Analysis and Systems of Big Data Assignment Report

Ajay Narayanan (COE16B044)

1 Part A

Descriptive and Predictive Analysis on plants dataset. The dataset is a copy of the “US States Plants dataset for Connecticut" from Plants Databse. https://plants.sc.egov.usda.gov/java/stateDownload?statefips=US09

[1]:
```
# Read the dataset
import pandas as pd
import numpy as np
df = pd.read_csv("plants.csv",)
```

1.1 Descriptive Analytics

[2]:
```
# Display few records from the dataset and summarize the details about the␣
↪dataset contents
print("Number of records in the dataset = ", len(df), end ="\n\n")
```

```
Number of records in the dataset =  12461
```

[3]:
```
print("===================== Sample records ===================== ")
print(df.head(), end ="\n\n")
```

```
===================== Sample records =====================
   Symbol Synonym Symbol                           Scientific Name with Author  \
0  ACARO2            NaN                                Acarospora A. Massal.
1    ACER            NaN                                              Acer L.
2    ACGI            NaN                                 Acer ginnala Maxim.
3    ACGI          ACTAG  Acer tataricum L. ssp. ginnala (Maxim.) Wesmael
4   ACNE2            NaN                                     Acer negundo L.

  National Common Name           Family
0       cracked lichen  Acarosporaceae
1                maple       Aceraceae
2           Amur maple       Aceraceae
```

```
3                  NaN       Aceraceae
4             boxelder       Aceraceae
```

[4]:
```
print("====================== Statistics of data ====================== ")
print(df.describe())
```

```
====================== Statistics of data ======================
       Symbol Synonym Symbol  \
count   12461           7926
unique   4535           7926
top      MIGU         CAPA22
freq       70              1

                              Scientific Name with Author  \
count                                               12461
unique                                              12459
top     Sarracenia purpurea L. ssp. purpurea var. purp...
freq                                                    2

       National Common Name      Family
count                  4526       12461
unique                 3587         193
top          hybrid violet  Asteraceae
freq                     14        1592
```

[5]:
```
print("====================== Dataset Information ====================== ")
print(df.info())
```

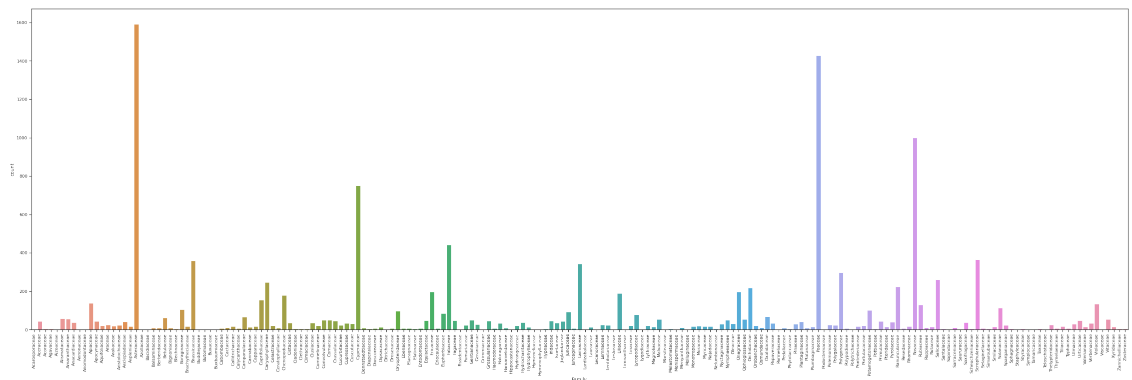
```
====================== Dataset Information ======================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12461 entries, 0 to 12460
Data columns (total 5 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Symbol                       12461 non-null  object
 1   Synonym Symbol               7926 non-null   object
 2   Scientific Name with Author  12461 non-null  object
 3   National Common Name         4526 non-null   object
 4   Family                       12461 non-null  object
dtypes: object(5)
memory usage: 486.9+ KB
None
```

[6]:
```
print("====================== Total number of NAN values per column␣
 ↪====================== ")
print(df.isnull().sum(axis = 0))
```

```
===================== Total number of NAN values per column
=====================
Symbol                          0
Synonym Symbol               4535
Scientific Name with Author     0
National Common Name         7935
Family                          0
dtype: int64
```

[7]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[8]:
```python
# Frequency vs Family plot
plt.figure(figsize=(50,15))
sns.set(style="ticks", color_codes=True)
sns.countplot(df['Family'])
plt.xticks(rotation = 90);
```



[9]:
```python
df['Family'].value_counts()
```

[9]:
```
Asteraceae          1592
Poaceae             1428
Rosaceae            1000
Cyperaceae           751
Fabaceae             441
                    ...
Fissidentaceae         1
Teloschistaceae        1
Hymeneliaceae          1
Acarosporaceae         1
Bacidiaceae            1
Name: Family, Length: 193, dtype: int64
```

[10]:
```
df['Synonym Symbol'].value_counts()
```

[10]:
```
CAPA22    1
APMEL2    1
POMO8     1
ALPLA     1
RUWI2     1
         ..
IPPUD     1
SAPEA2    1
RUCL3     1
ARAT5     1
TAGL2     1
Name: Synonym Symbol, Length: 7926, dtype: int64
```

1.2 Data Preprocessing

We can see that the columns, 'Synonym Symbol' and 'National Common Name' has significant amount of 'NAN' values. These 'NAN' values will be a hindrance while doing analysis on the dataset. There are many methods to handle the missing/NAN values. We adopt the simplest among them - replacing the 'NAN' values with the mode of the column.

[11]:
```
print("====================== Replacing nan with mode  ====================== ")
# Mode computation ignores NAN
for column in df:
    df[column] = df[column].fillna(df[column].mode()[0])

print("====================== Statistics of data ====================== ")
print(df.describe())

print("====================== Total number of NAN values per column␣
 ↪====================== ")
print(df.isnull().sum(axis = 0))

df.to_csv("plants_preprocessed.csv", encoding='utf-8', index=False)
```

```
====================== Replacing nan with mode  ======================
====================== Statistics of data ======================
       Symbol Synonym Symbol  \
count   12461          12461
unique   4535           7926
top      MIGU          ABAB3
freq       70           4536

                              Scientific Name with Author  \
count                                               12461
unique                                              12459
```

```
top     Sarracenia purpurea L. ssp. purpurea var. purp...
freq                                                   2

       National Common Name      Family
count                 12461       12461
unique                 3587         193
top           hybrid violet  Asteraceae
freq                   7949        1592
====================== Total number of NAN values per column
======================
Symbol                          0
Synonym Symbol                  0
Scientific Name with Author     0
National Common Name            0
Family                          0
dtype: int64
```

1.3 Predictive Analytics

Here we try to predict the family given Symbol, Synonym Symbol and National Common Name

[12]:
```python
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
import pandas as pd
import numpy as np
```

[13]:
```python
# Read the preprocessed dataset
df = pd.read_csv("plants_preprocessed.csv")
X_labels = [ 'Symbol', 'Synonym Symbol', 'National Common Name']
Y_labels = ['Family']

print("====================== Dataset before encoding ======================")
print(df[X_labels + Y_labels].head())
# Convert categorical data to integers using a labelencoder
for label in X_labels + Y_labels:
    df[label] = LabelEncoder().fit_transform(df[label])

print("\n\n====================== Dataset after encoding ======================")
print(df[X_labels + Y_labels].head())
```

```
====================== Dataset before encoding ======================
   Symbol Synonym Symbol National Common Name          Family
0  ACARO2          ABAB3       cracked lichen  Acarosporaceae
1    ACER          ABAB3                maple       Aceraceae
2    ACGI          ABAB3           Amur maple       Aceraceae
3    ACGI          ACTAG        hybrid violet       Aceraceae
```

```
4   ACNE2         ABAB3            boxelder       Aceraceae


===================== Dataset after encoding ======================
   Symbol  Synonym Symbol  National Common Name  Family
0      11               0                  1167       0
1      13               0                  2083       1
2      14               0                    71       1
3      14              61                  1852       1
4      21               0                   817       1
```

[14]:
```
# Split the dataset into test and trainig set
X_train, X_test, y_train, y_test = train_test_split(df[X_labels], df[Y_labels],␣
 ↪test_size =0.2)
```

[15]:
```
# Fit a Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train.values.ravel()).predict(X_test)
acc_nb = sum(y_test.values.ravel() == y_pred)/len(y_pred)
print("Accuracy of Gaussian Naive Bayes classifier = ", acc_nb)
```

```
Accuracy of Gaussian Naive Bayes classifier =  0.14640994785399117
```

[16]:
```
# Fit a Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
clf = clf.fit(X_train, y_train.values.ravel())
y_pred = clf.predict(X_test)
acc_rfc = sum(y_test.values.ravel() == y_pred)/len(y_pred)
print("Accuracy of Random Forest classifier = ", acc_rfc)
```

```
Accuracy of Random Forest classifier =  0.6550340954673085
```

[17]:
```
# Fit a Decision Tree Classifier
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc_tc = sum(y_test.values.ravel() == y_pred)/len(y_pred)
print("Accuracy of Decision Tree classifier = ", acc_tc)
```

```
Accuracy of Decision Tree classifier =  0.7557160048134778
```

[18]:
```
# Fit Extremely randomized tree classifier
from sklearn.tree import ExtraTreeClassifier
clf = ExtraTreeClassifier()
clf = clf.fit(X_train, y_train.values.ravel())
```

```
y_pred = clf.predict(X_test)
acc_etc = sum(y_test.values.ravel() == y_pred)/len(y_pred)
print("Accuracy of Extremely randomized tree classifier. = ", acc_etc)
```

```
Accuracy of Extremely randomized tree classifier. =  0.6209386281588448
```

[19]:
```
#Best Model

results = pd.DataFrame({
'Model': ['ExtraTreeClassifier',
'Decision Tree Classifier',
'Random Forest Classifier',
'Gaussian Naive Bayes',
],
'Score': [acc_etc,acc_tc, acc_rfc, acc_nb]})
result_df = results.sort_values(by='Score', ascending=False)
print(result_df)
```

```
                      Model     Score
1  Decision Tree Classifier  0.755716
2  Random Forest Classifier  0.655034
0       ExtraTreeClassifier  0.620939
3      Gaussian Naive Bayes  0.146410
```

For predicting family of a plant (using Symbol, Synonym Symbol and National Common Name), Decision Tree classifier performs the best

2 Part B

The various metrics for evaluating association rules are as follows Given a rule 'A -> C', A stands for antecedent and C stands for consequent.

1. Support
   $\text{support}(A \rightarrow C) = \text{P}(A \cup C), \;\; \text{support}(A) = \text{P}(A)$
   Support(A->C) is the probability of transactions that contain both itemsets A and C together in the database. Also, Support(A) = P(A) where A is an itemset.

2. Confidence
   $\text{confidence}(A \rightarrow C) = \frac{\text{support}(A \rightarrow C)}{\text{support}(A)}$
   It is the probability of transactions that contain itemsets C given that itemset A occurs in the transcation.

3. Lift
   $\text{lift}(A \rightarrow C) = \frac{\text{confidence}(A \rightarrow C)}{\text{support}(C)}$

The lift metric is commonly used to measure how much more often the antecedent and consequent of a rule A->C occur together than we would expect if they were statistically independent. If A and C are independent, the Lift score will be exactly 1.

4. Leverage

$$\text{Levarage}(A \to C) = \text{support}(A \to C) - \text{support}(A) \times \text{support}(C)$$

Leverage computes the difference between the observed frequency of A and C appearing together and the frequency that would be expected if A and C were independent. An leverage value of 0 indicates independence.

5. Conviction
$\text{conviction}(A \to C) = \frac{1 - \text{support}(C)}{1 - \text{confidence}(A \to C)}$

A high conviction value means that the consequent is highly depending on the antecedent. For instance, in the case of a perfect confidence score, the denominator becomes 0 (due to 1 - 1) for which the conviction score is defined as 'inf'. Similar to lift, if items are independent, the conviction is 1.

2.1 Association rule mining (ARM)

ARM involves two steps,

1. Finding itemsets
2. Generating rules based on itemsets

The itemsets are further classfied into

- FIs(Frequent Itemsets)
- MFIs(Maximal Frequent Itemsets)
- CFIs(Closed Frequent Itemsets)

In this section, we perform mining of all the above mentioned itemsets and show rules for few algorithms alone.

[20]:
```
# Import the necessary packages
from mlxtend.frequent_patterns import apriori, association_rules, fpmax
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.preprocessing import TransactionEncoder
from tabulate import tabulate

# One-hot encoding of the preprocessed plants dataset
df = pd.read_csv("plants_preprocessed.csv")
encoder = TransactionEncoder()
encoded_values = encoder.fit(df.values.tolist()).transform(df.values.tolist())
df_onehot = pd.DataFrame(encoded_values, columns=encoder.columns_)
```

2.1.1 Frequent Itemset Mining

For frequent itemset mining, we use two algorithms

- Apriori
- FP Growth

[21]:
```
# Frequent Itemset Mining
print("=============== Apriori ===============")
```

```python
itemsets = apriori(df_onehot,min_support=0.009,use_colnames=True)
print(itemsets)
print("\n\n RULES based on Apriori : \n\n")
association_rules(itemsets, metric="confidence", min_threshold=0.7)
```

```
================ Apriori ================
     support                           itemsets
0   0.364016                            (ABAB3)
1   0.011075                         (Apiaceae)
2   0.127759                       (Asteraceae)
3   0.028810                     (Brassicaceae)
4   0.012359                   (Caprifoliaceae)
5   0.019822                  (Caryophyllaceae)
6   0.014285                   (Chenopodiaceae)
7   0.060268                       (Cyperaceae)
8   0.015890                        (Ericaceae)
9   0.035390                         (Fabaceae)
10  0.027446                        (Lamiaceae)
11  0.015167                        (Liliaceae)
12  0.015809                       (Onagraceae)
13  0.017495                      (Orchidaceae)
14  0.114598                          (Poaceae)
15  0.023915                     (Polygonaceae)
16  0.017976                    (Ranunculaceae)
17  0.080250                         (Rosaceae)
18  0.010352                        (Rubiaceae)
19  0.020865                       (Salicaceae)
20  0.029372                 (Scrophulariaceae)
21  0.009068                       (Solanaceae)
22  0.010673                        (Violaceae)
23  0.637910                    (hybrid violet)
24  0.043255                (Asteraceae, ABAB3)
25  0.010834              (Brassicaceae, ABAB3)
26  0.025038                (Cyperaceae, ABAB3)
27  0.014124                  (Fabaceae, ABAB3)
28  0.012519                 (Lamiaceae, ABAB3)
29  0.032983                   (Poaceae, ABAB3)
30  0.021427                  (Rosaceae, ABAB3)
31  0.010673          (Scrophulariaceae, ABAB3)
32  0.084504        (hybrid violet, Asteraceae)
33  0.017976      (hybrid violet, Brassicaceae)
34  0.012599   (hybrid violet, Caryophyllaceae)
35  0.009791    (hybrid violet, Chenopodiaceae)
36  0.035230        (hybrid violet, Cyperaceae)
37  0.011476         (hybrid violet, Ericaceae)
38  0.021266          (hybrid violet, Fabaceae)
39  0.015007         (hybrid violet, Lamiaceae)
```

```
40  0.012198        (hybrid violet, Onagraceae)
41  0.010754       (hybrid violet, Orchidaceae)
42  0.081615           (Poaceae, hybrid violet)
43  0.018377      (hybrid violet, Polygonaceae)
44  0.011476     (hybrid violet, Ranunculaceae)
45  0.058904          (hybrid violet, Rosaceae)
46  0.017414        (Salicaceae, hybrid violet)
47  0.018698  (hybrid violet, Scrophulariaceae)


 RULES based on Apriori :
```

[21]:

```
      antecedents      consequents  antecedent support  consequent support  \
0     (Ericaceae)  (hybrid violet)            0.015890             0.63791
1    (Onagraceae)  (hybrid violet)            0.015809             0.63791
2       (Poaceae)  (hybrid violet)            0.114598             0.63791
3  (Polygonaceae)  (hybrid violet)            0.023915             0.63791
4      (Rosaceae)  (hybrid violet)            0.080250             0.63791
5    (Salicaceae)  (hybrid violet)            0.020865             0.63791

    support  confidence      lift  leverage  conviction
0  0.011476    0.722222  1.132169  0.001340    1.303523
1  0.012198    0.771574  1.209533  0.002113    1.585148
2  0.081615    0.712185  1.116434  0.008512    1.258064
3  0.018377    0.768456  1.204646  0.003122    1.563808
4  0.058904    0.734000  1.150632  0.007711    1.361240
5  0.017414    0.834615  1.308359  0.004104    2.189380
```

[22]:

```python
# Frequent Itemset Mining
print("================ FP Growth ================")
itemsets = fpgrowth(df_onehot,min_support=0.009,use_colnames=True)
print(itemsets)
print("\n\n RULES based on FP Growth : \n\n")
print(association_rules(itemsets, metric="confidence", min_threshold=0.7))
```

```
================ FP Growth ================
     support                           itemsets
0   0.364016                            (ABAB3)
1   0.637910                    (hybrid violet)
2   0.011075                         (Apiaceae)
3   0.127759                       (Asteraceae)
4   0.028810                     (Brassicaceae)
5   0.012359                   (Caprifoliaceae)
6   0.019822                  (Caryophyllaceae)
7   0.014285                   (Chenopodiaceae)
8   0.060268                       (Cyperaceae)
```

```
9   0.015890                       (Ericaceae)
10  0.035390                        (Fabaceae)
11  0.027446                       (Lamiaceae)
12  0.015167                       (Liliaceae)
13  0.015809                      (Onagraceae)
14  0.017495                     (Orchidaceae)
15  0.114598                         (Poaceae)
16  0.023915                    (Polygonaceae)
17  0.017976                   (Ranunculaceae)
18  0.080250                        (Rosaceae)
19  0.010352                       (Rubiaceae)
20  0.020865                      (Salicaceae)
21  0.029372                (Scrophulariaceae)
22  0.009068                      (Solanaceae)
23  0.010673                       (Violaceae)
24  0.043255               (Asteraceae, ABAB3)
25  0.084504       (hybrid violet, Asteraceae)
26  0.010834             (Brassicaceae, ABAB3)
27  0.017976     (hybrid violet, Brassicaceae)
28  0.012599  (hybrid violet, Caryophyllaceae)
29  0.009791   (hybrid violet, Chenopodiaceae)
30  0.025038               (Cyperaceae, ABAB3)
31  0.035230       (hybrid violet, Cyperaceae)
32  0.011476        (hybrid violet, Ericaceae)
33  0.014124                 (Fabaceae, ABAB3)
34  0.021266         (hybrid violet, Fabaceae)
35  0.012519                (Lamiaceae, ABAB3)
36  0.015007        (hybrid violet, Lamiaceae)
37  0.012198       (hybrid violet, Onagraceae)
38  0.010754      (hybrid violet, Orchidaceae)
39  0.032983                  (Poaceae, ABAB3)
40  0.081615          (Poaceae, hybrid violet)
41  0.018377     (hybrid violet, Polygonaceae)
42  0.011476    (hybrid violet, Ranunculaceae)
43  0.021427                 (Rosaceae, ABAB3)
44  0.058904         (hybrid violet, Rosaceae)
45  0.017414       (Salicaceae, hybrid violet)
46  0.010673         (Scrophulariaceae, ABAB3)
47  0.018698 (hybrid violet, Scrophulariaceae)


 RULES based on FP Growth :


     antecedents      consequents  antecedent support  consequent support  \
0    (Ericaceae)  (hybrid violet)            0.015890             0.63791
1   (Onagraceae)  (hybrid violet)            0.015809             0.63791
2      (Poaceae)  (hybrid violet)            0.114598             0.63791
```

```
3  (Polygonaceae)  (hybrid violet)           0.023915              0.63791
4      (Rosaceae)  (hybrid violet)           0.080250              0.63791
5    (Salicaceae)  (hybrid violet)           0.020865              0.63791

    support  confidence      lift  leverage  conviction
0  0.011476    0.722222  1.132169  0.001340    1.303523
1  0.012198    0.771574  1.209533  0.002113    1.585148
2  0.081615    0.712185  1.116434  0.008512    1.258064
3  0.018377    0.768456  1.204646  0.003122    1.563808
4  0.058904    0.734000  1.150632  0.007711    1.361240
5  0.017414    0.834615  1.308359  0.004104    2.189380
```

2.1.2 Closed Itemset Mining

For closed itemset mining, we use the following two algorithms

- ECLAT-Close
- Apriori-Close

[23]:

```
# Closed Itemset Mining
from fim import eclat, apriori

# df = pd.read_csv("plants_preprocessed.csv")
dataset = df.values.tolist()

print("================  ECLAT - CLOSE ================ ")
itemsets = eclat(dataset,target='c',supp=2,report='s')
print(tabulate(itemsets,  headers=['Itemset', 'Support'], tablefmt='pretty'))


print("\n\n\n================  Apriori - CLOSE  ================ ")
itemsets = apriori(dataset,target='c',supp=2,report='s')
print(tabulate(itemsets,  headers=['Itemset', 'Support'], tablefmt='pretty'))
```

```
================  ECLAT - CLOSE ================
+---------------------------------+----------------------+
|             Itemset             |       Support        |
+---------------------------------+----------------------+
|          ('Salicaceae',)        | 0.02086509910922077  |
|         ('Polygonaceae',)       | 0.023914613594414574 |
|          ('Lamiaceae',)         | 0.02744563036674424  |
|         ('Brassicaceae',)       | 0.028809886846962524 |
|       ('Scrophulariaceae',)     | 0.029371639515287696 |
|   ('Fabaceae', 'hybrid violet') | 0.02126635101516732  |
|           ('Fabaceae',)         | 0.035390418104485996 |
|      ('Cyperaceae', 'ABAB3')    | 0.02503811893106492  |
|  ('Cyperaceae', 'hybrid violet') | 0.03522991734210738  |
|          ('Cyperaceae',)        | 0.060268036273172294 |
```

('Rosaceae', 'ABAB3')	0.021426851777545945
('Rosaceae', 'hybrid violet')	0.05890377979295402
('Rosaceae',)	0.08025038118931065
('Poaceae', 'ABAB3')	0.032982906668806676
('Poaceae', 'hybrid violet')	0.08161463766952894
('Poaceae',)	0.1145975443383356
('Asteraceae', 'ABAB3')	0.04325495546103844
('Asteraceae', 'hybrid violet')	0.08450365139234411
('Asteraceae',)	0.12775860685338256
('ABAB3',)	0.3640157290747131
('hybrid violet',)	0.6379102800738303

=============== Apriori - CLOSE ===============

Itemset	Support
('Salicaceae',)	0.02086509910922077
('Polygonaceae',)	0.023914613594414574
('Lamiaceae',)	0.02744563036674424
('Brassicaceae',)	0.028809886846962524
('Scrophulariaceae',)	0.029371639515287696
('Fabaceae',)	0.035390418104485996
('Fabaceae', 'hybrid violet')	0.02126635101516732
('Cyperaceae',)	0.060268036273172294
('Cyperaceae', 'ABAB3')	0.02503811893106492
('Cyperaceae', 'hybrid violet')	0.03522991734210738
('Rosaceae',)	0.08025038118931065
('Rosaceae', 'ABAB3')	0.021426851777545945
('Rosaceae', 'hybrid violet')	0.05890377979295402
('Poaceae',)	0.1145975443383356
('Poaceae', 'ABAB3')	0.032982906668806676
('Poaceae', 'hybrid violet')	0.08161463766952894
('Asteraceae',)	0.12775860685338256
('Asteraceae', 'ABAB3')	0.04325495546103844
('Asteraceae', 'hybrid violet')	0.08450365139234411
('ABAB3',)	0.3640157290747131
('hybrid violet',)	0.6379102800738303

2.1.3 Maximal Itemset Mining

For Maximal itemset mining, we use the following two algorithms

- FP growth - MFI
- ECLAT - MFI

[24]:
```python
# Maximal Frequent Itemset Mining

print("================ FP growth - MFI ==================")
itemsets = fpmax(df_onehot, min_support=0.001, use_colnames=True,max_len = 10)
print(itemsets)

print("\n\n RULES based on FP growth : \n\n")
rules = association_rules(itemsets, min_threshold=0.0001,support_only=True)
print(rules[['antecedents', 'consequents', 'support']])
```

```
================ FP growth - MFI ==================
      support             itemsets
0    0.001043  (Asteraceae, EUGRG)
1    0.001043    (Rosaceae, RUFR4)
2    0.001043    (Rosaceae, RUAL9)
3    0.001043     (Poaceae, SCSCS)
4    0.001043    (ROCAC, Rosaceae)
..        ...                  ...
223  0.014124    (Fabaceae, ABAB3)
224  0.025038  (Cyperaceae, ABAB3)
225  0.021427    (Rosaceae, ABAB3)
226  0.032983     (Poaceae, ABAB3)
227  0.043255  (Asteraceae, ABAB3)

[228 rows x 2 columns]


 RULES based on FP growth :


      antecedents   consequents   support
0    (Asteraceae)       (EUGRG)  0.001043
1         (EUGRG)  (Asteraceae)  0.001043
2      (Rosaceae)       (RUFR4)  0.001043
3         (RUFR4)    (Rosaceae)  0.001043
4      (Rosaceae)       (RUAL9)  0.001043
..            ...           ...       ...
717       (ABAB3)    (Rosaceae)  0.021427
718     (Poaceae)       (ABAB3)  0.032983
719       (ABAB3)     (Poaceae)  0.032983
720  (Asteraceae)       (ABAB3)  0.043255
721       (ABAB3)  (Asteraceae)  0.043255

[722 rows x 3 columns]
```

[25]:
```python
# Maximal Frequent Itemset Mining
```

```
print("================ ECLAT-MFI ================ ")
itemsets = eclat(dataset,target='m',supp=2,report='s')
print(tabulate(itemsets,  headers=['Itemset', 'Support'], tablefmt='pretty'))
```

```
================ ECLAT-MFI ================
+--------------------------------+----------------------+
|            Itemset             |       Support        |
+--------------------------------+----------------------+
|         ('Salicaceae',)        | 0.02086509910922077  |
|        ('Polygonaceae',)       | 0.023914613594414574 |
|         ('Lamiaceae',)         | 0.0274456303667442 4 |
|        ('Brassicaceae',)       | 0.028809886846962524 |
|      ('Scrophulariaceae',)     | 0.029371639515287696 |
|  ('Fabaceae', 'hybrid violet') | 0.0212663510151673 2 |
|     ('Cyperaceae', 'ABAB3')    | 0.02503811893106492  |
| ('Cyperaceae', 'hybrid violet') | 0.03522991734210738  |
|      ('Rosaceae', 'ABAB3')     | 0.021426851777545945 |
|  ('Rosaceae', 'hybrid violet') | 0.05890377979295402  |
|      ('Poaceae', 'ABAB3')      | 0.032982906668806676 |
|  ('Poaceae', 'hybrid violet')  | 0.08161463766952894  |
|     ('Asteraceae', 'ABAB3')    | 0.04325495546103844  |
| ('Asteraceae', 'hybrid violet') | 0.08450365139234411  |
+--------------------------------+----------------------+
```

3 Part C

In this part, we test drive Decision tree classifier and Naive Bayes on IRIS dataset.

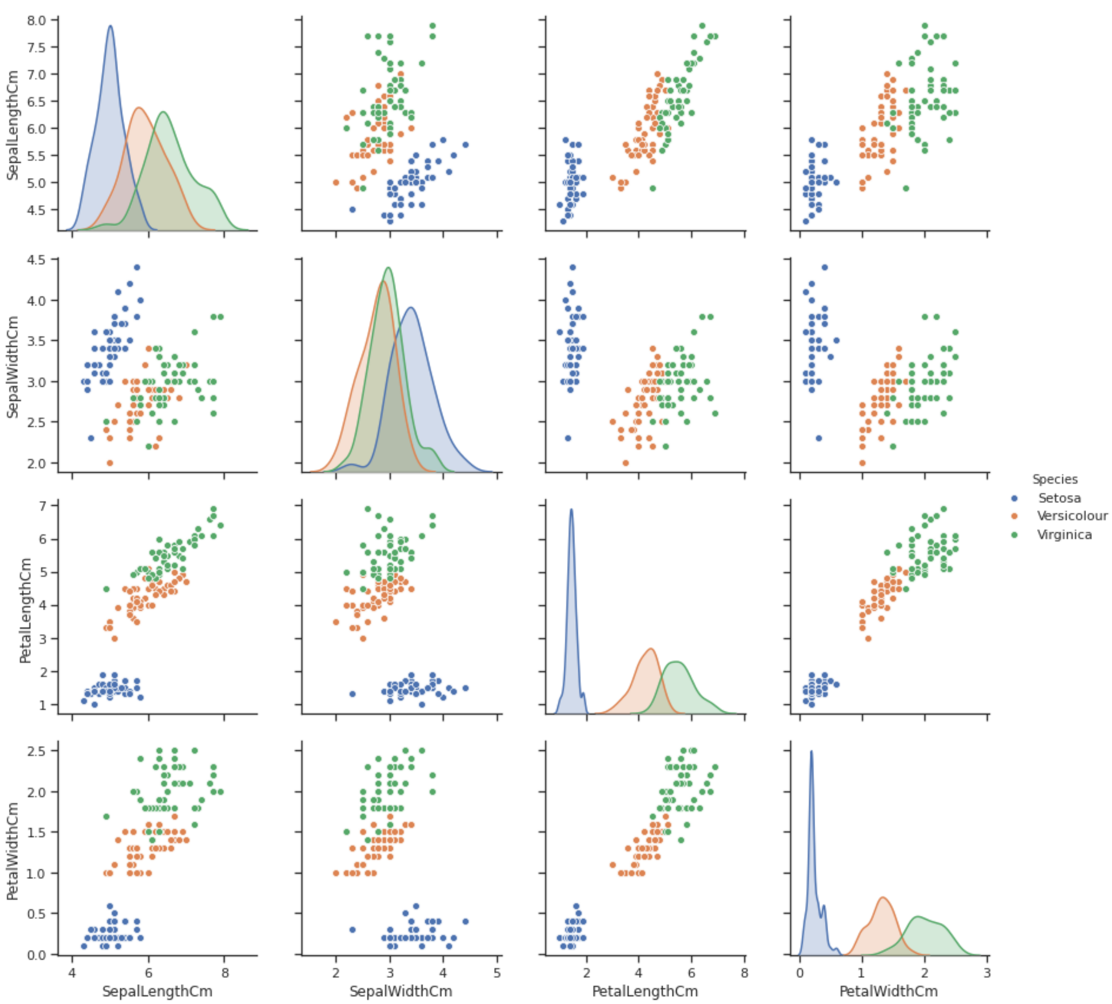
[26]:
```
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
import graphviz
```

[27]:
```
# Load IRIS dataset
iris = load_iris();
print(iris['DESCR'])
# Convert the dictionary to pandas dataframe
df = pd.DataFrame(iris['data'])
df.columns = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']
# Replace integer coded values to their corresponding names
df['Species'] = iris['target']
df['Species'].replace([0, 1, 2], ['Setosa', 'Versicolour', 'Virginica'],␣
↪inplace=True)
df['target'] = iris['target']
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References
```

```
- Fisher, R.A. "The use of multiple measurements in taxonomic problems"
  Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
  Mathematical Statistics" (John Wiley, NY, 1950).
- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
  (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
  Structure and Classification Rule for Recognition in Partially Exposed
  Environments".  IEEE Transactions on Pattern Analysis and Machine
  Intelligence, Vol. PAMI-2, No. 1, 67-71.
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
  on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
  conceptual clustering system finds 3 classes in the data.
- Many, many more ...
```

[28]:
```python
# Data visualization
sns.pairplot(df.drop("target", axis=1), hue="Species", height=3)
plt.show()
```



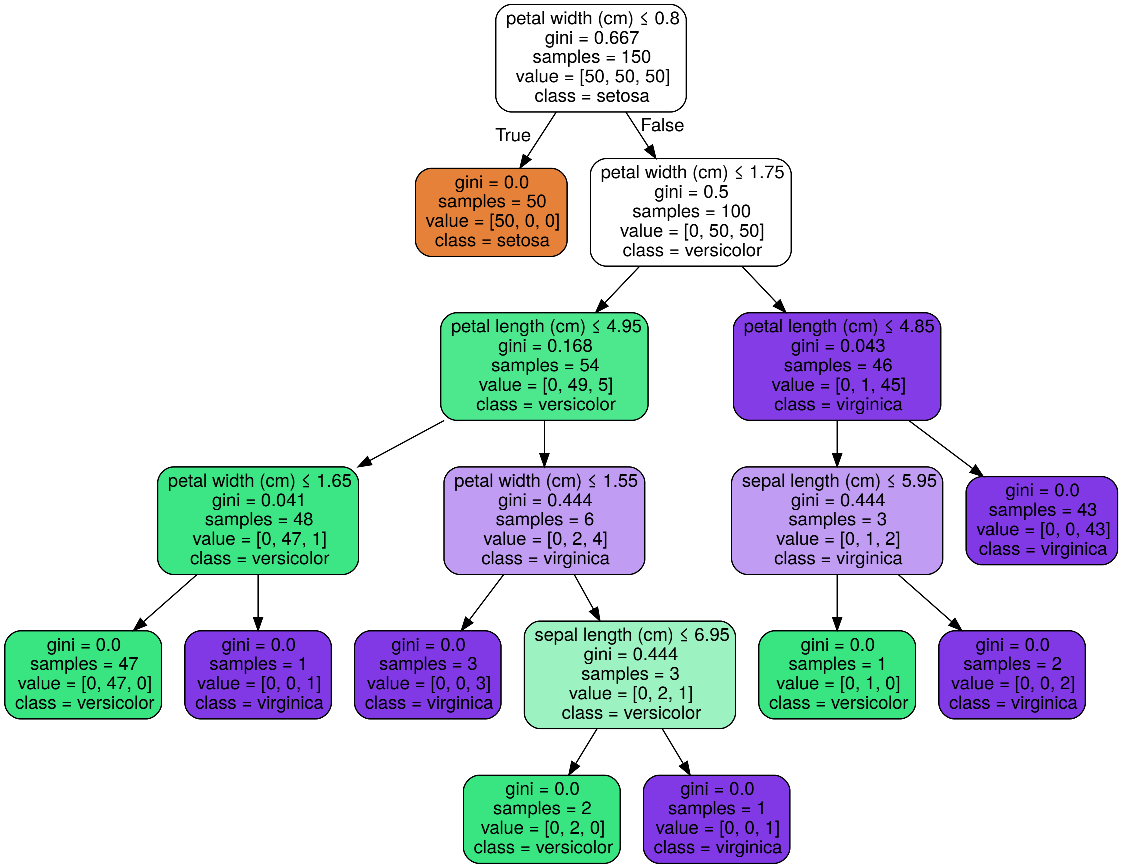
Decision Tree Classifier

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

[29]:
```
# Decision Tree Classifier
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.3)
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)
y_pred = clf.predict(X_test)
```

[30]:
```
# Plotting the Decision tree
dot_data = tree.export_graphviz(
    clf,
    out_file=None,
    feature_names=iris.feature_names,
    class_names=iris.target_names,
    filled=True,
    rounded=True,
    special_characters=True,
)
graph = graphviz.Source(dot_data)
graph
```

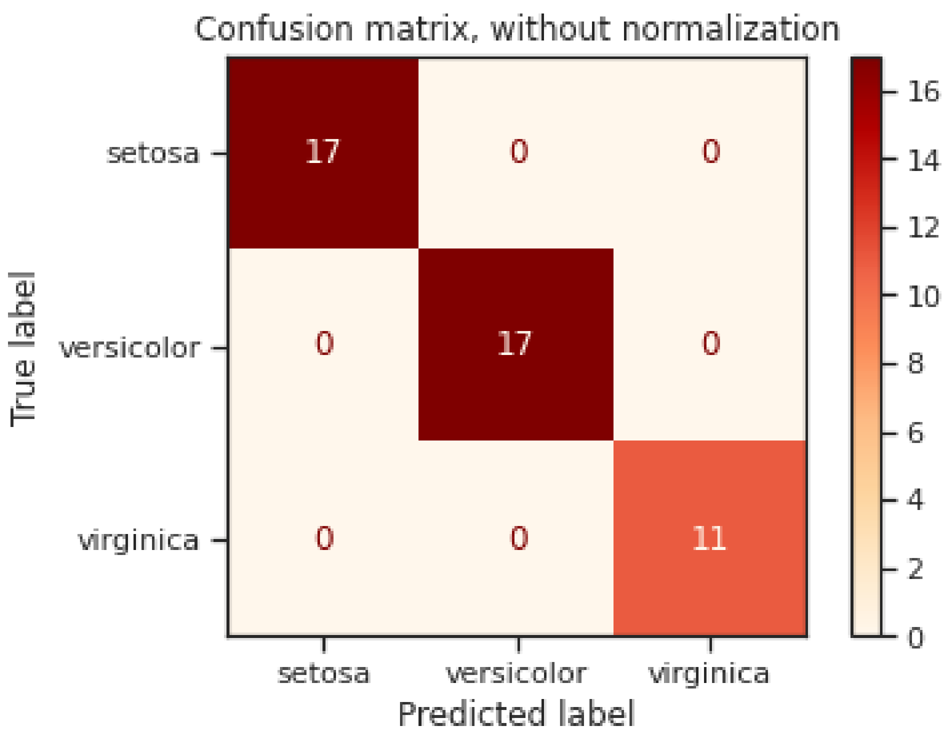
[30]:

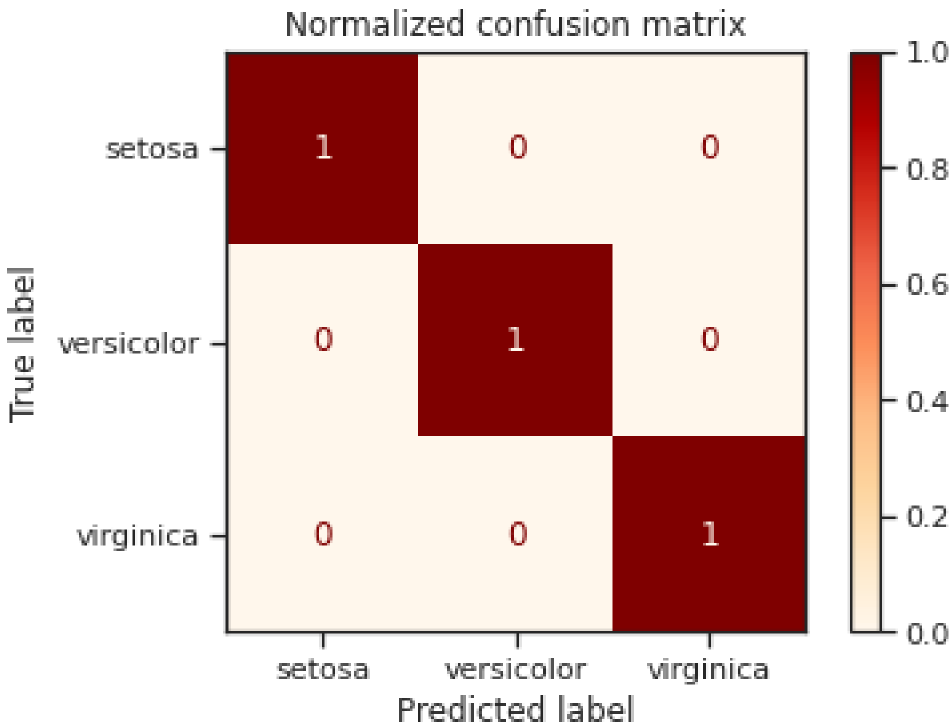


[31]:
```python
class_names = iris.target_names
# Accuracy and other metrics
print(classification_report(y_test, y_pred, target_names=class_names))

# Confusion matrix for the Decision tree classifier
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(clf, X_test, y_test,
                                 display_labels=class_names,
                                 cmap=plt.cm.OrRd,
                                 normalize=normalize)
    disp.ax_.set_title(title)


plt.show()
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        17
```

```
  versicolor       1.00      1.00      1.00        17
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```





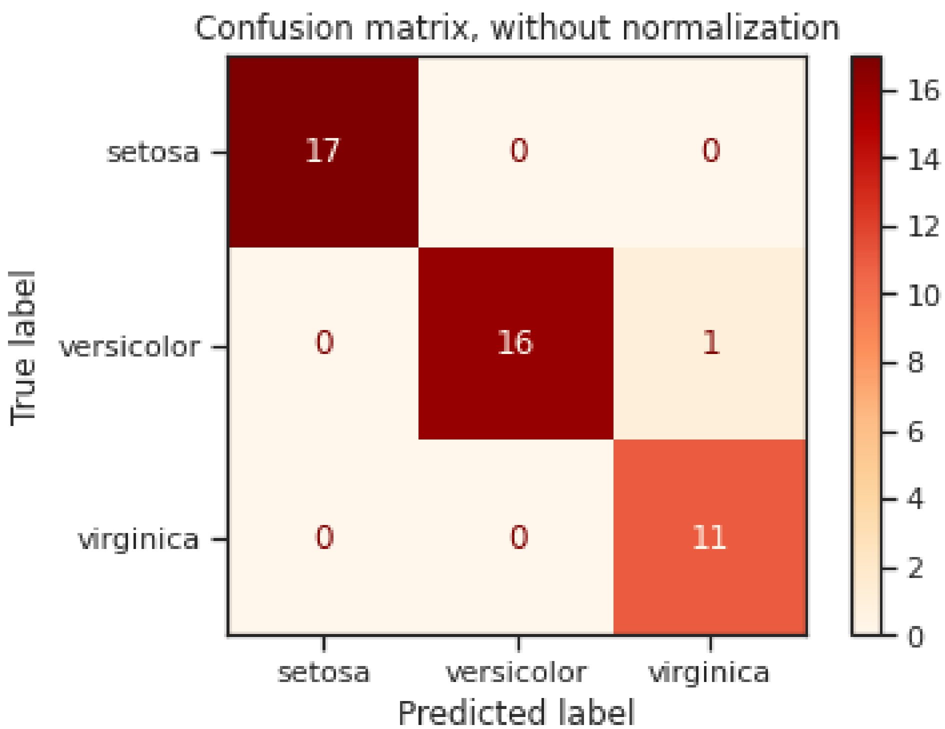
Gaussian Naive Bayes Classifier

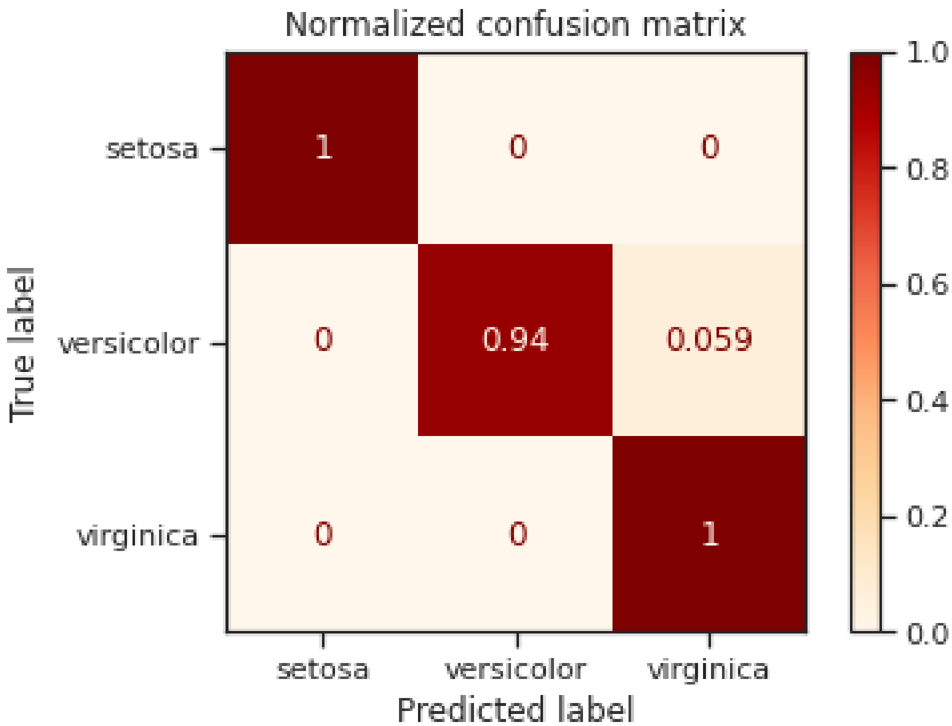
Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. For Gaussian Naive bayes classifier, the likelihood of the features is assumed to be Gaussian.

[32]:
```
from sklearn.naive_bayes import GaussianNB

# Fit a Gaussian Naive Bayes on Iris dataset
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)

# Accuracy and other metrics
print(classification_report(y_test, y_pred, target_names=class_names))



# plot confusion matrix for Naive Bayes
for title, normalize in titles_options:
    disp = plot_confusion_matrix(gnb, X_test, y_test,
                                 display_labels=class_names,
                                 cmap=plt.cm.OrRd,
                                 normalize=normalize)
```

```
    disp.ax_.set_title(title)


plt.show()
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        17
  versicolor       1.00      0.94      0.97        17
   virginica       0.92      1.00      0.96        11

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```






```
Comparing the accuracies of both the methods, we can see that decision tree
performs better than naive bayes classifier
```