```
# https://drive.google.com/file/d/1HyMVlPB50Tjz0GtCTkQaBMHD1qioJGsn/view?usp=sharing

!pip install gdown
import gdown
file_id = '1HyMVlPB50Tjz0GtCTkQaBMHD1qioJGsn'
url = f'https://drive.google.com/uc?id={file_id}'
output = 'ICRISAT_Merge_Crop_Climate_Cost.csv'
gdown.download(url, output, quiet=False)
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.12.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.16.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.67.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.6)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.4.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2024.12.14
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
Downloading...
From: https://drive.google.com/uc?id=1HyMVlPB50Tjz0GtCTkQaBMHD1qioJGsn
To: /content/ICRISAT_Merge_Crop_Climate_Cost.csv
100%|██████████| 787k/787k [00:00<00:00, 43.4MB/s]
'ICRISAT_Merge_Crop_Climate_Cost.csv'
```

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Load the dataset
data = pd.read_csv('/content/Crop_Data_Final.csv')

# Define climate and crop columns
climatic_columns = ["Min Temp (Centigrate)", "Max Temp (Centigrate)", "Precipitation (mm)"]
crop_columns = [col for col in data.columns if any(crop in col for crop in [
    "RICE", "WHEAT", "SORGHUM", "PEARL MILLET", "MAIZE", "FINGER MILLET",
    "BARLEY", "CHICKPEA", "PIGEONPEA", "GROUNDNUT", "SESAMUM",
    "RAPESEED AND MUSTARD", "SAFFLOWER", "CASTOR", "LINSEED",
    "SUNFLOWER", "SOYABEAN", "OILSEEDS", "SUGARCANE", "COTTON"
])]

# Dictionary to store association rules for each crop
crop_rules = {}

for crop in crop_columns:
    # Discretize crop production data
    data[f'{crop}_Category'] = pd.cut(data[crop], bins=3, labels=["Low", "Medium", "High"])

    for climate in climatic_columns:
        # Discretize climatic parameters
        data[f'{climate}_Category'] = pd.cut(data[climate], bins=3, labels=["Low", "Medium", "Hig

    # One-hot encode the discretized categories
    categories = [f'{crop}_Category'] + [f'{climate}_Category' for climate in climatic_columns]
    one_hot_data = pd.get_dummies(data[categories])

    # Apply Apriori and extract association rules
    frequent_itemsets = apriori(one_hot_data, min_support=0.1, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

    # Store the extracted rules for each crop
```

```
    crop_rules[crop] = rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']]
    print(f"\nRules for {crop}:")
    print(crop_rules[crop].head())  # Display first few rules for each crop
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-0d9002162dcf> in <cell line: 19>()
     31     # Apply Apriori and extract association rules
     32     frequent_itemsets = apriori(one_hot_data, min_support=0.1, use_colnames=True)
---> 33     rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
     34
     35     # Store the extracted rules for each crop

TypeError: association_rules() missing 1 required positional argument: 'num_itemsets'
```

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Load the dataset
data = pd.read_csv('/content/Crop_Data_Final.csv')
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
```

```
data.head(5)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
```

	DistCode	Year	DistName	RiceArea	RiceProduction	RiceYield	WheatArea	WheatProduction	WheatYield	MaizeArea	...	phosphateShar
0	99	1966	Nasik	38	22	579	86	38	442	3	...	
1	99	1967	Nasik	36	38	1056	86	46	535	1	...	
2	99	1968	Nasik	34	34	1000	89	49	549	2	...	
3	99	1969	Nasik	34	24	703	79	40	504	2	...	
4	99	1970	Nasik	35	36	1040	78	56	723	2	...	

5 rows × 42 columns

skip the cell

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/ICRISAT_Merge_Crop_Climate_Cost.csv')
# Preprocess: Binning continuous variables
bins = [0, 500, 1000, 1500, float('inf')]
labels = ['Low', 'Moderate', 'High', 'Very High']
df['Rice_Yield_Category'] = pd.cut(df['RICE YIELD (Kg per ha)'], bins=bins, labels=labels)

# Convert to transactional format
transactions = df.apply(lambda row: [f"{col}={row[col]}" for col in df.columns if pd.notnull(row[c

# Encode data for Apriori
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)
```

```
# Apply Apriori
frequent_itemsets = apriori(df_encoded, min_support=0.1, use_colnames=True)

# Generate Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)

# Analyze Rules
filtered_rules = rules[rules['lift'] > 1.2]  # Example filter for meaningful rules
print(filtered_rules)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
```

skip the cell

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/ICRISAT_Merge_Crop_Climate_Cost.csv')

# Preprocessing: Binning continuous variables for RICE YIELD
bins_rice_yield = [0, 500, 1000, 1500, float('inf')]
labels_rice_yield = ['Low', 'Moderate', 'High', 'Very High']
df['Rice_Yield_Category'] = pd.cut(df['RICE YIELD (Kg per ha)'], bins=bins_rice_yield, labels=lab

# Repeat binning for other relevant yield columns if necessary
bins_precipitation = [0, 200, 400, 600, float('inf')]
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l

# Add similar bins for temperature and other numerical columns, as needed.

# Convert to transactional format: Each row becomes a list of attribute-value pairs
transactions = df.apply(lambda row: [f"{col}={row[col]}" for col in df.columns if pd.notnull(row[

# Encode data for Apriori
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply Apriori
frequent_itemsets = apriori(df_encoded, min_support=0.1, use_colnames=True)

# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)

# Analyze Rules: Filtering meaningful rules
filtered_rules = rules[(rules['lift'] > 1.2) & (rules['confidence'] > 0.7)]
print(filtered_rules)

# Save rules to a CSV for further analysis
```

```
#filtered_rules.to_csv('association_rules.csv', index=False)
```

```
!pip install --upgrade mlxtend
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Requirement already satisfied: mlxtend in /usr/local/lib/python3.10/dist-packages (0.23.3)
Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.13.1)
Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.26.4)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (2.2.2)
Requirement already satisfied: scikit-learn>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.5.2)
Requirement already satisfied: matplotlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (3.8.0)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.4.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (4.55.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (2.8.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->mlxtend) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->mlxtend) (2024.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.3.1->mlxtend) (3.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxt
```

skip the cell

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/ICRISAT_Merge_Crop_Climate_Cost.csv')

# Preprocessing: Binning for selected variables
bins_rice_yield = [0, 500, 1000, 1500, float('inf')]
labels_rice_yield = ['Low', 'Moderate', 'High', 'Very High']
df['Rice_Yield_Category'] = pd.cut(df['RICE YIELD (Kg per ha)'], bins=bins_rice_yield, labels=lab

bins_precipitation = [0, 200, 400, 600, float('inf')]
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l

# Subset columns to reduce dimensionality
columns_to_keep = ['Rice_Yield_Category', 'Precipitation_Category', 'Min Temp (Centigrate)', 'Max
df_subset = df[columns_to_keep]

# Convert to transactional format
transactions = df_subset.apply(lambda row: [f"{col}={row[col]}" for col in df_subset.columns if p

# Encode data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.2, use_colnames=True)

# Count itemsets of each size
```

```
num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().sort_index().to_dict()

# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.3, num_itemsets

# Filter and analyze rules
filtered_rules = rules[(rules['lift'] > 1.2)]
print(filtered_rules)

# Optionally save to CSV
# filtered_rules.to_csv('association_rules.csv', index=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Empty DataFrame
Columns: [antecedents, consequents, antecedent support, consequent support, support, confidence, lift, representativity, leverage, con
Index: []
```

skip the cell

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/ICRISAT_Merge_Crop_Climate_Cost.csv')

# Preprocessing: Binning for selected variables
bins_rice_yield = [0, 500, 1000, 1500, float('inf')]
labels_rice_yield = ['Low', 'Moderate', 'High', 'Very High']
df['Rice_Yield_Category'] = pd.cut(df['RICE YIELD (Kg per ha)'], bins=bins_rice_yield, labels=lab

bins_precipitation = [0, 200, 400, 600, float('inf')]
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l

# Subset columns to reduce dimensionality
columns_to_keep = ['Rice_Yield_Category', 'Precipitation_Category', 'Min Temp (Centigrate)', 'Max
df_subset = df[columns_to_keep]

# Convert to transactional format
transactions = df_subset.apply(lambda row: [f"{col}={row[col]}" for col in df_subset.columns if p

# Encode data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.05, use_colnames=True)  # Reduced min_supp

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

# Generate Association Rules
```

```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.3)  # Lowered c

# Filter and analyze rules (lower lift threshold)
filtered_rules = rules[(rules['lift'] > 1.0)]  # Reduced lift threshold

# Display filtered rules
print("\nFiltered Association Rules:")
print(filtered_rules)

# Optionally save to CSV
# filtered_rules.to_csv('association_rules.csv', index=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Frequent Itemsets:
    support                                         itemsets
0  0.167052                      (Rice_Yield_Category=Low)
1  0.799846             (Precipitation_Category=Very High)
2  0.307929                 (Rice_Yield_Category=Moderate)
3  0.070824                  (Precipitation_Category=High)
4  0.226328                     (Rice_Yield_Category=High)
5  0.255581                (Rice_Yield_Category=Very High)
6  0.115473  (Precipitation_Category=Very High, Rice_Yield_...
7  0.248653  (Precipitation_Category=Very High, Rice_Yield_...
8  0.200924  (Precipitation_Category=Very High, Rice_Yield_...
9  0.201694  (Precipitation_Category=Very High, Rice_Yield_...
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-b5555e8eb477> in <cell line: 37>()
     35
     36 # Generate Association Rules
---> 37 rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.3)  # Lowered confidence threshold
     38
     39 # Filter and analyze rules (lower lift threshold)

TypeError: association_rules() missing 1 required positional argument: 'num_itemsets'
```

skip the cell

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/ICRISAT_Merge_Crop_Climate_Cost (1).csv')

# Preprocessing: Binning for selected variables
bins_rice_yield = [0, 500, 1000, 1500, float('inf')]
labels_rice_yield = ['Low', 'Moderate', 'High', 'Very High']
df['Rice_Yield_Category'] = pd.cut(df['RICE YIELD (Kg per ha)'], bins=bins_rice_yield, labels=lab

bins_precipitation = [0, 200, 400, 600, float('inf')]
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l

# Subset columns to reduce dimensionality
columns_to_keep = ['Rice_Yield_Category', 'Precipitation_Category', 'Min Temp (Centigrate)', 'Max
df_subset = df[columns_to_keep]

# Convert to transactional format
transactions = df_subset.apply(lambda row: [f"{col}={row[col]}" for col in df_subset.columns if p
```

```
# Encode data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.2, use_colnames=True)  # Reduced min_suppo

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

# Generate Association Rules with correct argument for frequent itemsets
# Use the length of the itemsets to set num_itemsets
num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.3, num_itemsets

# Filter and analyze rules (lower lift threshold)
filtered_rules = rules[(rules['lift'] > 1.0)]  # Reduced lift threshold

# Display filtered rules
print("\nFiltered Association Rules:")
print(filtered_rules)

# Optionally save to CSV
# filtered_rules.to_csv('association_rules.csv', index=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-8-0dc45c2ca478> in <cell line: 6>()
      4
      5 # Load data
----> 6 df = pd.read_csv('/content/ICRISAT_Merge_Crop_Climate_Cost (1).csv')
      7
      8 # Preprocessing: Binning for selected variables

                                 4 frames
/usr/local/lib/python3.10/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mode, encoding, compression, memory_map,
is_text, errors, storage_options)
    871         if ioargs.encoding and "b" not in ioargs.mode:
    872             # Encoding
--> 873             handle = open(
    874                 handle,
    875                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: '/content/ICRISAT_Merge_Crop_Climate_Cost (1).csv'
```

/content/Crop_Data_Final.csv

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'WheatYield'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
 2 frames 
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'WheatYield'
```

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/Crop_Data_Final.csv')

# Feature Engineering: Yield Efficiency and Ratios for Rice
df['Rice_Yield_Per_Area'] = df['RICE YIELD (Kg per ha)'] / df['RICE AREA (1000 ha)']
df['Temperature_Range'] = df['Max Temp (Centigrate)'] - df['Min Temp (Centigrate)']
df['Nitrogen_Use_Efficiency'] = df['RICE YIELD (Kg per ha)'] / df['NITROGEN PER HA OF NCA (Kg per

# Preprocessing: Binning for Rice Yield Categories
bins_yield = [0, 500, 1000, 1500, float('inf')]
labels_yield = ['Low', 'Moderate', 'High', 'Very High']
df['Rice_Yield_Category'] = pd.cut(df['RICE YIELD (Kg per ha)'], bins=bins_yield, labels=labels_y

# Binning for Precipitation
bins_precipitation = [0, 200, 400, 600, float('inf')]
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l

# Subset columns to reduce dimensionality
columns_to_keep = [
    'Rice_Yield_Category', 'Precipitation_Category', 'Min Temp (Centigrate)',
    'Max Temp (Centigrate)', 'Temperature_Range', 'Rice_Yield_Per_Area',
    'Nitrogen_Use_Efficiency'
]
df_subset = df[columns_to_keep]

# Convert to transactional format
transactions = df_subset.apply(
    lambda row: [f"{col}={row[col]}" for col in df_subset.columns if pd.notnull(row[col])], axis=
```

```
).tolist()

# Encode data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.3, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

# Use the length of the itemsets to set num_itemsets
num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()
# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.4, num_itemsets

# Filter and analyze rules
filtered_rules = rules[rules['lift'] > 1.0]

# Display filtered rules
print("\nFiltered Association Rules:")
print(filtered_rules)

# Save filtered rules to CSV
#filtered_rules.to_csv('rice_association_rules.csv', index=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Frequent Itemsets:
     support                                        itemsets
0   0.398693                     (Max Temp (Centigrate)=33)
1   0.872082                (Precipitation_Category=Very High)
2   0.362278                          (Temperature_Range=13)
3   0.346405                    (Rice_Yield_Category=Moderate)
4   0.359477                          (Temperature_Range=12)
5   0.303455                       (Min Temp (Centigrate)=20)
6   0.342670                       (Min Temp (Centigrate)=21)
7   0.363212  (Max Temp (Centigrate)=33, Precipitation_Categ...
8   0.309991  (Temperature_Range=13, Precipitation_Category=...
9   0.339869  (Temperature_Range=12, Precipitation_Category=...
10  0.322129  (Min Temp (Centigrate)=21, Precipitation_Categ...

Filtered Association Rules:
                         antecedents                       consequents  \
0         (Max Temp (Centigrate)=33)  (Precipitation_Category=Very High)
1  (Precipitation_Category=Very High)         (Max Temp (Centigrate)=33)
3             (Temperature_Range=12)  (Precipitation_Category=Very High)
4         (Min Temp (Centigrate)=21)  (Precipitation_Category=Very High)

   antecedent support  consequent support   support  confidence      lift  \
0            0.398693            0.872082  0.363212    0.911007  1.044634
1            0.872082            0.398693  0.363212    0.416488  1.044634
3            0.359477            0.872082  0.339869    0.945455  1.084135
4            0.342670            0.872082  0.322129    0.940054  1.077943

   representativity  leverage  conviction  zhangs_metric   jaccard  certainty  \
0               1.0  0.015519    1.437393       0.071057  0.400206   0.304296
1               1.0  0.015519    1.030497       0.334021  0.400206   0.029595
3               1.0  0.026376    2.345160       0.121159  0.381152   0.573590
4               1.0  0.023292    2.133902       0.110001  0.360879   0.531375

   kulczynski
0    0.663748
1    0.663748
3    0.667588
```

```
    4    0.654717
```

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/Crop_Data_Final.csv')

# Feature Engineering: Yield Efficiency and Ratios for Wheat
df['Wheat_Yield_Per_Area'] = df['WHEAT YIELD (Kg per ha)'] / df['WHEAT AREA (1000 ha)']
df['Temperature_Range'] = df['Max Temp (Centigrate)'] - df['Min Temp (Centigrate)']
df['Nitrogen_Use_Efficiency'] = df['WHEAT YIELD (Kg per ha)'] / df['NITROGEN PER HA OF NCA (Kg pe

# Preprocessing: Binning for Wheat Yield Categories
bins_yield = [0, 500, 1000, 1500, float('inf')]
labels_yield = ['Low', 'Moderate', 'High', 'Very High']
df['Wheat_Yield_Category'] = pd.cut(df['WHEAT YIELD (Kg per ha)'], bins=bins_yield, labels=labels

# Binning for Precipitation
bins_precipitation = [0, 200, 400, 600, float('inf')]
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l

# Subset columns to reduce dimensionality
columns_to_keep = [
    'Wheat_Yield_Category', 'Precipitation_Category', 'Min Temp (Centigrate)',
    'Max Temp (Centigrate)', 'Temperature_Range', 'Wheat_Yield_Per_Area',
    'Nitrogen_Use_Efficiency'
]
df_subset = df[columns_to_keep]

# Convert to transactional format
transactions = df_subset.apply(
    lambda row: [f"{col}={row[col]}" for col in df_subset.columns if pd.notnull(row[col])], axis=
).tolist()

# Encode data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.3, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

# Use the length of the itemsets to set num_itemsets
num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()
# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.4, num_itemsets

# Filter and analyze rules
filtered_rules = rules[rules['lift'] > 1.0]
```

```
# Display filtered rules
print("\nFiltered Association Rules:")
print(filtered_rules)

# Save filtered rules to CSV
#filtered_rules.to_csv('wheat_association_rules.csv', index=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Frequent Itemsets:
     support                                        itemsets
0   0.398693                     (Max Temp (Centigrate)=33)
1   0.872082            (Precipitation_Category=Very High)
2   0.362278                        (Temperature_Range=13)
3   0.358543               (Wheat_Yield_Category=Moderate)
4   0.325864                   (Wheat_Yield_Category=High)
5   0.359477                        (Temperature_Range=12)
6   0.303455                     (Min Temp (Centigrate)=20)
7   0.342670                     (Min Temp (Centigrate)=21)
8   0.363212  (Max Temp (Centigrate)=33, Precipitation_Categ...
9   0.309991  (Temperature_Range=13, Precipitation_Category=...
10  0.307190  (Wheat_Yield_Category=Moderate, Precipitation_...
11  0.339869  (Temperature_Range=12, Precipitation_Category=...
12  0.322129  (Min Temp (Centigrate)=21, Precipitation_Categ...

Filtered Association Rules:
                        antecedents                       consequents  \
0         (Max Temp (Centigrate)=33)  (Precipitation_Category=Very High)
1  (Precipitation_Category=Very High)         (Max Temp (Centigrate)=33)
4             (Temperature_Range=12)  (Precipitation_Category=Very High)
5         (Min Temp (Centigrate)=21)  (Precipitation_Category=Very High)

   antecedent support  consequent support   support  confidence      lift  \
0            0.398693            0.872082  0.363212    0.911007  1.044634
1            0.872082            0.398693  0.363212    0.416488  1.044634
4            0.359477            0.872082  0.339869    0.945455  1.084135
5            0.342670            0.872082  0.322129    0.940054  1.077943

   representativity  leverage  conviction  zhangs_metric   jaccard  certainty  \
0               1.0  0.015519    1.437393       0.071057  0.400206   0.304296
1               1.0  0.015519    1.030497       0.334021  0.400206   0.029595
4               1.0  0.026376    2.345160       0.121159  0.381152   0.573590
5               1.0  0.023292    2.133902       0.110001  0.360879   0.531375

   kulczynski
0    0.663748
1    0.663748
4    0.667588
5    0.654717
```

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/Crop_Data_Final.csv')

# Feature Engineering: Yield Efficiency and Ratios for Groundnut
df['Groundnut_Yield_Per_Area'] = df['GROUNDNUT YIELD (Kg per ha)'] / df['GROUNDNUT AREA (1000 ha)
df['Temperature_Range'] = df['Max Temp (Centigrate)'] - df['Min Temp (Centigrate)']
df['Nitrogen_Use_Efficiency'] = df['GROUNDNUT YIELD (Kg per ha)'] / df['NITROGEN PER HA OF NCA (K

# Preprocessing: Binning for Groundnut Yield Categories
bins_yield = [0, 500, 1000, 1500, float('inf')]
labels_yield = ['Low', 'Moderate', 'High', 'Very High']
df['Groundnut_Yield_Category'] = pd.cut(df['GROUNDNUT YIELD (Kg per ha)'], bins=bins_yield, label

# Binning for Precipitation
bins_precipitation = [0, 200, 400, 600, float('inf')]
```

```
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l

# Subset columns to reduce dimensionality
columns_to_keep = [
    'Groundnut_Yield_Category', 'Precipitation_Category', 'Min Temp (Centigrate)',
    'Max Temp (Centigrate)', 'Temperature_Range', 'Groundnut_Yield_Per_Area',
    'Nitrogen_Use_Efficiency'
]
df_subset = df[columns_to_keep]

# Convert to transactional format
transactions = df_subset.apply(
    lambda row: [f"{col}={row[col]}" for col in df_subset.columns if pd.notnull(row[col])], axis=
).tolist()

# Encode data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.3, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()
# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.4, num_itemsets


# Filter and analyze rules
filtered_rules = rules[rules['lift'] > 1.0]

# Display filtered rules
print("\nFiltered Association Rules:")
print(filtered_rules)

# Save filtered rules to CSV
#filtered_rules.to_csv('groundnut_association_rules.csv', index=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Frequent Itemsets:
     support                                          itemsets
0   0.398693                         (Max Temp (Centigrate)=33)
1   0.872082                 (Precipitation_Category=Very High)
2   0.362278                             (Temperature_Range=13)
3   0.505135                (Groundnut_Yield_Category=Moderate)
4   0.359477                             (Temperature_Range=12)
5   0.303455                         (Min Temp (Centigrate)=20)
6   0.342670                         (Min Temp (Centigrate)=21)
7   0.363212  (Max Temp (Centigrate)=33, Precipitation_Categ...
8   0.309991  (Temperature_Range=13, Precipitation_Category=...
9   0.441643  (Groundnut_Yield_Category=Moderate, Precipitat...
10  0.339869  (Temperature_Range=12, Precipitation_Category=...
11  0.322129  (Min Temp (Centigrate)=21, Precipitation_Categ...

Filtered Association Rules:
                          antecedents                         consequents  \
0          (Max Temp (Centigrate)=33)   (Precipitation_Category=Very High)
```

```
1   (Precipitation_Category=Very High)          (Max Temp (Centigrate)=33)
3  (Groundnut_Yield_Category=Moderate)   (Precipitation_Category=Very High)
4   (Precipitation_Category=Very High)  (Groundnut_Yield_Category=Moderate)
5              (Temperature_Range=12)   (Precipitation_Category=Very High)
6          (Min Temp (Centigrate)=21)   (Precipitation_Category=Very High)

   antecedent support  consequent support   support  confidence      lift  \
0            0.398693            0.872082  0.363212    0.911007  1.044634
1            0.872082            0.398693  0.363212    0.416488  1.044634
3            0.505135            0.872082  0.441643    0.874307  1.002551
4            0.872082            0.505135  0.441643    0.506424  1.002551
5            0.359477            0.872082  0.339869    0.945455  1.084135
6            0.342670            0.872082  0.322129    0.940054  1.077943

   representativity  leverage  conviction  zhangs_metric   jaccard  certainty  \
0               1.0  0.015519    1.437393       0.071057  0.400206   0.304296
1               1.0  0.015519    1.030497       0.334021  0.400206   0.029595
3               1.0  0.001124    1.017699       0.005142  0.472056   0.017391
4               1.0  0.001124    1.002611       0.019892  0.472056   0.002604
5               1.0  0.026376    2.345160       0.121159  0.381152   0.573590
6               1.0  0.023292    2.133902       0.110001  0.360879   0.531375

   kulczynski
0    0.663748
1    0.663748
3    0.690365
4    0.690365
5    0.667588
6    0.654717
```

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/Crop_Data_Final.csv')

# Feature Engineering: Yield Efficiency and Ratios for Maize
df['Maize_Yield_Per_Area'] = df['MAIZE YIELD (Kg per ha)'] / df['MAIZE AREA (1000 ha)']
df['Temperature_Range'] = df['Max Temp (Centigrate)'] - df['Min Temp (Centigrate)']
df['Nitrogen_Use_Efficiency'] = df['MAIZE YIELD (Kg per ha)'] / df['NITROGEN PER HA OF NCA (Kg pe

# Preprocessing: Binning for Maize Yield Categories
bins_yield = [0, 500, 1000, 1500, float('inf')]
labels_yield = ['Low', 'Moderate', 'High', 'Very High']
df['Maize_Yield_Category'] = pd.cut(df['MAIZE YIELD (Kg per ha)'], bins=bins_yield, labels=labels

# Binning for Precipitation
bins_precipitation = [0, 200, 400, 600, float('inf')]
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l

# Subset columns to reduce dimensionality
columns_to_keep = [
    'Maize_Yield_Category', 'Precipitation_Category', 'Min Temp (Centigrate)',
    'Max Temp (Centigrate)', 'Temperature_Range', 'Maize_Yield_Per_Area',
    'Nitrogen_Use_Efficiency'
]
df_subset = df[columns_to_keep]

# Convert to transactional format
transactions = df_subset.apply(
    lambda row: [f"{col}={row[col]}" for col in df_subset.columns if pd.notnull(row[col])], axis=
).tolist()
```

```
# Encode data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.3, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)


num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()
# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.4, num_itemsets

# Filter and analyze rules
filtered_rules = rules[rules['lift'] > 1.0]

# Display filtered rules
print("\nFiltered Association Rules:")
print(filtered_rules)

# Save filtered rules to CSV
#filtered_rules.to_csv('maize_association_rules.csv', index=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Frequent Itemsets:
     support                                        itemsets
0   0.398693                        (Max Temp (Centigrate)=33)
1   0.872082                (Precipitation_Category=Very High)
2   0.426704                  (Maize_Yield_Category=Very High)
3   0.362278                            (Temperature_Range=13)
4   0.359477                            (Temperature_Range=12)
5   0.303455                        (Min Temp (Centigrate)=20)
6   0.342670                        (Min Temp (Centigrate)=21)
7   0.363212  (Max Temp (Centigrate)=33, Precipitation_Categ...
8   0.391223  (Maize_Yield_Category=Very High, Precipitation...
9   0.309991  (Temperature_Range=13, Precipitation_Category=...
10  0.339869  (Temperature_Range=12, Precipitation_Category=...
11  0.322129  (Min Temp (Centigrate)=21, Precipitation_Categ...

Filtered Association Rules:
                        antecedents                       consequents  \
0         (Max Temp (Centigrate)=33)  (Precipitation_Category=Very High)
1  (Precipitation_Category=Very High)         (Max Temp (Centigrate)=33)
2    (Maize_Yield_Category=Very High)  (Precipitation_Category=Very High)
3  (Precipitation_Category=Very High)    (Maize_Yield_Category=Very High)
5             (Temperature_Range=12)  (Precipitation_Category=Very High)
6         (Min Temp (Centigrate)=21)  (Precipitation_Category=Very High)

   antecedent support  consequent support   support  confidence      lift  \
0            0.398693            0.872082  0.363212    0.911007  1.044634
1            0.872082            0.398693  0.363212    0.416488  1.044634
2            0.426704            0.872082  0.391223    0.916849  1.051333
3            0.872082            0.426704  0.391223    0.448608  1.051333
5            0.359477            0.872082  0.339869    0.945455  1.084135
6            0.342670            0.872082  0.322129    0.940054  1.077943

   representativity  leverage  conviction  zhangs_metric   jaccard  certainty  \
0               1.0  0.015519    1.437393       0.071057  0.400206   0.304296
1               1.0  0.015519    1.030497       0.334021  0.400206   0.029595
2               1.0  0.019102    1.538380       0.085169  0.431070   0.349966
3               1.0  0.019102    1.039725       0.381705  0.431070   0.038207
5               1.0  0.026376    2.345160       0.121159  0.381152   0.573590
6               1.0  0.023292    2.133902       0.110001  0.360879   0.531375

   kulczynski
```

```
0    0.663748
1    0.663748
2    0.682729
3    0.682729
5    0.667588
6    0.654717
```

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/Crop_Data_Final.csv')

# Feature Engineering: Yield Efficiency and Ratios
df['ChickPea_Yield_Per_Area'] = df['CHICKPEA YIELD (Kg per ha)'] / df['CHICKPEA AREA (1000 ha)']
df['Temperature_Range'] = df['Max Temp (Centigrate)'] - df['Min Temp (Centigrate)']
df['Nitrogen_Use_Efficiency'] = df['CHICKPEA YIELD (Kg per ha)'] / df['NITROGEN PER HA OF NCA (Kg

# Preprocessing: Binning for Yield Categories
bins_yield = [0, 500, 1000, 1500, float('inf')]
labels_yield = ['Low', 'Moderate', 'High', 'Very High']
df['ChickPea_Yield_Category'] = pd.cut(df['CHICKPEA YIELD (Kg per ha)'], bins=bins_yield, labels=

# Binning for Precipitation
bins_precipitation = [0, 200, 400, 600, float('inf')]
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l

# Subset columns to reduce dimensionality
columns_to_keep = [
    'ChickPea_Yield_Category', 'Precipitation_Category', 'Min Temp (Centigrate)',
    'Max Temp (Centigrate)', 'Temperature_Range', 'ChickPea_Yield_Per_Area',
    'Nitrogen_Use_Efficiency'
]
df_subset = df[columns_to_keep]

# Convert to transactional format
transactions = df_subset.apply(
    lambda row: [f"{col}={row[col]}" for col in df_subset.columns if pd.notnull(row[col])], axis=
).tolist()

# Encode data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.3, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)
```

```
num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()
# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.4, num_itemsets
```

```
# Filter and analyze rules
filtered_rules = rules[rules['lift'] > 1.0]

# Display filtered rules
print("\nFiltered Association Rules:")
print(filtered_rules)

# Save filtered rules to CSV
#filtered_rules.to_csv('association_rules.csv', index=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Frequent Itemsets:
     support                                        itemsets
0   0.533147                     (ChickPea_Yield_Category=Low)
1   0.398693                        (Max Temp (Centigrate)=33)
2   0.872082                 (Precipitation_Category=Very High)
3   0.362278                             (Temperature_Range=13)
4   0.428571                 (ChickPea_Yield_Category=Moderate)
5   0.359477                             (Temperature_Range=12)
6   0.303455                        (Min Temp (Centigrate)=20)
7   0.342670                        (Min Temp (Centigrate)=21)
8   0.433240  (ChickPea_Yield_Category=Low, Precipitation_Ca...
9   0.363212  (Max Temp (Centigrate)=33, Precipitation_Categ...
10  0.309991  (Temperature_Range=13, Precipitation_Category=...
11  0.401494  (ChickPea_Yield_Category=Moderate, Precipitati...
12  0.339869  (Temperature_Range=12, Precipitation_Category=...
13  0.322129  (Min Temp (Centigrate)=21, Precipitation_Categ...

Filtered Association Rules:
                        antecedents                         consequents  \
2         (Max Temp (Centigrate)=33)  (Precipitation_Category=Very High)
3  (Precipitation_Category=Very High)         (Max Temp (Centigrate)=33)
5  (ChickPea_Yield_Category=Moderate)  (Precipitation_Category=Very High)
6  (Precipitation_Category=Very High)  (ChickPea_Yield_Category=Moderate)
7             (Temperature_Range=12)  (Precipitation_Category=Very High)
8         (Min Temp (Centigrate)=21)  (Precipitation_Category=Very High)

   antecedent support  consequent support   support  confidence      lift  \
2            0.398693            0.872082  0.363212    0.911007  1.044634
3            0.872082            0.398693  0.363212    0.416488  1.044634
5            0.428571            0.872082  0.401494    0.936819  1.074233
6            0.872082            0.428571  0.401494    0.460385  1.074233
7            0.359477            0.872082  0.339869    0.945455  1.084135
8            0.342670            0.872082  0.322129    0.940054  1.077943

   representativity  leverage  conviction  zhangs_metric   jaccard  certainty  \
2               1.0  0.015519    1.437393       0.071057  0.400206   0.304296
3               1.0  0.015519    1.030497       0.334021  0.400206   0.029595
5               1.0  0.027744    2.024631       0.120930  0.446521   0.506083
6               1.0  0.027744    1.058957       0.540214  0.446521   0.055675
7               1.0  0.026376    2.345160       0.121159  0.381152   0.573590
8               1.0  0.023292    2.133902       0.110001  0.360879   0.531375

   kulczynski
2    0.663748
3    0.663748
5    0.698602
6    0.698602
7    0.667588
8    0.654717
```

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/Crop_Data_Final.csv')

# Feature Engineering: Yield Efficiency and Ratios for Sugarcane
```

```
df['Sugarcane_Yield_Per_Area'] = df['SUGARCANE YIELD (Kg per ha)'] / df['SUGARCANE AREA (1000 ha)
df['Temperature_Range'] = df['Max Temp (Centigrate)'] - df['Min Temp (Centigrate)']
df['Nitrogen_Use_Efficiency'] = df['SUGARCANE YIELD (Kg per ha)'] / df['NITROGEN PER HA OF NCA (K

# Preprocessing: Binning for Sugarcane Yield Categories
bins_yield = [0, 500, 1000, 1500, float('inf')]
labels_yield = ['Low', 'Moderate', 'High', 'Very High']
df['Sugarcane_Yield_Category'] = pd.cut(df['SUGARCANE YIELD (Kg per ha)'], bins=bins_yield, label

# Binning for Precipitation
bins_precipitation = [0, 200, 400, 600, float('inf')]
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l

# Subset columns to reduce dimensionality
columns_to_keep = [
    'Sugarcane_Yield_Category', 'Precipitation_Category', 'Min Temp (Centigrate)',
    'Max Temp (Centigrate)', 'Temperature_Range', 'Sugarcane_Yield_Per_Area',
    'Nitrogen_Use_Efficiency'
]
df_subset = df[columns_to_keep]

# Convert to transactional format
transactions = df_subset.apply(
    lambda row: [f"{col}={row[col]}" for col in df_subset.columns if pd.notnull(row[col])], axis=
).tolist()

# Encode data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.3, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()
# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.4, num_itemsets

# Filter and analyze rules
filtered_rules = rules[rules['lift'] > 1.0]

# Display filtered rules
print("\nFiltered Association Rules:")
print(filtered_rules)

# Save filtered rules to CSV
#filtered_rules.to_csv('sugarcane_association_rules.csv', index=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_
  and should_run_async(code)
Frequent Itemsets:
     support                                itemsets
0   0.968254     (Sugarcane_Yield_Category=Very High)
1   0.398693               (Max Temp (Centigrate)=33)
```

```
2   0.872082                (Precipitation_Category=Very High)
3   0.362278                           (Temperature_Range=13)
4   0.359477                           (Temperature_Range=12)
5   0.303455                      (Min Temp (Centigrate)=20)
6   0.342670                      (Min Temp (Centigrate)=21)
7   0.380952  (Max Temp (Centigrate)=33, Sugarcane_Yield_Cat...
8   0.363212  (Max Temp (Centigrate)=33, Precipitation_Categ...
9   0.345472  (Sugarcane_Yield_Category=Very High, Max Temp ...
10  0.840336  (Sugarcane_Yield_Category=Very High, Precipita...
11  0.347339  (Temperature_Range=13, Sugarcane_Yield_Categor...
12  0.309991  (Temperature_Range=13, Precipitation_Category=...
13  0.342670  (Temperature_Range=12, Sugarcane_Yield_Categor...
14  0.339869  (Temperature_Range=12, Precipitation_Category=...
15  0.323063  (Sugarcane_Yield_Category=Very High, Temperatu...
16  0.300654  (Sugarcane_Yield_Category=Very High, Min Temp ...
17  0.322129  (Min Temp (Centigrate)=21, Precipitation_Categ...
18  0.316527  (Min Temp (Centigrate)=21, Sugarcane_Yield_Cat...

Filtered Association Rules:
                                      antecedents  \
1                        (Max Temp (Centigrate)=33)
2                (Precipitation_Category=Very High)
3   (Max Temp (Centigrate)=33, Sugarcane_Yield_Cat...
4   (Precipitation_Category=Very High, Sugarcane_Y...
6                        (Max Temp (Centigrate)=33)
12                           (Temperature_Range=12)
13  (Temperature_Range=12, Sugarcane_Yield_Categor...
15                           (Temperature_Range=12)
16                       (Min Temp (Centigrate)=20)
17                       (Min Temp (Centigrate)=21)

                                      consequents  antecedent support  \
1                (Precipitation_Category=Very High)            0.398693
2                        (Max Temp (Centigrate)=33)            0.872082
3                (Precipitation_Category=Very High)            0.380952
4                        (Max Temp (Centigrate)=33)            0.840336
6   (Precipitation_Category=Very High, Sugarcane_Y...            0.398693
12               (Precipitation_Category=Very High)            0.359477
13               (Precipitation_Category=Very High)            0.342670
15  (Precipitation_Category=Very High, Sugarcane_Y...            0.359477
16              (Sugarcane_Yield_Category=Very High)            0.303455
17               (Precipitation_Category=Very High)            0.342670

    consequent support   support  confidence      lift  representativity  \
1             0.872082  0.363212    0.911007  1.044634               1.0
2             0.398693  0.363212    0.416488  1.044634               1.0
3             0.872082  0.345472    0.906863  1.039882               1.0
4             0.398693  0.345472    0.411111  1.031148               1.0
6             0.840336  0.345472    0.866511  1.031148               1.0
12            0.872082  0.339869    0.945455  1.084135               1.0
```

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/Crop_Data_Final.csv')

# Feature Engineering: Yield Efficiency and Ratios for Oilseed
df['Oilseed_Yield_Per_Area'] = df['OILSEEDS YIELD (Kg per ha)'] / df['OILSEEDS AREA (1000 ha)']
df['Temperature_Range'] = df['Max Temp (Centigrate)'] - df['Min Temp (Centigrate)']
df['Nitrogen_Use_Efficiency'] = df['OILSEEDS YIELD (Kg per ha)'] / df['NITROGEN PER HA OF NCA (Kg

# Preprocessing: Binning for Oilseed Yield Categories
bins_yield = [0, 500, 1000, 1500, float('inf')]
labels_yield = ['Low', 'Moderate', 'High', 'Very High']
df['Oilseed_Yield_Category'] = pd.cut(df['OILSEEDS YIELD (Kg per ha)'], bins=bins_yield, labels=l

# Binning for Precipitation
bins_precipitation = [0, 200, 400, 600, float('inf')]
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l
```

```
# Subset columns to reduce dimensionality
columns_to_keep = [
    'Oilseed_Yield_Category', 'Precipitation_Category', 'Min Temp (Centigrate)',
    'Max Temp (Centigrate)', 'Temperature_Range', 'Oilseed_Yield_Per_Area',
    'Nitrogen_Use_Efficiency'
]
df_subset = df[columns_to_keep]

# Convert to transactional format
transactions = df_subset.apply(
    lambda row: [f"{col}={row[col]}" for col in df_subset.columns if pd.notnull(row[col])], axis=
).tolist()

# Encode data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.3, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()
# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.4, num_itemsets

# Filter and analyze rules
filtered_rules = rules[rules['lift'] > 1.0]

# Display filtered rules
print("\nFiltered Association Rules:")
print(filtered_rules)

# Save filtered rules to CSV
filtered_rules.to_csv('oilseed_association_rules.csv', index=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Frequent Itemsets:
     support                                        itemsets
0   0.398693                     (Max Temp (Centigrate)=33)
1   0.388422                    (Oilseed_Yield_Category=Low)
2   0.872082                (Precipitation_Category=Very High)
3   0.362278                          (Temperature_Range=13)
4   0.380952                 (Oilseed_Yield_Category=Moderate)
5   0.359477                          (Temperature_Range=12)
6   0.303455                     (Min Temp (Centigrate)=20)
7   0.342670                     (Min Temp (Centigrate)=21)
8   0.363212  (Max Temp (Centigrate)=33, Precipitation_Categ...
9   0.310924  (Precipitation_Category=Very High, Oilseed_Yie...
10  0.309991  (Temperature_Range=13, Precipitation_Category=...
11  0.338936  (Oilseed_Yield_Category=Moderate, Precipitatio...
12  0.339869  (Temperature_Range=12, Precipitation_Category=...
13  0.322129  (Min Temp (Centigrate)=21, Precipitation_Categ...

Filtered Association Rules:
                        antecedents                       consequents  \
0         (Max Temp (Centigrate)=33)  (Precipitation_Category=Very High)
1  (Precipitation_Category=Very High)         (Max Temp (Centigrate)=33)
4   (Oilseed_Yield_Category=Moderate)  (Precipitation_Category=Very High)
5             (Temperature_Range=12)  (Precipitation_Category=Very High)
```

```
6         (Min Temp (Centigrate)=21)  (Precipitation_Category=Very High)

   antecedent support  consequent support   support  confidence      lift  \
0            0.398693            0.872082  0.363212    0.911007  1.044634
1            0.872082            0.398693  0.363212    0.416488  1.044634
4            0.380952            0.872082  0.338936    0.889706  1.020209
5            0.359477            0.872082  0.339869    0.945455  1.084135
6            0.342670            0.872082  0.322129    0.940054  1.077943

   representativity  leverage  conviction  zhangs_metric   jaccard  certainty  \
0               1.0  0.015519    1.437393       0.071057  0.400206   0.304296
1               1.0  0.015519    1.030497       0.334021  0.400206   0.029595
4               1.0  0.006714    1.159788       0.031998  0.370787   0.137774
5               1.0  0.026376    2.345160       0.121159  0.381152   0.573590
6               1.0  0.023292    2.133902       0.110001  0.360879   0.531375

   kulczynski
0    0.663748
1    0.663748
4    0.639178
5    0.667588
6    0.654717
```

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load data
df = pd.read_csv('/content/Crop_Data_Final.csv')

# Feature Engineering: Yield Efficiency and Ratios for Cotton
df['Cotton_Yield_Per_Area'] = df['COTTON YIELD (Kg per ha)'] / df['COTTON AREA (1000 ha)']
df['Temperature_Range'] = df['Max Temp (Centigrate)'] - df['Min Temp (Centigrate)']
df['Nitrogen_Use_Efficiency'] = df['COTTON YIELD (Kg per ha)'] / df['NITROGEN PER HA OF NCA (Kg p

# Preprocessing: Binning for Cotton Yield Categories
bins_yield = [0, 500, 1000, 1500, float('inf')]
labels_yield = ['Low', 'Moderate', 'High', 'Very High']
df['Cotton_Yield_Category'] = pd.cut(df['COTTON YIELD (Kg per ha)'], bins=bins_yield, labels=labe

# Binning for Precipitation
bins_precipitation = [0, 200, 400, 600, float('inf')]
labels_precipitation = ['Low', 'Moderate', 'High', 'Very High']
df['Precipitation_Category'] = pd.cut(df['Precipitation (mm)'], bins=bins_precipitation, labels=l

# Subset columns to reduce dimensionality
columns_to_keep = [
    'Cotton_Yield_Category', 'Precipitation_Category', 'Min Temp (Centigrate)',
    'Max Temp (Centigrate)', 'Temperature_Range', 'Cotton_Yield_Per_Area',
    'Nitrogen_Use_Efficiency'
]
df_subset = df[columns_to_keep]

# Convert to transactional format
transactions = df_subset.apply(
    lambda row: [f"{col}={row[col]}" for col in df_subset.columns if pd.notnull(row[col])], axis=
).tolist()

# Encode data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)
```

```
# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.3, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)


num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()
# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.4, num_itemsets

# Filter and analyze rules
filtered_rules = rules[rules['lift'] > 1.0]

# Display filtered rules
print("\nFiltered Association Rules:")
print(filtered_rules)

# Save filtered rules to CSV
#filtered_rules.to_csv('cotton_association_rules.csv', index=False)
```

```
Frequent Itemsets:
     support                                        itemsets
0   0.912232                     (Cotton_Yield_Category=Low)
1   0.398693                      (Max Temp (Centigrate)=33)
2   0.872082                (Precipitation_Category=Very High)
3   0.362278                           (Temperature_Range=13)
4   0.359477                           (Temperature_Range=12)
5   0.303455                       (Min Temp (Centigrate)=20)
6   0.342670                       (Min Temp (Centigrate)=21)
7   0.359477  (Cotton_Yield_Category=Low, Max Temp (Centigra...
8   0.363212  (Max Temp (Centigrate)=33, Precipitation_Categ...
9   0.323996  (Cotton_Yield_Category=Low, Max Temp (Centigra...
10  0.785247  (Cotton_Yield_Category=Low, Precipitation_Cate...
11  0.343604  (Cotton_Yield_Category=Low, Temperature_Range=13)
12  0.309991  (Temperature_Range=13, Precipitation_Category=...
13  0.339869  (Temperature_Range=12, Precipitation_Category=...
14  0.308123  (Cotton_Yield_Category=Low, Temperature_Range=12)
15  0.322129  (Min Temp (Centigrate)=21, Precipitation_Categ...
16  0.304388  (Cotton_Yield_Category=Low, Min Temp (Centigra...

Filtered Association Rules:
                                    antecedents  \
1                       (Max Temp (Centigrate)=33)
2                 (Precipitation_Category=Very High)
3   (Cotton_Yield_Category=Low, Max Temp (Centigra...
4   (Cotton_Yield_Category=Low, Precipitation_Cate...
6                       (Max Temp (Centigrate)=33)
9                          (Temperature_Range=13)
11                         (Temperature_Range=12)
13                      (Min Temp (Centigrate)=21)

                                    consequents  antecedent support  \
1                 (Precipitation_Category=Very High)            0.398693
2                       (Max Temp (Centigrate)=33)            0.872082
3                 (Precipitation_Category=Very High)            0.359477
4                       (Max Temp (Centigrate)=33)            0.785247
6   (Cotton_Yield_Category=Low, Precipitation_Cate...            0.398693
9                      (Cotton_Yield_Category=Low)            0.362278
11                (Precipitation_Category=Very High)            0.359477
13                (Precipitation_Category=Very High)            0.342670

    consequent support   support  confidence      lift  representativity  \
1             0.872082  0.363212    0.911007  1.044634               1.0
2             0.398693  0.363212    0.416488  1.044634               1.0
3             0.872082  0.323996    0.901299  1.033502               1.0
4             0.398693  0.323996    0.412604  1.034892               1.0
6             0.785247  0.323996    0.812646  1.034892               1.0
9             0.912232  0.343604    0.948454  1.039707               1.0
11            0.872082  0.339869    0.945455  1.084135               1.0
13            0.872082  0.322129    0.940054  1.077943               1.0
```

	leverage	conviction	zhangs_metric	jaccard	certainty	kulczynski
1	0.015519	1.437393	0.071057	0.400206	0.304296	0.663748
2	0.015519	1.030497	0.334021	0.400206	0.029595	0.663748
3	0.010503	1.296010	0.050609	0.356996	0.228401	0.636410
4	0.010924	1.023683	0.156998	0.376764	0.023135	0.612625
6	0.010924	1.146242	0.056071	0.376764	0.127584	0.612625

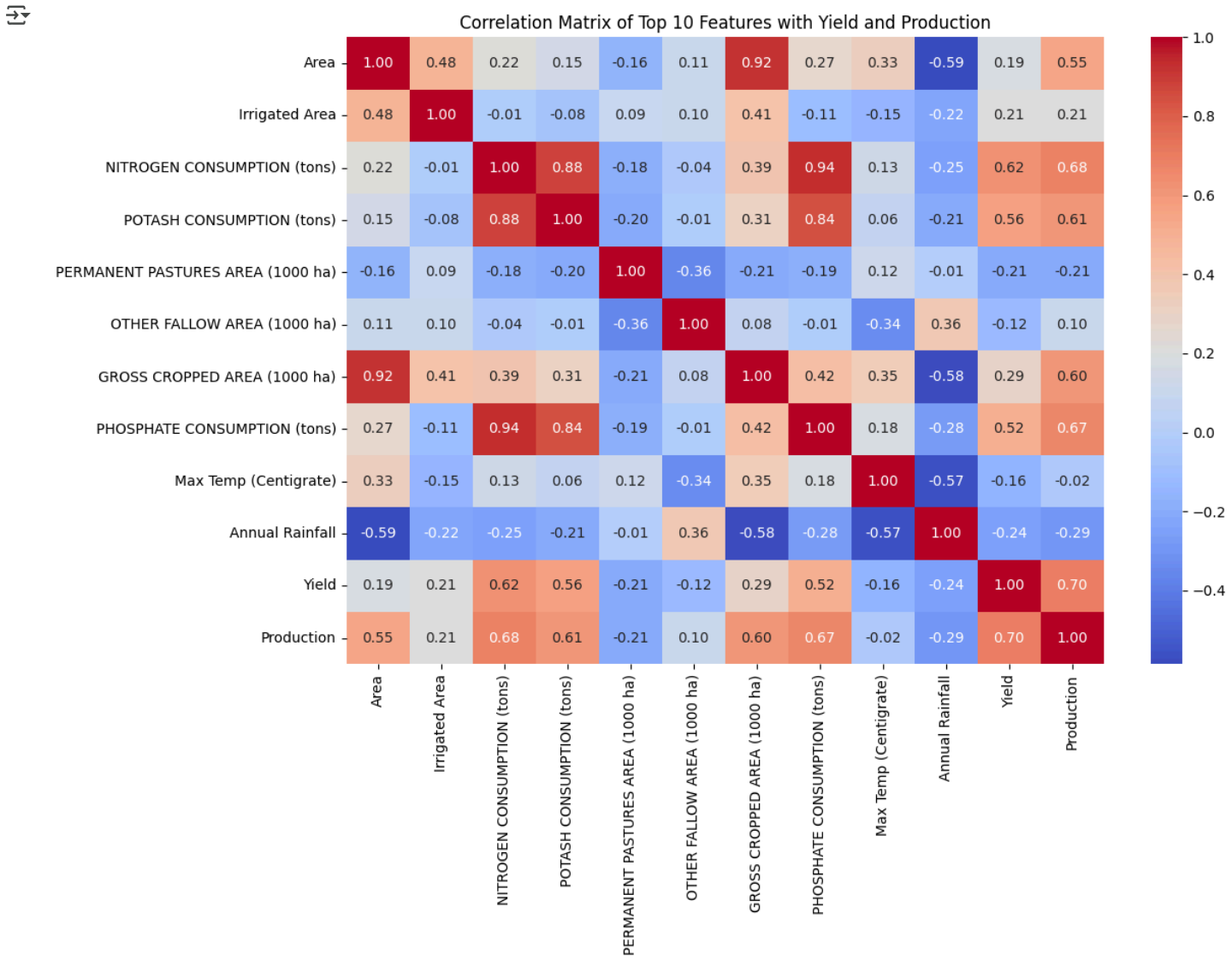
AGG

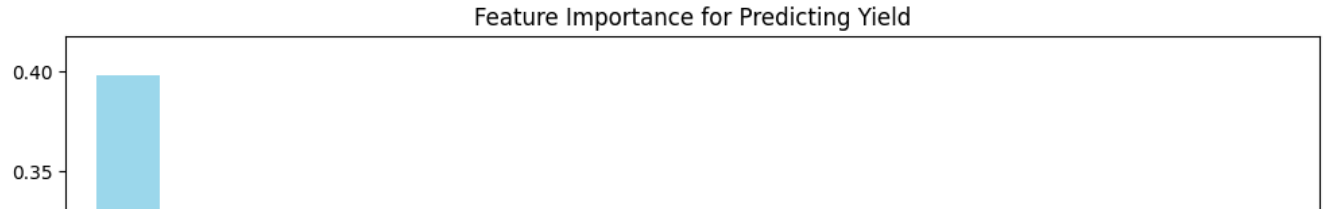
```
import gdown
file_id = '1KXMl4EVNofFnGsypt1UpZ4GJL6pGLJCe'
url = f'https://drive.google.com/uc?id={file_id}'
output = 'maharashtra_aggregate_1966_2017.csv'
gdown.download(url, output, quiet=False)
import pandas as pd
df = pd.read_csv('maharashtra_aggregate_1966_2017.csv')
```

```
Downloading...
From: https://drive.google.com/uc?id=1KXMl4EVNofFnGsypt1UpZ4GJL6pGLJCe
To: /content/maharashtra_aggregate_1966_2017.csv
100%|██████████| 277k/277k [00:00<00:00, 26.9MB/s]
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the data
df = pd.read_csv('/content/maharashtra_aggregate_1966_2017.csv')

# Columns of interest
columns_of_interest = [
    'Area', 'Irrigated Area', 'NITROGEN CONSUMPTION (tons)',
    'POTASH CONSUMPTION (tons)', 'PERMANENT PASTURES AREA (1000 ha)',
    'OTHER FALLOW AREA (1000 ha)', 'GROSS CROPPED AREA (1000 ha)',
    'PHOSPHATE CONSUMPTION (tons)', 'Max Temp (Centigrate)',
    'Annual Rainfall', 'Yield', 'Production'
]

# Subset the data
df_subset = df[columns_of_interest].dropna()

# Correlation matrix for visualizing relationships
correlation_matrix = df_subset.corr()

# Plot the correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix of Top 10 Features with Yield and Production")
plt.show()

# Prepare data for predictive modeling
X = df_subset.drop(['Yield', 'Production'], axis=1)
y_yield = df_subset['Yield']
y_production = df_subset['Production']

# Split the data
```

```
X_train_yield, X_test_yield, y_train_yield, y_test_yield = train_test_split(X, y_yield, test_size
X_train_prod, X_test_prod, y_train_prod, y_test_prod = train_test_split(X, y_production, test_siz

# Standardize the data
scaler = StandardScaler()
X_train_yield = scaler.fit_transform(X_train_yield)
X_test_yield = scaler.transform(X_test_yield)
X_train_prod = scaler.fit_transform(X_train_prod)
X_test_prod = scaler.transform(X_test_prod)

# Train RandomForestRegressor for Yield
model_yield = RandomForestRegressor(random_state=42)
model_yield.fit(X_train_yield, y_train_yield)
feature_importance_yield = pd.Series(model_yield.feature_importances_, index=X.columns)

# Train RandomForestRegressor for Production
model_prod = RandomForestRegressor(random_state=42)
model_prod.fit(X_train_prod, y_train_prod)
feature_importance_prod = pd.Series(model_prod.feature_importances_, index=X.columns)

# Plot Feature Importance
plt.figure(figsize=(12, 8))
feature_importance_yield.sort_values(ascending=False).plot(kind='bar', color='skyblue', alpha=0.8
plt.title("Feature Importance for Predicting Yield")
plt.ylabel("Importance Score")
plt.xlabel("Features")
plt.show()

plt.figure(figsize=(12, 8))
feature_importance_prod.sort_values(ascending=False).plot(kind='bar', color='coral', alpha=0.8)
plt.title("Feature Importance for Predicting Production")
plt.ylabel("Importance Score")
plt.xlabel("Features")
plt.show()
```



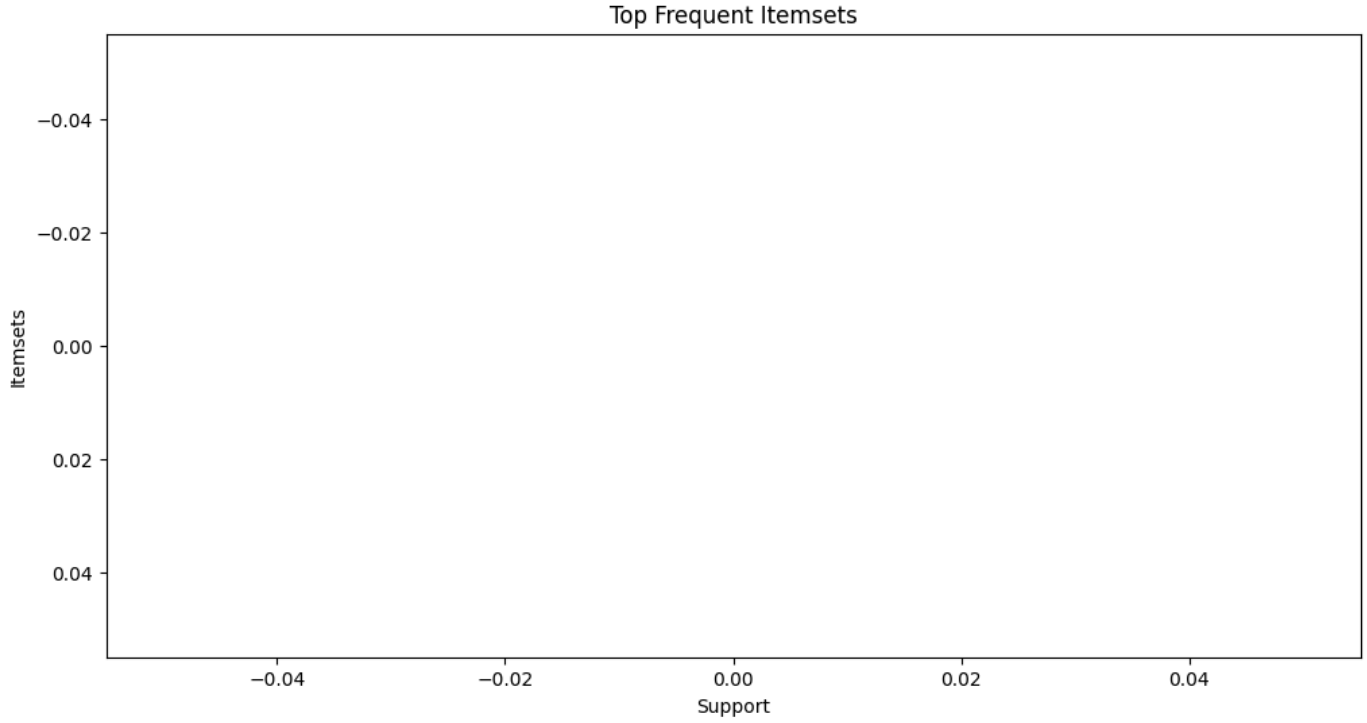



```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.preprocessing import TransactionEncoder
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('/content/maharashtra_aggregate_1966_2017.csv')

# Define the columns of interest
columns_of_interest = [
    'Area', 'Irrigated Area', 'NITROGEN CONSUMPTION (tons)',
    'POTASH CONSUMPTION (tons)', 'PERMANENT PASTURES AREA (1000 ha)',
    'OTHER FALLOW AREA (1000 ha)', 'GROSS CROPPED AREA (1000 ha)',
    'PHOSPHATE CONSUMPTION (tons)', 'Max Temp (Centigrate)',
    'Annual Rainfall', 'Yield', 'Production'
]
```

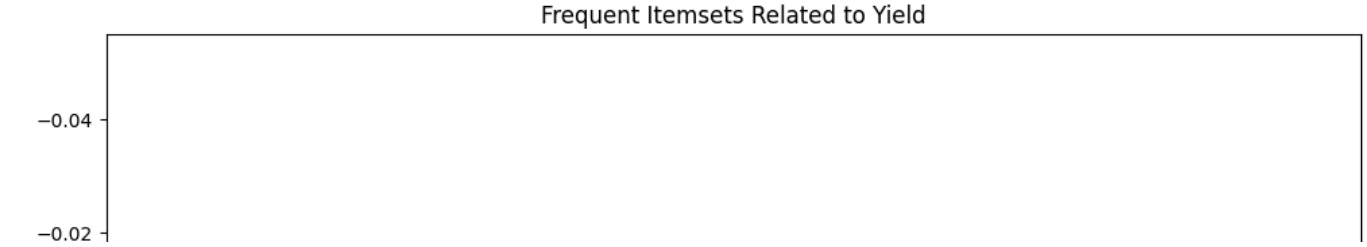
```
# Subset the data to keep only the columns of interest
df_subset = df[columns_of_interest].dropna()

# Convert to transactional format (binning and discretizing numerical variables)
transactions = df_subset.apply(
    lambda row: [f"{col}={row[col]}" for col in df_subset.columns if pd.notnull(row[col])], axis=
).tolist()

# Encode transactions
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth
frequent_itemsets = fpgrowth(df_encoded, min_support=0.1, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

# Visualize the top frequent itemsets
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(len)
top_itemsets = frequent_itemsets[frequent_itemsets['length'] > 1].nlargest(10, 'support')

# Plot top frequent itemsets
plt.figure(figsize=(12, 6))
plt.barh(
    top_itemsets['itemsets'].apply(lambda x: ', '.join(list(x))),
    top_itemsets['support'],
    color='skyblue'
)
plt.xlabel('Support')
plt.ylabel('Itemsets')
plt.title('Top Frequent Itemsets')
plt.gca().invert_yaxis()
plt.show()

# Generate relationships and insights for Yield and Production
yield_related = frequent_itemsets[
    frequent_itemsets['itemsets'].apply(lambda x: any('Yield=' in item for item in x))
]
production_related = frequent_itemsets[
    frequent_itemsets['itemsets'].apply(lambda x: any('Production=' in item for item in x))
]

# Display yield-related and production-related frequent itemsets
print("\nYield-Related Frequent Itemsets:")
print(yield_related)

print("\nProduction-Related Frequent Itemsets:")
print(production_related)

# Plot findings for Yield
plt.figure(figsize=(12, 6))
plt.barh(
    yield_related['itemsets'].apply(lambda x: ', '.join(list(x))),
    yield_related['support'],
    color='orange'
```

```
)
plt.xlabel('Support')
plt.ylabel('Itemsets')
plt.title('Frequent Itemsets Related to Yield')
plt.gca().invert_yaxis()
plt.show()

# Plot findings for Production
plt.figure(figsize=(12, 6))
plt.barh(
    production_related['itemsets'].apply(lambda x: ', '.join(list(x))),
    production_related['support'],
    color='green'
)
plt.xlabel('Support')
plt.ylabel('Itemsets')
plt.title('Frequent Itemsets Related to Production')
plt.gca().invert_yaxis()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Frequent Itemsets:
    support               itemsets
0  0.173913  (Irrigated Area=-20.0)
```




```
Yield-Related Frequent Itemsets:
Empty DataFrame
Columns: [support, itemsets, length]
Index: []

Production-Related Frequent Itemsets:
Empty DataFrame
Columns: [support, itemsets, length]
Index: []
```

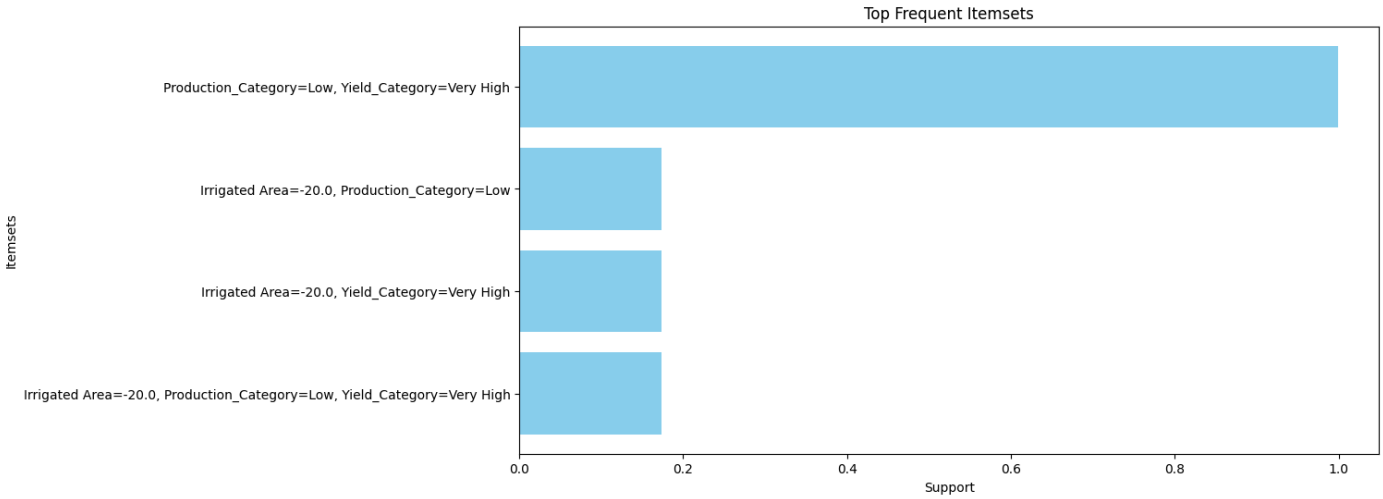



```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.preprocessing import TransactionEncoder
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('/content/maharashtra_aggregate_1966_2017.csv')

# Define the columns of interest
columns_of_interest = [
    'Area', 'Irrigated Area', 'NITROGEN CONSUMPTION (tons)',
    'POTASH CONSUMPTION (tons)', 'PERMANENT PASTURES AREA (1000 ha)',
    'OTHER FALLOW AREA (1000 ha)', 'GROSS CROPPED AREA (1000 ha)',
    'PHOSPHATE CONSUMPTION (tons)', 'Max Temp (Centigrate)',
    'Annual Rainfall', 'Yield', 'Production'
]
```

```
# Subset the data to keep only the columns of interest
df_subset = df[columns_of_interest].dropna()

# Discretize numerical features
df_subset['Yield_Category'] = pd.cut(
    df_subset['Yield'], bins=[0, 1000, 2000, 3000, float('inf')],
    labels=['Low', 'Medium', 'High', 'Very High']
)
df_subset['Production_Category'] = pd.cut(
    df_subset['Production'], bins=[0, 10000, 20000, 30000, float('inf')],
    labels=['Low', 'Medium', 'High', 'Very High']
)

# Drop the original Yield and Production columns after discretization
df_subset = df_subset.drop(['Yield', 'Production'], axis=1)

# Convert to transactional format
transactions = df_subset.apply(
    lambda row: [f"{col}={row[col]}" for col in df_subset.columns if pd.notnull(row[col])], axis=
).tolist()

# Encode transactions
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth with lower support threshold
frequent_itemsets = fpgrowth(df_encoded, min_support=0.05, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

# Generate relationships and insights for Yield and Production
yield_related = frequent_itemsets[
    frequent_itemsets['itemsets'].apply(lambda x: any('Yield_Category=' in item for item in x))
]
production_related = frequent_itemsets[
    frequent_itemsets['itemsets'].apply(lambda x: any('Production_Category=' in item for item in
]

# Display yield-related and production-related frequent itemsets
print("\nYield-Related Frequent Itemsets:")
print(yield_related)

print("\nProduction-Related Frequent Itemsets:")
print(production_related)

# Visualize top frequent itemsets
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(len)
top_itemsets = frequent_itemsets[frequent_itemsets['length'] > 1].nlargest(10, 'support')

plt.figure(figsize=(12, 6))
plt.barh(
    top_itemsets['itemsets'].apply(lambda x: ', '.join(list(x))),
    top_itemsets['support'],
    color='skyblue'
)
```

```
plt.xlabel('Support')
plt.ylabel('Itemsets')
plt.title('Top Frequent Itemsets')
plt.gca().invert_yaxis()
plt.show()

# Plot findings for Yield
plt.figure(figsize=(12, 6))
plt.barh(
    yield_related['itemsets'].apply(lambda x: ', '.join(list(x))),
    yield_related['support'],
    color='orange'
)
plt.xlabel('Support')
plt.ylabel('Itemsets')
plt.title('Frequent Itemsets Related to Yield')
plt.gca().invert_yaxis()
plt.show()

# Plot findings for Production
plt.figure(figsize=(12, 6))
plt.barh(
    production_related['itemsets'].apply(lambda x: ', '.join(list(x))),
    production_related['support'],
    color='green'
)
plt.xlabel('Support')
plt.ylabel('Itemsets')
plt.title('Frequent Itemsets Related to Production')
plt.gca().invert_yaxis()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Frequent Itemsets:
    support                                            itemsets
0  1.000000                          (Production_Category=Low)
1  0.999164                        (Yield_Category=Very High)
2  0.173913                           (Irrigated Area=-20.0)
3  0.999164  (Production_Category=Low, Yield_Category=Very ...
4  0.173913    (Irrigated Area=-20.0, Production_Category=Low)
5  0.173913   (Irrigated Area=-20.0, Yield_Category=Very High)
6  0.173913  (Irrigated Area=-20.0, Production_Category=Low...

Yield-Related Frequent Itemsets:
    support                                            itemsets
1  0.999164                        (Yield_Category=Very High)
3  0.999164  (Production_Category=Low, Yield_Category=Very ...
5  0.173913   (Irrigated Area=-20.0, Yield_Category=Very High)
6  0.173913  (Irrigated Area=-20.0, Production_Category=Low...

Production-Related Frequent Itemsets:
    support                                            itemsets
0  1.000000                          (Production_Category=Low)
3  0.999164  (Production_Category=Low, Yield_Category=Very ...
4  0.173913    (Irrigated Area=-20.0, Production_Category=Low)
6  0.173913  (Irrigated Area=-20.0, Production_Category=Low...
```




```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('/content/maharashtra_aggregate_1966_2017.csv')

# Define the columns of interest
columns_of_interest = [
    'Area', 'Irrigated Area', 'NITROGEN CONSUMPTION (tons)',
    'POTASH CONSUMPTION (tons)', 'PERMANENT PASTURES AREA (1000 ha)',
    'OTHER FALLOW AREA (1000 ha)', 'GROSS CROPPED AREA (1000 ha)',
    'PHOSPHATE CONSUMPTION (tons)', 'Max Temp (Centigrate)',
    'Annual Rainfall', 'Yield', 'Production'
]
```

```
# Subset the data to keep only the columns of interest
df_subset = df[columns_of_interest].dropna()

# Discretize numerical features
df_subset['Yield_Category'] = pd.cut(
    df_subset['Yield'], bins=[0, 1000, 2000, 3000, float('inf')],
    labels=['Low', 'Medium', 'High', 'Very High']
)
df_subset['Production_Category'] = pd.cut(
    df_subset['Production'], bins=[0, 10000, 20000, 30000, float('inf')],
    labels=['Low', 'Medium', 'High', 'Very High']
)

# Drop the original Yield and Production columns after discretization
df_subset = df_subset.drop(['Yield', 'Production'], axis=1)

# Convert to transactional format
transactions = df_subset.apply(
    lambda row: [f"{col}={row[col]}" for col in df_subset.columns if pd.notnull(row[col])], axis=
).tolist()

# Encode transactions
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_array, columns=te.columns_)

# Apply FP-Growth with lower support threshold
frequent_itemsets = fpgrowth(df_encoded, min_support=0.05, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()
# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6, num_itemsets

# Add antecedent and consequent filtering for Yield and Production
yield_rules = rules[rules['consequents'].apply(lambda x: any('Yield_Category=' in item for item i
production_rules = rules[rules['consequents'].apply(lambda x: any('Production_Category=' in item

# Display rules related to Yield and Production
print("\nYield-Related Rules:")
print(yield_rules)

print("\nProduction-Related Rules:")
print(production_rules)

# Visualize top frequent itemsets
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(len)
top_itemsets = frequent_itemsets[frequent_itemsets['length'] > 1].nlargest(10, 'support')

plt.figure(figsize=(12, 6))
plt.barh(
    top_itemsets['itemsets'].apply(lambda x: ', '.join(list(x))),
    top_itemsets['support'],
    color='skyblue'
)
```

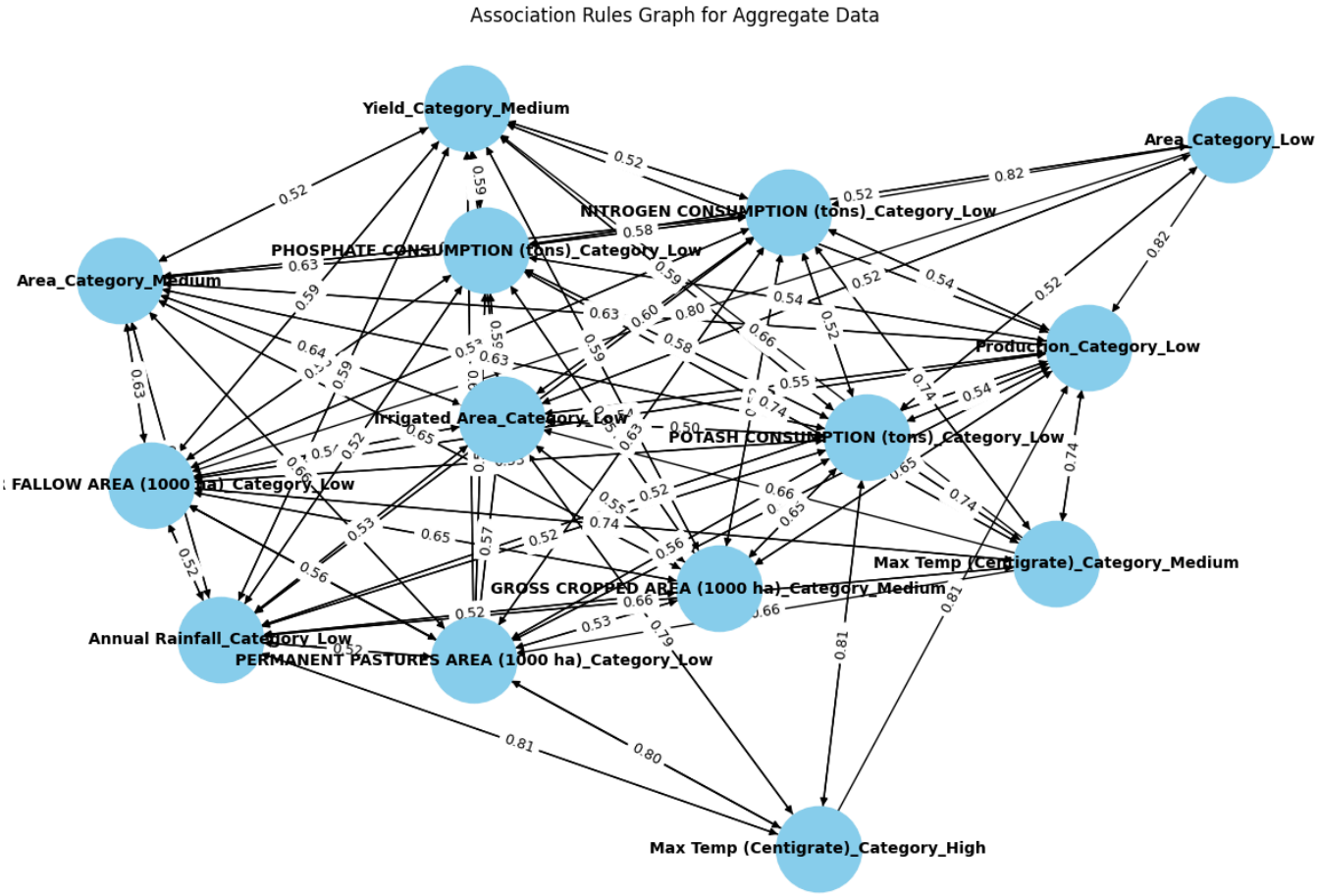
```
plt.xlabel('Support')
plt.ylabel('Itemsets')
plt.title('Top Frequent Itemsets')
plt.gca().invert_yaxis()
plt.show()

# Plot findings for Yield rules
plt.figure(figsize=(12, 6))
plt.barh(
    yield_rules['consequents'].apply(lambda x: ', '.join(list(x))),
    yield_rules['confidence'],
    color='orange'
)
plt.xlabel('Confidence')
plt.ylabel('Consequents')
plt.title('Rules with Consequents Related to Yield')
plt.gca().invert_yaxis()
plt.show()

# Plot findings for Production rules
plt.figure(figsize=(12, 6))
plt.barh(
    production_rules['consequents'].apply(lambda x: ', '.join(list(x))),
    production_rules['confidence'],
    color='green'
)
plt.xlabel('Confidence')
plt.ylabel('Consequents')
plt.title('Rules with Consequents Related to Production')
plt.gca().invert_yaxis()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Frequent Itemsets:
    support                                        itemsets
0  1.000000                          (Production_Category=Low)
1  0.999164                         (Yield_Category=Very High)
2  0.173913                            (Irrigated Area=-20.0)
3  0.999164  (Production_Category=Low, Yield_Category=Very ...
4  0.173913    (Irrigated Area=-20.0, Production_Category=Low)
5  0.173913   (Irrigated Area=-20.0, Yield_Category=Very High)
6  0.173913  (Irrigated Area=-20.0, Production_Category=Low...

Yield-Related Rules:
                                   antecedents  \
0                       (Production_Category=Low)
3                          (Irrigated Area=-20.0)
4  (Irrigated Area=-20.0, Production_Category=Low)
6                          (Irrigated Area=-20.0)

                                     consequents  antecedent support  \
0                         (Yield_Category=Very High)            1.000000
3                         (Yield_Category=Very High)            0.173913
4                         (Yield_Category=Very High)            0.173913
6  (Production_Category=Low, Yield_Category=Very ...            0.173913

   consequent support   support  confidence      lift  representativity  \
0            0.999164  0.999164    0.999164  1.000000               1.0
3            0.999164  0.173913    1.000000  1.000837               1.0
4            0.999164  0.173913    1.000000  1.000837               1.0
6            0.999164  0.173913    1.000000  1.000837               1.0

   leverage  conviction  zhangs_metric   jaccard  certainty  kulczynski
0  0.000000         1.0       0.000000  0.999164        0.0    0.999582
3  0.000145         inf       0.001012  0.174059        1.0    0.587029
4  0.000145         inf       0.001012  0.174059        1.0    0.587029
6  0.000145         inf       0.001012  0.174059        1.0    0.587029

Production-Related Rules:
                                   antecedents  \
1                       (Yield_Category=Very High)
2                          (Irrigated Area=-20.0)
5  (Irrigated Area=-20.0, Yield_Category=Very High)
6                          (Irrigated Area=-20.0)

                                     consequents  antecedent support  \
1                         (Production_Category=Low)            0.999164
2                         (Production_Category=Low)            0.173913
5                         (Production_Category=Low)            0.173913
6  (Production_Category=Low, Yield_Category=Very ...            0.173913

   consequent support   support  confidence      lift  representativity  \
1            1.000000  0.999164         1.0  1.000000               1.0
2            1.000000  0.173913         1.0  1.000000               1.0
5            1.000000  0.173913         1.0  1.000000               1.0
6            0.999164  0.173913         1.0  1.000837               1.0

   leverage  conviction  zhangs_metric   jaccard  certainty  kulczynski
```

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/maharashtra_aggregate_1966_2017.csv'
data = pd.read_csv(file_path)

# Columns of interest
columns_of_interest = [
    'Area', 'Irrigated Area', 'NITROGEN CONSUMPTION (tons)', 'POTASH CONSUMPTION (tons)',
    'PERMANENT PASTURES AREA (1000 ha)', 'OTHER FALLOW AREA (1000 ha)', 'GROSS CROPPED AREA (1000
    'PHOSPHATE CONSUMPTION (tons)', 'Max Temp (Centigrate)', 'Annual Rainfall', 'Yield', 'Product
]

# Focus on 'Yield' and 'Production' for association rules
```

```
data = data[columns_of_interest]

# Discretize the columns into categories
def discretize_column(col):
    return pd.cut(col, bins=3, labels=["Low", "Medium", "High"])

# Discretizing the relevant columns
for col in columns_of_interest:
    data[f'{col}_Category'] = discretize_column(data[col])

# One-hot encode the discretized categories
one_hot_data = pd.get_dummies(data[[f'{col}_Category' for col in columns_of_interest]])

# Apply Apriori and extract association rules
frequent_itemsets = apriori(one_hot_data, min_support=0.3, use_colnames=True)


num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Ensure we have itemsets before applying association rules
if len(frequent_itemsets) > 0:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5,num_items
else:
    rules = pd.DataFrame()  # If no frequent itemsets, return empty dataframe

# Function to draw association rule graph
def draw_rules_graph(rules, title="Association Rules Graph"):
    G = nx.DiGraph()
    for _, rule in rules.iterrows():
        for antecedent in rule['antecedents']:
            for consequent in rule['consequents']:
                G.add_edge(antecedent, consequent, weight=rule['confidence'])

    pos = nx.spring_layout(G, k=1)
    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, with_labels=True, node_size=3000, node_color="skyblue", font_size=10, font_we
    edge_labels = nx.get_edge_attributes(G, 'weight')
    edge_labels = {k: f'{v:.2f}' for k, v in edge_labels.items()}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)
    plt.title(title)
    plt.show()

# If there are any rules, draw the graph
if not rules.empty:
    draw_rules_graph(rules, title="Association Rules Graph for Aggregate Data")

# Display the first few rules
print("Association Rules Extracted:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
```



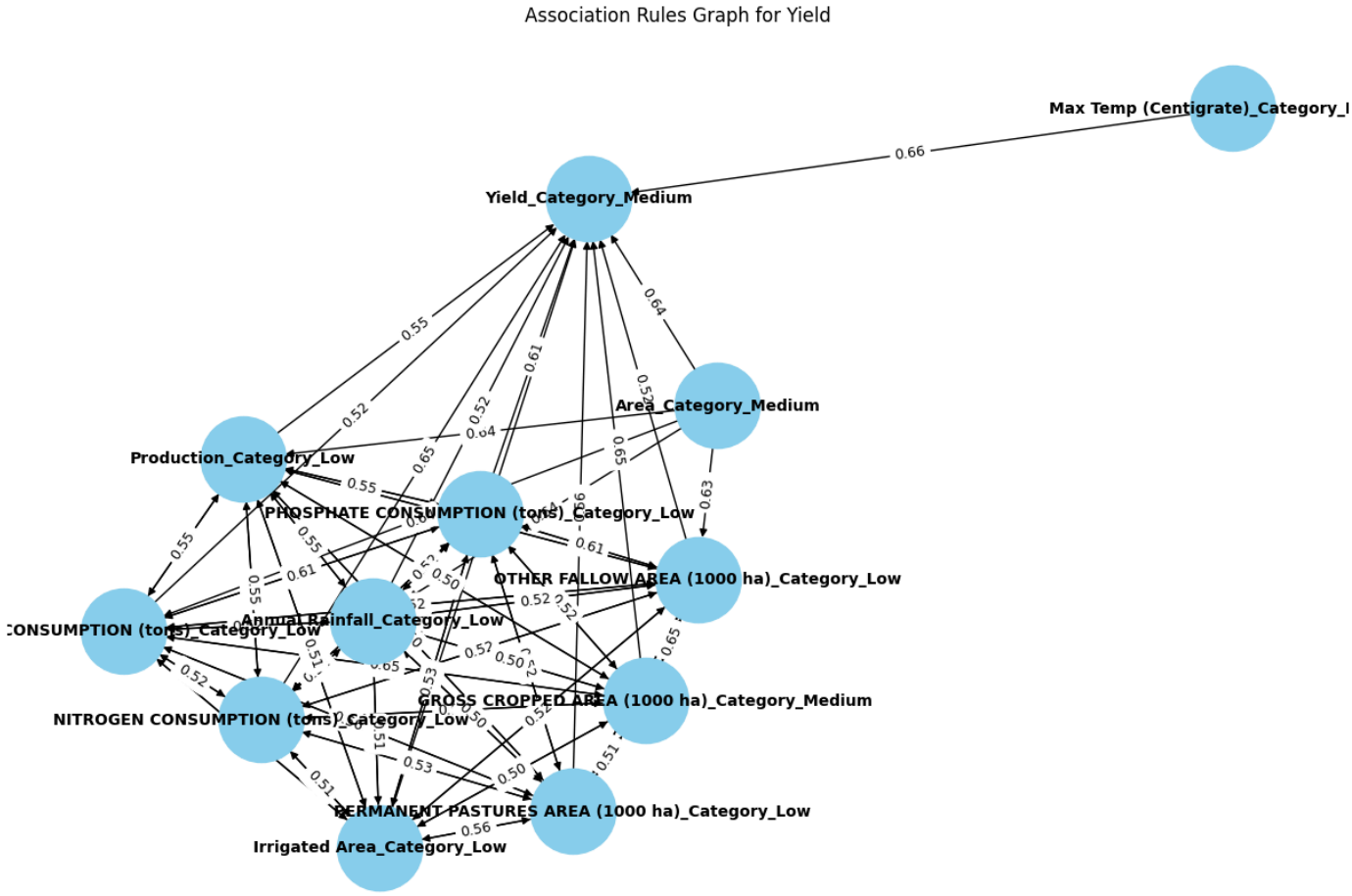

```
Association Rules Extracted:
           antecedents                                consequents   support  \
0  (Area_Category_Low)                (Irrigated Area_Category_Low)  0.350334
1  (Area_Category_Low)   (NITROGEN CONSUMPTION (tons)_Category_Low)  0.333612
2  (Area_Category_Low)     (POTASH CONSUMPTION (tons)_Category_Low)  0.390468
3  (Area_Category_Low)   (OTHER FALLOW AREA (1000 ha)_Category_Low)  0.355351
4  (Area_Category_Low)  (PHOSPHATE CONSUMPTION (tons)_Category_Low)  0.357023

   confidence      lift
0    0.876569  1.183269
1    0.834728  1.074634
2    0.976987  1.022290
3    0.889121  0.985532
4    0.893305  1.091311
```

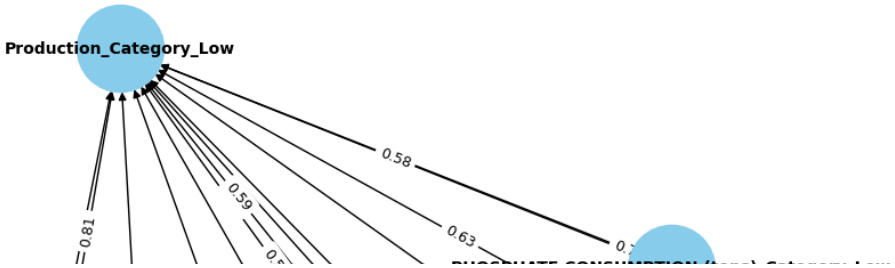
```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/maharashtra_aggregate_1966_2017.csv'
data = pd.read_csv(file_path)

# Columns of interest
columns_of_interest = [
    'Area', 'Irrigated Area', 'NITROGEN CONSUMPTION (tons)', 'POTASH CONSUMPTION (tons)',
    'PERMANENT PASTURES AREA (1000 ha)', 'OTHER FALLOW AREA (1000 ha)', 'GROSS CROPPED AREA (1000
    'PHOSPHATE CONSUMPTION (tons)', 'Max Temp (Centigrate)', 'Annual Rainfall', 'Yield', 'Product
]

# Focus on 'Yield' and 'Production' for association rules
```

```
data = data[columns_of_interest]

# Discretize the columns into categories
def discretize_column(col):
    return pd.cut(col, bins=3, labels=["Low", "Medium", "High"])

# Discretizing the relevant columns
for col in columns_of_interest:
    data[f'{col}_Category'] = discretize_column(data[col])

# One-hot encode the discretized categories
one_hot_data = pd.get_dummies(data[[f'{col}_Category' for col in columns_of_interest]])

# Apply Apriori and extract association rules
frequent_itemsets = apriori(one_hot_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Ensure we have itemsets before applying association rules
if len(frequent_itemsets) > 0:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5,num_items
else:
    rules = pd.DataFrame()  # If no frequent itemsets, return empty dataframe

# Separate rules related to Yield and Production
yield_rules = rules[rules['consequents'].apply(lambda x: any("Yield_Category" in str(item) for it
production_rules = rules[rules['consequents'].apply(lambda x: any("Production_Category" in str(it

# Function to draw association rule graphs
def draw_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()
    for _, rule in rules_subset.iterrows():
        for antecedent in rule['antecedents']:
            for consequent in rule['consequents']:
                G.add_edge(antecedent, consequent, weight=rule['confidence'])

    pos = nx.spring_layout(G, k=1)
    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, with_labels=True, node_size=3000, node_color="skyblue", font_size=10, font_we
    edge_labels = nx.get_edge_attributes(G, 'weight')
    edge_labels = {k: f'{v:.2f}' for k, v in edge_labels.items()}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)
    plt.title(title)
    plt.show()

# Plot graphs for Yield and Production separately
if not yield_rules.empty:
    draw_rules_graph(yield_rules, title="Association Rules Graph for Yield")

if not production_rules.empty:
    draw_rules_graph(production_rules, title="Association Rules Graph for Production")

# Display the first few rules for each
print("Yield-Related Rules:")
print(yield_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())

print("\nProduction-Related Rules:")
```

```
print(production_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
```



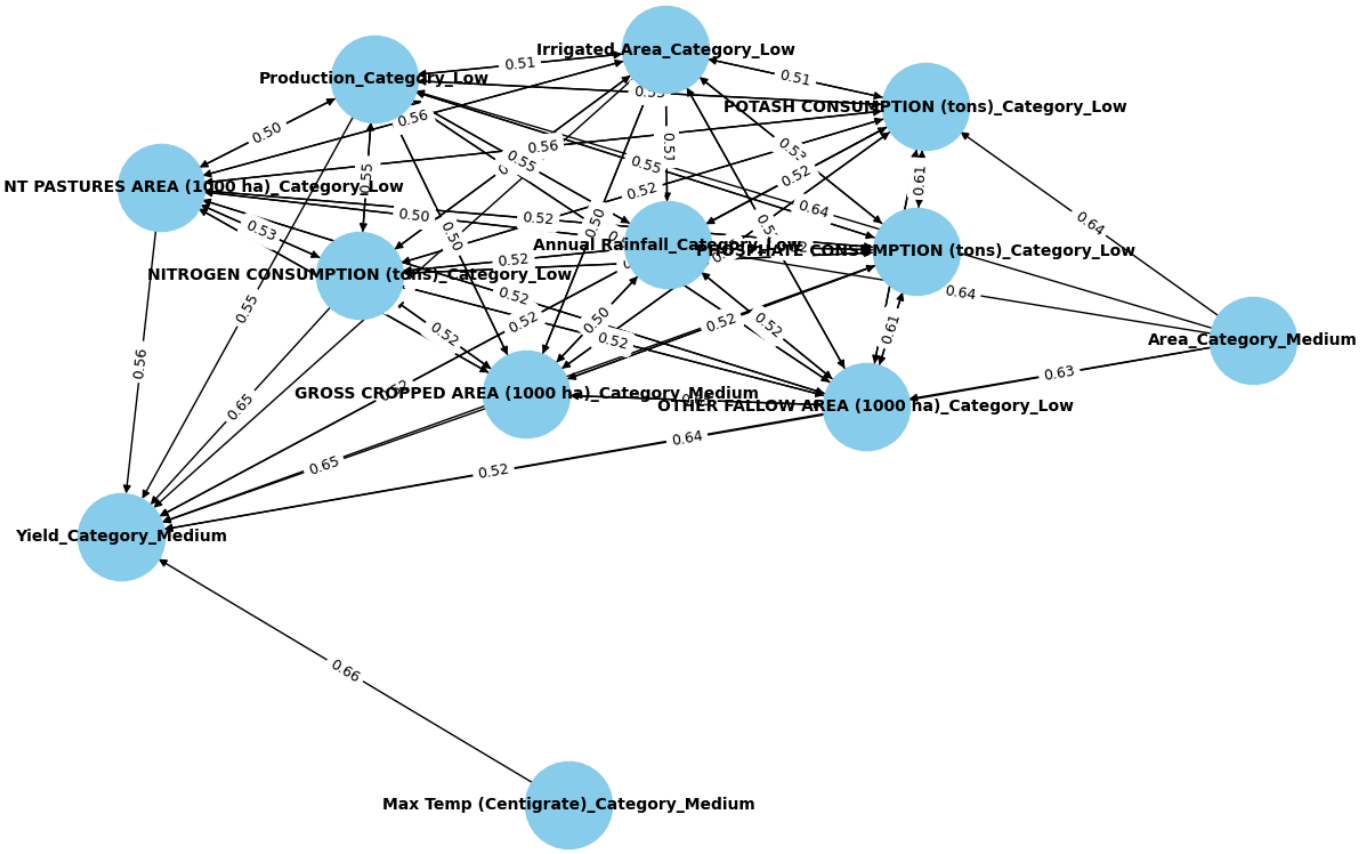



```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/maharashtra_aggregate_1966_2017.csv'
data = pd.read_csv(file_path)

# Columns of interest
columns_of_interest = [
    'Area', 'Irrigated Area', 'NITROGEN CONSUMPTION (tons)', 'POTASH CONSUMPTION (tons)',
    'PERMANENT PASTURES AREA (1000 ha)', 'OTHER FALLOW AREA (1000 ha)', 'GROSS CROPPED AREA (1000
    'PHOSPHATE CONSUMPTION (tons)', 'Max Temp (Centigrate)', 'Annual Rainfall', 'Yield', 'Product
]
```

```
# Focus on 'Yield' and 'Production' for association rules
data = data[columns_of_interest]

# Discretize the columns into categories
def discretize_column(col):
    return pd.cut(col, bins=3, labels=["Low", "Medium", "High"])

# Discretizing the relevant columns
for col in columns_of_interest:
    data[f'{col}_Category'] = discretize_column(data[col])

# One-hot encode the discretized categories
one_hot_data = pd.get_dummies(data[[f'{col}_Category' for col in columns_of_interest]])

# Apply Apriori and extract association rules
frequent_itemsets = apriori(one_hot_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Ensure we have itemsets before applying association rules
if len(frequent_itemsets) > 0:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5,num_items
else:
    rules = pd.DataFrame()  # If no frequent itemsets, return empty dataframe

# Separate rules related to Yield and Production
yield_rules = rules[rules['consequents'].apply(lambda x: any("Yield_Category" in str(item) for it
production_rules = rules[rules['consequents'].apply(lambda x: any("Production_Category" in str(it

# Function to draw association rule graphs
def draw_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()
    for _, rule in rules_subset.iterrows():
        for antecedent in rule['antecedents']:
            for consequent in rule['consequents']:
                G.add_edge(antecedent, consequent, weight=rule['confidence'])

    pos = nx.spring_layout(G, k=1)
    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, with_labels=True, node_size=3000, node_color="skyblue", font_size=10, font_we
    edge_labels = nx.get_edge_attributes(G, 'weight')
    edge_labels = {k: f'{v:.2f}' for k, v in edge_labels.items()}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)
    plt.title(title)
    plt.show()

# Plot graphs for Yield and Production separately
if not yield_rules.empty:
    print("Yield-Related Rules:")
    print(yield_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())
    draw_rules_graph(yield_rules, title="Association Rules Graph for Yield")

if not production_rules.empty:
    print("\nProduction-Related Rules:")
    print(production_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(
    draw_rules_graph(production_rules, title="Association Rules Graph for Production")
```

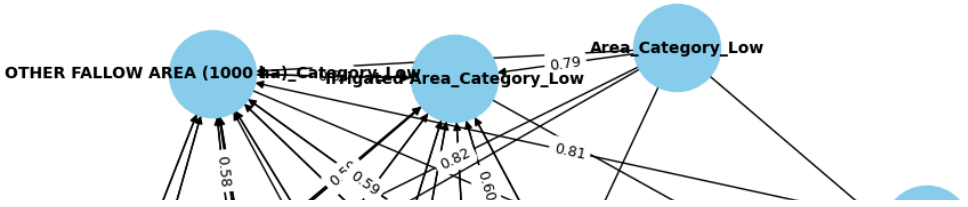
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
Yield-Related Rules:
                                      antecedents             consequents  \
19                          (Area_Category_Medium)  (Yield_Category_Medium)
38                   (Irrigated Area_Category_Low)  (Yield_Category_Medium)
55        (NITROGEN CONSUMPTION (tons)_Category_Low)  (Yield_Category_Medium)
71          (POTASH CONSUMPTION (tons)_Category_Low)  (Yield_Category_Medium)
84  (PERMANENT PASTURES AREA (1000 ha)_Category_Low)  (Yield_Category_Medium)

     support  confidence      lift
19  0.338629    0.700692  1.111443
38  0.447324    0.603837  0.957811
55  0.515050    0.663079  1.051780
71  0.616221    0.644794  1.022777
84  0.488294    0.655443  1.039669
```

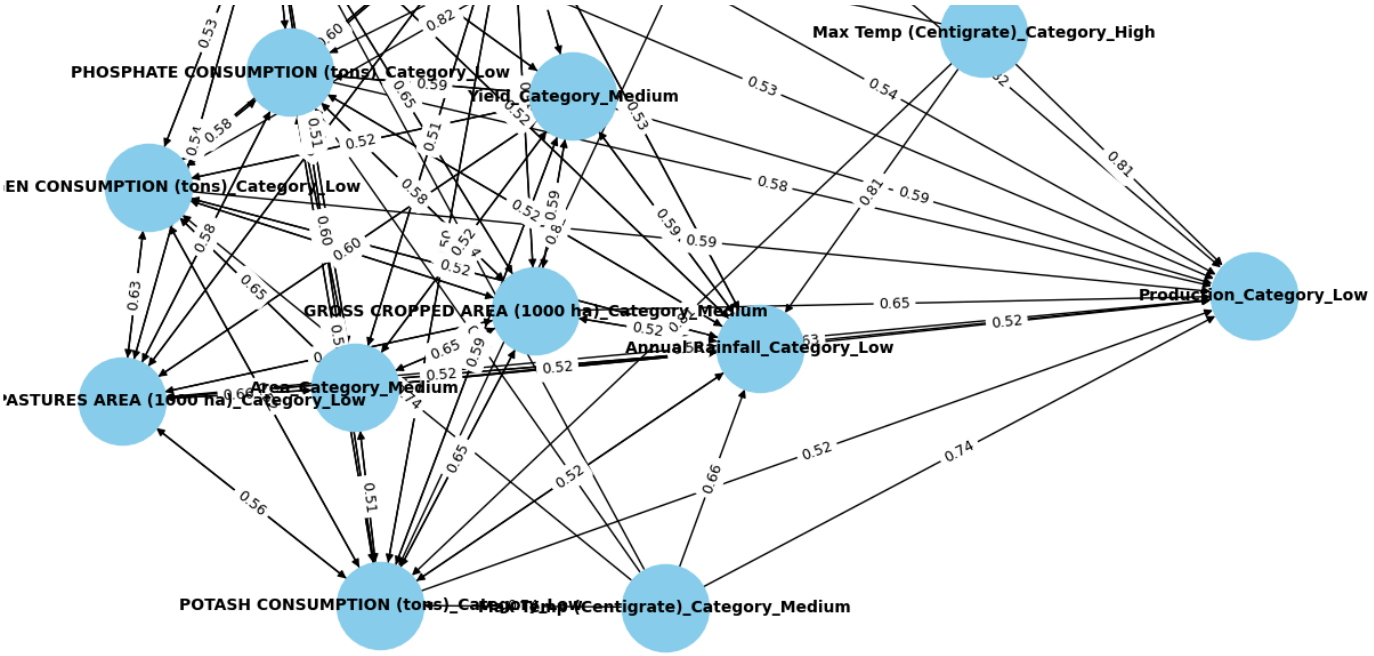
Association Rules Graph for Yield




```
Production-Related Rules:
                                    antecedents               consequents  \
5                          (Area_Category_Low)  (Production_Category_Low)
20                    (Area_Category_Medium)  (Production_Category_Low)
40               (Irrigated Area_Category_Low)  (Production_Category_Low)
57  (NITROGEN CONSUMPTION (tons)_Category_Low)  (Production_Category_Low)
73    (POTASH CONSUMPTION (tons)_Category_Low)  (Production_Category_Low)

     support  confidence      lift
5   0.368729    0.922594  1.060983
20  0.433110    0.896194  1.030623
40  0.661371    0.892777  1.026693
57  0.733278    0.944026  1.085630
73  0.853679    0.893263  1.027253
```

Association Rules Graph for Production

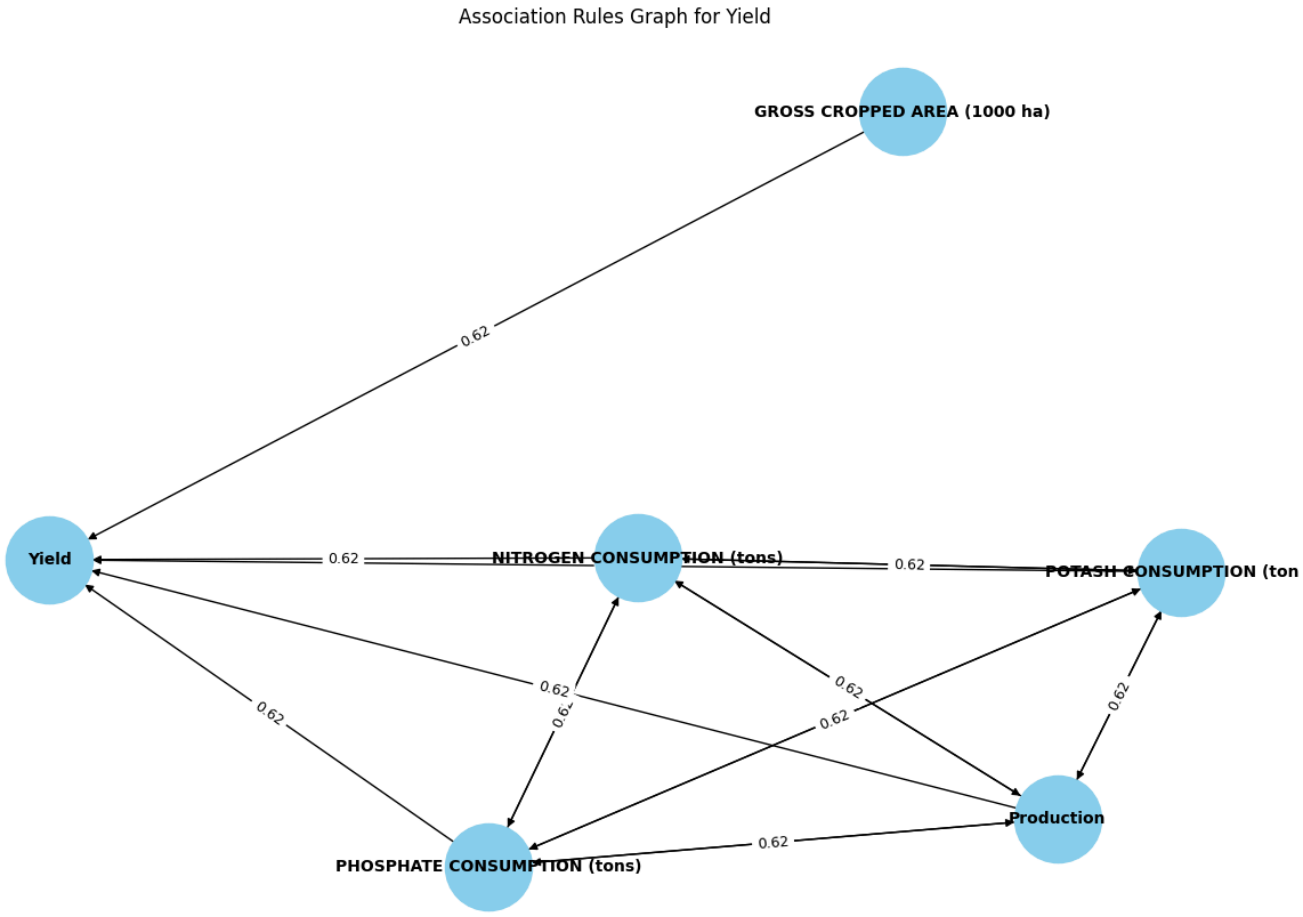





```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/maharashtra_aggregate_1966_2017.csv'
data = pd.read_csv(file_path)

# Columns of interest
columns_of_interest = [
     'Irrigated Area', 'NITROGEN CONSUMPTION (tons)', 'POTASH CONSUMPTION (tons)',
    'PERMANENT PASTURES AREA (1000 ha)', 'OTHER FALLOW AREA (1000 ha)', 'GROSS CROPPED AREA (1000
    'PHOSPHATE CONSUMPTION (tons)', 'Max Temp (Centigrate)', 'Annual Rainfall', 'Yield', 'Product
]

# Focus on 'Yield' and 'Production'
data = data[columns_of_interest]

# Convert continuous data to binary using median threshold
def binary_conversion(df):
    binary_data = df.apply(lambda col: (col > col.median()).astype(int), axis=0)
    return binary_data

# Apply binary conversion
binary_data = binary_conversion(data)

# Apply Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(binary_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()
# Ensure we have itemsets before applying association rules
```

```
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5,num_items
else:
    rules = pd.DataFrame()

# Filter rules for 'Yield' and 'Production'
yield_rules = rules[rules['consequents'].apply(lambda x: any('Yield' in str(item) for item in x))
production_rules = rules[rules['consequents'].apply(lambda x: any('Production' in str(item) for i

# Function to draw association rules graph
def draw_filtered_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()
    for _, rule in rules_subset.iterrows():
        for antecedent in rule['antecedents']:
            for consequent in rule['consequents']:
                G.add_edge(antecedent, consequent, weight=rule['confidence'])

    pos = nx.spring_layout(G, k=1)
    plt.figure(figsize=(12, 8))
    nx.draw(
        G, pos, with_labels=True, node_size=3000, node_color="skyblue",
        font_size=10, font_weight="bold", arrows=True
    )
    edge_labels = nx.get_edge_attributes(G, 'weight')
    edge_labels = {k: f'{v:.2f}' for k, v in edge_labels.items()}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)
    plt.title(title)
    plt.show()

# Display rules and plot graphs
if not yield_rules.empty:
    print("Yield-Related Rules:")
    print(yield_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())
    draw_filtered_rules_graph(yield_rules, title="Association Rules Graph for Yield")

if not production_rules.empty:
    print("\nProduction-Related Rules:")
    print(production_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(
    draw_filtered_rules_graph(production_rules, title="Association Rules Graph for Production")
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types
  warnings.warn(
Yield-Related Rules:
                       antecedents consequents   support  confidence      lift
17   (NITROGEN CONSUMPTION (tons))     (Yield)  0.375418    0.750836  1.501672
24     (POTASH CONSUMPTION (tons))     (Yield)  0.382107    0.764214  1.528428
32  (GROSS CROPPED AREA (1000 ha))     (Yield)  0.311873    0.623746  1.247492
36  (PHOSPHATE CONSUMPTION (tons))     (Yield)  0.372910    0.745819  1.491639
40                     (Production)     (Yield)  0.403846    0.807692  1.615385
```



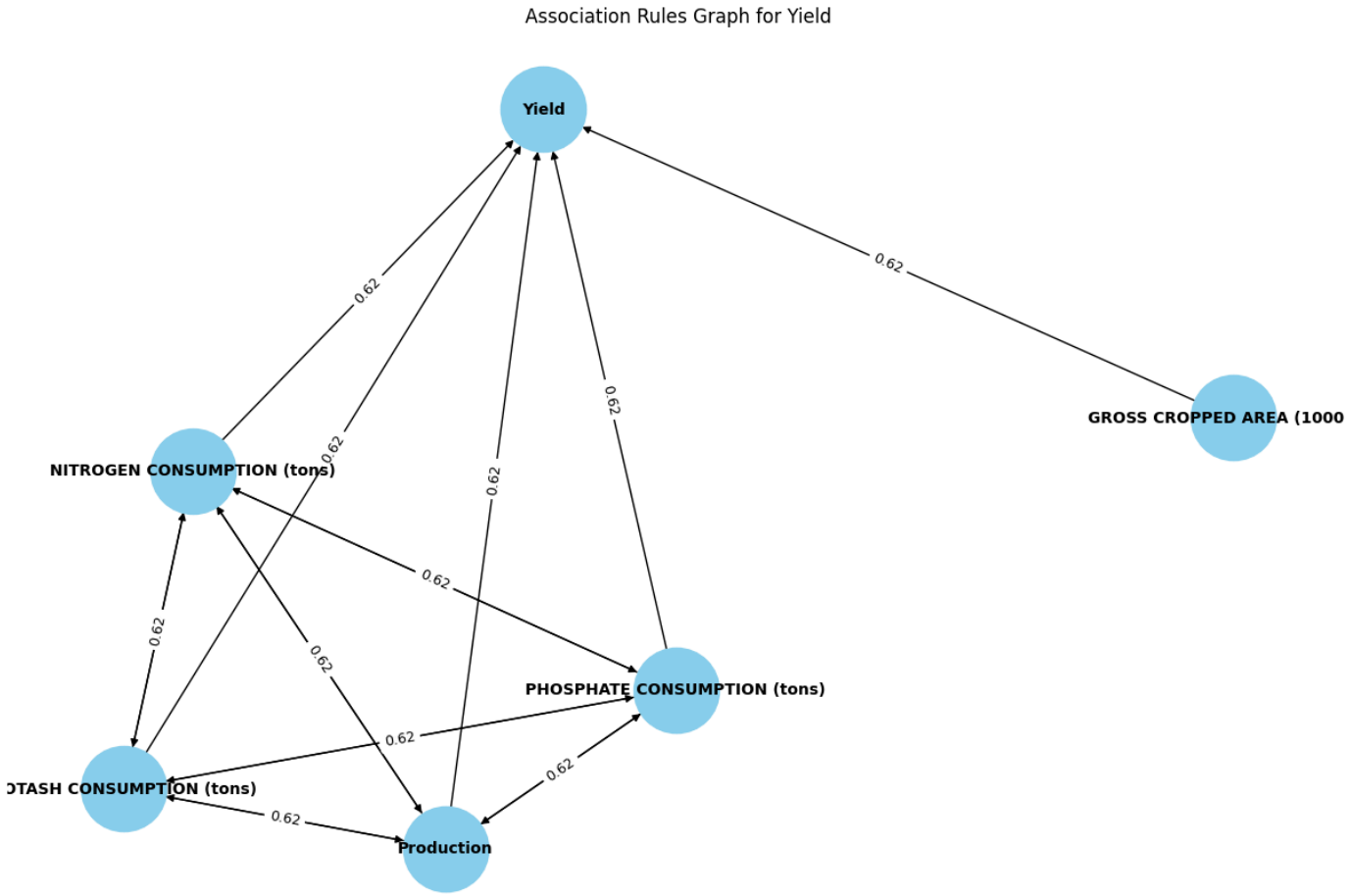

```
Production-Related Rules:
                      antecedents   consequents   support  confidence  \
4                            (Area)  (Production)  0.334448    0.668896
8                   (Irrigated Area)  (Production)  0.307692    0.616415
```

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/maharashtra_aggregate_1966_2017.csv'
data = pd.read_csv(file_path)

# Columns of interest
columns_of_interest = [
    'Area', 'Irrigated Area', 'NITROGEN CONSUMPTION (tons)', 'POTASH CONSUMPTION (tons)',
    'PERMANENT PASTURES AREA (1000 ha)', 'OTHER FALLOW AREA (1000 ha)', 'GROSS CROPPED AREA (1000
    'PHOSPHATE CONSUMPTION (tons)', 'Max Temp (Centigrate)', 'Annual Rainfall', 'Yield', 'Product
]

# Focus on 'Yield' and 'Production'
```

```
data = data[columns_of_interest]

# Convert continuous data to binary using median threshold
def binary_conversion(df):
    binary_data = df.apply(lambda col: (col > col.median()).astype(int), axis=0)
    return binary_data

# Apply binary conversion
binary_data = binary_conversion(data)


frequent_itemsets = apriori(binary_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Ensure we have itemsets before applying association rules
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5, num_item
else:
    rules = pd.DataFrame()

# Filter rules for 'Yield' and 'Production'
yield_rules = rules[rules['consequents'].apply(lambda x: any('Yield' in str(item) for item in x))
production_rules = rules[rules['consequents'].apply(lambda x: any('Production' in str(item) for i

# Function to draw association rules graph
def draw_filtered_rules_graph(rules_subset, title="Association Rules Graph"):
  G = nx.DiGraph()

  # Add edges with weights
  for _, rule in rules_subset.iterrows():
    antecedents = sorted(list(rule['antecedents']))  # Convert set to sorted list
    consequents = sorted(list(rule['consequents']))
    for antecedent in antecedents:
      for consequent in consequents:
        # Access confidence from the current rule
        confidence = rule['confidence']
        G.add_edge(antecedent, consequent, weight=confidence)

  # Graph layout
  pos = nx.spring_layout(G, k=1)

  # Draw the graph
  plt.figure(figsize=(12, 8))
  nx.draw(
      G, pos, with_labels=True, node_size=3000, node_color="skyblue",
      font_size=10, font_weight="bold", arrows=True
  )

  # Add edge labels (weights)
  edge_labels = {(antecedent, consequent): f'{weight:.2f}'
                 for antecedent, consequent, weight in G.edges(data='weight')}
  nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

  # Add title and display
  plt.title(title)
  plt.show()
```

```
# Display rules and plot graphs
if not yield_rules.empty:
    print("Yield-Related Rules:")
    print(yield_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())
    draw_filtered_rules_graph(yield_rules, title="Association Rules Graph for Yield")

if not production_rules.empty:
    print("\nProduction-Related Rules:")
    print(production_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(
    draw_filtered_rules_graph(production_rules, title="Association Rules Graph for Production")
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types
  warnings.warn(
Yield-Related Rules:
                      antecedents consequents   support  confidence      lift
17   (NITROGEN CONSUMPTION (tons))     (Yield)  0.375418    0.750836  1.501672
24     (POTASH CONSUMPTION (tons))     (Yield)  0.382107    0.764214  1.528428
32  (GROSS CROPPED AREA (1000 ha))     (Yield)  0.311873    0.623746  1.247492
36  (PHOSPHATE CONSUMPTION (tons))     (Yield)  0.372910    0.745819  1.491639
40                     (Production)     (Yield)  0.403846    0.807692  1.615385
```

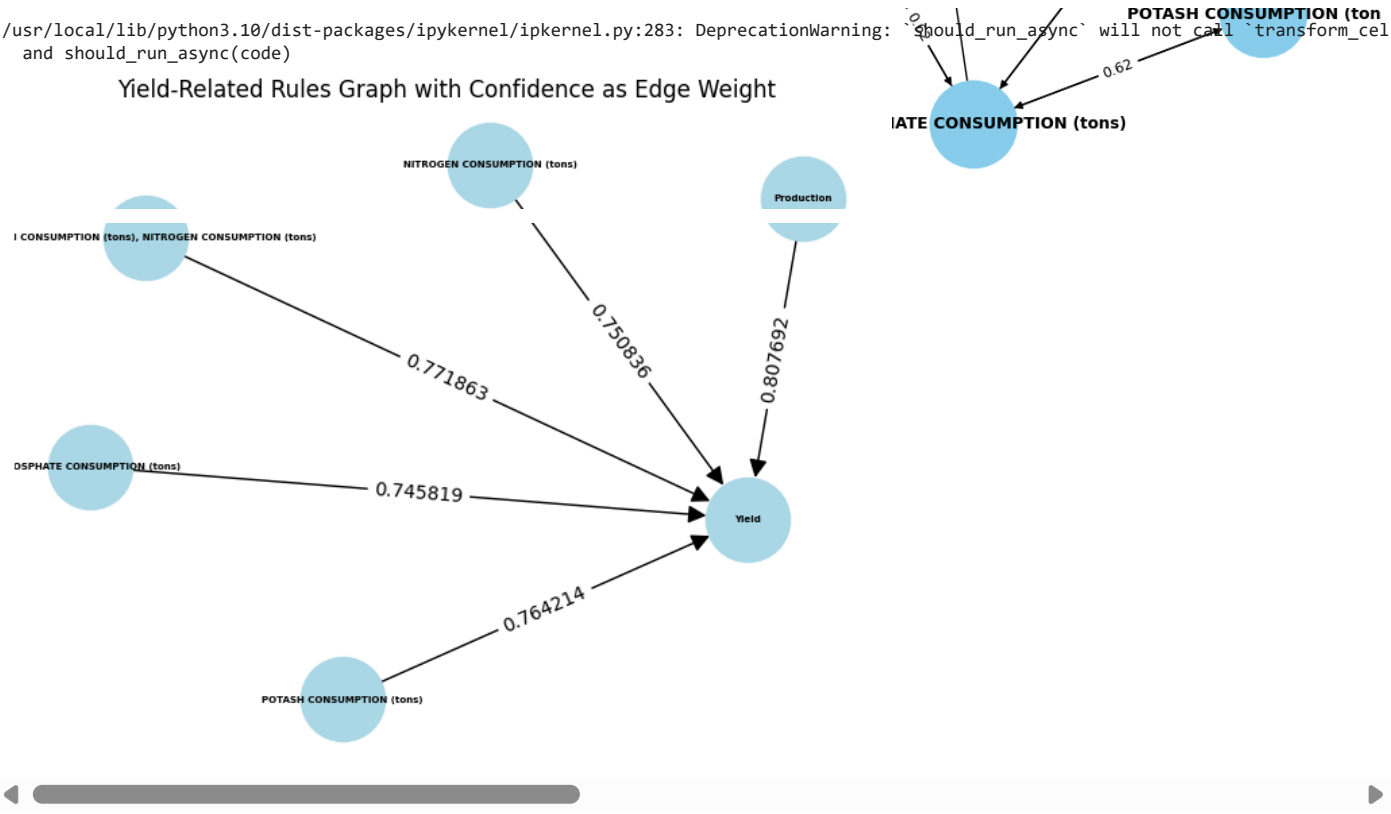



```
Production-Related Rules:
                      antecedents   consequents   support  confidence  \
4                          (Area)  (Production)  0.334448    0.668896
8                (Irrigated Area)  (Production)  0.307692    0.616415
```

```
import networkx as nx
import matplotlib.pyplot as plt

# Data for yield-related apriori rules
yield_rules = [
    {"antecedents": "NITROGEN CONSUMPTION (tons)", "consequents": "Yield", "confidence": 0.750836
```

```
    {"antecedents": "POTASH CONSUMPTION (tons)", "consequents": "Yield", "confidence": 0.764214},
    {"antecedents": "PHOSPHATE CONSUMPTION (tons)", "consequents": "Yield", "confidence": 0.74581
    {"antecedents": "Production", "consequents": "Yield", "confidence": 0.807692},
    {"antecedents": "POTASH CONSUMPTION (tons), NITROGEN CONSUMPTION (tons)", "consequents": "Yie
]

# Create directed graph
G = nx.DiGraph()

# Add edges with weights based on confidence
for rule in yield_rules:
    G.add_edge(rule["antecedents"], rule["consequents"], weight=rule["confidence"])

# Draw the graph
pos = nx.spring_layout(G, seed=42)  # Layout for better visualization
nx.draw(G, pos, with_labels=True, node_size=2000, node_color='lightblue', font_size=5, font_weigh
nx.draw_networkx_edge_labels(G, pos, edge_labels={(u, v): f'{d["weight"]:.6f}' for u, v, d in G.e

# Display the graph
plt.title("Yield-Related Rules Graph with Confidence as Edge Weight")
plt.show()
```

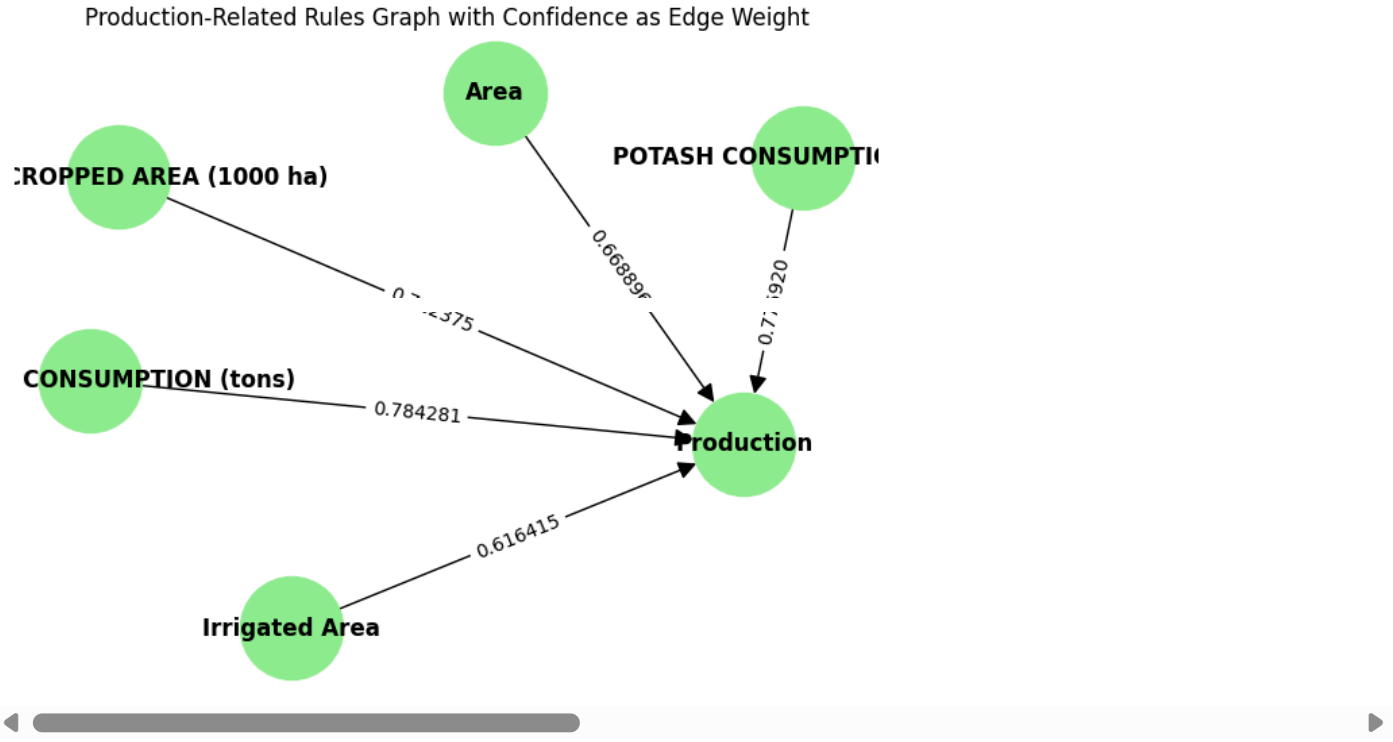
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
```




```
import networkx as nx
import matplotlib.pyplot as plt

# Data for production-related apriori rules
production_rules = [
    {"antecedents": "Area", "consequents": "Production", "confidence": 0.668896},
    {"antecedents": "Irrigated Area", "consequents": "Production", "confidence": 0.616415},
    {"antecedents": "NITROGEN CONSUMPTION (tons)", "consequents": "Production", "confidence": 0.7
    {"antecedents": "POTASH CONSUMPTION (tons)", "consequents": "Production", "confidence": 0.775
    {"antecedents": "GROSS CROPPED AREA (1000 ha)", "consequents": "Production", "confidence": 0.
]
```

```
# Create directed graph
G = nx.DiGraph()

# Add edges with weights based on confidence
for rule in production_rules:
    G.add_edge(rule["antecedents"], rule["consequents"], weight=rule["confidence"])

# Draw the graph
pos = nx.spring_layout(G, seed=42)  # Layout for better visualization
nx.draw(G, pos, with_labels=True, node_size=3000, node_color='lightgreen', font_size=12, font_wei
nx.draw_networkx_edge_labels(G, pos, edge_labels={(u, v): f'{d["weight"]:.6f}' for u, v, d in G.e

# Display the graph
plt.title("Production-Related Rules Graph with Confidence as Edge Weight")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
```

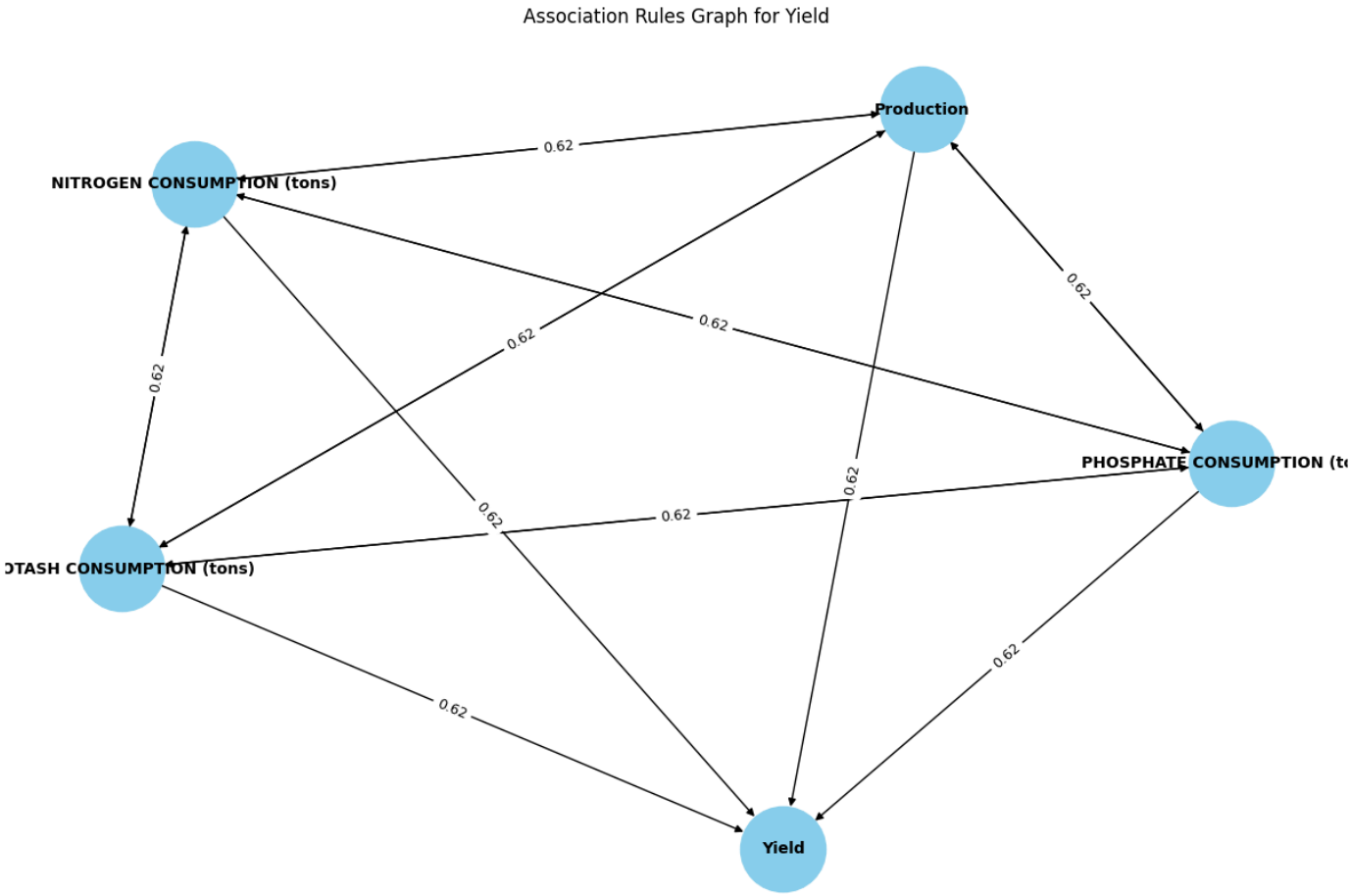



```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/maharashtra_aggregate_1966_2017.csv'
data = pd.read_csv(file_path)

# Top 10 features of interest
top_10_features = [
    'Irrigated Area', 'NITROGEN CONSUMPTION (tons)', 'POTASH CONSUMPTION (tons)', 'PHOSPHATE CONS
    'Max Temp (Centigrate)', 'Annual Rainfall']

# Include 'Yield' and 'Production' as features
selected_features = top_10_features + ['Yield', 'Production']
```

```
filtered_data = data[selected_features]

# Convert continuous data to binary using median threshold
def binary_conversion(df):
    binary_data = df.apply(lambda col: (col > col.median()).astype(int), axis=0)
    return binary_data

# Apply binary conversion
binary_data = binary_conversion(filtered_data)

# Apply apriori algorithm
frequent_itemsets = apriori(binary_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Apply association rules
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5, num_item
else:
    rules = pd.DataFrame()

# Split rules for 'Yield' and 'Production' separately
yield_rules = rules[rules['consequents'].apply(lambda x: any('Yield' in str(item) for item in x))
production_rules = rules[rules['consequents'].apply(lambda x: any('Production' in str(item) for i

# Function to draw association rules graph
def draw_filtered_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()

    # Add edges with weights
    for _, rule in rules_subset.iterrows():
        antecedents = sorted(list(rule['antecedents']))  # Convert set to sorted list
        consequents = sorted(list(rule['consequents']))
        for antecedent in antecedents:
            for consequent in consequents:
                confidence = rule['confidence']
                G.add_edge(antecedent, consequent, weight=confidence)

    # Graph layout
    pos = nx.spring_layout(G, k=1)

    # Draw the graph
    plt.figure(figsize=(12, 8))
    nx.draw(
        G, pos, with_labels=True, node_size=3000, node_color="skyblue",
        font_size=10, font_weight="bold", arrows=True
    )

    # Add edge labels (weights)
    edge_labels = {(antecedent, consequent): f'{weight:.2f}'
                   for antecedent, consequent, weight in G.edges(data='weight')}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

    # Add title and display
    plt.title(title)
    plt.show()

# Display rules and plot graphs
```

```
if not yield_rules.empty:
    print("Yield-Related Rules:")
    print(yield_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())
    draw_filtered_rules_graph(yield_rules, title="Association Rules Graph for Yield")

if not production_rules.empty:
    print("\nProduction-Related Rules:")
    print(production_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(
    draw_filtered_rules_graph(production_rules, title="Association Rules Graph for Production")
```

```
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types
  warnings.warn(
Yield-Related Rules:
                                    antecedents consequents   support  \
6                     (NITROGEN CONSUMPTION (tons))     (Yield)  0.375418
12                      (POTASH CONSUMPTION (tons))     (Yield)  0.382107
16                   (PHOSPHATE CONSUMPTION (tons))     (Yield)  0.372910
20                                     (Production)     (Yield)  0.403846
28  (POTASH CONSUMPTION (tons), NITROGEN CONSUMPTI...     (Yield)  0.339465

    confidence      lift
6     0.750836  1.501672
12    0.764214  1.528428
16    0.745819  1.491639
20    0.807692  1.615385
28    0.771863  1.543726
```




```
#https://drive.google.com/file/d/1SxtA2qvR2jyRibDYoTUflQQ11TNywF33/view?usp=sharing

import gdown
file_id = '1SxtA2qvR2jyRibDYoTUflQQ11TNywF33'
url = f'https://drive.google.com/uc?id={file_id}'
output = 'Major_5_Crops_Dataset.csv'
gdown.download(url, output, quiet=False)
```

```
import pandas as pd
df = pd.read_csv('Major_5_Crops_Dataset.csv')
```

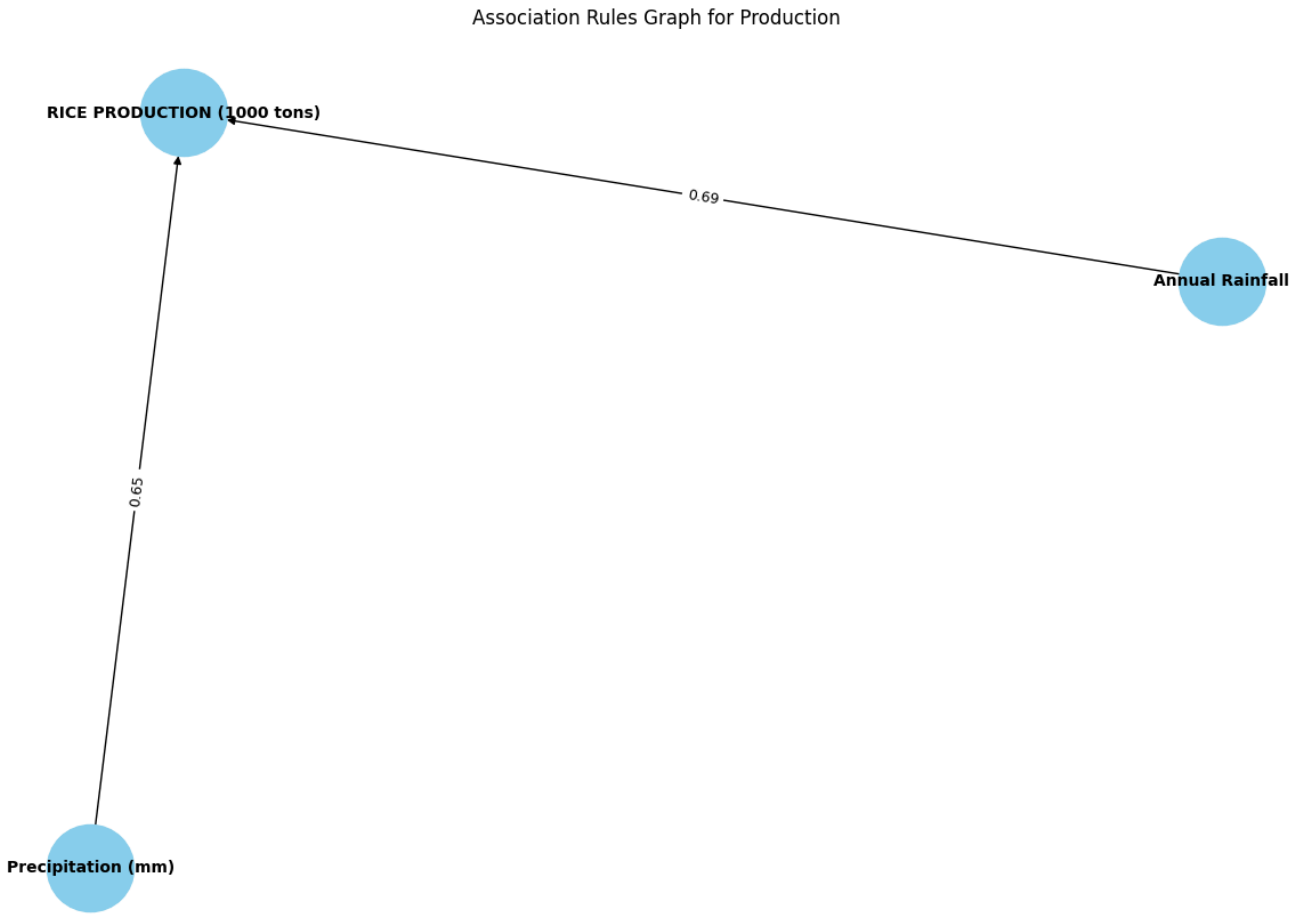
```
Downloading...
From: https://drive.google.com/uc?id=1SxtA2qvR2jyRibDYoTUflQQ11TNywF33
To: /content/Major_5_Crops_Dataset.csv
100%|██████████| 130k/130k [00:00<00:00, 10.3MB/s]
```


```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/Major_5_Crops_Dataset.csv'
data = pd.read_csv(file_path)

# Top 10 features of interest
top_10_features = ['Precipitation (mm)','Annual Rainfall','Irrigated Area','POTASH SHARE IN NPK (

# Include 'Yield' and 'Production' as features
selected_features = top_10_features + ['RICE PRODUCTION (1000 tons)',]
filtered_data = data[selected_features]

# Convert continuous data to binary using median threshold
def binary_conversion(df):
    binary_data = df.apply(lambda col: (col > col.median()).astype(int), axis=0)
    return binary_data

# Apply binary conversion
binary_data = binary_conversion(filtered_data)

# Apply apriori algorithm
frequent_itemsets = apriori(binary_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Apply association rules
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5, num_item
else:
    rules = pd.DataFrame()

# Split rules for 'Yield' and 'Production' separately
# Filter rules for 'RICE YIELD (Kg per ha)' and 'RICE PRODUCTION (1000 tons)'
production_rules = rules[rules['consequents'].apply(lambda x: any('RICE PRODUCTION (1000 tons)' i

# Function to draw association rules graph
def draw_filtered_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()

    # Add edges with weights
    for _, rule in rules_subset.iterrows():
        antecedents = sorted(list(rule['antecedents']))  # Convert set to sorted list
        consequents = sorted(list(rule['consequents']))
        for antecedent in antecedents:
            for consequent in consequents:
                confidence = rule['confidence']
                G.add_edge(antecedent, consequent, weight=confidence)
```

Association Rules Graph for Production

```
    # Graph layout
    pos = nx.spring_layout(G, k=1)

    # Draw the graph
    plt.figure(figsize=(12, 8))
    nx.draw(
        G, pos, with_labels=True, node_size=3000, node_color="skyblue",
        font_size=10, font_weight="bold", arrows=True
    )

    # Add edge labels (weights)
    edge_labels = {(antecedent, consequent): f'{weight:.2f}'
                   for antecedent, consequent, weight in G.edges(data='weight')}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

    # Add title and display
    plt.title(title)
    plt.show()

# Display rules and plot graphs
if not production_rules.empty:
    print("\nProduction-Related Rules:")
    print(production_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(
    draw_filtered_rules_graph(production_rules, title="Association Rules Graph for Production")
```

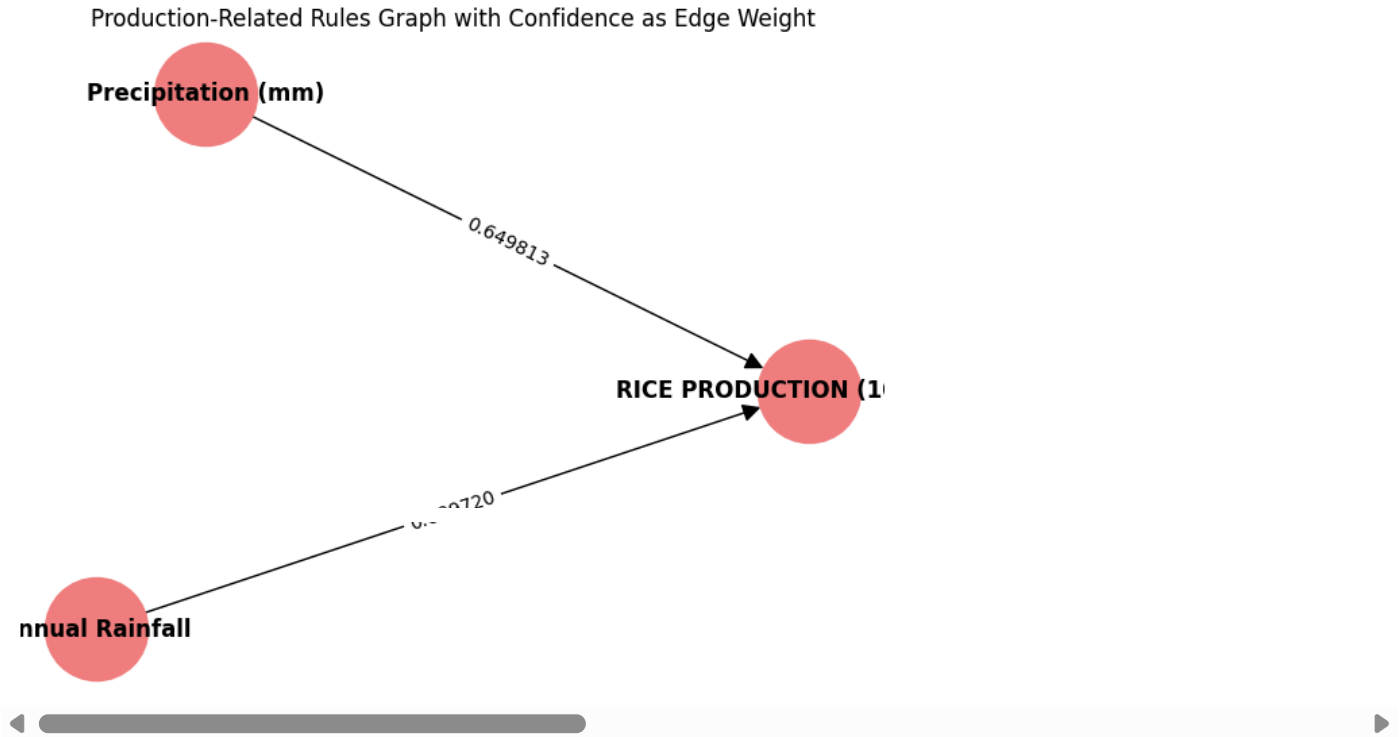
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types
  warnings.warn(

Production-Related Rules:
            antecedents                   consequents   support  confidence  \
2  (Precipitation (mm))  (RICE PRODUCTION (1000 tons))  0.323996    0.649813
5     (Annual Rainfall)  (RICE PRODUCTION (1000 tons))  0.344538    0.689720

       lift
2  1.330687
5  1.412409
```




```
import networkx as nx
import matplotlib.pyplot as plt

# Data for production-related apriori rules
production_rules = [
    {"antecedents": "Precipitation (mm)", "consequents": "RICE PRODUCTION (1000 tons)", "confiden
    {"antecedents": "Annual Rainfall", "consequents": "RICE PRODUCTION (1000 tons)", "confidence"
]

# Create directed graph
G = nx.DiGraph()

# Add edges with weights based on confidence
for rule in production_rules:
    G.add_edge(rule["antecedents"], rule["consequents"], weight=rule["confidence"])

# Draw the graph
```

```
pos = nx.spring_layout(G, seed=42)  # Layout for better visualization
nx.draw(G, pos, with_labels=True, node_size=3000, node_color='lightcoral', font_size=12, font_wei
nx.draw_networkx_edge_labels(G, pos, edge_labels={(u, v): f'{d["weight"]:.6f}' for u, v, d in G.e

# Display the graph
plt.title("Production-Related Rules Graph with Confidence as Edge Weight")
plt.show()
```

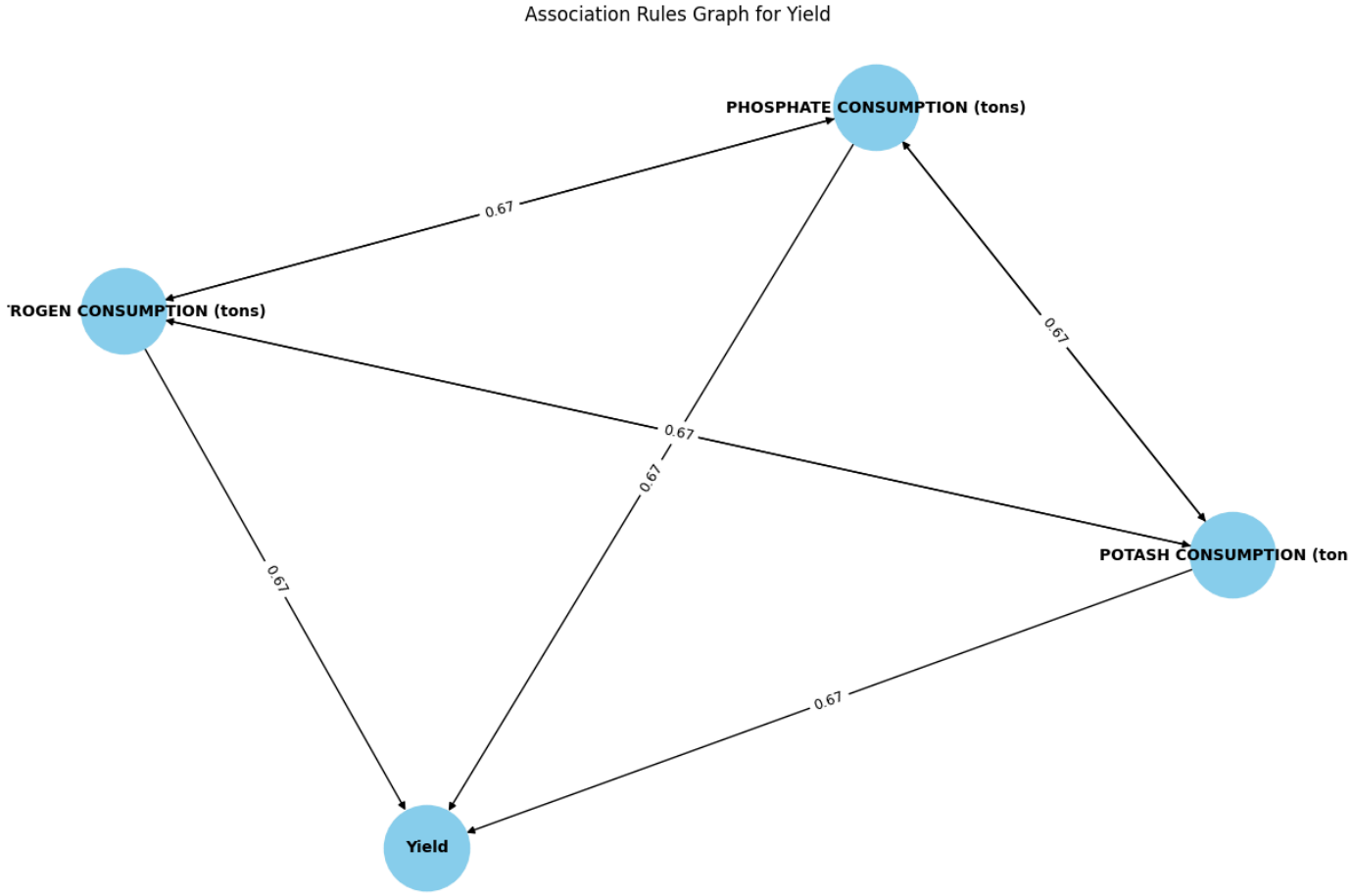
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
```




```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/maharashtra_aggregate_1966_2017.csv'
data = pd.read_csv(file_path)

# Top 10 features of interest
top_10_features = ['Irrigated Area', 'NITROGEN CONSUMPTION (tons)', 'POTASH CONSUMPTION (tons)',
    'PHOSPHATE CONSUMPTION (tons)', 'Max Temp (Centigrate)', 'Annual Rainfall']

# Include 'Yield' and 'Production' as features
selected_features = top_10_features + ['Yield']
filtered_data = data[selected_features]

# Convert continuous data to binary using median threshold
def binary_conversion(df):
    binary_data = df.apply(lambda col: (col > col.median()).astype(int), axis=0)
    return binary_data

# Apply binary conversion
binary_data = binary_conversion(filtered_data)
```

```
# Apply apriori algorithm
frequent_itemsets = apriori(binary_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Apply association rules
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5, num_item
else:
    rules = pd.DataFrame()

# Split rules for 'Yield' and 'Production' separately
# Filter rules for 'RICE YIELD (Kg per ha)' and 'RICE PRODUCTION (1000 tons)'
yield_rules = rules[rules['consequents'].apply(lambda x: any('Yield' in str(item) for item in x))
#production_rules = rules[rules['consequents'].apply(lambda x: any('RICE PRODUCTION (1000 tons)'

# Function to draw association rules graph
def draw_filtered_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()

    # Add edges with weights
    for _, rule in rules_subset.iterrows():
        antecedents = sorted(list(rule['antecedents']))  # Convert set to sorted list
        consequents = sorted(list(rule['consequents']))
        for antecedent in antecedents:
            for consequent in consequents:
                confidence = rule['confidence']
                G.add_edge(antecedent, consequent, weight=confidence)

    # Graph layout
    pos = nx.spring_layout(G, k=1)

    # Draw the graph
    plt.figure(figsize=(12, 8))
    nx.draw(
        G, pos, with_labels=True, node_size=3000, node_color="skyblue",
        font_size=10, font_weight="bold", arrows=True
    )

    # Add edge labels (weights)
    edge_labels = {(antecedent, consequent): f'{weight:.2f}'
                   for antecedent, consequent, weight in G.edges(data='weight')}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

    # Add title and display
    plt.title(title)
    plt.show()

# Display rules and plot graphs
if not yield_rules.empty:
    print("Yield-Related Rules:")
    print(yield_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())
    draw_filtered_rules_graph(yield_rules, title="Association Rules Graph for Yield")
```

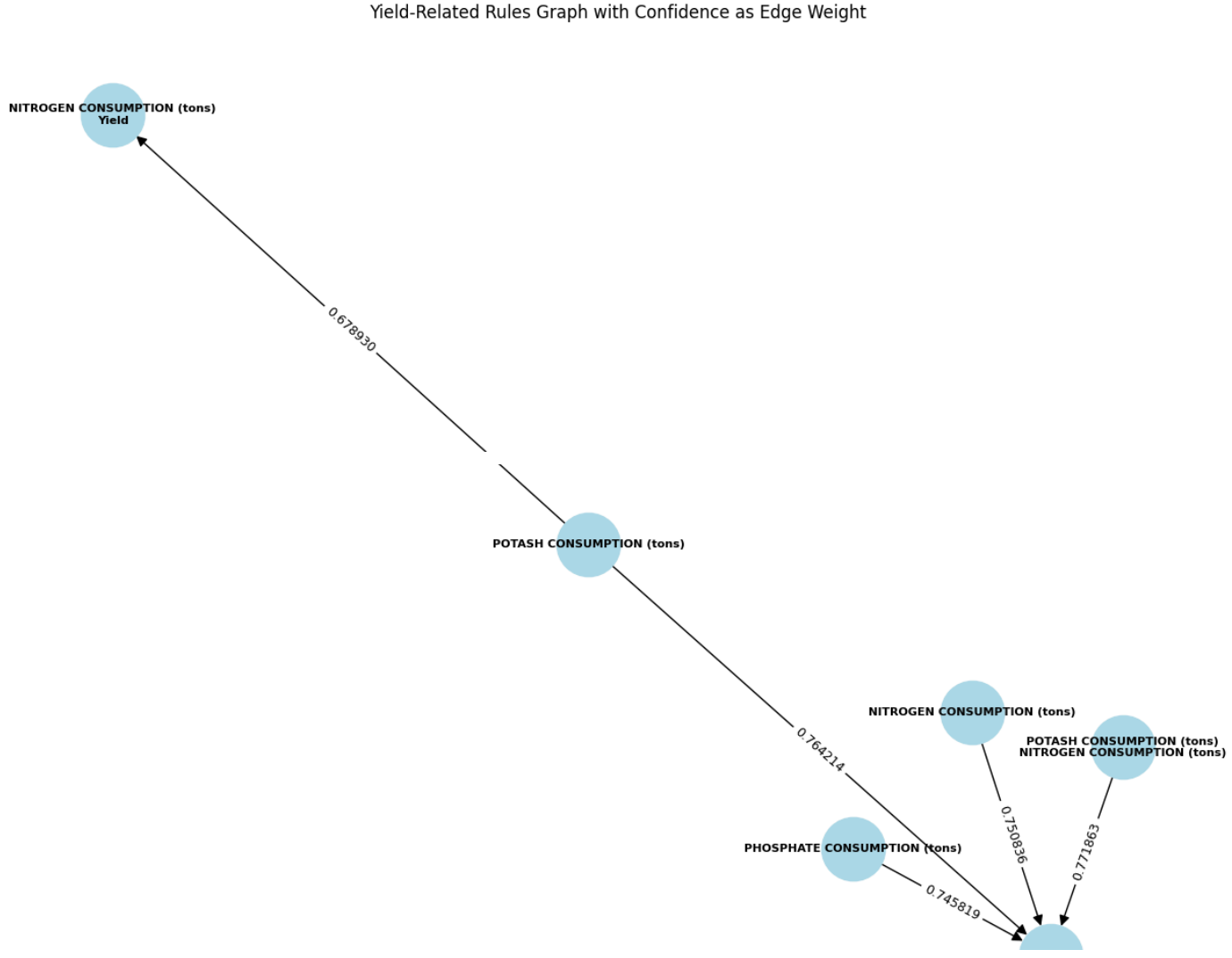
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types
  warnings.warn(
Yield-Related Rules:
                                    antecedents  \
4                    (NITROGEN CONSUMPTION (tons))
8                      (POTASH CONSUMPTION (tons))
10                   (PHOSPHATE CONSUMPTION (tons))
18  (POTASH CONSUMPTION (tons), NITROGEN CONSUMPTI...
21                     (POTASH CONSUMPTION (tons))

                           consequents   support  confidence      lift
4                              (Yield)  0.375418    0.750836  1.501672
8                              (Yield)  0.382107    0.764214  1.528428
10                             (Yield)  0.372910    0.745819  1.491639
18                             (Yield)  0.339465    0.771863  1.543726
21  (NITROGEN CONSUMPTION (tons), Yield)  0.339465    0.678930  1.808463
```




```
import networkx as nx
import matplotlib.pyplot as plt

# Data for yield-related apriori rules based on the table provided
yield_rules = [
    {"antecedents": "NITROGEN CONSUMPTION (tons)", "consequents": "Yield", "confidence": 0.750836
    {"antecedents": "POTASH CONSUMPTION (tons)", "consequents": "Yield", "confidence": 0.764214},
    {"antecedents": "PHOSPHATE CONSUMPTION (tons)", "consequents": "Yield", "confidence": 0.74581
    {"antecedents": "POTASH CONSUMPTION (tons), NITROGEN CONSUMPTION (tons)", "consequents": "Yie
    {"antecedents": "POTASH CONSUMPTION (tons)", "consequents": "NITROGEN CONSUMPTION (tons), Yie
]

# Create directed graph
G = nx.DiGraph()

# Add edges with weights based on confidence
for rule in yield_rules:
```

```
    G.add_edge(rule["antecedents"], rule["consequents"], weight=rule["confidence"])

# Prepare layout and adjust positions for clarity
pos = nx.spring_layout(G, seed=42, k=0.5)  # Increased 'k' value to 1.5 for better spacing

# Update node labels to ensure full text visibility, with multi-attribute nodes stacked
node_labels = {}
for node in G.nodes():
    # Split multi-attribute nodes into separate lines
    node_labels[node] = "\n".join(node.split(", "))

# Draw the graph with customized node sizes and colors
plt.figure(figsize=(12, 10))  # Adjusted size for better visualization
nx.draw(G, pos, with_labels=True, node_size=2000, node_color='lightblue', font_size=8, font_weigh

# Add edge labels to show confidence
edge_labels = {(u, v): f'{d["weight"]:.6f}' for u, v, d in G.edges(data=True)}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

# Display the graph
plt.title("Yield-Related Rules Graph with Confidence as Edge Weight")
plt.show()
```

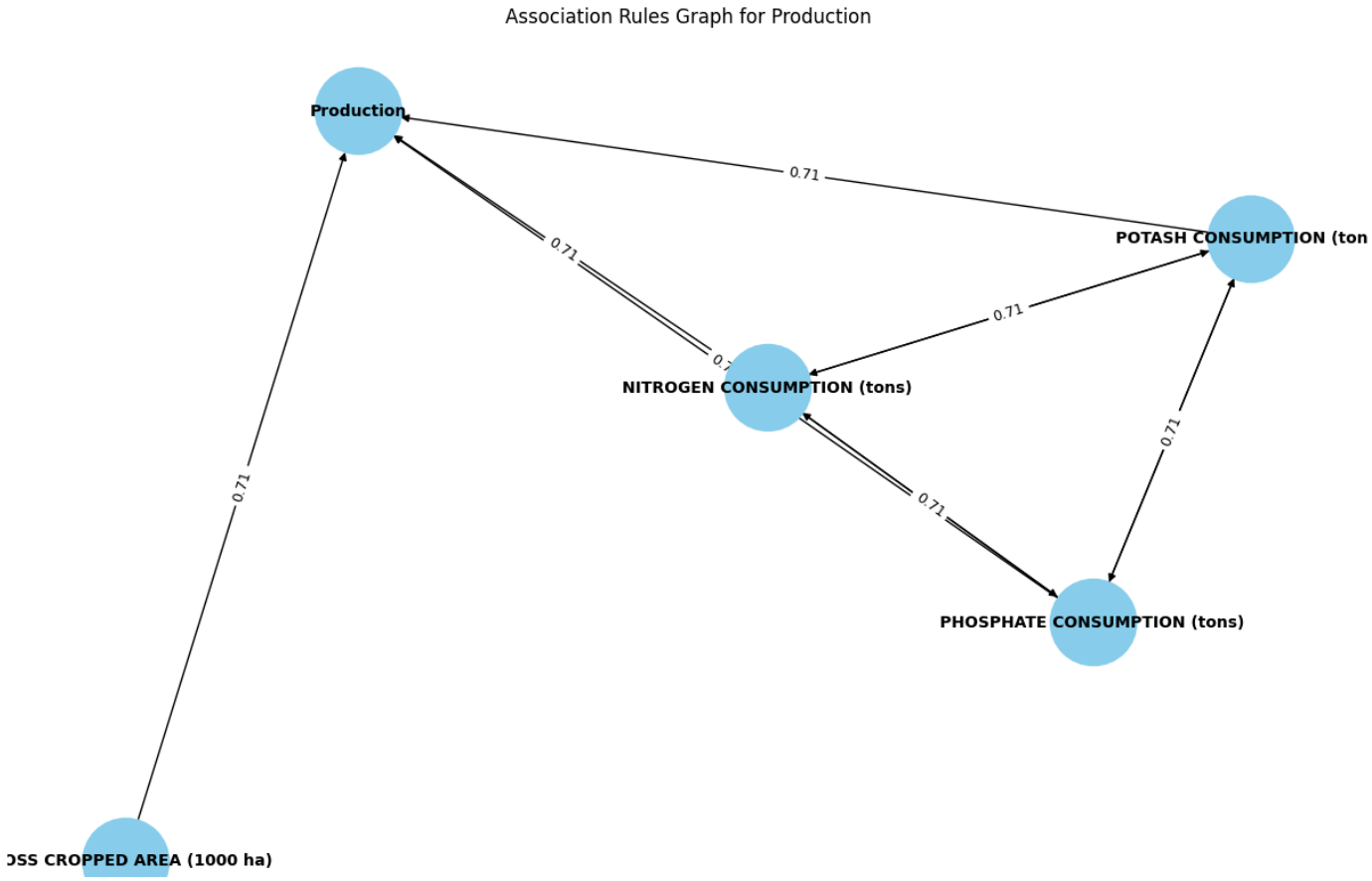
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
```



```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/maharashtra_aggregate_1966_2017.csv'
data = pd.read_csv(file_path)

# Top 10 features of interest
top_10_features = ['GROSS CROPPED AREA (1000 ha)','NITROGEN CONSUMPTION (tons)','Max Temp (Centig
# Include 'Yield' and 'Production' as features
selected_features = top_10_features + ['Production']
filtered_data = data[selected_features]

# Convert continuous data to binary using median threshold
def binary_conversion(df):
    binary_data = df.apply(lambda col: (col > col.median()).astype(int), axis=0)
    return binary_data

# Apply binary conversion
binary_data = binary_conversion(filtered_data)

# Apply apriori algorithm
frequent_itemsets = apriori(binary_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Apply association rules
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5, num_item
else:
    rules = pd.DataFrame()

# Split rules for 'Yield' and 'Production' separately
# Filter rules for 'RICE YIELD (Kg per ha)' and 'RICE PRODUCTION (1000 tons)'
production_rules = rules[rules['consequents'].apply(lambda x: any('Production' in str(item) for i

# Function to draw association rules graph
def draw_filtered_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()

    # Add edges with weights
    for _, rule in rules_subset.iterrows():
        antecedents = sorted(list(rule['antecedents']))  # Convert set to sorted list
        consequents = sorted(list(rule['consequents']))
        for antecedent in antecedents:
            for consequent in consequents:
                confidence = rule['confidence']
                G.add_edge(antecedent, consequent, weight=confidence)

    # Graph layout
    pos = nx.spring_layout(G, k=1)

    # Draw the graph
    plt.figure(figsize=(12, 8))
    nx.draw(
        G, pos, with_labels=True, node_size=3000, node_color="skyblue",
```

```
        font_size=10, font_weight="bold", arrows=True
    )

    # Add edge labels (weights)
    edge_labels = {(antecedent, consequent): f'{weight:.2f}'
                   for antecedent, consequent, weight in G.edges(data='weight')}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

    # Add title and display
    plt.title(title)
    plt.show()

# Display rules and plot graphs
if not production_rules.empty:
    print("\nProduction-Related Rules:")
    print(production_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(
    draw_filtered_rules_graph(production_rules, title="Association Rules Graph for Production")
```

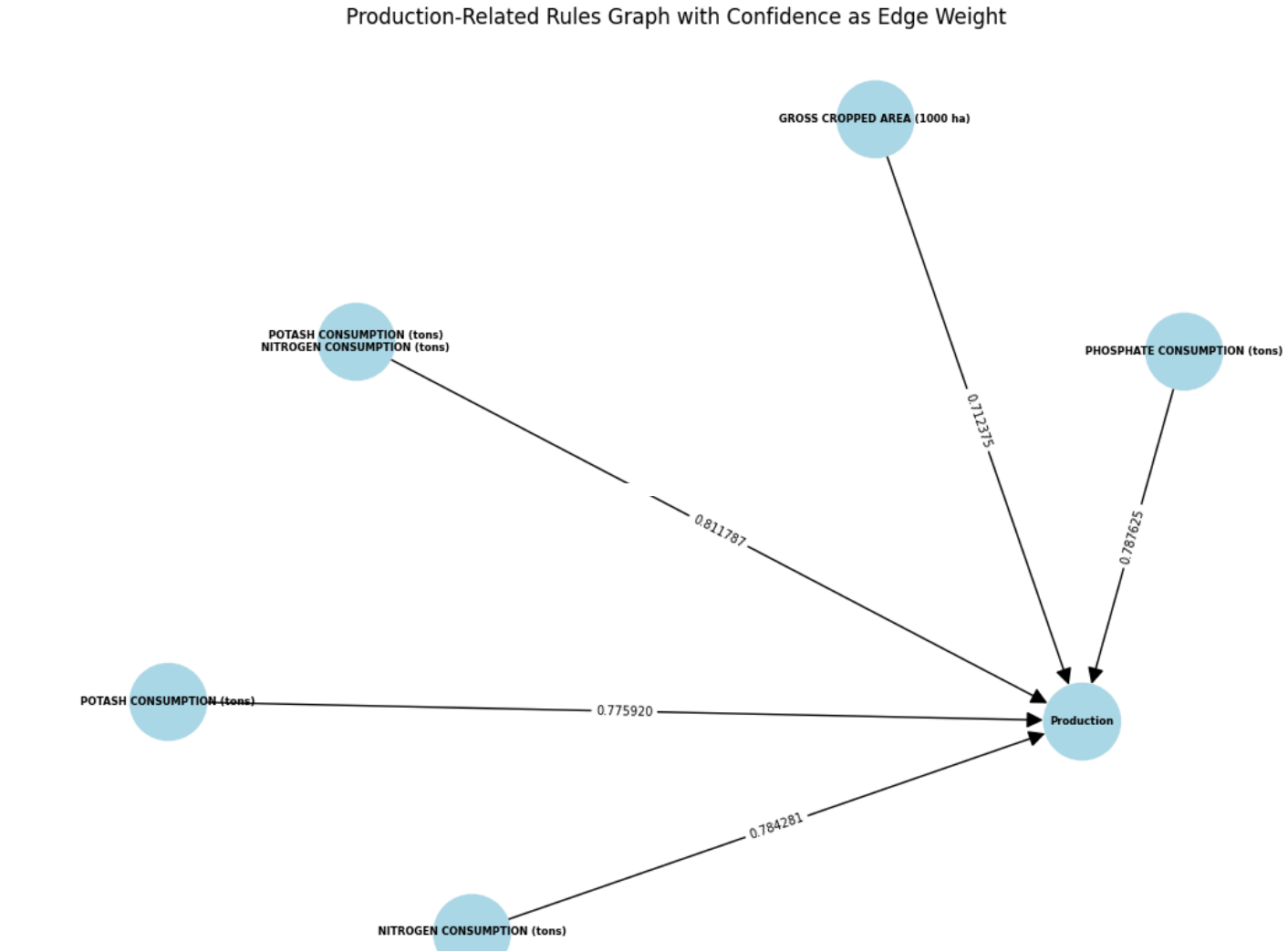
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types
  warnings.warn(

Production-Related Rules:
                                       antecedents   consequents   support  \
9                      (GROSS CROPPED AREA (1000 ha))  (Production)  0.356187
15                      (NITROGEN CONSUMPTION (tons))  (Production)  0.392140
18                        (POTASH CONSUMPTION (tons))  (Production)  0.387960
20                      (PHOSPHATE CONSUMPTION (tons))  (Production)  0.393813
41  (POTASH CONSUMPTION (tons), NITROGEN CONSUMPTI...  (Production)  0.357023

    confidence      lift
9     0.712375  1.424749
15    0.784281  1.568562
18    0.775920  1.551839
20    0.787625  1.575251
41    0.811787  1.623574
```




```
import networkx as nx
import matplotlib.pyplot as plt

# Data for production-related apriori rules based on the provided table
production_rules = [
    {"antecedents": "GROSS CROPPED AREA (1000 ha)", "consequents": "Production", "confidence": 0.
    {"antecedents": "NITROGEN CONSUMPTION (tons)", "consequents": "Production", "confidence": 0.7
    {"antecedents": "POTASH CONSUMPTION (tons)", "consequents": "Production", "confidence": 0.775
    {"antecedents": "PHOSPHATE CONSUMPTION (tons)", "consequents": "Production", "confidence": 0.
    {"antecedents": "POTASH CONSUMPTION (tons), NITROGEN CONSUMPTION (tons)", "consequents": "Pro
]

# Create directed graph
G = nx.DiGraph()

# Add edges with weights based on confidence
for rule in production_rules:
    G.add_edge(rule["antecedents"], rule["consequents"], weight=rule["confidence"])
```

```
# Update node labels to ensure full text visibility, with multi-attribute nodes stacked
node_labels = {}
for node in G.nodes():
    # Split multi-attribute nodes into separate lines
    node_labels[node] = "\n".join(node.split(", "))

# Draw the graph with adjusted layout for more space between nodes
pos = nx.spring_layout(G, seed=42, k=1.5)  # Increase 'k' for more space between nodes
plt.figure(figsize=(10, 8))  # Adjusted size for better visualization
nx.draw(G, pos, with_labels=True, labels=node_labels, node_size=2000, node_color='lightblue', fon

# Add edge labels to show confidence
edge_labels = {(u, v): f'{d["weight"]:.6f}' for u, v, d in G.edges(data=True)}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=7)

# Display the graph
plt.title("Production-Related Rules Graph with Confidence as Edge Weight")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
```

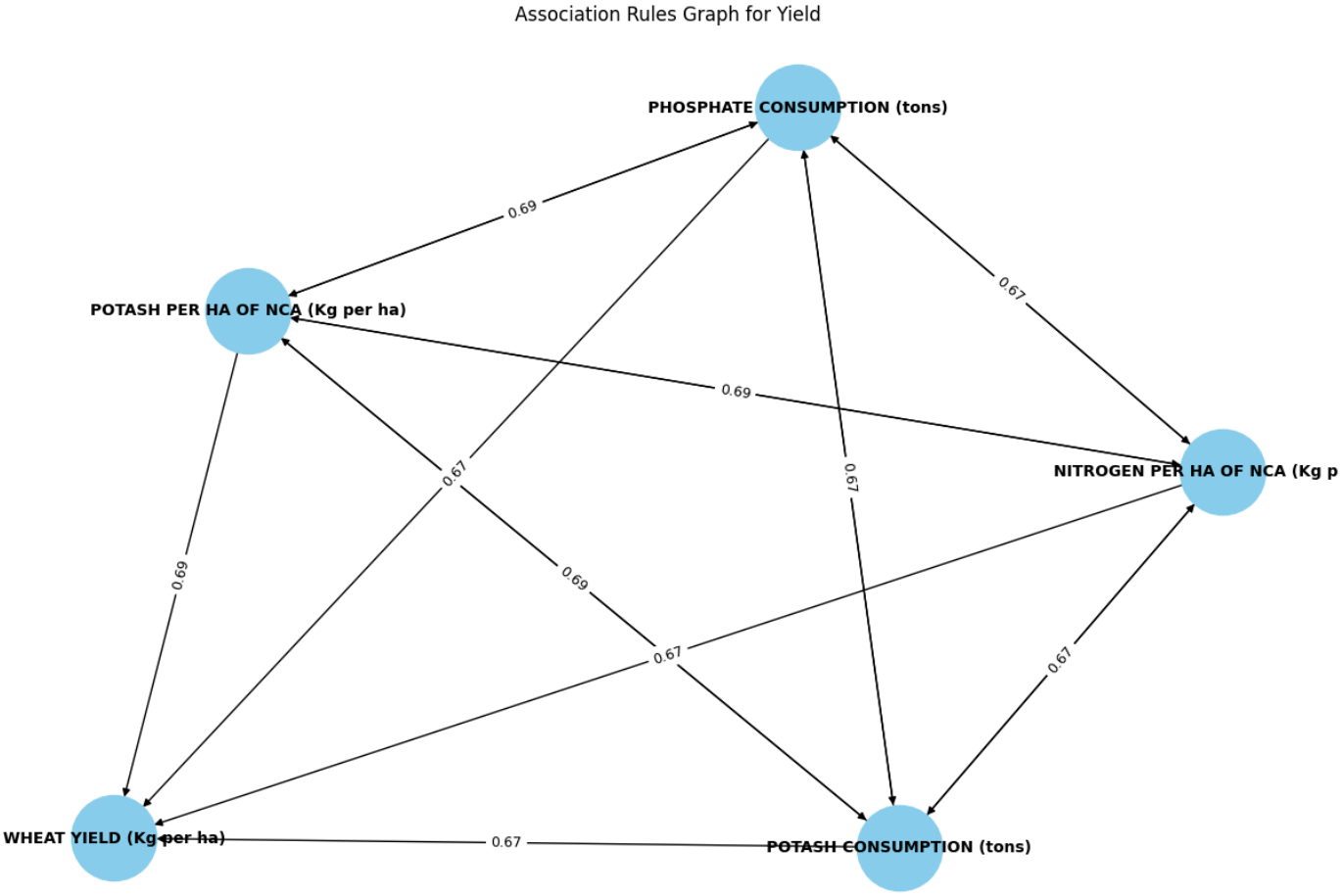



```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx
```

```
# Load the dataset
file_path = '/content/Major_5_Crops_Dataset.csv'
data = pd.read_csv(file_path)

# Top 10 features of interest
top_10_features = ['WHEAT AREA (1000 ha)','NITROGEN PER HA OF NCA (Kg per ha)','PHOSPHATE CONSUMP
,'Irrigated Area']

# Include 'Yield' and 'Production' as features
selected_features = top_10_features + ['WHEAT YIELD (Kg per ha)',]
filtered_data = data[selected_features]

# Convert continuous data to binary using median threshold
def binary_conversion(df):
    binary_data = df.apply(lambda col: (col > col.median()).astype(int), axis=0)
    return binary_data

# Apply binary conversion
binary_data = binary_conversion(filtered_data)

# Apply apriori algorithm
frequent_itemsets = apriori(binary_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Apply association rules
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5, num_item
else:
    rules = pd.DataFrame()

# Split rules for 'Yield' and 'Production' separately
# Filter rules for 'RICE YIELD (Kg per ha)' and 'RICE PRODUCTION (1000 tons)'
yield_rules = rules[rules['consequents'].apply(lambda x: any('WHEAT YIELD (Kg per ha)' in str(ite
#production_rules = rules[rules['consequents'].apply(lambda x: any('RICE PRODUCTION (1000 tons)'

# Function to draw association rules graph
def draw_filtered_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()

    # Add edges with weights
    for _, rule in rules_subset.iterrows():
        antecedents = sorted(list(rule['antecedents']))  # Convert set to sorted list
        consequents = sorted(list(rule['consequents']))
        for antecedent in antecedents:
            for consequent in consequents:
                confidence = rule['confidence']
                G.add_edge(antecedent, consequent, weight=confidence)

    # Graph layout
    pos = nx.spring_layout(G, k=1)

    # Draw the graph
    plt.figure(figsize=(12, 8))
    nx.draw(
        G, pos, with_labels=True, node_size=3000, node_color="skyblue",
        font_size=10, font_weight="bold", arrows=True
```

```
    )

    # Add edge labels (weights)
    edge_labels = {(antecedent, consequent): f'{weight:.2f}'
                   for antecedent, consequent, weight in G.edges(data='weight')}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

    # Add title and display
    plt.title(title)
    plt.show()

# Display rules and plot graphs
if not yield_rules.empty:
    print("Yield-Related Rules:")
    print(yield_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())
    draw_filtered_rules_graph(yield_rules, title="Association Rules Graph for Yield")
```

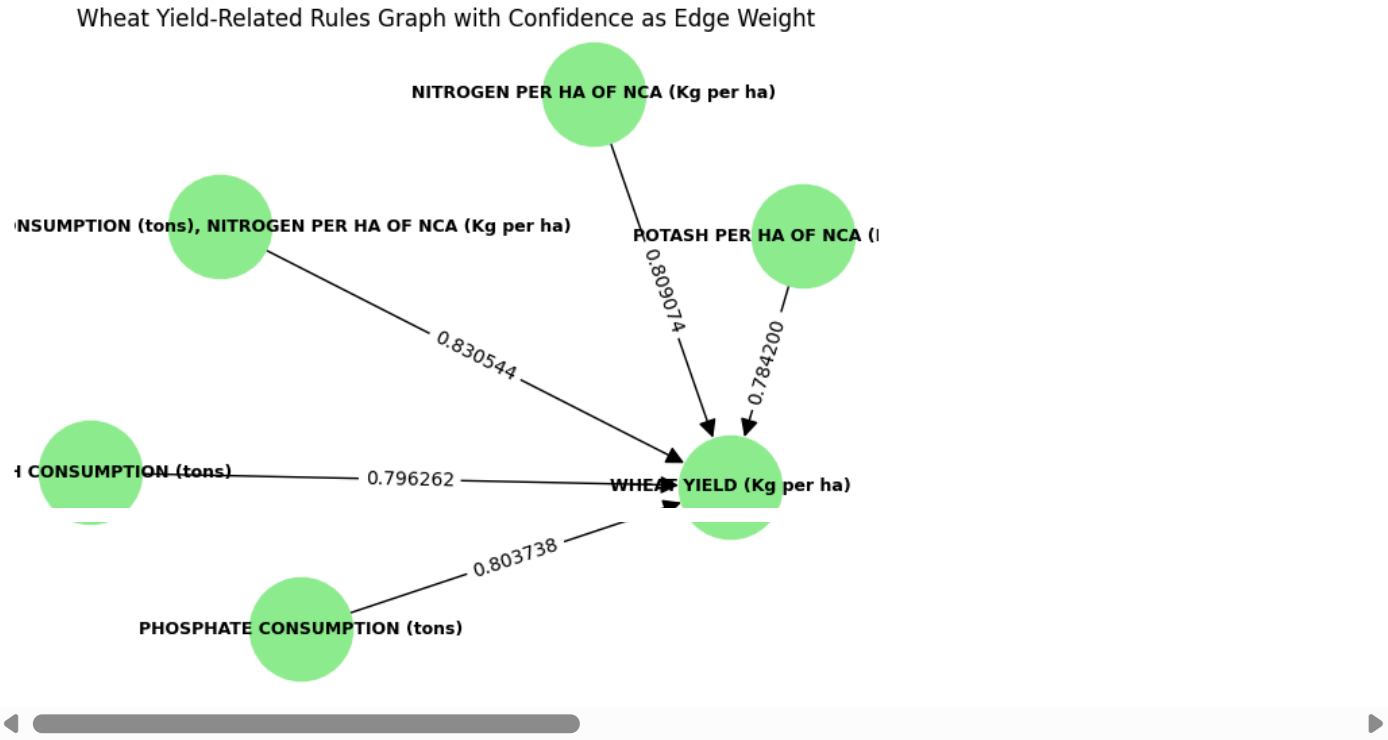
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types
  warnings.warn(
Yield-Related Rules:
                                    antecedents  \
6               (NITROGEN PER HA OF NCA (Kg per ha))
12                   (PHOSPHATE CONSUMPTION (tons))
17                      (POTASH CONSUMPTION (tons))
18                (POTASH PER HA OF NCA (Kg per ha))
32  (PHOSPHATE CONSUMPTION (tons), NITROGEN PER HA...

                 consequents   support  confidence      lift
6   (WHEAT YIELD (Kg per ha))  0.399627    0.809074  1.622693
12  (WHEAT YIELD (Kg per ha))  0.401494    0.803738  1.611992
17  (WHEAT YIELD (Kg per ha))  0.397759    0.796262  1.596997
18  (WHEAT YIELD (Kg per ha))  0.380019    0.784200  1.572806
32  (WHEAT YIELD (Kg per ha))  0.370682    0.830544  1.665754
```




```
import networkx as nx
import matplotlib.pyplot as plt

# Data for yield-related apriori rules
yield_rules = [
    {"antecedents": "NITROGEN PER HA OF NCA (Kg per ha)", "consequents": "WHEAT YIELD (Kg per ha)
    {"antecedents": "PHOSPHATE CONSUMPTION (tons)", "consequents": "WHEAT YIELD (Kg per ha)", "co
    {"antecedents": "POTASH CONSUMPTION (tons)", "consequents": "WHEAT YIELD (Kg per ha)", "confi
    {"antecedents": "POTASH PER HA OF NCA (Kg per ha)", "consequents": "WHEAT YIELD (Kg per ha)",
    {"antecedents": "PHOSPHATE CONSUMPTION (tons), NITROGEN PER HA OF NCA (Kg per ha)", "conseque
]

# Create directed graph
G = nx.DiGraph()

# Add edges with weights based on confidence
for rule in yield_rules:
```

```
    G.add_edge(rule["antecedents"], rule["consequents"], weight=rule["confidence"])

# Draw the graph with adjusted layout for more space between nodes
pos = nx.spring_layout(G, seed=42, k=1.5)  # Increase 'k' for more space between nodes
nx.draw(G, pos, with_labels=True, node_size=3000, node_color='lightgreen', font_size=9, font_weig
nx.draw_networkx_edge_labels(G, pos, edge_labels={(u, v): f'{d["weight"]:.6f}' for u, v, d in G.e

# Display the graph
plt.title("Wheat Yield-Related Rules Graph with Confidence as Edge Weight")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
```



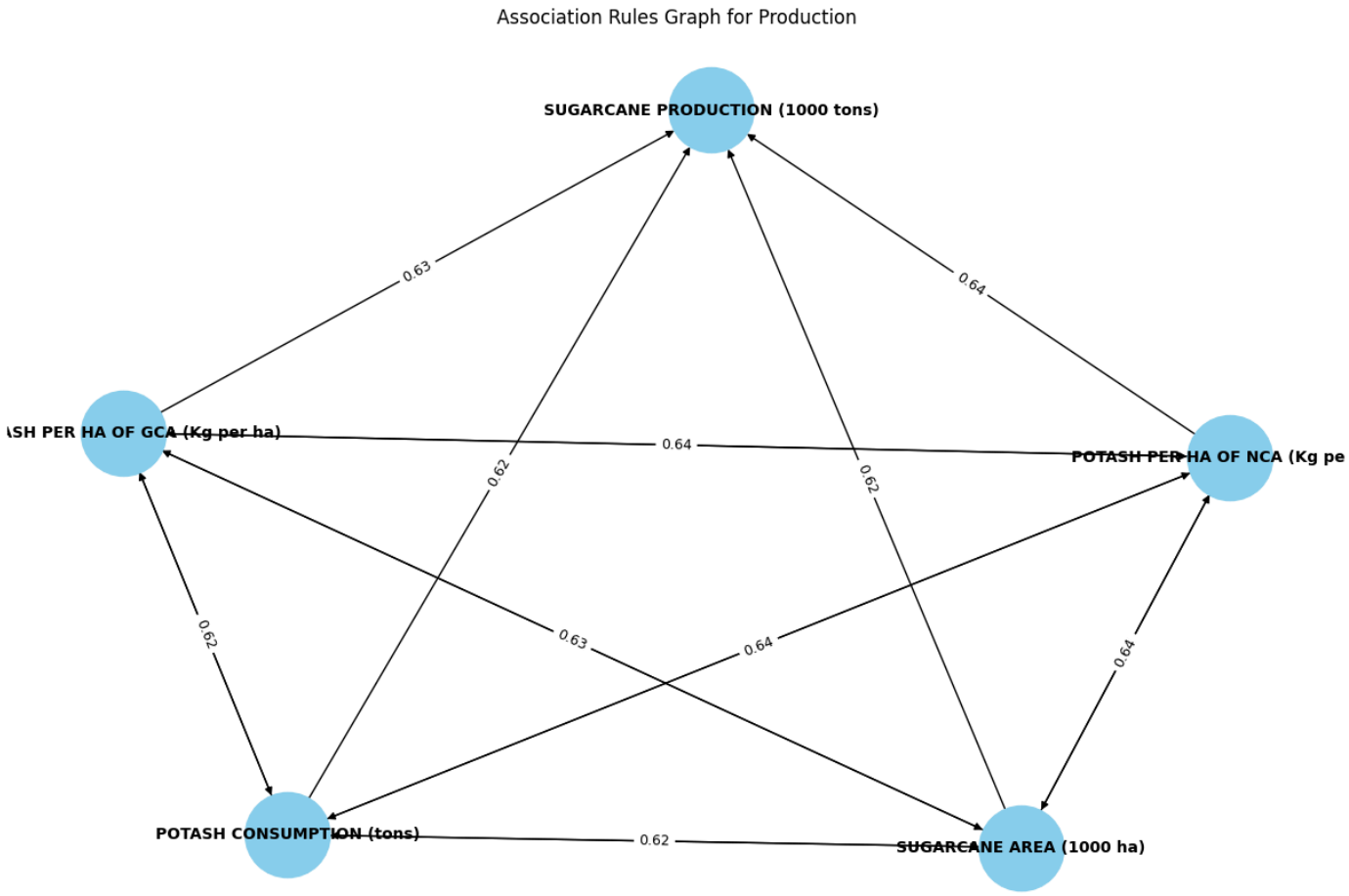

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/Major_5_Crops_Dataset.csv'
data = pd.read_csv(file_path)

# Top 10 features of interest
top_10_features = ['SUGARCANE AREA (1000 ha)','POTASH PER HA OF GCA (Kg per ha)','POTASH CONSUMPT

# Include 'Yield' and 'Production' as features
selected_features = top_10_features + ['SUGARCANE PRODUCTION (1000 tons)',]
filtered_data = data[selected_features]

# Convert continuous data to binary using median threshold
def binary_conversion(df):
    binary_data = df.apply(lambda col: (col > col.median()).astype(int), axis=0)
    return binary_data

# Apply binary conversion
```

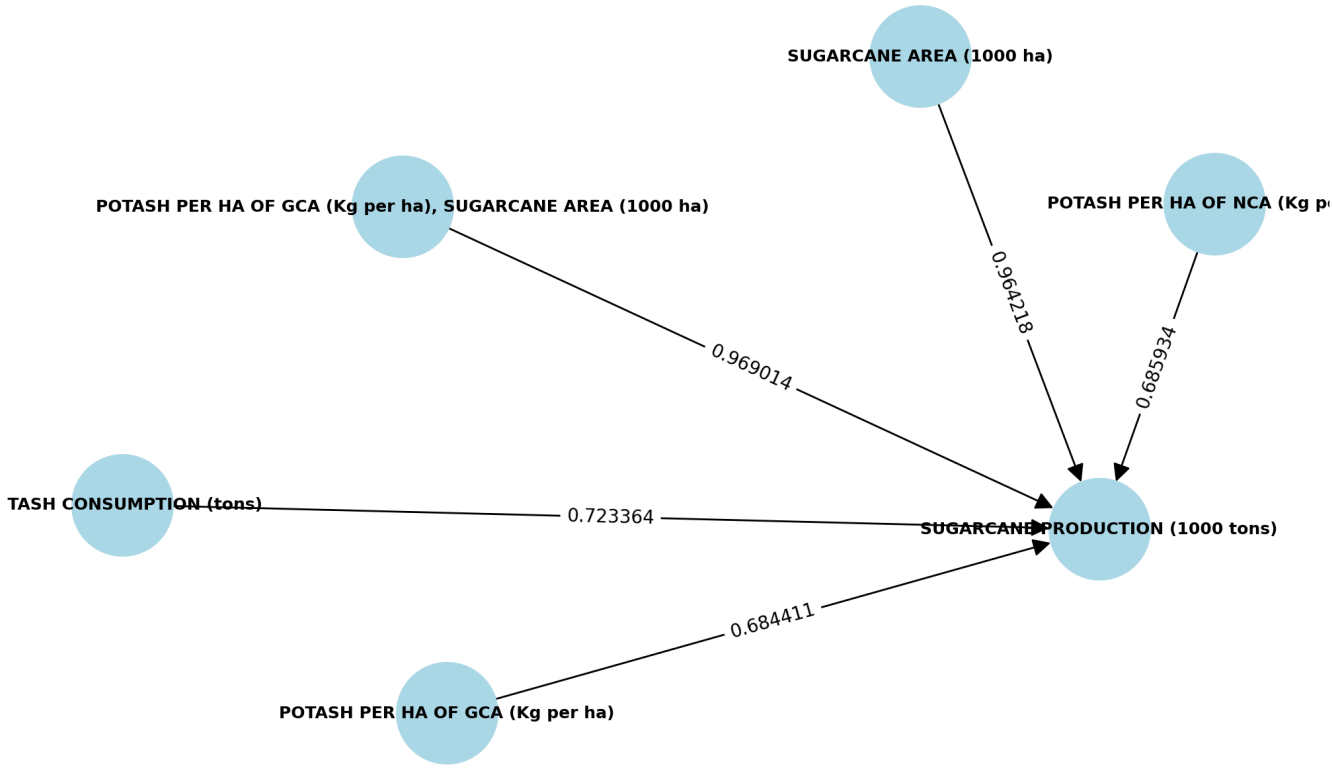
```
binary_data = binary_conversion(filtered_data)

# Apply apriori algorithm
frequent_itemsets = apriori(binary_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Apply association rules
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5, num_item
else:
    rules = pd.DataFrame()

# Split rules for 'Yield' and 'Production' separately
# Filter rules for 'RICE YIELD (Kg per ha)' and 'RICE PRODUCTION (1000 tons)'
production_rules = rules[rules['consequents'].apply(lambda x: any('SUGARCANE PRODUCTION (1000 ton

# Function to draw association rules graph
def draw_filtered_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()

    # Add edges with weights
    for _, rule in rules_subset.iterrows():
        antecedents = sorted(list(rule['antecedents']))  # Convert set to sorted list
        consequents = sorted(list(rule['consequents']))
        for antecedent in antecedents:
            for consequent in consequents:
                confidence = rule['confidence']
                G.add_edge(antecedent, consequent, weight=confidence)

    # Graph layout
    pos = nx.spring_layout(G, k=1)

    # Draw the graph
    plt.figure(figsize=(12, 8))
    nx.draw(
        G, pos, with_labels=True, node_size=3000, node_color="skyblue",
        font_size=10, font_weight="bold", arrows=True
    )

    # Add edge labels (weights)
    edge_labels = {(antecedent, consequent): f'{weight:.2f}'
                   for antecedent, consequent, weight in G.edges(data='weight')}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

    # Add title and display
    plt.title(title)
    plt.show()

# Display rules and plot graphs
if not production_rules.empty:
    print("\nProduction-Related Rules:")
    print(production_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(
    draw_filtered_rules_graph(production_rules, title="Association Rules Graph for Production")
```

/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types
  warnings.warn(

Production-Related Rules:

	antecedents	consequents	support	confidence	lift
7	(SUGARCANE AREA (1000 ha))	(SUGARCANE PRODUCTION (1000 tons))	0.478058	0.964218	1.930239
12	(POTASH PER HA OF GCA (Kg per ha))	(SUGARCANE PRODUCTION (1000 tons))	0.336134	0.684411	1.370101
16	(POTASH CONSUMPTION (tons))	(SUGARCANE PRODUCTION (1000 tons))	0.361345	0.723364	1.448081
19	(POTASH PER HA OF NCA (Kg per ha))	(SUGARCANE PRODUCTION (1000 tons))	0.332400	0.685934	1.373151
33	(POTASH PER HA OF GCA (Kg per ha), SUGARCANE A...	(SUGARCANE PRODUCTION (1000 tons))	0.321195	0.969014	1.939839

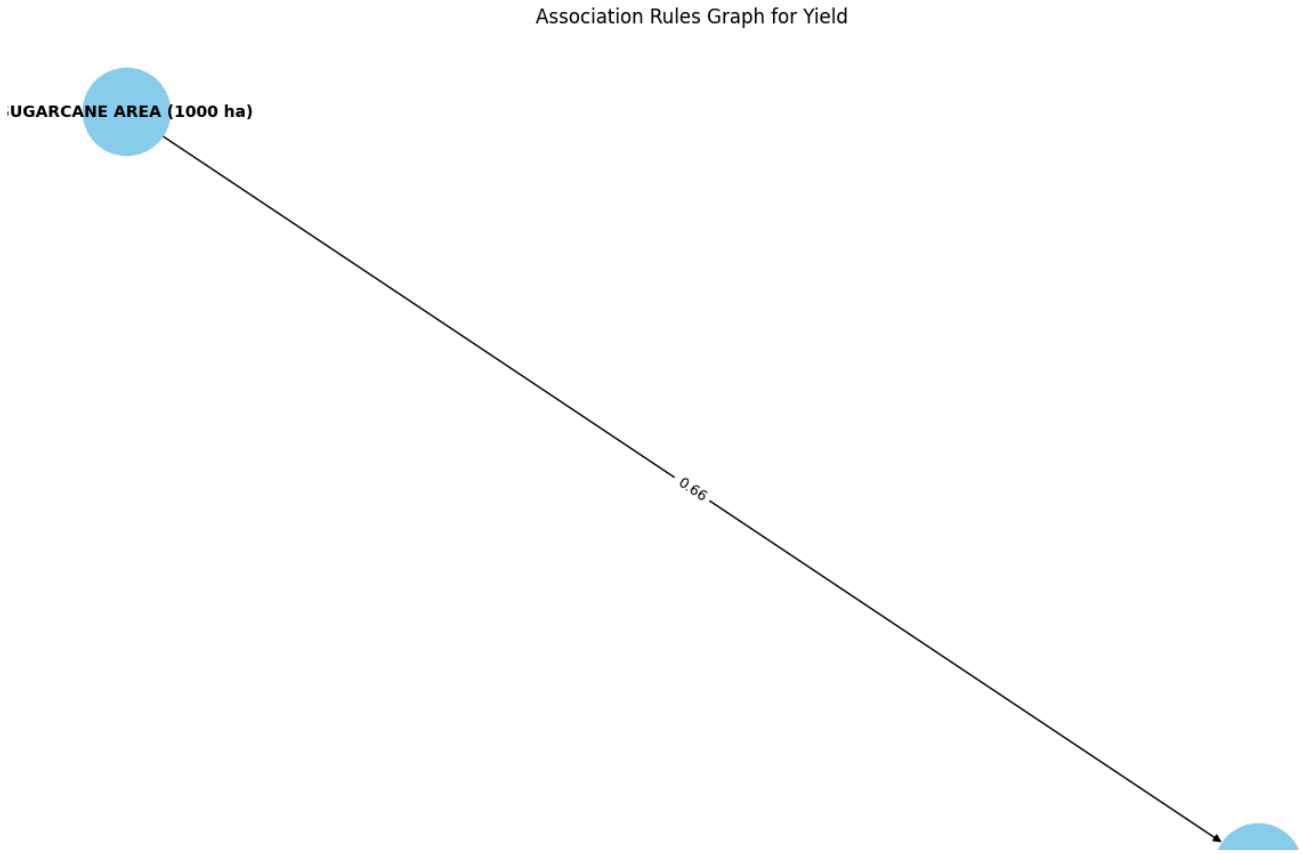





```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/Major_5_Crops_Dataset.csv'
data = pd.read_csv(file_path)

# Top 10 features of interest
top_10_features = ['SUGARCANE AREA (1000 ha)','NITROGEN PER HA OF NCA (Kg per ha)','PHOSPHATE SHA
,'Precipitation (mm)','Irrigated Area']

# Include 'Yield' and 'Production' as features
selected_features = top_10_features + ['SUGARCANE YIELD (Kg per ha)',]
filtered_data = data[selected_features]

# Convert continuous data to binary using median threshold
def binary_conversion(df):
    binary_data = df.apply(lambda col: (col > col.median()).astype(int), axis=0)
    return binary_data

# Apply binary conversion
binary_data = binary_conversion(filtered_data)

# Apply apriori algorithm
frequent_itemsets = apriori(binary_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Apply association rules
```

```
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5, num_item
else:
    rules = pd.DataFrame()

# Split rules for 'Yield' and 'Production' separately
# Filter rules for 'RICE YIELD (Kg per ha)' and 'RICE PRODUCTION (1000 tons)'
yield_rules = rules[rules['consequents'].apply(lambda x: any('SUGARCANE YIELD (Kg per ha)' in str
#production_rules = rules[rules['consequents'].apply(lambda x: any('RICE PRODUCTION (1000 tons)'

# Function to draw association rules graph
def draw_filtered_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()

    # Add edges with weights
    for _, rule in rules_subset.iterrows():
        antecedents = sorted(list(rule['antecedents']))  # Convert set to sorted list
        consequents = sorted(list(rule['consequents']))
        for antecedent in antecedents:
            for consequent in consequents:
                confidence = rule['confidence']
                G.add_edge(antecedent, consequent, weight=confidence)

    # Graph layout
    pos = nx.spring_layout(G, k=1)

    # Draw the graph
    plt.figure(figsize=(12, 8))
    nx.draw(
        G, pos, with_labels=True, node_size=3000, node_color="skyblue",
        font_size=10, font_weight="bold", arrows=True
    )

    # Add edge labels (weights)
    edge_labels = {(antecedent, consequent): f'{weight:.2f}'
                   for antecedent, consequent, weight in G.edges(data='weight')}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

    # Add title and display
    plt.title(title)
    plt.show()

# Display rules and plot graphs
if not yield_rules.empty:
    print("Yield-Related Rules:")
    print(yield_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())
    draw_filtered_rules_graph(yield_rules, title="Association Rules Graph for Yield")
```

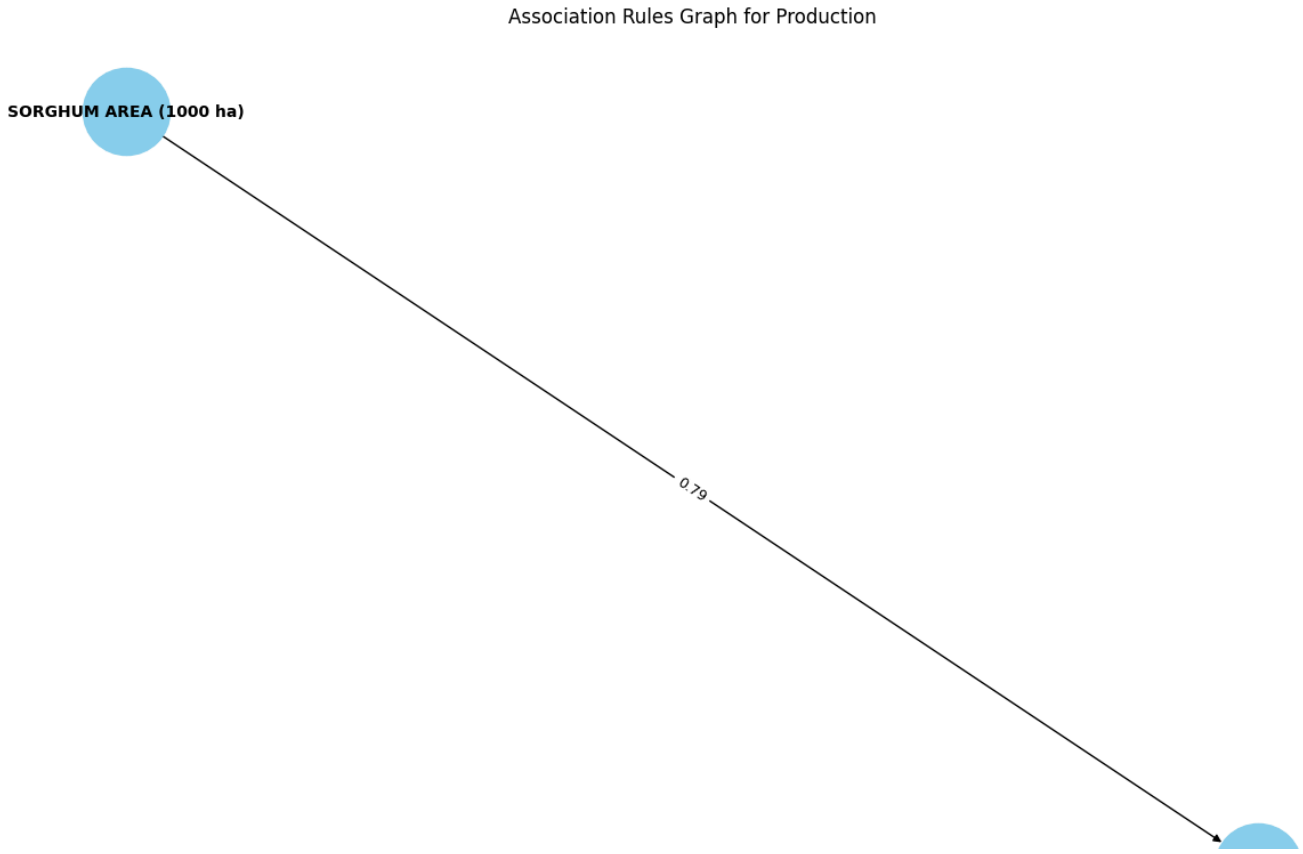
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types
  warnings.warn(
Yield-Related Rules:
                  antecedents                   consequents   support  \
5  (SUGARCANE AREA (1000 ha))  (SUGARCANE YIELD (Kg per ha))  0.328665

   confidence      lift
5      0.6629  1.327039
```




```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/Major_5_Crops_Dataset.csv'
data = pd.read_csv(file_path)

# Top 10 features of interest
top_10_features = ['SORGHUM AREA (1000 ha)','POTASH PER HA OF GCA (Kg per ha)','PHOSPHATE CONSUMP

# Include 'Yield' and 'Production' as features
selected_features = top_10_features + ['SORGHUM PRODUCTION (1000 tons)',]
filtered_data = data[selected_features]

# Convert continuous data to binary using median threshold
def binary_conversion(df):
    binary_data = df.apply(lambda col: (col > col.median()).astype(int), axis=0)
    return binary_data

# Apply binary conversion
binary_data = binary_conversion(filtered_data)

# Apply apriori algorithm
```

```
frequent_itemsets = apriori(binary_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Apply association rules
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5, num_item
else:
    rules = pd.DataFrame()

# Split rules for 'Yield' and 'Production' separately
# Filter rules for 'RICE YIELD (Kg per ha)' and 'RICE PRODUCTION (1000 tons)'
production_rules = rules[rules['consequents'].apply(lambda x: any('SORGHUM PRODUCTION (1000 tons)

# Function to draw association rules graph
def draw_filtered_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()

    # Add edges with weights
    for _, rule in rules_subset.iterrows():
        antecedents = sorted(list(rule['antecedents']))  # Convert set to sorted list
        consequents = sorted(list(rule['consequents']))
        for antecedent in antecedents:
            for consequent in consequents:
                confidence = rule['confidence']
                G.add_edge(antecedent, consequent, weight=confidence)

    # Graph layout
    pos = nx.spring_layout(G, k=1)

    # Draw the graph
    plt.figure(figsize=(12, 8))
    nx.draw(
        G, pos, with_labels=True, node_size=3000, node_color="skyblue",
        font_size=10, font_weight="bold", arrows=True
    )

    # Add edge labels (weights)
    edge_labels = {(antecedent, consequent): f'{weight:.2f}'
                   for antecedent, consequent, weight in G.edges(data='weight')}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

    # Add title and display
    plt.title(title)
    plt.show()

# Display rules and plot graphs
if not production_rules.empty:
    print("\nProduction-Related Rules:")
    print(production_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(
    draw_filtered_rules_graph(production_rules, title="Association Rules Graph for Production")
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cel
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types
  warnings.warn(

Production-Related Rules:
                antecedents                      consequents   support  \
2  (SORGHUM AREA (1000 ha))  (SORGHUM PRODUCTION (1000 tons))  0.394958

   confidence     lift
2    0.792135  1.58872
```




```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import networkx as nx

# Load the dataset
file_path = '/content/Major_5_Crops_Dataset.csv'
data = pd.read_csv(file_path)

# Top 10 features of interest
top_10_features = ['SORGHUM AREA (1000 ha)','NITROGEN PER HA OF NCA (Kg per ha)','PHOSPHATE CONSU
,'Precipitation (mm)','Irrigated Area','NITROGEN PER HA OF GCA (Kg per ha)']

# Include 'Yield' and 'Production' as features
selected_features = top_10_features + ['SORGHUM YIELD (Kg per ha)',]
filtered_data = data[selected_features]

# Convert continuous data to binary using median threshold
def binary_conversion(df):
    binary_data = df.apply(lambda col: (col > col.median()).astype(int), axis=0)
    return binary_data

# Apply binary conversion
binary_data = binary_conversion(filtered_data)
```

```
# Apply apriori algorithm
frequent_itemsets = apriori(binary_data, min_support=0.3, use_colnames=True)

num_itemsets = frequent_itemsets['itemsets'].apply(len).value_counts().to_dict()

# Apply association rules
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5, num_item
else:
    rules = pd.DataFrame()

# Split rules for 'Yield' and 'Production' separately
# Filter rules for 'RICE YIELD (Kg per ha)' and 'RICE PRODUCTION (1000 tons)'
yield_rules = rules[rules['consequents'].apply(lambda x: any('SORGHUM YIELD (Kg per ha)' in str(i
#production_rules = rules[rules['consequents'].apply(lambda x: any('RICE PRODUCTION (1000 tons)'

# Function to draw association rules graph
def draw_filtered_rules_graph(rules_subset, title="Association Rules Graph"):
    G = nx.DiGraph()

    # Add edges with weights
    for _, rule in rules_subset.iterrows():
        antecedents = sorted(list(rule['antecedents']))  # Convert set to sorted list
        consequents = sorted(list(rule['consequents']))
        for antecedent in antecedents:
            for consequent in consequents:
                confidence = rule['confidence']
                G.add_edge(antecedent, consequent, weight=confidence)

    # Graph layout
    pos = nx.spring_layout(G, k=1)

    # Draw the graph
    plt.figure(figsize=(12, 8))
    nx.draw(
        G, pos, with_labels=True, node_size=3000, node_color="skyblue",
        font_size=10, font_weight="bold", arrows=True
    )

    # Add edge labels (weights)
    edge_labels = {(antecedent, consequent): f'{weight:.2f}'
                   for antecedent, consequent, weight in G.edges(data='weight')}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

    # Add title and display
    plt.title(title)
    plt.show()

# Display rules and plot graphs
```