Wild Blueberry Yield Prediction

Upload Daataset file

```
from google.colab import files
files.upload()
```

Choose files WildBlueber…tionData.csv

- **WildBlueberryPollinationSimulationData.csv**(text/csv) - 85259 bytes, last modified: 15/02/2024 - 100% done

```
Saving WildBlueberryPollinationSimulationData.csv to WildBlueberryPollinationSimulationData.csv
{'WildBlueberryPollinationSimulationData.csv':
b'Row#,clonesize,honeybee,bumbles,andrena,osmia,MaxOfUpperTRange,MinOfUpperTRange,AverageOfUpperTRange,MaxOfLowerTRange,MinOfLowerTRange,AverageOfLowerTRange,RainingDays,AverageRainingDays,fruitset
```

```
import pandas as pd

# Provide the path to your dataset file
file_path = "/content/WildBlueberryPollinationSimulationData.csv"

# Read the dataset into a Pandas DataFrame
df = pd.read_csv(file_path)

# Display the first few rows of the DataFrame
df.head()
```

	Row#	clonesize	honeybee	bumbles	andrena	osmia	MaxOfUpperTRange	MinOfUpperTRange	AverageOfUpperTRange	MaxOfLowerTRange	MinOfLowerTRange	AverageOfLowerTRange	RainingDays	AverageRainingD
0	0	37.5	0.75	0.25	0.25	0.25	86.0	52.0	71.9	62.0	30.0	50.8	16.0	
1	1	37.5	0.75	0.25	0.25	0.25	86.0	52.0	71.9	62.0	30.0	50.8	1.0	
2	2	37.5	0.75	0.25	0.25	0.25	94.6	57.2	79.0	68.2	33.0	55.9	16.0	
3	3	37.5	0.75	0.25	0.25	0.25	94.6	57.2	79.0	68.2	33.0	55.9	1.0	
4	4	37.5	0.75	0.25	0.25	0.25	86.0	52.0	71.9	62.0	30.0	50.8	24.0	

Next steps: View recommended plots

Task 1: Exploratory Data Analysis

```
print(df.columns)
```

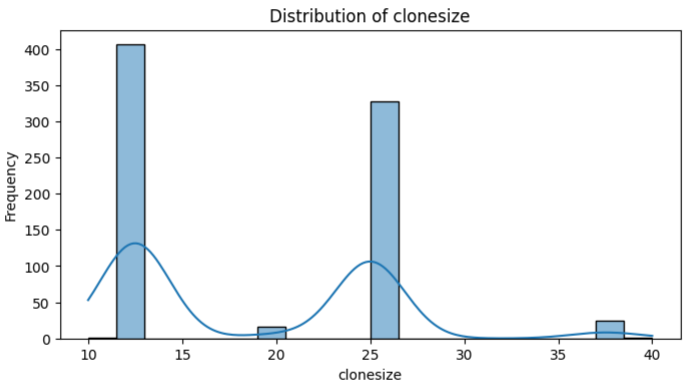
```
Index(['Row#', 'clonesize', 'honeybee', 'bumbles', 'andrena', 'osmia',
       'MaxOfUpperTRange', 'MinOfUpperTRange', 'AverageOfUpperTRange',
       'MaxOfLowerTRange', 'MinOfLowerTRange', 'AverageOfLowerTRange',
       'RainingDays', 'AverageRainingDays', 'fruitset', 'fruitmass', 'seeds',
       'yield'],
      dtype='object')
```

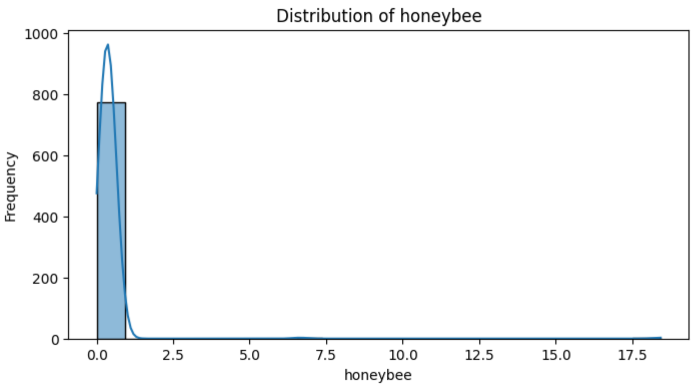
```
print(df['yield'].describe())
```

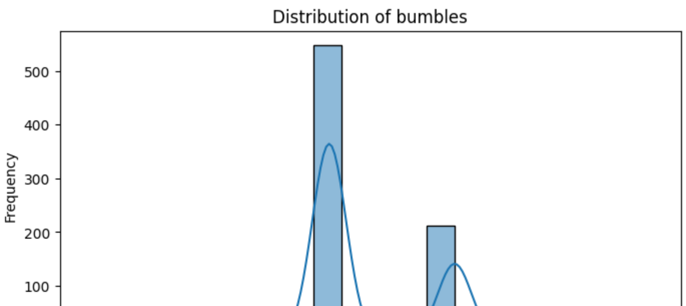
```
count     777.000000
mean     6012.849165
std      1356.955318
min      1637.704022
25%      5124.854901
50%      6107.382466
75%      7022.189731
max      8969.401842
Name: yield, dtype: float64
```

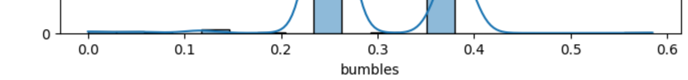
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Display the first few rows of the DataFrame
print(df.head())

# Display summary statistics
print(df.describe())

# Visualize the distribution of each numerical variable
numerical_columns = df.select_dtypes(include=['float64']).columns
for column in numerical_columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[column], bins=20, kde=True)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```

```
   Row#  clonesize  honeybee  bumbles  andrena  osmia  MaxOfUpperTRange  \
0     0       37.5      0.75     0.25     0.25   0.25              86.0
1     1       37.5      0.75     0.25     0.25   0.25              86.0
2     2       37.5      0.75     0.25     0.25   0.25              94.6
3     3       37.5      0.75     0.25     0.25   0.25              94.6
4     4       37.5      0.75     0.25     0.25   0.25              86.0

   MinOfUpperTRange  AverageOfUpperTRange  MaxOfLowerTRange  MinOfLowerTRange  \
0              52.0                  71.9              62.0              30.0
1              52.0                  71.9              62.0              30.0
2              57.2                  79.0              68.2              33.0
3              57.2                  79.0              68.2              33.0
4              52.0                  71.9              62.0              30.0

   AverageOfLowerTRange  RainingDays  AverageRainingDays  fruitset  fruitmass  \
0                  50.8         16.0                0.26  0.410652   0.408159
1                  50.8          1.0                0.10  0.444254   0.425458
2                  55.9         16.0                0.26  0.383787   0.399172
3                  55.9          1.0                0.10  0.407564   0.408789
4                  50.8         24.0                0.39  0.354413   0.382703

       seeds        yield
0  31.678898  3813.165795
1  33.449385  4947.605663
2  30.546306  3866.798965
3  31.562586  4303.943030
4  28.873714  3436.493543
             Row#   clonesize    honeybee     bumbles     andrena       osmia  \
count  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
std    224.444871    6.999063    0.978904    0.066343    0.161052    0.169119
min      0.000000   10.000000    0.000000    0.000000    0.000000    0.000000
25%    194.000000   12.500000    0.250000    0.250000    0.380000    0.500000
50%    388.000000   12.500000    0.250000    0.250000    0.500000    0.630000
75%    582.000000   25.000000    0.500000    0.380000    0.630000    0.750000
max    776.000000   40.000000   18.430000    0.585000    0.750000    0.750000

       MaxOfUpperTRange  MinOfUpperTRange  AverageOfUpperTRange  \
count        777.000000        777.000000            777.000000
mean          82.277091         49.700515             68.723037
std            9.193745          5.595769              7.676984
min           69.700000         39.000000             58.200000
25%           77.400000         46.800000             64.700000
50%           86.000000         52.000000             71.900000
75%           89.000000         52.000000             71.900000
max           94.600000         57.200000             79.000000

       MaxOfLowerTRange  MinOfLowerTRange  AverageOfLowerTRange  RainingDays  \
count        777.000000        777.000000            777.000000   777.000000
mean          59.309395         28.690219             48.613127    18.309292
std            6.647760          3.209547              5.417072    12.124226
min           50.200000         24.300000             41.200000     1.000000
25%           55.800000         27.000000             45.800000     3.770000
50%           62.000000         30.000000             50.800000    16.000000
75%           66.000000         30.000000             50.800000    24.000000
max           68.200000         33.000000             55.900000    34.000000

       AverageRainingDays    fruitset   fruitmass       seeds        yield
count          777.000000  777.000000  777.000000  777.000000   777.000000
mean             0.320000    0.502121    0.445983   36.122432  6012.849165
std              0.171279    0.079445    0.040333    4.377889  1356.955318
min              0.060000    0.192732    0.311921   22.079199  1637.704022
25%              0.100000    0.454725    0.416281   33.116091  5124.854901
50%              0.260000    0.508297    0.445587   36.166044  6107.382466
75%              0.390000    0.561297    0.476149   39.239668  7022.189731
max              0.560000    0.652144    0.535660   46.585105  8969.401842
```

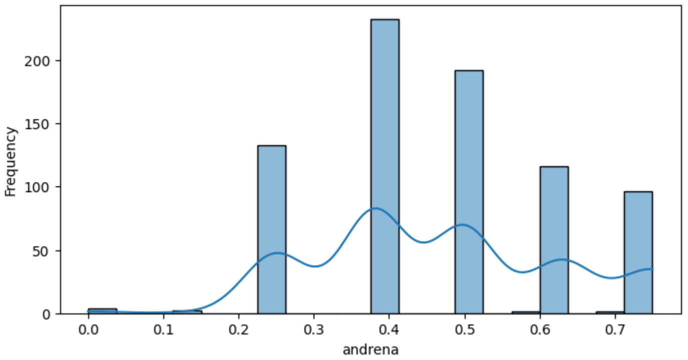




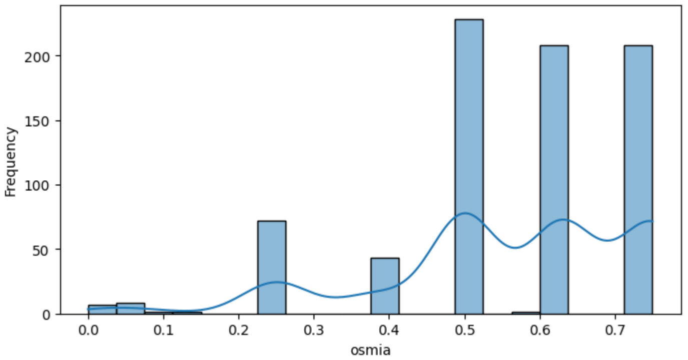




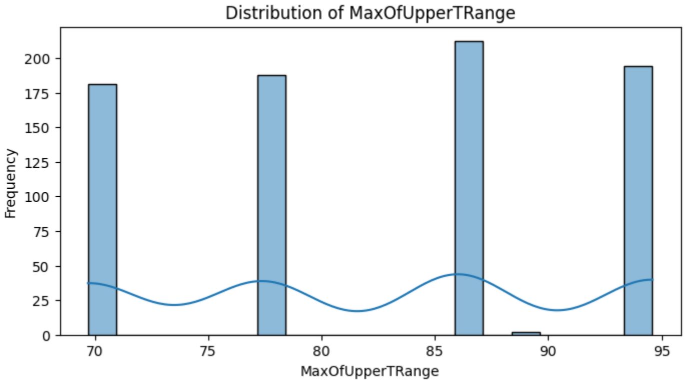
Distribution of andrena



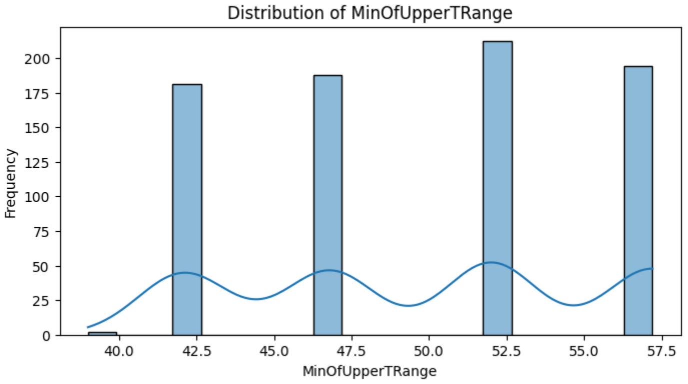
Distribution of osmia



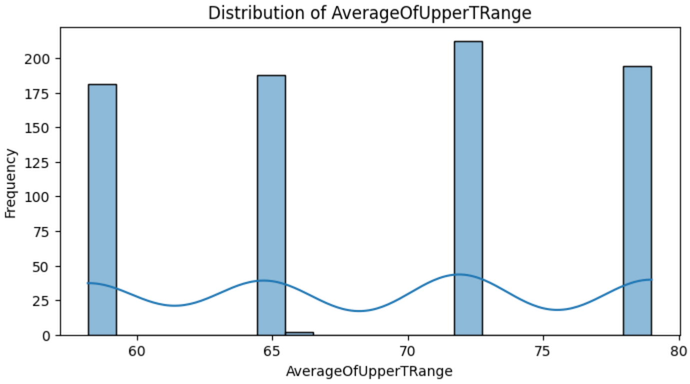
Distribution of MaxOfUpperTRange



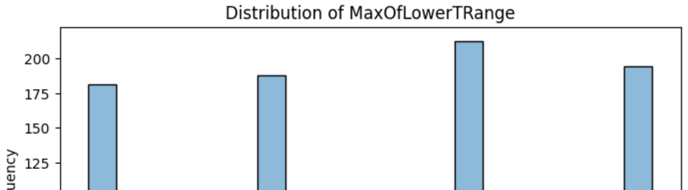
Distribution of MinOfUpperTRange

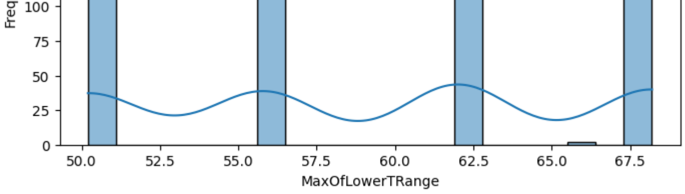


Distribution of AverageOfUpperTRange

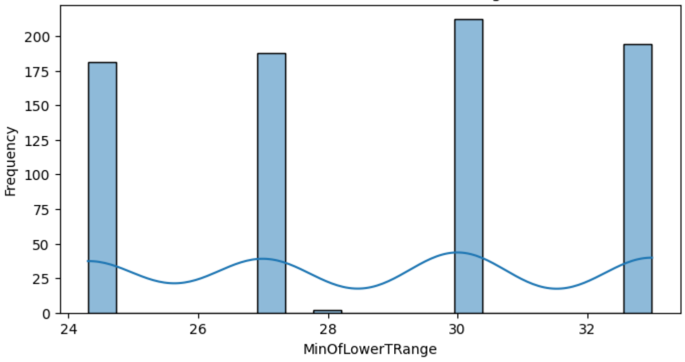


Distribution of MaxOfLowerTRange

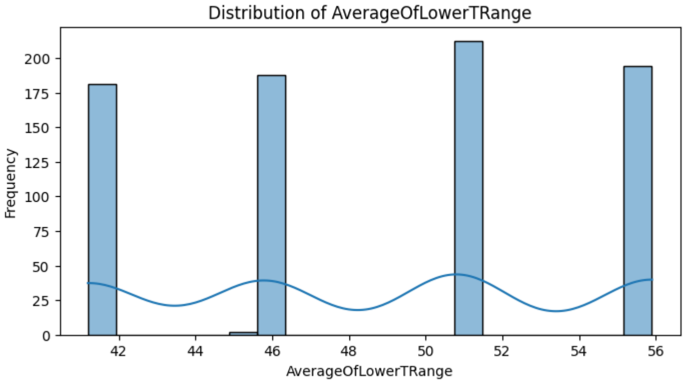




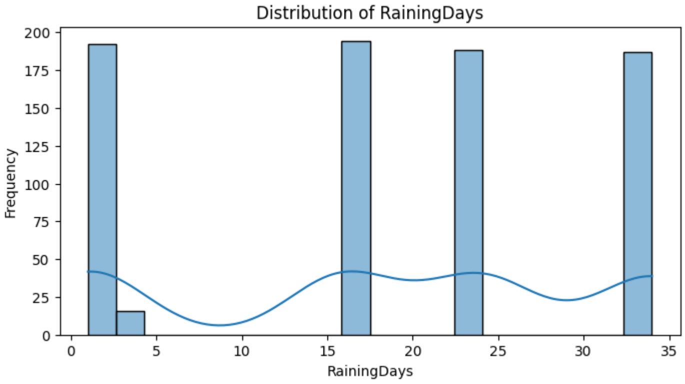
Distribution of MinOfLowerTRange



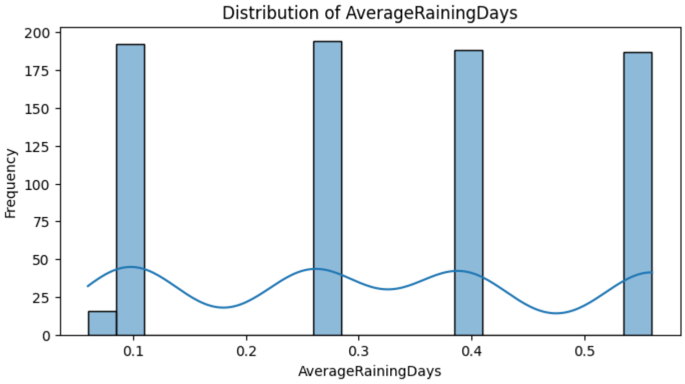
Distribution of AverageOfLowerTRange



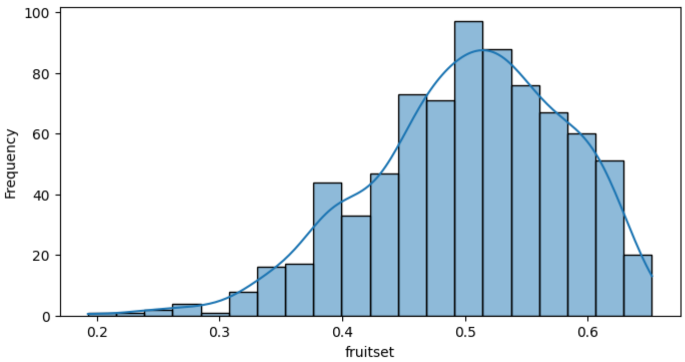
Distribution of RainingDays



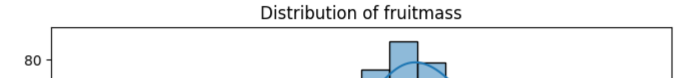
Distribution of AverageRainingDays

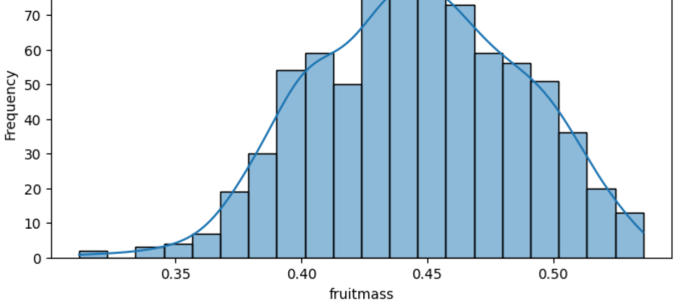


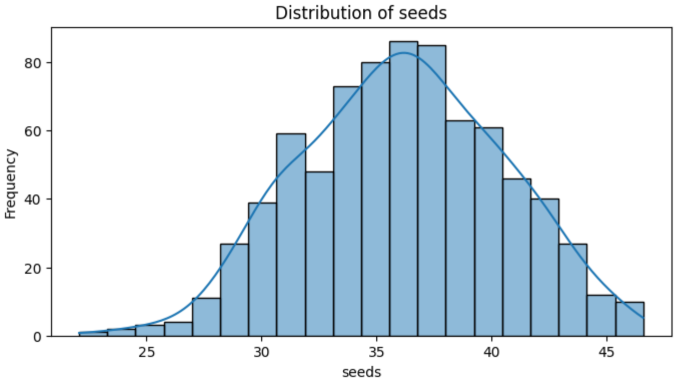
Distribution of fruitset

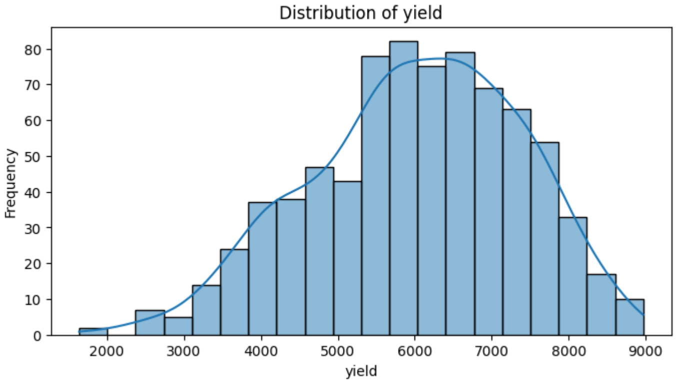


Distribution of fruitmass

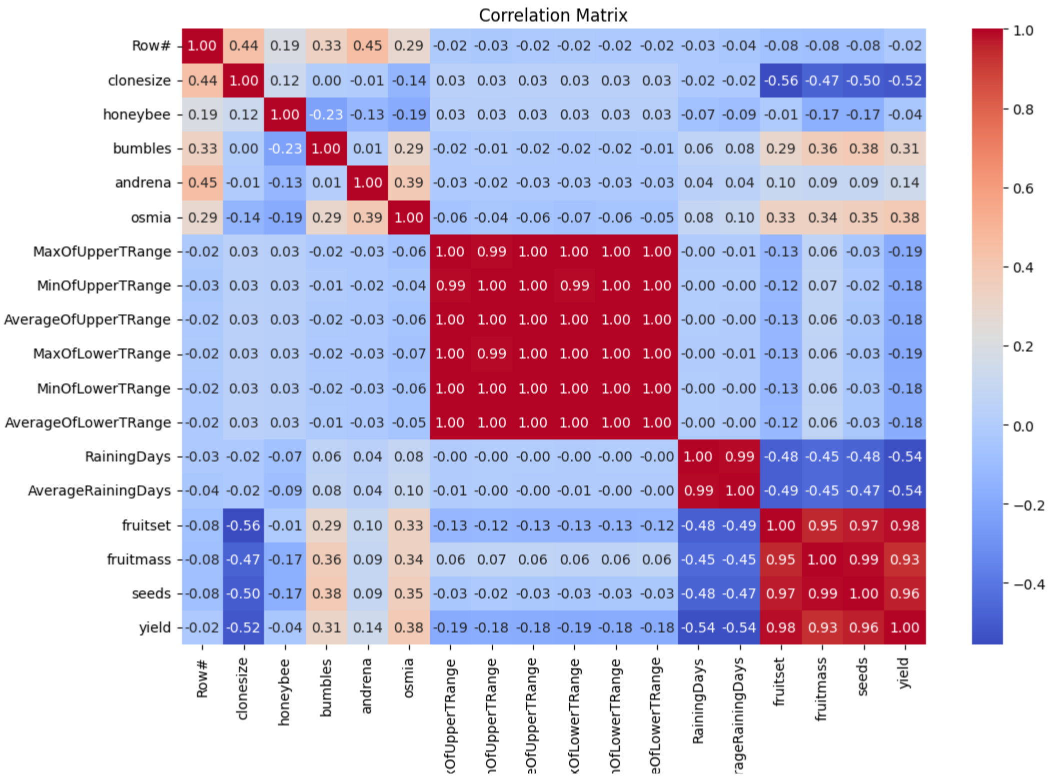






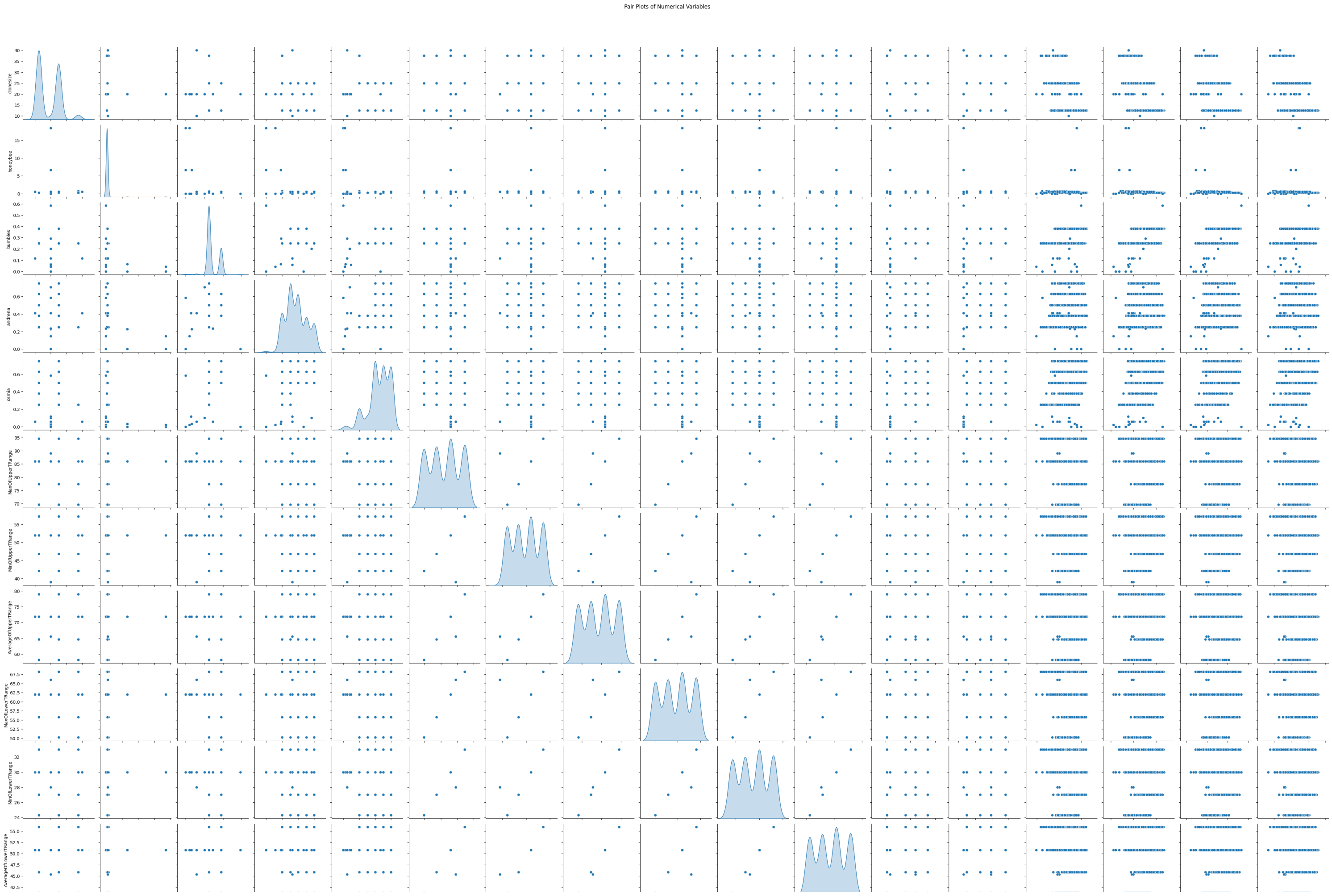



```
# Correlation matrix
correlation_matrix = df.corr()

# Heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



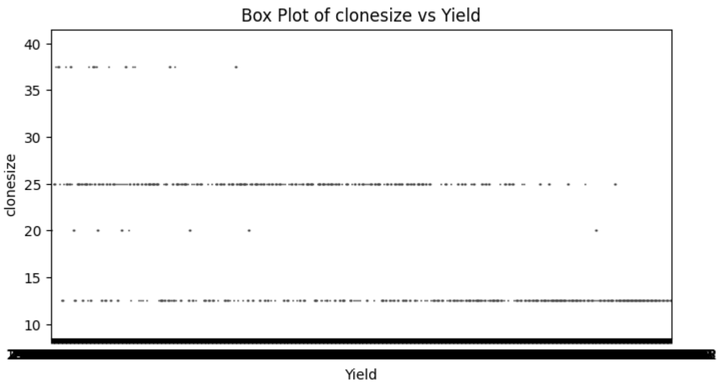
With Heatmaps we can identify patterns of correlation or dependence between different variables in the DataFrame. Positive correlations are indicated by warmer colors, negative correlations by cooler colors, and the intensity of the color represents the strength of the correlation. The annotated values in each cell provide the exact correlation coefficient between the corresponding variables.

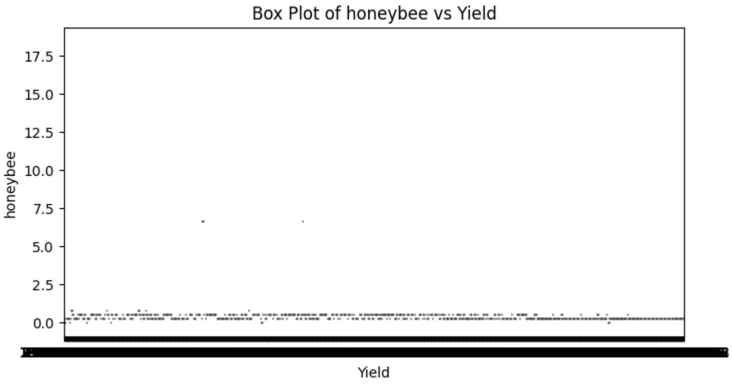
```
# Pair plots
sns.pairplot(df, vars=numerical_columns, diag_kind='kde')
plt.suptitle("Pair Plots of Numerical Variables", y=1.02)
plt.show()
```

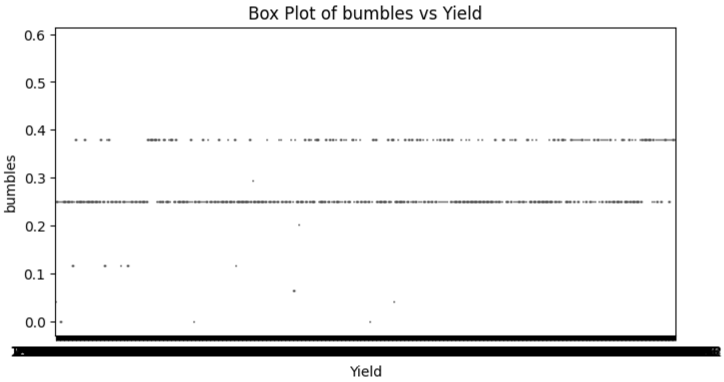


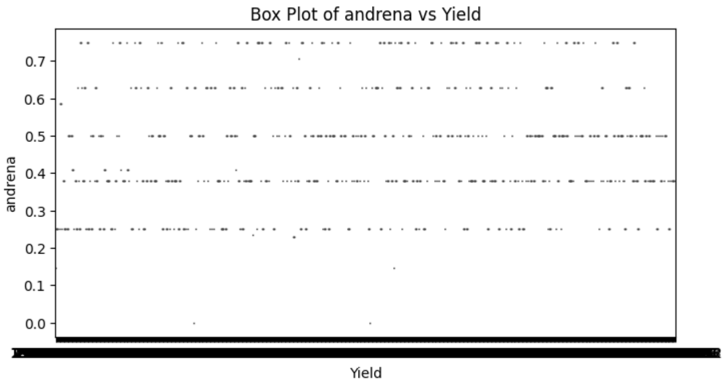
The resulting visualization is a matrix of scatterplots, where each cell shows the relationship between two numerical variables. The diagonal contains kernel density plots for each variable, and the scatterplots in the lower and upper triangles show the bivariate relationships. This type of visualization is useful for identifying patterns, trends, and potential correlations between different numerical features in the dataset.

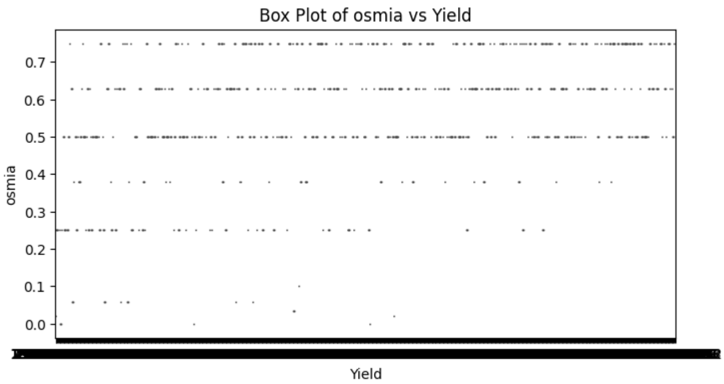
```
# Box plots
for column in numerical_columns:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x='yield', y=column, data=df)
    plt.title(f'Box Plot of {column} vs Yield')
    plt.xlabel('Yield')
    plt.ylabel(column)
    plt.show()
```

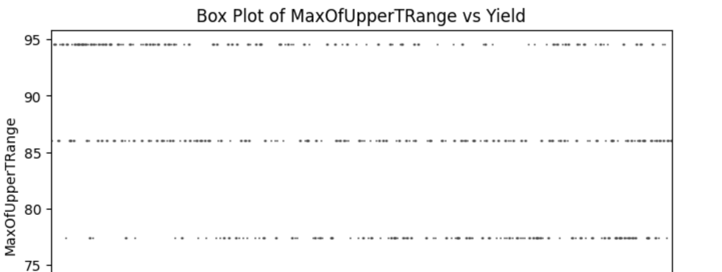

Box Plot of clonesize vs Yield
clonesize
40
35
30
25
20
15
10
Yield

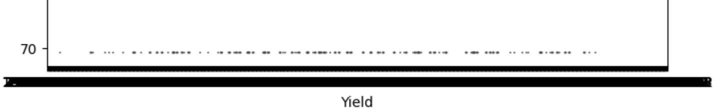

Box Plot of honeybee vs Yield
honeybee
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Yield


Box Plot of bumbles vs Yield
bumbles
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Yield

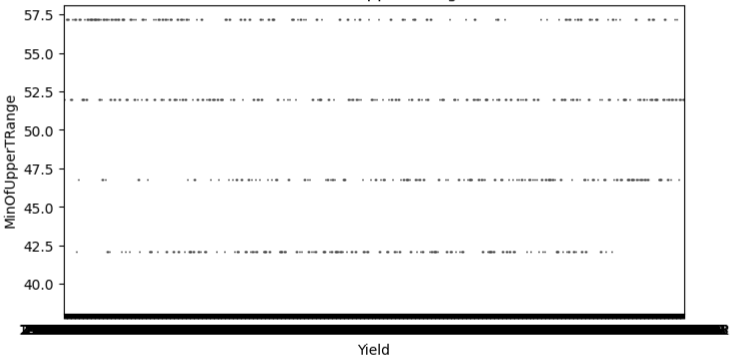

Box Plot of andrena vs Yield
andrena
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Yield


Box Plot of osmia vs Yield
osmia
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Yield

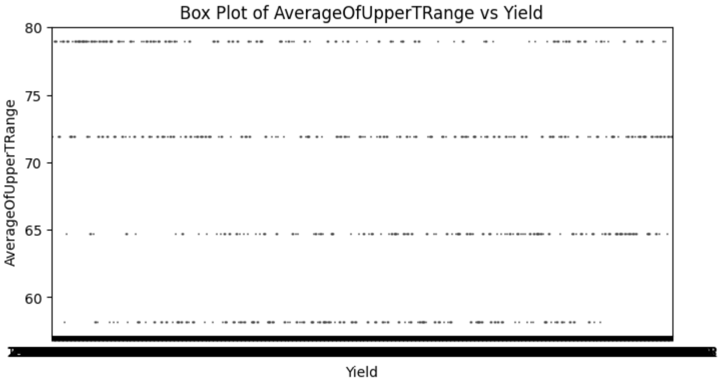

Box Plot of MaxOfUpperTRange vs Yield
MaxOfUpperTRange
95
90
85
80
75



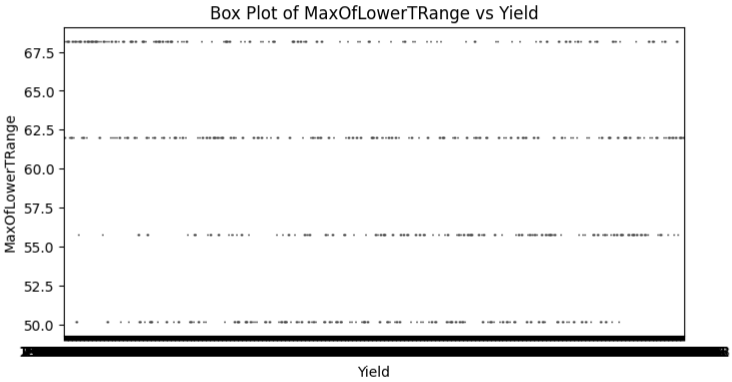
Box Plot of MinOfUpperTRange vs Yield



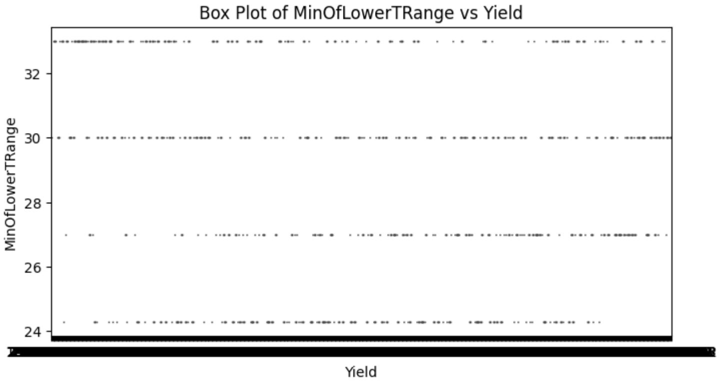
Box Plot of AverageOfUpperTRange vs Yield



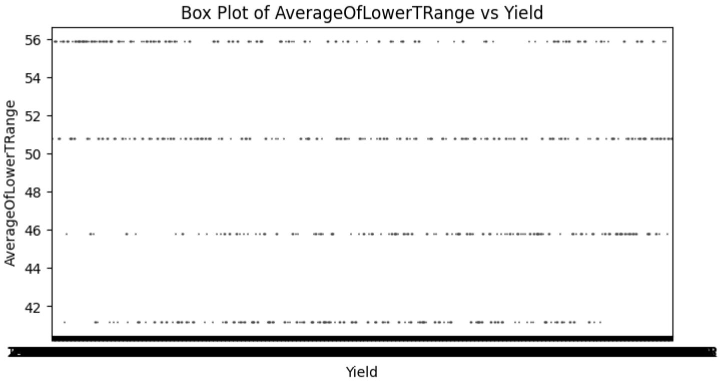
Box Plot of MaxOfLowerTRange vs Yield



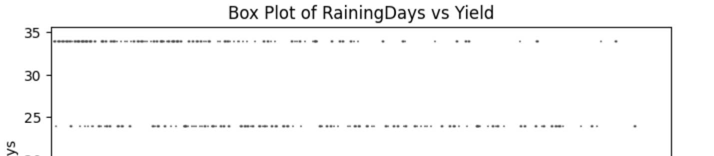
Box Plot of MinOfLowerTRange vs Yield

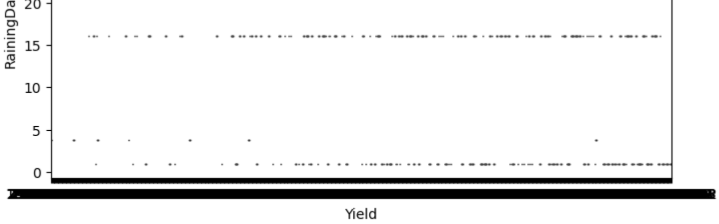


Box Plot of AverageOfLowerTRange vs Yield

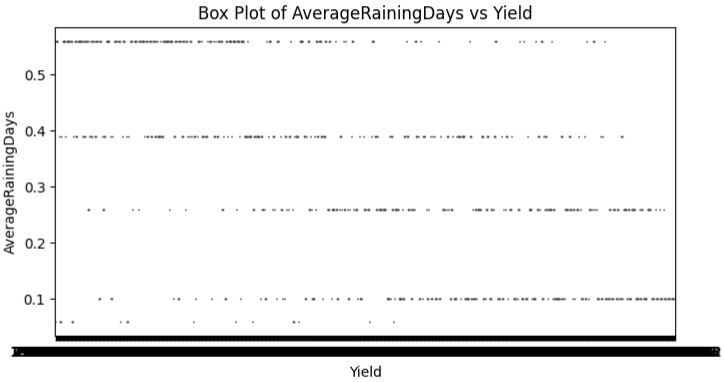


Box Plot of RainingDays vs Yield

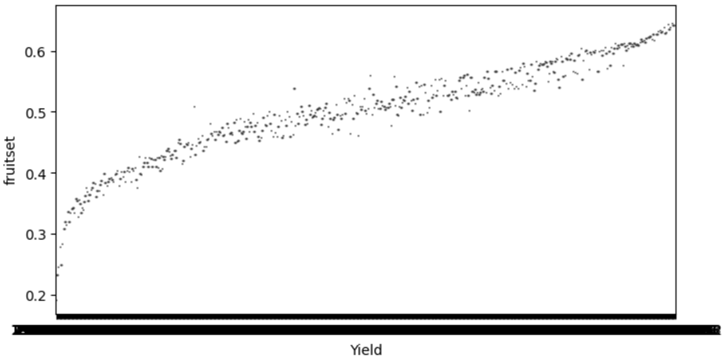




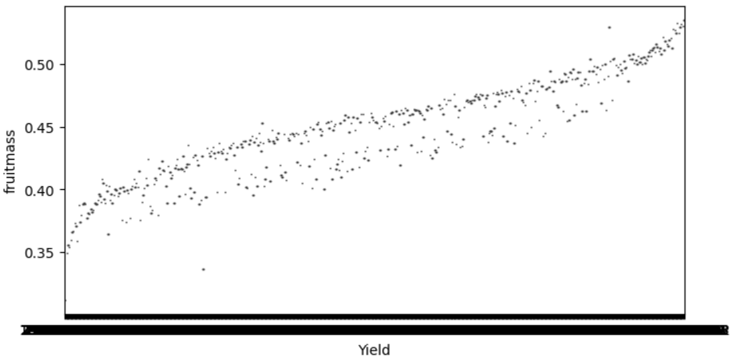
Box Plot of AverageRainingDays vs Yield



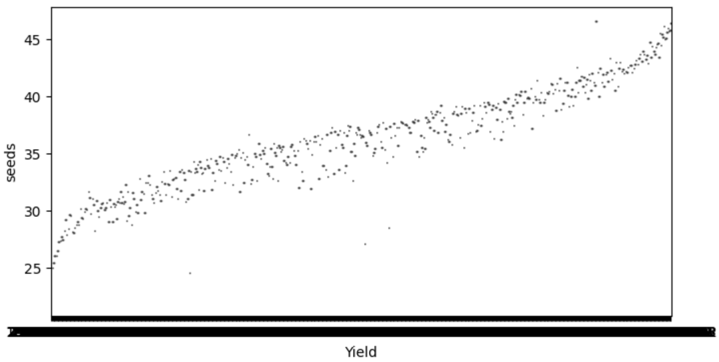
Box Plot of fruitset vs Yield



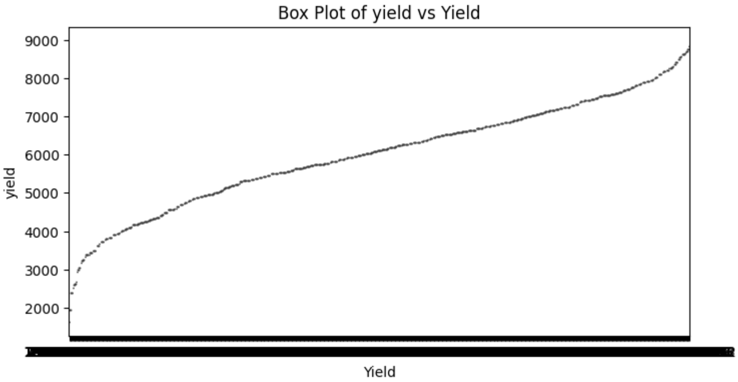
Box Plot of fruitmass vs Yield



Box Plot of seeds vs Yield



Box Plot of yield vs Yield



The resulting visualizations are a series of box plots, each showing the distribution of a numerical variable with respect to the 'yield'. Box plots provide information about the median, quartiles, and potential outliers in the distribution of each variable for different levels of the 'yield'.

Task 2: Preprocessing

```
# Check for missing values
missing_values = df.isnull().sum()

# Print missing values
print("Missing Values:")
print(missing_values)

# Handle missing values (example: fill with mean)
df.fillna(df.mean(), inplace=True)

# Print the updated dataframe
print("\nUpdated DataFrame after handling missing values:")
print(df.head())
```

```
Missing Values:
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

Updated DataFrame after handling missing values:
   Row#  clonesize  honeybee  bumbles  andrena  osmia  MaxOfUpperTRange  \
0     0       37.5      0.75     0.25     0.25   0.25              86.0
1     1       37.5      0.75     0.25     0.25   0.25              86.0
2     2       37.5      0.75     0.25     0.25   0.25              94.6
3     3       37.5      0.75     0.25     0.25   0.25              94.6
4     4       37.5      0.75     0.25     0.25   0.25              86.0

   MinOfUpperTRange  AverageOfUpperTRange  MaxOfLowerTRange  MinOfLowerTRange  \
0              52.0                  71.9              62.0              30.0
1              52.0                  71.9              62.0              30.0
2              57.2                  79.0              68.2              33.0
3              57.2                  79.0              68.2              33.0
4              52.0                  71.9              62.0              30.0

   AverageOfLowerTRange  RainingDays  AverageRainingDays  fruitset  fruitmass  \
0                  50.8         16.0                0.26  0.410652   0.408159
1                  50.8          1.0                0.10  0.444254   0.425458
2                  55.9         16.0                0.26  0.383787   0.399172
3                  55.9          1.0                0.10  0.407564   0.408789
4                  50.8         24.0                0.39  0.354413   0.382703

       seeds        yield
0  31.678898  3813.165795
1  33.449385  4947.605663
2  30.546306  3866.798965
3  31.562586  4303.943030
4  28.873714  3436.493543
```

```
print(df.columns)
```

```
Index(['Row#', 'clonesize', 'honeybee', 'bumbles', 'andrena', 'osmia',
       'MaxOfUpperTRange', 'MinOfUpperTRange', 'AverageOfUpperTRange',
       'MaxOfLowerTRange', 'MinOfLowerTRange', 'AverageOfLowerTRange',
       'RainingDays', 'AverageRainingDays', 'fruitset', 'fruitmass', 'seeds',
       'yield'],
      dtype='object')
```

```
from sklearn.preprocessing import LabelEncoder

# List of columns to encode
columns_to_encode = ['Row#', 'clonesize', 'honeybee', 'bumbles', 'andrena', 'osmia',
                      'MaxOfUpperTRange', 'MinOfUpperTRange', 'AverageOfUpperTRange',
                      'MaxOfLowerTRange', 'MinOfLowerTRange', 'AverageOfLowerTRange',
                      'RainingDays', 'AverageRainingDays', 'fruitset', 'fruitmass', 'seeds', 'yield']

# Instantiate LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding to each column
for column in columns_to_encode:
    df[column] = label_encoder.fit_transform(df[column])



# Apply one-hot encoding to each column
df = pd.get_dummies(df, columns=columns_to_encode)



# Display the first few rows of the DataFrame after one-hot encoding
print(df.head())

# Check the data types of the columns
print(df.dtypes)

# Check unique values for each column
for column in df.columns:
    print(f"Unique values in {column}: {df[column].unique()}")
```

```
Unique values in yield_726: [0 1]
Unique values in yield_727: [0 1]
Unique values in yield_728: [0 1]
Unique values in yield_729: [0 1]
Unique values in yield_730: [0 1]
Unique values in yield_731: [0 1]
Unique values in yield_732: [0 1]
Unique values in yield_733: [0 1]
Unique values in yield_734: [0 1]
Unique values in yield_735: [0 1]
Unique values in yield_736: [0 1]
Unique values in yield_737: [0 1]
Unique values in yield_738: [0 1]
Unique values in yield_739: [0 1]
Unique values in yield_740: [0 1]
Unique values in yield_741: [0 1]
Unique values in yield_742: [0 1]
Unique values in yield_743: [0 1]
Unique values in yield_744: [0 1]
Unique values in yield_745: [0 1]
Unique values in yield_746: [0 1]
Unique values in yield_747: [0 1]
Unique values in yield_748: [0 1]
Unique values in yield_749: [0 1]
Unique values in yield_750: [0 1]
Unique values in yield_751: [0 1]
Unique values in yield_752: [0 1]
Unique values in yield_753: [0 1]
Unique values in yield_754: [0 1]
Unique values in yield_755: [0 1]
Unique values in yield_756: [0 1]
Unique values in yield_757: [0 1]
Unique values in yield_758: [0 1]
Unique values in yield_759: [0 1]
Unique values in yield_760: [0 1]
Unique values in yield_761: [0 1]
Unique values in yield_762: [0 1]
Unique values in yield_763: [0 1]
Unique values in yield_764: [0 1]
Unique values in yield_765: [0 1]
Unique values in yield_766: [0 1]
Unique values in yield_767: [0 1]
Unique values in yield_768: [0 1]
Unique values in yield_769: [0 1]
Unique values in yield_770: [0 1]
Unique values in yield_771: [0 1]
Unique values in yield_772: [0 1]
Unique values in yield_773: [0 1]
Unique values in yield_774: [0 1]
Unique values in yield_775: [0 1]
Unique values in yield_776: [0 1]
```

```
# Check the columns in the DataFrame
print(df.columns)
```

```
Index(['Row#_0', 'Row#_1', 'Row#_2', 'Row#_3', 'Row#_4', 'Row#_5', 'Row#_6',
       'Row#_7', 'Row#_8', 'Row#_9',
       ...
       'yield_767', 'yield_768', 'yield_769', 'yield_770', 'yield_771',
       'yield_772', 'yield_773', 'yield_774', 'yield_775', 'yield_776'],
      dtype='object', length=3972)
```

```
# Separate features and target variables
X = df.drop(columns=df.filter(like='yield').columns)  # Exclude the one-hot encoded 'yield' columns
y = df.filter(like='yield')  # Include all the new one-hot encoded 'yield' columns

# Print the first few rows of features and target variables
print("Features:")
print(X.head())

print("\nTarget:")
print(y.head())
```

```
Features:
   Row#_0  Row#_1  Row#_2  Row#_3  Row#_4  Row#_5  Row#_6  Row#_7  Row#_8  \
0       1       0       0       0       0       0       0       0       0
1       0       1       0       0       0       0       0       0       0
2       0       0       1       0       0       0       0       0       0
3       0       0       0       1       0       0       0       0       0
4       0       0       0       0       1       0       0       0       0

   Row#_9  ...  seeds_767  seeds_768  seeds_769  seeds_770  seeds_771  \
0       0  ...          0          0          0          0          0
1       0  ...          0          0          0          0          0
2       0  ...          0          0          0          0          0
3       0  ...          0          0          0          0          0
4       0  ...          0          0          0          0          0

   seeds_772  seeds_773  seeds_774  seeds_775  seeds_776
0          0          0          0          0          0
1          0          0          0          0          0
2          0          0          0          0          0
3          0          0          0          0          0
4          0          0          0          0          0

[5 rows x 3195 columns]

Target:
   yield_0  yield_1  yield_2  yield_3  yield_4  yield_5  yield_6  yield_7  \
0        0        0        0        0        0        0        0        0
1        0        0        0        0        0        0        0        0
2        0        0        0        0        0        0        0        0
3        0        0        0        0        0        0        0        0
4        0        0        0        0        0        0        0        0

   yield_8  yield_9  ...  yield_767  yield_768  yield_769  yield_770  \
0        0        0  ...          0          0          0          0
1        0        0  ...          0          0          0          0
2        0        0  ...          0          0          0          0
3        0        0  ...          0          0          0          0
4        0        0  ...          0          0          0          0

   yield_771  yield_772  yield_773  yield_774  yield_775  yield_776
0          0          0          0          0          0          0
1          0          0          0          0          0          0
2          0          0          0          0          0          0
3          0          0          0          0          0          0
4          0          0          0          0          0          0

[5 rows x 777 columns]
```

Task 3: Modeling and Evaluation

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error


X = df.drop(columns=df.filter(like='yield').columns)
y = df.filter(like='yield')

# Split the 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)


# Create a Linear Regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 0.0013361585157398892
```

```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

```
X_train shape: (621, 3195)
y_train shape: (621, 777)
```

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

# Extract the first target variable for SVR
y_train_single = y_train.iloc[:, 0]
y_test_single = y_test.iloc[:, 0]

# Create a Support Vector Regressor model
svr_model = SVR()

# Train the model on the training data
svr_model.fit(X_train, y_train_single)

# Make predictions on the test set
svr_y_pred_single = svr_model.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)
svr_mse_single = mean_squared_error(y_test_single, svr_y_pred_single)
print(f'Support Vector Regressor Mean Squared Error: {svr_mse_single}')
```

```
Support Vector Regressor Mean Squared Error: 0.01033312492949816
```

```
from sklearn.multioutput import MultiOutputRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# Create a Decision Tree Regressor model
tree_model = DecisionTreeRegressor()

# Wrap the Decision Tree Regressor model in a MultiOutputRegressor
multioutput_tree = MultiOutputRegressor(tree_model)

# Train the model on the training data
multioutput_tree.fit(X_train, y_train)

# Make predictions on the test set
tree_y_pred = multioutput_tree.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)
tree_mse = mean_squared_error(y_test, tree_y_pred)
print(f'MultiOutput Decision Tree Regressor Mean Squared Error: {tree_mse}')
```

```
MultiOutput Decision Tree Regressor Mean Squared Error: 0.0012952512952512953
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_squared_error

# Create a Decision Tree Regressor model
dt_base_model = DecisionTreeRegressor(random_state=42)

# Wrap the base model in a MultiOutputRegressor
multioutput_dt_model = MultiOutputRegressor(dt_base_model)

# Train the model on the training data
multioutput_dt_model.fit(X_train, y_train)

# Make predictions on the test set
dt_y_pred = multioutput_dt_model.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)
dt_mse = mean_squared_error(y_test, dt_y_pred)
print(f'Decision Tree Mean Squared Error: {dt_mse}')
```

```
Decision Tree Mean Squared Error: 0.0012952512952512953
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_squared_error

# Create a Random Forest Regressor model
```

```
rf_base_model = RandomForestRegressor(random_state=42)

# Wrap the base model in a MultiOutputRegressor
multioutput_rf_model = MultiOutputRegressor(rf_base_model)

# Train the model on the training data
multioutput_rf_model.fit(X_train, y_train)

# Make predictions on the test set
rf_y_pred = multioutput_rf_model.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)
rf_mse = mean_squared_error(y_test, rf_y_pred)
print(f'Random Forest Mean Squared Error: {rf_mse}')
```

```
Random Forest Mean Squared Error: 0.0012888963138963142
```

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

# Assuming df is your DataFrame
#correlation_matrix = df.corr()

# Set up the matplotlib figure
#plt.figure(figsize=(12, 10))

# Create a heatmap using Seaborn
#sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=.5)

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

```
Session crashed when executed the above
```

```
  File "<ipython-input-28-3cedb23d147e>", line 1
    Session crashed when executed the above
            ^
SyntaxError: invalid syntax
```

Task 4: Hyperparameter Tuning

```
from sklearn.model_selection import train_test_split

# Split the 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)

# Print the shapes to ensure consistency
print(f'X_train shape: {X_train.shape}, y_train shape: {y_train.shape}')
print(f'X_test shape: {X_test.shape}, y_test shape: {y_test.shape}')
```

```
X_train shape: (621, 3195), y_train shape: (621, 777)
X_test shape: (156, 3195), y_test shape: (156, 777)
```

```
# Last column as the target
y_train_single = y_train.iloc[:, -1]
y_test_single = y_test.iloc[:, -1]
```

```
from sklearn.model_selection import GridSearchCV

from sklearn.multioutput import MultiOutputRegressor
from sklearn.svm import SVR

# Create a Support Vector Regressor model
svr_model = SVR()

# Wrap the SVR model in MultiOutputRegressor
multioutput_svr = MultiOutputRegressor(svr_model)

# Define the hyperparameter grid to search
param_grid = {
    'estimator__C': [0.1, 1, 10],
    'estimator__kernel': ['linear', 'rbf', 'poly'],
    'estimator__gamma': ['scale', 'auto']
}

# Create GridSearchCV
grid_search = GridSearchCV(estimator=multioutput_svr, param_grid=param_grid, scoring='neg_mean_squared_error', cv=5)

# Fit the model to the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Print the best hyperparameters
print(f'Best Hyperparameters: {best_params}')

# Make predictions on the test set using the best model
svr_y_pred = grid_search.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)
svr_mse = mean_squared_error(y_test, svr_y_pred)
print(f'Support Vector Regressor Mean Squared Error: {svr_mse}')
```

```
Best Hyperparameters: {'estimator__C': 0.1, 'estimator__gamma': 'scale', 'estimator__kernel': 'linear'}
Support Vector Regressor Mean Squared Error: 0.003515131926795013
```

Task 5: Explainable AI

```
!pip install shap==0.40.0
```

```
Collecting shap==0.40.0
  Downloading shap-0.40.0.tar.gz (371 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 371.7/371.7 kB 8.4 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from shap==0.40.0) (1.25.2)
```

```
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from shap==0.40.0) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from shap==0.40.0) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from shap==0.40.0) (1.5.3)
Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.10/dist-packages (from shap==0.40.0) (4.66.2)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap==0.40.0) (23.2)
Collecting slicer==0.0.7 (from shap==0.40.0)
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap==0.40.0) (0.58.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap==0.40.0) (2.2.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap==0.40.0) (0.41.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap==0.40.0) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap==0.40.0) (2023.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap==0.40.0) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap==0.40.0) (3.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->shap==0.40.0) (1.16.0)
Building wheels for collected packages: shap
  Building wheel for shap (pyproject.toml) ... done
  Created wheel for shap: filename=shap-0.40.0-cp310-cp310-linux_x86_64.whl size=515069 sha256=4032d5df8578aec2c53d025e335c1ea159b17d4ae4ebf5ea11165a6535cc97ad
  Stored in directory: /root/.cache/pip/wheels/33/28/e3/62a9dc612c58c1b8d1c16fa51e64941bbb38ac8a6decbad39c
Successfully built shap
Installing collected packages: slicer, shap
Successfully installed shap-0.40.0 slicer-0.0.7
```

```python
!pip install shap

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import shap
import numpy as np
import matplotlib.pyplot as plt

import pandas as pd

# Provide the path to your dataset file
file_path = "/content/WildBlueberryPollinationSimulationData.csv"

# Read the dataset into a Pandas DataFrame
df = pd.read_csv(file_path)

# Check the column names
print(df.columns)

# Assuming 'yield' is one of the column names
# If 'yield' is present, define features and target variables
if 'yield' in df.columns:
    features = df.drop('yield', axis=1, errors='ignore')  # Drop 'yield' column if present
    target = df['yield']

    # Create a Linear Regression model
    linear_model = LinearRegression()

    # Assuming you have your dataset loaded and split into features and target variables
    X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

    # Create and fit the linear regression model
    linear_model.fit(X_train, y_train)

    # Define a callable predict function
    def predict_fn(X):
        return linear_model.predict(X)

    # Create an explainer object using KernelExplainer
    explainer = shap.KernelExplainer(predict_fn, X_train)

    # Explain the predictions on a single instance (you can choose any index)
    sample_index = 0
    shap_values = explainer.shap_values(X_train.iloc[sample_index])

    # Reshape shap_values to have two dimensions
    shap_values = shap_values.reshape(1, -1)

    # Force plot for the chosen instance
    shap.force_plot(explainer.expected_value, shap_values, X_train.iloc[sample_index])

    # Summary plot for all instances
    shap.summary_plot(shap_values, X_train, plot_type='bar')  # Use plot_type='bar' to show feature importance

else:
    print("The 'yield' column is not present in the DataFrame.")
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.10/dist-packages (0.44.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from shap) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from shap) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from shap) (1.5.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-packages (from shap) (4.66.2)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap) (23.2)
Requirement already satisfied: slicer==0.0.7 in /usr/local/lib/python3.10/dist-packages (from shap) (0.0.7)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap) (0.58.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap) (0.41.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2023.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (3.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->shap) (1.16.0)
X does not have valid feature names, but LinearRegression was fitted with feature names
WARNING:shap:Using 621 background data samples could cause slower run times. Consider using shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K samples.
Index(['Row#', 'clonesize', 'honeybee', 'bumbles', 'andrena', 'osmia',
       'MaxOfUpperTRange', 'MinOfUpperTRange', 'AverageOfUpperTRange',
       'MaxOfLowerTRange', 'MinOfLowerTRange', 'AverageOfLowerTRange',
       'RainingDays', 'AverageRainingDays', 'fruitset', 'fruitmass', 'seeds',
       'yield'],
      dtype='object')
X does not have valid feature names, but LinearRegression was fitted with feature names
X does not have valid feature names, but LinearRegression was fitted with feature names
```