Project 2 : Crop Recommendation Analysis

Source Code :

```
#Importing libraries
import pandas as pd
import numpy as np
#Uploading files
from google.colab import files
uploaded=files.upload()

#Reading the whole dataset
df = pd.read_csv('Crop_recommendation.csv')

#Displaying the dataset
df

#Printing the size
df.size

#Printing the shape
df.shape

#Dropping NaN values
df.dropna()

#Importing columns of dataset
df.columns

#Importing unique levels from dataset
df['label'].unique()

#Importing all data types
df.dtypes

#Importing lebels/columns
df['label'].value_counts()

#Importing One-Hot-Encoder
from sklearn.preprocessing import OneHotEncoder
onehot=OneHotEncoder()
result=onehot.fit_transform(df[['label']])
print(result)
```

```
df1 = df.join(pd.DataFrame(result.toarray(), columns = onehot.categories_))
df1

#Dropping the NaN Values
df1.dropna()

#Dropping the duplicate values
df1.drop_duplicates()

#Dropping unnecessary columns
df1.drop(["label"], axis=1)

#Calculating the size
df1.size

#Calculating the shape
df1.shape

#Find out the columns of dataset
df1.columns

#Importing all data types of dataset
df1.dtypes

#Creating new dataset by dropping some columns not necessary
df2 = df1.drop(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall',"label"], axis=1)
df2

#Importing OneHotEncoder to convert data into a specific group
from mlxtend.preprocessing import TransactionEncoder
encoder = TransactionEncoder().fit(df2)
onehot1 = encoder.transform(df2)
onehot2 = pd.DataFrame(onehot1,columns = encoder.columns_)
onehot2

import pandas as pd
from sklearn.metrics import jaccard_score
from sklearn.feature_extraction.text import TfidfVectorizer

# Assuming your dataset has a column named 'text' containing text data
```

```
texts = df['label']

# Use TfidfVectorizer to convert text data to TF-IDF vectors
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(texts)

# Calculate Jaccard Distance
jaccard_distance = 1 - jaccard_score(tfidf_matrix, tfidf_matrix, average='samples')

print(f"Jaccard Distance: {jaccard_distance}")

from scipy.spatial import distance
print(onehot2.mean())

# Install mlxtend library
!pip install mlxtend

# Import necessary libraries
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori

# Sample dataset (replace this with your own dataset)


# Transform the dataset into a binary format
te = TransactionEncoder()
te_ary = te.fit(df).transform(df)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori algorithm to find frequent itemsets
min_support = 0.2  # Adjust the minimum support as needed
frequent_itemsets = apriori(df, min_support=min_support, use_colnames=True)

# Display the frequent itemsets
frequent_itemsets

# Specify the column for which you want to find unique values and their counts
target_column='label'
# Get unique values and their counts
unique_values_counts = df[target_column].value_counts()
```

```
# Display the results
print("Unique Values and Their Counts in '{}' Column:".format(target_column))
print(unique_values_counts)
```

Project Explanation :

This Python program appears to perform several data analysis and preprocessing tasks using the Pandas library in a Jupyter notebook or Colab environment. Let's break down the code step by step :

1. **Importing Libraries :**

```
import pandas as pd
import numpy as np
from google.colab import files
```

- Imports necessary libraries: Pandas for data manipulation, NumPy for numerical operations, and the Google Colab **files** module for file upload.

2. **Uploading Files :**

```
uploaded=files.upload()
```

- Allows the user to upload files. The uploaded files are not visible in the provided code snippet.

3. **Reading the Dataset :**

```
df = pd.read_csv('Crop_recommendation.csv')
```

- Reads a CSV file named 'Crop_recommendation.csv' into a Pandas DataFrame called **df**.

4. **Displaying the Dataset :**

```
df
```

- Prints the entire DataFrame to display its contents.

5. **Printing the Size and Shape :**

```
df.size
df.shape
```

- Prints the size (total number of elements) and shape (number of rows and columns) of the DataFrame.

6. **Dropping NaN Values :**

```
df.dropna()
```

- Drops rows containing NaN (missing) values from the DataFrame. Note: This operation doesn't modify the original DataFrame unless explicitly assigned.

7. **Importing Columns of Dataset :**

```
df.columns
```

- Prints the column names of the DataFrame.

8. **Importing Unique Levels from Dataset**

```
df['label'].unique()
```

- Prints the unique values present in the 'label' column of the DataFrame.

9. Importing All Data Types :

```
df.dtypes
```

- Prints the data types of each column in the DataFrame.

10. Importing Labels/Columns :

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

- Prints the counts of each unique value in the 'label' column.

11. Importing One-Hot-Encoder :

```
from sklearn.preprocessing import OneHotEncoder
onehot=OneHotEncoder()
result=onehot.fit_transform(df[['label']])
print(result)
```

- Uses scikit-learn's **OneHotEncoder** to convert the 'label' column into one-hot encoded format.

12. Creating a New DataFrame with One-Hot Encoding :

```
df1 = df.join(pd.DataFrame(result.toarray(), columns = onehot.categories_))
df1
```

- Joins the one-hot encoded values to the original DataFrame and creates a new DataFrame called **df1**.

13. Dropping NaN Values, Duplicate Values and Unnecessary Columns :

```
df1.dropna()
df1.drop_duplicates()
df1.drop(["label"], axis=1)
```

- These operations are not assigned to variables, so they don't modify the DataFrame in place.

14. Calculating the Size and Shape :

```
df1.size
df1.shape
```

- Prints the size and shape of the new DataFrame (**df1**).

15. Find Out the Columns of the Dataset :

```
df1.columns
```

- Prints the column names of **df1**.

16. Importing All Data Types of the Dataset :

```
df1.dtypes
```

- Prints the data types of each column in **df1**.

17. Creating a New Dataset by Dropping Some Columns :

```
df2 = df1.drop(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall',"label"], axis=1)
df2
```

- Creates a new DataFrame (**df2**) by dropping specific columns from **df1**.

18. Using One Hot Encoder to Convert Data into a Specific Group :

```
from mlxtend.preprocessing import TransactionEncoder
encoder = TransactionEncoder().fit(df2)
onehot1 = encoder.transform(df2)
onehot2 = pd.DataFrame(onehot1, columns = encoder.columns_)
onehot2
```

- Uses **TransactionEncoder** from the **mlxtend** library to convert data into a binary format and creates a new DataFrame (**onehot2**).

19. Jaccard Distance Calculation :

```
from sklearn.metrics import jaccard_score
from sklearn.feature_extraction.text import TfidfVectorizer

texts = df['label']
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(texts)
jaccard_distance = 1 - jaccard_score(tfidf_matrix, tfidf_matrix, average='samples')
print(f"Jaccard Distance: {jaccard_distance}")
```

- Calculates the Jaccard distance for text data in the 'label' column using TF-IDF vectors.

20. Calculating Mean of Columns :

```
from scipy.spatial import distance
print(onehot2.mean())
```

- Calculates the mean of each column in the DataFrame (**onehot2**).

21. Installing mlxtend Library :

```
!pip install mlxtend
```

- Installs the **mlxtend** library if it's not already installed.

22. Apriori Algorithm for Frequent Itemsets

```
from mlxtend.frequent_patterns import apriori
min_support = 0.2
frequent_itemsets = apriori(df, min_support=min_support, use_colnames=True)
frequent_itemsets
```

- Applies the Apriori algorithm to find frequent itemsets in the DataFrame (**df**).

23. Unique Values and Their Counts :

```
target_column='label'
unique_values_counts = df[target_column].value_counts()
```

```
print("Unique Values and Their Counts in '{}' Column:".format(target_column))
print(unique_values_counts)
```

- Prints unique values and their counts in the specified column (**'label'**).

Please note that the code has various standalone operations that don't modify the original DataFrames unless explicitly assigned. If you need the modified DataFrames, consider assigning the results to new variables.