StackExchange Text Mining and Visualisation Project

Bailey Mulcahy

GitHub | LinkedIn

Introduction

This project explores the **Personal Finance & Money StackExchange site** through a text-based data analysis. The site contains a large volume of user-generated text and structured interactions, making it an excellent target for data cleansing, transformation, and exploratory analysis. Using Python, Pandas, Matplotlib, Seaborn, and TF-IDF vectorisation, I processed over 1 GB of XML data into CSV format for visualisation and analysis.

In this project, I experimented with new file formats - in particular, **XML → CSV → Pandas DataFrame → visualisation**. The report is generated from a `.qmd` Quarto Markdown file and converted to PDF. The process included:

- **Converting XML Files to CSV**
- **Loading CSV Files as Pandas DataFrames**
- **Text Analysis and Visualisations**
- **Insights from the Analysis**
- **Conclusion**

Converting XML Files to CSV

StackExchange data is provided in XML format. The following function converts an XML file to CSV for easier processing in Python.

```
# Import required libraries
import xml.etree.ElementTree as ET
import pandas as pd
import os
import re
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer

# Function to convert an XML file to a CSV file
def convert_xml_to_csv(xml_file, csv_file):
    tree = ET.parse(xml_file)
    root = tree.getroot()
    rows = [child.attrib for child in root]
    df = pd.DataFrame(rows)
    df.to_csv(csv_file, index=False)
    print(f"Converted: {xml_file} -> {csv_file}")
```

This function is used to convert all XML files to CSV:

```
convert_xml_to_csv("Posts.xml", "Posts.csv")
convert_xml_to_csv("Users.xml", "Users.csv")
convert_xml_to_csv("Tags.xml", "Tags.csv")
convert_xml_to_csv("Comments.xml", "Comments.csv")
convert_xml_to_csv("Votes.xml", "Votes.csv")
convert_xml_to_csv("PostLinks.xml", "PostLinks.csv")
convert_xml_to_csv("PostHistory.xml", "PostHistory.csv")
convert_xml_to_csv("Badges.xml", "Badges.csv")
```

```
Converted: Posts.xml -> Posts.csv
Converted: Users.xml -> Users.csv
Converted: Tags.xml -> Tags.csv
Converted: Comments.xml -> Comments.csv
Converted: Votes.xml -> Votes.csv
```

```
Converted: PostLinks.xml -> PostLinks.csv
Converted: PostHistory.xml -> PostHistory.csv
Converted: Badges.xml -> Badges.csv
```

Loading CSV files as Pandas DataFrames

Load the CSV files and inspect their structure:

```python
posts_df = pd.read_csv("Posts.csv")
users_df = pd.read_csv("Users.csv")
tags_df = pd.read_csv("Tags.csv")
comments_df = pd.read_csv("Comments.csv")
votes_df = pd.read_csv("Votes.csv")
postlinks_df = pd.read_csv("PostLinks.csv")
posthistory_df = pd.read_csv("PostHistory.csv")
badges_df = pd.read_csv("Badges.csv")

# Display the shape of each DataFrame
print("Posts:", posts_df.shape)
print("Users:", users_df.shape)
print("Tags:", tags_df.shape)
print("Comments:", comments_df.shape)
print("Votes:", votes_df.shape)
print("PostLinks:", postlinks_df.shape)
print("PostHistory:", posthistory_df.shape)
print("Badges:", badges_df.shape)
```

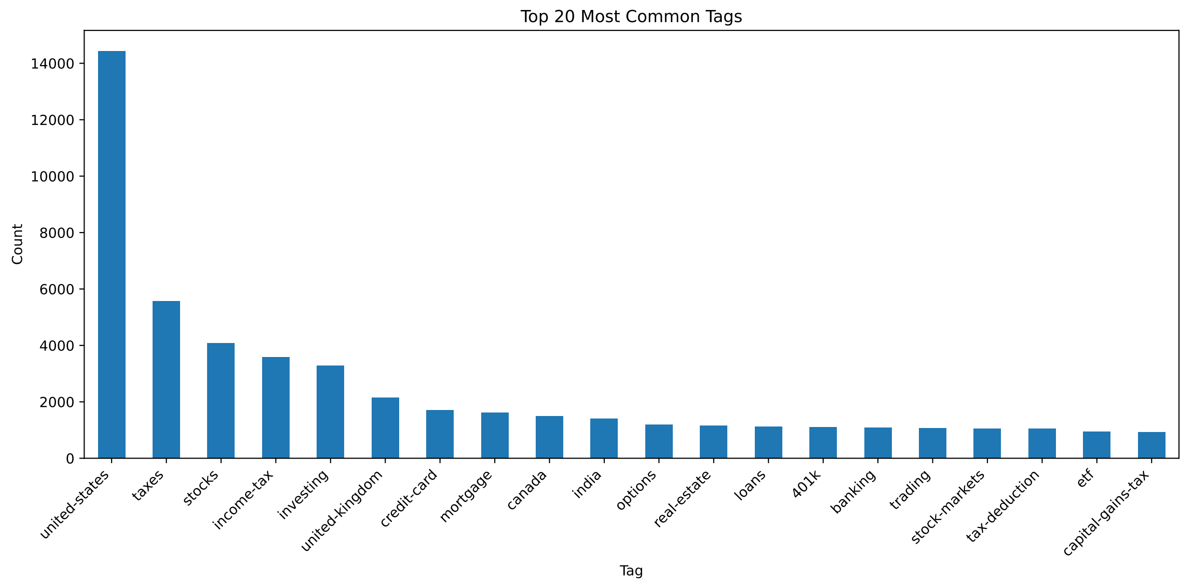
```
Posts: (112485, 22)
Users: (96900, 12)
Tags: (1010, 5)
Comments: (228682, 7)
Votes: (691728, 6)
PostLinks: (8260, 5)
PostHistory: (321234, 10)
Badges: (169040, 6)
```

Text Analysis and Visualisations

This section presents several visualisations and text-based analyses of the StackExchange data.

Visualisation 1: Top 20 Most Common Tags

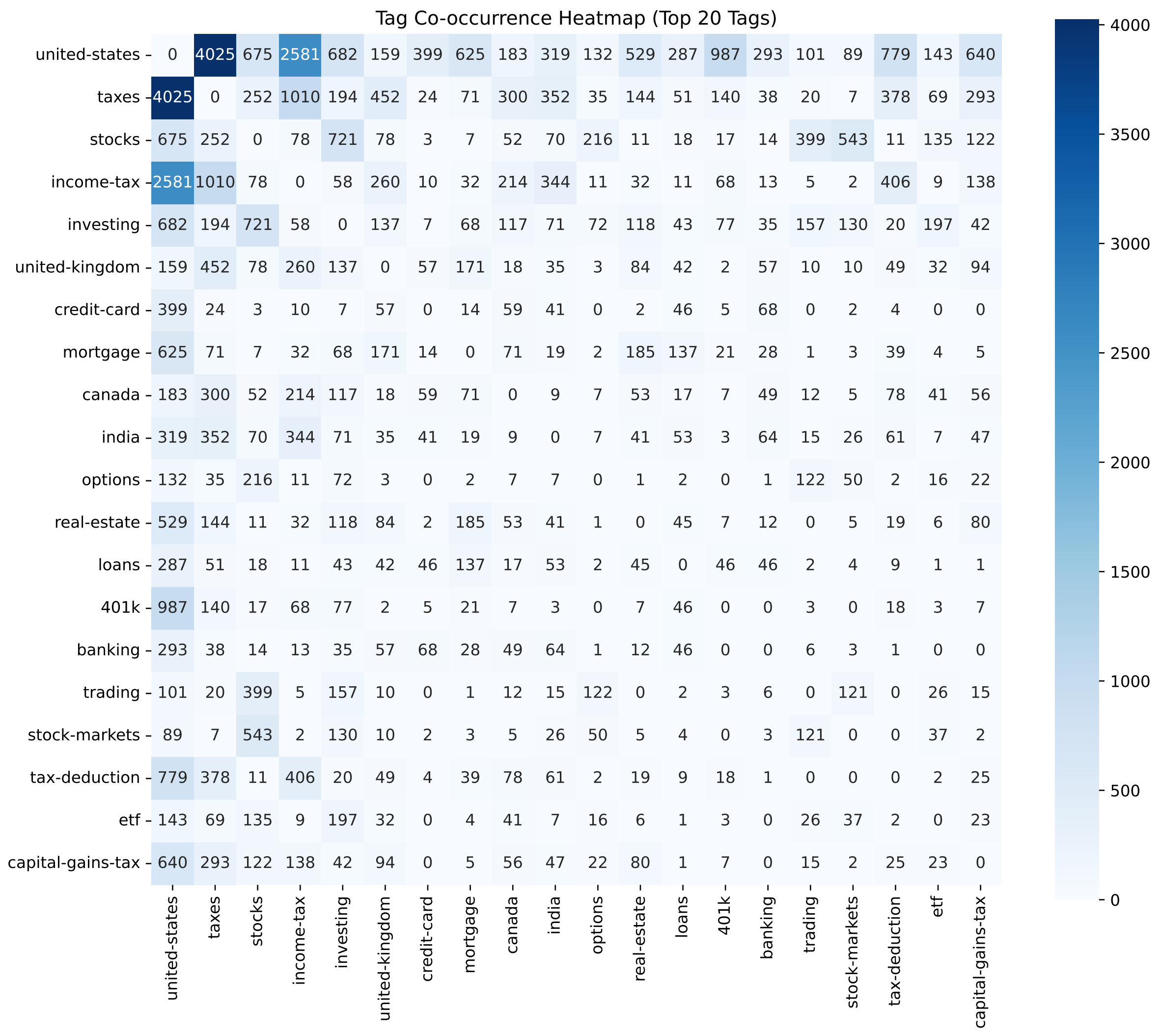
```python
# Function to split tags by '|', removing empty entries
def extract_tags_pipe(s):
    return [tag for tag in s.split('|') if tag]

# Apply pipe splitting function to 'Tags' column, after dropping NaN values
all_tags = posts_df['Tags'].dropna().apply(extract_tags_pipe)

# Flatten list of tags (separates groups from each post into one single list)
flat_tags = [tag for sublist in all_tags for tag in sublist]

# Count occurrences of each tag and select the top 20
tag_counts = pd.Series(flat_tags).value_counts().head(20)

# Plot top 20 tags bar chart if tag list is not empty
if not tag_counts.empty:
    tag_counts.plot(kind='bar', title='Top 20 Most Common Tags',
 ↪  figsize=(12,6))
    plt.xlabel('Tag')
    plt.ylabel('Count')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
else:
    print("No tags found to plot.")
```



In this first visualisation we can see the top 20 most common tags to be included in a post. The highest is “united-states” with around 14,000 examples seen, with the second highest being “taxes” with less than half this amount.

Visualisation 2: Tag Co-occurrence Heatmap (Top 20 Tags)

```
# Reuse the tag extraction function defined in visualisation 1
tags_series = posts_df['Tags'].dropna().apply(extract_tags_pipe)

# Flatten list of tags (separates groups from each post into one single list)
 ↪  and get top 20 by count
all_tags = [tag for tags_list in tags_series for tag in tags_list]
top_tags = pd.Series(all_tags).value_counts().head(20).index.tolist()

# If the list of tags is not empty, create co-occurence matrix and heatmap
if not top_tags:
    print("No tags available to build a co-occurrence matrix.")
else:
    # Initialise matrix
    co_matrix = pd.DataFrame(0, index=top_tags, columns=top_tags)

    # Count co-occurrences and keep only those in top 20
    for tags_list in tags_series:
        filtered_tags = [tag for tag in tags_list if tag in top_tags]
        for i in range(len(filtered_tags)):
            for j in range(i+1, len(filtered_tags)):
                co_matrix.loc[filtered_tags[i], filtered_tags[j]] += 1
                co_matrix.loc[filtered_tags[j], filtered_tags[i]] += 1

    # Plot co-occurrence matrix as heatmap using seaborn
    plt.figure(figsize=(12,10))
    sns.heatmap(co_matrix, cmap='Blues', annot=True, fmt='d', square=True)
    plt.title('Tag Co-occurrence Heatmap (Top 20 Tags)')
    plt.show()
```

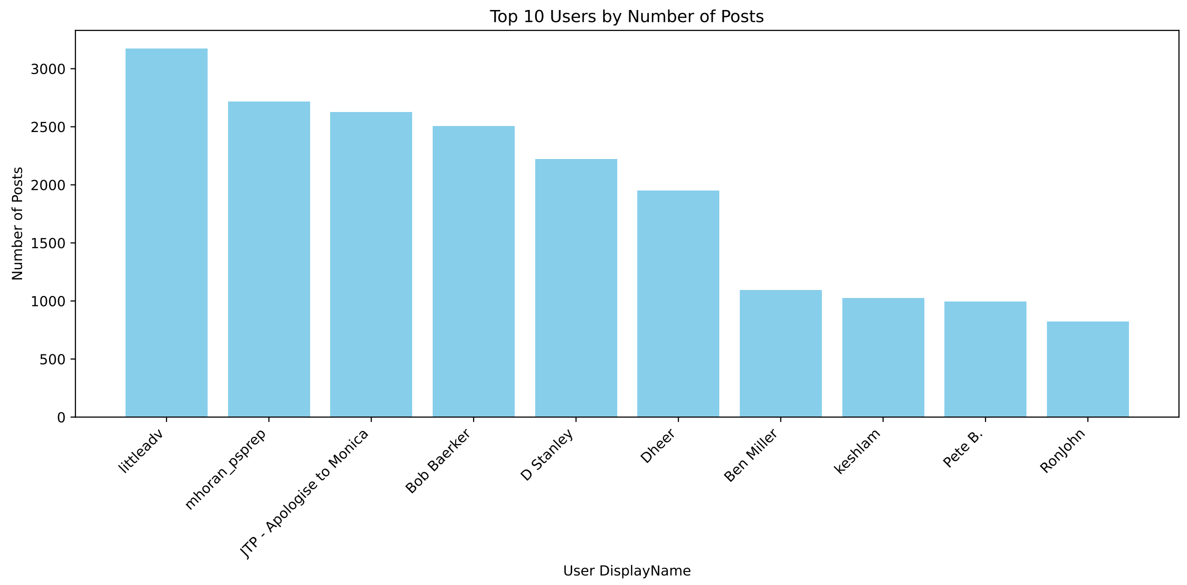


	united-states	taxes	stocks	income-tax	investing	united-kingdom	credit-card	mortgage	canada	india	options	real-estate	loans	401k	banking	trading	stock-markets	tax-deduction	etf	capital-gains-tax
united-states	0	4025	675	2581	682	159	399	625	183	319	132	529	287	987	293	101	89	779	143	640
taxes	4025	0	252	1010	194	452	24	71	300	352	35	144	51	140	38	20	7	378	69	293
stocks	675	252	0	78	721	78	3	7	52	70	216	11	18	17	14	399	543	11	135	122
income-tax	2581	1010	78	0	58	260	10	32	214	344	11	32	11	68	13	5	2	406	9	138
investing	682	194	721	58	0	137	7	68	117	71	72	118	43	77	35	157	130	20	197	42
united-kingdom	159	452	78	260	137	0	57	171	18	35	3	84	42	2	57	10	10	49	32	94
credit-card	399	24	3	10	7	57	0	14	59	41	0	2	46	5	68	0	2	4	0	0
mortgage	625	71	7	32	68	171	14	0	71	19	2	185	137	21	28	1	3	39	4	5
canada	183	300	52	214	117	18	59	71	0	9	7	53	17	7	49	12	5	78	41	56
india	319	352	70	344	71	35	41	19	9	0	7	41	53	3	64	15	26	61	7	47
options	132	35	216	11	72	3	0	2	7	7	0	1	2	0	1	122	50	2	16	22
real-estate	529	144	11	32	118	84	2	185	53	41	1	0	45	7	12	0	5	19	6	80
loans	287	51	18	11	43	42	46	137	17	53	2	45	0	46	46	2	4	9	1	1
401k	987	140	17	68	77	2	5	21	7	3	0	7	46	0	0	3	0	18	3	7
banking	293	38	14	13	35	57	68	28	49	64	1	12	46	0	0	6	3	1	0	0
trading	101	20	399	5	157	10	0	1	12	15	122	0	2	3	6	0	121	0	26	15
stock-markets	89	7	543	2	130	10	2	3	5	26	50	5	4	0	3	121	0	0	37	2
tax-deduction	779	378	11	406	20	49	4	39	78	61	2	19	9	18	1	0	0	0	2	25
etf	143	69	135	9	197	32	0	4	41	7	16	6	1	3	0	26	37	2	0	23
capital-gains-tax	640	293	122	138	42	94	0	5	56	47	22	80	1	7	0	15	2	25	23	0

In this second plot we can see a co-occurrence heatmap including the top 20 tags. This allows us to see which tags are often paired with other tags - for example, we can see that "united-states" and "taxes" often appear together, as well as "united-states" and "income-tax". Other common pairs include "stocks" and "investing", as well as "income-tax" and "tax-deduction".

Visualisation 3: Top 10 Users by Number of Posts

```python
# Count posts per user id and get top 10
post_counts = posts_df['OwnerUserId'].value_counts().head(10)

# Get user display names from users_df for those IDs
top_users = users_df.set_index('Id').loc[post_counts.index, 'DisplayName']

# Combine counts and names into DataFrame
top_users_posts = pd.DataFrame({'DisplayName': top_users, 'PostCount':
 ↪  post_counts.values})

# Plot bar chart
plt.figure(figsize=(12,6))
plt.bar(top_users_posts['DisplayName'], top_users_posts['PostCount'],
 ↪  color='skyblue')
plt.xticks(rotation=45, ha='right')
plt.title('Top 10 Users by Number of Posts')
plt.ylabel('Number of Posts')
plt.xlabel('User DisplayName')
plt.tight_layout()
plt.show()
```



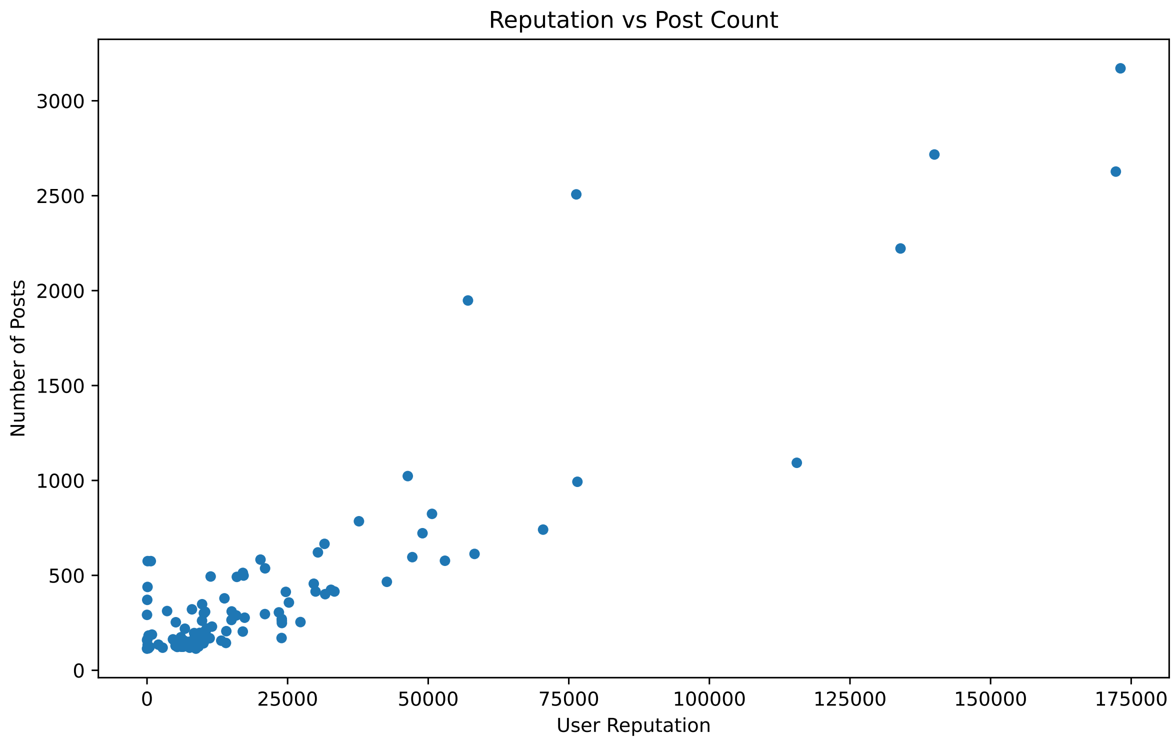
In this third visualisation we can see the top 10 users by their number of posts. This chart is not given to identify the users, but rather to see the number of posts that the top users have posted. We can see that the top user has over 3000 posts, and the 10th user around 1000 posts.

Visualisation 4: User Reputation vs. Post Count (Top 100 Users)

```
# Merge posts and users dataframes on user ID into one table
merged_df = posts_df.merge(users_df, left_on='OwnerUserId', right_on='Id',
 ↪  suffixes=('_post', '_user'))

# Group by display name to compute post count and get reputation
user_post_counts = merged_df.groupby('DisplayName').agg({
    'Id_post': 'count',
    'Reputation': 'first'
}).rename(columns={'Id_post': 'PostCount'})

# Sort users by number of posts in descending order, and get top 100 most
 ↪  active users
top_users = user_post_counts.sort_values(by='PostCount',
 ↪  ascending=False).head(100)

# Create a scatterplot of Reputation vs PostCount for the top 100 users
ax = top_users.plot.scatter(x='Reputation', y='PostCount', title='Reputation
 ↪  vs Post Count', figsize=(10,6))
```

```
# Add axis labels
ax.set_xlabel('User Reputation')
ax.set_ylabel('Number of Posts')
```

```
Text(0, 0.5, 'Number of Posts')
```



In this fourth visualisation we can see the reputation of users in respect to the number of posts they have created. We can see a general positive correlation between the two variables, indicating that those that post more, generally have a higher reputation than those that post less frequently. This plot shows the top 100 most active users (by post count).

Visualisation 5: Posts Mentioning "Tax" or "Loan" in the Title

```
# Replace missing titles with empty strings
posts_df['Title'] = posts_df['Title'].fillna('')

# New boolean columns for mentions of tax or loan that is true if it does
↪  contain the word - not case-sensitive
```

```
posts_df['Tax'] = posts_df['Title'].str.contains(r'\btax\b',
 ↪  flags=re.IGNORECASE)
posts_df['Loan'] = posts_df['Title'].str.contains(r'\bloan\b',
 ↪  flags=re.IGNORECASE)

# Sum of true values in each column to count the total number of posts for
 ↪  each word
keyword_counts = posts_df[['Tax', 'Loan']].sum()

# Convert to DataFrame for formatting
keyword_counts_df = keyword_counts.reset_index()
keyword_counts_df.columns = ['Keyword', 'Number of Posts']

# Display table
keyword_counts_df
```

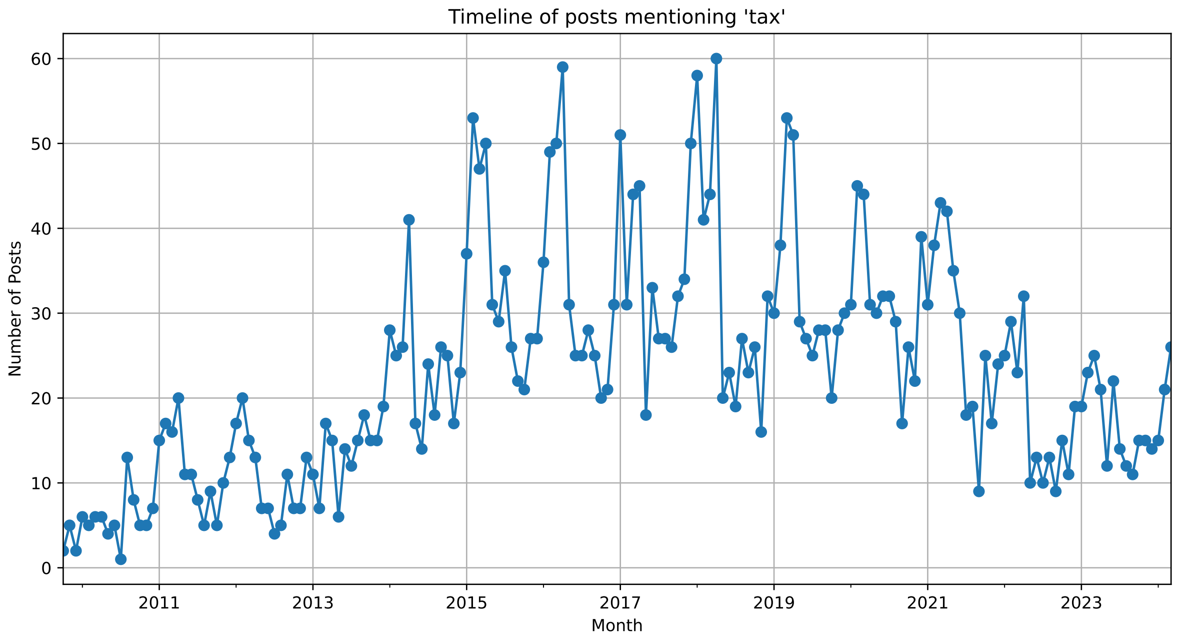
	Keyword	Number of Posts
0	Tax	3976
1	Loan	1271

In this fifth visualisation we have a simple table showing the number of mentions of the words “tax” and “loan”. We can see that tax is mentioned 3976 times, and loan is mentioned 1271 times.

Visualisation 6a: Timeline of Posts Mentioning “Tax”

```
# Make sure CreationDate is datetime format
posts_df['CreationDate'] = pd.to_datetime(posts_df['CreationDate'],
 ↪  errors='coerce')

# Set keyword as "tax" and filter for posts with this keyword - not
 ↪  case-sensitive
keyword = 'tax'
posts_df['Title'] = posts_df['Title'].fillna('') # Replace missing titles
 ↪  with an empty string
posts_df['KeywordMatch'] = posts_df['Title'].str.contains(rf'\b{keyword}\b',
 ↪  flags=re.IGNORECASE, regex=True) # Boolean column to indicate match
```

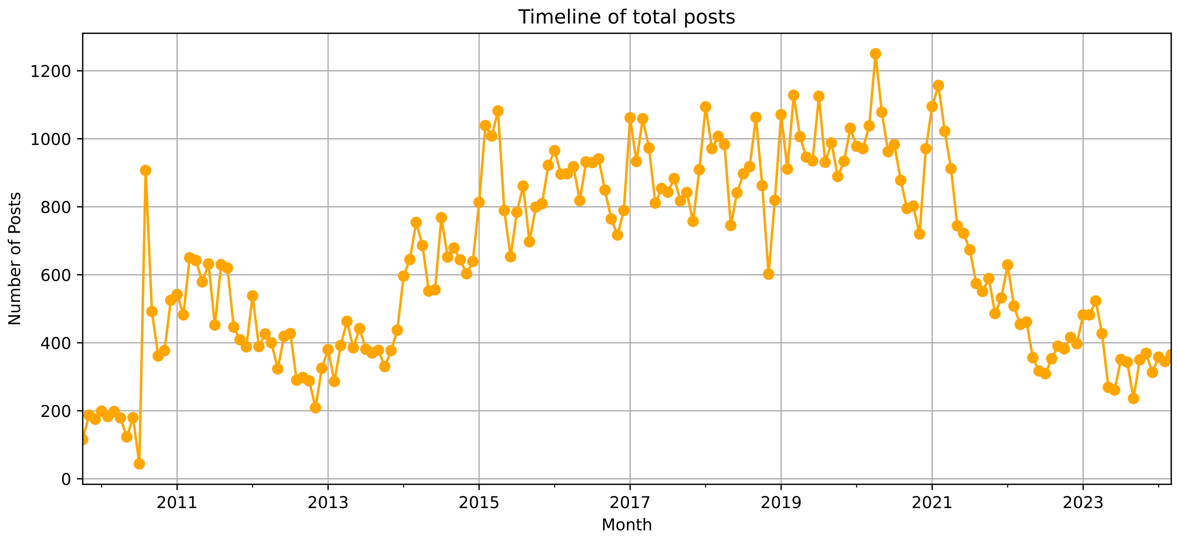
```
# Filter matching posts and group by month, and count number of posts for
 ↪  each month
timeline = posts_df[posts_df['KeywordMatch']].groupby(posts_df[⌋
 ↪  'CreationDate'].dt.to_period('M')).size()

# Plot as a line chart with one dot per month
plt.figure(figsize=(12,6))
timeline.plot(kind='line', marker='o', title=f"Timeline of posts mentioning
 ↪  '{keyword}'")
plt.xlabel('Month')
plt.ylabel('Number of Posts')
plt.grid(True)
plt.show()
```



In this sixth visualisation (part a) we can see the number of posts mentioning “tax” over time. This is grouped into months, with one dot representing each month over the timeline presented in the graph. We can see that the number of posts mentioning tax was in an upwards trend from the beginning of the data collection period until roughly the middle, before experiencing a downwards trend following this peak. This may not necessarily be due to a lower percentage of posts mentioning tax, but rather a lower number of posts overall.

Visualisation 6b: Timeline of Total Posts

```
# Group by month and count total posts
total_timeline =
 ↪  posts_df.groupby(posts_df['CreationDate'].dt.to_period('M')).size()

# Plot as a line chart with one dot per month
plt.figure(figsize=(12,5))
total_timeline.plot(kind='line', marker='o', color='orange', title="Timeline
 ↪  of total posts")
plt.xlabel('Month')
plt.ylabel('Number of Posts')
plt.grid(True)
plt.show()
```

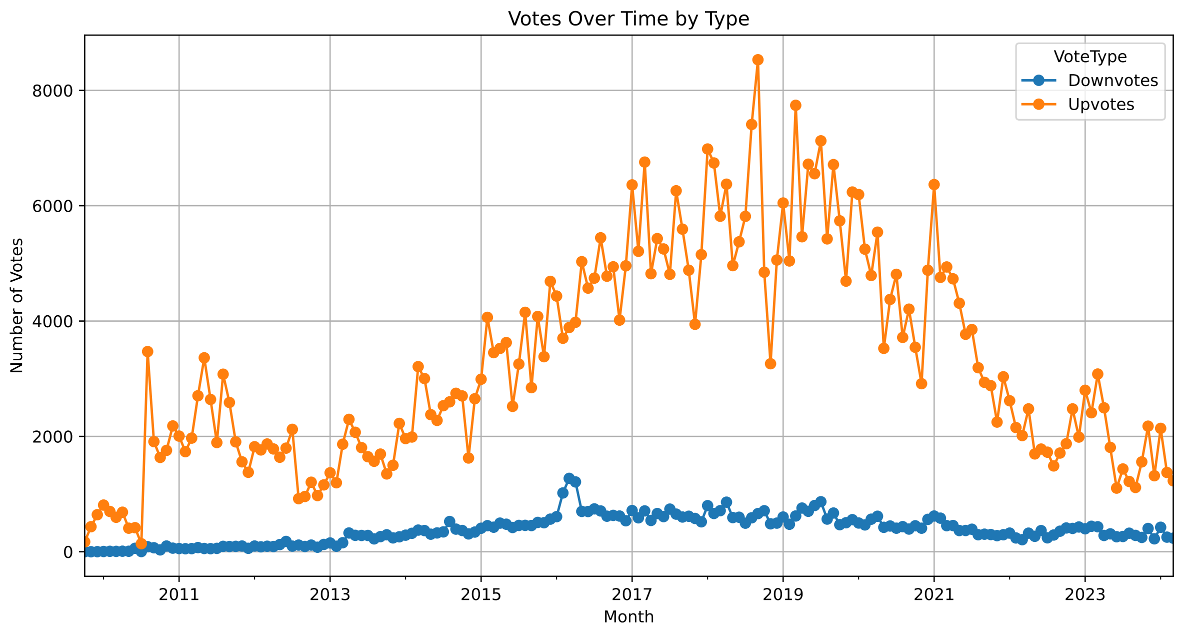


In fact, after looking at the sixth visualisation (part b) we can see that the total number of posts follows a similar pattern. This suggests that the changing number of tax mentions is more likely due to changes in the total number of posts than any other reason.

Visualisation 7: Votes Over Time by Type

```
# Ensure creation data column is in datetime format
votes_df['CreationDate'] = pd.to_datetime(votes_df['CreationDate'],
 ↪  errors='coerce')
```

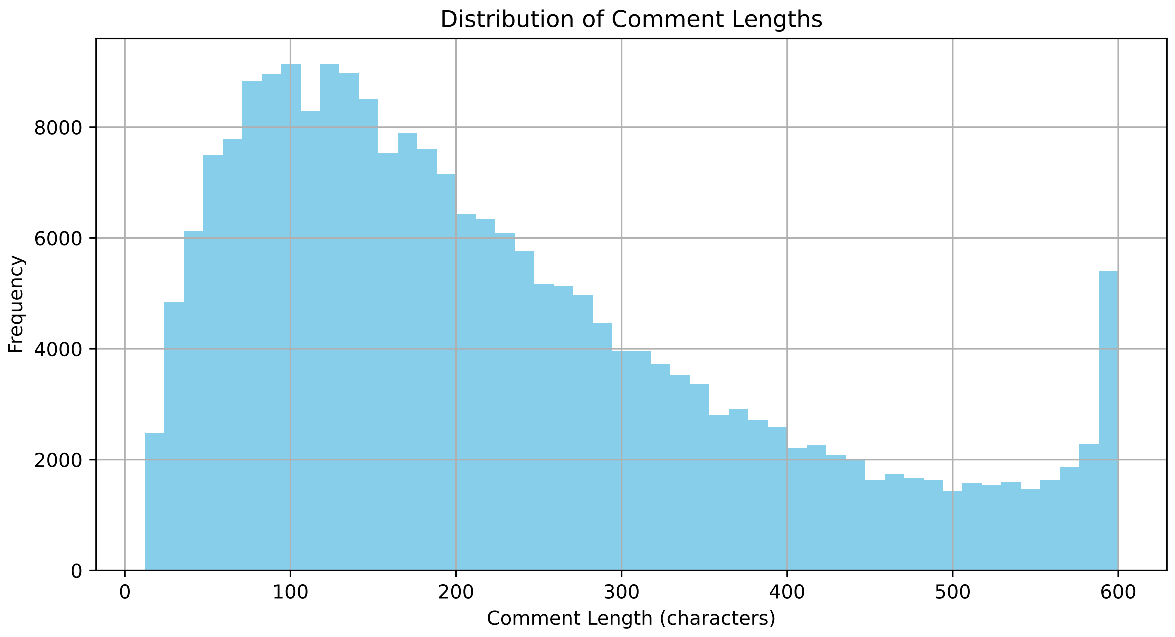
```
# Categorise votes as 2 = upvote and 3 = downvote
vote_types = {2: 'Upvotes', 3: 'Downvotes'}

# Filter votes_df to just those types
votes_filtered =
 ↪  votes_df[votes_df['VoteTypeId'].isin(vote_types.keys())].copy()
votes_filtered['VoteType'] = votes_filtered['VoteTypeId'].map(vote_types)

# Group by month and vote type, and count votes
votes_timeline =
 ↪  votes_filtered.groupby([votes_filtered['CreationDate'].dt.to_period('M'),
 ↪  'VoteType']).size().unstack(fill_value=0)

# Plot time series of vote by type
votes_timeline.plot(kind='line', figsize=(12,6), marker='o', title='Votes
 ↪  Over Time by Type')
plt.xlabel('Month')
plt.ylabel('Number of Votes')
plt.grid(True)
plt.show()
```



In this seventh visualisation we can see each type of vote (upvotes and downvotes) over the entire data collection period. We can see that upvotes are a lot more popular than downvotes, and followed a similar pattern to that seen in the previous plot showing the number of posts.

Visualisation 8: Distribution of Comment Lengths

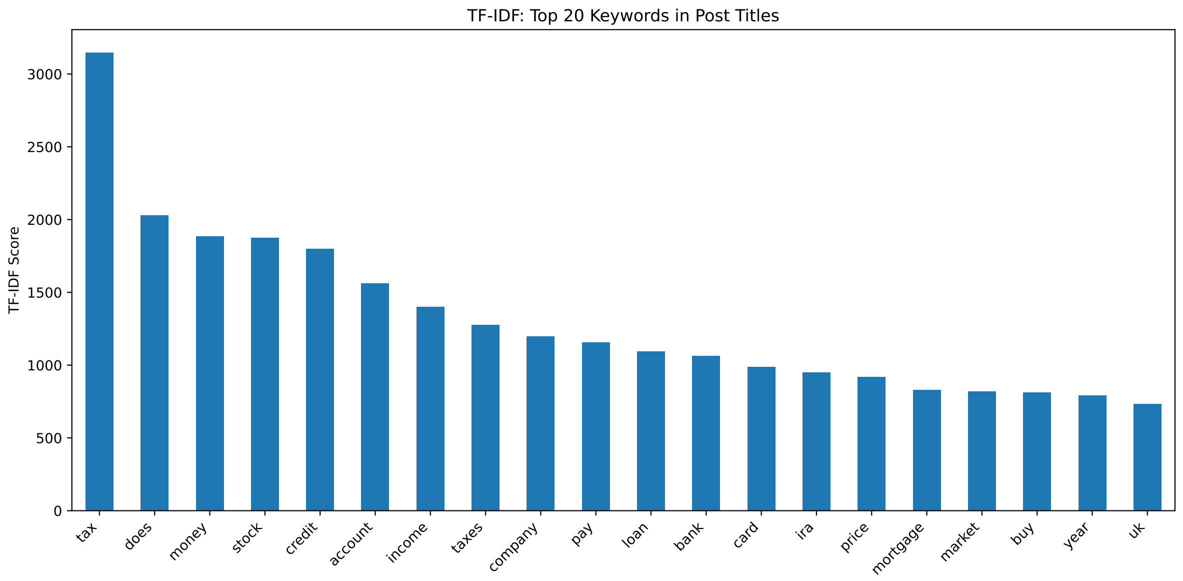
```
# New length column to store number of characters in each comment
comments_df['Length'] = comments_df['Text'].str.len()

# Histogram of comment lengths using 50 bins to show frequency of each length
comments_df['Length'].hist(bins=50, figsize=(10, 5), color='skyblue')
plt.title('Distribution of Comment Lengths')
plt.xlabel('Comment Length (characters)')
plt.ylabel('Frequency')
plt.show()
```



In this eighth visualisation we can see the distribution of comment lengths, by number of characters. This gives us a good idea of what the average comment length is, with the highest frequency being around 100 characters, dropping off consistently towards 600.

Visualisation 9: TF-IDF Analysis on Post Titles

```
# Replace missing post titles with an empty string
posts_df['Title'] = posts_df['Title'].fillna('')

# Initialise the TF-IDF vectoriser with a max of 20 terms, using English stop
 ↪  words
vectorizer = TfidfVectorizer(max_features=20, stop_words='english')

# Apply TF-IDF to the post titles
tfidf_matrix = vectorizer.fit_transform(posts_df['Title'])

# Sum TF-IDF scores across all documents for each term
tfidf_scores = pd.Series(tfidf_matrix.sum(axis=0).A1,
 ↪  index=vectorizer.get_feature_names_out()).sort_values(ascending=False)

# Plot a bar chart of the top 20 TF-IDF keywords
tfidf_scores.plot(kind='bar', figsize=(12,6), title='TF-IDF: Top 20 Keywords
 ↪  in Post Titles')
plt.ylabel('TF-IDF Score')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



In this ninth visualisation, we can see the top 20 keywords extracted from post titles using TF-IDF (Term Frequency–Inverse Document Frequency), which highlights terms that are important within this dataset. This gives us an overview of the key topics and terminology that frequently appear in post titles, after filtering out common English stop words. It allows us to quickly identify the most distinctive words driving discussion on the Personal Finance & Money StackExchange site.

Insights from the Analysis

We have been able to extract meaningful data from the text information recorded in the Pandas dataframes. Each plot was paired with an explanation below, and some of the most notable insights are:

- "united-states" and "taxes" are the most popular tags.
- "united-states" and "taxes" often appear as pairs of tags, in the same post.
- The top users have thousands of posts, with number 1 being over 3000.
- Reputation generally increases as the number of posts increases for any given user.
- Tax is mentioned in almost 4000 posts, with loans mentioned in over 1000.
- The number of posts per month, as well as the number of posts mentioning tax, was highest in the 2015-2019 period.
- Upvotes are much more popular than downvotes, peaking around 2019.
- The most frequent comment length is around 100 characters.
- The TF-IDF analysis of post titles highlights 'tax' as the top keyword, indicating its high relevance in community discussions (consistent with its high frequency count from earlier analysis)

Conclusion

This project demonstrates my ability to work with unstructured data formats, perform data cleansing and transformation, and conduct exploratory text analysis. The project also reflects good data ethics, as sensitive user information is treated responsibly.

By converting the original XML data to CSV and conducting exploratory analysis with Pandas and visualisation libraries, I gained valuable experience in data wrangling and text-based EDA.

Further improvements could include deeper NLP analysis (such as sentiment analysis) and integrating interactive dashboards. Sentiment analysis could reveal shifts in community mood regarding financial topics, while interactive dashboards would allow users to explore specific trends or filter data dynamically.