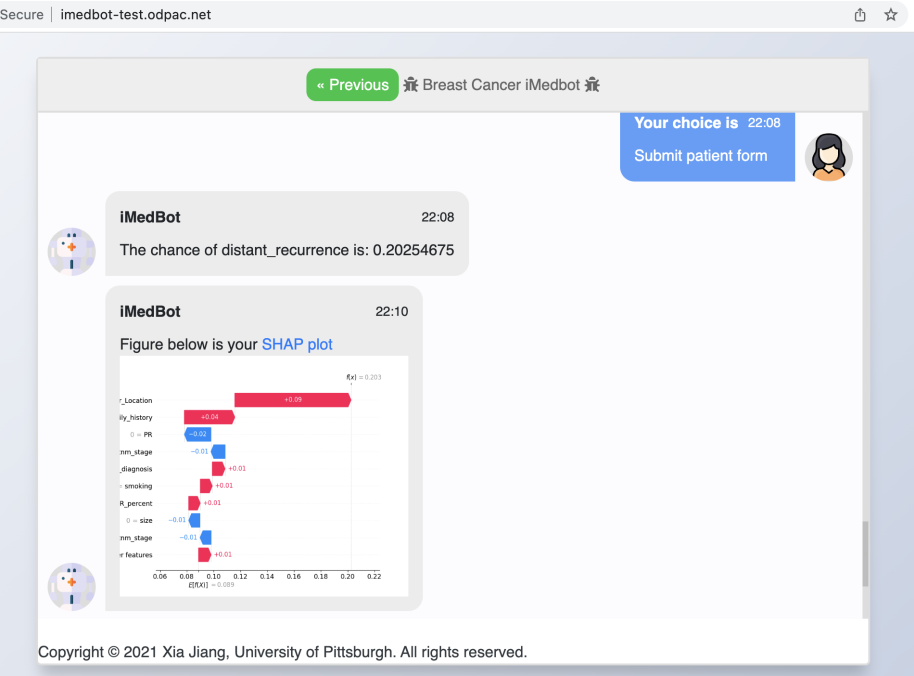
**To fix the timeout error in shap plot:**

if shap\_check == "true":  
 @after\_this\_request  
 def plot\_shap(response):  
 @response.call\_on\_close  
 def process\_after\_request():  
 print("long\_running\_task ", img\_src)  
  
 def f(X):  
 # return best\_model.predict(X).flatten()  
 # print("++++++++++++++++++++++++")  
 result = []  
 for item in X:  
 prob = user\_training\_model.predict\_proba(item.reshape(1, len(predset[0])))  
 # print(prob)  
 # print(prob[0][0])  
 result.append(prob[0][0])  
 print(np.array(result))  
 # the reason why we have 5 results is because we use kmeans to shrink the x\_cv(background dataset) dataset to only 5 samples  
 # [0.4565038 0.3262849 0.3953898 0.23958007 0.3785722]  
  
 print(type(result))  
 return np.array(result)  
  
 # shap.kmeans(data, K) to summarize the background as K samples, in our case it transfer  
 X\_train\_summary = shap.kmeans(X\_CV, 1)  
 print(X\_train\_summary)  
 # < shap.utils.\_legacy.DenseData object at 0x0000024682E412B0 >  
 print("111111111111111111111111111111111111111")  
 explainer = shap.KernelExplainer(f, X\_train\_summary)  
 print("222222222222222222222222222222222222222")  
 shap\_values = explainer.shap\_values(np.array([category\_list]))  
 print("333333333333333333333333333333333333333")  
 print(shap\_values)  
 print(explainer.expected\_value)  
 plt.clf()  
 # shap.waterfall\_plot(shap.Explanation(values=shap\_values, base\_values=explainer.expected\_value, data=np.array([category\_list]),feature\_names=X\_columns))  
 shap.waterfall\_plot(shap.Explanation(values=shap\_values[0], base\_values=explainer.expected\_value,  
 data=np.array([category\_list])[0], feature\_names=X\_columns))  
  
 if os.path.exists(img\_src):  
 os.remove(img\_src)  
 if os.path.exists(img\_src):  
 print("png exist")  
 else:  
 print("png does not exist")  
 print(img\_src)  
 plt.savefig(img\_src)  
 plt.clf()  
 return response

I use @response.call\_on\_close to plot shap in the backend after the function return the probability and shap image path. And then in the frontend i let the website wait for 2 minutes for the completion of plot. Therefore, it prevents the timeout error of server side.

The result: the shap plot can now be shown not only in localhost but also in aws environment.



Error handling of training and dataset in the backend:

@application.route("/Examdataset", methods=['GET', 'POST'])  
def get\_model\_Examdataset():  
 if request.method == "POST":  
 datasetname = request.form.get('name')  
 print("data set name is ", datasetname)  
 try:  
 validation\_auc,img\_src = train\_mode(datasetname)  
 return {"auc":str(validation\_auc),"src":str(img\_src)}  
 except Exception as e:  
 print(e)  
 return {"auc":"error","src":str(e)}

I catch the error in the training process and if there is an error it can return the error type to frontend and show it to users.

The result:

When uploading data-00.200.0-2.txt as dataset, it failed last time and now the system can catch error when training.

