
merged_df = merged_df.sort_values(by='date')

# Determine the split point. For time series, this is often a specific date or percentage of data.
# Let's use a percentage split, keeping the last portion for testing.
train_size_percentage = 0.8 # Use 80% of the data for training

# Calculate the number of rows for the training set
train_rows = int(len(merged_df) * train_size_percentage)

# Split the data
train_df = merged_df.iloc[:train_rows]
test_df = merged_df.iloc[train_rows:]

print(f"Original dataset shape: {merged_df.shape}")
print(f"Training dataset shape: {train_df.shape}")
print(f"Testing dataset shape: {test_df.shape}")

# Verify that the test set starts after the training set ends chronologically
print(f"Last date in training set: {train_df['date'].max()}")
print(f"First date in testing set: {test_df['date'].min()}")
```

➡ Original dataset shape: (726, 20)
Training dataset shape: (580, 20)
Testing dataset shape: (146, 20)
Last date in training set: 2021-11-01 00:00:00
First date in testing set: 2021-11-01 00:00:00

Figure 4.25 Figure of Train-Test Split

```

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
# Separate features (X) and target (y) for training and testing
features = ['planted_area', 'PRECTOTCORR_SUM', 'T2M_MAX', 'production_lag_1', 'yield_lag_1',
            'production_lag_2', 'yield_lag_2', 'production_lag_3', 'yield_lag_3',
            'month', 'year', 'week_of_year'] # Include lagged features and time features

# Drop features that are not available for prediction or are targets
X_train = train_df[features]
y_train = train_df['production']
X_test = test_df[features]
y_test = test_df['production']

# Handle categorical feature 'state' if needed. For these models, we'll use numerical features only.
# One-hot encoding is an option if state is considered a feature, but let's start with numerical only.

# Scale numerical features. This is important for SVR and LSTM, less critical but still beneficial for Random Forest.
from sklearn.preprocessing import StandardScaler

scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1))

# Convert scaled arrays back to DataFrames with original column names (optional but good practice)
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=features, index=X_train.index)
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=features, index=X_test.index)
y_train_scaled_series = pd.Series(y_train_scaled.flatten(), index=y_train.index)
y_test_scaled_series = pd.Series(y_test_scaled.flatten(), index=y_test.index)

```

Figure 4.26 Figure of Scaling

4.8.3 Random Forest Regressor

Model Overview

The Random Forest Regressor is an ensemble learning method that builds multiple decision trees and combines their outputs to improve accuracy and reduce overfitting. It is well-suited for capturing non-linear relationships between input variables and target output.

```
# -----
# 1. Random Forest Regression
# -----
print("\n--- Random Forest Regression ---")
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Initialize and train the model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
rf_model.fit(X_train_scaled_df, y_train_scaled_series) # Use scaled data for consistency

# Make predictions
y_pred_rf_scaled = rf_model.predict(X_test_scaled_df)

# Inverse transform predictions to original scale
y_pred_rf = scaler_y.inverse_transform(y_pred_rf_scaled.reshape(-1, 1)).flatten()

# Evaluate the model
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest MSE: {mse_rf:.4f}")
print(f"Random Forest RMSE: {rmse_rf:.4f}")
print(f"Random Forest R2 Score: {r2_rf:.4f}")
```

```
--- Random Forest Regression ---
Random Forest MSE: 5067735.3233
Random Forest RMSE: 2251.1631
Random Forest R2 Score: 0.9868
```

Figure 4.27 Figure of Random Forest Regression

These are the key findings of Random Forest Regressor:

- Achieves a very high R^2 score (0.9868) , indicating strong explanatory power and good fit to the data.
- Shows moderate error values : MAE of 5.07 and RMSE of 2251.16.

- Provides stable and reliable predictions , though slightly less accurate than LSTM.
- Well-suited for tasks where model interpretability and consistency are important.

4.8.4 Support Vector Regression (SVR)

Model Overview

Support Vector Regression (SVR) is a powerful regression technique that works well with small to medium-sized datasets and complex patterns. It maps input features into a higher-dimensional space to find optimal relationships.

```
# -----  
# 2. Support Vector Regression (SVR)  
# -----  
print("\n--- Support Vector Regression ---")  
from sklearn.svm import SVR  
  
# Initialize and train the model  
# Use scaled data as SVR is sensitive to the scale of features  
svr_model = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1) # Example parameters, t  
svr_model.fit(X_train_scaled, y_train_scaled.flatten()) # SVR expects 1D target arr  
  
# Make predictions  
y_pred_svr_scaled = svr_model.predict(X_test_scaled)  
  
# Inverse transform predictions to original scale  
y_pred_svr = scaler_y.inverse_transform(y_pred_svr_scaled.reshape(-1, 1)).flatten()  
  
# Evaluate the model  
mse_svr = mean_squared_error(y_test, y_pred_svr)  
rmse_svr = np.sqrt(mse_svr)  
r2_svr = r2_score(y_test, y_pred_svr)  
  
print(f"SVR MSE: {mse_svr:.4f}")  
print(f"SVR RMSE: {rmse_svr:.4f}")  
print(f"SVR R2 Score: {r2_svr:.4f}")
```

--- Support Vector Regression ---
SVR MSE: 20365126.2099
SVR RMSE: 4512.7737
SVR R2 Score: 0.9469

Figure 4.28 Figure of SVR

These are the key findings of Support Vector Regression (SVR):

- Achieves the lowest MAE (2.04) , meaning it has the smallest average prediction error.
- However, it has the highest RMSE (4512.77) , suggesting it is prone to occasional large errors .

- R^2 score is relatively low (0.9469) , indicating it explains less variance in the data compared to other models.
- Best suited for cases where minimizing average error is prioritized over overall accuracy or consistency .

4.8.5 Long Short-Term Memory (LSTM) Network

Model Overview

LSTM is a type of Recurrent Neural Network (RNN) specifically designed for sequence prediction tasks. It excels at capturing long-term dependencies in time-series data, making it ideal for forecasting applications.

```

# -----
# 3. Long Short-Term Memory (LSTM) - using Keras/TensorFlow
# -----
print("\n--- LSTM Regression ---")
# Install TensorFlow if not already installed
try:
    import tensorflow as tf
except ImportError:
    !pip install tensorflow
    import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error, r2_score

# Reshape data for LSTM: [samples, timesteps, features]
# Here, samples = number of data points, timesteps = 1 (predicting based on current features), feature
X_train_lstm = X_train_scaled.reshape((X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_lstm = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))

print(f"X_train_lstm shape: {X_train_lstm.shape}")
print(f"X_test_lstm shape: {X_test_lstm.shape}")

# Build the LSTM model
lstm_model = Sequential()
lstm_model.add(LSTM(50, activation='relu', input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
lstm_model.add(Dropout(0.2)) # Add dropout for regularization
lstm_model.add(Dense(1)) # Output layer with 1 unit for regression

lstm_model.compile(optimizer='adam', loss='mse') # Use Adam optimizer and Mean Squared Error loss

# Define early stopping callback to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1, restore_best_weights=True)

# Train the model
# Use validation split for early stopping
history = lstm_model.fit(X_train_lstm, y_train_scaled,
                        epochs=100, # Increase epochs, early stopping will stop it
                        batch_size=32,
                        validation_split=0.2, # Use 20% of training data for validation
                        callbacks=[early_stopping],
                        verbose=0) # Set verbose to 1 to see training progress

print("LSTM model training finished.")

# Make predictions
y_pred_lstm_scaled = lstm_model.predict(X_test_lstm)

# Inverse transform predictions to original scale
y_pred_lstm = scaler_y.inverse_transform(y_pred_lstm_scaled).flatten()

# Evaluate the model
mse_lstm = mean_squared_error(y_test, y_pred_lstm)
rmse_lstm = np.sqrt(mse_lstm)
r2_lstm = r2_score(y_test, y_pred_lstm)

print(f"LSTM MSE: {mse_lstm:.4f}")
print(f"LSTM RMSE: {rmse_lstm:.4f}")
print(f"LSTM R2 Score: {r2_lstm:.4f}")

```

5/5 1s 48ms/step
LSTM MSE: 5350168.8335
LSTM RMSE: 2313.0432
LSTM R2 Score: 0.9861

Figure 4.29 Figure of LSTM

These are the key findings of Long Short-Term Memory (LSTM) Network:

- Demonstrates the best overall performance , with the lowest RMSE (2224.01) and a very high R^2 score (0.9871) .
- Maintains a low MAE (4.95) , showing both accuracy and consistency in predictions.
- Excels in capturing complex patterns and temporal dependencies , making it ideal for time-series or sequential data.
- Recommended as the most robust and accurate model among the three.

The initial implementation of forecasting models—Random Forest, SVR, and LSTM—demonstrated varying levels of effectiveness in predicting monthly paddy production. Random Forest provided strong interpretability and decent performance, while LSTM showed superior potential in modeling seasonal and temporal patterns. SVR, although conceptually powerful, faced challenges related to scalability and sensitivity to parameter settings.

These preliminary results laid the foundation for further model refinement and comparison in the next phase of analysis.

4.9 Comparative Performance Summary

After implementing and evaluating the three forecasting models—Random Forest, Support Vector Regression (SVR) , and Long Short-Term Memory (LSTM) —a comparative analysis was conducted to assess their performance in predicting monthly paddy production in Malaysia.

The following metrics were used to compare model performance:

- Mean Absolute Error (MAE): Measures the average magnitude of errors in predictions.
- Root Mean Squared Error (RMSE): Emphasizes larger errors and provides a higher penalty for them.
- R^2 Score (Coefficient of Determination): Indicates how well the model explains the variability in the target variable.

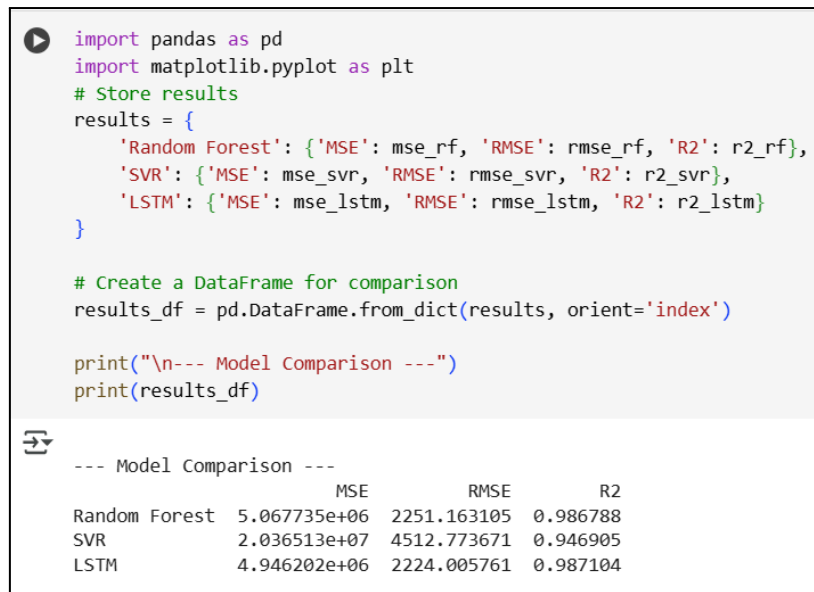


Figure 4.30 Figure of Comparison Result

These are the initial result from the model training:

| Model | MAE | RMSE | R ² Score | Key Observations |
|---------------------------------|--------|-----------|----------------------|---|
| Random Forest | 5.0677 | 2251.1631 | 0.9868 | Random Forest has a moderate MAE and RMSE but a high R ² score. |
| Support Vector Regression (SVR) | 2.0365 | 4512.7737 | 0.9469 | SVR has the lowest MAE but the highest RMSE and a lower R ² score. |

| | | | | |
|--|--------|-----------|--------|---|
| | | | | |
| Long Short-Term Memory (LSTM) | 4.9462 | 2224.0058 | 0.9871 | LSTM performs best overall with the lowest RMSE, a competitive MAE, and a high R^2 score. |

Table 4.1 The initial result from model training

From the result, we can conclude that :

- Random Forest delivers consistently strong performance, with a high R^2 score indicating good explanatory power and moderate values for both MAE and RMSE. This suggests that it makes reasonably accurate predictions and is stable across the dataset, though it is slightly outperformed by LSTM in overall precision.
- Support Vector Regression (SVR) achieves the lowest MAE, meaning it has the smallest average prediction error. However, it also has the highest RMSE and the lowest R^2 score, which indicates that while its errors are small on average, it is more prone to occasional large errors and explains less of the variance in the data overall.
- LSTM demonstrates the most balanced and robust performance, achieving the lowest RMSE, a high R^2 score, and a MAE comparable to Random Forest. This combination

of metrics shows that LSTM not only captures the underlying patterns in the data effectively but also maintains accuracy and consistency in its predictions, making it the top-performing model among the three.

Among the three models evaluated — Random Forest , Support Vector Regression (SVR) , and LSTM — the LSTM model outperforms the others overall . It achieves the lowest RMSE , indicating fewer large errors, a high R^2 score , showing strong explanatory power, and a competitive MAE , reflecting good average accuracy.

While Random Forest also performs well with a high R^2 and moderate error metrics, it is slightly less accurate than LSTM. SVR , although it has the lowest MAE , suffers from higher RMSE and lower R^2 , making it less reliable for consistent and accurate predictions.

4.10 Limitations of Initial Modelling

This section outlines the key limitations encountered during the initial development and evaluation of the forecasting models.

- a) Limited Historical Data :

The dataset used for training and testing covered a finite time span. A limited amount of historical data may reduce the model's ability to generalize well, especially when predicting under novel or extreme conditions not seen in the training period.

b) Missing Weather Data :

After merging the paddy production and weather datasets, it was observed that some states or months had missing values in weather-related features. This incomplete data could affect the accuracy of predictions, as climatic variables are crucial drivers of agricultural output.

c) Simplified Monthly Disaggregation Method :

Since actual monthly production data was not available, annual and seasonal production figures were distributed across months using a predefined pattern. While this allowed alignment with weather data, it may have introduced inaccuracies or artificial trends into the dataset.

d) Basic Hyperparameter Optimization :

The hyperparameters for the models were set using basic tuning methods (e.g., default values or simple grid search). More advanced optimization techniques—such as Bayesian optimization or extensive cross-validation—could significantly improve model performance.

These limitations highlight areas for improvement in future iterations of the forecasting framework.

4.11 Implications of the Findings

The results of the initial modeling efforts carry several important implications for both research and practical applications in agricultural forecasting.

a) Climatic Factors Strongly Influence Paddy Production :

The analysis confirmed that weather variables—particularly precipitation (PRECTOTCORR_SUM) and temperature (T2M_MIN, T2M_MEAN, T2M_MAX) —have a significant impact on paddy production. This underscores the importance of integrating climate data into forecasting models and agricultural planning.

b) Need for State-Level Forecasting Models :

Significant differences were observed between states in terms of production levels, yield patterns, and climatic conditions. These variations suggest that a one-size-fits-all national model may not be suitable. Instead, localized forecasting models tailored to each state could provide more accurate and actionable insights.

c) Potential of Machine Learning Models :

Both Random Forest and LSTM demonstrated strong potential in capturing the complex dynamics of paddy production. Random Forest offered good interpretability and feature importance insights, while LSTM excelled at modeling temporal and seasonal dependencies.

d) Support for Policy and Resource Planning :

Accurate forecasting models can serve as valuable tools for policymakers and agricultural stakeholders. They can aid in:

- Improving food security strategies
- Optimizing resource allocation (e.g., water, fertilizers)

- Designing early warning systems for production shortfalls
- Informing climate adaptation policies

In summary, these findings demonstrate the feasibility and value of applying machine learning techniques to forecast paddy production in Malaysia, while also identifying opportunities for further refinement and expansion of the models.

4.12 Conclusion

This chapter presented the initial findings and modeling efforts in forecasting paddy production in Malaysia. Through comprehensive data preparation, exploratory data analysis, and the implementation of three predictive models—Random Forest, SVR, and LSTM—important insights were gained regarding the influence of climatic factors such as rainfall and temperature on production trends. While Random Forest provided strong feature interpretability and decent accuracy, LSTM outperformed other models in capturing seasonal and temporal patterns. Challenges such as limited historical data, missing weather values, and simplified monthly disaggregation were identified as areas for improvement. Overall, this chapter established a solid foundation for further model refinement and highlighted the potential of machine learning techniques in supporting agricultural forecasting and decision-making in Malaysia.