Credit Card Default Challenge

Coding challenge for the Cynaptics Club Inductions

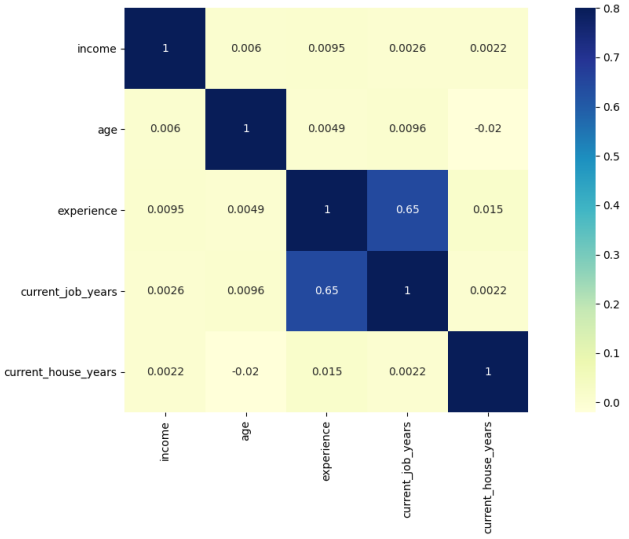
-Ashutosh Nayak

Problem Statement: To predict whether an account has a risk of default or not, when given various parameters about the account

Solution:

Initially, I dropped all the parameters that I considered to be garbage parameters. These dropped parameters included Id, profession, city and state.

Then, I plotted a heatmap to check for any correlation between the remaining parameters.

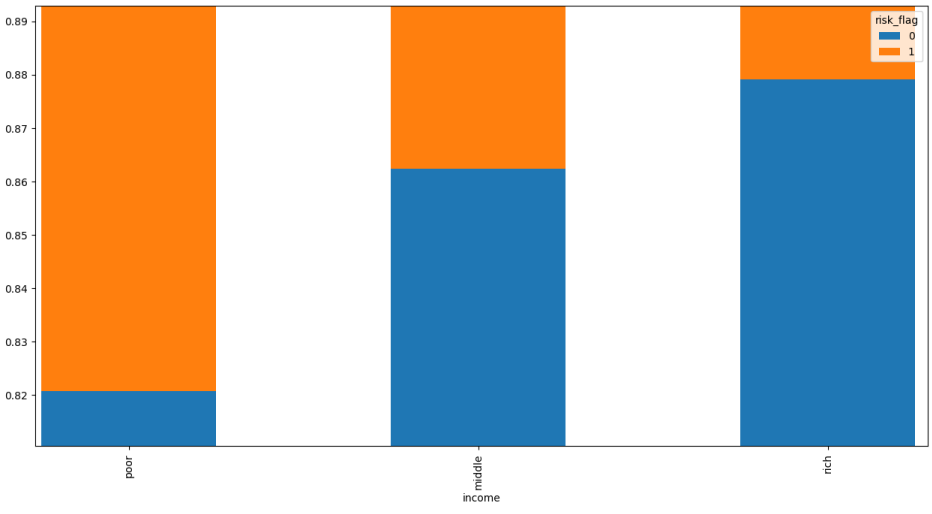


There was considerable correlation between the parameters experience and current_job_years.

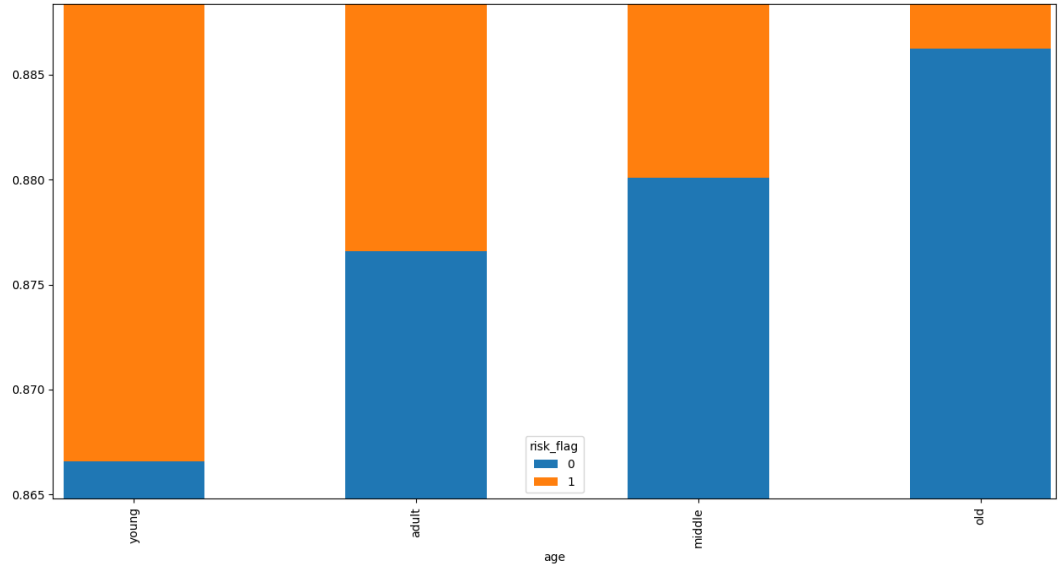
Next, just to ensure I was doing the right thing by dropping those parameters earlier, I plotted graphs to check whether the various parameters of a given sample actually affected the chance of the credit card defaulting. I did this for profession, state as well as all the other parameters I hadn't removed. For many of the parameters (like income, experience, age etc) I made bins with appropriate ranges to have a clearer graph. The results were:

<u>Income</u>:

After dividing the samples into bins, this graph was obtained. The variation is not as large as I initially expected (only 6% difference between poor and rich), but it is not negligible.

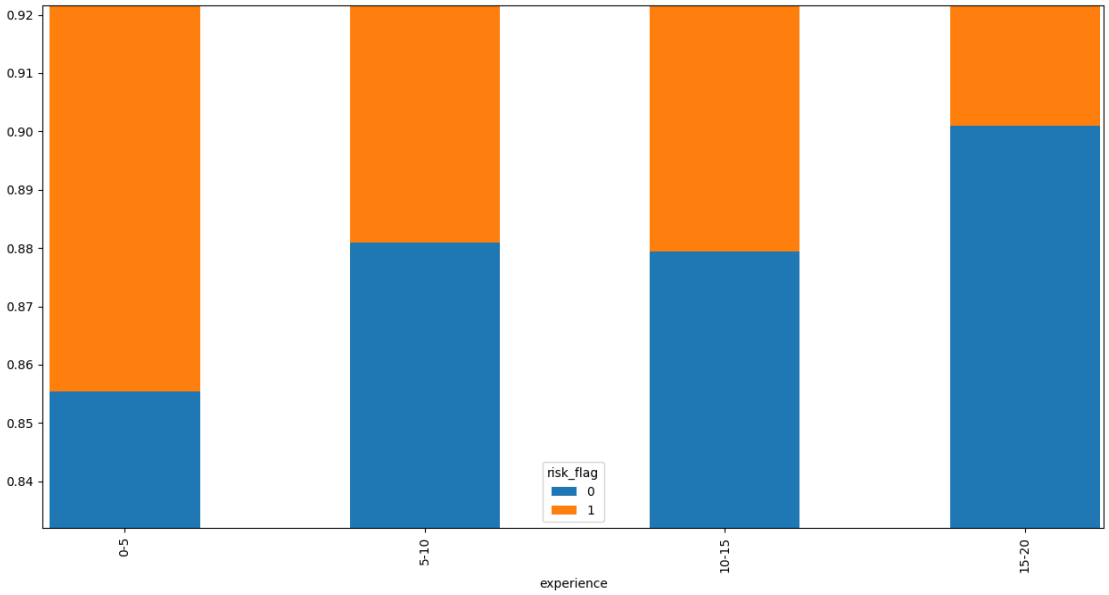


Age:



The variation is even lesser in case of age. There is only about 2% difference between the lowest bin and the highest.

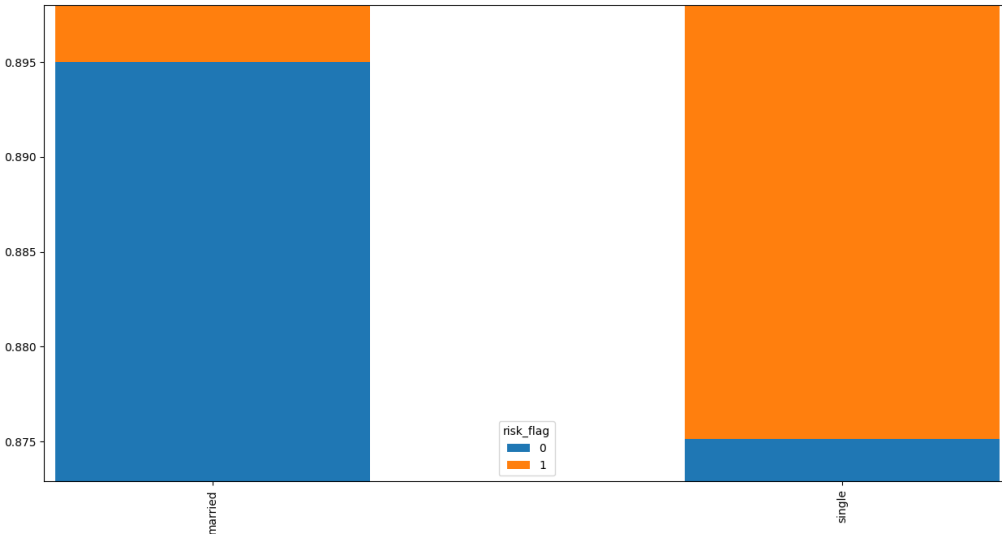
Experience:



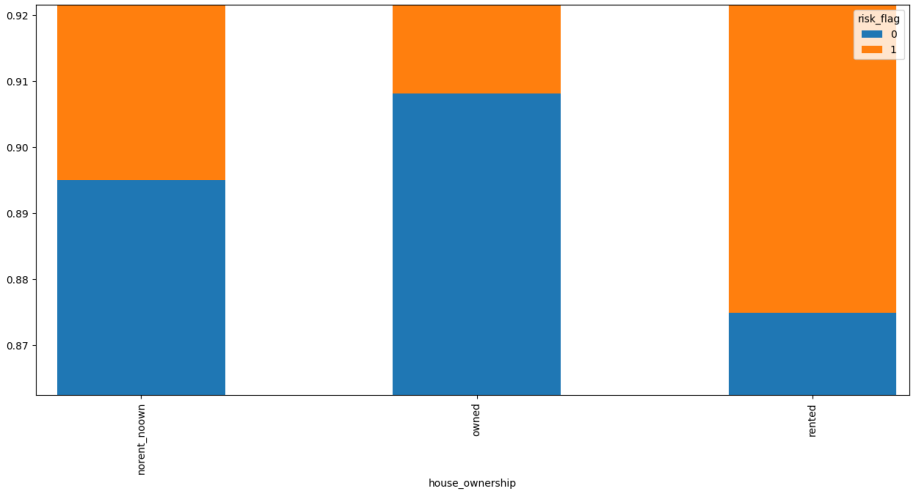
After dividing into bins as shown, this graph was obtained. The variation is slightly higher than in case of age.

Married:

Married people are slightly less at risk of defaulting on their credit card. The difference is less than 2%.



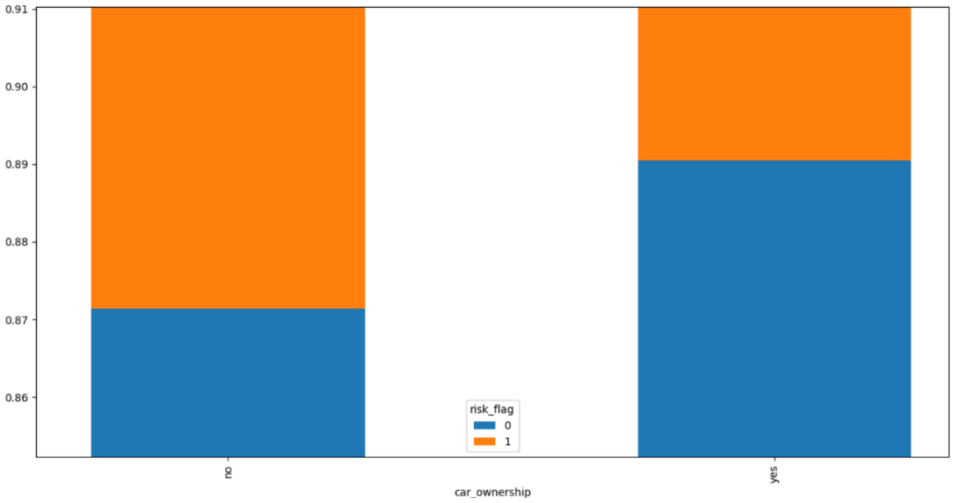
House Ownership:



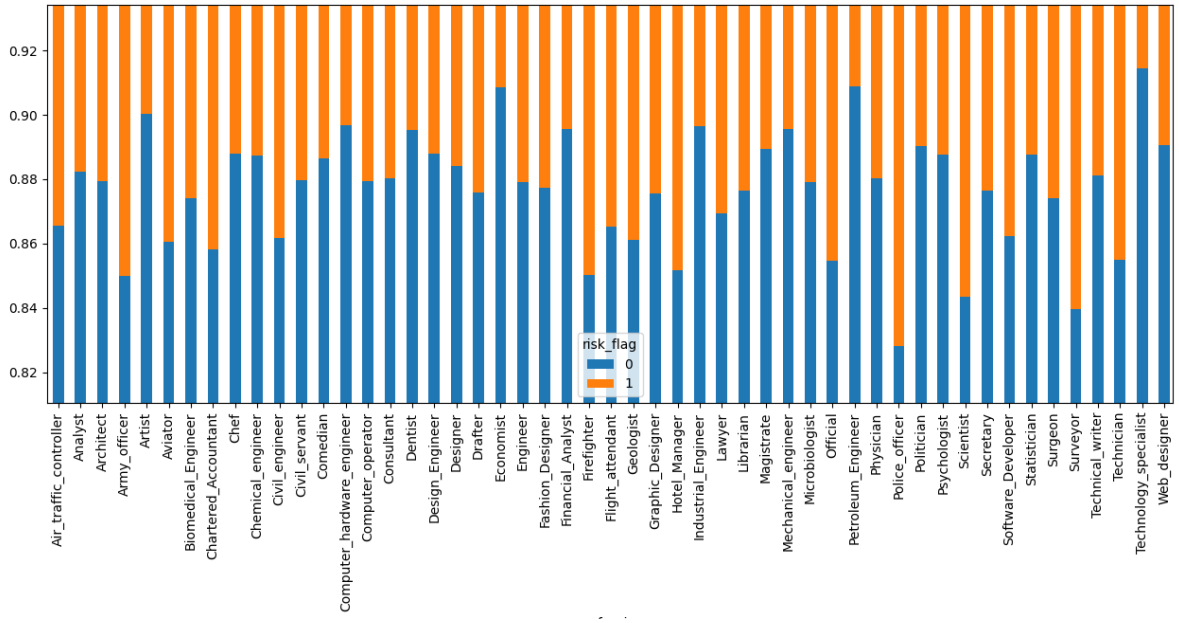
Surprisingly, the people who live in a rented house are slightly more likely to default on their credit card than those who neither own a house nor pay rent (people dependent on others for shelter maybe).

Car Ownership:

It seems like people who own a car are slightly less likely to default on their credit card. But the difference is only 2%.

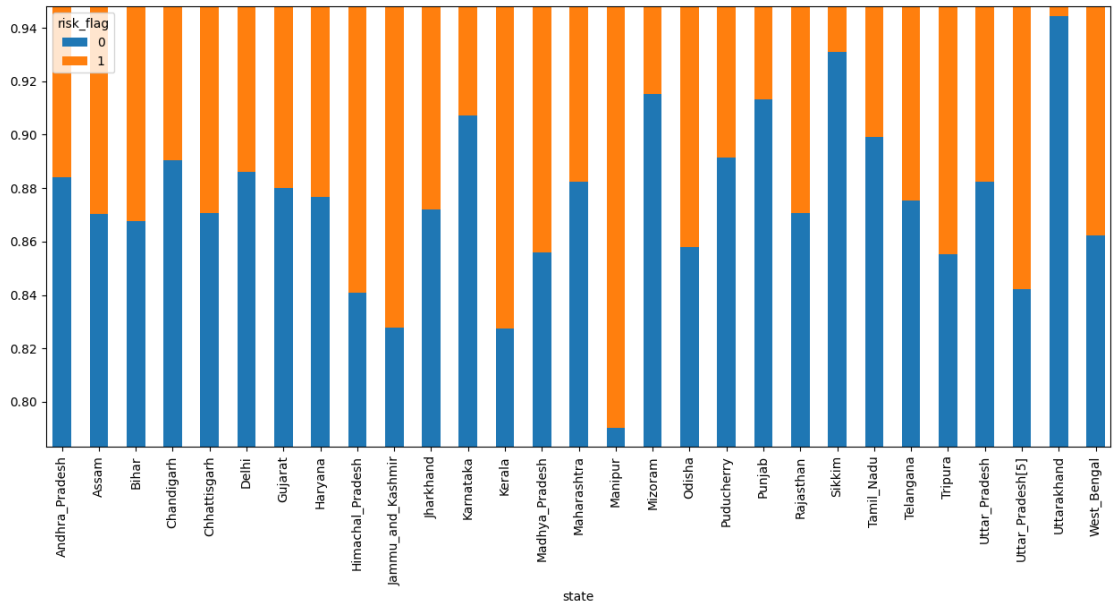


Profession:



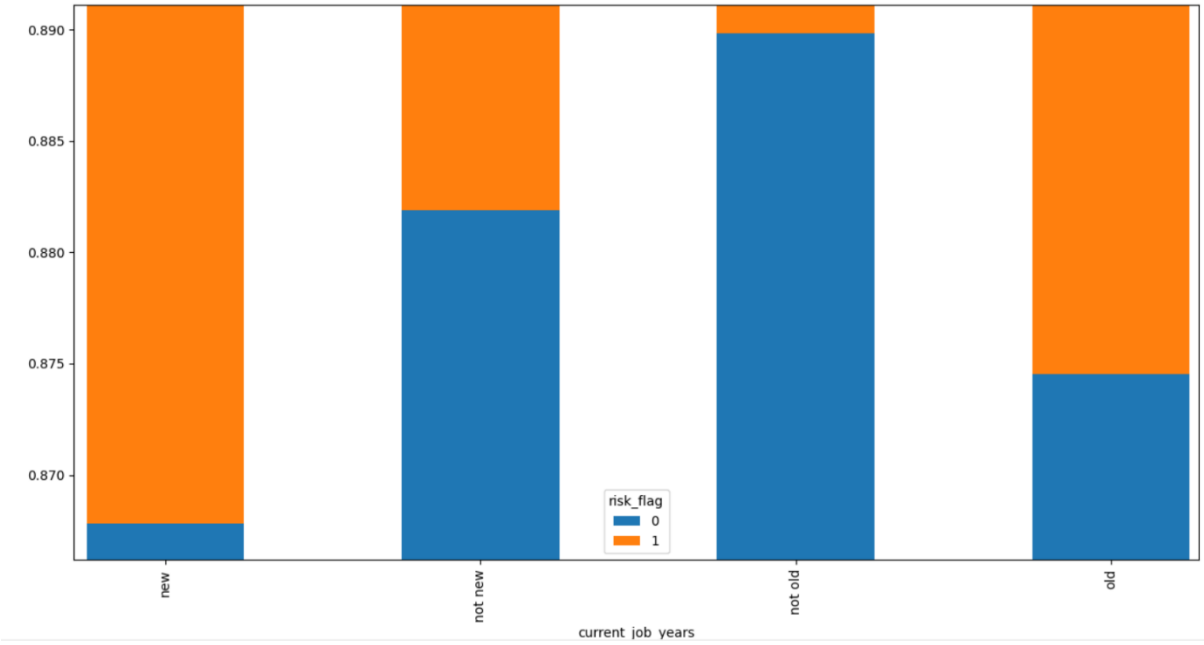
In general, there will probably be no significant difference between any two given professions. But there are a few professions which have considerably lower or higher proportions of credit card defaulters. If the model can support taking into account so many extra parameters (extra parameters will be generated upon creating dummies) without sacrificing its accuracy, it is not a bad decision to consider profession as a parameter.

State:



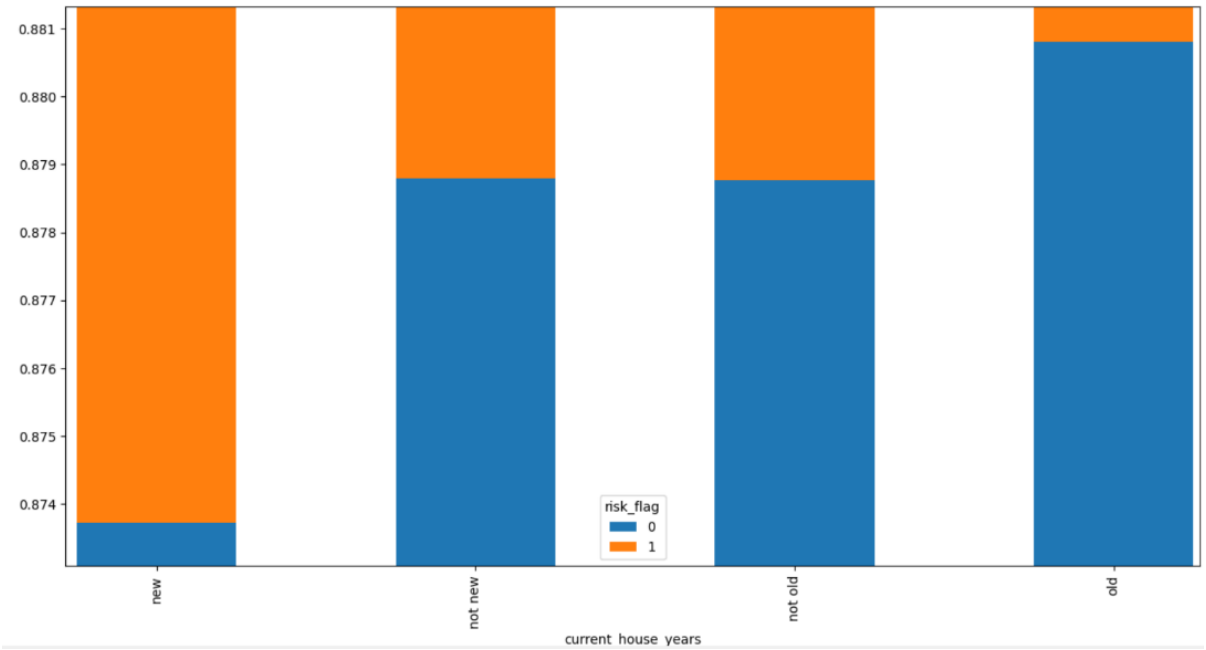
Very similar situation to that of the Profession parameter. Again, this may be a considered as a parameter if the model can handle all the dummies generated without losing accuracy.

Current Job Years:



The variation is only by around 2% at max, between the two extreme cases. Furthermore, there was a correlation between current_job_years and experience parameters. So, I decided to keep experience and drop this parameter.

Current House Years:



The maximum variation between the extreme cases is only about 0.7%. This is a negligible difference, so I dropped this parameter.

Finally, after considering above data, I dropped the parameters “city”, “current_job_years”, “current_house_years” as well as “married” from both train as well as test datasets.

Then, I got dummy values to replace non-numeric parameters like “state”, “profession”, “house_ownership” and “car_ownership” in both sets. I then separated the risk_flag column from the train dataset.

Then, I used a scaler to scale the parameters of both train and test parameters. I tried both StandardScaler() as well as MinMaxScaler(), but StandardScaler() seemed to result in slightly better predictions so I stuck with StandardScaler().

I observed that only about 12% of the samples in the train dataset had a risk_flag value of 1. If I used the train dataset as it is, the model would learn to predict abnormally more 0s than 1s just to get a higher precision. Hence, I used RandomOverSampler() to oversample the cases with risk_flag equal to 1. I also tried using SMOTE() and ADASYN() for oversampling but it seems they could not synthesize appropriate minority class samples as the model’s performance became very bad as compared to RandomOverSampler().

Finally, coming to the model, I used the RandomForestClassifier(). I tried several other models, including LogisticRegression(), LogisticRegressionCV(), DecisionTreeClassifier() and even an Artificial Neural Network.

In the end, RandomForestClassifier() and DecisionTreeClassifier() had the best results, with RandomForestClassifier() being slightly better and more consistent with the results.

After testing (by splitting the train.csv file using train_test_split())for multiple days with different models, hyperparameters, scalers, oversampling methods etc, I finally decided the optimum model as described above, and ran it upon the full train.csv file. Then I exported the predictions to a .csv file and uploaded it to Kaggle.

Results:

The model’s results would be something like this, very precise at predictions of one label but not as precise for the other. Overall, there was a good level of accuracy. The confusion matrix was also suggesting a good model.

```
              precision    recall  f1-score   support

           0       0.97      0.92      0.94      4382
           1       0.58      0.78      0.66       618

    accuracy                           0.90      5000
   macro avg       0.77      0.85      0.80      5000
weighted avg       0.92      0.90      0.91      5000

================================================================================
The confusion matrix is:
[[4035  347]
 [ 138  480]]

Process finished with exit code 0
```