Fetching Historical XRP Data using Binance API with Python

# Introduction

In this article, we will demonstrate how to fetch historical XRP data from the Binance API using Python. We will cover two scenarios: fetching the data for the last N days, and fetching data for a custom date range. This flexibility helps avoid duplication and confusion, allowing users to choose the approach that suits their needs.

## Why Flexibility Matters

When dealing with APIs, especially for cryptocurrency data, it's common to need historical data based on a custom time range. For example, you might want data for the last 7 days or for a specific date range, like October 1 to October 7. To prevent duplication and make the API call intuitive, we designed the script to support only two modes of operation.

## Functionality Overview

The Python script provides two ways of fetching XRP historical data from Binance's API:  
1. \*\*Last N Days\*\* - Fetches historical data for the last N days (where N is specified by the user).  
2. \*\*Custom Date Range\*\* - Fetches data for a specific time period by providing a 'from\_date' and 'to\_date'.  
  
If no 'to\_date' is provided, the script will fetch data until the current time.

## Code Breakdown

Here's the complete code for the solution. The function `fetch\_historical\_xrp\_data` takes either the number of days back or a 'from' and 'to' date range to fetch the data. It ensures that the user only needs to worry about one approach.

**Below For Live Last 1 Minute Extraction**

## import time

## import requests

## import pandas as pd

## from datetime import datetime

## def fetch\_xrp\_price():

## file\_path = "E:\\Data\_Practice\\xrp\_rates\_live.csv"

## 

## # Create the CSV file with headers if it does not exist

## if not pd.io.common.file\_exists(file\_path):

## df = pd.DataFrame(columns=["Timestamp", "XRP Price"])

## df.to\_csv(file\_path, index=False)

## while True:

## try:

## # Fetch XRP price from Binance API

## response = requests.get("https://api.binance.com/api/v3/ticker/price?symbol=XRPUSDT")

## data = response.json()

## 

## if 'price' in data:

## xrp\_price = float(data['price']) # Ensure the price is a float

## timestamp = datetime.now()

## # Create a new DataFrame for the new data point

## new\_row = pd.DataFrame([[timestamp, xrp\_price]], columns=["Timestamp", "XRP Price"])

## 

## # Read the existing CSV file

## df = pd.read\_csv(file\_path)

## # Concatenate the new row with the existing DataFrame

## df = pd.concat([df, new\_row], ignore\_index=True)

## 

## # Save the updated DataFrame back to the CSV file

## df.to\_csv(file\_path, index=False)

## print(f"Current XRP price: {xrp\_price} at {timestamp}")

## else:

## print("Error fetching XRP price:", data)

## except requests.exceptions.RequestException as e:

## print("Request error:", e)

## except Exception as e:

## print("An error occurred:", e)

## # Wait for 1 minute

## time.sleep(60)

## # Start fetching the XRP price

## fetch\_xrp\_price()

## Two Modes of Operation

The script simplifies the API call process by limiting the way you can fetch the data to two options:

1. \*\*Last N Days\*\*: If you don't pass a `from\_date`, you can fetch the data for the last N days by specifying the number of days as `days\_back`. For example, if you want data for the last 7 days, call the function like this:

fetch\_historical\_xrp\_data(days\_back=7)

2. \*\*Custom Date Range\*\*: If you provide both a `from\_date` and a `to\_date`, the script will fetch the data between those two dates. The dates should be passed in the format `yyyy-mm-dd`. Here's how to use this feature:

fetch\_historical\_xrp\_data(from\_date="2023-10-01", to\_date="2023-10-07")

## Conclusion

This Python script provides a simple and effective way to fetch historical XRP data using the Binance API. By offering two ways to fetch the data—based on the last N days or a custom date range—it ensures flexibility while avoiding confusion. You can now easily integrate this script into your own projects to analyze XRP price data or modify it for other cryptocurrencies.

**Below Code For Real Time Fetching With Customised Periodic:**

import requests

import pandas as pd

from datetime import datetime, timedelta

def fetch\_historical\_xrp\_data(from\_date, to\_date):

# Define parameters

symbol = "XRPUSDT"

interval = "1m"

# Convert from\_date and to\_date to timestamp in milliseconds (UTC)

start\_time = int(datetime.strptime(from\_date, "%Y-%m-%d").replace(hour=0, minute=0, second=0, microsecond=0).timestamp() \* 1000)

end\_time = int(datetime.strptime(to\_date, "%Y-%m-%d").replace(hour=23, minute=59, second=59, microsecond=999).timestamp() \* 1000)

# Initialize a list to hold all data

all\_data = []

# Fetch historical data in chunks

while start\_time < end\_time:

url = f"https://api.binance.com/api/v3/klines?symbol={symbol}&interval={interval}&startTime={start\_time}&endTime={end\_time}&limit=1000"

response = requests.get(url)

data = response.json()

if not data or isinstance(data, dict) and 'msg' in data: # Check for error messages

print("Error fetching historical data:", data)

break

all\_data.extend(data)

# Update start\_time to the last timestamp fetched

start\_time = int(data[-1][6]) + 1 # Move to the next minute after the last data point's Close Time

# Prepare DataFrame

df = pd.DataFrame(all\_data, columns=["Open Time", "Open", "High", "Low", "Close", "Volume",

"Close Time", "Quote Asset Volume", "Number of Trades",

"Taker Buy Base Asset Volume", "Taker Buy Quote Asset Volume", "Ignore"])

# Convert Open Time to readable format (UTC)

df["Open Time"] = pd.to\_datetime(df["Open Time"], unit='ms')

# Convert UTC to Gulf Time (UTC+4)

df["Open Time"] = df["Open Time"] + timedelta(hours=4)

# Select relevant columns

df = df[["Open Time", "Close"]].rename(columns={"Open Time": "Timestamp", "Close": "XRP Price"})

# Filter data to only include rows between the specified dates (optional, if additional filtering is needed)

df\_filtered = df[(df["Timestamp"].dt.date >= pd.to\_datetime(from\_date).date()) &

(df["Timestamp"].dt.date <= pd.to\_datetime(to\_date).date())]

# Print the filtered data

print(df\_filtered)

# Save to CSV (optional)

file\_path = f"E:\\Data\_Practice\\xrp\_rate\_custom\_{from\_date}\_to\_{to\_date}.csv"

df\_filtered.to\_csv(file\_path, index=False)

print(f"Data saved to {file\_path}")

# Example usage

fetch\_historical\_xrp\_data(from\_date="2024-01-01", to\_date="2024-10-21")

# Example usage

fetch\_historical\_xrp\_data(from\_date="2024-10-18", to\_date="2024-10-20")

**Unit Test:**

Predition val

A screen shot of a computer

Description automatically generated

Actual Out come

A screenshot of a computer

Description automatically generated

Before

A screenshot of a computer

Description automatically generated

After

A screenshot of a computer

Description automatically generated