Build a Python Application for your Bitcoin Data

Generate Bitcoin statements for in-depth reports on your address current holdings and previous transactions, including price at time of transaction.

	Sent Amount	Block time	Transaction Hash	BTC/USD
ne	Address statement	Received Amount	Server Time	Server Time
d Amount	Transaction Hash	Sent Amount	Address statement	Address statement
ount	Block Time	BTC/USD	Received Amount	Transaction Hash
	Received Amount	Address statement	BTC/USD	Block Time

Public Practice Edition Volume II

Developed By:

Norma Escobar

Bitcoin Quantitative Developer

Table of Contents

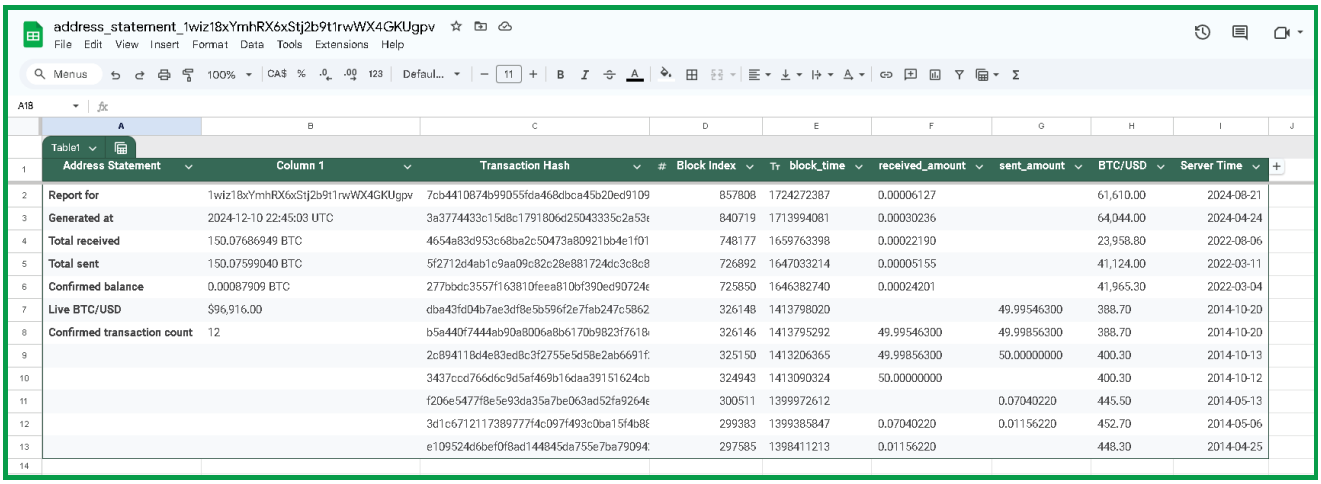
Application Overview

A python application built for your Bitcoin data will transform your work in impactful ways:

- **Generate in-depth address reports in .xlsx** format using Python and Mempool API.
- Learn how to **structure an API request** and transform the response into a human-readable table.
- **Automate** gathering your Bitcoin purchase history for financial and tax planning purposes.
- **Expand your career** opportunities and clientele by truly understanding the Bitcoin network.

Who is this application for

- **Bitcoiners** looking to build a successful computer program to manage their historical Bitcoin purchases.
- **Developers** and post-graduate students looking for development practices and tools to work in the Bitcoin industry.
- **Finance and accounting professionals** looking to expand their skill set to work with clients invested in Bitcoin.



Address Statement	Column 1	Transaction Hash	# Block Index	Tr block_time	received_amount	sent_amount	BTC/USD	Server Time
Report for	1wiz18xYmhRX6xStj2b9t1rwWX4GKUgpv	7cb4410874b99055fda468dbca45b20ed9109	857808	1724272387	0.00006127		61,610.00	2024-08-21
Generated at	2024-12-10 22:45:03 UTC	3a3774433c15d8c1791806d25043335c2a53e	840719	1713994081	0.00030236		64,044.00	2024-04-24
Total received	150.07686949 BTC	4654a83d953c68ba2c50473a80921bb4e1f01	748177	1659763398	0.00022190		23,958.80	2022-08-06
Total sent	150.07599040 BTC	5f2712d4ab1c9aa09c82c28e881724dc3c8c8	726892	1647033214	0.00005155		41,124.00	2022-03-11
Confirmed balance	0.00087909 BTC	277bbdc3557f163810feea810bf390ed90724e	725850	1646382740	0.00024201		41,965.30	2022-03-04
Live BTC/USD	$96,916.00	dba43fd04b7ae3df8e5b596f2e7fab247c5862	326148	1413798020		49.99546300	388.70	2014-10-20
Confirmed transaction count	12	b5a440f7444ab90a8006a8b6170b9823f7618	326146	1413795292	49.99546300	49.99856300	388.70	2014-10-20
		2c894118d4e83ed8c3f2755e5d58e2ab6691f	325150	1413206365	49.99856300	50.00000000	400.30	2014-10-13
		3437ccd766d6c9d5af469b16daa39151624cb	324943	1413090324	50.00000000		400.30	2014-10-12
		f206e5477f8e5e93da35a7be063ad52fa9264e	300511	1399972612		0.07040220	445.50	2014-05-13
		3d1c67121173897777f4c097f493c0ba15f4b8e	299383	1399385847	0.07040220	0.01156220	452.70	2014-05-06
		e109524d6bef0f8ad144845da755e7ba79094	297585	1398411213	0.01156220		448.30	2014-04-25

Abstract. Technical resources for accounting for Bitcoin are limited. Existing resources tend to focus on digital assets other than Bitcoin. Guidance from the traditional financial sector often emphasizes assertions related to existence and materiality, utilizing tools such as block explorers.

This guide introduces an application framework designed for the technical processes involved in Bitcoin accounting. The application is tailored exclusively for the Bitcoin mainnet (by design and preference). It verifies the existence and completeness of Bitcoin transaction data on the blockchain. The application is accessible to anyone globally and generates a Bitcoin address "statement," saving the report as an Excel workbook containing historical transaction data, including the price at the time of each transaction. The application is open-source, with the code fully auditable.

Introduction

Bitcoiners, developers, and financial professionals rely on block explorers to verify transactions. This framework utilizes Mempool, a comprehensive mempool visualizer, explorer, and API service hosted at mempool.space [1]. It offers a user-friendly dashboard for interacting with the Bitcoin ecosystem. However, relying solely on the dashboard presents limitations, such as the absence of automation, inefficient bulk data copying, and minimal file export capabilities. The API service is better suited for addresses with larger transaction volumes.

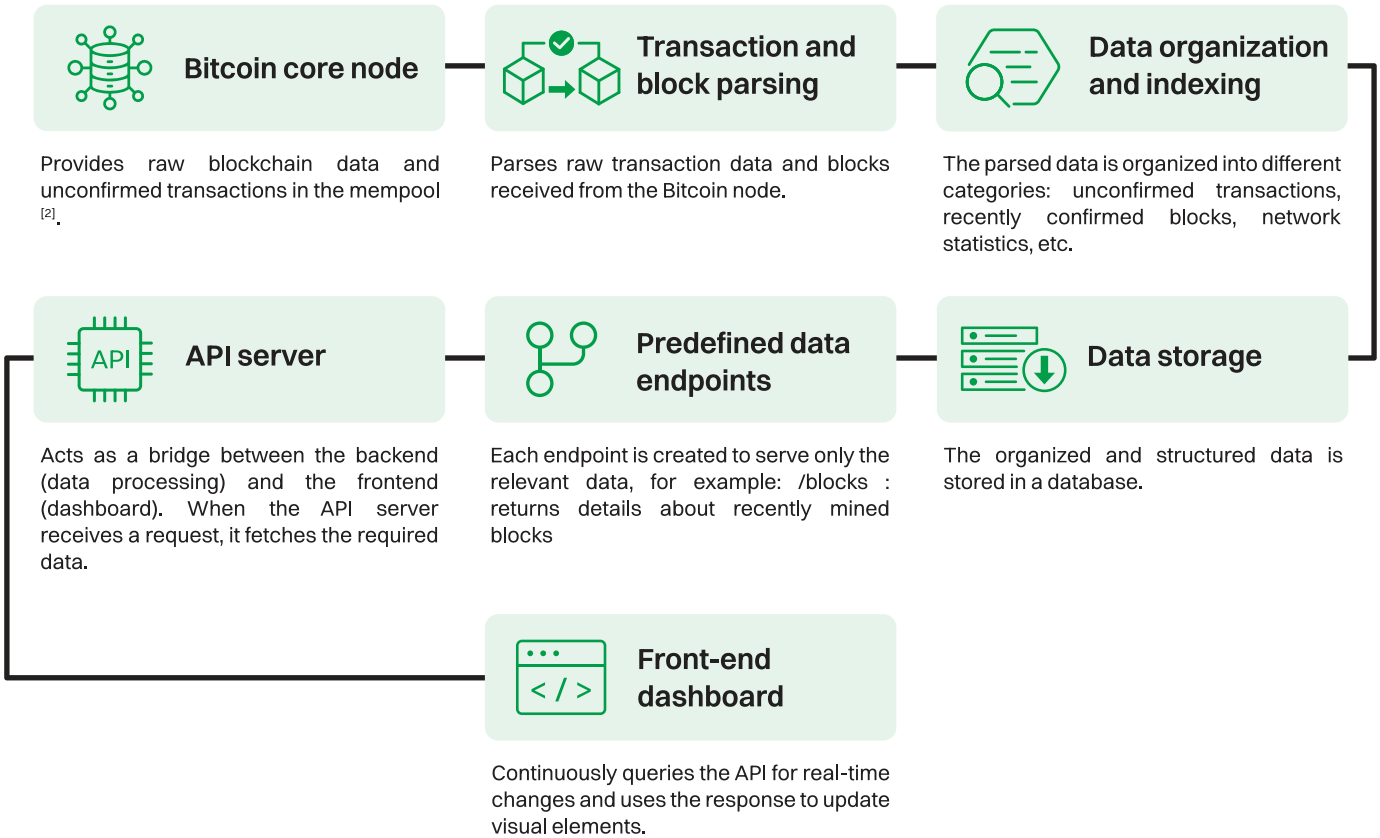


Diagram 1.1 - Breaking down the architecture of mempool.space.

[1] - Learn more about Mempool at https://github.com/mempool/mempool
[2] - mempool (lowercase) - short for "memory pool", temporary storage for unconfirmed transactions. Each node has its own mempool. Mempool (uppercase) is the open-source project.

Mempool API Server

It formats data into JSON responses. Here is a sample URL request and its corresponding response:

Sample URL to make server request:

`https://mempool.space/api/tx/15e10745f15593a899cef391191bdd3d7c12412cc4696b7bcb669d0feadc8521`

`/api`: serves programmatic access to the platform's data, as opposed to its user-friendly dashboard.

`/tx`: stands for "transaction" and specifies that the request relates to a Bitcoin transaction.

`/15e10745f15593a899cef391191bdd3d7c12412cc4696b7bcb669d0feadc8521`: this is the transaction ID and it is the unique identifier of a Bitcoin transaction.

Sample response[3] in JSON:


```
{
  "txid": "15e10745f15593a899cef391191bdd3d7c12412cc4696b7bcb669d0feadc8521",
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "1fdfed84588cb826b876cd761ecebcf1726453437f0a6826e82ed54b2807a036",
      "vout": 12,
      "prevout": {
        "scriptpubkey": "76a9142d110e1702a73c56fb6ea709cd529ea00680114388ac",
        "scriptpubkey_asm": "OP_DUP OP_HASH160 OP_PUSHBYTES_20 2d110e1702a73c56fb6ea709cd529ea006801143 OP_EQUALVERIFY OP_CHECKSIG",
        "scriptpubkey_type": "p2pkh",
        "scriptpubkey_address": "157HqdxdT8dxTjeRLVT5HPtFc1LH4CeuVC",
        "value": 686860000
      },
[cont...]
  ],
  "vout": [
    {
      "scriptpubkey": "76a91472d52e2f5b88174c35ee29844cce0d6d24b921ef88ac",
      "scriptpubkey_asm": "OP_DUP OP_HASH160 OP_PUSHBYTES_20 72d52e2f5b88174c35ee29844cce0d6d24b921ef OP_EQUALVERIFY OP_CHECKSIG",
      "scriptpubkey_type": "p2pkh",
      "scriptpubkey_address": "1BUBQuPV3gEV7P2XLNuAJQjf5t265Yyj9t",
      "value": 1240000000
    },
    {
      "scriptpubkey": "76a914c15b731d0116ef8192f240d4397a8cdbce5fe8bc88ac",
      "scriptpubkey_asm": "OP_DUP OP_HASH160 OP_PUSHBYTES_20 c15b731d0116ef8192f240d4397a8cdbce5fe8bc OP_EQUALVERIFY OP_CHECKSIG",
      "scriptpubkey_type": "p2pkh",
      "scriptpubkey_address": "1JdNy4KCNVQ6ay8qsc52DW1TtS7ZCnvJ5W",
      "value": 782740000
    },
    {
      "scriptpubkey": "76a914c7ee32e6945d7de5a4541dd2580927128c11517488ac",
      "scriptpubkey_asm": "OP_DUP OP_HASH160 OP_PUSHBYTES_20 c7ee32e6945d7de5a4541dd2580927128c115174 OP_EQUALVERIFY OP_CHECKSIG",
```


[3] - Sample response is shortened

```
      "scriptpubkey_type": "p2pkh",
      "scriptpubkey_address": "1KE8pX7V7D8b4Cd5DL1jZwjy2vS5NtZpBT",
      "value": 1375350000
    },
    {
      "scriptpubkey": "76a9140a59837ccd4df25adc31cdad39be6a8d97557ed688ac",
      "scriptpubkey_asm": "OP_DUP OP_HASH160 OP_PUSHBYTES_20 0a59837ccd4df25adc31cdad39be6a8d97557ed6 OP_EQUALVERIFY OP_CHECKSIG",
      "scriptpubkey_type": "p2pkh",
      "scriptpubkey_address": "1wizSAYSbuyXbt9d8JV8ytm5acqq2TorC",
      "value": 2615350000
    }
  ],
  "size": 884,
  "weight": 3536,
  "sigops": 16,
  "fee": 20000,
  "status": {
    "confirmed": true,
    "block_height": 363348,
    "block_hash": "00000000000000000139385d7aa78ffb45469e0c715b8d6ea6cb2ffa98acc7171",
    "block_time": 1435754650
  }
}
```


The relevant keys are:

"txid": refers to the transaction ID of the main object.
"vin": stands for value inputs, and it is an array containing the inputs of the transaction.
"prevout": located inside the vin array because it represents the previous output of another transaction that is being used as an input in the current transaction. This is necessary to prove that the funds being spent exist.
"scriptpubkey_address": this is the address derived from the `scriptpublickey`. Every transaction input and output specify an address.
"vout": stands for value outputs, and this array contains the outputs where Bitcoin is being sent.
"status" :contains the keys "`block_height`" : and "`block_time`" : with the block index and timestamp (UNIX) for when the transaction was confirmed.

Important Information

If a Bitcoin address matches any `scriptpubkey_address` within the `vin.prevout` array, that address is being utilized to send Bitcoin. ➡

If a Bitcoin address matches any `scriptpubkey_address` within the `vout` array, that address is being used to receive Bitcoin. ⬅

Why can we trust the data from Mempool?
It is sourced from a Bitcoin node, which maintain a decentralized and public ledger. These nodes verify transaction validity through consensus rules. The decentralized nature of the Bitcoin blockchain ensures that no single entity controls or manipulates the data.

Mempool API Terminal

Proof-of-concept application, written in Python, to generate Bitcoin “statements”. This application can be downloaded by anyone in the world and used under MIT License. It generates an Excel workbook report with the historical transaction data and price[4] at time of transaction.

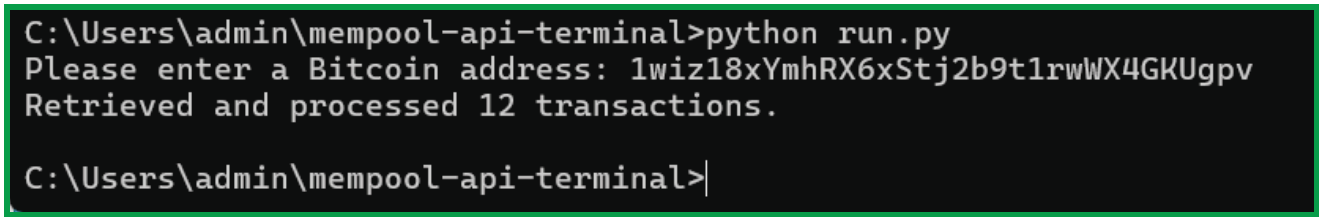


Image 2.1 - Executing Mempool API Terminal locally.

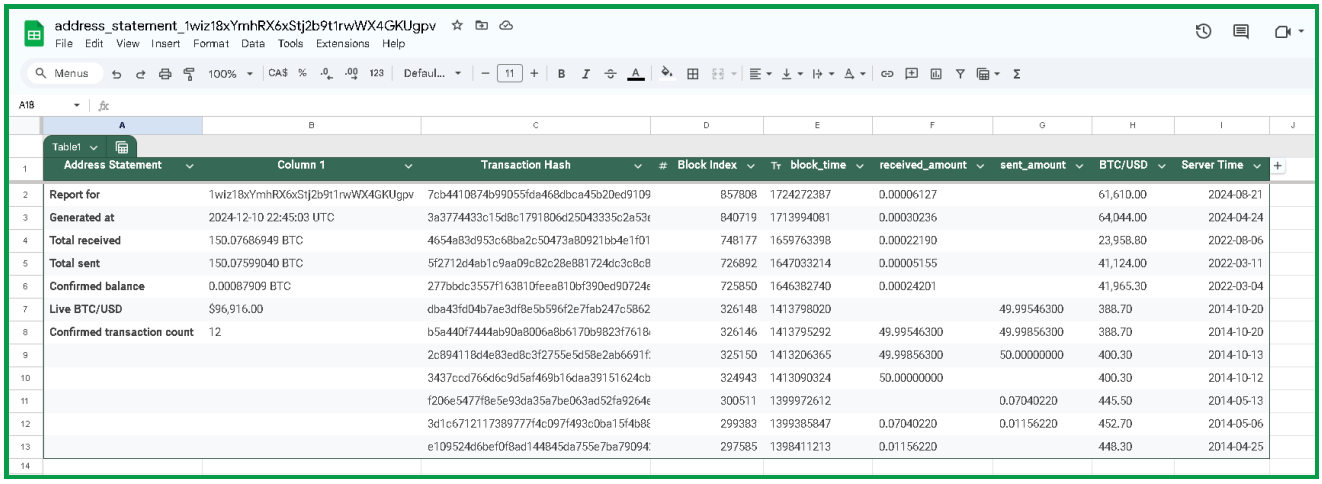


Image 2.2 - Generated address statement.

Key Features:

- **Seamless download and code base under 150 lines:** optimized for simplicity, with a compact codebase that ensures quick setup and minimal maintenance.
- **Built-in functions to fetch the data from mempool.space:** with a set of pre-configured functions, directly retrieve transactions and price information from the Mempool API server.
- **Transformation of request response into a manageable dataframe:** data is automatically parsed and converted into a structured dataframe, making it easy to analyze and export.
- **Automation and organized Bitcoin data:** automates data collection, cleaning and categorization, providing a well-organized report.

[4] - Price values are denominated in USD

Application Architecture:

Mempool API Terminal consists of 4 modules:

- **Data:** The backbone of communication between the API server and terminal, providing:
  - Source code: functions to fetch transaction data, price at time of transaction and the current price of Bitcoin denominated in USD.
  - Infrastructure: built-in API calls to Mempool, ensuring seamless data retrieval for both transaction details and prices.

- **Processing:** The engine that powers data analysis, featuring:
  - Data transformation: converts raw response data into structured insights, including parsing transactions to extract key information.
  - Price correlation: Integrates price data per transaction and real-time Bitcoin market value.

- **Dataframe:** The visualizer of results, offering:
  - Data presentation: transforms processed data into a structured dataframe, summarizing key metrics.
  - Excel Workbook: formats and saves the dataframe into a user-friendly .xlsx file for easy reporting and analysis.

- **Run:** The module responsible for executing the application, handling:
  - User interaction: captures a Bitcoin address from the user and coordinates data fetching, processing and reporting.
  - Output automation: provides the user with an automatically generated statement.

Main Application Components

Functions for fetching the raw data from Mempool API:

```
def getTransactionData(address):
    staticUrl = f"https://mempool.space/api/address/{address}/txs"
    rawData, seenTransactions = [], set()
    after_txid = None
    try:
        while True:
            queryUrl = requests.get(f"{staticUrl}?after_txid={after_txid}" if after_txid else staticUrl)
            if queryUrl.status_code != 200:
                return None
            paginatedTransactions = queryUrl.json()
            if not paginatedTransactions:
                break
            for tx in paginatedTransactions:
                if tx["txid"] not in seenTransactions:
                    seenTransactions.add(tx["txid"])
                    rawData.append(tx)
            after_txid = paginatedTransactions[-1]["txid"] if paginatedTransactions else None
        return rawData
    except requests.RequestException:
        return None
```

`getTransactionData` fetches all transaction data for a specified Bitcoin address using pagination. It repeatedly makes requests, appending each page of transactions to a list (rawData), and continues until all available data has been retrieved. `after_txid` parameter assumes perfect continuity and the API does not guarantee strict ordering. `seenTransactions` list records unique transactions. If there are any issues during the request process, the function returns None.

```
def getPriceLog(block_times, currency="USD"):
    priceLog = []
    staticUrl = "https://mempool.space/api/v1/historical-price"
    for block_time in block_times:
        try:
            queryUrl = requests.get(staticUrl, params={"currency": currency, "timestamp": block_time})
            price = queryUrl.json().get("prices", [{}])[0].get(currency)
            if price is not None:
                priceLog.append({"timestamp": block_time, f"BTC/{currency}": price})
        except requests.RequestException:
            continue
    return priceLog
```

`getPriceLog` retrieves historical Bitcoin prices for a list of provided block times. It sends requests for each block time, parses the returned data for the price in the specified currency, and stores it in a `priceLog` list. If a price is found, it adds the timestamp and price to the list. The function continues retrieving prices even if one request fails, and returns a list of price data after processing all the block times.

```
def getCurrentPrice():
    try:
        return requests.get("https://mempool.space/api/v1/prices").json().get("USD")
    except requests.RequestException:
        return None
```

`getCurrentPrice` fetches the current price of Bitcoin. If the request is successful, it returns the price in USD. If there is an issue with the request, it returns None.

Function for processing the raw data:

```
def processing(rawData, userAddress):

    processedData = []
    for transaction in rawData:
        if "txid" in transaction and "status" in transaction:
            if "block_height" in transaction["status"] and "block_time" in transaction["status"]:
                txid = transaction["txid"]
                blockHeight = transaction["status"]["block_height"]
                blockTime = transaction["status"]["block_time"]
                inputs, outputs, sentAmount, receivedAmount = [], [], [], []
```

It iterates through `rawData`, checking if each transaction contains keys for "txid" and "status", and further ensures "block_height" and "block_time" are present in the "status" dictionary. Within this context, it prepares to extract transaction details such as `txid`, `block_height`, and `block_time`, and initializes empty lists for input and output addresses, as well as sent and received amounts.

```
for vin in transaction.get("vin", []):
                    if "prevout" in vin and "scriptpubkey_address" in vin["prevout"]:
                        address = vin["prevout"]["scriptpubkey_address"]
                        inputs.append(address)

                        if address == userAddress and "value" in vin["prevout"]:
                            sentAmount.append(vin["prevout"]["value"])

                for vout in transaction.get("vout", []):
                    if "scriptpubkey_address" in vout:
                        address = vout["scriptpubkey_address"]
                        outputs.append(address)

                        if address == userAddress and "value" in vout:
                            receivedAmount.append(vout["value"])
```

For each input, it checks for a scriptpubkey_address in prevout and appends it to inputs, while also recording the value sent by `userAddress`. Similarly, for each output, it checks for a `scriptpubkey_address` and appends it to outputs, recording the value received by `userAddress`. This effectively tracks addresses and amounts associated with the user's transactions.

```
                priceData = getPriceLog([blockTime])
                                btcUsd = priceData[0].get("BTC/USD", "N/A") if priceData else
                "N/A"
```

Extracts the BTC/USD price from the first element of the priceData list, defaulting to "N/A" if the list is empty. The result is stored in the variable `btcUsd`.

```
                processedData.append({
                          "Transaction Hash": txid,
                          "Block Index": blockHeight,
                          "Block Time": blockTime,
                          "receivedAmount": sum(receivedAmount) / 1e8 if receivedAmount else None,
                          "sentAmount": sum(sentAmount) / 1e8 if sentAmount else None,
                          "BTC/USD": btcUsd
                      })

        return processedData
```

Appends a dictionary to processedData for each transaction, summarizing key details: Transaction Hash, Block Index, Block Time, received and sent amounts (denominated in Bitcoin instead of sats), and the BTC/USD price. If no values are available for receivedAmount or sentAmount, it returns None. Finally, the function returns the processedData list containing all processed transaction details.

Function for creating the data frame out of the processed data:

```
def createDataframe(processedData, currentPrice, userAddress):

    totalReceived = sum(
        float(tx.get("receivedAmount") or 0.0)
        for tx in processedData
    )

    totalSent = sum(
        float(tx.get("sentAmount") or 0.0)
        for tx in processedData
    )

    confirmedBalance = totalReceived - totalSent

    timestamp = datetime.now(timezone.utc).strftime("%Y-%m-%d %H:%M:%S UTC")
```

Calculates final balances and prepares data for conversion into a Dataframe. It computes totalReceived and totalSent by summing transaction amounts from processedData. The confirmedBalance is derived by subtracting totalSent from totalReceived, while the timestamp records the current UTC time.

```
summaryData = {
        "Address Statement": [
            "Report for",
            "Generated at",
            "Total received (BTC)",
            "Total sent (BTC)",
            "Confirmed balance (BTC)",
            "Live BTC/USD",
            "Confirmed transaction count"
        ],
        "": [
            userAddress,
            timestamp,
            totalReceived,
            totalSent,
            confirmedBalance,
            currentPrice,
            len(processedData)
        ]
    }
    summaryDataframe = pd.DataFrame(summaryData)
    transactionsDataframe = pd.DataFrame(processedData)

    return pd.concat([summaryDataframe, transactionsDataframe], axis=1).fillna("")
```

Creates two dataframes: summaryDataframe for a high-level summary of the user's transactions and transactionsDataframe for detailed transaction history. The summaryData dictionary stores labels and corresponding values like the user's address, timestamp, total amounts, confirmed balance, live BTC/USD price, and transaction count. The function concatenates both dataframes side-by-side, and returns the combined dataframe.

Function to save data frames as Excel workbook:

```
def excelReport(dataframe, userAddress, filenamePrefix="address_statement"):
    dataframe.to_excel(f"{filenamePrefix}_{userAddress}.xlsx", index=False)
```

Saves dataframe to an .xlsx file. The filename is constructed using the filenamePrefix (defaulting to "address_statement") and the userAddress, creating a file named like address_statement_<userAddress>.xlsx. The index=False parameter excludes DataFrame indices from the saved file.

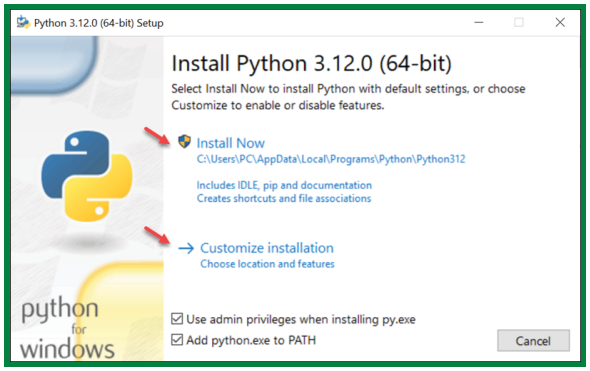
Installing the Application

Setting up the environment

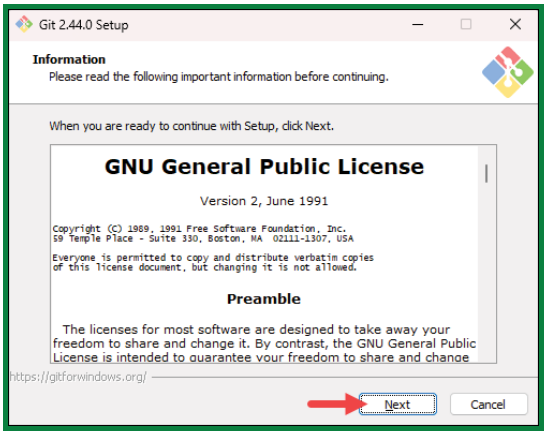
Downloading programming languages:

A python application built for your Bitcoin data will transform your work in impactful ways:

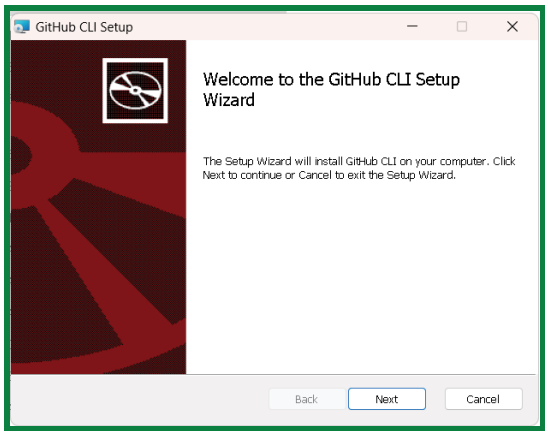
01 Install Python
(https://www.python.org/downloads/)



02 Install Git
(https://git-scm.com/downloads)



03 Install Github CLI
(https://cli.github.com/)



Checking Python version install:

- Open your terminal and type the command `python --version`

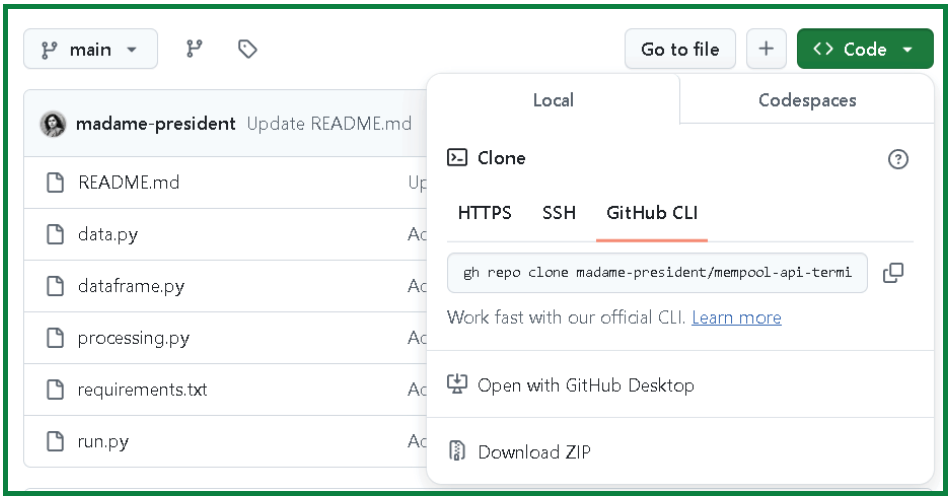
```
C:\Users\PC>python --version
Python 3.12.0
```

- For Git, type the command `git --version`

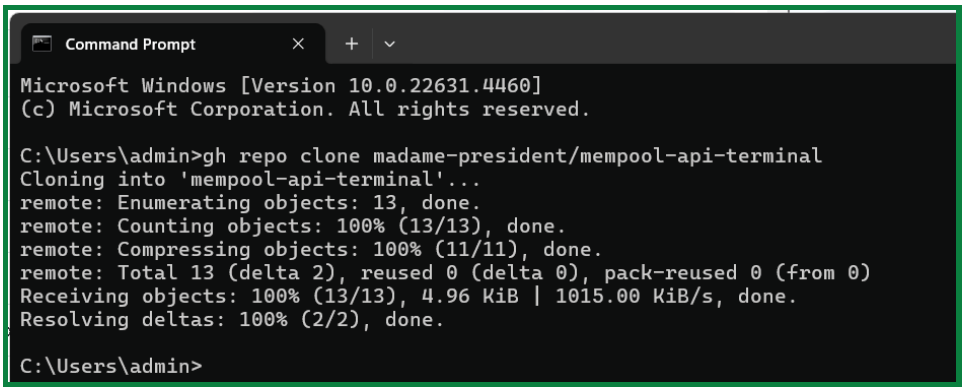
```
PS C:\Windows\system32> git --version
git version 2.44.0.windows.1
PS C:\Windows\system32>
```

Cloning the repository

01 Go to this website: (https://github.com/madame-president/mempool-api-terminal). Under Code > Github CLI > copy the command: `gh repo clone madame-president/mempool-api-terminal`

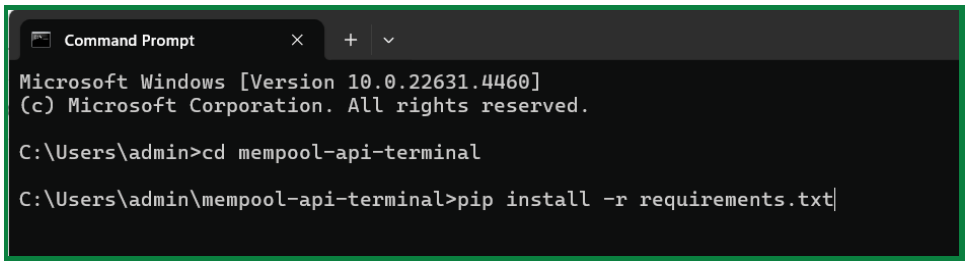


02 Open your terminal and paste the command:

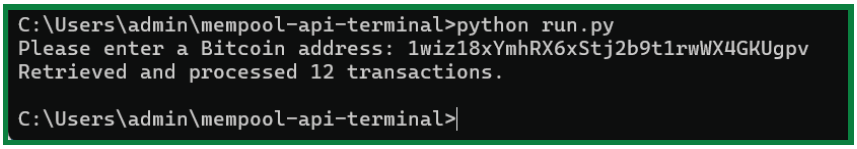


03 Go to the repository directory. Type `cd mempool-api-terminal`

04 Inside the directory, install dependencies typing the command `pip install -r requirements.txt`



05 To run the program, type `python run.py` and enter your Bitcoin address when prompted. The application will store your .xlsx report in the same folder.



Author's Note

I am very passionate about building applications for Bitcoin data. Better data leads to better decisions. I am a Bloomberg Terminal enthusiast and believe that a similar application is highly needed in the Bitcoin industry. Resources on programming Bitcoin are scarce, and even more so are those specifically focused on Bitcoin data visualization tailored to the public practice industry.

I created this guide with financial professionals and accountants in mind. Not all Bitcoin developers ought to create the next wallet or exchange — some of us have a role on the other side. Though perhaps less glamorous, it is equally essential to provide our financial wizards with the tools needed to enhance assurance, audit efficiency and financial modelling.

Professionally, I spend most of my time working at Kingston, Ross and Pasnak LLP. I work for Justin Rousseau, the partner leading all bitcoin-related audits. I may be biased, but I think we make a great team. On a personal note, when I am not working, I spend my time at the gym, playing soccer and writing research articles related to things I find interesting.

Lastly, I would like to give thanks to my parents, Francisco & Norma Sr. Thank you for giving me everything, so that I could pursue my dreams fearlessly. Thank you to my brother and sister, who have to constantly hear me talk about Bitcoin.

You can follow my work at www.normaescobar.com or contact me directly via e-mail: norma@normaescobar.com.