Exercise 1

We have a binary classification problem, where our data is ***x*** and labels are ***Y*** (y = 0 or y = 1). The logistic regression model is given by:

1) **P(y_l = 1 | x_l, w) = 1 / (1 + e^(-w*x_l))** (***w*:** weights, l = lth value/element)
2) **P(y_l = 0 | x_l, w) = e^(-w*x_l) / (1 + e^(-w*x_l))** (***w*:** weights, l = lth value/element)

The log of the conditional likelihood:

w = argmax_w Σ [lnP(y_l | x_l, w)]

Which can be written as:

l(w) = Σ [y_l * lnP(y_l = 1 | x_l, w) + (1-y_l)*lnP(y_l = 0 | x_l, w)]

And given the 2 regression models for the two classes:

l(w) = Σ [y_l * ln(1 / (1 + e^(-w*x_l))) + (1-y_l)*ln(e^(-w*x_l) / (1 + e^(-w*x_l)))]

Because our data is linearly separable it means there exists a hyperplane that separates the data perfectly (based on classes). Since **w** is weights, it indicates the orientation of said hyperplane. Also, because the data is centered around 0 we assume separation happens at 0. Then we can say that:

1) For every **y_l = 1**, each corresponding **x_l** is on one side of the hyperplane, and there exists a weight vector **w** that: **w*x_l** > 0 and so **x_l > 0**
2) For every **y_l = 0**, each corresponding **x_l** is on the other side of the hyperplane, and there exists a weight vector **w** that: **w*x_l** < 0 and so **x_l < 0**

Since we are utilizing the fact that Y can only take values {0,1}, only one of the two terms will be non-zero given y_l. So due to this, the above can become:

1) For y = 1 and x_l > 0:
   l(w) = Σ [ln(1 / (1 + e^(-w*x_l)))] or l(w) = Σ [lnP(y_l = 1 | x_l, w)]
   As w increases: -w*x_l becomes very negative and so e^(-w*x_l) tends to 0. This means P(y_1 = 1 | x_l,w) tends to 1 log-likelihood to 0.

2) For y =0 and x_l < 0:
   l(w) = Σ [ln(e^(-w*x_l) / (1 + e^(-w*x_l)))] or l(w) = Σ [lnP(y_l = 0 | x_l, w)]
   As w increases: -w*x_l becomes very positive (since x_l is negative) and so e^(-w*x_l) tends to a very large number on both the numerator and the denominator.. This means P(y_1 = 0 | x_l,w) tends to 1 and the log-likelihood to 0.

Since our goal is to maximize the log-likelihoods above, we can do it by increasing the weights (**w**) to very large values (infinite weights), ensuring a definite probability for each class's element x_l since the probability tends to 1 and the log to 0 (maximum value). Basically the weights will "move" each point towards opposite ends, making the classification clearer.

Exercise 2

The function created called ***run_analysis(Xcat, Ycat, Xcont, Ycont, Xmix = None, Ymix = None, L=1, num_runs=100, size=0.25)*** uses as input 2 different dataset types, one continuous and one categorical. The only requirement is that both X matrix (Samples x Features) and Y array (Feature Classes) are provided. The parameter L is used for the Laplace hyperparameter (for NBC – Laplace smoothing) and for the inverse of regularization (for Logistic Regression – C = 1/L). ***num_runs*** and ***size*** indicate the number of iterations/repetitions and the size of the test set respectively. Other functions used here are ***trivial_train, trivial_predict*** and the previously implemented ***train_NBC, predict_NBC***. The former simply finds the unique classes and counts them, keeping the one with the most counts and then assigns every sample to that class, respectively.

As a first step we create a dictionary that will include as first keys the data type **'categorical'** and **'continuous'**. Then in turn, for each data type, we will add a new empty dictionary for each classifier. Since One-Hot Encoding (OHE) is necessary as a pre-processing step, we do this simultaneously by adding an **'_OHE'** at the end of names. Instead of rerunning the analysis without pre-processing, we include in the same function to compare. Keys for each data type:

1) Trivial Classifier – '**triv'** (continuous + categorical)
2) Naive Bayes Classifier – '**NBC'** (continuous + categorical)
3) Logistic Regression – '**LR'** (continuous + categorical)
4) Trivial Classifier – '**triv_OHE'** (categorical)
5) Naive Bayes Classifier – '**NBC_OHE'** (categorical)
6) Logistic Regression – '**LR_OHE'** (categorical)

As is normal for classification models, we will split our dataset into a 75% and 25% training and test set respectively. Another purpose of this exercise is to check how different starting training set sizes **(K)** affect the prediction model (sample classification) so we will keep 50% ,60% ,70% ,80% ,90% and all of the starting training set (we keep the percent after initially splitting into the 75%-25% sets). These are also embedded into each of the classifier keys as empty lists, so we will end up for each classifier with 6 keys containing the % of initial training size kept. An example is:

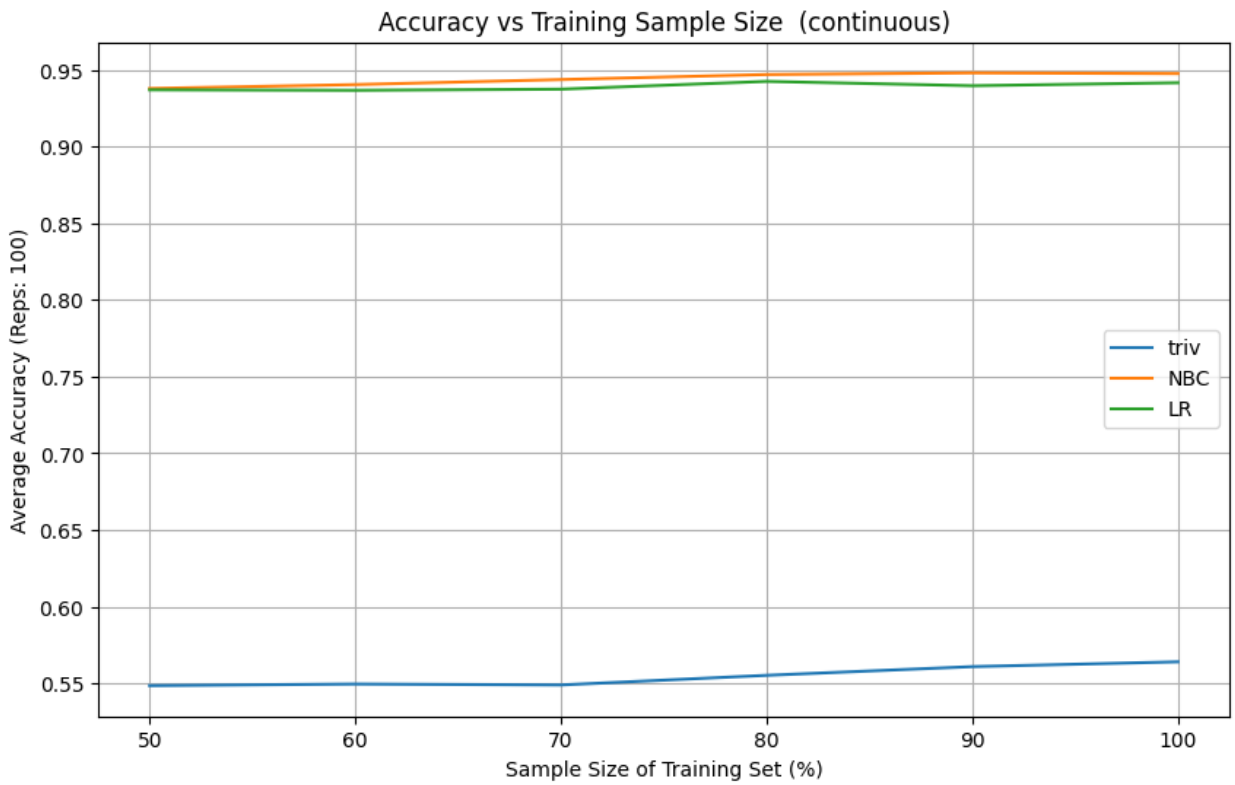
accuracies['categorical']['LR_OHE']['80'], containing the mean accuracies of Logistic Regression with OHE pre-processing, for the categorical data, when keeping the 80% of the training set.

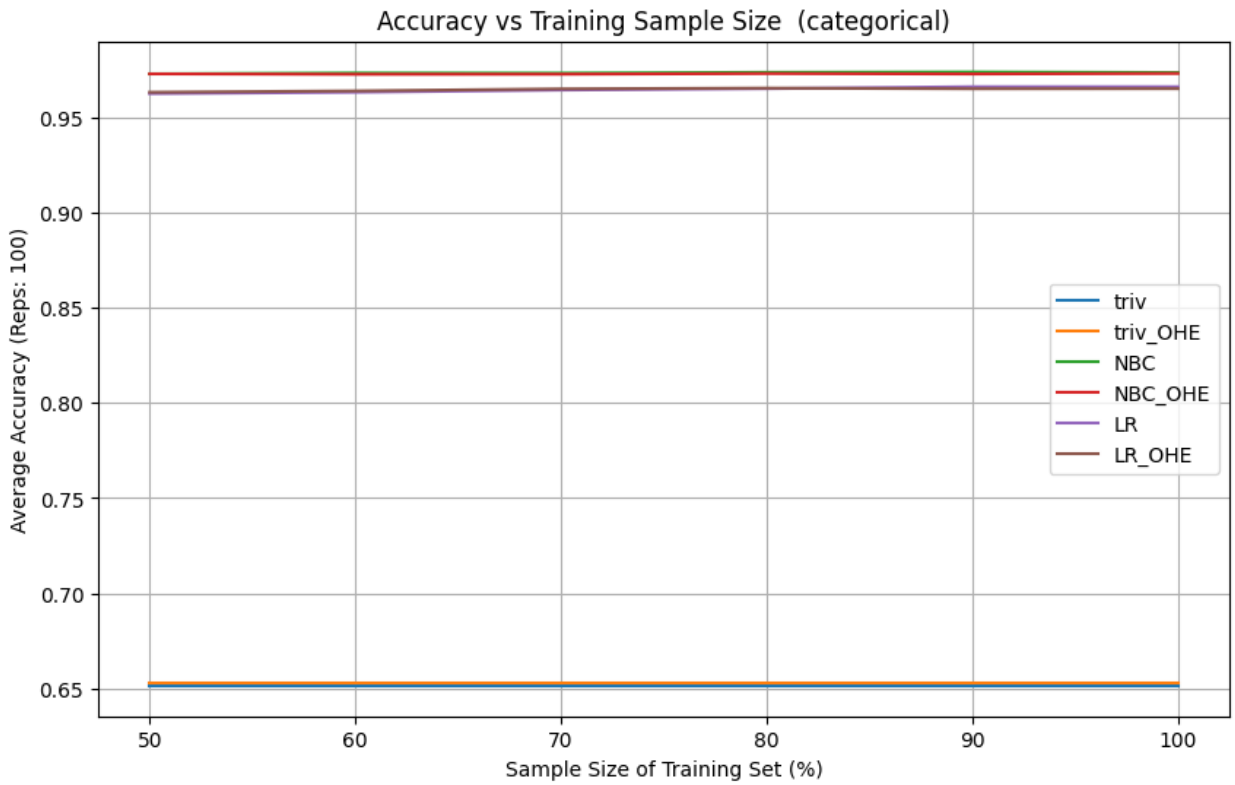
Because we are not provided with a D_categorical array, which is essentially a list containing the number of unique values for each feature, we create it for usage during NBC without OHE. Since we also have OHE transformed dataset X, we also create a D_categorical_OHE for the same purpose. For binary encoded data, this array will always be an array of twos (2) and a length equal to the columns (features). The reason is that we have only 0 and 1 as possible values, hence the array of twos.

First, we start an iteration for as many times specified in ***num_runs*** (repetitions), each iteration, we split our initial data (we do this for OHE encoded, unprocessed categorical and unprocessed continuous) and then we iterate through the different starting training set sizes **(K)**. For each training set size, we simultaneously use all 3 classifications (Trivial, NBC, LR) for all 3 data types (categorical, categorical_OHE, continuous). After each classification we calculate accuracy as a proportion (correct predictions / total predictions) and store it in its corresponding list. This is done for each **K**, after which we start a new repetition of splitting and going through different sample sizes.

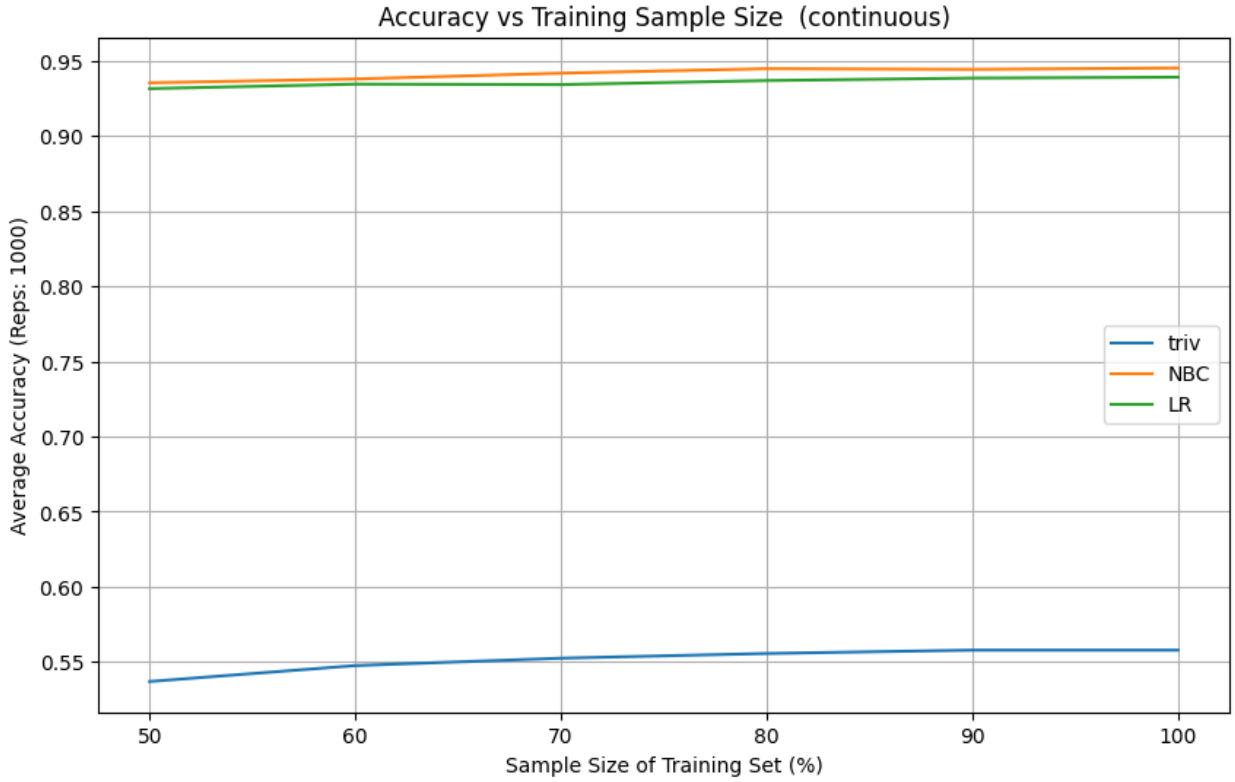
When the iteration has stopped and we have stored all of our accuracies, we calculate the mean for each data type, for each classifier, for each **K**, which is also returned as an output (dictionary) of the function.

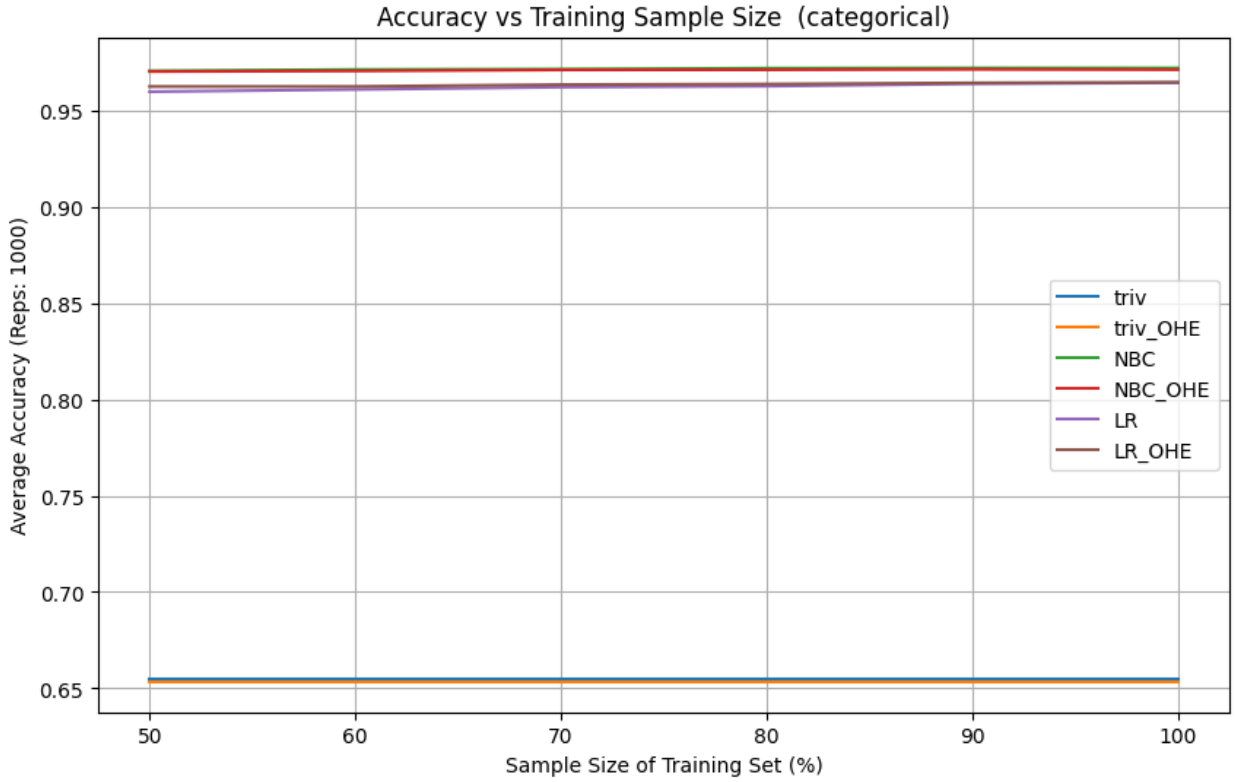
In order to visualize the differences between classifiers, as well as how accuracy is affected by starting training sample size, we plot the mean accuracies for 100 repetitions as seen below. As expected, the trivial classifier underperforms by a lot in both data types, since it just classifies all samples based on the most common class. That being said, we do notice a slight increase as starting training size increases for the continuous data, which might be expected given the classes might all have similar frequencies as well as different inputs as feature values (continuous). Concerning LR and NBC, we notice the mean accuracy for both is about the same with NBC slightly outperforming LR for both data types. For categorical we notice straight lines, indicating starting training sample size does not affect the predictions as much as it does when compared to continuous data, where we notice even some fluctuations, probably due to the number of iterations. Another worthy mention is that data with and without One Hot Encoding perform the same. But in general, the pattern is that as the training set increases the accuracy increases as well, which is to be expected since we train our model with more samples, meaning more possible combinations.


Accuracy vs Training Sample Size (continuous)
Average Accuracy (Reps: 100)
Sample Size of Training Set (%)
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
50
60
70
80
90
100
triv
NBC
LR


Accuracy vs Training Sample Size (categorical)
Average Accuracy (Reps: 100)
Sample Size of Training Set (%)
0.95
0.90
0.85
0.80
0.75
0.70
0.65
50
60
70
80
90
100
triv
triv_OHE
NBC
NBC_OHE
LR
LR_OHE

To investigate the fluctuations we notice for Logistic Regression, the function was repeated, this time for 1.000 repetitions.





As expected, it is clear now that increasing training set size increases accuracy, and the fluctuations noticed before were due to the number of iterations. NBC seems to slightly outperform LR for smaller training sizes. While the difference with OHE is not clear for 100 repetitions, the accuracy is better (even if only a little) for LR using OHE data. For trivial and NBC it does not seem to affect the accuracy, while we do notice again a very slight increased accuracy for non-preprocessed data.

Overall, it is clear that Trivial Classifier underperforms NBC and LR and NBC seems to be consistent with different training set sizes. The increase in accuracy in continuous data for Trivial classification is probably because larger training sets offer a better understanding of the distribution.
Concerning OHE, since our categorical values have few values and the Y is already binary, One-Hot Encoding does not seem to improve performance by a lot. Also, since both NBC and LR are generally robust models, including the fact we don't have an overflow of features and values, they can potentially capture the nature of the categorical variables without the need for OHE.
Logistic Regression works by finding a decision boundary that separates the classes based on the training data. When we use more data for training, the model gets more information, which can help improve its predictions. But, if the extra data is noisy or less relevant, it might make the model less accurate due to overfitting. This is probably why we see fluctuations in accuracy as the training data size changes.

Exercise 3

For a binary classification problem with classes $y = 0$ and $y = 1$, we model the decision boundary with a linear combination of the features (***x***) and weights (***w*** *– vector of adjustable parameters)*, plus the intercept (***w_0*** *– shifts decision boundary away from origin*) like so:
$$w^*x + w\_0 = \Sigma[\, w\_i^*x\_i \,] + w\_0 \quad \textit{(for } i = 1 \textit{ and d features)}$$

But we don't want to carry the intercept, so we can set a dummy feature ***x_0*** taking values 1 that would represent the intercept and include it in the summation. Then the above summation becomes:
$$\Sigma[\, w\_i^*x\_i \,] \quad \textit{(for } i = 0 \textit{ and d features)}$$

The estimated probability P(y=1|x;w) or p_1(x;w), of class 1:
$$p\_1(x;w) = 1 / (1 + e^{(-w^*x)})$$

The estimated probability P(y=0|x;w) or p_0(x;w), of class 0 is the complement of p_1(x;w):
$$p\_0(x;w) = 1 - p\_1(x;w)$$

The log-odds, which is the logarithm of the odds ratio of p_1/p_0 is:
log(p_1(x;w) / p_0(x;w)) = w*x (log odds of class 1 is a linear function of x).
Intercept is included as ***x_0***

The decision boundary in logistic regression is defined where the probability of both classes is equal. This occurs when p_1(x;w) = 0.5
Substituting this into the logistic function then the decision surface is:
w*x = 0
Without the dummy variable this becomes:
w*x + w_0 = 0

Based on the above we can say that if p_1(x;w) >= 0.5, we predict class y = 1. Otherwise if p_1(x;w)<0.5, we predict class y = 0. The surface where p_1(x;w) = 0.5 is the decision surface of the logistic regression model.

We create all our models separately for no penalization and "Lasso" penalization (λ = [0.5, 10, 100]). After we fit the model, we can find the weights using ***model.coef_[0].***
Each weight array here has a length equal to the features (in our case we have 1000x30 initial matrix, so 30 features)
The absolute value of the weight shows how much the feature contributes to the model's predictions (higher absolute values, higher impact).

a) A positive weight indicates that as the feature value increases, the log-odds of the predicted outcome increases (outcome becomes more likely).

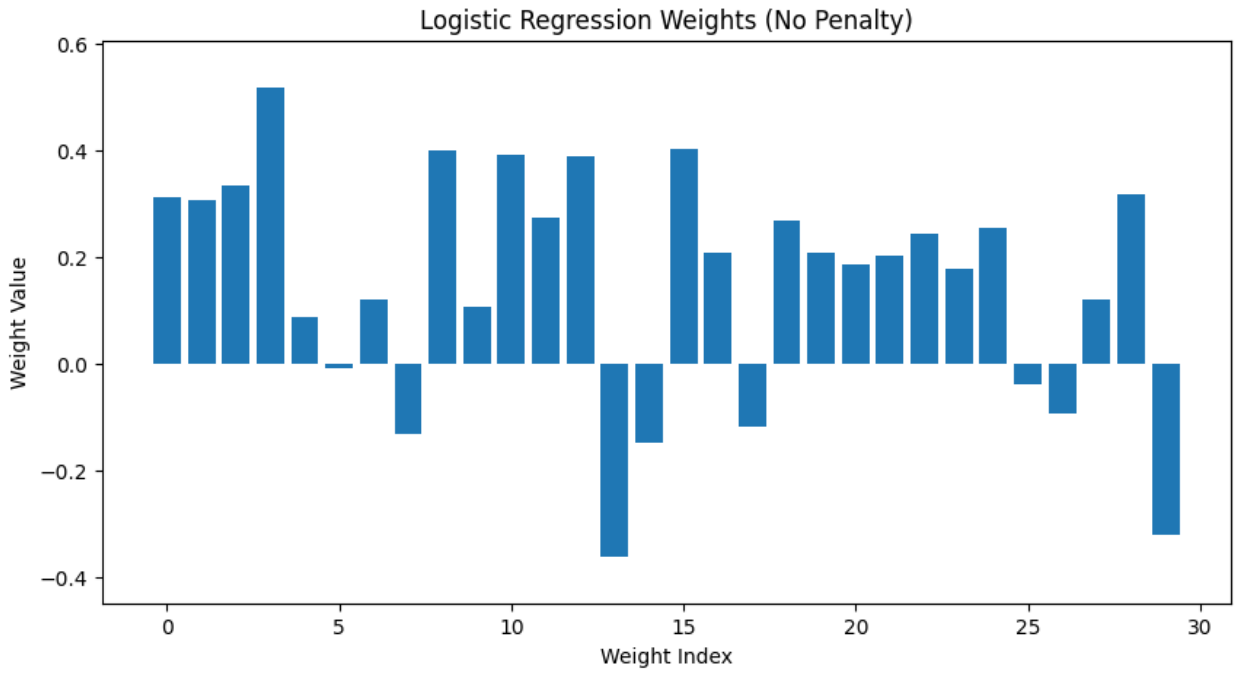
b) A negative weight means that as the feature value increases, the log-odds of the predicted outcome decreases (outcome becomes less likely).

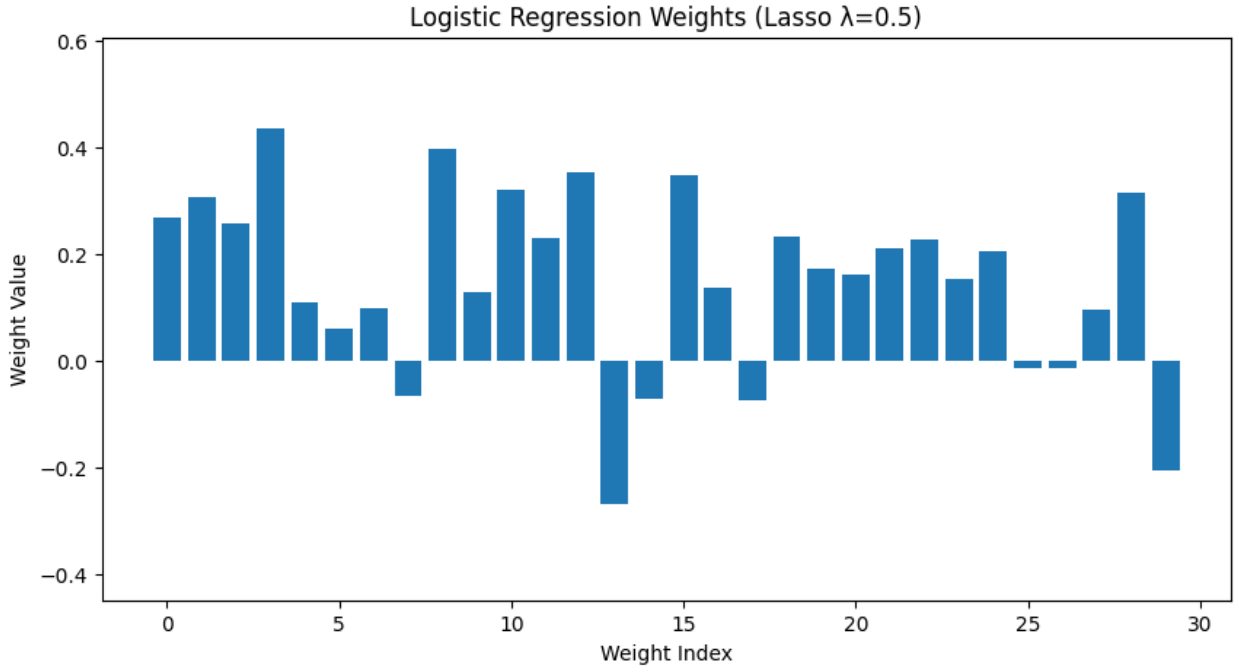
c) A weight equal to zero (or close to zero) suggests that the feature has little to no influence on the prediction. This occurs for non-informative features, and due to regularization, as we will see later from the plots.

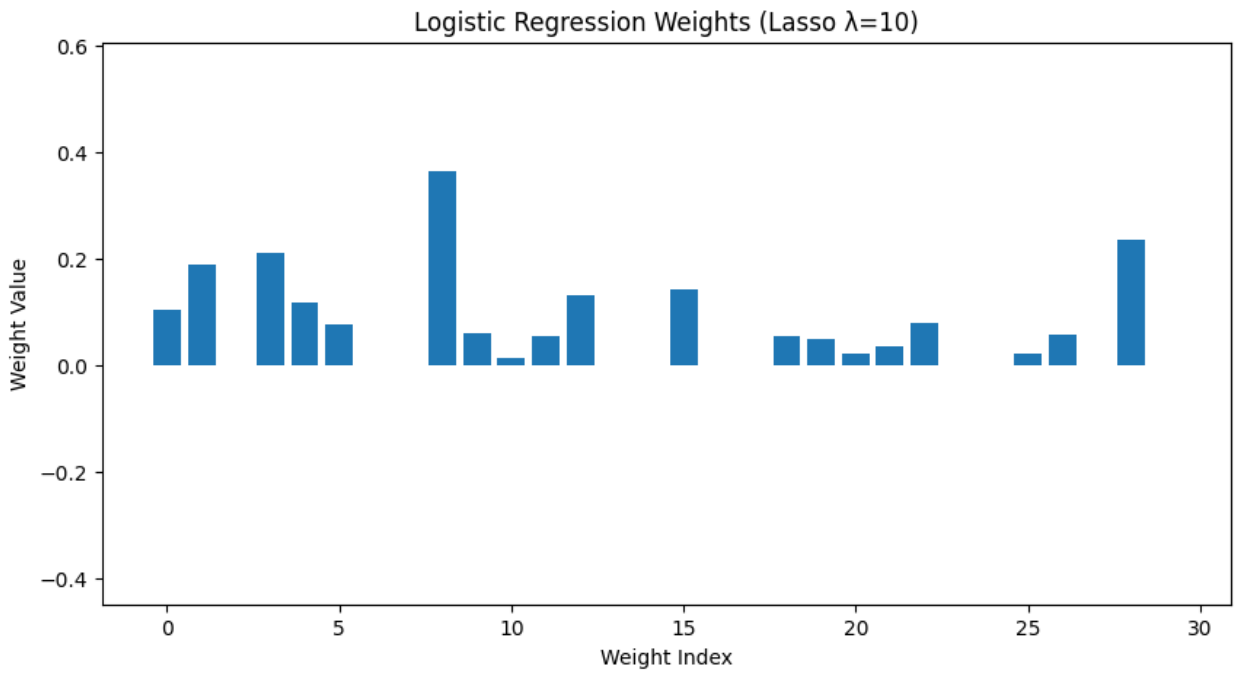
The purpose of regularization is to reduce overfitting. This is done by creating a modified penalized log likelihood function, punishing large weights. In our case we use **Lasso** or **L1 norm penalty**. The modified function to include the penalty term:
w* = argmax Σ_l [lnP(y^l | x^l, w) - λ||w||_1, where λ is the regularization strength and ||w||_1 is the L1 norm of the weight vector.

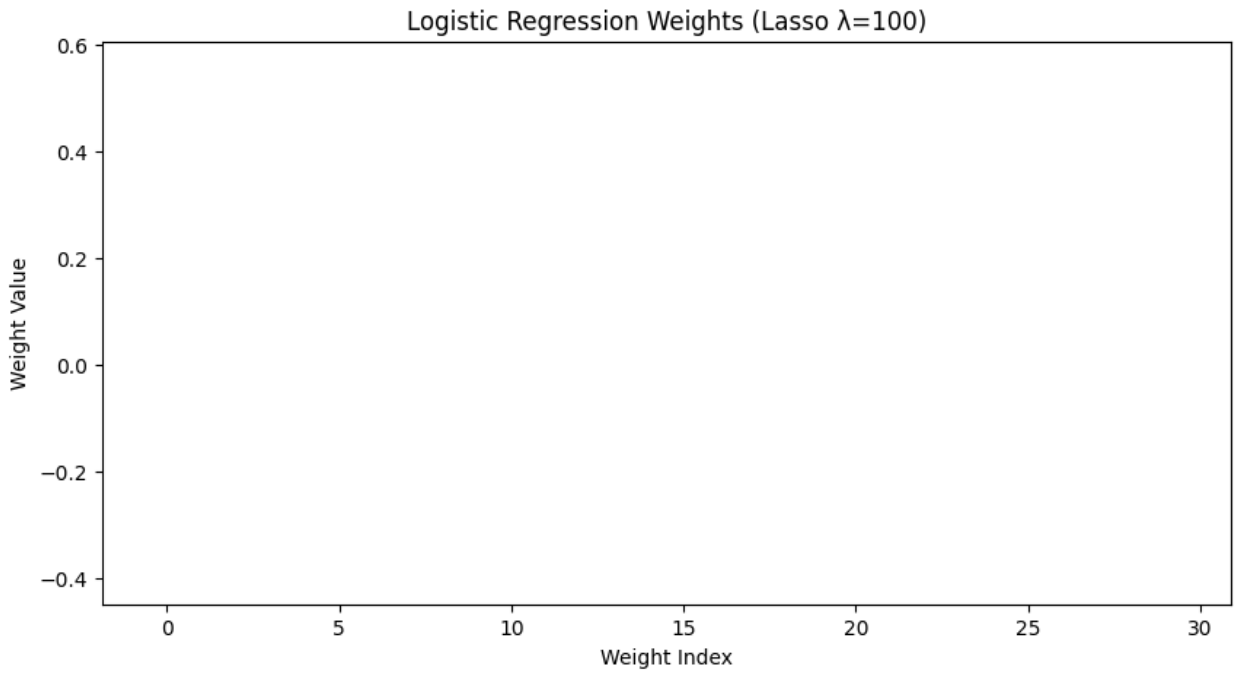
From the formula above, we can see that as the regularization strength λ increases, the penalty on higher weights increases. This potentially will lead to more coefficients being zero and so a sparser model. While this might simplify feature selection, if λ is set too high, the model may become too simple, losing most of the information and a weaker model. On the other hand, if λ is

too low, the regularization may not be effective enough to prevent overfitting, especially when we have a lot of features.









Plotting the weights of these models we can say for each one of them:

1) **<u>No Penalty</u>**: We have a lot of weights with high absolute values and no zero weights, meaning that each feature is being considered by the model as important (importance shown by absolute values)

2) **Lasso λ = 0.5**: We notice almost the same pattern, but with a slight change of some weights towards zero, but still all of them are being considered in the model, even if only with a small impact.

3) **Lasso λ = 10**: The change now is very noticeable. A lot of weights are now zero, meaning we have a sparser model, with fewer candidates when considering feature extraction and feature importance.

4) **Lasso λ = 100**: As expected, since we have no features with weights that are extremely high, this penalty provides no important features, meaning we used a very high penalty. No features are kept, thus we have no model.

The overall pattern is that as regularization strength increases, more weights become zero, simplifying the model and making it easier for feature selection, yet too high values result in an overly sparse model that probably does not capture all the information of our data.