1: Load and Restructure Data

We will:

1. Extract **longitude** and **latitude** from the first and second rows.
2. Treat the rest of the data as **daily precipitation values** (row 3 onwards).
3. Assign artificial dates starting from a known date (e.g., `1965-01-01`), assuming daily resolution.

Code:

```python
import pandas as pd
import numpy as np

# Load the data
file_path = './precipitation_data.xlsx'
data = pd.read_excel(file_path, header=None)

# Extract longitude and latitude from the first two rows
longitude = data.iloc[0, 1:].values  # Longitude in the first row
latitude = data.iloc[1, 1:].values   # Latitude in the second row

# Extract daily precipitation data
precipitation_values = data.iloc[2:].reset_index(drop=True)

# Generate artificial dates for daily data (assuming daily data from 1965-01-01)
start_date = "1965-01-01"
dates = pd.date_range(start=start_date, periods=len(precipitation_values), freq="D")

# Assign column names: Date + Latitude-Longitude pairs
precipitation_values.columns = ["Date"] + [f"{lat},{lon}" for lat, lon in zip(latitude, longitude)]
precipitation_values["Date"] = dates

# Display the first few rows of the structured data
print(precipitation_values.head())
```

[2] ✓ 7.8s

```
        Date  23.5,92.5  23.5,93.5  24.5,91.5  24.5,92.5  24.5,93.5  \
0 1965-01-01        0.0        0.0        0.0   0.000000   0.000000
1 1965-01-02        0.0        0.0        0.0   0.000000   0.000000
2 1965-01-03        0.0        0.0        0.0   0.000000   0.000000
3 1965-01-04        0.0        0.0        0.0   0.000000   0.000000
4 1965-01-05        0.0        0.0        0.0   0.208477   0.478065

   24.5,94.5  25.5,90.5  25.5,91.5  25.5,92.5  ...  27.5,92.5  27.5,93.5  \
0        0.0        0.0   0.000000   0.000000  ...   0.000000   0.000000
1        0.0        0.0   0.000000   0.000000  ...   0.000000   0.000000
2        0.0        0.0   0.000000   0.000000  ...   0.000000   0.000000
3        0.0        0.0   0.000000   0.000000  ...   0.000000   0.000000
4        0.0        0.0   1.877032   3.606973  ...   7.945834   7.846818

   27.5,94.5  27.5,95.5  27.5,96.5  28.5,91.5  28.5,92.5  28.5,93.5  \
0   0.000000    0.00000   0.000000   0.000000   0.000000   0.000000
1   0.000000    0.00000   0.000000   0.000000   0.000000   0.000000
2   0.000000    0.00000   0.000000   0.000000   0.000000   0.000000
3   1.915927    1.99999   2.000000   0.000000   0.000000   1.507616
4  19.647551   20.19994  20.200001   9.199995   9.193398  17.485004

   28.5,94.5  28.5,95.5
0   0.000000   0.000000
1   0.000000   0.000000
2   0.000000   0.000000
3   1.998025   2.000000
4  20.189138  20.200001

[5 rows x 30 columns]
```

Spaces: 4 CRLF Cell 1 of 1 Go Live

2: Handle Missing Data

Check for missing values and fill them out appropriately:

1. **Interpolation** for small gaps.
2. **Column-wise Mean** for larger gaps.

Code:

```python
# Check for missing values
missing_summary = precipitation_values.isnull().sum()
print("Missing Values Summary:\n", missing_summary)

# Fill missing values with linear interpolation
precipitation_values.iloc[:, 1:] = precipitation_values.iloc[:, 1:].interpolate(method='linear', axis=0)

# Confirm no missing values remain
print("Missing Values After Imputation:", precipitation_values.isnull().sum().sum())
```

✓ 0.0s Python

```
26.5,95.5    0
27.5,90.5    0
27.5,91.5    0
27.5,92.5    0
27.5,93.5    0
27.5,94.5    0
27.5,95.5    0
...
28.5,94.5    0
28.5,95.5    0
dtype: int64
Missing Values After Imputation: 0
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

+ Code + Markdown

No missing data found

3: Aggregate Data

Aggregate daily precipitation data into:

1. **Monthly Averages**: Average precipitation per month.
2. **Seasonal Averages**: Group data into seasons (e.g., Winter, Summer, Monsoon).I have done division as 4 months for each season for easier data handling.

Code:

```python
# Ensure the 'Date' column is in datetime format
precipitation_values["Date"] = pd.to_datetime(precipitation_values["Date"])

# Set 'Date' as the index
precipitation_values.set_index("Date", inplace=True)

# Define the function to assign seasons
def assign_season(month):
    if month in [11, 12, 1, 2]:  # November to February -> Winter
        return "Winter"
    elif month in [3, 4, 5, 6]:  # March to June -> Summer
        return "Summer"
    elif month in [7, 8, 9, 10]:  # July to October -> Monsoon
        return "Monsoon"

# Create a new column for 'Season'
precipitation_values["Season"] = precipitation_values.index.month.map(assign_season)

# Group data by 'Season' and calculate mean precipitation
seasonal_data = precipitation_values.groupby("Season").mean()

# Display the seasonal data
print(seasonal_data)


seasonal_data.to_csv("seasonal_precipitation.csv")
```

[5] ✓ 0.4s

```
         23.5,92.5  23.5,93.5  24.5,91.5  24.5,92.5  24.5,93.5  24.5,94.5  \
Season
Monsoon   8.513811   6.742806   8.795605   7.989484   5.975378   5.759271
Summer    8.485887   5.944070   8.757782   7.830801   4.793596   4.396407
Winter    0.847868   0.829585   0.779704   0.768388   0.661323   0.722740

         25.5,90.5  25.5,91.5  25.5,92.5  25.5,93.5  ...  27.5,92.5  \
Season                                               ...
Monsoon  14.029250   8.498672   6.657661   6.227189  ...   7.579940
Summer   11.272165   7.808263   5.565561   4.984542  ...   6.766455
Winter    0.428707   0.617756   0.560463   0.603360  ...   0.509372

         27.5,93.5  27.5,94.5  27.5,95.5  27.5,96.5  28.5,91.5  28.5,92.5  \
Season
Monsoon   7.546130  15.738563  15.746679  15.604741   8.534784   8.209897
Summer    6.625324  11.899112  11.844120  11.692928   7.397011   7.199412
Winter    0.558541   1.456492   1.500115   1.539522   0.490071   0.520308

         28.5,93.5  28.5,94.5  28.5,95.5
Season
Monsoon  13.455564  16.093256  15.896138
Summer   10.423886  12.165344  11.987959
Winter    1.124356   1.451109   1.473341

[3 rows x 29 columns]
```

```python
import pandas as pd

# Ensure Date column exists and is set as the index
start_date = "1965-01-01"
dates = pd.date_range(start=start_date, periods=len(precipitation_values), freq="D")
precipitation_values.insert(0, "Date", dates)
precipitation_values["Date"] = pd.to_datetime(precipitation_values["Date"])
precipitation_values.set_index("Date", inplace=True)

# Exclude the 'Season' column
numeric_data = precipitation_values.drop(columns=["Season"])

# Aggregate daily data into monthly averages
monthly_data = numeric_data.resample("M").mean()

# Save the aggregated data
monthly_data.to_csv("monthly_precipitation_data_cleaned.csv")

# Display the first few rows of the monthly data
print(monthly_data.head())
```

✓ 0.0s

Pythor

4: Trend Analysis

Performing trend analysis to identify increasing or decreasing precipitation trends:

1. Use **Mann-Kendall Test** for significance.
2. Use **Sen's Slope Estimator** to quantify trends.

```
import pymannkendall as mk

# Example: Analyze trend for a single grid point
grid_point = monthly_data.iloc[:, 0]  # Select the first grid point for analysis

# Perform Mann-Kendall Test
trend_result = mk.original_test(grid_point)


#print("Trend:", trend_result.trend)   Increasing, Decreasing, or No trend
#print("p-value:", trend_result.p)    # Statistical significance
#print("Slope:", trend_result.slope)  # Rate of change
# Perform trend analysis for all grid points
trend_results = {}
for column in monthly_data.columns:
    trend_result = mk.original_test(monthly_data[column])
    trend_results[column] = {
        "Trend": trend_result.trend,
        "p-value": trend_result.p,
        "Slope": trend_result.slope,
    }

# Convert the results to a DataFrame
trend_results_df = pd.DataFrame.from_dict(trend_results, orient="index")
print(trend_results_df.head())
```

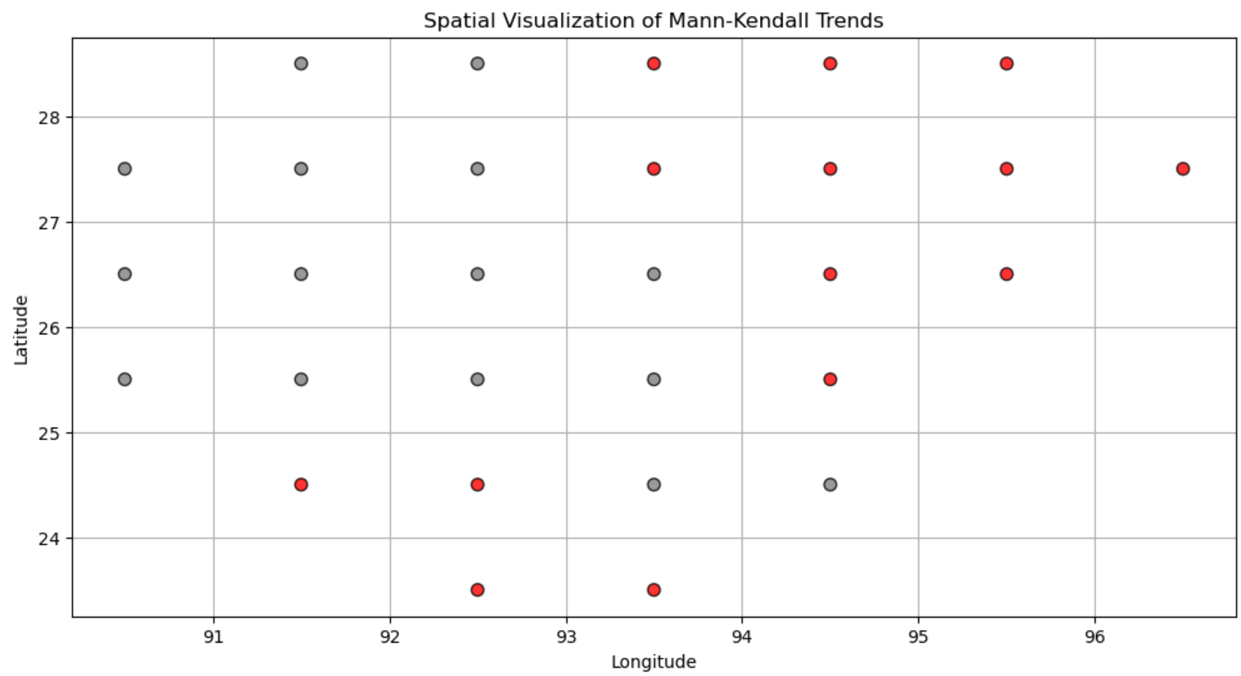
The **p-value** indicates the probability that the observed trend is due to random chance.A **small p-value** suggests a statistically significant trend.(generally people take <0.05).The **Sen's Slope** (mm/month) is the **median** of all the individual slopes.

```
  Grid_Point       Trend   p-value     Slope  Latitude  Longitude
0  23.5,92.5  decreasing  0.019068 -0.001485      23.5       92.5
1  23.5,93.5  decreasing  0.007702 -0.001467      23.5       93.5
2  24.5,91.5  decreasing  0.044332 -0.001142      24.5       91.5
3  24.5,92.5  decreasing  0.029620 -0.001130      24.5       92.5
4  24.5,93.5    no trend  0.156908 -0.000595      24.5       93.5
```

I have also plotted visually to get the trend-

```python
# Assign colors based on trend type
color_map = {
    "increasing": "blue",
    "decreasing": "red",
    "no trend": "gray",
}
```

Grid_Point	Trend	p-value	Slope	Latitude	Longitude	Color
23.5,92.5	decreasing	0.019068322	-0.0014849	23.5	92.5	red
23.5,93.5	decreasing	0.007701657	-0.0014669	23.5	93.5	red
24.5,91.5	decreasing	0.044331742	-0.001142	24.5	91.5	red
24.5,92.5	decreasing	0.029620323	-0.0011298	24.5	92.5	red
24.5,93.5	no trend	0.156908476	-0.000595	24.5	93.5	gray
24.5,94.5	no trend	0.089383574	-0.0007008	24.5	94.5	gray
25.5,90.5	no trend	0.070305407	-0.0005442	25.5	90.5	gray
25.5,91.5	no trend	0.782043272	-0.0001052	25.5	91.5	gray
25.5,92.5	no trend	0.092172361	-0.0007319	25.5	92.5	gray
25.5,93.5	no trend	0.15014508	-0.0005811	25.5	93.5	gray
25.5,94.5	decreasing	0.04595954	-0.0008142	25.5	94.5	red
26.5,90.5	no trend	0.273272975	-0.0002288	26.5	90.5	gray
26.5,91.5	no trend	0.599594695	-0.0002205	26.5	91.5	gray
26.5,92.5	no trend	0.244196284	-0.0004899	26.5	92.5	gray
26.5,93.5	no trend	0.297870496	-0.0004304	26.5	93.5	gray
26.5,94.5	decreasing	0.012857616	-0.0014205	26.5	94.5	red
26.5,95.5	decreasing	0	-0.0117546	26.5	95.5	red
27.5,90.5	no trend	0.195684802	-0.000427	27.5	90.5	gray
27.5,91.5	no trend	0.388687536	-0.0003481	27.5	91.5	gray
27.5,92.5	no trend	0.145698318	-0.0006255	27.5	92.5	gray
27.5,93.5	decreasing	0.013833266	-0.001143	27.5	93.5	red
27.5,94.5	decreasing	1.55E-06	-0.0051951	27.5	94.5	red
27.5,95.5	decreasing	1.83E-09	-0.0060612	27.5	95.5	red
27.5,96.5	decreasing	9.45E-10	-0.0062149	27.5	96.5	red
28.5,91.5	no trend	0.791445751	-9.17E-05	28.5	91.5	gray
28.5,92.5	no trend	0.652439481	-0.0001725	28.5	92.5	gray
28.5,93.5	decreasing	2.27E-07	-0.0049766	28.5	93.5	red
28.5,94.5	decreasing	1.85E-07	-0.0055993	28.5	94.5	red
28.5,95.5	decreasing	2.89E-09	-0.0058803	28.5	95.5	red

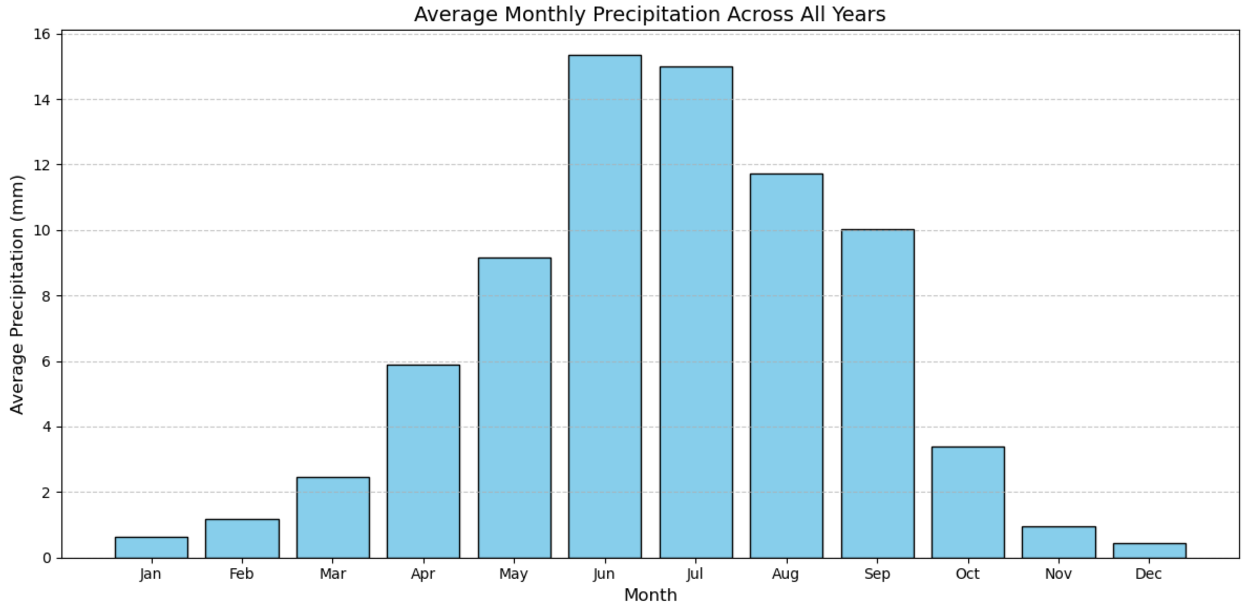


5: Temporal distribution

Monthly wise-

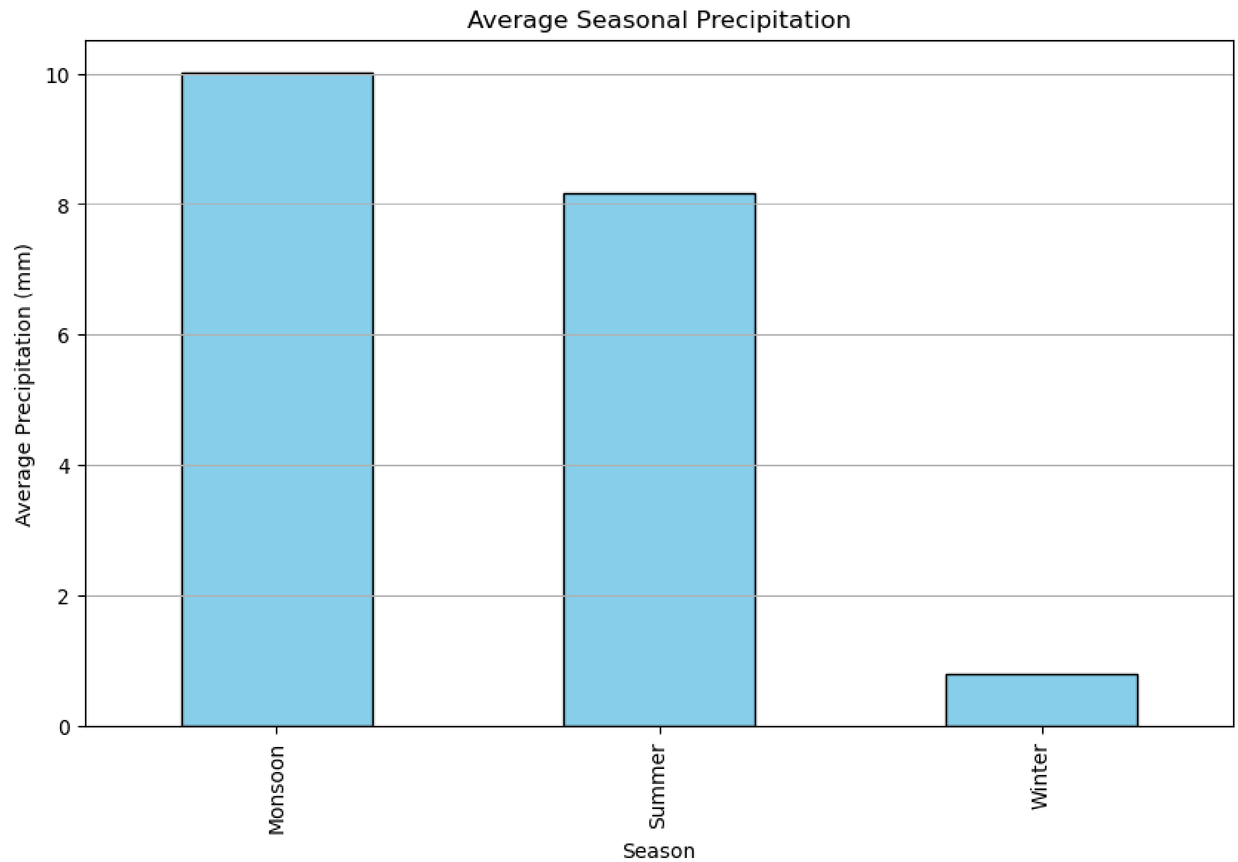
```
# Plot the average monthly precipitation
plt.figure(figsize=(12, 6))
plt.bar(
    x=range(1, 13),
    height=monthly_pattern,
    color="skyblue",
    edgecolor="black"
)
plt.title("Average Monthly Precipitation Across All Years", fontsize=14)
plt.xlabel("Month", fontsize=12)
plt.ylabel("Average Precipitation (mm)", fontsize=12)
plt.xticks(range(1, 13), [
    "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
], fontsize=10)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.tight_layout()
plt.show()
```

✓ 0.2s



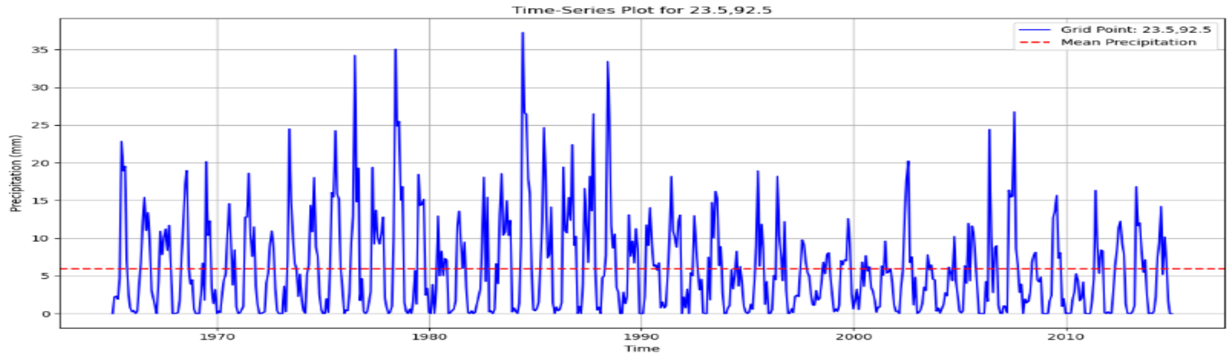
Season wise-

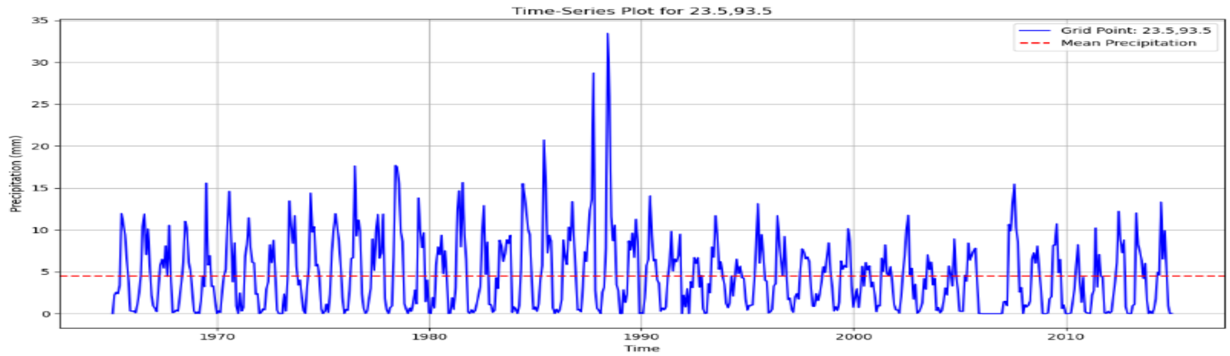
```
# Plot seasonal averages
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
seasonal_data.mean(axis=1).plot(kind="bar", color="skyblue", edgecolor="black")
plt.title("Average Seasonal Precipitation")
plt.xlabel("Season")
plt.ylabel("Average Precipitation (mm)")
plt.grid(axis="y")
plt.show()
```

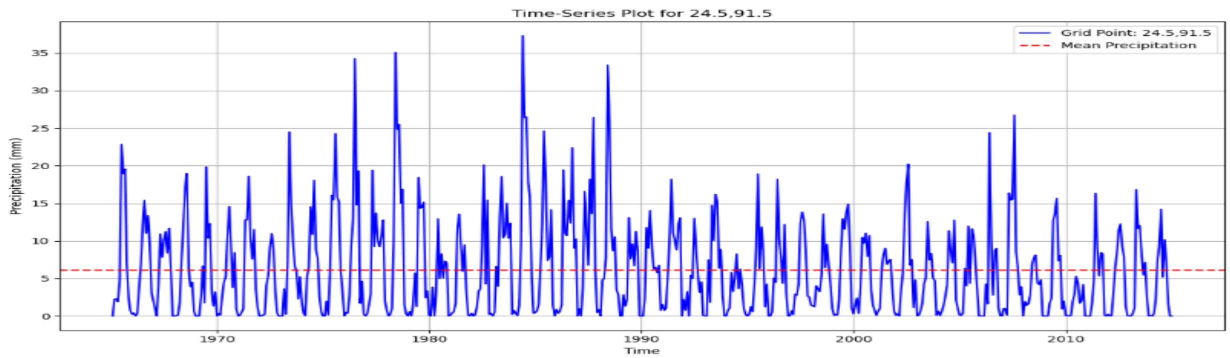


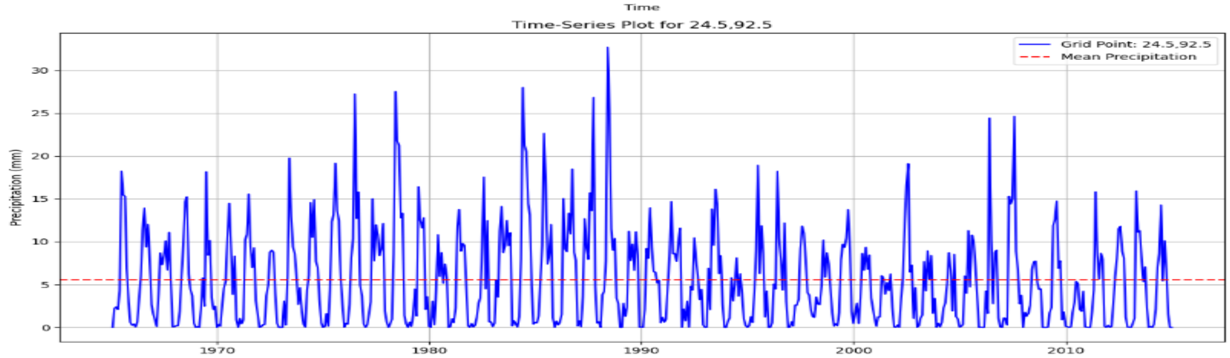
We can also draw the time-series plots for individual grid points-

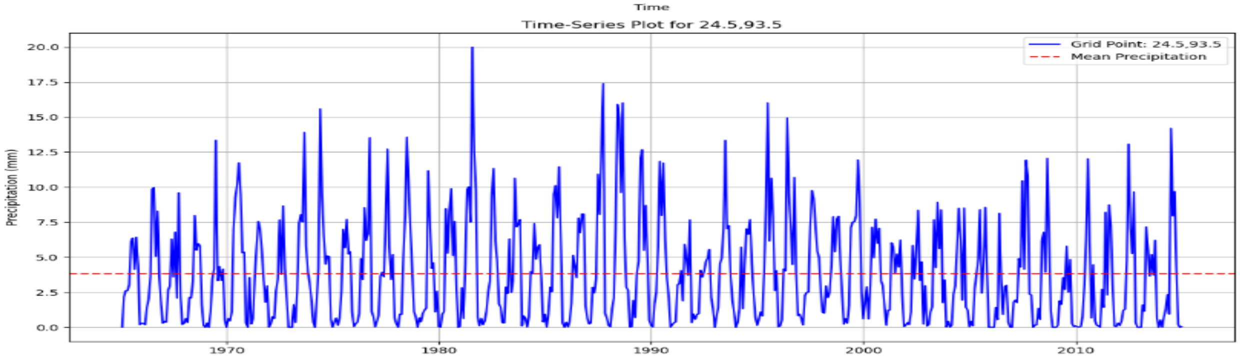
```
import matplotlib.pyplot as plt

# Number of grid points to visualize
num_plots = 5

# Create a figure for the plots
plt.figure(figsize=(12, 6 * num_plots))

# Loop through selected grid points
for i, column in enumerate(monthly_data.columns[:num_plots], start=1):
    plt.subplot(num_plots, 1, i)  # Create subplots
    plt.plot(monthly_data.index, monthly_data[column], label=f"Grid Point: {column}", color="blue")
    plt.axhline(monthly_data[column].mean(), color="red", linestyle="--", label="Mean Precipitation")
    plt.title(f"Time-Series Plot for {column}")
    plt.xlabel("Time")
    plt.ylabel("Precipitation (mm)")
    plt.legend()
    plt.grid()

# Adjust layout to avoid overlapping
plt.tight_layout()
plt.show()
```


Time-Series Plot for 23.5,92.5
Grid Point: 23.5,92.5
Mean Precipitation
Precipitation (mm)
Time
0
5
10
15
20
25
30
35
1970
1980
1990
2000
2010


Time-Series Plot for 23.5,93.5
Grid Point: 23.5,93.5
Mean Precipitation
Precipitation (mm)
Time
0
5
10
15
20
25
30
35
1970
1980
1990
2000
2010


Time-Series Plot for 24.5,91.5
Grid Point: 24.5,91.5
Mean Precipitation
Precipitation (mm)
Time
0
5
10
15
20
25
30
35
1970
1980
1990
2000
2010


Time-Series Plot for 24.5,92.5
Grid Point: 24.5,92.5
Mean Precipitation
Precipitation (mm)
Time
0
5
10
15
20
25
30
1970
1980
1990
2000
2010

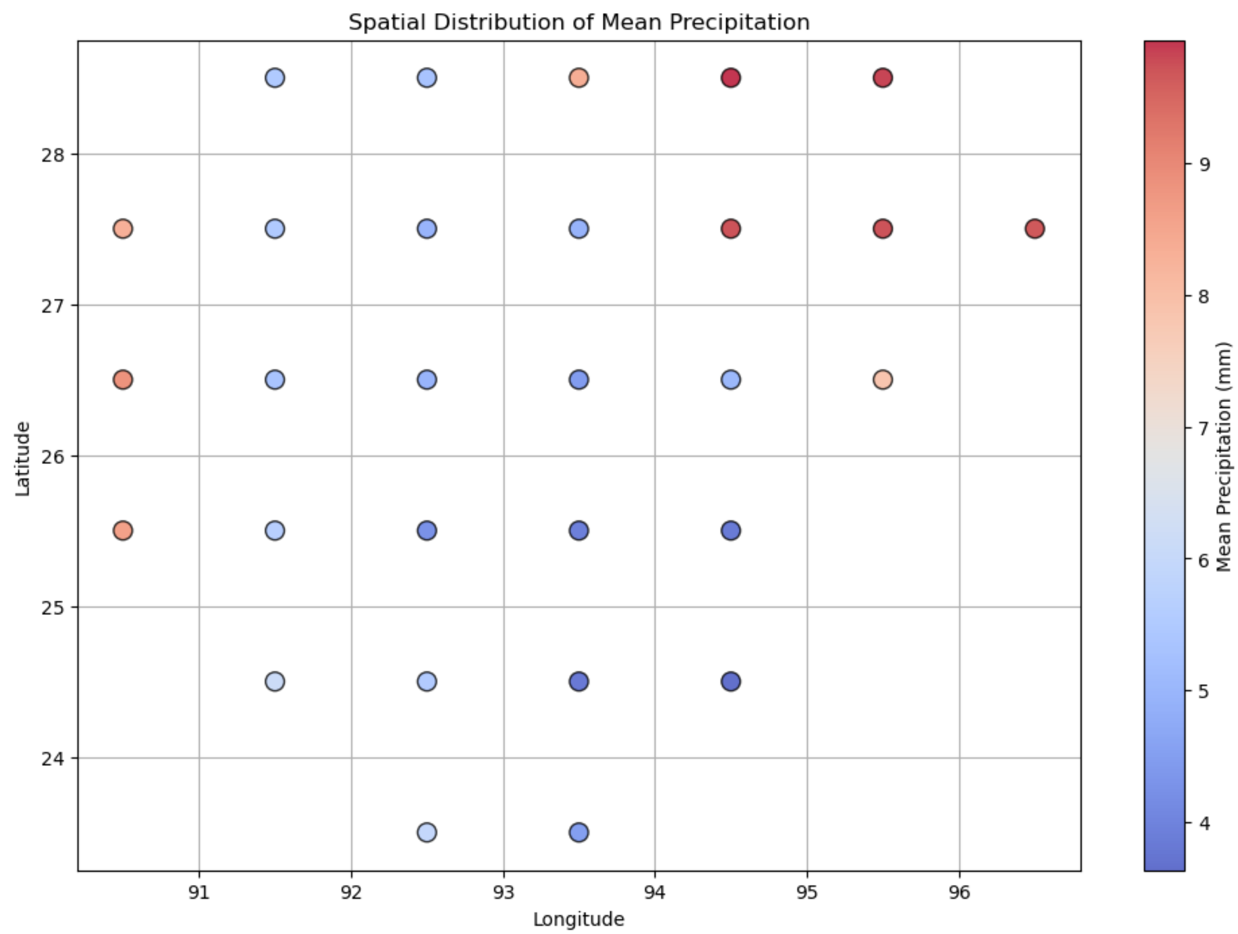

Time-Series Plot for 24.5,93.5
Grid Point: 24.5,93.5
Mean Precipitation
Precipitation (mm)
Time
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
1970
1980
1990
2000
2010

5: Spatial distribution

```python
import matplotlib.pyplot as plt

# Extract longitude and latitude from the grid points
trend_results_df["Longitude"] = trend_results_df["Grid_Point"].apply(lambda x: float(x.split(",")[1]))
trend_results_df["Latitude"] = trend_results_df["Grid_Point"].apply(lambda x: float(x.split(",")[0]))

# Calculate mean precipitation for each grid point
mean_precipitation = monthly_data.mean()

# Add mean precipitation to trend_results_df
trend_results_df["Mean_Precipitation"] = trend_results_df["Grid_Point"].map(mean_precipitation)

# Scatter plot for spatial distribution
plt.figure(figsize=(12, 8))
sc = plt.scatter(
    trend_results_df["Longitude"],
    trend_results_df["Latitude"],
    c=trend_results_df["Mean_Precipitation"],
    cmap="coolwarm",
    s=100,
    alpha=0.8,
    edgecolor="black"
)
plt.colorbar(sc, label="Mean Precipitation (mm)")
plt.title("Spatial Distribution of Mean Precipitation")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.show()
```

[75]

0 Ln 23, Col 1 Spaces: 4 Spaces: 4 CRLF Cell 15 of 17 Go Live



Step 6: Clustering

1.K means-

-To identify regions with similar precipitation characteristics

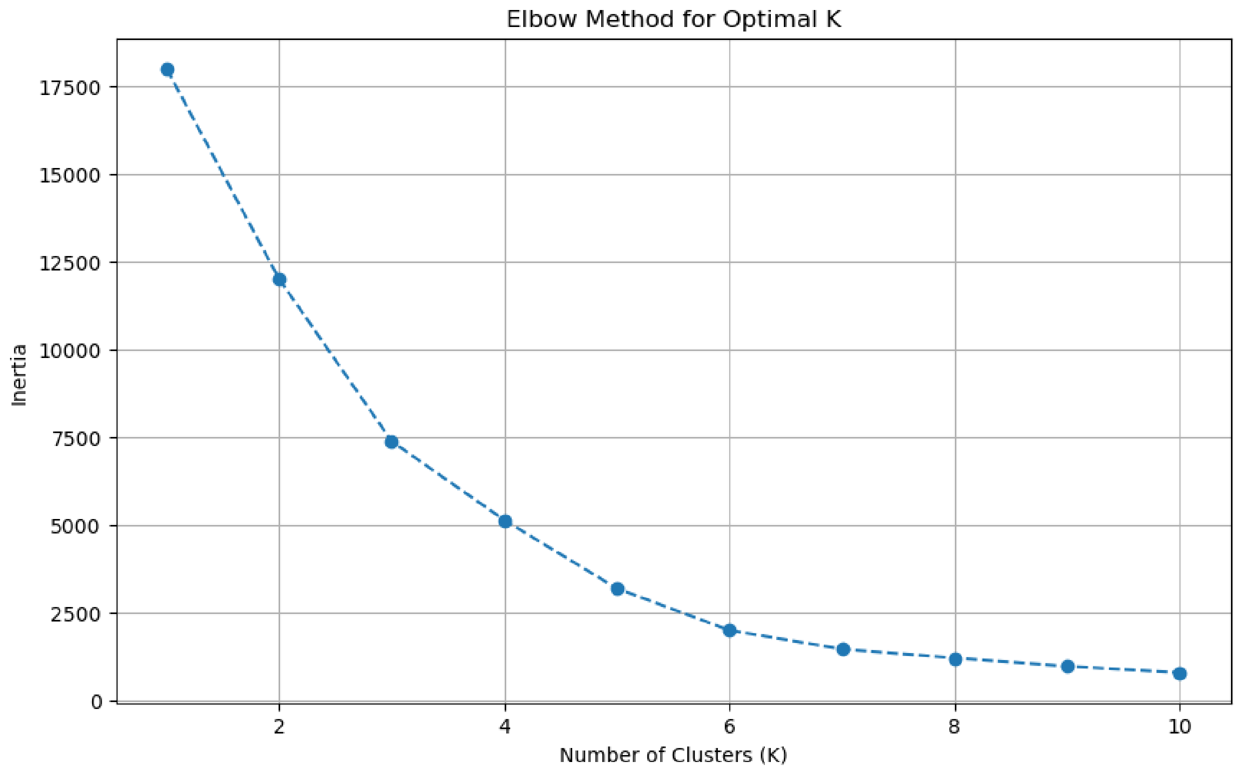
-To detect seasonal and spatial variations.

-To assess anomalies and outliers, such as areas experiencing excess precipitation.

- To compare regions' vulnerability to climate change by observing shifting clusters.

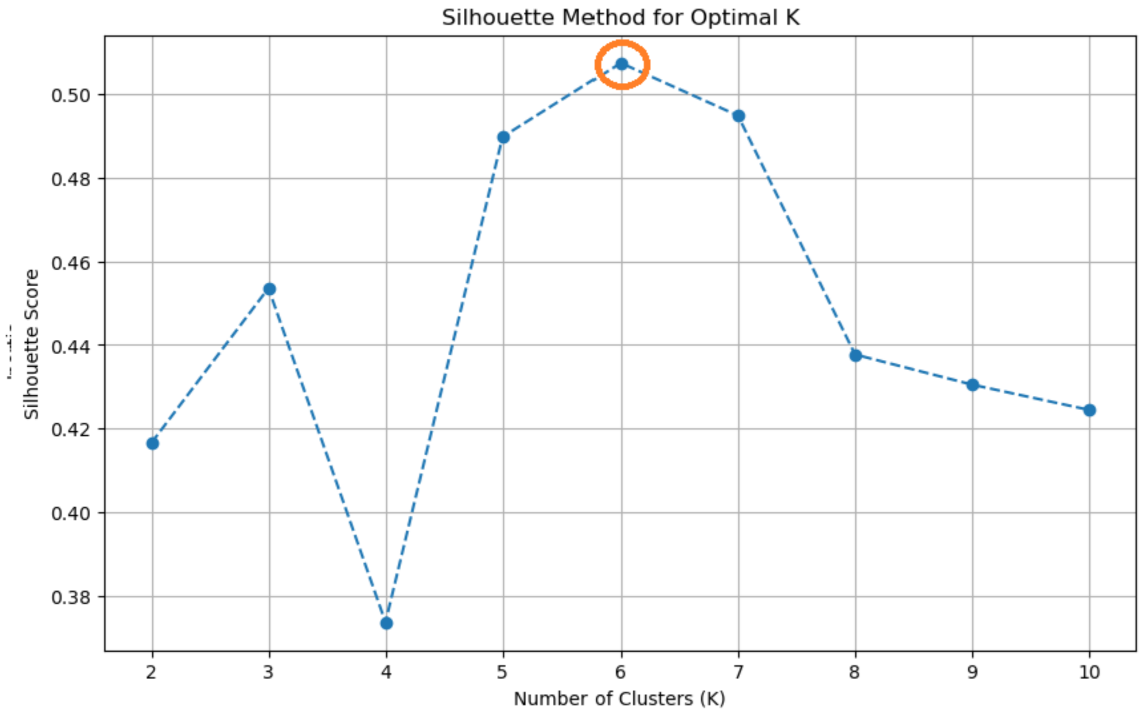
Determine Optimal K Using the Elbow Method

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Step 1: Prepare Data (Transpose the dataset to make grid points as rows)
clustering_data = monthly_data.T

# Step 2: Normalize Data
scaler = StandardScaler()
clustering_data_scaled = scaler.fit_transform(clustering_data)

# Step 3: Calculate Inertia for Different K Values
inertia = []
k_values = range(1, 11)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(clustering_data_scaled)
    inertia.append(kmeans.inertia_)

# Step 4: Plot the Elbow Curve
plt.figure(figsize=(10, 6))
plt.plot(k_values, inertia, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()
```



I wasn’t sure because k lies between 5-6 ,so i even applied silhouette method to find optimal k;

```
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Calculate silhouette scores for different k values
silhouette_scores = []
k_values = range(2, 11)  # Silhouette requires at least 2 clusters

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(clustering_data_scaled)
    silhouette_avg = silhouette_score(clustering_data_scaled, cluster_labels)
    silhouette_scores.append(silhouette_avg)

# Plot the silhouette scores
plt.figure(figsize=(10, 6))
plt.plot(k_values, silhouette_scores, marker='o', linestyle='--')
plt.title('Silhouette Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.grid(True)
plt.show()
```




```python
# Optimal number of clusters from the elbow plot (replace `optimal_k` with your selected value)
optimal_k = 6

# Apply K-Means with the selected number of clusters
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(clustering_data_scaled)

# Add cluster labels to the data
clustering_data["Cluster"] = clusters

# Display the cluster counts
print(clustering_data["Cluster"].value_counts())
```

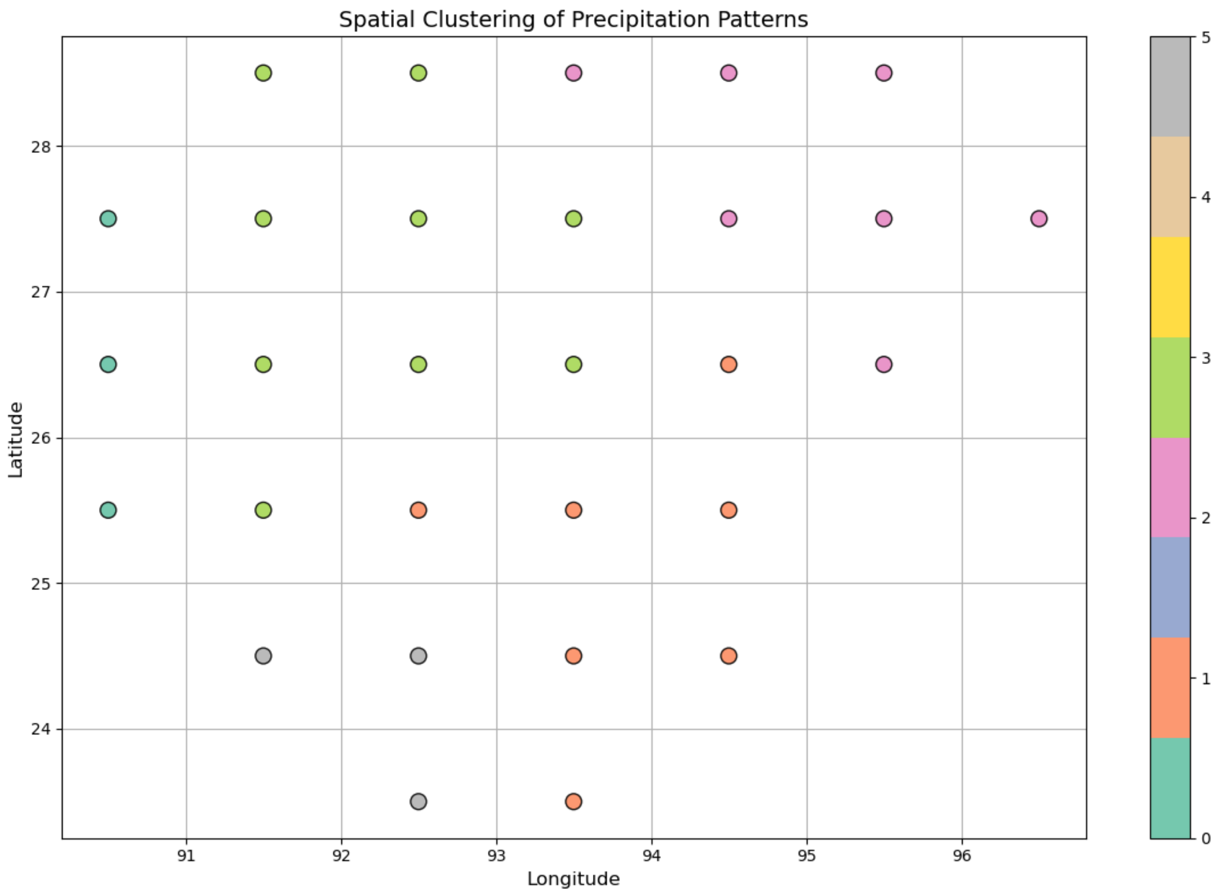
✓ 0.5s

```
d:\Staad\PictoBlox\Lib\site
  warnings.warn(
Cluster
3    9
1    7
2    7
5    3
0    3
4    1
Name: count, dtype: int64
```


```
Cluster
3    9
1    7
2    7
5    3
0    3
4    1
Name: count, dtype: int64
```


Plotting clusters-

```
# Extract longitude and latitude
clustering_data["Longitude"] = clustering_data["index"].apply(lambda x: float(x.split(",")[1]))
clustering_data["Latitude"] = clustering_data["index"].apply(lambda x: float(x.split(",")[0]))

import matplotlib.pyplot as plt

# Scatter plot for spatial clustering with 6 distinct colors
plt.figure(figsize=(12, 8))
sc = plt.scatter(
    clustering_data["Longitude"],
    clustering_data["Latitude"],
    c=clustering_data["Cluster"],
    cmap="Set2",  # Use the 'Set2' colormap for distinct, visually appealing colors
    s=100,
    alpha=0.9,
    edgecolor="black"
)
plt.colorbar(sc, label="Cluster")
plt.title("Spatial Clustering of Precipitation Patterns", fontsize=14)
plt.xlabel("Longitude", fontsize=12)
plt.ylabel("Latitude", fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
```

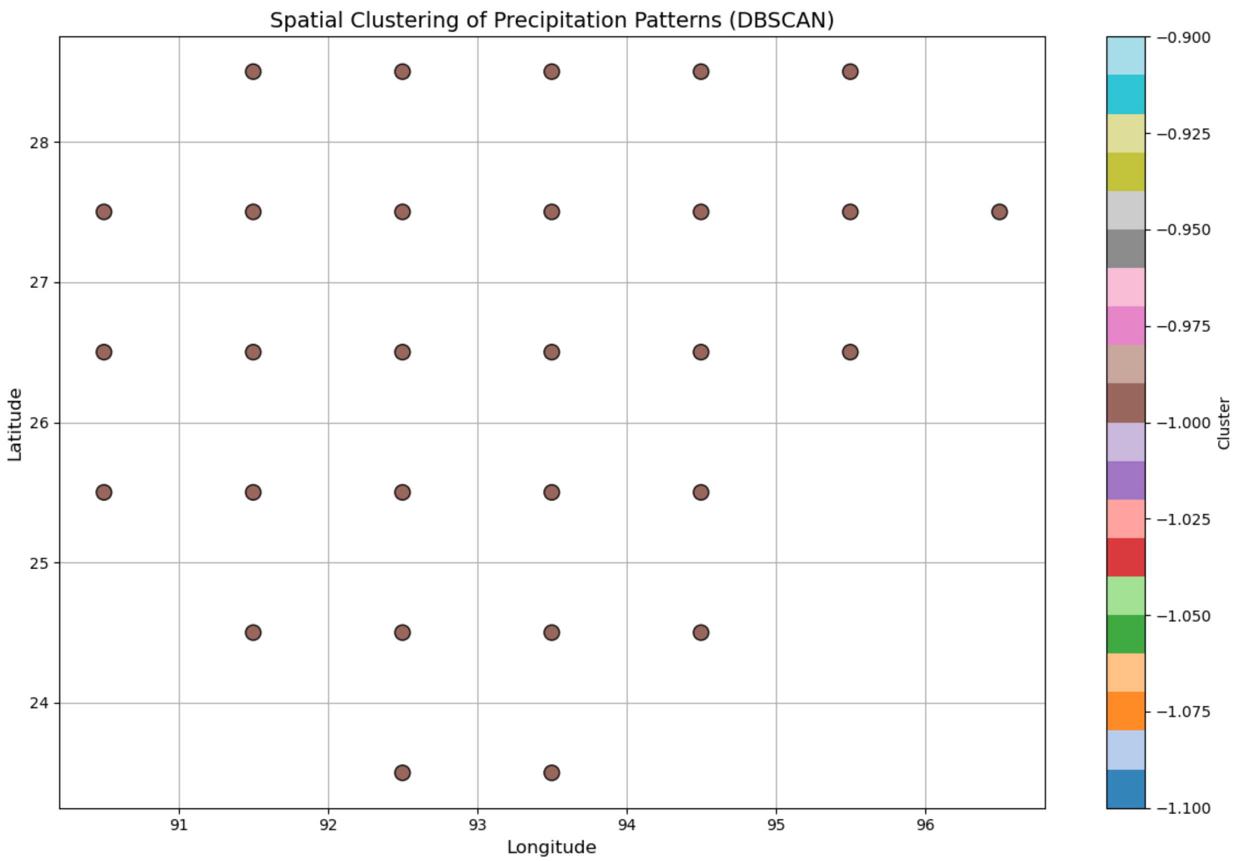


Advantages of the k means clustering-

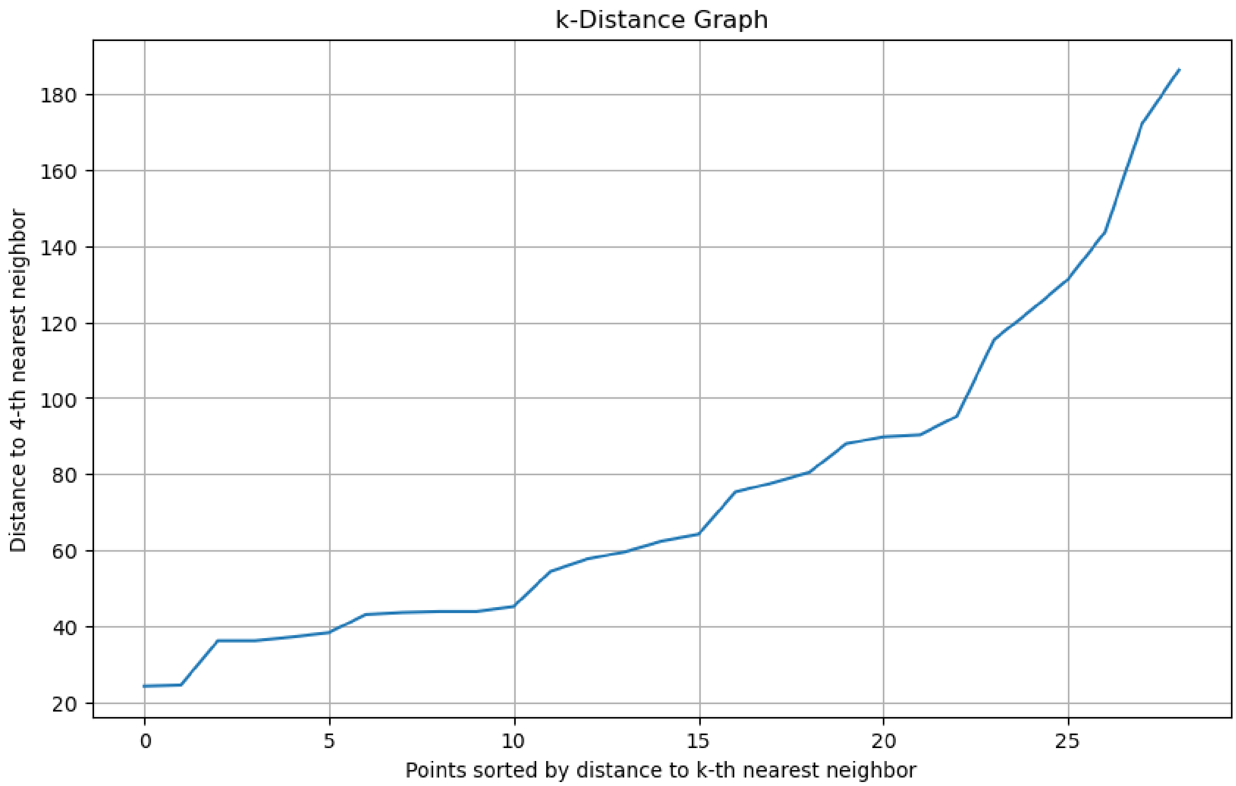
DBSCAN

```python
# Verify extracted columns
print(precipitation_data[["Longitude", "Latitude"]].head())
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

dbscan = DBSCAN(eps=0.5, min_samples=8)

plt.figure(figsize=(12, 8))
sc = plt.scatter(
    precipitation_data["Longitude"],
    precipitation_data["Latitude"],
    c=precipitation_data["Cluster"],
    cmap="tab20",  # Use distinct colors for clusters
    s=100,
    alpha=0.9,
    edgecolor="black"
)
plt.colorbar(sc, label="Cluster")
plt.title("Spatial Clustering of Precipitation Patterns (DBSCAN)", fontsize=14)
plt.xlabel("Longitude", fontsize=12)
plt.ylabel("Latitude", fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
```

[122]



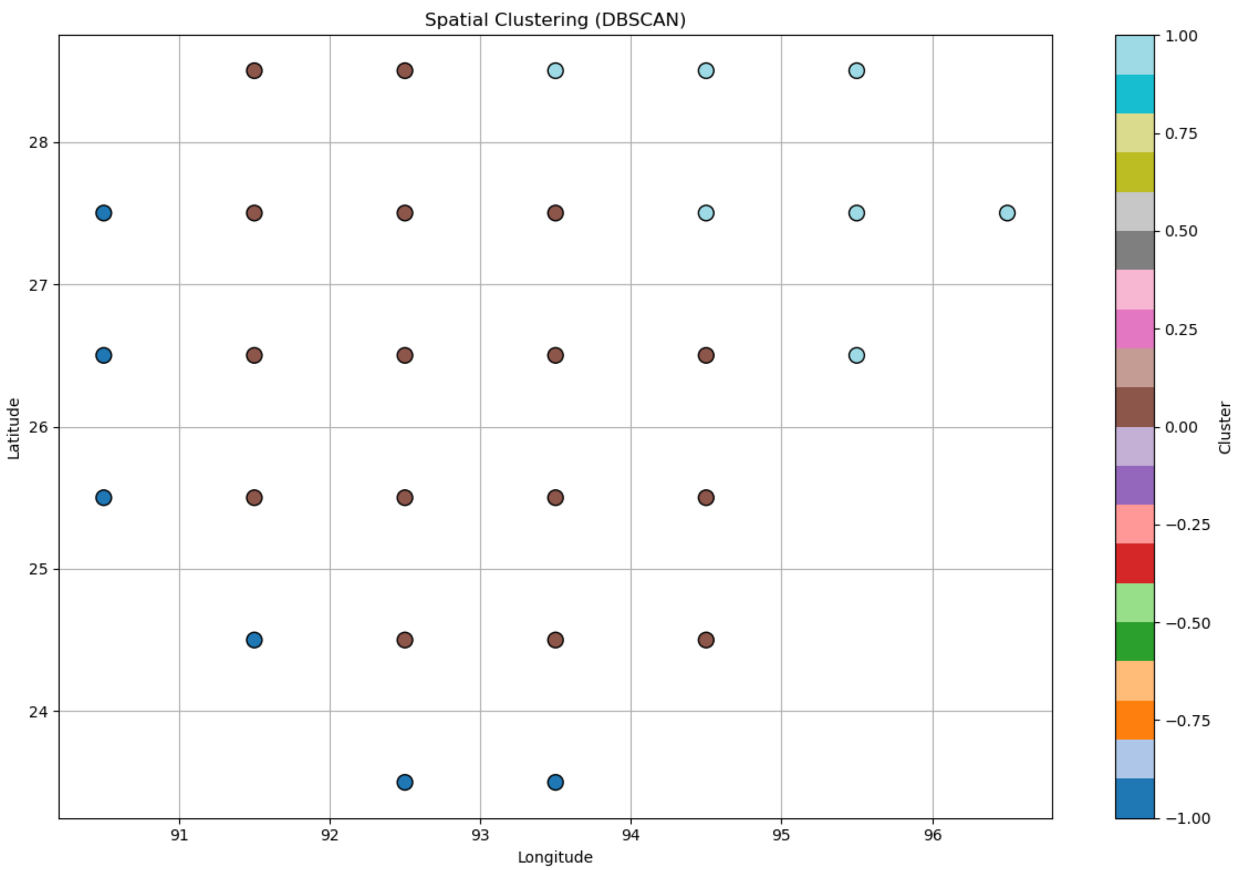
Spatial Clustering of Precipitation Patterns (DBSCAN)
Latitude
Longitude
Cluster
28
27
26
25
24
91
92
93
94
95
96
−0.900
−0.925
−0.950
−0.975
−1.000
−1.025
−1.050
−1.075
−1.100



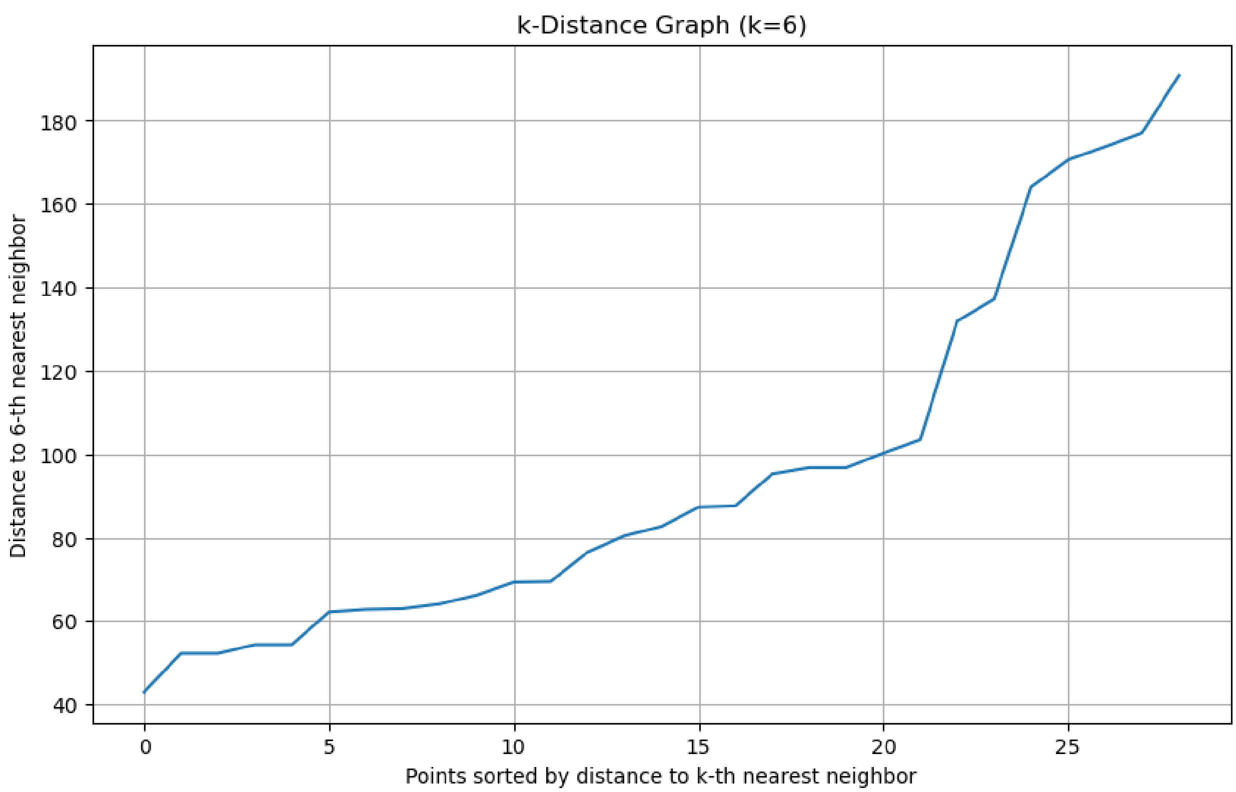
Updated

```
# Apply DBSCAN Clustering
dbscan = DBSCAN(eps=105.0, min_samples=7)
clustering_data["Cluster"] = dbscan.fit_predict(clustering_data_scaled)

# Print number of clusters and points in each cluster
cluster_counts = clustering_data["Cluster"].value_counts()
print(f"Number of clusters (excluding noise): {len(cluster_counts) - 1 if -1 in cluster_counts.index else len(cluster_counts)}")
print("Points per cluster:")
print(cluster_counts)
plt.figure(figsize=(12, 8))
sc = plt.scatter(
    clustering_data["Longitude"],
    clustering_data["Latitude"],
    c=clustering_data["Cluster"],
    cmap="tab20",
    s=100,
    edgecolor="black"
)
plt.colorbar(sc, label="Cluster")
plt.title("Spatial Clustering (DBSCAN)")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Number of clusters (excluding noise): 2
Points per cluster:
Cluster
 0    16
 1     7
-1     6
Name: count, dtype: int64
```

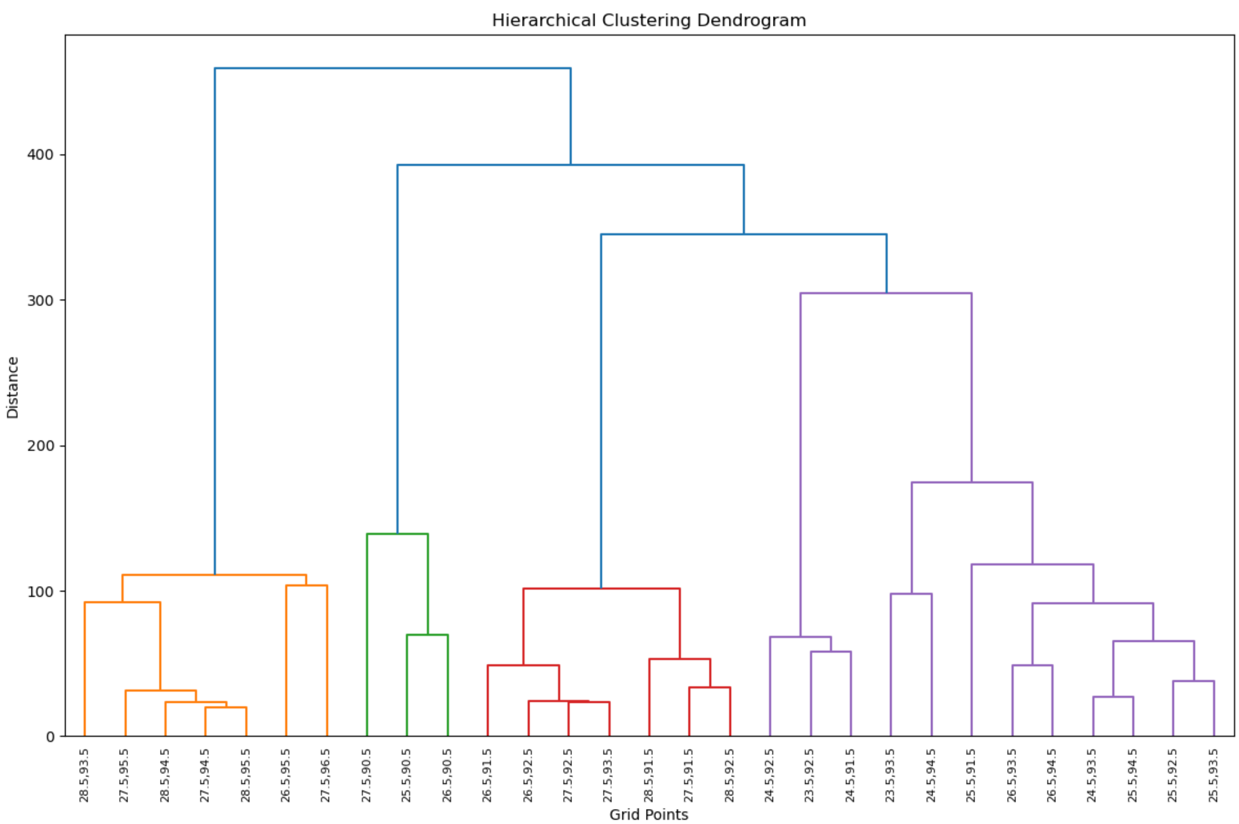


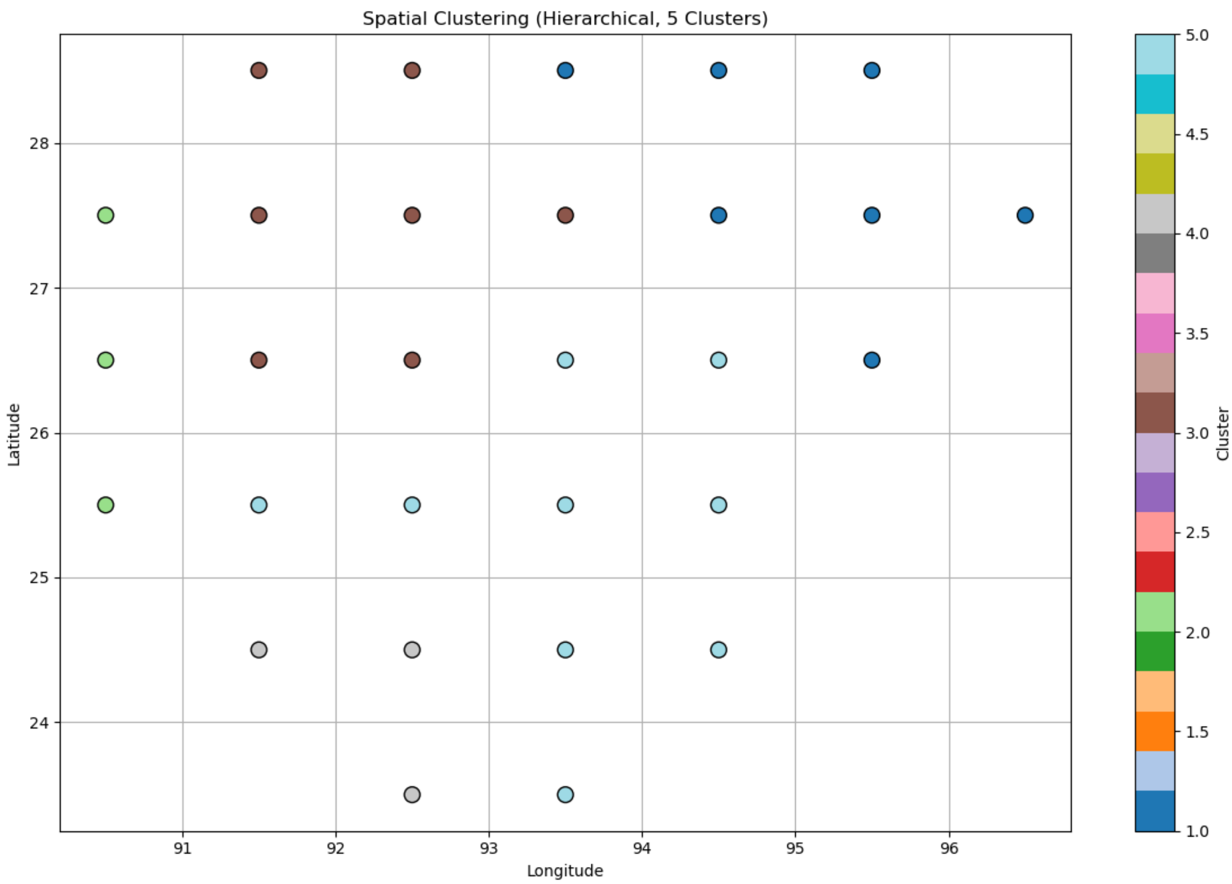
```python
# Perform Hierarchical Clustering
linkage_matrix = linkage(clustering_data_scaled, method='ward')  # Using Ward's method for linkage
plt.figure(figsize=(12, 8))
dendrogram(linkage_matrix, labels=clustering_data["grid_point"].values, leaf_rotation=90, leaf_font_size=8)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Grid Points")
plt.ylabel("Distance")
plt.tight_layout()
plt.show()
num_clusters = 5
clustering_data["Cluster"] = fcluster(linkage_matrix, num_clusters, criterion='maxclust')
plt.figure(figsize=(12, 8))
sc = plt.scatter(
    clustering_data["Longitude"],
    clustering_data["Latitude"],
    c=clustering_data["Cluster"],
    cmap="tab20",
    s=100,
    edgecolor="black"
)
plt.colorbar(sc, label="Cluster")
plt.title(f"Spatial Clustering (Hierarchical, {num_clusters} Clusters)")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()
```
[26]



Hierarchical data -

```
Number of clusters: 5
Points per cluster:
Cluster
5    9
3    7
1    7
4    3
2    3
Name: count, dtype: int64
```





Spatial Clustering (Hierarchical, 5 Clusters)
Latitude
Longitude
Cluster
24
25
26
27
28
91
92
93
94
95
96
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0