

# NYCU\_Artificial\_Intelligence Capstone\_HW1

---

- This dataset is available at: [here](https://github.com/hsuiyee/NYCU_Artificial_Intelligence_Capstone_Labs/tree/main/Lab1)  
([https://github.com/hsuiyee/NYCU\\_Artificial\\_Intelligence\\_Capstone\\_Labs/tree/main/Lab1](https://github.com/hsuiyee/NYCU_Artificial_Intelligence_Capstone_Labs/tree/main/Lab1)).
- This pdf is available at: [here](https://hackmd.io/@Origamyee/B1pfNatqJJ) (<https://hackmd.io/@Origamyee/B1pfNatqJJ>).
- This code is available at: [here](https://github.com/hsuiyee/NYCU_Artificial_Intelligence_Capstone/tree/main/Lab1)  
([https://github.com/hsuiyee/NYCU\\_Artificial\\_Intelligence\\_Capstone/tree/main/Lab1](https://github.com/hsuiyee/NYCU_Artificial_Intelligence_Capstone/tree/main/Lab1)).

## Part 0. Overview

---

In this homework, our goal is to predict the BTC/USDT cryptocurrency close price change five hours ahead and use this prediction to design our trading strategy. Below, we outline the process step by step:

1. Create Dataset
2. Clean Up Dataset
3. Algorithm and Analysis
4. Experiments
5. Discussion
6. References

## Part 1. Create Dataset

---

To make predictions, we need OHLC (open, high, low, close) price data. Therefore, we use the Binance API to retrieve such data (details available at: [Binance Public Data](https://github.com/binance/binance-public-data) (<https://github.com/binance/binance-public-data>), and [Kline/Candlestick Data](https://developers.binance.com/docs/derivatives/usds-margined-futures/market-data/rest-api/Kline-Candlestick-Data) (<https://developers.binance.com/docs/derivatives/usds-margined-futures/market-data/rest-api/Kline-Candlestick-Data>)). The implementation is handled by `preprocess.py`. We set the K-line parameters as follows:

- a. `symbol = BTCUSDT`
- b. `interval = '1h'`
- c. `start_date = '2024-11-01'`
- d. `end_date = '2025-02-01'`

If more column values are missing, we simply use `dropna` to remove them.

After running `preprocess.py`, we obtain a CSV file named `klines_BTC.csv`, which follows the format below:

klines_BTC								
open_time	open	high	low	close	volume	taker_buy_base_asset_volume	taker_buy_quote_asset_volume	
2024-11-01 00:00:00	70321.9	70500.0	70160.9	70188.6	6293.539		3122.97	219650270.5021
2024-11-01 01:00:00	70188.7	70384.6	69812.2	69424.9	23094.976		9133.652	637407952.6586
2024-11-01 02:00:00	69424.9	69621.2	68870.4	69592.7	25645.369		12269.503	849623340.7967
2024-11-01 03:00:00	69592.7	69739.0	69357.1	69397.7	6709.285		3069.884	213563233.8988
2024-11-01 04:00:00	69397.7	69650.0	69363.9	69618.5	3543.899		1949.728	135516392.2082

## Part 2. Clean Up Dataset

### 2.1 Feature Engineering

We add the target label **target\_pct\_change** to the dataset, defined by  $(Close_{i+1} - Close_i) / Close_i$ . Since OHLC, Volume, Taker buy base asset volume, and Taker buy quote asset volume alone may not fully represent the price trend, we incorporate additional common factors to enhance the model's ability to fit price changes.

Specifically, we add volatility, price\_range, ma\_7, and other similar features. The implementation is handled by `add_factors.py`. After running `add_factors.py`, we obtain a CSV file named `klines_BTC_with_factors.csv`, which follows the format below:

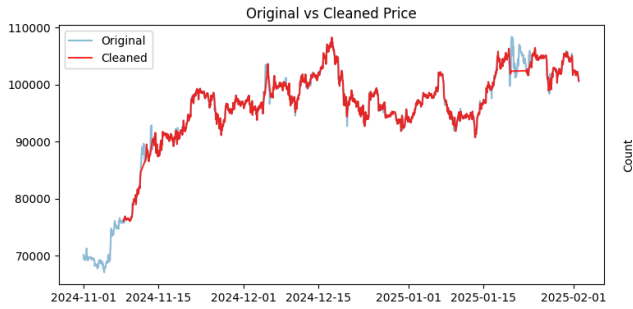
klines_BTC														
open_time	open	high	low	close	volume	taker_buy_base_asset_volume	taker_buy_quote_asset_volume	ma_7	volatility	price_range	ma_7	ma_14	ma_21	ma_28
2024-11-01 00:00:00	70321.9	70500.0	70160.9	70188.6	6293.539		3122.97	219650270.5021	0.0000000000	0.0000000000	70321.9	70321.9	70321.9	70321.9
2024-11-01 01:00:00	70188.7	70384.6	69812.2	69424.9	23094.976		9133.652	637407952.6586	0.0000000000	0.0000000000	70188.7	70188.7	70188.7	70188.7
2024-11-01 02:00:00	69424.9	69621.2	68870.4	69592.7	25645.369		12269.503	849623340.7967	0.0000000000	0.0000000000	69424.9	69424.9	69424.9	69424.9
2024-11-01 03:00:00	69592.7	69739.0	69357.1	69397.7	6709.285		3069.884	213563233.8988	0.0000000000	0.0000000000	69592.7	69592.7	69592.7	69592.7
2024-11-01 04:00:00	69397.7	69650.0	69363.9	69618.5	3543.899		1949.728	135516392.2082	0.0000000000	0.0000000000	69397.7	69397.7	69397.7	69397.7

klines_BTC_with_factors														
open_time	open	high	low	close	volume	taker_buy_base_asset_volume	taker_buy_quote_asset_volume	ma_7	volatility	price_range	ma_7	ma_14	ma_21	ma_28
2024-11-01 00:00:00	70321.9	70500.0	70160.9	70188.6	6293.539		3122.97	219650270.5021	0.0000000000	0.0000000000	70321.9	70321.9	70321.9	70321.9
2024-11-01 01:00:00	70188.7	70384.6	69812.2	69424.9	23094.976		9133.652	637407952.6586	0.0000000000	0.0000000000	70188.7	70188.7	70188.7	70188.7
2024-11-01 02:00:00	69424.9	69621.2	68870.4	69592.7	25645.369		12269.503	849623340.7967	0.0000000000	0.0000000000	69424.9	69424.9	69424.9	69424.9
2024-11-01 03:00:00	69592.7	69739.0	69357.1	69397.7	6709.285		3069.884	213563233.8988	0.0000000000	0.0000000000	69592.7	69592.7	69592.7	69592.7
2024-11-01 04:00:00	69397.7	69650.0	69363.9	69618.5	3543.899		1949.728	135516392.2082	0.0000000000	0.0000000000	69397.7	69397.7	69397.7	69397.7

### 2.2 Data Cleaning with Autoencoders and Z-score

Due to the high level of noise in the market, I use autoencoders and Z-score to filter out noise while preserving useful data. The refined result is shown below:



## Part 3. Algorithm and Analysis

### Models Selection

We use the following models from the `sklearn` library to predict `target_pct_change` or `target_multiclass`:

- Supervised learning: XGBoost regression and Transformer
- Unsupervised learning: K-Means (Class 1: lower than -2%, Class 2: -2% to -0.5%, Class 3: -0.5% to 0.5%, Class 4: 0.5% to 2%, Class 4: greater than 2%)

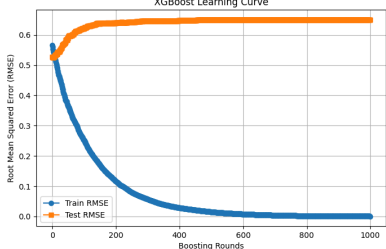
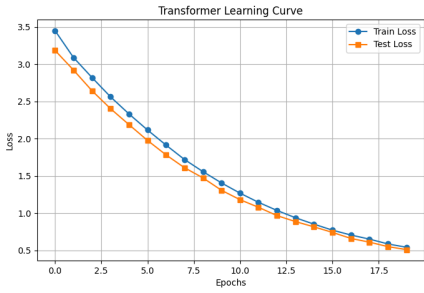
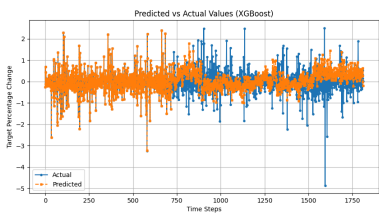
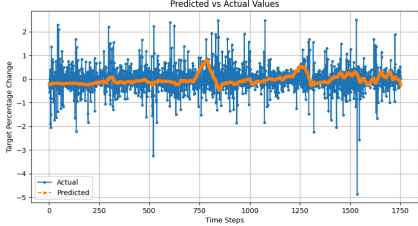
And the default hyperparameters is below:

Transformer	XGBoost	Kmeans
<pre># Define configuration settings class Config:     """     Configuration class to centralize all hyperparameters and settings.     """     # File paths and names     FOLDER_PATH = ""     INPUT_FILE = "klines_BTC_with_factors.csv"     OUTPUT_FILE = "klines_BTC_factors_with_direction.csv"     VISUALIZATION_FILE = "learning_curve_transformer.png"     PREDICTION_PLOT_FILE = "predictions_vs_actual_transformer.png"     MODEL_FILE = "transformer_model.h5"  # Train-test split parameters TEST_SIZE = 0.6 FORECAST_HORIZON = 1  # Training parameters EPOCHS = 20 BATCH_SIZE = 32 TIME_STEPS = 60</pre>	<pre># Define configuration settings class Config:     """     Configuration class to centralize all hyperparameters and settings.     """     # File paths and names     FOLDER_PATH = ""     INPUT_FILE = "klines_BTC_PCA.csv"     OUTPUT_FILE = "klines_BTC_factor_with_direction.csv"     VISUALIZATION_FILE = "learning_curve_xgboost.png"     PREDICTION_PLOT_FILE = "predictions_vs_actual_xgboost.png"     MODEL_FILE = "xgboost_model.json"  # Train-test split parameters TEST_SIZE = 0.6  # Training parameters N_ESTIMATORS = 1000 LEARNING_RATE = 0.05 MAX_DEPTH = 6 SUBSAMPLE = 0.8 COLSAMPLE_BYTREE = 0.8</pre>	<pre># Define configuration settings class Config:     """     Configuration class to centralize all hyperparameters and settings.     """     # File paths and names     INPUT_FILE = "klines_BTC_with_factors.csv"     OUTPUT_FILE = "klines_BTC_clusters.csv"     CLUSTER_PLOT_FILE = "kmeans_clusters.png"  # Clustering parameters TIME_STEPS = 60 N_CLUSTERS = 5 # 5個クラス RANDOM_STATE = 42 # 固定値で結果を再現</pre>

### Reference Public Libraries

- [scikit-learn \(https://scikit-learn.org/stable/\)](https://scikit-learn.org/stable/)
- [hmmlearn \(https://hmmlearn.readthedocs.io/en/latest/\)](https://hmmlearn.readthedocs.io/en/latest/)
- [XGBoost \(https://xgboost.readthedocs.io/en/stable/\)](https://xgboost.readthedocs.io/en/stable/)
- [TensorFlow \(https://www.tensorflow.org/?hl=zh-tw\)](https://www.tensorflow.org/?hl=zh-tw)

## Evaluate the performance (supervised learning)

compare	XGBoost	Transformer
loss		
prediction		

## Evaluate the performance (unsupervised learning)

### Kmeans

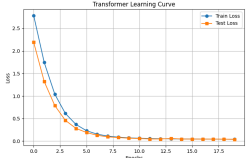
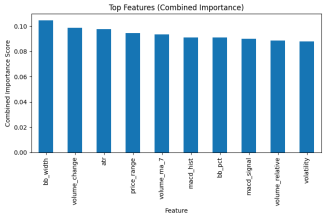
1. Silhouette Score: 0.0959 (higher is better)
2. Inertia (SSE): 2260203.5053 (lower is better)
3. Adjusted Rand Index (ARI): 0.0064 (higher is better)
4. Normalized Mutual Information (NMI): 0.0063 (higher is better)
5. Accuracy: 0.3333 (higher is better)

## Part 4. Experiments

### Problem 1

What if we use a different time interval?

We modified the dataset's time interval setting from 1h to 15m, and the comparison results are shown below.

time_interval	Transformer	XGBoost	important features
15m			
1h			

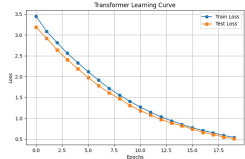
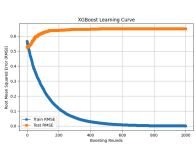
As we can see, both models achieve better performance with a smaller time interval, as indicated by their lower testing loss. Intuitively, a smaller time interval functions similarly to data augmentation. Additionally, the Transformer converges more efficiently, while XGBoost overfits at a slower rate. Moreover, the important features remain unchanged.

## Problem 2

What if we do not perform data cleaning with Autoencoders and Z-score?

Intuitively, this would lead to poor training and testing results.

We compare the effect of data cleaning using Autoencoders and Z-score below:

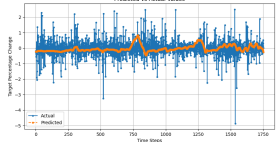
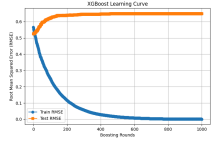
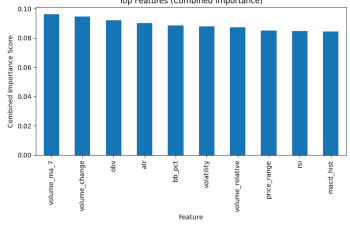
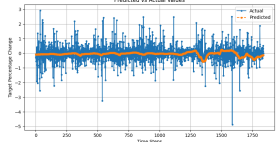
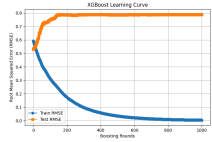
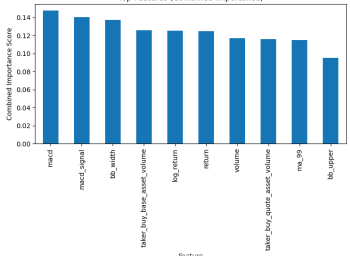
data_cleaning	Transformer	XGBoost	Important features
YES			
NO			

As we can see, if we do not perform data cleaning, the training and testing performance of XGBoost deteriorates significantly, while the Transformer model remains nearly the same. Additionally, the important features differ slightly; for example, `rsi` and `macd_signal` do not appear in the same diagram. Why do we obtain such a result? Since we did not perform data cleaning, XGBoost may learn from noise, whereas the Transformer is more resistant to noise as it can capture temporal behavior.

## Problem 3

What if we delete some important features?

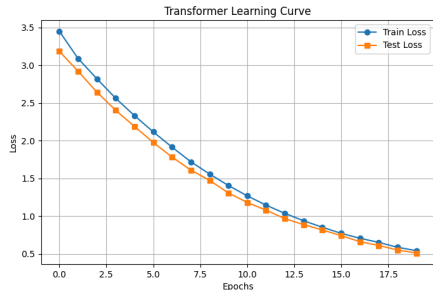
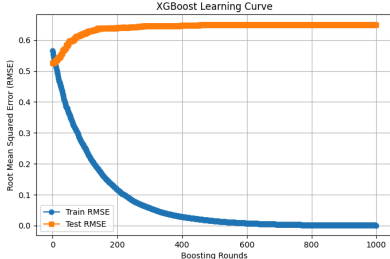
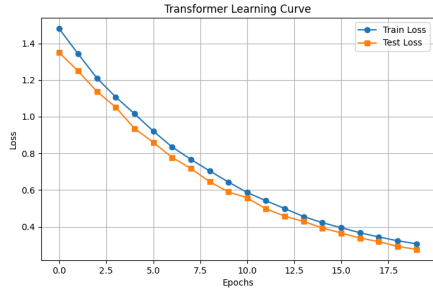
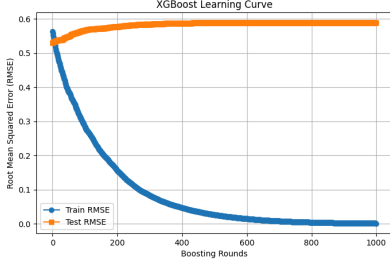
First, we define "importance" as the total decrease in impurity when a feature is used for splitting across all trees. Second, we remove the top 10 most important features and analyze their effect. The experiment results are shown below.

compare	Transformer	XGBoost	Important features
before			
after			

As we can see, when we change only the feature vectors, the Transformer model becomes more cautious after removing important features. Without key features, the model has less confidence and predicts price changes more conservatively. Additionally, this leads to an increase in XGBoost's testing loss.

## Problem 4

What if we apply the PCA method to reduce noise, remove unimportant features, and decrease training time?

compare	Transformer	XGBoost
before		
after		

```

✓ Initial feature count (excluding targets): 34
✓ PCA reduced dimensions to 12, preserving 95.0% variance.
✗ Dropped 22 features (from 34 → 12)
✓ Retained feature names: ['open', 'high', 'low', 'close', 'volume', 'taker_buy_base_asset_volume',
'taker_buy_quote_asset_volume', 'return', 'log_return', 'volatility', 'price_range', 'ma_7']
✓ Explained variance plot saved to pca_explained_variance.png
✓ PCA-transformed data saved to klines_BTC_PCA.csv
origamye@Hsiu-IdeMacBook-Pro ~ % cd ~/Desktop/NYCU_Artificial_Intelligence_Capstone/Lab1 && python3.12 predict_transformer.py
✓ Selected 12 features for training.

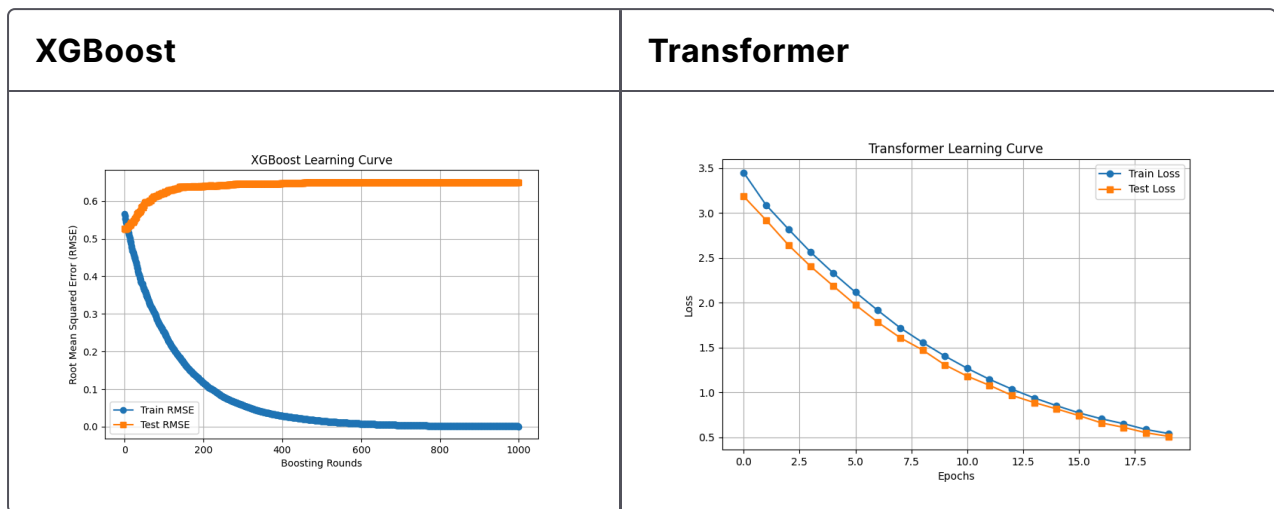
```

As we can see, when we apply the PCA method to the dataset, the training and testing loss significantly decrease after a few epochs in both models. This is because the PCA model reduces correlation among feature vectors and simplifies the complexity of the problem.

## Problem 5

What if we use a non-temporal model to predict temporal behavior compared to using a temporal model?

The XGBoost model tends to overfit compared to the Transformer model. Below is an example where we use similar loss functions: Pseudo-Huber error in XGBoost and Huber loss in the Transformer model.



As we can see, the testing loss increases as the number of training iterations increases. This is a typical sign of overfitting. Intuitively, since the data depends on time, the model may fail to capture temporal effects properly.

## Part 5. Discussion

### Problem 1

Describe experiments that you would do if there were more time available.

1. Add more statistical indices and analyze their effects.
2. Expand our project—since we have the next price change, we can develop a long/short strategy and evaluate its performance using PnL as a metric.
3. Predict more hyperparameters, such as the stop-loss threshold, to improve balance.
4. Try more models and analyze their effects.

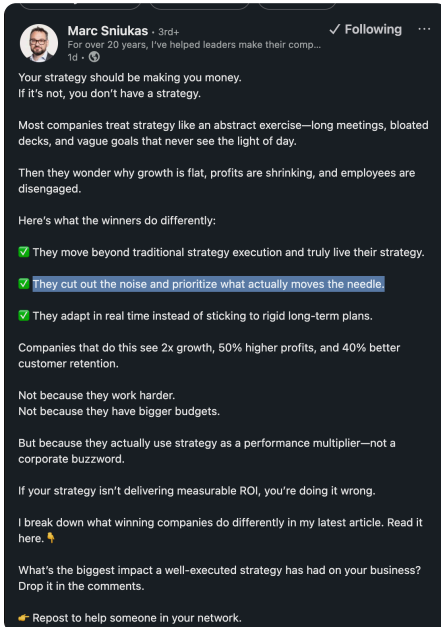
### Problem 2

What I have learned from the experiments as well as your remaining questions?



## Parts 1 and 2

In Parts 1 and 2, I learned how to clean up data. Initially, I thought we could simply add statistical indices to enhance our fitting power. However, after reading a LinkedIn post, I realized that the data might contain noise. This naturally led to an important question: "How can we filter out noise while keeping useful data?"



I tried to find the answer on the Internet, and the top three posts that inspired me are:

### 1. Towards Denoised Market Indices Pt. 1 — Static Indices

(<https://medium.com/@lavaronic/towards-denoised-market-indices-pt-1-static-indices-ac9004e930b1>).

### 2. Autoencoders Simplified: Real-World Data Cleaning Application

Satejraste (<https://medium.com/@satejraste/autoencoders-simplified-real-world-data-cleaning-application-5dda1b3c686d>).

### 3. AutoEncoder (一)-認識與理解 (<https://medium.com/ml-note/autoencoder-%E4%B8%80-%E8%AA%8D%E8%AD%98%E8%88%87%E7%90%86%E8%A7%A3-725854ab25e8>).

Therefore, I used an autoencoder and Z-score for denoising.

## Part 3

In Part 3, I learned how to use PCA for denoising. Thanks to the following tutorial, I gained an understanding of its mathematical intuition and why it works intuitively.

1. Principal Component Analysis(PCA) (<https://www.geeksforgeeks.org/principal-component-analysis-pca/>).
2. Learning Model : Unsupervised Machine Learning\_主成分分析 (PCA) 原理詳解 (<https://medium.com/ai%E5%8F%8D%E6%96%97%E5%9F%8E/preprocessing-data-%E4%B8%BB%E6%88%90%E5%88%86%E5%88%86%E6%9E%90-pca-%E5%8E%9F%E7%90%86%E8%A9%B3%E8%A7%A3-afe1fd044d4f>).
3. 【机器学习】降维——PCA（非常详细） (<https://zhuanlan.zhihu.com/p/77151308>).

## Part 6. References

---

1. Binance Public Data (<https://github.com/binance/binance-public-data>).
2. Kline/Candlestick Data (<https://developers.binance.com/docs/derivatives/usds-margined-futures/market-data/rest-api/Kline-Candlestick-Data>).
3. scikit-learn (<https://scikit-learn.org/stable/>).
4. hmmlearn (<https://hmmlearn.readthedocs.io/en/latest/>).
5. XGBoost (<https://xgboost.readthedocs.io/en/stable/>).
6. TensorFlow (<https://www.tensorflow.org/?hl=zh-tw>).
7. Towards Denoised Market Indices Pt. 1 — Static Indices (<https://medium.com/@lavaronic/towards-denoised-market-indices-pt-1-static-indices-ac9004e930b1>).
8. Autoencoders Simplified: Real-World Data Cleaning Application Satejraste (<https://medium.com/@satejraste/autoencoders-simplified-real-world-data-cleaning-application-5dda1b3c686d>).
9. AutoEncoder (一)-認識與理解 (<https://medium.com/ml-note/autoencoder-%E4%B8%80-%E8%AA%8D%E8%AD%98%E8%88%87%E7%90%86%E8%A7%A3-725854ab25e8>).
10. Principal Component Analysis(PCA) (<https://www.geeksforgeeks.org/principal-component-analysis-pca/>).
11. Learning Model : Unsupervised Machine Learning\_主成分分析 (PCA) 原理詳解 (<https://medium.com/ai%E5%8F%8D%E6%96%97%E5%9F%8E/preprocessing-data-%E4%B8%BB%E6%88%90%E5%88%86%E5%88%86%E6%9E%90-pca-%E5%8E%9F%E7%90%86%E8%A9%B3%E8%A7%A3-afe1fd044d4f>).
12. 【机器学习】降维——PCA（非常详细） (<https://zhuanlan.zhihu.com/p/77151308>).