```
import gdown
file_id = '1MCmntA3BOquC7BWxLcYDItohDgTPRKx8'
url = f'https://drive.google.com/uc?id={file_id}'
output = 'Crop_Data_Final.csv'
gdown.download(url, output, quiet=False)
```

```
Downloading...
From: https://drive.google.com/uc?id=1MCmntA3BOquC7BWxLcYDItohDgTPRKx8
To: /content/Crop_Data_Final.csv
100%|██████████| 214k/214k [00:00<00:00, 25.7MB/s]
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import pandas as pd
import numpy as np
from joblib import Parallel, delayed

# Load the dataset
file_path = '/content/Crop_Data_Final.csv'
data = pd.read_csv(file_path)

# Future years to forecast
future_years = list(range(2017, 2024)) + [2025, 2030, 2035, 2040, 2045, 2050]

# Districts to process
districts = data['Dist Name'].unique()

# Crop-related target columns
crop_columns = {
    'Rice': ['RICE AREA (1000 ha)', 'RICE PRODUCTION (1000 tons)', 'RICE YIELD (Kg per ha)'],
    'Wheat': ['WHEAT AREA (1000 ha)', 'WHEAT PRODUCTION (1000 tons)', 'WHEAT YIELD (Kg per ha)'],
    'Sorghum': ['SORGHUM AREA (1000 ha)', 'SORGHUM PRODUCTION (1000 tons)', 'SORGHUM YIELD (Kg pe
    'Pearl Millet': ['PEARL MILLET AREA (1000 ha)', 'PEARL MILLET PRODUCTION (1000 tons)', 'PEARL
    'Maize': ['MAIZE AREA (1000 ha)', 'MAIZE PRODUCTION (1000 tons)', 'MAIZE YIELD (Kg per ha)'],
    'Chickpea': ['CHICKPEA AREA (1000 ha)', 'CHICKPEA PRODUCTION (1000 tons)', 'CHICKPEA YIELD (K
    'Pigeonpea': ['PIGEONPEA AREA (1000 ha)', 'PIGEONPEA PRODUCTION (1000 tons)', 'PIGEONPEA YIEL
    'Minor Pulses': ['MINOR PULSES AREA (1000 ha)', 'MINOR PULSES PRODUCTION (1000 tons)', 'MINOR
    'Groundnut': ['GROUNDNUT AREA (1000 ha)', 'GROUNDNUT PRODUCTION (1000 tons)', 'GROUNDNUT YIEL
    'Sesamum': ['SESAMUM AREA (1000 ha)', 'SESAMUM PRODUCTION (1000 tons)', 'SESAMUM YIELD (Kg pe
    'Oilseeds': ['OILSEEDS AREA (1000 ha)', 'OILSEEDS PRODUCTION (1000 tons)', 'OILSEEDS YIELD (K
    'Sugarcane': ['SUGARCANE AREA (1000 ha)', 'SUGARCANE PRODUCTION (1000 tons)', 'SUGARCANE YIEL
    'Cotton': ['COTTON AREA (1000 ha)', 'COTTON PRODUCTION (1000 tons)', 'COTTON YIELD (Kg per ha
    'Fruits and Vegetables': ['FRUITS AND VEGETABLES AREA (1000 ha)'],
    'Fertilizers': ['NITROGEN SHARE IN NPK (Percent)', 'PHOSPHATE SHARE IN NPK (Percent)', 'POTAS
    'Soil Nutrients': ['NITROGEN PER HA OF NCA (Kg per ha)', 'NITROGEN PER HA OF GCA (Kg per ha)'
                       'PHOSPHATE PER HA OF NCA (Kg per ha)', 'PHOSPHATE PER HA OF GCA (Kg per ha
                       'POTASH PER HA OF NCA (Kg per ha)', 'POTASH PER HA OF GCA (Kg per ha)'],
    'Weather': ['Min Temp (Centigrate)', 'Max Temp (Centigrate)', 'Precipitation (mm)', 'Irrigate
}

# Hyperparameter tuning grid
param_distributions = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
```

```
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Function to calculate additional evaluation metrics (MAE, MAPE, RMSE)
def calculate_metrics(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100  # MAPE in percentage
    rmse = np.sqrt(mse)
    return mse, r2, mae, mape, rmse

# Function to process each district and target column
def process_district(district):
    results = {'fine_tuning': [], 'forecasting': []}
    district_data = data[data['Dist Name'] == district]

    for crop, columns in crop_columns.items():
        X = district_data.drop(columns=['Year', 'Dist Name'] + [col for col in data.columns if co

        for target_column in columns:
            y = district_data[target_column]

            if len(y) < 5:
                continue

            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

            # Randomized search for hyperparameter tuning
            rfr = RandomForestRegressor(random_state=42)
            random_search = RandomizedSearchCV(
                estimator=rfr,
                param_distributions=param_distributions,
                n_iter=10,
                cv=2,
                scoring='neg_mean_squared_error',
                n_jobs=-1,
                verbose=0
            )
            random_search.fit(X_train, y_train)

            # Best parameters and model
            best_params = random_search.best_params_
            best_model = random_search.best_estimator_

            # Evaluate the model on test data
            y_pred = best_model.predict(X_test)
            mse, r2, mae, mape, rmse = calculate_metrics(y_test, y_pred)

            # Store fine-tuning results
            results['fine_tuning'].append({
                'District': district,
                'Crop': crop,
                'Target Column': target_column,
                'Best Parameters': best_params,
                'MSE': mse,
                'R²': r2,
```

```
            'MAE': mae,
            'MAPE': mape,
            'RMSE': rmse
        })

        # Forecasting future years
        future_data = district_data.iloc[:len(future_years)].copy()
        future_data['Year'] = future_years
        future_X = future_data.drop(columns=['Year', 'Dist Name'] + [col for col in data.colu
        future_predictions = best_model.predict(future_X)

        # Store forecasted results
        for year, prediction in zip(future_years, future_predictions):
            results['forecasting'].append({
                'District': district,
                'Year': year,
                'Crop': crop,
                'Target Column': target_column,
                'Forecasted Value': prediction
            })

    return results

# Run processing in parallel
all_results = Parallel(n_jobs=-1)(delayed(process_district)(district) for district in districts)

# Combine results
fine_tuning_results = [item for result in all_results for item in result['fine_tuning']]
forecasted_results = [item for result in all_results for item in result['forecasting']]

# Save fine-tuning metrics to CSV
metrics_df = pd.DataFrame(fine_tuning_results)
metrics_csv_path = '/content/fine_tuning_results.csv'
metrics_df.to_csv(metrics_csv_path, index=False)
print(f"Fine-tuning results saved to {metrics_csv_path}")

# Save forecasted results to CSV
forecasted_df = pd.DataFrame(forecasted_results)
forecasted_csv_path = '/content/forecasted_future_values.csv'
forecasted_df.to_csv(forecasted_csv_path, index=False)
print(f"Forecasted values saved to {forecasted_csv_path}")
```

```
Fine-tuning results saved to /content/fine_tuning_results.csv
Forecasted values saved to /content/forecasted_future_values.csv
```

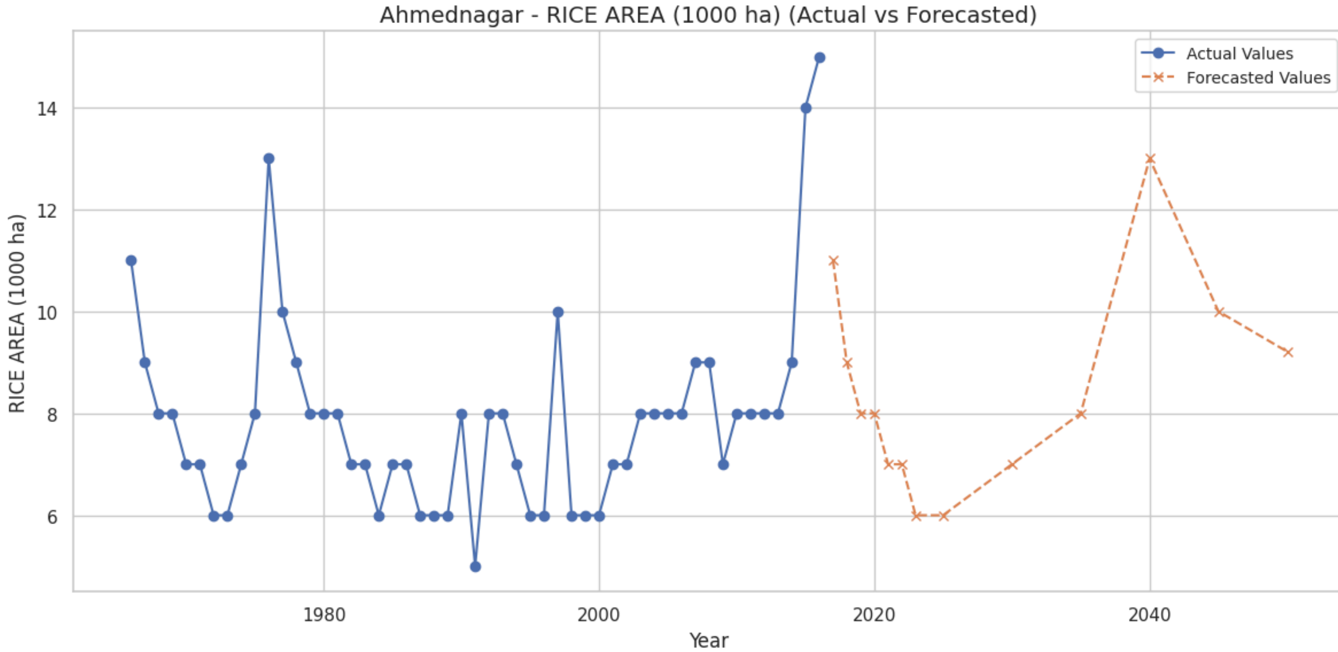
```
# Calculate aggregate average R² for each district
def calculate_aggregate_r2(results):
    district_r2_scores = {}
    for result in results:
        for record in result['fine_tuning']:
            district = record['District']
            r2 = record['R²']
            if district not in district_r2_scores:
                district_r2_scores[district] = []
            district_r2_scores[district].append(r2)

    # Calculate average R² for each district
```

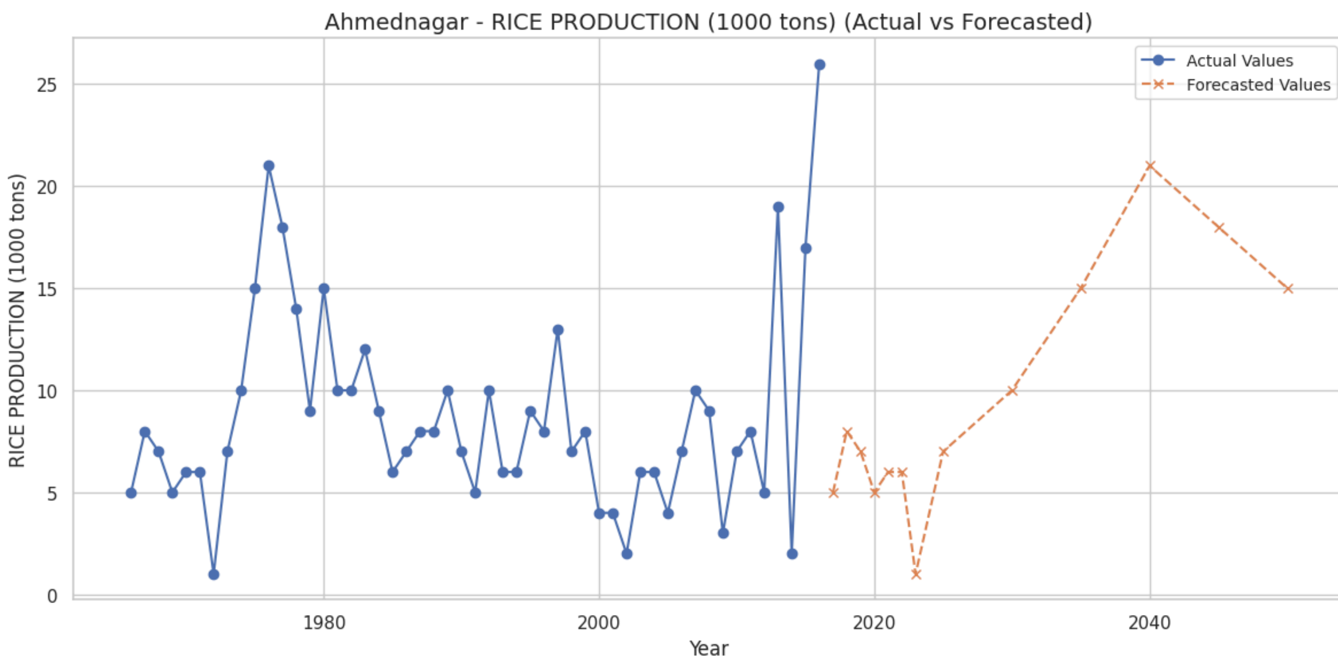
```
    for district, r2_scores in district_r2_scores.items():
        avg_r2 = np.mean(r2_scores)
        print(f"District: {district}, Aggregate Avg R²: {avg_r2:.4f}")

# Run the function to calculate and print the aggregate average R²
calculate_aggregate_r2(all_results)
```

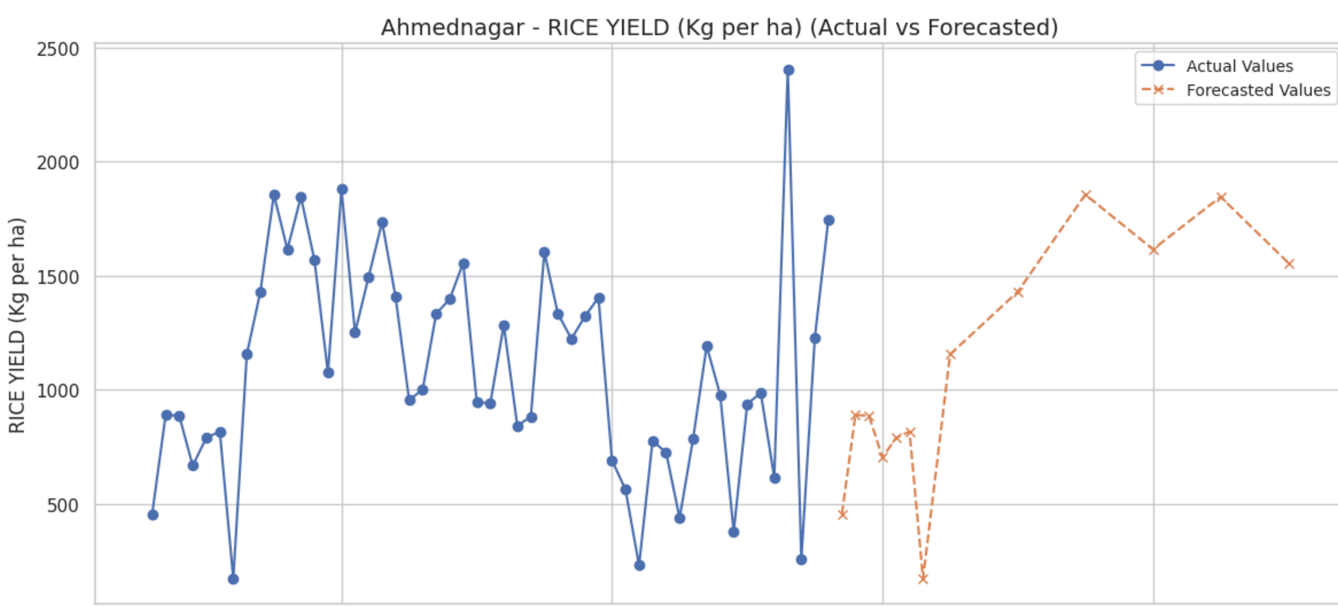
```
District: Ahmednagar, Aggregate Avg R²: 0.9724
District: Akola, Aggregate Avg R²: 0.9706
District: Amarawati, Aggregate Avg R²: 0.9530
District: Aurangabad, Aggregate Avg R²: 0.9667
District: Beed, Aggregate Avg R²: 0.9698
District: Bhandara, Aggregate Avg R²: 0.9546
District: Buldhana, Aggregate Avg R²: 0.9664
District: Chandrapur, Aggregate Avg R²: 0.9651
District: Dhule, Aggregate Avg R²: 0.9830
District: Jalgaon, Aggregate Avg R²: 0.9669
District: Kolhapur, Aggregate Avg R²: 0.9787
District: Nagpur, Aggregate Avg R²: 0.9729
District: Nanded, Aggregate Avg R²: 0.9599
District: Nasik, Aggregate Avg R²: 0.9740
District: Osmanabad, Aggregate Avg R²: 0.9535
District: Parbhani, Aggregate Avg R²: 0.9816
District: Pune, Aggregate Avg R²: 0.9710
District: Sangli, Aggregate Avg R²: 0.9738
District: Satara, Aggregate Avg R²: 0.9588
District: Solapur, Aggregate Avg R²: 0.9752
District: Yeotmal, Aggregate Avg R²: 0.9613
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set seaborn style for plots
sns.set(style="whitegrid")

# Iterate over districts and crop target columns to visualize fitting and forecasting
for district in districts:
    district_data = data[data['Dist Name'] == district]
    forecasted_district_data = forecasted_df[forecasted_df['District'] == district]

    for crop, columns in crop_columns.items():
        for target_column in columns:
            # Check if the target column exists in the district data
            if target_column not in district_data.columns:
                continue

            # Prepare the actual data
            actual_data = district_data[['Year', target_column]].dropna()

            # Prepare the forecasted data
            forecasted_data = forecasted_district_data[
                forecasted_district_data['Target Column'] == target_column
            ]

            # Plot the data
            plt.figure(figsize=(12, 6))
            plt.plot(
                actual_data['Year'], actual_data[target_column], label="Actual Values", marker='c
            )
            plt.plot(
                forecasted_data['Year'], forecasted_data['Forecasted Value'], label="Forecasted V
            )
```

```
# Add title, labels, and legend
plt.title(f"{district} - {target_column} (Actual vs Forecasted)", fontsize=14)
plt.xlabel("Year", fontsize=12)
plt.ylabel(target_column, fontsize=12)
plt.legend(loc="best", fontsize=10)
plt.grid(True)
plt.tight_layout()
plt.show()
```


Ahmednagar - RICE AREA (1000 ha) (Actual vs Forecasted)
Actual Values
Forecasted Values
RICE AREA (1000 ha)
Year
1980
2000
2020
2040
6
8
10
12
14


Ahmednagar - RICE PRODUCTION (1000 tons) (Actual vs Forecasted)
Actual Values
Forecasted Values
RICE PRODUCTION (1000 tons)
Year
1980
2000
2020
2040
0
5
10
15
20
25


Ahmednagar - RICE YIELD (Kg per ha) (Actual vs Forecasted)
Actual Values
Forecasted Values
RICE YIELD (Kg per ha)
500
1000
1500
2000
2500