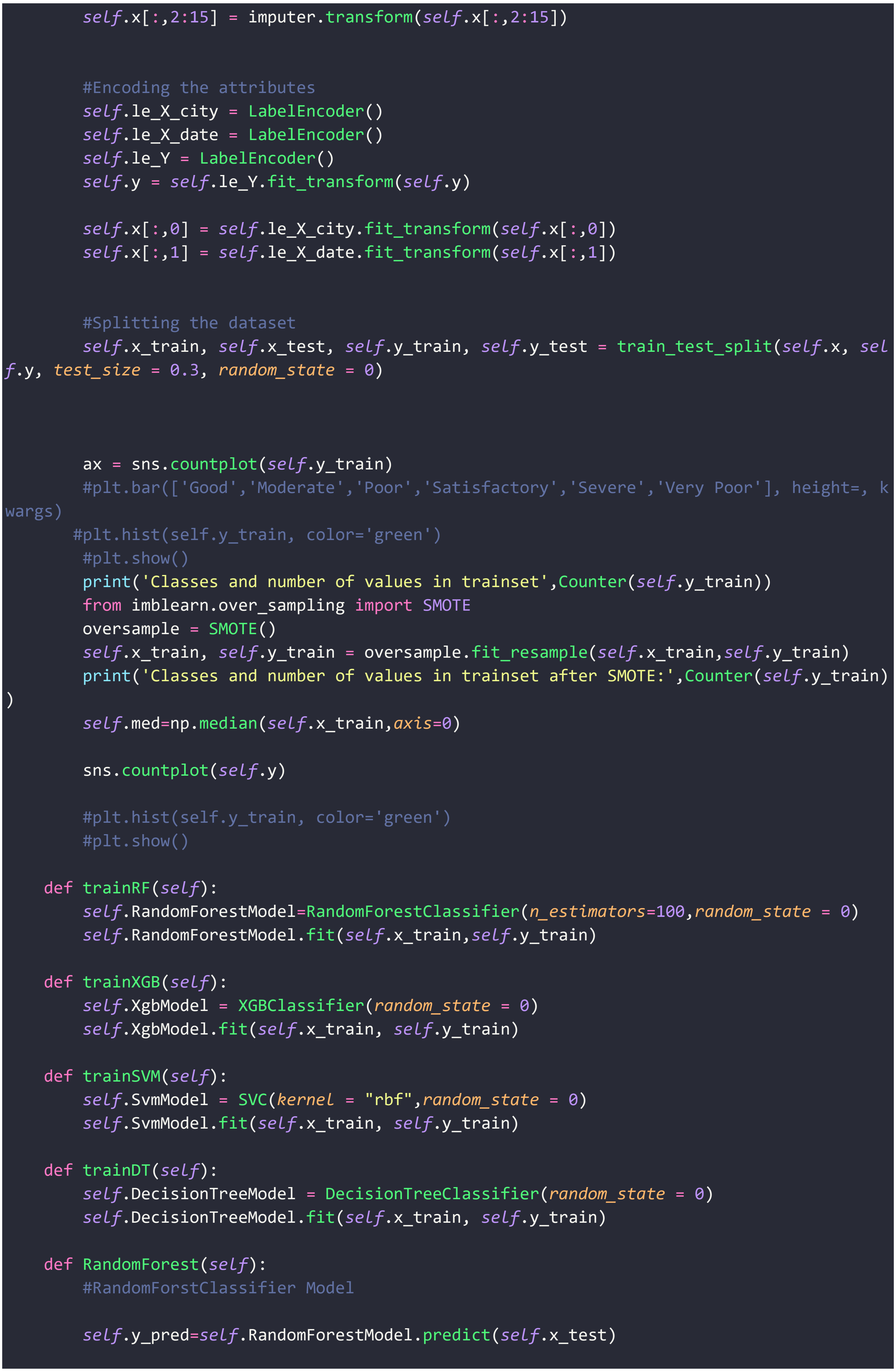
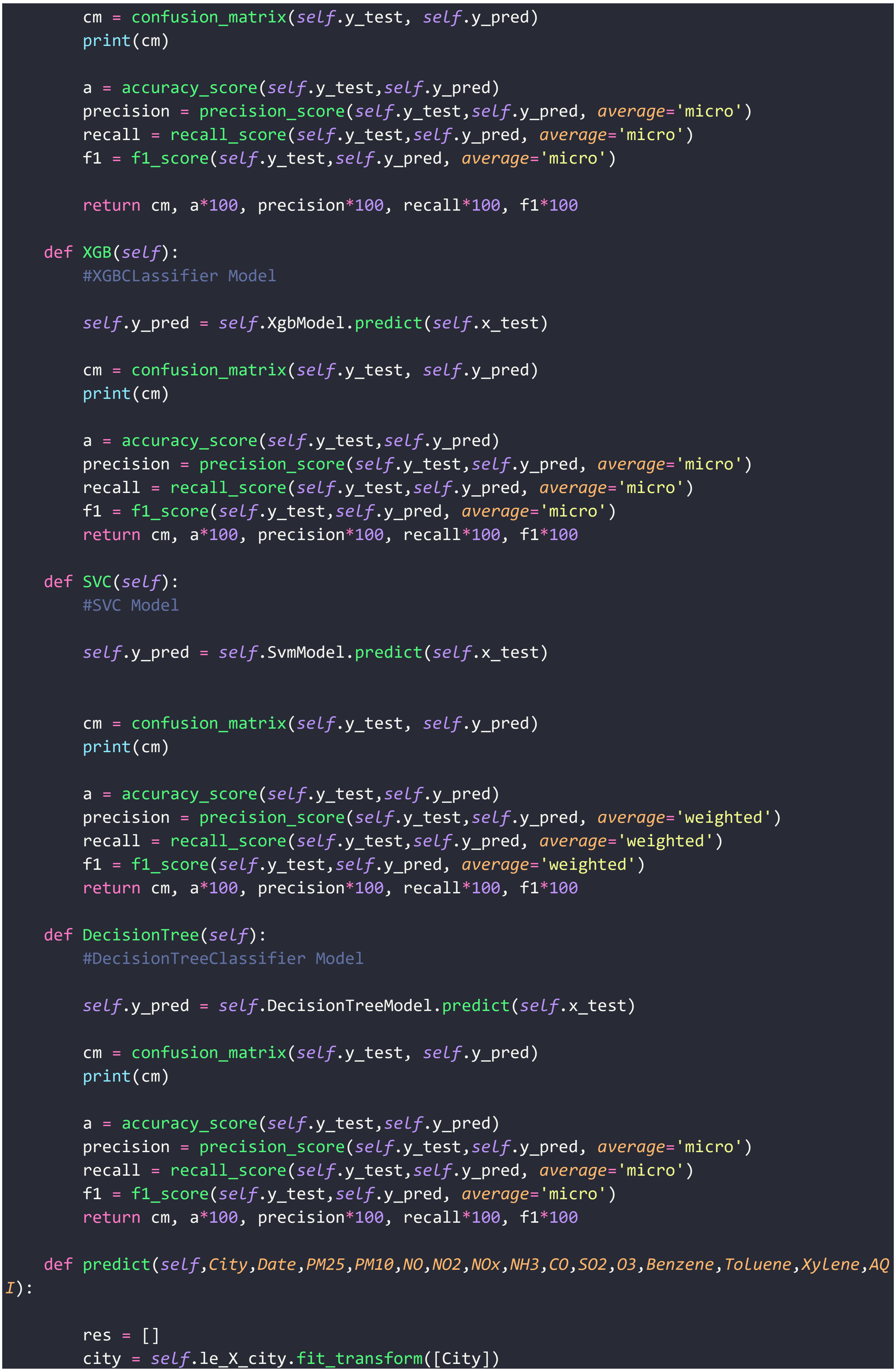
Project.py File

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from xgboost import XGBClassifier


class AirQuality:

    dataset = ""
    x = ""
    y = ""
    x_train = ""
    x_test = ""
    y_train = ""
    y_test = ""
    RandomForestModel = ""
    XgbModel = ""
    SvmModel = ""
    DecisionTreeModel = ""
    le_X_city = ""
    le_X_date = ""
    le_Y = ""


    def readCsv(self, file_name):

        #Importing the datset
        self.dataset = pd.read_csv(file_name);

        self.dataset.dropna(axis=0, subset = ["Air_quality", "Xylene", "AQI", "Toluene",
                                         "Benzene", "O3", "SO2", "CO", "NH3", "NOx",
                                         "NO2", "PM10", "PM2.5", "NO"], how = 'all', inplac
e= True)
        self.dataset.dropna(subset = ["Air_quality"], inplace=True)
#         print("asasd")

        self.x = self.dataset.iloc[:, :-1].values
        self.y = self.dataset.iloc[:,15].values

        #Filling the missing values
        imputer = SimpleImputer(missing_values = np.nan, strategy = 'median')
        imputer = imputer.fit(self.x[:,2:15])
```



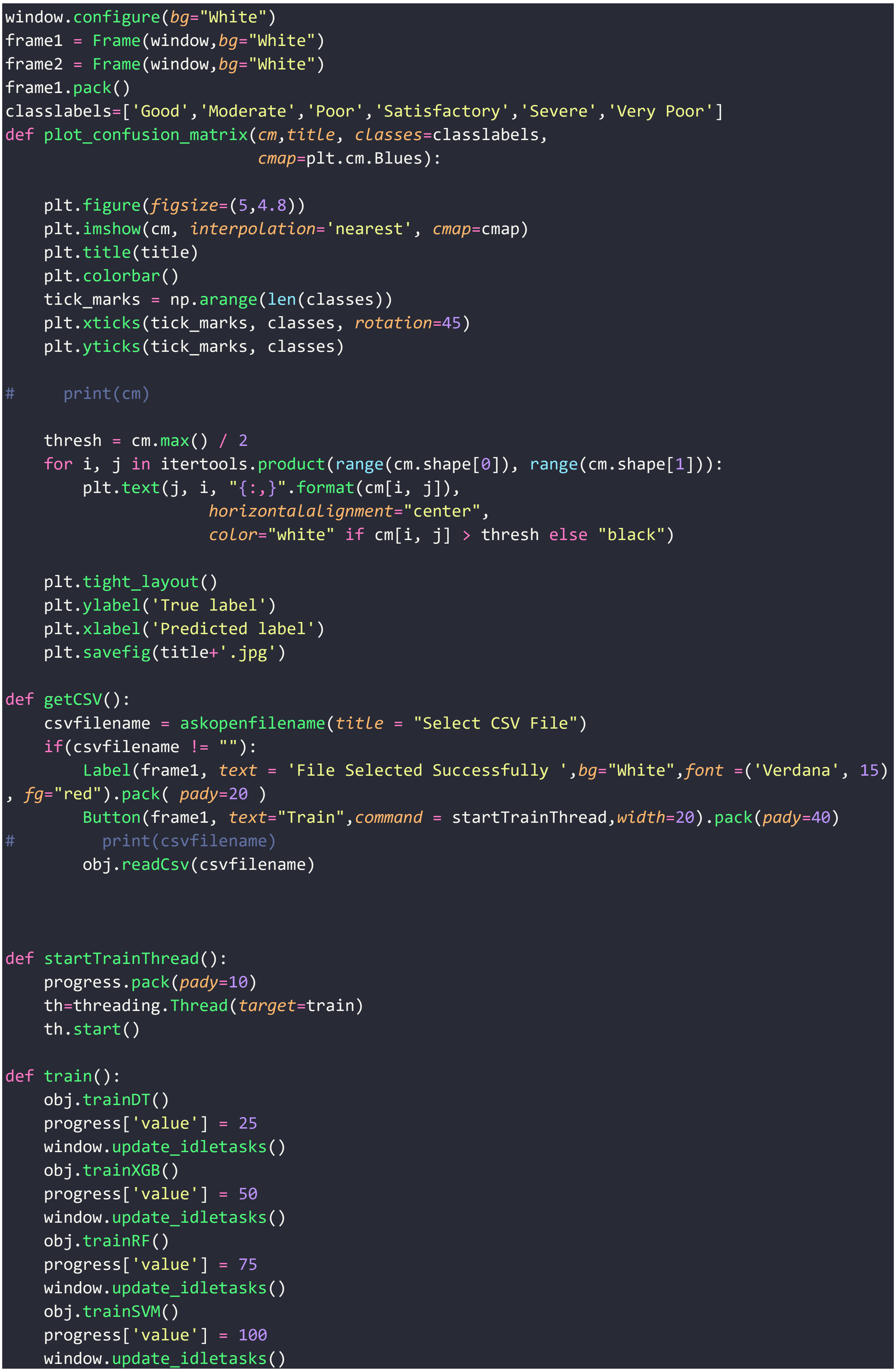
```
        self.x[:,2:15] = imputer.transform(self.x[:,2:15])


        #Encoding the attributes
        self.le_X_city = LabelEncoder()
        self.le_X_date = LabelEncoder()
        self.le_Y = LabelEncoder()
        self.y = self.le_Y.fit_transform(self.y)

        self.x[:,0] = self.le_X_city.fit_transform(self.x[:,0])
        self.x[:,1] = self.le_X_date.fit_transform(self.x[:,1])


        #Splitting the dataset
        self.x_train, self.x_test, self.y_train, self.y_test = train_test_split(self.x, self.y, test_size = 0.3, random_state = 0)



        ax = sns.countplot(self.y_train)
        #plt.bar(['Good','Moderate','Poor','Satisfactory','Severe','Very Poor'], height=, kwargs)
       #plt.hist(self.y_train, color='green')
        #plt.show()
        print('Classes and number of values in trainset',Counter(self.y_train))
        from imblearn.over_sampling import SMOTE
        oversample = SMOTE()
        self.x_train, self.y_train = oversample.fit_resample(self.x_train,self.y_train)
        print('Classes and number of values in trainset after SMOTE:',Counter(self.y_train))
        self.med=np.median(self.x_train,axis=0)

        sns.countplot(self.y)

        #plt.hist(self.y_train, color='green')
        #plt.show()

    def trainRF(self):
        self.RandomForestModel=RandomForestClassifier(n_estimators=100,random_state = 0)
        self.RandomForestModel.fit(self.x_train,self.y_train)

    def trainXGB(self):
        self.XgbModel = XGBClassifier(random_state = 0)
        self.XgbModel.fit(self.x_train, self.y_train)

    def trainSVM(self):
        self.SvmModel = SVC(kernel = "rbf",random_state = 0)
        self.SvmModel.fit(self.x_train, self.y_train)

    def trainDT(self):
        self.DecisionTreeModel = DecisionTreeClassifier(random_state = 0)
        self.DecisionTreeModel.fit(self.x_train, self.y_train)

    def RandomForest(self):
        #RandomForstClassifier Model

        self.y_pred=self.RandomForestModel.predict(self.x_test)
```



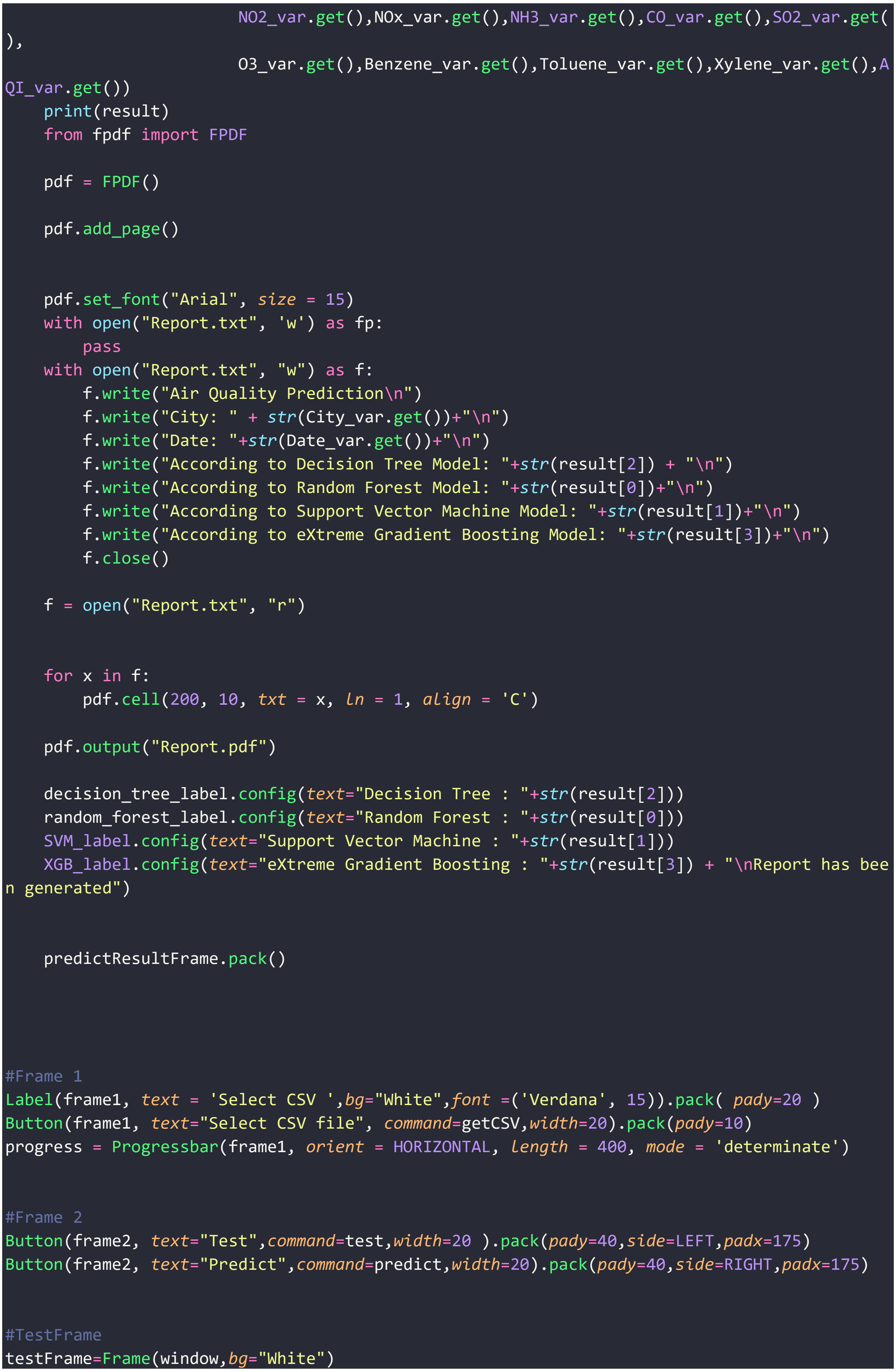
```
        cm = confusion_matrix(self.y_test, self.y_pred)
        print(cm)

        a = accuracy_score(self.y_test,self.y_pred)
        precision = precision_score(self.y_test,self.y_pred, average='micro')
        recall = recall_score(self.y_test,self.y_pred, average='micro')
        f1 = f1_score(self.y_test,self.y_pred, average='micro')

        return cm, a*100, precision*100, recall*100, f1*100

    def XGB(self):
        #XGBCLassifier Model

        self.y_pred = self.XgbModel.predict(self.x_test)

        cm = confusion_matrix(self.y_test, self.y_pred)
        print(cm)

        a = accuracy_score(self.y_test,self.y_pred)
        precision = precision_score(self.y_test,self.y_pred, average='micro')
        recall = recall_score(self.y_test,self.y_pred, average='micro')
        f1 = f1_score(self.y_test,self.y_pred, average='micro')
        return cm, a*100, precision*100, recall*100, f1*100

    def SVC(self):
        #SVC Model

        self.y_pred = self.SvmModel.predict(self.x_test)


        cm = confusion_matrix(self.y_test, self.y_pred)
        print(cm)

        a = accuracy_score(self.y_test,self.y_pred)
        precision = precision_score(self.y_test,self.y_pred, average='weighted')
        recall = recall_score(self.y_test,self.y_pred, average='weighted')
        f1 = f1_score(self.y_test,self.y_pred, average='weighted')
        return cm, a*100, precision*100, recall*100, f1*100

    def DecisionTree(self):
        #DecisionTreeClassifier Model

        self.y_pred = self.DecisionTreeModel.predict(self.x_test)

        cm = confusion_matrix(self.y_test, self.y_pred)
        print(cm)

        a = accuracy_score(self.y_test,self.y_pred)
        precision = precision_score(self.y_test,self.y_pred, average='micro')
        recall = recall_score(self.y_test,self.y_pred, average='micro')
        f1 = f1_score(self.y_test,self.y_pred, average='micro')
        return cm, a*100, precision*100, recall*100, f1*100

    def predict(self,City,Date,PM25,PM10,NO,NO2,NOx,NH3,CO,SO2,O3,Benzene,Toluene,Xylene,AQ
I):

        res = []
        city = self.le_X_city.fit_transform([City])
        date = self.le_X_date.fit_transform([Date])

        if(not PM25):
            PM25val=self.med[2]
        else:
            PM25val=float(PM25)
        if(not PM10):
            PM10val=self.med[3]
        else:
            PM10val=float(PM10)
        if(not NO):
            NOval=self.med[4]
        else:
            NOval=float(NO)
        if(not NO2):
            NO2val=self.med[5]
        else:
            NO2val=float(NO2)
        if(not NOx):
            NOxval=self.med[6]
        else:
            NOxval=float(NOx)
        if(not NH3):
            NH3val=self.med[7]
        else:
            NH3val=float(NH3)
        if(not CO):
            COval=self.med[8]
        else:
            COval=float(CO)
        if(not SO2):
            SO2val=self.med[9]
        else:
            SO2val=float(SO2)
        if(not O3):
            O3val=self.med[10]
        else:
            O3val=float(O3)
        if(not Benzene):
            Benzeneval=self.med[11]
        else:
            Benzeneval=float(Benzene)
        if(not Toluene):
            Tolueneval=self.med[12]
        else:
            Tolueneval=float(Toluene)
        if(not Xylene):
            Xyleneval=self.med[13]
        else:
            Xyleneval=float(Xylene)
        if(not AQI):
            AQIval=self.med[14]
        else:
            AQIval=float(AQI)

        ls = [city[0] ,date[0], PM25val, PM10val, NOval, NO2val, NOxval, NH3val, COval, SO2
val, O3val, Benzeneval, Tolueneval, Xyleneval, AQIval]
        lst = [];
        lst.append(ls);

        temp = self.le_Y.inverse_transform(self.RandomForestModel.predict(lst))
        temp = temp.tolist()
        res.append(temp[0])

        temp = self.le_Y.inverse_transform(self.SvmModel.predict(lst))
        temp = temp.tolist()
        res.append(temp[0])

        temp = self.le_Y.inverse_transform(self.DecisionTreeModel.predict(lst))
        temp = temp.tolist()
        res.append(temp[0])

#         print(lst)
        ll=np.array(lst).reshape(1,-1)
#         print(ll)
        temp = self.le_Y.inverse_transform(self.XgbModel.predict(ll))
        temp = temp.tolist()
        res.append(temp[0])


        return res
```

App.py File

```
from Project import AirQuality
import matplotlib.pyplot as plt
import itertools
import numpy as np
from PIL import ImageTk, Image
import threading
import time
from tkinter import *
from tkinter.ttk import Progressbar
from tkinter.filedialog import askopenfilename
obj = AirQuality()
window = Tk()
window.title("Air Quality Predictor")
width, height = window.winfo_screenwidth(), window.winfo_screenheight()
window.geometry('%dx%d+0+0' % (width,height))
```

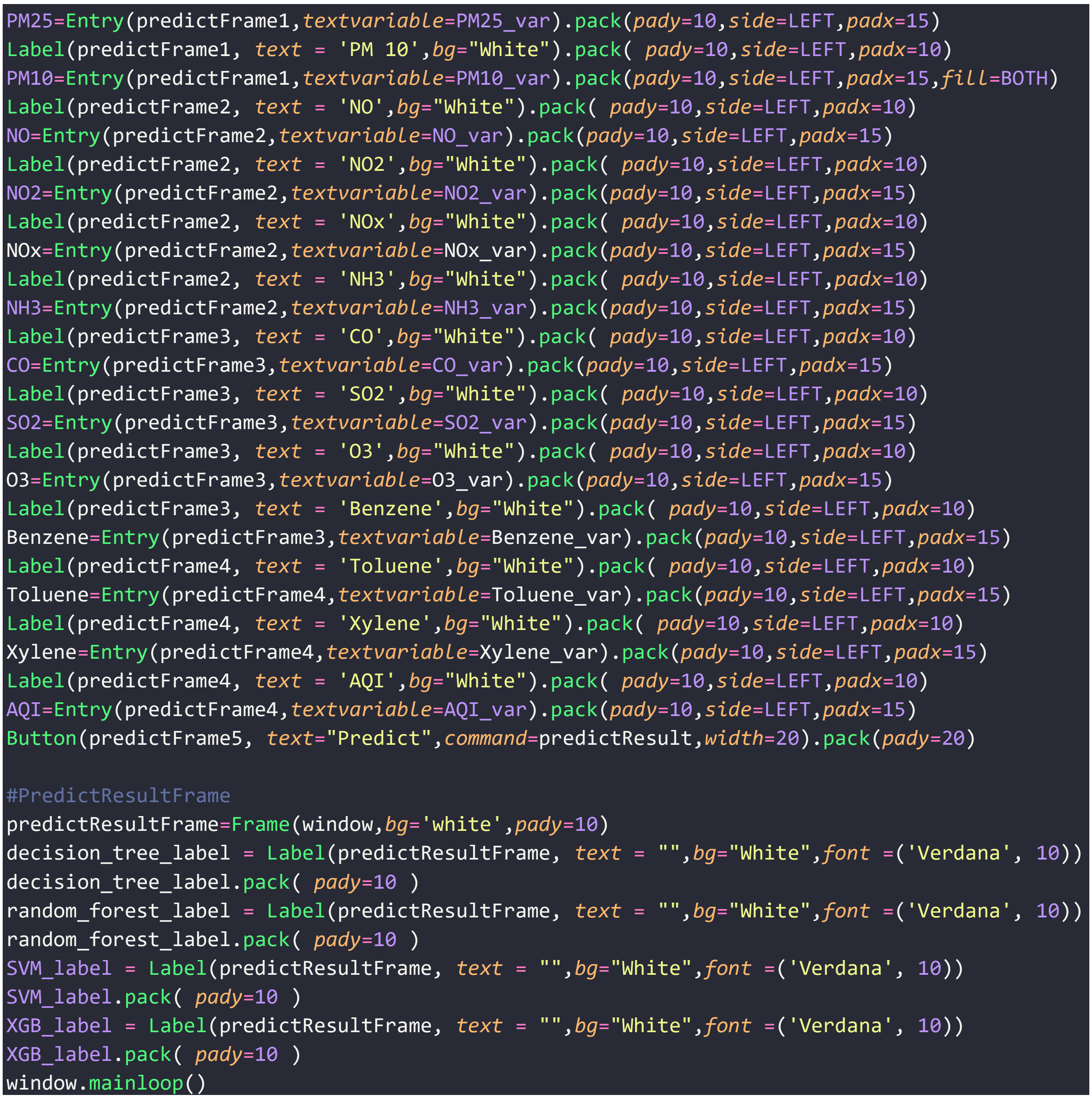


```
window.configure(bg="White")
frame1 = Frame(window,bg="White")
frame2 = Frame(window,bg="White")
frame1.pack()
classlabels=['Good','Moderate','Poor','Satisfactory','Severe','Very Poor']
def plot_confusion_matrix(cm,title, classes=classlabels,
                          cmap=plt.cm.Blues):

    plt.figure(figsize=(5,4.8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

#     print(cm)

    thresh = cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.savefig(title+'.jpg')

def getCSV():
    csvfilename = askopenfilename(title = "Select CSV File")
    if(csvfilename != ""):
        Label(frame1, text = 'File Selected Successfully ',bg="White",font =('Verdana', 15)
, fg="red").pack( pady=20 )
        Button(frame1, text="Train",command = startTrainThread,width=20).pack(pady=40)
#         print(csvfilename)
        obj.readCsv(csvfilename)



def startTrainThread():
    progress.pack(pady=10)
    th=threading.Thread(target=train)
    th.start()

def train():
    obj.trainDT()
    progress['value'] = 25
    window.update_idletasks()
    obj.trainXGB()
    progress['value'] = 50
    window.update_idletasks()
    obj.trainRF()
    progress['value'] = 75
    window.update_idletasks()
    obj.trainSVM()
    progress['value'] = 100
    window.update_idletasks()
    time.sleep(1)
    frame1.destroy()
    frame2.pack()

def test():
    predictFrame1.pack_forget()
    predictFrame2.pack_forget()
    predictFrame3.pack_forget()
    predictFrame4.pack_forget()
    predictFrame5.pack_forget()
    predictResultFrame.pack_forget()
    resultFrame.pack_forget()
    testFrame.pack()
    testPanel.pack()

def testClassifier():
    selector = var.get()
    if selector=='Random Forest':
        cm,a,precision,recall,f1=obj.RandomForest()
    elif selector=='Decision Tree':
        cm,a,precision,recall,f1=obj.DecisionTree()
    elif selector=='Support Vector Machine':
        cm,a,precision,recall,f1=obj.SVC()
    else:
        cm,a,precision,recall,f1=obj.XGB()
    acc_label.config(text="Accuracy :" +str(a))
    precision_label.config(text="Precision : "+str(precision))
    recall_label.config(text="Recall : "+str(recall))
    f1score_label.config(text="F1 Score : "+str(f1))
    plot_confusion_matrix(cm,title=selector)
    from PIL import ImageTk, Image
    im = Image.open(selector+'.jpg')
#     newsize=(432,350)
#     im = im.resize(newsize)
    img = ImageTk.PhotoImage(im)
    cm_label.config(image=img)
    cm_label.image=img
    resultFrame.pack()

#     print(cm,a,precision,recall,f1)




def predict():
    testFrame.pack_forget()
    testPanel.pack_forget()
    resultFrame.pack_forget()
    predictResultFrame.pack_forget()
    predictFrame1.pack()
    predictFrame2.pack()
    predictFrame3.pack()
    predictFrame4.pack()
    predictFrame5.pack()

def predictResult():
    result = obj.predict(City_var.get(),Date_var.get(),PM25_var.get(),PM10_var.get(),NO_var
.get(),
```



```
                        NO2_var.get(),NOx_var.get(),NH3_var.get(),CO_var.get(),SO2_var.get(
),
                        O3_var.get(),Benzene_var.get(),Toluene_var.get(),Xylene_var.get(),A
QI_var.get())
    print(result)
    from fpdf import FPDF

    pdf = FPDF()

    pdf.add_page()


    pdf.set_font("Arial", size = 15)
    with open("Report.txt", 'w') as fp:
        pass
    with open("Report.txt", "w") as f:
        f.write("Air Quality Prediction\n")
        f.write("City: " + str(City_var.get())+"\n")
        f.write("Date: "+str(Date_var.get())+"\n")
        f.write("According to Decision Tree Model: "+str(result[2]) + "\n")
        f.write("According to Random Forest Model: "+str(result[0])+"\n")
        f.write("According to Support Vector Machine Model: "+str(result[1])+"\n")
        f.write("According to eXtreme Gradient Boosting Model: "+str(result[3])+"\n")
        f.close()

    f = open("Report.txt", "r")


    for x in f:
        pdf.cell(200, 10, txt = x, ln = 1, align = 'C')

    pdf.output("Report.pdf")

    decision_tree_label.config(text="Decision Tree : "+str(result[2]))
    random_forest_label.config(text="Random Forest : "+str(result[0]))
    SVM_label.config(text="Support Vector Machine : "+str(result[1]))
    XGB_label.config(text="eXtreme Gradient Boosting : "+str(result[3]) + "\nReport has bee
n generated")


    predictResultFrame.pack()





#Frame 1
Label(frame1, text = 'Select CSV ',bg="White",font =('Verdana', 15)).pack( pady=20 )
Button(frame1, text="Select CSV file", command=getCSV,width=20).pack(pady=10)
progress = Progressbar(frame1, orient = HORIZONTAL, length = 400, mode = 'determinate')


#Frame 2
Button(frame2, text="Test",command=test,width=20 ).pack(pady=40,side=LEFT,padx=175)
Button(frame2, text="Predict",command=predict,width=20).pack(pady=40,side=RIGHT,padx=175)


#TestFrame
testFrame=Frame(window,bg="White")
```

```
var = StringVar()
var.set('Random Forest')
choices = { 'Random Forest','Decision Tree','Support Vector Machine','eXtreme Gradient Boosting'}
popupMenu = OptionMenu(testFrame, var, *choices)
Label(testFrame, text = 'Testing ',bg="White",font =('Verdana', 15)).pack(pady=20)
Label(testFrame, text = 'Select Classifier',bg="White",font =('Verdana', 15)).pack( side=LEFT,padx=20,pady=10 )
popupMenu.pack(side=LEFT)

#TestPanel
testPanel=Frame(window,bg="White")
Button(testPanel, text="Test",command=testClassifier,width=20).pack(pady=20)

#TestResultFrame
resultFrame=Frame(window,bg='white',pady=10)
acc_label = Label(resultFrame, text = "",bg="White",font =('Verdana', 10))
acc_label.pack( pady=10 )
precision_label = Label(resultFrame, text = "",bg="White",font =('Verdana', 10))
precision_label.pack( pady=10 )
recall_label = Label(resultFrame, text = "",bg="White",font =('Verdana', 10))
recall_label.pack( pady=10 )
f1score_label = Label(resultFrame, text = "",bg="White",font =('Verdana', 10))
f1score_label.pack( pady=10 )
from PIL import ImageTk, Image
# img = ImageTk.PhotoImage(Image.open("mkbhd1.jpg"))
cm_label = Label(resultFrame)
# cm_label.image=img
cm_label.pack(pady=10)

#predictFrame
predictFrame1=Frame(window,bg="White",pady=10)
predictFrame2=Frame(window,bg="White",pady=10)
predictFrame3=Frame(window,bg="White",pady=10)
predictFrame4=Frame(window,bg="White",pady=10)
predictFrame5=Frame(window,bg="White",pady=10)

City_var = StringVar()
Date_var = StringVar()
PM25_var = StringVar()
PM10_var = StringVar()
NO_var = StringVar()
NO2_var = StringVar()
NOx_var = StringVar()
NH3_var = StringVar()
CO_var = StringVar()
SO2_var = StringVar()
O3_var = StringVar()
Benzene_var = StringVar()
Toluene_var = StringVar()
Xylene_var = StringVar()
AQI_var = StringVar()

Label(predictFrame1, text = 'City',bg="White").pack( pady=10,side=LEFT,padx=10)
City=Entry(predictFrame1,textvariable=City_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame1, text = 'Date',bg="White").pack( pady=10,side=LEFT,padx=10)
Date=Entry(predictFrame1,textvariable=Date_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame1, text = 'PM 2.5',bg="White").pack( pady=10,side=LEFT,padx=10)
```



```
PM25=Entry(predictFrame1,textvariable=PM25_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame1, text = 'PM 10',bg="White").pack( pady=10,side=LEFT,padx=10)
PM10=Entry(predictFrame1,textvariable=PM10_var).pack(pady=10,side=LEFT,padx=15,fill=BOTH)
Label(predictFrame2, text = 'NO',bg="White").pack( pady=10,side=LEFT,padx=10)
NO=Entry(predictFrame2,textvariable=NO_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame2, text = 'NO2',bg="White").pack( pady=10,side=LEFT,padx=10)
NO2=Entry(predictFrame2,textvariable=NO2_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame2, text = 'NOx',bg="White").pack( pady=10,side=LEFT,padx=10)
NOx=Entry(predictFrame2,textvariable=NOx_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame2, text = 'NH3',bg="White").pack( pady=10,side=LEFT,padx=10)
NH3=Entry(predictFrame2,textvariable=NH3_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame3, text = 'CO',bg="White").pack( pady=10,side=LEFT,padx=10)
CO=Entry(predictFrame3,textvariable=CO_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame3, text = 'SO2',bg="White").pack( pady=10,side=LEFT,padx=10)
SO2=Entry(predictFrame3,textvariable=SO2_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame3, text = 'O3',bg="White").pack( pady=10,side=LEFT,padx=10)
O3=Entry(predictFrame3,textvariable=O3_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame3, text = 'Benzene',bg="White").pack( pady=10,side=LEFT,padx=10)
Benzene=Entry(predictFrame3,textvariable=Benzene_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame4, text = 'Toluene',bg="White").pack( pady=10,side=LEFT,padx=10)
Toluene=Entry(predictFrame4,textvariable=Toluene_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame4, text = 'Xylene',bg="White").pack( pady=10,side=LEFT,padx=10)
Xylene=Entry(predictFrame4,textvariable=Xylene_var).pack(pady=10,side=LEFT,padx=15)
Label(predictFrame4, text = 'AQI',bg="White").pack( pady=10,side=LEFT,padx=10)
AQI=Entry(predictFrame4,textvariable=AQI_var).pack(pady=10,side=LEFT,padx=15)
Button(predictFrame5, text="Predict",command=predictResult,width=20).pack(pady=20)

#PredictResultFrame
predictResultFrame=Frame(window,bg='white',pady=10)
decision_tree_label = Label(predictResultFrame, text = "",bg="White",font =('Verdana', 10))
decision_tree_label.pack( pady=10 )
random_forest_label = Label(predictResultFrame, text = "",bg="White",font =('Verdana', 10))
random_forest_label.pack( pady=10 )
SVM_label = Label(predictResultFrame, text = "",bg="White",font =('Verdana', 10))
SVM_label.pack( pady=10 )
XGB_label = Label(predictResultFrame, text = "",bg="White",font =('Verdana', 10))
XGB_label.pack( pady=10 )
window.mainloop()
```