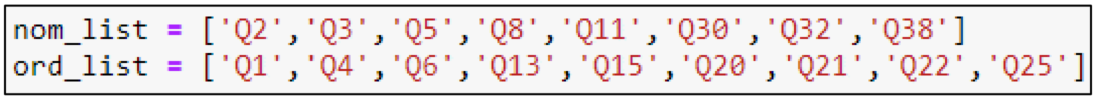
Name: Fendi Halim Tjoa

Multi-class Ordinal Logistic Regression on Data Scientists' Salary

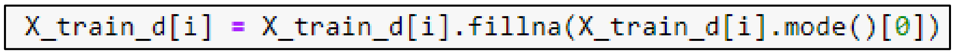
Part 1: Data Cleaning

The raw *clean_kaggle_data_2020.csv* file contains surveys of participants in data science community. There are 39 main questions and 8 supplementary questions, and all have been appended into *.csv* file. Firstly, the raw data is converted to pd.DataFrame and split into 70% train and 30% test. The reason the data is split in the beginning is beneficial for filling missing data because train and test sets should be treated independently to each other. All the data in this dataset are categorical and therefore one of the main goal of data cleaning and preparation is to fill in missing value and convert all the categorical (string format) into numerical (integer). During this stage, what is done at the training set is applicable to testing set. First, missing value can be filled in using **.mode()** because the data is still in string form. The columns depicted on figure 2 can be encoded using **OneHotEncoder()** if it is under nom_list, and **OrdinalEncoder()** if it is under ord_list (orders can be observed in the jupyter notebook). After all of this, the dataset now contains only numerical value.



```
nom_list = ['Q2','Q3','Q5','Q8','Q11','Q30','Q32','Q38']
ord_list = ['Q1','Q4','Q6','Q13','Q15','Q20','Q21','Q22','Q25']
```

Figure 1: Nominal List and Ordinal List

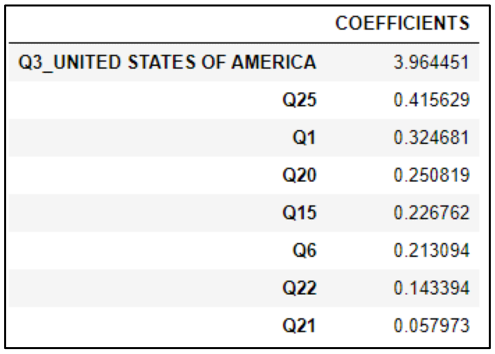


```
X_train_d[i] = X_train_d[i].fillna(X_train_d[i].mode()[0])
```

Figure 2: Using Mode to Fill Missing Data

Part 2: Exploratory Data Analysis and Feature Selection

After data cleaning, the datasets will be very massive (413 features) due to encoding. Feature selections are important to tackle this to weed out unnecessary features. Not only that, reducing features can prevent the model from the curse of dimensionality. There were two models were created to do feature selection. **Linear_model.Lasso** is the preferred model over **ensemble.RandomForestRegressor** as the latter is not very good at detecting highly correlated features. Higher lasso alpha means more restriction on the coefficients and low alpha means more generalization. In this dataset, alpha = 0.1 is chosen as the baseline. Feature selection will be the features that have positive and nonzero coefficients. From observing figure 4, there are not too many highly correlated features which are good for implementation of machine learning. Highly correlated features can increase variance and make the model very sensitive to data.



	COEFFICIENTS
Q3_UNITED STATES OF AMERICA	3.964451
Q25	0.415629
Q1	0.324681
Q20	0.250819
Q15	0.226762
Q6	0.213094
Q22	0.143394
Q21	0.057973

Figure 3: Non-zero Positive Coefficient from Lasso

Part 3 & 4: Model Implementation and Model Tuning

Ordinal logistic regression can be achieved by utilizing **LogisticRegression()** and modifying y_train as a classifier by grouping them into its own category. Firstly, y_train needs to be derived. For example, if the target = 3, then the value of target<0 will be 1, target<1 will be 2, target<3 will be 0. By using that logic, modified y_train.Q24_ENCODED look like figure 5.

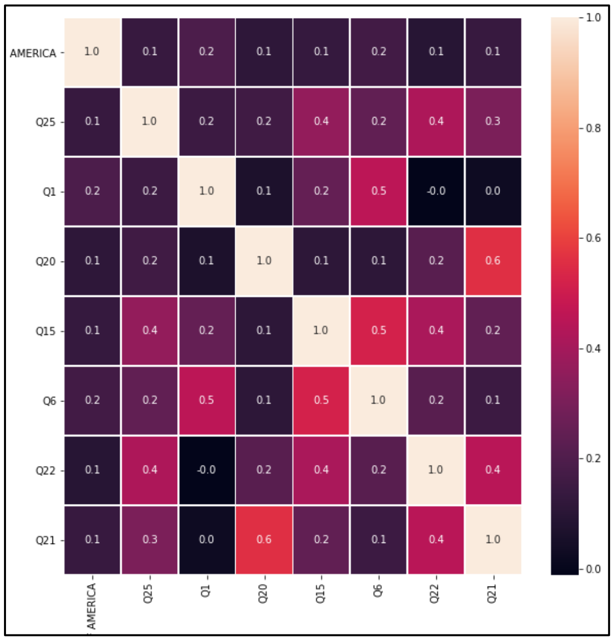
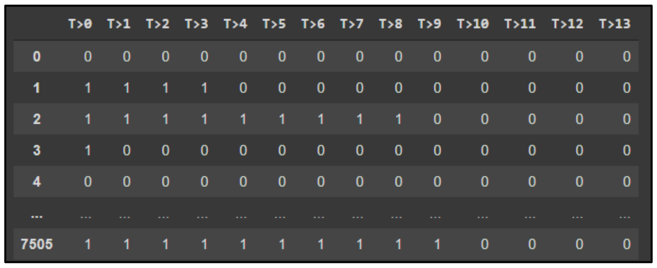


Figure 4: Heatmap of Feature Correlation

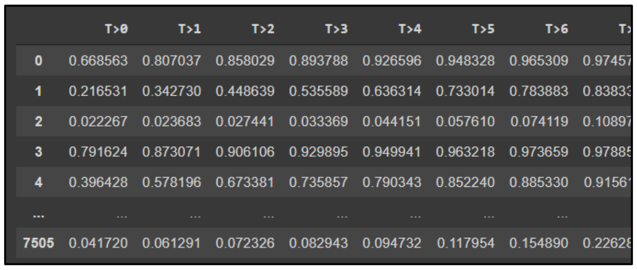
The steps of ordinal logistic regression classification:

1. Creating a binary classification. Model is be fitted using X_train and Y_test[T>0] and predict probabilities using **predict_proba(X_test)**.
2. The result is a tuple of probability of y=0 and probability of y<0, respectively.
    a. The necessary probability is of index 0 of the tuple.
3. Store the probabilities of step 2 in a DataFrame for calculation later.
4. Subtracting the probabilities to get the correct probability of each class.
5. Repeat step 1-4 for all derived train data.



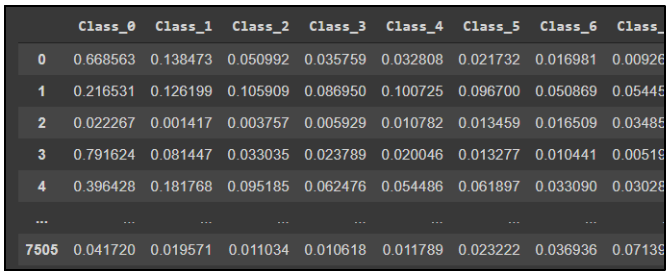
	T>0	T>1	T>2	T>3	T>4	T>5	T>6	T>7	T>8	T>9	T>10	T>11	T>12	T>13
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7505	1	1	1	1	1	1	1	1	1	1	0	0	0	0

Figure 6: Table of Derived Train Target Data (Pre-Step 1)



	T>0	T>1	T>2	T>3	T>4	T>5	T>6	T>
0	0.668563	0.807037	0.858029	0.893788	0.926596	0.948328	0.965309	0.97457
1	0.216531	0.342730	0.448639	0.535589	0.636314	0.733014	0.783883	0.83833
2	0.022267	0.023683	0.027441	0.033369	0.044151	0.057610	0.074119	0.10897
3	0.791624	0.873071	0.906106	0.929895	0.949941	0.963218	0.973659	0.97885
4	0.396428	0.578196	0.673381	0.735857	0.790343	0.852240	0.885330	0.9156
...	...	...	...	...	...	...	...	
7505	0.041720	0.061291	0.072326	0.082943	0.094732	0.117954	0.154890	0.22628

Figure 5: Snapshot of Step 3



	Class_0	Class_1	Class_2	Class_3	Class_4	Class_5	Class_6	Class_
0	0.668563	0.138473	0.050992	0.035759	0.032808	0.021732	0.016981	0.00926
1	0.216531	0.126199	0.105909	0.086950	0.100725	0.096700	0.050869	0.05445
2	0.022267	0.001417	0.003757	0.005929	0.010782	0.013459	0.016509	0.03485
3	0.791624	0.081447	0.033035	0.023789	0.020046	0.013277	0.010441	0.00519
4	0.396428	0.181768	0.095185	0.062476	0.054486	0.061897	0.033090	0.03028
...	...	...	...	...	...	...	...	
7505	0.041720	0.019571	0.011034	0.010618	0.011789	0.023222	0.036936	0.07139

Figure 7: Snapshot of Step 4

The preferred performance metric for this dataset is micro F1-score because the dataset is multiclass and unbalanced. Micro F1 score takes the precision and recall for each class and average them to calculate F1 score. The average F1-score using default setting of this training model is 53.47% with variance of 0.01%. To improve the performance of this model, model tuning can be done by tuning two chosen hyperparameters which are C and solver. Using C = 0.001 and solver = liblinear, tuned training model average f1-score is 56.01% with variance of 0.01%. In theory, if C is small, the model will lead to underfitting as lambda is large. When C is large, the model will lead to overfitting because lambda is small. In this model, various C has been tested and cross-validated and therefore the variance and bias are not an issue for underfitting or overfitting. Scaling / normalization is not necessary because the all the values in this dataset have been encoded.

Part 5: Testing & Discussion

The F1 score of test dataset is 52.18% and it is slightly lower than the training model by approximately 3.8%. The overall fit is the model is good as the F1 score for training and test are similar. Linear regression model is one of the machine learning models that is considered resistant to overfitting because of limited number of parameters. So, it is safe to say that this model is not overfitting. Underfitting on the other hand could be a potential as the f1 score is close to 50% which makes the model not sure when it is right or wrong.

The distribution depicted from figure 8 and figure 9 show the same true target distribution between training data and testing data. They share the similarities in unbalanced data with majority of the class is class 0 (salary of $0-$9,999). Additionally, the predicted target distribution between training data and testing data are similar as well in which the machine predicts more class 0 vs the other classes.

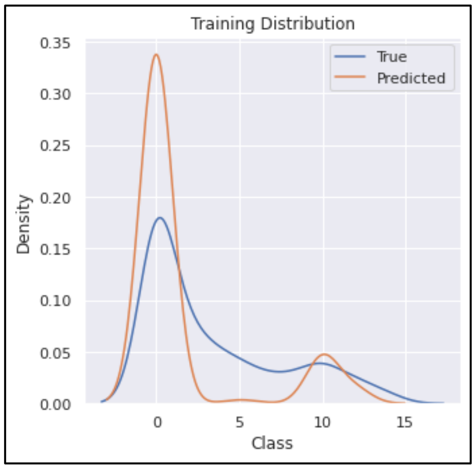


Figure 8: Training Distribution

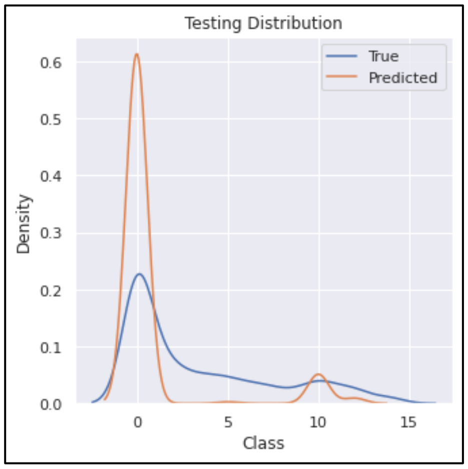


Figure 9: Testing Distribution

There are few suggestions to improve this model to get better accuracy / score. The first suggestion is to improve the training data. The challenges with this data set include unbalanced output and variety of classes as the output. If there is a way to get a better training data, then the predictions and accuracy will improve greatly. However, these challenges are hard to combat as they come from survey and are unpredictable. The next suggestion is to use different model with tree-based learners such as EBOC, C4.5, or Random-Tree. By using these models, tree-based learners can help to create branches (conditions) to predict multiclass outputs and therefore more robust in for this dataset.