- **Data Preprocessing for Question 1-3**

a. I first import all the packages and specify the device that the model would be train and evaluate on.
b. I read in the csv file and extracted the diabetes column out as y and extracted other 21 columns out as predictors X. Since the data is carefully curated according to the spec sheet, I didn't check for NA values.
c. Since among predictors there are continuous variables like BMI and binary values like HighBP, I normalized the data to make the predictiors in the same scale. Note that when I ran the model on the data with only one predictor dropped for the extra credit, this technique was also used.
d. Then I did the 80-20 train-test split on the dataset with a random_state of 0 to set the train and test dataset. This is to let all model share the same train and test dataset, which is best for comparing model performance.
e. I changed the train and test data from numpy arrays to tensors. I also constructed the train tensor dataset and test tensor dataset and the corresponding loaders for mini-batch training with a batch size of 64.

- **train, test, and predict_proba function for Question 1-3. (No part c and d in this section because this section is not suppose to show any results or conclusions)**

1. train function
    a. For the train function, I set the parameter to be model, train_loader, and optimizer. Then I call train function on the model. Then I set the train loss to be 0 and write a for-loop to process each mini-batch. Inside each round, the data(predictor) and target(outcome variable) is first moved to the device. Then each gradient of the weight is set to zero. Then I pass the data to the model and get the output from the model. Then I compute the loss function using the output and target. Then I add the current loss to the variable train loss. Then I called the backward function on loss for the backpropagation and then I call the step function on optimizer. In the end, I divide the train loss by the number of batches and returns it.
    b. The reason I call train function on the model is to let the model know we are training it now and hence it could activat some fields. The reason that I set the parameter to model, train_loader, and optimizer is that I design this train function to take any models from question 1-3, the train dataset, and the optimizer of any kinds. The reason I write the for-loop because it is used for mini batches. Each batch contains 64 lines. In this way, the model could be trained batch by batch with more precision on the optimizer's operation on the weights. The reason I called the zero_grad() function on the optimizer is that it will set the previous gradient on each weight and bias to be 0 and hence when the gradient is accumulated by the nature of pytorch, it would not overcount. The reason I called the nll_loss is that cross-entropy loss function is a good way to solve classification problems, and this is a classification problem. The reason I called

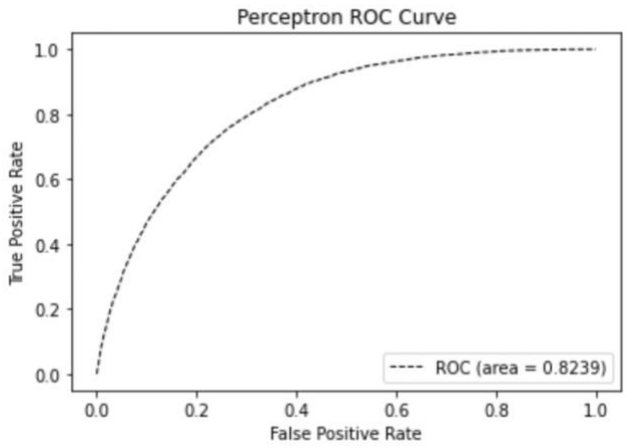
backward is to compute the direction that each weight should move when in gradient descent and do the back propagation. The reason I called step is to actually apply those direction moves on the weights. The reason I devide the train_loss by the number of batches is to calculate the average loss per data to present.

2. test function
    a. For the test function, I set the parameter to be model, test_loader, and optimizer. Then I call the eval() function on the model. Then I set the test loss to be 0 and write a for-loop to process each mini-batch under the no_grad condition. Inside each round, the data(predictor) and target(outcome variable) is first moved to the device. Then I pass the data to the model and get the output from the model. Then I compute the loss function using the output and target. Then I add the current loss to the test loss. Then I collect the correct classification on the target data. In the end, I divide the train loss by the number of batches and compute the accuracy and return it.
    b. The reason I call eval() function on the model is to let the model know we are testing it now and hence it could activate some fields. The reason that I set the parameter to model, test_loader, and optimizer is that I design this test function to take any models from question 1-3, the test dataset, and the optimizer of any kinds. The reason set the torch to no_grad() is because we do not need to compute the grad when testing. This is to make the testing process more efficient. I write the for-loop because it is used for mini batches. Each batch contains 64 lines. In this way, the model could be tested batch by batch with more precision on the optimizer's operation on the weights. The reason I called the nll_loss is that cross-entropy loss function is a good way to solve classification problems, and this is a classification problem. The reason I collect the correct classification is to better compute accuracy. The reason I divide the test_loss by the number of batches is to calculate the average loss per data to present. The reason I calculate accuracy in the end is just to print it and add some intuition and visualization in the testing process.
3. predict_proba function
    a. I set the parameter to be model and test data. I pass the whole test data to the model and get the output from the model. I then transform the output to a numpy array and then apply an exponential function on it.
    b. The reason I set the parameter to be model and test data is because this model should be a trained model solving classification problem, and this function should take the trained model and the test dataset as an input and returns the probability of each class (outcome variable) of each data in the dataset. The reason I apply an exponential function on the output is that I assume the classification model would take logsoftmax() as the probability mapping function in the final layer and hence the output space would range from (-inf, 1). Therefore, to map it to the real probability space, I need to apply exponential function to transform the range to (0,1). This range would make using scikit metrics function easier as an input.

- **Question 1-3**

1. Build and train a Perceptron (one input layer, one output layer, no hidden layers and no activation functions) to classify diabetes from the rest of the dataset. What is the AUC of this model?
    a. I designed a perceptron class for this question. The perceptron class consists of a constructor method and a forward function. The network has two layers, a Linear layer and a logsoftmax layer. The forward function specifies the workflow of the data. It is first changed to a 1d data and feed into the network. Then, I instantiate this model with an input_size of 21 and output_size of 2, and I choose the stochastic gradient descent optimizer with learning rate of 0.01 and momentum of 0.5. Then I trained and tested the model for 100 epochs and print the train_loss, test_loss, and the accuracy every ten epochs. In the end, I used the predict_proba function to calculate the probability of class 1 from all of the test data, and use them to calculate the AUC and show the ROC curve.
    b. The reason that the network has no activation function is that this is only a perceptron, and hence no linear activation function is needed. The reason the network still has the logsoftmax layers is that this is classification problem, and logsoftmax would not do the learning part, and it will only map the output to a probability space. Hence, it is still needed. Note logsoftmax rather than the softmax is used to prevent overflow. The reason that the data is reshaped in the forward function is that the nueral network I build can only take in 1d data. The reason I instantiate the model with an input_size of 21 and output_size of 2 is that I would use 21 predictors to classifies the data, which has a total of two classes (0 and 1). Note that the model would transform the outcome variable to a one-hot vector for the classifcation problem. The reason I chose SGD optimizer with learning rate 0.01 and momentum 0.5 is because that SGD is a common optimizer and a learning rate of 0.01 is enough for the model to converge and a momentum of 0.5 could help us get out of the local maximum if met. The reason I trained the model for 100 epochs is that 100 epochs is enough to make the loss function converges within an acceptable error. The reason I print the accuracy is merely for showing intuition and visualization of the training/testing process. I use my predict_proba function because the probability is needed when computing AUC score. I computed the AUC score because it is a common metrics to validate the model of classification problem.
    c. The AUC of the perceptron is near 0.8239. An ROC curve is shown below to visualize it, where AUC is the area under the curve.

```
perceptron auc: 0.8239011526760722
```



   d. Note that this result aligns with the findings from the logistic regression model in my last homework, which has an AUC of 0.8252. This is correct because what logsoftmax(or softmax) basically does is to map a continuous output values to a probability space and then the discrete values, which is to generalize the logistic regression from binary classification to multiple classification. Overall, I think this model has a good performance when classifying whether a person has diabetes based on all the predictors because it has a rather high AUC score (between 0.8-0.9, and 1 is the highest possible value).

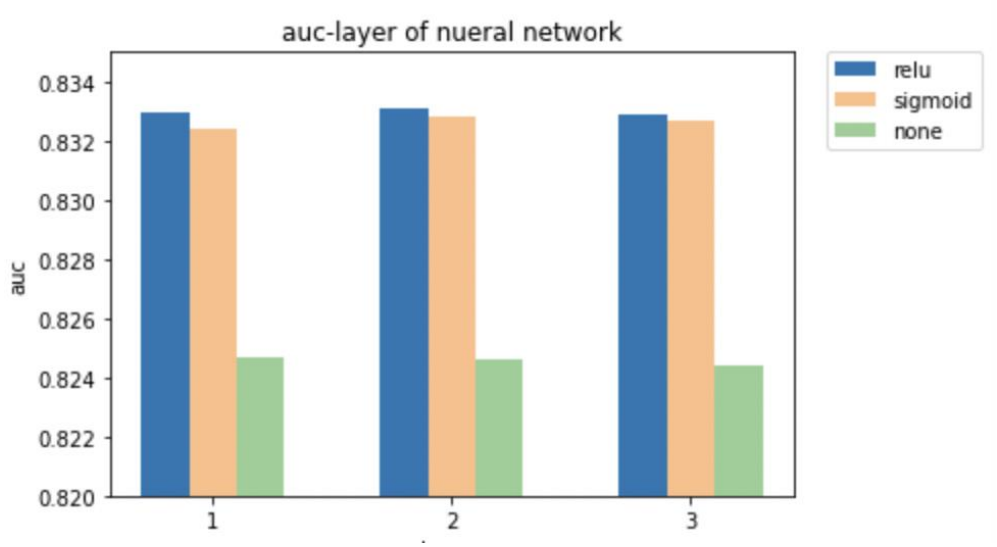
2. Build and train a feedforward neural network with at least one hidden layer to classify diabetes from the rest of the dataset. Make sure to try different numbers of hidden layers and different activation functions (at a minimum reLU and sigmoid). Doing so: How does AUC vary as a function of the number of hidden layers and is it dependent on the kind of activation function used (make sure to include "no activation function" in your comparison). How does this network perform relative to the Perceptron?
   a. I design the FC1, FC2, FC3 classes, each having the input layer size, hidden layer size, output layer size, and the activation function as parameters. The forward function in each class specifies the workflow of the input data. Basically, the data would first be transformed to 1d data, and then feed into the input layer, and then go to 1-3 hidden layers depending on the class with a certain hidden size, and then apply the activation function as specified, and then go to the output layer, and then apply the logsoftmax function. The difference between those classes of networks is the number of layers and the activation function specified before the output layer. Then, using these classes, I create 9 models, 3 layers * 3 activation functions (relu, sigmoid, and no activation function). Then I use the SGD optimizer with train and test function and train the 9 models with 101 test epoch and test it after each epoch. In the end, I use my predict_proba to compute the probability for each class the whole test data is predicted to be and compute and store the 9 auc score in a 3 x 3 arrays.
   b. The reason I design the FC1, FC2, FC3 classes with input layer size, hidden layer size, and output layer size parameters is that I want the user to have more

freedom of creating the FC network. The reason why the networks is design to be no activation function in between the hidden layers is that this is not a deep network problem, and we should sort of control variables for our analysis of different layers vs different activation functions. The reason I transformed data to 1d, and the way I used the SGD optimizer, and why I print the accuracy is the same as question 1. I also computed the AUC in this case because it is a good metric for classification method.

c. A 3x3 AUC matrix is shown below. Note that the column is from 1 to 3 hidden layers, and the row is the activation function used.

```
relu:  [0.8329807299477289, 0.8330695166958451, 0.8329082642344474]
sigmoid:  [0.8324333069865579, 0.8328442003891718, 0.8326560759691999]
none:  [0.8246734012773739, 0.824585873057078, 0.8244203049604077]
```

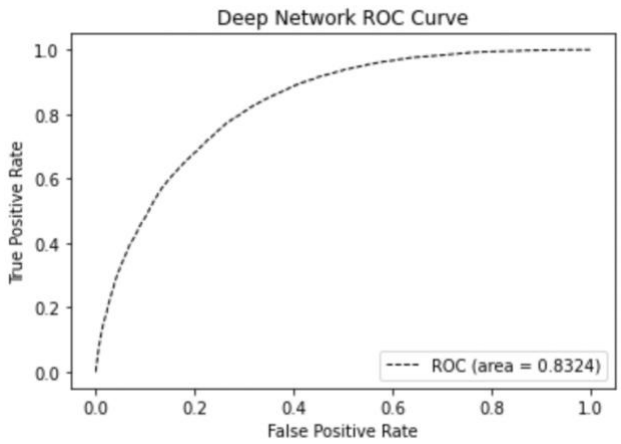
A graph is shown below for a clearer illustration.



d. According to the AUC matrix and the graph, we can approximate a function of AUC score regarding different layers and different activation. There seems to be no explicit relationships between the number of layers and the AUC. This is legit because I have not added any activation function in between, and so it is just doing a linear approximation process, and so the number of layers would not matter much. However, overall there is a huge difference between using activation functions and using no activation function (approximately 1% difference). This is reasonable because the neural network features the activation function, a network with no activation function for classification problem is no different than a perceptron, or a linear regression model. It is the activation function makes the model nonlinear and fit better to the test data with the learning ability. Note that using relu is slightly better than using sigmoid as activation function, this means that the AUC score would vary slightly depending on which activation function to use. This is reasonable because relu generally converges faster than sigmoid because it in nature has fewer gradient vanishing problems than sigmoid. Generally, using activation functions in FC network performs better than using a simple perceptron.

3. Build and train a “deep” network (at least 2 hidden layers) to classify diabetes from the rest of the dataset. Given the nature of this dataset, is there a benefit of using a CNN or RNN for the classification?
    a. I designed a deep network class for this question. The DeepFCLayer class consists of a constructor method and a forward function. The network has an input layer, two hidden layers with activation function in between, and a output layer with logsoftmax function. The forward function specifies the workflow of the data. It is first changed to a 1d data and feed into the network. Then, I instantiate this model with an input_size of 21 and output_size of 2, and I choose the stochastic gradient descent optimizer with learning rate of 0.01 and momentum of 0.5. Then I trained and tested the model for 100 epochs and print the train_loss, test_loss, and the accuracy every ten epochs. In the end, I used the predict_proba function to calculate the probability of class 1 from all of the test data, and use them to calculate the AUC and show the ROC curve.
    b. The reason that the network has 2 hidden layers with activation functions in between is that this is a deep learning, and adding activation function in between could help with the nonlinear approximation. The reason the network has the logsoftmax layers is that this is classification problem, and it will only map the output to a probability space. The reason that the data is reshaped in the forward function is that the neural network I build can only take in 1d data. The reason I instantiate the model with an input_size of 21 and output_size of 2 is that I would use 21 predictors to classifies the data, which has a total of two classes (0 and 1). The reason I chose SGD optimizer with learning rate 0.01 and momentum 0.5 is because that SGD is a common optimizer and a learning rate of 0.01 is enough for the model to converge and a momentum of 0.5 could help us get out of the local maximum if met. The reason I trained the model for 100 epochs is that 100 epochs is enough to make the loss function converges within an acceptable error. The reason I print the train loss, test loss, and accuracy is only for showing intuition and visualization of the training/testing process. I use my predict_proba function because the probability is needed when computing AUC score. I computed the AUC score because it is a common metric to validate the model of classification problem.
    c. The AUC of the perceptron is near 0.8324. An ROC curve is shown below to visualize it, where AUC is the area under the curve.

```
deep network auc:  0.8324155155451683
```



d. Overall, I think this model has a good performance when classifying whether a person has diabetes based on all the predictors because it has a rather high AUC score (between 0.8-0.9, and 1 is the highest possible value). Note that it seems this result is not better than the non-deep FC network in question 2, and this is reasonable. A deeper network does not necessarily mean it fit better to the data, there is a possibility of data overfitting. Also, given the nature of this dataset, there would not be any benefit to use CNN or RNN. CNN is for local feature extracting, however there is not local features when considering our data as a group, which means that our predictors columns of diabetes would not form a recognizable feature group when examined. RNN is for learning on sequence data, but our data has no sequence or time variable implicit casted in it. Hence, neither CNN nor RNN fits this classification.

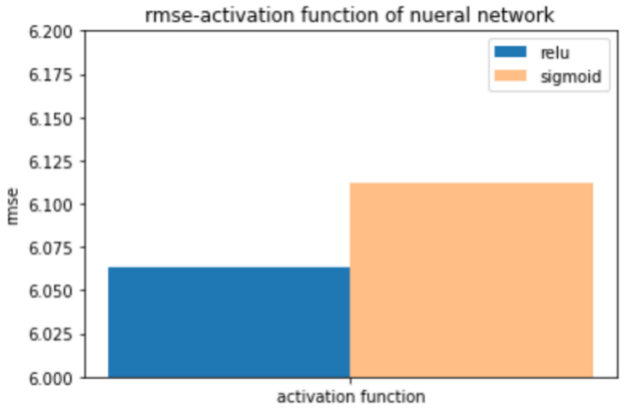
- **Data Preprocessing for Question 4-5**

a. I extracted the BMI column out as y and extracted other 21 columns out as predictors X. Since the data is carefully curated according to the spec sheet, I didn't check for NA values.
b. Since among predictors there are continuous variables like MentalHealth and binary values like HighBP, I normalized the data to make the predictiors in the same scale. Note that when I ran the model on the data with only one predictor dropped for the extra credit, this technique was also used.
c. Then I did the 80-20 train-test split on the dataset with a random_state of 0 to set the train and test dataset. This is to let all model share the same train and test dataset, which is best for comparing model performance.
d. I transformed the train and test data from numpy arrays to tensors.

- **Question 4-5**

4. Build and train a feedforward neural network with one hidden layer to predict BMI from the rest of the dataset. Use RMSE to assess the accuracy of your model. Does the RMSE depend on the activation function used?
    a. I design the FCLayer class with the input layer size, hidden layer size, output layer size, and the activation function as parameters. The network consists of an input layer, a hidden layer, an activation function and an output layer. No softmax function is applied. Basically, the data would first be transformed to 1d data, and then feed into the network for training and testing. I create 2 models based on this network with input layer size 21, hidden layer size 100, and output layer size 1, and one with relu activation function and the other one with sigmoid activation function. I used the SGD optimizer of a learning rate of 0.01 and momentum of 0.05 for both models. Then, I pass the train data to the model and train the model with mse_loss function, and the zero_grad(), backward(), step() function are applied. The training epoch is 1000. After each training epoch I computed the RMSE to better visualize the training process. In the end of the training, I pass the whole test predictor data to the trained model and get the predicted output and use it and the test outcome data to compute the RMSE. Two model's RMSE are then compared.
    b. The reason why the model has no softmax function is that this is a regression problem. The reason I set the output layer size to be 1 is because we want the model to give us one value, the predicted BMI. The reason I used the SGD optimizer with certain learning rate and momentum is the same as question 1. The reason I used the mse_loss function instead of nll_loss is that mse_loss function is designed to compute loss for a regression problem. And it has the same tendency of RMSE as specified in the problem description because RMSE is just the square root of MSE. The reason zero_grad(), backward(), step() function are applied is the same as question 1. The reason training epoch is 1000 is that this will converge the loss better. The RMSE is computed because RMSE is a good metric to validate the regression model.
    c. The RMSE using relu in the model is approximately 6.06 and the RMSE using sigmoid in the model is approximately 6.11. A graph is also plotted.

```
rmse_relu: 6.063618253197626
rmse_sigmoid: 6.112287982301386
```

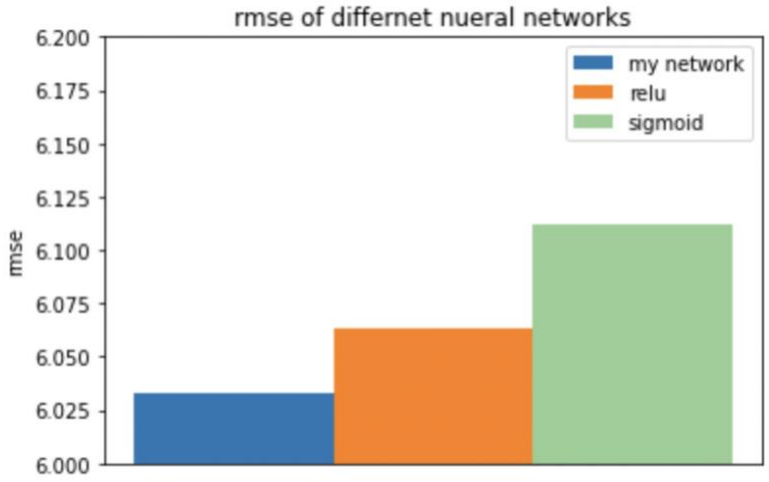


d. I think this result shows that RMSE depend on which activation function used because the model using relu has lower RMSE than sigmoid, hence the model with relu performs better. This is reasonable because relu generally converges faster than sigmoid because it in nature has fewer gradient vanishing problems than sigmoid. Note that the RMSE does not range from 0 to 1 because I did not normalize the outcome variable when doing the regression. It would not be reasonable if I normalize the outcome variable because in that way it would only predict the normalized BMI and I couldn't change the predicted BMI back to the real BMI, the data I really want.

5. Build and train a neural network of your choice to predict BMI from the rest of your dataset. How low can you get RMSE and what design choices does RMSE seem to depend on?
    a. I design the MyLayer class with the input layer size, hidden layer size, output layer size. The network consists of an input layer, 5 hidden layers, a relu activation function and an output layer. No softmax function is applied. Basically, the data would first be transformed to 1d data, and then feed into the network for training and testing. I create my_regression_model based on this network with input layer size 21, hidden layer size 10, and output layer size 1. I used the Adam optimizer of a learning rate of 0.1 and weight_decay of 1e-5. Then, I pass the train data to the model and train the model with mse_loss function, and the zero_grad(), backward(), step() function are applied. The training epoch is 1000. After each training epoch I computed the RMSE to better visualize the training process. In the end of the training, I pass the whole test predictor data to the trained model and get the predicted output and use it and the test outcome data to compute the RMSE. Two model's RMSE are then compared.
    b. The reason why the model has no softmax function is that this is a regression problem. The reason I set the output layer size to be 1 is because we want the model to give us one value, the predicted BMI. The reason I set the hidden layer to be 5 and each layer to have 10 nodes is to add more depth to the network and reduce the width, trying to get a better performance. The reason I used the Adam optimizer is that this is a relatively faster optimizer. The reason I set learning rate to 0.1 is that it converges faster, and the reason I set the weight_decay is to add a regularization term to prevent overfitting. The reason I used the mse_loss function instead of nll_loss is that mse_loss function is designed to compute loss for a regression method. And it has the same tendency of RMSE as specified in the problem description because RMSE is just the square root of MSE. The reason zero_grad(), backward(), step() function are applied is the same as question 1. The reason training epoch is 1000 is that longer epochs will converge the loss better. The RMSE is computed because RMSE is a good metric to validate the regression model.
    c. My neural network has a RMSE of 6.03, better than the models in question 4, this is how low I can get.

```
my nueral network: 6.0332834341974975
```

The graph shows the comparison between my model and the 2 models in question 4. Clearly my model has a lower RMSE.



d. I think RMSE seems to depend on a lot of design choices. For this question, when I add more layers and reduce the number of nodes of the layers in my model, and use the Adam optimizer with a different learning rate and regularity term, the RMSE reduces. The RMSE dependence includes the network structure, number of layers(depth) and number of nodes(width), the activation function, the optimizer function used, the learning rate, the epochs training, the regularity term, and even the batch size when training. In fact, there seems not to have a general rule of how they influence the model performance, it is very specific to the data and the questions we should solve, and we should note that since there would always be trade-offs between underfitting and overfitting, our job is to tune those hyperparameters and find a design choice that balance in between.

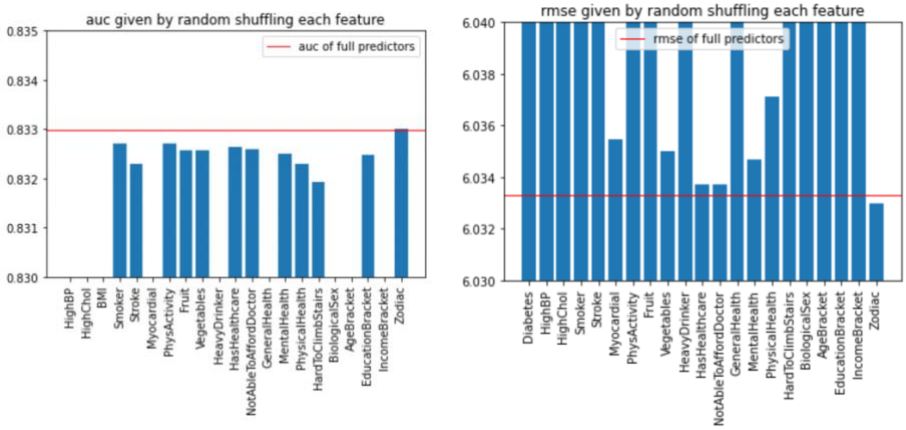
- Extra credit

1. Are there any predictors/features that have effectively no impact on the accuracy of these models? If so, please list them and comment briefly on your findings
   a. In this question, we would take one model from the classification problem 1-3 and one model from the regression problem 4-5 to present my answer.
   b. For both models, I ran a for-loop of all the predictors. In each round, I randomly shuffle the current predictor of the test data, and feed it into both models and compute the AUC and RMSE separately and store them in a dictionary. Then the AUC and RMSE are compared with the original test data to see shuffling which predictor does not change the original AUC or RMSE very much.
   c. The AUC dictionary of the classification problem and the RMSE dictionary of the regression model are shown below.

```
auc of full predictors: 0.8330
HighBP: 0.8154
HighChol: 0.8206
BMI: 0.7954
Smoker: 0.8327
Stroke: 0.8323
Myocardial: 0.8299
PhysActivity: 0.8327
Fruit: 0.8326
Vegetables: 0.8326
HeavyDrinker: 0.8296
HasHealthcare: 0.8326
NotAbleToAffordDoctor: 0.8326
GeneralHealth: 0.7746
MentalHealth: 0.8325
PhysicalHealth: 0.8323
HardToClimbStairs: 0.8319
BiologicalSex: 0.8294
AgeBracket: 0.8007
EducationBracket: 0.8325
IncomeBracket: 0.8297
Zodiac: 0.8330
```

```
rmse of full predictors: 6.033
Diabetes: 6.172
HighBP: 6.225
HighChol: 6.041
Smoker: 6.054
Stroke: 6.041
Myocardial: 6.035
PhysActivity: 6.071
Fruit: 6.042
Vegetables: 6.035
HeavyDrinker: 6.045
HasHealthcare: 6.034
NotAbleToAffordDoctor: 6.034
GeneralHealth: 6.180
MentalHealth: 6.035
PhysicalHealth: 6.037
HardToClimbStairs: 6.176
BiologicalSex: 6.082
AgeBracket: 6.433
EducationBracket: 6.048
IncomeBracket: 6.049
Zodiac: 6.033
```

A graph with a certain range is drawn and the line of the original test data (auc of full predictors) is labeled as red, as shown below.



d. According to the graph, in the classification problem, only the AUC of shuffling Zodiac has almost no effect on the original test data's corresponding AUC score. In the regession problem, the RMSE of shuffling HasHealthCare and NotAbleToAffordDoctor and Zodiac has almost no effect the original test data's corresponding RMSE score. Hence, I conclude that Zodiac has effectively no impact on the accuracy of the models from the classification problem (predicting diabetes or not). HasHealthCare, NotAbleToAffordDoctor, and Zodiac has effectively no impact on the accuracy of the models from the regression problem (predicting BMI).

2. Write a summary statement on the overall pros and cons of using neural networks to learn from the same dataset as in the prior homework, relative to using classical methods (logistic regression, SVM, trees, forests, boosting methods). Any overall lessons?
    a. Pros: neural networks give the user more freedom to design the model, and it can fit nonlinear data and more complex data well. This is important because in reality most data have nonlinear and very complex relationship. For example,

neural networks like CNN and RNN could attain the results from specific complicated dataset that classical methods cannot attain. Also, it would not presuppose the relationship between the data, and it could instead learn the implicit relationship between them. They are quite strong learners.

b. Cons: the freedom also makes neural networks requires more hyperparameter tuning like learning rate and design choices like the model structures. Generally, it might be difficult to find the best performance design choice of the model. In this way, underfitting and overfitting problem could be more serious. The development of the neural networks is also more complex and painstaking. Also, when a model is well developed, it is hard to improve its overall performance.