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

Presented by:
NURHAFIZAH BINTI MOHD YUNOS (MCS241048)

JUNE 2025

RESEARCH BACKGROUND AND PROBLEM

Importance of Rice in Malaysia

Why rice is important:

- Staple food for the population
- Contributes significantly to agricultural GDP
- Still dependent on imports despite government efforts
- Grown mainly in MADA, KADA, and IADA regions

Challenges of Rice Production in Malaysia

Challenges:

- Unpredictable weather (rainfall shifts, floods, droughts)
- Climate change → more extreme events
- Pest outbreaks, land use changes
- Affects crop cycles and yield stability

OBJECTIVES
AND
RESEARCH GAPS

Objectives

Objective of the project:

- Analyze historical rice production and climate trends in Malaysia
- Identify climate variables that strongly affect rice yield
- Develop and train machine learning models (RF, SVR, LSTM)
- Compare model performance using accuracy metrics

Research Gaps

1. Limited Use of Machine Learning in Local Forecasting
   - Most studies in Malaysia still rely on traditional statistical models
   - Lack of advanced ML applications (e.g., Random Forest, SVR, LSTM)

2. Weak Integration of Climate Data
   - Many models ignore key climate variables
   - Use of outdated or low-resolution data sources

3. Lack of Model Comparison Studies
   - Few existing works evaluate multiple ML models side by side
   - No benchmarking to identify the most suitable model for Malaysia

4. Underuse of Time-Series Methods
   - Models rarely consider seasonal trends and temporal dependencies
   - Long Short-Term Memory (LSTM) remains underexplored in rice forecasting.

DATA SOURCES AND PREPROCESSING

Data Sources

1. Crop Production Data (2017–2022)

- Source: Department of Statistics Malaysia (DOSM)
- Monthly paddy production by state
- Includes: production, planted_area, yield, state, date

2. Climate Data

- Source: NASA POWER (Prediction of Worldwide Energy Resources)
- Monthly climate data by state
- Variables:
  - Rainfall (PRECTOTCORR_SUM)
  - Max/Min Temperature (T2M_MAX, T2M_MIN)
  - Humidity (RH2M)
  - Solar Radiation (ALLSKY_SFC_LW_DWN)

Data Sources

Crop Production (DOSM)

Climate Data (NASA POWER)

state

date

crop_type

planted_area

T2M_MAX

production

ALLSKY_SFC_LW_DWN

PRECTOTCORR_SUM

RH2M

T2M_MIN

Preprocessing

Cleaning & Formatting

- Renamed and standardized column names
- Converted annual rice production into monthly distribution
- Removed missing values and duplicates

Merging Datasets

- Merged paddy production with climate data by state and month
- Aligned temporal granularity (monthly format from 2017–2022)

Preprocessing

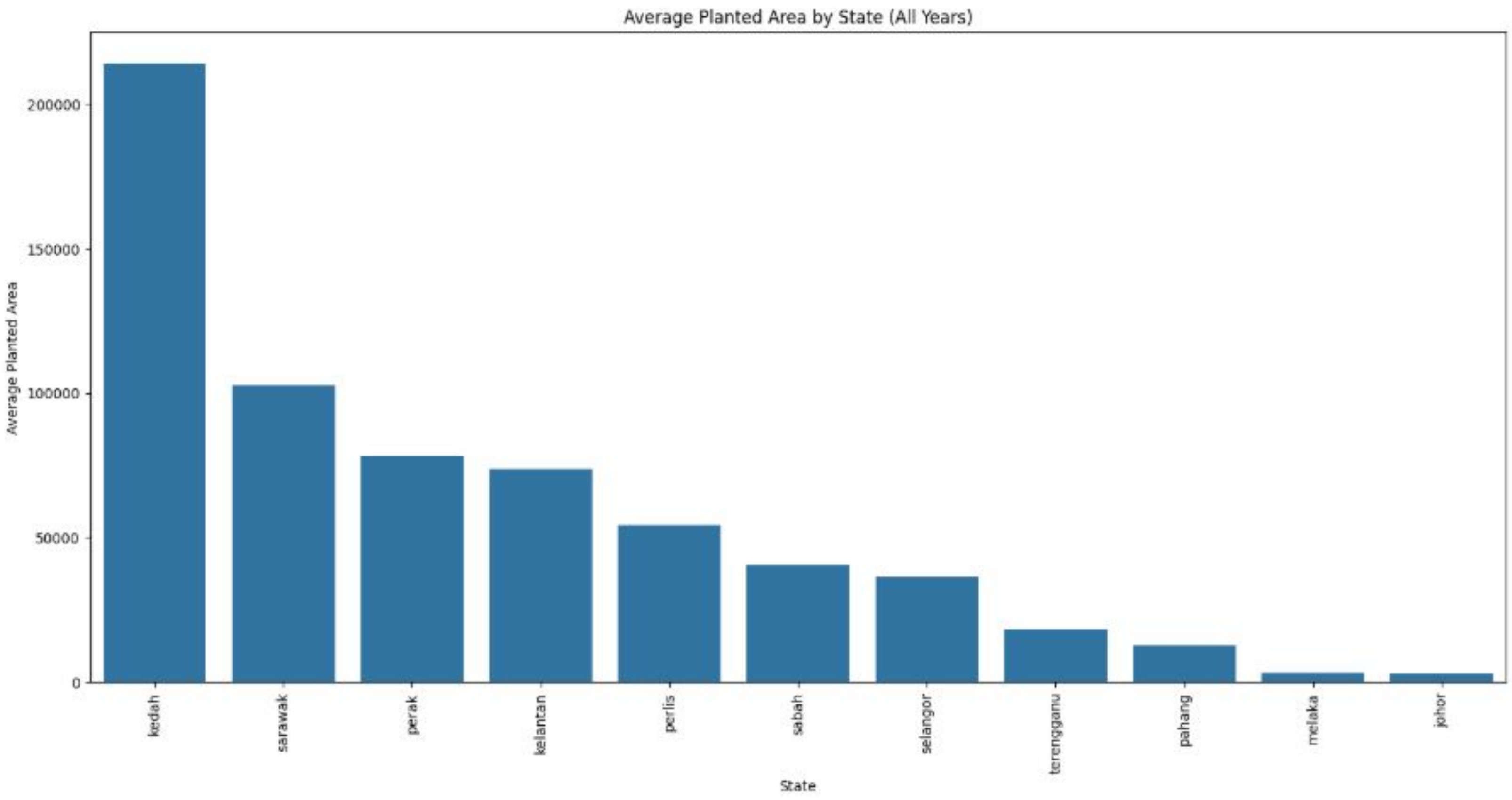
Feature Engineering

- Created lag features (e.g., production_lag_1, yield_lag_1)
- Generated time features: month, year, week_of_year
- Computed new variable: yield = production / planted area

Scaling (Normalization)

- Applied StandardScaler to input and output features
- Necessary for SVR and LSTM models to improve convergence

EDA (Total Paddy Production by State (All Years))

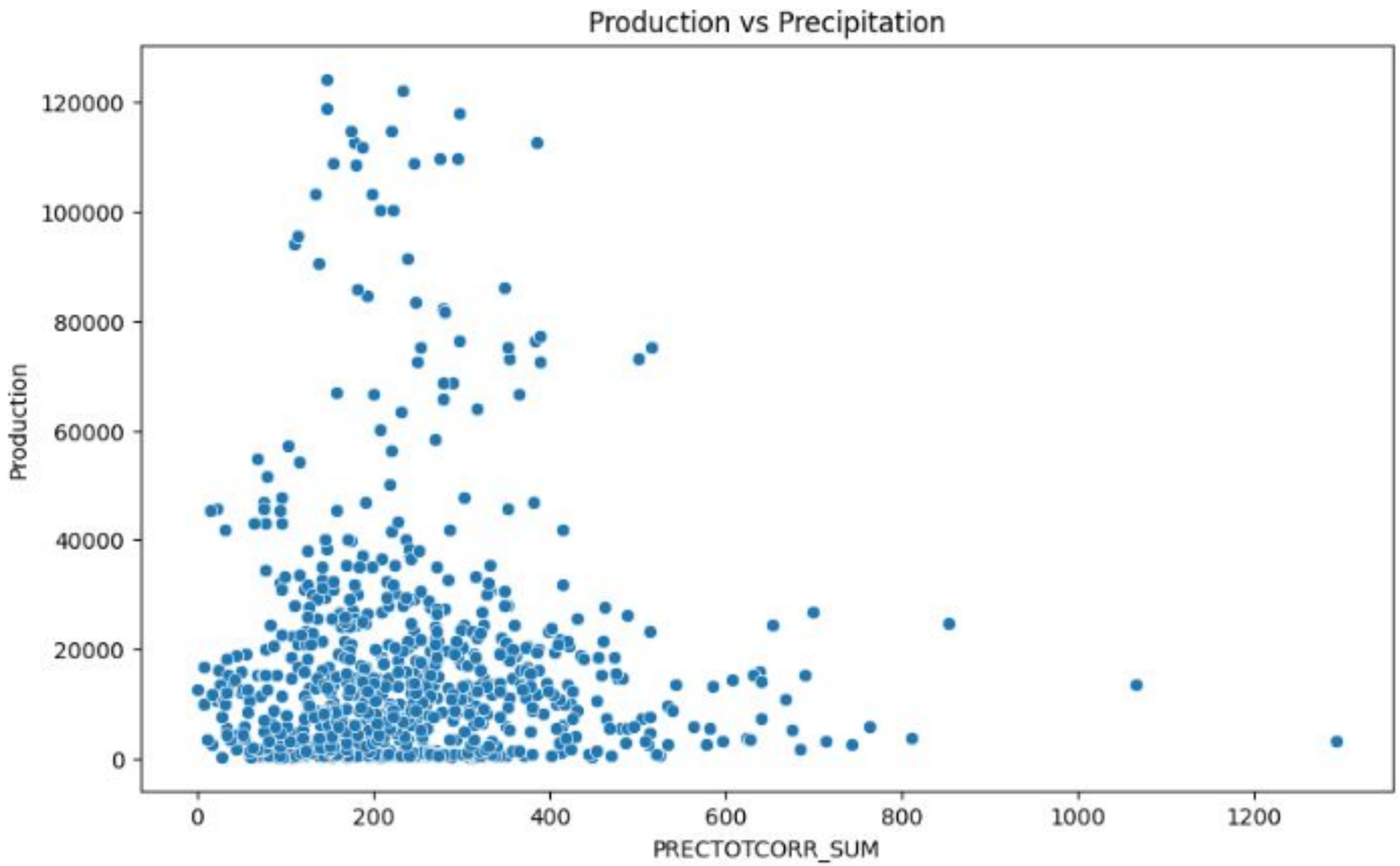


This graph shows the total amount of paddy (rice) produced in different states over all the years combined.

To compare how much each state contributes to the overall paddy production. Helps identify the top-producing states.

Insight:
Kedah with the tallest bars are the largest producers of paddy.

EDA (Production vs Precipitation)

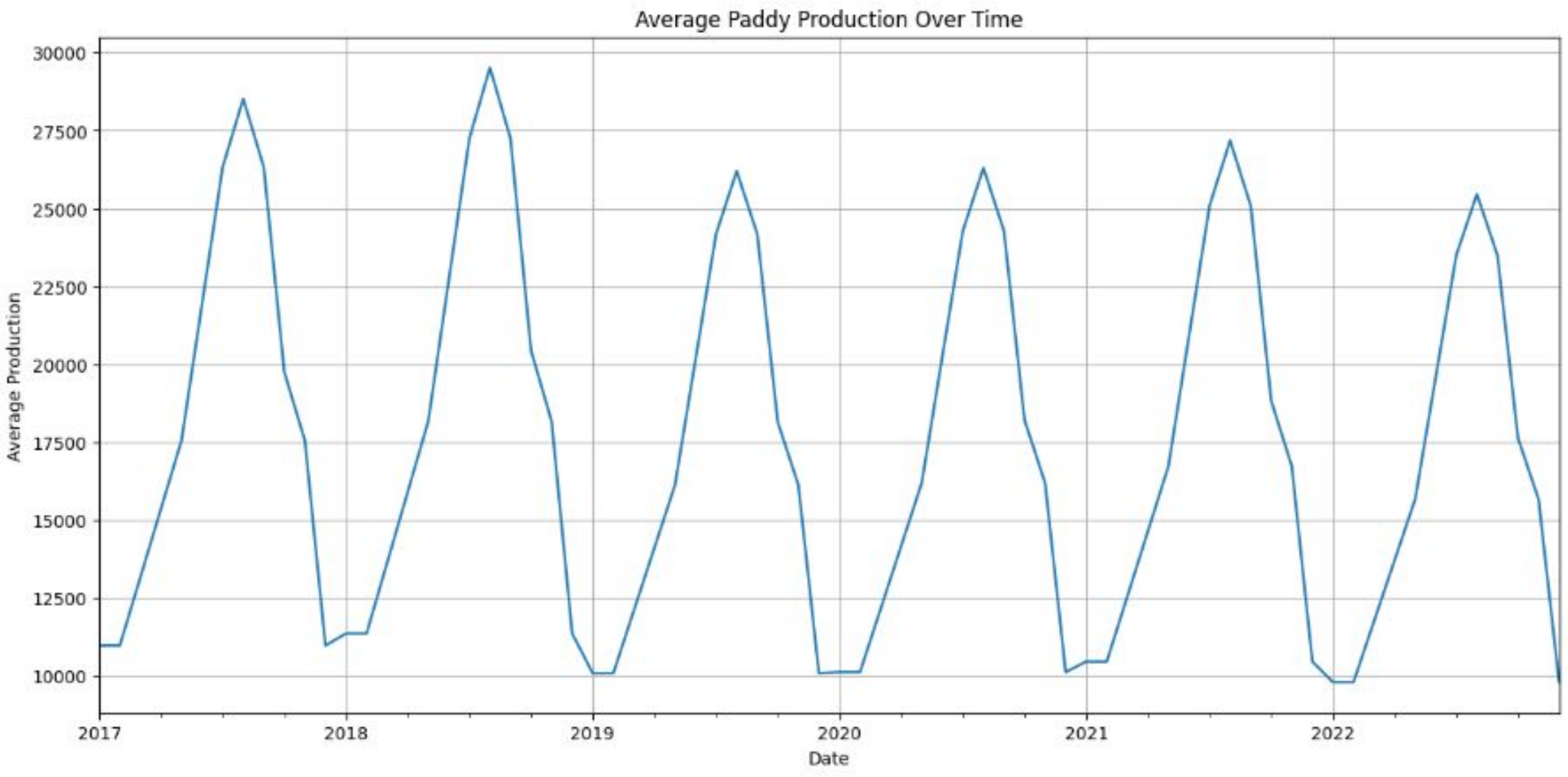


This graph compares paddy production with the amount of rainfall or precipitation received in the area.

To understand if there is a relationship between rainfall and crop yield. Helps determine whether more rain leads to higher production or if too much or too little rain affects production negatively.

Insight:
It may show a positive trend (more rain → more production), a negative trend, or no clear pattern, depending on other factors like irrigation and farming practices.

EDA (Average Paddy Production Over Time)

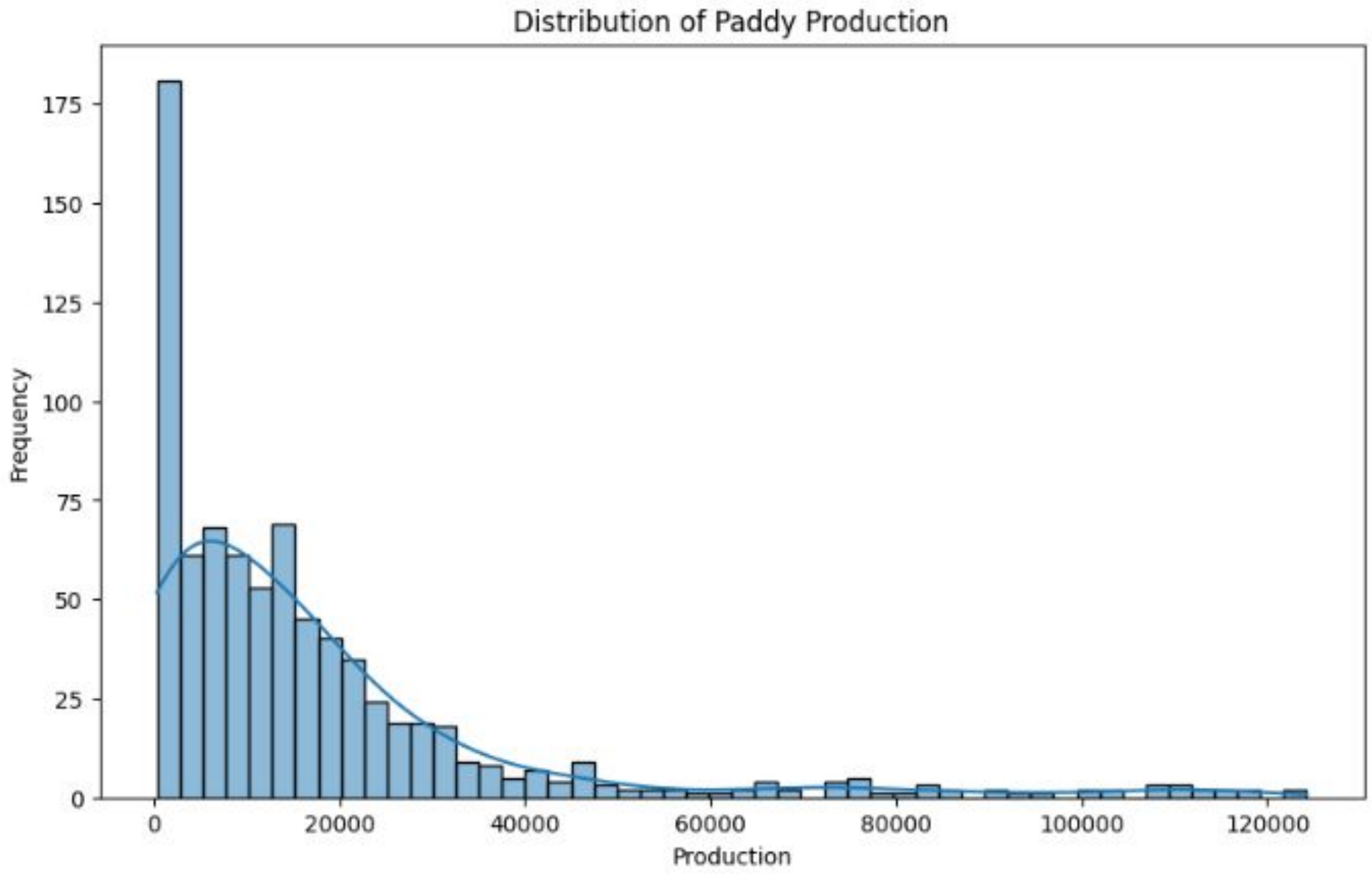


This graph shows the average amount of paddy produced per year over a period of time.

To observe trends in production across years. Helps detect increases, decreases, or stability in production levels.

Insight:
A rising line indicates improving production over time; a falling line could signal issues like droughts or policy changes affecting agriculture.

EDA (Distribution of Paddy Production)

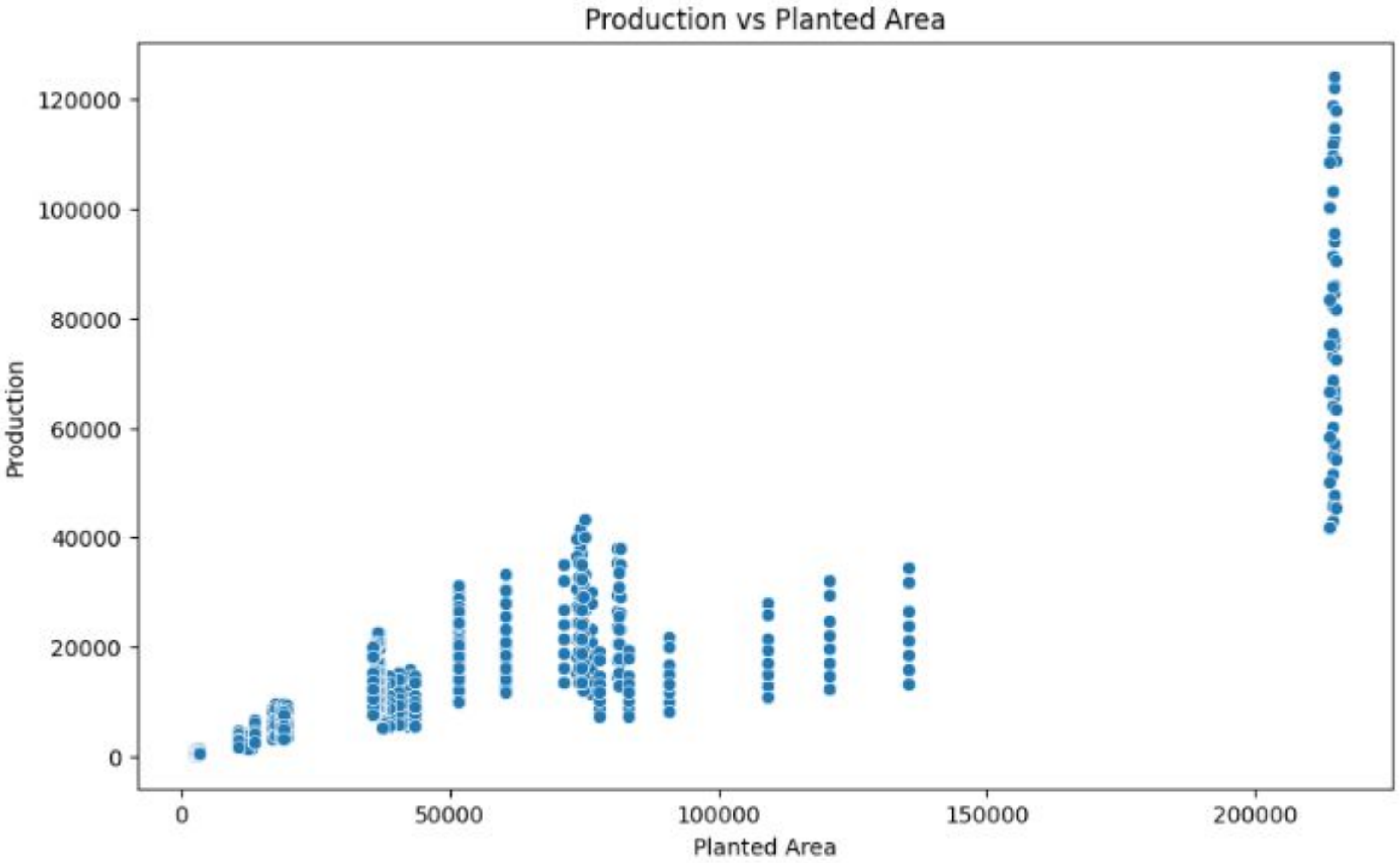


This graph shows how paddy production values are spread out or clustered.

To understand the variability in production amounts from year to year or region to region. Helps identify outliers or unusual data points.

Insight:
The histogram show that most years have similar production levels.

EDA (Production vs Planted Area)



This graph compares the amount of paddy produced against the area of land used for planting.

To see if larger planted areas result in higher production. Can help assess the productivity of land usage.

Insight:
If the points form a clear upward trend, it suggests that increasing the planted area generally increases production. If not, other factors like soil quality or farming techniques may be influencing yield.

MACHINE LEARNING MODELS

Common Features Used

- Climate: Rainfall, Max/Min Temp, Humidity
- Lagged production & yield (e.g., production_lag_1)
- Time features: Month, Year, Week of Year

Random Forest Regressor (RF)

- Ensemble of decision trees
- Handles non-linear relationships well
- Offers feature importance ranking
- Good interpretability

[45]
```python
# -------------------------------------------------------
# 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 consist

# 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: 7628189.4167
Random Forest RMSE: 2761.9177
Random Forest R2 Score: 0.9796
```

Support Vector Regression (SVR)

- Effective for small, high-dimensional datasets
- Uses RBF kernel to model non-linear patterns
- Sensitive to input scaling

```python
# ------------------------------------------------
# 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, tu
svr_model.fit(X_train_scaled, y_train_scaled.flatten()) # SVR expects 1D target arra

# 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: 19690735.8984
SVR RMSE: 4437.4245
SVR R2 Score: 0.9473
```

Long Short-Term Memory (LSTM)

- A type of recurrent neural network (RNN)
- Specializes in time series data and temporal dependency
- Captures long-term patterns in sequences
- Requires more computational resources

```python
# ---------------------------------------------------
# 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), featur
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)
```

```python
# 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}")
```

```
--- LSTM Regression ---
X_train_lstm shape: (607, 1, 12)
X_test_lstm shape: (152, 1, 12)
/usr/local/lib/python3.11/dist-packages/keras/src/l
  super().__init__(**kwargs)
Epoch 99: early stopping
Restoring model weights from the end of the best ep
LSTM model training finished.
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 107ms/step
LSTM MSE: 5242600.5277
LSTM RMSE: 2289.6726
LSTM R2 Score: 0.9860
```

RESULT
OF
EVALUATION METRICS

```
[48] import pandas as pd
     import matplotlib.pyplot as plt
     # Store results
     results = {
         'Random Forest': {'MSE': mse_rf, 'RMSE': rmse_rf, 'R2': r2_rf},
         'SVR': {'MSE': mse_svr, 'RMSE': rmse_svr, 'R2': r2_svr},
         'LSTM': {'MSE': mse_lstm, 'RMSE': rmse_lstm, 'R2': r2_lstm}
     }

     # Create a DataFrame for comparison
     results_df = pd.DataFrame.from_dict(results, orient='index')

     print("\n--- Model Comparison ---")
     print(results_df)
```

```
--- Model Comparison ---
                        MSE         RMSE        R2
Random Forest  7.628189e+06  2761.917706  0.979581
SVR            1.969074e+07  4437.424467  0.947291
LSTM           5.242601e+06  2289.672581  0.985966
```

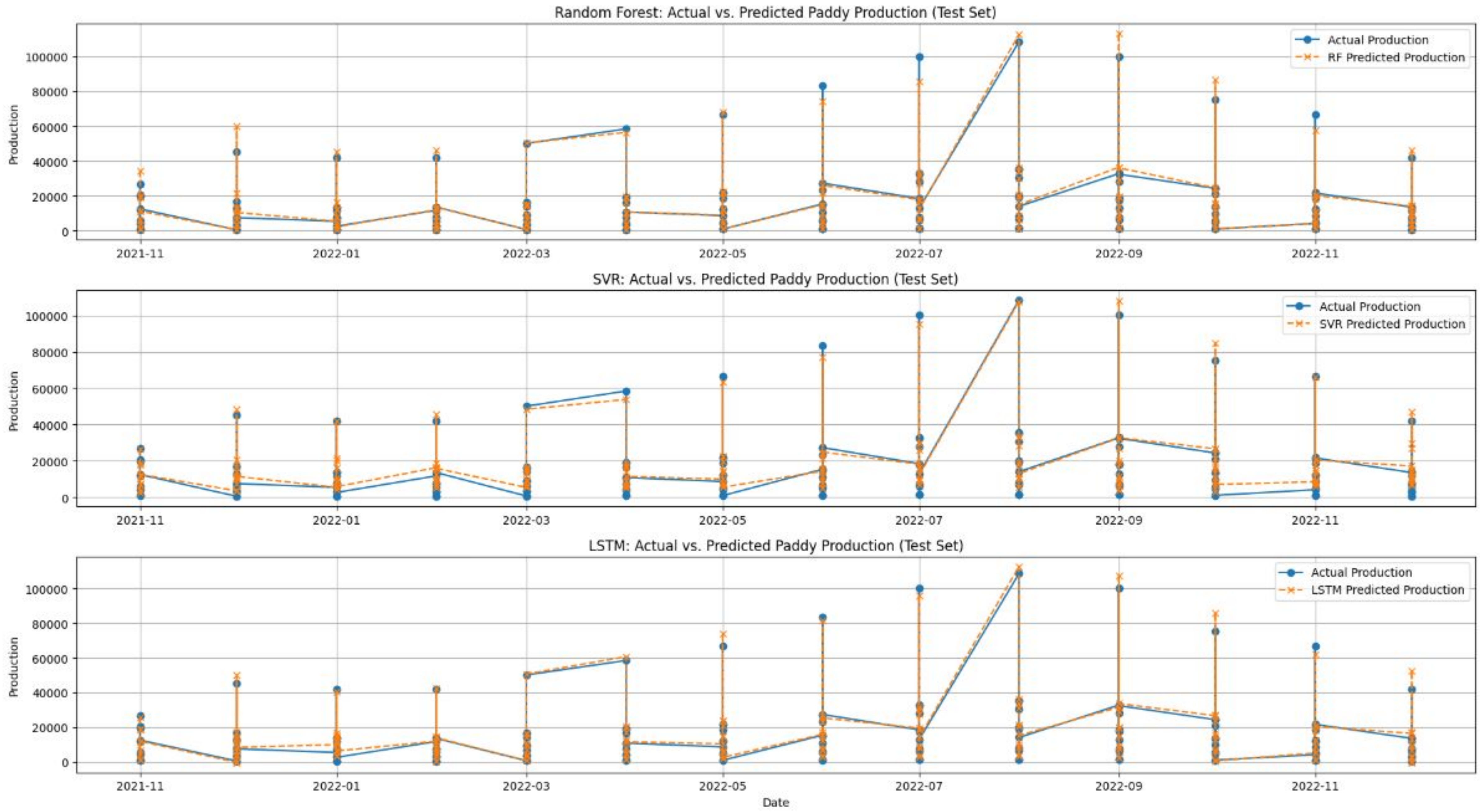
Result of Evaluation Metric

Model	MAE	RMSE	R2 Score
Random Forest	7.6289	2761.9177	0.9796
Support Vector Regression (SVR)	1.9691	4437.4245	0.9473
Long Short-Term Memory (LSTM)	5.2426	2289.6726	0.9860

Conclusion from Result

Based on the performance metrics presented in the table, the Long Short-Term Memory (LSTM) model demonstrates the best overall performance for forecasting, with the highest R^2 score of 0.9860, indicating excellent predictive accuracy. Although Support Vector Regression (SVR) has the lowest Mean Absolute Error (MAE) of 1.9691, it also has the highest Root Mean Square Error (RMSE) of 4437.4245, suggesting inconsistent prediction accuracy. LSTM offers a balanced and superior performance across all three evaluation metrics—achieving a low RMSE of 2289.6726 and a moderate MAE of 5.2426.

The best ML :Long Short-Term Memory (LSTM)


Random Forest: Actual vs. Predicted Paddy Production (Test Set)
Actual Production
RF Predicted Production
Production
100000
80000
60000
40000
20000
0
2021-11
2022-01
2022-03
2022-05
2022-07
2022-09
2022-11
SVR: Actual vs. Predicted Paddy Production (Test Set)
Actual Production
SVR Predicted Production
LSTM: Actual vs. Predicted Paddy Production (Test Set)
Actual Production
LSTM Predicted Production
Date

These plots highlight the strengths and weaknesses of each model in predicting paddy production. While all models attempt to follow the trend of actual production, they exhibit varying degrees of accuracy, particularly during rapid changes in production levels. This comparison helps evaluate which model performs best for forecasting paddy production based on historical data.

Random Forest: While it captures some trends, it struggles with sudden changes in production, indicating that it may not fully account for temporal dependencies.

SVR: This model tends to smooth out fluctuations, which can be beneficial for stable trends but less effective for capturing rapid changes.

LSTM: As a sequential model designed to handle time-series data, LSTM demonstrates a closer alignment with the actual production curve, especially during periods of high variability. This suggests that LSTM is better at capturing temporal patterns and adapting to sudden changes in production.

Why LSTM Performs Best:

1. Temporal Dependencies: LSTM is specifically designed to handle sequential data and can capture long-term dependencies, making it well-suited for forecasting time-series data like paddy production.

2. Handling Variability: The graph shows that LSTM performs better during periods of sharp increases or decreases in production, indicating its ability to adapt to dynamic changes.

3. Alignment with Actual Data: The LSTM predictions are visually closer to the actual production curve compared to Random Forest and SVR, suggesting higher accuracy overall.

KEY FINDINGS
AND
CONTRIBUTIONS

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Contributions

- Developed a localized machine learning framework for rice yield forecasting in Malaysia
- First comparative study using RF, SVR, and LSTM for Malaysian paddy prediction
- Successfully integrated high-resolution NASA climate data with state-level agricultural records
- Supports data-driven decision making for food security, resource planning, and smart farming

Key Findings

- LSTM outperformed Random Forest and SVR with the lowest RMSE and highest R^2
- Rainfall and temperature were the most influential climate variables
- Lag features (e.g., previous month's production) significantly improved forecasting accuracy
- Seasonal trends and state-level variability were observed across Malaysia

CONCLUSION
AND
FUTURE WORK

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Conclusion

- LSTM model showed highest accuracy in forecasting paddy production
- Climate variables, especially rainfall and temperature, are key predictors
- Lagged production values and time features improve prediction performance
- Machine learning models outperform traditional methods for yield forecasting

Future Work/Recommendations

1. Incorporate additional variables such as:
   a. Soil quality
   b. Socio-economic factors
   c. Farm-level practices

2. Develop real-time forecasting systems
3. Expand model for other crops or regions in Malaysia
4. Improve data resolution and coverage beyond 2022

THANK YOU