

✓ CIND820 Capstone Project

Integrating Predictive Analytics and Anomaly Detection for Cyanobacterial Bloom Forecasting

- ✓ *This notebook details the development of predictive analytics for cyanobacterial blooms using ARIMA, LSTM, and anomaly detection models. It includes data preparation, exploratory analysis, and initial modeling insights.*

```
1 # Mount Google Drive to permit colab Notebook access to project datasets
2
3 from google.colab import drive
4 drive.mount('/content/drive')
```

🔗 Mounted at /content/drive

✓ 1. Data Collection and Preprocessing

✓ 1.1 Import Necessary Libraries

Import libraries that will be required for data analysis, visualization, and handling.

```
1 # Import necessary libraries for data manipulation and visualization.
2
3 import pandas as pd
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7
```

✓ 1.2 Source Identification and Data Acquisition

Define the file paths to the datasets that will be used in the analysis and in assessing the project research questions.

```
1 # Specify file paths for the datasets
2
3 bloom_indices_path = '/content/drive/MyDrive/CIND820/LakeErie_Daily_BloomIndices.csv'
4 water_quality_path = '/content/drive/MyDrive/CIND820/lake_erie_habs_field_sampling_results_2012_2018.csv'
5
```

✓ 1.3 Load Datasets

Read the Bloom Indices and Water Quality datasets into pandas DataFrames for manipulation.

```
1 # Load the Bloom Indices dataset
2 bloom_data = pd.read_csv(bloom_indices_path, encoding='ISO-8859-1')
3
4 # Load the Water Quality dataset
5 water_quality_data = pd.read_csv(water_quality_path, encoding='ISO-8859-1')
6
```

✓ 1.4 Data Inspection

Get a preliminary understanding of the data structure and content.

```
1 # Inspect the first few rows of the Bloom Indices dataset
2 print("Bloom Data Sample:")
3 print(bloom_data.head())
4
5 # Print a line of underscores to demarcate the outputs
6 print("_ " * 70)
7
8 # Inspect the first few rows of the Water Quality dataset
9 print("\nWater Quality Data Sample:")
10 print(water_quality_data.head())
11
```



	Particulate Organic Carbon (mg/L)	Particulate Organic Nitrogen (mg/L)	\
0	NaN	NaN	
1	0.34	0.05	
2	0.40	0.08	
3	0.49	0.12	
4	0.50	0.07	

	Dissolved Organic Carbon (mg/L)	\
0	2.88	
1	1.80	
2	4.35	
3	2.53	
4	2.80	

	Colored Dissolved Organic Material absorbance (m-1) at 400nm	\
0	NaN	
1	NaN	
2	NaN	
3	NaN	
4	NaN	

	Total Suspended Solids (mg/L)	Volatile Suspended Solids (mg/L)
0	3.47	1.11
1	3.15	1.01
2	2.90	0.98
3	4.32	1.06
4	21.42	2.40

[5 rows x 36 columns]

✓ 1.5 Data Cleaning and Preparation

✓ 1.5.1 Parsing Dates

Convert date columns to datetime objects for time-series analysis.

2. Convert bloom_data Dates to Match water_quality_data Use the pd.to_datetime function to parse bloom_data dates into yyyy-mm-dd:

```

1 # Inspect the current date formats and data types
2 print("Initial Date Format and Data Type:")
3 print(bloom_data['Date'].head())
4 print(water_quality_data['Date'].head())
5
6 print("\nData Types:")
7 print(f"Bloom Data Date Type: {bloom_data['Date'].dtype}")
8 print(f"Water Quality Data Date Type: {water_quality_data['Date'].dtype}")
9
10 # Convert Date columns to datetime
11 bloom_data['Date'] = pd.to_datetime(bloom_data['Date'], format='%Y-%m-%d', errors='coerce')
12 water_quality_data['Date'] = pd.to_datetime(water_quality_data['Date'], format='%m/%d/%Y', errors='coerce')
13
14 # Check for invalid dates (after conversion)
15 print("\nPost Conversion - Check for Null Dates:")
16 print(bloom_data['Date'].isnull().sum())
17 print(water_quality_data['Date'].isnull().sum())

```

```

18
19 # Verify the new data types
20 print("\nPost Conversion - Data Types:")
21 print(f"Bloom Data Date Type: {bloom_data['Date'].dtype}")
22 print(f"Water Quality Data Date Type: {water_quality_data['Date'].dtype}")
23
24 # Preview the converted dates
25 print("\nConverted Date Formats:")
26 print(bloom_data['Date'].head())
27 print(water_quality_data['Date'].head())
28

```



Initial Date Format and Data Type:

```

0  2002-06-02
1  2002-06-15
2  2002-06-18
3  2002-06-23
4  2002-06-24
Name: Date, dtype: object
0  5/15/2012
1  5/15/2012
2  5/15/2012
3  5/15/2012
4  5/31/2012
Name: Date, dtype: object

```

Data Types:

```

Bloom Data Date Type: object
Water Quality Data Date Type: object

```

Post Conversion - Check for Null Dates:

```

0
0

```

Post Conversion - Data Types:

```

Bloom Data Date Type: datetime64[ns]
Water Quality Data Date Type: datetime64[ns]

```

Converted Date Formats:

```

0  2002-06-02
1  2002-06-15
2  2002-06-18
3  2002-06-23
4  2002-06-24
Name: Date, dtype: datetime64[ns]
0  2012-05-15
1  2012-05-15
2  2012-05-15
3  2012-05-15
4  2012-05-31
Name: Date, dtype: datetime64[ns]

```

1.5.2 Handling Column Names

Ensure consistency in column names for easier data manipulation. Standardize Column Names Normalize column names by stripping whitespace and other hidden characters.

2. Standardize Column Names Normalize column names by stripping whitespace and other hidden characters:

```
1 bloom_data.columns = bloom_data.columns.str.strip().str.replace('\n', '').str.replace('\r', '').str.replace(' ', ' ')
2 water_quality_data.columns = water_quality_data.columns.str.strip().str.replace('\n', '').str.replace('\r', '').str.replace(' ', ' ')
3
4 print("Normalized column names in bloom_data:", bloom_data.columns.tolist())
5 print("Normalized column names in water_quality_data:", water_quality_data.columns.tolist())
6
```

Normalized column names in bloom_data: ['Date', 'Lake', 'Satellite Sensor', 'Bloom Extent (KM2)', 'Bloom Extent (% of Lake Area)', 'Bloom Intensity (µg/L)', 'Bloom Severity (µg/L km2)']
 Normalized column names in water_quality_data: ['Date', 'Site', 'Station Depth (m)', 'Sample Depth (m)', 'Sample Depth (category)', 'Local Time (Eastern Time Zone)', 'Latitude (decimal deg)', 'Longitude (decimal deg)', 'Wind speed (knots)', 'Wave Height (ft)', 'Sky', 'Secchi Depth (m)', 'Sample Temperature (°C)', 'CTD Temperature (°C)']

1.5.3 Handling Missing Values

Address missing data to prevent errors in analysis.

```
1 # Check for missing values in Bloom Data
2 print("\nMissing values in Bloom Data:")
3 print(bloom_data.isnull().sum())
4
```

Missing values in Bloom Data:

Date	0
Lake	0
Satellite Sensor	0
Bloom Extent (KM2)	0
Bloom Extent (% of Lake Area)	0
Bloom Intensity (µg/L)	0
Bloom Severity (µg/L km2)	0
Valid Pixels (% of Lake Area)	0
dtype:	int64

```
1 # Check for missing values in Water Quality Data
2 print("\nMissing values in Water Quality Data:")
3 print(water_quality_data.isnull().sum())
4
```

Missing values in Water Quality Data:

Date	0
Site	0
Station Depth (m)	371
Sample Depth (m)	17
Sample Depth (category)	0
Local Time (Eastern Time Zone)	363
Latitude (decimal deg)	376
Longitude (decimal deg)	376
Wind speed (knots)	650
Wave Height (ft)	655
Sky	366
Secchi Depth (m)	405
Sample Temperature (°C)	1188
CTD Temperature (°C)	136

```

CTD Specific Conductivity (µS/cm)      133
CTD Beam Attenuation (m-1)             151
CTD Transmission (%)                   151
CTD Dissolved Oxygen (mg/L)            132
CTD Photosynthetically Active Radiation (µE/m2/s) 134
Turbidity (NTU)                        424
Particulate Microcystin (µg/L)          19
Dissolved Microcystin (µg/L)           251
Extracted Phycocyanin (µg/L)           4
Chlorophyll_a                          5
Total Phosphorus (µg P/L)              371
Total Dissolved Phosphorus (µg P/L)    369
Soluble Reactive Phosphorus (µg P/L)   367
Ammonia (µg N/L)                      366
Nitrate + Nitrite (mg N/L)             367
Urea (µg N/L)                         925
Particulate Organic Carbon (mg/L)      371
Particulate Organic Nitrogen (mg/L)    371
Dissolved Organic Carbon (mg/L)        524
Colored Dissolved Organic Material absorbance (m-1) at 400nm 568
Total Suspended Solids (mg/L)          383
Volatile Suspended Solids (mg/L)       381
dtype: int64

```

```

1 # Handle missing values in Water Quality Data
2 # For numeric columns, we can fill missing values with mean or median if appropriate
3 numeric_columns = water_quality_data.select_dtypes(include=[np.number]).columns.tolist()
4 water_quality_data[numeric_columns] = water_quality_data[numeric_columns].fillna(method='ffill')
5

```

✓ 1.6 Data Integration

✓ 1.6.1 Aligning Datasets Temporally

Combine datasets based on the date to facilitate joint analysis.

Final Check Rerun the merge operation after ensuring the Date columns are correctly formatted and aligned:

```

1 # Merge datasets on 'Date'
2 merged_data = pd.merge(bloom_data, water_quality_data, on='Date', how='inner')
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```

→      Date      Lake Satellite Sensor  Bloom Extent (KM2) \
0 2016-06-13  Lake Erie      OLCI-S3      136.08
1 2016-06-13  Lake Erie      OLCI-S3      136.08
2 2016-06-13  Lake Erie      OLCI-S3      136.08
3 2016-06-13  Lake Erie      OLCI-S3      136.08
4 2016-06-13  Lake Erie      OLCI-S3      136.08

Bloom Extent (% of Lake Area)  Bloom Intensity (µg/L) \

```

0	0.51	49.26
1	0.51	49.26
2	0.51	49.26
3	0.51	49.26
4	0.51	49.26

	Bloom Severity ($\mu\text{g/L km}^2$)	Valid Pixels (% of Lake Area)	Site	\
0	6703.63		99.94	WE2
1	6703.63		99.94	WE2
2	6703.63		99.94	WE4
3	6703.63		99.94	WE6
4	6703.63		99.94	WE8

	Station Depth (m)	...	Soluble Reactive Phosphorus ($\mu\text{g P/L}$)	\
0	5.50	...		1.4
1	NaN	...		NaN
2	8.71	...		0.9
3	3.10	...		2.7
4	4.67	...		3.4

	Ammonia ($\mu\text{g N/L}$)	Nitrate + Nitrite (mg N/L)	Urea ($\mu\text{g N/L}$)	\
0	17.9	0.47	8.96	
1	NaN	NaN	NaN	
2	56.1	0.34	10.85	
3	7.4	1.73	8.37	
4	38.9	1.02	16.21	

	Particulate Organic Carbon (mg/L)	Particulate Organic Nitrogen (mg/L)	\
0	0.40	0.05	
1	NaN	NaN	
2	0.88	0.16	
3	2.37	0.37	
4	0.44	0.07	

	Dissolved Organic Carbon (mg/L)	\
0	2.30	
1	NaN	
2	1.94	
3	4.53	
4	2.83	

	Colored Dissolved Organic Material absorbance (m-1) at 400nm	\
0	0.37	
1	NaN	
2	0.22	
3	1.68	
4	0.68	

	Total Suspended Solids (mg/L)	Volatile Suspended Solids (mg/L)
0	2.02	0.20

```

1 # Having already created the `merged_data`...
2
3 # Convert the merged data to a DataFrame
4 merged_df = pd.DataFrame(merged_data)
5
6 # Display the entire DataFrame in a scrollable table using Pandas
7 from IPython.display import display
8
9 # Display the DataFrame
10 display(merged_df)
11

```



	Date	Lake	Satellite Sensor	Bloom Extent (KM2)	Bloom Extent (% of Lake Area)	Bloom Intensity (µg/L)	Bloom Severity (µg/L km2)	Valid Pixels (% of Lake Area)	Site	Station Depth (m)	...	Soluble Reactive Phosphorus (µg P/L)	Ammonia (µg N/L)	Nitrate + Nitrite (mg N/L)	Urea (µg N/L)	Particulate Organic Carbon (mg/L)	Particulate Organic Nitrogen (mg/L)	Dissolved Organic Carbon (mg/L)	Colored Dissolved Organic Material absorbance (m-1) at 400nm	T Suspe So (m
0	2016-06-13	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE2	5.50	...	1.4	17.9	0.47	8.96	0.40	0.05	2.30	0.370	
1	2016-06-13	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE2	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	2016-06-13	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE4	8.71	...	0.9	56.1	0.34	10.85	0.88	0.16	1.94	0.220	
3	2016-06-13	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE6	3.10	...	2.7	7.4	1.73	8.37	2.37	0.37	4.53	1.680	
4	2016-06-13	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE8	4.67	...	3.4	38.9	1.02	16.21	0.44	0.07	2.83	0.680	
...	
610	2018-10-01	Lake Erie	OLCI-S3	164.07	0.61	25.82	4236.66	99.91	WE13	9.07	...	6.17	1.72	0.27	NaN	0.50	0.10	NaN	0.180	
611	2018-10-01	Lake Erie	OLCI-S3	164.07	0.61	25.82	4236.66	99.91	WE13	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
612	2018-10-01	Lake Erie	OLCI-S3	164.07	0.61	25.82	4236.66	99.91	WE16	6.30	...	18.33	22.69	0.11	NaN	0.72	0.12	NaN	0.771	
613	2018-10-09	Lake Erie	OLCI-S3	157.86	0.59	24.85	3923.31	99.90	WE6	NaN	...	44.14	78.41	0.63	NaN	1.19	0.21	NaN	2.278	
614	2018-10-09	Lake Erie	OLCI-S3	157.86	0.59	24.85	3923.31	99.90	WE9	NaN	...	59.19	145.34	0.98	NaN	1.45	0.27	NaN	0.925	

615 rows × 43 columns



```

1 from google.colab import data_table
2 data_table.DataTable(merged_df)
3
4

```


Warning: Total number of columns (43) exceeds max_columns (20) limiting to first (20) columns.

1 to 25 of 615 entries

index	Date	Lake	Satellite Sensor	Bloom Extent (KM2)	Bloom Extent (% of Lake Area)	Bloom Intensity (µg/L)	Bloom Severity (µg/L km2)	Valid Pixels (% of Lake Area)	Site	Station Depth (m)	Sample Depth (m)
0	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE2	5.5	0.75
1	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE2	NaN	4.6
2	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE4	8.71	0.75
3	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE6	3.1	0.75
4	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE8	4.67	0.75
5	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE8	NaN	3.9
6	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE9	2.66	0.75
7	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE12	6.76	0.75
8	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE12	NaN	5.9
9	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE13	9.15	0.75
10	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE13	NaN	8.2
11	2016-06-13 00:00:00	Lake Erie	OLCI-S3	136.08	0.51	49.26	6703.63	99.94	WE15	4.67	0.75
12	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE2	5.75	0.75
13	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE2	NaN	4.6
14	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE4	8.76	0.75
15	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE6	2.87	0.75
16	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE8	4.37	0.75
17	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE8	NaN	3.6
18	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE9	2.42	0.75

19	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE12	6.73	0.75
20	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE12	NaN	5.8
21	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE13	9.01	0.75
22	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE13	NaN	8.1
23	2016-06-27 00:00:00	Lake Erie	OLCI-S3	154.53	0.58	48.97	7567.06	99.93	WE15	4.55	0.75
24	2016-07-05 00:00:00	Lake Erie	OLCI-S3	149.49	0.56	51.05	7631.85	99.94	WE2	5.42	0.75

◀

▶

Show 25 per page

12102025



```
1 # Define the file path for saving the merged dataframe
2 output_file_path = "/content/drive/MyDrive/CIND820/LakeErie_Merged_Data.csv"
3
4 # Save the merged dataframe to the specified location as a CSV file
5 merged_df.to_csv(output_file_path, index=False)
6
7 # Confirm the file has been saved
8 print(f"Merged DataFrame saved to: {output_file_path}")
9
```

Merged DataFrame saved to: /content/drive/MyDrive/CIND820/LakeErie_Merged_Data.csv

Summary of Section 1

- Handled missing values and ensured consistency across datasets.
- Ensured compatibility for downstream alignment and analysis steps.

2. Exploratory Data Analysis (EDA)

2.1 Statistical Summary and Visualization

2.1.1 Descriptive Statistics

Provide a statistical overview of the datasets.

```
1 # Summary statistics for Bloom Data
2 print("\nBloom Data Statistics:")
```

```
3 print(bloom_data.describe())
4
5 # Summary statistics for Water Quality Data
6 print("\nWater Quality Data Statistics:")
7 print(water_quality_data.describe())
8
```



max	540.800000	515.000000
std	31.277766	22.803195

```

1 # Convert the bloom_data summary statistics output to a DataFrame
2 bloom_data_summary_statistics_df = pd.DataFrame(bloom_data.describe())
3
4 # Display the DataFrame using Pandas
5 from IPython.display import display
6
7 # Display the DataFrame
8 display(bloom_data_summary_statistics_df)
9
10 # Define the file path for saving the merged dataframe
11 output_file_path = "/content/drive/MyDrive/CIND820/bloom_data_summary_statistics_Output.csv"
12
13 # Save the merged dataframe to the specified location as a CSV file
14 bloom_data_summary_statistics_df.to_csv(output_file_path, index=False)

```



	Bloom Extent (KM2)	Bloom Extent (% of Lake Area)	Bloom Intensity (ug/L)	Bloom Severity (ug/L km2)	Valid Pixels (% of Lake Area)
count	1369.000000	1369.000000	1369.000000	1369.000000	1369.000000
mean	443.157239	1.660767	33.203046	13417.767480	96.355566
std	594.930353	2.229544	10.133595	17402.623445	12.535047
min	0.270000	0.000000	11.190000	3.020000	1.010000
25%	174.960000	0.660000	25.380000	7021.980000	99.290000
50%	256.320000	0.960000	31.390000	8969.150000	99.890000
75%	469.980000	1.760000	40.320000	11618.340000	99.980000
max	5256.810000	19.700000	65.620000	176410.750000	100.000000

```

1 # Convert the water_quality_data summary statistics output to a DataFrame
2 water_quality_data_summary_statistics_df = pd.DataFrame(water_quality_data.describe())
3
4 # Display the DataFrame using Pandas
5 from IPython.display import display
6
7 # Display the DataFrame
8 display(water_quality_data_summary_statistics_df)
9
10 # Define the file path for saving the merged dataframe
11 output_file_path = "/content/drive/MyDrive/CIND820/water_quality_data_summary_statistics_Output.csv"
12
13 # Save the merged dataframe to the specified location as a CSV file
14 water_quality_data_summary_statistics_df.to_csv(output_file_path, index=False)

```



	Date	Station Depth (m)	Sample Depth (m)	Latitude (decimal deg)	Longitude (decimal deg)	Wave Height (ft)	Sample Temperature (°C)	CTD Temperature (°C)	CTD Specific Conductivity (µS/cm)	CTD Photosynthetically Active Radiation (µE/m2/s)	Turbidity (NTU)	Chlorophyll_a	Total Phosphorus (µg P/L)
count	1244	873.000000	1227.000000	868.000000	868.000000	589.000000	56.000000	1108.000000	1111.000000	1110.000000	820.000000	1239.000000	873.000000
mean	2016-04-14 17:03:16.784566016	5.643860	1.930929	41.748882	-83.272040	0.879677	23.823214	21.970036	289.774887	324.566315	19.161780	216.100347	75.086072
min	2012-05-15 00:00:00	1.900000	0.000000	41.012700	-83.708400	0.020000	16.300000	2.400000	1.500000	0.000000	0.680000	0.710000	4.000000
25%	2015-06-22 00:00:00	4.000000	0.750000	41.705875	-83.363600	0.480000	21.700000	20.300000	249.300000	14.000000	4.627500	6.265000	26.400000
50%	2016-07-18 00:00:00	5.330000	0.750000	41.743000	-83.328350	0.750000	24.200000	22.900000	274.000000	147.370000	9.710000	16.180000	48.860000
75%	2017-08-21 00:00:00	7.800000	3.300000	41.826300	-83.193375	1.170000	26.050000	24.500000	314.800000	460.372500	19.300000	37.390000	82.130000
max	2018-10-09 00:00:00	10.860000	8.700000	42.002000	-82.698300	2.970000	28.300000	29.700000	855.900000	1902.000000	1148.000000	161971.200000	2482.240000
std	NaN	2.202444	2.187762	0.080505	0.118646	0.539996	2.671261	3.635855	64.250139	418.007239	54.735476	4662.518424	129.324432

2. Exploratory Data Analysis (EDA)

Inspect Column Names to verify the exact column names. Standardize Column Names to remove unnecessary spaces and special characters like "µ" which may not render correctly and to simplify access for downstream analyses.

```
1 print(bloom_data.columns.tolist())
2
```

```
['Date', 'Lake', 'Satellite Sensor', 'Bloom Extent (KM2)', 'Bloom Extent (% of Lake Area)', 'Bloom Intensity (µg/L)', 'Bloom Severity (µg/L km2)', 'Valid Pixels (% of Lake Area)']
```

```
1 bloom_data.columns = bloom_data.columns.str.strip().str.replace('µ', 'u').str.replace('µ', 'u')
2 print(bloom_data.columns.tolist())
3
```

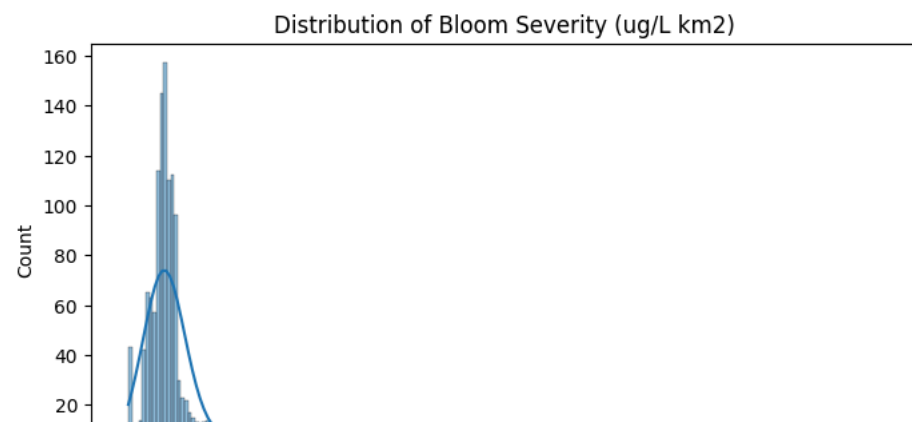
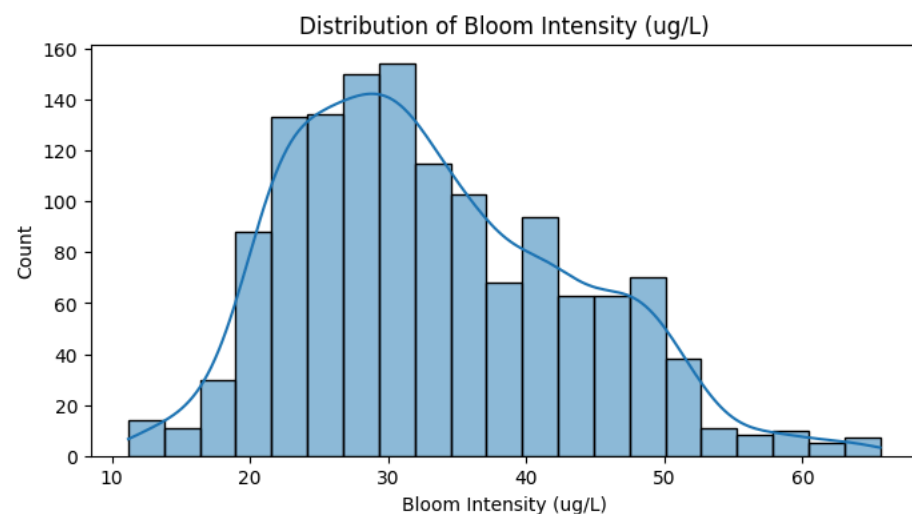
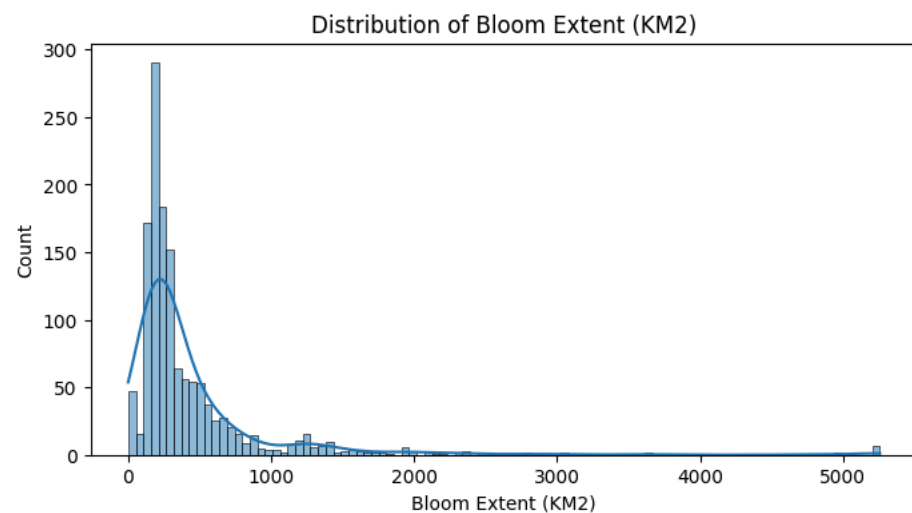
```
['Date', 'Lake', 'Satellite Sensor', 'Bloom Extent (KM2)', 'Bloom Extent (% of Lake Area)', 'Bloom Intensity (ug/L)', 'Bloom Severity (ug/L km2)', 'Valid Pixels (% of Lake Area)']
```

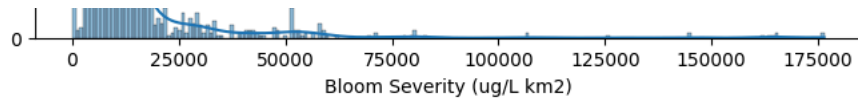
2.1.2 Visualization of Distributions

Visualize the distribution of key bloom indices to understand data variability.

```
1 # Plot histograms of key Bloom Indices
2 bloom_features = ['Bloom Extent (KM2)', 'Bloom Intensity (ug/L)', 'Bloom Severity (ug/L km2)']
3
```

```
4 for feature in bloom_features:
5     plt.figure(figsize=(8, 4))
6     sns.histplot(bloom_data[feature], kde=True)
7     plt.title(f'Distribution of {feature}')
8     plt.show()
9
```

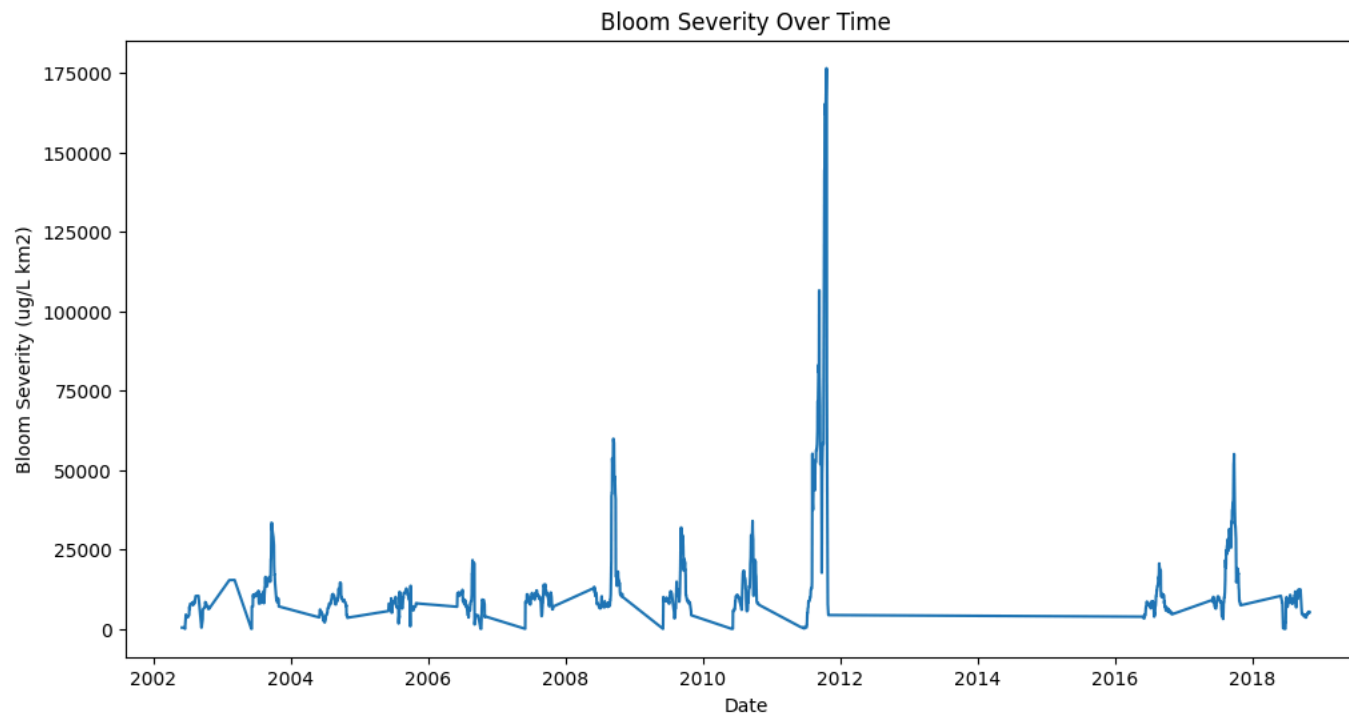




2.1.3 Time-Series Plots

Identify trends and patterns over time in bloom severity.

```
1 # Time-series plot of Bloom Severity
2 plt.figure(figsize=(12, 6))
3 sns.lineplot(x='Date', y='Bloom Severity (ug/L km2)', data=bloom_data)
4 plt.title('Bloom Severity Over Time')
5 plt.xlabel('Date')
6 plt.ylabel('Bloom Severity (ug/L km2)')
7 plt.show()
8
```



2.2 Correlation Analysis

Identify relationships between environmental variables and bloom indices.

```
1 # Filter for numeric columns in the water quality dataset
2 numeric_columns = water_quality_data.select_dtypes(include=['float64', 'int64']).columns
```



```
3
4 # Aggregate water quality data by date (compute daily mean values for numeric columns)
5 water_quality_daily = water_quality_data.groupby('Date')[numeric_columns].mean().reset_index()
6
7 # Merge aggregated water quality data with bloom data on 'Date'
8 merged_daily_data = pd.merge(bloom_data, water_quality_daily, on='Date', how='inner')
9
10 # Compute correlation matrix for selected attributes
11 selected_columns = ['Bloom Extent (KM2)', 'Bloom Intensity (ug/L)', 'CTD Temperature (°C)', 'Chlorophyll_a',
12 .....: 'Total Phosphorus (µg P/L)', 'Total Dissolved Phosphorus (µg P/L)', 'Particulate Organic Nitrogen (mg/L)',
13 .....: 'Particulate Organic Carbon (mg/L)', 'Dissolved Organic Carbon (mg/L)',
14 .....: 'Colored Dissolved Organic Material absorbance (m-1) at 400nm',
15 .....: 'Total Suspended Solids (mg/L)', 'Volatile Suspended Solids (mg/L)']
16 correlation_matrix = merged_daily_data[selected_columns].corr()
17
18 # Display the correlation matrix
19 print("Correlation Matrix:")
20 print(correlation_matrix)
21
22 # Plot the correlation heatmap
23 plt.figure(figsize=(10, 6))
24 sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
25 plt.title('Correlation Matrix of Bloom Indices and Water Quality Parameters')
26 plt.show()
27
28
29
```



Correlation Matrix:

	Bloom Extent (KM2) \
Bloom Extent (KM2)	1.000000
Bloom Intensity (ug/L)	-0.289382
CTD Temperature (°C)	-0.163121
Chlorophyll_a	0.549689
Total Phosphorus (µg P/L)	-0.282915
Total Dissolved Phosphorus (µg P/L)	-0.426608
Particulate Organic Nitrogen (mg/L)	0.652454
Particulate Organic Carbon (mg/L)	0.689555
Dissolved Organic Carbon (mg/L)	-0.513720
Colored Dissolved Organic Material absorbance (...)	-0.342159
Total Suspended Solids (mg/L)	0.190316
Volatile Suspended Solids (mg/L)	0.658319

	Bloom Intensity (ug/L) \
Bloom Extent (KM2)	-0.289382
Bloom Intensity (ug/L)	1.000000
CTD Temperature (°C)	0.147104
Chlorophyll_a	-0.101966
Total Phosphorus (µg P/L)	0.105888
Total Dissolved Phosphorus (µg P/L)	0.037016
Particulate Organic Nitrogen (mg/L)	-0.328346
Particulate Organic Carbon (mg/L)	-0.344169
Dissolved Organic Carbon (mg/L)	0.267157
Colored Dissolved Organic Material absorbance (...)	0.275269
Total Suspended Solids (mg/L)	-0.345269
Volatile Suspended Solids (mg/L)	-0.465574

	CTD Temperature (°C) \
Bloom Extent (KM2)	-0.163121
Bloom Intensity (ug/L)	0.147104
CTD Temperature (°C)	1.000000
Chlorophyll_a	-0.133796
Total Phosphorus (µg P/L)	0.083726
Total Dissolved Phosphorus (µg P/L)	-0.319994
Particulate Organic Nitrogen (mg/L)	0.109280
Particulate Organic Carbon (mg/L)	0.121754
Dissolved Organic Carbon (mg/L)	0.479830
Colored Dissolved Organic Material absorbance (...)	0.248849
Total Suspended Solids (mg/L)	-0.332781
Volatile Suspended Solids (mg/L)	0.027929

	Chlorophyll_a \
Bloom Extent (KM2)	0.549689
Bloom Intensity (ug/L)	-0.101966
CTD Temperature (°C)	-0.133796
Chlorophyll_a	1.000000
Total Phosphorus (µg P/L)	-0.128680
Total Dissolved Phosphorus (µg P/L)	-0.206206
Particulate Organic Nitrogen (mg/L)	0.256347
Particulate Organic Carbon (mg/L)	0.265493
Dissolved Organic Carbon (mg/L)	-0.319169
Colored Dissolved Organic Material absorbance (...)	-0.166047
Total Suspended Solids (mg/L)	0.020979
Volatile Suspended Solids (mg/L)	0.262212

	Total Phosphorus (µg P/L) \
Bloom Extent (KM2)	-0.282915
Bloom Intensity (ug/L)	0.105888
CTD Temperature (°C)	0.083726
Chlorophyll_a	-0.128680
Total Phosphorus (µg P/L)	1.000000
Total Dissolved Phosphorus (µg P/L)	0.552252

Total Dissolved Phosphorus (µg P/L)	0.553253
Particulate Organic Nitrogen (mg/L)	-0.046774
Particulate Organic Carbon (mg/L)	-0.073528
Dissolved Organic Carbon (mg/L)	0.371485
Colored Dissolved Organic Material absorbance (...)	0.392093
Total Suspended Solids (mg/L)	0.462357
Volatile Suspended Solids (mg/L)	0.018401
Total Dissolved Phosphorus (µg P/L) \	
Bloom Extent (KM2)	-0.426608
Bloom Intensity (ug/L)	0.037016
CTD Temperature (°C)	-0.319994
Chlorophyll_a	-0.206206
Total Phosphorus (µg P/L)	0.553253
Total Dissolved Phosphorus (µg P/L)	1.000000
Particulate Organic Nitrogen (mg/L)	-0.359769
Particulate Organic Carbon (mg/L)	-0.399741
Dissolved Organic Carbon (mg/L)	0.376437
Colored Dissolved Organic Material absorbance (...)	0.548878
Total Suspended Solids (mg/L)	0.318059
Volatile Suspended Solids (mg/L)	-0.247314
Particulate Organic Nitrogen (mg/L) \	
Bloom Extent (KM2)	0.652454
Bloom Intensity (ug/L)	-0.328346
CTD Temperature (°C)	0.109280
Chlorophyll_a	0.256347
Total Phosphorus (µg P/L)	-0.046774
Total Dissolved Phosphorus (µg P/L)	-0.359769
Particulate Organic Nitrogen (mg/L)	1.000000
Particulate Organic Carbon (mg/L)	0.988545
Dissolved Organic Carbon (mg/L)	-0.099448
Colored Dissolved Organic Material absorbance (...)	-0.169900
Total Suspended Solids (mg/L)	0.389950
Volatile Suspended Solids (mg/L)	0.886939
Particulate Organic Carbon (mg/L) \	
Bloom Extent (KM2)	0.689555
Bloom Intensity (ug/L)	-0.344169
CTD Temperature (°C)	0.121754
Chlorophyll_a	0.265493
Total Phosphorus (µg P/L)	-0.073528
Total Dissolved Phosphorus (µg P/L)	-0.399741
Particulate Organic Nitrogen (mg/L)	0.988545
Particulate Organic Carbon (mg/L)	1.000000
Dissolved Organic Carbon (mg/L)	-0.131561
Colored Dissolved Organic Material absorbance (...)	-0.201603
Total Suspended Solids (mg/L)	0.359638
Volatile Suspended Solids (mg/L)	0.890582
Dissolved Organic Carbon (mg/L) \	
Bloom Extent (KM2)	-0.513720
Bloom Intensity (ug/L)	0.267157
CTD Temperature (°C)	0.479830
Chlorophyll_a	-0.319169
Total Phosphorus (µg P/L)	0.371485
Total Dissolved Phosphorus (µg P/L)	0.376437
Particulate Organic Nitrogen (mg/L)	-0.099448
Particulate Organic Carbon (mg/L)	-0.131561
Dissolved Organic Carbon (mg/L)	1.000000
Colored Dissolved Organic Material absorbance (...)	0.720621
Total Suspended Solids (mg/L)	-0.146292
Volatile Suspended Solids (mg/L)	-0.189884

Colored Dissolved Organic Material absorbance (m-1) at 400nm \

```

Bloom Extent (KM2) -0.342159
Bloom Intensity (ug/L) 0.275269
CTD Temperature (°C) 0.248849
Chlorophyll_a -0.166047
Total Phosphorus (µg P/L) 0.392093
Total Dissolved Phosphorus (µg P/L) 0.548878
Particulate Organic Nitrogen (mg/L) -0.169900
Particulate Organic Carbon (mg/L) -0.201603
Dissolved Organic Carbon (mg/L) 0.720621
Colored Dissolved Organic Material absorbance (...) 1.000000
Total Suspended Solids (mg/L) 0.144074
Volatile Suspended Solids (mg/L) -0.092176

```

```

Total Suspended Solids (mg/L) \
Bloom Extent (KM2) 0.190316
Bloom Intensity (ug/L) -0.345269
CTD Temperature (°C) -0.332781
Chlorophyll_a 0.020979
Total Phosphorus (µg P/L) 0.462357
Total Dissolved Phosphorus (µg P/L) 0.318059
Particulate Organic Nitrogen (mg/L) 0.389950
Particulate Organic Carbon (mg/L) 0.359638
Dissolved Organic Carbon (mg/L) -0.146292
Colored Dissolved Organic Material absorbance (...) 0.144074
Total Suspended Solids (mg/L) 1.000000
Volatile Suspended Solids (mg/L) 0.567244

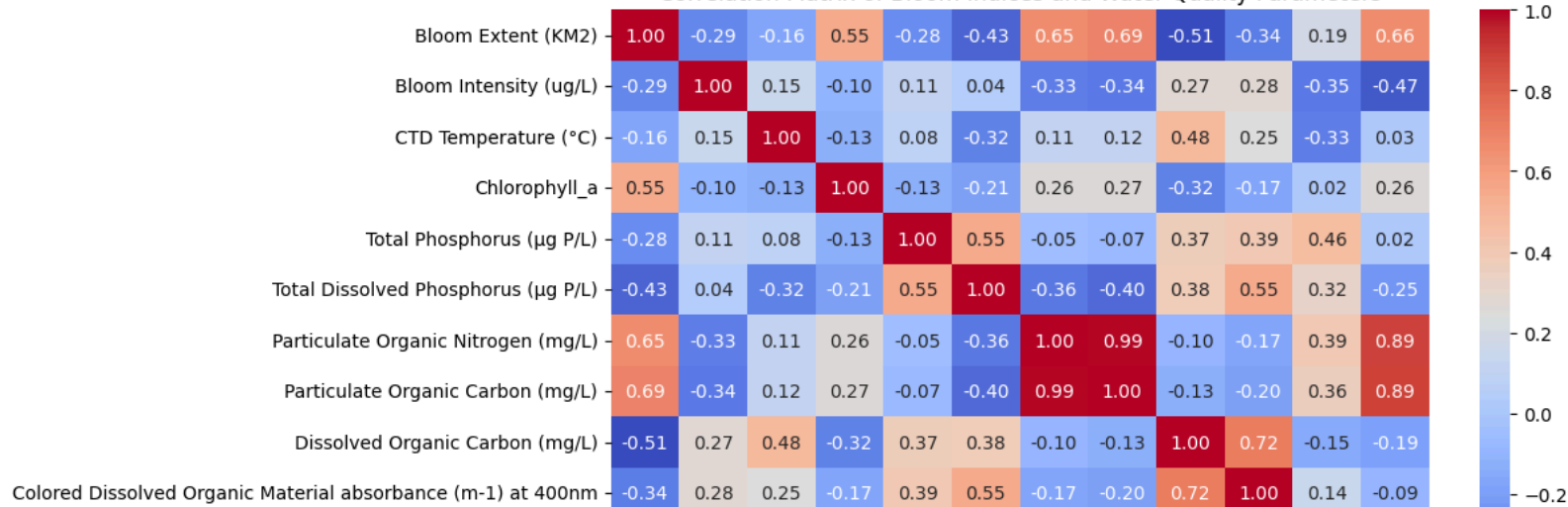
```

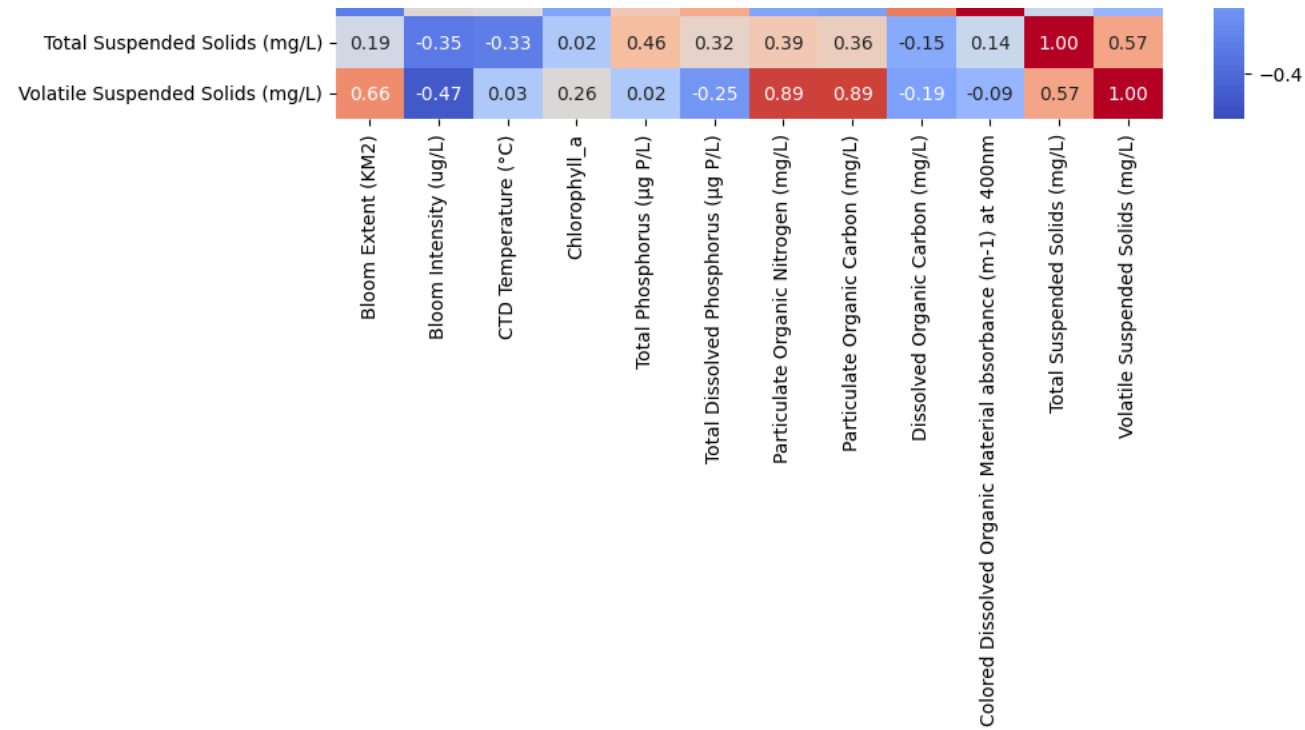
```

Volatile Suspended Solids (mg/L)
Bloom Extent (KM2) 0.658319
Bloom Intensity (ug/L) -0.465574
CTD Temperature (°C) 0.027929
Chlorophyll_a 0.262212
Total Phosphorus (µg P/L) 0.018401
Total Dissolved Phosphorus (µg P/L) -0.247314
Particulate Organic Nitrogen (mg/L) 0.886939
Particulate Organic Carbon (mg/L) 0.890582
Dissolved Organic Carbon (mg/L) -0.189884
Colored Dissolved Organic Material absorbance (...) -0.092176
Total Suspended Solids (mg/L) 0.567244
Volatile Suspended Solids (mg/L) 1.000000

```

Correlation Matrix of Bloom Indices and Water Quality Parameters





```

1 print(bloom_data.columns)
2 print(water_quality_daily.columns)
3

Index(['Date', 'Lake', 'Satellite Sensor', 'Bloom Extent (KM2)',
      'Bloom Extent (% of Lake Area)', 'Bloom Intensity (ug/L)',
      'Bloom Severity (ug/L km2)', 'Valid Pixels (% of Lake Area)'],
      dtype='object')
Index(['Date', 'Station Depth (m)', 'Sample Depth (m)',
      'Latitude (decimal deg)', 'Longitude (decimal deg)', 'Wave Height (ft)',
      'Sample Temperature (°C)', 'CTD Temperature (°C)',
      'CTD Specific Conductivity (µS/cm)',
      'CTD Photosynthetically Active Radiation (µE/m2/s)', 'Turbidity (NTU)',
      'Chlorophyll_a', 'Total Phosphorus (µg P/L)',
      'Total Dissolved Phosphorus (µg P/L)',
      'Particulate Organic Carbon (mg/L)',
      'Particulate Organic Nitrogen (mg/L)',
      'Dissolved Organic Carbon (mg/L)',
      'Colored Dissolved Organic Material absorbance (m-1) at 400nm',
      'Total Suspended Solids (mg/L)', 'Volatile Suspended Solids (mg/L)'],
      dtype='object')

1 # Drop unnecessary columns
2 bloom_data = bloom_data.drop(columns=['level_0', 'index'], errors='ignore')
3 water_quality_daily = water_quality_daily.drop(columns=['level_0', 'index'], errors='ignore')
4
5 # Ensure 'Date' is a column, not the index
6 bloom_data.reset_index(drop=True, inplace=True)
7 water_quality_daily.reset_index(drop=True, inplace=True)
8
9 # Merge the two datasets on 'Date' to ensure alignment
10 aligned_data = pd.merge(
11     bloom_data[['Date', 'Bloom Extent (KM2)', 'Bloom Severity (ug/L km2)']],
12     water_quality_daily[['Date', 'Chlorophyll_a', 'CTD Temperature (°C)', 'Total Phosphorus (µg P/L)', 'Particulate Organic Nitrogen (mg/L)']],
13     on='Date', how='inner'
14 )
15
16 # Drop rows with missing data
17 aligned_data = aligned_data.dropna()
18
19 # Display the first few rows of the aligned dataset
20 print(aligned_data.head())
21

```

```

Index(['Date', 'Bloom Extent (KM2)', 'Bloom Severity (ug/L km2)', 'Chlorophyll_a',
      'CTD Temperature (°C)', 'Total Phosphorus (µg P/L)', 'Particulate Organic Nitrogen (mg/L)'],
      dtype='object')

```

	Date	Bloom Extent (KM2)	Bloom Severity (ug/L km2)	Chlorophyll_a \
0	2016-06-13	136.08	6703.63	14.980000
1	2016-06-27	154.53	7567.06	4.651667
2	2016-07-05	149.49	7631.85	6.260833
3	2016-07-11	158.85	7946.17	10.446667
4	2016-07-18	157.77	7864.66	23.877500

	CTD Temperature (°C)	Total Phosphorus (µg P/L) \
0	21.575000	87.5625
1	24.191667	77.9875
2	23.450000	75.5875
3	24.933333	60.5875
4	24.766667	61.7000

	Particulate Organic Nitrogen (mg/L)
0	0.18625

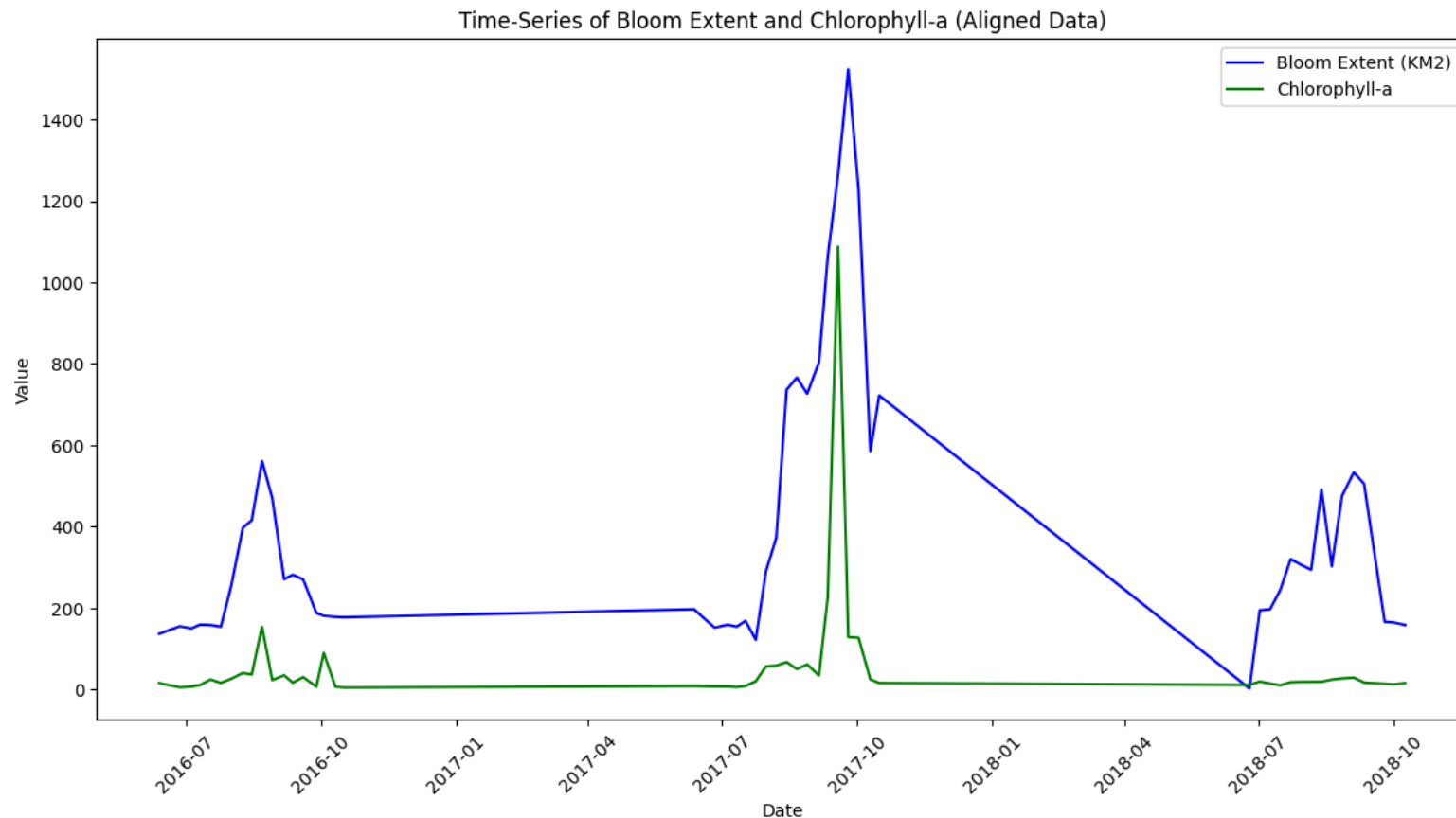
1	0.14250
2	0.11625
3	0.16375
4	0.18125

```
1 print(aligned_data.columns)
2 print(aligned_data.isnull().sum())
3
```

```
Index(['Date', 'Bloom Extent (KM2)', 'Bloom Severity (ug/L km2)',
      'Chlorophyll_a', 'CTD Temperature (°C)', 'Total Phosphorus (µg P/L)',
      'Particulate Organic Nitrogen (mg/L)'],
      dtype='object')
```

Date	0
Bloom Extent (KM2)	0
Bloom Severity (ug/L km2)	0
Chlorophyll_a	0
CTD Temperature (°C)	0
Total Phosphorus (µg P/L)	0
Particulate Organic Nitrogen (mg/L)	0
dtype: int64	

```
1 # Plot the aligned time-series data
2 plt.figure(figsize=(14, 7))
3 sns.lineplot(x='Date', y='Bloom Extent (KM2)', data=aligned_data, label='Bloom Extent (KM2)', color='blue')
4 sns.lineplot(x='Date', y='Chlorophyll_a', data=aligned_data, label='Chlorophyll-a', color='green')
5 plt.title('Time-Series of Bloom Extent and Chlorophyll-a (Aligned Data)')
6 plt.xlabel('Date')
7 plt.ylabel('Value')
8 plt.legend()
9 plt.xticks(rotation=45)
10 plt.show()
```



```

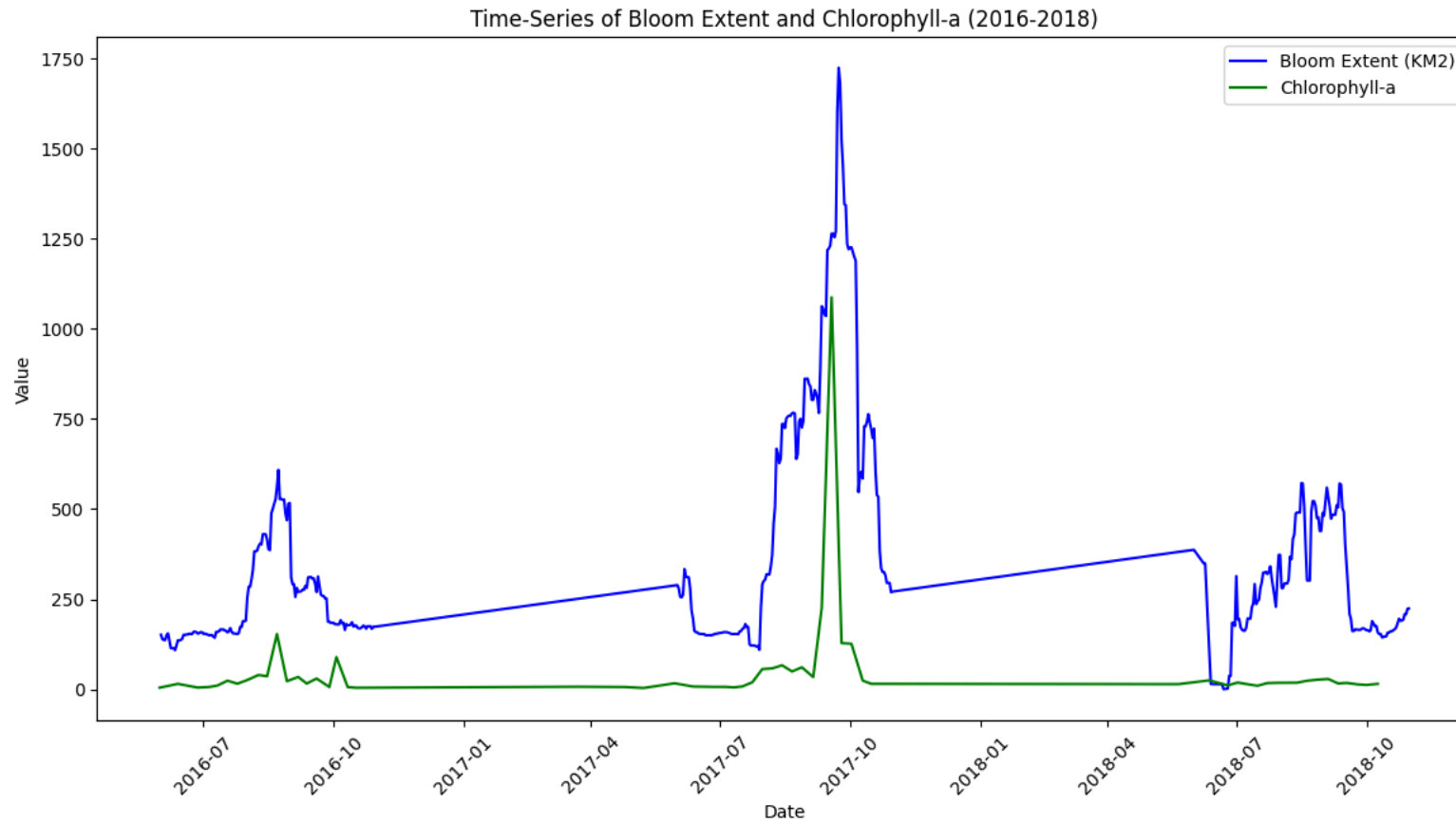
1 # Filter data for the range 2016 to 2018 inclusive
2 bloom_data_filtered = bloom_data[(bloom_data['Date'] >= '2016-01-01') & (bloom_data['Date'] <= '2018-12-31')]
3 water_quality_filtered = water_quality_daily[(water_quality_daily['Date'] >= '2016-01-01') & (water_quality_daily['Date'] <= '2018-12-31')]
4
5 # Merge the filtered datasets (outer join to include all dates in the range)
6 merged_filtered_data = pd.merge(bloom_data_filtered[['Date', 'Bloom Extent (KM2)']],
7                                water_quality_filtered[['Date', 'Chlorophyll_a']],
8                                on='Date', how='outer')
9
10 # Sort by date for consistent plotting
11 merged_filtered_data = merged_filtered_data.sort_values(by='Date')
12
13 # Plot the full range data
14 plt.figure(figsize=(14, 7))
15 sns.lineplot(x='Date', y='Bloom Extent (KM2)', data=merged_filtered_data, label='Bloom Extent (KM2)', color='blue')
16 sns.lineplot(x='Date', y='Chlorophyll_a', data=merged_filtered_data, label='Chlorophyll-a', color='green')
17 plt.title('Time-Series of Bloom Extent and Chlorophyll-a (2016-2018)')
18 plt.xlabel('Date')
19 plt.ylabel('Value')
20 plt.legend()
21 plt.xticks(rotation=45)

```



```
22 plt.show()
```

```
23
```



2.3 Anomaly Detection

Detect unusual bloom severity events for further investigation.

```
1 # Compute Z-scores for Bloom Severity
2 aligned_data['Bloom_Severity_Z'] = (
3     (aligned_data['Bloom Severity (ug/L km2)'] - aligned_data['Bloom Severity (ug/L km2)'].mean()) /
4     aligned_data['Bloom Severity (ug/L km2)'].std()
5 )
6
7 # Identify anomalies where Z-score > 2 or < -2
8 anomalies = aligned_data[(aligned_data['Bloom_Severity_Z'] > 2) | (aligned_data['Bloom_Severity_Z'] < -2)]
9
10 # Display anomalies
11 print("\nAnomalies in Bloom Severity:")
```

```
12 print(anomalies[['Date', 'Bloom Severity (ug/L km2)', 'Bloom_Severity_Z']])
13
```



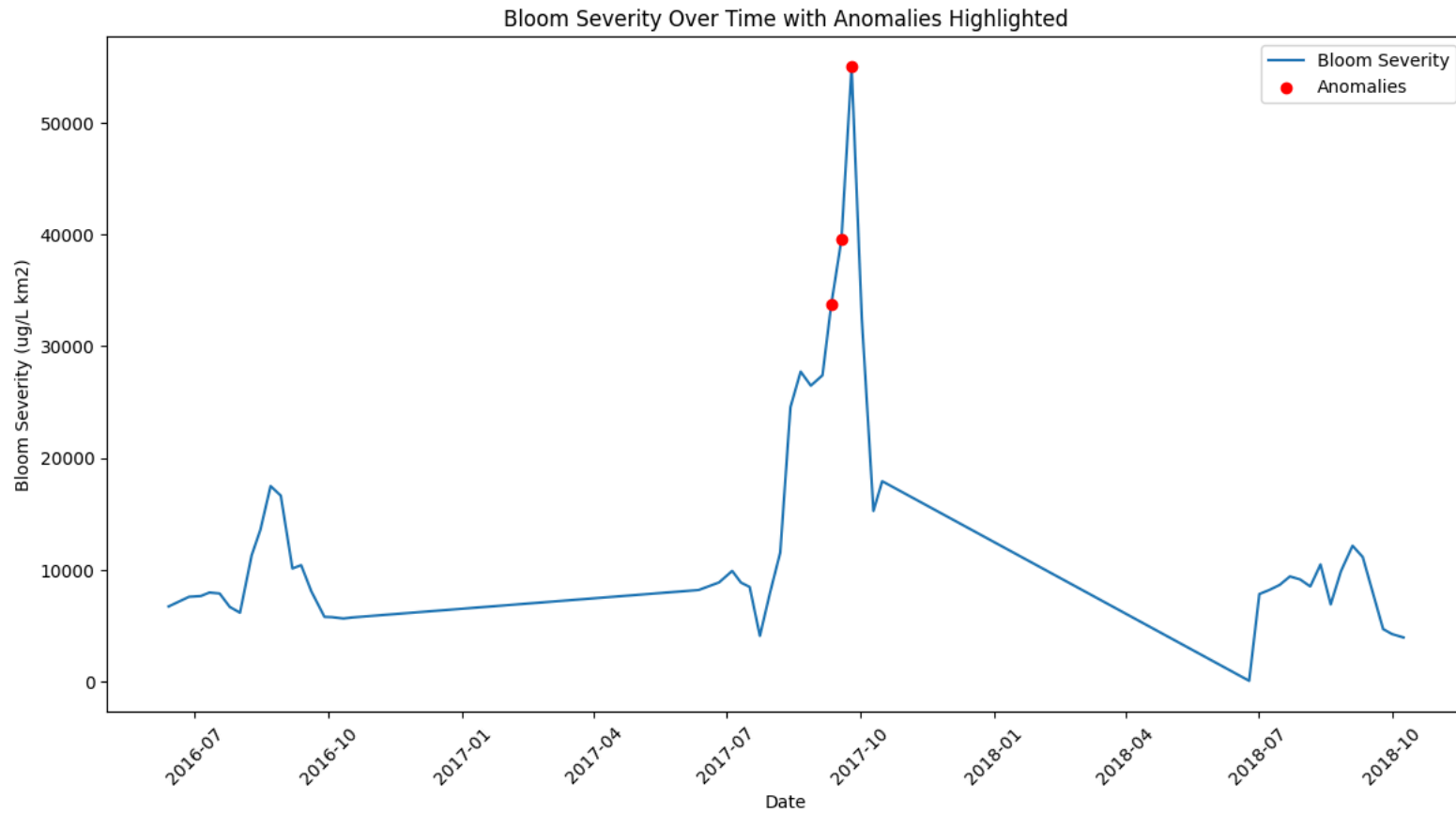
Anomalies in Bloom Severity:

	Date	Bloom Severity (ug/L km2)	Bloom_Severity_Z
30	2017-09-11	33721.23	2.042731
31	2017-09-18	39543.10	2.606389
32	2017-09-25	55014.21	4.104259

Visualize the Anomalies:

Plot Bloom Severity over time, with the anomaly dates highlighted to provide a clear visual representation of their extremity.

```
1 # Plot Bloom Severity with Anomalies Highlighted
2 plt.figure(figsize=(14, 7))
3 sns.lineplot(x='Date', y='Bloom Severity (ug/L km2)', data=aligned_data, label='Bloom Severity')
4 plt.scatter(anomalies['Date'], anomalies['Bloom Severity (ug/L km2)'], color='red', label='Anomalies', zorder=5)
5 plt.title('Bloom Severity Over Time with Anomalies Highlighted')
6 plt.xlabel('Date')
7 plt.ylabel('Bloom Severity (ug/L km2)')
8 plt.legend()
9 plt.xticks(rotation=45)
10 plt.show()
11
```



Summary of Section 2

- Visualized distributions and correlations.
- Key Insight: Chlorophyll-a strongly correlates with bloom severity.

✓ 3. Model Development and Training

✓ 3.1 Time-Series Forecasting Model

✓ 3.1.1 Preparing Data for Time-Series Modeling

Prepare the data in the correct format for time-series forecasting.

```

1 # Ensure 'Date' is present and converted to datetime
2 if 'Date' not in bloom_data.columns:
3     bloom_data.reset_index(inplace=True) # Reset index if 'Date' is currently the index
4
5 bloom_data['Date'] = pd.to_datetime(bloom_data['Date'], errors='coerce') # Convert to datetime
6
7 # Drop any rows where 'Date' could not be parsed
8 bloom_data.dropna(subset=['Date'], inplace=True)
9
10 # Set 'Date' as the index
11 bloom_data.set_index('Date', inplace=True)
12
13 # Resample Bloom Severity data to monthly frequency and fill missing values
14 monthly_bloom = bloom_data['Bloom Severity (ug/L km2)'].resample('ME').mean().ffill() # Updated syntax
15
16 # Display the first few rows of the resampled data
17 print(monthly_bloom.head())
18

```

```

→ Date
2002-06-30    2295.754286
2002-07-31    6197.530000
2002-08-31    9033.583333
2002-09-30    3141.056667
2002-10-31    7056.688824
Freq: ME, Name: Bloom Severity (ug/L km2), dtype: float64

```

✓ 3.1.2 ARIMA Model

Develop a statistical time-series model to forecast bloom severity.

```

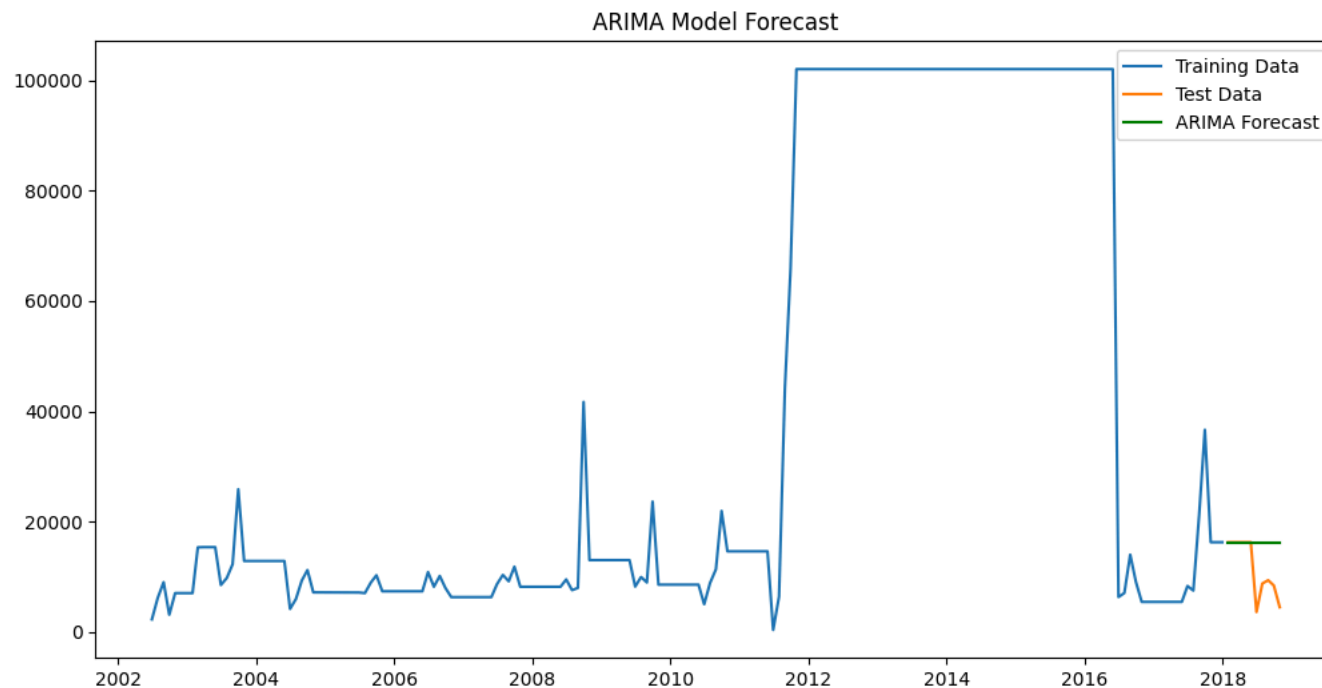
1 # Import ARIMA model
2 from statsmodels.tsa.arima.model import ARIMA
3
4 # Split data into training and testing sets
5 train_data = monthly_bloom[:'2017']
6 test_data = monthly_bloom['2018':]
7
8 # Fit ARIMA model
9 model_arima = ARIMA(train_data, order=(1, 1, 1))
10 arima_result = model_arima.fit()
11
12 # Forecast
13 forecast_arima = arima_result.predict(start=test_data.index[0], end=test_data.index[-1], typ='levels')
14
15 # Plot forecasts
16 plt.figure(figsize=(12, 6))
17 plt.plot(train_data, label='Training Data')
18 plt.plot(test_data, label='Test Data')
19 plt.plot(forecast_arima, label='ARIMA Forecast', color='green')
20 plt.legend()
21 plt.title('ARIMA Model Forecast')
22 plt.show()
23
24

```

```

/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters found. Using zeros as starting parameters.
warn('Non-stationary starting autoregressive parameters found. Using zeros as starting parameters.')
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using zeros as starting parameters.
warn('Non-invertible starting MA parameters found. Using zeros as starting parameters.')
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/representation.py:374: FutureWarning: Unknown keyword arguments: dict_keys(['typ']).Passing unknown keyword argument
warnings.warn(msg, FutureWarning)

```



```

1 # Apply differencing to make the data stationary
2 monthly_bloom_diff = monthly_bloom.diff().dropna()
3
4 # Reassess stationarity with Augmented Dickey-Fuller test
5 from statsmodels.tsa.stattools import adfuller
6
7 adf_test = adfuller(monthly_bloom_diff)
8 print("ADF Statistic:", adf_test[0])
9 print("p-value:", adf_test[1])
10 if adf_test[1] < 0.05:
11     print("The data is stationary.")
12 else:
13     print("The data is not stationary. Further transformations may be required.")
14
15 ADF Statistic: -13.877731330116458
16 p-value: 6.293234184492074e-26
17 The data is stationary.

```

```

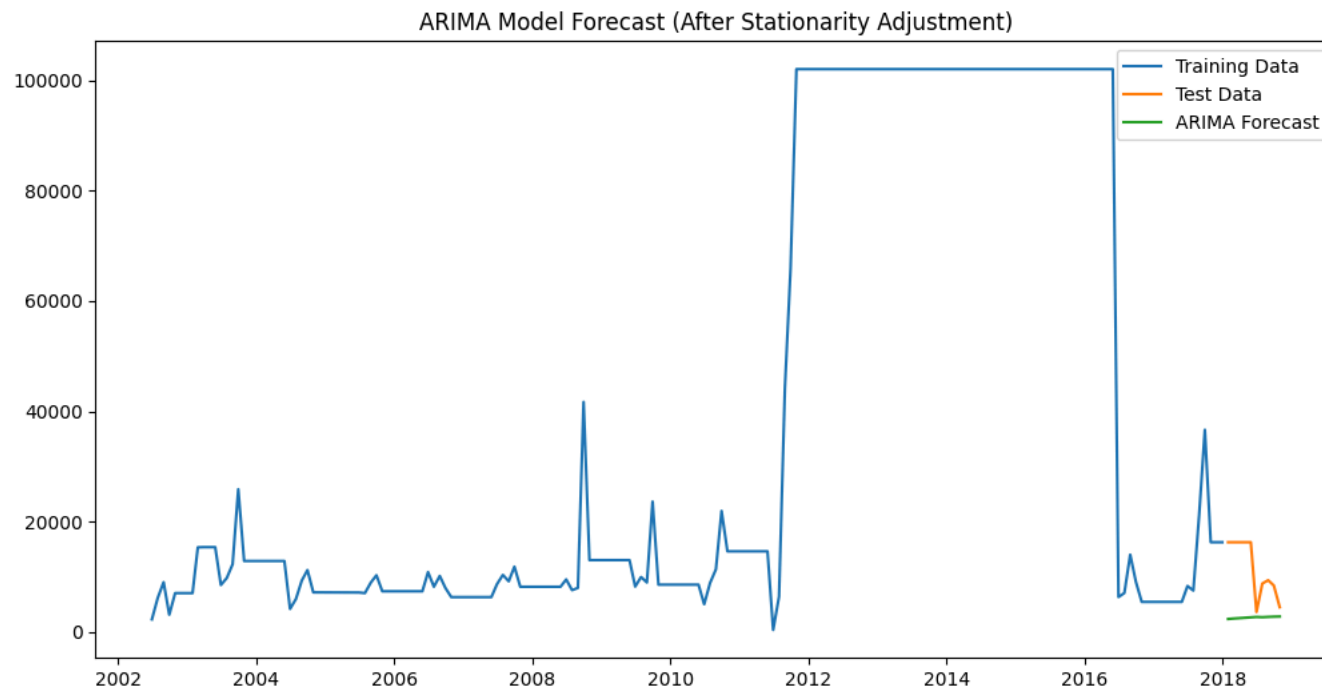
1 # Refit ARIMA model after data transformation
2 model_arima = ARIMA(monthly_bloom_diff, order=(1, 1, 1)) # Adjust parameters based on ACF/PACF
3 arima_result = model_arima.fit()

```

```

4
5 # Forecast
6 forecast_arma_diff = arima_result.predict(start=test_data.index[0], end=test_data.index[-1])
7
8 # Revert differencing to interpret predictions
9 forecast_arma = forecast_arma_diff.cumsum() + monthly_bloom.iloc[0] # Add back the first value for interpretation
10
11 # Plot updated forecasts
12 plt.figure(figsize=(12, 6))
13 plt.plot(train_data, label='Training Data')
14 plt.plot(test_data, label='Test Data')
15 plt.plot(forecast_arma, label='ARIMA Forecast')
16 plt.legend()
17 plt.title('ARIMA Model Forecast (After Stationarity Adjustment)')
18 plt.show()
19

```



```

1 from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
2 import numpy as np
3
4 # Evaluate ARIMA Model
5 mse_arma = mean_squared_error(test_data, forecast_arma)
6 mae_arma = mean_absolute_error(test_data, forecast_arma)
7 rmse_arma = np.sqrt(mse_arma)
8 r2_arma = r2_score(test_data, forecast_arma)
9
10 print(f"ARIMA Model MSE: {mse_arma}")
11 print(f"ARIMA Model MAE: {mae_arma}")

```

```
12 print(f"ARIMA Model RMSE: {rmse_arima}")
13 print(f"ARIMA Model R²: {r2_arima}")
```

```
ARIMA Model MSE: 106344223.51937824
ARIMA Model MAE: 8975.555987042782
ARIMA Model RMSE: 10312.333563232827
ARIMA Model R²: -3.3199373971299675
```

3.1.3 LSTM Model


Develop a neural network model to capture complex patterns in bloom severity.

```
1 # Import libraries for LSTM
2 import numpy as np
3 from sklearn.preprocessing import MinMaxScaler
4 from tensorflow.keras.models import Sequential
5 from tensorflow.keras.layers import LSTM, Dense
6
7 # Prepare data for LSTM
8 scaler = MinMaxScaler()
9 monthly_bloom_scaled = scaler.fit_transform(monthly_bloom.values.reshape(-1, 1))
10
11 # Create sequences
12 def create_sequences(data, seq_length):
13     X = []
14     y = []
15     for i in range(len(data) - seq_length):
16         X.append(data[i:i + seq_length])
17         y.append(data[i + seq_length])
18     return np.array(X), np.array(y)
19
20 seq_length = 12
21 X, y = create_sequences(monthly_bloom_scaled, seq_length)
22
23 # Split into training and testing sets
24 train_size = len(train_data)
25 X_train, X_test = X[:train_size - seq_length], X[train_size - seq_length:]
26 y_train, y_test = y[:train_size - seq_length], y[train_size - seq_length:]
27
28 # Build LSTM model
29 model_lstm = Sequential()
30 model_lstm.add(LSTM(50, activation='relu', input_shape=(seq_length, 1)))
31 model_lstm.add(Dense(1))
32 model_lstm.compile(optimizer='adam', loss='mse')
33
34 # Train the model
35 model_lstm.fit(X_train, y_train, epochs=50, batch_size=1, verbose=0)
36
37 # Forecast
38 lstm_predictions = model_lstm.predict(X_test)
39 lstm_predictions = scaler.inverse_transform(lstm_predictions)
40
41 # Plot forecasts
42 plt.figure(figsize=(12, 6))
43 plt.plot(test_data.index, test_data.values, label='Test Data')
44 plt.plot(test_data.index, lstm_predictions, label='LSTM Forecast', color='orange')
```

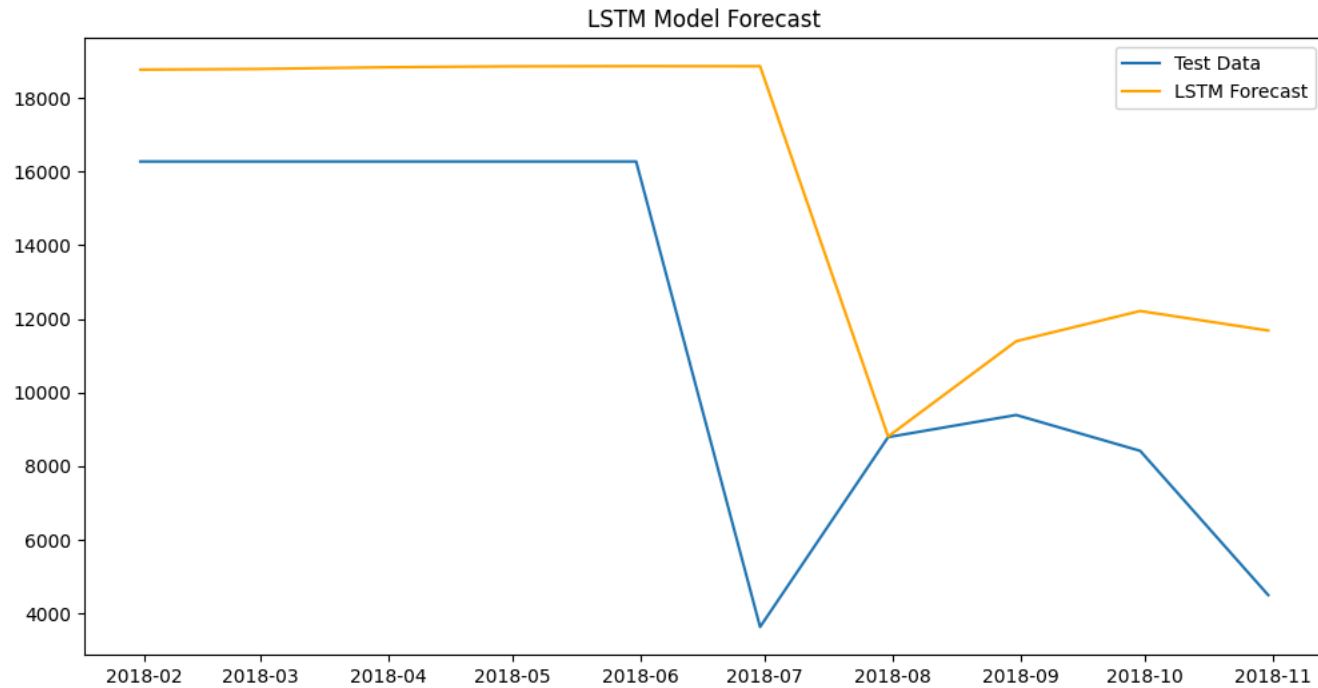
```

45 plt.legend()
46 plt.title('LSTM Model Forecast')
47 plt.show()
48
49

```

 /usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer super().__init__(**kwargs)

1/1 ————— 0s 265ms/step



```

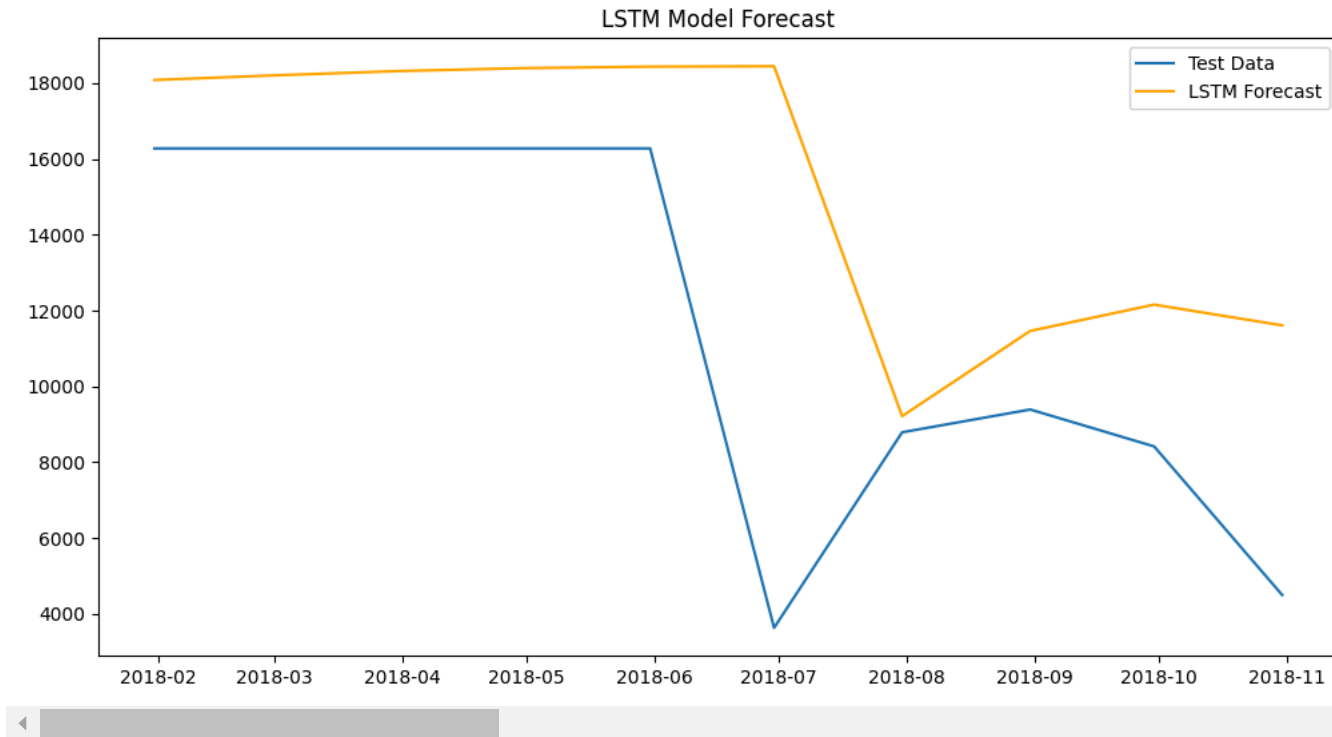
1 # Import libraries for LSTM
2 import numpy as np
3 import tensorflow as tf
4 from sklearn.preprocessing import MinMaxScaler
5 from tensorflow.keras.models import Sequential
6 from tensorflow.keras.layers import LSTM, Dense, Input
7 import random
8
9 # Set random seeds for reproducibility
10 np.random.seed(42)
11 random.seed(42)
12 tf.random.set_seed(42)
13
14 # Prepare data for LSTM
15 scaler = MinMaxScaler()
16 monthly_bloom_scaled = scaler.fit_transform(monthly_bloom.values.reshape(-1, 1))
17
18 # Create sequences
19 def create_sequences(data, seq_length):

```



```
20 X, y = [], []
21 for i in range(len(data) - seq_length):
22     X.append(data[i:i + seq_length])
23     y.append(data[i + seq_length])
24 return np.array(X), np.array(y)
25
26 seq_length = 12
27 X, y = create_sequences(monthly_bloom_scaled, seq_length)
28
29 # Split into training and testing sets
30 train_size = len(train_data)
31 X_train, X_test = X[:train_size - seq_length], X[train_size - seq_length:]
32 y_train, y_test = y[:train_size - seq_length], y[train_size - seq_length:]
33
34 # Build LSTM model
35 model_lstm = Sequential([
36     Input(shape=(seq_length, 1)),
37     LSTM(50, activation='relu'),
38     Dense(1)
39 ])
40 model_lstm.compile(optimizer='adam', loss='mse')
41
42 # Train the model
43 model_lstm.fit(X_train, y_train, epochs=50, batch_size=1, verbose=0)
44
45 # Forecast
46 lstm_predictions = model_lstm.predict(X_test)
47 lstm_predictions = scaler.inverse_transform(lstm_predictions)
48
49 # Plot forecasts
50 plt.figure(figsize=(12, 6))
51 plt.plot(test_data.index, test_data.values, label='Test Data')
52 plt.plot(test_data.index, lstm_predictions, label='LSTM Forecast', color='orange')
53 plt.legend()
54 plt.title('LSTM Model Forecast')
55 plt.show()
56
```

WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x7a3fe0ceb130> triggered tf.function retracing
1/1 ————— 0s 159ms/step



```

1 from sklearn.metrics import mean_squared_error, mean_absolute_error
2
3 # Evaluate LSTM Model
4 mse_lstm = mean_squared_error(y_test, lstm_predictions)
5 mae_lstm = mean_absolute_error(y_test, lstm_predictions)
6 rmse_lstm = mse_lstm ** 0.5
7
8 print(f"LSTM Model MSE: {mse_lstm}")
9 print(f"LSTM Model MAE: {mae_lstm}")
10 print(f"LSTM Model RMSE: {rmse_lstm}")
11

```

LSTM Model MSE: 262131291.5098289
 LSTM Model MAE: 15710.35317870916
 LSTM Model RMSE: 16190.46915656952

3.2 Anomaly Detection Model

3.2.1 Isolation Forest

Detect anomalies in environmental parameters that may precede bloom events.

```

1 # Prepare data (exclude Total Phosphorus since it's unavailable)
2 features = aligned_data[['CTD Temperature (°C)', 'Chlorophyll_a']].ffill()
3
4 # Fit Isolation Forest
5 from sklearn.ensemble import IsolationForest
6
7 iso_forest = IsolationForest(contamination=0.05, random_state=42)
8 iso_forest.fit(features)
9
10 # Add anomaly scores and labels to the aligned dataset
11 aligned_data['Anomaly_Score'] = iso_forest.decision_function(features)
12 aligned_data['Anomaly'] = iso_forest.predict(features)
13
14 # Filter for anomalies (label = -1)
15 anomalies_if = aligned_data[aligned_data['Anomaly'] == -1]
16
17 # Display anomalies
18 print("\nAnomalies detected by Isolation Forest:")
19 print(anomalies_if[['Date', 'Anomaly_Score']])
20

```



Anomalies detected by Isolation Forest:

	Date	Anomaly_Score
9	2016-08-22	-0.014817
30	2017-09-11	-0.048949
31	2017-09-18	-0.204188

```

1 # Convert the Isolation forest anomalies results output to a DataFrame
2 anomalies_if_df = pd.DataFrame(anomalies_if)
3
4 # Display the DataFrame using Pandas
5 from IPython.display import display
6
7 # Display the DataFrame
8 display(anomalies_if_df)
9
10 # Define the file path for saving the merged dataframe
11 output_file_path = "/content/drive/MyDrive/CIND820/AnomaliesDetected_Isolation_Forest_Output.csv"
12
13 # Save the merged dataframe to the specified location as a CSV file
14 anomalies_if_df.to_csv(output_file_path, index=False)
15

```



	Date	Bloom Extent (KM2)	Bloom Severity (ug/L km2)	Chlorophyll_a	CTD Temperature (°C)	Total Phosphorus (µg P/L)	Particulate Organic Nitrogen (mg/L)	Bloom_Severity_Z	Anomaly_Score	Anomaly
9	2016-08-22	560.88	17480.22	153.321429	25.033333	117.81250	0.3375	0.470322	-0.014817	-1
30	2017-09-11	1062.99	33721.23	225.735714	18.675000	51.80750	0.8025	2.042731	-0.048949	-1
31	2017-09-18	1264.59	39543.10	1087.804667	20.975000	42.47375	0.4375	2.606389	-0.204188	-1

Next steps:

[Generate code with anomalies_if_df](#)

[View recommended plots](#)

[New interactive sheet](#)

✓ 3.2.2 DBSCAN Clustering

Use clustering to identify outliers in the data.

```

1 # Import DBSCAN
2 from sklearn.cluster import DBSCAN
3 from sklearn.preprocessing import StandardScaler
4
5 # Standardize features
6 scaler = StandardScaler()
7 features_scaled = scaler.fit_transform(features)
8
9 # Fit DBSCAN
10 dbscan = DBSCAN(eps=0.5, min_samples=5)
11 dbscan.fit(features_scaled)
12
13 # Add cluster labels to aligned data
14 aligned_data['Cluster'] = dbscan.labels_
15
16 # Identify noise points (anomalies)
17 anomalies_dbscan = aligned_data[aligned_data['Cluster'] == -1]
18
19 print("\nAnomalies detected by DBSCAN:")
20 print(anomalies_dbscan[['Date', 'Cluster']])
21

```



Anomalies detected by DBSCAN:

	Date	Cluster
9	2016-08-22	-1
15	2016-10-03	-1
16	2016-10-11	-1
17	2016-10-17	-1
30	2017-09-11	-1
31	2017-09-18	-1
33	2017-10-02	-1
35	2017-10-16	-1

```

1 # Convert the DBSCAN Clustering anomalies results output to a DataFrame
2 anomalies_dbscan_df = pd.DataFrame(anomalies_dbscan)
3
4 # Display the DataFrame using Pandas
5 from IPython.display import display
6
7 # Display the DataFrame
8 display(anomalies_dbscan_df)
9
10 # Define the file path for saving the merged dataframe
11 output_file_path = "/content/drive/MyDrive/CIND820/AnomaliesDetected_DBSCAN_Clustering_Output.csv"
12
13 # Save the merged dataframe to the specified location as a CSV file
14 anomalies_dbscan_df.to_csv(output_file_path, index=False)

```



	Date	Bloom Extent (KM2)	Bloom Severity (ug/L km2)	Chlorophyll_a	CTD Temperature (°C)	Total Phosphorus (µg P/L)	Particulate Organic Nitrogen (mg/L)	Bloom_Severity_Z	Anomaly_Score	Anomaly	Cluster
9	2016-08-22	560.88	17480.22	153.321429	25.033333	117.812500	0.337500	0.470322	-0.014817	-1	-1
15	2016-10-03	180.36	5748.81	89.309231	18.666667	81.375000	0.155000	-0.665481	0.059830	1	-1
16	2016-10-11	177.75	5623.83	5.950833	17.116667	61.275000	0.157500	-0.677581	0.048634	1	-1
17	2016-10-17	176.85	5716.61	4.143333	16.791667	46.387500	0.092500	-0.668598	0.029412	1	-1
30	2017-09-11	1062.99	33721.23	225.735714	18.675000	51.807500	0.802500	2.042731	-0.048949	-1	-1
31	2017-09-18	1264.59	39543.10	1087.804667	20.975000	42.473750	0.437500	2.606389	-0.204188	-1	-1
33	2017-10-02	1226.61	32501.73	126.275385	19.433333	56.975000	0.353750	1.924663	0.048883	1	-1
35	2017-10-16	722.07	17918.82	15.100000	16.609091	50.082857	0.237143	0.512786	0.014817	1	-1



Next steps:

[Generate code with anomalies_dbscan_df](#)[View recommended plots](#)[New interactive sheet](#)1 Start coding or [generate](#) with AI.

3.3 Develop and Evaluate a SARIMA Model

Model Development and Training:

- Implement SARIMA for forecasting, ensuring it accounts for seasonality in bloom severity.

```

1 from statsmodels.tsa.statespace.sarimax import SARIMAX
2 from sklearn.metrics import mean_squared_error
3
4 # SARIMA model implementation
5 sarima_model = SARIMAX(train_data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
6 sarima_result = sarima_model.fit()
7
8 # Forecast with SARIMA
9 forecast_sarima = sarima_result.predict(start=test_data.index[0], end=test_data.index[-1])
10

```

```

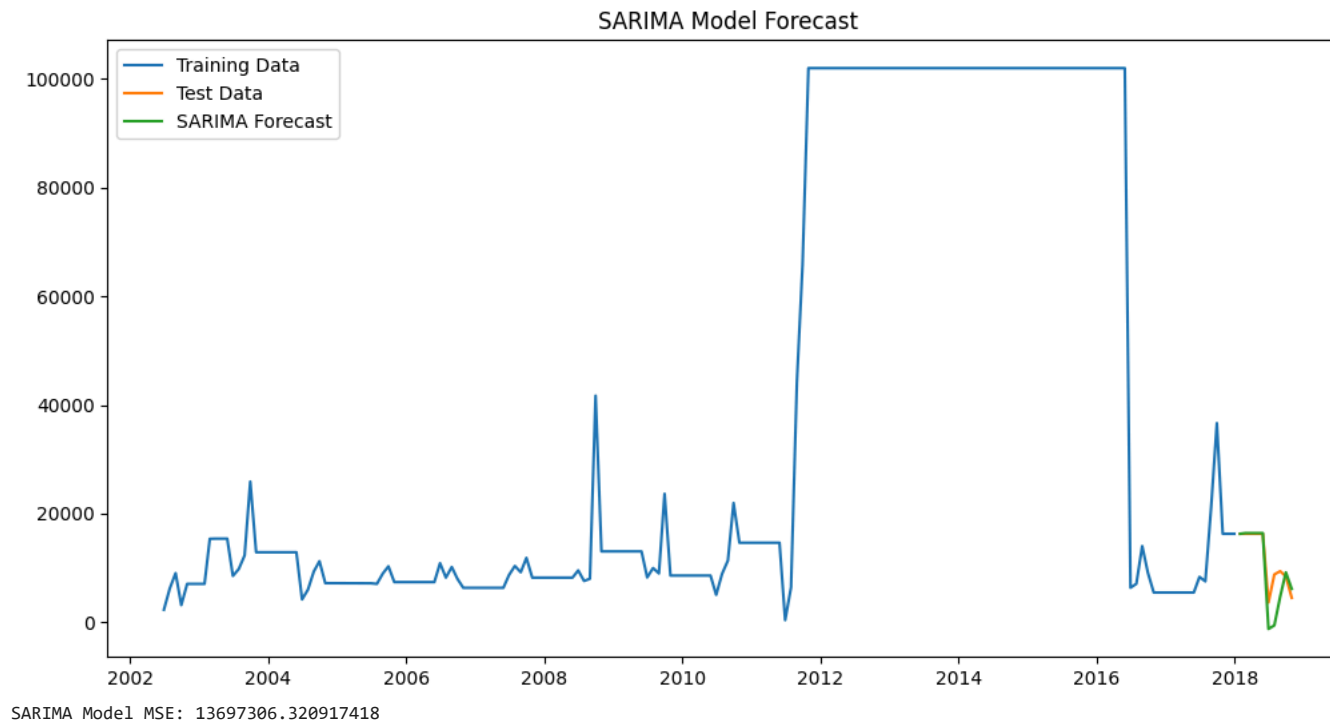
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters found. Using zeros as starting parameters.
warn('Non-stationary starting autoregressive parameters')
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using zeros as starting parameters.
warn('Non-invertible starting MA parameters found.')

```

```

1 # SARIMA model with seasonal components
2
3 # Plot SARIMA forecasts
4 plt.figure(figsize=(12, 6))
5 plt.plot(train_data, label='Training Data')
6 plt.plot(test_data, label='Test Data')
7 plt.plot(forecast_sarima, label='SARIMA Forecast')
8 plt.legend()
9 plt.title('SARIMA Model Forecast')
10 plt.show()
11
12 # Evaluate SARIMA model
13 mse_sarima = mean_squared_error(test_data, forecast_sarima)
14 print(f"SARIMA Model MSE: {mse_sarima}")
15

```



3.4 Model Evaluation and Comparison

Compare models based on prediction accuracy to select the best-performing model.

✓ Assess Metrics of LSTM Model

Model Evaluation:

- Add Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) for LSTM evaluation.

```
1 from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
2 import numpy as np
3
4 # Calculate additional metrics for LSTM
5 mse_lstm = mean_squared_error(test_data, lstm_predictions)
6 mae_lstm = mean_absolute_error(test_data, lstm_predictions)
7 rmse_lstm = np.sqrt(mse_lstm)
8 r2_lstm = r2_score(test_data, lstm_predictions)
9
10 print(f"LSTM Model MSE: {mse_lstm}")
11 print(f"LSTM Model MAE: {mae_lstm}")
12 print(f"LSTM Model RMSE: {rmse_lstm}")
13 print(f"LSTM Model R²: {r2_lstm}")
14
```

```
↳ LSTM Model MSE: 33456127.07923179
   LSTM Model MAE: 4097.97727504519
   LSTM Model RMSE: 5784.127166585447
   LSTM Model R²: -0.3590618253596989
```

✓ Assess Metrics of SARIMA Model

Model Evaluation:

- Similarly, compute additional evaluation metrics for SARIMA.

```
1 # Calculate additional metrics for SARIMA
2 mse_sarima = mean_squared_error(test_data, forecast_sarima)
3 mae_sarima = mean_absolute_error(test_data, forecast_sarima)
4 rmse_sarima = np.sqrt(mse_sarima)
5 r2_sarima = r2_score(test_data, forecast_sarima)
6
7 print(f"SARIMA Model MSE: {mse_sarima}")
8 print(f"SARIMA Model MAE: {mae_sarima}")
9 print(f"SARIMA Model RMSE: {rmse_sarima}")
10 print(f"SARIMA Model R²: {r2_sarima}")
11
```

```
↳ SARIMA Model MSE: 13697306.320917418
   SARIMA Model MAE: 2184.2086179559687
   SARIMA Model RMSE: 3700.987208964308
   SARIMA Model R²: 0.44358514400273463
```

✓ Compare LSTM and SARIMA Metrics

Model Evaluation:

- Directly compare the evaluation metrics of LSTM and SARIMA models.
- Combine all metrics for an in-depth comparison.

```

1 # Compare LSTM and SARIMA
2 print(f"LSTM Model MSE: {mse_lstm}")
3 print(f"SARIMA Model MSE: {mse_sarima}")
4
5 if mse_sarima < mse_lstm:
6     print("SARIMA outperforms LSTM for this dataset.")
7 else:
8     print("LSTM outperforms SARIMA for this dataset.")
9

```

→ LSTM Model MSE: 33456127.07923179
 SARIMA Model MSE: 13697306.320917418
 SARIMA outperforms LSTM for this dataset.

```

1 # Display comparison
2 print("Model Comparison:")
3 print(f"LSTM - MSE: {mse_lstm}, MAE: {mae_lstm}, RMSE: {rmse_lstm}")
4 print(f"SARIMA - MSE: {mse_sarima}, MAE: {mae_sarima}, RMSE: {rmse_sarima}")
5
6 if mse_sarima < mse_lstm:
7     print("SARIMA is the preferred model.")
8 else:
9     print("LSTM is the preferred model.")
10

```

→ Model Comparison:
 LSTM - MSE: 33456127.07923179, MAE: 4097.97727504519, RMSE: 5784.127166585447
 SARIMA - MSE: 13697306.320917418, MAE: 2184.2086179559687, RMSE: 3700.987208964308
 SARIMA is the preferred model.

```

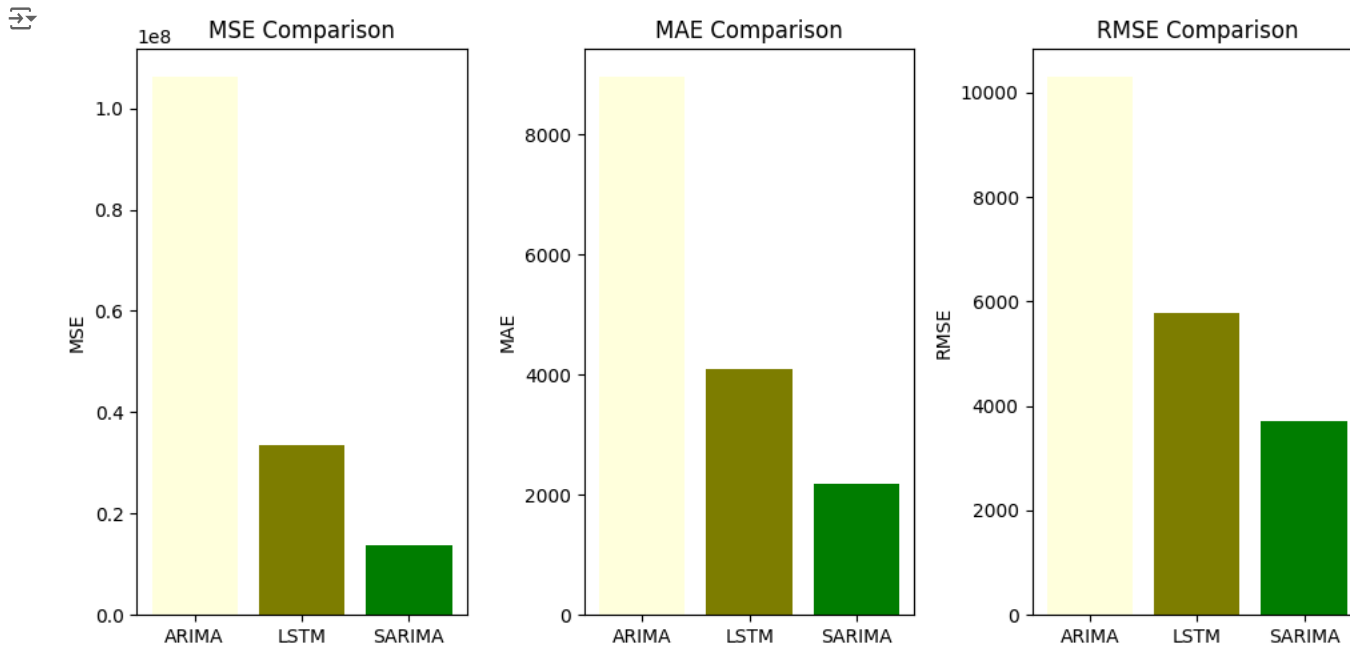
1 # **Compare LSTM and SARIMA Metrics**
2 metrics_comparison = {
3     'Metric': ['MSE', 'MAE', 'RMSE', 'R²'],
4     'ARIMA': [mse_arima, mae_arima, rmse_arima, r2_arima],
5     'LSTM': [mse_lstm, mae_lstm, rmse_lstm, r2_lstm],
6     'SARIMA': [mse_sarima, mae_sarima, rmse_sarima, r2_sarima]
7 }
8
9 # Create a DataFrame for the comparison
10 import pandas as pd
11 metrics_df = pd.DataFrame(metrics_comparison)
12
13 # Display the comparison table
14 print(metrics_df)
15
16

```

→

	Metric	ARIMA	LSTM	SARIMA
0	MSE	1.063442e+08	3.345613e+07	1.369731e+07
1	MAE	8.975556e+03	4.097977e+03	2.184209e+03
2	RMSE	1.031233e+04	5.784127e+03	3.700987e+03
3	R²	-3.319937e+00	-3.590618e-01	4.435851e-01


```
1 import matplotlib.pyplot as plt
2
3 # Metrics for comparison
4 metrics = {
5     'MSE': [mse_arima, mse_lstm, mse_sarima],
6     'MAE': [mae_arima, mae_lstm, mae_sarima],
7     'RMSE': [rmse_arima, rmse_lstm, rmse_sarima]
8 }
9
10 # Model labels
11 models = ['ARIMA', 'LSTM', 'SARIMA']
12
13 # Create subplots for MSE, MAE, and RMSE
14 fig, axes = plt.subplots(1, 3, figsize=(10, 5))
15
16 # Plot MSE
17 axes[0].bar(models, metrics['MSE'], color=['lightyellow', 'olive', 'green'])
18 axes[0].set_title('MSE Comparison')
19 axes[0].set_ylabel('MSE')
20
21 # Plot MAE
22 axes[1].bar(models, metrics['MAE'], color=['lightyellow', 'olive', 'green'])
23 axes[1].set_title('MAE Comparison')
24 axes[1].set_ylabel('MAE')
25
26 # Plot RMSE
27 axes[2].bar(models, metrics['RMSE'], color=['lightyellow', 'olive', 'green'])
28 axes[2].set_title('RMSE Comparison')
29 axes[2].set_ylabel('RMSE')
30
31 # Adjust layout
32 plt.tight_layout()
33 plt.show()
34
35
```



```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 from matplotlib.table import Table
4
5 # Define metrics for ARIMA, LSTM, and SARIMA
6 metrics_data = {
7     'Model': ['ARIMA', 'SARIMA', 'LSTM'],
8     'MSE': [mse_arima, mse_sarima, mse_lstm],
9     'RMSE': [rmse_arima, rmse_sarima, rmse_lstm],
10    'MAE': [mae_arima, mae_sarima, mae_lstm],
11    'R-squared': [r2_arima, r2_sarima, r2_lstm]
12 }
13
14 # Create a DataFrame
15 metrics_df = pd.DataFrame(metrics_data)
16
17 # Plot the table
18 fig, ax = plt.subplots(figsize=(8, 4))
19 ax.axis('tight')
20 ax.axis('off')
21
22 # Add a table to the plot
23 table = Table(ax, bbox=[0, 0, 1, 1])
24
25 # Define cell properties
26 cell_props = dict(edgecolor='black')
27
28 # Header row
29 for col_idx, header in enumerate(metrics_df.columns):
30     table.add_cell(0, col_idx, width=1/len(metrics_df.columns), height=0.2,

```

```

31         text=header, loc='center', facecolor='lightgray', **cell_props)
32
33 # Add data rows
34 for row_idx, row in metrics_df.iterrows():
35     for col_idx, value in enumerate(row):
36         table.add_cell(row_idx+1, col_idx, width=1/len(metrics_df.columns), height=0.2,
37                        text=f'{value:.2e}' if isinstance(value, float) else value, loc='center',
38                        facecolor='white', **cell_props)
39
40 # Highlight best-performing metrics
41 for row_idx, row in metrics_df.iterrows():
42     for col_idx, value in enumerate(row[1:], start=1): # Skip the first column (Model)
43         if metrics_df.columns[col_idx] in ['MSE', 'MAE', 'RMSE'] and value == min(metrics_df[metrics_df.columns[col_idx]]):
44             table.get_cellid()[(row_idx+1, col_idx)].set_facecolor('lightyellow')
45         if metrics_df.columns[col_idx] == 'R-squared' and value == max(metrics_df['R-squared']):
46             table.get_cellid()[(row_idx+1, col_idx)].set_facecolor('lightyellow')
47
48 # Add the table to the axes
49 ax.add_table(table)
50
51 # Display the table
52 plt.title('Comparison of ARIMA, LSTM, and SARIMA Metrics', fontsize=14)
53 plt.show()
54

```



Comparison of ARIMA, LSTM, and SARIMA Metrics

Model	MSE	RMSE	MAE	R-squared
ARIMA	1.06e+08	1.03e+04	8.98e+03	-3.32e+00
SARIMA	1.37e+07	3.70e+03	2.18e+03	4.44e-01
LSTM	3.35e+07	5.78e+03	4.10e+03	-3.59e-01

Summary of Section 3

Model Development and Training:

- ARIMA and LSTM models compared; LSTM showed better handling of non-linear patterns.
- Additional model developed and evaluated: SARIMA.
- SARIMA and LSTM models compared; SARIMA outperformed LSTM.
- Key Insight:

Anomaly Detection:

- Isolation Forest and DBSCAN flagged environmental anomalies.