Question 1: Data Cleaning/Encoding

In the data cleaning/encoding step, we first want to understand the data we are working with. This is especially important when deciding the encoding method. We need to determine the number of classes per question and, as a bonus, find out how many responses there are per question. This will help us understand the structure of the questions and their potential importance to the people surveyed. After carrying out the tallying operation and examining the CSV file, the following observations become clear:

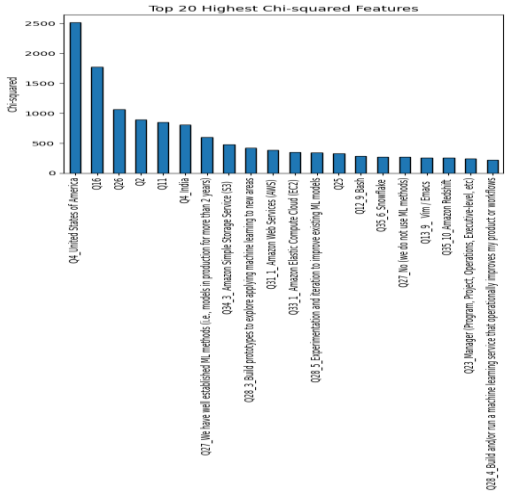
1) The column regarding duration, the first row, and the original target variable column used to create the encoded salaries can be removed. This is because they are redundant. For example, the first row contains only the questions and the duration, which has no impact on the target variable. Additionally, there is a duplicate un-encoded column for the salaries.
2) It is also evident that a few columns will need to be label encoded, and a majority will need to be one-hot encoded.

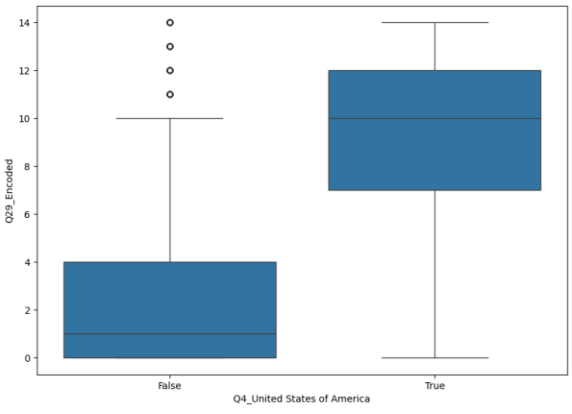
Questions 2, 11, 16, 25, 26, 30, and 43 will need to be label encoded because they represent ordinal data, and we will manually assign their labels. If they have missing values, we will use the mean; however, there aren't many missing values, so the impact on the model's bias will be negligible. Question 9 will be binary encoded since the responses are either 'true' or 'false.' For the remaining questions, we will use one-hot encoding. This choice is logical because they are already one-hot encoded in the CSV, they are categorical, and it will prevent the model from inferring any spurious relationships between the categories. The downside is that it will significantly increase the dimensionality of the dataset.

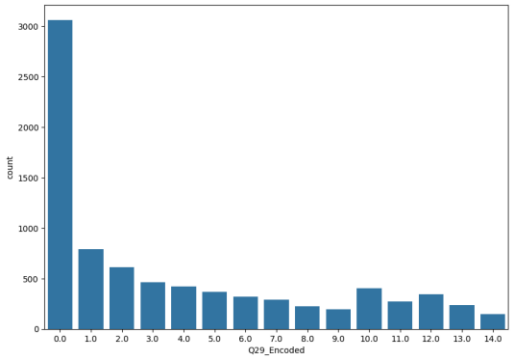
Question 2: Feature Selection

Before proceeding further, we want to examine our target variable and our general features after encoding. By creating a plot of the encoded target variable, we can observe that it is heavily left-skewed, indicating that most data scientists are earning salaries in the range of -20,000$. Therefore, we should anticipate similar behavior from the model.

In the feature selection process, our goal is to choose the features that have the greatest impact on the accuracy of our model, essentially identifying the key characteristics of individuals that significantly influence their salaries. The motivation behind feature selection is twofold: to reduce the computational resources required for training the model and to decrease the dimensionality of the dataset. One approach is to use the chi-squared method, which returns both the chi-squared value and the p-value. A higher p-value indicates a more profound impact of that feature on our target variable. However, one drawback of this method is that it doesn't capture the interrelationships between features and how they might affect each other. In this study, we will use all of the available features, thanks to the availability of a GPU and the use of GPU-accelerated libraries.







Above figures show the importance of each feature and the impact of living in the USA on the target variable, as well as the distribution of the target variable.

Question 3: Model Implementation

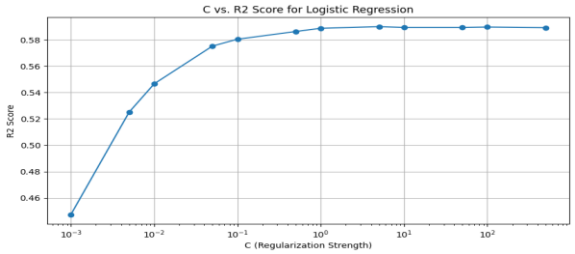
The first step in model implementation is to prepare the training and testing datasets. We will allocate 80% for training and 20% for testing. Despite using the data for generating plots and exploring the dataset, we will start with a fresh set, ensuring it is clean and encoded again to guarantee its reliability. This clean dataset will be readily available for debugging purposes. It's crucial to ensure that the target variable or the testing data does not inadvertently leak into the training data, which would lead to data leakage and an overly optimistic model.

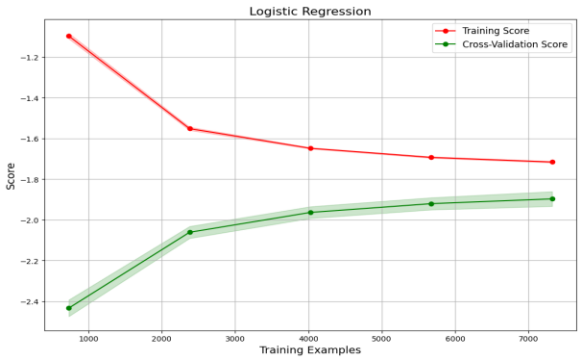
Our model will utilize multiclass logistic regression, and we will implement it using both PyTorch and scikit-learn. Initially, we will use the scikit-learn package to explore relationships between hyperparameters associated with logistic regression and visualize the problem and the model with ease. However, the final model will be built using PyTorch, allowing for GPU acceleration and a more customized and low-level approach to solving our problem.

The multiclass logistic regression model is closely related to ordinal logistic regression, even though it may not be the ideal choice for our application. In the scikit-learn model, the primary hyperparameter is "C," the regularization parameter. In our early model testing, we will iterate through different values of C and plot the R^2. "C," also known as the "Weight Decay" parameter, directly influences the aggressiveness of the decision boundary and significantly impacts the bias-variance trade-off. The plot will help us understand the relationship between variance and bias by adjusting the C parameter. In the PyTorch model, the C parameter is represented as "weight decay," which is 1/C. Notably, in our testing, we've found that scaling our values has no effect on accuracy. This is because our value range is not extensive, and the scaling factor won't make a significant difference.

We will employ 10-fold cross-validation, and since the PyTorch model uses CrossEntropyLoss, it effectively calculates the log-loss for the multiclassification problem. We will use a very stringent performance metric for our problem to avoid presenting the model in an overly optimistic light and to highlight areas for improvement. Our chosen metric is accuracy, which is aggressive in the sense that it marks a prediction as incorrect as long as it doesn't match the correct class, regardless of how close the response is to the true class. It doesn't consider the ordinal nature of the target variable.

It's essential to note that as the number of training examples increases, the negative log-loss decreases, indicating that the model is moving toward overfitting, which is expected when considering the bias-variance trade-off.





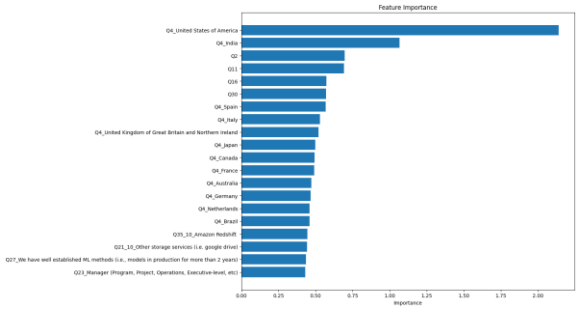
The plots above show the effect of the weight decay variable on the bias variance complex.

Question 4: Model Tuning

We will conduct hyperparameter tuning for the PyTorch model, focusing on several key parameters. First, the weight decay parameter, which governs the aggressiveness of the decision boundary, will be explored. Second, the learning rate, which determines the size of the step the model takes in the direction of the negative gradient during each iteration, will be tuned. Lastly, we will optimize the number of epochs.

To efficiently search for these hyperparameters, we will employ a grid search method combined with 10-fold cross-validation, primarily in question 3. To reduce time complexity, especially as it approaches O(n^3), we will perform separate grid searches for weight decay and learning rate. Once we have determined optimal values for these two parameters, we will incrementally increase the number of epochs until we begin to observe asymptotic behavior in model performance.

After establishing these foundational hyperparameters, we will have the flexibility to experiment with different optimizers and model architectures. While logistic regression models typically employ only one layer, exploring other possibilities can lead to further optimization. Our findings suggest that a learning rate of 0.01 and a weight decay of 0.1, along with 300 epochs, provide favorable results.



As observed, the country of residence has the most significant impact on determining the target variable, which aligns with our earlier predictions from question 2.

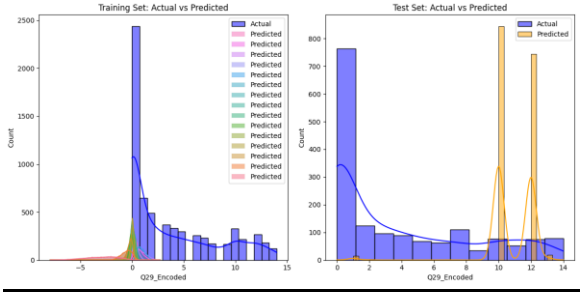
Furthermore, by experimenting with different solvers and using the optimized learning rate and weight decay values, we managed to enhance the model's accuracy by approximately 5%.

Question 5: Testing and Discussion

The model is undoubtedly underfitting and performing poorly, achieving only around 5% accuracy compared to the 41% accuracy during training. The root of the problem likely lies in the model's lack of complexity relative to the large number of features, making it ill-equipped to handle the dataset's diversity. Additionally, it exhibits a strong bias towards predicting two of the encoded categories, which may be attributed to suboptimal encoding. Other potential causes could include the need to explore a wider range of "C" values, inversely lower weight decay, and learning rate values, tested at much higher epochs.

Another potential reason for the model's struggle is the choice of optimizer and performance metric. Accuracy may be too harsh of a metric for a problem with this many features, leading to incorrect results. There might be alternative optimizers specifically designed to address such issues.

To tackle a problem with a high feature dimensionality, a better approach would involve using deeper models with multiple layers or considering feature reduction techniques to reduce the number of features being used in the model. This could help capture more complex relationships within the data and potentially improve performance.



Side note: The code runs completely on google collab after restarting run time and setting it to gpu, thew csv will need to be upload with the gui in the first code block. Some of the code blocks do take a while to run, however models and their state dicts are saved, especially the model needed for question 5. Thank you