# G-Research Crypto Forecasting

Ricky Chan Tsz Wai

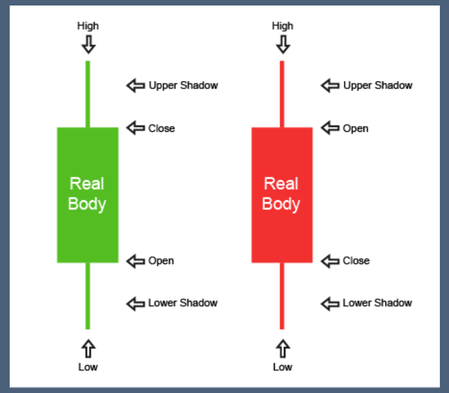
2024 04 27

# Our Task

- For a certain asset at time t, predict its Weighted Average log returns of 15 minutes.

$$R^a(t) = log(P^a(t+16) \,/\, P^a(t+1))$$

- Given the OCHLV, VWAP (K-line Data)
- And trade count within that bar.(L2 Data)

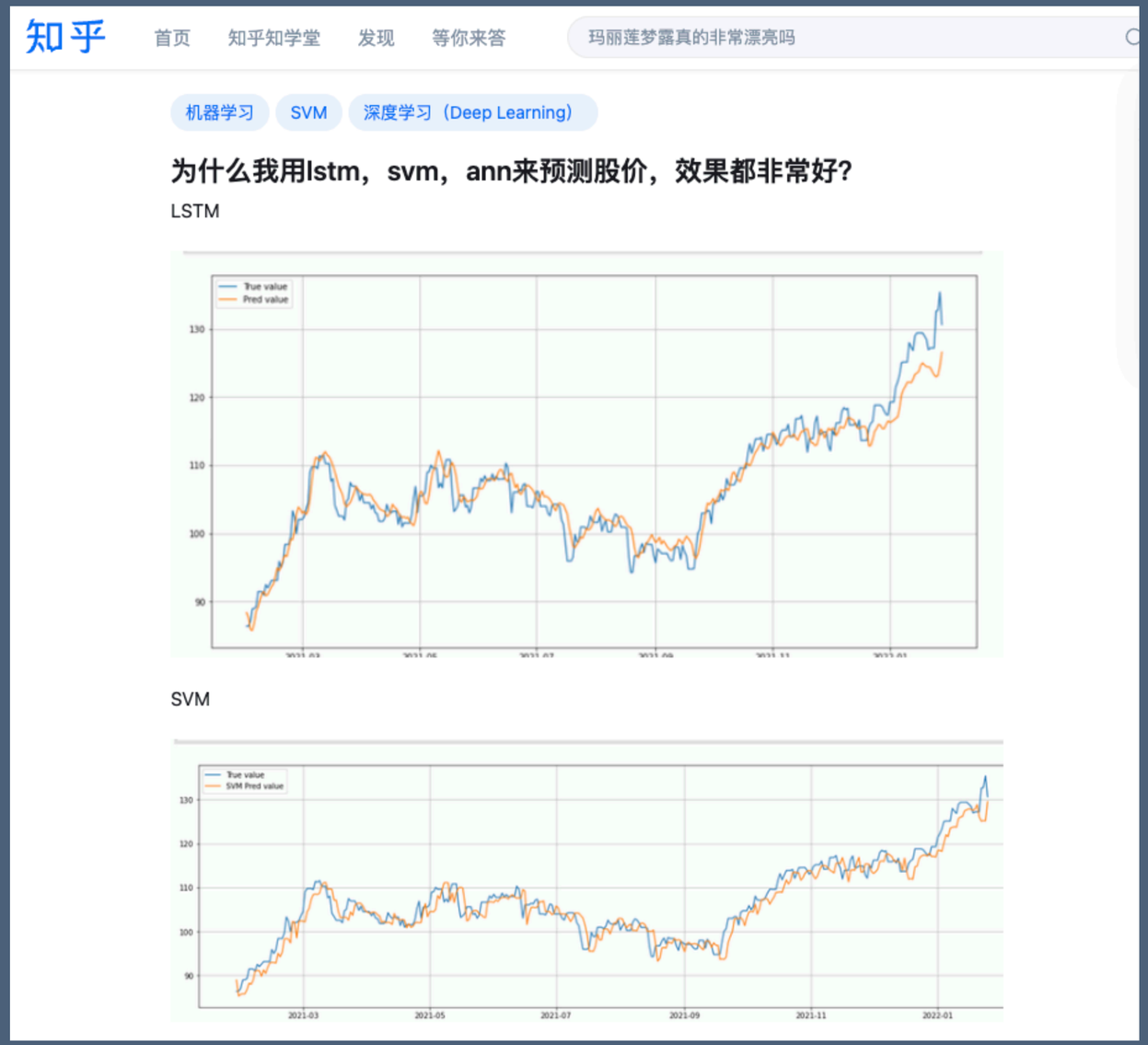



$$M(t) = \frac{\sum_a w^a R^a(t)}{\sum_a w^a}$$

$$\beta^a = \frac{\langle M \cdot R^a \rangle}{\langle M^2 \rangle}$$

$$\text{Target}^a(t) = R^a(t) - \beta^a M(t)$$

# Why don’t we directly predict the price?



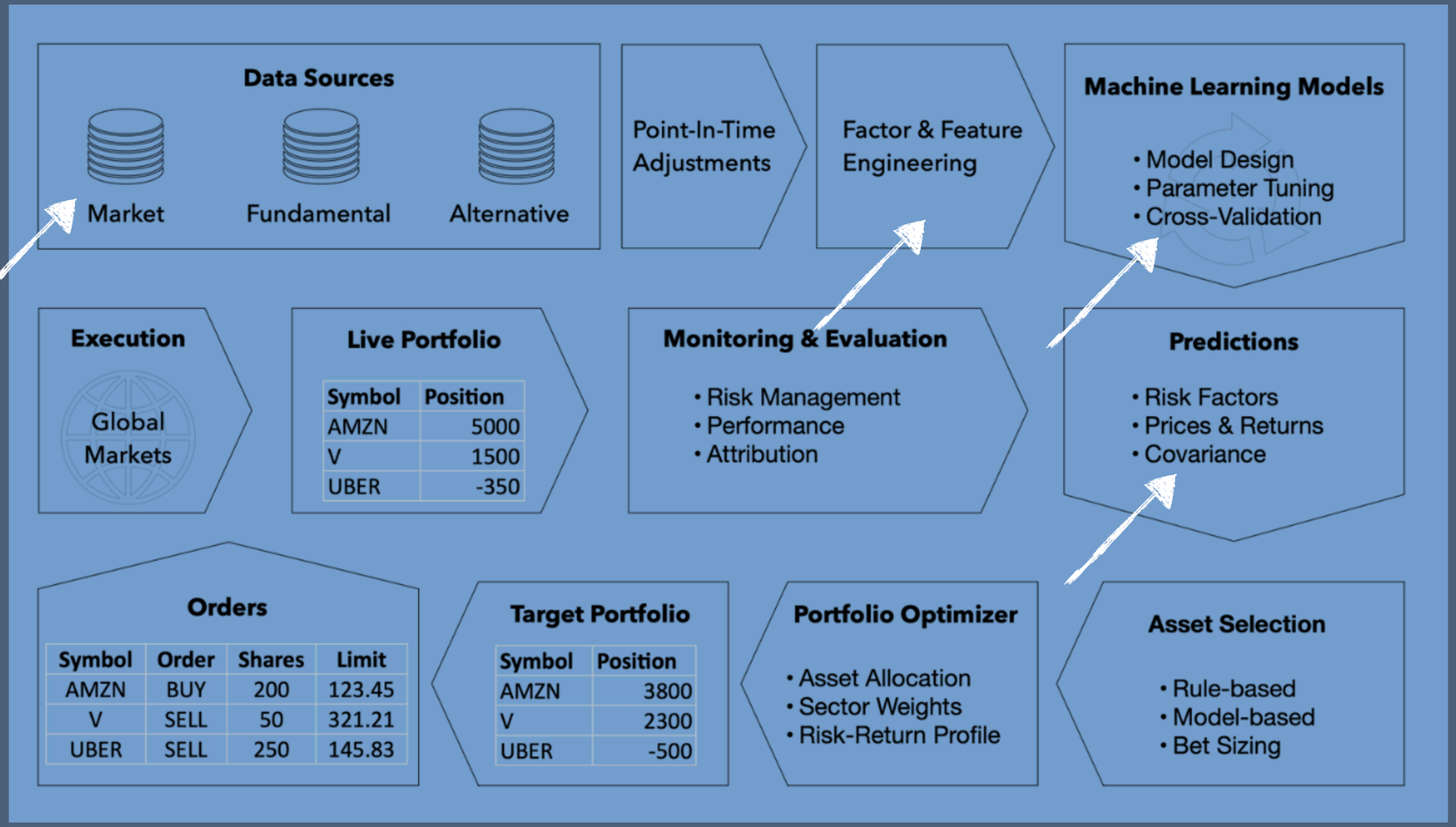

**Random Walk Hypothesis** Price movements are equivalent to a random step either up or down from the last price, with no predictable pattern. It suggests that changes in stock prices are random and unpredictable, making the last observed price $P_{t-1}$ the most reasonable estimate for the next price $P_t$ because it incorporates all known information up to that point.

The Model $P_t = P_{t-1} + \epsilon_t$

$$E[P_t \mid P_{t-1}] = E[P_{t-1} + \epsilon_t \mid P_{t-1}] = P_{t-1} + E[\epsilon_t] = P_{t-1}$$

**Efficient Market Hypothesis (EMH)**, which states that **asset prices fully reflect all available information**. In an efficient market, where prices behave like a random walk, the **best estimate of tomorrow’s price, absent new information, is simply today’s price.**

# End2End ML Trading pipeline



# Feature Engineering

Feature engineering plays a pivotal role in enhancing the performance of predictive models in the financial markets. By creatively transforming and expanding the original data set through feature engineering, analysts and traders can capture **more complex patterns and relationships that affect asset prices.**

However, in this project, relying solely on **Open, High, Low, and Close (OHLC)** data in financial modeling can be **limiting**. OHLC data primarily captures the price movements within specific **time frames** but often misses underlying market dynamics. It does not account for the volume of trades or the **volatility** during the trading period, which can provide critical insights into the strength of price movements.

**ATR (Average True Range)**: ATR measures market volatility by decomposing the entire range of an asset price for that period.

**RSI (Relative Strength Index)**: RSI is a momentum oscillator that measures the speed and change of price movements, typically used to identify overbought or oversold conditions in a market.

**BBBAND (Bollinger Bands)**: Bollinger Bands consist of a middle band being an N-period simple moving average (SMA) surrounded by an upper and lower band at a standard deviation distance from the SMA, used to measure volatility and identify 'overbought' or 'oversold' conditions.

**MACD (Moving Average Convergence Divergence)**: MACD is a trend-following momentum indicator that shows the relationship between two moving averages of a security's price, used to identify potential buy and sell signals.

$$\text{True Range} = \max((\text{High} - \text{Low}), |\text{High} - \text{Close}_{\text{previous}}|, |\text{Low} - \text{Close}_{\text{previous}}|)$$

$$\text{ATR} = \text{SMA}_{14}(\text{True Range})$$

$$\text{EMA}_t = (\text{P}_t \times \alpha) + (\text{EMA}_{t\text{-}1} \times (1 - \alpha))$$

$$\text{RS} = \frac{\text{Average Gain over N periods}}{\text{Average Loss over N periods}}$$

$$\text{MACD} = \text{EMA}_{12}(\text{Close}) - \text{EMA}_{26}(\text{Close})$$

$$\text{Signal Line} = \text{EMA}_9(\text{MACD})$$

$$\text{RSI} = 100 - \left(\frac{100}{1 + \text{RS}}\right)$$

$$\text{BBBand Up} = \text{SMA}_{20}(\text{Close}) + (2 \times \text{SD}_{20}(\text{Close}))$$

$$\text{BBBand Mid} = \text{SMA}_{20}(\text{Close})$$

$$\text{BBBand Low} = \text{SMA}_{20}(\text{Close}) - (2 \times \text{SD}_{20}(\text{Close}))$$

# Feature Engineering

```python
def make_lag_return(df):
    lag = [1, 2, 3, 6, 9, 12, 15]
    outlier_cutoff = 0.01
    for l in lag:
        df[f"return_{l}m"] = (
            df["Close"]
            .pct_change(l)
            .pipe(
                lambda x: x.clip(
                    lower=x.quantile(outlier_cutoff),
                    upper=x.quantile(1 - outlier_cutoff),
                )
            )
            .add(1)
            .pow(1 / l)
            .sub(1)
        )
```

$$\text{return\_}l\text{m} = \left( \left( \frac{\text{Close}_t}{\text{Close}_{t-l}} - 1 \right)_{\text{clip}(q_{0.01}, q_{0.99})} + 1 \right)^{\frac{1}{l}} - 1$$

# Feature Engineering

```python
def make_momentum(df):
    for l in [2, 3, 6, 9, 12, 15]:
        df[f"momentum_{l}m"] = df[f'return_{l}m'].sub(df.return_1m)
    df[f'momentum_3_12'] = df[f'return_12m'].sub(df.return_3m)
```

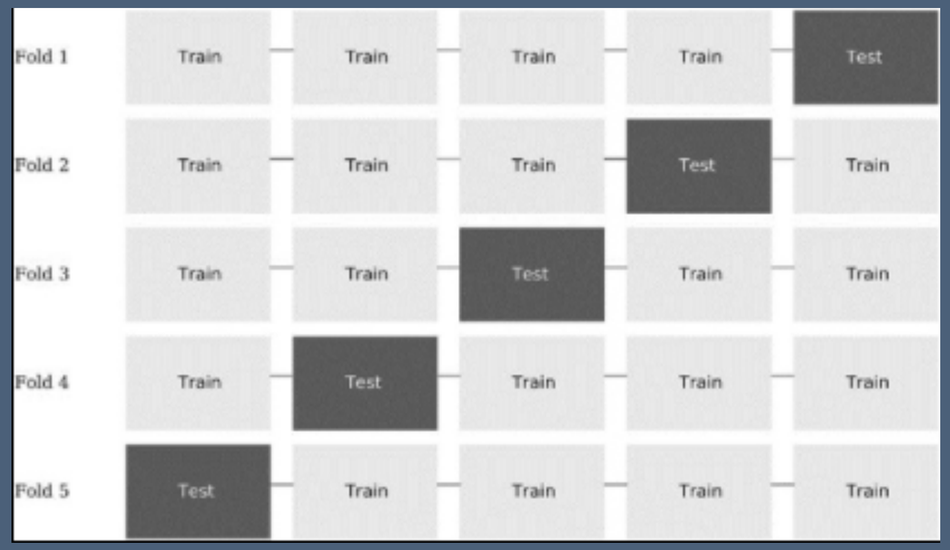
expecting the trends in these factors to continue in the future.

# Feature Engineering

```
def make_date_indicator(df):
    dates = df.index.get_level_values("timestamp")
    df['month'] = dates.month
    df['day'] = dates.day
```

# Methodology and Design

## Prevent Data Leakage




*"k-fold CV fails in finance is because observations cannot be assumed to be drawn from an IID process" Advances in Financial Machine Learning. Marcos M. López de Prado*

```
📅 Train from: 2021-05-19 ------> to: 2021-08-19 ||||||||  Test From: 2021-08-21------> to: 2021-09-20
📊 Train Shape: (132479, 29) Test Shape: (43200, 29)
📅 Train from: 2021-04-19 ------> to: 2021-07-20 ||||||||  Test From: 2021-07-22------> to: 2021-08-21
📊 Train Shape: (132479, 29) Test Shape: (43200, 29)
📅 Train from: 2021-03-20 ------> to: 2021-06-20 ||||||||  Test From: 2021-06-22------> to: 2021-07-22
📊 Train Shape: (132479, 29) Test Shape: (43200, 29)
```

Leakage takes place when the training set contains information that also appears in the testing set.

```python
class MultipleTimeSeriesCV:
    """Generates tuples of train_idx, test_idx pairs
    Assumes the MultiIndex contains levels 'symbol' and 'date'
    purges overlapping outcomes"""

    def __init__(self,
                 n_splits=3,
                 train_period_length=126,
                 test_period_length=21,
                 lookahead=None,
                 shuffle=False):
        self.n_splits = n_splits
        self.lookahead = lookahead
        self.test_length = test_period_length
        self.train_length = train_period_length
        self.shuffle = shuffle

    def split(self, X, y=None, groups=None):
        unique_dates = X.index.get_level_values('timestamp').unique()
        minutes = sorted(unique_dates, reverse=True)

        split_idx = []
        for i in range(self.n_splits):
            test_end_idx = i * self.test_length
            test_start_idx = test_end_idx + self.test_length
            train_end_idx = test_start_idx + self.lookahead - 1
            train_start_idx = train_end_idx + self.train_length + self.lookahead - 1
            split_idx.append([train_start_idx, train_end_idx,
                              test_start_idx, test_end_idx])

        dates = X.reset_index()[['timestamp']]
        for train_start, train_end, test_start, test_end in split_idx:
            train_idx = dates[(dates.timestamp > minutes[train_start])
                              & (dates.timestamp <= minutes[train_end])].index
            test_idx = dates[(dates.timestamp > minutes[test_start])
                             & (dates.timestamp <= minutes[test_end])].index
            if self.shuffle:
                np.random.shuffle(list(train_idx))
            yield train_idx, test_idx

    def get_n_splits(self, X, y, groups=None):
        return self.n_splits
```

# Methodology and Design

First, we selected few simple linear models for baseline studying, including **linear regression, ridge regression** and **lasso regression**. Then, we test those model on raw data then on the engineered data.

Second, we choose few **boosting models** (catboost, Xgboost, ExtraTree,lightgbm) to check if they could learn extra information(hyperparams-Tuning). At last we **stack** them into a regressor so as to improve the model performance.

Lastly, we tried deep neural nets. Again we first use **fully connected network** as the baseline, then we design a **lstm structure** to model the data. We first **transform the data into** 'sentence' and label them. Then we used warmup **technique** to control the learning rate for better leanrning.

**Methodology and Design**

Hyperparams-Tuning

*Suffering.Exhausting*

Optuna for params space search.

```python
def xgb_objective(trial, X_train, y_train, X_test, y_test):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        'max_depth': trial.suggest_int('max_depth', 3, 10),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3),
        'subsample': trial.suggest_discrete_uniform('subsample', 0.6, 0.9, 0.1),
        'colsample_bytree': trial.suggest_discrete_uniform('colsample_bytree', 0.6, 0.9, 0.1),
        'gamma': trial.suggest_float('gamma', 1e-8, 1.0),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 300),
        'reg_alpha': trial.suggest_loguniform('reg_alpha', 1e-8, 1.0),
        'reg_lambda': trial.suggest_loguniform('reg_lambda', 1e-8, 1.0),
        'n_jobs': -1,
        'random_state': 42,

        'lambda': trial.suggest_float('lambda', 1e-3, 10.0),
        'alpha': trial.suggest_float('alpha', 1e-3, 10.0),
        'colsample_bytree': trial.suggest_categorical('colsample_bytree', [0.3,0.4,0.5,0.6,0.7,0.8,0.9,
1.0]),
        'subsample': trial.suggest_categorical('subsample', [0.4,0.5,0.6,0.7,0.8,1.0]),
        'learning_rate': trial.suggest_categorical('learning_rate',
[0.008,0.01,0.012,0.014,0.016,0.018, 0.02]),
        'n_estimators': 10000,
        'max_depth': trial.suggest_categorical('max_depth', [5,7,9,11,13,15,17]),
        'random_state': trial.suggest_categorical('random_state', [2020]),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 300),
    }
    clf = XGBRegressor(**params)
    clf.fit(X_train, y_train)
    return mse(y_test, X_test)
```

**Methodology and Design**

# Evaluation

The **Information Coefficient (IC)** is a measure used in finance to quantify the predictive skill of a financial forecast, specifically the correlation between predicted and actual asset returns. The mathematical formula for the Information Coefficient (IC) is:

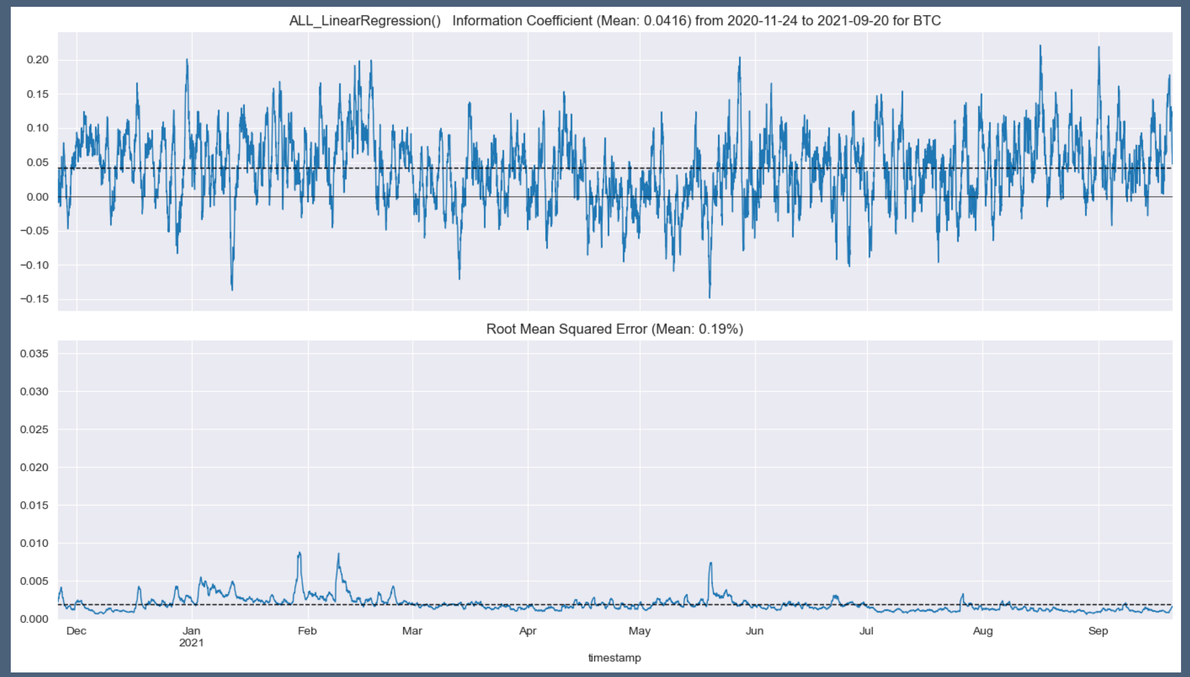
$$IC = corr(R_t, \hat{R}_t)$$

The Information Coefficient (IC) is considered valuable for predicting returns because it directly measures the strength and direction of the linear relationship between **forecasted values and actual outcomes.** A **higher absolute value** of the IC indicates a **stronger predictive power** of the forecast model,

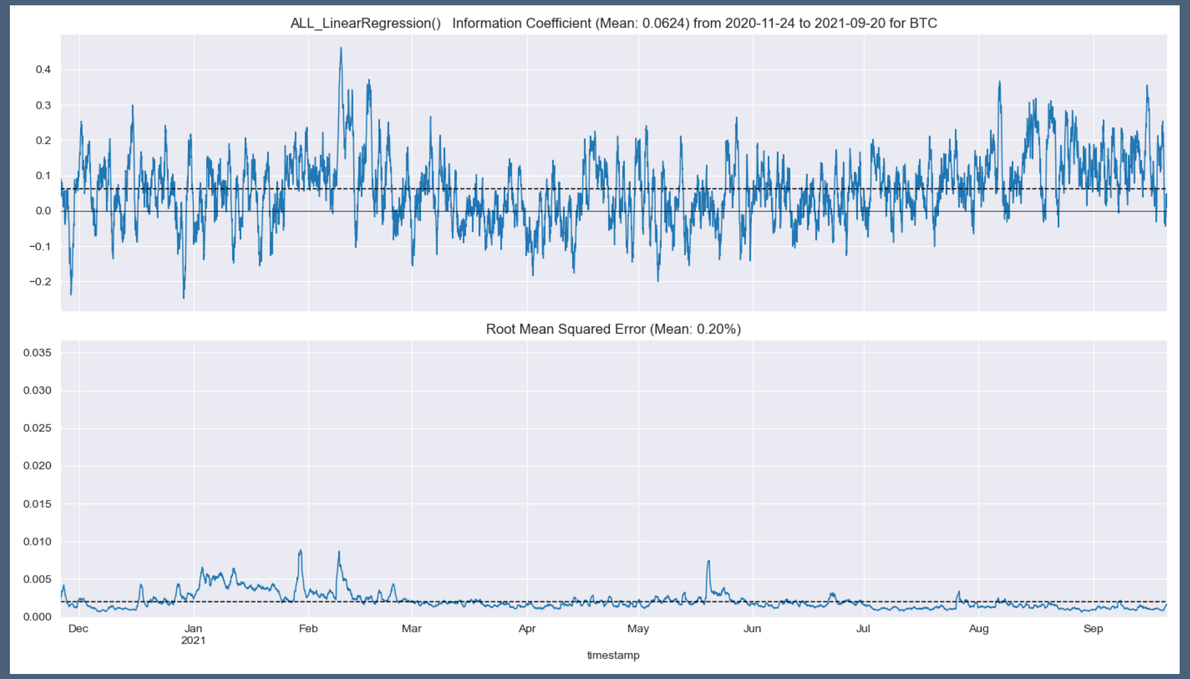
**Result**

# Linear Regression Baseline

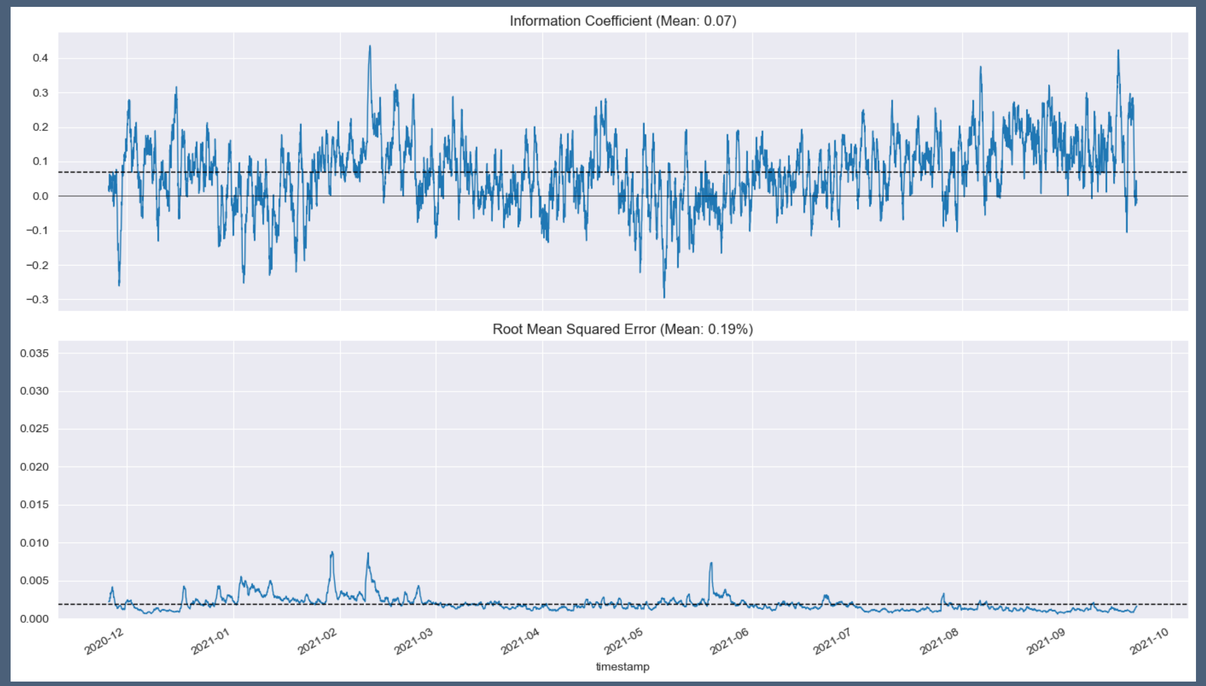
## Raw input

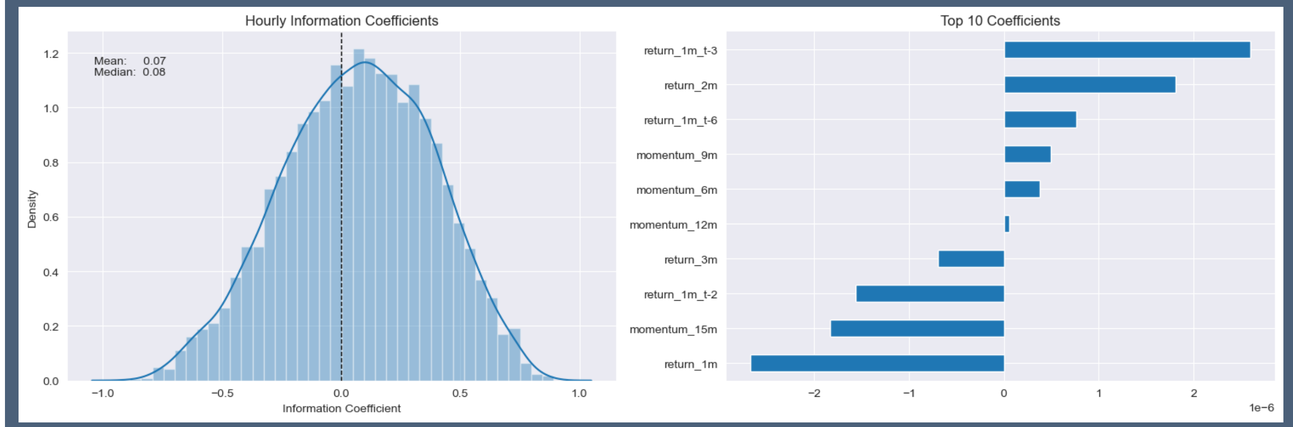


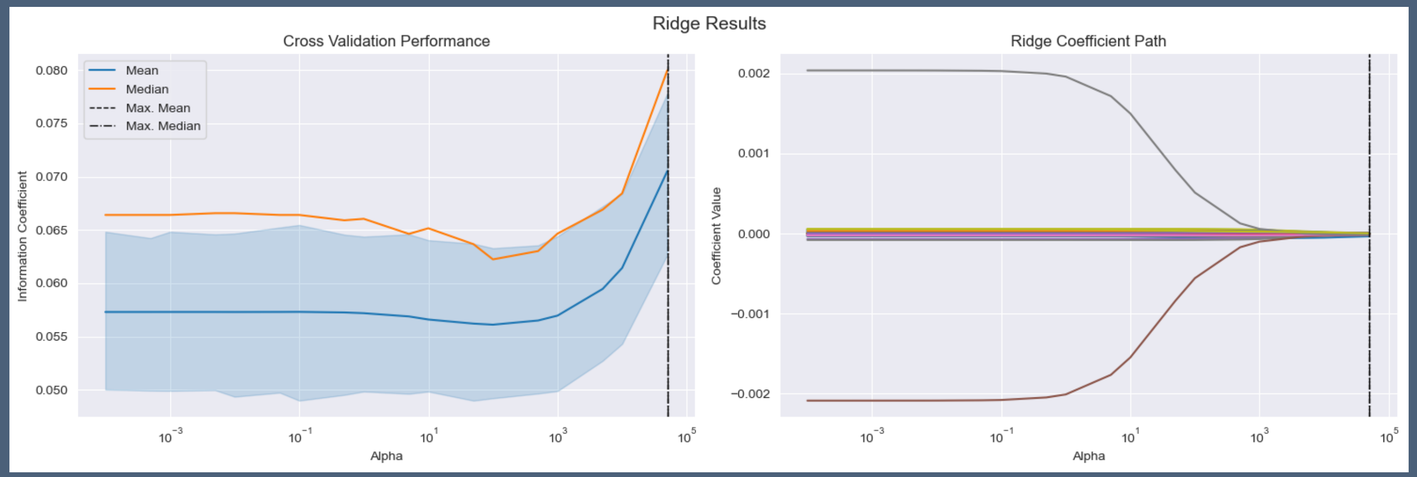

## FE input



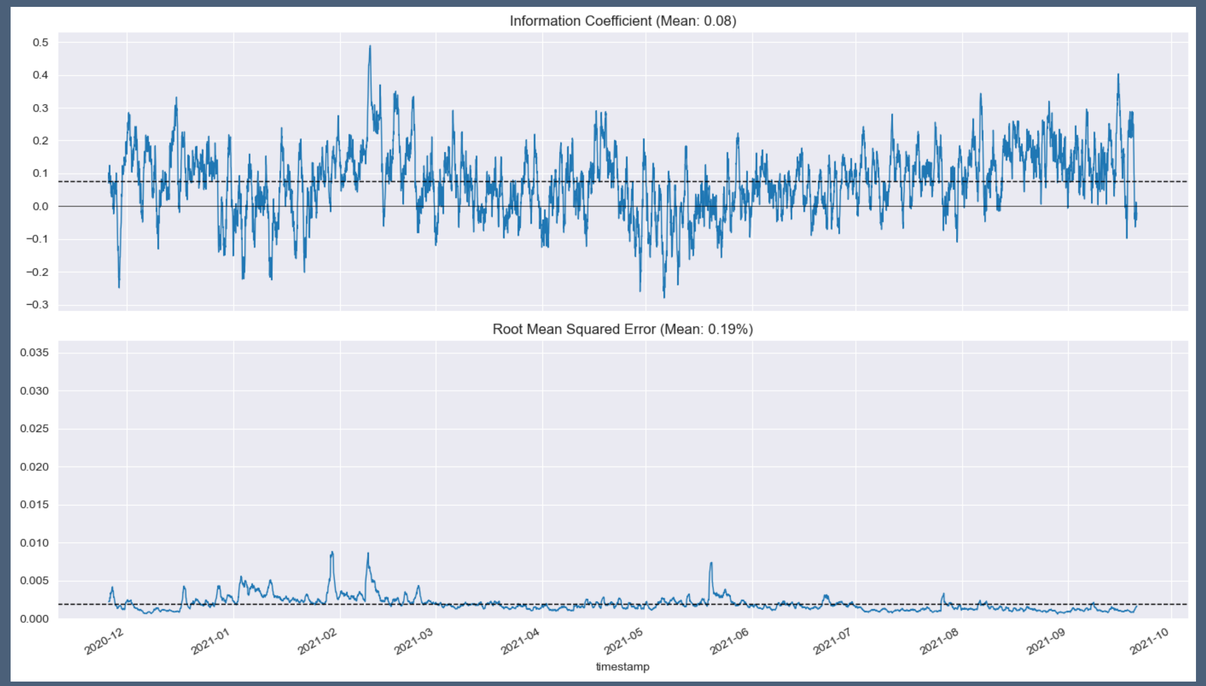
# **Result**
## Ridge Regression

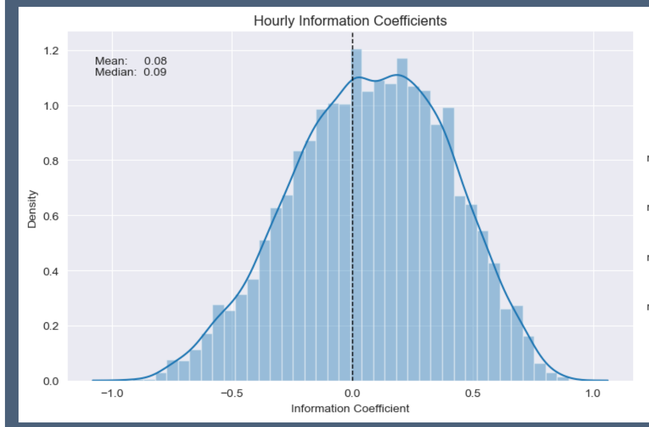


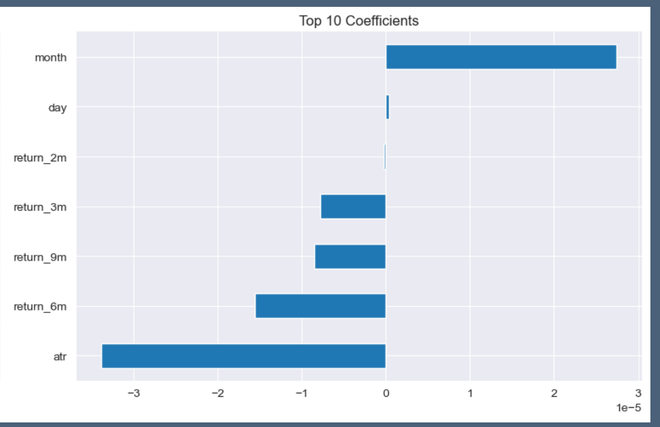



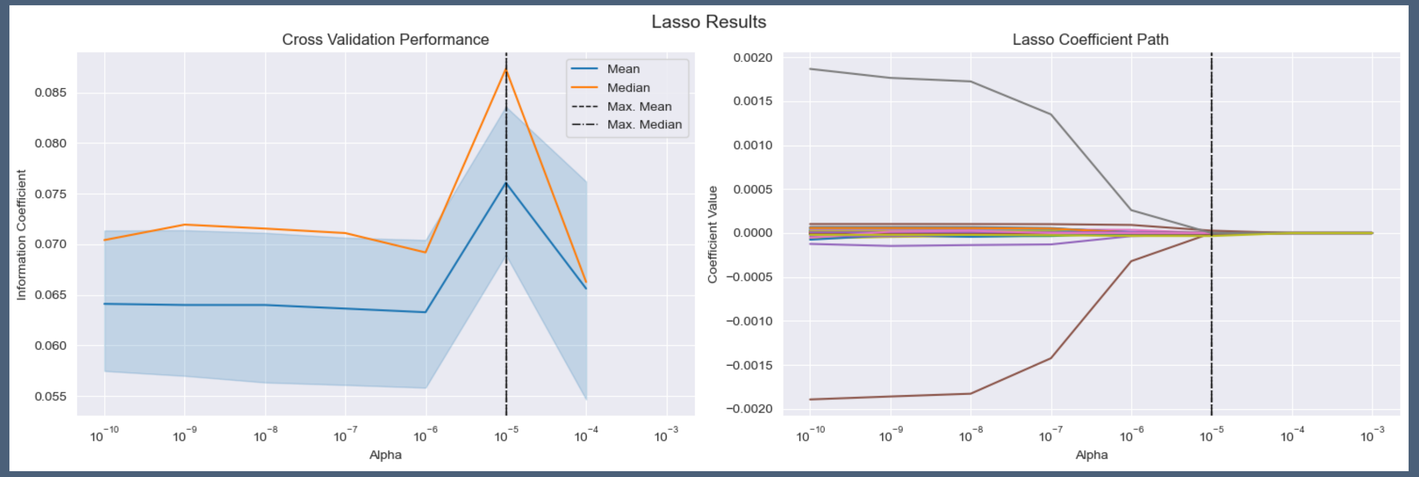



# **Result**
## Lasso Regression



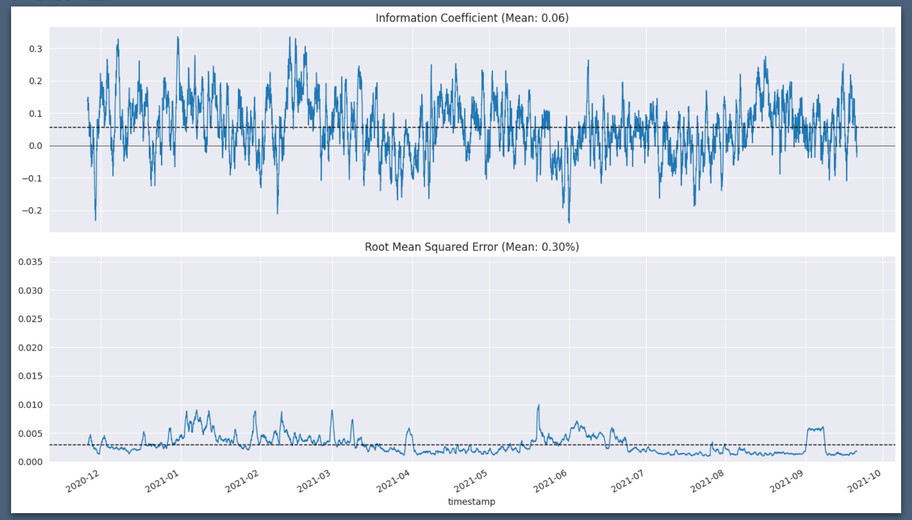



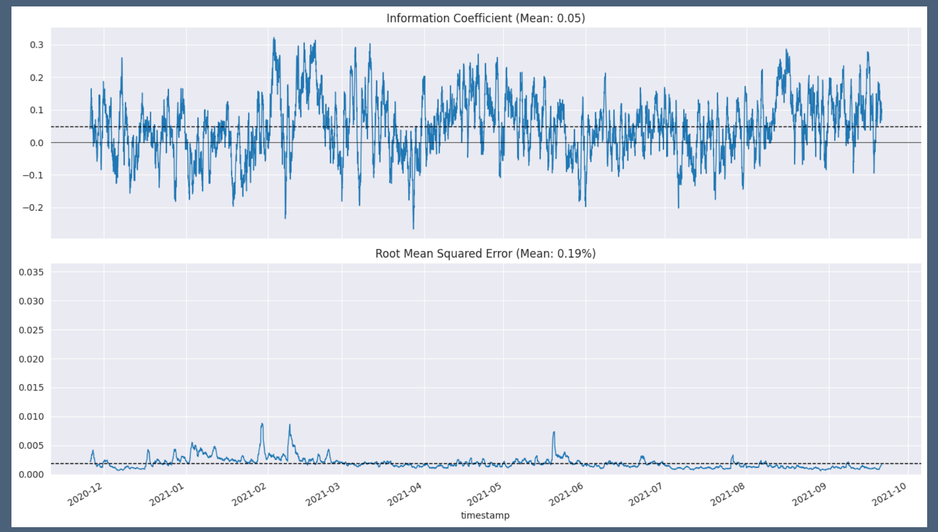
**Result**

# ML Regression

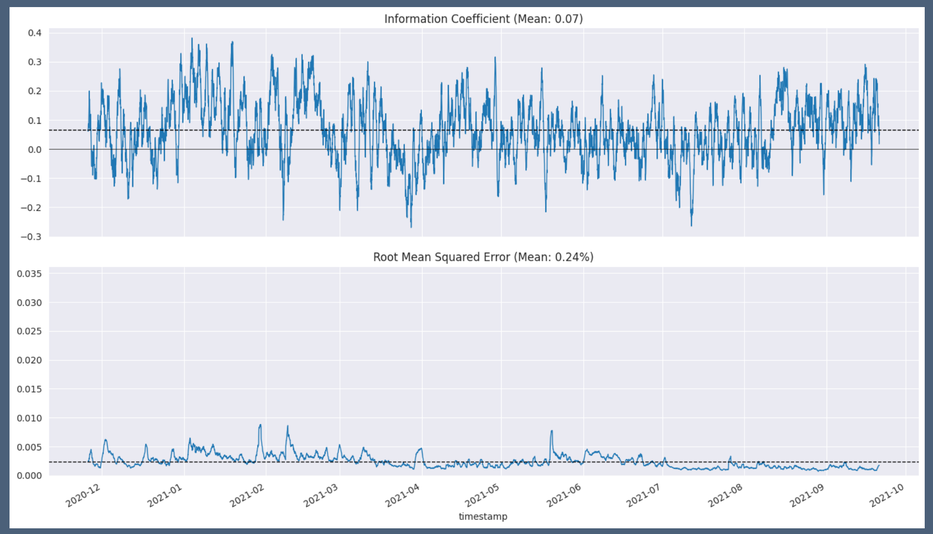
## XGB



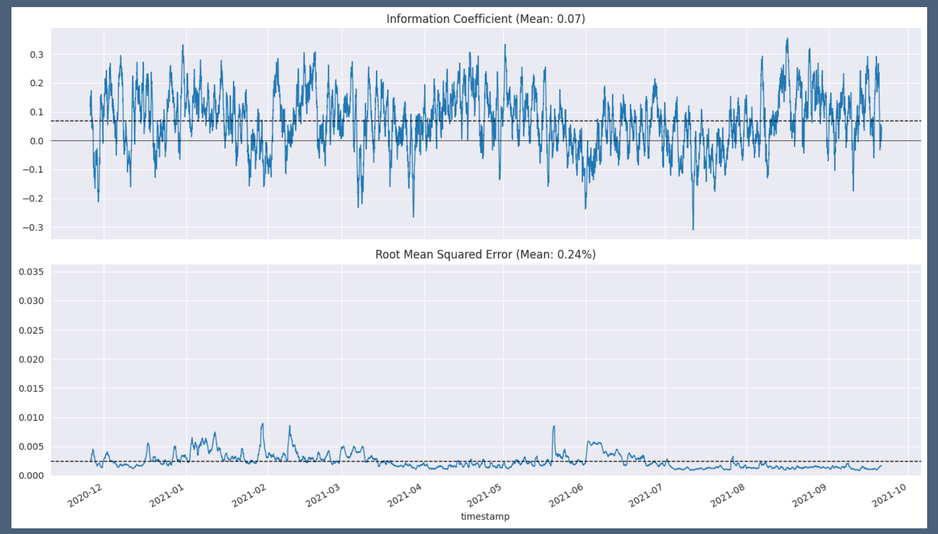

## Stacker(all)



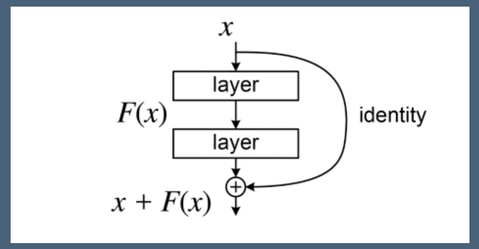

## CatBoost




## LGBM



# Neural Nets




Since the legend Kaiming He invented resnet, I took some of his idea for this project. First I defined the residualBlock, which is proven to be capable of solving gradient vanishing problem by adding the residual connection. So I think it would be a good idea to make the nets deeper with his methods instead of no-brain Linear(30)->Linear(128)->Linear(256)->/...../ (Well Encoder-Decoder might work, i dont know for now.)

```python
class ResidualBlock(nn.Module):
    def __init__(self, input_size, output_size):
        super(ResidualBlock, self).__init__()
        self.linear1 = nn.Linear(input_size, output_size)
        self.relu = nn.ReLU()
        self.linear2 = nn.Linear(output_size, output_size)

    def forward(self, x):
        identity = x
        out = self.linear1(x)
        out = self.relu(out)
        out = self.linear2(out)
        out += identity  # Add input to the output (residual connection)
        out = self.relu(out)  # Apply activation after adding the residual
        return out
```

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        input_size = 7
        num_layers = 4
        hidden_size = 128
        num_blocks = 52
        layers = []
        act_fn = nn.ReLU()
        layers.append(nn.Linear(input_size, hidden_size))
        layers.append(act_fn)
        ###########ver1
        # for _ in range(num_layers - 1):
        #     layers.append(nn.Dropout(0.1))
        #     layers.append(nn.Linear(hidden_size, hidden_size))
        #     layers.append(act_fn)

        # Adding residual blocks and dropout
        #########ver2
        for i in range(num_layers - 1):
            if i % 2 == 0:  # Add a residual block every second iteration
                layers.append(ResidualBlock(hidden_size, hidden_size))
            else:
                layers.append(nn.Dropout(0.1))
                layers.append(nn.Linear(hidden_size, hidden_size))
                layers.append(act_fn)
        layers.append(nn.Linear(hidden_size, 1))
        layers.append(nn.Sigmoid())


        self.main_block = nn.Sequential(*layers)
        ###########
        #layers = [nn.Linear(input_size, 64), nn.ReLU()]

        # Add more residual blocks
        # for _ in range(num_blocks):
        #     layers.append(ResidualBlock(64, 64))
        #     layers.append(nn.Dropout(0.1))  # Regularization with dropout

        # # Output layer
        # layers.append(nn.Linear(64, 1))
        # layers.append(nn.Sigmoid())

        # self.main_block = nn.Sequential(*layers)
```

# Result

| Model | Information Coefficient | RMSE |
|---|---|---|
| LASSO - FE | 0.08 | 0.19% |
| Ridge - FE | 0.07 | 0.19% |
| CatBoost - FE | 0.07 | 0.24% |
| LGBM - FE | 0.07 | 0.24% |
| Linear Regression - FE | 0.0624 | 0.2% |
| XGBoost - FE | 0.06 | 0.30% |
| Ensemble - FE | 0.05 | 0.19% |
| Linear Regression - RAW | 0.04 | 0.19% |
| LASSO - RAW | 0.03 | 0.19% |
| Ridge - RAW | 0.03 | 0.19% |
| DNN - FE | 0.01 | 0.37% |
| LSTM - FE | 0.01 | 0.19% |

**LASSO** outperforms all other models including deep learning one. I would say the features
itself already contained lots of noise, adding more layers/making the model more complex
would probably fit the noise seriously resulting in **bad predicting power**(low ic), but 'good'
metric (low rise).
Deeper isn't always the best choice for learning, especially in financial market. By my experience in the industry, feature engineering/factor mining is the key of reducing noise.
Making the data with **higher frequency** like in nano seconds would achieve that goal also.

# Future Work

1. Use **L2 data** ( snapshot of order book, spread, imbalance..)
2. **AutoFE** for feature engineering
3. AutoML

# Thanks