```
from tkinter import *
from tkinter import messagebox
import re,pymysql
from tkinter import Tk
from PIL import ImageTk, Image
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.dates as mdates
from seaborn import regplot
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from openpyxl import *
from tkinter import *
from tkinter import messagebox
import re , pymysql
from tkinter import Tk
from PIL import ImageTk, Image
from matplotlib.pyplot import title
from matplotlib.pyplot import subplots
from matplotlib.pyplot import subplot
from numpy import array
from numpy import mean
from numpy import nan
from pandas import read_csv
from pandas import to_numeric
from pandas import notnull
from fractions import Fraction
from matplotlib.pyplot import bar
from matplotlib.pyplot import ylim
from matplotlib.pyplot import xlabel
from matplotlib.pyplot import ylabel
from matplotlib.pyplot import title
from matplotlib.pyplot import xticks
from matplotlib.pyplot import subplots
from matplotlib.pyplot import subplot
from matplotlib.pyplot import show
from matplotlib.pyplot import scatter
from matplotlib.pyplot import plot
from matplotlib.pyplot import tight_layout
from seaborn import barplot
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from tkinter import BOTH
from tkinter import BOTTOM
from tkinter import TOP
from tkinter import Text
from tkinter import END
from tkinter import Label
from tkinter import Scrollbar
from tkinter import messagebox
from tkinter import W
from tkinter import Entry
from csv import writer
from PIL import ImageTk, Image
import plotly.express as px
import plotly.graph_objects as go
from plotly.offline import plot
import numpy as np

data_priya=pd.read_csv("Internship-project-Dataset2.csv")
data_priya = data_priya[notnull(data_priya['Translated_Review'])]
data_priya = data_priya[notnull(data_priya['Sentiment'])]
data_priya= data_priya[notnull(data_priya['Sentiment_Polarity'])]
data_priya= data_priya[notnull(data_priya['Sentiment_Subjectivity'])]
```

```
data_rohan=pd.read_csv("sentiment syrus 1 - Sheet1.csv")
data_arnav=pd.read_csv("sentiment syrus 2 - Sheet1.csv",encoding='latin1')
data_naren=pd.read_csv("bird1.csv")
data_safety=pd.read_csv("Aviation safety.csv")
data_aviate=pd.read_csv("AviationData.csv",encoding='latin1')
data_aviate.dropna(axis=1)
data_arnav.dropna(axis=1)
data_naren.dropna(axis=1)
data_naren1=pd.read_csv("temanipulation - Sheet1 (1).csv")
data_arnav['Sentiment_Polarity'] = pd.to_numeric(data_arnav['Sentiment_Polarity'],
errors='coerce')
data_arnav['Sentiment_Subjectivity'] =
pd.to_numeric(data_arnav['Sentiment_Subjectivity'], errors='coerce')
data_naren['FlightDate'] = pd.to_datetime(data_naren['FlightDate'])
data_adithya=pd.read_csv("strikenew.csv")
# This function is used for adjusting window size and making the necessary
configuration on start of window
def adjustWindow(window):
    w = 800  # width for the window size
    h = 600  # height for the window size
    ws = screen.winfo_screenwidth()  # width of the screen
    hs = screen.winfo_screenheight()  # height of the screen
    x = (ws/2) - (w/2)  # calculate x and y coordinates for the Tk window
    y = (hs/2) - (h/2)
    window.geometry('%dx%d+%d+%d' % (w, h, x, y))  # set the dimensions of the
screen and where it is placed
    window.resizable(False, False)    # disabling the resize option for the window
    window.configure(background='white')    # making the background white of the
window

def adjustWindow3(window):
    w = 870  # width for the window size
    h = 600  # height for the window size
    ws = screen.winfo_screenwidth()  # width of the screen
    hs = screen.winfo_screenheight()  # height of the screen
    x = (ws/2) - (w/2)  # calculate x and y coordinates for the Tk window
    y = (hs/2) - (h/2)
    window.geometry('%dx%d+%d+%d' % (w, h, x, y))  # set the dimensions of the
screen and where it is placed
    window.resizable(False, False)    # disabling the resize option for the window
    window.configure(background='white')    # making the background white of the
window

# This function is used for adjusting window size and making the necessary
configuration on start of window
def adjustWindow1(window):
    w = screen.winfo_screenwidth()  # width of the screen
    h = screen.winfo_screenheight()  # height of the screen
    x = (w/2)  # calculate x and y coordinates for the Tk window
    y = (h/2)
    window.geometry('%dx%d+%d+%d' % (w, h, x, y))  # set the dimensions of the
screen and where it is placed
    #window.resizable(False, False)    # disabling the resize option for the window
    window.configure(background='white')    # making the background white of the
window

def adjustWindow2(window):
    w = 900
    h = 750
    ws = welcome_screen.winfo_screenwidth()
    hs = welcome_screen.winfo_screenheight()
    x = (ws/3.5) - (w/3.5)
    y = (hs/3.5) - (h/3.5)
    window.geometry('%dx%d+%d+%d' % (w, h, x, y))
    window.resizable(False, False)
# registration window
def register():
```

```
    global screen1, fullname, email, password, repassword, country, gender, tnc # 
making all entry field variable global
    fullname = StringVar()
    email = StringVar()
    password = StringVar()
    repassword = StringVar()
    gender = StringVar()
    country = IntVar()
    tnc = IntVar()
    screen1 = Toplevel(screen)
    screen1.title("Civil Aviation")
    adjustWindow(screen1) # configuring the window
    Label(screen1, text="Registration Form", width='55', height="2", font=
("Calibri", 22, 'bold'), fg='#d9660a').place(x=0, y=0)
    #Label(screen1, text="", bg='#174873', width='55', height='30').place(x=105, 
y=100)

    photo = PhotoImage(file="reg.png") # opening left side image - Note: If image is 
in same folder then no need to mention the full path
    label = Label(screen1, image=photo, text="") # attaching image to the label
    label.place(x=-1, y=80)
    label.image = photo # it is necessary in Tkinter to keep a instance of image to 
display image in label


    #Label(screen1, text="REGISTRATION FORM", font=("Open Sans", 22, 'bold'), 
fg='white', bg='#174873', anchor=W).place(x=250, y=30)
    #Label(screen1, text="Registration Form", width='55', height="1", font=
("Calibri", 22, 'bold'), fg='#d9660a').place(x=0, y=20)
    Label(screen1, text="Full Name:", font=("Open Sans", 11, 'bold'), fg='white', 
bg='#174873', anchor=W).place(x=350, y=130)
    Entry(screen1, textvar=fullname).place(x=480, y=130)
    Label(screen1, text="Email ID:", font=("Open Sans", 11, 'bold'), fg='white', 
bg='#174873', anchor=W).place(x=350, y=180)
    Entry(screen1, textvar=email).place(x=480, y=180)
    Label(screen1, text="Country:", font=("Open Sans", 11, 'bold'), fg='white', 
bg='#174873', anchor=W).place(x=350, y=230)
    Radiobutton(screen1, text="India", variable=country, value=1,fg='white', 
bg='#174873').place(x=480, y=230)
    Radiobutton(screen1, text="Other", variable=country, value=2,fg='white', 
bg='#174873').place(x=550, y=230)

    Label(screen1, text="gender:", font=("Open Sans", 11, 'bold'), fg='white', 
bg='#174873', anchor=W).place(x=350, y=280)
    list1 = ['Male','Female','Other']
    droplist = OptionMenu(screen1, gender, *list1)
    droplist.config(width=18)
    gender.set('--select your gender--')
    droplist.place(x=480, y=275)
    Label(screen1, text="Password:", font=("Open Sans", 11, 'bold'), 
fg='white',bg='#174873', anchor=W).place(x=350, y=330)
    Entry(screen1, textvar=password, show="*").place(x=480, y=330)
    Label(screen1, text="Re-Password:", font=("Open Sans", 11, 'bold'), 
fg='white',bg='#174873', anchor=W).place(x=350, y=380)
    entry_4 = Entry(screen1, textvar=repassword, show="*")
    entry_4.place(x=480, y=380)
    Checkbutton(screen1, text="I accept all terms and conditions", variable=tnc, 
font=("Open Sans", 9, 'bold'), fg='brown').place(x=375, y=430)
    Button(screen1, text='Submit', width=15, font=("Open Sans", 13, 'bold'), 
bg='brown', fg='black',command=register_user).place(x=330, y=480)
    Button(screen1, text='Back', width=10, font=("Open Sans", 13, 'bold'), 
bg='brown', fg='black',command=screen1.destroy).place(x=550, y=480)

    photo = PhotoImage(file="log.png") # opening left side image - Note: If image is 
in same folder then no need to mention the full path
    label = Label(screen1, image=photo, text="") # attaching image to the label
    label.place(x=130, y=0)
```

```
    label.image = photo # it is necessary in Tkinter to keep a instance of image to
display image in label



def register_user():
    if fullname.get() and email.get() and password.get() and repassword.get() and
country.get(): # checking for all empty values in entry field
        if gender.get() == "--select your gender--": # checking for selection of
university
            Label(screen1, text="Please select your gender", fg="red",
                  font=("calibri", 11), width='30', anchor=W, bg='white').place(x=0,
y=570)
            return
        else:
            if tnc.get(): # checking for acceptance of agreement
                if re.match("^.+@(\[?)[a-zA-Z0-9-.]+.([a-zA-Z]{2,3}|[0-9]{1,3})
(]?)$", email.get()): # validating the email
                    if password.get() == repassword.get(): # checking both password
match or not
                        # if u enter in this block everything is fine just enter the
values in database
                        country_value = 'India'
                        if country.get() == 2:

                            country_value = 'Other'
                        connection = pymysql.connect(host="localhost", user="root",
passwd="", database="bruteforce") # database connection
                        cursor = connection.cursor()
                        insert_query = "INSERT INTO registration_details (fullname,
email, password, country,gender) VALUES('"+ fullname.get() + "', '"+ email.get() +
"', '"+ password.get() + "', '"+ country_value + "', '"+ gender.get() + "' );" #
queries for inserting values
                        cursor.execute(insert_query) # executing the queries
                        connection.commit() # commiting the connection then closing
it.
                        connection.close() # closing the connection of the database
                        Label(screen1, text="Registration Sucess", fg="green", font=
( "calibri", 11), width='30', anchor=W, bg='white').place(x=0, y=570) # printing
successful registration message
                        Button(screen1, text='Proceed to Login ->', width=20, font=
("Open Sans", 9, 'bold'), bg='brown',
fg='white',command=screen1.destroy).place(x=170, y=565) # button to navigate back to
login page
                    else:
                        Label(screen1, text="Password does not match", fg="red",
font=( "calibri", 11), width='30', anchor=W, bg='white').place(x=0, y=570)
                        return
                else:
                    Label(screen1, text="Please enter valid email id", fg="red",
font=("calibri", 11), width='30', anchor=W, bg='white').place(x=0, y=570)
                    return
            else:
                Label(screen1, text="Please accept the agreement", fg="red",
                      font=("calibri", 11), width='30', anchor=W,
bg='white').place(x=0, y=570)
                return
    else:
        Label(screen1, text="Please fill all the details", fg="red",
        font=("calibri", 11), width='30', anchor=W, bg='white').place(x=0, y=570)
        return

# login creditentials verification
def login_verify():
    global registrationID
    connection = pymysql.connect(host="localhost", user="root", passwd="",
database="bruteforce") # database connection
```

```
    cursor = connection.cursor()
    select_query =  "SELECT * FROM registration_details where email = '" +
username_verify.get() + "' AND password = '" + password_verify.get() + "';" #
queries for retrieving values
    cursor.execute(select_query) # executing the queries
    registration_info = cursor.fetchall()
    connection.commit() # commiting the connection then closing it.
    connection.close() # closing the connection of the database
    if registration_info:
        messagebox.showinfo("Congratulation", "Login Succesfull") # displaying
message for successful login
        registrationID = registration_info[0][0]
        welcome_page(registration_info) # opening welcome window
    else:
        messagebox.showerror("Error", "Invalid Username or Password") # displaying
message for invalid details


# welcome window
def welcome_page(registration_info):
    global screen2
    screen2 = Toplevel(screen)
    screen2.title("Civil Aviation")
    adjustWindow(screen2) # configuring the window
    Label(screen2, text="Welcome " + registration_info[0][1], width='47',
height="2", font=("Calibri", 25, 'bold'), fg='white', bg='#d9660a').place(x=0, y=0)
    Label(screen2, text="", bg='#174873', width='20', height='20').place(x=0, y=96)
    Message(screen2, text='"If we have data, let's look at data. If all we have are
opinions, let's go with mine."\n\n - — Jim Barksdale', width='100', font=
("Helvetica", 10, 'bold', 'italic'), fg='white', bg='#174873', anchor =
CENTER).place(x=10, y=100)

    photo1 = PhotoImage(file="analy.png") # opening right side image - Note: If
image is in same folder then no need to mention the full path
    label1 = Label(screen2, image=photo1, text="") # attaching image to the label
    label1.place(x=150, y=96)
    label1.image = photo1 # it is necessary in Tkinter to keep a instance of image
to display image in label

    photo1 = PhotoImage(file="pay.png") # opening right side image - Note: If image
is in same folder then no need to mention the full path
    label1 = Label(screen2, image=photo1, text="") # attaching image to the label
    label1.place(x=180, y=0)
    label1.image = photo1 # it is necessary in Tkinter to keep a instance of image
to display image in label


    Button(screen2, text='Fetch Raw Data', width=20,height=2, font=("Open Sans", 13,
'bold'),command=narenocr, bg='brown', fg='white').place(x=270, y=150)
    Button(screen2, text='Insert Record', width=20,height=2, font=("Open Sans", 13,
'bold'), bg='brown', fg='white',command=question_18_first_page).place(x=270, y=220)
    Button(screen2, text='Perform Function', width=20,height=2, font=("Open Sans",
13, 'bold'), bg='brown', fg='white',command=next_page).place(x=270, y=290)
    Button(screen2, text='Report Generation', width=20,height=2, font=("Open Sans",
13, 'bold'), bg='brown', fg='white',command=misgeneration).place(x=270, y=360)
    Button(screen2, text='Back', width=10, font=("Open Sans", 13, 'bold'),
bg='brown', fg='black',command=screen2.destroy).place(x=380, y=500)


def next_page():
    global screen3
    screen3 = Toplevel(screen)
    screen3.title("BRUTE FORCE")
    #adjustWindow(screen3)
    screen3.geometry("950x650")
    screen3.resizable(False,False)
```

```
    photo1 = PhotoImage(file="smok1.png") # opening right side image - Note: If
image is in same folder then no need to mention the full path
    label1 = Label(screen3, image=photo1, text="") # attaching image to the label
    label1.place(x=-5, y=50)
    label1.image = photo1 # it is necessary in Tkinter to keep a instance of image
to display image in label

    Label(screen3, text="--Functions-- ",width=40, height=1, font=("Open Sans", 30,
'bold'), bg='#174873', fg='white').place(x=0,y=0)

    Label(screen3, text="Page 1/3 ",width=8, height=1, font=("Open Sans", 13,
'bold'), fg='black', bg='white').place(x=400,y=600)

    Button(screen3, text="1) What is the percentage of birdstrike occuring in
India??",width=100, height=2, bg='brown', fg='white',font=("Open Sans", 10,
'bold'),command=question_1).place(x=80,y=84)
    Button(screen3, text="2) Average Altitude at which the bird strikes have occured
and the number of distinct bird species",width=100, height=2, bg='brown',
fg='white',font=("Open Sans", 10, 'bold'),command=question_2).place(x=80,y=166)
    Button(screen3, text="3) Were the Pilots given a prior warning for a bird strike
occurence???",width=100, height=2,font=("Open Sans", 10, 'bold'),command=question_3,
bg='brown', fg='white').place(x=80,y=248)
    Button(screen3, text="4) Predction compulsion related to time of the
day",width=100, height=2,font=("Open Sans", 10, 'bold'),command=question_4,
bg='brown', fg='white').place(x=80,y=332)
    Button(screen3, text="5) What is the damage cost trend compared to the size of
the bird over the period for which the data is being made available.",width=100,
height=2,font=("Open Sans", 10, 'bold'),command=question_5, bg='brown',
fg='white').place(x=80,y=416)
    Button(screen3, text="6) Phase of flight",width=100, height=2,font=("Open Sans",
10, 'bold'),command=question_6, bg='brown', fg='white').place(x=80,y=500)
    Button(screen3, text='< Back', width=10, font=("Open Sans", 13, 'bold'),
bg='brown', fg='black',command=screen3.destroy).place(x=40, y=590)
    Button(screen3, text='Next >', width=10, font=("Open Sans", 13, 'bold'),
bg='brown', fg='black',command=next_page1).place(x=800, y=590)


def next_page1():
    global screen311
    screen311 = Toplevel(screen)
    screen311.title("BRUTE FORCE")
    #adjustWindow(screen311)
    screen311.geometry("950x650")
    screen311.resizable(False,False)

    photo1 = PhotoImage(file="smok1.png") # opening right side image - Note: If
image is in same folder then no need to mention the full path
    label1 = Label(screen311, image=photo1, text="") # attaching image to the label
    label1.place(x=-5, y=50)
    label1.image = photo1 # it is necessary in Tkinter to keep a instance of image
to display image in label

    Label(screen311, text="--Functions-- ",width=40, height=1, font=("Open Sans",
30, 'bold'), bg='#174873', fg='white').place(x=0,y=0)

    Label(screen311, text="Page 2/3 ",width=8, height=1, font=("Open Sans", 13,
'bold'), fg='black', bg='white').place(x=400,y=600)


    Button(screen311, text="7) Number of Birdstrikes taking place yearly",width=100,
height=2,font=("Open Sans", 10, 'bold'),command=question_7, bg='brown',
fg='white').place(x=80,y=84)
    Button(screen311, text="8) Prediction of Birdstrikes in the forthcoming
years",width=100, height=2, bg='brown', fg='white',font=("Open Sans", 10,
'bold'),command=mlwindow).place(x=80,y=164)
    Button(screen311, text="9) Precipitation Status Analysis",width=100, height=2,
bg='brown', fg='white',font=("Open Sans", 10,
```

```
'bold'),command=question_9).place(x=80,y=248)
    Button(screen311, text="10) Remains of Wildlife Corrected",width=100,
height=2,font=("Open Sans", 10, 'bold'),command=question_10, bg='brown',
fg='white').place(x=80,y=332)
    Button(screen311, text="11) Monthly analysis of birdstrikes",width=100,
height=2,font=("Open Sans", 10, 'bold'),command=question_11, bg='brown',
fg='white').place(x=80,y=416)
    Button(screen311, text="12) Type of Aerial vehicle meeting accident in civil
aviation",width=100, height=2,font=("Open Sans", 10, 'bold'),command=question_12,
bg='brown', fg='white').place(x=80,y=500)


    Button(screen311, text='< Back', width=10, font=("Open Sans", 13, 'bold'),
bg='brown', fg='black',command=screen311.destroy).place(x=40, y=590)
    Button(screen311, text='Next >', width=10, font=("Open Sans", 13, 'bold'),
bg='brown', fg='black',command=next_page2).place(x=800, y=590)

def next_page2():
    global screen312
    screen312 = Toplevel(screen)
    screen312.title("BRUTE FORCE")
    #adjustWindow(screen312)
    screen312.geometry("950x650")
    screen312.resizable(False,False)

    photo1 = PhotoImage(file="smok1.png") # opening right side image - Note: If
image is in same folder then no need to mention the full path
    label1 = Label(screen312, image=photo1, text="") # attaching image to the label
    label1.place(x=-5, y=50)
    label1.image = photo1 # it is necessary in Tkinter to keep a instance of image
to display image in label

    Label(screen312, text="--Functions-- ",width=40, height=1, font=("Open Sans",
30, 'bold'), bg='#174873', fg='white').place(x=0,y=0)

    Label(screen312, text="Page 3/3 ",width=8, height=1, font=("Open Sans", 13,
'bold'), fg='black', bg='white').place(x=400,y=600)


    Button(screen312, text="13) Study and find out the relation between the
Sentiment-polarity and sentiment-subjectivity of all the apps.",width=90,
height=2,font=("Open Sans", 10, 'bold'), bg='brown',
fg='white',command=question_13).place(x=80,y=84)
    Button(screen312, text="""14) Generate an interface where the client can see the
reviews categorized as positive.negative and neutral
    ,once they have selected the app from a list of apps available for the
study.""",width=90, height=2,font=("Open Sans", 10, 'bold'), bg='brown',
fg='white',command=question_14).place(x=80,y=164)


    Button(screen312, text="15) Number of Fatal and Non Fatal incidents",width=95,
height=2, bg='brown', fg='white',command=question_15,font=("Open Sans", 10,
'bold')).place(x=80,y=248)
    '''
    Button(screen312, text="16) Which month(s) of the year, is the best indicator to
the avarage downloads that an app will generate over the entire year?",width=90,
height=2,font=("Open Sans", 10, 'bold'), bg='brown',
fg='white',command=question_16).place(x=80,y=332)
    Button(screen312, text="""17) Does the size of the App influence the number of
installs that it gets ?
    if,yes the trend is positive or negative with the increase in the app
size.""",width=90, height=2,font=("Open Sans", 10, 'bold'), bg='brown',
fg='white',command=question_17).place(x=80,y=416)
    Button(screen312, text="18) Provide an interface to add new data to both the
datasets provided.",width=90, height=2,font=("Open Sans", 10, 'bold'), bg='brown',
fg='white',command=question_18_first_page).place(x=80,y=500)
    '''
```

```
    Button(screen312, text='< Back', width=10, font=("Open Sans", 13, 'bold'),
bg='brown', fg='black',command=screen312.destroy).place(x=40, y=590)
    #Button(screen312, text='Next >', width=10, font=("Open Sans", 13, 'bold'),
bg='brown', fg='black').place(x=800, y=590)



def main_screen():
    global screen, username_verify, password_verify
    screen = Tk()
    username_verify = StringVar()
    password_verify = StringVar()
    screen.title("BRUTE FORCE")
    adjustWindow(screen)

    img2 = ImageTk.PhotoImage(Image.open("bigdata.png"))
    d1 = Label(image= img2)
    d1.place(x =-1,y = 78)

    Label(screen,text="Civil Aviation",width="55",height="2",font=
("Calibri",22,'bold'),bg='white',fg='#174873').pack()
    Label(screen, text="Please enter details below to login", bg='#174873',
fg='white').place(x=312,y=150)
    Label(screen, text="Username * ", font=("Open Sans", 10, 'bold'), bg='#174873',
fg='white').place(x=358,y=192)
    Entry(screen, textvar=username_verify).place(x=337,y=215)
    Label(screen, text="Password * ", font=("Open Sans", 10, 'bold'), bg='#174873',
fg='white').place(x=358,y=262)
    Entry(screen, textvar=password_verify, show="*").place(x=337,y=285)
    Button(screen, text="LOGIN", bg="#e79700", width=15, height=1, font=("Open
Sans", 13, 'bold'), fg='white', command=login_verify).place(x=320,y=325)
    Button(screen, text="New User? Register Here", height="2", width="30",
bg='#e79700', font=("Open Sans", 10, 'bold'), fg='white',
command=register).place(x=270,y=380)

    img1 = ImageTk.PhotoImage(Image.open("bft.png"))
    c1 = Label(image= img1)
    c1.place(x =142,y = 0)



    screen.mainloop()

def question_1() :
    global screen1
    screen1 = Tk()
    screen1.title('Question-1')
    adjustWindow1(screen1)


    x1=data_naren.loc[(data_naren['Airport: Name']=='BOMBAY/MUMBAI'),['Record
ID']].count()
    x2=data_naren.loc[(data_naren['Airport: Name']=='BANGALORE INTL ARPT'),['Record
ID']].count()
    x3=data_naren.loc[(data_naren['Airport: Name']=='KOLKATA'),['Record
ID']].count()
    x4=data_naren.loc[(data_naren['Airport: Name']=='DELHI'),['Record ID']].count()

    name=data_naren['Airport: Name']

    a=0 #loop for mumbai
    mumlist=[]
    for x in name:
        if (x=='BOMBAY/MUMBAI'):
```

```
            mumlist.append(0)
        else:
            a=0

    countmum=mumlist.count(0)

    b=0 #loop for bangalore
    banglist=[]
    for x in name:
        if (x=='BANGALORE'):
            banglist.append(0)
        else:
            b=0

    countbang=banglist.count(0)

    c=0 #loop for kolkata
    kollist=[]
    for x in name:
        if (x=='KOLKATA'):
            kollist.append(0)
        else:
            c=0

    countkol=kollist.count(0)

    d=0 # loop for delhi
    dellist=[]
    for x in name:
        if (x=='DELHI'):
            dellist.append(0)
        else:
            d=0

    countdel=dellist.count(0)

    total_rows=len(data_naren.axes[0])

    per1=(countmum/total_rows)*100 #percentage birdstrike in mumbai
    print(per1)

    per2=(countbang/total_rows)*100 #percentage birdstrike in mumbai
    print(per2)

    per3=(countkol/total_rows)*100 #percentage birdstrike in mumbai
    print(per3)

    per4=(countdel/total_rows)*100 #percentage birdstrike in mumbai
    print(per4)

    per6=((total_rows-countmum)/total_rows)*100 #percentage overseas birdstrike
    print(per6)

    Label(screen1,text="Percentage Birdstrikes in Mumbai",fg = "black",bg =
"white",font = "Verdana 10 bold").place(x=10,y=0)
    Label(screen1,text=per1,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=45,y=20)
    Label(screen1,text="Percentage Birdstrikes in Bangalore",fg = "black",bg =
"white",font = "Verdana 10 bold").place(x=282,y=0)
    Label(screen1,text=per2,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=400,y=20)
    Label(screen1,text="Percentage Birdstrikes in Kolkata",fg = "black",bg =
"white",font = "Verdana 10 bold").pack()
    Label(screen1,text=per3,fg = "black",bg = "white",font = "Verdana 10
bold").pack()
    Label(screen1,text="Percentage Birdstrikes in Delhi",fg = "black",bg =
"white",font = "Verdana 10 bold").place(x=820,y=0)
```

```
    Label(screen1,text=per4,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=927.5,y=20)

    x = ['Bombay', 'Bangalore', 'Kolkata', 'Delhi', 'Overseas']
    popularity = [x1, x2, x3, x4, total_rows]
    x_pos = [i for i, _ in enumerate(x)]

    fig, ax = plt.subplots()
    rects1 = ax.bar(x_pos, popularity, color='b')
    plt.xlabel("Indian Metropolitian and Overseas")
    plt.ylabel("Number of Incidents")
    plt.title("Count of Birdstrikes in Indian Metropolitian")
    plt.xticks(x_pos, x)
    # Turn on the grid
    plt.minorticks_on()
    plt.grid(which='major', linewidth='0.5', color='red')
    # Customize the minor grid
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,'%f' %
float(height),ha='center', va='bottom')
    autolabel(rects1)

    plt.show()

    canvas = FigureCanvasTkAgg(fig, screen1)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)

    button = Button(screen1, text="Quit", command=screen1.destroy)
    button.pack(side=BOTTOM)


def question_2() :
    global screen2
    screen2 = Tk()
    screen2.title('Question-2')
    adjustWindow(screen2)

    total_rows=len(data_naren.axes[0])

    chars_to_remove = [',']
    cols_to_clean = ['Feet above ground']
    for col in cols_to_clean:
        for char in chars_to_remove:
            data_naren[col] = data_naren[col].str.replace(char, '')
    data_naren[col] = to_numeric(data_naren[col])

    name=data_naren['Airport: Name']
    altitude=data_naren['Feet above ground'].sum()


    averagealtitude=(altitude/total_rows)

    Label(screen2,text="Average altitude at which bird strikes occur:",fg =
"black",bg = "white",font = "Verdana 10 bold").place(x=10,y=0)
    Label(screen2,text=averagealtitude,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=335,y=0)

    accident1=data_safety['incidents_85_99'].sum()

    Label(screen2,text="Number of Civil Aviation Incidents from 1985-1999",fg =
"black",bg = "white",font = "Verdana 10 bold").place(x=10,y=50)
    Label(screen2,text=accident1,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=410,y=50)
```

```
    accident2=data_safety['incidents_00_14'].sum()

    Label(screen2,text="Number of Civil Aviation Incidents from 2000-2014",fg =
"black",bg = "white",font = "Verdana 10 bold").place(x=10,y=100)
    Label(screen2,text=accident2,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=410,y=100)

    data_naren6868=pd.read_csv("bird.csv")
    species=data_naren6868['Wildlife: Species']

    rows = data_naren6868['Wildlife: Species']

    result = []

    for r in rows:
        key = r
        if key not in result:
            result.append(r)


    a=len(result)
    print(a)

    Label(screen2,text="Number of Distinct Bird Species",fg = "black",bg =
"white",font = "Verdana 10 bold").place(x=10,y=20)
    Label(screen2,text=a,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=335,y=20)
    button = Button(screen2, text="Quit", command=screen2.destroy)
    button.pack(side=BOTTOM)

def question_3() :
    global screen3
    screen3 = Tk()
    screen3.title('Question-3')
    adjustWindow(screen3)

    total_rows=len(data_naren.axes[0])
    warning=data_naren['Pilot warned of birds or wildlife?']


    ylist=[]
    for x in warning:
        if (x=='Y'):
            ylist.append(0)
        else:
            y=0

    county=ylist.count(0)

    n=0
    nlist=[]
    for x in warning:
        if (x=='N'):
            nlist.append(0)
        else:
            b=0

    countn=nlist.count(0)

    x1=data_naren.loc[(data_naren['Pilot warned of birds or wildlife?']=='Y'),
['Record ID']].count()
    x2=data_naren.loc[(data_naren['Pilot warned of birds or wildlife?']=='N'),
['Record ID']].count()
    x3=total_rows-(countn+county)
    y=0
```

```
    per1=(county/total_rows)*100 #percentage birdstrike in mumbai
    print(per1)

    per2=(countn/total_rows)*100 #percentage birdstrike in mumbai
    print(per2)

    per6=((total_rows-(county+countn))/total_rows)*100 #percentage overseas
birdstrike
    print(per6)

    Label(screen3,text="Percentage of pilots warned",fg = "black",bg = "white",font
= "Verdana 10 bold").place(x=290,y=0)
    Label(screen3,text=per1,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=320,y=20)
    Label(screen3,text="Percentage of pilots not warned",fg = "black",bg =
"white",font = "Verdana 10 bold").pack()
    Label(screen3,text=per2,fg = "black",bg = "white",font = "Verdana 10
bold").pack()
    Label(screen3,text="No information provided",fg = "black",bg = "white",font =
"Verdana 10 bold").place(x=890,y=0)
    Label(screen3,text=per6,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=900,y=20)
    Label(screen3,text='''As we can see from this graph,
          in majority of the cases pilots were not warned
          and therefore we must ensure that pilots are warned of birdstrikes in
advance''',fg = "black",bg = "white",font = "Verdana 10 bold").pack()

    x = ['Yes', 'No', 'No information']
    popularity = [x1, x2, x3]
    x_pos = [i for i, _ in enumerate(x)]

    fig, ax = plt.subplots()
    rects1 = ax.bar(x_pos, popularity, color='g')
    plt.xlabel("Pilots Warning Status")
    plt.ylabel("Number of Warnings")
    plt.title("Pilot Warnings for Bird Strikes")
    plt.xticks(x_pos, x)
    # Turn on the grid
    plt.minorticks_on()
    plt.grid(which='major', linewidth='0.5', color='blue')
    # Customize the minor grid
    plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,'%f' %
float(height),ha='center', va='bottom')
    autolabel(rects1)

    plt.show()


    canvas = FigureCanvasTkAgg(fig, screen3)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)
    button = Button(screen3, text="Quit", command=screen3.destroy)
    button.pack(side=BOTTOM)

def question_4() :
    global screen4
    screen4 = Tk()
    screen4.title('Question-4')
    adjustWindow1(screen4)

    total_rows=len(data_naren.axes[0])
```

```
    timeofday=data_naren['When: Time of day']

    d=0
    daylist=[]
    for x in timeofday:
        if (x=='Day'):
            daylist.append(0)
        else:
            d=0

    countday=daylist.count(0)

    n=0
    nightlist=[]
    for x in timeofday:
        if (x=='Night'):
            nightlist.append(0)
        else:
            b=0

    countnight=nightlist.count(0)

    dawn=0
    dawnlist=[]
    for x in timeofday:
        if (x=='Dawn'):
            dawnlist.append(0)
        else:
            b=0

    countdawn=dawnlist.count(0)

    dusk=0
    dusklist=[]
    for x in timeofday:
        if (x=='Dusk'):
            dusklist.append(0)
        else:
            b=0

    countdusk=dusklist.count(0)


    x1=data_naren.loc[(data_naren['When: Time of day']=='Day'),['Record
ID']].count()
    x2=data_naren.loc[(data_naren['When: Time of day']=='Night'),['Record
ID']].count()
    x3=data_naren.loc[(data_naren['When: Time of day']=='Dawn'),['Record
ID']].count()
    x4=data_naren.loc[(data_naren['When: Time of day']=='Dusk'),['Record
ID']].count()

    x5=total_rows-(countday+countnight+countdawn+countdusk)




    per1=(countday/total_rows)*100 #percentage birdstrike in mumbai
    print(per1)

    per2=(countnight/total_rows)*100 #percentage birdstrike in mumbai
    print(per2)

    per3=(countdawn/total_rows)*100 #percentage birdstrike in mumbai
    print(per3)

    per4=(countdusk/total_rows)*100 #percentage birdstrike in mumbai
```

```
    print(per4)

    per5=((total_rows-(countday+countnight+countdawn+countdusk))/total_rows)*100
#percentage overseas birdstrike
    print(per5)

    Label(screen4,text="No. of birdstrikes during day",fg = "black",bg =
"white",font = "Verdana 10 bold").place(x=120,y=0)
    Label(screen4,text=per1,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=120,y=20)
    Label(screen4,text="No. of birdstrikes at night",fg = "black",bg = "white",font
= "Verdana 10 bold").place(x=362,y=0)
    Label(screen4,text=per2,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=382,y=20)
    Label(screen4,text="No. of birdstrikes at dawn",fg = "black",bg = "white",font =
"Verdana 10 bold").pack()
    Label(screen4,text=per3,fg = "black",bg = "white",font = "Verdana 10
bold").pack()
    Label(screen4,text="No. of birdstrikes at dusk",fg = "black",bg = "white",font =
"Verdana 10 bold").place(x=820,y=0)
    Label(screen4,text=per4,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=820,y=20)
    Label(screen4,text="Flights taking off during time of day with maximum bird
strikes should be compulsorily warned",fg = "black",bg = "white",font = "Verdana 10
bold").pack()
    x = ['Day', 'Night', 'Dawn','Dusk','Not Known']
    popularity = [x1, x2, x3, x4, x5]
    x_pos = [i for i, _ in enumerate(x)]

    fig, ax = plt.subplots()
    rects1 = ax.bar(x_pos, popularity, color='y')
    plt.xlabel("Bird Strike Timing Status")
    plt.ylabel("Number of Bird Strikes")
    plt.title("Bird Strike Occurence Period")
    plt.xticks(x_pos, x)
    # Turn on the grid
    plt.minorticks_on()
    plt.grid(which='major', linewidth='0.5', color='black')
    # Customize the minor grid
    plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,'%f' %
float(height),ha='center', va='bottom')
    autolabel(rects1)

    plt.show()


    show()
    canvas = FigureCanvasTkAgg(fig, screen4)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)
    button = Button(screen4, text="Quit", command=screen4.destroy)
    button.pack(side=BOTTOM)




def question_5() :
    global screen5
    screen5 = Tk()
    screen5.title('Question-5')
    adjustWindow(screen5)
    data_naren99=pd.read_csv("bird.csv")
    data_naren12=pd.read_csv("bird.csv")
```

```
    data_naren21=pd.read_csv("bird.csv")

    chars_to_remove = [',']
    cols_to_clean = ['Feet above ground','Cost: Total $']
    for col in cols_to_clean:
        for char in chars_to_remove:
            data_naren99[col] = data_naren99[col].str.replace(char, '')
            data_naren12[col] = data_naren12[col].str.replace(char, '')
            data_naren21[col] = data_naren21[col].str.replace(char, '')
        data_naren99[col] = to_numeric(data_naren99[col])
        data_naren12[col] = to_numeric(data_naren12[col])
        data_naren21[col] = to_numeric(data_naren21[col])
    data_naren99.dropna(axis=1)
    data_naren12.dropna(axis=1)
    data_naren21.dropna(axis=1)

    data_naren1=data_naren99[(data_naren99['Wildlife: Size']=='Small')&
(data_naren99['Cost: Total $']>10000)]#wildlife size Small and damage cost above
10000
    data_naren2=data_naren12[(data_naren12['Wildlife: Size']=='Medium')&
(data_naren12['Cost: Total $']>10000)]#wildlife Medium large and damage cost above
10000
    data_naren3=data_naren21[(data_naren21['Wildlife: Size']=='Large')&
(data_naren21['Cost: Total $']>10000)]#wildlife size large and damage cost above
10000

    t1=data_naren1['Wildlife: Size']

    p=0
    t1list=[]
    for i in t1:
        if (i=='Small'):
            t1list.append(0)
        else:
            p=0
    x1=len(t1list)

    t2=data_naren2['Wildlife: Size']

    q=0
    t2list=[]
    for j in t2:
        if (j=='Medium'):
            t2list.append(0)
        else:
            q=0

    x2=len(t2list)
    t3=data_naren3['Wildlife: Size']

    r=0
    t3list=[]
    for k in t3:
        if (k=='Large'):
            t3list.append(0)
        else:
            p=0
    x3=len(t3list)


    x = ['''Small Size
       &>$10000 Cost''', '''Medium Size
       &>$10000 Cost''', '''Large Size
       &>$10000 Cost''']
    popularity = [x1, x2, x3]
    x_pos = [i for i, _ in enumerate(x)]
```

```
    fig, ax = plt.subplots()
    rects1 = ax.bar(x_pos, popularity, color='b')
    plt.xlabel("")
    plt.title("Count of Birdstrikes with Size of Bird and Damage Cost>10000")
    plt.xticks(x_pos, x)
        # Turn on the grid
    plt.minorticks_on()
    plt.grid(which='major', linewidth='0.5', color='red')
        # Customize the minor grid
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,'%f' %
float(height),ha='center', va='bottom')
    autolabel(rects1)

    plt.show()
    show()
    canvas = FigureCanvasTkAgg(fig, screen5)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)
    button = Button(screen5, text="Quit", command=screen5.destroy)
    button.pack(side=BOTTOM)

def question_6():
    global screen6
    screen6 = Tk()
    screen6.title('Question-6')
    adjustWindow(screen6)

    p1=data_naren['When: Phase of flight']

    p=0
    p1list=[]
    for i in p1:
        if (i=='Climb'):
            p1list.append(0)
        else:
            p=0
    x1=len(p1list)



    q=0
    p2list=[]
    for j in p1:
        if (j=='Take-off run'):
            p2list.append(0)
        else:
            q=0

    x2=len(p2list)


    r=0
    p3list=[]
    for k in p1:
        if (k=='Landing Roll'):
            p3list.append(0)
        else:
            r=0

    x3=len(p3list)

    s=0
    p4list=[]
    for k in p1:
```

```
        if (k=='Approach'):
            p4list.append(0)
        else:
            s=0
    x4=len(p4list)

    t=0
    p5list=[]
    for k in p1:
        if (k=='Taxi'):
            p5list.append(0)
        else:
            t=0
    x5=len(p5list)

    total_rows=len(data_naren.axes[0])

    unknown=(total_rows-(x1+x2+x3+x4+x5))

    x = ['Climb','Take-off run','Landing Roll','Approach','Taxi','Unknown']

    popularity = [x1,x2,x3,x4,x5,unknown]
    x_pos = [i for i, _ in enumerate(x)]

    fig, ax = plt.subplots()
    rects1 = ax.bar(x_pos, popularity, color='b')
    plt.xlabel("Phase of Flight")
    plt.title("Count of Birdstrikes as per Phases of Flight")
    plt.xticks(x_pos, x)
        # Turn on the grid
    plt.minorticks_on()
    plt.grid(which='major', linewidth='0.5', color='red')
        # Customize the minor grid
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,'%d' %
float(height),ha='center', va='bottom')
    autolabel(rects1)

    plt.show()

    show()
    canvas = FigureCanvasTkAgg(fig, screen6)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)
    button = Button(screen6, text="Quit", command=screen6.destroy)
    button.pack(side=BOTTOM)

def question_7():
    p1=data_naren['Year']

    a=0
    alist=[]
    for i in p1:
        if (i==2005):
            alist.append(0)
        else:
            a=0
    x1=len(alist)

    b=0
    blist=[]
    for i in p1:
        if (i==2006):
            blist.append(0)
        else:
```

```
            b=0
    x2=len(blist)

    c=0
    clist=[]
    for i in p1:
        if (i==2007):
            clist.append(0)
        else:
            c=0
    x3=len(clist)

    d=0
    dlist=[]
    for i in p1:
        if (i==2008):
            dlist.append(0)
        else:
            d=0
    x4=len(dlist)

    e=0
    elist=[]
    for i in p1:
        if (i==2009):
            elist.append(0)
        else:
            e=0
    x5=len(elist)

    f=0
    flist=[]
    for i in p1:
        if (i==2010):
            flist.append(0)
        else:
            f=0
    x6=len(flist)

    g=0
    glist=[]
    for i in p1:
        if (i==2011):
            glist.append(0)
        else:
            g=0
    x7=len(glist)

    fig = {
        "data": [{"type": "bar",
                  "x": ['2005','2006','2007','2008','2009','2010','2011'],
                  "y": [x1,x2,x3,x4,x5,x6,x7]}],
        "layout": {"title": {"text": "A Bar Chart"}}
    }

# To display the figure defined by this dict, use the low-level plotly.io.show
function
    import plotly.io as pio
    pio.show(fig)
    plot(fig)


def check(s) :
    a = 0
    for i in s :
        if (i >='0' and i<='9') or i == '.' :
            a+=1
```

```
    if a == len(s) :
        return True
    else :
        return False

def check_review(s) :
    a = 0
    for i in s :
        if (i >='0' and i<='9'):
            a+=1
    if a == len(s) :
        return True
    else :
        return False

def check_string(s):
    for i in s :
        if (i>='a' and i<='z') or (i>='A' and i<='Z'):
            return False
    return True

def hide(choice):
    global e, hidden_text
    hidden_text= StringVar()
    if choice == "Paid":
        Label(screen18, text= 'Enter the price for damage:',
fg='black').place(x=200, y=300)
        e = Entry(screen18, textvariable = hidden_text, width=45)
        e.place(x=400, y=300)
    else:
        e = Entry(screen18, textvariable = hidden_text, width=45)
        e.place(x=400, y=300)

def validate() :
    if app_variable.get() != '' :
        if rating_variable.get() != '' and check(rating_variable.get()) :
            if review_variable.get() != '' and check_review(review_variable.get()) :
                if size_variable.get() != '' and check(size_variable.get()) :
                    if current_version_variable.get() != '' and
check(current_version_variable.get()):
                        if android_version_variable.get() != '' and
check(android_version_variable.get()) :
                            if category_variable.get() != '--Select Category--' and
genre_variable.get() != '--Select Genre--' and install_variable.get() != '--Select
No. of Installs--' and type_variable.get() != '--Select Type--'  and
content_variable.get() != '--Select Content Rating--' and day_variable.get() != '--
Select Day' and month_variable.get() != '--Select Month--' and year_variable.get()
!= '--Select Year--':
                                if type_variable.get() == 'Paid' and e.get() != ''
and check(e.get()):
                                    app_cost = e.get()
                                    day_list = month_variable.get() + ' ' +
day_variable.get() + ',' + year_variable.get()
                                    csv_data = []
                                    csv_data.append([app_variable.get(),
category_variable.get(), rating_variable.get(), review_variable.get(),
size_variable.get(), install_variable.get(),type_variable.get(), app_cost,
content_variable.get(), genre_variable.get(), day_list,
current_version_variable.get(), android_version_variable.get()])
                                    df = pd.DataFrame(csv_data)
                                    df.to_csv("dummy_data_bird.csv")
                                    messagebox.showinfo("Success", "Data has been
stored into the csv file!")
                                    screen18.destroy()
                                elif type_variable.get() == 'Free':
                                    app_cost = 0
                                    day_list = month_variable.get() + ' ' +
```

```
day_variable.get() + ',' + year_variable.get()
                                    csv_data = []
                                    csv_data.append([app_variable.get(),
category_variable.get(), rating_variable.get(), review_variable.get(),
size_variable.get(), install_variable.get(),type_variable.get(), app_cost,
content_variable.get(), genre_variable.get(), day_list,
current_version_variable.get(), android_version_variable.get()])
                                    df = pd.DataFrame(csv_data)
                                    df.to_csv("dummy_data_bird.csv")
                                    messagebox.showinfo("Success", "Data has been
stored into the csv file!")
                                    screen18.destroy()
                                else :
                                    messagebox.showerror("Error", "Please enter a
proper price for the app!", parent=screen18)
                            else :
                                messagebox.showerror("Error", "Please select an
option from the drop list!", parent=screen18)
                        else :
                            messagebox.showerror("Error", "Please enter android
version for the app!", parent=screen18)
                    else :
                        messagebox.showerror("Error", "Please enter correct data!",
parent=screen18)
                else :
                    messagebox.showerror("Error", "Please enter data!",
parent=screen18)
            else :
                messagebox.showerror("Error", "Please enter data", parent=screen18)
        else :
            messagebox.showerror("Error", "Please enter data", parent=screen18)
    else :
        messagebox.showerror("Error", "Please fill in the data", parent=screen18)

def make_unique(dummy_list) :
    true_list = []
    for item in dummy_list :
        if item not in true_list  :
            true_list.append(item)
    return true_list

def question_18() :
    global screen18, category_variable, genre_variable, install_variable,
content_variable, type_variable, rating_variable, review_variable, size_variable,
app_variable, current_version_variable, android_version_variable, month_variable,
day_variable, year_variable
    screen18 = Tk()
    screen18.title('Question-18')
    adjustWindow3(screen18)
    category_variable = StringVar(screen18)
    genre_variable = StringVar(screen18)
    install_variable = StringVar(screen18)
    type_variable = StringVar(screen18)
    content_variable = StringVar(screen18)
    rating_variable = StringVar()
    review_variable = StringVar()
    size_variable = StringVar()
    app_variable = StringVar()
    current_version_variable = StringVar()
    android_version_variable = StringVar()
    month_variable = StringVar(screen18)
    day_variable = StringVar(screen18)
    year_variable = StringVar(screen18)
    Label(screen18, text= 'Fill up all the details of the app', fg='black').pack()
    Label(screen18, text= 'Enter name of the airport : ', fg='black').place(x=230,
y=40)
    app_variable = Entry(screen18, width = 50)
```

```
    app_variable.place(x=400, y=40)
    dummy_category = data_naren1['Category']
    categories = make_unique(dummy_category)
    #Label(screen18, text= 'Select Category', fg='black').place(x=50, y=320)
    droplist_category = OptionMenu(screen18, category_variable, *categories)
    category_variable.set('-Select Aircraft Model-')
    droplist_category.config(width=20)
    droplist_category.place(x=30, y=340)
    dummy_genre = data_naren1['Genres']
    genres = make_unique(dummy_genre)
    #Label(screen18, text= 'Select Genre', fg='black').place(x=200, y=320)
    droplist_genre = OptionMenu(screen18, genre_variable, *genres)
    genre_variable.set('-Select Bird Species-')
    droplist_genre.config(width=20)
    droplist_genre.place(x=200, y=340)
    dummy_install = data_naren1['Installs']
    installs = make_unique(dummy_install)
    #Label(screen18, text= 'Select No. of Installs', fg='black').place(x=350, y=320)
    droplist_install = OptionMenu(screen18, install_variable, *installs)
    install_variable.set('-Were the Pilots Warned?-')
    droplist_install.config(width=20)
    droplist_install.place(x=370, y=340)
    dummy_type = data_naren1['Type']
    types = make_unique(dummy_type)
    #Label(screen18, text= 'Select Type', fg='black').place(x=500, y=320)
    droplist_type = OptionMenu(screen18, type_variable, *types, command = hide)
    type_variable.set('-Damage Cost??-')
    droplist_type.config(width=20)
    droplist_type.place(x=540, y=340)
    dummy_content = data_naren1['Content Rating']
    contents = make_unique(dummy_content)
    droplist_content = OptionMenu(screen18, content_variable, *contents)
    content_variable.set('-Casulties?-')
    droplist_content.config(width=20)
    droplist_content.place(x=710, y=340)
    Label(screen18, text= 'Enter altitude(in thousand of feet) : ',
fg='black').place(x=160, y=80)
    rating_variable = Entry(screen18, width = 50)
    rating_variable.place(x=400, y=80)
    Label(screen18, text= 'Enter speed of aircraft (in knots) : ',
fg='black').place(x=70, y=120)
    review_variable = Entry(screen18, width = 50)
    review_variable.place(x=400, y=120)
    Label(screen18, text= 'Enter cost of damage : ', fg='black').place(x=160, y=160)
    size_variable = Entry(screen18, width = 50)
    size_variable.place(x=400, y=160)
    Label(screen18, text= 'Enter record id of bird strike : ',
fg='black').place(x=210, y=200)
    current_version_variable = Entry(screen18, width = 50)
    current_version_variable.place(x=400, y=200)
    Label(screen18, text= 'Enter miles from nearest airport : ',
fg='black').place(x=210, y=240)
    android_version_variable = Entry(screen18, width = 50)
    android_version_variable.place(x=400, y=240)
    month_of_year =
['January','February','March','April','May','June','July','August','September','Octo
ber','November','December']
    #Label(screen18, text= 'Enter month', fg='black').place(x=200, y=380)
    droplist_month = OptionMenu(screen18, month_variable, *month_of_year)
    month_variable.set('--Select Month--')
    droplist_month.config(width=20)
    droplist_month.place(x=170, y=400)
    days_31 =
['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','1
9','20','21','22','23','24','25','26','27','28','29','30','31']
    #Label(screen18, text= 'Enter day', fg='black').place(x=400, y=380)
    droplist_day = OptionMenu(screen18, day_variable, *days_31)
```

```
    day_variable.set('--Select Day--')
    droplist_day.config(width=20)
    droplist_day.place(x=370, y=400)
    years_from_2015 =
['2015','2016','2017','2018','2019','2020','2021','2022','2023','2024','2025']
    #Label(screen18, text= 'Enter year', fg='black').place(x=600, y=380)
    droplist_year = OptionMenu(screen18, year_variable, *years_from_2015)
    year_variable.set('--Select Year--')
    droplist_year.config(width=20)
    droplist_year.place(x=570, y=400)
    Button(screen18, text='Submit', width=20, font=("Open Sans", 13, 'bold'),
bg='green', fg='white', command= validate).place(x=350, y=500)
    button = Button(screen18, text="Quit", command=screen18.destroy)
    button.pack(side=BOTTOM)

def validate_2() :
    if application_variable.get() != '':
        if sentiment_option_variable.get() != '--Select Sentiment--':
            if review_text.get("1.0",END) != '\n' :
                if check(sentiment_polarity_variable.get()) and
check(sentiment_subjectivity_variable.get()):
                    csv_data = []
                    csv_data.append([application_variable.get(),
sentiment_polarity_variable.get(), sentiment_subjectivity_variable.get(),
sentiment_option_variable.get()])
                    df = pd.DataFrame(csv_data)
                    df.to_csv("dummy_data_bird2.csv")
                    messagebox.showinfo("Success", "Data has been stored into the
csv file!")
                    screen18_2.destroy()
                else :
                    messagebox.showerror("Error", "Please enter a numeric value!",
parent=screen18_2)
            else :
                messagebox.showerror("Error", "Please enter something in the
reviews!", parent=screen18_2)
        else :
            messagebox.showerror("Error", "Please select an option from the drop
list!", parent=screen18_2)
    else :
        messagebox.showerror("Error", "Please provide a name!", parent=screen18_2)

def question_18_2() :
    global screen18_2, application_variable, sentiment_option_variable,
sentiment_polarity_variable, sentiment_subjectivity_variable, review_text
    screen18_2 = Tk()
    screen18_2.title('Question-18_2')
    adjustWindow(screen18_2)
    application_variable = StringVar()
    sentiment_option_variable = StringVar(screen18_2)
    sentiment_polarity_variable = StringVar()
    sentiment_subjectivity_variable = StringVar()
    Label(screen18_2, text= 'Fill up all the details of incident',
fg='black').pack()
    Label(screen18_2, text= 'Enter name of the passenger : ',
fg='black').place(x=230, y=40)
    application_variable = Entry(screen18_2, width=30)
    application_variable.place(x=400, y=40)
    sentiment_options = ['Positive', 'Negative', 'Neutral']
    Label(screen18_2, text= 'Select the sentiment for person: ',
fg='black').place(x=220, y=100)
    droplist_category = OptionMenu(screen18_2, sentiment_option_variable,
*sentiment_options)
    sentiment_option_variable.set('--Select Sentiment--')
    droplist_category.config(width=20)
    droplist_category.place(x=400, y=100)
    Label(screen18_2, text= 'Type the review for bird strike',
```

```
fg='black').place(x=0, y=160)
    review_text = Text(screen18_2, height=15, width=150)
    review_text.place(x=0, y=180)
    review_text.configure(state='normal')
    Label(screen18_2, text= 'Select the sentiment polarity for the incident(use
decimal): ', fg='black').place(x=90, y=450)
    sentiment_polarity_variable = Entry(screen18_2, width=20)
    sentiment_polarity_variable.place(x=450, y=450)
    Label(screen18_2, text= 'Select the sentiment subjectivity for the incident(use
decimal): ', fg='black').place(x=90, y=490)
    sentiment_subjectivity_variable = Entry(screen18_2, width=20)
    sentiment_subjectivity_variable.place(x=450, y=490)
    Button(screen18_2, text='Submit', width=20, font=("Open Sans", 13, 'bold'),
bg='green', fg='white', command= validate_2).place(x=350, y=550)
    button = Button(screen18_2, text="Quit", command=screen18_2.destroy)
    button.pack(side=BOTTOM)

def question_18_first_page() :
    global screen18_first
    screen18_first = Tk()
    screen18_first.title('Question-18-First')
    adjustWindow(screen18_first)
    Label(screen18_first, text= 'Choose any dataset for filling the details of bird
strike', fg='black').pack()
    Button(screen18_first, text='Dataset-1', width=50, height=10, font=("Open Sans",
13, 'bold'), bg='green', fg='white', command= question_18).place(x=200, y=100)
    Button(screen18_first, text='Dataset-2', width=50, height=10, font=("Open Sans",
13, 'bold'), bg='green', fg='white', command= question_18_2).place(x=200, y=400)
    button = Button(screen18_first, text="Quit", command=screen18_first.destroy)
    button.pack(side=BOTTOM)


def mlwindow():
    global screen55
    screen55 = Tk()
    screen55.title('Question-1')
    adjustWindow1(screen55)

    Label(screen55, text='The Prediction for birdstrikes is as follows' ,font=
('helvetica', 10, 'bold')).pack()
    Label(screen55, text='To predict for a particular year click the button below'
,font=('helvetica', 10, 'bold')).pack()
    Button(screen55,text='Get the Prediction',command=mlpredict, bg='brown',
fg='white', font=('helvetica', 9, 'bold')).pack()
    fig, ax = plt.subplots()

    data=np.array([
    [2005,7804],
    [2006,8010],
    [2007,8746],
    [2008,8903],
    [2009,10741],
    [2010,10923],
    [2011,10483]])

    x,y=data.T
    plt.ylabel("Incidents of Birdstrike")
    plt.xlabel("Year")
    plt.scatter(x,y)


    data1=np.array([
    [2005,7673.7857143],
    [2006,8240.1428572],
    [2007,8806.5000001],
    [2008,9372.857143],
    [2009,9939.2142859],
```

```
    [2010,10505.5714288],
    [2011,11071.9286717]])

    x,y=data1.T
    plt.ylabel("Incidents of Birdstrike")
    plt.xlabel("Year")
    plt.scatter(x,y)
    plt.plot(x,y)


    canvas = FigureCanvasTkAgg(fig, screen55)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)

    button = Button(screen55, text="Quit", command=screen55.destroy)
    button.pack(side=BOTTOM)

def mlpredict():
    import tkinter as tk

    root= tk.Tk()

    canvas1 = tk.Canvas(root, width = 400, height = 300,  relief = 'raised')
    canvas1.pack()

    label1 = tk.Label(root, text='Prediction of Number of Birdstrikes')
    label1.config(font=('helvetica', 14))
    canvas1.create_window(200, 25, window=label1)

    label2 = tk.Label(root, text='Type your Number:')
    label2.config(font=('helvetica', 10))
    canvas1.create_window(200, 100, window=label2)

    entry1 = tk.Entry (root)
    canvas1.create_window(200, 140, window=entry1)

    def getSquareRoot ():

        x1 = entry1.get()
        x2=(float(x1)*(566.35714729))-(1127872.286)
        x3=str(x2)
        label3 = tk.Label(root, text= 'The Prediction of Birdstrikes'' is:',font=
('helvetica', 10))
        canvas1.create_window(200, 210, window=label3)

        label4 = tk.Label(root, text=x3 ,font=('helvetica', 10, 'bold'))
        canvas1.create_window(200, 230, window=label4)

    button1 = tk.Button(root,text='Get the Prediction', command=getSquareRoot,
bg='brown', fg='white', font=('helvetica', 9, 'bold'))

    canvas1.create_window(200, 180, window=button1)

    root.mainloop()

def question_shri():

    fig1 = go.Figure()
    fig1.add_trace(go.Scatter(x=data_adithya['Record ID'], y=data_adithya['Wildlife:
Number struck'], name="estimate",line_color='deepskyblue'))
    fig1.add_trace(go.Scatter(x=data_adithya['Record ID'], y=data_adithya['Wildlife:
Number Struck Actual'], name="actual",line_color='green'))

    fig1.update_layout(title_text='DP',xaxis_rangeslider_visible=True)
    fig1.show()
    plot(fig1)
```

```
def question_9() :
    global screen9
    screen9 = Tk()
    screen9.title('Question-9')
    adjustWindow(screen9)


    data_adithya=pd.read_csv("strikenew.csv")

    total_rows=len(data_adithya.axes[0])
    precofday=data_adithya['Conditions: Precipitation']

    d=0
    foglist=[]
    for x in precofday:
        if (x=='Fog'):
            foglist.append(0)
        else:
            d=0

    countfog=foglist.count(0)

    n=0
    rainlist=[]
    for x in precofday:
        if (x=='Rain'):
            rainlist.append(0)
        else:
            n=0

    countrain=rainlist.count(0)

    snow=0
    snowlist=[]
    for x in precofday:
        if (x=='Snow'):
            snowlist.append(0)
        else:
            snow=0

    countsnow=snowlist.count(0)

    fograin=0
    fograinlist=[]
    for x in precofday:
        if (x=='Fog, Rain'):
            fograinlist.append(0)
        else:
            b=0

    countfograin=fograinlist.count(0)

    none=0
    nonelist=[]
    for x in precofday:
        if (x=='None'):
            nonelist.append(0)
        else:
            none=0

    countnone=nonelist.count(0)

    x1=data_adithya.loc[(data_adithya['Conditions: Precipitation']=='Fog'),['Record
ID']].count()
    x2=data_adithya.loc[(data_adithya['Conditions: Precipitation']=='Rain'),['Record
ID']].count()
    x3=data_adithya.loc[(data_adithya['Conditions: Precipitation']=='Snow'),['Record
```

```
ID']].count()
    x4=data_adithya.loc[(data_adithya['Conditions: Precipitation']=='Fog, Rain'),
['Record ID']].count()
    x5=data_adithya.loc[(data_adithya['Conditions: Precipitation']=='None'),['Record
ID']].count()

    x6=total_rows-(countsnow+countfog+countrain+countfograin+countnone)




    per1=(countfog/total_rows)*100 #percentage birdstrike in mumbai
    print(per1)

    per2=(countrain/total_rows)*100 #percentage birdstrike in mumbai
    print(per2)

    per3=(countsnow/total_rows)*100 #percentage birdstrike in mumbai
    print(per3)

    per4=(countfograin/total_rows)*100 #percentage birdstrike in mumbai
    print(per4)

    per5=(countnone/total_rows)*100 #percentage overseas birdstrike
    print(per5)

    per6=((total_rows-
(countsnow+countrain+countfog+countfograin+countnone))/total_rows)*100 #percentage
overseas birdstrike
    print(per6)

    x = ['Fog', 'Rain', 'Snow','Fog+Rain','None','No information']
    popularity = [x1, x2, x3, x4, x5, x6]
    x_pos = [i for i, _ in enumerate(x)]

    fig, ax = plt.subplots()
    rects1 = ax.bar(x_pos, popularity, color='cyan')
    plt.xlabel("Precipitation Status during Bird Strike ")
    plt.ylabel("Number of Bird Strikes")
    plt.title("Bird Strike: Precipitation Analysis")
    plt.xticks(x_pos, x)
    # Turn on the grid
    plt.minorticks_on()
    plt.grid(which='major', linewidth='0.5', color='black')
    # Customize the minor grid
    plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,'%f' %
float(height),ha='center', va='bottom')
    autolabel(rects1)

    plt.show()


    show()
    canvas = FigureCanvasTkAgg(fig, screen9)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)
    button = Button(screen9, text="Quit", command=screen9.destroy)
    button.pack(side=BOTTOM)

def question_10() :
    global screen10
    screen10 = Tk()
    screen10.title('Question-10')
```

```
    adjustWindow(screen10)


    data_adithya=pd.read_csv("strikenew.csv")

    total_rows=len(data_adithya.axes[0])
    remains=data_adithya['Remains of wildlife collected?']

    d=0
    truelist=[]
    for x in remains:
        if (x==TRUE):
            truelist.append(0)
        else:
            d=0

    counttrue=truelist.count(0)

    n=0
    falselist=[]
    for x in remains:
        if (x==FALSE):
            falselist.append(0)
        else:
            n=0

    countfalse=falselist.count(0)

    x1=data_adithya.loc[(data_adithya['Remains of wildlife collected?']==TRUE),
['Record ID']].count()
    x2=data_adithya.loc[(data_adithya['Remains of wildlife collected?']==FALSE),
['Record ID']].count()

    x6=total_rows-(counttrue+countfalse)




    per1=(counttrue/total_rows)*100 #percentage birdstrike in mumbai
    print(per1)

    per2=(countfalse/total_rows)*100 #percentage birdstrike in mumbai
    print(per2)


    per6=((total_rows-(counttrue+countfalse))/total_rows)*100 #percentage overseas
birdstrike
    print(per6)

    x = ['True', 'False', 'No information']
    popularity = [x1, x2, x6]
    x_pos = [i for i, _ in enumerate(x)]

    fig, ax = plt.subplots()
    rects1 = ax.bar(x_pos, popularity, color='black')
    plt.xlabel("Status of remains collected")
    plt.ylabel("Number of Bird Strikes")
    plt.title("Whether remains of birds were collected ??")
    plt.xticks(x_pos, x)
    # Turn on the grid
    plt.minorticks_on()
    plt.grid(which='major', linewidth='0.5', color='yellow')
    # Customize the minor grid
    plt.grid(which='minor', linestyle=':', linewidth='0.5', color='yellow')
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
```

```
            ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,'%f' %
float(height),ha='center', va='bottom')
    autolabel(rects1)

    plt.show()


    show()
    canvas = FigureCanvasTkAgg(fig, screen10)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)
    button = Button(screen10, text="Quit", command=screen10.destroy)
    button.pack(side=BOTTOM)

def question_11() :
    global screen10
    screen10 = Tk()
    screen10.title('Question-10')
    adjustWindow1(screen10)


    data_month=pd.read_csv("monthtest.csv")

    total_rows=len(data_month.axes[0])
    month=data_month['Month']

    jan=0
    janlist=[]
    for x in month:
        if (x=='Jan'):
            janlist.append(0)
        else:
            jan=0

    countjan=janlist.count(0)

    feb=0
    feblist=[]
    for x in month:
        if (x=='Feb'):
            feblist.append(0)
        else:
            feb=0
    countfeb=feblist.count(0)

    march=0
    marlist=[]
    for x in month:
        if (x=='Mar'):
            marlist.append(0)
        else:
            march=0
    countmar=marlist.count(0)

    april=0
    aprlist=[]
    for x in month:
        if (x=='Apr'):
            aprlist.append(0)
        else:
            april=0

    countapr=aprlist.count(0)

    may=0
    maylist=[]
    for x in month:
```

```
    if (x=='May'):
        maylist.append(0)
    else:
        may=0
countmay=maylist.count(0)

june=0
junlist=[]
for x in month:
    if (x=='Jun'):
        junlist.append(0)
    else:
        june=0

countjun=junlist.count(0)

jul=0
jullist=[]
for x in month:
    if (x=='Jul'):
        jullist.append(0)
    else:
        jul=0
countjul=jullist.count(0)

aug=0
auglist=[]
for x in month:
    if (x=='Aug'):
        auglist.append(0)
    else:
        aug=0
countaug=auglist.count(0)

sep=0
seplist=[]
for x in month:
    if (x=='Sep'):
        seplist.append(0)
    else:
        sep=0
countsep=seplist.count(0)

octo=0
octolist=[]
for x in month:
    if (x=='Oct'):
        octolist.append(0)
    else:
        octo=0
countocto=octolist.count(0)

nov=0
novlist=[]
for x in month:
    if (x=='Nov'):
        novlist.append(0)
    else:
        nov=0
countnov=novlist.count(0)

dec=0
declist=[]
for x in month:
    if (x=='Dec'):
        declist.append(0)
    else:
```

```
            dec=0

    countdec=declist.count(0)


    x1=data_month.loc[(data_month['Month']=='Jan'),['Record ID']].count()
    x2=data_month.loc[(data_month['Month']=='Feb'),['Record ID']].count()
    x3=data_month.loc[(data_month['Month']=='Mar'),['Record ID']].count()
    x4=data_month.loc[(data_month['Month']=='Apr'),['Record ID']].count()
    x5=data_month.loc[(data_month['Month']=='May'),['Record ID']].count()
    x6=data_month.loc[(data_month['Month']=='Jun'),['Record ID']].count()
    x7=data_month.loc[(data_month['Month']=='Jul'),['Record ID']].count()
    x8=data_month.loc[(data_month['Month']=='Aug'),['Record ID']].count()
    x9=data_month.loc[(data_month['Month']=='Sep'),['Record ID']].count()
    x10=data_month.loc[(data_month['Month']=='Oct'),['Record ID']].count()
    x11=data_month.loc[(data_month['Month']=='Nov'),['Record ID']].count()
    x12=data_month.loc[(data_month['Month']=='Dec'),['Record ID']].count()
    x13=total_rows-
(countjan+countfeb+countmar+countapr+countmay+countjun+countjul+countaug+countsep+co
untocto+countnov+countdec)




    per1=(countjan/total_rows)*100 #percentage birdstrike in mumbai
    per2=(countfeb/total_rows)*100 #percentage birdstrike in mumbai
    per3=(countmar/total_rows)*100 #percentage overseas birdstrike
    per4=(countapr/total_rows)*100
    per5=(countmay/total_rows)*100
    per6=(countjun/total_rows)*100
    per7=(countjul/total_rows)*100
    per8=(countaug/total_rows)*100
    per9=(countsep/total_rows)*100
    per10=(countocto/total_rows)*100
    per11=(countnov/total_rows)*100
    per12=(countdec/total_rows)*100

    x = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August',
'September', 'October', 'November', 'December','No information']
    popularity = [x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13]
    x_pos = [i for i, _ in enumerate(x)]

    fig, ax = plt.subplots()
    rects1 = ax.bar(x_pos, popularity, color='yellow')
    plt.xlabel("Status of remains collected")
    plt.ylabel("Number of Bird Strikes")
    plt.title("Whether remains of birds were collected ??")
    plt.xticks(x_pos, x,rotation=65)
    # Turn on the grid
    plt.minorticks_on()
    plt.grid(which='major', linewidth='0.5', color='orange')
    # Customize the minor grid
    plt.grid(which='minor', linestyle=':', linewidth='0.5', color='orange')
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,'%f' %
float(height),ha='center', va='bottom')
    autolabel(rects1)

    plt.show()


    show()
    canvas = FigureCanvasTkAgg(fig, screen10)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)
```

```
    button = Button(screen10, text="Quit", command=screen10.destroy)
    button.pack(side=BOTTOM)

def narenocr():
    import pytesseract
    pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-
OCR\tesseract.exe'

    try:
        from PIL import Image
    except ImportError:
        import Image



    def ocr_core(filename):
        """
        This function will handle the core OCR processing of images.
        """
        text = pytesseract.image_to_string(Image.open(filename))  # We'll use
Pillow's Image class to open the image and pytesseract to detect the string in the
image
        return text

    print(ocr_core('narentest123.png'))


def misgeneration():
    data_naren79=pd.read_csv("bird1.csv")
    data_naren89=pd.read_csv("bird1.csv")
    data_naren13=data_naren79[(data_naren79['Airport: Name']=='BOMBAY/MUMBAI')]
    data_naren14=data_naren79[(data_naren89['Effect: Indicated Damage']=='Caused
damage')]


    export_csv = data_naren13.to_csv (r'Mumbai Airport.csv', index = None,
header=True)
    export_csv = data_naren14.to_csv (r'Damage Caused.csv', index = None,
header=True)


def question_14() :
    global screen14, category_variable, droplist_app, droplist_category
    screen14 = Tk()
    screen14.title('Question-14')
    adjustWindow(screen14)
    category_variable = StringVar(screen14)
    apps_list_1 = data_rohan['Name']
    apps_1 = []
    required_apps_from_1 = []
    category_list_1 = data_rohan['Category']
    categories = []
    required_category_from_1 = []
    required_unique_categories = []
    apps_list_2 = data_arnav['Name']
    apps_2 = []
    apps_unique_2 = []
    required_unique_apps = []
    for cat in category_list_1 :
        categories.append(cat)
    for app in apps_list_2 :
        if app not in apps_unique_2 :
            apps_unique_2.append(app)
        apps_2.append(app)
    for app in apps_list_1 :
        apps_1.append(app)
    for app in apps_unique_2 :
```

```
        for i in range(len(apps_1)) :
            if app == apps_1[i] :
                required_category_from_1.append(categories[i])
                required_apps_from_1.append(app)
    for each_category in required_category_from_1 :
        if each_category not in required_unique_categories :
            required_unique_categories.append(each_category)
    for element in required_unique_categories :
        dummy_apps = []
        for i in range(len(required_category_from_1)) :
            if element == required_category_from_1[i] and required_apps_from_1[i]
not in dummy_apps:
                dummy_apps.append(required_apps_from_1[i])
        required_unique_apps.append(dummy_apps)
    Label(screen14, text= 'Category', fg='black').pack()
    category_variable.set('--Select Category--')
    droplist_category = OptionMenu(screen14, category_variable,
*required_unique_categories)
    droplist_category.config(anchor='center', width=35)
    droplist_category.pack()
    button1 = Button(screen14, text="Tap", command= lambda:
display_apps(required_unique_categories, required_unique_apps))
    button1.pack()
    button = Button(screen14, text="Quit", command=screen14.destroy)
    button.pack(side=BOTTOM)

def display_reviews(app_list, apps, sentiments, reviews) :
    global caught_app_variable
    caught_app_variable = StringVar()
    caught_app_variable = app_variable.get()
    if caught_app_variable != "--Select Name--" :
        positive_text.delete('1.0', END)
        negative_text.delete('1.0', END)
        neutral_text.delete('1.0', END)
        scroll=Scrollbar(positive_text)
        positive_text.configure(state='normal')
        negative_text.configure(state='normal')
        neutral_text.configure(state='normal')
        positive_list = []
        negative_list = []
        neutral_list = []
        positive = 0
        negative = 0
        neutral = 0
        for app in app_list :
            for i in range(len(apps)) :
                if caught_app_variable == apps[i] and sentiments[i] == 'Positive'
and reviews[i] not in positive_list:
                    positive += 1
                    positive_text.insert(END, str(positive)+'-->
'+reviews[i]+'\n\n')
                    positive_list.append(reviews[i])
                if caught_app_variable == apps[i] and sentiments[i] == 'Negative'
and reviews[i] not in negative_list:
                    negative += 1
                    negative_text.insert(END, str(negative)+'-->
'+reviews[i]+'\n\n')
                    negative_list.append(reviews[i])
                if caught_app_variable == apps[i] and sentiments[i] == 'Neutral' and
reviews[i] not in neutral_list:
                    neutral += 1
                    neutral_text.insert(END, str(neutral)+'--> '+reviews[i]+'\n\n')
                    neutral_list.append(reviews[i])
    else :
        messagebox.showerror("Error", "Please select any name from the drop list!",
parent=screen14_1)
    positive_text.configure(yscrollcommand=scroll.set)
```

```
    negative_text.configure(yscrollcommand=scroll.set)
    neutral_text.configure(yscrollcommand=scroll.set)

def display_apps(required_unique_categories, required_unique_apps) :
    global screen14_1, app_variable, caught_category_variable, positive_text,
negative_text, neutral_text, apps_to_display
    caught_category_variable = StringVar()
    caught_category_variable = category_variable.get()
    if caught_category_variable == "--Select Category--" :
        messagebox.showerror("Error", "Please select any category from the drop
list!", parent=screen14)
    else :
        screen14_1=  Tk()
        screen14_1.title('Question-14.1')
        adjustWindow(screen14_1)
        app_variable = StringVar(screen14_1)
        apps_2 = []
        apps_list_2 = data_arnav['Name']
        sentiments_list = data_arnav['Sentiment']
        sentiments = []
        reviews_list = data_arnav['Translated_Review']
        reviews= []
        for review in reviews_list :
            reviews.append(review)
        for sentiment in sentiments_list :
            sentiments.append(sentiment)
        for app in apps_list_2 :
            apps_2.append(app)
        for cat in range(len(required_unique_categories)) :
            if required_unique_categories[cat] == caught_category_variable :
                apps_to_display = required_unique_apps[cat]
                break
        Label(screen14_1, text= 'Name', fg='black').pack()
        droplist = OptionMenu(screen14_1, app_variable, *apps_to_display)
        droplist.config(anchor='center', width=35)
        app_variable.set('--Select Name--')
        droplist.pack()
        button2 = Button(screen14_1, text="Watch", command= lambda:
display_reviews(apps_to_display, apps_2, sentiments, reviews))
        button2.pack()
        Label(screen14_1, text= 'Positive', fg='black').pack()
        positive_text = Text(screen14_1, height=9, width=100)
        positive_text.pack()
        positive_text.configure(state='disabled')
        Label(screen14_1, text= 'Negative', fg='black').pack()
        negative_text = Text(screen14_1, height=9, width=100)
        negative_text.pack()
        negative_text.configure(state='disabled')
        Label(screen14_1, text= 'Neutral', fg='black').pack()
        neutral_text = Text(screen14_1, height=9, width=100)
        neutral_text.pack()
        neutral_text.configure(state='disabled')
        button = Button(screen14_1, text="Quit", command=screen14_1.destroy)
        button.pack(side=BOTTOM)


def question_12():
    global screen12
    screen12 = Tk()
    screen12.title('Question-12')
    adjustWindow(screen12)

    p1=data_aviate['Aircraft.Category']

    p=0
    p1list=[]
    for i in p1:
```

```
        if (i=='Airplane'):
            p1list.append(0)
        else:
            p=0
    x1=len(p1list)



    q=0
    p2list=[]
    for j in p1:
        if (j=='Helicopter'):
            p2list.append(0)
        else:
            q=0

    x2=len(p2list)



    total_rows=len(data_aviate.axes[0])

    unknown=(total_rows-(x1+x2))

    x = ['Aircraft','Helicopter','No information']

    popularity = [x1,x2,unknown]
    x_pos = [i for i, _ in enumerate(x)]

    fig, ax = plt.subplots()
    rects1 = ax.bar(x_pos, popularity, color='b')
    plt.xlabel("Type of Aerial Vehicle")
    plt.title("Count of Aerial Vehicles")
    plt.xticks(x_pos, x)
        # Turn on the grid
    plt.minorticks_on()
    plt.grid(which='major', linewidth='0.5', color='red')
        # Customize the minor grid
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,'%d' %
float(height),ha='center', va='bottom')
    autolabel(rects1)

    plt.show()

    show()
    canvas = FigureCanvasTkAgg(fig, screen12)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)
    button = Button(screen12, text="Quit", command=screen12.destroy)
    button.pack(side=BOTTOM)

def question_15():
    global screen12
    screen12 = Tk()
    screen12.title('Question-12')
    adjustWindow1(screen12)

    p1=data_aviate['Injury.Severity']

    p=0
    p1list=[]
    for i in p1:
        if (i=='Non-Fatal'):
            p1list.append(0)
```

```
        else:
            p=0
    x1=len(p1list)



    q=0
    p2list=[]
    for j in p1:
        if (j=='Unavailable'):
            p2list.append(0)
        else:
            q=0

    x2=len(p2list)



    total_rows=len(data_aviate.axes[0])

    unknown=(total_rows-(x1+x2))

    x = ['Non-Fatal','Unavailable','Fatal']

    popularity = [x1,x2,unknown]
    x_pos = [i for i, _ in enumerate(x)]

    fig, ax = plt.subplots()
    rects1 = ax.bar(x_pos, popularity, color='y')
    plt.xlabel("Type of Incident")
    plt.title("Count of Incidents")
    plt.xticks(x_pos, x)
        # Turn on the grid
    plt.minorticks_on()
    plt.grid(which='major', linewidth='0.5', color='black')
        # Customize the minor grid
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,'%d' %
float(height),ha='center', va='bottom')
    autolabel(rects1)

    total_rows=len(data_aviate.axes[0])

    per1=(x1/total_rows)*100 #percentage birdstrike in mumbai
    print(per1)

    per2=(x2/total_rows)*100 #percentage birdstrike in mumbai
    print(per2)

    per3=(unknown/total_rows)*100 #percentage birdstrike in mumbai
    print(per3)

    Label(screen12,text="Percentage of Fatal Incidents",fg = "black",bg =
"white",font = "Verdana 10 bold").place(x=10,y=0)
    Label(screen12,text=per1,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=45,y=20)
    Label(screen12,text="Percentage data not available",fg = "black",bg =
"white",font = "Verdana 10 bold").place(x=282,y=0)
    Label(screen12,text=per2,fg = "black",bg = "white",font = "Verdana 10
bold").place(x=400,y=20)
    Label(screen12,text="Percentage Fatal accidents",fg = "black",bg = "white",font
= "Verdana 10 bold").pack()
    Label(screen12,text=per3,fg = "black",bg = "white",font = "Verdana 10
bold").pack()
```

```
    plt.show()

    show()
    canvas = FigureCanvasTkAgg(fig, screen12)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)
    button = Button(screen12, text="Quit", command=screen12.destroy)
    button.pack(side=BOTTOM)

def display_relation(left_bar, right_bar) :
    global screen13_1
    screen13_1 = Tk()
    screen13_1.title('Question-13.1')
    adjustWindow(screen13_1)
    left_bar = array(left_bar)
    left_bar = left_bar.reshape(-1,1)
    right_bar = array(right_bar)
    right_bar = right_bar.reshape(-1,1)
    reg = LinearRegression()
    reg.fit(left_bar, right_bar)
    relation_predictions = reg.predict(left_bar)
    X2 = add_constant(left_bar)
    est = OLS(right_bar,X2)
    est2 = est.fit()
    f, ax = subplots(figsize=(9, 7))
    Label(screen13_1, text= 'Linear Model : Y = {:.5}X + {:.5}'.format(reg.coef_[0]
[0],reg.intercept_[0]), fg='black').pack()
    Label(screen13_1, text= 'Accuracy : {}'.format(est2.rsquared),
fg='black').pack()
    scatter(left_bar, right_bar, c='black')
    plot(left_bar,relation_predictions,c='red',linewidth=2)
    xlabel('Sentiment Polarity')
    ylabel('Sentiment Subjectivity')
    show()
    canvas = FigureCanvasTkAgg(f, screen13_1)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)
    button = Button(screen13_1, text="Quit", command=screen13_1.destroy)
    button.pack(side=BOTTOM)

def question_13() :
    global screen13
    screen13 = Tk()
    screen13.title('Question-13')
    adjustWindow(screen13)
    sentiment_polarity = []
    sentiment_subjectivity = []
    unique_apps_list = []
    apps = []
    sum_sp = 0
    sum_ss = 0
    sentiment_polarity_list = data_priya['Sentiment_Polarity']
    sentiment_subjectivity_list = data_priya['Sentiment_Subjectivity']
    apps_list = data_priya['App']
    for app in apps_list :
        if app not in unique_apps_list :
            unique_apps_list.append(app)
        apps.append(app)
    for sp in sentiment_polarity_list :
        sentiment_polarity.append(sp)
    for ss in sentiment_subjectivity_list :
        sentiment_subjectivity.append(ss)
    for i in range(len(apps)) :
        sum_sp += sentiment_polarity[i]
        sum_ss += sentiment_subjectivity[i]
    ss_sp_list = ['Sentiment-Subjectivity', 'Sentiment-Polarity']
```

```
    ss_sp_sum = [sum_ss, sum_sp]
    #set(style="darkgrid")
    f, ax = subplots(figsize=(9, 7))
    Label(screen13, text= 'For value = 1 of sentiment polarity, approximate value of
sentiment subjectivity: {}'.format(round(sum_ss / sum_sp)), fg='black').pack()
    rects = bar(ss_sp_list, ss_sp_sum, color=['red', 'blue'])
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 1.01*height,
                '%.1f' % float(height),
        ha='center', va='bottom')
    xlabel('Sentiments')
    ylabel('Values')
    show()
    canvas = FigureCanvasTkAgg(f, screen13)
    canvas.draw()
    canvas.get_tk_widget().pack(side = TOP, fill = BOTH, expand = True)
    button = Button(screen13, text="Quit", command=screen13.destroy)
    button.pack(side=BOTTOM)


main_screen()
```