Overview of API Notes,
Blank Page,
Actual Notes & Helpful Links

🔍 Project Overview: AI-Powered FEC Donor Data Analysis

📁 Dataset Description

The **Federal Election Commission (FEC)** maintains structured datasets on campaign contributions to ensure transparency and public accountability. Each record typically includes:

- CMTE_ID: Unique committee identifier
- AMNDT_IND: Amendment indicator (new, changed, or terminated)
- RPT_TP: Report type
- TRANSACTION_PGI: Primary/General Indicator for election cycle
- ENTITY_TP: Entity type (individual, PAC, etc.)
- Donor fields: name, city, state, zip code, employer, occupation
- TRANSACTION_DT: Date of donation
- TRANSACTION_AMT: Donation amount
- TRAN_ID: Unique transaction ID
- Memo fields: text/comments indicating special conditions or reattributions

This consistent schema makes the dataset ideal for both **manual auditing** and **automated machine learning** pipelines.

🧠 AI & NLP Integration

The structured and detailed nature of the FEC dataset lends itself to various AI applications, especially:

🔷 Pattern Recognition

- Identify high-frequency or repeat donors
- Detect anomalies or irregular contributions
- Forecast donation surges in relation to campaign events

🔷 Supervised Learning

- Predict future donation amounts
- Classify donors by influence or political leanings

🔷 Natural Language Processing (NLP)

- Analyze memo fields for reattribution, refund notes, or earmarking
- Resolve name ambiguities (e.g., “Bill Gates” vs. “William Gates”)

🚧 Current Challenges: Name Disambiguation

One of the biggest issues we're facing is **name recognition**. While humans understand that "Bill" is a nickname for "William," AI models—especially out-of-the-box large language models—struggle with this.

Example Problem:

Two contributions are made by:

- **Bill Gates**
- **William H. Gates**

AI might treat them as two different people. Our solution is to develop a **disambiguation layer** using:

- Fuzzy matching (e.g., Levenshtein distance)
- Entity resolution models
- Unique transaction patterns

Code-based searches using Python have been far more accurate than natural language prompts, making programmatic filtering a critical component.

🖥️ Getting Started with Jupyter Notebook (macOS)

Step-by-Step:

1. Open Terminal and type:

```
jupyter notebook
```

2. In your browser:
   - Click **File** → **New Notebook**
   - Choose **Python (percentage format)**
3. Manually upload your FEC dataset:
   - Click **Upload** (top-right corner)
   - Select your .csv or .xlsx file
   - Confirm upload

Or, use a **file path method**:

```
import pandas as pd
df = pd.read_csv("yourfile.csv")
df.head()
```

Use sequential and intuitive naming conventions (e.g., FEC_2024_Q1.csv) for scalability.

🔗 Downloading FEC Data Programmatically

Rather than manually downloading datasets each time, we can **scrape the FEC site** or **use their API**.

✅ Manual Web Scraping (Using Python)

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
```

```
import zipfile
import io
url = "https://www.fec.gov/data/browse-data/?tab=bulk-data"
headers = {"User-Agent": "Mozilla/5.0"}
response = requests.get(url, headers=headers)
soup = BeautifulSoup(response.text, "html.parser")
links = []
for link in soup.find_all("a", href=True):
    href = link["href"]
    if "indiv" in href and href.endswith(".zip"):
        full_link = "https://www.fec.gov" + href if href.startswith("/") else href
        links.append(full_link)
print("Found files:")
for l in links:
    print(l)
dfs = []
for file_url in links:
    print(f"Downloading: {file_url}")
    r = requests.get(file_url)
    z = zipfile.ZipFile(io.BytesIO(r.content))
    for filename in z.namelist():
        print(f"Extracting: {filename}")
        with z.open(filename) as f:
            df = pd.read_csv(f, sep="|", low_memory=False)
            dfs.append(df)
            break
combined_df = pd.concat(dfs, ignore_index=True)
combined_df.head()
```

🔐 Using the FEC API

Why use the API?

- More efficient and scalable
- Access specific fields (e.g., by candidate, ZIP code, amount)
- Reduced need to clean and merge massive CSVs

Steps:

1. Sign up for an API key at [FEC Developer Portal]
2. Use this key to make authenticated requests

Example:

```
import requests
api_key = "your_api_key"
endpoint = "https://api.open.fec.gov/v1/schedules/schedule_a/"
params = {
    "api_key": api_key,
    "contributor_name": "William Gates",
```

```
    "per_page": 50,
    "two_year_transaction_period": 2024
}
response = requests.get(endpoint, params=params)
data = response.json()
for item in data["results"]:
    print(item["contributor_name"], item["contribution_receipt_amount"])
```

🎯 Future Steps

1. **Train LLMs** using fuzzy name matching datasets and real-world donor disambiguation examples
2. Develop an **interactive dashboard** for journalists, researchers, and watchdog groups
3. Integrate **anomaly detection algorithms** for fraud analysis
4. Implement **entity resolution** to cluster donations by real-world individuals and organizations

🤖 Want to Include Sample AI Models?

Yes! If you're interested, I can add:

- Specific ML models (e.g., XGBoost for prediction, DBSCAN for anomaly detection)
- Entity-matching libraries (like dedupe or recordlinkage)
- Sample NLP pipelines for memo field analysis

Let me know, and I'll include those next.
Would you like me to turn this into a downloadable .ipynb notebook too?

Untitled.ipynb – Note to self this is the file with the successful code in it.

file description page:
The Federal Election Commission (FEC) organizes donor data in a structured format to make campaign finance contributions transparent and traceable. Each record begins with a unique **committee ID (CMTE_ID)**, identifying which political group received the donation. The **amendment indicator (AMNDT_IND)** shows whether the record is new, changed, or part of a terminated report. **Report type (RPT_TP)** and **election code (TRANSACTION_PGI)** explain the kind of report and which election cycle the contribution applies to—such as a general or primary election. **Donor information** includes the name, city, state, employer, occupation, and zip code. The **transaction date (TRANSACTION_DT)** and **amount (TRANSACTION_AMT)** show when the donation was made and for how much. For electronic filings, **TRAN_ID** is a unique ID used to track each donation entry. The dataset also includes technical codes like **entity type (ENTITY_TP)**—indicating if the donor is an individual, PAC, or organization—and **memo codes**, which flag special notes about the contribution (e.g., if it's part of a joint donation or previously reported). This standardized format allows researchers, journalists, and the public to analyze political contributions across campaigns and election cycles with consistency and detail.

The structured format of the FEC's individual contributions dataset makes it highly suitable for analysis by artificial intelligence (AI) systems. Each field—such as donation amount, donor occupation, employer, and geographic location—provides a clear data point that AI can use to detect patterns, trends, and anomalies across campaigns and election cycles. For example, machine learning models can analyze donor behavior to predict future contributions, identify influential donor groups, or flag suspicious donation patterns that may indicate fraud or coordinated efforts. Natural language processing (NLP) techniques can even interpret memo text fields to extract context about reattributions or earmarked contributions. The consistent use of unique identifiers, such as committee IDs and transaction IDs, ensures data integrity and allows AI to track relationships between donors and political entities over time. This level of granularity is critical for building reliable, predictive models that inform policy decisions, transparency initiatives, or campaign strategy. By feeding this organized data into AI systems, analysts can generate actionable insights far more efficiently than through manual review alone.

But currently using A.I. Techniques like downloadable large language models to help the technology understand the data using Natural language processing (NLP) is the next step. One issue we are currently facing is name recognition. Over all we have publicly accessible FEC donor files, which contain Hundreds of thousands of donations. A.I. currently has problems understanding that Bill is William. Thus when prompting it we have found the best results using code instead of Human language when finding specific information on a single contributor, since the Python commands are the most straightforward. We are hoping to introduce large language model technology into our code that we could train the A.I. to recognise patterns in donations, the same donor ID given to Bill gates as William Gates, so it will know , not to only search for the same, but to clarify it's search and find information from the same person under different names.

The detailed and consistently structured nature of the FEC's individual contribution dataset provides an ideal foundation for artificial intelligence (AI) applications, particularly in the realms of supervised learning and pattern recognition. The schema includes categorical variables (e.g., ENTITY_TP, STATE, OCCUPATION), numerical variables (e.g., TRANSACTION_AMT), temporal data (TRANSACTION_DT), and unique identifiers (TRAN_ID, CMTE_ID)—all essential components for training high-quality machine learning models. Using techniques such as regression analysis or classification algorithms, AI can learn to predict donation behaviors, identify high-frequency contributors, or even detect outlier transactions potentially indicative of campaign finance violations. The inclusion of memo codes and memo text fields further supports the use of Natural Language Processing (NLP) to extract qualitative insights from semi-structured data, such as the reason behind a reattribution or a refunded donation. Furthermore, with time-series modeling, AI can forecast donation surges in relation to campaign events or policy announcements. The well-normalized format also simplifies preprocessing and feature engineering, minimizing data cleaning efforts and enabling efficient integration with relational databases or big data platforms. This level of granularity and consistency makes the FEC dataset not only readable by machines but also highly actionable for building transparent, accountable, and intelligent political finance analysis tools.
Would you like me to pair this with examples of specific AI algorithms or tools that could be used?

But currently using A.I. Techniques like downloadable large language models to help the technology understand the data using Natural language processing (NLP) is the next step. One issue we are currently facing is name recognition. Over all we have publicly accessible FEC donor files, which contain Hundreds of thousands of donations. A.I. currently has problems understanding that Bill is William. Thus when prompting it we have found the best results using code instead of Human language when finding specific information on a single contributor, since the Python commands are the most straightforward. We are hoping to introduce large language model technology into our code that we could train the A.I. to recognise patterns in donations, the same donor ID given to Bill gates as William Gates, so it will know , not to only search for the same, but to clarify it's search and find information from the same person under different names.

POI 487

Open Terminal on Mac

Type into terminal Jupyter Notebook

Jupter should open in your browser,

Then press File<

Press new Notebook

Press new Phython text note book with percentage format

A new tab will open with the proper format to place code.

First you nee dto upload the FEC data sets to this platform for the code to apply to the data.

I suggest that you dont manually do this process, but instead use a file pathway “written out in code” t access the files. This is because, if in the future you wanted files to automatically be uploaded this method would be earlier to adapt to that preference instead of manialy commanding the computer.

Bellow you will see step by step on how to manually uploaded the data.

The data must be a CSV or Exex File

You should see a file explorer-like interface.

2. **Navigate to the folder where you want to upload your file.**
3. **Click the “Upload” button (top right corner).**
4. **Select your file** from your computer (CSV, Excel, JSON, etc.).
5. **Click “Upload” again next to the file name** (in Jupyter, after it shows up).

To use a pathway : Use this format >>>

import pandas as pd

df = pd.read_csv("yourfile.csv")

df.head()

You want to name the file something that you will remember and that follows a sequential order.

So to do this manually again, you would need to upload it through this pathway with the exact name of the file you manually downloaded from the FEC website and are now trying to upload to the interface (AKA jupiter notebook).

Instead we are going to try to figure out a way to have the computer using A.I. strategies to data scrape from the FEC website. How can we do this?

We can first ask Chat GPT if there is code that would make it possible to, insert (the web link to the data sets instead of the downloaded file from the computer).

Instead of this :

```
import pandas as pd

df = pd.read_csv("yourfile.csv")
df.head()
```

We can do this :

```
import pandas as pd

url = "https://example.com/data.csv"
df = pd.read_csv(url)

df.head()  # Show the first few rows
```

____

BEllow this is for a Ziped format:

```
import pandas as pd
import zipfile
import requests
import io

url = "https://example.com/data.zip"
r = requests.get(url)
z = zipfile.ZipFile(io.BytesIO(r.content))
df = pd.read_csv(z.open("data.csv"))

df.head()
```

This is the Step by Step process>>

✅ **What We'll Do:**

1. **Scrape the dropdown** for all file links under "Contributions by Individuals."
2. **Download each file** programmatically (they're .zip files).
3. **Unzip and read the data** (usually .txt or .csv).
4. **Load into pandas DataFrames** for analysis in Jupyter.

>>>

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import zipfile
import io

# Step 1: Go to the page and get HTML
url = "https://www.fec.gov/data/browse-data/?tab=bulk-data"
headers = {"User-Agent": "Mozilla/5.0"}
response = requests.get(url, headers=headers)

# Step 2: Parse HTML
soup = BeautifulSoup(response.text, "html.parser")

# Step 3: Find all "Contributions by Individuals" zip file links
# We'll scan for links ending with .zip
links = []
for link in soup.find_all("a", href=True):
    href = link["href"]
    if "indiv" in href and href.endswith(".zip"):
        full_link = "https://www.fec.gov" + href if href.startswith("/") else href
        links.append(full_link)

# Optional: Show all found links
print("Found files:")
for l in links:
    print(l)

# Step 4: Download and extract the first file as example
# (Can loop through all if needed)

dfs = []  # Store DataFrames here
```

```
for file_url in links:
    print(f"Downloading: {file_url}")
    r = requests.get(file_url)
    z = zipfile.ZipFile(io.BytesIO(r.content))

    for filename in z.namelist():
        print(f"Extracting and reading: {filename}")
        with z.open(filename) as f:
            # Files are usually tab-separated
            df = pd.read_csv(f, sep="|", low_memory=False)
            dfs.append(df)
            break  # Remove this break if you want to read all files in the zip

# Example: Concatenate or just look at one
combined_df = pd.concat(dfs, ignore_index=True)
combined_df.head()
```

So we need to do it through API because Which is more efficient, because there are restrictions:

So first we need to register our API key with the FEC itself: a unique identifier, like a password, used to authenticate and authorize applications or users to access an API .

Then we Copy my API key: which will be used to authenticate my request for the data.

So i used this website:

https://api.open.fec.gov/developers/ which is the FEC developer portal, It's kinda long and wordy, but it instructs me to:... (for Bulk data downloads).

" If you are interested in individual donors, check out contributor information in the `/schedule_a/` endpoints."

" If you would like to use the FEC's API programmatically, you can sign up for your own API key using our form you can still try out our API without an API key by using the web interface and using DEMO_KEY. Note that whe openFEC API you are subject to the Terms of Service and Acceptable Use policy. "

“ Signing up for an API key will enable you to place up to 1,000 calls an hour. Each call is limited to 100 results per page. You can email questions, comments or a request to get a key for 7,200 calls an hour (120 calls per minute) to APIinfo@fec.gov. You can also ask questions and discuss the data in a community led group. “
This is the Group on Google Email which i joined, so far it seems like people are looking for data:https://groups.google.com/g/fec-data

Here is some information>> The model definitions and schema are available at /swagger. This is useful for making wrappers and exploring the data.

/swagger, Will be helpful in creating a Wrapper which In data scraping, wrappers are essential for transforming extracted data into a usable format. They act as a bridge between the unstructured data from web pages and a structured, relational database or file format. Essentially, they help organize and process the scraped data for efficient analysis and utilization.

Wrappers can also perform data transformations, such as converting text to numeric values, cleaning data, or standardizing formats.

A wrapper allows us to use code compounds like this:
fec.candidates.get(candidate_id='P00003392')

Chat GPT said 👍
🧠 **How You Can Use It in Jupyter:**
Once you’ve installed the wrapper (if they provide it as a package or as Python files), you can:

1. **Import the wrapper into your notebook.**
2. **Use it to fetch FEC data** without manual requests.
3. **Build your scraper on top of it** – use it to loop through candidates, committees, or contributors, and store the info in DataFrames.

📦 **Example Flow (After Installing the Wrapper):**

```
from fec_wrapper import FECClient  # Assuming that's what it's called

fec = FECClient(api_key='YOUR_API_KEY')

# Get individual contributions

contributions = fec.schedule_a.get(contributor_state='NY', per_page=100)

# Convert to DataFrame

import pandas as pd
```

df = pd.DataFrame(contributions['results'])

df.head()

—-------------

This is a snipid of the Wrapper

…{"definitions":{"AuditCandidateSearch":{"properties":{"id":{"type":"string"},"name":{"type":"string"}},"type":"object"},"AuditCandidateSearchList":{"properties":{"results":{"items":{"$ref":"#/definitions/AuditCandidateSearch"},"type":"array"}},"type":"object"},"AuditCase":{"properties":{"audit_case_id":{"type":"string"},"audit_id":{"type":"integer"},"candidate_id":{"type":"string"},"candidate_name":{"type":"string"},"committee_description":{"type":"string"},"committee_designation":{"type":"string"},"committee_id":{"type":"string"},"committee_name":{"type":"string"},"committee_type":{"type":"string"},"cycle":{"type":"integer"},"far_release_date":{"format":"date","type":"string"},"link_to_report":{"type":"string"},"primary_category_list":{"items":{"$ref":"#/definitions/AuditCaseCategoryRelation"},"type":"array"}},"type":"object"},"AuditCaseCategoryRelation":{"properties":{"primary_category_id":{"type":"string"},"primary_category_name":{"type":"string"},"sub_category_list":{"items":

Awesome — thanks for sharing that snippet! What you've got here is **part of an OpenAPI (Swagger) specification**. This kind of document defines how an API works, including:

- The **data structures** (like AuditCase, AuditCandidateSearch, etc.)
- The **response formats**
- The **types of data** (e.g., strings, integers, arrays)
- Relationships between different data components

🔍 What This Snippet Shows

From the snippet, you can tell:

- The **API provides audit case data** (e.g., AuditCase, AuditCandidateSearch)
- Each audit case includes things like:
  - Candidate ID and name
  - Committee info (ID, name, type)
  - Cycle and release dates
  - Link to the report
  - Category/subcategory data

These are the **data types** you'll get when querying certain FEC API endpoints (like /audit-case/).

✅ How This Is Helpful to You

This schema lets you:

- Understand **what fields you can extract**
- Know the **structure** of the API's response before writing code
- Build a data scraper that knows what to expect (e.g., you can directly parse committee_name or link_to_report)

🛠️ How to Use This in Jupyter

Once you know the structure from this schema, you can:

1. Use the **FEC API endpoint** that returns audit case data (e.g., /audit-case/)
2. Pull the data into Jupyter using requests
3. Store it as a **pandas DataFrame**
4. Use the field names (from the schema) to access data like candidate names, committees, or audit categories

📦 Example Jupyter Code (Using the Schema)

```
import requests
import pandas as pd
API_KEY = 'your_api_key'
BASE_URL = 'https://api.open.fec.gov/v1/audit-case/'
params = {
    'api_key': API_KEY,
    'per_page': 50,
    'page': 1,
    'cycle': 2022  # example cycle
}
response = requests.get(BASE_URL, params=params)
data = response.json()
# Grab just the results
results = data['results']
# Load into DataFrame
df = pd.DataFrame(results)
df[['candidate_name', 'committee_name', 'link_to_report']].head()
```

🚀 Next Steps

- Let me know if the FEC gave you this schema in a full file (like a .json or .yaml)
- I can help you create a Python wrapper or plug this into openapi-generator or Swagger Codegen to auto-generate Python client code
- We can also build functions to filter data by candidate, cycle, or committee

Would you like me to show how to turn this into a working API client or wrapper script for your project?

So i must create a API key for this to work.
So i applied for the APIs and they asked for an explanation of use:

This is what i put:
I am a university student working on a data analysis and research project focused on campaign finance transparency. I plan to use the FEC API, particularly the /audit-case/ and related endpoints, to retrieve structured audit case data. My goal is to build a Jupyter Notebook-based tool that automatically fetches, parses, and analyzes public data such as candidate and committee information, financial cycles, and audit outcomes. This will involve developing a Python-based wrapper to streamline data collection, followed by analysis using pandas. The OpenAPI specification provided by the FEC will guide how I structure requests and interpret responses. This project is strictly educational and research-based.

This is the APIs key i received: `HYpWWe3uT0R5n5DDNPbXh0eGlGxsnsNIGUz3fgsX`

"You can start using this key to make web service requests by referring to the relevant agency's API documentation. This API key is for your use and should not be shared.
"

✅ **Example: Pulling Audit Case Data Using Your API Key**

```
import requests
import pandas as pd

# Insert your FEC API key here
API_KEY = 'your_api_key_here'

# Define the endpoint URL
BASE_URL = 'https://api.open.fec.gov/v1/audit-case/'

# Set your parameters
params = {
    'api_key': API_KEY,
    'per_page': 50,       # Number of results per page
    'page': 1,            # Start at page 1
    'cycle': 2022         # You can change this to another election 
cycle
}

# Make the GET request
response = requests.get(BASE_URL, params=params)
data = response.json()

# Extract the results
```

```
results = data.get('results', [])

# Convert to a DataFrame
df = pd.DataFrame(results)

# Show some important fields
df[['candidate_name', 'committee_name', 'link_to_report']].head()
```

🔄 What You Can Do Next

- Try other endpoints, like:
  - /schedules/schedule_a/ → Individual contributions
  - /committee/ → Committee-level data
  - /candidate/ → Candidate-specific info
- Loop through pages to get more data:

```
all_results = []
for page in range(1, 6):  # Adjust range for more pages
    params['page'] = page
    response = requests.get(BASE_URL, params=params)
    page_data = response.json().get('results', [])
    all_results.extend(page_data)

df = pd.DataFrame(all_results)
```

- Save your data:

```
df.to_csv('audit_cases.csv', index=False)
```

So now we take the URL for the Donor data :
https://www.fec.gov/data/browse-data/?tab=bulk-data

Since we are doing the method with API, we will go about it this way :

Add this which is pagination to scrape more than one page. It also can filter by year, candidate , zip code, ect… also you can save the data to a CSV file.: df.to_csv('fec_donors_ny.csv', index=False)

- **Visualize top donors** or contribution trends using matplotlib or seaborn.

So before we can use CHAT GPT to make a prompt we must give it think information.

- What **specific data** do you want to extract? (e.g., donor name, amount, date, employer, etc.)
- Any **filters** you want to apply? (e.g., state, candidate, year, amount greater than $500, etc.)
- How many **pages** or how much **data** do you want to pull?
- Do you want to **save it to a file** like CSV?
- Any **analysis or visualization** you want to do afterward?

Once you send me that, I'll write a full Jupyter-ready script for you.

!pip install requests pandas is a compound so you can use pandas