Shock Relaxed Approaches With Refactored ARIMA Model

Time-Series Analysis Final Group Project

Instructor: Hope Han
Subject: Time-Series Analysis [FIA52301]
Group: Group 12
Name: Daehyuk Bu
Jae Yeon Park
Song Kim
Taehoon Kim

Introduction

i. Background

People have always had a desire to fully understand the financial market, and research in this regard has been ongoing for a long time. Despite ongoing research, the complexity of the financial market remains a challenge due to various influencing factors. However, various macro and micro factors have made it challenging for humans to comprehend the financial market entirely, and research in this field is still ongoing.

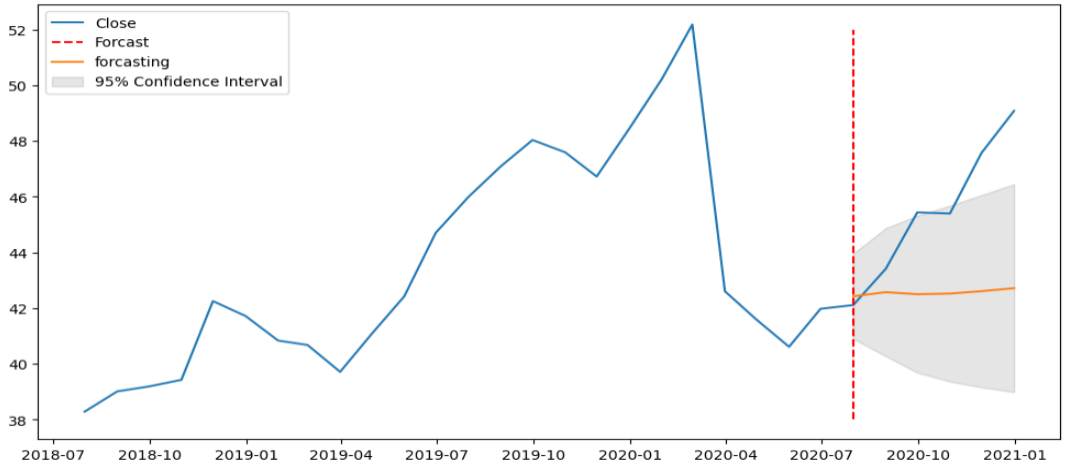
Especially in recent years, individual stock investment has shown a steady increase, garnering significant attention. This has underscored the importance of accurately analyzing stocks and highlighted the need for more precise prediction methods. In the stock market, research is underway to find price patterns through the reflection of all market information, utilizing various analytical techniques to predict stock indices. However, for various reasons, stock prices often exhibit unpredictable volatility, making it challenging to create models for predicting stock prices. In response to this challenge, efforts are being made to develop models that enhance stable stock price predictions.

ii. Motivation

We have contemplated the impact of shocks as a factor disrupting stock price predictions. These shocks can arise from various factors such as global economic trends, political events, and corporate performances. Shocks often surprise market participants, making predictions challenging and sensitively influencing the financial system. The ARIMA model we learned in class tends to be responsive to significant deviations from steady trends, such as shocks. In response to this, we aimed to devise a complementary model to address these problems. Our goal is to make our predictions stronger and contribute to making better models for understanding the ups and downs of the financial market.

Methodology

i. *ARIMA Model*



<Figure.1 Vanilla ARIMA>

As evident from the illustration above, ARIMA struggles to provide accurate predictions, attributed to inherent characteristics of the model. The following factors shed light on why ARIMA may fall short in forecasting accuracy:

Stationarity Assumption

ARIMA assumes data stationarity, expecting statistical properties to remain constant over time. Yet, real-world time series data frequently deviates from this, posing forecasting inaccuracies. Market dynamics and economic shifts challenge the requirement for stationarity.

Linear Relationships

ARIMA primarily relies on linear relationships, limiting its ability to capture complex nonlinear patterns in dynamic financial systems. This constraint can hinder its effectiveness in handling nuanced and evolving time series data.

Sensitivity to Outliers

ARIMA's sensitivity to outliers, particularly its strong response to significant deviations like shocks in stock prices, presents a notable challenge. This heightened responsiveness can result in forecasting inaccuracies, particularly in the face of unpredictable market dynamics.

Considering these factors, ARIMA faces challenges in making precise predictions due to the non-stationary nature of real-world time series data, limitations of linear relationships, and sensitivity to outliers. It's evident that ARIMA may not be the optimal solution for forecasting in dynamic financial environments. Exploring complementary models is imperative to enhance accuracy.

ii. VAE-Outlier Detection (Deep Learning-based approach)

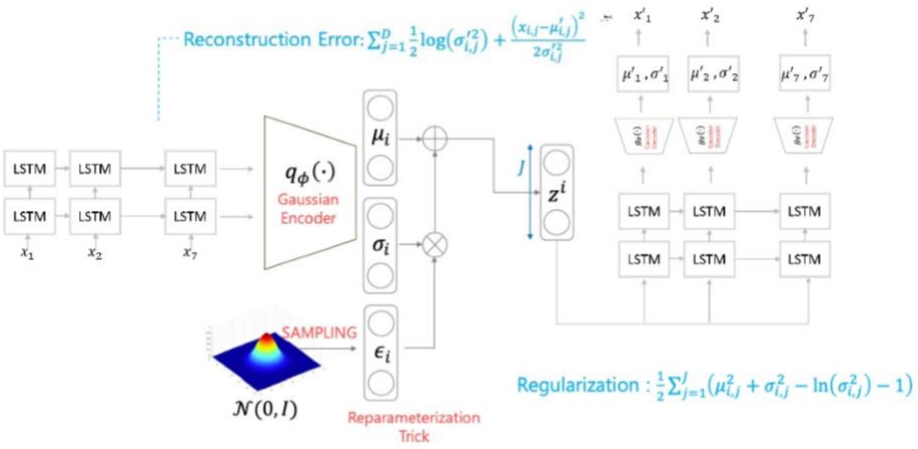
In this section, we will discuss the application of a Variational Autoencoder (VAE) to analyze stock prices. VAE is a type of neural network that helps capture patterns and anomalies in data. To simplify, think of it like a tool that learns and generates similar data patterns.

Dataset

Our dataset consists of daily closing prices from January 4, 2000, to December 31, 2020. We organized the data into sequences of 7-day time steps, resulting in a format of (5275, 7, 1). This sequence format is beneficial for analyzing stock prices over a short-term period.

VAE Architecture

Our VAE structure includes an LSTM layer processing the 7 by 1 sequence, followed by a Gaussian Encoder network extracting mean and variance from the Latent Space Z. The data then undergoes another LSTM layer and a Gaussian Decoder Network for reconstruction. We employ a loss function to measure reconstruction and embedding into the Latent Space, guiding the training process.



<*Figure.2 VAE Architecture*>

Anomaly Detection

Anomalies, or unusual data points, are identified by comparing the actual X with the reconstructed X'. If the difference surpasses the 99th percentile, we label it as an anomaly. This method aligns with anomalies detected by the DBSCAN method.

Statistical Validation

To test the significance of anomaly points, we conducted a Dummy Variable Regression Analysis. This involves assigning a value of 1 to anomaly days and 0 otherwise. Table 1 contains the statistical results of dummy regression. The results, with a T value of 3.573 and P

values of 0.0003, signifying statistical significance.

	Coef	T
Intercept	-.0049(.01)	1.100
Anomalies indicator(d)	-.1599(.01)***	3.573

$*p < .10$. $**p < .05$. $*p < .01$. Notes: Standard errors are in parentheses. (d) = dummy variables.

<Table.1 Regression Result for Dummy Analysis: Anomalies or not >

In summary, our application of VAE to stock prices involves transforming data into sequences, training the VAE model, and detecting anomalies using statistical validation methods. This approach provides insights into unusual stock behavior, contributing to a more nuanced understanding of stock market dynamics.

iii. DBSCAN (Density-based approach) Validation

For validating VAE-detected outliers, we leveraged DBSCAN, a robust algorithm highlighting data points significantly deviating from the norm. Its unique approach based on density distinguishes between dense and sparse regions, making it a powerful tool for outlier detection.

DBSCAN Algorithm

DBSCAN, or Density-Based Spatial Clustering of Applications with Noise, is a versatile clustering algorithm widely used for outlier detection. Unlike methods like K-means, DBSCAN doesn't demand the upfront specification of cluster numbers. Operating on data point density, it adeptly discerns between dense and sparse regions. What sets DBSCAN apart is its proficiency in detecting arbitrarily shaped clusters, a unique capability often elusive to heuristic methods. This distinct feature makes DBSCAN a potent and widely applicable tool in outlier detection scenarios.

How DBSCAN Works

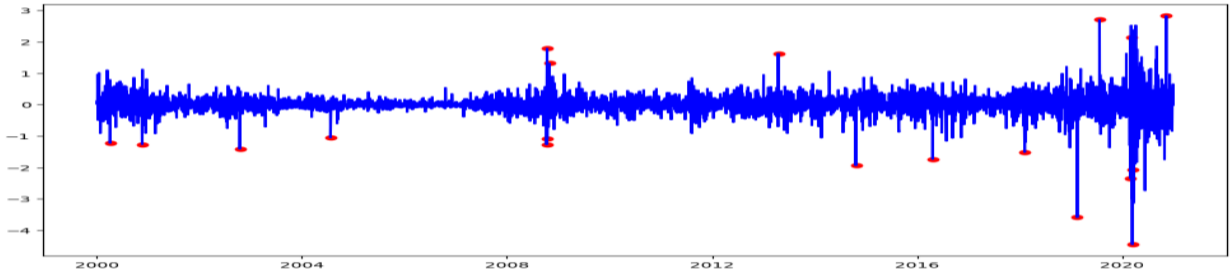
1. Initialization: The algorithm starts by selecting an unvisited data point.
2. Density Check Examines the ε-neighborhood to check for enough neighboring points.
3. Cluster Formation: Initiates a new cluster if the ε-neighborhood is sufficiently dense, recursively adding neighbors.
4. Iterative Process: The algorithm iteratively identifies and expands clusters while labeling noise points that do not meet the density criteria.
5. Assignment of Non-core Points: Points lacking sufficient neighbors to form a dense cluster are assigned to the nearest cluster within an ε-neighborhood or labeled as noise.
6. Completion: Continues until all points are visited and appropriately assigned.

Insight

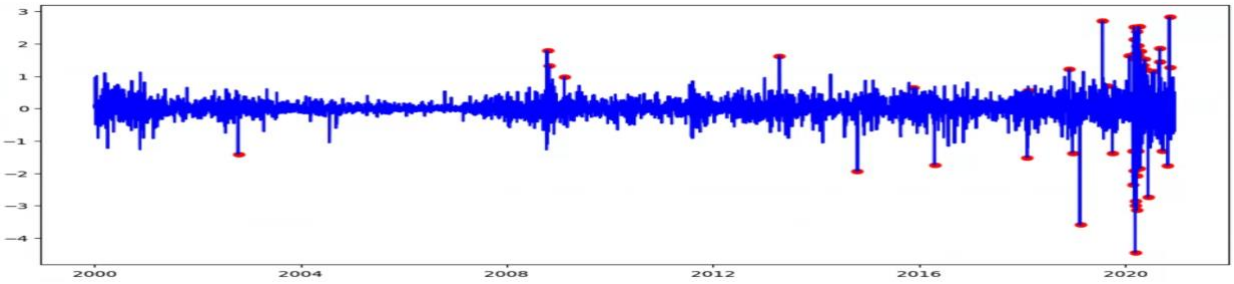
DBSCAN's adaptability to varying data densities and cluster shapes makes it suitable for handling intricate datasets. By dynamically assessing the density of data points, DBSCAN ensures a thorough identification of outliers, enhancing the credibility of our analysis.

Comparison of Outlier Detection

Visual inspection reveals significant overlap in outlier results between DBSCAN and VAE, showcasing effective anomaly identification alignment. VAE demonstrates consistency with the traditional DBSCAN algorithm in pinpointing outliers.



<Figure.3 Anomalies Detection from DBSCAN>



<Figure.4 Anomalies Detection from VAE>

Support for VAE Algorithm

Consistent results between DBSCAN and VAE for outlier detection affirm VAE's reliability. Dummy analysis underscores the significant impact of VAE-detected outliers in regression, emphasizing their statistical importance.

	Coef	Std err	T	P value
constant	0.0049	0.004	1.100	0.271
anomaly	0.1599	0.045	3.573	0.000

<Table.2 Dummy Statistics >

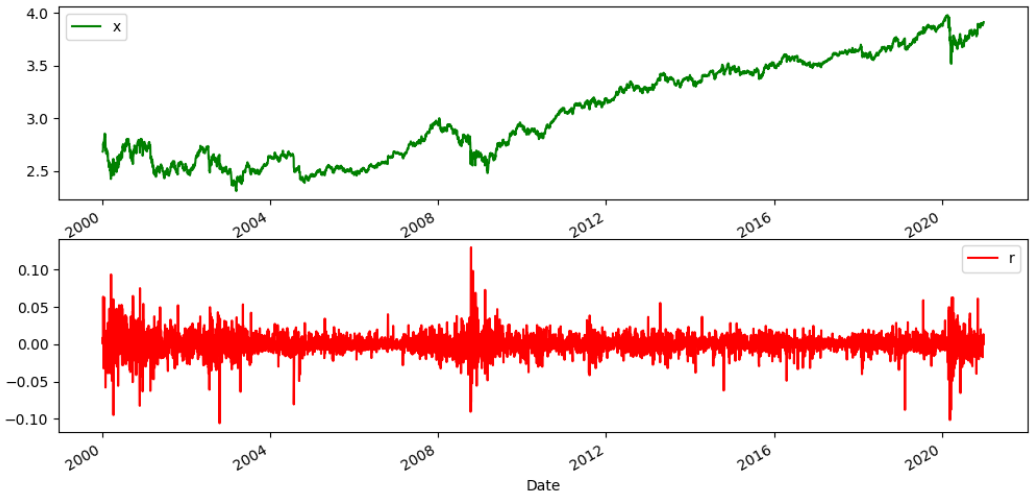
In conclusion, the validation process involving DBSCAN confirms the accuracy of outliers detected by the VAE algorithm. This consistency boosts confidence in subsequent analysis steps. The VAE algorithm accurately identifies anomalies, confirmed by DBSCAN.

Experiment

i. Data Description

We selected Coca-Cola stock as our dataset, encompassing daily closing prices from January 4, 2000, to December 31, 2020. The choice of Coca-Cola was based on its relatively stable trend and lower susceptibility to external influences compared to other stocks. While not exhibiting the lowest volatility, Coca-Cola demonstrated shock tendencies suitable for our purposes.

For experimentation, one dataset utilized the original data, while the other incorporated smoothed anomalies. Percentiles of 80, 95, and 99 were tested to observe changes. Smoothing involved the following methodology.



<Figure.5 time plot of x_t and r_t>

ii. Smoothing Anomalies through VAE

In this section, we delve into the practical application of utilizing anomaly points, identified through the Variational Autoencoder (VAE), to enhance the stability and reliability of stock price predictions.

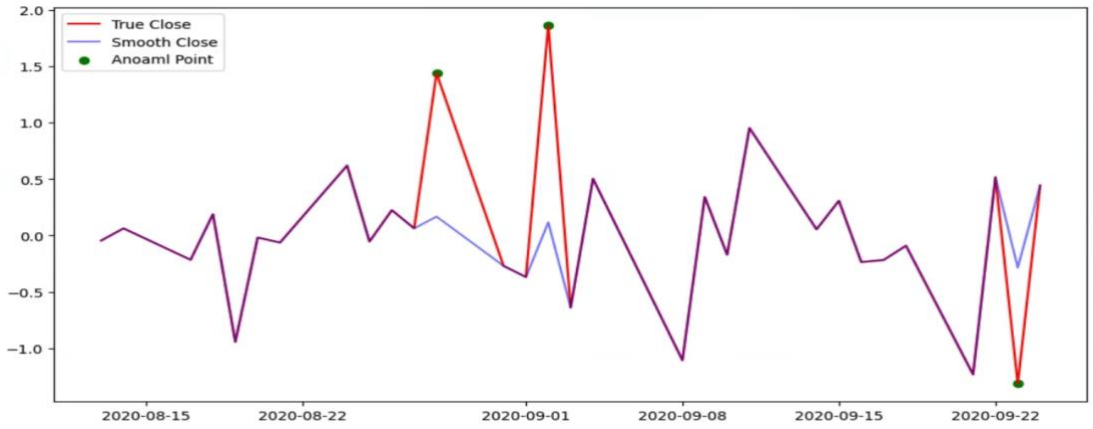
Anomaly Points Identification

As outlined earlier, anomaly points represent instances where a notable difference exists between the actual closing price and the reconstructed closing price obtained through VAE. These points serve as indicators of significant deviations in the stock price data.

Smoothing Methodology

The methodology for smoothing involves leveraging the information from identified anomaly points and their associated probability distribution. Specifically, when the actual closing price

exceeds the reconstructed closing price and aligns with an anomaly point, a smoothing technique is implemented. This involves substituting the original closing price with the reconstructed mean plus three times the reconstructed sigma. The visual representation of this approach is elucidated in the accompanying diagram.



<Figure.6 Smoothing from VAE>

iii. Quantitative Testing

In this section, we validate our model quantitatively using the following metrics:

- MAE (Mean Absolute Error): Measures the average absolute differences between predicted and actual values.
- MSE (Mean Squared Error): Quantifies the average squared differences between predicted and actual values.
- RMSE (Root Mean Squared Error): Represents the square root of the MSE, providing a comparable scale to the dependent variable.
- MPE (Mean Percentage Error): Measures the average percentage difference between predicted and actual values.

Model	MAE	MSE	RMSE	MPE
Naïve Model	0.317	0.144	0.379	80.044
Ref_80	0.805	2.615	1.617	58.46
Ref_95	0.346	0.238	0.488	74.432
Ref_99	0.287	0.131	0.362	74.482

<Table.3 Evaluation metrics of model>

Through this comparison table, it is evident that our approach, represented by Ref_99, demonstrates superior predictive performance.

iv. Qualitative Testing

Outlier Qualitative Testing

Analyzing events on specific dates, the table below provides context for considering them as outliers.

Date	Event
2000-03-15	Racial Discrimination Issue
2000-11-20	Quaker Oats Acquisition Under-the-table Negotiation
2000-11-22	Nasdaq Below 3000 & Quaker Oats $1.5 Billion Acquisition Pursuit
2002-10-16	Coca-Cola Earnings Announcement
2004-07-23	Market decline attributed to widespread poor performance
2008-10-09	Global Financial Crisis Triggered by Subprime Mortgage
2008-10-14	Declining Amidst Recession Concerns
2009-02-12	Positive Quarterly Earnings
2019-02-14	Q4 Sales Disappointment
2020-03-12	Market Collapse Due to COVID-19

<Table.4 Stock Price Volatility Events for Coca-Cola>

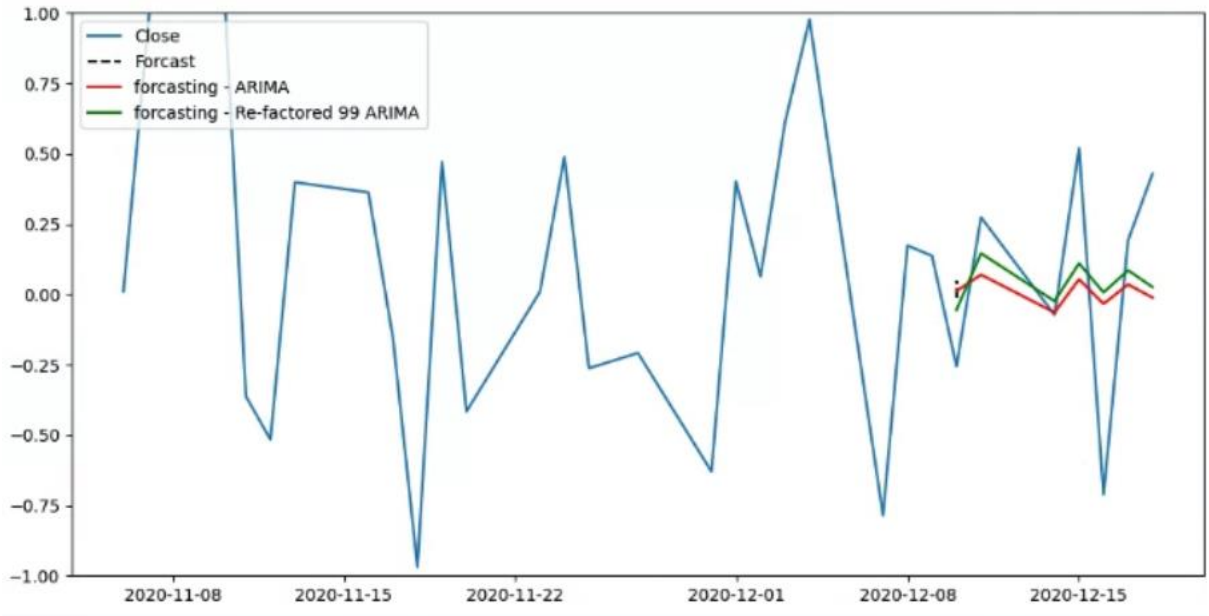
The table outlines the rationale behind designating these events as outliers.

Smoothing Qualitative Testing

Explaining the necessity of smoothing involves addressing the non-linearity of time series data and the complexity of handling noisy data with traditional methods like ARIMA. Smoothing supports the significance of our results.

The studies by Majumder and Hossain (2019) and Petrică, Stancu, and Tindeche (2016) highlight the limitations of ARIMA in handling extremely collapsed markets and in financial and monetary economics, respectively. ARIMA assumes linearity, while many time-series data exhibit non-linear patterns. Moreover, ARIMA faces challenges in dealing with complex or noisy datasets, making it difficult to find optimal parameters (pdq). Therefore, smoothing becomes essential, supporting the meaningfulness of our results.

These qualitative assessments contribute to a comprehensive understanding of the model's performance and the validity of applied techniques. Additionally, in the figure below, the mention of our enhanced model reflects the progress made, further validating our approach.



<Figure.7 Re-factored ARIMA>

Conclusion

This report outlines the development of a stock value prediction model for Coca-Cola spanning from January 4, 2000, to December 31, 2020. In response to the need for complementing ARIMA, various models were compared, highlighting the effectiveness of our approach in achieving enhanced predictions.

Key Findings

Successful anomaly detection with VAE, validated using DBSCAN, has led to an incremental improvement in ARIMA through our methodology. As we move forward, our aspiration is to apply this approach to diverse datasets with additional resources. Additionally, exploring alternative methods to mitigate shocks remains an avenue for future research.

Benefits of Smoothing:

- Enhanced Stability: Addressing anomaly points contributes to a more stable representation of stock prices.
- Reduced Volatility: Smoothing mitigates abrupt fluctuations caused by anomalies, enhancing the reliability of predictions.
- Improved Predictive Accuracy: The refined dataset resulting from smoothing contributes to heightened accuracy in predicting stock prices.

Limitation

While efforts were made to better align the data with the ARIMA model, there is a slight element of forced manipulation in our model, addressing the inherent limitations of ARIMA. This may be viewed as a form of data manipulation.

In summary, the incorporation of VAE-detected anomaly points for stock price smoothing emerges as a pivotal strategy in refining predictive models. By effectively addressing anomalies, our approach enhances the stability and reliability of stock price predictions, offering a practical solution to the challenges posed by anomalies in stock price analysis. This marks the completion of our comprehensive methodology, providing valuable insights for navigating the complexities of financial markets.

Reference

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In kdd (Vol. 96, No. 34, pp. 226-231).

Jung, J., & Kim, J. (2020). A performance analysis by adjusting learning methods in stock price prediction model using LSTM. Journal of Digital Convergence, 18(11), 259-266.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

Majumder, M. M. R., & Hossain, M. I. (2019, February). Limitation of ARIMA in extremely collapsed market: A proposed method. In 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE) (pp. 1-5). IEEE.

Petrică, A. C., Stancu, S., & Tindeche, A. (2016). Limitation of ARIMA models in financial and monetary economics. Theoretical & Applied Economics, 23(4).

Shumway, R. H., & Stoffer, D. S. (2017). ARIMA models. Time series analysis and its applications: with R examples, 75-163.

Siami-Namini, S., & Namin, A. S. (2018). Forecasting economics and financial time series: ARIMA vs. LSTM. arXiv preprint arXiv:1803.06386.

Stevenson, S. (2007). A comparison of the forecasting ability of ARIMA models. Journal of Property Investment & Finance, 25(3), 223-240.

Tian, X., Xu, L., Liu, L., & Wang, S. (2010). Analysis and forecasting of port logistics using TEI@ I methodology. In Business Intelligence in Economic Forecasting: Technologies and Techniques (pp. 248-264). IGI Global.

Wang, S., Li, C., & Lim, A. (2019). Why are the ARIMA and SARIMA not sufficient. arXiv preprint arXiv:1904.07632.

APPENDIX. codes

```
# import modules


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import FinanceDataReader as fdr
from statsmodels.tsa.arima_model import ARIMA
from pmdarima.arima import auto_arima
from keras.models import Sequential
from keras.models import Model
from keras.layers import LSTM, Input, Dropout
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error


# Data Load
start_date = '2000-01-01'
end_date = '2020-12-31'


raw_df = fdr.DataReader('KO', start_date, end_date)
df = raw_df[['Close']].copy()#.resample('M').mean()
#df.Close = np.log(df.Close)


diff_df = df.diff().fillna(method='bfill')
diff_df


train = diff_df.copy()
train = train.reset_index()
```

```
scaler = MinMaxScaler().fit(train[['Close']])
train['Close']  = scaler.transform(train[['Close']])
seq_size = 7

def to_sequence(x, y, seq_szie=1):
    x_values = []
    y_values = []

    for i in range(len(x)-seq_size):
        x_values.append(x.iloc[i:(i+seq_size)].values)
        y_values.append(y.iloc[i+seq_size])

    return np.array(x_values), np.array(y_values)

X_train, Y_train = to_sequence(train[['Close']], train['Close'], seq_size)

# modeling

import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Lambda, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras import backend as K
from tensorflow.keras import optimizers
import math
from keras.layers import Input, LSTM, Lambda, TimeDistributed, Dense, Dropout
from keras.models import Model
from keras import backend as K
from keras import optimizers
from keras.callbacks import Callback
from keras.layers import LeakyReLU
from keras.layers import BatchNormalization
```

```
X_values = X_train.copy()


pi = K.constant(math.pi)


input_dim = X_values.shape[2]  # number of features

time_steps = X_values.shape[1]  # number of time steps

intermediate_dim = 64

latent_dim = 8


# define encoder

inputs = Input(shape=(time_steps, input_dim))

#h = Dense(intermediate_dim, activation='relu')(inputs)

h = LSTM(intermediate_dim, activation='relu', recurrent_activation='sigmoid',
recurrent_dropout=0.4, unroll=True)(inputs)

h = BatchNormalization()(h)

h = Dropout(0.4)(h)


z_mean = Dense(latent_dim)(h)

z_log_var = Dense(latent_dim)(h)


# Latent space function

def sampling(args):

    z_mean, z_log_var = args

    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], latent_dim), mean=0.,
stddev=1.0)

    return z_mean + K.exp(z_log_var) * epsilon


z = Lambda(sampling)([z_mean, z_log_var])


# define decoder

repeat_layer = RepeatVector(time_steps)

decoder_h = LSTM(intermediate_dim, activation='relu',
recurrent_activation='sigmoid', return_sequences=True, recurrent_dropout=0.4,
unroll=True)

decoder_mean = TimeDistributed(Dense(input_dim, 'sigmoid'))

decoder_var = TimeDistributed(Dense(input_dim, activation='softplus'))

#decoder_var = TimeDistributed(Dense(input_dim, activation=LeakyReLU(alpha=0.01)))
```

```
h_decoded = repeat_layer(z)

h_decoded = decoder_h(h_decoded)

h_decoded = BatchNormalization()(h_decoded)

x_decoded_mean = decoder_mean(h_decoded)

x_decoded_var = decoder_var(h_decoded)


prob_model = Model(inputs, [z_mean, z_log_var, x_decoded_mean, x_decoded_var,
h_decoded])


#reconstruction_loss = 0.5 * (K.square(inputs - x_decoded_mean) / (x_decoded_var +
K.epsilon()) + K.log10(x_decoded_var + K.epsilon()) + K.log10(2 * pi))

# reconstruction_loss


reconstruction_loss = 0.5 * (K.square(inputs - x_decoded_mean) / (x_decoded_var +
K.epsilon()) + K.log(x_decoded_var + K.epsilon()) + K.log(2 * pi))

reconstruction_loss = K.sum(reconstruction_loss, axis=[1, 2])

reconstruction_loss /= input_dim


# KL Loss

kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)

kl_loss = K.sum(kl_loss, axis=-1)

kl_loss *= -0.5


vae_loss = K.mean(reconstruction_loss + kl_loss)


# VAE complile

opt = optimizers.Adam(learning_rate=5e-5)

prob_model.add_loss(vae_loss)

prob_model.compile(optimizer=opt)


# print model summary result

prob_model.summary()


#reconstruction_loss = 0.5 * (tf.square(inputs - x_decoded_mean) / (x_decoded_var +
tf.keras.backend.epsilon())
```

```
#                          + tf.math.log(x_decoded_var + tf.keras.backend.epsilon())
/ log_10_base_e

#                         + tf.math.log(tf.constant(2 * pi,
dtype=x_decoded_var.dtype)) / log_10_base_e)


z_mean, z_log_var, x_decoded_mean, x_decoded_var, h_decoded=
prob_model.predict(X_values)


def diagonal_avg(df):
    days = df[['day', 'day+1', 'day+2', 'day+3', 'day+4', 'day+5', 'day+6']]


    head_x = 0

    head_y = 0

    diag_ave = []


    while head_y<7: # 7 is the number of columns in the dataframe

        sum = 0

        x,y = head_x, head_y

        while y>=0:

            sum+=days.iloc[x,y]

            x+=1

            y-=1

        sum = sum/(x)

        diag_ave.append(sum)

        head_y+=1


    head_x = 1

    head_y = 6

    while head_x<len(df): # 5276 is the number of rows in the dataframe

        sum = 0

        x,y = head_x, head_y

        while y >= 0 and x<len(df):

            sum += days.iloc[x, y]

            x += 1

            y -= 1

        sum = sum / (head_y-y)
```

```
        diag_ave.append(sum)
        head_x += 1

    # build dataframe for diagonal average
    # index = df.Date
    # column named value = diag_ave
    df_diag_ave = pd.DataFrame(diag_ave, columns=['day'])
    return df_diag_ave

# Data Preprocessing

pred_df = train[['Date']].copy()
pred_df['Close'] = 0
pred_df['counter'] = 0
temp = []
for i in range(len(x_decoded_mean)):
    temp.append(x_decoded_mean[:][i].reshape(-1))
temp = pd.DataFrame(temp)

temp_var = []
for i in range(len(x_decoded_var)):
    temp_var.append(x_decoded_var[:][i].reshape(-1))
temp_var = pd.DataFrame(temp_var)

temp_var['Date'] = pred_df.Date
temp_var = temp_var.set_index('Date')
temp_var.columns = ['day', 'day+1', 'day+2', 'day+3', 'day+4', 'day+5', 'day+6']

temp['Date'] = pred_df.Date
temp = temp.set_index('Date')
temp.columns = ['day', 'day+1', 'day+2', 'day+3', 'day+4', 'day+5', 'day+6']



df_diag_ave = diagonal_avg(temp.reset_index())
```

```
df_diag_ave.index = df[1:].index
pred_df = df_diag_ave.copy()

df_diag_var = diagonal_avg(temp_var.reset_index())
df_diag_var.index = df[1:].index
pred_var_df = df_diag_var.copy()

Y_train = train[1:].set_index('Date').copy()

Y_train = Y_train[:-7]
pred_df = pred_df[:-7]
pred_var_df = pred_var_df[:-7]

#reconstruction_error = np.abs(Y_train['Close'] - pred_df['day'])
temp_df = pd.concat([Y_train['Close'], pred_df['day']], axis=1)
temp_df['cal'] = 0

for i in temp_df.index:
    if temp_df.loc[i, 'Close'] >= temp_df.loc[i, 'day']:
        temp_df.loc[i, 'cal'] = temp_df.loc[i, 'Close'] - temp_df.loc[i, 'day']
    else:
        temp_df.loc[i, 'cal'] = temp_df.loc[i, 'day'] - temp_df.loc[i, 'Close']

reconstruction_error = temp_df['cal']



threshold = np.percentile(reconstruction_error, 99)
outliers = np.where(reconstruction_error > threshold)

print("Anomaly Indices:", outliers)

outlier_idx = np.zeros(pred_df.shape, dtype=int)
indices = outliers[0]
outlier_idx[indices] = 1
sig_scaler = MinMaxScaler()
```

```
pred_var_df['day'] = sig_scaler.fit_transform(pred_var_df[['day']])


ano_df = pd.DataFrame(pred_df.copy())
ano_df['anomal'] = outlier_idx
ano_df['sigma'] = pred_var_df['day'].values
ano_df['sigma'] = np.sqrt(ano_df['sigma'].values)


ano_sig_min = ano_df['sigma'].quantile(0.05)
ano_sig_max = ano_df['sigma'].quantile(0.95)


ano_df.loc[ano_df['sigma']>ano_sig_max, 'sigma'] = ano_sig_max
ano_df.loc[ano_df['sigma']<ano_sig_max, 'sigma'] = ano_sig_min


ano_sig_min, ano_sig_max
ano_df['true_close'] = Y_train
ano_df['smooth_close'] = ano_df['true_close'].copy()


for idx in ano_df.index:
   if ano_df.loc[idx, 'anomal'] ==1:
       if ano_df.loc[idx, 'day'] < ano_df.loc[idx, 'true_close']:
          ano_df.loc[idx, 'smooth_close'] =  ano_df.loc[idx, 'day'] +
3*ano_df.loc[idx, 'sigma']
       else:
          ano_df.loc[idx, 'smooth_close'] =  ano_df.loc[idx, 'day'] -
3*ano_df.loc[idx, 'sigma']


final_df = pd.DataFrame([], columns=['true_close', 'smooth_close'])
final_df['true_close'] = scaler.inverse_transform(ano_df[['true_close']]).reshape(-
1)
final_df['smooth_close'] =
scaler.inverse_transform(ano_df[['smooth_close']]).reshape(-1)
final_df['anomal'] = ano_df['anomal'].values
final_df.index = Y_train.index




# Final Test ARIMA vs Refactored ARIMA
```

```
from sklearn.metrics import mean_squared_error

X = final_df.iloc[:-7, 0]
y = final_df.iloc[-7:, 0]

naive_model= auto_arima(X,
                    max_p = 10, max_q = 10, max_d = 3,
                    alpha = 0.05, n_jobs = 1,
                    seasonal = False)

pred = naive_model.predict(n_periods=7, return_conf_int=True)
y_pred=pred[0]
print(mean_squared_error(y_pred , y))
y_pred , y

y_pred = pred[0]
pred_ub = pred[1][:,0]
pred_lb = pred[1][:,1]
pred_idx = y.index

plot_final_df = final_df.iloc[-30: ,0].copy()

plt.figure(figsize=(12,6))

plt.plot(plot_final_df.index, plot_final_df.values, label= 'Close')
plt.vlines(pred_idx[0], -0.9, 0.9, linestyles='dashed',color='black',
label='Forcast')
plt.plot(pred_idx, y_pred.values, label='forcasting_ARIMA', color = 'r')
plt.fill_between(pred_idx, pred_lb, pred_ub, color='k', alpha=0.1, label='95%
Confidence Interval')

plt.ylim(-1.0, 1.0)
plt.legend(loc='upper left')
plt.show()
```

DBSACN

```
import FinanceDataReader as fdr
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import MinMaxScaler

# scaled data array - to use broadcasting easily since it is not the feature.
array_idx_df = np.array( [i/(len(df)-1) for i in range(len(df))] )
array_idx_diff_df = np.array( [i/(len(diff_df)-1) for i in range(len(diff_df))] )

mm_scaler = MinMaxScaler()

# data frame 만들기
df['time_idx'] = array_idx_df
df['scaled_close'] = mm_scaler.fit_transform(df[['Close']])
diff_df['time_idx'] = array_idx_diff_df
diff_df['scaled_close'] = mm_scaler.fit_transform(diff_df[['Close']])

# epsilon은 heuristically 하게...
eps_df = 0.15
eps_diff_df = 0.15

print(f"eps_df={eps_df}")
print("===== ===== ===== ===== =====")
print(f"eps_diff_df={eps_diff_df}")

# epsilon, set minimum number of samples
model_for_df = DBSCAN(eps=eps_df, min_samples=3)
model_for_diff_df = DBSCAN(eps=eps_diff_df, min_samples=3)

model_for_df.fit(df[['time_idx', 'scaled_close']])
df['cluster'] = model_for_df.fit_predict(df)

model_for_diff_df.fit(df[['time_idx', 'scaled_close']])
```

```
diff_df['cluster'] = model_for_diff_df.fit_predict(diff_df)


print(f"the number of outliers (by sclaed close value and time_index):
{ len(df[df['cluster'] == -1]) }")

df[df['cluster'] == -1
```