Crop Yield Prediction: XGBoost with Optuna (Part 5)

Overview

This notebook trains an **XGBoost (Extreme Gradient Boosting)** 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]:
```python
import pandas as pd
import numpy as np
import xgboost as xgb
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_ra
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}")
```

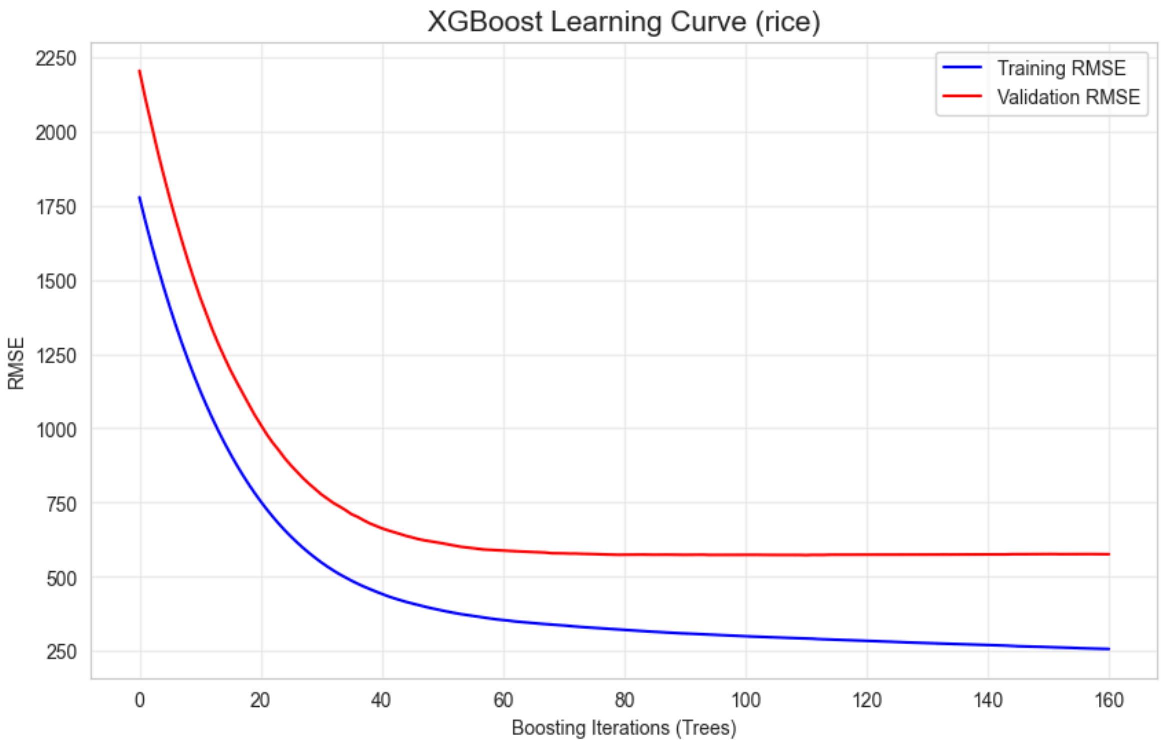
```
Baseline RMSE: 533.44
```

4. Initial Model Testing

We train a basic XGBoost model using default settings. We plot a learning curve to ensure the model is learning patterns rather than just memorizing data (overfitting). We then calculate the initial accuracy on the Test set to see how it compares to the baseline.

In [6]:
```python
# --- INITIAL MODEL TRAINING ---

# 1. Initialize XGBoost
model_init = xgb.XGBRegressor(
    n_estimators=1000,
    learning_rate=0.05,
    max_depth=6,
    random_state=42,
    n_jobs=-1,
    early_stopping_rounds=50
)

# 2. Train on TRAIN, Early Stop on VALIDATION
model_init.fit(
    X_train, y_train,
    eval_set=[(X_train, y_train), (X_val, y_val)],
    verbose=100
)

# Retrieve training metrics
results = model_init.evals_result()

# --- PLOT LEARNING CURVE (RMSE over Iterations) ---
def plot_learning_curve(results, metric='rmse'):
    plt.figure(figsize=(10, 6))

    # XGBoost stores results under 'validation_0', 'validation_1' by default whe
    train_metric = results['validation_0'][metric]
    val_metric = results['validation_1'][metric]

    plt.plot(train_metric, label='Training RMSE', color='blue')
    plt.plot(val_metric, label='Validation RMSE', color='red')

    plt.title(f'XGBoost Learning Curve ({CHOSEN_CROP})', fontsize=15)
    plt.xlabel('Boosting Iterations (Trees)')
    plt.ylabel('RMSE')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()

plot_learning_curve(results)

# Evaluate on TEST Set
y_pred_init_test = model_init.predict(X_test)
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}")
```

```
[0]	validation_0-rmse:1777.56755	validation_1-rmse:2203.36484
[100]	validation_0-rmse:300.50648	validation_1-rmse:574.82557
[160]	validation_0-rmse:257.61792	validation_1-rmse:576.52844
```



```
Initial Model Test RMSE: 604.18
```

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 learning rate and tree depth, to minimize the error on the validation data.

In [7]:
```python
# --- OPTUNA OBJECTIVE FUNCTION ---
def objective(trial):
    params = {
        'n_estimators': 1000,
        'learning_rate': trial.suggest_float('learning_rate', 0.005, 0.1),
        'max_depth': trial.suggest_int('max_depth', 3, 12),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 10),
        'subsample': trial.suggest_float('subsample', 0.6, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.3, 1.0),
        'reg_alpha': trial.suggest_float('reg_alpha', 1e-8, 10.0, log=True),
        'reg_lambda': trial.suggest_float('reg_lambda', 1e-8, 10.0, log=True),
        'random_state': 42,
        'n_jobs': -1,
        'early_stopping_rounds': 30
    }

    model = xgb.XGBRegressor(**params)

    model.fit(
        X_train, y_train,
        eval_set=[(X_val, y_val)],
        verbose=False
    )

    # XGBoost handles early stopping by returning the best iteration automatical
    # if early_stopping_rounds is set
    preds = model.predict(X_val)
    rmse = np.sqrt(mean_squared_error(y_val, preds))
    return rmse

# --- RUN OPTIMIZATION ---
study_name = f'{CHOSEN_CROP.capitalize()}_Yield_XGBoost'
study = optuna.create_study(direction='minimize', study_name=study_name)
study.optimize(objective, n_trials=50)

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

```
[I 2025-11-29 18:52:55,116] A new study created in memory with name: Rice_Yield_X
GBoost
[I 2025-11-29 18:52:55,707] Trial 0 finished with value: 555.4906156785869 and pa
rameters: {'learning_rate': 0.047945425451283066, 'max_depth': 9, 'min_child_weig
ht': 7, 'subsample': 0.9741454841767025, 'colsample_bytree': 0.7669904090705824,
'reg_alpha': 5.827819501852322e-05, 'reg_lambda': 9.88851294353984e-06}. Best is
trial 0 with value: 555.4906156785869.
[I 2025-11-29 18:52:55,918] Trial 1 finished with value: 505.6539218878878 and pa
rameters: {'learning_rate': 0.05650578346446251, 'max_depth': 3, 'min_child_weigh
t': 3, 'subsample': 0.655416731326327, 'colsample_bytree': 0.8593304937993997, 'r
eg_alpha': 0.0021572523047115135, 'reg_lambda': 0.0710101528619008}. Best is tria
l 1 with value: 505.6539218878878.
[I 2025-11-29 18:52:56,236] Trial 2 finished with value: 538.2623600445506 and pa
rameters: {'learning_rate': 0.07037707503450213, 'max_depth': 6, 'min_child_weigh
t': 7, 'subsample': 0.8405244329978423, 'colsample_bytree': 0.6330694144266522,
'reg_alpha': 1.8535237492388377, 'reg_lambda': 5.581326696580131e-06}. Best is tr
ial 1 with value: 505.6539218878878.
[I 2025-11-29 18:52:57,024] Trial 3 finished with value: 610.5645715771371 and pa
rameters: {'learning_rate': 0.07876791761401152, 'max_depth': 10, 'min_child_weig
ht': 1, 'subsample': 0.6416867460071787, 'colsample_bytree': 0.48646664137879925,
'reg_alpha': 4.750110309478105e-07, 'reg_lambda': 8.224823571242444e-05}. Best is
trial 1 with value: 505.6539218878878.
[I 2025-11-29 18:52:57,172] Trial 4 finished with value: 508.42731773025525 and p
arameters: {'learning_rate': 0.07771849739371166, 'max_depth': 4, 'min_child_weig
ht': 8, 'subsample': 0.8451322790397982, 'colsample_bytree': 0.6636548912647966,
'reg_alpha': 0.00047635110910977804, 'reg_lambda': 0.000494021866105343}. Best is
trial 1 with value: 505.6539218878878.
[I 2025-11-29 18:52:57,430] Trial 5 finished with value: 569.1624548841631 and pa
rameters: {'learning_rate': 0.08504010469235825, 'max_depth': 8, 'min_child_weigh
t': 3, 'subsample': 0.622723539032055, 'colsample_bytree': 0.41003565820682464,
'reg_alpha': 0.5349158141818392, 'reg_lambda': 0.040268753309101686}. Best is tri
al 1 with value: 505.6539218878878.
[I 2025-11-29 18:52:57,998] Trial 6 finished with value: 557.812600200591 and par
ameters: {'learning_rate': 0.08339894339129711, 'max_depth': 12, 'min_child_weigh
t': 4, 'subsample': 0.6093989807361325, 'colsample_bytree': 0.9796841711255333,
'reg_alpha': 3.087327987520137e-06, 'reg_lambda': 3.436771631803764e-07}. Best is
trial 1 with value: 505.6539218878878.
[I 2025-11-29 18:52:58,562] Trial 7 finished with value: 534.2556262895362 and pa
rameters: {'learning_rate': 0.017544275712340584, 'max_depth': 6, 'min_child_weig
ht': 1, 'subsample': 0.7133959930092948, 'colsample_bytree': 0.9278023189230387,
'reg_alpha': 0.5710509203006463, 'reg_lambda': 8.973748672601828e-05}. Best is tr
ial 1 with value: 505.6539218878878.
[I 2025-11-29 18:52:58,828] Trial 8 finished with value: 549.3753026718871 and pa
rameters: {'learning_rate': 0.054093753632198906, 'max_depth': 9, 'min_child_weig
ht': 8, 'subsample': 0.6033048785110645, 'colsample_bytree': 0.5580626922633529,
'reg_alpha': 0.02391945807730839, 'reg_lambda': 0.005480888995717982}. Best is tr
ial 1 with value: 505.6539218878878.
[I 2025-11-29 18:52:59,557] Trial 9 finished with value: 559.136057884451 and par
ameters: {'learning_rate': 0.05856733580076202, 'max_depth': 10, 'min_child_weigh
t': 4, 'subsample': 0.9185632284390293, 'colsample_bytree': 0.8658957915313357,
'reg_alpha': 0.0011433832905696525, 'reg_lambda': 0.4246791489318131}. Best is tr
ial 1 with value: 505.6539218878878.
[I 2025-11-29 18:52:59,706] Trial 10 finished with value: 510.8912410635318 and p
arameters: {'learning_rate': 0.03186863542821216, 'max_depth': 3, 'min_child_wei
ght': 10, 'subsample': 0.7423163911012093, 'colsample_bytree': 0.796028856389662
7, 'reg_alpha': 2.1620573489648374e-08, 'reg_lambda': 5.086653052417473}. Best is
trial 1 with value: 505.6539218878878.
[I 2025-11-29 18:52:59,788] Trial 11 finished with value: 509.7289858491848 and p
arameters: {'learning_rate': 0.09904717384383849, 'max_depth': 3, 'min_child_weig
ht': 10, 'subsample': 0.83112616113685, 'colsample_bytree': 0.30941883848920776,
'reg_alpha': 0.0012709674804275706, 'reg_lambda': 0.00402796718724073}. Best is t
rial 1 with value: 505.6539218878878.
[I 2025-11-29 18:52:59,950] Trial 12 finished with value: 521.4174844562729 and p
arameters: {'learning_rate': 0.03985184178964225, 'max_depth': 5, 'min_child_weig
ht': 5, 'subsample': 0.7551658482033431, 'colsample_bytree': 0.7043218540621763,
'reg_alpha': 0.013258581691545975, 'reg_lambda': 1.7521820094521887e-08}. Best is
trial 1 with value: 505.6539218878878.
[I 2025-11-29 18:53:00,080] Trial 13 finished with value: 511.7637792955483 and p
arameters: {'learning_rate': 0.0648267401889288, 'max_depth': 4, 'min_child_weigh
t': 8, 'subsample': 0.882987719495245, 'colsample_bytree': 0.6358874082221118, 'r
eg_alpha': 4.8590547327993075e-05, 'reg_lambda': 0.0036482039850568044}. Best is
trial 1 with value: 505.6539218878878.
[I 2025-11-29 18:53:00,161] Trial 14 finished with value: 522.3489254106697 and p
arameters: {'learning_rate': 0.09732566032051988, 'max_depth': 5, 'min_child_weig
ht': 3, 'subsample': 0.6912020445022571, 'colsample_bytree': 0.841380616108597,
'reg_alpha': 0.030020607959160197, 'reg_lambda': 0.28103638188874136}. Best is tr
ial 1 with value: 505.6539218878878.
[I 2025-11-29 18:53:00,545] Trial 15 finished with value: 506.48380024963626 and
parameters: {'learning_rate': 0.008531815229038953, 'max_depth': 3, 'min_child_we
ight': 7, 'subsample': 0.789607912461977, 'colsample_bytree': 0.7228242983339818,
'reg_alpha': 8.692066837833956e-05, 'reg_lambda': 0.0005316951329987938}. Best is
trial 1 with value: 505.6539218878878.
[I 2025-11-29 18:53:00,784] Trial 16 finished with value: 511.42190084213456 and
parameters: {'learning_rate': 0.017945673933923347, 'max_depth': 3, 'min_child_we
ight': 6, 'subsample': 0.780274848487107, 'colsample_bytree': 0.7436565480320178,
'reg_alpha': 7.450956832838383e-06, 'reg_lambda': 7.160760368970008}. Best is tri
al 1 with value: 505.6539218878878.
[I 2025-11-29 18:53:02,111] Trial 17 finished with value: 548.2933922986078 and p
arameters: {'learning_rate': 0.006741578784114945, 'max_depth': 7, 'min_child_wei
ght': 2, 'subsample': 0.6857985421314966, 'colsample_bytree': 0.9067227166984023,
'reg_alpha': 2.1776026807501044e-07, 'reg_lambda': 0.07927700023942395}. Best is
trial 1 with value: 505.6539218878878.
[I 2025-11-29 18:53:02,294] Trial 18 finished with value: 518.0480984256745 and p
arameters: {'learning_rate': 0.03210596190734619, 'max_depth': 4, 'min_child_weig
ht': 5, 'subsample': 0.9908657331404347, 'colsample_bytree': 0.8099699480504376,
'reg_alpha': 4.786513515122548e-05, 'reg_lambda': 0.0016187097142532781}. Best is
trial 1 with value: 505.6539218878878.
[I 2025-11-29 18:53:02,435] Trial 19 finished with value: 522.2772046141805 and p
arameters: {'learning_rate': 0.04314100039495764, 'max_depth': 5, 'min_child_weig
ht': 6, 'subsample': 0.660863519497297, 'colsample_bytree': 0.5666356825771234,
'reg_alpha': 0.003860985819689891, 'reg_lambda': 0.02819758924564236}. Best is tr
ial 1 with value: 505.6539218878878.
[I 2025-11-29 18:53:03,648] Trial 20 finished with value: 552.6860035460115 and p
arameters: {'learning_rate': 0.0077801111034989144, 'max_depth': 7, 'min_child_we
ight': 3, 'subsample': 0.9238647246288865, 'colsample_bytree': 0.941943111172094
6, 'reg_alpha': 0.16032287154805605, 'reg_lambda': 7.383749210873957e-06}. Best i
s trial 1 with value: 505.6539218878878.
[I 2025-11-29 18:53:03,747] Trial 21 finished with value: 505.03936495652033 and
parameters: {'learning_rate': 0.07316284892427405, 'max_depth': 4, 'min_child_wei
ght': 9, 'subsample': 0.8164103913737544, 'colsample_bytree': 0.7138641605802839,
'reg_alpha': 0.00037398831834924715, 'reg_lambda': 0.00030539539502360294}. Best
is trial 21 with value: 505.03936495652033.
[I 2025-11-29 18:53:03,828] Trial 22 finished with value: 499.74224552133353 and
parameters: {'learning_rate': 0.06370193802821, 'max_depth': 3, 'min_child_weigh
t': 9, 'subsample': 0.7961307761975178, 'colsample_bytree': 0.7184818302624372,
'reg_alpha': 6.528184897138473e-05, 'reg_lambda': 0.00011684014870910516}. Best i
s trial 22 with value: 499.74224552133353.
[I 2025-11-29 18:53:03,925] Trial 23 finished with value: 504.5663012985019 and p
arameters: {'learning_rate': 0.06551949215626532, 'max_depth': 4, 'min_child_weig
ht': 9, 'subsample': 0.8127197165470138, 'colsample_bytree': 0.8706293244629475,
'reg_alpha': 9.991257345049593, 'reg_lambda': 6.27532155658388e-05}. Best is tria
l 22 with value: 499.74224552133353.
[I 2025-11-29 18:53:04,018] Trial 24 finished with value: 509.8984975070219 and p
arameters: {'learning_rate': 0.06663821660965417, 'max_depth': 4, 'min_child_weig
ht': 9, 'subsample': 0.8105820739067968, 'colsample_bytree': 0.9998301911661949,
'reg_alpha': 5.160083990584157, 'reg_lambda': 3.138444509413328e-05}. Best is tri
al 22 with value: 499.74224552133353.
[I 2025-11-29 18:53:04,135] Trial 25 finished with value: 536.4142704827865 and p
arameters: {'learning_rate': 0.07207784869593552, 'max_depth': 6, 'min_child_weig
ht': 9, 'subsample': 0.8765229098439522, 'colsample_bytree': 0.5862520398829061,
'reg_alpha': 8.126342185670864e-06, 'reg_lambda': 2.2894878637157081e-07}. Best i
s trial 22 with value: 499.74224552133353.
[I 2025-11-29 18:53:04,263] Trial 26 finished with value: 538.4669007478988 and p
arameters: {'learning_rate': 0.08901570405402813, 'max_depth': 5, 'min_child_weig
ht': 9, 'subsample': 0.7582969735374259, 'colsample_bytree': 0.6859740102156601,
'reg_alpha': 9.248431690531445e-07, 'reg_lambda': 8.609078384699606e-07}. Best is
trial 22 with value: 499.74224552133353.
[I 2025-11-29 18:53:04,378] Trial 27 finished with value: 509.1272047398219 and p
arameters: {'learning_rate': 0.0631256663364159, 'max_depth': 4, 'min_child_weigh
t': 10, 'subsample': 0.810100369739626, 'colsample_bytree': 0.7724214879965837,
'reg_alpha': 4.435086829295099e-08, 'reg_lambda': 0.00018045803359974426}. Best i
s trial 22 with value: 499.74224552133353.
[I 2025-11-29 18:53:04,510] Trial 28 finished with value: 529.7544652577778 and p
arameters: {'learning_rate': 0.07349445342106613, 'max_depth': 5, 'min_child_weig
ht': 9, 'subsample': 0.874079841062663, 'colsample_bytree': 0.49683505698834507,
'reg_alpha': 0.0002857424479167513, 'reg_lambda': 1.960176718468848e-06}. Best is
trial 22 with value: 499.74224552133353.
[I 2025-11-29 18:53:05,142] Trial 29 finished with value: 547.3250272118464 and p
arameters: {'learning_rate': 0.04445746868729125, 'max_depth': 12, 'min_child_wei
ght': 8, 'subsample': 0.9300884099195321, 'colsample_bytree': 0.7661772777349768,
'reg_alpha': 0.08552300885954917, 'reg_lambda': 1.930258295905705e-05}. Best is t
rial 22 with value: 499.74224552133353.
[I 2025-11-29 18:53:05,300] Trial 30 finished with value: 518.9397170774039 and p
arameters: {'learning_rate': 0.05253472829153129, 'max_depth': 6, 'min_child_weig
ht': 10, 'subsample': 0.7268317400203574, 'colsample_bytree': 0.8222987038866751,
'reg_alpha': 0.00015342646604117396, 'reg_lambda': 0.0008761733199574859}. Best i
s trial 22 with value: 499.74224552133353.
[I 2025-11-29 18:53:05,390] Trial 31 finished with value: 500.6582711748276 and p
arameters: {'learning_rate': 0.059457764791436496, 'max_depth': 3, 'min_child_wei
ght': 7, 'subsample': 0.7847113649750779, 'colsample_bytree': 0.8808922995411942,
'reg_alpha': 1.8105603791373856e-05, 'reg_lambda': 4.698189351516156e-05}. Best i
s trial 22 with value: 499.74224552133353.
[I 2025-11-29 18:53:05,470] Trial 32 finished with value: 499.32089152112826 and
parameters: {'learning_rate': 0.0585057569234853, 'max_depth': 3, 'min_child_weig
ht': 7, 'subsample': 0.7787481205447561, 'colsample_bytree': 0.8817267791191168,
'reg_alpha': 2.5647153174816273e-05, 'reg_lambda': 2.4446299551092613e-05}. Best
is trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:05,561] Trial 33 finished with value: 502.86993277579717 and
parameters: {'learning_rate': 0.06003868146242852, 'max_depth': 3, 'min_child_wei
ght': 7, 'subsample': 0.7759022962430897, 'colsample_bytree': 0.8665982608103711,
'reg_alpha': 1.9166160760886925e-05, 'reg_lambda': 3.230979346259666e-05}. Best i
s trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:05,660] Trial 34 finished with value: 499.3232935499192 and p
arameters: {'learning_rate': 0.04828683198680191, 'max_depth': 3, 'min_child_weig
ht': 7, 'subsample': 0.7700721432895637, 'colsample_bytree': 0.8835019828674142,
'reg_alpha': 1.8156591996639654e-05, 'reg_lambda': 5.826590433158976e-06}. Best i
s trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:05,754] Trial 35 finished with value: 502.9534330419848 and p
arameters: {'learning_rate': 0.04779582626644456, 'max_depth': 3, 'min_child_weig
ht': 6, 'subsample': 0.729180903531158, 'colsample_bytree': 0.9030420007311766,
'reg_alpha': 2.0140268580069003e-06, 'reg_lambda': 3.2413391685784252e-06}. Best
is trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:05,932] Trial 36 finished with value: 505.6615882032202 and p
arameters: {'learning_rate': 0.03784778406176327, 'max_depth': 3, 'min_child_weig
ht': 7, 'subsample': 0.8508984248624551, 'colsample_bytree': 0.9573152338166535,
'reg_alpha': 1.4768467923225832e-05, 'reg_lambda': 9.535050243975414e-06}. Best i
s trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:06,027] Trial 37 finished with value: 502.5058469910859 and p
arameters: {'learning_rate': 0.05036978626348005, 'max_depth': 3, 'min_child_weig
ht': 8, 'subsample': 0.762082196620240З, 'colsample_bytree': 0.774379343915373,
'reg_alpha': 9.813377619013606e-08, 'reg_lambda': 1.1071519919509308e-06}. Best i
s trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:06,158] Trial 38 finished with value: 509.13356456668834 and
parameters: {'learning_rate': 0.056669110695278455, 'max_depth': 4, 'min_child_we
ight': 6, 'subsample': 0.7907479472047607, 'colsample_bytree': 0.893972145148957,
'reg_alpha': 2.632553023249673e-06, 'reg_lambda': 3.8860776619998896e-08}. Best i
s trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:06,366] Trial 39 finished with value: 548.7396438070306 and p
arameters: {'learning_rate': 0.07926137292665852, 'max_depth': 8, 'min_child_weig
ht': 7, 'subsample': 0.7041714729274186, 'colsample_bytree': 0.9692971317010344,
'reg_alpha': 5.163509759655891e-07, 'reg_lambda': 1.3830417916465458e-07}. Best i
s trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:06,919] Trial 40 finished with value: 538.7126320357203 and p
arameters: {'learning_rate': 0.04830424324178221, 'max_depth': 11, 'min_child_wei
ght': 7, 'subsample': 0.8405772956326699, 'colsample_bytree': 0.8305748963350774,
'reg_alpha': 2.854186993082141e-05, 'reg_lambda': 9.107281321439613e-05}. Best is
trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:07,056] Trial 41 finished with value: 503.3844012702982 and p
arameters: {'learning_rate': 0.05120431157013512, 'max_depth': 3, 'min_child_weig
ht': 8, 'subsample': 0.7644996726373063, 'colsample_bytree': 0.7762873801855926,
'reg_alpha': 1.997557222056691e-07, 'reg_lambda': 1.3943828093151021e-06}. Best i
s trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:07,197] Trial 42 finished with value: 505.13387844851076 and
parameters: {'learning_rate': 0.06038700911255625, 'max_depth': 3, 'min_child_wei
ght': 8, 'subsample': 0.7424913017903676, 'colsample_bytree': 0.7463778525524039,
'reg_alpha': 0.00013264240990433393, 'reg_lambda': 1.448731841501193e-05}. Best i
s trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:07,330] Trial 43 finished with value: 500.7935691691444 and p
arameters: {'learning_rate': 0.05631655872699935, 'max_depth': 3, 'min_child_weig
ht': 8, 'subsample': 0.7700524690362534, 'colsample_bytree': 0.9174622529066886,
'reg_alpha': 5.364180662809602e-06, 'reg_lambda': 4.4862044079347025e-07}. Best i
s trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:07,467] Trial 44 finished with value: 512.0661586299412 and p
arameters: {'learning_rate': 0.05531034318133195, 'max_depth': 4, 'min_child_weig
ht': 5, 'subsample': 0.791572472525222, 'colsample_bytree': 0.922924823685887, 'r
eg_alpha': 7.97101889485471e-06, 'reg_lambda': 6.977896721488317e-06}. Best is tr
ial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:07,548] Trial 45 finished with value: 516.6725210070502 and p
arameters: {'learning_rate': 0.06755221432080465, 'max_depth': 4, 'min_child_weig
ht': 8, 'subsample': 0.7371508183719653, 'colsample_bytree': 0.8842328320428181,
'reg_alpha': 1.3419308121171366e-06, 'reg_lambda': 4.978804352683302e-07}. Best i
s trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:07,661] Trial 46 finished with value: 514.6371579194794 and p
arameters: {'learning_rate': 0.061315616343532074, 'max_depth': 5, 'min_child_wei
ght': 7, 'subsample': 0.8319133931727982, 'colsample_bytree': 0.8488705722193566,
'reg_alpha': 0.0006904327782088252, 'reg_lambda': 9.845542091514817e-08}. Best is
trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:07,813] Trial 47 finished with value: 501.9068100854262 and p
arameters: {'learning_rate': 0.03391156466663921, 'max_depth': 3, 'min_child_weig
ht': 6, 'subsample': 0.858792988871395, 'colsample_bytree': 0.947444301509027, 'r
```

```
eg_alpha': 0.003502048839289607, 'reg_lambda': 4.097740112221327e-06}. Best is tr
ial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:08,318] Trial 48 finished with value: 538.4750156960908 and p
arameters: {'learning_rate': 0.027267106845709405, 'max_depth': 9, 'min_child_wei
ght': 7, 'subsample': 0.7758855833592675, 'colsample_bytree': 0.9847580458458927,
'reg_alpha': 3.976547659961174e-06, 'reg_lambda': 0.00022124375534366707}. Best i
s trial 32 with value: 499.32089152112826.
[I 2025-11-29 18:53:08,397] Trial 49 finished with value: 503.0420631384252 and p
arameters: {'learning_rate': 0.05445369898778719, 'max_depth': 3, 'min_child_weig
ht': 8, 'subsample': 0.7143497864185664, 'colsample_bytree': 0.6565023632597239,
'reg_alpha': 4.343836062502668e-05, 'reg_lambda': 4.048483282139138e-05}. Best is
trial 32 with value: 499.32089152112826.
Best Parameters found:
{'learning_rate': 0.0585057569234853, 'max_depth': 3, 'min_child_weight': 7, 'sub
sample': 0.7787481205447561, 'colsample_bytree': 0.8817267791191168, 'reg_alpha':
2.5647153174816273e-05, 'reg_lambda': 2.4446299551092613e-05}
```

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_Model"

# 1. Optimization History
fig = plot_optimization_history(study)
fig.update_layout(title=f'{name} Optimization History', width=900, height=500)
# fig.write_image(f'optuna_{name}_history.png') # Optional save
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.write_image(f'optuna_{name}_parallel_coordinate.png')
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.write_image(f'optuna_{name}_slice.png')
fig.show()

# 4. Parameter Importance
try:
    fig = plot_param_importances(study)
    fig.update_layout(title=f'{name} Hyperparameter Importance', width=900, heig
    # fig.write_image(f'optuna_{name}_param_importance.png')
    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])

# 2. Initialize with Best Params
best_params = study.best_params
best_params['n_estimators'] = 1000
best_params['random_state'] = 42
best_params['n_jobs'] = -1
best_params['early_stopping_rounds'] = 50

final_model = xgb.XGBRegressor(**best_params)

# 3. Train on Full History
final_model.fit(
    X_train_full, y_train_full,
    eval_set=[(X_train_full, y_train_full), (X_test, y_test)],
    verbose=100
)

# 4. Final Prediction on TEST Data
y_pred_final_test = final_model.predict(X_test)
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)
```

```
[0]     validation_0-rmse:1826.48491    validation_1-rmse:2226.88210
[100]   validation_0-rmse:461.59994     validation_1-rmse:489.30483
[129]   validation_0-rmse:451.48675     validation_1-rmse:488.89265
```

8. Results and Analysis

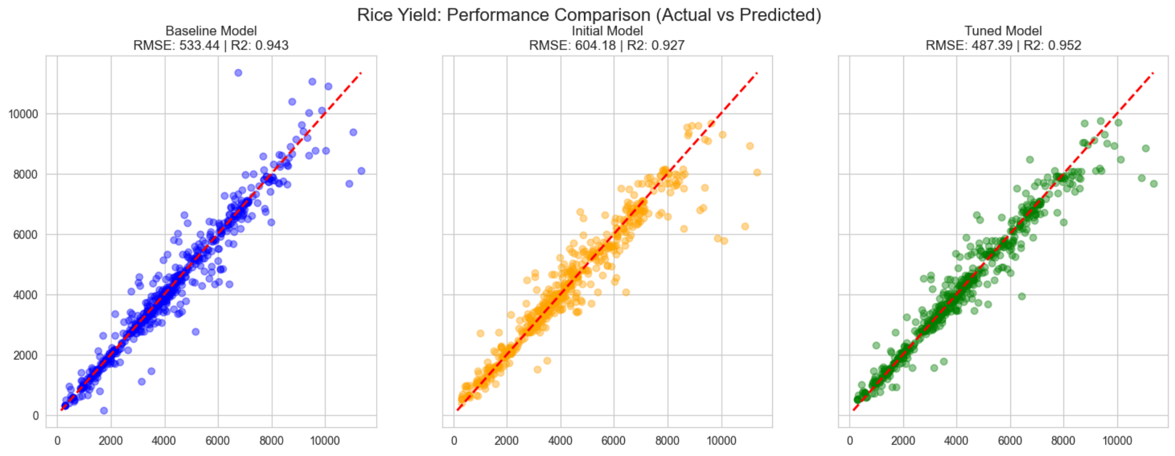
We evaluate the final performance on the Test data (2019–2023).

- **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()
```

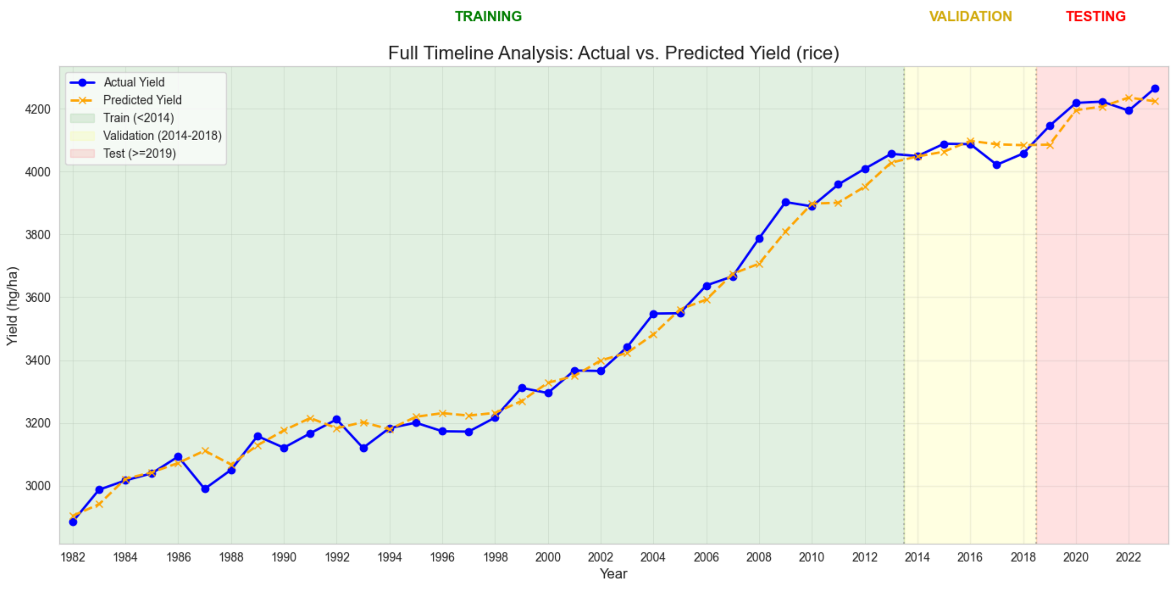
```
--- Final Performance Report (Test Set) ---
Baseline Model: RMSE=533.44, R2=0.9427
Initial Model:  RMSE=604.18, R2=0.9266
Tuned Model:    RMSE=487.39, R2=0.9522 (RMSE Improved 8.63%)
```



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

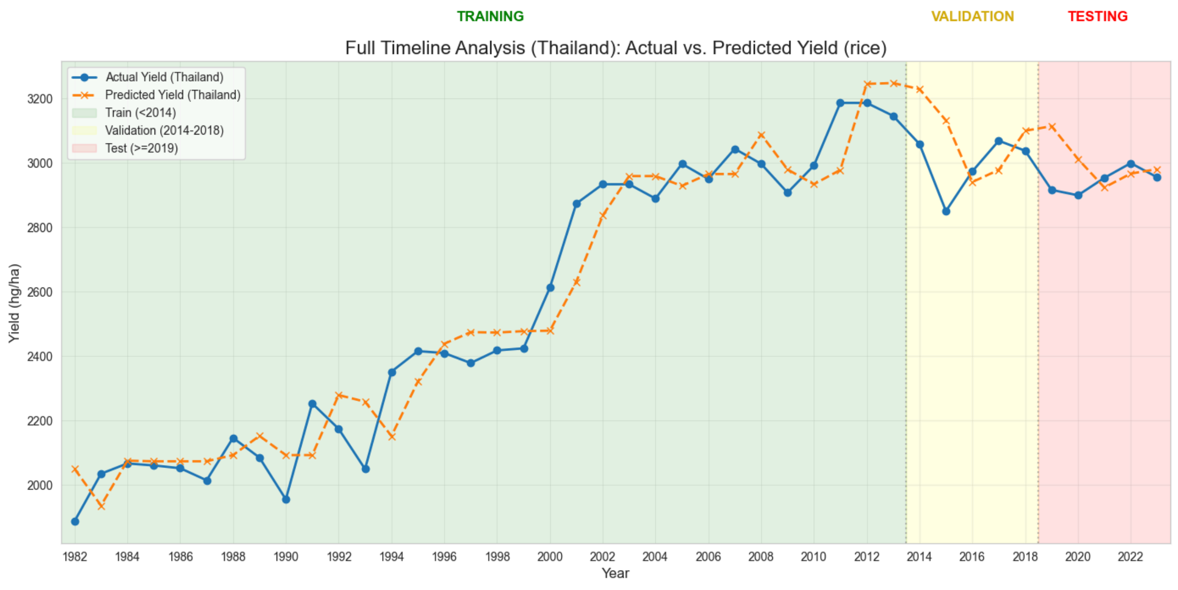
In [11]:
```
import matplotlib.pyplot as plt

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

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

# 2. Create DataFrame
df_full_trend = pd.DataFrame({
    'Year': df_model['year'],
    'Actual': df_model[TARGET_COL],
    'Predicted': all_predictions
})

# Aggregate by Year
yearly_trend = df_full_trend.groupby('Year').mean()

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

```python
# Plot Lines
plt.plot(yearly_trend.index, yearly_trend['Actual'],
         marker='o', label='Actual Yield', linewidth=2, color='blue')
plt.plot(yearly_trend.index, yearly_trend['Predicted'],
         marker='x', linestyle='--', label='Predicted Yield', linewidth=2, color

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

# --- CALCULATE OFFSETS ---
# We shift the boundary back by 0.5 so the line falls *between* years.
# e.g., Split at 2014 becomes 2013.5 (Between 2013 and 2014)
train_boundary = TRAIN_END_YEAR - 0.5
val_boundary = VAL_END_YEAR - 0.5

# --- Highlight Periods ---
# 1. Training Period (< 2014) -> Stops visually at 2013.5
plt.axvspan(MIN_YEAR - 0.5, train_boundary, color='green', alpha=0.1,
            label=f'Train (<{TRAIN_END_YEAR})')

# 2. Validation Period (2014 - 2018) -> From 2013.5 to 2018.5
plt.axvspan(train_boundary, val_boundary, color='yellow', alpha=0.1,
            label=f'Validation ({TRAIN_END_YEAR}-{VAL_END_YEAR - 1})')

# 3. Test Period (>= 2019) -> Starts visually at 2018.5
plt.axvspan(val_boundary, MAX_YEAR + 0.5, color='red', alpha=0.1,
            label=f'Test (>={VAL_END_YEAR})')

# Add Vertical Lines "In Between"
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 * 1.05

# Center text based on the new boundaries
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: Actual vs. Predicted Yield ({CHOSEN_CROP})',
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) # Add slight padding to x-axis
plt.tight_layout()
plt.show()
```



In [22]:
```
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]

# 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 * 1.08

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()
```



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

```
In [12]: # --- REFRESHING THE TEST DATA CONTEXT ---
# 3. Test Set (>= 2019)
VAL_END_YEAR = 2019
mask_test = df_model['year'] >= VAL_END_YEAR

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

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

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

# 2. Join the desired feature column(s) from the test set
# The join works automatically because y_test (and comparison_df) retained
# the index from the original df_model[mask_test].
comparison_df = comparison_df.join(test_set_context)

# 3. Reorder the columns for better readability (optional)
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      4589.864258
38  2020  Afghanistan        4441.7      4596.236328
39  2021  Afghanistan        4406.5      4596.236328
40  2022  Afghanistan        4625.0      4535.262207
41  2023  Afghanistan        4627.9      4610.206543
```

```
In [13]: 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',
```

```python
    '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 ---

    # Squared Error (for RMSE)
    comparison_df['Squared_Error'] = (
        comparison_df['Actual_Value'] - comparison_df['Predicted_Value']
    ) ** 2

    # Squared Percentage Error (for RMSPE)
    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 ---

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

    # Calculate RMSPE
    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 (for hover display)
    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') # Merge RMSP
    error_stats = error_stats.merge(ap_df, on='Country', how='left') # Merge wit

    # 5. Choropleth Map with hover info (Now includes RMSE)
    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', # <-- ADDED RMSE HERE
            '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 updated visualization
plot_geographic_error(comparison_df)
```

```
C:\Users\PavinP\AppData\Local\Temp\ipykernel_10664\1150405013.py:73: DeprecationW
arning:

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*.
```

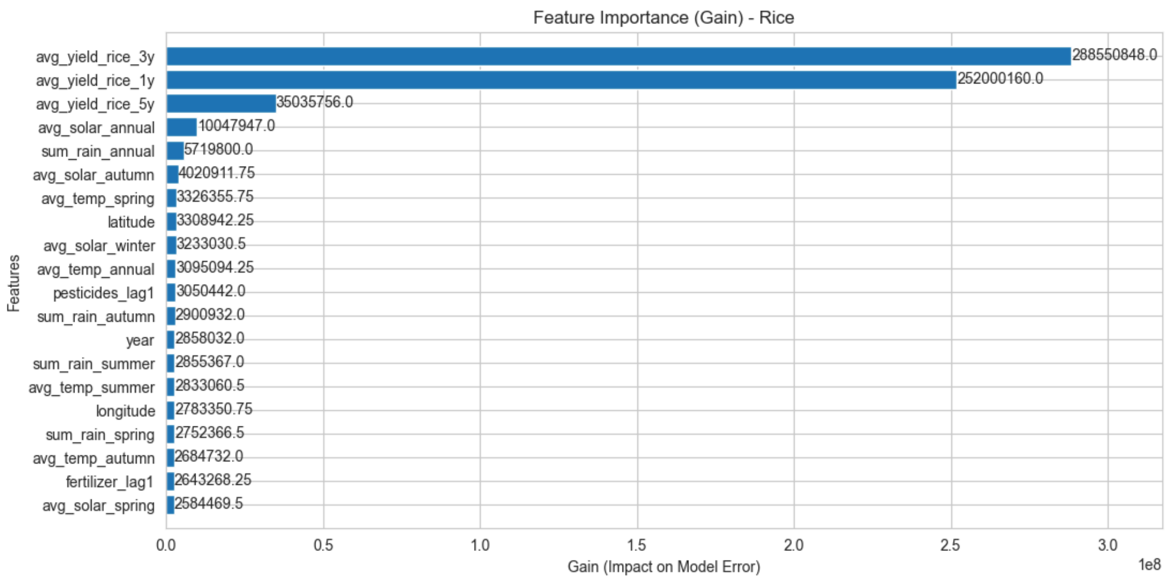
9. Key Factors (Feature Importance)

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 [14]:
```
# --- FEATURE IMPORTANCE: PLOT & TEXT ---

# 1. Extract feature importances
importances = final_model.feature_importances_
feature_names = final_model.feature_names_in_

# 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 XGBoost's built-in plotter
xgb.plot_importance(
    final_model,
    max_num_features=20,
    importance_type='gain',
    height=0.8,
    xlabel='Gain (Impact on Model Error)',
    title=f'Feature Importance (Gain) - {CHOSEN_CROP.capitalize()}'
)
plt.show()
```

```
--- Top 20 Most Important Features (Text Report) ---
              Feature  Importance
0   avg_yield_rice_3y    0.448599
1   avg_yield_rice_1y    0.391775
2   avg_yield_rice_5y    0.054469
3    avg_solar_annual    0.015621
4     sum_rain_annual    0.008892
5    avg_solar_autumn    0.006251
6     avg_temp_spring    0.005171
7            latitude    0.005144
8    avg_solar_winter    0.005026
9     avg_temp_annual    0.004812
10    pesticides_lag1    0.004742
11    sum_rain_autumn    0.004510
12               year    0.004443
13    sum_rain_summer    0.004439
14    avg_temp_summer    0.004404
15          longitude    0.004327
16    sum_rain_spring    0.004279
17    avg_temp_autumn    0.004174
18    fertilizer_lag1    0.004109
19   avg_solar_spring    0.004018
```



Feature Importance (Gain) - Rice
avg_yield_rice_3y
avg_yield_rice_1y
avg_yield_rice_5y
avg_solar_annual
sum_rain_annual
avg_solar_autumn
avg_temp_spring
latitude
avg_solar_winter
avg_temp_annual
pesticides_lag1
sum_rain_autumn
year
sum_rain_summer
avg_temp_summer
longitude
sum_rain_spring
avg_temp_autumn
fertilizer_lag1
avg_solar_spring
288550848.0
252000160.0
35035756.0
10047947.0
5719800.0
4020911.75
3326355.75
3308942.25
3233030.5
3095094.25
3050442.0
2900932.0
2858032.0
2855367.0
2833060.5
2783350.75
2752366.5
2684732.0
2643268.25
2584469.5
Features
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e8
Gain (Impact on Model Error)