# K-MEANS CLUSTERING USING SCIKIT-LEARN

**Objective**

The primary objective of this project is to apply k-means clustering to weather data to identify distinct weather patterns. Specifically, the project aims to:

1. **Load and preprocess** minute-level weather data.
2. **Perform k-means clustering** to group the data into 12 clusters.
3. **Analyze and visualize** the clusters to understand the underlying weather patterns.

**Summary**

The dataset used for this project consists of minute-level weather measurements from a weather station over a three-year period. Each record includes various weather-related variables such as air temperature, air pressure, wind speed, and relative humidity. Due to the large volume of data, we sampled it to make it manageable for clustering.

1. **Data Description**:
    - **Variables**: Includes air pressure, air temperature, wind direction, wind speed, and relative humidity.
    - **Sampling**: Data was sampled by selecting every 10th row to reduce the dataset size.
    - **Preprocessing**: Rows with missing values in rain_accumulation and rain_duration were dropped.
2. **Feature Selection and Scaling**:
    - Selected relevant features for clustering.
    - Scaled features using StandardScaler to standardize the data.
3. **k-Means Clustering**:
    - Applied k-means clustering to partition the data into 12 clusters.
    - Obtained cluster centers and analyzed them to understand the characteristics of each cluster.
4. **Visualization**:
    - Used parallel coordinates plots to visualize and interpret the clusters.
    - Analyzed specific clusters related to dry days, warm days, and cool days based on weather attributes.

**Results**

1. **Data Sampling**:
    - Successfully sampled and cleaned the dataset by removing rows with missing values in specific columns.

2. **Feature Selection and Scaling**:
    - Selected key weather features and scaled them appropriately to ensure effective clustering.
3. **Clustering**:
    - Created 12 distinct clusters using k-means clustering.
    - Cluster centers were computed to represent the average feature values for each cluster.
4. **Visualization**:
    - **Dry Days**: Identified clusters with low relative humidity.
    - **Warm Days**: Identified clusters with high air temperature.
    - **Cool Days**: Identified clusters with high relative humidity and low air temperature.

**Conclusion**

The k-means clustering approach effectively grouped the weather data into distinct clusters, each representing different weather patterns. The analysis provides valuable insights into various types of weather conditions, such as dry, warm, and cool days. These clusters can be used for further analysis or to inform decisions related to weather patterns.

**Code:**

'''

We will learn how to perform k-means clustering using scikit-learn in Python.

 We will use cluster analysis to generate a big picture model of the weather at a local station using a minute-graunlarity data. In this dataset, we have in the order of millions records. How do we create 12 clusters our of them?

'''

```
#Importing the Necessary Libraries
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
from itertools import cycle, islice
import matplotlib.pyplot as plt
from pandas.plotting import parallel_coordinates
#Creating a Pandas DataFrame from a CSV file
data = pd.read_csv("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\PROPER DATASETS Non Edited\\ML Projects and Datasets\\data\\weather.zip",compression = 'zip')
print("Data Shape: ",data.shape)
print("Sample Data: \n",data.head())
```

'''

Minute Weather Data Description

 The minute weather dataset comes from the same source as the daily weather dataset that we used

in the decision tree based classifier notebook. The main difference between these two datasets is that the minute weather dataset contains raw sensor measurements captured at one-minute intervals. Daily weather dataset instead contained processed and well curated data. The data is in the file minute_weather.csv, which is a comma-separated file.

 As with the daily weather data, this data comes from a weather station. The weather station is

equipped with sensors that capture weather-related measurements such as air temperature, air pressure, and relative humidity. Data was collected for a period of three years, from September 2011 to September 2014, to ensure that sufficient data for different seasons and weather conditions is captured.

 Each row in minute_weather.csv contains weather data captured for a one-minute interval. Each row, or sample, consists of the following variables:

o rowID: unique number for each row (Unit: NA)

o hpwren_timestamp: timestamp of measure (Unit: year-month-day hour:minute:second)

o air_pressure: air pressure measured at the timestamp (Unit: hectopascals)

o air_temp: air temperature measure at the timestamp (Unit: degrees Fahrenheit)

o avg_wind_direction: wind direction averaged over the minute before the timestamp (Unit: degrees, with 0 means coming from the North, and increasing clockwise)

o avg_wind_speed: wind speed averaged over the minute before the timestamp (Unit: meters per second)

o max_wind_direction: highest wind direction in the minute before the timestamp (Unit: degrees, with 0 being North and increasing clockwise)

o max_wind_speed: highest wind speed in the minute before the timestamp (Unit: meters per second)

o min_wind_direction: smallest wind direction in the minute before the timestamp (Unit: degrees, with 0 being North and inceasing clockwise)

o min_wind_speed: smallest wind speed in the minute before the timestamp (Unit: meters per second)

o rain_accumulation: amount of accumulated rain measured at the timestamp (Unit: millimeters)

o rain_duration: length of time rain has fallen as measured at the timestamp (Unit: seconds)

o relative_humidity: relative humidity measured at the timestamp (Unit: percent)

'''

###Data Sampling

#Lots of rows, so let us sample down by taking every 10th row.

sampled_df = data[(data['rowID'] % 10) == 0]

```
print("Sample Data Shape: ",sampled_df.shape)
#Statistics
sampled_df.describe().transpose()
sampled_df[sampled_df['rain_accumulation'] == 0].shape
sampled_df[sampled_df['rain_duration'] == 0].shape
#Drop all the Rows with Empty rain_duration and rain_accumulation
del sampled_df['rain_accumulation']
del sampled_df['rain_duration']
rows_before = sampled_df.shape[0]
sampled_df = sampled_df.dropna()
rows_after = sampled_df.shape[0]
#How many rows did we drop ?
print("How many rows did we drop ? ",rows_before - rows_after)
print("Columns in Sample Data: ",sampled_df.columns)

#Select Features of Interest for Clustering
features = ['air_pressure', 'air_temp', 'avg_wind_direction',
'avg_wind_speed', 'max_wind_direction',
 'max_wind_speed','relative_humidity']
select_df = sampled_df[features]
select_df.columns
select_df
#Scale the Features using StandardScaler
X = StandardScaler().fit_transform(select_df)
X

#Use k-Means Clustering
kmeans = KMeans(n_clusters=12)
model = kmeans.fit(X)
print("model\n", model)
#What are the centers of 12 clusters we formed ?
```

```
centers = model.cluster_centers_
print("centers: ", centers)
###Plots
#Let us first create some utility functions which will help us in plotting graphs:
# Function that creates a DataFrame with a column for Cluster Number
def pd_centers(featuresUsed, centers):
    colNames = list(featuresUsed)
    colNames.append('prediction')
    # Zip with a column called 'prediction' (index)
    Z = [np.append(A, index) for index, A in enumerate(centers)]
    # Convert to pandas data frame for plotting
    P = pd.DataFrame(Z, columns=colNames)
    P['prediction'] = P['prediction'].astype(int)
    return P

# Function that creates Parallel Plots
def parallel_plot(data):
    my_colors = list(islice(cycle(['b', 'r', 'g', 'y', 'k']), None,
    len(data)))
    plt.figure(figsize=(15,8)).gca().axes.set_ylim([-3,+3])
    parallel_coordinates(data, 'prediction', color = my_colors,
    marker='o')
    plt.show()

P = pd_centers(features, centers)
print("pd_centers: ",P)

#Output :
###Dry Days
parallel_plot(P[P['relative_humidity'] < -0.5])
###Warm Days
```

```
parallel_plot(P[P['air_temp'] > 0.5])
```

###Cool Days

```
parallel_plot(P[(P['relative_humidity'] > 0.5) & (P['air_temp'] < 0.5)])
```