WDB Mini-Challenge Dokumentation

Thema: Scrapen der Posts über Crypto-Coins von Reddit

Autoren: Aaron Brülisauer & Florian Baumgartner

06. Dezember 2024

git repo: *git@github.com:CryptoFraudDetection/wdb-submission-fa.git*

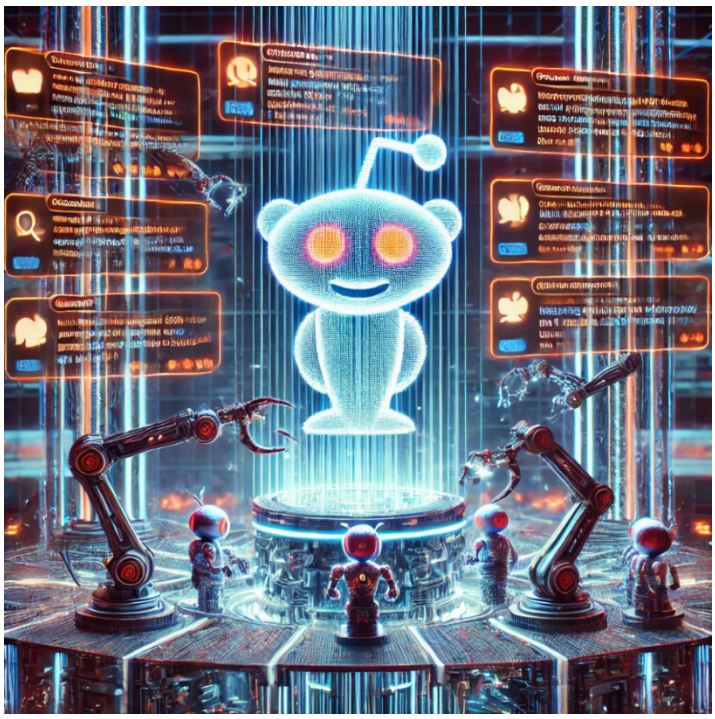


Bild generiert von ChatGPT

Einleitung

In unserer Challenge-X benötigen wir Daten von Posts von diversen sozialen Medien um anhand diesen Informationen ein ‘Warnsystem’ für betrügerische Krypto-Coins zu entwickeln.
Reddit ist beispielsweise ein sehr beliebtes soziales Medium, um über Kryto-News miteinander zu kommunizieren. Es ist so aufgebaut, dass sich Nutzer in einem "Überthema" (Subreddit) über genau dieses Thema unterhalten können und Posts sowie Kommentare verfassen.

Für die Datenbeschaffung von Reddit wurden wegen den Kriterien vom Modul ‘Webdatenbeschaffung’ und diverser Einschränkungen beim Scrapen mit Selenium, zwei verschiedene Vorgehensweisen angewendet. Eimal ein Approach mit Selenium und einmal über GoogleResults mit einer API.
Aaron und **Florian** kümmern sich deswegen in dieser Mini-Challenge um die Datenbeschaffung der Posts von Reddit mit Selenium (ohne API). Alle relevanten Dateien die verwendet werden, sind im `**README.md**` des Git-Repo’s (siehe Titelseite) angegeben.

Das Ziel dieser Mini-Challenge besteht darin, diverse Posts inkl. Kommentare in verschiedenen Subreddits mit diversen Suchanfragen (Queries) zu scrapen. Mit einer EDA werden die Daten untersucht. Zum Schluss soll dann mit einer API auf ein vorgefertigtes Modell zugegriffen werden, welches pro Post ein Sentiment ausgeben wird. Die weitere Verwendung wird dann in der Challenge-X behandelt.

Problematik

Da mit Selenium gescraped werden soll, muss die Seite direkt angesteuert und analysiert werden. Die aktuelle Reddit-Website erlaubt aber fast keine ferngesteuerten Zugriffe auf die Website, so wird man sehr schnell geblockt. Nach einiger Recherche stellt sich heraus, dass sich old.Reddit.com, die alte Reddit-Website, die nach-wie-vor online ist, für solche Fernzugriffe gut eignet. Der einzige Haken: Pro Query in Subreddit können nur ~250 Posts geladen werden. Nach ~300 Beiträgen ist also Schluss, auf ältere Einträge haben wir keinen Zugriff.

Viele Anfragen auf die Website führen auch bei old.reddit dazu, dass man geblockt wird, deshalb wurden Proxies verwendet. Die Schwierigkeit bei Proxies ist, dass nicht jeder geeignet ist, um reddit zu scrapen (z. B. wenn sich der Proxy in einem Land befindet wo Reddit verboten wurde). Das führt auch dazu, dass längere Suchzeiten aufkommen, bis ein guter/geeigneter Proxy gefunden wird.

Die Integration von Exceptions im Code ist unabdinglich. Da kein Crash des Codes stattfinden soll, haben wir lediglich auf dem Warning-Level geloggt, denn von dem, in Ramen der CX entwickelten, Logger führt der Error-Level zu einem Programmabbruch. Reddit ist nicht einfach zu scrapen, so mussten wir sehr viel Zeit investieren um unseren Code robust und lauffähig zu machen. Es kam immer wieder vor, dass der Scraper auf Probleme stiess, die dann behandelt werden mussten.

Funktionsweise des Scrapers

Der Kern des Codes besteht aus der Klasse `**RedditScraper**`**,** die verschiedene Methoden enthält, um Beiträge zu suchen, Metadaten und Inhalte von Beiträgen zu extrahieren, Kommentare zu sammeln und die gesammelten Daten zu speichern und zu verarbeiten.

Initialisierung und Konfiguration:

Die Klasse `**RedditScraper**` wird mit mehreren Parametern initialisiert:

`logger`:	Ein Logger-Objekt zur Protokollierung von Ereignissen und Fehlern.
`base_url`:	Die Basis-URL von Reddit, standardmäßig "https://old.reddit.com".
`wait_range`:	Ein Zeitintervall für zufällige Wartezeiten zwischen den Anfragen, um das Scraping menschlicher wirken zu lassen und somit seltener blockiert zu werden.
`headless`:	Ein boolescher Wert, der angibt, ob der Webbrowser im Headless-Modus laufen soll (ohne sichtbare Benutzeroberfläche).
Weitere Parameter	für Wiederholungsversuche und Limits.

Proxy-Management:

Um IP-Blockierungen durch Reddit zu umgehen, verwendet der Scraper Proxys. Die Methode **`_get_next_proxy`** lädt eine Liste kostenloser Proxys von einer angegebenen URL und wählt zufällig einen Proxy aus dieser Liste aus. Dies ermöglicht es dem Scraper, bei Bedarf die IP-Adresse zu wechseln. Im Allgemeinen wird dies von der Klasse autonom gehandhabt. Tritt ein Problem auf so wird der Browser beendet, mit einem anderen Proxy erneut gestartet und erneut probiert (**`load_page`**).

Warten und Laden von Seiten:

`_wait` führt eine zufällige Pause innerhalb des definierten **`wait_range`** durch. Die Methode **`_wait_for_element`** wartet darauf, dass ein bestimmtes Element auf der Seite geladen ist, bevor der Code fortfährt. Dies ist wichtig, um sicherzustellen, dass alle notwendigen Elemente vorhanden sind, bevor versucht wird, Daten zu extrahieren.

Blockierungsüberprüfung:

`_check_if_blocked` prüft, ob der Scraper von Reddit blockiert wurde, indem geschaut wird, ob ein leeres HTML zurückgegeben wurde.

Laden von Seiten:

Die Funktion **`_load_page`** lädt eine gegebene URL und überprüft mithilfe von **`_check_if_blocked`**, ob der Zugriff erfolgreich war. Bei Fehlern oder Blockierungen versucht sie, den Proxy zu wechseln und die Seite erneut zu laden, basierend auf der Anzahl der erlaubten Wiederholungsversuche (**`page_retry`**).

Extraktion von Beitragsmetadaten:

`_extract_post_metadata` extrahiert wichtige Informationen aus einzelnen Beiträgen, wie:

`id`:	Die eindeutige Kennung des Beitrags.
`title`:	Der Titel des Beitrags.
`url`:	Die URL des Beitrags.
`num_comment`:	Die Anzahl der Kommentare.
`date`:	Das Datum der Erstellung.
`author`:	Der Benutzername des Autors.
`author_url`:	Die Profil-URL des Autors.

Dies wird erreicht, indem spezifische XPath-Ausdrücke verwendet werden, um die entsprechenden HTML-Elemente auf der Seite zu finden.

Extraktion von Kommentaren:

`**_extract_comments**` ruft `**_extract_comments_rec**` auf, eine rekursive Funktion, die die hierarchische Struktur der Kommentare durchläuft. Sie sammelt den Text und den Autor jedes Kommentars und fügt, falls vorhanden, auch die Antworten (Child-kommentare) hinzu.

Suchen und Sammeln von Beiträgen:

`**scrape_post_list**` führt eine Suche in einem bestimmten Subreddit nach einem bestimmten Suchbegriff durch und sammelt bis zu einem festgelegten Limit Beiträge. Sie konstruiert die Such-URL mit den entsprechenden Parametern und extrahiert die Suchergebnisse.

`**scrape_multipage_post_list**` erweitert diese Funktionalität, indem sie mehrere Seiten von Suchergebnissen durchläuft, um mehr Beiträge zu sammeln, als auf einer einzelnen Seite angezeigt werden können. Sie berücksichtigt dabei auch ein mögliches Startdatum, um ältere Beiträge auszuschließen.

Sammeln von Beitragsinhalten:

Nachdem die Liste der Beiträge gesammelt wurde, werden möglicherweise noch die Inhalte der einzelnen Beiträge und deren Kommentare benötigt. Die Methode `**scrape_post_content**` lädt die Seite eines einzelnen Beitrags und extrahiert den Haupttext sowie die Kommentare mittels der Funktion `**_extract_comments**`.

Verarbeiten unvollständiger Daten:

`**get_posts_without_content**` identifiziert Beiträge, bei denen der Inhalt noch fehlt.
`**scrape_all_missing_post_contents**` durchläuft diese Beiträge und versucht, die fehlenden Inhalte zu sammeln, wobei sie bei Bedarf Wiederholungsversuche unternimmt.

Steuerung des WebDrivers:

`**start_driver**` initialisiert den WebDriver von Selenium, unter Berücksichtigung der Proxy-Einstellungen. Wenn ein Proxy nicht funktioniert, wird versucht, einen neuen zu verwenden. Die Methode `**quit**` beendet die WebDriver-Sitzung.

Datenumwandlung und Speichern:

`**to_dataframe**` konvertiert die gesammelten Daten in ein pandas DataFrame, was die weitere Verarbeitung und Speicherung erleichtert. Die Daten können dann exportiert werden.

Haupt-Scraping-Methode:

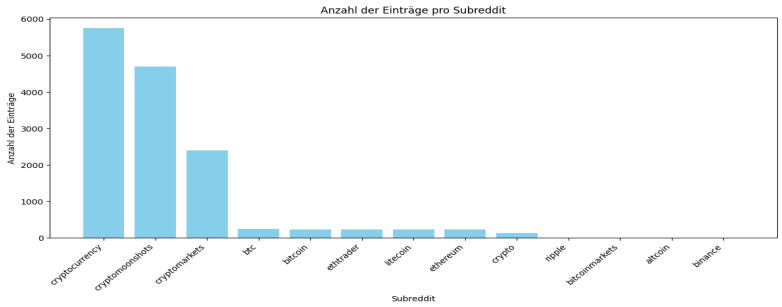
Die Methode `**scrape**` fasst den gesamten Prozess zusammen. Sie startet den WebDriver, sammelt die Beitragsliste, extrahiert die Inhalte der Beiträge und Kommentare, beendet den WebDriver und gibt die gesammelten Daten als DataFrame zurück.

Datastorage

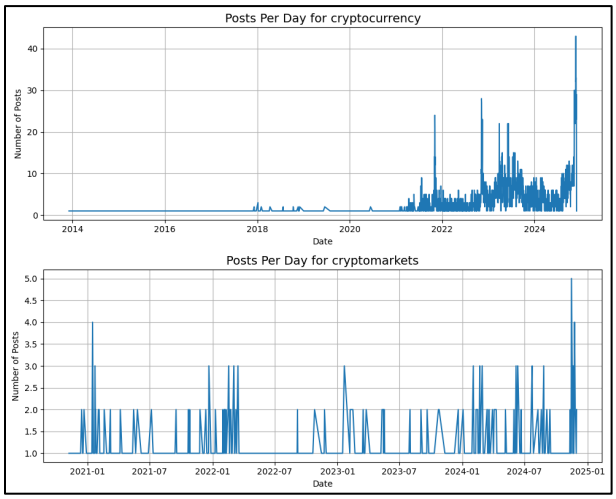
Die Daten werden mittels dem Notebook `**90-wdb-scrape-reddit.qmd**` gescraped und danach auf der Elasticsearch Datenbank unter dem index ‘wdb’ abgespeichert. Die Datenbank läuft auf einem von Gabriel Torres Gamez gehosteten Raspberry Pi 4b und kann nur von Teammitgliedern der Challenge-X und Herrn Gamez über einen Tailscale-VPN-Tunnel erreicht werden. Für die Verbindung zur Datenbank wurde die Library 'CryptoFraudDetection.elasticsearch' im Challenge-X Repo angelegt.

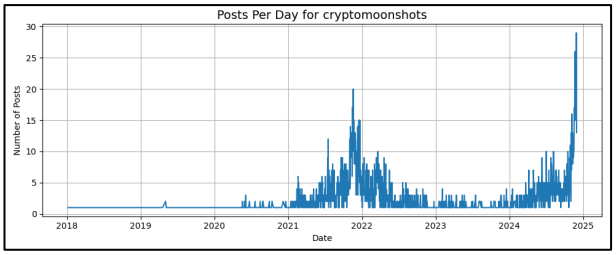
EDA

Nach der Datensammlung und Speicherung erfolgt eine EDA (`**90-wdb-scrape-reddit.qmd**`). Die wichtigsten insights werden im folgenden Absatz erläutert.
In dieser Visualisierung ist die Anzahl an Einträge pro Subreddit der gescrapten Daten zu sehen. Der Subreddit `CryptoCurrency` hat am meisten Einträge. Dies ist nachvollziehbar, denn es ist der grösste und bekannteste Subreddit in der Kryptoszene. Es wurden bewusst viele Subreddits und Coins zum Scrapen ausgewählt, denn es sollte nicht an Daten mangeln, doch nach etwas über 10'000 Posts wurde der Scrapeprozess bewusst beendet, da dies für den Ramen des Moduls WDB ausreichend ist. In den Subreddits Ripple, Bitcoinmarkets, Altcoin und Biance wurden keine Daten gefunden, bzw. sie wurden mangels Zeit nie abgefragt (Prozess vorher beendet).



Für die folgenden Graphiken wurde von den 3 grössten Subreddits die gescrapten Posts auf einer Timeline projiziert. Da wir auf eine gewisse Anzahl an Posts pro Subreddit-Query-Kombination scrapen konnten, sind die meisten Posts sehr neu.





Data Enrichment mittels API

Nachdem die Reddit-Posts gesammelt wurden, werden die Einträge nun mithilfe einer Sentiment-Analyse bewertet (`**90-wdb-scrape-reddit.qmd**`). Hierzu nutzen wir die **Azure OpenAI API**, die den Sentiment-Wert eines Textes auf einer Skala von 1 bis 10 ausgibt. Diese Bewertung gibt uns eine quantitative Einschätzung der emotionalen Stimmung der Beiträge. Die API ermöglicht uns, große Datenmengen effizient und standardisiert zu bewerten und GPT-4 ist hierzu gut geeignet. Die ermittelten Sentiment-Werte werden

anschliessend als Features in Modellierungen integriert. Dadurch verbessern wir die Qualität und Aussagekraft unserer Modelle.

Unit Tests

`test_scraper_reddit.py` testet den Scraper, wobei die Initialisierung allgemein, die Sammlung der Post's, sowie Extrahierung der Kommentare getestet wird. **`test_sentiment.py`** testet die Verwendung der Azure OpenAI API, wobei das Erzeugen des Clients, das Erzeugen einer Chat-Bot-Instanz, das Generieren einer Antwort mittels Chat-Bot und das Klassifizieren mittels Chat-Bot getestet wird.