Objective: Construct a model that will predict if the customer will click on 'apply' icon based on the several features

Following are the steps I took to solve this problem in sequential order:

- **Sorting**: Sort the value according to the date and separate training and testing as mentioned in the question
- **Dropping:** Dropped the column which contained the timestamp and also class_id which was supposed to be used for the second part
- **Distribution:** Checked the distribution for apply and non-apply rate – It was skewed with only 10% users clicked on apply. This intuitively made sense because not all users who visit the website, apply for the position
- **Metric**: Also, because it is a binary classification problem and data is skewed, I would be using AUC metric for measuring how good my model is (Same is mentioned in the question as well)
- **Null values**: Checked null values – Only 3 columns contained null values. First 2 columns contained around 95% values as '0' and hence I replaced the missing value with '0' for the first two column. Last column had an equal distribution of 1 and 0. I had several options to with me to replace the null values like replacing with mean, median or mode or completely removing them. Because the column contained only 1 or 0 values, I chose not to go with replacing with mean because then it would give a non-integer value. I tried removing the null values and again check the distribution of apply and non-apply. If the distribution is same, I would go ahead with removing the null values. After removing the values, distribution was indeed similar and hence I decided to remove the rows which contained NaN values.
- **Correlation**: Next thing I checked was if any column is correlated with each other - When predictor variables are correlated, the precision of the estimated regression coefficients decreases as more predictor variables are added to the model. None of the columns were significantly correlated with each other
- **Visualization**: Because the data is a 7-dimensional data, it is difficult to visualize it. Using t-SNE we can visualize the data in 2-dimensional space. T-SNE is not only used for better visualization but also for clustering the data. In our case, apply and non-apply rate. After plotting the t-SNE plot, it was clear that apply and non-apply cases were very close and hence it would be difficult for the algorithm to separate out the two classes.
- **Auto-Encoder**: We will create an autoencoder model in which we only show the model non-apply cases. The model will try to learn the best representation of non-apply cases. The same model will be used to generate the representations of apply cases and we expect them to be different from non-apply ones. But even after using auto-encoder and then plotting t-SNE plot, apply and non-apply cases were not separable.

- **Distribution plot**: I plotted the distribution plot of all the features with apply rate feature and there was an unusual thing that I noticed. City_match had a bimodal distribution. There were two normal distributions with mean 0 and 1. (for both training and testing set). This indicates that there is some pattern in the dataset. I wasn't really clear how there were two peaks in the model. Also, the other distributions were highly skewed with mean almost equal to zero. One reason may be due large number of outliers

- **Outliers**: I used boxplot to check the number of outliers in the dataset. As expected, there large number of outliers in the dataset. One way to remove outliers is to make the distribution normal, take the z score and remove the values which lie beyond +/- 3 standard deviation as approximately 99.6% values lie within 3 standard deviation. After making the transformation, I saw that values which are inside 3 standard deviation did not contain any apply values. This was a great insight and helped me the build the model accordingly.

1. **Modeling**

- As we saw in the last step, values which are inside the 3 standard deviation only contains non-apply values. Therefore, I build a model where if the z score of the test data set falls within 3 standard deviation of the it will be considered non-apply (0) and for values which are outside +/- 3 standard deviation, I build 5 models i.e. Random Forest, XGBoost, SVM, Logistic Regression and Neural Network.

2. **Results for Part A.**

- Random Forest, XGBoost and Neural Network all performed almost similar with an AUC score of 0.94
- Logistic Regression performed the worst with AUC score of 0.84
- SVM took too much time to run and therefore had to stopped

3. **Part B**: where we had to use class-id column, I used One Hot encoding and then used XGBoosting and Lasso. I used Lasso because it is used for dimensionality reduction, since after one hot encoding, number of features increased by 157.

4. **Results for Part B.**

- Lasso performed better than XGBoosting although AUC score for both the algorithms was poor.

5. **Suggestions**

- Given more time, I would have first performed PCA on part b. to reduce the number of features and then tuned the machine learning algorithms. Because of time constraint, I was able to get only 0.55 AUC for Lasso and 0.50 for XGBoosting which is almost equal to a random guess
- I would have tried hyperparameter tuning on the above-mentioned algorithms which would further improve the AUC score
- Getting more data is always useful as we are able to train our model better (also being cautious about not over fitting the model)
- SVM might have performed better than other algorithms in part a. because we obtain a maximal separation margin between classes which allows best generalization for unseen data
- **NOTE: T-SNE takes lot of time to run, therefore I had taken a screenshot of before and after applying autoencoder and using those files for the representation in jupyter**