Real-Time Market Data Forecasting

Final Report for CS 4563

Mingjian Li, Wes Simpson
Professor Linda N. Sellie
NYU Tandon School of Engineering

Abstract
This project focuses on forecasting real-time market data using various machine learning techniques, including linear regression, neural networks, and Long Short-Term Memory (LSTM) networks. The data, sourced from a Kaggle competition, consists of time-series with 79 features and 9 responders. We employed preprocessing strategies like sampling, outlier handling, and normalization to manage computational resources and improve data quality. Our experiments explored different modeling approaches to identify the most effective methods for predicting market trends, demonstrating varying levels of success across the models.

Contents

1 Introduction

The Dataset was chosen from a Kaggle Competition: Jane Street Real-Time Market Data Forecasting. It comprises a set of time series with 79 features and 9 responders, anonymized but representing real market data. The goal of the competition is to forecast one of these responders, i.e., responder_6, for up to six months in the future. It also comprises date_id and time_id providing a chronological structure to the data, symbol_id identifying a unique financial instrument, and weights for calculating the scoring function.

We have sampled and preprocessed the data and chose a linear regression model with feature transformations, various neural network models, and an LSTM to fit our data. During the process, we tried different regularization values, hyper-parameters and architectures.

2 Data Analysis

2.1 Data Preprocessing

Overview

In this section, we implemented data preprocessing, which includes: sampling, missing value methods, outlier handling, categorical data encoding, scaling and normalization. The associated work can be found in the notebooks ***data-sampling-js.ipynb*** and ***data-preprocess-exploration-part-i.ipynb***.

Sampling and Memory Reduction

The dataset provided in the competition comprises 4.5 million rows and 79 features, which exceeds our available computational resources. To manage this, we employed a sampling strategy that respects the property and structure of the data, ensuring that our sample is representative of the original dataset.

The dataset is structured as a time-series, with date_id being a key feature. We assumed that if the data volume from each date_id in our sample is proportionate to that in the original dataset, then our sample would be representative. Under this assumption, we grouped the data by date_id and sampled 1% of it. Additionally, we adopted a memory reduction technique from a notebook provided in the competition discussion forum. This method involves manipulating data types to reduce memory usage, which successfully cut our data size by 47.6%.

Categorical Data

The data set contains only numerical data, thus there is no need to for categorical encoding. But, we still identified that feature_09, feature_10, feature_11 might be numerical categorical data from the scaled and normalized histograms [1] .

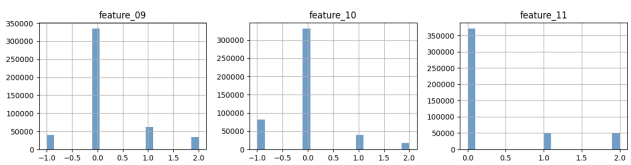

Figure 1: Categorical Features Histograms

Missing Values

We identified missing values across several features in the dataset. Our approach involved:

Categorical Features: Missing values in categorical features were filled using the most frequent value in each category.

Numerical Float Features: For numerical features, missing values were imputed using the mean of each column to minimize the impact of missing values.

Outliers

Outlier detection was conducted using the inter-quartile range (IQR) methodology and visualized with box plots. We initially chose the most commonly used range: $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$ but found that the outlier ratio was too high, indicating that we should raise the threshold.

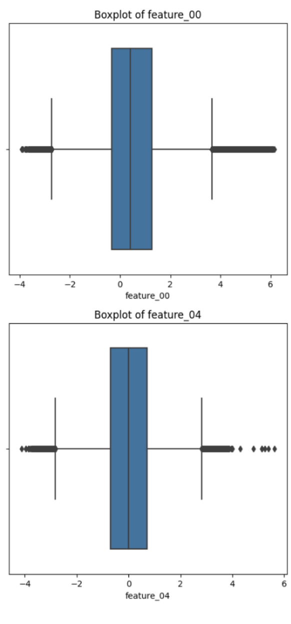

Figure 2: Low Threshold Boxplots

Feature	Outlier Percentage
feature_00	1.136831
feature_01	0.702036
feature_02	1.101411
feature_03	1.117743
feature_04	0.547630
feature_05	7.437761
feature_06	8.649037
feature_07	8.076804
feature_08	5.697518
feature_09	0.000000
feature_10	0.000000
feature_11	10.542625
feature_12	6.336349
feature_13	9.247570
feature_14	7.267872
feature_15	8.429094
feature_16	8.731330
feature_17	8.655824

Table 1: Low Threshold Outlier Percentage

Thus, we raised the threshold, yielding a new range: $[Q1 - 5 \times IQR, Q3 + 5 \times IQR]$. This threshold has a more satisfying result.

Feature	Outlier Percentage
feature_00	0.000000
feature_01	0.000000
feature_02	0.000000
feature_03	0.000000
feature_04	0.000000
feature_05	0.339989
feature_06	0.577960
feature_07	0.496515
feature_08	0.202339
feature_09	0.000000
feature_10	0.000000
feature_11	0.000000
feature_12	0.905647
feature_13	2.302295
feature_14	1.259422
feature_15	2.200490
feature_16	2.378862
feature_17	2.320111

Table 2: High Threshold Outlier Percentage

We initially tried to remove all the outliers, but this method reduced our data size from (471486, 79) to (37822, 79). We want to retain the size of our data, thus we handle the outliers by replacing them with mean values. For categorical features (int type column feature), we replace the outliers with rounded mean for data type consistency.

Scaling and Normalizing

For scaling, we remove the mean of the data and scale them to unit variance. For Normalizing, we use the Min and Max scaling method. We only scale and normalize feature 00 to 78 and responders. We do not scale or normalize date_id, time_id, and symbol_id. Because they are data to structure the data set.

For integer columns, we cast the float part.

Sometimes, normalization will not be useful. Thus, we have 2 versions of new data. One is normalized and the other is not. Both of them are scaled.

For consistency, we used the same method to scale and normalize the training set and the test set.

2.2 Unsupervised Learning

Overview

In our exploration of the data, we used multiple visualization methods and unsupervised learning methods including clustering and principle component analysis. The associated work can be found in the notebooks ***data-preprocess-exploration-part-ii.ipynb***.

Visualize Data Distribution

To analyze the data distribution, we utilize histograms and density plots for each feature.

Before scaling and normalizing, the histograms revealed three main types of distributions: Well-distributed, extremely unbalanced, and multi modal.

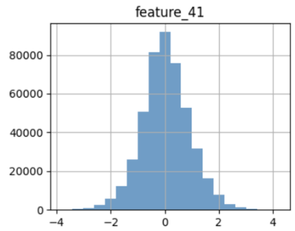

Figure 3: Well-Distributed Before Scaling and Normalizing

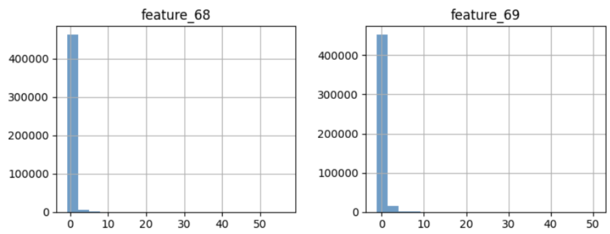

Figure 4: Extremely Unbalanced Before Scaling and Normalizing

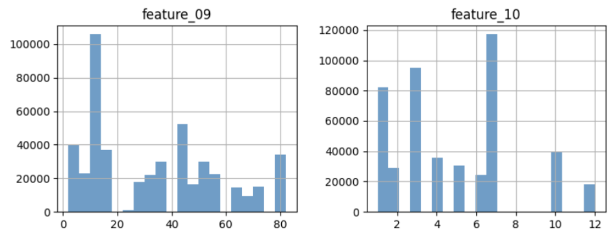

Figure 5: Multimodal Before Scaling and Normalizing

Some features exhibited very small variance, with data clustered around a single value. This behavior could be attributed to the following reasons:

1. Plot Size Limitations: The current plot dimensions may obscure key distribution details. A more tailored visualization approach will be applied in "Histograms Part II."

2. Small Scale of Original Data: Some features may have inherently small scales. We will address this issue using scaling techniques in subsequent steps.

3. Imbalanced Features: Certain features are inherently unbalanced. These will be analyzed further during the modeling process. Additionally, irrelevant features may be considered for reduction. Examples of such features include: [05, 06, 07, 08, 12, 13, 14, 15, 16, 17, 21, 29, 30, 31, 37, 38, 47, 48, 49, 58, 59, 60, 62, 63, 64, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78].

After applying scaling, we observed significant improvements in the distribution of most originally unbalanced features.

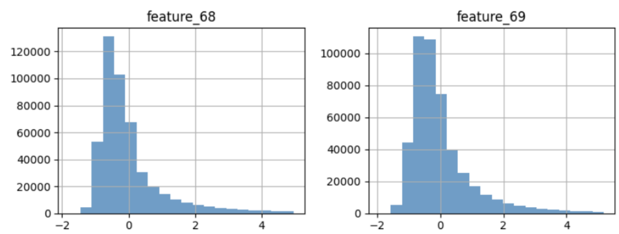

Figure 6: After Scaling

We also plotted density graphs after scaling. The results align closely with our observations and analyses based on histograms, further validating our findings.

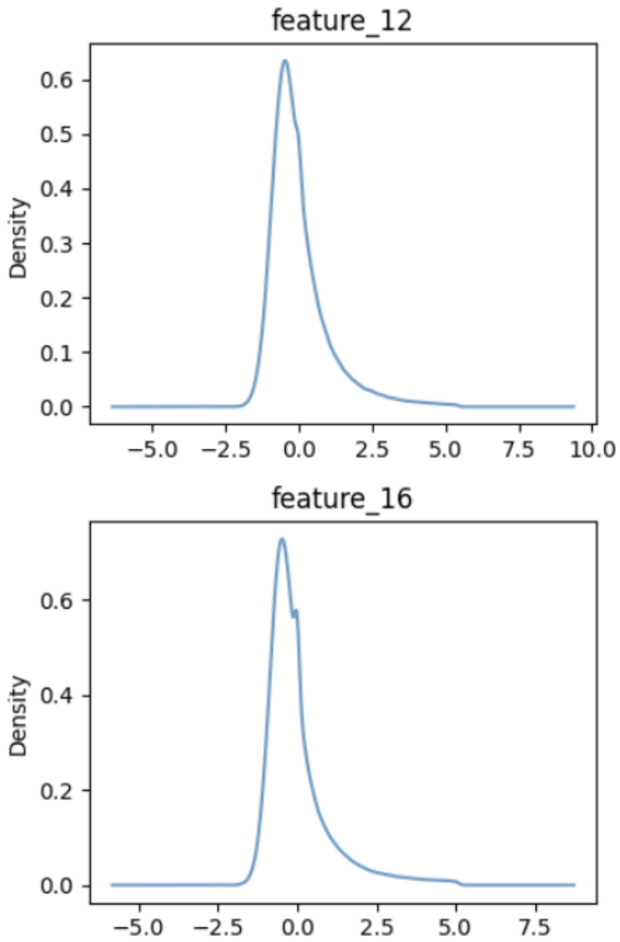

Figure 7: Enter Caption

Visualize Individual Features with Target

To examine the relationship between each feature and the target variable, we created scatter plots for all features.

For most features, the scatter plots displayed a highly random pattern, as shown in Figure 8. This indicates the absence of a simple linear relationship between these features and the target. Furthermore, some features may not provide valuable information. To address this, we plan to conduct further analyses, including clustering and principal component analysis (PCA).

Another possible explanation for the randomness is that the scatter plots do not account for temporal aspects. If the data exhibit seasonal trends as part of a time-series structure, scatter plots alone may fail to provide accurate insights.

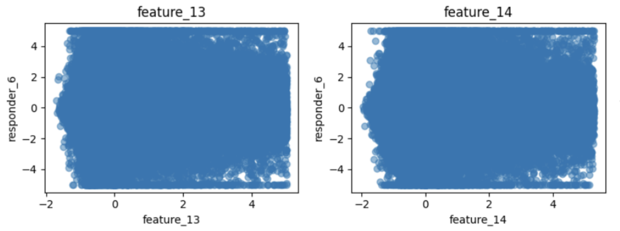

Figure 8: Random

As shown in Figure 9, features 9, 10, and 11 exhibit behavior characteristic of categorical features that have been encoded. This confirms their categorical nature.

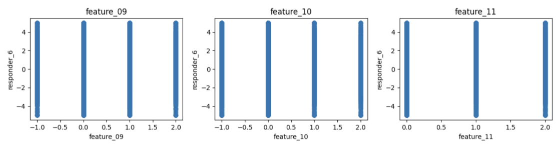

Figure 9: Categorical

Some features form round-shaped patterns in the scatter plots, as seen in Figure 10. This may indicate a polynomial relationship. We will explore this possibility further by applying polynomial feature transformations.

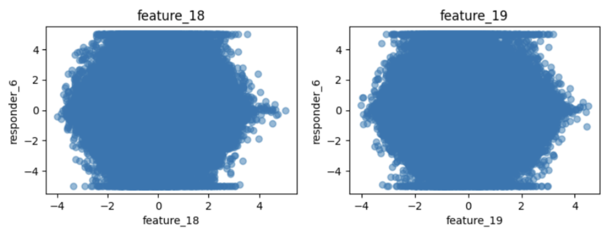

Figure 10: Round-shaped

In summary, the scatter plot analysis suggests two possibilities:

1. If all features contribute meaningful information, the relationship between features and

the target variable is likely complex, necessitating the use of advanced models.

2. If some features are irrelevant or redundant, feature reduction techniques will be required to improve model performance.

Visualize Correlation Matrix

To explore the relationships between pairs of features, we generated a heat map of the correlation matrix. This visualization highlights the degree of correlation between each pair of features.

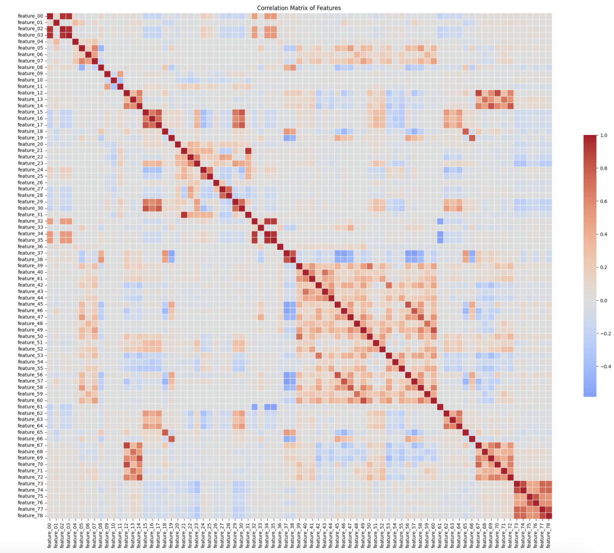

Figure 11: Correlation Matrix Hea tMap

Table 3 lists all the highly correlated feature pairs identified from the correlation matrix.

Table 3: Highly Correlated Features

Feature 1	Feature 2	Correlation
feature_00	feature_02	0.943807
feature_00	feature_03	0.945857
feature_02	feature_03	0.947548
feature_12	feature_67	0.876071
feature_12	feature_70	0.862875
feature_14	feature_69	0.823701
feature_14	feature_72	0.807690
feature_15	feature_17	0.821089
feature_15	feature_30	0.809970
feature_21	feature_31	0.930829
feature_32	feature_34	0.909946
feature_32	feature_35	0.914050
feature_34	feature_35	0.943466
feature_37	feature_38	0.824361
feature_45	feature_56	0.825425
feature_46	feature_57	0.814879
feature_47	feature_58	0.809230
feature_49	feature_60	0.830737
feature_73	feature_74	0.861515
feature_75	feature_76	0.815475
feature_77	feature_78	0.858338

For these highly correlated features, we propose the following strategies:

1. Feature Removal: Eliminate some of the highly correlated features to reduce multicollinearity and simplify the model.

2. Regularization: Apply techniques such as Lasso or Ridge regression to minimize the impact of less important features.

3. Feature Engineering: Create new features that represent combinations or transformations of the highly correlated features to better capture their relationships.

Pattern Exploration - Clustering

To gain a deeper understanding of the relationships within the data, we applied the K-Means clustering algorithm.

We first determined the optimal number of clusters, K, using the elbow method.

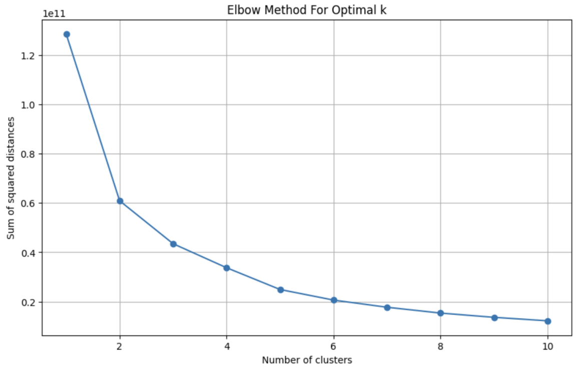

Figure 12: Elbow Method

The elbow method indicated that the best K is 2.

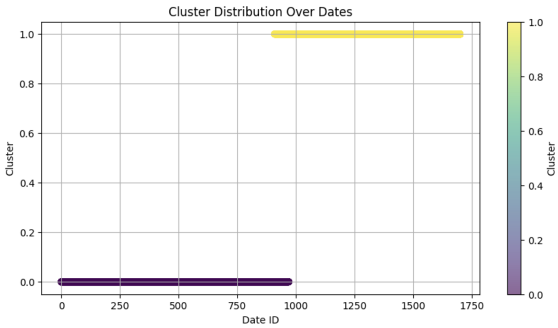

Figure 13: Cluster Distribution

Our data are separated into 2 clusters. After observing the cluster distribution with respect to date_id, we have the following guessing:

1. Temporal Patterns: Features exhibit distinct patterns during two different time periods.

2. Continuous Change Over Time: Since the data are time-series, the observed distribution may reflect a gradual evolution in patterns over time.

Pattern Exploration - PCA

Building on our previous analysis, we utilized principal component analysis (PCA) to reduce the dimensionality of the dataset. We performed PCA with N = 2 and N = 3 components, tracking the explained variance ratio for each case.

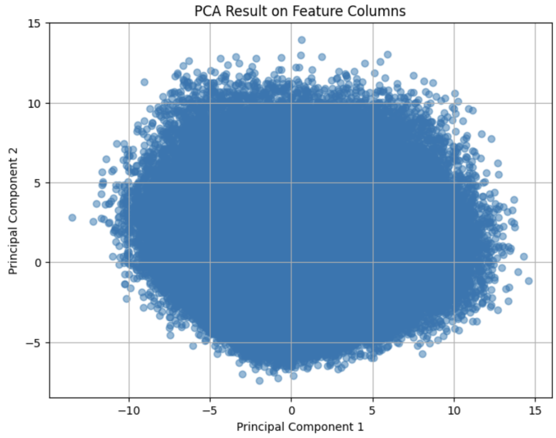

Figure 14: Component Number = 2

For N = 2, Principal Component 1 (PC1) and Principal Component 2 (PC2) explain 11.01% and 8.98% of the variance, respectively.

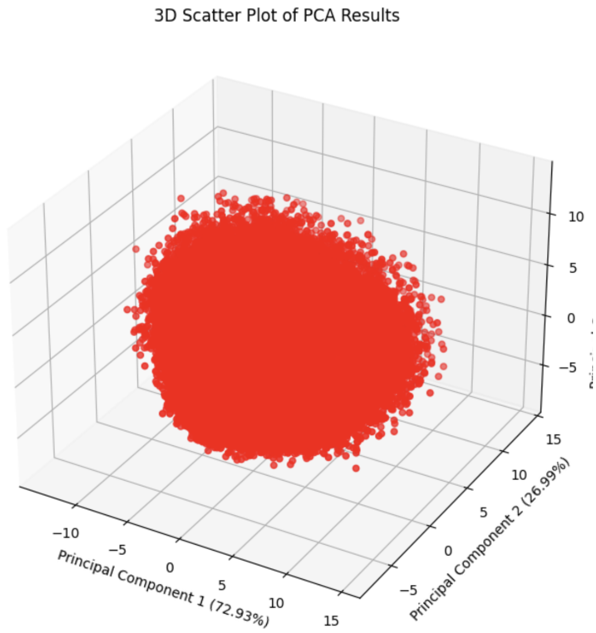

Figure 15: Component Number = 3

For N = 3, PC1, PC2, and Principal Component 3 (PC3) explain 11.01%, 8.98%, and 7.93% of the variance, respectively.

Observations:

1. The data points form an elongated, slightly circular distribution, extending diagonally from the bottom-left to the top-right in the scatter plot.

2. This pattern suggests a possible linear relationship between PC1 and PC2 for a significant portion of the data, as the points tend to align along a line or curve.

3. PC1 captures the largest share of variability, with data points showing the greatest spread along this axis. No clear clustering is evident in the PCA scatter plots, indicating that the dataset does not contain strongly distinct or separable groups based on these two or three components.

Conclusion: The PCA results indicate that while dimensionality reduction helps in identifying key patterns, the explained variance for the first few components is relatively low. This suggests that the variability is spread across multiple dimensions, requiring further exploration to effectively capture the structure of the data. Future steps may include experimenting with more components or leveraging domain-specific knowledge to refine feature engineering.

3 Model Fitting

3.1 Linear Regression

Overview

We explored linear regression with various feature transformations, including the polynomial kernel, RBF kernel, and PCA, combined with regularization techniques. These approaches aimed to capture potential linear relationships after data transformations.The associated work can be found in the notebooks ***linear-regression.ipynb***

Preparation

Data: We have preprocessed out data in Section 2. We only split the processed data into training set(80%) and validation set(20%) here.

Metric: The competition used sample weighted zero-mean R-squared score, which has a much smaller value compared to R square. Since the data is complex and we expect linear regression will not perform well, we used R square in this part which can be easier to distinguish.

Polynomial Transformation

Observing the scatter plot between some features and the target, which exhibits a rounded shape, we assumed that a polynomial transformation with a degree of 2 might capture some valuable information.

Surprisingly, this method yielded a negative R-squared value of -0.00413879.

The negative R-square of regression with polynomial kernel indicates that out model's performance is slightly worse than simply predicting the mean of the target variable for all observations.The model does not fit the data well. It is not capturing the variance in the target variable effectively.

Due to its excessive computational demands and extremely poor results, we discontinued this method in favor of exploring alternative approaches.

RBF Kernel + Regularization

Given the high dimensionality and complexity of our data, the RBF Kernel is a suitable method for learning complex patterns and relationships. We also incorporated Ridge regularization to mitigate overfitting and experimented with different hyperparameter values.

Alpha	Training R^2	Validation R^2
0.1	0.020284	0.000957
0.5	0.011361	0.000885
1.0	0.009593	0.001028
5	0.007052	0.001818
10	0.005928	0.001964
100	0.001860	0.000769
1000	0.000271	-0.000033
10000	0.000029	-0.000170

Table 4: Training and validation R^2 scores for different alpha values

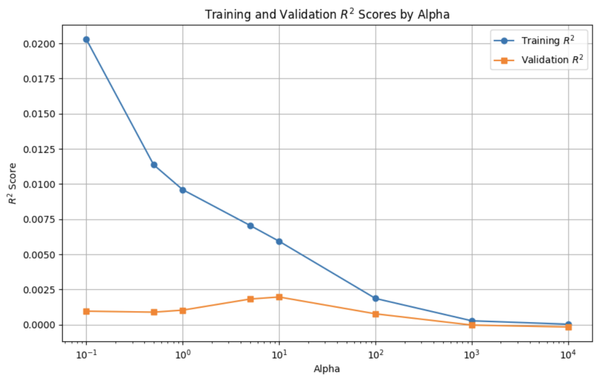

Figure 16: RBF Kernel With Regularization

The RBF Kernel displayed better performance than polynomial transformation, and Ridge Regression proved effective. We observed a significant gap between Training R-squared and Validation R-squared at lower alpha values, indicating over-fitting. The optimal performance was noted at $\alpha = 10$, although the model began under-fitting at higher alpha values.

PCA + Regularization

One of our assumption during data analysis is that in out 79 features, some of them do not carry valuable information. Thus, we applied PCA to reduce the feature space, hoping to achieve better results, and combined this with Ridge regularization to prevent over-fitting.

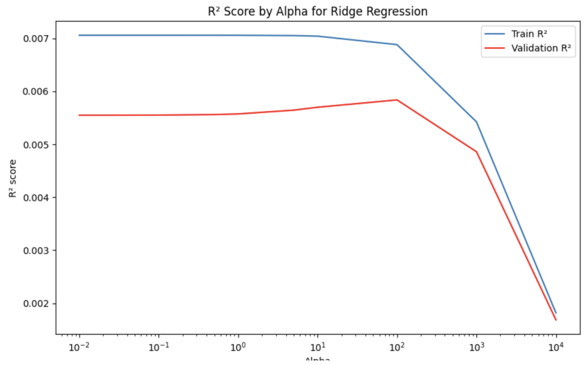

Figure 17: Best Performing PCA

Observing with the best number of components, the model performs best when $\alpha = 10$

Components	Training R^2	Validation R^2
1-4	0.000	0.000
5-11	0.001	0.001
12-19	0.002	0.002
20-22	0.003	0.002
23-24	0.004	0.004
25-29	0.005	0.004
30-45	0.006	0.005
46	0.006	0.006
47-77	0.007	0.006

Table 5: General Mean R Square

We tracked the mean R-squared across different numbers of components, with the best performance observed at 46 components.

The PCA model outperformed the RBF Kernel with regularization, though it still tended to under-fit.

Model Evaluation

Of all the methods tested, the polynomial transformation performed the worst, while PCA with regularization yielded the best results. Given its lower computational demands, PCA is the preferred method in linear regression analysis.

3.2 Neural Network

Overview

In this part, we tried Neural Network with 3 types of Multi-Layer Perceptrons. The associative work can be found in ***nn_supervised_analysis.ipynb***.

Preparation

Data: We have preprocessed out data in Section 2. We only split the processed data into training set(80%) and validation set(20%) here.

Metric: We used R square for model evaluation. The model is trained using MSE loss function.

Architecture

Model I: 2 hidden layers with RELU Activation Function
1st hidden layer: 128 neurons
2nd hidden layer: 64 neurons

Model II: 2 hidden layers with RELU Activation Function
1st hidden layer: 512 neurons
2nd hidden layer: 256 neurons

Model III: 4 hidden layers with RELU Activation Function
1st hidden layer: 128 neurons
2nd hidden layer: 128 neurons
3rd hidden layer: 64 neurons
4th hidden layer: 64 neurons

Results

Model	MSE	R^2 Score
Model 1	0.8013	0.0008
Model 2	0.8019	-0.0000
Model 3	0.8021	-0.0001

Table 6: Evaluation Metrics (MSE and R^2 Score) for Neural Networks

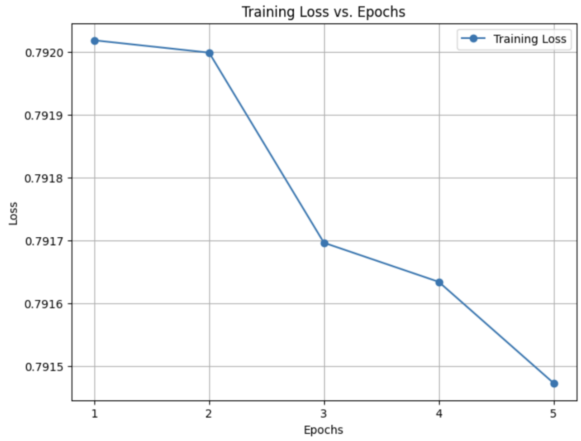

Figure 18: Model I

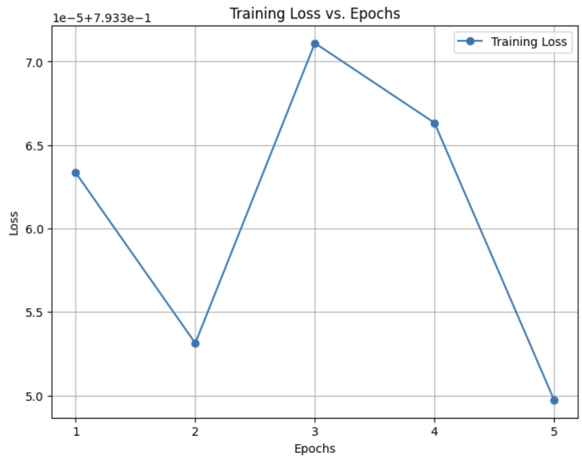

Figure 19: Model II

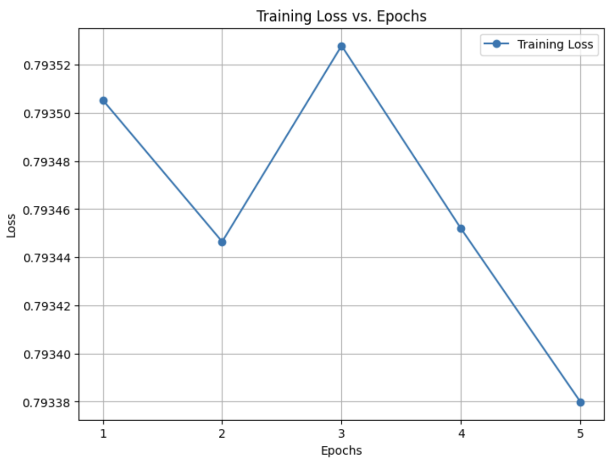

Figure 20: Model III

Model Overview Model I outperformed the rest. Its MSE converges faster and steadier with the increase of epoch and it has the highest validation R square.

3.3 LSTM

Overview

LSTM (Long Short-Term Memory) networks are particularly suited for forecasting market data and can effectively manage time-series data, capturing long-term dependencies that other models might overlook.

In this part, we implemented LSTM model with PyTorch. We tried 3 different architectures and added regularization. Since the model takes too much computational power, we only try one regularization value for each architecture.

Preparation

Data: We have preprocessed out data in Section 2. We only split the processed data into training set(80%) and validation set(20%) here.

Metric: We used the MSE as the loss function to perform optimization. We also kept track of sample weighted zero-mean R-squared score to select model. It has a much smaller value compared to R square.

Epoch: We chose Epoch number to be 20 due to limited computational power but still yielded a decent result.

$$R^2 = 1 - \frac{\sum w_i (y_i - \hat{y}_i)^2}{\sum w_i y_i^2}$$

Architecture I

Firstly, we tried an LSTM with 2 hidden layers and 64 hidden size. We keep track of the loss and the weighted R square.

Epoch	Train Loss	Val Loss
1	1.5805	1.6041
2	1.5815	1.5998
3	1.5820	1.5996
4	1.5819	1.5994
5	1.5798	1.5991
6	1.5815	1.5985
7	1.5800	1.6015
8	1.5797	1.5994
9	1.5819	1.5987
10	1.5793	1.6001
11	1.5790	1.5996
12	1.5804	1.5990
13	1.5794	1.5993
14	1.5800	1.5799
15	1.5797	1.5998
16	1.5788	1.5977
17	1.5781	1.5994
18	1.5780	1.5975
19	1.5780	1.5987
20	1.5773	1.5976

Table 7: Train and Validation Loss per Epoch

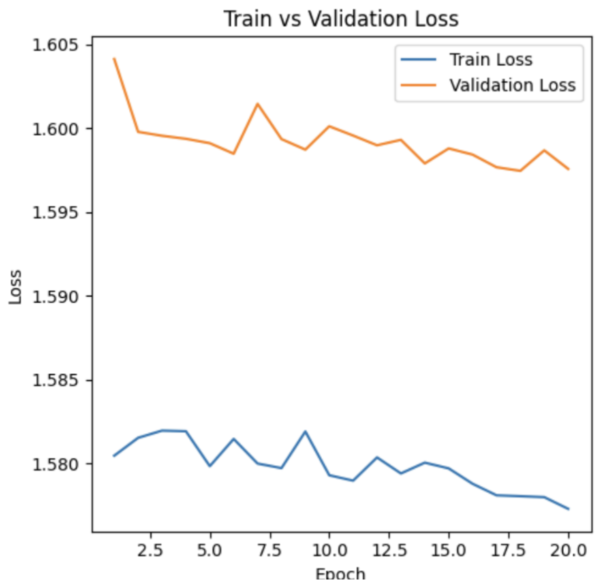

Figure 21: Train and Validation Loss

Epoch	Train R^2	Val R^2
1	0.0028	0.0027
2	0.0051	0.0051
3	0.0054	0.0055
4	0.0053	0.0052
5	0.0054	0.0054
6	0.0060	0.0059
7	0.0045	0.0044
8	0.0057	0.0055
9	0.0060	0.0057
10	0.0046	0.0044
11	0.0053	0.0048
12	0.0058	0.0056
13	0.0058	0.0056
14	0.0064	0.0062
15	0.0060	0.0056
16	0.0061	0.0056
17	0.0069	0.0064
18	0.0068	0.0064
19	0.0059	0.0055
20	0.0068	0.0062

Table 8: Train and Validation R^2 per Epoch

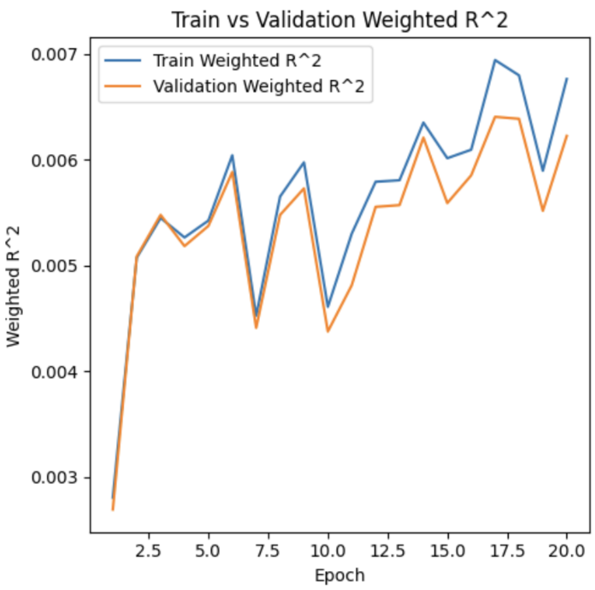

Figure 22: Train and Validation R^2

From the perspective of the weighted R square, this model performs well and we might choose the model weights at epoch 17 or increase the number of epochs and try different regularization parameters.

However, the MSE loss function presents interesting behavior. The gap between training and validation loss is quite stable thus we cannot determine whether it over-fits. We might try different values for regularization parameters in the future.

Architecture II In this part, we tried a much more complex LSTM with 3 hidden layers and 128 hidden size. We keep track of the loss and the weighted R square.

Epoch	Train Loss	Val Loss
1	1.5875	1.6086
2	1.5884	1.6084
3	1.5876	1.6084
4	1.5888	1.6085
5	1.5882	1.6083
6	1.5880	1.6084
7	1.5887	1.6091
8	1.5878	1.6085
9	1.5869	1.6084
10	1.5881	1.6086
11	1.5883	1.6085
12	1.5874	1.6086
13	1.5860	1.6092
14	1.5884	1.6084
15	1.5870	1.6085
16	1.5878	1.6084
17	1.5878	1.6091
18	1.5875	1.6085
19	1.5877	1.6083
20	1.5870	1.6083

Table 9: Train and Validation Loss per Epoch

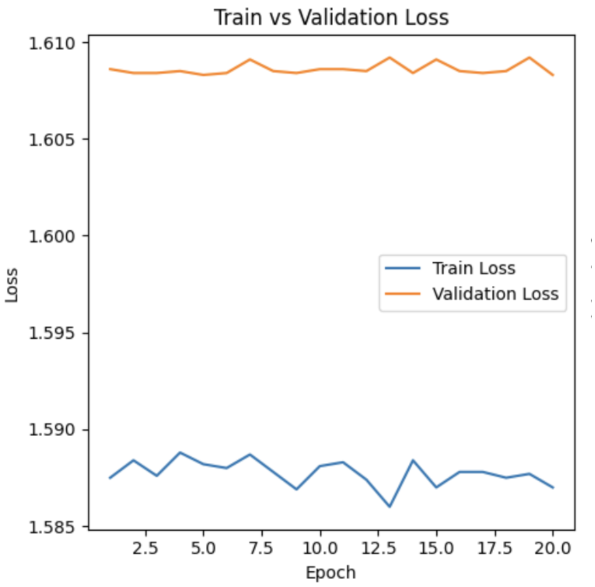

Figure 23: Train and Validation Loss

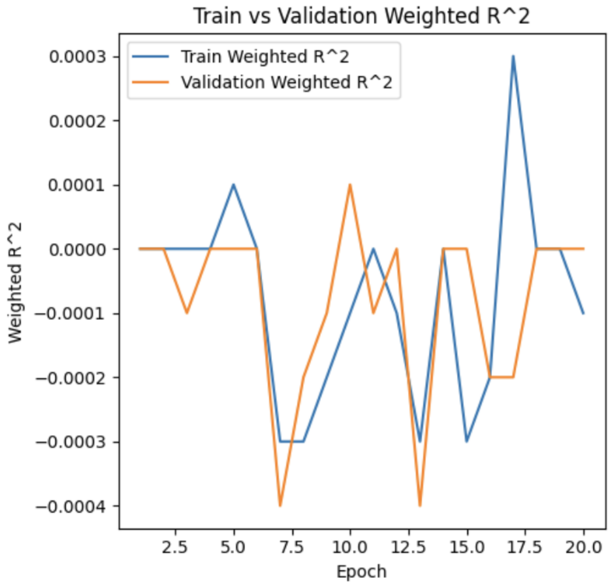

Figure 24: Train and Validation R^2

Epoch	**Train** R^2	**Val** R^2
1	-0.0000	-0.0001
2	-0.0000	0.0000
3	-0.0000	0.0000
4	-0.0000	-0.0001
5	-0.0001	0.0000
6	0.0000	0.0000
7	-0.0003	0.0004
8	-0.0003	-0.0002
9	-0.0002	-0.0001
10	-0.0001	-0.0001
11	0.0000	0.0000
12	-0.0001	0.0001
13	-0.0003	-0.0004
14	0.0000	0.0000
15	-0.0003	-0.0002
16	-0.0002	0.0002
17	-0.0003	0.0004
18	0.0000	-0.0000
19	0.0000	0.0000
20	-0.0001	-0.0000

Table 10: Train and Validation R^2 per Epoch

We can tell that we are over-fitting in this model and it performs poorly. Thus, we are going to try simpler architecture.

Architecture III

We finally tried the simplest LSTM with 1 hidden layer and 32 hidden neurons. We keep track of the loss and the weighted R square.

Epoch	**Train Loss**	**Val Loss**
1	1.5864	1.6029
2	1.5820	1.6038
3	1.5819	1.6000
4	1.5802	1.6044
5	1.5799	1.6008
6	1.5791	1.5998
7	1.5781	1.5982
8	1.5787	1.5991
9	1.5767	1.5998
10	1.5781	1.5976
11	1.5761	1.5975
12	1.5772	1.5981
13	1.5763	1.5971
14	1.5765	1.6032
15	1.5756	1.5978
16	1.5758	1.5990
17	1.5757	1.6000
18	1.5750	1.6000
19	1.5752	1.5972
20	1.5736	1.5970

Table 11: Train and Validation Loss per Epoch

Epoch	**Train** R^2	**Val** R^2
1	0.0036	0.0035
2	0.0016	0.0017
3	0.0052	0.0050
4	0.0027	0.0023
5	0.0040	0.0040
6	0.0051	0.0047
7	0.0062	0.0058
8	0.0066	0.0061
9	0.0056	0.0049
10	0.0073	0.0065
11	0.0074	0.0063
12	0.0077	0.0063
13	0.0080	0.0069
14	0.0047	0.0030
15	0.0082	0.0064
16	0.0076	0.0061
17	0.0072	0.0049
18	0.0073	0.0048
19	0.0094	0.0071
20	0.0089	0.0071

Table 12: Train and Validation R^2 per Epoch

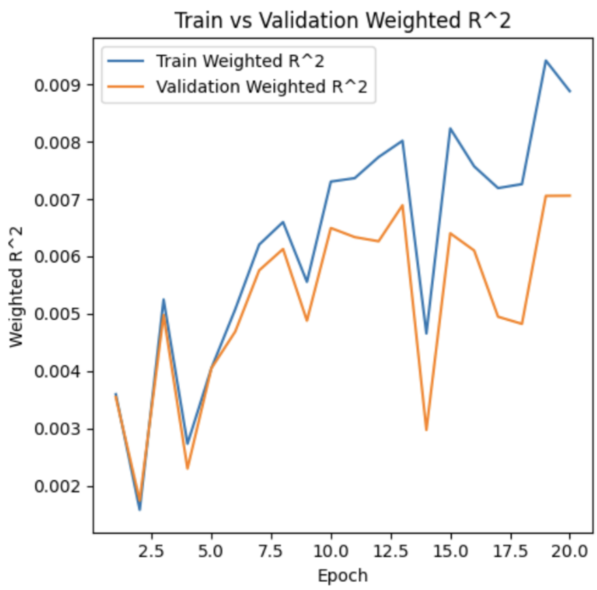

Figure 26: Train and Validation R^2

We can observe that this simplest LSTM performs really well relatively. However the weighted

R square is not quite stable. We will increase the number of epochs and evaluate how it works further.

Model Evaluation

The findings from these evaluations suggest that while each architecture has its strengths, a simpler, well-tuned model (as seen in Architecture III) often achieves a robust balance between accuracy and computational efficiency. Moving forward, continuous monitoring and adaptive tuning based on incoming data will be essential to maintain the relevance and accuracy of the forecasting models in dynamic market conditions. Further research may also explore ensemble methods that combine the strengths of different architectures to improve predictive performance and reliability.

4 Observations and Findings

4.1 Observations

We referred to a comment by Tucker Arrants on Kaggle who analyzed the metadata using their domain knowledge. We decided not to fit our models on symbol_id.

These data are highly likely to be generated by high-speed algorithms and the trades Jane-Street executes.Symbol_id likely represents different financial instruments, which can exhibit unique trading characteristics and behaviors.

These differences can skew model performance when generalized across multiple symbols. Additionally, the dynamic nature of financial markets and the evolution of trading algorithms means that the trading patterns and characteristics associated with a specific symbol_id can change over time. Thus, a model trained on historical symbol_id data might not be effective or accurate when predicting future market behaviors, as it fails to generalize well across different periods and conditions. This can lead to over-fitting

4.2 Key Findings

- The sampling strategy employed ensured the representativeness of the data, capturing the chronological structure crucial for time-series analysis.
- Memory reduction techniques successfully decreased the data size by 47.6%, enhancing computational efficiency.
- Outlier handling was optimized by adjusting the threshold, which preserved a significant amount of data while maintaining data integrity.
- Data preprocessing included effective handling of missing values and outliers, ensuring robustness and reliability in the subsequent analysis.
- Scaling and normalization significantly improved the distribution of features, which was crucial for the accuracy of predictive models.
- The unsupervised learning phase, including clustering and PCA, provided insights into the underlying structure of the data, revealing patterns not immediately apparent.
- The linear regression models, especially with PCA and regularization, showed potential, although they often under-fit, suggesting complexity in the data beyond linear relationships.
- Neural networks did not significantly outperform simpler models, indicating that more complex models might not always yield better forecasts in this dataset.
- LSTM models demonstrated the ability to capture long-term dependencies in

time-series data, although further tuning and regularization are required to optimize performance.

4.3 Improvement in the Future

- New methods to handle missing values. Maybe use regression or clustering method to fill missing values.
- Use a larger amount of the data
- Try more regularization methods and different hyper-parameters.
- Do more feature engineering.
- More time-series related visualization and analysis

5 Conclusions

The exploration of various machine learning models in forecasting real-time market data provided valuable insights into the complexity and challenges of predictive modeling in financial markets. Our findings revealed that while linear models with regularization and PCA showed promise, they often under-fitted, highlighting the complexity of the data. Neural networks did not yield significantly better results either, suggesting that complexity in model architecture did not necessarily correlate with forecasting accuracy. On the other hand, LSTM networks demonstrated potential in capturing long-term dependencies, though they certainly require further tuning. Future research should focus on enhancing model performance through more sophisticated feature engineering, and perhaps exploring ensemble methods. These efforts will improve the reliability and accuracy of forecasts, which will ideally improve model performance on the highly complex dataset.

Reference

https://www.kaggle.com/code/vroger11/quick-dataset-analysis-and-suggestions

https://www.kaggle.com/code/kevinlam/2024-jane-street-preprocessing-public

https://www.kaggle.com/code/yuanzhezhou/jane-street-baseline-lgb-xgb-and-catboost

https://www.kaggle.com/code/mpwolke/plotting-on-jane-street

https://www.kaggle.com/competitions/jane-street-real-time-market-data-forecasting/discussion/548636

https://www.kaggle.com/code/shiyili/js2024-rmf-understanding-the-data

https://www.kaggle.com/code/malakafaqahmad/xgboost-and-lstm

https://www.kaggle.com/code/michau96/jane-street-features-by-financial-instruments