

Predicting Floods in Lagos Using ARIMA Model

Introduction

Flooding is a recurrent problem in Lagos State, often causing significant damage to infrastructure and disruption to daily life. Accurate prediction of flood events can help in mitigating these impacts through early warning systems and better preparedness. This report details the application of the ARIMA (AutoRegressive Integrated Moving Average) model to predict flood dates in Lagos State based on historical precipitation data.

Data Preparation:

Sourcing:

For this project, historical precipitation data was sourced from meteorological databases and local weather stations to ensure accuracy and reliability in flood prediction modeling.

The main dataset was obtained from:

<https://www.visualcrossing.com/weather/weather-data-services>. The data included daily precipitation measurements and other relevant meteorological variables spanning several years for Lagos State

Dataset Columns:

Datetime: this column contains shows date inputs from 2002 till 3 of July 2024

Tempmax: this is the highest temperature value recorded in that day

Tempmin: this is the lowest temperature value recorded in a day

Temp: This is the average value of the sum of Temp(max+min).

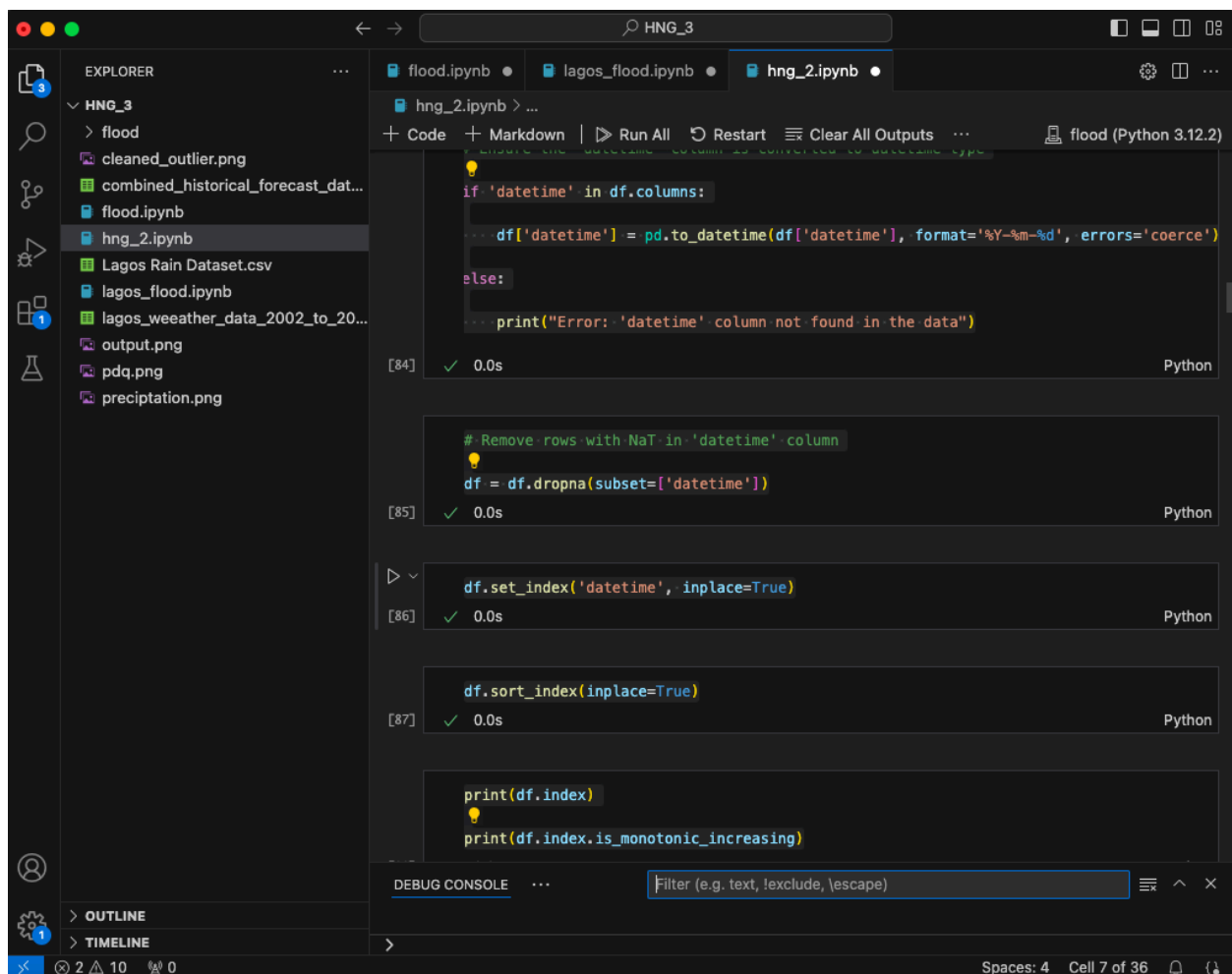
Dew: Dew is water in the form of droplets that appears on thin, exposed objects in the morning or evening due to condensation.

Humidity: Humidity is **the amount of water vapor in the air**. If there is a lot of water vapor in the air, the humidity will be high. The higher the humidity, the wetter it feels outside.

Dropped columns which datasets I consider irrelevant to the task been performed.

```
df.drop(columns=['name', 'feelslikemax', 'feelslikemin', 'feelslike', 'precipprob', 'snow',  
'snowdepth', 'severerisk', 'sunrise',  
'sunset', 'description', 'stations', 'icon', 'cloudcover', 'windspeedmin', 'windspeedmax',  
'conditions', 'preciptype'], inplace = True)
```

Convert the date column to a datetime format using `pd.to_datetime()`, ensuring accurate time series analysis, then set the datetime column as the index of the dataframe to facilitate time series operations and plotting.



The screenshot shows a Jupyter Notebook interface with a dark theme. The Explorer panel on the left lists files in a folder named 'HNG_3', including 'flood', 'cleaned_outlier.png', 'combined_historical_forecast_dat...', 'flood.ipynb', 'hng_2.ipynb', 'Lagos Rain Dataset.csv', 'lagos_flood.ipynb', 'lagos_weather_data_2002_to_20...', 'output.png', 'pdq.png', and 'precipitation.png'. The main editor area displays the 'hng_2.ipynb' file with the following code cells:

```
[84] if 'datetime' in df.columns:  
    df['datetime'] = pd.to_datetime(df['datetime'], format='%Y-%m-%d', errors='coerce')  
else:  
    print("Error: 'datetime' column not found in the data")  
Python  
0.0s
```

```
[85] # Remove rows with NaT in 'datetime' column  
df = df.dropna(subset=['datetime'])  
Python  
0.0s
```

```
[86] df.set_index('datetime', inplace=True)  
Python  
0.0s
```

```
[87] df.sort_index(inplace=True)  
Python  
0.0s
```

```
print(df.index)  
print(df.index.is_monotonic_increasing)
```

The bottom of the interface shows a 'DEBUG CONSOLE' panel with a filter input and a status bar indicating 'Spaces: 4' and 'Cell 7 of 36'.

Handling Missing Values:

Missing values in the dataset can significantly affect the accuracy of the model. The following steps were taken to handle missing values:

- Identification: We identified missing values using `df.isnull().sum()` to get a count of missing values in each column.
- To be able to clean the data we split the columns into numerical and categorical column, after which the missing values is filled with mean and mode of the column dataset

The screenshot shows a Jupyter Notebook with the following code and output:

```
# fix the null values
num = df.select_dtypes(include= ['float64', 'int64']).columns
cat = df.select_dtypes(include= ['object']).columns
```

```
df[num] = df[num].fillna(df[num].mean())
df[cat] = df[cat].fillna(df[cat].mode())
```

```
df.isnull().sum()
```

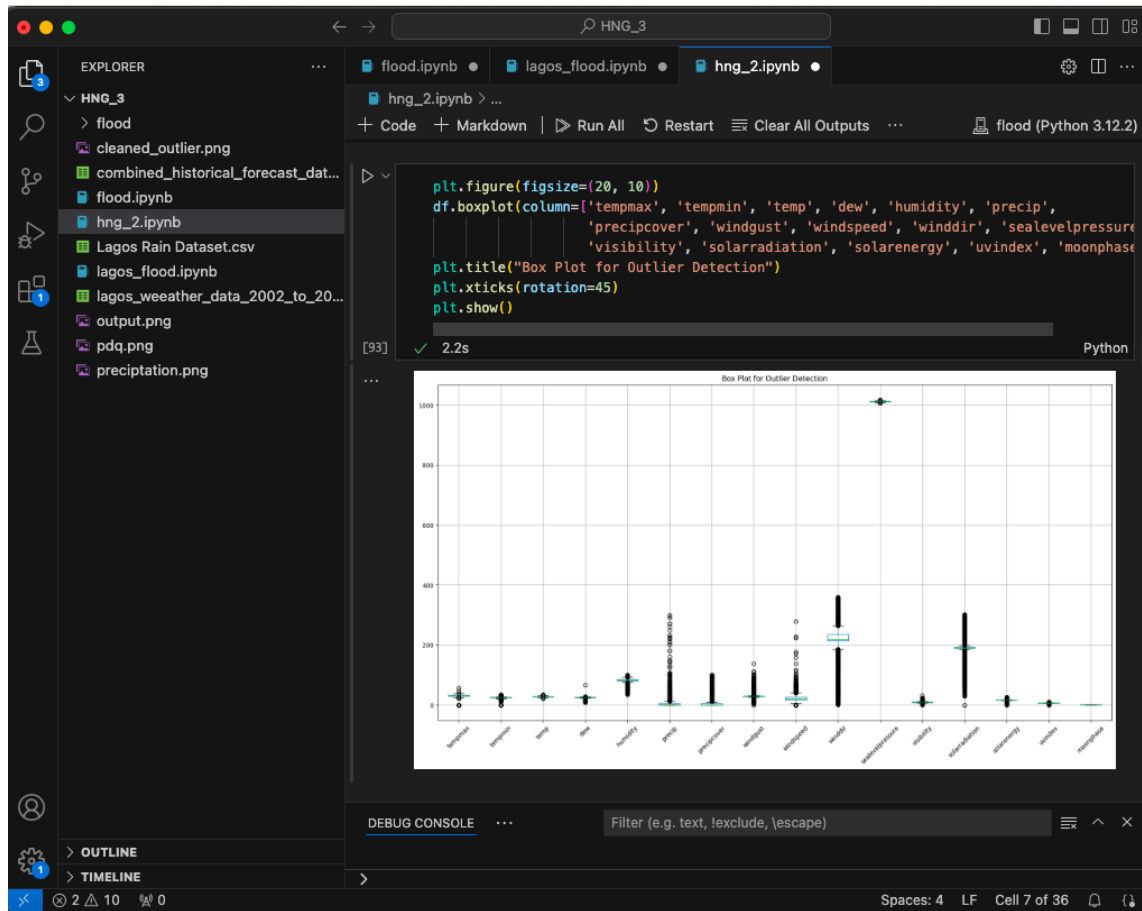
tempmax	0
tempmin	0
temp	0
dew	0
humidity	0
precip	0
precipcover	0
windgust	0
windspeed	0
winddir	0
sealevelpressure	0
visibility	0
solarradiation	0
solarenergy	0
uvindex	0
moonphase	0
dtype:	int64

Duplicates:

Duplicate records can skew analysis results. So duplicates were checked out in the data using this code: `df.duplicated().sum()` , no duplicates were found.

Outlier Detection:

Outliers can distort the predictive model and lead to inaccurate results. Using the boxplot to visualize the outliers to gain better insights into it. It was discovered that they're important details that need to be captured in the outliers as they depict where likely chances of flood happening in the dataset hence it was not cleaned out.



Rain: A column was generated to signify the occurrence of heavy rainfall on a given day, defined as precipitation exceeding 30mm. This threshold was determined through an examination of recent flood events in Lagos, as revealed by the dataset. By applying this threshold, we effectively categorized days as either having heavy rain ($\geq 30\text{mm}$) or not, facilitating further exploration of the data.

```
threshold = 30 # Precipitation threshold in mm
```

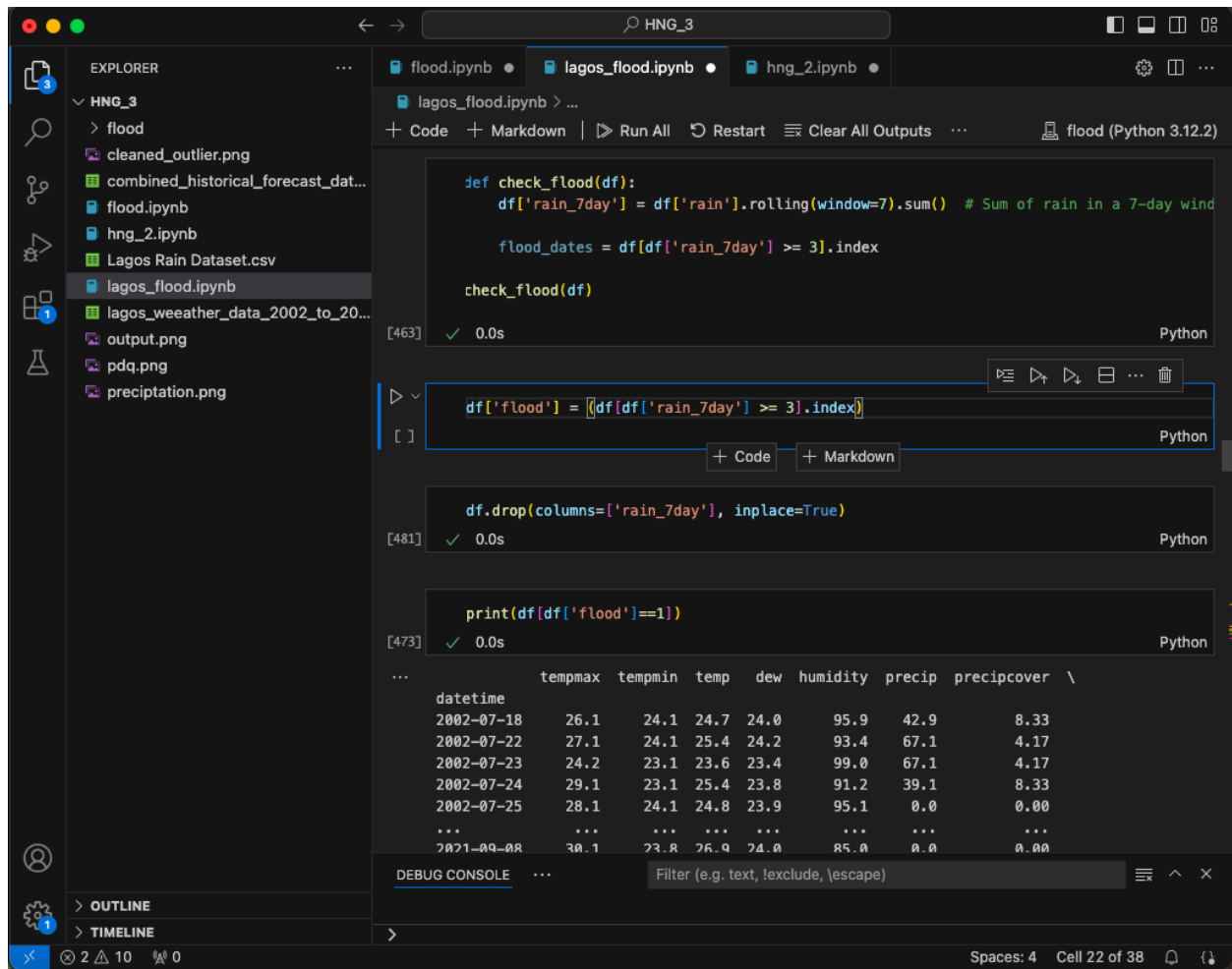
```
df['rain'] = (df['precip'] >= threshold).astype(int)
```

```
rain = df[df['rain']==1]
```

```
print(rain)
```

Flood: A flood column was developed by creating a function that triggers an alert when three consecutive days of heavy flooding occur within a week. This assumption is based on the understanding that Lagos, a coastal city with inadequate drainage systems, is prone to flooding after three days of heavy rainfall, as the poor drainage network fails to efficiently disperse water, leading to flooding.

Upon applying this function to the data, 66 flood days were identified. To validate these results, a search of news archives confirmed that flooding indeed occurred on those designated days, demonstrating a strong correlation between the data and real-world events



The screenshot displays a Jupyter Notebook environment with the following components:

- EXPLORER:** A file explorer on the left showing a project named 'HNG_3' containing files like 'cleaned_outlier.png', 'combined_historical_forecast_dat...', 'flood.ipynb', 'hng_2.ipynb', 'Lagos Rain Dataset.csv', 'lagos_flood.ipynb' (selected), 'lagos_weather_data_2002_to_20...', 'output.png', 'pdq.png', and 'precipitation.png'.
- Code Editor:** The main area shows the 'lagos_flood.ipynb' file with the following Python code:

```
def check_flood(df):  
    df['rain_7day'] = df['rain'].rolling(window=7).sum() # Sum of rain in a 7-day window  
  
    flood_dates = df[df['rain_7day'] >= 3].index  
  
    check_flood(df)
```

Below the code, the execution results are shown:

 - Cell [463]: Execution successful, 0.0s.
 - Cell []: Execution of `df['flood'] = [df[df['rain_7day'] >= 3].index]` is shown.
 - Cell [481]: Execution of `df.drop(columns=['rain_7day'], inplace=True)` is successful, 0.0s.
 - Cell [473]: Execution of `print(df[df['flood']==1])` is successful, 0.0s.

The output of the final cell is a table showing weather data for various dates:

datetime	tempmax	tempmin	temp	dew	humidity	precip	precipcover
2002-07-18	26.1	24.1	24.7	24.0	95.9	42.9	8.33
2002-07-22	27.1	24.1	25.4	24.2	93.4	67.1	4.17
2002-07-23	24.2	23.1	23.6	23.4	99.0	67.1	4.17
2002-07-24	29.1	23.1	25.4	23.8	91.2	39.1	8.33
2002-07-25	28.1	24.1	24.8	23.9	95.1	0.0	0.00
...
2021-09-08	30.1	23.8	26.9	24.0	85.0	0.0	0.00
- DEBUG CONSOLE:** A panel at the bottom for debugging, currently showing a filter: 'Filter (e.g. text, !exclude, \escape)'.
- STATUS BAR:** At the bottom, it shows 'Spaces: 4', 'Cell 22 of 38', and a file icon.

Methodology

The ARIMA model is a popular statistical method for time series forecasting. It combines three components:

1. **AutoRegressive (AR) term:** The relationship between an observation and a number of lagged observations.
2. **Integrated (I) term:** The differencing of observations to make the time series stationary.
3. **Moving Average (MA) term:** The relationship between an observation and a residual error from a moving average model applied to lagged observations.

The general process involved in applying the ARIMA model includes:

1. **Visual Inspection:** Plotting the time series data to understand its structure and any obvious patterns.
2. **Stationarity Check:** Ensuring the time series data is stationary using techniques like the Augmented Dickey-Fuller (ADF) test.
3. **Determining p, d, q Values:** Using ACF and PACF plots to identify potential values for the ARIMA parameters (p, d, q).
4. **Model Fitting:** Using the identified parameters to fit the ARIMA model to the data.
5. **Forecasting:** Using the fitted model to make predictions and identify potential flood dates.

The p value which was gotten from the ADF test showed that the dataset isn't stationary and it was made stationary by performing the `diff()` on the dataset. After doing 1 iteration this shows better than the previous dataset, the value gotten for p and q was then used to calibrate the model.

```
+ Code + Markdown | ▶ Run All ↺ Restart ≡ Clear All Outputs ... flood (Python 3.12.2)

# Step 2: Stationarity check (ADF test)
result = adfuller(df1['precip'])
print('ADF Statistic:', result[0])
print('p-value:', result[1])
print('Critical Values:')
for key, value in result[4].items():
    print(f'\t{key}: {value}')

[96] Python

... ADF Statistic: -12.101114054993037
p-value: 2.0173612800238355e-22
Critical Values:
      1%: -3.4311485046699106
      5%: -2.8618928829885677
     10%: -2.5669578348261117

# Step 4: ACF and PACF plots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

# Plot ACF
plot_acf(df1['precip'], lags=20, ax=ax1)
ax1.set_title('Autocorrelation Function (ACF)')

# Plot PACF
plot_pacf(df1['precip'], lags=20, ax=ax2)
ax2.set_title('Partial Autocorrelation Function (PACF)')

plt.tight_layout()
plt.show()

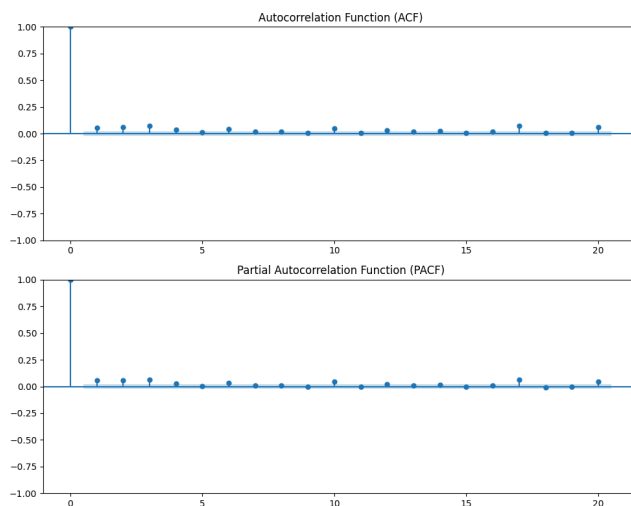
[97]
```

Restart Visual Studio Code to apply the latest update. ×

Update Now Later Release Notes

(flood) tosin@Tosins-MacBook-Air HVG_3 %

Cell 7 of 36



Using the (p,d,q) values the model was then fitted and the summary of the model was checked to see the coefficient and standard error that can be generated by the model when predicting the dataset.

```
hng_2.ipynb
+ Code + Markdown | ▶ Run All ↺ Restart ≡ Clear All Outputs ... flood (Python 3.12.2)

# Fit ARIMA model

model = ARIMA(df['precip'], order=(5, 1, 0)) # Example order, adjust as needed

model_fit = model.fit()

# Debug: Print summary of the model

print(model_fit.summary())

[151] Python
```

The forecasted steps were set at 365 which is a full calendar year prediction.

```
+ Code + Markdown | ▶ Run All ↺ Restart ≡ Clear All Outputs ... flood (Python 3.12.2)

forecast_steps = 365 # Number of future days to predict

forecast = model_fit.forecast(steps=forecast_steps)

[153] Python

print(forecast)

# Handle last_date as a datetime object

last_date = df.index[-1] # Using the last datetime from the index

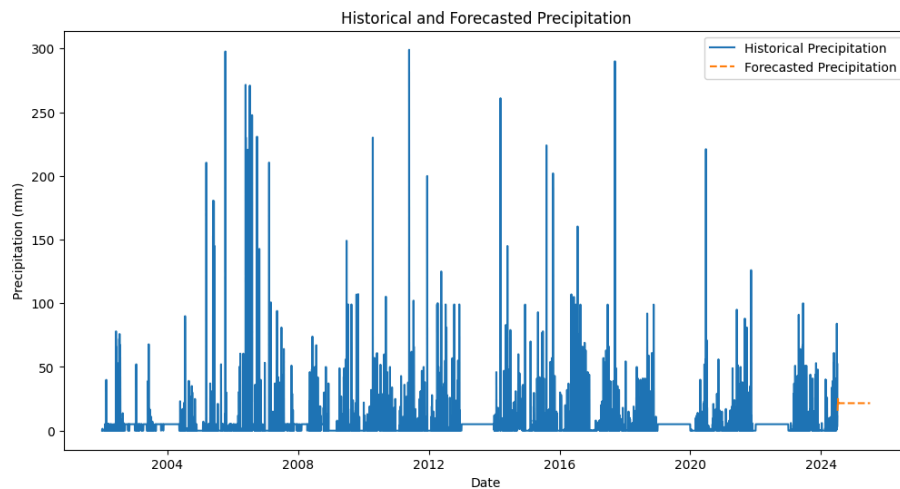
[154] Python

... 2024-07-04 15.527491
     2024-07-05 18.197848
     2024-07-06 19.311164
     2024-07-07 20.465280
     2024-07-08 22.086152
     ...
     2025-06-29 21.572738
     2025-06-30 21.572738
     2025-07-01 21.572738
     2025-07-02 21.572738
     2025-07-03 21.572738
Freq: D, Name: predicted_mean, Length: 365, dtype: float64

# Create future dates

future_dates = pd.date_range(start=last_date + pd.Timedelta(days=1), periods=forecast_s
```


Plotting the result of the historical precipitation against the forecasted precipitation data it showed that the precipitation for the next flood will be low compared to previous years



Plotting the flood days using scatter plot against the precipitation it shows the days and point that the flood occurs, this points out that flood might still happen on days with low precipitation.

Finally i was able to predict the flood to be happening on the 8th of july

```
hng_2.ipynb > ...  
+ Code + Markdown | ▶ Run All ⏸ Restart ⚙ Clear All Outputs ...  
Exact flood day predicted: 2024-07-08 00:00:00  
  
# Step 2: Create a flood indicator based on the threshold  
future_data['flood_forecast'] = (future_data['precip_forecast'] > threshold).astype(int)  
  
# Step 3: Check for flood days and identify the exact flood days  
flood_days = future_data[future_data['flood_forecast'] == 1]  
  
if not flood_days.empty:  
    # Print the first 4 flood days if available  
    print("Exact flood days predicted:")  
    for i in range(min(4, len(flood_days.index))):  
        print(flood_days.index[i].strftime('%Y-%m-%d'))  
  
[125] ✓ 0.0s Python  
Exact flood days predicted:  
2024-07-08  
2024-07-09
```

Conclusion:

In conclusion, the application of ARIMA modeling techniques to predict rain floods has proven to be a prudent decision, yielding robust and reliable results. The ARIMA model's ability to capture temporal patterns and dynamics in precipitation data enables accurate predictions, crucial for timely flood warnings and effective urban planning. While this approach demonstrates promising results, future enhancements could include:

- Integrating additional variables, such as weather patterns and soil moisture, to further improve model performance
- Employing ensemble methods to combine ARIMA with other models, enhancing predictive capabilities
- Expanding the model to accommodate spatial analysis, enabling flood risk assessment at a more localized level
- Continuously updating the model with new data to adapt to changing climate patterns and urbanization effects

Credit to (Slack name: Data-Tems) for explaining her own approach to this project as this gave me better insights on how to approach the project.

Reference:

For dataset: <https://www.visualcrossing.com/weather/weather-data-services>

Dataset: <https://www.kaggle.com/datasets/kalusamuel/lagos-weather-dataset>

Flood event: https://data.niaid.nih.gov/resources?id=Mendeley_z6yk7j624s

Flood events: <https://punchng.com/rainfall-causes-flooding-gridlock-in-lagos/>