**Introduction**:

Importance of 990 forms

Value for our economy: effects governance, financial performance, and social impact.

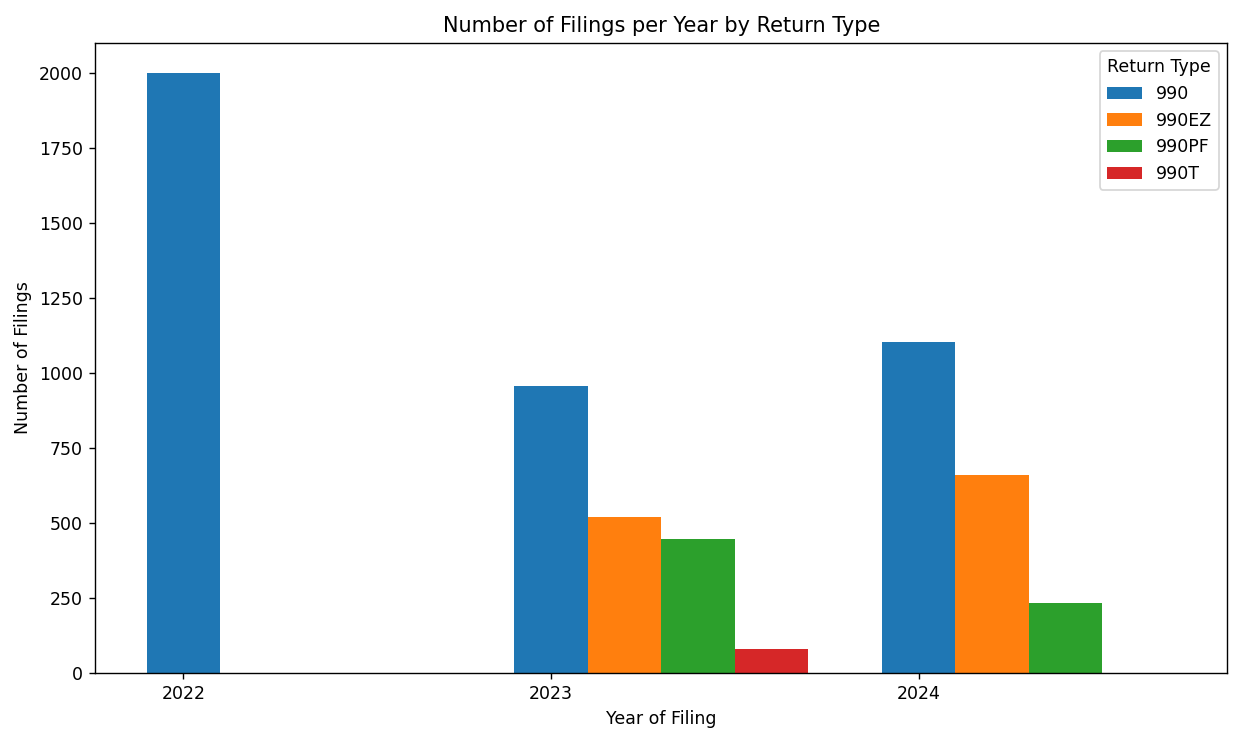
Diagnostic analytics, to see if there are any insufficiencies or benefits that could be yielded by organizations via one 990 form over another.

I am not going to focus on frequency of filing, but more about variety, and why one is chosen over the other, and if there are any issues with that economically, politically, and socially.

Someone has to foot the bill at the end of the day

Any space for efficiencies like format conversion from XML to another file to make filing and data analysis easier?

Previously, there was research into the benefits of XML files and the broader access to these files for public use, especially for IRS data, which was previously only available in PDF format (unstructured data) (Barreto et al., 2019). Researching, manipulating, and even downloading this data can be challenging due to its complex structure. The process of handling large zipped, hierarchical XML files (3D or 4D) with millions of filings can be time-consuming, particularly when trying to process, load, or delete them if they are accidentally downloaded. A notable advancement for 2016 to 2021 data was made by Abu-Khadra et al. (2023), who successfully converted XML files into flat 2D or 1D Excel files using SQL. While XML files and other limitations may hinder effective nonprofit research, they do not obstruct access to the millions of records in Form 990 filings. Today, IRS Form 990 data is more accessible than ever before, but the website still only provides XML files, and there is no public access to the SQL tools that could convert these files into more user-friendly formats like CSV or Excel.



*\*Sample size per year was 2,000 filings.*

*\*They were all e-file*

Used columns: SUB\_DATE and RETURN\_TYPE

Modified SUB-DATE to be a full date in the YEAR-MONTH-DAY format (XXXX-XX-XX)

* Python did not accept year alone (i.e 2022, 2023, and 2024)

|  |
| --- |
| Instructions per Abu-Khadra et al. (2023),  The following steps describe the shredding process:  1. Create a table in SQL server using the following code: CREATETABLEdbo.XMLFilesTable ( Id INT IDENTITY PRIMARYKEY, FileName VARCHAR(100), XMLDataXML, LoadedDateTime DATETIME ) GO  2. Download all XML compressed folders from the IRS repository and then extract them to a predefined loca tion. The following directory path was used in the included code. If the storage folder is different, please update the storage path in the code. F:\FinancialResearch\IRS\2017\Files\  3. Runthe code.  4. The code will create a “MasterFinal” virtual table. Verify the outcome.  **To export the dataset to Excel, please follow these steps**:  1. Open anew Excel sheet.  2. Choose the data from the menu bar and click on “Get Data,” then select “From Database” and “From SQL Server Database.”  3. A dialogue box titled “SQL Server Database” will appear. In the field titled Server, enter the IP address for your SQL server.  4. Anewdialogue box titled “Navigator” will appear. Choose your SQL folder and the “MasterFinal” Table.  5. At the bottom right side of the screen, click “load.”  6. You can convert the Excel file to CSV by using the “Save As” menu and choosing the “CSV Comma Delimited” file type. |

**References**:

Abu-Khadra, H., & Olsen, D. (2023). Towards Automating Shredding Nonprofit XML Files The Case of IRS

Form 990 Data. *Journal of Information Systems*, *37*(1), 169–188. EBSCOhost.

https://doi.org/10.2308/isys-2022-031

Barreto, H., & Villinski, M. T. (2019). Accessing IRS Form 990 Data with Excel. *Indiana University Press*, *2*(2),

95–110. ProQuest. https://doi.org/10.2979/phileduc.2.2.05

IRS. (2019, July 24). *Form 990 series downloads | Internal Revenue Service*. Irs.gov; IRS.

https://www.irs.gov/charities-non-profits/form-990-series-downloads

|  |
| --- |
| Appendix: |
| import pandas as pd  import matplotlib.pyplot as plt  # Load the dataset  file\_path = r'C:\Users\ \*\*\*'  data = pd.read\_csv(file\_path)  # Check the first few rows  print(data.head())  # Handle the SUB\_DATE column to extract the year  # Try to convert SUB\_DATE to a datetime, if it fails (e.g., it is just a year), we will handle it separately  data['SUB\_DATE'] = data['SUB\_DATE'].astype(str)  # Ensure it's treated as a string  # If the date is in a full date format like "01/01/2024", we extract the year  # If it's just a year (e.g., "2024"), we directly use it as the year  data['YEAR'] = pd.to\_datetime(data['SUB\_DATE'], errors='coerce', format='%m/%d/%Y').dt.year  # For rows where SUB\_DATE is just a year, handle them  data['YEAR'] = data['YEAR'].fillna(data['SUB\_DATE'].apply(lambda x: int(x) if x.isdigit() else None))  # Check for any missing or NaN years  print(data[data['YEAR'].isna()])  # Group by YEAR and RETURN\_TYPE and count the number of filings  grouped\_df = data.groupby(['YEAR', 'RETURN\_TYPE']).size().reset\_index(name='COUNT')  # Print the grouped data to check counts  print(grouped\_df)  # Plotting the grouped data  plt.figure(figsize=(10, 6))  # Plot bars for each return type  for return\_type in grouped\_df['RETURN\_TYPE'].unique():      data\_to\_plot = grouped\_df[grouped\_df['RETURN\_TYPE'] == return\_type]      plt.bar(data\_to\_plot['YEAR'] + (0.2 \* (return\_type == '990EZ')) +              (0.4 \* (return\_type == '990PF')) + (0.6 \* (return\_type == '990T')),              data\_to\_plot['COUNT'], width=0.2, label=return\_type)  # Adding labels and title  plt.title('Number of Filings per Year by Return Type')  plt.xlabel('Year of Filing')  plt.ylabel('Number of Filings')  plt.xticks([2022, 2023, 2024])  # Set x-axis ticks to be the years 2022, 2023, 2024  plt.legend(title='Return Type')  # Show the plot  plt.tight\_layout()  plt.show() |