Research on the Dataset for The "Jane Street Market Data Forecasting" Competition

Xie Zuoyu & Li Sinuan

June 2025

Abstract

This report presents our research on the dataset from the "Jane Street Real-Time Market Data Forecasting" competition, hosted by Jane Street Capital on Kaggle. The competition focuses on building models that can predict whether a given transaction should be executed based on the high-dimensional real-time market features.

Although we joined this project after the official registration had closed and were unable to submit predictions, we obtained the complete dataset from the Kaggle community for independent experimentation.

The dataset contains over 60 million rows of historical trading opportunities, with 79 anonymized numerical features (denoted as `feature_00` to `feature_78`) and a key response variable `responder_6`, which represents the weighted return of the transaction. Each data point is also identified by `symbol_id` and `time_id`, providing a structure that supports time-series modeling.

To evaluate various modeling strategies for trade direction prediction, we implemented and compared four supervised learning models: Logistic Regression, LightGBM, XGBoost, and Multi-Layer Perceptron (MLP). Each model was trained using a carefully controlled pipeline that includes missing value handling, feature selection (via regularization and importance ranking), and evaluation through metrics such as AUC and directional accuracy.

Our primary goal is to assess each model's ability to distinguish profitable trade signals from neutral or loss-making trades, thereby simulating realistic financial forecasting under market data constraints.

Contents

1 Initial Data Handling and Partitioning

1.1 Data Partitioning Strategy

The original dataset is stored in several folders named from `partition_id=0` to `partition_id=9`. According to our modeling needs, we reorganized these partitions into five subsets:

- **Training set:** `partition_id=0--4`, used for model fitting;
- **Validation set:** `partition_id=5`, used for hyperparameter tuning and early stopping;
- **Test set:** `partition_id=6--7`, used for final model evaluation;
- **Future prediction set:** `partition_id=8`, used as the actual input for model prediction (labels are removed);
- **Future evaluation set:** `partition_id=9`, contains the true `responder_6` labels along with metadata for visualization and evaluation purposes.

1.2 Feature Extraction and Label Construction

From each partition, we extracted 79 anonymized numerical features, named from `feature_00` to `feature_78`. The target variable was constructed as follows:

```
If responder_6 > 0, the label is 1; otherwise, it is 0
```

1.3 Future Data Handling

To prevent data leakage, we removed all columns starting with `responder_` when processing `partition_id=8`. Meanwhile, `partition_id=9` retains only the columns `responder_6`, `symbol_id`, and `time_id`, which are used for alignment and visualization.

1.4 Output File Summary

After the above processing, we saved each dataset as a separate `.pkl` file using `joblib` to facilitate direct loading for downstream modeling. These files include:

- `Xy_train_raw.pkl` – features and labels for training;
- `Xy_val_raw.pkl` – features and labels for validation;

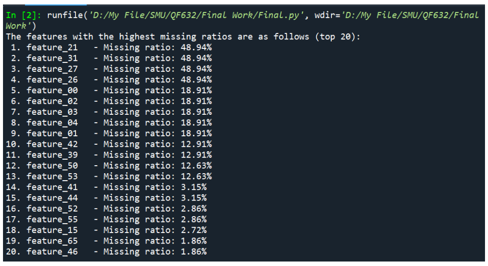
- `Xy_test_raw.pkl` – features and labels for testing;
- `X_future_raw.pkl` – features for future prediction (no labels);
- `df_future_eval_raw.pkl` – future evaluation set with labels and metadata.

This data structure ensures the reproducibility of training, validation, and prediction, while maintaining strict independence between input features and target labels across time.

2 Logistic Regression

2.1 Advanced Cleaning for Logistic Regression

We conducted further preprocessing specifically for the Logistic Regression (LR) model, focusing on removing unnecessary features. The entire cleaned dataset was saved into new `.pkl` files (only training-related files are described below).



```
In [2]: runfile('D:/My File/SMU/QF632/Final Work/Final.py', wdir='D:/My File/SMU/QF632/Final
Work')
The features with the highest missing ratios are as follows (top 20):
 1. feature_21   - Missing ratio: 48.94%
 2. feature_31   - Missing ratio: 48.94%
 3. feature_27   - Missing ratio: 48.94%
 4. feature_26   - Missing ratio: 48.94%
 5. feature_00   - Missing ratio: 18.91%
 6. feature_02   - Missing ratio: 18.91%
 7. feature_03   - Missing ratio: 18.91%
 8. feature_04   - Missing ratio: 18.91%
 9. feature_01   - Missing ratio: 18.91%
10. feature_42   - Missing ratio: 12.91%
11. feature_39   - Missing ratio: 12.91%
12. feature_50   - Missing ratio: 12.63%
13. feature_53   - Missing ratio: 12.63%
14. feature_41   - Missing ratio: 3.15%
15. feature_44   - Missing ratio: 3.15%
16. feature_52   - Missing ratio: 2.86%
17. feature_55   - Missing ratio: 2.86%
18. feature_15   - Missing ratio: 2.72%
19. feature_65   - Missing ratio: 1.86%
20. feature_46   - Missing ratio: 1.86%
```

Figure 1: Top 20 features with the highest missing ratios

Features `21`, `31`, `27`, and `26` had excessively high missing ratios and were directly removed. Then, we filled missing values using the **mean imputation** strategy and applied L1 regularization to further select meaningful features.

```
In [1]: runfile('D:/My File/SMU/QF632/Final Work/Final.py', wdir='D:/My File/SMU/QF632/Final
Work')
The number of features retained after L1 regularization: 75
 1: feature_00
 2: feature_01
 3: feature_02
 4: feature_03
 5: feature_04
 6: feature_05
 7: feature_06
 8: feature_07
 9: feature_08
10: feature_09
11: feature_10
12: feature_11
13: feature_12
14: feature_13
15: feature_14
16: feature_15
17: feature_16
18: feature_17
19: feature_18
20: feature_19
conda: base (Python 3.12.7)  Completions: conda(base)  ✓ LSP: Pyth  col 4  UTF-8  CRLF  RW
```

Figure 2: Number of features retained after L1 regularization

`The number of features retained after L1 regularization: 75` Even under strong penalty (C=0.01), L1 regularization did not eliminate any features. Therefore, we finalized the cleaned dataset for the LR model and proceeded with model training.

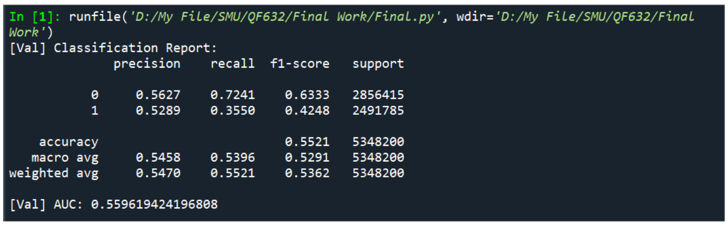
2.2 Modeling and Results

2.2.1 Model Evaluation

We trained a standard Logistic Regression model using default L2 regularization. The model was fitted on the training set and evaluated on the validation and test sets. The configuration is as follows:

- Model type: `LogisticRegression(solver='lbfgs', max_iter=1000)`
- Label definition: if `responder_6 > 0`, the label is 1; otherwise, it is 0
- Training data: corresponds to `partition_id=0--4` in the original dataset

The performance of the model on the validation set is shown below.



```
In [1]: runfile('D:/My File/SMU/QF632/Final Work/Final.py', wdir='D:/My File/SMU/QF632/Final Work')
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5627    0.7241    0.6333   2856415
           1     0.5289    0.3550    0.4248   2491785

    accuracy                         0.5521   5348200
   macro avg     0.5458    0.5396    0.5291   5348200
weighted avg     0.5470    0.5521    0.5362   5348200

[Val] AUC: 0.559619424196808
```

Figure 3: Classification report on validation set (Precision, Recall, F1)

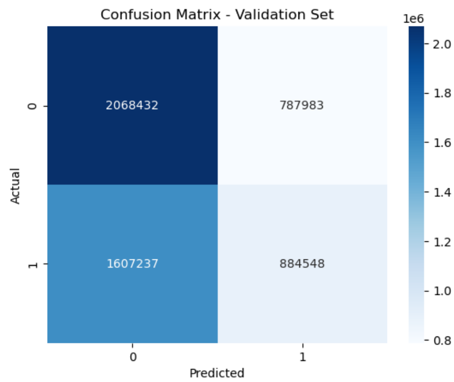


Figure 4: Confusion matrix on validation set

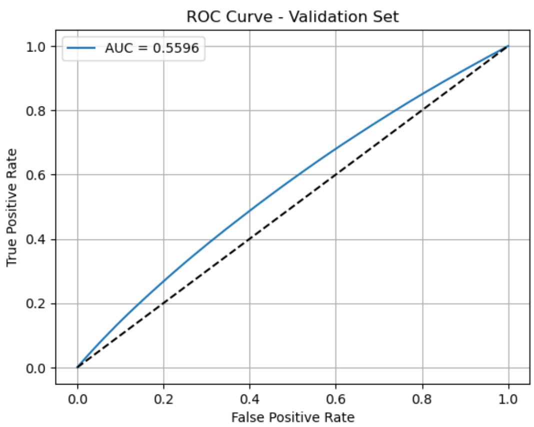


Figure 5: ROC curve on validation set

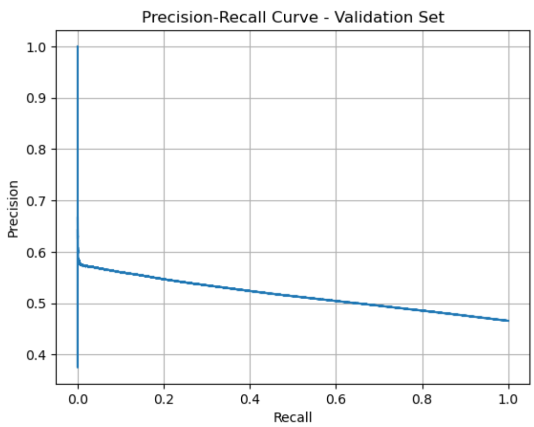


Figure 6: Precision-Recall curve on validation set

The evaluation result on the test set is as follows:

```
[TEST] AUC: 0.5511124048409815
```

2.2.2 Prediction and Visualization on Future Samples

Since we were unable to officially participate in the competition and thus could not access real-time data, we treated `partition_id=9` as our "future" evaluation dataset. To prevent data leakage, we removed all features from this dataset except for `responder_6`, `symbol_id`, and `time_id`.

We applied the trained model to `X_future_ready.pkl` and visualized the prediction results against the true labels provided in `df_future_eval_ready.pkl`. The predicted probabilities were compared with the actual trade direction (defined by `responder_6 > 0`). The resulting visualization is shown below. Since the full chart would be too long due to the volume of data, we present a cropped segment here for illustration purposes.

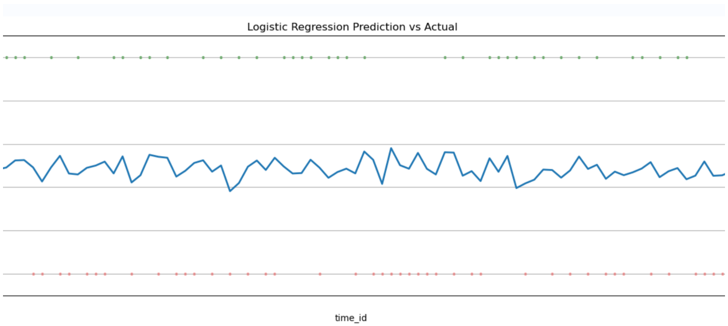


Figure 7: Logistic Regression prediction vs "Future" Data

Explanation of the plot:

- Blue solid line: predicted probability
- Green dots: true direction is positive (`responder_6 > 0`)
- Red dots: true direction is zero or negative

This visualization helps examine whether the prediction trend aligns with the actual labels over time and whether overfitting or signal drift may be present.

Overall, although Logistic Regression is a linear model, it performs robustly in a high-dimensional and large-scale dataset, which is quite impressive. However, there remains substantial room for improvement in terms of AUC, overall accuracy, and especially the recall for class 1. Therefore, we proceed to explore more complex models in the following sections to potentially enhance predictive power and recall quality.

3 LightGBM

3.1 Data Processing

First, following the same approach as in Section 2.1, we removed four features with very high proportions of missing values.

Subsequently, leveraging LightGBM's inherent capabilities, we experimented with different missing value handling methods and feature engineering strategies. Since this model has a built-in mechanism for handling missing values (`use_missing=True`, `zero_as_missing=False`), which can distinguish true zeros from missing values and process them appropriately, training directly on the data without imputation is feasible.

The model also inherently calculates feature importance, enabling Recursive Feature Elimination (RFE). Ultimately, we selected features whose importance was equal to or greater than the median of overall importance of the feature, reducing the set of characteristics to 38 of the original 75 characteristics.

```
Based on median(573.00)Number of features retained after filtering: 38
Examples of reserved features: ['feature_00' 'feature_01' 'feature_02' 'feature_03' 'feature_04'
 'feature_05' 'feature_07' 'feature_08' 'feature_14' 'feature_15'
 'feature_17' 'feature_19' 'feature_20' 'feature_22' 'feature_23'
 'feature_24' 'feature_25' 'feature_28' 'feature_29' 'feature_30'
 'feature_36' 'feature_39' 'feature_47' 'feature_49' 'feature_50'
 'feature_51' 'feature_52' 'feature_53' 'feature_54' 'feature_55'
 'feature_58' 'feature_59' 'feature_60' 'feature_66' 'feature_68'
 'feature_69' 'feature_71' 'feature_72']
```

Figure 8: 38 features with with a importance level higher than the overall median

Although imputing missing values is not strictly necessary for LightGBM, we still employed the Multiple Imputation by Chained Equations (MICE) method to fill missing values. This was done in consideration of the subsequent Multi-Layer Perceptron (MLP) model, which cannot inherently handle missing data. We trained the MICE imputer using a 20% random sample of the training data and then applied it to impute missing values in the entire training set as well as other sets. This allowed us to explore whether this specific imputation approach could improve model performance.

```
# === MICE ===
# sample shape: (3365013, 75)
# using: HistGradientBoostingRegressor
#
# dataset: train
# process chunking: 100%|██████████| 10/10 [02:42<00:00, 16.25s/it]
#
# dataset: val
# process chunking: 100%|██████████| 10/10 [00:29<00:00,  2.95s/it]
#
# dataset: test
# process chunking: 100%|██████████| 10/10 [01:11<00:00,  7.14s/it]
#
# dataset: future
# process chunking: 100%|██████████| 10/10 [00:32<00:00,  3.21s/it]
#
# === time: 28 minute 27.74 sec ===
# save:
# - train: X_train_mice.pkl
# - val: X_val_mice.pkl
# - test: X_test_mice.pkl
# - future: X_future_mice.pkl
```

Figure 9: MICE algorithm processing display

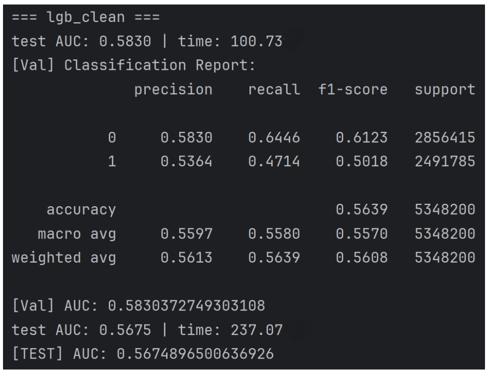
3.2 Modeling and Results

3.2.1 Model Evaluation

We use `lgb.train` for training, fitting the model to the training set, and evaluating it in the validation and test sets. Key hyperparameters included:

- Tree structure: 511 leaves with unlimited depth (`max_depth=-1`) and minimum 500 samples per leaf
- Regularization: L1/L2 penalties (0.05), feature/bagging fractions (0.7)
- Training: 0.02 learning rate with 1000 boosting rounds
- GPU optimization: 255 max bins for histogram acceleration

The performance of the model on the validation set is shown below.



```
=== lgb_clean ===
test AUC: 0.5830 | time: 100.73
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5830    0.6446    0.6123   2856415
           1     0.5364    0.4714    0.5018   2491785

    accuracy                         0.5639   5348200
   macro avg     0.5597    0.5580    0.5570   5348200
weighted avg     0.5613    0.5639    0.5608   5348200

[Val] AUC: 0.5830372749303108
test AUC: 0.5675 | time: 237.07
[TEST] AUC: 0.5674896500636926
```

Figure 10: Classification report on validation set
(Precision, Recall, F1)

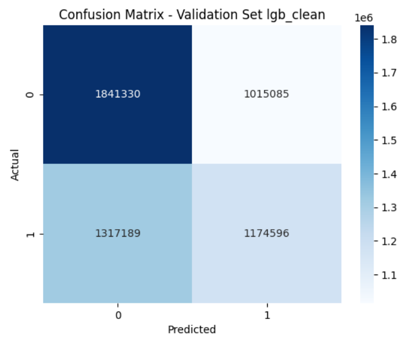


Figure 11: Confusion matrix on validation set

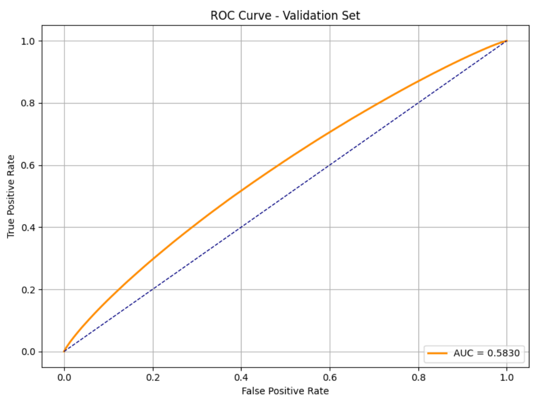


Figure 12: ROC curve on validation set

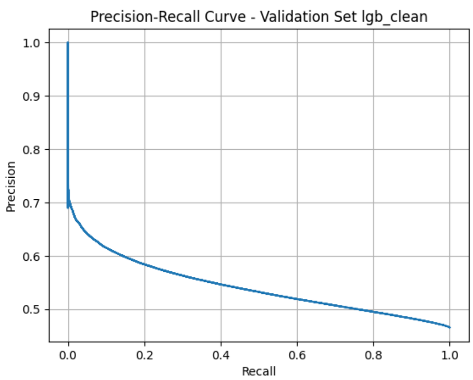


Figure 13: Precision-Recall curve on validation set

We also experimented with two alternative approaches:

- Method 2: Training the model using the filtered feature subset
- Method 3: A combined approach using:
  - The filtered feature subset
  - MICE imputation for missing values
  - Z-score normalization with extreme values replaced by medians

Comparative results showed no significant improvement over the original dataset performance.

```
=== lgb_filtered ===
test AUC: 0.5766 | time: 94.32
[Val] Classification Report:
              precision    recall  f1-score   support
|
           0     0.5785    0.6467    0.6107   2856415
           1     0.5318    0.4600    0.4933   2491785

    accuracy                         0.5597   5348200
   macro avg     0.5551    0.5533    0.5520   5348200
weighted avg     0.5567    0.5597    0.5560   5348200

[Val] AUC: 0.5765979914375632
test AUC: 0.5616 | time: 214.41
[TEST] AUC: 0.5615735208682778
```

Figure 14: Classification report of Method 2
(Precision, Recall, F1)

```
=== lgb_3pre ===
test AUC: 0.5743 | time: 90.90
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5768    0.6482    0.6104   2856415
           1     0.5301    0.4549    0.4896   2491785

    accuracy                         0.5581   5348200
   macro avg     0.5534    0.5515    0.5500   5348200
weighted avg     0.5550    0.5581    0.5541   5348200

[Val] AUC: 0.5743418126637555
test AUC: 0.5593 | time: 207.60
[TEST] AUC: 0.5592606564786361
```

Figure 15: Classification report of Method 3
(Precision, Recall, F1)

3.2.2 Prediction and Visualization on Future Samples

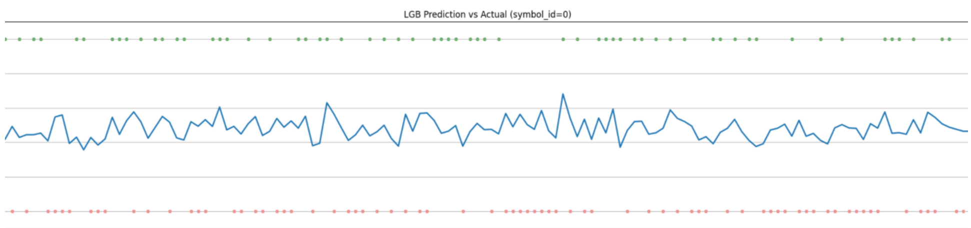


Figure 16: LightGBM prediction vs "Future" Data

The LightGBM model's performance constraints may primarily reflect inherent limitations of tree-based approaches, decision trees split features orthogonally, potentially missing complex multidimensional relationships present in the data, also piecewise constant outputs may poorly approximate smooth underlying functions.

4 XGBoost

4.1 Data Cleaning for XGBoost

We trained an XGBoost model using `XGBClassifier` and specified `logloss` as the evaluation metric. The importance score of each feature was extracted to quantify its contribution to the model's predictions. All features were sorted by importance in descending order, and those with zero contribution were removed.
Interestingly, all 75 features were found to have non-zero importance and could be retained. Based on score thresholds and empirical evaluation, we decided to explore the following three configurations:

- **Top 28 features** with importance > 0.01 (strongly recommended);
- **Top 65 features** with importance $\geq$ 0.006 (recommended);
- **All 75 features** retained (optional).

```
  bst.update(dtrain, iteration=i, fobj=obj)
Top 30 most important features (by XGBoost):
 1. feature_06   - importance: 0.1102
 2. feature_60   - importance: 0.0682
 3. feature_59   - importance: 0.0605
 4. feature_07   - importance: 0.0464
 5. feature_54   - importance: 0.0325
 6. feature_51   - importance: 0.0289
 7. feature_68   - importance: 0.0240
 8. feature_04   - importance: 0.0206
 9. feature_30   - importance: 0.0189
10. feature_55   - importance: 0.0177
11. feature_75   - importance: 0.0171
12. feature_52   - importance: 0.0150
13. feature_15   - importance: 0.0146
14. feature_71   - importance: 0.0143
15. feature_69   - importance: 0.0140
16. feature_14   - importance: 0.0128
17. feature_50   - importance: 0.0127
18. feature_01   - importance: 0.0124
19. feature_02   - importance: 0.0123
20. feature_53   - importance: 0.0123
21. feature_56   - importance: 0.0123
22. feature_58   - importance: 0.0116
23. feature_48   - importance: 0.0114
24. feature_36   - importance: 0.0107
25. feature_10   - importance: 0.0107
26. feature_23   - importance: 0.0106
27. feature_76   - importance: 0.0100
28. feature_19   - importance: 0.0100
29. feature_18   - importance: 0.0099
30. feature_29   - importance: 0.0097
```

```
31. feature_05   - importance: 0.0095
32. feature_17   - importance: 0.0094
33. feature_47   - importance: 0.0093
34. feature_66   - importance: 0.0092
35. feature_09   - importance: 0.0091
36. feature_00   - importance: 0.0091
37. feature_72   - importance: 0.0090
38. feature_22   - importance: 0.0089
39. feature_11   - importance: 0.0088
40. feature_20   - importance: 0.0086
41. feature_73   - importance: 0.0085
42. feature_61   - importance: 0.0084
43. feature_28   - importance: 0.0082
44. feature_08   - importance: 0.0081
45. feature_40   - importance: 0.0080
46. feature_13   - importance: 0.0079
47. feature_44   - importance: 0.0079
48. feature_24   - importance: 0.0079
49. feature_65   - importance: 0.0077
50. feature_67   - importance: 0.0075
51. feature_37   - importance: 0.0074
52. feature_03   - importance: 0.0073
53. feature_49   - importance: 0.0073
54. feature_38   - importance: 0.0072
55. feature_25   - importance: 0.0070
56. feature_39   - importance: 0.0070
57. feature_78   - importance: 0.0070
58. feature_70   - importance: 0.0068
59. feature_46   - importance: 0.0068
60. feature_12   - importance: 0.0067
61. feature_41   - importance: 0.0067
62. feature_62   - importance: 0.0066
63. feature_57   - importance: 0.0065
64. feature_42   - importance: 0.0062
65. feature_45   - importance: 0.0061
66. feature_33   - importance: 0.0059
67. feature_74   - importance: 0.0059
68. feature_32   - importance: 0.0058
69. feature_35   - importance: 0.0058
70. feature_34   - importance: 0.0057
71. feature_43   - importance: 0.0057
72. feature_64   - importance: 0.0055
73. feature_77   - importance: 0.0055
74. feature_63   - importance: 0.0047
75. feature_16   - importance: 0.0036
```

Figure 17: Each feature's importance score in the XGBoost model

4.2 XGBoost with All 75 Features

```
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5783    0.6479    0.6111   2856415
           1     0.5318    0.4584    0.4924   2491785

    accuracy                         0.5596   5348200
   macro avg     0.5550    0.5531    0.5517   5348200
weighted avg     0.5566    0.5596    0.5558   5348200

[Val] AUC: 0.5760809452025897
[TEST] AUC: 0.561606945142231
```

Figure 18: XGBoost on 75 features

Compared to the linear model, this configuration offers a noticeable improvement. We proceed to evaluate whether reducing the number of features would further benefit performance.

4.3 XGBoost with 65 Features

Only features with importance ≥ 0.006 were retained, resulting in 65 features.

```
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5785    0.6472    0.6109   2856415
           1     0.5318    0.4594    0.4930   2491785

    accuracy                         0.5597   5348200
   macro avg     0.5551    0.5533    0.5519   5348200
weighted avg     0.5567    0.5597    0.5559   5348200

[Val] AUC: 0.5761781268468457
[TEST] AUC: 0.5624955605090792
```

Figure 19: XGBoost on 65 features

4.4 XGBoost with 28 Features

We further tested the scenario of retaining only the most critical features with importance > 0.01 (top 28 features).

```
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5779    0.6497    0.6117   2856415
           1     0.5317    0.4560    0.4910   2491785

    accuracy                         0.5594   5348200
   macro avg     0.5548    0.5528    0.5513   5348200
weighted avg     0.5564    0.5594    0.5554   5348200

[Val] AUC: 0.5756430997223801
[TEST] AUC: 0.5623682031790292
```

Figure 20: XGBoost on 28 features

4.5 Performance Comparison with Different Feature Counts

To assess the effect of feature dimensionality on model performance, we trained XGBoost classifiers with three different levels of feature selection: 75 features (all retained), 65 features (importance score > 0.006), and 28 features (top features only). The results are summarized in Table 1.

Table 1: XGBoost Performance Comparison across Different Feature Sets

Feature Count	Val AUC	Test AUC	Accuracy	Class 1 Recall
75 features	0.5761	0.5616	0.5596	**0.4584**
65 features	**0.5762**	**0.5625**	**0.5597**	0.4594
28 features	0.5756	0.5624	0.5594	0.4560

From the table above, we make the following observations:

- **AUC:** The model with 65 features achieved the highest AUC on both validation and test sets, slightly outperforming the full 75-feature model.
- **Accuracy:** The 65-feature version also achieved the highest classification accuracy.
- **Class 1 Recall:** The 75-feature model retained the highest recall for profitable signals (`responder_6 > 0`), although the differences are marginal.

4.6 Comparison with Logistic Regression

To assess the effectiveness of XGBoost over traditional linear models, we compared its performance with the previously trained Logistic Regression (LR) model. The summary is provided in Table 2.

Table 2: XGBoost vs Logistic Regression Performance (Validation Set)

Model	Features	Val AUC	Accuracy	Class 1 Recall
Logistic Regression	75	0.5596	0.5521	0.3550
XGBoost	75	0.5761	0.5596	0.4584
XGBoost	65	**0.5762**	**0.5597**	**0.4594**
XGBoost	28	0.5756	0.5594	0.4560

Key Observations:

- **AUC:** All XGBoost variants outperform LR in AUC by approximately 1.6–1.7%, demonstrating stronger discriminative power.

- **Accuracy:** XGBoost yields higher accuracy overall, especially with the 65-feature setting.
- **Class 1 Recall:** Recall for profitable signals (`responder_6 > 0`) is significantly improved by XGBoost (from 0.3550 to 0.4594), which is crucial for identifying positive trading opportunities.

Why is the improvement still limited?

- **Feature-linearity:** Many of the 79 anonymized features may relate linearly to the outcome, which limits XGBoost's nonlinear advantage.
- **No advanced feature engineering:** We did not include interactions, polynomial terms, or derived features that could unlock additional predictive power.
- **Data noise:** In financial forecasting, label noise and randomness in asset prices often cap the model's performance.
- **Ceiling effect:** The AUC nearing 0.58 may already approach the practical upper bound given current inputs.

In conclusion, while XGBoost provides measurable gains over LR, especially in detecting class 1 signals, the improvement is moderate. To achieve greater performance boosts, future work should explore richer feature construction, alternative model architectures, or ensemble methods.

4.7 Trade-off Between Performance and Model Complexity

Across the three XGBoost models, reducing the number of features from 75 to 65:

- Made the model more interpretable and computationally efficient;
- Slightly improved AUC and accuracy on both validation and test sets.

However, further reducing to only 28 features led to a small drop in AUC and recall, indicating that several of the excluded features still carried predictive value. When compared to the Logistic Regression (LR) model:

- XGBoost (with any feature count) outperformed LR in terms of AUC, accuracy, and precision for both classes;
- The performance gain, while consistent, was not substantial — suggesting that either the features are only weakly predictive or the binary classification task is inherently difficult given the available information.

Conclusion: The XGBoost model with **65 features** achieves the best balance between predictive power and simplicity. It slightly outperforms both the full-featured XGBoost model and the baseline LR model. We recommend this configuration for downstream deployment or further experimentation.

4.8 Prediction and Visualization on Future Samples

We take the prediction maps of the same area from three XGBoost models for comparison.

Figure 21: XGBoost prediction vs "Future"Datan on using 75, 65, and 28 features

Conclusion: The 65-feature version provides the most visually aligned signal-to-noise balance, demonstrating clearer agreement with true market direction compared to the other two settings. This aligns with our earlier metrics-based analysis that favored the 65-feature configuration as the best trade-off between complexity and accuracy.

5 Multi-Layer Perceptron (MLP)

5.1 Data Processing

Since the Multi-Layer Perceptron (MLP) cannot inherently handle missing values, we preprocessed the data using MICE imputation as described in Section 3.1. Unlike LightGBM, MLP requires mandatory feature standardization, which we strictly implemented.

To enhance model performance, we selected the top 10 most important features from the 38 previously identified in Section 3.1 and created interaction features for the training set. The results were then compared with baseline performance.

	Feature	Importance	Selected	Ranking
4	feature_04	2234	True	1
1	feature_01	1833	True	1
15	feature_15	1410	True	1
55	feature_59	1388	True	1
5	feature_05	1349	True	1
48	feature_52	1289	True	1
51	feature_55	1281	True	1
54	feature_58	1227	True	1
50	feature_54	1084	True	1
49	feature_53	1069	True	1

Figure 22: Top ten most important features from LightGBM

5.2 Baseline MLP model

The implemented Multi-Layer Perceptron (MLP) consists of:

- **Architecture**: 4 fully-connected layers (256-128-64-1 neurons) with ReLU activation
- **Training**: Optimized using Adam ($lr = 10^{-3}$) with BCEWithLogitsLoss, Batch size of 4096 over 30 epochs

```
test AUC: 0.5544
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5664    0.6276    0.5955   2856415
           1     0.5128    0.4493    0.4789   2491785

    accuracy                         0.5445   5348200
   macro avg     0.5396    0.5384    0.5372   5348200
weighted avg     0.5414    0.5445    0.5412   5348200

[Val] AUC: 0.5543638987687336
test AUC: 0.5376
[TEST] AUC: 0.5376174109499735
```

Figure 23: Classification report of Baseline MLP model
(Precision, Recall, F1)

```
Epoch 15/30, Loss: 0.6645, Val AUC: 0.5594, Time: 199.56s
Epoch 16/30, Loss: 0.6640, Val AUC: 0.5575, Time: 199.75s
Epoch 17/30, Loss: 0.6636, Val AUC: 0.5573, Time: 199.72s
Epoch 18/30, Loss: 0.6632, Val AUC: 0.5569, Time: 200.39s
Epoch 19/30, Loss: 0.6629, Val AUC: 0.5582, Time: 200.47s
Epoch 20/30, Loss: 0.6626, Val AUC: 0.5564, Time: 199.59s
Epoch 21/30, Loss: 0.6622, Val AUC: 0.5567, Time: 200.15s
Epoch 22/30, Loss: 0.6620, Val AUC: 0.5558, Time: 199.10s
Epoch 23/30, Loss: 0.6617, Val AUC: 0.5557, Time: 198.30s
Epoch 24/30, Loss: 0.6615, Val AUC: 0.5567, Time: 199.40s
Epoch 25/30, Loss: 0.6612, Val AUC: 0.5556, Time: 199.33s
Epoch 26/30, Loss: 0.6610, Val AUC: 0.5557, Time: 198.10s
Epoch 27/30, Loss: 0.6608, Val AUC: 0.5562, Time: 198.58s
Epoch 28/30, Loss: 0.6606, Val AUC: 0.5561, Time: 198.95s
Epoch 29/30, Loss: 0.6604, Val AUC: 0.5553, Time: 202.49s
Epoch 30/30, Loss: 0.6602, Val AUC: 0.5544, Time: 198.58s
```

Figure 24: Training process demonstration

5.3 MLP model with feature selection and functional improvement

The enhanced MLP model incorporates several key improvements:

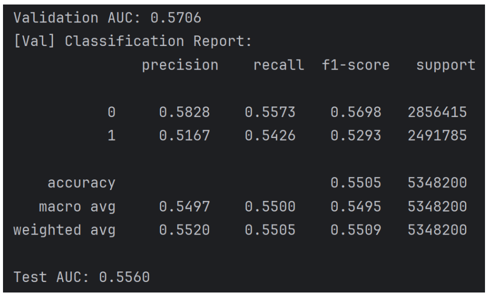
- Added `Dropout` layers (p=0.3) after hidden layers for better regularization
- Implemented class weighting via `pos_weight` to address imbalance
- Used feature selection (top 38 features) instead of all original features
- Enhanced reproducibility through comprehensive random seed control
- Added training AUC monitoring alongside validation metrics

```
Validation AUC: 0.5732
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5828    0.5798    0.5813   2856415
           1     0.5211    0.5242    0.5226   2491785

    accuracy                         0.5539   5348200
   macro avg     0.5520    0.5520    0.5520   5348200
weighted avg     0.5541    0.5539    0.5540   5348200

[Val] AUC: 0.5732464366179105
Test AUC: 0.5562
[TEST] AUC: 0.5562462054863556
```

Figure 25: Classification report of enhanced MLP model(Precision, Recall, F1)

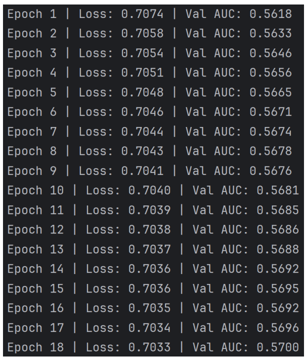
5.4 RobustMLP model with interaction features

- **Feature Engineering**: Created 45 interaction features from top-10 important features (${}_{10}C_2$ combinations)
- **Architecture**: Implemented residual connections with:
  - Batch normalization layers
  - LeakyReLU activation ($\alpha = 0.01$)
  - Skip connection between hidden layers
- **Training Optimization**:
  - AdamW optimizer with weight decay ($1e^{-4}$)
  - Cyclic learning rate scheduling (triangular2 policy)
  - Kaiming/Xavier initialization for stable training



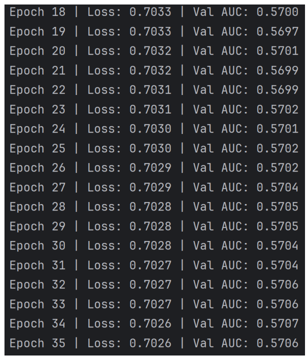
```
Validation AUC: 0.5706
[Val] Classification Report:
              precision    recall  f1-score   support

           0     0.5828    0.5573    0.5698   2856415
           1     0.5167    0.5426    0.5293   2491785

    accuracy                         0.5505   5348200
   macro avg     0.5497    0.5500    0.5495   5348200
weighted avg     0.5520    0.5505    0.5509   5348200

Test AUC: 0.5560
```

Figure 26: Classification report of RobustMLP MLP model (Precision, Recall, F1)



```
Epoch 1 | Loss: 0.7074 | Val AUC: 0.5618
Epoch 2 | Loss: 0.7058 | Val AUC: 0.5633
Epoch 3 | Loss: 0.7054 | Val AUC: 0.5646
Epoch 4 | Loss: 0.7051 | Val AUC: 0.5656
Epoch 5 | Loss: 0.7048 | Val AUC: 0.5665
Epoch 6 | Loss: 0.7046 | Val AUC: 0.5671
Epoch 7 | Loss: 0.7044 | Val AUC: 0.5674
Epoch 8 | Loss: 0.7043 | Val AUC: 0.5678
Epoch 9 | Loss: 0.7041 | Val AUC: 0.5676
Epoch 10 | Loss: 0.7040 | Val AUC: 0.5681
Epoch 11 | Loss: 0.7039 | Val AUC: 0.5685
Epoch 12 | Loss: 0.7038 | Val AUC: 0.5686
Epoch 13 | Loss: 0.7037 | Val AUC: 0.5688
Epoch 14 | Loss: 0.7036 | Val AUC: 0.5692
Epoch 15 | Loss: 0.7036 | Val AUC: 0.5695
Epoch 16 | Loss: 0.7035 | Val AUC: 0.5692
Epoch 17 | Loss: 0.7034 | Val AUC: 0.5696
Epoch 18 | Loss: 0.7033 | Val AUC: 0.5700
```

Figure 27: Training process1 of RobustMLP MLP model



```
Epoch 18 | Loss: 0.7033 | Val AUC: 0.5700
Epoch 19 | Loss: 0.7033 | Val AUC: 0.5697
Epoch 20 | Loss: 0.7032 | Val AUC: 0.5701
Epoch 21 | Loss: 0.7032 | Val AUC: 0.5699
Epoch 22 | Loss: 0.7031 | Val AUC: 0.5699
Epoch 23 | Loss: 0.7031 | Val AUC: 0.5702
Epoch 24 | Loss: 0.7030 | Val AUC: 0.5701
Epoch 25 | Loss: 0.7030 | Val AUC: 0.5702
Epoch 26 | Loss: 0.7029 | Val AUC: 0.5702
Epoch 27 | Loss: 0.7029 | Val AUC: 0.5704
Epoch 28 | Loss: 0.7028 | Val AUC: 0.5705
Epoch 29 | Loss: 0.7028 | Val AUC: 0.5705
Epoch 30 | Loss: 0.7028 | Val AUC: 0.5704
Epoch 31 | Loss: 0.7027 | Val AUC: 0.5704
Epoch 32 | Loss: 0.7027 | Val AUC: 0.5706
Epoch 33 | Loss: 0.7027 | Val AUC: 0.5706
Epoch 34 | Loss: 0.7026 | Val AUC: 0.5707
Epoch 35 | Loss: 0.7026 | Val AUC: 0.5706
```

Figure 28: Training process2 of RobustMLP MLP model

5.5 Performance Comparison with Different MLP models

We tried different feature engineering and gradually replaced the model's feature selection based on the training situation.

Table 3: Comparison of different MLP models performance

Model Name	Val AUC	Test AUC	Accuracy	Class 1 Recall
Baseline MLP	0.5544	0.5376	0.5445	0.4493
feature selection MLP	**0.5732**	**0.5562**	**0.5539**	0.5242
Robust MLP	0.5706	0.5560	0.5505	**0.5426**

From the table above, we make the following observations:

- **AUC:** The feature-selected MLP achieved the highest validation AUC, showing better ranking capability than both baseline and robust variants
- **Accuracy:** All models showed similar accuracy, suggesting the task's difficulty and potential class imbalance
- **Class 1 Recall:** he robust MLP achieved the best recall for the class 1, indicating superior minority-class identification

5.6 Comparison with LightGBM

We compare the LightGBM and MLP models to examine their respective advantages. The tree-based LightGBM demonstrates inherent strengths through:

Table 4: Comparison of MLP models and LightGBM

Model Name	Val AUC	Test AUC	Accuracy	Class 1 Recall
LightGBM	**0.5830**	**0.5675**	**0.5639**	0.4716
Baseline MLP	0.5544	0.5376	0.5445	0.4493
feature selection MLP	0.5732	0.5562	0.5539	0.5242
Robust MLP	0.5706	0.5560	0.5505	**0.5426**

- **AUC**: LightGBM outperforms all MLP variants, demonstrating superior ranking capability
- **Accuracy**: LightGBM achieves highest accuracy, though MLP models show comparable performance
- **Class 1 Recall**: The robust MLP shows best recall, exceeding LightGBM by 7.1 percentage points

5.7 Prediction and Visualization on Future Samples

We take the prediction maps of the same area from feature selection MLP and Robust MLP for comparison.

Figure 29: MLP prediction vs "Future" Data of feature selection MLP and Robust MLP

5.8 Summary analysis of MLP and other models

Unlike the comparison between XGBoost and Logistic Regression, the LightGBM model and the MLP model each have their own advantages and disadvantages. Here, we will conduct a relevant analysis.

- **LightGBM's Superior AUC (0.5830 vs MLP's 0.5732)**:
  - Missing Value Handling: LightGBM's native treatment of missing values (`use_missing=True`) preserves information that MLPs lose during imputation
  - Model Bias: Tree models' axis-aligned splits better capture the dominant, simple patterns that contribute most to AUC
- **Robust MLP's Higher Recall (0.5426 vs LightGBM's 0.4716)**:
  - Feature Interaction: The cross-feature engineering ($\mathbf{x}_i \odot \mathbf{x}_j$) explicitly models nonlinear relationships
  - Representation Learning: Hidden layers ($\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$ learn transformed feature spaces
  - Class Rebalancing: **The MLP demonstrates superior handling of class imbalance through:**
    - Explicit pos_weight adjustment ($\alpha = \frac{N_{neg}}{N_{pos}}$)
    - End-to-end optimization of recall-oriented loss
    - Flexible decision boundaries via sigmoid activations
  - Decision Boundary: Continuous sigmoid outputs ($\hat{y} = \frac{1}{1+e^{-z}}$) allow finer-grained minority class identification
- **Comparable Accuracy (0.55)**:
  - Metric Sensitivity: Accuracy measures overall correctness, masking class-specific performance
  - Error Tradeoff: LightGBM's FP/FN balance differs from MLP's recall-oriented optimization
  - Data Characteristics: Suggests the dataset has irreducible label noise affecting all models similarly

All tested models (linear, tree-based, and neural networks) demonstrated suboptimal performance in both accuracy and AUC metrics, indicating the dataset likely contains mixed linear and nonlinear relationships that cannot be fully captured by any single model architecture. The consistently modest results across different approaches suggest fundamental data challenges: low signal-to-noise ratio, potentially noisy labels, and limited feature expressiveness due to anonymization. While more sophisticated feature engineering and hybrid modeling could potentially improve results, these approaches currently lie beyond our project scope.

6 Conclusion

In this project, we explored multiple machine learning models to address the Jane Street market forecasting task, including Logistic Regression, XGBoost, LightGBM, and various Multi-Layer Perceptron (MLP) architectures. Across all models, the predictive performance remained modest, with AUC values ranging from 0.55 to 0.58 and accuracy fluctuating around 55% to 56%. This suggests the dataset is noisy, weakly informative, or inherently difficult due to a low signal-to-noise ratio.

XGBoost and LightGBM consistently outperformed the linear baseline in terms of AUC and accuracy, showing their ability to capture nonlinear patterns. Meanwhile, the Robust MLP model achieved the highest recall for the minority class (profitable trades), highlighting its strength in handling imbalanced data and complex feature interactions.

These results imply the dataset contains both linear and nonlinear dependencies. While linear models capture simple relations, expressive models like MLPs and boosted trees are better suited for learning high-order feature interactions. However, no single model excelled across all metrics, indicating the potential value of hybrid methods that combine linear and nonlinear modeling.

Future improvements may include:

- Richer feature engineering and transformation;
- Ensemble approaches combining diverse model families;
- Specialized architectures for tabular data, such as Wide & Deep models, TabNet, or TabTransformer.

Overall, the XGBoost model with 65 selected features and LightGBM models offered the best balance of interpretability, complexity, and efficiency. For scenarios prioritizing minority class recall, Robust MLP is preferable. Nevertheless, the relatively limited gains across all models indicate a need for further exploration into model design and feature representation to fully realize the predictive potential of financial data.