In [1]:
```
import pandas_datareader as pdr
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
```

In [2]:
```
df = pd.read_csv('goldstock.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Unnamed: 0	Date	Close	Volume	Open	High	Low
0	0	2024-01-19	2029.3	166078.0	2027.4	2041.9	2022.2
1	1	2024-01-18	2021.6	167013.0	2009.1	2025.6	2007.7
2	2	2024-01-17	2006.5	245194.0	2031.7	2036.1	2004.6
3	3	2024-01-16	2030.2	277995.0	2053.4	2062.8	2027.6
4	4	2024-01-12	2051.6	250946.0	2033.2	2067.3	2033.1

In [4]:
```
df.tail()
```

Out[4]:

	Unnamed: 0	Date	Close	Volume	Open	High	Low
2506	2528	2014-01-28	1250.5	81426.0	1254.9	1261.9	1248.0
2507	2529	2014-01-27	1263.5	63419.0	1269.9	1280.1	1252.0
2508	2530	2014-01-24	1264.5	34998.0	1264.3	1273.2	1256.9
2509	2531	2014-01-23	1262.5	41697.0	1235.1	1267.1	1230.8
2510	2532	2014-01-22	1238.6	80262.0	1240.5	1243.5	1235.5

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Unnamed: 0    0
Date          0
Close         0
Volume        0
Open          0
High          0
Low           0
dtype: int64
```

In [6]:
```
df.dtypes
```

Out[6]:
```
Unnamed: 0      int64
Date           object
Close         float64
Volume        float64
Open          float64
High          float64
Low           float64
dtype: object
```

In [7]:
```
df.describe()
```

Out[7]:

	Unnamed: 0	Close	Volume	Open	High	Low
count	2511.000000	2511.000000	2511.000000	2511.000000	2511.000000	2511.000000
mean	1260.792911	1498.726085	185970.770609	1498.725528	1508.451454	1488.869932
std	729.262879	298.824811	97600.769382	299.118187	301.262244	296.417703
min	0.000000	1049.600000	1.000000	1051.500000	1062.700000	1045.400000
25%	630.500000	1249.850000	126693.500000	1249.500000	1257.300000	1242.350000
50%	1259.000000	1332.800000	175421.000000	1334.000000	1342.400000	1326.600000
75%	1888.500000	1805.850000	234832.000000	1805.600000	1815.450000	1793.050000
max	2532.000000	2093.100000	787217.000000	2094.400000	2098.200000	2074.600000

In [8]:
```
df.columns
```

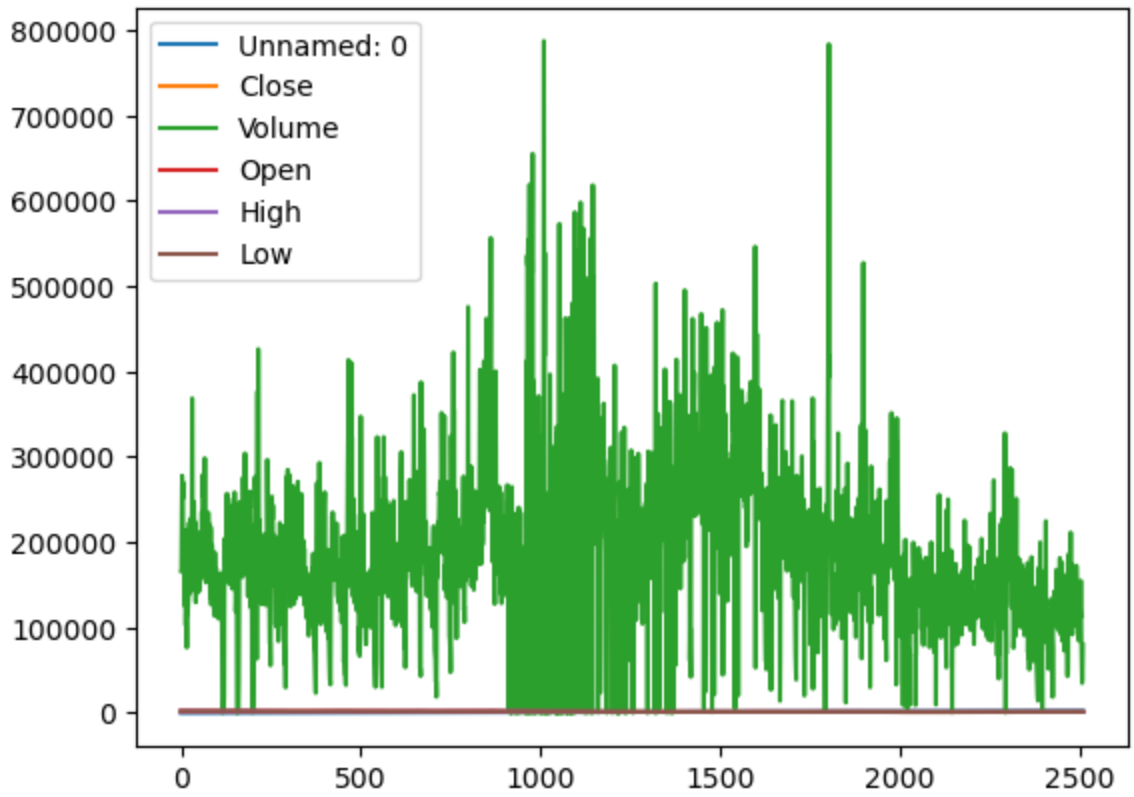
Out[8]:
```
Index(['Unnamed: 0', 'Date', 'Close', 'Volume', 'Open', 'High', 'Low'], dtype='object')
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2511 entries, 0 to 2510
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  2511 non-null   int64
 1   Date        2511 non-null   object
 2   Close       2511 non-null   float64
 3   Volume      2511 non-null   float64
 4   Open        2511 non-null   float64
 5   High        2511 non-null   float64
 6   Low         2511 non-null   float64
dtypes: float64(5), int64(1), object(1)
memory usage: 137.4+ KB
```

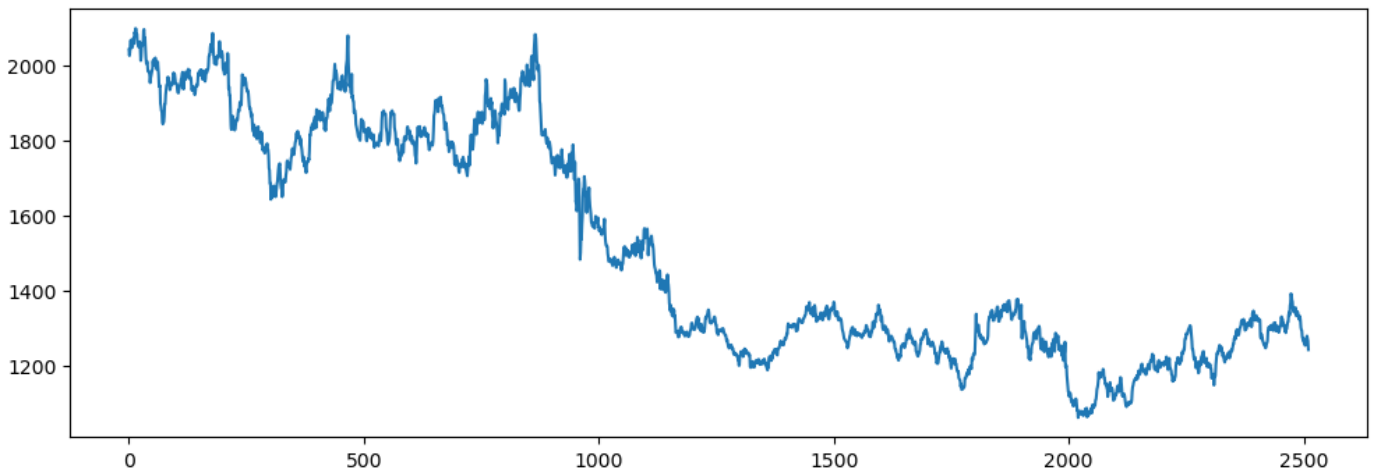
In [10]:
```
df.plot()
```

Out[10]:
```
<Axes: >
```



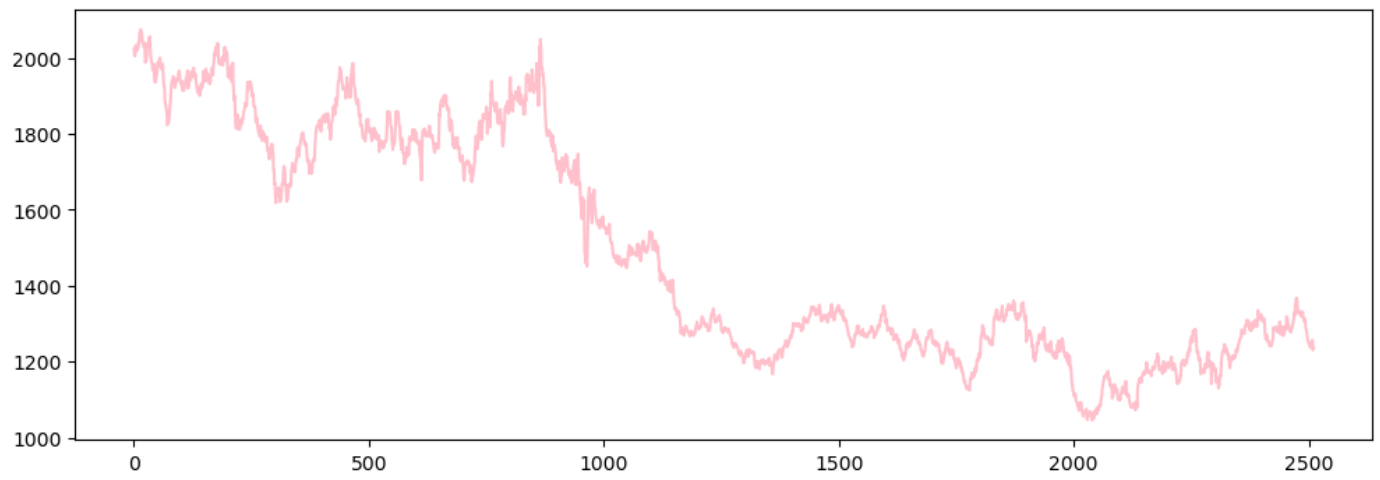
In [11]:
```
df['High'].plot(figsize=(12,4))
```

Out[11]:
```
<Axes: >
```



In [12]:
```
df['Low'].plot(figsize=(12,4), c='pink')
```

Out[12]:
```
<Axes: >
```



In [13]:
```
df['Open'].plot(figsize=(12,4),c = 'green')
```

Out[13]:
```
<Axes: >
```

In [14]:
```
df = df.drop(columns=['Unnamed: 0'])
```

In [15]:
```
df.head()
```

Out[15]:

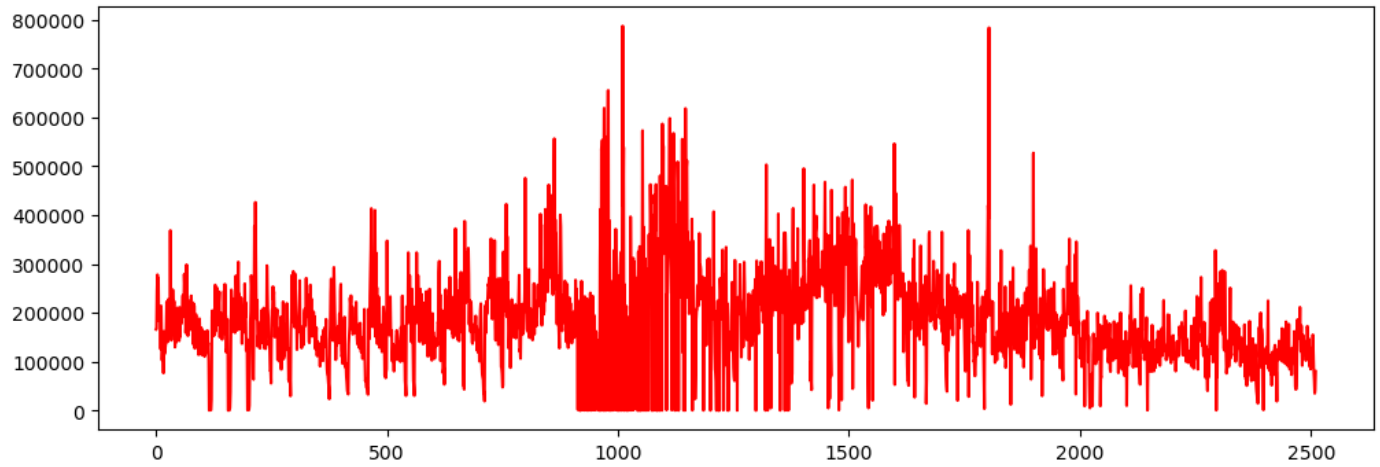
	Date	Close	Volume	Open	High	Low
0	2024-01-19	2029.3	166078.0	2027.4	2041.9	2022.2
1	2024-01-18	2021.6	167013.0	2009.1	2025.6	2007.7
2	2024-01-17	2006.5	245194.0	2031.7	2036.1	2004.6
3	2024-01-16	2030.2	277995.0	2053.4	2062.8	2027.6
4	2024-01-12	2051.6	250946.0	2033.2	2067.3	2033.1

In [16]:
```
df['Close'].plot(figsize=(12,4),c = 'red')
```

Out[16]:
```
<Axes: >
```

In [17]:
```
df['Volume'].plot(figsize=(12,4), c = 'red')
```

Out[17]:
```
<Axes: >
```



In [18]:
```
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date',inplace=True)
```

In [19]:
```
df
```

Out[19]:

Date	Close	Volume	Open	High	Low
2024-01-19	2029.3	166078.0	2027.4	2041.9	2022.2
2024-01-18	2021.6	167013.0	2009.1	2025.6	2007.7
2024-01-17	2006.5	245194.0	2031.7	2036.1	2004.6
2024-01-16	2030.2	277995.0	2053.4	2062.8	2027.6
2024-01-12	2051.6	250946.0	2033.2	2067.3	2033.1
...	...	...	...	...	...
2014-01-28	1250.5	81426.0	1254.9	1261.9	1248.0
2014-01-27	1263.5	63419.0	1269.9	1280.1	1252.0
2014-01-24	1264.5	34998.0	1264.3	1273.2	1256.9
2014-01-23	1262.5	41697.0	1235.1	1267.1	1230.8
2014-01-22	1238.6	80262.0	1240.5	1243.5	1235.5

2511 rows × 5 columns

In [20]:
```
data = df.filter(['Close'])
data
```

Out[20]:

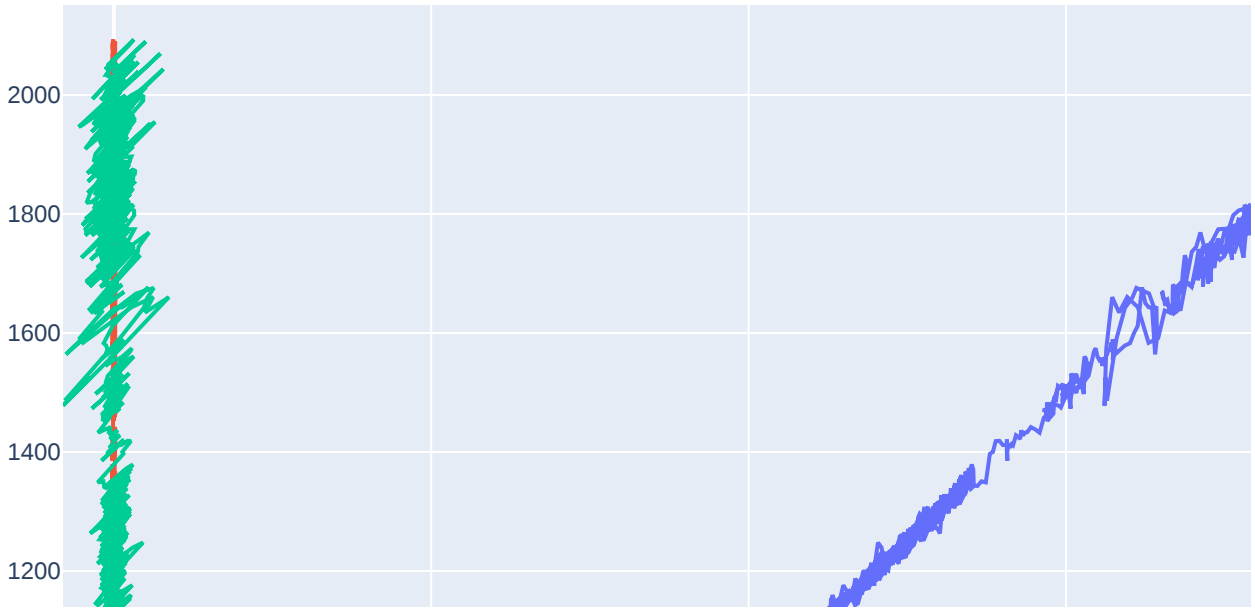
Date	Close
2024-01-19	2029.3
2024-01-18	2021.6
2024-01-17	2006.5
2024-01-16	2030.2
2024-01-12	2051.6
...	...
2014-01-28	1250.5
2014-01-27	1263.5
2014-01-24	1264.5
2014-01-23	1262.5
2014-01-22	1238.6

2511 rows × 1 columns

In [21]:
```
import statsmodels.api as sm
```

In [22]:
```
import plotly.graph_objects as go
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [23]:
```
result = seasonal_decompose(data, period=12)
fig = go.Figure()
fig.add_trace(go.Scatter(x=result.trend, y=result.observed, name="Trend"))
fig.add_trace(go.Scatter(x=result.seasonal, y=result.observed, name="Seasonal"))
fig.add_trace(go.Scatter(x=result.resid, y=result.observed, name="Residual"))
fig.show()
```



In [24]:
```
result = sm.tsa.stattools.adfuller(data['Close'])
print(f'Test statistic: {result[0]}')
print(f'p-value: {result[1]}')
```

```
Test statistic: -1.717275578970478
p-value: 0.4222342775667282
```

In [25]:
```
data['Close_diff'] = data['Close'].diff()
```

In [26]:
```
data = data.dropna()
```

In [27]:
```
result_diff = sm.tsa.stattools.adfuller(data['Close_diff'])
print(f'Test statistic: {result_diff[0]}')
print(f'p-value: {result_diff[1]}')
```

```
Test statistic: -51.58288745483633
p-value: 0.0
```

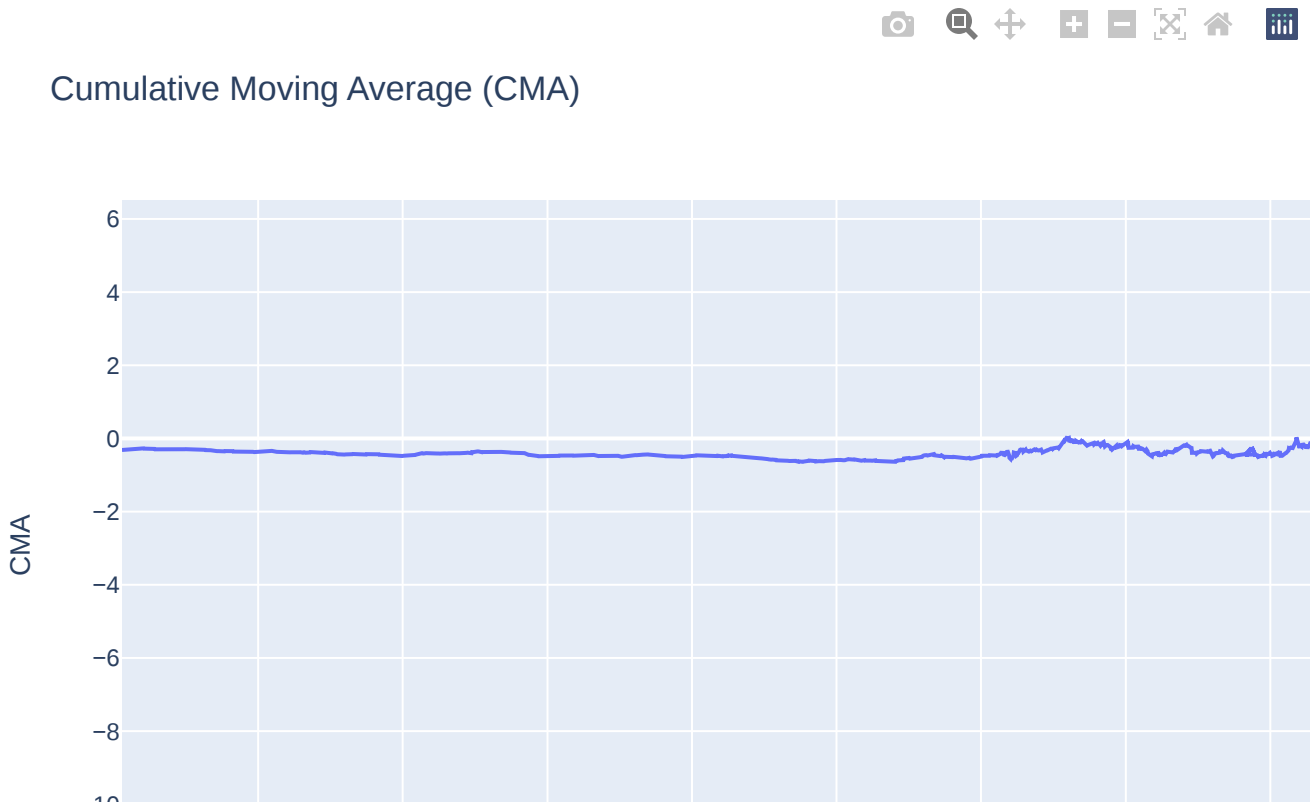
In [28]:
```
sma_window = 10
data['SMA'] = data['Close_diff'].rolling(window=sma_window).mean()

data['CMA'] = data['Close_diff'].expanding().mean()

ema_window = 10
data['EMA'] = data['Close_diff'].ewm(span=ema_window, adjust=False).mean()
```

```
C:\Users\Acer\AppData\Local\Temp\ipykernel_15512\3022869577.py:2: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Acer\AppData\Local\Temp\ipykernel_15512\3022869577.py:4: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Acer\AppData\Local\Temp\ipykernel_15512\3022869577.py:7: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
```

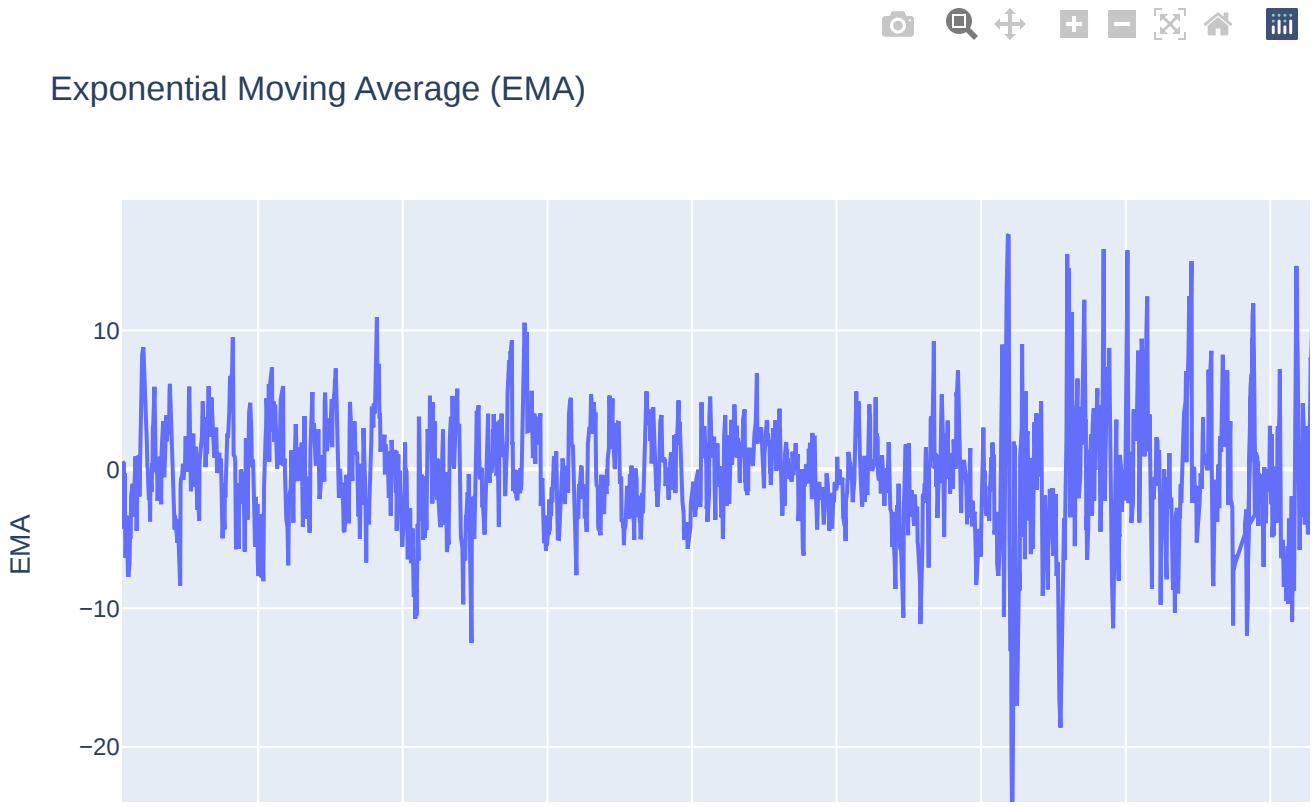
In [29]:
```
import plotly.graph_objects as go
```

In [30]:
```
fig_sma = go.Figure()
fig_sma.add_trace(go.Scatter(x=data.index, y=data['SMA'], mode='lines', name='SMA'))
fig_sma.update_layout(title='Simple Moving Average (SMA)', xaxis_title='Date', yaxis_tit
```

In [31]:
```
fig_cma = go.Figure()
fig_cma.add_trace(go.Scatter(x=data.index, y=data['CMA'], mode='lines', name='CMA'))
fig_cma.update_layout(title='Cumulative Moving Average (CMA)', xaxis_title='Date', yaxis
```

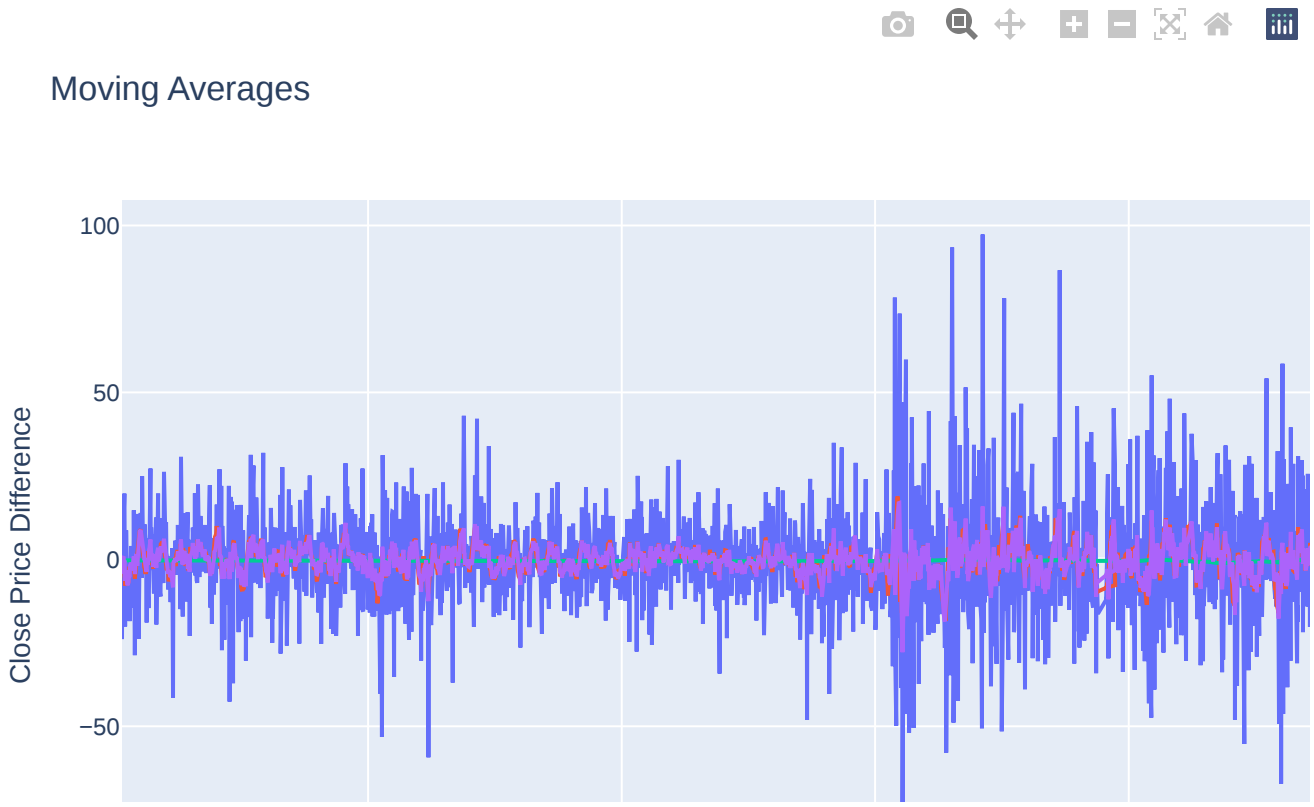


In [32]:
```
fig_ema = go.Figure()
fig_ema.add_trace(go.Scatter(x=data.index, y=data['EMA'], mode='lines', name='EMA'))
fig_ema.update_layout(title='Exponential Moving Average (EMA)', xaxis_title='Date', yaxi
```



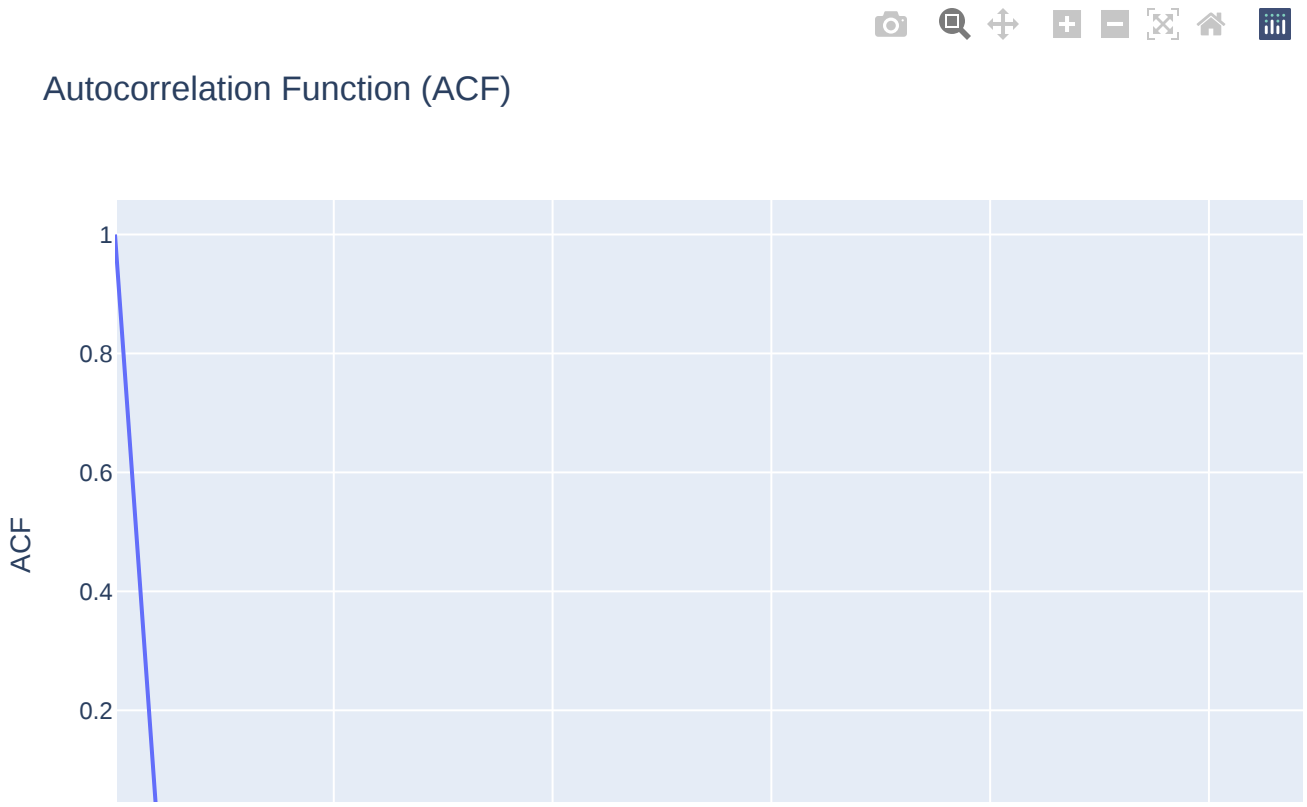
In [33]:

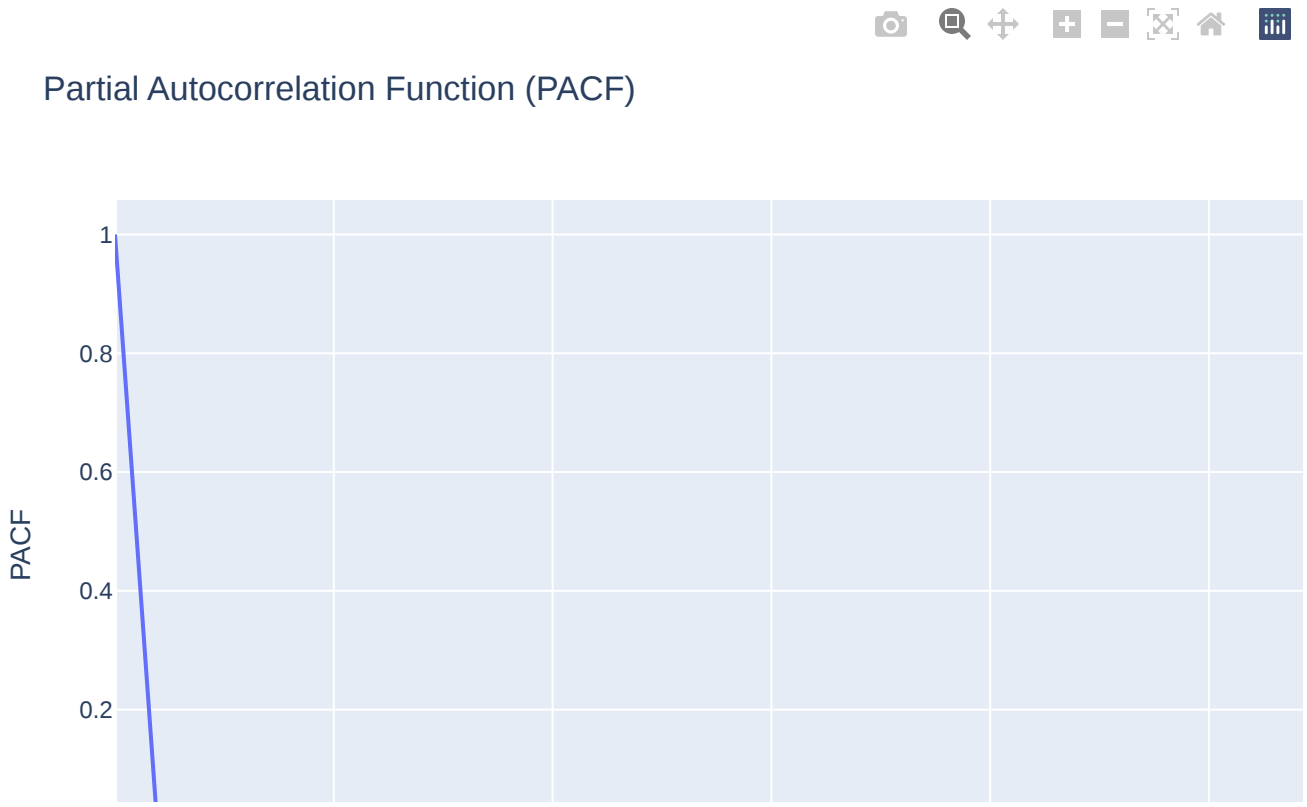
```
fig = go.Figure()

fig.add_trace(
    go.Scatter(x=data.index, y=data['Close_diff'], mode='lines', name='Differenced Serie
)

fig.add_trace(
    go.Scatter(x=data.index, y=data['SMA'], mode='lines', name='SMA')
)

fig.add_trace(
    go.Scatter(x=data.index, y=data['CMA'], mode='lines', name='CMA')
)

fig.add_trace(
    go.Scatter(x=data.index, y=data['EMA'], mode='lines', name='EMA')
)

fig.update_layout(title='Moving Averages', xaxis_title='Date', yaxis_title='Close Price

fig.show()
```



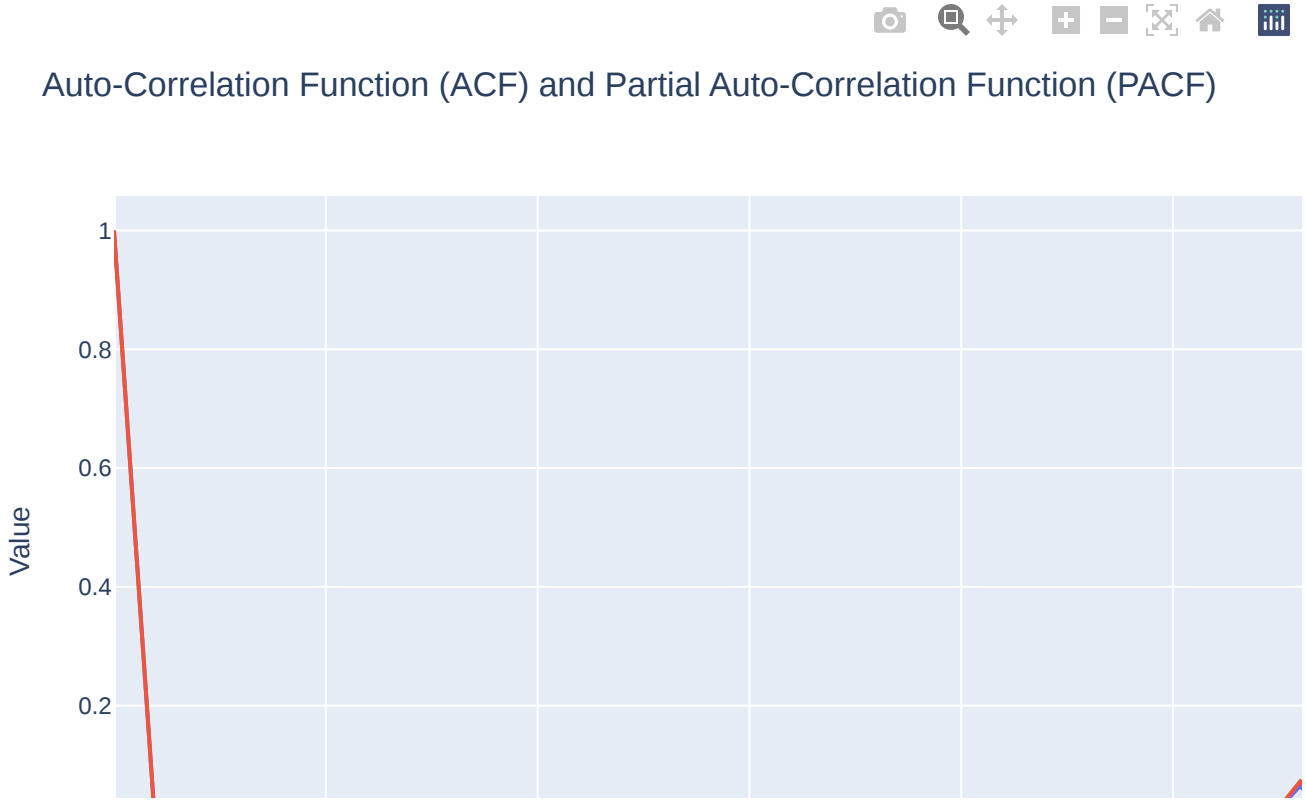
In [34]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

In [35]:
```
# Assuming the differenced series is stored in a DataFrame named "data" with column "Clo
# Compute ACF
acf_result = sm.tsa.stattools.acf(data['Close_diff'])

# Compute PACF
pacf_result = sm.tsa.stattools.pacf(data['Close_diff'])

# Create a figure for ACF
fig_acf = go.Figure(data=go.Scatter(x=list(range(len(acf_result))), y=acf_result, mode='
fig_acf.update_layout(title='Autocorrelation Function (ACF)', xaxis_title='Lag', yaxis_t

# Create a figure for PACF
fig_pacf = go.Figure(data=go.Scatter(x=list(range(len(pacf_result))), y=pacf_result, mod
fig_pacf.update_layout(title='Partial Autocorrelation Function (PACF)', xaxis_title='Lag

# Show the plots
fig_acf.show()
fig_pacf.show()
```





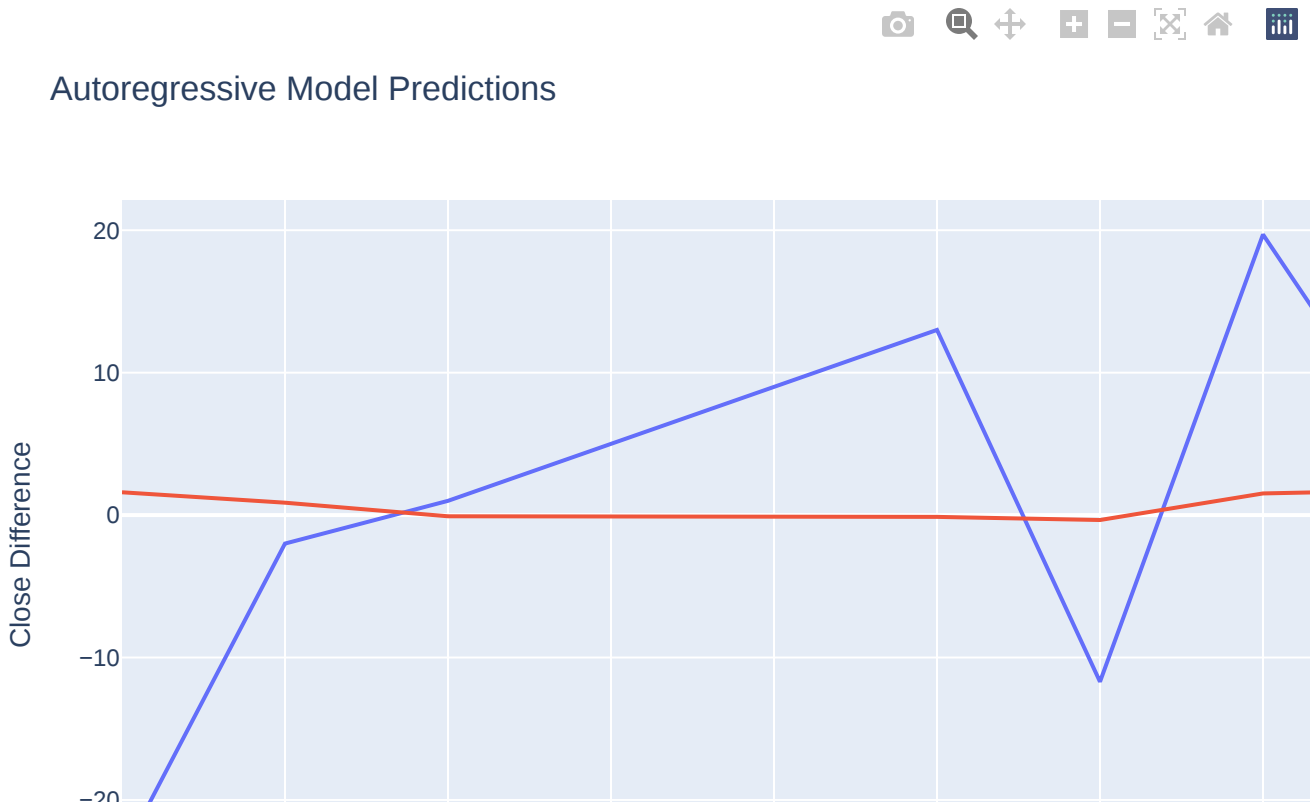
In [36]:
```
# Create a figure for ACF and PACF
fig = go.Figure()
fig.add_trace(go.Scatter(x=list(range(len(acf_result))), y=acf_result, mode='lines', nam
fig.add_trace(go.Scatter(x=list(range(len(pacf_result))), y=pacf_result, mode='lines', n
fig.update_layout(title='Auto-Correlation Function (ACF) and Partial Auto-Correlation Fu
                  xaxis_title='Lag', yaxis_title='Value')

# Show the plot
fig.show()
```



In [37]:

```
from statsmodels.tsa.ar_model import AutoReg
from sklearn.metrics import mean_squared_error
from math import sqrt

# Assuming the differenced series is stored in a DataFrame named "data" with column "Clo
X = data['Close_diff'].values
train, test = X[1:len(X)-7], X[len(X)-7:]

# Train autoregression
model = AutoReg(train, lags=20)
model_fit = model.fit()
print('Coefficients: %s' % model_fit.params)

# Predictions
predictions = model_fit.predict(start=len(train), end=len(train)+len(test)-1, dynamic=Fa
for i in range(len(predictions)):
    print('predicted=%f, expected=%f' % (predictions[i], test[i]))

# Calculate RMSE
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)

# Plot results
fig = go.Figure()
fig.add_trace(go.Scatter(x=data.index[-7:], y=test, mode='lines', name='Actual'))
fig.add_trace(go.Scatter(x=data.index[-7:], y=predictions, mode='lines', name='AR Predic
fig.update_layout(title='Autoregressive Model Predictions', xaxis_title='Date', yaxis_ti
fig.show()
```

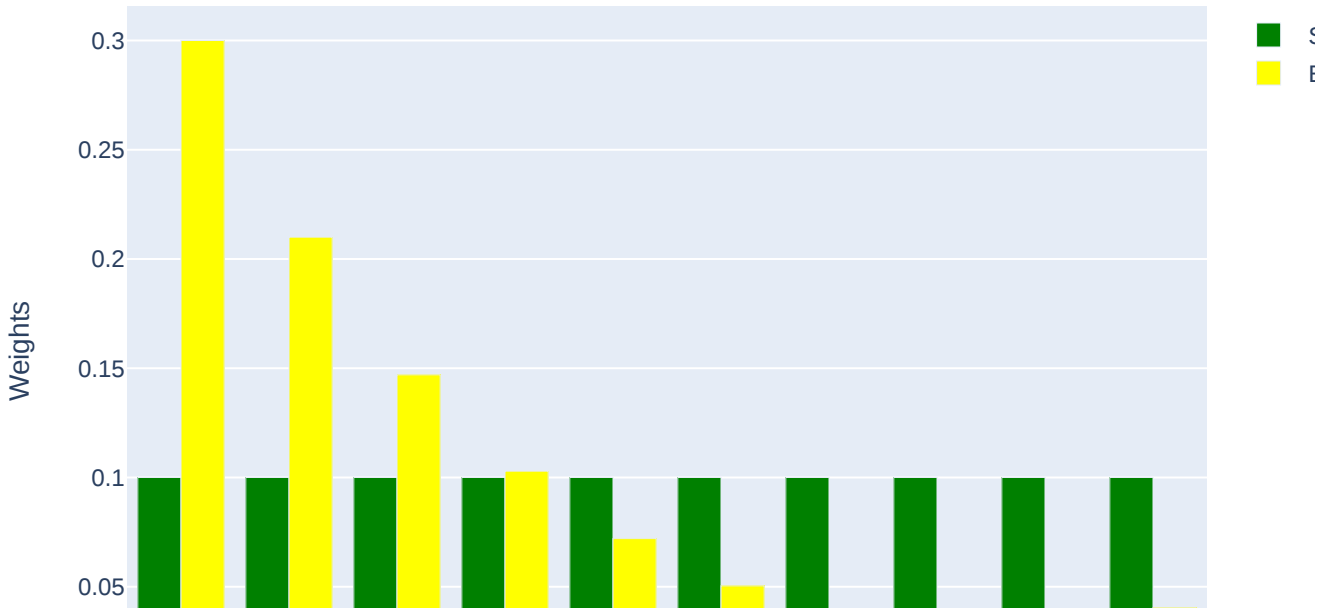
```
Coefficients: [-0.39987236 -0.03119116 -0.00738946  0.02411217 -0.01813114 -0.00546191
 -0.03452155 -0.03835032  0.01615527 -0.02006784 -0.01862806  0.01676793
  0.00174926  0.01591247 -0.00567496 -0.03557757 -0.04564573 -0.04499839
  0.0127257  -0.0218608  -0.01909351]
predicted=1.756797, expected=2.700000
predicted=1.519320, expected=19.700000
predicted=-0.345492, expected=-11.700000
predicted=-0.126109, expected=13.000000
predicted=-0.084123, expected=1.000000
predicted=0.869374, expected=-2.000000
predicted=1.601489, expected=-23.900000
Test RMSE: 13.588
```



In [38]:

```python
alpha = 0.3
n = 10
w_sma = np.repeat(1 / n, n)
colors = ['green', 'yellow']

# Calculate exponential moving average weights (EMA)
w_ema = [(1 - alpha) ** i if i == n - 1 else alpha * (1 - alpha) ** i for i in range(n)]

# Create a DataFrame to hold the weights
weights_df = pd.DataFrame({'w_sma': w_sma, 'w_ema': w_ema})

# Plot the weights using Plotly
fig = go.Figure()
fig.add_trace(go.Bar(x=weights_df.index, y=weights_df['w_sma'], name='Simple moving aver
fig.add_trace(go.Bar(x=weights_df.index, y=weights_df['w_ema'], name='Exponential moving

fig.update_layout(
    title='Moving Average Weights',
    le='Weights'
```

```
)

fig.show()
```



In [39]:
```
df_mean_1 =  data['Close_diff']
```

In [40]:
```
from statsmodels.tsa.arima.model import ARIMA

# Fit ARIMA model
order_arima = (1, 1, 0)  # (p, d, q) order of ARIMA model
arima_model = ARIMA(df_mean_1, order=order_arima)
arima_model_fit = arima_model.fit()
```

```
C:\Users\Acer\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWa
rning:

A date index has been provided, but it has no associated frequency information and so wi
ll be ignored when e.g. forecasting.

C:\Users\Acer\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWa
rning:

A date index has been provided, but it is not monotonic and so will be ignored when e.g.
forecasting.

C:\Users\Acer\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWa
rning:

A date index has been provided, but it has no associated frequency information and so wi
ll be ignored when e.g. forecasting.

C:\Users\Acer\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWa
rning:

A date index has been provided, but it is not monotonic and so will be ignored when e.g.
forecasting.

C:\Users\Acer\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWa
rning:

A date index has been provided, but it has no associated frequency information and so wi
ll be ignored when e.g. forecasting.

C:\Users\Acer\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWa
rning:

A date index has been provided, but it is not monotonic and so will be ignored when e.g.
forecasting.
```

In [41]:
```
# Generate predictions using ARIMA model
predictions_arima = arima_model_fit.predict(start=0, end=len(df_mean_1)-1)
```

In [42]:
```
# Create Plotly figure
fig = go.Figure()

# Add original data trace
fig.add_trace(go.Scatter(
    x=df_mean_1.index,
    y=df_mean_1.values,
    mode='lines',
    name='Original Data'
))

# Add ARIMA predictions trace
fig.add_trace(go.Scatter(
    x=df_mean_1.index,
    y=predictions_arima,
    mode='lines',
    name='ARIMA Predictions'
))

# Configure layout
fig.update_layout(
    title='ARIMA Predictions',
    xaxis_title='Time',
    yaxis_title='Value',
```

```
    showlegend=True
)

# Show the plot
fig.show()
```