Milestone 2: Increasing Yield and Production with Machine Learning

Author: Barbara Jean

Date: 06/30/2024

output: pdf_document

In [1]:
```
# Importing the necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statistics
import scipy.stats as st
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
import xgboost as xgb
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
```

In [2]:
```
# Handling warnings as errors
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
# Reading dataset
data=pd.read_csv("C:/Users/79bar/dsc_680/yield_df.csv")
df_yield=pd.DataFrame(data)
print("The loading of the dataset was successful.\n")
```

```
The loading of the dataset was successful.
```

In [4]:
```
df_yield.head() # Reading the first records by using the head() metho
```

Out[4]:

	Unnamed: 0	Area	Item	Year	hg/ha_yield	average_rain_fall_mm_per_year	pesticides_tonnes	avg_temp
0	0	Albania	Maize	1990	36613	1485	121.0	16.37
1	1	Albania	Potatoes	1990	66667	1485	121.0	16.37
2	2	Albania	Rice, paddy	1990	23333	1485	121.0	16.37
3	3	Albania	Sorghum	1990	12500	1485	121.0	16.37
4	4	Albania	Soybeans	1990	7000	1485	121.0	16.37

Data exploration

In [5]:
```
df_yield.shape # Returning a tuple representing the dimensionality of the DataFrame
```

Out[5]:
```
(28242, 8)
```

In [6]:
```
df_yield.size # Returning an int representing the number of elements in this object
```

Out[6]:
```
225936
```

In [7]:
```
df_yield.info() # Printing a summary of the dataframe, index dtype and columns, non-null values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28242 entries, 0 to 28241
Data columns (total 8 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Unnamed: 0                     28242 non-null  int64
 1   Area                           28242 non-null  object
 2   Item                           28242 non-null  object
 3   Year                           28242 non-null  int64
 4   hg/ha_yield                    28242 non-null  int64
 5   average_rain_fall_mm_per_year  28242 non-null  int64
 6   pesticides_tonnes              28242 non-null  float64
 7   avg_temp                       28242 non-null  float64
dtypes: float64(2), int64(4), object(2)
memory usage: 1.7+ MB
```

In [8]:
```
df_yield.dtypes.unique()
```

Out[8]:
```
array([dtype('int64'), dtype('O'), dtype('float64')], dtype=object)
```

In [9]:
```
df_yield.describe() # looking at the statistical summary of the variables with describe()
```

Out[9]:

	Unnamed: 0	Year	hg/ha_yield	average_rain_fall_mm_per_year	pesticides_tonnes	avg_temp
count	28242.000000	28242.000000	28242.000000	28242.00000	28242.000000	28242.000000
mean	14120.500000	2001.544296	77053.332094	1149.05598	37076.909344	20.542627
std	8152.907488	7.051905	84956.612897	709.81215	59958.784665	6.312051
min	0.000000	1990.000000	50.000000	51.00000	0.040000	1.300000
25%	7060.250000	1995.000000	19919.250000	593.00000	1702.000000	16.702500
50%	14120.500000	2001.000000	38295.000000	1083.00000	17529.440000	21.510000
75%	21180.750000	2008.000000	104676.750000	1668.00000	48687.880000	26.000000
max	28241.000000	2013.000000	501412.000000	3240.00000	367778.000000	30.650000

Data cleaning

In [10]:
```
df_yield.isnull().any() # Checking missing data
```

Out[10]:
```
Unnamed: 0                       False
Area                             False
Item                             False
Year                             False
hg/ha_yield                      False
average_rain_fall_mm_per_year    False
pesticides_tonnes                False
avg_temp                         False
dtype: bool
```

In [11]:
```
df_yield.duplicated() # Finding duplicate value in the data
```

Out[11]:
```
0        False
1        False
2        False
3        False
4        False
         ...
28237    False
28238    False
28239    False
28240    False
28241    False
Length: 28242, dtype: bool
```

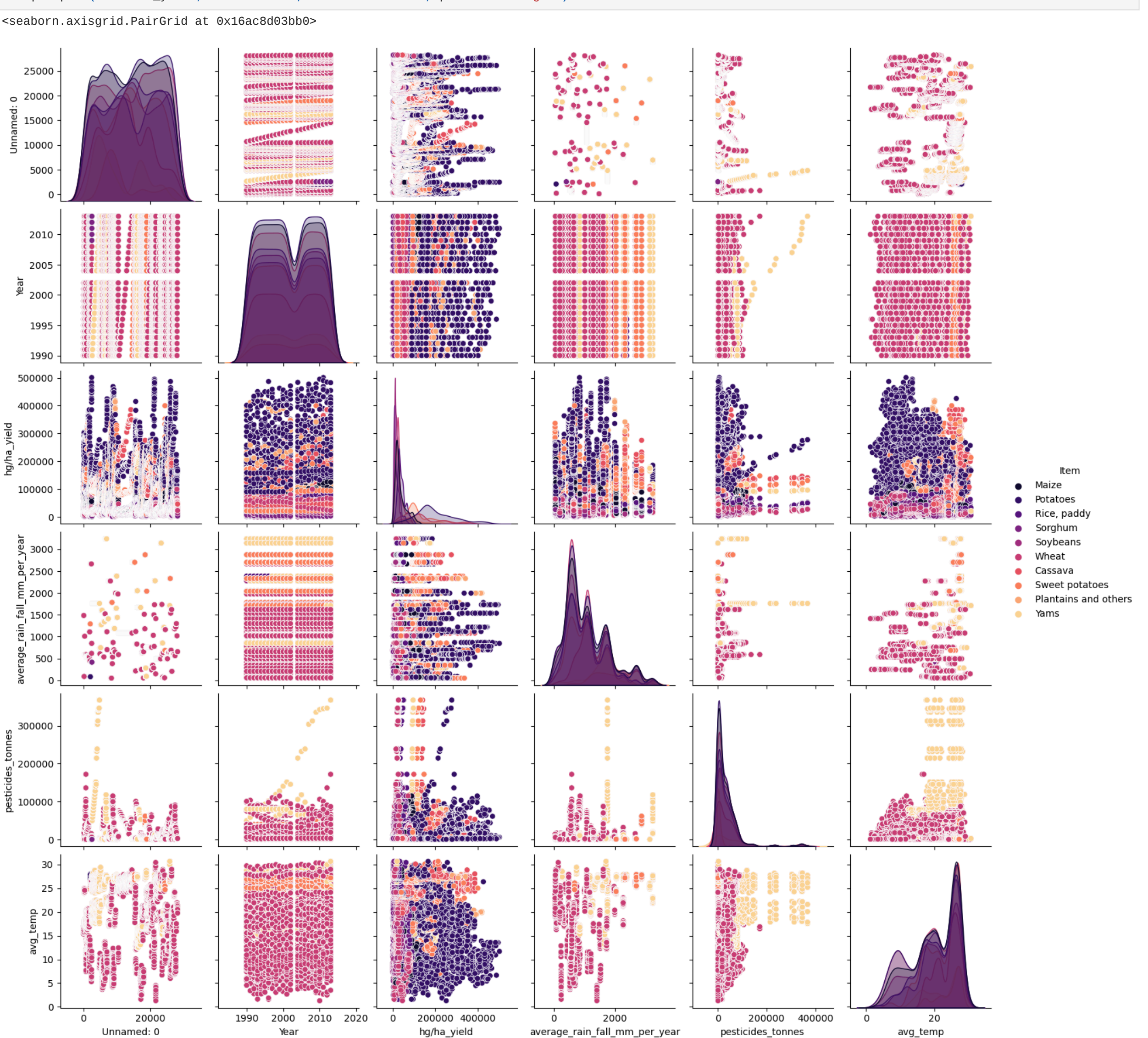
In [12]:
```
df_yield.isnull().sum() # Detecting missing value using interger
```

Out[12]:
```
Unnamed: 0                       0
Area                             0
Item                             0
Year                             0
hg/ha_yield                      0
average_rain_fall_mm_per_year    0
pesticides_tonnes                0
avg_temp                         0
dtype: int64
```

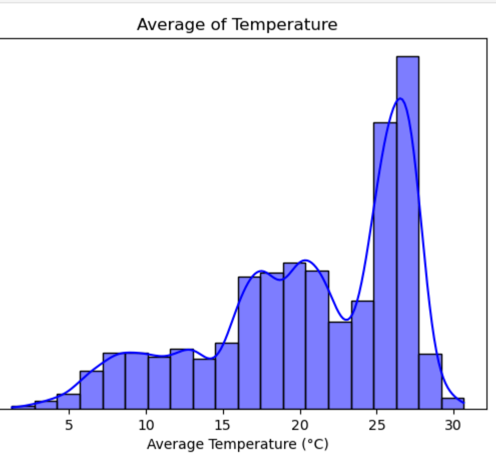
Data visualization

In [13]:
```
sns.pairplot(data =df_yield, hue = 'Item', kind = 'scatter', palette = 'magma')
```

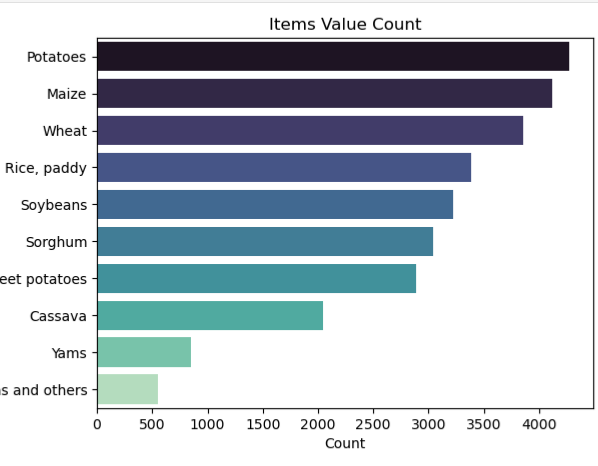
Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x16ac8d03bb0>
```



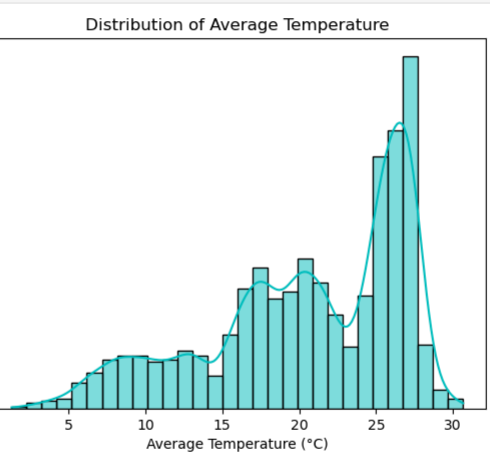
In [14]:
```
# Display avg_temp
plt.figure()
sns.histplot(df_yield['avg_temp'], bins=20, kde=True, color='b')
plt.title('Average of Temperature')
plt.xlabel('Average Temperature (°C)')
plt.ylabel('Frequency')
plt.show()
```



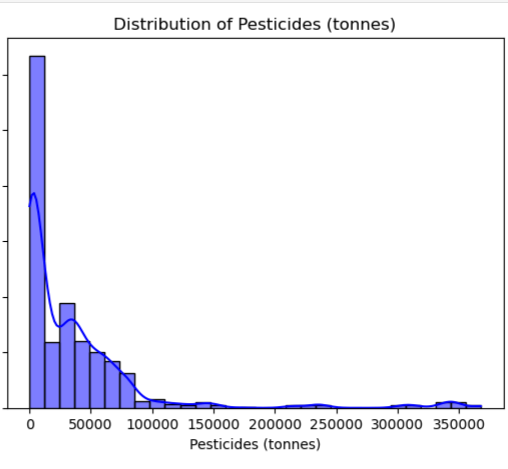
In [15]:
```
# Display "Item" using barplot
plt.figure()
item_value= df_yield['Item'].value_counts()
sns.barplot(x=item_value.values, y=item_value.index, palette='mako')
plt.title('Items Value Count')
plt.xlabel('Count')
plt.ylabel('Item')
plt.show()
```



In [16]:
```
## avg_temp (Histogram)
plt.figure()
sns.histplot(df_yield['avg_temp'], bins=30, kde=True, color='c')
plt.title('Distribution of Average Temperature')
plt.xlabel('Average Temperature (°C)')
plt.ylabel('Frequency')
plt.show()
```



In [17]:
```
## pesticides_tonnes (Histogram)
plt.figure()
sns.histplot(df_yield['pesticides_tonnes'], bins=30, kde=True, color='b')
plt.title('Distribution of Pesticides (tonnes)')
plt.xlabel('Pesticides (tonnes)')
plt.ylabel('Frequency')
plt.show()
```



In [18]:
```
corr_Mat=df_yield.corr().abs()
plt.figure(figsize=(8,6))
sns.heatmap(corr_Mat,annot=True)
```

Out[18]:
```
<AxesSubplot:>
```

In [19]:
```
# Drop the index column if it's not needed
yield_clean= df_yield.drop(columns=["Unnamed: 0"])
```

In [20]:
```
yield_clean.head(5)
```

Out[20]:

	Area	Item	Year	hg/ha_yield	average_rain_fall_mm_per_year	pesticides_tonnes	avg_temp
0	Albania	Maize	1990	36613	1485	121.0	16.37
1	Albania	Potatoes	1990	66667	1485	121.0	16.37
2	Albania	Rice, paddy	1990	23333	1485	121.0	16.37
3	Albania	Sorghum	1990	12500	1485	121.0	16.37
4	Albania	Soybeans	1990	7000	1485	121.0	16.37

Deal with inconsistent values

In [21]:
```
# Converting categorical data into dummy or indicator variables
yield_clean_dummies =pd.get_dummies(yield_clean, columns=['Area','Item'],drop_first=True)
yield_clean_dummies.head()
```

Out[21]:

	Year	hg/ha_yield	average_rain_fall_mm_per_year	pesticides_tonnes	avg_temp	Area_Algeria	Area_Angola	Area_Argentina	Area_Armenia	Area_Australia	...	Area_Zimbabwe	Item_Maize	Item_Plant and ot
0	1990	36613	1485	121.0	16.37	0	0	0	0	0	...	0	1	
1	1990	66667	1485	121.0	16.37	0	0	0	0	0	...	0	0	
2	1990	23333	1485	121.0	16.37	0	0	0	0	0	...	0	0	
3	1990	12500	1485	121.0	16.37	0	0	0	0	0	...	0	0	
4	1990	7000	1485	121.0	16.37	0	0	0	0	0	...	0	0	

5 rows × 114 columns

In [22]:
```
# Creating features and target
X=yield_clean_dummies.drop(['hg/ha_yield'],axis=1)
y=yield_clean_dummies['hg/ha_yield']
```

In [23]:
```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

1- Training the model by using **Linear Regression**

In [24]:
```
# Creating an instance of the LinearRegression class called lr
lr= LinearRegression()
# Fitting the model to the data
lr.fit(X_train, y_train)
```

Out[24]:
```
LinearRegression()
```

In [25]:
```
# Applying the model to make a prediction
y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)
```

Evaluate model performance Linear Regression

In [26]:
```
# Evaluate the models
def evaluate_model(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    return mse, r2
```

In [ ]:

In [27]:
```
# Calling the evaluate_models function
mse_train, r2_train = evaluate_model(y_train, y_train_pred)
mse_test, r2_test = evaluate_model(y_test, y_test_pred )
```

In [28]:
```
print("Yield and Production Prediction Train - MSE (Linear Regression):", '%.2f' %mse_train)
print("Yield and Production Prediction Train - R-squared (Linear Regression):", '%.2f' %r2_train)
```

```
Yield and Production Prediction Train - MSE (Linear Regression): 1764522894.09
Yield and Production Prediction Train - R-squared (Linear Regression): 0.76
```

In [29]:
```
print("Yield and Production Prediction Test - MSE (Linear Regression):", '%.2f' %mse_test)
print("Yield and Production Prediction Test - R-squared (Linear Regression):", '%.2f' %r2_test)
```

```
Yield and Production Prediction Test - MSE (Linear Regression): 1752035583.81
Yield and Production Prediction Test - R-squared (Linear Regression): 0.76
```

2- Training the model by using Random Forest Regression model

In [30]:
```
# Creating an instance of the Random Forest Regression model class called rf
rf= RandomForestRegressor(max_depth=2, random_state=0)
rf.fit(X_train, y_train)
```

Out[30]:
```
RandomForestRegressor(max_depth=2, random_state=0)
```

In [31]:
```
# Applying the model to make a prediction
y_rf_train_pred = rf.predict(X_train)
y_rf_test_pred = rf.predict(X_test)
```

Evaluate model performance Random Forest Regression

In [32]:
```
# Calling the evaluate_models function
mse_dst_train, r2_dst_train = evaluate_model(y_train, y_train_pred)
mse_dst_test, r2_rf_test = evaluate_model(y_test, y_test_pred )
```

In [33]:
```
print("Yield and Production Prediction Train - MSE (Random Forest Regression):", '%.2f' %mse_dst_train)
print("Yield and Production Prediction Train - R-squared (Random Forest Regression):", '%.2f' %r2_dst_train)
```

```
Yield and Production Prediction Train - MSE (Random Forest Regression): 1764522894.09
Yield and Production Prediction Train - R-squared (Random Forest Regression): 0.76
```

In [34]:
```
print("Yield and Production Prediction Test - MSE (Random Forest Regression):", '%.2f' %mse_dst_test)
print("Yield and Production Prediction Test - R-squared (Random Forest Regression):", '%.2f' %r2_rf_test)
```

```
Yield and Production Prediction Test - MSE (Random Forest Regression): 1752035583.81
Yield and Production Prediction Test - R-squared (Random Forest Regression): 0.76
```

3- Training the model by using **Decision Tree Regressor model**

In [35]:
```
# Creating an instance of the Decision Tree Regresso class called dst
dst= DecisionTreeRegressor(random_state = 0)
dst.fit(X_train, y_train)
```

Out[35]:
```
DecisionTreeRegressor(random_state=0)
```

In [36]:
```
# Applying the model to make a prediction
y_dst_train_pred = dst.predict(X_train)
y_dst_test_pred = dst.predict(X_test)
```

Evaluate model performance Decision Tress Regressor

In [37]:
```
# Calling the evaluate_models function
mse_dst_train, r2_dst_train = evaluate_model(y_train, y_train_pred)
mse_dst_test, r2_rf_test = evaluate_model(y_test, y_test_pred )
```

In [38]:
```
print("Yield and Production Prediction Train - MSE (Decision Tree Regressor):", '%.2f' %mse_dst_train)
print("Yield and Production Prediction Train - R-squared (Random Forest Regression):", '%.2f' %r2_dst_train)
```

```
Yield and Production Prediction Train - MSE (Decision Tree Regressor): 1764522894.09
Yield and Production Prediction Train - R-squared (Random Forest Regression): 0.76
```

In [39]:
```
print("Yield and Production Prediction Test - MSE (Decision Tree Regressor):", '%.2f' %mse_dst_test)
print("Yield and Production Prediction Test - R-squared (Decision Tree Regressor):", '%.2f' %r2_rf_test)
```

```
Yield and Production Prediction Test - MSE (Decision Tree Regressor): 1752035583.81
Yield and Production Prediction Test - R-squared (Decision Tree Regressor): 0.76
```

In [ ]: