REPORT ON PREDICTING VEGETATION HEALTH IN MARSABIT, KENYA.

Image by bess.hamiti@gmail.com from Pixabay

1. Introduction

Marsabit is located at the northern part of Kenya, 170 kilometres east of the centre of East African Rift. With a population of 5000 people, It's situated above a desert and has most of the hills heavily forested.

The county of Marsabit is known as very small town with tropical savanna climate influenced by altitude and strong southeasterly wind called 'Turkana Jet'. The World Meteorological Organization reports an average monthly temperature of 25°C high and 15.7°C low. It's also report to have the highest rainfall of 149 mm in April with a total of 693mm in a year. The presence of altitude helps to mitigate heat, making the place cooler in the nights.

This challenge project is aimed at providing valuable information on the vegetation of Marsabit to help avert climate disasters like drought and boost the economical wellbeing of the place. The project is organised by Desight in partnership with the Shamba Network. Data was provided Shamba Network. The data is captured using the technology of eVIIRS-NDVI.

Findings on the NASA website states that the EROS Visible Infrared Imaging Radiometer Suite (eVIIRS) Normalized Difference Vegetation Index (NDVI) collection uses the Visible Infrared Imaging Radiometer Suite (VIIRS) collection that is available at NASA's Land Atmosphere Near-real-time Capability for EOS (LANCE) for the expedited product and Level-1 and Atmosphere Archive & Distribution System (LAADS) for the historical products.

2. Data Exploration

2.1 Data Analysis:

Three different categories of data was provided for the analysis. The first one was a static raster or Cloud Optimised GeoTIFF (COG), and the second one was a dynamic JSON data of last 30 days with 10 days lag. The last data was a geoJSON coordinates of the Marsabit County. After an initial challenge, I got the data downloaded (special thanks to @CryptoGeek) and proceeded to perform initial processing. Further assistance was provided by the Desight and OPF team too. The summary data provided important information on the NDVI which made it adequate for the modelling process.

Key features seen in the the various data was NDVI summary stats especially the NDVI_mean which is a measure of central tendency crucial for an analysis like this. Another feature was the date which is important in time-dependent analysis like time-series. Looking into the COGs files also revealed important metadata like the coordinates, boundaries, width, height, units, and band number. All these features were important for future analysis like resampling and merging/integration

2.1.1 Data access from directory

```
# Directory containing all GeoTIFF (COG) files
static_cog_ndvi_directory = “../data/eVIIRS-NDVI-historical-data/marsabit”

# asset containing the geoJSON coordinates
geojson_file = '../data/eVIIRS-NDVI-historical-data/coords.geojson'

dynamic_ndvi_file = ‘../data/eVIIRS-NDVI-historical-data/last-30-days.json’

# updated summary of the eVIIRS-NDVI Historical Data
updated_tabular_summary_file = '../data/extra-data/updated_tabular_summary.xlsx'
```

```
# Define a pattern to match GeoTIFF files (e.g., all files with .tif extension)
file_pattern = os.path.join(static_cog_ndvi_directory, "*.tif")

# Fetch all GeoTIFF files in the directory
all_static_ndvi_files = glob.glob(file_pattern)

# Loop through the entire COGs and retrieve the metadata
for file_path in all_static_ndvi_files:
    # Open the GeoTIFF file
    all_cog_dataset = rioxarray.open_rasterio(file_path, masked=True)
```

2.1.2 Analysing the Static COG for the eVIIRS-NDVI Historical Data

The static COG data had a shape of (1, 882, 912) translated as Band 1, Height of 882, and Width of 912. This was determined by running the following python code:

```
all_cog_dataset:<xarray.DataArray (band: 1, y: 882, x: 912)>
[804384 values with dtype=uint8]
Coordinates:
  * band         (band) int64 1
  * x            (x) float64 36.05 36.05 36.06 36.06 ... 39.33 39.34 39.34 39.35
  * y            (y) float64 4.453 4.45 4.446 4.443 ... 1.276 1.272 1.269 1.265
    spatial_ref  int64 0
Attributes:
    AREA_OR_POINT:  Area
    _FillValue:     0
    scale_factor:   1.0
    add_offset:     0.0
```

To further confirm other important metadata like the units of the geometry, we can use the following code:

```
# EPSG codes are great for succinctly representing a particular coordinate reference
system.

#  let's retrieve other details about the CRS, eg. the unit:
epsg = all_cog_dataset_crs.to_epsg()

all_cog_dataset_epsg_codes = CRS(epsg)
all_cog_dataset_epsg_codes
```

```
Output:
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
```

The Geographic coordinates are typically measured in degrees and the Axis Info outputted by the CRS code above, indicates that too. The bounds, width and height are also computed using the **rioxarray.rio.bounds()**, **rioxarray.rio.width**, and **rioxarray.rio.height** respectively.

```
all_cog_dataset_height = all_cog_dataset.rio.height

all_cog_dataset_height
```

```
Output: 882
```

```
# Print and view the the values of the COGs files


all_cog_dataset.values
```

Output:

```
           array([[[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, 95., 97., ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]]], dtype=float32)
```

2.1.3 Analysing the Summary Table for eVIIR-NDVI Historical Data

The summary table comprised of statistics of the NDVI values. Using the shapely library I was able to confirm the number of rows and columns in the summary table. I then printed a subset of the entire data to view it as shown below.

```
# Load the summary data into a Pandas DataFrame
updated_summary_df = pd.read_excel(updated_tabular_summary_file)

# Display the first few rows of the summary DataFrame to get an overview
print(updated_summary_df.head())

all_cog_dataset_epsg_codes
```

```
Output:

           Date  NDVI_max  NDVI_mean  NDVI_median  NDVI_min  NDVI_stdDev  \

0  Jan 30, 2012       0.9      0.252         0.25     -0.08        0.125
1   Feb 4, 2012       0.9      0.251         0.25     -0.08        0.125
2   Feb 9, 2012       0.9      0.247         0.25     -0.08        0.122
3  Feb 14, 2012       0.9      0.242         0.24     -0.08        0.119
4  Feb 19, 2012       0.9      0.237         0.24     -0.08        0.115

   NDVI_variance
0          0.016
1          0.016
2          0.015
3          0.014
4          0.013
```

```
updated_summary_df.shape
```

```
Output: (825, 7)
```

2.2 Identifying Missing Values and Potential Outliers:

Raster data often have missing data associated with it and keeping that in mind, I took a number of initiative to deal with it. One strategy was to use NaN to represent possible missing numbers by specifying `masked = True` when loading the raster. However I should point out that while this helps temporarily, I will have to watch out for the side effect of having the `DataArray` type changed from integers to float.

2.2.1 Missing Values and Potential Outliers in the static COGs

The code below indicated a lot of missing values in the raster dataset.

```
print('missing-values-all_cog:', all_cog_dataset.isnull().sum().values)
```

```
missing-values-all_cog: 333676
```

2.2.2 Missing Values and Potential Outliers in the Summary Table

No missing values was noted in the summary table as expected

```
print('missing-values-summary_table:', updated_summary_df.isnull().sum().values)
```

```
missing-values-summary_table: [0 0 0 0 0 0
```

2.3 Statistical Summaries: Rasters Statistical summaries where computed in the form of mean, std, percentiles and max.

2.2.3 Stats for Summary table

```
# Descriptive Statistics:
# Calculate basic statistics for the NDVI data, such as mean, standard deviation, and
more.
ndvi_stats = updated_summary_df['NDVI_mean'].describe()
print(ndvi_stats)
```

```
count    825.000000
mean       0.290210
std        0.084692
min       -0.087000
25%        0.231000
50%        0.264000
75%        0.338000
max        0.605000
Name: NDVI_mean, dtype:
float64
```

2.2.3 Stats for COGs

```
all_cog_dataset_min = all_cog_dataset.min().values

print(all_cog_dataset_min)
```

```
Output: 92.0
```

```
all_cog_dataset_max = all_cog_dataset.max().values

print(all_cog_dataset_max)
```

```
Output: 190.0
```

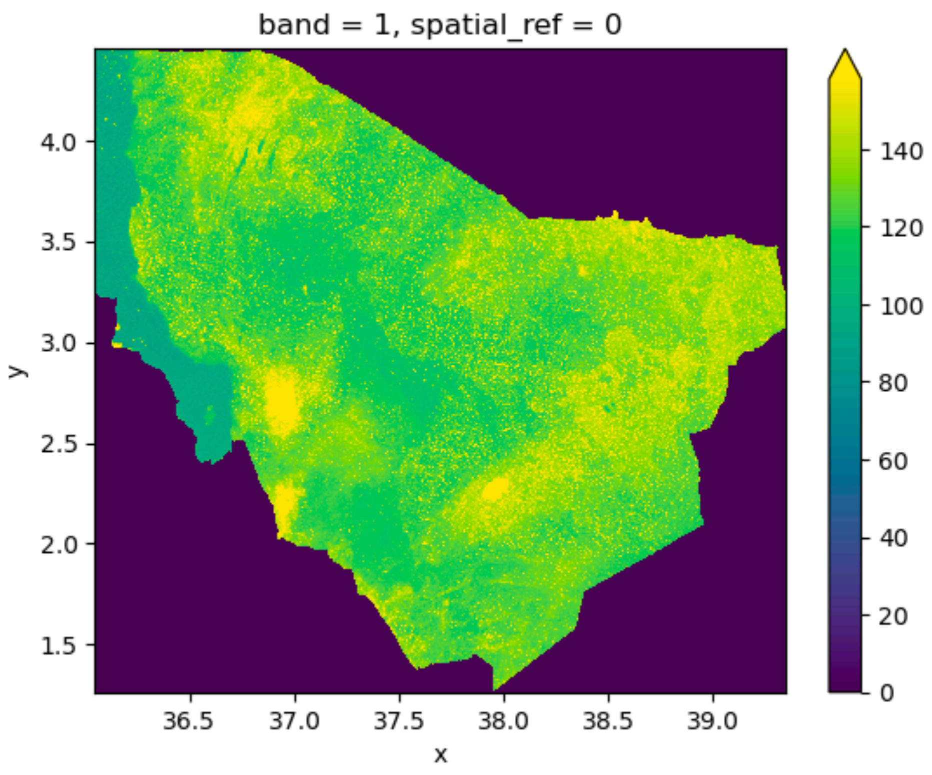
```
all_cog_dataset_std = all_cog_dataset.std().values

print(all_cog_dataset_std)
```

```
Output: 63.93047844262327
```

2.4 Data Visualisation: Data visualisation was provided for both COGs and summary files in the form various plots as explained below.

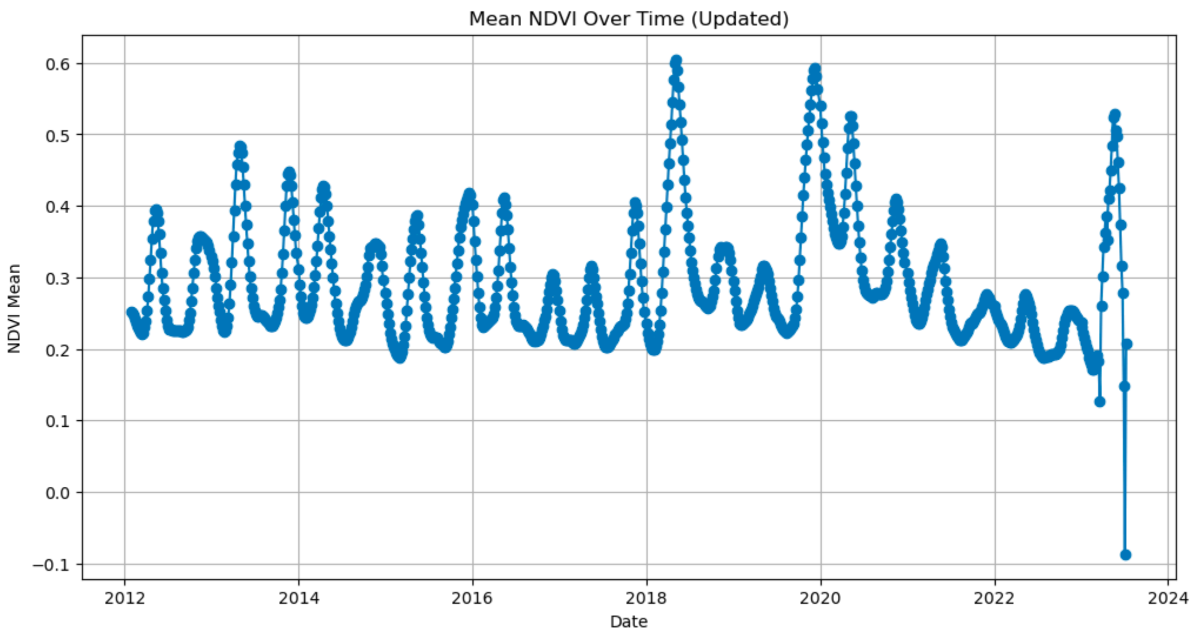
2.4.1 Data Visualisation for the COGs Files

```
all_cog_dataset.plot(robust=True)
```

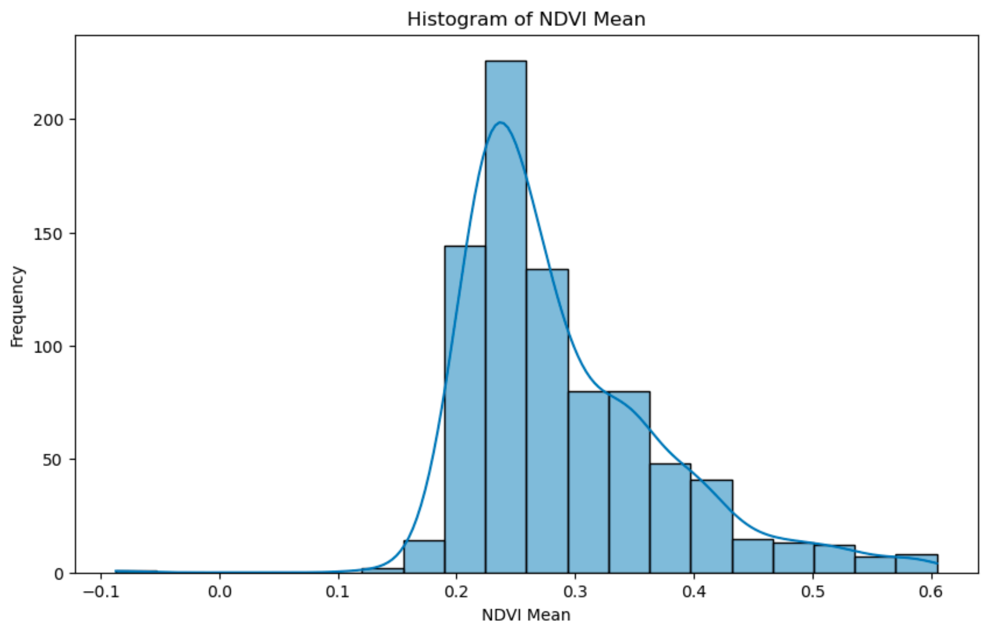


2.4.2 Data Visualisation for the Summary Table Stats

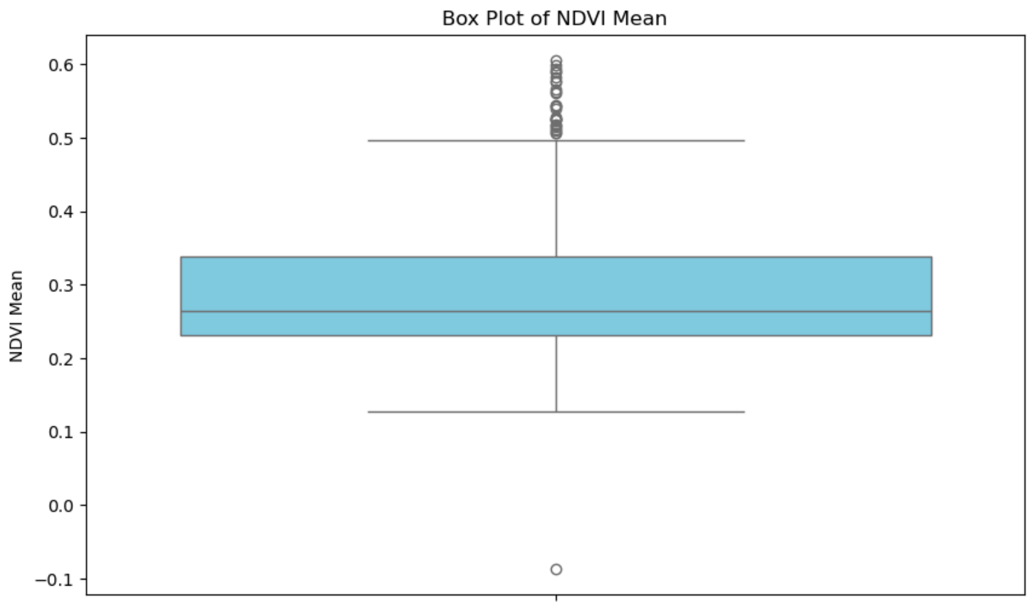
```
# i. Time Series Visualizations Showing Patterns and Trends

updated_summary_df['Date'] = pd.to_datetime(updated_summary_df['Date'])

# Plotting the mean NDVI values over time
plt.figure(figsize=(12, 6))
plt.plot(updated_summary_df['Date'], updated_summary_df['NDVI_mean'], marker='o',
linestyle='-')
plt.title('Mean NDVI Over Time (Updated)')
plt.xlabel('Date')
plt.ylabel('NDVI Mean')
plt.grid(True)
```



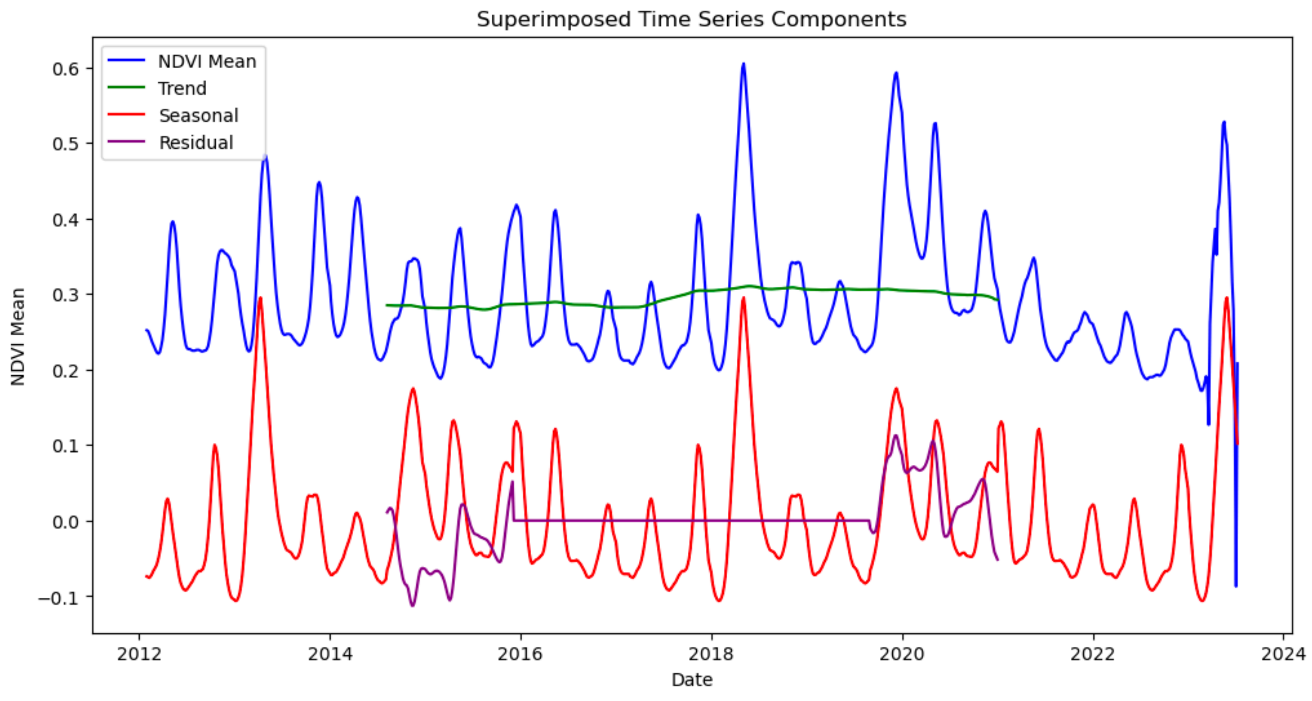
```
#  Histogram of NDVI Mean values
plt.figure(figsize=(10, 6))
sns.histplot(data=updated_summary_df, x='NDVI_mean', bins=20, kde=True)
plt.title('Histogram of NDVI Mean')
plt.xlabel('NDVI Mean')
plt.ylabel('Frequency')
plt.show()
```

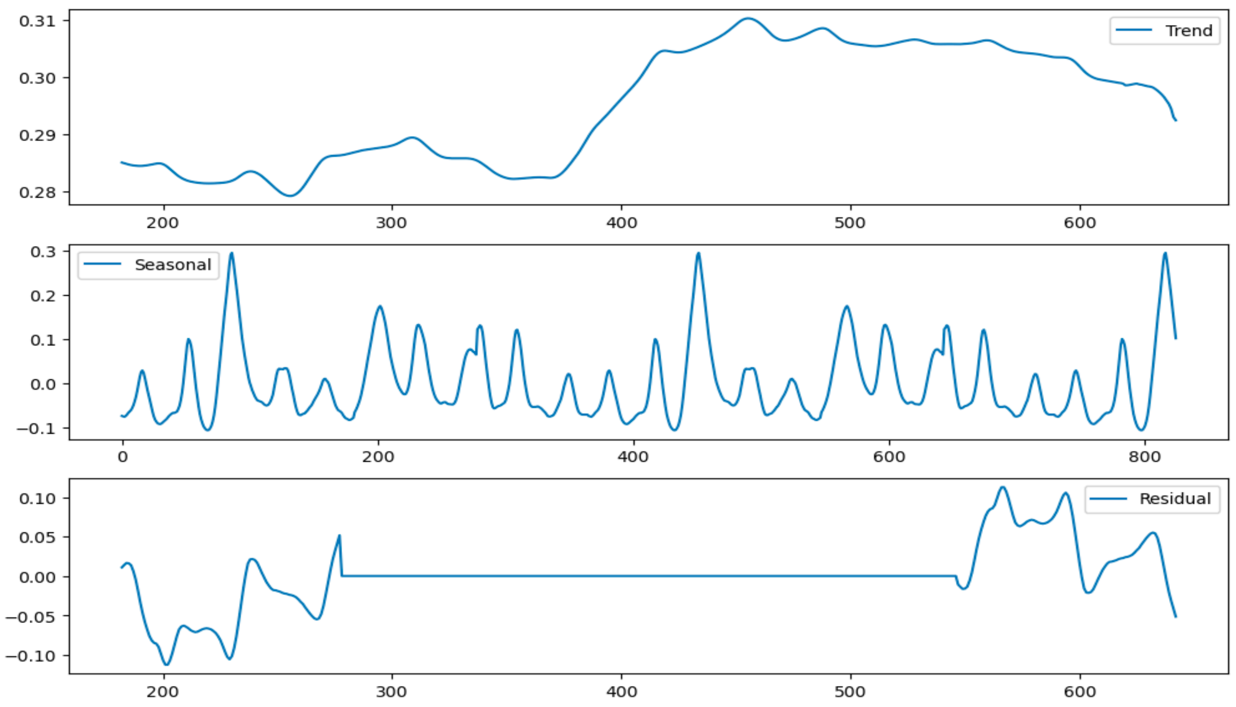


```
# Box plot of NDVI Mean values
plt.figure(figsize=(10, 6))
sns.boxplot(data=updated_summary_df, y='NDVI_mean', color='skyblue')
plt.title('Box Plot of NDVI Mean')
plt.ylabel('NDVI Mean')
plt.show()
```



```
# Decompose the time series
decomposition = seasonal_decompose(updated_summary_df['NDVI_mean'], model='additive',
period=365)

# Access the decomposed components
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

# Create a figure and axis for the superimposed plot
fig, ax = plt.subplots(figsize=(12, 6))

# Plot the NDVI mean values
ax.plot(updated_summary_df['Date'], updated_summary_df['NDVI_mean'], label='NDVI Mean',
color='blue')

# Plot the trend component
ax.plot(updated_summary_df['Date'], trend, label='Trend', color='green')

# Plot the seasonal component
ax.plot(updated_summary_df['Date'], seasonal, label='Seasonal', color='red')

# Plot the residual component
ax.plot(updated_summary_df['Date'], residual, label='Residual', color='purple')

ax.set_xlabel('Date')
ax.set_ylabel('NDVI Mean')
ax.set_title('Superimposed Time Series Components')
ax.legend()

plt.show()
```

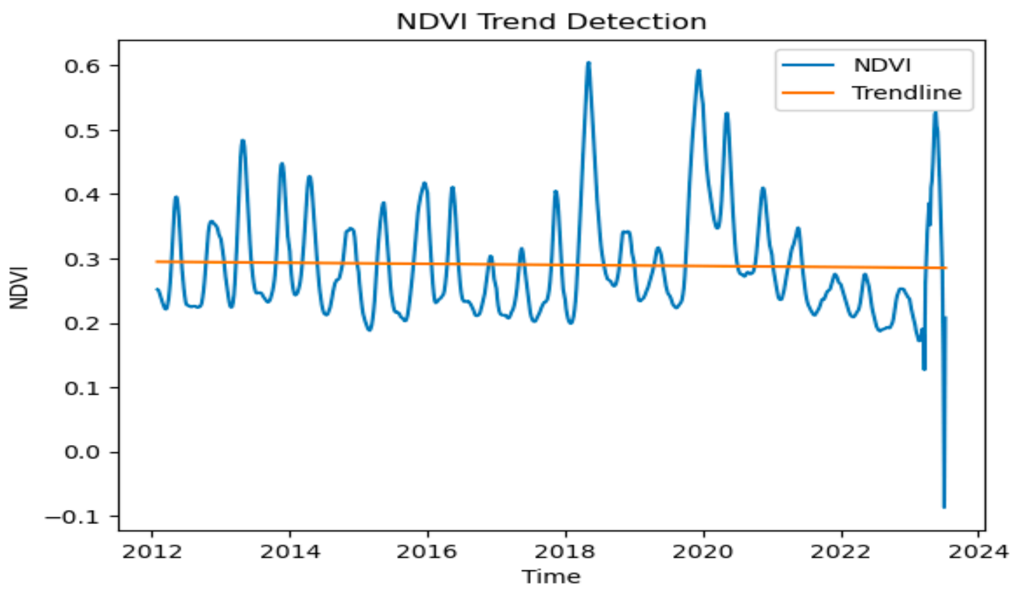




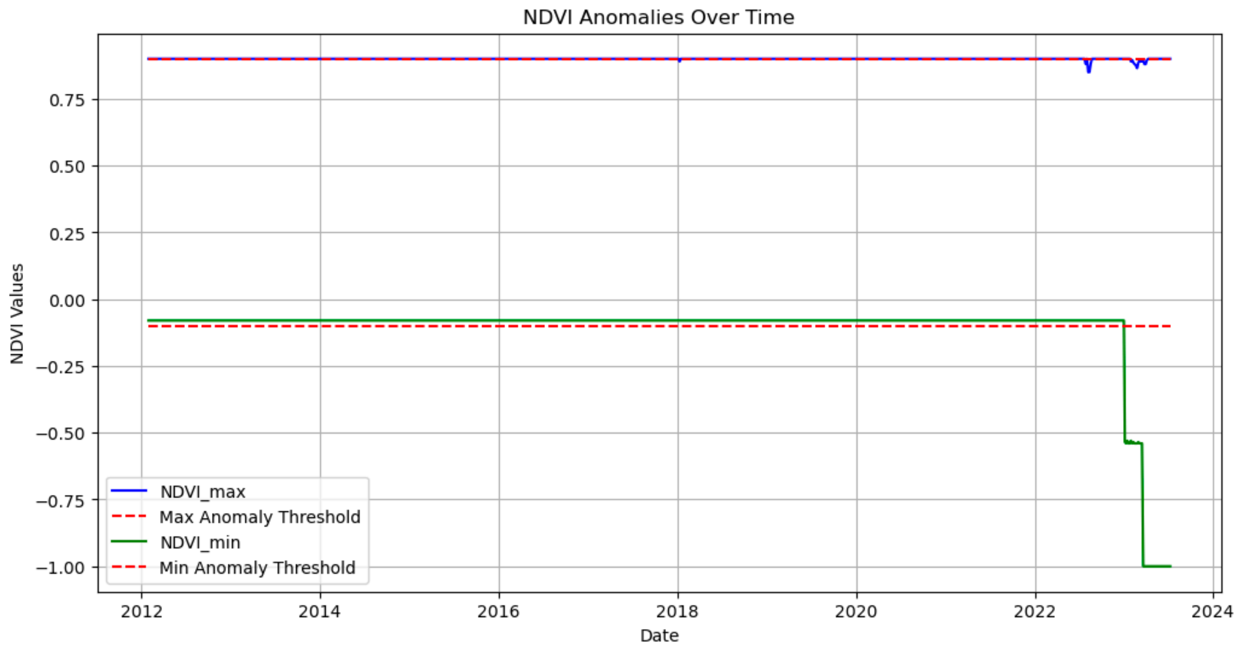
```
#  Trend Detection

# Perform linear regression
x = range(len(updated_summary_df))  # Assuming your data spans multiple years
slope, intercept, r_value, p_value, std_err = stats.linregress(x,
updated_summary_df['NDVI_mean'])

# Analyze the results
if p_value < 0.05:
    # The trend is significant
    if slope > 0:
        print("Positive NDVI trend")
    else:
        print("Negative NDVI trend")
else:
    print("No significant trend")
```

```
Output:  No significant trend
```

```
dates, ndvi_values = updated_summary_df['Date'],
updated_summary_df['NDVI_mean']

plt.plot(dates, ndvi_values, label='NDVI')
plt.plot(dates, slope * np.arange(len(dates)) + intercept, label='Trendline')
plt.xlabel('Time')
```



```
 NDVI Anomalies Visualization

# Define thresholds for anomalies
max_threshold = 0.9  # Define a threshold for high NDVI (anomalously high)
min_threshold = -0.1  # Define a threshold for low NDVI (anomalously low)

# Create a figure and axis
fig, ax = plt.subplots(figsize=(12, 6))

# Plot the 'NDVI_max' values
updated_summary_df['Date'] = pd.to_datetime(updated_summary_df['Date'])  #
Ensure the 'Date' column is in datetime format
ax.plot(updated_summary_df['Date'], updated_summary_df['NDVI_max'],
label='NDVI_max', color='b')
ax.plot(updated_summary_df['Date'], [max_threshold] *
len(updated_summary_df), '--r', label='Max Anomaly Threshold')

# Plot the 'NDVI_min' values
ax.plot(updated_summary_df['Date'], updated_summary_df['NDVI_min'],
label='NDVI_min', color='g')
ax.plot(updated_summary_df['Date'], [min_threshold] *
len(updated_summary_df), '--r', label='Min Anomaly Threshold')

# Set plot title and labels
ax.set_title('NDVI Anomalies Over Time')
ax.set_xlabel('Date')
ax.set_ylabel('NDVI Values')

# Show legend
ax.legend()

# Show the plot
plt.grid(True)
plt.show()
```



Standardised anomalies are useful in monitoring climate change. Standardised anomalies subtract the long-term mean from an observation of interest and then divide the result by the long-term standard deviation, thus removing seasonal variability and highlighting change related to longer-term drivers.

We'll calculate monthly *standardised* NDVI anomalies for any given month and year in this analysis.

The tabulated summary data which has pre-processed NDVI statics is used to calculate the monthly mean NDVI for the month of interest. The standardised anaomaly is then calculated as:

$$\text{Standardised anomaly } = \frac{x - m}{s}$$

where x is NDVI for the month of interest, m is the long-term mean, and s is the long-term standard deviation

```
# Calculate baseline (e.g., multi-year average)
baseline = updated_summary_df['NDVI_mean'].mean()

# Calculate anomalies
updated_summary_df['Anomaly'] = updated_summary_df['NDVI_mean'] - baseline

# Calculate z-scores
mean_anomaly = updated_summary_df['Anomaly'].mean()
std_deviation_anomaly = updated_summary_df['Anomaly'].std()
updated_summary_df['Z-Score'] = (updated_summary_df['Anomaly'] -
mean_anomaly) / std_deviation_anomaly

# visualize or analyze the 'Z-Score' column
updated_summary_df.sample(10)
```

	Date	NDVI_max	NDVI_mean	NDVI_median	NDVI_min	NDVI_stdDev	NDVI_variance	Anomaly	Z-Score
821	2023-06-24	0.9	0.278	0.25	-1.00	0.182	0.033	-0.01221	-0.144165
221	2015-02-24	0.9	0.189	0.19	-0.08	0.102	0.010	-0.10121	-1.195027
558	2019-10-27	0.9	0.440	0.43	-0.08	0.236	0.056	0.14979	1.768639
330	2016-08-27	0.9	0.223	0.21	-0.08	0.123	0.015	-0.06721	-0.793574
353	2016-12-20	0.9	0.268	0.23	-0.08	0.170	0.029	-0.02221	-0.262240
591	2020-04-14	0.9	0.416	0.41	-0.08	0.206	0.042	0.12579	1.485261
497	2018-12-21	0.9	0.309	0.28	-0.08	0.172	0.030	0.01879	0.221865
201	2014-11-11	0.9	0.344	0.31	-0.08	0.199	0.040	0.05379	0.635126

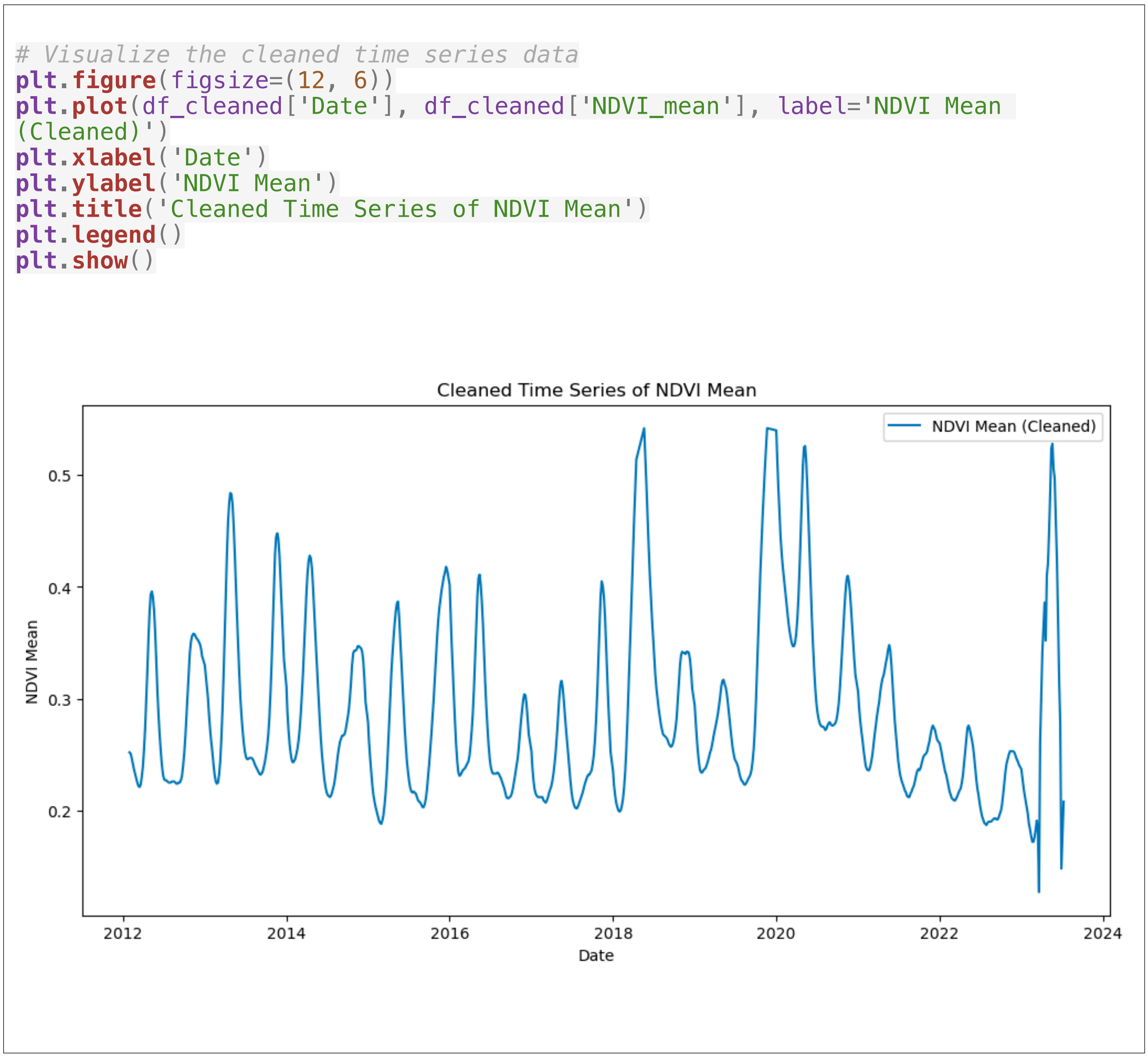
3. Data Improvement & Feature Engineering

3.1 Transformation & Imputation on Summary Table: As shown on the various plots there were some anomalies and missing values that required handling before the modelling. To ensure that further manipulations will not affect the state of the previous table, I created a copy of the dataframe and performed inplace forward fill on the mean column which is the target for the model.

```
updated_summary_df_copy = updated_summary_df.copy()

# Handle missing values on the entire data series (example: forward fill)
updated_summary_df_copy.ffill(inplace=True)


# Detect and handle outliers (example: Z-score)
z_scores = zscore(updated_summary_df_copy['NDVI_mean'])
outlier_threshold = 3  # You can adjust this threshold
outliers = (z_scores > outlier_threshold) | (z_scores < -outlier_threshold

# Remove rows with outliers
df_cleaned = updated_summary_df_copy[~outliers]

df_cleaned.sample(10)
```

	Date	NDVI_max	NDVI_mean	NDVI_median	NDVI_min	NDVI_stdDev	NDVI_variance	Anomaly	Z-Score
675	2021-06-14	0.9	0.280	0.27	-0.08	0.149	0.022	-0.01021	-0.120550
192	2014-09-27	0.9	0.277	0.26	-0.08	0.150	0.022	-0.01321	-0.155973
762	2022-08-28	0.9	0.192	0.19	-0.08	0.101	0.010	-0.09821	-1.159605
413	2017-10-22	0.9	0.304	0.28	-0.08	0.167	0.028	0.01379	0.162828
754	2022-07-19	0.9	0.189	0.19	-0.08	0.101	0.010	-0.10121	-1.195027
83	2013-03-26	0.9	0.321	0.30	-0.08	0.175	0.031	0.03079	0.363555
266	2015-10-07	0.9	0.256	0.22	-0.08	0.157	0.025	-0.03421	-0.403929
620	2020-09-06	0.9	0.276	0.26	-0.08	0.144	0.021	-0.01421	-0.167780

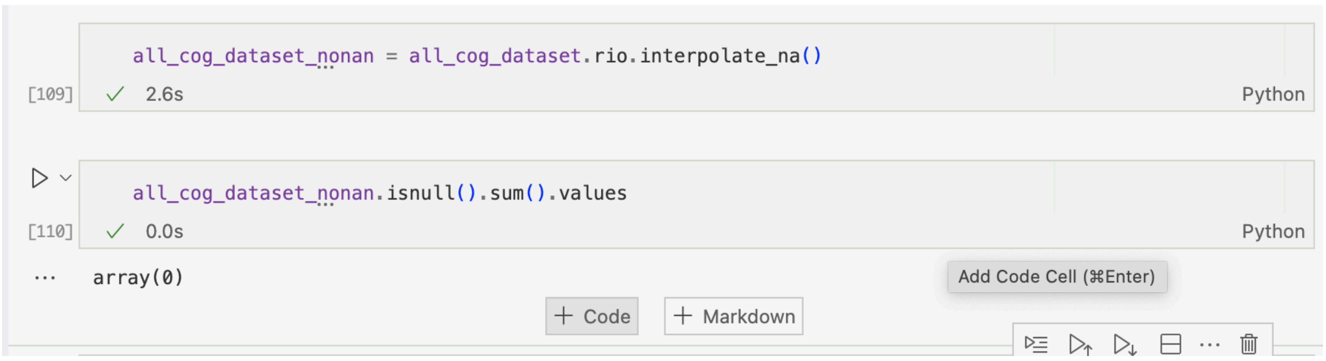
```
#  Linear interpolation for missing values
updated_summary_df['NDVI_mean'].interpolate(method='linear', inplace=True)
```



```python
# Visualize the cleaned time series data
plt.figure(figsize=(12, 6))
plt.plot(df_cleaned['Date'], df_cleaned['NDVI_mean'], label='NDVI Mean 
(Cleaned)')
plt.xlabel('Date')
plt.ylabel('NDVI Mean')
plt.title('Cleaned Time Series of NDVI Mean')
plt.legend()
plt.show()
```

3.2 Transformation & Imputation on static COGs:

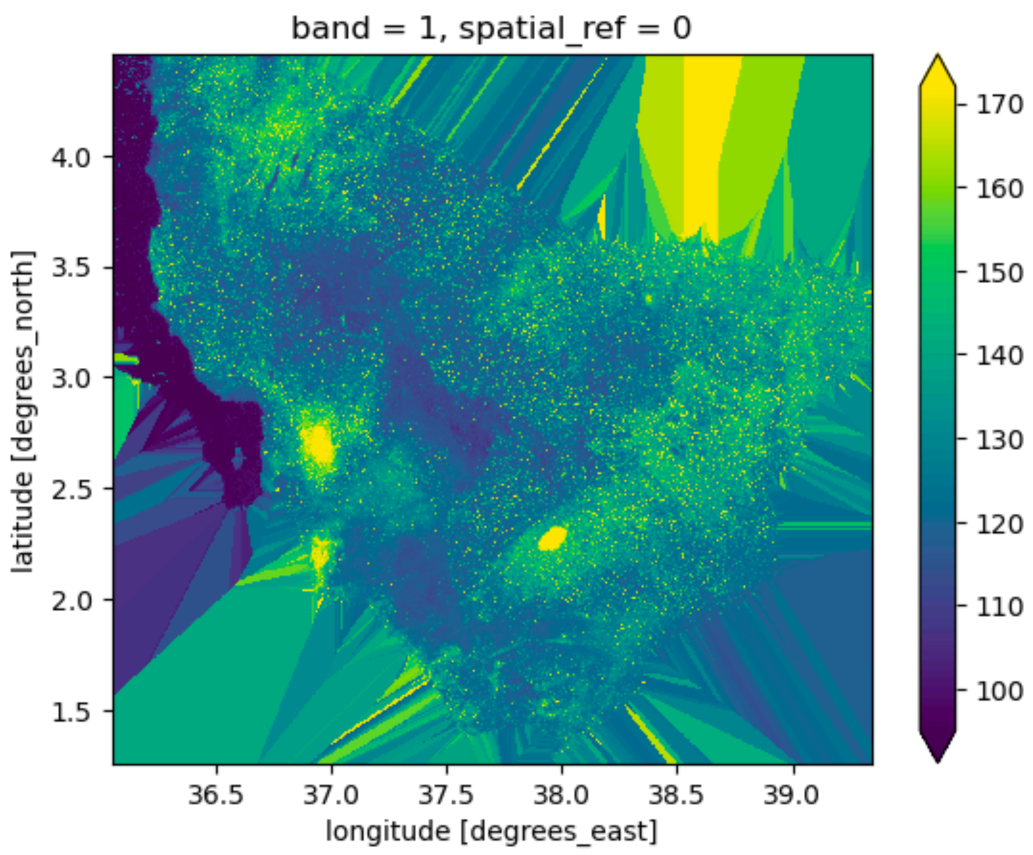
The raster files where interpolated to finally handle missing values. The plot later showed a better view with the all noise at the edge removed. This step will then prepare it for future analysis. In the context of this challenge more emphasis was laid on the summary tables that already contained NDVI stats and proved more ready for modelling. However, the as shown further steps, the static COGs could still be resampled and merged with additional data that add metadata like nir, red, and green bands to enable adequate calculations appropriate for vegetation or climate modelling.



```
all_cog_dataset_nonan = all_cog_dataset.rio.interpolate_na()
[109] ✓ 2.6s                                                    Python

all_cog_dataset_nonan.isnull().sum().values
[110] ✓ 0.0s                                                    Python

array(0)
```

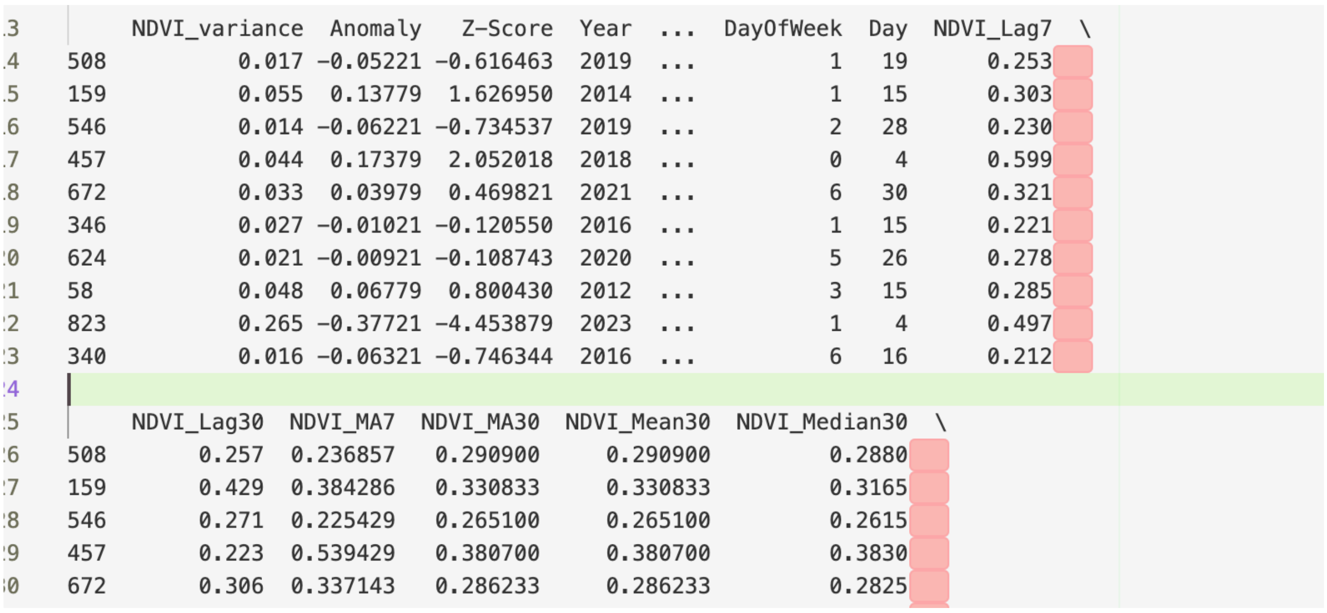
3.3 Feature Creation: A couple of features were created including seasonal components and rolling averages to further boost the dataset for modelling.

```
all_cog_dataset_nonan.plot(robust=True)
```

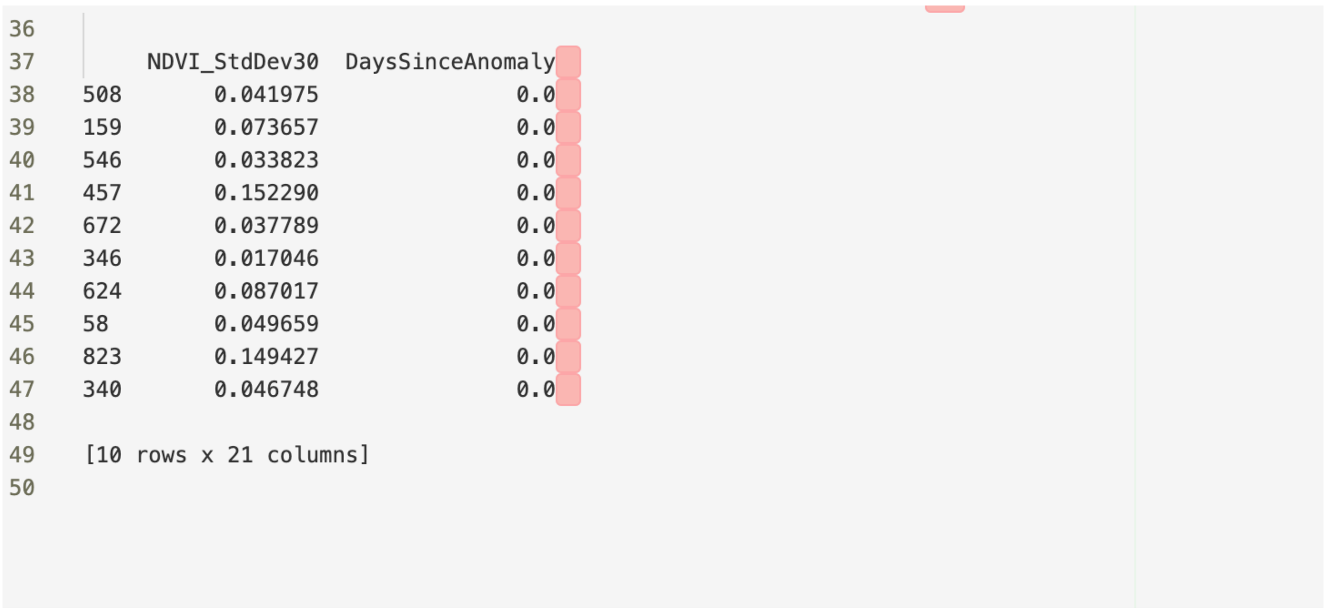


```python
# Extract temporal features
updated_summary_df_copy['Year'] = updated_summary_df_copy['Date'].dt.year
updated_summary_df_copy['Month'] = updated_summary_df_copy['Date'].dt.month
updated_summary_df_copy['DayOfWeek'] =
updated_summary_df_copy['Date'].dt.dayofweek  # Monday: 0, Sunday: 6
updated_summary_df_copy['Day'] = updated_summary_df_copy['Date'].dt.day
# Calculate lagged features (e.g., 7-day and 30-day lag)
updated_summary_df_copy['NDVI_Lag7'] =
updated_summary_df_copy['NDVI_mean'].shift(7)
updated_summary_df_copy['NDVI_Lag30'] =
updated_summary_df_copy['NDVI_mean'].shift(30)

# Calculate moving averages (e.g., 7-day and 30-day moving averages)
updated_summary_df_copy['NDVI_MA7'] =
updated_summary_df_copy['NDVI_mean'].rolling(window=7).mean()
updated_summary_df_copy['NDVI_MA30'] =
updated_summary_df_copy['NDVI_mean'].rolling(window=30).mean()
# Statistical aggregations
updated_summary_df_copy['NDVI_Mean30'] =
updated_summary_df_copy['NDVI_mean'].rolling(window=30).mean()
updated_summary_df_copy['NDVI_Median30'] =
updated_summary_df_copy['NDVI_mean'].rolling(window=30).median()
updated_summary_df_copy['NDVI_StdDev30'] =
updated_summary_df_copy[‘NDVI_mean'].rolling(window=30).std()

# Time since last anomaly (assuming anomalies are values below a certain 
threshold)
threshold = -0.1
updated_summary_df_copy['DaysSinceAnomaly'] =
updated_summary_df_copy['Date'].diff().dt.days
updated_summary_df_copy.loc[updated_summary_df_copy['NDVI_mean'] > threshold,
'DaysSinceAnomaly'] = 0


# Display the updated DataFrame
print(updated_summary_df_copy.sample(10))
```

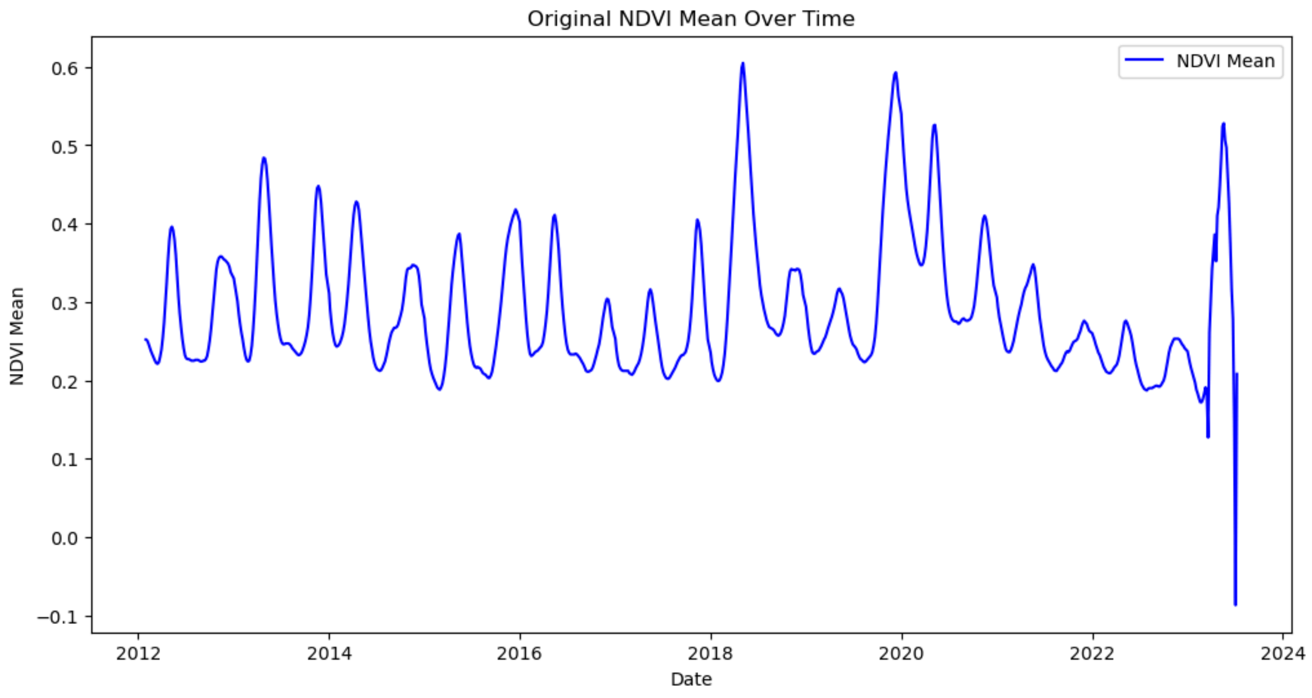
```
          Date  NDVI_max  NDVI_mean  NDVI_median  NDVI_min  NDVI_stdDev  \
508 2019-02-19       0.9      0.238         0.22     -0.08        0.131   
159 2014-04-15       0.9      0.428         0.42     -0.08        0.234   
546 2019-08-28       0.9      0.228         0.22     -0.08        0.120   
457 2018-06-04       0.9      0.464         0.48     -0.08        0.210   
672 2021-05-30       0.9      0.330         0.31     -0.08        0.183   
346 2016-11-15       0.9      0.280         0.25     -0.08        0.165   
624 2020-09-26       0.9      0.281         0.27     -0.08        0.145   
58  2012-11-15       0.9      0.358         0.31     -0.08        0.220   
823 2023-07-04       0.9     -0.087         0.15     -1.00        0.515   
340 2016-10-16       0.9      0.227         0.21     -0.08        0.126   

     NDVI_variance  Anomaly   Z-Score  Year  ...  DayOfWeek  Day  NDVI_Lag7  \
508          0.017 -0.05221 -0.616463  2019  ...          1   19      0.253   
159          0.055  0.13779  1.626950  2014  ...          1   15      0.303   
546          0.014 -0.06221 -0.734537  2019  ...          2   28      0.230   
```

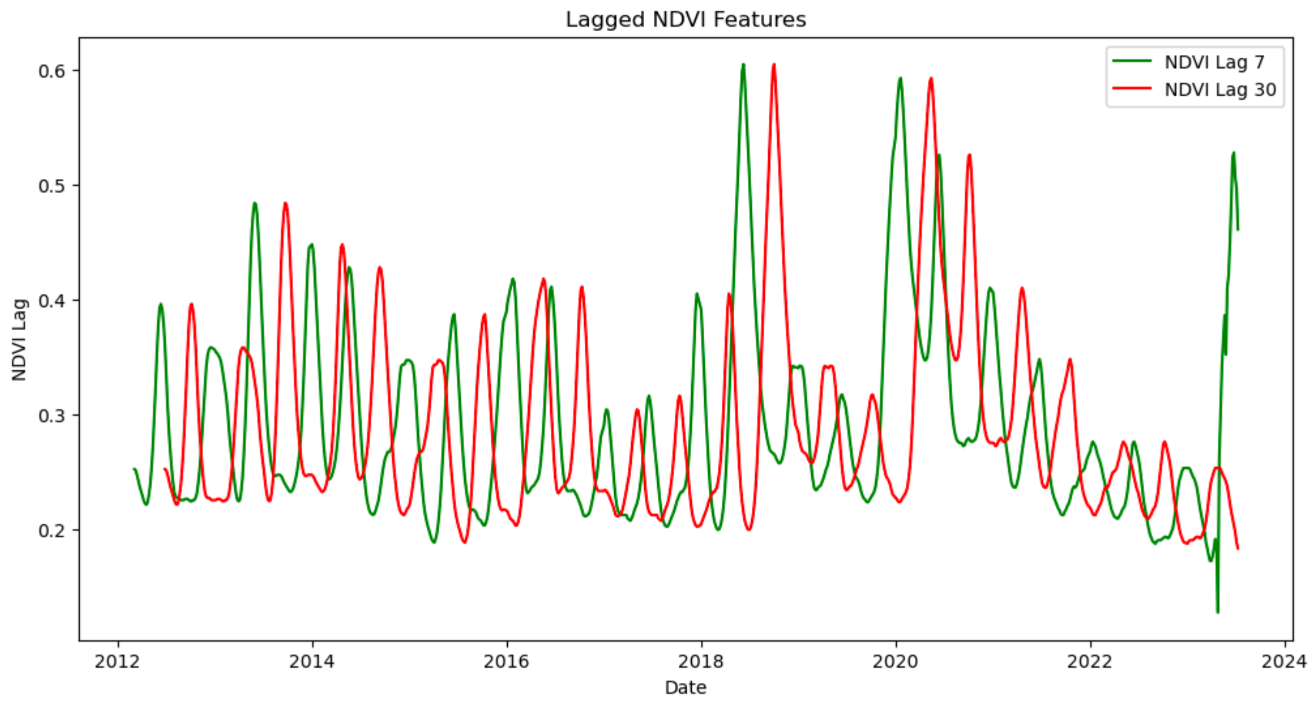


```
     NDVI_variance  Anomaly   Z-Score  Year  ...  DayOfWeek  Day  NDVI_Lag7  \
508          0.017 -0.05221 -0.616463  2019  ...          1   19      0.253
159          0.055  0.13779  1.626950  2014  ...          1   15      0.303
546          0.014 -0.06221 -0.734537  2019  ...          2   28      0.230
457          0.044  0.17379  2.052018  2018  ...          0    4      0.599
672          0.033  0.03979  0.469821  2021  ...          6   30      0.321
346          0.027 -0.01021 -0.120550  2016  ...          1   15      0.221
624          0.021 -0.00921 -0.108743  2020  ...          5   26      0.278
58           0.048  0.06779  0.800430  2012  ...          3   15      0.285
823          0.265 -0.37721 -4.453879  2023  ...          1    4      0.497
340          0.016 -0.06321 -0.746344  2016  ...          6   16      0.212

     NDVI_Lag30  NDVI_MA7  NDVI_MA30  NDVI_Mean30  NDVI_Median30  \
508       0.257  0.236857   0.290900     0.290900         0.2880
159       0.429  0.384286   0.330833     0.330833         0.3165
546       0.271  0.225429   0.265100     0.265100         0.2615
457       0.223  0.539429   0.380700     0.380700         0.3830
672       0.306  0.337143   0.286233     0.286233         0.2825
```

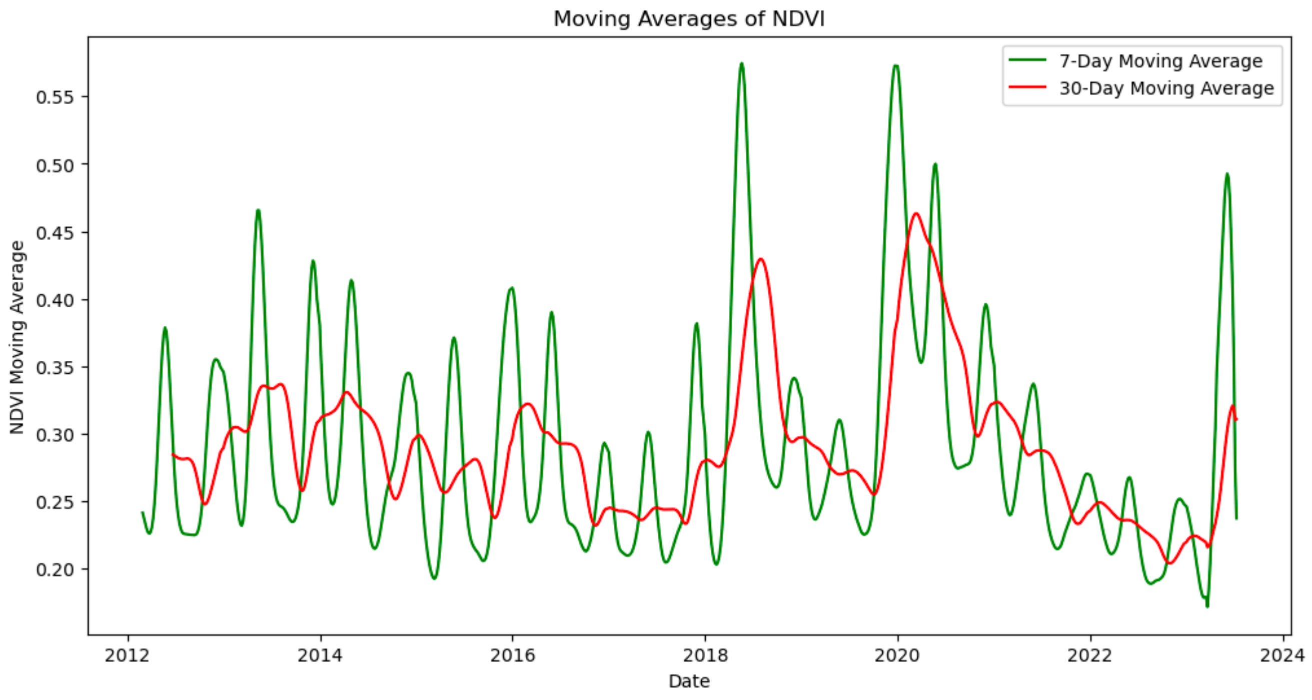


```
     NDVI_StdDev30  DaysSinceAnomaly
508       0.041975               0.0
159       0.073657               0.0
546       0.033823               0.0
457       0.152290               0.0
672       0.037789               0.0
346       0.017046               0.0
624       0.087017               0.0
58        0.049659               0.0
823       0.149427               0.0
340       0.046748               0.0

[10 rows x 21 columns]
```

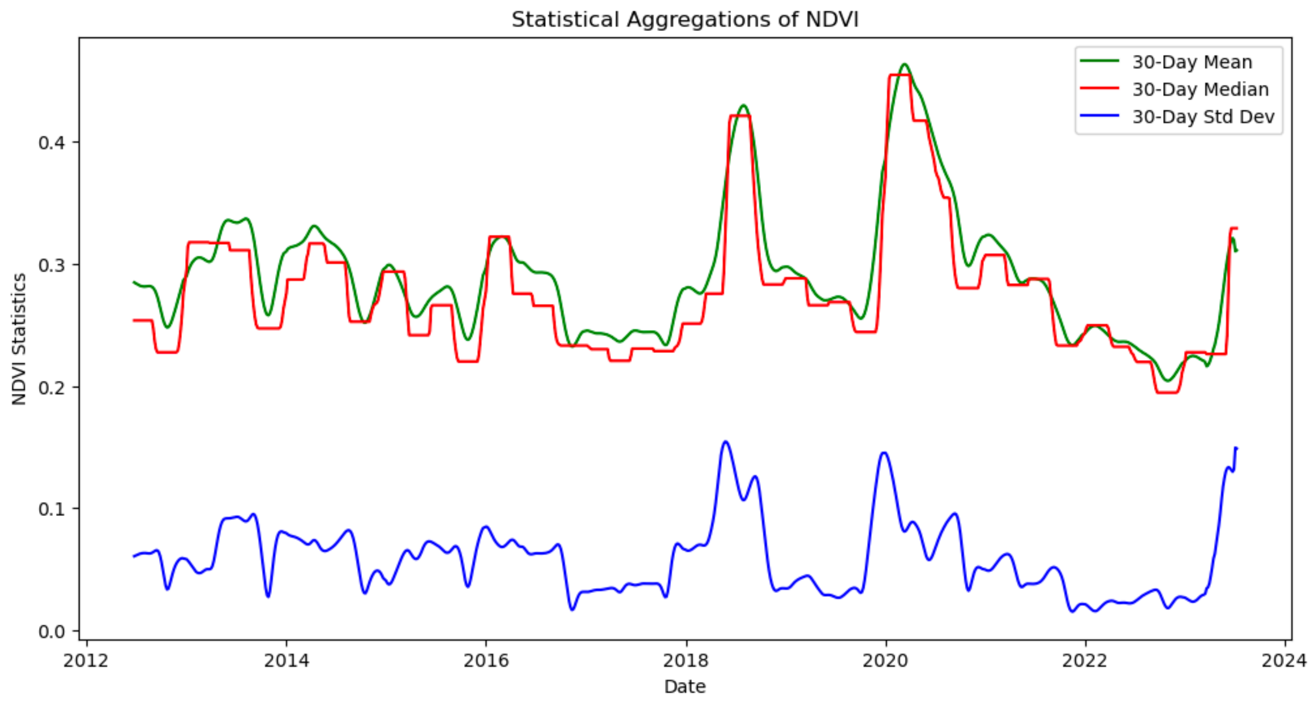




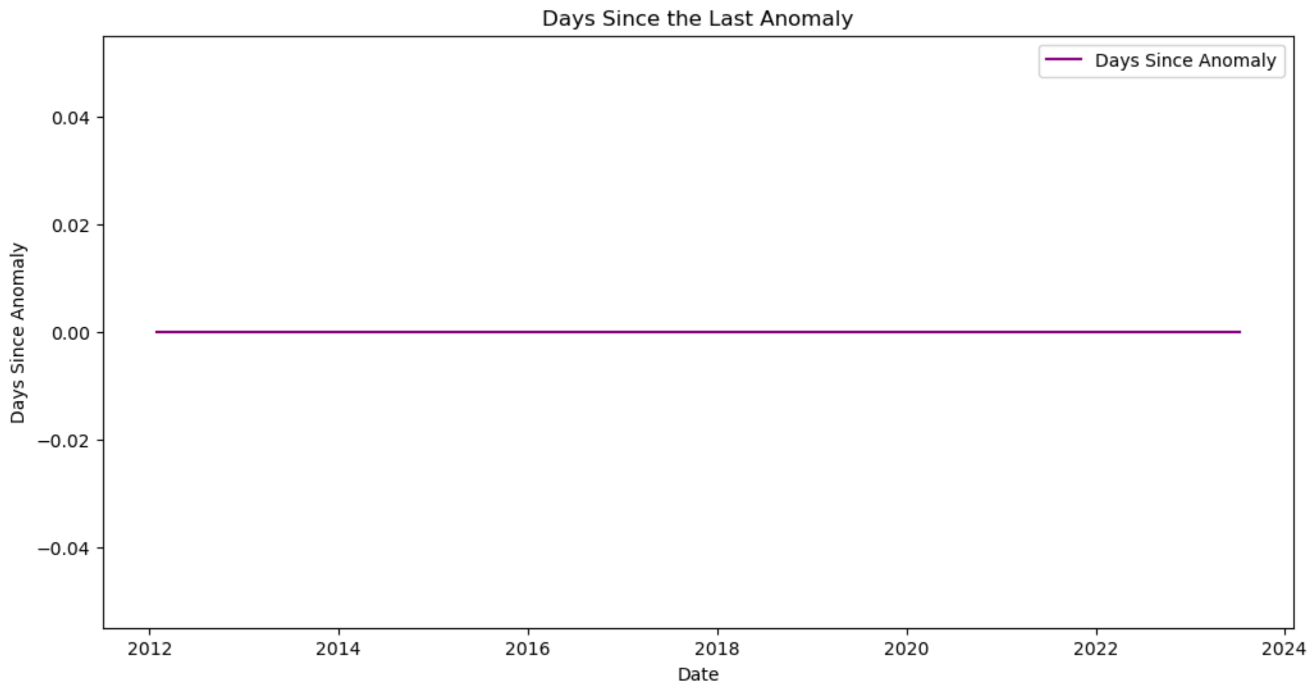
Lagged NDVI Features
NDVI Lag 7
NDVI Lag 30
NDVI Lag
Date
0.6
0.5
0.4
0.3
0.2
2012
2014
2016
2018
2020
2022
2024



Moving Averages of NDVI
7-Day Moving Average
30-Day Moving Average
NDVI Moving Average
Date
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
2012
2014
2016
2018
2020
2022
2024



Statistical Aggregations of NDVI
30-Day Mean
30-Day Median
30-Day Std Dev
NDVI Statistics
0.0
0.1
0.2
0.3
0.4
2012
2014
2016
2018
2020
2022
2024
Date



Days Since the Last Anomaly
Days Since Anomaly
Days Since Anomaly
0.04
0.02
0.00
−0.02
−0.04
2012
2014
2016
2018
2020
2022
2024
Date

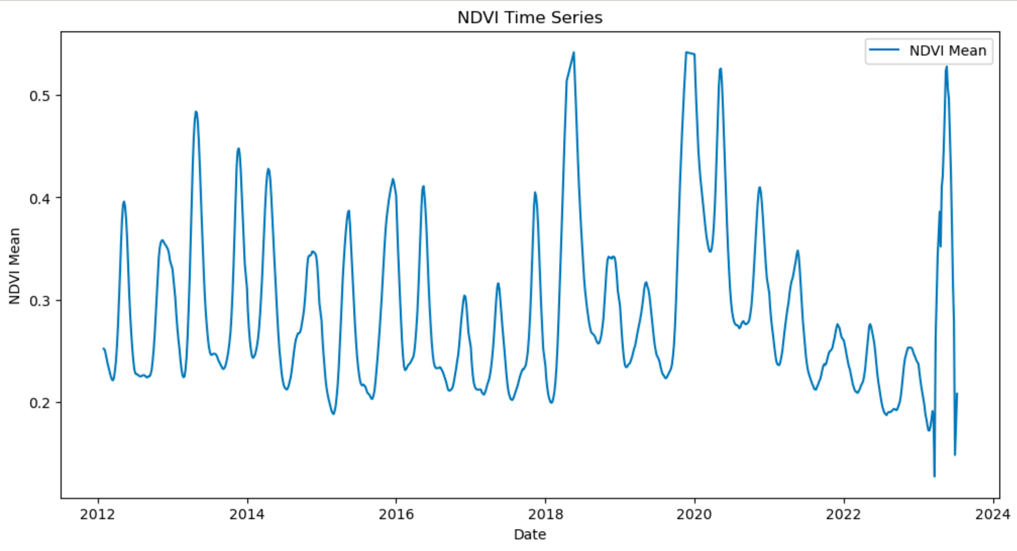
4. Model Building

Model Selection: I Started first with the ARIMA model, and noticed the the algorithm smartly detected seasonality in the dataset and consequently performed computations with it. Overall the model could detect seasonal variations and patterns with NDVI corresponding to high and low vegetation densities. The ARIMA/SARIMAX model also found a good fit of the dataset as shown by the various tests. The model summary suggests that the model provides a good fit to your data, and the low p-values for the AR and MA components indicate that they are statistically significant. However, potential issues like heteroskedasticity and a review the model's assumptions are necessary

Additional model will need to be created after integrating the water vapour data that I retrieved to better understand the variability of the vegetation with water vapour.

```
# let's start with a simple model, the Autoregressive Integrated Moving Average 
(ARIMA) model,
# which is a classic choice for time series forecasting

# let's confirm the data is good enough and ready for this stage

# Display the first few rows of the data to ensure it's loaded correctly
# updated_summary_df.set_index('NDVI_mean', inplace=True)

print(df_cleaned.head())
```

```
         Date  NDVI_max  NDVI_mean  NDVI_median  NDVI_min  NDVI_stdDev  \
0 2012-01-30       0.9      0.252         0.25     -0.08        0.125
1 2012-02-04       0.9      0.251         0.25     -0.08        0.125
2 2012-02-09       0.9      0.247         0.25     -0.08        0.122
3 2012-02-14       0.9      0.242         0.24     -0.08        0.119
4 2012-02-19       0.9      0.237         0.24     -0.08        0.115

   NDVI_variance  Anomaly   Z-Score
0          0.016 -0.03821 -0.451159
1          0.016 -0.03921 -0.462966
2          0.015 -0.04321 -0.510196
3          0.014 -0.04821 -0.569233
4          0.013 -0.05321 -0.628270
```



```python
# Check for Stationarity
# ARIMA models require the data to be stationary, which means that statistical 
properties like the mean and variance
# do not change over time. Then perform tests like the Augmented Dickey-Fuller 
test to check for stationarity and apply differencing if needed.

# define a threshold
adf_threshold = 0.05
# Perform the Augmented Dickey-Fuller test
adf_test_result = adfuller(df_cleaned['NDVI_mean'])

# Print the test results
print('ADF Statistic:', adf_test_result[0])
print('p-value:', adf_test_result[1])
print('Critical Values:', adf_test_result[4])

if (adf_test_result[1]) < adf_threshold:
   print('data is stationary and fit for ARIMA')
else:
    print('data is not stationary so not fit for ARIMA')

# If p-value is less than a threshold (e.g., 0.05), to means the data stationary
```

```
ADF Statistic: -3.168334307339745
p-value: 0.021892351233569137
Critical Values: {'1%': -3.4386546523763837, '5%': 
-2.865205472974755, '10%': -2.568721842653421}
data is stationary and fit for ARIMA
```

```python
# Now let's create and fit the ARIMA model to the cleaned time-
series data

# Define the order of the ARIMA model (p, d, q)
p = 1  # Autoregressive (AR) order
d = 1  # Differencing (I) order
q = 1  # Moving Average (MA) order

# Create and fit the ARIMA model
model = ARIMA(df_cleaned['NDVI_mean'], order=(p, d, q))
model_fit = model.fit()

# Display a summary of the model
print(model_fit.summary(10))
```

```
                               SARIMAX Results                                
==============================================================================
Dep. Variable:              NDVI_mean   No. Observations:                  812
Model:                 ARIMA(1, 1, 1)   Log Likelihood                2515.652
Date:                Thu, 12 Oct 2023   AIC                          -5025.305
Time:                        08:40:08   BIC                          -5011.210
Sample:                             0   HQIC                         -5019.894
                                - 812                                         
Covariance Type:                  opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [5.0       -4.0]
------------------------------------------------------------------------------
ar.L1          0.8633      0.014     60.378      0.000        nan         nan
ma.L1         -0.3431      0.027    -12.573      0.000        nan         nan
sigma2         0.0001   1.69e-06     69.915      0.000        nan         nan
===================================================================================
Ljung-Box (L1) (Q):                   4.76   Jarque-Bera (JB):            378318.01
Prob(Q):                              0.03   Prob(JB):                         0.00
Heteroskedasticity (H):               8.77   Skew:                             5.68
Prob(H) (two-sided):                  0.00   Kurtosis:                       108.20
```

```
# Obtain predictions on testing data
```

```
718   -0.122168
698   -0.123482
657         NaN
749   -0.149298
662   -0.035849
808   -0.010307
665   -0.025652
759   -0.172303
708   -0.088969
798   -0.186307
dtype: float64
```

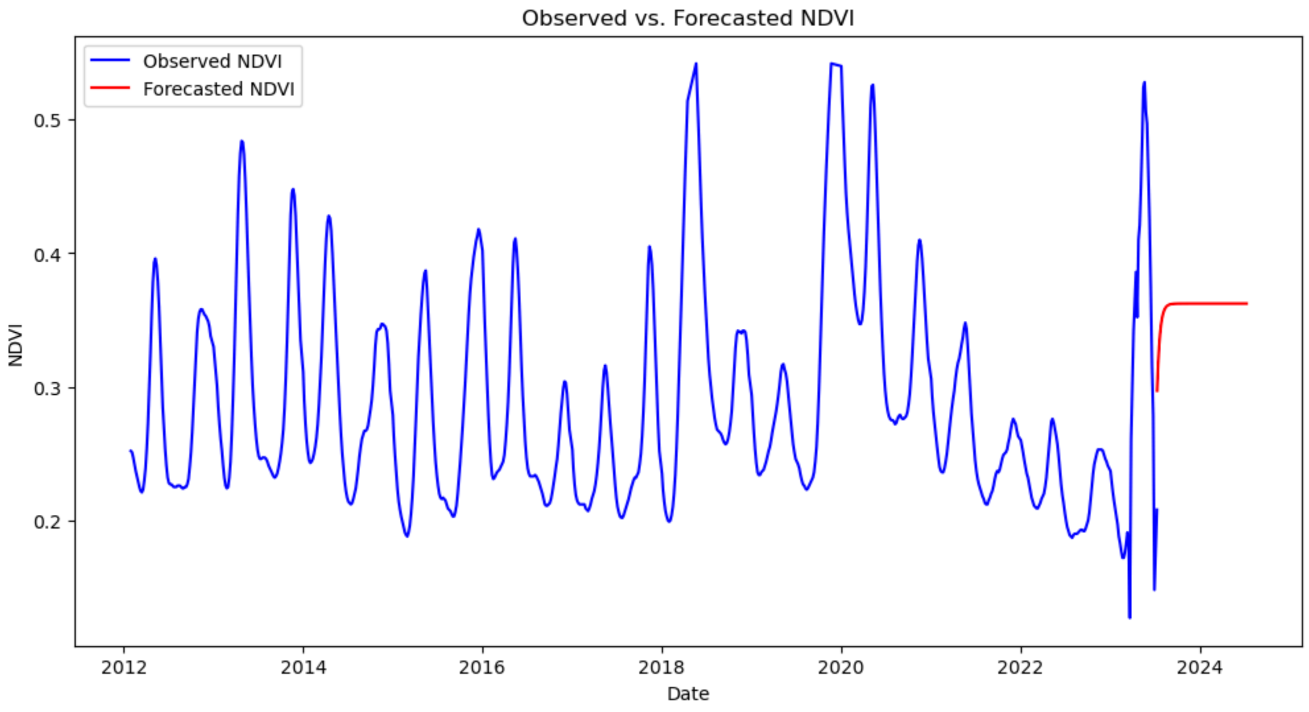
```
sm_predictions = split_model_fit.forecast(steps=len(sm_test))
# Calculate residuals
sm_residuals = sm_test - sm_predictions
sm_residuals.sample(10)
```

```python
# Calculate summary statistics of residuals
sm_residual_mean = np.mean(sm_residuals)
sm_residual_std = np.std(sm_residuals)
# Perform statistical tests (e.g., Ljung-Box test, Shapiro-Wilk test)

# Calculate performance metrics (e.g., MAE, MSE, RMSE)
sm_mae = np.mean(np.abs(sm_residuals))
sm_mse = np.mean(sm_residuals**2)
sm_rmse = np.sqrt(sm_mse)
print('sm_mae:', sm_mae)
print('sm_mse:', sm_mse)
print('sm_rmse:', sm_rmse)
```

[134] Python

```
sm_mae: 0.12024585323344028
sm_mse: 0.01662670402991391
sm_rmse: 0.12894457735753725
```

```python
#  Let's assume we are okay with the current model setup
#  We can then perform a forward-looking prediction

# Forecast NDVI values using your ARIMA model
sm_forecast_periods = 365  # Specify the number of periods to forecast

sm_forecast = split_model_fit.forecast(steps=sm_forecast_periods)

# Create a date range for the forecasted periods
sm_last_observed_date = df_cleaned['Date'].max()

sm_forecast_dates = pd.date_range(start=sm_last_observed_date,
periods=sm_forecast_periods + 1)

# Create a DataFrame for the forecasts
sm_forecast_df = pd.DataFrame({'Date': sm_forecast_dates[1:],
'NDVI_mean_forecast': sm_forecast})

# Merge the forecasts with the original dataset
sm_merged_df = pd.concat([df_cleaned, sm_forecast_df], ignore_index=True)

# Plot the comparison of observed and forecasted NDVI
plt.figure(figsize=(12, 6))
plt.plot(sm_merged_df['Date'], sm_merged_df['NDVI_mean'], label='Observed NDVI',
color='blue')
plt.plot(sm_merged_df['Date'], sm_merged_df['NDVI_mean_forecast'],
label='Forecasted NDVI', color='red')
plt.xlabel('Date')
plt.ylabel('NDVI')
plt.title('Observed vs. Forecasted NDVI')
plt.legend()
plt.show()
```



5. Model Validation

Validation Technique: Cross-validation was performed on the NDVI mean values by splitting the data into testing and training subsets. To further validate the model, various tests were performed including Shapiro-Wilk test. The following steps explains the process:

```
# Split data into training and testing
# model = ARIMA(df_cleaned['NDVI_mean'], order=(p, d, q))
sm_train_size = int(len(df_cleaned['NDVI_mean']) * 0.8)
sm_train, sm_test = df_cleaned['NDVI_mean'][:sm_train_size],
df_cleaned['NDVI_mean'][sm_train_size:]
```

```
# Fit ARIMA model to training data
# model = ARIMA(df_cleaned['NDVI_mean'], order=(p, d, q))
split_model = ARIMA(sm_train, order=(p, d, q))
split_model_fit = split_model.fit()
```

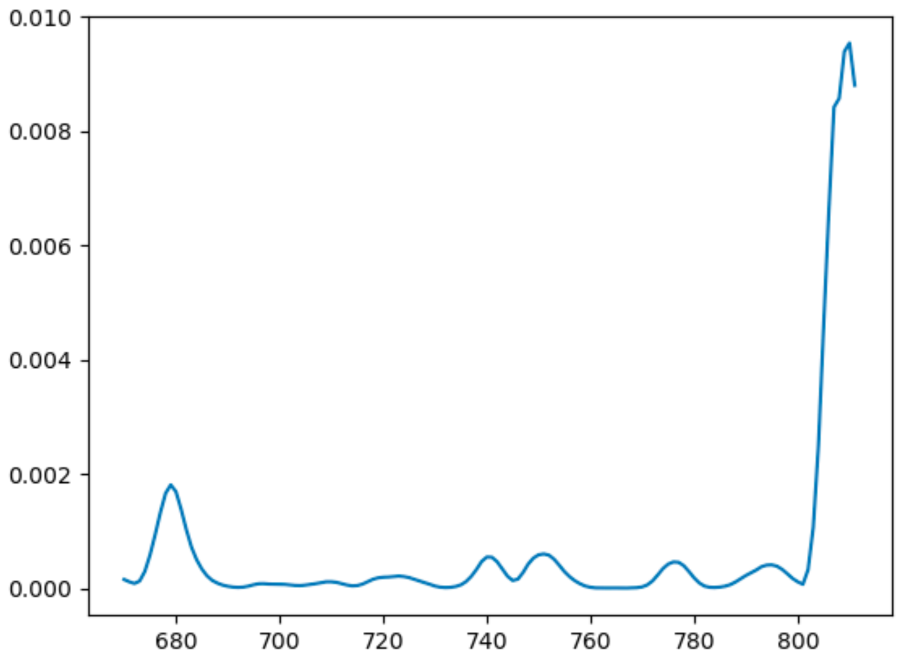
```
# The Shapiro-Wilk test is a statistical test used to assess whether a dataset
follows a normal distribution.

# Assume 'residuals' is your array of residuals
```

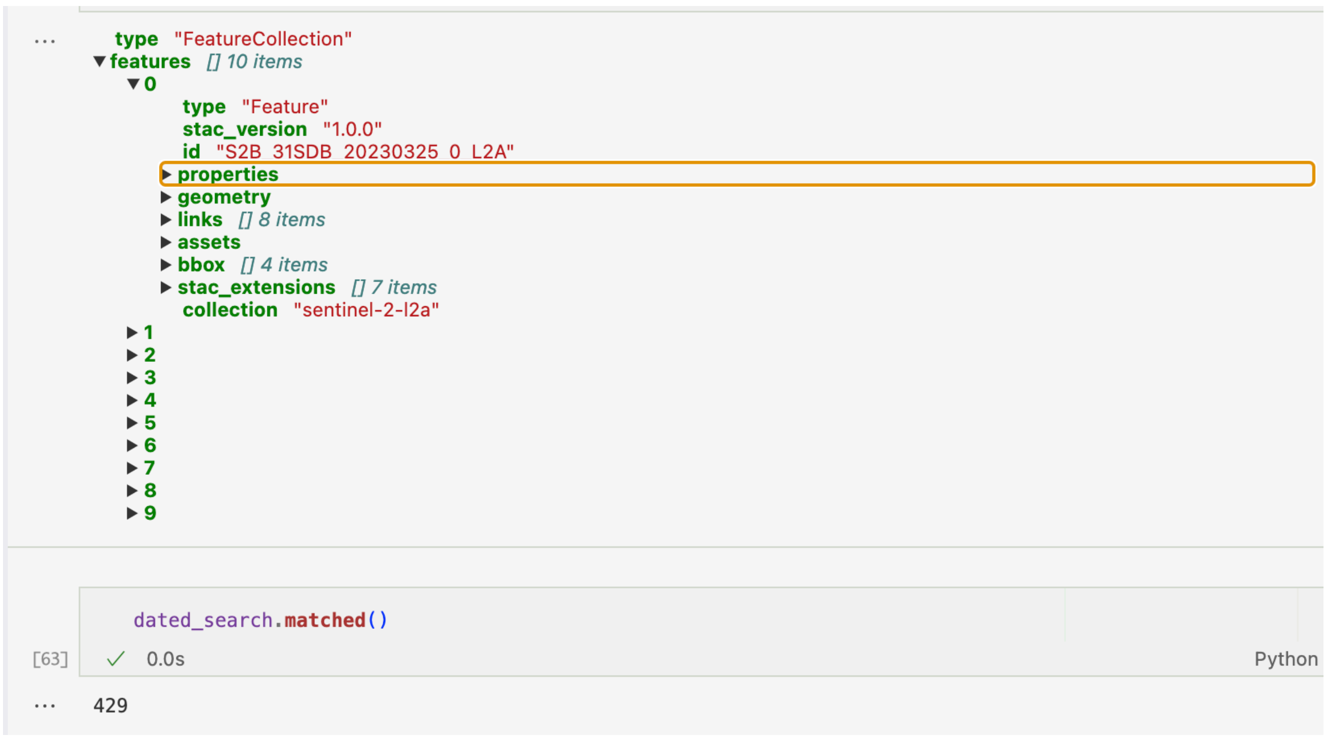
```
statistic, p_value = shapiro(sm_residuals)

if p_value < 0.05:
    print("Reject the null hypothesis: Data is not normally distributed.")
else:
    print("Fail to reject the null hypothesis: Data is normally distributed.")
```

Output:

```
Fail to reject the null hypothesis: Data is normally distributed.
```

```
#2. Heteroskedasticity:
# Plot the rolling variance of the residuals over time.
# This helps you visualize changing variance. A noticeable pattern in the plot,
could indicate heteroskedasticity.

sm_rolling_variance = sm_residuals.rolling(window=window_size).var()
plt.plot(sm_rolling_variance)
```





```
...   type  "FeatureCollection"
    ▼features  [] 10 items
        ▼0
            type  "Feature"
            stac_version  "1.0.0"
            id  "S2B_31SDB_20230325_0_L2A"
          ▶ properties
          ▶ geometry
          ▶ links  [] 8 items
          ▶ assets
          ▶ bbox  [] 4 items
          ▶ stac_extensions  [] 7 items
            collection  "sentinel-2-l2a"
        ▶1
        ▶2
        ▶3
        ▶4
        ▶5
        ▶6
        ▶7
        ▶8
        ▶9
```

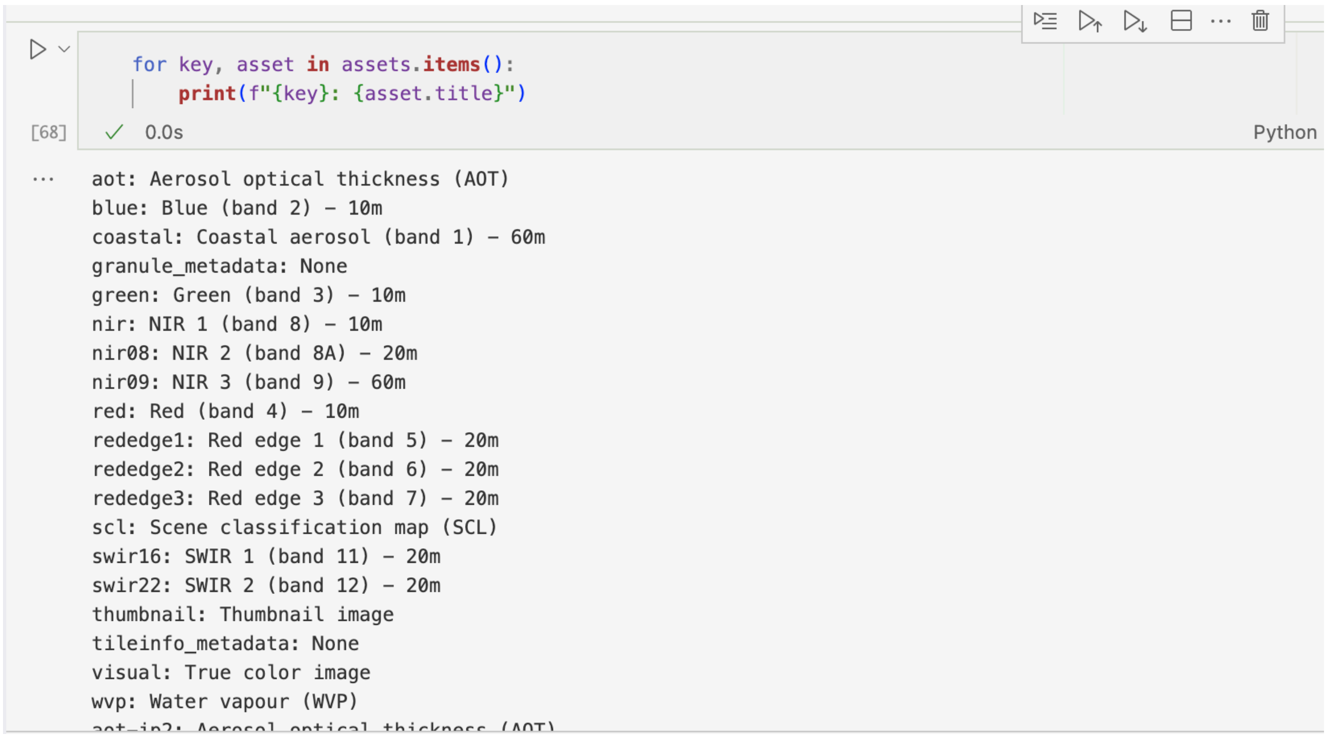
```
dated_search.matched()
```

[63] ✓ 0.0s Python

```
... 429
```

Performance Metrics: RSME, MSE, and MAE were used to measure the

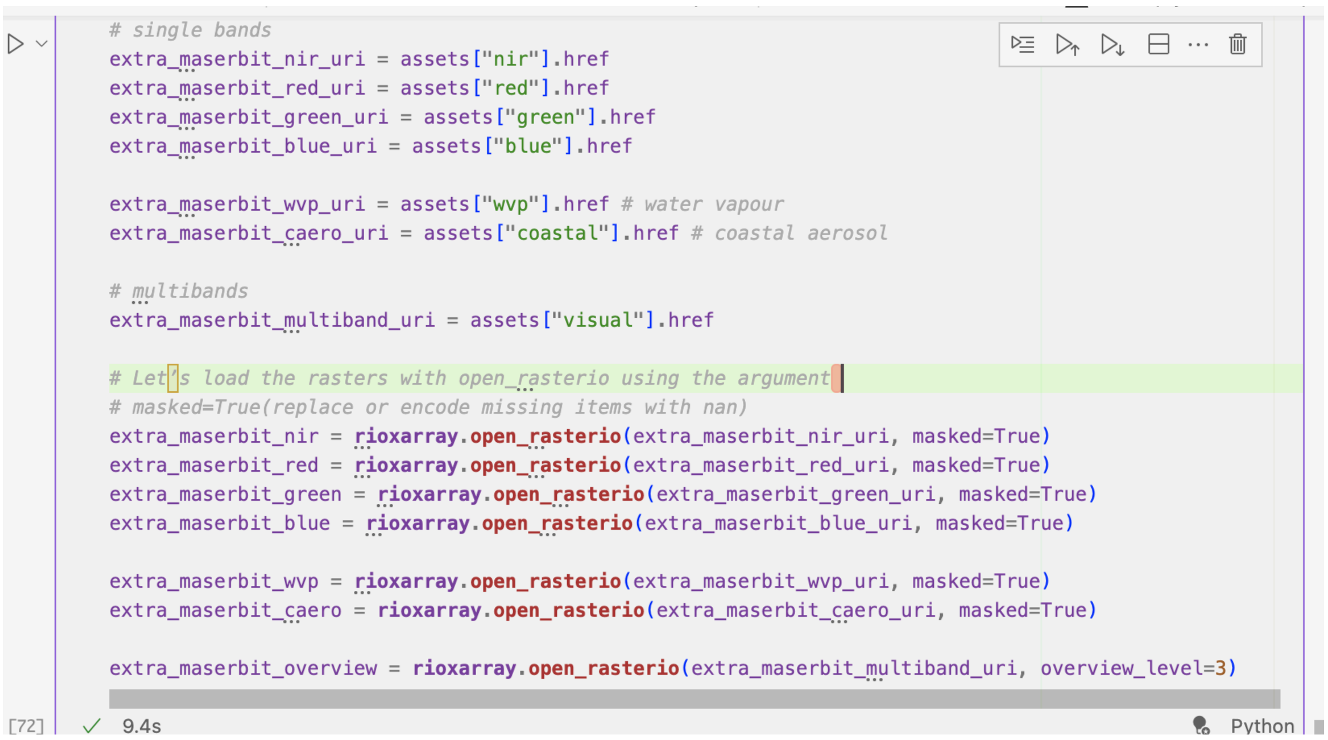
performance of the model. The error gotten was good enough. For example the RSME was 0.1289. Other performance metrics was the various test carried out like normality test, before the forecasting.



```
for key, asset in assets.items():
    print(f"{key}: {asset.title}")
```

[68] ✓ 0.0s Python

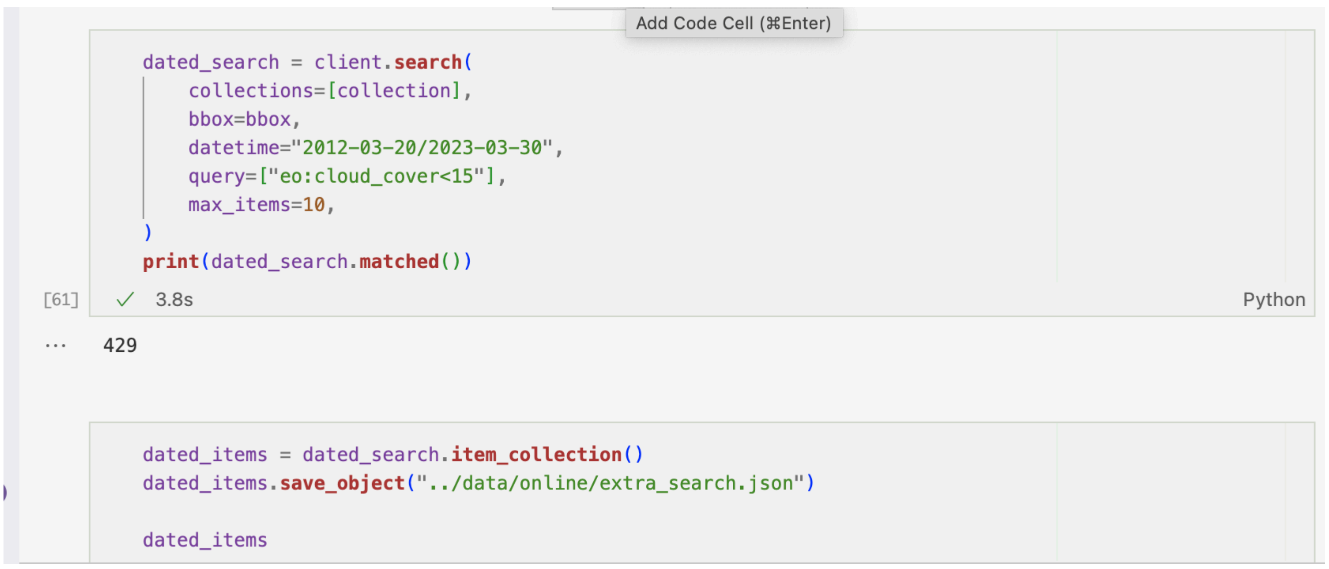
```
... aot: Aerosol optical thickness (AOT)
    blue: Blue (band 2) - 10m
    coastal: Coastal aerosol (band 1) - 60m
    granule_metadata: None
    green: Green (band 3) - 10m
    nir: NIR 1 (band 8) - 10m
    nir08: NIR 2 (band 8A) - 20m
    nir09: NIR 3 (band 9) - 60m
    red: Red (band 4) - 10m
    rededge1: Red edge 1 (band 5) - 20m
    rededge2: Red edge 2 (band 6) - 20m
    rededge3: Red edge 3 (band 7) - 20m
    scl: Scene classification map (SCL)
    swir16: SWIR 1 (band 11) - 20m
    swir22: SWIR 2 (band 12) - 20m
    thumbnail: Thumbnail image
    tileinfo_metadata: None
    visual: True color image
    wvp: Water vapour (WVP)
    aot-jp2: Aerosol optical thickness (AOT)
```


```
# single bands
extra_maserbit_nir_uri = assets["nir"].href
extra_maserbit_red_uri = assets["red"].href
extra_maserbit_green_uri = assets["green"].href
extra_maserbit_blue_uri = assets["blue"].href

extra_maserbit_wvp_uri = assets["wvp"].href # water vapour
extra_maserbit_caero_uri = assets["coastal"].href # coastal aerosol

# multibands
extra_maserbit_multiband_uri = assets["visual"].href

# Let's load the rasters with open_rasterio using the argument
# masked=True(replace or encode missing items with nan)
extra_maserbit_nir = rioxarray.open_rasterio(extra_maserbit_nir_uri, masked=True)
extra_maserbit_red = rioxarray.open_rasterio(extra_maserbit_red_uri, masked=True)
extra_maserbit_green = rioxarray.open_rasterio(extra_maserbit_green_uri, masked=True)
extra_maserbit_blue = rioxarray.open_rasterio(extra_maserbit_blue_uri, masked=True)

extra_maserbit_wvp = rioxarray.open_rasterio(extra_maserbit_wvp_uri, masked=True)
extra_maserbit_caero = rioxarray.open_rasterio(extra_maserbit_caero_uri, masked=True)

extra_maserbit_overview = rioxarray.open_rasterio(extra_maserbit_multiband_uri, overview_level=3)
```
[72] ✓ 9.4s Python

6. Additional Data Integration

Data Sourcing: Additional data was sourced from the aws sentinel collection. The goal of sourcing an extra data was to boost the static COG data especially since it was missing some metadata like water vapour, red, blue, and green, bands that could help to generate additional information for the forecast. Fortunately, the retrieved data possessed this additional information. Most important one at this stage is the presence of water vapour in the metadata, which is crucial in predicting the vegetation health of any place and Marsabit in this context.

```
# access data


api_url = "https://earth-search.aws.element84.com/v1"

Client = Client.open(api_url)
```

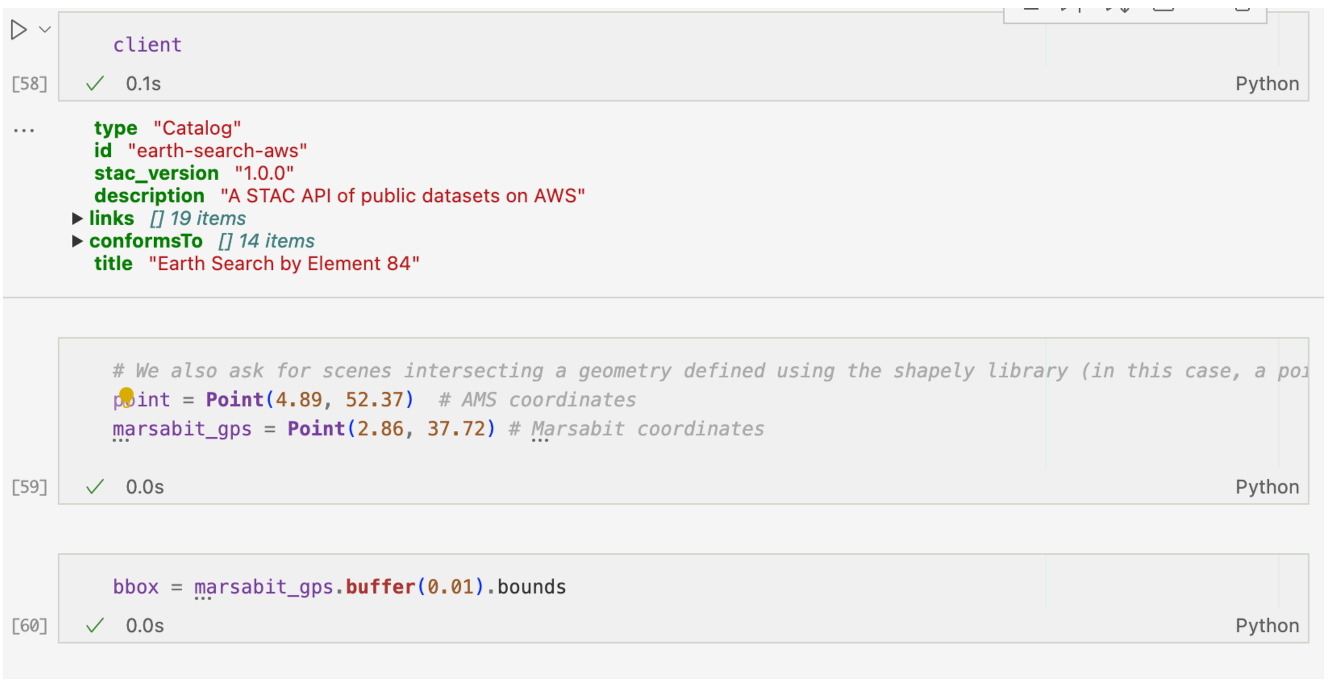


```
dated_search = client.search(
    collections=[collection],
    bbox=bbox,
    datetime="2012-03-20/2023-03-30",
    query=["eo:cloud_cover<15"],
    max_items=10,
)
print(dated_search.matched())
```

[61] ✓ 3.8s Python

```
429
```

```
dated_items = dated_search.item_collection()
dated_items.save_object("../data/online/extra_search.json")

dated_items
```



```
client
```

[58] ✓ 0.1s Python

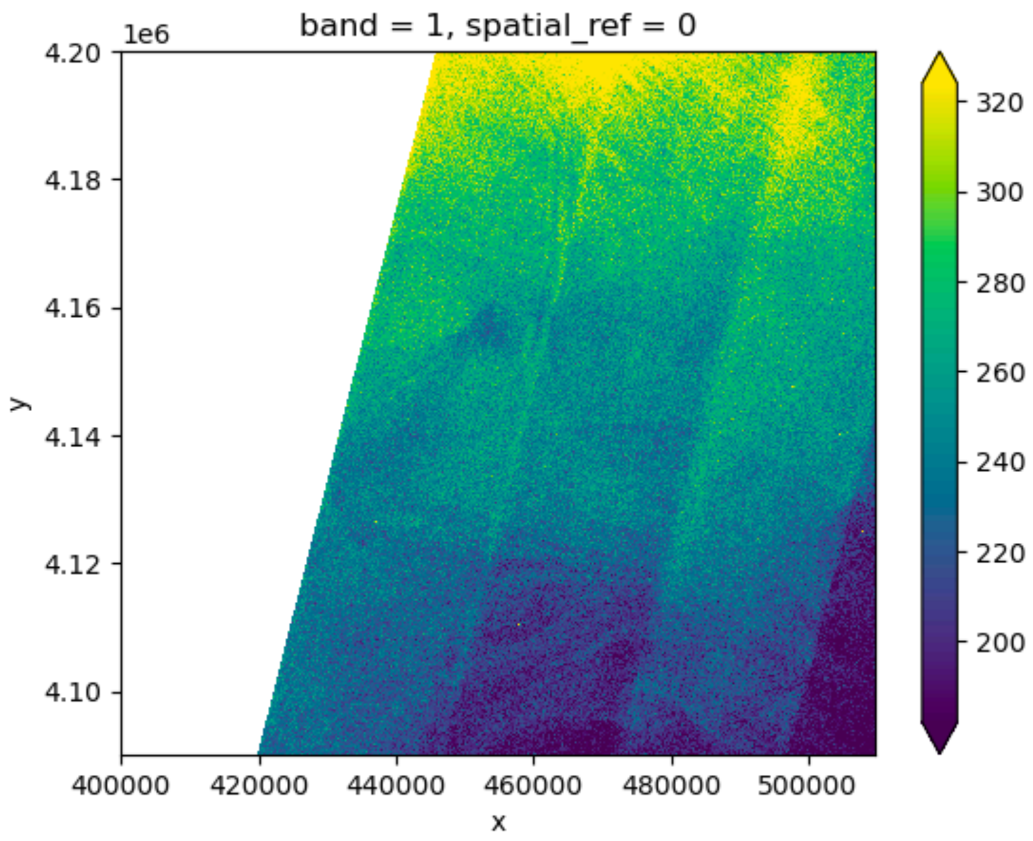
```
  type  "Catalog"
  id  "earth-search-aws"
  stac_version  "1.0.0"
  description  "A STAC API of public datasets on AWS"
▶ links  [] 19 items
▶ conformsTo  [] 14 items
  title  "Earth Search by Element 84"
```

```
# We also ask for scenes intersecting a geometry defined using the shapely library (in this case, a poi
point = Point(4.89, 52.37)  # AMS coordinates
marsabit_gps = Point(2.86, 37.72) # Marsabit coordinates
```

[59] ✓ 0.0s Python

```
bbox = marsabit_gps.buffer(0.01).bounds
```

[60] ✓ 0.0s Python



Data Integration: The data integration that could be done at the space of time allowed was to fetch the coordinates of Marsabit County as referenced by the coordinates geoJSON information and use it to narrow our data search. This ensured that the exact data for Marsabit coordinates was fetched for further modelling.

7. Model Interpretation

Model Selection: I Started first with the ARIMA model, and noticed the the algorithm smartly detected seasonality in the dataset and consequently performed computations with it. Overall the model could detect seasonal.

The following output is the result of fitting an ARIMA(1, 1, 1) model to the NDVI mean time series data:

Model Order: The ARIMA model is specified as (1, 1, 1), which indicates:

p (Autoregressive order) = 1
d (Differencing order) = 1
q (Moving Average order) = 1

This means that the model incorporates one autoregressive term, one differencing step, and one moving average term.
Dep. Variable: This specifies the dependent variable, which is 'NDVI_mean' in this case.
No. Observations: This tells the number of data points in the dataset, which is 649. This sample size is high enough for the analysis.
Log Likelihood: The log likelihood is a measure of how well the model fits the data. Higher values indicate a better fit. In this case, a Log Likelihood of 2702.230 suggests that the model fits the data well.
Information Criteria (AIC, BIC, HQIC): These are model selection criteria. Lower values are generally preferred. The Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), and Hannan-Quinn Information Criterion (HQIC) are measures used to compare the relative quality of different models. In this output, you can see that the AIC is -5398.459, the BIC is -5385.038, and the HQIC is -5393.253. These values are used for model selection, with lower values indicating a better fit.
Covariance Type: This indicates how the covariance matrix was calculated for the model. The output mentions "opg," which typically stands for "outer product of gradients."
Parameter Coefficients: The "coef" column displays the estimated coefficients for the autoregressive (ar.L1) and moving average (ma.L1) terms. These coefficients represent the strength and direction of the influence of these terms on the time series.
Residual Variance (sigma2): This represents the variance of the model residuals. In this case, it's approximately 1.388e-05.

Ljung-Box Statistic (Q): This is a test for autocorrelation in the residuals. The result shows that the Q statistic at lag 1 is 7.55, and the p-value associated with it is 0.01. A small p-value (less than 0.05) indicates the presence of autocorrelation in the residuals.
Jarque-Bera Statistic (JB): This is a test for normality of the residuals. A significant p-value suggests non-normality. In this case, the p-value is very close to zero, indicating non-normality.
Heteroskedasticity (H): This is a test for heteroskedasticity in the residuals. A significant p-value suggests the presence of heteroskedasticity. In this case, the p-value is very close to zero, indicating heteroskedasticity.
Skew and Kurtosis: These are measures of the skewness and kurtosis of the residuals. High skewness and kurtosis values can indicate non-normality.

In summary, the ARIMA(1, 1, 1) model seems to fit the data well based on the Log Likelihood and information criteria. However, there are indications of issues such as non-normality, autocorrelation, and heteroskedasticity in the residuals. These issues may need further investigation, and additional diagnostic checks and model adjustments might be necessary.

8. References

Additional Data Source: `https://earth-search.aws.element84.com/v1`

`https://www.digitalearthafrica.org/platform-resources/analysis-tools`

https://en.wikipedia.org/wiki/Marsabit

https://kenyarapid.acaciadata.com/media/cluster3/Marsabit_County_WR_factsheet.pdf