Summative Assessment

Section 1: Theory supported by code samples

Section 1 - A

The area of the program relating to saving, extracting and converting the client's data was redesigned to allow concurrency with python threads [1] (appendix A.3). The process identified along with other file managing processes required significant computing time in comparison to other process areas such as data cleaning and generating graphical visualisations. This was therefore an area that required improvement resulting in potential performance gains. [2]

There were specific issues to be considered when implementing python threading in the given process, as the existing program flow (appendix A.2) required certain actions to be carried out in sequential order. In the existing program flow, converted files from csv to JSON must be inserted into their respective collections before extracting and converting individual files into their own pandas dataframes, and the conversion from JSON to pandas dataframe of extracted collections must happen before data cleaning can be processed. If the threading processes are still running before the next process begins, this will create an error and terminate the program. This is due to empty collections in the database creating an issue where there are no available collections to be extracted for the next process.

In order to implement threading in the section of interest, pythons threading module must be imported to access its functions. The process of conversion and insertion of JSON data into their respective collections consisted of a sequential operation on each individual file. This process also applies to converting JSON to pandas dataframe. (appendix A.3) In order to achieve threading, the data processing for each file was made into their own functions, assigning them as individual functions enabling them to be called and grouped when implementing the threading.Thread class. [3] (appendix A.1) The threading process was grouped into two. The first process was the conversion and storage of JSON into the database. To start threading, the threading.Thread class was applied to each function and assigned as separate variables, applying the .start() method to each variables starts the thread's activity and applying the .join() method after the start process will ensure all threads are terminated before the next process begins. [3] (appendix A.1) The same method was applied to the second process of extracting JSON and converting into pandas dataframe, ensuring the threading.Thread class for the next group of functions were assigned after the previous .join() method was called. This handled the issue specified on processes overlapping by ensuring the threading processes completed before continuing.

Successfully implementing threading allows the sequential process in the existing program flow to be processed concurrently; file storage, conversion and extraction are executed simultaneously resulting in performance gain by optimizing the usage of central processing unit cores. [4] (appendix A.1, A.4)

Section 1 - B

To begin the development of the tkinter GUI interface, one level hierarchy was used for all the main controls, resulting in simplicity when navigating around the program to produce the client's requirements. [5] The controls each have their own segments, these are identified by the use of a labelled frames. The design consistency of the frame alignment and size is maintained throughout the program in order to develop user usage pattern when discovering its interaction. [6] (appendix B.1, B4)

When the user opens the program for the first time, a message highlighted as 'No File Loaded' indicates the user to load a new dataset. The importance of visual feedback will provide encouragement to the user when interacting with the program controls, [7] when clicking on other buttons prior to file loading, the program will prompt different messages until a file has been loaded. [5]Once a file is loaded, a message confirmation will be displayed where the visualization controls become accessible (appendix B.2). When saving the dataset, the next time the program is ran, the user can load the previously saved dataset with the "load prepared dataset" button. (appendix B.1, B.5)

The output requirement to generate the mean, mode and median inspection score per year are produced within the Inspection score visualization box. There are two controls, a drop-down box that lists either Vendor's Seating or Zip Code and their statistical requirement and a button that loads the visualization in the client frame instantly. [6] The user selects and loads their choice and a table is produced displaying the statistical values for each year. Scroll bars are implemented with the table to view every value. Drop down menu was selected over entry box as options can be pre-defined displaying all the possible choices to the user. Radio buttons may have been used however due to the lengthy option descriptions, this would impact the minimalistic aesthetics of the GUI design by creating cluttered text. [6] (appendix B.1, B.3, B5)

In order to produce a suitable graph which displays the number of establishments that have committed each type of violation, radio buttons enable the user to select the available violations range and generate the graphical visualization in a separate window with the 'Load Graph Button'. Radio buttons are used instead of dropdown menu to allow the user to produce multiple graph visualizations without the need to repeatedly reopen a menu. [6] (appendix B.1, B5)

In order to produce a visualisation to demonstrate if there is any significant correlation between the number of violations committed per vendor and their zip code, the use of radio buttons, an entry box and a button to load the graph was implemented. The user selects the zip code format and manually enters the name of the vendor then clicks the "Load Graph" button to display the graphical visualization in a separate window. An entry box was used instead of a radio button or dropdown box due to large amount of vendor's available making it unsuitable for pre-defined choices. [6] (appendix B.1, B5)

Section 1 - C

The most effective in terms of ‘speed of development’ is Python due to the following factors that will be discussed: Dynamic typing versus Static Typing and the comparison between Syntax requirements to achieve the same results. [8]

Firstly, variable declaration is dynamic in Python unlike Java where it is statically typed, resulting in a linear approach reducing the amount of code and more flexibility when developing. [9] For example, local variables don’t need to be initialized as they are assumed at run-time, enabling the ability to generate a variable at any given time which is useful in data cleaning when implementing checks. (appendix C.1) In addition, when creating variables in Python, variables are not fixed to one data type, they can be reassigned to different data types. In Java this is not the case, variables must be initialized before they can be used and are fixed to the datatype declared, resulting in a non-linear approach and additional rules to obey.

Python has a clear advantage in performing similar tasks. This is because the syntax is more concise in comparison to Java, resulting in faster development and testing of programs as you go. [9] Python code can be executed in the terminal whereas Java requires the need to compile the entire program in order to run. [10] For example, when developing in Jupyter Notebook, during data cleaning many checks are performed throughout development to preview dataframes and anomalies, this makes it easier to output quick comparisons and display implementations. (appendix C.2) This isn’t the case for Java, as it does not have a command line interpreter. In order to perform the same task, you must write a complete program and then compile it which consists of a class and a main function to indicate Java where to begin. [11]

A Python program may take longer to run due to its interpreted language in comparison to Java’s compiled language. However, for a developer whose importance is the speed of development to produce a working program, one can argue the time lost during the execution of a program is recovered by the time saved in the development process. These factors lead to a conclusion that Python is the most effective language for this purpose.

Section 2: Design decisions supported by code samples

Section 2: A

The selected data format used to meet the client's data manipulation requirement was JSON. The choice of selection was due to incorporating MongoDB to enable data storage and backup, and pandas for data cleaning. [12]

Saving documents in MongoDB are stored as BSON which is a binary JSON. Representing the data as JSON can be natively stored in MongoDB. The advantage of using JSON is the similarity with python dictionaries, as text in JSON is done through quoted-string which contains a value in key-value mapping within {}, [16] (appendix D.6) enabling ease of reading, converting and storing of data with python. [13][15] (appendix D.1) However, the disadvantages when using JSON with MongoDB was the speed of processing the data stored, as it was slower in comparison to using solely pandas when handling .csv. This had an impacted development as testing GUI integration required more computing time.

The data manipulation process required .csv files to be converted into JSON for storage, the compatibility with python and pandas involved two methods for conversion, one to read the file [14] and one to convert the file. [13] A conversion function was created containing methods which were applied to individual files. (appendix D.1, D.3) Enabling the connection to MongoDB client where a new database is created (appendix D.2) allows JSON data to be stored and extracted for converting into pandas dataframe, which involves pandas and pymongo methods. [17] [18] (appendix D.3, D.4) After cleaning the data with pandas to meet the clients requirements, the data is stored and backed up. This required a method of converting a dataframe back into JSON enabling data to be saved into the mongoDB database. The backup data can now be loaded the next time the program is opened using the same method for extracting collections. (appendix D.5)

Section 2: B

To achieve the client's 3rd requirement consisted of pre-processing techniques to ensure the data is prepared. [19] The libraries imported were numpy and pandas for dataframe manipulation and tkinter for GUI.

Data integration consisted of merging the inspections and inventory dataset on the inspections dataframe 'RECORD ID' column. The inventory dataset filled the missing Zip Code values for the inspection's dataset. [20] (appendix E.2) After merging, duplicated data and rows containing missing values within important columns were removed (appendix E.1, E.2).

The next step required the use of regular expression to extract information within the parenthesis from the 'PE DESCRIPTION' values and to replace a typo in a description, [21] [22] this resulted in 16 unique 'PE DESCRIPTION'. (appendix E.3, E.7) Conversion of 'ACTIVITY DATE' into date format to extract the yearly information into a new column. [23] (appendix E.4, E.7) Lastly, the dataset is filtered to 'ACTIVE' only in the 'PROGRAM STATUS' column. [24] (appendix E.5, E.7)

The dataset is now prepared for manipulating (appendix E.7), to display the statistical inspection score per year for each type of vendor's seating and for each Zip Code, the use of the groupby method was applied to produce a multi-index dataframe containing Zip Codes/Vendor's seating [25] and the statistics for each year using numpy and pythons mathematical operations. [26] (appendix E.6, E.8)

Implementing the GUI with tkinter required unstacking to provide the correct format to display in a treeview table, [27] displaying columns for Zip Codes and years containing the required values. (appendix E.9) Lastly, the dataframe is converted into a list where columns and values are iterated and inserted into the table. [28] (appendix E.10) Treeview was used instead of Matplotlib due to the treeview format was able to display the entire pandas dataframe with horizontal and vertical scroll bars. [29]

Section 2: C

To meet the client's 4th requirement, the prepared dataset from the client's 3rd requirement 'df' was merged with the violation's dataset. Matplotlib bar charts were used to display the number of establishments which committed each type of violation. In addition, radio buttons in tkinter were implemented to group the data into ranges to make visualisation's feasible. Implementing Matplotlib over the treeview widget produced a more suitable visualization for data comparison for the client by enabling comparison of bar heights of multiple values.

To begin, the violations dataset was merged with 'df' on the 'df' dataframe key 'SERIAL NUMBER', [20] rows containing missing values in column 'VIOLATION CODE' were removed. [30] (appendix F.1) The .groupby() method was used to create a dataframe consisting of unique columns for violation codes and the number of establishments. [25] (appendix F.2) It was assumed that a discussion with the client to confirm the definition of establishment had taken place, as the names of each individual facility name, 'FACILITY NAME' column was used to define establishment. The dataframe was sorted from highest to lowest by violations count and then split into 5 range categories. (appendix F.4)

Creating a function called graphvis(): contains the code to generate a tkinter frame consisting of a bar chart of the client's requirement. [33] FigureCanvasTkAgg was imported in order to enable Matplotlib integration with tkinter. [31] The chart values displayed are dependent on the selected radio button which represent each range group of violation counts. [32] (appendix F.6) The x-axis displays the unique violations codes and the y-axis represents the number of establishments committed each type of violation. In order to initiate the function, a button was created, and the function was applied with a lambda expression, when 'load graph' is clicked the graph visualization is displayed in a separate window. (appendix F.6, F.7)

Section 2: D

The dataset consisting of all three datasets merged ‘prep_df’ was shaped and grouped into two dataframes for seaborn visualization. Seaborn bar chart was used to display vendor’s facilities locations and their number of violations, [34] queried by the client with a choice of ‘Grouped Locations’ or ‘Individual Locations’. (appendix G.1, G.3, G.8)

Creating a function called correlation_plot(): contains the code to implement button decisions, grabbing vendor name input and a button to load visualisations. When ‘Grouped Locations’ is selected the Zip Codes are grouped in relation to their first three digits. This is due to the first digit representing a certain group of US states and the second and third digits representing a region within that group. When ‘Individual Locations’ is selected the default zip codes are used to identify exact locations. [35] (appendix G.1)

The groupby method was applied to the dataframe ‘locations_grouping’ which varies between ‘group_zip’ and ‘prep_df’ , depending on the button of choice consisting of specified vendor’s facilities zip codes and the number of violations. (appendix G.2, G.3) The x-axis represents the Zip Codes for the vendor input and y axis represents the number of violations. (appendix G.7) For example, querying the vendor ‘subway’ shows no sign of correlation between Zip Codes and the number of violations committed as there is no noticeable correlation pattern in the distribution of data observed. However, there are tendencies for facilities in specific locations to have more violations such as group zip code 189 and 227 show a significant result in comparison to other locations. (appendix G.8)

Seaborn was used over the Matplotlib implementation in requirement 4 due to the categorical variable ‘Zip Codes’. When inserting into Matplotlib the numerical values were represented as a range from min to max rather than individual ‘Zip Codes’. This required an additional process to correct whereas seaborn automatically identifies the individual zip codes as categorial variables resulting in simple and fast implementation.

Section 3: Reflection on the ethics, moral and legal aspects

Section 3: A

Automated decision making introduces a large problem in our society that have real world consequences, as these decisions are strictly guided by algorithms trained on large datasets and experience, [36] they are susceptible to being biased. Algorithm bias can be categorised into three types- I will be focusing on selection bias. [37]

Selection bias in algorithms occurs when a group or individual is exposed to information decided by a dataset that differs from a population interest to another. [38]This creates an effect known as filter bubbles and echo chambers, limiting the topics that reach individuals by companies based on their data collected from interaction and interest. [39] This results in intellectual isolation [40] where an echo chamber is formed, where beliefs and opinion of likeminded people are only ever considered, alternative views are not. [38]

These effects can become damaging to individuals and even to a larger scale; an entire country by reinforcing extreme political views such as the U.S presidential election in 2016. Over 60% of U.S adults obtain news on social media platforms. [42] Facebook alone was exposed to have collected around 87 million profiles for Cambridge Analytica in a data scandal, which was claimed by a co-founder, Christopher Wylie, which was used to develop psychographic profiles of people and deliver pro-Trump propaganda. [41] It was found that Facebook pages of both extreme ends of the political spectrum were publishing and spreading false and misleading materials during the 2016 election campaign at a concerning rate. For instance, 38% of the content posted by popular right-wing pages and nearly 20% of the content posted by popular left-wing pages were misleading and/or false information. [43] As false information is spread echo chambers are reinforced, emphasising the narrow self-interest of individuals developing false beliefs further extending the filter bubble effect in the negative direction.

The data scandal demonstrated the capability of digital technologies in relation to the manipulation of people's decisions without acknowledgement and consent. This was a breach of Article 8 of the European Convention on Human Rights. [44] With the implementation of technology, is it acceptable for algorithms to dictate what we see based on our personal data? The things we are exposed to can influence our beliefs and actions. This is why it's important for legislation such as Data Protection Act 2018 [46] and GDPR [45] were put in place to protect individual's sensitive information from being used for unlawful manipulative purposes and to protect those with minimal knowledge in the area of subject.

References

[1]C. Schafer, *Python Threading Tutorial: Run Code Concurrently Using the Threading Module*. 2019.

[2]P. Boag, "15 User Interface Design Principles for Better Websites", *Paul Boag - User Experience Advice*, 2020. [Online]. Available: https://boagworld.com/design/user-interface-design-principles/#15_Prioritise_Performance. [Accessed: 08- Jan- 2021].

[3]"threading — Thread-based parallelism — Python 3.9.1 documentation", *Docs.python.org*. [Online]. Available: https://docs.python.org/3/library/threading.html. [Accessed: 8- Jan- 2021].

[4]"Python Programming/Threading - Wikibooks, open books for an open world", *En.wikibooks.org*, 2020. [Online]. Available: https://en.wikibooks.org/wiki/Python_Programming/Threading#:~:text=Threading%20in%20python%20is%20used,calls)%20at%20the%20same%20time.&text=Python%20threads%20are%20used%20in,computer%2C%20such%20as%20a%20webserver. [Accessed: 10- Jan- 2021].

[5]D. Fadeyev, "8 Characteristics Of Successful User Interfaces", *Usabilitypost.com*, 2021. [Online]. Available: https://usabilitypost.com/2009/04/15/8-characteristics-of-successful-user-interfaces/. [Accessed: 10- Jan- 2021].

[6]E. Wong, "User Interface Design Guidelines: 10 Rules of Thumb", *The Interaction Design Foundation*, 2020. [Online]. Available: https://www.interaction-design.org/literature/article/user-interface-design-guidelines-10-rules-of-thumb. [Accessed: 10- Jan- 2020].

[7]A. Smith, "6 GUI Design Principles Every Designer Should Know | Hacker Noon", *Hackernoon.com*, 2020. [Online]. Available: https://hackernoon.com/6-must-know-gui-design-principles-to-make-you-an-excellent-ui-designer-4643dadf8459. [Accessed: 10- Jan- 2021].

[8]"Dynamic typing vs. static typing", *Docs.oracle.com*, 2015. [Online]. Available: https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html#:~:text=There%20are%20two%20main%20differences,type%20checking%20at%20compile%20time. [Accessed: 3- Jan- 2021].

[9]y. nader, "Advantages of Python over Java - Java vs Python - DataFlair", *DataFlair*, 2021. [Online]. Available: https://hackr.io/blog/python-vs-java. [Accessed: 3- Jan- 2021].

[10]E. Goebelbecker, "Python vs Java: Which is best? Code examples and comparison | Raygun Blog", *Raygun Blog*, 2018. [Online]. Available: https://raygun.com/blog/java-vs-python/. [Accessed: 4- Jan- 2021].

[11]R. Saini, "Difference Between Compiler and Interpreter with respect to JVM (Java virtual machine) and PVM…", *Medium*, 2019. [Online]. Available: https://medium.com/@rahul77349/difference-between-compiler-and-interpreter-with-respect-to-jvm-java-virtual-machine-and-pvm-22fc77ae0eb7. [Accessed: 4- Jan- 2021].

[12]"How to Create a Database in MongoDB", *MongoDB*. [Online]. Available: https://www.mongodb.com/basics/create-database. [Accessed: 5- Jan- 2021].

[13]"pandas.DataFrame.to_json — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_json.html. [Accessed: 6- Jan- 2021].

[14]"pandas.read_csv — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html. [Accessed: 6- Jan- 2021].

[15]"pandas.DataFrame.to_dict — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_dict.html. [Accessed: 6- Jan- 2021].

[16]"How To Convert Python Dictionary To JSON? - GeeksforGeeks", *GeeksforGeeks*, 2020. [Online]. Available: https://www.geeksforgeeks.org/how-to-convert-python-dictionary-to-json/. [Accessed: 10- Jan- 2021].

[17]"db.collection.find() — MongoDB Manual", *Docs.mongodb.com*. [Online]. Available: https://docs.mongodb.com/manual/reference/method/db.collection.find/. [Accessed: 10- Jan- 2021].

[18]"pandas.DataFrame — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html. [Accessed: 10- Jan- 2021].

[19]P. Pandey, "Data Preprocessing : Concepts", *Medium*, 2019. [Online]. Available: https://towardsdatascience.com/data-preprocessing-concepts-fa946d11c825. [Accessed: 03- Jan- 2021].

[20]"pandas.DataFrame.merge — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.merge.html. [Accessed: 3- Jan- 2021].

[21]"pandas.Series.str.extract — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.extract.html. [Accessed: 2- Jan- 2021].

[22]"pandas.Series.str.replace — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.replace.html. [Accessed: 2- Jan- 2021].

[23]"pandas.to_datetime — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html. [Accessed: 2- Jan- 2021].

[24]"pandas.DataFrame.loc — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html. [Accessed: 2- Jan- 2021].

[25]"pandas.DataFrame.groupby — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html. [Accessed: 2- Jan- 2021].

[26]"How to Compute the Mean, Median, and Mode in Python", *Learningaboutelectronics.com*, 2018. [Online]. Available: http://www.learningaboutelectronics.com/Articles/How-to-compute-the-mean-median-and-mode-in-Python.php. [Accessed: 2- Jan- 2021].

[27]"pandas.DataFrame.unstack — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.unstack.html. [Accessed: 3- Jan- 2021].

[28]"Python | Convert set into a list - GeeksforGeeks", *GeeksforGeeks*. [Online]. Available: https://www.geeksforgeeks.org/python-convert-set-into-a-list/. [Accessed: 3- Jan- 2021].

[29]R. Williams, *How to view Excel File or Pandas DataFrame in Tkinter (Python GUI)*. 2020.

[30]"pandas.DataFrame.notna — pandas 1.2.0 documentation", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.notna.html. [Accessed: 9- Jan- 2021].

[31]"Embedding in Tk — Matplotlib 3.1.2 documentation", *Matplotlib.org*, 2012. [Online]. Available: https://matplotlib.org/3.1.1/gallery/user_interfaces/embedding_in_tk_sgskip.html. [Accessed: 9- Jan- 2021].

[32]S. jeev, "RadioButton in Tkinter | Python - GeeksforGeeks", *GeeksforGeeks*, 2019. [Online]. Available: https://www.geeksforgeeks.org/radiobutton-in-tkinter-python/. [Accessed: 09- Jan- 2021].

[33]"matplotlib.pyplot.bar — Matplotlib 3.1.2 documentation", *Matplotlib.org*. [Online]. Available: https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.bar.html. [Accessed: 10- Jan- 2021].

[34]M. CJ, "How can I integrate Seaborn plot into Tkinter GUI", *Stack Overflow*, 2019. [Online]. Available: https://stackoverflow.com/questions/55680827/how-can-i-integrate-seaborn-plot-into-tkinter-gui. [Accessed: 10- Jan- 2021].

[35]B. Tasch, "Why ZIP codes have 5 numbers — and what they each mean", *Business Insider*, 2015. [Online]. Available: https://www.businessinsider.com/what-do-zip-codes-mean-2015-6?r=US&IR=T. [Accessed: 09- Dec- 2020].

[36]"What is automated individual decision-making and profiling?", *Ico.org.uk*, 2020. [Online]. Available: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/automated-decision-making-and-profiling/what-is-automated-individual-decision-making-and-profiling/#:~:text=Automated%20decision%2Dmaking%20is%20the,created%20profiles%20or%20inferred%20data. [Accessed: 21- Dec- 2020].

[37]A. Brown, "Biased Algorithms Learn From Biased Data: 3 Kinds Biases Found In AI Datasets", *Forbes*, 2020. [Online]. Available:

https://www.forbes.com/sites/cognitiveworld/2020/02/07/biased-algorithms/?sh=2c1fa5c476fc. [Accessed: 21- Dec- 2020].

[38]"echo-chamber noun - Definition, pictures, pronunciation and usage notes | Oxford Advanced Learner's Dictionary at OxfordLearnersDictionaries.com", *Oxfordlearnersdictionaries.com*, 2020. [Online]. Available: https://www.oxfordlearnersdictionaries.com/us/definition/english/echo-chamber. [Accessed: 21- Dec- 2020].

[39]L. Garcia, "How algorithms and filter bubbles decide what we see on social media", *Bbc.co.uk*, 2020. [Online]. Available: https://www.bbc.co.uk/bitesize/articles/zd9tt39. [Accessed: 21- Dec- 2020].

[40]"What is a Filter Bubble? - Definition from Techopedia", *Techopedia.com*, 2020. [Online]. Available: https://www.techopedia.com/definition/28556/filter-bubble. [Accessed: 21- Dec- 2020].

[41]S. Meredith, "Facebook-Cambridge Analytica: A timeline of the data hijacking scandal", *CNBC*, 2018. [Online]. Available: https://www.cnbc.com/2018/04/10/facebook-cambridge-analytica-a-timeline-of-the-data-hijacking-scandal.html. [Accessed: 21- Dec- 2020].

[42]J. Gottfried and E. Shearer, "News Use Across Social Media Platforms 2016", *Pew Research Center's Journalism Project*, 2016. [Online]. Available: https://www.journalism.org/2016/05/26/news-use-across-social-media-platforms-2016/#fn-55250-1. [Accessed: 21- Dec- 2020].

[43]C. Silverman, L. Strapgaiel, H. Shaban, E. Hall and J. Singer-Vine, "Hyperpartisan Facebook Pages Are Publishing False And Misleading Information At An Alarming Rate", *Buzzfeednews.com*, 2016. [Online]. Available: https://www.buzzfeednews.com/article/craigsilverman/partisan-fb-pages-analysis. [Accessed: 22- Dec- 2020].

[44]"Article 8: Respect for your private and family life | Equality and Human Rights Commission", *Equalityhumanrights.com*, 2018. [Online]. Available: https://www.equalityhumanrights.com/en/human-rights-act/article-8-respect-your-private-and-family-life. [Accessed: 09- Jan- 2021].

[45]*General Data Protection Regulation (GDPR)*. ICO. Information Comissioners Office, 2018, pp. 3-194.

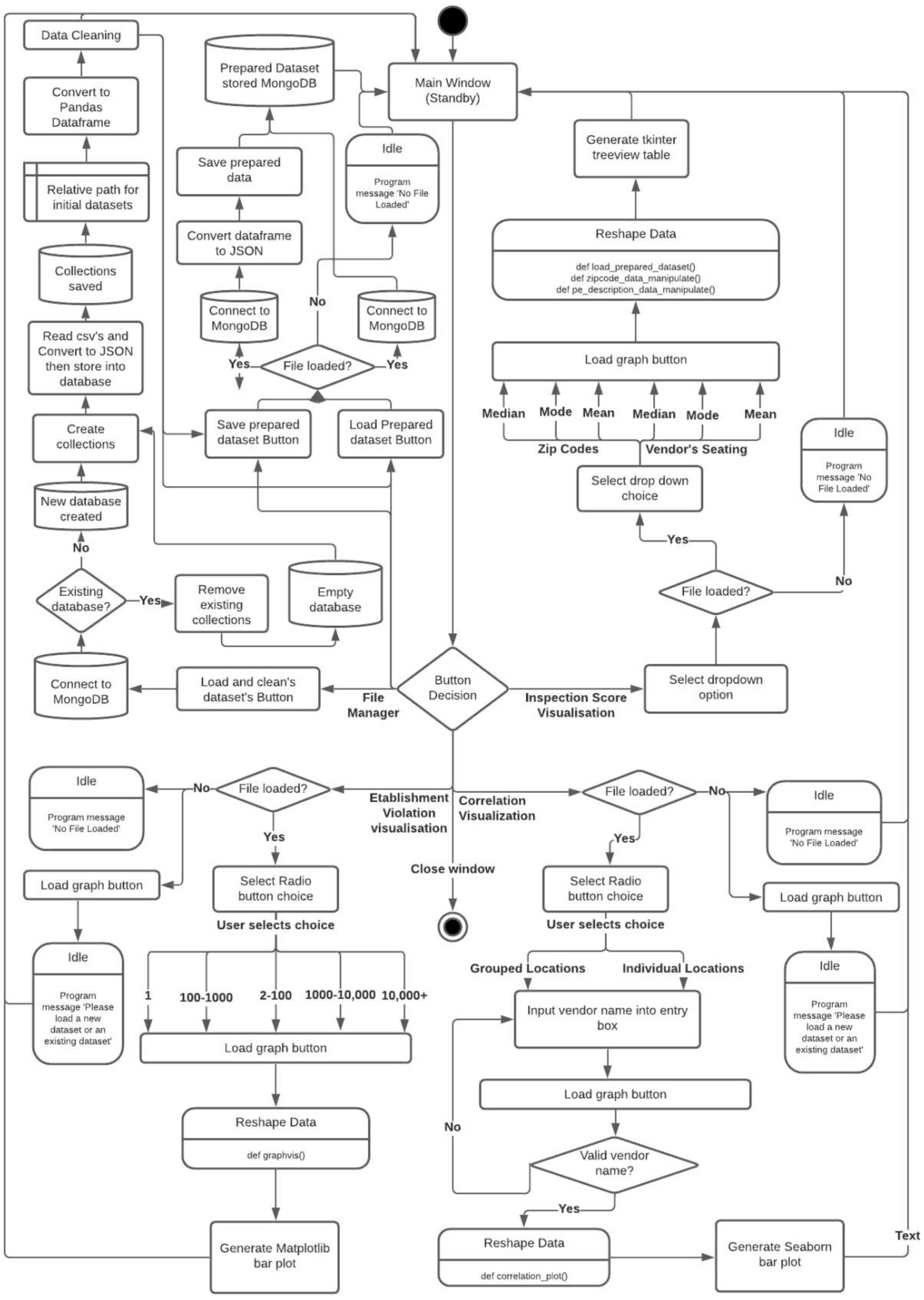
[46] UK Public General Acts, Data Protection Act 2018, Section 49. [Online]. Available: https://www.legislation.gov.uk/ukpga/2018/12/section/49#:~:text=49Right%20not%20to%20be%20subject%20to%20automated%20decision%2Dmaking&text=(1)A%20controller%20may%20not,required%20or%20authorised%20by%20law. [Accessed: Jan. 01, 2021].

Appendices

Appendix A.1 – Threading applied to JSON conversion, insert into collections and JSON to pandas dataframe conversion

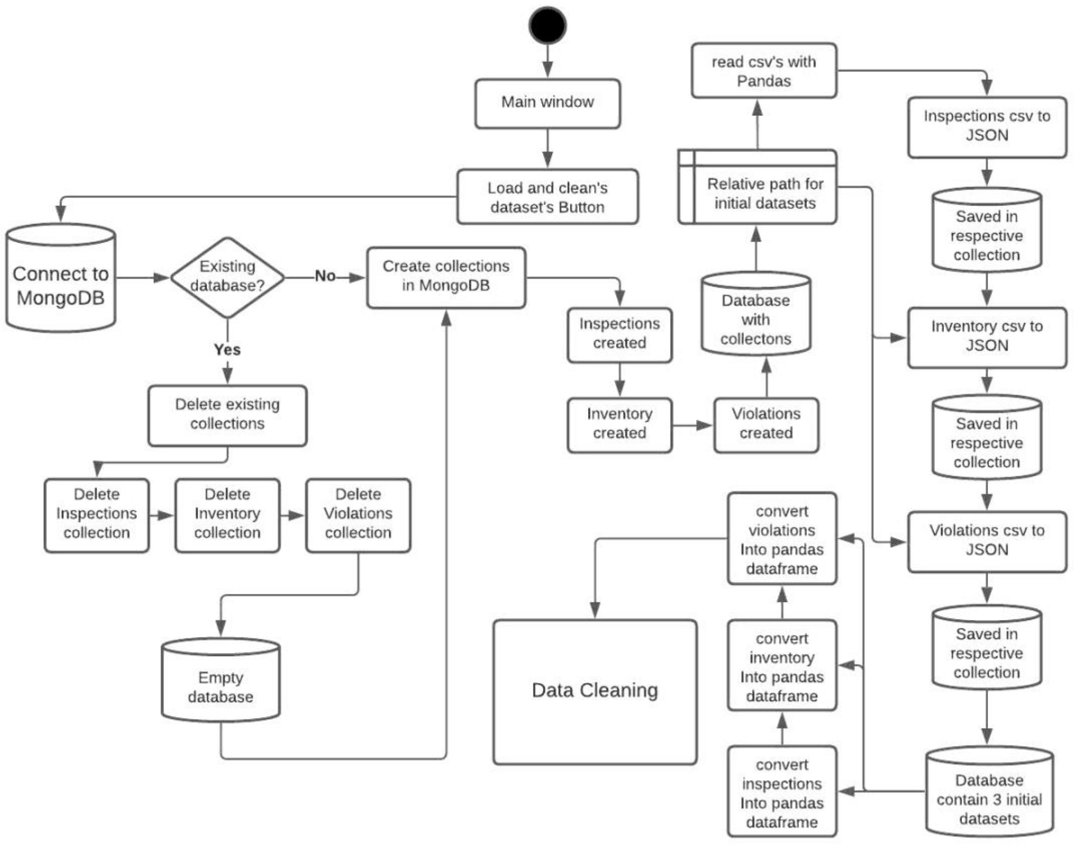
```
Import threading

def convert_ins_json():
        ins_collection.insert_many(convert_to_json('Inspections.csv'))
def convert_inv_json():
        inv_collection.insert_many(convert_to_json('Inventroy.csv'))
def convert_vio_json():
        vio_collection.insert_many(convert_to_json('Violations.csv'))

t1 = threading.Thread(target=convert_ins_json)
t2 = threading.Thread(target=convert_inv_json)
t3 = threading.Thread(target=convert_vio_json)

t1.start()
t2.start()
t3.start()

t1.join()
t2.join()
t3.join()

def convert_ins_pandas():
        global inspections_df
        inspections_df = pd.DataFrame(list(ins_collection.find({}, {'_id': False})))
def convert_inv_pandas():
        global inventory_df
        inventory_df = pd.DataFrame(list(inv_collection.find({}, {'_id': False})))
def convert_vio_pandas():
        global violations_df
violations_df = pd.DataFrame(list(vio_collection.find({}, {'_id': False})))

t4 = threading.Thread(target=convert_ins_pandas)
t5 = threading.Thread(target=convert_inv_pandas)
t6 = threading.Thread(target=convert_vio_pandas)

t4.start()
t5.start()
t6.start()

t4.join()
t5.join()
t6.join()
```

Appendix A.2 – Existing Program Flow



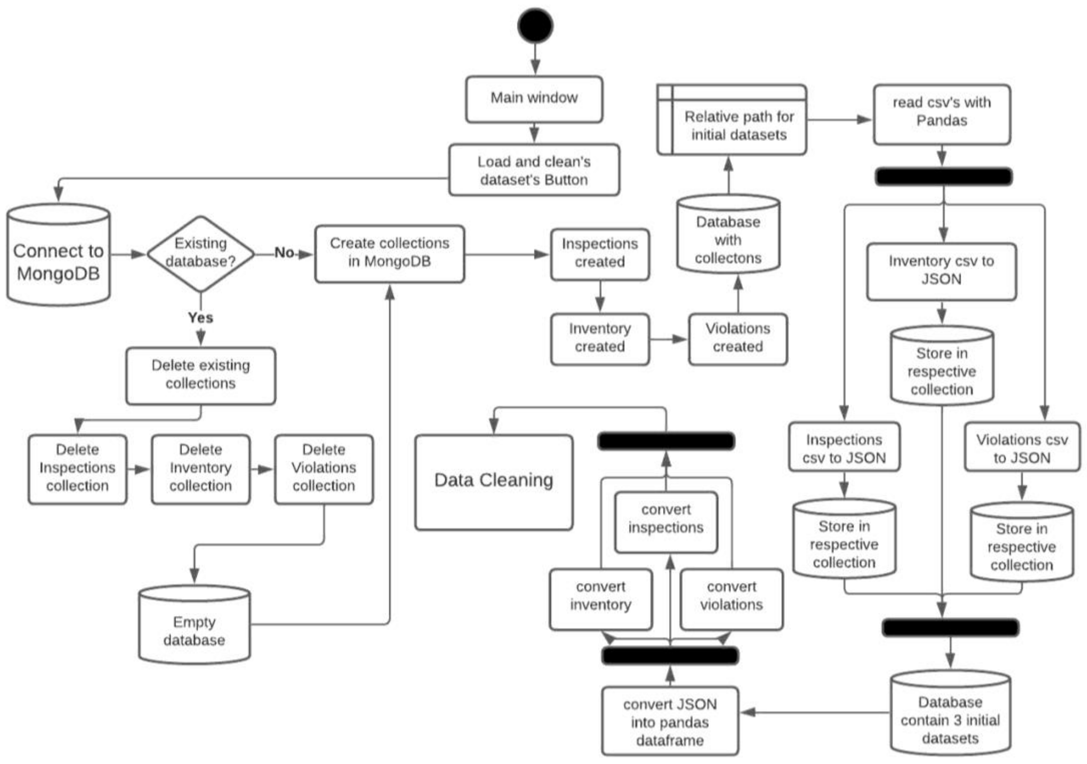
Main Window (Standby)
Data Cleaning
Convert to Pandas Dataframe
Relative path for initial datasets
Collections saved
Read csv's and Convert to JSON then store into database
Create collections
New database created
No
Existing database?
Yes
Remove existing collections
Empty database
Connect to MongoDB
Load and clean's dataset's Button
File Manager
Prepared Dataset stored MongoDB
Save prepared data
Convert dataframe to JSON
Connect to MongoDB
Yes
File loaded?
No
Yes
Connect to MongoDB
Idle
Program message 'No File Loaded'
Save prepared dataset Button
Load Prepared dataset Button
Button Decision
Inspection Score Visualisation
Select dropdown option
File loaded?
Yes
No
Idle
Program message 'No File Loaded'
Select drop down choice
Zip Codes
Vendor's Seating
Median
Mode
Mean
Median
Mode
Mean
Load graph button
Reshape Data
def load_prepared_dataset()
def zipcode_data_manipulate()
def pe_description_data_manipulate()
Generate tkinter treeview table
Etablishment Violation visualisation
File loaded?
No
Idle
Program message 'No File Loaded'
Load graph button
Idle
Program message 'Please load a new dataset or an existing dataset'
Yes
Select Radio button choice
User selects choice
1
100-1000
2-100
1000-10,000
10,000+
Load graph button
Reshape Data
def graphvis()
Generate Matplotlib bar plot
Close window
Correlation Visualization
File loaded?
No
Idle
Program message 'No File Loaded'
Load graph button
Idle
Program message 'Please load a new dataset or an existing dataset'
Yes
Select Radio button choice
User selects choice
Grouped Locations
Individual Locations
Input vendor name into entry box
Load graph button
Valid vendor name?
No
Yes
Reshape Data
def correlation_plot()
Generate Seaborn bar plot
Text

Appendix A.3 – File extracting, storing and converting from MongoDB Database, no threading



Main window
Load and clean's dataset's Button
Connect to MongoDB
Existing database?
No
Yes
Create collections in MongoDB
Delete existing collections
Delete Inspections collection
Delete Inventory collection
Delete Violations collection
Empty database
Inspections created
Inventory created
Violations created
Database with collectons
Relative path for initial datasets
read csv's with Pandas
Inspections csv to JSON
Saved in respective collection
Inventory csv to JSON
Saved in respective collection
Violations csv to JSON
Saved in respective collection
Database contain 3 initial datasets
convert inspections Into pandas dataframe
convert inventory Into pandas dataframe
convert violations Into pandas dataframe
Data Cleaning

Appendix A.4 - File extracting, storing and converting from MongoDB Database, with threading



Main window
Load and clean's dataset's Button
Connect to MongoDB
Existing database?
No
Yes
Create collections in MongoDB
Delete existing collections
Delete Inspections collection
Delete Inventory collection
Delete Violations collection
Empty database
Inspections created
Inventory created
Violations created
Database with collectons
Relative path for initial datasets
read csv's with Pandas
Inventory csv to JSON
Store in respective collection
Inspections csv to JSON
Store in respective collection
Violations csv to JSON
Store in respective collection
Database contain 3 initial datasets
convert JSON into pandas dataframe
convert inventory
convert inspections
convert violations
Data Cleaning

Appendix B.1 – frame created for each requirement controls. Radio, click, Option menu and entry buttons are positioned within frames by specifying frame names.

```
frame1 = tk.LabelFrame(root, text="Client Data")
file_frame = tk.LabelFrame(root,text="Inspection Score Visualization")
file_frame2 = tk.LabelFrame(root,text="Establishment Violations Visualization")
file_frame3 = tk.LabelFrame(root,text='File Manager')
file_frame4 = tk.LabelFrame(root,text='Correlation Visualization')
file_frame5 = tk.LabelFrame(root)

- tk.OptionMenu(file_frame, clicked, *options)
- show_button = tk.Button(file_frame,text="Show Table",command=lambda: Load_csv_data())

- tk.Radiobutton(file_frame2,text=text, variable=clicked2,value=mode).pack(anchor=tk.N)
- button3 = tk.Button(file_frame2,text="Load Graph",command=lambda: graphvis())

- save_button = tk.Button(file_frame3,text="Save prepared dataset",command=lambda:
                              save_dataset())
- load_clean_button = tk.Button(file_frame3,text="Load and clean dataset's",command=lambda:
                                    load_and_clean_data())
- load_prep_button = tk.Button(file_frame3,text="Load prepared dataset",command=lambda:
                                   load_prepared_dataset())

- tk.Radiobutton(file_frame4,text=text, variable=clicked3,value=mode).pack(anchor=tk.N)
- e = tk.Entry(file_frame4,width=30,)
- button4 = tk.Button(file_frame4,text="Load Graph",command=lambda: correlation_plot())
```

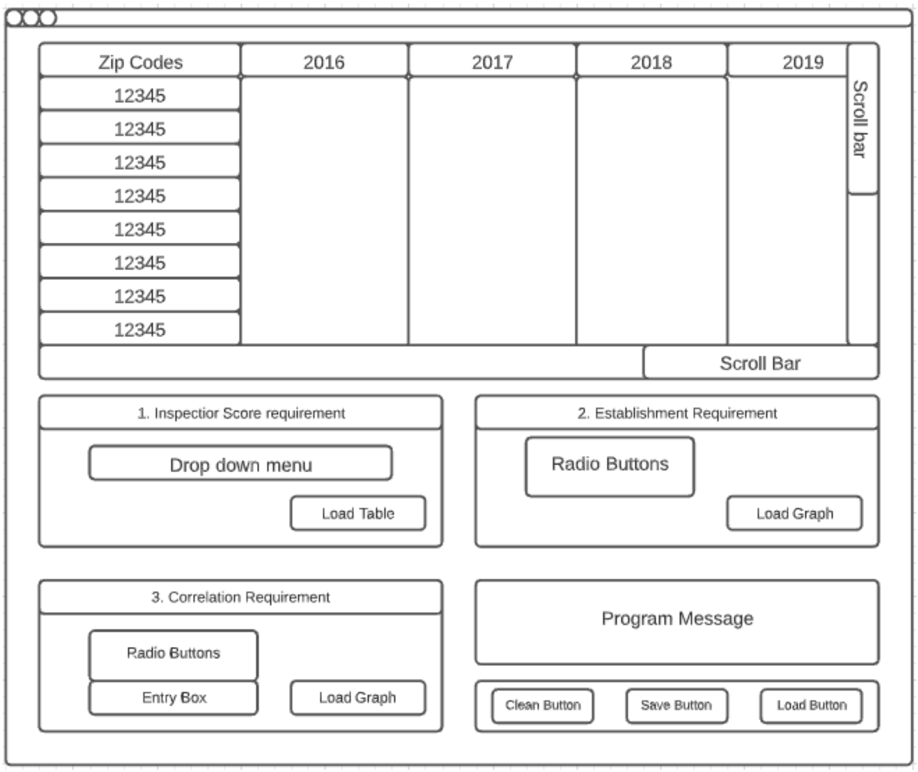
Appendix B.2 – two functions created for program message prompt to user, message variable is assigned depending on the user interaction, mainly used within Errors and exceptions implementation.

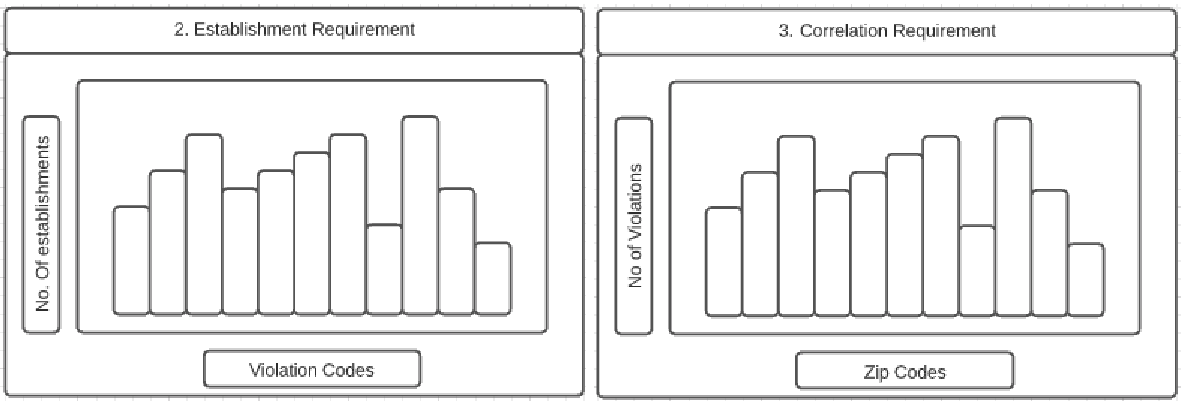
```
def message_delete():
        global label_file_frame5
        label_file_frame5.destroy()

def message_box():
        global message
        global label_file_frame5

        label_file_frame5 = ttk.Label(file_frame5,text=message)
        label_file_frame5.place(rely=0,relx=0)
```

Appendix B.3 – tkinter tree view table implementation

```
tv1 = ttk.Treeview(frame1)
tv1.place(relheight=1,relwidth=1)
treescrolly = tk.Scrollbar(frame1,orient="vertical",command=tv1.yview)
treescrollx = tk.Scrollbar(frame1,orient="horizontal",command=tv1.xview)
tv1.configure(xscrollcommand=treescrollx.set, yscrollcommand=treescrolly.set)
treescrollx.pack(side="bottom",fill="x")
treescrolly.pack(side="right",fill="y"
```

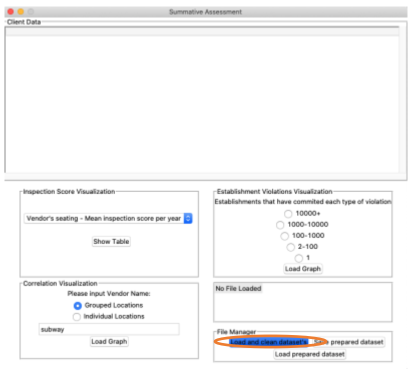
Appendix B.4 - Wireframe





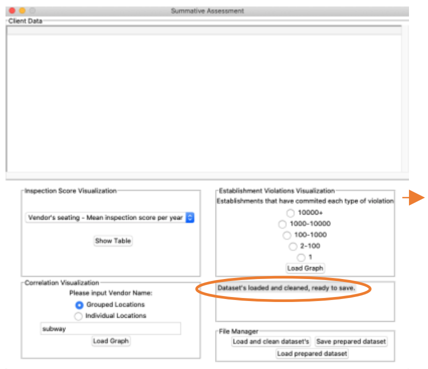
Appendix B.5 – GUI Interaction walkthrough

1. **First time running program - Loading, cleaning and saving dataset.**


Summative Assessment
Client Data
Inspection Score Visualization
Vendor's seating - Mean inspection score per year
Show Table
Establishment Violations Visualization
Establishments that have commited each type of violation
10000+
1000-10000
100-1000
2-100
1
Load Graph
Correlation Visualization
Please input Vendor Name:
Grouped Locations
Individual Locations
subway
Load Graph
File Manager
Load and clean dataset's
Save prepared dataset
Load prepared dataset

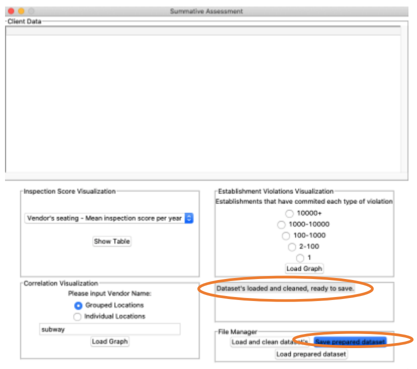
No File Loaded

Click load and clean dataset's, saved in mongo database


Summative Assessment
Client Data
Inspection Score Visualization
Vendor's seating - Mean inspection score per year
Show Table
Establishment Violations Visualization
Establishments that have commited each type of violation
10000+
1000-10000
100-1000
2-100
1
Load Graph
Correlation Visualization
Please input Vendor Name:
Grouped Locations
Individual Locations
subway
Load Graph
File Manager
Load and clean dataset's
Save prepared dataset
Load prepared dataset

Dataset's loaded and cleaned, ready to save.

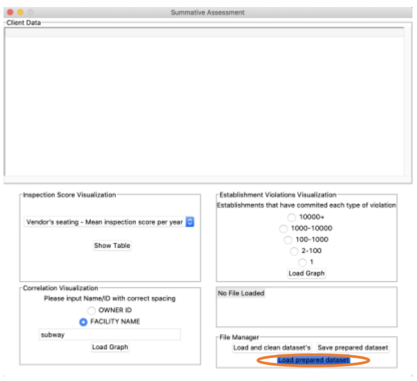
Program confirmation


Summative Assessment
Client Data
Inspection Score Visualization
Vendor's seating - Mean inspection score per year
Show Table
Establishment Violations Visualization
Establishments that have commited each type of violation
10000+
1000-10000
100-1000
2-100
1
Load Graph
Correlation Visualization
Please input Vendor Name:
Grouped Locations
Individual Locations
subway
Load Graph
File Manager
Load and clean dataset's
Save prepared dataset
Load prepared dataset

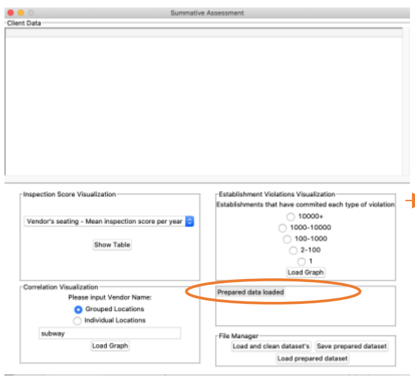
Dataset's loaded and cleaned, ready to save.

Click save prepared dataset – stored in mongo database

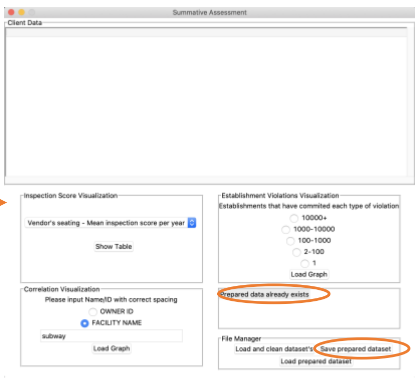
2. **Running program after save – Reloading saved prepared data.**


Summative Assessment
Client Data
Inspection Score Visualization
Vendor's seating - Mean inspection score per year
Show Table
Establishment Violations Visualization
Establishments that have commited each type of violation
10000+
1000-10000
100-1000
2-100
1
Load Graph
Correlation Visualization
Please input Name/ID with correct spacing
OWNER ID
FACILITY NAME
subway
Load Graph
No File Loaded
File Manager
Load and clean dataset's
Save prepared dataset
Load prepared dataset

Click Load prepared dataset

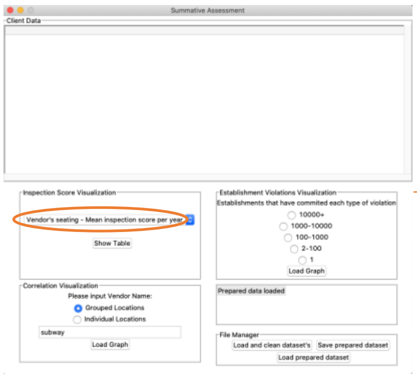


Program confirmation

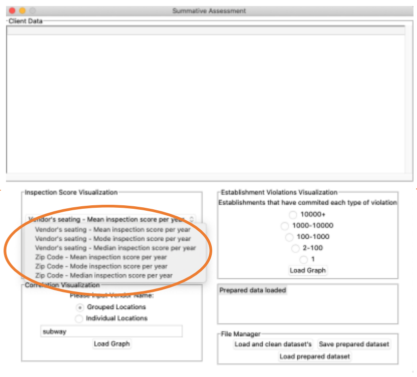

Summative Assessment
Client Data
Inspection Score Visualization
Vendor's seating - Mean inspection score per year
Show Table
Establishment Violations Visualization
Establishments that have commited each type of violation
10000+
1000-10000
100-1000
2-100
1
Load Graph
Correlation Visualization
Please input Name/ID with correct spacing
OWNER ID
FACILITY NAME
subway
Load Graph
Prepared data already exists
File Manager
Load and clean dataset's
Save prepared dataset
Load prepared dataset

Saving prepared dataset, program will flag existing file

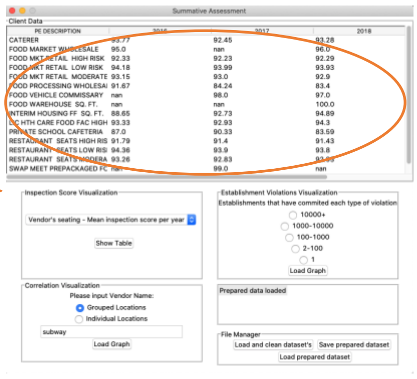
3. **Mean, Median and Mode Inspection Score Visualization – Loading data in treeview table**



Select dropdown menu


Summative Assessment
Client Data
Inspection Score Visualization
Vendor's seating - Mean inspection score per year
Vendor's seating - Mean inspection score per year
Vendor's seating - Mode inspection score per year
Vendor's seating - Median inspection score per year
Zip Code - Mean inspection score per year
Zip Code - Mode inspection score per year
Zip Code - Median inspection score per year
Correlation Visualization
Please input Vendor Name:
Grouped Locations
Individual Locations
subway
Load Graph
Establishment Violations Visualization
Establishments that have commited each type of violation
10000+
1000-10000
100-1000
2-100
1
Load Graph
Prepared data loaded
File Manager
Load and clean dataset's
Save prepared dataset
Load prepared dataset

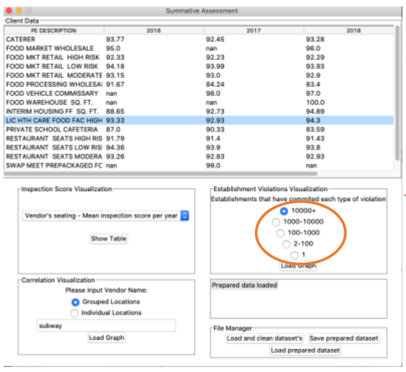
Select choice and click show



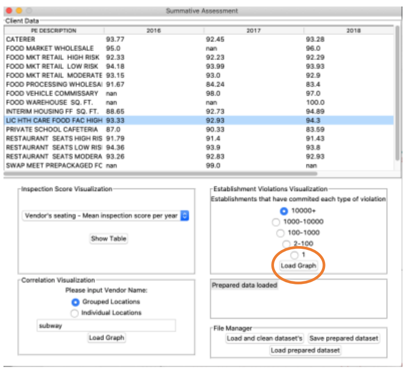
PE DESCRIPTION	2016	2017	2018
CATERER	93.77	92.45	93.28
FOOD MARKET WHOLESALE	95.0	nan	96.0
FOOD MKT RETAIL HIGH RISK	92.33	92.23	92.29
FOOD MKT RETAIL LOW RISK	94.18	93.99	93.93
FOOD MKT RETAIL MODERATE	93.15	93.0	92.9
FOOD PROCESSING WHOLESAI	91.67	84.24	83.4
FOOD VEHICLE COMMISSARY	nan	98.0	97.0
FOOD WAREHOUSE SQ. FT.	nan	nan	100.0
INTERIM HOUSING FF SQ. FT.	88.65	92.73	94.89
LIC HTH CARE FOOD FAC HIGH	93.33	92.93	94.3
PRIVATE SCHOOL CAFETERIA	87.0	90.33	83.59
RESTAURANT SEATS HIGH RIS	91.79	91.4	91.43
RESTAURANT SEATS LOW RIS	94.36	93.9	93.8
RESTAURANT SEATS MODERA	93.26	92.83	92.93
SWAP MEET PREPACKAGED FC	nan	99.0	nan

Treeview table loaded

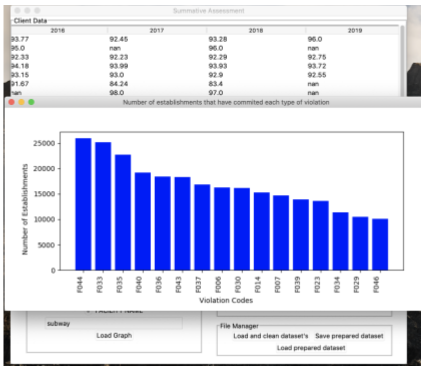
4. **Establishment violation visualization – Loading Matplotlib bar graphs**



Select radio button range

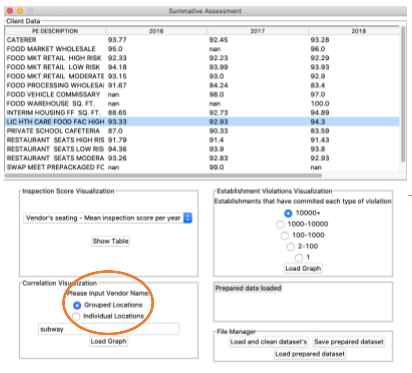


Click load graph button

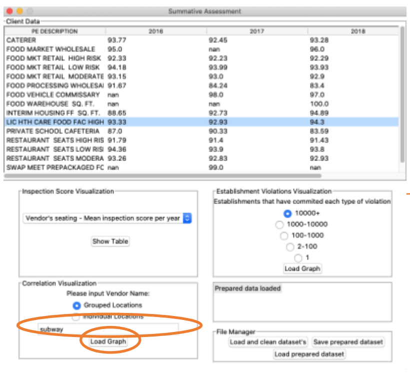


Bar graph generated

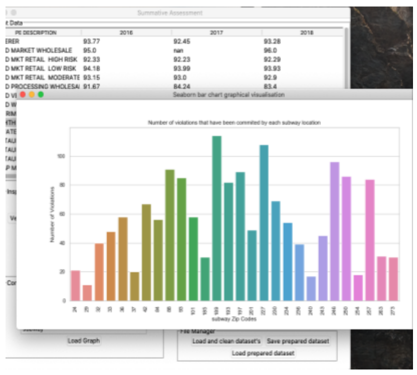
5. **Correlation visualization – Loading Seaborn bar graphs**



Select radio button to filter input option

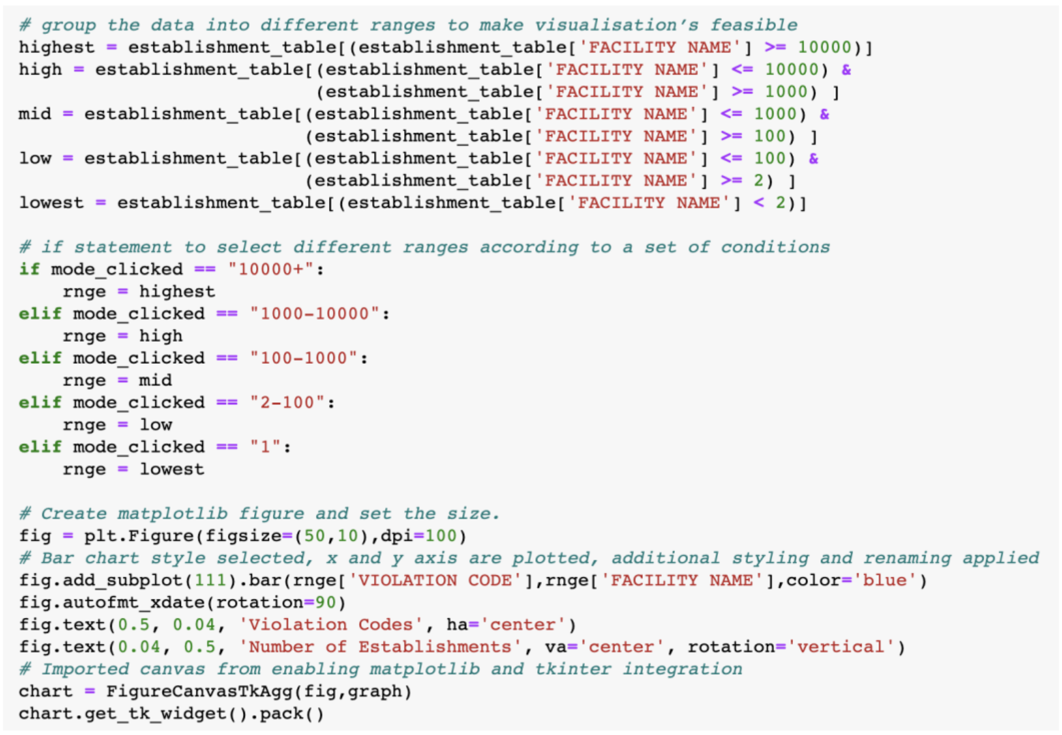


Enter ID/Name of Vendor and loading graph



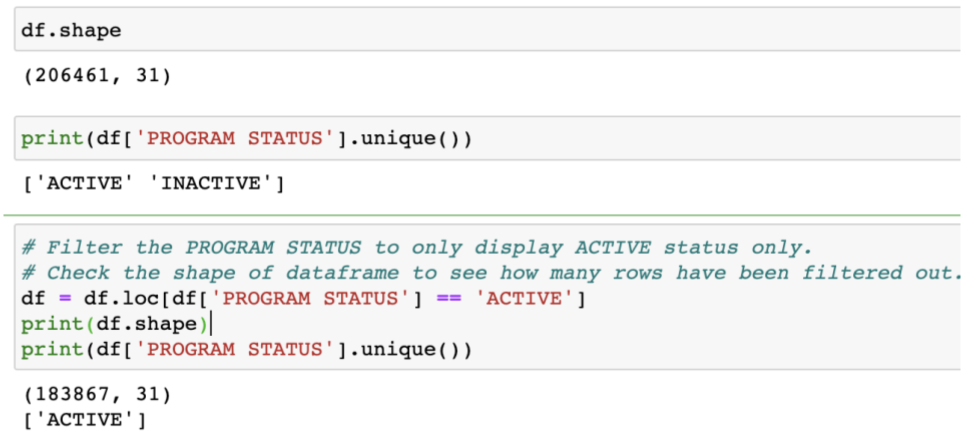
Seaborn Bar Plot

Appendix C.1 – assigning variables at any given time and reassigning them into new variables



```
# group the data into different ranges to make visualisation's feasible
highest = establishment_table[(establishment_table['FACILITY NAME'] >= 10000)]
high = establishment_table[(establishment_table['FACILITY NAME'] <= 10000) &
                           (establishment_table['FACILITY NAME'] >= 1000) ]
mid = establishment_table[(establishment_table['FACILITY NAME'] <= 1000) &
                          (establishment_table['FACILITY NAME'] >= 100) ]
low = establishment_table[(establishment_table['FACILITY NAME'] <= 100) &
                          (establishment_table['FACILITY NAME'] >= 2) ]
lowest = establishment_table[(establishment_table['FACILITY NAME'] < 2)]

# if statement to select different ranges according to a set of conditions
if mode_clicked == "10000+":
    rnge = highest
elif mode_clicked == "1000-10000":
    rnge = high
elif mode_clicked == "100-1000":
    rnge = mid
elif mode_clicked == "2-100":
    rnge = low
elif mode_clicked == "1":
    rnge = lowest

# Create matplotlib figure and set the size.
fig = plt.Figure(figsize=(50,10),dpi=100)
# Bar chart style selected, x and y axis are plotted, additional styling and renaming applied
fig.add_subplot(111).bar(rnge['VIOLATION CODE'],rnge['FACILITY NAME'],color='blue')
fig.autofmt_xdate(rotation=90)
fig.text(0.5, 0.04, 'Violation Codes', ha='center')
fig.text(0.04, 0.5, 'Number of Establishments', va='center', rotation='vertical')
# Imported canvas from enabling matplotlib and tkinter integration
chart = FigureCanvasTkAgg(fig,graph)
chart.get_tk_widget().pack()
```

Appendix C.2 - Python running checks on dataframe without the need of compiling the entire program at once, displaying shape, unique values, dataframe columns, index and values.



```
df.shape
```

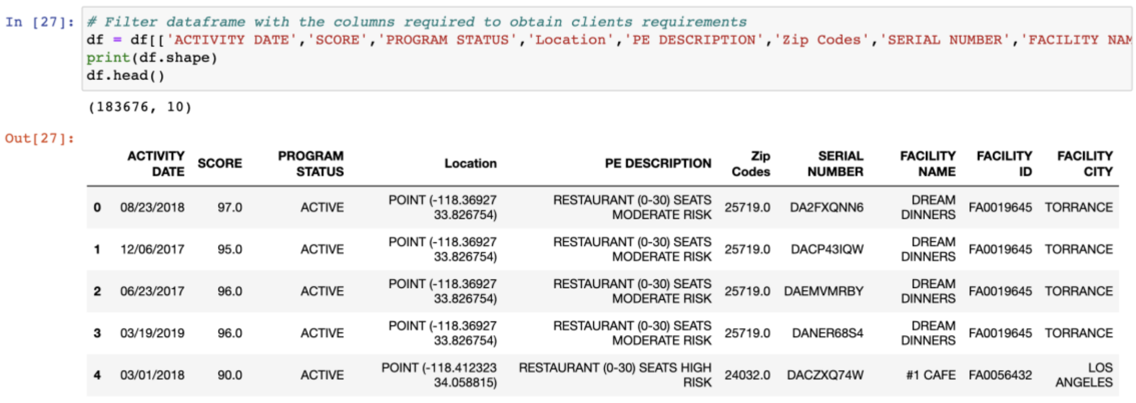
```
(206461, 31)
```

```
print(df['PROGRAM STATUS'].unique())
```

```
['ACTIVE' 'INACTIVE']
```

```
# Filter the PROGRAM STATUS to only display ACTIVE status only.
# Check the shape of dataframe to see how many rows have been filtered out.
df = df.loc[df['PROGRAM STATUS'] == 'ACTIVE']
print(df.shape)
print(df['PROGRAM STATUS'].unique())
```

```
(183867, 31)
['ACTIVE']
```



```
In [27]: # Filter dataframe with the columns required to obtain clients requirements
         df = df[['ACTIVITY DATE','SCORE','PROGRAM STATUS','Location','PE DESCRIPTION','Zip Codes','SERIAL NUMBER','FACILITY NAM
         print(df.shape)
         df.head()

         (183676, 10)

Out[27]:
```

	ACTIVITY DATE	SCORE	PROGRAM STATUS	Location	PE DESCRIPTION	Zip Codes	SERIAL NUMBER	FACILITY NAME	FACILITY ID	FACILITY CITY
0	08/23/2018	97.0	ACTIVE	POINT (-118.36927 33.826754)	RESTAURANT (0-30) SEATS MODERATE RISK	25719.0	DA2FXQNN6	DREAM DINNERS	FA0019645	TORRANCE
1	12/06/2017	95.0	ACTIVE	POINT (-118.36927 33.826754)	RESTAURANT (0-30) SEATS MODERATE RISK	25719.0	DACP43IQW	DREAM DINNERS	FA0019645	TORRANCE
2	06/23/2017	96.0	ACTIVE	POINT (-118.36927 33.826754)	RESTAURANT (0-30) SEATS MODERATE RISK	25719.0	DAEMVMRBY	DREAM DINNERS	FA0019645	TORRANCE
3	03/19/2019	96.0	ACTIVE	POINT (-118.36927 33.826754)	RESTAURANT (0-30) SEATS MODERATE RISK	25719.0	DANER68S4	DREAM DINNERS	FA0019645	TORRANCE
4	03/01/2018	90.0	ACTIVE	POINT (-118.412323 34.058815)	RESTAURANT (0-30) SEATS HIGH RISK	24032.0	DACZXQ74W	#1 CAFE	FA0056432	LOS ANGELES

Appendix D .1 – conversion function for .csv to .json, pd.read_csv to read the .csv files into a pandas dataframe and .to_dict('records') to convert pandas dataframe into json format, 'records' parameters to format JSON into list like structure.

```
def convert_to_json(file):
    csv_data = pd.read_csv(file)
    json_data = csv_data.to_dict('records')

    return json_data
```

Appendix D.2 – Connecting to the MongoDB client and creating a new database and new database created called 'data'.

```
connection = pymongo.MongoClient("mongodb://localhost:27017/")
MongoDB = connection["data"]
```

Appendix D.3 – Creating collection for each file for storage in mongoDB database.

```
ins_collection = MongoDB["inspections"]
inv_collection = MongoDB ["inventory"]
vio_collection = MongoDB ["violations"]
```

using pymong .insert_many() method to store collection into their respective collection and convert_to_json function() to convert .csv to .json.

```
ins_collection.insert_many(convert_to_json('Inspections.csv'))
inv_collection.insert_many(convert_to_json('Inventroy.csv'))
vio_collection.insert_many(convert_to_json('Violations.csv'))
```

Appendix D.4 – Extracting collections from database and converting them in pandas dataframe using the combination of .find() method where '_id': False removes the MongoDB id's in the data and pd.DataFrame for finallizing conversion.

```
inspections_df = pd.DataFrame(list(ins_collection.find({}, {'_id': False})))
inventory_df = pd.DataFrame(list(inv_collection.find({}, {'_id': False})))
violations_df = pd.DataFrame(list(vio_collection.find({}, {'_id': False})))
```

Appendix D.5 – Storing manipulated data by creating new collection as 'prepared data', the use of to_json for pandas dataframe to json conversion, json.loads() method to parse the json and finally inserting the collection into the relative collection.

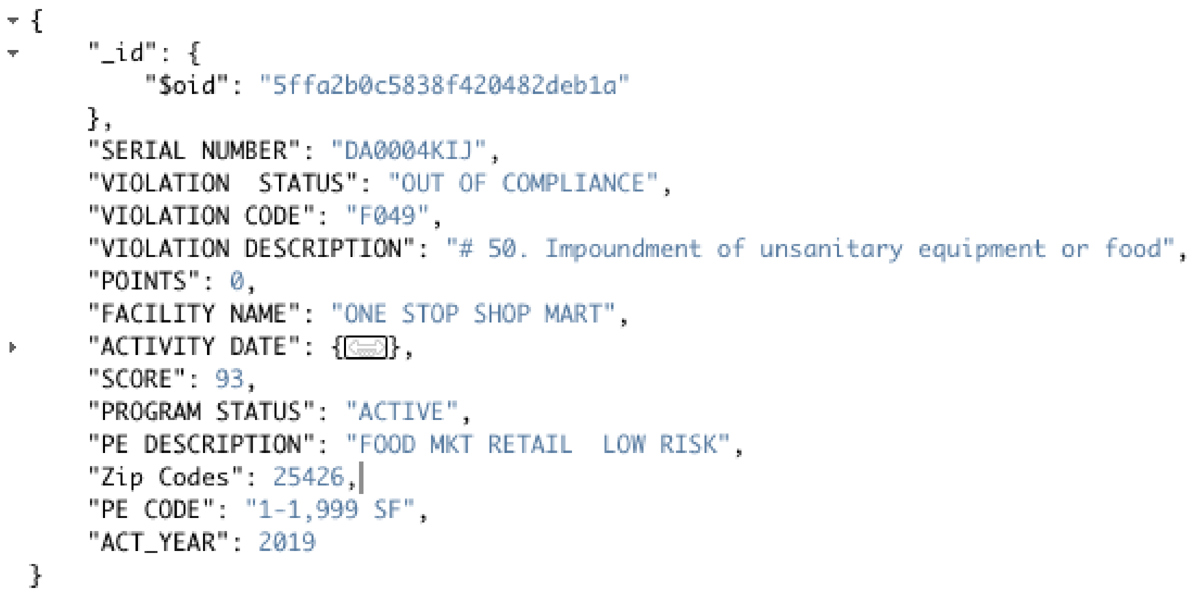
```
def save():

    saved_collection = MongoDB ["prepared data"].delete_many({})
    saved_collection = MongoDB ["prepared data"]
    df_save = prep_df.to_json(orient="records")
    parsed = json.loads(df_save)
    saved_collection.insert_many(parsed)
```

Loading the prepared data the next time the application using .find and pd.DataFrame.

```
prepared_dataset = MongoDB ["prepared data"]
prep_df = pd.DataFrame(list(prepared_dataset.find({}, {'_id': False})))
```

Appendix D.6 – JSON Data Format

```
{
   "_id": {
      "$oid": "5ffa2b0c5838f420482deb1a"
   },
   "SERIAL NUMBER": "DA0004KIJ",
   "VIOLATION  STATUS": "OUT OF COMPLIANCE",
   "VIOLATION CODE": "F049",
   "VIOLATION DESCRIPTION": "# 50. Impoundment of unsanitary equipment or food",
   "POINTS": 0,
   "FACILITY NAME": "ONE STOP SHOP MART",
   "ACTIVITY DATE": {
      "$numberLong": "1549497600000"
   },
   "SCORE": 93,
   "PROGRAM STATUS": "ACTIVE",
   "PE DESCRIPTION": "FOOD MKT RETAIL  LOW RISK",
   "Zip Codes": 25426,
   "PE CODE": "1-1,999 SF",
   "ACT_YEAR": 2019
}
```



```
{
    "_id": {
        "$oid": "5ffa2b0c5838f420482deb1a"
    },
    "SERIAL NUMBER": "DA0004KIJ",
    "VIOLATION  STATUS": "OUT OF COMPLIANCE",
    "VIOLATION CODE": "F049",
    "VIOLATION DESCRIPTION": "# 50. Impoundment of unsanitary equipment or food",
    "POINTS": 0,
    "FACILITY NAME": "ONE STOP SHOP MART",
    "ACTIVITY DATE": {},
    "SCORE": 93,
    "PROGRAM STATUS": "ACTIVE",
    "PE DESCRIPTION": "FOOD MKT RETAIL  LOW RISK",
    "Zip Codes": 25426,
    "PE CODE": "1-1,999 SF",
    "ACT_YEAR": 2019
}
```

Appendix E.1 – Removal of duplicate and missing values in rows 'SCORE' and 'Zip Codes' using .drop_duplicates and .notna() methods.

```
inspections_df.drop_duplicates(inplace=True)
inventory_df.drop_duplicates(inplace=True)
violations_df.drop_duplicates(inplace=True)
df = df[df['SCORE'].notna()]
df = df[df['Zip Codes'].notna()
```

Appendix E.2 – Merging of Inventory and Inspections data set, filling missing Zip Code values in inspection dataset with inventory dataset using pd.merge() parameter how='right'

```
 df = pd.merge(inventory_df,inspections_df,on=['RECORD ID'],how='right')

df["Zip Codes"] = df["Zip Codes_x"].combine_first(df["Zip Codes_y"])
df = df.drop(["Zip Codes_x", "Zip Codes_y"], axis=1)
```

Appendix E.3 – Use of regular expression with str.extract() and str.replace() for the removal and replacing of strings within parenthesis and .replace function to fix a typo in the values.

```
df['PE CODE'] = df['PE DESCRIPTION'].str.extract(r"\((.*?)\)", expand=False)
df['PE DESCRIPTION'] = df['PE DESCRIPTION'].str.replace(r"\((.*?)\)","")

df["PE DESCRIPTION"]= df["PE DESCRIPTION"].replace('LIC HTH CARE FOOD FAC  HIGH RISK',
                                                    'LIC HTH CARE FOOD FAC HIGH RISK',inplace=False)
```

Appendix E.4 – Conversion of 'ACTIVITY DATE' column into Datetime format to create a new 'ACT_YEAR' column using pd.to_datetime() and .map(lambda x:x) to extract years.

```
df['ACTIVITY DATE'] = pd.to_datetime(df['ACTIVITY DATE'])
ACT_YEAR = df['ACTIVITY DATE'].map(lambda x: x.year)
df['ACT_YEAR'] = ACT_YEAR
```

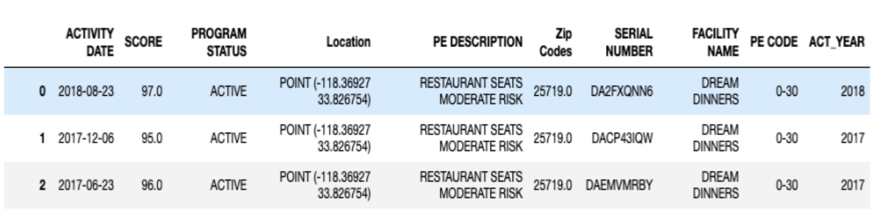
Appendix E.5 – Filtering 'PROGRAM STATUS' column to values containing 'ACTIVE' only.

```
df = df.loc[df['PROGRAM STATUS'] == 'ACTIVE']
```

Append E.6 – use of .groupby() to create multi-index dataframe and applying pandas built in statistical methods .median() and.mean() and pythons .value_counts() with lambda expression for mode. (df was renamed to prep_df during merging with violations dataset)

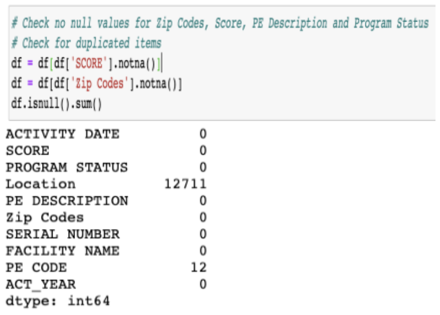
```
prep_df.groupby(['ACT_YEAR','PE DESCRIPTION'])[['SCORE']].median())
prep_df.groupby(['ACT_YEAR','PE DESCRIPTION'])[['SCORE']].mean())
prep_df.groupby(['ACT_YEAR','PE DESCRIPTION'])[['SCORE']].agg(lambda
x:x.value_counts().index[0])

prep_df.groupby(['ACT_YEAR','Zip Codes'])[['SCORE']].median()
prep_df.groupby(['ACT_YEAR','Zip Codes'])[['SCORE']].mean()
prep_df.groupby(['ACT_YEAR','Zip Codes'])[['SCORE']].agg(lambda x:x.value_counts().index[0])
```

Appendix E.7 – Successfully cleaned dataset, extracting only columns of interest, 0 duplicated values, filtered PROGRAM STATUS to ‘ACTIVE’ and 0 missing values in Zip Codes and PE Description



	ACTIVITY DATE	SCORE	PROGRAM STATUS	Location	PE DESCRIPTION	Zip Codes	SERIAL NUMBER	FACILITY NAME	PE CODE	ACT_YEAR
0	2018-08-23	97.0	ACTIVE	POINT (-118.36927 33.826754)	RESTAURANT SEATS MODERATE RISK	25719.0	DA2FXQNN6	DREAM DINNERS	0-30	2018
1	2017-12-06	95.0	ACTIVE	POINT (-118.36927 33.826754)	RESTAURANT SEATS MODERATE RISK	25719.0	DACP43IQW	DREAM DINNERS	0-30	2017
2	2017-06-23	96.0	ACTIVE	POINT (-118.36927 33.826754)	RESTAURANT SEATS MODERATE RISK	25719.0	DAEMVMRBY	DREAM DINNERS	0-30	2017

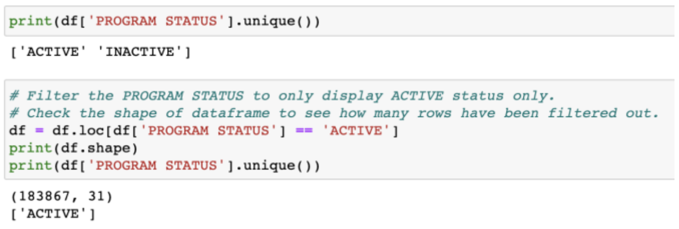
Filtered Dataframe



```
# Check no null values for Zip Codes, Score, PE Description and Program Status
# Check for duplicated items
df = df[df['SCORE'].notna()]
df = df[df['Zip Codes'].notna()]
df.isnull().sum()
```

```
ACTIVITY DATE        0
SCORE                0
PROGRAM STATUS       0
Location         12711
PE DESCRIPTION       0
Zip Codes            0
SERIAL NUMBER        0
FACILITY NAME        0
PE CODE             12
ACT_YEAR             0
dtype: int64
```

No missing data in Zip Codes and PE Description



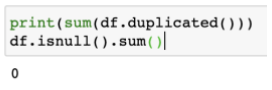
```
print(df['PROGRAM STATUS'].unique())
```

```
['ACTIVE' 'INACTIVE']
```

```
# Filter the PROGRAM STATUS to only display ACTIVE status only.
# Check the shape of dataframe to see how many rows have been filtered out.
df = df.loc[df['PROGRAM STATUS'] == 'ACTIVE']
print(df.shape)
print(df['PROGRAM STATUS'].unique())
```

```
(183867, 31)
['ACTIVE']
```

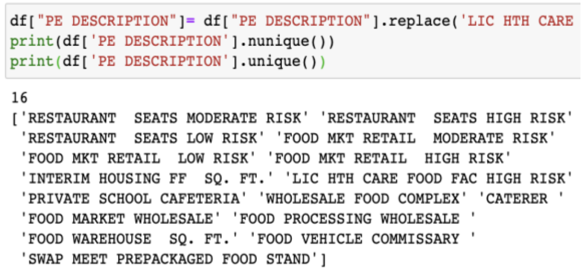
Filtered to ‘ACTIVE’ only



```
print(sum(df.duplicated()))
df.isnull().sum()
```

```
0
```

No Duplicated values

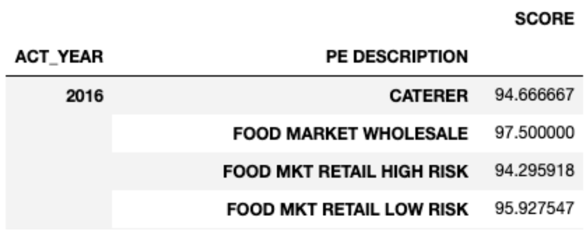


```
df["PE DESCRIPTION"]= df["PE DESCRIPTION"].replace('LIC HTH CARE
print(df['PE DESCRIPTION'].nunique())
print(df['PE DESCRIPTION'].unique())
```

```
16
['RESTAURANT  SEATS MODERATE RISK' 'RESTAURANT  SEATS HIGH RISK'
 'RESTAURANT  SEATS LOW RISK' 'FOOD MKT RETAIL  MODERATE RISK'
 'FOOD MKT RETAIL  LOW RISK' 'FOOD MKT RETAIL  HIGH RISK'
 'INTERIM HOUSING FF  SQ. FT.' 'LIC HTH CARE FOOD FAC HIGH RISK'
 'PRIVATE SCHOOL CAFETERIA' 'WHOLESALE FOOD COMPLEX' 'CATERER '
 'FOOD MARKET WHOLESALE' 'FOOD PROCESSING WHOLESALE '
 'FOOD WAREHOUSE  SQ. FT.' 'FOOD VEHICLE COMMISSARY '
 'SWAP MEET PREPACKAGED FOOD STAND']
```

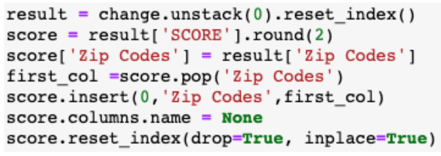
Unique Vendor’s seating’s

Appendix E.8 – Multi-index dataframe output with groupby function: Example Vendors Seating type – Mean

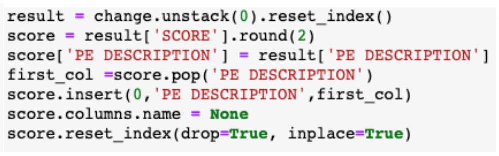


		SCORE
ACT_YEAR	PE DESCRIPTION	
2016	CATERER	94.666667
	FOOD MARKET WHOLESALE	97.500000
	FOOD MKT RETAIL HIGH RISK	94.295918
	FOOD MKT RETAIL LOW RISK	95.927547

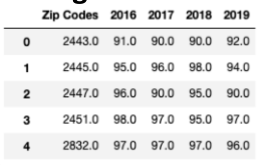
Appendix E.9 – Further manipulation of the groupby dataframes, unstacking the dataset at index 0 and resetting the index, rounding the scores to 2 decimal place and reordering the columns.



```
result = change.unstack(0).reset_index()
score = result['SCORE'].round(2)
score['Zip Codes'] = result['Zip Codes']
first_col =score.pop('Zip Codes')
score.insert(0,'Zip Codes',first_col)
score.columns.name = None
score.reset_index(drop=True, inplace=True)
```

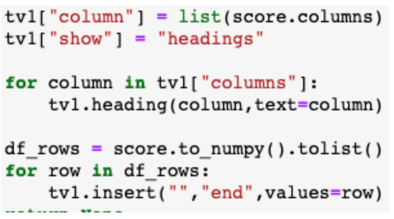


```
result = change.unstack(0).reset_index()
score = result['SCORE'].round(2)
score['PE DESCRIPTION'] = result['PE DESCRIPTION']
first_col =score.pop('PE DESCRIPTION')
score.insert(0,'PE DESCRIPTION',first_col)
score.columns.name = None
score.reset_index(drop=True, inplace=True)
```



	Zip Codes	2016	2017	2018	2019
0	2443.0	91.0	90.0	90.0	92.0
1	2445.0	95.0	96.0	98.0	94.0
2	2447.0	96.0	90.0	95.0	90.0
3	2451.0	98.0	97.0	95.0	97.0
4	2832.0	97.0	97.0	97.0	96.0

Appendix E.10 – convert columns and rows into list for iteration to insert into tkinter treeview table frame enabling GUI Implementation with pandas dataframe



```
tv1["column"] = list(score.columns)
tv1["show"] = "headings"

for column in tv1["columns"]:
    tv1.heading(column,text=column)

df_rows = score.to_numpy().tolist()
for row in df_rows:
    tv1.insert("","end",values=row)
```

Appendix F.1 – Merging ‘df’ dataframe with ‘violations_df’ dataframe on ‘SERIAL NUMBER’ column, rows containing missing values in column ‘VIOLATION CODE’ and ‘PE CODE’ were removed.

```
prep_df = pd.merge(violations_df,df,on=['SERIAL NUMBER'],how='right')
prep_df = prep_df [prep_df ['VIOLATION CODE'].notna()]
```

Appendix F.2 – applying .groupby method to create a dataframe consisting of columns for unique violation codes and the number of violations committed by each establishment. Using .reset_index() to reset multi-index into one level.

```
establishment_table = prep_df.groupby(['VIOLATION CODE'])[['FACILITY NAME']].nunique()
establishment_table = establishment_table.reset_index()
```

Appendix F.3 – applying .sort_values() to sort the violation count from highest to lowest.

```
establishment_table = establishment_table.sort_values('FACILITY NAME',ascending = False)
```

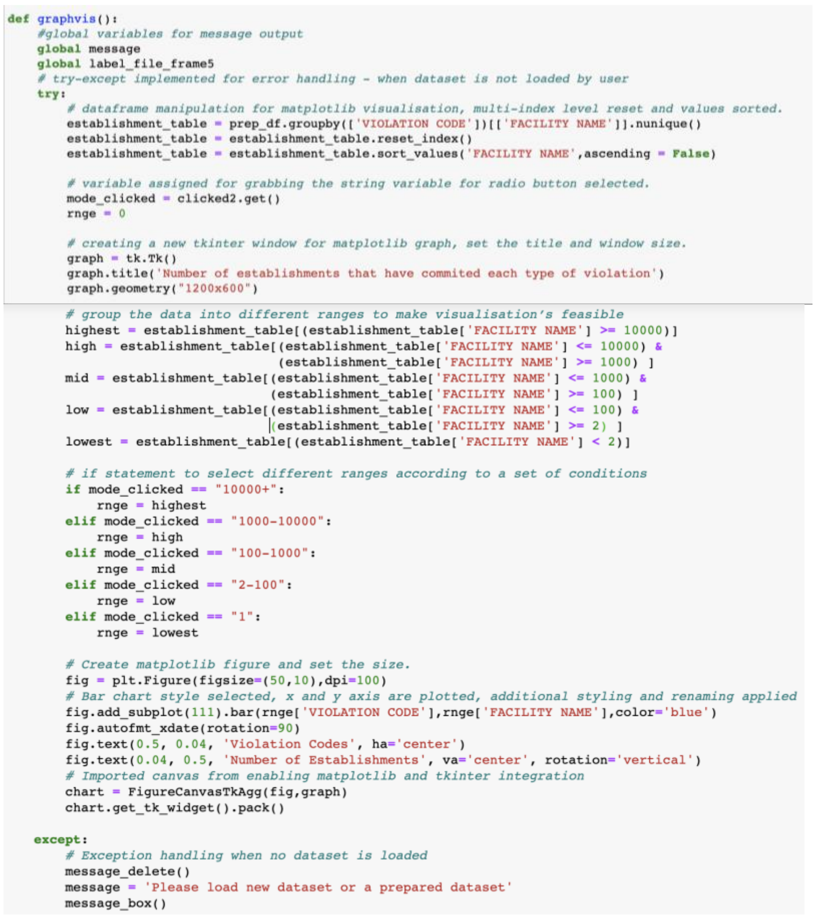
Appendix F.4 – Split violations count into 5 different range categories lowest, low, mid, high and highest.

```
highest = establishment_table[(establishment_table['FACILITY ADDRESS'] > =10000)]
high = establishment_table[(establishment_table['FACILITY ADDRESS'] <= 10000) &
(establishment_table['FACILITY ADDRESS'] >= 1000) ]
mid = establishment_table[(establishment_table['FACILITY ADDRESS'] <= 1000) &
(establishment_table['FACILITY ADDRESS'] >= 100) ]
low = establishment_table[(establishment_table['FACILITY ADDRESS'] <= 100) &
(establishment_table['FACILITY ADDRESS'] >= 2) ]
lowest = establishment_table[(establishment_table['FACILITY ADDRESS'] < 2)]
```

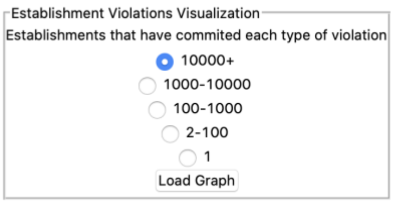
Appendix F.5 – Creating radio button and Load Graph button for user interaction, MODES consists of the radio button selection labels and the load graph button is applied with the graphvis() function to display the matlplotlib bar charts.

```
MODES = [
   ("10000+",("10000+"),
   ("1000-10000","1000-10000"),
   ("100-1000","100-1000"),
   ("2-100","2-100"),
   ("1","2 > 1")
]

for text, mode in MODES:
   tk.Radiobutton(file_frame2,text=text, variable=clicked2,value=mode).pack(anchor=tk.W)
button3 = tk.Button(file_frame2,text="Load Graph",command=lambda: graphvis())
button3.pack()
```

Appendix F.6 – graphvis() function containing if else statement for deciding on range of violation count selected by client to produce matplotlib barchart in tkinter. Violations Count radio button and Load Graph button GUI.

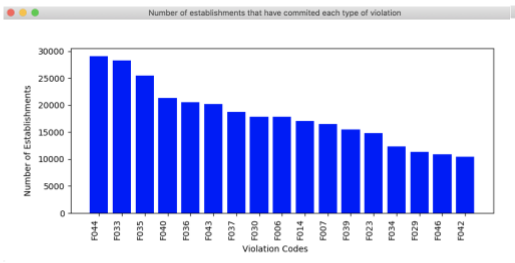


```
def graphvis():
    #global variables for message output
    global message
    global label_file_frame5
    # try-except implemented for error handling - when dataset is not loaded by user
    try:
        # dataframe manipulation for matplotlib visualisation, multi-index level reset and values sorted.
        establishment_table = prep_df.groupby(['VIOLATION CODE'])[['FACILITY NAME']].nunique()
        establishment_table = establishment_table.reset_index()
        establishment_table = establishment_table.sort_values('FACILITY NAME',ascending = False)

        # variable assigned for grabbing the string variable for radio button selected.
        mode_clicked = clicked2.get()
        rnge = 0

        # creating a new tkinter window for matplotlib graph, set the title and window size.
        graph = tk.Tk()
        graph.title('Number of establishments that have commited each type of violation')
        graph.geometry("1200x600")
        # group the data into different ranges to make visualisation’s feasible
        highest = establishment_table[(establishment_table['FACILITY NAME'] >= 10000)]
        high = establishment_table[(establishment_table['FACILITY NAME'] <= 10000) &
                                   (establishment_table['FACILITY NAME'] >= 1000) ]
        mid = establishment_table[(establishment_table['FACILITY NAME'] <= 1000) &
                                  (establishment_table['FACILITY NAME'] >= 100) ]
        low = establishment_table[(establishment_table['FACILITY NAME'] <= 100) &
                                  (establishment_table['FACILITY NAME'] >= 2) ]
        lowest = establishment_table[(establishment_table['FACILITY NAME'] < 2)]

        # if statement to select different ranges according to a set of conditions
        if mode_clicked == "10000+":
            rnge = highest
        elif mode_clicked == "1000-10000":
            rnge = high
        elif mode_clicked == "100-1000":
            rnge = mid
        elif mode_clicked == "2-100":
            rnge = low
        elif mode_clicked == "1":
            rnge = lowest

        # Create matplotlib figure and set the size.
        fig = plt.Figure(figsize=(50,10),dpi=100)
        # Bar chart style selected, x and y axis are plotted, additional styling and renaming applied
        fig.add_subplot(111).bar(rnge['VIOLATION CODE'],rnge['FACILITY NAME'],color='blue')
        fig.autofmt_xdate(rotation=90)
        fig.text(0.5, 0.04, 'Violation Codes', ha='center')
        fig.text(0.04, 0.5, 'Number of Establishments', va='center', rotation='vertical')
        # Imported canvas from enabling matplotlib and tkinter integration
        chart = FigureCanvasTkAgg(fig,graph)
        chart.get_tk_widget().pack()

    except:
        # Exception handling when no dataset is loaded
        message_delete()
        message = 'Please load new dataset or a prepared dataset'
        message_box()
```

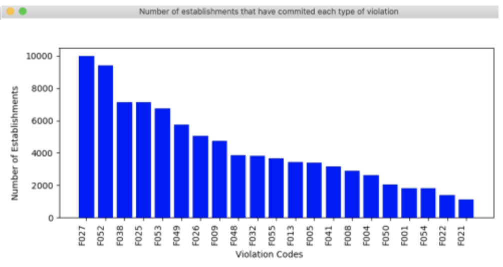

Establishment Violations Visualization
Establishments that have commited each type of violation
● 10000+
○ 1000-10000
○ 100-1000
○ 2-100
○ 1
Load Graph

Grouped range radio buttons

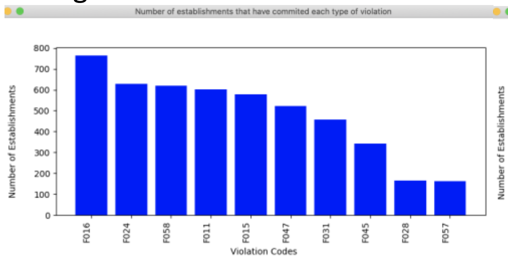
Appendix F.7 – Violations Count radio buttion and Load Graph button GUI.



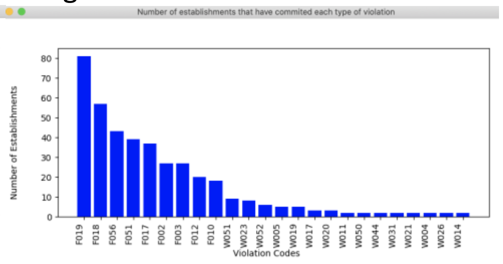
Highest – 10000+



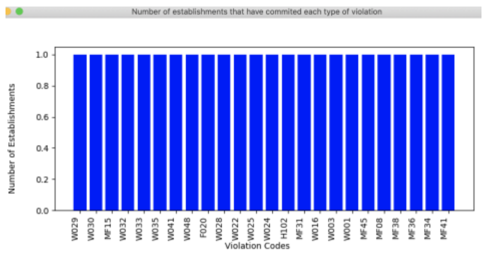
high – 1000-10000



Mid – 100-1000



low – 2-100



Lowest - 1

Appendix G.1 – merging ‘df’ dataframe consisting of inventory and inspections with violations and group_zip dataframe consists of grouped Zip Codes according to first 3 digits.

```
prep_df = pd.merge(violations_df,df,on=['SERIAL NUMBER'],how='right')
prep_df = prep_df[prep_df['VIOLATION CODE'].notna()]
group_zip = prep_df.copy()
group_zip['Zip Codes'] = group_zip['Zip Codes'].astype(str).str[:-2].astype(np.int64)
```

Appendix G.2 – tkinter radio button created with labels ‘Grouped Locations’ and ‘Individual Locations’, radio button choice is set as variable ‘clicked3’.

```
clicked3 = tk.StringVar()
clicked3.set('FACILITY NAME')

selection = [("Grouped Locations","Grouped Locations"),
        ("Individual Locations",'Individual Locations'),]
for text, mode in selection:
    tk.Radiobutton(file_frame4,text=text, variable=clicked3,value=mode).pack(anchor=tk.N)
```

Appendix G.3 – correlation_plot() function created, use of if statements to set radio buttons, decided dataframe is set using ‘locations_grouping’ variable. Entry input is grabbed and set as variable ‘facility_input’.

```
def correlation_plot():

    locations_grouping = 0
    facility_input = 0
    mode_clicked = clicked3.get()
    facility_input = e.get()

    if mode_clicked == "Grouped Locations":
        locations_grouping = group_zip
    elif mode_clicked == "Individual Locations":
        locations_grouping = prep_df
```

Appendix G.4 – Entry button created outside of correlation_plot() function, set as variable ‘e’

```
e = tk.Entry(file_frame4,width=30,)
e.insert(0,'SUBWAY')
```

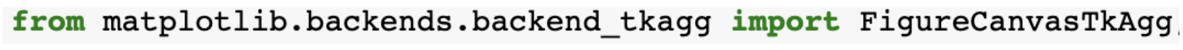
Appendix G.5 – Dataframe is filtered according to the ‘locations_grouping ‘variable consisting of either dataframe group_zip or prep_df

```
correlation_table = locations_grouping.loc[locations_grouping['FACILITY NAME'] ==
facility_input.upper()]
```

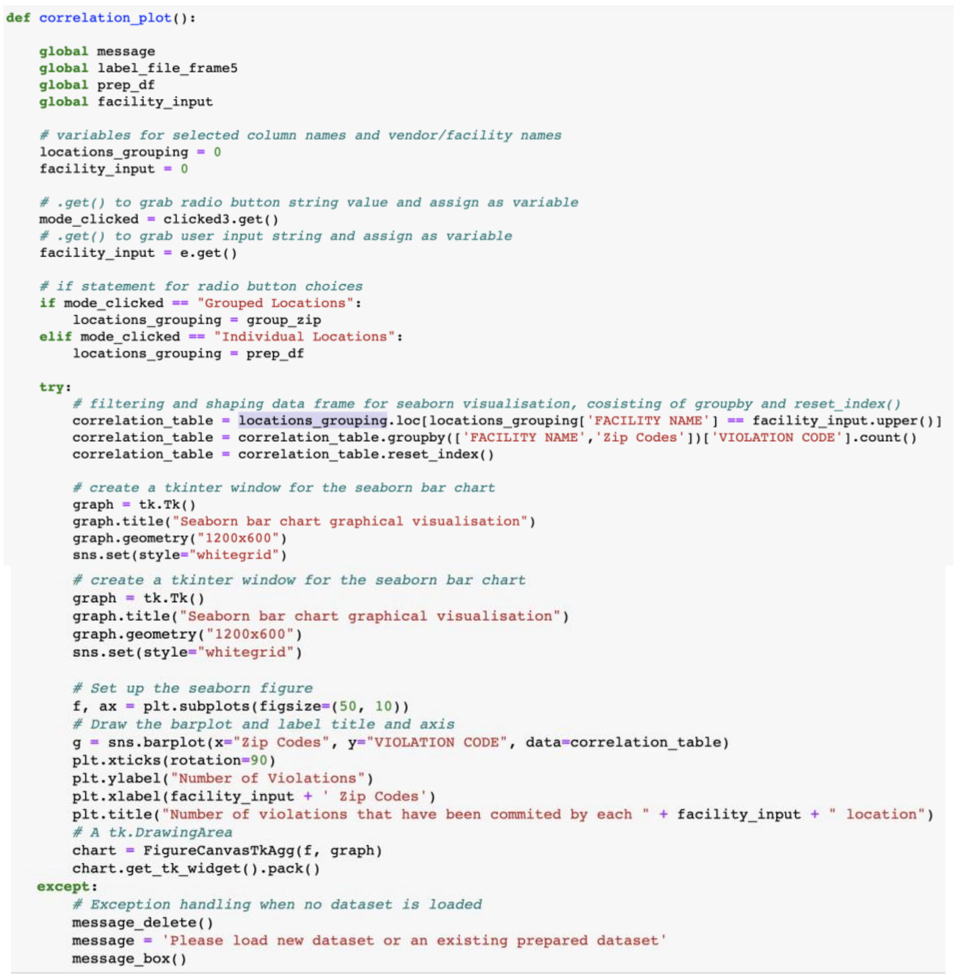
Appendix G.6 – applying groupby() method to shape dataframe into multi-index, index columns consisting of FACILITY NAME and Zip Codes, values consisting of number of violations.

```
correlation_table = correlation_table.groupby([‘FACILITY NAME’,'Zip Codes'])['VIOLATION
CODE'].count()
correlation_table = correlation_table.reset_index()
```

Appendix G.7 – correlation_plot() function containing Seaborn bar PLOT visualisation code integrated with tkinter, FigureCanvasTkAgg import.

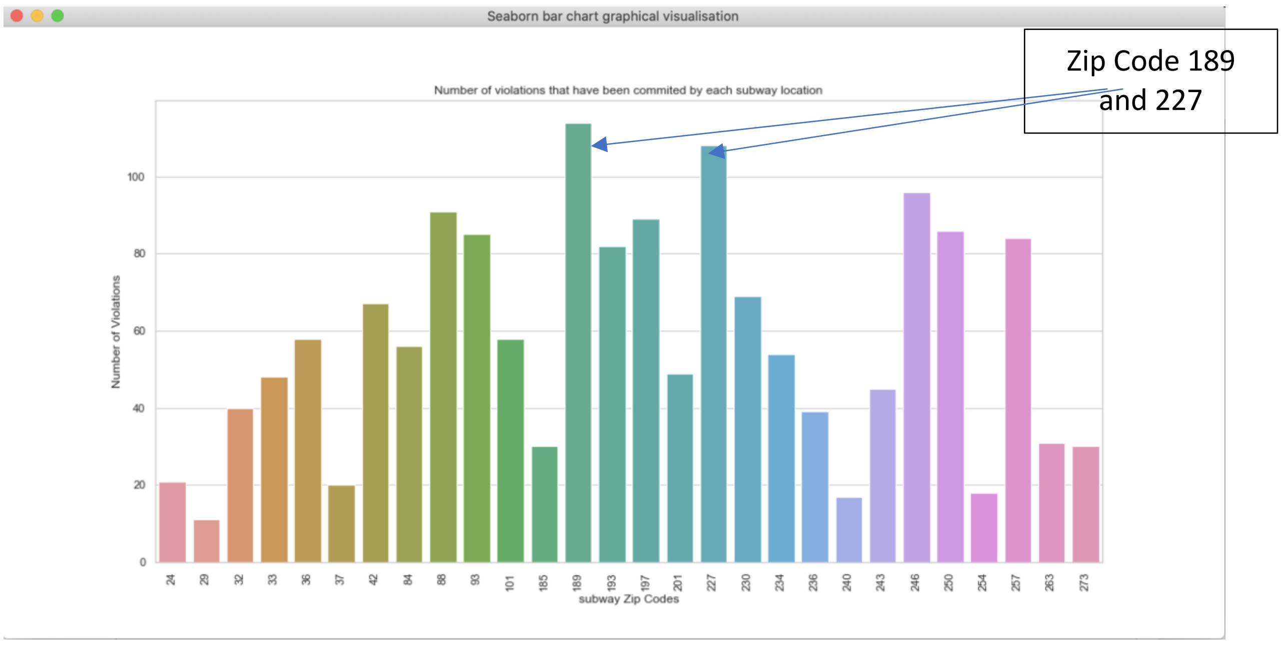


```
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```



```
def correlation_plot():

    global message
    global label_file_frame5
    global prep_df
    global facility_input

    # variables for selected column names and vendor/facility names
    locations_grouping = 0
    facility_input = 0

    # .get() to grab radio button string value and assign as variable
    mode_clicked = clicked3.get()
    # .get() to grab user input string and assign as variable
    facility_input = e.get()

    # if statement for radio button choices
    if mode_clicked == "Grouped Locations":
        locations_grouping = group_zip
    elif mode_clicked == "Individual Locations":
        locations_grouping = prep_df

    try:
        # filtering and shaping data frame for seaborn visualisation, cosisting of groupby and reset_index()
        correlation_table = locations_grouping.loc[locations_grouping['FACILITY NAME'] == facility_input.upper()]
        correlation_table = correlation_table.groupby(['FACILITY NAME','Zip Codes'])['VIOLATION CODE'].count()
        correlation_table = correlation_table.reset_index()

        # create a tkinter window for the seaborn bar chart
        graph = tk.Tk()
        graph.title("Seaborn bar chart graphical visualisation")
        graph.geometry("1200x600")
        sns.set(style="whitegrid")
        # create a tkinter window for the seaborn bar chart
        graph = tk.Tk()
        graph.title("Seaborn bar chart graphical visualisation")
        graph.geometry("1200x600")
        sns.set(style="whitegrid")

        # Set up the seaborn figure
        f, ax = plt.subplots(figsize=(50, 10))
        # Draw the barplot and label title and axis
        g = sns.barplot(x="Zip Codes", y="VIOLATION CODE", data=correlation_table)
        plt.xticks(rotation=90)
        plt.ylabel("Number of Violations")
        plt.xlabel(facility_input + ' Zip Codes')
        plt.title("Number of violations that have been commited by each " + facility_input + " location")
        # A tk.DrawingArea
        chart = FigureCanvasTkAgg(f, graph)
        chart.get_tk_widget().pack()
    except:
        # Exception handling when no dataset is loaded
        message_delete()
        message = 'Please load new dataset or an existing prepared dataset'
        message_box()
```

Appendix G.8 – Seaborn Bar chart for vendor SUBWAY, group location zip code 18907 shows significant number of violations committed in comparison to other Zip Codes, however no correlation between Zip Codes and number of violations due to no pattern identified.


Seaborn bar chart graphical visualisation
Zip Code 189 and 227
Number of violations that have been commited by each subway location
Number of Violations
subway Zip Codes