期末報告 機器學習

Jane Street Market Prediction

組別：大地之蓋亞

系級：資訊管理系三 A

組長：B10756038 施宗佑

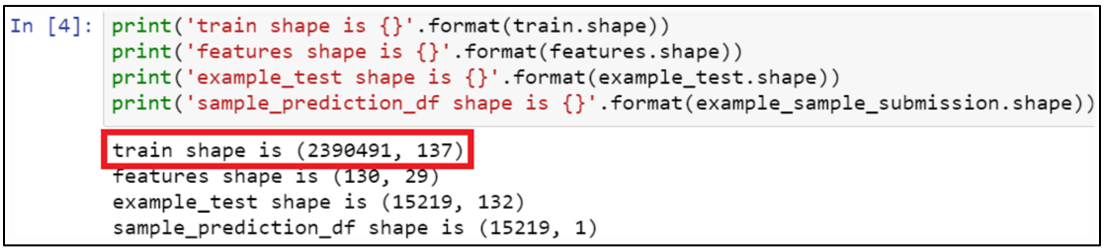
組員：B10756026 林峻儀

組員：B10756040 郭家偉

報告日期：2021 年 1 月 11 日

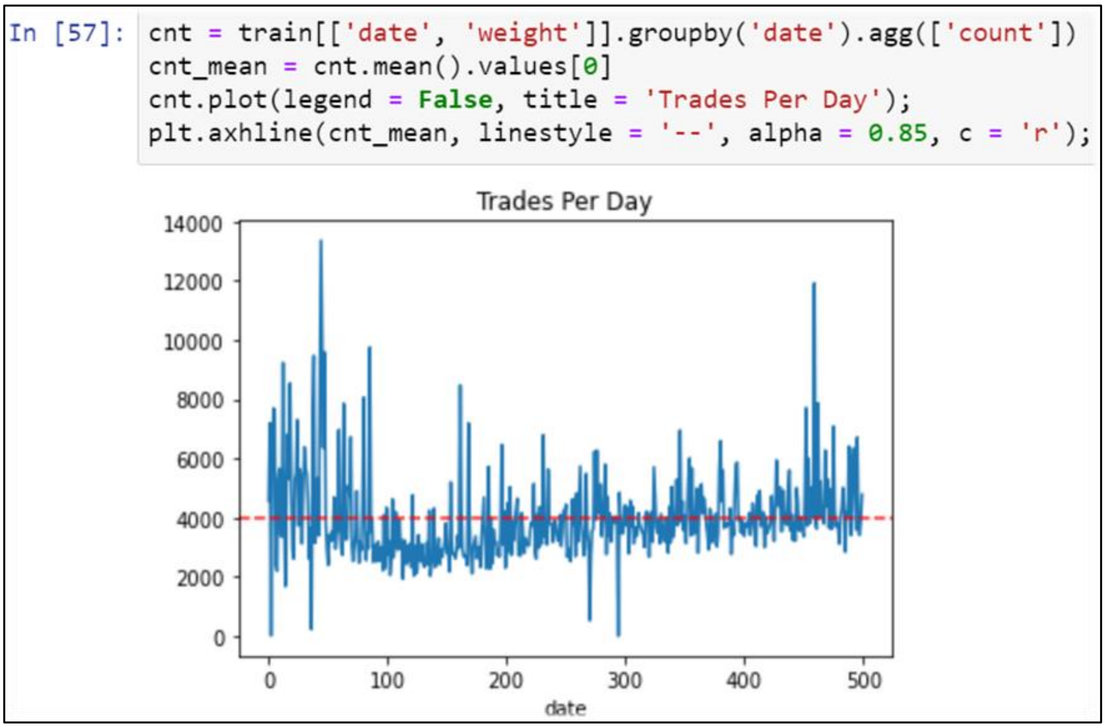
1.資料分析

(1)觀察 train.csv 有將近 5.8GB 的資料量，且有破百萬筆的歷史交易資料。



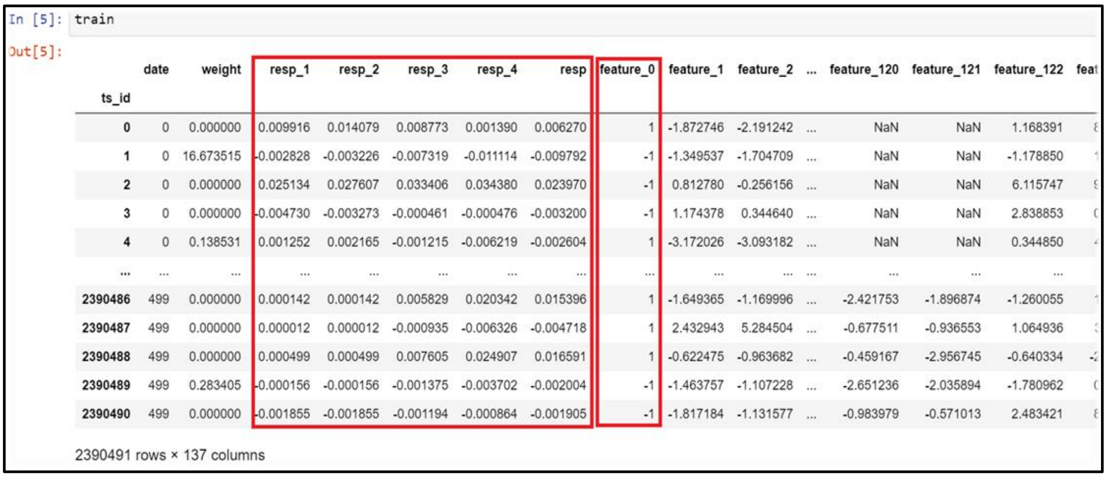
```
In [4]: print('train shape is {}'.format(train.shape))
        print('features shape is {}'.format(features.shape))
        print('example_test shape is {}'.format(example_test.shape))
        print('sample_prediction_df shape is {}'.format(example_sample_submission.shape))

        train shape is (2390491, 137)
        features shape is (130, 29)
        example_test shape is (15219, 132)
        sample_prediction_df shape is (15219, 1)
```

(2)每日當天的平均交易量。



```
In [57]: cnt = train[['date', 'weight']].groupby('date').agg(['count'])
         cnt_mean = cnt.mean().values[0]
         cnt.plot(legend = False, title = 'Trades Per Day');
         plt.axhline(cnt_mean, linestyle = '--', alpha = 0.85, c = 'r');
```

(3)資料中以 resp~resp_4 與 feature_0 當作我們觀測數據的指標。

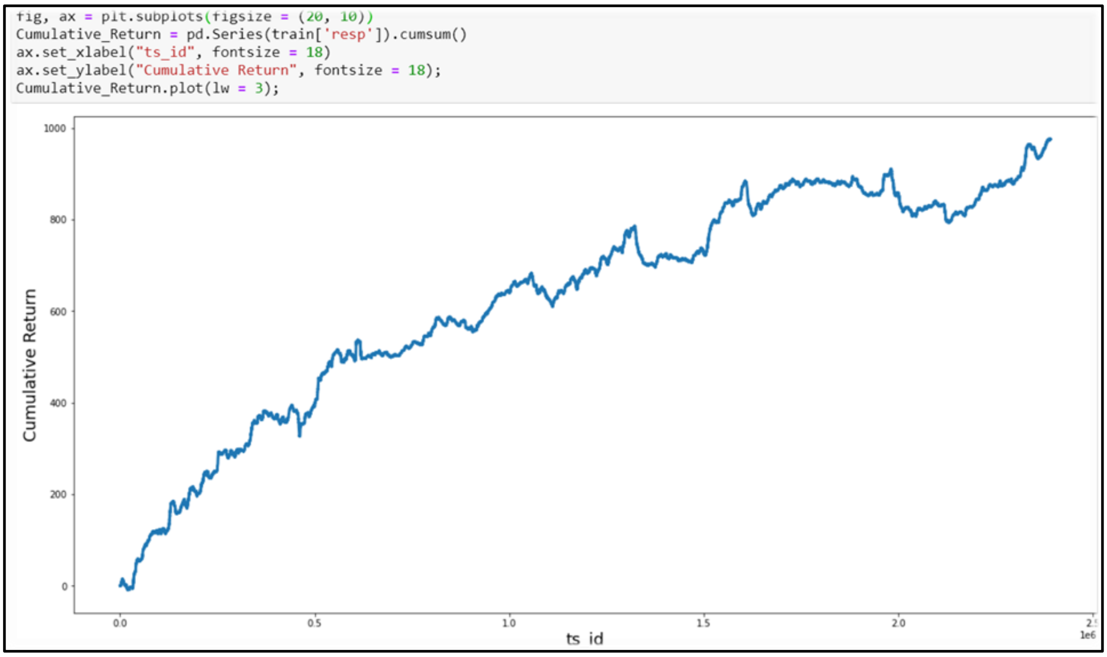


```
In [5]: train
Out[5]:
```

	date	weight	resp_1	resp_2	resp_3	resp_4	resp	feature_0	feature_1	feature_2	...	feature_120	feature_121	feature_122	feat
ts_id															
0	0	0.000000	0.009916	0.014079	0.008773	0.001390	0.006270	1	-1.872746	-2.191242	...	NaN	NaN	1.168391	8
1	0	16.673515	-0.002828	-0.003226	-0.007319	-0.011114	-0.009792	-1	-1.349537	-1.704709	...	NaN	NaN	-1.178850	1
2	0	0.000000	0.025134	0.027607	0.033406	0.034380	0.023970	-1	0.812780	-0.256156	...	NaN	NaN	6.115747	9
3	0	0.000000	-0.004730	-0.003273	-0.000461	-0.000476	-0.003200	-1	1.174378	0.344640	...	NaN	NaN	2.838853	0
4	0	0.138531	0.001252	0.002165	-0.001215	-0.006219	-0.002604	1	-3.172026	-3.093182	...	NaN	NaN	0.344850	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2390486	499	0.000000	0.000142	0.000142	0.005829	0.020342	0.015396	1	-1.649365	-1.169996	...	-2.421753	-1.896874	-1.260055	1
2390487	499	0.000000	0.000012	0.000012	-0.000935	-0.006326	-0.004718	1	2.432943	5.284504	...	-0.677511	-0.936553	1.064936	3
2390488	499	0.000000	0.000499	0.000499	0.007605	0.024907	0.016591	1	-0.622475	-0.963682	...	-0.459167	-2.956745	-0.640334	-2
2390489	499	0.283405	-0.000156	-0.000156	-0.001375	-0.003702	-0.002004	-1	-1.463757	-1.107228	...	-2.651236	-2.035894	-1.780962	0
2390490	499	0.000000	-0.001855	-0.001855	-0.001194	-0.000864	-0.001905	-1	-1.817184	-1.131577	...	-0.983979	-0.571013	2.483421	8

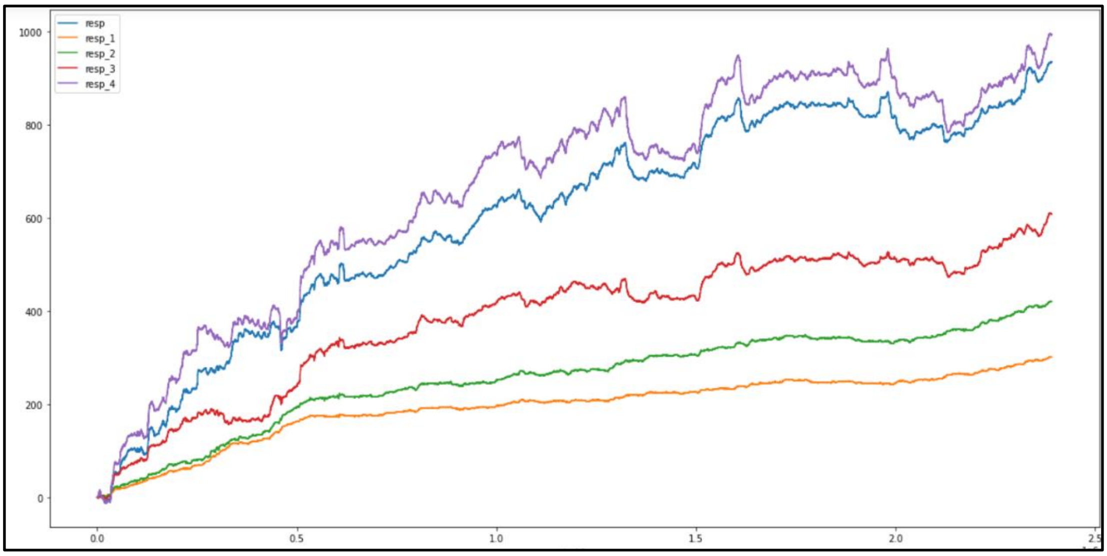
2390491 rows × 137 columns

(4) train.csv 中 resp 隨著時間的累積收益。

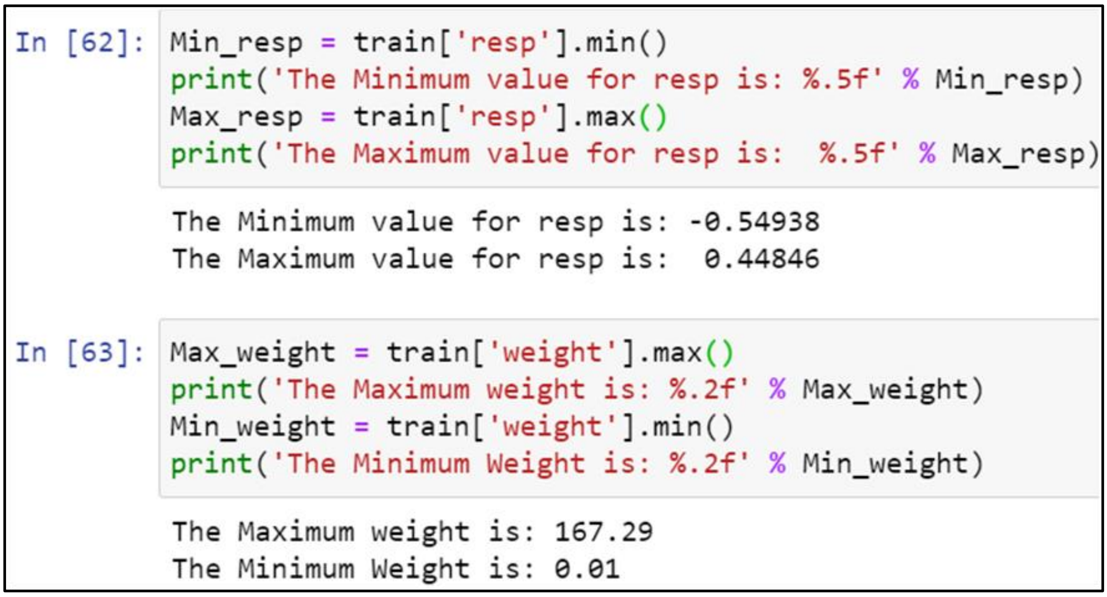


```
fig, ax = plt.subplots(figsize = (20, 10))
Cumulative_Return = pd.Series(train['resp']).cumsum()
ax.set_xlabel("ts_id", fontsize = 18)
ax.set_ylabel("Cumulative Return", fontsize = 18);
Cumulative_Return.plot(lw = 3);
```

(5)其中藍色 resp 與紫色 resp_4 的時間序列最接近。

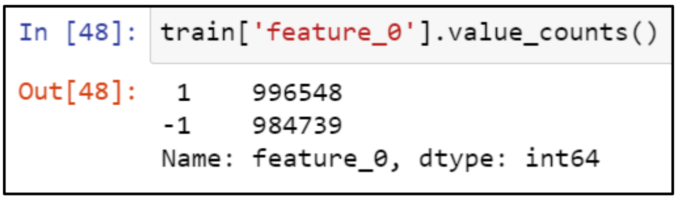


(6)每筆交易都有關聯的權重(weight)和回報(resp)，代表每次交易的收益。

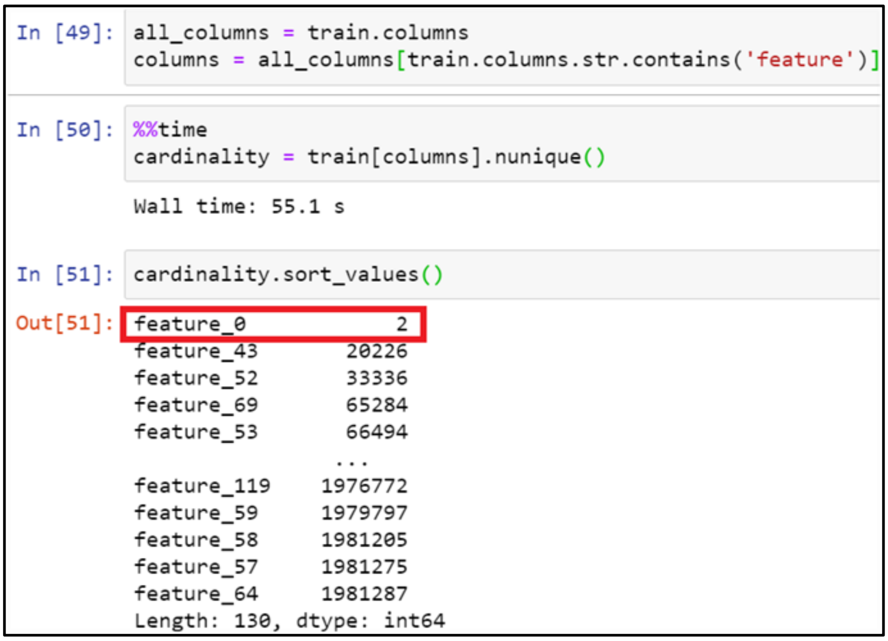


```
In [62]: Min_resp = train['resp'].min()
         print('The Minimum value for resp is: %.5f' % Min_resp)
         Max_resp = train['resp'].max()
         print('The Maximum value for resp is:  %.5f' % Max_resp)

         The Minimum value for resp is: -0.54938
         The Maximum value for resp is:  0.44846

In [63]: Max_weight = train['weight'].max()
         print('The Maximum weight is: %.2f' % Max_weight)
         Min_weight = train['weight'].min()
         print('The Minimum Weight is: %.2f' % Min_weight)

         The Maximum weight is: 167.29
         The Minimum Weight is: 0.01
```

(7) train.csv 中的 feature 除了 feature_0 明顯有分類外，其餘則沒有。

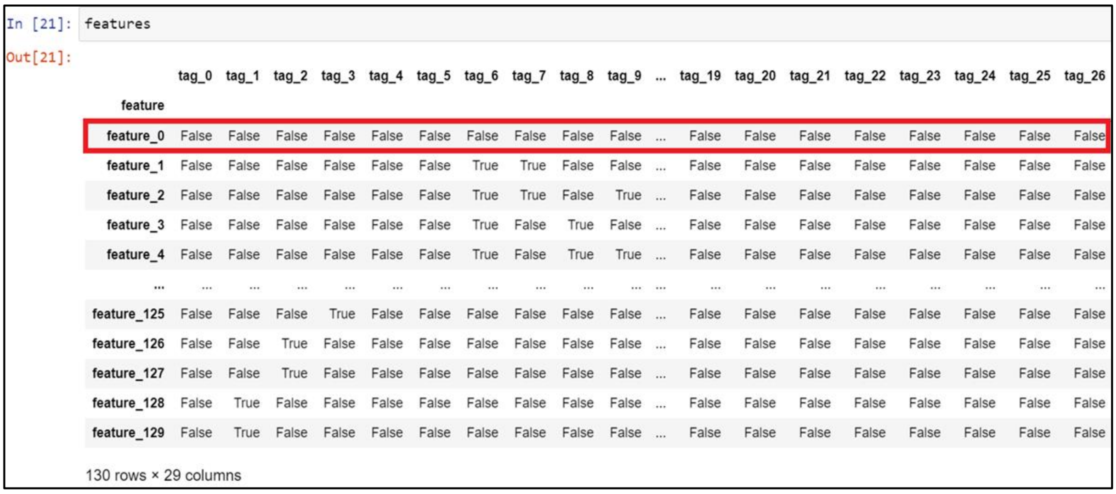


```
In [48]:  train['feature_0'].value_counts()

Out[48]:   1     996548
          -1     984739
          Name: feature_0, dtype: int64
```



```
In [49]: all_columns = train.columns
         columns = all_columns[train.columns.str.contains('feature')]

In [50]: %%time
         cardinality = train[columns].nunique()

         Wall time: 55.1 s

In [51]: cardinality.sort_values()

Out[51]: feature_0            2
         feature_43       20226
         feature_52       33336
         feature_69       65284
         feature_53       66494
                          ...
         feature_119    1976772
         feature_59     1979797
         feature_58     1981205
         feature_57     1981275
         feature_64     1981287
         Length: 130, dtype: int64
```

(8)feature_0 與其他特徵不同的點在於它是唯一沒有 True 的特徵。

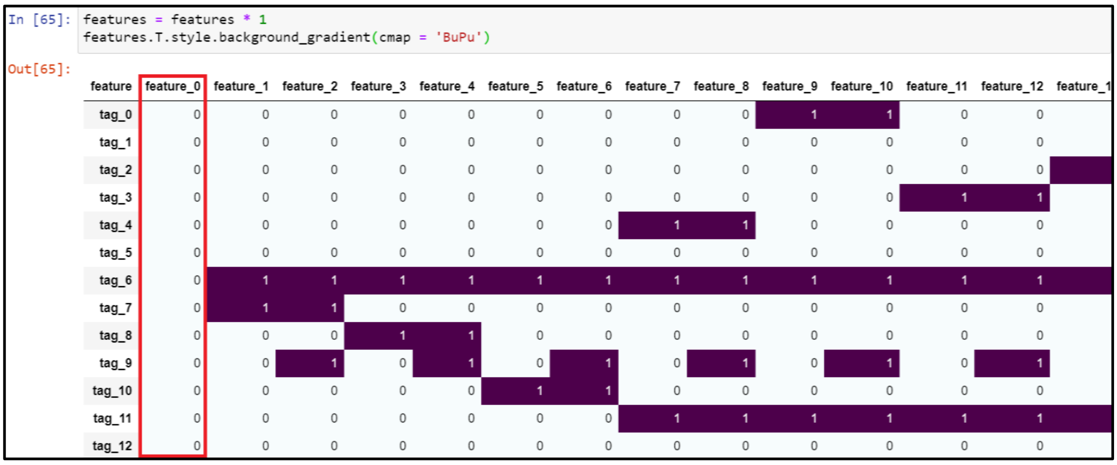


```
In [21]: features
Out[21]:
```

	tag_0	tag_1	tag_2	tag_3	tag_4	tag_5	tag_6	tag_7	tag_8	tag_9	...	tag_19	tag_20	tag_21	tag_22	tag_23	tag_24	tag_25	tag_26
feature																			
feature_0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
feature_1	False	False	False	False	False	False	True	True	False	False	...	False	False	False	False	False	False	False	False
feature_2	False	False	False	False	False	False	True	True	False	True	...	False	False	False	False	False	False	False	False
feature_3	False	False	False	False	False	False	True	False	True	False	...	False	False	False	False	False	False	False	False
feature_4	False	False	False	False	False	False	True	False	True	True	...	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
feature_125	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
feature_126	False	False	True	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
feature_127	False	False	True	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
feature_128	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
feature_129	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False

130 rows × 29 columns

(9) features.csv 中 True 為 1，False 為 0，29 個 Tag 關係著每筆 features。



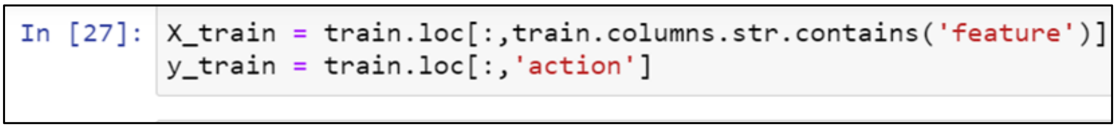
```
In [65]: features = features * 1
         features.T.style.background_gradient(cmap = 'BuPu')
```

Out[65]:

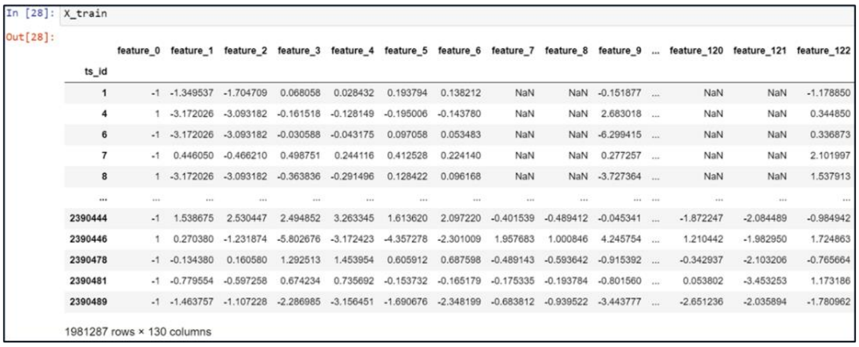
feature	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	feature_11	feature_12	feature_1
tag_0	0	0	0	0	0	0	0	0	0	1	1	0	0	
tag_1	0	0	0	0	0	0	0	0	0	0	0	0	0	
tag_2	0	0	0	0	0	0	0	0	0	0	0	0	0	
tag_3	0	0	0	0	0	0	0	0	0	0	0	1	1	
tag_4	0	0	0	0	0	0	0	1	1	0	0	0	0	
tag_5	0	0	0	0	0	0	0	0	0	0	0	0	0	
tag_6	0	1	1	1	1	1	1	1	1	1	1	1	1	
tag_7	0	1	1	0	0	0	0	0	0	0	0	0	0	
tag_8	0	0	0	1	1	0	0	0	0	0	0	0	0	
tag_9	0	0	1	0	1	0	1	0	1	0	1	0	1	
tag_10	0	0	0	0	0	1	1	0	0	0	0	0	0	
tag_11	0	0	0	0	0	0	0	1	1	1	1	1	1	
tag_12	0	0	0	0	0	0	0	0	0	0	0	0	0	

2.建立訓練模型

(1)將 train.csv 中的 feature 和 action 分別切割成 X_train 與 y_train。



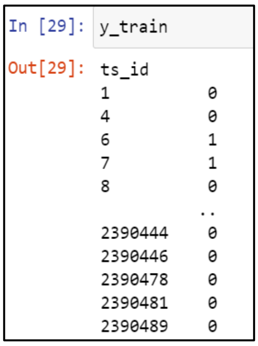
```
In [27]: X_train = train.loc[:,train.columns.str.contains('feature')]
         y_train = train.loc[:,'action']
```



```
In [28]: X_train
Out[28]:
```

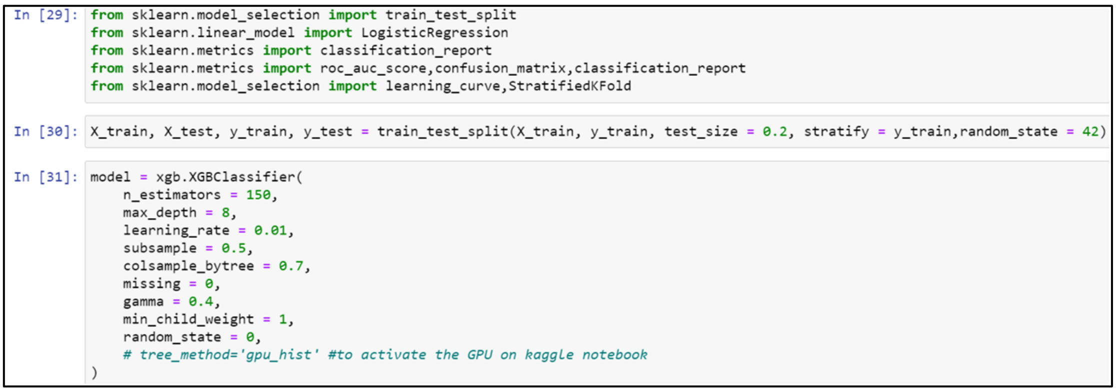
	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	...	feature_120	feature_121	feature_122
ts_id														
1	-1	-1.349537	-1.704709	0.068058	0.028432	0.193794	0.138212	NaN	NaN	-0.151877	...	NaN	NaN	-1.178850
4	1	-3.172026	-3.093182	-0.161518	-0.128149	-0.195006	-0.143780	NaN	NaN	2.683018	...	NaN	NaN	0.344850
6	-1	-3.172026	-3.093182	-0.030588	-0.043175	0.097058	0.053483	NaN	NaN	-6.299415	...	NaN	NaN	0.336873
7	-1	0.446050	-0.466210	0.498751	0.244116	0.412528	0.224140	NaN	NaN	0.277257	...	NaN	NaN	2.101997
8	1	-3.172026	-3.093182	-0.363836	-0.291496	0.128422	0.096168	NaN	NaN	-3.727364	...	NaN	NaN	1.537913
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2390444	-1	1.538675	2.530447	2.494852	3.263345	1.613620	2.097220	-0.401539	-0.489412	-0.045341	...	-1.872247	-2.084489	-0.984942
2390446	1	0.270380	-1.231874	-5.802676	-3.172423	-4.357278	-2.301009	1.957683	1.000846	4.245754	...	1.210442	-1.982950	1.724863
2390478	-1	-0.134380	0.160580	1.292513	1.453954	0.605912	0.687598	-0.489143	-0.593642	-0.915392	...	-0.342937	-2.103206	-0.765664
2390481	-1	-0.779554	-0.597258	0.674234	0.735692	-0.153732	-0.165179	-0.175335	-0.193784	-0.801560	...	0.053802	-3.453253	1.173186
2390489	-1	-1.463757	-1.107228	-2.286985	-3.156451	-1.690676	-2.348199	-0.683812	-0.939522	-3.443777	...	-2.651236	-2.035894	-1.780962

1981287 rows × 130 columns



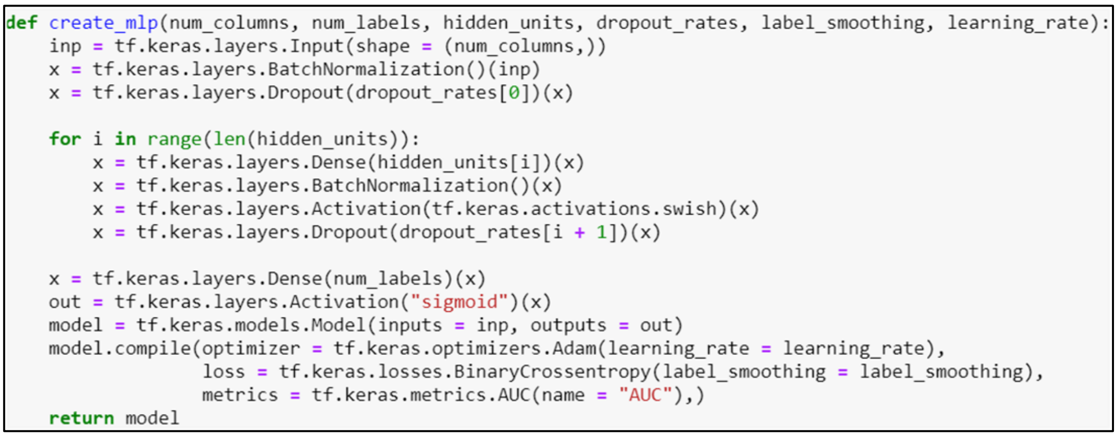
```
In [29]: y_train

Out[29]: ts_id
         1          0
         4          0
         6          1
         7          1
         8          0
                   ..
         2390444    0
         2390446    0
         2390478    0
         2390481    0
         2390489    0
```

(2)使用 XGBClassifier 模型對 X_train 和 y_train 進行訓練。



```
In [29]: from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import classification_report
         from sklearn.metrics import roc_auc_score,confusion_matrix,classification_report
         from sklearn.model_selection import learning_curve,StratifiedKFold

In [30]: X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size = 0.2, stratify = y_train,random_state = 42)

In [31]: model = xgb.XGBClassifier(
             n_estimators = 150,
             max_depth = 8,
             learning_rate = 0.01,
             subsample = 0.5,
             colsample_bytree = 0.7,
             missing = 0,
             gamma = 0.4,
             min_child_weight = 1,
             random_state = 0,
             # tree_method='gpu_hist' #to activate the GPU on kaggle notebook
         )
```

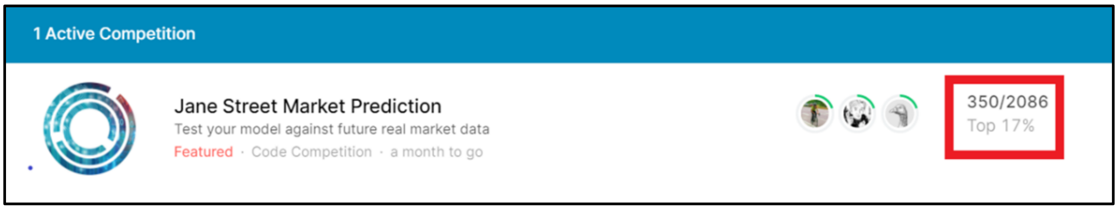
(3)使用 TensorFlow Neural Networks 模型進行預



```
def create_mlp(num_columns, num_labels, hidden_units, dropout_rates, label_smoothing, learning_rate):
    inp = tf.keras.layers.Input(shape = (num_columns,))
    x = tf.keras.layers.BatchNormalization()(inp)
    x = tf.keras.layers.Dropout(dropout_rates[0])(x)

    for i in range(len(hidden_units)):
        x = tf.keras.layers.Dense(hidden_units[i])(x)
        x = tf.keras.layers.BatchNormalization()(x)
        x = tf.keras.layers.Activation(tf.keras.activations.swish)(x)
        x = tf.keras.layers.Dropout(dropout_rates[i + 1])(x)

    x = tf.keras.layers.Dense(num_labels)(x)
    out = tf.keras.layers.Activation("sigmoid")(x)
    model = tf.keras.models.Model(inputs = inp, outputs = out)
    model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = learning_rate),
                  loss = tf.keras.losses.BinaryCrossentropy(label_smoothing = label_smoothing),
                  metrics = tf.keras.metrics.AUC(name = "AUC"),)
    return model
```

3.最佳分數

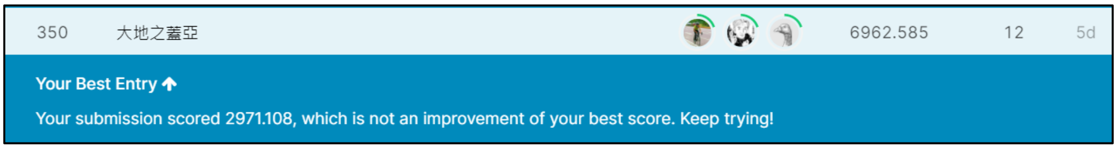
總共上傳 12 次，成功 4 次，失敗 8 次，由於每次上傳到 Kaggle 都需要 2-3 個小時，還不一定能成功，所以每次都要耗費時間做模型調整，光是上傳就耗費了多天。



1 Active Competition

Jane Street Market Prediction
Test your model against future real market data
Featured · Code Competition · a month to go

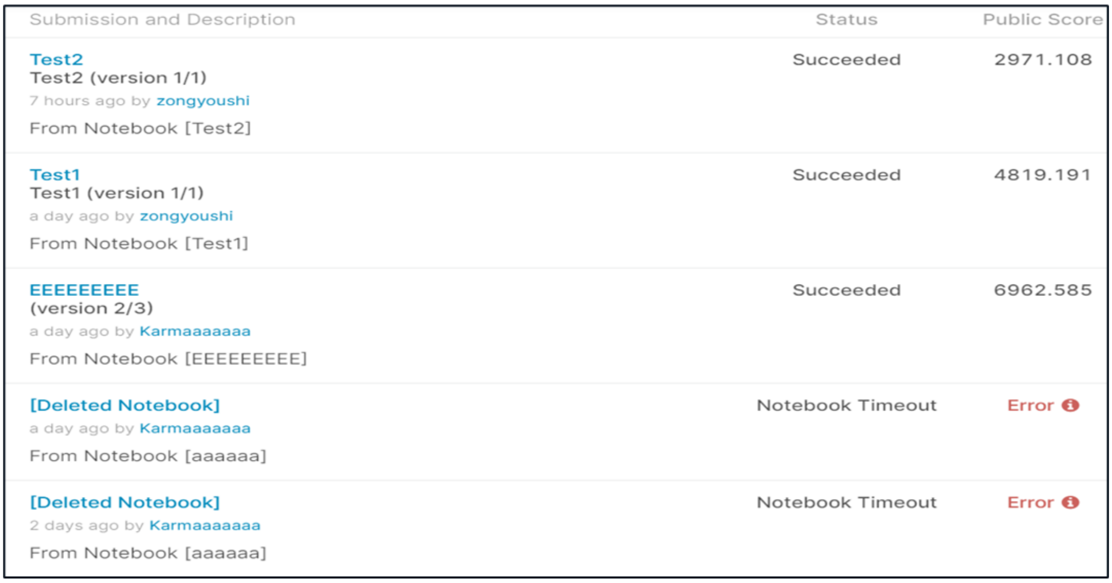
350/2086
Top 17%



350	大地之蓋亞	6962.585	12	5d

Your Best Entry ↑

Your submission scored 2971.108, which is not an improvement of your best score. Keep trying!



Submission and Description	Status	Public Score
Test2 Test2 (version 1/1) 7 hours ago by zongyoushi From Notebook [Test2]	Succeeded	2971.108
Test1 Test1 (version 1/1) a day ago by zongyoushi From Notebook [Test1]	Succeeded	4819.191
EEEEEEEEE (version 2/3) a day ago by Karmaaaaaaa From Notebook [EEEEEEEEE]	Succeeded	6962.585
[Deleted Notebook] a day ago by Karmaaaaaaa From Notebook [aaaaaa]	Notebook Timeout	Error
[Deleted Notebook] 2 days ago by Karmaaaaaaa From Notebook [aaaaaa]	Notebook Timeout	Error

結論

這次的期末專案，一開始我們是專注在預測模型的抉擇，比賽中公開的 Notebook 給了我們很多選擇，最後選取了 XGBoost 和 Keras，主要是課程中有學到，也比較好上手，兩種方法都有各自的差異，最明顯的就是速度，也許是我們的參數設定各有不同，直接影響最後的預測跑分，而一開始我們的方向就是把分數衝高，最後才做資料的特徵分析，最終分數也讓我們覺得不錯，有前 20%的名次，過程中看到各國的高手做出的預測模型，不同的思維有著多種的寫法，值得我們學習。

組員分工

	B10756026 林峻儀	B10756038 施宗佑	B1075640 郭家偉
Coding	✓	✓	✓
資料蒐集	✓	✓	✓
Word		✓	
PPT		✓	

Github

https://github.com/zongyoushi/NPUST_ML_Final_Project_Jane_Street_Market_Prediction_20210111.git

Kaggle

https://www.kaggle.com/c/jane-street-market-prediction