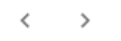







# Appendices

## Appendix A:

### Email with client after first meeting



**Nipun Rustagi** <nipun.rustagi1@gmail.com>  
to careercounselor.intl@chirec.ac.in, Sirisha, CHIREC International Principal, ▾





Respected Ma'am,

I am writing to you to finalize the criteria and details of the product that was discussed in our meeting. If you could specify the basic outline and expectations of the product to be developed, we can further decide how to go about the application.

--

With Best Regards,  
Nipun Rustagi  
(IB12-A)

---

**career counselor.intl@chirec.ac.in** via chirecinternational.onmicrosoft.com  
to me ▾

Hi Nipun. As we discussed in our meeting and upon confirming with the Principal and co-ordinator, I would like you to develop a software that could calculate the probability of securing admission into a college based on quantitative factors such as SAT, ACT as these are widely written by students, as well as qualitative factors such as the residency or nationality of the student. While I have used such tools before, I haven't come across any software that could even take into account the IB Predicted Score or the predicted percentage score. So, I would like you to incorporate that as well. As you come up with the basic outline, we will add more features. If any questions, please message or call me.

\*\*\*

### Call Transcript (Second Meeting)

**Me:** Ma'am, you have stated the need for a college probability admissions calculator. I think I have a general idea. There can be 4 factors based on which the probability of admission into a particular college can be calculated. This could be an ML Application based on past student data that I can retrieve from CDS. The factors can be SAT/ACT Score, IB/GPA Predicted Score, and Residency Status.

**Client:** Sounds good. I had a question. Why an application based on ML and not just using data like top 15% percentile?

**Me:** The problem with using statistics like top 15% score of students admitted and rejected is that it's not very true of the actual data. This is so because at many times we see that students who are in the top 15% percentile for the college they applied to still can get rejected. So using actual student data and whether they were accepted or not to build a predictive model would be more precise.

**Client:** I guess it is better to build a predictive model based on actual data of each student then. As you know that we cannot breach the confidentiality of our students' data, you must obtain the data from elsewhere, if that is possible.

**Me:** Yes, ma'am. I understand. That shouldn't be a problem as there is huge collection of datasets readily available online.

**Client:** Also, I was thinking if we could add the choice a major the student would apply to, that would be much more beneficial because certain programs in every university usually tend to be more competitive than others.

**Me:** Yes, ma'am, I agree with that. I'm pretty sure I will be able to do that.

**Client:** Thank you. Another thing, since the school does not follow the GPA system but rather only IB and Percentage, is it perhaps possible to avoid GPA as a factor and build the application for IB and Percentage?

**Me:** Well since universities in USA primarily utilize the GPA system, I might not be able to find student data with Percentage. However, I believe that there is a conversion scale from GPA to Percentage based on which I can convert the GPA Score to Percentage, so that shouldn't be a problem.

**Client:** Ah, that sounds good. Overall, I don't have any specific features in mind and these are mine and the school's basic requirements. I just want it to be a comfortable product that is easy to use. I want the program to be very neat and straightforward so that I can easily get the final probability.

**Me:** Sure ma'am, not a problem!

**Client:** Great! So, by the end of the day, I will send you the list of the US Colleges that I would require as I would have to see the colleges that are most applied to in the US.

## Appendix C:

### Annotated Code

#### Backend

##### sigmoid.py

```
import numpy as np
def sigmoid(z):
    # sigmoid(z) computes and returns the sigmoid (activation function) of z.
    g = np.zeros(z.shape)
    g = 1 / (1 + np.exp(-z))

    return g
```

##### sigmoidGradient.py

```
import sigmoid as s
def sigmoidGradient(z):
    # sigmoidGradient(z) computes and returns the gradient of the sigmoid function
    # evaluated at z. This should work regardless if z is a matrix or a
    # vector. In particular, if z is a vector or matrix, it should return
    # the gradient for each element.

    g = s.sigmoid(z)
    g = g * (1 - g)

    return g
```

##### predict.py

```
import sigmoid as s
import numpy as np
def predict(Theta1, Theta2, X):
    # predict(Theta1, Theta2, X) computes and returns the values
    # of the input layer a1, hidden layer a2, output layer a3
    # and the matrix product z2 of input layer a1 and weight Theta 1
    m = X.shape[0]
    a1 = np.column_stack((np.ones((m, 1)), X))
    z2 = np.dot(a1, Theta1.T)
    a2 = np.column_stack((np.ones((z2.shape[0], 1)), s.sigmoid(z2)))
    z3 = np.dot(a2, Theta2.T)
    a3 = s.sigmoid(z3)

    return z2, a1, a2, a3
```

##### nnCostFunction.py

```
import numpy as np
import predict as pr
import sigmoidGradient as sg
def nnCostFunction(nn_params, input_layer_size, hidden_layer_size, num_labels, X, y,
lambda_reg):
    # nnCostFunction (nn_params, input_layer_size, hidden_layer_size, num_labels, X, y,
lambda reg)
    # implements the neural network cost function.
    # It computes the cost and gradient of the neural network. The
    # parameters for the neural network are "unrolled" into the vector
    # nn_params and need to be converted back into the weight matrices.
    # The returned parameter grad should be a "unrolled" vector of the
    # partial derivatives of the neural network.
```

```

Theta1 = np.reshape(nn_params[:hidden_layer_size * (input_layer_size + 1)],
                    (hidden_layer_size, input_layer_size + 1), order='F')
Theta2 = np.reshape(nn_params[hidden_layer_size * (input_layer_size + 1):],
                    (num_labels, hidden_layer_size + 1), order='F')

m = X.shape[0]
z2, a1, a2, a3 = pr.predict(Theta1, Theta2, X)
## =====
# Unregularized Cost function is calculated
cost = np.sum((-y) * np.log(a3) - ((1 - y) * np.log(1 - a3)))
J = (1.0 / m) * cost
## =====
# Regularized Cost function is calculated
sumOfTheta1 = np.sum(Theta1[:, 1:] ** 2)
sumOfTheta2 = np.sum(Theta2[:, 1:] ** 2)
J = J + ((lambda_reg / (2.0 * m)) * (sumOfTheta1 + sumOfTheta2))
## =====
# Backpropagation is implemented and thus the analytical gradients are calculated
and unrolled into single vector
delta3 = a3 - y
delta2 = (np.dot(delta3, Theta2)) * np.column_stack((np.ones((z2.shape[0], 1)),
sg.sigmoidGradient(z2)))
delta2 = delta2[:, 1:]
Theta2_grad = (np.dot(delta3.T, a2)) * (1 / m)
Theta1_grad = (np.dot(delta2.T, a1)) * (1 / m)
Theta2_grad = Theta2_grad + (
    (float(lambda_reg) / m) * np.column_stack((np.zeros((Theta2.shape[0],
1)), Theta2[:, 1:])))
Theta1_grad = Theta1_grad + (
    (float(lambda_reg) / m) * np.column_stack((np.zeros((Theta1.shape[0],
1)), Theta1[:, 1:])))
grad = np.concatenate(
    (Theta1_grad.reshape(Theta1_grad.size, order='F'),
Theta2_grad.reshape(Theta2_grad.size, order='F')))
## =====

return J, grad

```

### computeNumericalGradient.py

```

import numpy as np
def computeNumericalGradient(J, theta):
    # computeNumericalGradient(J, theta) computes the numerical
    # gradient of the function J around theta. Calling y = J(theta) should
    # return the function value at theta.

    # Note: The following code implements numerical gradient checking, and
    # returns the numerical gradient. It sets numgrad(i) to (a numerical
    # approximation of) the partial derivative of J with respect to the
    # i-th input argument, evaluated at theta. (i.e., numgrad(i) should
    # be the (approximately) the partial derivative of J with respect
    # to theta(i))
    #
    numgrad = np.zeros(theta.shape[0])
    perturb = np.zeros(theta.shape[0])
    e = 1e-4
    for p in range(theta.size):
        # Set perturbation vector
        perturb[p] = e
        loss1 = J(theta - perturb)
        loss2 = J(theta + perturb)

        # Compute Numerical Gradient
        numgrad[p] = (loss2[0] - loss1[0]) / (2*e)
        perturb[p] = 0

    return numgrad

```

## debugging.py

```
import numpy as np
def debugging(rows, cols):
    # debugging(rows, cols) Initialize the weights of cols as
    # incoming connections and rows as outgoing connections using a fixed
    # strategy, this will help later in debugging

    # Note that data is to be set to a matrix of size(rows, cols)

    data = np.zeros((rows, cols))
    data = np.reshape(np.sin(range(data.size)), data.shape) / 10

    return data
```

## checkNeuralNetworkGrad.py

```
import numpy as np
import debugging as deb
import nnCostFunction as nnCF
import computeNumericalGradient as cng
from decimal import Decimal
import sys

def checkNeuralNetworkGrad(lambda_reg):
    ## =====
    # checkNeuralNetworkGrad(lambda reg) creates a small neural network to check the
    # backpropagation gradients. It will output the analytical gradients
    # produced by the backpropagation code and the numerical gradients (computed
    # using computeNumericalGradient). These two gradient computations should
    # result in very similar values.
    input_layer_size = 5
    hidden_layer_size = 10
    num_labels = 1
    m = 6
    Theta1 = deb.debugging(hidden_layer_size, input_layer_size + 1)
    Theta2 = deb.debugging(num_labels, hidden_layer_size + 1)
    X = deb.debugging(m, input_layer_size)
    y = np.random.randint(0, num_labels + 1, (m, num_labels))
    nn_params = np.concatenate((Theta1.reshape(Theta1.size, order='F'),
    Theta2.reshape(Theta2.size, order='F')))

    def costFunc(p):
        return nnCF.nnCostFunction(p, input_layer_size, hidden_layer_size,
                                   num_labels, X, y, lambda_reg)

    _, grad = costFunc(nn_params)
    numgrad = cng.computeNumericalGradient(costFunc, nn_params)
    ## =====
    # Relative difference between numerical and analytical gradient is calculated.
    # If the relative difference is greater than 1e-9, a warning is displayed and the
    program is terminated.
    diff = Decimal(np.linalg.norm(numgrad - grad)) / Decimal(np.linalg.norm(numgrad +
    grad))
    if diff > 1e-9:
        print("Error, relative difference is greater than 1e-9, check difference")
        sys.exit()
```

## main.py

```
import numpy as np
import nnCostFunction as nncf
import checkNeuralNetworkGrad as cnnng
import predict as pr
from sklearn import metrics
from sklearn.model_selection import train_test_split
from joblib import dump
import os
from scipy.optimize import minimize
from sklearn import preprocessing

# obtaining path of the file directory
my_path = os.path.abspath(os.path.dirname(__file__))
# initializing hidden layer size and output layer size
hidden_layer_size = 10
num_labels = 1
## =====
# initializing lists for storing the name of the colleges, for storing path of features
data
# and admission status data of each corresponding college
all_statusfiles = []
colleges = []
all_featurefiles = []

## =====

# A depth-first preorder traversal is done in 'my_path' and the necessary values are
found and stored
for r, d, f in os.walk(my_path + '/College Data/'):
    colleges.append(d)
    for item in f:
        if 'Admissionstatus.csv' in item:
            all_statusfiles.append(os.path.join(r, item))
        elif 'Featuresdata.csv' in item:
            all_featurefiles.append(os.path.join(r, item))

collegelist = np.array(colleges[0])
num_of_colleges = len(collegelist)
collegelist = np.repeat(collegelist, 2)
# collegelist is duplicated and appended to the end of
# collegelist as the process is to be repeated for calculating
# optimized weights for dataset with IB Predicted and GPA Score

## =====
# Based on the outer loop, the calculations, optimized weights for all the colleges are
computed for dataset with
# either GPA Score or with GPA

# Based on inner loop, the the calculations, optimized weights for all the colleges are
computed for dataset with
# ACT Score or SAT Score
for j in range(2):
    for i in range(len(all_featurefiles)):
        x = np.loadtxt(all_featurefiles[i], delimiter=",")
        if j == 1:
            x = np.delete(x, 1, 1)
        elif j == 0:
            x = np.delete(x, 2, 1)
        input_layer_size = x.shape[1]
        ## =====
        # Preprocessing of data is done
        scaler = preprocessing.StandardScaler().fit(x)
        x = scaler.transform(x)
        abc = np.loadtxt(all_statusfiles[i])[np.newaxis]
        Y = abc.T
```

```

x_train, x_test, y_train, y_test = train_test_split(x, Y, test_size=0.3)
X = x_train
y = y_train
m = X.shape[0]
Theta1 = np.random.rand(hidden_layer_size, input_layer_size + 1)
Theta2 = np.random.rand(num_labels, hidden_layer_size + 1)
nn_params = np.concatenate((Theta1.reshape(Theta1.size, order='F'),
Theta2.reshape(Theta2.size, order='F')))
lambda_reg = 0
## =====
# Checking Backpropagation
cnng.checkNeuralNetworkGrad(lambda_reg)
# Checking Backpropagation with Regularization
lambda_reg = 0.01
cnng.checkNeuralNetworkGrad(lambda_reg)
## =====
# The Neural Network is trained and optimized weights Theta 1 and Theta 2 are
#calculated

maxiter = 100000
myargs = (input_layer_size, hidden_layer_size, num_labels, X, y, lambda_reg)
results = minimize(nncf.nnCostFunction, x0=nn_params, args=myargs,
options={'disp': True, 'maxiter': maxiter, 'ftol': 0},
method="L-BFGS-B", jac=True)

nn_params = results.x
Theta1 = np.reshape(nn_params[:hidden_layer_size * (input_layer_size + 1)],
(hidden_layer_size, input_layer_size + 1), order='F')
Theta2 = np.reshape(nn_params[hidden_layer_size * (input_layer_size + 1):],
(num_labels, hidden_layer_size + 1), order='F')

## =====
# Area under the precision-recall curve is caclulated

y_test_prob = pr.predict(Theta1, Theta2, x_test)[3]
precision, recall, thresholds = metrics.precision_recall_curve(y_test,
y_test_prob)
auc = metrics.auc(recall, precision)
aucarray = np.array([auc])

## =====
# Based on the inner loop and outer loop, the optimized weights calculated are
#stored in the appropriate
# folder
if j == 1:
    if i % 2 == 0:
        IBPath = my_path + "/Optimized Weights for IB/" + collegelist[i] + "/"
+ "ACT"
    elif i % 2 != 0:
        IBPath = my_path + "/Optimized Weights for IB/" + collegelist[i] + "/"
+ "SAT"
    # If the folder does not exist, it's created
    if not os.path.exists(IBPath):
        os.makedirs(IBPath)

    np.savetxt(os.path.join(IBPath, 'Theta1.csv'), Theta1, delimiter=',')
    np.savetxt(os.path.join(IBPath, 'Theta2.csv'), Theta2, delimiter=',')
    np.savetxt(os.path.join(IBPath, 'Auc.csv'), aucarray)
    dump(scaler, os.path.join(IBPath, 'scaler_file.joblib'))
elif j == 0:
    if i % 2 == 0:
        GPAPath = my_path + "/Optimized Weights for GPA/" + collegelist[i] +
"/" + "ACT"
    elif i % 2 != 0:
        GPAPath = my_path + "/Optimized Weights for GPA/" + collegelist[i] +
"/" + "SAT"
    # If the folder does not exist, it's created

```



```

if not os.path.exists(GPAPath):
    os.makedirs(GPAPath)
# scaling attributes, Area Under the Precision-Recall curve value
# and the optimized weights are then stored
np.savetxt(os.path.join(GPAPath, 'Theta1.csv'), Theta1, delimiter=',')
np.savetxt(os.path.join(GPAPath, 'Theta2.csv'), Theta2, delimiter=',')
np.savetxt(os.path.join(GPAPath, 'Auc.csv'), aucarray, )
dump(scaler, os.path.join(GPAPath, 'scaler_file.joblib'))
## =====
## =====

```

## GUI

### Homepage.py

```

import tkinter as tk
from tkinter import font as tkfont
from tkinter import *
import os
from tkinter import ttk
from tkinter import messagebox

root = tk.Tk()
## =====
# obtaining path of the GUI folder
my_path = os.path.abspath(os.path.dirname(__file__))
drop_GUI_path = os.path.normpath(my_path + os.sep + os.pardir)
# obtaining path of backend folder
backend_path = (drop_GUI_path + '/backend')
## =====
# Setting the aesthetics of the Home page
LabelFont = tkfont.Font(family='Times', size=16)
ButtonFont = tkfont.Font(family='Times', size=12)
## =====
# Getting the list of colleges available
coll = []
for r, d, f in os.walk(backend_path + '/College Data'):
    coll.append(d)
colleges = coll[0]
## =====
# Boolean variable is created having the default value of false
# that would later be utilized by method on_closing() based on which the message
# confirm and close the application might be displayed

QuitApplication = tk.BooleanVar()

## =====
# Default value for the TestType and ScoreType chosen is set so as to avoid
# error of moving to the next window without choosing any of the values
Testtype = tk.StringVar()
Testtype.set("SAT")
Scoretype = tk.StringVar()
Scoretype.set("IB")
# Default value for the College chosen is set so as to avoid
# error of moving to the next window without choosing any of the values
CollegeChoice = tk.StringVar()
CollegeChoice.set(colleges[0])
## =====
# Combo box is created so as to choose the College
labell = tk.Label(text="Choose the College", font=LabelFont)
labell.grid(row=2, column=0, pady=20)
collegelist = ttk.Combobox(state="readonly", textvariable=CollegeChoice,
values=coll[0], width=30)
collegelist.grid(row=2, column=1, pady=10)
## =====

```

```

# Radio button is created so as to choose Standardized test type
label2 = tk.Label(text="Choose Type of Standardized Test", font=LabelFont)
label2.grid(row=3, column=0, padx=10, pady=20)
Radiobutton(text="SAT", variable=Testtype, value="SAT").grid(row=3, column=1)
Radiobutton(text="ACT", variable=Testtype, value="ACT").grid(row=3, column=2)
## =====
# Radio button is created so as to choose high school score type
# The text for one of the radio button is set to Percentage while the value of that is
set to GPA
# because ultimately the client would like to use percentage to calculate the
probability. However,
# since the data available was for GPA, the Percentage score entered in the next
window, if chosen, would
# be converted to GPA as the optimized weights are calculated for GPA
label3 = tk.Label(text="Choose High School Score Type", font=LabelFont)
label3.grid(row=4, column=0, pady=20, padx=10)
Radiobutton(text="IB", variable=Scoretype, value="IB").grid(row=4, column=1)
Radiobutton(text="Percentage", variable=Scoretype, value="GPA").grid(row=4, column=2)
## =====
# Calculate button is created that would based on the client's input would open the
appropriate
# window and close the Home Page window
Calculate = tk.Button(text="Click to Proceed", height=1, width=14,
command=root.destroy, font=ButtonFont)
Calculate.grid(row=7, column=1, pady=20)
## =====
# Creating method that would quit the application in case the close button
# is clicked
def on_closing():
    if messagebox.askokcancel('Quit', 'Do you want to quit?'):
        root.destroy()
        QuitApplication.set(True)

root.protocol('WM_DELETE_WINDOW', on_closing)
## =====
root.mainloop()

```

## Mainapp.py

```

from joblib import load
import sklearn
from tkinter import *
import tkinter as tk
import numpy as np
import os
from tkinter import ttk
from tkinter import messagebox
import sys
import predict as pr
from datetime import datetime

from tkinter import tix
## =====
# An instantiation (object) of the Home Page is created. By doing so, it is ensured
that the Home Page
# and the main page are all created through one python file and executable. The values
therefore
# chosen in the Homepage by the user(client) would be accessible to the Main Page based
# on which the appropriate labels and data is set
import Homepage as hp
## =====
from tkinter import font as tkfont
root = tk.Tk()
class CreateToolTip(object):
    """

```

```

create a tooltip for a given widget
"""
def __init__(self, widget, text='widget info'):
    self.waittime = 500      #milliseconds
    self.wraplength = 180    #pixels
    self.widget = widget
    self.text = text
    self.widget.bind("<Enter>", self.enter)
    self.widget.bind("<Leave>", self.leave)
    self.widget.bind("<ButtonPress>", self.leave)
    self.id = None
    self.tw = None

def enter(self, event=None):
    self.schedule()

def leave(self, event=None):
    self.unschedule()
    self.hidetip()

def schedule(self):
    self.unschedule()
    self.id = self.widget.after(self.waittime, self.showtip)

def unschedule(self):
    id = self.id
    self.id = None
    if id:
        self.widget.after_cancel(id)

def showtip(self, event=None):
    x = y = 0
    x, y, cx, cy = self.widget.bbox("insert")
    x += self.widget.winfo_rootx() + 25
    y += self.widget.winfo_rooty() + 20
    # creates a toplevel window
    self.tw = tk.Toplevel(self.widget)
    # Leaves only the label and removes the app window
    self.tw.wm_overrideredirect(True)
    self.tw.wm_geometry("+%d+%d" % (x, y))
    label = tk.Label(self.tw, text=self.text, justify='left',
                     background="#ffffff", relief='solid', borderwidth=1,
                     wraplength = self.wraplength)
    label.pack(ipadx=1)

def hidetip(self):
    tw = self.tw
    self.tw= None
    if tw:
        tw.destroy()

## =====
# Dynamic size of the window is chosen that would therefore have the size based on the
# values chosen by the client in the Home page
root.geometry("")
## =====
# Setting the aesthetics of the Main page
LabelFont = tkfont.Font(family='Times', size=16)
ButtonFont = tkfont.Font(family='Times', size=12)
## =====
# If the user chooses to close the application from Home Page, the Main Page will also
get closed
if hp.QuitApplication.get():
    sys.exit()
## =====
# Based on the College Choice, Score type, and Test type chosen by the user in the Home
Page,
# the appropriate directory is accessed and files are loaded

```

```

MajorDir = (
    hp.backend_path + '/Original College Data/Datasets/' + hp.CollegeChoice.get() +
    '.csv')
ScalerFile = (
    hp.backend_path + '/Optimized Weights for ' + hp.Scoretype.get() + '/' +
    hp.CollegeChoice.get() + '/' + hp.Testtype.get() + '/scaler_file.joblib')
Theta1Dir = (
    hp.backend_path + '/Optimized Weights for ' + hp.Scoretype.get() + '/' +
    hp.CollegeChoice.get() + '/' + hp.Testtype.get() + '/Theta1.csv')
Theta2Dir = (
    hp.backend_path + '/Optimized Weights for ' + hp.Scoretype.get() + '/' +
    hp.CollegeChoice.get() + '/' + hp.Testtype.get() + '/Theta2.csv')
Features = np.loadtxt(
    hp.backend_path + '/College Data/' + hp.CollegeChoice.get() + '/' +
    hp.Testtype.get() + '/Featuresdata.csv',
    delimiter=',')
## =====
# No of Features is calculated so as to know whether the college is public or private
as
# private colleges do not consider the residency status of the applicant
NoOfFeatures = Features.shape[1]
# Entry widget is created for entering the values of the Test type and Score type
chosen by the user
SATScore = tk.Entry()
ACTScore = tk.Entry()
IBScore = tk.Entry()
PercentageScore = tk.Entry()
# The optimized weights Theta1 and Theta2 based on the user's input in the Home page is
loaded
Theta1 = np.loadtxt(Theta1Dir, delimiter=',')
Theta2 = np.loadtxt(Theta2Dir, delimiter=',')
# The appropriate list of majors and scaling values are loaded that are based
# on the user's input in the Home page
X = np.loadtxt(MajorDir, delimiter=',', dtype='object', skiprows=0)
scaler = load(ScalerFile)
# Integer variable is created that will hold the appropriate integer equivalent of the
major that was
# mapped for each and every major in each college. This will thus be utilized as part
of the probability
# calculation
MajorEquivalentVal = tk.IntVar()

## =====
# A callback method is created that will be utilized to check if the values entered are
digits
def callback(P):
    if str.isdigit(P):
        return True
    elif P == "":
        return True
    else:
        return False

## =====
## =====
# Following methods are created that will based, on the values chosen in the Home Page
and the Main page,
# will perform the appropriate validations, display the appropriate messages,
# and calculate the correct probability
# In case the user has chosen Percentage as Score type in the Home Page, the
percentage entered in the entry
# field will be recalculated to percentage
## =====
def CalcProbwithSATandIB():
    SATValidation = tk.BooleanVar(value=False)
    IBValidation = tk.BooleanVar(value=False)
    if (SATScore.get()).isdigit():
        if (int(SATScore.get()) >= 720) and (int(SATScore.get()) <= 1600) and

```

```

(int(SATScore.get()) % 10 == 0):
    SATValidation.set(True)
else:
    messagebox.showerror("Error", "Enter appropriate SAT Score between 720 and
1600")
elif (SATScore.get() == " ") or ((SATScore.get()).isdigit() == False):
    messagebox.showerror("Error", "Enter appropriate SAT Score between 720 and
1600")
if (IBScore.get()).isdigit():
    if (int(IBScore.get()) >= 24) and (int(IBScore.get()) <= 42):
        IBValidation.set(True)
    else:
        messagebox.showerror("Error", "Enter IB Predicted Score between 24 and 42")
elif (IBScore.get() == " ") or ((IBScore.get()).isdigit() == False):
    messagebox.showerror("Error", "Enter IB Predicted Score between 24 and 42")
if (IBValidation.get() == True) and (SATValidation.get() == True):
    for i in range(len(list1)):
        if MajorChoice.get() == list1[i]:
            MajorEquivalentVal.set((MajEquivalent[i]))
if NoOfFeatures == 5:
    Prob = np.array([SATScore.get(), IBScore.get(), ResidencyStatus.get(),
MajorEquivalentVal.get()])[
        np.newaxis]
    Prob = scaler.transform(Prob)
    probability = float(pr.predict(Theta1, Theta2, Prob)[3])
    message = ("The probability of getting admission into " +
hp.CollegeChoice.get() + " is " + str(
        round(probability * 100, 2)) + "%")
    messagebox.showinfo("Probability", message)
else:
    Prob = np.array([SATScore.get(), IBScore.get(),
MajorEquivalentVal.get()])[np.newaxis]
    Prob = scaler.transform(Prob)
    probability = float(pr.predict(Theta1, Theta2, Prob)[3])
    message = ("The probability of getting admission into " +
hp.CollegeChoice.get() + " is " + str(
        round(probability * 100, 2)) + "%")
    messagebox.showinfo("Probability", message)

def CalcProbwithACTandIB():
    ACTValidation = tk.BooleanVar(value=False)
    IBValidation = tk.BooleanVar(value=False)
    if (ACTScore.get()).isdigit():
        if (int(ACTScore.get()) >= 15) and (int(ACTScore.get()) <= 36):
            ACTValidation.set(True)
        else:
            messagebox.showerror("Error", "Enter ACT Score between 15 and 36")
    elif (ACTScore.get() == " ") or ((ACTScore.get()).isdigit() == False):
        messagebox.showerror("Error", "Enter ACT Score between 15 and 36")
    if (IBScore.get()).isdigit():
        if (int(IBScore.get()) >= 24) and (int(IBScore.get()) <= 42):
            IBValidation.set(True)
        else:
            messagebox.showerror("Error", "Enter IB Predicted Score between 24 and 42")
    elif (IBScore.get() == " ") or ((IBScore.get()).isdigit() == False):
        messagebox.showerror("Error", "Enter IB Predicted Score between 24 and 42")
    if (IBValidation.get() == True) and (ACTValidation.get() == True):
        for i in range(len(list1)):
            if MajorChoice.get() == list1[i]:
                MajorEquivalentVal.set((MajEquivalent[i]))
    if NoOfFeatures == 5:
        Prob = np.array([ACTScore.get(), IBScore.get(), ResidencyStatus.get(),
MajorEquivalentVal.get()])[
            np.newaxis]
        Prob = scaler.transform(Prob)
        probability = float(pr.predict(Theta1, Theta2, Prob)[3])
        message = ("The probability of getting admission into " +

```

```

hp.CollegeChoice.get() + " is " + str(
    round(probability * 100, 2)) + "%"
messagebox.showinfo("Probability", message)
#print(probability)
else:
    Prob = np.array([ACTScore.get(), IBScore.get(),
MajorEquivalentVal.get()])[np.newaxis]
    Prob = scaler.transform(Prob)
    probability = float(pr.predict(Theta1, Theta2, Prob)[3])
    message = ("The probability of getting admission into " +
hp.CollegeChoice.get() + " is " + str(
    round(probability * 100, 2)) + "%")
    messagebox.showinfo("Probability", message)
    #print(probability)

def CalcProbwithSATandPercentage():
    SATValidation = tk.BooleanVar(value=False)
    PercentageValidation = tk.BooleanVar(value=False)
    if (SATScore.get()).isdigit():
        if (int(SATScore.get()) >= 720) and (int(SATScore.get()) <= 1600) and
(int(SATScore.get()) % 10 == 0):
            SATValidation.set(True)
        else:
            messagebox.showerror("Error", "Enter appropriate SAT Score between 720 and
1600")
    elif (SATScore.get() == " ") or ((SATScore.get()).isdigit() == False):
        messagebox.showerror("Error", "Enter appropriate SAT Score between 720 and
1600")
    if (PercentageScore.get()).isdigit():
        if (int(PercentageScore.get()) >= 40) and (int(PercentageScore.get()) <= 100):
            PercentageValidation.set(True)
        else:
            messagebox.showerror("Error", "Enter Percentage between 40 and 100")
    elif (PercentageScore.get() == " ") or ((PercentageScore.get()).isdigit() ==
False):
        messagebox.showerror("Error", "Enter Percentage between 40 and 100")
    if (PercentageValidation.get() == True) and (SATValidation.get() == True):
        ## =====
        # Validated percentage will be converted to GPA Score
        PercentagetogPA = (int(PercentageScore.get()) / 20) - 1
        for i in range(len(list1)):
            if MajorChoice.get() == list1[i]:
                MajorEquivalentVal.set(MajEquivalent[i])
    if NoOfFeatures == 5:
        Prob = np.array([SATScore.get(), PercentagetogPA, ResidencyStatus.get(),
MajorEquivalentVal.get()])[np.newaxis]
        Prob = scaler.transform(Prob)
        probability = float(pr.predict(Theta1, Theta2, Prob)[3])
        message = ("The probability of getting admission into " +
hp.CollegeChoice.get() + " is " + str(
    round(probability * 100, 2)) + "%")
        messagebox.showinfo("Probability", message)
    else:
        Prob = np.array([SATScore.get(), PercentagetogPA,
MajorEquivalentVal.get()])[np.newaxis]
        Prob = scaler.transform(Prob)
        probability = float(pr.predict(Theta1, Theta2, Prob)[3])
        message = ("The probability of getting admission into " +
hp.CollegeChoice.get() + " is " + str(
    round(probability * 100, 2)) + "%")
        messagebox.showinfo("Probability", message)

def CalcProbwithACTandPercentage():
    ACTValidation = tk.BooleanVar(value=False)
    PercentageValidation = tk.BooleanVar(value=False)
    if (ACTScore.get()).isdigit():

```

```

    if (int(ACTScore.get()) >= 15) and (int(ACTScore.get()) <= 36):
        ACTValidation.set(True)
    else:
        messagebox.showerror("Error", "Enter ACT Score between 15 and 36")
    elif (ACTScore.get() == " ") or ((ACTScore.get()).isdigit() == False):
        messagebox.showerror("Error", "Enter ACT Score between 15 and 36")
    if (PercentageScore.get()).isdigit():
        if (int(PercentageScore.get()) >= 40) and (int(PercentageScore.get()) <= 100):
            PercentageValidation.set(True)
        else:
            messagebox.showerror("Error", "Enter Percentage between 40 and 100")
    elif (PercentageScore.get() == " ") or ((PercentageScore.get()).isdigit() ==
False):
        messagebox.showerror("Error", "Enter Percentage between 40 and 100")
    if (PercentageValidation.get() == True) and (ACTValidation.get() == True):
        ## =====
        # Validated percentage will be converted to GPA Score
        PercentagetogPA = (int(PercentageScore.get()) / 20) - 1
        for i in range(len(list1)):
            if MajorChoice.get() == list1[i]:
                MajorEquivalentVal.set((MajEquivalent[i]))
        if NoOfFeatures == 5:
            Prob = np.array([ACTScore.get(), PercentagetogPA, ResidencyStatus.get(),
MajorEquivalentVal.get()])[
                np.newaxis]
            Prob = scaler.transform(Prob)
            probability = float(pr.predict(Theta1, Theta2, Prob)[3])
            message = ("The probability of getting admission into " +
hp.CollegeChoice.get() + " is " + str(
                round(probability * 100, 2)) + "%")
            messagebox.showinfo("Probability", message)
        else:
            Prob = np.array([ACTScore.get(), PercentagetogPA,
MajorEquivalentVal.get()])[np.newaxis]
            Prob = scaler.transform(Prob)
            probability = float(pr.predict(Theta1, Theta2, Prob)[3])
            message = ("The probability of getting admission into " +
hp.CollegeChoice.get() + " is " + str(
                round(probability * 100, 2)) + "%")
            messagebox.showinfo("Probability", message)
        ## =====

# method is created, a function that will be utilized to create the button Go Back, in
case
# the user would like to go back the the Home Page
def method():
    python = sys.executable
    os.execl(python, python, *sys.argv)

# String Variables are initialized so as to to store the Major and Residency status
that will
# be chosen by the user
MajorChoice = tk.StringVar()
ResidencyStatus = tk.IntVar()
# Residency status is by default set to 9, the integer equivalent mapped to In-state
Residency
ResidencyStatus.set(9)
## =====
# This is done to get the the chosen college's list of major's integer equivalent
mapped
# to the list of majors
size = X.shape[1]
List = X[:, size - 4]
list2 = X[:, size - 3]
MajEquivalent = np.unique(list2)
list1 = np.unique(List)
Majorlist = list1.tolist()

```

```

## =====

## =====
## =====
# Based on the Test type and Score type chosen by the user in the Home Page,
# the appropriate window is created to display the appropriate labels, entry
# widgets.

# Configurations are done to ensure that the values entered in the entry field are
limited
# to integer values

# In addition, based on the college chosen, the type of college is detected i.e. public
or private,
# using the number of features in the featuresdata file of the college. If the no of
features is 5,
# a label and radia button is added to choose the residency status (as it is a public
college), else
# it is omitted
## =====
if (hp.Testtype.get() == "SAT") and (hp.Scoretype.get() == "IB"):
    reg = root.register(callback)
    label1 = tk.Label(text="Enter SAT Score", font=LabelFont)
    label1.grid(row=2, column=0, pady=30, padx=10)

    SATScore.grid(row=2, column=1)
    SATScoreTooltip = CreateToolTip(SATScore, 'Enter numeric digits only')

    SATScore.config(validate="key",
                    validatecommand=(reg, '%P'))
    label2 = tk.Label(text="Enter your IB Score ( /42)", font=LabelFont)
    label2.grid(row=3, column=0, padx=10, pady=20)

    IBScore.grid(row=3, column=1)
    IBScoreTooltip = CreateToolTip(IBScore, 'Enter numeric digits only')
    IBScore.config(validate="key",
                  validatecommand=(reg, '%P'))
    label3 = tk.Label(text="Choose your Major", font=LabelFont)
    label3.grid(row=4, column=0, padx=10, pady=20)
    Majors = ttk.Combobox(state="readonly", textvariable=MajorChoice, values=Majorlist,
width=35)
    Majors.current(0)
    Majors.grid(row=4, column=1)
    if NoOfFeatures == 5:
        label4 = tk.Label(text="Choose Residency Status", font=LabelFont)
        label4.grid(row=5, column=0, pady=20, padx=10)
        Radiobutton(text="In-State", variable=ResidencyStatus, value=9).grid(row=5,
column=1)
        Radiobutton(text="Out-of-State", variable=ResidencyStatus,
value=10).grid(row=5, column=2)
        Calculate = tk.Button(text="Calculate Chance ", height=1, width=14,
font=ButtonFont,
                           command=CalcProbwithSATandIB)
        Calculate.grid(row=6, column=1, pady=10)
        GoBack = tk.Button(text="Click to go back ", height=1, width=14, font=ButtonFont,
command=method)
        GoBack.grid(row=6, column=0, pady=10)
if (hp.Testtype.get() == "ACT") and (hp.Scoretype.get() == "IB"):
    reg = root.register(callback)
    label1 = tk.Label(text="Enter ACT Score", font=LabelFont)
    label1.grid(row=2, column=0, pady=30, padx=10)

    ACTScore.grid(row=2, column=1)
    ACTScoreTooltip = CreateToolTip(ACTScore, 'Enter numeric digits only')
    ACTScore.config(validate="key",
                    validatecommand=(reg, '%P'))
    label2 = tk.Label(text="Enter your IB Score ( /42)", font=LabelFont)

```



```

label2.grid(row=3, column=0, padx=10, pady=20)

IBScore.grid(row=3, column=1)
IBScoreTooltip = CreateToolTip(IBScore, 'Enter numeric digits only')
IBScore.config(validate="key",
                validatecommand=(reg, '%P'))
label3 = tk.Label(text="Choose your Major", font=LabelFont)
label3.grid(row=4, column=0, padx=10, pady=20)
Majors = ttk.Combobox(state="readonly", textvariable=MajorChoice, values=Majorlist,
width=35)
Majors.current(0)
Majors.grid(row=4, column=1)
if NoOfFeatures == 5:
    label4 = tk.Label(text="Choose Residency Status", font=LabelFont)
    label4.grid(row=5, column=0, pady=20, padx=10)
    Radiobutton(text="In-State", variable=ResidencyStatus, value=9).grid(row=5,
column=1)
    Radiobutton(text="Out-of-State", variable=ResidencyStatus,
value=10).grid(row=5, column=2)
    Calculate = tk.Button(text="Calculate Chance ", height=1, width=14,
font=ButtonFont,
                        command=CalcProbwithACTandIB)
    Calculate.grid(row=6, column=1, pady=10)
    GoBack = tk.Button(text="Click to go back ", height=1, width=14, font=ButtonFont,
command=method)
    GoBack.grid(row=6, column=0, pady=10)
if (hp.Testtype.get() == "SAT") and (hp.Scoretype.get() == "GPA"):
    reg = root.register(callback)
    label1 = tk.Label(text="Enter SAT Score", font=LabelFont)
    label1.grid(row=2, column=0, pady=30, padx=10)

    SATScore.grid(row=2, column=1)
    SATScoreTooltip = CreateToolTip(SATScore, 'Enter numeric digits only')
    SATScore.config(validate="key",
                    validatecommand=(reg, '%P'))
    label2 = tk.Label(text="Enter your Percentage ( /100)", font=LabelFont)
    label2.grid(row=3, column=0, padx=10, pady=20)

    PercentageScore.grid(row=3, column=1)
    PercentageScoreTooltip = CreateToolTip(PercentageScore, 'Enter numeric digits
only')
    PercentageScore.config(validate="key",
                          validatecommand=(reg, '%P'))
    label3 = tk.Label(text="Choose your Major", font=LabelFont)
    label3.grid(row=4, column=0, padx=10, pady=20)
    Majors = ttk.Combobox(state="readonly", textvariable=MajorChoice, values=Majorlist,
width=35)
    Majors.current(0)
    Majors.grid(row=4, column=1)
    if NoOfFeatures == 5:
        label4 = tk.Label(text="Choose Residency Status", font=LabelFont)
        label4.grid(row=5, column=0, pady=20, padx=10)
        Radiobutton(text="In-State", variable=ResidencyStatus, value=9).grid(row=5,
column=1)
        Radiobutton(text="Out-of-State", variable=ResidencyStatus,
value=10).grid(row=5, column=2)
        Calculate = tk.Button(text="Calculate Chance ", height=1, width=14,
font=ButtonFont,
                            command=CalcProbwithSATandPercentage)
        Calculate.grid(row=6, column=1, pady=10)
        GoBack = tk.Button(text="Click to go back ", height=1, width=14, font=ButtonFont,
command=method)
        GoBack.grid(row=6, column=0, pady=10)
if (hp.Testtype.get() == "ACT") and (hp.Scoretype.get() == "GPA"):
    reg = root.register(callback)
    label1 = tk.Label(text="Enter ACT Score", font=LabelFont)
    label1.grid(row=2, column=0, pady=30, padx=10)

```

```

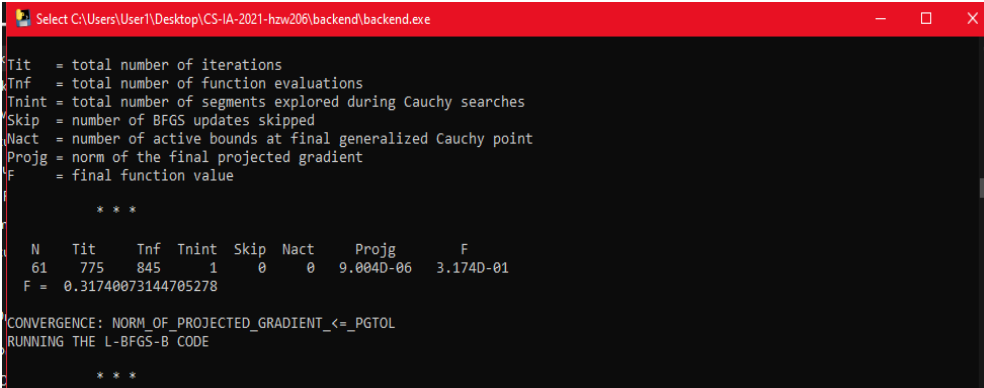
ACTScore.grid(row=2, column=1)
ACTScoreTooltip = CreateToolTip(ACTScore, 'Enter numeric digits only')
ACTScore.config(validate="key",
                 validatecommand=(reg, '%P'))
label2 = tk.Label(text="Enter your Percentage ( /100)", font=LabelFont)
label2.grid(row=3, column=0, padx=10, pady=20)

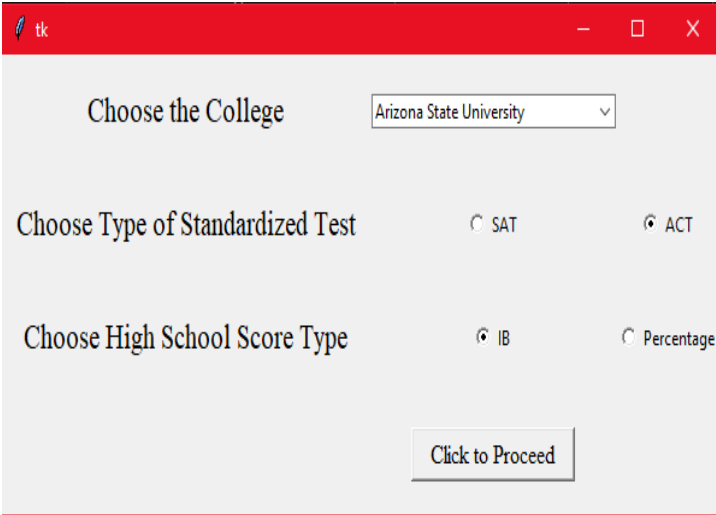
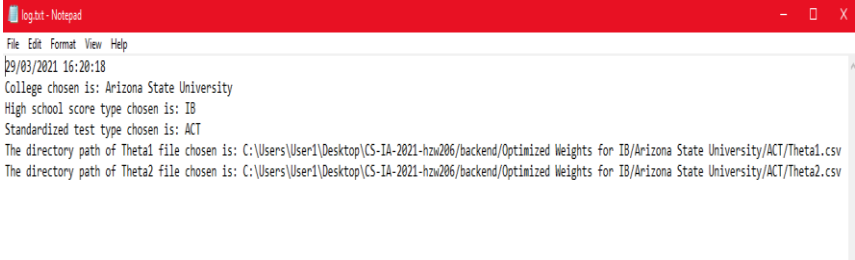
PercentageScore.grid(row=3, column=1)
PercentageScoreTooltip = CreateToolTip(PercentageScore, 'Enter numeric digits
only')
PercentageScore.config(validate="key",
                      validatecommand=(reg, '%P'))
label3 = tk.Label(text="Choose your Major", font=LabelFont)
label3.grid(row=4, column=0, padx=10, pady=20)
Majors = ttk.Combobox(state="readonly", textvariable=MajorChoice, values=Majorlist,
width=35)
Majors.current(0)
Majors.grid(row=4, column=1)
if NoOfFeatures == 5:
    label4 = tk.Label(text="Choose Residency Status", font=LabelFont)
    label4.grid(row=5, column=0, pady=20, padx=10)
    Radiobutton(text="In-State", variable=ResidencyStatus, value=9).grid(row=5,
column=1)
    Radiobutton(text="Out-of-State", variable=ResidencyStatus,
value=10).grid(row=5, column=2)
    Calculate = tk.Button(text="Calculate Chance ", height=1, width=14,
font=ButtonFont,
                      command=CalcProbwithACTandPercentage)
    Calculate.grid(row=6, column=1, pady=10)
    GoBack = tk.Button(text="Click to go back ", height=1, width=14, font=ButtonFont,
command=method)
    GoBack.grid(row=6, column=0, pady=10)
## =====
## =====
# Creating method that would quit the application in case the close button
# is clicked
def on_closing():
    if messagebox.askokcancel('Quit', 'Do you want to quit?'):
        root.destroy()
root.protocol('WM_DELETE_WINDOW', on_closing)
## =====
# Creating log file to store the directory path of Theta1, Theta2 and the
# high school score type, standardized test type chosen by the user
now = datetime.now()
# dd/mm/YY H:M:S
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
log = open("log.txt", "a") # write mode
log.write(dt_string + '\n')
log.write("College chosen is: " + hp.CollegeChoice.get() + '\n')
if hp.Scoretype.get() == 'GPA':
    log.write("High school score type chosen is: " + 'Percentage' + '\n')
else:
    log.write("High school score type chosen is: " + hp.Scoretype.get() + '\n')
log.write("Standardized test type chosen is: " + hp.Testtype.get() + '\n')
log.write("The directory path of Theta1 file chosen is: " + Theta1Dir + '\n')
log.write("The directory path of Theta2 file chosen is: " + Theta2Dir + '\n' + '\n')
log.close()
## =====
root.mainloop()

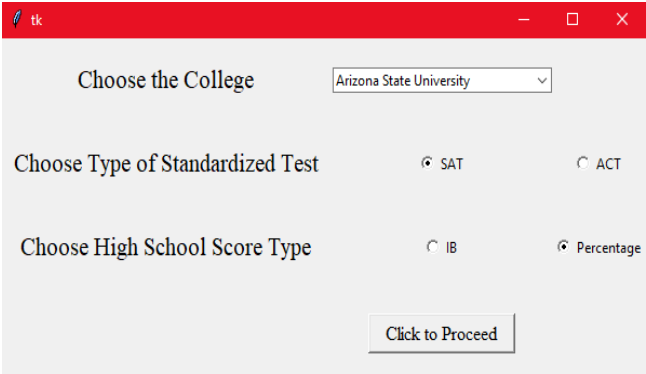
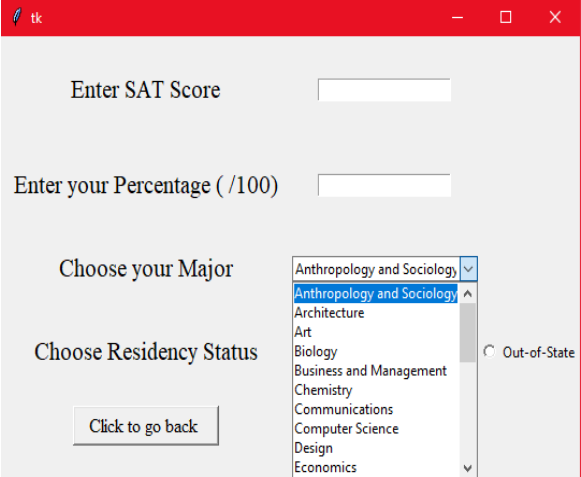
```

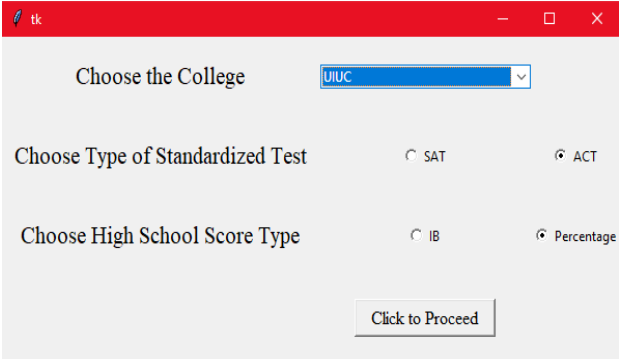
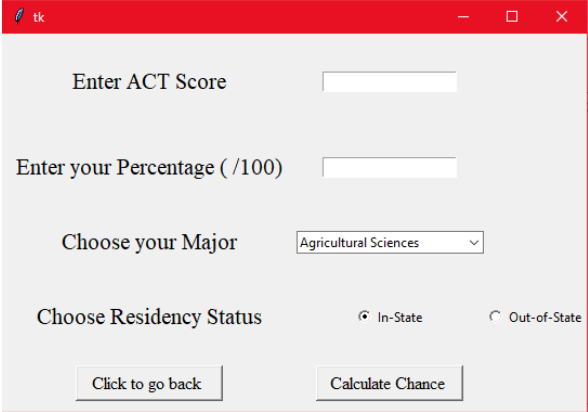
Appendix E:

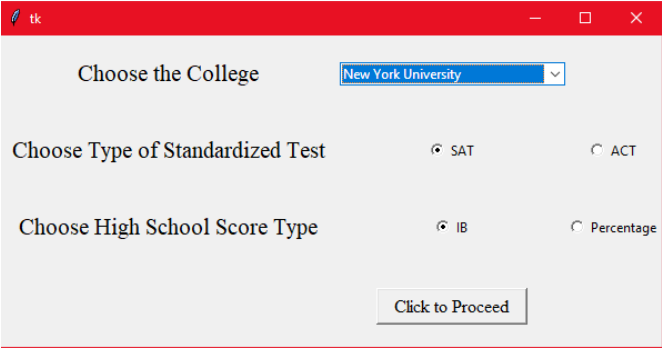
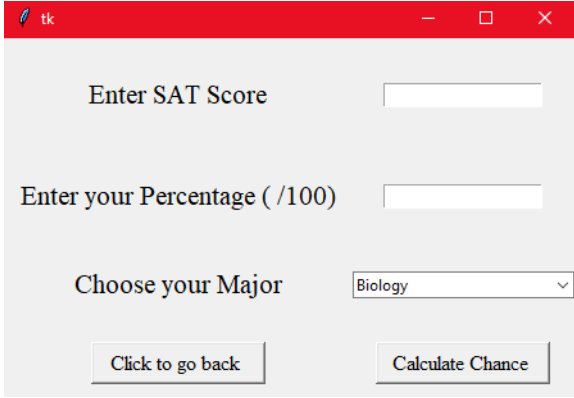
Completed Tasks

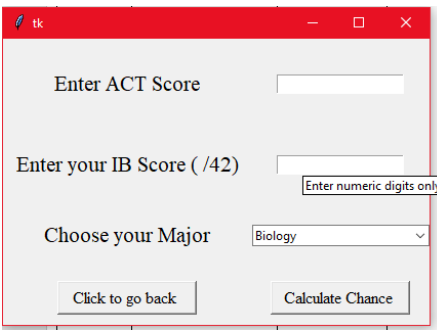
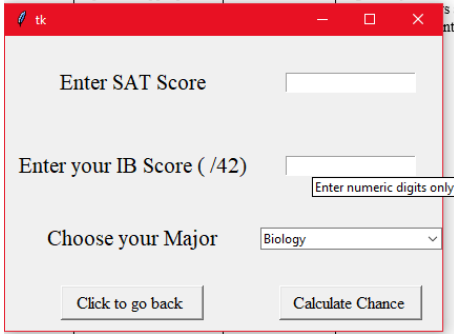
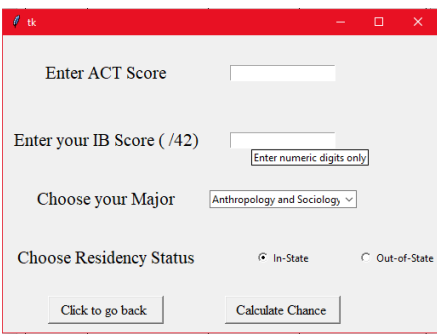
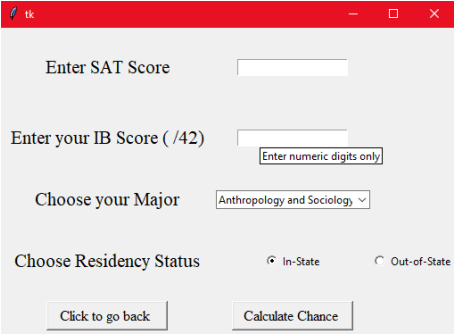
Test Number	Test description	Test data	Expected Outcome	Actual Outcome	Pass/Fail
1	Requirement 1:  When the backend.exe file is launched, the cost function for the colleges should always decrease until convergence is met	Training dataset for a college for which the cost function is to be minimized.	Convergence:  Norm_Of_Projected_Gradient_<=_Pgtol  i.e. convergence is met		Pass

2	Requirement 2:  Automatically load the weights (Theta1 and Theta2) to be used for calculation based on user's selection of college, standardized test type and high school exam score type.	Choose any college, high school score type, standardized test type.	Correct path of Theta1.csv and Theta2.csv files based on the test data should be present in the 'log.txt' file.	<div></div> <div></div>	Pass
---	---	---	---	--	------

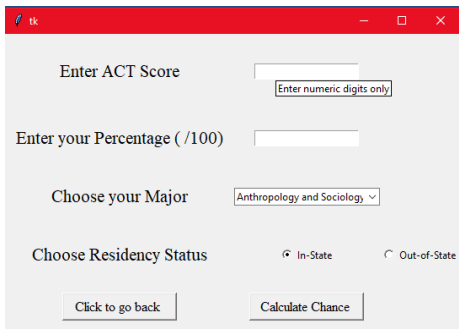
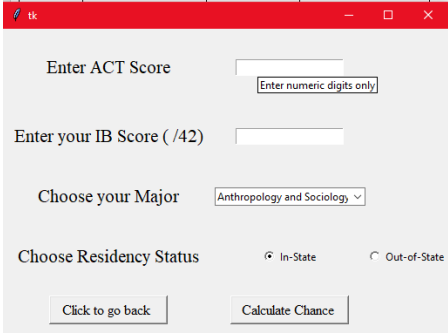
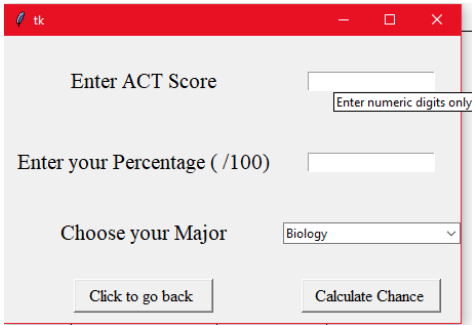
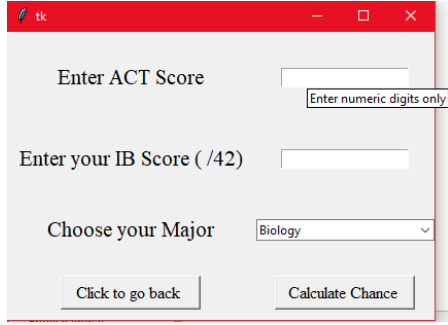
3	<p>Requirement 3:</p> <p>Automatically load list of majors (drop-down list) to choose choice of major based on user's selection of college.</p> <p>.</p>	<p>Choose a college and any high school test, standardized score type.</p>	<p>Based on availability of majors of the college, regardless of the high school test, standardized score type, appropriate major list should be displayed.</p>	<div></div> <div></div>	<p>Pass</p>
---	--	--	---	--	-------------

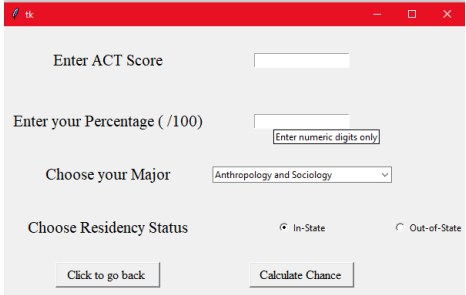
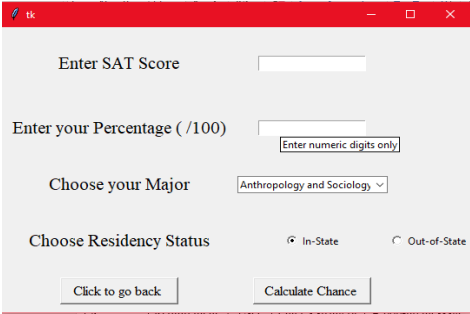
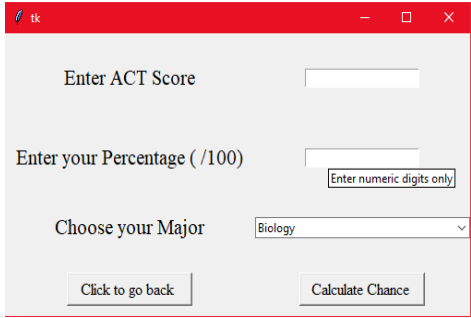
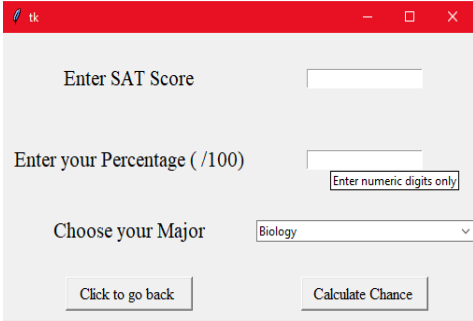
4	Requirement 4:  Automatically load the menu to choose In-State/Out-of-State Status based on the college selected.	Choose a public college.	Radio button to choose In-state/Out-of-state residency should appear on the next screen.	<div></div> <div></div>	Pass
---	---	--------------------------	--	--	------

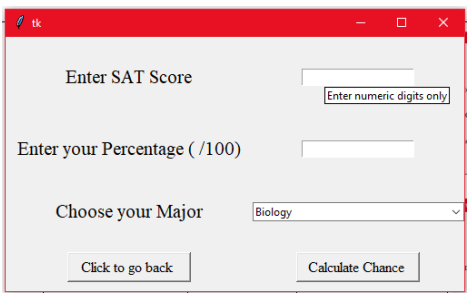
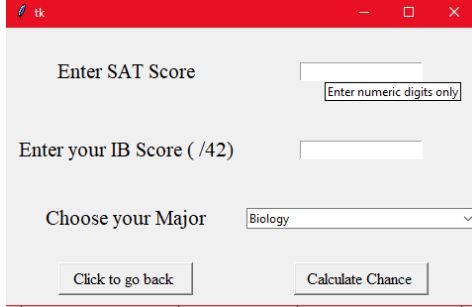
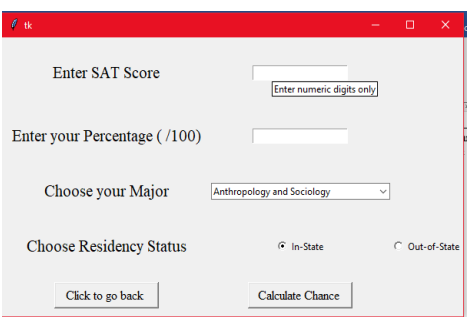
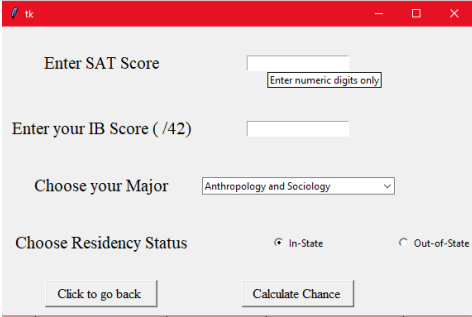
5	Requirement 4:  Automatically load the menu to choose In-State/Out-of-State Status based on the college selected.	Choose a private college.	Radio button to choose In-state/Out-of-state residency should not appear on the next screen.	<div></div> <div></div>	Pass
---	---	---------------------------	--	--	------

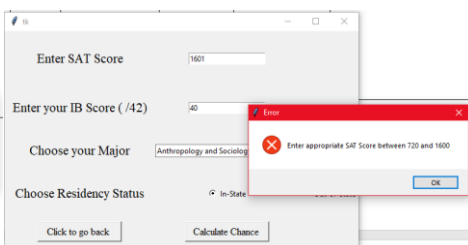
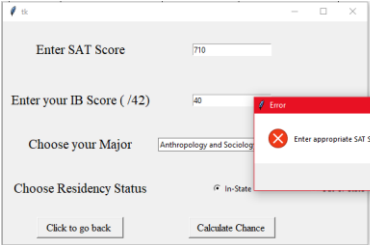
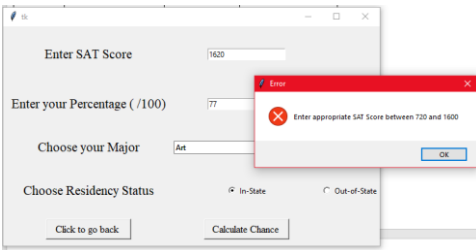
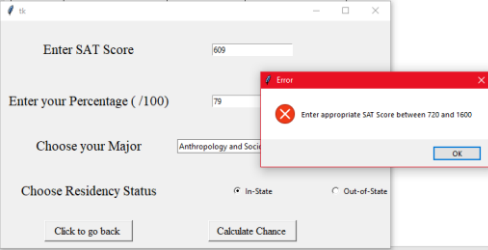
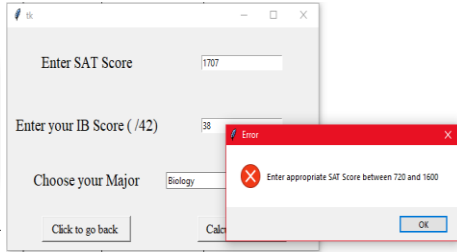
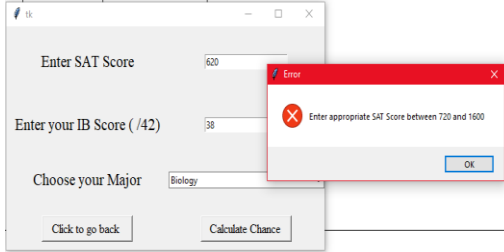
6	Requirement 5: User should be able to input appropriate values for IB Predicted Score.	Enter a string or non-integer character.	A tooltip should appear saying “Enter numeric digits only” as non-integer characters are restricted from entry.	<div></div>	Pass
---	--	--	---	---	------

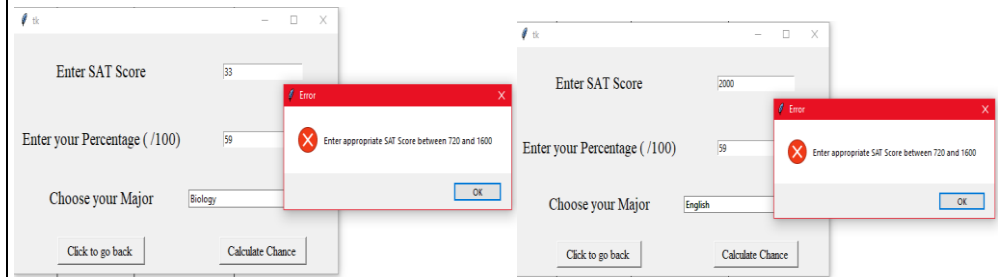


7	Requirement 5: User should be able to input appropriate values for ACT Score.	Enter a string or non-integer character.	A tooltip should appear saying “Enter numeric digits only” as non-integer characters are restricted from entry.	<div></div> <div></div>	Pass
---	---	--	---	---	------

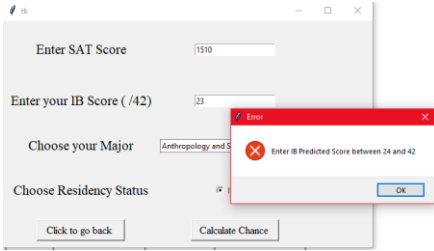
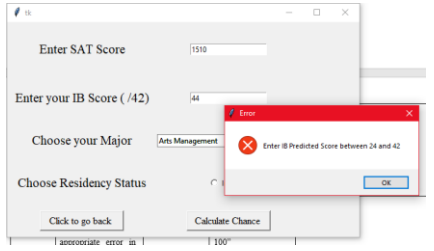
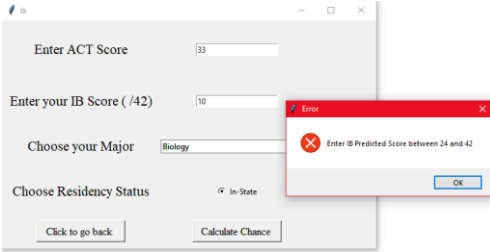
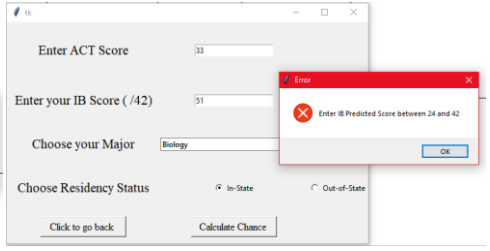
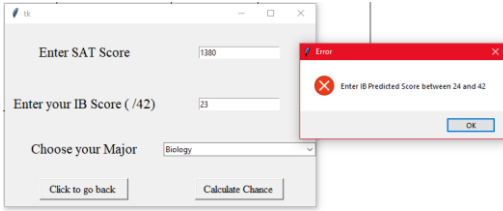
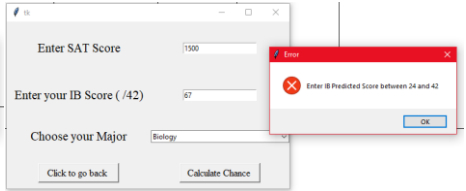
8	Requirement 5: User should be able to input appropriate values for Percentage score.	Enter a string or non-integer character.	A tooltip should appear saying “Enter numeric digits only” as non-integer characters are restricted from entry.	<div></div> <div></div>	Pass
---	--	--	---	---	------

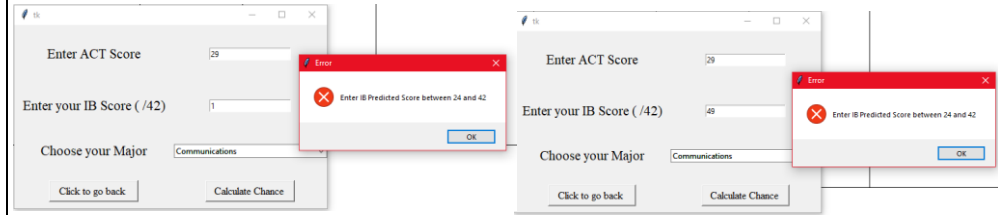
9	Requirement 5: User should be able to input appropriate values for SAT Score.	Enter a string or non-integer character.	A tooltip should appear saying “Enter numeric digits only” as non-integer characters are restricted from entry.	<div></div>	Pass
---	---	--	---	---	------

10	<p>Requirement 6:</p> <p>Validate user input for each parameter and display appropriate error in case validation returns false.</p>	<p>Enter a SAT Score not in the range of 720-1600 or not in the multiple of 10</p>	<p>A pop-up message should appear saying “Enter appropriate SAT Score between 720 and 1600”</p>	<div></div> <div></div> <div></div>	Pass
----	---	--	---	---	------

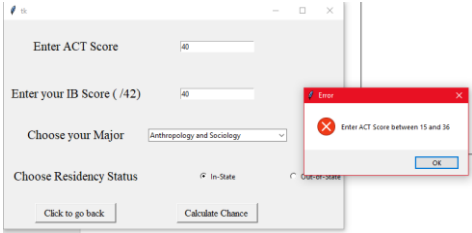
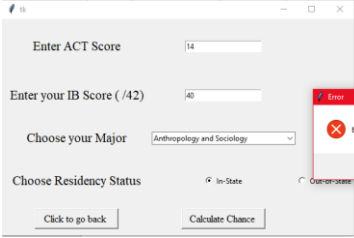
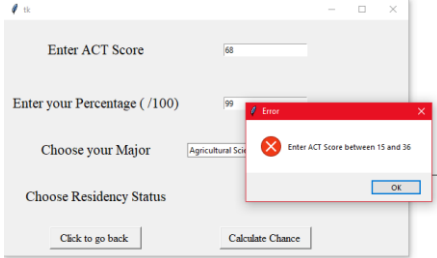
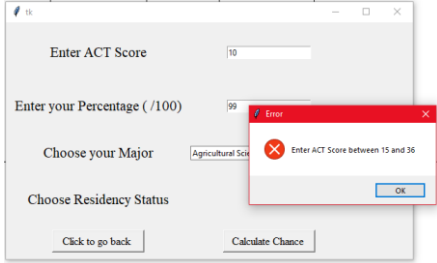
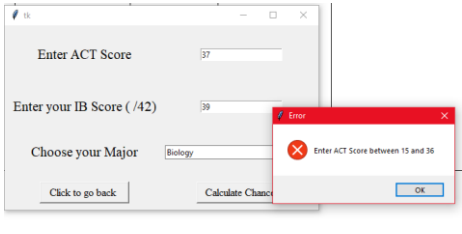
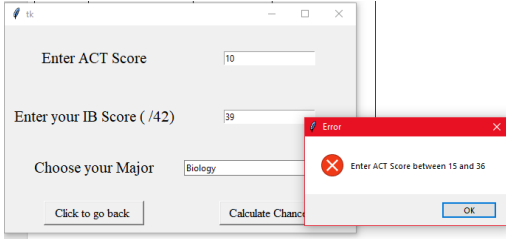


All possible combinations for a private and public college are considered

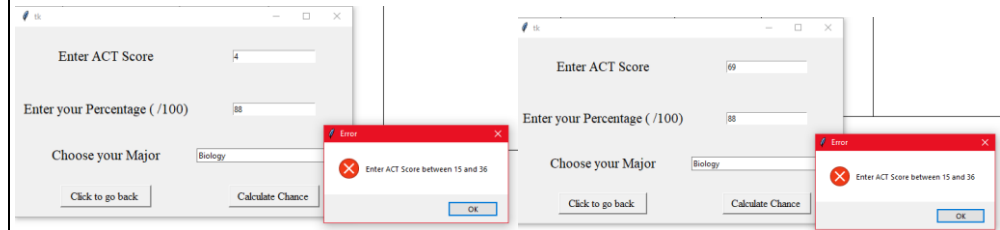
11	<p>Requirement 6:</p> <p>Validate user input for each parameter and display appropriate error in case validation returns false.</p>	<p>Enter a IB Score not in the range of 24-42</p>	<p>A pop-up message should appear saying “Enter IB Predicted Score between 24 and 42”</p>	<div><p>Enter SAT Score: 1510 Enter your IB Score (/42): 23 Choose your Major: Anthropology and S Choose Residency Status:  Click to go back Calculate Chance</p><p>Error: Enter IB Predicted Score between 24 and 42</p></div> <div><p>Enter SAT Score: 1510 Enter your IB Score (/42): 44 Choose your Major: Arts Management Choose Residency Status:  Click to go back Calculate Chance</p><p>Error: Enter IB Predicted Score between 24 and 42</p></div> <div><p>Enter ACT Score: 33 Enter your IB Score (/42): 10 Choose your Major: Biology Choose Residency Status: In-State Click to go back Calculate Chance</p><p>Error: Enter IB Predicted Score between 24 and 42</p></div> <div><p>Enter ACT Score: 33 Enter your IB Score (/42): 51 Choose your Major: Biology Choose Residency Status: In-State Out-of-State Click to go back Calculate Chance</p><p>Error: Enter IB Predicted Score between 24 and 42</p></div> <div><p>Enter SAT Score: 1380 Enter your IB Score (/42): 23 Choose your Major: Biology Click to go back Calculate Chance</p><p>Error: Enter IB Predicted Score between 24 and 42</p></div> <div><p>Enter SAT Score: 1500 Enter your IB Score (/42): 67 Choose your Major: Biology Click to go back Calculate Chance</p><p>Error: Enter IB Predicted Score between 24 and 42</p></div>	Pass
----	---	---	---	---	------



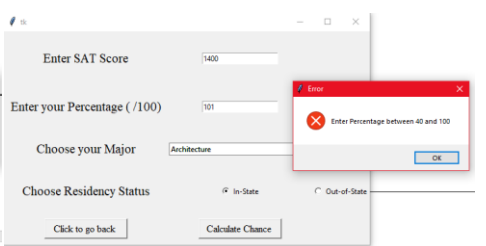
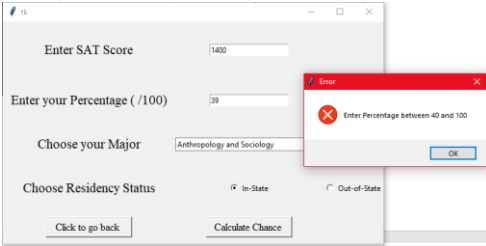
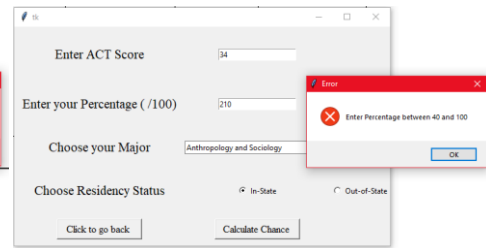
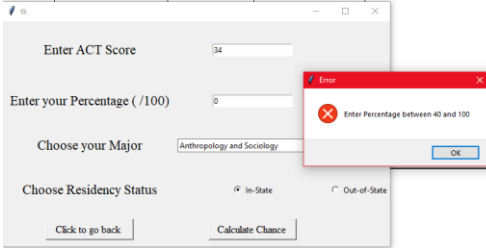
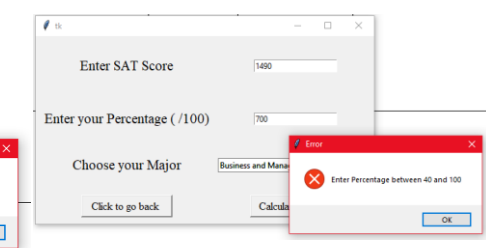
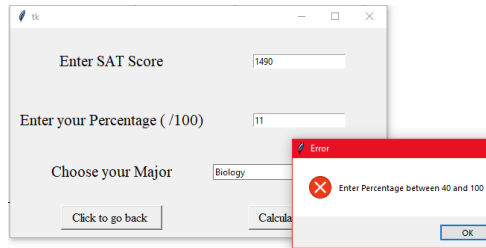
All possible combinations for a private and public college are considered

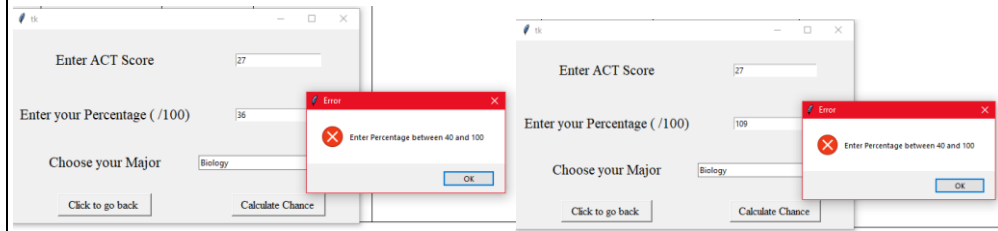
12	Requirement 6:  Validate user input for each parameter and display appropriate error in case validation returns false.	Enter an ACT Score not in the range of 15-36	A pop-up message should appear saying “Enter ACT Score between 15 and 36”	<div></div> <div></div> <div></div>	Pass
----	--	--	---	---	------



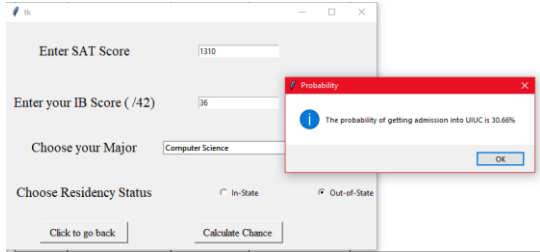
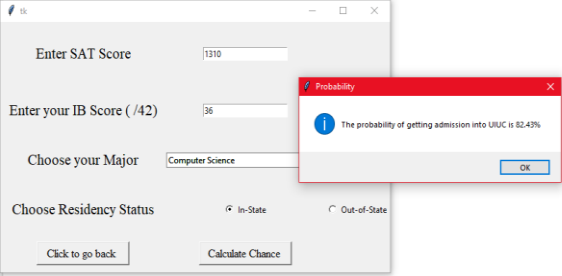


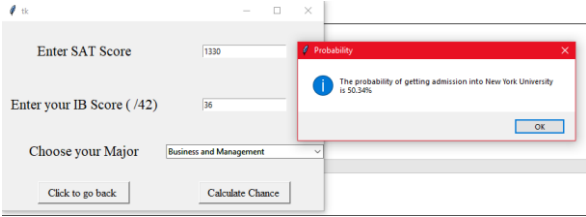
All possible combinations for a private and public college are considered

13	Requirement 6:  Validate user input for each parameter and display appropriate error in case validation returns false.	Enter a Percentage  Score not in the range of 40-100	A pop-up message should appear saying “Enter Percentage Score between 40 and 100”	<div></div> <div></div> <div></div>	Pass
----	--	--	---	---	------



All possible combinations for a private and public college are considered

14	Requirement 7: Based on inputs, calculate and display the probability of securing admission into user's choice of college.	Choose any public college, any high school test type, standardized score type, choice of major, and residency status.	A pop-up message should appear saying “ The probability of admission into [College Name] is [XX.XX] %.	<div></div> <p>The similar output was displayed for other high school test type, standardized score type, choice of major, and residency status for the public college</p>	Pass
----	--	---	--	--	------

15	Requirement 7: Based on inputs, calculate and display the probability of securing admission into user's choice of college.	Choose any private college, any high school test type, standardized score type, choice of major.	A pop-up message should appear saying “ The probability of admission into [University Name] is [XX.XX] %.	<div></div> <p>The similar output was displayed for other high school test type, standardized score type, choice of major, and private college.</p>	Pass
----	--	--	---	--	------