Volume Prediction

Jie Yu

jennyjie20@g.ucla.com

INTRODUCTION

This project uses machine learning to predict Russell stock volume, spanning 7 steps from data analysis to model training, outlined in the Jupyter notebook below:

1. Data Analysis & Data Analysis after Mapping
2. Create predictors & Build dataset for models
3. Dense model & LSTM model
4. Analysis-dense & Analysis-lstm
5. Adjust predictors & Build new dataset for models
6. Enhancement-predictor & Enhancement-predictor
7. Analysis-metrics & Analysis-feature importance

Other notebooks: 5. DQ & 7. Validation of Model Effectiveness

In Section 1: Data Analysis (notebooks 1), I addressed data quality issues such as duplicates and missing values, validated assumptions based on field definitions, and used the intersection of datasets to develop features/predictors. Section 2: Predictors (notebooks 2) involved creating factors like technical signals, market capitalization, style and industry factors, and calendar/earnings release schedules.

In Section 3: Base Model and Section 4: Results (notebooks 3&4), I implemented Dense Neural Networks and LSTM Neural Networks based on the paper's structure, training with cumulative and single-factor sets to evaluate feature selection based on model performance and feature importance. In Section 5: Enhancement (notebooks 5&6&7), I adjusted predictors and model structures, analyzing results with the same method. In Sections 6 and 7 (notebooks 5&6&7), I compared my model to the one in the paper, discussing limitations and next steps.

Section 1: DATA ANALYSIS

To process four datasets including r3k_sec_mapping, pricing_and_volumes, earnings and barra_factor_exposures, I performed two-steps check: 1. Individual dataset validation, and 2. Integration with the Russell 3000 universe symbology mapping.

In the first step, I checked for duplicates and missing values, with the results shown in Table 1. No duplicate issues are observed. The Russell 3000 universe symbology mapping dataset has missing values in BARRID and fsym_id. Therefore, I will use ISIN, which has no missing values, as one of the primary keys for training the model. For other field with missing values, I will show more analysis in the second step.

	r3k_sec_mapping	**pricing_and_volumes**	**earning**	**barra_factor_exposures**
Column names	[barrid, barra_name, fsym_id]	[adj_volume]	[announce_datetime]	N/A

Table 1: Columns with Missing Values

r3k_sec_mapping		**pricing_and_volumes**	
Coverage check	1. The coverage line has sudden sruge and drop. 2. the number is about 3000~4000	**Coverage check**	1. The coverage line is smooth 2. the number is about 2450~3800
Each stock's weight should be positive	passed	**Field check (price)**	Outliners are found; the returns are winsorized to the range of [-1, 1].
The sum of stock weight on each date should be 1	passed	**Field check (volume)**	No outliners are found in histogram.
earning		**barra_factor_exposures**	
Coverage check	1. The coverage line is zig-zag 2. the number is about 3000~4000	**Coverage check (1-Risk Indices)**	1. The coverage line is smooth 2. the number is about 2450~3800 3. EFMUSATRD_CARBONEFF is added since 2020-12-31
Field check (announce_datetime)	1. Missing values 2. Some values are later than effective date 3. Some values are far away from fiscal_period_end_date	**Field check (1-Risk Indices)**	On each date, all the risk factors have the same number of barrids.
Field check (effective_date)	1. No missing values 2. Some values are far away from fiscal_period_end_date	**Coverage check (2-Industries)**	1. The coverage line is smooth 2. the number is about 2450~3800 3. EFMUSATRD_NETRET is removed since 2023-03-16
		Field check (2-Industries)	Each stock is classified into at least one industry
		Coverage check (5-Market)	Each stock has one value on each date
		Field check (5-Market)	All exposure value is 1; Ignore this part

Table 2: Data Quality Check Summary

For each dataset, I reviewed coverage changes over time and performed field-specific checks. The price dataset contains some outliers. The 'announce_datetime' in the earnings dataset has values requiring further verification. While the effective date is more reliable than the announcement date, suspicious values are found far from the fiscal period's end date. The Barra factors data shows good quality. The '5-Market' factors can be ignored as the universe consists solely of US stocks. Additionally, some factors were added and removed over time: EFMUSATRD_NETRET was removed after 2023-03-16, and EFMUSATRD_CARBONEFF was added starting 2020-12-31. In the predictors building process, I retained the industry factor EFMUSATRD_NETRET and removed the style factor EFMUSATRD_CARBONEFF.

In the second step, lots of missing stocks are observed after mapping these datasets via Russell 300 symbology dataset. Price and volume dataset has 67 missing stocks; earnings dataset has 331 missing stocks; barra dataset has 35 stocks because of the missing BARRID in Russell 3000 dataset. So, to create predictors for model training, I took the intersection of the universe. After constructing the predictors, the dataset consists of an average of 2,897 stocks per day, with data spanning from January 3, 2020, to December 29, 2023, after excluding missing values.

Section 2: PREDICTORS

Based on the paper and the available data, I will present five blocks of predictor construction, which I later classified into five types of predictors/factors.

1. Technical signals ("tech")
   The moving average of return with window size (1, 5, 22, 252)
   The moving average of log volume with window size (1, 5, 22, 252)
   Note: the return values have outliners. So, I winsorized them to the range of [-1,1]

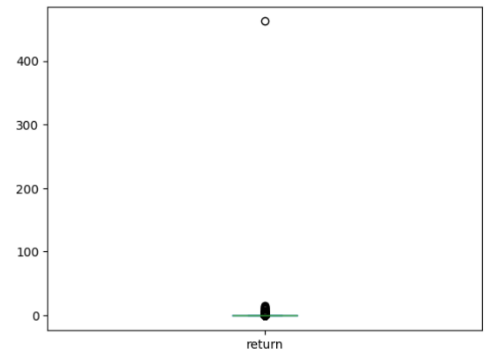
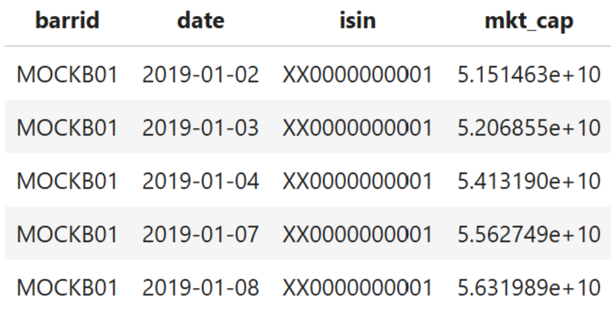


Figure 1: the box plot of return before winsorization

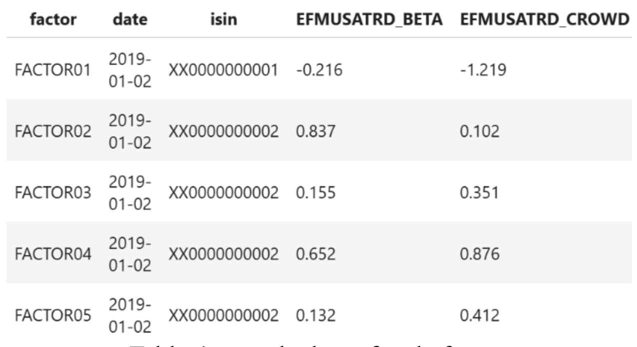
2. Fundamental firm characteristics: market capitalization
   Market capitalization is not a technical factor, so I processed it in a separate block. Initially, I fed the values directly into the model, but the performance was poor due to the scale of this field being much larger than the other predictors. As a result, I used the logarithmic transformation of market capitalization.



barrid	date	isin	mkt_cap
MOCKB01	2019-01-02	XX0000000001	5.151463e+10
MOCKB01	2019-01-03	XX0000000001	5.206855e+10
MOCKB01	2019-01-04	XX0000000001	5.413190e+10
MOCKB01	2019-01-07	XX0000000001	5.562749e+10
MOCKB01	2019-01-08	XX0000000001	5.631989e+10

Table 3: sample data of market capitalization before logarithmic transformation

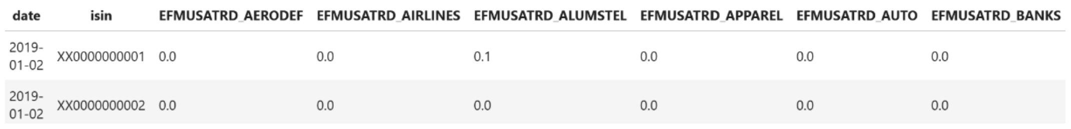
3. Style and industry factors
   According to Section 1, each stock has values for all style factors ("1-Risk Indices") on each date, which I treat as numerical variables.



factor	date	isin	EFMUSATRD_BETA	EFMUSATRD_CROWD
FACTOR01	2019-01-02	XX0000000001	-0.216	-1.219
FACTOR02	2019-01-02	XX0000000002	0.837	0.102
FACTOR03	2019-01-02	XX0000000002	0.155	0.351
FACTOR04	2019-01-02	XX0000000002	0.652	0.876
FACTOR05	2019-01-02	XX0000000002	0.132	0.412

Table 4: sample data of style factors

For industry factors (“2-Industries”), each stock has a value for at least one factors on each date. So, I convert the industry factors into dummy variables, with the multiplier value representing the exposure.



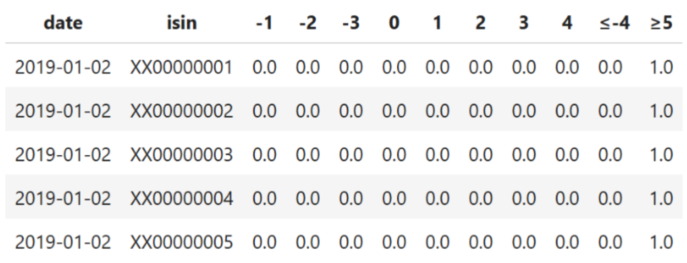
date	isin	EFMUSATRD_AERODEF	EFMUSATRD_AIRLINES	EFMUSATRD_ALUMSTEL	EFMUSATRD_APPAREL	EFMUSATRD_AUTO	EFMUSATRD_BANKS
2019-01-02	XX0000000001	0.0	0.0	0.1	0.0	0.0	0.0
2019-01-02	XX0000000002	0.0	0.0	0.0	0.0	0.0	0.0

Table 5: sample data of industry factors

4. Calendar dates with large effects on trading volume (“calendar”)
   Based on the definition, four dummy variables are created for early exchange closing days (July 3rd, Black Friday, Christmas Eve, and New Year’s Eve), triple witching, double witching days, and the Russell index rebalancing date.

5. Earnings release schedule (“earnings”)
   I used the effective date as the scheduled earnings release date, strictly following the method outlined in the referenced paper. The only modification is that I utilized only the first 9 dummy variables in my model, as the 10 variable is a linear combination of the first 9.

   The paper states:
   “We construct 10 categorical dummy variables (one-hot encoding) indicating whether the firm has an upcoming earnings release or just had one in the past few days.
   - We first construct the number of days until the next known scheduled earnings release event. For example, a value of zero implies the current day is previously known to have a scheduled release. A negative value means there are no known scheduled events in the future and indicates how many days since the last event.
   - We convert this number into 10 dummy variables of categorical bins: ≤ −4, −3, −2, −1, 0, 1, 2, 3, 4, ≥ 5. The data source is the Capital IQ Key Developments dataset. (10 predictors.)”



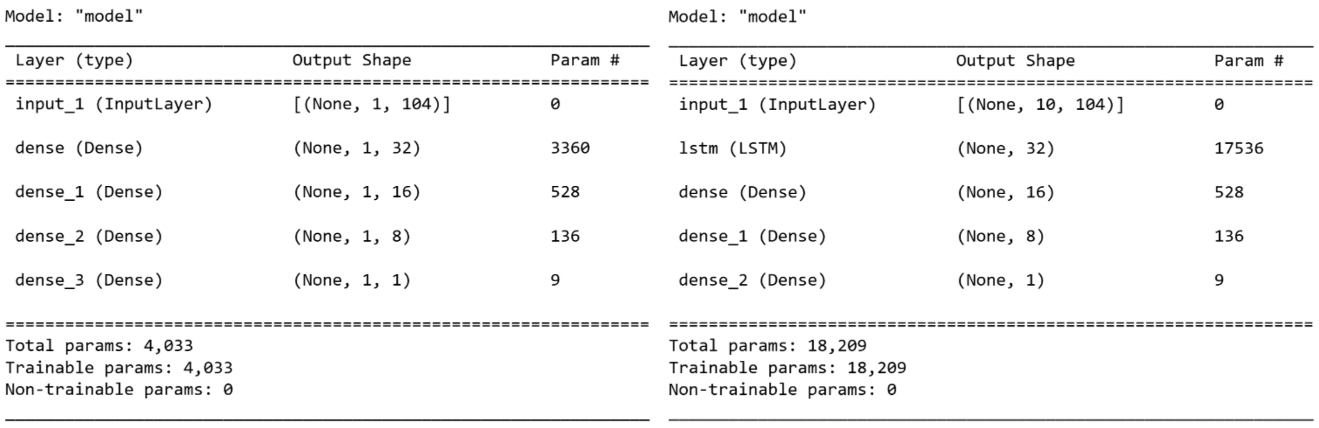
date	isin	-1	-2	-3	0	1	2	3	4	≤-4	≥5
2019-01-02	XX00000001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2019-01-02	XX00000002	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2019-01-02	XX00000003	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2019-01-02	XX00000004	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2019-01-02	XX00000005	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Table 6: sample data of earning release schedule

To create the predictors, I used ISIN and date as the grouping key for calculating returns, as there are no missing values in the symbology mapping dataset. Data quality checks comparing BARRID and ISIN were performed in the notebook '5. DQ – idtype.ipynb,' with ISIN having fewer cases to verify. Given the time constraints, I have chosen to use ISIN in this report.

Section 3: BASE MODELS

Using the paper’s model as a benchmark, I built dense and LSTM models, with their structures shown in Chart 1. To fine-tune the models, early stopping and a learning rate reducer were applied during training. Unlike traditional n-fold cross-validation, sliding window cross-validation was used to evaluate time-series model performance, thereby avoiding look-ahead bias. The sliding window starts from 2020-01-03 and shifts by 60 days for each validation. In each window, 200 days are used for training and 60 days for validation, and the testing data spans from 2022-03-25 to 2023-12-29.



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1, 104)]          0

 dense (Dense)               (None, 1, 32)             3360

 dense_1 (Dense)             (None, 1, 16)             528

 dense_2 (Dense)             (None, 1, 8)              136

 dense_3 (Dense)             (None, 1, 1)              9

=================================================================
Total params: 4,033
Trainable params: 4,033
Non-trainable params: 0
_________________________________________________________________
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 10, 104)]         0

 lstm (LSTM)                 (None, 32)                17536

 dense (Dense)               (None, 16)                528

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 18,209
Trainable params: 18,209
Non-trainable params: 0
_________________________________________________________________
```

Chart 1: Left: Dense Model Structure; Right: LSTM Model Structure

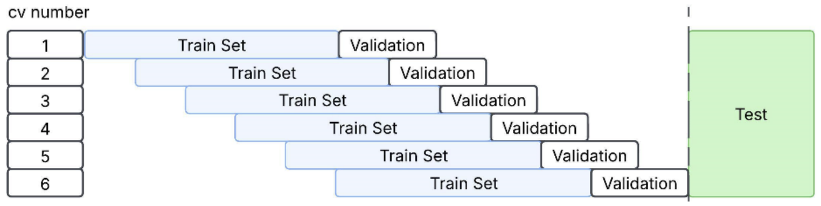
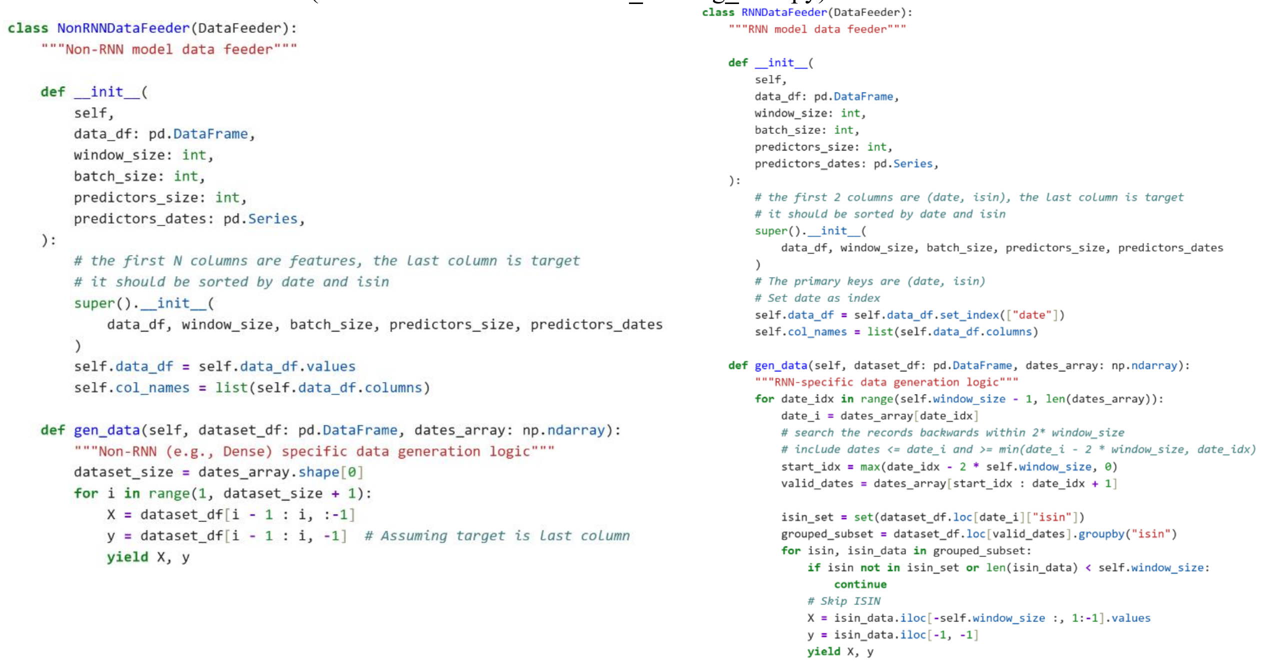


Chart 2: Cross-Validation Dataset Split

Although the paper does not specify how data is fed to the models, I implemented separate data feeders for the dense and LSTM models. To avoid look-ahead bias, I sorted the dataframe by date and shifted all the features by one day forward, ensuring that earlier data is used for training before later data.

For Dense model, the data feeder is straightforward, following the time order from the earliest to the latest date, with each data point having a window size of 1. (NonRNNDataFeeder from model_training_utils.py)
For LSTM mode, the data feeder is more complex. For the LSTM model, the data feeder is more complex. For each date, it looks back a fixed number of days equal to the window size, and then delivers a data point with a window size of 10 for each stock. (RNNDataFeeder from model_training_utils.py)



```python
class NonRNNDataFeeder(DataFeeder):
    """Non-RNN model data feeder"""

    def __init__(
        self,
        data_df: pd.DataFrame,
        window_size: int,
        batch_size: int,
        predictors_size: int,
        predictors_dates: pd.Series,
    ):
        # the first N columns are features, the last column is target
        # it should be sorted by date and isin
        super().__init__(
            data_df, window_size, batch_size, predictors_size, predictors_dates
        )
        self.data_df = self.data_df.values
        self.col_names = list(self.data_df.columns)

    def gen_data(self, dataset_df: pd.DataFrame, dates_array: np.ndarray):
        """Non-RNN (e.g., Dense) specific data generation logic"""
        dataset_size = dates_array.shape[0]
        for i in range(1, dataset_size + 1):
            X = dataset_df[i - 1 : i, :-1]
            y = dataset_df[i - 1 : i, -1]  # Assuming target is last column
            yield X, y
```

```python
class RNNDataFeeder(DataFeeder):
    """RNN model data feeder"""

    def __init__(
        self,
        data_df: pd.DataFrame,
        window_size: int,
        batch_size: int,
        predictors_size: int,
        predictors_dates: pd.Series,
    ):
        # the first 2 columns are (date, isin), the last column is target
        # it should be sorted by date and isin
        super().__init__(
            data_df, window_size, batch_size, predictors_size, predictors_dates
        )
        # The primary keys are (date, isin)
        # Set date as index
        self.data_df = self.data_df.set_index(["date"])
        self.col_names = list(self.data_df.columns)

    def gen_data(self, dataset_df: pd.DataFrame, dates_array: np.ndarray):
        """RNN-specific data generation logic"""
        for date_idx in range(self.window_size - 1, len(dates_array)):
            date_i = dates_array[date_idx]
            # search the records backwards within 2* window_size
            # include dates <= date_i and >= min(date_i - 2 * window_size, date_idx)
            start_idx = max(date_idx - 2 * self.window_size, 0)
            valid_dates = dates_array[start_idx : date_idx + 1]

            isin_set = set(dataset_df.loc[date_i]["isin"])
            grouped_subset = dataset_df.loc[valid_dates].groupby("isin")
            for isin, isin_data in grouped_subset:
                if isin not in isin_set or len(isin_data) < self.window_size:
                    continue
                # Skip ISIN
                X = isin_data.iloc[-self.window_size :, 1:-1].values
                y = isin_data.iloc[-1, -1]
                yield X, y
```

Chart 3: Data feeder codes screenshots

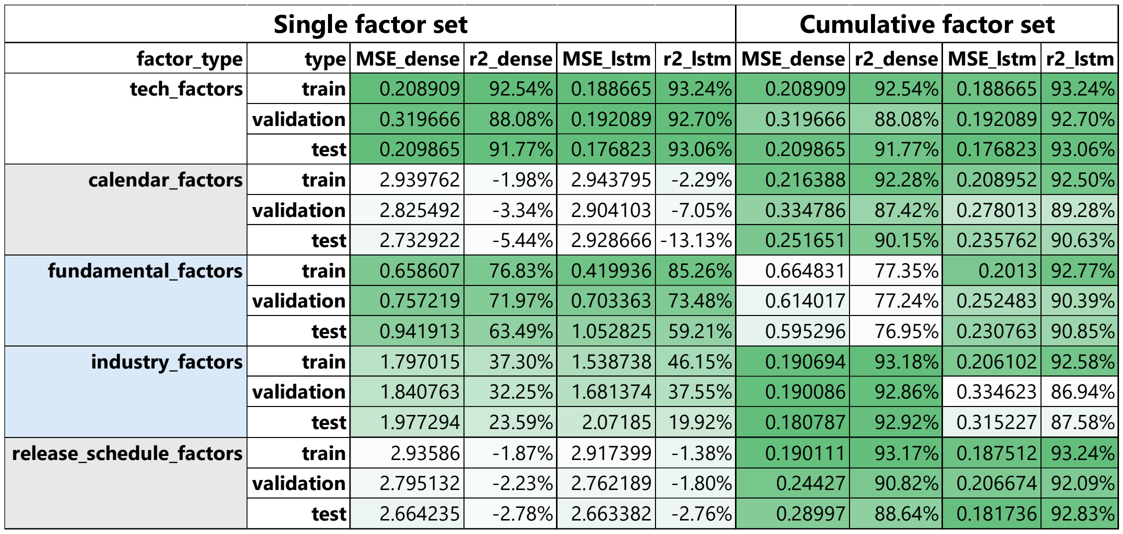
Section 4: ANALYSIS OF BASE MODEL RESULTS

Part 1 Model Metrics

In line with the approach in the paper, I analyzed the results by testing each predictors set and cumulative sets. There are 5 factor sets.

1. Tech factors include price return and volume.
2. Calendar factors are the known early close dates, witch dates and holidays.
3. Fundamental factors include market capitalization and Barra style factors.
4. Industry factors consist of Barra industry factors.

Release schedule factors are indicators representing the number of days before and after the closest earnings release.



Single factor set						Cumulative factor set			
factor_type	type	MSE_dense	r2_dense	MSE_lstm	r2_lstm	MSE_dense	r2_dense	MSE_lstm	r2_lstm
tech_factors	train	0.208909	92.54%	0.188665	93.24%	0.208909	92.54%	0.188665	93.24%
	validation	0.319666	88.08%	0.192089	92.70%	0.319666	88.08%	0.192089	92.70%
	test	0.209865	91.77%	0.176823	93.06%	0.209865	91.77%	0.176823	93.06%
calendar_factors	train	2.939762	-1.98%	2.943795	-2.29%	0.216388	92.28%	0.208952	92.50%
	validation	2.825492	-3.34%	2.904103	-7.05%	0.334786	87.42%	0.278013	89.28%
	test	2.732922	-5.44%	2.928666	-13.13%	0.251651	90.15%	0.235762	90.63%
fundamental_factors	train	0.658607	76.83%	0.419936	85.26%	0.664831	77.35%	0.2013	92.77%
	validation	0.757219	71.97%	0.703363	73.48%	0.614017	77.24%	0.252483	90.39%
	test	0.941913	63.49%	1.052825	59.21%	0.595296	76.95%	0.230763	90.85%
industry_factors	train	1.797015	37.30%	1.538738	46.15%	0.190694	93.18%	0.206102	92.58%
	validation	1.840763	32.25%	1.681374	37.55%	0.190086	92.86%	0.334623	86.94%
	test	1.977294	23.59%	2.07185	19.92%	0.180787	92.92%	0.315227	87.58%
release_schedule_factors	train	2.93586	-1.87%	2.917399	-1.38%	0.190111	93.17%	0.187512	93.24%
	validation	2.795132	-2.23%	2.762189	-1.80%	0.24427	90.82%	0.206674	92.09%
	test	2.664235	-2.78%	2.663382	-2.76%	0.28997	88.64%	0.181736	92.83%

Table 7: Dense and LSTM models metrics

Based on the average metrics values of the LSTM model, tech factors contribute the most among all predictor sets, achieving the lowest mean squared error and the highest R-squared values within the single-factor testing. The fundamental factor set is the second most important contributor, although it shows significant overfitting. In single-factor testing, the model performs poorly with calendar, release schedule, and industry factors. However, in cumulative factor set testing, release schedule and fundamental factors enhance performance, while calendar and industry factors degrade it.

Compared to the Dense model, the LSTM model outperforms across all datasets except for calendar factors in single-factor testing. Both models struggle to learn from calendar factors, which may be due to improper feature representation. Additionally, in cumulative factor set testing, the Dense model significantly outperforms the LSTM model with the industry factor set, with LSTM even performing worse in this case. Based on these findings, I make the following deductions:

1. There may be interactions between release schedule and tech factors.
2. The LSTM model struggles to learn from calendar, release schedule, and industry factors effectively. Model structure adjustments or enhanced feature engineering may be necessary.
3. Adding dense layers may help improve learning from industry factors.

Part 2 Feature Importance

To identify the key features in these models, I used permutation methods to determine feature importance. Permutation feature importance measures the change in metrics after randomly permuting the feature values. In this case, it evaluates the increase in mean squared error.

In the primary contributor - tech factor set (technical indicators), the log volume features are the most important, while price return contributes minimally. In the single-factor set testing, LSTM models show a larger increase in mean squared error than Dense models for fundamental and industry factors. Calendar factors show no significant importance relative to the baseline mean squared error in both Dense and LSTM models. Regarding earning release schedule factors, at most 3 out of 9 features demonstrate importance.

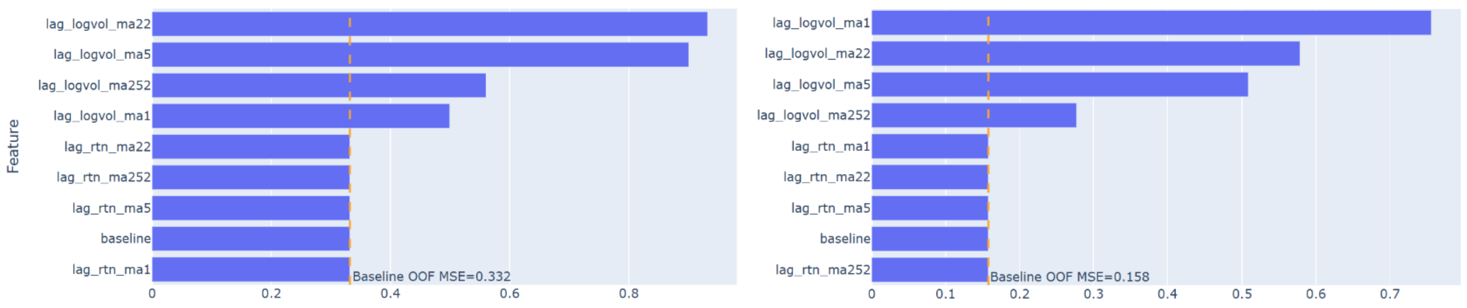


Chart 4: Left: Dense Model with tech factor set; Right: LSTM Model with tech factor set

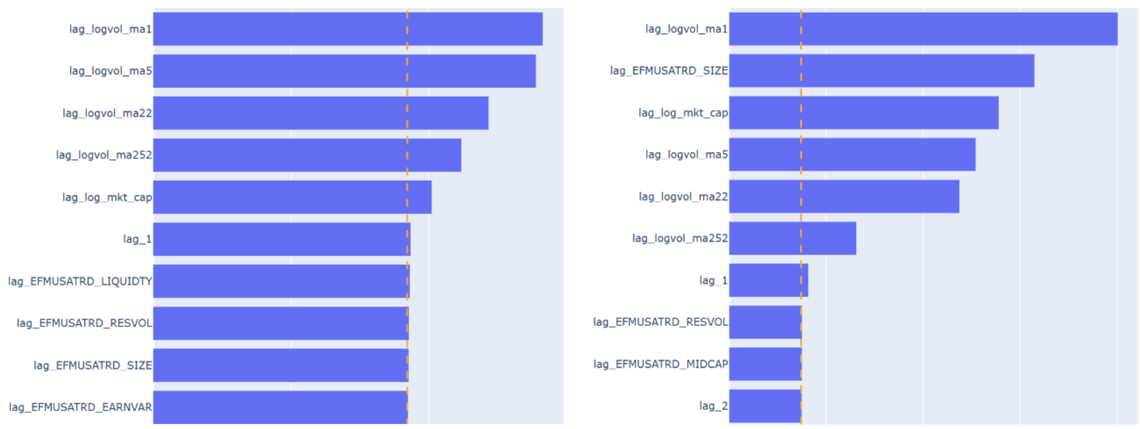


Chart 5: Top 10 important predictors
Left: Dense Model with all the factor sets; Right: LSTM Model with all the factor sets

In cumulative factor testing, Dense models primarily capture signals from log volume across all tests, from technical set to full sets. In the full set test, market capitalization and the lag of 1 day of earning release also contribute slightly. In addition to these key predictors, LSTM models place more importance on size and market capitalization.

In summary, LSTM models capture more information from these datasets than Dense models. However, improvements are needed in
1) dummy variables (calendar and earning release factors),
2) sparse variables (industry factors), and
3) a large set of continuous variables (fundamental factors).

Section 5: ENHANCEMENT

Part 1 Predictor Enhancement

Based on previous analyses, I revisited the database to incorporate features from multiple time zones, refined the representation of calendar-related variables, and applied aggregation to preserve industry classification information.

1. **Tech factors**: This includes price return and volume.
   Both Dense and LSTM models fail to extract meaningful insights from return features, as most return values are close to zero. To align with asset pricing assumptions that log returns follow a normal distribution, I transformed the return predictors into log return values.

2. **Calendar Factors**: These include early close dates, witch dates, and holidays.
   In Sections 3 and 4, I used dummy variables to indicate these calendar events, shifting them by one day to prevent future data leakage. However, since we know these dates in advance, there is no need for shifting, and we can utilize the most recent data without risk of leakage.
   Additionally, I enhanced the dummy variables by adding indicators for dates before and after these special dates. For example, for early close dates, I created two indicators:
   - Indicator(x) = 1 if x is on an early close date or within two days after.
   - Indicator(x) = 1 if x is within two days before the early close date.
     I applied this approach to all calendar factors, merging the 'three witch dates' and four witch dates' into one category to reduce the number of dummy variables.

3. **Fundamental factors:** This includes market capitalization and Barra style factors.
   To capture information across multiple time zones, I added the 5-day and 22-day moving averages of each Barra style factor and market capitalization.

4. **Industry Factors**: This includes Barra industry factors.
   To address the sparse variable issue, I aim to retain industry division information while reducing sparsity. The industry factors consist of both time series (exposure values) and cross-sectional (industry classification) features. For the time series, I combine the exposure values into a single feature. For stocks belonging to multiple industries, I retain only the one with the highest exposure. For the cross-sectional features, I aggregate log volume data (1-, 5-, and 22-day moving averages) by industry classification to preserve the industry information.

5. **Earning Release Schedule Factors**: These represent the number of days before and after earnings releases. In Section 4, I observed that only 1-2 of these dummy variables were informative. To reduce the number of variables, I merged some into broader categories. Since release dates are known at least three days in advance, there is no need to shift these variables. The revised categories are as follows:
   - From ['lag_-1', 'lag_-2', 'lag_-3', 'lag_0', 'lag_1', 'lag_2', 'lag_3', 'lag_-4', 'lag_≥5']
   - To ['≤-3', '-1_2', '0', '1_2', '≥3']

   Additionally, I introduced a new feature based on the 'is_adj_date' field in the price and volume dataset. Corporate actions, which adjust price and volume data, are publicly disclosed at least three days in advance. Therefore, similar to calendar factors, I added two dummy variables: 'before_adj_date' and 'on_after_adj_date', representing the dates before and on/after the adjustment date.

Single factor set								Cumulative factor set							
				no change in industry		aggregation indutrsy				no change in industry		aggregation indutrsy		no change in industry	
factor_type	type	MSE_v1	r2_v1	MSE_v2	r2_v2	MSE_v3	r2_v3	MSE_v1	r2_v1	MSE_v2	r2_v2	MSE_v3	r2_v3	MSE_v4	r2_v4
tech_factors	train	0.188665	93.24%	0.188753	93.24%			0.188665	93.24%	0.188753	93.24%				
	validation	0.192089	92.70%	0.191848	92.71%			0.192089	92.70%	0.191848	92.71%				
	test	0.176823	93.06%	0.177148	93.05%			0.176823	93.06%	0.177148	93.05%				
calendar_factors	train	2.943795	-2.29%	2.928466	-1.73%			0.208952	92.50%	0.194635	93.04%				
	validation	2.904103	-7.05%	2.813014	-3.61%			0.278013	89.28%	0.212802	91.98%				
	test	2.928666	-13.13%	2.69486	-3.97%			0.235762	90.63%	0.191246	92.51%				
fundamental_factors	train	0.419936	85.26%	0.337659	88.03%			0.2013	92.77%	0.185809	93.38%			0.183675	93.46%
	validation	0.703363	73.48%	0.633209	76.24%			0.252483	90.39%	0.195223	92.63%			0.192026	92.76%
	test	1.052825	59.21%	1.076532	58.32%			0.230763	90.85%	0.182545	92.87%			0.178453	93.02%
industry_factors	train	1.538738	46.15%	1.523516	46.71%	2.214445	22.67%	0.206102	92.58%	0.186221	93.36%	0.191401	93.18%	0.18384	93.45%
	validation	1.681374	37.55%	1.694847	37.06%	2.192936	18.82%	0.334623	86.94%	0.196352	92.58%	0.196997	92.55%	0.190836	92.80%
	test	2.07185	19.92%	2.121662	17.98%	2.111926	18.45%	0.315227	87.58%	0.186856	92.70%	0.192868	92.46%	0.176661	93.09%
release_schedule_factors	train	2.917399	-1.38%	2.924034	-1.61%			0.187512	93.24%	0.17946	93.59%	0.199659	92.83%	0.182792	93.48%
	validation	2.762189	-1.80%	2.762343	-1.79%			0.206674	92.09%	0.189331	92.82%	0.352938	86.68%	0.19704	92.50%
	test	2.663382	-2.76%	2.666896	-2.89%			0.181736	92.83%	0.176203	93.10%	0.190422	92.54%	0.180737	92.91%

Table 8: LSTM models metrics
v1: no enhancement; v2: dataset enhancement except industry factor;
v3: dataset enhancement; v4: model structure enhancement

Version 2 (v2):
All factor sets were updated as mentioned, except for industry factors. Calendar factors show improved performance, while other factor sets show no significant improvement according to the metrics. Furthermore, the fundamental factor set experiences more pronounced overfitting with the new features. Finally, the LSTM model incorporating all factor sets demonstrates slightly better performance.
Version 3 (v3):
All factor sets were updated as described. Both the single set and cumulative set tests performed worse, likely due to incomplete capture of industry features. Therefore, I did not include the industry factor changes in the part 2 enhancement test.

Part 2 Model Structure Enhancement
Based on the results in Section 4, I will address the last issue - a large set of continuous variables (fundamental factors) through improvements in the model structure. To handle this, I reduce their dimensions for more efficient processing in the LSTM layers. To capture non-linear relationships, I implement an LSTM autoencoder to extract features and reduce dimensionality. After training, I keep the encoder and feed the processed data—along with other factors—into the LSTM model.

1). **Encoder non-trainable**:
When the encoder is non-trainable, it serves as a filter for fundamental factors before they enter the LSTM model. To prevent data leakage, I use only historical data prior to the LSTM training dataset, which prevents the filter from updating with new batches. While this avoids data leakage, it also means the filter doesn't incorporate the latest data. The benefit is that more information is preserved from the features, but the drawback is the lack of up-to-date data in the filter.
2). **Encoder trainable**:
Since the optimization objective of the autoencoder differs from that of the LSTM model, the encoder essentially provides initial values based on pretraining and serves as an additional layer for the LSTM.

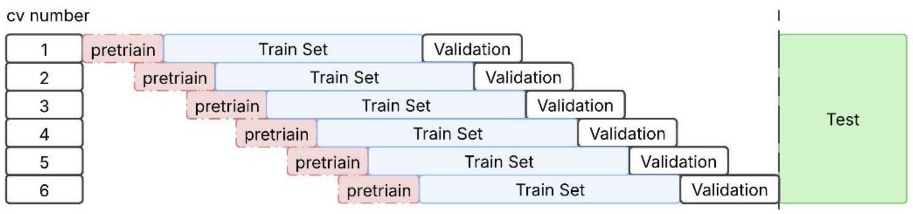
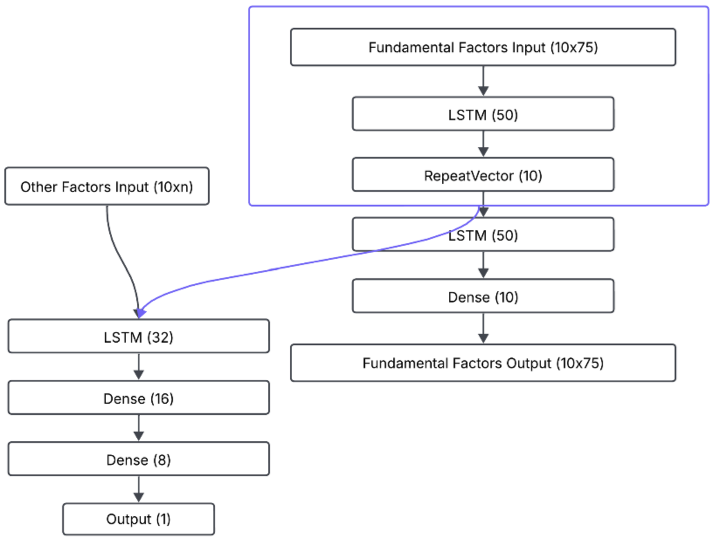


Chart 6: Cross-Validation Dataset Split


Fundamental Factors Input (10x75)
LSTM (50)
RepeatVector (10)
Other Factors Input (10xn)
LSTM (50)
Dense (10)
Fundamental Factors Output (10x75)
LSTM (32)
Dense (16)
Dense (8)
Output (1)
Chart 7: LSTM with Autoencoder Pretraining Structure

Non-Trainable				
type	**mean_MSE**	**mean_r2**	**std_MSE**	**std_r2**
train	0.190037	93.19%	0.00874	0.60%
validation	0.193529	92.65%	0.02572	0.97%
test	0.178861	92.98%	0.00647	0.26%
Trainable				
train	0.188397	93.25%	0.00802	0.59%
validation	0.193984	92.62%	0.02528	0.96%
test	0.180179	92.93%	0.00760	0.30%

CV number	Train				Test			
	nontrainable		**trainable**		**nontrainable**		**trainable**	
	MSE	**r2**	**MSE**	**r2**	**MSE**	**r2**	**MSE**	**r2**
1	0.17573	94.17%	0.17502	94.19%	0.19201	92.45%	0.19568	92.31%
2	0.19425	93.44%	0.19156	93.53%	0.17686	93.07%	0.17690	93.07%
3	0.19446	93.38%	0.19323	93.43%	0.17575	93.11%	0.17725	93.05%
4	0.20094	92.74%	0.19804	92.85%	0.17643	93.08%	0.17728	93.05%
5	0.18543	92.89%	0.18442	92.93%	0.17534	93.13%	0.17655	93.08%
6	0.18942	92.51%	0.18811	92.56%	0.17678	93.07%	0.17741	93.04%

Table 9: Metrics of technical and fundamental factors with model structure enhancement
Left: the average metrics; Right: the metrics of each cross-validation test

The performance difference between the trainable and non-trainable models is minimal. As shown in Table 9, the non-trainable model slightly outperforms the trainable model in testing, while the trainable model performs slightly better in training. To ensure the use of the latest data, I prefer to use trainable models for further exploration.

I tested cumulative factor sets and other factor combinations using the trainable setting. With the model structure enhancement, the MSE of the cumulative factor set test (technical, calendar, and fundamental factors) improved from 0.182545 to 0.178453, a 2.24% increase. However, adding earnings release schedule factors raised the MSE in testing to 0.190422, which deviates significantly from the initial result. The earnings release schedule factors, while improving performance, were offset by the industry factor, suggesting strong interactions driven by industry factors (see MSE_v1, v2, and v4 in Table 8).

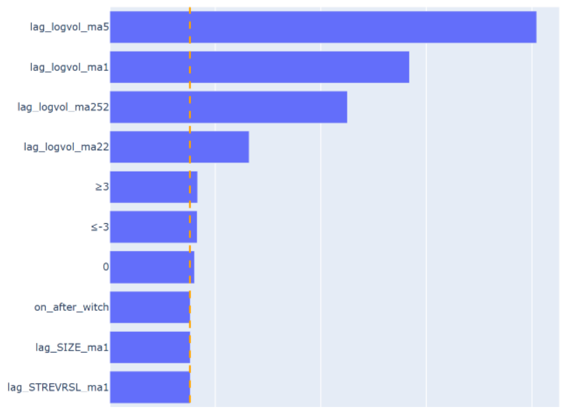
factor_type	**type**	**mean_MSE**	**mean_r2**	**std_MSE**	**std_r2**
technical & fundamental factors	**train**	0.188397	93.25%	0.80%	0.59%
	validation	0.193984	92.62%	2.53%	0.96%
	test	0.180179	92.93%	0.76%	0.30%
technical & calendar & fundamental factors	**train**	0.184442	93.43%	0.66%	0.51%
	validation	0.195411	92.63%	3.50%	1.18%
	test	0.178686	93.01%	0.82%	0.32%
technical & calendar & fundamental & earning release factors	**train**	0.177302	93.67%	0.60%	0.50%
	validation	0.182525	93.08%	2.41%	0.97%
	test	0.168729	93.38%	0.61%	0.25%

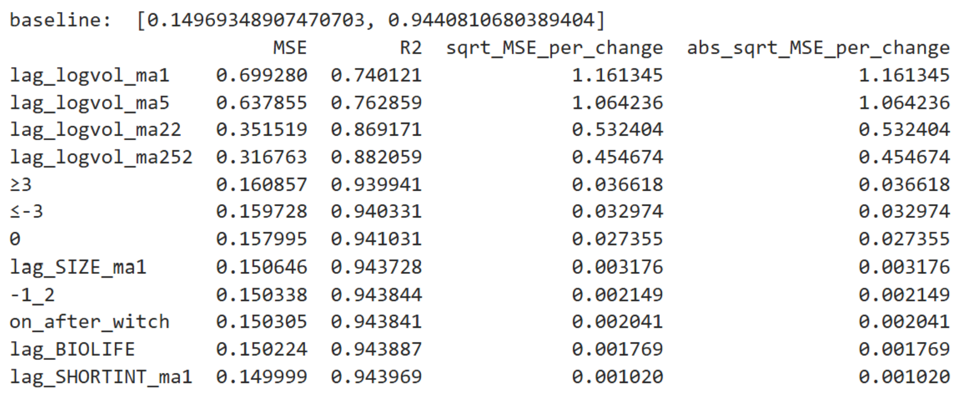
Table 10: Metrics of tests with factor set combinations excluding industry factors, using a trainable autoencoder with the fundamental factor set

To minimize the noise from industry factors, I test with factor set combinations excluding industry factors. This adjustment shows that both calendar and earnings release factors help improve performance, resulting in a 4.58% improvement—from 0.1768 in the initial LSTM model to 0.1687 (Table 10). This confirms the utility of incorporating the autoencoder in the model.

Part 3 Final Model Analysis

Based on the metrics, I decide to select the LSTM model with technical, calendar, fundamental, and earnings release factors, and the autoencoder for fundamental factors. The model, trained on the last cross-validation set, achieved an MSE of 0.1497 and an R-squared of 94.4% in validation. Key factors influencing the model include log volume and earnings release schedule. Additionally, the size and short interest factors from Barra style, along with Biolife from Barra industry, contribute to improved performance. The LSTM model also assigns weights to the witch dates (on a witch date or within two days after).





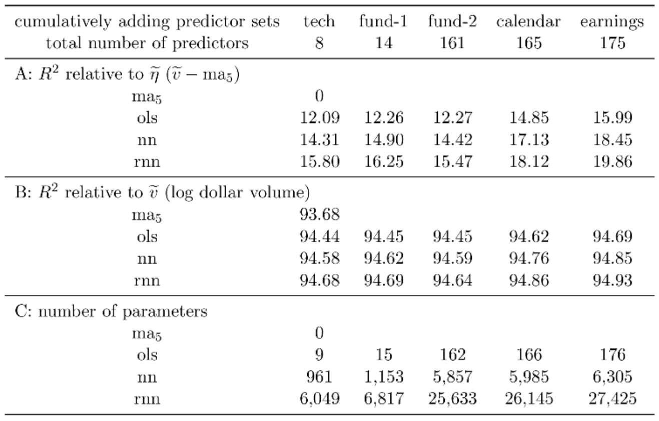
```
baseline:  [0.14969348907470703, 0.9440810680389404]
                       MSE        R2  sqrt_MSE_per_change  abs_sqrt_MSE_per_change
lag_logvol_ma1    0.699280  0.740121             1.161345                 1.161345
lag_logvol_ma5    0.637855  0.762859             1.064236                 1.064236
lag_logvol_ma22   0.351519  0.869171             0.532404                 0.532404
lag_logvol_ma252  0.316763  0.882059             0.454674                 0.454674
≥3                0.160857  0.939941             0.036618                 0.036618
≤-3               0.159728  0.940331             0.032974                 0.032974
0                 0.157995  0.941031             0.027355                 0.027355
lag_SIZE_ma1      0.150646  0.943728             0.003176                 0.003176
-1_2              0.150338  0.943844             0.002149                 0.002149
on_after_witch    0.150305  0.943841             0.002041                 0.002041
lag_BIOLIFE       0.150224  0.943887             0.001769                 0.001769
lag_SHORTINT_ma1  0.149999  0.943969             0.001020                 0.001020
```

Chart 8: Feature Importance in the Final model
Left: MSE after shuffling each feature; Right: Features with an MSE increase greater than 0.1% after shuffling.

Section 6: COMPARISON WITH PREVIOUS RESEARCH

The paper claims that their LSTM model achieves R-squared values of 94.68% with only the technical predictors and 94.93% with all predictor sets. They also note that, within their dataset and time range, “[t]he five-day moving average predicts log dollar volume with an R^2 of 93.68%, outperforming the one-day lag (92.53%), 22-day moving average (92.60%), and 252-day moving average (86.12%).”

To ensure comparability, I computed the R-squared values for the lag moving averages in my universe. I excluded MSE as their 5-day moving average in Table 2, Panel B, is 0.437, while mine is 0.203, suggesting differences in scale. The highest R-squared value in my tests for the five-day moving average is 92.30%. (notebook: 7. Validation of Model Effectiveness) Using the same model structure, the technical predictor set yields an R-squared of 93.05%, surpassing the best moving average method. Their paper shows an improvement from 93.68% to 94.93%, a 1.25% increase, while my results show an improvement from 92.30% to 93.05%, a 0.81% increase. This confirms that I have successfully replicated this part of the analysis.



cumulatively adding predictor sets	tech	fund-1	fund-2	calendar	earnings
total number of predictors	8	14	161	165	175
A: R^2 relative to $\widetilde{\eta}$ $(\widetilde{v} - \text{ma}_5)$					
ma_5	0				
ols	12.09	12.26	12.27	14.85	15.99
nn	14.31	14.90	14.42	17.13	18.45
rnn	15.80	16.25	15.47	18.12	19.86
B: R^2 relative to $\widetilde{v}$ (log dollar volume)					
ma_5	93.68				
ols	94.44	94.45	94.45	94.62	94.69
nn	94.58	94.62	94.59	94.76	94.85
rnn	94.68	94.69	94.64	94.86	94.93
C: number of parameters					
ma_5	0				
ols	9	15	162	166	176
nn	961	1,153	5,857	5,985	6,305
rnn	6,049	6,817	25,633	26,145	27,425

Table from Trading Volume Alpha: Prediction accuracy

The paper also states that “[m]arket-wide calendar events are quite effective in capturing volume changes” and “[s]cheduled earning announcements also add a sizable gain in prediction accuracy”. In contrast, my training results show that calendar events negatively impacted performance, while scheduled earnings predictors did improve accuracy. These differences may stem from variations in our datasets, time ranges, or potential interactions between the fundamental factors and calendar predictors.

In addition to the base models from the paper, I enhanced both the predictors and the model structure. For predictors, I included features from multiple time zones and refined the representation of calendar-related variables. I also utilized an LSTM autoencoder to reduce and enhance the feature extraction from the large set of continuous variables. In my final model, which incorporates enhanced predictors (technical, calendar, and earnings release schedule factors) and a trainable autoencoder with the fundamental factor set, the MSE is 0.168729, and the R-squared is 93.38%, an improvement over my base LSTM model, which had an MSE of 0.176823 and R-squared of 93.05%.

Section 7: LIMITATIONS, AND OTHER DATA CONSIDERATIONS

Part 1 Limitation

The limitations primarily lie in dataset verification, feature engineering, and model enhancement techniques.

- **Dataset Verification**: Due to time and resource constraints, further verification of the earnings dataset is needed, particularly addressing BARRID and ISIN mapping issues. With additional verification, the earnings dataset could offer more accurate insights, and BARRID may prove to be a better ID type with more data points for testing.

- **Industry Factor Enhancement**: My approach does not incorporate industry information across all features. Additionally, I observed potential interactions between calendar predictors and other features that require further investigation.

- **Model Enhancement**: I developed an LSTM autoencoder to extract features and reduce dimensionality. However, the trainable setting, which uses the latest data points without retaining the nature of the predictors, slightly underperformed compared to the non-trainable setting. Updating the autoencoder during training while keeping later layers of the encoder non-trainable might yield better results. Additionally, exploring the use of GRU layers in the decoders could be a promising alternative. Due to time and GPU resource limitations, this was not implemented in the report.

Beyond my current implementation, I also see potential for improvement in the following areas.

- **Model Complexity**: The final model includes many inputs, some of which may introduce noise. With additional time, I would further investigate these predictors, examine interactions between categorical and continuous variables, and remove those contributing to noise.

- **Alternative Approaches**: Beyond the autoencoder, alternative approaches such as conditional neural networks or using separate factor sets to generate forecasts, followed by an ensemble algorithm to select the final result, could also be explored.

- **Feature Importance**: The permutation method does not capture interactions between features effectively. With more time, I would experiment with methods from the timeSHAP library to provide a more comprehensive understanding of feature importance.

Part 2 Other Data Considerations

Market data such as bid-ask spread and related option data, including the implied volatility of call and put options, can aid in forecasting volume. A higher bid-ask spread typically indicates low volume, while a lower spread suggests higher volume. High implied volatility in the options market reflects increased market speculation, signaling potential trading activity in the underlying asset (stock). Additionally, high-frequency volume data, such as intraday data, provides more timely insights.

Besides, alternative data, such as social media for sentiment analysis and credit card data to track consumption patterns, can identify surges in spending in specific areas.

REFERENCE

1. Goyenko, R., Kelly, B. T., Moskowitz, T. J., Su, Y., & Zhang, C. (2024). Trading Volume Alpha. *Social Science Research Network*. https://doi.org/10.2139/ssrn.4802345

2. Molnar, C. (n.d.). 8.5 Permutation Feature Importance | Interpretable Machine Learning. In *christophm.github.io*. https://christophm.github.io/interpretable-ml-book/feature-importance.html#feature-importance

3. *LSTM Feature Importance*. (n.d.). Kaggle.com. https://www.kaggle.com/code/cdeotte/lstm-feature-importance

4. feedzai. (2023, May 31). *GitHub - feedzai/timeshap: TimeSHAP explains Recurrent Neural Network predictions.* GitHub. https://github.com/feedzai/timeshap

APPENDIX

1. Early stopping and learning rate reducer from train_model.py:

```python
reducelr_cb = tf.keras.callbacks.ReduceLROnPlateau(
    monitor="loss", factor=0.5, patience=5, verbose=0, min_delta=1e-2
)
earlystop_cb = tf.keras.callbacks.EarlyStopping(
    monitor="loss", patience=10, restore_best_weights=True
)
```

2. Data loader from model_training_utils.py: Since the dataset is large, I am using filters and column selection to avoid unnecessary memory usage.

```python
def read_data(filename, filters=None, columns=None, folder_path=FOLDER_PATH):
    """read data from parquet file
    columns and filters are useful to save memory
    """
    return pq.read_table(
        f"{folder_path}/{filename}.parquet", filters=filters, columns=columns
    ).to_pandas()
```

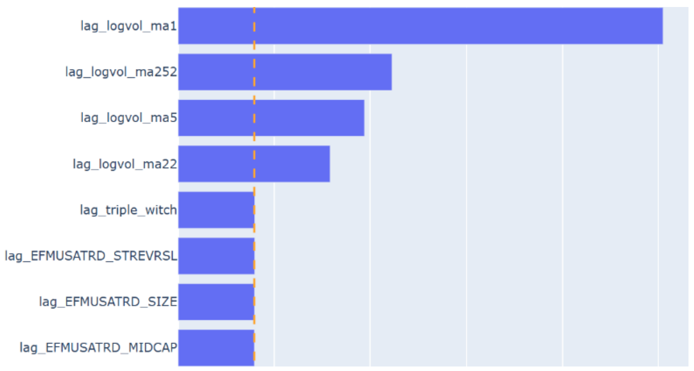
3. Data feeder from model_training_utils.py: The shape of output should match in data generator and model structure. Otherwise, the metrics computation will give wrong values.

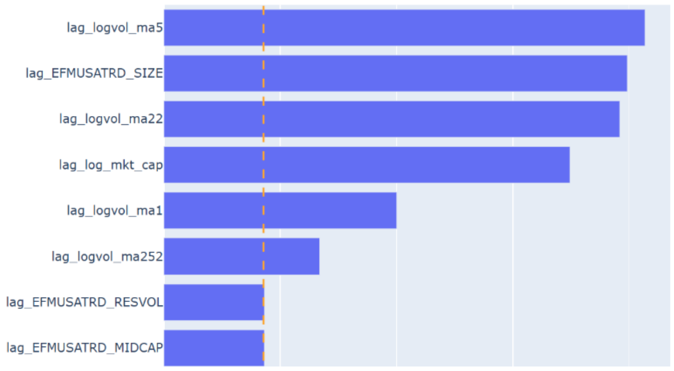
4. GPU setup from train_model.py:

```python
def setup_gpu():
    """setup gpu"""
    physical_gpus = tf.config.list_physical_devices("GPU")
    if physical_gpus:
        try:
            # Check if GPUs have already been set as visible devices
            visible_devices = tf.config.get_visible_devices()
            if not any(device.device_type == "GPU" for device in visible_devices):
                tf.config.set_visible_devices(physical_gpus[0], "GPU")
                # Restrict GPU memory to 8GB for the first GPU
                tf.config.set_logical_device_configuration(
                    physical_gpus[0],
                    # Restrict TensorFlow to only allocate 8GB of memory on the first GPU
                    [tf.config.LogicalDeviceConfiguration(memory_limit=1024 * 8)],
                )
                logical_gpus = tf.config.list_logical_devices("GPU")
                print(
                    len(physical_gpus),
                    "Physical GPUs,",
                    len(logical_gpus),
                    "Logical GPUs",
                )
            else:
                print("GPU devices are already configured, skipping setup.")
        except RuntimeError as e:
            # Virtual devices must be set before GPUs have been initialized
            print(e)

setup_gpu()
```

5. Cumulative testing- fundamental set





Left: Dense Model with fundamental factor set;
Right: LSTM Model with fundamental factor set

6. Standard deviation of base models

Individuel factor set

factor_type	type	MSE_dense	r2_dense	MSE_lstm	r2_lstm
tech_factors	train	0.02309	0.01004	0.009194	0.006
	validation	0.18306	0.06752	0.026039	0.0098
	test	0.058585	0.02294	0.004001	0.0016
calendar_factors	train	0.172713	0.00295	0.16695	0.0058
	validation	0.222857	0.01948	0.270702	0.0747
	test	0.07183	0.02806	0.404487	0.1582
fundamental_factors	train	0.023279	0.01688	0.017269	0.0092
	validation	0.026951	0.01785	0.062457	0.0223
	test	0.040269	0.01557	0.041502	0.0163
industry_factors	train	0.064517	0.02866	0.02932	0.0289
	validation	0.065439	0.03807	0.032173	0.029
	test	0.06607	0.02588	0.079799	0.0315
release_schedule_factors	train	0.169527	0.00262	0.170339	0.0024
	validation	0.189461	0.00707	0.17179	0.0048
	test	0.031557	0.01255	0.042625	0.017

Cumulative factor set

factor_type	type	MSE_dense	r2_dense	MSE_lstm	r2_lstm
tech_factors	train	0.02309	0.01004	0.009194	0.006
	validation	0.18306	0.06752	0.026039	0.0098
	test	0.058585	0.02294	0.004001	0.0016
calendar_factors	train	0.026337	0.01147	0.023422	0.009
	validation	0.291711	0.10916	0.111366	0.0484
	test	0.146917	0.0573	0.070513	0.0289
fundamental_factors	train	1.165876	0.38797	0.015986	0.0084
	validation	1.047038	0.38784	0.070898	0.0275
	test	1.016893	0.39172	0.047171	0.0195
industry_factors	train	0.006907	0.00623	0.016746	0.0096
	validation	0.024226	0.00841	0.230248	0.0998
	test	0.010182	0.00407	0.175173	0.0686
release_schedule_factors	train	0.01664	0.01021	0.010799	0.0074
	validation	0.139502	0.05216	0.043399	0.0187
	test	0.292763	0.11427	0.013107	0.0054

Left: standard deviation of metrics in single factor set tests;
Right: standard deviation of metrics in cumulative factor set tests