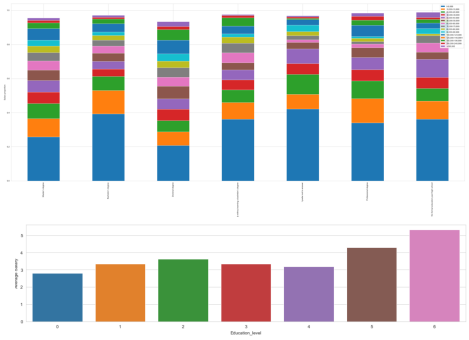
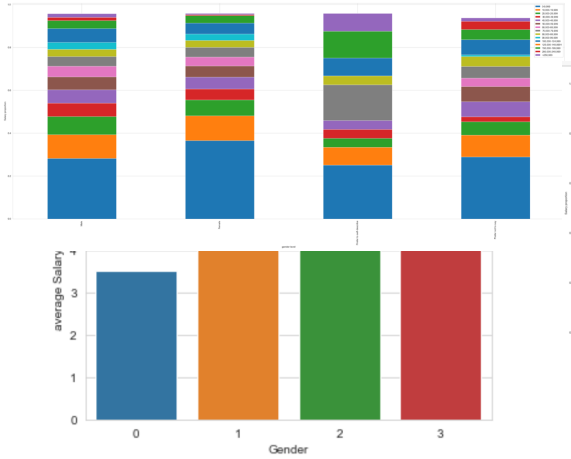
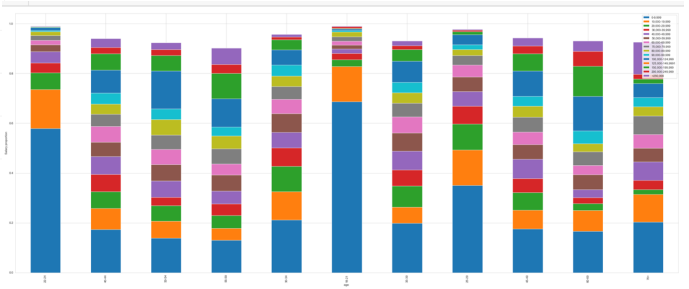
Exploratory data analysis



Education level vs salary



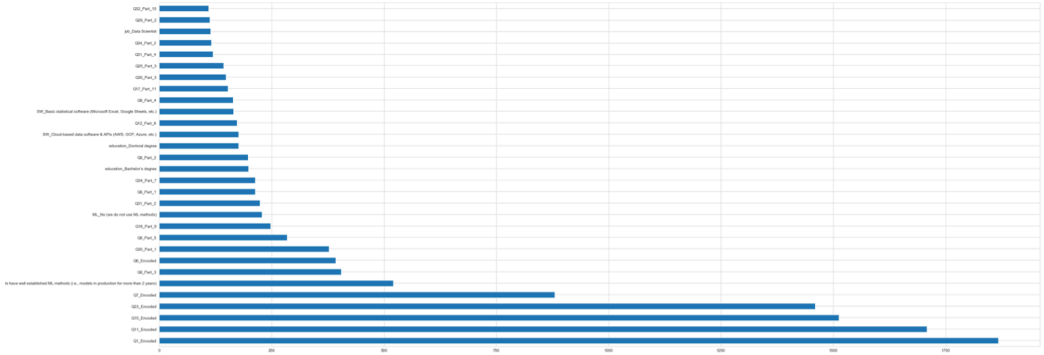
Gender vs salary



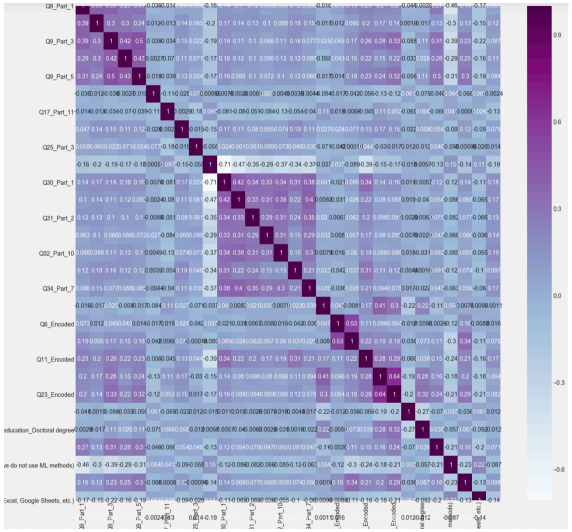
age vs salary

The male has higher income than female. But, we cannot compare the feature 'prefer not to say' and 'prefer to self-describe'. The people with age in range from 18-21 and 22-24 has lower salary, which mean senses because young people usually has lower salary in the real world. As age goes up, we can see the distribution with more higher salary level. The doctoral degree and professional degree has the higher average salary incomes, which can show the trend that higher education level gives the higher salary income, and they have implicit relations.

important features & feature selection



The chi-square test helps us to solve the problem in feature selection by testing the relationship between the features. When two features are independent, the observed count is close to the expected count, thus we will have smaller Chi-Square value. So high Chi-Square value indicates that the hypothesis of independence is incorrect. In simple words, higher the Chi-Square value the feature is more dependent on the response and it can be selected for model training



Features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable. So, when two features have high correlation, we can drop one of the two features. For example, Q9_part_5 and Q9_part_3 has a high correlation, and Q23_encoded and bachelor's degree has a high correlation, which is 0.64. We can drop one of them. Because they are duplicate value.

Model implementation
Model tuning

```
kfold = KFold(n_splits=10)
kfold.get_n_splits(X)

accuracy = np.zeros(10)
np_idx = 0

for train_idx, test_idx in kfold.split(X):
    X_train, X_test = X.values[train_idx], X.values[test_idx]
    y_train, y_test = y.values[train_idx], y.values[test_idx]

    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)


    pro_train=multi_class_ordinary_train(X_train, y_train,X_test,1,'liblinear')
    class_index_train=np.argmax(pro_train, axis=1)
    Accuracy_score=accuracy_score(y_train, class_index_train)*100
    accuracy[np_idx] = Accuracy_score
    np_idx += 1

    print ("Fold {}: Accuracy: {}%".format(np_idx, round(Accuracy_score,3)))

print ("Average Score: {}%({}%)".format(round(np.mean(accuracy),3),round(np.std(accuracy
```

Cross validation

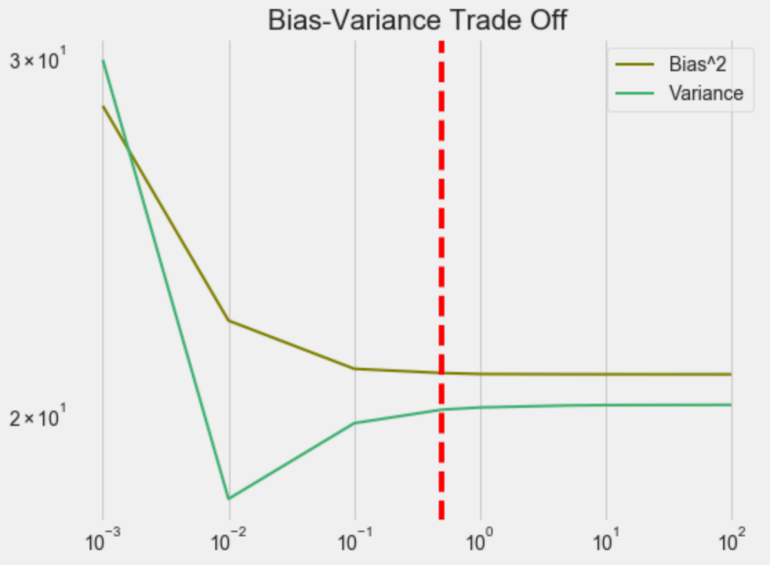
The accuracy is around 30 percent, the lowest is 31.20 %, the highest is 32.44% . The average is around 31.922 %. The variance is 3.249%. The higher the average accuracy, the lower the bias. The lower the standard deviation, the lower the variance. This better represents the true performance of the model on the training set.

```
for C in [0.001,0.01,0.1,0.5,1,5,10, 100]:
    for solver in ['newton-cg','lbfgs','liblinear','sag']:

        accuracy = np.zeros(10)
        np_idx = 0

        for train_idx, test_idx in kfold.split(X):
            X_train, X_test = X.values[train_idx], X.values[test_idx]
            y_train, y_test = y.values[train_idx], y.values[test_idx]

            X_train = scaler.fit_transform(X_train)
            X_test = scaler.transform(X_test)


            pro_train=multi_class_ordinary_train(X_train, y_train,X_test,C,solver)
            class_index_train=np.argmax(pro_train, axis=1)
            Accuracy_score=accuracy_score(y_train, class_index_train)*100
            accuracy[np_idx] = Accuracy_score
            np_idx += 1

            #print ("Fold {}: Accuracy: {}%".format(np_idx, round(Accuracy_score,3)))
        if np.mean(accuracy) > best_accuracy:
            best_params = {'C':C, 'solver':solver}
            best_accuracy = np.mean(accuracy)
            best_std = np.std(accuracy)
print (best_params)
print ("Average Score: {}%({}%)".format(round(np.mean(accuracy),3),round(np.std(accuracy

print ("\nThe optimal log model uses C={}, and a {} solver, and has a cross validation s
```

Grid search Tuning the hyperparameter

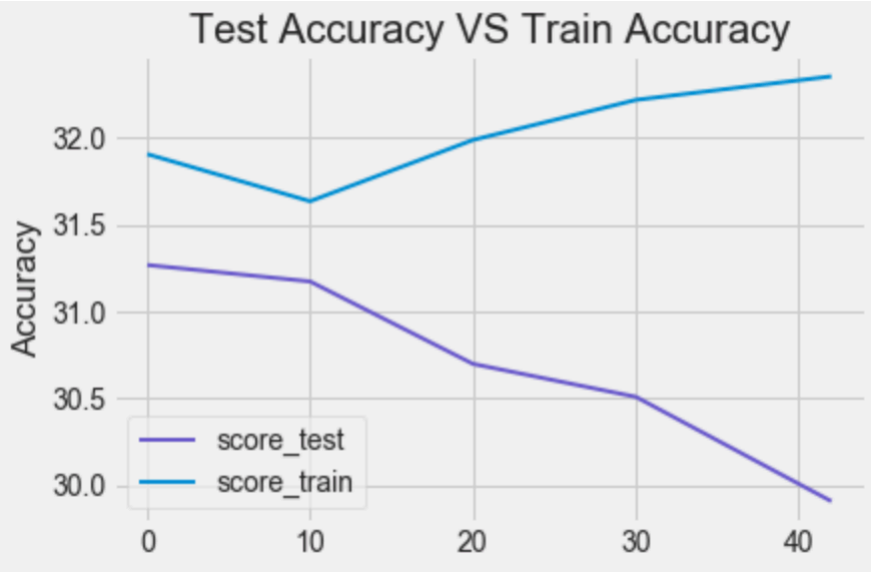
In this result, we use accuracy as our metric to measure the model performance. The result shows that the optimal log model uses C=0.1, and a newton-cg solver, and has a cross validation score of 31.924% with a standard deviation of 0.387%

Bias and Variance Trade Off



We need to trade off the variance and bias. So, when C is equal to around 0.1 and 0.5, the both bias and variance both turn to be small. We can see the result match the optimal model result obtained from the grid search.

Model result



As seen from the graph below, the random state for train test dataset split is set as variable, the model perform better for training set. The test set performance is very close to the training set, which all have the similar accuracy. The difference is the accuracy of model applied to test dataset is a little bit lower than the accuracy for the training dataset. It makes sense because the training accuracy should be higher than the test accuracy. The model fit the training accuracy first. However, the both training accuracy and test accuracy is about 30 percent, which means the ordinal logistic regression model does not work for this dataset that include many categorical feature. It is not overfitting or underfitting? Because these two accuracy are so close. It is hard to increase the accuracy very much. Unless, we add more numerical features to our dataset. Or we need to change a model which can fit the categorical feature better.