Part 1: Loss minimization with gradient descent

1. $\nabla L_{\omega 1,\omega_2}$ calculation by hand:

$$\mathrm{L}(\omega_1,\omega_2) = -10\left(0.4\cos(\omega_1) - \omega_1^2 - 0.2\omega_2^7 + \sin(\omega_2)\right)\mathrm{e}^{-\omega_1^2-\omega_2^2}$$

For the gradient calculation we used the product rule – Derivative of a product of two functions, where the two functions are: $\left(0.4\cos(\omega_1) - \omega_1^2 - 0.2\omega_2^7 + \sin(\omega_2)\right)$ and $\mathrm{e}^{-\omega_1^2-\omega_2^2}$.

$$\frac{\partial L}{\partial \omega_1} = -10\left[(-0.4\sin(\omega_1) - 2\omega_1)\cdot e^{-\omega_1^2-\omega_2^2} + \left(0.4\cos(\omega_1) - \omega_1^2 - 0.2\omega_2^7 + \sin(\omega_2)\right)(-2\omega_1)\mathrm{e}^{-\omega_1^2-\omega_2^2}\right]$$

$$\Rightarrow \frac{\partial L}{\partial \omega_1} = -10\mathrm{e}^{-\omega_1^2-\omega_2^2}\left[(-0.4\sin(\omega_1) - 2\omega_1) + \left(0.4\cos(\omega_1) - \omega_1^2 - 0.2\omega_2^7 + \sin(\omega_2)\right)(-2\omega_1)\right]$$

$$\frac{\partial L}{\partial \omega_2} = -10\left[(-1.4\omega_2^6 + \cos(\omega_2)\cdot e^{-\omega_1^2-\omega_2^2} + \left(0.4\cos(\omega_1) - \omega_1^2 - 0.2\omega_2^7 + \sin(\omega_2)\right)(-2\omega_2)\mathrm{e}^{-\omega_1^2-\omega_2^2}\right]$$

$$\Rightarrow \frac{\partial L}{\partial \omega_2} = -10\mathrm{e}^{-\omega_1^2-\omega_2^2}\left[(-1.4\omega_2^6 + \cos(\omega_2) + \left(0.4\cos(\omega_1) - \omega_1^2 - 0.2\omega_2^7 + \sin(\omega_2)\right)(-2\omega_2)\right]$$

$$\nabla L_{\omega_1,\omega_2} = \left(\frac{\partial L}{\partial \omega_1}, \frac{\partial L}{\partial \omega_2}\right)$$

2. grad_desc function results:

For [w01,w02] = [1,1] and η = 0.001:

Optimal Weights are: w1=0.022, w2=0.519

The final loss value with the optimal weights is: -6.820

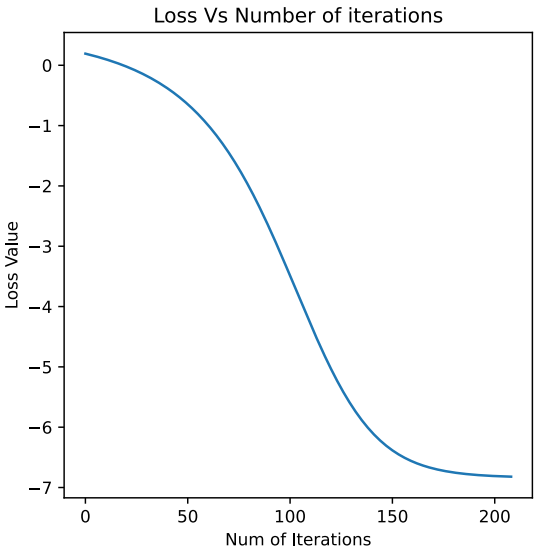


Figure 1 – Loss vs. iterations when [w01,w02] = [1,1] and η = 0.001

For [w01,w02] = [0,-2.2] and η = 0.001:

Optimal Weights are: w1=0.0, w2=-1.952

The final loss value with the optimal weights is: -4.664

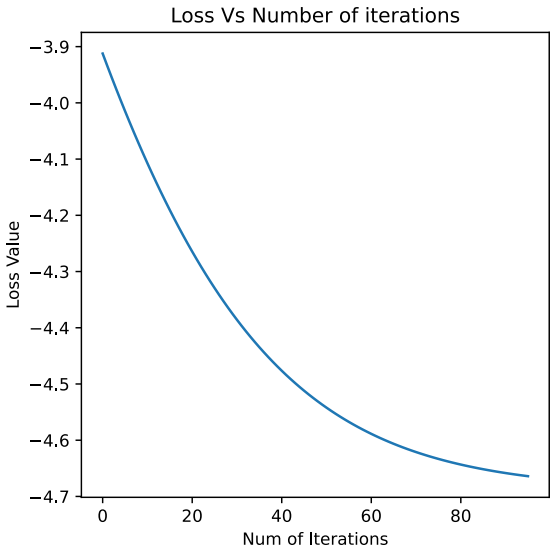


Figure 2 – Loss vs. iterations
when [w01,w02] = [0,-2.2] and η = 0.001

For [w01,w02] = [1,1] and η = 0.1:

Optimal Weights are: w1=0.491, w2=0.477

The final loss value with the optimal weights is: -4.216

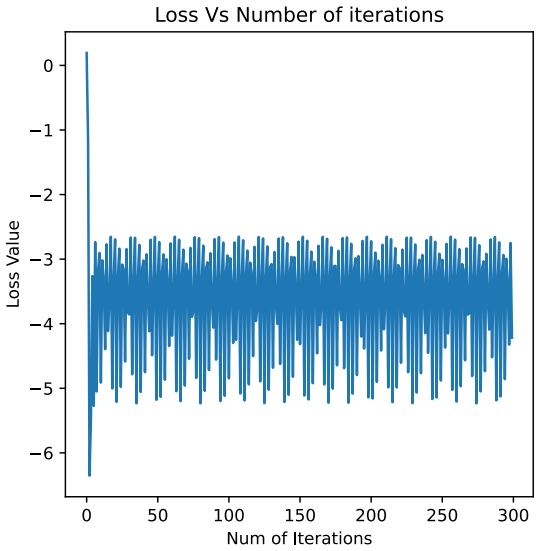


Figure 3 – Loss vs. iterations
when [w01,w02] = [1,1] and η = 0.1

For [w01,w02] = [2,2] and η = 0.001:

Optimal Weights are: w1=2.0003725813203115, w2=2.0000852494349304

The final loss value with the optimal weights is: 0.097

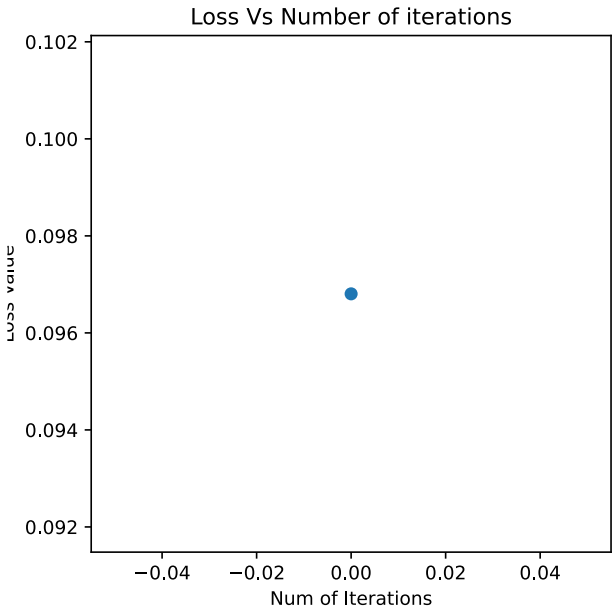


Figure 4 – Loss vs. iterations
when [w01,w02] = [2,2] and η = 0.001

3. As we can see in the 3D visualization of the loss function (Figure 5), the function is not convex – it has local minima and one global minimum, in addition to three different maxima.
From this reason, the achieved minima when using the grad_desc function is widely depends on the starting weights values w01 and w02 and the learning rate η.

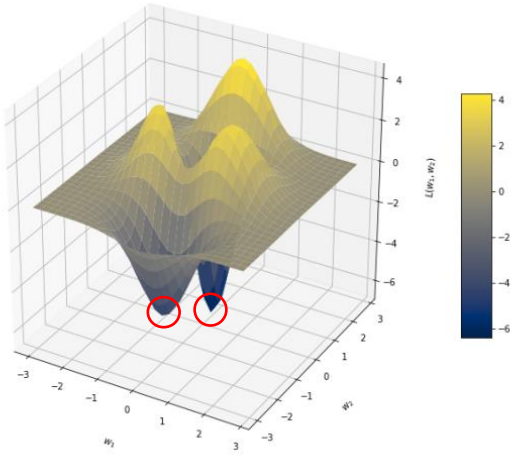


Figure 5 - 3D Visualization of the loss function with the different minima.

Figures 1,2 represents converges of the loss to minima with different starting weights w01,w02. The first conditions cause the gradient to point toward the global minima, while the second conditions cause the

gradient to point toward the local minima. That is why we ended up with two different final loss values. That is, the loss curve shape and the final loss value depend on w01,w02 values.

In figure 3, the learning rate value cause the algorithm to never achieve the stop condition of $\left\|\overrightarrow{W_{n+1}} - \overrightarrow{W_n}\right\|_2 < 10^{-4}$. The relatively high learning rate value appears to cause the loss to oscillate around the global minimum over the iterations, so the maximum iterations number is achieved before satisfying the stop condition, and the loss does not converge.

Figure 4 represents a very fast convergence of the loss – only a single iteration was needed for satisfying the stop condition. This happened because the starting weights are very close to the weights that leads to the minimal loss value, and the learning rate is small enough to allow convergence.

4. Adding L_2 regularization term to the loss:

$$\mathrm{L}(\omega_1,\omega_2) = -10\left(0.4\cos(\omega_1) - \omega_1^2 - \ 0.2\omega_2^7 + \sin(\omega_2)\right)\mathrm{e}^{-\omega_1^2-\omega_2^2} + \lambda\cdot(\omega_1^2+\omega_2^2)$$

Gradient calculation using the results from section (1) :

$$\frac{\partial L}{\partial \omega_1} = -10\mathrm{e}^{-\omega_1^2-\omega_2^2}\left[(-0.4\sin(\omega_1) - 2\omega_1) + \left(0.4\cos(\omega_1) - \omega_1^2 - \ 0.2\omega_2^7 + \sin(\omega_2)\right)(-2\omega_1)\right] + 2\lambda\omega_1$$

$$\frac{\partial L}{\partial \omega_2} = -10\mathrm{e}^{-\omega_1^2-\omega_2^2}\left[(-1.4\omega_2^6 + \cos(\omega_2) + \left(0.4\cos(\omega_1) - \omega_1^2 - \ 0.2\omega_2^7 + \sin(\omega_2)\right)(-2\omega_2)\right] + 2\lambda\omega_2$$

$$\nabla L_{\omega_1,\omega_2} = \left(\frac{\partial L}{\partial \omega_1}, \frac{\partial L}{\partial \omega_2}\right)$$

The reason we can use the term from section (1) is the linearity of the derivative operator (That is $(\alpha f + \beta g)' = \alpha f' + \beta g'$).

5. Results after modifying the grad_desc function with the keyword argument 'lmbda':

For [w01,w02] = [1,1] , η = 0.001 , $\lambda = 0.01$:

Optimal Weights are: w1=0.021, w2=0.518

The final loss value with the optimal weights is: -6.818

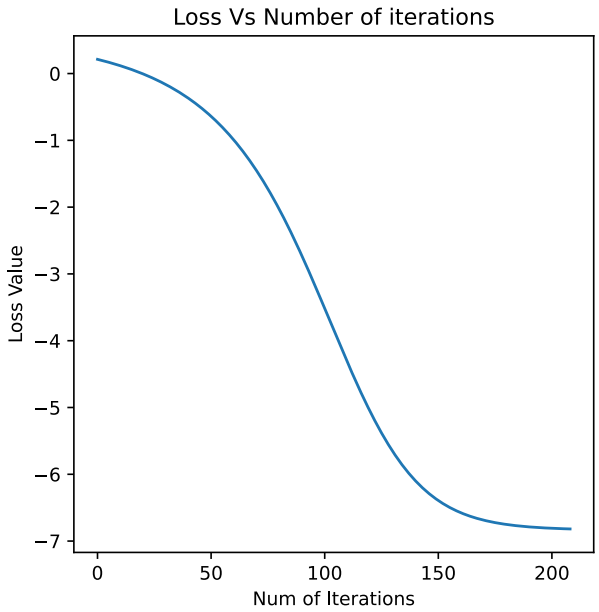


Figure 6 – Loss vs. iterations
when [w01,w02] = [1,1] , η = 0.001 , $\lambda = 0.01$

For [w01,w02] = [1,1] , η = 0.001 , $\lambda = 100$:

Optimal Weights are: w1=0.002, w2=0.051

The final loss value with the optimal weights is: -4.237

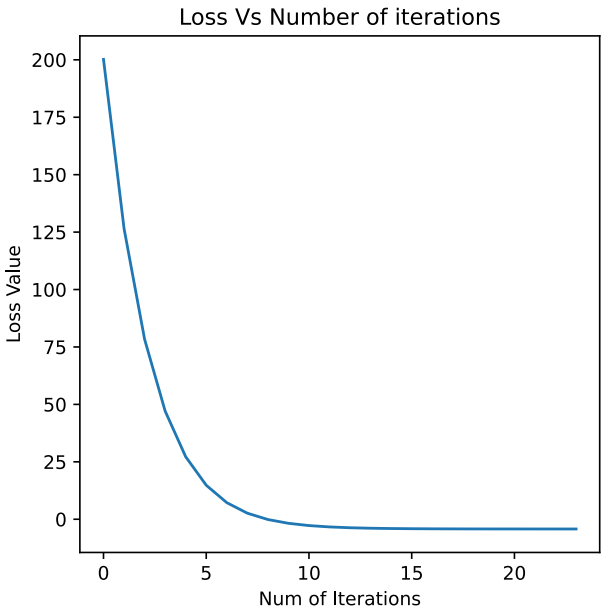


Figure 7 – Loss vs. iterations
when [w01,w02] = [1,1] , η = 0.001 , $\lambda = 100$

Figures 5,6 present the loss when using the same start conditions as in figure 1, but with the addition of regularization term.

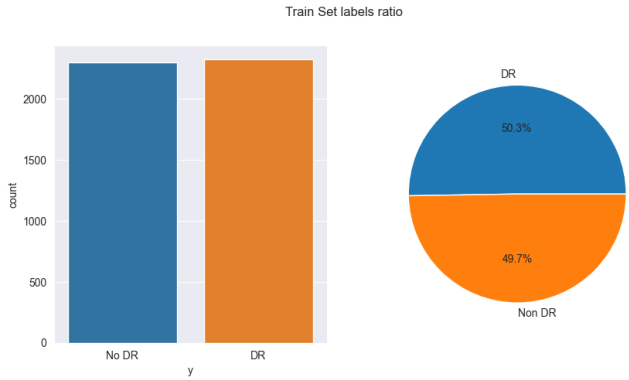
In general, higher λ makes the importance of the regularization term higher, and thus the model yields lower weights in order to minimize the regularized loss. As expected, the optimal weights when $\lambda = 100$ are much lower than the optimal weights when $\lambda = 0$ or $\lambda = 0.01$.

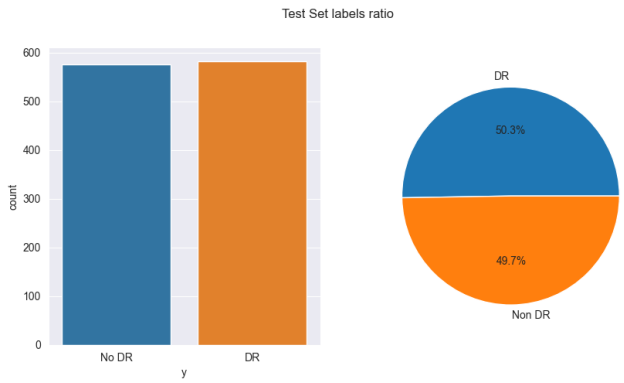
Part 2: Binary classifiers

2.2.1 Instructions

2.a.1

Features distributions:





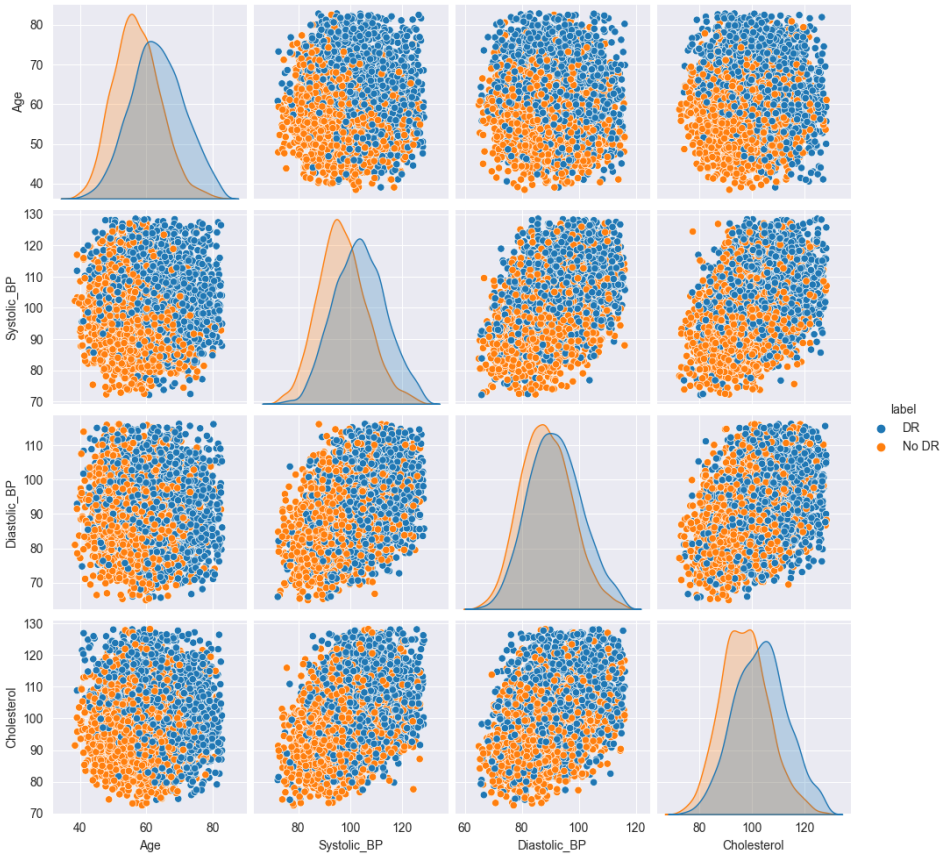
We can see there is almost a perfect balance between the classes (labels), both in Train and Test sets (As we used Stratified Train-Test split). This is very convenient, as in 'real life' this scenario is quite rare. In reality, our data will be mostly imbalanced, in a way that our model will learn more from the 'majority' class and less from the 'minority' class, which will cause bias in the model. To deal with these issues, there are few tactics; One of them is changing the performance metrics. Instead of looking at 'Accuracy' alone, which declares how accurate our model is, we can look at other statistics that will give us a better understanding on how 'well' our model is doing. These statistics include: Confusion matrix (A table that showing the amount of correct predictions along the amount of incorrect predictions and their types (False negative, False positive)), Precision (The ratio between True Positive and all positive predictions) and sensitivity (The ratio between False positive predictions and all false predictions) and so on. By tuning our hyperparameters based on these statistics, we can get results that are satisfying enough for our application, even if the data is imbalanced.

2.a.2

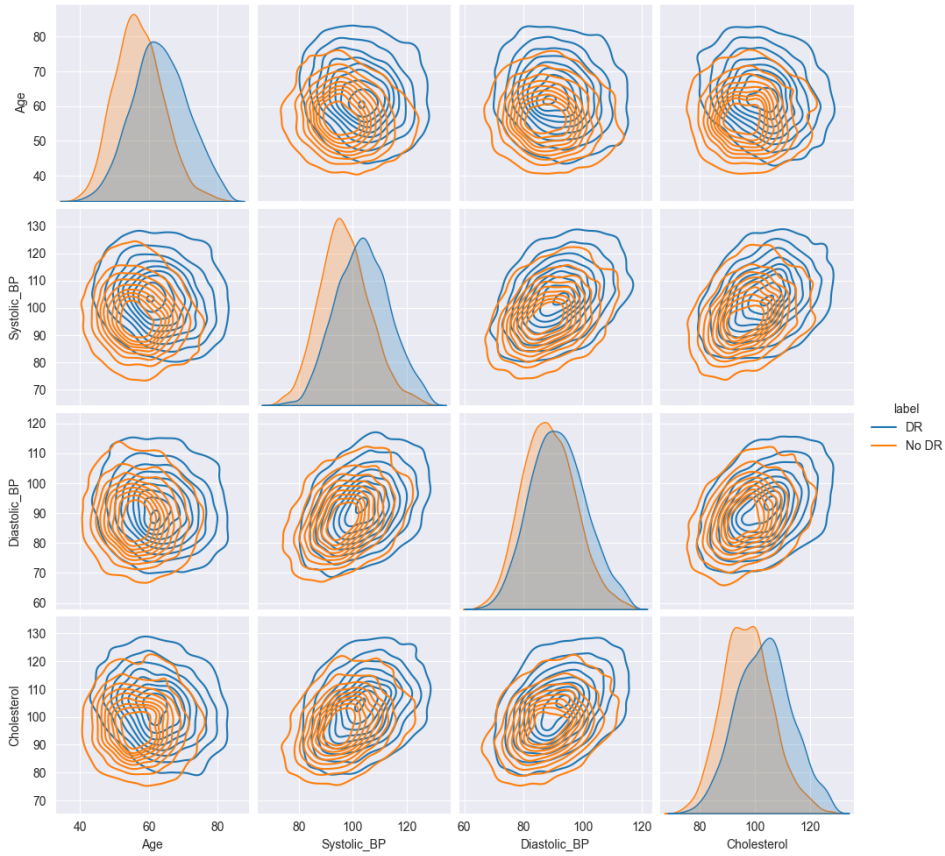
Labels imbalance between train and test is a different case. We would like both sets to have the same ‘conditions’ (As the reason we scale our data for example), so we could really test our model with weights relevant to the test dataset. Training and evaluating on uneven proportions of labels will lead to predictions errors, while the solution is simple. All we need to do is perform *Stratified* Test-Train Split, which means we force our split to have the same proportions in labels between the Train and Test sets.

2.c

Feature-Label relationship:



Here we can see a matrix of scatter plots, in which every graph is a scatter plot of 2 features and their labels. On the diagonal there's the distribution of a specific feature for each label. As we can see, the features are hardly separated. There is a large portion of the samples that mixed with one another, but there are portions that have mainly blue or orange dots (generally on the top right or bottom left values which are the edge values). To understand better the distribution of the features and their labels, we'll use kde plot (kernel distribution estimation).



Here, every graph shows us the estimated 2D distribution of the features for each label. As we can see, the distribution overlaps in the majority of the area, which strengthen our claim that the data is hardly separated. With that being said, we can also see that the 'Age' feature has the most separated distribution compared to other features, so we would assume it is particularly important for our model (It also makes physiological sense and is backed up by this reference: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8213493/).
Furthermore, we would say that BP (Specifically Systolic) is also important for our model for the same reasons.

3.c

Evaluation metrics for the three models are:

- Logistic Regression: AUC Score: 0.717, F1 Score: 0.720, Accuracy Score: 0.717
- SVM: AUC Score: 0.719, F1 Score: 0.724, Accuracy Score: 0.719
- RFC: AUC Score: 0.717, F1 Score: 0.727, Accuracy Score: 0.717

3.d

We can see that all the models performed about the same on the test set. RFC has the best F1 score, while SVM has the best AUC and Accuracy scores, but the differences are extreme subtle. So in terms of F1 score, the non-linear model performed better, while SVM (Notice that's a *linear* SVM, as the linear kernel was chosen by the GridSearch) has AUC and ACC scores of 0.002 higher than the non-linear model (RFC). LR isn't leading in any of the metrics. Overall, we'd say that RFC performed best, by a small margin compared to SVM.
A naive classifier would have gotten an accuracy score of 0.503, so obviously we can say that our estimators performed better, but not by a great margin.

4.a.1

Train Set Scores for RFC: AUC Score: 0.781, F1 score:0.786, Accuracy score: 0.781

Test Set Scores for RFC: AUC Score: 0.717, F1 score:0.727, Accuracy score: 0.717

4.a.2

We can see there's a difference between the scores of the train and test sets because naturally our model has 'seen' the train set and learned to make predictions on it. The test set exists in order to test how well our model can generalize to cases it's never seen before. In order to reduce the difference (In other words, make our model better), we need to fine the sweet spot between bias and variance. The more our model is over-fitted on the train set there's more variance, and the more our model is under-fitted it has more bias. There are few techniques that helps find this sweet spot such as regularization (helps prevent overfitting), validation of hyperparameters (Hyperparameters define and affect the learning process so tuning them widely affect the results), and so on.

4.a.3

There are some major advantages of RFC over SVM, and here are few:

- RFCs are faster to train and predict compared to SVMs, especially for large datasets. That's because RFC are based on decision trees, which are faster to train and predict compared to SVM which based on complex kernels.
- RFCs are much less sensitive to noise. That's because RFC use a lot of decision trees which are trained on randomly generated data from the training data ('Bagging' process), and therefore are less sensitive to noise and outliers compared to SVM.
- RFCs can also handle missing values while SVMs cannot. That's because RFC can impute missing values by using values from other features, which it already normally does in the training period. In SVM, all features require a value in order to make a prediction.

4.a.4

According to LR model, the most important feature is: 'Age'
and the second most important feature is: 'Systolic_BP'

According to RFC model, the most important feature is: 'Age'
and the second most important feature is: 'Systolic_BP'

According to both LR and RFC models, the most important feature is 'Age' and the second most important feature is 'Systolic_BP'.

This match up with the claims we made in the feature exploration part we preformed earlier, that 'Age' is a particularly important feature for our model because it has the most separated distribution compared to other features distributions (And also ‘Systolic_BP’, for the same reasons, but in a less distinct way).