Practical – 1: Read soil health card data and find the statistics from the data

- Learn python library installation and use in python
- Read Soil health card data file
- Extract the required data from the file
- Understanding and statistics of data

Code:

```
#Reading an excel file using Python
import xlrd
import numpy as np

# Give the location of the file
loc = ("Mehasana_EC_f.xlsx")

# To open Workbook
wb = xlrd.open_workbook(loc)
sheet = wb.sheet_by_index(0)

# For row 0 and column 0
sheet.cell_value(0, 0)
# Extracting number of rows
print('No of rows in the sheet:',sheet.nrows)

#Extracting number of columns
print('No of columns in the sheet:',sheet.ncols)

# Extracting all columns name
for i in range(sheet.ncols):
    print(sheet.cell_value(0, i))

# save particular column
x = []
for i in range (sheet.nrows):
    x.append(sheet.cell_value(i,5))

   # print(x)

Min_EC = np.min(x[1:21]);
print('Minimum value of EC:',Min_EC)
Max_EC = np.max(x[1:21]);
print ('Maximum value of EC:',Max_EC)
Mean_EC = np.mean(x[1:21]);
print('Mean value of EC:',Mean_EC)
Std_EC = np.std(x[1:21]);
print('Standard Deviation of EC:',Std_EC)
Median_EC = np.median(x[1:21]);
print('Median of EC:',Median_EC)
cov_EC = np.cov(x[1:21])
print ('Covariance of EC:',cov_EC)

#convert list to matrix to find variance
y = np.matrix(x[1:21])
var_EC = y.var
print('Variance of EC:',var_EC)
```

Output:

```
No of rows in the sheet: 21
No of columns in the sheet: 6
District
Taluka
Village
Latitude
Longitude
EC
Minimum value of EC: 0.24
Maximum value of EC: 1.37
Mean value of EC: 0.44749999999999995
Standard Deviation of EC: 0.2501374622082826
Median of EC: 0.415
Covariance of EC: 0.06586184210526316
Variance of EC: <bound method matrix.var of matrix([[0.36, 0.29, 0.44, 0.41, 0.33,
0.34, 0.46, 0.28, 0.24, 0.46,
         0.45, 0.43, 0.55, 0.24, 0.29, 0.46, 0.28, 0.42, 0.85, 1.37]])>
```

Practical – 2: Plot different types of graphs using soil health card data

- Plot histogram of any one variable from given data file
- Scatter plot of any one property with its respective geo-location
- Show property value distribution in scatter plot with respect to marker size

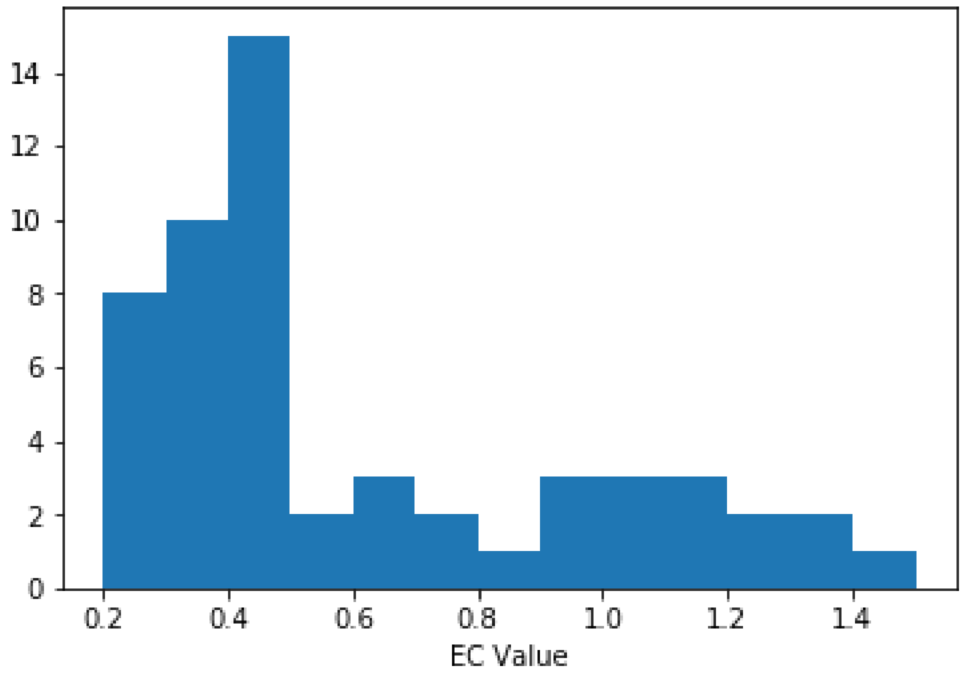
Code:

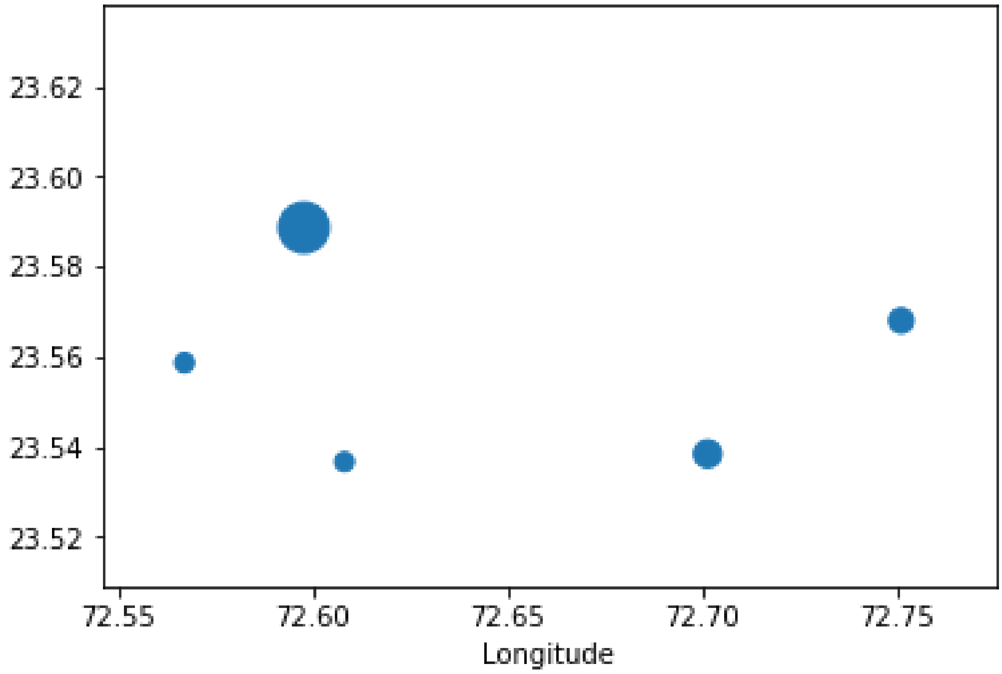
```
# -*- coding: utf-8 -*-
"""
Created on Thu May 30 15:01:23 2019

@author: GeoSpatial-3
"""

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

#Read csv data
Mehsana_SH = pd.read_csv('Mehsana_Soil_Health_Card.csv')
Mehsana_SH.head()
Mehsana_SH.info()
Mehsana_SH.describe()
Mehsana_SH.columns

# Extract Latitude
Lat = Mehsana_SH['Latitude']

#Convert to array
Lat = np.array(Lat)

#reshape array 1d to 2d
Lat = Lat.reshape(-1,1)

#Extract Longitude
Lon = Mehsana_SH['Longitude']

#Convert to array
Lon = np.array(Lon)
```

```
#reshape array 1d to 2d
Lon = Lon.reshape(-1,1)

EC = Mehsana_SH['EC']

EC = np.array(EC)

EC = EC.reshape(-1,1)

x = Mehsana_SH[['EC','pH']]
y = np.array(x)
z = y[0:510,0]
plt.hist(z[0:50],bins=[0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.4,1.5])
plt.xlabel('EC Value')


p = Mehsana_SH['P2O5']
p = np.array(p)
p = p.reshape(-1,1)
```

```
import matplotlib.pyplot as plt
import numpy as np


Lon1 = Lon[0:5]
Lat1 = Lat[0:5]
p1 = p[0:5]

plt.scatter(Lon1, Lat1, p1, marker='o',cmap='viridis')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
```

Output:





<u>Practical – 3: Estimating Soil properties using simple and multiple linear regression</u>

- Read csv data
- Data preparation for regression
- Simple linear regression using statesmodel
- Multiple linear regression using statesmodel
- Scatter plot for regression result

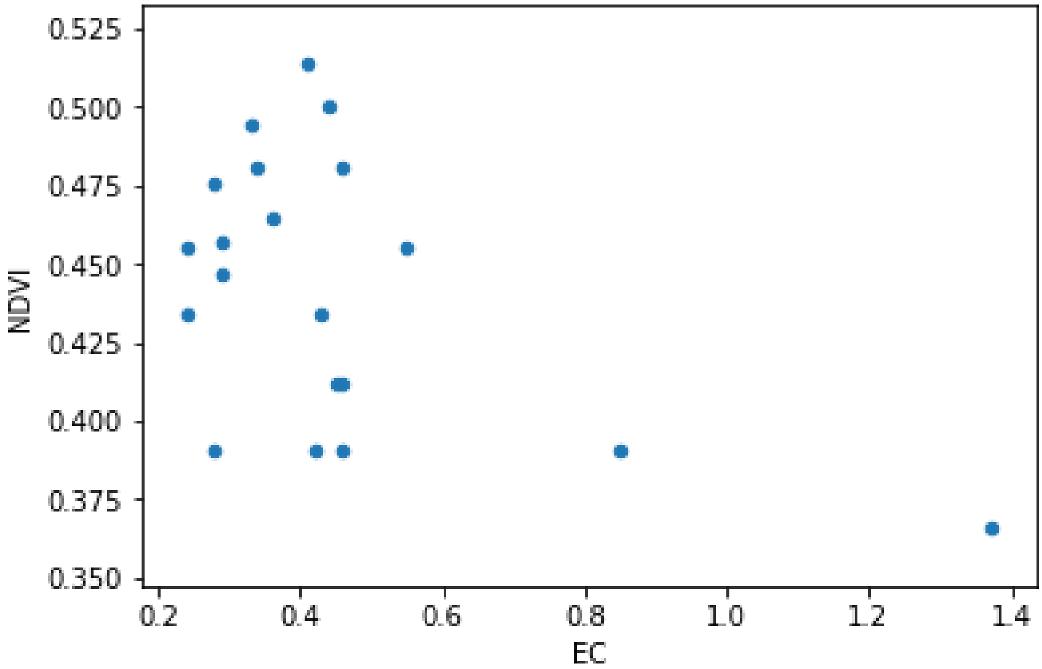
Code:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import seaborn as sns

Mehsana = pd.read_csv('Mehasana_EC_pH_f.csv')
Mehsana.head()
Mehsana.info()
Mehsana.describe()
Mehsana.columns

#correlation between the variables in the dataset
Mehsana.corr()

#Training a Linear Regression Model
X1 = Mehsana['EC']
y1 = Mehsana['NDVI']
x =np.array( X1[1:20])
y = np.array(y1[1:20])

# fit the statsmodel
X_constant = sm.add_constant(x)
model = sm.OLS(y, X_constant)
lin_reg = model.fit()
lin_reg.summary()

#Scatter plot
Mehsana.plot(kind='scatter',x='EC',y='NDVI')
```

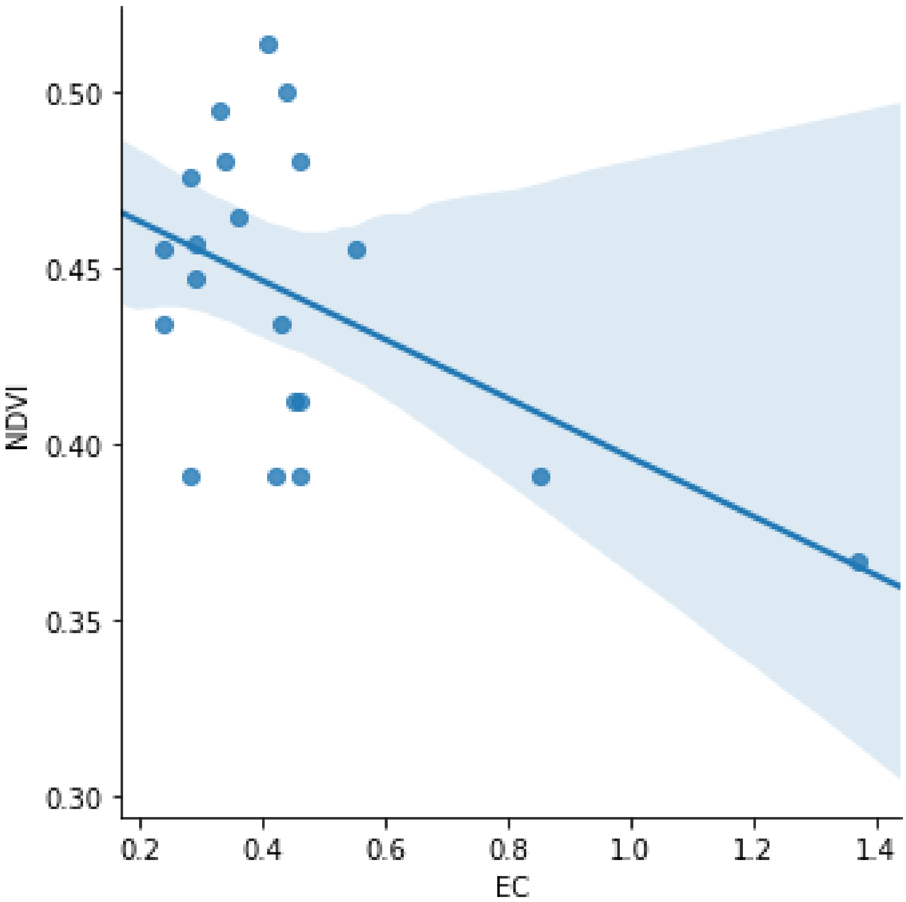
```python
#Fit the regression line
sns.lmplot(x='EC',y='NDVI',data=Mehsana)


"""
Multiple linear regression
"""

import numpy as np
import pandas as pd
import statsmodels.api as sm


data = pd.read_csv('Multiple_LR.csv')
data.head()
data.info()
data.describe()
data.columns

data.corr()


X2 = data['EC']
y2 = data[['DEM','NDVI','Rain']]

x1 =np.array( X2[1:21])
y1 = np.array(y2[1:21])

Y_constant_ = sm.add_constant(y1)
model = sm.OLS(x1, Y_constant_)
lin_reg = model.fit()
lin_reg.summary()
```

Output:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.249
Model:                            OLS   Adj. R-squared:                  0.205
Method:                 Least Squares   F-statistic:                     5.645
Date:                Thu, 30 May 2019   Prob (F-statistic):             0.0295
Time:                        22:54:05   Log-Likelihood:                 35.847
No. Observations:                  19   AIC:                            -67.69
Df Residuals:                      17   BIC:                            -65.80
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.4784      0.018     26.486      0.000       0.440       0.517
x1            -0.0826      0.035     -2.376      0.030      -0.156      -0.009
==============================================================================
Omnibus:                        0.554   Durbin-Watson:                   0.570
Prob(Omnibus):                  0.758   Jarque-Bera (JB):                0.600
Skew:                           0.121   Prob(JB):                        0.741
Kurtosis:                       2.164   Cond. No.                         4.75
==============================================================================
```



0.525
0.500
0.475
0.450
0.425
0.400
0.375
0.350
NDVI
0.2
0.4
0.6
0.8
1.0
1.2
1.4
EC



0.50
0.45
0.40
0.35
0.30
NDVI
0.2
0.4
0.6
0.8
1.0
1.2
1.4
EC

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.341
Model:                            OLS   Adj. R-squared:                  0.209
Method:                 Least Squares   F-statistic:                     2.584
Date:                Thu, 30 May 2019   Prob (F-statistic):             0.0919
Time:                        22:55:38   Log-Likelihood:                 2.9014
No. Observations:                  19   AIC:                             2.197
Df Residuals:                      15   BIC:                             5.975
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.1528      0.775      1.488      0.157      -0.498       2.804
x1            -0.0030      0.002     -1.434      0.172      -0.007       0.001
x2            -1.3881      1.697     -0.818      0.426      -5.006       2.229
x3             0.0015      0.006      0.247      0.808      -0.012       0.015
==============================================================================
Omnibus:                        9.483   Durbin-Watson:                   0.905
Prob(Omnibus):                  0.009   Jarque-Bera (JB):                7.727
Skew:                           0.925   Prob(JB):                       0.0210
Kurtosis:                       5.518   Cond. No.                     2.84e+03
```

Practical – 4: Predicting Soil property using deep learning method

- Load csv data
- Data preparation for neural network model
- Define neural network model using keras
- Compile the model
- Fit the model
- Evaluate the model
- Find the rmse for predicted data
- Plot actual and predicted data using matplotlib

Code:

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

#Input dimension
tot1 = 13
perce = 70*0.01

#Load the dataset
import csv
ecs = []
dems = []
feats = []
with open('Gandhinagar_Patan.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0

    for row in csv_reader:
        newfeat = []
        #print(row)
        ecs.append(row[0])
        dems.append(row[1])
        newfeat.append(row[2])
        newfeat.append(row[3])
        newfeat.append(row[4])
        newfeat.append(row[5])
        newfeat.append(row[6])
        newfeat.append(row[7])
        newfeat.append(row[8])
        newfeat.append(row[9])
        newfeat.append(row[10])
        newfeat.append(row[11])
```

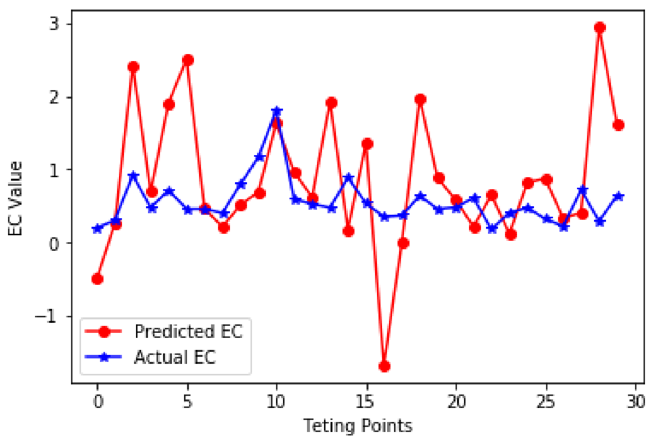
```
        newfeat.append(row[11])
        newfeat.append(row[12])
        newfeat.append(row[13])
        newfeat.append(row[14])
        feats.append(newfeat)

        n = len(ecs) -1
y = np.zeros((n,1))
X = np.zeros((n,tot1))
for i in range(len(ecs)):
    if(i==0):
        pass
    else:
        y[i-1] = float(ecs[i])
        X[i-1,0] = float(dems[i])
        for k in range(tot1-1):
            X[i-1,k+1] = float(feats[i][k])

        X.shape

#Scale the data
scalarX, scalarY = MinMaxScaler(), MinMaxScaler()
scalarX.fit(X)
scalarY.fit(y)
X2 = scalarX.transform(X)
Y2 = scalarY.transform(y)

Y2.shape

#Random seed for the model
np.random.seed(10)
```

```
#Arrange the data
ind = np.random.permutation(X2.shape[0])
X3 =X2[ind]
Y3 =Y2[ind]

# Data Splitting for training and testing
m = X3.shape[0]
x2train = X3[0: int(m*perce)]
x2test = X3[int(m*0.7):]

y2train = Y3[0: int(m*perce)]
y2test = Y3[int(m*0.7):]

#Define and Compile
model = Sequential()
model.add(Dense(30, input_dim=tot1, activation='relu'))
model.add(Dense(20, activation = 'relu'))
model.add(Dense(15,  activation='linear'))
model.add(Dense(1))

model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

#Fit the model
model.fit(x2train, y2train,validation_data= (x2test,y2test), epochs=8500, verbose=1)
y2pred = model.predict (x2test)
y2inverted_pred = scalarY.inverse_transform(y2pred)
y2inverted_real = scalarY.inverse_transform(y2test)
x2inverted = scalarX.inverse_transform(x2test)

#Evaluate the model
mse = np.mean((y2inverted_real- y2inverted_pred)**2  )

```

```
# root mean squared error
# m is the number of training examples
m = 70
rmse = np.sqrt(mse/m)
print ('RMSE Value for the model is',rmse)

#Plot the actual and predicted value for the EC
plt.plot(y2inverted_pred,'r-o')
plt.plot(y2inverted_real, 'b-*')
plt.xlabel("Teting Points")
plt.ylabel("EC Value")
plt.gca().legend(('Predicted EC','Actual EC'))
```

Output:

RMSE Value for the model is 0.114478814275550338



Practical – 5: Soil supervised classification using Support Vector Machine

- Read shape file of soil data
- From shape file make the label data for SVM in sklearn
- Split the data for training and testing
- Apply SVM using sklearn

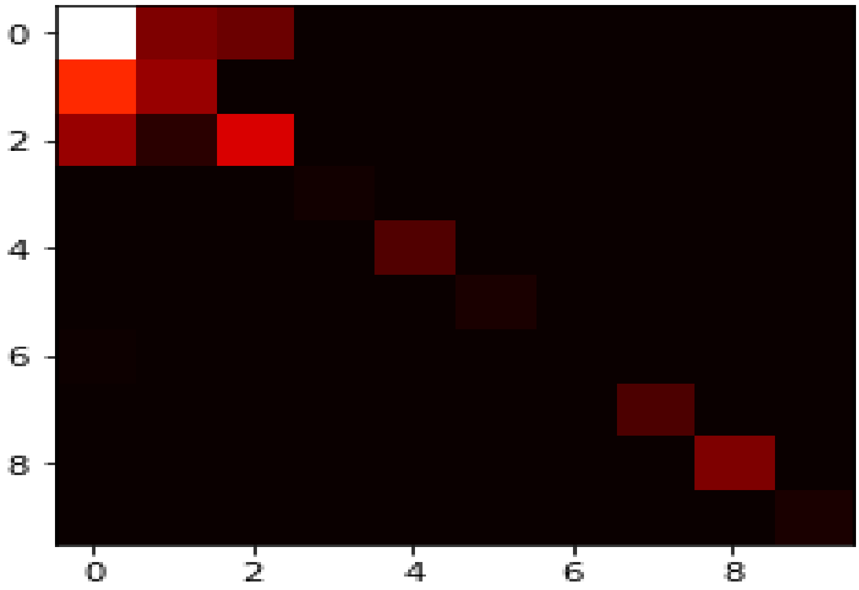
Code:

```
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import pandas as pd
import numpy as np
from pandas import DataFrame
import matplotlib.pyplot as plt
#Read data
data = pd.read_csv('NBSS_SVM_DATA.csv')
data_gt = np.array(data['PH'])
df =  DataFrame(data,columns=['Soil_Group','Soil_depth','Parent_Material',
                              'Erosion','Slope','Ground_Water','Soil_drain',
                              'Surface'])
data_fe = df.values

#Train, test splitting
X_train, X_test, y_train, y_test = train_test_split(data_fe, data_gt, random_state = 0)

#Fit model
svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train[0:500], y_train[0:500])

#Fit model
svm_predictions = svm_model_linear.predict(X_test[0:500])

#Find the accuracy
accuracy = svm_model_linear.score(X_test, y_test)

#Generate confusion matrix
cm = confusion_matrix(y_test, svm_predictions)

print('Accuracy of the SVM is:', accuracy*100)
print ('Confusion Matrix:',cm)
plt.imshow(cm,cmap='hot')
```

Output:

Accuracy of the SVM is: 65.91928251121077

```
Confusion Matrix: [[149  26  22   0   0   0   0   0   0   0]
                   [ 64  32   0   0   0   0   0   0   0   0]
                   [ 32   7  46   0   0   0   0   0   0   0]
                   [  0   0   0   2   0   0   0   0   0   0]
                   [  0   0   0   0  16   0   0   0   0   0]
                   [  0   0   0   0   0   4   0   0   0   0]
                   [  1   0   0   0   0   0   0   0   0   0]
                   [  0   0   0   0   0   0   0  15   0   0]
                   [  0   0   0   0   0   0   0   0  26   0]
                   [  0   0   0   0   0   0   0   0   0   4]]
```



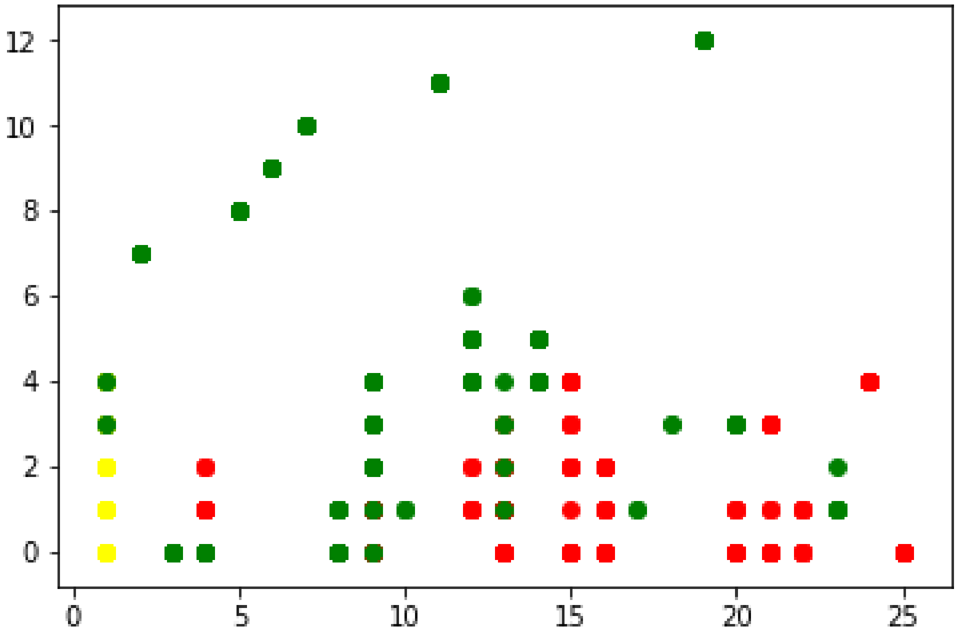
Practical – 6: Soil unsupervised classification using Gaussian Mixture Model

- Load and understand input data for unsupervised classification
- Apply GMM using sklearn

Code:

```python
import pandas as pd
import matplotlib.pyplot as plt
from pandas import DataFrame
from sklearn.mixture import GaussianMixture

data = pd.read_csv('NBSS_SVM_DATA.csv')
df =  DataFrame(data,columns=['Soil_Group','Soil_depth','Parent_Material'])
data_fe = df.values

# turn it into a dataframe
data_fea = pd.DataFrame(data_fe)

# plot the data
plt.scatter(df['Soil_Group'], df['Soil_depth'], df['Parent_Material'])

gmm = GaussianMixture(n_components = 3)

# Fit the GMM model for the dataset which expresses the dataset as a
# mixture of 3 Gaussian Distribution
gmm.fit(data_fea)

# Assign a label to each sample
labels = gmm.predict(data_fea)
data_fea['labels']= labels
d0 = data_fea[data_fea['labels']== 0]
d1 = data_fea[data_fea['labels']== 1]
d2 = data_fea[data_fea['labels']== 2]

# plot three clusters in same plot
plt.scatter(d0[0], d0[1], c ='r')
plt.scatter(d1[0], d1[1], c ='yellow')
plt.scatter(d2[0], d2[1], c ='g')

# print the converged log-likelihood value
#print(gmm.lower_bound_)

# print the number of iterations needed for the log-likelihood value to converge
print('Number of iterations needed to converge:',gmm.n_iter_)
print(gmm.score)
```

Output:Number of iterations needed to converge: 19



Practical – 7: Yieldoptimization using N, P, K value of soil

- Load data
- Develop multiple linear regression model to estimate the yield using N,P,K values from the data
- Change the N,P,K combinations to optimize yield

Code:

```
import pandas as pd
import numpy as np
#import matplotlib.pyplot as plt
import statsmodels.api as sm

# read data-set
data = pd.read_csv('optimization.csv')
data.head()
data.info()
data.describe()
data.columns

#Training a Multiple Linear Regression Model
X = data[['N','P','K']]
y = data['yield']

x1 =np.array( X[0:7])
y1 = np.array(y[0:7])

X_constant_ = sm.add_constant(x1)
model = sm.OLS(y1, X_constant_)
lin_reg = model.fit()
lin_reg.summary()
```

Output:

```
==============================================================================
Dep. Variable:                      y   R-squared:                       0.936
Model:                            OLS   Adj. R-squared:                  0.872
Method:                 Least Squares   F-statistic:                     14.64
Date:                Thu, 30 May 2019   Prob (F-statistic):             0.0269
Time:                        23:48:10   Log-Likelihood:                0.84167
No. Observations:                   7   AIC:                             6.317
Df Residuals:                       3   BIC:                             6.100
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.2619      0.296      4.256      0.024       0.318       2.205
x1             0.0074      0.004      2.004      0.139      -0.004       0.019
x2             0.0123      0.006      1.944      0.147      -0.008       0.032
x3             0.0030      0.004      0.726      0.521      -0.010       0.016
==============================================================================
Omnibus:                          nan   Durbin-Watson:                   1.219
Prob(Omnibus):                    nan   Jarque-Bera (JB):                0.660
Skew:                           0.275   Prob(JB):                        0.719
Kurtosis:                       1.600   Cond. No.                         343.
==============================================================================
```