untitled39

April 16, 2025

[ ]:
```python
import pandas as pd
import numpy as np

# Load dataset
file_path = "/content/weatherHistory (2).csv"
data = pd.read_csv(file_path)

# Display column names for verification
print("Original Columns:", data.columns)

# Select relevant columns
relevant_columns = ['Formatted Date', 'Temperature (C)', 'Wind Speed (km/h)',␣
 ↪'Pressure (millibars)']
data = data[relevant_columns]
data.columns = ['Timestamp', 'Temperature', 'Wind Speed', 'Pressure']

# Convert Timestamp to datetime format
data['Timestamp'] = pd.to_datetime(data['Timestamp'])

# Drop missing values
data = data.dropna()

# Normalize data for better numerical stability
data['Temperature'] = (data['Temperature'] - data['Temperature'].mean()) /␣
 ↪data['Temperature'].std()
data['Wind Speed'] = (data['Wind Speed'] - data['Wind Speed'].mean()) /␣
 ↪data['Wind Speed'].std()
data['Pressure'] = (data['Pressure'] - data['Pressure'].mean()) /␣
 ↪data['Pressure'].std()

# Save the modified dataset
modified_file_path = "modified_weather_data.csv"
data.to_csv(modified_file_path, index=False)

print("Modified dataset saved at:", modified_file_path)
```

```
Original Columns: Index(['Formatted Date', 'Temperature (C)', 'Apparent
Temperature (C)',
```

```
       'Humidity', 'Wind Speed (km/h)', 'Wind Bearing (degrees)',
       'Pressure (millibars)'],
      dtype='object')
```

```
<ipython-input-1-058ca38fe5b5>:17: FutureWarning: In a future version of pandas,
parsing datetimes with mixed time zones will raise an error unless `utc=True`.
Please specify `utc=True` to opt in to the new behaviour and silence this
warning. To create a `Series` with mixed offsets and `object` dtype, please use
`apply` and `datetime.datetime.strptime`
  data['Timestamp'] = pd.to_datetime(data['Timestamp'])
<ipython-input-1-058ca38fe5b5>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['Temperature'] = (data['Temperature'] - data['Temperature'].mean()) /
data['Temperature'].std()
<ipython-input-1-058ca38fe5b5>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['Wind Speed'] = (data['Wind Speed'] - data['Wind Speed'].mean()) /
data['Wind Speed'].std()
<ipython-input-1-058ca38fe5b5>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['Pressure'] = (data['Pressure'] - data['Pressure'].mean()) /
data['Pressure'].std()
```

```
Modified dataset saved at: modified_weather_data.csv
```

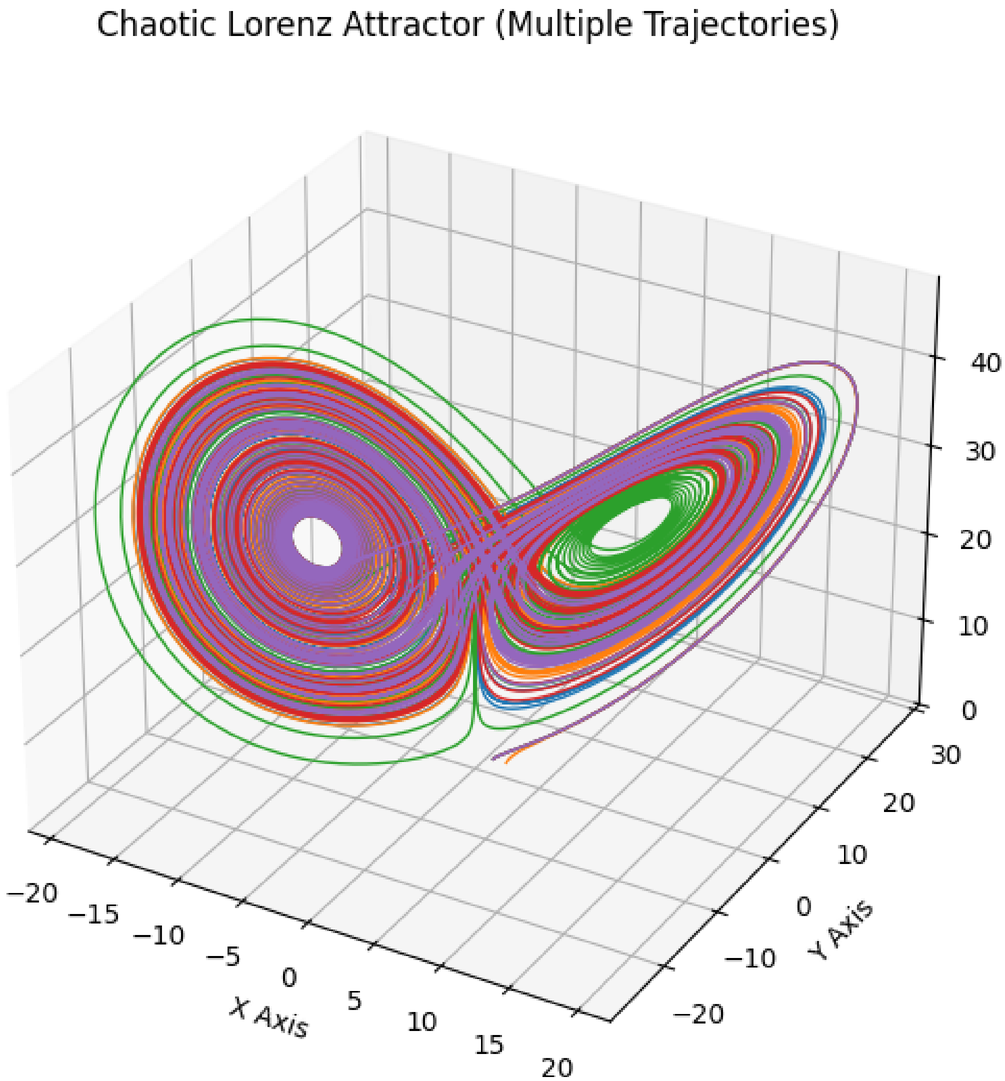
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the Lorenz system parameters
sigma = 10
beta = 8/3
rho = 28

# Lorenz system equations
def lorenz_system(state):
```

```
    x, y, z = state
    dx_dt = sigma * (y - x)
    dy_dt = x * (rho - z) - y
    dz_dt = x * y - beta * z
    return np.array([dx_dt, dy_dt, dz_dt])

# RK4 method to solve the system
def runge_kutta4(f, y0, t):
    n = len(t)
    y = np.zeros((n, len(y0)))
    y[0] = y0

    dt = t[1] - t[0]  # Time step

    for i in range(n - 1):
        k1 = f(y[i])
        k2 = f(y[i] + 0.5 * dt * k1)
        k3 = f(y[i] + 0.5 * dt * k2)
        k4 = f(y[i] + dt * k3)

        y[i + 1] = y[i] + (dt / 6) * (k1 + 2*k2 + 2*k3 + k4)

    return y

# Generate time points with a high-resolution step
time_points = np.linspace(0, 50, 10000)

# Generate multiple initial conditions for accuracy comparison
initial_conditions = [
    [1.0, 1.0, 1.0],
    [2.01, 1.0, 1.0],
    [0.99, 1.0, 1.0],
    [1.0, 1.01, 1.0],
    [1.0, 0.99, 1.0]
]

# Solve the Lorenz system for different initial conditions
solutions = [runge_kutta4(lorenz_system, ic, time_points) for ic in␣
 ↪initial_conditions]
```

[ ]:

```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(projection='3d')

for sol in solutions:
    ax.plot(sol[:, 0], sol[:, 1], sol[:, 2], linewidth=1)

ax.set_xlabel("X Axis")
```

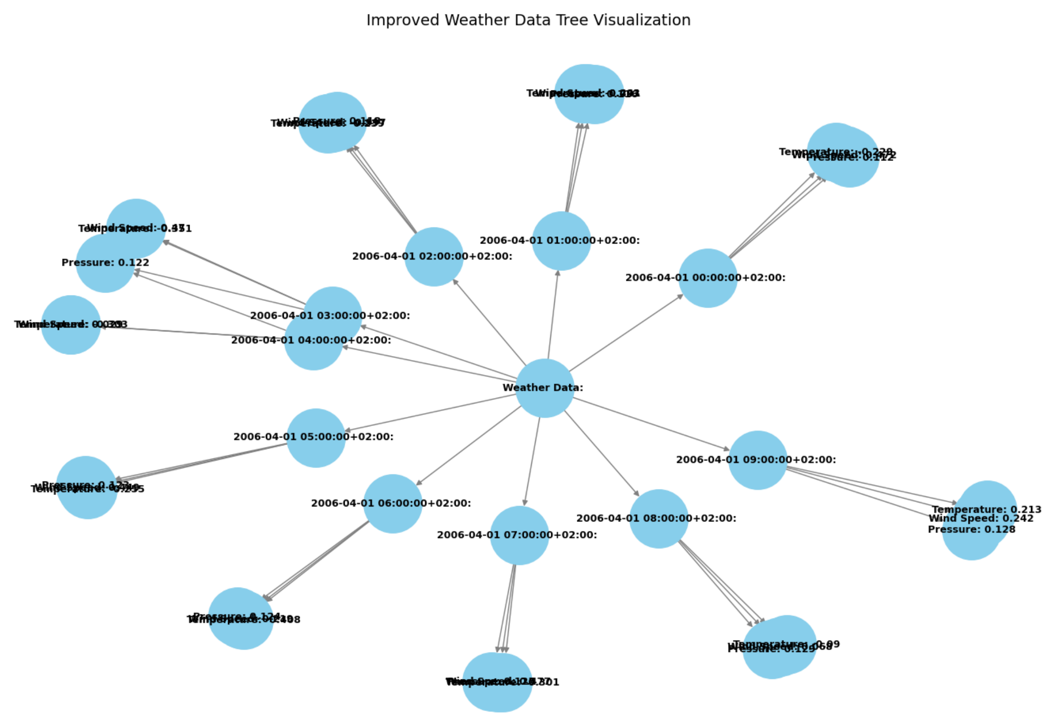
```
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
ax.set_title("Chaotic Lorenz Attractor (Multiple Trajectories)")

plt.show()
```




```
[ ]: import networkx as nx

# Define an N-ary Tree Node
class TreeNode:
    def __init__(self, name, value=None):
        self.name = name
        self.value = value
```

```
        self.children = []

    def add_child(self, child_node):
        self.children.append(child_node)

# Define N-ary Tree for Weather Data
class WeatherTree:
    def __init__(self):
        self.root = TreeNode("Weather Data")

    def insert_data(self, timestamp, temperature, wind_speed, pressure):
        timestamp_node = None

        # Check if timestamp node exists
        for child in self.root.children:
            if child.name == timestamp:
                timestamp_node = child
                break

        if timestamp_node is None:
            timestamp_node = TreeNode(timestamp)
            self.root.add_child(timestamp_node)


        temp_node = TreeNode("Temperature", temperature)
        wind_node = TreeNode("Wind Speed", wind_speed)
        press_node = TreeNode("Pressure", pressure)

        timestamp_node.add_child(temp_node)
        timestamp_node.add_child(wind_node)
        timestamp_node.add_child(press_node)

# Create tree and insert first 10 records
weather_tree = WeatherTree()

for index, row in data.head(10).iterrows():
    weather_tree.insert_data(str(row['Timestamp']), row['Temperature'],␣
 ↪row['Wind Speed'], row['Pressure'])

import networkx as nx
import matplotlib.pyplot as plt

def visualize_tree(root):
    G = nx.DiGraph()

    def add_edges(node, parent_name=None):
```

```python
        node_label = f"{node.name}: {round(node.value, 3) if node.value is not None else ''}"
        if parent_name:
            G.add_edge(parent_name, node_label)
        for child in node.children:
            add_edges(child, node_label)

    add_edges(root)

    plt.figure(figsize=(14, 9))

    # Improve layout spacing using Kamada-Kawai layout
    pos = nx.kamada_kawai_layout(G)

    # Draw nodes with different colors for clarity
    nx.draw(G, pos, with_labels=True, node_size=2800, node_color="skyblue", edge_color="gray", font_size=9, font_weight="bold")

    plt.title("Improved Weather Data Tree Visualization", fontsize=14)
    plt.show()

visualize_tree(weather_tree.root)
```



[ ]:

```
# Load the modified weather dataset
data = pd.read_csv("modified_weather_data.csv")

# Extract real values
real_temp = data['Temperature'].values[:10000]
real_wind_speed = data['Wind Speed'].values[:10000]
real_pressure = data['Pressure'].values[:10000]

# Extract Lorenz predictions (first solution set)
lorenz_x, lorenz_y, lorenz_z = solutions[0][:, 0], solutions[0][:, 1],␣
↪solutions[0][:, 2]

# Apply smoothing using a moving average
def smooth_data(data, window_size=50):
    return np.convolve(data, np.ones(window_size)/window_size, mode='same')

real_temp_smooth = smooth_data(real_temp)
real_wind_speed_smooth = smooth_data(real_wind_speed)
real_pressure_smooth = smooth_data(real_pressure)

# Compare real vs. predicted values
plt.figure(figsize=(12, 8))

# Temperature comparison
plt.subplot(3, 1, 1)
plt.plot(time_points, real_temp_smooth[:len(time_points)], label="Real␣
↪Temperature (Smoothed)", color='red')
plt.plot(time_points, lorenz_x, label="Predicted Temperature (RK4)",␣
↪color='blue', linestyle='dashed')
plt.legend()
plt.title("Comparison: Real vs. Predicted Temperature")

# Wind Speed comparison
plt.subplot(3, 1, 2)
plt.plot(time_points, real_wind_speed_smooth[:len(time_points)], label="Real␣
↪Wind Speed (Smoothed)", color='green')
plt.plot(time_points, lorenz_y, label="Predicted Wind Speed (RK4)",␣
↪color='blue', linestyle='dashed')
plt.legend()
plt.title("Comparison: Real vs. Predicted Wind Speed")

# Pressure comparison
plt.subplot(3, 1, 3)
plt.plot(time_points, real_pressure_smooth[:len(time_points)], label="Real␣
↪Pressure (Smoothed)", color='purple')
plt.plot(time_points, lorenz_z, label="Predicted Pressure (RK4)", color='blue',␣
↪linestyle='dashed')
```

```
plt.legend()
plt.title("Comparison: Real vs. Predicted Pressure")

plt.tight_layout()
plt.show()
```