Crop Prediction Model Notes

Notes about the crop prediction model

step 1

```
import piplite
await piplite.install('seaborn')
import numpy as np
import pandas as pd
```

`numpy` library, a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, as well as a collection of mathematical functions to operate on these arrays efficiently

`pandas` library, which is a powerful data manipulation and analysis tool.

step 2

```
crop = pd.read_csv("crop.csv")
crop.head()
```

This line reads the CSV file named "crop.csv" using the `read_csv()`**, file should be located in the current working directory**

`head()` function is used to display the first five rows of the DataFrame by default. This helps to get a quick overview of the data and its structure.

step 3

```
crop.shape
```

which represents the dimensions of the data frames in terms of rows and columns

(1st value is row,2nd is value col)

step 4

```
crop.info()
```

Provide summary of the Dataframes.

- The total number of rows in the DataFrame.
- The column names and their respective data types.
- The count of non-null values for each column, which indicates the number of available values.
- The data type and memory usage information.

step 5

```
crop.isnull().sum()
crop.duplicated().sum()
```

return count of null values and return count of duplicated values

step 6

```
crop.describe()
```

`crop.describe()` provides a statistical summary

- Count: The number of non-null values in the column.
- Mean: The average value of the column.

- Standard deviation (std): The measure of the dispersion or variability in the column's values.
- Minimum: The smallest value in the column.
- 25th percentile (25%): The value below which 25% of the data falls.
- 50th percentile (50%)/Median: The middle value of the column when sorted in ascending order.
- 75th percentile (75%): The value below which 75% of the data falls.
- Maximum: The largest value in the column.

step 7

```
corr = crop.corr()
corr
```

1. The code `corr = crop.corr()` calculates the correlation coefficient between all pairs of columns in the `crop` **DataFrame**. This coefficient measures the strength and direction of the linear relationship between two variables. The resulting correlation matrix is stored in the `corr` DataFrame.
2. The values of the correlation coefficient range from -1 to 1. A value of -1 indicates a strong negative correlation, 0 indicates no correlation, and 1 indicates a strong positive correlation.
3. By analyzing the correlation matrix, you can study the relationships between different variables in the `crop` **DataFrame**. This can help you understand the dependencies and patterns among the features, identify potential multicollinearity, and select relevant features for further analysis or modeling.

step 8

```
import seaborn as sns
sns.heatmap(corr,annot=True,cbar=True,cmap='coolwarm')
```

1. To create a heatmap visualization of the correlation matrix stored in the `corr` DataFrame, use the Seaborn library.

- `corr`: The correlation matrix (`corr` DataFrame) that you calculated earlier.
- `annot=True`: This parameter displays the correlation coefficient values on the heatmap.
- `cbar=True`: This parameter includes a colorbar to indicate the color scale.
- `cmap='coolwarm'`: This parameter sets the color palette for the heatmap. In this case, 'coolwarm' is used, which ranges from cool colors (blue) indicating negative correlation to warm colors (red) indicating positive correlation.

step 9

```
crop['label'].value_counts()
```

1. `crop['label']`: This code accesses the 'label' column of the `crop` DataFrame. Assuming that 'label' is a column in the DataFrame, `crop['label']` retrieves the values in that column.
2. `.value_counts()`: This method is applied to the 'label' column, and it counts the occurrences of each unique value in the column. It returns a series where the unique values in the 'label' column are displayed as the index, and the corresponding counts of each unique value are shown as the values.

step 10

```
import matplotlib.pyplot as plt
sns.distplot(crop['N'])
plt.show()
```

1. The `matplotlib.pyplot` and `seaborn` libraries are used to create a distribution plot (histogram) of the 'N' column in the `crop` DataFrame.
2. To generate a distribution plot of the 'N' column, use a histogram to display the distribution of numerical values along the x-axis and the frequency of occurrence

along the y-axis.

step 11

```
crop_dict = {'rice':1, 'maize':2, 'jute':3, 'cotton':4, 'coconut':5, 'papaya':6, 'orange':
7, 'apple':8, 'muskmelon':9, 'watermelon':10, 'grapes':11, 'mango':12, 'banana':13, 'pomeg
ranate':14, 'lentil':15, 'blackgram':16, 'mungbean':17, 'mothbeans':18, 'pigeonpeas':19,
 'kidneybeans':20, 'chickpea':21, 'coffee':22}

crop['crop_num'] = crop['label'].map(crop_dict)
```

1. Mapping can be useful for representing categorical crop names as numerical values, which can be used for further analysis or modeling purposes.
2. For example, if a row in the 'label' column has the value 'rice', the corresponding row in the 'crop_num' column will have a value of 1.

step 12

```
crop.drop('label',axis=1,inplace=True)
crop.head()
```

To remove the "label" column from the `crop` DataFrame, use the `drop()` function with `axis=1` as the parameter to indicate that the column is being dropped. Use `inplace=True` to ensure that the change is applied directly to the DataFrame.

- **`'label'`**: The column name to be dropped.
- **`axis=1`**: Specifies that the column is being dropped. **`axis=1`** refers to columns, while **`axis=0`** refers to rows.
- **`inplace=True`**: This parameter ensures that the change is applied to the **`crop`** DataFrame directly, modifying it.

step 13

```
x = crop.drop('crop_num',axis=1)
y = crop['crop_num']
```

1. `x = crop.drop('crop_num', axis=1)`: This line creates a new DataFrame `x` by dropping the 'crop_num' column from the `crop` DataFrame. The `drop()` function is called on the DataFrame (`crop`) with the following parameters:

   - `'crop_num'`: The column name to be dropped.
   - `axis=1`: Specifies that the column is being dropped (along the columns axis).

   The resulting DataFrame `x` will contain all the columns from the original `crop` DataFrame except for the 'crop_num' column, representing the feature dataset.

2. `y = crop['crop_num']`: This line assigns the 'crop_num' column from the `crop` DataFrame to the variable `y`. The column represents the target variable or the labels for the crop classification task.

3. You have the features stored in the `x` DataFrame and the corresponding labels stored in the `y` Series. This separation enables you to perform machine learning tasks, such as training a model using `x` as input and `y` as the target variable.

step 14

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
```

- `x`: The feature dataset.
- `y`: The target variable.
- `test_size=0.2`: This parameter specifies the proportion of the dataset that will be allocated for testing. In this case, it is set to 0.2, which means 20% of the data will be used for testing, and the remaining 80% will be used for training.
- `random_state=42`: This parameter sets the random seed to ensure reproducibility. The same seed value will result in the same split each time the code is executed.

step 15

```
from sklearn.preprocessing import MinMaxScaler
ms = MinMaxScaler()

ms.fit(x_train)
```

```
x_train = ms.transform(x_train)
x_test = ms.transform(x_test)
```

1. `ms.fit(x_train)`: This line fits the `MinMaxScaler` to the training set (`x_train`). The scaler calculates the minimum and maximum values for each feature in the training set.

1. `ms.fit(x_train)`: This line fits the `MinMaxScaler` to the training set (`x_train`). The scaler calculates the minimum and maximum values for each feature in the training set.
2. `x_train = ms.transform(x_train)`: This line applies the scaling transformation to the training set (`x_train`). The `transform()` method scales the features in `x_train` using the calculated minimum and maximum values. The transformed data is then assigned back to `x_train`.
3. `x_test = ms.transform(x_test)`: This line applies the same scaling transformation to the testing set (`x_test`). The `transform()` method scales the features in `x_test` based on the minimum and maximum values calculated from the training set. The transformed data is then assigned back to `x_test`
4. Min-max scaling scales the features to a specific range

step 16

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

sc.fit(x_train)
x_train = sc.transform(x_train)
x_test = sc.transform(x_test)
```

1. `sc = StandardScaler()`: This line creates an instance of the `StandardScaler` class and assigns it to the variable `sc`.
2. `sc.fit(x_train)`: This line fits the `StandardScaler` to the training set (`x_train`). The scaler computes the mean and standard deviation for each feature in the training set.
3. `x_train = sc.transform(x_train)`: This line applies the standardization transformation to the training set (`x_train`). The `transform()` method standardizes the features in

`x_train` using the computed mean and standard deviation. The transformed data is then assigned back to `x_train`.

4. `x_test = sc.transform(x_test)`: This line applies the same standardization transformation to the testing set (`x_test`). The `transform()` method standardizes the features in `x_test` based on the mean and standard deviation computed from the training set. The transformed data is then assigned back to `x_test`.
5. Standardization scales the features to have a mean of 0 and a standard deviation of 1. This type of scaling ensures that the features have zero mean and similar variance, which can be beneficial for certain algorithms that assume normally distributed features or when regularization is applied.

step 17

```
# importing model
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

# create instances of all models
models = {
    'Logistic Regression':LogisticRegression(),
    'Naive Bayes':GaussianNB(),
    'Support Vector Machine':SVC(),
    'K-Nearest Neighbors':KNeighborsClassifier(),
    'Decision Tree':DecisionTreeClassifier(),
    'Random Forest':RandomForestClassifier(),
    'Bagging':BaggingClassifier(),
    'AdaBoost':AdaBoostClassifier(),
    'Gradient Boosting':GradientBoostingClassifier(),
    'Extra Trees':ExtraTreeClassifier(),
}

for name,md in models.items():
    md.fit(x_train,y_train)
    ypred = md.predict(x_test)

    print(f"{name} with accuracy : {accuracy_score(y_test,ypred)}")
```

- `md.fit(x_train, y_train)`: The model is trained using the `fit()` method, with the training set `x_train` and corresponding labels `y_train`.
- `ypred = md.predict(x_test)`: The model predicts the labels for the testing set `x_test` using the `predict()` method, and the predicted labels are stored in `ypred`.
- `accuracy_score(y_test, ypred)`: The accuracy of the model's predictions is computed by comparing the predicted labels (`ypred`) with the true labels from the testing set (`y_test`) using the `accuracy_score()` function.
- `print(f"{name} with accuracy: {accuracy_score(y_test, ypred)}")`: The name of the model and its corresponding accuracy score are printed.

step 18

```
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
ypred = rfc.predict(x_test)
accuracy_score(y_test,ypred)
```

1. `rfc = RandomForestClassifier()`: This line creates an instance of the Random Forest Classifier and assigns it to the variable `rfc`. The classifier is initialized with default parameters.
2. `rfc.fit(x_train, y_train)`: This line trains the Random Forest Classifier using the `fit()` method. The method takes the training features (`x_train`) and their corresponding labels (`y_train`) as inputs.
3. `ypred = rfc.predict(x_test)`: This line uses the trained Random Forest Classifier (`rfc`) to make predictions on the testing features (`x_test`). The `predict()` method predicts the labels based on the learned patterns in the training data and assigns the predicted labels to the variable `ypred`.
4. `accuracy_score(y_test, ypred)`: This line calculates the accuracy score of the predicted labels (`ypred`) by comparing them with the true labels from the testing set (`y_test`). The `accuracy_score()` function from scikit-learn's metrics module is used to compute the accuracy

step 19

```
# predictive system
def recommendation(N,P,K,temperature,humidity,ph,rainfall):
    features = np.array([[N,P,K,temperature,humidity,ph,rainfall]])
    prediction = rfc.predict(features).reshape(1,-1)

    return prediction[0] #1 2 3 ... 22
```

1. `features = np.array([[N, P, K, temperature, humidity, ph, rainfall]])`: This line creates a NumPy array with a shape of (1, 7) containing the input features for crop prediction. The input values are provided as arguments to the function (`N`, `P`, `K`, `temperature`, `humidity`, `ph`, `rainfall`).
2. `prediction = rfc.predict(features).reshape(1, -1)`: This line uses the trained Random Forest Classifier (`rfc`) to predict the crop based on the provided input features. The `predict()` method predicts the crop label for the given features. The result is reshaped to have a shape of (1, -1) to ensure compatibility with the return statement.
3. `return prediction[0]`: This line returns the predicted crop label as the output of the function. The predicted label is accessed using indexing (`prediction[0]`) to retrieve the value.