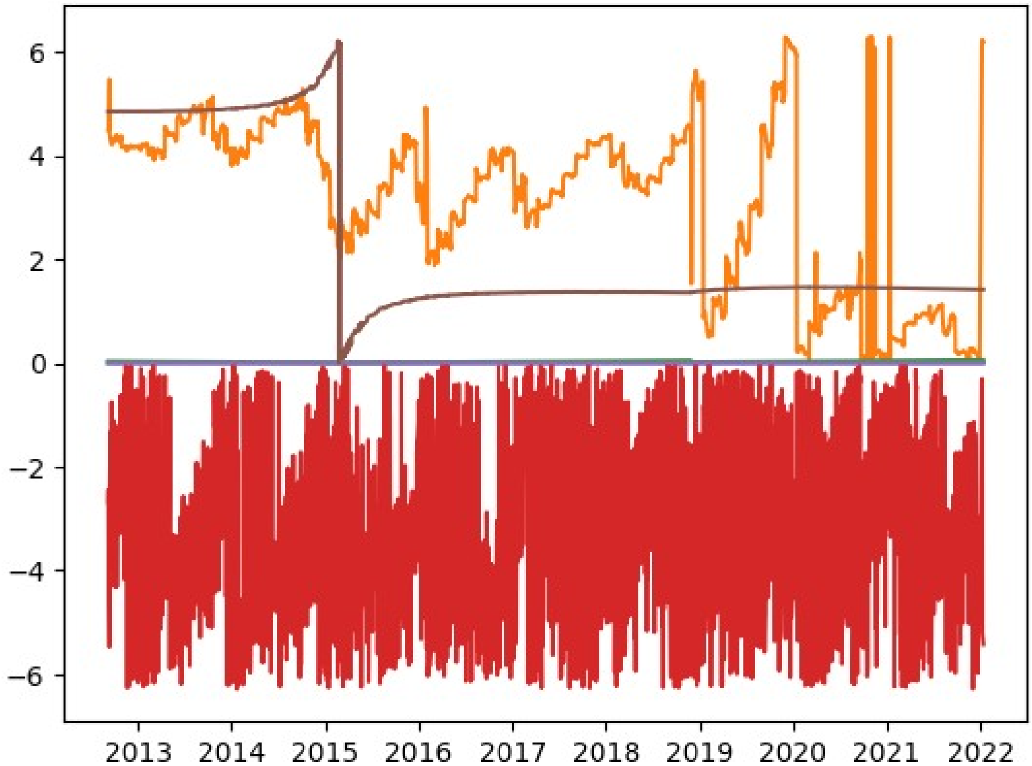
```
!pip install statsmodels==0.14.4
```

```
Requirement already satisfied: statsmodels==0.14.4 in c:\users\prakh\desktop\research_project\venv\lib\site-packages (0.14.4)
Requirement already satisfied: numpy<3,>=1.22.3 in c:\users\prakh\desktop\research_project\venv\lib\site-packages (from statsmodels==0.14.4) (2.0.2)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in c:\users\prakh\desktop\research_project\venv\lib\site-packages (from statsmodels==0.14.4) (1.13.1)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in c:\users\prakh\desktop\research_project\venv\lib\site-packages (from statsmodels==0.14.4) (2.2.3)
Requirement already satisfied: patsy>=0.5.6 in c:\users\prakh\desktop\research_project\venv\lib\site-packages (from statsmodels==0.14.4) (1.0.1)
Requirement already satisfied: packaging>=21.3 in c:\users\prakh\desktop\research_project\venv\lib\site-packages (from statsmodels==0.14.4) (24.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\prakh\desktop\research_project\venv\lib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels==0.14.4) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\prakh\desktop\research_project\venv\lib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels==0.14.4) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\prakh\desktop\research_project\venv\lib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels==0.14.4) (2025.1)
Requirement already satisfied: six>=1.5 in c:\users\prakh\desktop\research_project\venv\lib\site-packages (from python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4->statsmodels==0.14.4) (1.17.0)


[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.dates as mdates
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
import warnings
warnings.filterwarnings(action='ignore')
```

```
# Load the dataset (adjust file path and format as needed)
df = pd.read_csv(r"C:\Users\prakh\Desktop\Research_Project\satellite_data\orbital_elements\Fengyun-2F.csv", parse_dates=[0])  #
```

```
Assuming first column is datetime

# Rename columns to remove spaces
df.columns = df.columns.str.strip().str.replace(" ", "_")

# Convert datetime column to proper format (if not done already)
#df.iloc[:, 0] = pd.to_datetime(df.iloc[:, 0], dayfirst=True)  #
Adjust based on date format

# Convert numeric columns to float (excluding datetime column)
#df.iloc[:, 1:] = df.iloc[:, 1:].apply(pd.to_numeric, errors="coerce")

# Handle missing values (fill or drop)
#df.dropna(inplace=True)  # Or df.fillna(value, inplace=True)

# Display processed data
print(df.head())

# Save cleaned data
df.to_csv("cleaned_data.csv", index=False)
df.rename(columns={'Unnamed:_0':'Datetime'},inplace=True)
df.set_index("Datetime",inplace=True)

                  Unnamed:_0  eccentricity  argument_of_perigee
inclination  \
0 2012-09-06 18:48:32.050655      0.000488             4.483911
0.032940
1 2012-09-07 19:39:45.383327      0.000487             4.481215
0.032901
2 2012-09-08 15:43:39.075167      0.000487             4.475122
0.032868
3 2012-09-09 12:53:36.595967      0.000492             4.481063
0.032835
4 2012-09-10 13:15:22.135391      0.000495             4.512943
0.032798

   mean_anomaly  Brouwer_mean_motion  right_ascension
0     -2.689729             0.004374         4.842139
1     -2.446726             0.004374         4.842234
2     -3.457326             0.004374         4.842332
3     -4.190783             0.004374         4.842201
4     -4.111348             0.004374         4.842257

import matplotlib.pyplot as plt
plt.plot(df.index, df)
plt.plot()

[]
```



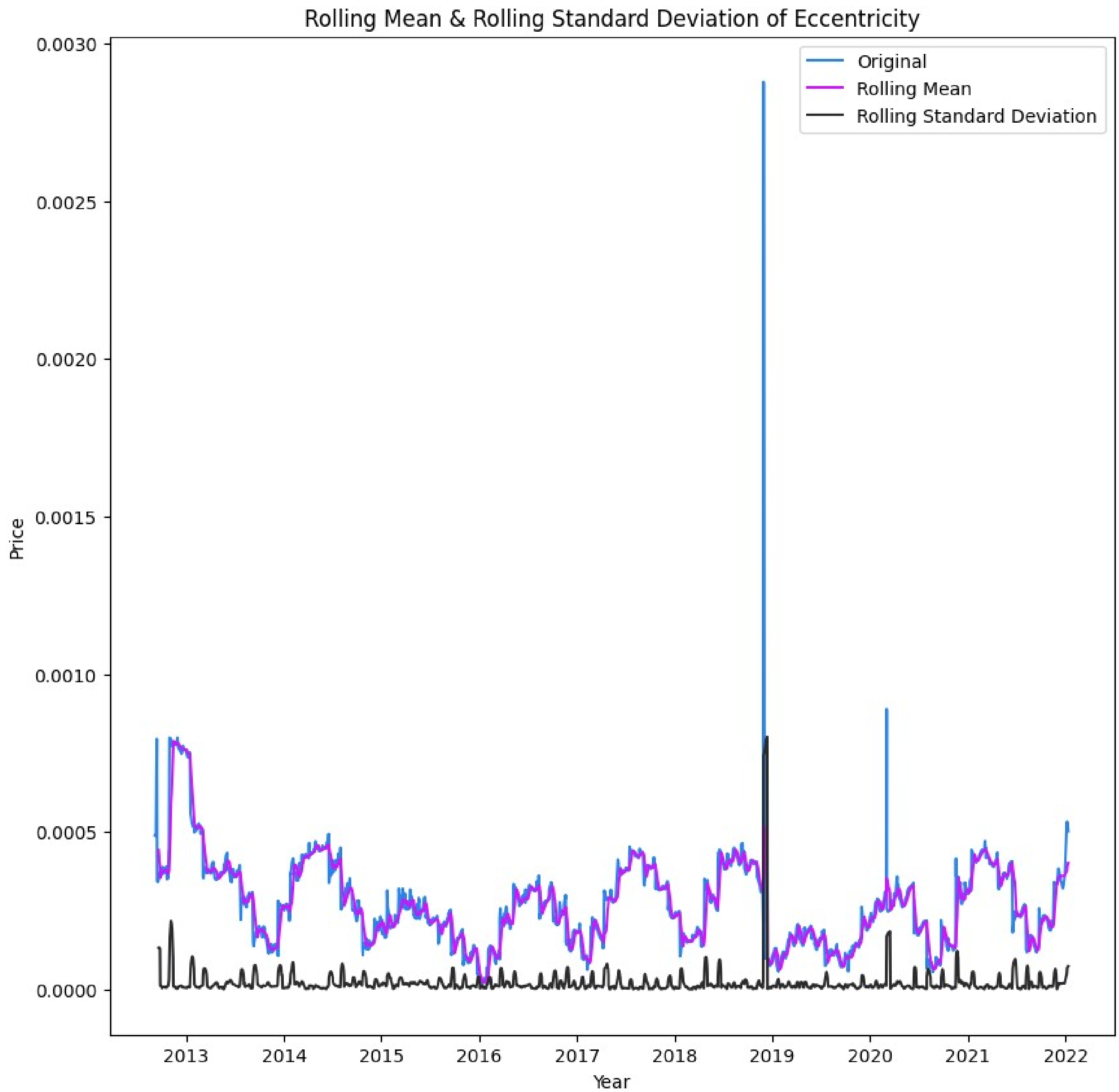

```
rolling_mean_eccentricity= df[:]
['eccentricity'].rolling(window=12).mean()
rolling_std_eccentricity= df[:]
['eccentricity'].rolling(window=12).std()

rolling_mean_argument_of_perigee= df[:]
['argument_of_perigee'].rolling(window=12).mean()
rolling_std_argument_of_perigee= df[:]
['argument_of_perigee'].rolling(window=12).std()

rolling_mean_inclination= df[:]
['inclination'].rolling(window=12).mean()
rolling_std_inclination= df[:]['inclination'].rolling(window=12).std()

rolling_mean_mean_anomaly= df[:]
['mean_anomaly'].rolling(window=12).mean()
rolling_std_mean_anomaly= df[:]
['mean_anomaly'].rolling(window=12).std()

rolling_mean_Brouwer_mean_motion= df[:]
['Brouwer_mean_motion'].rolling(window=12).mean()
rolling_std_Brouwer_mean_motion= df[:]
['Brouwer_mean_motion'].rolling(window=12).std()

rolling_mean_right_ascension= df[:]
['right_ascension'].rolling(window=12).mean()
```

```
rolling_std_right_ascension= df[:]
['right_ascension'].rolling(window=12).std()

mean=df.rolling(window=12).mean()
std=df.rolling(window=12).std()

print(mean)
print(std)
#print(rolling_mean)
#print(rolling_std)
```

```
                            eccentricity  argument_of_perigee
inclination  \
Datetime

2012-09-06 18:48:32.050655           NaN                  NaN
NaN
2012-09-07 19:39:45.383327           NaN                  NaN
NaN
2012-09-08 15:43:39.075167           NaN                  NaN
NaN
2012-09-09 12:53:36.595967           NaN                  NaN
NaN
2012-09-10 13:15:22.135391           NaN                  NaN
NaN
...                                  ...                  ...
...
2021-12-25 04:39:15.108767      0.000360             0.218050
0.058125
2021-12-28 21:00:18.080928      0.000361             0.210942
0.058220
2022-01-06 13:25:24.541247      0.000376             0.707352
0.058343
2022-01-07 18:42:19.086048      0.000390             1.203023
0.058467
2022-01-11 17:26:36.259583      0.000402             1.695039
0.058602

                            mean_anomaly  Brouwer_mean_motion
right_ascension
Datetime

2012-09-06 18:48:32.050655           NaN                  NaN
NaN
2012-09-07 19:39:45.383327           NaN                  NaN
NaN
2012-09-08 15:43:39.075167           NaN                  NaN
NaN
2012-09-09 12:53:36.595967           NaN                  NaN
NaN
```

```
2012-09-10 13:15:22.135391           NaN                NaN
NaN
...                                  ...                ...
...
2021-12-25 04:39:15.108767     -3.213270           0.004375
1.414584
2021-12-28 21:00:18.080928     -3.323417           0.004375
1.414355
2022-01-06 13:25:24.541247     -2.825800           0.004375
1.414092
2022-01-07 18:42:19.086048     -3.146032           0.004375
1.413843
2022-01-11 17:26:36.259583     -3.326725           0.004375
1.413550

[2985 rows x 6 columns]
                            eccentricity  argument_of_perigee
inclination  \
Datetime

2012-09-06 18:48:32.050655           NaN                NaN
NaN
2012-09-07 19:39:45.383327           NaN                NaN
NaN
2012-09-08 15:43:39.075167           NaN                NaN
NaN
2012-09-09 12:53:36.595967           NaN                NaN
NaN
2012-09-10 13:15:22.135391           NaN                NaN
NaN
...                                  ...                ...
...
2021-12-25 04:39:15.108767      0.000021           0.057474
0.000308
2021-12-28 21:00:18.080928      0.000020           0.061539
0.000357
2022-01-06 13:25:24.541247      0.000052           1.734288
0.000465
2022-01-07 18:42:19.086048      0.000068           2.344797
0.000534
2022-01-11 17:26:36.259583      0.000075           2.722676
0.000591

                            mean_anomaly  Brouwer_mean_motion
right_ascension
Datetime

2012-09-06 18:48:32.050655           NaN                NaN
NaN
2012-09-07 19:39:45.383327           NaN                NaN
```

```
NaN
2012-09-08 15:43:39.075167           NaN                  NaN
NaN
2012-09-09 12:53:36.595967           NaN                  NaN
NaN
2012-09-10 13:15:22.135391           NaN                  NaN
NaN
...                                  ...                  ...
...
2021-12-25 04:39:15.108767      1.861222         1.708880e-07
0.000601
2021-12-28 21:00:18.080928      1.926873         2.024589e-07
0.000825
2022-01-06 13:25:24.541247      1.864282         2.539249e-07
0.001074
2022-01-07 18:42:19.086048      1.919575         2.891206e-07
0.001233
2022-01-11 17:26:36.259583      2.029227         3.115571e-07
0.001392

[2985 rows x 6 columns]

plt.figure(figsize=(10,10))

plt.plot(df[:]['eccentricity'], color = '#2a83e8', label = 'Original')
#BLUE


plt.plot(rolling_mean_eccentricity, color = '#d014fa', label =
'Rolling Mean') #PINK
plt.plot(rolling_std_eccentricity, color = '#2b2b2e', label = 'Rolling
Standard Deviation')  #BLACK

#plt.plot(rolling_mean_argument_of_perigee, color = '#a633ff', label =
'Rolling Mean') #PURPLR
#plt.plot(rolling_std_argument_of_perigee, color = '#ffd700', label =
'Rolling Standard Deviation') #YELLOW

plt.legend(loc = 'best')
plt.xlabel('Year')
plt.ylabel('Price')
plt.title('Rolling Mean & Rolling Standard Deviation of Eccentricity')
plt.show()
```



Analysis of the Graph: Rolling Mean & Rolling Standard Deviation of Eccentricity

1. Understanding the Graph Components

- **Blue Line (Original Data):** Represents the actual values of eccentricity over time.
- **Black Line (Rolling Standard Deviation):** Measures the variability (volatility) of eccentricity over time.

---

2. Key Insights

A. General Trend of Eccentricity (Blue Line)

- **2013 to 2018:**
  - The eccentricity appears to have fluctuating but relatively stable values within a low range (~0.0005 or lower).
  - There are periodic variations, possibly indicating cyclic behavior.
- **2019:**
  - A **sharp spike in eccentricity** is observed, reaching above 0.0025.
  - This could indicate a sudden anomaly or a temporary perturbation in the system.
- **2020 to 2022:**
  - The eccentricity returns to lower values, fluctuating similarly to pre-2019 but showing a slight increasing trend.

B. Volatility Analysis (Black Line - Rolling Standard Deviation)

- **2013 to 2018:**
  - The rolling standard deviation remains **very low**, suggesting stable fluctuations.
- **2019:**
  - A **significant increase in volatility** coincides with the large spike in eccentricity.
  - This suggests that the anomaly was not just a small deviation but part of a larger instability event.
- **2020 to 2022:**
  - Volatility decreases again, though it remains slightly elevated compared to earlier years.

---

3. Possible Explanations for Observed Behavior

The 2019 Anomaly

A sudden spike in eccentricity could be due to:

- **Orbital perturbations** (e.g., gravitational influence from the Moon, Sun, or other celestial bodies).
- **Atmospheric drag effects** (if in low Earth orbit).
- **Maneuvers or station-keeping adjustments**.
- **Data anomalies or measurement errors**.

Cyclic Behavior Pre- and Post-2019

- The periodic oscillations in eccentricity suggest a **natural variation in orbital shape**, potentially linked to:
  - The precession of the orbit.
  - Long-term gravitational perturbations.

Slight Upward Trend in Eccentricity (Post-2020)

- If the eccentricity continues increasing, it may indicate **orbital decay** or **external influences** modifying the orbit.

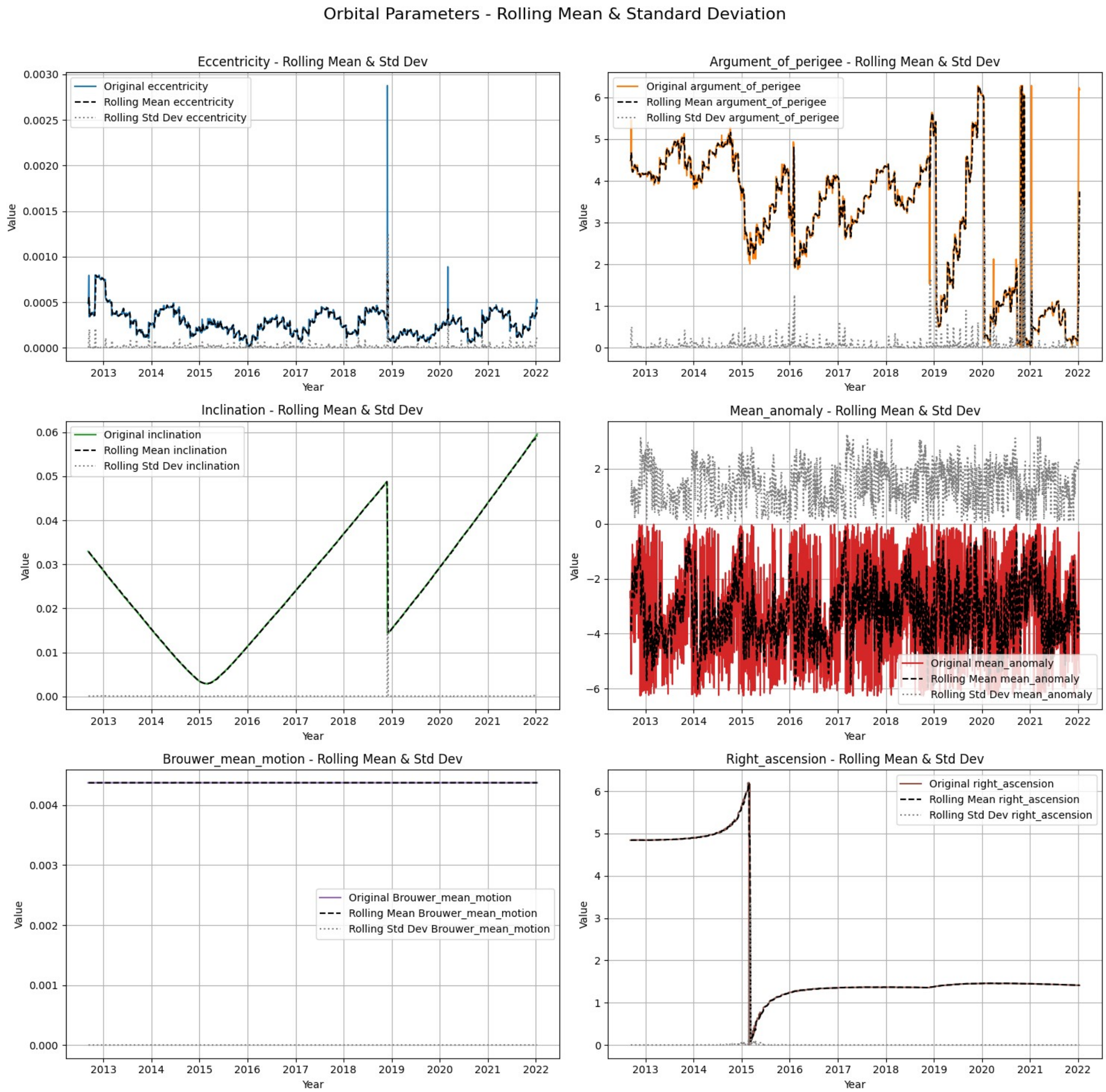
---

4. Key Takeaways

✔ Eccentricity remained relatively stable except for a major anomaly in 2019.
✔ 2019 saw an extreme spike in eccentricity, along with a surge in volatility.
✔ Post-2020, eccentricity remains within a normal range but shows a slight increasing trend.
✔ Potential causes include orbital perturbations, station-keeping events, or anomalies in data recording.

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
#file_path = r"C:\Users\prakh\Desktop\Research_Project\satellite_data\
orbital_elements\Fengyun-2F.csv"
#df = pd.read_csv(file_path)

# Define available parameters from the dataset
parameters = [
    "eccentricity", "argument_of_perigee", "inclination",
    "mean_anomaly", "Brouwer_mean_motion", "right_ascension"
]

# Define colors for each parameter
colors = [
    "#1f77b4", "#ff7f0e", "#2ca02c", "#d62728",
    "#9467bd", "#8c564b"
]

# Rolling window size
rolling_window = 5

# Compute rolling mean & standard deviation
rolling_data = {
    col: {
        "mean": df[col].rolling(rolling_window).mean(),
        "std": df[col].rolling(rolling_window).std()
    }
    for col in parameters
}

# Create subplots (3 rows, 2 columns) since we have 6 parameters
fig, axes = plt.subplots(3, 2, figsize=(15, 15))
fig.suptitle("Orbital Parameters - Rolling Mean & Standard Deviation",
```

```python
fontsize=16)

# Loop through each parameter and plot it
for i, param in enumerate(parameters):
    row, col = divmod(i, 2)  # Convert index to subplot row/column

    axes[row, col].plot(df.index, df[param], color=colors[i],
label=f"Original {param}")
    axes[row, col].plot(df.index, rolling_data[param]["mean"],
color="black", linestyle="dashed", label=f"Rolling Mean {param}")
    axes[row, col].plot(df.index, rolling_data[param]["std"],
color="gray", linestyle="dotted", label=f"Rolling Std Dev {param}")

    axes[row, col].set_title(f"{param.capitalize()} - Rolling Mean &
Std Dev")
    axes[row, col].set_xlabel("Year")
    axes[row, col].set_ylabel("Value")
    axes[row, col].legend(loc="best")
    axes[row, col].grid(True)

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.show()
```



Insights from the Graph

The graph consists of **six subplots**, each representing different **orbital parameters** of the **Fengyun-2F satellite** over time, along with their rolling mean and rolling standard deviation.

---

1 Eccentricity (Top-Left)

- **Blue Line (Original Data):** The eccentricity shows minor variations over time, with a few spikes around **2018-2019**.

- **Black Dashed Line (Rolling Mean):** The overall trend remains nearly constant, indicating a stable orbit.
- **Gray Dotted Line (Rolling Std Dev):** A small standard deviation suggests **low variations**, except for a **large spike in 2019**, indicating an anomaly.

□ **Insight:** The orbit remains nearly circular, but **2019 shows unusual behavior**.

---

2 Argument of Perigee (Top-Right)

- **Orange Line (Original Data):** The argument of perigee has a fluctuating increasing trend with occasional **sharp peaks**.
- **Black Dashed Line (Rolling Mean):** A steady increase over time.
- **Gray Dotted Line (Rolling Std Dev):** The variation is **highly volatile**, with **large fluctuations in 2020-2021**.

□ **Insight:** The orbital orientation changes significantly over time, likely due to **gravitational perturbations** or station-keeping maneuvers.

---

3 Inclination (Middle-Left)

- **Green Line (Original Data):** Shows a cyclic trend, **decreasing and increasing periodically**.
- **Black Dashed Line (Rolling Mean):** Follows the cyclical trend.
- **Gray Dotted Line (Rolling Std Dev):** Minimal variations except for sharp jumps.

□ **Insight:** The inclination is likely controlled actively, with **periodic station-keeping adjustments**.

---

4 Mean Anomaly (Middle-Right)

- **Red Line (Original Data):** The mean anomaly exhibits **high-frequency oscillations**.
- **Black Dashed Line (Rolling Mean):** A stable trend.
- **Gray Dotted Line (Rolling Std Dev):** Significant variations suggest **orbital perturbations**.

□ **Insight:** The satellite follows **expected periodic variations**, but noise indicates **external disturbances**.

---

5 Brouwer Mean Motion (Bottom-Left)

- **Purple Line (Original Data):** The mean motion remains **almost constant**.
- **Black Dashed Line (Rolling Mean):** No significant changes.
- **Gray Dotted Line (Rolling Std Dev):** Close to **zero**, confirming stability.

□ **Insight:** The satellite follows a **predictable orbit** with minimal variations.

---

6 Right Ascension (Bottom-Right)

- **Brown Line (Original Data):** A **sudden shift in 2015-2016**, after which it stabilizes.
- **Black Dashed Line (Rolling Mean):** Follows the shift.
- **Gray Dotted Line (Rolling Std Dev):** Large standard deviation during the shift.

 Insight: This suggests a **major orbital event in 2015-2016**, possibly a **maneuver or external force impact**.

Summary of Findings

- **Eccentricity: Stable**, except for **2019 anomaly**.
- **Argument of Perigee: Increasing trend** with **large fluctuations**.
- **Inclination: Cyclic variations**, likely **station-keeping adjustments**.
- **Mean Anomaly: High-frequency oscillations**, indicating **external perturbations**.
- **Mean Motion: Stable**, with **no significant changes**.
- **Right Ascension: Major shift in 2015-2016**, possibly a **maneuver or external force**.

 Conclusion: The satellite's orbit shows **expected long-term trends**, with **some anomalies** likely due to **station-keeping, external forces, or unexpected events**.

Checking if data is stationary using Dickey-Fuller Test

```
#Perform Augmented Dickey—Fuller test:
#Dickey—Fuller test on eccentricity
print('Results of Dickey Fuller Test on Eccentricity:')
result = adfuller(df[:]['eccentricity'], autolag='AIC')
result_output = pd.Series(result[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
for key,value in result[4].items():
    result_output['Critical Value (%s)'%key] = value
print(result_output)
```

```
Results of Dickey Fuller Test on Eccentricity:
Test Statistic                   -4.339020
p-value                           0.000380
#Lags Used                        7.000000
Number of Observations Used    2977.000000
Critical Value (1%)              -3.432549
Critical Value (5%)              -2.862511
Critical Value (10%)             -2.567287
dtype: float64
```

```
#Perform Augmented Dickey—Fuller test:
#Dickey—Fuller test on Argument of Perigee
```

```
print('Results of Dickey Fuller Test on Argument of Perigee:')
result = adfuller(df[:]['argument_of_perigee'], autolag='AIC')
result_output = pd.Series(result[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
for key,value in result[4].items():
    result_output['Critical Value (%s)'%key] = value
print(result_output)
```

```
Results of Dickey Fuller Test on Argument of Perigee:
Test Statistic                   -3.395790
p-value                           0.011103
#Lags Used                       28.000000
Number of Observations Used    2956.000000
Critical Value (1%)              -3.432564
Critical Value (5%)              -2.862518
Critical Value (10%)             -2.567291
dtype: float64
```

```
#Perform Augmented Dickey–Fuller test:
#Dickey–Fuller test on Inclination
print('Results of Dickey Fuller Test on Inclination')
result = adfuller(df[:]['inclination'], autolag='AIC')
result_output = pd.Series(result[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
for key,value in result[4].items():
    result_output['Critical Value (%s)'%key] = value
print(result_output)
```

```
Results of Dickey Fuller Test on Inclination
Test Statistic                   -0.165752
p-value                           0.942483
#Lags Used                        2.000000
Number of Observations Used    2982.000000
Critical Value (1%)              -3.432545
Critical Value (5%)              -2.862510
Critical Value (10%)             -2.567286
dtype: float64
```

```
#Perform Augmented Dickey–Fuller test:
#Dickey–Fuller test on Mean Anomaly
print('Results of Dickey Fuller Test on Mean Anomaly')
result = adfuller(df[:]['mean_anomaly'], autolag='AIC')
result_output = pd.Series(result[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
for key,value in result[4].items():
    result_output['Critical Value (%s)'%key] = value
print(result_output)
```

```
Results of Dickey Fuller Test on Mean Anomaly
Test Statistic                -7.838658e+00
```

```
p-value                           6.019126e-12
#Lags Used                        1.600000e+01
Number of Observations Used       2.968000e+03
Critical Value (1%)              -3.432555e+00
Critical Value (5%)              -2.862514e+00
Critical Value (10%)             -2.567289e+00
dtype: float64
```

```
#Perform Augmented Dickey–Fuller test:
#Dickey–Fuller test on Brouwer Mean Motion
print('Results of Dickey Fuller Test on Brouwer Mean Motion')
result = adfuller(df[:]['Brouwer_mean_motion'], autolag='AIC')
result_output = pd.Series(result[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
for key,value in result[4].items():
    result_output['Critical Value (%s)'%key] = value
print(result_output)
```

```
Results of Dickey Fuller Test on Brouwer Mean Motion
Test Statistic                   -1.644576e+01
p-value                           2.404200e-29
#Lags Used                        2.700000e+01
Number of Observations Used       2.957000e+03
Critical Value (1%)              -3.432563e+00
Critical Value (5%)              -2.862518e+00
Critical Value (10%)             -2.567291e+00
dtype: float64
```

```
#Perform Augmented Dickey–Fuller test:
#Dickey–Fuller test on Right Ascension
print('Results of Dickey Fuller Test on Right Ascension')
result = adfuller(df[:]['right_ascension'], autolag='AIC')
result_output = pd.Series(result[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
for key,value in result[4].items():
    result_output['Critical Value (%s)'%key] = value
print(result_output)
```

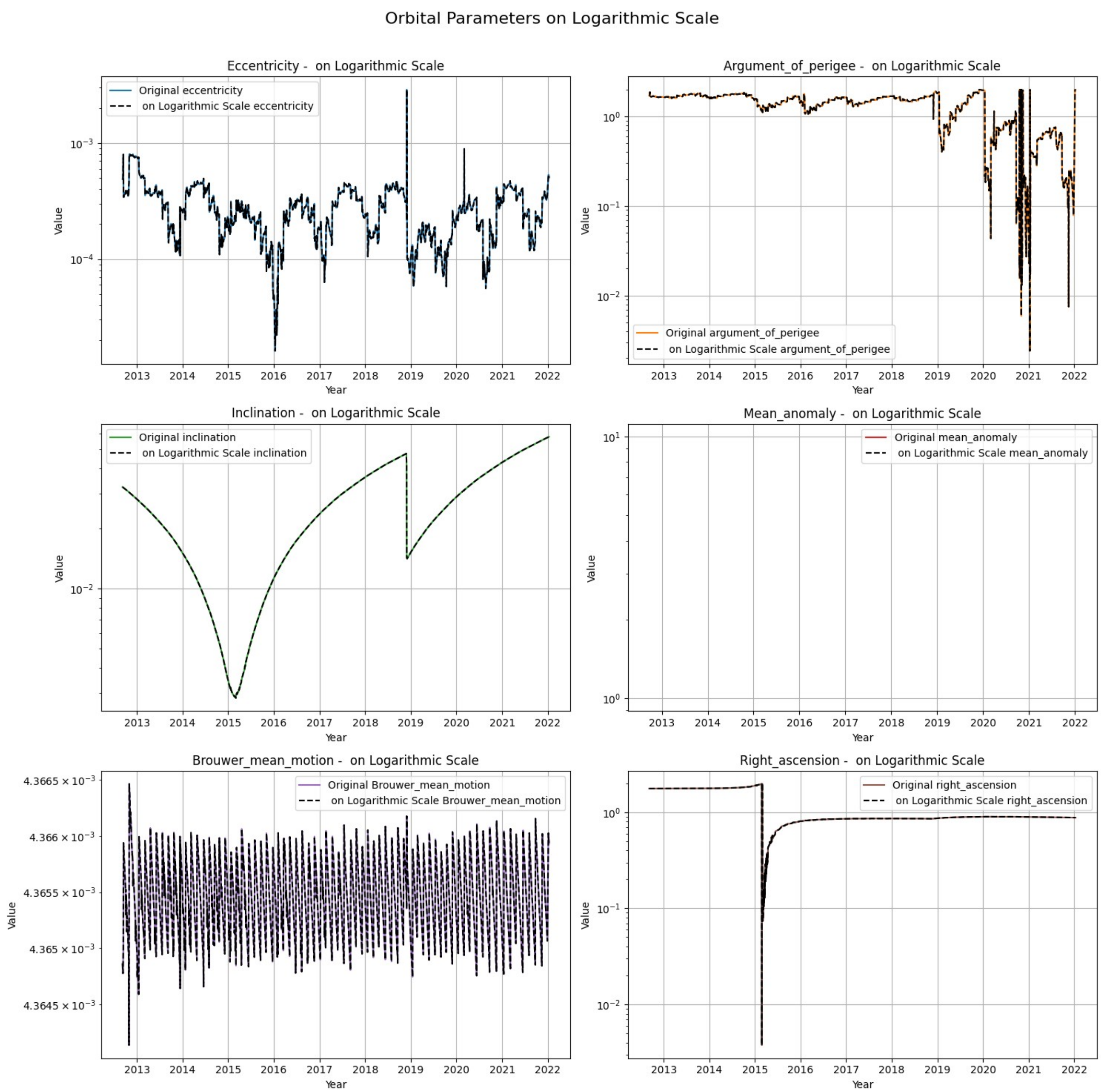
```
Results of Dickey Fuller Test on Right Ascension
Test Statistic                     -2.199394
p-value                             0.206465
#Lags Used                         20.000000
Number of Observations Used      2964.000000
Critical Value (1%)                -3.432558
Critical Value (5%)                -2.862516
Critical Value (10%)               -2.567289
dtype: float64
```

**The ADF statistics are near to critical and the p-value of columns - Inclinations, Mean Anaomoly, Brower Mean Motion and Right ascension is greater than the threshold (0,05) whereas Eccentricity p-value is less than threshold (0,05), Therefore, we can conclude that the time series of column Eccentricity is stationary. Whereas rest of the columns are not stationary hence they will be transformed to Logarithmic Scale.*

Log Tranformation

```
# Define available parameters from the dataset
df_log  = df.copy()
df_log = np.log(df_log +1)

parameters = [
    "eccentricity", "argument_of_perigee", "inclination",
    "mean_anomaly", "Brouwer_mean_motion", "right_ascension"
]

# Define colors for each parameter
colors = [
    "#1f77b4", "#ff7f0e", "#2ca02c", "#d62728",
    "#9467bd", "#8c564b"
]

# Rolling window size
rolling_window = 5

# Compute rolling mean & standard deviation
data_log = {
    col: {
        "log": df_log[col],
    }
    for col in parameters
}

# Create subplots (3 rows, 2 columns) since we have 6 parameters
fig, axes = plt.subplots(3, 2, figsize=(15, 15))
fig.suptitle("Orbital Parameters on Logarithmic Scale", fontsize=16)

# Loop through each parameter and plot it
for i, param in enumerate(parameters):
    row, col = divmod(i, 2)  # Convert index to subplot row/column
```

```
    axes[row, col].plot(df_log.index, df_log[param], color=colors[i],
label=f"Original {param}")
    axes[row, col].plot(df_log.index, data_log[param]["log"],
color="black", linestyle="dashed", label=f" on Logarithmic Scale
{param}")
    #axes[row, col].plot(df_log.index, rolling_data_log[param]["std"],
color="gray", linestyle="dotted", label=f"Rolling Std Dev {param}")

    axes[row, col].set_title(f"{param.capitalize()} -  on Logarithmic
Scale")
    axes[row, col].set_xlabel("Year")
    axes[row, col].set_ylabel("Value")
    axes[row, col].legend(loc="best")
    axes[row, col].grid(True)
    axes[row, col].set_yscale('log')

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.show()
```




```
df_log

                             eccentricity  argument_of_perigee
inclination  \
Datetime

2012-09-06 18:48:32.050655      0.000488             1.701819
0.032409
2012-09-07 19:39:45.383327      0.000487             1.701327
0.032372
2012-09-08 15:43:39.075167      0.000487             1.700215
0.032339
2012-09-09 12:53:36.595967      0.000492             1.701299
```

```
0.032307
2012-09-10 13:15:22.135391      0.000495            1.707099
0.032272
...                                  ...                 ...
...
2021-12-25 04:39:15.108767      0.000330            0.078944
0.057125
2021-12-28 21:00:18.080928      0.000355            0.131605
0.057232
2022-01-06 13:25:24.541247      0.000527            1.975629
0.057655
2022-01-07 18:42:19.086048      0.000532            1.978508
0.057708
2022-01-11 17:26:36.259583      0.000502            1.971985
0.057853

                            mean_anomaly  Brouwer_mean_motion
right_ascension
Datetime

2012-09-06 18:48:32.050655           NaN            0.004365
1.765097
2012-09-07 19:39:45.383327           NaN            0.004365
1.765113
2012-09-08 15:43:39.075167           NaN            0.004365
1.765130
2012-09-09 12:53:36.595967           NaN            0.004365
1.765108
2012-09-10 13:15:22.135391           NaN            0.004365
1.765117
...                                  ...                 ...
...
2021-12-25 04:39:15.108767           NaN            0.004365
0.880964
2021-12-28 21:00:18.080928           NaN            0.004365
0.880645
2022-01-06 13:25:24.541247       -0.3674            0.004366
0.880384
2022-01-07 18:42:19.086048           NaN            0.004366
0.880371
2022-01-11 17:26:36.259583           NaN            0.004366
0.880161

[2985 rows x 6 columns]
```

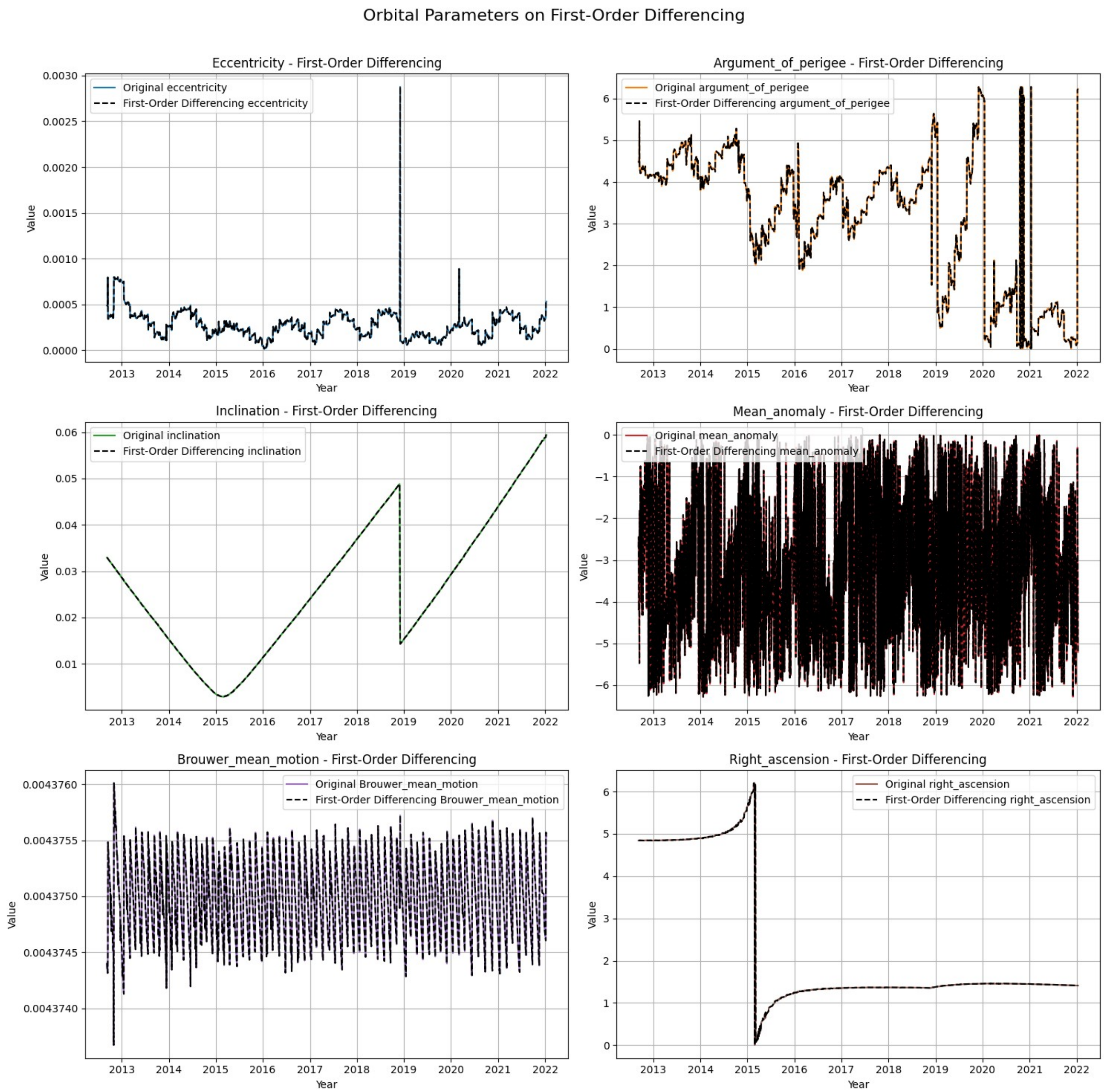
First-Order Differencing

Subtract the previous observation from the current one:

[Y_t' = Y_t - Y_{t-1}]

```
df_shift = df.copy()
df_shift = df_shift.shift(1)

parameters = [
    "eccentricity", "argument_of_perigee", "inclination",
    "mean_anomaly", "Brouwer_mean_motion", "right_ascension"
]

# Define colors for each parameter
colors = [
    "#1f77b4", "#ff7f0e", "#2ca02c", "#d62728",
    "#9467bd", "#8c564b"
]

# Rolling window size
rolling_window = 5

# Compute rolling mean & standard deviation
data_shift = {
    col: {
        "first order": df_shift[col]
    }
    for col in parameters
}

# Create subplots (3 rows, 2 columns) since we have 6 parameters
fig, axes = plt.subplots(3, 2, figsize=(15, 15))
fig.suptitle("Orbital Parameters on First-Order Differencing",
fontsize=16)

# Loop through each parameter and plot it
for i, param in enumerate(parameters):
    row, col = divmod(i, 2)  # Convert index to subplot row/column

    axes[row, col].plot(df_shift.index, df_shift[param],
color=colors[i], label=f"Original {param}")
    axes[row, col].plot(df_shift.index, data_shift[param]["first
order"], color="black", linestyle="dashed", label=f"First-Order
Differencing {param}")
    #axes[row, col].plot(df_shift.index, rolling_data_shift[param]
["std"], color="gray", linestyle="dotted", label=f"Rolling Std Dev
{param}")

    axes[row, col].set_title(f"{param.capitalize()} - First-Order
Differencing")
    axes[row, col].set_xlabel("Year")
    axes[row, col].set_ylabel("Value")
    axes[row, col].legend(loc="best")
    axes[row, col].grid(True)
```

```
# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.show()
```



Boxcox and Yeojohnson Transformation

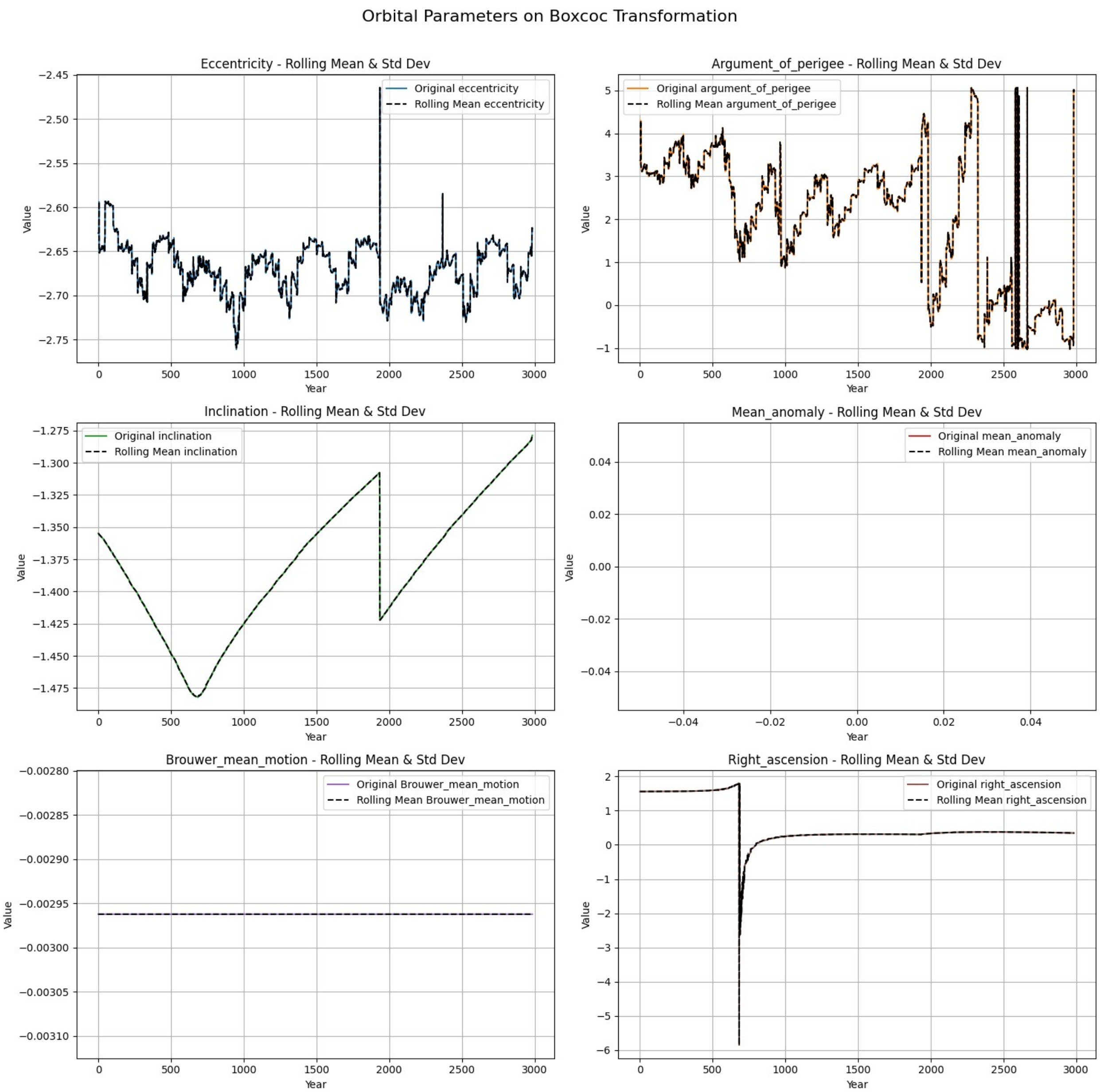
```
from scipy.stats import boxcox, yeojohnson
# Select numerical columns
numerical_cols = df.select_dtypes(include=[np.number]).columns

# Apply Box-Cox transformation (only for positive values)
boxcox_transformed = {}
```

```
for col in numerical_cols:
    if (df[col] > 0).all():  # Box-Cox requires strictly positive
values
        boxcox_transformed[col], _ = boxcox(df[col])
    else:
        print(f"Skipping Box-Cox for {col} (contains non-positive
values)")

# Apply Yeo-Johnson transformation (works for both positive & negative
values)
yeojohnson_transformed = {}
for col in numerical_cols:
    yeojohnson_transformed[col], _ = yeojohnson(df[col])

# Convert transformed data back to DataFrame
df_boxcox = pd.DataFrame(boxcox_transformed)
df_yeojohnson = pd.DataFrame(yeojohnson_transformed)
if 'mean_anomaly' in df.columns:
    df_boxcox['mean_anomaly'] = df['mean_anomaly']
    df_yeojohnson['mean_anomaly'] = df['mean_anomaly']
# Display the transformed data
print("Box-Cox Transformed Data:\n", df_boxcox.head())
print("Yeo-Johnson Transformed Data:\n", df_yeojohnson.head())
```

```
Skipping Box-Cox for mean_anomaly (contains non-positive values)
Box-Cox Transformed Data:
    eccentricity  argument_of_perigee  inclination
Brouwer_mean_motion  \
0     -2.629296             3.372852    -1.354793            -0.002962

1     -2.629460             3.370293    -1.354916            -0.002962

2     -2.629487             3.364510    -1.355021            -0.002962

3     -2.628804             3.370149    -1.355127            -0.002962

4     -2.628383             3.400399    -1.355244            -0.002962


   right_ascension  mean_anomaly
0         1.556583           NaN
1         1.556602           NaN
2         1.556622           NaN
3         1.556596           NaN
4         1.556607           NaN
Yeo-Johnson Transformed Data:
    eccentricity  argument_of_perigee  inclination  mean_anomaly  \
0      0.000298             6.154941     0.029262           NaN
1      0.000298             6.150560     0.029231           NaN
2      0.000297             6.140663     0.029205           NaN
```

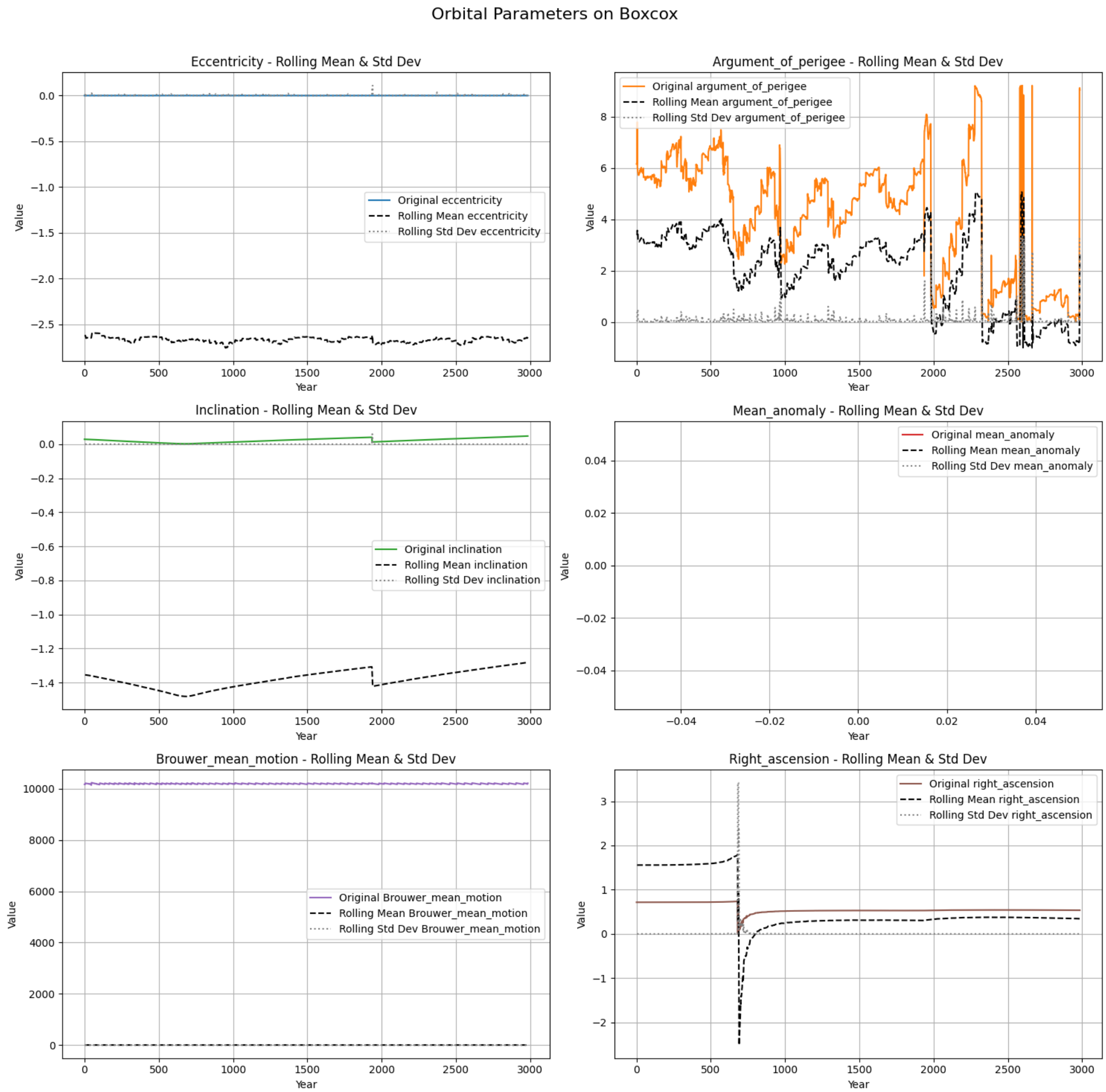
```
3      0.000299             6.150313     0.029179           NaN
4      0.000300             6.202149     0.029150           NaN

   Brouwer_mean_motion  right_ascension
0         10174.765551         0.716108
1         10173.809045         0.716109
2         10173.017643         0.716111
3         10172.162432         0.716109
4         10171.216816         0.716110

parameters = [
    "eccentricity", "argument_of_perigee", "inclination",
    "mean_anomaly", "Brouwer_mean_motion", "right_ascension"
]

# Define colors for each parameter
colors = [
    "#1f77b4", "#ff7f0e", "#2ca02c", "#d62728",
    "#9467bd", "#8c564b"
]

# Rolling window size
rolling_window = 5

# Compute rolling mean & standard deviation
data_boxcox = {
    col: {
        "boxcox": df_boxcox[col]
    }
    for col in parameters
}

# Create subplots (3 rows, 2 columns) since we have 6 parameters
fig, axes = plt.subplots(3, 2, figsize=(15, 15))
fig.suptitle("Orbital Parameters on Boxcoc Transformation",
fontsize=16)

# Loop through each parameter and plot it
for i, param in enumerate(parameters):
    row, col = divmod(i, 2)  # Convert index to subplot row/column

    axes[row, col].plot(df_boxcox.index, df_boxcox[param],
color=colors[i], label=f"Original {param}")
    axes[row, col].plot(df_boxcox.index, data_boxcox[param]["boxcox"],
color="black", linestyle="dashed", label=f"Rolling Mean {param}")

    axes[row, col].set_title(f"{param.capitalize()} - Rolling Mean &
Std Dev")
    axes[row, col].set_xlabel("Year")
    axes[row, col].set_ylabel("Value")
```

```python
    axes[row, col].legend(loc="best")
    axes[row, col].grid(True)

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.show()
```




```python
parameters = [
    "eccentricity", "argument_of_perigee", "inclination",
    "mean_anomaly", "Brouwer_mean_motion", "right_ascension"
]
```

```python
# Define colors for each parameter
colors = [
    "#1f77b4", "#ff7f0e", "#2ca02c", "#d62728",
    "#9467bd", "#8c564b"
]

# Rolling window size
rolling_window = 5

# Compute rolling mean & standard deviation
rolling_data_shift = {
    col: {
        "mean": df_boxcox[col].rolling(rolling_window).mean(),
        "std": df_boxcox[col].rolling(rolling_window).std()
    }
    for col in parameters
}

# Create subplots (3 rows, 2 columns) since we have 6 parameters
fig, axes = plt.subplots(3, 2, figsize=(15, 15))
fig.suptitle("Orbital Parameters on Boxcox", fontsize=16)

# Loop through each parameter and plot it
for i, param in enumerate(parameters):
    row, col = divmod(i, 2)  # Convert index to subplot row/column

    axes[row, col].plot(df_yeojohnson.index, df_yeojohnson[param],
color=colors[i], label=f"Original {param}")
    axes[row, col].plot(df_yeojohnson.index, rolling_data_shift[param]
["mean"], color="black", linestyle="dashed", label=f"Rolling Mean
{param}")
    axes[row, col].plot(df_yeojohnson.index, rolling_data_shift[param]
["std"], color="gray", linestyle="dotted", label=f"Rolling Std Dev
{param}")

    axes[row, col].set_title(f"{param.capitalize()} - Rolling Mean &
Std Dev")
    axes[row, col].set_xlabel("Year")
    axes[row, col].set_ylabel("Value")
    axes[row, col].legend(loc="best")
    axes[row, col].grid(True)

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.show()
```




```python
# Choose a seasonal lag (e.g., 12 for yearly seasonality in monthly data)
seasonal_lag = 1

# Apply seasonal differencing to all numeric columns
df_diff = df.copy()
numeric_cols = df.select_dtypes(include=['number']).columns
df_diff[numeric_cols] = df[numeric_cols].diff(seasonal_lag)

# Display the first few rows
print(df_diff.head())
```

```
                              eccentricity  argument_of_perigee  
inclination  \
Datetime                                                          

2012-09-06 18:48:32.050655           NaN                  NaN          
NaN   
2012-09-07 19:39:45.383327 -1.200000e-06            -0.002697     -
0.000038   
2012-09-08 15:43:39.075167 -2.000000e-07            -0.006093     -
0.000033   
2012-09-09 12:53:36.595967  5.000000e-06             0.005941     -
0.000033   
2012-09-10 13:15:22.135391  3.100000e-06             0.031880     -
0.000037   

                            mean_anomaly  Brouwer_mean_motion  
right_ascension  
Datetime                                                          

2012-09-06 18:48:32.050655           NaN                  NaN          
NaN  
2012-09-07 19:39:45.383327      0.243002        -2.351689e-08         
0.000096  
2012-09-08 15:43:39.075167     -1.010600        -1.945927e-08         
0.000098  
2012-09-09 12:53:36.595967     -0.733457        -2.102993e-08        -
0.000131  
2012-09-10 13:15:22.135391      0.079435        -2.325509e-08         
0.000056  
```

Train Test split

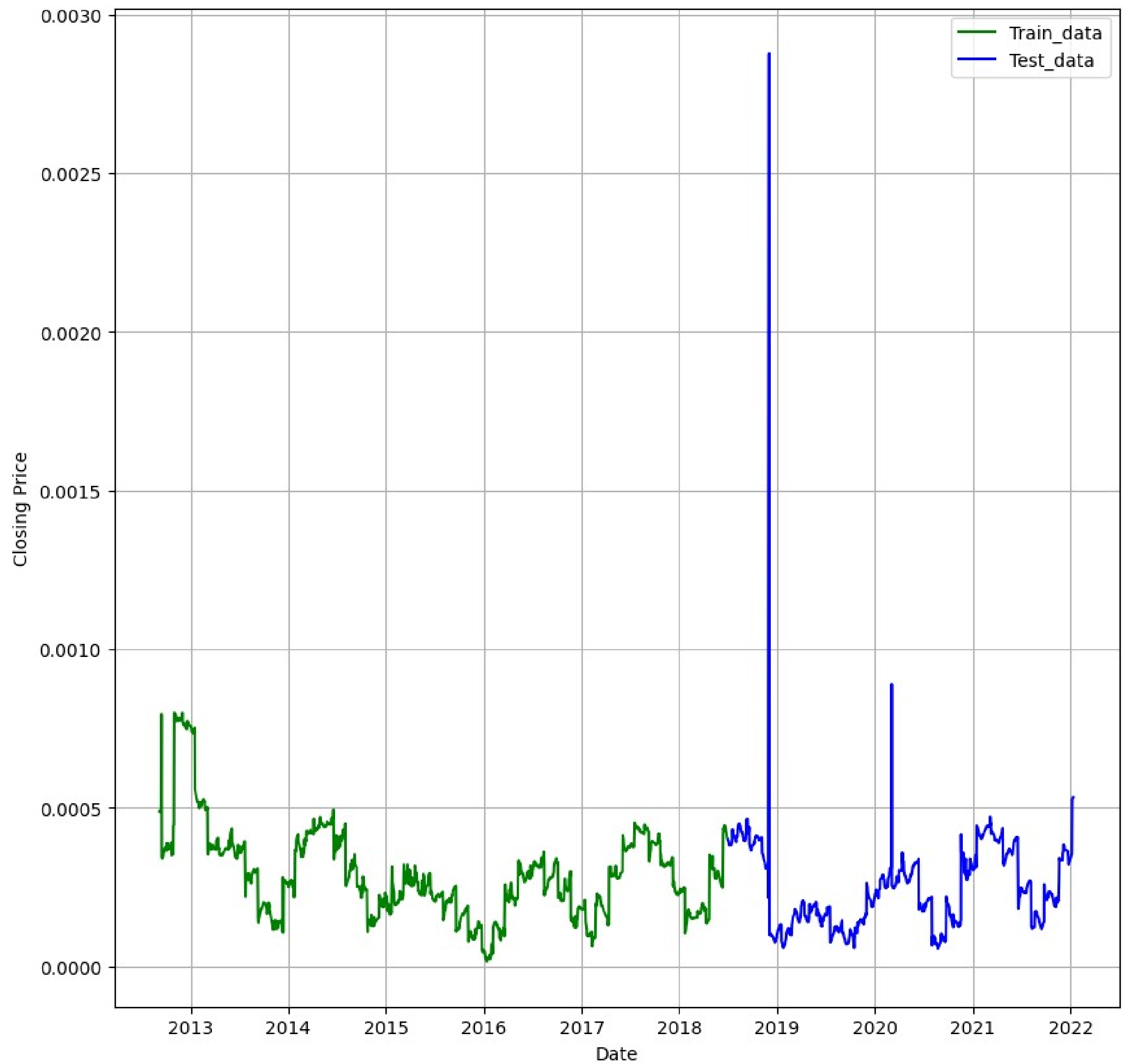
```python
#Train Test split
to_row = int(len(df_shift)*0.60)
print(to_row)
training_data=list(df_shift[:to_row]["eccentricity"])
training_data
testing_data=df_shift[to_row:]["eccentricity"]
print(len(training_data),len(testing_data))
print(testing_data)
```

```
1791
1791 1194
Datetime
2018-07-01 03:45:55.600991    0.000399
2018-07-02 02:52:23.012256    0.000399
2018-07-03 05:46:27.939360    0.000397
2018-07-04 14:00:11.167775    0.000396
2018-07-05 14:23:50.142623    0.000387
```

```
                                ...
2021-12-25 04:39:15.108767    0.000321
2021-12-28 21:00:18.080928    0.000330
2022-01-06 13:25:24.541247    0.000355
2022-01-07 18:42:19.086048    0.000527
2022-01-11 17:26:36.259583    0.000533
Name: eccentricity, Length: 1194, dtype: float64
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))
plt.grid(True)
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.plot(df_shift[:to_row]["eccentricity"],'green',label='Train_data')
plt.plot(df_shift[to_row:]["eccentricity"],'blue',label='Test_data')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1ba0223f220>
```



ARIMA

We are training a ARIMA model with satellite eccentricity data

```
model_predictions = []
training_data = list(training_data)  # Convert to a list to allow
appending

for i in range(len(testing_data)):
    # Fit ARIMA model on the current training set
    model = ARIMA(training_data, order=(2,1,0))
    model_fit = model.fit()
```

```
    # Generate forecast
    yhat = model_fit.forecast()[0]  # Extract the predicted value
    model_predictions.append(yhat)

    # Append the actual test value to training data for the next
iteration
    training_data.append(testing_data[i])

# Convert predictions back to NumPy array for further analysis if
needed
print(len(model_predictions))
#model_predictions = np.array(model_predictions)
```

```
1194
```

```
print(model_fit.summary())
```

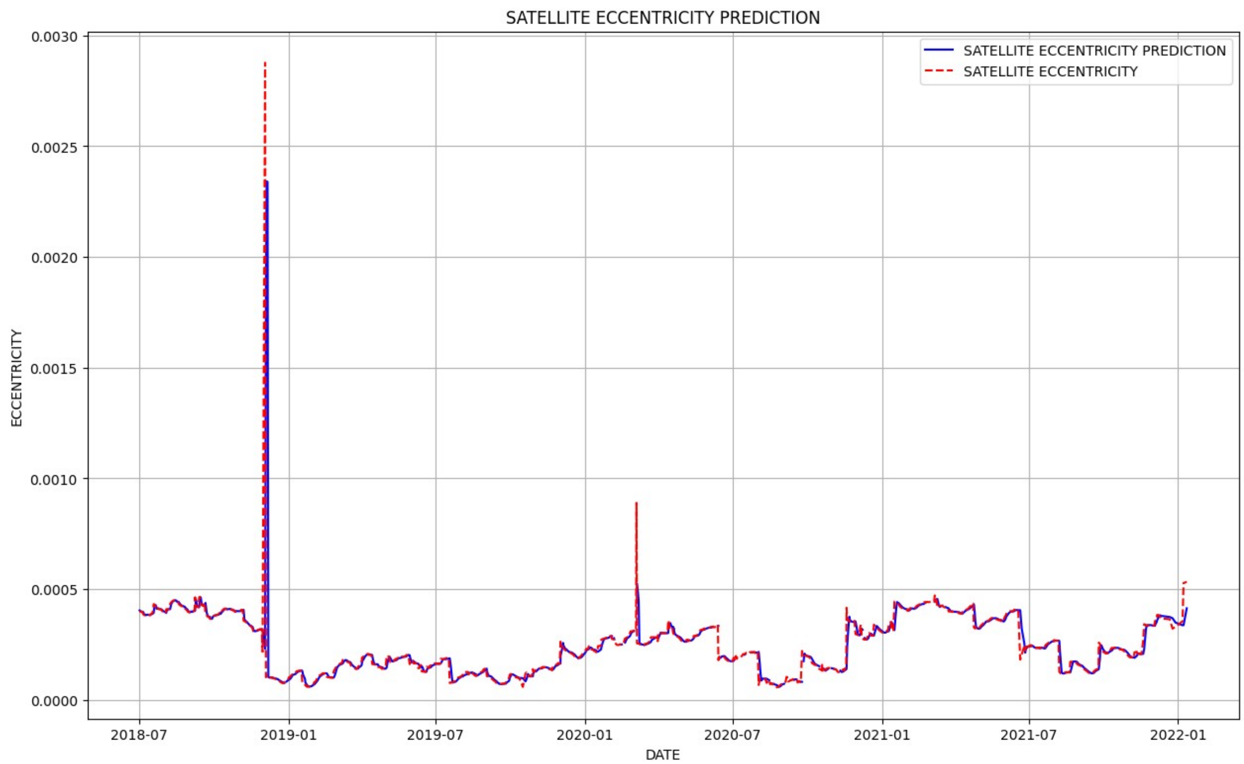
```
                               SARIMAX Results

======================================================================
========
Dep. Variable:                      y   No. Observations:
2984
Model:                 ARIMA(2, 1, 0)   Log Likelihood
24570.959
Date:                Wed, 09 Apr 2025   AIC                          -
49135.918
Time:                        10:49:54   BIC                          -
49117.916
Sample:                             0   HQIC                         -
49129.440
                               - 2984

Covariance Type:                  opg

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1         -0.6196   3.44e-21   -1.8e+20      0.000      -0.620
-0.620
ar.L2         -0.2824   7.39e-22  -3.82e+20      0.000      -0.282
-0.282
sigma2      4.049e-09   5.33e-12    759.634      0.000    4.04e-09
4.06e-09
======================================================================
=============
Ljung-Box (L1) (Q):                   7.69   Jarque-Bera (JB):
```

```
121964153.55
Prob(Q):                              0.01   Prob(JB):
0.00
Heteroskedasticity (H):               1.00   Skew:
19.23
Prob(H) (two-sided):                  0.94   Kurtosis:
992.85
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
[2] Covariance matrix is singular or near-singular, with condition
number    inf. Standard errors may be unstable.
```

```
print(len(model_predictions))
```

```
1194
```

```
plt.figure(figsize=(15,9))
plt.grid(True)

date_range=df_shift[to_row:]["eccentricity"].index

#print(date_range)
#print(model_predictions)

plt.plot(date_range, model_predictions[:], color='blue',
linestyle='-', label="SATELLITE ECCENTRICITY PREDICTION")
plt.plot(date_range, testing_data, color='red', linestyle='dashed',
label="SATELLITE ECCENTRICITY")

plt.title("SATELLITE ECCENTRICITY PREDICTION")
plt.xlabel("DATE")
plt.ylabel("ECCENTRICITY")
plt.legend()
plt.show()
```



SARIMAX

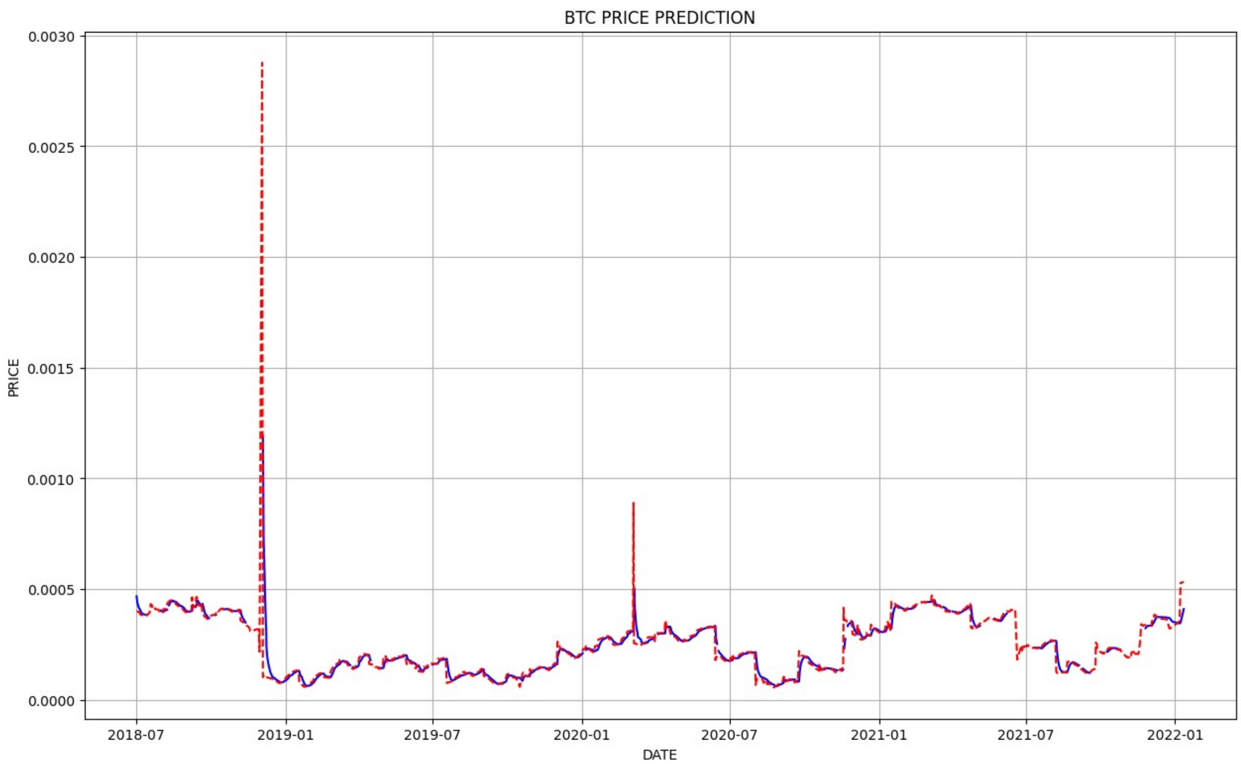
We are training a SARIMAX model with satellite eccentricity data

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
model=SARIMAX(training_data, order=(0,1,1))
model_fit=model.fit()
output= model_fit.forecast()
print(output[0])
```

```
0.00046713440379356273
```

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

model_prediction=[]
for i in range(len(testing_data)):
  model=SARIMAX(training_data, order=(0,1,1))
  model_fit=model.fit()
  output= model_fit.forecast()
  yhat=((output[0]))
  model_prediction.append(yhat)
  actual_test_value=testing_data[i]
  #print(actual_test_value)
  training_data.append(actual_test_value)
```

```
                               SARIMAX Results                                
==============================================================================
========
Dep. Variable:                      y   No. Observations:                 
4178
Model:               SARIMAX(0, 1, 1)   Log Likelihood                 
33938.871
Date:                Wed, 09 Apr 2025   AIC                            -
67873.743
Time:                        11:10:02   BIC                            -
67861.068
Sample:                             0   HQIC                           -
67869.260
                                - 4178                                         

Covariance Type:                  opg                                         

==============================================================================
========
                 coef    std err          z      P>|z|      [0.025      
0.975]
------------------------------------------------------------------------------
--------
ma.L1         -0.7021   1.62e-20  -4.34e+19      0.000      -0.702      
-0.702
sigma2      4.709e-09   5.37e-12    876.464      0.000     4.7e-09    
4.72e-09
==============================================================================
=============
Ljung-Box (L1) (Q):                  38.20   Jarque-Bera (JB):          
131202035.13
Prob(Q):                              0.00   Prob(JB):                        
0.00
Heteroskedasticity (H):               7.36   Skew:                           
21.08
Prob(H) (two-sided):                  0.00   Kurtosis:                      
870.22
==============================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients 
(complex-step).
[2] Covariance matrix is singular or near-singular, with condition 
number 8.15e+23. Standard errors may be unstable.
```




```
print(model_fit.summary())

plt.figure(figsize=(15,9))
plt.grid(True)

date_range=df[to_row:].index

plt.plot(date_range, model_prediction[:], color='blue', linestyle='-',
label="SATELLITE ECCENTRICITY PREDICTION")
plt.plot(date_range, testing_data, color='red', linestyle='dashed',
label="SATELLITE ECCENTRICITY PREDICTION")

plt.title("BTC PRICE PREDICTION")
plt.xlabel("DATE")
plt.ylabel("PRICE")
plt.show()

!pip install pmdarima

!pip list

#!pip uninstall pmdarima numpy -y
#!pip install numpy pmdarima
#from pmdarima import auto_arima
# step_wise=auto_arima(
#     training_data,
#     start_p=1,
```

```
#     start_q=1,
#     max_p=7,
#     max_q=7,
#     d=1,
#     max_d=7,
#     trace=True,
#     m=12,
#     error_action='ignore',
#     suppress_warnings=True,
#     stepwise=True
# )
```