HW2- Bias Detection

Case Study 1:

How I approached this and got it's understanding:

1. The first part was reading the model from the pickle file and seeing how the model scored on both the test set and the train set.
2. Then to understand bias I needed to understand which file is the one we used to test and the one which our supervisor worked on. Once I got that figured out the next part was understanding what the bias was and how it was caused in one of the files and not the other.
3. One inference I could draw from the result was, that it should be heavily biased in one way or the other because this difference in result must be caused by it.
4. Based on this assumption when I went through the data, I could see that there was bias towards the sex column in the dataset. The dataset used by the supervisor had people for a certain gender having their loan approved and other having their loan denied, which could be the reason for the model to show the false accuracy.
5. Moreover, when I checked the file based on which we did our testing, there was not such bias. We'll talk about this more in the Analysis section. But this is my initial analysis.

Custom Functions used:

Based on the 5th point above I created the custom functions as I wanted to draw more inferences.

def calculating_bias(clf, file_to_load, sex_column):

1. This function takes in the model that we loaded from the pickle file, file to load (either test supervisor or our test file in the case study) and sex_column(which will have value either 0 or 1).
2. This was a simple function in which we were predicting the confusion matrix and Test Scores based on the difference gender sent as an argument.

def proving_bias(clf, file_to_load, sex_column,approval):

1. This method is also like the above method. The only difference is, in this method, I added the approval argument, to prove the different combinations of confusion matrix for the Supervisor.
2. This is the method that helps me drive my point home.

Steps I used to find bias and How I used above functions for my advantage:

1. The first feature I stumbled upon when searching the features was the sex column and when I fired simple single line code in python to filter out the results where the sex was 0 and one query for what the result was when sex was equal to 1, I could see that there were no instance of loan being approved for sex 1 and no loan being disapproved for sex as 0.
   This inference would only be true if the same bias is not found in the test set which gave us score of 47%, otherwise it would be some other feature causing or set of features causing the model to behave weirdly. So, using the same query I printed out the approval for sex 1 and sex 0 to see if the bias is present in it or not. As expected, the test set which we used didn't has bias as both the sexes were almost equally divided among the loan approved or not.
2. The above point helped me make my case but to drive my point home I had to use the confusion matrix topshow what I thought is occurring. To do this, I leveraged the function described above to prove my point.

3. I first used the calculating_bias function where I printed out the confusion matrix for different instances and they are below:
Table: Model Evaluation Results

Instance	Test Score	TP	TN	FP	FN
Our Test Set(gender=0)	0.47	230	0	258	0
Our Test Set(gender=1)	0.47	0	245	0	267
Supervisor Test Set (gender=1)	1.0	0	484	0	0
Supervisor Test Set (gender=0)	1.0	516	0	0	0

(TP=True Positive, TN=True Negative, FP=False Positive, FN=False Negative)

With this we can see that the test set used by our supervisor is predicting the loan to be approved as either yes or no as based on the gender. In the dataset it's showing that when gender is 0 the dataset has approval as 1, and the model predicts that itself, which can be seen as True Positive which is nothing but no. of instances that were in the test set positive and were predicted positive by the model.
While when we consider the gender as 1 the dataset has all the instances as 0 for it, and the model predicts them as 0 as well, that is the reason we got true negative as 484, which shows that all the instances of loan ,the model predicted loan as disapproved. In contrast the test set used by us, had mixtures of the loan approval and disapproval for both the genders, that is why we can see the score dropping, along with the drop in True Positive and Increase in False Positive. What these numbers mean is, for the gender 0 where the loan was approved the model approved the loan, and where the loan should be disapproved, the model approved the loan there as well, showing the gender bias.
Similarly, when we predicted with the gender subset of 1, on our test set, we can see that the model showed decrease in the number of True Negative and Increase in the False Negative, which states that model always predicted the loan as disapproved (0) when the gender was set as 1.
This thus proves our bias. I took it one step further to prove my point that model is biased and the dataset used by the supervisor is biased as well. I do this by using the prove_bias function on our test set, by selecting different sex along with different combinations of the loan approval to create more confusion matrix. Below is the result for the same:

Sex	Loan Approved	Test Score	TP	TN	FP	FN
1	0	1.0	0	245	0	0
1	1	0.0	0	0	0	267
0	1	1.0	230	0	0	0
0	0	0.0	0	0	258	0

(TP=True Positive, TN=True Negative, FP=False Positive, FN=False Negative)

As we can see from the above table as well, when I passed the data to the proving_bias function, we can see the model only predicts the Loan as approved for Gender 0. We see the accuracy (Test Scores) as 1 only for the Sex being 0 and loan being approved. While for sex 1 with a disapproved loan the test score is 1 and all the data are true negatives, when we consider sex 0 with a disapproved loan, the model score drops to 0 and all instances become false positives. Similarly, when we take sex 1 with an approved loan, the accuracy drops to 0 and all instances are classified as false negatives.

Conclusion:

The above helps us prove the bias in the model and the bias in the dataset used by our supervisor to predict the loan of the model.

Case Study 2:

Now this one was a tough one to think about. Challenging but rewarding at the same time. This case study had combinations of insight to drive the point home.

First, I'll go on by explaining all the methods that I used so that you can reference them, when I explain how I went ahead with my approach:

Functions defined:

def read_file_initialy_and_get_result(file_name):

1. This function only had one job, that was to read the file, normalize it like our fellow student and get the result.
2. The function is being used to see if the initial stated results are true or not.
3. This function returns the test scores and confusion matrix for the file location passed in the function argument.

def read_file_initialy_and_get_additioal_result(file_name):

1. This is an advanced version of the previous function.
2. This function too reads the file location, prints the score, confusion matrix, along with the feature weights and intercept of the LR model.

def retraing_the_model(file_name):
1. This function is used to train the logistic Regression after finding the bias and mitigating it. This method uses the standard scalar for the normalization. Here I've used pipeline to implement this along with the Logistic Regression Model.

2. This functions return the pipeline so that the same pipeline can be used in the test function.
3. It also, prints the confusion matrix and Training accuracy on the train data as well.

def predicting_true_function(file_name,pipeline):

1. This function takes the file name and pipeline to test the model using the same standardization, which was used to train the model. This maintains the consistency in the train data and the test data.
2. This function is used to display the confusion matrix along with the test accuracy on the test data.
3. The pipeline passed as the argument is the same pipeline which is retured from the def retraing_the_model(file_name) function.

Steps I took to find the bias, how it was mitigated and the new improved model scores along with their analysis.

This one was very different from the previous one to approach. This one required deep analysis and this is how I went. I'll explain my though process and how I approached it and found the solution.

1. First, I loaded the model and printed the test scores for both the train and test file along with their confusion matrixes. And below is the result I got:

File	**Score**	**TP**	**TN**	**FP**	**FN**	**Bias Seen**
Train	**0.686**	**1098**	0	**502**	0	**Yes**
Test	**0.705**	**282**	0	**118**	0	**Yes**

   Now with this I can see that the model is always predicting the answers as 1 in both the cases, whether it be train data or the test data. So, this confirm that the bias is definitely there, which is causing our model to predict answer as 1 always.
2. Now the next task was to understand how the data can be biased. I went ahead and looked at the data and tried to find, if by a glance I could see a feature like I saw in the Case Study 1. But with my first glance I couldn't find it.
3. Then I thought that the imbalance in the data set could be the reason, so then I checked for the instances of will_pass as 1 or will_pass as 0, to see if the imbalance in the class could be causing the issue.
   a. So, I was able to find the imbalance in the class label, with the approx. ratio of 2:1. Where 2/3 were the instances of class label(will_pass) as 1 while 1/3 were instances of class label being 0.
   b. Now this can be the bias issue, but not something to cause the model to predict all the instances as 1. So this became the dead end for me, or something that will lead me to local minima in this case.
4. Then after getting stuck with 3rd approach, I tried to check the weights of the model, and to my surprise this is where I started to understand the issue. I'll walk you through how I thought of the bias.

a. I called upon the function read_file_initialy_and_get_additioal_result(file_name) for this. This function prints the weights of the model.

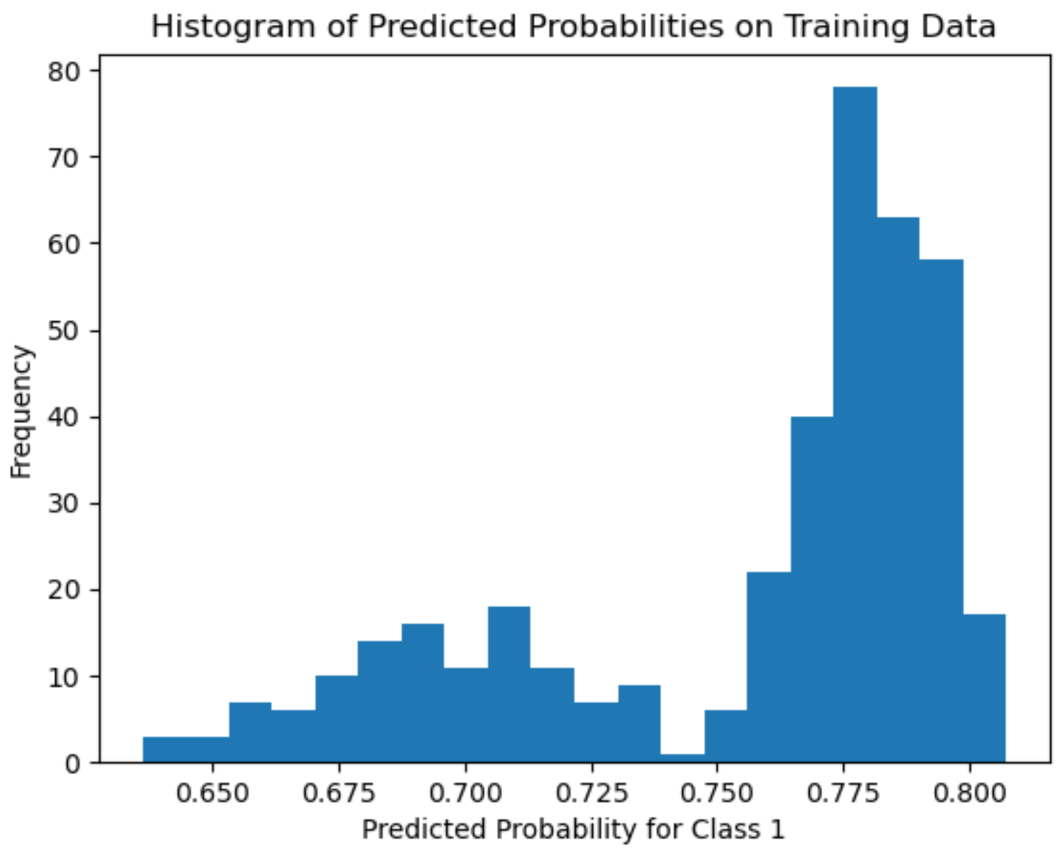
b. The weights were shown as below:

Feature	**Value**
average_HW_score	5.1144
exam1_score	5.4467
attendance_percentage	4.4992

All the weights for the model are +ve and the intercept is 0.4336. Which means that the model could be predicting high probabilities which could cause the model to predict all the value as 1.

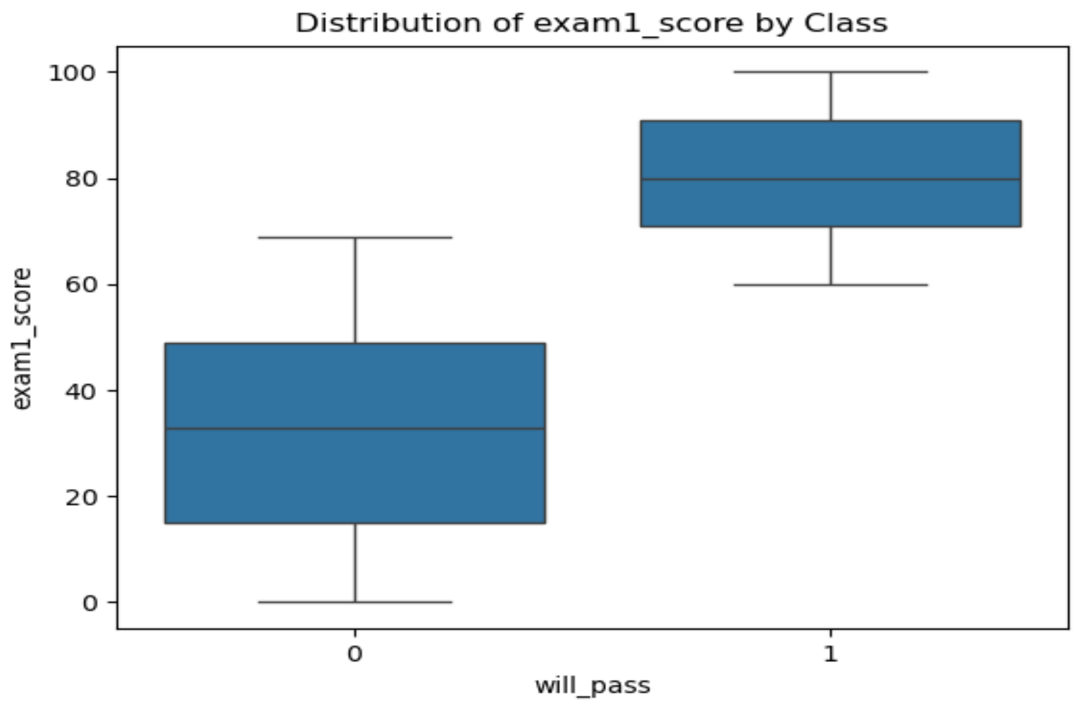
5. Now to see if my assumption from point 4 was correct or not. I decided to print the probabilities and plot a histogram as well for this. I didn't create a function for this as, it was a one-time thing, so writing the code directly was a good call. Below is the values for the probability and the histogram as well. This was done for the train data on the model from the pickle file.

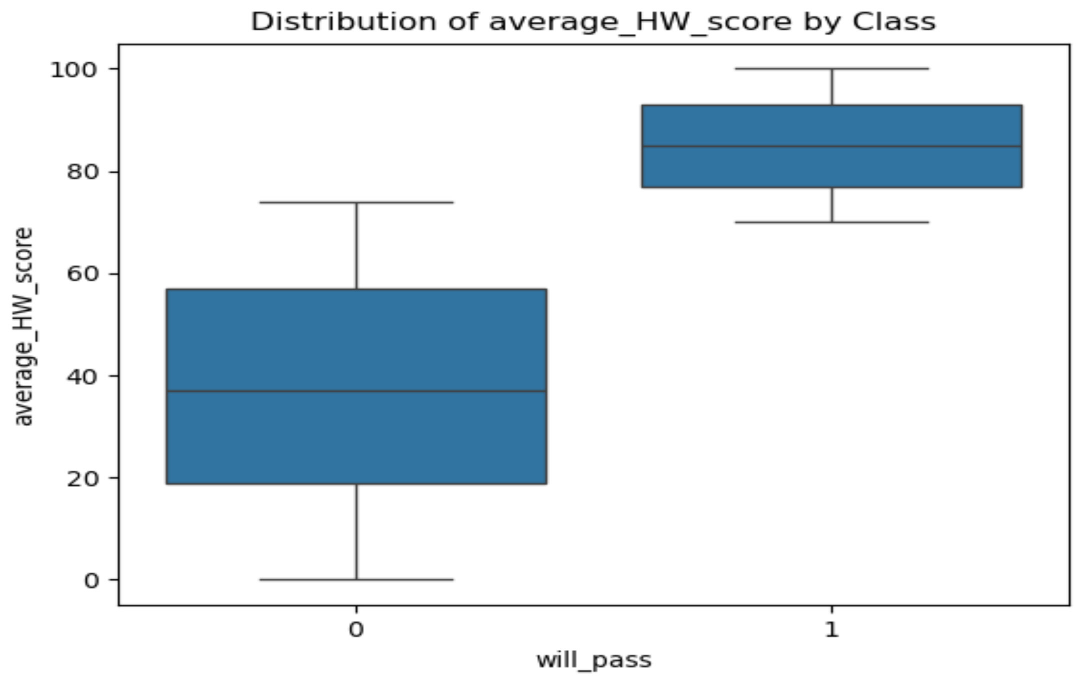
Min Probability	**Max Probability**	**Mean Probability**
0.63	0.80	0.75

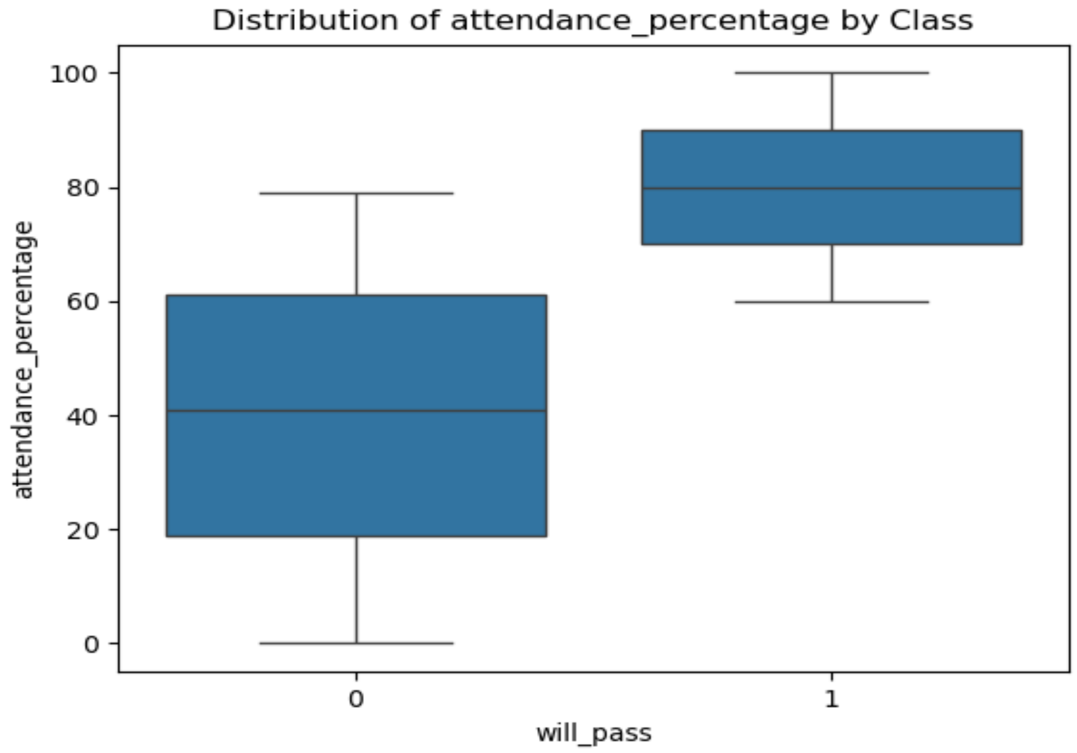


Now with this my assumption proved to be correct, that the model is predicting the probability more than 0.5(default threshold), which is why all the values are coming as 1 for this.
I can deep dive into this, please stay with me on this. In logistic regression model, the model uses a simple method for logits which is LR=B0+B1x1+B2x2+B3x3... and so on, where B0 is your intercept, and B1 is your weight of the feature and x1 is the input value. So with the weights all being positive, along with all the values being positive, the model is going to predict high logits, which when we pass though the sigmoid function is going to give us high probabilities.

6. So based on 5, I was coming close to my understanding as to why the bias could be, so now, I understood that there is some distribution issue in the data. Now to visualize this, if I could gather some insights, I plot the box plot for all the three features along with the values for their class and this is what I got.







With this I could see that though the mean of the value was different for both of the values(0 and 1) of will_pass, there were a lot of outliers in each of the following features. This got me thinking that the way the data is distributed, and the normalization technique we've used is probably the reason for the dataset to be biased.

7. Based on observation from 6th, I then moved ahead and looked again at the data, and voila, my assumption was correct. The data for the instance where it was predicting 1 all generally had high values and where it was predicting 0, at least 1 field had somewhat high value. Now this alone was fine, but when we include the part the weights of the features of the model along with the normalization technique we're using, it is going to predict high value.
    a. Our normalization technique, how it's messing up with our data. So, when we normalize with this along the column, the all the sum of square of all the values comes up to 1. So, which means the values which has greater value will have value close to 1. Now this coupled with high weights of the model, is going to produce high logits which in turn is going to produce high probabilities.
    b. To back this claim here is my table below of the normalized data of the train file where will_pass is equal to 1.

Index	**Average_HW_score**	**Exam1_score**	**Attendance_percentage**	**Will_pass**
4	0.023448	0.024776	0.027498	1
6	0.026462	0.032917	0.031327	1
9	0.028807	0.026546	0.023669	1
10	0.032157	0.024776	0.026454	1
11	0.027467	0.034686	0.032023	1
13	0.030147	0.024068	0.030283	1
15	0.024118	0.028315	0.032719	1
16	0.024453	0.028669	0.032371	1
18	0.025123	0.026546	0.034111	1

Here is my table below of the normalized data of the train file where will_pass is equal to 0

Index	Average_HW_score	Exam1_score	Attendance_percentage	Will_pass
0	**0.023113**	0.005309	**0.021929**	0
1	**0.021773**	0.011326	**0.026454**	0
2	0.008374	0.010972	0.004177	0
3	0.004690	0.006017	**0.020188**	0
5	0.000670	0.004955	**0.022973**	0
7	**0.014739**	0.006017	0.002437	0
8	**0.024118**	**0.014158**	0.005917	0
12	0.007369	**0.013096**	0.007658	0
14	0.002010	0.001062	**0.025758**	0
17	0.002680	0.010264	**0.020885**	0

Now with this you can see that almost 1 of the columns has the value which is similar to the high values displayed in the normalized data for the will_pass=1. I'll bold those high values in the above table. Now when we combine these high values with the high weights, we've for the model, it creates high logits, which creates high probabilities, which crosses the default threshold of the 0.5 thus the model predicts the value of 1 for every instance.

8. Now I understood the bias, the next work was to check the threshold, I fist though of increasing the threshold, but from the histogram above, I can see that the probabilities are distributed. It would be like firing an arrow in the dark, and the

result I would get from this, would again lead to the local minima, so not an optimal solution. The straightforward answer was using standard scalar, with standard scalar, it'll take the mean of the whole column and include the unit variance for the data. The data would be normalized optimally, as the higher values will have value more on the positive side and closer to the mean while smaller value will have values as negative or far away from the mean, as the mean would be influenced by the bigger values as their quantities is more. With this the weights will also be optimized better and even if we take the same weight, negative values of the input feature when multiplied my weight will produce negative value for the logits, and thus overcoming our issue for calculating large logits, which in turn creates lower probabilities for the negative case and higher for the positive cases of will pass.

9. Now based on my thinking in the step 8, I then used the function retraining_the_model to train the model with standardization and unit variance and got amazing result as follows on the training data:

File	**Score**	**TP**	**TN**	**FP**	**FN**	**Bias Seen**
Train	**0.998**	**1098**	**499**	**3**	**0**	No
Test	**0.995**	**282**	**166**	**2**	**0**	No

With this I can say that I had mitigated the bias with the standard scalar approach. T o show that what I'm saying is correct, here is the standardized data for the pass as 0:

Index	**Average_HW_score**	**Exam1_score**	**Attendance_percentage**	**Will_pass**
0	-0.035689	-1.900754	-0.178764	0
1	-0.189085	-1.260325	0.346493	0
2	-1.723053	-1.297997	-2.239386	0
3	-2.144894	-1.825409	-0.380786	0
5	-2.605084	-1.938426	-0.057551	0
7	-0.994418	-1.825409	-2.441408	0

8	0.079359	-0.958947	-2.037364	0
12	-1.838100	-1.071964	-1.835342	0

Here you can see that standardization has most of values as negative for the situation where the pass is set as 0 while the standardized data looks like below where the pass is set as 1 for the training data

Index	**Average_HW_score**	**Exam1_score**	**Attendance_percentage**	**Will_pass**
4	0.002660	0.171220	0.467706	1
6	0.347803	1.037682	0.912154	1
9	0.616247	0.359582	0.023258	1
10	0.999739	0.171220	0.346493	1
11	0.462851	1.226043	0.992962	1
13	0.769644	0.095876	0.790940	1
15	0.079359	0.547943	1.073771	1
16	0.117708	0.585615	1.033367	1
18	0.194406	0.359582	1.235388	1
20	0.117708	0.434926	0.184875	1

Here we can see that the values are all in the positive side for where the pass is set as 1.

Difficulties faced:

1. There were no difficulties faced, the case study 1 was straightforward, whereas the second one though was challenging was rewarding as well. There were situations, where the steps I took lead to a dead end but then that lead to thinking of a new way and connecting different aspects together.