

SC1015

Mini-Project

Cryptocurrency Stock Price Time Series Analysis

SC1015 B126 Group 3

Wang Yangming

Wang Anqi

Yang Ziyu

Problem and Goals

Exploratory Data Analysis

Machine Learning

Data-driven Insights

Contents

Problems and Goals

Decentralized Digital Currency



Goal

World's **First**
Cryptocurrency



Machine Learning



The Dataset

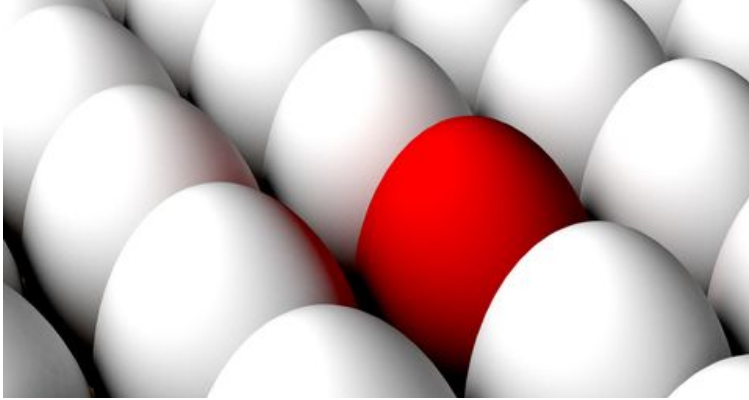


3108 Days

2014-9-17

2023-3-21

Exploratory Data Analysis



DATA CURATION



DATA ANALYSIS

RangeIndex: 3108 entries, 0 to 3107

Data columns (total 7 columns):

#	Column	Non-Null Count	Dtype
---	-----	-----	-----
0	date	3108 non-null	object
1	open	3108 non-null	float64
2	high	3108 non-null	float64
3	low	3108 non-null	float64
4	close	3108 non-null	float64
5	adj_close	3108 non-null	float64
6	volume	3108 non-null	int64

dtypes: float64(5), int64(1), object(1)

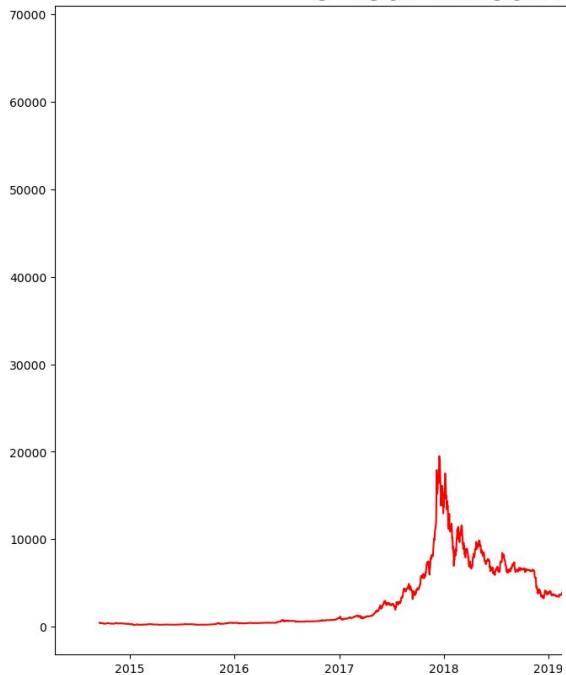
```
bitcoindf = bitcoindf.fillna(method = 'ffill')
```



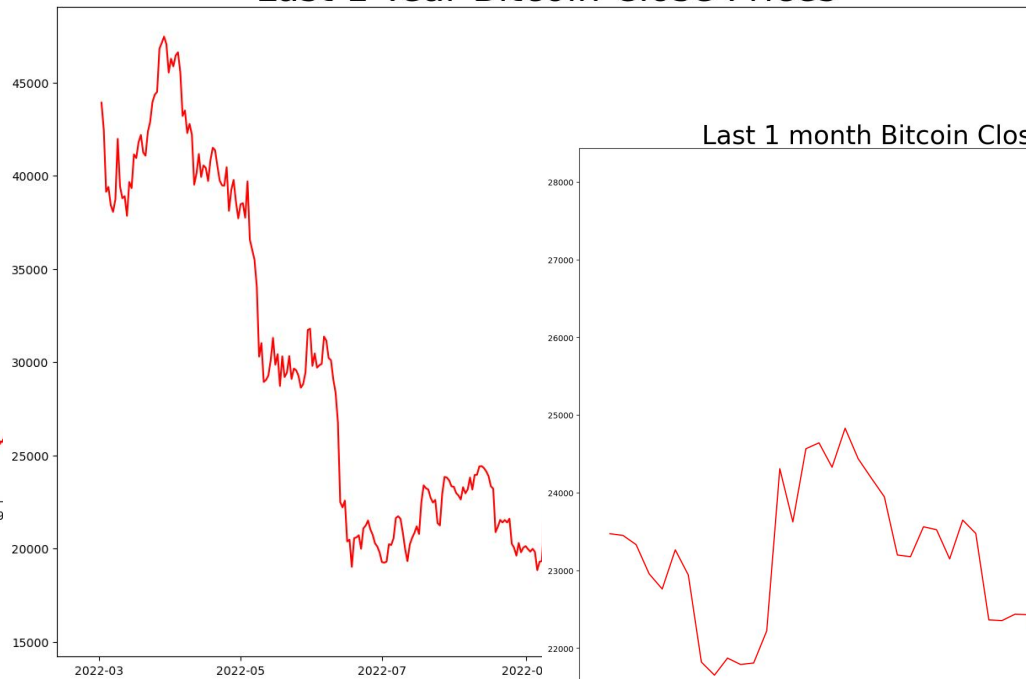
```
bitcoindf['date'] = pd.to_datetime(bitcoindf.date)
```



8-Year Bitcoin Close Price



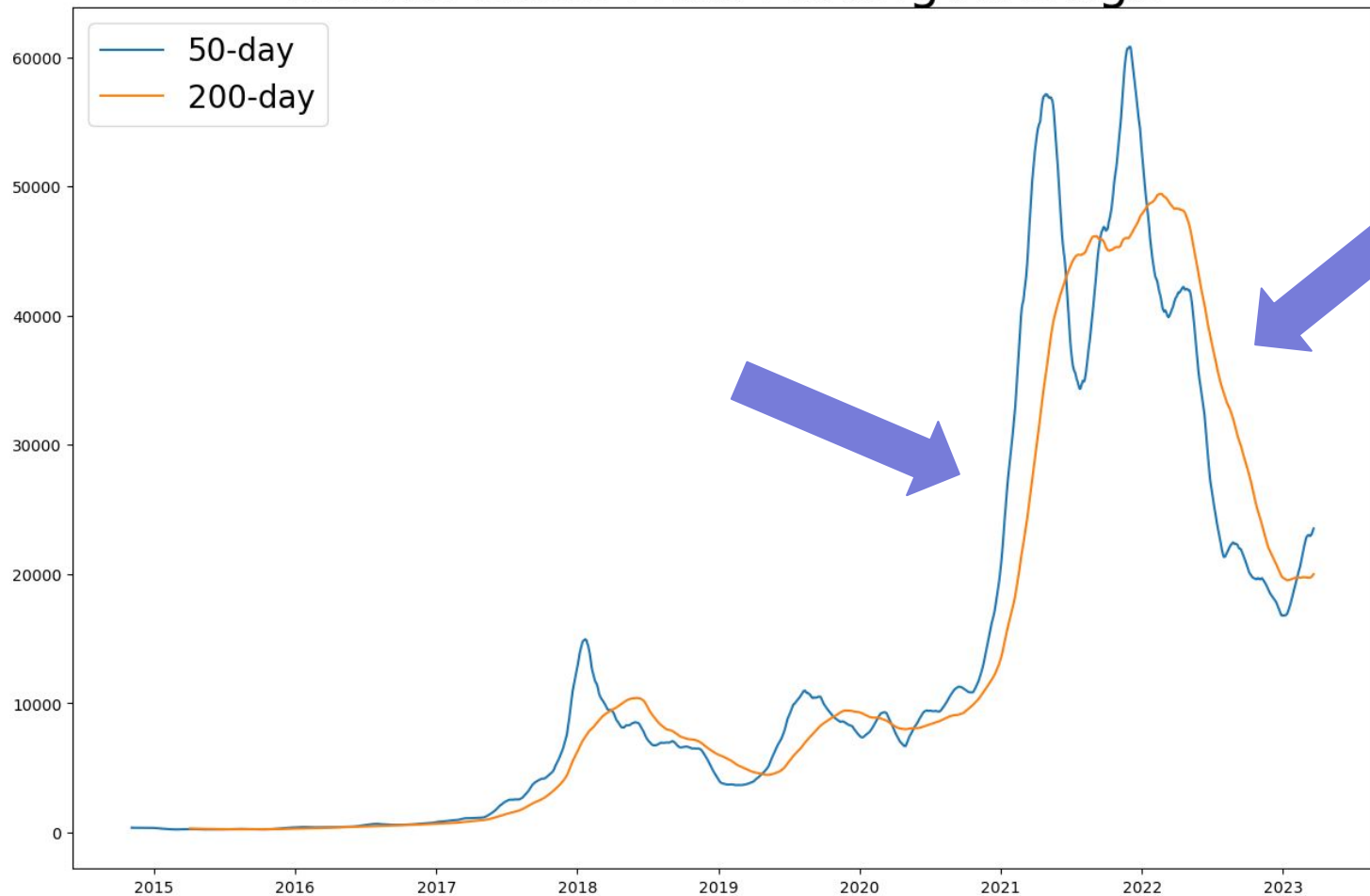
Last 1 Year Bitcoin Close Prices



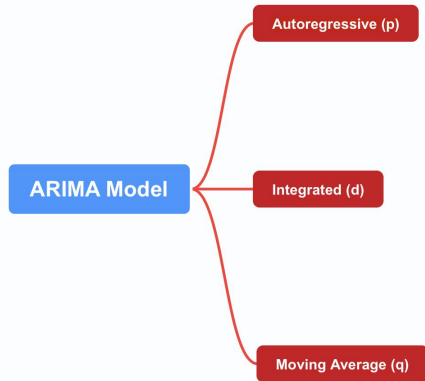
Last 1 month Bitcoin Close Prices



Bitcoin Close Price Moving Average

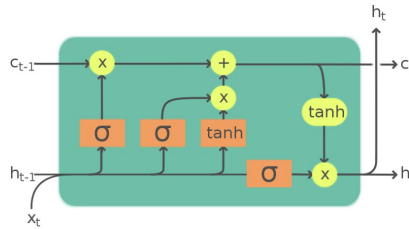


Machine Learning



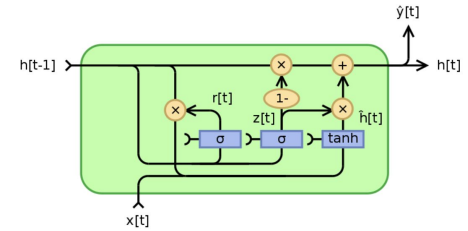
ARIMA

 Statsmodels



LSTM

 Pytorch



GRU

 Pytorch

Non-
Seasonal

Seasonal

SARIMAX Model

p

P

Auto Regression

The number of lag observations to include in the model

d

D

Integrated

The number of times that the raw observations are differenced

q

Q

Moving Average

The size of the moving average window

Preprocessing

Box-Cox Transformation

Seasonal Differentiation

Regular Differentiation

Box-Cox Transformation

$$y_i^{(\lambda)} = \begin{cases} \frac{y_i^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln(y_i) & \text{if } \lambda = 0, \end{cases}$$

Augmented Dickey-Fuller Test

$$y_t = c + \beta t + \alpha y_{t-1} + \phi_1 \Delta Y_{t-1} + \phi_2 \Delta Y_{t-2} \dots + \phi_p \Delta Y_{t-p} + e_t$$

Solve for p-value:

The probability of occurrence under the null hypothesis

```
1 seasonal_decompose(btc_month.close).plot()
2 print('Dickey_Fuller test: p=%f' % adfuller(btc_month.close)[1])
3 plt.show()
```

Dickey_Fuller test: p=0.497078

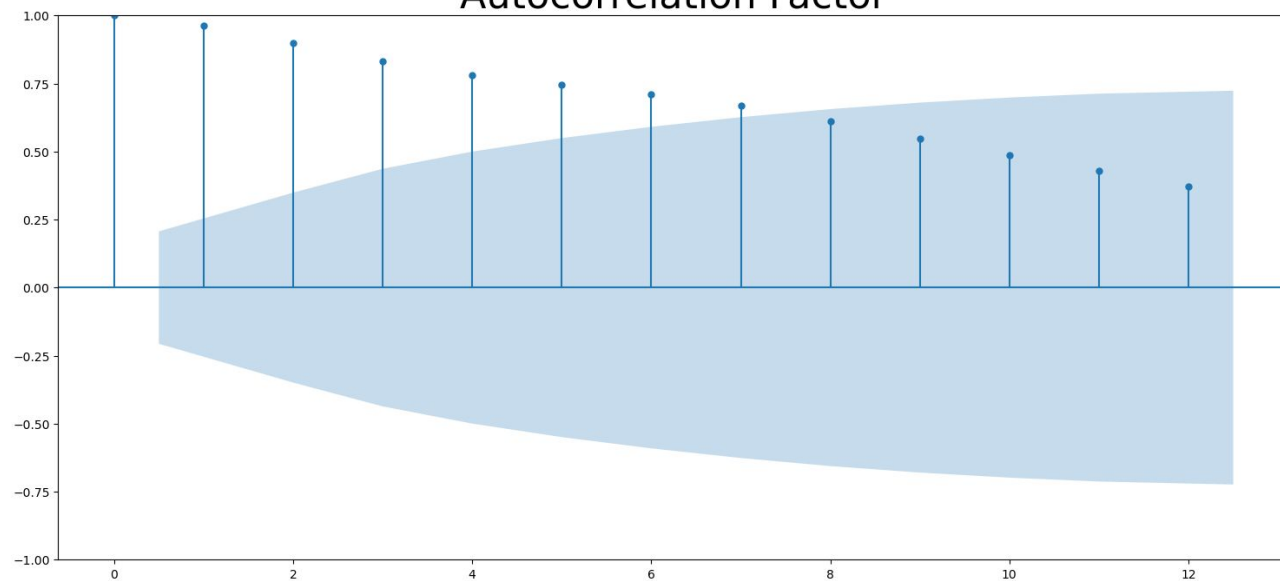
```
1 # Box-Cox Transformations
2 btc_month['close_box'], lambda = stats.boxcox(btc_month.close)
3 print('Dickey_Fuller test: p=%f' % adfuller(btc_month.close_box)[1])
```

Dickey_Fuller test: p=0.661977

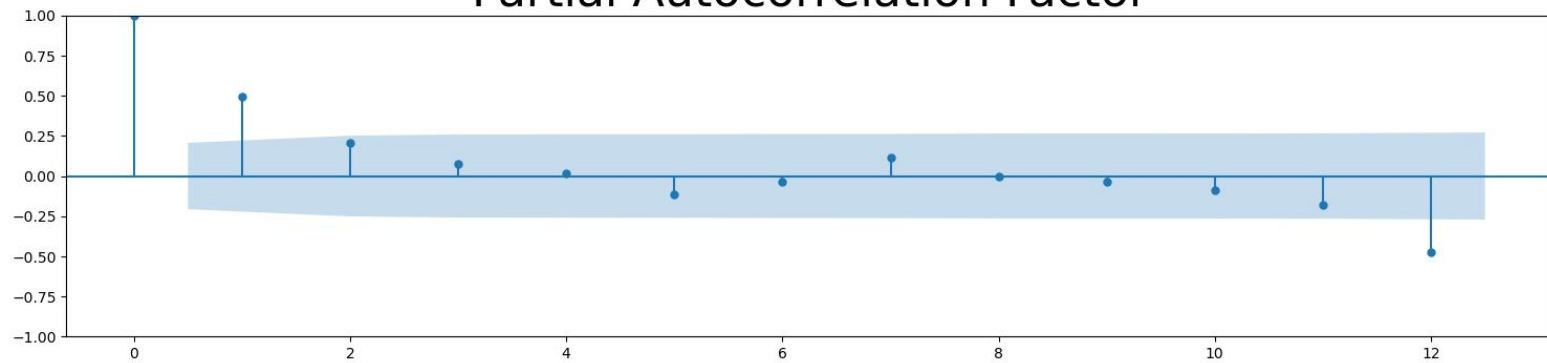
```
1 # Seasonal differentiation (3 months)
2 btc_month['box_diff_seasonal_3'] = btc_month.close_box - btc_month.close_box.shift(3)
3 print('Dickey-Fuller test: p=%f' % adfuller(btc_month.box_diff_seasonal_3[3:])[1])
```

Dickey-Fuller test: p=0.006681

Autocorrelation Factor



Partial Autocorrelation Factor



Parameter Optimizing

```
# Initial approximation of parameters
Qs = range(0, 3)
qs = range(0, 3)
Ps = range(0, 3)
ps = range(0, 3)
D = 1
d = 1
parameters = product(ps, qs, Ps, Qs)
parameters_list = list(parameters)
len(parameters_list)

# Model Selection
results = []
best_aic = float('inf')
warnings.filterwarnings('ignore')
for param in parameters_list:
    try:
        # model = SARIMAX(
        #     btc_month.close_box,
        #     order = (param[0], d, param[1]),
        #     seasonal_order = (param[2], D, param[3], 12)).fit(dispatch = -1)
        model = SARIMAX(
            btc_month.close_box,
            order = (param[0], d, param[1]),
            seasonal_order = (param[2], D, param[3], 4)).fit(dispatch = -1)
    except ValueError:
        print('bad parameter combination: ', param)
        continue
    aic = model.aic
    if aic < best_aic:
        best_model = model
        best_aic = aic
        best_param = param
    results.append([param, model.aic])
```

Best Model

```
SARIMAX Results
=====
Dep. Variable:          close_box    No. Observations:          103
Model:                SARIMAX(1, 1, 0)x(0, 1, [1], 4)    Log Likelihood            -97.002
Date:                  Sun, 26 Mar 2023    AIC                       200.004
Time:                  21:38:31    BIC                       207.759
Sample:                09-30-2014    HQIC                      203.140
                        - 03-31-2023
Covariance Type:                opg
=====
              coef    std err          z      P>|z|      [0.025      0.975]
-----
ar.L1             0.4112     0.078     5.295     0.000     0.259     0.563
ma.S.L4           -0.9984     3.826    -0.261     0.794    -8.496     6.499
sigma2             0.3715     1.406     0.264     0.792    -2.384     3.127
=====
Ljung-Box (L1) (Q):                0.09    Jarque-Bera (JB):                0.60
Prob(Q):                          0.76    Prob(JB):                0.74
Heteroskedasticity (H):            2.15    Skew:                    -0.05
...
=====

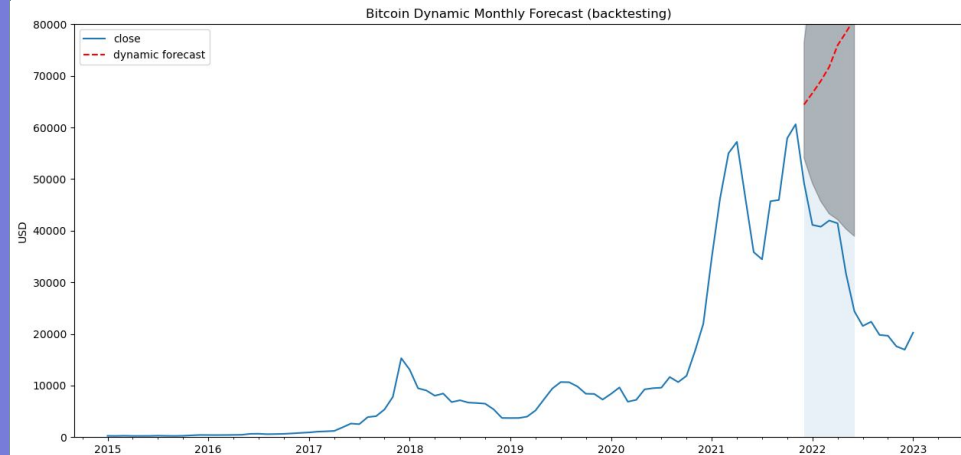
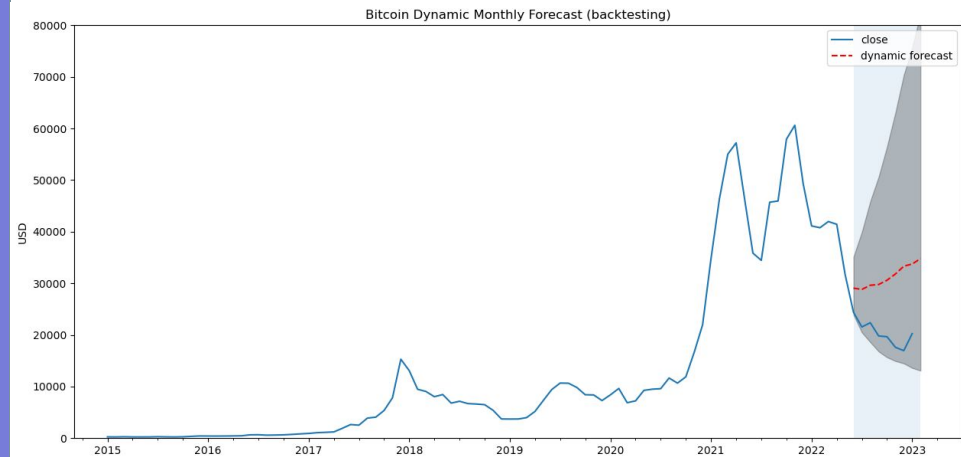
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

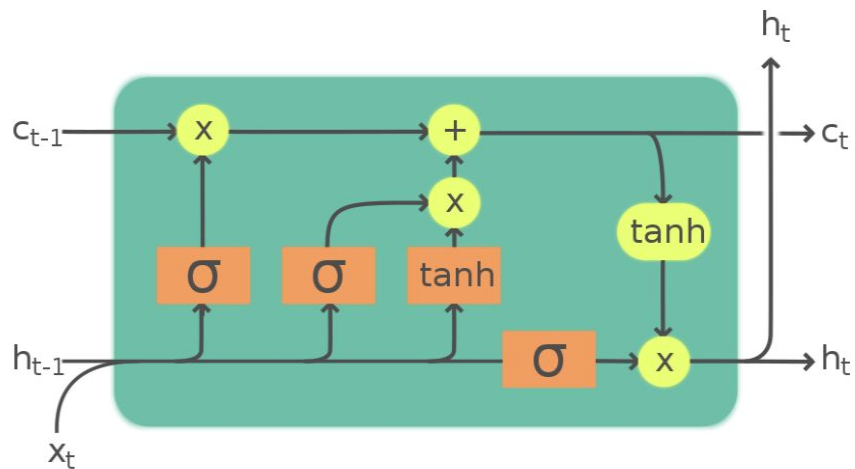
Model Evaluation

```
1 y_forecasted = btc_month2.forecast
2 y_truth = btc_month2['2017-01-01' : '2021-01-01'].close
3
4 # Compute the root mean squared error
5 rmse = np.sqrt(((y_forecasted - y_truth) ** 2).mean())
6 print('Root Mean Squared Error: {}'.format(round(rmse, 2)))
```

Root Mean Squared Error: 1953.7

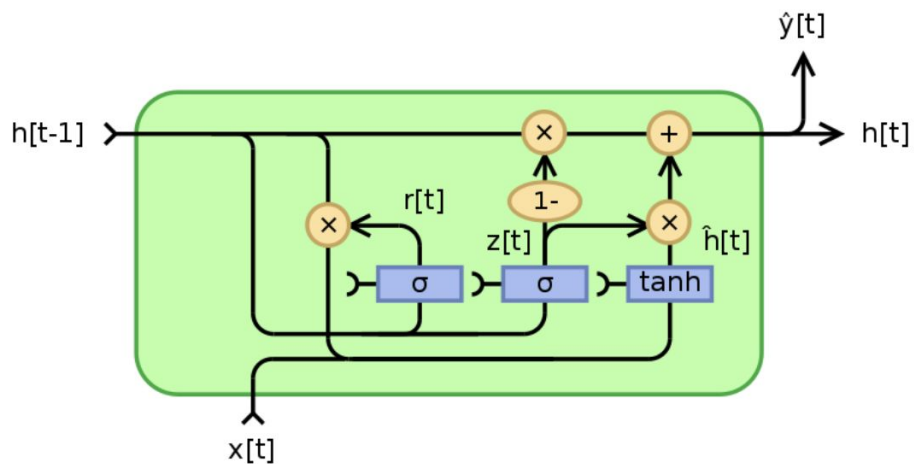
Dynamic Forecast





Long Short-Term Memory

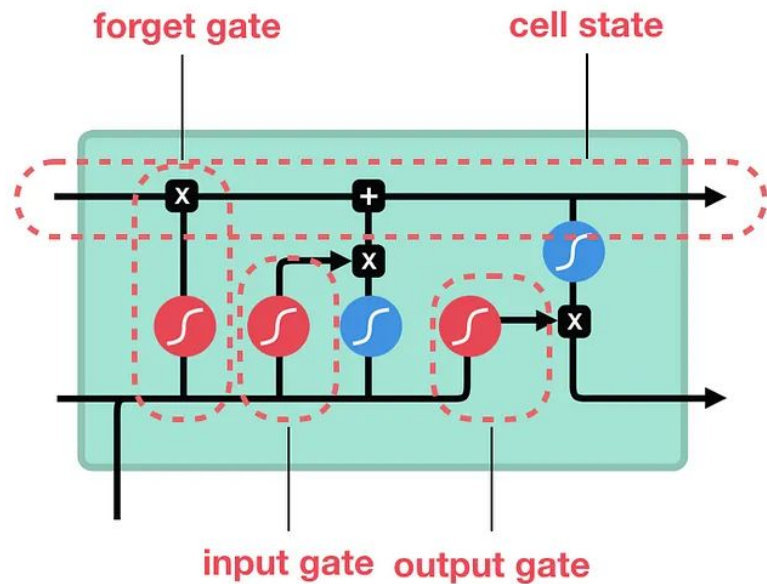
LSTM



Gated Recurrent Units

GRU

LSTM



sigmoid



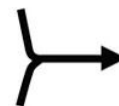
tanh



pointwise
multiplication

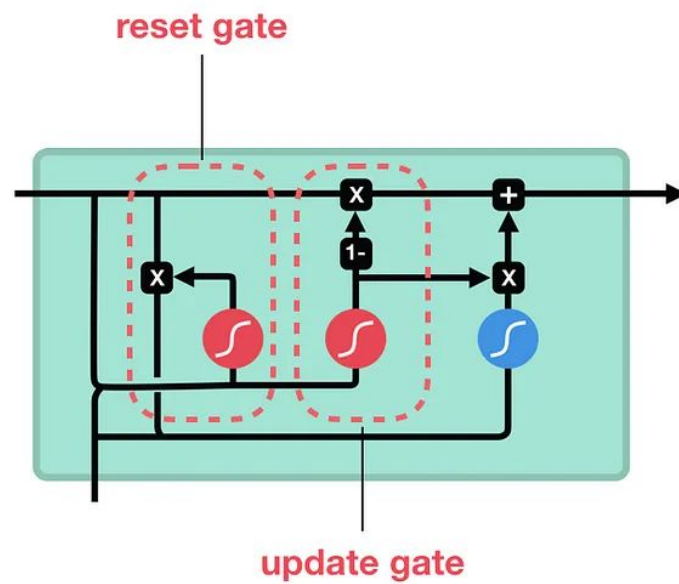


pointwise
addition



vector
concatenation

GRU



update gate

Preprocessing

MinMaxScaler

```
scaler = MinMaxScaler(feature_range = (-1, 1))  
price['close'] = scaler.fit_transform(price['close'].values.reshape(-1, 1))
```



Scikit-Learn

LSTM

```
1 import time
2
3 hist = np.zeros(num_epochs)
4 start_time = time.time()
5 lstm = []
6
7 for t in range(num_epochs):
8     y_train_pred = model(x_train)
9
10     loss = criterion(y_train_pred, y_train_lstm)
11     print("Epoch ", t, "MSE: ", loss.item())
12     hist[t] = loss.item()
13
14     optimiser.zero_grad()
15     loss.backward()
16     optimiser.step()
17
18 training_time = time.time() - start_time
19 print("Training time: {}".format(training_time))
```

Output exceeds the [size limit](#). Open the full output data [in a text editor](#)

```
Epoch 0 MSE: 0.6126973032951355
Epoch 1 MSE: 0.42915019392967224
Epoch 2 MSE: 0.24500030279159546
Epoch 3 MSE: 0.08170756697654724
Epoch 4 MSE: 0.44214609265327454
Epoch 5 MSE: 0.15407656133174896
Epoch 6 MSE: 0.07960957288742065
Epoch 7 MSE: 0.12146790325641632
Epoch 8 MSE: 0.15753395855426788
Epoch 9 MSE: 0.17221978306770325
Epoch 10 MSE: 0.17004160583019257
Epoch 11 MSE: 0.15656591951847076
Epoch 12 MSE: 0.13644909858703613
Epoch 13 MSE: 0.11375288665294647
Epoch 14 MSE: 0.09221626073122025
Epoch 15 MSE: 0.07505214214324951
Epoch 16 MSE: 0.06414283812046051
Epoch 17 MSE: 0.05887597054243088
Epoch 18 MSE: 0.05581199377775192
Epoch 19 MSE: 0.05115620791912079
Epoch 20 MSE: 0.04398057237267494
Epoch 21 MSE: 0.03420542925596237
Epoch 22 MSE: 0.027142776176333427
Epoch 23 MSE: 0.022492844611406326
Epoch 24 MSE: 0.01789293996989727
...
Epoch 97 MSE: 0.0011399909853935242
Epoch 98 MSE: 0.0011199767468497157
Epoch 99 MSE: 0.001101750647649169
Training time: 10.477274179458618
```

GRU

```
1 hist = np.zeros(num_epochs)
2 start_time = time.time()
3 gru = []
4
5 for t in range(num_epochs):
6     y_train_pred = model(x_train)
7
8     loss = criterion(y_train_pred, y_train_gru)
9     print("Epoch ", t, "MSE: ", loss.item())
10    hist[t] = loss.item()
11
12    optimiser.zero_grad()
13    loss.backward()
14    optimiser.step()
15
16 training_time = time.time() - start_time
17 print("Training time: {}".format(training_time))
```

Output exceeds the [size limit](#). Open the full output data [in a text editor](#)

```
Epoch 0 MSE: 0.8184217214584351
Epoch 1 MSE: 0.2846358120441437
Epoch 2 MSE: 0.08693624287843704
Epoch 3 MSE: 0.18822167813777924
Epoch 4 MSE: 0.1494576334953308
Epoch 5 MSE: 0.07656984031200409
Epoch 6 MSE: 0.0659845620393753
Epoch 7 MSE: 0.08769886195659637
Epoch 8 MSE: 0.09999230504003595
Epoch 9 MSE: 0.08909653127193451
Epoch 10 MSE: 0.06190858781337738
Epoch 11 MSE: 0.03424866870045662
Epoch 12 MSE: 0.023775583133101463
Epoch 13 MSE: 0.03569463640451431
Epoch 14 MSE: 0.0448136106133461
Epoch 15 MSE: 0.03066970594227314
Epoch 16 MSE: 0.009500355459749699
Epoch 17 MSE: 0.002467151964083314
Epoch 18 MSE: 0.010785665363073349
Epoch 19 MSE: 0.020730337128043175
Epoch 20 MSE: 0.020765794441103935
Epoch 21 MSE: 0.012438496574759483
Epoch 22 MSE: 0.005496680270880461
Epoch 23 MSE: 0.00562502583488822
Epoch 24 MSE: 0.008937880396842957
...
Epoch 97 MSE: 0.0004731537774205208
Epoch 98 MSE: 0.0004706674662884325
Epoch 99 MSE: 0.0004675117670558393
Training time: 9.262976884841919
```

Results (LSTM)



Results (GRU)



Results (LSTM)



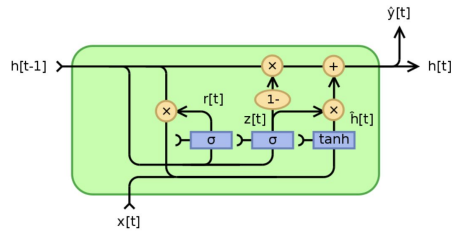
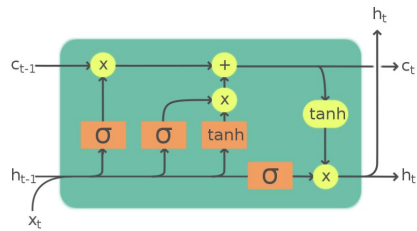
Results (GRU)



	LSTM	GRU
Train RMSE	1118.404444	728.540107
Test RMSE	42.371615	32.661090
Train Time	10.477274	9.262977

GRU
performed
better than
LSTM

Data-driven Insights



Recommendation

LSTM

 Pytorch

GRU

 Pytorch

Thank You!

References

1. Wikipedia contributors. (2023). Cryptocurrency. Wikipedia. <https://en.wikipedia.org/wiki/Cryptocurrency>
2. Wikipedia contributors. (2023a). Bitcoin. Wikipedia. <https://en.wikipedia.org/wiki/Bitcoin>
3. pandas - Python Data Analysis Library. (n.d.). <https://pandas.pydata.org/>
4. Nguyen, B. (2021, December 29). End-to-End Time Series Analysis and Forecasting: a Trio of SARIMAX, LSTM and Prophet (Part 1). Medium.
<https://towardsdatascience.com/end-to-end-time-series-analysis-and-forecasting-a-trio-of-sarimax-lstm-and-prophet-part-1-306367e57db8>
5. statsmodels.tsa.arima.model.ARIMA — statsmodels. (n.d.).
<https://www.statsmodels.org/dev/generated/statsmodels.tsa.arima.model.ARIMA.html>
6. Wikipedia contributors. (2023a). Power transform. Wikipedia.
https://en.wikipedia.org/wiki/Power_transform#Example

References (continued)

7. Wikipedia contributors. (2022). Augmented Dickey-Fuller test. Wikipedia.

[https://en.wikipedia.org/wiki/Augmented Dickey%E2%80%93Fuller test](https://en.wikipedia.org/wiki/Augmented_Dickey%E2%80%93Fuller_test)

8. LSTM — PyTorch 2.0 documentation. (n.d.).

<https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html>

9. GRU — PyTorch 2.0 documentation. (n.d.).

<https://pytorch.org/docs/stable/generated/torch.nn.GRU.html>

10. Phi, M. (2020, June 28). Illustrated Guide to LSTM's and GRU's: A step by step explanation. Medium.

<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

11. sklearn.preprocessing.MinMaxScaler. (n.d.). Scikit-learn.

<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>