Number of words used = 2356

Report of Data Analysis Tool

Section 1: Theory supported by code samples

a. Adaptation to a concurrent model (word count = 429)

The flowchart diagram (Figure 1) illustrates how the flow of application can be used from loading to exporting. There are two separate applications may be need: data cleaning (and parsing the data) and next loading the data to generate visualisations of graphs. The application would benefit from concurrency because there are simultaneous tasks that should be run. These tasks include loading all csv files and reading csv files into pandas DataFrame [1]. There is a need to perform multiple task/operation at the backend. Working without threads results in the delayed process. Also, the window does not move until full execution occurs [2]. The python package Threading [3] can help. The approach to create a Tkinter [4] GUI without threading is as follows: First, create Normal Tkinter Window; Second, Add Button with command; Third, Execute Tkinter. The approach to create a Tkinter with Threading is as follows: First, create Normal Tkinter Window; Second, Add Button with command for threading; Third, Execute Tkinter. (Figure 2 shows the example code to create Tkinter without and with Threading.)

There are multiple advantages of threading [5]. First, multi-threaded programs can run much faster on computer systems with multiple CPUs because the threads can actually run concurrently, although it is not really the case for threading in python [6]. Second, the program can continue to respond to inputs. This is true for both single and multiple CPUs. Thirdly, threads in a process can share the memory of global variables. If a global variable is modified in one thread, the change will take effect in all threads. Threads can have local variables. Fourth, threading is lightweight that it has low memory footprint [7]. Fifth, CPython C extension modules can properly release the GIL [8] and run in parallel. Thus, threading is a great choice for I/O-bound applications [9]. However, there are some disadvantages of threading [10]. First, the more threads there are, the more difficult it is to debug and maintain the code [11]. Second, the creation of threads puts a strain on the system in terms of memory and CPU resources [12]. Thirdly, GIL causes python can only use single core of the computer, but not multi-core[13]. Fourth, threading in python is not interruptible or killable [14]. Moreover, due to the way CPython's Python implementation works, threads do not necessarily speed up all tasks. This is due to its interaction with the GIL, which essentially limits the number of Python threads that can run simultaneously to one. Tasks that spend a lot of time waiting for external events are usually good candidates for threads [15].

b. Implementing user interaction (word count = 552)

Designing the GUI

The client required a method to input the datafiles visually and clean the data. Thus, adopting buttons, upload files dialog and input fields is the best practise to eliminate errors and avoid complication of inputting incorrect data. Therefore, the user is only able to choose from given options and load files available on the desktop computer. A status area is required to give consistent feedback in the application. Moreover, labelled input fields are added for different types of data files so that the users know what files need to be uploaded as well as the upload locations of the files. Buttons like “Clean & Export” are added to let users have more control on what they can do with the application. A status bar is added to give continuous feedback to users. Green message is used to verify the user’s action and red message is to show errors. (GUI figure is in Figure 3 and code is in Figure 4.)

Status Bar and Messaging

A status bar is added to tell users about the status of the application by feedback. The status bar is added at the bottom since many applications consider it as secondary in hierarchy [16]. (Figure 5 and Figure 6 show the code of status bar and status messages.) Implementing a status bar with Tkinter is easy. One can simply use a Label widget and configure the text option.

Integrating Tkinter GUI

Widgets like labels, checkboxes, input fields and buttons are available in Tkinter to help build the GUI wireframe. The built-in file dialog function can generate a pop-up window to help locate files, load files and close the program. (Figure 7 illustrates the code of integrating the Tkinter GUI.) There are some advantages of using Tkinter. First, Tkinter is easier and faster to implement than other GUI toolkits. Second, Tkinter ships with three geometry managers: place, pack, and grid. It is easy and powerful to use [17].

Loading the Data

Openfiledialogue function in Tkinter lets locating csv file on the computer system. It is then loaded into pandas DataFrame. After that, the file location is returned in the input field to let users know. (Figures 8 to 10 show the code for loading the data.)

Cleaning the Data

To clean the data, all column names are converted to uppercase. All missing values in categorical features are marked as “Missing”. All missing values in numerical features are replaced with median of the column. Moreover, there are three different files to be cleaned and combined. Thus, cleaning functions are created to be reused and to ease data wrangling consistently. (Figures 11 and 12 show the code for cleaning the data.)

Data Visualization Tool

The client required a tool to produce visualizations from the cleaned dataset and manipulate the range of values to generate the visualization. Thus, Matplotlib [18] library is used to integrate with Tkinter and Pandas [19] to produce visualizations. UI such as slider and checklist is provided by Tkinter to help manipulate the data values. (Figures 13 and 14 shows the code of the Data Visualization Tool.) One advantage of the data visualization tool is that it can show all figures in one window. The disadvantage is that the frame of the plot still appears if one does not select the option of the plot.

c. Evaluating high level languages (word count = 277)

I believe Python to be the most effective in terms of "manipulation of data containers" (compared to Java). Python offers the Pandas library that is used for processing large datasets. Pandas renders fast, flexible and expressive data structures. For instance, DataFrame object in Pandas is able to store two-dimensional heterogeneous tabular data. pandas.read_csv() [20] can be used to read tabular data to store it as DataFrame. (Example from my code is in Figure 8.) Another example is pandas Series. pandas.Series() [21] in pandas is used to convert the numpy array generated to pandas Series object, which can then be fed as a column in the pandas DataFrame. Pandas also provides features such as data conversion and handling of missing values. For data cleaning, DataFrame.select_dtypes().columns.tolist() [22] [23] [24] in pandas is used to select columns of features that are categorical or numerical, and to convert them into lists. DataFrame[column].fillna(DataFrame [column].median(), inplace=True) [25] [26] in pandas is used to replace missing values in numerical features in place with median of the column. DataFrame[column].replace(to_replace = np.nan, value = "Missing",inplace=True) [27] in pandas is used to mark missing values in categorical features as 'Missing' in place. (Example from my code is in Figure 11.)

On the other hand, Java provides Apache Spark [28] API to store data as DataFrame object. But Apache Spark also provides API in Python. Apart from Spark, Java does not provide an API like Pandas in Python for easy data manipulation. Java can only rely on its native data structures to handle data. These data structures include array, linked list, stack, queue, tree and graph [29]. However, the above-mentioned data structures are also available in Python [30].

Section 2: Design decisions supported by code samples

a. Data format (word count = 345)

The data is chosen to be reformatted or exported as JSON file. The datasets are large that they contain more than 140000 records. JSON data format is selected because it

has low data usage and high parsing speed [31]. All files are combined into one data file which contains all necessary features for data visualization and descriptive statistics. (The code for data format is shown in Figure 9.) JSON is faster than XML because it is specifically designed for data interchange. JSON encodes concisely and therefore requires fewer bytes to transfer. JSON parsers are less complex and therefore require less processing time and memory overhead. XML is not only used for data interchange but is designed to be versatile and therefore slower to process [32]. There are many advantages of JSON. First, JSON is easy to use. The syntax of JSON is very simple. The simple use of -> as syntax simplifies data parsing and speeds up data execution: the JSON syntax is so small and lightweight that responses are executed much faster [32]. Second, objects align in code for JSON. This is useful for quickly creating domain objects in dynamic languages, as JSON objects and code objects are matched [32]. Nevertheless, there are also some disadvantages of JSON. First, JSON has no error handling. JSON calls to JSON have no error handling. If the dynamic script is inserted successfully, it will be called and get a perfect response. If it is not inserted, nothing happens. It just fails silently. For example, you cannot catch a 404 error from the server, nor can you cancel or resume the request. You can, however, let the request time out after waiting an appropriate amount of time [32]. Second, JSON is vulnerable. JSON is very dangerous when using untrusted services and untrusted browsers. This is because the JSON service returns a JSON response wrapped in a function call which, if used in an untrusted browser, can be executed by the browser and hacked, which can leave the hosted web application open to a variety of attacks Creation [32].

b. Code constructs (word count = 123)

Data is wrangled into Pandas DataFrame after loading the cleaned JSON file. DataFrames of “small airports”, “medium airports”, “large airports” and “airports with frequencies more than 100 MHz” are created by applying conditions to select data. Pandas and Numpy [33] functions are used to compute statistics of airport frequency such as mean, mode, median and standard deviation. These functions are fast and built-in so that we do not need to worry about the details of mathematical formula. (The code for code construct is shown in Figures 15 and 16.) There are a lot of advantages of pandas DataFrame. First, thanks to default and customised indexes, data frame objects are fast and efficient. Second, pandas has high performance of merging and joining data [29].

c. Data visualisation (word count = 122)

Three histograms and one box-and-whisker plot are generated by the Data Visualization Tool mentioned above. Matplotlib is used. Users can select the maximum value allowed and delete the values higher than the chosen value. Users can also check the box to choose which data visualization(s) to be shown. (The code for the functions of data visualization is shown in Figures 17 and 18.) (Figure 19 illustrates Box-and Whisker Plots and Histograms of Airport frequency.) Histograms facilitate the comparison of data and can also handle a large range of information.

Box-and-whisker charts are very effective and easy to read charts because they combine multiple data into one chart. Box-and-whisker charts make decision-making easier and more effective by comparing different categories of data [41].

d. Data analysis (word count = 114)

From the histograms, box plot and statistics, frequencies around 120 MHz are used most. The median of frequencies used by 'large', 'medium' and 'small' airports are around 121 to 124 MHz. For 'large_airport', the average frequency is the lowest (120 MHz), and the spread/standard deviation of the distribution of the frequency is the smallest (19 MHz). For 'medium_airport', the average frequency is the highest (154 MHz), and the spread/standard deviation of the distribution of the frequency is the smallest (19 MHz). For 'small_airport', the average frequency is the medium (128 MHz), and the spread/standard deviation of the distribution of the frequency is also medium (19 MHz). (Table 1 and Figure 19 support this analysis.)

Section 3: Reflection on the ethics, moral and legal aspects (word count = 394)

People are worried about data in terms of privacy, data collection, secure storage, bias in data sets and misuse of data. They expressed concern that software developers often lack empathy and do not completely comprehend the user base they are developing technology for [34]. Industry practitioners emphasize the legal aspects of the GDPR, entailing its importance and the lawful use of data [35]. Citizens are worried about the use of social media. Social media technology can lead to the normalisation of unacceptable behaviour. The need for training in the online community should be also identified, including how to deal with cyberbullying and how to identify fraud and dishonesty [36]. The academics focus on how computer ethics is taught to computer science students, usually as stand-alone courses or modules, but do not reflect the distributed and interrelated nature of computer ethics issues with many topics in computer science [37]. Researchers such as Moore (2020) argue that computer ethics curricula need to be dynamic, evolving, and related to students' beliefs and lives. Thus, topics that are truly related to students, especially the political nature of computer technology, are considered to be seen as strong drivers for the education of future generations of computer science students [38].

I agree that software engineer should not be regulated by a central body. For instance, professional licensing of software engineer has been criticized [39]. Firstly, the discipline of software engineering is very immature. Second, years of calculus, physics and chemistry that software engineers must study to pass their exams are irrelevant to most developers. Many computer science students do not have an engineering degree and may not be eligible to pass the engineering exams. In addition, many regulations, laws, standards, and other authoritative rules directly affect how regulated organizations develop, test, and maintain the software systems they use [40]. Compliance with laws and regulations is often a cumbersome task, especially for global software companies with customers in different industry sectors and geographically dispersed. In addition, laws are often created by different legislatures with little effort to harmonise or articulate similar legal requirements, often resulting in overlapping and contradictory regulations [39]. Inconsistency is also due to the fact that regulations are rarely developed from scratch. They often rely on other

existing legislation to form an overall network of provisions, and changes in one place may spread to many other places [40].

Appendix

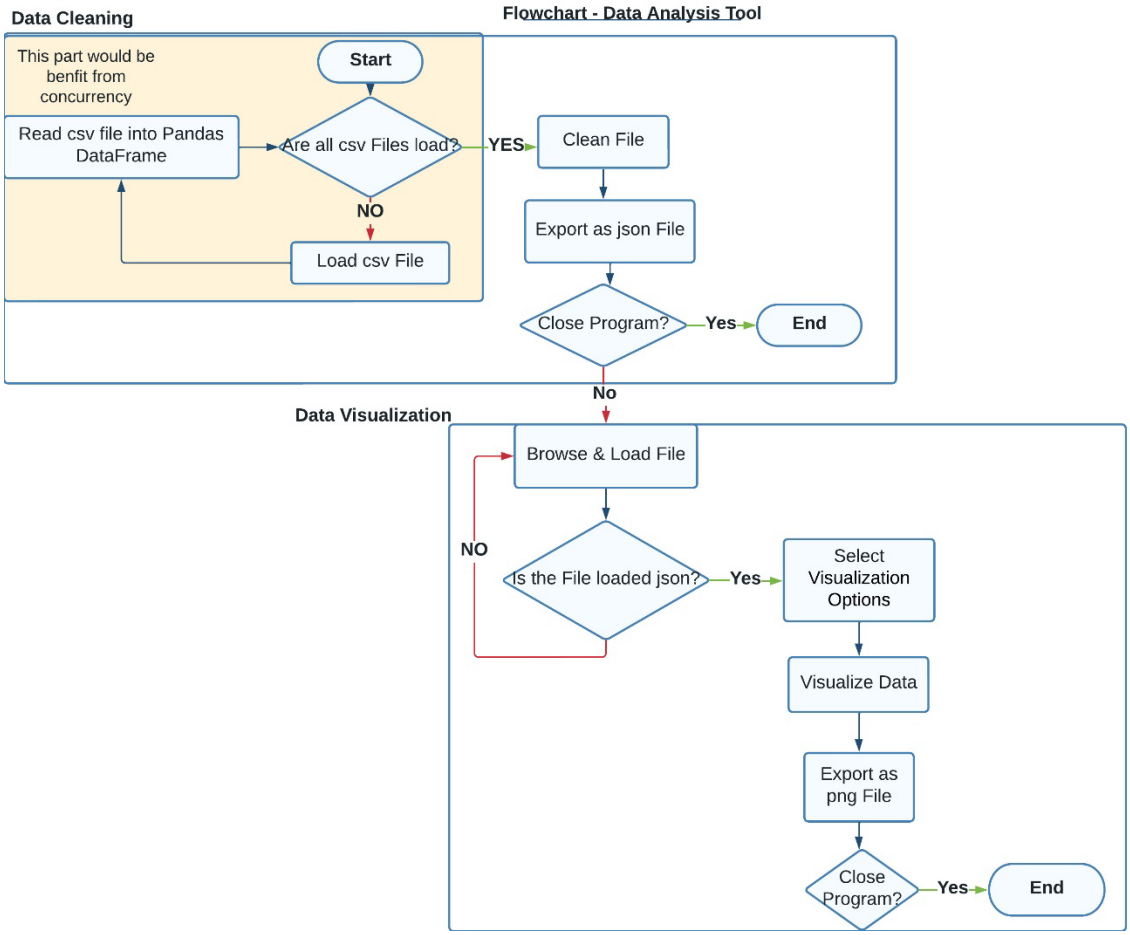

Figure 1: Flowchart of the Data Analysis Tool Program.

Without Threading

In [ ]:
```
def open_file_1():
    """
    Browse Airport Data File to upload function.
    """
    global df_airport, status_bar
    try:
        # open csv file only
        file = filedialog.askopenfile(filetypes =[('Data Files', '*.csv')])
        # fill in missing value with NaN
        df_airport = pd.read_csv(file.name,keep_default_na=False, na_values=missing_values)
        input_text = file.name
        # update input field with file upload directory/path
        input_airport.delete(0, tk.END)
        input_airport.insert(0, input_text)
    except AttributeError:
        status_bar.configure(foreground='red')
        status_text.set('*ERROR* Please open a valid location for the Airport Data file')
        print("Please open a valid location for the Airport Data file")
```

In [ ]:
```
# Import Module
from tkinter import *
from tkinter import ttk

# create instance of Tk class
window = tk.Tk()
window.title("Data Analysis Tool")

# add text label
ttk.Label(clean_tab, text="Airport Data File", \
          style="STD.Label").grid(row= 1, column=0, padx=10, pady=10, sticky=W)

# add input fields to clean_tab
input_airport = ttk.Entry(clean_tab)

# position of input fields
input_airport.grid(row=1, column=1, padx=5, pady=0)

# add Tabs to window
tab_control = ttk.Notebook(window)
clean_tab = ttk.Frame(tab_control)

# file browse buttons
btn_browse_1 = ttk.Button(clean_tab, text="Browse", command = lambda:open_file_1())

# browse button layout
btn_browse_1.grid(row=1, column=2)

# execute Tkinter
window.mainloop()
```

With Threading

In [ ]:
```
# Import Module
from tkinter import *
from tkinter import ttk
from threading import *

# add text label
ttk.Label(clean_tab, text="Airport Data File", \
          style="STD.Label").grid(row= 1, column=0, padx=10, pady=10, sticky=W)

# add input fields to clean_tab
input_airport = ttk.Entry(clean_tab)

# position of input fields
input_airport.grid(row=1, column=1, padx=5, pady=0)

# add Tabs to window
tab_control = ttk.Notebook(window)
clean_tab = ttk.Frame(tab_control)

# use threading
def threading():
    # Call open_file_1 function
    t1 = Thread(target=open_file_1)
    t1.start()

# file browse buttons
btn_browse_1 = ttk.Button((clean_tab,text="Browse",command = threading)

# browse button layout
btn_browse_1.grid(row=1, column=2)

# execute Tkinter
window.mainloop()

# Create Object
window = Tk()
```

Figure 2: Example code to create Tkinter without and with Threading.

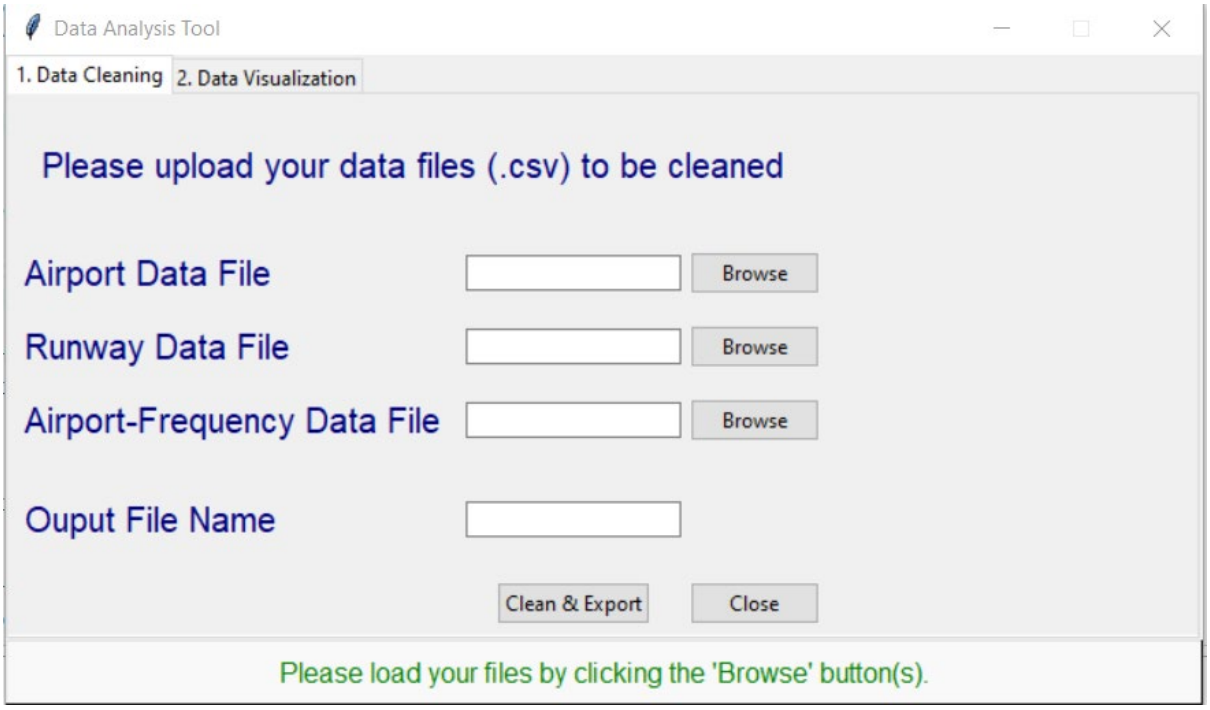

Figure 3: Data Cleaning Tool.

7.1 Tab 1: Data Cleaning

In [23]:
```
# add text label
ttk.Label(clean_tab, text="Airport Data File", \
          style="STD.Label").grid(row= 1, column=0, padx=10, pady=10, sticky=W)
ttk.Label(clean_tab, text="Runway Data File", \
          style="STD.Label").grid(row= 2, column=0, padx=10, pady=10, sticky=W)
ttk.Label(clean_tab, text="Airport-Frequency Data File", \
          style="STD.Label").grid(row= 3, column=0, padx=10, pady=10, sticky=W)
ttk.Label(clean_tab, text="Ouput File Name", \
          style="STD.Label").grid(row=4, column=0, padx=10, pady=25, sticky=W)
ttk.Label(clean_tab, text="Please upload your data files (.csv) to be cleaned", \
          style='STD.Label').grid(row=0, column=0, columnspan=3, padx=10, pady=30)

# add input fields to clean_tab
input_airport = ttk.Entry(clean_tab)
input_runway = ttk.Entry(clean_tab)
input_frequency = ttk.Entry(clean_tab)
input_filename = ttk.Entry(clean_tab)

# position of input fields
input_airport.grid(row=1, column=1, padx=5, pady=0)
input_runway.grid(row=2, column=1, padx=5, pady=0)
input_frequency.grid(row=3, column=1, padx=5, pady=0)
input_filename.grid(row=4, column=1, padx=0, pady=0)

# file browse buttons
btn_browse_1 = ttk.Button(clean_tab, text="Browse", command = lambda:open_file_1())
btn_browse_2 = ttk.Button(clean_tab, text="Browse", command = lambda:open_file_2())
btn_browse_3 = ttk.Button(clean_tab, text="Browse", command = lambda:open_file_3())

# browse button layout
btn_browse_1.grid(row=1, column=2)
btn_browse_2.grid(row=2, column=2)
btn_browse_3.grid(row=3, column=2)

# action buttons
btn_clean = ttk.Button(clean_tab, text="Clean & Export", command= lambda:clean_files())
btn_close = ttk.Button(clean_tab, text="Close", command=exit)

# action button layout
btn_clean.grid(row=5, column=1)
btn_close.grid(row=5, column=2, sticky=W)
```

Figure 4: Code for Data Cleaning Tool.

4 Status Bar ¶

In [5]:
```
status_text = StringVar(window)
status_text.set("Please load your files by clicking the 'Browse' button(s).")
status_bar = Label(window, textvariable=status_text, bd=1, relief=tk.SUNKEN, font=('Helvetica', 12, 'normal'))
status_bar.configure(foreground='green')
status_bar.pack(side=BOTTOM, fill=X)
status_bar.config(background='#F9F9F9', relief=RAISED, height=2)
```

Figure 5: Code for Status Bar.

6.1 Status Messages

In [7]:
```
def status_preparing():
    """
    Show Status of "Preparing to clean loaded files---".
    """
    status_bar.configure(foreground='green')
    status_text.set("*STATUS* Preparing to clean loaded files---")

def status_exported():
    """
    Show Status of exporting data files.
    """
    status_text.set("*SUCCESS* You have exported the data files!")
    status_bar.configure(foreground='green')

def status_cleaned():
    """
    Show Status of 'All files are cleaned'.
    """
    status_bar.configure(foreground='green')
    status_text.set("*SUCCESS* All files are cleaned!")

def error_files():
    """
    Show Status of error with file(s).
    """
    status_bar.configure(foreground='red')
    status_text.set('*ERROR* There is a problem with your file(s), please check before exporting!')
    print("There is a problem with your files, please check before exporting!")

def error_location():
    """
    Show Status of error with file location.
    """
    status_bar.configure(foreground='red')
    status_text.set('*ERROR* Please load valid location for the data file by clicking "Browse".')
```

Figure 6: Code for Status Messages.

2 Import packages

In [2]:
```
import json # handle JSON file
import os  # File Input/Output
import numpy as np # linear algebra
import pandas as pd # data processing
import matplotlib.pyplot as plt # visualization
import matplotlib.cbook as cbook
import tkinter as tk # GUI application
from tkinter import ttk
from tkinter import *
from tkinter.filedialog import asksaveasfile
from tkinter.filedialog import asksaveasfilename
from tkinter import filedialog
from tkinter import messagebox
from datetime import datetime # datetime object manipulation
```

In [3]:
```
# data visualization in new window
%matplotlib qt
```

3 Global Variables

In [4]:
```
# handle missing values
missing_values = ["NA", "NaN", "N/A", "na", r"^\s*$", "", " "]

# create instance of Tk class
window = tk.Tk()
window.title("Data Analysis Tool")

# disable windows from resizing
window.resizable(False, False)

# window sizing
window_height = 380
window_width = 600
screen_width = window.winfo_screenwidth()
screen_height = window.winfo_screenheight()

# place window in the middle of screen
x_coordinate = int((screen_width/2) - (window_width/2))
y_coordinate = int((screen_height/2) - (window_height/2))
window.geometry("{}x{}+{}+{}".format(window_width, window_height, x_coordinate, y_coordinate))

# add Tabs to window
tab_control = ttk.Notebook(window)
clean_tab = ttk.Frame(tab_control)
graph_tab = ttk.Frame(tab_control)

# add text to tab and styling
tab_control.add(clean_tab, text="1. Data Cleaning")
tab_control.add(graph_tab, text="2. Data Visualization")
tab_control.pack(expand=1, fill="both")
```

Figure 7: Code for integrating Tkinter GUI.

6.4 Upload/Browse for Files

In [10]:

```
def open_file_1():
    """
    Browse Airport Data File to upload function.
    """
    global df_airport, status_bar
    try:
        # open csv file only
        file = filedialog.askopenfile(filetypes =[('Data Files', '*.csv')])
        # fill in missing value with NaN
        df_airport = pd.read_csv(file.name,keep_default_na=False, na_values=missing_values)
        input_text = file.name
        # update input field with file upload directory/path
        input_airport.delete(0, tk.END)
        input_airport.insert(0, input_text)
    except AttributeError:
        status_bar.configure(foreground='red')
        status_text.set('*ERROR* Please open a valid location for the Airport Data file')
        print("Please open a valid location for the Airport Data file")

def open_file_2():
    """
    Browse Runway Data File to upload function.
    """
    global df_runway
    try:
        file = filedialog.askopenfile(filetypes =[('Data Files', '*.csv')])
        df_runway = pd.read_csv(file.name,keep_default_na=False, na_values=missing_values)
        input_text = file.name
        # update input field with file upload directory/path
        input_runway.delete(0, tk.END)
        input_runway.insert(0, input_text)
    except AttributeError:
        status_bar.configure(foreground='red')
        status_text.set('*ERROR* Please open a valid location for the Runway Data file')
        print("Please open a valid location for the Runway Data file")

def open_file_3():
    """
    Browse Airport-Frequency Data File to upload function.
    """
    global df_frequency
    try:
        file = filedialog.askopenfile(filetypes =[('Data Files', '*.csv')])
        df_frequency = pd.read_csv(file.name,keep_default_na=False, na_values=missing_values)
        input_text = file.name
        # update input field with file upload directory/path
        input_frequency.delete(0, tk.END)
        input_frequency.insert(0, input_text)
    except AttributeError:
        status_bar.configure(foreground='red')
        status_text.set('*ERROR* Please open a valid location for the Airport-Frequency Data file')
        print("Please open a valid location for the Airport-Frequency Data file")
def open_file_4():
    """
    Browse Cleaned Data File (JSON).
    """
    global json_file
    try:
        file = filedialog.askopenfile(filetypes =[('JSON Files', '*.json')])
        # read json file
        json_file = pd.read_json(file)
        input_text = file.name
        # update input field with file upload directory/path
        input_graph.delete(0, tk.END)
        input_graph.insert(0, input_text)
        status_text.set('Your JSON file has been loaded.')
    except AttributeError or ValueError:
        error_location()
        print("Please open a valid location for the data file")
```

Figure 8: Code for loading the data – Browse for Files.

6.2 Export to JSON

In [8]:
```
def export_json(file):
    """
    Export cleaned file as JSON file.
    """
    global status_bar, new_json_name, file_name
    file_name = "output"
    if len(input_filename.get()) != 0:
        file_name = input_filename.get()
    else:
        file_name = "output"
    try:
        # New json name
        new_json_name = file_name + ".json"
        file.to_json(new_json_name, index='true')
        status_exported()
    except NameError:
        error_files()
```

Figure 9: Code for loading the data – Export to JSON.

In [23]:
```
# add text label
ttk.Label(clean_tab, text="Airport Data File", \
          style="STD.Label").grid(row= 1, column=0, padx=10, pady=10, sticky=W)
ttk.Label(clean_tab, text="Runway Data File", \
          style="STD.Label").grid(row= 2, column=0, padx=10, pady=10, sticky=W)
ttk.Label(clean_tab, text="Airport-Frequency Data File", \
          style="STD.Label").grid(row= 3, column=0, padx=10, pady=10, sticky=W)
ttk.Label(clean_tab, text="Ouput File Name", \
          style="STD.Label").grid(row=4, column=0, padx=10, pady=25, sticky=W)
ttk.Label(clean_tab, text="Please upload your data files (.csv) to be cleaned", \
          style='STD.Label').grid(row=0, column=0, columnspan=3, padx=10, pady=30)

# add input fields to clean_tab
input_airport = ttk.Entry(clean_tab)
input_runway = ttk.Entry(clean_tab)
input_frequency = ttk.Entry(clean_tab)
input_filename = ttk.Entry(clean_tab)

# position of input fields
input_airport.grid(row=1, column=1, padx=5, pady=0)
input_runway.grid(row=2, column=1, padx=5, pady=0)
input_frequency.grid(row=3, column=1, padx=5, pady=0)
input_filename.grid(row=4, column=1, padx=0, pady=0)

# file browse buttons
btn_browse_1 = ttk.Button(clean_tab, text="Browse", command = lambda:open_file_1())
btn_browse_2 = ttk.Button(clean_tab, text="Browse", command = lambda:open_file_2())
btn_browse_3 = ttk.Button(clean_tab, text="Browse", command = lambda:open_file_3())

# browse button layout
btn_browse_1.grid(row=1, column=2)
btn_browse_2.grid(row=2, column=2)
btn_browse_3.grid(row=3, column=2)

# action buttons
btn_clean = ttk.Button(clean_tab, text="Clean & Export", command= lambda:clean_files())
btn_close = ttk.Button(clean_tab, text="Close", command=exit)

# action button layout
btn_clean.grid(row=5, column=1)
btn_close.grid(row=5, column=2, sticky=W)
```

Figure 10: Code for loading the data – Tkinter Buttons.

6.5 Generic Clean

In [11]:
```
def generic_clean(df):
    """
    Clean data generically.
    """
    print("***Convert all column names to uppercase")
    df.columns = [x.upper() for x in df.columns]

    # get all categorical features
    cat_features = df.select_dtypes(include=['object','category']).columns.tolist()
    # get all numerical features
    num_features = df.select_dtypes(include=['number']).columns.tolist()

    # replace NaN in categorical features with 'Missing'
    print("***Mark missing values in categorical features as 'Missing'")
    for cat in cat_features:
        df[cat].replace(to_replace = np.nan, value = "Missing",inplace=True)

    # replace NaN in numerical features with median of the column
    print("***Replace missing values in numerical features with median of the column")
    for num in num_features:
        df[num].fillna(df[num].median(), inplace=True)

    print("***Make some columns into upper case")
    # turn yes/no into 1/0 in 'SCHEDULED_SERVICE' in 'airports.csv'
    if 'SCHEDULED_SERVICE' in df.columns:
        df["SCHEDULED_SERVICE"] = pd.Series(np.where(df.SCHEDULED_SERVICE.values == 'yes', 1, 0), df.index)

    # make "SURFACE" in 'runways.csv' into upper case
    if "SURFACE" in df.columns:
        df["SURFACE"] = df["SURFACE"].str.upper()

    # make "DESCRIPTION" in 'airport-frequencies.csv' into upper case
    if "DESCRIPTION" in df.columns:
        df["DESCRIPTION"] = df["DESCRIPTION"].str.upper()

    print("***Finish Cleaning ")
    print(df.head())

    return df
```

Figure 11: Code for cleaning the data – Generic Clean.

```
In [13]: def clean_files():
             """
             Clean File Function Button.
             """
             global df_merge
             status_preparing()
             try:
                 # only clean if all files are present
                 if (len(input_frequency.get()) != 0) and (len(input_runway.get()) != 0) and (len(input_airport.get()) != 0):
                     global df1, df2, df3
                     status_cleaned()

                     # clean and created subset of airport
                     df1 = generic_clean(df_airport)
                     # clean and created subset of runway
                     df2 = generic_clean(df_runway)
                     # clean and created subset of frequency
                     df3 = generic_clean(df_frequency)

                     ### remove data from airports that have a 'type' 'closed' ###
                     # remove records
                     df_airports_cleaned_remove = df1[df1["TYPE"]!="closed"]
                     # reset index
                     df_airports_cleaned_remove = df_airports_cleaned_remove.reset_index(drop=True)

                     ### get only UK (GB) airports ###
                     # get uk records
                     df_airports_cleaned_remove_uk = df_airports_cleaned_remove[df_airports_cleaned_remove["ISO_COUNTRY"]=="GB"]
                     # reset index
                     df_airports_cleaned_remove_uk = df_airports_cleaned_remove_uk.reset_index(drop=True)

                     ### extract "type" column out into a new column, one for each category of airport,
                     ### for all UK(GB) airports, i.e., large_airport, medium_airport, small_airport ###
                     df_airports_cleaned_remove_uk["TYPE_SELECTED"] = df_airports_cleaned_remove_uk["TYPE"].apply(select_type)

                     ### join each category, large_airport, medium_airport, small_airport
                     ### to the communication frequencies ' frequency_mhz'
                     ### that the airport uses for communication ensuring
                     ### that each airport in all categories is correctly matched with its communication frequencies
                     # select only large_airport, medium_airport, small_airport
                     df_airports_selected = df_airports_cleaned_remove_uk[df_airports_cleaned_remove_uk["TYPE_SELECTED"]==True]
                     # reset index
                     df_airports_selected = df_airports_selected.reset_index(drop=True)

                     # join df_airports_selected and df_frequencies_cleaned on ID
                     df_merge = pd.merge(df_airports_selected, df3, left_on="IDENT", right_on="AIRPORT_IDENT", how="inner")
                     # reset index
                     df_merge = df_merge.reset_index(drop=True)
                     print(df_merge.head())

                     # export as JSON file
                     export_json(df_merge)

                 else:
                     # if one or more files are not present, there is an error
                     error_files()

             except NameError:
                 print("There is an issue with one or more of your file uploads, please check them then try again.")
```

Figure 12: Code for cleaning the data – clean_files().

7.2 Tab 2: Data Visualization ¶

```
ttk.Label(graph_tab, \
          text="Please upload your cleaned data file for data visualization/analysis", \
          style='STD.Label').grid(row=0, column=0, columnspan=3, padx=10, pady=20)

#############################################################################################
# I. upload data file
#############################################################################################
ttk.Label(graph_tab, text="Cleaned Data File", \
          style='STD.Label').grid(row=1, column=0, padx=10, pady=10, sticky=W)

# add input fields to graph_tab
input_graph = ttk.Entry(graph_tab)
# position of input fields
input_graph.grid(row=1, column=1, sticky=W)

# browse button
btn_graph_browse = ttk.Button(graph_tab, text="Browse", command = lambda:open_file_4())
btn_graph_browse.grid(row=1, column=2, sticky=W)

#############################################################################################
```

Figure 13: code of the Data Visualization Tool Part I.

```
#############################################################################################
# II. select graphs
#############################################################################################
ttk.Label(graph_tab, \
          text="Select the visualisation (s)/table you would like to produce", \
          style="STD.Label").grid(row= 2, column=0, padx=10, pady=10, sticky=W, columnspan=3)

# check boxes
v_stat = tk.IntVar()
v_hist = tk.IntVar()
graph = tk.IntVar()

# table
check_table = Checkbutton(graph_tab, \
                          text="Statistics of Airport Frequencies", variable=v_stat, \
                          onvalue=1, offvalue=0, background='#ececec')
check_table.grid(row=3,column=0, sticky=W,columnspan=2,  padx=5, pady=5)

# histograms
check_hist = Checkbutton(graph_tab, \
                        text="Histograms of Airport Frequencies", variable=v_hist, \
                        onvalue=1, offvalue=0, background='#ececec')
check_hist.grid(row=3,column=1, sticky=W)

# box-and-whisker plots
check_box = Checkbutton(graph_tab, \
                          text="Box-and-whisker plots of Airport Frequencies", \
                          variable=graph, onvalue=1, offvalue=0, background='#ececec')
check_box.grid(row=4,column=0, sticky=W, columnspan=3, padx=5)

# Remove values between + Slider
ttk.Label(graph_tab, text="Delete values higher than the chosen value", \
          style="STD.Label").grid(row= 5, column=0, padx=10, sticky=W)
# Add slider so they dont enter wrong values
var = DoubleVar()
slider_max = Scale(graph_tab, from_=0, to=1000, \
                   orient=HORIZONTAL, bg = "#ececec", length=200, variable=var)
slider_max.grid(row=5, column=1, columnspan=2, sticky=W, padx=5)


# option buttons
# reset
btn_export = ttk.Button(graph_tab, text="Export", command = lambda:export_graphs())
btn_export.grid(row=9, column=0, padx=0, sticky=E)

# visualize data
btn_graph = ttk.Button(graph_tab, text="Visualize Data", command = lambda:visualize_data())
btn_graph.grid(row=9, column=1, pady=20)

# close button
btn_close = ttk.Button(graph_tab, text="Close", command=exit)
btn_close.grid(row=9, column=2, sticky=W)

# execute Tkinter
window.mainloop()
```

Figure 14: code of the Data Visualization Tool Part II.

6.8 Data Wrangling ¶

```
def data_wrangling():
    """
    Data Wrangling.
    """
    global df_small, df_medium, \
           df_la, df_large
    # select records of "small_airport"
    df_small = json_file[json_file["TYPE_x"]=="small_airport"]
    # select records of "small_airport"
    df_medium = json_file[json_file["TYPE_x"]=="medium_airport"]
    # select records for 'large_airport'
    df_la = json_file[json_file["TYPE_x"]=="large_airport"]
    # select records for frequencies more than 100 mhz
    df_large = json_file[json_file["FREQUENCY_MHZ"]>100]
```

```
def delete_values_higher():
    """
    Delete values higher than the chosen value.
    """
    global a, b, c, d
    # delete values beyond a specified value
    a = df_la["FREQUENCY_MHZ"]
    b = df_medium["FREQUENCY_MHZ"]
    c = df_small["FREQUENCY_MHZ"]
    d = df_large["FREQUENCY_MHZ"]
    a = a[a <= var.get()]
    b = b[b <= var.get()]
    c = c[c <= var.get()]
    d = d[d <= var.get()]
```

6.9 Calculate Statistics

```
def calculate(x):
    """
    Calculate the mean, mode, median,
    and standard deviation of frequency of the input airport.
    """
    #calculate the mean
    mean = x.mean()
    # calculate the mode
    mode = x.mode()
    # convert series of mode into list
    mode = mode.tolist()
    # calculate median
    median = x.median()
    # calculate standard deviation
    sd = np.std(x)
    return mean, mode, median, sd
```

Figure 15: code of the Data Wrangling and Statistics Calculation.

6.10 Table of Statistics

```
def table():
    """
    Produce a table of mean, mode and median
    for frequencies of different categories of airport.
    """
    global table_la
    # delete values
    delete_values_higher()
    # calculate statistics
    stat_small = list(calculate(c))
    stat_medium = list(calculate(b))
    stat_la = list(calculate(a))
    stat_large = list(calculate(d))
    table_la = pd.DataFrame([stat_la])
    table_la.columns =['Mean', 'Mode', 'Median', 'Standard Deviation']
    # append rows
    table_la = table_la.append(pd.Series(stat_medium, index = table_la.columns), ignore_index=True)
    table_la = table_la.append(pd.Series(stat_small, index = table_la.columns), ignore_index=True)
    table_la = table_la.append(pd.Series(stat_large, index = table_la.columns), ignore_index=True)
    # change the row indexes
    table_la.index = ['large_airport', 'medium_airport', 'small_airport', 'Frequencies more than 100 mhz']
    cols = list(table_la.columns)
    root = tk.Tk()
    root.title("Statistics of Airport Frequencies")
    tree = ttk.Treeview(root)
    tree.pack()
    tree["columns"] = cols
    for i in cols:
        tree.column(i, anchor="w")
        tree.heading(i, text=i, anchor='w')

    for index, row in table_la.iloc[::-1].iterrows():
        tree.insert("",0,text=index,values=list(row))
```

Figure 16: code of the Data Wrangling and Statistics Calculation.

6.11 Create Graphs

```
def create_graphs():
    """
    Create graphs.
    """
    # name graph window
    global graphs, graph_box_whisker, \
    graph_histogram1, graph_histogram2, graph_histogram3
    graphs = plt.figure()
    graphs.set_figwidth(35)
    graphs.set_figheight(6)
    # add location of subplots
    graph_box_whisker = graphs.add_subplot(1, 4, 1)
    graph_histogram1 = graphs.add_subplot(1, 4, 2)
    graph_histogram2 = graphs.add_subplot(1, 4, 3)
    graph_histogram3 = graphs.add_subplot(1, 4, 4)
```

6.12 Box-and-Whisker Plots

```
def box_whisker():
    """
    Produce the box-and-whisker plots of frequencies for all airports.
    """
    # delete values
    delete_values_higher()
    # produce the box-and-whisker plots of frequencies for all airports
    data = [a, b, c]
    # Creating plot
    graph_box_whisker.boxplot(data, showfliers=False)
    graph_box_whisker.set_xticks([1, 2, 3], ['large', 'medium', 'small'])
    graph_box_whisker.set_ylabel("Frequency (MHz)")
    graph_box_whisker.set_xlabel("Airport")
    graph_box_whisker.set_title("Box-and-whisker plots of Airport Frequencies")
```

Figure 17: code of the Functions to create graphs and box-and-whisker plots.

6.13 Histograms

```
def histograms():
    """
    Produce histograms of frequencies for all airports.
    """
    binwidth = 10
    # delete values
    delete_values_higher()
    # plot histogram of frequency of "small_airport"
    # select records of "small_airport"
    data = c
    graph_histogram1.hist(data, bins=np.arange(min(data), max(data) + binwidth, binwidth), alpha=0.5, histtype='bar', ec='black')
    graph_histogram1.set_title("Histogram of Frequency of 'small_airport'")
    graph_histogram1.set_xlabel("Frequency (MHz)")
    graph_histogram1.set_ylabel("Count")

    # plot histogram of frequency of "medium_airport"
    # select records of "small_airport"
    data = b
    graph_histogram2.hist(data, bins=np.arange(min(data), max(data) + binwidth, binwidth), alpha=0.5, histtype='bar', ec='black')
    graph_histogram2.set_title("Histogram of Frequency of 'medium_airport'")
    graph_histogram2.set_xlabel("Frequency (MHz)")
    graph_histogram2.set_ylabel("Count")

    # plot histogram of frequency of "large_airport"
    # select records for 'large_airport'
    data = a
    graph_histogram3.hist(data, bins=np.arange(min(data), max(data) + binwidth, binwidth), alpha=0.5, histtype='bar', ec='black')
    graph_histogram3.set_title("Histogram of Frequency of 'large_airport'")
    graph_histogram3.set_xlabel("Frequency (MHz)")
    graph_histogram3.set_ylabel("Count")
```

6.14 Graphs Function Button

```
def select_graph():
    """
    Select graphs.
    """
    data_wrangling()
    create_graphs()
    if graph.get() == 1:
        # produce the box-and-whisker plots of frequencies for all airports
        box_whisker()
    if v_stat.get() == 1:
        # produce a table of mean, mode and median
        # for frequencies of different categories of airport
        table()
    if v_hist.get() == 1:
        # produce histograms
        histograms()
    else:
        status_text.set('No graphs have been selected. Please pick at least 1')
```

Figure 18: code of the Functions to select graphs and produce histograms.

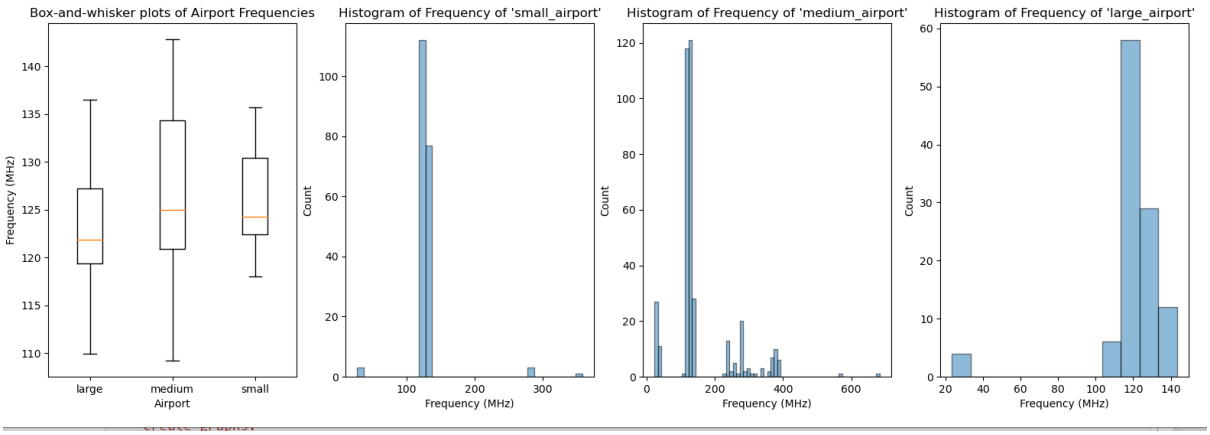

Figure 19: Box-and Whisker Plots and Histograms of Airport frequency .

	Mean	Mode	Median	Standard Deviation
large_airport	120.0075229	[121.75, 122.1]	121.85	19.28501632
medium_airport	154.2177948	[122.1]	124.955	91.85308547
small_airport	128.2513214	[135.475]	124.2125	28.38135591
Frequencies more than 100 mhz	149.2412527	[122.1]	124.4	68.09055123

Table 1: A table of statistics for frequencies of different categories of airport.

Reference

[1] “Pandas.dataframe¶,” pandas.DataFrame - pandas 1.4.2 documentation. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html. [Accessed: 01-May-2022].

[2] “How to use thread in Tkinter Python,” GeeksforGeeks, 17-Dec-2020. [Online]. Available: https://www.geeksforgeeks.org/how-to-use-thread-in-tkinter-python/. [Accessed: 01-May-2022].

[3] “Threading - thread-based parallelism¶,” threading - Thread-based parallelism - Python 3.10.4 documentation. [Online]. Available: https://docs.python.org/3/library/threading.html. [Accessed: 01-May-2022].

[4] “Tkinter - Python interface to TCL/TK¶,” tkinter - Python interface to Tcl/Tk - Python 3.10.4 documentation. [Online]. Available: https://docs.python.org/3/library/tkinter.html. [Accessed: 01-May-2022].

[5] M. Gorelick and I. Ozsvald, High performance python: Practical performant programming for humans. Sebastopol: O'Reilly, 2020.

[6] B. Slatkin, Effective python: 90 specific ways to write better python. USA: Addison-Wesley, 2020.

[7] A. Martelli, A. M. Ravenscroft, and D. Ascher, Python cookbook. Beijing: O'Reilly, 2005.

[8] “Page,” GlobalInterpreterLock - Python Wiki. [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock. [Accessed: 02-Jul-2022].

[9] A. Martelli, Python in a Nutshell. Sebastopol, CA: O'Reilly, 2006.

[10] Q. Nguyen, Advanced python programming: Accelerate your python programs using proven techniques and Design Patterns. Birmingham: Packt Publishing, Limited, 2022.

[11] Python Parallel Programming Cookbook. Erscheinungsort nicht ermittelbar: Packt Publishing Limited, 2015.

[12] E. Forbes, Learning concurrency in Python: Speed up your python code with clean, readable, and advanced concurrency techniques. Birmingham, UK: Packt Publishing, 2017.

[13] J. Palach, Parallel programming with python: Develop efficient parallel systems using the robust python environment. Birmingham: Packt Pub., 2014.

[14] Q. Nguyen, Mastering concurrency in python: Create faster programs using concurrency, asynchronous, multithreading, and Parallel Programming. Birmingham: Packt Publishing, 2018.

[15] G. Lanaro, Python high performance: Build robust applications by implementing concurrent and distributed processing techniques. Birmingham: Packt, 2017.

[16] B. P. BETTIG and R. P. HAN, “An object-oriented framework for interactive numerical analysis in a graphical user interface environment,” International Journal for Numerical Methods in Engineering, vol. 39, no. 17, pp. 2945–2971, 1996.

[17] J. E. Grayson, Python and Tkinter Programming: Graphical user interfaces for python programs. Greenwich, CT: Manning, 2000.

[18] “Pandas,” pandas. [Online]. Available: https://pandas.pydata.org/. [Accessed: 01-May-2022].

[19] “PANDAS.READ_CSV¶,” pandas.read_csv - pandas 1.4.1 documentation. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html. [Accessed: 01-May-2022].

[20] “Pandas.series¶,” pandas.Series - pandas 1.4.1 documentation. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.Series.html. [Accessed: 01-May-2022].

[21] “Pandas.dataframe.select_dtypes¶,” pandas.DataFrame.select_dtypes - pandas 1.4.1 documentation. [Online]. Available:

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.select_dtypes.html. [Accessed: 01-May-2022].

[22] “Pandas.dataframe.columns¶,” pandas.DataFrame.columns - pandas 1.4.1 documentation. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.columns.html. [Accessed: 01-May-2022].

[23] “Pandas.series.tolist¶,” pandas.Series.tolist - pandas 1.4.1 documentation. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.Series.tolist.html. [Accessed: 01-May-2022].

[24] “Pandas.dataframe.fillna¶,” pandas.DataFrame.fillna - pandas 1.4.1 documentation. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html. [Accessed: 01-May-2022].

[25] “Pandas.dataframe.median¶,” pandas.DataFrame.median - pandas 1.4.1 documentation. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.median.html. [Accessed: 01-May-2022].

[26] “Pandas.dataframe.replace¶,” pandas.DataFrame.replace - pandas 1.4.1 documentation. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.replace.html. [Accessed: 01-May-2022].

[27] “Spark overview,” Overview - Spark 3.2.1 Documentation. [Online]. Available: https://spark.apache.org/docs/latest/. [Accessed: 01-May-2022].

[28] R. Lafore, Data Structures and algorithms in Java. Indianapolis, IN: Sams, 2003.

[29] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Data structures and algorithms in Python. New Delhi: Wiley, 2018.

[30] S. Downes, L. Belliveau, S. Samet, M. A. Rahman and R. Savoie, "Managing Digital Rights Using JSON," 2010 7th IEEE Consumer Communications and Networking Conference, 2010, pp. 1-10, doi: 10.1109/CCNC.2010.5421662.

[31] NumPy. [Online]. Available: https://numpy.org/. [Accessed: 01-May-2022].

[32] A. Grinberg, XML and JSON recipes for SQL server: A problem-solution approach. Berkeley, CA: Apress, 2018.

[33] D. Gordon, I. Stavrakakis, J. P. Gibson, B. Tierney, A. Becevel, A. Curley, M. Collins, W. O’Mahony, and D. O’Sullivan, “Perspectives on Computing Ethics: A multi-stakeholder analysis,” Journal of Information, Communication and Ethics in Society, vol. 20, no. 1, pp. 72–90, 2021.

[34] H. Li, L. Yu, and W. He, “The impact of GDPR on Global Technology Development,” Journal of Global Information Technology Management, vol. 22, no. 1, pp. 1–6, 2019.

[35] Y. K. Dwivedi, G. Kelly, M. Janssen, N. P. Rana, E. L. Slade, and M. Clement, “Social Media: The good, the bad, and the ugly,” Information Systems Frontiers, vol. 20, no. 3, pp. 419–423, 2018.

[36] R. M. Baecker and E. B. Koffman, “Enriching courses on computers and society and Computer Ethics,” Proceedings of the 50th ACM Technical Symposium on Computer Science Education, 2019.

[37] J. Moore, “Towards a more representative politics in the ethics of Computer Science,” Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, 2020.

[38] J. C. Knight and N. G. Leveson, “Should software engineers be licensed?,” Communications of the ACM, vol. 45, no. 11, pp. 87–90, 2002.

[39] “Regulatory compliance and its impact on software development.” [Online]. Available: https://users.encs.concordia.ca/~abdelw/papers/ICOMPLY10.pdf. [Accessed: 30-Apr-2022].

[40] A. Hamou-Lhadj and A. Hamou-Lhadj, “Towards a compliance support framework for global software companies,” in Proc. of the 11th IASTED Int. Conf. on Software Engineering and Applications, USA, 2007, pp. 31–36.

[41] K. Healy, Data Visualization: A practical introduction. Princeton, NJ: Princeton university press, 2019.