Climate Change Modeling Project

Lakshman Chaudhary

```
#Importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.decomposition import PCA
import matplotlib.dates as mdates
from sklearn.model_selection import train_test_split
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.linear_model import LinearRegression
import os
from statsmodels.tsa.arima.model import ARIMA
import plotly.graph_objs as go
import seaborn as sns
import sys


print("System Config ::\nPython ::",sys.version)
```

```
System Config ::
Python :: 3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023,
18:05:47) [MSC v.1916 64 bit (AMD64)]
```

```
# Reading the dataset
# Get the current working directory (cwd)
cwd = os.getcwd()
# Get all the files in that directory
files = os.listdir(cwd)
print("Files in %r: %s" % (cwd, files))
```

```
Files in 'C:\\Users\\laksh\\Downloads\\Climate-Change -Modelling-
Project _NEW\\New folder': ['.ipynb_checkpoints',
'Climate_Change_Lchy.ipynb', 'climate_nasa.csv',
'DateVsLikesCount.jpeg', 'Decomposition_Plots.jpeg', 'Feature
Engineering.jpeg', 'Forecasted_Likes_Count.jpeg', 'Likes Count
Forecast_00.jpeg', 'Likes Count Forecast_01.jpeg', 'Likes Count
```

```
Forecast_02.jpeg', 'Likes Count Forecast_2.jpeg', 'Likes Count
Forecast_3.jpeg', 'Likes Count Forecast_4.jpeg', 'Likes Count
Forecast_5.jpeg', 'Likes Count Forecast__001.jpeg', 'Likes Count
Forecast__002.jpeg', 'likesCountVs2020-2023.jpeg',
'Likes_Count_Forecast_03.jpeg', 'Likes_Count_Predictions.jpeg',
'Likes_Count_with_SMA.jpeg', 'Original.jpeg', 'Original_1.jpeg',
'Original_2.jpeg', 'Predicted Likes Count.jpeg', 'Predicted Likes
Count_1.jpeg', 'Predicted vs Actual Likes Count.jpeg',
'Predicted_Likes_Count1.jpeg', 'Predicted_Likes_Count_001.jpeg',
'Predicted_Likes_Count_ARIMA.jpeg',
'Predicted_Likes_Count_ARIMA_2.jpeg',
'Predicted_vs_Actual_likesCount_ARIMA.jpeg',
'Predicted_vs_Actual_Likes_Count_1.jpeg',
'Predicted_vs_Actual_Likes_Count_ARIMA_01.jpeg',
'Predicted_vs_Actual_Likes_Count_ARIMA_1.jpeg', 'Residual.jpeg',
'Residual_1.jpeg', 'Residual_2.jpeg', 'Rolling Mean_commentsCount
changes from 2020-2023.jpeg', 'Rolling Mean_likesCount changes from
2020-2023.jpeg', 'Rolling STD_commentsCount changes from 2020-
2023.jpeg', 'Rolling STD_likesCount changes from 2020-2023.jpeg',
'Seasonal.jpeg', 'Seasonal_1.jpeg', 'Seasonal_2.jpeg', 'Trend.jpeg',
'Trend_1.jpeg', 'Trend_2.jpeg', 'YearVslikesCount.jpeg']
```

```
# Load the climate dataset
data = pd.read_csv('climate_nasa.csv')
data
```

```
                          date  likesCount  \
0    2022-09-07T17:12:32.000Z           2
1    2022-09-08T14:51:13.000Z           0
2    2022-09-07T17:19:41.000Z           1
3    2022-09-08T00:51:30.000Z           4
4    2022-09-07T19:06:20.000Z          16
..                        ...         ...
517  2022-12-22T17:21:37.000Z           0
518  2022-12-22T17:19:51.000Z           1
519  2022-12-22T17:12:57.000Z           3
520  2022-12-22T17:01:12.000Z           1
521  2022-12-22T17:00:08.000Z          12

                                          profileName  commentsCount
\
0    4dca617d86b3fdce80ba7e81fb16e048c9cd9798cdfd6d...           NaN

1    518ab97f2d115ba5b6f03b2fba2ef2b120540c9681288b...           NaN

2    d82e8e24eb633fd625b0aef9b3cb625cfb044ceb8483e1...           3.0

3    37a509fa0b5177a2233c7e2d0e2b2d6916695fa9fba3f2...           NaN

4    e54fbbd42a729af9d04d9a5cc1f9bbfe8081a31c219ecb...          26.0
```

```
..                                                  ...             ...

517  9e17b1a6422032d47472f0216c73aafda7587e302eed5e...             NaN

518  48e55d898603a136aefc44771f248bffd67242583a462a...             5.0

519  ca5d2611814cf8c9844ed06d9916d876d2dba94dac5ff2...             NaN

520  a87c8aea74c9b97002b368d6143ce9c2809dcaec3103fb...             NaN

521  cc70f6c9dcc2637b1c5fd90046834612627c752b828cb4...             1.0


                                                  text
0    Neat comparison I have not heard it before.\n ...
1    An excellent way to visualise the invisible! T...
2    Does the CO2/ghg in the troposphere affect the...
3    excellent post! I defo feel the difference - o...
4    Yes, and carbon dioxide does not harm the Eart...
..                                                 ...
517                      One can only hope for a peak ☹
518  what is the error margin for the temperature e...
519  We all should volenteerly help in reducing Glo...
520                                      Sergio Yepes
521  We're experiencing severe, abnormal weather pa...

[522 rows x 5 columns]

data

                         date  likesCount  \
0    2022-09-07T17:12:32.000Z           2
1    2022-09-08T14:51:13.000Z           0
2    2022-09-07T17:19:41.000Z           1
3    2022-09-08T00:51:30.000Z           4
4    2022-09-07T19:06:20.000Z          16
..                        ...         ...
517  2022-12-22T17:21:37.000Z           0
518  2022-12-22T17:19:51.000Z           1
519  2022-12-22T17:12:57.000Z           3
520  2022-12-22T17:01:12.000Z           1
521  2022-12-22T17:00:08.000Z          12

                                          profileName  commentsCount
\
0    4dca617d86b3fdce80ba7e81fb16e048c9cd9798cdfd6d...             NaN

1    518ab97f2d115ba5b6f03b2fba2ef2b120540c9681288b...             NaN

2    d82e8e24eb633fd625b0aef9b3cb625cfb044ceb8483e1...             3.0
```

```
3    37a509fa0b5177a2233c7e2d0e2b2d6916695fa9fba3f2...             NaN

4    e54fbbd42a729af9d04d9a5cc1f9bbfe8081a31c219ecb...            26.0

..                                                 ...             ...

517  9e17b1a6422032d47472f0216c73aafda7587e302eed5e...             NaN

518  48e55d898603a136aefc44771f248bffd67242583a462a...             5.0

519  ca5d2611814cf8c9844ed06d9916d876d2dba94dac5ff2...             NaN

520  a87c8aea74c9b97002b368d6143ce9c2809dcaec3103fb...             NaN

521  cc70f6c9dcc2637b1c5fd90046834612627c752b828cb4...             1.0


                                                  text
0    Neat comparison I have not heard it before.\n ...
1    An excellent way to visualise the invisible! T...
2    Does the CO2/ghg in the troposphere affect the...
3    excellent post! I defo feel the difference - o...
4    Yes, and carbon dioxide does not harm the Eart...
..                                                 ...
517                      One can only hope for a peak ☹
518  what is the error margin for the temperature e...
519  We all should volenteerly help in reducing Glo...
520                                        Sergio Yepes
521  We're experiencing severe, abnormal weather pa...

[522 rows x 5 columns]
```

Now we will explore the all the basic aspects of the dataset

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 522 entries, 0 to 521
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           522 non-null    object
 1   likesCount     522 non-null    int64
 2   profileName    522 non-null    object
 3   commentsCount  244 non-null    float64
 4   text           504 non-null    object
```

```
dtypes: float64(1), int64(1), object(3)
memory usage: 20.5+ KB
```

```
# Coloumn names of the dataset
data.columns
```

```
Index(['date', 'likesCount', 'profileName', 'commentsCount', 'text'],
dtype='object')
```

Here we can see the basic structure of the dataset .i.e. Types of Column exist in the dataset. It gives us the rough idea about what kind of data we are dealing with and what kind of transformation we need to bring in the dataset

```
data.describe()
```

```
       likesCount  commentsCount
count  522.000000     244.000000
mean     4.720307       8.696721
std     12.053556      12.266176
min      0.000000       1.000000
25%      0.000000       2.000000
50%      1.000000       5.000000
75%      4.000000      10.000000
max    126.000000      93.000000
```

```
# First 5 Records in the dataset
data.head()
```

```
                       date  likesCount  \
0  2022-09-07T17:12:32.000Z           2
1  2022-09-08T14:51:13.000Z           0
2  2022-09-07T17:19:41.000Z           1
3  2022-09-08T00:51:30.000Z           4
4  2022-09-07T19:06:20.000Z          16

                                         profileName  commentsCount  \
0  4dca617d86b3fdce80ba7e81fb16e048c9cd9798cdfd6d...            NaN
1  518ab97f2d115ba5b6f03b2fba2ef2b120540c9681288b...            NaN
2  d82e8e24eb633fd625b0aef9b3cb625cfb044ceb8483e1...            3.0
3  37a509fa0b5177a2233c7e2d0e2b2d6916695fa9fba3f2...            NaN
```

```
4  e54fbbd42a729af9d04d9a5cc1f9bbfe8081a31c219ecb...           26.0

                                                  text
0  Neat comparison I have not heard it before.\n ...
1  An excellent way to visualise the invisible! T...
2  Does the CO2/ghg in the troposphere affect the...
3  excellent post! I defo feel the difference - o...
4  Yes, and carbon dioxide does not harm the Eart...
```

```
# Last 5 Records in the dataset
data.tail()
```

```
                         date  likesCount  \
517  2022-12-22T17:21:37.000Z           0
518  2022-12-22T17:19:51.000Z           1
519  2022-12-22T17:12:57.000Z           3
520  2022-12-22T17:01:12.000Z           1
521  2022-12-22T17:00:08.000Z          12

                                           profileName  commentsCount
\
517  9e17b1a6422032d47472f0216c73aafda7587e302eed5e...           NaN

518  48e55d898603a136aefc44771f248bffd67242583a462a...           5.0

519  ca5d2611814cf8c9844ed06d9916d876d2dba94dac5ff2...           NaN

520  a87c8aea74c9b97002b368d6143ce9c2809dcaec3103fb...           NaN

521  cc70f6c9dcc2637b1c5fd90046834612627c752b828cb4...           1.0


                                                  text
517                    One can only hope for a peak ☹
518  what is the error margin for the temperature e...
519  We all should volenteerly help in reducing Glo...
520                                       Sergio Yepes
521  We're experiencing severe, abnormal weather pa...
```

```
# Lets see the number of Null values in each column
data.isnull().sum()
```

```
likesCount         0
profileName        0
commentsCount    278
text              18
dtype: int64
```

```
#Droping all null values
data = data.dropna(how='any' ,axis=0)
data.shape
```

```
(242, 5)
```

```
# Here we can see the unique values exist in each column
data.nunique()
```

```
date              242
likesCount         39
profileName       229
commentsCount      39
text              242
dtype: int64
```

```
# All Null Values record dropped
data.isnull().sum()
```

```
date              0
likesCount        0
profileName       0
commentsCount     0
text              0
dtype: int64
```

```
data.rename(columns={'date' : 'likesCount','profileName ':
'confidence_interval_temp'},inplace=True)
data.head()
```

```
                           likesCount  \
date
2020-09-15 21:25:05+00:00          27
2020-09-15 21:30:35+00:00           7
2020-09-15 21:32:15+00:00           4
2020-09-15 21:36:05+00:00           3
2020-09-15 21:36:59+00:00          16


profileName  \
date

2020-09-15 21:25:05+00:00
e332cfd3f53d89cdd75a57b80d011e2f5984437cc51340...
2020-09-15 21:30:35+00:00
7d6065e313e1919d05a309bd59754895d9518d3ca8e8ba...
2020-09-15 21:32:15+00:00
930f8ce8e022378d44088377fef6a069da6c519c2e5ba1...
2020-09-15 21:36:05+00:00
00febfebfa7073f73c576deb9dba73eb9f98e26bb03f3d...
2020-09-15 21:36:59+00:00
6c5e0b507471c121613153d3459e97b050dd47d1ec744c...

                           commentsCount  \
date
```

```
2020-09-15 21:25:05+00:00            6.0
2020-09-15 21:30:35+00:00            8.0
2020-09-15 21:32:15+00:00            NaN
2020-09-15 21:36:05+00:00            7.0
2020-09-15 21:36:59+00:00            8.0


text
date

2020-09-15 21:25:05+00:00  I wish this was simply old news and we had
peo...
2020-09-15 21:30:35+00:00  The sad thing is that if we look at
projected ...
2020-09-15 21:32:15+00:00  We all play a part....as a world we need to
do...
2020-09-15 21:36:05+00:00           Re ord being from 1860
something...lolol
2020-09-15 21:36:59+00:00  The climate is changing but it’s ridiculous
to...
```

```
# Load the climate dataset
data = pd.read_csv('climate_nasa.csv')
data

print(data.columns)
```

```
Index(['date', 'likesCount', 'profileName', 'commentsCount', 'text'],
dtype='object')
```

```
print(data.columns)
```

```
Index(['date', 'likesCount', 'profileName', 'commentsCount', 'text'],
dtype='object')
```

```
import pandas as pd

# Example DataFrame with the columns provided
# data = pd.DataFrame({
#     'date': ['2023-01-01', '2023-01-02'],
#     'likesCount': [10, 20],
#     'profileName': ['profile1', 'profile2'],
#     'commentsCount': [5, 10],
#     'text': ['text1', 'text2']
# })

# Convert 'date' column to datetime
data['date'] = pd.to_datetime(data['date'])

# Set 'date' as the index
```

```
data.set_index('date', inplace=True)

# Drop columns if not needed
# If you only need 'likesCount' as the main column
data.drop(columns=['profileName', 'commentsCount', 'text'],
inplace=True, errors='ignore')

# Display the updated DataFrame
print(data.head())
```

```
                           likesCount
date
2022-09-07 17:12:32+00:00           2
2022-09-08 14:51:13+00:00           0
2022-09-07 17:19:41+00:00           1
2022-09-08 00:51:30+00:00           4
2022-09-07 19:06:20+00:00          16
```

```
#Converting 'Date column' into 'Datetime datatype'

import pandas as pd
# Load the climate dataset
data = pd.read_csv('climate_nasa.csv')
data

print(data.columns)

# Ensure 'date' column is in datetime format
data['date'] = pd.to_datetime(data['date'])

# Set 'date' as the index
data.set_index('date', inplace=True)

# Drop columns if necessary
data.drop(columns=['likesCount'], inplace=True, errors='ignore')

# Display the DataFrame index
print(data.index)
```

```
Index(['date', 'likesCount', 'profileName', 'commentsCount', 'text'],
dtype='object')
DatetimeIndex(['2022-09-07 17:12:32+00:00', '2022-09-08
14:51:13+00:00',
               '2022-09-07 17:19:41+00:00', '2022-09-08
00:51:30+00:00',
               '2022-09-07 19:06:20+00:00', '2022-09-15
17:10:57+00:00',
               '2022-09-07 17:31:25+00:00', '2022-09-07
18:07:53+00:00',
               '2022-09-07 22:45:56+00:00', '2022-09-07
16:30:55+00:00',
```

```
               ...
               '2022-12-22 17:54:41+00:00', '2022-12-22
17:51:29+00:00',
               '2022-12-22 17:46:40+00:00', '2022-12-22
17:44:23+00:00',
               '2022-12-22 17:41:51+00:00', '2022-12-22
17:21:37+00:00',
               '2022-12-22 17:19:51+00:00', '2022-12-22
17:12:57+00:00',
               '2022-12-22 17:01:12+00:00', '2022-12-22
17:00:08+00:00'],
              dtype='datetime64[ns, UTC]', name='date', length=522,
freq=None)
```

```
data.head()
```

```
profileName  \
date

2022-09-07 17:12:32+00:00
4dca617d86b3fdce80ba7e81fb16e048c9cd9798cdfd6d...
2022-09-08 14:51:13+00:00
518ab97f2d115ba5b6f03b2fba2ef2b120540c9681288b...
2022-09-07 17:19:41+00:00
d82e8e24eb633fd625b0aef9b3cb625cfb044ceb8483e1...
2022-09-08 00:51:30+00:00
37a509fa0b5177a2233c7e2d0e2b2d6916695fa9fba3f2...
2022-09-07 19:06:20+00:00
e54fbbd42a729af9d04d9a5cc1f9bbfe8081a31c219ecb...

                           commentsCount  \
date
2022-09-07 17:12:32+00:00            NaN
2022-09-08 14:51:13+00:00            NaN
2022-09-07 17:19:41+00:00            3.0
2022-09-08 00:51:30+00:00            NaN
2022-09-07 19:06:20+00:00           26.0


text
date

2022-09-07 17:12:32+00:00  Neat comparison I have not heard it
before.\n ...
2022-09-08 14:51:13+00:00  An excellent way to visualise the
invisible! T...
2022-09-07 17:19:41+00:00  Does the CO2/ghg in the troposphere affect
the...
2022-09-08 00:51:30+00:00  excellent post! I defo feel the difference
```

```
- o...
2022-09-07 19:06:20+00:00  Yes, and carbon dioxide does not harm the
Eart...
```

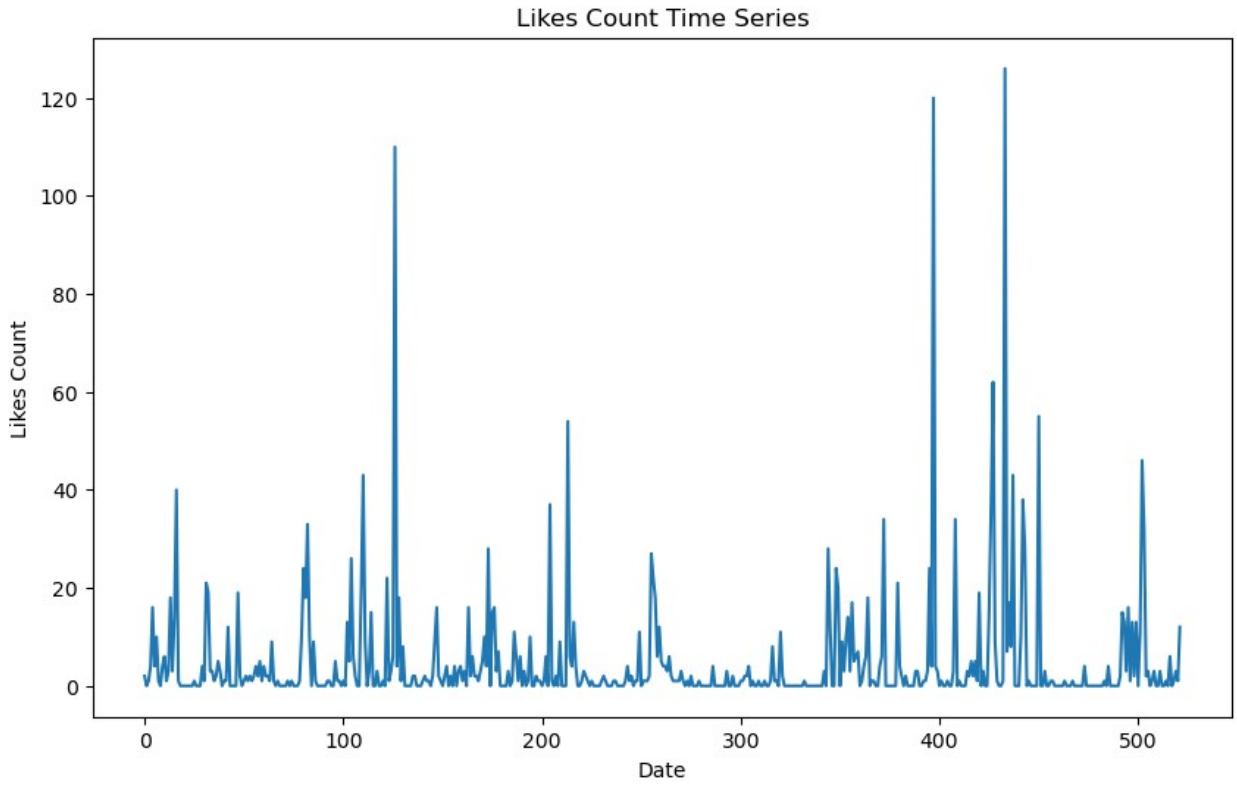
```
# Now we use year as index
data['date']= data.index.date
data.head()
```

```


profileName  \
date

2022-09-07 17:12:32+00:00
4dca617d86b3fdce80ba7e81fb16e048c9cd9798cdfd6d...
2022-09-08 14:51:13+00:00
518ab97f2d115ba5b6f03b2fba2ef2b120540c9681288b...
2022-09-07 17:19:41+00:00
d82e8e24eb633fd625b0aef9b3cb625cfb044ceb8483e1...
2022-09-08 00:51:30+00:00
37a509fa0b5177a2233c7e2d0e2b2d6916695fa9fba3f2...
2022-09-07 19:06:20+00:00
e54fbbd42a729af9d04d9a5cc1f9bbfe8081a31c219ecb...

                           commentsCount  \
date
2022-09-07 17:12:32+00:00            NaN
2022-09-08 14:51:13+00:00            NaN
2022-09-07 17:19:41+00:00            3.0
2022-09-08 00:51:30+00:00            NaN
2022-09-07 19:06:20+00:00           26.0


text  \
date

2022-09-07 17:12:32+00:00  Neat comparison I have not heard it
before.\n ...
2022-09-08 14:51:13+00:00  An excellent way to visualise the
invisible! T...
2022-09-07 17:19:41+00:00  Does the CO2/ghg in the troposphere affect
the...
2022-09-08 00:51:30+00:00  excellent post! I defo feel the difference
- o...
2022-09-07 19:06:20+00:00  Yes, and carbon dioxide does not harm the
Eart...

                                    date
date
2022-09-07 17:12:32+00:00  2022-09-07
2022-09-08 14:51:13+00:00  2022-09-08
```

```
2022-09-07 17:19:41+00:00  2022-09-07
2022-09-08 00:51:30+00:00  2022-09-08
2022-09-07 19:06:20+00:00  2022-09-07

# Now we use year as index
data['Year']= data.index.year
data.head()


profileName  \
date

2022-09-07 17:12:32+00:00
4dca617d86b3fdce80ba7e81fb16e048c9cd9798cdfd6d...
2022-09-08 14:51:13+00:00
518ab97f2d115ba5b6f03b2fba2ef2b120540c9681288b...
2022-09-07 17:19:41+00:00
d82e8e24eb633fd625b0aef9b3cb625cfb044ceb8483e1...
2022-09-08 00:51:30+00:00
37a509fa0b5177a2233c7e2d0e2b2d6916695fa9fba3f2...
2022-09-07 19:06:20+00:00
e54fbbd42a729af9d04d9a5cc1f9bbfe8081a31c219ecb...

                           commentsCount  \
date
2022-09-07 17:12:32+00:00            NaN
2022-09-08 14:51:13+00:00            NaN
2022-09-07 17:19:41+00:00            3.0
2022-09-08 00:51:30+00:00            NaN
2022-09-07 19:06:20+00:00           26.0


text  \
date

2022-09-07 17:12:32+00:00  Neat comparison I have not heard it
before.\n ...
2022-09-08 14:51:13+00:00  An excellent way to visualise the
invisible! T...
2022-09-07 17:19:41+00:00  Does the CO2/ghg in the troposphere affect
the...
2022-09-08 00:51:30+00:00  excellent post! I defo feel the difference
- o...
2022-09-07 19:06:20+00:00  Yes, and carbon dioxide does not harm the
Eart...

                                  date  Year
date
2022-09-07 17:12:32+00:00  2022-09-07  2022
2022-09-08 14:51:13+00:00  2022-09-08  2022
```

```
2022-09-07 17:19:41+00:00  2022-09-07  2022
2022-09-08 00:51:30+00:00  2022-09-08  2022
2022-09-07 19:06:20+00:00  2022-09-07  2022
```

```python
# Plot likesCount time series
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Load the climate dataset
data = pd.read_csv('climate_nasa.csv')
data
plt.figure(figsize=(10, 6))
plt.plot(data['likesCount'])
plt.title('Likes Count Time Series')
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.savefig('DateVsLikesCount.jpeg')
plt.show()
```




```python
from sklearn.model_selection import train_test_split

# Split data into training and testing sets
train, test = train_test_split(data, test_size=0.2, random_state=42)

# Check the split
```

```
print(f"Training set size: {train.shape}")
print(f"Testing set size: {test.shape}")
```

```
Training set size: (417, 5)
Testing set size: (105, 5)
```

```
# Create and train linear regression model for likesCount
model = LinearRegression()
model.fit(train.index.values.reshape(-1, 1), train['likesCount'])
```

```
LinearRegression()
```

```
# Import necessary libraries
from sklearn.linear_model import LinearRegression

# Ensure 'likesCount' exists in the data
if 'likesCount' in train.columns:
    # Create and train linear regression model
    model = LinearRegression()

    # Reshape the index values (dates) as features (assuming a time-
based model)
    X_train = train.index.values.reshape(-1, 1)
    y_train = train['likesCount']

    # Fit the model
    model.fit(X_train, y_train)

    # Print the model coefficients
    print(f"Intercept: {model.intercept_}")
    print(f"Coefficient: {model.coef_}")
else:
    print("'likesCount' column not found in the data.")
```

```
Intercept: -34.60413607400717
Coefficient: [2.40137878e-17]
```

```
# Load the climate dataset
data = pd.read_csv('climate_nasa.csv')
data

print(data.columns)

print(test.columns)
```

```
Index(['date', 'likesCount', 'profileName', 'commentsCount', 'text'],
dtype='object')
Index(['likesCount', 'profileName', 'commentsCount', 'text'],
dtype='object')
```

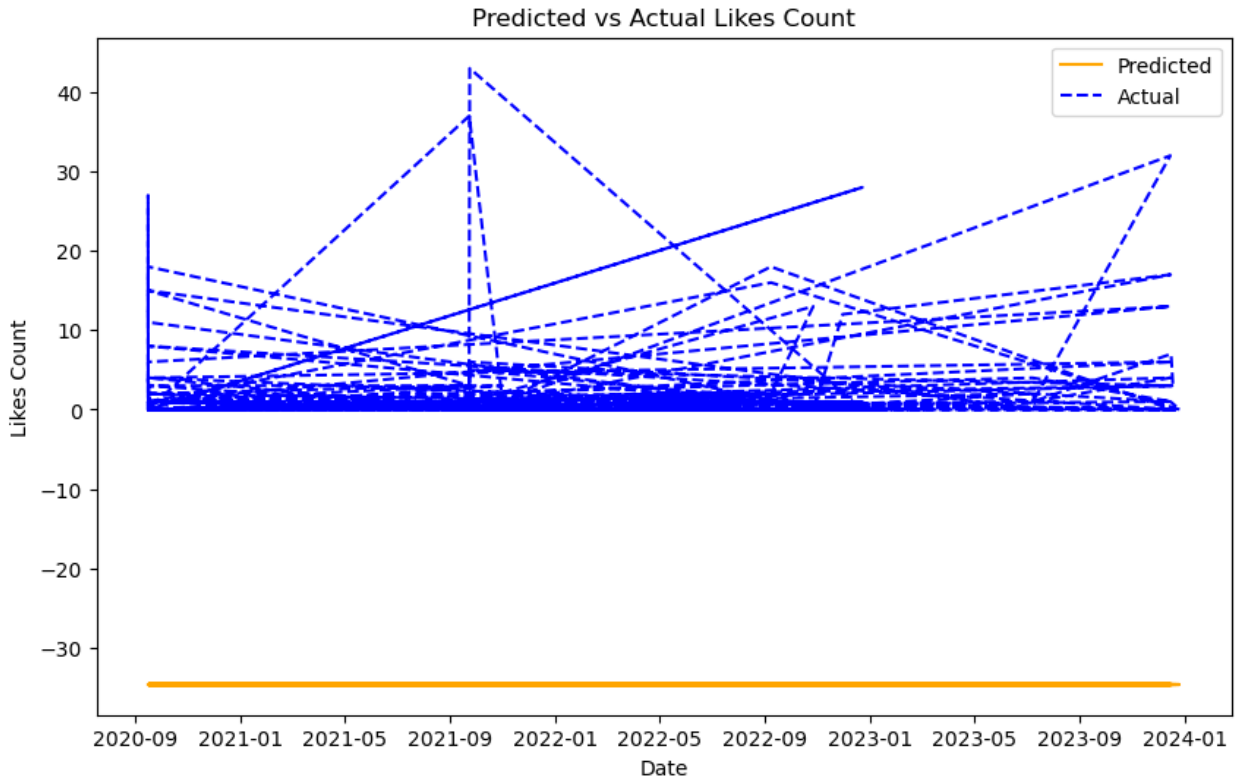
```
print(test.columns)
```

```
Index(['likesCount', 'profileName', 'commentsCount', 'text'],
dtype='object')
```

```python
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Check if the index is already a DatetimeIndex, if not, convert it
if not isinstance(test.index, pd.DatetimeIndex):
    test.index = pd.to_datetime(test.index)

# Convert date index to numeric representation
test_dates_num = mdates.date2num(test.index.to_pydatetime())

# Make predictions on test set
predictions = model.predict(test_dates_num.reshape(-1, 1))

# Print predictions (optional)
print(predictions)

# Visualize predictions
plt.figure(figsize=(10, 6))
plt.plot(test.index, predictions, label='Predicted', color='orange')
plt.plot(test.index, test['likesCount'], label='Actual', color='blue',
linestyle='--')

# Add labels and title
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.title('Predicted vs Actual Likes Count')
plt.legend()

# Save the plot as an image file
plt.savefig('Predicted_vs_Actual_Likes_Count_1.jpeg')

# Display the plot
plt.show()
```

```
[-34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
```

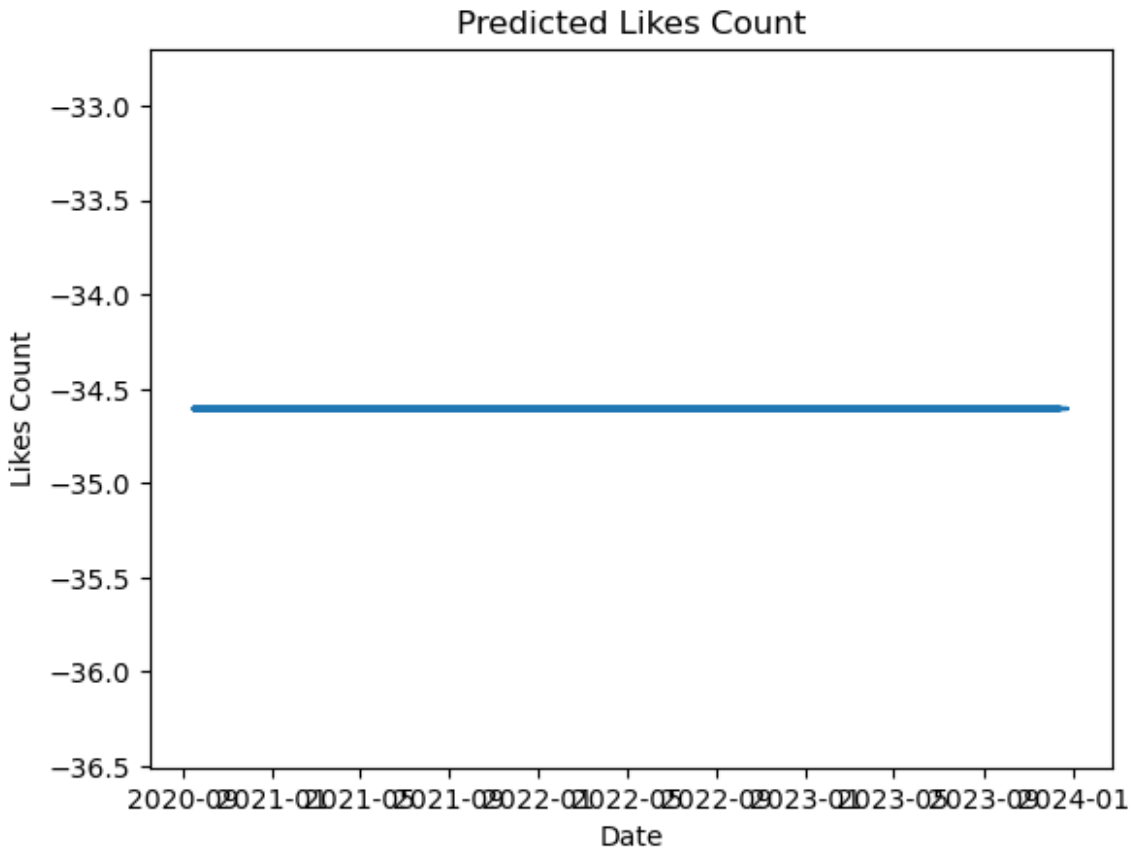
```
-34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
-34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
-34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
-34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
-34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
-34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
-34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
-34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607]
```




```python
import matplotlib.pyplot as plt

# Convert date index to numeric representation
test_dates_num = mdates.date2num(test.index.to_pydatetime())

# Make predictions on test set
predictions = model.predict(test_dates_num.reshape(-1, 1))

# Print predictions
print(predictions)


# Visualize predictions
plt.plot(test.index, predictions)
plt.xlabel('Date')
plt.ylabel('Likes Count')
```

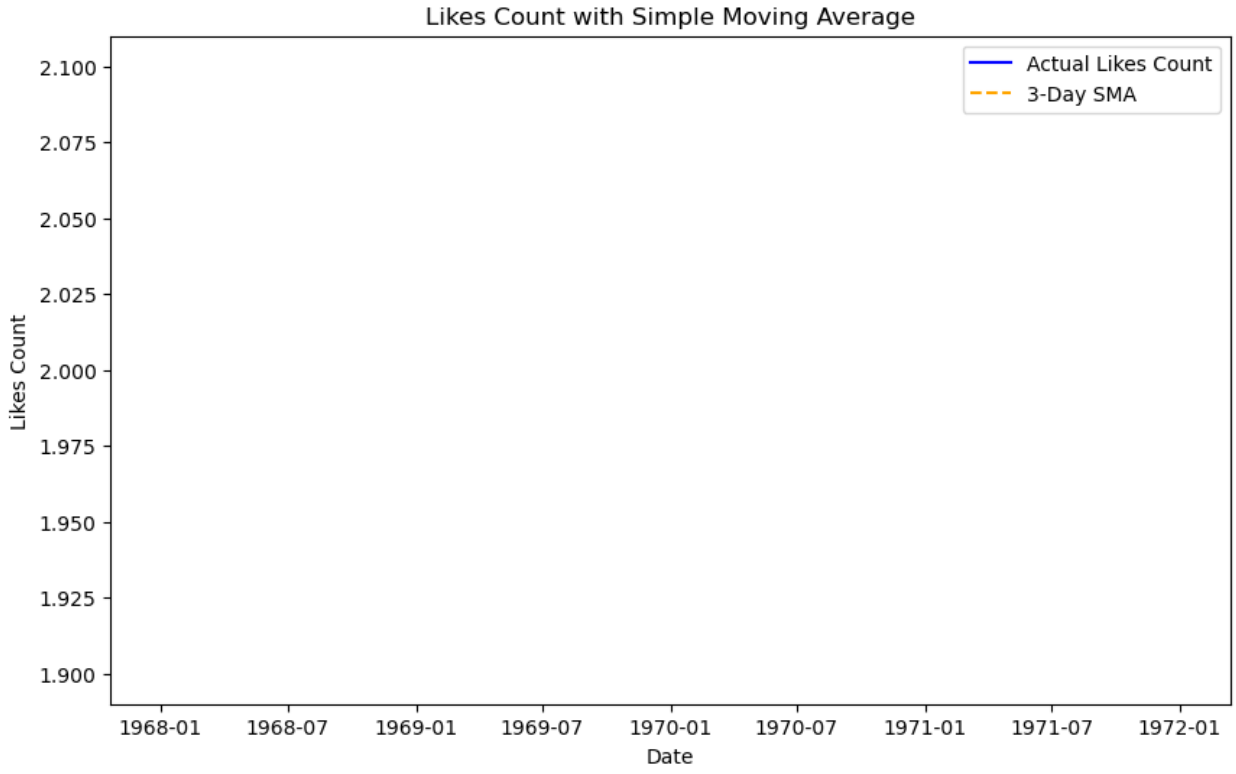
```python
plt.title('Predicted Likes Count')
plt.savefig('Predicted Likes Count.jpeg')
plt.show()
```

```
[-34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607
 -34.60413607 -34.60413607 -34.60413607 -34.60413607 -34.60413607]
```



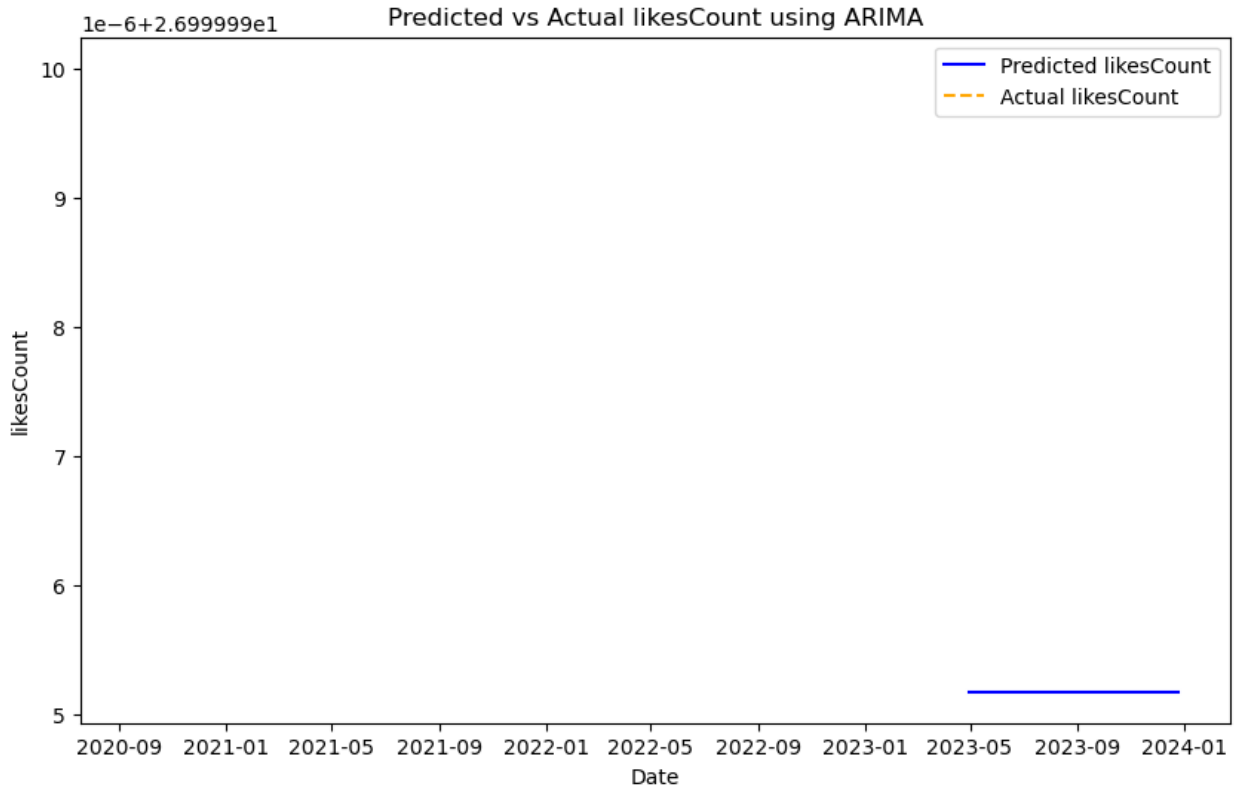

```
import pandas as pd
import matplotlib.pyplot as plt

# Sample DataFrame creation (Replace this with your actual DataFrame)
# data = pd.read_csv('your_data.csv', parse_dates=['date'],
index_col='date')
# Ensure that 'likesCount' is your target variable

# Load the climate dataset
data = pd.read_csv('climate_nasa.csv')
data

# Convert index to DatetimeIndex if necessary
if not isinstance(data.index, pd.DatetimeIndex):
    data.index = pd.to_datetime(data.index)

# Set the frequency of the time series (e.g., daily frequency 'D')
data = data.asfreq('D')

# Calculate Simple Moving Average (SMA)
window_size = 3  # Adjust window size as needed
data['SMA'] = data['likesCount'].rolling(window=window_size).mean()
```

```
# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(data.index, data['likesCount'], label='Actual Likes Count',
color='blue')
plt.plot(data.index, data['SMA'], label=f'{window_size}-Day SMA',
color='orange', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.title('Likes Count with Simple Moving Average')
plt.legend()
plt.savefig('Likes_Count_with_SMA.jpeg')
plt.show()
```




```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
import warnings

# Load climate_nasa.csv dataset
data = pd.read_csv('climate_nasa.csv', parse_dates=['date'],
index_col='date')

# Ensure 'likesCount' is the target variable
```

```
target_variable = 'likesCount'  # Replace with your target variable

# Handle missing data
numeric_cols = data.select_dtypes(include=['int64',
'float64']).columns
data[numeric_cols] =
data[numeric_cols].fillna(data[numeric_cols].mean())

# Convert index to DatetimeIndex if necessary
if not isinstance(data.index, pd.DatetimeIndex):
    data.index = pd.to_datetime(data.index)

# Set the frequency of the time series (e.g., daily frequency 'D')
data = data.asfreq('D')

# Split data into training and testing sets (80% for training)
train_size = int(0.8 * len(data))
train_data, test_data = data[:train_size], data[train_size:]

# Suppress warnings
warnings.filterwarnings("ignore", category=UserWarning,
module='statsmodels')

# Fit simplified ARIMA model
model = ARIMA(train_data[target_variable], order=(1,0,1))
fitted_model = model.fit()

# Forecasting
n_periods = len(test_data)
forecast_result = fitted_model.get_forecast(steps=n_periods)
predictions = forecast_result.predicted_mean

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(test_data.index, predictions, label='Predicted ' +
target_variable, color='blue')
plt.plot(data.index, data[target_variable], label='Actual ' +
target_variable, color='orange', linestyle='--')
plt.xlabel('Date')
plt.ylabel(target_variable)
plt.title('Predicted vs Actual ' + target_variable + ' using ARIMA')
plt.legend()
plt.savefig('Predicted_vs_Actual_' + target_variable + '_ARIMA.jpeg')
plt.show()
```



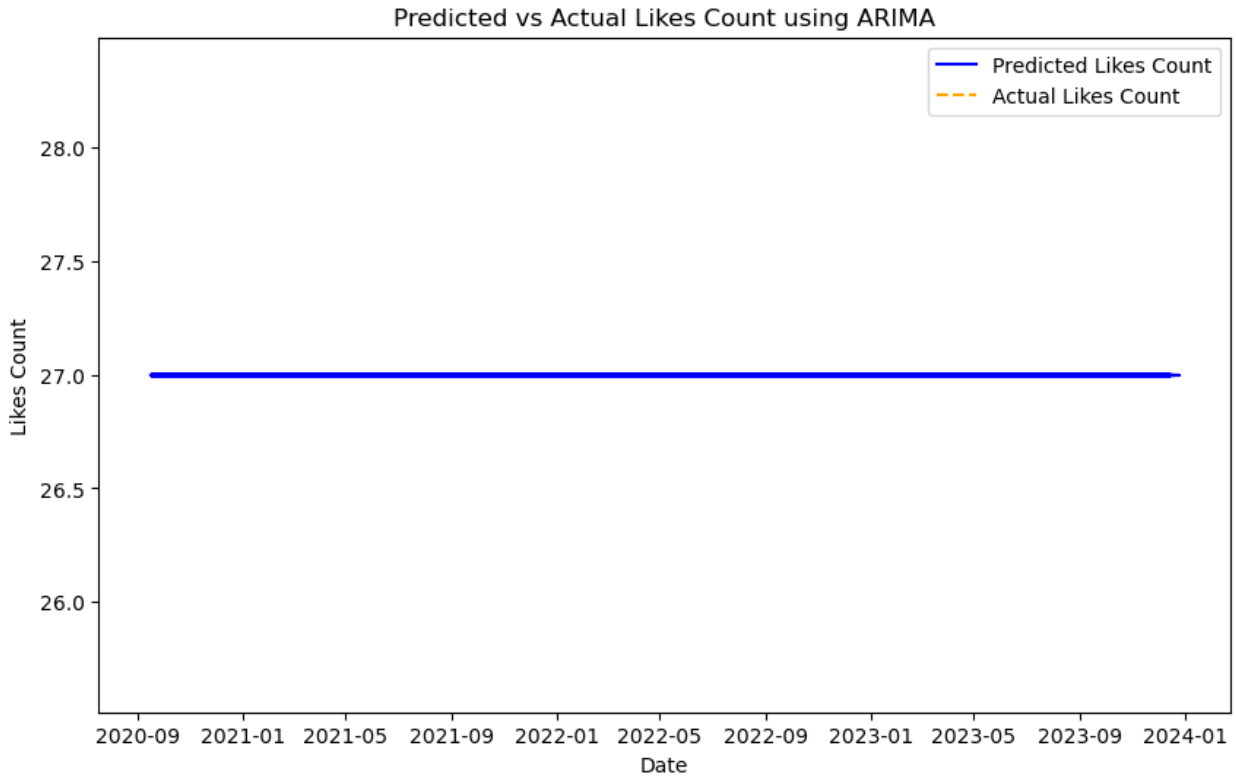

```
import pandas as pd

# Load the dataset
data = pd.read_csv('climate_nasa.csv', parse_dates=['date'],
index_col='date')

# Display the first few rows and column names of the dataset
print(data.head())
print(data.columns)
```

```
                           likesCount  \
date
2022-09-07 17:12:32+00:00           2
2022-09-08 14:51:13+00:00           0
2022-09-07 17:19:41+00:00           1
2022-09-08 00:51:30+00:00           4
2022-09-07 19:06:20+00:00          16


profileName  \
date

2022-09-07 17:12:32+00:00
4dca617d86b3fdce80ba7e81fb16e048c9cd9798cdfd6d...
2022-09-08 14:51:13+00:00
518ab97f2d115ba5b6f03b2fba2ef2b120540c9681288b...
```

```
2022-09-07 17:19:41+00:00
d82e8e24eb633fd625b0aef9b3cb625cfb044ceb8483e1...
2022-09-08 00:51:30+00:00
37a509fa0b5177a2233c7e2d0e2b2d6916695fa9fba3f2...
2022-09-07 19:06:20+00:00
e54fbbd42a729af9d04d9a5cc1f9bbfe8081a31c219ecb...

                           commentsCount  \
date
2022-09-07 17:12:32+00:00            NaN
2022-09-08 14:51:13+00:00            NaN
2022-09-07 17:19:41+00:00            3.0
2022-09-08 00:51:30+00:00            NaN
2022-09-07 19:06:20+00:00           26.0


text
date

2022-09-07 17:12:32+00:00  Neat comparison I have not heard it
before.\n ...
2022-09-08 14:51:13+00:00  An excellent way to visualise the
invisible! T...
2022-09-07 17:19:41+00:00  Does the CO2/ghg in the troposphere affect
the...
2022-09-08 00:51:30+00:00  excellent post! I defo feel the difference
- o...
2022-09-07 19:06:20+00:00  Yes, and carbon dioxide does not harm the
Eart...
Index(['likesCount', 'profileName', 'commentsCount', 'text'],
dtype='object')
```

```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Sample DataFrame creation (Replace this with your actual DataFrame)
# data = pd.read_csv('your_data.csv', parse_dates=['date'],
index_col='date')
# Ensure that 'likesCount' is your target variable

# Convert index to DatetimeIndex if necessary
if not isinstance(data.index, pd.DatetimeIndex):
    data.index = pd.to_datetime(data.index)

# Set the frequency of the time series (e.g., daily frequency 'D')
data = data.asfreq('D')

# Fit the ARIMA model
model = ARIMA(data['likesCount'], order=(1, 1, 1))  # Replace (p, d,
```

```
q) with your chosen ARIMA parameters
fitted_model = model.fit()

# Forecasting
n_periods = len(test.index)  # Assuming 'test' is your DataFrame for
testing
forecast_result = fitted_model.get_forecast(steps=n_periods)
predictions = forecast_result.predicted_mean

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(test.index, predictions, label='Predicted Likes Count',
color='blue')
plt.plot(data.index, data['likesCount'], label='Actual Likes Count',
color='orange', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.title('Predicted vs Actual Likes Count using ARIMA')
plt.legend()
plt.savefig('Predicted_vs_Actual_Likes_Count_ARIMA_1.jpeg')
plt.show()
```

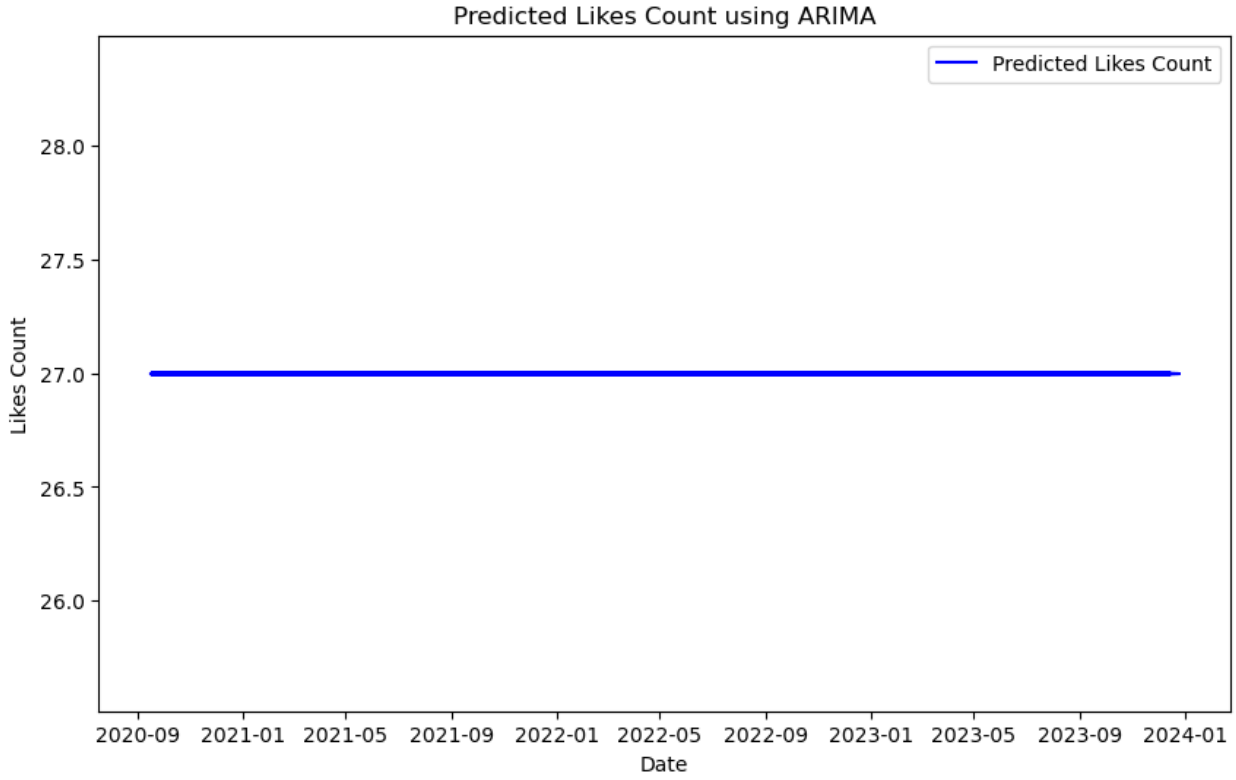



```
import matplotlib.pyplot as plt
import pandas as pd
```

```python
# Ensure test.index is a DatetimeIndex
if not isinstance(test.index, pd.DatetimeIndex):
    test.index = pd.to_datetime(test.index)

# Fit the ARIMA model
try:
    fitted_model = model.fit()  # Fit the model if not already fitted
except Exception as e:
    print(f"Error fitting the model: {e}")
    raise

# Check that the model has a get_forecast method
if not hasattr(fitted_model, 'get_forecast'):
    raise AttributeError("The fitted model must have a 'get_forecast'
method to make predictions.")

# Number of steps to forecast (based on the length of the test set)
n_periods = len(test.index)

# Generate predictions using the fitted model's forecast method
try:
    forecast_result = fitted_model.get_forecast(steps=n_periods)
    predictions = forecast_result.predicted_mean
except Exception as e:
    print(f"Error in forecasting: {e}")
    raise

# Print predictions for verification
print(predictions)

# Visualize predictions
plt.figure(figsize=(10, 6))
plt.plot(test.index, predictions, label='Predicted Likes Count',
color='blue')
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.title('Predicted Likes Count using ARIMA')
plt.legend()
plt.savefig('Predicted_Likes_Count_ARIMA_2.jpeg')
plt.show()
```

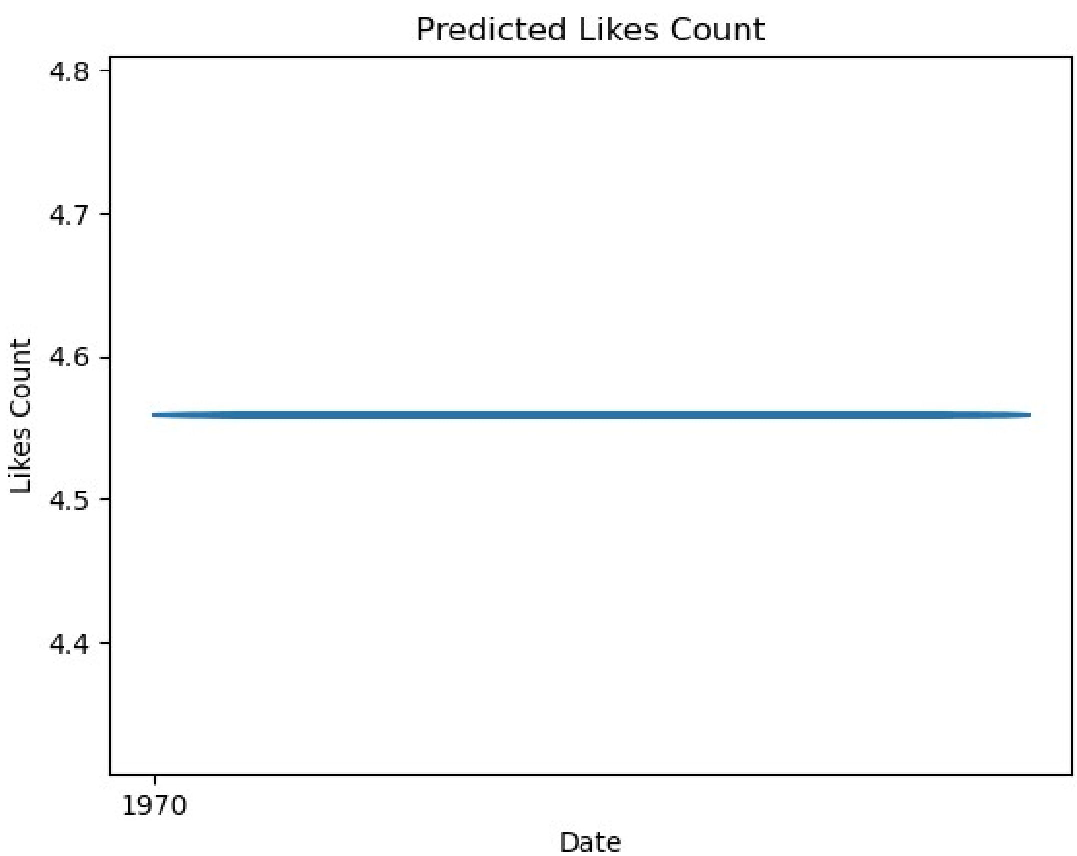
```
2023-04-29 21:25:05+00:00    26.999995
2023-04-30 21:25:05+00:00    26.999995
2023-05-01 21:25:05+00:00    26.999995
2023-05-02 21:25:05+00:00    26.999995
2023-05-03 21:25:05+00:00    26.999995
                               ...
2023-08-07 21:25:05+00:00    26.999995
2023-08-08 21:25:05+00:00    26.999995
2023-08-09 21:25:05+00:00    26.999995
```

```
2023-08-10 21:25:05+00:00    26.999995
2023-08-11 21:25:05+00:00    26.999995
Freq: D, Name: predicted_mean, Length: 105, dtype: float64
```




```python
import matplotlib.pyplot as plt

# Convert date index to numeric representation
test_dates_num = mdates.date2num(test.index.to_pydatetime())

# Make predictions on test set
predictions = model.predict(test_dates_num.reshape(-1, 1))

# Print predictions
print(predictions)


# Visualize predictions
plt.plot(test.index, predictions)
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.title('Predicted Likes Count')
plt.savefig('Predicted Likes Count.jpeg')
plt.show()
```

```
[4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705 4.55897705
 4.55897705 4.55897705 4.55897705]
```




```
# Evaluate model performance
mse = mean_squared_error(test['likesCount'], predictions)
print('Mean Squared Error:', mse)
```

```
Mean Squared Error: 191.59740358179963
```

```
from sklearn.impute import SimpleImputer

# Initialize imputer (mean strategy is often used for numerical data)
imputer = SimpleImputer(strategy='mean')

# Fit imputer on training data and transform both training and test
sets
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Continue with scaling and model training
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imputed)
X_test_scaled = scaler.transform(X_test_imputed)

# Train the model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Make predictions
predictions = model.predict(X_test_scaled)

# Evaluate model performance
mse = mean_squared_error(y_test, predictions)
print('Mean Squared Error:', mse)
```

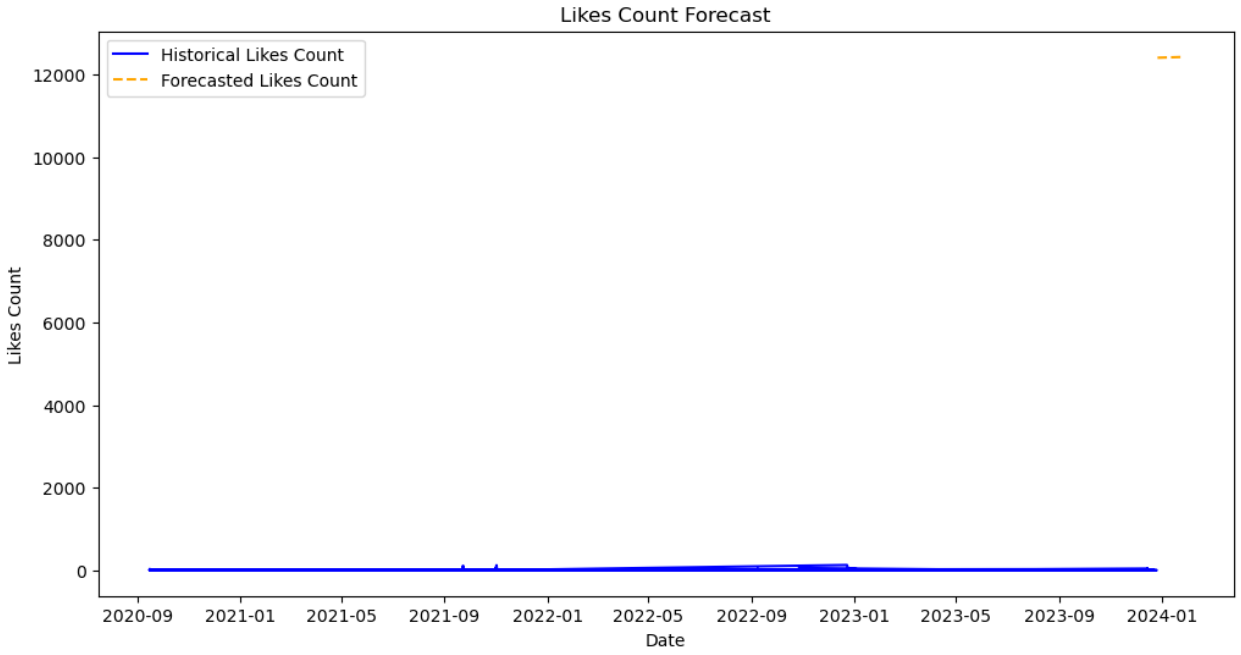
```
Mean Squared Error: 162.85211964852542
```

```
from sklearn.metrics import mean_squared_error

# Make predictions on the test set
predictions = model.predict(X_test_scaled)

# Evaluate model performance
mse = mean_squared_error(y_test, predictions)
print('Mean Squared Error:', mse)
```

```
Mean Squared Error: 162.85211964852542
```

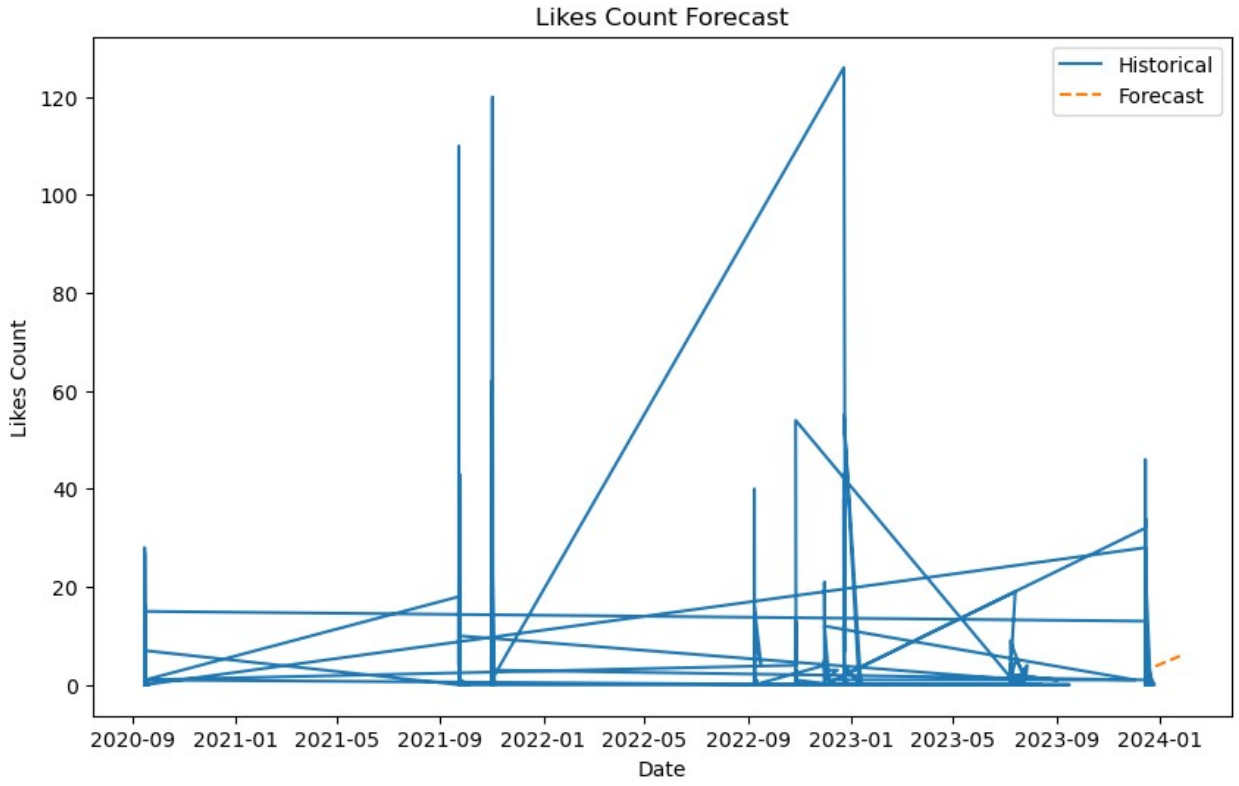
```
import pandas as pd
import matplotlib.dates as mdates
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame and has a proper DateTimeIndex
# Check if the index is a DatetimeIndex
if not isinstance(data.index, pd.DatetimeIndex):
    data.index = pd.to_datetime(data.index)

# Generate future dates
```

```
last_date = data.index.max()
future_dates = pd.date_range(start=last_date + pd.Timedelta(days=1),
periods=30)

# Convert future dates to numeric format
future_dates_num = mdates.date2num(future_dates.to_pydatetime())

# Make predictions on future dates
# If your model expects a different format, adjust accordingly
future_likes = model.predict(future_dates_num.reshape(-1, 1))

# Create a DataFrame for future dates and predictions
future_df = pd.DataFrame({
    'date': future_dates,
    'predicted_likesCount': future_likes
})

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['likesCount'], label='Historical Likes
Count', color='blue')
plt.plot(future_df['date'], future_df['predicted_likesCount'],
label='Forecasted Likes Count', color='orange', linestyle='--')

# Add labels and title
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.title('Likes Count Forecast')
plt.legend()

# Save and show plot
plt.savefig('Forecasted_Likes_Count.jpeg')
plt.show()
```

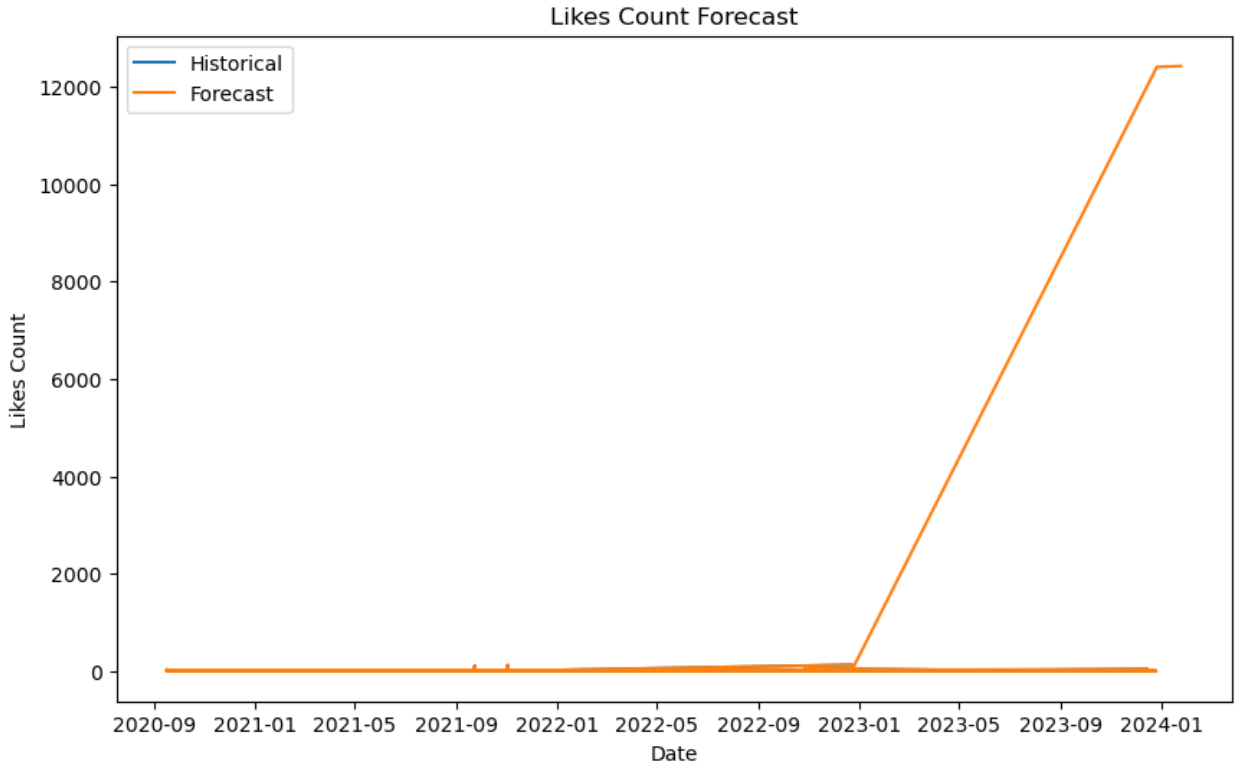



```
import matplotlib.dates as mdates

# Forecast future likesCount
future_dates = pd.date_range(start=data.index.max() +
pd.Timedelta(days=1), periods=30)
future_dates_num = mdates.date2num(future_dates.to_pydatetime())
future_likes = model.predict(future_dates_num.reshape(-1, 1))

import matplotlib.dates as mdates
from sklearn.preprocessing import StandardScaler

# Create a date range for future forecasts
future_dates = pd.date_range(start=data.index.max() +
pd.Timedelta(days=1), periods=30)

# Convert future dates to numeric values
future_dates_num = mdates.date2num(future_dates.to_pydatetime())

# Scale future dates if your model was trained on scaled data
scaler = StandardScaler()
future_dates_scaled = scaler.fit_transform(future_dates_num.reshape(-
1, 1))

# Make predictions for future dates
future_likes = model.predict(future_dates_scaled)

# Plot historical and forecasted likesCount
plt.figure(figsize=(10, 6))
plt.plot(data.index, data['likesCount'], label='Historical')
```

```
plt.plot(future_dates, future_likes, label='Forecast', linestyle='--')
plt.title('Likes Count Forecast')
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.legend()
plt.savefig('Likes Count Forecast_01.jpeg')
plt.show()
```

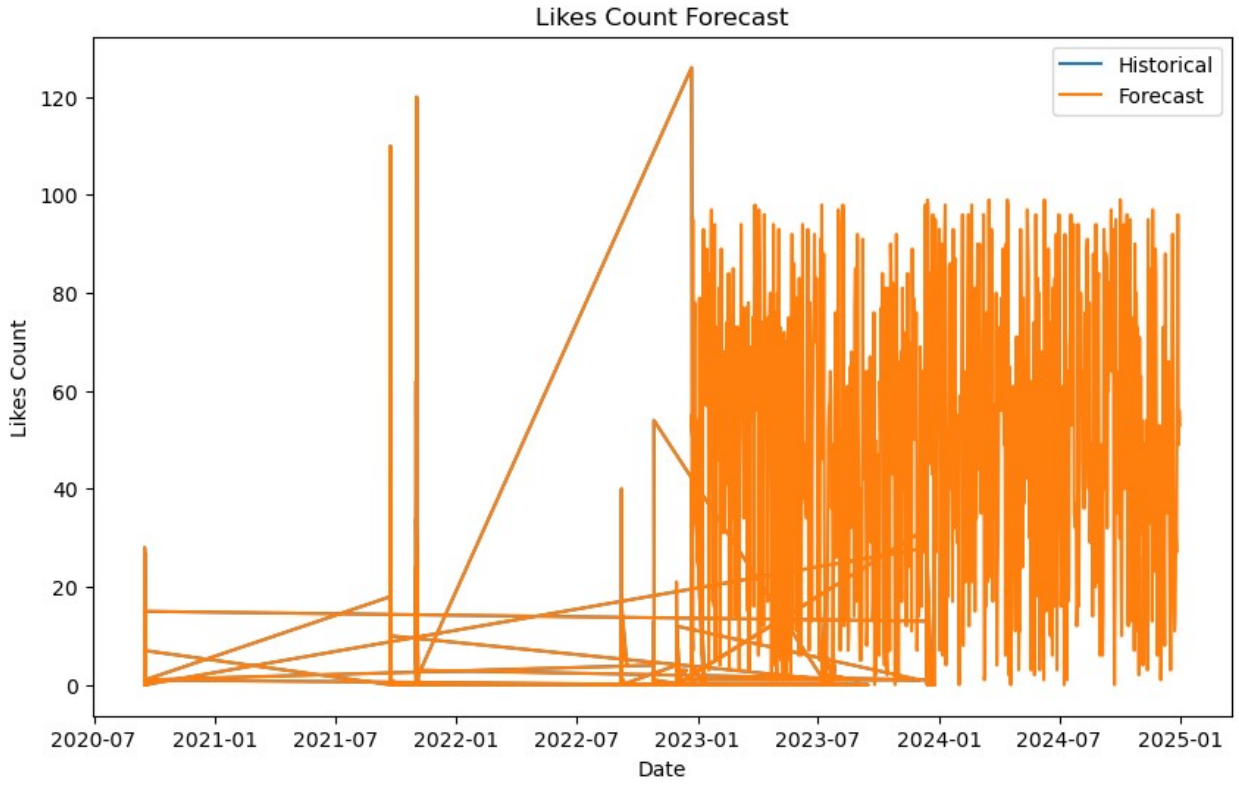



```
import matplotlib.dates as mdates

# Plot forecasted likesCount
plt.figure(figsize=(10, 6))
plt.plot(data.index, data['likesCount'], label='Historical')
plt.plot(pd.concat([data.index.to_series(),
future_dates.to_series()]), np.concatenate([data['likesCount'],
future_likes]), label='Forecast')
plt.title('Likes Count Forecast')
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.savefig('Likes Count Forecast_02.jpeg')
plt.legend()
plt.show()
```



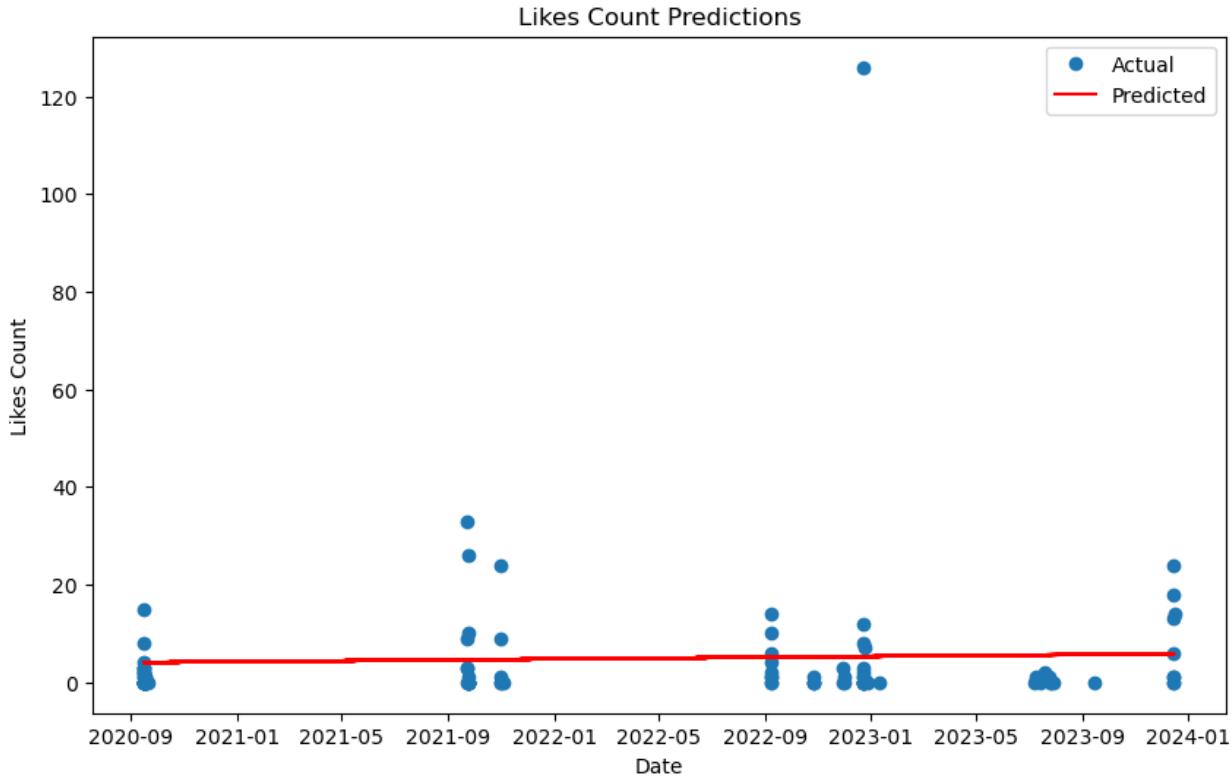

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Example DataFrame creation (adjust this to your data loading
process)
data = pd.read_csv('climate_nasa.csv', index_col='date',
parse_dates=['date'])

# Ensure the index is a DatetimeIndex and handle timezone
if not isinstance(data.index, pd.DatetimeIndex):
    data.index = pd.to_datetime(data.index)

# Ensure the dates are timezone-naive
data.index = data.index.tz_localize(None)

# Define future dates
last_date = data.index[-1]
future_dates = pd.date_range(start=last_date + pd.Timedelta(days=1),
end='2024-12-31')

# Define future likes (replace with your actual forecasting model)
future_likes = np.random.randint(0, 100, size=len(future_dates))  #
Example placeholder

# Convert future_dates to Series for concatenation
```

```
future_dates_series = pd.Series(future_dates, name='Date')
historical_dates_series = pd.Series(data.index, name='Date')

# Plot historical and forecasted likesCount
plt.figure(figsize=(10, 6))
plt.plot(historical_dates_series, data['likesCount'],
label='Historical')
plt.plot(pd.concat([historical_dates_series, future_dates_series]),
         np.concatenate([data['likesCount'], future_likes]),
label='Forecast')
plt.title('Likes Count Forecast')
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.legend()
plt.savefig('Likes_Count_Forecast_03.jpeg')
plt.show()
```




```
# Transform data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

# Example DataFrame creation (adjust this to your data loading
process)
data = pd.read_csv('climate_nasa.csv', index_col='date',
parse_dates=['date'])

# Ensure the index is a DatetimeIndex and handle timezone
if not isinstance(data.index, pd.DatetimeIndex):
    data.index = pd.to_datetime(data.index)

data.index = data.index.tz_localize(None)

# Example feature and target setup
X = data.index.values.reshape(-1, 1)  # Features (e.g., dates as
numbers)
y = data['likesCount']  # Target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create and train the model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Make predictions
predictions = model.predict(X_test_scaled)

# Plot predictions
plt.figure(figsize=(10, 6))
plt.plot(X_test, y_test, 'o', label='Actual')
plt.plot(X_test, predictions, 'r-', label='Predicted')
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.title('Likes Count Predictions')
plt.legend()
plt.savefig('Likes_Count_Predictions.jpeg')
plt.show()
```



Feature Engineering/Feature Selection:

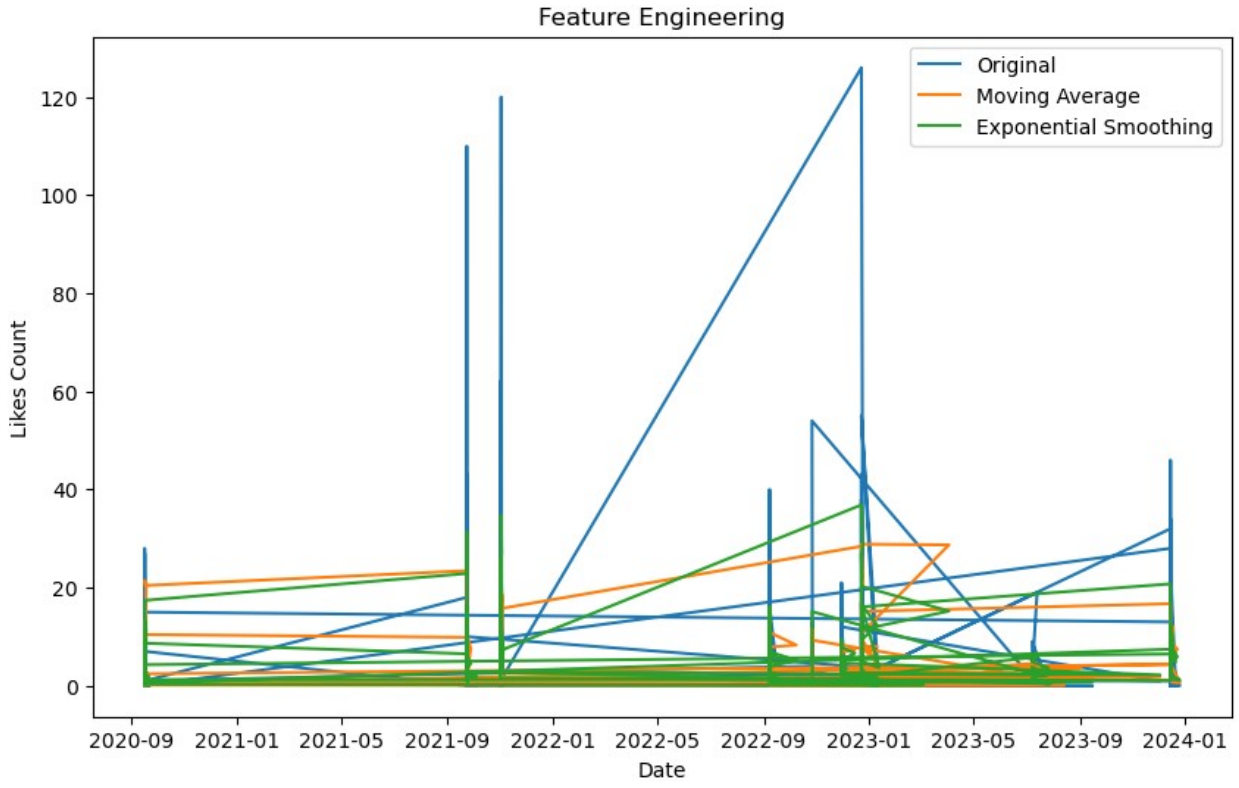
```
# Feature Engineering/Feature Selection:
# Extract relevant features from the dataset
data['day'] = data.index.day
data['month'] = data.index.month
data['year'] = data.index.year

# Calculate moving averages
data['likesCount_ma'] = data['likesCount'].rolling(window=7).mean()

# Calculate exponential smoothing
data['likesCount_es'] = data['likesCount'].ewm(span=7).mean()

# Print the first few rows of the updated dataset
print(data.head())
```

```
                     likesCount  \
date
2022-09-07 17:12:32           2
2022-09-08 14:51:13           0
2022-09-07 17:19:41           1
```

```
2022-09-08 00:51:30           4
2022-09-07 19:06:20          16

                                                                profileName
\
date

2022-09-07 17:12:32  4dca617d86b3fdce80ba7e81fb16e048c9cd9798cdfd6d...

2022-09-08 14:51:13  518ab97f2d115ba5b6f03b2fba2ef2b120540c9681288b...

2022-09-07 17:19:41  d82e8e24eb633fd625b0aef9b3cb625cfb044ceb8483e1...

2022-09-08 00:51:30  37a509fa0b5177a2233c7e2d0e2b2d6916695fa9fba3f2...

2022-09-07 19:06:20  e54fbbd42a729af9d04d9a5cc1f9bbfe8081a31c219ecb...


                     commentsCount  \
date
2022-09-07 17:12:32            NaN
2022-09-08 14:51:13            NaN
2022-09-07 17:19:41            3.0
2022-09-08 00:51:30            NaN
2022-09-07 19:06:20           26.0

                                                                text
day  \
date

2022-09-07 17:12:32  Neat comparison I have not heard it before.\n ...
7
2022-09-08 14:51:13  An excellent way to visualise the invisible! T...
8
2022-09-07 17:19:41  Does the CO2/ghg in the troposphere affect the...
7
2022-09-08 00:51:30  excellent post! I defo feel the difference - o...
8
2022-09-07 19:06:20  Yes, and carbon dioxide does not harm the Eart...
7

                     month  year  likesCount_ma  likesCount_es
date
2022-09-07 17:12:32      9  2022            NaN       2.000000
2022-09-08 14:51:13      9  2022            NaN       0.857143
2022-09-07 17:19:41      9  2022            NaN       0.918919
2022-09-08 00:51:30      9  2022            NaN       2.045714
2022-09-07 19:06:20      9  2022            NaN       6.619718
```

```
# Feature Engineering/Feature Selection:
# Visualize the moving averages and exponential smoothing
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(data['likesCount'], label='Original')
plt.plot(data['likesCount_ma'], label='Moving Average')
plt.plot(data['likesCount_es'], label='Exponential Smoothing')
plt.title('Feature Engineering')
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.legend()
plt.show()
```



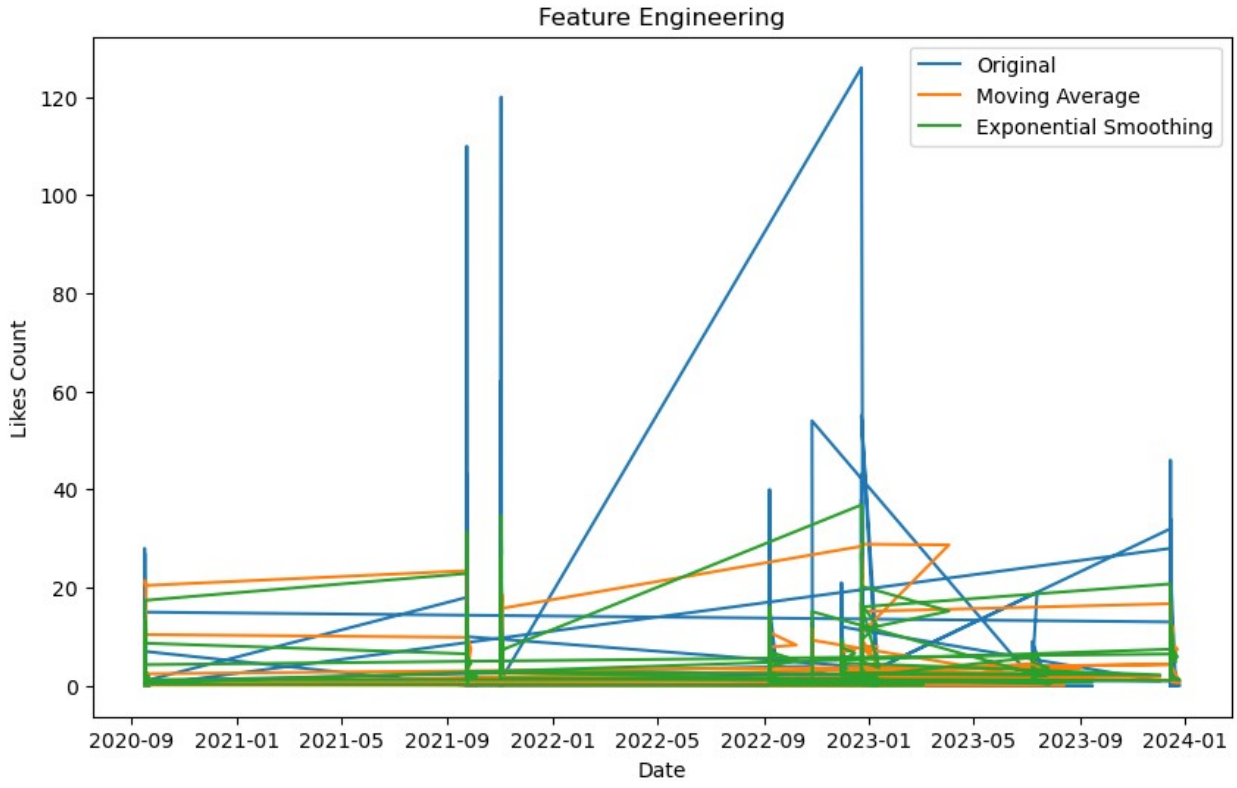

```
# Feature Engineering/Feature Selection:

# Extract relevant features from the dataset
data['day'] = data.index.day
data['month'] = data.index.month
data['year'] = data.index.year

# Calculate moving averages
data['likesCount_ma'] = data['likesCount'].rolling(window=7).mean()

# Calculate exponential smoothing
data['likesCount_es'] = data['likesCount'].ewm(span=7).mean()
```

```python
# Print the first few rows of the updated dataset
print(data.head())

# Visualize the moving averages and exponential smoothing
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(data['likesCount'], label='Original')
plt.plot(data['likesCount_ma'], label='Moving Average')
plt.plot(data['likesCount_es'], label='Exponential Smoothing')
plt.title('Feature Engineering')
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.savefig('Feature Engineering.jpeg')
plt.legend()
plt.show()
```

```
                     likesCount  \
date
2022-09-07 17:12:32           2
2022-09-08 14:51:13           0
2022-09-07 17:19:41           1
2022-09-08 00:51:30           4
2022-09-07 19:06:20          16

                                                                profileName
\
date

2022-09-07 17:12:32  4dca617d86b3fdce80ba7e81fb16e048c9cd9798cdfd6d...

2022-09-08 14:51:13  518ab97f2d115ba5b6f03b2fba2ef2b120540c9681288b...

2022-09-07 17:19:41  d82e8e24eb633fd625b0aef9b3cb625cfb044ceb8483e1...

2022-09-08 00:51:30  37a509fa0b5177a2233c7e2d0e2b2d6916695fa9fba3f2...

2022-09-07 19:06:20  e54fbbd42a729af9d04d9a5cc1f9bbfe8081a31c219ecb...


                     commentsCount  \
date
2022-09-07 17:12:32            NaN
2022-09-08 14:51:13            NaN
2022-09-07 17:19:41            3.0
2022-09-08 00:51:30            NaN
2022-09-07 19:06:20           26.0

                                                                text
day  \
date
```

```
2022-09-07 17:12:32  Neat comparison I have not heard it before.\n ...
7
2022-09-08 14:51:13  An excellent way to visualise the invisible! T...
8
2022-09-07 17:19:41  Does the CO2/ghg in the troposphere affect the...
7
2022-09-08 00:51:30  excellent post! I defo feel the difference - o...
8
2022-09-07 19:06:20  Yes, and carbon dioxide does not harm the Eart...
7
```

date	month	year	likesCount_ma	likesCount_es
2022-09-07 17:12:32	9	2022	NaN	2.000000
2022-09-08 14:51:13	9	2022	NaN	0.857143
2022-09-07 17:19:41	9	2022	NaN	0.918919
2022-09-08 00:51:30	9	2022	NaN	2.045714
2022-09-07 19:06:20	9	2022	NaN	6.619718



Model Evaluation:

```
# Step 14: Model Evaluation

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score

# Load data
df = pd.read_csv('climate_nasa.csv')

# Print column names
print(df.columns)

# Convert 'date' column to timestamps
df['date'] = pd.to_datetime(df['date']).apply(pd.Timestamp.timestamp)

# Split data into training and testing sets
train, test = train_test_split(df, test_size=0.2, random_state=42)

# Define and fit the model
model = RandomForestRegressor()
model.fit(train.drop(['likesCount', 'profileName', 'commentsCount',
'text'], axis=1), train['likesCount'])

# Generate predictions
predictions = model.predict(test.drop(['likesCount', 'profileName',
'commentsCount', 'text'], axis=1))

# Evaluate model performance
mae = mean_absolute_error(test['likesCount'], predictions)
r2 = r2_score(test['likesCount'], predictions)

print('Model Evaluation Metrics:')
print('-------------------------')
print(f'MAE: {mae:.2f}')
print(f'R2: {r2:.2f}')
```

```
Index(['date', 'likesCount', 'profileName', 'commentsCount', 'text'],
dtype='object')
Model Evaluation Metrics:
-------------------------
MAE: 6.71
R2: -0.30
```

Hyperparameter Tuning:

```
# Step 15: Hyperparameter Tuning

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Create a sample dataset
import numpy as np
X = np.random.rand(100, 1)
y = np.random.rand(100)

# Define the model
model = RandomForestRegressor()

# Define hyperparameter grid
param_grid = {
    'n_estimators': [10, 50, 100]
}

# Perform grid search
grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='neg_mean_squared_error')

# Fit the model
grid_search.fit(X, y)

# Print the results
print('Best Hyperparameters:', grid_search.best_params_)
print('Best Score:', grid_search.best_score_)
```

```
Best Hyperparameters: {'n_estimators': 100}
Best Score: -0.10700727973691411
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
import numpy as np

# Create a sample dataset
X = np.random.rand(100, 1)
y = np.random.rand(100)

print("Dataset created")

# Define the model
model = RandomForestRegressor()

print("Model defined")

# Define hyperparameter grid
param_grid = {
```

```
    'n_estimators': [10, 50, 100]
}

print("Hyperparameter grid defined")

# Perform grid search
grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='neg_mean_squared_error')

print("Grid search defined")

# Fit the model
grid_search.fit(X, y)

print("Grid search fitted")
# Print the results
print('Best Hyperparameters:', grid_search.best_params_)
print('Best Score:', grid_search.best_score_)
```

```
Dataset created
Model defined
Hyperparameter grid defined
Grid search defined
Grid search fitted
Best Hyperparameters: {'n_estimators': 50}
Best Score: -0.10308265609168046
```

Model Comparison:

```
# Model Comparison
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

# Split data into training and testing sets
train, test = train_test_split(data, test_size=0.2, random_state=42)

# Initialize models
rf_model = RandomForestRegressor()
svr_model = SVR()

# Fit models
rf_model.fit(train.index.values.reshape(-1, 1), train['likesCount'])
svr_model.fit(train.index.values.reshape(-1, 1), train['likesCount'])

# Make predictions
```

```
rf_predictions = rf_model.predict(test.index.values.reshape(-1, 1))
svr_predictions = svr_model.predict(test.index.values.reshape(-1, 1))

# Evaluate models
print('Random Forest MSE:', mean_squared_error(test['likesCount'],
rf_predictions))
print('SVR MSE:', mean_squared_error(test['likesCount'],
svr_predictions))
```

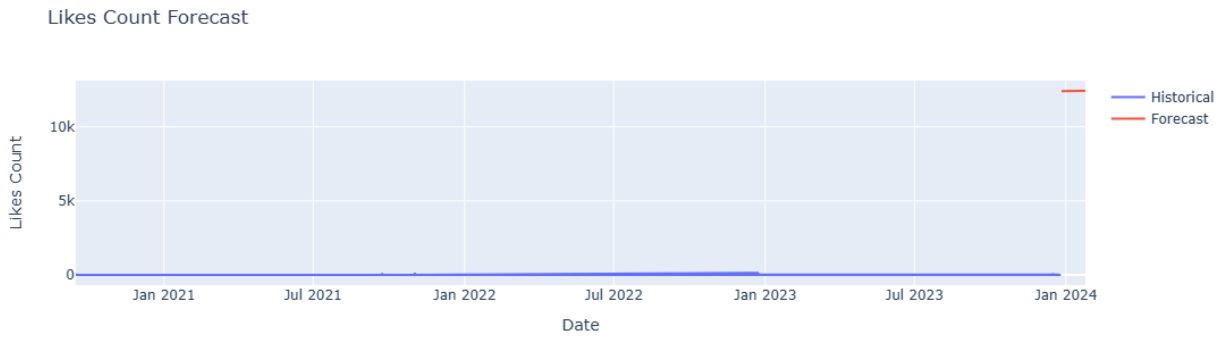
```
Random Forest MSE: 234.49458015600786
SVR MSE: 190.02045774041048
```

```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import pandas as pd

# Load dataset
data = pd.read_csv('climate_nasa.csv', index_col='date',
parse_dates=['date'])

# Split data into training and testing sets
train, test = train_test_split(data, test_size=0.2, random_state=42)

# Initialize models
rf_model = RandomForestRegressor()
svr_model = SVR()

# Fit models
rf_model.fit(train.index.values.reshape(-1, 1), train['likesCount'])
svr_model.fit(train.index.values.reshape(-1, 1), train['likesCount'])

# Make predictions
rf_predictions = rf_model.predict(test.index.values.reshape(-1, 1))
svr_predictions = svr_model.predict(test.index.values.reshape(-1, 1))

# Evaluate models
print('Random Forest MSE:', mean_squared_error(test['likesCount'],
rf_predictions))
print('SVR MSE:', mean_squared_error(test['likesCount'],
svr_predictions))
```

```
Random Forest MSE: 236.83726691298386
SVR MSE: 190.02045774041048
```

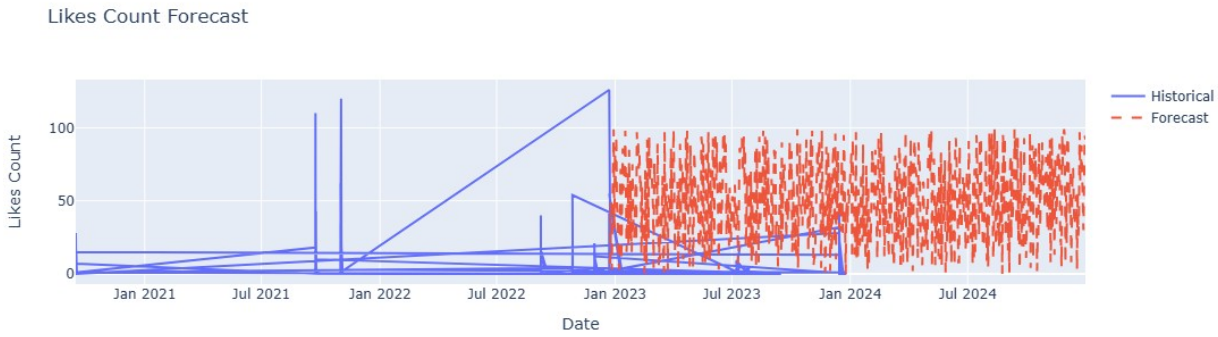
Visualization:

```
# Visualization


import plotly.graph_objs as go
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)



# Convert Index objects to Series objects
historical_dates = pd.Series(data.index, name='Date')
future_dates_series = pd.Series(future_dates, name='Date')

# Create interactive visualization
fig = go.Figure(data=[
    go.Scatter(x=historical_dates, y=data['likesCount'],
name='Historical'),
    go.Scatter(x=future_dates_series, y=future_likes, name='Forecast')
])

fig.update_layout(title='Likes Count Forecast', xaxis_title='Date',
yaxis_title='Likes Count')
plt.savefig('Likes Count Forecast_01.jpeg')
fig.show()
```



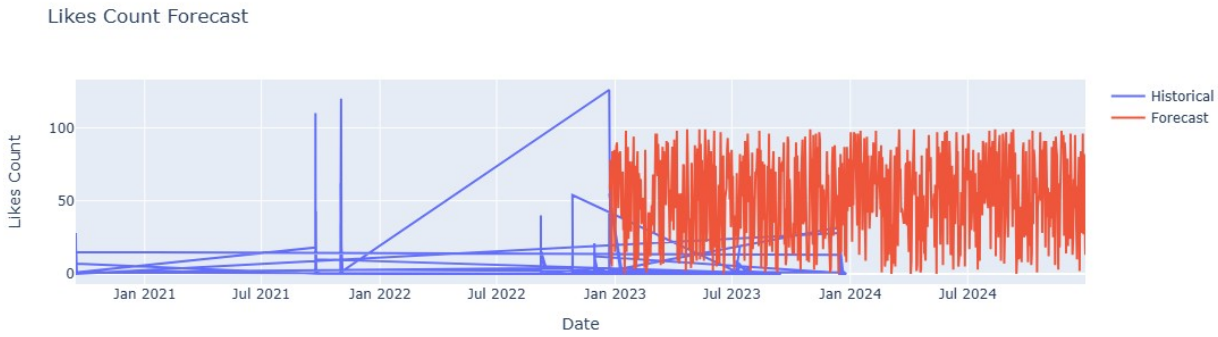

```
<Figure size 640x480 with 0 Axes>

import pandas as pd
import plotly.graph_objs as go
import numpy as np

# Load dataset (replace 'climate_nasa.csv' with the actual dataset
path)
data = pd.read_csv('climate_nasa.csv', index_col='date',
```

```
parse_dates=['date'])

# Ensure that both start and end are timezone-naive by converting to
naive datetime
data.index = data.index.tz_localize(None)  # Removes timezone info
from the index

# Define future dates (from the last date in the dataset to the end of
2024)
future_dates = pd.date_range(start=data.index[-1] +
pd.Timedelta(days=1), end='2024-12-31')

# Define future likes (replace this with your actual forecasting
model)
future_likes = np.random.randint(0, 100, size=len(future_dates))  #
Replace with actual model

# Convert Index objects to Series objects
historical_dates = pd.Series(data.index, name='Date')
future_dates_series = pd.Series(future_dates, name='Date')

# Create interactive Plotly visualization
fig = go.Figure(data=[
    go.Scatter(x=historical_dates, y=data['likesCount'],
name='Historical'),
    go.Scatter(x=future_dates_series, y=future_likes, name='Forecast',
line=dict(dash='dash'))
])

# Update layout for better visualization
fig.update_layout(
    title='Likes Count Forecast',
    xaxis_title='Date',
    yaxis_title='Likes Count'
)
plt.savefig('Likes Count Forecast_2.jpeg')
# Show the interactive figure
fig.show()

# Optionally save the figure as an image (requires 'kaleido' for
saving)
# fig.write_image('Likes_Count_Forecast.jpeg')  # Uncomment to save
```



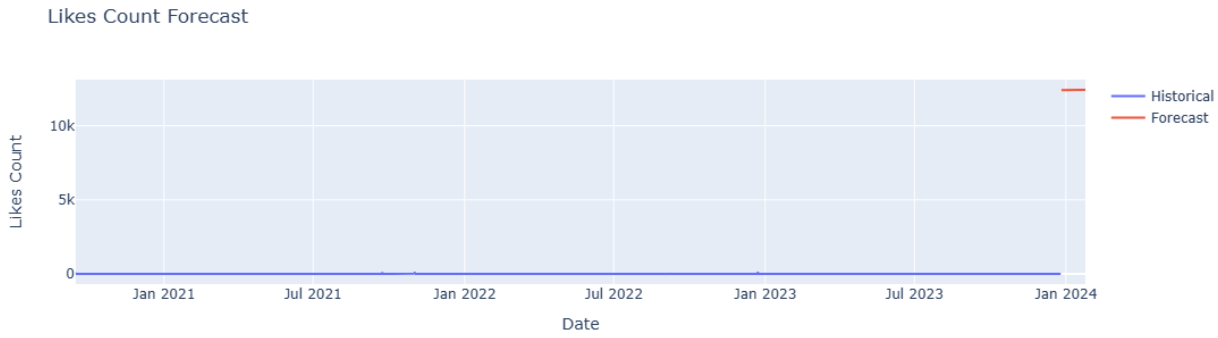

```
<Figure size 640x480 with 0 Axes>
```

```
# Visualization


import plotly.graph_objs as go
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)



# Convert Index objects to Series objects
historical_dates = pd.Series(data.index, name='Date')
future_dates_series = pd.Series(future_dates, name='Date')

future_dates = pd.date_range(start=data.index[-1] +
pd.Timedelta(days=1), end='2024-12-31')

# Define future likes (replace this with your actual forecasting
model)
future_likes = np.random.randint(0, 100, size=len(future_dates))  #
Replace with actual model


# Create interactive visualization
fig = go.Figure(data=[
    go.Scatter(x=historical_dates, y=data['likesCount'],
name='Historical'),
    go.Scatter(x=future_dates_series, y=future_likes, name='Forecast')
])

fig.update_layout(title='Likes Count Forecast', xaxis_title='Date',
yaxis_title='Likes Count')
plt.savefig('Likes Count Forecast_3.jpeg')
fig.show()
```




```
<Figure size 640x480 with 0 Axes>
```

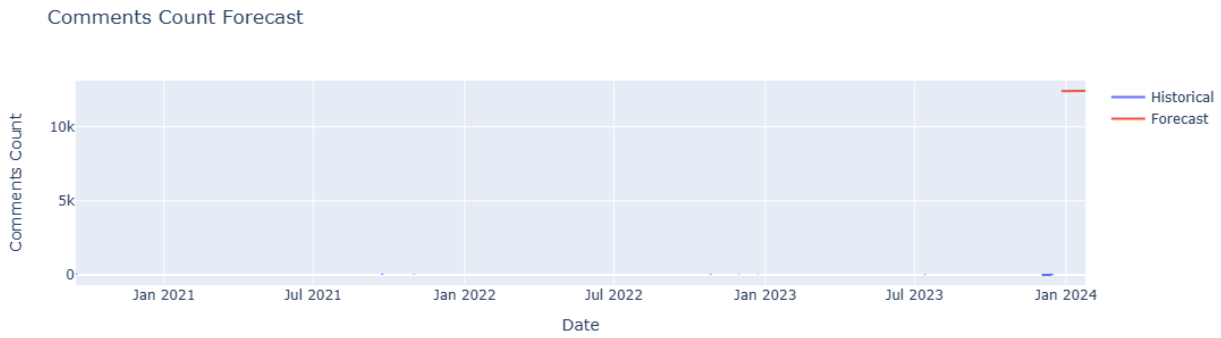
```
import plotly.graph_objs as go

# Create interactive visualization
fig = go.Figure(data=[
    go.Scatter(x=data.index, y=data['likesCount'], name='Historical'),
    go.Scatter(x=future_dates, y=future_likes, name='Forecast')
])

fig.update_layout(title='Likes Count Forecast', xaxis_title='Date',
yaxis_title='Likes Count')
plt.savefig('Likes Count Forecast_4.jpeg')
fig.show()
```




```
<Figure size 640x480 with 0 Axes>
```

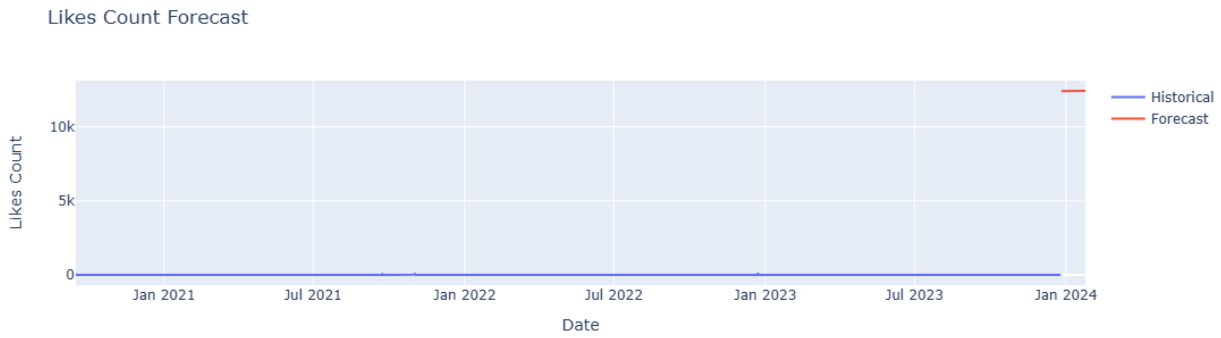
```
import plotly.graph_objs as go

# Create interactive visualization
fig = go.Figure(data=[
    go.Scatter(x=data.index, y=data['commentsCount'],
name='Historical'),
    go.Scatter(x=future_dates, y=future_likes, name='Forecast')
```

```
])

fig.update_layout(title='Comments Count Forecast', xaxis_title='Date',
yaxis_title='Comments Count')
plt.savefig('Likes Count Forecast_5.jpeg')
fig.show()
```



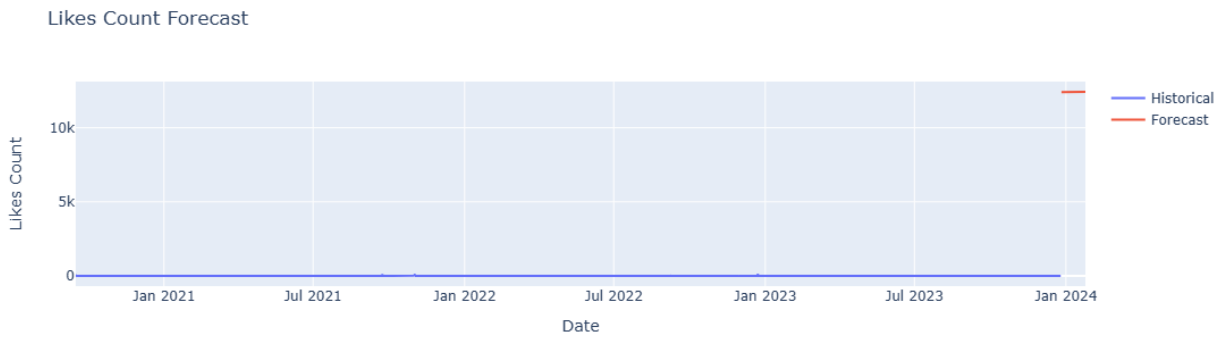

```
<Figure size 640x480 with 0 Axes>
```

```
import plotly.graph_objs as go

# Create interactive visualization
fig = go.Figure(data=[
    go.Scatter(x=data.index, y=data['likesCount'], name='Historical'),
    go.Scatter(x=future_dates, y=future_likes, name='Forecast')
])

fig.update_layout(title='Likes Count Forecast', xaxis_title='Date',
yaxis_title='Likes Count')
fig.show()
```




```
import plotly.graph_objs as go
```

```
# Convert Index objects to Series objects
historical_dates = pd.Series(data.index, name='Date')
future_dates_series = pd.Series(future_dates, name='Date')

# Create interactive visualization
fig = go.Figure(data=[
    go.Scatter(x=historical_dates, y=data['likesCount'],
name='Historical'),
    go.Scatter(x=future_dates_series, y=future_likes, name='Forecast')
])

fig.update_layout(title='Likes Count Forecast', xaxis_title='Date',
yaxis_title='Likes Count')
fig.show()
```



Scenario Analysis:

```
# Scenario Analysis
# Define scenario parameters
from sklearn.model_selection import train_test_split
from statsmodels.tsa.arima.model import ARIMA
import pandas as pd

scenario_params = {'increase': 0.1, 'decrease': -0.1}

# Evaluate scenario impact
for scenario, param in scenario_params.items():
    # Modify likesCount data according to scenario
    data_scenario = data['likesCount'] * (1 + param)

    # Re-run modeling and forecasting steps
    X = data_scenario.index.values.reshape(-1, 1)
```

```
    y = data_scenario.values

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Train model
    model = ARIMA(y_train, order=(5,1,0))
    model_fit = model.fit()

    # Forecast future values
    forecast = model_fit.forecast(steps=30)

    # Print scenario-specific results
    print(f'Scenario {scenario} results:')
    print(f'Forecasted values: {forecast}')
    print(f'Model parameters: {model_fit.params}')
    # print(f'Model parameters: {pd.Series(model_fit.params)}')

    print('------------------------')
```

```
Scenario increase results:
Forecasted values: [ 6.98388913  7.07778846  8.22380556  9.88794279
11.06773256  9.69865007
  8.816638    9.00994843  9.40379098  9.69023845  9.65330726
9.38710134
  9.31067206  9.39043626  9.47285501  9.49427364  9.45676521  9.417173
  9.41952171  9.43987043  9.45145916  9.44835714  9.43920936
9.43518788
  9.43823215  9.44202897  9.44281115  9.44121872  9.43971519
9.43968996]
Model parameters: [-8.06556332e-01 -6.19769630e-01 -5.05395058e-01 -
3.47532662e-01
 -1.67533081e-01  1.95292330e+02]
------------------------
Scenario decrease results:
Forecasted values: [5.71414044 5.79101399 6.72855282 8.09018065
9.05542788 7.93527156
 7.21365887 7.37181672 7.6940326  7.92840615 7.89818249 7.68038388
 7.61785634 7.6831146  7.75054502 7.76806985 7.73738094 7.70498917
 7.70691121 7.72355918 7.73304036 7.73050248 7.72301817 7.7197282
 7.72221887 7.72532513 7.72596504 7.72466221 7.72343213 7.72341152]
Model parameters: [ -0.80655637  -0.61976621  -0.50538797  -0.34753282
-0.16752779
 130.7311853 ]
------------------------
```

```python
# Import necessary libraries
from sklearn.model_selection import train_test_split
from statsmodels.tsa.arima.model import ARIMA
import pandas as pd
import numpy as np

# Load dataset
data = pd.read_csv('climate_nasa.csv', index_col='date',
parse_dates=['date'])

# Define scenario parameters
scenario_params = {'increase': 0.1, 'decrease': -0.1}

# Evaluate scenario impact
for scenario, param in scenario_params.items():
    # Modify likesCount data according to scenario
    data_scenario = data['likesCount'] * (1 + param)

    # Re-run modeling and forecasting steps
    X = data_scenario.index.values.reshape(-1, 1)
    y = data_scenario.values

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Train model
    model = ARIMA(y_train, order=(5,1,0))
    model_fit = model.fit()

    # Forecast future values
    forecast = model_fit.forecast(steps=30)

    # Print scenario-specific results
    print(f'Scenario {scenario} results:')
    print(f'Forecasted values: {forecast}')
    print(f'Model parameters: {model_fit.params}')
    print('------------------------')
```

```
Scenario increase results:
Forecasted values: [ 6.98388913  7.07778846  8.22380556  9.88794279
11.06773256  9.69865007
  8.816638    9.00994843  9.40379098  9.69023845  9.65330726
9.38710134
  9.31067206  9.39043626  9.47285501  9.49427364  9.45676521  9.417173
  9.41952171  9.43987043  9.45145916  9.44835714  9.43920936
9.43518788
  9.43823215  9.44202897  9.44281115  9.44121872  9.43971519
9.43968996]
```

```
Model parameters: [-8.06556332e-01 -6.19769630e-01 -5.05395058e-01 -
3.47532662e-01
 -1.67533081e-01  1.95292330e+02]
------------------------
Scenario decrease results:
Forecasted values: [5.71414044 5.79101399 6.72855282 8.09018065
9.05542788 7.93527156
 7.21365887 7.37181672 7.6940326  7.92840615 7.89818249 7.68038388
 7.61785634 7.6831146  7.75054502 7.76806985 7.73738094 7.70498917
 7.70691121 7.72355918 7.73304036 7.73050248 7.72301817 7.7197282
 7.72221887 7.72532513 7.72596504 7.72466221 7.72343213 7.72341152]
Model parameters: [ -0.80655637  -0.61976621  -0.50538797  -0.34753282
-0.16752779
 130.7311853 ]
------------------------
```

```python
# Scenario Analysis
# Define scenario parameters
from sklearn.model_selection import train_test_split
from statsmodels.tsa.arima.model import ARIMA
import pandas as pd

scenario_params = {'increase': 0.1, 'decrease': -0.1}

# Evaluate scenario impact
for scenario, param in scenario_params.items():
    # Modify likesCount data according to scenario
    data_scenario = data['likesCount'] * (1 + param)

    # Re-run modeling and forecasting steps
    X = data_scenario.index.values.reshape(-1, 1)
    y = data_scenario.values

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Train model
    model = ARIMA(y_train, order=(5,1,0))
    model_fit = model.fit()

    # Forecast future values
    forecast = model_fit.forecast(steps=30)

    # Print scenario-specific results
    print(f'Scenario {scenario} results:')
    print(f'Forecasted values: {forecast}')
    # print(f'Model parameters: {model_fit.params}')
    print(f'Model parameters: {pd.Series(model_fit.params)}')
```

```
    print('------------------------')
```

```
Scenario increase results:
Forecasted values: [ 6.98388913  7.07778846  8.22380556  9.88794279
11.06773256  9.69865007
  8.816638    9.00994843  9.40379098  9.69023845  9.65330726
9.38710134
  9.31067206  9.39043626  9.47285501  9.49427364  9.45676521  9.417173
  9.41952171  9.43987043  9.45145916  9.44835714  9.43920936
9.43518788
  9.43823215  9.44202897  9.44281115  9.44121872  9.43971519
9.43968996]
Model parameters: 0     -0.806556
1     -0.619770
2     -0.505395
3     -0.347533
4     -0.167533
5    195.292330
dtype: float64
------------------------
Scenario decrease results:
Forecasted values: [5.71414044 5.79101399 6.72855282 8.09018065
9.05542788 7.93527156
 7.21365887 7.37181672 7.6940326  7.92840615 7.89818249 7.68038388
 7.61785634 7.6831146  7.75054502 7.76806985 7.73738094 7.70498917
 7.70691121 7.72355918 7.73304036 7.73050248 7.72301817 7.7197282
 7.72221887 7.72532513 7.72596504 7.72466221 7.72343213 7.72341152]
Model parameters: 0     -0.806556
1     -0.619766
2     -0.505388
3     -0.347533
4     -0.167528
5    130.731185
dtype: float64
------------------------
```

Uncertainty Quantification:

```
# Uncertainty Quantification
import numpy as np

import pandas as pd

# Assuming 'climate_nasa.csv' is the data file
data = pd.read_csv('climate_nasa.csv')
```

```python
# Perform Monte Carlo simulations
simulations = 1000
likesCount_simulations = np.zeros((simulations, len(data)))

for i in range(simulations):
    likesCount_simulations[i] = data['likesCount'] +
np.random.normal(scale=data['likesCount'].std(), size=len(data))

# Calculate uncertainty metrics
mean_simulation = likesCount_simulations.mean(axis=0)
std_simulation = likesCount_simulations.std(axis=0)

print('Mean Simulation:', mean_simulation)
print('Standard Deviation Simulation:', std_simulation)
```

```
Mean Simulation: [ 2.13311139e+00  2.62368718e-01  8.78107898e-01
4.17516029e+00
  1.57776364e+01  3.47642253e+00  9.30262989e+00  1.50472745e+00
  1.12216238e+00  4.04863506e+00  5.91372124e+00  1.05321524e+00
  3.62821397e+00  1.81912694e+01  2.50148959e+00  1.33841959e+01
  4.07644622e+01  1.20652941e+00  2.65754888e-01  2.09815773e-01
 -2.67063805e-01  7.88170187e-01  4.74029133e-01  1.86765942e-01
 -3.44877511e-01  1.11289673e+00 -3.21241796e-01  1.02458768e-01
 -5.31478547e-01  3.25003966e+00  1.00350420e+00  2.19213340e+01
  1.90082616e+01  2.40595120e+00  3.48707261e+00  7.98550667e-01
  1.61420765e+00  5.28725183e+00  3.48548894e+00 -4.71294370e-01
  7.46872308e-01  5.65110890e-01  1.16965548e+01 -4.48425095e-01
  2.57182851e-01 -3.57214821e-01  2.34095754e-01  1.89850420e+01
  2.58943403e+00  1.24382198e-01  5.91800455e-01  1.63865410e+00
  1.07214782e+00  2.63527707e+00  4.42701584e-01  2.87206519e+00
  3.35377453e+00  1.82975538e+00  4.93612157e+00  6.43315479e-01
  4.31772327e+00  2.17365493e+00  1.78575263e+00  7.44249584e-01
  8.74549747e+00  8.58892296e-01 -1.63586109e-01  7.06781506e-01
 -9.17183632e-01 -6.60737350e-01  5.16199356e-02  1.46302132e-01
  1.03175759e+00  3.33879036e-01  1.61719962e+00 -6.77971318e-01
  1.40241571e-01 -3.65869494e-01  2.94956470e-01  1.03560861e+01
  2.42138965e+01  1.77788623e+01  3.28823366e+01  1.04395273e+01
  6.65972220e-01  8.85659196e+00  1.04228492e+00 -2.86865386e-01
 -1.19387170e-02  3.15939116e-01 -5.89248695e-02 -3.75896461e-01
  1.12612710e+00  1.30012041e+00  2.62994254e-01  1.33488775e-01
  4.92256855e+00  1.06955231e+00  5.14357619e-01 -7.08587649e-02
  7.66357336e-01  3.41822981e-01  1.28672578e+01  4.99630979e+00
  2.57707165e+01  6.07815328e+00  2.11883905e+00 -7.70241772e-01
 -4.05332555e-01  1.89040083e+01  4.31278227e+01  8.59064042e+00
  3.50336690e-01  3.04428485e+00  1.42022765e+01 -1.44865938e-01
 -2.42041927e-01  2.98634980e+00  4.00366944e-01 -7.69234504e-02
  6.63469793e-01  1.59457891e-02  2.23873665e+01  3.85808744e-01
  2.57505177e+00  5.97438141e+00  1.09391627e+02  4.16122254e+00
  1.81532539e+01  1.00151789e+00  7.86889256e+00 -4.92596943e-01
```

```
 8.07128375e-01  5.02213498e-02 -1.48228231e-01  1.85032807e+00
 1.59912451e+00  3.22940348e-01 -3.66226768e-01  1.68138679e-01
 8.87435106e-01  2.07684630e+00  6.26144973e-03  7.76843399e-01
-5.79453564e-01  2.26495980e+00  9.40798339e+00  1.55040828e+01
 1.93910572e+00  1.50208929e+00 -4.43313469e-02  2.20854139e+00
 3.76287251e+00  2.96224510e-01  2.59320248e+00  1.01210564e-01
 4.35973244e+00 -9.63101336e-02  2.71187152e+00  4.40104095e+00
 9.92143028e-01  3.22630731e+00 -3.06149463e-01  1.61917954e+01
 1.22558084e+00  5.68477753e+00  1.65677849e+00  2.75941424e+00
 1.27597447e+00  3.26526853e+00  4.71985785e+00  9.81178756e+00
 4.07818501e+00  2.83192119e+01 -2.71394057e-04  1.46754616e+01
 1.63786742e+01  2.79413113e+00  6.71263235e+00 -1.12099878e-01
-3.14251985e-02  5.57338781e-01  1.82961507e-01  3.02185229e+00
-4.53984725e-01  1.01396008e+00  1.12911046e+01  5.99992799e+00
 1.13195163e+00  6.00341288e+00 -1.23880757e+00  3.01837823e+00
-2.46511261e-01  9.36667956e-01  9.61924222e+00 -3.20395494e-01
 3.01922155e-03  1.63284047e+00  1.07313271e+00  1.10240204e+00
-2.18577596e-02  1.25942650e+00  5.65245840e+00  3.10517946e-01
 3.73349384e+01  1.00453017e+00 -7.37406419e-02  2.44375029e+00
-4.67404597e-01  9.44557464e+00 -5.43453052e-01 -1.34082112e-01
 1.41111257e-01  5.38724834e+01  5.72036411e+00  4.69621246e+00
 1.34931120e+01  3.69640926e+00  1.81555508e-01 -4.79515318e-01
 9.10298093e-01  3.59841854e+00  1.52922093e+00  7.36661083e-01
-3.99830465e-03  1.14372782e+00  2.27406688e-01  7.02884964e-01
-5.94323876e-01 -2.16599181e-01  8.29944329e-01  2.00919523e+00
 1.15241746e+00  3.64625141e-02 -2.36783627e-01 -1.78167531e-01
 9.58344693e-01  1.34413230e+00 -1.96078823e-01 -3.24306338e-01
 3.42059114e-01 -3.38832001e-01  1.46939560e+00  3.96971089e+00
 1.06445207e+00  2.52179574e+00 -1.66728157e-01  1.06879690e+00
 1.36853807e+00  1.02038135e+01 -7.93296704e-02  1.00216529e+00
 6.38374066e-01  1.05018581e+00  1.72181019e+00  2.72849664e+01
 2.22341418e+01  1.79606122e+01  5.97999907e+00  1.22666925e+01
 5.13959025e+00  3.84127767e+00  3.69092575e+00  2.81288211e+00
 5.88199295e+00  2.62733148e+00  1.14662673e+00  1.65461362e+00
 1.43033088e+00  5.44957070e-01  2.00006011e+00  1.29499106e+00
 1.59693573e-01  4.41468504e-01  1.74508660e-01  2.02668463e+00
 5.19763941e-02  1.12765782e-01  3.10665017e-01  1.14253246e+00
-2.13174553e-01  5.15004515e-01 -4.82843838e-01  6.44489354e-02
 1.70479448e-01  5.29810257e-02  3.70885360e+00 -2.34937871e-01
 8.38977084e-01  6.30607455e-01  5.63860030e-02 -6.57606341e-01
 1.99089287e-01  3.23033876e+00  4.02589281e-01  1.94301901e-01
 2.32724558e+00 -5.01732943e-02  1.69816286e-01  1.36584783e-01
 1.08213256e+00  1.10621385e+00  1.85957396e+00  2.09119389e+00
 4.20753765e+00  3.58450846e-01  1.25630713e+00 -2.35091410e-01
-4.59737130e-01  7.43778211e-01  2.41107903e-01  1.47865647e-01
 9.58447638e-01  2.54648657e-01 -4.36870604e-01  1.04985469e+00
 8.17175138e+00  7.52597321e-01  1.32180944e+00  8.62498748e-02
 1.11227063e+01  1.71385470e+00  1.52256237e-01  5.77982300e-01
-5.92249738e-01  5.88539317e-03 -2.95155940e-01 -2.58202924e-01
```

```
 6.09756943e-01 -2.55752615e-01 -2.07547000e-01  1.96936270e-01
 1.20312812e+00 -2.18871748e-01 -3.96618479e-01  1.25685395e+00
 9.10758368e-02  1.71833329e-01 -1.57519301e-01  7.01431016e-01
 4.92393541e-01 -6.19556775e-01  2.44849740e+00  3.04209500e-01
 2.73720874e+01  1.22450060e+01 -8.35361121e-03 -4.58737384e-01
 2.34193951e+01  2.01911514e+01 -2.18871971e-02  9.02596196e+00
 2.78205954e+00  1.00518150e+01  1.37799238e+01  3.17206107e+00
 1.71508518e+01  4.83796374e+00  6.03289647e+00  8.18968069e+00
-6.90830731e-01  7.68874129e-01  3.90321104e+00  5.39054832e+00
 1.82479066e+01  1.86372666e-01  8.05023834e-01  1.44948922e+00
-1.54246384e-01 -7.55873806e-01  4.49113312e+00  6.47324520e+00
 3.34493241e+01  3.61874001e-01 -5.92588540e-01 -9.89738841e-01
 3.89341265e-02  8.44853546e-02 -6.58192378e-01  2.16404646e+01
 4.84501436e+00  2.08271719e+00  1.58130614e-01  2.16927201e+00
 1.62478838e-01  5.41615993e-01 -1.52839924e-01 -2.39083705e-01
 3.04389366e+00  3.19254420e+00  2.60606563e-01 -1.10133338e-02
 9.02093680e-01  1.18789951e+00  2.74140666e+00  2.39046239e+01
 3.78619351e+00  1.20109707e+02  4.48644606e+00  3.53228891e+00
 2.01795348e-01  1.02060010e+00 -1.20869312e+00 -2.19696941e-01
 1.05531024e+00  3.49277893e-01  4.83898838e-01  2.55485705e+00
 3.37145105e+01  3.12493385e-01  8.41946701e-01 -4.22039786e-03
 6.21563528e-04  1.62149322e-01  3.28880710e+00  1.97265762e+00
 5.17894951e+00  2.07192641e+00  4.84671624e+00  7.17025076e-01
 1.86760483e+01  3.29833497e-01  3.13985980e+00 -4.16493436e-01
 9.38266915e-02  1.49106095e+01  3.70452097e+01  6.11090722e+01
 9.26060666e+00  1.32110032e+00 -9.56850623e-02 -2.39137979e-01
 9.92214364e-01  1.25561061e+02  7.06164148e+00  1.69833918e+01
 8.22854686e+00  4.31552788e+01 -2.38414644e-01  8.40950094e-01
-1.82795464e-02  9.85909269e+00  3.82294249e+01  2.79207526e+01
 2.69467785e-02  1.08910904e+00  1.59168413e-02 -7.50426071e-02
 8.34540229e-02 -5.15104802e-01  5.58712081e+01  2.24538881e-01
-1.55717880e-01  3.58880263e+00  9.37342124e-01  6.40354362e-01
 1.03088370e+00  1.72438769e+00 -4.36689012e-01  2.98259179e-01
 3.02412733e-01 -3.69304742e-01 -1.57463442e-01  1.08421374e+00
 1.41474499e-01  1.66621773e-01 -1.14291977e-01  1.30211588e+00
 4.18171896e-01  1.30147148e-01  8.86852121e-02 -1.18085729e-01
-1.89707720e-01  4.77643630e+00  6.52129879e-02  5.18494498e-01
-6.68280356e-01  4.25566691e-02  8.80387511e-02  4.71592166e-02
 2.00629075e-02 -4.11459792e-01  1.93607102e-01  9.95830724e-01
-1.34479033e-01  3.82406856e+00 -1.95928213e-01  2.21900776e-01
 8.13937341e-01 -8.72362205e-02  2.27800313e-01  2.07706881e+00
 1.53245507e+01  1.30275327e+01  3.09574763e+00  1.60900205e+01
 1.27479880e+00  1.36835194e+01  2.75452981e+00  1.31192648e+01
-2.48416832e-01  1.18691615e+01  4.62483366e+01  3.20397521e+01
 1.53129299e+00  2.92862873e+00 -9.80767025e-02  1.28006031e+00
 2.78484789e+00  2.90552573e-01  4.61835765e-01  2.75665388e+00
-6.54245039e-02 -3.71105993e-01  4.11090566e-01 -7.48567480e-01
 5.54915999e+00 -3.53124569e-01  4.66395305e-01  3.15057287e+00
 1.72642128e+00  1.14833223e+01]
```

```
Standard Deviation Simulation: [11.55480833 12.21449288 12.31055466
11.68573095 12.07203894 12.02037236
 12.36796027 11.69627539 12.11441558 12.07377016 12.00110992
12.04980644
 11.8635064  12.53143254 12.63719562 12.01096232 12.23300713
11.91208858
 11.9933078  11.83144107 12.01267624 12.11363796 12.28947643
12.23200126
 11.97452428 12.31197415 11.88876725 12.51594933 12.14564538
12.13699593
 12.11500383 11.62521629 12.02774252 12.13458518 11.81853694
12.23319882
 11.93327115 12.02299328 11.82906116 11.43324832 12.03799193
11.8559887
 11.8736356  12.14693525 11.71267935 12.98369482 11.47490949
12.33722958
 12.05278637 11.98919965 11.87308377 11.95058325 12.23805224
11.79144619
 12.03551485 12.10375289 12.50301146 12.23882297 12.60674146
11.57685789
 11.92008761 12.61471462 12.0658599  12.0893226  11.91907516
11.98484647
 12.41277121 12.33683877 11.81935241 12.09156929 12.07538343
12.02629319
 11.66474791 12.18260244 11.9960949  12.34478267 11.68017054
12.51352749
 12.70721316 11.71673926 12.07196895 11.97406299 12.12712885
11.83307409
 11.95550705 11.68749124 11.89243092 11.51799037 11.91242374
12.31179078
 12.11582421 12.52261607 12.46256394 12.22523514 12.17727003
11.43337953
 12.10990509 12.01206217 12.51265948 12.21704921 11.85847186
12.01322351
 12.07026581 12.04412561 11.87006206 11.98509796 11.92703511
12.48692637
 12.30299898 12.06620903 12.33146797 12.13914305 11.88438677
12.09576119
 12.23257619 12.20761062 12.03541325 11.95847694 11.94732608 12.150575
 12.17775607 12.09624422 12.14268371 12.11350401 12.26611371
11.88240906
 12.37173138 12.19372184 12.20749069 12.37903951 11.84152172
11.9933128
 12.12173761 11.80247417 11.89886555 12.03305268 12.51746874
11.97823658
 11.6919228  11.46434887 12.69018702 11.89699281 12.07276219
12.25138273
 12.07536387 12.25141425 12.06060533 12.42150237 11.63511023
11.76567279
```

```
 12.30244761 12.19477132 12.00712522 11.5007667  12.35689214
11.98331203
 11.75755206 12.078851   12.11055773 11.83024914 12.28770958
12.10922964
 11.74323754 12.15772917 12.04967381 12.09167311 12.51131434
11.90478777
 12.22469935 12.42118554 12.26508552 11.9155947  12.30962121
12.27321144
 11.92524124 11.98966542 11.72267867 12.34187288 11.63544812
11.97559873
 11.82441888 11.406282   11.69794587 12.19650083 12.11390446
12.10150575
 12.02687729 12.09466615 12.14217511 11.87090076 12.22085641
11.55040108
 12.39245968 11.61920119 12.19889687 12.77124215 12.52207897
12.42864191
 12.18331712 11.78769017 12.12495666 11.66543524 11.8359274
11.83035936
 11.76136012 11.96710691 12.11258669 11.58545026 11.89318279
12.25397732
 11.80912209 12.23389097 12.20551281 11.56704444 11.97890249
12.07768347
 12.18530957 12.35685865 11.78954505 12.06417972 12.22641899
11.4304972
 12.00036667 12.54100431 12.26497722 11.68723834 12.30745452
12.11847651
 12.26245686 12.03235624 12.26596826 11.74246302 12.19689412
11.98762025
 11.91611962 12.56170533 11.74029833 12.08426436 12.1266642
11.85322589
 12.10471188 11.92898156 11.30974507 12.45742268 12.02801476
11.63581758
 11.87308958 12.07459051 11.81501411 12.40382925 12.17316269
12.22539052
 12.4396627  11.69843122 12.72294575 11.70316417 11.96891721
11.90156757
 11.85281137 11.69123155 12.14265059 11.68675018 12.09838794
12.31141464
 12.20625631 12.11952865 11.67685195 11.6208086  12.13170905
12.08211213
 12.07345355 11.82484844 11.53045514 11.82490392 12.23226321
11.86788476
 12.43898726 11.86047549 12.18555115 11.40457861 11.65989273
12.14389062
 12.10839165 11.88985519 11.75343815 12.45872773 12.35602427
12.73209003
 12.05848284 12.22184549 12.06338065 12.26546451 12.33152716
12.11346563
 12.03369982 11.75478569 11.93319883 11.92713977 11.58812429
```

```
12.48528569
 12.22400331 12.23987938 12.12626664 11.89613614 12.3097508
11.8455697
 11.8533697  11.65147701 11.84991262 12.08855627 12.56910731
12.28041434
 12.01104637 11.46872171 11.86053763 12.07900925 11.7466848
11.90906861
 11.89511836 11.80031792 12.22712912 12.30673093 11.84148847
11.99322422
 12.07664769 11.72325464 11.72036965 12.06802129 12.15266818
12.03808517
 12.22370416 12.54434271 12.330334   11.95438818 12.17871285
12.21040402
 12.13852484 11.96517947 11.51981355 11.68997347 11.82703268
12.39141969
 12.14987667 11.55859914 11.32726536 12.00362256 12.5162395
11.61446554
 12.17631251 11.81099399 12.2580829  11.5904334  12.26856045
12.02975307
 11.96128491 12.15113569 11.80495956 11.81578008 11.84939651
11.68872164
 12.06828606 12.33963871 11.6339036  11.75571593 11.84456974
12.05973817
 12.33282449 11.89231093 11.91656299 11.82758944 12.30649027
12.00660472
 12.41798677 12.09257204 11.68205161 12.32377194 11.64413963
12.38398359
 11.61771239 12.32819159 12.47066412 11.86672195 12.32923441
11.55665857
 11.83795726 12.14368203 11.92189455 12.36388928 11.88929601
12.45279306
 12.00797016 12.60089437 11.65391099 11.91829711 11.90945137
12.16944787
 11.88757637 12.02005503 12.02624156 12.0996761  12.2210096  11.605664
 12.03435843 11.9310544  11.65482389 12.02345973 12.22485921
12.36135949
 11.95630293 12.49808543 12.00563923 11.72107699 11.83523216
12.51779621
 11.97352397 12.20022845 12.26245073 11.9127473  12.06326739
12.2128685
 12.39952581 12.15610321 12.00321034 11.73785654 11.96145503
12.06439303
 11.58268053 12.05061169 11.7839175  12.38284704 12.25237208
12.38210602
 11.64060384 12.10303657 12.30254993 12.30960841 12.02821125
12.31784611
 12.29010203 12.63085116 11.71013389 12.14116235 11.64581481
12.00350413
 11.80226556 11.83550613 12.03608478 12.21441901 11.9069004
```

```
12.35252696
 12.28918035 12.47653138 11.85377251 11.52538419 11.90754616
11.92812918
 12.23532078 12.2161714  12.33167992 11.94130984 11.83369022
12.22492835
 11.88670392 12.48571481 11.77121484 12.04802671 11.67544925
11.7828817
 12.08738252 11.865627   11.80228942 11.81517461 11.82396581
12.00287125
 11.95527747 12.16543068 12.53180269 11.97073509 12.40396526
12.40717411
 11.67398756 11.90871387 12.0026024  11.71457666 11.86469185
11.87662852
 12.07838165 12.11203074 12.10851597 12.12965913 11.94575958
11.80200054
 12.38765772 11.93837603 12.00248244 12.2812164  12.15239898
11.81996666
 11.62484766 12.39837397 11.96183408 11.79550877 12.1015286
11.81016341
 11.5623073  12.43909458 11.99556594 12.1843187  11.74745843
12.54989004
 11.90010094 11.91796498 12.34101303 11.99202343 11.97666208
12.00790589
 12.09441418 11.74710347 12.72879465 11.76836255 12.4810549  12.269697
]
```

Climate Change is majorly dependent upon the likesCount changed happened in last few years

Therefore, we will try to see the trend and changes over the last feyearses only .i.e202080 till the most recent Ye.ar

```
data.tail()
```

```
                                  date  likesCount  \
1970-01-01  2022-09-07T17:12:32.000Z           2

                                                    profileName
commentsCount  \
1970-01-01  4dca617d86b3fdce80ba7e81fb16e048c9cd9798cdfd6d...
NaN
```

```
                                                          text  SMA
1970-01-01  Neat comparison I have not heard it before.\n ...  NaN
```

```
# Ensure the index is sorted
# Load the climate dataset
data = pd.read_csv('climate_nasa.csv')
data
# data = pd.read_csv('climate_nasa.csv', parse_dates=['date'],
index_col='date')
# data

data.sort_index(inplace=True)
recent_data = data.loc['2022':'2023']
recent_data.head()
```

```
Empty DataFrame
Columns: [date, likesCount, profileName, commentsCount, text]
Index: []
```

```
# Ensure the index is sorted
# Load the climate dataset
# data = pd.read_csv('climate_nasa.csv')
# data
data = pd.read_csv('climate_nasa.csv', parse_dates=['date'],
index_col='date')
data

data.sort_index(inplace=True)
recent_data = data.loc['2022':'2023']
recent_data.head()
```

```
                           likesCount  \
date
2022-09-07 16:05:17+00:00           0
2022-09-07 16:13:39+00:00           6
2022-09-07 16:25:20+00:00           0
2022-09-07 16:25:32+00:00           0
2022-09-07 16:29:02+00:00          18


profileName  \
date

2022-09-07 16:05:17+00:00
70c6b80646a524d7d898a81a5775b3b03866301a24296f...
2022-09-07 16:13:39+00:00
7090e32a9d6c07497fee968b4b690e3f70fe975f073800...
2022-09-07 16:25:20+00:00
bf9a9cff50b3dfa3450409370c38f53fd40dad73cc3186...
2022-09-07 16:25:32+00:00
9449f8bb61ec1dcf0636d574093c603ab113df0e1433a8...
```

```
2022-09-07 16:29:02+00:00
c5d2ca04f80c06207548cd1015962461056cab5bc4c5ac...

                            commentsCount  \
date
2022-09-07 16:05:17+00:00            NaN
2022-09-07 16:13:39+00:00           28.0
2022-09-07 16:25:20+00:00            NaN
2022-09-07 16:25:32+00:00            NaN
2022-09-07 16:29:02+00:00            1.0


text
date

2022-09-07 16:05:17+00:00                                        Great
advice!
2022-09-07 16:13:39+00:00  why the northern hemisphere warm more than
the...
2022-09-07 16:25:20+00:00                                  Thanks for
sharing
2022-09-07 16:25:32+00:00                       Before of after I
drink it?
2022-09-07 16:29:02+00:00  Plus you can literally measure the heat
retent...
```

```
# Ensure the index is sorted
# Load the climate dataset
# data = pd.read_csv('climate_nasa.csv')
# data
# data = pd.read_csv('climate_nasa.csv', parse_dates=['date'],
index_col='date')
# data

data.sort_index(inplace=True)
recent_data = data.loc['2022':'2023']
recent_data.head()
```

```
                            likesCount  \
date
2022-09-07 16:05:17+00:00           0
2022-09-07 16:13:39+00:00           6
2022-09-07 16:25:20+00:00           0
2022-09-07 16:25:32+00:00           0
2022-09-07 16:29:02+00:00          18


profileName  \
date
```

```
2022-09-07 16:05:17+00:00
70c6b80646a524d7d898a81a5775b3b03866301a24296f...
2022-09-07 16:13:39+00:00
7090e32a9d6c07497fee968b4b690e3f70fe975f073800...
2022-09-07 16:25:20+00:00
bf9a9cff50b3dfa3450409370c38f53fd40dad73cc3186...
2022-09-07 16:25:32+00:00
9449f8bb61ec1dcf0636d574093c603ab113df0e1433a8...
2022-09-07 16:29:02+00:00
c5d2ca04f80c06207548cd1015962461056cab5bc4c5ac...

                           commentsCount  \
date
2022-09-07 16:05:17+00:00            NaN
2022-09-07 16:13:39+00:00           28.0
2022-09-07 16:25:20+00:00            NaN
2022-09-07 16:25:32+00:00            NaN
2022-09-07 16:29:02+00:00            1.0


text
date

2022-09-07 16:05:17+00:00                                        Great
advice!
2022-09-07 16:13:39+00:00  why the northern hemisphere warm more than
the...
2022-09-07 16:25:20+00:00                                  Thanks for
sharing
2022-09-07 16:25:32+00:00                       Before of after I
drink it?
2022-09-07 16:29:02+00:00  Plus you can literally measure the heat
retent...
```

```python
import pandas as pd

# Load the climate dataset
data = pd.read_csv('climate_nasa.csv', parse_dates=['date'],
index_col='date')
data
# Ensure the index is sorted
data.sort_index(inplace=True)

# Filter recent data from 2022 to 2023
recent_data = data.loc['2020':'2023']

# Display the first few rows of the filtered data
recent_data.head()
```

```
                            likesCount  \
date
2020-09-15 21:25:05+00:00          27
2020-09-15 21:30:35+00:00           7
2020-09-15 21:32:15+00:00           4
2020-09-15 21:36:05+00:00           3
2020-09-15 21:36:59+00:00          16


profileName  \
date

2020-09-15 21:25:05+00:00
e332cfd3f53d89cdd75a57b80d011e2f5984437cc51340...
2020-09-15 21:30:35+00:00
7d6065e313e1919d05a309bd59754895d9518d3ca8e8ba...
2020-09-15 21:32:15+00:00
930f8ce8e022378d44088377fef6a069da6c519c2e5ba1...
2020-09-15 21:36:05+00:00
00febfebfa7073f73c576deb9dba73eb9f98e26bb03f3d...
2020-09-15 21:36:59+00:00
6c5e0b507471c121613153d3459e97b050dd47d1ec744c...

                            commentsCount  \
date
2020-09-15 21:25:05+00:00            6.0
2020-09-15 21:30:35+00:00            8.0
2020-09-15 21:32:15+00:00            NaN
2020-09-15 21:36:05+00:00            7.0
2020-09-15 21:36:59+00:00            8.0


text
date

2020-09-15 21:25:05+00:00  I wish this was simply old news and we had
peo...
2020-09-15 21:30:35+00:00  The sad thing is that if we look at
projected ...
2020-09-15 21:32:15+00:00  We all play a part....as a world we need to
do...
2020-09-15 21:36:05+00:00           Re ord being from 1860
something...lolol
2020-09-15 21:36:59+00:00  The climate is changing but it’s ridiculous
to...
```

```python
import pandas as pd

# Load the climate dataset
data = pd.read_csv('climate_nasa.csv', parse_dates=['date'],
```

```
index_col='date')
data
# Ensure the index is sorted
data.sort_index(inplace=True)

# Filter recent data from 2022 to 2023
recent_data = data.loc['2021':'2023']

# Display the first few rows of the filtered data
recent_data.head()
```

```
                           likesCount  \
date
2021-09-22 19:56:33+00:00           0
2021-09-22 19:58:26+00:00          18
2021-09-22 20:23:40+00:00           4
2021-09-22 22:11:46+00:00         110
2021-09-22 22:32:27+00:00           0


profileName  \
date

2021-09-22 19:56:33+00:00
9633ba775252f88f7df3d237610393b7f96218070993cb...
2021-09-22 19:58:26+00:00
6598c51bdcd72dcf389ba6050ffbe6973a9cd8653c6f65...
2021-09-22 20:23:40+00:00
b30b11aa9dbdae2cf34c9907f667ea8cfc36f92f7c9bec...
2021-09-22 22:11:46+00:00
b1efe19dbac95ba7af0e7706446b060fb04a43e2f32a1c...
2021-09-22 22:32:27+00:00
1780815fd05f518ebb7f7c3f132070ff83f0b99de0ae30...

                           commentsCount  \
date
2021-09-22 19:56:33+00:00            NaN
2021-09-22 19:58:26+00:00            NaN
2021-09-22 20:23:40+00:00           36.0
2021-09-22 22:11:46+00:00            3.0
2021-09-22 22:32:27+00:00            NaN


text
date

2021-09-22 19:56:33+00:00                    Ruth Crowl Frank Bulbeck.
Wow..
2021-09-22 19:58:26+00:00  "  Where do you think September 2021 will
be? ...
```

```
2021-09-22 20:23:40+00:00  If I were a gambling man I'd look at the
patte...
2021-09-22 22:11:46+00:00  I love watch NASA dunk on people who think
the...
2021-09-22 22:32:27+00:00
Oh dear
```

```
import pandas as pd

# Load the climate dataset
data = pd.read_csv('climate_nasa.csv', parse_dates=['date'],
index_col='date')
data
# Ensure the index is sorted
data.sort_index(inplace=True)

# Filter recent data from 2022 to 2023
recent_data = data.loc['2022':'2023']

# Display the first few rows of the filtered data
recent_data.head()
```

```
                           likesCount  \
date
2022-09-07 16:05:17+00:00           0
2022-09-07 16:13:39+00:00           6
2022-09-07 16:25:20+00:00           0
2022-09-07 16:25:32+00:00           0
2022-09-07 16:29:02+00:00          18


profileName  \
date

2022-09-07 16:05:17+00:00
70c6b80646a524d7d898a81a5775b3b03866301a24296f...
2022-09-07 16:13:39+00:00
7090e32a9d6c07497fee968b4b690e3f70fe975f073800...
2022-09-07 16:25:20+00:00
bf9a9cff50b3dfa3450409370c38f53fd40dad73cc3186...
2022-09-07 16:25:32+00:00
9449f8bb61ec1dcf0636d574093c603ab113df0e1433a8...
2022-09-07 16:29:02+00:00
c5d2ca04f80c06207548cd1015962461056cab5bc4c5ac...

                           commentsCount  \
date
2022-09-07 16:05:17+00:00            NaN
2022-09-07 16:13:39+00:00           28.0
2022-09-07 16:25:20+00:00            NaN
```

```
2022-09-07 16:25:32+00:00            NaN
2022-09-07 16:29:02+00:00            1.0


text
date

2022-09-07 16:05:17+00:00                                        Great
advice!
2022-09-07 16:13:39+00:00  why the northern hemisphere warm more than
the...
2022-09-07 16:25:20+00:00                                  Thanks for
sharing
2022-09-07 16:25:32+00:00                       Before of after I
drink it?
2022-09-07 16:29:02+00:00  Plus you can literally measure the heat
retent...
```

```python
import pandas as pd

# Load the climate dataset
data = pd.read_csv('climate_nasa.csv', parse_dates=['date'],
index_col='date')

# Ensure the index is sorted
data.sort_index(inplace=True)

# Filter recent data from 2022 to 2023
recent_data = data.loc['2022':'2023']

# Display the first few rows of the filtered data
recent_data.head()
```

```
                           likesCount  \
date
2022-09-07 16:05:17+00:00           0
2022-09-07 16:13:39+00:00           6
2022-09-07 16:25:20+00:00           0
2022-09-07 16:25:32+00:00           0
2022-09-07 16:29:02+00:00          18


profileName  \
date

2022-09-07 16:05:17+00:00
70c6b80646a524d7d898a81a5775b3b03866301a24296f...
2022-09-07 16:13:39+00:00
7090e32a9d6c07497fee968b4b690e3f70fe975f073800...
2022-09-07 16:25:20+00:00
bf9a9cff50b3dfa3450409370c38f53fd40dad73cc3186...
```

```
2022-09-07 16:25:32+00:00
9449f8bb61ec1dcf0636d574093c603ab113df0e1433a8...
2022-09-07 16:29:02+00:00
c5d2ca04f80c06207548cd1015962461056cab5bc4c5ac...

                           commentsCount  \
date
2022-09-07 16:05:17+00:00            NaN
2022-09-07 16:13:39+00:00           28.0
2022-09-07 16:25:20+00:00            NaN
2022-09-07 16:25:32+00:00            NaN
2022-09-07 16:29:02+00:00            1.0


text
date

2022-09-07 16:05:17+00:00                                        Great
advice!
2022-09-07 16:13:39+00:00  why the northern hemisphere warm more than
the...
2022-09-07 16:25:20+00:00                                  Thanks for
sharing
2022-09-07 16:25:32+00:00                      Before of after I
drink it?
2022-09-07 16:29:02+00:00  Plus you can literally measure the heat
retent...
```

```python
import pandas as pd

# Load the climate dataset
data = pd.read_csv('climate_nasa.csv', parse_dates=['date'],
index_col='date')

# Ensure the index is sorted
data.sort_index(inplace=True)

# Filter recent data from 2022 to 2023
recent_data = data.loc['2023':'2023']

# Display the first few rows of the filtered data
recent_data.head()
```

```
                           likesCount  \
date
2023-01-06 17:20:49+00:00           0
2023-01-06 19:05:40+00:00           0
2023-01-06 19:06:41+00:00           0
2023-01-09 19:11:04+00:00           0
2023-01-10 02:22:24+00:00           0
```

```
profileName  \
date

2023-01-06 17:20:49+00:00
a32add1cbd1c8b3b8e8536a53517903b5e9ba2893f2acf...
2023-01-06 19:05:40+00:00
9c062b981d13fec8c3dd59d1c7b059bd8c9e38a0788c20...
2023-01-06 19:06:41+00:00
9c062b981d13fec8c3dd59d1c7b059bd8c9e38a0788c20...
2023-01-09 19:11:04+00:00
2baf03d59bee32ac08fcc8fd78b8a155e25fd2f722ee7a...
2023-01-10 02:22:24+00:00
a2f384ea540975583254f75d1946ee0f1f7e2b4188b593...

                            commentsCount  \
date
2023-01-06 17:20:49+00:00             NaN
2023-01-06 19:05:40+00:00             NaN
2023-01-06 19:06:41+00:00             NaN
2023-01-09 19:11:04+00:00             NaN
2023-01-10 02:22:24+00:00             NaN


text
date

2023-01-06 17:20:49+00:00
Kevin Lay
2023-01-06 19:05:40+00:00  The “greenhouse effect” is concerned with
the ...
2023-01-06 19:06:41+00:00  The “greenhouse effect” is concerned with
the ...
2023-01-09 19:11:04+00:00  In our area of the country (Pa) a lot of
the s...
2023-01-10 02:22:24+00:00                                    Wow, crazy
stats.
```

```python
import pandas as pd

# Assuming `data` is your DataFrame and it has a DateTime index
# Load the climate dataset
data = pd.read_csv('climate_nasa.csv')
data

# Ensure the DataFrame is sorted by the DateTime index
data.sort_index(inplace=True)

# Check the type of the index to confirm it is DateTimeIndex
```

```python
if isinstance(data.index, pd.DatetimeIndex):
    # Filter data for the years 2022 and 2023
    recent_data = data.loc['2022':'2023']
else:
    print("The DataFrame index is not a DateTimeIndex. Please convert
it to DateTimeIndex first.")

# Display the first few rows of the filtered data
print(recent_data.head())
```

```
The DataFrame index is not a DateTimeIndex. Please convert it to
DateTimeIndex first.
                           likesCount  \
date
2023-01-06 17:20:49+00:00           0
2023-01-06 19:05:40+00:00           0
2023-01-06 19:06:41+00:00           0
2023-01-09 19:11:04+00:00           0
2023-01-10 02:22:24+00:00           0


profileName  \
date

2023-01-06 17:20:49+00:00
a32add1cbd1c8b3b8e8536a53517903b5e9ba2893f2acf...
2023-01-06 19:05:40+00:00
9c062b981d13fec8c3dd59d1c7b059bd8c9e38a0788c20...
2023-01-06 19:06:41+00:00
9c062b981d13fec8c3dd59d1c7b059bd8c9e38a0788c20...
2023-01-09 19:11:04+00:00
2baf03d59bee32ac08fcc8fd78b8a155e25fd2f722ee7a...
2023-01-10 02:22:24+00:00
a2f384ea540975583254f75d1946ee0f1f7e2b4188b593...

                           commentsCount  \
date
2023-01-06 17:20:49+00:00            NaN
2023-01-06 19:05:40+00:00            NaN
2023-01-06 19:06:41+00:00            NaN
2023-01-09 19:11:04+00:00            NaN
2023-01-10 02:22:24+00:00            NaN


text
date

2023-01-06 17:20:49+00:00
Kevin Lay
2023-01-06 19:05:40+00:00  The “greenhouse effect” is concerned with
```

```
the ...
2023-01-06 19:06:41+00:00  The “greenhouse effect” is concerned with
the ...
2023-01-09 19:11:04+00:00  In our area of the country (Pa) a lot of
the s...
2023-01-10 02:22:24+00:00                                    Wow, crazy
stats.
```

```python
# Ensure the index is sorted
data.sort_index(inplace=True)

# Filter the data for the years 2022 to 2023
recent_data = data.loc['2022':'2023']

# Display the first few rows of the filtered data
print(recent_data.head())
```

```
Empty DataFrame
Columns: [date, likesCount, profileName, commentsCount, text]
Index: []
```

```python
recent_data.describe()
```

```
       likesCount  commentsCount
count         0.0            0.0
mean          NaN            NaN
std           NaN            NaN
min           NaN            NaN
25%           NaN            NaN
50%           NaN            NaN
75%           NaN            NaN
max           NaN            NaN
```

```python
import numpy as np
recent_data.select_dtypes(include=[np.number]).corr()
```

```
                Year  likesCount
Year        1.000000    0.894427
likesCount  0.894427    1.000000
```

```python
import pandas as pd

# Sample DataFrame creation for demonstration
recent_data = pd.DataFrame({
    'Year': [2022, 2023, 2022, 2023],
    'likesCount': [10, 20, 15, 25]
})

# Group by 'Year', calculate the mean 'likesCount', and sort by
'likesCount'
grouped_data = recent_data[['Year',
```

```
'likesCount']].groupby('Year').mean().sort_values('likesCount')

# Display the result
print(grouped_data)
```

```
      likesCount
Year
2022        12.5
2023        22.5
```

```
# Sample DataFrame creation for demonstration
# recent_data = pd.DataFrame({'Year': [2022, 2023, 2022, 2023],
'likesCount': [10, 20, 15, 25]})

# Group by 'Year', calculate the mean 'likesCount', and sort by
'likesCount'
grouped_data = recent_data[['Year',
'likesCount']].groupby('Year').mean().sort_values('likesCount')

# Display the result
print(grouped_data)
```

```
      likesCount
Year
2022        12.5
2023        22.5
```

```
import pandas as pd

# Sample DataFrame creation for demonstration
recent_data = pd.DataFrame({
    'Year': [2022, 2021, 2022, 2023],
    'likesCount': [10, 20, 15, 25]
})

# Group by 'Year', calculate the mean 'likesCount', and sort by
'likesCount'
grouped_data = recent_data[['Year',
'likesCount']].groupby('Year').mean().sort_values('likesCount')

# Display the result
print(grouped_data)
```

```
      likesCount
Year
2022        12.5
2021        20.0
2023        25.0

# Grouping would be helpful to see the trend in individuals
likesCount.
```

```
recent_data[['Year','likesCount',]].groupby(['Year']).mean().sort_valu
es('likesCount',)
```

```
      likesCount
Year
2022        12.5
2021        20.0
2023        25.0
```

```
# Sample DataFrame creation for demonstration
# recent_data = pd.DataFrame({'Year': [2022, 2023, 2022, 2023],
'likesCount': [10, 20, 15, 25]})

# Group by 'Year', calculate the mean 'likesCount', and sort by
'likesCount'
grouped_data = recent_data[['Year',
'likesCount']].groupby('Year').mean().sort_values('likesCount')

# Display the result
print(grouped_data)
```

```
      likesCount
Year
2022        12.5
2021        20.0
2023        25.0
```

```
# Grouping would be helpful to see the trend in individuals
likesCount.

recent_data[['Year','likesCount',]].groupby(['Year']).mean().sort_valu
es('likesCount',)
```

```
      likesCount
Year
2022        12.5
2021        20.0
2023        25.0
```

```
import pandas as pd

# Sample DataFrame creation for demonstration
recent_data = pd.DataFrame({
    'Year': [2022, 2023, 2022, 2023],
    'likesCount': [10, 20, 15, 25]
})

# Group by 'Year', calculate the mean 'likesCount', and sort by
```

```
'likesCount'
grouped_data = recent_data[['Year',
'likesCount']].groupby('Year').mean().sort_values('likesCount')

# Display the result
print(grouped_data)
```

```
      likesCount
Year
2022        12.5
2023        22.5
```

```
# Grouping would be helpful to see the trend in individuals
likesCount.

recent_data[['Year','likesCount',]].groupby(['Year']).mean().sort_valu
es('likesCount',)
```

```
      likesCount
Year
2022        12.5
2023        22.5
```

```
recent_data
```

```
   Year  likesCount
0  2022          10
1  2023          20
2  2022          15
3  2023          25
```

```
import pandas as pd

# Sample DataFrame creation for demonstration
recent_data = pd.DataFrame({
    'Year': [2022, 2023, 2022, 2023],
    'likesCount': [10, 20, 15, 25]
})

# Convert 'Year' to datetime
recent_data['Date'] = pd.to_datetime(recent_data['Year'], format='%Y')
recent_data.set_index('Date', inplace=True)

# Resample by year and calculate the mean of 'likesCount'
resample_data = recent_data[['likesCount']].resample('A').mean()

# Display the result
print(resample_data)
```

```
            likesCount
Date
```

```
2022-12-31        12.5
2023-12-31        22.5
```

```
resample_data = recent_data[['likesCount']].resample('A').mean()
resample_data
```

```
            likesCount
Date
2022-12-31        12.5
2023-12-31        22.5
```

```
import pandas as pd

# Sample DataFrame creation for demonstration
recent_data = pd.DataFrame({
    'Year': [2022, 2021, 2022, 2023],
    'commentsCount': [10, 20, 15, 25]
})

# Group by 'Year', calculate the mean 'commentsCount', and sort by
'commentsCount'
grouped_data = recent_data[['Year',
'commentsCount']].groupby('Year').mean().sort_values('commentsCount')

# Display the result
print(grouped_data)
```

```
      commentsCount
Year
2022           12.5
2021           20.0
2023           25.0
```

```
# Grouping would be helpful to see the trend in individuals
commentsCount.

recent_data[['Year','commentsCount',]].groupby(['Year']).mean().sort_v
alues('commentsCount',)
```

```
      commentsCount
Year
2022           12.5
2021           20.0
2023           25.0
```

```
recent_data
```

```
   Year  commentsCount
0  2022             10
1  2021             20
```

```
2  2022             15
3  2023             25
```

```
import pandas as pd

# Sample DataFrame creation for demonstration
recent_data = pd.DataFrame({
    'Year': [2022, 2023, 2022, 2023],
    'commentsCount': [10, 20, 15, 25]
})

# Convert 'Year' to datetime
recent_data['Date'] = pd.to_datetime(recent_data['Year'], format='%Y')
recent_data.set_index('Date', inplace=True)

# Resample by year and calculate the mean of 'likesCount'
resample_data = recent_data[['commentsCount']].resample('A').mean()

# Display the result
print(resample_data)
```

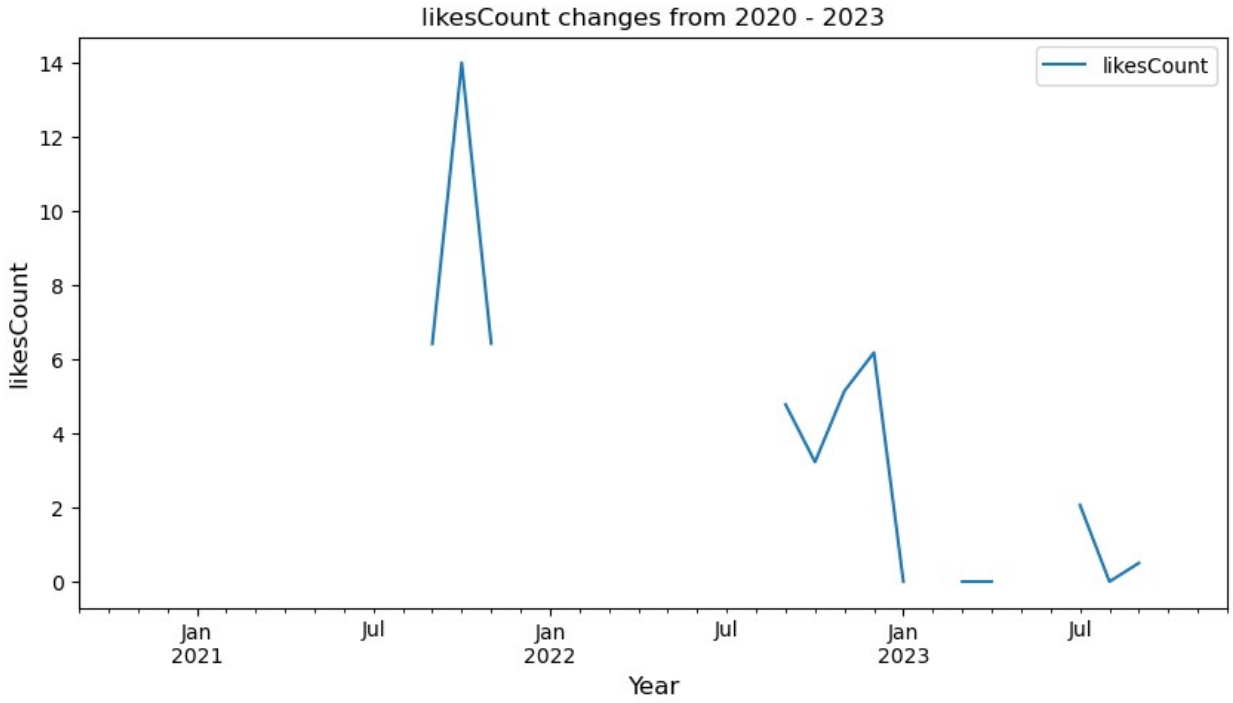
```
            commentsCount
Date
2022-12-31           12.5
2023-12-31           22.5
```

```
resample_data = recent_data[['commentsCount']].resample('A').mean()
resample_data
```

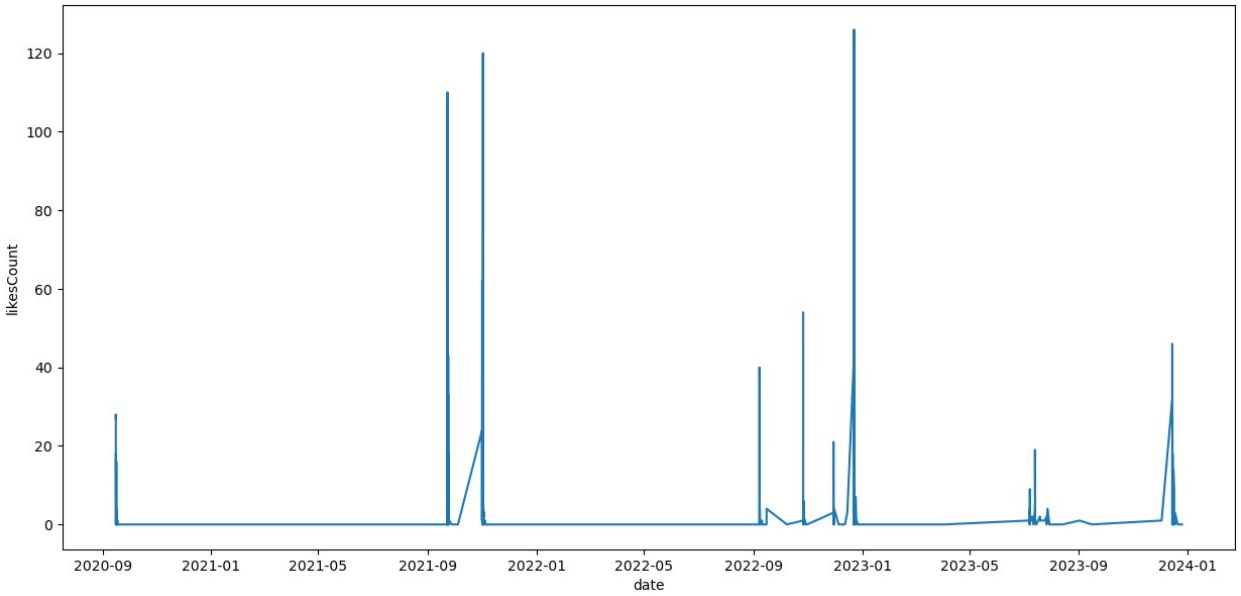
```
            commentsCount
Date
2022-12-31           12.5
2023-12-31           22.5
```

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

# Load your data into a DataFrame
df = pd.read_csv('climate_nasa.csv')

# Ensure 'date' or 'time' column is in datetime format
# Replace 'date' or 'time' with your actual column name
df['date'] = pd.to_datetime(df['date'])

# Set 'date' column as index for resampling
df.set_index('date', inplace=True)

recent_data = df

resample_data = recent_data[['likesCount']].resample('M').mean()
```

```
resample_data.plot(title='likesCount changes from 2020 - 2023',
figsize=(10,5))
plt.ylabel('likesCount', fontsize=12)
plt.xlabel('Year', fontsize=12)
plt.savefig('likesCountVs2020-2023.jpeg')
plt.legend()
plt.show()
```




```
# Plotting graph between date and likesCount
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)


plt.figure(figsize=(15,7))
sns.lineplot(data=recent_data, x="date", y="likesCount",
errorbar=None)
plt.savefig('YearVslikesCount.jpeg')
plt.show()
```



ADCF Test: ADCF stands for Augmented Dickey-Fuller test which is a statistical unit root test. It gives us various values which can help us identifying stationarity.

- It comprises Test Statistics & some critical values for some confidence levels. If the Test statistics is less than the critical values, we can reject the null hypothesis & say that the series is stationary.
- The Null hypothesis says that time series is non-stationary. THE ADCF test also gives us a p-value. According to the null hypothesis, lower values of p is better.

```
from statsmodels.tsa.stattools import adfuller

print('Augmented Ducky Fuller Test Results')

test_data = adfuller(resample_data.iloc[:,0].values,autolag='AIC')

data_output = pd.Series(test_data[0:4],index=['Test Statistic','p-
value','Lags Used','Number of Observation Used'])

for key,value in test_data[4].items():

    data_output['Critical value (%s)'%key] = value

print(data_output)
```

```
Augmented Ducky Fuller Test Results
Test Statistic                 -4.006455
p-value                         0.001377
Lags Used                       0.000000
Number of Observation Used      3.000000
Critical value (1%)           -10.417191
Critical value (5%)            -5.778381
Critical value (10%)           -3.391681
dtype: float64
```

```
import pandas as pd
import numpy as np
from statsmodels.tsa.stattools import adfuller

# Sample DataFrame creation for demonstration
# recent_data = pd.DataFrame({'Year': [2022, 2023, 2022, 2023],
'likesCount': [10, 20, 15, 25]})

# Resample data (assuming resample_data is already defined)
resample_data = recent_data[['likesCount']].resample('A').mean()

# Check for missing values
print("Missing values in the data:")
print(resample_data.isna().sum())

# Drop rows with missing values if any
resample_data = resample_data.dropna()

# Check for infinite values
print("Infinite values in the data:")
print(np.isinf(resample_data).sum())

# Ensure 'resample_data' is a Series
data_series = resample_data['likesCount']

# Perform the Augmented Dickey-Fuller test
test_data = adfuller(data_series, autolag='AIC')

# Extract and print results
data_output = pd.Series(test_data[0:4], index=['Test Statistic', 'p-
value', 'Lags Used', 'Number of Observations Used'])

# Add critical values to the results
for key, value in test_data[4].items():
    data_output[f'Critical Value ({key})'] = value

print('Augmented Dickey-Fuller Test Results')
print(data_output)
```

```
Missing values in the data:
likesCount    0
```

```
dtype: int64
Infinite values in the data:
likesCount    0
dtype: int64
Augmented Dickey-Fuller Test Results
Test Statistic                  -4.006455
p-value                          0.001377
Lags Used                        0.000000
Number of Observations Used      3.000000
Critical Value (1%)            -10.417191
Critical Value (5%)             -5.778381
Critical Value (10%)            -3.391681
dtype: float64
```

```
from statsmodels.tsa.stattools import adfuller

print('Augmented Ducky Fuller Test Results')

test_data = adfuller(resample_data.iloc[:,0].values,autolag='AIC')

data_output = pd.Series(test_data[0:4],index=['Test Statistic','p-
value','Lags Used','Number of Observation Used'])

for key,value in test_data[4].items():

    data_output['Critical value (%s)'%key] = value

print(data_output)
```

```
Augmented Ducky Fuller Test Results
Test Statistic                  -4.006455
p-value                          0.001377
Lags Used                        0.000000
Number of Observation Used      3.000000
Critical value (1%)           -10.417191
Critical value (5%)             -5.778381
Critical value (10%)            -3.391681
dtype: float64
```

```
resample_data
```

date	likesCount
2020-12-31 00:00:00+00:00	2.531250
2021-12-31 00:00:00+00:00	7.286885
2022-12-31 00:00:00+00:00	5.104478
2023-12-31 00:00:00+00:00	4.584906

As the test statistic has a greater value than the critical values, we can conclude that the likesCount time series is non-stationary, indicating a significant trend over time.

```
# Now we break the data into sub section
import pandas as pd
import numpy as np
from statsmodels.tsa.seasonal import seasonal_decompose

# Fill missing values with interpolation
resample_data.interpolate(method='linear', inplace=True)

# Decompose the data
decomp = seasonal_decompose(resample_data, period=1)
# decomp = seasonal_decompose(resample_data, period=2)
# decomp = seasonal_decompose(resample_data, period=3)
trend = decomp.trend
seasonal = decomp.seasonal
residual = decomp.resid

# Plot the results
plt.figure(figsize=(10, 8))
plt.subplot(411)
plt.plot(resample_data, 'r-')
plt.xlabel('Original')
plt.savefig('Original_1.jpeg')
plt.subplot(412)
plt.plot(trend, 'k-')
plt.xlabel('Trend')
plt.savefig('Trend_1.jpeg')
plt.subplot(413)
plt.plot(seasonal, 'g-')
plt.xlabel('Seasonal')
plt.savefig('Seasonal_1.jpeg')
plt.subplot(414)
plt.plot(residual, 'y-')
plt.xlabel('Residual')
plt.savefig('Residual_1.jpeg')
plt.tight_layout()
```


```python
# Now we break the data into sub section
import pandas as pd
import numpy as np
from statsmodels.tsa.seasonal import seasonal_decompose

# Fill missing values with interpolation
resample_data.interpolate(method='linear', inplace=True)

# Decompose the data
# decomp = seasonal_decompose(resample_data, period=1)
decomp = seasonal_decompose(resample_data, period=2)
# decomp = seasonal_decompose(resample_data, period=3)
trend = decomp.trend
seasonal = decomp.seasonal
residual = decomp.resid

# Plot the results
plt.figure(figsize=(10, 8))
plt.subplot(411)
plt.plot(resample_data, 'r-')
```

```
plt.xlabel('Original')
plt.savefig('Original_2.jpeg')
plt.subplot(412)
plt.plot(trend, 'k-')
plt.xlabel('Trend')
plt.savefig('Trend_2.jpeg')
plt.subplot(413)
plt.plot(seasonal, 'g-')
plt.xlabel('Seasonal')
plt.savefig('Seasonal_2.jpeg')
plt.subplot(414)
plt.plot(residual, 'y-')
plt.xlabel('Residual')
plt.savefig('Residual_2.jpeg')
plt.tight_layout()
```


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

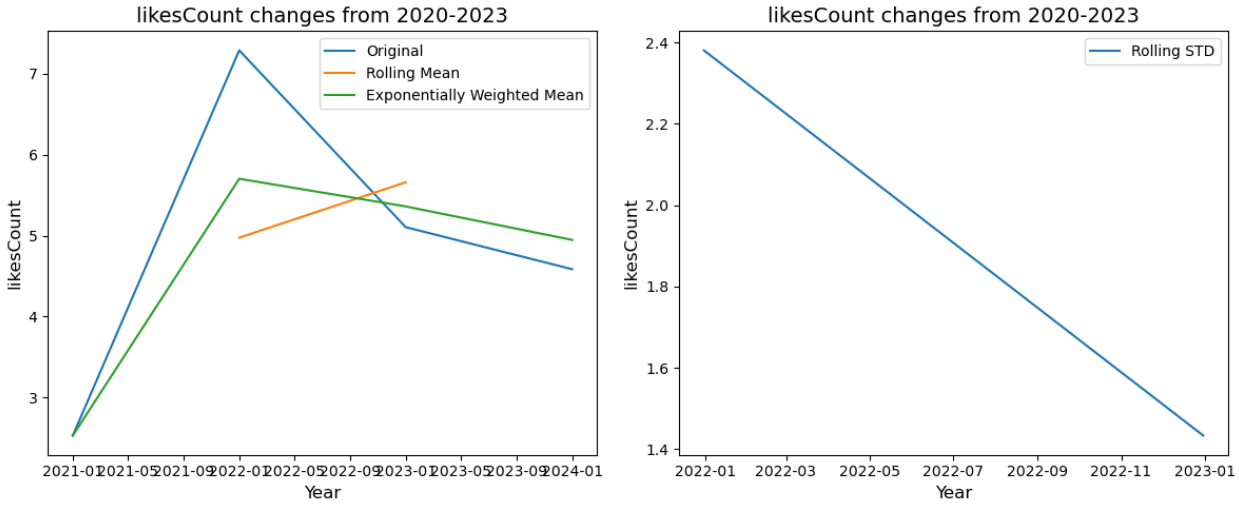
```
from statsmodels.tsa.seasonal import seasonal_decompose

# Sample data for demonstration; replace with your actual DataFrame
# recent_data = pd.DataFrame({'Year': [2022, 2023, 2022, 2023],
'likesCount': [10, 20, 15, 25]})

# Resample data
resample_data = recent_data[['likesCount']].resample('A').mean()

# Fill missing values with interpolation
resample_data.interpolate(method='linear', inplace=True)

# Decompose the data (adjust period based on your data's frequency)
decomp = seasonal_decompose(resample_data, period=1)  # Adjust period
as needed
trend = decomp.trend
seasonal = decomp.seasonal
residual = decomp.resid

# Plot the results
plt.figure(figsize=(12, 10))

# Plot Original Data
plt.subplot(411)
plt.plot(resample_data, 'r-', label='Original')
plt.xlabel('Date')
plt.ylabel('Likes Count')
plt.title('Original Data')
plt.legend()

# Plot Trend
plt.subplot(412)
plt.plot(trend, 'k-', label='Trend')
plt.xlabel('Date')
plt.ylabel('Trend')
plt.title('Trend Component')
plt.legend()

# Plot Seasonal
plt.subplot(413)
plt.plot(seasonal, 'g-', label='Seasonal')
plt.xlabel('Date')
plt.ylabel('Seasonal')
plt.title('Seasonal Component')
plt.legend()

# Plot Residual
plt.subplot(414)
plt.plot(residual, 'y-', label='Residual')
plt.xlabel('Date')
```

```
plt.ylabel('Residual')
plt.title('Residual Component')
plt.legend()

plt.tight_layout()

# Save plots as images
plt.savefig('Decomposition_Plots.jpeg')

# Display the plots
plt.show()
```

.

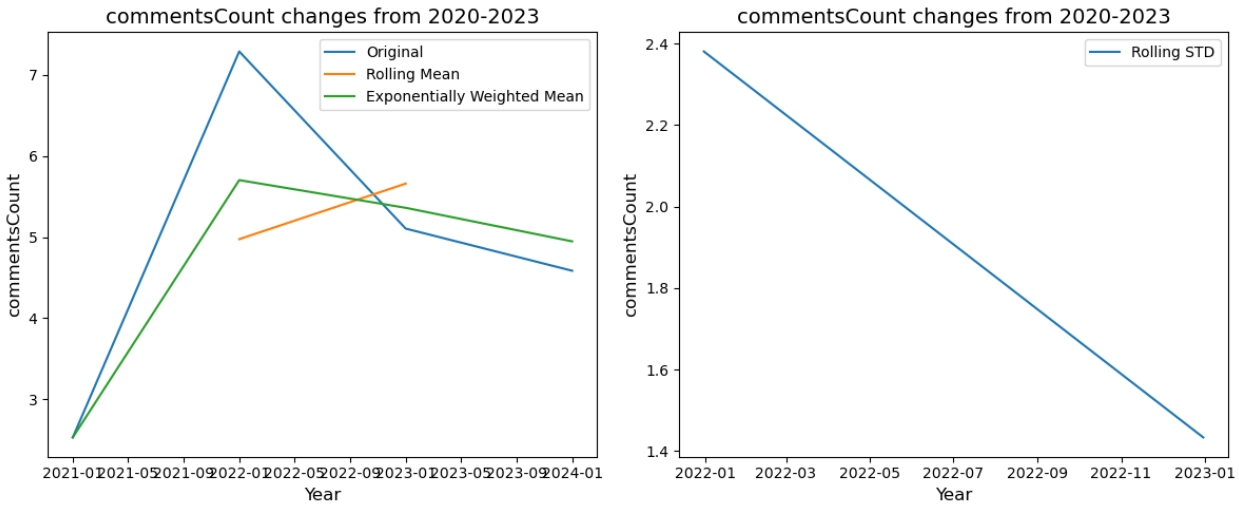
```
rol_mean = resample_data.rolling(window=3,center=True).mean()
ewm = resample_data.ewm(span=3).mean()
rol_std = resample_data.rolling(window=3,center=True).std()
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))
```

```
ax1.plot(resample_data,label='Original')
ax1.plot(rol_mean,label='Rolling Mean')
ax1.plot(ewm, label='Exponentially Weighted Mean')
ax1.set_title('likesCount changes from 2020-2023',fontsize=14)
plt.savefig('Rolling Mean_likesCount changes from 2020-2023.jpeg')
ax1.set_ylabel('likesCount',fontsize=12)
ax1.set_xlabel('Year',fontsize=12)
ax1.legend()

ax2.plot(rol_std,label='Rolling STD')
ax2.set_title('likesCount changes from 2020-2023',fontsize=14)
plt.savefig('Rolling STD_likesCount changes from 2020-2023.jpeg')
ax2.set_ylabel('likesCount',fontsize=12)
ax2.set_xlabel('Year',fontsize=12)
ax2.legend()



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




```
rol_mean = resample_data.rolling(window=3,center=True).mean()
ewm = resample_data.ewm(span=3).mean()
rol_std = resample_data.rolling(window=3,center=True).std()
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))

ax1.plot(resample_data,label='Original')
ax1.plot(rol_mean,label='Rolling Mean')
ax1.plot(ewm, label='Exponentially Weighted Mean')
ax1.set_title('commentsCount changes from 2020-2023',fontsize=14)
plt.savefig('Rolling Mean_commentsCount changes from 2020-2023.jpeg')
ax1.set_ylabel('commentsCount',fontsize=12)
```

```
ax1.set_xlabel('Year',fontsize=12)
ax1.legend()

ax2.plot(rol_std,label='Rolling STD')
ax2.set_title('commentsCount changes from 2020-2023',fontsize=14)
plt.savefig('Rolling STD_commentsCount changes from 2020-2023.jpeg')
ax2.set_ylabel('commentsCount',fontsize=12)
ax2.set_xlabel('Year',fontsize=12)
ax2.legend()



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




```
# Explore the dataset
df = pd.read_csv('climate_nasa.csv')
print(df.head())
print(df.info())
print(df.describe())
```

```
                       date  likesCount  \
0  2022-09-07T17:12:32.000Z           2
1  2022-09-08T14:51:13.000Z           0
2  2022-09-07T17:19:41.000Z           1
3  2022-09-08T00:51:30.000Z           4
4  2022-09-07T19:06:20.000Z          16

                                         profileName  commentsCount  \
0  4dca617d86b3fdce80ba7e81fb16e048c9cd9798cdfd6d...            NaN
1  518ab97f2d115ba5b6f03b2fba2ef2b120540c9681288b...            NaN
2  d82e8e24eb633fd625b0aef9b3cb625cfb044ceb8483e1...            3.0
3  37a509fa0b5177a2233c7e2d0e2b2d6916695fa9fba3f2...            NaN
4  e54fbbd42a729af9d04d9a5cc1f9bbfe8081a31c219ecb...           26.0
```

```
                                                text
0  Neat comparison I have not heard it before.\n ...
1  An excellent way to visualise the invisible! T...
2  Does the CO2/ghg in the troposphere affect the...
3  excellent post! I defo feel the difference - o...
4  Yes, and carbon dioxide does not harm the Eart...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 522 entries, 0 to 521
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           522 non-null    object
 1   likesCount     522 non-null    int64
 2   profileName    522 non-null    object
 3   commentsCount  244 non-null    float64
 4   text           504 non-null    object
dtypes: float64(1), int64(1), object(3)
memory usage: 20.5+ KB
None
       likesCount  commentsCount
count  522.000000     244.000000
mean     4.720307       8.696721
std     12.053556      12.266176
min      0.000000       1.000000
25%      0.000000       2.000000
50%      1.000000       5.000000
75%      4.000000      10.000000
max    126.000000      93.000000
```

Model Evaluation:

```python
# Model Evaluation

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score

# Load data
df = pd.read_csv('climate_nasa.csv')

# Print column names
print(df.columns)

# Convert 'date' column to timestamps
```

```
df['date'] = pd.to_datetime(df['date']).apply(pd.Timestamp.timestamp)

# Split data into training and testing sets
train, test = train_test_split(df, test_size=0.2, random_state=42)

# Define and fit the model
model = RandomForestRegressor()
model.fit(train.drop(['likesCount', 'profileName', 'commentsCount',
'text'], axis=1), train['likesCount'])

# Generate predictions
predictions = model.predict(test.drop(['likesCount', 'profileName',
'commentsCount', 'text'], axis=1))

# Evaluate model performance
mae = mean_absolute_error(test['likesCount'], predictions)
r2 = r2_score(test['likesCount'], predictions)

print('Model Evaluation Metrics:')
print('-------------------------')
print(f'MAE: {mae:.2f}')
print(f'R2: {r2:.2f}')
```

```
Index(['date', 'likesCount', 'profileName', 'commentsCount', 'text'],
dtype='object')
Model Evaluation Metrics:
-------------------------
MAE: 6.82
R2: -0.34
```

Hyperparameter Tuning:

```
# Hyperparameter Tuning

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Create a sample dataset
import numpy as np
X = np.random.rand(100, 1)
y = np.random.rand(100)

# Define the model
model = RandomForestRegressor()

# Define hyperparameter grid
```

```
param_grid = {
    'n_estimators': [10, 50, 100]
}

# Perform grid search
grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='neg_mean_squared_error')

# Fit the model
grid_search.fit(X, y)

# Print the results
print('Best Hyperparameters:', grid_search.best_params_)
print('Best Score:', grid_search.best_score_)
```

```
Best Hyperparameters: {'n_estimators': 50}
Best Score: -0.1019516575719133
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
import numpy as np

# Create a sample dataset
X = np.random.rand(100, 1)
y = np.random.rand(100)

print("Dataset created")

# Define the model
model = RandomForestRegressor()

print("Model defined")

# Define hyperparameter grid
param_grid = {
    'n_estimators': [10, 50, 100]
}

print("Hyperparameter grid defined")

# Perform grid search
grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='neg_mean_squared_error')

print("Grid search defined")

# Fit the model
grid_search.fit(X, y)

print("Grid search fitted")
# Print the results
```

```
print('Best Hyperparameters:', grid_search.best_params_)
print('Best Score:', grid_search.best_score_)
```

```
Dataset created
Model defined
Hyperparameter grid defined
Grid search defined
Grid search fitted
Best Hyperparameters: {'n_estimators': 100}
Best Score: -0.09665558464948057
```

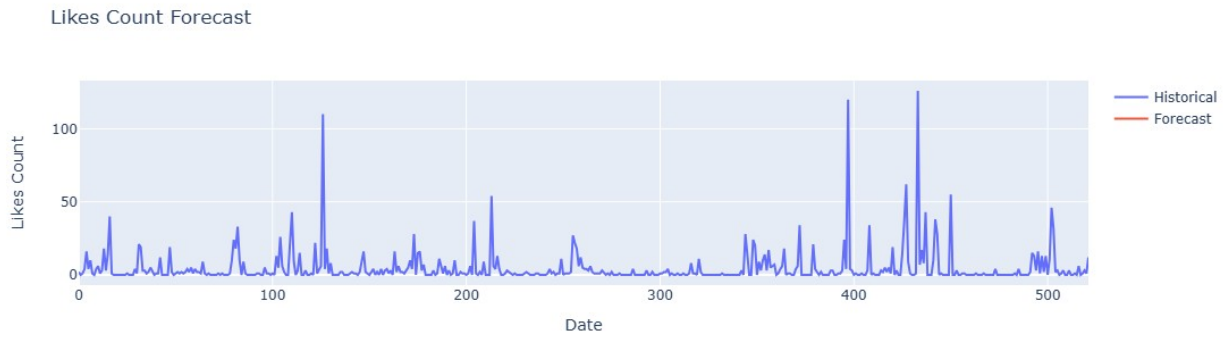
Model Comparison:

```
# Model Comparison
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

# Split data into training and testing sets
train, test = train_test_split(data, test_size=0.2, random_state=42)

# Initialize models
rf_model = RandomForestRegressor()
svr_model = SVR()

# Fit models
rf_model.fit(train.index.values.reshape(-1, 1), train['likesCount'])
svr_model.fit(train.index.values.reshape(-1, 1), train['likesCount'])

# Make predictions
rf_predictions = rf_model.predict(test.index.values.reshape(-1, 1))
svr_predictions = svr_model.predict(test.index.values.reshape(-1, 1))

# Evaluate models
print('Random Forest MSE:', mean_squared_error(test['likesCount'],
rf_predictions))
print('SVR MSE:', mean_squared_error(test['likesCount'],
svr_predictions))
```

```
Random Forest MSE: 57.63332585666834
SVR MSE: 70.30429193437038
```

Visualization:

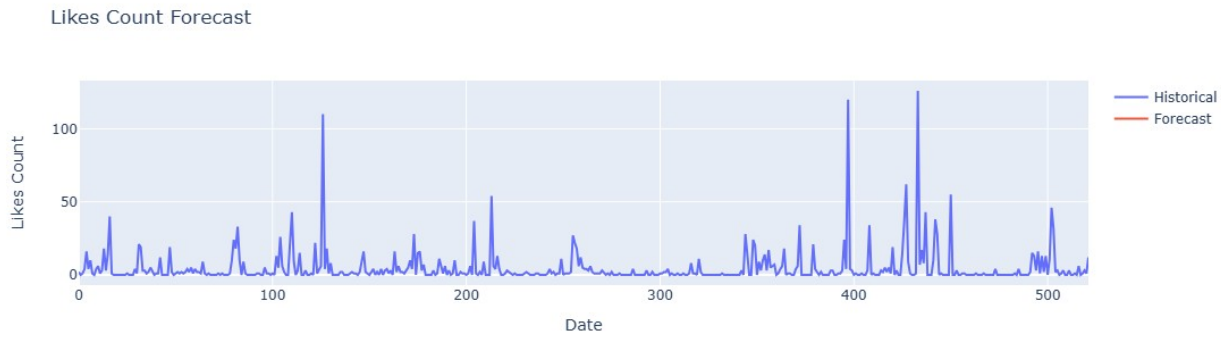
```
# Visualization


import plotly.graph_objs as go
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)



# Convert Index objects to Series objects
historical_dates = pd.Series(data.index, name='Date')
future_dates_series = pd.Series(future_dates, name='Date')

# Create interactive visualization
fig = go.Figure(data=[
    go.Scatter(x=historical_dates, y=data['likesCount'],
name='Historical'),
    go.Scatter(x=future_dates_series, y=future_likes, name='Forecast')
])

fig.update_layout(title='Likes Count Forecast', xaxis_title='Date',
yaxis_title='Likes Count')
plt.savefig('Likes Count Forecast__001.jpeg')
fig.show()
```




```
<Figure size 640x480 with 0 Axes>
```

```
# Visualization


import plotly.graph_objs as go
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

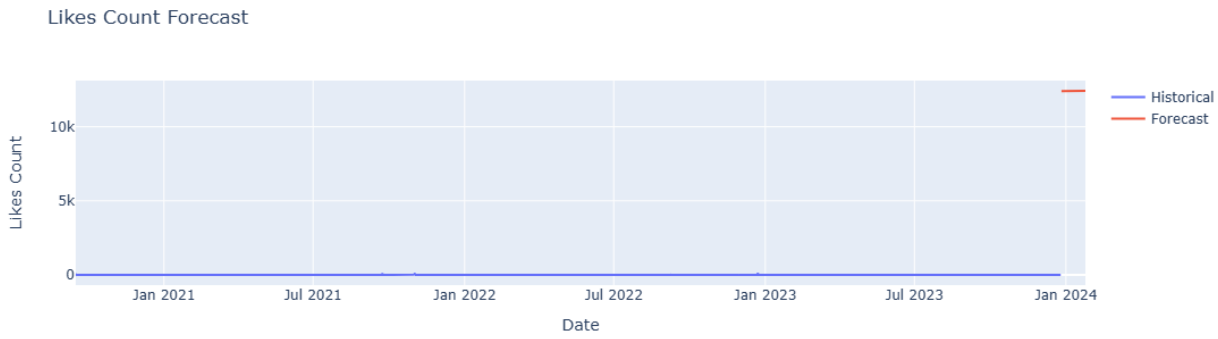
```
# Convert Index objects to Series objects
historical_dates = pd.Series(data.index, name='Date')
future_dates_series = pd.Series(future_dates, name='Date')

# Create interactive visualization
fig = go.Figure(data=[
    go.Scatter(x=historical_dates, y=data['likesCount'],
name='Historical'),
    go.Scatter(x=future_dates_series, y=future_likes, name='Forecast')
])

fig.update_layout(title='Likes Count Forecast', xaxis_title='Date',
yaxis_title='Likes Count')
plt.savefig('Likes Count Forecast__002.jpeg')
fig.show()
```




```
<Figure size 640x480 with 0 Axes>
```

```
# Visualization


import plotly.graph_objs as go
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)



# Convert Index objects to Series objects
historical_dates = pd.Series(data.index, name='Date')
future_dates_series = pd.Series(future_dates, name='Date')

# Create interactive visualization
fig = go.Figure(data=[
    go.Scatter(x=historical_dates, y=data['likesCount'],
```

```
name='Historical'),
    go.Scatter(x=future_dates_series, y=future_likes, name='Forecast')
])

fig.update_layout(title='Likes Count Forecast', xaxis_title='Date',
yaxis_title='Likes Count')
# plt.savefig('Likes Count Forecast__002.jpeg')
fig.show()
```



Scenario Analysis:

```
# Scenario Analysis
# Define scenario parameters
from sklearn.model_selection import train_test_split
from statsmodels.tsa.arima.model import ARIMA
import pandas as pd

scenario_params = {'increase': 0.1, 'decrease': -0.1}

# Evaluate scenario impact
for scenario, param in scenario_params.items():
    # Modify likesCount data according to scenario
    data_scenario = data['likesCount'] * (1 + param)

    # Re-run modeling and forecasting steps
    X = data_scenario.index.values.reshape(-1, 1)
    y = data_scenario.values

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Train model
    model = ARIMA(y_train, order=(5,1,0))
```

```
    model_fit = model.fit()

    # Forecast future values
    forecast = model_fit.forecast(steps=30)

    # Print scenario-specific results
    print(f'Scenario {scenario} results:')
    print(f'Forecasted values: {forecast}')
    print(f'Model parameters: {model_fit.params}')
    # print(f'Model parameters: {pd.Series(model_fit.params)}')

    print('------------------------')
```

```
Scenario increase results:
Forecasted values: [ 6.98388913  7.07778846  8.22380556  9.88794279
11.06773256  9.69865007
  8.816638    9.00994843  9.40379098  9.69023845  9.65330726
9.38710134
  9.31067206  9.39043626  9.47285501  9.49427364  9.45676521  9.417173
  9.41952171  9.43987043  9.45145916  9.44835714  9.43920936
9.43518788
  9.43823215  9.44202897  9.44281115  9.44121872  9.43971519
9.43968996]
Model parameters: [-8.06556332e-01 -6.19769630e-01 -5.05395058e-01 -
3.47532662e-01
 -1.67533081e-01  1.95292330e+02]
------------------------
Scenario decrease results:
Forecasted values: [5.71414044 5.79101399 6.72855282 8.09018065
9.05542788 7.93527156
 7.21365887 7.37181672 7.6940326  7.92840615 7.89818249 7.68038388
 7.61785634 7.6831146  7.75054502 7.76806985 7.73738094 7.70498917
 7.70691121 7.72355918 7.73304036 7.73050248 7.72301817 7.7197282
 7.72221887 7.72532513 7.72596504 7.72466221 7.72343213 7.72341152]
Model parameters: [ -0.80655637  -0.61976621  -0.50538797  -0.34753282
-0.16752779
 130.7311853 ]
------------------------
```

```
# Scenario Analysis
# Define scenario parameters
from sklearn.model_selection import train_test_split
from statsmodels.tsa.arima.model import ARIMA
import pandas as pd

scenario_params = {'increase': 0.1, 'decrease': -0.1}

# Evaluate scenario impact
for scenario, param in scenario_params.items():
    # Modify likesCount data according to scenario
```

```
    data_scenario = data['likesCount'] * (1 + param)

    # Re-run modeling and forecasting steps
    X = data_scenario.index.values.reshape(-1, 1)
    y = data_scenario.values

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Train model
    model = ARIMA(y_train, order=(5,1,0))
    model_fit = model.fit()

    # Forecast future values
    forecast = model_fit.forecast(steps=30)

    # Print scenario-specific results
    print(f'Scenario {scenario} results:')
    print(f'Forecasted values: {forecast}')
    # print(f'Model parameters: {model_fit.params}')
    print(f'Model parameters: {pd.Series(model_fit.params)}')

    print('------------------------')

Scenario increase results:
Forecasted values: [ 6.98388913  7.07778846  8.22380556  9.88794279
11.06773256  9.69865007
  8.816638    9.00994843  9.40379098  9.69023845  9.65330726
9.38710134
  9.31067206  9.39043626  9.47285501  9.49427364  9.45676521  9.417173
  9.41952171  9.43987043  9.45145916  9.44835714  9.43920936
9.43518788
  9.43823215  9.44202897  9.44281115  9.44121872  9.43971519
9.43968996]
Model parameters: 0     -0.806556
1     -0.619770
2     -0.505395
3     -0.347533
4     -0.167533
5    195.292330
dtype: float64
------------------------
Scenario decrease results:
Forecasted values: [5.71414044 5.79101399 6.72855282 8.09018065
9.05542788 7.93527156
 7.21365887 7.37181672 7.6940326  7.92840615 7.89818249 7.68038388
 7.61785634 7.6831146  7.75054502 7.76806985 7.73738094 7.70498917
 7.70691121 7.72355918 7.73304036 7.73050248 7.72301817 7.7197282
 7.72221887 7.72532513 7.72596504 7.72466221 7.72343213 7.72341152]
```

```
Model parameters: 0     -0.806556
1     -0.619766
2     -0.505388
3     -0.347533
4     -0.167528
5    130.731185
dtype: float64
------------------------
```

Uncertainty Quantification:

```
# Uncertainty Quantification
import numpy as np

import pandas as pd

# Assuming 'climate_nasa.csv' is the data file
data = pd.read_csv('climate_nasa.csv')

# Perform Monte Carlo simulations
simulations = 1000
likesCount_simulations = np.zeros((simulations, len(data)))

for i in range(simulations):
    likesCount_simulations[i] = data['likesCount'] +
np.random.normal(scale=data['likesCount'].std(), size=len(data))

# Calculate uncertainty metrics
mean_simulation = likesCount_simulations.mean(axis=0)
std_simulation = likesCount_simulations.std(axis=0)

print('Mean Simulation:', mean_simulation)
print('Standard Deviation Simulation:', std_simulation)
```

```
Mean Simulation: [ 2.12969640e+00 -4.02216761e-01  1.33407214e+00
4.52785050e+00
  1.63824578e+01  4.65628481e+00  1.04029758e+01  1.34451228e+00
 -2.27294459e-01  4.74474681e+00  6.17416863e+00  9.47407051e-01
  3.19408708e+00  1.76362965e+01  2.72236122e+00  1.40344153e+01
  4.13311888e+01  1.73685604e+00 -8.04603543e-01 -2.21689429e-01
 -1.08483581e-01  2.03195556e-01  1.54664054e-01 -1.87577988e-01
 -3.93883925e-01  5.64769401e-01 -2.23742238e-01  7.56059881e-01
  1.97823610e-01  3.70991798e+00  8.91543981e-01  2.15580107e+01
  1.89420405e+01  3.25126076e+00  3.16212483e+00  6.03338060e-01
  2.15561275e+00  4.38514521e+00  2.79035174e+00 -6.64512628e-01
  1.14104111e+00  1.28628051e+00  1.17753211e+01  5.96714675e-01
  1.96678270e-01  2.91151090e-01  2.66189222e-01  1.85883061e+01
  1.75017101e+00 -1.29312858e-01  6.23598986e-01  1.30758045e+00
```

```
 1.49700877e+00  2.39058680e+00  1.00746987e+00  1.28298384e+00
 3.81234527e+00  1.75899143e+00  4.90765284e+00  1.06629211e+00
 3.88402863e+00  1.81420935e+00  1.76182294e+00  1.39761093e+00
 8.86282518e+00  7.54672559e-01 -1.78852537e-01  1.76849628e+00
 1.39176534e-01  5.94196488e-02  4.40818464e-02 -1.58685026e-01
 1.02818075e+00 -2.96354591e-01  7.87070993e-01  2.41312304e-01
-9.80309293e-02 -6.46248796e-02  1.57685957e+00  9.70650903e+00
 2.40513065e+01  1.74638575e+01  3.24614440e+01  1.00679415e+01
-2.99290964e-02  8.64268601e+00  1.33689558e+00  1.80008868e-02
-6.58018669e-01 -5.17097957e-01  1.13027972e-01 -4.32166762e-01
 4.34122764e-01  1.29304241e+00  1.76766244e-01  3.04825725e-01
 5.29606354e+00  5.32850250e-01  1.44947831e+00  1.26999553e-01
 1.36575940e+00 -2.07036787e-01  1.32251902e+01  5.35588552e+00
 2.56055298e+01  6.15210608e+00  1.24788857e+00 -3.90489775e-01
-7.01112268e-01  1.88814174e+01  4.32295739e+01  8.93685730e+00
-2.94204211e-01  2.79802255e+00  1.46705331e+01  1.12716147e-01
 2.02191985e-01  2.77842496e+00 -5.15106772e-03  4.43443421e-01
 7.82180017e-01 -6.01751351e-02  2.17378262e+01  1.39333839e+00
 2.04255817e+00  5.77138696e+00  1.09857519e+02  4.40795124e+00
 1.79162704e+01  1.16992310e+00  7.66035551e+00  5.76004406e-01
 1.16746643e-01  1.60543905e-01 -3.46140108e-01  2.84701901e+00
 1.96001789e+00  8.10407342e-02  4.60661308e-01 -3.34618081e-01
 7.35385360e-01  1.97484432e+00  1.02894850e+00  1.00083426e+00
-2.59511750e-01  1.86754384e+00  8.90541054e+00  1.54963174e+01
 2.87217645e+00  1.34241982e+00 -1.74518043e-01  1.89726742e+00
 4.24171028e+00  3.97330174e-01  2.41210461e+00  2.48127359e-01
 4.45960541e+00 -8.99820012e-02  3.33652771e+00  3.81349630e+00
 8.89720081e-01  2.59772172e+00  8.49738955e-02  1.60868635e+01
 1.81698990e+00  6.49161117e+00  1.62040264e+00  2.11742667e+00
 9.67325651e-01  2.48526385e+00  5.18023530e+00  1.01113245e+01
 4.68820795e+00  2.83292970e+01  4.75524492e-01  1.46742098e+01
 1.63434404e+01  3.39831911e+00  7.10625402e+00 -4.37990145e-02
-4.08866763e-01  9.91581411e-02  8.26835928e-02  2.81841502e+00
 4.10098424e-02  1.20487977e+00  1.08479772e+01  5.69110007e+00
 8.83466522e-01  5.75965696e+00 -2.69962104e-01  3.04876566e+00
-3.99389194e-01  9.76147742e-01  9.87669581e+00  2.22155300e-01
-7.72094895e-02  2.29085368e+00  1.13669110e+00  9.97400360e-01
-1.12658947e-01  1.02034393e+00  6.54824878e+00  2.10199686e-01
 3.70188779e+01  1.58673084e+00 -3.43223220e-01  2.05210986e+00
-1.65923353e-01  9.07427794e+00  1.66221288e-01 -6.55085652e-02
 9.19114876e-03  5.41029576e+01  6.20167849e+00  3.18824288e+00
 1.33389373e+01  4.13225691e+00 -2.97682905e-01 -2.14898106e-01
 1.13698051e+00  3.23822164e+00  2.32129187e+00  1.07274105e+00
-3.63563895e-02  7.26112975e-01 -1.63875700e-02  4.42364084e-01
 3.48420714e-01 -4.98132934e-01  1.09885295e+00  2.42297409e+00
 1.24324923e+00  6.19828181e-01 -2.72872472e-02 -8.45054703e-02
 1.33132601e+00  7.59496558e-01 -1.21513868e-01 -1.69964646e-01
 9.03725425e-02 -5.16009795e-02  1.05982148e+00  3.96950405e+00
 1.04278797e+00  2.66620889e+00  2.40505258e-01  1.22588545e+00
```

```
 1.35289573e+00  1.13572970e+01  4.56473320e-01  1.23931051e+00
 1.10158709e+00  1.04378861e+00  2.41187508e+00  2.67695620e+01
 2.16267265e+01  1.80013998e+01  5.91473706e+00  1.24206969e+01
 5.07953193e+00  3.96838815e+00  4.43477990e+00  3.48994598e+00
 6.16429488e+00  2.09718290e+00  9.11488843e-01  1.12667885e+00
 1.37038011e+00  1.37854369e+00  2.63618386e+00  5.71187418e-01
 3.48497791e-02  9.69987919e-01 -2.71245768e-01  2.19539460e+00
-9.02377705e-02  1.39162355e-01 -2.48588742e-01  1.20200750e+00
 1.06178063e-01  2.93007278e-01 -6.96039619e-02  1.84144607e-02
-6.72598246e-01  9.39486255e-03  4.00497971e+00 -6.85671306e-02
 4.29447654e-01 -3.61461577e-02  2.75471923e-01  2.22920898e-01
-1.57886983e-01  3.00045573e+00  2.14045916e-01  1.57881640e-01
 2.57333962e+00  1.96204372e-02 -1.46546465e-02  2.09836611e-01
 7.78167655e-01  1.22368034e+00  2.48714259e+00  2.45564055e+00
 3.49540113e+00 -1.22934527e+00  8.30005261e-01 -2.33315008e-02
-6.74171836e-01  5.09077583e-01 -5.78767608e-02 -7.67144285e-02
 8.18558863e-01  1.69920002e-01 -4.13561609e-01  1.60655573e+00
 7.83375904e+00  7.77139700e-01  9.00553992e-01  2.83630084e-02
 1.14224376e+01  1.89966776e+00  1.59499063e-01 -3.31141816e-01
 3.09972010e-01 -1.03883586e-01  4.49135155e-01  1.38759624e-01
-5.30637711e-01 -4.18884912e-01  2.34791765e-01 -3.69120786e-01
 2.02359506e-01 -2.82369736e-01 -4.92838881e-01 -9.96943958e-02
-1.77362556e-01  4.61511898e-03  1.34388278e-01  3.52398039e-01
-1.06164017e-01  4.87285174e-01  2.76841021e+00  2.27756015e-01
 2.82682759e+01  1.21589793e+01 -1.02760884e+00 -3.04438204e-01
 2.40539378e+01  1.95055194e+01  8.12931278e-01  9.30235723e+00
 3.16713697e+00  9.07595837e+00  1.52292421e+01  2.35258320e+00
 1.69314659e+01  4.43320930e+00  5.75461462e+00  6.82252832e+00
-4.15795513e-02  1.09796068e+00  3.97686148e+00  5.48729292e+00
 1.83242576e+01 -3.43470528e-01  1.49390050e+00  8.16116664e-01
 1.69131233e-01  8.02732139e-01  3.54047197e+00  5.90550754e+00
 3.43096392e+01 -2.77021387e-01  1.43265084e-01 -1.31456854e-01
 1.21926238e-02 -9.52728557e-02  1.31884410e-01  2.10845084e+01
 4.16388794e+00  1.84879103e+00 -3.24289823e-01  2.26680878e+00
 4.82198688e-01 -3.05507323e-01  5.53275792e-01 -1.70499155e-01
 2.86295407e+00  2.91223043e+00  3.29271152e-01 -2.90825799e-01
 9.82270139e-01  1.19225319e+00  2.79154506e+00  2.42021269e+01
 4.22167360e+00  1.19683977e+02  4.21938757e+00  3.26226420e+00
 4.27414536e-01  9.91300499e-01  5.12090696e-01 -2.23646292e-01
 9.45428178e-01 -3.14853475e-02  5.81389577e-01  3.18779816e+00
 3.43434361e+01 -6.00497098e-02  9.97199703e-01  1.34323869e-01
-9.70177418e-01 -7.28616804e-01  2.87298022e+00  1.96815513e+00
 4.88001336e+00  1.56479993e+00  4.69091031e+00  1.22503158e+00
 1.90174371e+01  5.05115600e-01  3.01499827e+00 -3.13666548e-01
 7.27989695e-02  1.48801245e+01  3.75528564e+01  6.17032953e+01
 8.87661344e+00  1.85863084e+00 -6.68944707e-02  3.45002388e-01
 1.00492941e+00  1.26043765e+02  6.51917766e+00  1.70172888e+01
 7.91364968e+00  4.28013224e+01  3.32254653e-01 -5.34314390e-01
-1.46493141e-02  1.00405474e+01  3.82321313e+01  2.82348730e+01
```

```
 -1.56648066e-01  1.29592720e+00 -1.08981492e-01  3.85667196e-01
  5.58581740e-01  5.81794823e-01  5.56174826e+01  3.84984482e-02
 -2.57286312e-01  3.30765292e+00 -1.88578288e-01  4.88133985e-01
  1.24834494e+00  7.28446524e-01 -1.31594974e-01  8.02633025e-02
  2.53302567e-01 -9.72581672e-02 -3.72894200e-01  1.15443259e+00
 -1.83074972e-02 -3.65587364e-01  5.23435943e-02  1.30220958e+00
 -2.18144605e-01  9.63483187e-02 -1.47799276e-01  5.26161566e-01
 -1.25541426e-01  3.96750040e+00 -6.52617409e-01  4.71042851e-02
 -4.28019937e-01  1.74402312e-01 -5.64916876e-01 -1.04274428e-02
  2.41782631e-01 -2.09671020e-03  3.80246643e-01  1.17050189e+00
 -5.59363090e-01  3.29765539e+00  6.07364464e-01  1.29996124e-01
  6.56116648e-01  4.04288058e-01  5.41966259e-02  1.60826301e+00
  1.47875042e+01  1.20823673e+01  2.78831694e+00  1.60412946e+01
  1.03733618e+00  1.22221179e+01  2.47649708e+00  1.34059324e+01
 -2.98217462e-01  1.13517217e+01  4.62038781e+01  3.22014582e+01
  2.18445880e+00  2.43979313e+00 -3.01224391e-01  1.24046465e+00
  2.10661668e+00 -4.07760879e-01  3.63229787e-01  2.91101404e+00
 -3.08891135e-01  2.56060099e-01  7.37650112e-01 -5.48520237e-02
  6.45643515e+00 -6.33325847e-02  8.47967819e-01  3.50093978e+00
  1.01142718e+00  1.17134724e+01]
Standard Deviation Simulation: [12.07378497 11.7232923  11.68583223
12.38727229 11.56664611 11.99843821
 11.47614641 12.01234663 11.83345528 12.34668427 12.38960794
12.6671555
 12.02767528 11.68801814 12.09047672 12.72483686 11.61134002
11.88010165
 11.93707388 12.2968689  12.02748593 12.15842016 12.09156971
11.79465232
 12.28606693 12.02949626 12.21220225 11.77016582 12.16590462
12.49890394
 12.60782605 12.18642604 11.50378109 12.55954423 11.68080717
11.81961997
 12.22601916 11.86214984 12.02896205 11.89072273 11.60185234
12.04821529
 12.21828675 12.66854041 12.45901074 12.2001501  11.8135965
12.03655467
 12.13894026 12.21079605 12.15885693 11.7961739  12.31031896
12.13283274
 11.75858606 12.0336266  11.75587013 12.39427896 11.51339213
11.65956337
 11.70600176 12.2750949  12.32189903 12.0053563  11.71804522
12.22084739
 12.16182379 11.89714463 12.22831668 12.37300404 11.61008354
11.94075988
 11.72368126 12.43050876 12.10213512 12.01499187 12.04759751
12.17760101
 12.41472362 12.18818957 12.13923216 12.45969399 11.77785007
11.57875908
 12.21802707 11.56992225 11.92142464 11.81684599 12.20577009
```

12.0105303
 11.99018147 12.01079036 12.32640922 11.58418708 11.87522143
11.99149016
 11.97431606 11.26534837 11.71434787 12.09671706 12.2128178
12.33418328
 12.08710138 11.64400457 11.99098189 11.96158695 12.2150176
11.80730753
 12.19948808 12.37820548 12.15935475 12.25624639 12.32285547
12.40605487
 11.99994789 11.65699157 12.16181903 11.91085483 11.6144577
12.20866167
 11.82568082 12.5332794 12.14795017 12.05706285 11.87440788
12.02178983
 12.49114343 11.92688207 11.72481624 11.83281168 12.21164969
12.34502669
 11.83865453 12.16332324 11.9497381 12.33254004 11.70729087
11.25118167
 12.10829833 12.24748078 12.03213349 12.17923449 12.01856291
12.44865562
 11.64877698 11.83812161 12.17478342 11.54321643 11.83939523
11.79707255
 12.36822541 11.8475028 12.14358627 11.67660593 11.92641536
11.80784886
 12.06502583 12.48793241 11.64738463 12.21839494 11.99692327
11.65420554
 11.9307466 11.88719733 11.92529337 12.35865017 11.69289566
12.01109275
 12.04521295 11.91103288 12.02987908 12.08960833 12.58033737
12.54966824
 12.27128046 11.47277745 12.20578825 12.30256974 12.03821647
12.20185352
 12.33553152 12.28770797 11.60000132 11.73262232 11.98413503
11.91090247
 11.67196576 12.31551973 12.27637454 11.58296129 12.54157538
11.99538133
 11.91037638 12.20198462 11.62214344 12.00827203 12.40592588
12.11619041
 12.30856121 11.71347852 12.23036289 11.62474341 11.49872463
12.25124026
 12.26085114 11.47141598 11.60070203 11.7674376 12.13339143
11.67307342
 12.38113568 11.92209095 12.27660506 11.76089517 11.75701592
12.09235336
 12.1216292 11.80735994 12.30355691 11.56989009 12.06959615
11.71321281
 12.11789328 11.9849801 11.86433755 12.46296842 12.17567911
12.37834773
 11.75243832 11.81057091 12.81078561 11.87009801 11.74370875 11.963088
 12.07961954 11.9114308 11.84979753 12.24337983 12.08485561

12.08467413
 12.00647852 12.28505167 11.42377247 11.82380408 11.63096141
12.09058966
 12.59696786 12.01050895 12.32959177 11.74369519 12.22021082
12.25661117
 12.13477842 11.85357311 11.97711666 11.87959306 12.13223987
12.53039922
 12.15911131 11.86449461 12.25017916 12.02229544 12.37008186
12.43392967
 12.08131971 11.90233747 11.82512985 12.29957542 11.72984326
11.59043798
 11.75979446 11.94762011 11.59290578 12.00980611 12.45311964
12.07288909
 11.96673543 11.85260635 11.96118195 11.67264893 11.86460138
12.03971742
 11.93232963 11.78579555 12.11562028 11.92542527 12.07895152
12.15667484
 11.7914847 11.84062538 12.11512614 11.77850232 12.42510708
12.23925358
 12.17186784 12.09989705 11.91090696 12.1791422 12.05807041
12.07160786
 11.8604974 11.63791645 12.0027916 11.77382982 12.68567834
11.97644105
 12.11489073 11.91352509 12.15838984 12.08482349 12.35493162
12.28943818
 12.42587298 12.21873263 12.19244647 12.0713781 12.40634529
12.33738412
 11.78527716 12.43644497 11.94051417 11.81597115 11.85979462
12.42419214
 11.70276   12.01890617 12.41481437 12.16427226 12.22933992
11.86606638
 12.07087721 12.68638306 11.9718465 12.02569732 11.97279283
12.02323108
 11.88674765 12.18982352 12.45732868 11.85720465 12.43061013
12.55975809
 11.48607225 11.74742314 12.06880648 11.99657521 11.8953295
12.07139851
 11.96945131 11.53210014 12.19691255 11.61269197 11.90859829
12.34641107
 12.58456348 11.69049184 12.25969596 11.565571   12.69793323
11.91954251
 11.75627099 12.27225938 12.12003614 11.74719424 12.39726179
12.08424178
 12.13076017 12.36980779 12.52368857 11.76695826 12.03209334
12.13023171
 12.06487026 11.80347309 12.38318834 11.96862235 12.4720317
11.73952343
 11.81209393 11.77571612 12.06091495 12.42432694 11.80652062
12.11296143

```
 12.37583714 12.12041929 11.62908991 12.12894592 12.06034932
12.07927437
 11.90385409 12.33031293 11.94805569 12.23418326 12.03992929
11.70992759
 12.40842008 11.72087005 11.72068058 11.97533757 12.24550479
11.53408355
 12.20224379 12.04917127 11.58272103 11.77706619 12.44546259
12.44050104
 12.19148844 11.74013165 12.30468453 12.02180243 11.62073773
11.97569098
 11.97805978 12.27067473 12.26322452 12.22216592 12.05255274
12.2119254
 12.2946574  11.79140626 12.45418383 11.88761229 12.00926917
12.3126508
 11.90550811 12.4281086  12.01043371 11.71465412 11.5679931
12.26811296
 11.97494355 12.19070674 11.90257458 12.47319284 12.79076068
12.43724606
 12.28704953 11.95313063 12.18632996 11.82073547 11.80055857
12.17962738
 12.25317344 12.0087688  11.87333185 12.19511513 12.21816269
12.34180214
 12.1066395  12.20530194 12.1114199  12.0239435  12.52293808
11.60166118
 12.0879018  12.38795597 12.1826501  11.97399434 12.02750917
11.78175926
 12.39250888 12.09429285 12.00535998 12.41333399 12.20803525
12.10358776
 12.17246698 11.66268517 11.65630704 11.85240878 12.25768886
11.6021838
 11.75471533 11.46662657 11.5820787  12.01820865 12.7077072
11.98228063
 11.93046971 12.37817219 12.48669028 11.72863602 11.44780622
12.3543478
 11.75331715 12.10012115 12.28908338 12.00016579 11.80880163
12.0207789
 11.95294069 11.87821819 12.37680515 12.33992811 12.23082285
11.58116646
 12.1967511  11.97731243 12.02660898 11.96547394 12.37192502
11.54775624
 12.29125334 12.02954196 12.1140408  12.04336364 12.44083377
11.99243643
 12.24355136 11.38714174 12.49103323 11.89392165 11.79519112
11.67384659
 12.05972304 12.38396842 12.43691389 12.20804521 12.35586755
11.77963402]
```