# SC1015 Mini-Project

## Cryptocurrency Stock Price Time Series Analysis

SC1015 Group 3
Wang Yangming
Wang Anqi
Yang Ziyu

# Contents

# Problems and Goals

**Decentralized**

**Digital Currency**

World's First Cryptocurrency

# Goal

# Machine Learning

# The Dataset

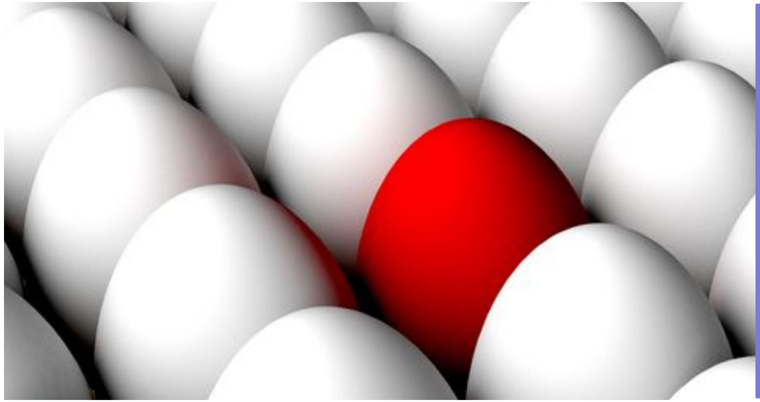

70000
60000
50000
40000
30000
20000
10000
0
2015
2016
2017
2018
2019
2020
2021
2022
2023
yahoo!
finance

# Exploratory Data Analysis

# 3108 Days

2014-9-17

2023-3-21



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')
```

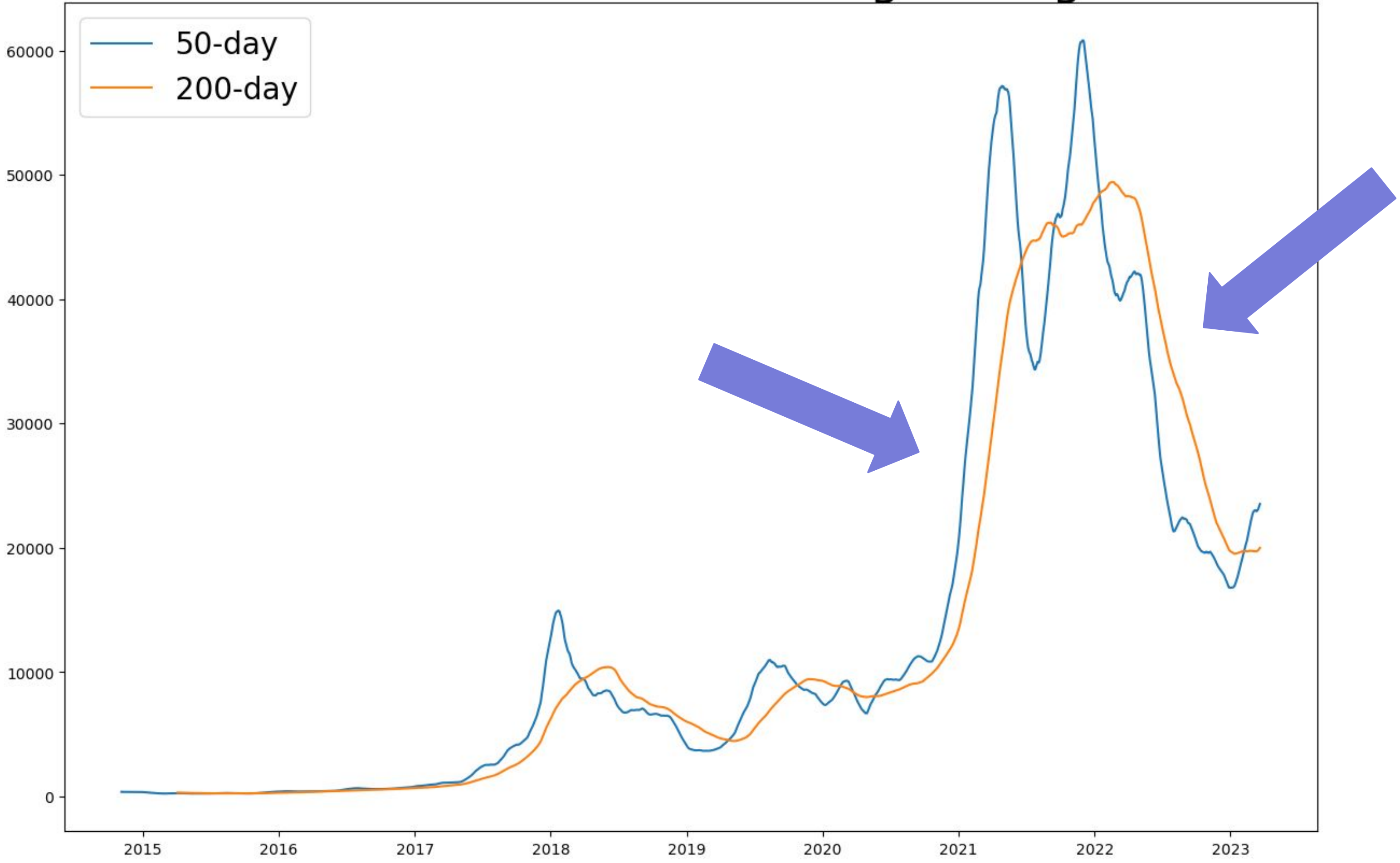
| | | |
|---|---|---|
| Num<br>NaN | → | Num<br>Num |

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

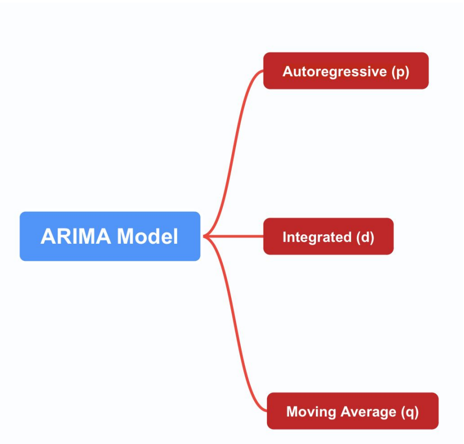
Object → Datetime

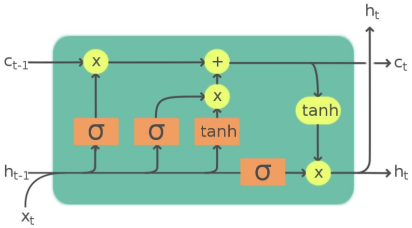

8-Year Bitcoin Close Price
70000
60000
50000
40000
30000
20000
10000
0
2015
2016
2017
2018
2019
Last 1 Year Bitcoin Close Prices
45000
40000
35000
30000
25000
20000
15000
2022-03
2022-05
2022-07
Last 1 month Bitcoin Close Prices
28000
27000
26000
25000
24000
23000
22000
21000
20000
2023-02-01
2023-02-08
2023-02-15
2023-02-22
2023-03-01
2023-03-08
2023-03-15
2023-03-22

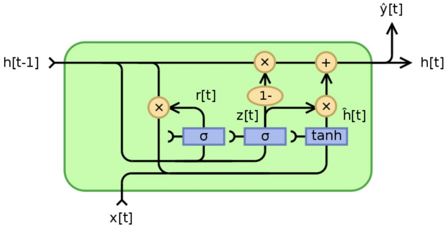
# Bitcoin Close Price Moving Average



# Machine Learning




# ARIMA

Statsmodels

# LSTM

Pytorch

# GRU

Pytorch

# SARIMAX Model

| Non-Seasonal | Seasonal | |
|---|---|---|
| p | P | **Auto Regression**<br>The number of lag observations ot include in the model |
| d | D | **Integrated**<br>The number of times that the raw observations are differenced |
| q | Q | **Moving Average**<br>The size of the moving average window |

# Preprocessing

Box-Cox Transformation

Seasonal Differentiation

Regular Differentiation

# Box-Cox Transformation

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

# Augmented Dickey-Fuller Test

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

**Solve for p-value:**

**The probability of occurrence under the null hypothesis**

Statsmodels

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

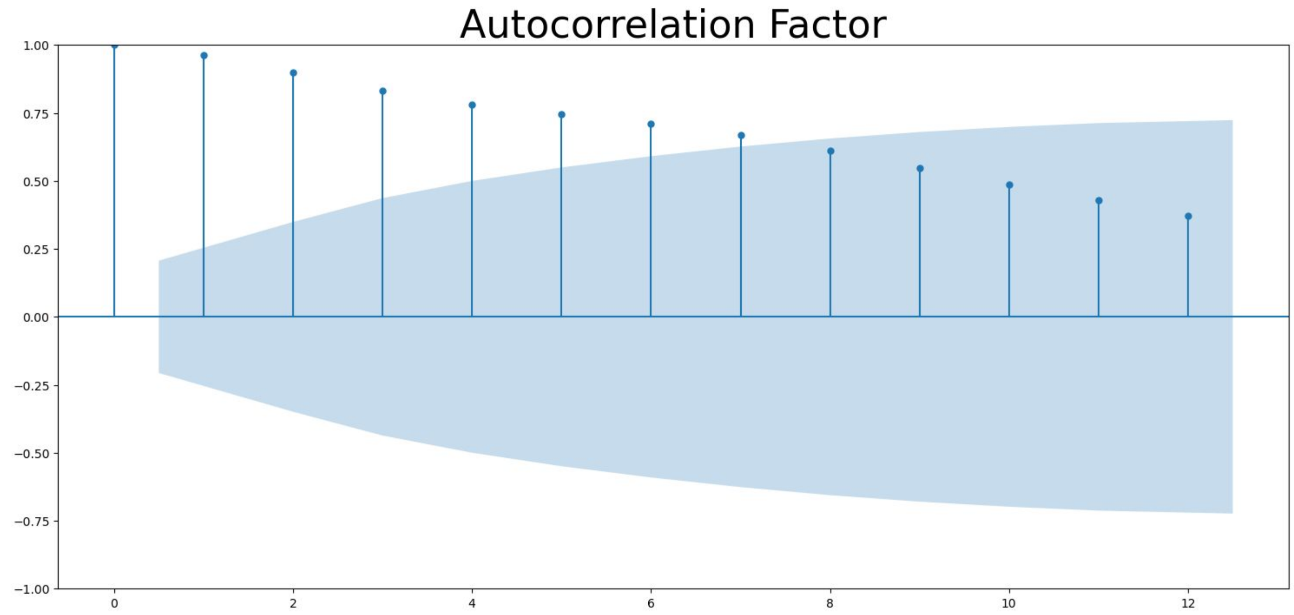
```
Dickey_Fuller test: p=0.497078
```

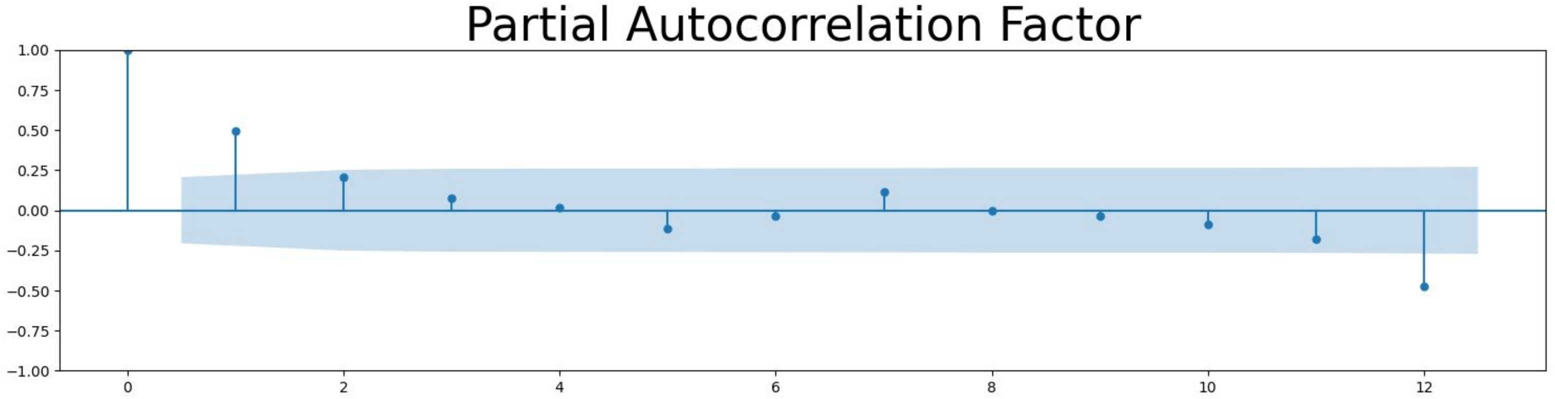
```python
# Box-Cox Transformations
btc_month['close_box'], lmbda = stats.boxcox(btc_month.close)
print('Dickey_Fuller test: p=%f' % adfuller(btc_month.close_box)[1])
```

```
Dickey_Fuller test: p=0.661977
```

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

```
Dickey-Fuller test: p=0.006681
```


Autocorrelation Factor
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12


Partial Autocorrelation Factor
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12

# Parameter Optimizing

```python
# 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(disp = -1)
        model = SARIMAX(
            btc_month.close_box,
            order = (param[0], d, param[1]),
            seasonal_order = (param[2], D, param[3], 4)).fit(disp = -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

```
y_forecasted = btc_month2.forecast
y_truth = btc_month2['2017-01-01' : '2021-01-01'].close

# Compute the root mean squared error
rmse = np.sqrt(((y_forecasted - y_truth) ** 2).mean())
print('Root Mean Squared Error: {}'.format(round(rmse, 2)))
```

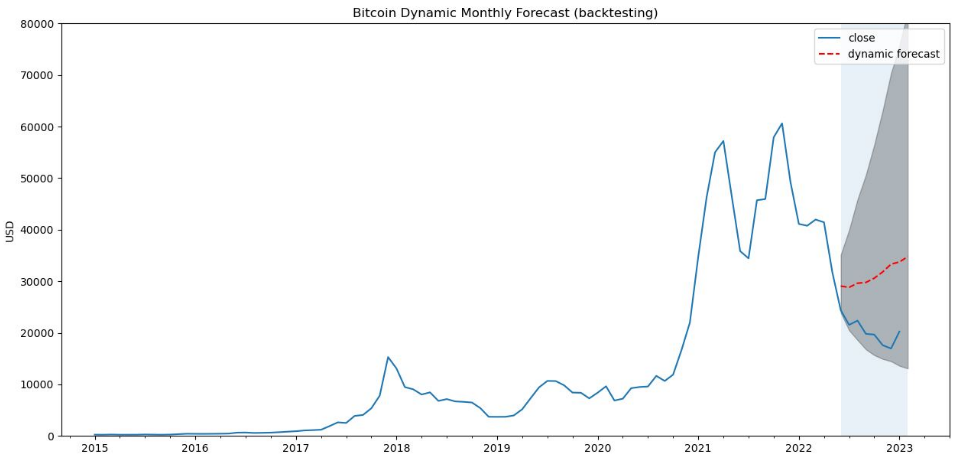
```
Root Mean Squared Error: 1953.7
```

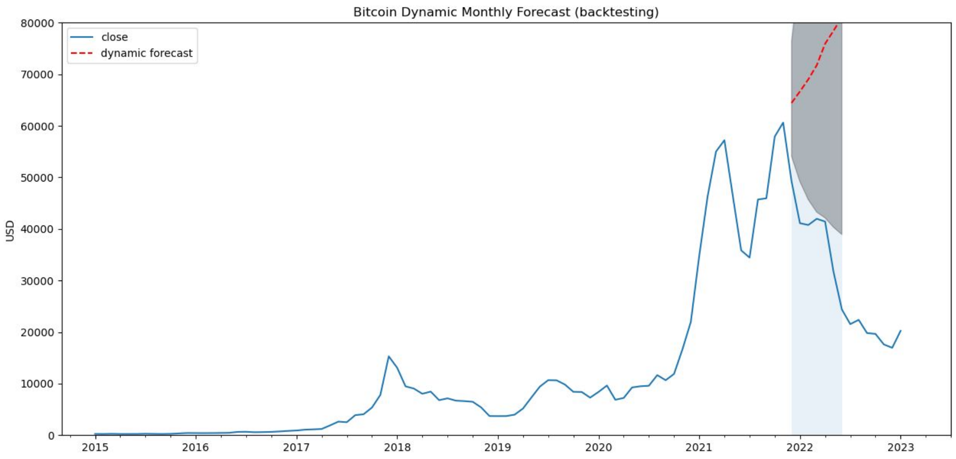
# Model Evaluation

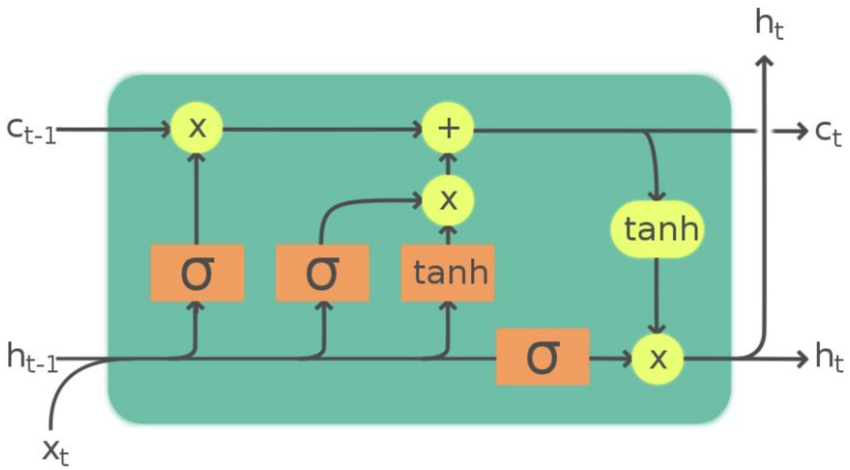
```python
y_forecasted = btc_month2.forecast
y_truth = btc_month2['2017-01-01' : '2021-01-01'].close

# Compute the root mean squared error
rmse = np.sqrt(((y_forecasted - y_truth) ** 2).mean())
print('Root Mean Squared Error: {}'.format(round(rmse, 2)))
```

```
Root Mean Squared Error: 1953.7
```

# Dynamic Forecast

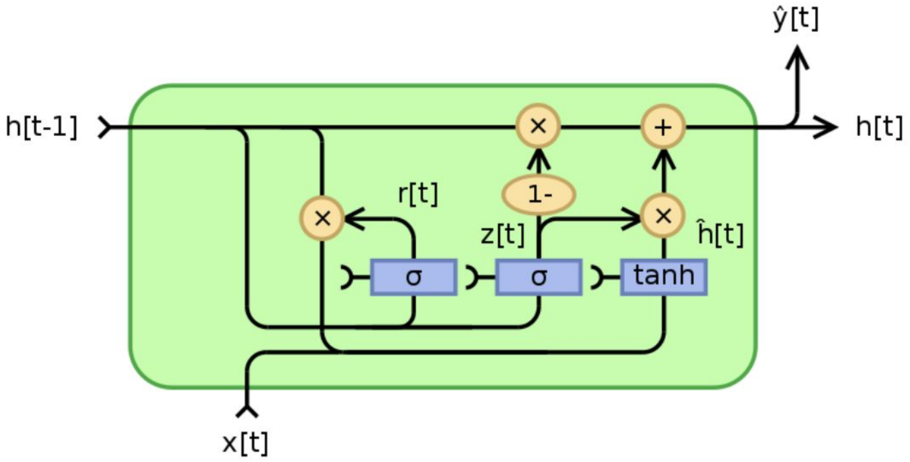




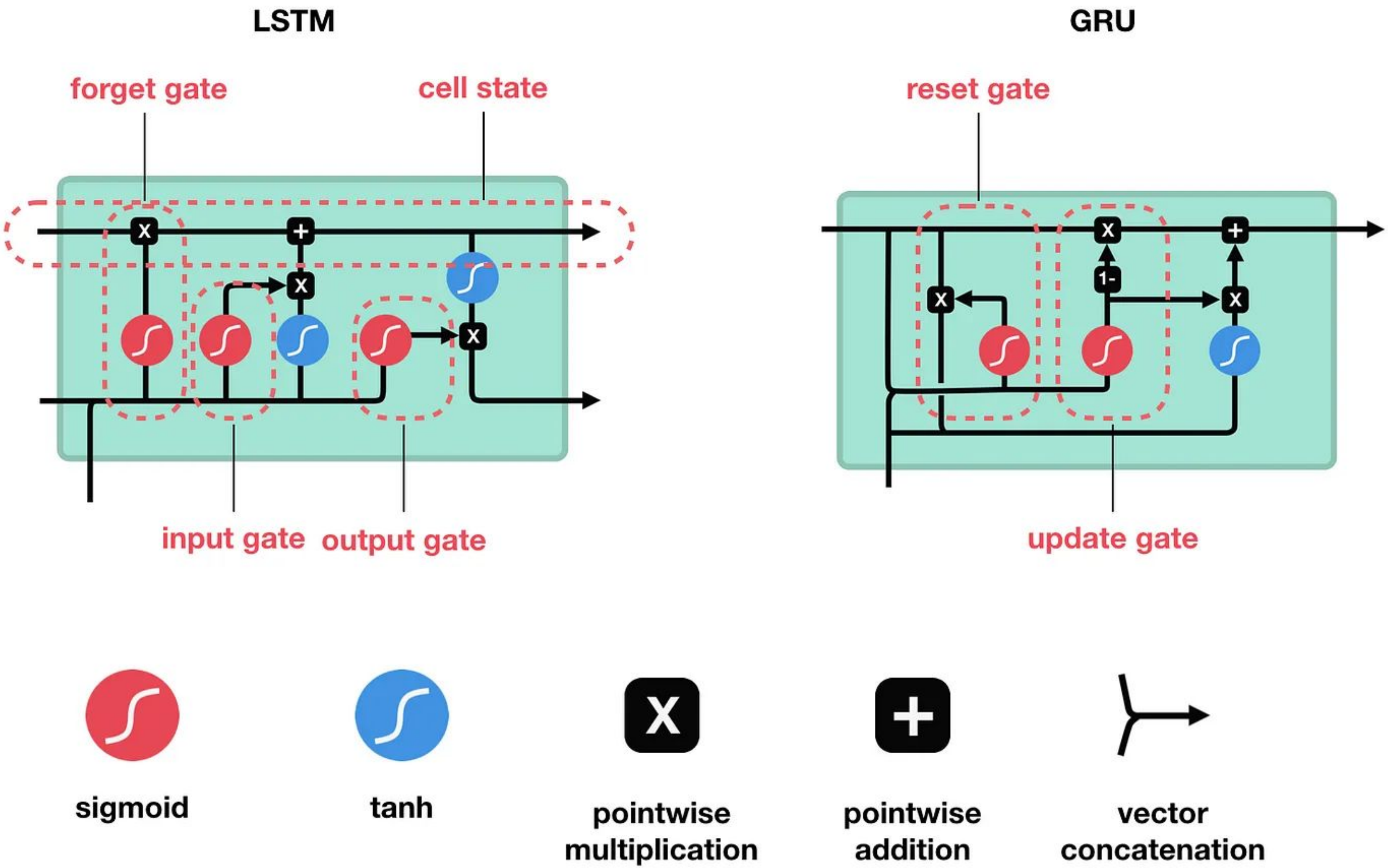

# Long Short-Term Memory

# LSTM




Gated Recurrent Units

# GRU


LSTM
forget gate
cell state
input gate
output gate
GRU
reset gate
update gate
1-
sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation

# Preprocessing

## MinMaxScaler

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

Scikit-Learn

# LSTM

```python
import time

hist = np.zeros(num_epochs)
start_time = time.time()
lstm = []

for t in range(num_epochs):
    y_train_pred = model(x_train)

    loss = criterion(y_train_pred, y_train_lstm)
    print("Epoch ", t, "MSE: ", loss.item())
    hist[t] = loss.item()

    optimiser.zero_grad()
    loss.backward()
    optimiser.step()

training_time = time.time() - start_time
print("Training time: {}".format(training_time))
```

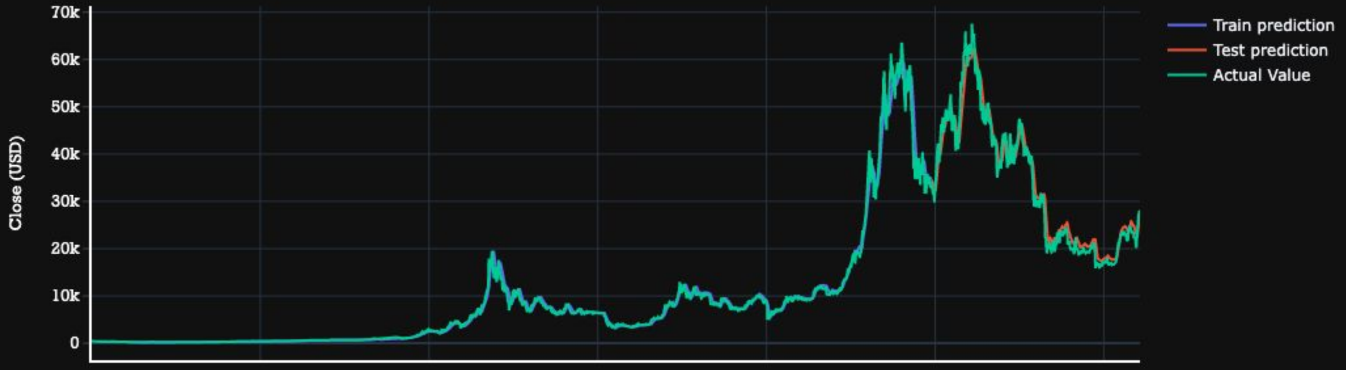
```
Output exceeds the size limit. Open the full output data in a text editor
Epoch  0 MSE:  0.6126973032951355
Epoch  1 MSE:  0.42915019392967224
Epoch  2 MSE:  0.24500030279159546
Epoch  3 MSE:  0.08170756697654724
Epoch  4 MSE:  0.44214609265327454
Epoch  5 MSE:  0.15407656133174896
Epoch  6 MSE:  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

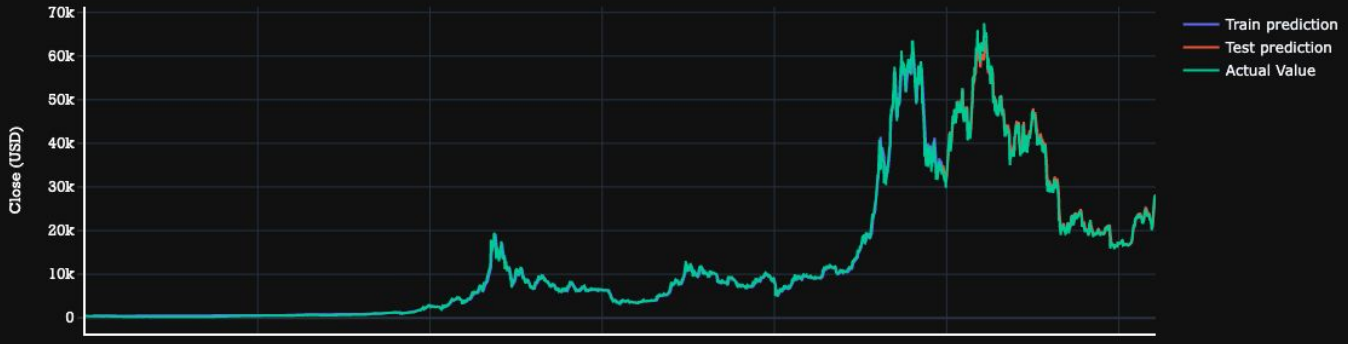
```python
hist = np.zeros(num_epochs)
start_time = time.time()
gru = []

for t in range(num_epochs):
    y_train_pred = model(x_train)

    loss = criterion(y_train_pred, y_train_gru)
    print("Epoch ", t, "MSE: ", loss.item())
    hist[t] = loss.item()

    optimiser.zero_grad()
    loss.backward()
    optimiser.step()

training_time = time.time() - start_time
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:  0.0999923050403595
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:  0.030669705942227314
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
```

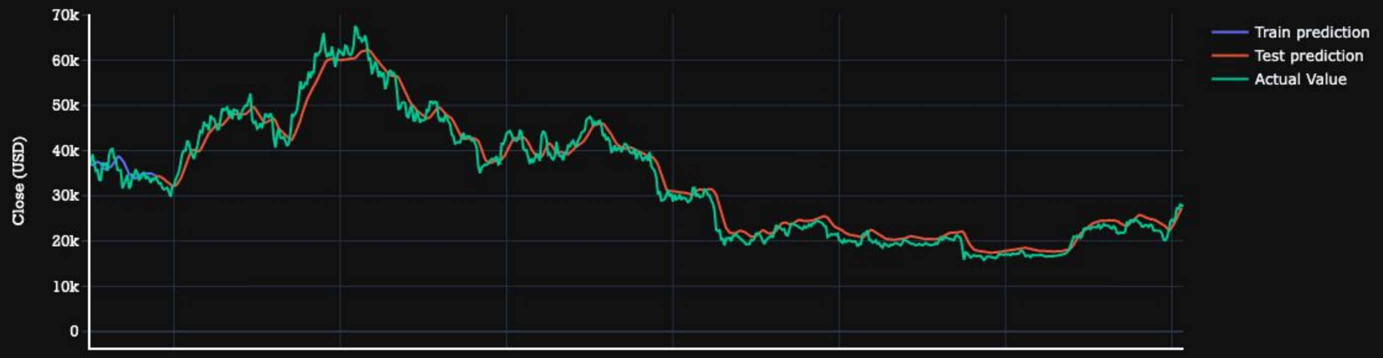
# Results (LSTM)




# Results (GRU)



# Results (LSTM)

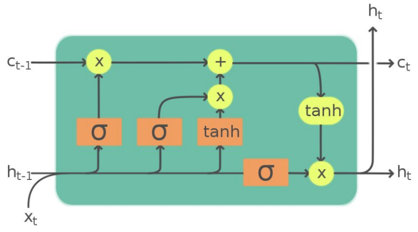


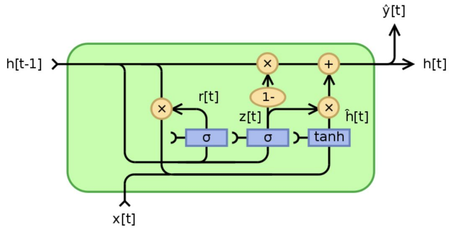

# Results (GRU)

| | 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




# LSTM

Pytorch

# GRU

Pytorch

# Recommendation

# Thank You!

# References