Final Project Report Template

1. Introduction

   1.1. Project overview: Blueberry Yield Prediction

   Our project aims to predict blueberry yields using machine learning models to optimize agricultural practices. We start by gathering comprehensive datasets from Kaggle. After preprocessing, which includes cleaning, feature engineering, and scaling, we conduct exploratory data analysis to understand relationships and distributions within the data. Four regression models—Linear Regression, Decision Tree Regressor, Random Forest Regressor, and XGB Regressor—are implemented and fine-tuned to predict blueberry yields. Evaluation metrics like R-squared are used to compare model performance. The best-performing model, which is XGB regressor, is deployed as an API for scalability, integrating it with Flask for web service. Our project not only aims to provide accurate yield predictions but also serves to optimize farming decisions, benefiting agricultural stakeholders with data-driven insights.

   1.2. Objectives

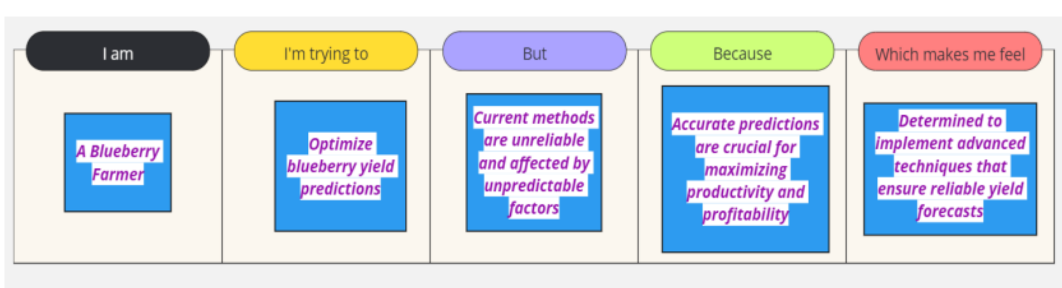
   The main objectives of our project are as follows:

   - Use machine learning to predict blueberry yields accurately, aiming to optimize agricultural practices such as irrigation, fertilization, and pest control.
   - Provide farmers with data-driven insights to make informed decisions on resource allocation and crop management strategies.
   - Develop models that can reliably forecast blueberry yields based on environmental factors and historical data, reducing uncertainty in crop outcomes.
   - Deploy a scalable machine learning model as an API to integrate seamlessly with existing agricultural systems, ensuring accessibility and usability for stakeholders.

2. Project Initialization and Planning Phase

   2.1. Define Problem Statement

   Accurately predicting blueberry yields is essential for optimizing farming practices amidst unpredictable weather, soil conditions, and pests. Farmers often struggle with these uncertainties, impacting productivity and profitability. Our project aims to develop precise machine learning models using historical yield data, weather patterns, soil characteristics, and pest incidence rates. By providing farmers with reliable yield forecasts, we empower them to make informed decisions, allocate resources efficiently, and enhance overall productivity while mitigating environmental risks.



2.2. Project Proposal (Proposed Solution)

We utilized four different machine learning models, namely linear regression, random forest regressor, decision tree regressor and XGB regressor to train our predictive model for blueberry yield. Out of the four models used, we found that XGB regressor has the best outcome and we have proceeded with that to create a front end.

2.3. Initial Project Planning

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
Sprint-1	Data Collection and Preprocessing	BYPS-9	Loading the dataset	6	High	Sona Subramanian	4th June 2024	5th June 2024
Sprint-1	Data Collection and Preprocessing	BYPS-10	Preprocessing the Data	8	High	Sona Subramanian	5th June 2024	12th June 2024
Sprint-2	Model Building	BYPS-11	Training Model using Multiple Algorithms	8	High	Sai Veshwa B	13th June 2024	26th June 2024
Sprint-2	Model Building	BYPS-12	Testing the Model	9	High	Sai Veshwa B	27th June 2024	1st July 2024
Sprint-3	Performance Testing and Hyperparameter Tuning	BYPS-13	Hyperparameter Tuning	8	Medium	Kumar Abhishek	2nd July 2024	5th July 2024
Sprint-3	Performance Testing and Hyperparameter Tuning	BYPS-14	Testing model with evaluation metrics	7	High	Kumar Abhishek	6th July 2024	8th July 2024
Sprint-4	Model Deployment	BYPS-15	Saving the best model	5	High	Dinari Ajay Kumar	9th July 2024	11th July 2024
Sprint-4	Model Deployment	BYPS-16	Integrating with web framework	9	High	Dinari Ajay Kumar	11th July 2024	12th July 2024

3. Data Collection and Preprocessing Phase

3.1. Data Collection Plan and Raw Data Sources Identified

(i)Data Collection Plan

Section	Description
Project Overview	This project focuses on developing an ML solution to predict blueberry yields accurately. It involves collecting and analyzing historical data on blueberry yields. Four different machine learning models were trained and evaluated, with linear regression identified as the most effective. The goal is to provide blueberry farmers with reliable predictions to optimize farming practices, enhance productivity, and support sustainable agriculture.
Data Collection Plan	Data will be collected from a vast platform on Internet named Kaggle.
Raw Data Sources Identified	The data has been collected over a period of 30 years from a wild blueberry plantation in Maine, which is situated in The United States of America.

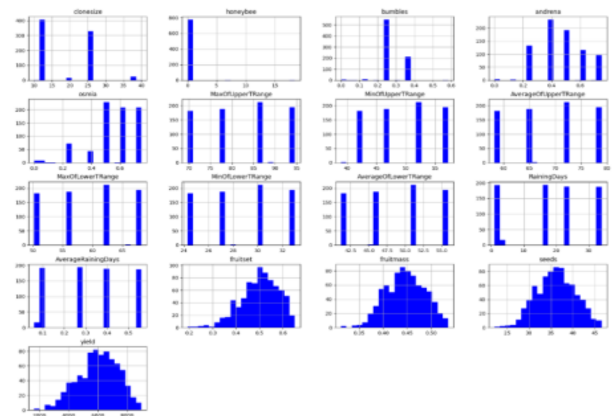
(ii)Raw Data Sources Identified

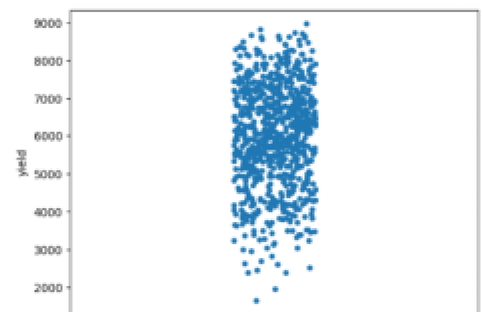
Source Name	Description	Location/URL	Format	Size	Access Permissions
Dataset 1	The dataset being used to train the model in our project is **WildBlueberryPollinationSimulationData.csv** . The data here is experimental and it was collected in Maine, USA during the last 30 years.	https://www.kaggle.com/datasets/saurabhshahane/wild-blueberry-yield-prediction	CSV	85.26kB	Public

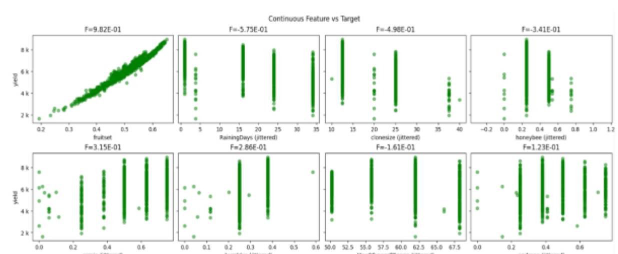
3.2. Data Quality Report

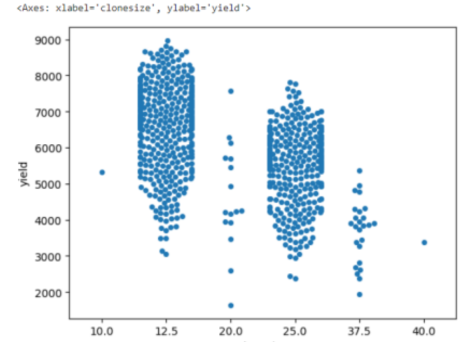
Data Source	Data Quality Issue	Severity	Resolution Plan
Kaggle Dataset	Outliers present in the dataset	Moderate	z-scores are used to standardize the data and effectively detect and remove extreme values that may distort model performance.
Kaggle Dataset	Redundant columns in the dataset	Moderate	Through data visualizations like bar graphs, histograms and correlation heatmaps, columns with high correlation are recognized and removed.

3.3. Data Exploration and Preprocessing

<table>
<tr><th>Section</th><th>Description</th></tr>
<tr><td>Data Overview</td><td>
<table>
<tr><th></th><th>clonesize</th><th>honeybee</th><th>bumbles</th><th>andrena</th><th>osmia</th><th>MaxOfUpperTRange</th><th>MinOfUpperTRange</th><th>AverageOfUpperTRange</th><th>MaxOfLowerTRange</th></tr>
<tr><td>count</td><td>777.000000</td><td>777.000000</td><td>777.000000</td><td>777.000000</td><td>777.000000</td><td>777.000000</td><td>777.000000</td><td>777.000000</td><td>777.000000</td></tr>
<tr><td>mean</td><td>18.767696</td><td>0.417133</td><td>0.282389</td><td>0.468817</td><td>0.562062</td><td>82.277091</td><td>49.700515</td><td>68.723037</td><td>59.309395</td></tr>
<tr><td>std</td><td>6.999063</td><td>0.978904</td><td>0.066343</td><td>0.161052</td><td>0.169119</td><td>9.193745</td><td>5.595769</td><td>7.676984</td><td>6.647760</td></tr>
<tr><td>min</td><td>10.000000</td><td>0.000000</td><td>0.000000</td><td>0.000000</td><td>0.000000</td><td>69.700000</td><td>39.000000</td><td>58.200000</td><td>50.200000</td></tr>
<tr><td>25%</td><td>12.500000</td><td>0.250000</td><td>0.250000</td><td>0.380000</td><td>0.500000</td><td>77.400000</td><td>46.800000</td><td>64.700000</td><td>55.800000</td></tr>
<tr><td>50%</td><td>12.500000</td><td>0.250000</td><td>0.250000</td><td>0.500000</td><td>0.630000</td><td>86.000000</td><td>52.000000</td><td>71.900000</td><td>62.000000</td></tr>
<tr><td>75%</td><td>25.000000</td><td>0.500000</td><td>0.380000</td><td>0.630000</td><td>0.750000</td><td>89.000000</td><td>52.000000</td><td>71.900000</td><td>66.000000</td></tr>
<tr><td>max</td><td>40.000000</td><td>18.430000</td><td>0.585000</td><td>0.750000</td><td>0.750000</td><td>94.600000</td><td>57.200000</td><td>79.000000</td><td>68.200000</td></tr>
</table>
</td></tr>
<tr><td>Univariate Analysis</td><td>


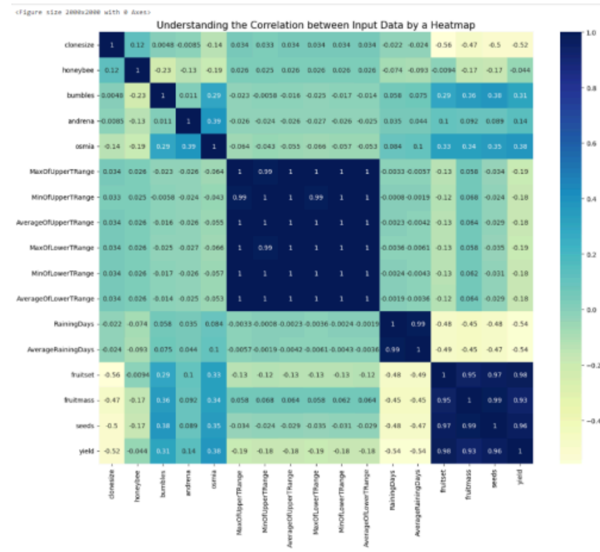


</td></tr>
<tr><td>Bivariate Analysis</td><td>


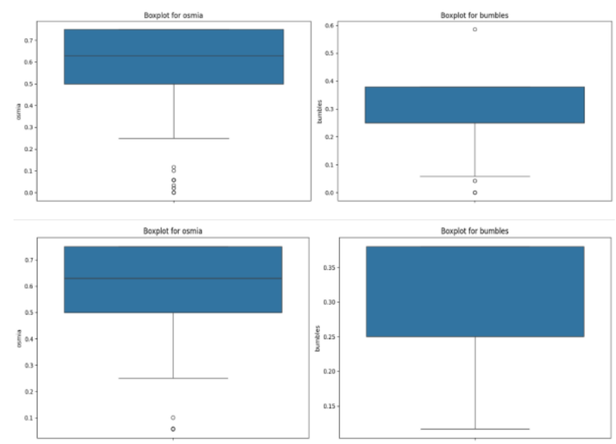
<Axes: xlabel='clonesize', ylabel='yield'>


</td></tr>
</table>

Multivariate Analysis



Outliers and Anomalies



Loading Data

```
df = pd.read_csv('WildBlueberryPollinationSimulationData (1).csv')
df
```

	Row#	clonesize	honeybee	bumbles	andrena	osmia	MaxOfUpperTRange	MinOfUpperTRange	AverageOfUpperTRange	MaxOfLowerTRange	MinOfLowerTRa
0	0	37.5	0.750	0.250	0.250	0.250	86.0	52.0	71.9	62.0	
1	1	37.5	0.750	0.250	0.250	0.250	86.0	52.0	71.9	62.0	
2	2	37.5	0.750	0.250	0.250	0.250	94.6	57.2	79.0	68.2	
3	3	37.5	0.750	0.250	0.250	0.250	94.6	57.2	79.0	68.2	
4	4	37.5	0.750	0.250	0.250	0.250	86.0	52.0	71.9	62.0	
...	...	...	...	...	...	...	...	...	...	...	
772	772	10.0	0.537	0.117	0.409	0.058	86.0	52.0	71.9	62.0	
773	773	40.0	0.537	0.117	0.409	0.058	86.0	52.0	71.9	62.0	
774	774	20.0	0.537	0.117	0.409	0.058	86.0	52.0	71.9	62.0	

Handling Missing Data

No of missing values in the dataset:

```
df.isna().sum()
```

```
clonesize               0
honeybee                0
bumbles                 0
andrena                 0
osmia                   0
MaxOfUpperTRange        0
MinOfUpperTRange        0
AverageOfUpperTRange    0
MaxOfLowerTRange        0
MinOfLowerTRange        0
AverageOfLowerTRange    0
RainingDays             0
AverageRainingDays      0
fruitset                0
fruitmass               0
seeds                   0
yield                   0
dtype: int64
```

<table>
<tr>
<td>Data Transformation</td>
<td>Removing outliers –

```
from scipy import stats
# Removing outliers
z_scores = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))
threshold = 3
outliers = (z_scores > threshold).any(axis=1)
new_df = df[~outliers]
num_outliers_removed = outliers.sum()
print(f"Number of outliers removed: {num_outliers_removed}")
```

```
Number of outliers removed: 13
```
</td>
</tr>
<tr>
<td>Feature Engineering</td>
<td>Removing unwanted columns after visualizing the dataset –

```
new_df = new_df.drop(columns=['bumbles','fruitmass','AverageRainingDays','fruitset','MaxOfUpperTRange','MaxOfLowerTRange', 'MinOfLowerTRange'])
new_df.head()

# ALL THE ABOVE COLUMNS HAVE HIGH CORRELATION WITH OTHER COLUMNS, SO THEY ARE BEING REMOVED
```

	clonesize	honeybee	andrena	osmia	MinOfUpperTRange	AverageOfUpperTRange	AverageOfLowerTRange	RainingDays	seeds	yield
0	37.5	0.75	0.25	0.25	52.0	71.9	50.8	16.0	31.678898	3813.165795
1	37.5	0.75	0.25	0.25	52.0	71.9	50.8	1.0	33.449385	4947.605663
2	37.5	0.75	0.25	0.25	57.2	79.0	55.9	16.0	30.546306	3866.798965
3	37.5	0.75	0.25	0.25	57.2	79.0	55.9	1.0	31.562586	4303.943030
4	37.5	0.75	0.25	0.25	52.0	71.9	50.8	24.0	28.873714	3436.493543
</td>
</tr>
<tr>
<td>Save Processed Data</td>
<td>–</td>
</tr>
</table>

4. Model Development Phase

4.1. Feature Selection Report

This report outlines the process and rationale behind the selection and removal of features for the Wild Blueberry Pollination Simulation dataset. The goal of feature selection is to improve model performance and interpretability by focusing on the most impactful variables. Through data visualization and analysis, we identified key features that significantly contribute to predicting the target outcomes while removing redundant or non-informative variables. The following sections detail the reasons for selecting or removing each feature, ensuring a robust and efficient model.

Feature	Description	Selected (Yes/No)	Reasoning
Row#	Unique identifier for each row of the dataset	No	For predicting the yield value, Row number is not required.
clonesize	The size of the blueberry clone being studied.	Yes	Influences yield and fruit production due to varying pollination dynamics in different clone sizes.

honeybee	The density or activity level of honey bee pollinators.	Yes	Key pollinator affecting yield and seed production.
bumbles	The density or activity level of bumblebee pollinators.	No	Likely redundant with other bee density variables or showed little correlation with the target variable.
andrena	The density or activity level of Andrena bee pollinators.	Yes	Significant pollinator, adding diversity in understanding pollination effects.
osmia	The density or activity level of Osmia bee pollinators.	Yes	Important pollinator, contributing to overall pollination success.
MaxOfUpperTRange	The maximum temperature of the upper temperature range recorded.	No	Little variation or weak correlation with outcomes; redundant with other temperature measures.
MinOfUpperTRange	The minimum temperature of the upper temperature range recorded.	Yes	Influences pollinator activity and plant growth under extreme temperatures.
AverageOfUpperTRange	The average temperature of the upper temperature range.	Yes	Impacts pollinator behavior and plant physiology under consistent temperatures.
MaxOfLowerTRange	The maximum temperature of the lower temperature range recorded.	No	Limited variation or significance; possibly redundant with other temperature features.

MinOfLowerTRange	The minimum temperature of the lower temperature range recorded.	No	Showed little correlation with outcomes; redundancy with other temperature variables.
AverageOfLowerTRange	The average temperature of the lower temperature range.	Yes	Important for understanding cooler conditions affecting plant stress and pollination.
RainingDays	The total number of raining days.	Yes	Affects pollinator activity and plant health; significant for crop success.
AverageRainingDays	The average number of raining days.	No	Non-informative or redundant with total raining days (RainingDays).
fruitset	The proportion of flowers that set fruit.	No	Redundant with other yield-related measures.
fruitmass	The mass of the fruit produced.	No	Highly correlated with yield, making it redundant.
seeds	The number of seeds per fruit.	Yes	Direct measure of successful pollination and fruit development.

4.2. Model Selection Report

In the forthcoming Model Selection Report, various models will be outlined, detailing their descriptions, hyperparameters, and performance metrics, including MSE, MAE or R2 score. This comprehensive report will provide insights into the chosen models and their effectiveness

Model	Description	Hyperparameters	Performance Metric (e.g., MSE, MAE, R2 Score)
Linear Regression	A simple model that assumes a linear relationship between the input variables (features) and the output variable (target). It fits a straight line to the data.	-	R2 score = 0.985
Random Forest Regressor	An ensemble learning method that constructs multiple decision trees during training and outputs the average prediction of the individual trees. It reduces overfitting and improves accuracy	-	R2 score = 0.979
Decision Tree Regressor	A non-linear model that splits the data into subsets based on feature values, creating a tree-like structure. Each leaf represents a predicted outcome.	-	R2 score = 0.961
XGB Regressor	An advanced ensemble technique that uses gradient boosting on decision trees. It iteratively improves model performance by minimizing errors of previous models, known for its high accuracy and efficiency.	-	R2 score = 0.982

4.3. Initial Model Training Code, Model Validation and Evaluation Report

This document outlines the process and results of developing and validating regression models to predict the outcomes for the Wild Blueberry Pollination Simulation dataset. The goal of this phase is to build accurate and reliable models by selecting relevant features, evaluating various algorithms, and fine-tuning hyperparameters. This report includes detailed evaluation metrics, visualization, and insights into the performance of each model, ensuring a comprehensive understanding of their predictive capabilities.

Initial Model Training Code:

```
# Importing and building Linear Regression model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
model.score(X_test, y_test)

y_preds = model.predict(X_test)

print("Regression metrics on the test set")
print(f"R2 score: {r2_score(y_test, y_preds)}")
print(f"MAE: {mean_absolute_error(y_test,y_preds)}")
print(f"MSE: {mean_squared_error(y_test, y_preds)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_preds))}")
```

```
# Importing and building the Random forest regressor model
from sklearn.ensemble import RandomForestRegressor

model2 = RandomForestRegressor()
model2.fit(X_train,y_train)
```

```python
# Importing and building Decision tree regressor model
from sklearn.tree import DecisionTreeRegressor

model3 = DecisionTreeRegressor()
model3.fit(X_train,y_train)
y_preds3 = model3.predict(X_test)

print("Regression metrics on the test set")
print(f"R2 score: {r2_score(y_test, y_preds3)}")
print(f"MAE: {mean_absolute_error(y_test,y_preds3)}")
print(f"MSE: {mean_squared_error(y_test, y_preds3)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_preds3))}")
```

```python
# Importing and building XGB Regressor model
from xgboost import XGBRegressor
model4 = XGBRegressor()
model4.fit(X_train,y_train)
model4.score(X_test,y_test)

y_preds4 = model4.predict(X_test)

print("Regression metrics on the test set")
print(f"R2 score: {r2_score(y_test, y_preds4)}")
print(f"MAE: {mean_absolute_error(y_test,y_preds4)}")
print(f"MSE: {mean_squared_error(y_test, y_preds4)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_preds4))}")
```

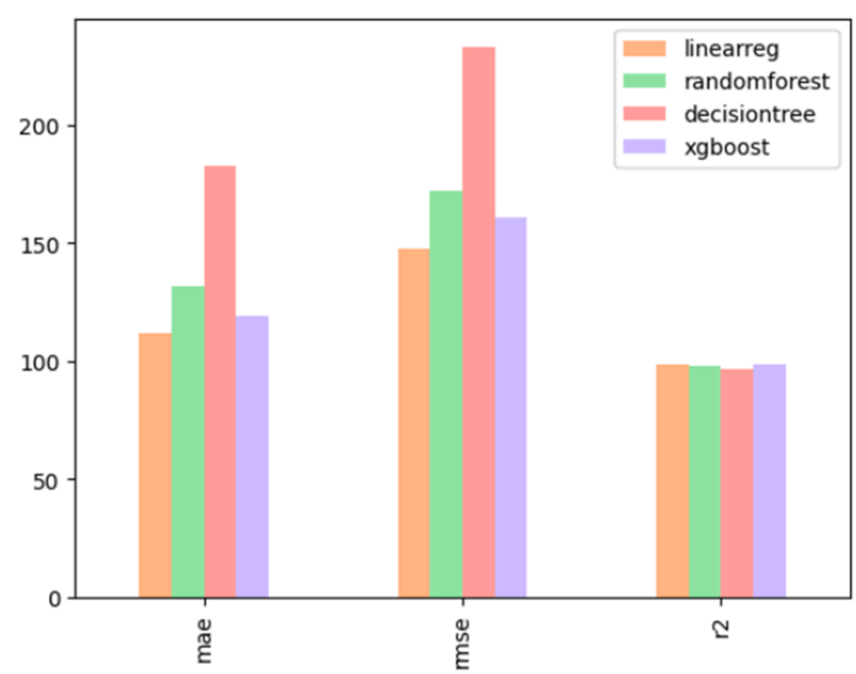
Model Validation and Evaluation Report:

Model	Mean Absolute Error (MAE)	Mean Squared Error (MSE)	R2 Score	Screenshorts of the evaluation report

Linear Regression	111.5204986497179	21684.627147497344	0.9850297685259484	```print("Regression metrics on the test set")```
				```print(f"R2 score: {r2_score(y_test, y_preds)}")```
				```print(f"MAE: {mean_absolute_error(y_test,y_preds)}")```
				```print(f"MSE: {mean_squared_error(y_test, y_preds)}")```
				```print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_preds))}")```
				Regression metrics on the test set
				R2 score: 0.9850297685259484
				MAE: 111.5204986497179
				MSE: 21684.627147497344
				RMSE: 147.25701052071287

Random Forest Regressor	133.300591 27843145	30238.3785 84205282	0.97912458 79522628	print("Regression metrics on the test set") print(f"R2 score: {r2_score(y_test, y_preds2)}") print(f"MAE: {mean_absolute_error(y_test,y_preds2)}") print(f"MSE: {mean_squared_error(y_test, y_preds2)}") print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_preds2))}") Regression metrics on the test set R2 score: 0.9791245879522628 MAE: 133.30059127843145 MSE: 30238.378584205282 RMSE: 173.8918588784572
Decision Tree Regressor	177.927503 29411768	51633.5104 3756918	0.96435421 28803687	print("Regression metrics on the test set") print(f"R2 score: {r2_score(y_test, y_preds3)}") print(f"MAE: {mean_absolute_error(y_test,y_preds3)}") print(f"MSE: {mean_squared_error(y_test, y_preds3)}") print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_preds3))}") Regression metrics on the test set R2 score: 0.9643542128803687 MAE: 177.92750329411768 MSE: 51633.51043756918 RMSE: 227.23008259816564
XGB Regressor	119.196047 83486517	25823.1611 81161377	0.98217268 4010463	print("Regression metrics on the test set") print(f"R2 score: {r2_score(y_test, y_preds4)}") print(f"MAE: {mean_absolute_error(y_test,y_preds4)}") print(f"MSE: {mean_squared_error(y_test, y_preds4)}") print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_preds4))}") Regression metrics on the test set R2 score: 0.982172684010463 MAE: 119.19604783486517 MSE: 25823.161181161377 RMSE: 160.69586547625107

This bar chart compares the performance of four regression models—Linear Regression, Random Forest, Decision Tree, and XGBoost—using three evaluation metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (R^2).



5. Model Optimization and Tuning Phase

5.1. Hyperparameter Tuning Documentation

Model	Tuned Hyperparameters	Optimal Values
Linear Regression	-	Regression metrics on the test set R2 score: 0.9850297685259484 MAE: 111.5204986497179 MSE: 21684.627147497344 RMSE: 147.25701052071287

Random Forest Regressor	from sklearn.model_selection import GridSearchCV
	param_grid = {
	'n_estimators': [100, 200, 300],
	'max_features': ['auto', 'sqrt', 'log2'],
	'max_depth': [10, 20, 30, None],
	'min_samples_split': [2, 5, 10],
	'min_samples_leaf': [1, 2, 4],
	'bootstrap': [True, False]
	}
	
	rf = RandomForestRegressor()
	grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
	cv=3, n_jobs=-1, verbose=2)
		Fitting 3 folds for each of 648 candidates, totalling 1944 fits
		Best Parameters: {'bootstrap': False, 'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
		Mean Squared Error: 23036.512805030303
		R-squared: 0.9840964787311984

DecisionTree Regressor	param_grid = {
	'max_depth': [None, 10, 20, 30],
	'min_samples_split': [2, 5, 10],
	'min_samples_leaf': [1, 2, 4],
	'max_features': [None, 'auto', 'sqrt', 'log2']
	}
	
	dt = DecisionTreeRegressor()
	grid_search = GridSearchCV(estimator=dt, param_grid=param_grid,
	cv=3, n_jobs=-1, verbose=2)
		Fitting 3 folds for each of 144 candidates, totalling 432 fits
		Best Parameters: {'max_depth': 30, 'max_features': None, 'min_samples_leaf': 4, 'min_samples_split': 2}
		Mean Squared Error: 39945.42992962877
		R-squared: 0.9724232135369657

XGB Regressor	param_grid = {
	'n_estimators': [100, 200, 300],
	'learning_rate': [0.01, 0.1, 0.2],
	'max_depth': [3, 6, 9],
	'subsample': [0.6, 0.8, 1.0],
	'colsample_bytree': [0.6, 0.8, 1.0]
	}
	
	xgb = XGBRegressor()
	grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid,
	cv=3, n_jobs=-1, verbose=2)
		Fitting 3 folds for each of 243 candidates, totalling 729 fits
		Best Parameters: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200, 'subsample': 0.6}
		Mean Squared Error: 19206.509150671092
		R-squared: 0.9867405657547946

5.2. Performance Metrics Comparison Report

Model	Baseline Metric	Optimized Metric
Linear Regression	Regression metrics on the test set R2 score: 0.9850297685259484 MAE: 111.5204986497179 MSE: 21684.627147497344 RMSE: 147.25701052071287	-
Random Forest Regressor	Regression metrics on the test set R2 score: 0.9794220417457439 MAE: 132.1608756852944 MSE: 29807.511859370235 RMSE: 172.64852116183977	Regression metrics on the test set R2 score: 0.9840964787311984 MAE: 120.52941809457525 MSE: 23036.512805030303 RMSE: 151.7778402963697
Decision Tree Regressor	Regression metrics on the test set R2 score: 0.9621834493005946 MAE: 181.75958724836605 MSE: 54777.89727850463 RMSE: 234.04678437975736	Regression metrics on the test set R2 score: 0.9724232135369657 MAE: 155.86761325904146 MSE: 39945.42992962877 RMSE: 199.86352826273423
XGB Regressor	Regression metrics on the test set R2 score: 0.982172684010463 MAE: 119.19604783486517 MSE: 25823.161181161377 RMSE: 160.69586547625107	Regression metrics on the test set R2 score: 0.9867405657547946 MAE: 108.15756897906451 MSE: 19206.509150671092 RMSE: 138.5875504894689

5.3. Final Model Selection Justification

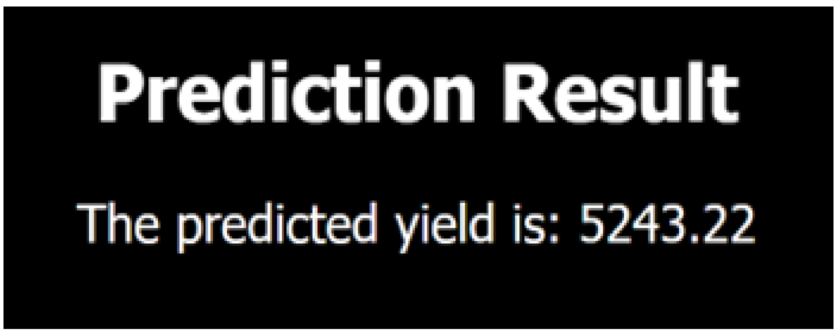
Final Model	Reasoning
XGB Regressor	The XGB Regressor model was selected for its superior performance, exhibiting the highest R2 score during hyperparameter tuning. Its ability to handle complex relationships, minimize overfitting, and optimize predictive accuracy aligns with project objectives, justifying its selection as the final model

6. Results

6.1. Output Screenshots

1) index.html <input template>

2) predict.html <output template>



7. Advantages & Disadvantages

Advantages:

Optimized Resource Management:

Predicting berry yield allows farmers to manage resources more efficiently, such as water, fertilizers, and labor, leading to cost savings and better crop management.

Enhanced Decision Making:

Accurate yield predictions can help farmers make informed decisions regarding market supply, pricing strategies, and harvesting schedules, maximizing profitability and reducing waste.

Disadvantages:

Data Dependency:

The accuracy of yield predictions heavily depends on the quality and quantity of available data. Incomplete or inaccurate data can lead to unreliable predictions, potentially harming the farmer's planning and decision-making processes.

Implementation Costs:

Developing and maintaining a sophisticated prediction model involves significant costs, including data collection, technology investments, and expert consultations. This can be a financial burden, especially for small-scale farmers.

8. Conclusion

In conclusion, berry yield prediction offers significant benefits in optimizing resource management and enhancing decision-making for farmers, leading to increased profitability and reduced waste. However, the reliance on high-quality data and the associated implementation costs pose challenges that need careful consideration. Balancing these factors is crucial for effectively leveraging prediction models in agricultural practices. Overall, the adoption of yield prediction technology holds great promise for advancing sustainable and efficient farming.

9. Future Scope

The future scope of berry yield prediction is vast and promising. Advances in machine learning and data analytics can enhance the accuracy and reliability of predictions, leading to smarter and more sustainable farming practices. Integration with Internet of Things (IoT) devices can provide real-time data and insights, enabling proactive management and timely interventions. Additionally, expanding these models to include predictions for various environmental and climatic conditions can help farmers adapt to changing weather patterns, further securing crop yields. As technology continues to evolve, yield prediction systems will become increasingly accessible and beneficial for farmers worldwide.

10. Appendix

10.1. Source Code

index.html:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Predict Wild Blueberry Yield</title>
    <style>
        body {
            background-image: url("D:/Flask/templates/blueberry.jpg"); /* Background image */
            background-size: cover; /* Cover the entire page */
            background-position: center; /* Center the image */
            font-family: Tahoma, sans-serif;
            display: flex;
            justify-content: flex-start; /* Align content to the left */
            align-items: center;
            height: 100vh;
            margin: 0;
            padding: 0;
```

```
            color: white; /* Changed text color to white */
        }

        .form-container {
            background-color: rgba(37, 37, 99, 0.5); /* Semi-transparent grey
background for the form container */
            padding: 30px;
            border-radius: 30px;
            box-shadow: 0 0 20px rgba(0, 0, 0, 0.1);
            width: 80%;
            max-width: 600px;
            overflow-y: auto;
            max-height: calc(100vh - 150px); /* Slightly reduced height */
            color: white; /* Changed text color to white */
            margin-left: 50px; /* Move form 50px to the left */
        }

        .form-container h1 {
            text-align: center;
            color: white; /* Heading color */
            font-size: 3rem; /* Increased font size */
            margin-bottom: 20px; /* Added some bottom margin */
        }

        .form-container label {
            display: block;
            margin-bottom: 12px;
            font-weight: bold;
            color: white; /* Label text color */
        }

        .form-container input[type="text"],
        .form-container input[type="date"],
        .form-container input[type="submit"] {
```

```
            width: calc(100% - 24px);
            height: 60px; /* Reduced input height */
            padding: 12px;
            margin-bottom: 15px;
            border: 4px solid white;
            border-radius: 10px;
            transition: all 0.3s ease;
            font-size: 1.2rem; /* Increased font size */
            box-sizing: border-box;
            background-color: rgba(0, 0, 0, 0.5); /* Darkened background color */
            color: white; /* Text color */
        }

        .form-container input[type="submit"] {
            background-color: #4CAF50;
            color: white;
            cursor: pointer;
            font-size: 1.3rem; /* Increased font size */
            font-weight: bold;
            border: none;
            border-radius: 10px;
            padding: 14px 24px; /* Increased padding */
        }

        .form-container input[type="submit"]:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <div class="form-container">
        <h1>BERRY_PREDICTOR.IO</h1>
        <form action="/predict" method="post">
```

```
            <label for="clonesize">Clonesize:</label>

            <input type="text" id="clonesize" name="clonesize"><br>

            <label for="honeybee">Honeybee:</label>

            <input type="text" id="honeybee" name="honeybee"><br>

            <label for="andrena">Andrena:</label>

            <input type="text" id="andrena" name="andrena"><br>

            <label for="osmia">Osmia:</label>

            <input type="text" id="osmia" name="osmia"><br>

            <label for="MinOfUpperTRange">Min Of Upper T Range:</label>

            <input type="text" id="MinOfUpperTRange" name="MinOfUpperTRange"><br>

            <label for="AverageOfUpperTRange">Average Of Upper T Range:</label>

            <input type="text" id="AverageOfUpperTRange"
name="AverageOfUpperTRange"><br>

            <label for="AverageOfLowerTRange">Average Of Lower T Range:</label>

            <input type="text" id="AverageOfLowerTRange"
name="AverageOfLowerTRange"><br>

            <label for="RainingDays">Raining Days:</label>

            <input type="text" id="RainingDays" name="RainingDays"><br>

            <label for="seeds">Seeds:</label>

            <input type="text" id="seeds" name="seeds"><br>

            <input type="submit" value="Predict">

        </form>

    </div>

</body>

</html>
```

predict.html:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Prediction Result</title>

    <style>
```

```
        body {
            background-color: #000000; /* Pitch black background color */
            font-family: Tahoma, sans-serif;
            display: flex;
            justify-content: center; /* Center content horizontally */
            align-items: center;
            height: 100vh;
            margin: 0;
            padding: 0;
            color: white; /* Changed text color to white */
        }

        .result-container {
            text-align: center;
        }

        .result-container h1 {
            font-size: 3rem;
            margin-bottom: 20px;
        }

        .result-container p {
            font-size: 2rem;
        }
    </style>
</head>
<body>
    <div class="result-container">
        <h1>Prediction Result</h1>
        <p>The predicted yield is: {{ '%.2f' % prediction }}</p>
    </div>
</body>
</html>
```

app.py:

```
from flask import Flask, request, render_template
import joblib
import numpy as np

app = Flask(__name__)

# Load the model
model = joblib.load('D:/Flask/models/model0.pkl')

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        # Get the data from the form
        data = [
            float(request.form['clonesize']),
            float(request.form['honeybee']),
            float(request.form['andrena']),
            float(request.form['osmia']),
            float(request.form['MinOfUpperTRange']),
            float(request.form['AverageOfUpperTRange']),
            float(request.form['AverageOfLowerTRange']),
            float(request.form['RainingDays']),
            float(request.form['seeds'])
        ]

        # Convert to numpy array and reshape for the model
        data = np.array(data).reshape(1, -1)
```

```
        # Predict using the model
        prediction = model.predict(data)

        return render_template('predict.html', prediction=prediction[0])

if __name__ == '__main__':
    app.run(debug=True)
```

Jupyter notebook -

```
[1]: import tensorflow as tf
     import warnings
     warnings.filterwarnings("ignore")
```

```
[2]: print(tf.config.list_physical_devices())
```

```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'), PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
[3]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[4]: df = pd.read_csv('WildBlueberryPollinationSimulationData (1).csv')
     df
```

3	3	37.5	0.750	0.250	0.250	0.250	94.6	57.2	79.0	68.2
4	4	37.5	0.750	0.250	0.250	0.250	86.0	52.0	71.9	62.0
...	...	...	...	...	...	...	...	...	...	...
772	772	10.0	0.537	0.117	0.409	0.058	86.0	52.0	71.9	62.0
773	773	40.0	0.537	0.117	0.409	0.058	86.0	52.0	71.9	62.0
774	774	20.0	0.537	0.117	0.409	0.058	86.0	52.0	71.9	62.0
775	775	20.0	0.537	0.117	0.409	0.058	89.0	39.0	65.6	66.0
776	776	20.0	0.537	0.117	0.409	0.058	89.0	39.0	65.6	66.0

777 rows × 18 columns

```
[5]: df.describe()
```

[5]:

	Row#	clonesize	honeybee	bumbles	andrena	osmia	MaxOfUpperTRange	MinOfUpperTRange	AverageOfUpperTRange	MaxOfLowerTRange
count	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000
mean	388.000000	18.767696	0.417133	0.282389	0.468817	0.562062	82.277091	49.700515	68.723037	59.309395
std	224.444871	6.999063	0.978904	0.066343	0.161052	0.169119	9.193745	5.595769	7.676984	6.647760
min	0.000000	10.000000	0.000000	0.000000	0.000000	0.000000	69.700000	39.000000	58.200000	50.200000
25%	194.000000	12.500000	0.250000	0.250000	0.380000	0.500000	77.400000	46.800000	64.700000	55.800000
50%	388.000000	12.500000	0.250000	0.250000	0.500000	0.630000	86.000000	52.000000	71.900000	62.000000
75%	582.000000	25.000000	0.500000	0.380000	0.630000	0.750000	89.000000	52.000000	71.900000	66.000000
max	776.000000	40.000000	18.430000	0.585000	0.750000	0.750000	94.600000	57.200000	79.000000	68.200000

```
[6]: df.isna().sum()
```

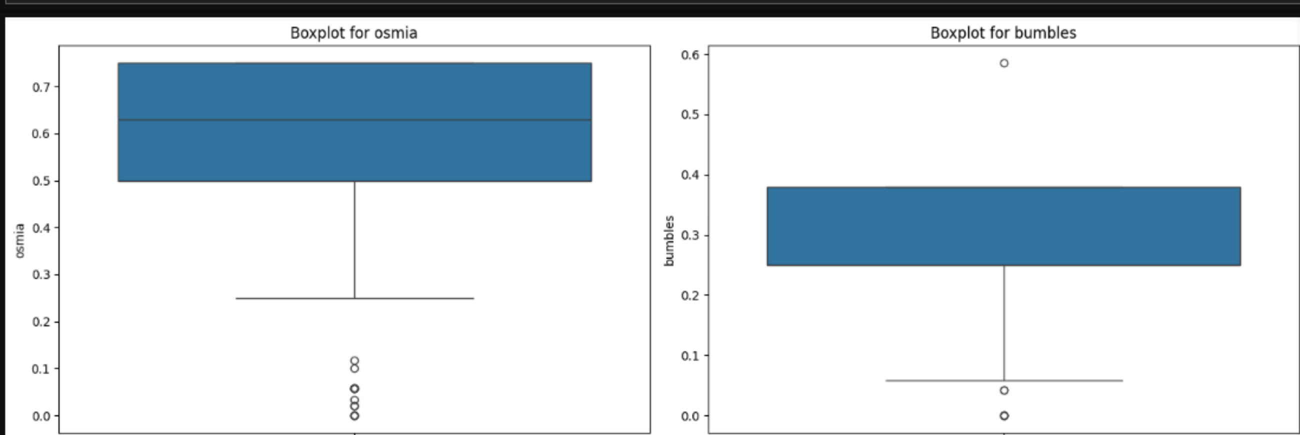
```
[6]: Row#                    0
     clonesize               0
     honeybee                0
     bumbles                 0
     andrena                 0
     osmia                   0
     MaxOfUpperTRange        0
     MinOfUpperTRange        0
     AverageOfUpperTRange    0
     MaxOfLowerTRange        0
     MinOfLowerTRange        0
     AverageOfLowerTRange    0
     RainingDays             0
     AverageRainingDays      0
     fruitset                0
     fruitmass               0
     seeds                   0
     yield                   0
     dtype: int64
```

```
[7]: df = df.drop('Row#',axis=1)
```

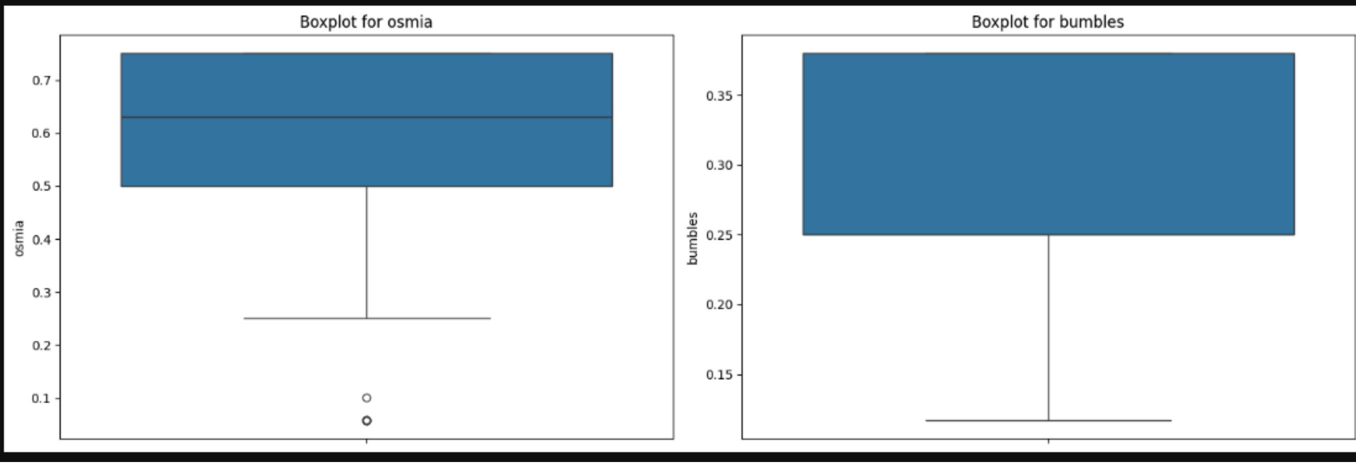
```
[50]: from scipy import stats
      # Removing outliers
      z_scores = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))
      threshold = 3
      outliers = (z_scores > threshold).any(axis=1)
      new_df = df[~outliers]
      num_outliers_removed = outliers.sum()
      print(f"Number of outliers removed: {num_outliers_removed}")
```

```
Number of outliers removed: 13
```

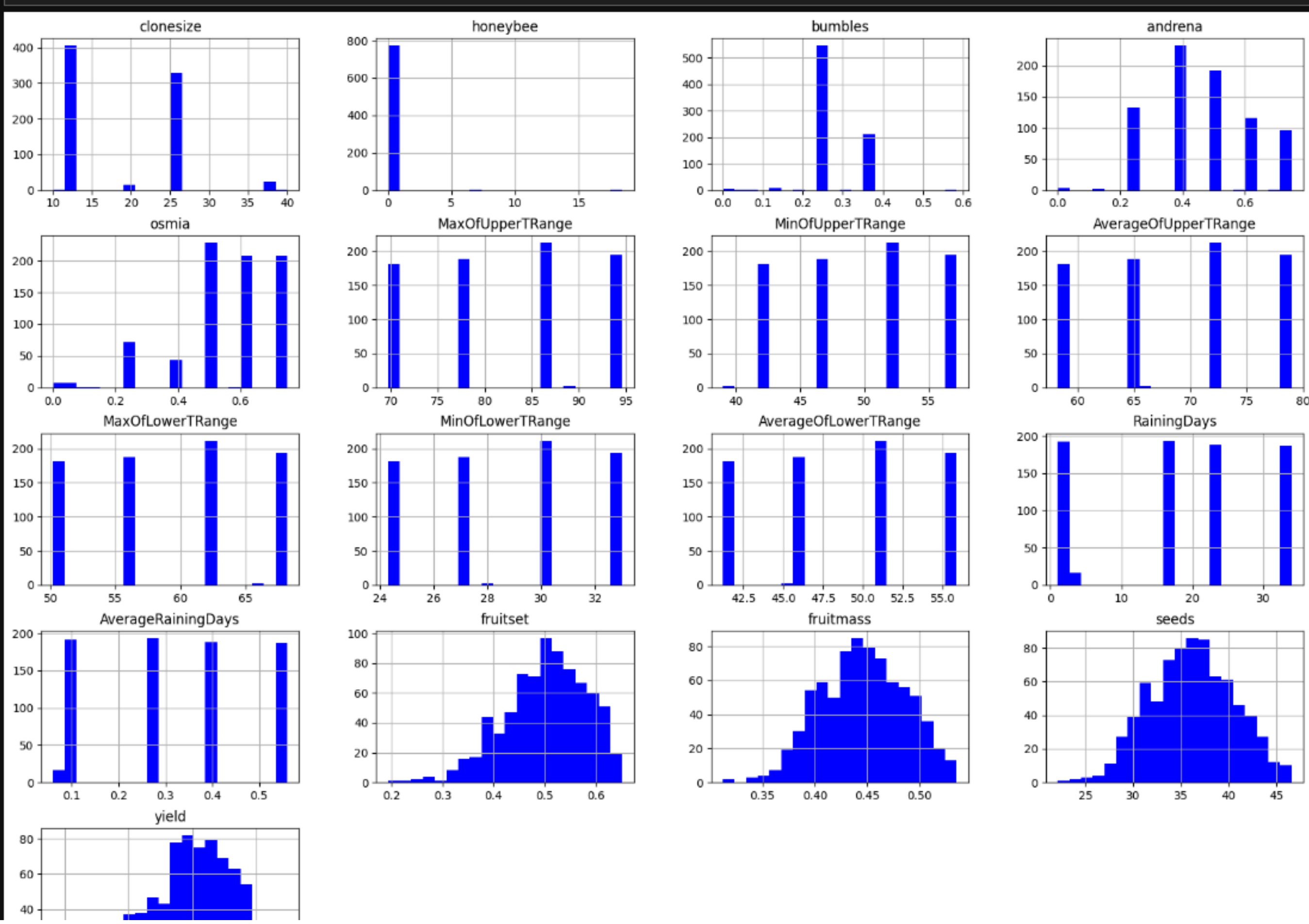
```
[51]: column1 = 'osmia'
      column2 = 'bumbles'

      fig, axes = plt.subplots(1, 2, figsize=(15, 5))

      sns.boxplot(y= df[column1], ax=axes[0])
      axes[0].set_title(f'Boxplot for {column1}')

      sns.boxplot(y=df[column2], ax=axes[1])
      axes[1].set_title(f'Boxplot for {column2}')

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



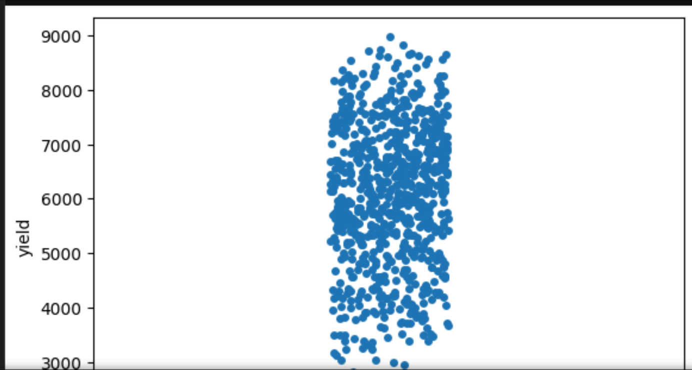
[52]:
```python
column1 = 'osmia'
column2 = 'bumbles'

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

sns.boxplot(y= new_df[column1], ax=axes[0])
axes[0].set_title(f'Boxplot for {column1}')

sns.boxplot(y=new_df[column2], ax=axes[1])
axes[1].set_title(f'Boxplot for {column2}')

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



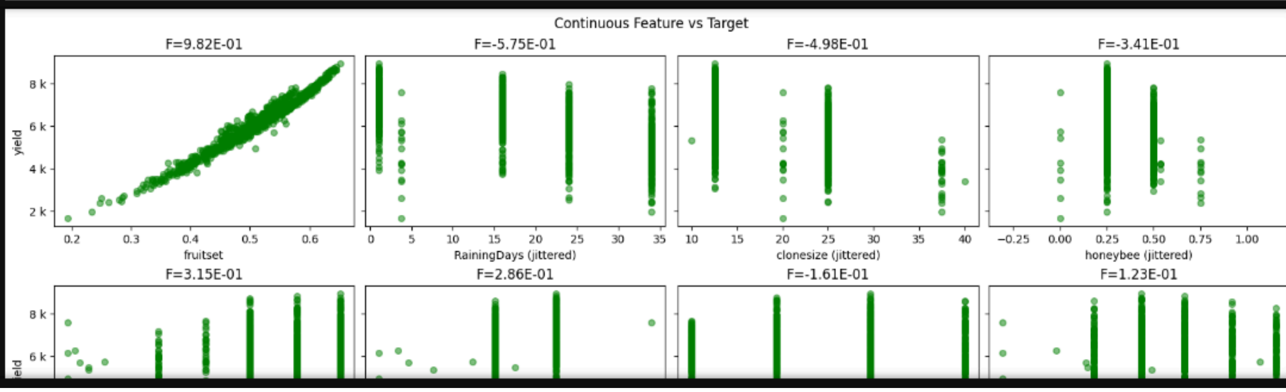
[11]:
```python
df.hist(layout=(5,4), figsize=(20,15), bins=20, color='blue')
plt.title('histogram of data')
plt.show()
```



[12]:
```
sns.stripplot(y=df['yield']);
```

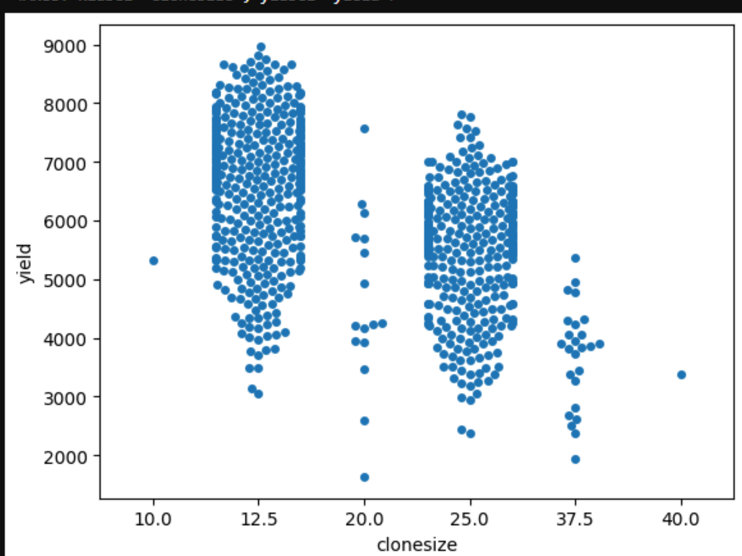


[13]:
```
# !pip install dabl
import dabl
dabl.plot(df, target_col='yield',color='green')
```



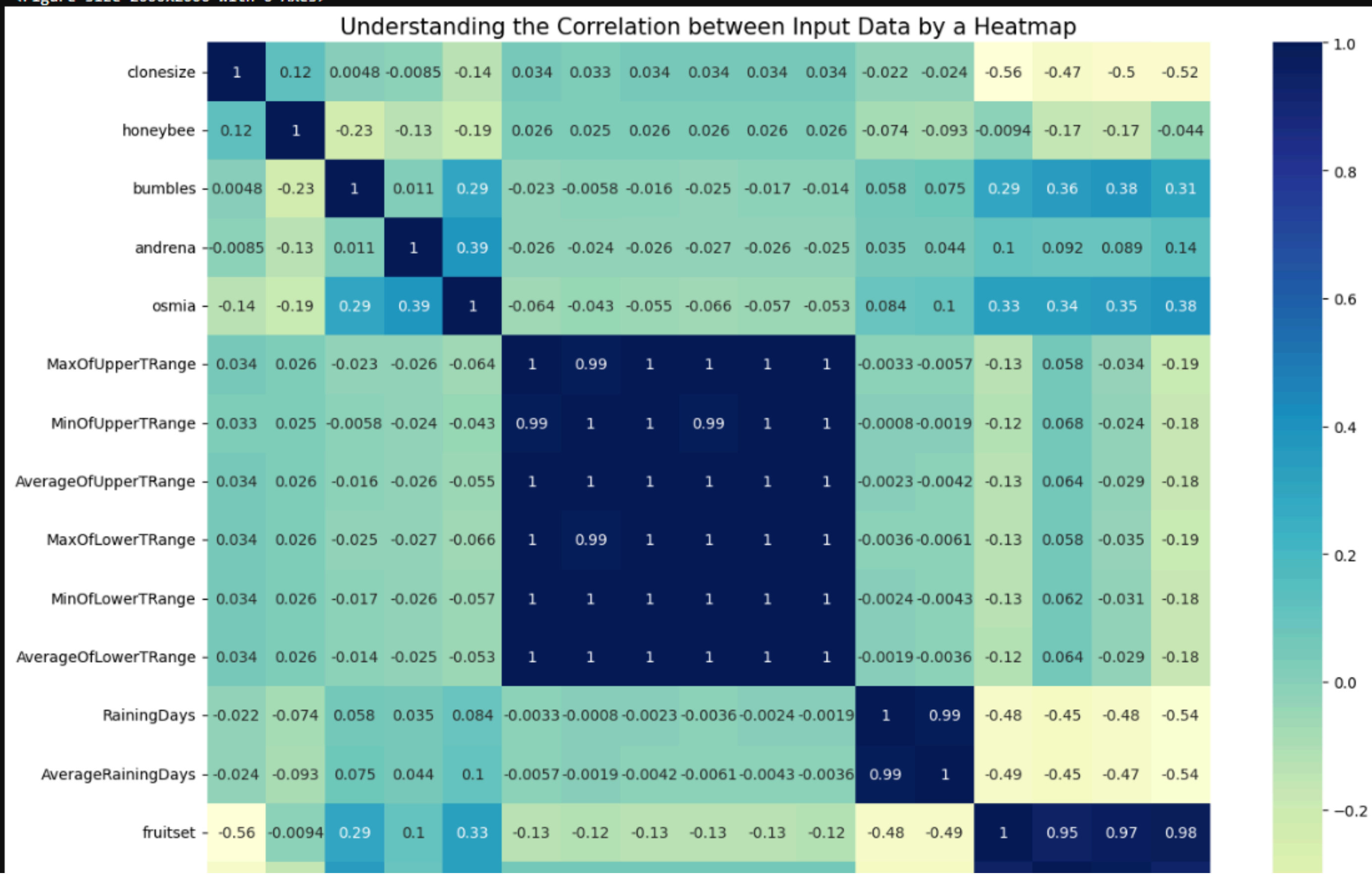
[14]:
```
sns.swarmplot(x='clonesize', y='yield', data=df)
```

[14]: <Axes: xlabel='clonesize', ylabel='yield'>



[15]:
```python
plt.figure(figsize=(20,20))
c = df.corr()
plt.figure(figsize=(15,12))
sns.heatmap(c, annot=True, cmap="YlGnBu")
plt.title('Understanding the Correlation between Input Data by a Heatmap',fontsize=15)
plt.show();
```

<Figure size 2000x2000 with 0 Axes>



[16]:
```python
new_df = new_df.drop(columns=['bumbles','fruitmass','AverageRainingDays','fruitset','MaxOfUpperTRange','MaxOfLowerTRange', 'MinOfLowerTRange'])
new_df.head()

# ALL THE ABOVE COLUMNS HAVE HIGH CORRELATION WITH OTHER COLUMNS, SO THEY ARE BEING REMOVED
```

[16]:

	clonesize	honeybee	andrena	osmia	MinOfUpperTRange	AverageOfUpperTRange	AverageOfLowerTRange	RainingDays	seeds	yield
0	37.5	0.75	0.25	0.25	52.0	71.9	50.8	16.0	31.678898	3813.165795
1	37.5	0.75	0.25	0.25	52.0	71.9	50.8	1.0	33.449385	4947.605663
2	37.5	0.75	0.25	0.25	57.2	79.0	55.9	16.0	30.546306	3866.798965
3	37.5	0.75	0.25	0.25	57.2	79.0	55.9	1.0	31.562586	4303.943030
4	37.5	0.75	0.25	0.25	52.0	71.9	50.8	24.0	28.873714	3436.493543

[17]:
```python
X = new_df.drop('yield',axis=1)
y = new_df['yield']
```

[18]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[19]:
```python
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

# Importing and building Linear Regression model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
model.score(X_test, y_test)

y_preds = model.predict(X_test)

print("Regression metrics on the test set")
print(f"R2 score: {r2_score(y_test, y_preds)}")
print(f"MAE: {mean_absolute_error(y_test,y_preds)}")
print(f"MSE: {mean_squared_error(y_test, y_preds)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_preds))}")
```

```
Regression metrics on the test set
R2 score: 0.9850297685259484
MAE: 111.5204986497179
MSE: 21684.627147497344
RMSE: 147.25701052071287
```

[20]:
```python
y_preds
```

[20]:
```
array([5553.49902135, 7392.74693061, 6647.80113106, 8382.40031049,
       6930.64931162, 6352.50676704, 4054.69954309, 7336.83277398,
       6609.80503208, 6745.23976281, 6767.52482335, 4450.5401118 ,
       7684.13956704, 6820.78074249, 7897.18906937, 5745.15287388,
       4580.56903787, 3506.78616555, 6188.04857017, 4991.24250519,
       4965.41679286, 4759.20149474, 6440.9976362 , 6257.32322812,
       4841.56489301, 7516.33865101, 5686.59310825, 7007.30726303,
       4369.53813837, 5578.88677834, 7803.46358844, 5811.01118421,
       5508.85174708, 8089.34918352, 5514.50687125, 6135.55503859,
       4800.35989681, 5363.49717772, 6030.4763171 , 5637.76360793,
       6365.81888218, 6161.63689153, 5264.49014542, 5816.03052381,
       6564.26452154, 7879.89474672, 5716.62723721, 6739.58465965,
```

[21]:
```python
yield_comparison = pd.DataFrame({
    'Original yeild':y_test,
    'Predicted yeild': y_preds
})

yield_comparison
```

[21]:

	Original yeild	Predicted yeild
360	5527.425034	5553.499021
262	7570.608619	7392.746931
753	6663.000678	6647.801131
196	8357.067222	8382.400310
336	6852.979712	6930.649312
...	...	...
63	5556.372277	5700.256050
554	6595.456285	6346.255211
345	5675.721494	5634.894539

[22]:
```python
# Importing and building the Random forest regressor model
from sklearn.ensemble import RandomForestRegressor

model2 = RandomForestRegressor()
model2.fit(X_train,y_train)

y_preds2 = model2.predict(X_test)

print("Regression metrics on the test set")
print(f"R2 score: {r2_score(y_test, y_preds2)}")
print(f"MAE: {mean_absolute_error(y_test,y_preds2)}")
print(f"MSE: {mean_squared_error(y_test, y_preds2)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_preds2))}")
```

```
Regression metrics on the test set
R2 score: 0.9786564974999222
MAE: 135.726436200327
MSE: 30916.415323176283
RMSE: 175.8306438684005
```

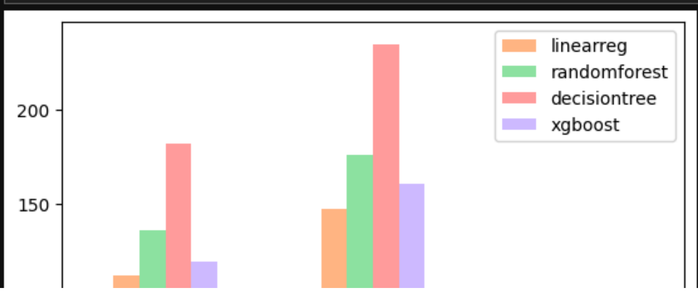
```
[23]: # Importing and building Decision tree regressor model
from sklearn.tree import DecisionTreeRegressor

model3 = DecisionTreeRegressor()
model3.fit(X_train,y_train)
y_preds3 = model3.predict(X_test)

print("Regression metrics on the test set")
print(f"R2 score: {r2_score(y_test, y_preds3)}")
print(f"MAE: {mean_absolute_error(y_test,y_preds3)}")
print(f"MSE: {mean_squared_error(y_test, y_preds3)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_preds3))}")
```

```
Regression metrics on the test set
R2 score: 0.9619595421872434
MAE: 181.7127012156863
MSE: 55102.23042439545
RMSE: 234.73864280172415
```

```
[24]: # Importing and building XGB Regressor model
from xgboost import XGBRegressor
model4 = XGBRegressor()
model4.fit(X_train,y_train)
model4.score(X_test,y_test)

y_preds4 = model4.predict(X_test)

print("Regression metrics on the test set")
print(f"R2 score: {r2_score(y_test, y_preds4)}")
print(f"MAE: {mean_absolute_error(y_test,y_preds4)}")
print(f"MSE: {mean_squared_error(y_test, y_preds4)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_preds4))}")
```

```
Regression metrics on the test set
R2 score: 0.982172684010463
MAE: 119.19604783486517
MSE: 25823.161181161377
RMSE: 160.69586547625107
```

```
[25]: error_rec = {
    "linearreg":{
        "mae": mean_absolute_error(y_test,y_preds),
        "rmse":np.sqrt(mean_squared_error(y_test, y_preds)),
        "r2": r2_score(y_test, y_preds) *100
    },

    "randomforest":{
    "mae": mean_absolute_error(y_test,y_preds2),
    "rmse":np.sqrt(mean_squared_error(y_test, y_preds2)),
    "r2": r2_score(y_test, y_preds2) *100
    },

    "decisiontree":{
    "mae": mean_absolute_error(y_test,y_preds3),
    "rmse":np.sqrt(mean_squared_error(y_test, y_preds3)),
    "r2": r2_score(y_test, y_preds3) *100
    },

    "xgboost":{
    "mae": mean_absolute_error(y_test,y_preds4),
    "rmse":np.sqrt(mean_squared_error(y_test, y_preds4)),
    "r2": r2_score(y_test, y_preds4) *100
    },

}

pd.DataFrame(error_rec).plot(kind="bar",color = [
    sns.color_palette("pastel")[1],
    sns.color_palette("pastel")[2],
    sns.color_palette("pastel")[3],
    sns.color_palette("pastel")[4]
]);
```



```
[26]: from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}
```

```
[27]: rf = RandomForestRegressor()
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=3, n_jobs=-1, verbose=2)
```

```
[28]: grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# You can now evaluate the performance of the best model
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Fitting 3 folds for each of 648 candidates, totalling 1944 fits
Best Parameters: {'bootstrap': False, 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Mean Squared Error: 23928.954602206504
R-squared: 0.9834803712837439
```

```
[29]: print("Regression metrics on the test set")
print(f"R2 score: {r2_score(y_test, y_pred)}")
print(f"MAE: {mean_absolute_error(y_test,y_pred)}")
print(f"MSE: {mean_squared_error(y_test, y_pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred))}")
```

```
Regression metrics on the test set
R2 score: 0.9834803712837439
MAE: 123.25041424676502
MSE: 23928.954602206504
RMSE: 154.68986586782762
```

```
[30]: param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'auto', 'sqrt', 'log2']
}
```

```
[31]: dt = DecisionTreeRegressor()
grid_search = GridSearchCV(estimator=dt, param_grid=param_grid,
                           cv=3, n_jobs=-1, verbose=2)
```

```
[32]: grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate the performance
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Fitting 3 folds for each of 144 candidates, totalling 432 fits
Best Parameters: {'max_depth': None, 'max_features': None, 'min_samples_leaf': 4, 'min_samples_split': 2}
Mean Squared Error: 39945.42992962877
R-squared: 0.9724232135369657
```

```
[33]: print("Regression metrics on the test set")
print(f"R2 score: {r2_score(y_test, y_pred)}")
print(f"MAE: {mean_absolute_error(y_test,y_pred)}")
print(f"MSE: {mean_squared_error(y_test, y_pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred))}")
```

```
Regression metrics on the test set
R2 score: 0.9724232135369657
MAE: 155.86761325904146
MSE: 39945.42992962877
RMSE: 199.86352826273423
```

```
[34]: param_grid = {
          'n_estimators': [100, 200, 300],
          'learning_rate': [0.01, 0.1, 0.2],
          'max_depth': [3, 6, 9],
          'subsample': [0.6, 0.8, 1.0],
          'colsample_bytree': [0.6, 0.8, 1.0]
      }
```

```
[35]: xgb = XGBRegressor()
      grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid,
                                 cv=3, n_jobs=-1, verbose=2)
```

```
[36]: grid_search.fit(X_train, y_train)
      best_params = grid_search.best_params_
      print("Best Parameters:", best_params)

      best_model = grid_search.best_estimator_
      y_pred = best_model.predict(X_test)

      # Evaluate the performance
      from sklearn.metrics import mean_squared_error, r2_score
      mse = mean_squared_error(y_test, y_pred)
      r2 = r2_score(y_test, y_pred)
      print("Mean Squared Error:", mse)
      print("R-squared:", r2)
```

```
Fitting 3 folds for each of 243 candidates, totalling 729 fits
Best Parameters: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200, 'subsample': 0.6}
Mean Squared Error: 19206.509150671092
R-squared: 0.9867405657547946
```

```
[37]: print("Regression metrics on the test set")
      print(f"R2 score: {r2_score(y_test, y_pred)}")
      print(f"MAE: {mean_absolute_error(y_test,y_pred)}")
      print(f"MSE: {mean_squared_error(y_test, y_pred)}")
      print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred))}")
```

```
Regression metrics on the test set
R2 score: 0.9867405657547946
MAE: 108.15756897906451
MSE: 19206.509150671092
RMSE: 138.5875504894689
```

```
[38]: best_model.score(X_test,y_test)
```

```
[38]: 0.9867405657547946
```

```
[40]: new_df.describe()
```

[40]:

	clonesize	honeybee	andrena	osmia	MinOfUpperTRange	AverageOfUpperTRange	AverageOfLowerTRange	RainingDays	seeds	yield
count	764.000000	764.000000	764.000000	764.000000	764.000000	764.000000	764.000000	764.000000	764.000000	764.000000
mean	18.668194	0.356263	0.473292	0.569551	49.640969	68.641099	48.555890	18.437984	36.263019	6046.236614
std	6.944938	0.132262	0.156988	0.158953	5.617130	7.706229	5.437822	12.072542	4.210077	1322.283274
min	10.000000	0.000000	0.234000	0.058000	39.000000	58.200000	41.200000	1.000000	26.054692	2452.680747
25%	12.500000	0.250000	0.380000	0.500000	46.800000	64.700000	45.800000	1.000000	33.280391	5154.078554
50%	12.500000	0.250000	0.500000	0.630000	52.000000	71.900000	50.800000	16.000000	36.255705	6121.585642
75%	25.000000	0.500000	0.630000	0.750000	53.300000	73.675000	52.075000	24.000000	39.333527	7031.850168
max	37.500000	0.750000	0.750000	0.750000	57.200000	79.000000	55.900000	34.000000	46.369344	8969.401842

```
[41]: print(model.predict([[37,0.85,0.40,0.35,52,72,52,16,30.5]]))
      print(model.predict([[20,0.34,0.50,0.32,41,67,50,24,33]]))
      print(model.predict([[2,8,0.9,1,10,5,9,5,8]]))
      print(model.predict([[8,6.5,7,5,9.9,6.2,5.3,8,10]]))
      print(model.predict([[25.5,1,0.5,0,40,30,44,30,28]]))
```

```
[4386.98503759]
[5465.03422871]
[7483.90078284]
[10591.79073137]
[9985.25257197]
```

```
[42]: import pickle
      pickle.dump(best_model,open('BestModel.pkl','wb'))
```

10.2 GitHub & Project Demo Link

Project Demo Link:https://youtu.be/Igx38ajI07o

Github repositories:

Sona Subramanian: https://github.com/SonaSubramanian294/Blueberry-Yield-Prediction

Sai Veshwa B: https://github.com/Sai-1574/Blueberry-Yield-Prediction

Kumar Abhishek:https://github.com/sleepcoder20/22BCE1907_Blueberry_Prediction

Dinari Ajay Kumar:https://github.com/ajaykumard12/Blueberry-Yield-Prediction