

MSCI530 Python Programming for Problem Solving

**Assignment: Energy Call Centre Analysis**

Submitted By: Apoorva Dixit

Student Number: 36222321

1. **Introduction**

We are provided with Energy Call Centre data to analyse the effect of their Virtual Hold Technology last winter. The dataset includes data about the number of agents, calls offered, calls handled, calls abandoned, the average speed of answer, and average handle time at various times of the day when VHT is on or off. We originally have data for peak hours of the day and the population size is 504.

1. **Data Analysis**

The original dataset of size 504 is sampled and a sample size of 100 is taken where the starting point of the random number generator is 2322. Firstly, the two count plots are plotted to calculate the frequency of Time of Day (ToD) and VHT.

Chart, bar chart

Description automatically generated Chart, bar chart

Description automatically generated

The above count plots help to determine the amount of data available for each category. It can be easily concluded that the maximum available data is for afternoon peak hours and when VHT is ON. Then scatterplots are used to determine the relationship between the agents and the calls offered, calls handled, and calls abandoned respectively. This helps to determine how the change in the number of agents is affecting the calls handled, calls offered, and calls abandoned. After plotting these graphs using a scatter plot a positive linear correlation is shown between the number of agents and the number of calls offered and between the number of agents and the number of calls handled. There isn’t any correlation between the number of agents and the number of calls abandoned. This can also be justified by plotting the heatmap of the correlation matrix. In the scatterplot, VHT is used as a third variable to determine the spread based on whether VHT is On or Off. Since in the overall sample we have more data when VHT is On so in our scatter plots also we can see the number of points when VHT is On is more compared to when VHT is Off.

Chart, scatter chart

Description automatically generated Chart, scatter chart

Description automatically generated

Graphical user interface, application

Description automatically generated

Now the bar plots are plotted to identify the impact of VHT on the number of calls abandoned based on the time of the day. The bar plot for the number of calls offered is also plotted against VHT based on the time of the day to establish the true relationship between calls offered and calls abandoned. We can also calculate the percentage of calls abandoned with respect to calls offered when VHT is On or Off at the given time of the day. This helps to analyse the performance based on VHT at each peak hour.

|  |  |  |
| --- | --- | --- |
| Time of the day | Percentage of calls abandoned when VHT is On | Percentage of calls abandoned when VHT is Off |
| Morning | 3.8% | 0.66% |
| Afternoon | 2.0% | 1.33% |
| Evening | 1.8% | 4.2% |

\*The values in the table are approximate

Based on the above data derived from bar plots it is clearly visible that the VHT On is adversely affecting the number of calls abandoned during morning and afternoon. The impact of VHT On is having a positive impact during the evening when the number of abandoned calls is reduced.

Chart, bar chart

Description automatically generated Chart

Description automatically generated

From the below box plot, it is evident that the average handle time is minimum for morning peak hours out of all the peak hours when VHT is Off. But when VHT is On, the average handle time is maximum for morning peak hours out of all the peak hours. This can also be because we receive the maximum number of calls during the morning when VHT is On. Also, from the second box plot i.e. CallsHandled vs ToD, we can see that the maximum number of calls are handled during the morning whether VHT is On or Off. (Refer to the box plots below)

Chart, box and whisker chart

Description automatically generated

Chart, box and whisker chart

Description automatically generated

The heatmap also shows a strong correlation between the average speed of answers and calls abandoned. By plotting the scatter plot, we cannot visually conclude this, but it is quite evident from the bottom left of the plot that the lower the average speed of answer, lesser calls are abandoned.Chart, scatter chart

Description automatically generated

1. **Conclusion**

The above data analysis gives us some useful insights about using the VHT or not. The impact of VHT last winter is quite dependent on the time of the day and the number of incoming calls. In this sample, though we have the maximum amount of data for the afternoon, the number of calls offered, or incoming calls are maximum during the morning time. So, mornings are the busiest for the agents at the company. The least number of calls are received during the evening. Based on our analysis we can conclude that percentage of calls abandoned during the morning when VHT is On is more than the percentage of calls abandoned during the morning when VHT is Off. The same pattern is followed for afternoon peak hours as well. But it is evident from our calculation that for the evening percentage of calls abandoned when VHT is On is less than the percentage of calls abandoned when VHT is Off. So, VHT has a negative impact during the morning and afternoon when incoming calls are high, whereas VHT has a positive impact during the evening when incoming calls are relatively low. So, indirectly VHT’s impact is dependent on the number of calls offered. There also exists a correlation that a greater number of calls offered chances are more calls will be abandoned. So, VHT plays a vital role in determining this.

There is no such evidence that can prove that due to the use of VHT the chances of calls being abandoned can be reduced. This is because average handle time plays a vital role. When calls offered are more average handle time is more which in turn affects the number of calls being abandoned. The average speed of answers positively impacts the number of calls abandoned but based on data we don’t have enough information on whether VHT impacts the average speed of answers too.

So, using VHT to reduce the chance that callers abandon their calls depends on multiple factors such as the number of incoming calls, time of the day, and average handle time. We need to consider all of them while making a decision. In my sample, using VHT during evening hours is a good decision while using VHT during morning and afternoon hours is just a bad idea.

PYTHON CODE (For Reference)

**from tkinter import \***

**from tkinter import ttk, messagebox, filedialog**

**from matplotlib.figure import Figure**

**from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**import pickle**

**#StudentNumber-36222321; Random number generator = 2321+1; random\_state= 2322**

**#instantiating the tkinter root class**

**main\_root = Tk()**

**main\_root.title("Welcome")**

**#defining the class**

**class Data:**

**#constructor function of the class**

**def \_\_init\_\_(self):**

**pass**

**#function to upload file from local**

**def browse\_file(self):**

**'''function to upload file from local'''**

**try:**

**filename = filedialog.askopenfilename(filetypes=[("Excel files", "\*.xlsx")])**

**self.df = pd.read\_excel(filename)**

**self.df = self.df.sample(n=100,random\_state=2322)**

**messagebox.showinfo(title=None, message="File uploaded and sampling done!")**

**process\_label = Label(main\_root, text= "Press view data to proceed!", font=('Arial', 10), foreground="green")**

**process\_label.pack(padx=10, pady=10)**

**except:**

**print("Wrong file format!")**

**raise Exception**

**#function to display the dataframe created from the excel and perform various operations on it**

**def tree\_view(self):**

**'''function to display the dataframe created from the excel and perform various operations on it'''**

**try:**

**#defining new window to open on top of the original**

**self.root = Toplevel(main\_root)**

**self.root.geometry("1000x1000")**

**self.root.title("Energy Call Centre Analysis")**

**#add style**

**self.style = ttk.Style()**

**self.style.theme\_use("default")**

**self.style.configure("Treeview", background= "#D3D3D3", foreground = "black", rowheight = 25, fieldbackground="#D3D3D3")**

**self.style.map('Treeview', background=[('selected', 'blue')])**

**#define tree view for the data**

**self.tree\_data =ttk.Treeview(self.root)**

**self.df\_columns = list(self.df.columns)**

**#Define our columns**

**self.tree\_data['columns'] = (self.df\_columns[0], self.df\_columns[1], self.df\_columns[2], self.df\_columns[3], self.df\_columns[4], self.df\_columns[5], self.df\_columns[6], self.df\_columns[7], self.df\_columns[8])**

**#Format our columns**

**self.tree\_data.column("#0", width= 80, minwidth=5)**

**for col in self.df\_columns:**

**self.tree\_data.column(col, anchor=W, width=150)**

**#Create headings**

**self.tree\_data.heading("#0", text="Row number", anchor=W)**

**for col in self.df\_columns:**

**self.tree\_data.heading(col, text = col, anchor=W)**

**#Add data**

**global count**

**count = 1**

**for ind in self.df.index:**

**self.tree\_data.insert(parent='', index='end', text=count, values=(self.df[self.df\_columns[0]][ind], self.df[self.df\_columns[1]][ind], self.df[self.df\_columns[2]][ind], self.df[self.df\_columns[3]][ind], self.df[self.df\_columns[4]][ind], self.df[self.df\_columns[5]][ind], self.df[self.df\_columns[6]][ind], self.df[self.df\_columns[7]][ind], self.df[self.df\_columns[8]][ind]))**

**count += 1**

**self.tree\_data.pack(pady=20)**

**#frame for manipulation**

**manipulation\_frame = LabelFrame(self.root, text= 'Modify Data')**

**manipulation\_frame.pack(pady=20)**

**#labels for columns**

**Label(manipulation\_frame, text=self.df\_columns[0]).grid(row=0, column=0)**

**Label(manipulation\_frame, text=self.df\_columns[1]).grid(row=0, column=1)**

**Label(manipulation\_frame, text=self.df\_columns[2]).grid(row=0, column=2)**

**Label(manipulation\_frame, text=self.df\_columns[3]).grid(row=0, column=3)**

**Label(manipulation\_frame, text=self.df\_columns[4]).grid(row=0, column=4)**

**Label(manipulation\_frame, text=self.df\_columns[5]).grid(row=0, column=5)**

**Label(manipulation\_frame, text=self.df\_columns[6]).grid(row=0, column=6)**

**Label(manipulation\_frame, text=self.df\_columns[7]).grid(row=0, column=7)**

**Label(manipulation\_frame, text=self.df\_columns[8]).grid(row=0, column=8)**

**#input boxes for data**

**self.month\_box = Entry(manipulation\_frame)**

**self.vht\_box = Entry(manipulation\_frame)**

**self.tod\_box = Entry(manipulation\_frame)**

**self.agents\_box = Entry(manipulation\_frame)**

**self.callsoff\_box = Entry(manipulation\_frame)**

**self.callsaban\_box = Entry(manipulation\_frame)**

**self.callshand\_box = Entry(manipulation\_frame)**

**self.asa\_box = Entry(manipulation\_frame)**

**self.avghandtime\_box = Entry(manipulation\_frame)**

**self.month\_box.grid(row=1, column=0)**

**self.vht\_box.grid(row=1, column=1)**

**self.tod\_box.grid(row=1, column=2)**

**self.agents\_box.grid(row=1, column=3)**

**self.callsoff\_box.grid(row=1, column=4)**

**self.callsaban\_box.grid(row=1, column=5)**

**self.callshand\_box.grid(row=1, column=6)**

**self.asa\_box.grid(row=1, column=7)**

**self.avghandtime\_box.grid(row=1, column=8)**

**#buttons to perform manipulation in data frame**

**add\_record = Button(self.root, text='Add Record', command=lambda: self.add\_record())**

**add\_record.pack(padx= 10, pady=10)**

**remove\_record = Button(self.root, text='Delete Record', command=lambda: self.remove\_record())**

**remove\_record.pack(padx= 10, pady=10)**

**select\_record = Button(self.root, text='Select Record', command=lambda: self.select\_record())**

**select\_record.pack(padx= 10, pady=10)**

**update\_record = Button(self.root, text='Update Record', command=lambda: self.update\_record())**

**update\_record.pack(padx= 10, pady=10)**

**#menu for fetching data based on conditions**

**self.options = [**

**self.df\_columns[0],**

**self.df\_columns[1],**

**self.df\_columns[2],**

**self.df\_columns[3],**

**self.df\_columns[4],**

**self.df\_columns[5],**

**self.df\_columns[6],**

**self.df\_columns[7],**

**self.df\_columns[8]**

**]**

**#defining combobox to fetch data based on conditions**

**self.selectOptions = ttk.Combobox(self.root, values=self.options)**

**self.selectOptions.current(0)**

**self.selectOptions.bind("<<ComboboxSelected>>", self.selectOptions\_click)**

**self.selectOptions.pack(padx= 20, pady=20)**

**#button to visualise the analysis**

**plot = Button(self.root, text='Plot Graphs', command=lambda: self.plot\_graph())**

**plot.pack(padx= 20, pady=20)**

**except:**

**print("An exception occurred")**

**#function to redirect to a different window with different graph options**

**def plot\_graph(self):**

**'''function to redirect to a different window with different graph options'''**

**try:**

**#defining new window to open on top of the original**

**self.plot\_window = Toplevel(self.root)**

**self.plot\_window.geometry("1000x1000")**

**self.plot\_window.title("Visualizations")**

**#defining widgets for the Visualisation window**

**vis\_label = Label(self.plot\_window, text= "Graphical Analysis", font=('Arial', 20))**

**vis\_label.pack(padx=20, pady=20)**

**countplot\_frame = LabelFrame(self.plot\_window, text= 'Countplot options', width=300, height=500)**

**countplot\_frame.pack(padx=20, pady=20)**

**btn\_VHT = Button(countplot\_frame, text="Countplot for VHT", command=lambda:self.countplot\_VHT())**

**btn\_VHT.pack(padx=20, pady=20)**

**btn\_ToD = Button(countplot\_frame, text="Countplot for ToD", command=lambda:self.countplot\_ToD())**

**btn\_ToD.pack(padx=20, pady=20)**

**scatterplot\_frame = LabelFrame(self.plot\_window, text= 'Scatterplot options relative to Agents variable', width=300, height=500)**

**scatterplot\_frame.pack(padx=20, pady=20)**

**plot\_options = [**

**"CallsOffered",**

**"CallsHandled",**

**"CallsAbandoned"**

**]**

**self.select\_plotOptions = ttk.Combobox(scatterplot\_frame, values=plot\_options)**

**self.select\_plotOptions.current(0)**

**self.select\_plotOptions.bind("<<ComboboxSelected>>", self.scatterplot\_Agents)**

**self.select\_plotOptions.pack(padx= 20, pady=20)**

**btn\_scatter = Button(scatterplot\_frame, text="Scatterplot for AvgHandle time vs CallsHandled", command=lambda:self.scatter\_plot())**

**btn\_scatter.pack(padx=20, pady=20)**

**boxplot\_frame = LabelFrame(self.plot\_window, text= 'Boxplot options relative to ToD variable', width=300, height=500)**

**boxplot\_frame.pack(padx=20, pady=20)**

**box\_options = [**

**"CallsOffered",**

**"CallsHandled",**

**"CallsAbandoned",**

**"Avehandletime",**

**"ASA"**

**]**

**self.select\_boxOptions = ttk.Combobox(boxplot\_frame, values=box\_options)**

**self.select\_boxOptions.current(0)**

**self.select\_boxOptions.bind("<<ComboboxSelected>>", self.boxplot\_ToD)**

**self.select\_boxOptions.pack(padx= 20, pady=20)**

**barplot\_frame = LabelFrame(self.plot\_window, text= 'Barplot options relative to VHT variable', width=300, height=500)**

**barplot\_frame.pack(padx=20, pady=20)**

**bar\_options = [**

**"CallsOffered",**

**"CallsHandled",**

**"CallsAbandoned",**

**]**

**self.select\_barOptions = ttk.Combobox(barplot\_frame, values=bar\_options)**

**self.select\_barOptions.current(0)**

**self.select\_barOptions.bind("<<ComboboxSelected>>", self.barplot\_VHT)**

**self.select\_barOptions.pack(padx= 20, pady=20)**

**mainplot\_frame = LabelFrame(self.plot\_window, text= 'Plot to find correlation of all the variables with each other', width=300, height=500)**

**mainplot\_frame.pack(padx=20, pady=20)**

**btn\_Heatmap = Button(mainplot\_frame, text="HeatMap", command=lambda:self.heatmap())**

**btn\_Heatmap.pack(padx=20, pady=20)**

**except:**

**print("An exception occurred")**

**#function to add a new record**

**def add\_record(self):**

**'''function to add a new record'''**

**try:**

**global count**

**#check to validate empty fields**

**if self.month\_box.get() == "" or self.vht\_box.get()=="" or self.tod\_box.get()=="" or self.agents\_box.get()=="" or self.callsoff\_box.get() == "" or self.callsaban\_box. get()== "" or self.callshand\_box.get()=="" or self.asa\_box.get()== "" or self.avghandtime\_box.get()=="":**

**messagebox.showerror(title=None, message="Empty fields!")**

**else:**

**self.tree\_data.insert(parent='', index='end', text=count, values=(self.month\_box.get(), self.vht\_box.get(), self.tod\_box.get(), self.agents\_box.get(), self.callsoff\_box.get(), self.callsaban\_box.get(), self.callshand\_box.get(), self.asa\_box.get(), self.avghandtime\_box.get()))**

**count +=1**

**#Clear the input boxes**

**self.month\_box.delete(0, END)**

**self.vht\_box.delete(0, END)**

**self.tod\_box.delete(0, END)**

**self.agents\_box.delete(0, END)**

**self.callsoff\_box.delete(0, END)**

**self.callsaban\_box.delete(0, END)**

**self.callshand\_box.delete(0, END)**

**self.asa\_box.delete(0, END)**

**self.avghandtime\_box.delete(0, END)**

**messagebox.showinfo(title=None, message="Record added successfully!")**

**except:**

**print("An exception occurred")**

**#function to delete selected records**

**def remove\_record(self):**

**'''function to delete selected records'''**

**try:**

**selected\_records = self.tree\_data.selection()**

**for record in selected\_records:**

**self.tree\_data.delete(record)**

**messagebox.showinfo(title=None, message="Record deleted successfully!")**

**except:**

**print("An exception occurred")**

**#function to select record**

**def select\_record(self):**

**'''function to select record'''**

**try:**

**#Clear the input boxes**

**self.month\_box.delete(0, END)**

**self.vht\_box.delete(0, END)**

**self.tod\_box.delete(0, END)**

**self.agents\_box.delete(0, END)**

**self.callsoff\_box.delete(0, END)**

**self.callsaban\_box.delete(0, END)**

**self.callshand\_box.delete(0, END)**

**self.asa\_box.delete(0, END)**

**self.avghandtime\_box.delete(0, END)**

**#selecting the data**

**selected = self.tree\_data.focus()**

**values = self.tree\_data.item(selected, 'values')**

**#putting selected data to input boxes respectively**

**self.month\_box.insert(0, values[0])**

**self.vht\_box.insert(0, values[1])**

**self.tod\_box.insert(0, values[2])**

**self.agents\_box.insert(0, values[3])**

**self.callsoff\_box.insert(0, values[4])**

**self.callsaban\_box.insert(0, values[5])**

**self.callshand\_box.insert(0, values[6])**

**self.asa\_box.insert(0, values[7])**

**self.avghandtime\_box.insert(0, values[8])**

**except:**

**print("An exception occurred")**

**#function to update selected record**

**def update\_record(self):**

**'''function to update selected record'''**

**try:**

**#updating the selected data**

**selected = self.tree\_data.focus()**

**self.tree\_data.item(selected, values=(self.month\_box.get(), self.vht\_box.get(), self.tod\_box.get(), self.agents\_box.get(), self.callsoff\_box.get(), self.callsaban\_box.get(), self.callshand\_box.get(), self.asa\_box.get(), self.avghandtime\_box.get()))**

**messagebox.showinfo(title=None, message="Record updated successfully!")**

**except:**

**print("An exception occurred")**

**#function to select menu for fetching data**

**def selectOptions\_click(self, event):**

**'''function to select menu for fetching data'''**

**try:**

**if self.selectOptions.get() == self.df\_columns[0]:**

**#defining new window to open on top of the original**

**window = Toplevel(self.root)**

**window.geometry("400x400")**

**#defining widget for the new window**

**frame = LabelFrame(window, text= 'Options for the month', width=200, height=300)**

**frame.pack(pady=20)**

**self.selectMonth = ttk.Combobox(frame, values=["Oct-Nov","Dec-Jan","Feb-Mar"])**

**self.selectMonth.current(0)**

**self.selectMonth.bind("<<ComboboxSelected>>", self.search\_month)**

**self.selectMonth.pack(pady=20)**

**if self.selectOptions.get() == self.df\_columns[1]:**

**#defining new window to open on top of the original**

**window = Toplevel(self.root)**

**window.geometry("400x400")**

**frame = LabelFrame(window, text= 'Options for the VHT', width=200, height=300)**

**frame.pack(pady=20)**

**self.selectVHT = ttk.Combobox(frame, values=["On","Off"])**

**self.selectVHT.current(0)**

**self.selectVHT.bind("<<ComboboxSelected>>", self.search\_VHT)**

**self.selectVHT.pack(pady=20)**

**if self.selectOptions.get() == self.df\_columns[2]:**

**#defining new window to open on top of the original**

**window = Toplevel(self.root)**

**window.geometry("400x400")**

**frame = LabelFrame(window, text= 'Options for time of the day', width=200, height=300)**

**frame.pack(pady=20)**

**self.selectToD = ttk.Combobox(frame, values=["morning","afternoon","evening"])**

**self.selectToD.current(0)**

**self.selectToD.bind("<<ComboboxSelected>>", self.search\_ToD)**

**self.selectToD.pack(pady=20)**

**if self.selectOptions.get() == self.df\_columns[3]:**

**#defining new window to open on top of the original**

**window = Toplevel(self.root)**

**window.geometry("400x400")**

**frame = LabelFrame(window, text= 'Range for agents', width=200, height=100)**

**frame.pack(pady=20)**

**self.min\_label = Label(frame, text="Min")**

**self.min\_label.pack(pady=10)**

**self.Min\_Val = Entry(frame, width=20)**

**self.Min\_Val.pack(pady=5)**

**self.max\_label = Label(frame, text="Max")**

**self.max\_label.pack(pady=10)**

**self.Max\_Val = Entry(frame, width=20)**

**self.Max\_Val.pack(pady=5)**

**self.submit\_agent = Button(window, text = "Submit", command= lambda: self.search\_Agents())**

**self.submit\_agent.pack(pady=10)**

**if self.selectOptions.get() == self.df\_columns[4]:**

**#defining new window to open on top of the original**

**window = Toplevel(self.root)**

**window.geometry("400x400")**

**frame = LabelFrame(window, text= 'Range for Calls Offered', width=200, height=100)**

**frame.pack(pady=20)**

**self.min\_label = Label(frame, text="Min")**

**self.min\_label.pack(pady=10)**

**self.Min\_Val = Entry(frame, width=20)**

**self.Min\_Val.pack(pady=5)**

**self.max\_label = Label(frame, text="Max")**

**self.max\_label.pack(pady=10)**

**self.Max\_Val = Entry(frame, width=20)**

**self.Max\_Val.pack(pady=5)**

**self.submit\_agent = Button(window, text = "Submit", command= lambda: self.search\_CallsOffered())**

**self.submit\_agent.pack(pady=10)**

**if self.selectOptions.get() == self.df\_columns[5]:**

**#defining new window to open on top of the original**

**window = Toplevel(self.root)**

**window.geometry("400x400")**

**frame = LabelFrame(window, text= 'Range for Calls Abandoned', width=200, height=100)**

**frame.pack(pady=20)**

**self.min\_label = Label(frame, text="Min")**

**self.min\_label.pack(pady=10)**

**self.Min\_Val = Entry(frame, width=20)**

**self.Min\_Val.pack(pady=5)**

**self.max\_label = Label(frame, text="Max")**

**self.max\_label.pack(pady=10)**

**self.Max\_Val = Entry(frame, width=20)**

**self.Max\_Val.pack(pady=5)**

**self.submit\_agent = Button(window, text = "Submit", command= lambda: self.search\_CallsAbandoned())**

**self.submit\_agent.pack(pady=10)**

**if self.selectOptions.get() == self.df\_columns[6]:**

**#defining new window to open on top of the original**

**window = Toplevel(self.root)**

**window.geometry("400x400")**

**frame = LabelFrame(window, text= 'Range for Calls Handled', width=200, height=100)**

**frame.pack(pady=20)**

**self.min\_label = Label(frame, text="Min")**

**self.min\_label.pack(pady=10)**

**self.Min\_Val = Entry(frame, width=20)**

**self.Min\_Val.pack(pady=5)**

**self.max\_label = Label(frame, text="Max")**

**self.max\_label.pack(pady=10)**

**self.Max\_Val = Entry(frame, width=20)**

**self.Max\_Val.pack(pady=5)**

**self.submit\_agent = Button(window, text = "Submit", command= lambda: self.search\_CallsHandled())**

**self.submit\_agent.pack(pady=10)**

**if self.selectOptions.get() == self.df\_columns[7]:**

**#defining new window to open on top of the original**

**window = Toplevel(self.root)**

**window.geometry("400x400")**

**frame = LabelFrame(window, text= 'Range for average speed of answer', width=200, height=100)**

**frame.pack(pady=20)**

**self.min\_label = Label(frame, text="Min")**

**self.min\_label.pack(pady=10)**

**self.Min\_Val = Entry(frame, width=20)**

**self.Min\_Val.pack(pady=5)**

**self.max\_label = Label(frame, text="Max")**

**self.max\_label.pack(pady=10)**

**self.Max\_Val = Entry(frame, width=20)**

**self.Max\_Val.pack(pady=5)**

**self.submit\_agent = Button(window, text = "Submit", command= lambda: self.search\_ASA())**

**self.submit\_agent.pack(pady=10)**

**if self.selectOptions.get() == self.df\_columns[8]:**

**#defining new window to open on top of the original**

**window = Toplevel(self.root)**

**window.geometry("400x400")**

**frame = LabelFrame(window, text= 'Range for average handle time', width=200, height=100)**

**frame.pack(pady=20)**

**self.min\_label = Label(frame, text="Min")**

**self.min\_label.pack(pady=10)**

**self.Min\_Val = Entry(frame, width=20)**

**self.Min\_Val.pack(pady=5)**

**self.max\_label = Label(frame, text="Max")**

**self.max\_label.pack(pady=10)**

**self.Max\_Val = Entry(frame, width=20)**

**self.Max\_Val.pack(pady=5)**

**self.submit\_agent = Button(window, text = "Submit", command= lambda: self.search\_AvgHandleTime())**

**self.submit\_agent.pack(pady=10)**

**except:**

**print("An exception occurred")**

**#function to display data based on month**

**def search\_month(self,event):**

**'''function to display data based on month'''**

**try:**

**count=1**

**self.tree\_data.delete(\*self.tree\_data.get\_children())**

**for index, row in self.df.loc[self.df["Month"].eq(self.selectMonth.get())].iterrows():**

**self.tree\_data.insert("", "end", text =count, values =list(row))**

**count+=1**

**except:**

**print("An exception occurred")**

**#function to display data based on VHT**

**def search\_VHT(self,event):**

**'''function to display data based on VHT'''**

**try:**

**count=1**

**self.tree\_data.delete(\*self.tree\_data.get\_children())**

**for index, row in self.df.loc[self.df["VHT"].eq(self.selectVHT.get())].iterrows():**

**self.tree\_data.insert("", "end", text =count, values =list(row))**

**count+=1**

**except:**

**print("An exception occurred")**

**#function to display data based on ToD**

**def search\_ToD(self, event):**

**'''function to display data based on ToD'''**

**try:**

**count=1**

**self.tree\_data.delete(\*self.tree\_data.get\_children())**

**for index, row in self.df.loc[self.df["ToD"].eq(self.selectToD.get())].iterrows():**

**self.tree\_data.insert("", "end", text =count, values =list(row))**

**count+=1**

**except:**

**print("An exception occurred")**

**#function to display data based on agent range**

**def search\_Agents(self):**

**'''function to display data based on agent range'''**

**try:**

**count=1**

**self.tree\_data.delete(\*self.tree\_data.get\_children())**

**for index, row in self.df.loc[self.df["Agents"].between(int(self.Min\_Val.get()), int(self.Max\_Val.get()))].iterrows():**

**self.tree\_data.insert("", "end", text =count, values =list(row))**

**count+=1**

**except:**

**print("An exception occurred")**

**#function to display data based on Calls offered range**

**def search\_CallsOffered(self):**

**'''function to display data based on Calls offered range'''**

**try:**

**count=1**

**self.tree\_data.delete(\*self.tree\_data.get\_children())**

**for index, row in self.df.loc[self.df["CallsOffered"].between(int(self.Min\_Val.get()), int(self.Max\_Val.get()))].iterrows():**

**self.tree\_data.insert("", "end", text =count, values =list(row))**

**count+=1**

**except:**

**print("An exception occurred")**

**#function to display data based on Calls abandoned range**

**def search\_CallsAbandoned(self):**

**'''function to display data based on Calls abandoned range'''**

**try:**

**count=1**

**self.tree\_data.delete(\*self.tree\_data.get\_children())**

**for index, row in self.df.loc[self.df["CallsAbandoned"].between(int(self.Min\_Val.get()), int(self.Max\_Val.get()))].iterrows():**

**self.tree\_data.insert("", "end", text =count, values =list(row))**

**count+=1**

**except:**

**print("An exception occurred")**

**#function to display data based on Calls handled range**

**def search\_CallsHandled(self):**

**'''function to display data based on Calls handled range'''**

**try:**

**count=1**

**self.tree\_data.delete(\*self.tree\_data.get\_children())**

**for index, row in self.df.loc[self.df["CallsHandled"].between(int(self.Min\_Val.get()), int(self.Max\_Val.get()))].iterrows():**

**self.tree\_data.insert("", "end", text =count, values =list(row))**

**count+=1**

**except:**

**print("An exception occurred")**

**#function to display data based on average speed of answer range**

**def search\_ASA(self):**

**'''function to display data based on average speed of answer range'''**

**try:**

**count=1**

**self.tree\_data.delete(\*self.tree\_data.get\_children())**

**for index, row in self.df.loc[self.df["ASA"].between(float(self.Min\_Val.get()), float(self.Max\_Val.get()))].iterrows():**

**self.tree\_data.insert("", "end", text =count, values =list(row))**

**count += 1**

**except:**

**print("An exception occurred")**

**#function to display data based on average handle time range**

**def search\_AvgHandleTime(self):**

**'''function to display data based on average handle time range'''**

**try:**

**count=1**

**self.tree\_data.delete(\*self.tree\_data.get\_children())**

**for index, row in self.df.loc[self.df["Avehandletime"].between(float(self.Min\_Val.get()), float(self.Max\_Val.get()))].iterrows():**

**self.tree\_data.insert("", "end", text =count, values =list(row))**

**count += 1**

**except:**

**print("An exception occurred")**

**#function to plot the countplot of VHT**

**def countplot\_VHT(self):**

**'''function to plot the countplot of VHT'''**

**try:**

**plot\_root = Toplevel(self.plot\_window)**

**plot\_root.geometry("1000x1000")**

**frame = LabelFrame(plot\_root, text= 'Countplot', width=500, height=500)**

**frame.pack(pady=20)**

**fig = Figure(figsize=(70,70), dpi=100)**

**a\_subplot = fig.add\_subplot(111)**

**sns.countplot(x=self.df['VHT'],data=self.df, ax= a\_subplot)**

**canvas = FigureCanvasTkAgg(fig, frame)**

**canvas.get\_tk\_widget().pack()**

**canvas.draw()**

**except:**

**print("An exception occurred")**

**#function to plot the countplot of ToD**

**def countplot\_ToD(self):**

**'''function to plot the countplot of ToD'''**

**try:**

**plot\_root = Toplevel(self.plot\_window)**

**plot\_root.geometry("1000x1000")**

**frame = LabelFrame(plot\_root, text= 'Countplot', width=500, height=500)**

**frame.pack(pady=20)**

**fig = Figure(figsize=(70,70), dpi=100)**

**a\_subplot = fig.add\_subplot(111)**

**sns.countplot(x=self.df['ToD'],data=self.df, ax= a\_subplot)**

**canvas = FigureCanvasTkAgg(fig, frame)**

**canvas.get\_tk\_widget().pack()**

**canvas.draw()**

**except:**

**print("An exception occurred")**

**#function to plot scatterplot relative to Agents**

**def scatterplot\_Agents(self, event):**

**'''function to plot scatterplot relative to Agents'''**

**try:**

**plot\_root = Toplevel(self.plot\_window)**

**plot\_root.geometry("1500x1500")**

**frame = LabelFrame(plot\_root, text= 'Scatterplot', width=500, height=500)**

**frame.pack(pady=20)**

**fig = Figure(figsize=(100,100), dpi=100)**

**a\_subplot = fig.add\_subplot(111)**

**if self.select\_plotOptions.get() == "CallsOffered":**

**sns.scatterplot(data=self.df, x=self.df['Agents'], y=self.df['CallsOffered'], hue=self.df['VHT'], ax=a\_subplot)**

**if self.select\_plotOptions.get() == "CallsAbandoned":**

**sns.scatterplot(data=self.df, x=self.df['Agents'], y=self.df['CallsAbandoned'], hue=self.df['VHT'], ax=a\_subplot)**

**if self.select\_plotOptions.get() == "CallsHandled":**

**sns.scatterplot(data=self.df, x=self.df['Agents'], y=self.df['CallsHandled'], hue=self.df['VHT'], ax=a\_subplot)**

**canvas = FigureCanvasTkAgg(fig, frame)**

**canvas.get\_tk\_widget().pack()**

**canvas.draw()**

**except:**

**print("An exception occurred")**

**#function to plot scatterplot between Avghandletime and CallsHandled**

**def scatter\_plot(self):**

**'''function to plot scatterplot between Avghandletime and CallsHandled'''**

**try:**

**plot\_root = Toplevel(self.plot\_window)**

**plot\_root.geometry("1500x1500")**

**frame = LabelFrame(plot\_root, text= 'Scatterplot', width=500, height=500)**

**frame.pack(pady=20)**

**fig = Figure(figsize=(100,100), dpi=100)**

**a\_subplot = fig.add\_subplot(111)**

**sns.scatterplot(data=self.df, x=self.df['Avehandletime'], y=self.df['CallsHandled'], hue=self.df['VHT'], ax=a\_subplot)**

**canvas = FigureCanvasTkAgg(fig, frame)**

**canvas.get\_tk\_widget().pack()**

**canvas.draw()**

**except:**

**print("An exception occurred")**

**#function to plot boxplot relative to ToD**

**def boxplot\_ToD(self, event):**

**'''function to plot boxplot relative to ToD'''**

**try:**

**plot\_root = Toplevel(self.plot\_window)**

**plot\_root.geometry("1500x1500")**

**frame = LabelFrame(plot\_root, text= 'Boxplot', width=500, height=500)**

**frame.pack(pady=20)**

**fig = Figure(figsize=(100,100), dpi=100)**

**a\_subplot = fig.add\_subplot(111)**

**if self.select\_boxOptions.get() == "CallsOffered":**

**sns.boxplot(x=self.df['ToD'], y=self.df['CallsOffered'],data=self.df, palette='rainbow', hue=self.df['VHT'], ax=a\_subplot)**

**if self.select\_boxOptions.get() == "CallsHandled":**

**sns.boxplot(x=self.df['ToD'], y=self.df['CallsHandled'],data=self.df, palette='rainbow', hue=self.df['VHT'], ax=a\_subplot)**

**if self.select\_boxOptions.get() == "CallsAbandoned":**

**sns.boxplot(x=self.df['ToD'], y=self.df['CallsAbandoned'],data=self.df, palette='rainbow', hue=self.df['VHT'], ax=a\_subplot)**

**if self.select\_boxOptions.get() == "Avehandletime":**

**sns.boxplot(x=self.df['ToD'], y=self.df['Avehandletime'],data=self.df, palette='rainbow', hue=self.df['VHT'], ax=a\_subplot)**

**if self.select\_boxOptions.get() == "ASA":**

**sns.boxplot(x=self.df['ToD'], y=self.df['ASA'],data=self.df, palette='rainbow', hue=self.df['VHT'], ax=a\_subplot)**

**canvas = FigureCanvasTkAgg(fig, frame)**

**canvas.get\_tk\_widget().pack()**

**canvas.draw()**

**except:**

**print("An exception occurred")**

**#function to plot the heatmap**

**def heatmap(self):**

**'''function to plot the heatmap'''**

**try:**

**plot\_root = Toplevel(self.plot\_window)**

**plot\_root.geometry("1500x1500")**

**frame = LabelFrame(plot\_root, text= 'Heatmap', width=500, height=500)**

**frame.pack(pady=20)**

**fig = Figure(figsize=(100,100), dpi=100)**

**a\_subplot = fig.add\_subplot(111)**

**corr\_var=self.df.corr()**

**sns.heatmap(corr\_var, square=True, cbar=False, ax=a\_subplot, annot= True)**

**canvas = FigureCanvasTkAgg(fig, frame)**

**canvas.get\_tk\_widget().pack()**

**canvas.draw()**

**except:**

**print("An exception occurred")**

**#function to plot the baroplot relative to VHT**

**def barplot\_VHT(self, event):**

**'''function to plot the baroplot relative to VHT'''**

**try:**

**plot\_root = Toplevel(self.plot\_window)**

**plot\_root.geometry("1000x1000")**

**frame = LabelFrame(plot\_root, text= 'Barplot', width=500, height=500)**

**frame.pack(pady=20)**

**fig = Figure(figsize=(100,100), dpi=100)**

**a\_subplot = fig.add\_subplot(111)**

**if self.select\_barOptions.get() == "CallsOffered":**

**sns.barplot(data=self.df, x=self.df['VHT'], y=self.df['CallsOffered'], hue=self.df['ToD'], ax=a\_subplot)**

**if self.select\_barOptions.get() == "CallsAbandoned":**

**sns.barplot(data=self.df, x=self.df['VHT'], y=self.df['CallsAbandoned'], hue=self.df['ToD'], ax=a\_subplot)**

**if self.select\_barOptions.get() == "CallsHandled":**

**sns.barplot(data=self.df, x=self.df['VHT'], y=self.df['CallsHandled'], hue=self.df['ToD'], ax=a\_subplot)**

**canvas = FigureCanvasTkAgg(fig, frame)**

**canvas.get\_tk\_widget().pack()**

**canvas.draw()**

**except:**

**print("An exception occurred")**

**#instantiating the class**

**view = Data()**

**#checking if there already exists pickle file**

**try :**

**file = open("record.ms","rb")**

**prev\_run = pickle.load(file)**

**print("Previous analysis", prev\_run)**

**file.close()**

**except FileNotFoundError :**

**#file doesn't exist**

**print("A new file is created")**

**#new file is created to store the current state in serialized form**

**file = open("record.ms","wb")**

**pickle.dump(view,file)**

**file.close()**

**#defining the widgets of the main window here**

**welcome\_label = Label(main\_root, text= "Welcome User!", font=('Arial', 20))**

**welcome\_label.pack(padx=20, pady=20)**

**upload\_file = Button(main\_root, text="Upload file", command=lambda: view.browse\_file())**

**upload\_file.pack(padx=10, pady=10)**

**view\_data = Button(main\_root, text = "View Data", command= lambda: view.tree\_view())**

**view\_data.pack(padx=10, pady=10)**

**main\_root.geometry("600x600")**

**main\_root.mainloop()**