Importing Open CV for extracting feature and Numpy for mathematical manipulation ¶

In [1]:
```
import cv2
import numpy as np
```

In [2]:
```
# Changing to directory contains classes of fruits
```

In [3]:
```
cd /kaggle/input/fruits/fruits-360/Training
```

```
/kaggle/input/fruits/fruits-360/Training
```

In [4]:
```
  # Can chose any fruits for clustering
```

In [5]:
```
ls
```

```
'Apple Braeburn'/       'Grape Blue'/          'Pear Monster'/
'Apple Crimson Snow'/   'Grape Pink'/          'Pear Red'/
'Apple Golden 1'/       'Grape White'/         'Pear Stone'/
'Apple Golden 2'/       'Grape White 2'/       'Pear Williams'/
'Apple Golden 3'/       'Grape White 3'/        Pepino/
'Apple Granny Smith'/   'Grape White 4'/       'Pepper Green'/
'Apple Pink Lady'/      'Grapefruit Pink'/     'Pepper Orange'/
'Apple Red 1'/          'Grapefruit White'/    'Pepper Red'/
'Apple Red 2'/           Guava/                'Pepper Yellow'/
'Apple Red 3'/           Hazelnut/              Physalis/
'Apple Red Delicious'/   Huckleberry/          'Physalis with Husk'/
'Apple Red Yellow 1'/    Kaki/                  Pineapple/
'Apple Red Yellow 2'/    Kiwi/                 'Pineapple Mini'/
 Apricot/                Kohlrabi/             'Pitahaya Red'/
 Avocado/                Kumquats/              Plum/
'Avocado ripe'/          Lemon/                'Plum 2'/
 Banana/                'Lemon Meyer'/         'Plum 3'/
'Banana Lady Finger'/    Limes/                 Pomegranate/
'Banana Red'/            Lychee/               'Pomelo Sweetie'/
 Beetroot/               Mandarine/            'Potato Red'/
 Blueberry/              Mango/                'Potato Red Washed'/
'Cactus fruit'/         'Mango Red'/           'Potato Sweet'/
'Cantaloupe 1'/          Mangostan/            'Potato White'/
'Cantaloupe 2'/          Maracuja/              Quince/
 Carambula/             'Melon Piel de Sapo'/  Rambutan/
 Cauliflower/            Mulberry/             Raspberry/
'Cherry 1'/              Nectarine/            Redcurrant/
'Cherry 2'/             'Nectarine Flat'/       Salak/
'Cherry Rainier'/       'Nut Forest'/          Strawberry/
'Cherry Wax Black'/     'Nut Pecan'/           'Strawberry Wedge'/
'Cherry Wax Red'/       'Onion Red'/           Tamarillo/
'Cherry Wax Yellow'/    'Onion Red Peeled'/    Tangelo/
 Chestnut/              'Onion White'/        'Tomato 1'/
 Clementine/             Orange/              'Tomato 2'/
 Cocos/                  Papaya/              'Tomato 3'/
 Corn/                  'Passion Fruit'/      'Tomato 4'/
'Corn Husk'/             Peach/               'Tomato Cherry Red'/
'Cucumber Ripe'/        'Peach 2'/            'Tomato Heart'/
'Cucumber Ripe 2'/      'Peach Flat'/         'Tomato Maroon'/
 Dates/                  Pear/                'Tomato Yellow'/
 Eggplant/              'Pear 2'/             'Tomato not Ripened'/
 Fig/                   'Pear Abate'/          Walnut/
'Ginger Root'/          'Pear Forelle'/        Watermelon/
 Granadilla/            'Pear Kaiser'/
```

In [6]:
```
lists_fruit = ['Apple Red 1','Banana','Lychee','Watermelon','Kohlrabi','Tomato not Ripened']      # Creatin
g lists of fruits
```

Reading all images of listed class and taking average value of r , g and b channels of every image

In [7]:

```
import os
def feature_gen(lists_fruits):
    my_lists = {key:[] for key in lists_fruits}
    for file in lists_fruits:
        list_files = os.listdir(file)
        os.chdir(file)
        for files in list_files:
            my_lists[str(file)].append(cv2.imread(files))

        os.chdir('..')
    return my_lists

lis = feature_gen(lists_fruit)

lists_n = [*lis]
for file in lists_n:
    val = []
    for fil in lis[file]:
        rgb = [np.average(fil[:,:,2]),np.average(fil[:,:,1]),np.average(fil[:,:,0])]
        val.append(rgb)
    lis[file] = val
```

In [9]:
```
lists=list(lis.values())
```

Finding all the interval for divide classes which will help in plotting

In [10]:
```
lower_bound = []
upper_bound= []
lower = 0
upper = 0
for num in lists:
    upper  = (len(num))+upper
    lower_bound.append(lower)
    upper_bound.append(upper)
    lower = (len(num))+lower
```

In [11]:
```
lower_bound
```

Out[11]:
```
[0, 492, 982, 1472, 1947, 2418]
```

In [12]:
```
upper_bound
```

Out[12]:
```
[492, 982, 1472, 1947, 2418, 2892]
```

Creating lists of r , g and b for plotting from dictionary

In [13]:
```
import itertools
lists
lists = list(itertools.chain.from_iterable(lists))
lists = np.array(lists)
lists.shape

x = []
y = []
z = []
for i in range(0,lists.shape[0]):
    x.append(lists[i][0])
    y.append(lists[i][1])
    z.append(lists[i][2])
```

In [14]:
```
lower_bound[1]
```

Out[14]:
```
492
```

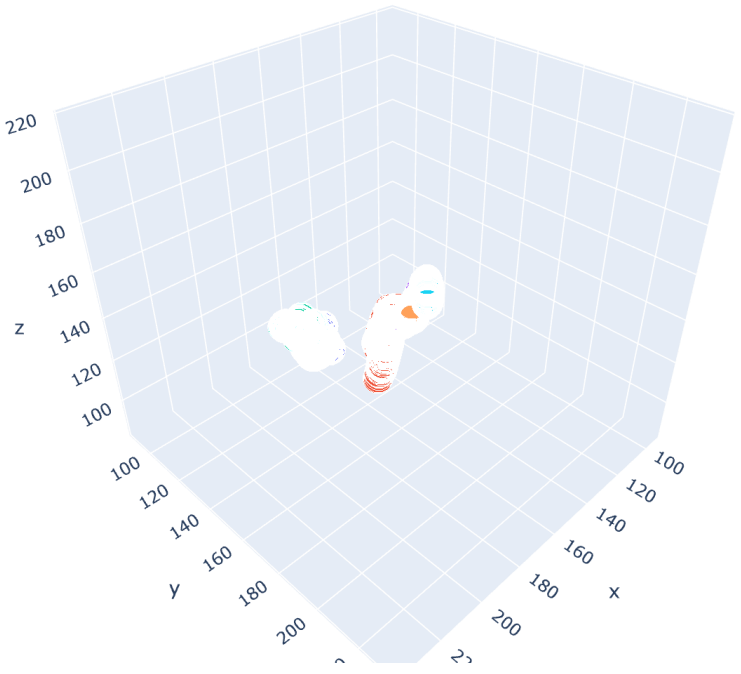
Plotting the clusters before applying Expectation Maximization

In [15]:

```
import plotly.offline as py
import plotly.graph_objs as go
py.init_notebook_mode(connected=True)

data = []
for i in range(0, len(lists_fruit)):
    data.append(go.Scatter3d(
        x=x[lower_bound[i]:upper_bound[i]-1],
        y=y[lower_bound[i]:upper_bound[i]-1],
        z=z[lower_bound[i]:upper_bound[i]-1],
        mode='markers',
        marker=dict(
            size=12,
            line=dict(
                color='rgba(217, 217, 217, 0.14)',
                width=0.5
            ),
            opacity=1
        ),
        name = lists_fruit[i]
    ))

layout = go.Layout(
    margin=dict(
        l=0,
        r=0,
        b=0,
        t=0
    )
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig)
```



```
from scipy.stats import multivariate_normal



class Expectation_Maximization:
    def __init__(self, num_cluster, max_iter=5):
        self.num_cluster = num_cluster
        self.max_iter = int(max_iter)

    def initialize(self, X):
        self.shape = X.shape
        self.n, self.m = self.shape

        self.phi = np.full(shape=self.num_cluster, fill_value=1/self.num_cluster)               # Initializi
ng scales for all clusters
        self.weights = np.full( shape=self.shape, fill_value=1/self.num_cluster)                # Initializi
ng weights for all points

        random_row = np.random.randint(low=0, high=self.n, size=self.num_cluster)               # Setting th
e size of initial clusters randomly
        self.mu = [  X[row_index,:] for row_index in random_row ]                               # Initializi
ng the mean
        self.sigma = [ np.cov(X.T) for _ in range(self.num_cluster) ]                           # Initializi
ng the variance

    def e_step(self, X):
        self.weights = self.predict_proba(X)                                                    # Updadting
weights
        self.phi = self.weights.mean(axis=0)                                                    # Updating p
hi
        # here mu and sigma is constant

    def m_step(self, X):
        # Updating mu and sigma but weight and phi is constant
        for i in range(self.num_cluster):
            weight = self.weights[:, [i]]
            total_weight = weight.sum()
            self.mu[i] = (X * weight).sum(axis=0) / total_weight
            self.sigma[i] = np.cov(X.T,
                aweights=(weight/total_weight).flatten(),
                bias=True)

    def fit(self, X):                                                                           # fit the mod
el
        self.initialize(X)

        for iteration in range(self.max_iter):
            self.e_step(X)
            self.m_step(X)

    def predict_proba(self, X):                                                                 # Function fo
r calculating pdf
        likelihood = np.zeros( (self.n, self.num_cluster) )
        for i in range(self.num_cluster):
            distribution = multivariate_normal(
                mean=self.mu[i],
                cov=self.sigma[i])
            likelihood[:,i] = distribution.pdf(X)

        numerator = likelihood * self.phi
        denominator = numerator.sum(axis=1)[:, np.newaxis]
        weights = numerator / denominator
```

```
        return weights

    def predict(self, X):                                                    # Predict the
cluster
        weights = self.predict_proba(X)
        return np.argmax(weights, axis=1)
```

In [18]:
```
np.random.seed(42)
expm = Expectation_Maximization(num_cluster=6, max_iter=10)
expm.fit(X)
```

In [19]:
```
# Pedicting cluster number custom

num = np.unique(expm.predict([[180,212,178]]))
print(num)
```

```
[5]
```

Creating 3-D cluster predicted by EM model

In [20]:
```
lists_new = list(lis.values())
```

In [21]:
```
import itertools
lists_new = list(itertools.chain.from_iterable(lists_new))
lists_new = np.array(lists)
lists_new.shape
```

Out[21]:
```
(2892, 3)
```

Creating a dictionary of predicted cluster and points

In [22]:
```
dicts = {}

for i in range(0,lists_new.shape[0]):
        dicts.setdefault(int(np.unique(expm.predict(lists[i]))),[]).append(lists[i])
```

Plotting predicted clusters

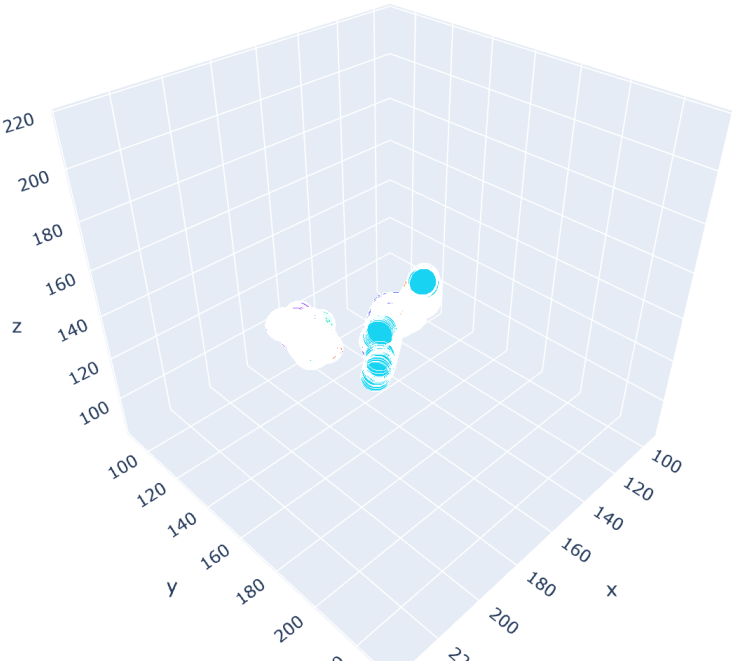
In [23]:

```
import plotly.offline as py
import plotly.graph_objs as go
py.init_notebook_mode(connected=True)

data = []
for j in range(0,6):
    x=[]
    y=[]
    z=[]
    for i in range(0, len(dicts[j])):

        x.append(dicts[j][i][0])
        y.append(dicts[j][i][1])
        z.append(dicts[j][i][2])
    data.append(go.Scatter3d(
        x=x,
        y=y,
        z=z,
        mode='markers',
        marker=dict(
            size=12,
            line=dict(
                color='rgba(217, 217, 217, 0.14)',
                width=0.5
                ),
            opacity=1
            ),
            name = j
            ))

layout = go.Layout(
    margin=dict(
        l=0,
        r=0,
        b=0,
        t=0
    )
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig)
```


220
200
180
160
140
120
100
z
100
120
140
160
180
200
y
100
120
140
160
180
200
x