Crop Yield Prediction: Random Forest with Optuna (Part 5)

Overview

This notebook trains a **Random Forest** model to predict crop yields. You can configure the specific target crop in the data loading section.

Methodology

1. **Crop Selection:** Choose the specific crop to predict.
2. **Feature Analysis:** Review the input variables.
3. **Time-Series Split:** Divide data by year to ensure we don't predict the past using the future:
    - **Train:** Learn patterns.
    - **Validation:** Tune settings.
    - **Test:** Final evaluation.
4. **Baseline:** Compare against a simple guess (Last Year's Yield).
5. **Initial Model:** Train a default model and check learning curves for errors.
6. **Optimization:** Use **Optuna** to automatically find the best model settings.
7. **Final Evaluation:** Compare accuracy (RMSE) across all stages.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
import optuna
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Optuna Visualization Tools
from optuna.visualization import plot_optimization_history
from optuna.visualization import plot_parallel_coordinate
from optuna.visualization import plot_slice
from optuna.visualization import plot_param_importances

# Set plotting style
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)
```

```
C:\Users\PavinP\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n
2kfra8p0\LocalCache\local-packages\Python312\site-packages\tqdm\auto.py:21: TqdmW
arning: IProgress not found. Please update jupyter and ipywidgets. See https://ip
ywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

1. Data Preparation and Crop Choice

We load the main dataset and identify the available crops. For this analysis, we focus specifically on **Rice**. We clean the data by removing columns related to other crops and deleting any rows where the target yield information is missing.

In [2]:
```
# Load dataset
df = pd.read_parquet('Parquet/XY_v2.parquet')

# --- LIST AVAILABLE CROPS ---
# Assumes targets start with 'Y_'
target_columns = [col for col in df.columns if col.startswith('Y_')]
available_crops = [col.replace('Y_', '') for col in target_columns]

print("--- Available Crops found in Dataset ---")
print(available_crops)
print("-" * 40)

# --- CONFIGURATION: SET CROP HERE ---
CHOSEN_CROP = 'rice'  # <--- CHANGE THIS to 'lettuce', 'pepper', etc. based on l
# -----------------------------------

# Define Target and Dynamic Lag Features
TARGET_COL = f'Y_{CHOSEN_CROP}'
LAG_1_FEATURE = f'avg_yield_{CHOSEN_CROP}_1y'

if TARGET_COL not in df.columns:
    raise ValueError(f"Target {TARGET_COL} not found in dataset. Check spelling.

print(f"Predicting Target: {TARGET_COL}")
print(f"Using Lag 1 Feature: {LAG_1_FEATURE}")

# Clean Missing Targets for the chosen crop
df_model = df.dropna(subset=[TARGET_COL])

print(f"Data Loaded. Rows with valid target: {len(df_model)}")
```

```
--- Available Crops found in Dataset ---
['bananas', 'barley', 'cassava_fresh', 'cucumbers_and_gherkins', 'maize_corn', 'o
il_palm_fruit', 'other_vegetables_fresh_nec', 'potatoes', 'rice', 'soya_beans',
'sugar_beet', 'sugar_cane', 'tomatoes', 'watermelons', 'wheat']
----------------------------------------
Predicting Target: Y_rice
Using Lag 1 Feature: avg_yield_rice_1y
Data Loaded. Rows with valid target: 4729
```

2. Selecting Features and Splitting Data

We identify the input variables (features), such as weather data and previous years' yields. To prevent the model from "cheating" by seeing the future, we split the data based on time:

- **Training Data:** Years before 2014.
- **Validation Data:** Years 2014 to 2018.
- **Test Data:** Years 2019 and later.

In [3]:
```
# --- DROP UNWANTED COLUMNS ---
# Drop all columns that start with "avg_yield_" but do NOT match the chosen crop
# Example: If predicting Rice, we drop 'avg_yield_lettuce_1y', etc.
cols_to_drop = [c for c in df_model.columns
                if c.startswith("avg_yield_") and CHOSEN_CROP not in c]

df_model = df_model.drop(columns=cols_to_drop)

# --- FEATURE SELECTION ---
# Select independent variables (exclude 'Y_' columns and metadata)
feature_cols = [c for c in df_model.columns
                if not c.startswith('Y_') and c not in ['area']]

# --- DISPLAY FEATURES TABLE ---
print(f"Total Features Used: {len(feature_cols)}")
print("-" * 30)
feature_preview = pd.DataFrame(feature_cols, columns=['Feature Name']).T
display(feature_preview)

# --- TIME-SERIES SPLIT ---
TRAIN_END_YEAR = 2014
VAL_END_YEAR = 2019

# 1. Training Set (< 2014)
mask_train = df_model['year'] < TRAIN_END_YEAR
X_train = df_model[mask_train][feature_cols]
y_train = df_model[mask_train][TARGET_COL]

# 2. Validation Set (>= 2014 and < 2019) -> This covers 2014 to 2018
mask_val = (df_model['year'] >= TRAIN_END_YEAR) & (df_model['year'] < VAL_END_YE
X_val = df_model[mask_val][feature_cols]
y_val = df_model[mask_val][TARGET_COL]

# 3. Test Set (>= 2019)
mask_test = df_model['year'] >= VAL_END_YEAR
X_test = df_model[mask_test][feature_cols]
y_test = df_model[mask_test][TARGET_COL]

print(f"\nTraining Samples   (<{TRAIN_END_YEAR})     : {len(X_train)}")
# Subtract 1 here to show the inclusive range (2014-2018)
print(f"Validation Samples ({TRAIN_END_YEAR}-{VAL_END_YEAR - 1}): {len(X_val)}")
print(f"Testing Samples    (>={VAL_END_YEAR})    : {len(X_test)}")
```

```
Total Features Used: 23
------------------------------
```

	0	1	2	3	4
Feature Name	year	avg_yield_rice_1y	avg_yield_rice_3y	avg_yield_rice_5y	sum_rain_winter	sum_rair

1 rows × 23 columns

```
Training Samples   (<2014)     : 3579
Validation Samples (2014-2018): 575
Testing Samples    (>=2019)    : 575
```

In [4]:
```
X_train.head()
```

Out[4]:

	year	avg_yield_rice_1y	avg_yield_rice_3y	avg_yield_rice_5y	sum_rain_winter	sum_rai
0	1982	2241.4	2181.766667	2097.64	150.63	
1	1983	2199.4	2204.533333	2156.56	139.74	
2	1984	2258.1	2232.966667	2200.56	60.60	
3	1985	2241.6	2233.033333	2222.66	68.62	
4	1986	2248.2	2249.300000	2237.74	75.65	

5 rows × 23 columns

3. Setting a Baseline

Before using complex AI, we create a simple baseline to measure success. We assume that the yield this year will be exactly the same as last year. We calculate the error (RMSE) of this simple guess to establish a score we must beat.

In [5]:
```python
# Baseline: yield(t) = yield(t-1)
y_pred_baseline = X_test[LAG_1_FEATURE]

# Clean NaNs for metric calculation
mask_valid = ~y_pred_baseline.isna() & ~y_test.isna()
y_test_clean = y_test[mask_valid]
y_pred_clean = y_pred_baseline[mask_valid]

rmse_baseline = np.sqrt(mean_squared_error(y_test_clean, y_pred_clean))
r2_baseline = r2_score(y_test_clean, y_pred_clean)

print(f"Baseline RMSE: {rmse_baseline:.2f}")
```

```
Baseline RMSE: 533.44
```

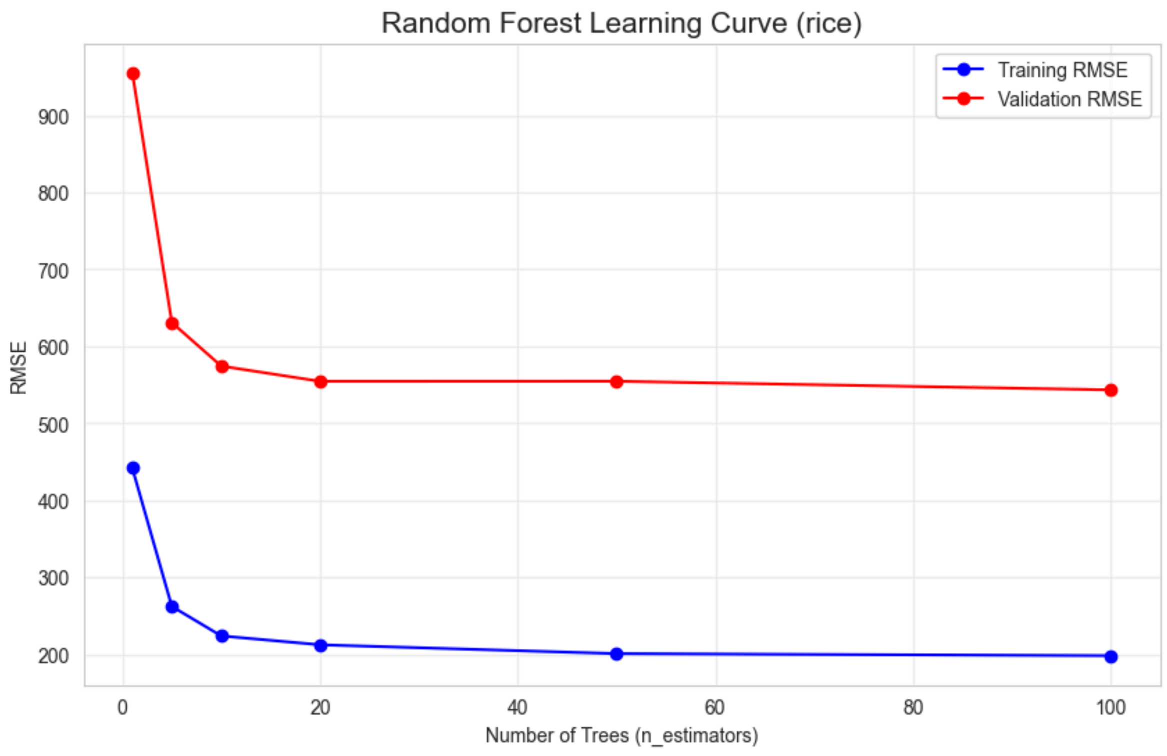
4. Initial Model Testing

We train a basic Random Forest model using default settings. We plot a learning curve to ensure the model is learning patterns rather than just memorizing data (overfitting). Since Random Forest (Scikit-Learn) doesn't allow step-by-step evaluation during fitting like LightGBM, we simulate the learning curve by training forests of increasing size.

In [6]:
```python
# --- PREPROCESSING FOR RANDOM FOREST ---
# Scikit-learn Random Forest does not handle NaNs natively.
# We use SimpleImputer to fill missing values with the mean.
imputer = SimpleImputer(strategy='mean')

# Fit on training data, transform others
X_train_imp = imputer.fit_transform(X_train)
X_val_imp = imputer.transform(X_val)
X_test_imp = imputer.transform(X_test)

# --- INITIAL MODEL TRAINING & LEARNING CURVE ---

# To visualize training progress, we train models with increasing n_estimators
```

```python
n_trees_list = [1, 5, 10, 20, 50, 100]
train_rmse_scores = []
val_rmse_scores = []

print("Training Random Forests for Learning Curve...")
for n in n_trees_list:
    rf = RandomForestRegressor(
        n_estimators=n,
        max_depth=None,          # unlimited depth initially
        min_samples_split=2,     # default
        min_samples_leaf=1,      # default
        random_state=42,
        n_jobs=-1
    )
    rf.fit(X_train_imp, y_train)

    # Evaluate
    train_pred = rf.predict(X_train_imp)
    val_pred = rf.predict(X_val_imp)

    train_rmse_scores.append(np.sqrt(mean_squared_error(y_train, train_pred)))
    val_rmse_scores.append(np.sqrt(mean_squared_error(y_val, val_pred)))

# Define the final initial model (usually 100 trees is default)
model_init = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
model_init.fit(X_train_imp, y_train)

# --- PLOT LEARNING CURVE (RMSE over Number of Trees) ---
def plot_learning_curve(n_trees_list, train_scores, val_scores):
    plt.figure(figsize=(10, 6))

    plt.plot(n_trees_list, train_scores, label='Training RMSE', color='blue', ma
    plt.plot(n_trees_list, val_scores, label='Validation RMSE', color='red', mar

    plt.title(f'Random Forest Learning Curve ({CHOSEN_CROP})', fontsize=15)
    plt.xlabel('Number of Trees (n_estimators)')
    plt.ylabel('RMSE')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()

plot_learning_curve(n_trees_list, train_rmse_scores, val_rmse_scores)

# Evaluate on TEST Set
y_pred_init_test = model_init.predict(X_test_imp)
rmse_init_test = np.sqrt(mean_squared_error(y_test, y_pred_init_test))
r2_init_test = r2_score(y_test, y_pred_init_test)

print(f"Initial Model Test RMSE: {rmse_init_test:.2f}")
```

```
Training Random Forests for Learning Curve...
```



```
Initial Model Test RMSE: 541.56
```

5. Tuning the Model (Optuna)

To improve performance, we use **Optuna** to find the best model settings (hyperparameters). We run 50 different trials, adjusting settings like tree depth, split criteria, and number of features, to minimize the error on the validation data.

In [7]:
```
# --- OPTUNA OBJECTIVE FUNCTION ---
def objective(trial):
    params = {
    'n_estimators': trial.suggest_int('n_estimators', 250, 300),  # narrowed arc
    'max_depth': trial.suggest_int('max_depth', 6, 10),            # around 7
    'min_samples_split': trial.suggest_int('min_samples_split', 2, 5),  # around
    'min_samples_leaf': trial.suggest_int('min_samples_leaf', 8, 10),   # around
    'max_features': trial.suggest_float('max_features', 0.9, 1.0),      # around
    'random_state': 42,
    'n_jobs': -1
}

    # Initialize and Train RF
    model = RandomForestRegressor(**params)
    model.fit(X_train_imp, y_train)

    # Predict on Validation
    preds = model.predict(X_val_imp)
    rmse = np.sqrt(mean_squared_error(y_val, preds))
    return rmse

# --- RUN OPTIMIZATION ---
study_name = f'{CHOSEN_CROP.capitalize()}_Yield_RF'
study = optuna.create_study(direction='minimize', study_name=study_name)
study.optimize(objective, n_trials=50) # Reduced trials for speed in demo, typic
```

```
print("\nBest Parameters found:")
print(study.best_params)
```

```
[I 2025-11-29 19:06:55,950] A new study created in memory with name: Rice_Yield_R
F
[I 2025-11-29 19:06:57,255] Trial 0 finished with value: 519.6806143430294 and pa
rameters: {'n_estimators': 269, 'max_depth': 8, 'min_samples_split': 3, 'min_samp
les_leaf': 8, 'max_features': 0.9924246445225184}. Best is trial 0 with value: 51
9.6806143430294.
[I 2025-11-29 19:06:58,626] Trial 1 finished with value: 520.8872154876525 and pa
rameters: {'n_estimators': 269, 'max_depth': 10, 'min_samples_split': 5, 'min_sam
ples_leaf': 9, 'max_features': 0.9600852931482907}. Best is trial 0 with value: 5
19.6806143430294.
[I 2025-11-29 19:07:00,033] Trial 2 finished with value: 520.6963795597195 and pa
rameters: {'n_estimators': 296, 'max_depth': 7, 'min_samples_split': 5, 'min_samp
les_leaf': 10, 'max_features': 0.9685818475454447}. Best is trial 0 with value: 5
19.6806143430294.
[I 2025-11-29 19:07:01,280] Trial 3 finished with value: 519.6645596089433 and pa
rameters: {'n_estimators': 284, 'max_depth': 7, 'min_samples_split': 5, 'min_samp
les_leaf': 9, 'max_features': 0.9700088512408372}. Best is trial 3 with value: 51
9.6645596089433.
[I 2025-11-29 19:07:02,592] Trial 4 finished with value: 521.1236981564417 and pa
rameters: {'n_estimators': 259, 'max_depth': 9, 'min_samples_split': 4, 'min_samp
les_leaf': 9, 'max_features': 0.9744955079821445}. Best is trial 3 with value: 51
9.6645596089433.
[I 2025-11-29 19:07:03,977] Trial 5 finished with value: 521.7230107823 and param
eters: {'n_estimators': 280, 'max_depth': 9, 'min_samples_split': 2, 'min_samples
_leaf': 9, 'max_features': 0.9037767257885072}. Best is trial 3 with value: 519.6
645596089433.
[I 2025-11-29 19:07:05,030] Trial 6 finished with value: 519.2365261697988 and pa
rameters: {'n_estimators': 278, 'max_depth': 6, 'min_samples_split': 5, 'min_samp
les_leaf': 9, 'max_features': 0.9982932011151715}. Best is trial 6 with value: 51
9.2365261697988.
[I 2025-11-29 19:07:06,404] Trial 7 finished with value: 521.6652231205343 and pa
rameters: {'n_estimators': 273, 'max_depth': 9, 'min_samples_split': 2, 'min_samp
les_leaf': 10, 'max_features': 0.9903531085533573}. Best is trial 6 with value: 5
19.2365261697988.
[I 2025-11-29 19:07:07,772] Trial 8 finished with value: 521.1886065437724 and pa
rameters: {'n_estimators': 270, 'max_depth': 7, 'min_samples_split': 2, 'min_samp
les_leaf': 8, 'max_features': 0.9625436703685786}. Best is trial 6 with value: 51
9.2365261697988.
[I 2025-11-29 19:07:09,071] Trial 9 finished with value: 521.7336239917017 and pa
rameters: {'n_estimators': 275, 'max_depth': 8, 'min_samples_split': 3, 'min_samp
les_leaf': 9, 'max_features': 0.9283005931069341}. Best is trial 6 with value: 51
9.2365261697988.
[I 2025-11-29 19:07:10,014] Trial 10 finished with value: 521.1072224682025 and p
arameters: {'n_estimators': 254, 'max_depth': 6, 'min_samples_split': 4, 'min_sam
ples_leaf': 10, 'max_features': 0.9383588174708136}. Best is trial 6 with value: 
519.2365261697988.
[I 2025-11-29 19:07:11,784] Trial 11 finished with value: 519.7469327465697 and p
arameters: {'n_estimators': 289, 'max_depth': 6, 'min_samples_split': 5, 'min_sam
ples_leaf': 8, 'max_features': 0.9991051875157454}. Best is trial 6 with value: 5
19.2365261697988.
[I 2025-11-29 19:07:13,193] Trial 12 finished with value: 519.7103746035506 and p
arameters: {'n_estimators': 285, 'max_depth': 7, 'min_samples_split': 4, 'min_sam
ples_leaf': 9, 'max_features': 0.9774992394348166}. Best is trial 6 with value: 5
19.2365261697988.
[I 2025-11-29 19:07:14,565] Trial 13 finished with value: 520.9317871416973 and p
arameters: {'n_estimators': 298, 'max_depth': 6, 'min_samples_split': 5, 'min_sam
ples_leaf': 9, 'max_features': 0.9489404801792206}. Best is trial 6 with value: 5
19.2365261697988.
[I 2025-11-29 19:07:15,884] Trial 14 finished with value: 520.7364519310353 and p
arameters: {'n_estimators': 288, 'max_depth': 7, 'min_samples_split': 4, 'min_sam
```

```
ples_leaf': 8, 'max_features': 0.9822901802900265}. Best is trial 6 with value: 5
19.2365261697988.
[I 2025-11-29 19:07:17,036] Trial 15 finished with value: 521.1008215129028 and p
arameters: {'n_estimators': 280, 'max_depth': 6, 'min_samples_split': 5, 'min_sam
ples_leaf': 9, 'max_features': 0.9510904117135193}. Best is trial 6 with value: 5
19.2365261697988.
[I 2025-11-29 19:07:18,226] Trial 16 finished with value: 521.0339731985835 and p
arameters: {'n_estimators': 263, 'max_depth': 7, 'min_samples_split': 5, 'min_sam
ples_leaf': 10, 'max_features': 0.9853196138341411}. Best is trial 6 with value:
519.2365261697988.
[I 2025-11-29 19:07:19,412] Trial 17 finished with value: 519.1248351857517 and p
arameters: {'n_estimators': 281, 'max_depth': 6, 'min_samples_split': 3, 'min_sam
ples_leaf': 9, 'max_features': 0.998864918876735}. Best is trial 17 with value: 5
19.1248351857517.
[I 2025-11-29 19:07:20,581] Trial 18 finished with value: 519.6533160430129 and p
arameters: {'n_estimators': 293, 'max_depth': 6, 'min_samples_split': 3, 'min_sam
ples_leaf': 8, 'max_features': 0.9938208184044844}. Best is trial 17 with value:
519.1248351857517.
[I 2025-11-29 19:07:21,877] Trial 19 finished with value: 520.6622815635274 and p
arameters: {'n_estimators': 280, 'max_depth': 8, 'min_samples_split': 3, 'min_sam
ples_leaf': 10, 'max_features': 0.9072292421049708}. Best is trial 17 with value:
519.1248351857517.
[I 2025-11-29 19:07:22,898] Trial 20 finished with value: 521.0921698484843 and p
arameters: {'n_estimators': 265, 'max_depth': 6, 'min_samples_split': 4, 'min_sam
ples_leaf': 9, 'max_features': 0.9253012547592727}. Best is trial 17 with value:
519.1248351857517.
[I 2025-11-29 19:07:24,088] Trial 21 finished with value: 519.6533160430129 and p
arameters: {'n_estimators': 293, 'max_depth': 6, 'min_samples_split': 3, 'min_sam
ples_leaf': 8, 'max_features': 0.9988909010980003}. Best is trial 17 with value:
519.1248351857517.
[I 2025-11-29 19:07:25,367] Trial 22 finished with value: 519.5920534187284 and p
arameters: {'n_estimators': 292, 'max_depth': 6, 'min_samples_split': 3, 'min_sam
ples_leaf': 8, 'max_features': 0.9915115198598523}. Best is trial 17 with value:
519.1248351857517.
[I 2025-11-29 19:07:26,486] Trial 23 finished with value: 520.07023958232 and par
ameters: {'n_estimators': 277, 'max_depth': 6, 'min_samples_split': 3, 'min_sampl
es_leaf': 8, 'max_features': 0.9844562728898345}. Best is trial 17 with value: 51
9.1248351857517.
[I 2025-11-29 19:07:27,944] Trial 24 finished with value: 520.7376731812025 and p
arameters: {'n_estimators': 287, 'max_depth': 7, 'min_samples_split': 2, 'min_sam
ples_leaf': 8, 'max_features': 0.9980112318334678}. Best is trial 17 with value:
519.1248351857517.
[I 2025-11-29 19:07:29,167] Trial 25 finished with value: 518.6411777666584 and p
arameters: {'n_estimators': 300, 'max_depth': 6, 'min_samples_split': 3, 'min_sam
ples_leaf': 9, 'max_features': 0.9866291328934667}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:30,513] Trial 26 finished with value: 519.4246654286562 and p
arameters: {'n_estimators': 300, 'max_depth': 7, 'min_samples_split': 4, 'min_sam
ples_leaf': 9, 'max_features': 0.9826144276304307}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:31,866] Trial 27 finished with value: 519.5621629707991 and p
arameters: {'n_estimators': 278, 'max_depth': 8, 'min_samples_split': 3, 'min_sam
ples_leaf': 9, 'max_features': 0.9762251590100861}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:33,367] Trial 28 finished with value: 520.6075177772476 and p
arameters: {'n_estimators': 283, 'max_depth': 10, 'min_samples_split': 2, 'min_sa
mples_leaf': 9, 'max_features': 0.9879416961646458}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:34,507] Trial 29 finished with value: 519.3359729408377 and p
arameters: {'n_estimators': 272, 'max_depth': 6, 'min_samples_split': 3, 'min_sam
```

```
ples_leaf': 9, 'max_features': 0.9928589294792313}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:35,585] Trial 30 finished with value: 519.6027642229028 and p
arameters: {'n_estimators': 266, 'max_depth': 6, 'min_samples_split': 4, 'min_sam
ples_leaf': 10, 'max_features': 0.9993798692662954}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:36,669] Trial 31 finished with value: 519.3359729408377 and p
arameters: {'n_estimators': 272, 'max_depth': 6, 'min_samples_split': 3, 'min_sam
ples_leaf': 9, 'max_features': 0.9914783300123241}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:37,734] Trial 32 finished with value: 519.3121633999626 and p
arameters: {'n_estimators': 273, 'max_depth': 6, 'min_samples_split': 3, 'min_sam
ples_leaf': 9, 'max_features': 0.9941863277987104}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:38,913] Trial 33 finished with value: 519.960117511264 and pa
rameters: {'n_estimators': 268, 'max_depth': 7, 'min_samples_split': 3, 'min_samp
les_leaf': 9, 'max_features': 0.9789496430654134}. Best is trial 25 with value: 5
18.6411777666584.
[I 2025-11-29 19:07:39,970] Trial 34 finished with value: 519.0197958678843 and p
arameters: {'n_estimators': 262, 'max_depth': 6, 'min_samples_split': 3, 'min_sam
ples_leaf': 9, 'max_features': 0.969863721548407}. Best is trial 25 with value: 5
18.6411777666584.
[I 2025-11-29 19:07:41,083] Trial 35 finished with value: 519.5549969635465 and p
arameters: {'n_estimators': 251, 'max_depth': 7, 'min_samples_split': 3, 'min_sam
ples_leaf': 9, 'max_features': 0.9681008481958856}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:42,143] Trial 36 finished with value: 518.9770016070737 and p
arameters: {'n_estimators': 260, 'max_depth': 6, 'min_samples_split': 2, 'min_sam
ples_leaf': 9, 'max_features': 0.9701059394460888}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:43,525] Trial 37 finished with value: 520.2847503252173 and p
arameters: {'n_estimators': 258, 'max_depth': 10, 'min_samples_split': 2, 'min_sa
mples_leaf': 9, 'max_features': 0.9595429920014357}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:44,586] Trial 38 finished with value: 518.9941192994653 and p
arameters: {'n_estimators': 258, 'max_depth': 6, 'min_samples_split': 2, 'min_sam
ples_leaf': 9, 'max_features': 0.9650012072565363}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:45,885] Trial 39 finished with value: 521.2419806843194 and p
arameters: {'n_estimators': 261, 'max_depth': 9, 'min_samples_split': 2, 'min_sam
ples_leaf': 9, 'max_features': 0.9674624594088379}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:47,095] Trial 40 finished with value: 519.7294666879909 and p
arameters: {'n_estimators': 256, 'max_depth': 7, 'min_samples_split': 2, 'min_sam
ples_leaf': 9, 'max_features': 0.9545755098798092}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:48,467] Trial 41 finished with value: 519.0330201409396 and p
arameters: {'n_estimators': 261, 'max_depth': 6, 'min_samples_split': 2, 'min_sam
ples_leaf': 9, 'max_features': 0.9730691895517865}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:49,589] Trial 42 finished with value: 518.684255954307 and pa
rameters: {'n_estimators': 250, 'max_depth': 6, 'min_samples_split': 2, 'min_samp
les_leaf': 9, 'max_features': 0.9709974194480028}. Best is trial 25 with value: 5
18.6411777666584.
[I 2025-11-29 19:07:50,741] Trial 43 finished with value: 518.6842559543069 and p
arameters: {'n_estimators': 250, 'max_depth': 6, 'min_samples_split': 2, 'min_sam
ples_leaf': 9, 'max_features': 0.9642935459748782}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:51,781] Trial 44 finished with value: 518.6842559543069 and p
arameters: {'n_estimators': 250, 'max_depth': 6, 'min_samples_split': 2, 'min_sam
```

```
ples_leaf': 9, 'max_features': 0.9639478421437355}. Best is trial 25 with value:
518.6411777666584.
[I 2025-11-29 19:07:53,075] Trial 45 finished with value: 518.4945125066965 and p
arameters: {'n_estimators': 250, 'max_depth': 8, 'min_samples_split': 2, 'min_sam
ples_leaf': 9, 'max_features': 0.9565464415634675}. Best is trial 45 with value:
518.4945125066965.
[I 2025-11-29 19:07:54,500] Trial 46 finished with value: 520.7304453287131 and p
arameters: {'n_estimators': 250, 'max_depth': 9, 'min_samples_split': 2, 'min_sam
ples_leaf': 9, 'max_features': 0.9584682294624007}. Best is trial 45 with value:
518.4945125066965.
[I 2025-11-29 19:07:55,685] Trial 47 finished with value: 521.270528844438 and pa
rameters: {'n_estimators': 253, 'max_depth': 8, 'min_samples_split': 2, 'min_samp
les_leaf': 9, 'max_features': 0.9457571639772834}. Best is trial 45 with value: 5
18.4945125066965.
[I 2025-11-29 19:07:56,891] Trial 48 finished with value: 520.0719932737169 and p
arameters: {'n_estimators': 253, 'max_depth': 8, 'min_samples_split': 2, 'min_sam
ples_leaf': 10, 'max_features': 0.942560401988229}. Best is trial 45 with value:
518.4945125066965.
[I 2025-11-29 19:07:58,009] Trial 49 finished with value: 519.5775455394543 and p
arameters: {'n_estimators': 255, 'max_depth': 7, 'min_samples_split': 2, 'min_sam
ples_leaf': 9, 'max_features': 0.9544156882245551}. Best is trial 45 with value:
518.4945125066965.
Best Parameters found:
{'n_estimators': 250, 'max_depth': 8, 'min_samples_split': 2, 'min_samples_leaf':
9, 'max_features': 0.9565464415634675}
```

6. Visualizing Optimization

We generate charts to understand the tuning process. These visual tools show us which specific settings had the biggest impact on reducing the model's error and how the optimization improved over time.

In [8]:

```python
# --- OPTUNA VISUALIZATIONS ---
name = f"{CHOSEN_CROP.capitalize()}_Yield_RF_Model"

# 1. Optimization History
fig = plot_optimization_history(study)
fig.update_layout(title=f'{name} Optimization History', width=900, height=500)
fig.show()

# 2. Parallel Coordinate (Hyperparameter Relationships)
fig = plot_parallel_coordinate(study)
fig.update_layout(title=f'{name} Parallel Coordinate Plot', width=900, height=50
fig.show()

# 3. Slice Plot (Individual Parameter impact)
fig = plot_slice(study)
fig.update_layout(title=f'{name} Slice Plot', width=900, height=500)
fig.show()

# 4. Parameter Importance
try:
    fig = plot_param_importances(study)
    fig.update_layout(title=f'{name} Hyperparameter Importance', width=900, heig
    fig.show()
except (ValueError, RuntimeError) as e:
    print(f'Could not plot parameter importance: {e}')
```

7. Final Model Training

Using the best settings found during the tuning phase, we build the final model. We train this model on both the Training and Validation data combined to maximize the information available for learning.

In [9]:
```
# 1. Combine Train + Validation for Final Training
X_train_full = pd.concat([X_train, X_val])
y_train_full = pd.concat([y_train, y_val])

# Fit Imputer on full training data
imputer_final = SimpleImputer(strategy='mean')
X_train_full_imp = imputer_final.fit_transform(X_train_full)
X_test_final_imp = imputer_final.transform(X_test)

# 2. Initialize with Best Params
best_params = study.best_params
best_params['random_state'] = 42
best_params['n_jobs'] = -1

final_model = RandomForestRegressor(**best_params)

# 3. Train on Full History
final_model.fit(X_train_full_imp, y_train_full)

# 4. Final Prediction on TEST Data
y_pred_final_test = final_model.predict(X_test_final_imp)
rmse_final_test = np.sqrt(mean_squared_error(y_test, y_pred_final_test))
r2_final_test = r2_score(y_test, y_pred_final_test)
```

8. Results and Analysis

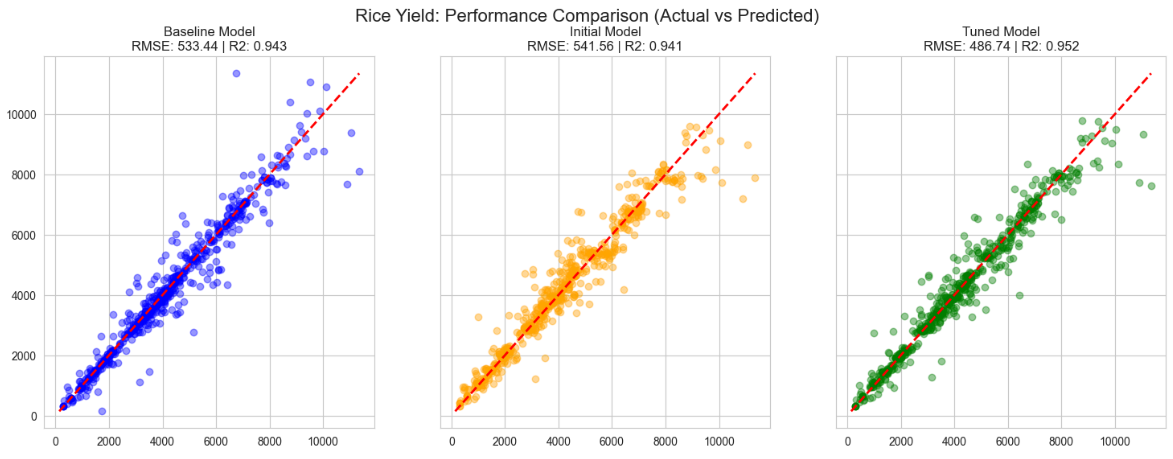
We evaluate the final performance on the Test data (2019–2023).

- **Comparison:** We check if the Tuned Model beats the Baseline and the Initial Model.

In [10]:
```
# Calculate Improvement %
imp_final = (rmse_baseline - rmse_final_test) / rmse_baseline * 100

print("--- Final Performance Report (Test Set) ---")
print(f"Baseline Model: RMSE={rmse_baseline:.2f}, R2={r2_baseline:.4f}")
print(f"Initial Model:  RMSE={rmse_init_test:.2f}, R2={r2_init_test:.4f}")
print(f"Tuned Model:    RMSE={rmse_final_test:.2f}, R2={r2_final_test:.4f} (RMSE

# --- PLOTTING RESULTS ---
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)

# Axis Limits
all_preds = np.concatenate([y_pred_clean, y_pred_init_test, y_pred_final_test])
all_true = np.concatenate([y_test_clean, y_test, y_test])
min_val, max_val = min(min(all_preds), min(all_true)), max(max(all_preds), max(a

# 1. Baseline Plot
axes[0].scatter(y_test_clean, y_pred_clean, alpha=0.4, color='blue')
axes[0].plot([min_val, max_val], [min_val, max_val], 'r--', linewidth=2)
```

```
axes[0].set_title(f'Baseline Model\nRMSE: {rmse_baseline:.2f} | R2: {r2_baseline

# 2. Initial Model Plot
axes[1].scatter(y_test, y_pred_init_test, alpha=0.4, color='orange')
axes[1].plot([min_val, max_val], [min_val, max_val], 'r--', linewidth=2)
axes[1].set_title(f'Initial Model\nRMSE: {rmse_init_test:.2f} | R2: {r2_init_tes

# 3. Tuned Model Plot
axes[2].scatter(y_test, y_pred_final_test, alpha=0.4, color='green')
axes[2].plot([min_val, max_val], [min_val, max_val], 'r--', linewidth=2)
axes[2].set_title(f'Tuned Model\nRMSE: {rmse_final_test:.2f} | R2: {r2_final_tes

plt.suptitle(f'{CHOSEN_CROP.capitalize()} Yield: Performance Comparison (Actual
plt.show()
```

```
--- Final Performance Report (Test Set) ---
Baseline Model: RMSE=533.44, R2=0.9427
Initial Model:  RMSE=541.56, R2=0.9410
Tuned Model:    RMSE=486.74, R2=0.9523 (RMSE Improved 8.75%)
```

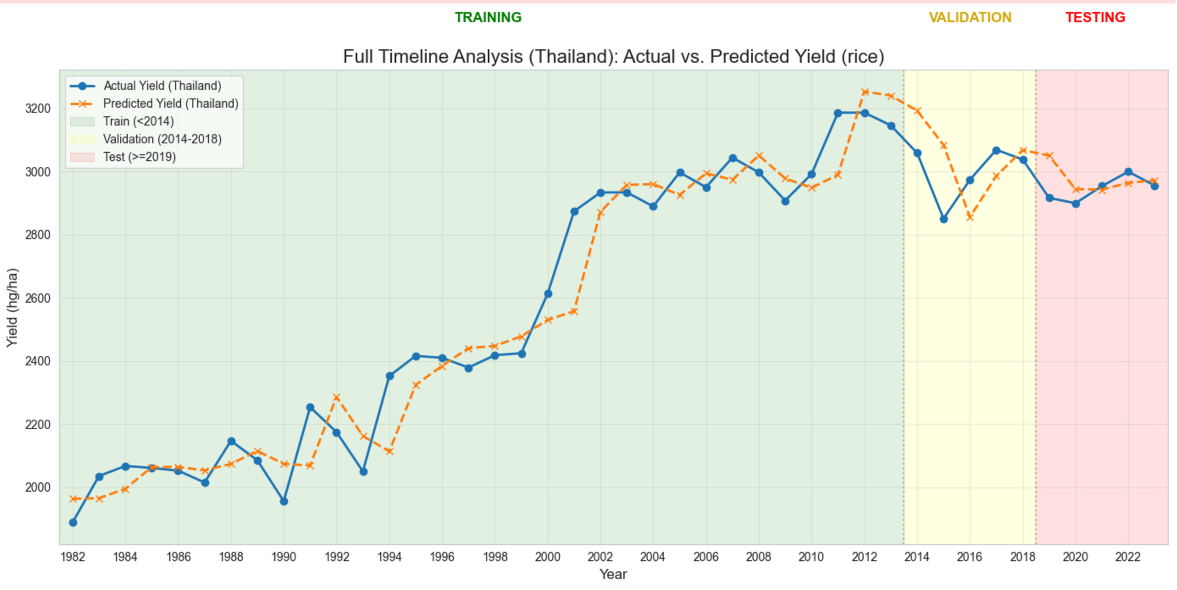


- **Trend Analysis:** We plot the predicted yields against actual yields over time to visualize accuracy.

In [16]:
```
import matplotlib.pyplot as plt

# Ensure these match your previous variables
TRAIN_END_YEAR = 2014
VAL_END_YEAR = 2019

# Country parameter
TARGET_COUNTRY = "Thailand"

# 1. Generate Predictions
all_predictions = final_model.predict(df_model[feature_cols])

# 2. Create DataFrame WITH AREA column
df_full_trend = pd.DataFrame({
    'Year': df_model['year'],
    'Area': df_model['area'],       # added so we can filter by country
    'Actual': df_model[TARGET_COL],
    'Predicted': all_predictions
})

# ❗ 3. Filter for Thailand only
df_th = df_full_trend[df_full_trend['Area'] == TARGET_COUNTRY]
```

```python
# Aggregate by Year
yearly_trend = df_th.groupby('Year')[['Actual', 'Predicted']].mean()

# 3. Plotting
plt.figure(figsize=(14, 7))

# Plot Lines
plt.plot(yearly_trend.index, yearly_trend['Actual'],
         marker='o', label=f'Actual Yield ({TARGET_COUNTRY})', linewidth=2)
plt.plot(yearly_trend.index, yearly_trend['Predicted'],
         marker='x', linestyle='--', label=f'Predicted Yield ({TARGET_COUNTRY})'

# Define Split Boundaries
MIN_YEAR = yearly_trend.index.min()
MAX_YEAR = yearly_trend.index.max()

# CALCULATE OFFSETS
train_boundary = TRAIN_END_YEAR - 0.5
val_boundary   = VAL_END_YEAR - 0.5

# Highlight Periods
plt.axvspan(MIN_YEAR - 0.5, train_boundary, color='green', alpha=0.1,
            label=f'Train (<{TRAIN_END_YEAR})')
plt.axvspan(train_boundary, val_boundary, color='yellow', alpha=0.1,
            label=f'Validation ({TRAIN_END_YEAR}-{VAL_END_YEAR - 1})')
plt.axvspan(val_boundary, MAX_YEAR + 0.5, color='red', alpha=0.1,
            label=f'Test (>={VAL_END_YEAR})')

# Add Vertical Lines
plt.axvline(train_boundary, color='grey', linestyle=':', alpha=0.5)
plt.axvline(val_boundary, color='grey', linestyle=':', alpha=0.5)

# Add Text Labels
y_max = yearly_trend['Actual'].max()
text_y = y_max * 1.09

plt.text((MIN_YEAR + train_boundary)/2, text_y, 'TRAINING',
         ha='center', fontsize=12, fontweight='bold', color='green')
plt.text((train_boundary + val_boundary)/2, text_y, 'VALIDATION',
         ha='center', fontsize=12, fontweight='bold', color='#D4AC0D')
plt.text((val_boundary + MAX_YEAR)/2, text_y, 'TESTING',
         ha='center', fontsize=12, fontweight='bold', color='red')

# Final Formatting
plt.title(f'Full Timeline Analysis ({TARGET_COUNTRY}): Actual vs. Predicted Yiel
          fontsize=16)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Yield (hg/ha)', fontsize=12)
plt.legend(loc='upper left')
plt.grid(True, alpha=0.3)

# Force x-axis to show integer years
plt.xticks(np.arange(MIN_YEAR, MAX_YEAR + 1, 2))
plt.xlim(MIN_YEAR - 0.5, MAX_YEAR + 0.5)

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

```
C:\Users\PavinP\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n
2kfra8p0\LocalCache\local-packages\Python312\site-packages\sklearn\utils\validati
on.py:2742: UserWarning:

X has feature names, but RandomForestRegressor was fitted without feature names
```



- **Geographic Error:** We map the error rates by country to see where the model performs best and where it struggles.

In [13]:
```python
# --- REFRESHING THE TEST DATA CONTEXT ---
VAL_END_YEAR = 2019
mask_test = df_model['year'] >= VAL_END_YEAR

# We need the original 'area' column from the test set for joining
test_set_context = df_model[mask_test][['area', 'year']]

# --- RE-CREATE COMPARISON DF WITH FEATURE JOINED ---

# 1. Create the Comparison DataFrame
comparison_df = pd.DataFrame({
    'Actual_Value': y_test,
    'Predicted_Value': y_pred_final_test
})

# 2. Join the desired feature column(s)
comparison_df = comparison_df.join(test_set_context)

# 3. Reorder the columns for better readability
comparison_df = comparison_df[['year', 'area', 'Actual_Value', 'Predicted_Value'

# 4. Display the header
print("--- Actual vs. Predicted Test Set Results with Feature Context ---")
print(comparison_df.head())
```

```
--- Actual vs. Predicted Test Set Results with Feature Context ---
    year         area  Actual_Value  Predicted_Value
37  2019  Afghanistan        4476.6      4664.162613
38  2020  Afghanistan        4441.7      4647.084426
39  2021  Afghanistan        4406.5      4632.266283
40  2022  Afghanistan        4625.0      4545.905574
41  2023  Afghanistan        4627.9      4632.900235
```

In [14]:
```python
import plotly.express as px

# Assuming comparison_df is defined and country names are cleaned up here...
comparison_df['area'] = comparison_df['area'].replace({
    'United_States_of_America': 'United States',
    'United_Kingdom_of_Great_Britain_and_Northern_Ireland': 'United Kingdom',
    'Russian_Federation': 'Russia',
    'Viet_Nam': 'Vietnam',
    'Türkiye': 'Turkey',
    'Bolivia_(Plurinational_State_of)': 'Bolivia',
    'Iran_(Islamic_Republic_of)': 'Iran',
    "Lao_People's_Democratic_Republic": 'Laos',
    'China,_mainland': 'China',
    'China,_Taiwan_Province_of': 'Taiwan',
    "Democratic_People's_Republic_of_Korea": 'North Korea',
    'Republic_of_Korea': 'South Korea',
    'Côte_d\'Ivoire': "Cote d'Ivoire",
    'United_Republic_of_Tanzania': 'Tanzania',
    'Micronesia_(Federated_States_of)': 'Micronesia',
    'Venezuela_(Bolivarian_Republic_of)': 'Venezuela'
})

def plot_geographic_error(comparison_df):
    """
    Aggregates prediction RMSPE and RMSE by country and plots the distribution
    on a world map using Plotly Express, with both metrics in the hover data.
    """

    # --- 1. Compute Errors ---
    comparison_df['Squared_Error'] = (
        comparison_df['Actual_Value'] - comparison_df['Predicted_Value']
    ) ** 2

    epsilon = 1e-6
    comparison_df['Squared_Percentage_Error'] = (
        (comparison_df['Actual_Value'] - comparison_df['Predicted_Value']) /
        (comparison_df['Actual_Value'] + epsilon)
    ) ** 2

    # --- 2. Aggregate Errors by Country ---
    rmse_df = (
        comparison_df.groupby('area')['Squared_Error']
        .mean()
        .apply(np.sqrt)
        .reset_index()
        .rename(columns={'area': 'Country', 'Squared_Error': 'RMSE'})
    )

    rmspe_df = (
        comparison_df.groupby('area')['Squared_Percentage_Error']
        .mean()
        .apply(np.sqrt)
        .multiply(100)
        .reset_index()
        .rename(columns={'area': 'Country', 'Squared_Percentage_Error': 'RMSPE'}
    )

    # 3. Compute mean Actual & Predicted per Country
    ap_df = comparison_df.groupby('area')[['Actual_Value', 'Predicted_Value']].m
    ap_df = ap_df.rename(columns={'area': 'Country'})

    # 4. Merge RMSPE, RMSE, Actual, and Predicted
    error_stats = rmspe_df.merge(rmse_df, on='Country', how='left')
    error_stats = error_stats.merge(ap_df, on='Country', how='left')

    # 5. Choropleth Map
    fig = px.choropleth(
        error_stats,
        locations='Country',
        color='RMSPE',
        locationmode='country names',
        color_continuous_scale=['green', 'red'],
        range_color=[0, 50],
        title='Geographic Distribution of Prediction Error (RMSPE)',
        labels={'RMSPE': 'RMSPE (%)'},
        hover_name='Country',
        hover_data={
            'RMSPE': ':.2f',
            'RMSE': ':.2f',
            'Actual_Value': ':.2f',
            'Predicted_Value': ':.2f',
        },
        projection='natural earth'
    )

    # 6. Layout adjustments
    fig.update_layout(
        title_font_size=18,
        coloraxis_colorbar=dict(
            title='RMSPE (%)',
            orientation='h',
            len=0.5,
            yanchor='bottom',
            y=-0.12
        ),
        geo=dict(
            showframe=False,
            showcoastlines=True,
            showcountries=True,
            countrycolor='black',
            bgcolor='lightgrey'
        )
    )

    fig.show()

# Run visualization
plot_geographic_error(comparison_df)
```

```
C:\Users\PavinP\AppData\Local\Temp\ipykernel_3136\2231010933.py:67: DeprecationWa
rning:

The library used by the *country names* `locationmode` option is changing in an u
pcoming version. Country names in existing plots may not work in the new version.
To ensure consistent behavior, consider setting `locationmode` to *ISO-3*.
```

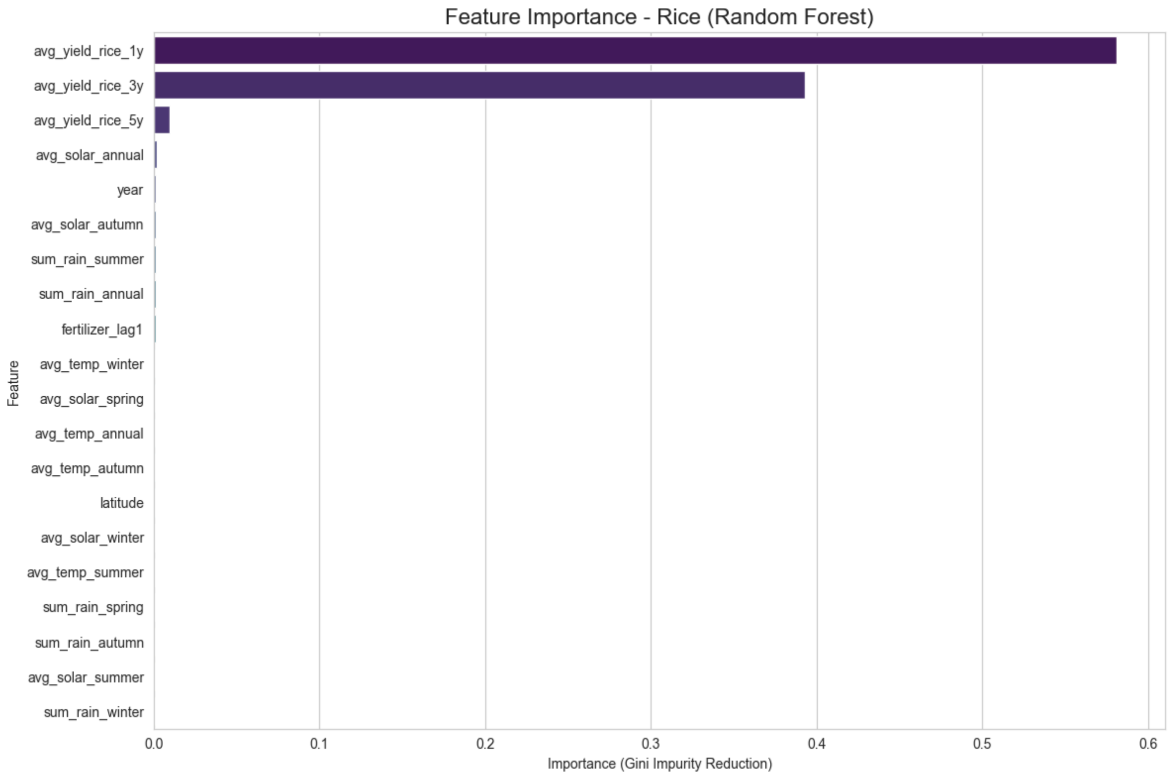
9. Key Factors (Feature Importance)

Finally, we analyze what drives the predictions. We list the top 20 features that influence rice yield. Typically, the most important factors are previous yields (history) and specific climate metrics like temperature and solar radiation.

In [15]:
```
# --- FEATURE IMPORTANCE: PLOT & TEXT ---

# 1. Extract feature importances from Random Forest
importances = final_model.feature_importances_
feature_names = feature_cols

# 2. Create a DataFrame to display as text
fi_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
})

# 3. Sort by importance
fi_df = fi_df.sort_values(by='Importance', ascending=False).reset_index(drop=Tru

# 4. PRINT TEXT: Display the Top 20 features
print("\n--- Top 20 Most Important Features (Text Report) ---")
print(fi_df.head(20))

# 5. PLOT GRAPH: Use Seaborn barplot since RF doesn't use lgb.plot_importance
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=fi_df.head(20), palette='viridis')
plt.title(f'Feature Importance - {CHOSEN_CROP.capitalize()} (Random Forest)', fc
plt.xlabel('Importance (Gini Impurity Reduction)')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```

```
--- Top 20 Most Important Features (Text Report) ---
              Feature  Importance
0    avg_yield_rice_1y    0.581296
1    avg_yield_rice_3y    0.393150
2    avg_yield_rice_5y    0.009440
3     avg_solar_annual    0.001865
4                 year    0.001183
5     avg_solar_autumn    0.001019
6      sum_rain_summer    0.000992
7      sum_rain_annual    0.000969
8      fertilizer_lag1    0.000882
9      avg_temp_winter    0.000879
10    avg_solar_spring    0.000812
11     avg_temp_annual    0.000763
12     avg_temp_autumn    0.000720
13            latitude    0.000673
14    avg_solar_winter    0.000662
15     avg_temp_summer    0.000653
16     sum_rain_spring    0.000637
17     sum_rain_autumn    0.000603
18    avg_solar_summer    0.000583
19     sum_rain_winter    0.000573
```

```
C:\Users\PavinP\AppData\Local\Temp\ipykernel_3136\449948700.py:22: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effe
ct.
```


Feature Importance - Rice (Random Forest)
avg_yield_rice_1y
avg_yield_rice_3y
avg_yield_rice_5y
avg_solar_annual
year
avg_solar_autumn
sum_rain_summer
sum_rain_annual
fertilizer_lag1
avg_temp_winter
avg_solar_spring
avg_temp_annual
avg_temp_autumn
latitude
avg_solar_winter
avg_temp_summer
sum_rain_spring
sum_rain_autumn
avg_solar_summer
sum_rain_winter
Feature
Importance (Gini Impurity Reduction)