– Tax Data Project –
Development of an IRS 990 Database

May 8, 2022
Blacksburg, VA, 24061

Submitted to: Dr. Edward A. Fox
CS 4624 – Multimedia, Hypertext, and Information Access Capstone
Client: Dr. Florian Zach

Prepared by:
Kaleb Barker, Nicky Huynh, Jack Lacey, Long Nguyen

Table of Contents

Table of Figures

Table of Tables

Abstract

The Internal Revenue Service (IRS) provides a plethora of data related to tax-exempt organizations through the publication of IRS Form 990 tax filings in Extensible Markup Language (XML) format, hosted between their website and Amazon Web Services (AWS). These data sources possess filing data beginning in tax year 2012, and ending in the most recently filed and uploaded tax year of 2020. This defines the project's study window as 2012-2020. The primary goal of this project is to create a database of Form 990 filings to support research related to tourism offices and various other tax-exempt organizations. The primary challenge of this project is to process filings from all years within the study window and upload them to the database in a unified manner. The development of this database utilizes tools such as Jupyter Notebooks, SQLite, and various Python libraries for scraping, preprocessing, and analysis. Due to the number of different return types and the massive amount of data contained in the forms, understanding the forms in their standard format is incredibly challenging. Additionally, most documentation about 990 forms is oriented to accountants or tax experts who are well versed in financial jargon. This issue extends to the XML data files themselves, as many of the XML tags are heavily abbreviated, and cross referencing each of them with its corresponding location on Form 990 is a tedious and near impossible task. The solution to these problems lies in archiving the data but also having it accessible for use.

This project can be divided into four phases: scraping, preprocessing, uploading, and analysis. The project begins with scraping 990 filings from the two sources highlighted above. The next phase, preprocessing, involves creating a common schema and converting the XML files into Comma Separated Values (CSV) and JavaScript Object Notation (JSON) formats. This is the most difficult and lengthy phase of the project as it involves understanding the 990 filings to the greatest possible extent through both automated and manual processes. Next is the uploading phase, where the database is built and populated with the preprocessed data. Finally, queries can be made to the database for the analysis phase to extract interesting financial trends. This final phase allows the team to maximize its familiarity with the database and supports the development of extensive documentation and the users' guide that are included in the Users' Manual section. The result of this project comes in two forms: the aforementioned database, and a set of CSV data pertaining to the 990 filings of all tourism offices present in the XML data. The database is structured in order to maximize the breadth and depth of analysis that is made available to the project's client and other stakeholders. These other stakeholders include fellow researchers of tourism offices, and any other business researchers who may be concerned with the financial data of non-profits and tax-exempt organizations. The database contains tables that allow users to access specific data across an organization, or multiple organizations' Form 990 filings. These tables are complemented by overview data tables, allowing for users to locate specific organizations based on the type of business they carry out (such as tourism offices), rather than limiting users to querying based on Form 990 filing data. Finally, per the client's request, all tourism office data is separately outputted into a set of CSV files.

Introduction

The IRS stores a wealth of information regarding tax exempt organizations in various online records of 990 filings. The data used in this project was from the IRS website and an AWS S3 bucket, with years ranging from TY 2012 to TY 2020. These two data sources contain three different filing formats: standard Form 990, Form 990-EZ, and Form 990-PF [1, 2, 3, 6, 7]. Form 990 is the standard filing form for tax exempt organizations, while Form 990-PF is the 990 form filed by private foundations. Finally, Form 990-EZ is filed by tax exempt organizations with gross receipts of less than $200,000 and total assets of less than $500,000 in a given tax year. Some filings were paired with one or more from an assortment of attachments, or Schedule forms. These forms, labeled A through R, provide more specific information about the filing organization. Some examples of these forms include Schedule H, which discloses information regarding hospitals operated by the organization, and Schedule O, which allows for the providing of further details on the data contained in 990 and 990-EZ [1, 2, 4, 5]. Additionally, Schedule J includes relevant information regarding the compensation of an organization's officers. Amongst this wealth of data, the client is primarily interested in tourism offices, some of which are contained within file Form 990 while others file Form 990-EZ.

Objectives

The team's initial objectives were to parse all of the data contained in the union of AWS and IRS data across the years of study [6, 7]. After the data was parsed, the team would then use the parsed data to populate a SQL database, and would create a library of CSV and JSON files for all of the tourism office data. The team would then conduct a preliminary analysis to identify trends in the data for tourism offices and for various other exempt organizations. Throughout these phases of the project, the team was tasked with storing all raw and intermediate data within a shared Google Drive workspace.

Changes

After starting the project, the team ran into a few issues that resulted in changes to the aforementioned objectives. Some of the more specific details on these challenges are discussed in the Lessons Learned section. These issues led to changes to the scope and objectives of the project.

Originally, the client tasked the team with processing filings dating back to the start of the 2000s. However, due to the AWS data source clearing out earlier records, the team was not able to access records for TY 2011 and earlier [6]. Some other changes to the project objectives came about in order to narrow the scope of the project. Due to the preprocessing phase being much more tedious and time consuming than originally expected, the team and the client were concerned that the original set of objectives would not be feasible in the available time frame. As

a result, the parsing of the 990-PF forms and the processing of each of the schedule forms, except for Schedules O and J, was omitted from the scope of this project [3, 5]. The team excluded Form 990-PF filings due to the fact that tourism offices, the project's primary focus, would not be found in private foundation filings [3]. The team excluded the Schedule attachments due to the sheer amount of data they would contribute, despite most of the forms being irrelevant to tourism offices. That being said, the team still included Schedules O and J because they hold information directly related to the 990 and 990-EZ forms. This information typically came in the form of further details on the activities and expenses of an organization [1, 2, 5].

Finally, due to exceeding the storage capacity in the team and client's shared Google Drive workspace, the team did not end up storing all raw and intermediate data. The workspace reached capacity when only a few years worth of raw data were uploaded to the drive, and it therefore was infeasible to store the full raw or intermediate data across the study window. The workspace for sharing raw XML was moved to Advanced Research Computing (ARC) storage at Virginia Tech. The client decided that the original task of converting all 990 filings into CSV and JSON formats could be scrapped in favor of prioritizing tourism offices when converting 990 filings.

Final Objectives

The final objectives for the project are outlined below, as agreed upon by the team and client:

1) Parse all of the data contained in AWS and IRS for the study time frame (2012-2020), excluding 990-PF and all schedule attachments, other than Schedule O.
2) Design a database and populate it with 990 filing data.
3) Convert tourism office filing data into CSV files.
4) Conduct preliminary analysis on the database in order to identify trends in the financial data of tourism offices.

Client

The client for the project was Dr. Florian Zach, Ph.D. from the Howard Feiertag Department of Hospitality and Tourism Management within the Virginia Tech Pamplin College of Business. He and his colleagues are interested in gaining a better understanding of the financial dealings of tourism offices across the country. This database will serve to aid him and his colleagues in their research. Due to Dr. Zach's lack of a computing background, it is crucial that this project's database is easy to use, both in its design and rich documentation, which is present in the User's Manual.

Outline

In the remainder of this report, the team will discuss the requirements involved in the various phases of the project. Following this will be an outline of the high-level design considerations that went into each of the crucial objectives for the project. This discussion will then be complemented with a detail-oriented dive into the implementation of the programs that governed each of the primary steps outlined in the design sections. After breaking down the requirements, design, and implementation of the project, the team will address the criteria for the assessment and evaluation of the key project deliverables: the converted tourism office filings and the SQLite database.

These sections breaking down the essential details of the project are followed by manuals to support continued work on the project and continued research leveraging the project. The user's manual will serve to inform the client and other stakeholders of some of the ways they can use the data and tools associated with the project. Following this, the developer's manual will serve to inform future students or other developers of the techniques used by the team and how they could be applied to expanding upon the project. This report concludes with some lessons learned by the team along with the team's acknowledgements of the various individuals and resources that supported the project.

Requirements

This section of the report details the major requirements involved in each of the phases of the project. These requirements include the goals of each phase along with some of the data sources and programming libraries involved in each phase.

Deliverables

The client and team agreed on a set of key deliverables to complement the key objectives associated with this project. These deliverables are:

1) SQLite database containing 990 filing data from 2012 to 2020 and containing Exempt Organization Business Master File (EO BMF) data that was last updated 4/11/2022
2) Tourism offices' tax information converted into CSV files
3) Python code for the scraping, processing, database ingestion, and preliminary database analysis

Data Extraction

The 990 filing data used for this project was sourced from two different providers containing 990 tax filing data in XML format. The two data sources were from AWS and the IRS website, which contained data from TYs 2012-2020 for the standard 990 form, 990-EZ forms, and 990-PF forms [1, 2, 3, 6, 7]. As discussed above, the original requirement for this data involved scraping it all from the two sources and uploading all of the raw data files to a shared Google Drive workspace. However, this requirement was scrapped due to the Google Drive workspace reaching file capacity. Instead, the data was downloaded in pieces and worked with locally. All of the data for this project takes up dozens of gigabytes of hard drive space, and therefore it could not be stored all at once on any of the developers' machines.

Beyond the XML data, there were two other key resources that needed to be extracted for this project. The first of these is a set of annual index files which serve to complement the AWS data. As the AWS XML data is only available through scraping individual filings from web addresses, the index files serve to provide the file IDs that point to the AWS endpoints where each organization's filing data can be found. The other resource is the Exempt Organization Business Master File (EO BMF) [8]. The file contains information about the different groupings, overall financial data such as total asset amount and revenue amount, and locality for each organization, but does not contain actual 990 tax filing data for an organization. This file is updated each year with information on all of the tax-exempt organizations that filed a 990 form for a given tax year. In the context of this project, the file is used for the different grouping information under the fields Activity Code and National Taxonomy of Exempt Organization (NTEE) Code. Activity Code is a 9-digit code composed of three 3-digit numbers that correspond to different business activities. NTEE Code is a letter prefix, which relates to overall type of organization, and 2-digit number, which relates to a specific scope of organization within a type. With the activity and NTEE codes the Employer Identification Number (EIN) could be identified for every tourism office. Further information regarding how those fields were used is detailed in the Master File Tourism Office Filtering & Conversion Section.

Data Processing

The key challenge of this project involved handling the variability between data tags across the annually changing tax forms. Changes to the regulations set by the federal government and the IRS are reflected in changes to each year's tax forms. Not only this, but these changes can be even more stark when comparing the XML versions of the tax forms across the years of study. Specifically, a massive transition of data tag names occurred between TY 2012 and TY 2013, while more minute changes occurred between TY 2013 and all of the following years of study up to TY 2020. In order for the team to create a robust and evolving database for the client and his colleagues, the variable data tags across the years of study needed to be aligned into a unified schema to the greatest extent possible. This involved analyzing the tags that appear in given

years, and leveraging set theory to assess intersections and differences between those tags. The design considerations and software implementation of this tag alignment process are discussed in further detail in later sections of this report.

Database Creation

The creation of a 990 filings database was necessary in order to store the filing information of tax-exempt organizations. The team and client agreed upon using SQLite as the database framework. As discussed in the deliverables section above, the database needed to contain tax data for all 990 filings from 2012 to 2020 as well as the annual EO BMF files [8]. The process to create and populate the database was written in Python using Jupyter Notebooks.

Data Conversion

As discussed in the introduction, an original requirement of the project involved converting all 990 XML data into both CSV and JSON formats in order to improve the readability of all of the 990 data. However, there are hundreds of thousands of 990 filings for a given tax year. Due to the file size of the individual XML files, this resulted in each year of raw data requiring at least ten gigabytes of storage. This would similarly result in an even greater amount of storage required to store all 990 filings in both CSV and JSON format. As a result, this project requirement was changed through an agreement with the client. This agreement involved a reduction of the scope of this requirement: from requiring all filings to be converted, to simply requiring the conversion of tourism office filings. This requires the filtering of tourism offices using the EO BMF for a given year, pulling their data from the database, and subsequently using a program to convert the filing data into CSV and JSON formats and outputting to a workspace of the client's choosing [8].

Analysis & Documentation

Although the primary requirement of this project is the creation of a database of 990 filings, a crucial complementary requirement to the database involved performing introductory analysis and the development of documentation on the usage of the database. The primary stakeholders for this project were the client and his colleagues in the Department of Hospitality and Tourism Management, many of whom do not have a computing background. Due to this, it was essential that the database for this project was paired with rich documentation to assist the project's stakeholders in utilizing the database. This came in two forms: the team performing analysis on the database and the creation of a user guide. Preliminary analysis must be performed on the database in order for the team to gain a comprehensive understanding of the various analyses at the disposal of the client, and of any shortcomings of the database. Taking these steps helped guide the team in the creation of a user guide, which is included in a later section. This guide

detailed various examples of queries that can be performed on the database and various Python libraries that can be leveraged to perform these queries, including SQLite and Pandas.

Design

This section of the report details high level considerations regarding the design of the various phases of the project. These considerations were primarily concerned with the various file formats that the team had to handle, some libraries or modules that could be used to support the data processing, and the structure of the database.

Data Scraping

All 990 data is officially sourced from the IRS; however, the IRS allocated some of their filings to an Amazon S3 bucket called *irs-form-990*. This data was publicly available at the IRS990 endpoint through the AWS Open Data Registry [6]. The IRS discontinued updates to these filings beginning in 2022. All of the 990 electronic filings in AWS S3 can be downloaded through the standard process for downloading any S3 object: using either the AWS Command Line Interface (CLI) or any of the AWS Software Development Kits (SDKs) [9, 10]. This project utilized a Python SDK to scrape and download the files.

The other data source came directly from the IRS website [7]. The individual files were in the form of XML and can be downloaded in a series of ZIP files. The ZIP files were labeled on the website from the year 2015 to 2021; however, each year contained filings from the previous three tax years, making the actual range of 990 filings from TY 2012 to 2020. Downloading the data was a manual process, but upon downloading the data, this project made use of scripts to unzip many files at once. Each year contained anywhere from two to eight zip files, each containing hundreds of thousands of 990 filings. Due to the constraints associated with storing this many files, no more than one year of filings was unzipped at a time.

Database Creation

As specified in the project requirements, the database needed to contain all tax information from all EOs (exempt organizations) and the IRS master files (EO BMFs) [8]. The IRS website contained two different types of master files: one grouped by regions and another grouped by states [8]. Due to a discrepancy in the number of entries between the two file types, the team decided to store both in the database. A database table for these master files was created by performing a union operation on the two tables and removing the duplicate entries based on their unique EINs.

The 990 filings database needed to be organized into various subtables for an efficient and easy to query database design. One such table concerns the salaries and information regarding officers working for various EOs. As the organizations have varying numbers of officers, the team and client agreed to organize this information into a separate table, labeled “Officers”. These separate tables will be tied to the primary 990 filing data (in either the 990 or 990-EZ table) through the unique EIN possessed by each organization. Figure 1 shows the organizational breakdown of the database:

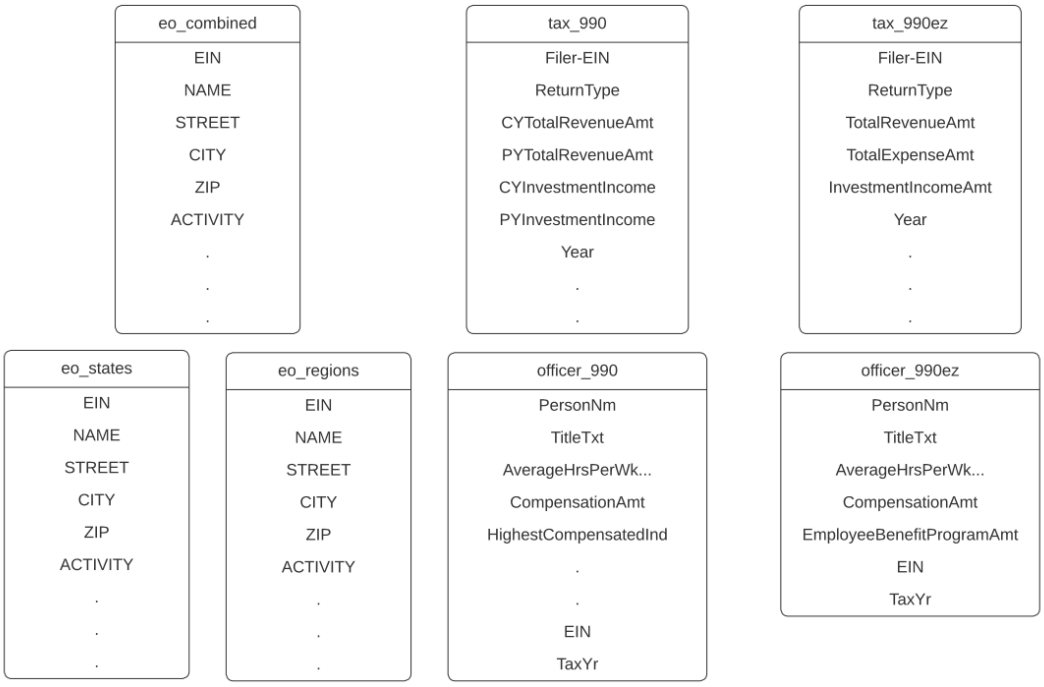


Figure 1: Schema tables within the EO (Exempt Organization) database.

Schema Alignment

Each type of 990 form ranged from 2012-2020. Each year had a difference in XML tags which must be accounted for. Based on the organization and filing year, a tag may only appear a handful of times for an organization and not at all for another organization. These differences in XML tags were the cause behind the principal challenge of the project: schema alignment. This task was made more difficult by the IRS not readily providing a schema for the various tax forms. However, the client had previously found a GitHub repository that contained a mostly complete schema for all file types and years until 2015 [11].

990-EZ

Although there were far fewer tags in Form 990-EZ compared to Form 990, there were still a substantial number of tags [1, 2]. The team prepared an estimate of the schema for a given year using a subset of files for each year. The innermost tags (which the team called the data tag)

containing the actual data were pooled together into a tag set for a given year. After extracting the tags from 100 files versus 1000 files in a year, the number of tags did not increase by a linear factor so the schema estimate was deemed to be sufficient. Using the estimate of the schema, the team manually compared the tags across all years to the actual inputs presented in the 990-EZ form for tax year 2013 for verification [2].

Main Section	***Box/Line***	***Subsection***	***Checkbox/Data Entry***	***2013 XML Tag***
Info	**A - Dates**	Start Month	Data Entry	TaxPeriodBeginDt (2013)
		End Month		TaxPeriodEndDt (2013)
		End Year		TaxPeriodEndDt (2013)
	B - Updates	Address Change	Checkbox	AddressChangeInd
		Name Change		NameChangeInd
		Initial Return		InitialReturnInd
		Final Return/Terminated		TerminatedReturnInd
		Amended Return		AmendedReturnInd
		Application Pending		ApplicationPendingInd
	C - Info	Name of Organization	Data Entry	BusinessNameLine1 and BusinessNameLine2
		Street Address/PO Box		AddressLine1
		Room/Suite		AddressLine2
		City		City
		State		State
		Zip		ZIPCode
	D	Employee ID Number (EIN)	Data Entry	EIN
	E	Telephone Number	Data Entry	PhoneNum
	F	Group Exemption Number	Data Entry	GroupExemptionNum
	G - Accounting	Cash	Checkbox	MethodOfAccountingCashInd
		Accrual		MethodOfAccountingIndAccrualInd
		Other	Data Entry	MethodOfAccountiingOtherDesc
	H	Check if the organization is not required to attach Schedule B	Checkbox	ScheduleBNotRequiredInd

	I	Website	Data Entry	WebsiteAddressTxt
	J - Exempt Status	501(c)(3)	Checkbox	Organization501c3Ind
		501(c)		Organization501cInd (2013)
		Enter Number: 501(c)(#)	Data Entry	typeOf501cOrganization
		4947(a)(1)	Checkbox	NA in 2013 (Only 2012)
		527		NA in 2013 (Only 2012)
	K - Form of Org.	Corporation	Checkbox	TypeOfOrganizationCorpInd
		Trust		TypeOfOrganizationTrustInd
		Association		TypeOfOrganizationCorpInd
		Other		TypeOfOrganizationOtherInd
		Other Entry	Data Entry	TypeOfOrganizationOtherDesc
	L	Gross Receipts	Data Entry	GrossReceiptsAmt

Table 1: Section of XML schema for 2013 990-EZ

This revealed a weakness in this approach: all tags within the actual 990-EZ form were present in the schema estimate; however, some data tags alone were not enough to describe the data that it refers to. In this case, the path to the data tag was a better way to determine the schema.

Part II - Balance Sheets		Check if used schedule O for this part 2		InfoInScheduleOPartIIInd
	22 (A) BOY	Cash, savings, investments	Data Entry	CashSavingsAndInvestmentsGrp/BOYAmt
	22 (B) EOY			CashSavingsAndInvestmentsGrp/EOYAmt
	23 (A) BOY	Land and buildings		LandAndBuildingsGrp/BOYAmt
	23 (B) EOY			LandAndBuildingsGrp/EOYAmt

	24 (A) BOY	Other assets (describe in Schedule O)		OtherAssetsTotalGrp/BOYAmt
	24 (B) EOY			OtherAssetsTotalGrp/EOYAmt
	25 (A) BOY	Total assets		TotalAssetsGrp/BOYAmt
	25 (B) EOY			TotalAssetsGrp/EOYAmt
	26 (A) BOY	Total liabilities		SumOfTotalLiabilitiesGrp/BOYAmt
	26 (B) EOY			SumOfTotalLiabilitiesGrp/EOYAmt
	27 (A) BOY	Net assets or fund balances		NetAssetsOrFundBalancesGrp/BOYAmt
	27 (B) EOY			NetAssetsOrFundBalancesGrp/EOYAmt

Table 2: Paths for XML Tags

To account for the difference in tags each year, the team computed the union of the tags, while preserving the path to each tag. This was not the best approach since it led to empty values for tags that were less frequently used over the years; however, this ensured that there was no loss in data. As expressed by the client, irrelevant columns will be dropped as needed when using the database if they are not relevant to tourism office research. Meanwhile, highly prevalent tags that referred to the same data were manually combined into one column.

990

Due to the massive number of tags in the standard 990 forms, a slightly different approach was taken to ensure that all data tags were accounted for and to ensure that some level of schema alignment would be possible. This approach involved the extraction of sets of tags for each of the given tax years from 2012 to 2020. To ensure that all the data was accounted for, every single 990 file from the IRS website was processed to produce a set of tags, which required a large amount of processing time [8]. Upon obtaining these tag sets, alignment was completed primarily through manual processes. Attempting to automate this process would have taken a massive amount of processing time and would likely not be much more accurate, if more accurate at all. This manual alignment was completed through computing the intersections and symmetric differences of the sets of tags. Since the team decided to align the tags to the 2013 data, these computations were completed in a pairwise manner, comparing every other year with 2013. Upon completing these computations, the symmetric differences were used as the primary

means for aligning the tags. If a tag appeared in 2014's symmetric difference, there may exist a tag in 2013's symmetric difference with a different name, but that refers to the same type of data. In Table 3, this process is shown by blank fields under m2013 representing that the tag exists exactly the same in m2013 as 2014. Otherwise, the corresponding tags are equivalent in 2013.

m2013	2014
@documentCount	@documentCnt
	@interestAmt
@softwareVersion	@softwareVersionNum
	ActivitiesEngagedOrgInvlmntInd
	ActivitiesFurtherExemptPrpsInd
	ActivitiesTestInd
AddressLine1	AddressLine1Txt
AddressLine2	AddressLine2Txt
	AffiliatedScheduleGrp
	AffiliateListingGrp
	AppointElectMajorityOfficerInd
AssetLevelInd	AssetLevelCriteriaInd
BusinessNameLine1	BusinessNameLine1Txt
BusinessNameLine2	BusinessNameLine2Txt
	CHNAConductedWithNonFcltsInd
	CommuntityNotifiedFAPInd
	CompensationExplanation
	CompensationHighestPaidEmplGrp
	CompensationOfHghstPdCntrctGrp

Table 3: Aligning 2014 with 2013

For 2014 onwards, the intersections between each year and 2013 were quite large, with the symmetric differences containing only a few dozen to around 200 tags. As a result, this alignment process worked in a relatively consistent manner for all years between 2014-2020 (inclusive), with mostly the same new tags aligning to some of the 2013 tags. Since the XML data was massively transformed from 2012 to 2013, the alignment of the 2012 data was a much more tedious process, as the intersection of the two years was very small, and each year's symmetric difference was quite large.

m2013	2012
@binaryAttachmentCnt	@binaryAttachmentCount
	@contributionsReportedOnLine1a
	@note
@organization501cTypeTxt	@typeOf501cOrganization
AccountantCompileOrReviewInd	AccountantCompileOrReview
AccountActivitiesOutsideUSGrp	AcctsActvsOutUSTable
FAPActionsOnNonpaymentInd	ActionsOnNonpayment
ActivitiesConductedPrtshpInd	ActivitiesConductedPartnership
ActivitiesNotPreviouslyRptInd	ActivitiesNotPreviouslyRpt
ActivityCd	Activity
ActivityOrMissionDesc	ActivityOrMissionDescription
	ActivityOther
AddressChangeInd	AddressChange
ContractorAddress	AddressOfContractor
AdequateBooksAndRecMaintInd	AdequateBooksAndRecdsMaintaind
AdministrativeExpensesAmt	AdministrativeExpenses
AdoptBudgetInd	AdoptBudget
AdoptImplementationStrategyInd	AdoptImplementStrategy
AdvanceRefundingInd	AdvanceRefunding
AgentTrusteeEtcInd	AgentTrusteeEtc
AggregateReportedDuesNtcAmt	AggrAmtReportedInDuesNotices
AttorneyGeneralNotifiedInd	AGNotified
	AgreementRefFundraising
AgreeCarryoverPriorYearInd	AgreeToCarryoverPriorYear
AllAffiliatesIncludedInd	AllAffiliatesIncluded
AllNeedsAddressedInd	AllNeedsAddressed
AllOtherContributionsAmt	AllOtherContributions

Table 4: Aligning 2012 with 2013

Populating the Database

After the preprocessing step and schema alignment, the data was ready to be uploaded to the database. The database component of this project leveraged the SQLite framework [12]. SQLite was used rather than a typical relational database management system due to SQLite being lightweight and flexible. The amount of data was also constant and did not require an excessive amount of storage once in a database.

In order to populate the database, a Python script was necessary. The team used libraries like Pandas and SQLite3 to create and populate the database [12, 13]. SQLite3 was needed because the database will be constructed using SQLite, and SQLite3 is the latest version of the Python library that supports SQLite [15]. The team used Pandas for parsing files and inserting the data into the tables. Pandas has many functions to parse various file formats like Excel Workbook, CSV, and JSON [3]. Pandas also supports many different database tools, making it flexible in case the client wants to change to a different database structure in the future. Overall, Pandas and SQLite3 were optimal choices for the initial iteration of this database because they were easy to use and understand. Additionally, these libraries provided built-in tools for quickly visualizing trends in the data, such as MatPlotLib's integration in Pandas.

Implementation

This section describes the low-level technical implementations that the team used for the various phases in the project. It is broken down into sections on the scraping, the preprocessing (which primarily involves the extraction of XML tag sets), and the database population. The final subsections discuss the deployment of and introductory analysis performed on the database.

Scraping Code

Although the IRS provided a wealth of data regarding the 990 tax forms, the data was not organized or well documented. All of the data within AWS S3 needed to be downloaded individually as document objects and cannot be bulk downloaded like a ZIP file [6]. The data in AWS also provided an index file for each year that referenced a filing's year, location in S3, and return type.

Key	Value
DLN	9349131500445
EIN	742661023
FormType	990PF
LastUpdated	2016-09-09T23:14:27
ObjectId	201543159349100344
OrganizationName	HARRIET AND HARMON KELLEY FOUNDATION FOR THE ARTS
SubmittedOn	2016-02-09
TaxPeriod	201412
URL	https://s3.amazonaws.com/irs-form-990/201543159349100344_public.xml

Table 5: Breakdown of a JSON entry contained in index files

Using these index files, the team determined the filings of interest before downloading the actual XML files. Manually sifting through the index files was not the most efficient use of time, so the team developed a script, called *aws_download,* for downloading the XML files given the tax period, filing type, and number of desired files to download. First, an index file for a certain year was fetched as a JSON object. The value of the JSON object was an array of filing references, which were also JSON objects themselves with their values shown in Table 5. The team then looped through the references and conditionally checked for the form type. Filings were grouped into 990 or 990-EZ. The reference link to these filings of interest were populated into their own array. Finally, using an AWS SDK for Python (Boto3), a S3 client was initialized, and the team downloaded each filing as XML to the local machine using the location link. Figure 2 shows an example of a downloaded XML file [10].

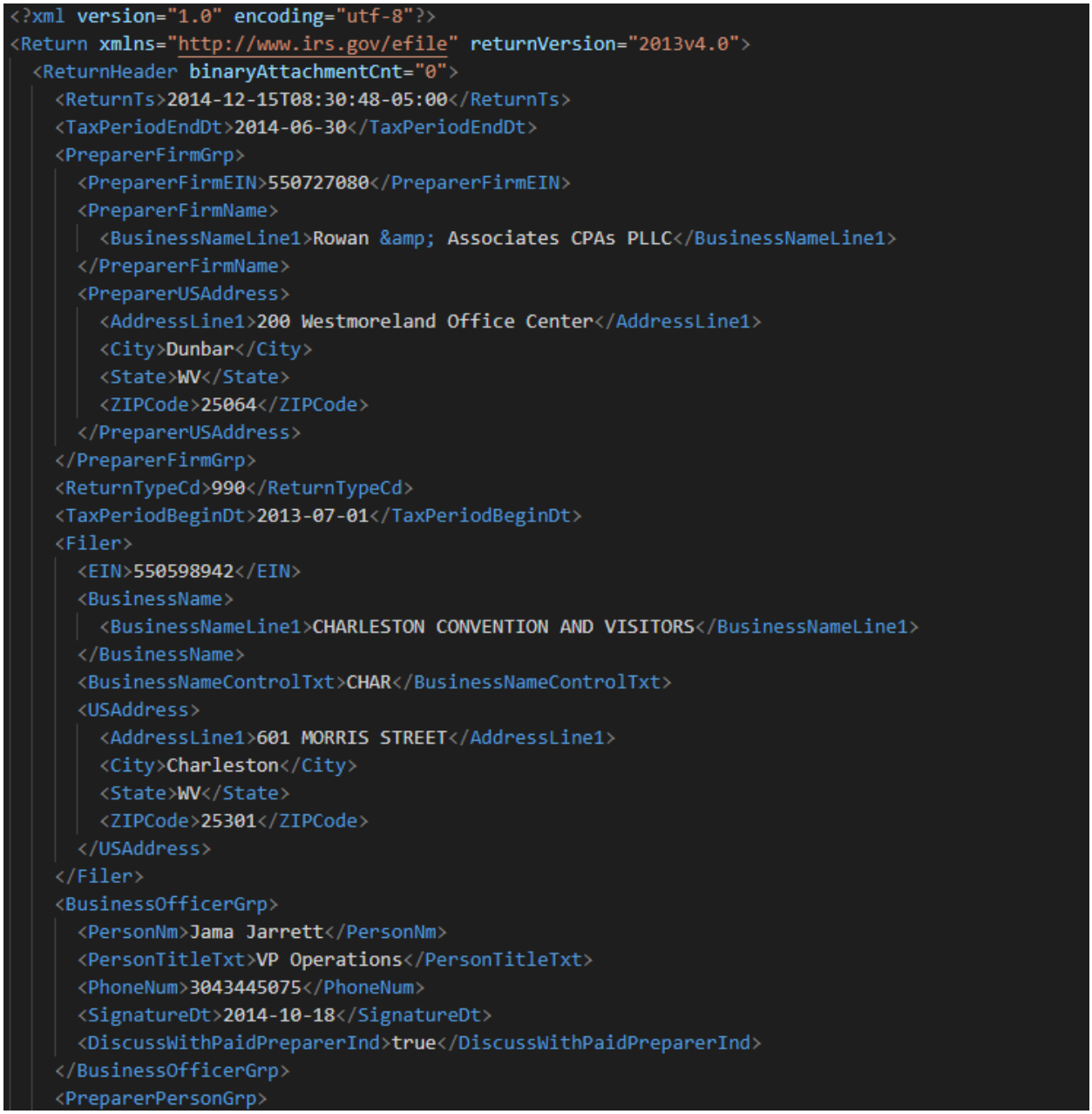


Figure 2: XML structure of a 990

Data Processing

Obtaining XML Tag Sets

One of the crucial analytical steps that took place in order to align the XML tags for the 990 and 990-EZ filings involved the synthesizing of each tax year's data into a Python set. This synthesis allowed for an efficient extraction of all data tags that appear in at least one filing in a given year, while also allowing for extremely quick comparisons between pairs of tax years. The code used

to perform this process was encapsulated in two functions: *get_tag_set* and *key_adder*. A screenshot of these functions is included in Figure A7 in the Appendix.

The *get_tag_set* function governs the primary logic for the extraction of XML tag sets for a given set of XML files. Its crucial inputs are a list of file names and a reference to a list of keys that the function populated with XML tags before being converted into a set. It leveraged the XmlToDict Python module for an initial conversion of the XML structure to a dictionary [14]. This step was followed by converting the standard dictionary to a JSON format using the JSON library's *dumps* and *loads* methods. This process was handled in a loop through each of the XML files that were passed into the function. Each loop iteration in this function then handed off the logic of extracting individual keys to the *key_adder* function.

The *key_adder* function took the tax return data section of the XML file as its data input, along with the same reference to the list of keys that was passed into *get_tag_set*. This list was passed as a reference such that it could be easily built up through the recursion that is run in *key_adder*. The key adder used the logic of traversing a series of nested dictionaries, which was the structure of the XML converted to JSON. It checks if the current reference to a tag in the input dictionary is of type "list", "dict", or otherwise. If it was a list, the key adder looped through the list and used recursive calls to populate the tags list with the tags and subtags in that input list. If it was a dictionary, the key adder traverses down the hierarchy of dictionary keys with recursive calls in order to populate the tags list. Finally, if the key adder encountered a non-iterable type, the logic reached its base case and the located tag was added to the tags list shown in Figure 3.

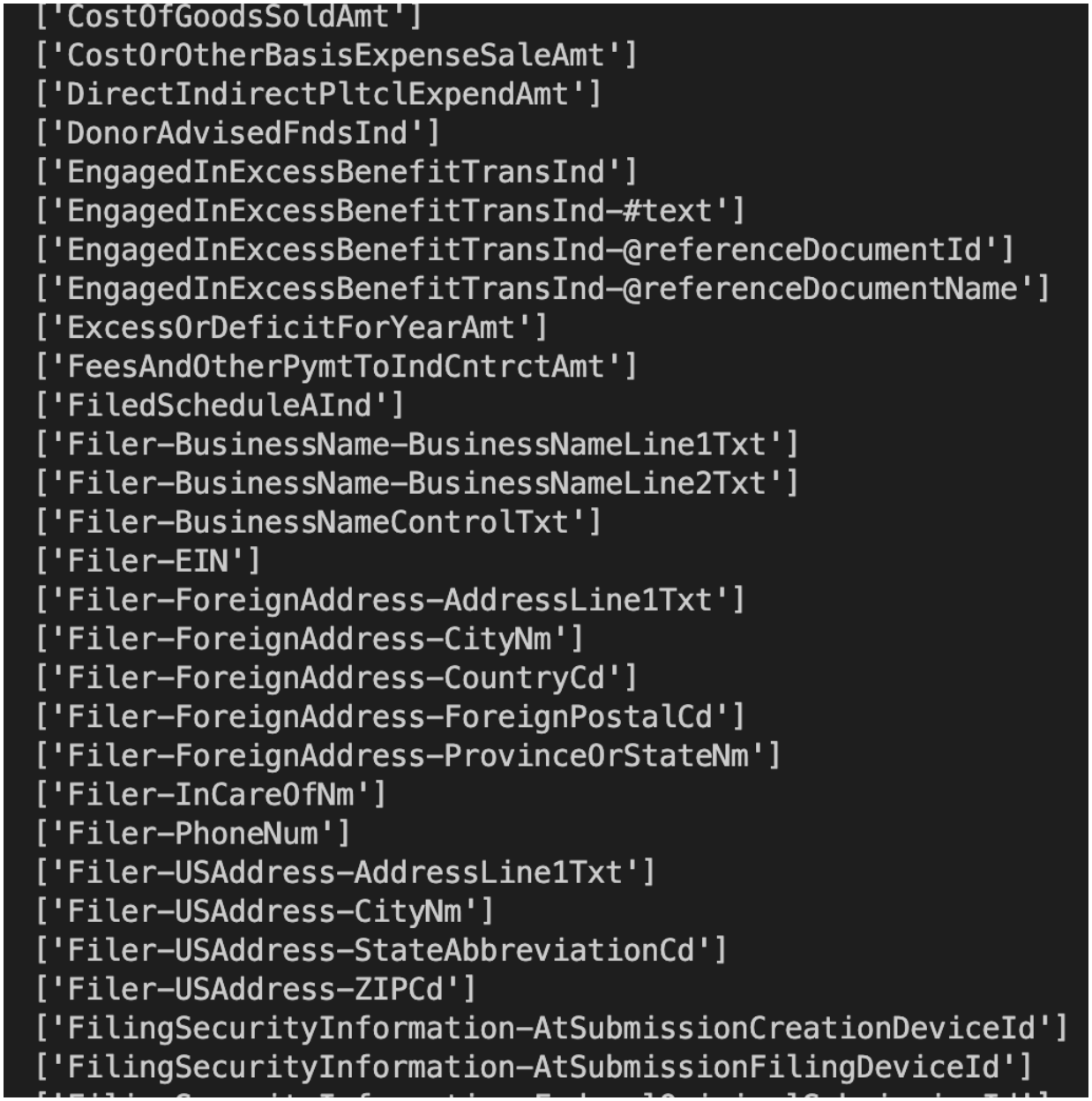


Figure 3: A few tags extracted from a 990-EZ XML

Upon the completion of the *key_adder* logic for each file in the input list, the tags list was converted into a set, cleaned up, and written to a CSV file. For the purposes of this project, an entire year's worth of data was run through the logic in order to obtain the XML tag set for each tax year.

Aligning Schema & Outputting to CSV

The process of preparing the Form 990 data for database ingestion came in two phases: schema alignment and CSV conversion. As discussed in the design section, schema alignment serves to

match up as many XML tags as possible across the different years' tax codes. This allows for data to be accessed and analyzed in a unified manner in the database. The aligned data was then converted into CSV files for each tax year. This allowed for the processed data to be easily ingested into the database using a manner consistent with how the project's other data was added to the database. XML contains two types of tags which the team referred to as path tags and data tags. The path tag is an outer tag that does not contain the tax information to extract, but contains more outer tag or a data tag. A data tag contains the tax information needed to be extracted. If the XML is viewed as a tree structure, the path tags would be the value or name of the internal nodes while the data tags are the name of the leaf nodes.

Aligning Schema

Table 3 and Table 4 display pairs of Form 990 XML tags that were used for aligning data to the 2013 schema. These pairs of columns (each other year paired with 2013) were stored in a CSV file called *alignment_table*. This table was loaded into a Jupyter notebook as a dictionary, with each non-2013 year as the outermost keys. Each outer key was subsequently mapped to an inner dictionary. These inner dictionaries had simple key-value pairs, which mapped the non-2013 tag to its matching 2013 tag. This structure allowed for tags in need of alignment to be located and renamed as efficiently as possible.

Form 990-EZ data was aligned similarly. A dictionary mapped pre-2012 path and data tags to the post-2013 path and data tags. These tags are stored in CSV files called *ez_alignment_data_tag* and *ez_alignment_path_tag.* As the data extraction script transverses the 990-EZ XMLs, if a path tag or data tag in the processing XML was a key in the dictionary, then the value to the key was used as the header to the output CSV file.

CSV Conversion

This schema alignment logic was all encapsulated within the logic used to output the 990 and 990-EZ data to CSV files. This logic primarily relies on two functions: *adder* and *save*, which are called in a nested loop structure that traverses the years of the study timeframe, and the filings in each of those years. The *adder* function works similarly to the previously mentioned *key_adder* function; however, it shifts its focus from simply adding data tags to adding the raw filing data to an output data structure. The function traverses the XML in a tree-like manner. At internal nodes, the tag names (referred to as "path tags") are aggregated into a path string. This string serves to describe the entire subtree path to an eventual piece of data at a data tag. This path string then serves as the column header for the data at that data tag. The *key_adder* logic still exists in order to create a set of these *headers*, which must be eventually written to the output file before the data itself. Beyond the headers, the data stored within each key is accessed and appended to a *data* list for each filing. Each filings *data* list is appended to an overall *data*

list for each Tax Year. At the completion of processing for an entire year's worth of data, the headers set and data list were ready to be saved to a CSV file.

In order to better assess potential bugs or issues with particular XML files, this adder logic was processed in four different phases. First the *ReturnHeader* section of the XML was added to the *headers* and *data*, followed by the main body of the return data. This main body is contained within the *ReturnData* tag under the tag name of either *IRS990* or *IRS990EZ*. The Schedule forms that needed to be processed came next. Both Schedule O and Schedule J also existed within the *ReturnData* tag under the names *IRS990ScheduleO* and *IRS990ScheduleJ*. These four calls to the *adder* resulted in the final representations of the *headers* and *data* for a given filing. These pieces of data for each filing were then aggregated into *headers* and *data* lists for the entire tax year, which finalized and prepared this data for the next step.

This final step of the conversion process was managed by the *save* function. This function uses a simple if statement and the *OS* library in order to see if a suitable output directory exists. These output directories were called *ProcessingOutput990* and *ProcessingOutput990EZ* for Form 990 and 990-EZ, respectively. If the proper output directory did not exist, it was created using the *OS* library. Once the output directory was established, the *CSV* library was the next to be leveraged in order to create proper CSV representations of the data headers and the data itself. The *DictWriter* object was the optimal choice in order to govern the CSV writing logic. The headers were placed in the CSV file first using *writeheader* before the data was subsequently added using *writerows*.

990 Filing Data

After schema alignment each tax filing was converted into a JSON object, but only the combination of various tax filings were converted and saved into a CSV file. Similar to the schema extraction, which was handled in a recursive manner, the extraction of the data also used a recursive method. As the script traverses through the dictionary within the dictionary structure of JSON like in the *key_adder* method, when the data tag was reached, it meant the value of the current key was the actual data that needed to be stored in the database. Therefore, the team created a mapping with the path as the key and the IRS data as the value in a dictionary structure. This resulted in a dictionary for every filing with the path to the data as the key and the value as the actual data. Finally, each filing dictionary is stored in an array to be converted to a CSV file. The path or the key was also stored in a set to produce the CSV output. The dictionary structure of each filing was not saved because each row in the output CSV contained the same information. The result of the conversion and extraction resulted in a CSV file for a given year with a unified schema header and data within the body cells.

Processing Officers

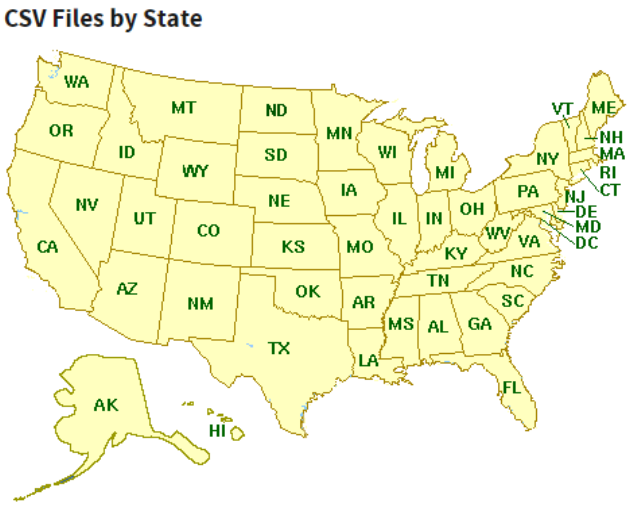
Due to the difficulty of storing array data within a singular CSV entry, the team and the client agreed to process the officers from an organization separately from the main body of the filing data. In Form 990, the officers existed within the tag *Form990PartVIISectionA* in 2012, and *Form990PartVIISectionAGrp* in 2013-202. In Form 990-EZ this data was present within the tag *OfficerDirectorTrusteeEmplGrp*. These officer lists were extracted from these tags and converted into CSV files using the same *save* logic that was described in the section above. Due to the large number of officers and the large number of 990 filings in each Tax Year, the Form 990 officers were saved to a single CSV file for each XML file. Meanwhile, due to the 990-EZ forms having fewer officers and fewer filings, these officers tables were outputted to a singular CSV for each Tax Year, alongside each filing's *EIN* and *tax year* in order to differentiate the adjacent tables.

Database Population Code

Adding the Exempt Organization Business Master Files

In order to have up-to-date master files from the IRS, it was best to download the master files when the database was created and populated. First, the necessary packages for this program, Requests, IO, SQLite3, and Pandas, were imported. Requests were used to request master files from the IRS website [8, 13, 15, 16, 17]. The IO library was there for test printing and to announce tasks being executed successfully. The SQLite3 library was necessary to create an SQLite database, and Pandas was used to parse the data and for data visualization.

Next, master files were downloaded and imported to the database. There are two types of master files available on IRS: the master files separated into 4 regions and the master files separated by state. They would be called *Region Files* and *State Files,* respectively. Both types were downloaded and imported to the database due to discrepancies between totals for the types.



CSV Files by Region

Region 1: Northeast Area (CSV): Includes Connecticut, Maine, Massachusetts, New Hampshire, New Jersey, New York, Rhode Island, Vermont.

Region 2: Mid-Atlantic and Great Lakes Areas (CSV): Includes Delaware, District of Columbia, Illinois, Indiana, Iowa, Kentucky, Maryland, Michigan, Minnesota, Nebraska, North Carolina, North Dakota, Ohio, Pennsylvania, South Carolina, South Dakota, Virginia, West Virginia, Wisconsin.

Region 3: Gulf Coast and Pacific Coast Areas (CSV): Includes Alabama, Alaska, Arkansas, Arizona, California, Colorado, Florida, Georgia, Hawaii, Idaho, Kansas, Louisiana, Mississippi, Missouri, Montana, Nevada, New Mexico, Oklahoma, Oregon, Texas, Tennessee, Utah, Washington, Wyoming

Region 4: All Other Areas (CSV): Includes International and all others.

- International (CSV) (Excluding P.R.)
- Puerto Rico (CSV)

Figure 4: Different Master File Types from IRS Website

The downloading process started with region files, as shown in Figure 5.1.

```
In [2]: # start downloading eo region files and put it in "eo.db" database in eo_regions
eos = [1, 2, 3, 4]
total_lines = 0
conn = sql.connect('eo.db')
for eo in eos:
    cur_eo = 'eo' + str(eo) + '.csv'
    URL = 'https://www.irs.gov/pub/irs-soi/' + cur_eo
    response = requests.get(URL)
    open(cur_eo, 'wt').write(response.text)
    master_file = pd.read_csv(cur_eo, index_col=False)
    master_file.to_sql('eo_regions', conn, if_exists='append', index=False)
print("Finished putting items in table eo_regions.")
```

Figure 5.1: Downloading region files

Here, *sql.connect* was used to create a database called “eo.db” to hold the master file data. Then, the directories to receive the region files were formed, labeled one through four, and “get” was used to download the file to the current directory. After that, the downloaded files were parsed using *pandas.read_csv* and dumped to “eo.db” as a table named “eo_regions” using the *to_sql* function. This process was additionally used for *State Files*, which can be found in Figure 3.

```
In [3]: # start downloading eo state files and put it in "eo.db" database in eo_states table
        states = ['AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'ID', 'IL', 'IN',
                  'IA', 'KS', 'KY', 'LA', 'ME', 'MD', 'MA', 'MI', 'MN', 'MS', 'MO', 'MT', 'NE', 'NV', 'NH',
                  'NJ', 'NM', 'NY', 'NC', 'ND', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT',
                  'VT', 'VA', 'WA', 'WV', 'WI', 'WY']
        for state in states:
            cur_eo = 'eo_' + str(state).lower() + '.csv'
            URL = 'https://www.irs.gov/pub/irs-soi/' + cur_eo
            response = requests.get(URL)
            open(cur_eo, 'wt').write(response.text)
            master_file = pd.read_csv(cur_eo, index_col=False)
            master_file.to_sql('eo_states', conn, if_exists='append', index=False)
        print("Finished putting items in table eo_states.")
```

Figure 5.2: Downloading region files

After downloading and importing the two types – regions and states – of master files, a count was performed to see whether the two files matched up. This matching refers to determining whether the regions and states files contain the same company data. This was determined based upon comparing the EINs present in each of the two files. This analysis in Jupyter is described in Figure A1 in the Appendix. From this analysis, it was determined that there are 4,049 more organizations in the *Region Files* compared to the *State Files*. In order to have one list of organizations, a union was necessary to form a table that contained all entries; see Figure 6.

```
In [7]: cur = conn.cursor()
        cur.execute('CREATE TABLE eo_combined AS SELECT * FROM eo_regions UNION SELECT * FROM eo_states')
        cur.close()
        conn.commit()
        print('Finished combine two tables.')
```

Figure 6: Unioning region and state EO BMFs

After unioning the data sets, another count was performed again to double check the results. The results of this can be found in Figure A2 in the Appendix. This analysis determined that the union SQL query correctly combines the tables, and thus the team could proceed with using this combined table in the database.

Database Creation & Data Population

The database was created using the SQLite3 *connect* function that forms a connection to a database indicated by the database file name that is passed in as a parameter. This connection is

represented by a *Connection* object, which must be passed in to any database queries using Pandas. If a connection is formed with a non-existing table, the Connection object will create a new, empty database.

```
In [7]: conn = sql.connect("eo.db")

In [8]: df = pd.read_sql("SELECT * FROM tax_990 LIMIT 5", conn)
        df
```

Out[8]:

	@binaryAttachmentCnt	@documentId	@referenceDocumentId
0	0.0	RetDoc1038000001	RetDoc1044400001
1	0.0	RetDoc1038000001	RetDoc1044400001
2	0.0	RetDoc1038000001	RetDoc1044400001
3	0.0	RetDoc1038000001	RetDoc1044400001
4	0.0	990	None

5 rows × 972 columns

Figure 7: Connection Object creation and application using through Pandas

After the database connection was formed, the next step was to populate the database with tax filings. Form 990 data was added to the database first. The process that the team planned to populate the database began with creating a series of Pandas *DataFrames* for each year from 2012 to 2020. Although some of the headers were aligned, there were still many differences between the different Tax Years' data. In order for the database to function in a unified manner, a union of all column headers needed to be added to the database before the data itself could be uploaded. Consequently, the team used the Pandas *concat* function to union the columns together.

After going through all the years, the CSV file in 2012 and 2013 had some mismatched column names due to text case irregularities that were missed in the primary processing phase. Specifically, there was a tag called "ZIPCode" in the 2012 and 2013 data, while that same column was named "ZipCode" in the remaining years. The team used a small script to address this issue in Figure 8.

```
In [6]: dfs = []
        for year in [2012,2013]:
            query = "SELECT * FROM tax_990_" + str(year)
            df_year = pd.read_sql(query, conn)
            zip_list = []
            for col in list(df_year):
                if 'ZIP' in col:
                    zip_list.append(col)
            knvs = {}
            for key in zip_list:
                val = key.replace('ZIPCode', 'ZipCode')
                knvs.update({key:val})
            df_year = df_year.rename(columns = knvs)
            dfs.append(df_year)
```

Figure 8: Fix "ZIP" capitalization for 2012 and 2013 990 filings

The variable *dfs* was created as an empty list to store all of the *DataFrames* in order to union all the column headers, as previously discussed. As the for-loop went through each year, the *DataFrame* for that year was stored in the df_year variable, and each *DataFrame* was searched for all column names that contained "ZIP". A key-value pair was created using the original column name and the result of running it through *replace* to obtain the properly augmented name. This pair was passed through the rename function in order to update the column name. After updating the column names, *df_year* was appended to *dfs* for later use. Next, the remaining years were converted to DataFrames and stored in *dfs*, as shown in Figure 9.

```
In [7]: years = list(range(2014,2021))
        for year in years:
            query = "SELECT * FROM tax_990_" + str(year)
            df = pd.read_sql(query, conn)
            dfs.append(df)
            print("Finished year ", year)

        Finished year  2014
        Finished year  2015
        Finished year  2016
        Finished year  2017
        Finished year  2018
        Finished year  2019
        Finished year  2020
```

Figure 9: Append tax filing year 2014-2020 as DataFrame to list dfs

The last step was to union all DataFrames in the list of DataFrames *dfs*. The newly unioned DataFrame was stored in *final_frame* and then converted to a table *tax_990* in the database. The code for this step is found in Figure A8 in the Appendix.

The same process was applied to populate the database with 990-EZ filings and its table is *tax_990ez*. The officers data for both form types were the next to populate the database, following the same method, but omitting the zip code renaming step.

Tourism Office Filtering & Conversion

Exempt Organization Business Master Files

The criteria to filter tourism offices was given by the client. These criteria were used as strings for easier management and for easier query writing.

Field	**Values that can indicate a tourism office**
Activity Code (Three 3-digit subcodes) (AAABBBCCC)	213 in any of the 3 positions
NTEE	P61, S01, S20, S30, S41, T99
Starting Keywords (Only first word)	VISIT, EXPLORE, TRAVEL, DISCOVER
Keywords	CVB, VISITOR BUREAU, VISITORS BUREAU, CONVENTION BUREAU, CONVENTION & VISITOR, VISITOR & CONVENTION, VISITOR AND CONVENTION, VISITORS & CONVENTION, VISITORS AND CONVENTION, CONVENTION BUREAU, VISITOR CENTER, TOURISM OFFICE, TOURISM BUREAU, TOURISM ASSOCIATION, TOURIST ASSOCIATION, TOURISM ASSN, VISITOR AUTHORITY, VISITORS AUTHORITY, CONVENTION AUTHORITY, DESTINATION MARKETING, DESTINATION MANAGEMENT, VISITOR, BUREAU

Table 6: Filtering criteria for Tourism Offices

The master table had a column called “Activity” that holds a nine-digit integer. That number was broken down into three-digit activity codes. Division and modulus calculations were used to filter out the first, middle, and last three digits of the number. Activity Code of 213 represents “Tourism Bureaus” according to the guide for the EO BMF [8]. For the rest, there was the extra condition required when filtering company names. Sometimes companies would have many activities, but the “Activity” field only includes the top three most prevalent types. That is why keywords and the criteria were used to identify tourism offices. The National Taxonomy of Exempt Organization (NTEE) code is similar to the Activity code but is often broader in scope. The NTEE codes P61, S01, S20, S30, S41, T99 mean Traveler’s Aid; Alliance/Advocacy

Organizations; Community, Neighborhood Development, Improvement (General); Economic Development; Promotion of Business; and Philanthropy, Voluntarism, and Grantmaking Foundations N.E.C., respectively. SQL query strings were formed using these criteria to filter out tourism offices. Some helper functions were made to make this process easier to understand and maintain. These functions and logic are grouped together in Figure A3 in the Appendix. After query strings were created, they were combined into a long string. Together, they created the SQL query to filter the tourism offices out of the master files. This query was executed through Pandas in order to return a DataFrame, and with this DataFrame, a CSV file was created as a deliverable for the client.

Database Deployment

The method of transferring the database involved exporting the database as a .db file. With a .db file, the client was able to load the database on any machine and run the predetermined queries and analysis. The team has a shared file space through Virginia Tech Advanced Research Computing (ARC) in order to store the .db file for the client's usage.

Testing, Evaluation, and Assessment

After the deployment of the database, queries and analysis were performed to obtain interesting information about the tax data. Per the client's request, the team researched various crucial variables of interest like compensation of employees, number of voting members, and many more. This preliminary analysis was performed on the database in order for the team to gain a more comprehensive understanding of the various analyses at the disposal of the client, and of any shortcomings of the database. The database allows obtaining various fields for a form version in a set year as shown in Figure 10 and Figure 11.

```
In [91]: query = "SELECT " + column_names + " FROM tax_990_" + str(year)
         df = pd.read_sql(query, conn)

In [92]: df
```

Out[92]:

	Filer-EIN	GoverningBodyVotingMembersCnt	IndependentVotingMemberCnt	TotalEmployeeCnt	TotalVolunteersCnt
0	43159713	6	6	0	0.0
1	262185891	7	7	0	20.0
2	920040101	12	12	2	75.0
3	680492065	10	9	202	7087.0
4	752286990	10	10	9	70.0
...	...	...	...	...	...
213550	550694953	3	3	9	NaN
213551	473932789	5	0	0	NaN
213552	204793354	3	4	2	NaN
213553	364168063	30	30	0	30.0
213554	237441833	13	13	28	320.0

213555 rows × 274 columns

Figure 10: Test query for required fields for 990 filings in 2015

```
In [36]: df = pd.read_sql(query, conn)

In [37]: df
```

Out[37]:

	Filer-EIN	GoverningBodyVotingMembersCnt	IndependentVotingMemberCnt	TotalEmployeeCnt	TotalVolunteersCnt
0	426077901	1	0	0	0.0
1	980684822	16	16	38	650.0
2	203319173	5	5	0	12.0
3	943141712	10	10	87	20.0
4	161339956	23	22	12	0.0
...	...	...	...	...	...
1790	593486867	0	0	11	11.0
1791	570886869	7	7	0	0.0
1792	510181418	4	0	0	0.0
1793	954712218	16	13	217	20.0
1794	561677675	10	10	0	10.0

1795 rows × 271 columns

Figure 11: Test query for required fields for 990 filings in 2013

The database also allows filtering of shown fields and filtering based on different fields that will be elaborated on in the Users' Manual. A Python function allowing for the simple filtering of fields of interest is included in Figure A4 in the Appendix.

Preliminary Analysis

In order to test the database, a set of objectives to analyze was set. These objectives included the use of the database to output CSV the top 20 organizations based on reported revenue for a given year, along with the top 20 tourism organizations based on reported revenue. The results for the first objective helped the team better understand the scope of the data within the database. While tourism offices were the primary focus and motivation, the database contains information about education institutions such as various universities and many healthcare organizations. The organizations that were in the top 20 in 2019 can be seen in Table 7. One of the primary focuses for this analysis was to see the changes over the years. While the organizations in the top 20 from 2013-2019 were relatively constant, that trend did not continue into 2020. The universities and health organizations that were a part of the top 20 through 2019 were no longer within the list. They were replaced by charity health organizations like The Global Fund to Fight Aids, Tuberculosis, and Malaria and many electrical companies. Much of this change can likely be attributed to the COVID-19 pandemic, although the client did suggest that this might be caused due to companies asking for extensions and the data not being available yet.

Business Name	Revenue	City	State
KAISER FOUNDATION HEALTH PLAN INC	62,519,341,516	OAKLAND	CA
KAISER FOUNDATION HOSPITALS	30,444,780,937	OAKLAND	CA
UPMC GROUP	16,575,357,648	PITTSBURGH	PA
FIDELITY INVESTMENTS CHARITABLE GIFT FUND	12,252,844,500	BOSTON	MA
THE CLEVELAND CLINIC FOUNDATION	11,558,538,378	INDEPENDENCE	OH
MAYO CLINIC GROUP RETURN	10,416,976,478	ROCHESTER	MN
Thrivent Financial for Lutherans	9,585,386,020	MINNEAPOLIS	MN
BATTELLE MEMORIAL INSTITUTE	9,245,496,160	COLUMBUS	OH
DIGNITY HEALTH	8,760,338,732	SAN FRANCISCO	CA

Healthfirst PHSP Inc	8,698,913,059	NEW YORK	NY
CareSource	8,150,209,969	Dayton	OH
NEW YORK UNIVERSITY	8,011,264,159	NEW YORK	NY
TRUSTEES OF THE UNIVERSITY OF PENNSYLVANIA	7,624,539,000	PHILADELPHIA	PA
JOHNS HOPKINS UNIVERSITY	7,505,891,000	BALTIMORE	MD
President and Fellows of Harvard College	7,350,273,474	CAMBRIDGE	MA
Banner Health	7,138,466,898	PHOENIX	AZ
Mercy Health	7,010,225,301	Cincinnati	OH
IHC HEALTH SERVICES INC	6,979,289,805	SALT LAKE CITY	UT
THE BOARD OF TRUSTEES OF THE LELAND STANFORD	6,848,913,893	REDWOOD CITY	CA
THE CLEVELAND CLINIC FOUNDATION	6,830,413,008	INDEPENDENCE	OH

Table 7: Top 20 for all organizations in 2019

Business Name	Revenue	City	State
THE GLOBAL FUND TO FIGHT AIDS TUBERCULOSIS AND MALARIA	6,951,923,532		
University Health Network	2,026,595,052		
IBM MEDICAL BENEFITS TRUST C/O JPMORGAN CHASE BANK NA	1,146,807,350	BROOKLYN	NY
NORTH CAROLINA ELECTRIC MEMBERSHIP CORPORATION	1,109,305,874	RALEIGH	NC
SHRINERS HOSPITALS FOR CHILDREN	8,961,69,390	TAMPA	FL
SAMARITAN'S PURSE	894,308,893	BOONE	NC

MCLAREN HEALTH PLAN INC	845,052,646	FLINT	MI
SAVE THE CHILDREN FEDERATION INC	808,658,178	FAIRFIELD	CT
National Trust for Places of Historic Interest or Natural Beauty	797,350,000		
University of Manitoba	743,535,000		
Network for Good	683,013,296	Washington	DC
Upper Missouri G&T Electric Cooperative Inc	637,652,492	Sidney	MT
CHRISTIAN HEALTHCARE MINISTRIES INC	633,361,869	BARBERTON	OH
HealthWell Foundation	558,747,759	Germantown	MD
MANHATTAN AND BRONX SURFACE TRANSIT OPERATING AUTHORITY	558,511,422	New York	NY
MEDECINS SANS FRONTIERES USA INC	558,340,480	NEW YORK	NY
COMMUNITY CARE INC	553,758,821	BROOKFIELD	WI
UNITED STATES GOLF ASSOCIATION	517,714,574	LIBERTY CORNER	NJ
SOUTH TEXAS ELECTRIC COOPERATIVE INC	509,268,124	NURSERY	TX
WITHLACOOCHEE RIVER ELECTRIC COOPERATIVE INC	487,757,302	DADE CITY	FL

Table 8: Top 20 for all organizations in 2020

The database was also used to run a similar suite of tests to find top 20 tourism offices for each year. The top 20 are expected tourist destinations like Honolulu, Hawaii and Orlando, Florida that stay relatively consistent until 2020 similar to the previous test. Top tourism offices had a few entries from 2019 and the ones that remained had a heavy drop in revenue like the Denver tourism office that went from 40 million to 12 million. This data can be seen in Table 9 and 10. This drop is expected due to Covid-19 and demonstrates the database's ability to show the effects of Covid-19 on these organizations.

Filer-BusinessName-BusinessNameLine1	CYTotalRevenueAmt	Filer-USAddress-CityNm	Filer-USAddress-StateAbbreviationCd
OrlandoOrange County Convention & Visitors Bureau Inc	80519755	Orlando	FL
FLORIDA TOURISM INDUSTRY MARKETING CORP INC	48182560	TALLAHASSEE	FL
SAN DIEGO CONVENTION AND TOURIST BUREAU	41202845	SAN DIEGO	CA
Denver Metro Convention & Visitors Bureau	40992367	Denver	CO
ATLANTA CONVENTION AND VISITORS BUREAU INC	35729622	ATLANTA	GA
HAWAII VISITORS & CONVENTION BUREAU	34731961	HONOLULU	HI
CHICAGO CONVENTION AND TOURISM BUREAU	32617112	CHICAGO	IL
DALLAS CONVENTION & VISITORS BUREAU	28732606	DALLAS	TX
GREATER MIAMI CONVENTION & VISITORS BUREAU INC	26319264	MIAMI	FL
NASHVILLE CONVENTION & VISITORS BUREAU	25987835	Nashville	TN
SEATTLE-KING COUNTY CONVENTION AND VISITORS BUREAU	24903448	SEATTLE	WA
GWINNETT CONVENTION AND VISITORS BUREAU INC	22990818	DULUTH	GA
THE CONVENTION AND VISITORS BUREAU OF GREATER CLEVELAND INC	21456678	CLEVELAND	OH
DETROIT METRO CONVENTION AND VISITORS BUREAU	19753266	DETROIT	MI
Osceola Convention and Visitors Bureau INC	19380577	Kissimmee	FL
ANAHEIMORANGE COUNTY VISITOR & CONVENTION BUREAU	19256262	ANAHEIM	CA
CHARLESTON AREA CONVENTION & VISITORS BUREAU	18884277	CHARLESTON	SC
PHILADELPHIA CONVENTION & VISITORS BUREAU	17346622	PHILADELPHIA	PA
SAINT PAUL RIVERCENTRE CONVENTION AND VISITORS AUTHORITY	17184231	ST PAUL	MN

MEMPHIS CONVENTION AND VISITORS BUREAU	17172027	MEMPHIS	TN

Table 9: Top 20 Tourism Offices in 2019

Filer-BusinessName-BusinessNameLine1	Filer-USAddress-CityNm	Filer-USAddress-StateAbbreviationCd	CYTotalRevenueAmt
Denver Metro Convention & Visitors Bureau	Denver	CO	12927884
CONVENTION & VISITORS BUREAU OF GREATER KANSAS CITY	KANSAS CITY	MO	9870365
SAINT PAUL RIVERCENTRE CONVENTION AND VISITORS AUTHORITY	ST PAUL	MN	9717353
BUFFALO NIAGARA CONVENTION & VISITORS BUREAU INC	BUFFALO	NY	6896089
GRAND RAPIDSKENT COUNTY CONVENTION & VISITORS BUREAU DBA EXPERIENCE GRAND RAP	GRAND RAPIDS	MI	6496669
VISITGREENVILLESC	GREENVILLE	SC	6156731
GREATER PITTSBURGH CONVENTION & VISITORS BUREAU INC	PITTSBURGH	PA	5259727
Colorado Springs Conv & Visitors Bureau	Colorado Springs	CO	4844458
DOOR COUNTY VISITOR BUREAU INC	STURGEON BAY	WI	4472230
ASPEN CHAMBER RESORT ASSOCIATION INC	ASPEN	CO	4252308
BRECKENRIDGE TOURISM OFFICE	BRECKENRIDGE	CO	4066977
HAMILTON COUNTY TOURISM INC	CARMEL	IN	4030602
WASHTENAW COUNTY CONVENTION AND VISITORS BUREAU INC	ANN ARBOR	MI	3078567
CHESTER COUNTY CONFERENCE AND VISITORS BUREAU INC	KENNETT SQUARE	PA	2822839
WARREN COUNTY CONVENTION & VISITORS BUREAU	MASON	OH	2770923
GUNNISON CRESTED BUTTE TOURISM ASSOCIATION	GUNNISON	CO	2642041
TRI-CITIES VISITOR & CONVENTION BUREAU	KENNEWICK	WA	2477657
FORT WAYNEALLEN COUNTY CONVENTION AND VISITORS BUREAU	FORT WAYNE	IN	2162090
ELKHART COUNTY CONVENTION & VISITOR BUREAU INC	ELKHART	IN	2056340

DEKALB CONVENTION & VISITORS BUREAU INC	TUCKER	GA	1737476

Table 10: Top 20 Tourism Offices in 2020

Visualizing Data

An important use case for the EO database is extracting information from the tables using queries. Figures 10 and 11 show data displayed in data frames which may be enough for certain use cases for simple information extraction; however, the data can be further analyzed by creating visualizations for specific columns of data.

Visual charts and graphs were made using Pandas and MatPlotLib library in Python. Figure 12 shows the trend of total revenue for the top 3 bureaus in 2020 from tax year 2014 to 2020. From the chart, inferences and observations can be quickly made, such as the sharp decline in revenue from 2019 to 2020. An expert may ponder how COVID-19 affected the trend in the chart or draw their own conclusions and observations, and this could additionally inspire further research. Additional visual charts can be found in the Appendix, Figures A9 and A10. Figure A9 describes the amount of organizations involved in college related activities while Figure 10 shows the amount of organizations in various business and professional groups.

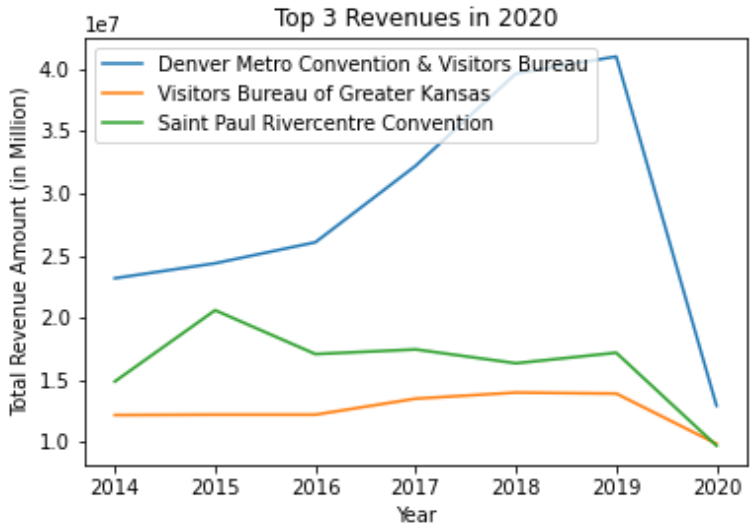


Figure 12: Change in total revenue amount for the top three tourism bureaus of 2020 from 2014 to 2020

Users' Manual

Using a .db File

In order to access the database, SQLite3 and Pandas are imported. To read the tables in the database file, a Connection object must be created to represent the database. This process is displayed in Figure 13.

```
In [1]: # import the neccessary modules
        import requests
        import io
        import sqlite3 as sql
        import pandas as pd
```

```
In [2]: conn = sql.connect("eo.db")
        df = pd.read_sql("SELECT name FROM sqlite_master WHERE type='table'", conn)
```

```
In [3]: df
```

Out[3]:

	name
0	eo_regions
1	eo_states
2	eo_combined
3	tax_990
4	tax_990ez
5	officer_990ez
6	officer_990

Figure 13: Example of how to access data in .db database

In this example, the SQLite database being used was *eo.db*, and a Connection object was created by calling the SQLite3 *connect* function and was stored in a variable called *conn*. This Connection object was necessary to access the database. After *conn* was created, Pandas queries could be used to access data in the database. In this example, the query's purpose was to see the list of tables in the database.

Example Queries

Upon obtaining access to the database, a user is ready to perform some queries to support their analysis. For this project, most queries were done through Pandas, but it was possible to query the database using SQLite3, along with other SQL processing software. As mentioned above, the required packages for this were IO, SQLite3, Pandas, and Numpy. An example query is shown in Figure 14.

```
In [3]: # make connection object with the database
        conn = sql.connect('eo.db')

In [4]: df = pd.read_sql("SELECT * FROM tax_990 LIMIT 10", conn)
        df
```

Out[4]:

	@binaryAttachmentCnt	@documentId	@referenceDocumentId	@referenceDocumentName	@softwareId
0	0.0	RetDoc1038000001	RetDoc1044400001	None	NaN
1	0.0	RetDoc1038000001	RetDoc1044400001	None	NaN
2	0.0	RetDoc1038000001	RetDoc1044400001	None	NaN
3	0.0	RetDoc1038000001	RetDoc1044400001	None	NaN
4	0.0	990	None	None	12000057.0
5	0.0	RetDoc1038000001	RetDoc1044400001	None	NaN
6	0.0	RetDoc1038000001	RetDoc1044400001	None	NaN
7	0.0	RetDoc1038000001	RetDoc1044400001	None	NaN
8	0.0	DOC260082834ID990	None	None	NaN
9	0.0	RetDoc1038000001	RetDoc1044400001	None	NaN

10 rows × 972 columns

Figure 14: Example of how to query an entire database table using Pandas

The query used in Figure 14 was “SELECT * FROM tax_990 LIMIT 10”. A *SELECT* statement was a statement to select the column name listed after that, and in this example the column name is an asterisk. An asterisk (*) denoted for any name, meaning this query would return all columns existing in the table *tax_990EZ*. The *LIMIT* keyword indicated that the query would only want the top number of rows in the table, and in this example, the limit was 10.

Performing a query for specific columns; however, is slightly more complicated. A schema table containing table’s information was provided as the project's deliverable for users to know which column name they were looking for. A part of the schema table is shown in Figure 15.

Based on form 990 for tax year 2013, these are the fields I would need. Please see the	XML Tags 990
PART IX Functional expenses	
All columns (A) to (D) for each row 1 to 24e.	
1A Grants and other assistance to org and gov	GrantsToDomesticOrgsGrp-TotalAmt
1B Grants and other assistance	GrantsToDomesticOrgsGrp-ProgramServicesAmt
2A Grants and other assistance to individual	GrantsToDomesticIndividualsGrp-TotalAmt
2B Grants and other assistance to individual	GrantsToDomesticIndividualsGrp-ProgramServicesAmt
3A Grants and other assistance to foreign entities	ForeignGrantsGrp-TotalAmt
3B Grants and other assistance to foreign entities	ForeignGrantsGrp-ProgramServicesAmt
4A Benefits paid to or for members	BenefitsToMembersGrp-TotalAmt
4B Benefits paid to or for members	BenefitsToMembersGrp-ProgramServicesAmt
5 Compensation of current officers,etc	CompCurrentOfcrDirectorsGrp-TotalAmt
5 Compensation of current officers,etc	CompCurrentOfcrDirectorsGrp-ProgramServicesAmt
5 Compensation of current officers,etc	CompCurrentOfcrDirectorsGrp-ManagementAndGeneralAmt
5 Compensation of current officers,etc	CompCurrentOfcrDirectorsGrp-FundraisingAmt
6 Compensation not included above to disqualified persons	CompDisqualPersonsGrp-TotalAmt
6 Compensation not included above to disqualified persons	CompDisqualPersonsGrp-ProgramServicesAmt
6 Compensation not included above to disqualified persons	CompDisqualPersonsGrp-ManagementAndGeneralAmt
6 Compensation not included above to disqualified persons	CompDisqualPersonsGrp-FundraisingAmt
7 Other salaries and wages	OtherSalariesAndWagesGrp-TotalAmt
7 Other salaries and wages	OtherSalariesAndWagesGrp-ProgramServicesAmt
7 Other salaries and wages	OtherSalariesAndWagesGrp-ManagementAndGeneralAmt
7 Other salaries and wages	OtherSalariesAndWagesGrp-FundraisingAmt
Pension plan accruals and contributions	PensionPlanContributionsGrp-TotalAmt

Figure 15: Snippet of the Schema Table from Form 990 Part X

Since all the column names were listed in the schema table, the table was downloaded as a CSV file, which led to the creation of a list of required columns. Due to some column names in the crucial schema table having some unnecessary white spaces, the *strip* string function was used to remove them; see Figure 16.

```
In [6]: # setting up to extract crucial data for 990
        df = pd.read_csv('crucial_schema.csv')
        schema_990 = list(df['XML Tags 990'].dropna())
```

```
In [8]: # Form a list of column names for SELECT query for 990
        cols = ""
        for name in schema_990:
            cols += "`" + name.strip() + "`"
            if name != schema_990[-1]:
                cols += ", "
```

Figure 16: Column names extracted and formed into query format

Another use-case for the database was to filter out companies matching EINs given by the client. The list of required EINs was given as a text file, so a script was written to extract and convert to a series of strings that would match SQLite condition format. Now that all the parts of the query were created, the query was performed through Pandas to retrieve the DataFrame, as shown in Figure 17. A more generic example of the structure and output of a Pandas SQL query is shown in Figure A5 in the Appendix.

```
In [9]: # load in interesting EIN from txt file
        EIN = np.loadtxt("./unique_EIN.txt", dtype=int)
        # Convert Array into string tuple for SQL Query
        EIN_as_string_tuple = str(tuple(EIN))

In [10]: # perform query
         frame = pd.read_sql('SELECT DISTINCT ' + cols + ' FROM tax_990 WHERE `Filer-EIN` in ' + EIN_as_string_tuple, conn)
         frame
```

Out[10]:

	Filer-EIN	Filer-BusinessName-BusinessNameLine1	Filer-BusinessName-BusinessNameLine2	Filer-USAddress-AddressLine1	Filer-USAddress-CityNm	Filer-USAddress-City	Filer-USAddress-StateAbbreviationCd	Filer-USAddress-ZipCode
0	462546411.0	SALISBURY-ROWAN COUNTY CONVENTION	VISITOR BUREAU INC	244 E INNES ST STE 120	None	SALISBURY	None	28144.0
1	521057866.0	TOURISM COUNCIL OF FREDERICK COUNTYINC	None	151 S EAST STREET	None	FREDERICK	None	21701.0
2	462010960.0	MARSHFIELD CONVENTION & VISITORS BUREAU INC	None	700 S CENTRAL AVE	MARSHFIELD	None	WI	54449.0
3	10775286.0	CAMBRIDGEGUERNSEY COUNTY VISITORS &	CONVENTION BUREAU	627 WHEELING AVE NO 200	CAMBRIDGE	None	OH	43725.0

Figure 17: Example code of how to extract list of EINs and form query through Pandas

Adding New Fields to Queries

Sometimes, users may need to add subsequent columns to the query. This could be done by appending the column name to the *schema_990* list (Figure 15) in the example query.
In order to find the specific column name, a pragma table info was created to search for the exact column name; see Figure 18.

In [12]:
```
# search column in the table using keyword
fields = "Yr"
table_info = pd.read_sql("pragma table_info('tax_990')", conn)
containingFields = table_info.loc[table_info['name'].str.contains(fields, case=False)]
containingFields
```

Out[12]:

	cid	name	type	notnull	dflt_value	pk
44	44	CYRevenuesLessExpensesAmt	REAL	0	None	0
116	116	FormationYr	REAL	0	None	0
196	196	PYRevenuesLessExpensesAmt	REAL	0	None	0
275	275	TaxYr	REAL	0	None	0
664	664	PayrollTaxesGrp-FundraisingAmt	REAL	0	None	0
665	665	PayrollTaxesGrp-ManagementAndGeneralAmt	REAL	0	None	0
666	666	PayrollTaxesGrp-ProgramServicesAmt	REAL	0	None	0
667	667	PayrollTaxesGrp-TotalAmt	REAL	0	None	0
674	674	PermanentlyRstrNetAssetsGrp-BOYAmt	REAL	0	None	0
675	675	PermanentlyRstrNetAssetsGrp-EOYAmt	REAL	0	None	0
765	765	TemporarilyRstrNetAssetsGrp-BOYAmt	REAL	0	None	0
766	766	TemporarilyRstrNetAssetsGrp-EOYAmt	REAL	0	None	0
938	938	PayrollTaxesGrp	TEXT	0	None	0

Figure 18: Example code of how to search for specific column name

In Figure 18, a search for the tax year field was conducted. By using the keyword “Yr” a list of all the fields containing the substring “yr” was extracted from the *tax_990* table, and the tax year field in the table is “TaxYr”. Now that the column name was identified, the next step was to add this column name into the list of schemas created earlier in Figure 15. This can be done easily by appending “TaxYr” into the list, and a query can be performed to extract the data from the database. SQLite accepts most queries from other SQL frameworks, and more information for specific SQLite keywords can be found in online documentation [18]. Further Python code detailing the structure and syntax of adding new fields to SQL queries can be found in Figure A6 in the Appendix.

Developer's Manual

Adding New Records

AWS

As explained before, the data within AWS is no longer updated by the IRS but remains to be publicly available. However, this project utilized a subset of data from AWS (about 10,000 filings per year 2012-2020). In addition to these 10,000 files, all 990 and 990-EZ files for the tourism offices of interest were accounted for. To download more filings from AWS, use the aws_download.py script. The Python script takes in four parameters specified in the comments in the file. The arguments are the year (or range of years), maximum number of files to download, output directory name, and the return type(s) to look for (Form 990 or Form 990-EZ). Next, to process the XML data and convert to CSV files, use the same code as described in the Implementation section to process any new XML downloads. The resulting output will be a CSV file containing all processed XML in the specified directory in the code.

IRS

Adding new data – from subsequent tax filing years, or newly released data from the study window (2012-2020) – is the most crucial aspect of continuing the work on this project. This will first require leveraging the scraping and unzipping programs that were used in this project for the extraction of data from the IRS website. Upon acquiring this raw data, similar steps will need to be taken in order to determine the filing year, EINs, and return types of the new data. This will allow particular data of interest to be determined based on EIN, and will serve to initially label the XML files such that they can be processed. Any new XML data that is part of a filing year from 2012-2020 can be processed using the exact same code and alignment tables that were used to process data in this project. New XML data from 2021 and beyond will require slight adjustments to the processing techniques to ensure that the data is properly ingested, XML tag sets are effectively captured, and the database is properly populated.

The first crucial step that will need to be taken is determining whether the crucial information in the new XML data can be found under the same tag names. This project's study window accounts for the three crucial data attributes, filing year, return form type, and EIN, through capturing the XML tags *TaxYr*/*TaxYear*, *ReturnType*/*FormType*, and *EIN* respectively. If any of these change, the initial XML processing code will need to be tweaked in order to capture this data before moving on to subsequent processing steps. These changes will need to be updated in the *get_EIN_and_ReturnType_and_Yr* function in the 'Processing.ipynb' file for the IRS website data.

Beyond this initial phase, the major processing steps for new years' 990 and 990-EZ data will be primarily the same. The final difference that future developers must account for is the alignment of XML tags for 2021 and beyond to the base schema for 2013. This will involve using the *get_tag_set* (discussed in the Implementation section) in order to acquire XML tag sets for subsequent years and compare them to the tag set for 2013's Form 990 and Form 990-EZ. The schema alignment will then need to be performed between the years. This will require using primarily manual methods to determine schema name changes, before using the *validate_schema* function (contained in "Tag_Sets_990.ipynb") to find files containing the potentially aligned tags. Upon finding these files, future developers will need to manually validate that the tags contain the same data, and will need to update "alignment_table.csv" accordingly with the aligning tags from each new year. Once these adjustments are made to the schema alignment, any new developer will be ready to add new years' data to the 990 filings database.

Conclusions and Lessons Learned

Throughout the project, the team experienced many different challenges that had to be solved and learned from along the way. This helped to bolster the team's technical and communication skills in order to complete the project objectives in a timely fashion; see Table 11. One of the main lessons learned was that communication with the client during difficulties was very important. This was especially true when it came to clearing up confusion or obtaining clarification.

Timeline

Week	**Milestone(s)**
1/24 – 1/30	Team discussed requirements in the initial meeting with the client.
1/31 – 2/6	Team read and understood project documentation. Tuned up project agreement and submitted OKRs
2/7 – 2/13	Downloaded/accessed data files, keeping them organized. Met with the client to discuss current progress
2/14 – 2/20	Presentation #1 Developed Jupyter Notebooks to align data and convert it into CSV/JSON
2/21 – 2/27	

2/28 – 3/6	Met once with client to discuss issues with data alignment Worked on database design (breaking it down into various tables) Populated initial database with EOBMFs; used queries to filter out tourism offices
3/7 – 3/13	SPRING BREAK
3/14 – 3/20	Prepared a filtered version of the EOBMFs containing data solely on tourism offices Continued work on data alignment
3/21 – 3/27	Continued work on database design and data alignment Met with the client to discuss issues with Google Drive and schedule forms
3/28 – 4/3	Completed 990-EZ schema alignment Developed XML to CSV conversion script Developed XML to JSON conversion script
4/4 – 4/10	Presentation #2 Completed report outline and began filling out sections for the interim report submission Met with the client to discuss key schema, database documentation, and schema validation
4/11 – 4/17	Completed interim report submission Began development of alignment scripts
4/18 – 4/24	Completed alignment table and alignment scripts Transitioned project work to ARC in order to run all files through alignment, conversion, and database population scripts Met with client to discuss final alignment and database questions
4/25 – 5/1	Deployed database Performed preliminary database analyses Developed user guide Filtered tourism offices from the database and convert them to CSV and JSON, populated the client's desired workspace with the converted files

	Final meeting with the client
5/2 – 5/4 (Project Deadline)	Final presentation Submitted final report

Table 11: Timeline

Problems

One of the major problems that the team encountered was creating a universal schema for the various different years and forms. To support the ease of use of the 990 database, the team wanted to ensure that if a variable name changed, all different variable names would be stored under the same column (e.g., TaxYear in 2012 and TaxYr in 2013 should be stored under the same column). It was extremely difficult determining how to treat the nested structure of the XML, especially with outer tags that are identical but refer to different information (EOY in CashSavingsAndInvestmentsGrp vs. EOY in TotalAssets).

The other key issue that the team experienced concerned the massive amount of 990 filing data contained in the project's years of study. As discussed in the objectives, changes in the introduction, the original shared Google Drive workspace used by the team and the client exceeded its file capacity when a few years worth of raw XML files were added to the Drive. This resulted in the abandonment of working on shared notebooks in Google Colab, and the transition to primarily local development. However, the individual machines of the team members were still constrained and could not store the entire dataset at any given time. This brought about difficulties when it came to processing data files from every year, and massively slowed down development.

Solutions

To solve the problem of creating the schema, the team went about creating a way to automate the acquisition of the different tags and corresponding information using 1000 records for each year to get a schema estimate. This raised another problem regarding whether the automated process outputs a complete and correct schema. This problem eventually required the team to manually create a sample schema for the 2013 990-EZ form, providing an exact reference to the form to compare to the automated schema creation process. The sample schema was created from using a GitHub repository that contained a mostly complete schema and manually filling in the gaps of tags that were not available in the repository [11]. This approach proved successful and useful in showing how certain non-intuitive data fields were represented. This also showed another issue of the sheer amount of data and fields stored in the different schedules, which accounting for would have prevented the project from being completed in the given time frame. The client was fine with this since many schedules seemed irrelevant to the primary focus of his research.

To solve the problem of identical tags, the team used an approach of treating the nested structure of the XML as a path in a similar way that directories use (CashSavingsAndInvestmentsGrp-EOY, TotalAssets-EOY). This approach also has the added benefit of more directly showing how the information is contained.

Future Work

Due to the various limitations and scope changes associated with this project, there are a variety of future tasks that could be carried out to improve or expand upon this project's database. Some of the data that was omitted from this database, including the various Schedule forms, along with the Form 990-PF data, can be added to the database. Populating the database with further data from the Form 990 Schedules will require new developers to assess the best techniques for capturing and storing the various types of data held in the schedule forms. Meanwhile, processing the 990-PF forms will require an entire new process of schema alignment to take place, and will require a new table in the database. Although these 990-PF forms are not applicable to the tourism office research that inspired this project, they could provide valuable insights into the financial dealings of many other non-profits or tax-exempt organizations.

Some other possibilities for future work concern the nature of this project's database. Due to the intense amount of work required by the team to understand the Form 990 data used for this project, it was difficult to address the possibility of using different database formats. Through peer, mentor, and client feedback, some other database structures were considered, but not implemented for this project. Some of these ideas include MongoDB and PostgreSQL [19, 20]. MongoDB has the capability to handle data in a schema-independent manner, which would be incredibly useful for adding new years' data to this project's database [19]. However, future developers will need to be wary with this database framework, as the lack of schema alignment does pose potential disadvantages when it comes to querying data for filings from many different years. PostgreSQL databases have a few different features that would add value to a 990 filings database. One such advantage is the capability of georeferencing database entries (based on address data present in Form 990) such that analyses can be performed in a geospatial manner. This will allow 990 filings to be compared between organizations in the same city, state, or region. Alternatively, this can be used to assess the differences between various locations using non-profit or exempt organization financial data as a metric.

Acknowledgements

Client: Professor Florian Zach, Ph.D., Howard Feiertag Department of Hospitality and Tourism Management, Pamplin College of Business
Mentor: Dr. Edward A. Fox
Virginia Tech Advanced Research Computing (ARC)

References

[1] Internal Revenue Service, "About form 990, return of organization exempt from Income Tax," *Internal Revenue Service*, 09-Jul-2021. [Online]. Available: https://www.irs.gov/forms-pubs/about-form-990. [Accessed: 26-Apr-2022].

[2] Internal Revenue Service, "About Form 990-ez, short form return of organization exempt from Income Tax," *Internal Revenue Service*, 09-Jul-2021. [Online]. Available: https://www.irs.gov/forms-pubs/about-form-990-ez. [Accessed: 26-Apr-2022].

[3] Internal Revenue Service, "About Form 990-PF, Return of Private Foundation or Section 4947(a)(1) nonexempt charitable trust treated as a private foundation," *Internal Revenue Service*, 2022. [Online]. Available: https://www.irs.gov/forms-pubs/about-form-990-pf. [Accessed: 26-Apr-2022].

[4] Internal Revenue Service, "About schedule H (Form 990), Hospitals," *Internal Revenue Service*, 10-Jun-2021. [Online]. Available: https://www.irs.gov/forms-pubs/about-schedule-h-form-990. [Accessed: 26-Apr-2022].

[5] Internal Revenue Service, "About schedule O (Form 990), supplemental information to Form 990 or 990-EZ," *Internal Revenue Service*, 2022. [Online]. Available: https://www.irs.gov/forms-pubs/about-schedule-o-form-990-or-990-ez. [Accessed: 26-Apr-2022].

[6] Amazon Web Services, "IRS 990 filings," *IRS 990 Filings - Registry of Open Data on AWS*, 2021. [Online]. Available: https://registry.opendata.aws/irs990/. [Accessed: 26-Apr-2022].

[7] Internal Revenue Service, "Form 990 series downloads," *Internal Revenue Service*, 2022. [Online]. Available: https://www.irs.gov/charities-non-profits/form-990-series-downloads. [Accessed: 26-Apr-2022].

[8] Internal Revenue Service, "Exempt Organizations Business Master File Extract (EO BMF)," *Internal Revenue Service*, 11-Apr-2022. [Online]. Available: https://www.irs.gov/charities-non-profits/exempt-organizations-business-master-file-extract-eo-bmf. [Accessed: 26-Apr-2022].

[9] Amazon Web Services, "A - CLI," *Amazon*, 2022. [Online]. Available: https://aws.amazon.com/cli/. [Accessed: 26-Apr-2022].

[10] Amazon Web Services, "Tools to Build on AWS," *Amazon Web Services*, 2022. [Online]. Available: https://aws.amazon.com/tools/. [Accessed: 26-Apr-2022].

[11] CharityNavigator, "Charitynavigator/990_metadata: Code for adding metadata to IRS 990 change logs.," *GitHub*, 26-Jul-2018. [Online]. Available: https://github.com/CharityNavigator/990_metadata. [Accessed: 26-Apr-2022].

[12] SQLite, "What is SQLite," *Sqlite*, 2022. [Online]. Available: https://www.sqlite.org/index.html. [Accessed: 26-Apr-2022].

[13] Pandas, "Pandas," *pandas*, 02-Apr-2022. [Online]. Available: https://pandas.pydata.org/. [Accessed: 26-Apr-2022].

[14] M. Blech, "Xmltodict," *PyPI*, 2022. [Online]. Available: https://pypi.org/project/xmltodict/. [Accessed: 26-Apr-2022].

[15] Python Software Foundation, “SQLITE3 - DB-API 2.0 interface for SQLite databases,” *sqlite3 - DB-API 2.0 interface for SQLite databases - Python 3.10.4 documentation*, 26-Apr-2022. [Online]. Available: https://docs.python.org/3/library/sqlite3.html. [Accessed: 26-Apr-2022].
[16] Requests, “HTTP for humans,” *Requests*, 2022. [Online]. Available: https://docs.python-requests.org/en/latest/. [Accessed: 26-Apr-2022].
[17] Python Software Foundation, “IO - core tools for working with streams,” *io - Core tools for working with streams - Python 3.10.4 documentation*, 26-Apr-2022. [Online]. Available: https://docs.python.org/3/library/io.html. [Accessed: 26-Apr-2022].
[18] SQLite, “SQLite Keywords,” *SQLite*, 2022. [Online]. Available: https://www.sqlite.org/lang_keywords.html. [Accessed: 26-Apr-2022].
[19] MongoDB, “MongoDB Atlas: Cloud Document Database,” *MongoDB*, 2022. [Online]. Available: https://www.mongodb.com/cloud/atlas/lp/try-cloud. [Accessed: 26-Apr-2022].
[20] PostgreSQL, “PostgreSQL: The World's Most Advanced Open Source Relational Database”, *PostgreSQL*, 2022. [Online]. Available: https://www.postgresql.org/. [Accessed: 26-Apr-2022].

Appendix

```
In [4]: # check number of entries for each table to see if there's any difference
        regions_table = pd.read_sql('SELECT * FROM eo_regions', conn)
        states_table = pd.read_sql('SELECT * FROM eo_states', conn)
        regions_sz = len(regions_table)
        states_sz = len(states_table)
        print('regions count:\t' + str(regions_sz))
        print('states count: \t' + str(states_sz))
        print('diff count:   \t' + str(abs(regions_sz - states_sz)))

        regions count:  1800540
        states count:   1796491
        diff count:     4049
```

Figure A1: Analyzing the number of entries in the States and Regions files

```
In [9]: print('regions:  ' + str(regions_sz))
        print('states:   ' + str(states_sz))
        print('combined: ' + str(len(combined_table)))

        regions:  1800540
        states:   1796491
        combined: 1800540
```

Figure A2: Analyzing the number of entries in the States, Regions, and Union (combined) files

```
def equal_criteria(items, col_name):
    retval = ''
    for item in items:
        retval += col_name + ' == ' + "'" + item + "'"
        if item != items[-1]:
            retval += ' OR '
    return retval
```

```
def name_filter(items, col_name):
    retval = ''
    for item in items:
        retval += col_name + " LIKE '%" + item + "%'"
        if item != items[-1]:
            retval += ' OR '
    return retval
```

```
starter_c = ''
for word in starterkeywords:
    starter_c += "NAME LIKE '" + word + "%'"
    if word != starterkeywords[-1]:
        starter_c += ' OR '
print(starter_c)
```

Figure A3: Functions/Logic for the structuring of queries used to filter tourism offices from the EOBMFs

```
In [10]: def filter_list(items, keys):
             retval = []
             for key in keys:
                 retval = retval + [item for item in items if key in item]
             return retval
```

Figure A4: filter_list function being used to filter out list of column names

```
In [45]: df = pd.read_sql(query, conn)

In [46]: df
```

Out[46]:

	Filer-EIN	CashNonInterestBearingGrp-BOYAmt	CashNonInterestBearingGrp-EOYAmt	SavingsAndTempCashInvstGrp-BOYAmt
0	912047382	200916.0	195731.0	NaN
1	426053835	NaN	NaN	15236.0
2	363206134	122273.0	114228.0	NaN
3	464203790	NaN	1368507.0	NaN
4	314341790	3257.0	6773.0	16010.0
...	...	...	...	...
612	621869198	5561.0	971.0	181535.0
613	237333783	NaN	NaN	310678.0
614	911862145	103579.0	42639.0	100.0
615	133975743	151978.0	120835.0	NaN
616	30434419	86490.0	90311.0	NaN

617 rows × 71 columns

Figure A5: Code and result of custom query line for example queries

In [47]:
```
column_names = "`Filer-EIN`, `OtherExpensesGrp-TotalAmt`, "
for row in rows:
    column_names += "`" + row + "`"
    if row != rows[-1]:
        column_names += ", "
```

In [48]:
```
query = "SELECT " + column_names + " FROM tax_990EZ"
print(query)
```

In [49]:
```
df = pd.read_sql(query, conn)
```

In [50]:
```
df
```

Out[50]:

	Filer-EIN	OtherExpensesGrp-TotalAmt	CashNonInterestBearingGrp-BOYAmt	CashNonInterestBearingGrp-EOYAmt	S
0	912047382	NaN	200916.0	195731.0	
1	426053835	NaN	NaN	NaN	
2	363206134	NaN	122273.0	114228.0	
3	464203790	NaN	NaN	1368507.0	
4	314341790	NaN	3257.0	6773.0	
...	...	...	...	...	
612	621869198	NaN	5561.0	971.0	
613	237333783	24108.0	NaN	NaN	
614	911862145	NaN	103579.0	42639.0	
615	133975743	498.0	151978.0	120835.0	
616	30434419	-571.0	86490.0	90311.0	

617 rows × 72 columns

Figure A6: Code and result pertaining to the addition of new fields to SQL queries

```
def key_adder(dictionary, finalkeys):
    if isinstance(dictionary, list):
        for elem in dictionary:
            if isinstance(elem, dict):
                key_adder(elem, finalkeys)
    else:
        for key in dictionary.keys():
            if isinstance(dictionary[key], dict):
                key_adder(dictionary[key], finalkeys)
            else:
                finalkeys.append(key)


def get_tag_set(files, file_year, year, finalkeys):
    missing_return_tag = 0
    for file in files:
        filename = file_year + '/' + file
        with open(filename, encoding='utf-8') as xml_file:

            data_dict = xmltodict.parse(xml_file.read())
            xml_file.close()

            # generate the object using json.dumps()
            # corresponding to json data

            json_data = json.dumps(data_dict, indent=4)

            # Write the json data to output
            # json file
            """
            with open("data.json", "w") as json_file:
                json_file.write(json_data)
                json_file.close()
            """
            json_dict = json.loads(json_data)

            # Not present in some files apparently? Beginning 2019
            if 'Return' in json_dict:
                key_adder(json_dict['Return'], finalkeys)
            else:
                # Count the number of filings missing their 'Return' tag (return this count)
                missing_return_tag += 1
            finalkeys = list(set(finalkeys))
            xml_file.close()


    tagSet = set(finalkeys)
    listOfTag = list(tagSet)

    sorted_tags = sorted(listOfTag)

    listOfTag = np.reshape(listOfTag, (-1, 1))
    sorted_tags = np.reshape(sorted_tags, (-1, 1))

    # Output tags in alphabetical order
    with open('./' + year + '_sorted.csv', 'w') as f:
        write = csv.writer(f)
        write.writerows(sorted_tags)
        f.close()

    return missing_return_tag
```

Figure A7: Functions used to obtain XML tag sets

```
In [8]: # Perform table union
        final_frame = pd.concat(dfs)

In [9]: len(list(final_frame))

Out[9]: 972

In [10]: # Convert dataframe to SQLite table "tax_990"
         final_frame.to_sql('tax_990', conn, if_exists='append', index=False)

In [11]: list(final_frame)
```

```
'ActivityOrMissionDesc',
'ActivityOther-Desc',
'ActivityOther-ExpenseAmt',
'ActivityOther-GrantAmt',
'ActivityOther-RevenueAmt',
'AddressChangeInd',
'AddressPrincipalOfficerUS-AddressLine1',
'AddressPrincipalOfficerUS-City',
'AddressPrincipalOfficerUS-State',
'AddressPrincipalOfficerUS-ZipCode',
'AllAffiliatesIncludedInd',
'AmendedReturnInd',
'BackupWthldComplianceInd',
'BusinessRlnWithFamMemInd-#text',
'BusinessRlnWithFamMemInd-@referenceDocumentId',
'BusinessRlnWithOrgMemInd-#text',
'BusinessRlnWithOrgMemInd-@referenceDocumentId',
'BuildTS',
'BusinessRlnWithFamMemInd',
```

Figure A8: Union all DataFrames and convert the unioned DataFrame to table

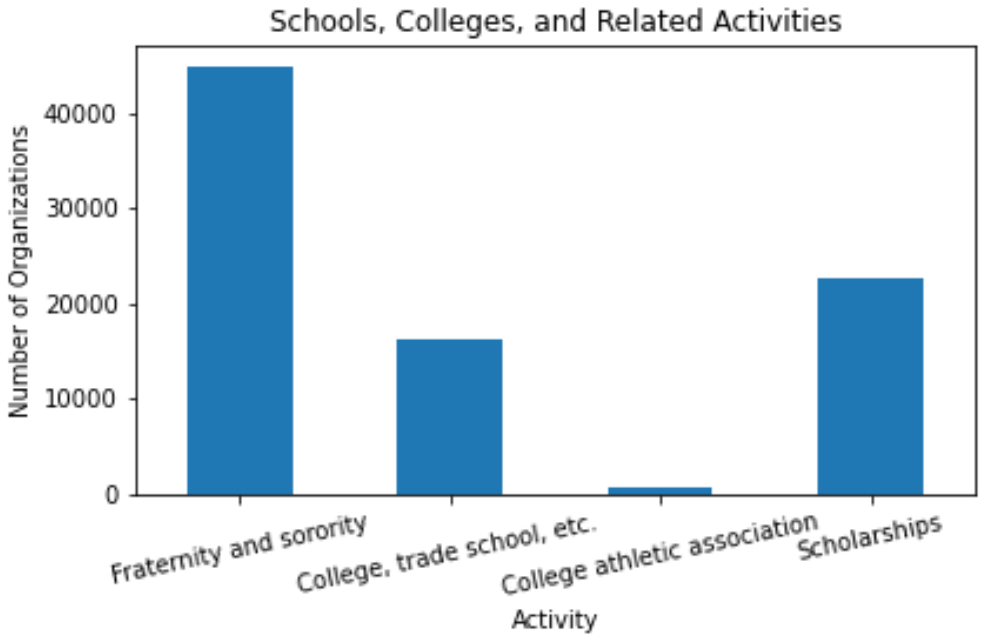


Figure A9: Count of organizations involved in college related activities

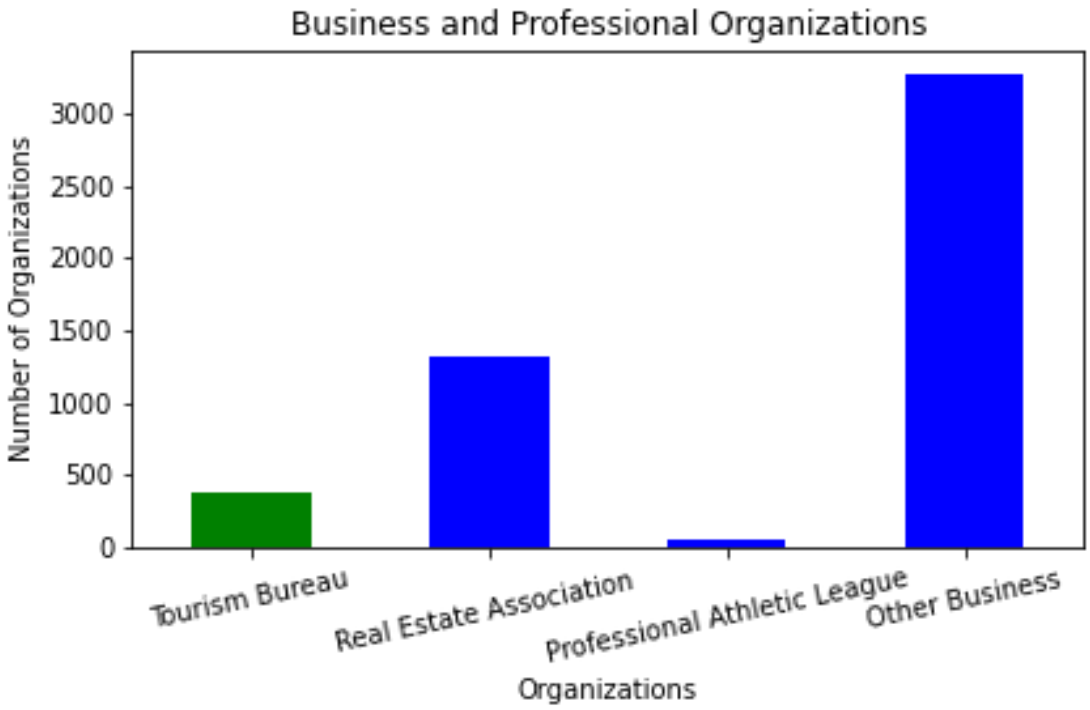


Figure A10: Count of organizations different organization types based on their activity