Crop Inventory and Condition Assessment

- **Objective 1**: Descriptive Measures.

- **Programming Language**: Python 3 or above.

- **Time Required**: 5 Hours

- **Prerequisites and Programming skill:**
  1. Python 3 or above should be installed on the computer.
  2. Student must have basic understanding of statistics.

- **Data:** Village level acreage data and crop condition in 4 categories (excellent, good, medium poor), is given for wheat and rice for different talukas in a district.

- **Introduction:** In this practical we will discuss about the Descriptive Measures in statistical data. We will find the mean, median, mode and variance values corresponding the data to statistically analyze it.

1.1 Express each village acreage as percent geographic area (normalize with geographic area).

1.2 Compute range, mean, median, mode, variance, std dev and coefficient of variation of village acreages in a taluka.

```
from osgeo import gdal
from osgeo import ogr
import numpy as np
import math
from scipy.stats import mode
import os

def calculateAcreage(crop,filepath):
    print("Analysing",crop,"data!")
    print(filepath)
    dataset = gdal.Open(filepath)
    if dataset:
        band = dataset.GetRasterBand(1)
    else:
        print("Cannot open file:")
        exit()


    rast_array = np.array(band.ReadAsArray())
    tcount = 0
    count = 0

    withoutNoData = []

    for row in rast_array:
        for element in row:
            tcount = tcount + 1
            if math.isnan(element) == False and element != 0.0:
                count = count+1
                withoutNoData.append(element)
```

```python

    minval = min(withoutNoData)
    maxval = max(withoutNoData)
    meanval = np.mean(withoutNoData)
    medianval = np.median(withoutNoData)
    modeval = float(mode(withoutNoData)[0])
    modefreq = int(mode(withoutNoData)[1])
    sdval = np.std(withoutNoData)
    varianceval = np.var(withoutNoData)
    rangeval = maxval - minval
    coefvariation = sdval * 100 / meanval

    print("Min Value:", minval)
    print("Max Value:", maxval)
    print("Mean :", meanval)
    print("Median :", medianval)
    print("Mode :", modeval)
    print("Mode frequency :", modefreq)
    print("Variance :", varianceval)
    print("Standard Deviation :", sdval)
    print("Range :", rangeval)
    print("Coefficient of Variation :", coefvariation)

    area = count * 9/1000000

    print("Total", crop ,"area:", area, "sqkm")
    print("------------------------------------")
    return(area)

```

```python
shapfile_path = r'C:\Users\dhaval.panchal.ISPL\Downloads\Wheat_Paddy_ahm\Wheat_Paddy_ahm\Dhok_ahm_shpFile'
shpfile = ogr.Open(shapfile_path)
shape = shpfile.GetLayer(0)
feature = shape.GetFeature(0)
villageArea = feature.geometry().GetArea()/1000000

tif_path = r'C:\Users\dhaval.panchal.ISPL\Downloads\Wheat_Paddy_ahm\Wheat_Paddy_ahm\NDVI_Mask'
paddy_tif = r'ndvi_mask_paddy.tif'
wheat_tif = r'ndvi_mask_wheat.tif'

paddyacreage = calculateAcreage("Paddy", os.path.join(tif_path,paddy_tif))
wheatacreage = calculateAcreage("Wheat", os.path.join(tif_path,wheat_tif))

print("Area of village:" , villageArea, "sqkm")
print("----------------------------------")
print("Percentage area of Paddy:", 100 * paddyacreage / villageArea, "%")
print("Percentage area of Wheat:", 100 * wheatacreage / villageArea, "%")
```

1.3 Check for any anomalous values in data – outliers. Assume normal distribution, compute and check outliers , if any.

1.4 Create histogram and cumulative histograms of data.

1.6 Create histograms and cumulative histograms for categorical crop condition data.

```
from osgeo import gdal
from osgeo import ogr
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import mode, zscore
import os
import math

filepath = "/home/ispluser/Dimple/Wheat_Paddy_ahm/NDVI_Mask_tiff"
paddy = "ndvi_mask_paddy_zero.tif"
wheat = "ndvi_mask_wheat_zero.tif"
def data(filepath,image_path):

    dataset = gdal.Open(os.path.join(filepath,image_path))
    band = dataset.GetRasterBand(1)

    rast_array = np.array(band.ReadAsArray())
    tcount = 0
    count = 0

    withoutNoData = []

    for row in rast_array:
        for element in row:
            tcount = tcount + 1
            if math.isnan(element) == False and element != 0.0:
                count = count+1
                withoutNoData.append(element)
    return withoutNoData

withoutNoData = data(filepath,wheat)

```

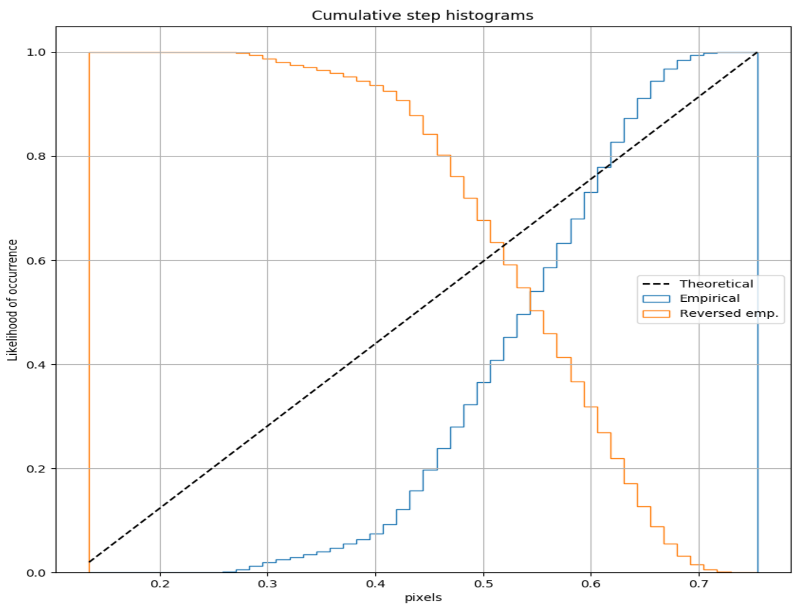
```python
# find ouotliers using zscore
outliers=[]
def detect_outlier(data_1):

    threshold=3
    mean_1 = np.mean(data_1)
    std_1 =np.std(data_1)

    z_score = []
    for y in data_1:
        z_score_= (y - mean_1)/std_1
        z_score.append(z_score_)
        if np.abs(z_score_) > threshold:
            outliers.append(y)
#    print("Function block")
#    print("Zscore", zscore)
#    print("Outliers",outliers )
    return z_score, outliers

z_score, outliers = detect_outlier(withoutNoData)
```

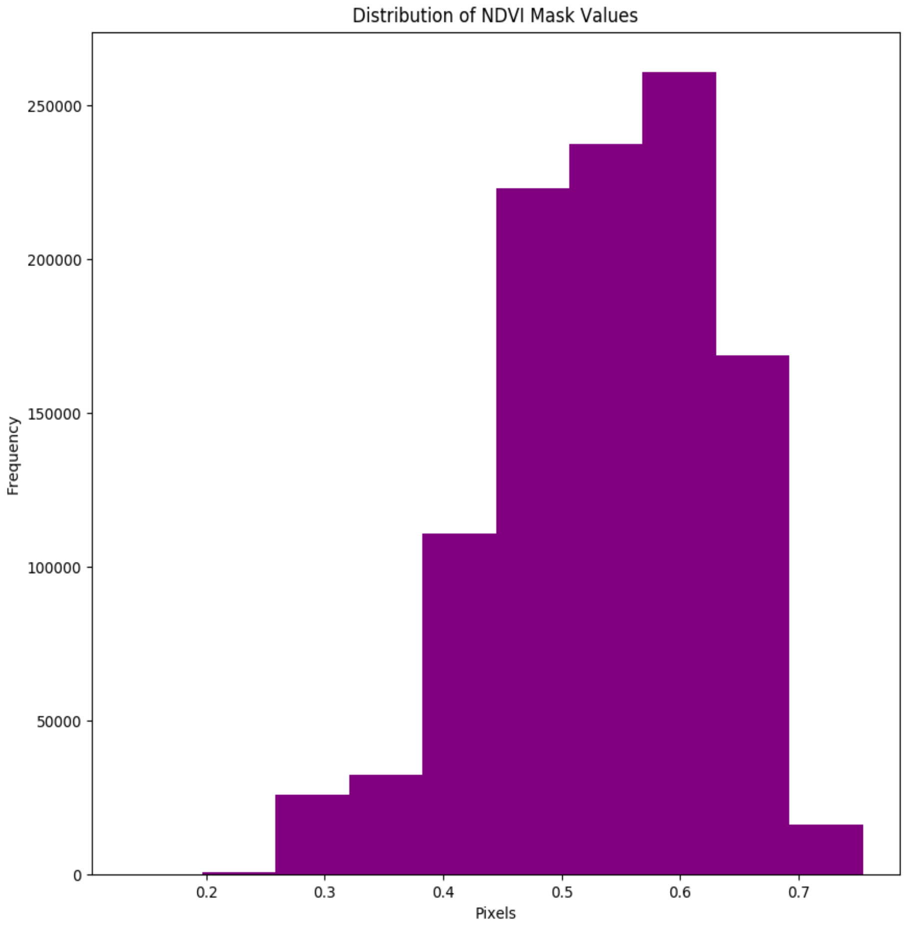
```python
# histogram:
def histogram(data):
    fig, ax = plt.subplots(figsize=(10,10))
    ax.hist(data, color='purple')

    ax.set(xlabel = 'Pixels',
           ylabel = 'Frequency',
           title = "Distribution of NDVI Mask Values");

```

```python
# cumulative histograms of data:
def cumulative_histogram(data):
    mu = 200
    sigma = 25
    n_bins = 50
    #x = NDVI_mask_hist
    x = data
    fig, ax = plt.subplots(figsize=(10, 10))

    # plot the cumulative histogram
    n, bins, patches = ax.hist(x, n_bins, density=True, histtype='step',
                               cumulative=True, label='Empirical')

    # Add a line showing the expected distribution.
    y = ((1 / (np.sqrt(2 * np.pi) * sigma)) *
         np.exp(-0.5 * (1 / sigma * (bins - mu))**2))
    y = y.cumsum()
    y /= y[-1]

    ax.plot(bins, y, 'k--', linewidth=1.5, label='Theoretical')

    # Overlay a reversed cumulative histogram.
    ax.hist(x, bins=bins, density=True, histtype='step', cumulative=-1,
            label='Reversed emp.')

    # tidy up the figure
    ax.grid(True)
    ax.legend(loc='right')
    ax.set_title('Cumulative step histograms')
    ax.set_xlabel('pixels')
    ax.set_ylabel('Likelihood of occurrence')

    plt.show()

cumulative_histogram(withoutNoData)
```

```python
# group array into categories:
def grouping(data):
    classified = {}
    counts = {'Poor':0,"Medium": 0, "Good": 0, "Excellent": 0}
    for i in range(len(withoutNoData)):
        if 0 < withoutNoData[i] <= 0.25:
            classified[withoutNoData[i]] = 'Poor';
            counts['Poor'] += 1
        elif 0.26 < withoutNoData[i] <= 0.50:
            classified[withoutNoData[i]] = 'Medium';
            counts['Medium'] += 1
        elif 0.51 < withoutNoData[i] <= 0.75:
            classified[withoutNoData[i]] = 'Good';
            counts['Good'] += 1
        else:
            classified[withoutNoData[i]] = 'Excellent'
            counts['Excellent'] += 1
    return classified,counts

categorized, count_dict = grouping(withoutNoData)

#histogram of conditional data:
plt.bar(count_dict.keys(),count_dict.values())
```

Output:




Distribution of NDVI Mask Values
Frequency
Pixels
250000
200000
150000
100000
50000
0
0.2
0.3
0.4
0.5
0.6
0.7

1.5 Find taluka having max and min variability.

1.7 Compare crop conditions across talukas.

```
def compare_taluka(shapefile_path_list, imagefile_path_list):
    acreage = []
    variability = []
    for i in range(len(shapefile_path_list)):
        shpfile = ogr.Open(shapefile_path_list[i])
        shape = shpfile.GetLayer(0)
        feature = shape.GetFeature(0)
        talukArea = feature.geometry().GetArea()/1000000
        variance, cropAcreage = calculateAcreage("Wheat", imagefile_path_list[i])
        acreage.append(cropAcreage)
        variability.append(variance)
    max_val , min_val = np.argmax(variability), np.argmin(variability)
    min_region_name  = imagefile_path_list[min_val].split('/')[-1].split('.')[0].split('_')[0]
    max_region_name = imagefile_path_list[max_val].split('/')[-1].split('.')[0].split('_')[0]
    return min_region_name , max_region_name

shape_list = ['shape file path']
image_list = ['shape file path']


minimum_region , maximum_region = compare_taluka(shape_list, image_list)
print("The Taluka with minimum variability is {}".format(minimum_region))
print("The Taluka with maximum variability is {}".format(maximum_region))
```

- **Objective 2**: Data Visualization.

- **Programming Language**: Python 3 or above.

- **Time Required**: 2 Hours

- **Prerequisites and Programming skill:**
  1. Python 3 or above should be installed on the computer.
  2. Student must have basic understanding of data visualization.

Data: Village level acreage and condition data in taluka for 2 years.

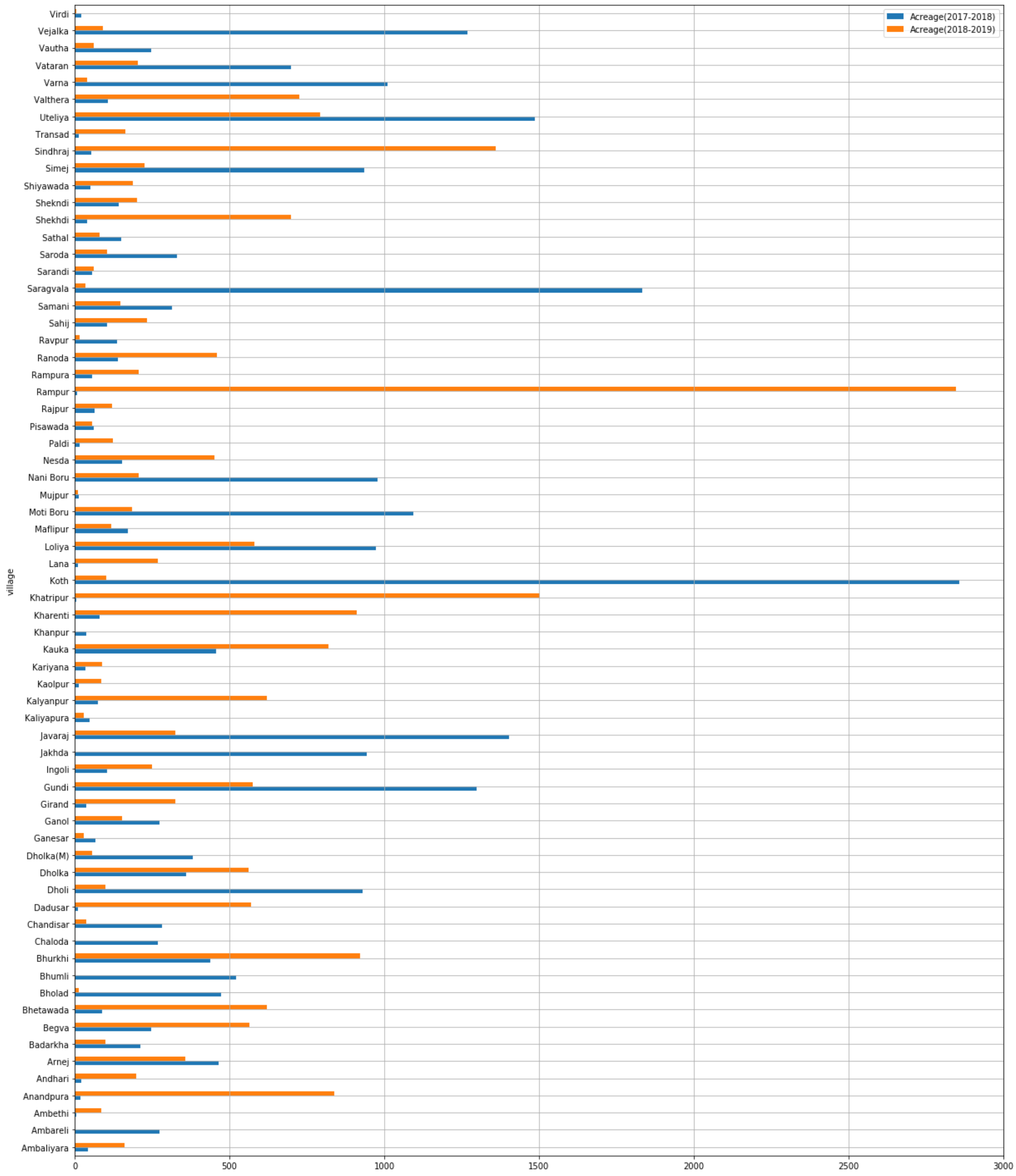
- 2.1 village level acreage and condition data in taluka for 2 years.
- 2.2 Prepare bar and column charts for both years and observe changes in acreage and health condition pattern.

Introduction: This practical will cover the Data Visualization part. Village level acreage data of two years will be visualized in charts. In addition, crop health condition pattern will also be visualized in the similar manner.

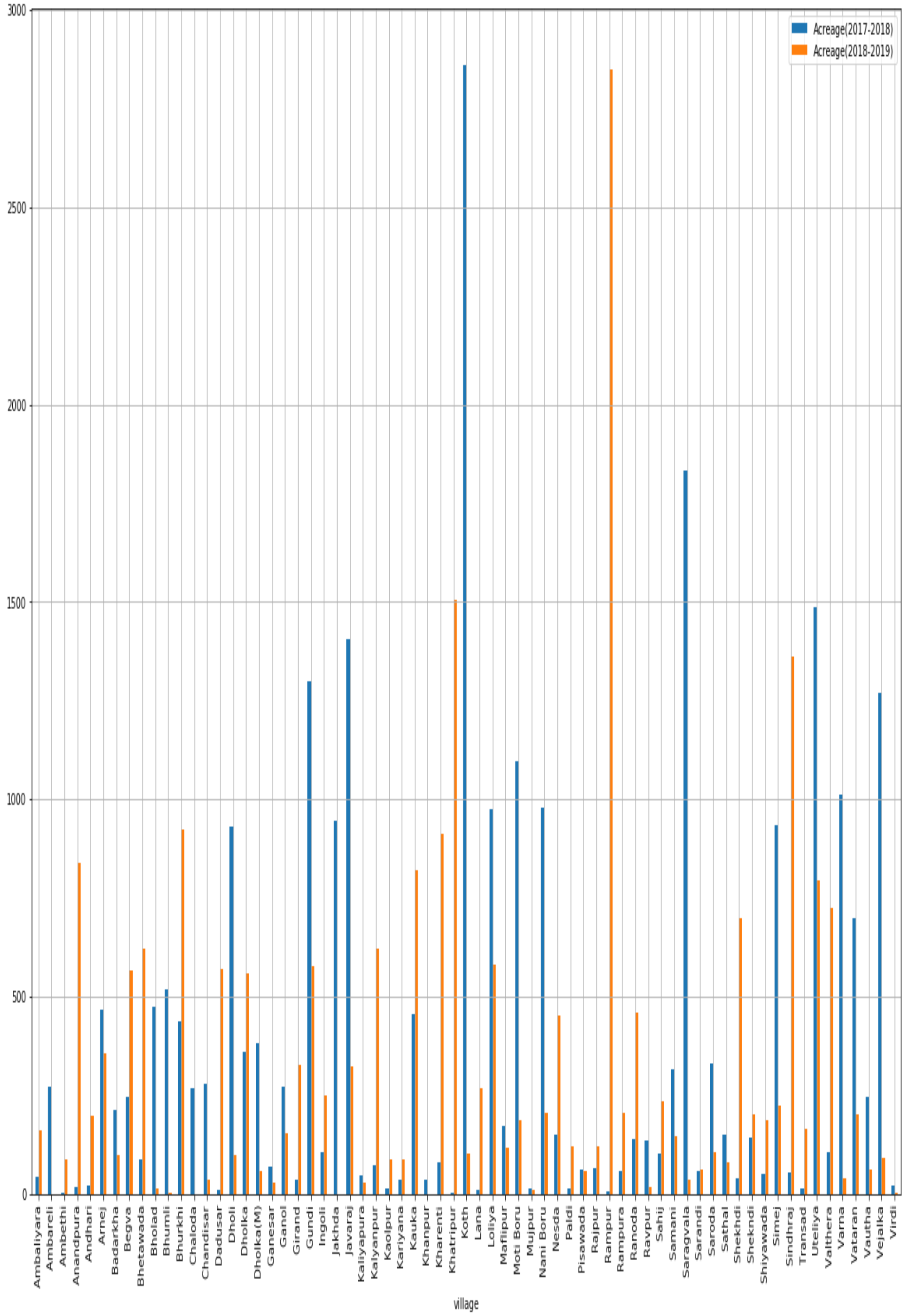
```
import matplotlib.pyplot as plt
import pandas as pd

from google.colab import drive
drive.mount('/content/gdrive', force_remount = True)

condition_data = pd.read_csv('./gdrive/My Drive/ColabNotebooks/condition_chart.csv')

condition_data.head()

# column chart for acreage:
condition_data[:].plot(x='village',y=['Acreage(2017-2018)', 'Acreage(2018-2019)'],figsize=(20,15),grid=True, kind = 'bar')

# bar chart for acreage:
condition_data[:].plot(x='village',y=['Acreage(2017-2018)', 'Acreage(2018-2019)'],figsize=(20,25),grid=True, kind = 'bar')

import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="darkgrid", color_codes=True)

ax = sns.countplot(x="Condition(2017-2018)", hue="Condition(2017-2018)", data=condition_data)

ax = sns.countplot(x="Condition(2018-2019)", hue="Condition(2018-2019)", data=condition_data)

ax = sns.countplot(x="Condition(2017-2018)",data=condition_data)

ax = sns.countplot(x="Condition(2018-2019)",data=condition_data)
```

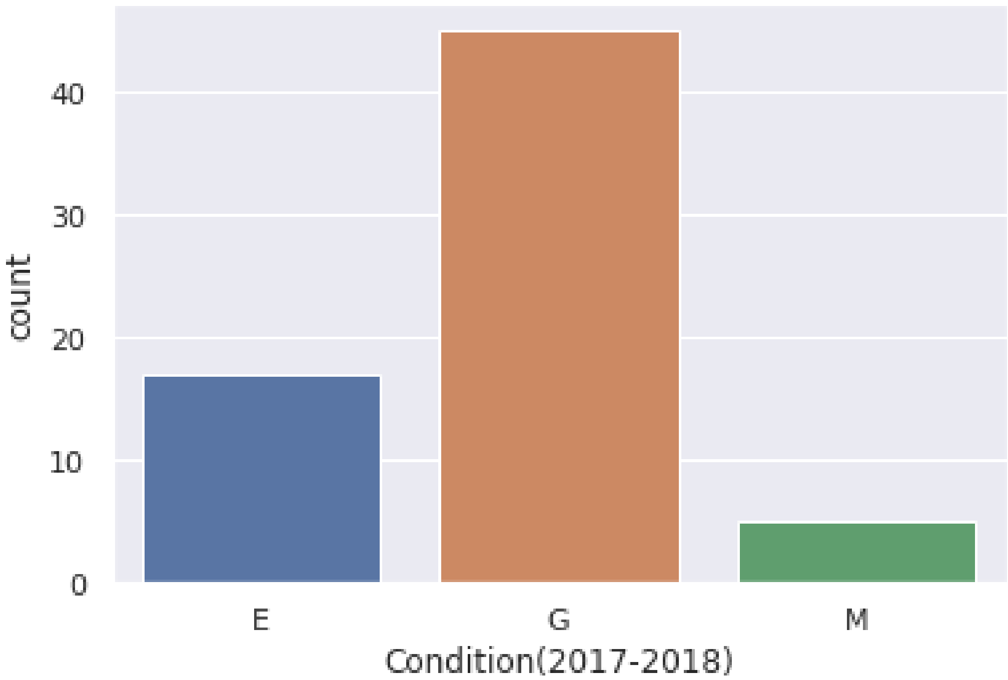
AMNEX

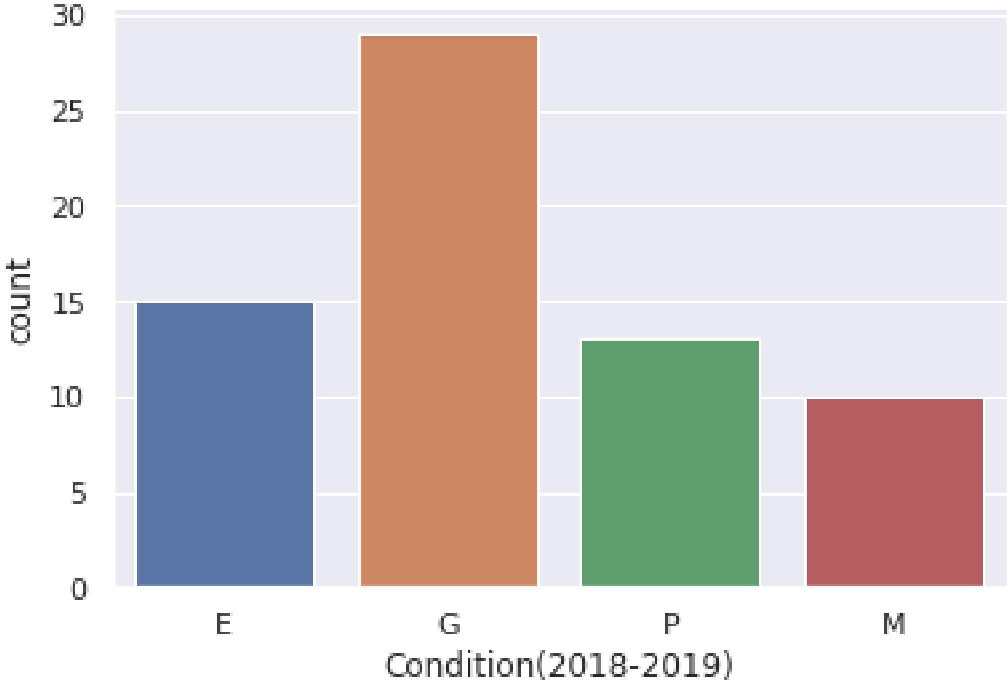
Output:(bar_chart)



Column_chart:




count
40
30
20
10
0
E
G
M
Condition(2017-2018)


count
30
25
20
15
10
5
0
E
G
P
M
Condition(2018-2019)

- **Objective 3**: **Linear Regression.**

- **Programming Language**: Python 3 or above.

- **Time Required**: 3 Hours

- **Prerequisites and Programming skill:**
  1. Python 3 or above should be installed on the computer.
  2. Student must have basic understanding of regression.

Data: Wheat and rice yield data for a district in Gujarat of 20 or more years corresponding Rainfall and temperature data.

- Study scatter-plots of rainfall vs. yield for rice and temp vs. yield for wheat.
- Prepare weekly, fortnightly rainfall and temp averages for different critical growth stages of wheat and rice.
- Perform MLR on the datasets and interpret regression coefficients, their significance, t-test, F-test.
- Use 75 % of data to develop regression model and predict yields for the rest 25% of the data, Study the errors and model performance in terms of mean absolute percentage error.

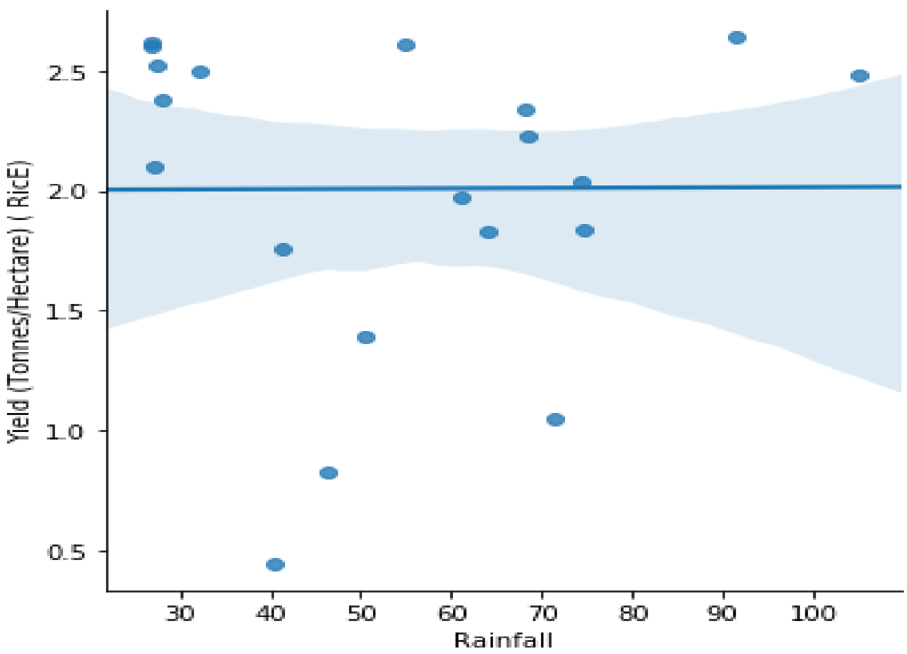
Introduction: These practical will deal with Linear Regression. Regression model will be prepared using 75% data and the yields will be predicted using remaining 25% data. The model's performance will be assessed subsequently in terms of absolute percentage error.

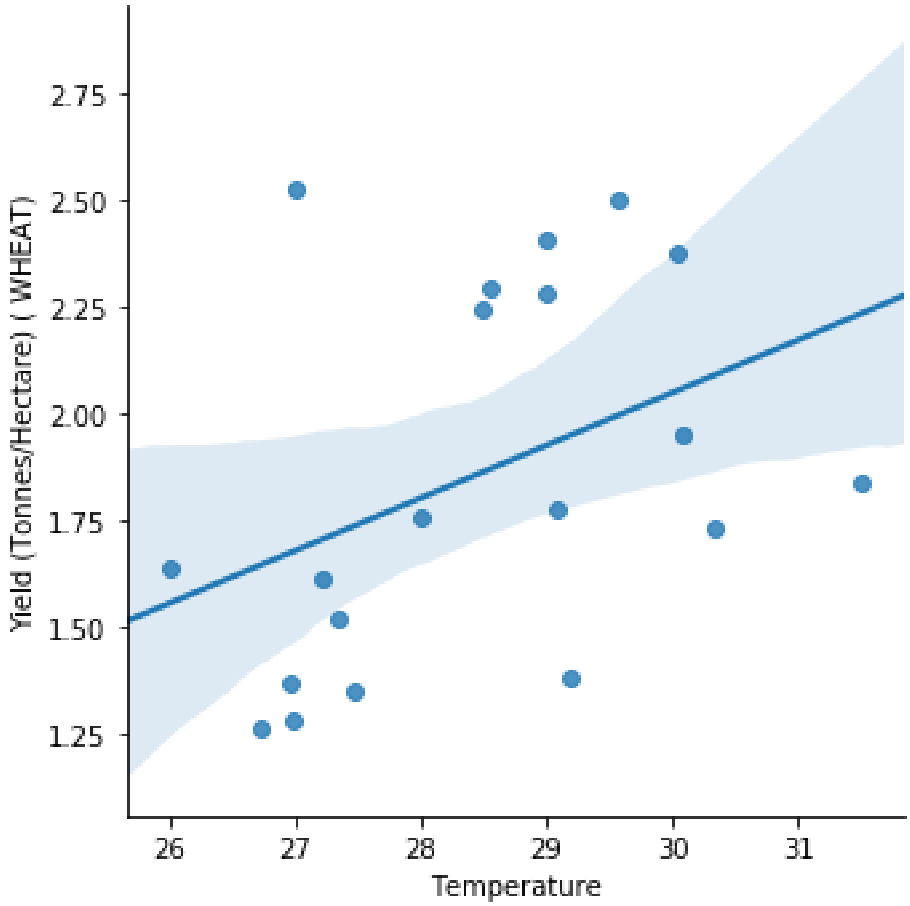
```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("./gdrive/My Drive/ColabNotebooks/Tutorial-3.csv")

df.plot()

import seaborn as sns
sns.lmplot(x='Rainfall', y='Yield (Tonnes/Hectare) ( RicE)', fit_reg=True, data=df);

sns.lmplot(x='Temperature', y='Yield (Tonnes/Hectare) ( WHEAT)', fit_reg=True, data=df);

"""MLR regression"""

# MLR:
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score

ml  = pd.read_csv("./gdrive/My Drive/ColabNotebooks/Tutorial-3.csv",   usecols = [2,3,4])#, unpack = True)

x = np.column_stack((ml["Rainfall"], ml["Temperature"]))
y = ml["Yield (Tonnes/Hectare) ( WHEAT)"]
# print("y",y)
# print("x", x)
# print("w", z)


X = np.column_stack((np.ones(len(x)),x))
X = np.matrix(X)
Y = np.row_stack(y)
#print(Y)
```

```python
XtX = (np.transpose(X)) * X
# print("XtX:",XtX)

XtY = (np.transpose(X)) * Y
# print("XtY:",XtY)


XtX_inv = (XtX).I
# print("xtx_inv:",XtX_inv)

B = XtX_inv * XtY
B = np.matrix((B))
# print("B:",B)

# n = 8
# m = 5
# S = B[0] + ((B[1])*x) + ((B[2])*m)
# print("S:",S)

Ycap = X * B
# print("ycap:",Ycap)
Ycap = np.matrix((Ycap))

error = Y - Ycap
# print("error:",error)

rmse = np.sqrt(mean_squared_error(Y,Ycap))
r2 = r2_score(Y,Ycap)

print("RMSE",rmse)
print("R2", r2)
```

```
"""Calculate Mean-absolute percentage error"""

# https://stats.stackexchange.com/questions/58391/mean-absolute-percentage-error-mape-in-scikit-learn
# from sklearn.utils import check_arrays
def mean_absolute_percentage_error(y_true, y_pred):
#     y_true, y_pred = check_arrays(y_true, y_pred)

    ## Note: does not handle mix 1d representation
    #if _is_1d(y_true):
    #    y_true, y_pred = _check_1d_array(y_true, y_pred)

    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

MAPE = mean_absolute_percentage_error(Y,Ycap)
print("MAPE", MAPE)

# https://towardsdatascience.com/inferential-statistics-series-t-test-using-numpy-2718f8f9bf2f
# t-test
from scipy import stats
t2, p2 = stats.ttest_ind(X,Y)
print("t-test", t2)

# https://stackoverflow.com/questions/28145938/f-test-with-p-value-in-python
# f-test:
d1 = ml["Yield (Tonnes/Hectare) ( WHEAT)"]
d2 = ml["Temperature"]
import statistics as stats
import scipy.stats as ss
def Ftest_pvalue(d1,d2):
    """docstring for Ftest_pvalue"""
    df1 = len(d1) - 1
    df2 = len(d2) - 1
    F = stats.variance(d1) / stats.variance(d2)
    single_tailed_pval = ss.f.cdf(F,df1,df2)
    double_tailed_pval = single_tailed_pval * 2
    return double_tailed_pval

print("F-test value:",Ftest_pvalue( d1,d2))
```

Output:





Yield (Tonnes/Hectare) (WHEAT)
2.75
2.50
2.25
2.00
1.75
1.50
1.25
26
27
28
29
30
31
Temperature

- **Objective 4**: Rainfall data analysis using Arima Analysis.

- **Programming Language**: Python 3 or above.

- **Time Required**: 1 Hours

- **Prerequisites and Programming skill:**

1. Python 3 or above should be installed on the computer.
2. Student must have basic understanding of deep learning.

- **Data:**

  Historical timeseries rainfall data

- **Introduction:**

  Auto regressive integrated moving average (ARIMA) models will be used to predict the weather parameter such as rainfall by using historical monthly and annual timeseries data.

1. Import libraries

```
import pandas as pd
import numpy as np

import statsmodels.api as sm
import statsmodels.tsa.api as smt
from statsmodels.tsa.stattools import adfuller
import statsmodels.formula.api as smf

import matplotlib.pyplot as plt
%matplotlib inline
import itertools

plt.style.use('bmh')
```

2.1. Preprocess the data ¶

```
data_matrix = pd.read_csv("D:/aRIMA/06-06/ARIMA-Latur-master/ARIMA-Latur-master/LaturRains_1965_2002.csv",sep="\t")
type(data_matrix)
```

```
data_matrix.set_index('Year', inplace=True)
data_matrix.tail()
```

2.2.1. Overall data plot

```
plt.figure(figsize=(13,7))
plt.plot(data_matrix)
plt.xlabel('Year')
plt.ylabel('Precipitation(mm)')
plt.title('Month vs Precipitation across all years')
```

```
plt.figure(figsize=(10,5))
# type(data_matrix)
plt.boxplot(data_matrix)
plt.xlabel('Month')
plt.ylabel('Precipitation(mm)')
plt.title('Month vs Precipitation across all years')
```

2.3. Convert the dataframe into series values

```
rainfall_data_matrix_np = data_matrix.transpose().as_matrix()
# rainfall_data_matrix_np.shape
shape = rainfall_data_matrix_np.shape
rainfall_data_matrix_np = rainfall_data_matrix_np.reshape((shape[0] * shape[1], 1))
rainfall_data_matrix_np.shape
```

Divide the data into Train and Test Sets

```
rainfall_data = pd.DataFrame({'Precipitation': rainfall_data_matrix_np[:,0]})
rainfall_data.set_index(dates, inplace=True)

test_data = rainfall_data.ix['1995': '2002']
train_data = rainfall_data.ix[: '1994']
# rainfall_data.head()
```

Applying Moving Average on different windows

```
fig, axes = plt.subplots(2, 2, sharey=False, sharex=False)
fig.set_figwidth(14)
fig.set_figheight(8)
axes[0][0].plot(rainfall_data.index, rainfall_data, label='Original')
axes[0][0].plot(rainfall_data.index, rainfall_data.rolling(window=4).mean(), label='4-Months Rolling Mean')
axes[0][0].set_xlabel("Years")
axes[0][0].set_ylabel("Precipitation in mm")
axes[0][0].set_title("4-Months Moving Average")
axes[0][0].legend(loc='best')
############
axes[0][1].plot(rainfall_data.index, rainfall_data, label='Original')
axes[0][1].plot(rainfall_data.index, rainfall_data.rolling(window=8).mean(), label='8-Months Rolling Mean')
axes[0][1].set_xlabel("Years")
axes[0][1].set_ylabel("Precipitation in mm")
axes[0][1].set_title("8-Months Moving Average")
axes[0][1].legend(loc='best')
############
axes[1][0].plot(rainfall_data.index, rainfall_data, label='Original')
axes[1][0].plot(rainfall_data.index, rainfall_data.rolling(window=12).mean(), label='12-Months Rolling Mean')
axes[1][0].set_xlabel("Years")
axes[1][0].set_ylabel("Precipitation in mm")
axes[1][0].set_title("12-Months Moving Average")
axes[1][0].legend(loc='best')
############
axes[1][1].plot(rainfall_data.index, rainfall_data, label='Original')
axes[1][1].plot(rainfall_data.index, rainfall_data.rolling(window=16).mean(), label='16-Months Rolling Mean')
axes[1][1].set_xlabel("Years")
axes[1][1].set_ylabel("Precipitation in mm")
axes[1][1].set_title("16-Months Moving Average")
axes[1][1].legend(loc='best')
# ############
plt.tight_layout()
plt.show()
```

Seasonality within a window of 12 months seems to more appealing from the above plots.

Let's plot the rolling mean and standard deviation on window of 12 months.

```
#Determing rolling statistics
rolmean = rainfall_data.rolling(window=12).mean()
rolstd = rainfall_data.rolling(window=12).std()

#Plot rolling statistics:
orig = plt.plot(rainfall_data, label='Original')
mean = plt.plot(rolmean, label='Rolling Mean')
std = plt.plot(rolstd, label = 'Rolling Std',color='green')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```

Let's run the Dicky Fuller Test on the timeseries

```
#dickey-fuller test
def adf_test(timeseries):
    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)

adf_test(rainfall_data.Precipitation)
```

Seasonality

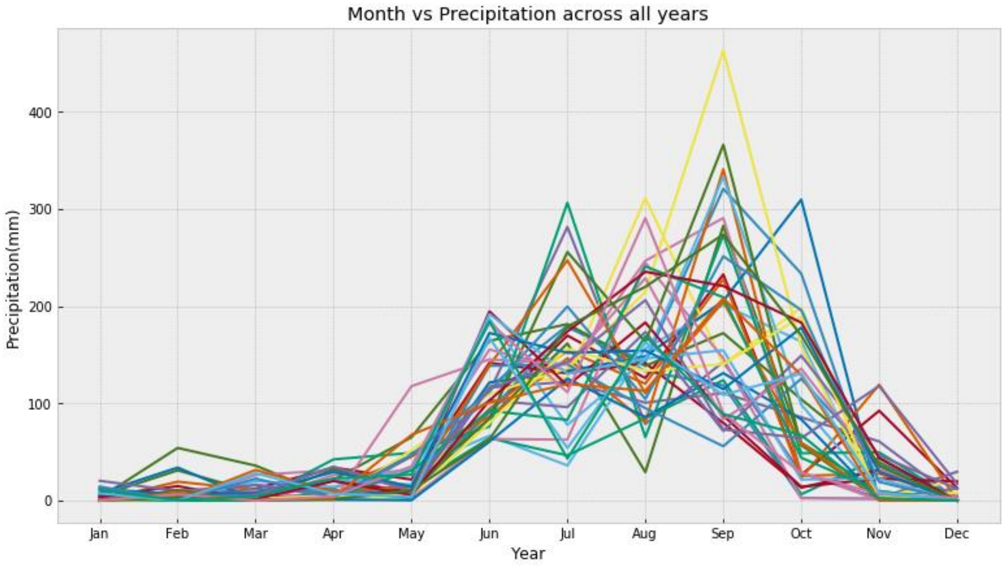
Seasonality effect could be seen from the below plot . Please refer to the Box plot shown in the beginning of the notebook for better inferences.

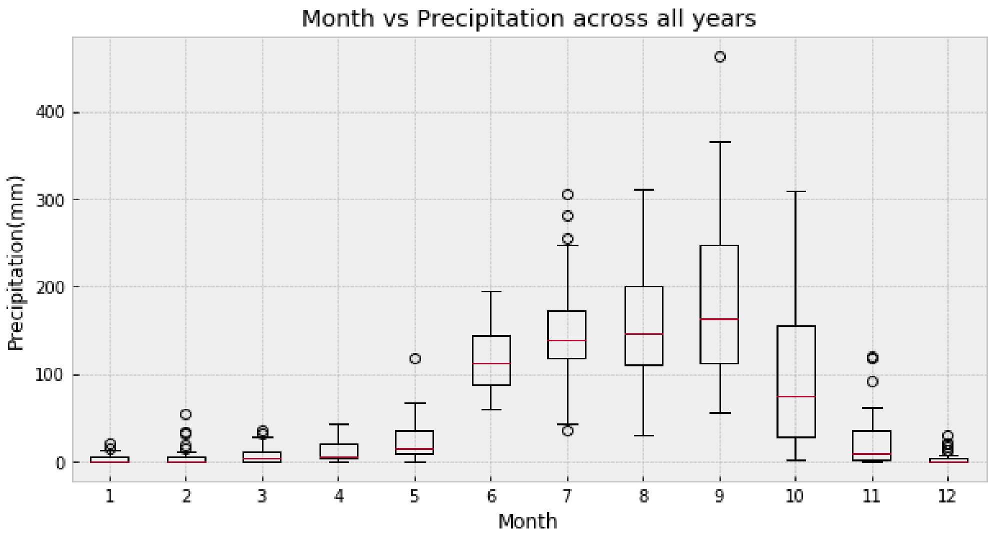
```
plt.figure(figsize=(13,7))
plt.plot(data_matrix)
plt.xlabel('Year')
plt.ylabel('Precipitation(mm)')
plt.title('Month vs Precipitation across all years')
```

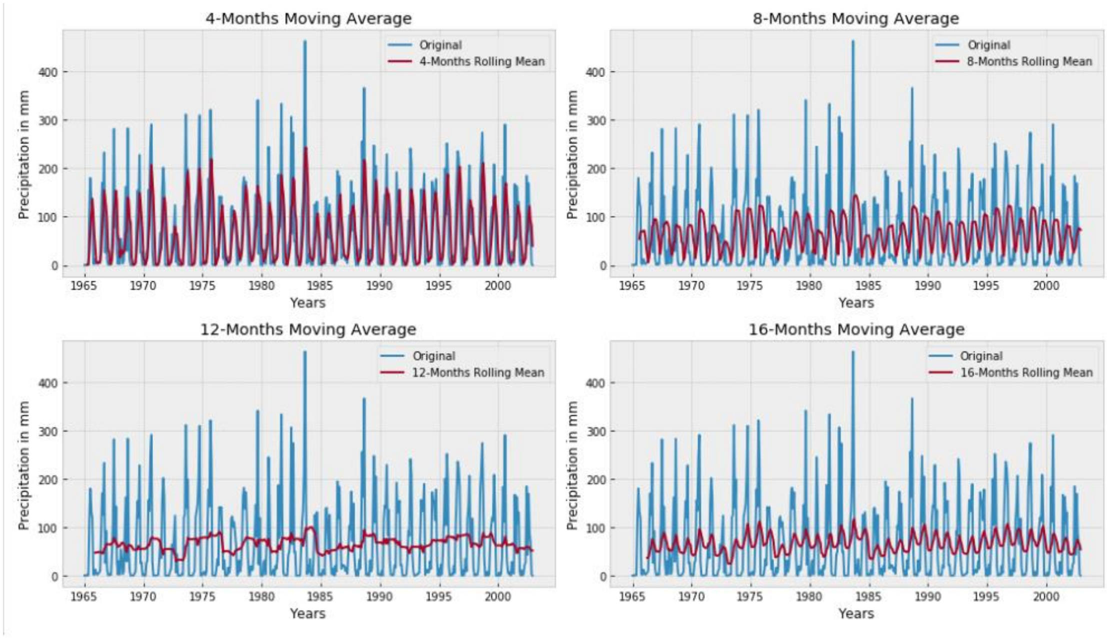
```
best_aic = np.inf
best_pdq = None
best_seasonal_pdq = None
temp_model = None

for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            temp_model = sm.tsa.statespace.SARIMAX(train_data,
                                                order = param,
                                                seasonal_order = param_seasonal,
                                                enforce_stationarity=True,
                                                enforce_invertibility=True)
            results = temp_model.fit()
            print("AIC for SARIMA{}x{}12 model - AIC:{}".format(param, param_seasonal, results.aic))
            if results.aic < best_aic:
                best_aic = results.aic
                best_pdq = param
                best_seasonal_pdq = param_seasonal
        except:
            continue

print("")
print("Best SARIMAX{}x{}12 model - AIC:{}".format(best_pdq, best_seasonal_pdq, best_aic))
```

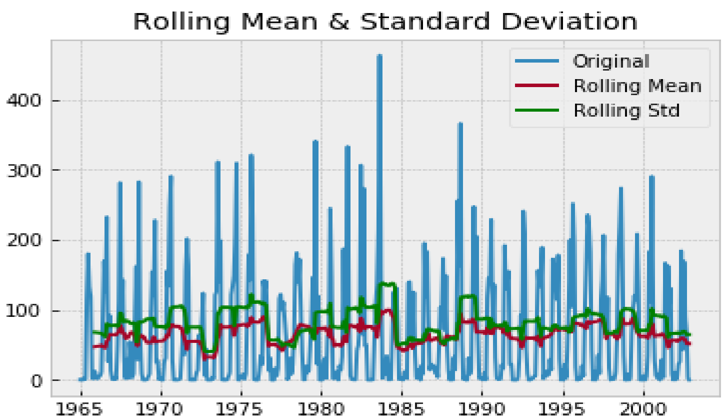
```
# rainfall_predicted_np = (list(np.array(rainfall_predicted)))
# print(rainfall_predicted_np)
# rainfall_truth_np = list(np.array(rainfall_truth))
# rainfall_predicted_np = rainfall_predicted_np.reshape(-1,1)
# rainfall_truth_np = rainfall_truth_np.reshape(-1,1)



# print(rainfall_predicted_np)
rainfall_predicted_np = (np.array(rainfall_predicted))
rainfall_truth_np = np.array(rainfall_truth)

# print len(rainfall_truth_np)

# print rainfall_predicted_np

num = abs(rainfall_predicted_np-rainfall_truth_np)
term = np.sum(num/ (1.0*rainfall_truth_np))
# print num
MAPE_error = (np.sum(abs(rainfall_predicted_np-rainfall_truth_np)/(1.0*rainfall_truth_np)))/(1.0*len(rainfall_predicted_np))


# y_true, y_pred = check_array(rainfall_predicted_np, rainfall_truth_np)
# print("MAPE : ",np.mean(np.abs((y_true - y_pred) / y_true)) * 100)
print (MAPE_error)
```

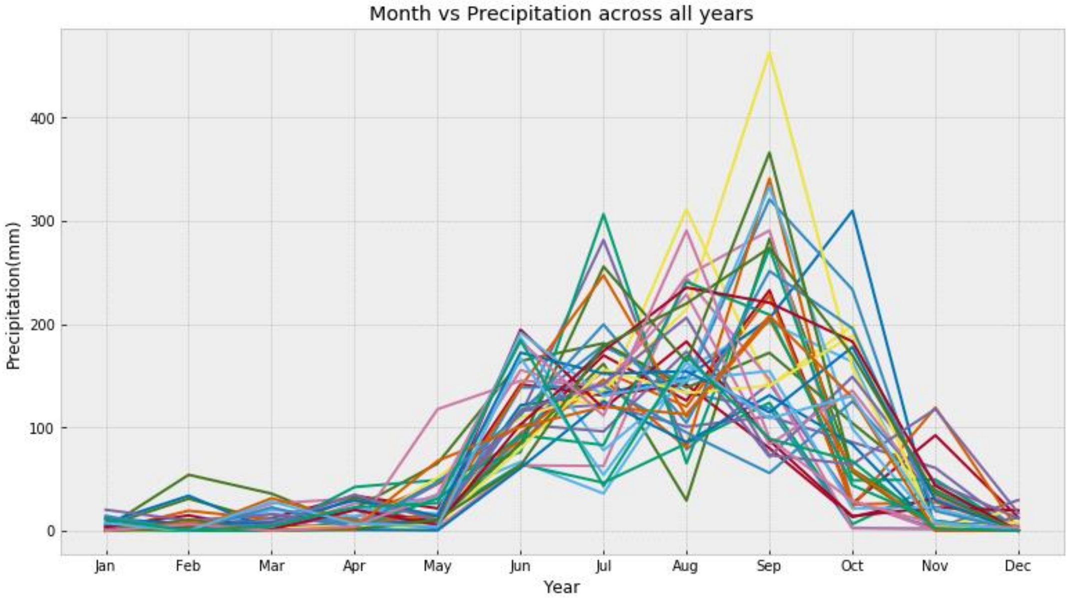
Output:

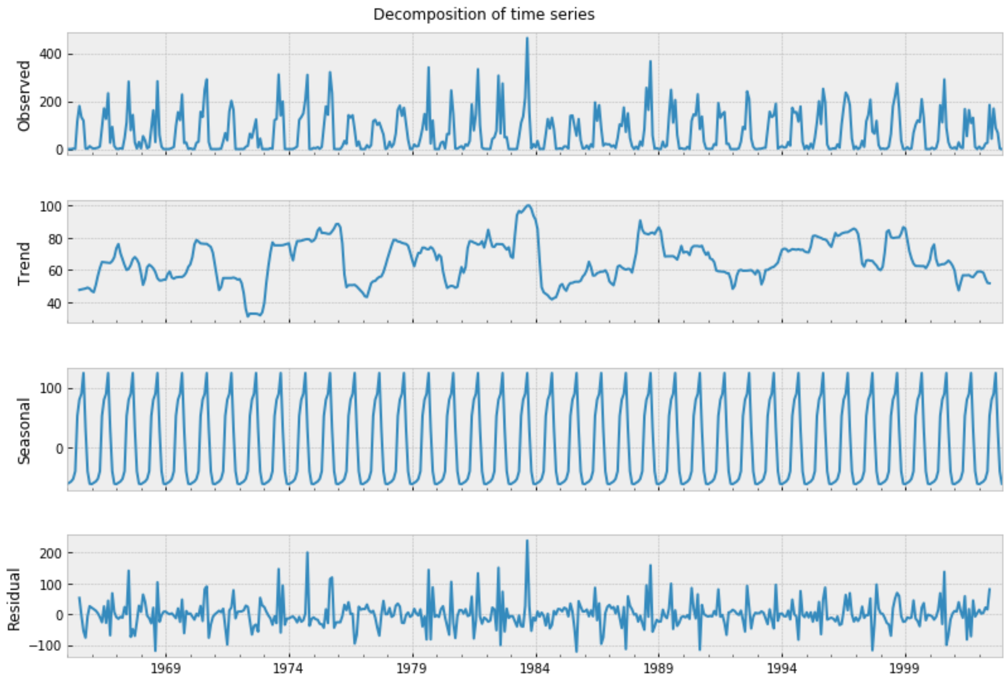

Month vs Precipitation across all years
Precipitation(mm)
Year
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec


Month vs Precipitation across all years
Precipitation(mm)
Month


4-Months Moving Average
Original
4-Months Rolling Mean
8-Months Moving Average
Original
8-Months Rolling Mean
12-Months Moving Average
Original
12-Months Rolling Mean
16-Months Moving Average
Original
16-Months Rolling Mean
Precipitation in mm
Years
0
100
200
300
400
1965
1970
1975
1980
1985
1990
1995
2000


Rolling Mean & Standard Deviation
Original
Rolling Mean
Rolling Std
0
100
200
300
400
1965
1970
1975
1980
1985
1990
1995
2000


Month vs Precipitation across all years
Precipitation(mm)
400
300
200
100
0
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Year


Decomposition of time series
Observed
Trend
Seasonal
Residual
1969
1974
1979
1984
1989
1994
1999

LSTM:

Data Preprocessing

```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
rainfall_data_matrix_np = data_matrix.transpose().as_matrix()

shape = rainfall_data_matrix_np.shape
rainfall_data_matrix_np = rainfall_data_matrix_np.reshape((shape[0] * shape[1], 1))
```

```
rainfall_data = pd.DataFrame({'Precipitation': rainfall_data_matrix_np[:,0]})
rainfall_data.set_index(dates, inplace=True)

plt.figure(figsize=(20,5))
plt.plot(rainfall_data, color='blue')
plt.xlabel('Year')
plt.ylabel('Precipitation(mm)')
plt.title('Precipitation in mm')
test_data = rainfall_data.ix['1995': '2002']
train_data = rainfall_data.ix[: '1994']
type(train_data)
```

```
plt.figure(figsize=(20,5))
plt.plot(rainfall_data, color='blue')
plt.xlabel('Year')
plt.ylabel('Precipitation(mm)')
plt.title('Precipitation data in mm of Latur from 1965-2002')
```

Building the LSTM Model

```
# Let's load the required libs.
# We'll be using the Tensorflow backend (default).
from keras.utils.vis_utils import plot_model
from keras.models import Sequential
from keras.layers.recurrent import LSTM
from keras.layers.core import Dense, Activation, Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.utils import shuffle
from keras.callbacks import LambdaCallback
```

Data Preparation

```
# Get the raw data values from the pandas data frame.
data_raw = rainfall_data.values.astype("float32")

scaler = MinMaxScaler(feature_range=(0,1))
dataset = scaler.fit_transform(data_raw)

# Print a few values.
dataset[0:5]
```

Split Data into Train and Test datasets

```
#using 80% of data to train amd 20% to test

TRAIN_SIZE = 0.8
train_size = int(len(dataset)*TRAIN_SIZE)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]
print (dataset.shape)
print (train.shape)
print (test.shape)
print("Number of entries (training set, test set): " + str((len(train), len(test))))
```

```
def create_dataset(dataset, window_size = 1):
    data_X, data_Y = [], []
    for i in range(len(dataset) - window_size - 1):
        a = dataset[i:(i + window_size), 0]
        data_X.append(a)
#         print("--",dataset[i+window_size,0],"--")
        data_Y.append(dataset[i + window_size, 0])
    return(np.array(data_X), np.array(data_Y))
```

```
# Create test and training sets for one-step-ahead regression.
window_size = 12
train_X, train_Y = create_dataset(train, window_size)
test_X, test_Y = create_dataset(test, window_size)
print("Original training data shape:")
# print(train_Y)
print(train_X.shape)
print(train_Y.shape)

# Reshape the input data into appropriate form for Keras.
train_X = np.reshape(train_X, (train_X.shape[0], 1, train_X.shape[1]))
test_X = np.reshape(test_X, (test_X.shape[0], 1, test_X.shape[1]))
train_Y = np.reshape(train_Y, (train_Y.shape[0], 1))
test_Y = np.reshape(test_Y, (test_Y.shape[0], 1))
print("New training data shape:")
print(train_X.shape)
print(train_Y.shape)
```

Build a simple LSTM

The LSTM architecture here consists of:

- One input layer.
- One LSTM layer of 4 blocks.
- One Dense layer to produce a single output.
- Use MSE as loss function.

Many different architectures could be considered. But this is just a quick test, so we'll keep things nice and simple.

```
def fit_model(train_X,train_Y,window_size=1):

    model = Sequential()
    model.add(LSTM(6,input_shape=(1,window_size)))
#     model.add(LSTM(6,input_shape=(1,window_size)))
    model.add(Dense(1))
#     print_weights = LambdaCallback(on_epoch_end=lambda batch, logs: print(model.layers[0].get_weights()))
    model.compile(loss='mean_squared_error',
                  optimizer='adam', metrics=['mape', 'accuracy'])
#     history = model.fit(train_X,train_Y,validation_data=(test_X, test_Y),epochs=250,batch_size=1,callbacks
    history = model.fit(train_X,train_Y,validation_data=(test_X, test_Y),epochs=250,batch_size=1,verbose=2)
    return model,history

#     on_epoch_end=lambda batch, logs: print (model.layers[1].get_weights())
#fit the model
model1, history = fit_model(train_X, train_Y, window_size)
plot_model(model1, to_file='LSTM_Latur_plot_1.png', show_shapes=True, show_layer_names=True)
model1.summary()
```

Predictions and model evaluation

```
train_Y.shape
```

```
(351, 1)
```

```
test_Y.shape
```

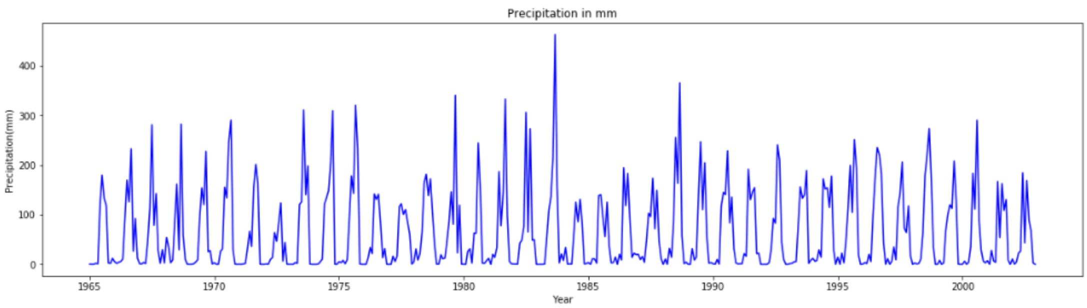
```
(79, 1)
```

```
def predict_and_score(model,X,Y):
    #Make predictions on the original scale of data
    pred = scaler.inverse_transform(model.predict(X))
    #Prepare Y also to be in original data scale
    orig_data = scaler.inverse_transform(Y)
#     print(orig_data)
#     print("-----")
#     print(pred[:,0])
    #Calculate RMSE
    score = math.sqrt(mean_squared_error(orig_data, pred[:, 0]))
    return (score,pred)

train_rmse, train_predict = predict_and_score(model1, train_X, train_Y)
test_rmse, test_predict = predict_and_score(model1, test_X, test_Y)

# train_rmse, train_predict = predict_and_score(model1, train_X, np.reshape(train_Y, (train_Y.shape[0], 1,1)))
# test_rmse, test_predict = predict_and_score(model1, test_X, np.reshape(test_Y, (test_Y.shape[0],1, 1)))

print("Training data score: %.2f RMSE" % train_rmse)
print("Test data score: %.2f RMSE" % test_rmse)
```

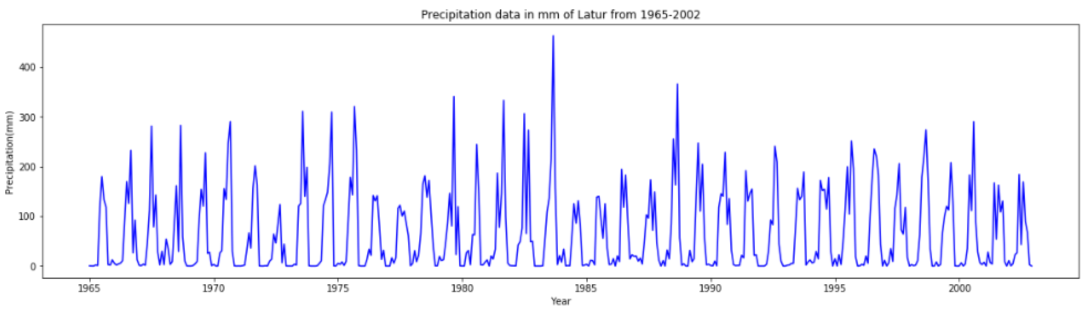
```python
# start with training predictions
train_predict_plot = np.empty_like(dataset)
train_predict_plot[:,:] = np.nan
train_predict_plot[window_size:len(train_predict) + window_size, :] = train_predict

# Add test predictions.
test_predict_plot = np.empty_like(dataset)
test_predict_plot[:, :] = np.nan
test_predict_plot[len(train_predict) + (window_size * 2) + 1:len(dataset) - 1, :] = test_predict
```

```python
# Create the plot.
plt.figure(figsize = (15, 5))
plt.plot(scaler.inverse_transform(dataset), label = "True value")
plt.plot(train_predict_plot, label = "Training set prediction")
plt.plot(test_predict_plot, label = "Test set prediction")
plt.xlabel("Months")
plt.ylabel("")
plt.title("Latur Rainfall Data Prediction")
plt.legend()
plt.show()
```

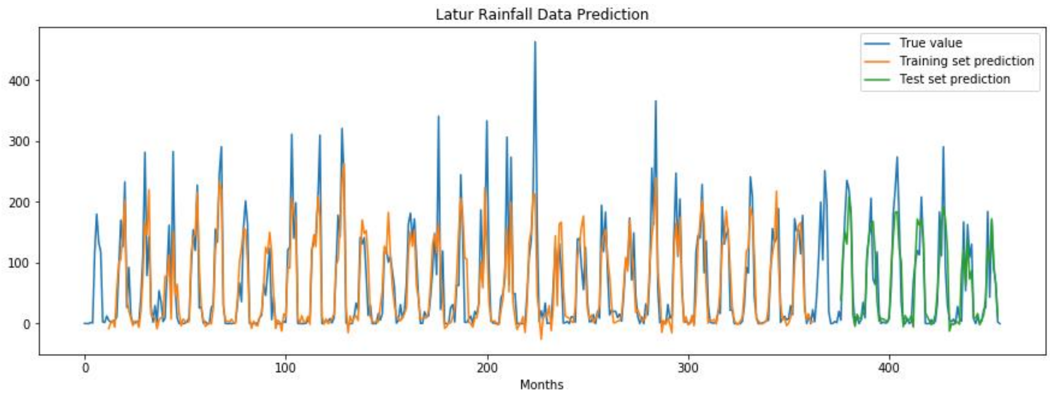
Output:




```
DatetimeIndex(['1965-01-01', '1965-02-01', '1965-03-01', '1965-04-01',
               '1965-05-01', '1965-06-01', '1965-07-01', '1965-08-01',
               '1965-09-01', '1965-10-01',
               ...
               '2002-03-01', '2002-04-01', '2002-05-01', '2002-06-01',
               '2002-07-01', '2002-08-01', '2002-09-01', '2002-10-01',
               '2002-11-01', '2002-12-01'],
              dtype='datetime64[ns]', length=456, freq='MS')
```



```
WARNING:tensorflow:From C:\Users\mihir.dakwala\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python\framewor
k\op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future v
ersion.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\mihir.dakwala\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\backend\tensorflow_b
ackend.py:1188: calling reduce_sum_v1 (from tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be removed
in a future version.
Instructions for updating:
keep_dims is deprecated, use keepdims instead
WARNING:tensorflow:From C:\Users\mihir.dakwala\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python\ops\math
_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 351 samples, validate on 79 samples
Epoch 1/250
2s - loss: 0.0330 - mean_absolute_percentage_error: 9853549.0617 - acc: 0.1111 - val_loss: 0.0184 - val_mean_absolute_percent
age_error: 7123052.0979 - val_acc: 0.1392
Epoch 2/250
```



- **Objective 5**: Time Series Analysis.

- **Programming Language**: Python 3 or above.

- **Time Required**: 3 Hours

- **Prerequisites and Programming skill:**
  1. Python 3 or above should be installed on the computer.
  2. Student must have basic understanding of regression.

Data: FAO India level wheat and rice data (1961-2017), n=56.

- 4.1 Plot time series for both crops.
- 4.2 Check for trend, seasonality in data.
- 4.3 If data is stationary, use moving average and smoothing methods for forecasting.

Introduction: In this practical, the participants will learn how to analyze the Time series data.

```
from google.colab import drive
drive.mount('/content/gdrive')

import matplotlib.pyplot as plt
import pandas as pd

wheat_data = pd.read_csv('./gdrive/My Drive/ColabNotebooks/rice_paddy.csv')

wheat_data.info()

wheat_data.head()

# wheat = wheat_data.groupby('Year')

wheat_data = wheat_data.set_index('Year')
wheat_data.index

y = wheat_data['Production in Tonne']

y.plot(figsize=(15, 6))
plt.ylabel("production in tonne")
plt.title("Produciton of wheat from 1961 - 2017")
plt.show()

import seaborn as sns

sns.set()

wheat_data.plot()
```

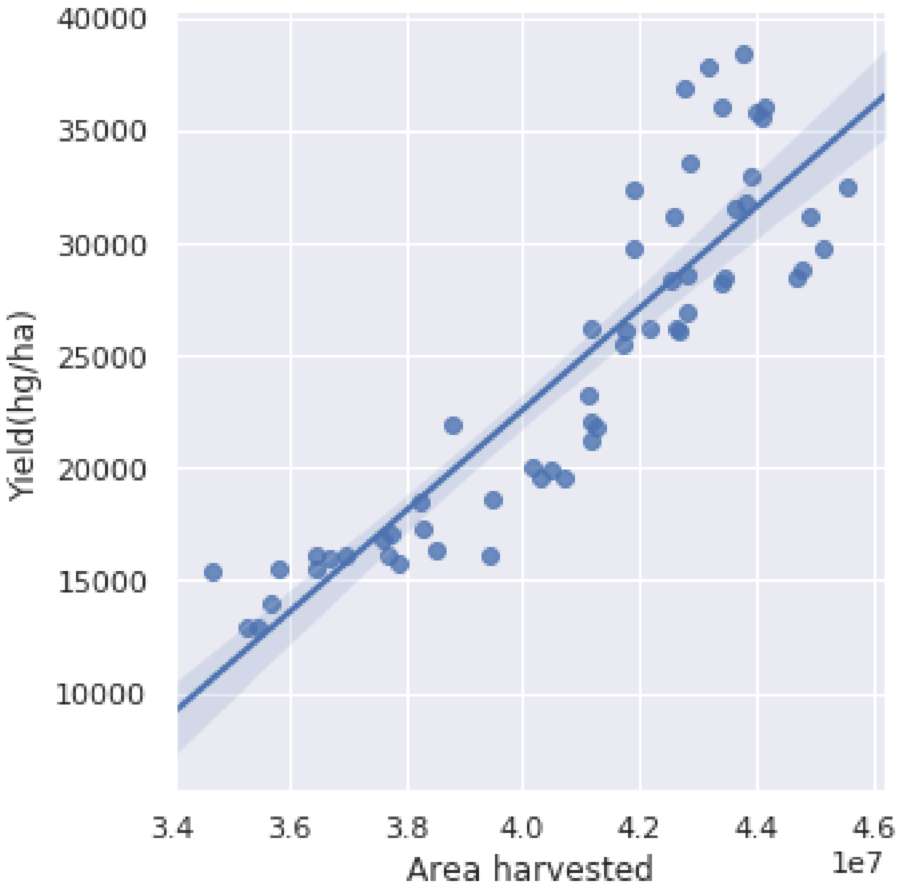
```python
"""Trend in Area Harvested and Production"""

area = wheat_data[['Area harvested']]
area.rolling(12).mean().plot(figsize=(20,10), linewidth=5, fontsize=20)
plt.xlabel('Year', fontsize=20)
plt.title("Area harvested")
plt.show()

production = wheat_data[['Production in Tonne']]
production.rolling(12).mean().plot(figsize=(20,10), linewidth=5, fontsize=20)
plt.xlabel('Year', fontsize=20)
plt.title("Production in Tonne")
plt.show()
```

```python
"""Seasonality in data"""

production.diff().plot(figsize=(10,5), linewidth=5, fontsize=20)
plt.xlabel('Year', fontsize=20)
plt.show()

area.diff().plot(figsize=(10,5), linewidth=5, fontsize=20)
plt.xlabel('Year', fontsize=20);

# co-relation Linear regression
sns.lmplot(x='Area harvested', y='Yield(hg/ha)', fit_reg=True, data=wheat_data);

#polynomial regression
sns.lmplot(x='Area harvested', y='Yield(hg/ha)', order = 4, fit_reg=True, data=wheat_data);

sns.lmplot(x='Area harvested', y='Production in Tonne', fit_reg=True, data=wheat_data);

sns.lmplot(x='Year Code', y='Production in Tonne', fit_reg=True, data=wheat_data);
```

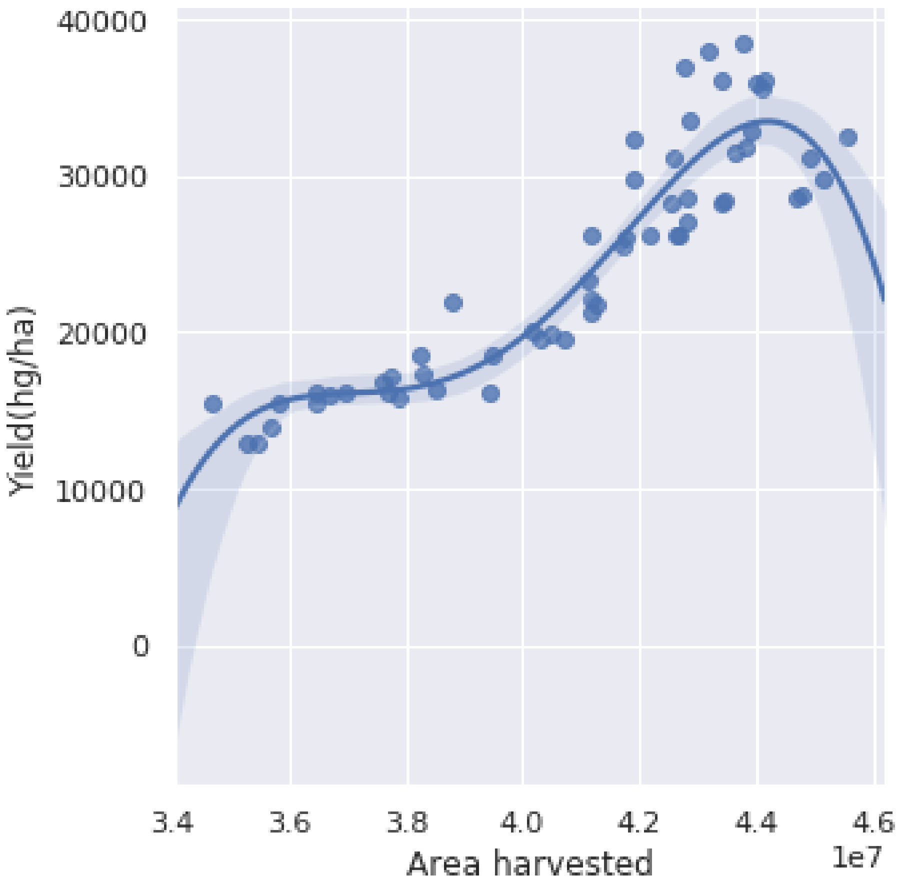
```
"""Moving Average"""

df=pd.DataFrame(wheat_data)
ts = pd.Series(df["Yield(hg/ha)"].values, index=df["Year Code"])
# print(ts.head(5))
mean_smoothed = ts.rolling(window=5).mean()
# print(mean_smoothed)
###### NEW #########
# mean_smoothed[0]=ts[0]
# mean_smoothed.interpolate(inplace=True)
####################
exp_smoothed = ts.ewm(alpha=0.5).mean()

h1 = ts.head(8)
h2 = mean_smoothed.head(8)
h3 = exp_smoothed.head(8)
k = pd.concat([h1, h2, h3], join='outer', axis=1)
k.columns = ["Actual", "Moving Average", "Exp Smoothing"]
print(k)


plt.figure(figsize=(16,5))
plt.plot(ts, label="Original")
plt.plot(mean_smoothed, label="Moving Average")
plt.plot(exp_smoothed, label="Exponentially Weighted Average")
plt.legend()
plt.show()
```

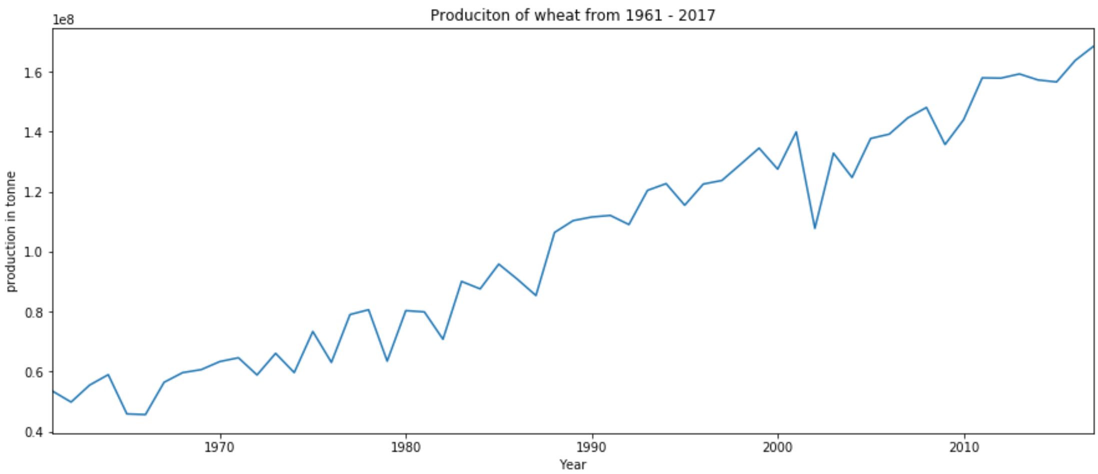
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures

# forecast error using polynomial regression

# transforming the data to include another axis
x = wheat_data['Area harvested'][:, np.newaxis]
y = wheat_data['Yield(hg/ha)'][:, np.newaxis]

polynomial_features= PolynomialFeatures(degree=2)
x_poly = polynomial_features.fit_transform(x)

model = LinearRegression()
model.fit(x_poly, y)
y_poly_pred = model.predict(x_poly)

rmse = np.sqrt(mean_squared_error(y,y_poly_pred))
r2 = r2_score(y,y_poly_pred)
print("RMSE",rmse)
print("R2",r2)
```

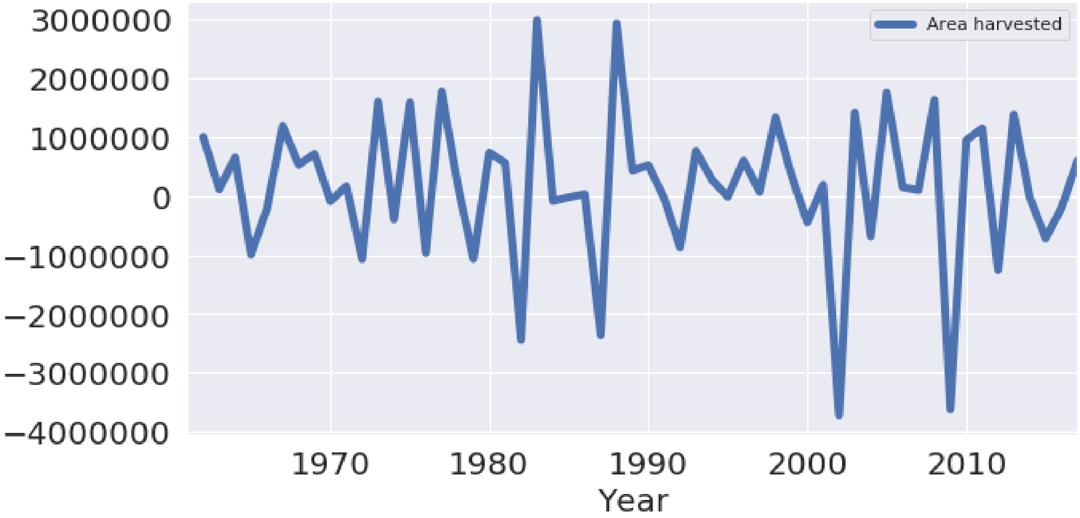
Output:

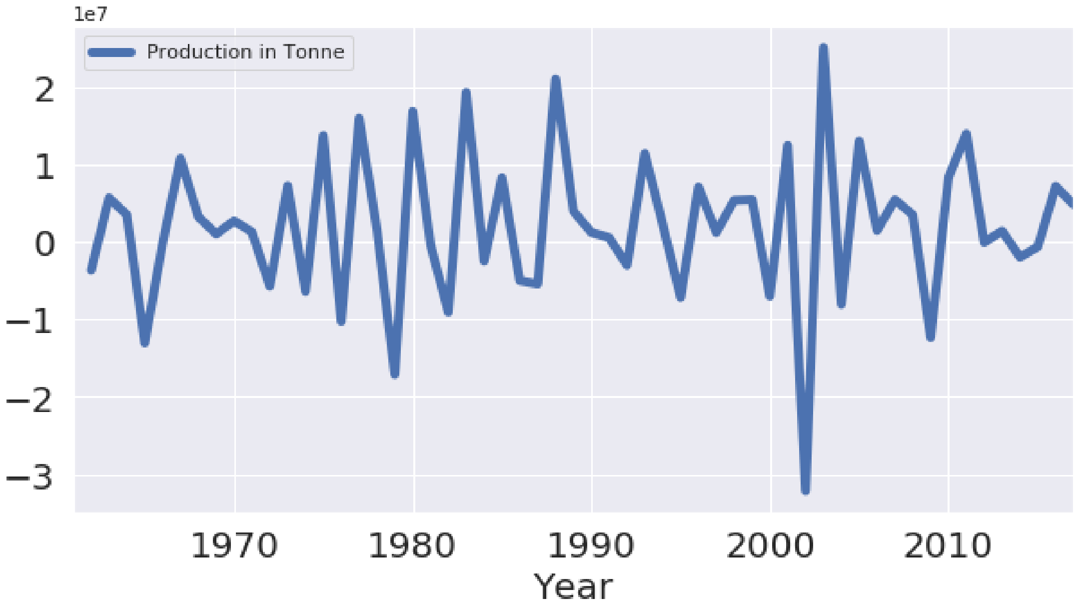

40000
35000
30000
25000
20000
15000
10000
Yield(hg/ha)
3.4
3.6
3.8
4.0
4.2
4.4
4.6
Area harvested
1e7

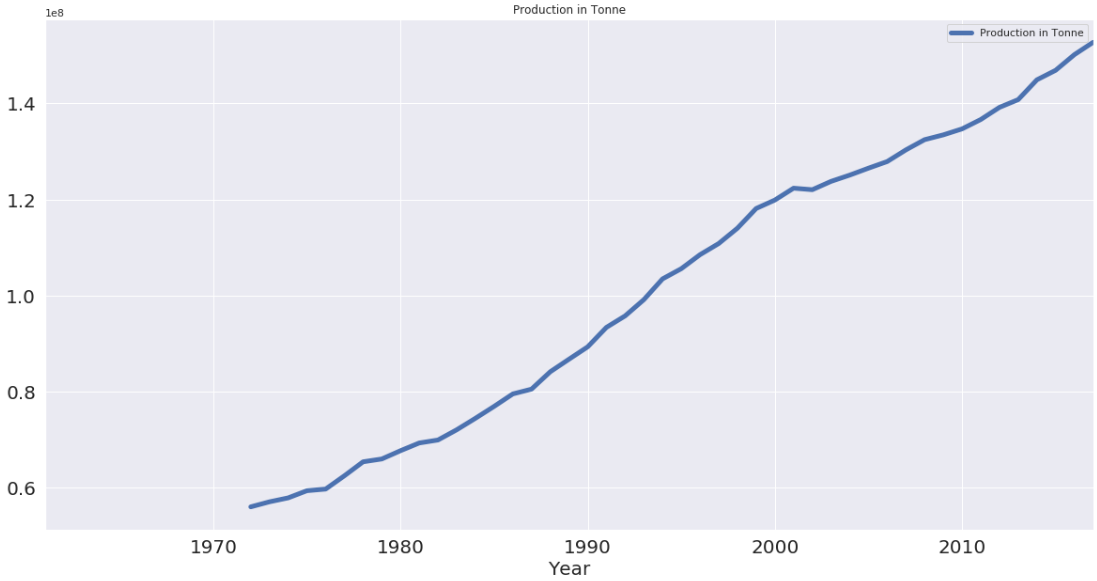

Original
Moving Average
Exponentially Weighted Average
35000
30000
25000
20000
15000
1960
1970
1980
1990
2000
2010

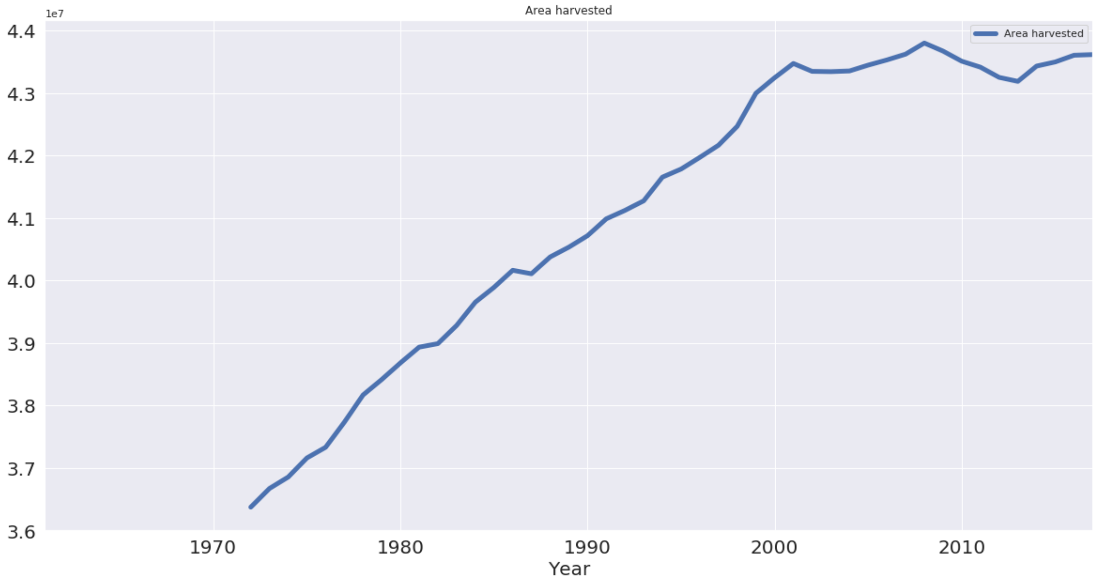

40000
30000
20000
10000
0
Yield(hg/ha)
3.4
3.6
3.8
4.0
4.2
4.4
4.6
1e7
Area harvested


Produciton of wheat from 1961 - 2017
1e8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
production in tonne
1970
1980
1990
2000
2010
Year


3000000
2000000
1000000
0
−1000000
−2000000
−3000000
−4000000
1970
1980
1990
2000
2010
Year
Area harvested


1e7
2
1
0
−1
−2
−3
1970
1980
1990
2000
2010
Year
Production in Tonne


Production in Tonne
1e8
Production in Tonne
1.4
1.2
1.0
0.8
0.6
1970
1980
1990
2000
2010
Year


Area harvested
1e7
Area harvested
4.4
4.3
4.2
4.1
4.0
3.9
3.8
3.7
3.6
1970
1980
1990
2000
2010
Year

- **Objective 6**: Data Mining.

- **Programming Language**: Python 3 or above.

- **Time Required**: 3 Hours

- **Prerequisites and Programming skill:**
  1. Python 3 or above should be installed on the computer.
  2. Student must have basic understanding of clustering and classification.

Data: satellite data for a district, village level crop data.

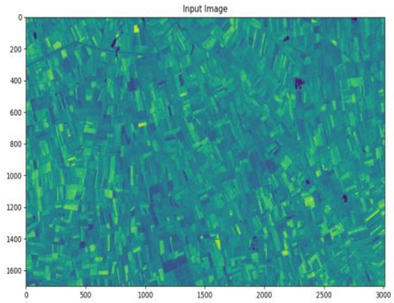
Introduction: This practical will cover crop classification tasks. Different classifiers will be used and their performance will be assessed on the basis of accuracy.

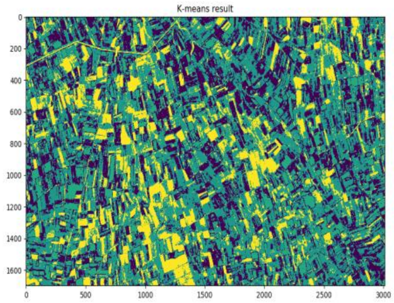
- 5.1 Classify satellite data using k-means clustering algorithm and supervised MXL, and NN classifier and compare classification accuracy.

```python
## Kmeans clustering
import cv2, numpy as np
import gdal,matplotlib.pyplot as plt
import statistics as st
from scipy import ndimage
import matplotlib.pyplot as plt
from sklearn import cluster
from sklearn.metrics import silhouette_score

#%%
k = 4 # number of clusters
## input image
out = gdal.Open('rabi_17_02_2019_v1.img')
out = out.ReadAsArray()
out= np.moveaxis(out,0,-1)
## TODO: Check your results with different bands
## or by taking the weighted average of all the bands
out = out[:,:,3] ##taking only 3channels

x, y = out.shape
image_2d = out.reshape(x*y,1)
image_2d.shape

kmeans_cluster = cluster.KMeans(n_clusters = k)
kmeans_cluster.fit(image_2d)
pred = kmeans_cluster.fit_predict(image_2d)
cluster_centers = kmeans_cluster.cluster_centers_
cluster_labels = kmeans_cluster.labels_
#%%
# compare your results with the ground truth data
## y_test are the actual label points in the ground truth image
#score = metrics.accuracy_score(y_test,k_means.predict())
#print('Accuracy:{0:f}'.format(score))
```

```python
plt.figure(1)
plt.subplot(211)
plt.title('Input Image')
plt.imshow(out)
plt.subplot(212)
plt.imshow(np.uint8(cluster_centers[cluster_labels].reshape(x, y)))
plt.title('K-means result')
```

Output:






```python
## SVM Supervised Learning

import gdal
import ogr
from sklearn import metrics
from sklearn import svm
import numpy as np
import pandas as pd
from timeit import default_timer as timer
from sklearn.metrics import cohen_kappa_score
import matplotlib.pyplot as plt



def rasterizeVector(path_to_vector, cols, rows, geo_transform, projection, n_class, raster):
    lblRaster = np.zeros((rows, cols))
    inputDS = ogr.Open(path_to_vector)
    driver = gdal.GetDriverByName('MEM')
    # Define spatial reference
    for j in range(n_class):
        shpLayer = inputDS.GetLayer(0)
        class_id = j + 1
        rasterDS = driver.Create('', cols, rows, 1, gdal.GDT_UInt16)
        rasterDS.SetGeoTransform(geo_transform)
        rasterDS.SetProjection(projection)
        shpLayer.SetAttributeFilter("Id = " + str(class_id))
        bnd = rasterDS.GetRasterBand(1)
        bnd.FlushCache()
        gdal.RasterizeLayer(rasterDS, [1], shpLayer, burn_values=[class_id])
        arr = bnd.ReadAsArray()
        lblRaster += arr
        rasterDS = None
        save_raster = gdal.GetDriverByName('GTiff').Create(raster, cols, rows, 1, gdal.GDT_UInt16)
        sband = save_raster.GetRasterBand(1)
        sband.WriteArray(lblRaster)
        sband.FlushCache()
    return lblRaster
```

```python
def createGeotiff(outRaster, data, geo_transform, projection, dtyp, bcount=1):
    # Create a GeoTIFF file with the given data
    driver = gdal.GetDriverByName('GTiff')
    rows, cols, _ = data.shape
    rasterDS = driver.Create(outRaster, cols, rows, bcount, dtyp)
    rasterDS.SetGeoTransform(geo_transform)
    rasterDS.SetProjection(projection)
    for i in range(bcount):
        band = rasterDS.GetRasterBand(i + 1)
        band.WriteArray(data[:, :, i])
        band.FlushCache()
    return 0


def check_accuracy(actual_labels, predicted_labels, label_count):
    error_matrix = np.zeros((label_count, label_count))
    for actual, predicted in zip(actual_labels, predicted_labels):
        error_matrix[int(actual) - 1][int(predicted) - 1] += 1
    return error_matrix
```

```python
start = timer()

inpRaster = r"Input/Image/1.tif"
outRaster = r"Output/svm/SVM.tif"
out_prob = r"Output/svm/Probability_Map.tif"

# Open raster dataset
rasterDS = gdal.Open(inpRaster, gdal.GA_ReadOnly)
# Get spatial reference
geo_transform = rasterDS.GetGeoTransform()
projection = rasterDS.GetProjectionRef()


# Extract band's data and transform into a numpy array
bandsData = []
for b in range(rasterDS.RasterCount):
    band = rasterDS.GetRasterBand(b + 1)
    band_arr = band.ReadAsArray()
    bandsData.append(band_arr)
bandsData = np.dstack(bandsData)
cols, rows, noBands = bandsData.shape
```

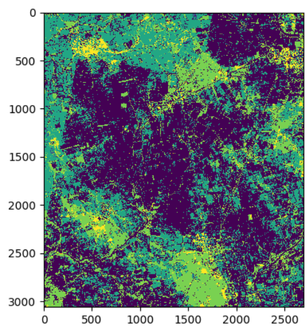
```
# Read vector data, and rasterize all the vectors in the given directory into a single labelled raster
shapefile = r"Input/Shapefile/Training_site.shp"
shapefile_test = r"Input/Shapefile/testing.shp"
rasterized_shp = r"Output/svm/Rasterized.tif"
rasterized_shp_test = r"Output/svm/Rasterized_test.tif"
lblRaster = rasterizeVector(shapefile, rows, cols, geo_transform, projection, n_class=6, raster=rasterized_shp)
lblRaster_test = rasterizeVector(shapefile_test,rows,cols,geo_transform,projection,n_class=6,raster=rasterized_shp_test)

print('Vectors Rasterized to Raster!')

# Prepare training data (set of pixels used for training) and labels
isTrain = np.nonzero(lblRaster)
isTest = np.nonzero(lblRaster_test)
trainingLabels = lblRaster[isTrain]
testingLabels = lblRaster_test[isTest]
trainingData = bandsData[isTrain]
testingData = bandsData[isTest]



# Train SVM Classifier
classifier = svm.SVC(C=100,gamma=0.1,kernel='linear',probability=True,random_state=None,shrinking=True,verbose=False)

classifier.fit(trainingData, trainingLabels)

print('Classifier fitting done!')
```

```
# Predict class label of unknown pixels
noSamples = rows * cols
flat_pixels = bandsData.reshape((noSamples, noBands))
result = classifier.predict(flat_pixels)
p_vals = classifier.predict_proba(flat_pixels)
predicted_labels = classifier.predict(trainingData)
lbl_cnt = (np.unique(trainingLabels)).size
df = pd.DataFrame(check_accuracy(trainingLabels, predicted_labels, 6))
df.to_csv('Output/svm/CM.csv', index=False)

score_oa = classifier.score(trainingData, trainingLabels)
print('training set OA:', score_oa)
score_oa_test = classifier.score(testingData, testingLabels)
print('testing set OA:', score_oa_test)

predicted_labels_test = classifier.predict(testingData)
test_lbl_cnt = (np.unique(testingLabels)).size

print('Testing Labels: ',np.unique(testingLabels))
print('Predicted Labels: ', np.unique(predicted_labels_test))

df_test = pd.DataFrame(check_accuracy(testingLabels, predicted_labels_test, 6))
df_test.to_csv('Output/svm/CM_test.csv', index=False)

print('Confusion Matrices Created!')
```

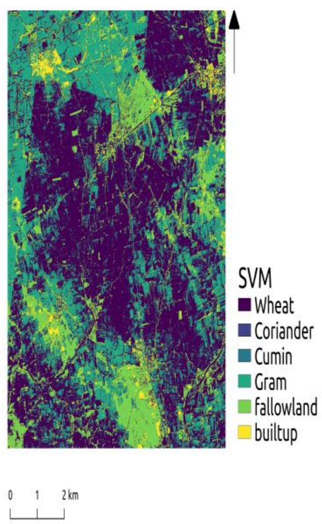
```python
###kappa value=======
kappa_score = cohen_kappa_score(trainingLabels, predicted_labels)
print('kappa value training: ', kappa_score)
kappa_score_test = cohen_kappa_score(testingLabels, predicted_labels_test)
print('kappa value testing: ', kappa_score_test)

b_count = p_vals.shape[1]

classification = result.reshape((cols, rows, 1))
prob_arr = p_vals.reshape((cols, rows, b_count))

# Create a GeoTIFF file with the given data
createGeotiff(outRaster, classification, geo_transform, projection, gdal.GDT_UInt16)
createGeotiff(out_prob, prob_arr, geo_transform, projection, gdal.GDT_Float32, b_count)

print('Classified Tiff Image created!')
img = plt.imread('Output/svm/SVM.tif')
plt.imshow(img)
plt.show()

end = timer()
print('The process took: ', end - start, ' seconds!')
```

Output:

```
ispluser@AIPLDES304Ubuntu:~/PycharmProjects/tutorial_classification$ python svm.py
Vectors Rasterized to Raster!
Classifier fitting done!
('training set OA:', 0.749080348499516)
('testing set OA:', 0.5245033112582781)
('Testing Labels: ', array([1., 2., 3., 4., 5., 6.]))
('Predicted Labels: ', array([1., 3., 4., 5., 6.]))
Confusion Matrices Created!
('kappa value training: ', 0.6320961413910198)
('kappa value testing: ', 0.3363956351385869 5)
Classified Tiff Image created!
('The process took: ', 468.4833040237427, ' seconds!')
ispluser@AIPLDES304Ubuntu:~/PycharmProjects/tutorial_classification$
```



SVM classified imagery



```python
#Random Forest

import gdal
import ogr
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
import numpy as np
import pandas as pd
from timeit import default_timer as timer
from sklearn.metrics import cohen_kappa_score
import matplotlib.pyplot as plt


def rasterizeVector(path_to_vector, cols, rows, geo_transform, projection, n_class, raster):
    lblRaster = np.zeros((rows, cols))
    inputDS = ogr.Open(path_to_vector)
    driver = gdal.GetDriverByName('MEM')
    # Define spatial reference
    for j in range(n_class):
        shpLayer = inputDS.GetLayer(0)
        class_id = j + 1
        rasterDS = driver.Create('', cols, rows, 1, gdal.GDT_UInt16)
        rasterDS.SetGeoTransform(geo_transform)
        rasterDS.SetProjection(projection)
        shpLayer.SetAttributeFilter("Id = " + str(class_id))
        bnd = rasterDS.GetRasterBand(1)
        bnd.FlushCache()
        gdal.RasterizeLayer(rasterDS, [1], shpLayer, burn_values=[class_id])
        arr = bnd.ReadAsArray()
        lblRaster += arr
        rasterDS = None
        save_raster = gdal.GetDriverByName('GTiff').Create(raster, cols, rows, 1, gdal.GDT_UInt16)
        sband = save_raster.GetRasterBand(1)
        sband.WriteArray(lblRaster)
        sband.FlushCache()
    return lblRaster
```

```python
def createGeotiff(outRaster, data, geo_transform, projection, dtype, bcount=1):
    # Create a GeoTIFF file with the given data
    driver = gdal.GetDriverByName('GTiff')
    rows, cols, _ = data.shape
    rasterDS = driver.Create(outRaster, cols, rows, bcount, dtyp)
    rasterDS.SetGeoTransform(geo_transform)
    rasterDS.SetProjection(projection)
    for i in range(bcount):
        band = rasterDS.GetRasterBand(i + 1)
        band.WriteArray(data[:, :, i])
        band.FlushCache()
    return 0


def check_accuracy(actual_labels, predicted_labels, label_count):
    error_matrix = np.zeros((label_count, label_count))
    for actual, predicted in zip(actual_labels, predicted_labels):
        error_matrix[int(actual) - 1][int(predicted) - 1] += 1
    return error_matrix


start = timer()

inpRaster = r"Input/Image/1.tif"
outRaster = r"Output/rf/rf.tif"
out_prob = r"Output/rf/Probability_Map.tif"

# Open raster dataset
rasterDS = gdal.Open(inpRaster, gdal.GA_ReadOnly)
# Get spatial reference
geo_transform = rasterDS.GetGeoTransform()
projection = rasterDS.GetProjectionRef()
```

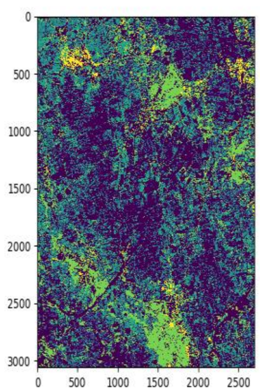
```python
# Extract band's data and transform into a numpy array
bandsData = []
for b in range(rasterDS.RasterCount):
    band = rasterDS.GetRasterBand(b + 1)
    band_arr = band.ReadAsArray()
    bandsData.append(band_arr)
bandsData = np.dstack(bandsData)
cols, rows, noBands = bandsData.shape

# Read vector data, and rasterize all the vectors in the given directory into a single labelled raster
shapefile = r"Input/Shapefile/Training_site.shp"
shapefile_test = r"Input/Shapefile/testing.shp"
rasterized_shp = r"Output/rf/Rasterized.tif"
rasterized_shp_test = r"Output/rf/Rasterized_test.tif"
lblRaster = rasterizeVector(shapefile, rows, cols, geo_transform, projection, n_class=6, raster=rasterized_shp)
lblRaster_test = rasterizeVector(shapefile_test,rows,cols,geo_transform,projection,n_class=6,raster=rasterized_shp_test)

print('Vectors Rasterized to Raster!')

# Prepare training data (set of pixels used for training) and labels
isTrain = np.nonzero(lblRaster)
isTest = np.nonzero(lblRaster_test)
trainingLabels = lblRaster[isTrain]
testingLabels = lblRaster_test[isTest]
trainingData = bandsData[isTrain]
testingData = bandsData[isTest]
```

```python
# Train RF Classifier
classifier = RandomForestClassifier(n_jobs=10, n_estimators=100, criterion='gini', oob_score= True, max_features= 2)

classifier.fit(trainingData, trainingLabels)

print('Classifier fitting done!')

# Predict class label of unknown pixels
noSamples = rows * cols
flat_pixels = bandsData.reshape((noSamples, noBands))
result = classifier.predict(flat_pixels)
p_vals = classifier.predict_proba(flat_pixels)
print("OOB Score: ", classifier.oob_score_)
predicted_labels = classifier.predict(trainingData)
lbl_cnt = (np.unique(trainingLabels)).size
df = pd.DataFrame(check_accuracy(trainingLabels, predicted_labels, 6))
df.to_csv('Output/rf/CM.csv', index=False)

score_oa = classifier.score(trainingData, trainingLabels)
print('training set OA:', score_oa)
score_oa_test = classifier.score(testingData, testingLabels)
print('testing set OA:', score_oa_test)

predicted_labels_test = classifier.predict(testingData)
test_lbl_cnt = (np.unique(testingLabels)).size

print('Testing Labels: ',np.unique(testingLabels))
print('Predicted Labels: ', np.unique(predicted_labels_test))

df_test = pd.DataFrame(check_accuracy(testingLabels, predicted_labels_test, 6))
df_test.to_csv('Output/rf/CM_test.csv', index=False)

print('Confusion Matrices Created!')
```

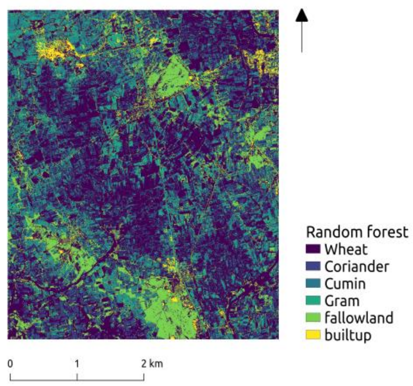
```python
###kappa value=======
kappa_score = cohen_kappa_score(trainingLabels, predicted_labels)
print('kappa value training: ', kappa_score)
kappa_score_test = cohen_kappa_score(testingLabels, predicted_labels_test)
print('kappa value testing: ', kappa_score_test)

b_count = p_vals.shape[1]

classification = result.reshape((cols, rows, 1))
prob_arr = p_vals.reshape((cols, rows, b_count))

# Create a GeoTIFF file with the given data
createGeotiff(outRaster, classification, geo_transform, projection, gdal.GDT_UInt16)
createGeotiff(out_prob, prob_arr, geo_transform, projection, gdal.GDT_Float32, b_count)

print('Classified Tiff Image created!')


img = plt.imread('Output/rf/rf.tif')
plt.imshow(img)
plt.show()

end = timer()
print('The process took: ', end - start, ' seconds!')
```

Output:

```
ispluser@AIPLDES304Ubuntu:~/PycharmProjects/tutorial_classification$ python rf.py
Vectors Rasterized to Raster!
Classifier fitting done!
('OOB Score: ', 0.8607938044530493)
('training set OA:', 1.0)
('testing set OA:', 0.48741721854304637)
('Testing Labels: ', array([1., 2., 3., 4., 5., 6.]))
('Predicted Labels: ', array([1., 2., 3., 4., 5., 6.]))
Confusion Matrices Created!
('kappa value training: ', 1.0)
('kappa value testing: ', 0.31446195091965035)
Classified Tiff Image created!
('The process took: ', 765.2242529392242, ' seconds!')
ispluser@AIPLDES304Ubuntu:~/PycharmProjects/tutorial_classification$
```



Random Forest classified imagery



➢ **Objective 7**: Data Mining.

- 6.1 Simple Spreadsheet model of exponential light absorption vs LAI.
- 6.2 Light absorption (R)=R0*exp(-k*LAI).
- 6.3 Given values of R0 and LAI, vary k (extinction coefficient) and study effect on transmission rate.
- 6.4 Sample values of LAI from a normal distribution and repeat exercise.

➢ **Programming Language**: Python 3 or above.

➢ **Time Required**: 3 Hours

➢ **Prerequisites and Programming skill:**

1. Python 3 or above should be installed on the computer.
2. Student must have basic understanding of exponential.

Data: satellite data for a district, village level crop data.

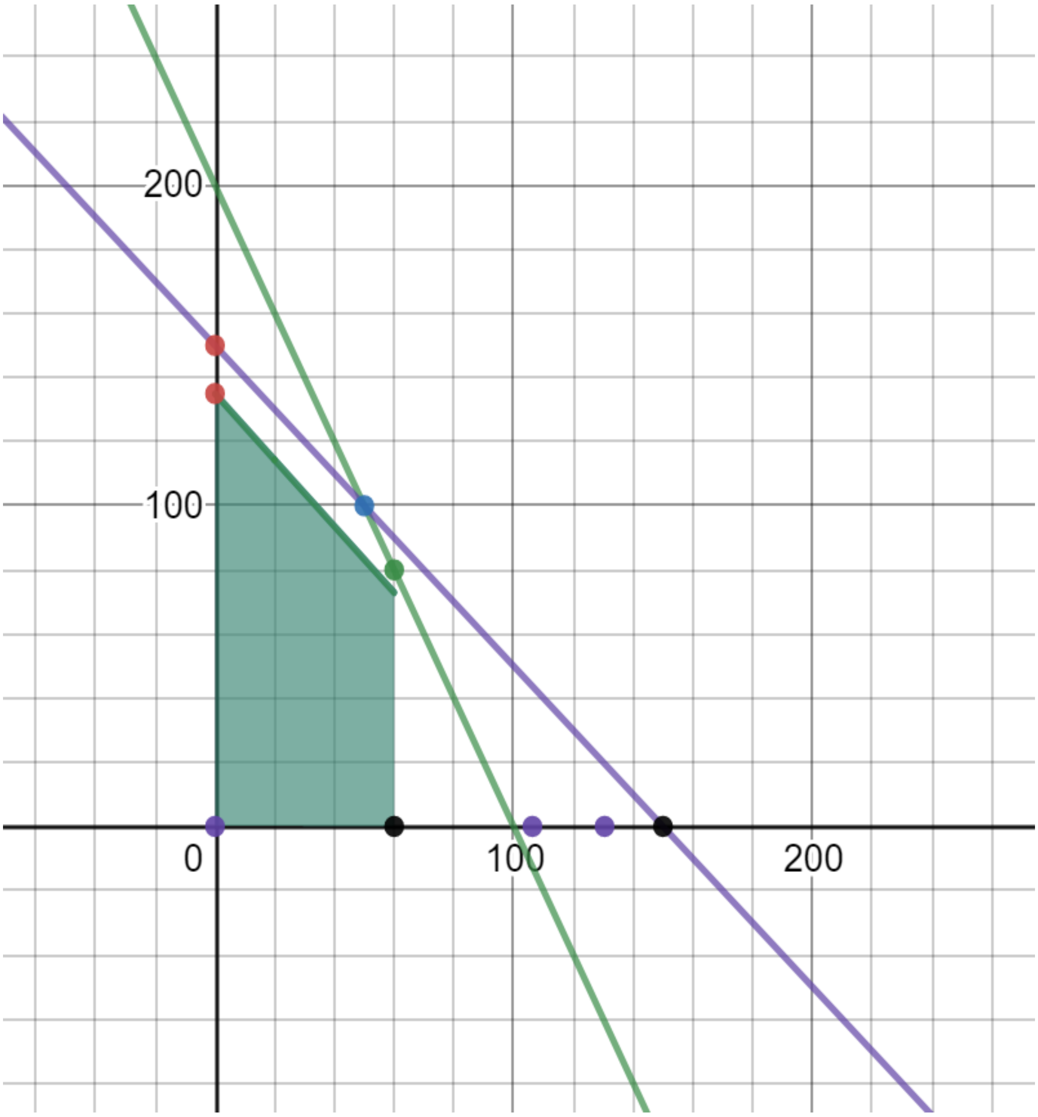
Introduction: Leaf Area Index (LAI) is an important factor in vegetation related studies. Its relation with Light absorption is crucial. This practical will focus on the relation between LAI and Light absorption at different k (extinction coefficient) values.

- **Objective 8**: Data Mining.
- **Programming Language**: Python 3 or above.
- **Time Required**: 1 Hours
- **Prerequisites and Programming skill:**
  1. Python 3 or above should be installed on the computer.
  2. Student must have basic understanding of simplex method.

Data: P=207x + 200 y Subject to x>=o y>=0 x+y<=150 x<=60 30x+15y<=3000.

7.1 Since this is two variable problem, solve it graphically.

Introduction: This practical will cover Linear Algebra related problem. An equation will be given and will be solved graphically. In addition, its application in the field of image processing will be explained.



7.2 Use simplex method to solve the same.

```python
from scipy.optimize import linprog
import numpy as np
# A_ub=None, b_ub=None, A_eq=None, b_eq=None, bounds=None, method='simplex',
# callback=None, options={'maxiter': 1000, 'disp': False, 'tol': 1e-12, 'bland': False}
c = [-207, -200]
A = [[1, 1], [30, 15]]
b = [150, 3000]
x1_bnds = (0, 60)
x0_bnds = (0, np.inf)

res = linprog(c,A_ub=A, b_ub=b, bounds=(x0_bnds, x1_bnds),method='simplex'
              ,options={'maxiter':1000, 'disp':True, 'tol':1e-12,'bland':False})
print(res)
```

Output:

```
Optimization terminated successfully.
         Current function value: -26490.000000
         Iterations: 2
     fun: -26490.0
 message: 'Optimization terminated successfully.'
     nit: 2
   slack: array([20.,  0.,  0.])
  status: 0
 success: True
       x: array([70., 60.])
```