**PROJECT NAME:**

Health-Care Chat Bot

**GROUP MEMBERS NAME:**

(27)Aryan Veturekar

(20)Danish Shaikh

(23)Himanshu Singh

**TYPE OF ALGORITHM:**

Supervised Learning

**NUMBER OF FEATURES:**

**1. Omni-capable**

The chatbot converses seamlessly across multiple digital channels and retains data and context for a seamless experience. In best cases, even passing that information to a live agent if needed.

**2. Free to Explore**

The chatbot can reach, consume, and process vast amounts of data– both structured and unstructured–to surface insights from any source - to gather relevant data to solve customer issues quickly.

**3. Autonomous Reasoning**

The chatbot can perform complex reasoning without human intervention. For example, a great Service chatbot should be able to infer solutions based on relevant case histories.

**4. Pre-Trained**

The chatbot is pre-trained to understand brand-specific or industry-specific knowledge and terms. Even better, it’s pre-configured to resolve common customer requests of a particular industry.

**5. Register/Log-in**

To access this chatbot and individual needs to register and then use the registration ID to log in to access the features.

**6. User Interface**

A user friendly interface which is engaging and easy to access.

**PERFORMANCE OR COMPLEXITY**

So, we have been working on Health-Care ChatBot for a month now and I can happily say that now it’s working properly. We have finished the training and testing of this ChatBot.

The UI/UX are designed just as we wanted. In the beginning we were stuck with the user environment we wanted for this software but we worked towards it and now have achieved the UI/UX just as we wanted.

**MODEL CODE**

1. **Health-Care Console**

######## A Healthcare Domain Chatbot to simulate the predictions of a General Physician ########

######## A pragmatic Approach for Diagnosis ############

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

training\_dataset = pd.read\_csv('Training.csv')

test\_dataset = pd.read\_csv('Testing.csv')

# Slicing and Dicing the dataset to separate features from predictions

X = training\_dataset.iloc[:, 0:132].values

#print(X)

y = training\_dataset.iloc[:, -1].values

#print(y)

# Dimensionality Reduction for removing redundancies

dimensionality\_reduction = training\_dataset.groupby(training\_dataset['prognosis']).max()

#print(dimensionality\_reduction)

# Encoding String values to integer constants

from sklearn.preprocessing import LabelEncoder

labelencoder = LabelEncoder()

y = labelencoder.fit\_transform(y)

#print(y)

# Splitting the dataset into training set and test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Implementing the Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier()

classifier.fit(X\_train, y\_train)

# Saving the information of columns

cols = training\_dataset.columns

cols = cols[:-1]

# Checking the Important features

importances = classifier.feature\_importances\_

indices = np.argsort(importances)[::-1]

features = cols

# Implementing the Visual Tree

from sklearn.tree import \_tree

# Method to simulate the working of a Chatbot by extracting and formulating questions

def execute\_bot():

print("Please reply with yes/Yes or no/No for the following symptoms")

def print\_disease(node):

#print(node)

node = node[0]

#print(len(node))

val = node.nonzero()

#print(val)

disease = labelencoder.inverse\_transform(val[0])

return disease

def tree\_to\_code(tree, feature\_names):

tree\_ = tree.tree\_

#print(tree\_)

feature\_name = [

feature\_names[i] if i != \_tree.TREE\_UNDEFINED else "undefined!"

for i in tree\_.feature

]

#print("def tree({}):".format(", ".join(feature\_names)))

symptoms\_present = []

def recurse(node, depth):

indent = " " \* depth

if tree\_.feature[node] != \_tree.TREE\_UNDEFINED:

name = feature\_name[node]

threshold = tree\_.threshold[node]

print(name + " ?")

ans = input()

ans = ans.lower()

if ans == 'yes':

val = 1

else:

val = 0

if val <= threshold:

recurse(tree\_.children\_left[node], depth + 1)

else:

symptoms\_present.append(name)

recurse(tree\_.children\_right[node], depth + 1)

else:

present\_disease = print\_disease(tree\_.value[node])

print( "You may have " + present\_disease )

print()

red\_cols = dimensionality\_reduction.columns

symptoms\_given = red\_cols[dimensionality\_reduction.loc[present\_disease].values[0].nonzero()]

print("symptoms present " + str(list(symptoms\_present)))

print()

print("symptoms given " + str(list(symptoms\_given)) )

print()

confidence\_level = (1.0\*len(symptoms\_present))/len(symptoms\_given)

print("confidence level is " + str(confidence\_level))

print()

print('The model suggests:')

print()

row = doctors[doctors['disease'] == present\_disease[0]]

print('Consult ', str(row['name'].values))

print()

print('Visit ', str(row['link'].values))

#print(present\_disease[0])

recurse(0, 1)

tree\_to\_code(classifier,cols)

# This section of code to be run after scraping the data

doc\_dataset = pd.read\_csv('doctors\_dataset.csv', names = ['Name', 'Description'])

diseases = dimensionality\_reduction.index

diseases = pd.DataFrame(diseases)

doctors = pd.DataFrame()

doctors['name'] = np.nan

doctors['link'] = np.nan

doctors['disease'] = np.nan

doctors['disease'] = diseases['prognosis']

doctors['name'] = doc\_dataset['Name']

doctors['link'] = doc\_dataset['Description']

record = doctors[doctors['disease'] == 'AIDS']

record['name']

record['link']

# Execute the bot and see it in Action

execute\_bot()

1. **New Login**

# import modules

from tkinter import \*

import os

# Designing window for registration

def destroyPackWidget(parent):

for e in parent.pack\_slaves():

e.destroy()

def register():

global root,register\_screen

destroyPackWidget(root)

register\_screen=root

# register\_screen = Toplevel(main\_screen)

register\_screen.title("Register")

register\_screen.geometry("300x250")

global username

global password

global username\_entry

global password\_entry

username = StringVar()

password = StringVar()

Label(register\_screen, text="Please enter details below", bg="blue").pack()

Label(register\_screen, text="").pack()

username\_lable = Label(register\_screen, text="Username \* ")

username\_lable.pack()

username\_entry = Entry(register\_screen, textvariable=username)

username\_entry.pack()

password\_lable = Label(register\_screen, text="Password \* ")

password\_lable.pack()

password\_entry = Entry(register\_screen, textvariable=password, show='\*')

password\_entry.pack()

Label(register\_screen, text="").pack()

Button(register\_screen, text="Register", width=10, height=1, bg="blue", command=register\_user).pack()

# Designing window for login

def login():

global login\_screen

login\_screen = Toplevel(main\_screen)

login\_screen.title("Login")

login\_screen.geometry("300x250")

Label(login\_screen, text="Please enter details below to login").pack()

Label(login\_screen, text="").pack()

global username\_verify

global password\_verify

username\_verify = StringVar()

password\_verify = StringVar()

global username\_login\_entry

global password\_login\_entry

Label(login\_screen, text="Username \* ").pack()

username\_login\_entry = Entry(login\_screen, textvariable=username\_verify)

username\_login\_entry.pack()

Label(login\_screen, text="").pack()

Label(login\_screen, text="Password \* ").pack()

password\_login\_entry = Entry(login\_screen, textvariable=password\_verify, show='\*')

password\_login\_entry.pack()

Label(login\_screen, text="").pack()

Button(login\_screen, text="Login", width=10, height=1, command=login\_verify).pack()

# Implementing event on register button

def btnSucess\_Click():

global root

destroyPackWidget(root)

def register\_user():

global root,username,password

username\_info = username.get()

password\_info = password.get()

print("abc",username\_info,password\_info,"xyz")

file = open(username\_info, "w")

file.write(username\_info + "\n")

file.write(password\_info)

file.close()

username\_entry.delete(0, END)

password\_entry.delete(0, END)

Label(root, text="Registration Success", fg="green", font=("calibri", 11)).pack()

Button(root,text="Click Here to proceed",command=btnSucess\_Click).pack()

# Implementing event on login button

def login\_verify():

username1 = username\_verify.get()

password1 = password\_verify.get()

username\_login\_entry.delete(0, END)

password\_login\_entry.delete(0, END)

list\_of\_files = os.listdir()

if username1 in list\_of\_files:

file1 = open(username1, "r")

verify = file1.read().splitlines()

if password1 in verify:

login\_sucess()

else:

password\_not\_recognised()

else:

user\_not\_found()

# Designing popup for login success

def login\_sucess():

global login\_success\_screen

login\_success\_screen = Toplevel(login\_screen)

login\_success\_screen.title("Success")

login\_success\_screen.geometry("150x100")

Label(login\_success\_screen, text="Login Success").pack()

Button(login\_success\_screen, text="OK", command=delete\_login\_success).pack()

# Designing popup for login invalid password

def password\_not\_recognised():

global password\_not\_recog\_screen

password\_not\_recog\_screen = Toplevel(login\_screen)

password\_not\_recog\_screen.title("Success")

password\_not\_recog\_screen.geometry("150x100")

Label(password\_not\_recog\_screen, text="Invalid Password ").pack()

Button(password\_not\_recog\_screen, text="OK", command=delete\_password\_not\_recognised).pack()

# Designing popup for user not found

def user\_not\_found():

global user\_not\_found\_screen

user\_not\_found\_screen = Toplevel(login\_screen)

user\_not\_found\_screen.title("Success")

user\_not\_found\_screen.geometry("150x100")

Label(user\_not\_found\_screen, text="User Not Found").pack()

Button(user\_not\_found\_screen, text="OK", command=delete\_user\_not\_found\_screen).pack()

# Deleting popups

def delete\_login\_success():

login\_success\_screen.destroy()

def delete\_password\_not\_recognised():

password\_not\_recog\_screen.destroy()

def delete\_user\_not\_found\_screen():

user\_not\_found\_screen.destroy()

# Designing Main(first) window

def main\_account\_screen(frmmain):

main\_screen=frmmain

main\_screen.geometry("300x250")

main\_screen.title("Account Login")

Label(main\_screen,text="Select Your Choice", bg="blue", width="300", height="2", font=("Calibri", 13)).pack()

Label(main\_screen,text="").pack()

Button(main\_screen,text="Login", height="2", width="30", command=login).pack()

Label(main\_screen,text="").pack()

Button(main\_screen,text="Register", height="2", width="30", command=register).pack()

root = Tk()

main\_account\_screen(root)

1. **Question Diagnosis**

####### A Healthcare Domain Chatbot to simulate the predictions of a General Physician ########

######## A pragmatic Approach for Diagnosis ############

# Importing the libraries

from tkinter import \*

from tkinter import messagebox

import os

import webbrowser

import numpy as np

import pandas as pd

class HyperlinkManager:

def \_\_init\_\_(self, text):

self.text = text

self.text.tag\_config("hyper", foreground="blue", underline=1)

self.text.tag\_bind("hyper", "<Enter>", self.\_enter)

self.text.tag\_bind("hyper", "<Leave>", self.\_leave)

self.text.tag\_bind("hyper", "<Button-1>", self.\_click)

self.reset()

def reset(self):

self.links = {}

def add(self, action):

# add an action to the manager. returns tags to use in

# associated text widget

tag = "hyper-%d" % len(self.links)

self.links[tag] = action

return "hyper", tag

def \_enter(self, event):

self.text.config(cursor="hand2")

def \_leave(self, event):

self.text.config(cursor="")

def \_click(self, event):

for tag in self.text.tag\_names(CURRENT):

if tag[:6] == "hyper-":

self.links[tag]()

return

# Importing the dataset

training\_dataset = pd.read\_csv('Training.csv')

test\_dataset = pd.read\_csv('Testing.csv')

# Slicing and Dicing the dataset to separate features from predictions

X = training\_dataset.iloc[:, 0:132].values

Y = training\_dataset.iloc[:, -1].values

# Dimensionality Reduction for removing redundancies

dimensionality\_reduction = training\_dataset.groupby(training\_dataset['prognosis']).max()

# Encoding String values to integer constants

from sklearn.preprocessing import LabelEncoder

labelencoder = LabelEncoder()

y = labelencoder.fit\_transform(Y)

# Splitting the dataset into training set and test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Implementing the Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier()

classifier.fit(X\_train, y\_train)

# Saving the information of columns

cols = training\_dataset.columns

cols = cols[:-1]

# Checking the Important features

importances = classifier.feature\_importances\_

indices = np.argsort(importances)[::-1]

features = cols

# Implementing the Visual Tree

from sklearn.tree import \_tree

# Method to simulate the working of a Chatbot by extracting and formulating questions

def print\_disease(node):

#print(node)

node = node[0]

#print(len(node))

val = node.nonzero()

#print(val)

disease = labelencoder.inverse\_transform(val[0])

return disease

def recurse(node, depth):

global val,ans

global tree\_,feature\_name,symptoms\_present

indent = " " \* depth

if tree\_.feature[node] != \_tree.TREE\_UNDEFINED:

name = feature\_name[node]

threshold = tree\_.threshold[node]

yield name + " ?"

# ans = input()

ans = ans.lower()

if ans == 'yes':

val = 1

else:

val = 0

if val <= threshold:

yield from recurse(tree\_.children\_left[node], depth + 1)

else:

symptoms\_present.append(name)

yield from recurse(tree\_.children\_right[node], depth + 1)

else:

strData=""

present\_disease = print\_disease(tree\_.value[node])

# print( "You may have " + present\_disease )

# print()

strData="You may have :" + str(present\_disease)

QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n')

red\_cols = dimensionality\_reduction.columns

symptoms\_given = red\_cols[dimensionality\_reduction.loc[present\_disease].values[0].nonzero()]

# print("symptoms present " + str(list(symptoms\_present)))

# print()

strData="symptoms present: " + str(list(symptoms\_present))

QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n')

# print("symptoms given " + str(list(symptoms\_given)) )

# print()

strData="symptoms given: " + str(list(symptoms\_given))

QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n')

confidence\_level = (1.0\*len(symptoms\_present))/len(symptoms\_given)

# print("confidence level is " + str(confidence\_level))

# print()

strData="confidence level is: " + str(confidence\_level)

QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n')

# print('The model suggests:')

# print()

strData='The model suggests:'

QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n')

row = doctors[doctors['disease'] == present\_disease[0]]

# print('Consult ', str(row['name'].values))

# print()

strData='Consult '+ str(row['name'].values)

QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n')

# print('Visit ', str(row['link'].values))

#print(present\_disease[0])

hyperlink = HyperlinkManager(QuestionDigonosis.objRef.txtDigonosis)

strData='Visit '+ str(row['link'].values[0])

def click1():

webbrowser.open\_new(str(row['link'].values[0]))

QuestionDigonosis.objRef.txtDigonosis.insert(INSERT, strData, hyperlink.add(click1))

#QuestionDigonosis.objRef.txtDigonosis.insert(END,str(strData)+'\n')

yield strData

def tree\_to\_code(tree, feature\_names):

global tree\_,feature\_name,symptoms\_present

tree\_ = tree.tree\_

#print(tree\_)

feature\_name = [

feature\_names[i] if i != \_tree.TREE\_UNDEFINED else "undefined!"

for i in tree\_.feature

]

#print("def tree({}):".format(", ".join(feature\_names)))

symptoms\_present = []

# recurse(0, 1)

def execute\_bot():

# print("Please reply with yes/Yes or no/No for the following symptoms")

tree\_to\_code(classifier,cols)

# This section of code to be run after scraping the data

doc\_dataset = pd.read\_csv('doctors\_dataset.csv', names = ['Name', 'Description'])

diseases = dimensionality\_reduction.index

diseases = pd.DataFrame(diseases)

doctors = pd.DataFrame()

doctors['name'] = np.nan

doctors['link'] = np.nan

doctors['disease'] = np.nan

doctors['disease'] = diseases['prognosis']

doctors['name'] = doc\_dataset['Name']

doctors['link'] = doc\_dataset['Description']

record = doctors[doctors['disease'] == 'AIDS']

record['name']

record['link']

# Execute the bot and see it in Action

#execute\_bot()

class QuestionDigonosis(Frame):

objIter=None

objRef=None

def \_\_init\_\_(self,master=None):

master.title("Question")

# root.iconbitmap("")

master.state("z")

# master.minsize(700,350)

QuestionDigonosis.objRef=self

super().\_\_init\_\_(master=master)

self["bg"]="light blue"

self.createWidget()

self.iterObj=None

def createWidget(self):

self.lblQuestion=Label(self,text="Question",width=12,bg="bisque")

self.lblQuestion.grid(row=0,column=0,rowspan=4)

self.lblDigonosis = Label(self, text="Digonosis",width=12,bg="bisque")

self.lblDigonosis.grid(row=4, column=0,sticky="n",pady=5)

# self.varQuestion=StringVar()

self.txtQuestion = Text(self, width=100,height=4)

self.txtQuestion.grid(row=0, column=1,rowspan=4,columnspan=20)

self.varDiagonosis=StringVar()

self.txtDigonosis =Text(self, width=100,height=14)

self.txtDigonosis.grid(row=4, column=1,columnspan=20,rowspan=20,pady=5)

self.btnNo=Button(self,text="No",width=12,bg="bisque", command=self.btnNo\_Click)

self.btnNo.grid(row=25,column=0)

self.btnYes = Button(self, text="Yes",width=12,bg="bisque", command=self.btnYes\_Click)

self.btnYes.grid(row=25, column=1,columnspan=20,sticky="e")

self.btnClear = Button(self, text="Clear",width=12,bg="bisque", command=self.btnClear\_Click)

self.btnClear.grid(row=27, column=0)

self.btnStart = Button(self, text="Start",width=12,bg="bisque", command=self.btnStart\_Click)

self.btnStart.grid(row=27, column=1,columnspan=20,sticky="e")

def btnNo\_Click(self):

global val,ans

global val,ans

ans='no'

str1=QuestionDigonosis.objIter.\_\_next\_\_()

self.txtQuestion.delete(0.0,END)

self.txtQuestion.insert(END,str1+"\n")

def btnYes\_Click(self):

global val,ans

ans='yes'

self.txtDigonosis.delete(0.0,END)

str1=QuestionDigonosis.objIter.\_\_next\_\_()

# self.txtDigonosis.insert(END,str1+"\n")

def btnClear\_Click(self):

self.txtDigonosis.delete(0.0,END)

self.txtQuestion.delete(0.0,END)

def btnStart\_Click(self):

execute\_bot()

self.txtDigonosis.delete(0.0,END)

self.txtQuestion.delete(0.0,END)

self.txtDigonosis.insert(END,"Please Click on Yes or No for the Above symptoms in Question")

QuestionDigonosis.objIter=recurse(0, 1)

str1=QuestionDigonosis.objIter.\_\_next\_\_()

self.txtQuestion.insert(END,str1+"\n")

class MainForm(Frame):

main\_Root = None

def destroyPackWidget(self, parent):

for e in parent.pack\_slaves():

e.destroy()

def \_\_init\_\_(self, master=None):

MainForm.main\_Root = master

super().\_\_init\_\_(master=master)

master.geometry("300x250")

master.title("Account Login")

self.createWidget()

def createWidget(self):

self.lblMsg=Label(self, text="Health Care Chatbot", bg="PeachPuff2", width="300", height="2", font=("Calibri", 13))

self.lblMsg.pack()

self.btnLogin=Button(self, text="Login", height="2", width="300", command = self.lblLogin\_Click)

self.btnLogin.pack()

self.btnRegister=Button(self, text="Register", height="2", width="300", command = self.btnRegister\_Click)

self.btnRegister.pack()

self.lblTeam=Label(self, text="Made by:", bg="slateblue4", width = "250", height = "1", font=("Calibri", 13))

self.lblTeam.pack()

self.lblTeam1=Label(self, text="Aryan Veturekar", bg="RoyalBlue1", width = "250", height = "1", font=("Calibri", 13))

self.lblTeam1.pack()

self.lblTeam2=Label(self, text="Himanshu Singh", bg="RoyalBlue2", width = "250", height = "1", font=("Calibri", 13))

self.lblTeam2.pack()

self.lblTeam3=Label(self, text="Danish Shaikh", bg="RoyalBlue3", width = "250", height = "1", font=("Calibri", 13))

self.lblTeam3.pack()

def lblLogin\_Click(self):

self.destroyPackWidget(MainForm.main\_Root)

frmLogin=Login(MainForm.main\_Root)

frmLogin.pack()

def btnRegister\_Click(self):

self.destroyPackWidget(MainForm.main\_Root)

frmSignUp = SignUp(MainForm.main\_Root)

frmSignUp.pack()

class Login(Frame):

main\_Root=None

def destroyPackWidget(self,parent):

for e in parent.pack\_slaves():

e.destroy()

def \_\_init\_\_(self, master=None):

Login.main\_Root=master

super().\_\_init\_\_(master=master)

master.title("Login")

master.geometry("300x250")

self.createWidget()

def createWidget(self):

self.lblMsg=Label(self, text="Please enter details below to login",bg="blue")

self.lblMsg.pack()

self.username=Label(self, text="Username \* ")

self.username.pack()

self.username\_verify = StringVar()

self.username\_login\_entry = Entry(self, textvariable=self.username\_verify)

self.username\_login\_entry.pack()

self.password=Label(self, text="Password \* ")

self.password.pack()

self.password\_verify = StringVar()

self.password\_login\_entry = Entry(self, textvariable=self.password\_verify, show='\*')

self.password\_login\_entry.pack()

self.btnLogin=Button(self, text="Login", width=10, height=1, command=self.btnLogin\_Click)

self.btnLogin.pack()

def btnLogin\_Click(self):

username1 = self.username\_login\_entry.get()

password1 = self.password\_login\_entry.get()

# messagebox.showinfo("Failure", self.username1+":"+password1)

list\_of\_files = os.listdir()

if username1 in list\_of\_files:

file1 = open(username1, "r")

verify = file1.read().splitlines()

if password1 in verify:

messagebox.showinfo("Sucess","Login Sucessful")

self.destroyPackWidget(Login.main\_Root)

frmQuestion = QuestionDigonosis(Login.main\_Root)

frmQuestion.pack()

else:

messagebox.showinfo("Failure", "Login Details are wrong try again")

else:

messagebox.showinfo("Failure", "User not found try from another user\n or sign up for new user")

class SignUp(Frame):

main\_Root=None

print("SignUp Class")

def destroyPackWidget(self,parent):

for e in parent.pack\_slaves():

e.destroy()

def \_\_init\_\_(self, master=None):

SignUp.main\_Root=master

master.title("Register")

super().\_\_init\_\_(master=master)

master.title("Register")

master.geometry("300x250")

self.createWidget()

def createWidget(self):

self.lblMsg=Label(self, text="Please enter details below", bg="blue")

self.lblMsg.pack()

self.username\_lable = Label(self, text="Username \* ")

self.username\_lable.pack()

self.username = StringVar()

self.username\_entry = Entry(self, textvariable=self.username)

self.username\_entry.pack()

self.password\_lable = Label(self, text="Password \* ")

self.password\_lable.pack()

self.password = StringVar()

self.password\_entry = Entry(self, textvariable=self.password, show='\*')

self.password\_entry.pack()

self.btnRegister=Button(self, text="Register", width=10, height=1, bg="blue", command=self.register\_user)

self.btnRegister.pack()

def register\_user(self):

file = open(self.username\_entry.get(), "w")

file.write(self.username\_entry.get() + "\n")

file.write(self.password\_entry.get())

file.close()

self.destroyPackWidget(SignUp.main\_Root)

self.lblSucess=Label(root, text="Registration Success", fg="green", font=("calibri", 11))

self.lblSucess.pack()

self.btnSucess=Button(root, text="Click Here to proceed", command=self.btnSucess\_Click)

self.btnSucess.pack()

def btnSucess\_Click(self):

self.destroyPackWidget(SignUp.main\_Root)

frmQuestion = QuestionDigonosis(SignUp.main\_Root)

frmQuestion.pack()

root = Tk()

frmMainForm=MainForm(root)

frmMainForm.pack()

root.mainloop()