**import** pandas **as** pd  
**import** importlib  
**import** numpy **as** np  
**import** seaborn **as** sns  
**from** tkinter **import** \*  
**from** tkinter **import** Text, Tk  
**from** tkinter **import** messagebox  
**from** PIL **import** ImageTk  
**from** PIL **import** Image  
**import** os  
  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn **import** linear\_model, metrics  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.neighbors **import** KNeighborsClassifier  
**from** sklearn.tree **import** DecisionTreeClassifier  
**import** matplotlib.pyplot **as** plt  
**import** pylab **as** pl  
*# from sklearn.linear\_model import LogisticRegression***from** collections **import** Counter  
**import** matplotlib.pyplot **as** plt  
  
*#creating login window*rootLogin = Tk()  
rootLogin.geometry(**"500x400"**)  
rootLogin.configure(background=**'white'**)  
rootLogin.title(**"Attrition System Login"**)  
  
*#global variables for login input fields*user = StringVar()  
paswd=StringVar()  
  
*#global variables for main window fields*ln=StringVar()  
fn1 = StringVar()  
fn2=StringVar()  
fn3=StringVar()  
fn4=StringVar()  
fn5=StringVar()  
fn6=StringVar()  
fn7=StringVar()  
fn8=StringVar()  
fn9=StringVar()  
fn10=StringVar()  
  
*#getting data\_set of employees*data\_set = pd.read\_csv(**'Employee Attrition.csv'**)  
C = pd.DataFrame(data\_set)  
  
  
*#function which predicts attrition***def** getAttrition():  
  
 *#getting field values* entry\_1=ln.get()  
 entry\_2=fn1.get()  
 entry\_3=fn2.get()  
 entry\_4=fn3.get()  
 entry\_5=fn4.get()  
 entry\_6=fn5.get()  
 entry\_7=fn6.get()  
 entry\_8=fn7.get()  
 entry\_9=fn8.get()  
 entry\_10=fn9.get()  
  
 *#converting field value to float* ent\_1 = float(entry\_1)  
 ent\_2 = float(entry\_2)  
 ent\_3 = float(entry\_3)  
 ent\_4 = float(entry\_4)  
 ent\_5 = float(entry\_5)  
 ent\_6 = float(entry\_6)  
 ent\_7 = float(entry\_7)  
 ent\_8 = float(entry\_8)  
 ent\_9 = float(entry\_9)  
 ent\_10 = float(entry\_10)  
  
 *#getting label* Y = C.loc[:, **'Attrition'**]  
 *#X = C.loc[:, 'BusinessTravel':'YearWithCurrManager']  
  
 #getting features* X = C.loc[:,[**'Age'**,**'DailyRate'**,**'DistanceFromHome'**,**'EmployeeNumber'**,**'HourlyRate'**,**'MonthlyIncome'**,**'MonthlyRate'**,**'NumCompaniesWorked'**,**'TotalWorkingYears'**,**'OverTime'**]]  
  
 *#splitting data into testing and training 80% for testing and 20% for training* x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2)  
  
 print(**"Keys"** + str(data\_set.keys()))  
 *#x\_train = pd.get\_dummies(X\_train)  
 #x\_test = pd.get\_dummies(X\_test)  
 # print(x\_test);  
 #y\_train = pd.get\_dummies(Y\_train)  
 # y\_test = pd.get\_dummies(Y\_test)  
  
 #print(x\_train)  
 #print(Y\_test)  
  
 #predecition models* model = RandomForestClassifier()  
 *# model = linear\_model.LinearRegression()  
 # model = KNeighborsClassifier(n\_neighbors=3)  
 # model =DecisionTreeClassifier()  
  
 #training model* model.fit(x\_train, y\_train)  
 tr\_acc = []  
 ts\_acc = []  
  
 *#getting training and testing accuracy* ts\_acc.append(model.score(x\_test, y\_test))  
 tr\_acc.append(model.score(x\_train, y\_train))  
  
 *#prediction on input values* x\_new=[[ent\_1,ent\_2,ent\_3,ent\_4,ent\_5,ent\_6,ent\_7,ent\_8,ent\_9,ent\_10]]  
 predict = model.predict(x\_new)  
  
  
 print(**"Prediction"** + str(predict))  
 predc=str(predict)  
 print(**"Training Accuracy"** + str(tr\_acc))  
 print(**"Testing Accuracy"** + str(ts\_acc))  
  
 *#checking if attrition or not message will be displayed accordingly* **if** predc==**"['Yes']"**:  
 entry\_11=fn10.set( **"Employee will leave the company"**);  
 **elif** predc==**"['No']"**:  
 entry\_11 = fn10.set(**"Employee will not leave the company"**);  
  
*#feature selection function***def** feature():  
  
 Y = C.loc[:, **'Attrition'**]  
 X = C.loc[:, **'BusinessTravel'**:**'YearWithCurrManager'**]  
 X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2)  
  
 *# print("Keys" + str(data\_set.keys()))* x\_train = pd.get\_dummies(X\_train)  
 x\_test = pd.get\_dummies(X\_test)  
  
 y\_train = pd.get\_dummies(Y\_train)  
 y\_test = pd.get\_dummies(Y\_test)  
  
 *# print(x\_train)  
 # print(Y\_test)* model = RandomForestClassifier()  
 *# model = linear\_model.LinearRegression()  
 # model = KNeighborsClassifier(n\_neighbors=3)  
 # model =DecisionTreeClassifier()* model.fit(x\_train, y\_train)  
 tr\_acc = []  
 ts\_acc = []  
 ts\_acc.append(model.score(x\_test, y\_test))  
 tr\_acc.append(model.score(x\_train, y\_train))  
 predict = model.predict(x\_test)  
 *# print("Prediction" + str(predict))* print(**"Training Accuracy"** + str(tr\_acc[0]))  
 print(**"Testing Accuracy"** + str(ts\_acc[0]))  
 cm = metrics.confusion\_matrix(  
 y\_test.values.argmax(axis=1), predict.argmax(axis=1))  
  
 *# feature Importance* f\_imp = list(model.feature\_importances\_)  
 f\_imp.sort(reverse=**True**)  
 x\_train.columns  
 th = f\_imp[8]  
 th\_dict = {}  
 cols = x\_train.columns  
  
 **for** idx, i **in** enumerate(model.feature\_importances\_):  
 **if** i >= th:  
 th\_dict[cols[idx]] = (i \* 100)  
  
 sns.barplot(x=list(th\_dict.values()), y=list(th\_dict.keys()), orient=**'h'**)  
 plt.xlabel(**"Importance"**)  
 plt.ylabel(**"Features"**)  
 cols\_names = list(th\_dict.keys())  
 cols\_names  
 plt.show()  
  
*#function to display how many employees have left the company and how many stayed***def** attritionlabel():  
  
 data\_set = pd.read\_csv(**'Employee Attrition.csv'**)  
 C = pd.DataFrame(data\_set)  
  
 *#plotting the graph* y\_bar = np.array([C[C[**'Attrition'**] == **'No'**].shape[0]  
 , C[C[**'Attrition'**] == **'Yes'**].shape[0]])  
 x\_bar = [**'No (0)'**, **'Yes (1)'**] *# Bar Visualization* plt.bar(x\_bar, y\_bar,color=**'b'**)  
 plt.xlabel(**'Labels/Classes'**)  
 plt.ylabel(**'Number of Instances'**)  
 plt.title(**'Distribution of Labels/Classes in the Dataset'**)  
 plt.show()  
  
*#function to display number of female and male in the dataset***def** gender():  
  
 *#plotting the graph* y\_bar = np.array([C[C[**'Gender'**] == **'Female'**].shape[0]  
 , C[C[**'Gender'**] == **'Male'**].shape[0]])  
 x\_bar = [**'Female (0)'**, **'Male (1)'**] *# Bar Visualization* plt.bar(x\_bar, y\_bar,color=**'r'**)  
 plt.xlabel(**'Labels/Classes'**)  
 plt.ylabel(**'Number of Instances'**)  
 plt.title(**'Distribution of Labels/Classes in the Dataset'**)  
 plt.show()  
*#function to display marital status graph***def** marital():  
  
 *#plotting the graph* y\_bar = np.array([C[C[**'MaritalStatus'**] == **'Single'**].shape[0]  
 , C[C[**'MaritalStatus'**] == **'Married'**].shape[0],C[C[**'MaritalStatus'**] == **'Divorced'**].shape[0]],)  
 x\_bar = [**'Single (0)'**, **'Married (1)'**,**'Divorced(2)'**] *# Bar Visualization* plt.bar(x\_bar, y\_bar,color=**'g'**)  
 plt.xlabel(**'Labels/Classes'**)  
 plt.ylabel(**'Number of Instances'**)  
 plt.title(**'Distribution of Labels/Classes in the Dataset'**)  
 plt.show()  
  
*#plotting confusion matrix***def** confmatrix():  
  
 Y = C.loc[:, **'Attrition'**]  
 X = C.loc[:, **'BusinessTravel'**:**'YearWithCurrManager'**]  
 X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2)  
  
 print(**"Keys"** + str(data\_set.keys()))  
 x\_train = pd.get\_dummies(X\_train)  
 x\_test = pd.get\_dummies(X\_test)  
  
 y\_train = pd.get\_dummies(Y\_train)  
 y\_test = pd.get\_dummies(Y\_test)  
  
 print(x\_train)  
 *# print(Y\_test)* model = RandomForestClassifier()  
 *# model = KNeighborsClassifier(n\_neighbors=3)  
  
 # model =DecisionTreeClassifier()  
 # model = linear\_model.LinearRegression()* model.fit(x\_train, y\_train)  
  
 tr\_acc = []  
 ts\_acc = []  
  
 ts\_acc.append(model.score(x\_test, y\_test))  
 tr\_acc.append(model.score(x\_train, y\_train))  
 predict = model.predict(x\_test)  
 print(**"Prediction"** + str(predict))  
  
 print(**"Training Accuracy"** + str(tr\_acc))  
 print(**"Testing Accuracy"** + str(ts\_acc))  
 cm = metrics.confusion\_matrix(  
 y\_test.values.argmax(axis=1), predict.argmax(axis=1))  
  
 ax = plt.subplot()  
 sns.heatmap(cm, annot=**True**, ax=ax)  
 plt.title(**"Confusion Matrix Using Random Forest"**)  
 plt.show()  
  
*#function to get correlation between features and label***def** heatMap():  
  
 corr = C.corr()  
 plt.subplots(figsize=(12, 10))  
 sns.heatmap(corr, annot=**True**)  
 plt.title(**"Correlations HeatMap"**)  
 plt.show()  
  
*#function to exit the system***def** exitt():  
 exit()  
  
*#main window that will be displayed right after login***class** Window(Frame):  
 **def** \_\_init\_\_(self, master=**None**):  
 Frame.\_\_init\_\_(self, master)  
 self.master = master  
 self.pack(fill=BOTH, expand=1)  
  
 load = Image.open(**"dashboard bilal 5.jpg"**)  
 render = ImageTk.PhotoImage(load)  
 img = Label(self, image=render,width=**"1800"**, height=**"1000"**)  
 img.image = render  
 img.place(x=-50, y=-30)  
  
  
**def** mainwindow():  
 username = user.get()  
 password = paswd.get()  
  
 *#checking credentials* **if** username==**'1' and** password==**'1'**:  
  
 *#hiding login window* rootLogin.withdraw()  
  
 *#creating main window on top level* root=Toplevel()  
 app = Window(root)  
 *#setting dimensions of main window* root.geometry(**"1400x800"**)  
 *#root.configure(background='brown')  
 #main window header* root.title(**"Employee Dashboard"**)  
  
  
 *#main window input fields* entry\_1=Entry(root,textvar=ln)  
 entry\_1.place(x=345,y=195,height=30)  
  
  
 entry\_2=Entry(root,textvar=fn1)  
 entry\_2.place(x=345,y=232,height=30)  
  
  
 entry\_3=Entry(root,textvar=fn2)  
 entry\_3.place(x=345,y=270,height=30)  
  
  
 entry\_4=Entry(root,textvar=fn3)  
 entry\_4.place(x=345,y=308,height=30)  
  
  
  
 entry\_5=Entry(root,textvar=fn4)  
 entry\_5.place(x=345,y=345,height=30)  
  
  
 entry\_6=Entry(root,textvar=fn5)  
 entry\_6.place(x=345,y=382,height=30)  
  
  
 entry\_7=Entry(root,textvar=fn6)  
 entry\_7.place(x=345,y=425,height=30)  
  
  
 entry\_8=Entry(root,textvar=fn7)  
 entry\_8.place(x=345,y=465,height=30)  
  
  
 entry\_9=Entry(root,textvar=fn8)  
 entry\_9.place(x=345,y=510,height=30)  
  
  
 entry\_10=Entry(root,textvar=fn9)  
 entry\_10.place(x=345,y=550,height=30)  
  
 entry\_11=Entry(root,textvar=fn10,width=50, font=(**"TimesNewRoman"**,25,**"bold"**),fg=**'red'** )  
 entry\_11.place(x = 645, y = 575, width=627, height=91, )  
 entry\_11.config(state=DISABLED)  
  
 *#main window buttons* b1=Button(root,text=**"SUBMIT"**,width=16,bg=**'forest green'**,fg=**'white'**, font=(**"TimesnewRoman"**,12,**"bold"**),command=getAttrition)  
 b1.place(x=205,y=630)  
  
 b2=Button(root,text=**"Heat Map"**,width=18,bg=**'gray11'**,fg=**'white'**,font=(**"TimesnewRoman"**,12,**"bold"**),command=heatMap)  
 b2.place(x=720,y=180)  
  
  
 b3=Button(root,text=**"Job Satisfaction"**,width=18,bg=**'gray11'**,fg=**'white'**,font=(**"TimesnewRoman"**,12,**"bold"**),command=print())  
 b3.place(x=720,y=280)  
  
 b4=Button(root,text=**"Confusion Matrix"**,width=18,bg=**'gray11'**,fg=**'white'**,font=(**"TimesnewRoman"**,12,**"bold"**),command=confmatrix)  
 b4.place(x=720,y=230)  
  
 b5=Button(root,text=**"Selected Features"**,width=18,bg=**'gray11'**,fg=**'white'**,font=(**"TimesnewRoman"**,12,**"bold"**),command=feature)  
 b5.place(x=970,y=280)  
  
 b6=Button(root,text=**"Male & Female"**,width=18,bg=**'gray11'**,fg=**'white'**,font=(**"arial"**,12,**"bold"**),command=gender)  
 b6.place(x=970,y=230)  
  
 b7=Button(root,text=**"Marital Details"**,width=18,bg=**'gray11'**,fg=**'white'**,font=(**"TimesnewRoman"**,12,**"bold"**),command=marital)  
 b7.place(x=970,y=180)  
  
 b8=Button(root,text=**"Attrition Graph"**,width=18,bg=**'gray11'**,fg=**'white'**,font=(**"TimesnewRoman"**,12,**"bold"**),command=attritionlabel)  
 b8.place(x=850,y=330)  
  
 b9=Button(root,text=**"X"**,width=2,height=1,bg=**'red'**,fg=**'gray11'**,font=(**"TimesnewRoman"**,12,**"bold"**),command=exitt)  
 b9.place(x=1286,y=36)  
  
 *#creating infinite loop for main window* root.mainloop()  
 *#if wrong crdentials* **else**:  
 messagebox.showinfo(**"Error"**, **"Wrong Credentials"**)  
  
  
  
*#Login window***class** Login(Frame):  
 **def** \_\_init\_\_(self, master=**None**):  
 Frame.\_\_init\_\_(self, master)  
 self.master = master  
 self.pack(fill=BOTH, expand=1)  
  
 load = Image.open(**"logobilal2.jpg"**)  
 render = ImageTk.PhotoImage(load)  
 img = Label(self, image=render,width=**"950"**, height=**"1000"**)  
 img.image = render  
 img.place(x=-250, y=-0)  
*#header*app1 = Login(rootLogin)  
*#loginLabel= Label (rootLogin, text="Sign In", relief="solid",width=10,bg='yellow',fg='black',font=("TimesNewRoman",20,"bold"))  
#loginLabel.place(x=150,y=30)  
  
#username field and label*usernameLabel=Label(rootLogin,text=**"User Name"**,relief=**"solid"**,width=18,bg=**'DarkGoldenrod1'**,fg=**'black'**,font=(**"TimesNewRoman"**,10,**"bold"**))  
usernameLabel.place(x=80,y=170)  
username=Entry(rootLogin,textvar=user,width=24,bg=**'gray20'**,fg=**'white'**)  
username.place(x=250,y=170)  
  
*#password field and label*passwordLabel=Label(rootLogin,text=**"Password"**,relief=**"solid"**,width=18,bg=**'DarkGoldenrod1'**,fg=**'black'**,font=(**"TimesNewRoman"**,10,**"bold"**))  
passwordLabel.place(x=80,y=230)  
password=Entry(rootLogin,textvar=paswd,width=24,bg=**'gray20'**,fg=**'white'**)  
password.place(x=250,y=230)  
password.config(show=**"\*"**)  
*#login button*loginbutton=Button(rootLogin,text=**"Login"**,width=10,bg=**'forestgreen'**,fg=**'white'**,font=(**"TimesnewRoman"**,10,**"bold"**),command=mainwindow)  
loginbutton.place(x=200,y=290)  
  
*#infinite loop for login window*rootLogin.mainloop()