RATINGS - WEB SCRAPING RELAZIONE

Lo scopo del progetto è l'implementazione di un insieme di tecniche di deanonimizzazione e di analisi dei miners della blockchain di Bitcoin. E' necessario lavorare sia su un DataSet (`/data/raw`) contenente un sottoinsieme delle transazioni di Bitcoin, che sul sito WalletExplorer, per la parte di scraping. Vengono richieste un insieme di analisi generali sulle transazioni contenute nel DataSet di Bitcoin, quindi un insieme di analisi che sfruttano dati ottenuti mediante scraping.

Struttura del progetto

Il progetto è strutturato su 2 cartelle principali:

- `/data`: contiene a sua volta 2 cartelle:
  - `/processed`: Contiene dei file `CSV` derivanti dallo scraping del sito "WalletExplorer.com". I files sono creati con una funzione che, prendendo uno per uno i dati richiesti per le analisi future, li riscrive in questo formato per poterci accedere più velocemente su run successive.
  - `/raw`: Contiene i 4 files `CSV` del Dataset allegato alla consegna. Su questi files vengono eseguite della analisi preliminari contenute nel file `test_dimensioni.py` dove, in breve, trovo la dimensione massima dei dati per fare un cast a un tipo inferiore risparmiando memoria.
- `/scripts/utils`: dove al suo interno troviamo 3 files:
  - `data.py`: Contiene tutte le funzioni che serviranno a eseguire le analisi sui dati forniti
  - `graph.py`: Contiene tutte le funzioni che servono a "plottare" i grafici
  - `scraping.py`: Contiene tutte le funzioni che servono a fare scraping del sito "WalletExplorer.com"

Main.py

Importazioni e Setup

Librerie Importate:

- pandas per la manipolazione dei dati,
- os per la gestione dei file e directory,
- UserAgent per generare user agent casuali durante lo scraping.
- Importa anche le funzioni personalizzate definite nei moduli utils/data, utils/scraping, e utils/graph.

Caricamento e Verifica dei Dati

Controllo della Directory dei Dati:

- Verifica se la directory /`data/raw/` esiste e se contiene file. In caso contrario, il programma termina con un errore.
- Definisce i percorsi dei file e le intestazioni di colonne previste per ciascun file.
- Chiama `import_dataset()` per caricare i dati, e verifica che tutti i DataFrame siano caricati correttamente e che non siano vuoti.
- `Risultato`: I dati vengono caricati nei DataFrame `df_mapping`, `df_inputs`, `df_outputs`, e `df_transactions`.

Grafico del Rapporto di Congestione e Fee

- Filtra `df_transactions` per escludere le transazioni `Coinbase`.
- Calcola la dimensione di ciascuna transazione sulla base dei campi di `input`, `output` e `script_size`, dove `script_size` varia in base al tipo di script.
- Raggruppa le transazioni per mese (`[month]`), calcolando la congestione e la somma delle fee per ogni mese.
- Calcola il rapporto congestion_fee.
- Chiama `plot_congestion_fee()` per visualizzare un grafico della congestione mensile e del rapporto fee/congestion.

Distribuzione dei Tipi di Script

- Unisce `df_outputs` e `df_transactions` per ottenere i campi `[timestamp]` e `[scripttype]` per ogni transazione.
- Raggruppa i dati mensilmente e per tipo di script, calcolando la quantità di ciascun tipo per mese.
- Chiama `plot_script_counts()` per visualizzare la distribuzione mensile di ciascun tipo di script.

Scraping dei Dati di WalletExplorer

- Crea una lista di `proxies` e inizializza `UserAgent` per evitare blocchi durante lo scraping.
- Verifica se i file CSV per i mining pool `DeepBit.net, Eligius.st, BTCGuild.com`, e `BitMinter.com` esistono e non sono vuoti.
- Se non esistono, chiama `scrape_wallet_explorer()` per ottenere e salvare gli indirizzi dei wallet.
- Carica gli indirizzi di ciascun pool in un dizionario `mining_pools_addresses` per un utilizzo successivo nella fase di Deanonimizzazione.

Deanonimizzazione delle Transazioni Coinbase

- Seleziona solo le transazioni `Coinbase`, rimuovendo i campi inutili.
- Aggiunge la colonna `mining_pool` usando `mining_pools_addresses` per associare ogni indirizzo al nome del pool minerario corrispondente (se presente).
- `Risultato`: Un DataFrame `tx_coinbase` contenente le transazioni `Coinbase` con identificazione del pool.

Identificazione dei Principali Miner Anonimi

- Analizza gli hash dei miner che non appartengono ai pool conosciuti (etichettati come `Others`).
- Ordina e seleziona i primi 4 miner più attivi tra quelli non etichettati, salvandoli in `top_4_miners`.

- Recupera e mappa questi miner nei rispettivi `wallet_id` utilizzando `found_miners()`.

Analisi dei Blocchi Minati

- Chiama `calculate_blocks_mined()` per calcolare il numero di blocchi minati dai pool noti e dai top 4 miner.
- I blocchi vengono calcolati a intervalli bimestrali per entrambi i gruppi e vengono anche calcolati i totali globali.
- Chiama `plot_blocks_mined()` per visualizzare i blocchi minati dai pool e dai top 4 miner.

Analisi delle Ricompense Ricevute

- Chiama `calculate_rewards()` per calcolare il numero di ricompense ricevute dai pool noti e dai top 4 miner.
- I blocchi vengono calcolati a intervalli bimestrali per entrambi i gruppi e vengono anche calcolati i totali globali.
- Chiama `plot_rewards()` per visualizzare le ricompense per ogni entità.

Analisi del percorso della transazione coinbase di Eligius

- Imposta il `base_url` e il `first_tx` per iniziare l'analisi del percorso.
- Costruisce un grafo delle transazioni chiamando `build_transaction_graph()`, che inizia con `first_tx` e segue gli output fino alla profondità `k`.
- Chiama `visualize_graph()` per mostrare una rappresentazione grafica delle transazioni e dei collegamenti di `input/output`.

Utils

Data.py

`import_dataset(files_path, dataframes, names):`

- Carica i file `CSV` indicati in `files_path` come DataFrame, assegnando le colonne specificate in `names` e i tipi di dati specificati in `dtype`. Ogni DataFrame viene aggiunto alla lista `dataframes`.

`df_column_bimonthly_period(df):`

- Crea una colonna `[bimonthly_period]` nel DataFrame `df`, raggruppando le date a intervalli bimestrali e mappando ciascun intervallo alla data di inizio in formato `YYYY-MM-DD`.

`calculate_blocks_mined(data, group_by_col, entity_col, value_col="block_id"):`

- Calcola il numero di blocchi minati per ogni entità e periodo, raggruppando per `group_by_col` e `entity_col`. Restituisce un DataFrame con i blocchi per ogni entità e un Series con i totali complessivi per ogni entità.

`calculate_rewards(data, group_by_col, entity_col, value_col="amount"):`

- Calcola le ricompense (`amount`) per ogni entità e periodo, raggruppando in base ai parametri forniti. Restituisce un DataFrame con le ricompense periodiche per ciascuna entità e un Series con le ricompense totali per ogni entità.

Graph.py

Funzioni per la Visualizzazione dei Dati

`plot_script_counts(script_counts):`

- Plotta la distribuzione mensile del conteggio degli script per ciascun tipo di script. Converte l'indice a una frequenza temporale mensile e disegna una linea per ogni tipo di script, con asse X in mesi, asse Y come conteggio degli script, e include una legenda per identificare i tipi di script.

`plot_congestion_fee(df_congestion_fee):`

- Visualizza sia la congestione mensile che il rapporto `fee/congestion` per ogni mese. Usa un grafico con due assi Y: uno per la congestione (linea blu tratteggiata) e uno per il rapporto `fee/congestion` (linea arancione punteggiata), con asse X nei mesi.

`plot_blocks_mined(blocks_entity, total_blocks_entity):`

- Plotta il numero di blocchi minati per ciascun periodo (grafico a barre) e il totale per ogni entità. Disegna i dati per ciascun periodo su un asse, con un secondo asse dedicato al totale dei blocchi minati, specificato per ogni entità.

`plot_rewards(rewards_entity, total_rewards_entity):`

- Funzione analoga alla precedente ma con `Titolo` e `Labels` diversi

Funzioni per la Costruzione e Visualizzazione del Grafo delle Transazioni

`build_transaction_graph(G, first_tx, k, base_url, proxies, ua):`

- La funzione `build_transaction_graph` costruisce un grafo diretto a partire da una transazione iniziale `first_tx`, esplorando le transazioni correlate fino a una profondità massima `k`. Utilizzando una lista `to_visit` per mantenere traccia delle transazioni da esplorare, la funzione visita iterativamente ciascuna transazione, recuperando i dati di input, output e transazioni successive attraverso `get_hash_and_transaction`.
- Ogni transazione è aggiunta come nodo nel grafo `G`, con etichette `inputs` e `outputs` che rappresentano gli indirizzi hash relativi. Per ciascun output della transazione corrente, viene stabilito un arco diretto verso una transazione successiva, con un'etichetta `output_used` che rappresenta l'output specifico usato come input nella transazione

successiva e un identificatore di `transaction_id` per tracciare il numero di transazione. Il processo continua fino al raggiungimento della profondità massima specificata o al completamento dell'esplorazione di tutte le transazioni.

`visualize_graph(G):`

- Visualizza il grafo delle transazioni costruito in `build_transaction_graph(...)`, mostrando i nodi colorati a seconda che siano solo output o siano entrambi (input e output). Etichetta ogni nodo con i suoi input e output abbreviati e aggiunge sulle etichette degli archi il numero di transazione e l'output usato, posizionato al centro degli archi per una rappresentazione chiara.

Scraping.py

`scrape_wallet_explorer(mining_pools, proxies, ua, base_url, output_dir):`

- Coordina l'intero processo di scraping, creando prima una struttura per memorizzare i link delle pool in `mining_pools_addresses`. Successivamente, estrae i link e gli indirizzi wallet delle pagine dei pool e li salva come file `CSV` nell'output directory specificata.

`generate_proxies(proxies, ua):`

- Questa funzione raccoglie una lista di proxy gratuiti dal sito `sslproxies.org` e li memorizza nella lista `proxies`. Utilizzando un `user-agent` casuale, maschera la richiesta per emulare il comportamento di un utente reale e seleziona una tabella HTML contenente i dettagli dei proxy, salvando ciascuno come un dizionario con `ip` e `port`.

`get_page_with_proxy(url, proxies, ua):`

- La funzione accede a una pagina web tramite un proxy casuale dalla lista proxies, tentando di evitare blocchi da parte del

server. A ogni tentativo fallito, elimina il proxy non funzionante e riprova con un altro, fino a trovare un proxy attivo. Se nessun proxy è disponibile, interrompe l'operazione. Questa funzione garantisce il rispetto delle politiche del server grazie all'uso di attese e di user-agent casuali.

`check_next_page(soup):`

- Questa funzione verifica se c'è una pagina successiva nella paginazione del sito WalletExplorer. Utilizza `BeautifulSoup` per individuare link con etichette "Next…" e "Last". Restituisce il `link` della pagina successiva se presente, altrimenti restituisce `None`.

`get_address_page_link(mining_pools_addresses, base_url, proxies, ua):`

- Naviga alla pagina di ciascun mining pool per ottenere il link alla pagina che mostra gli indirizzi wallet. Prende il primo link memorizzato in `mining_pools_addresses` e aggiorna il dizionario con il link alla pagina con i dettagli degli indirizzi.

`extract_mining_pool_addresses(single_mining_pool_addresses, mining_pools_addresses,base_url, proxies, ua):`

- Questa funzione scorre le pagine di ciascuna pool per estrarre tutti gli indirizzi wallet presenti. Per ogni indirizzo trovato all'interno della tabella, lo aggiunge alla lista di indirizzi `single_mining_pool_addresses` del pool corrispondente. Controlla inoltre se esistono pagine successive utilizzando la funzione `check_next_page()`.

`get_mining_pool_page_link(mining_pools, mining_pools_addresses, base_url, proxies, ua):`

- Accede alla pagina principale di WalletExplorer per trovare i link relativi ai mining pools specificati. Aggiunge i link

delle pagine iniziali di ciascun pool al dizionario `mining_pools_addresses`.

`check_csv_files(output_dir, mining_pools):`

- Questa funzione verifica che i file CSV contenenti gli indirizzi dei mining pool siano presenti e non vuoti. Se la directory `output_dir` non esiste, oppure se uno dei file `CSV` corrispondenti a ciascun mining pool è assente o vuoto, la funzione restituisce `False`; altrimenti, restituisce `True`.

`found_miners(top_4_miners, base_url, wallet_id):`

- Esegue una ricerca su WalletExplorer per i wallet ID dei miner principali. Utilizza `Selenium` per automatizzare l'accesso e la ricerca degli hash specificati nella pagina principale del sito e, tramite una ricerca testuale, memorizza i wallet ID associati agli hash di ciascun miner nel dizionario `wallet_id`. Il browser Chrome viene eseguito in modalità headless per operare in background.

`get_hash_and_transaction(txid, base_url, proxies, ua):`

- Estrae gli hash delle transazioni (input e output) e i relativi `txid` da una pagina di WalletExplorer, richiedendo la pagina con un proxy casuale. Analizza la tabella HTML della transazione corrente per ottenere l'hash di input e di output, memorizzandoli rispettivamente nelle liste `input_hash` e `output_hash`, mentre gli ID delle transazioni vengono aggiunti a `output_txids`.

`get_hash(trs, list_hash):`

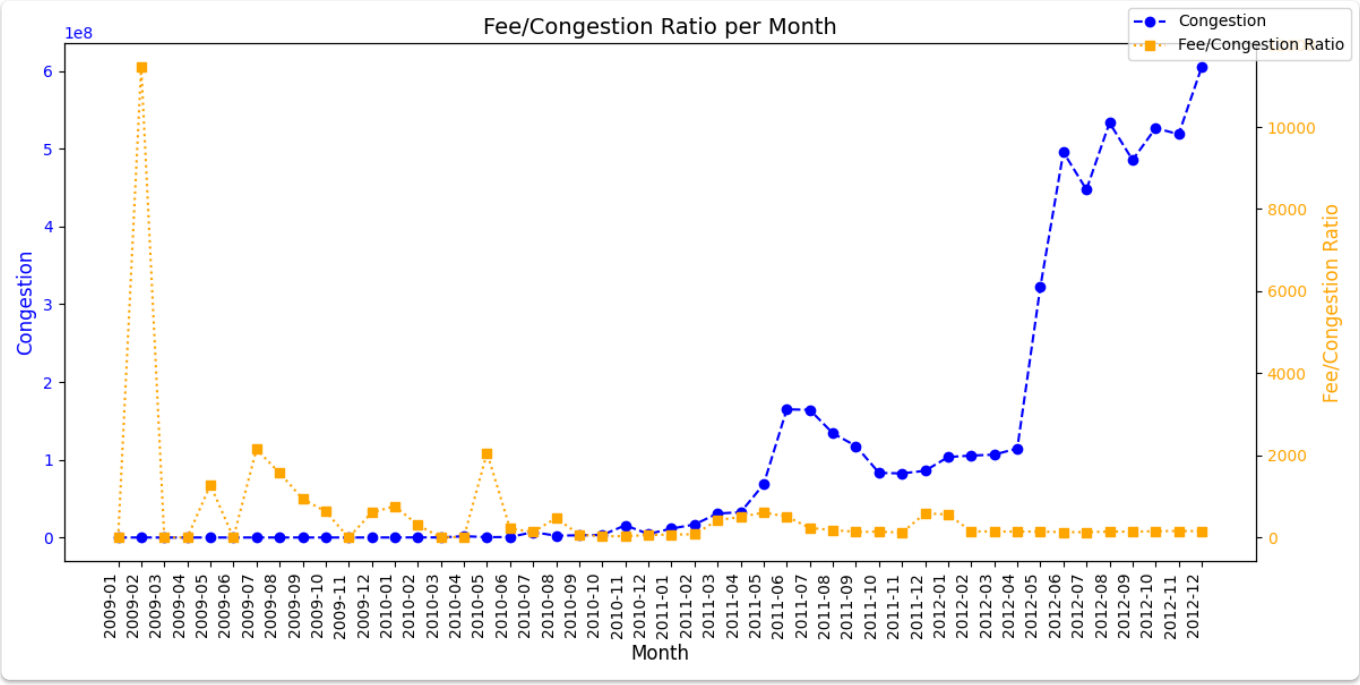
- Dato un elenco di righe di tabella, questa funzione estrae e aggiunge l'hash di ciascuna riga alla lista `list_hash`. Se l'hash non è presente, aggiunge "Coinbase" per identificare che si tratta di una transazione di quel tipo.

`get_output_txids(trs, output_txids):`

- Esamina le righe della tabella di output e aggiunge ciascun `txid` trovato nella lista `output_txids`. Viene selezionato specificamente il link associato a ciascun output.

GRAFICI

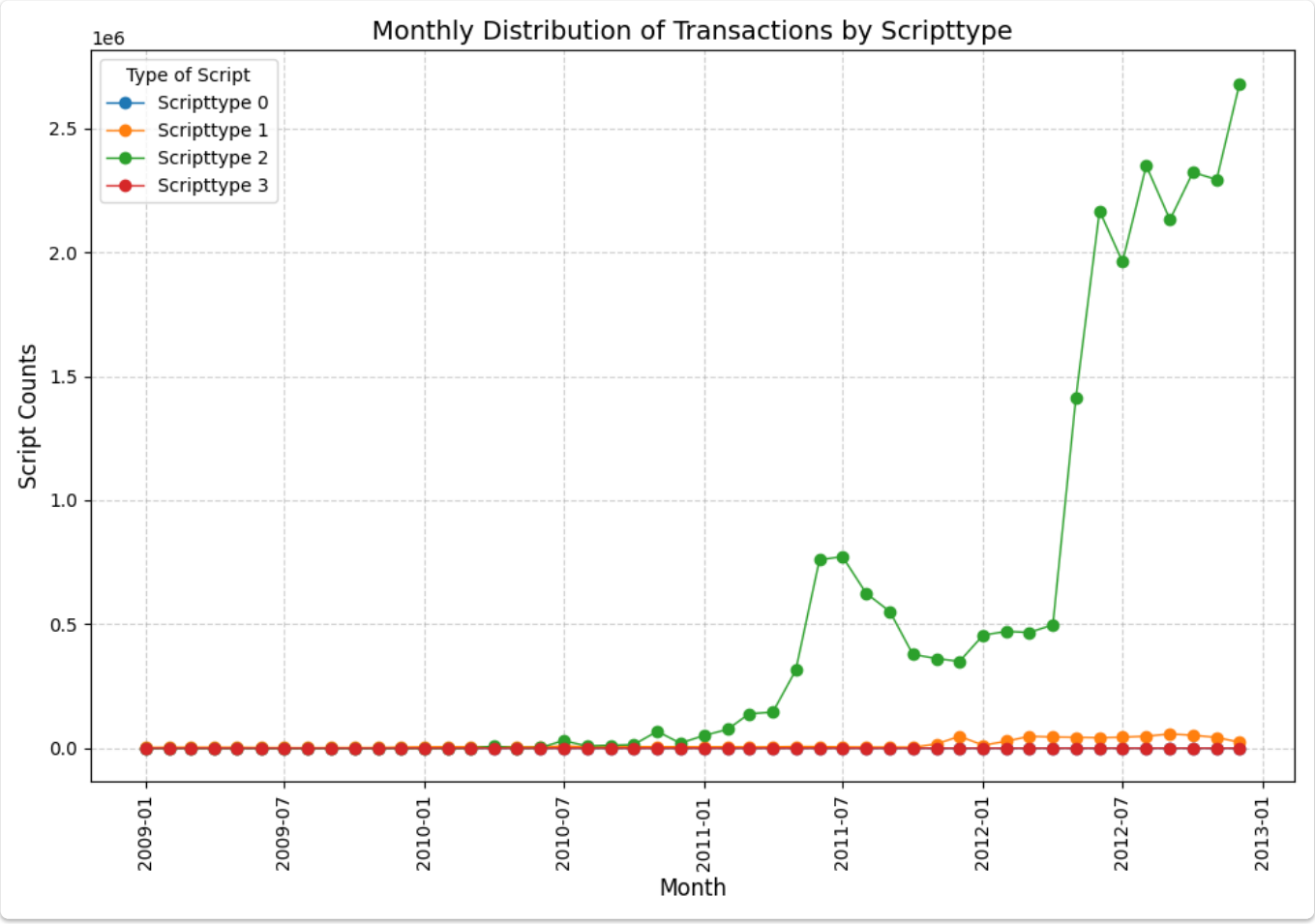
(1)



Dal grafico (1) emerge che la congestione della blockchain ha mostrato una crescita costante dal 2009 al 2012, con un picco nel 2012, indicando un aumento dell'adozione di Bitcoin. Tuttavia, il `Fee/Size ratio` (rapporto tra fee e dimensione delle transazioni) non è aumentato in parallelo; anzi, era alto nei primi anni e ha poi registrato un calo a partire dal 2011. Questo suggerisce che, nonostante l'aumento delle transazioni, il costo per byte è diminuito, probabilmente grazie a una maggiore efficienza del protocollo.

Alcuni picchi isolati del `Fee/Size ratio` nel 2009 e nel 2010 potrebbero essere dovuti a un numero limitato di transazioni con fee elevate.

(2)

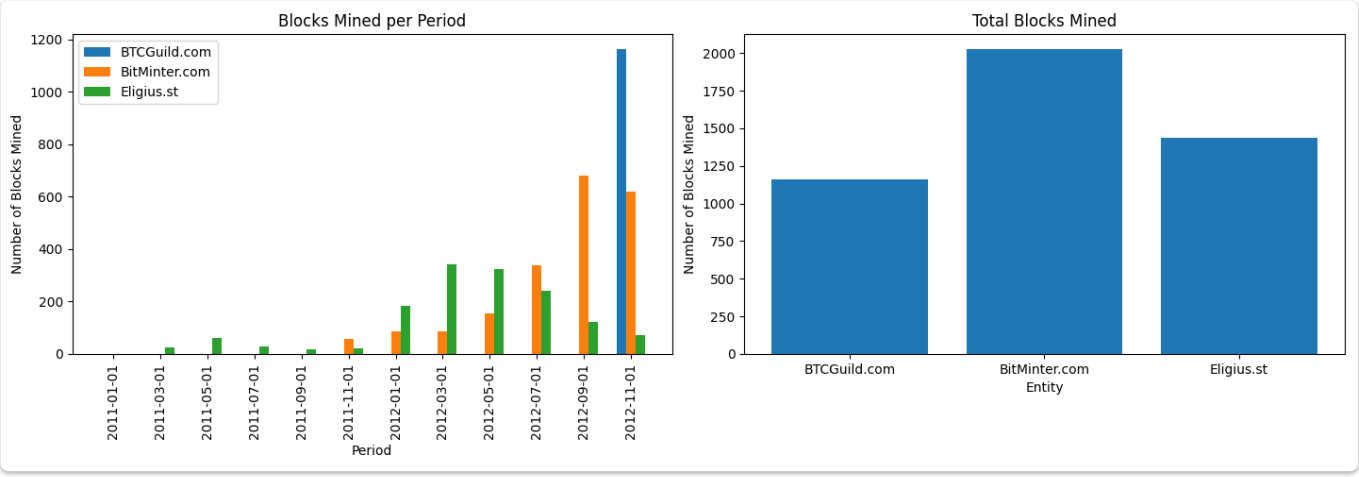


Il grafico (2) illustra l'andamento mensile delle transazioni nella blockchain, suddivise per `ScriptType`, dal 2009 al 2013. Sull'asse orizzontale sono riportati i mesi, mentre sull'asse verticale si osserva il numero totale di transazioni per ciascun tipo di script.

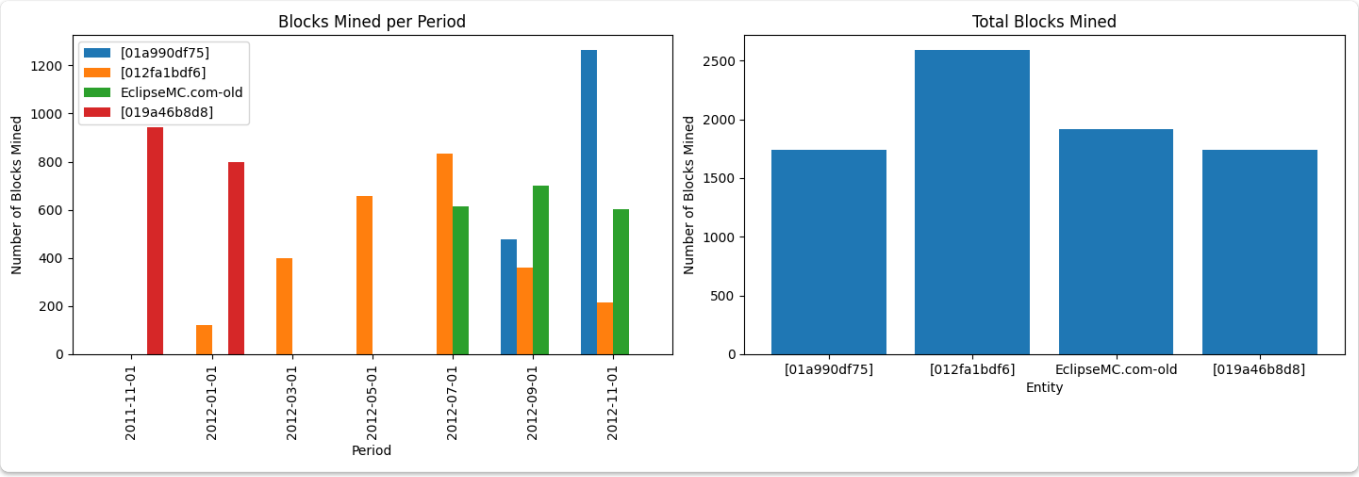
Dal 2011, si nota una crescita significativa per `ScriptType 2`, che si impone come il più utilizzato, raggiungendo i valori massimi alla fine del 2012 e nei primi mesi del 2013. Questo trend suggerisce un'adozione crescente di questo tipo di script, probabilmente per la sua efficienza o compatibilità con il volume di transazioni in aumento.

Gli altri tipi di script (0, 1 e 3) mostrano invece un uso costante e molto inferiore nel periodo analizzato, suggerendo che la rete ha preferito un tipo specifico di script per gestire la maggior parte delle transazioni.

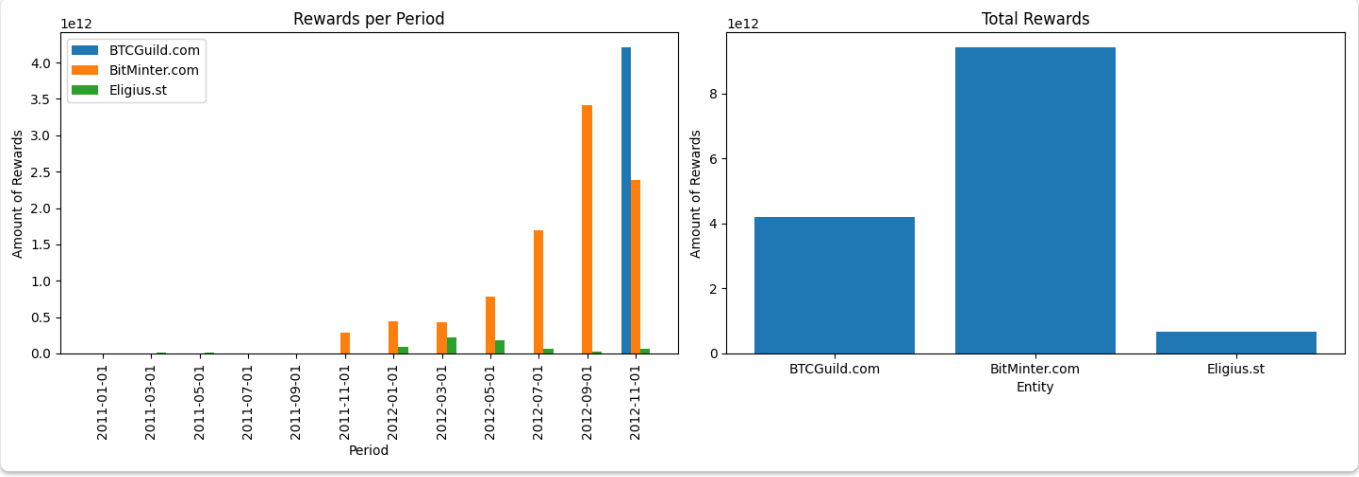
(3)


Blocks Mined per Period
BTCGuild.com
BitMinter.com
Eligius.st
Number of Blocks Mined
Period
Total Blocks Mined
Number of Blocks Mined
BTCGuild.com
BitMinter.com
Eligius.st
Entity

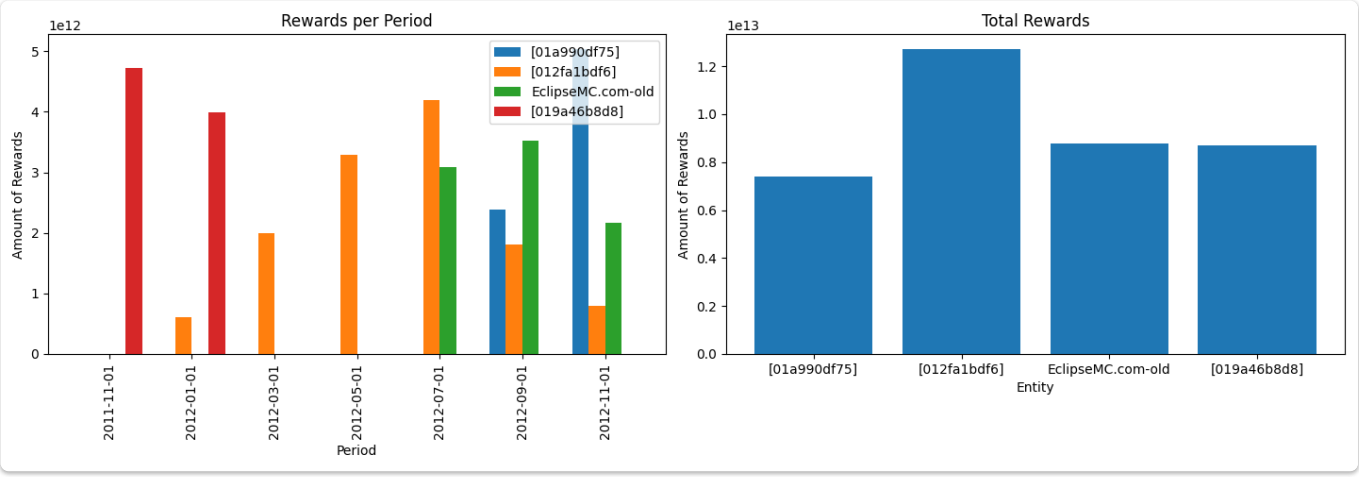
(4)


Blocks Mined per Period
[01a990df75]
[012fa1bdf6]
EclipseMC.com-old
[019a46b8d8]
Number of Blocks Mined
Period
Total Blocks Mined
Number of Blocks Mined
[01a990df75]
[012fa1bdf6]
EclipseMC.com-old
[019a46b8d8]
Entity

(5)


Rewards per Period
1e12
BTCGuild.com
BitMinter.com
Eligius.st
Amount of Rewards
Period
Total Rewards
1e12
Amount of Rewards
BTCGuild.com
BitMinter.com
Eligius.st
Entity

(6)



I quattro grafici presentati offrono una panoramica dettagliata sulle attività di mining relative a vari pool e wallet, evidenziando il numero di blocchi minati e le ricompense distribuite per periodi di tempo specifici.

Nel primo set di grafici, vediamo l'attività di mining suddivisa per `pool` (3), con un focus su `BTCGuild.com`, `BitMinter.com` ed `Eligius.st`, e per i `top_4_miner` (4). A sinistra, il grafico "Blocks Mined per Period" mostra l'andamento nel tempo del numero di blocchi minati da ciascuna entità. Notiamo un incremento significativo dell'attività di mining a partire dalla metà del 2012 sul lato pool (3), con `BitMinter.com` che domina in termini di blocchi minati nell'ultimo periodo del grafico. A destra, il grafico "Total Blocks Mined" riassume il numero totale di blocchi minati per ciascun pool durante l'intero periodo analizzato, dove `BitMinter.com` emerge come il pool più attivo, seguito da `BTCGuild.com` ed `Eligius.st`.
Nel grafico (4) `[01a990df75]` ha mostrato un significativo contributo durante il primo quadrimestre, per poi interrompere la propria attività di mining. Al contrario, l'entità `[012fa1bdf6]`, pur mostrando fluttuazioni, ha mantenuto una certa costanza nel tempo. Inoltre, è interessante notare che `[01a990df75]` ha registrato un picco nella quantità di blocchi minati nell'ultimo bimestre, nonostante la sua attività si fosse limitata a soli due mesi.

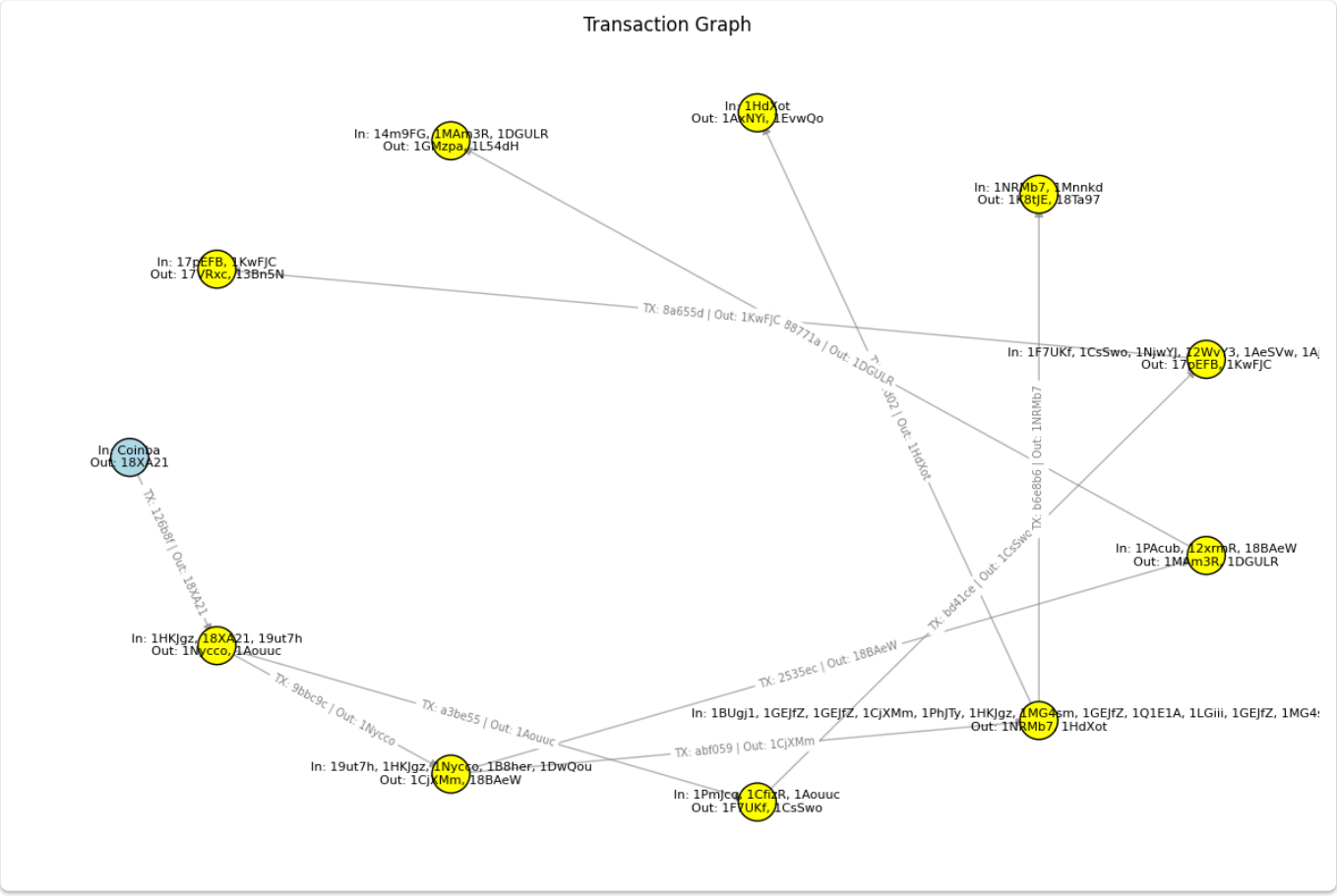
Il secondo set di grafici offre un'analisi dettagliata delle ricompense associate ai blocchi minati dai vari pool (5) e dai singoli wallet (6). È fondamentale sottolineare che le ricompense sono direttamente correlate al numero di blocchi minati, rendendo questa sezione particolarmente pertinente rispetto a quanto analizzato nel primo set.

Nel grafico (5), le ricompense totali per ciascun pool rivelano un chiaro legame con l'attività di mining esaminata in precedenza. Si osserva come `BitMinter.com`, che ha registrato il numero più elevato di blocchi minati, si distingua anche per l'ammontare delle ricompense ricevute. Analogamente, anche `BTCGuild.com`, pur avendo un volume di blocchi inferiore, riesce a ottenere ricompense significative, suggerendo una gestione operativa ottimale.

Nel grafico (6) gli indirizzi più attivi nel mining, come quello associato a `[012fa1bdf6]`, hanno accumulato ricompense consistenti nel tempo, dimostrando che una continuità nell'attività di mining, sebbene caratterizzata da fluttuazioni, può condurre a risultati finanziariamente vantaggiosi.

In conclusione, l'analisi congiunta di questi grafici permette di formulare importanti considerazioni sul nesso tra l'attività di mining e la distribuzione delle ricompense. I dati raccolti suggeriscono che i pool con una maggiore attività di mining non solo minano un numero superiore di blocchi, ma riescono anche a ottenere ricompense nettamente più elevate. Ciò mette in evidenza l'importanza della costanza e dell'efficienza nelle operazioni di mining all'interno del contesto economico delle criptovalute.

(7)



La funzione `build_transaction_graph` costruisce un grafo partendo da una transazione iniziale e analizzando le transazioni correlate fino a una certa profondità `k`. Essa mantiene un elenco di transazioni da esaminare e, per ogni transazione visitata, raccoglie informazioni sui suoi input e output.

Ogni transazione viene aggiunta come nodo nel grafo, con collegamenti che rappresentano le relazioni tra di esse. In questo modo, si crea una rete di transazioni che permette di visualizzare come gli output di una transazione vengano utilizzati come input in quelle successive. L’analisi continua fino a raggiungere la profondità massima stabilita o a esaurire le transazioni da esplorare.

Esecuzione dello script Python

Dopo aver decompresso l’archivio ZIP, il primo passo è creare e configurare un ambiente virtuale per gestire in modo isolato le librerie e le dipendenze del progetto.

- Apri il terminale, spostati nella directory principale del progetto e crea l'ambiente virtuale con il comando:

```
python3 -m venv venv
```

- Una volta creato, attiva l'ambiente virtuale con il comando appropriato per il tuo sistema operativo:

`macOS/Linux`:

```
source venv/bin/activate
```

`Windows`:

```
venv\Scripts\activate
```

- Con l'ambiente virtuale attivo, installa tutte le dipendenze necessarie eseguendo il comando:

```
pip install -r requirements.txt
```

- Questo assicurerà che tutte le librerie richieste per il progetto siano pronte all'uso nell'ambiente virtuale.

Per avviare il progetto, basta eseguire il file `main.py` con il comando:

```
python main.py
```

Lo script verrà eseguito utilizzando l'ambiente virtuale appena configurato.

All'interno della cartella `/data/processed/indirizzi_wallet_explorer` si trovano i quattro file `CSV` che contengono tutti i dati necessari per le analisi successive.
Se si desidera osservare il processo di scraping in tempo reale, si può eliminare (o spostare) la cartella

`/indirizzi_wallet_explorer`; lo script provvederà a ricrearla automaticamente nella posizione appropriata.