```
import gdown
file_id = '1rmJoBX-Hrm6MtPtjnVQ4lLUXqytyARPI'
url = f'https://drive.google.com/uc?id={file_id}'
output = 'Major_5_District.csv'
gdown.download(url, output, quiet=False)
import pandas as pd
df = pd.read_csv('Major_5_District.csv')
```

```
Downloading...
From: https://drive.google.com/uc?id=1rmJoBX-Hrm6MtPtjnVQ4lLUXqytyARPI
To: /content/Major_5_District.csv
100%|██████████| 57.8k/57.8k [00:00<00:00, 8.62MB/s]
```

Gradient Boosting

```
df.columns
```

```
Index(['Year', 'Dist Name', 'Area', 'Production', 'Yield', 'Irrigated Area',
       'Annual Rainfall', 'NITROGEN CONSUMPTION (tons)',
       'NITROGEN SHARE IN NPK (Percent)', 'NITROGEN PER HA OF NCA (Kg per ha)',
       'NITROGEN PER HA OF GCA (Kg per ha)', 'PHOSPHATE CONSUMPTION (tons)',
       'PHOSPHATE SHARE IN NPK (Percent)',
       'PHOSPHATE PER HA OF NCA (Kg per ha)',
       'PHOSPHATE PER HA OF GCA (Kg per ha)', 'POTASH CONSUMPTION (tons)',
       'POTASH SHARE IN NPK (Percent)', 'POTASH PER HA OF NCA (Kg per ha)',
       'POTASH PER HA OF GCA (Kg per ha)', 'TOTAL CONSUMPTION (tons)',
       'TOTAL PER HA OF NCA (Kg per ha)', 'TOTAL AREA (1000 ha)',
       'FOREST AREA (1000 ha)', 'BARREN AND UNCULTIVABLE LAND AREA (1000 ha)',
       'LAND PUT TO NONAGRICULTURAL USE AREA (1000 ha)',
       'CULTIVABLE WASTE LAND AREA (1000 ha)',
       'PERMANENT PASTURES AREA (1000 ha)', 'OTHER FALLOW AREA (1000 ha)',
       'CURRENT FALLOW AREA (1000 ha)', 'NET CROPPED AREA (1000 ha)',
       'GROSS CROPPED AREA (1000 ha)', 'CROPING INTENSITY (Percent)',
       'Min Temp (Centigrate)', 'Max Temp (Centigrate)', 'Precipitation (mm)',
       'Evapotranspiration (mm)'],
      dtype='object')
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# Load the dataset
file_path = '/content/Major_5_District.csv'
data = pd.read_csv(file_path)
data = data.drop(columns=['TOTAL AREA (1000 ha)'])

# List of districts to forecast
districts = ['Akola', 'Kolhapur', 'Pune', 'Ratnagiri', 'Wardha']  # Replace with actual district
target_columns = data.columns.drop(['Year', 'Dist Name'])  # Exclude 'Year' and 'Dist Name'

# Define a function to calculate additional metrics
def calculate_metrics(y_true, y_pred):
    r2 = r2_score(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    return {'R2': r2, 'MSE': mse, 'RMSE': rmse, 'MAE': mae, 'MAPE': mape}

# Dictionary to store results
results = {}
```

```
# Iterate over each district
for district in districts:
    print(f"Processing District: {district}")
    district_data = data[data['Dist Name'] == district]
    results[district] = {}

    # Loop through each target column
    for target_column in target_columns:
        print(f"  Forecasting: {target_column}")

        # Prepare features (X) and target (y)
        X = district_data.drop(columns=['Year', 'Dist Name', target_column])
        y = district_data[target_column]

        # Train-test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

        # Gradient Boosting Model
        model = GradientBoostingRegressor(random_state=42)
        model.fit(X_train, y_train)

        # Predictions
        y_pred = model.predict(X_test)

        # Evaluate the model
        metrics = calculate_metrics(y_test, y_pred)
        results[district][target_column] = metrics

# Print results
for district, district_metrics in results.items():
    print(f"\nResults for {district}:")
    for column, metrics in district_metrics.items():
        print(f"  {column}:")
        for metric, value in metrics.items():
            print(f"    {metric}: {value:.4f}")
```

```
Processing District: Akola
  Forecasting: Area
  Forecasting: Production
  Forecasting: Yield
  Forecasting: Irrigated Area
  Forecasting: Annual Rainfall
  Forecasting: NITROGEN CONSUMPTION (tons)
  Forecasting: NITROGEN SHARE IN NPK (Percent)
  Forecasting: NITROGEN PER HA OF NCA (Kg per ha)
  Forecasting: NITROGEN PER HA OF GCA (Kg per ha)
  Forecasting: PHOSPHATE CONSUMPTION (tons)
  Forecasting: PHOSPHATE SHARE IN NPK (Percent)
  Forecasting: PHOSPHATE PER HA OF NCA (Kg per ha)
  Forecasting: PHOSPHATE PER HA OF GCA (Kg per ha)
  Forecasting: POTASH CONSUMPTION (tons)
  Forecasting: POTASH SHARE IN NPK (Percent)
  Forecasting: POTASH PER HA OF NCA (Kg per ha)
  Forecasting: POTASH PER HA OF GCA (Kg per ha)
  Forecasting: TOTAL CONSUMPTION (tons)
  Forecasting: TOTAL PER HA OF NCA (Kg per ha)
  Forecasting: FOREST AREA (1000 ha)
  Forecasting: BARREN AND UNCULTIVABLE LAND AREA (1000 ha)
  Forecasting: LAND PUT TO NONAGRICULTURAL USE AREA (1000 ha)
  Forecasting: CULTIVABLE WASTE LAND AREA (1000 ha)
  Forecasting: PERMANENT PASTURES AREA (1000 ha)
  Forecasting: OTHER FALLOW AREA (1000 ha)
  Forecasting: CURRENT FALLOW AREA (1000 ha)
  Forecasting: NET CROPPED AREA (1000 ha)
  Forecasting: GROSS CROPPED AREA (1000 ha)
```

```
  Forecasting: CROPING INTENSITY (Percent)
  Forecasting: Min Temp (Centigrate)
  Forecasting: Max Temp (Centigrate)
  Forecasting: Precipitation (mm)
  Forecasting: Evapotranspiration (mm)
Processing District: Kolhapur
  Forecasting: Area
  Forecasting: Production
  Forecasting: Yield
  Forecasting: Irrigated Area
  Forecasting: Annual Rainfall
  Forecasting: NITROGEN CONSUMPTION (tons)
  Forecasting: NITROGEN SHARE IN NPK (Percent)
  Forecasting: NITROGEN PER HA OF NCA (Kg per ha)
  Forecasting: NITROGEN PER HA OF GCA (Kg per ha)
  Forecasting: PHOSPHATE CONSUMPTION (tons)
  Forecasting: PHOSPHATE SHARE IN NPK (Percent)
  Forecasting: PHOSPHATE PER HA OF NCA (Kg per ha)
  Forecasting: PHOSPHATE PER HA OF GCA (Kg per ha)
  Forecasting: POTASH CONSUMPTION (tons)
  Forecasting: POTASH SHARE IN NPK (Percent)
  Forecasting: POTASH PER HA OF NCA (Kg per ha)
  Forecasting: POTASH PER HA OF GCA (Kg per ha)
  Forecasting: TOTAL CONSUMPTION (tons)
  Forecasting: TOTAL PER HA OF NCA (Kg per ha)
  Forecasting: FOREST AREA (1000 ha)
  Forecasting: BARREN AND UNCULTIVABLE LAND AREA (1000 ha)
  Forecasting: LAND PUT TO NONAGRICULTURAL USE AREA (1000 ha)
  Forecasting: CULTIVABLE WASTE LAND AREA (1000 ha)
```

```python
# Dictionary to store all forecasted values
forecasted_values = []

# Iterate again over each district to generate forecasted data
for district in districts:
    district_data = data[data['Dist Name'] == district]

    for target_column in target_columns:
        # Prepare features (X) and target (y)
        X = district_data.drop(columns=['Year', 'Dist Name', target_column])
        y = district_data[target_column]

        # Train-test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

        # Gradient Boosting Model
        model = GradientBoostingRegressor(random_state=42)
        model.fit(X_train, y_train)

        # Predictions for the test set
        y_pred = model.predict(X_test)

        # Store results
        for i in range(len(X_test)):
            forecasted_values.append({
                'District': district,
                'Target Column': target_column,
                'Actual Value': y_test.iloc[i],
                'Predicted Value': y_pred[i]
            })

# Convert forecasted values to DataFrame
forecasted_df = pd.DataFrame(forecasted_values)

# Save to CSV
forecasted_csv_path = '/content/forecasted_values.csv'
forecasted_df.to_csv(forecasted_csv_path, index=False)
```

```
print(f"Forecasted values saved to {forecasted_csv_path}")
```

```
Forecasted values saved to /content/forecasted_values.csv
```

```
# Prepare data for performance metrics
metrics_list = []

for district, district_metrics in results.items():
    for column, metrics in district_metrics.items():
        metrics_row = {'District': district, 'Target Column': column}
        metrics_row.update(metrics)  # Add all metrics
        metrics_list.append(metrics_row)

# Convert metrics list to DataFrame
metrics_df = pd.DataFrame(metrics_list)

# Save to CSV
metrics_csv_path = '/content/performance_metrics.csv'
metrics_df.to_csv(metrics_csv_path, index=False)

print(f"Performance metrics saved to {metrics_csv_path}")
```

```
Performance metrics saved to /content/performance_metrics.csv
```

```
# Initialize a list to store aggregate metrics
aggregate_metrics = []

# Iterate through each district
for district in results.keys():
    district_metrics = results[district]

    # Initialize accumulators for each metric
    r2_sum = 0
    mse_sum = 0
    rmse_sum = 0
    mae_sum = 0
    mape_sum = 0
    count = 0

    # Aggregate metrics across all target columns
    for target_column, metrics in district_metrics.items():
        r2_sum += metrics['R2']
        mse_sum += metrics['MSE']
        rmse_sum += metrics['RMSE']
        mae_sum += metrics['MAE']
        mape_sum += metrics['MAPE']
        count += 1

    # Calculate averages
    aggregate_metrics.append({
        'District': district,
        'Avg R²': r2_sum / count,
        'Avg MSE': mse_sum / count,
        'Avg RMSE': rmse_sum / count,
        'Avg MAE': mae_sum / count,
```

```
        'Avg MAPE': mape_sum / count
    })

# Convert aggregate metrics to a DataFrame
aggregate_metrics_df = pd.DataFrame(aggregate_metrics)

# Save to CSV if needed
aggregate_metrics_csv_path = '/content/aggregate_performance_metrics.csv'
aggregate_metrics_df.to_csv(aggregate_metrics_csv_path, index=False)

print(f"Aggregate performance metrics saved to {aggregate_metrics_csv_path}")

# Display the DataFrame
aggregate_metrics_df
```

```
Aggregate performance metrics saved to /content/aggregate_performance_metrics.csv
```

	District	Avg R²	Avg MSE	Avg RMSE	Avg MAE	Avg MAPE
0	Akola	0.547307	1.004943e+06	366.135866	268.107496	11.079484
1	Kolhapur	0.679010	2.239698e+06	509.385907	384.459961	7.548004
2	Pune	0.745560	1.970016e+06	549.471303	394.446955	10.711596
3	Ratnagiri	0.515195	9.109478e+05	280.685871	208.320852	8.678274
4	Wardha	0.690796	8.265315e+05	335.163144	260.200483	14.867005

Forecasting

```
import pandas as pd
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split

# Load the dataset
file_path = '/content/Major_5_District.csv'
data = pd.read_csv(file_path)
data = data.drop(columns=['TOTAL AREA (1000 ha)'])

# Years to forecast
future_years = list(range(2018, 2026)) + [2030, 2035, 2040, 2045]

# List of districts to forecast
districts = ['Akola', 'Kolhapur', 'Pune', 'Ratnagiri', 'Wardha']  # Replace with actual district
target_columns = data.columns.drop(['Year', 'Dist Name'])  # Exclude 'Year' and 'Dist Name'

# Dictionary to store forecasted results
forecasted_results = []

# Iterate over each district
for district in districts:
    print(f"Processing District: {district}")
    district_data = data[data['Dist Name'] == district]

    for target_column in target_columns:
        print(f"  Forecasting: {target_column}")

        # Prepare features (X) and target (y)
```

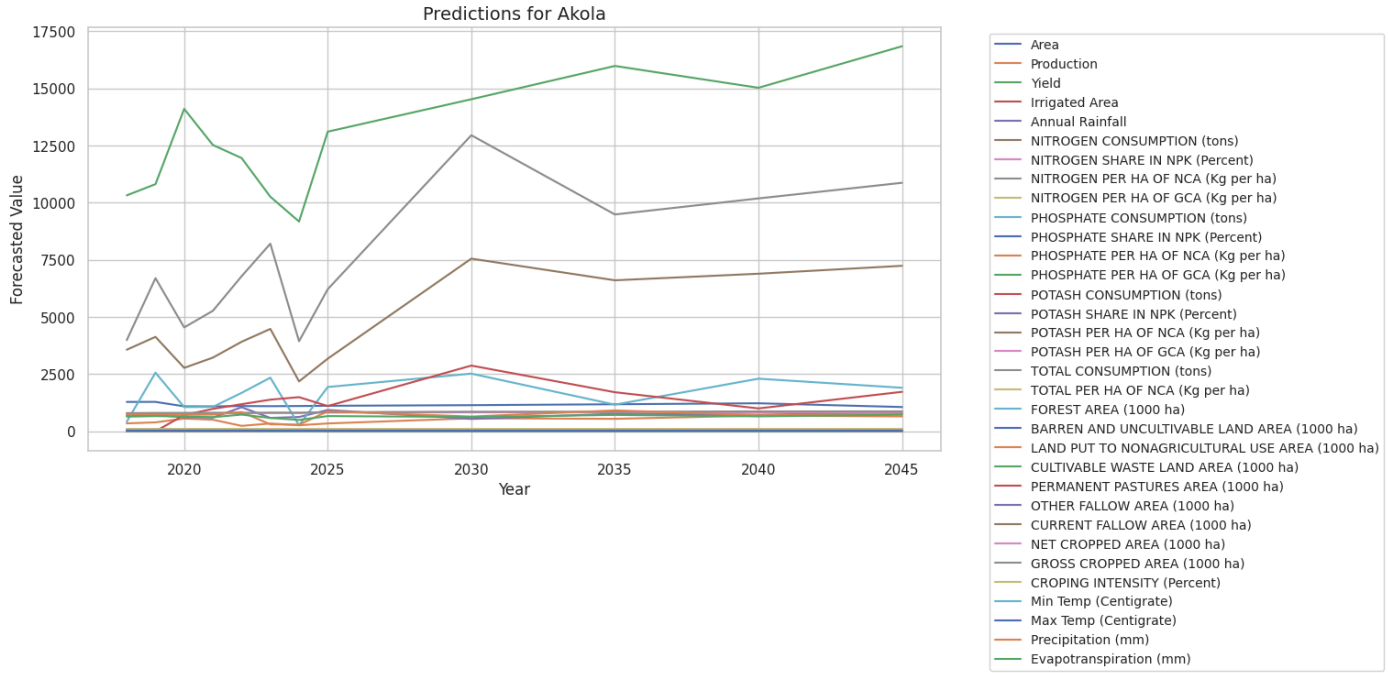
```
        X = district_data.drop(columns=['Year', 'Dist Name', target_column])
        y = district_data[target_column]

        # Train-test split (train on all historical data for forecasting)
        X_train = X
        y_train = y

        # Gradient Boosting Model
        model = GradientBoostingRegressor(random_state=42)
        model.fit(X_train, y_train)

        # Create future data for predictions
        future_data = district_data.iloc[:len(future_years)].copy()
        future_data['Year'] = future_years
        future_X = future_data.drop(columns=['Year', 'Dist Name', target_column], errors='ignore'

        # Predict future values
        future_predictions = model.predict(future_X)

        # Store results
        for year, prediction in zip(future_years, future_predictions):
            forecasted_results.append({
                'District': district,
                'Year': year,
                'Target Column': target_column,
                'Forecasted Value': prediction
            })

# Convert forecasted results to DataFrame
forecasted_df = pd.DataFrame(forecasted_results)

# Save to CSV
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}")
```

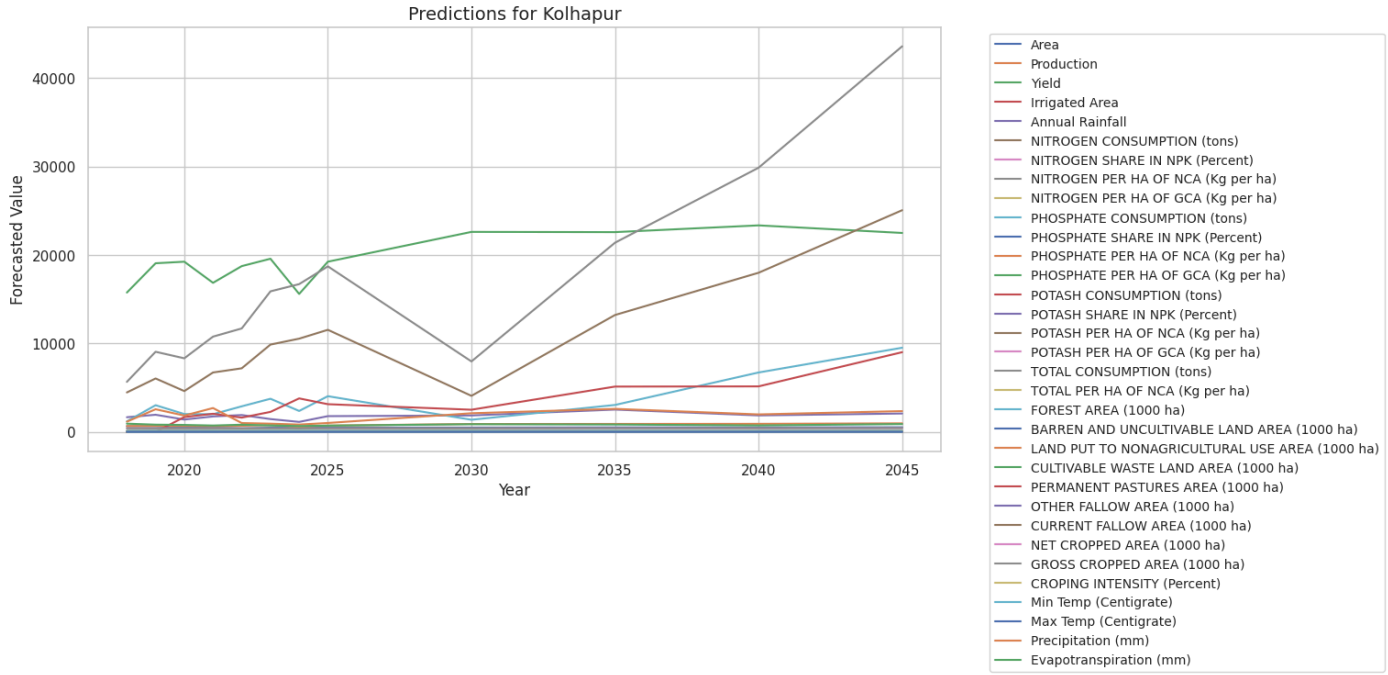
```
Processing District: Akola
  Forecasting: Area
  Forecasting: Production
  Forecasting: Yield
  Forecasting: Irrigated Area
  Forecasting: Annual Rainfall
  Forecasting: NITROGEN CONSUMPTION (tons)
  Forecasting: NITROGEN SHARE IN NPK (Percent)
  Forecasting: NITROGEN PER HA OF NCA (Kg per ha)
  Forecasting: NITROGEN PER HA OF GCA (Kg per ha)
  Forecasting: PHOSPHATE CONSUMPTION (tons)
  Forecasting: PHOSPHATE SHARE IN NPK (Percent)
  Forecasting: PHOSPHATE PER HA OF NCA (Kg per ha)
  Forecasting: PHOSPHATE PER HA OF GCA (Kg per ha)
  Forecasting: POTASH CONSUMPTION (tons)
  Forecasting: POTASH SHARE IN NPK (Percent)
  Forecasting: POTASH PER HA OF NCA (Kg per ha)
  Forecasting: POTASH PER HA OF GCA (Kg per ha)
  Forecasting: TOTAL CONSUMPTION (tons)
  Forecasting: TOTAL PER HA OF NCA (Kg per ha)
  Forecasting: FOREST AREA (1000 ha)
  Forecasting: BARREN AND UNCULTIVABLE LAND AREA (1000 ha)
  Forecasting: LAND PUT TO NONAGRICULTURAL USE AREA (1000 ha)
  Forecasting: CULTIVABLE WASTE LAND AREA (1000 ha)
  Forecasting: PERMANENT PASTURES AREA (1000 ha)
  Forecasting: OTHER FALLOW AREA (1000 ha)
  Forecasting: CURRENT FALLOW AREA (1000 ha)
  Forecasting: NET CROPPED AREA (1000 ha)
  Forecasting: GROSS CROPPED AREA (1000 ha)
```

```
  Forecasting: CROPING INTENSITY (Percent)
  Forecasting: Min Temp (Centigrate)
  Forecasting: Max Temp (Centigrate)
  Forecasting: Precipitation (mm)
  Forecasting: Evapotranspiration (mm)
Processing District: Kolhapur
  Forecasting: Area
  Forecasting: Production
  Forecasting: Yield
  Forecasting: Irrigated Area
  Forecasting: Annual Rainfall
  Forecasting: NITROGEN CONSUMPTION (tons)
  Forecasting: NITROGEN SHARE IN NPK (Percent)
  Forecasting: NITROGEN PER HA OF NCA (Kg per ha)
  Forecasting: NITROGEN PER HA OF GCA (Kg per ha)
  Forecasting: PHOSPHATE CONSUMPTION (tons)
  Forecasting: PHOSPHATE SHARE IN NPK (Percent)
  Forecasting: PHOSPHATE PER HA OF NCA (Kg per ha)
  Forecasting: PHOSPHATE PER HA OF GCA (Kg per ha)
  Forecasting: POTASH CONSUMPTION (tons)
  Forecasting: POTASH SHARE IN NPK (Percent)
  Forecasting: POTASH PER HA OF NCA (Kg per ha)
  Forecasting: POTASH PER HA OF GCA (Kg per ha)
  Forecasting: TOTAL CONSUMPTION (tons)
  Forecasting: TOTAL PER HA OF NCA (Kg per ha)
  Forecasting: FOREST AREA (1000 ha)
  Forecasting: BARREN AND UNCULTIVABLE LAND AREA (1000 ha)
  Forecasting: LAND PUT TO NONAGRICULTURAL USE AREA (1000 ha)
  Forecasting: CULTIVABLE WASTE LAND AREA (1000 ha)
```

Plot

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

# Ensure the plots use seaborn style
sns.set(style="whitegrid")

# Iterate over districts to create individual plots
districts = forecasted_df['District'].unique()

for district in districts:
    # Filter data for the specific district
    district_data = forecasted_df[forecasted_df['District'] == district]

    # Create a new plot for the district
    plt.figure(figsize=(12, 6))

    # Plot each target column
    target_columns = district_data['Target Column'].unique()
    for target in target_columns:
        target_data = district_data[district_data['Target Column'] == target]
        plt.plot(target_data['Year'], target_data['Forecasted Value'], label=target)

    # Add title, labels, and legend
    plt.title(f"Predictions for {district}", fontsize=14)
    plt.xlabel("Year", fontsize=12)
    plt.ylabel("Forecasted Value", fontsize=12)
    plt.legend(loc="upper left", bbox_to_anchor=(1.05, 1), fontsize=10)  # Legend outside the plo
    plt.show()
```





Finetuning


```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np

# Define districts and target columns
districts = ['Akola', 'Kolhapur', 'Pune', 'Ratnagiri', 'Wardha']
target_columns = data.columns.drop(['Year', 'Dist Name'])  # Exclude non-target columns

# Dictionary to store results
fine_tuning_results = []

# Loop through each district
```

```
for district in districts:
    print(f"Processing District: {district}")
    district_data = data[data['Dist Name'] == district]

    # Loop through each target column
    for target_column in target_columns:
        print(f"  Fine-tuning for Target Column: {target_column}")

        # Prepare features (X) and target (y)
        X = district_data.drop(columns=['Year', 'Dist Name', target_column], errors='ignore')
        y = district_data[target_column]

        # Skip if there is insufficient data
        if len(y) < 5:
            print(f"  Skipping {target_column} due to insufficient data.")
            continue

        # Split data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

        # Define the hyperparameter grid
        param_grid = {
            'n_estimators': [100, 200, 300],
            'learning_rate': [0.01, 0.05, 0.1],
            'max_depth': [3, 4, 5],
            'min_samples_split': [2, 5, 10],
            'min_samples_leaf': [1, 3, 5],
            'subsample': [0.8, 1.0]
        }

        # Instantiate the model
        gbr = GradientBoostingRegressor(random_state=42)

        # Use GridSearchCV to find the best parameters
        grid_search = GridSearchCV(estimator=gbr, param_grid=param_grid, cv=3, scoring='neg_mean_
        grid_search.fit(X_train, y_train)

        # Best parameters and evaluation
        best_params = grid_search.best_params_
        best_model = grid_search.best_estimator_
        y_pred = best_model.predict(X_test)

        # Calculate metrics
        mse = mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)
        r2 = best_model.score(X_test, y_test)

        print(f"    RMSE: {rmse:.4f}, R²: {r2:.4f}")

        # Store results
        fine_tuning_results.append({
            'District': district,
            'Target Column': target_column,
            'Best Parameters': best_params,
            'RMSE': rmse,
            'R²': r2
        })
```

```
# Convert results to a DataFrame
results_df = pd.DataFrame(fine_tuning_results)

# Save results to a CSV
results_csv_path = '/content/fine_tuning_results.csv'
results_df.to_csv(results_csv_path, index=False)

print(f"Fine-tuning results saved to {results_csv_path}")
```

```
Processing District: Akola
  Fine-tuning for Target Column: Area
    RMSE: 158.4976, R²: 0.4095
  Fine-tuning for Target Column: Production
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-19-6bc7b730baab> in <cell line: 15>()
     48         # Use GridSearchCV to find the best parameters
     49         grid_search = GridSearchCV(estimator=gbr, param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1,
verbose=0)
---> 50         grid_search.fit(X_train, y_train)
     51
     52         # Best parameters and evaluation

 7 frames
/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in _retrieve(self)
   1760                 (self._jobs[0].get_status(
   1761                     timeout=self.timeout) == TASK_PENDING)):
-> 1762                 time.sleep(0.01)
   1763                 continue
   1764

KeyboardInterrupt:
```

Hail Mary

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

# Load the dataset
file_path = '/content/Major_5_District.csv'
data = pd.read_csv(file_path)
data = data.drop(columns=['TOTAL AREA (1000 ha)'])  # Adjust as per actual column name

# Future years to forecast
future_years = list(range(2018, 2026)) + [2030, 2035, 2040, 2045]

# Districts to process
districts = ['Akola', 'Kolhapur', 'Pune', 'Ratnagiri', 'Wardha']
target_columns = data.columns.drop(['Year', 'Dist Name'])  # Exclude non-target columns

# Hyperparameter tuning grid
param_distributions = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 3],
    'subsample': [0.8, 1.0]
```

```
}

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

    for target_column in target_columns:
        # Prepare features (X) and target (y)
        X = district_data.drop(columns=['Year', 'Dist Name', target_column], errors='ignore')
        y = district_data[target_column]

        # Skip columns with insufficient data
        if len(y) < 5:
            continue

        # Train on historical data
        X_train = X
        y_train = y

        # Randomized search for hyperparameter tuning
        gbr = GradientBoostingRegressor(random_state=42)
        random_search = RandomizedSearchCV(
            estimator=gbr,
            param_distributions=param_distributions,
            n_iter=10,  # Reducing iterations for speed
            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_

        # 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', target_column], errors='ignore'
        future_predictions = best_model.predict(future_X)

        # Store fine-tuning results
        mse = mean_squared_error(y_train, best_model.predict(X_train))
        r2 = r2_score(y_train, best_model.predict(X_train))
        results['fine_tuning'].append({
            'District': district,
            'Target Column': target_column,
            'Best Parameters': best_params,
            'MSE': mse,
            'R²': r2
        })

        # Store forecasted results
        for year, prediction in zip(future_years, future_predictions):
            results['forecasting'].append({
                'District': district,
```

```
                'Year': year,
                '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
```

```
# Initialize lists for aggregate metrics and forecast summaries
aggregate_metrics = []
forecast_summaries = []

# Aggregate fine-tuning results by district
for district in metrics_df['District'].unique():
    district_metrics = metrics_df[metrics_df['District'] == district]

    # Calculate average MSE and R² for the district
    avg_mse = district_metrics['MSE'].mean()
    avg_r2 = district_metrics['R²'].mean()

    # Append aggregate metrics for the district
    aggregate_metrics.append({
        'District': district,
        'Avg MSE': avg_mse,
        'Avg R²': avg_r2
    })

# Aggregate forecasted results by district and year
for district in forecasted_df['District'].unique():
    district_forecasts = forecasted_df[forecasted_df['District'] == district]

    for year in district_forecasts['Year'].unique():
        year_forecasts = district_forecasts[district_forecasts['Year'] == year]

        # Calculate mean forecasted value for the year
        avg_forecast = year_forecasts['Forecasted Value'].mean()
```

```
        # Append forecast summary for the district and year
        forecast_summaries.append({
            'District': district,
            'Year': year,
            'Avg Forecasted Value': avg_forecast
        })

# Convert aggregate metrics and forecast summaries to DataFrames
aggregate_metrics_df = pd.DataFrame(aggregate_metrics)
forecast_summary_df = pd.DataFrame(forecast_summaries)

# Save aggregated metrics to CSV
aggregate_metrics_csv_path = '/content/aggregate_fine_tuning_metrics.csv'
aggregate_metrics_df.to_csv(aggregate_metrics_csv_path, index=False)
print(f"Aggregate fine-tuning metrics saved to {aggregate_metrics_csv_path}")

# Save forecast summaries to CSV
forecast_summary_csv_path = '/content/forecast_summary.csv'
forecast_summary_df.to_csv(forecast_summary_csv_path, index=False)
print(f"Forecast summaries saved to {forecast_summary_csv_path}")

# Display the DataFrames
print("\nAggregate Fine-Tuning Metrics:")
print(aggregate_metrics_df)

print("\nForecast Summary:")
print(forecast_summary_df)
```

```
Aggregate fine-tuning metrics saved to /content/aggregate_fine_tuning_metrics.csv
Forecast summaries saved to /content/forecast_summary.csv

Aggregate Fine-Tuning Metrics:
    District        Avg MSE    Avg R²
0      Akola    2254.957117  0.977871
1   Kolhapur     173.519715  0.955560
2       Pune    4612.143234  0.954814
3  Ratnagiri  136278.467375  0.950932
4     Wardha     297.810368  0.955835

Forecast Summary:
     District  Year  Avg Forecasted Value
0       Akola  2018            741.171287
1       Akola  2019            915.279504
2       Akola  2020            876.541802
3       Akola  2021            871.306690
4       Akola  2022            958.421581
5       Akola  2023            964.419282
6       Akola  2024            674.186669
7       Akola  2025            961.629556
8       Akola  2030           1396.317802
9       Akola  2035           1246.929210
10      Akola  2040           1264.442005
11      Akola  2045           1360.813980
12   Kolhapur  2018           1019.384957
13   Kolhapur  2019           1379.454188
14   Kolhapur  2020           1295.320752
15   Kolhapur  2021           1408.105150
16   Kolhapur  2022           1473.592296
17   Kolhapur  2023           1742.433403
18   Kolhapur  2024           1651.036059
19   Kolhapur  2025           1915.153772
20   Kolhapur  2030           1405.569863
21   Kolhapur  2035           2257.380928
22   Kolhapur  2040           2757.982042
23   Kolhapur  2045           3584.665112
24       Pune  2018           1072.613311
25       Pune  2019           1469.877096
26       Pune  2020           1175.535973
27       Pune  2021           1198.904664
```

```
    28       Pune  2022          1443.719994
    29       Pune  2023          1509.780069
    30       Pune  2024          1490.125363
    31       Pune  2025          1844.445292
    32       Pune  2030          1786.620069
    33       Pune  2035          1496.244709
    34       Pune  2040          1880.540247
    35       Pune  2045          1638.356534
    36  Ratnagiri  2018           804.320929
    37  Ratnagiri  2019           920.650142
    38  Ratnagiri  2020           896.443478
    39  Ratnagiri  2021           921.148232
    40  Ratnagiri  2022           977.503991
    41  Ratnagiri  2023          1013.426245
    42  Ratnagiri  2024           852.681707
    43  Ratnagiri  2025          1035.113936
    44  Ratnagiri  2030           868.411059
```

```python
# Initialize a list to store aggregate metrics
aggregate_metrics = []

# Get unique districts from the results
unique_districts = results_df['District'].unique()

# Iterate through each district
for district in unique_districts:
    # Filter results for the current district
    district_data = results_df[results_df['District'] == district]

    # Calculate average metrics for the district
    avg_rmse = district_data['RMSE'].mean()
    avg_r2 = district_data['R²'].mean()

    # Append aggregate metrics for the district
    aggregate_metrics.append({
        'District': district,
        'Avg RMSE': avg_rmse,
        'Avg R²': avg_r2
    })

# Convert aggregate metrics to a DataFrame
aggregate_metrics_df = pd.DataFrame(aggregate_metrics)

# Save to CSV if needed
aggregate_metrics_csv_path = '/content/aggregate_performance_metrics.csv'
aggregate_metrics_df.to_csv(aggregate_metrics_csv_path, index=False)

print(f"Aggregate performance metrics saved to {aggregate_metrics_csv_path}")

# Display the DataFrame
aggregate_metrics_df



# # Inside the process_district function, after finding the best hyperparameters
# random_search.fit(X_train, y_train)

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

# # Print the best hyperparameters for the current district and target column
# print(f"Best hyperparameters for {district} - {target_column}: {best_params}")
```

```
# Print the best hyperparameters for each district and target column
for result in fine_tuning_results:
    district = result['District']
    target_column = result['Target Column']
    best_params = result['Best Parameters']
    print(f"District: {district}, Target Column: {target_column}, Best Hyperparameters: {best_par
```

```
District: Akola, Target Column: Area, Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'min_samples_split': 5, 'min_sa
District: Akola, Target Column: Production, Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 200, 'min_samples_split': 5, '
District: Akola, Target Column: Yield, Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100, 'min_samples_split': 5, 'min_s
District: Akola, Target Column: Irrigated Area, Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100, 'min_samples_split':
District: Akola, Target Column: Annual Rainfall, Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 200, 'min_samples_split':
District: Akola, Target Column: NITROGEN CONSUMPTION (tons), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'min_sam
District: Akola, Target Column: NITROGEN SHARE IN NPK (Percent), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 300, 'min
District: Akola, Target Column: NITROGEN PER HA OF NCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 200, '
District: Akola, Target Column: NITROGEN PER HA OF GCA (Kg per ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 200, '
District: Akola, Target Column: PHOSPHATE CONSUMPTION (tons), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 300, 'min_sa
District: Akola, Target Column: PHOSPHATE SHARE IN NPK (Percent), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 300, 'mi
District: Akola, Target Column: PHOSPHATE PER HA OF NCA (Kg per ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100,
District: Akola, Target Column: PHOSPHATE PER HA OF GCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300,
District: Akola, Target Column: POTASH CONSUMPTION (tons), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'min_sampl
District: Akola, Target Column: POTASH SHARE IN NPK (Percent), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'min_s
District: Akola, Target Column: POTASH PER HA OF NCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'mi
District: Akola, Target Column: POTASH PER HA OF GCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 100, 'mi
District: Akola, Target Column: TOTAL CONSUMPTION (tons), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100, 'min_sample
District: Akola, Target Column: TOTAL PER HA OF NCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'min
District: Akola, Target Column: FOREST AREA (1000 ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100, 'min_samples_s
District: Akola, Target Column: BARREN AND UNCULTIVABLE LAND AREA (1000 ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators
District: Akola, Target Column: LAND PUT TO NONAGRICULTURAL USE AREA (1000 ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimat
District: Akola, Target Column: CULTIVABLE WASTE LAND AREA (1000 ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 100,
District: Akola, Target Column: PERMANENT PASTURES AREA (1000 ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100, 'm
District: Akola, Target Column: OTHER FALLOW AREA (1000 ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100, 'min_sam
District: Akola, Target Column: CURRENT FALLOW AREA (1000 ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 200, 'min_s
District: Akola, Target Column: NET CROPPED AREA (1000 ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100, 'min_samp
District: Akola, Target Column: GROSS CROPPED AREA (1000 ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'min_sa
District: Akola, Target Column: CROPING INTENSITY (Percent), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100, 'min_sam
District: Akola, Target Column: Min Temp (Centigrate), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 100, 'min_samples_s
District: Akola, Target Column: Max Temp (Centigrate), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 200, 'min_samples_s
District: Akola, Target Column: Precipitation (mm), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 200, 'min_samples_spli
District: Akola, Target Column: Evapotranspiration (mm), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'min_samples
District: Kolhapur, Target Column: Area, Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 100, 'min_samples_split': 2, 'min
District: Kolhapur, Target Column: Production, Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 200, 'min_samples_split': 2
District: Kolhapur, Target Column: Yield, Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'min_samples_split': 5, 'mi
District: Kolhapur, Target Column: Irrigated Area, Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 100, 'min_samples_split
District: Kolhapur, Target Column: Annual Rainfall, Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 300, 'min_samples_spli
District: Kolhapur, Target Column: NITROGEN CONSUMPTION (tons), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 200, 'min_
District: Kolhapur, Target Column: NITROGEN SHARE IN NPK (Percent), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 200, '
District: Kolhapur, Target Column: NITROGEN PER HA OF NCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 200
District: Kolhapur, Target Column: NITROGEN PER HA OF GCA (Kg per ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100
District: Kolhapur, Target Column: PHOSPHATE CONSUMPTION (tons), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'min
District: Kolhapur, Target Column: PHOSPHATE SHARE IN NPK (Percent), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 200,
District: Kolhapur, Target Column: PHOSPHATE PER HA OF NCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 20
District: Kolhapur, Target Column: PHOSPHATE PER HA OF GCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 10
District: Kolhapur, Target Column: POTASH CONSUMPTION (tons), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 100, 'min_sa
District: Kolhapur, Target Column: POTASH SHARE IN NPK (Percent), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 200, 'mi
District: Kolhapur, Target Column: POTASH PER HA OF NCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 100,
District: Kolhapur, Target Column: POTASH PER HA OF GCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300,
District: Kolhapur, Target Column: TOTAL CONSUMPTION (tons), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'min_sam
District: Kolhapur, Target Column: TOTAL PER HA OF NCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 200, '
District: Kolhapur, Target Column: FOREST AREA (1000 ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100, 'min_sample
District: Kolhapur, Target Column: BARREN AND UNCULTIVABLE LAND AREA (1000 ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimat
District: Kolhapur, Target Column: LAND PUT TO NONAGRICULTURAL USE AREA (1000 ha), Best Hyperparameters: {'subsample': 1.0, 'n_esti
District: Kolhapur, Target Column: CULTIVABLE WASTE LAND AREA (1000 ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 3
District: Kolhapur, Target Column: PERMANENT PASTURES AREA (1000 ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 200,
```

```
# Calculate and print aggregate metrics for each district
aggregate_metrics = []

for district in districts:
    district_results = [result for result in fine_tuning_results if result['District'] == distric

    if district_results:
        avg_r2 = np.mean([res['R²'] for res in district_results])
        avg_mse = np.mean([res['MSE'] for res in district_results])
```

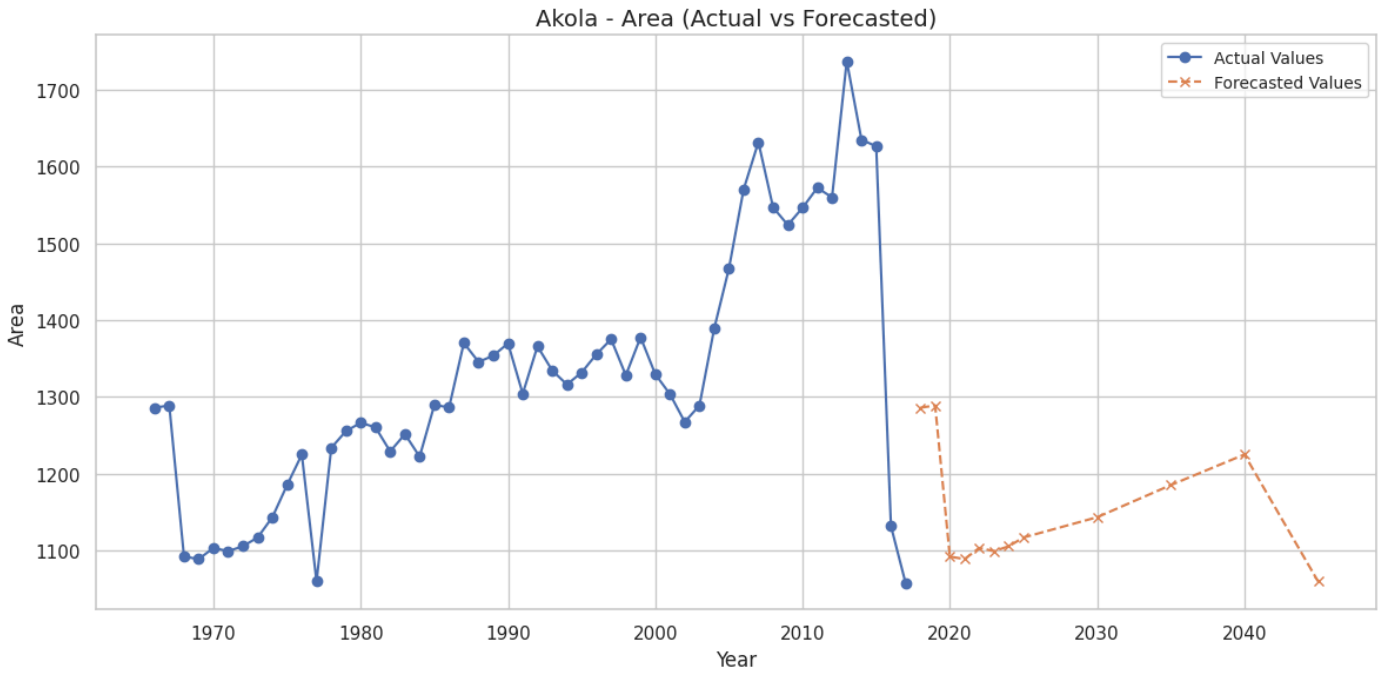
```
        count = len(district_results)

        aggregate_metrics.append({
            'District': district,
            'Average R²': avg_r2,
            'Average MSE': avg_mse,
            'Models Tuned': count
        })

        print(f"District: {district}")
        print(f"  Average R²: {avg_r2:.4f}")
        print(f"  Average MSE: {avg_mse:.4f}")
        print(f"  Models Tuned: {count}")
        print("-" * 40)

# Convert aggregate metrics to DataFrame for further use if needed
aggregate_metrics_df = pd.DataFrame(aggregate_metrics)
```

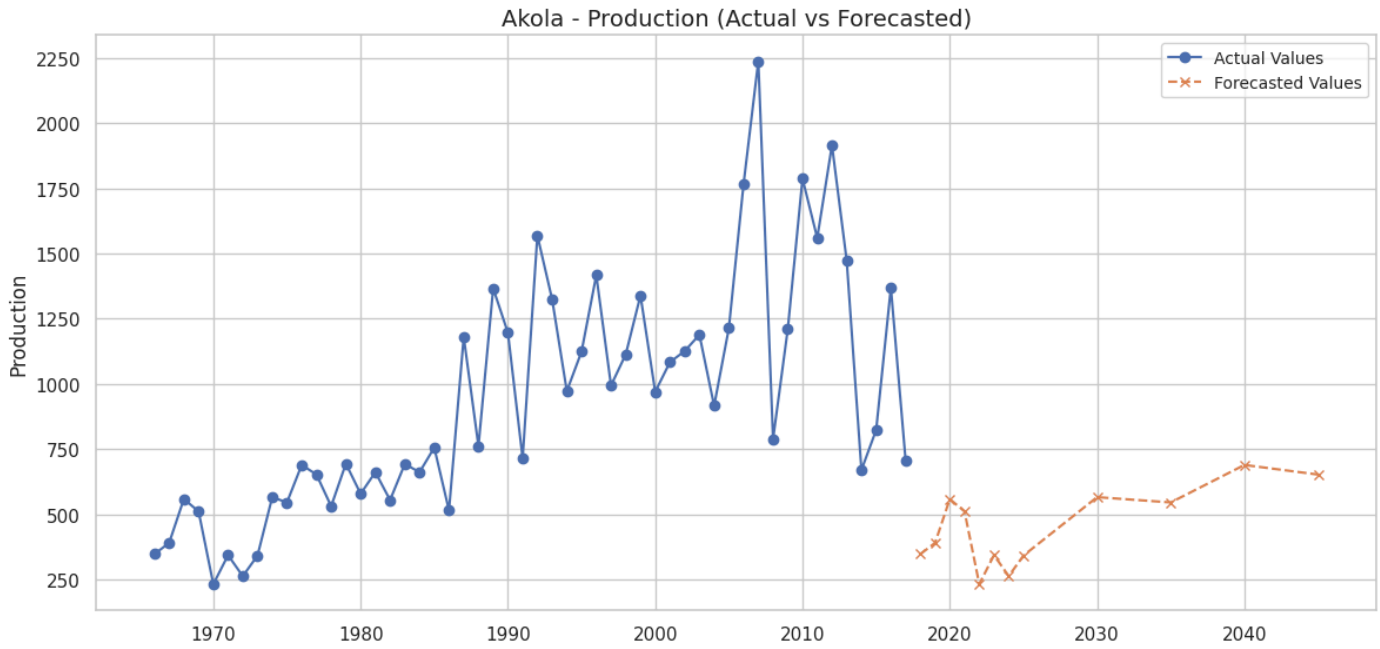
```
District: Akola
  Average R²: 0.9779
  Average MSE: 2254.9571
  Models Tuned: 33
----------------------------------------
District: Kolhapur
  Average R²: 0.9556
  Average MSE: 173.5197
  Models Tuned: 33
----------------------------------------
District: Pune
  Average R²: 0.9548
  Average MSE: 4612.1432
  Models Tuned: 33
----------------------------------------
District: Ratnagiri
  Average R²: 0.9509
  Average MSE: 136278.4674
  Models Tuned: 33
----------------------------------------
District: Wardha
  Average R²: 0.9558
  Average MSE: 297.8104
  Models Tuned: 33
----------------------------------------
```

Plot

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

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

# Iterate over districts and 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 target_column in target_columns:
        # Prepare the actual data
        if target_column not in district_data.columns:
            continue
        actual_data = district_data[['Year', target_column]].dropna()
```

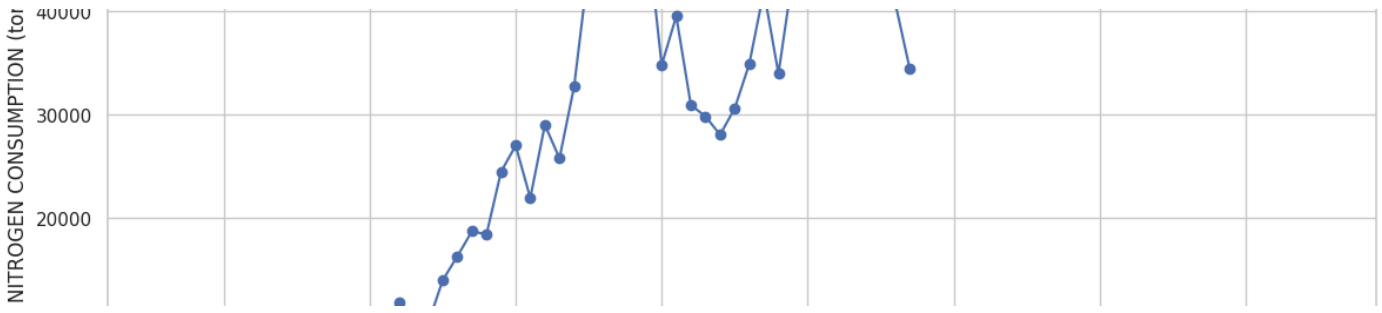
```
# Prepare the forecasted data
forecasted_data = forecasted_district_data[forecasted_district_data['Target Column'] == t

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

# 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()
```

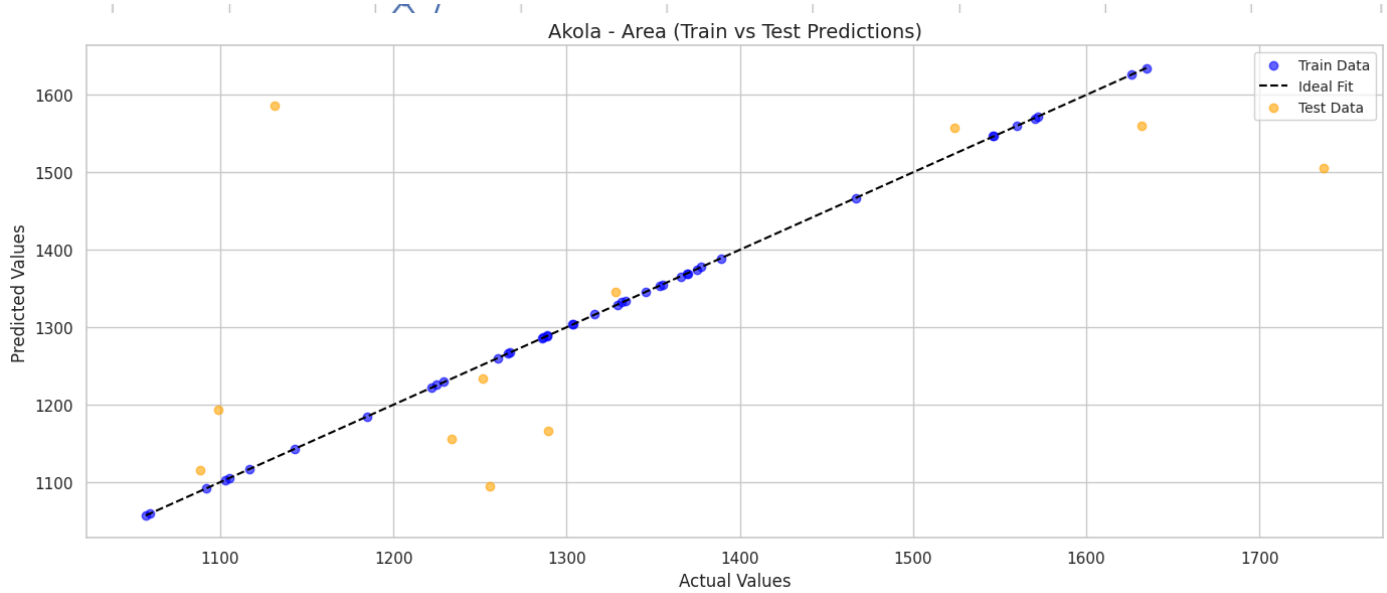


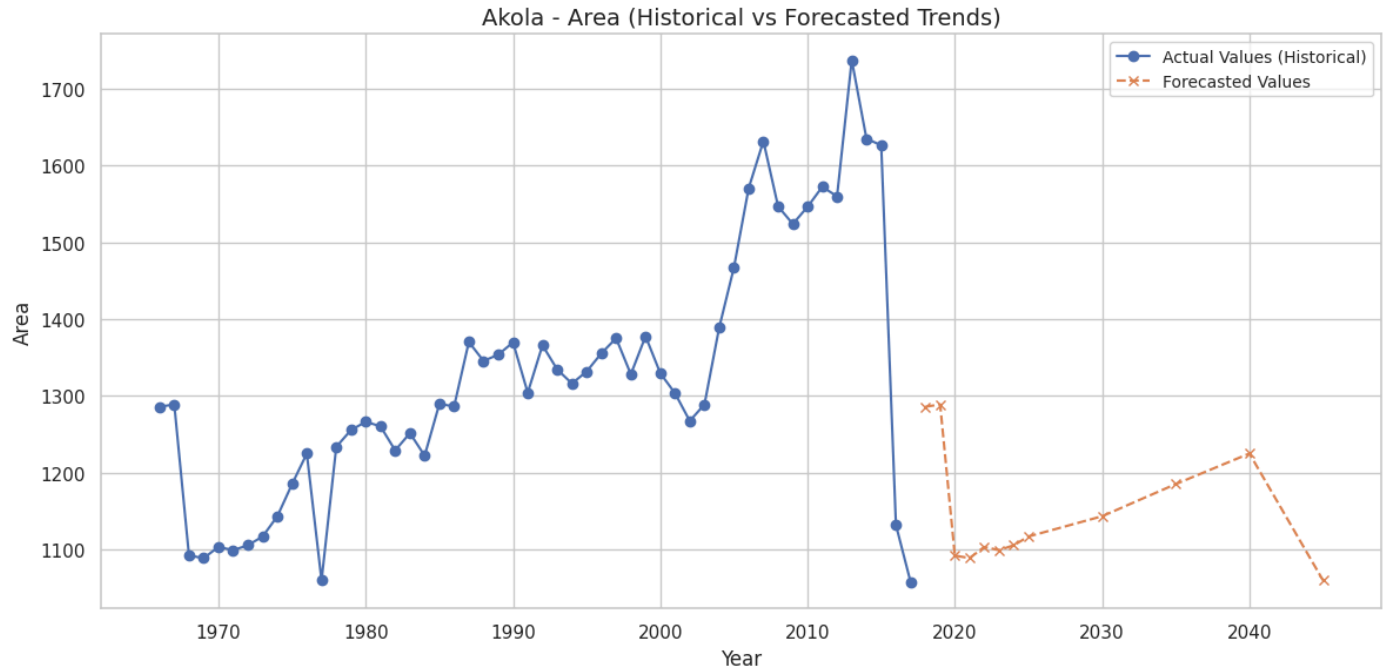


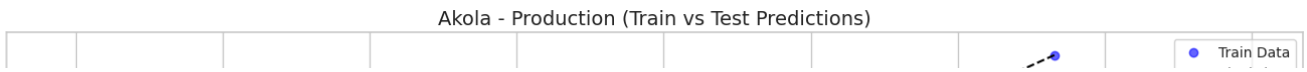

```
from sklearn.model_selection import train_test_split

# Iterate over districts and target columns to visualize train-test fitting and forecasting
for district in districts:
    district_data = data[data['Dist Name'] == district]

    for target_column in target_columns:
        if target_column not in district_data.columns:
            continue

        # Prepare features and target
        X = district_data.drop(columns=['Year', 'Dist Name', target_column], errors='ignore')
        y = district_data[target_column]

        if len(y) < 5:  # Skip if insufficient data
            continue
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the Gradient Boosting model
gbr = GradientBoostingRegressor(random_state=42)
gbr.fit(X_train, y_train)

# Predictions on train and test sets
train_predictions = gbr.predict(X_train)
test_predictions = gbr.predict(X_test)

# Plot actual vs predicted for train and test data
plt.figure(figsize=(14, 6))

# Train data
plt.scatter(y_train, train_predictions, label='Train Data', color='blue', alpha=0.6)
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)], color='black', lines

# Test data
plt.scatter(y_test, test_predictions, label='Test Data', color='orange', alpha=0.6)

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

# Forecasting future trends
forecasted_district_data = forecasted_df[forecasted_df['District'] == district]
forecasted_data = forecasted_district_data[forecasted_district_data['Target Column'] == t
actual_data = district_data[['Year', target_column]].dropna()

# Plot historical and forecasted trends
plt.figure(figsize=(12, 6))
plt.plot(actual_data['Year'], actual_data[target_column], label="Actual Values (Historica
plt.plot(forecasted_data['Year'], forecasted_data['Forecasted Value'], label="Forecasted

# Add title, labels, and legend
plt.title(f"{district} - {target_column} (Historical vs Forecasted Trends)", 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()
```



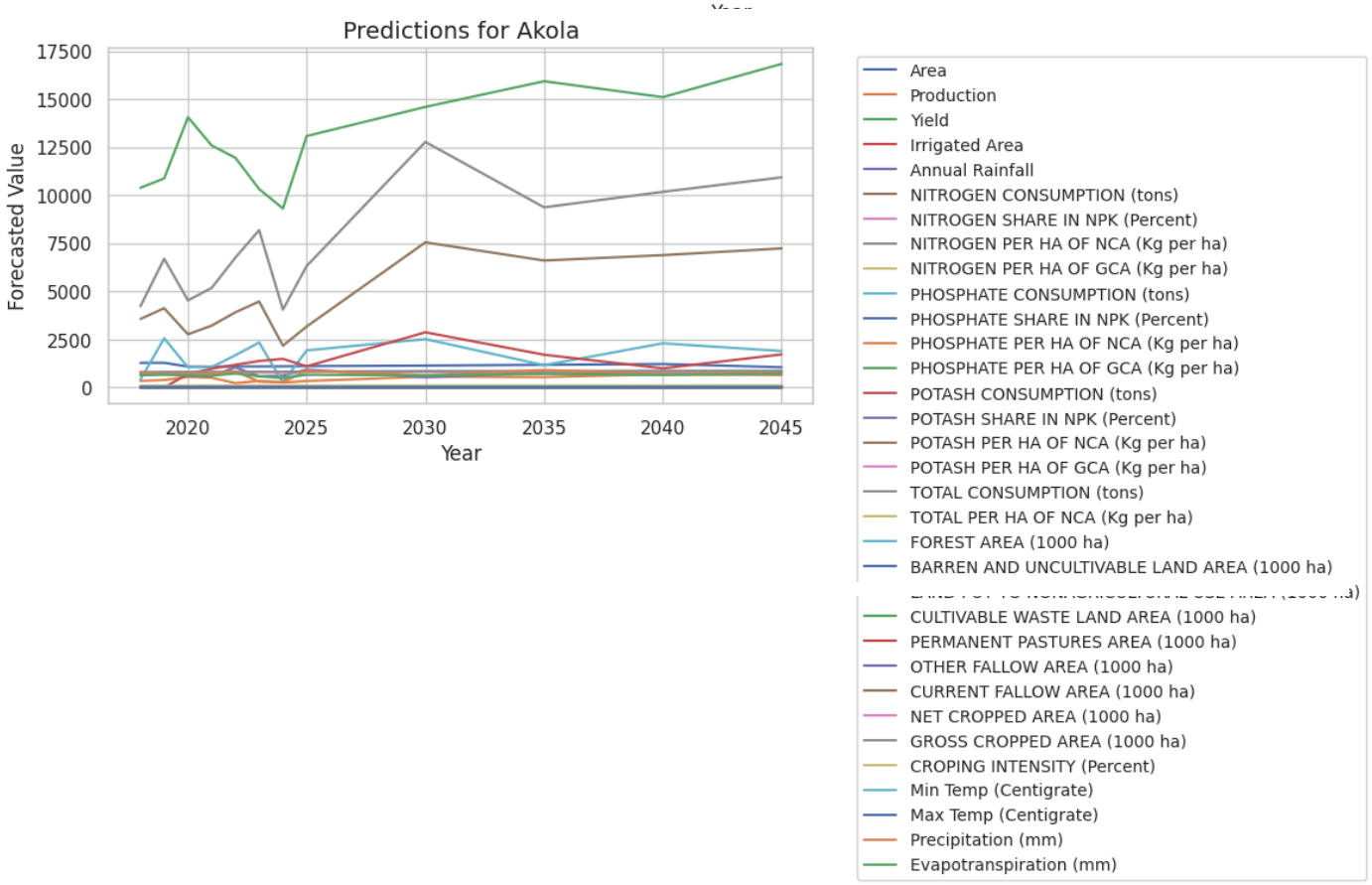


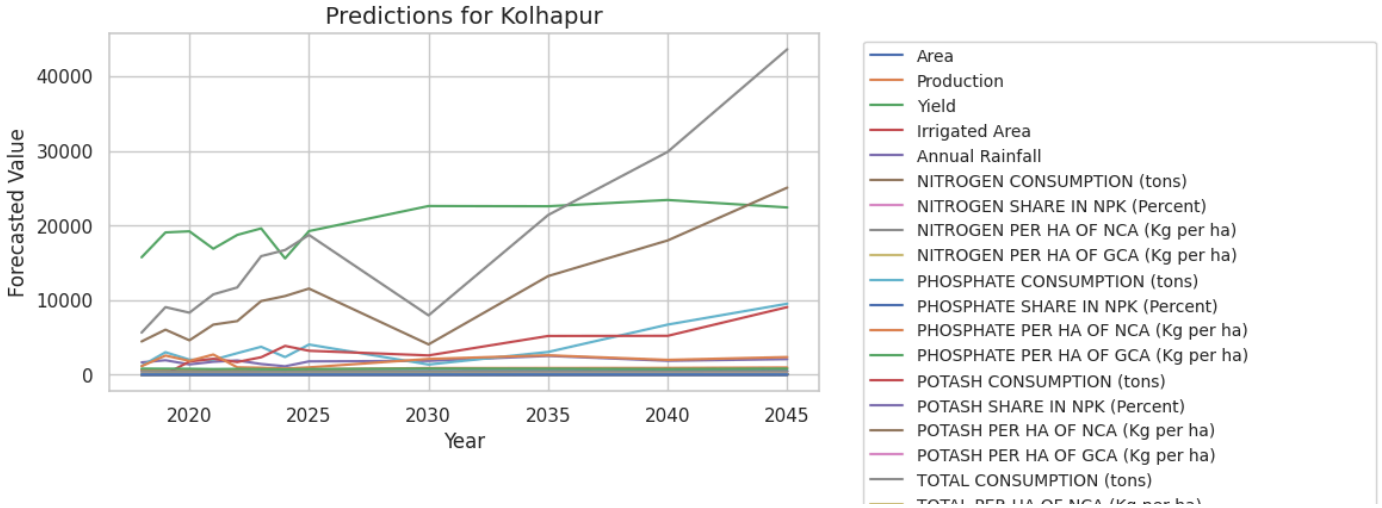




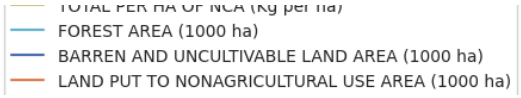

```
import matplotlib.pyplot as plt
import seaborn as sns

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

# Get unique districts
districts = forecasted_df['District'].unique()

# Iterate over districts to create individual plots
for district in districts:
    # Filter data for the specific district
    district_data = forecasted_df[forecasted_df['District'] == district]

    # Create a new plot for the district
    plt.figure(figsize=(12, 6))
```

```
# Plot each target column
target_columns = district_data['Target Column'].unique()
for target in target_columns:
    target_data = district_data[district_data['Target Column'] == target]
    plt.plot(target_data['Year'], target_data['Forecasted Value'], label=target)

# Add title, labels, and legend
plt.title(f"Predictions for {district}", fontsize=14)
plt.xlabel("Year", fontsize=12)
plt.ylabel("Forecasted Value", fontsize=12)
plt.legend(loc="upper left", bbox_to_anchor=(1.05, 1), fontsize=10)  # Legend outside the plo

# Show the plot
plt.tight_layout()
plt.show()
```



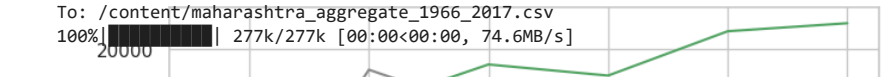


18 Jan




```
import gdown
file_id = '1KXMl4EVNofFnGsypt1UpZ4GJL6pGLJCe'
url = f'https://drive.google.com/uc?id={file_id}'
output = 'maharashtra_aggregate_1966_2017.csv'
gdown.download(url, output, quiet=False)
import pandas as pd
df = pd.read_csv('maharashtra_aggregate_1966_2017.csv')
```

```
Downloading...
From: https://drive.google.com/uc?id=1KXMl4EVNofFnGsypt1UpZ4GJL6pGLJCe
To: /content/maharashtra_aggregate_1966_2017.csv
100%|██████████| 277k/277k [00:00<00:00, 74.6MB/s]
```



Start coding or generate with AI.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# Load the dataset

data = pd.read_csv('maharashtra_aggregate_1966_2017.csv')
data = data.drop(columns=['TOTAL AREA (1000 ha)','State Name'])

# List of districts to forecast
districts = [
    "Ahmednagar", "Akola", "Amarawati", "Aurangabad", "Beed", "Bhandara",
    "Buldhana", "Chandrapur", "Dhule", "Jalgaon", "Kolhapur", "Nagpur",
    "Nanded", "Nasik", "Osmanabad", "Parbhani", "Pune", "Sangli",
    "Satara", "Solapur", "Yeotmal"
]

target_columns = data.columns.drop(['Year', 'Dist Name'])  # Exclude 'Year' and 'Dist Name'

# Define a function to calculate additional metrics
def calculate_metrics(y_true, y_pred):
    r2 = r2_score(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    return {'R2': r2, 'MSE': mse, 'RMSE': rmse, 'MAE': mae, 'MAPE': mape}

# Dictionary to store results
results = {}

# Iterate over each district
# Iterate over each district
for district in districts:
    print(f"Processing District: {district}")
    district_data = data[data['Dist Name'] == district]
    results[district] = {}

    # Loop through each target column
```

```
    for target_column in target_columns:
        print(f"  Forecasting: {target_column}")

        # Prepare features (X) and target (y)
        X = district_data.drop(columns=['Year', 'Dist Name', target_column])
        y = district_data[target_column]

        # Check if X and y are empty
        if X.empty or y.empty:
            print(f"    Skipping {target_column} for {district} due to insufficient data.")
            continue  # Skip to the next target column

        # Train-test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

        # Gradient Boosting Model
        model = GradientBoostingRegressor(random_state=42)
        model.fit(X_train, y_train)

        # Predictions
        y_pred = model.predict(X_test)

        # Evaluate the model
        metrics = calculate_metrics(y_test, y_pred)
        results[district][target_column] = metrics

# Print results
for district, district_metrics in results.items():
    print(f"\nResults for {district}:")
    for column, metrics in district_metrics.items():
        print(f"  {column}:")
        for metric, value in metrics.items():
            print(f"    {metric}: {value:.4f}")

# Dictionary to store all forecasted values
forecasted_values = []

# Iterate over each district
for district in districts:
    print(f"Processing District: {district}")
    district_data = data[data['Dist Name'] == district]
    results[district] = {}

    # Loop through each target column
    for target_column in target_columns:
        print(f"  Forecasting: {target_column}")

        # Prepare features (X) and target (y)
        X = district_data.drop(columns=['Year', 'Dist Name', target_column])
        y = district_data[target_column]

        # Check if there is sufficient data for this target column
        if X.empty or y.empty:
            print(f"    Skipping {target_column} for {district} due to insufficient data.")
            continue  # Skip to the next target column

        # Train-test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

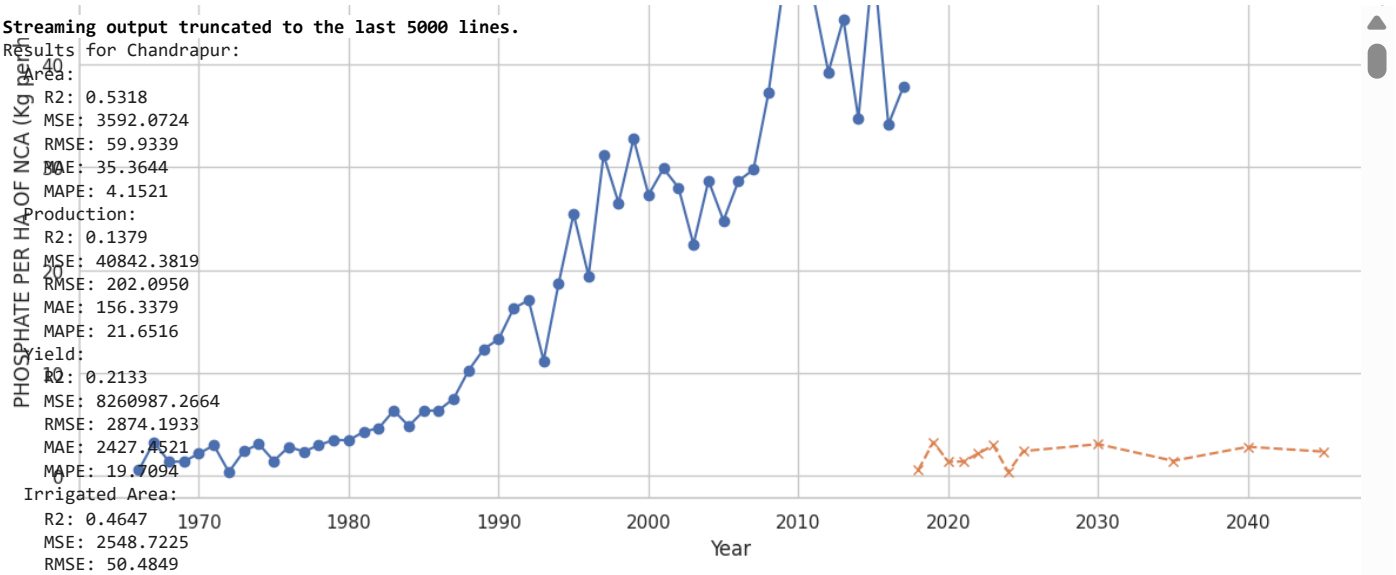
```
        # Gradient Boosting Model
        model = GradientBoostingRegressor(random_state=42)
        model.fit(X_train, y_train)

        # Predictions
        y_pred = model.predict(X_test)

        # Evaluate the model
        metrics = calculate_metrics(y_test, y_pred)
        results[district][target_column] = metrics

# Iterate over each district
for district in districts:
    print(f"Processing District: {district}")
    district_data = data[data['Dist Name'] == district]
    results[district] = {}

    # Loop through each target column
    for target_column in target_columns:
        print(f"  Forecasting: {target_column}")

        # Prepare features (X) and target (y)
        X = district_data.drop(columns=['Year', 'Dist Name', target_column])
        y = district_data[target_column]

        # Check if there is sufficient data for this target column
        if X.empty or y.empty:
            print(f"    Skipping {target_column} for {district} due to insufficient data.")
            continue  # Skip to the next target column

        # Train-test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


        # Store results
        for i in range(len(X_test)):
            forecasted_values.append({
                'District': district,
                'Target Column': target_column,
                'Actual Value': y_test.iloc[i],
                'Predicted Value': y_pred[i]
            })

# Convert forecasted values to DataFrame
forecasted_df = pd.DataFrame(forecasted_values)

# Save to CSV
forecasted_csv_path = '/content/forecasted_values_18Jan.csv'
forecasted_df.to_csv(forecasted_csv_path, index=False)

print(f"Forecasted values saved to {forecasted_csv_path}")

# Prepare data for performance metrics
metrics_list = []

for district, district_metrics in results.items():
    for column, metrics in district_metrics.items():
```

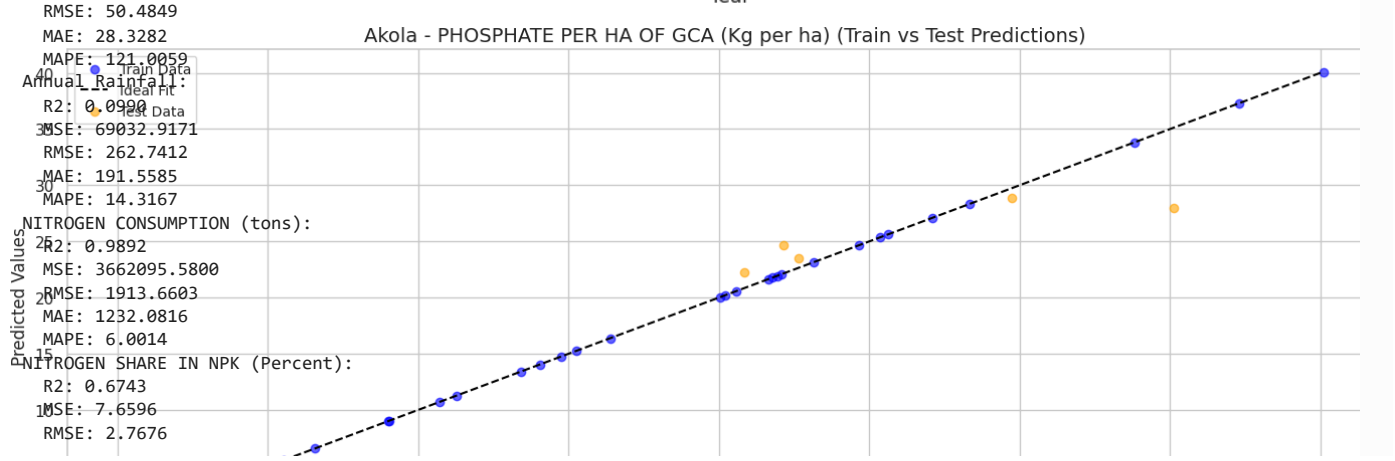
```
        metrics_row = {'District': district, 'Target Column': column}
        metrics_row.update(metrics)  # Add all metrics
        metrics_list.append(metrics_row)

# Convert metrics list to DataFrame
metrics_df = pd.DataFrame(metrics_list)

# Save to CSV
metrics_csv_path = '/content/performance_metrics_18Jan.csv'
metrics_df.to_csv(metrics_csv_path, index=False)

print(f"Performance metrics saved to {metrics_csv_path}")

# Initialize a list to store aggregate metrics
aggregate_metrics = []

# Iterate through each district
# Initialize a list to store aggregate metrics
aggregate_metrics = []

# Iterate through each district
for district in results.keys():
    district_metrics = results[district]

    # Initialize accumulators for each metric
    r2_sum = 0
    mse_sum = 0
    rmse_sum = 0
    mae_sum = 0
    mape_sum = 0
    count = 0

    # Aggregate metrics across all target columns
    for target_column, metrics in district_metrics.items():
        r2_sum += metrics['R2']
        mse_sum += metrics['MSE']
        rmse_sum += metrics['RMSE']
        mae_sum += metrics['MAE']
        mape_sum += metrics['MAPE']
        count += 1  # Increment count inside the target column loop

    # Calculate averages, but only if count is greater than 0
    if count > 0:  # Check to avoid division by zero
        aggregate_metrics.append({
            'District': district,
            'Avg R²': r2_sum / count,
            'Avg MSE': mse_sum / count,
            'Avg RMSE': rmse_sum / count,
            'Avg MAE': mae_sum / count,
            'Avg MAPE': mape_sum / count
        })
    else:
        print(f"Skipping district {district} due to insufficient data for all target columns")
        # Convert aggregate metrics to a DataFrame
aggregate_metrics_df = pd.DataFrame(aggregate_metrics)

# Save to CSV if needed
aggregate_metrics_csv_path = '/content/aggregate_performance_metrics_18Jan.csv'
```

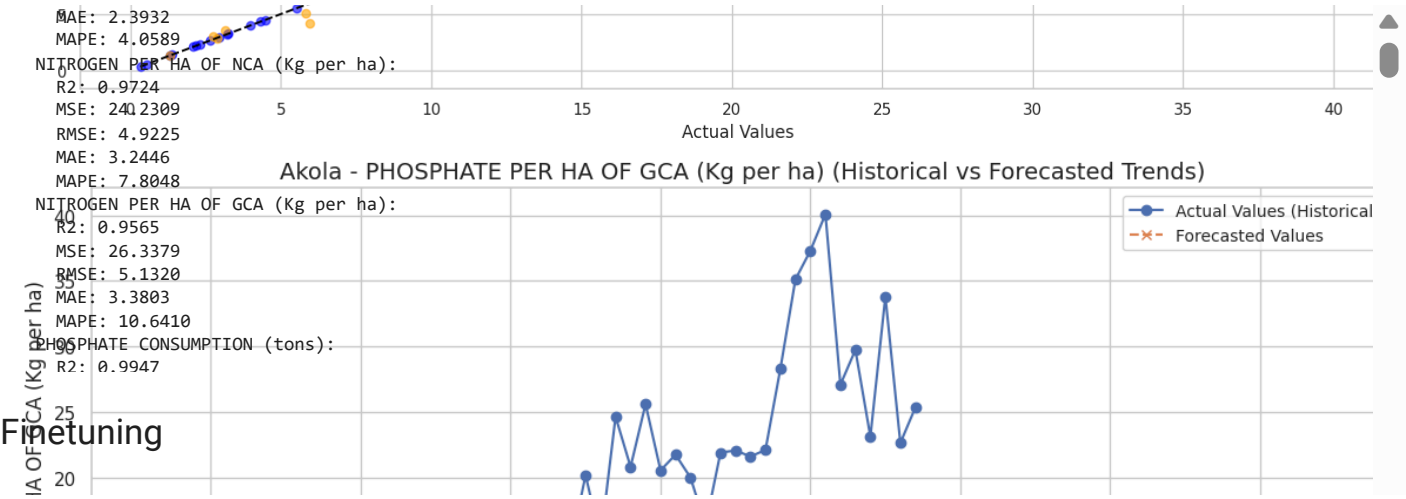
```
aggregate_metrics_df.to_csv(aggregate_metrics_csv_path, index=False)

print(f"Aggregate performance metrics saved to {aggregate_metrics_csv_path}")

# Display the DataFrame
aggregate_metrics_df


import pandas as pd
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split

# Load the dataset
data = pd.read_csv('maharashtra_aggregate_1966_2017.csv')
data = data.drop(columns=['TOTAL AREA (1000 ha)','State Name'])
# Years to forecast
future_years = list(range(2018, 2026)) + [2030, 2035, 2040, 2045]

# List of districts to forecast
districts = [
    "Ahmednagar", "Akola", "Amarawati", "Aurangabad", "Beed", "Bhandara",
    "Buldhana", "Chandrapur", "Dhule", "Jalgaon", "Kolhapur", "Nagpur",
    "Nanded", "Nasik", "Osmanabad", "Parbhani", "Pune", "Sangli",
    "Satara", "Solapur", "Yeotmal"
]
 # Replace with actual district names
target_columns = data.columns.drop(['Year', 'Dist Name'])  # Exclude 'Year' and 'Dist Name'

# Dictionary to store forecasted results
forecasted_results = []

# Iterate over each district
for district in districts:
    print(f"Processing District: {district}")
    # Ensure district name is consistent with the DataFrame
    district_data = data[data['Dist Name'].str.strip() == district.strip()]

    # Check if district_data is empty before proceeding
    if district_data.empty:
        print(f"Skipping {district} due to no matching data.")
        continue

    results[district] = {}

    for target_column in target_columns:
        print(f"  Forecasting: {target_column}")

        # Prepare features (X) and target (y)
        X = district_data.drop(columns=['Year', 'Dist Name', target_column])
        y = district_data[target_column]

        # Train-test split (train on all historical data for forecasting)
        X_train = X
        y_train = y

        # Gradient Boosting Model
        model = GradientBoostingRegressor(random_state=42)
```

```
        model.fit(X_train, y_train)

        # Create future data for predictions
        future_data = district_data.iloc[:len(future_years)].copy()
        future_data['Year'] = future_years
        future_X = future_data.drop(columns=['Year', 'Dist Name', target_column], errors='ignore'

        # Predict future values
        future_predictions = model.predict(future_X)

        # Store results
        for year, prediction in zip(future_years, future_predictions):
            forecasted_results.append({
                'District': district,
                'Year': year,
                'Target Column': target_column,
                'Forecasted Value': prediction
            })

# Convert forecasted results to DataFrame
forecasted_df = pd.DataFrame(forecasted_results)

# Save to CSV
forecasted_csv_path = '/content/18Jan_forecasted_future_values.csv'
forecasted_df.to_csv(forecasted_csv_path, index=False)

print(f"Forecasted values saved to {forecasted_csv_path}")
```

```
Streaming output truncated to the last 5000 lines.
Results for Chandrapur:
Area:
  R2: 0.5318
  MSE: 3592.0724
  RMSE: 59.9339
  MAE: 35.3644
  MAPE: 4.1521
Production:
  R2: 0.1379
  MSE: 40842.3819
  RMSE: 202.0950
  MAE: 156.3379
  MAPE: 21.6516
Yield:
  R2: 0.2133
  MSE: 8260987.2664
  RMSE: 2874.1933
  MAE: 2427.4521
  MAPE: 19.7094
Irrigated Area:
  R2: 0.4647
  MSE: 2548.7225
  RMSE: 50.4849
  MAE: 28.3282
  MAPE: 121.0059
Annual Rainfall:
  R2: 0.0990
  MSE: 69032.9171
  RMSE: 262.7412
  MAE: 191.5585
  MAPE: 14.3167
NITROGEN CONSUMPTION (tons):
  R2: 0.9892
  MSE: 3662095.5800
  RMSE: 1913.6603
  MAE: 1232.0816
  MAPE: 6.0014
NITROGEN SHARE IN NPK (Percent):
  R2: 0.6743
  MSE: 7.6596
  RMSE: 2.7676
```





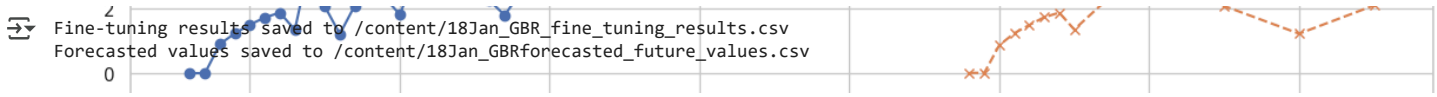


Finetuning

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

# Load the dataset
file_path = '/content/maharashtra_aggregate_1966_2017.csv'
data = pd.read_csv(file_path)
data = data.drop(columns=['TOTAL AREA (1000 ha)','State Name'])  # Adjust as per actual column na

# Future years to forecast
future_years = list(range(2018, 2026)) + [2030, 2035, 2040, 2045]

# Districts to process
districts = [
    "Ahmednagar", "Akola", "Amarawati", "Aurangabad", "Beed", "Bhandara",
    "Buldhana", "Chandrapur", "Dhule", "Jalgaon", "Kolhapur", "Nagpur",
    "Nanded", "Nasik", "Osmanabad", "Parbhani", "Pune", "Sangli",
    "Satara", "Solapur", "Yeotmal"
]

target_columns = data.columns.drop(['Year', 'Dist Name'])  # Exclude non-target columns

# Hyperparameter tuning grid
param_distributions = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 3],
    'subsample': [0.8, 1.0]
}

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

    for target_column in target_columns:
        # Prepare features (X) and target (y)
        X = district_data.drop(columns=['Year', 'Dist Name', target_column], errors='ignore')
```

```
        y = district_data[target_column]

        # Skip columns with insufficient data
        if len(y) < 5:
            continue

        # Train on historical data
        X_train = X
        y_train = y

        # Randomized search for hyperparameter tuning
        gbr = GradientBoostingRegressor(random_state=42)
        random_search = RandomizedSearchCV(
            estimator=gbr,
            param_distributions=param_distributions,
            n_iter=10,  # Reducing iterations for speed
            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_

        # 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', target_column], errors='ignore'
        future_predictions = best_model.predict(future_X)

        # Store fine-tuning results
        mse = mean_squared_error(y_train, best_model.predict(X_train))
        r2 = r2_score(y_train, best_model.predict(X_train))
        results['fine_tuning'].append({
            'District': district,
            'Target Column': target_column,
            'Best Parameters': best_params,
            'MSE': mse,
            'R²': r2
        })

        # Store forecasted results
        for year, prediction in zip(future_years, future_predictions):
            results['forecasting'].append({
                'District': district,
                'Year': year,
                '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/18Jan_GBR_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/18Jan_GBRforecasted_future_values.csv'
forecasted_df.to_csv(forecasted_csv_path, index=False)
print(f"Forecasted values saved to {forecasted_csv_path}")
```



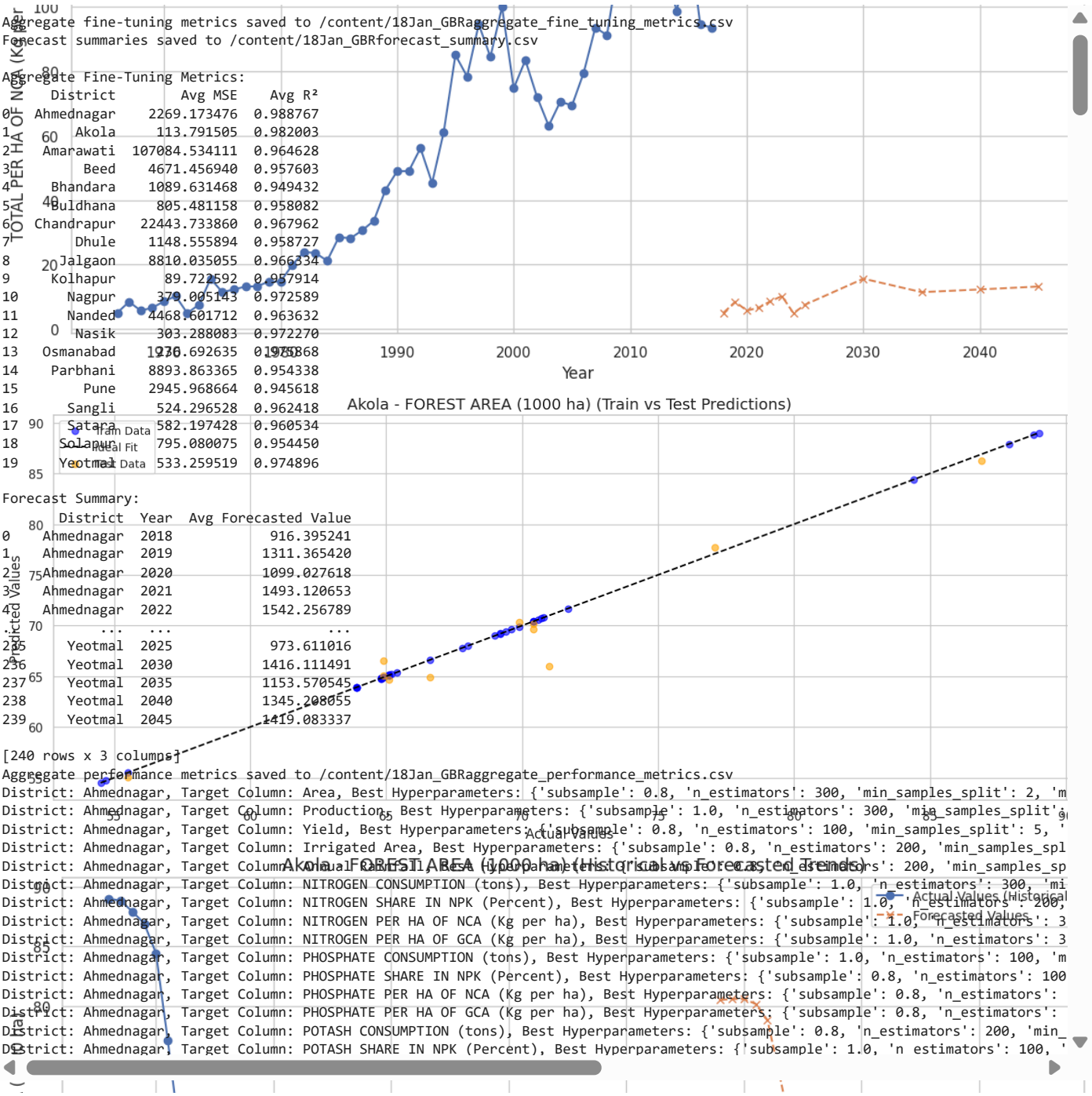

```
Fine-tuning results saved to /content/18Jan_GBR_fine_tuning_results.csv
Forecasted values saved to /content/18Jan_GBRforecasted_future_values.csv
```

```
# Initialize lists for aggregate metrics and forecast summaries
aggregate_metrics = []
forecast_summaries = []

# Aggregate fine-tuning results by district
for district in metrics_df['District'].unique():
    district_metrics = metrics_df[metrics_df['District'] == district]

    # Calculate average MSE and R² for the district
    avg_mse = district_metrics['MSE'].mean()
    avg_r2 = district_metrics['R²'].mean()

    # Append aggregate metrics for the district
    aggregate_metrics.append({
        'District': district,
        'Avg MSE': avg_mse,
        'Avg R²': avg_r2
    })

# Aggregate forecasted results by district and year
for district in forecasted_df['District'].unique():
    district_forecasts = forecasted_df[forecasted_df['District'] == district]

    for year in district_forecasts['Year'].unique():
        year_forecasts = district_forecasts[district_forecasts['Year'] == year]

        # Calculate mean forecasted value for the year
        avg_forecast = year_forecasts['Forecasted Value'].mean()

        # Append forecast summary for the district and year
        forecast_summaries.append({
            'District': district,
            'Year': year,
            'Avg Forecasted Value': avg_forecast
        })

# Convert aggregate metrics and forecast summaries to DataFrames
aggregate_metrics_df = pd.DataFrame(aggregate_metrics)
```

```
forecast_summary_df = pd.DataFrame(forecast_summaries)

# Save aggregated metrics to CSV
aggregate_metrics_csv_path = '/content/18Jan_GBRaggregate_fine_tuning_metrics.csv'
aggregate_metrics_df.to_csv(aggregate_metrics_csv_path, index=False)
print(f"Aggregate fine-tuning metrics saved to {aggregate_metrics_csv_path}")

# Save forecast summaries to CSV
forecast_summary_csv_path = '/content/18Jan_GBRforecast_summary.csv'
forecast_summary_df.to_csv(forecast_summary_csv_path, index=False)
print(f"Forecast summaries saved to {forecast_summary_csv_path}")

# Display the DataFrames
print("\nAggregate Fine-Tuning Metrics:")
print(aggregate_metrics_df)

print("\nForecast Summary:")
print(forecast_summary_df)

# Initialize a list to store aggregate metrics
aggregate_metrics = []

# Get unique districts from the results
# Changed 'results_df' to 'metrics_df'
unique_districts = metrics_df['District'].unique()

# Iterate through each district
for district in unique_districts:
    # Filter results for the current district
    # Changed 'results_df' to 'metrics_df'
    district_data = metrics_df[metrics_df['District'] == district]

    # Calculate average metrics for the district
    avg_rmse = district_data['MSE'].mean()  # Assuming 'MSE' is the desired metric
    avg_r2 = district_data['R²'].mean()

    # Append aggregate metrics for the district
    aggregate_metrics.append({
        'District': district,
        'Avg RMSE': avg_rmse,  # Or any other relevant metric
        'Avg R²': avg_r2
    })

# Convert aggregate metrics to a DataFrame
aggregate_metrics_df = pd.DataFrame(aggregate_metrics)

# Save to CSV if needed
aggregate_metrics_csv_path = '/content/18Jan_GBRaggregate_performance_metrics.csv'
aggregate_metrics_df.to_csv(aggregate_metrics_csv_path, index=False)

print(f"Aggregate performance metrics saved to {aggregate_metrics_csv_path}")

# Display the DataFrame
aggregate_metrics_df

# Print the best hyperparameters for each district and target column
for result in fine_tuning_results:
    district = result['District']
```

```
    target_column = result['Target Column']
    best_params = result['Best Parameters']
    print(f"District: {district}, Target Column: {target_column}, Best Hyperparameters: {best_par
```

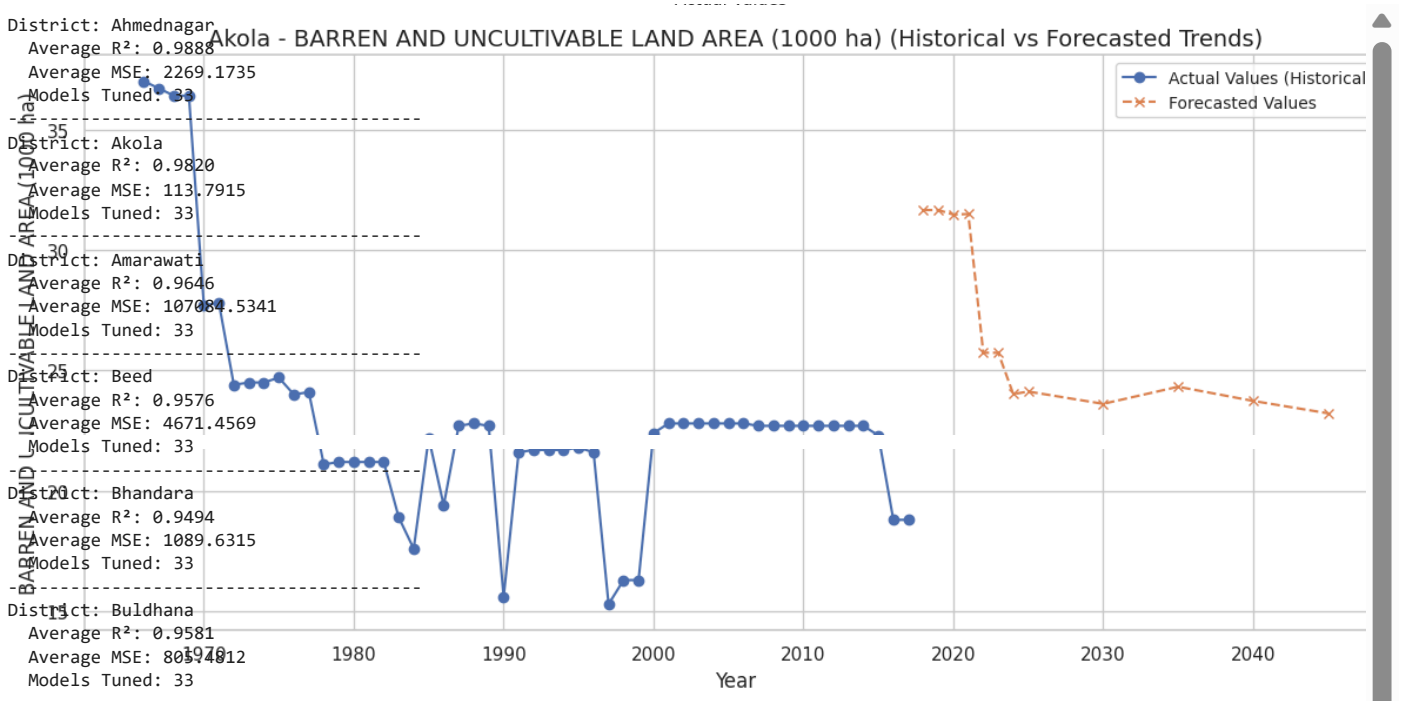
```
Aggregate fine-tuning metrics saved to /content/18Jan_GBRaggregate_fine_tuning_metrics.csv
Forecast summaries saved to /content/18Jan_GBRforecast_summary.csv

Aggregate Fine-Tuning Metrics:
    District        Avg MSE    Avg R²
0  Ahmednagar    2269.173476  0.988767
1      Akola     113.791505  0.982003
2   Amarawati  107084.534111  0.964628
3       Beed    4671.456940  0.957603
4   Bhandara    1089.631468  0.949432
5   Buldhana     805.481158  0.958082
6  Chandrapur   22443.733860  0.967962
7      Dhule    1148.555894  0.958727
8   Jalgaon    8810.035055  0.966334
9   Kolhapur      89.722592  0.987914
10    Nagpur     379.005143  0.972589
11    Nanded    4468.601712  0.963632
12     Nasik     303.288083  0.972270
13  Osmanabad   1276.692635  0.975868
14   Parbhani    8893.863365  0.954338
15       Pune    2945.968664  0.945618
16     Sangli     524.296528  0.962418
17     Satara     582.197428  0.960534
18    Solapur     795.080075  0.954450
19    Yeotmal     533.259519  0.974896

Forecast Summary:
    District  Year  Avg Forecasted Value
0  Ahmednagar  2018            916.395241
1  Ahmednagar  2019           1311.365420
2  Ahmednagar  2020           1099.027618
3  Ahmednagar  2021           1493.120653
4  Ahmednagar  2022           1542.256789
..        ...   ...                   ...
235    Yeotmal  2025            973.611016
236    Yeotmal  2030           1416.111491
237    Yeotmal  2035           1153.570545
238    Yeotmal  2040           1345.208055
239    Yeotmal  2045           1419.083337

[240 rows x 3 columns]
Aggregate performance metrics saved to /content/18Jan_GBRaggregate_performance_metrics.csv
District: Ahmednagar, Target Column: Area, Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 300, 'min_samples_split': 2, 'm
District: Ahmednagar, Target Column: Production, Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'min_samples_split':
District: Ahmednagar, Target Column: Yield, Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100, 'min_samples_split': 5, '
District: Ahmednagar, Target Column: Irrigated Area, Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 200, 'min_samples_spl
District: Ahmednagar, Target Column: Annual Rainfall, Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 200, 'min_samples_sp
District: Ahmednagar, Target Column: NITROGEN CONSUMPTION (tons), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 300, 'mi
District: Ahmednagar, Target Column: NITROGEN SHARE IN NPK (Percent), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 200,
District: Ahmednagar, Target Column: NITROGEN PER HA OF NCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 3
District: Ahmednagar, Target Column: NITROGEN PER HA OF GCA (Kg per ha), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 3
District: Ahmednagar, Target Column: PHOSPHATE CONSUMPTION (tons), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 100, 'm
District: Ahmednagar, Target Column: PHOSPHATE SHARE IN NPK (Percent), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 100
District: Ahmednagar, Target Column: PHOSPHATE PER HA OF NCA (Kg per ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimators':
District: Ahmednagar, Target Column: PHOSPHATE PER HA OF GCA (Kg per ha), Best Hyperparameters: {'subsample': 0.8, 'n_estimators':
District: Ahmednagar, Target Column: POTASH CONSUMPTION (tons), Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 200, 'min_
District: Ahmednagar, Target Column: POTASH SHARE IN NPK (Percent), Best Hyperparameters: {'subsample': 1.0, 'n_estimators': 100, '
```

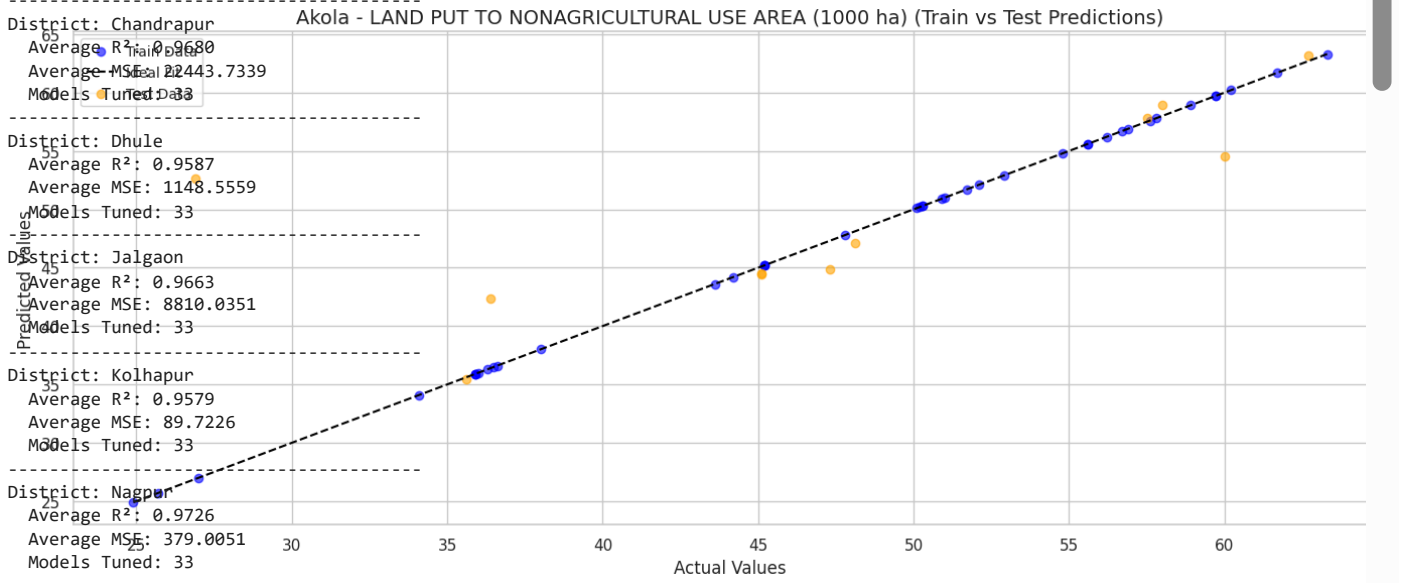


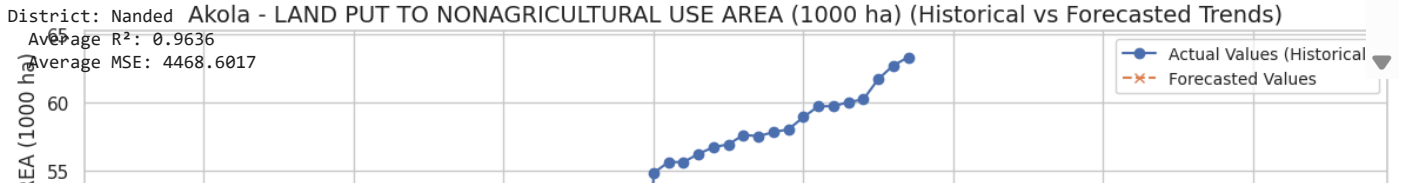

```
# Calculate and print aggregate metrics for each district
aggregate_metrics = []

for district in districts:
    district_results = [result for result in fine_tuning_results if result['District'] == distric

    if district_results:
        avg_r2 = np.mean([res['R²'] for res in district_results])
        avg_mse = np.mean([res['MSE'] for res in district_results])
        count = len(district_results)

        aggregate_metrics.append({
            'District': district,
```

```
            'Average R²': avg_r2,
            'Average MSE': avg_mse,
            'Models Tuned': count
        })

        print(f"District: {district}")
        print(f"  Average R²: {avg_r2:.4f}")
        print(f"  Average MSE: {avg_mse:.4f}")
        print(f"  Models Tuned: {count}")
        print("-" * 40)

# Convert aggregate metrics to DataFrame for further use if needed
aggregate_metrics_df = pd.DataFrame(aggregate_metrics)
```

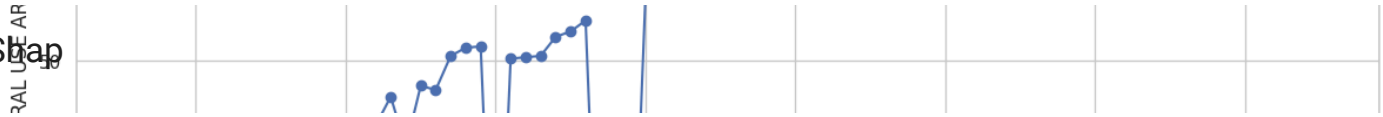
```
District: Ahmednagar
  Average R²: 0.9888
  Average MSE: 2269.1735
  Models Tuned: 33
----------------------------------------
District: Akola
  Average R²: 0.9820
  Average MSE: 113.7915
  Models Tuned: 33
----------------------------------------
District: Amarawati
  Average R²: 0.9646
  Average MSE: 107084.5341
  Models Tuned: 33
----------------------------------------
District: Beed
  Average R²: 0.9576
  Average MSE: 4671.4569
  Models Tuned: 33
----------------------------------------
District: Bhandara
  Average R²: 0.9494
  Average MSE: 1089.6315
  Models Tuned: 33
----------------------------------------
District: Buldhana
  Average R²: 0.9581
  Average MSE: 8054.4812
  Models Tuned: 33
----------------------------------------
District: Chandrapur
  Average R²: 0.9680
  Average MSE: 22443.7339
  Models Tuned: 33
----------------------------------------
District: Dhule
  Average R²: 0.9587
  Average MSE: 1148.5559
  Models Tuned: 33
----------------------------------------
District: Jalgaon
  Average R²: 0.9663
  Average MSE: 8810.0351
  Models Tuned: 33
----------------------------------------
District: Kolhapur
  Average R²: 0.9579
  Average MSE: 89.7226
  Models Tuned: 33
----------------------------------------
District: Nagpur
  Average R²: 0.9726
  Average MSE: 379.0051
  Models Tuned: 33
----------------------------------------
District: Nanded
  Average R²: 0.9636
  Average MSE: 4468.6017
```







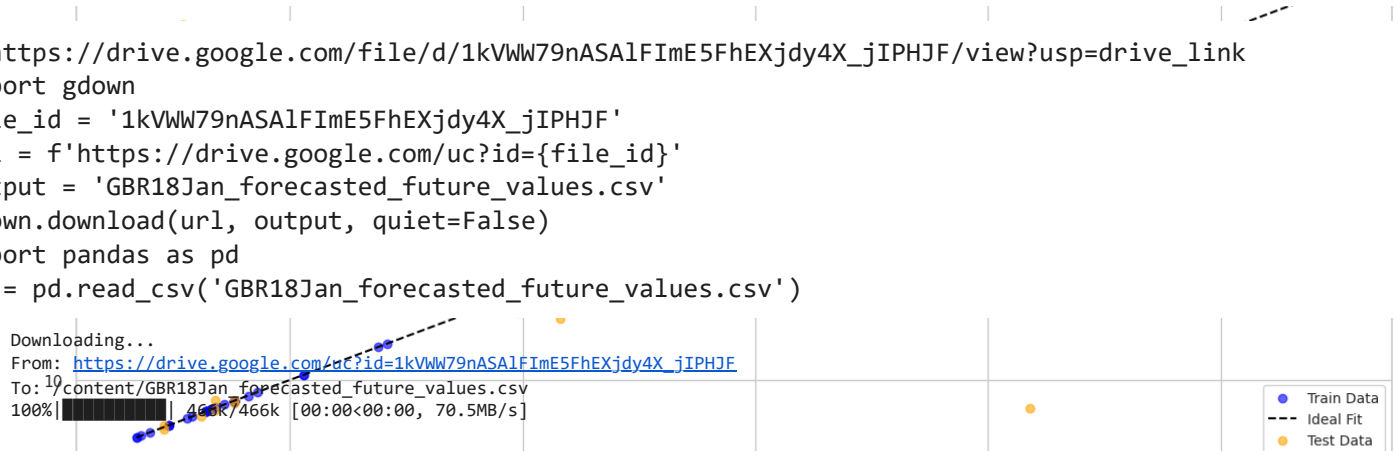
Shap



```
import gdown
file_id = '1KXMl4EVNofFnGsypt1UpZ4GJL6pGLJCe'
url = f'https://drive.google.com/uc?id={file_id}'
output = 'maharashtra_aggregate_1966_2017.csv'
gdown.download(url, output, quiet=False)
import pandas as pd
df = pd.read_csv('maharashtra_aggregate_1966_2017.csv')
```


```
Downloading...
From: https://drive.google.com/uc?id=1KXMl4EVNofFnGsypt1UpZ4GJL6pGLJCe
To: /content/maharashtra_aggregate_1966_2017.csv
100%|██████████| 277k/277k [00:00<00:00, 57.6MB/s]
```

Start coding or generate with AI.



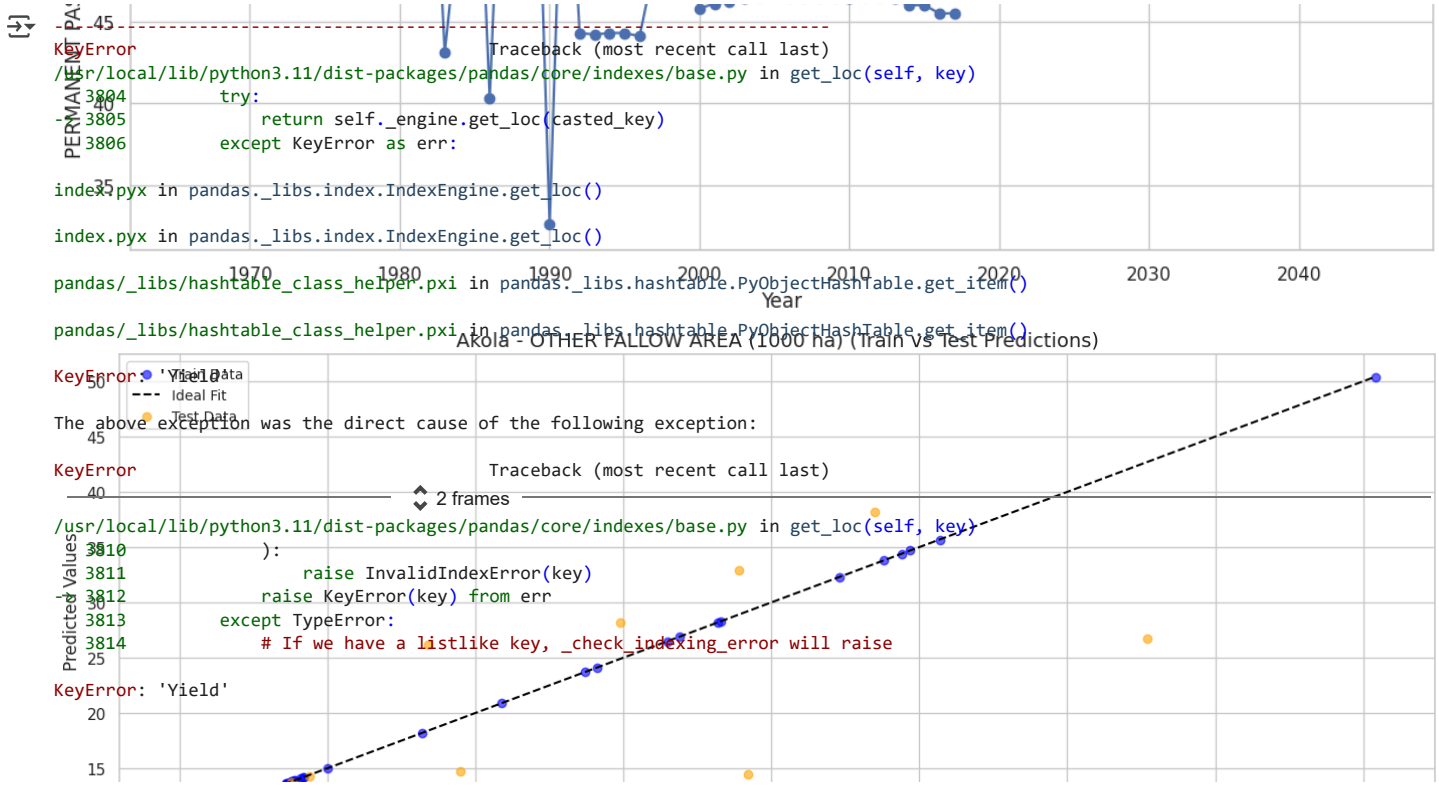

```
# https://drive.google.com/file/d/1kVWW79nASAlFImE5FhEXjdy4X_jIPHJF/view?usp=drive_link
import gdown
file_id = '1kVWW79nASAlFImE5FhEXjdy4X_jIPHJF'
url = f'https://drive.google.com/uc?id={file_id}'
output = 'GBR18Jan_forecasted_future_values.csv'
gdown.download(url, output, quiet=False)
import pandas as pd
df = pd.read_csv('GBR18Jan_forecasted_future_values.csv')
```

```
Downloading...
From: https://drive.google.com/uc?id=1kVWW79nASAlFImE5FhEXjdy4X_jIPHJF
To: /content/GBR18Jan_forecasted_future_values.csv
100%|██████████| 466k/466k [00:00<00:00, 70.5MB/s]
```

```
import pandas as pd
import shap
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# Load dataset
data_path = "GBR18Jan_forecasted_future_values.csv"
df = pd.read_csv(data_path)

# Define target and features
targets = ['Yield', 'Production']  # Specify your target columns
features = [col for col in df.columns if col not in targets]  # Exclude target columns

# Split data
X = df[features]
y_yield = df['Yield']
y_production = df['Production']

X_train, X_test, y_train_yield, y_test_yield = train_test_split(X, y_yield, test_size=0.2, random
X_train, X_test, y_train_production, y_test_production = train_test_split(X, y_production, test_s

# Train RandomForestRegressor for Yield
model_yield = RandomForestRegressor(n_estimators=100, random_state=42)
model_yield.fit(X_train, y_train_yield)
```

```
# Train RandomForestRegressor for Production
model_production = RandomForestRegressor(n_estimators=100, random_state=42)
model_production.fit(X_train, y_train_production)

# Apply SHAP for Yield
explainer_yield = shap.Explainer(model_yield, X_train)
shap_values_yield = explainer_yield(X_test)

# Apply SHAP for Production
explainer_production = shap.Explainer(model_production, X_train)
shap_values_production = explainer_production(X_test)

# Plot SHAP summary for Yield
shap.summary_plot(shap_values_yield, X_test, plot_type="bar", show=False)
plt.title("Feature Importance for Yield")
plt.show()

# Plot SHAP summary for Production
shap.summary_plot(shap_values_production, X_test, plot_type="bar", show=False)
plt.title("Feature Importance for Production")
plt.show()
```

```
[4]
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Yield'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
                                          2 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'Yield'
```




```
import pandas as pd
import shap
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# Load dataset
data_path = "GBR18Jan_forecasted_future_values.csv"
df = pd.read_csv(data_path)

# Print column names to verify their exact names
```

```
print("Dataset Columns:", df.columns)

# Define target and features
actual_targets = [col for col in df.columns if 'Yield' in col or 'Production' in col]  # Find clo
targets = actual_targets if actual_targets else ['Yield', 'Production']  # Default if not found

if not set(targets).issubset(df.columns):
    raise KeyError(f"Columns {targets} not found in dataset. Available columns: {df.columns}")
```