Weather Analytics Course Content

Theory Session

WEEK 1

1. Meteorology as a physics of atmosphere
2. Elements of weather and climate,
3. Observational techniques
4. Extreme weather events and their prediction (Floods, droughts, heat wave, cold wave and frost etc.)

WEEK 2

1. Crops-weather relationship
2. Photosysnthesis and respiration
3. Growing degree days and phenology of crops
4. Crop water requirement- evapotranspiration estimation

WEEK 3

1. Crop yield predictions
2. Correlation and regression techniques
3. Multiple and curvilinear regression techniques
4. Simulation approach

WEEK 4

1. Climate variability and change
2. Impact of climate change on crops
3. Weather based agro-advisory service
4. Application of remote sensing and GIS

Ref book: **Practical Meteorology** - An Algebra-based Survey of Atmospheric Science **by** ROLAND STULL

Tutorial Session

Lab 1 : To download and import weather data. Descriptive analysis of temperature data.

- To download weather data collected from AWS situated in different cities
- Extraction of required data from file (e.g. Temperature)
- Removing erroneous points in data
- Hourly data to daily data and daily to monthly data generation
- Understanding and statistics of data and plotting of data

Practical:

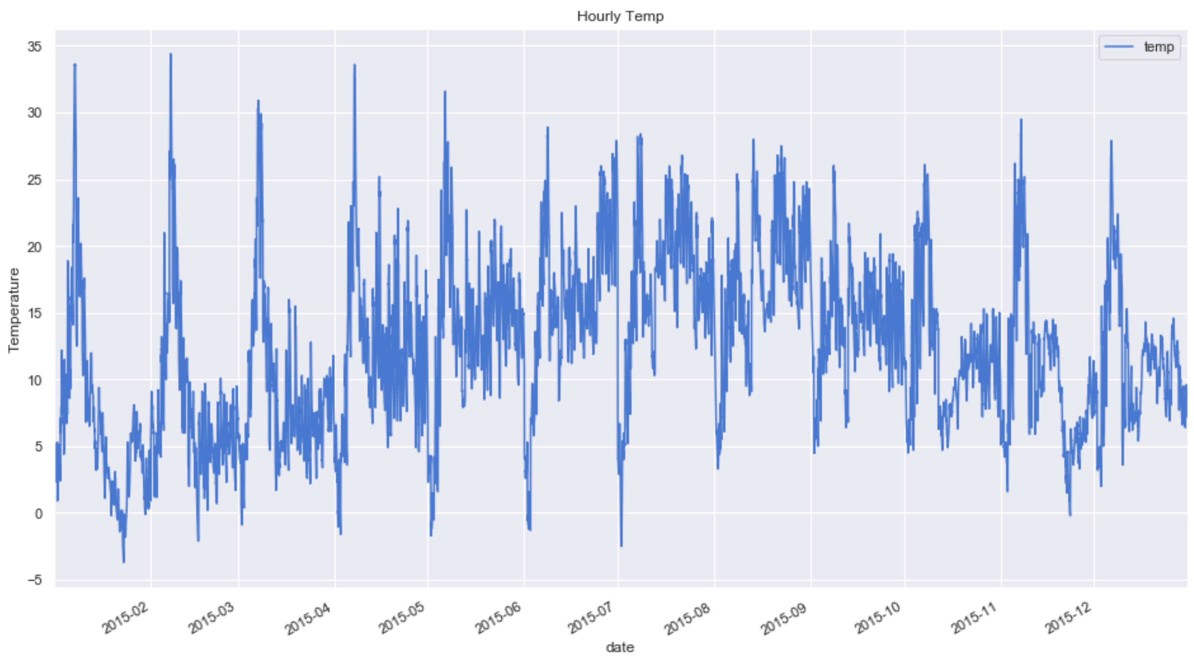
```python
import pandas as pd
from dateutil import parser, rrule
from datetime import datetime, time, date
import datetime as tm
import calendar
import matplotlib.pyplot as plt
import seaborn as sns

station = 'INDIA105' # Delhi
data_raw = pd.read_csv('{}_weather.csv'.format(station))

# Give the variables some friendlier names and convert types as necessary.
data_raw['temp'] = data_raw['TemperatureC'].astype(float)
data_raw['rain'] = data_raw['HourlyPrecipMM'].astype(float)
data_raw['total_rain'] = data_raw['dailyrainMM'].astype(float)
data_raw['date'] = data_raw['DateUTC'].apply(parser.parse)
data_raw['humidity'] = data_raw['Humidity'].astype(float)
data_raw['wind_direction'] = data_raw['WindDirectionDegrees']
data_raw['wind'] = data_raw['WindSpeedKMH']


# Extract out only the data we need.
data = data_raw.loc[:, ['date', 'temp']]
data = data[(data['date'] >= datetime(2015,1,1)) & (data['date'] <= datetime(2015,12,31))]


# There's an issue with some stations that record temperature ~-2500 where data is missing.
if (data['temp'] < -10).sum() > 0:
    print("There is messed up days for {}".format(station))


# remove the bad samples
data = data[data['temp'] > -50]


# Assign the "day" to every date entry
data['day'] = data['date'].apply(lambda x: x.date())


# Get the time, day, and hour of each timestamp in the dataset
data['time_of_day'] = data['date'].apply(lambda x: x.time())
data['day_of_week'] = data['date'].apply(lambda x: x.weekday())
data['hour_of_day'] = data['time_of_day'].apply(lambda x: x.hour)

```

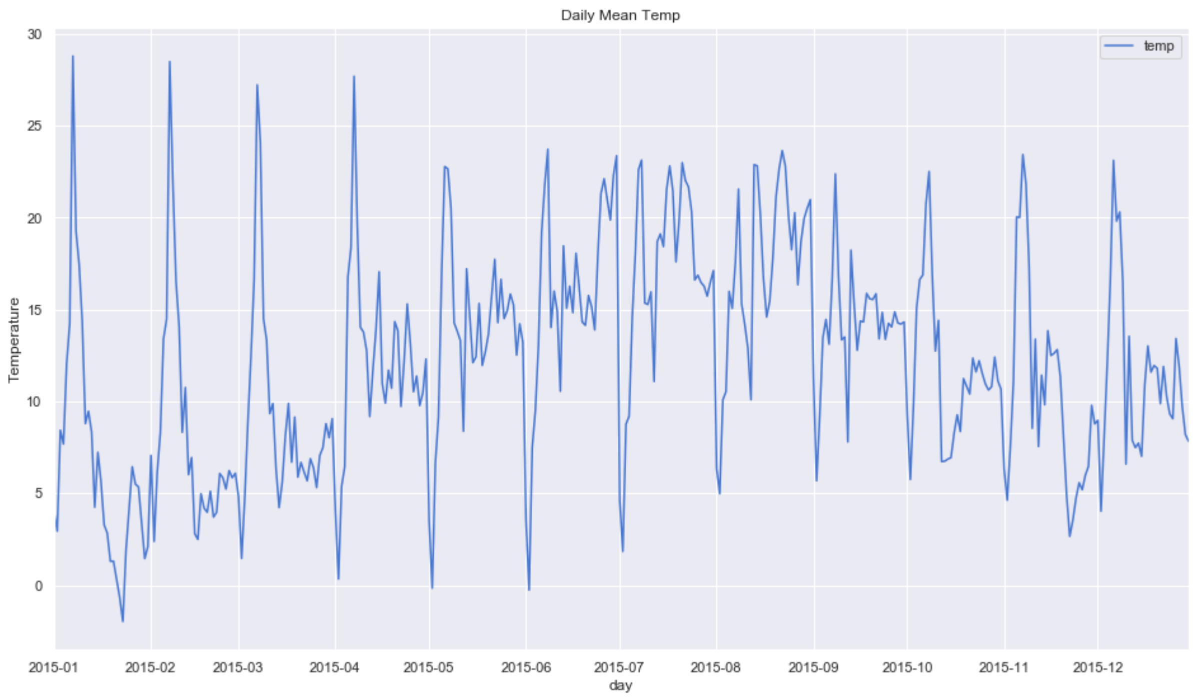
```python

# Mark the month for each entry so we can look at monthly patterns
data['month'] = data['date'].apply(lambda x: x.month)

# Is each time stamp on a working day (Mon-Fri)
data['working_day'] = (data['day_of_week'] >= 0) & (data['day_of_week'] <= 4)

# Classify into morning or evening times (assuming travel between 8.15-9am and 5.15-6pm)
data['morning'] = (data['time_of_day'] >= tm.time(7,0)) & (data['time_of_day'] <= tm.time(9,0))
data['evening'] = (data['time_of_day'] >= tm.time(14,0)) & (data['time_of_day'] <= tm.time(16,0))
data['obs'] = (data['working_day']) & ((data['morning'] & data['temp']) | (data['evening'] & data['temp']))



temp_days = data.groupby(['day']).agg({"temp": {"temp": "mean"}})
temp_days.reset_index(drop=False, inplace=True)
temp_days.columns = temp_days.columns.droplevel(level=1)
temp_days['month'] = temp_days['day'].apply(lambda x: x.month)


# Group by month for display - monthly data set for plots.
temp_month = data.groupby(['month']).agg({"temp": {"temp": "mean"}})
temp_month.reset_index(drop=False, inplace=True)
temp_month.columns = temp_month.columns.droplevel(level=1)


#Plotting Simple line plot
data.plot(x='date', y='temp')           #plotting hourly temrature
plt.title("Hourly Temp")
plt.ylabel('Temperature')
temp_days.plot(x='day', y='temp')       #plotting daily temrature
plt.ylabel('Temperature')
plt.title("Daily Mean Temp")
temp_month.plot(x='month', y='temp')    #plotting daily temrature
plt.ylabel('Temperature')
plt.title("Monthly mean Temp")
plt.show()

#Plotting Histogram and kde plot
temp_days['temp'].hist(color='r', alpha=0.5, bins=30)
plt.ylabel('Temperature')
plt.title("Daily Temp. Histogram")
plt.show()
temp_days['temp'].plot.kde()
plt.ylabel('Temperature')
plt.title("Daily Temp. kernel density")
plt.show()

sns.set(style="darkgrid", palette="muted",font_scale=1)
sns.distplot(temp_days['temp'],bins=30,kde=True,color="r")
plt.tight_layout()
plt.ylabel('Temperature')
plt.title("Daily Temp. Histogram and kernel density")
plt.show()

#Box plot of Daily Temp Data
temp_days.boxplot(['temp'],['month'])
plt.ylabel('Temperature')
plt.show()


```

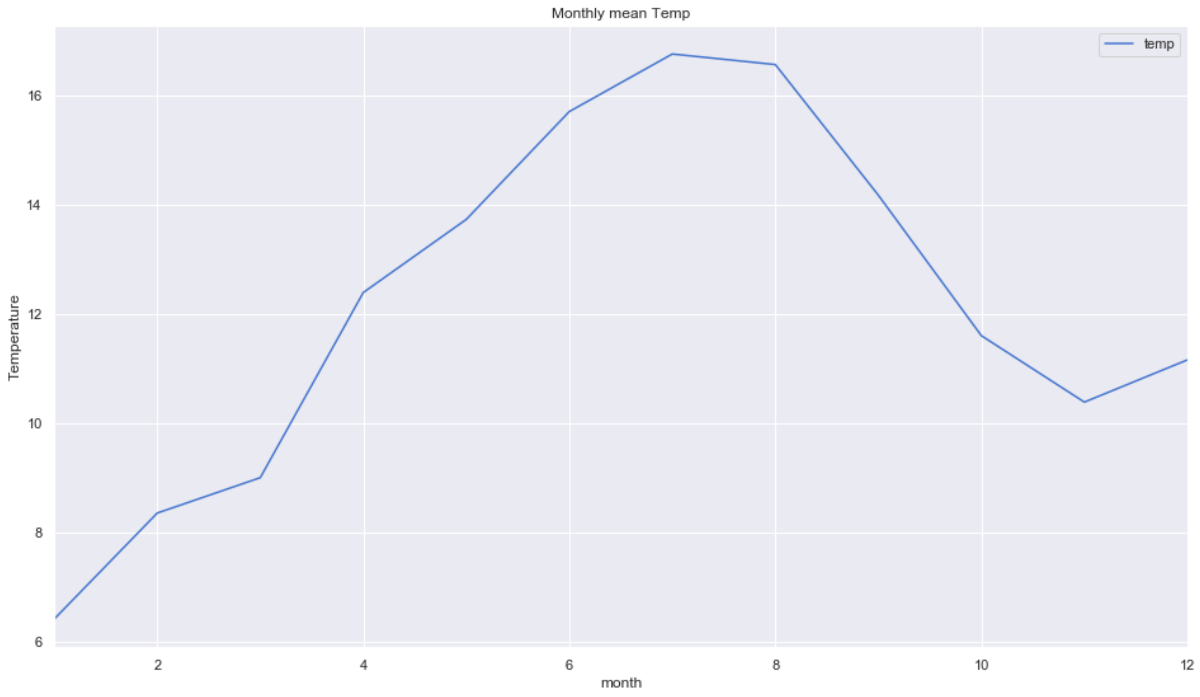
```
# Daily Temp. mean and Standard deviation
temp_days.set_index('day')
ma = temp_days['temp'].rolling(10).mean()
mstd = temp_days['temp'].rolling(10).std()
plt.figure(figsize=(15, 5))
plt.plot(temp_days['temp'], 'k')
plt.plot(ma.index-5, ma, 'b')
plt.fill_between(mstd.index-
    5, ma - 2 * mstd, ma + 2 * mstd,color='b', alpha=0.2)
plt.ylabel('Temperature')
plt.title("Daily Temp. mean and Standard deviation")
```

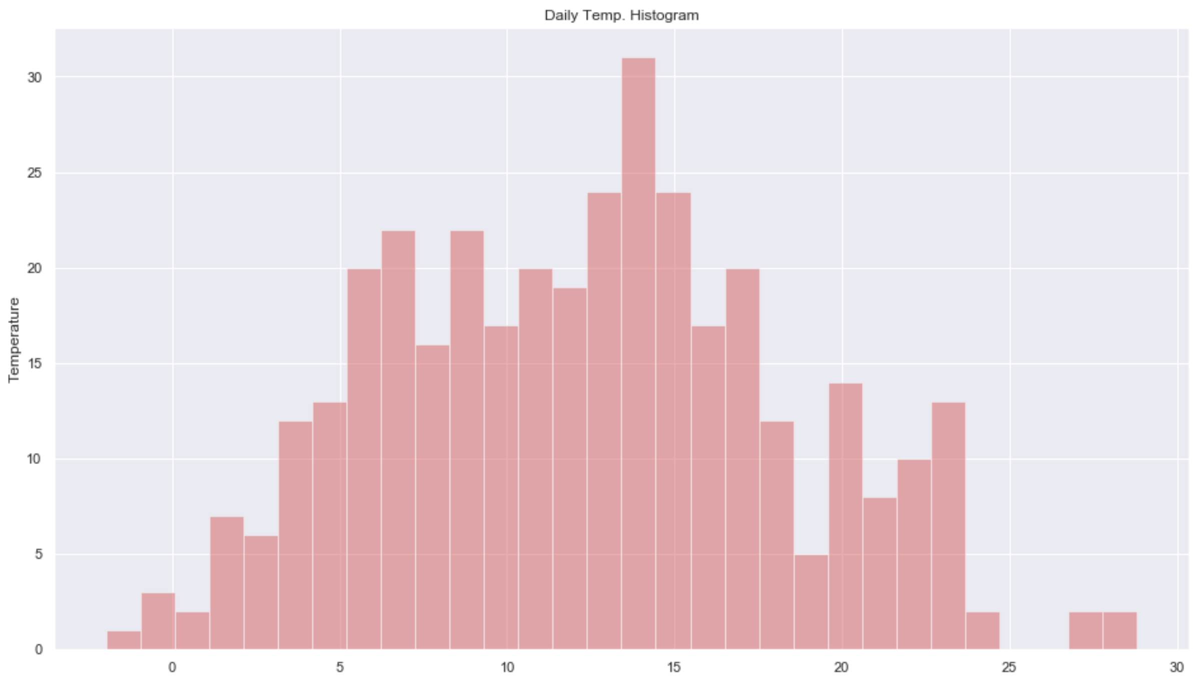
Output:

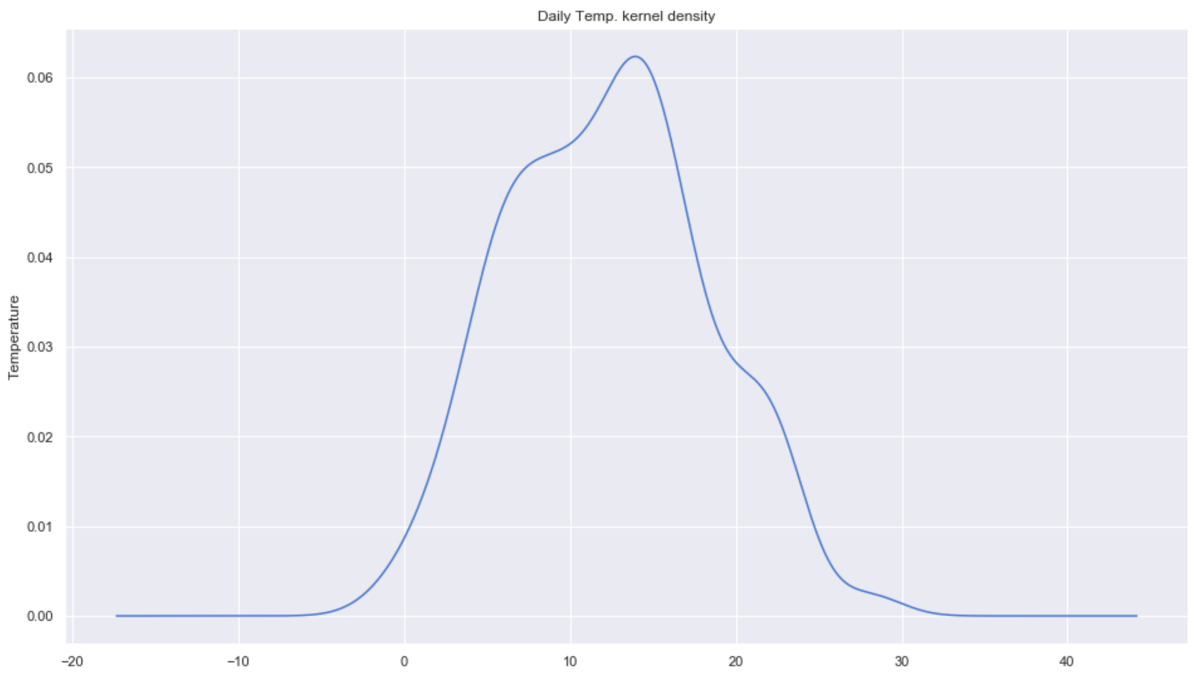
There is messed up days for INDIA105

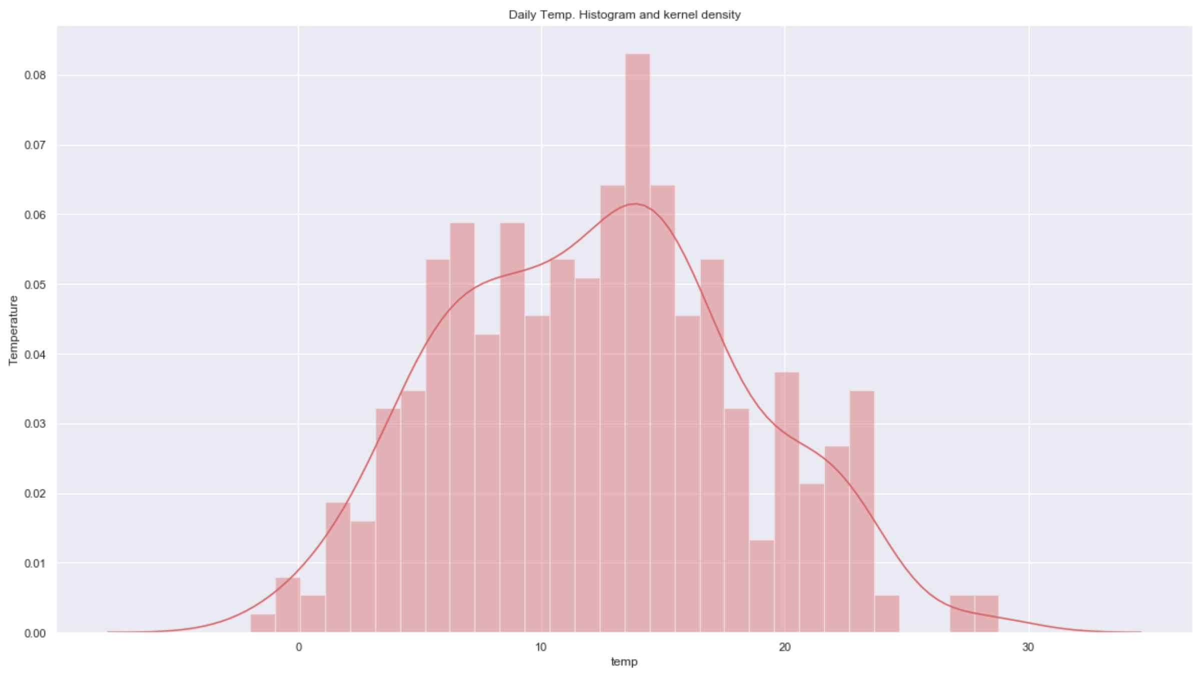


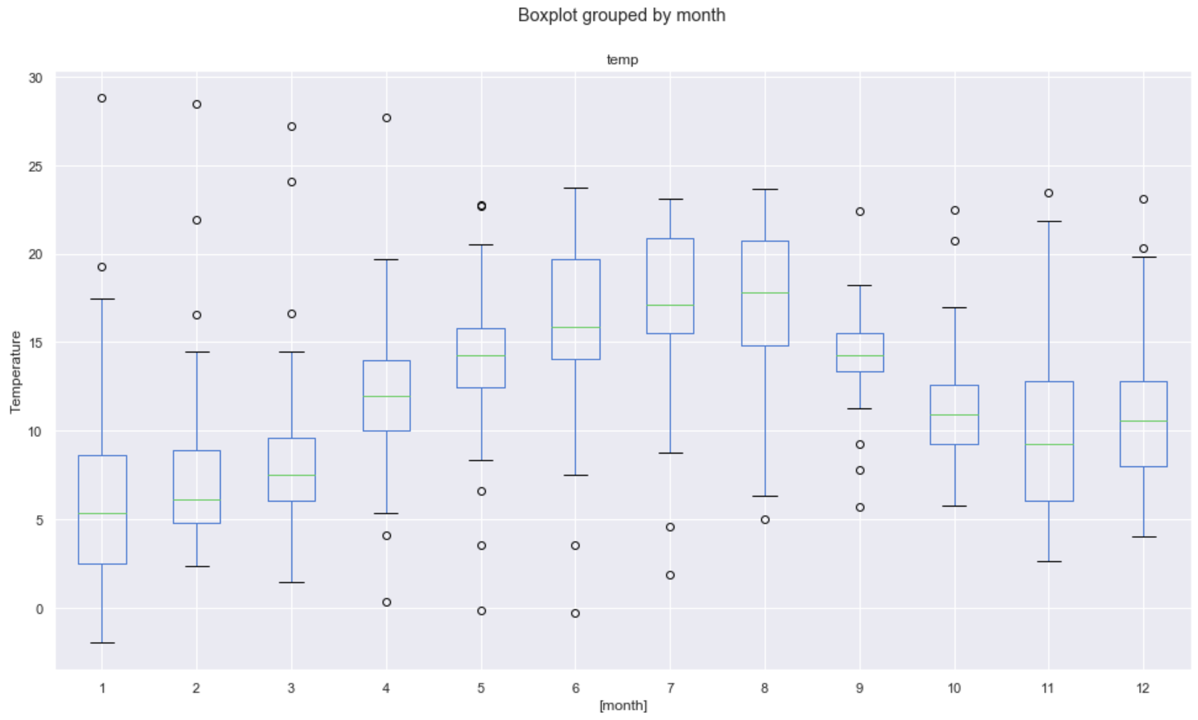

Daily Mean Temp
temp
Temperature
30
25
20
15
10
5
0
2015-01
2015-02
2015-03
2015-04
2015-05
2015-06
2015-07
2015-08
2015-09
2015-10
2015-11
2015-12
day

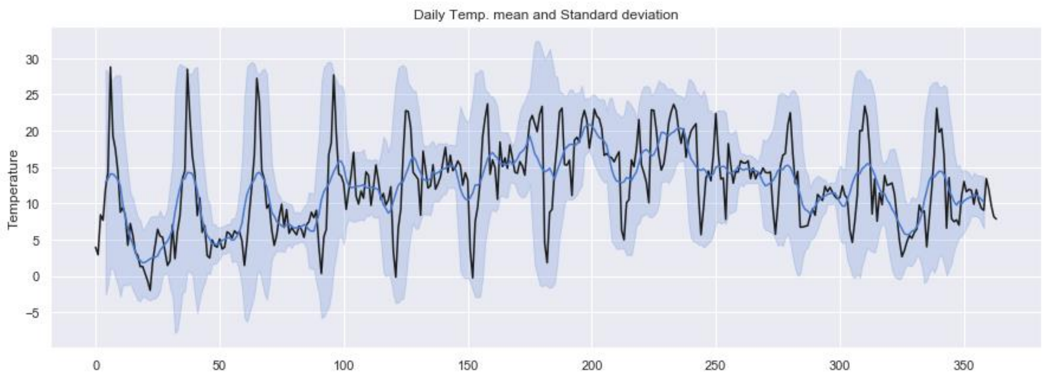

Monthly mean Temp
temp
Temperature
16
14
12
10
8
6
2
4
6
8
10
12
month


Daily Temp. Histogram
Temperature
0
5
10
15
20
25
30
0
5
10
15
20
25
30


Daily Temp. kernel density
Temperature
0.00
0.01
0.02
0.03
0.04
0.05
0.06
−20
−10
0
10
20
30
40


Daily Temp. Histogram and kernel density
Temperature
temp
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0
10
20
30


Boxplot grouped by month
temp
Temperature
[month]
0
5
10
15
20
25
30
1
2
3
4
5
6
7
8
9
10
11
12


Daily Temp. mean and Standard deviation
Temperature
−5
0
5
10
15
20
25
30
0
50
100
150
200
250
300
350

Lab 2 : To download and import weather data. Analysis and plotting of rainfall and wind data.

- To download weather data collected from AWS situated in different cities
- Extraction of required data from file (e.g. Rainfall and Wind)
- Removing erroneous points in data
- Hourly data to daily data and daily to monthly data generation
- Find the wet days and dry days over the year.
- Plotting rainfall data in heat map and plotting wind data in pi plot
- Compare different stations rainfall statistics

Practical: To plot Rainfall Data

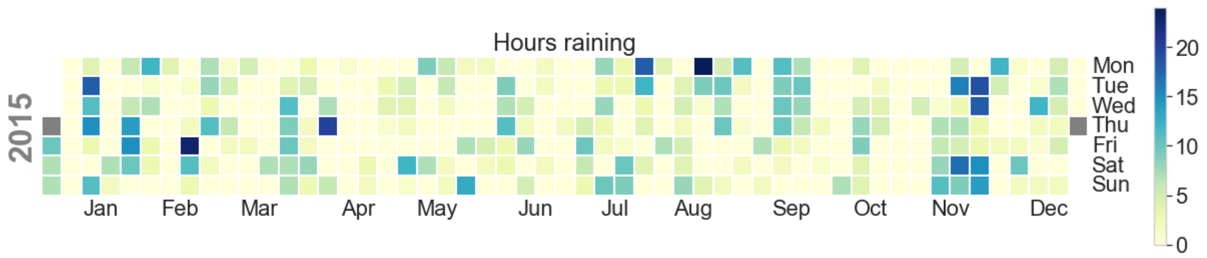
```
import requests
import pandas as pd
from dateutil import parser, rrule
from datetime import datetime, time, date
import datetime as tm
import os
import calendar
import matplotlib.pyplot as plt
import seaborn as sns
import calmap

station = 'INDIA105' # Delhi
data_raw = pd.read_csv('{}_weather.csv'.format(station))

# Give the variables some friendlier names and convert types as necessary.
data_raw['temp'] = data_raw['TemperatureC'].astype(float)
data_raw['rain'] = data_raw['HourlyPrecipMM'].astype(float)
data_raw['total_rain'] = data_raw['dailyrainMM'].astype(float)
data_raw['date'] = data_raw['DateUTC'].apply(parser.parse)
data_raw['humidity'] = data_raw['Humidity'].astype(float)
data_raw['wind_direction'] = data_raw['WindDirectionDegrees']
data_raw['wind'] = data_raw['WindSpeedKMH']

# Extract out only the data we need.
data = data_raw.loc[:, ['date', 'rain', 'total_rain']]
data = data[(data['date'] >= datetime(2015,1,1)) & (data['date'] <= datetime(201
5,12,31))]
# There's an issue with some stations that record rainfall ~-
2500 where data is missing.
if (data['rain'] < -500).sum() > 10:
    print("There's more than 10 messed up days for {}".format(station))
# remove the bad samples
data = data[data['rain'] > -500]

# Assign the "day" to every date entry
data['day'] = data['date'].apply(lambda x: x.date())

# Get the time, day, and hour of each timestamp in the dataset
data['time_of_day'] = data['date'].apply(lambda x: x.time())
data['day_of_week'] = data['date'].apply(lambda x: x.weekday())
data['hour_of_day'] = data['time_of_day'].apply(lambda x: x.hour)

# Mark the month for each entry so we can look at monthly patterns
data['month'] = data['date'].apply(lambda x: x.month)

# Is each time stamp on a working day (Mon-Fri)
data['working_day'] = (data['day_of_week'] >= 0) & (data['day_of_week'] <= 4)
```

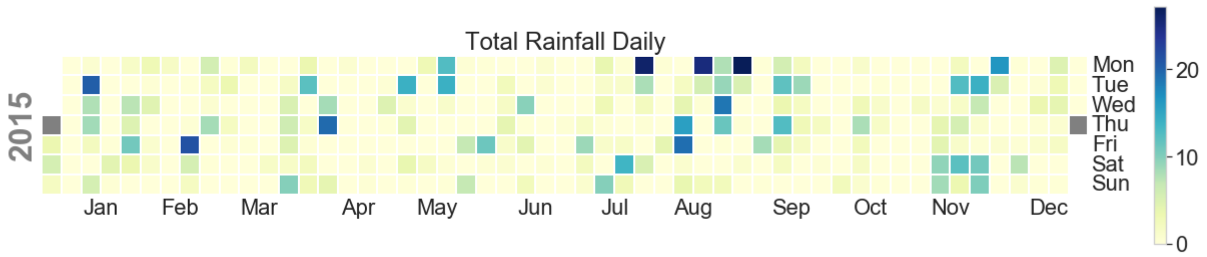
```python

# Classify into morning or evening times (assuming travel between 8.15-9am and 5.15-6pm)
data['morning'] = (data['time_of_day'] >= tm.time(8,15)) & (data['time_of_day'] <= tm.time(9,0))
data['evening'] = (data['time_of_day'] >= tm.time(17,15)) & (data['time_of_day'] <= tm.time(18,0))

# If there's any rain at all, mark that!
data['raining'] = data['rain'] > 0.0

# You get wet cycling if its a working day, and its raining at the travel times!
data['get_wet_cycling'] = (data['working_day']) & ((data['morning'] & data['rain']) | (data['evening'] & data['rain']))

temp = rainy_days.copy()
temp['date'] = pd.to_datetime(temp['date'])
temp.set_index('date', inplace=True)
#fig, ax = calmap.calendarplot(temp['hours_raining'], fig_kws={"figsize":(20,5)})
fig,ax=calmap.calendarplot(temp['hours_raining'],fillcolor='grey', linewidth=1,cmap='YlGnBu',fig_kws=dict(figsize=(20,4)))
plt.title("Hours raining")
fig.colorbar(ax[0].get_children()[1], ax=ax.ravel().tolist())
fig,ax=calmap.calendarplot(temp['total_rain'],fillcolor='grey', linewidth=1,cmap='YlGnBu',fig_kws=dict(figsize=(20,4)))
#fig, ax = calmap.calendarplot(temp['total_rain'], fig_kws={"figsize":(20,5)})
plt.title("Total Rainfall Daily")
fig.colorbar(ax[0].get_children()[1], ax=ax.ravel().tolist())

# Looking at the working days only and create a daily data set of working days:
wet_cycling = data[data['working_day'] == True].groupby('day')['get_wet_cycling'].any()
wet_cycling = pd.DataFrame(wet_cycling).reset_index()

# Group by month for display - monthly data set for plots.
wet_cycling['month'] = wet_cycling['day'].apply(lambda x: x.month)
monthly = pd.DataFrame(wet_cycling.groupby('month')['get_wet_cycling'].value_counts())
monthly.columns = ['Days']
monthly.reset_index(inplace = True)
monthly.columns = ['month', 'Rainy', 'Days']
monthly.replace({"Rainy": {True: "Wet", False:"Dry"}}, inplace=True)
monthly['month_name'] = monthly['month'].apply(lambda x: calendar.month_abbr[x])

# Get aggregate stats for each day in the dataset on rain in general - for heatmaps.
rainy_days = data.groupby(['day']).agg({
        "rain": {"rain": lambda x: (x > 0.0).any(),"rain_amount": "sum"},
        "total_rain": {"total_rain": "max"},
        "get_wet_cycling": {"get_wet_cycling": "any"}
        })

# clean up the aggregated data to a more easily analysed set:
rainy_days.reset_index(drop=False, inplace=True) # remove the 'day' as the index
rainy_days.rename(columns={"":"date"}, inplace=True) # The old index column didn't have a name - add "date" as name
rainy_days.columns = rainy_days.columns.droplevel(level=0) # The aggregation left us with a multi-index
```

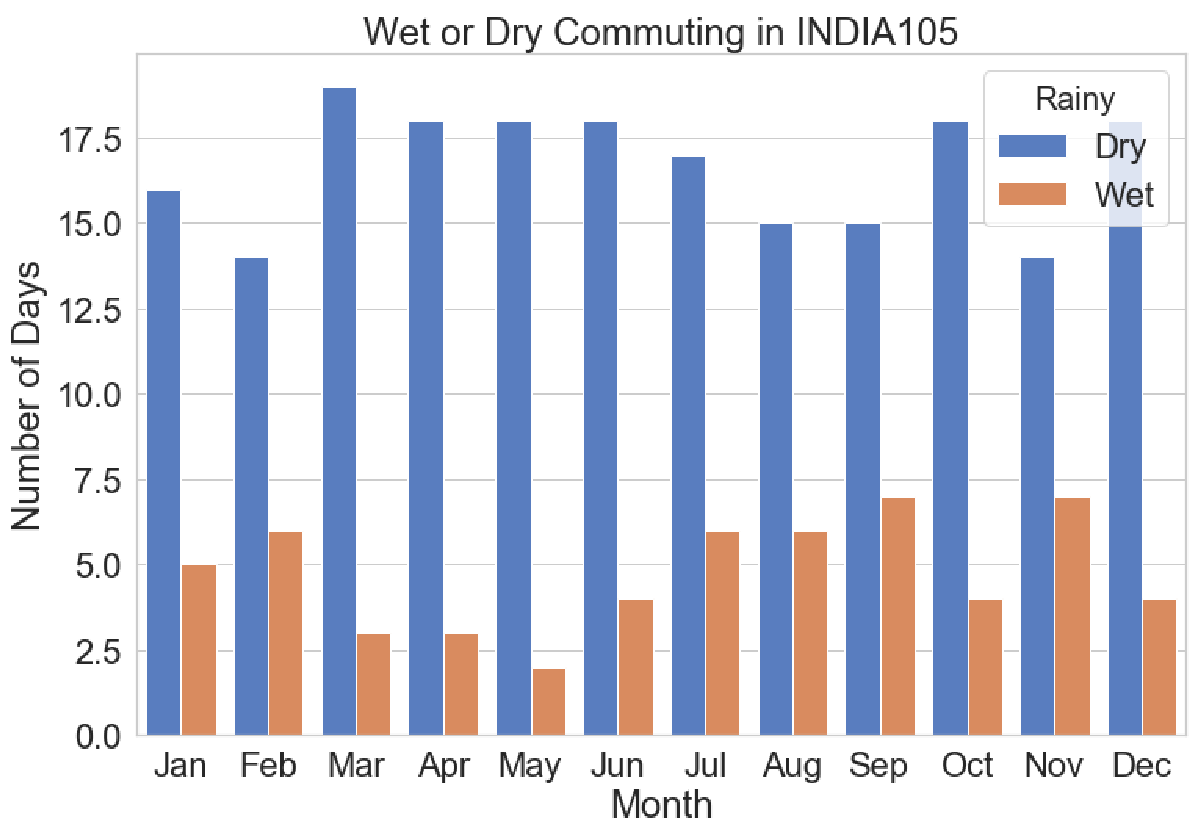
```
                                                             # Remove the top leve
l of this index.
rainy_days['rain'] = rainy_days['rain'].astype(bool)        # Change the "rain" c
olumn to True/False values

# Add the number of rainy hours per day this to the rainy_days dataset.
temp = data.groupby(["day", "hour_of_day"])['raining'].any()
temp = temp.groupby(level=[0]).sum().reset_index()
temp.rename(columns={'raining': 'hours_raining'}, inplace=True)
rainy_days = rainy_days.merge(temp, left_on='date', right_on='day', how='left')

rainy_days.drop('day', axis=1, inplace=True)

print ("In the year, there were {} rainy days of {} at {}".format(rainy_days['r
ain'].sum(), len(rainy_days), station))
print ("It was wet while cycling {} working days of {} at {}".format(wet_cyclin
g['get_wet_cycling'].sum(),len(wet_cycling),station))
print ("You get wet cycling {} % of the time!!".format(wet_cycling['get_wet_cyc
ling'].sum()*1.0*100/len(wet_cycling)))

# Monthly plot of rainy days
plt.figure(figsize=(12,8))
sns.set_style("whitegrid")
sns.set_context("notebook", font_scale=2)
sns.barplot(x="month_name", y="Days", hue="Rainy", data=monthly.sort_values(['m
onth', 'Rainy']))
plt.xlabel("Month")
plt.ylabel("Number of Days")
plt.title("Wet or Dry Commuting in {}".format(station))
```

Output 1:

```
There's more than 10 messed up days for INDIA105

In the year, there were 194 rainy days of 359 at INDIA105

It was wet while cycling 57 working days of 257 at INDIA105

You get wet cycling 22.17898832684825 % of the time!!
```







Practical 2: To analyze rainfall of different districts

```
import requests
import pandas as pd
from dateutil import parser, rrule
from datetime import datetime, time, date
import datetime as tm
import os
import calendar
import matplotlib.pyplot as plt
import seaborn as sns
import calmap

def analyse_station(data_raw, station):
    """
    Function to analyse weather data for a period from one weather station.

    Args:
        data_raw (pd.DataFrame): Pandas Dataframe made from CSV downloaded from
wunderground.com
        station (String): Name of station being analysed (for comments)

    Returns:
        dict: Dictionary with analysis in keys:
            data: Processed and cleansed data
            monthly: Monthly aggregated statistics on rainfall etc.
            wet_cycling: Data on working days and whether you get wet or not com
muting
            rainy_days: Daily total rainfall for each day in dataset.
    """
    # Give the variables some friendlier names and convert types as necessary.
    data_raw['temp'] = data_raw['TemperatureC'].astype(float)
```

```python
    data_raw['rain'] = data_raw['HourlyPrecipMM'].astype(float)
    data_raw['total_rain'] = data_raw['dailyrainMM'].astype(float)
    data_raw['date'] = data_raw['DateUTC'].apply(parser.parse)
    data_raw['humidity'] = data_raw['Humidity'].astype(float)
    data_raw['wind_direction'] = data_raw['WindDirectionDegrees']
    data_raw['wind'] = data_raw['WindSpeedKMH']

    # Extract out only the data we need.
    data = data_raw.loc[:, ['date', 'station', 'temp', 'rain', 'total_rain', 'humidity', 'wind']]
    data = data[(data['date'] >= datetime(2015,1,1)) & (data['date'] <= datetime(2015,12,31))]

    # There's an issue with some stations that record rainfall ~-2500 where data is missing.
    if (data['rain'] < -500).sum() > 10:
        print("There's more than 10 messed up days for {}".format(station))

    # remove the bad samples
    data = data[data['rain'] > -500]

    # Assign the "day" to every date entry
    data['day'] = data['date'].apply(lambda x: x.date())

    # Get the time, day, and hour of each timestamp in the dataset
    data['time_of_day'] = data['date'].apply(lambda x: x.time())
    data['day_of_week'] = data['date'].apply(lambda x: x.weekday())
    data['hour_of_day'] = data['time_of_day'].apply(lambda x: x.hour)
    # Mark the month for each entry so we can look at monthly patterns
    data['month'] = data['date'].apply(lambda x: x.month)

    # Is each time stamp on a working day (Mon-Fri)
    data['working_day'] = (data['day_of_week'] >= 0) & (data['day_of_week'] <= 4)

    # Classify into morning or evening times (assuming travel between 8.15-9am and 5.15-6pm)
    data['morning'] = (data['time_of_day'] >= time(8,15)) & (data['time_of_day'] <= time(9,0))
    data['evening'] = (data['time_of_day'] >= time(17,15)) & (data['time_of_day'] <= time(18,0))

    # If there's any rain at all, mark that!
    data['raining'] = data['rain'] > 0.0

    # You get wet cycling if its a working day, and its raining at the travel times!
    data['get_wet_cycling'] = (data['working_day']) & ((data['morning'] & data['rain']) | (data['evening'] & data['rain']))




    # Looking at the working days only and create a daily data set of working days:
    wet_cycling = data[data['working_day'] == True].groupby('day')['get_wet_cycling'].any()
    wet_cycling = pd.DataFrame(wet_cycling).reset_index()


    # Group by month for display
    wet_cycling['month'] = wet_cycling['day'].apply(lambda x: x.month)
    monthly = pd.DataFrame(wet_cycling.groupby('month')['get_wet_cycling'].value_counts())
    monthly.columns = ['Days']
    monthly.reset_index(inplace = True)
    monthly.columns = ['month', 'Rainy', 'Days']
    monthly.replace({"Rainy": {True: "Wet", False:"Dry"}}, inplace=True)
    monthly['month_name'] = monthly['month'].apply(lambda x: calendar.month_abbr[x])

# Get aggregate stats for each day in the dataset.
    rainy_days = data.groupby(['day']).agg({
            "rain": {"rain": lambda x: (x > 0.0).any(),
                     "rain_amount": "sum"},
            "total_rain": {"total_rain": "max"},
            "get_wet_cycling": {"get_wet_cycling": "any"}
            })
    rainy_days.reset_index(drop=False, inplace=True)
    rainy_days.columns = rainy_days.columns.droplevel(level=0)
    rainy_days['rain'] = rainy_days['rain'].astype(bool)
    rainy_days.rename(columns={"":"date"}, inplace=True)

    # Also get the number of hours per day where its raining, and add this to the rainy_days dataset.
    temp = data.groupby(["day", "hour_of_day"])['raining'].any()
    temp = temp.groupby(level=[0]).sum().reset_index()
    temp.rename(columns={'raining': 'hours_raining'}, inplace=True)
    #temp['day'] = temp['day'].apply(lambda x: x.to_datetime().date())
    rainy_days = rainy_days.merge(temp, left_on='date', right_on='day', how='left')
    rainy_days.drop('day', axis=1, inplace=True)

    print("In the year, there were {} rainy days of {} at {}".format(rainy_days['rain'].sum(), len(rainy_days), station))
    print("It was wet while cycling {} working days of {} at {}".format(wet_cycling['get_wet_cycling'].sum(),len(wet_cycling),station))
    print("You get wet cycling {} % of the time!!".format(wet_cycling['get_wet_cycling'].sum()*1.0*100/len(wet_cycling)))
    return {"data":data, 'monthly':monthly, "wet_cycling":wet_cycling, 'rainy_days': rainy_days}

# Load up each of the stations into memory.
stations = [
 ("INDIA105", "Delhi"),
 ("IBCNORTH17", "Vancouver"),
 ("IBELFAST4", "Belfast"),
 ("IBERLINB54", "Berlin"),
 ("ICOGALWA4", "Galway"),
 ("ICOMUNID56", "Madrid"),
 ("IDUBLIND35", "Dublin"),
 ("ILAZIORO71", "Rome"),
 ("ILEDEFRA6", "Paris"),
 ("ILONDONL28", "London"),
 ("IMUNSTER11", "Cork"),
 ("INEWSOUT455", "Sydney"),
 ("ISOPAULO61", "Sao Paulo"),
 ("IWESTERN99", "Cape Town"),
 ("KCASANFR148", "San Francisco"),
 ("KNYBROOK40", "New York"),
 ("IRENFREW4", "Glasgow"),
 ("IENGLAND64", "Liverpool"),
 ('IEDINBUR6', 'Edinburgh')
]
```

```
data = []
for station in stations:
    weather = {}
    print("Loading data for station: {}".format(station[1]))
    weather['data'] = pd.DataFrame.from_csv("{}_weather.csv".format(station[0]))
    weather['station'] = station[0]
    weather['name'] = station[1]
    data.append(weather)

for ii in range(len(data)):
    print("Processing data for {}".format(data[ii]['name']))
    data[ii]['result'] = analyse_station(data[ii]['data'], data[ii]['station'])

# Now extract the number of wet days, the number of wet cycling days, and the number of wet commutes for a single chart.
output = []
for ii in range(len(data)):
    temp = {
                "total_wet_days": data[ii]['result']['rainy_days']['rain'].sum(),
                "wet_commutes": data[ii]['result']['wet_cycling']['get_wet_cycling'].sum(),
                "commutes": len(data[ii]['result']['wet_cycling']),
                "city": data[ii]['name']
        }
    temp['percent_wet_commute'] = (temp['wet_commutes'] *1.0 / temp['commutes'])*100
    output.append(temp)
output = pd.DataFrame(output)

# Generate plot of percentage of wet commutes
plt.figure(figsize=(20,8))
sns.set_style("whitegrid")    # Set style for seaborn output
sns.set_context("notebook", font_scale=2)
sns.barplot(x="city", y="percent_wet_commute", data=output.sort_values('percent_wet_commute', ascending=False))
plt.xlabel("City")
plt.ylabel("Percentage of Wet Commutes (%)")
plt.suptitle("What percentage of your cycles to work do you need a raincoat?", y=1.05, fontsize=32)
plt.title("Based on Wundergroud.com weather data for 2015", fontsize=18)
plt.xticks(rotation=60)
plt.savefig("city_comparison_wet_commutes.png", bbox_inches='tight')
```

Output 2:

Loading data for station: Delhi

Loading data for station: Vancouver

Loading data for station: Belfast

Loading data for station: Berlin

Loading data for station: Galway

Loading data for station: Madrid

Loading data for station: Dublin

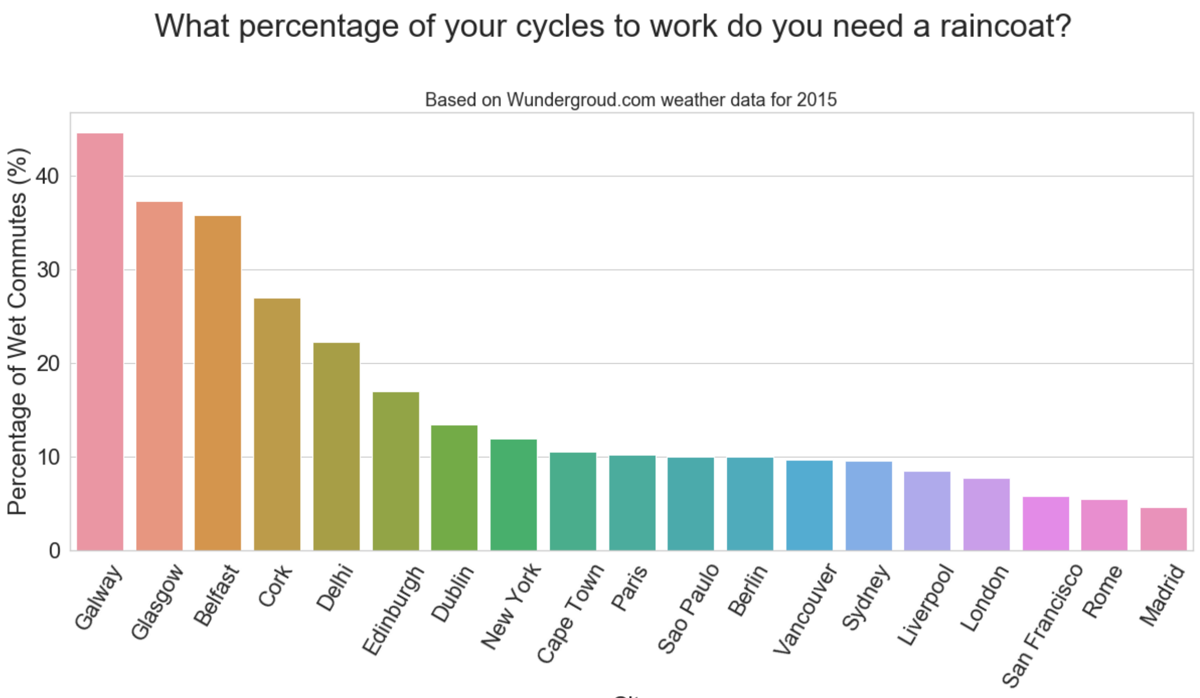
```
Loading data for station: Rome
Loading data for station: Paris
Loading data for station: London
Loading data for station: Cork
Loading data for station: Sydney
Loading data for station: Sao Paulo
Loading data for station: Cape Town
Loading data for station: San Francisco
Loading data for station: New York
Loading data for station: Glasgow
Loading data for station: Liverpool
Loading data for station: Edinburgh
Processing data for Delhi
There's more than 10 messed up days for INDIA105
In the year, there were 194 rainy days of 359 at INDIA105
It was wet while cycling 57 working days of 257 at INDIA105
You get wet cycling 22.17898832684825 % of the time!!
Processing data for Vancouver
In the year, there were 115 rainy days of 363 at IBCNORTH17
It was wet while cycling 25 working days of 259 at IBCNORTH17
You get wet cycling 9.652509652509652 % of the time!!
Processing data for Belfast
In the year, there were 235 rainy days of 364 at IBELFAST4
It was wet while cycling 93 working days of 260 at IBELFAST4
You get wet cycling 35.76923076923077 % of the time!!
Processing data for Berlin
There's more than 10 messed up days for IBERLINB54
In the year, there were 123 rainy days of 351 at IBERLINB54
It was wet while cycling 25 working days of 251 at IBERLINB54
You get wet cycling 9.9601593625498 % of the time!!
```

```
Processing data for Galway
In the year, there were 266 rainy days of 358 at ICOGALWA4
It was wet while cycling 114 working days of 256 at ICOGALWA4
You get wet cycling 44.53125 % of the time!!
Processing data for Madrid
In the year, there were 65 rainy days of 364 at ICOMUNID56
It was wet while cycling 12 working days of 260 at ICOMUNID56
You get wet cycling 4.615384615384615 % of the time!!
Processing data for Dublin
In the year, there were 180 rainy days of 365 at IDUBLIND35
It was wet while cycling 35 working days of 261 at IDUBLIND35
You get wet cycling 13.409961685823754 % of the time!!
Processing data for Rome
In the year, there were 61 rainy days of 363 at ILAZIORO71
It was wet while cycling 14 working days of 259 at ILAZIORO71
You get wet cycling 5.405405405405405 % of the time!!
Processing data for Paris
In the year, there were 116 rainy days of 344 at ILEDEFRA6
It was wet while cycling 25 working days of 246 at ILEDEFRA6
You get wet cycling 10.16260162601626 % of the time!!
Processing data for London
In the year, there were 101 rainy days of 364 at ILONDONL28
It was wet while cycling 20 working days of 260 at ILONDONL28
You get wet cycling 7.6923076923076925 % of the time!!
Processing data for Cork
In the year, there were 220 rainy days of 343 at IMUNSTER11
It was wet while cycling 66 working days of 245 at IMUNSTER11
You get wet cycling 26.93877551020408 % of the time!!
Processing data for Sydney
In the year, there were 120 rainy days of 365 at INEWSOUT455
```

```
It was wet while cycling 25 working days of 261 at INEWSOUT455
You get wet cycling 9.578544061302683 % of the time!!
Processing data for Sao Paulo
In the year, there were 143 rainy days of 364 at ISOPAULO61
It was wet while cycling 26 working days of 260 at ISOPAULO61
You get wet cycling 10.0 % of the time!!
Processing data for Cape Town
In the year, there were 123 rainy days of 361 at IWESTERN99
It was wet while cycling 27 working days of 257 at IWESTERN99
You get wet cycling 10.505836575875486 % of the time!!
Processing data for San Francisco
There's more than 10 messed up days for KCASANFR148
In the year, there were 61 rainy days of 364 at KCASANFR148
It was wet while cycling 15 working days of 260 at KCASANFR148
You get wet cycling 5.769230769230769 % of the time!!
Processing data for New York
In the year, there were 99 rainy days of 365 at KNYBROOK40
It was wet while cycling 31 working days of 261 at KNYBROOK40
You get wet cycling 11.877394636015326 % of the time!!
Processing data for Glasgow
There's more than 10 messed up days for IRENFREW4
In the year, there were 264 rainy days of 364 at IRENFREW4
It was wet while cycling 97 working days of 260 at IRENFREW4
You get wet cycling 37.30769230769231 % of the time!!
Processing data for Liverpool
In the year, there were 147 rainy days of 364 at IENGLAND64
It was wet while cycling 22 working days of 260 at IENGLAND64
You get wet cycling 8.461538461538462 % of the time!!
Processing data for Edinburgh
In the year, there were 194 rainy days of 364 at IEDINBUR6
```

It was wet while cycling 44 working days of 260 at IEDINBUR6

You get wet cycling 16.923076923076923 % of the time!!



Practical 3: To plot Wind data

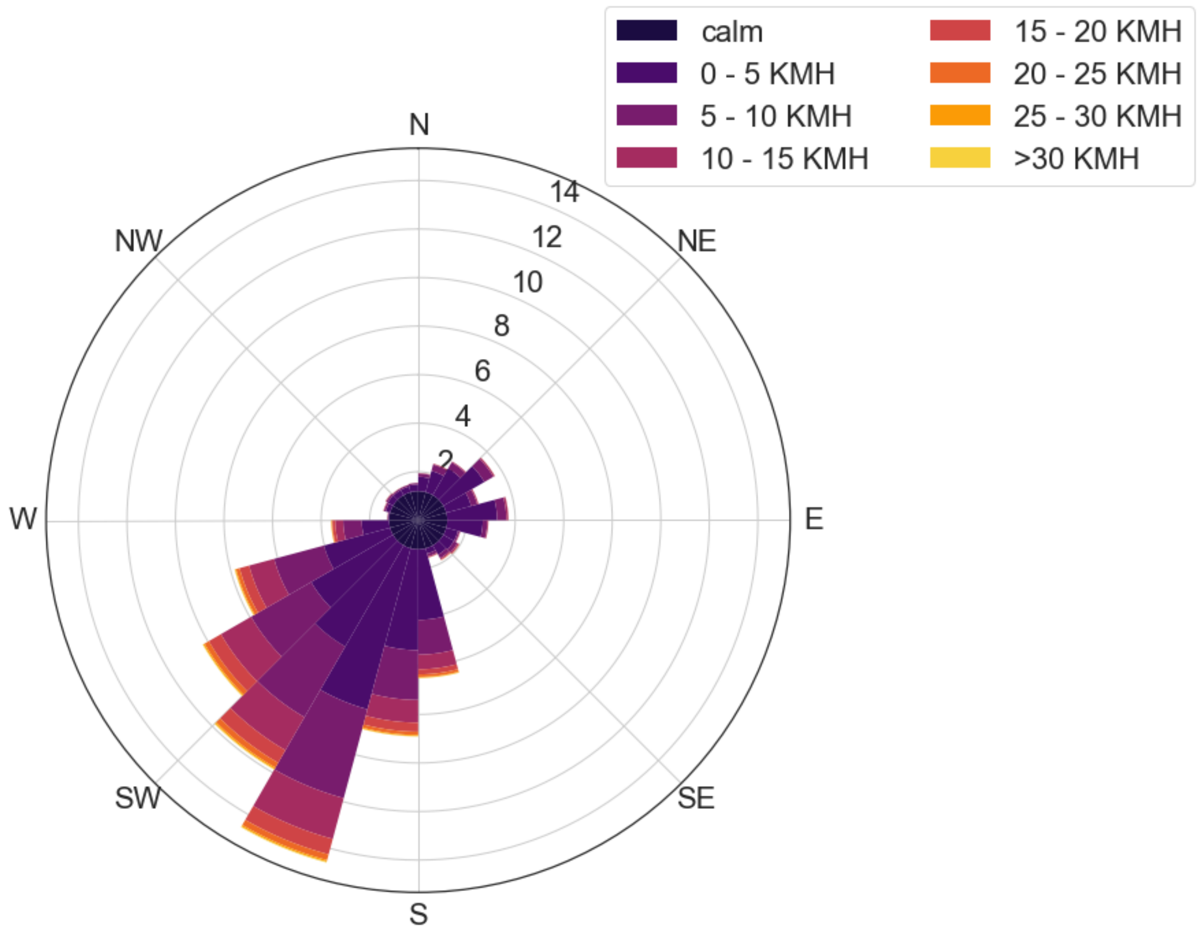
```
import numpy
import pandas as pd
from matplotlib import pyplot
from dateutil import parser, rrule
from datetime import datetime, time, date
import seaborn
import metar

def speed_labels(bins, units):
    labels = []
    for left, right in zip(bins[:-1], bins[1:]):
        if left == bins[0]:
            labels.append('calm'.format(right))
        elif numpy.isinf(right):
            labels.append('>{} {}'.format(left, units))
        else:
            labels.append('{} - {} {}'.format(left, right, units))

    return list(labels)




def _convert_dir(directions, N=None):
    if N is None:
        N = directions.shape[0]
    barDir = directions * numpy.pi/180. - numpy.pi/N
    barWidth = 2 * numpy.pi / N
    return barDir, barWidth
```

```python

spd_bins = [-1, 0, 5, 10, 15, 20, 25, 30, numpy.inf]
spd_labels = speed_labels(spd_bins, units='KMH')

dir_bins = numpy.arange(-7.5, 370, 15)
dir_labels = (dir_bins[:-1] + dir_bins[1:]) / 2




def wind_rose(rosedata, wind_dirs, palette=None):
    if palette is None:
        palette = seaborn.color_palette('inferno', n_colors=rosedata.shape[1])

    bar_dir, bar_width = _convert_dir(wind_dirs)

    fig, ax = pyplot.subplots(figsize=(10, 10), subplot_kw=dict(polar=True))
    ax.set_theta_direction('clockwise')
    ax.set_theta_zero_location('N')

    for n, (c1, c2) in enumerate(zip(rosedata.columns[:-
1], rosedata.columns[1:])):
        if n == 0:
            # first column only
            ax.bar(bar_dir, rosedata[c1].values,
                   width=bar_width,
                   color=palette[0],
                   edgecolor='none',
                   label=c1,
                   linewidth=0)

        # all other columns
        ax.bar(bar_dir, rosedata[c2].values,
               width=bar_width,
               bottom=rosedata.cumsum(axis=1)[c1].values,
               color=palette[n+1],
               edgecolor='none',
               label=c2,
               linewidth=0)

    leg = ax.legend(loc=(0.75, 0.95), ncol=2)
    xtl = ax.set_xticklabels(['N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW'])

    return fig


station = 'IEDINBUR6' # Edinburgh
data_raw = pd.read_csv('{}_weather.csv'.format(station))

# Give the variables some friendlier names and convert types as necessary.
data_raw['temp'] = data_raw['TemperatureC'].astype(float)
data_raw['rain'] = data_raw['HourlyPrecipMM'].astype(float)
data_raw['total_rain'] = data_raw['dailyrainMM'].astype(float)
data_raw['date'] = data_raw['DateUTC'].apply(parser.parse)
data_raw['humidity'] = data_raw['Humidity'].astype(float)
data_raw['wind_direction'] = data_raw['WindDirectionDegrees']
data_raw['wind'] = data_raw['WindSpeedKMH']
data_raw['dewpoint'] = data_raw['DewpointC'].astype(float)
data_raw['pressure'] = data_raw['PressurehPa'].astype(float)
data_raw['rad'] = data_raw['SolarRadiationWatts/m^2'].astype(float)

# Extract out only the data we need.
data = data_raw.loc[:, ['date', 'wind','wind_direction']]
data = data[(data['date'] >= datetime(2015,1,1)) & (data['date'] <= datetime(201
5,12,31))]
```

```

# total observation count
total_count = data.shape[0]
calm_count = data.query("wind == 0").shape[0]

print('Of {} total observations, {} have calm winds.'.format(total_count, calm_count))

rose = (
    data.assign(WindSpd_bins=lambda df:pandas.cut(df['wind'], bins=spd_bins, labels=spd_labels, right=True))
        .assign(WindDir_bins=lambda df:pandas.cut(df['wind_direction'], bins=dir_bins, labels=dir_labels, right=False))
        .replace({'WindDir_bins': {360: 0}})
        .groupby(by=['WindSpd_bins', 'WindDir_bins'])
        .size()
        .unstack(level='WindSpd_bins')
        .fillna(0)
        .assign(calm=lambda df: calm_count / df.shape[0])
        .sort_index(axis=1)
        .applymap(lambda x: x / total_count * 100)
)


directions = numpy.arange(0, 360, 15)
fig = wind_rose(rose, directions)
```

Output 3:

```
Of 101169 total observations, 29096 have calm winds.
```



Lab 3 : To predict data using Regression

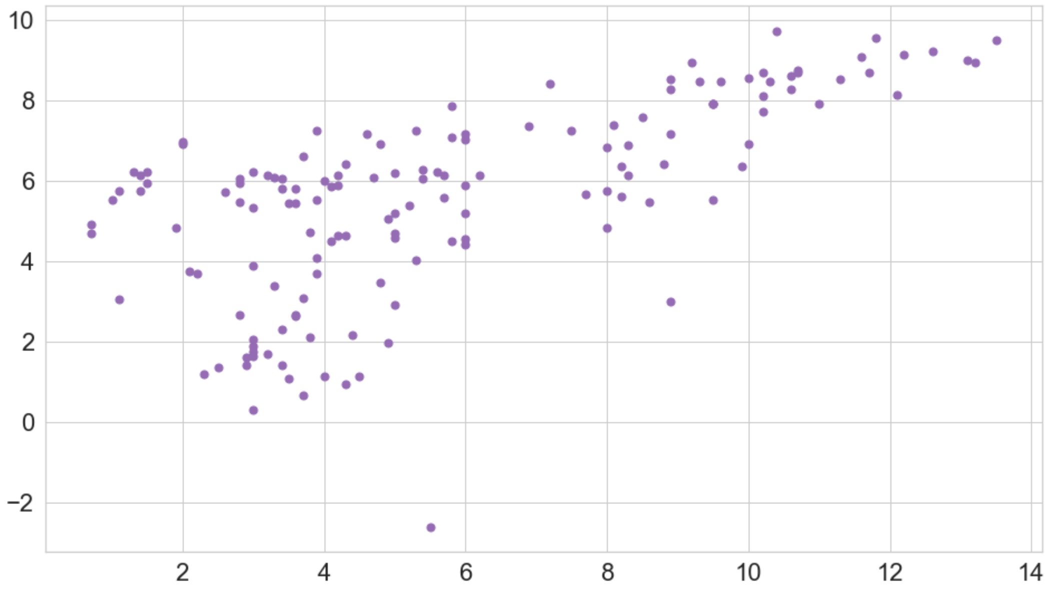
- Understand linear regression using equation
- Curve fitting with linear and polynomial line
- Predict data using model
- Library function for the regression model
- Weather prediction using weather data

Practical 1: Linear Regression using equation

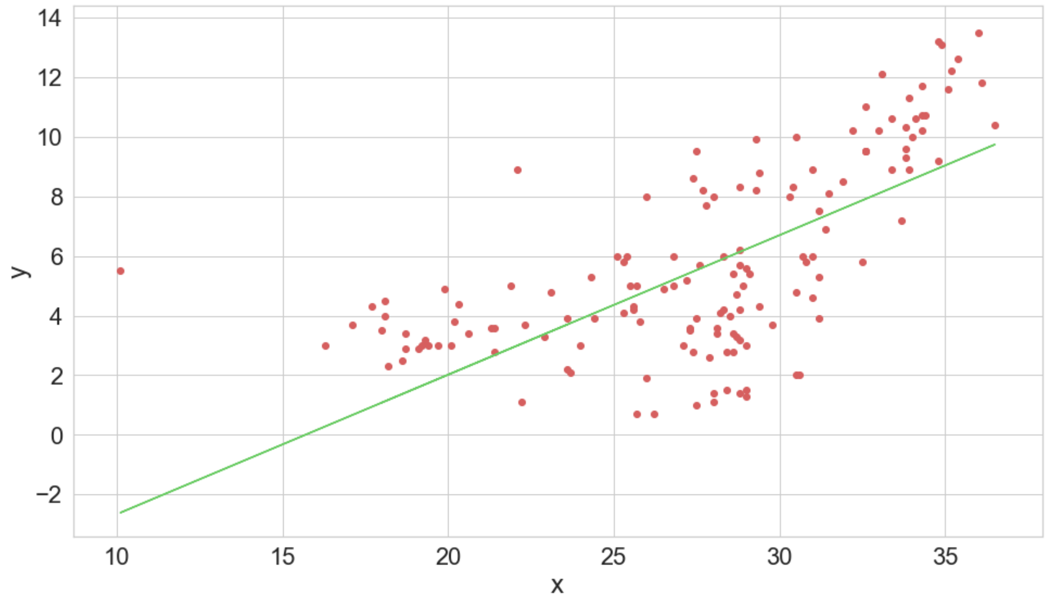
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

data = pd.read_csv('LineReg.csv')

X = data['MeanT']
Y1 = data['EP']
Y2 = data['MeanVP']

# splitting X and y into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, Y1, test_size=0.4,random_state=1)

correlationEP = X_train.corr(y_train)

# number of observations/points

x = X_train
y = y_train
n = np.size(x)

# mean of x and y vector
m_x, m_y = np.mean(x), np.mean(y)

# calculating cross-deviation and deviation about x
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x

# calculating regression coefficients
a = SS_xy / SS_xx
b = m_y - a*m_x
print("Estimated coefficients:\na = {} b = {}".format(a, b))

y_pred = b + a*X_test


# Plot outputs
# plotting the actual points as scatter plot
plt.scatter(y_test, y_pred, color='m', s = 50)
plt.show()

# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, y_pred))

```

```
# plotting regression line
# plotting the regression line
plt.plot(X_test, y_pred, color = "g")
plt.scatter(X_test, y_test, color = "r")

# putting labels
plt.xlabel('x')
plt.ylabel('y')

# function to show plot
plt.show()
```

Output 1:
Estimated coefficients:
a = 0.46753588689l7816 b = -7.3304635511440335



Mean squared error: 6.10
Variance score: 0.41



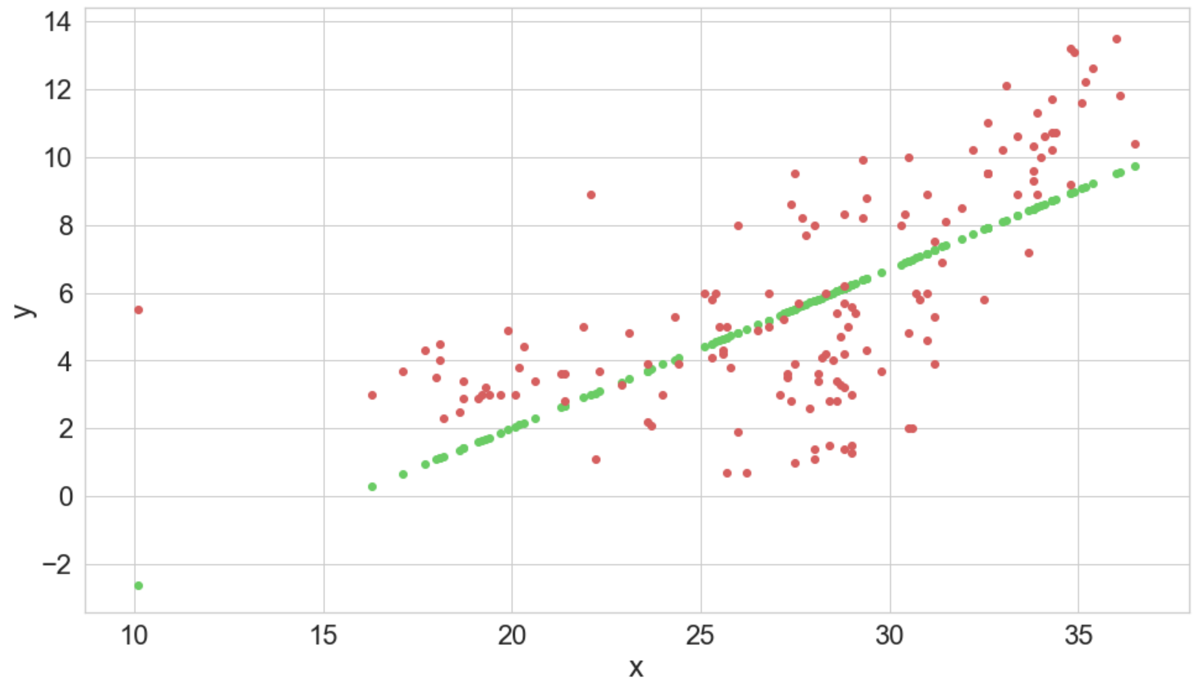
Practical 2: Curve Fitting – Linear and polynomial

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

data = pd.read_csv('LineReg.csv')

X = data['MeanT']
Y1 = data['EP']
Y2 = data['MeanVP']

# splitting X and y into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, Y1, test_size=0.4,random_state=1)

correlationEP = X_train.corr(y_train)


'''''###########################################################################

######################                Curve Fitting               ###############

#############################################################################'''

#####################               Linear            #######################
# number of observations/points
x = X_train
y = y_train
n = np.size(x)

# mean of x and y vector
m_x, m_y = np.mean(x), np.mean(y)

# calculating cross-deviation and deviation about x
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x

AA = np.zeros((2,2))
XX = np.zeros((2,1))
BB = np.zeros((2,1))

AA[0,0] = n
AA[0,1] = AA[1,0] = np.sum(x)
AA[1,1] = np.sum(x*x)

BB[0,0] = np.sum(y)
BB[1,0] = np.sum(x*y)

AI = np.linalg.inv(AA)

XX= np.matmul(AI,BB)

y_pred = XX[0] + XX[1]*X_test

# The mean squared error
print("Linear - Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, y_pred))
```

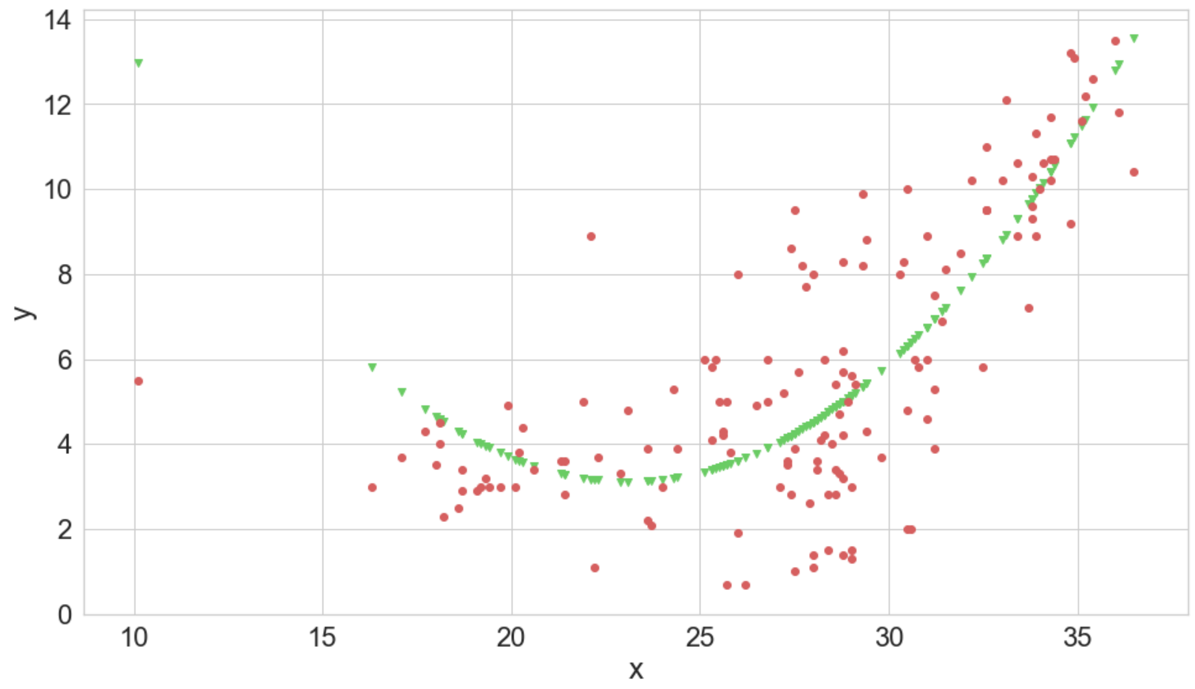
```python


# plotting the regression line
plt.scatter(X_test, y_pred, color = "g")
plt.scatter(X_test, y_test, color = "r")

# putting labels
plt.xlabel('x')
plt.ylabel('y')

# function to show plot
plt.show()

######################                   Polynomial Fitting                ######### 

AA = np.zeros((3,3))
XX = np.zeros((3,1))
BB = np.zeros((3,1))

AA[0,0] = n
AA[0,1] = AA[1,0] = np.sum(x)
AA[0,2] = AA[1,1] = AA[2,0] = np.sum(np.multiply(x,x))
AA[1,2] = AA[2,1] = np.sum(np.multiply(x,np.multiply(x,x)))
AA[2,2] = np.sum(np.multiply(np.multiply(x,x),np.multiply(x,x)))

BB[0,0] = np.sum(y)
BB[1,0] = np.sum(np.multiply(x,y))
BB[2,0] = np.sum(np.multiply(np.multiply(x,x),y))

AI = np.linalg.inv(AA)

XX= np.matmul(AI,BB)

y_pred = XX[0] + XX[1]*X_test + XX[2]*X_test*X_test


# The mean squared error
print("Polynomial - Mean squared error: %.2f" % mean_squared_error(y_test, y_pre
d))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, y_pred))

# plotting regression line
# plotting the regression line
plt.scatter(X_test, y_pred, color = "g", marker = "v")
plt.scatter(X_test, y_test, color = "r")


# putting labels
plt.xlabel('x')
plt.ylabel('y')

# function to show plot
plt.show()
```

Output 2:

Linear - Mean squared error: 6.10
Variance score: 0.41



Polynomial - Mean squared error: 4.32
Variance score: 0.58



Practical 3: Regression using Sklearn library

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

data = pd.read_csv('LineReg.csv')

X = data['MeanT']
Y1 = data['EP']
Y2 = data['MeanVP']

# splitting X and y into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, Y1, test_size=0.4,random_state=1)

correlationEP = X_train.corr(y_train)

'''
==========================================================
scikit-learn
==========================================================
'''

XX = np.array(X)
XX = XX.reshape(-1,1)

YY = np.array(Y1)
YY = YY.reshape(-1,1)

X_train, X_test, y_train, y_test = train_test_split(XX, YY, test_size=0.4,random_state=1)

reg = LinearRegression().fit(X_train, y_train)
y_pred = reg.predict(X_test)

# The mean squared error
print("Linear - Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, y_pred))

model = Pipeline([('poly', PolynomialFeatures(degree=2)),('linear', LinearRegression(fit_intercept=False))])
# fit to an order-3 polynomial data
model = model.fit(XX, YY)
poly_coef = model.named_steps['linear'].coef_
print('Polynomial coeeficients : ', poly_coef)
```

Output 3:

```
Linear - Mean squared error: 6.10
Variance score: 0.41
Polynomial coeeficients :  [[26.81885779 -2.14818585  0.0485469 ]]
```

Lab 4 : To perform Yield prediction of current year using previous year data

- To download past weather data with yield information
- Develop the yield prediction algorithm and calculate model
- Predict the yield using current weather data using developed model

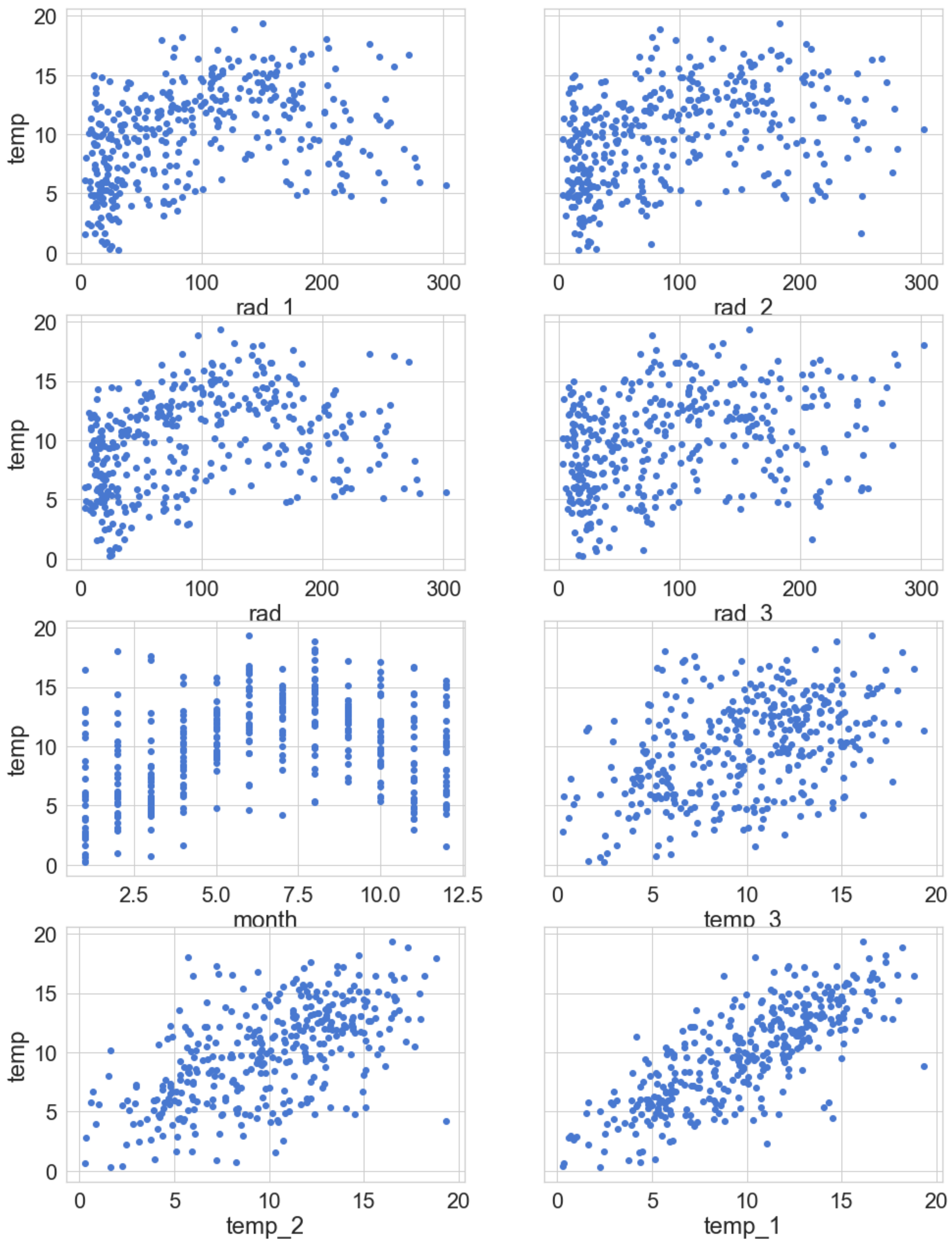
Practical 1: Temperature prediction using weather data

```python
import numpy as np
import pandas as pd
from matplotlib import pyplot
import matplotlib.pyplot as plt
from dateutil import parser, rrule
from datetime import datetime, time, date
import seaborn
import metar
import statsmodels.api as sm
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score


station = 'INDIA105' # Delhi
data_raw = pd.read_csv('{}_weather.csv'.format(station))

# Give the variables some friendlier names and convert types as necessary.
data_raw['temp'] = data_raw['TemperatureC'].astype(float)
data_raw['rain'] = data_raw['HourlyPrecipMM'].astype(float)
data_raw['total_rain'] = data_raw['dailyrainMM'].astype(float)
data_raw['date'] = data_raw['DateUTC'].apply(parser.parse)
data_raw['humidity'] = data_raw['Humidity'].astype(float)
data_raw['wind_direction'] = data_raw['WindDirectionDegrees']
data_raw['wind'] = data_raw['WindSpeedKMH']
data_raw['dewpoint'] = data_raw['DewpointC'].astype(float)
data_raw['pressure'] = data_raw['PressurehPa'].astype(float)
data_raw['rad'] = data_raw['SolarRadiationWatts/m^2'].astype(float)

# Extract out only the data we need.
data = data_raw.loc[:, ['date', 'station', 'temp', 'rain', 'total_rain', 'humidity',
                        'wind','wind_direction','pressure','dewpoint','rad']]
data = data[(data['date'] >= datetime(2015,1,1)) & (data['date'] <= datetime(2016,1,1))]

data = data[data['temp'] > -50 ]
data = data[data['rain'] > -500 ]
data = data[data['dewpoint'] > -10 ]
data = data[data['pressure'] > 0 ]


# Assign the "day" to every date entry
data['day'] = data['date'].apply(lambda x: x.date())

# Get the time, day, and hour of each timestamp in the dataset
data['time_of_day'] = data['date'].apply(lambda x: x.time())
data['day_of_week'] = data['date'].apply(lambda x: x.weekday())
data['hour_of_day'] = data['time_of_day'].apply(lambda x: x.hour)

# Mark the month for each entry so we can look at monthly patterns
data['month'] = data['date'].apply(lambda x: x.month)

daily_data = data.groupby(['day']).agg({"temp": {"temp5": "mean"} })
daily_data['rain'] = data.groupby(['day']).agg({"rain": {"sum"} })
```

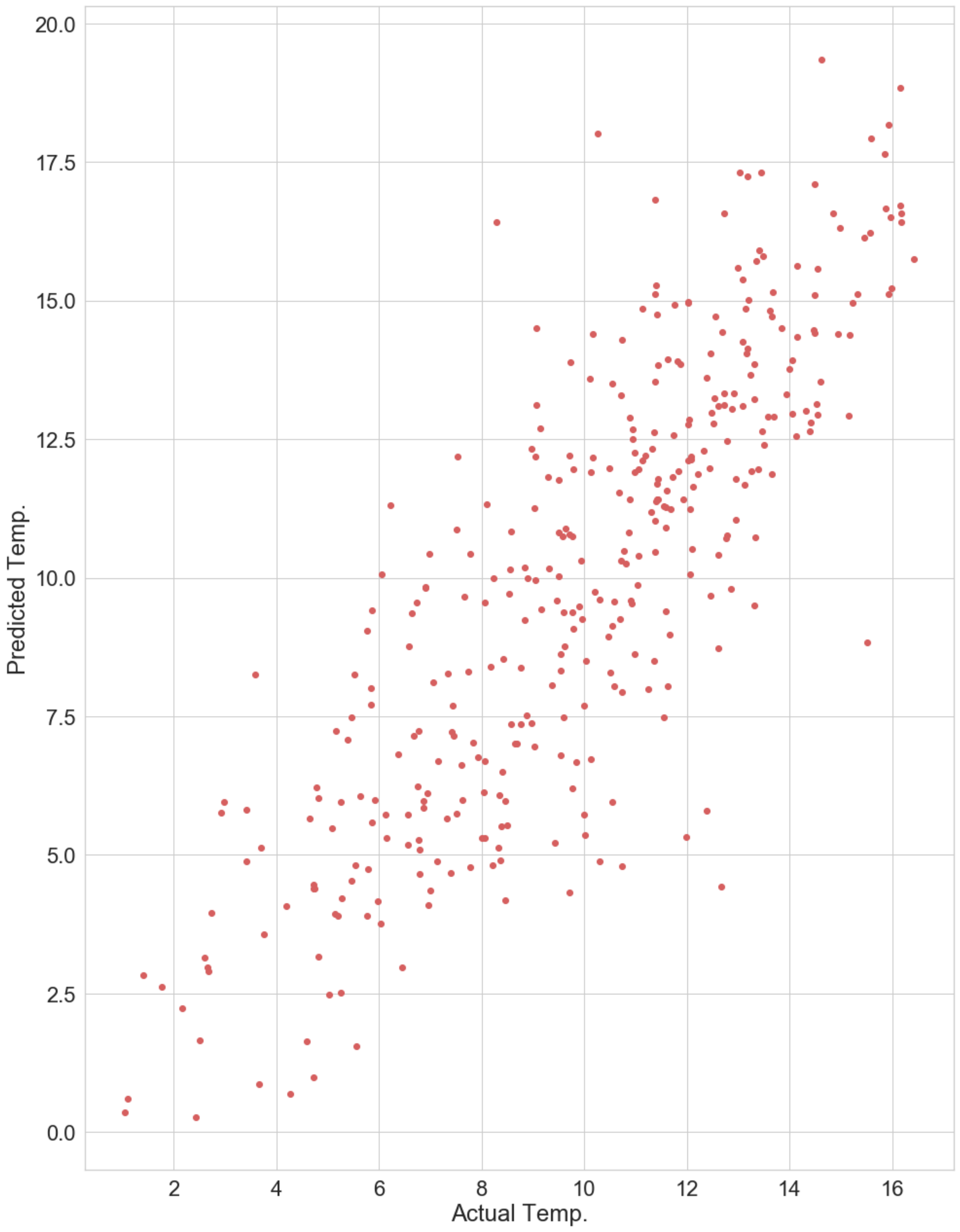
```python
daily_data['total_rain'] = data.groupby(['day']).agg({"total_rain": {"total_rain": "sum"} })
daily_data['humidity'] = data.groupby(['day']).agg({"humidity": {"humidity": "mean"} })
daily_data['rad'] = data.groupby(['day']).agg({"rad": {"rad": "mean"} })
daily_data['pressure'] = data.groupby(['day']).agg({"pressure": {"pressure": "mean"} })
daily_data.reset_index(drop=False, inplace=True)
daily_data.columns = daily_data.columns.droplevel(level=1)
daily_data['month'] = daily_data['day'].apply(lambda x: x.month)

# 1 day prior
N = 3

def derive_nth_day_feature(df, feature, N):
    rows = df.shape[0]
    nth_prior_measurements = [None]*N + [df[feature][i-N] for i in range(N, rows)]
    col_name = "{}_{}".format(feature, N)
    df[col_name] = nth_prior_measurements

for feature in ['temp', 'rain', 'total_rain', 'humidity',
                        'pressure','rad']:
    if feature != 'day':
        for N in range(1, 4):
            derive_nth_day_feature(daily_data, feature, N)

# Call describe on df and transpose it due to the large number of columns
spread = daily_data.describe().T

# precalculate interquartile range for ease of use in next calculation
IQR = spread['75%'] - spread['25%']

# create an outliers column which is either 3 IQRs below the first quartile or
# 3 IQRs above the third quartile
spread['outliers'] = (spread['min']<(spread['25%']-(3*IQR)))|(spread['max'] > (spread['75%']+3*IQR))

# just display the features containing extreme outliers
spread.ix[spread.outliers,]


daily_data.corr()[['temp']].sort_values('temp')

predictors = ['rad_1','rad_2','rad','rad_3','month','temp_3','temp_2','temp_1']


new_Data =  daily_data[['temp'] + predictors]

# manually set the parameters of the figure to and appropriate size
plt.rcParams['figure.figsize'] = [16, 22]

# call subplots specifying the grid structure we desire and that
# the y axes should be shared
fig, axes = plt.subplots(nrows=4, ncols=2, sharey=True)

# Since it would be nice to loop through the features in to build this plot
# let us rearrange our data into a 2D array of 6 rows and 3 columns
arr = np.array(predictors).reshape(4,2 )
```

```python

# use enumerate to loop over the arr 2D array of rows and columns
# and create scatter plots of each meantempm vs each feature
for row, col_arr in enumerate(arr):
    for col, feature in enumerate(col_arr):
        axes[row, col].scatter(new_Data[feature], new_Data['temp'])
        if col == 0:
            axes[row, col].set(xlabel=feature, ylabel='temp')
        else:
            axes[row, col].set(xlabel=feature)
plt.show()

# separate our my predictor variables (X) from my outcome variable y
X = new_Data[predictors]
Y = new_Data['temp']
X = sm.add_constant(X)
xx = X.ix[3:,3:]
yy = Y.ix[3:]
regr = linear_model.LinearRegression()
regr.fit(xx,yy)

# Make predictions using the testing set
y_pred = regr.predict(xx)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(yy, y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(yy, y_pred))

# Plot outputs
# plotting regression line
# plotting the regression line
plt.scatter(y_pred, yy, color = "r")
# putting labels
plt.xlabel('Actual Temp.')
plt.ylabel('Predicted Temp.')
plt.show()
```

Output 1:



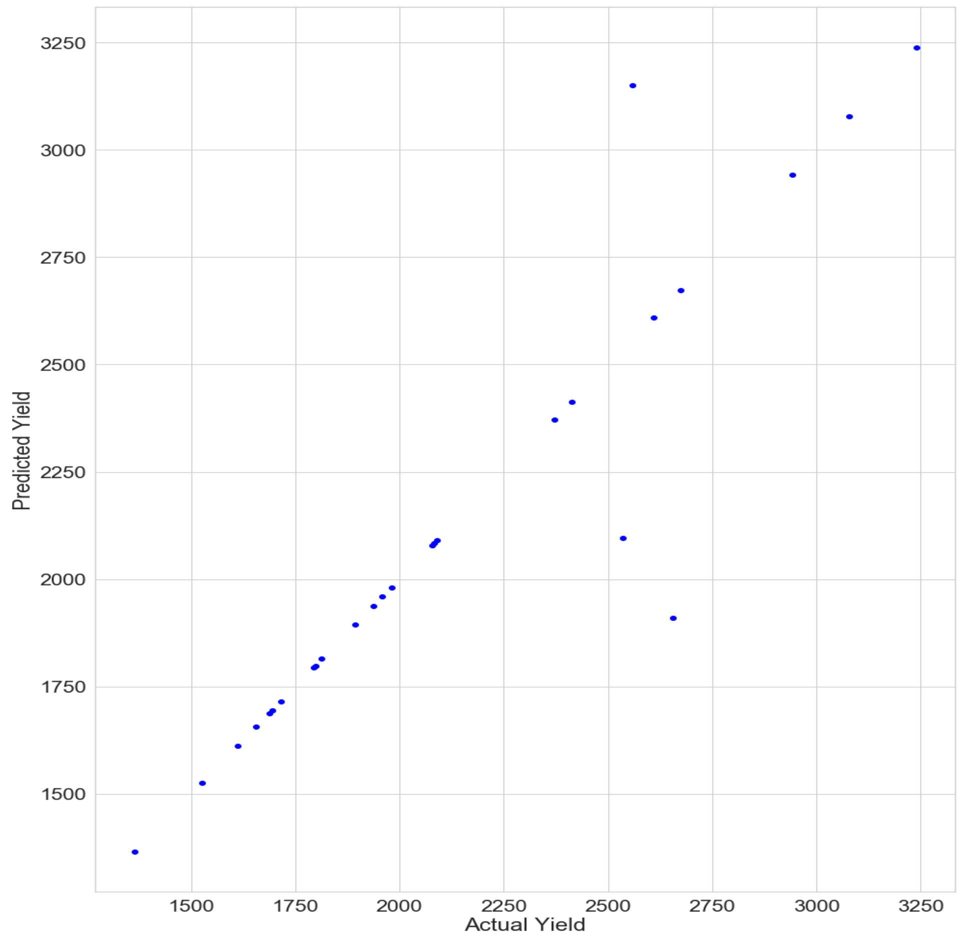
```
Coefficients:
 [ 1.51068971e-02 -5.27109593e-04  1.55530100e-01  9.05683699e-03
 -4.05726560e-04  7.17206170e-01]
Mean squared error: 5.22
Variance score: 0.68
```


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Predicted Temp.
2
4
6
8
10
12
14
16
Actual Temp.

Practical 2: Yield prediction using past weather data

```python
import numpy as np
import pandas as pd
from matplotlib import pyplot
import matplotlib.pyplot as plt
from dateutil import parser, rrule
from datetime import datetime, time, date
import seaborn
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
import metar

data = pd.read_csv('Yield_prediction.csv')
data.set_index('Unnamed: 0',inplace=True)

col = data.columns[3:]

correlation = data.corr()[['YIELD']]

multi = data.mul(correlation.squeeze().values,axis='columns')

def new_data(df, i):
    N = i*20
    new = df.T[2+N:22+N].sum(axis = 0)
    col_name = "{}".format(i)
    df[col_name] = new

for i in range(0,28):
    new_data(multi, i)
    new_data(data,i)

# create separate labels for df dataframe
arr= data.values
arr2= multi.values
arr1=arr[:,562:589]
arr2=arr2[:,562:589]
X = np.concatenate((arr1,arr2), axis=1)
Y = arr[:,0]

# linear regression

regr = linear_model.LinearRegression()
regr.fit(X[:24],Y[:24])

# Make predictions using the testing set
y_pred = regr.predict(X)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(Y, y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(Y, y_pred))

# Plot outputs
plt.scatter(Y, y_pred, color='blue')
plt.xticks(())
plt.yticks(())

plt.show()
```

```
Output 2:
Coefficients:
 [ 5.15982008e-03  1.77160473e-02 -5.48918935e-02 -1.40428367e-01
  3.79756586e-02  2.52421504e-02 -2.50831570e-02  4.50519851e-01
 -2.67159711e-01  3.39049094e-01 -3.37641431e-01  3.00026143e-01
 -4.07330280e-01 -5.13101963e-01 -2.17551493e-01  5.90634866e-01
  1.65866800e+00  4.04268364e-01  4.84008536e-01 -1.59953520e-01
 -1.42156112e+00 -1.94064850e+00 -1.37554737e-01  5.72751589e-01
 -2.24135997e-01 -1.86329100e+00  1.01318352e+00  2.64764554e-03
  4.58514627e-02  6.64602494e-03  1.77216353e-02  5.28644516e-03
 -1.32439117e-02 -4.55837812e-02  6.29101885e-01  5.91375545e-01
  2.83156516e+00  9.65426984e-01  2.00447787e-01 -3.72770148e-01
  4.14989738e-01  5.20254739e-01 -1.31841479e-01 -4.17885164e-02
 -4.11683648e-01 -9.86245802e-01  9.77491253e-01 -7.15948642e-01
 -7.97907888e-01  1.65179718e-01  2.06986013e-01  3.93182111e-01
 -1.08600036e+00  1.38240491e+00]
Mean squared error: 40757.14
Variance score: 0.83
```



Lab 5 : To develop code for unsupervised/supervised classification

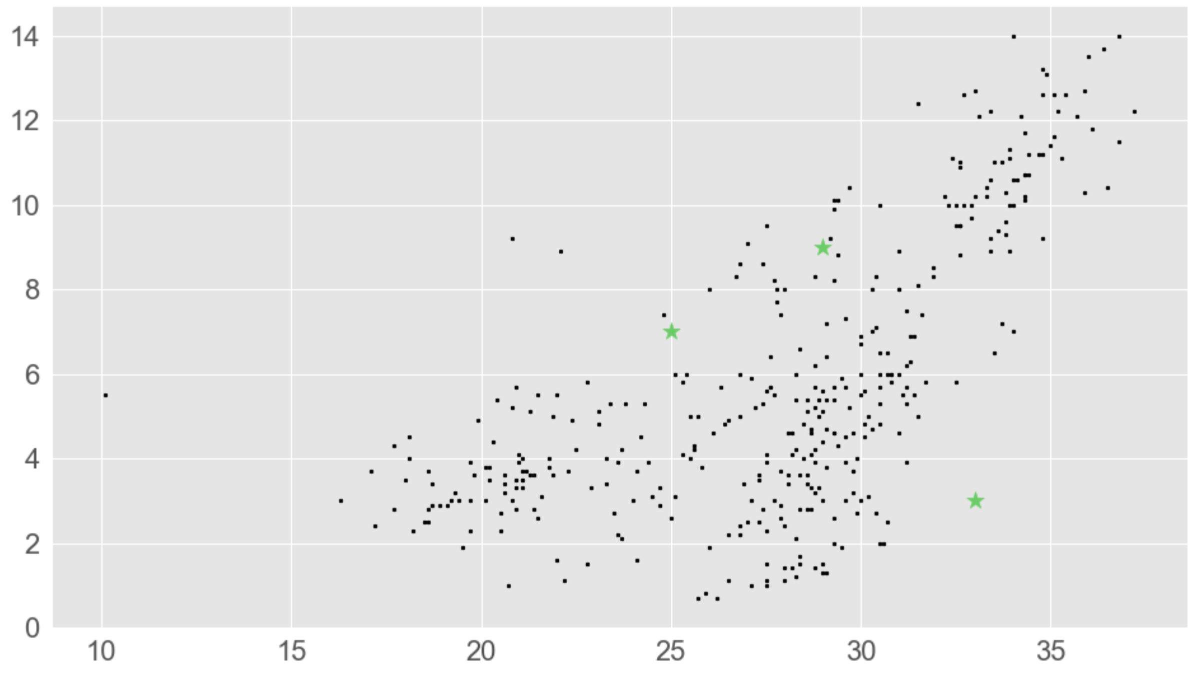
- Unsupervised classification e.g. K-means, fuzzy c means
- Supervised classification e.g. Maximum likelihood classification

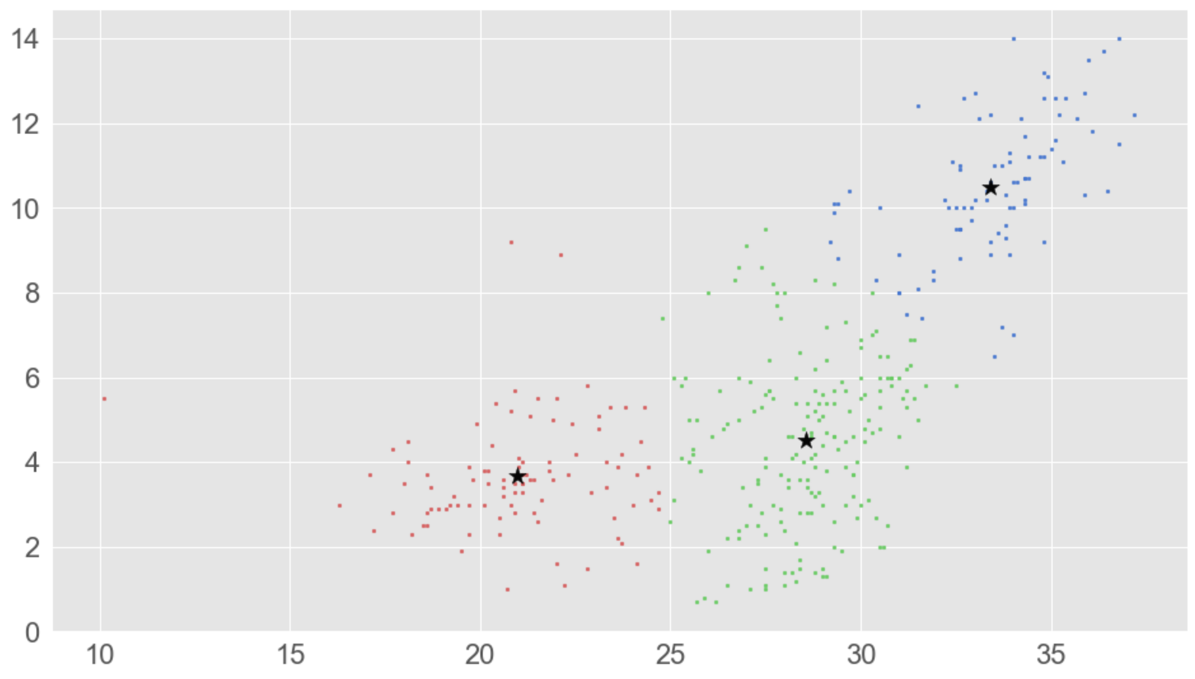
Practical 1: K-means clustering for temperature and ET data

```python
from copy import deepcopy
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')

# Importing the dataset
data = pd.read_csv('Temp_ET_Data.csv')
print("Input Data and Shape")
print(data.shape)
data.head()

# Getting the values and plotting it
f1 = data['MeanT'].values
f2 = data['EP'].values
X = np.array(list(zip(f1, f2)))
plt.scatter(f1, f2, c='black', s=7)



# Euclidean Distance Caculator
def dist(a, b, ax=1):
    return np.linalg.norm(a - b, axis=ax)

# Number of clusters
k = 3
# X coordinates of random centroids
C_x = np.random.randint(15, np.max(X), size=k)
# Y coordinates of random centroids
C_y = np.random.randint(0, np.max(X)-20, size=k)
C = np.array(list(zip(C_x, C_y)), dtype=np.float32)
print("Initial Centroids")
print(C)

# Plotting along with the Centroids
plt.scatter(f1, f2, c='#050505', s=7)
plt.scatter(C_x, C_y, marker='*', s=200, c='g')

# To store the value of centroids when it updates
C_old = np.zeros(C.shape)
# Cluster Lables(0, 1, 2)
clusters = np.zeros(len(X))
# Error func. - Distance between new centroids and old centroids
error = dist(C, C_old, None)
# Loop will run till the error becomes zero
while error != 0:
    # Assigning each value to its closest cluster
    for i in range(len(X)):
        distances = dist(X[i], C)
        cluster = np.argmin(distances)
        clusters[i] = cluster
    # Storing the old centroid values
    C_old = deepcopy(C)
    # Finding the new centroids by taking the average value
    for i in range(k):
        points = [X[j] for j in range(len(X)) if clusters[j] == i]
        C[i] = np.mean(points, axis=0)
    error = dist(C, C_old, None)

colors = ['r', 'g', 'b', 'y', 'c', 'm']
fig, ax = plt.subplots()
for i in range(k):
        points = np.array([X[j] for j in range(len(X)) if clusters[j] == i])
        ax.scatter(points[:, 0], points[:, 1], s=7, c=colors[i])
ax.scatter(C[:, 0], C[:, 1], marker='*', s=200, c='#050505')
```

Output 1:

```
Input Data and Shape
(365, 3)
Initial Centroids
[[25.  7.]
 [33.  3.]
 [29.  9.]]
Centroid values
Scratch
[[20.972631   3.6673684]
 [28.562637   4.521978 ]
 [33.41591   10.478409 ]]
```





Practical 2: Sklearn K-means clustering for temperature and ET data

```python
from copy import deepcopy
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans



plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')

# Importing the dataset
data = pd.read_csv('Temp_ET_Data.csv')
print("Input Data and Shape")
print(data.shape)
data.head()

# Getting the values and plotting it
f1 = data['MeanT'].values
f2 = data['EP'].values
X = np.array(list(zip(f1, f2)))

# Number of clusters
kmeans = KMeans(n_clusters=3)
# Fitting the input data
kmeans = kmeans.fit(X)
# Getting the cluster labels
labels = kmeans.predict(X)
# Centroid values
centroids = kmeans.cluster_centers_
```

```python

# Comparing with scikit-learn centroids
print("Centroid values")
print("sklearn")
print(centroids) # From sci-kit learn
```

Output 2:

```
Sklearn
[[28.56721311  4.54535519]
 [20.97263158  3.66736842]
 [33.46206897 10.49770115]]
```

Practical 3: Maximum likelihood classification for weather data

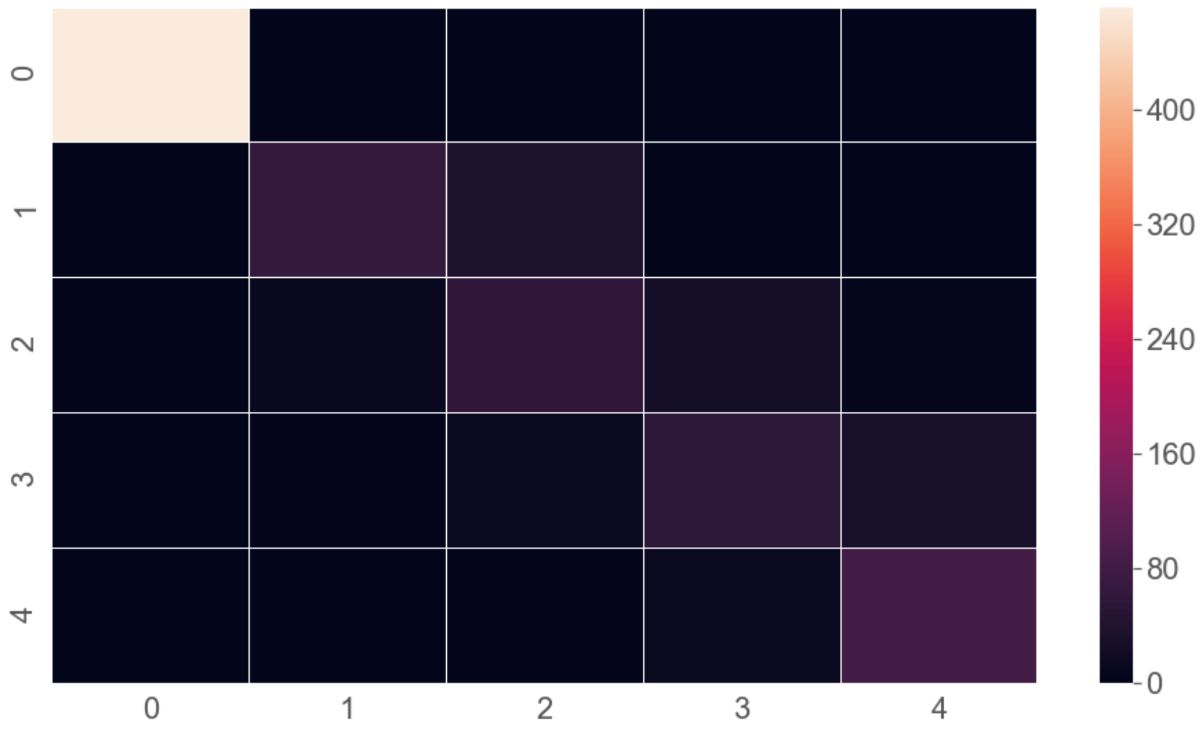
```python
import gdal
import matplotlib.pyplot as plt
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from math import pi
import seaborn as sns
import sklearn.metrics as sm

def MLestimator(dataset):
    mean = np.mean(dataset,axis=0)
    variance = np.var(dataset,axis=0)
    return(mean,variance)
def variance_squaremat(var):
    return(np.multiply(var,np.identity(7)))
def BayesClassifier(test_point,mean,variance):
    mean = np.array([mean])
    X = test_point-mean
    p1 = ((1/(2*pi)**0.5)/(np.linalg.det(variance)))**0.5
    p2 = np.exp(-
0.5*np.matmul(np.matmul(X,np.linalg.inv(variance)),np.transpose(X)))
    P = (1/3)*p1*p2     # 1/3 is the prior probability
    return(P)
#loading data to different variables as 2D array
img = gdal.Open('Weather_Guj.tif')
for i in range (1,9):
    T = img.GetRasterBand(i)
    if i==1:
        Tmin = T.ReadAsArray()
    if i==2:
        Tmax = T.ReadAsArray()
    if i==3:
        RH1 = T.ReadAsArray()
    if i==4:
        RH2 = T.ReadAsArray()
    if i==5:
        RF = T.ReadAsArray()
    if i==6:
        BSS = T.ReadAsArray()
    if i==7:
        WS = T.ReadAsArray()
    if i==8:
        AI = T.ReadAsArray()
```

```python

#Prepare Features & Targets
#creating 1D array of input data
minT = np.transpose(np.reshape(Tmin,(1,np.product(Tmin.shape))))
maxT = np.transpose(np.reshape(Tmax,(1,np.product(Tmax.shape))))
rh1 = np.transpose(np.reshape(RH1,(1,np.product(RH1.shape))))
rh2 = np.transpose(np.reshape(RH2,(1,np.product(RH2.shape))))
rf = np.transpose(np.reshape(RF,(1,np.product(RF.shape))))
bss = np.transpose(np.reshape(BSS,(1,np.product(BSS.shape))))
ws = np.transpose(np.reshape(WS,(1,np.product(WS.shape))))
ai = np.transpose(np.reshape(AI,(1,np.product(AI.shape))))


#removing error pixel from class image
for j in range (0,ai.shape[0]):
    if ai[j]<1:
        ai[j] = 0;

#creating input array with all features
x=np.concatenate((minT, maxT, rh1,rh2,rf,bss,ws,ai), axis=1)

class0 = x[x[:,7]==0]
class1 = x[x[:,7]==1]
class2 = x[x[:,7]==2]
class3 = x[x[:,7]==3]
class4 = x[x[:,7]==4]

m = 0.01

train_c0, test_c0 = train_test_split(class0,test_size=m)
train_c1, test_c1 = train_test_split(class1,test_size=m)
train_c2, test_c2 = train_test_split(class2,test_size=m)
train_c3, test_c3 = train_test_split(class3,test_size=m)
train_c4, test_c4 = train_test_split(class4,test_size=m)


test_data = np.concatenate((test_c0,test_c1,test_c2,test_c3,test_c4),axis = 0)


mean_c0,variance_c0= MLestimator(train_c0[:,0:7])
mean_c1,variance_c1= MLestimator(train_c1[:,0:7])
mean_c2,variance_c2= MLestimator(train_c2[:,0:7])
mean_c3,variance_c3= MLestimator(train_c3[:,0:7])
mean_c4,variance_c4= MLestimator(train_c4[:,0:7])


variance_c0 = variance_squaremat(variance_c0)
variance_c1 = variance_squaremat(variance_c1)
variance_c2 = variance_squaremat(variance_c2)
variance_c3 = variance_squaremat(variance_c3)
variance_c4 = variance_squaremat(variance_c4)



count = 0
n = np.shape(test_data)[0]
Pred = np.zeros([n,1])
for i in range(n):
    s = test_data[i,0:7]
    t = test_data[i,7]


    # virginica flower type probability calculation call
    prob_c0 = BayesClassifier(s,mean_c0,variance_c0)

    prob_c1 = BayesClassifier(s,mean_c1,variance_c1)

    prob_c2 = BayesClassifier(s,mean_c2,variance_c2)

    prob_c3 = BayesClassifier(s,mean_c3,variance_c3)

    prob_c4 = BayesClassifier(s,mean_c4,variance_c4)


    if(max(prob_c0,prob_c1,prob_c2,prob_c3,prob_c4)==prob_c0):
        p = 0
        Pred[i,:] = p
    elif(max(prob_c0,prob_c1,prob_c2,prob_c3,prob_c4)==prob_c1):
        p = 1
        Pred[i,:] = p
    elif(max(prob_c0,prob_c1,prob_c2,prob_c3,prob_c4)==prob_c2):
        p = 2
        Pred[i,:] = p
    elif(max(prob_c0,prob_c1,prob_c2,prob_c3,prob_c4)==prob_c3):
        p = 3
        Pred[i,:] = p
    elif(max(prob_c0,prob_c1,prob_c2,prob_c3,prob_c4)==prob_c4):
        p = 4
        Pred[i,:] = p

    if(t==p):
        count+=1

print("The accuracy of the Bayes classifier which uses ML estimation is: %0.2f
  %%" %(count*100/n))
print(sm.classification_report(test_data[:,7], Pred))
print('MxL\n', sm.confusion_matrix(test_data[:,7], Pred))
print('MxL accuracy is',sm.accuracy_score(Pred,test_data[:,7]))

ax = sns.heatmap(sm.confusion_matrix(test_data[:,7], Pred), linewidth=0.1)
plt.show()
```

Output 3:

```
The accuracy of the Bayes classifier which uses ML estimation is: 83.83 %
          precision   recall  f1-score   support

      0.0      1.00      1.00      1.00       473
      1.0      0.83      0.64      0.73       101
      2.0      0.53      0.59      0.56       101
      3.0      0.58      0.55      0.57       102
      4.0      0.70      0.83      0.76       101

  micro avg      0.84      0.84      0.84       878
  macro avg      0.73      0.72      0.72       878
```

```
weighted avg       0.84      0.84      0.84       878

MxL
 [[471   0   1   1   0]
 [  0  65  36   0   0]
 [  0  11  60  25   5]
 [  0   2  13  56  31]
 [  0   0   3  14  84]]
MxL accuracy is 0.8382687927107062
```



Lab 6 : To perform unsupervised/supervised classification using libraries

- To use different types of classification models from sklearn library to classify aridity index using weather parameters.

Practical : Sklearn classification models

```python
import gdal
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
from sklearn.metrics import mean_squared_error, r2_score



#loading data to different variables as 2D array
img = gdal.Open('Weather_Guj.tif')
for i in range (1,9):
    T = img.GetRasterBand(i)
    if i==1:
        Tmin = T.ReadAsArray()
    if i==2:
        Tmax = T.ReadAsArray()
    if i==3:
        RH1 = T.ReadAsArray()
    if i==4:
        RH2 = T.ReadAsArray()
    if i==5:
        RF = T.ReadAsArray()
    if i==6:
        BSS = T.ReadAsArray()
    if i==7:
        WS = T.ReadAsArray()
    if i==8:
        AI = T.ReadAsArray()


#Prepare Features & Targets
#creating 1D array of input data
minT = np.transpose(np.reshape(Tmin,(1,np.product(Tmin.shape))))
maxT = np.transpose(np.reshape(Tmax,(1,np.product(Tmax.shape))))
rh1 = np.transpose(np.reshape(RH1,(1,np.product(RH1.shape))))
rh2 = np.transpose(np.reshape(RH2,(1,np.product(RH2.shape))))
rf = np.transpose(np.reshape(RF,(1,np.product(RF.shape))))
bss = np.transpose(np.reshape(BSS,(1,np.product(BSS.shape))))
ws = np.transpose(np.reshape(WS,(1,np.product(WS.shape))))
ai = np.transpose(np.reshape(AI,(1,np.product(AI.shape))))


#creating input array with all features
x=np.concatenate((minT, maxT, rh1,rh2,rf,bss,ws), axis=1)


#removing error pixel from class image
for j in range (0,ai.shape[0]):
    if ai[j]<1:
        ai[j] = 0;


```

```python

#################### K-Means ##############################################

#Unsupervised classification KMeans
model = KMeans(n_clusters= 5, max_iter=500)    #selection of model with input cluster size
model.fit(x)  #fitting model on input array x

#using model parameters predicting output class
y_pred = model.labels_

# The mean squared error
print("Mean squared error for K-means: %.2f" % mean_squared_error(ai, y_pred))

# Explained variance score: 1 is perfect prediction
print('Variance score for K-means: %.2f' % r2_score(ai, y_pred))


#creating 1D array to 2D array of class image
ai_orig = np.reshape(ai,(AI.shape))   #Original class Image
kmeans = np.reshape(y_pred,(AI.shape))   #Predicted class image

##############################################################################

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, ai, test_size = 0.4, random_state = 0)


##################### K-Nearest Neighbours #################################
from sklearn.neighbors import KNeighborsClassifier

Model = KNeighborsClassifier(n_neighbors=8)
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print('KNN\n', sm.classification_report(y_test, y_pred))
print('KNN\n', sm.confusion_matrix(y_test, y_pred))

# Accuracy score
KNN_accuracy = sm.accuracy_score(y_pred,y_test)
print('KNN accuracy is',sm.accuracy_score(y_pred,y_test))

#creating 1D array to 2D array of class image
y_pred = Model.predict(x)
KNN = np.reshape(y_pred,(AI.shape))   #Predicted class image


################## Support Vector Machine  ###############################

from sklearn.svm import SVC

Model = SVC()
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print('SVM\n',sm.classification_report(y_test, y_pred))
print('SVM\n',sm.confusion_matrix(y_test, y_pred))

```

```
# Accuracy score
SVM_accuracy = sm.accuracy_score(y_pred,y_test)
print('SVM accuracy is',sm.accuracy_score(y_pred,y_test))

#creating 1D array to 2D array of class image
y_pred = Model.predict(x)
SVM = np.reshape(y_pred,(AI.shape))   #Predicted class image


##################################### Logistic Regression #####################
##########################

from sklearn.linear_model import LogisticRegression

Model = LogisticRegression()
Model.fit(X_train, y_train)

y_pred = Model.predict(X_test)

# Summary of the predictions made by the classifier
print('LogReg\n',sm.classification_report(y_test, y_pred))
print('LogReg\n',sm.confusion_matrix(y_test, y_pred))

# Accuracy score
LogisticRegression_accuracy = sm.accuracy_score(y_pred,y_test)
print('LogReg accuracy is',sm.accuracy_score(y_pred,y_test))

#creating 1D array to 2D array of class image
y_pred = Model.predict(x)
LogReg = np.reshape(y_pred,(AI.shape))   #Predicted class image


##################################### Neural Network ###########################
#####################

from sklearn.neural_network import MLPClassifier
Model=MLPClassifier()
Model.fit(X_train,y_train)
y_pred=Model.predict(X_test)
# Summary of the predictions
print('NN\n',sm.classification_report(y_test,y_pred))
print('NN\n',sm.confusion_matrix(y_test,y_pred))

#Accuracy Score
NN_accuracy = sm.accuracy_score(y_pred,y_test)
print('NN accuracy is ',sm.accuracy_score(y_pred,y_test))


#creating 1D array to 2D array of class image
y_pred = Model.predict(x)
NN = np.reshape(y_pred,(AI.shape))   #Predicted class image


#####################################  Plotting all classification with accurac
y #################################

# Plot the Original Classifications
plt.figure(figsize= (10,50))
plt.subplot(6, 1, 1)
plt.imshow(ai_orig)
plt.title("Real Class Image")

```

```python

# Plot the Models Classifications
plt.subplot(6, 1, 2)
plt.imshow(kmeans)
plt.title('Kmeans')


# Plot the Models Classifications
plt.subplot(6, 1, 3)
plt.imshow(KNN)
plt.title('KNN')
plt.xlabel('accuracy is {} '.format(np.round(KNN_accuracy,3)))

# Plot the Models Classifications
plt.subplot(6, 1, 4)
plt.imshow(SVM)
plt.title('SVM')
plt.xlabel('accuracy is {} '.format(np.round(SVM_accuracy,3)))

# Plot the Models Classifications
plt.subplot(6, 1, 5)
plt.imshow(LogReg)
plt.title('LogReg')
plt.xlabel('accuracy is {} '.format(np.round(LogisticRegression_accuracy,3)))

# Plot the Models Classifications
plt.subplot(6, 1, 6)
plt.imshow(NN)
plt.title('NN')
plt.xlabel('accuracy is {} '.format(np.round(NN_accuracy,3)))

plt.savefig('Class.png')   #Saving output comparison image
```

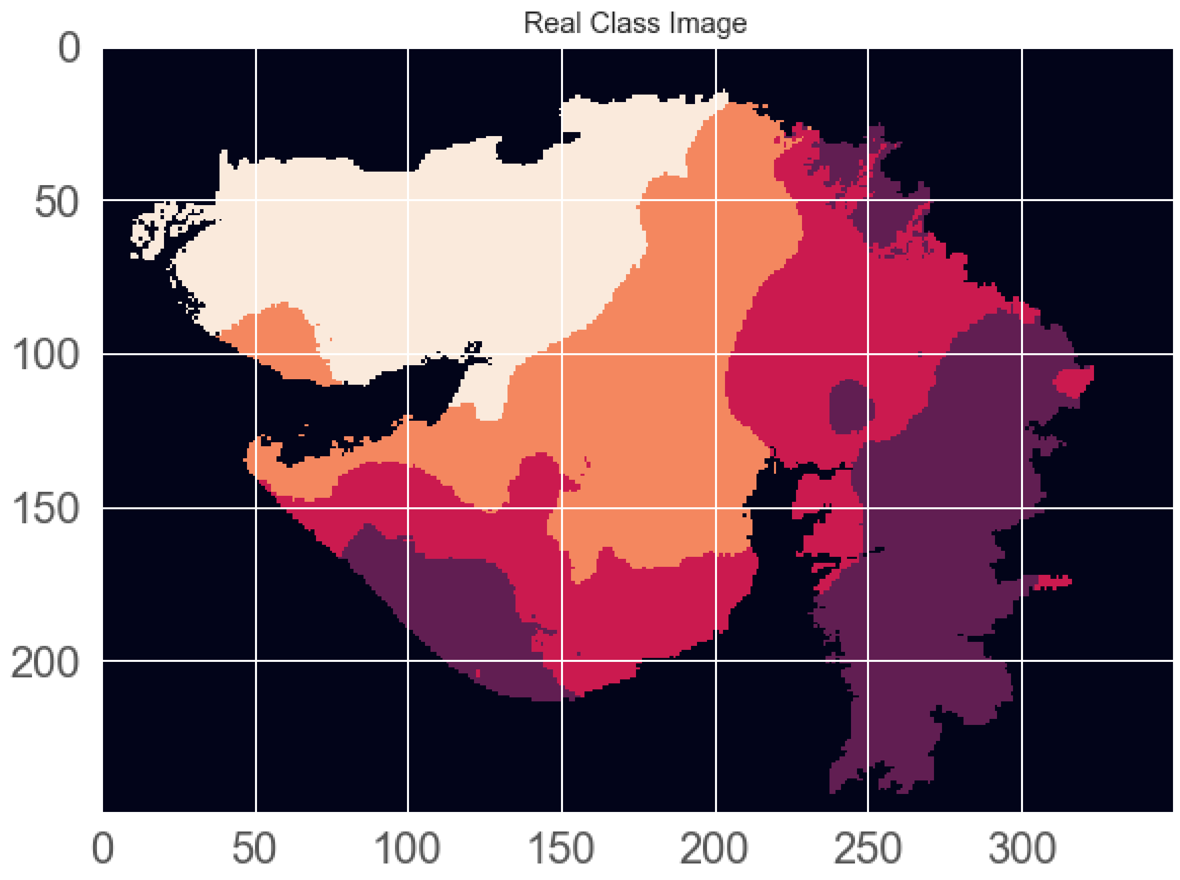
Output :

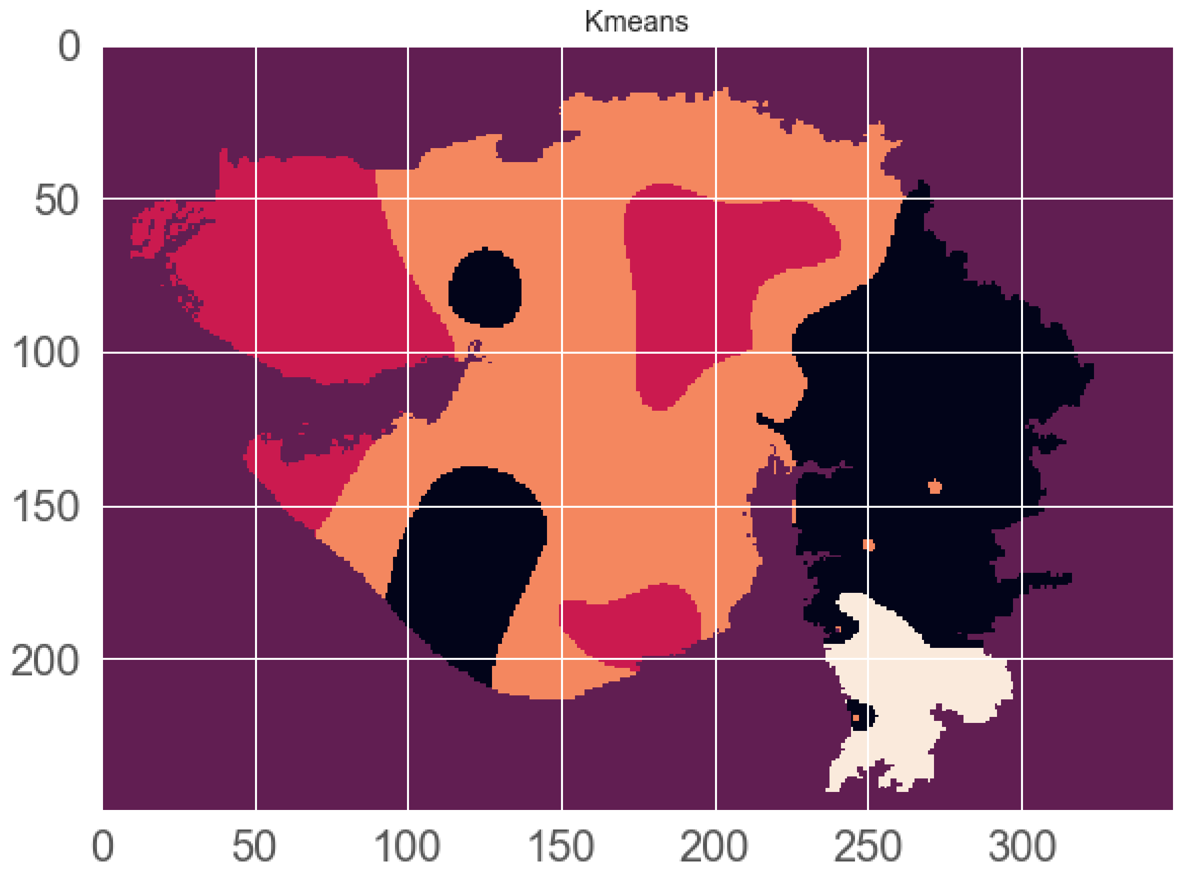
```
Mean squared error for K-means: 1.57

Variance score for K-means: 0.26


KNN

          precision   recall  f1-score   support


     0.0      1.00      0.99      1.00     18937

     1.0      0.93      0.93      0.93      4049

     2.0      0.86      0.83      0.85      4001

     3.0      0.89      0.92      0.91      4052

     4.0      0.94      0.96      0.95      3961


  micro avg      0.96      0.96      0.96     35000
  macro avg       0.92      0.93      0.93     35000
weighted avg       0.96      0.96      0.96     35000


KNN
 [[18800    57    26    12    42]
 [    3  3769   217    16    44]
 [   10   237  3325   336    93]
 [    0     1   232  3745    74]
 [    4     2    49   101  3805]]
KNN accuracy is 0.9555428571428571
SVM
             precision    recall  f1-score   support


        0.0       1.00      0.99      1.00     18937
        1.0       0.94      0.95      0.94      4049
        2.0       0.90      0.89      0.89      4001
        3.0       0.92      0.95      0.93      4052
        4.0       0.96      0.98      0.97      3961

  micro avg       0.97      0.97      0.97     35000
  macro avg       0.94      0.95      0.95     35000
weighted avg       0.97      0.97      0.97     35000


SVM
 [[18797    59    24    13    44]
 [    1  3840   198     2     8]
 [    3   178  3543   247    30]
 [    0     0   152  3830    70]
 [    0    12    20    54  3875]]
SVM accuracy is 0.9681428571428572
```

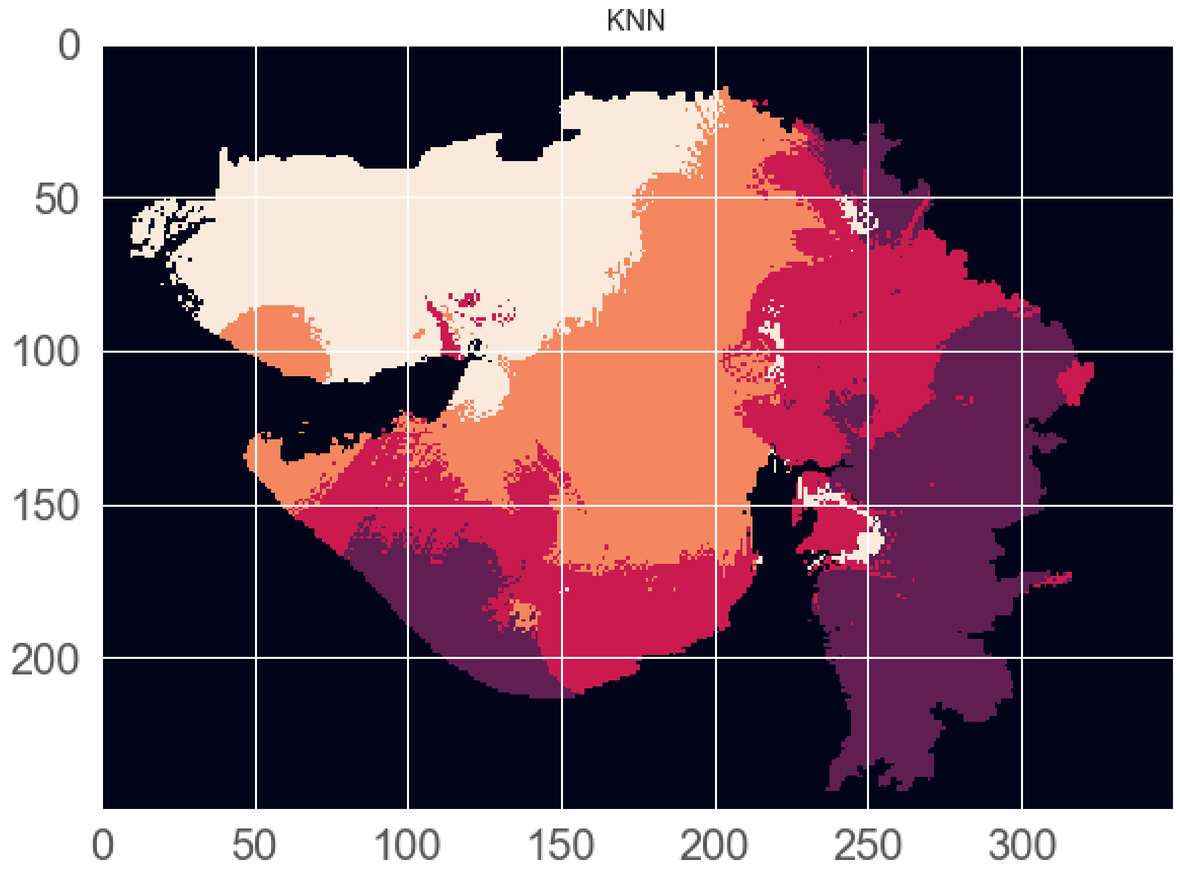
```
LogReg
           precision    recall  f1-score   support

      0.0       1.00      0.99      1.00     18937
      1.0       0.80      0.87      0.83      4049
      2.0       0.67      0.46      0.54      4001
      3.0       0.64      0.66      0.65      4052
      4.0       0.64      0.80      0.71      3961

  micro avg       0.86      0.86      0.86     35000
  macro avg       0.75      0.76      0.75     35000
weighted avg       0.86      0.86      0.85     35000

LogReg
 [[18775    67    24    43    28]
 [    0  3506   330     4   209]
 [    0   650  1839   949   563]
 [    0     0   412  2672   968]
 [    0   133   144   505  3179]]
NN
           precision    recall  f1-score   support

      0.0       1.00      0.99      1.00     18937
      1.0       0.85      0.95      0.90      4049
      2.0       0.73      0.76      0.74      4001
      3.0       0.82      0.81      0.81      4052
      4.0       0.94      0.85      0.89      3961

  micro avg       0.92      0.92      0.92     35000
  macro avg       0.87      0.87      0.87     35000
```

```
weighted avg       0.92      0.92      0.92     35000
```

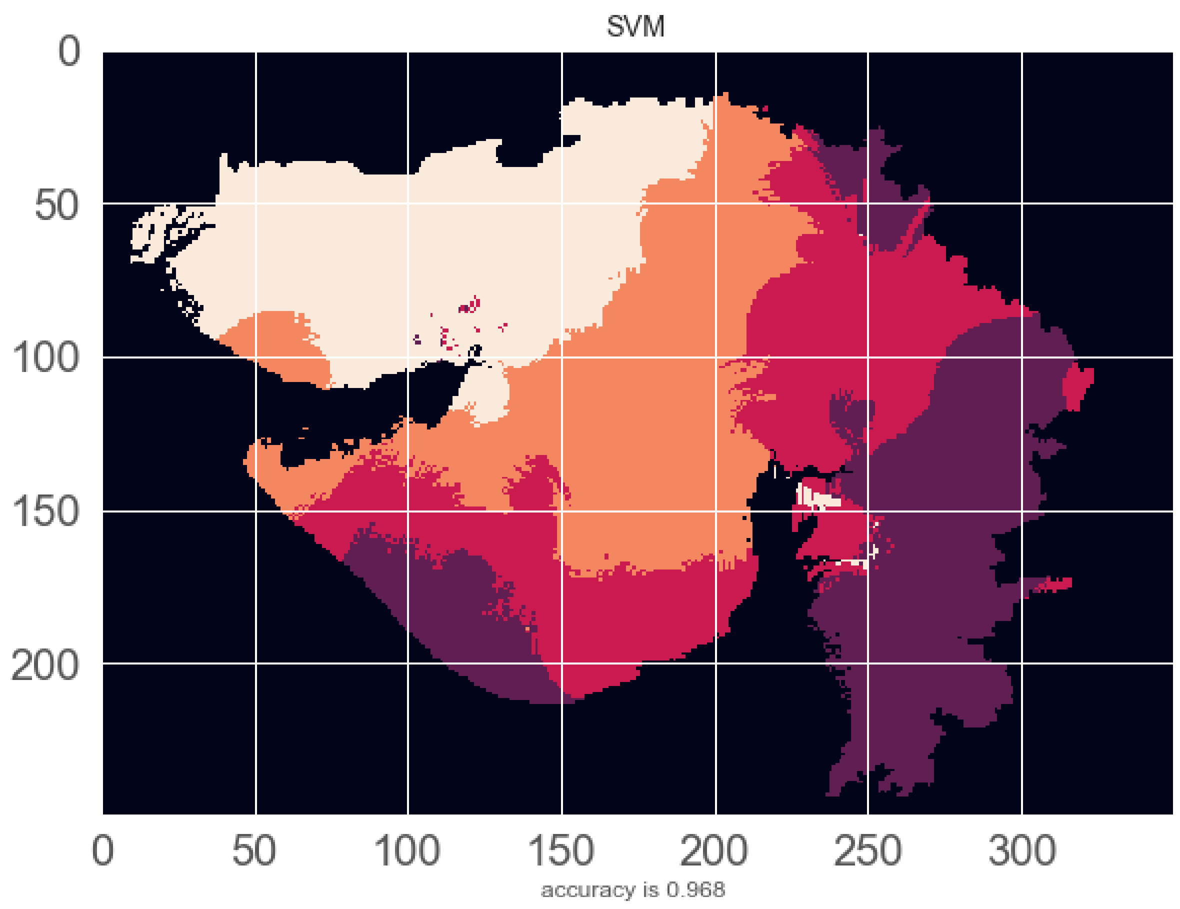
```
NN
 [[18775    64    54    15    29]
 [    0  3857   175     0    17]
 [    0   576  3030   332    63]
 [    0     0   690  3264    98]
 [    0    26   196   378  3361]]
NN accuracy is  0.9224857142857142
```

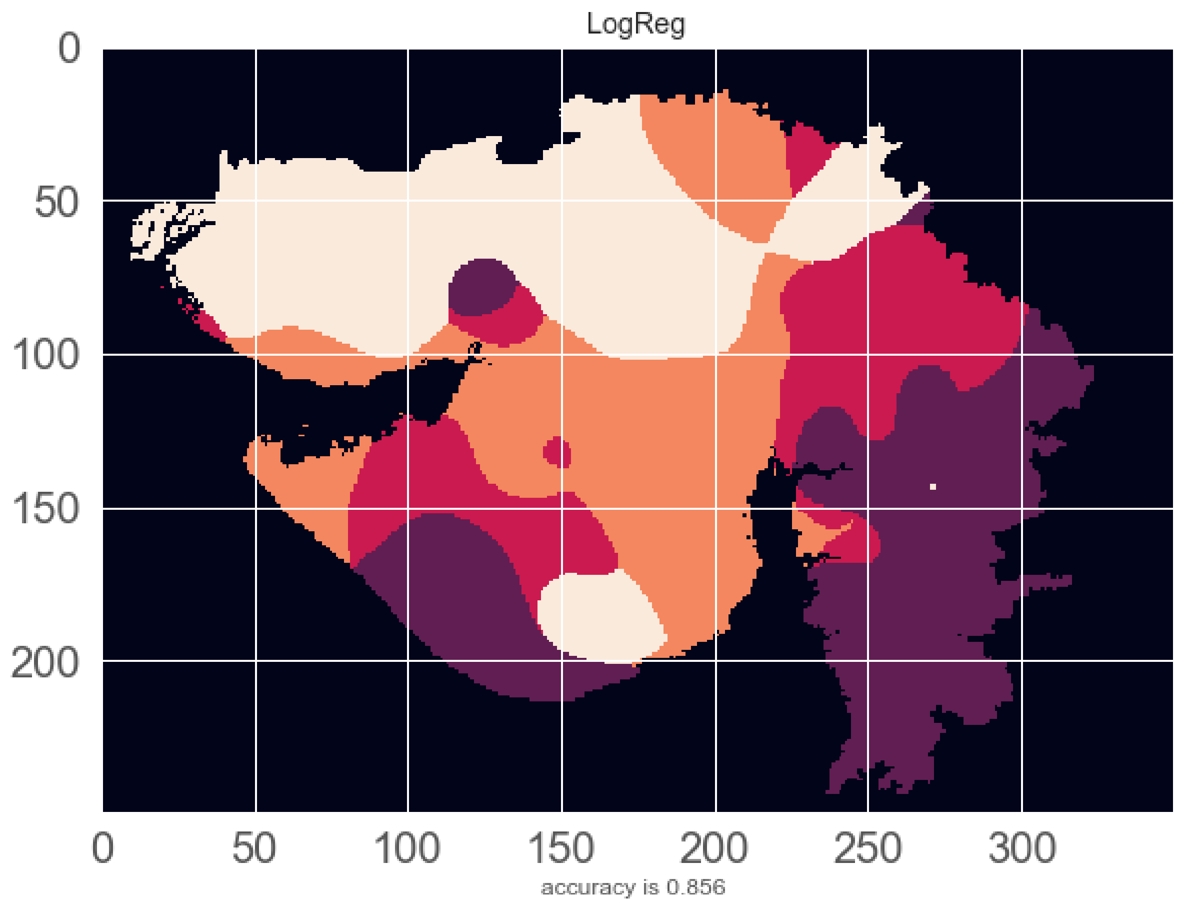


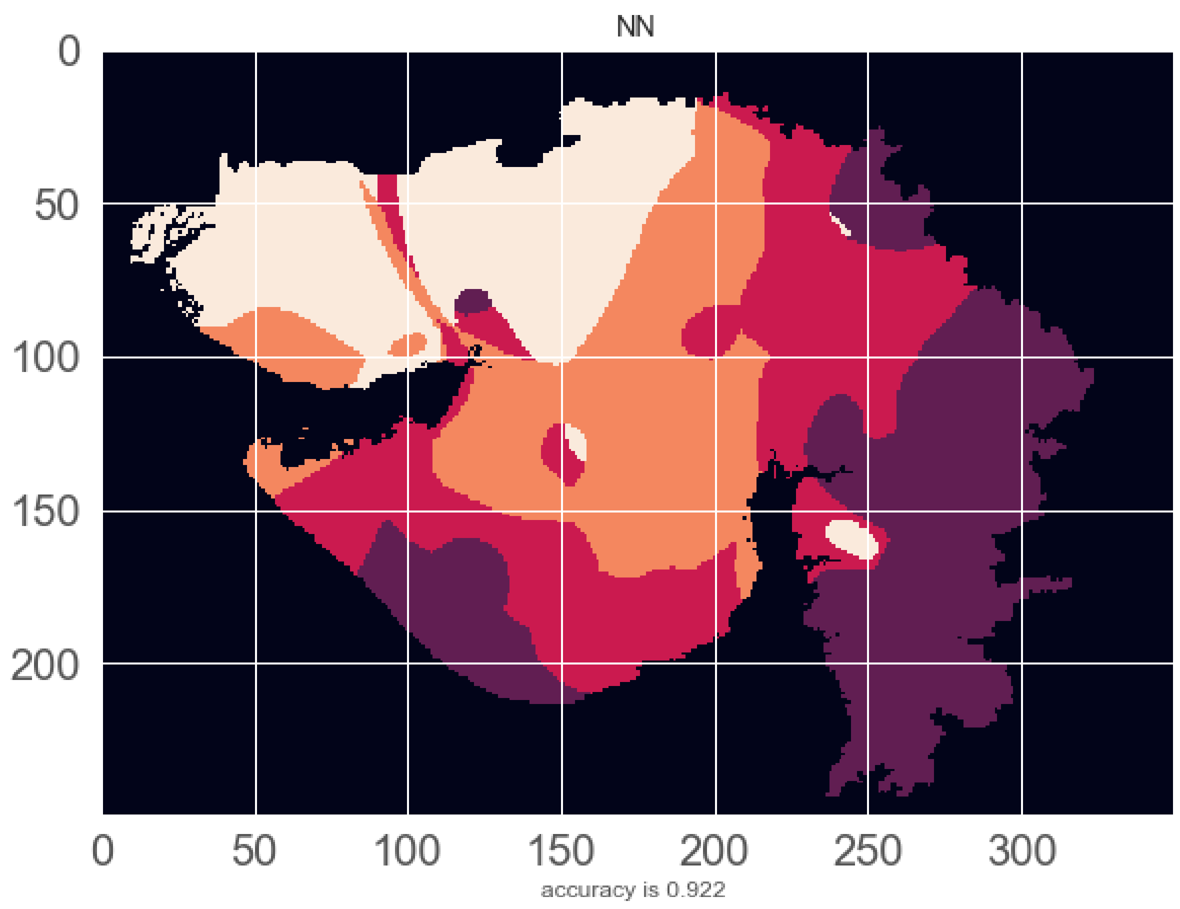




accuracy is 0.956


SVM
0
50
100
150
200
0
50
100
150
200
250
300
accuracy is 0.968


LogReg
0
50
100
150
200
0
50
100
150
200
250
300
accuracy is 0.856



Lab 7 : Tentatively will be used for optimization problem

Lab 8 : Tentatively will be used for exam