**Report**

**1) In this project, I designed an event driving program that allows users to:**

* load the initial dataset (CSV) files and then translate them into JSON format, which are saved for backup files
* load and clean the initial dataset files, then save them in CSV format (They are called prepared dataset files)
* use the above prepared data to generate some outputs (calculating mean, mode and median for the inspection score per year for each type of vendor’s seating and for each ‘zip code’)
* generate a bar chart to display the number of establishments that have committed each type of violation
* calculate the correlation between the number of violations committed per vendor and their zip codes and visualize it with the scatter plot.

Throughout my project, pandas and numpy were used as the main tools to access, clean and analyse the given data set and generate outcomes per client requirements. Besides, matplotlib and seaborn were used for plotting and visualization. For creating GUI, I used Tkinter.

Graphical user interface, text, application, email

Description automatically generated

**Task 1:**

I chose to translate the given dataset to JSON format file (by clicking button “Translate to Json”). The Json files will have the same name as the original files with the (end) json. For doing that, I created an empty dictionary data {}, read each line in the initial csv file using csv.DictReader() function from csv library of Python, then put them in the data{}, and finally parse them to the json file.

Code samples:

*#Functions to load and convert csv file to json file*

def convert\_cvs\_to\_json(csvFilePath):

data = {}

df = pd.read\_csv(csvFilePath)

head = df.columns[0]

jsonFilePath = csvFilePath[:-4]+".json"

show\_output(jsonFilePath)

with open (csvFilePath, encoding='utf-8-sig') as csvFile:

csvReader = csv.DictReader(csvFile)

for csvRow in csvReader:

data[csvRow[head]] = csvRow

with open (jsonFilePath, 'w') as jsonFile:

jsonFile.write(json.dumps(data, indent = 4))

*# load csv file (for converting into a Json file):*

def UploadAction1(event = None):

filename = filedialog.askopenfilename()

print("Select:", filename)

show\_output(filename)

convert\_cvs\_to\_json(filename)

I chose json format to translate the data because it’s much more lightweight than the other format, so it increases the parsing speed and it can be easily sent over a network.

It has the simple, compact style like CSV, but supports hierarchical data like XML, and has the structure of name/value pair which is easy for human and machine to read.

It has the built-in module in Python (json), which can be easily import in Python programme.

**Task 2:** Implementation client’s 3rd requirement, which is produce mean, mode and median for the inspection score per year for each type of vendor’s seating and for each zip codes

Firstly, for cleaning, handling errors, missing value and managing inconsistences, I access the file by pandas (df = pd.read\_csv(“Filename.csv”)), then wrote a function to clean data and fulfil some other client’s requirement.

This function filtered out all the missing values, change all columns name to upper case, then remove unnecessary '#'character and remove white space from both sides of string in dataframe. Given the client’s requirement, I filtered out all data from vendors that have a ‘PROGRAM STATUS’ of INACTIVE.

Code sample:

def cleanfile(initialFileName):

df = pd.read\_csv(initialFileName)

df = df.dropna(how = 'all') *# filtering out missing data*

df.columns = df.columns.str.upper(). *# change all columns name to upper case*

for col in df.columns:

if (is\_string\_dtype(df[col])):

df[col]= df[col].str.strip('#') *# remove unneccessary '#’*

df[col]= df[col].str.strip() *# remove white space from both sides of string in DF*

if col == 'PROGRAM STATUS':

active = df['PROGRAM STATUS'] =='ACTIVE' *# Remove vendors with status of inactive*

df = df.loc[active]

To implement client’s 3rd requirement, which is produce mean, mode and median for the inspection score per year for each type of vendor’s seating and for each zip codes, I divided the task into some steps:

1. Accessing to the Inspection\_PREPARED file (which is the initial Inspection file that had cleaned and save as a csv file)
2. Creating a column named YEAR by converting string values in ‘ACTIVITY DATE’ column to date/time object in python, then creating a new column “YEAR”

Code sample:

df\_prepared['ACTIVITY DATE'] = pd.to\_datetime(df\_prepared['ACTIVITY DATE'])

df\_prepared['YEAR'] = df\_prepared['ACTIVITY DATE'].dt.year *# creating "YEAR" column*

1. Creating a column named SEAT:

* From PE DESCRIPTION column, extracting the type of vendor’s seating, e.g. (1-1,999 SF), into another column named SEAT. For that purpose, I wrote a function to split the string values in PE DESCRIPTION column by using function split(), and split them by parenthesis.
* Retaining the rest of the string in column PE DESCRIPTION.

Code sample:

*# Function for extracting the type of vendor’s seating of col PE DESCRIPTION into another col:*

def get\_seat (seat):

x = seat.split('(')

if len(x) > 1:  *# check for missing parenthesis*

return "(" + seat.split('(')[1].split(')')[0] + ")"

else:

return 'NaN'

*# Function for retaining the rest of col PE DESCRIPTION:*

def get\_seat2 (seat):

x = seat.split('(')

if len(x) > 1:

return re.sub("\(\d+\-\d+\)"," ",seat)

else:

return 'NaN'

*# Extract the type of seating of col PE DESCRIPTION into another column (column: SEAT):*

if col =='PE DESCRIPTION':

df['SEAT'] = df[col].apply(lambda x: get\_seat(x))

df[col] = df[col].apply(lambda x: get\_seat2(x)) *# Retain the rest*

1. Grouping inspection scores in column SCORE by data in col YEAR and column SEAT to group each score by year and type of vendor’s seating.

Code sample:

grouped2 = df\_prepared["SCORE"].groupby([df\_prepared['YEAR'],df\_prepared['SEAT']])

1. Calculating mean, mode, and median for the inspection score per year for each type of vendor’s seating

Code sample:

print(grouped2.mean())

print(grouped2.median())

print(grouped2.agg(pd.Series.mode).to\_frame())

1. Grouping inspection scores in column SCORE by data in column YEAR and column ZIP CODES to group each score by year in each zip codes

Code sample:

grouped1 = df\_prepared["SCORE"].groupby([df\_prepared['YEAR'],df\_prepared['ZIP CODES']])

1. Calculating mean, mode, and median for the inspection score per year for each zip code.

Code sample:

print(grouped1.mean())

print(grouped1.median())

print(grouped1.agg(pd.Series.mode).to\_frame())

**Task 3:**

Implementing the client’s 4th requirement, which is to produce a suitable graph that displays the number of establishments that have committed each type of violation:

1. Merging prepared Inspections file and prepared Violations files by column “SERIAL NUMBER”.

Code sample:

df\_new = pd.merge(df\_prepared, df\_prepared2, on = "SERIAL NUMBER")

1. Because there are so many types of violation (98 different violation codes), and the number of establishments violated those codes are in an extremely wide range, making it infeasible to produce a suitable graph, I decided to group violation codes into 6 groups, then count the number of violations in each group, as follows

* Group 1: Violation codes F001 – F009
* Group 2: Violation codes F010 – F019
* Group 3: Violation codes F020 – F029
* Group 4: Violation codes F030 – F039
* Group 5: Violation codes F040 – F049
* Group 6: Violation codes F050 – F059 + all other violation codes

Code sample:

df\_new['grouped code'] = df\_new['VIOLATION CODE'] *# creating a copy of column 'VIOLATION CODE')*

*# creating 6 groups of violation codes:*

df\_new['groupedcode'].replace(regex={r'^F00\d':'1',r'^F01\d':'2',r'^F02\d':'3',r'^F03\d':'4',r'^F04\d':'5',r'^F05\d':'6',r'^W\d+':'7',r'^MF\d+':'8',r'^H\d+':'9'}, inplace = True)

df\_new["grouped code"].replace(['7','8','9'],'6',inplace = True) # grouping 6,7,8,9 = 6

*# counting number of violations based on grouped code*

results = df\_new["grouped code"].value\_counts().sort\_index()

print(results)

*>>> The result of counting those value are:*

* Group 1: 89192 violations
* Group 2: 41814 violations
* Group 3: 81994 violations
* Group 4: 358544 violations
* Group 5: 231827 violations
* Group 6: 33631 violations

1. Producing a bar chart to displays the number of violations committed by establishments in each group of violation codes. The x-axis is the grouped violation codes, and the y-axis is the number of violations committed by establishments in each grouped code.

Code sample:

grouped\_Val\_Code = range(1,7)

plt.bar(grouped\_Val\_Code, results)

plt.xlabel('grouped types of violation')

plt.ylabel('number of violated establishment')

plt.show()

Chart, bar chart

Description automatically generated

**Task 4:**

Implementing the client’s 5th requirement, which is to determine if there is any significant correlation between the number of violations committed per vendor and their zip code.

1. As in the previous task, the prepared Inspections and prepared Violations files are merged by column “SERIAL NUMBER”.

Code sample:

df\_new = pd.merge(df\_prepared, df\_prepared2, on = "SERIAL NUMBER")

1. Then, compute count of violations in each facility in their zip code, using the groupby.count() construct of pandas

Code sample:

res = df\_new.groupby(['ZIP CODES','FACILITY ID']).count()

1. Sort the obtained dataframe above by the average number of violations per facility in each zip code (in ascending order)

Code sample:

*#group by Zip Codes*

grp = res.groupby('ZIP CODES')

*#Within each group, calculate the mean over 'OWNER ID' (or any other column #since they have the same values from the groupby.count()), and broadcast the values using transform. Then sort by 'OWNER ID'*

sort1 = res.loc[grp[['OWNER ID']].transform(lambda x: *#continued next line*

x.mean()).sort\_values('OWNER ID').index]

1. To calculate the correlation, first create a new column (called ‘x\_value’) in the dataframe that provide zip codes’ values (in ascending order of the average violations), then use the **.corr()** method. It is also useful to show the scatter plot of the data and the regression line

Code sample:

x\_value = range(1,len(sort1)+1)

sort1['x\_value'] = x\_value

sort1['x\_value'].corr(sort1['VIOLATION CODE']))

*#scatter plot with a regression line*

sns.regplot('x\_value','VIOLATION CODE',data = sort1, scatter\_kws={"color": "blue"}, line\_kws={"color": "red"})

plt.xlabel('zip codes (sorted by average number of violations)')

plt.ylabel('number of violations per vendor')

Correlation coefficient = 0.188804625542713

Chart, scatter chart

Description automatically generated

With a correlation coefficient of 0.189, we can conclude that there is a positive correlation between the number of violations committed per vendor and their zip code, but this correlation is rather insignificant. From the scatter plot it can be observed that the numbers of violations for different vendors are quite diverse (between 0 and below 200), and there are several numbers that are out of range (which can be as high as 1200). That explains the insignificance of the correlation.