STATS 202: Final Project Report

Wajeeha Ahmad[a], **Pascual Eduardo Camacho**[a], **Fernando Rodriguez Silva Santisteban**[a]
[a]Stanford University, School of Engineering, Stanford, CA 94305

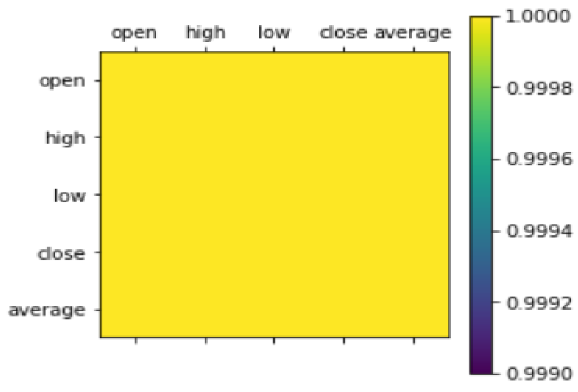
Kaggle Team Name: FWP

1 Introduction

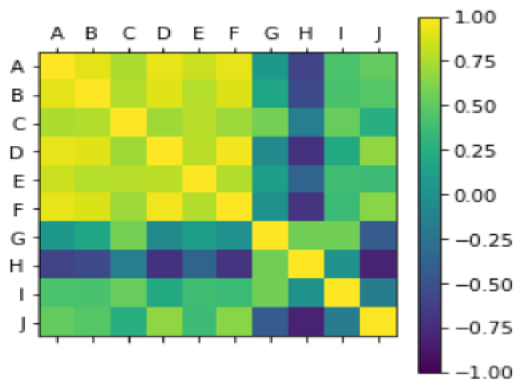
The project consisted of a prediction problem of the prices of 10 anonymized security instruments for up to 9 days in the future given a training dataset. The training data consisted of approximately 87 days of candlestick data at 5 second intervals, anonymized using random values for purposes of the project. Given that the data consisted of real-world observations, we checked for consistency of the data and potential issues with missing values. The data selection, preprocessing, and transformation process for each of the models is described in the corresponding section, as each model had different assumptions and we worked based on those assumptions.

The focus of the inference we did was on regression given that we had to provide point estimates for a continuous variable (open price). We approached the problem using different data mining and inference techniques: overall, we used four different approaches and compared them using cross-validation. Because the objective was to minimize the out-of-sample prediction error, we used the cross-validation set RMSE as our metric for model comparison. Since cross-validation RMSE is a proxy for the testing error, we consider it to be an adequate comparison metric.

Prior to starting with the data processing and model selection, we performed diagnostics in the data. We found that approximately 70.30% of the observations did not have change between the open and close price and that considering all symbols approximately 15.23% of the periods had no change in any security. We also found that the candlestick variables were strongly correlated with each other (Figure 1a) and that the open prices for the ten securities also had some degree of correlation (Figure 1b).



(a) Candlestick variables



(b) Securities open prices

Fig 1: Visualization of correlation matrices between data variables (*Note the different scales*).

In the following sections, we detail the modeling approaches we used, outline their underlying data-generating process, data preparation and data mining steps, and provide an interpretation of the results obtained. In Section 5, we compare the various models we explored and describe our preferred approach for the prediction problem, as well as potential extensions for similar projects in the future.

2 First Approach: Static and Linear Models

Our first approach involves testing a series of static models to develop baseline measures for predicting open prices several days into the future.

2.1 Data selection, preprocessing and transformation

We fit a series of simple models on the open prices of each stock individually to identify overall trends in the data for each stock. To easily interpret trends and changes in stock open prices over time for each stock, we combine the day and time variable into a single date timestamp as follows: we convert the first day into the first date of the year 2021 and all other days into the subsequent dates of the year. We then concatenate the date and time into a single $datetime$ variable and set that as the index of our data frame for ease of visualization.

To compare these models against one another, we divide the 87 days of data we are given into a training set (90%) and a test set (10%). We first fit our models on the training set and then use the model to make predictions for the open prices on the test set. We then calculate the mean squared errors of the differences between our predicted values and the actual open prices in the test set.

2.2 Hypothesis, data generating process and data mining

Given time series data where we need to predict open prices over the next nine days, we start with a simple approach that assumes that the expected values of future open prices will be equal to the average of all previously observed open prices. This simple average approach can be represented as follows, where y represents the open price and t is the time (date and timestamp):

$$\hat{y}_{t+1} = \frac{1}{t}\sum_{i=1}^{t} y_i \tag{1}$$

In several cases, stock open prices may have changed sharply several periods ago. In such cases, the simple average method, which takes the mean of all previous data, could vastly over or underestimate the predicted values. Thus, to improve upon the simple average method, we calculate the moving average, which takes the average of the past few time periods alone. Since we are interested in longer-term trends, we calculate the moving average using data from the previous 2100 observations (approximately 0.41 days), which can be represented as follows (where $p = 2100$):

$$\hat{y}_t = \frac{1}{p}(y_{i-1} + y_{i-2} + ... + y_{i-p}) \tag{2}$$

The above two methods, the simple average and the moving average, lie on opposite ends of the spectrum. An approach that lies between these two approaches would weigh the data points differently while taking the entire data set into account. This could involve giving greater weights to recent observations than to observations from the distant past. Simple exponential smoothing makes predictions using weighted averages such that the the smallest weights are associated with older observations and the weights decrease exponentially as observations come from further in the past as follows. In the equation below, $0 \leq \alpha \leq 1$ is the smoothing parameter, which controls the rate at which the weights decrease.

$$\hat{y}_{t+1|t} = \alpha y_t + \alpha(1-\alpha)y_{i-2} + \alpha(1-\alpha)^2 y_{i-2} + ... \tag{3}$$

The three methods above give static predictions that don't consider whether the data has an underlying trend, e.g. if the stock open prices are increasing or decreasing over time in general. To incorporate any underlying trends in the data, we make use of Holt's linear trend method to predict future open prices. This method extends simple exponential smoothing to enable predictions that incorporate a linear trend. It applies exponential smoothing to both the average value in the series and the trend as follows:

$$\hat{y}_{t+h|t} = l_t + h * b_t \tag{4}$$

In the above equation, the level l_t and the trend b_t are given by equations 5 and 6. The level equation is the weighted average of the observation and the one-step prediction from within the sample. The trend equation is the weighted average of the previous trend estimate (b_{t-1}) and the estimated trend at time t ($l_t - l_{t-1}$).

$$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) \tag{5}$$

$$b_t = \beta(l_t - l_{t-1}) + (1-\beta) * b_{t-1} \tag{6}$$

Since the holt linear trend method assumes a constant trend in the future that increases or decreases indefinitely, it can be problematic over long forecast horizons. We thus also test the Holt's dampened linear trend method, which flattens the trend by adding a dampening parameter (i.e. replacing parameter b with ωb) so that the trend converges to a constant value in the future.

We now apply each of the above models to each stock separately, applying each model and checking whether it improves our results or not.

2.3 Results and interpretation

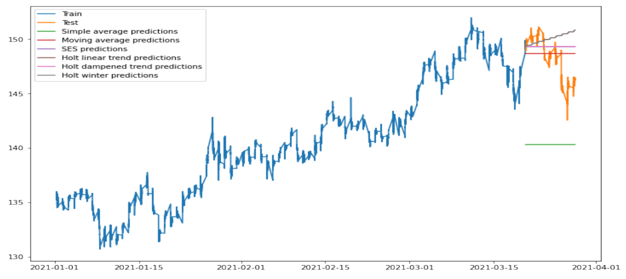
Table 1 shows the average RMSEs across all the stocks, which can be used as a measure of how the various models compare against one another. While some models perform better than others for different stocks, the overall RMSE is lowest for Holt's dampened linear trend followed by simple exponential smoothing and the moving average methods across all stocks (A to J).

Table 1: Models and their corresponding errors on the test data (10% of the data set)

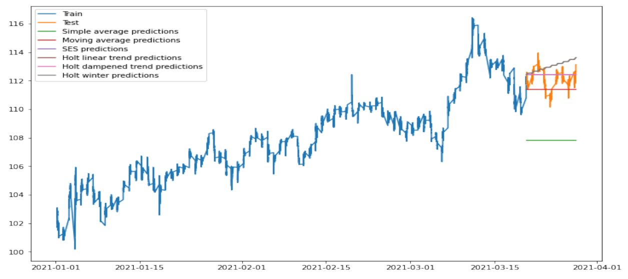
Model	**Root Mean Squared Error (RMSE)**
Simple average	12.221
Moving average	3.966
Simple exponential smoothing	3.642
Holt's linear trend	35.990
Holt's dampened linear trend	3.624

Figures 2a to 2j show the plots of the predicted values from each of the above models against the training data and test data. As expected, the simple average tends to over or under estimate the predicted values substantially in a number of cases, resulting in relatively high values for the RMSEs. While the Holt linear trend method performs relatively well in most cases, it increases indefinitely in certain cases (e.g. Stock I) such that the resulting RMSE is quite large. When the linear trend is dampened, the RMSEs improve substantially, resulting in the best overall fit amongst all models tested so far.

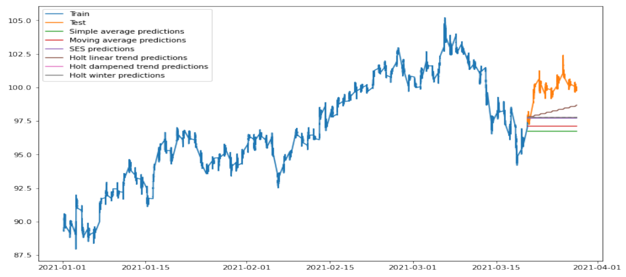
We decided to explore other non-parametric and semi-parametric models that may enable us to make more precise predictions for open prices in the future, which are described in the following sections.



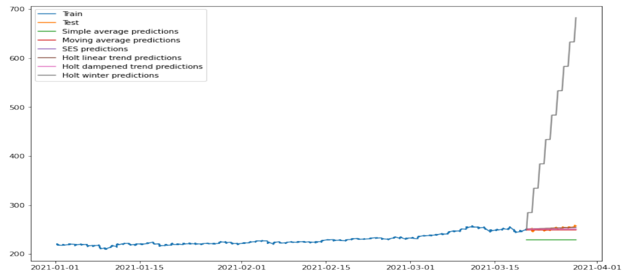
(a) Stock A



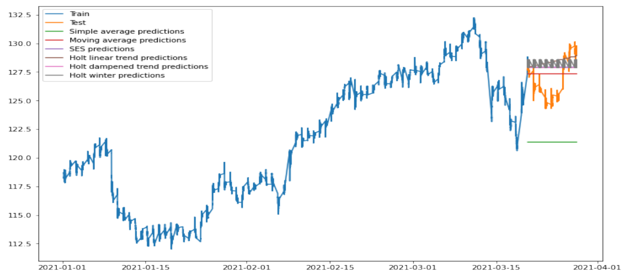
(b) Stock B



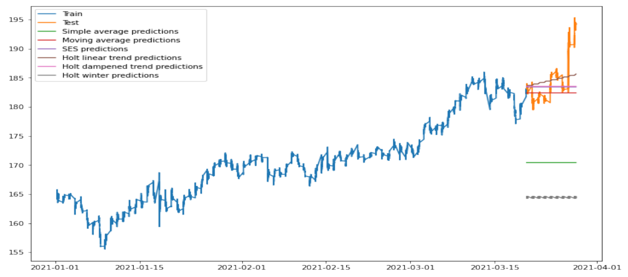
(c) Stock C



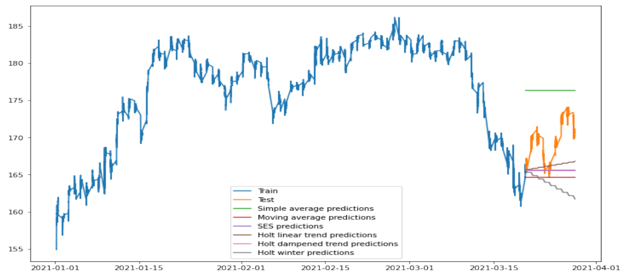
(d) Stock D



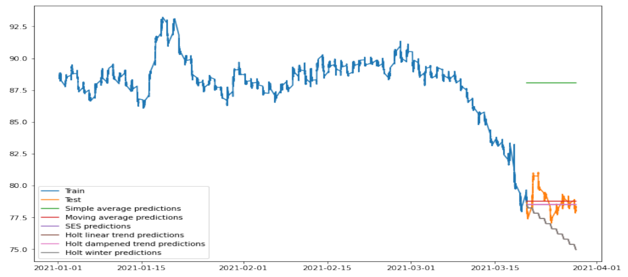
(e) Stock E



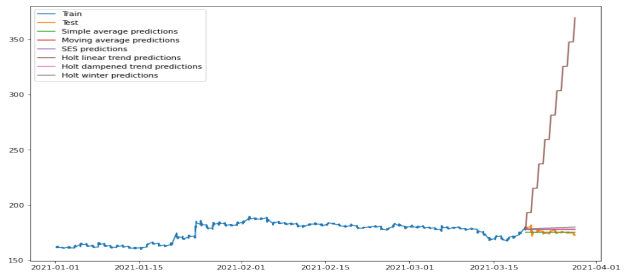
(f) Stock F



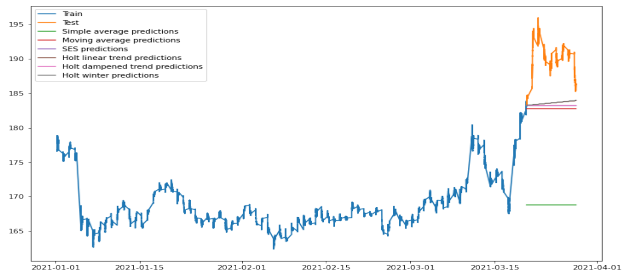
(g) Stock G



(h) Stock H



(i) Stock I



(j) Stock J

Fig 2: Static and linear models plotted against training and test data for Stocks A to J

3 Second Approach: Gradient Boosting

3.1 Hypothesis and data generating process

Due to the candlestick nature of the training data set, choosing an additive tree method seemed adequate, since a regression model could be built combining weak learners — also known as classifiers — to obtain a strong learner. This is the main idea behind boosting. Two specific methods were considered, namely AdaBoost and Gradient Boosting, where the latter was chosen due to personal familiarity with the method.

A weak learner is considered one that is slightly better or worse than randomly guessing a continuous value. By applying a boosting algorithm in a sequential manner, the hope is to minimize the error from individual weak classifiers. This is shown in Equation 7. Equation 8 is the loss function of the model, where the default least squares was implemented.

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m) \tag{7}$$

$$min_{\{\beta_m,\gamma_m\}_1^m} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{N} \beta_m b(x_i; \gamma_m)\right) \tag{8}$$

We did not perform shrinking for the entire data set (or computing the model in a slow learning fashion) due to the associated computational cost. To optimize, we conducted shrinking of the learning rate with values ranging from 0.01 to 0.5 for one stock (roughly 1/10 of the total data set) and selected the learning rate that yielded the lowest error; in this case, corresponding to a learning rate of 0.05. The additional parameters selected for the model are as follows: number of estimators (trees) was 2,000, the maximum depth (tree depth) was set at 3, and the validation fraction was equal to 10%. The remaining parameters of the model were kept to default. For replication purposes, the random state was chosen as 1.

3.2 Data selection, preprocessing and transformation

Our initial approach was to use the candlestick data as predictors and we derived additional parameters, namely, percent change, the difference between the previous close price and the new open price, and time in seconds (in order to use time as a quantitative predictor). The model was trained with this information and performed exceptionally well. To adapt to the nature of the problem at hand (predicting values based on past data only) we adapted the boosting model to only use time information – seconds in a day and day number – as predictors. As expected, the performance of the model decreased, but to our surprise, the magnitude of the decrease in accuracy was within an order of magnitude.

We arbitrarily chose January 1st, 2021 as the starting point of the data, which extended through March 28th, 2021. A unique identifier was added to each data point, comprised of stock symbol, day, and time of day. As discussed earlier in the first approach, the chosen split for our model was 90/10, 90% of the data set to train our model, leaving the remaining 10% for validation.

3.3 Data mining

Diving deeper into boosting, we look at the algorithm in Figure 3, borrowing from Trevor Hastie et.al. In the algorithm, the first step is to simply compute the value for gamma that minimizes the sum of the loss L, previously mentioned. In step 2(a) the pseudo residual r_{im} for the following tree is computed by calculating the derivative of the loss function with respect to the current tree, also known as the gradient. This is done for every tree m for a total number of trees M. A regression tree is fitted to the target r_{im} which gives terminal regions (tree leafs) in step 2(b). In step 2 (c) the output γ_{jm} is chosen as the one that minimizes the summation of the loss function, considering each newly predicted value. Finally the prediction is updated by adding the previous prediction to the new trees, where the learning rate comes into effect to reduce the contribution of the update, thus, enhancing the accuracy in the long run. A diagram of gradient boosting is shown in Figure 4 for reference.

```
1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
    (a) For $i = 1, 2, \ldots, N$ compute
            $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
    (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions
        $R_{jm},\ j = 1, 2, \ldots, J_m$.
    (c) For $j = 1, 2, \ldots, J_m$ compute
            $$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
    (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.
```

Fig 3: Gradient Tree Boosting Algorithm[1]

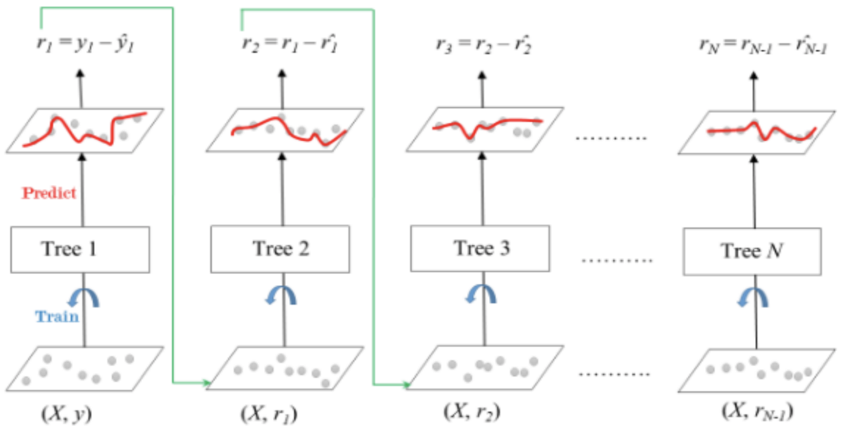


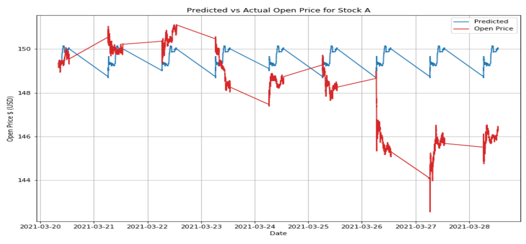
Fig 4: Diagram representing the gradient boosting method[2]

3.4 *Interpretation*

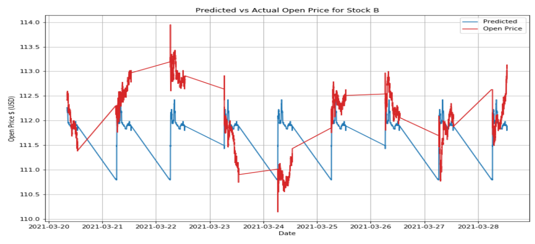
Figures 5a through 5j show the predicted values from the boosting model (in blue) against the actual open prices from our validation set (in red). The RMSE obtained from the boosting model was 2.425, slightly better than the best models presented in Section 2 of this report, the lowest of which was Holt's Damped Linear Trend, with RMSE of 3.624. For this reason, the boosting model was selected by the team to forecast nine days into the future for the Kaggle competition. The model was adapted as stated earlier for the competition's submission. In Section 5 we present a side-by-side comparison of the RMSE for all the models mentioned in this paper and further provide commentary on the results.

It is worth pointing out a few takeaways from the boosting model, since it was the model utilized for our forecasting and competition submission. First, the performance of the model was to an extent surprising, but it is reasonable to expect that this model performed as well as it did due to the nature of the modeling technique. The main idea behind the concept, learning from the errors of previous trees, to continually improve the model, makes it a prime choice for the task at hand. Second, the learning rate that we implemented could definitely be optimized further, but given the time constraints and associated computational cost, the variation of this parameter was minimal, and thus, it certainly factored into the model's performance. And lastly, perhaps the main takeaway from the boosting model, is that for the most part, it was able to accurately identify the short-period trends, but the same cannot be said about the long-period trends, as the plots show. This model would potentially be useful exclusively for short-term forecasting, without additional improvements.

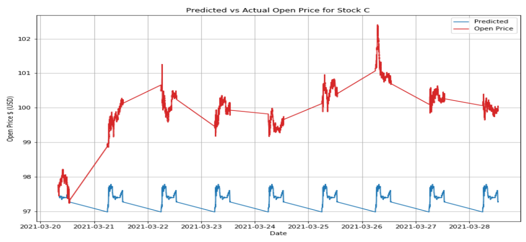
A short version of our model can be found in Appendix B, where Python code is shown for the training, testing, and predictions for Stock A.



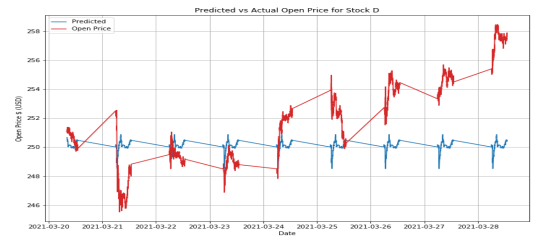
(a)



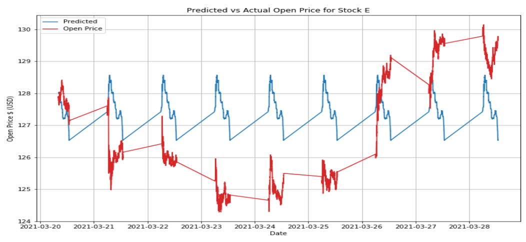
(b)



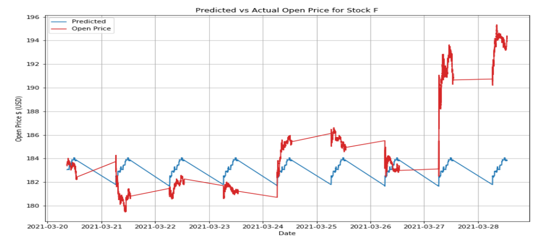
(c)



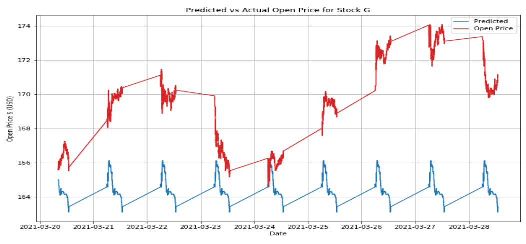
(d)



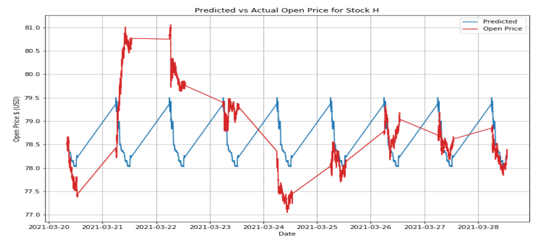
(e)



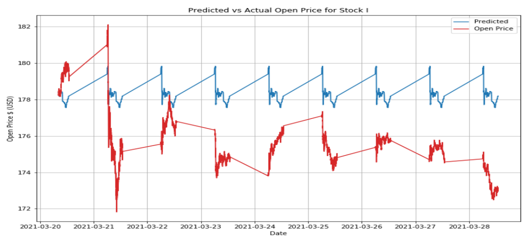
(f)



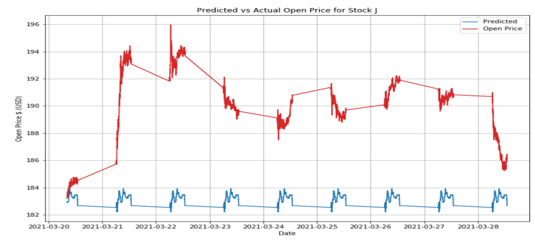
(g)



(h)



(i)



(j)

Fig 5: Gradient Boosting Regression prediction results against test data for Stocks A to J (Blue-predicted; Red-Open price).

4 Third Approach: Latent State Markov Process

4.1 Hypothesis and data generating process

This model represents a hypothesis of market behavior where prices follow a partially observable Markov process, meaning that the price in the next period will depend only on the current state of the market but some of the information is not directly observable. The change between periods (or 'step' in a random walk) will be randomly taken but the direction and magnitude of the step are influenced by a discrete latent state. The latent state can be interpreted as the market 'sentiment', although there does not need to be a direct interpretation as it represents the state of all information not directly observable.

The data generating process is described by equations 9 and 10. Each random step is Gaussian with mean and covariance conditional on the latent state. Moreover, the probability of each latent state is determined by a complex non-linear function of the previous time period's observed vector, hence the series of observations has the Markov property.

$$X_{t+1}|Z_{t+1} = X_t + \mathbb{N}(\mu_k, \sigma_k^2) \tag{9}$$

$$P\{Z_{t+1}|X_t\} = f(X_t) \tag{10}$$

The diagram in Figure 6 represents the model, where X_t is the augmented vector of market information at time t, Z_t is the latent state at time t, Θ represents the parameters of the complex function f, and Φ represents the mean and covariance parameters conditional on $Z_i = k$.

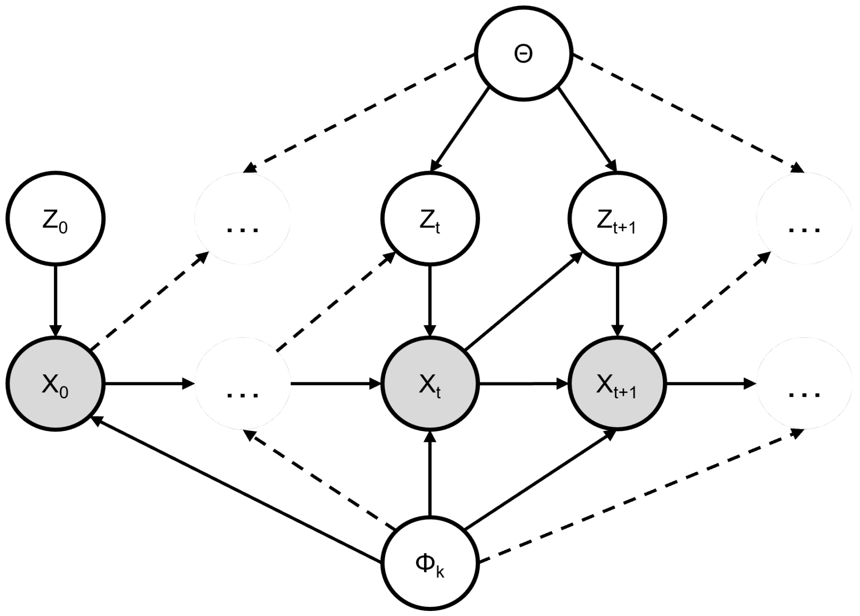


Fig 6: Diagram representing latent state Markov model data generating process.

Two important assumptions of the model are that the prices follow a Markov property and that the steps are Gaussian. The first assumption is based on the fact that individuals interacting with the market and affecting the prices are observing the prices as they evolve to make decisions. To better capture a complete picture of how the market is behaving, the state space for each time period can be grown to include lagged prices and external features (see next subsection for detail). A potential limitation is that the model does not capture external shocks such as public announcements, and will only capture their impact through the randomness inherent to each step. The second assumption is based on the fact that the price at each period is strongly correlated to the price of the previous with some degree of randomness in the change. Although there is no evidence that the randomness is necessarily Gaussian, it provides a good starting point and has properties that make statistical inference on the multivariate vector of observed prices easier. Overall the model is consistent with our understanding of the economic theory of stock price behavior.

4.2 Data selection, preprocessing, and transformation

The first step in data selection was choosing which variables to use. After some exploration of the variables, the minimum, maximum, average and closing prices were dropped because of the strong colinearity with the open price and the lack of observations of these features for the periods we wanted to predict. The day number was also dropped (after using it as an identifier for the preprocessing step) because our model already assumed a time series structure and the day number would be redundant.

After selecting the features to use (namely stock symbol, time, and open prices), we modified the data structure to a wide format where each observation consisted of a vector in $\mathbb{R}^{10}$ of observed prices for a specific day and time. Periods with missing observations were dropped due to the large data set and the difficulty of doing inference in the chosen model with missing values. Additional variables were created from the data to make the Markov property assumption more robust. A moving average of the last five periods was added (short-term memory), the percentage change since the last period (could be more relevant than price), the percentage change in the last hour (mid-term memory), and the percentage change in the last 10 hours (long-term memory). The choice of variables corresponded to what a rational individual would presumably observe in the price structure when trying to find trends. Finally, the time was converted to seconds for ease of processing in the statistical inference. Note that all variables were normalized when input into the neural network, but were de-normalized when reconstructing the vector of prices.

The data selection, preprocessing, and transformation was done in R due to familiarity with the tidyverse package. The code can be found in Appendix C.

4.3 Data mining

We can use the data to gain knowledge on the parameters Θ and Φ_k for each latent state. The model is based on a Bayesian paradigm, so the function f produces a probability distribution over the different latent states and the likelihood of the observed data is based on the estimated probability of each latent state. We used an Expectation Maximization algorithm to iteratively adjust our belief about Θ and Φ_k so that the expected likelihood of the observed data is maximized.

The complex function that transforms an observed vector to the probability of each latent state will be estimated using a Neural Network with 2 hidden layers (128 and 64 nodes respectively, with hyperbolic tangent activation and L2 regularization) and an output layer with softmax activation. The kernels and biases of the Neural Network (Θ) and the conditional means and covariances for the random step (Φ_k) are optimized using EM.[3]

The E-step consists of estimating the expected likelihood of the data conditional on the parameters and then obtaining the gradient with respect to each parameter. In order to estimate the likelihood of the data, we used assumptions similar to a variational autoencoder model. Because the latent state is dependent on the observed augmented vector of the previous time period, we can directly estimate the probability distribution without forward and backward message passing. The expected likelihood then becomes:

$$\begin{aligned} P\{X_{1:T}|\Theta, \Phi\} &= \prod_{t=1}^{T}\left(\sum_{k=1}^{K}(P\{X_t \mid Z_t = k\}P\{Z_t = k\}\right) \\ &= \prod_{t=1}^{T}\begin{bmatrix} P_{\mathbb{N}(\mu_0,\sigma_0^2)}\{X_t - X_{t-1}\} \\ P_{\mathbb{N}(\mu_1,\sigma_1^2)}\{X_t - X_{t-1}\} \\ \vdots \\ P_{\mathbb{N}(\mu_K,\sigma_K^2)}\{X_t - X_{t-1}\} \end{bmatrix}^{\top} f(X_{t-1}) \end{aligned} \tag{11}$$

The M-step consists of using the gradient of the expected likelihood with respect to each parameter and using a numerical optimization algorithm (Adam) to update the parameters in a direction that maximizes the expected likelihood.

$$\{\Theta^*, \Phi^*\} = \underset{\Theta,\Phi}{\text{argmax}} P\{X_{1:T}\}$$

We tried four possible numbers of latent states: 4, 8, 12 and 16. The algorithm was ran in 100 epochs with batches of 8192 observations for each alternative using 90% of the available data. Cross-validation based on the likelihood of the 10% held out data was used to determine the optimal hyperparameters.

After training the algorithm and obtaining the optimal hyperparameters, we used the posterior distributions for the conditional means and covariances and the posterior encoder to simulate data for the 10% of observations that were held out and obtain the RMSE. Each time period was simulated individually and the probability of latent states for the following obtained using the encoder on the simulated data. 100 simulations were performed due to limited computational resources, and the average of the simulations for each period was considered as the predicted price. Appendix D contains the code used for the implementation of the model.

4.4 Interpretation

The latent state model had a RMSE of 68.552 using the described simulation method to predict out-of-sample open prices. The model performed worse than the other simpler models, presumably because the simulation approach to prediction consists of adding consecutive independent Gaussian random variables to the last observed price. As the number of simulated periods grows, the variance of the predicted prices also grows linearly. Several runs of the simulation were done to reduce the variance in the result, but it quickly becomes too computationally expensive to simulate more runs through the several thousand periods that need to be predicted. An alternative approach would be to use the expected value of the sum of Gaussian random variables, however the mean and covariance of the steps depends on the latent state which in depends on the previous observed price. Given that the complex function f is unknown and approximated using a neural network that does not necessarily exhibit linear expected value properties, taking the expected value of the steps is no longer an option. Figure 7 shows a comparison of the observed and the predicted prices for all the securities. The model captured patterns similar to the observed in some cases (particularly stock D), but performed very poorly with others (Stocks E, H, and I).
In the following section we compare the models and explain our predictions for the test periods.

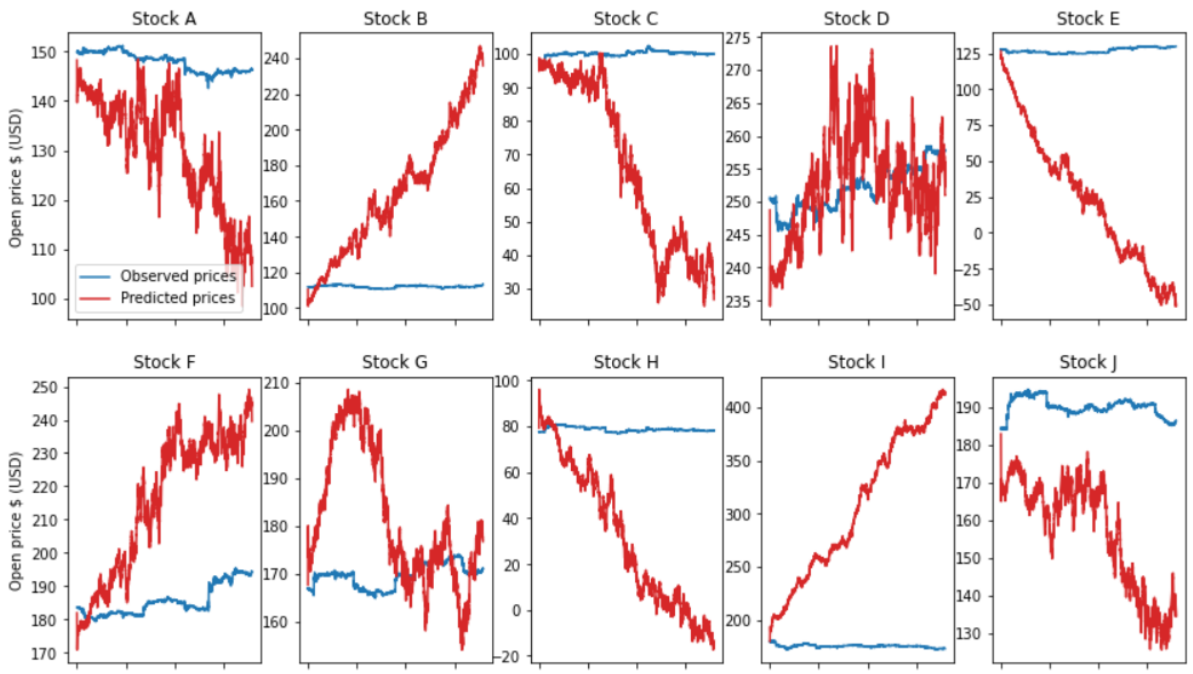


Fig 7: Comparison of predicted (red) and observed (blue) open prices for held out data in latent model ($N_{sim} = 100$).

5 Evaluation

Table 2 shows the Root Mean Squared Error (RMSE) for the seven models evaluated in this report. We use the RMSE as the chosen metric to compare models as an estimator of the out-of-sample accuracy in predicting open prices. Given that our task was a regression problem for predicting future values, it is a good measure of how succesful we might be in the task. We have highlited the model with the lowest RMSE, Gradient Boosting, which was then used for our submitted predictions (submitted as team FWP with submission ID 22527504).

Table 2: Models and their corresponding errors on the test data (10% of the data set)

Model	**Root Mean Squared Error (RMSE)**
Simple average	12.221
Moving average	3.966
Simple exponential smoothing	3.642
Holt's linear trend	35.990
Holt's dampened linear trend	3.624
Gradient boosting model	**2.425**
Latent state model	68.552

After choosing the boosting approach, we retrained the model using the full training data (after the cleaning and preprocessing steps) to leverage all the observations we had available. We then predicted the open prices based on the day numbers and time corresponding to the next 9 days, and formatted the data according to the submission instructions.

6 Individual Contributions

This report was developed as a team effort and each member's main contribution was the corresponding section, as outlined below.

6.1 Wajeeha

Implemented and wrote the Static and Linear approach (Section 2).

6.2 Pascual

Implemented and wrote the Gradient Boosting approach (Section 3).

6.3 Fernando

Implemented and wrote the Latent State approach (Section 4).

References

1 T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*, Springer, New York (2001).

2 Nikki2398, "ML - Gradient Boosting." GeeksforGeeks: user generated content for computer science. (September 2, 2020). https://www.geeksforgeeks.org/ml-gradient-boosting/ (Accessed: 27 August 2020).

3 A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society, Series B* **39**, 1–38 (1977). https://www.jstor.org/stable/2984875.

List of Figures

List of Tables

7 Appendices

Appendices start in the next page.

Appendix A: Static and linear models

In [185]:

```
import pandas as pd

import math
import matplotlib
import numpy as np
import seaborn as sns
import time

from datetime import date, datetime, time, timedelta
from matplotlib import pyplot as plt
from pylab import rcParams
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from tqdm import tqdm_notebook

from math import sqrt

%matplotlib inline
```

In [186]:

```
stocks = pd.read_csv('../train_data.csv')
stocks
```

Out[186]:

	symbol	open	high	low	close	average	time	day
0	B	101.72	101.72	101.72	101.72	101.72	06:00:00	0
1	B	101.72	101.72	101.72	101.72	101.72	06:00:05	0
2	B	101.72	101.72	101.72	101.72	101.72	06:00:10	0
3	B	101.72	101.72	101.72	101.72	101.72	06:00:15	0
4	B	101.72	101.72	101.72	101.72	101.72	06:00:20	0
...	...	...	...	...	...	...	...	...
4330249	H	78.26	78.29	78.25	78.29	78.28	12:59:35	86
4330250	H	78.28	78.29	78.28	78.29	78.28	12:59:40	86
4330251	H	78.29	78.30	78.28	78.30	78.29	12:59:45	86
4330252	H	78.29	78.30	78.26	78.26	78.29	12:59:50	86
4330253	H	78.28	78.29	78.25	78.26	78.27	12:59:55	86

4330254 rows × 8 columns

In [187]:

```python
# Initialize year
year = "2021"

def day_to_date(day_num):
    res = []
    for i in range(len(day_num)):
        res.append(datetime.strptime(year + "-" + day_num[i], "%Y-%j").strfti
    return res
```

Typesetting math: 0%

In [188]:

```
symbol = "J"

stocks_X_df = stocks.loc[(stocks['symbol'] == symbol)]
stocks_X_df = pd.DataFrame(stocks_X_df)

stocks_X_df['day_num'] = (stocks_X_df['day'] + 1).apply(str)
stocks_X_df['date'] = day_to_date(stocks_X_df["day_num"].values)
stocks_X_df['date_time'] = stocks_X_df.date.astype(str) + ' ' + stocks_X_df.t
stocks_X_df['date_time'] = pd.to_datetime(stocks_X_df['date_time'])
stocks_X_df = stocks_X_df.set_index('date_time')

stocks_X_df
```

Out[188]:

	symbol	open	high	low	close	average	time	day	day_num	date
date_time										
2021-01-01 06:00:00	J	176.84	176.84	176.84	176.84	176.84	06:00:00	0	1	01-01-2021
2021-01-01 06:00:05	J	176.84	176.84	176.84	176.84	176.84	06:00:05	0	1	01-01-2021
2021-01-01 06:00:10	J	176.84	176.84	176.84	176.84	176.84	06:00:10	0	1	01-01-2021
2021-01-01 06:00:15	J	176.84	176.84	176.84	176.84	176.84	06:00:15	0	1	01-01-2021
2021-01-01 06:00:20	J	176.84	176.84	176.84	176.84	176.84	06:00:20	0	1	01-01-2021
...	...	...	...	...	...	...	...	...	...	...
2021-03-28 12:59:35	J	186.37	186.40	186.37	186.39	186.40	12:59:35	86	87	03-28-2021
2021-03-28 12:59:40	J	186.37	186.39	186.37	186.39	186.37	12:59:40	86	87	03-28-2021
2021-03-28 12:59:45	J	186.40	186.43	186.40	186.43	186.42	12:59:45	86	87	03-28-2021
2021-03-28 12:59:50	J	186.42	186.44	186.38	186.44	186.41	12:59:50	86	87	03-28-2021
2021-03-28 12:59:55	J	186.45	186.45	186.26	186.26	186.31	12:59:55	86	87	03-28-2021

437322 rows × 10 columns

In [189]:
```
#Creating train and test sets

train_prop = 0.9

train = stocks_X_df[0: round(train_prop * len(stocks_X_df.index))]
test = stocks_X_df[round(train_prop * len(stocks_X_df.index)): ]
#train_B
```

In [190]:
```
# Simple average

y_hat_avg = test.copy()
y_hat_avg['avg_forecast'] = train['open'].mean()
y_hat_avg['avg_forecast']
```

Out[190]:
```
date_time
2021-03-20 08:15:40    168.793011
2021-03-20 08:15:45    168.793011
2021-03-20 08:15:50    168.793011
2021-03-20 08:15:55    168.793011
2021-03-20 08:16:00    168.793011
                          ...
2021-03-28 12:59:35    168.793011
2021-03-28 12:59:40    168.793011
2021-03-28 12:59:45    168.793011
2021-03-28 12:59:50    168.793011
2021-03-28 12:59:55    168.793011
Name: avg_forecast, Length: 43732, dtype: float64
```

In [191]:
```
# Simple average rms
rms_avg = sqrt(mean_squared_error(test.open, y_hat_avg.avg_forecast))
print(rms_avg)
```

```
21.311199871495177
```

Typesetting math: 0%

```
In [192]:  # Moving average

           # 5 days = 420
           # 10 days = 840
           # 15 days = 1260
           # 20 days = 1680
           # 25 days = 2100
           # 30 days = 2520

           y_hat_avg['moving_avg_forecast'] = train['open'].rolling(2100).mean().iloc[-1
           y_hat_avg['moving_avg_forecast']
           #y_hat_avg
```

```
Out[192]:  date_time
           2021-03-20 08:15:40    182.755148
           2021-03-20 08:15:45    182.755148
           2021-03-20 08:15:50    182.755148
           2021-03-20 08:15:55    182.755148
           2021-03-20 08:16:00    182.755148
                                     ...
           2021-03-28 12:59:35    182.755148
           2021-03-28 12:59:40    182.755148
           2021-03-28 12:59:45    182.755148
           2021-03-28 12:59:50    182.755148
           2021-03-28 12:59:55    182.755148
           Name: moving_avg_forecast, Length: 43732, dtype: float64
```

```
In [193]:  rms_moving_avg = sqrt(mean_squared_error(test.open, y_hat_avg.moving_avg_fore
           print(rms_moving_avg)
```

```
           7.650823031489364
```

Typesetting math: 0%

In [194]:

```
# Simple Exponential Smoothing

from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Hol

SES_fit = SimpleExpSmoothing(np.asarray(train['open'])).fit(smoothing_level=0
y_hat_avg['SES'] = SES_fit.forecast(len(test))

y_hat_avg['SES']
```

```
C:\Users\Wajeeha\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.p
y:731: RuntimeWarning: invalid value encountered in greater_equal
  loc = initial_p >= ub
```

Out[194]:

```
date_time
2021-03-20 08:15:40    183.21508
2021-03-20 08:15:45    183.21508
2021-03-20 08:15:50    183.21508
2021-03-20 08:15:55    183.21508
2021-03-20 08:16:00    183.21508
                         ...
2021-03-28 12:59:35    183.21508
2021-03-28 12:59:40    183.21508
2021-03-28 12:59:45    183.21508
2021-03-28 12:59:50    183.21508
2021-03-28 12:59:55    183.21508
Name: SES, Length: 43732, dtype: float64
```

In [195]:

```
rms_SES = sqrt(mean_squared_error(test.open, y_hat_avg.SES))
print(rms_SES)
```

```
7.220498249679009
```

Typesetting math: 0%

In [196]:
```
# Holt's Linear Trend Model
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Hol
import statsmodels.api as sm

model = Holt(np.asarray(train['open']))
#model._index = pd.to_datetime(train_A.date_time)

holt_fit1 = model.fit(smoothing_level = 0.3,smoothing_slope = 0.05)
holt_fit2 = model.fit(optimized=True)
holt_fit3 = model.fit(smoothing_level = 0.3, smoothing_slope = 0.2)
holt_fit4 = model.fit(smoothing_level = 0.8, smoothing_slope = 0.05)

y_hat_avg['Holt_linear'] = holt_fit2.forecast(len(test))
y_hat_avg['Holt_linear']
```

```
C:\Users\Wajeeha\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.p
y:731: RuntimeWarning: invalid value encountered in greater_equal
  loc = initial_p >= ub
```

Out[196]:
```
date_time
2021-03-20 08:15:40    183.239591
2021-03-20 08:15:45    183.239607
2021-03-20 08:15:50    183.239623
2021-03-20 08:15:55    183.239639
2021-03-20 08:16:00    183.239656
                          ...
2021-03-28 12:59:35    183.949690
2021-03-28 12:59:40    183.949707
2021-03-28 12:59:45    183.949723
2021-03-28 12:59:50    183.949739
2021-03-28 12:59:55    183.949755
Name: Holt_linear, Length: 43732, dtype: float64
```

In [197]:
```
# Holt RMS
rms_holt_linear = sqrt(mean_squared_error(test.open, y_hat_avg.Holt_linear))
print(rms_holt_linear)
```

```
6.876884453602919
```

Typesetting math: 0%

In [198]:

```
# Holt's Dampened Trend

from statsmodels.tsa.holtwinters import ExponentialSmoothing
import numpy as np

model = ExponentialSmoothing(np.asarray(train['open']), trend='mul', seasonal
model2 = ExponentialSmoothing(np.asarray(train['open']), trend='mul', seasona

fit1 = model.fit()
fit2 = model2.fit()

y_hat_avg['Holt_dampened'] = fit2.forecast(len(test))
y_hat_avg['Holt_dampened']
```

```
C:\Users\Wajeeha\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.p
y:731: RuntimeWarning: invalid value encountered in greater_equal
  loc = initial_p >= ub
```

Out[198]:

```
date_time
2021-03-20 08:15:40    183.239574
2021-03-20 08:15:45    183.239574
2021-03-20 08:15:50    183.239574
2021-03-20 08:15:55    183.239574
2021-03-20 08:16:00    183.239574
                          ...
2021-03-28 12:59:35    183.239574
2021-03-28 12:59:40    183.239574
2021-03-28 12:59:45    183.239574
2021-03-28 12:59:50    183.239574
2021-03-28 12:59:55    183.239574
Name: Holt_dampened, Length: 43732, dtype: float64
```

In [199]:

```
# Holt Dampened RMS
rms_holt_dampened = sqrt(mean_squared_error(test.open, y_hat_avg.Holt_dampene
print(rms_holt_dampened)
```

```
7.19768352541734
```

Typesetting math: 0%

In [200]:
```
# Holt Winter's Method
# seasonal periods: how to choose?

fit1 = ExponentialSmoothing(np.asarray(train['open']) ,seasonal_periods=2 ,tr
y_hat_avg['Holt_Winter'] = fit1.forecast(len(test))
y_hat_avg['Holt_Winter']
```

```
C:\Users\Wajeeha\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.p
y:725: RuntimeWarning: invalid value encountered in less_equal
  loc = initial_p <= lb
C:\Users\Wajeeha\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.p
y:731: RuntimeWarning: invalid value encountered in greater_equal
  loc = initial_p >= ub
```

Out[200]:
```
date_time
2021-03-20 08:15:40    183.239607
2021-03-20 08:15:45    183.239610
2021-03-20 08:15:50    183.239642
2021-03-20 08:15:55    183.239645
2021-03-20 08:16:00    183.239676
                          ...
2021-03-28 12:59:35    183.996724
2021-03-28 12:59:40    183.996755
2021-03-28 12:59:45    183.996758
2021-03-28 12:59:50    183.996790
2021-03-28 12:59:55    183.996793
Name: Holt_Winter, Length: 43732, dtype: float64
```

In [201]:
```
# Holt Winter's Method

rms_holt_winter = sqrt(mean_squared_error(test.open, y_hat_avg.Holt_Winter))
print(rms_holt_winter)
```

```
6.8559658287253
```

Typesetting math: 0%

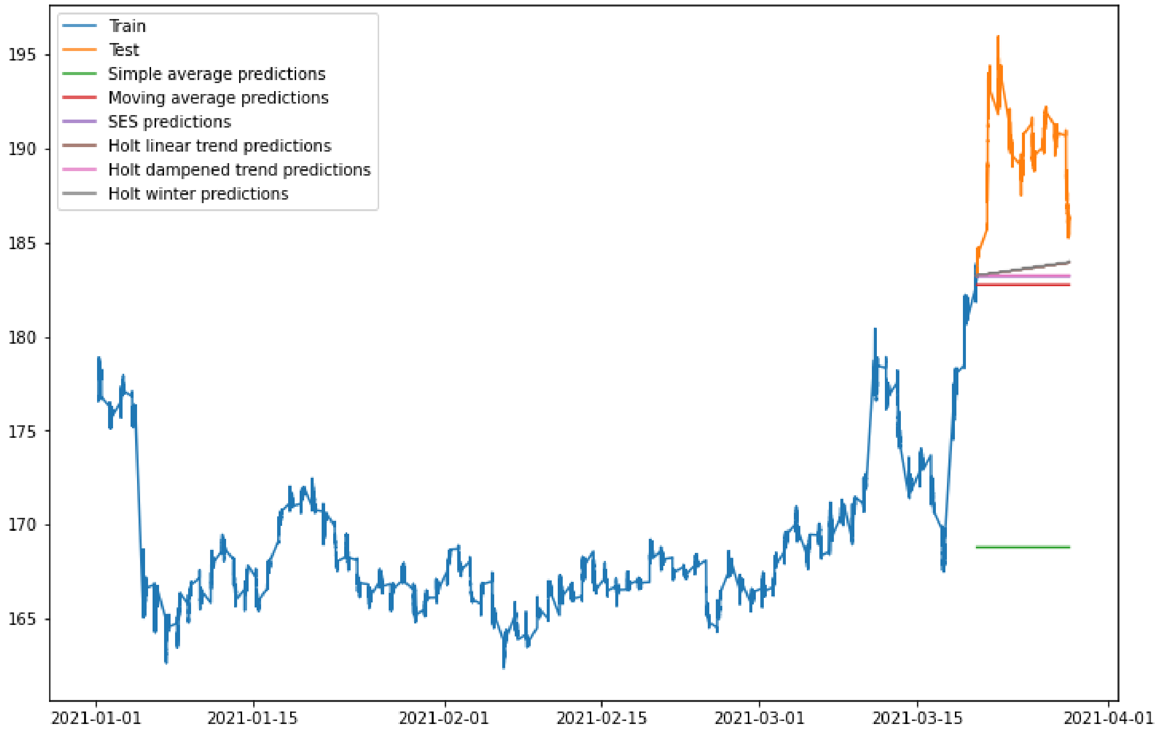
In [202]:

```
plt.figure(figsize=(12,8))

plt.plot(train['open'], label='Train')
plt.plot(test['open'], label='Test')

plt.plot(y_hat_avg['avg_forecast'], label='Simple average predictions')
plt.plot(y_hat_avg['moving_avg_forecast'], label='Moving average predictions'
plt.plot(y_hat_avg['SES'], label='SES predictions')
plt.plot(y_hat_avg['Holt_linear'], label='Holt linear trend predictions')
plt.plot(y_hat_avg['Holt_dampened'], label='Holt dampened trend predictions')
plt.plot(y_hat_avg['Holt_Winter'], label='Holt winter predictions')

plt.legend(loc='best')
plt.show()
```



Typesetting math: 0%

In [203]:
```
# Comparing various methods

print('rms_moving_avg: ', rms_moving_avg)
print('rms_holt_dampened: ', rms_holt_dampened)
print('rms_SES: ', rms_SES)
print('rms_avg: ', rms_avg)
print('rms_holt_linear: ', rms_holt_linear)
print('rms_holt_winter: ', rms_holt_winter)
```

```
rms_moving_avg:  7.650823031489364
rms_holt_dampened:  7.19768352541734
rms_SES:  7.220498249679009
rms_avg:  21.311199871495177
rms_holt_linear:  6.876884453602919
rms_holt_winter:  6.8559658287253
```

In [204]:
```
# Checking the data for stationarity

from statsmodels.tsa.stattools import kpss

print(" > Is the data stationary ?")
dftest = kpss(np.log(stocks_X_df.open), 'ct')

print("Test statistic = {:.3f}".format(dftest[0]))
print("P-value = {:.3f}".format(dftest[1]))
print("Critical values :")
for k, v in dftest[3].items():
    print("\t{}: {}".format(k, v))

# The test statistic is above the critical values, so we reject the null hypo
```

```
 > Is the data stationary ?
Test statistic = 70.010
P-value = 0.010
Critical values :
	10%: 0.119
	5%: 0.146
	2.5%: 0.176
	1%: 0.216

C:\Users\Wajeeha\anaconda3\lib\site-packages\statsmodels\tsa\stattools.py:1
661: FutureWarning: The behavior of using lags=None will change in the next
release. Currently lags=None is the same as lags='legacy', and so a sample-
size lag length is used. After the next release, the default will change to
be the same as lags='auto' which uses an automatic lag length selection met
hod. To silence this warning, either use 'auto' or 'legacy'
  warn(msg, FutureWarning)
C:\Users\Wajeeha\anaconda3\lib\site-packages\statsmodels\tsa\stattools.py:1
685: InterpolationWarning: p-value is smaller than the indicated p-value
  warn("p-value is smaller than the indicated p-value", InterpolationWarnin
g)
```

In [ ]: Typesetting math: 0%

Typesetting math: 0%

Appendix B – Gradient Boosting Regressor Model

August 27, 2021

Sample model for Stock A

Written by Pascual E. Camacho

1 Import Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from datetime import datetime
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error,accuracy_score

import warnings
warnings.filterwarnings("ignore")
```

2 Read Data

[2]:
```
symbol = 'A'

# Read data
data = pd.read_csv('train_data.csv')
data = data.dropna(axis=0)
data['id'] = data['symbol'] + '-' + data['day'].astype(str) + '-' + data['time']
data['day num'] = (data['day'] + 1).apply(str)
data = data.loc[data['symbol'] == symbol]
data
```

[2]:
```
        symbol    open    high     low   close  average      time  day  \
2590328      A  135.54  135.79  135.54  135.79   135.67  06:29:50    0
2590329      A  135.79  135.79  135.79  135.79   135.79  06:29:55    0
2590330      A  135.73  135.95  135.68  135.86   135.74  06:30:00    0
2590331      A  135.73  135.98  135.73  135.98   135.91  06:30:05    0
2590332      A  135.89  135.89  135.89  135.89   135.89  06:30:10    0
…          …       …       …       …       …        …         …    …
```

```
3017220      A  146.30  146.32  146.28  146.28   146.29  12:59:35   86
3017221      A  146.28  146.30  146.27  146.29   146.29  12:59:40   86
3017222      A  146.30  146.32  146.30  146.32   146.31  12:59:45   86
3017223      A  146.29  146.30  146.26  146.26   146.28  12:59:50   86
3017224      A  146.26  146.28  146.21  146.28   146.26  12:59:55   86

                    id day num
2590328   A-0-06:29:50       1
2590329   A-0-06:29:55       1
2590330   A-0-06:30:00       1
2590331   A-0-06:30:05       1
2590332   A-0-06:30:10       1
…                  …      …
3017220  A-86-12:59:35      87
3017221  A-86-12:59:40      87
3017222  A-86-12:59:45      87
3017223  A-86-12:59:50      87
3017224  A-86-12:59:55      87

[426897 rows x 10 columns]
```

3 Time + Date

[3]:
```
# Initialize year
year = "2021"

def day_to_date(day_num):
    res = []
    for i in range(len(day_num)):
        res.append(datetime.strptime(year + "-" + day_num[i], "%Y-%j").
 ↪strftime("%m-%d-%Y"))
    return res
```

[4]:
```
data['date'] = day_to_date(data["day num"].values)
data['date time'] = data.date.astype(str) + ' ' + data.time.astype(str)
data['sec'] = data.time.str[0:2].astype(int)*3600 + data.time.str[3:5].
 ↪astype(int)*60 + data.time.str[6:8].astype(int)
```

[5]:
```
data['date time'] = pd.to_datetime(data['date time'])
```

[6]:
```
data['pct change'] = data['open'].pct_change()
data['prev close diff'] = data['open'] - data['close'].shift(1)
data = data.dropna()
```

[7]:
```
data
```

```
[7]:         symbol    open    high     low   close  average      time  day  \
2590329           A  135.79  135.79  135.79  135.79   135.79  06:29:55    0
2590330           A  135.73  135.95  135.68  135.86   135.74  06:30:00    0
2590331           A  135.73  135.98  135.73  135.98   135.91  06:30:05    0
2590332           A  135.89  135.89  135.89  135.89   135.89  06:30:10    0
2590333           A  135.93  135.93  135.78  135.78   135.87  06:30:15    0
…               …      …       …       …       …        …         …   …
3017220           A  146.30  146.32  146.28  146.28   146.29  12:59:35   86
3017221           A  146.28  146.30  146.27  146.29   146.29  12:59:40   86
3017222           A  146.30  146.32  146.30  146.32   146.31  12:59:45   86
3017223           A  146.29  146.30  146.26  146.26   146.28  12:59:50   86
3017224           A  146.26  146.28  146.21  146.28   146.26  12:59:55   86

                     id day num        date           date time    sec  \
2590329    A-0-06:29:55       1  01-01-2021 2021-01-01 06:29:55  23395
2590330    A-0-06:30:00       1  01-01-2021 2021-01-01 06:30:00  23400
2590331    A-0-06:30:05       1  01-01-2021 2021-01-01 06:30:05  23405
2590332    A-0-06:30:10       1  01-01-2021 2021-01-01 06:30:10  23410
2590333    A-0-06:30:15       1  01-01-2021 2021-01-01 06:30:15  23415
…                     …     …           …                   …      …
3017220   A-86-12:59:35      87  03-28-2021 2021-03-28 12:59:35  46775
3017221   A-86-12:59:40      87  03-28-2021 2021-03-28 12:59:40  46780
3017222   A-86-12:59:45      87  03-28-2021 2021-03-28 12:59:45  46785
3017223   A-86-12:59:50      87  03-28-2021 2021-03-28 12:59:50  46790
3017224   A-86-12:59:55      87  03-28-2021 2021-03-28 12:59:55  46795

         pct change  prev close diff
2590329    0.001844             0.00
2590330   -0.000442            -0.06
2590331    0.000000            -0.13
2590332    0.001179            -0.09
2590333    0.000294             0.04
…                …                …
3017220    0.000000             0.00
3017221   -0.000137             0.00
3017222    0.000137             0.01
3017223   -0.000068            -0.03
3017224   -0.000205             0.00

[426896 rows x 15 columns]
```

4 Split data into Train & Test Sets

```
[8]: data.sort_values(['day','time'],ascending=[True,True],inplace=True)
```

```
X = data.
↪drop(['symbol','open','high','low','close','average','time','day','id','date','date␣
↪time','pct change','prev close diff'], axis=1)
# X = data.
↪drop(['symbol','open','high','low','close','average','time','day','id','date','date␣
↪time'], axis=1)
y = data['open']

data_size = len(data)
train_size = round(data_size*0.9)

# Train
X_train = X.iloc[:train_size, :]
y_train = y.iloc[:train_size]

# # Test
X_test = X.iloc[train_size:,:]
y_test = y.iloc[train_size:]

print(round(len(X_train) / data_size, 2)*100,'% of data set used for training')
```

```
90.0 % of data set used for training
```

5 Boosting Model (Gradient Boosting Regressor)

[9]:
```
boost = GradientBoostingRegressor(learning_rate=.05, n_estimators=2000,
            random_state=1, max_depth=3, validation_fraction=0.1)
boost.fit(X_train, y_train)
```

[9]:
```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.05, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=2000,
                          n_iter_no_change=None, presort='auto', random_state=1,
                          subsample=1.0, tol=0.0001, validation_fraction=0.1,
                          verbose=0, warm_start=False)
```

[10]:
```
predict = boost.predict(X_test)
print('Spread is:', predict.max() - predict.min())
predict
```

```
Spread is: 1.474479528696179
```

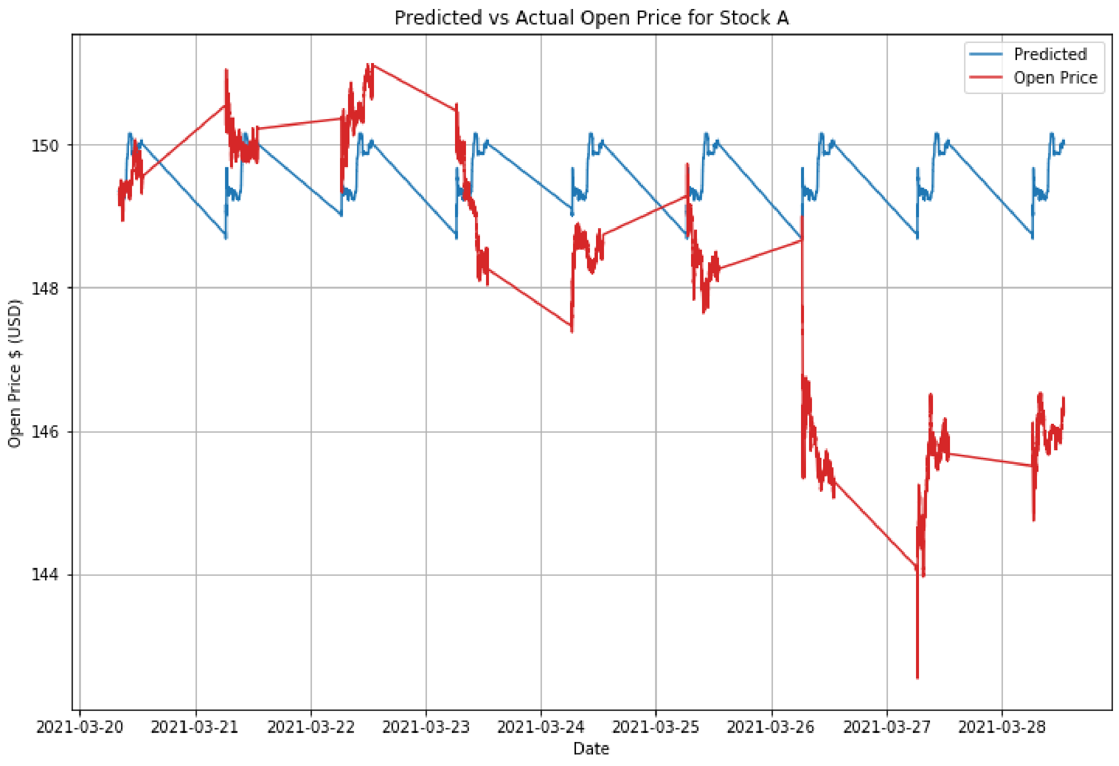
[10]:
```
array([149.34563449, 149.34563449, 149.35533058, …, 150.01025119,
       150.01025119, 150.01025119])
```

```
[11]: results = pd.DataFrame(y_test)
      results['prediction'] = predict
      results.max(), results.min()
```

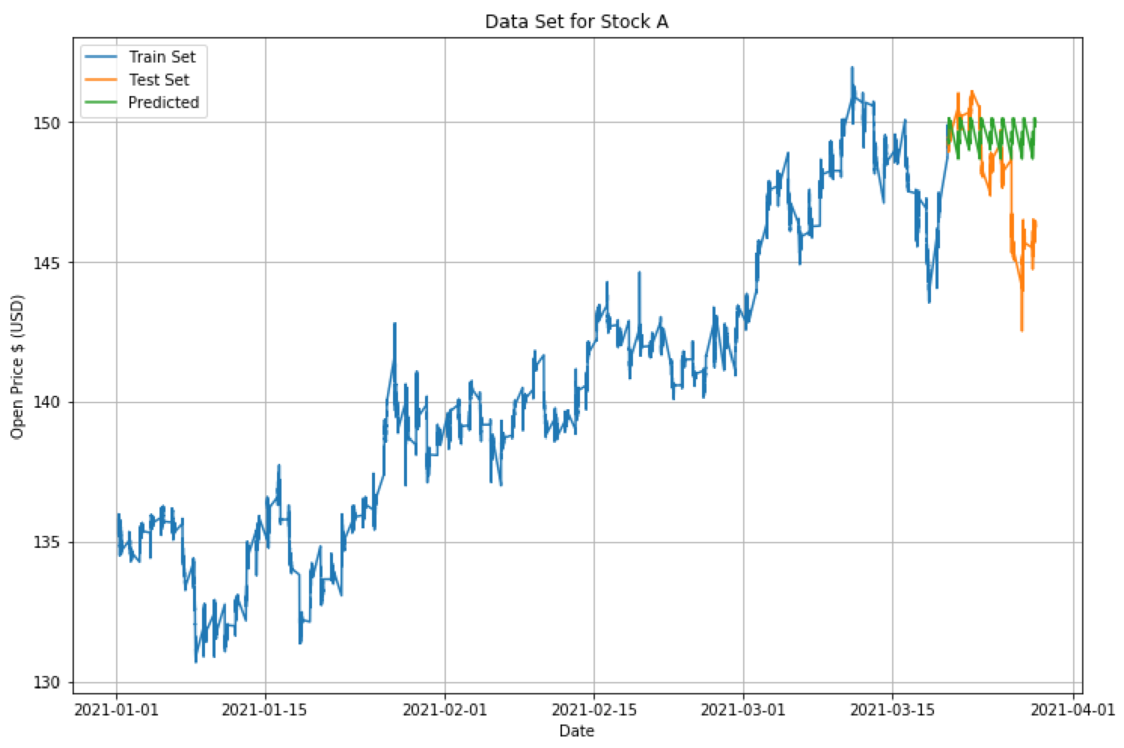
```
[11]: (open          151.120000
       prediction    150.154318
       dtype: float64, open          142.540000
       prediction    148.679838
       dtype: float64)
```

```
[25]: # Plot title
      stock = symbol
      plot_title = 'Predicted vs Actual Open Price for Stock ' + stock

      # Plot of 'predictions' vs 'open price'
      plt.figure(figsize=(12, 8))
      plt.plot(data['date time'].iloc[train_size:], predict[:], label='Predicted',␣
       ↪color='tab:blue')
      plt.plot(data['date time'].iloc[train_size:], data['open'].iloc[train_size:],␣
       ↪label='Open Price', color='tab:red')
      plt.xlabel('Date')
      plt.ylabel('Open Price $ (USD)')
      plt.title(plot_title)
      plt.legend()
      plt.grid()
      plt.savefig('boost_a')
```



[26]:
```
# Test And Train Data
plt.figure(figsize=(12, 8))
# fig = plt.figure()
plt.plot(data['date time'].iloc[:train_size], y_train, color='tab:blue',␣
 ↪label='Train Set')
plt.plot(data['date time'].iloc[train_size:], y_test, color='tab:orange',␣
 ↪label='Test Set')
plt.plot(data['date time'].iloc[train_size:], predict[:], label='Predicted',␣
 ↪color='tab:green')
plt.xlabel('Date')
plt.ylabel('Open Price $ (USD)')
plot_title = 'Data Set for Stock ' + stock
plt.title(plot_title)
plt.legend()
plt.grid()
```



[14]:
```
# Mean Squared Error
import math
mse = mean_squared_error(y_test, predict)
print('MSE is', mse)
```

```
MSE is 6.1184486982297575
```

6 Forecasting

[15]:
```
# Initialize unique time DataFrame
unique_time = pd.DataFrame(data.time.unique(), columns=['time'])
unique_time.sort_values(by=['time'],ascending=[True],inplace=True)

last_day = int(data['day num'].iloc[-1]) # last day in original data set (day␣
 ↪86)
dayspredict = (1, 2, 3, 4, 5, 6, 7, 8, 9) # days to forecast

# Initialize list to store DataFrames
stack = []

# Loop through days to forecast and create a DataFrame for each
# Append each DataFrame to list
for i in range(len(dayspredict)):
    # Create dataframe with time column
    temp = unique_time.copy()
    temp.sort_values(by=['time'],ascending=[True],inplace=True)

    first_future_day = last_day + dayspredict[i] # next future day to forecast

    # Add the day number in 'day num' column (same value for all time entries)
    temp['day num'] = first_future_day

    stack.append(temp)
    del temp # clear temporary dataframe
```

[16]:
```
zero = stack[0].copy()
one = stack[1].copy()
two = stack[2].copy()
three = stack[3].copy()
four = stack[4].copy()
five = stack[5].copy()
six = stack[6].copy()
seven = stack[7].copy()
eight = stack[8].copy()
future_time = pd.DataFrame()
future_time = future_time.append([zero,one,two,three,four,five,six,seven,eight])
```

[17]:
```
future_time['sec'] = future_time.time.str[0:2].astype(int)*3600 + future_time.
 ↪time.str[3:5].astype(int)*60 + future_time.time.str[6:8].astype(int)
future_time['day num'].astype(int)
```

[18]:
```
predict = boost.predict(future_time.drop(['time'], axis=1))
print('Spread is:', predict.max() - predict.min())
predict
```

[19]:
```
# Format ID
future_time['predicted'] = predict
future_time['symbol'] = symbol
future_time['ID'] = future_time['symbol'] + '-' + future_time['day num'].
 ↪astype(str) + '-' + future_time['time']
```

[20]:
```
export = pd.DataFrame()
export['id'] = future_time['ID']
export['open'] = future_time['predicted']
```

[21]:
```
# Export as csv file
file_name = 'export_stock_' + symbol + '.csv'
export.to_csv(file_name, index=False)
```

[22]:
```
export['day'] = (export.id.str[2:4].astype(int)).apply(str)
```

[23]:
```
export['date'] = day_to_date(export['day'].values)
export['time'] = export.id.str[5:]
export['date time'] = export.date.astype(str) + ' ' + export.time.astype(str)
export
```

Appendix C - Latent state model preprocessing

Fernando Rodriguez

8/27/2021

Load the data

```
"train_data.csv" %>%
  read_csv() ->
  train_df
```

```
## Rows: 4330254 Columns: 8
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): symbol
## dbl  (6): open, high, low, close, average, day
## time (1): time
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
cv_df <- train_df
```

Summary stats

```
summary(train_df)
```

```
##     symbol               open             high              low        
##  Length:4330254     Min.   : 77.05   Min.   : 77.05   Min.   : 77.04  
##  Class :character   1st Qu.:107.99   1st Qu.:107.99   1st Qu.:107.98  
##  Mode  :character   Median :150.81   Median :150.82   Median :150.81  
##                     Mean   :147.99   Mean   :147.99   Mean   :147.98  
##                     3rd Qu.:177.20   3rd Qu.:177.21   3rd Qu.:177.20  
##                     Max.   :258.45   Max.   :258.46   Max.   :258.45  
##      close           average            time               day       
##  Min.   : 77.04   Min.   : 77.05   Length:4330254    Min.   : 0.00  
##  1st Qu.:107.99   1st Qu.:107.99   Class1:hms        1st Qu.:21.00  
##  Median :150.81   Median :150.81   Class2:difftime   Median :43.00  
##  Mean   :147.99   Mean   :147.99   Mode  :numeric    Mean   :43.03  
##  3rd Qu.:177.20   3rd Qu.:177.20                     3rd Qu.:65.00  
##  Max.   :258.46   Max.   :258.46                     Max.   :86.00  
```

```
sum(is.na(train_df))
```

```
## [1] 0
```

```
head(train_df)
```

```
## # A tibble: 6 x 8
##   symbol  open  high   low close average time       day
##   <chr>  <dbl> <dbl> <dbl> <dbl>   <dbl> <time>   <dbl>
## 1 B       102.  102.  102.  102.    102. 06:00:00     0
## 2 B       102.  102.  102.  102.    102. 06:00:05     0
## 3 B       102.  102.  102.  102.    102. 06:00:10     0
## 4 B       102.  102.  102.  102.    102. 06:00:15     0
## 5 B       102.  102.  102.  102.    102. 06:00:20     0
## 6 B       102.  102.  102.  102.    102. 06:00:25     0
```

Diagnostics and visualization

Correlation of observed variables

```
#acf(train_df %>% select(!symbol))
candle_cor <- cor(as.matrix(train_df %>% select(!symbol) %>% select(!time) %>% select(!day)))
candle_cor
```

```
##              open      high       low     close average
## open    1.0000000 0.9999999 0.9999999 0.9999999       1
## high    0.9999999 1.0000000 0.9999999 0.9999999       1
## low     0.9999999 0.9999999 1.0000000 0.9999999       1
## close   0.9999999 0.9999999 0.9999999 1.0000000       1
## average 1.0000000 1.0000000 1.0000000 1.0000000       1
```

Max difference with average:

```
max(
  abs(
    (train_df %>% pull(open)) - (train_df %>% pull(average))
  )
)
```

```
## [1] 1.51
```

```
max(
  abs(
    (train_df %>% pull(close)) - (train_df %>% pull(average))
  )
)
```

```
## [1] 2.02
```

Spread and percentage change in observations:

```
train_df %>%
  mutate(
    spread = high - low,
    spread.100 = (high - low) / open,
    change = close - open,
    change.100 = (close - open) / open
  ) ->
  train_df

max(train_df %>% pull(spread.100))
```

```
## [1] 0.0245045
```

```r
max(train_df %>% pull(change.100))
```

```
## [1] 0.007727673
```

Ratio of observations with no change:

```r
N_obs <- nrow(train_df)
N_obs.nochange <- sum((train_df %>% pull(change)) == 0)

N_obs.nochange / N_obs
```

```
## [1] 0.7030186
```

Convert data frame into wide format:

```r
train_df %>%
  select(!close) %>%
  pivot_wider(
    names_from = symbol,
    values_from = c(
      open,
      high,
      low,
      average,
      spread,
      spread.100,
      change,
      change.100
    )
  ) ->
  train_wide
```

Correlation between symbols:

```r
train_wide %>%
  transmute(
    A = open_A,
    B = open_B,
    C = open_C,
    D = open_D,
    E = open_E,
    F = open_F,
    G = open_G,
    H = open_H,
    I = open_I,
    J = open_J
  ) %>%
  na.omit() %>%
  as.matrix() %>%
  cor() ->
  symbol_cor

symbol_cor
```

```
##           A         B         C          D         E           F
## A 1.0000000 0.9161873 0.7543735 0.92576878 0.8431415 0.925292626
## B 0.9161873 1.0000000 0.7687629 0.90050891 0.7754861 0.895155908
## C  0.7543735  0.7687629  1.0000000  0.71172813  0.7825631  0.710158373
## D  0.9257688  0.9005089  0.7117281  1.00000000  0.7915907  0.965549020
## E  0.8431415  0.7754861  0.7825631  0.79159071  1.0000000  0.769469741
## F  0.9252926  0.8951559  0.7101584  0.96554902  0.7694697  1.000000000
## G  0.0651154  0.1771003  0.5815256 -0.05200856  0.1108797  0.007067507
## H -0.5864132 -0.5493481 -0.1465759 -0.70619468 -0.3566818 -0.698956048
## I  0.4355926  0.4296730  0.5322960  0.21717358  0.3851715  0.363103328
## J  0.5209509  0.4733328  0.2626838  0.66832706  0.3695837  0.641024182
##              G           H           I          J
## A  0.065115404 -0.58641321  0.43559262  0.5209509
## B  0.177100329 -0.54934814  0.42967296  0.4733328
## C  0.581525554 -0.14657585  0.53229601  0.2626838
## D -0.052008557 -0.70619468  0.21717358  0.6683271
## E  0.110879679 -0.35668179  0.38517148  0.3695837
## F  0.007067507 -0.69895605  0.36310333  0.6410242
## G  1.000000000  0.56338582  0.57582838 -0.4171025
## H  0.563385823  1.00000000  0.02916504 -0.7981201
## I  0.575828375  0.02916504  1.00000000 -0.1492785
## J -0.417102537 -0.79812008 -0.14927849  1.0000000
```

Ratio of periods with no change:

```r
N_per <- nrow(train_wide)

train_wide %>%
  mutate(
    change_A = ifelse(
      is.na(change_A),
      0,
      change_A
    ),
    change_B = ifelse(
      is.na(change_B),
      0,
      change_B
    ),
    change_C = ifelse(
      is.na(change_C),
      0,
      change_C
    ),
    change_D = ifelse(
      is.na(change_D),
      0,
      change_D
    ),
    change_E = ifelse(
      is.na(change_E),
      0,
      change_E
    ),
    change_F = ifelse(
      is.na(change_F),
      0,
      change_F
    ),
    change_G = ifelse(
      is.na(change_G),
      0,
      change_G
    ),
    change_H = ifelse(
      is.na(change_H),
      0,
      change_H
    ),
    change_I = ifelse(
      is.na(change_I),
      0,
      change_I
    ),
    change_J = ifelse(
      is.na(change_J),
      0,
      change_J
    )
  ) %>%
  mutate(
    totchange = change_A + change_B + change_C + change_D + change_E + change_F + change_G + change_H +
  ) ->
  train_wide

N_per.nochange <- sum((train_wide %>% pull(totchange)) == 0)

N_per.nochange
```

```
## [1] 66805
```

```
N_per.nochange / N_per
```

```
## [1] 0.1523559
```

Modify sample space

Add features for model construction:

```
moving_avg <- function(x, nper) {
  Xsum <- x
  for(per in 1:nper) {
    Xsum <- Xsum + lag(
      x = x,
      n = per,
      default = 0
    )
  }
  return(
    Xsum / (nper + 1)
  )
}
```

```r
cv_df %>%
  transmute(
    symbol = symbol,
    price = open,
    time = time,
    day = day
  ) %>%
  pivot_wider(
    names_from = symbol,
    values_from = c(
      price
    )
  ) %>%
  select(time, A, B, C, D, E, F, G, H, I, J) %>%
  mutate(
    change.A = (A - lag(A)) / lag(A),
    change.B = (B - lag(B)) / lag(B),
    change.C = (C - lag(C)) / lag(C),
    change.D = (D - lag(D)) / lag(D),
    change.E = (E - lag(E)) / lag(E),
    change.F = (F - lag(F)) / lag(F),
    change.G = (G - lag(G)) / lag(G),
    change.H = (H - lag(H)) / lag(H),
    change.I = (I - lag(I)) / lag(I),
    change.J = (J - lag(J)) / lag(J)
  ) %>%
  mutate(
    mavg.A = moving_avg(A, 60),
    mavg.B = moving_avg(B, 60),
    mavg.C = moving_avg(C, 60),
    mavg.D = moving_avg(D, 60),
    mavg.E = moving_avg(E, 60),
    mavg.F = moving_avg(F, 60),
    mavg.G = moving_avg(G, 60),
    mavg.H = moving_avg(H, 60),
    mavg.I = moving_avg(I, 60),
    mavg.J = moving_avg(J, 60)
  ) %>%
  mutate(
    hourchange.A = (A - lag(A, 720)) / lag(A, 720),
    hourchange.B = (B - lag(B, 720)) / lag(B, 720),
    hourchange.C = (C - lag(C, 720)) / lag(C, 720),
    hourchange.D = (D - lag(D, 720)) / lag(D, 720),
    hourchange.E = (E - lag(E, 720)) / lag(E, 720),
    hourchange.F = (F - lag(F, 720)) / lag(F, 720),
    hourchange.G = (G - lag(G, 720)) / lag(G, 720),
    hourchange.H = (H - lag(H, 720)) / lag(H, 720),
    hourchange.I = (I - lag(I, 720)) / lag(I, 720),
    hourchange.J = (J - lag(J, 720)) / lag(J, 720)
  ) %>%
  mutate(
    tenhchange.A = (A - lag(A, 7200)) / lag(A, 7200),
    tenhchange.B = (B - lag(B, 7200)) / lag(B, 7200),
    tenhchange.C = (C - lag(C, 7200)) / lag(C, 7200),
    tenhchange.D = (D - lag(D, 7200)) / lag(D, 7200),
    tenhchange.E = (E - lag(E, 7200)) / lag(E, 7200),
    tenhchange.F = (F - lag(F, 7200)) / lag(F, 7200),
    tenhchange.G = (G - lag(G, 7200)) / lag(G, 7200),
    tenhchange.H = (H - lag(H, 7200)) / lag(H, 7200),
    tenhchange.I = (I - lag(I, 7200)) / lag(I, 7200),
    tenhchange.J = (J - lag(J, 7200)) / lag(J, 7200)
  ) ->
  cv_wide
```

Remove rows with NA values and save as .csv files:

```
cv_wide %>%
  na.omit() ->
  cv_augmented

cv_augmented %>%
  mutate(
    time = as.integer(time)
  ) %>%
  write_csv("augmented_data.csv")

cv_augmented %>%
  select(A, B, C, D, E, F, G, H, I, J) ->
  cv_clean

cv_clean %>%
  write_csv("clean_data.csv")
```

Sample viewing

```
"sample_predictions.csv" %>%
  read_csv() ->
  sample_df
```

```
## Rows: 453600 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): id
## dbl (1): open
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
sample_df %>%
  separate(
    id,
    c("Stock", "Day", "Time"),
    sep = "-"
  ) %>%
  pivot_wider(
    names_from = Stock,
    values_from = c(
      open
    )
  ) %>%
  arrange(
    Day,
    Time
  ) ->
  sample_mod

sample_mod %>%
  select(!Day) %>%
  write_csv("predict_format.csv")

head(sample_mod, 10)
```

```
## # A tibble: 10 x 12
##    Day   Time         A     B     C     D     E     F     G     H     I     J
##    <chr> <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 0     06:00:00     0     0     0     0     0     0     0     0     0     0
##  2 0     06:00:05     0     0     0     0     0     0     0     0     0     0
##  3 0     06:00:10     0     0     0     0     0     0     0     0     0     0
##  4 0     06:00:15     0     0     0     0     0     0     0     0     0     0
##  5 0     06:00:20     0     0     0     0     0     0     0     0     0     0
##  6 0     06:00:25     0     0     0     0     0     0     0     0     0     0
##  7 0     06:00:30     0     0     0     0     0     0     0     0     0     0
##  8 0     06:00:35     0     0     0     0     0     0     0     0     0     0
##  9 0     06:00:40     0     0     0     0     0     0     0     0     0     0
## 10 0     06:00:45     0     0     0     0     0     0     0     0     0     0
```

Formatting predictions

```
correct_id <- function(df){
  df %>%
    separate(
      id,
      c("Stock", "Day", "Time"),
      sep = "-"
    ) %>%
    mutate(
      Day = as.integer(Day) - 88
    ) %>%
    unite(
      col = "id",
      Stock,
      Day,
      Time,
      sep = "-"
    ) ->
    df
  return(df)
}
```

```
Ahat_df <- read_csv("Predictions/export_stock_A.csv") %>% correct_id()
```

```
## Rows: 45360 Columns: 2

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): id
## dbl (1): open

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Bhat_df <- read_csv("Predictions/export_stock_B.csv") %>% correct_id()
```

```
## Rows: 45360 Columns: 2

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): id
## dbl (1): open

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Chat_df <- read_csv("Predictions/export_stock_C.csv") %>% correct_id()
```

```
## Rows: 45360 Columns: 2

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): id
## dbl (1): open

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Dhat_df <- read_csv("Predictions/export_stock_D.csv") %>% correct_id()
```

```
## Rows: 45360 Columns: 2

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): id
## dbl (1): open

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Ehat_df <- read_csv("Predictions/export_stock_E.csv") %>% correct_id()
```

```
## Rows: 45360 Columns: 2

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): id
## dbl (1): open
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Fhat_df <- read_csv("Predictions/export_stock_F.csv") %>% correct_id()
```

```
## Rows: 45360 Columns: 2
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): id
## dbl (1): open
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Ghat_df <- read_csv("Predictions/export_stock_G.csv") %>% correct_id()
```

```
## Rows: 45360 Columns: 2
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): id
## dbl (1): open
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Hhat_df <- read_csv("Predictions/export_stock_H.csv") %>% correct_id()
```

```
## Rows: 45360 Columns: 2
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): id
## dbl (1): open
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Ihat_df <- read_csv("Predictions/export_stock_I.csv") %>% correct_id()
```

```
## Rows: 45360 Columns: 2
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): id
## dbl (1): open
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
Jhat_df <- read_csv("Predictions/export_stock_J.csv") %>% correct_id()
```

```
## Rows: 45360 Columns: 2
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): id
## dbl (1): open
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
predictions_df <- bind_rows(
  Ahat_df,
  Bhat_df,
  Chat_df,
  Dhat_df,
  Ehat_df,
  Fhat_df,
  Ghat_df,
  Hhat_df,
  Ihat_df,
  Jhat_df
)

sample_df %>%
  transmute(
    id = id,
    old = open
  ) %>%
  left_join(
    predictions_df,
    by = "id"
  ) %>%
  select(
    id,
    open
  ) ->
  formatted_pred_df

formatted_pred_df %>%
  write_csv("predicted_open.csv")
```

Appendix D - Latent state model

August 27, 2021

Written by Fernando Rodriguez Silva Santisteban

Reused portions of code from final project for STATS 271 in Spring 2021, also written by Fernando Rodriguez Silva Santisteban.

0.1 Load transformed data

[1]:
```
import pandas as pd
df_augmented = pd.read_csv('augmented_data.csv')
df_augmented
```

[1]:
```
         time       A       B       C       D       E       F       G      H  \
0       36000  134.53  101.01   90.03  218.37  119.48  164.09  159.05  88.53
1       36005  134.53  101.01   90.01  218.37  119.48  164.08  159.03  88.51
2       36010  134.53  101.01   90.01  218.35  119.48  164.08  159.01  88.51
3       36015  134.53  101.00   90.01  218.35  119.48  164.08  159.01  88.51
4       36020  134.53  101.00   90.00  218.35  119.48  164.08  159.03  88.53
…         …      …       …       …       …      …       …      …      …
358536  46775  146.30  113.11  100.02  257.86  129.74  194.37  171.08  78.26
358537  46780  146.28  113.11  100.04  257.84  129.73  194.37  171.04  78.28
358538  46785  146.30  113.11  100.04  257.86  129.73  194.37  171.09  78.29
358539  46790  146.29  113.12  100.04  257.89  129.74  194.39  171.12  78.29
358540  46795  146.26  113.13  100.05  257.85  129.78  194.36  171.12  78.28

             I  …  tenhchange.A  tenhchange.B  tenhchange.C  tenhchange.D  \
0       161.19  …     -0.007452     -0.015689     -0.001774     -0.007093
1       161.19  …     -0.007452     -0.015497     -0.002438     -0.007454
2       161.19  …     -0.007671     -0.015305     -0.002438     -0.007635
3       161.19  …     -0.008037     -0.016074     -0.002549     -0.007590
4       161.19  …     -0.008257     -0.016362     -0.002549     -0.007635
…          …  …            …             …             …             …
358536  172.99  …      0.004187      0.012170     -0.004578      0.013999
358537  172.99  …      0.004119      0.012170     -0.004379      0.013921
358538  173.01  …      0.004256      0.012170     -0.004379      0.014199
358539  172.99  …      0.004118      0.012260     -0.004280      0.014317
358540  172.96  …      0.003981      0.012349     -0.004180      0.014359

        tenhchange.E  tenhchange.F  tenhchange.G  tenhchange.H  tenhchange.I  \
0           0.008951    -0.003885     0.002332     0.001244    -0.002661
1           0.008696    -0.004309     0.001827     0.000791    -0.003401
2           0.008270    -0.004309     0.001953     0.000678    -0.003154
3           0.008100    -0.004309     0.001953     0.000905    -0.003647
4           0.008525    -0.004309     0.002079     0.001131    -0.003339
…              …           …           …            …           …
358536      0.003248     0.017591    -0.012297    -0.003057    -0.013684
358537      0.003326     0.017591    -0.012414    -0.002803    -0.013740
358538      0.003558     0.017644    -0.012182    -0.002675    -0.013682
358539      0.003636     0.017749    -0.012009    -0.002675    -0.013852
358540      0.004101     0.017858    -0.012009    -0.002676    -0.013967

        tenhchange.J
0          -0.014079
1          -0.014134
2          -0.014246
3          -0.014246
4          -0.013969
…              …
358536     -0.023013
358537     -0.023013
358538     -0.022856
358539     -0.022751
358540     -0.022543

[358541 rows x 51 columns]
```

0.2 Construct data mining structure

[2]:
```python
import numpy as np
from scipy.stats import multivariate_normal as mvnorm
from scipy.special import logsumexp
from tensorflow import keras
import tensorflow as tf
import tensorflow_probability.python.distributions as tfd
from tqdm.auto import trange

# Helper function to normalize data, storing recovery parameters
def feat_normalize(data):
    mu = np.mean(
      a = data,
      axis = 0
    )
    sigma = np.std(
      a = data,
      axis = 0
    )
    X = np.array(data)
    X = (X - mu[None,:]) / sigma[None,:]
    recov_param = {
    'mu': mu,
    'sigma': sigma
    }
    return X, recov_param

# Helper function to denormalize data based on recovery parameters
def feat_denormalize(normdata, recov_param):
    mu = recov_param['mu']
    sigma = recov_param['sigma']
    X = np.array(normdata)
    X = (X * sigma[None,:]) + mu[None,:]
    return X

# Helper function to renormalize an observation based on recovery parameters
def feat_renormalize(X, recov_param):
    mu = recov_param['mu']
    sigma = recov_param['sigma']
    X = (X - mu) / sigma
    return X

# Construct class for VAE maximizing observed data likelihood in each step
class VAE(keras.Model):

    def __init__(self,
                 encoder,
                 latent_states,
                 recov_param,
                 init_mu,
                 init_sigma):
        super(VAE, self).__init__()
        self.encoder = encoder
        self.latent_states = latent_states
        self.recov_param = recov_param
        self.ll_tracker = keras.metrics.Mean(name = "LogLikelihood")
        self.mu = tf.Variable(
            initial_value= init_mu,
            trainable= True,
            name= "Means"
        )
        self.sigma = tf.Variable(
            initial_value= init_sigma,
            trainable= True,
            name= "Std Deviations"
        )
    def get_mu(self):
        return self.mu

    def get_sigma(self):
        sigma = tf.linalg.diag(
            self.sigma ** 2
        )
        return sigma

    @property
    def metrics(self):
        return [self.ll_tracker]


    def train_step(self, data):
        with tf.GradientTape() as tape:
            tape.watch((self.mu, self.sigma))
            prices = data[:,1:11]
            z = self.encoder(data)
            #print(z)
            N_batch = data.shape[0]
            K = self.latent_states
            # Find conditional log_likelihood of X given z
            mvn = tfd.MultivariateNormalFullCovariance(
                loc= self.mu,
                covariance_matrix = tf.linalg.diag(
                    self.sigma ** 2
                )
            )
            #print(self.sigma ** 2)
            features = tf.convert_to_tensor(
                prices[1:,:] - prices[0:-1,:],
                dtype= tf.float32
            )
            feature_update = mvn.log_prob(
                features[:,None,:]
            )
            log_likelihoods = tf.concat(
                [tf.zeros((1,K), dtype= tf.float32), feature_update],
                axis = 0
            )
            #print(log_likelihoods)
            # Find probability of z given observed X
            initial_dist_np = np.ones(K) * 1/K,
            initial_dist = tf.convert_to_tensor(initial_dist_np,
                                                dtype=tf.float32)
            alphas = tf.math.log(initial_dist)
            alphas = tf.concat(
                [alphas, tf.math.log(z[0:-1,:])],
                axis = 0
            )
            #print(alphas)
            # Find likelihood for x
            A = tf.math.reduce_logsumexp(
                alphas + log_likelihoods,
                axis= 1
            )
            LL = tf.math.reduce_sum(A)
            neg_LL = -LL
            #print(LL)
        grads = tape.gradient(neg_LL, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        self.ll_tracker.update_state(LL)
        return {
            "LogLikelihood": self.ll_tracker.result()
        }

def fit_model(data, latent_states):

    X, recov_param = feat_normalize(data)
    N = np.shape(data)[0]
    M = 10
    M_aug = np.shape(data)[1]
    K = latent_states

    # Construct the encoder Neural Network (observed augmented vector to latent
    # space)
    encoder_input = keras.Input(
        shape=(M_aug),
        name="observed_vector"
    )
    hidden_1 = keras.layers.Dense(
        128,
        activation= "tanh",
        kernel_initializer= keras.initializers.RandomNormal(),
        kernel_regularizer= keras.regularizers.l1_l2(l1=0, l2=1e-2),
        name= "hidden_1"
    )(encoder_input)
    hidden_2 = keras.layers.Dense(
        64,
        activation= "tanh",
        kernel_initializer= keras.initializers.RandomNormal(),
        kernel_regularizer= keras.regularizers.l1_l2(l1=0, l2=1e-2),
        name="hidden_2"
    )(hidden_1)
    latent = keras.layers.Dense(
        K,
        activation= "softmax",
        kernel_initializer= keras.initializers.RandomNormal(),
        name = "latent",
        dtype= tf.float32
    )(hidden_2)
    encoder = keras.Model(
        inputs=encoder_input,
        outputs=latent,
        name="encoder"
    )

    # Run a variational autoencoder with E-step based on partially observable
    # Markov Model
    vae = VAE(
        encoder= encoder,
        latent_states= latent_states,
        recov_param = recov_param,
        init_mu = tf.convert_to_tensor(
            mvnorm.rvs(
                mean= np.zeros(M),
                cov = np.identity(M) * 0.1,
                size= K
            ),
            dtype = tf.float32
        ),
        init_sigma= tf.convert_to_tensor(
            np.broadcast_to(
                np.ones(M) * 2,
                (K,M)
            ),
            dtype = tf.float32
        )
    )
    vae.compile(optimizer = keras.optimizers.Adam(
        learning_rate= 0.001
      ), run_eagerly= True)
    vae.fit(X, epochs = 100, batch_size = 8192)
    parameters = {
        'mu': vae.get_mu(),
        'sigma': vae.get_sigma()
    }
    return encoder, parameters, recov_param
```

```python
# Sampling approach to prediction
def OOS_montecarlo(daytimes, train_data, encoder, parameters, N_sim):
    N_per = len(daytimes)
    prices = np.array(train_data)[:,1:11]
    Y = np.zeros((1,N_per,np.shape(prices)[1]))
    rng = np.random.default_rng(42)
    for sim in trange(N_sim):
        X_train = np.array(train_data)
        X, recov_param = feat_normalize(train_data)
        X = X[-1,:]
        last_price = prices[-1,:]
        Y_sim = np.zeros((1,len(last_price)))
        for per in trange(N_per):
            z_prob = encoder(X[None,:])
            #print(z_prob[0,:])
            z = rng.choice(
                a = range(len(z_prob[0,:])),
                p = np.round(z_prob[0,:],6) / np.sum(np.round(z_prob[0,:],6))
              )
            mu = parameters['mu'][z]
            sigma = parameters['sigma'][z]
            step = rng.multivariate_normal(
              mean = mu,
              cov = sigma
            )
            new_price = last_price + step
            #print(new_price)
            Y_sim = np.append(Y_sim, new_price[None,:], axis = 0)
            X = np.array(daytimes[per])
            X = np.append(X, np.array(new_price).flatten())
            change = step / last_price
            X = np.append(X, np.array(change).flatten())
            mavg = (np.sum(X_train[-59:-1,1:11], axis=0)+new_price)/60
            X = np.append(X, np.array(mavg).flatten())
            hourchange = (new_price - X_train[-720,1:11]) / X_train[-720,1:11]
            X = np.append(X, np.array(hourchange).flatten())
            tenhchange = (new_price - X_train[-7200,1:11]) / X_train[-7200,1:11]
            X = np.append(X, np.array(tenhchange).flatten())
            #print(X)
            X_train = np.append(X_train, X[None,:], axis = 0)
            X = feat_renormalize(X, recov_param)
            last_price = new_price
            #print(Y_sim)
        Y = np.append(Y, Y_sim[None,1:,:], axis = 0)
    Y = np.mean(a = Y[1:,:,:], axis = 0)
    return Y
```

0.3 Cross-validate hyperparameters

[3]:
```
# Split data into training and testing set
i_cutoff = int(np.round(0.9 * np.shape(df_augmented)[0]))
train_data = df_augmented[0:(i_cutoff-1)]
test_data = df_augmented[i_cutoff:]

# Helper function to get likelihood of test data
def test_likelihood(test_data, encoder, param, recov_param):
    prices = np.array(test_data)[:,1:11]
    mu = recov_param['mu']
    sigma = recov_param['sigma']
    X = np.array(test_data)
    X = (X - mu[None,:]) / sigma[None,:]
    z = encoder(X)
    N_test = np.shape(X)[0]
    K = np.shape(param['mu'])[0]
    # Find conditional log_likelihood of X given z
    mvn = tfd.MultivariateNormalFullCovariance(
        loc= param['mu'],
        covariance_matrix = param['sigma']
    )
    features = tf.convert_to_tensor(
        prices[1:,:] - prices[0:-1,:],
        dtype= tf.float32
    )
    feature_update = mvn.log_prob(
        features[:,None,:]
    )
    log_likelihoods = tf.concat(
        [tf.zeros((1,K), dtype= tf.float32), feature_update],
        axis = 0
    )
    # Find probability of z given observed X
    initial_dist_np = np.ones(K) * 1/K,
    initial_dist = tf.convert_to_tensor(initial_dist_np,
                                        dtype=tf.float32)
    alphas = tf.math.log(initial_dist)
    alphas = tf.concat(
        [alphas, tf.math.log(z[0:-1,:])],
        axis = 0
    )
    # Find likelihood for x
    A = tf.math.reduce_logsumexp(
        alphas + log_likelihoods,
        axis= 1
    )
    return(tf.math.reduce_sum(A))

# Cross-validation based on likelihood of test data
cv_states = [4, 8, 12, 16]
encoders = []
parameters = []
recov_params = []
likelihoods = []
for lat in cv_states:
    cv_encoder, cv_param, cv_recov_param = fit_model(
        train_data,
        lat
    )
    encoders.append(cv_encoder)
    parameters.append(cv_param)
    recov_params.append(cv_recov_param)
    ll = test_likelihood(
        test_data,
        cv_encoder,
        cv_param,
        cv_recov_param
    )
    likelihoods.append(ll)
```

```
<ipython-input-2-5eb3a4405722>:20: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  X = (X - mu[None,:]) / sigma[None,:]

Epoch 1/100
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-
packages\tensorflow_probability\python\distributions\distribution.py:298:
MultivariateNormalFullCovariance.__init__ (from
tensorflow_probability.python.distributions.mvn_full_covariance) is deprecated
and will be removed after 2019-12-01.
Instructions for updating:
`MultivariateNormalFullCovariance` is deprecated, use
`MultivariateNormalTriL(loc=loc,
scale_tril=tf.linalg.cholesky(covariance_matrix))` instead.
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-149181.0469
Epoch 2/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-147349.1250
Epoch 3/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-145749.6094
Epoch 4/100
```

```
40/40 [==============================] - 3s 77ms/step - LogLikelihood:
-144162.8438
Epoch 5/100
40/40 [==============================] - 3s 77ms/step - LogLikelihood:
-142687.3281
Epoch 6/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-141217.4531
Epoch 7/100
40/40 [==============================] - 3s 77ms/step - LogLikelihood:
-139726.9688
Epoch 8/100
40/40 [==============================] - 3s 78ms/step - LogLikelihood:
-138277.0938
Epoch 9/100
40/40 [==============================] - 3s 79ms/step - LogLikelihood:
-136826.7656
Epoch 10/100
40/40 [==============================] - 3s 81ms/step - LogLikelihood:
-135379.0781
Epoch 11/100
40/40 [==============================] - 3s 80ms/step - LogLikelihood:
-133912.0000
Epoch 12/100
40/40 [==============================] - 3s 78ms/step - LogLikelihood:
-132547.9375
Epoch 13/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-131157.4062
Epoch 14/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-129809.2734
Epoch 15/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-128527.4609
Epoch 16/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-127211.4219
Epoch 17/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-125983.5859
Epoch 18/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-124803.5156
Epoch 19/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-123674.7344
Epoch 20/100
```

```
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-122688.8125
Epoch 21/100
40/40 [==============================] - 3s 78ms/step - LogLikelihood:
-121621.1406
Epoch 22/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-120692.4609
Epoch 23/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-119860.8125
Epoch 24/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-119055.4531
Epoch 25/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-118270.1719
Epoch 26/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-117543.6875
Epoch 27/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-116794.2891
Epoch 28/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-116169.9766
Epoch 29/100
40/40 [==============================] - 3s 78ms/step - LogLikelihood:
-115515.7031
Epoch 30/100
40/40 [==============================] - 3s 78ms/step - LogLikelihood:
-114680.7656
Epoch 31/100
40/40 [==============================] - 3s 78ms/step - LogLikelihood:
-114007.4375
Epoch 32/100
40/40 [==============================] - 3s 77ms/step - LogLikelihood:
-113271.6250
Epoch 33/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-112720.4844
Epoch 34/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-112219.7656
Epoch 35/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-111856.3516
Epoch 36/100
```

```
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-111557.7109
Epoch 37/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-111166.4766
Epoch 38/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-110793.5781
Epoch 39/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-110462.7031
Epoch 40/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-110191.6484
Epoch 41/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-109977.3984
Epoch 42/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-109689.5391
Epoch 43/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-109424.1641
Epoch 44/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-109242.1406
Epoch 45/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-109112.9844
Epoch 46/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-109038.6641
Epoch 47/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-108924.6250
Epoch 48/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-108789.6250
Epoch 49/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-108637.5391
Epoch 50/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-108598.1094
Epoch 51/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-108552.0781
Epoch 52/100
```

```
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-108416.4375
Epoch 53/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-108369.4141
Epoch 54/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-108232.9844
Epoch 55/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-108212.4609
Epoch 56/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-108143.3750
Epoch 57/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-108048.8125
Epoch 58/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-107985.3516
Epoch 59/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-107912.3281
Epoch 60/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-107911.6719
Epoch 61/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-107774.8516
Epoch 62/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-107762.3359
Epoch 63/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-107543.2031
Epoch 64/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-107554.0859
Epoch 65/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-107405.2344
Epoch 66/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-107310.9219
Epoch 67/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-107097.1094
Epoch 68/100
```

```
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-106943.1250
Epoch 69/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-106849.3359
Epoch 70/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-106749.1719
Epoch 71/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106671.0234
Epoch 72/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106616.2656
Epoch 73/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-106452.0000
Epoch 74/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106501.6094
Epoch 75/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106453.6094
Epoch 76/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-106488.0781
Epoch 77/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106450.8516
Epoch 78/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-106436.1016
Epoch 79/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-106397.1719
Epoch 80/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106372.0000
Epoch 81/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106331.8125
Epoch 82/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106363.6484
Epoch 83/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106337.6719
Epoch 84/100
```

```
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106394.8359
Epoch 85/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106287.2656
Epoch 86/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106310.2109
Epoch 87/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-106304.8906
Epoch 88/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106363.7969
Epoch 89/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-106282.3750
Epoch 90/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106417.8359
Epoch 91/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-106360.8125
Epoch 92/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106363.0625
Epoch 93/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106315.7734
Epoch 94/100
40/40 [==============================] - 3s 76ms/step - LogLikelihood:
-106320.2109
Epoch 95/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106422.5625
Epoch 96/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106348.4844
Epoch 97/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106337.0625
Epoch 98/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106452.2031
Epoch 99/100
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106331.9219
Epoch 100/100
```

```
40/40 [==============================] - 3s 75ms/step - LogLikelihood:
-106311.9219
```

```
<ipython-input-3-eab9452eee6d>:12: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  X = (X - mu[None,:]) / sigma[None,:]
```

```
Epoch 1/100
40/40 [==============================] - 5s 111ms/step - LogLikelihood:
-148818.2969
Epoch 2/100
40/40 [==============================] - 4s 110ms/step - LogLikelihood:
-146244.5312
Epoch 3/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-144354.7656
Epoch 4/100
40/40 [==============================] - 4s 110ms/step - LogLikelihood:
-142584.7188
Epoch 5/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-140917.2969
Epoch 6/100
40/40 [==============================] - 4s 110ms/step - LogLikelihood:
-139267.0469
Epoch 7/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-137659.8594
Epoch 8/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-136074.5938
Epoch 9/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-134504.2500
Epoch 10/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-132975.6250
Epoch 11/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-131499.9375
Epoch 12/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-130028.9766
Epoch 13/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-128608.0000
Epoch 14/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
```

```
-127343.2500
Epoch 15/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-126097.0781
Epoch 16/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-124844.3125
Epoch 17/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-123717.5469
Epoch 18/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-122684.6094
Epoch 19/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-121663.7109
Epoch 20/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-120754.2734
Epoch 21/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-119942.5000
Epoch 22/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-119127.1719
Epoch 23/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-118359.1094
Epoch 24/100
40/40 [==============================] - 4s 110ms/step - LogLikelihood:
-117620.6641
Epoch 25/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-116934.8750
Epoch 26/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-116261.5234
Epoch 27/100
40/40 [==============================] - 4s 110ms/step - LogLikelihood:
-115573.8125
Epoch 28/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-114940.8750
Epoch 29/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-114266.3281
Epoch 30/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
```

```
-113561.2656
Epoch 31/100
40/40 [==============================] - 4s 110ms/step - LogLikelihood:
-112732.3281
Epoch 32/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-111866.4609
Epoch 33/100
40/40 [==============================] - 4s 110ms/step - LogLikelihood:
-110979.6719
Epoch 34/100
40/40 [==============================] - 4s 110ms/step - LogLikelihood:
-110231.7656
Epoch 35/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-109471.5625
Epoch 36/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-108842.2891
Epoch 37/100
40/40 [==============================] - 5s 112ms/step - LogLikelihood:
-108263.2266
Epoch 38/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-107712.9219
Epoch 39/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-107238.1641
Epoch 40/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-106887.6016
Epoch 41/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-106407.3359
Epoch 42/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-105770.5859
Epoch 43/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-105194.8125
Epoch 44/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-104560.7969
Epoch 45/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-104005.1797
Epoch 46/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
```

```
-103602.5469
Epoch 47/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-103187.3281
Epoch 48/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-102729.7969
Epoch 49/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-102346.2656
Epoch 50/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-101899.7344
Epoch 51/100
40/40 [==============================] - 4s 110ms/step - LogLikelihood:
-101563.1250
Epoch 52/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-101181.5938
Epoch 53/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-100807.4531
Epoch 54/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-100524.8516
Epoch 55/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-100149.2109
Epoch 56/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-99652.8906
Epoch 57/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-99244.3359
Epoch 58/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-98777.0547
Epoch 59/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-98194.2422
Epoch 60/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-97788.3906
Epoch 61/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-97332.8125
Epoch 62/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
```

```
-97041.1953
Epoch 63/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-96826.2812
Epoch 64/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-96506.7031
Epoch 65/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-96230.6641
Epoch 66/100
40/40 [==============================] - 4s 110ms/step - LogLikelihood:
-95930.8906
Epoch 67/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-95657.4609
Epoch 68/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-95157.3438
Epoch 69/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-94513.4844
Epoch 70/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-93993.8984
Epoch 71/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-93511.6484
Epoch 72/100
40/40 [==============================] - 5s 112ms/step - LogLikelihood:
-93026.8281
Epoch 73/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-92598.6875
Epoch 74/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-92185.6250
Epoch 75/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-91860.1172
Epoch 76/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-91550.7109
Epoch 77/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-91460.8047
Epoch 78/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
```

```
-91338.3203
Epoch 79/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-91330.8047
Epoch 80/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-91241.3516
Epoch 81/100
40/40 [==============================] - 5s 114ms/step - LogLikelihood:
-91102.2734
Epoch 82/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-91058.4375
Epoch 83/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-91107.6406
Epoch 84/100
40/40 [==============================] - 4s 111ms/step - LogLikelihood:
-91085.1641
Epoch 85/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-91092.2734
Epoch 86/100
40/40 [==============================] - 5s 114ms/step - LogLikelihood:
-91059.4531
Epoch 87/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-91036.4375
Epoch 88/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-90970.8906
Epoch 89/100
40/40 [==============================] - 5s 113ms/step - LogLikelihood:
-90963.5078
Epoch 90/100
40/40 [==============================] - 5s 114ms/step - LogLikelihood:
-90987.4922
Epoch 91/100
40/40 [==============================] - 5s 113ms/step - LogLikelihood:
-90941.7734
Epoch 92/100
40/40 [==============================] - 5s 112ms/step - LogLikelihood:
-90981.6250
Epoch 93/100
40/40 [==============================] - 5s 114ms/step - LogLikelihood:
-90995.7734
Epoch 94/100
40/40 [==============================] - 5s 114ms/step - LogLikelihood:
```

```
-90961.2344
Epoch 95/100
40/40 [==============================] - 5s 113ms/step - LogLikelihood:
-91025.3828
Epoch 96/100
40/40 [==============================] - 4s 112ms/step - LogLikelihood:
-91004.3281
Epoch 97/100
40/40 [==============================] - 5s 117ms/step - LogLikelihood:
-90990.2891
Epoch 98/100
40/40 [==============================] - 5s 114ms/step - LogLikelihood:
-91007.0156
Epoch 99/100
40/40 [==============================] - 5s 113ms/step - LogLikelihood:
-90868.9922
Epoch 100/100
40/40 [==============================] - 5s 113ms/step - LogLikelihood:
-90960.6094
Epoch 1/100
40/40 [==============================] - 6s 148ms/step - LogLikelihood:
-149450.5156
Epoch 2/100
40/40 [==============================] - 6s 151ms/step - LogLikelihood:
-146996.0000
Epoch 3/100
40/40 [==============================] - 6s 149ms/step - LogLikelihood:
-144834.3125
Epoch 4/100
40/40 [==============================] - 6s 149ms/step - LogLikelihood:
-142904.5000
Epoch 5/100
40/40 [==============================] - 6s 153ms/step - LogLikelihood:
-141206.4844
Epoch 6/100
40/40 [==============================] - 6s 155ms/step - LogLikelihood:
-139589.0469
Epoch 7/100
40/40 [==============================] - 6s 154ms/step - LogLikelihood:
-138068.6250
Epoch 8/100
40/40 [==============================] - 6s 156ms/step - LogLikelihood:
-136546.2969
Epoch 9/100
40/40 [==============================] - 6s 153ms/step - LogLikelihood:
-134973.1562
Epoch 10/100
40/40 [==============================] - 6s 153ms/step - LogLikelihood:
```

```
-133575.0781
Epoch 11/100
40/40 [==============================] - 6s 150ms/step - LogLikelihood:
-132057.9531
Epoch 12/100
40/40 [==============================] - 6s 150ms/step - LogLikelihood:
-130586.4375
Epoch 13/100
40/40 [==============================] - 6s 149ms/step - LogLikelihood:
-129128.3594
Epoch 14/100
40/40 [==============================] - 6s 157ms/step - LogLikelihood:
-127655.1250
Epoch 15/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-126246.4219
Epoch 16/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-124916.8594
Epoch 17/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-123593.2500
Epoch 18/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-122272.7734
Epoch 19/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-121048.9766
Epoch 20/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-119851.5469
Epoch 21/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-118794.1641
Epoch 22/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-117787.5859
Epoch 23/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-116815.5000
Epoch 24/100
40/40 [==============================] - 7s 166ms/step - LogLikelihood:
-115991.3906
Epoch 25/100
40/40 [==============================] - 6s 160ms/step - LogLikelihood:
-115339.3516
Epoch 26/100
40/40 [==============================] - 7s 164ms/step - LogLikelihood:
```

```
-114628.5391
Epoch 27/100
40/40 [==============================] - 7s 163ms/step - LogLikelihood:
-114037.9375
Epoch 28/100
40/40 [==============================] - 7s 166ms/step - LogLikelihood:
-113391.2109
Epoch 29/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-112659.1250
Epoch 30/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-111958.8984
Epoch 31/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-111116.0234
Epoch 32/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-110121.3594
Epoch 33/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-109025.2500
Epoch 34/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-107722.0859
Epoch 35/100
40/40 [==============================] - 7s 166ms/step - LogLikelihood:
-106620.1094
Epoch 36/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-105835.0625
Epoch 37/100
40/40 [==============================] - 7s 169ms/step - LogLikelihood:
-105124.6406
Epoch 38/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-104626.0547
Epoch 39/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-104054.2344
Epoch 40/100
40/40 [==============================] - 7s 169ms/step - LogLikelihood:
-103250.7500
Epoch 41/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-102564.0859
Epoch 42/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
```

```
-101751.3438
Epoch 43/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-100974.0156
Epoch 44/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-100248.3281
Epoch 45/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-99611.4531
Epoch 46/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-99135.8047
Epoch 47/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-98597.5938
Epoch 48/100
40/40 [==============================] - 7s 166ms/step - LogLikelihood:
-98113.4531
Epoch 49/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-97603.5469
Epoch 50/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-97098.6875
Epoch 51/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-96543.5156
Epoch 52/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-95867.3516
Epoch 53/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-95376.6953
Epoch 54/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-95073.2578
Epoch 55/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-94789.7188
Epoch 56/100
40/40 [==============================] - 7s 169ms/step - LogLikelihood:
-94767.1172
Epoch 57/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-94617.8594
Epoch 58/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
```

```
-94350.0312
Epoch 59/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-94211.3203
Epoch 60/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-93995.7969
Epoch 61/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-93746.1406
Epoch 62/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-93342.1406
Epoch 63/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-92911.8906
Epoch 64/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-92260.5312
Epoch 65/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-91744.9844
Epoch 66/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-91314.4531
Epoch 67/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-90913.4922
Epoch 68/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-90520.4609
Epoch 69/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-90395.4297
Epoch 70/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-90174.3750
Epoch 71/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-90055.4062
Epoch 72/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-89988.8438
Epoch 73/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-89912.9844
Epoch 74/100
40/40 [==============================] - 7s 169ms/step - LogLikelihood:
```

```
-89835.5781
Epoch 75/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-89649.1406
Epoch 76/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-89465.6172
Epoch 77/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-89228.9531
Epoch 78/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-88985.6719
Epoch 79/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-88695.1719
Epoch 80/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-88377.6250
Epoch 81/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-88155.7109
Epoch 82/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-87804.7266
Epoch 83/100
40/40 [==============================] - 7s 168ms/step - LogLikelihood:
-87366.3672 3s - Log
Epoch 84/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-86880.2969
Epoch 85/100
40/40 [==============================] - 7s 171ms/step - LogLikelihood:
-86457.2031
Epoch 86/100
40/40 [==============================] - 7s 170ms/step - LogLikelihood:
-86185.3750
Epoch 87/100
40/40 [==============================] - 7s 170ms/step - LogLikelihood:
-86006.6484
Epoch 88/100
40/40 [==============================] - 7s 170ms/step - LogLikelihood:
-85842.2656
Epoch 89/100
40/40 [==============================] - 7s 167ms/step - LogLikelihood:
-85626.8438
Epoch 90/100
40/40 [==============================] - 6s 161ms/step - LogLikelihood:
```

```
-85508.6250
Epoch 91/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-85229.2656
Epoch 92/100
40/40 [==============================] - 7s 164ms/step - LogLikelihood:
-85119.3750
Epoch 93/100
40/40 [==============================] - 7s 164ms/step - LogLikelihood:
-84889.1875
Epoch 94/100
40/40 [==============================] - 7s 164ms/step - LogLikelihood:
-84756.1953
Epoch 95/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-84602.5625
Epoch 96/100
40/40 [==============================] - 7s 164ms/step - LogLikelihood:
-84494.6875
Epoch 97/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-84394.2031
Epoch 98/100
40/40 [==============================] - 7s 164ms/step - LogLikelihood:
-84283.0938
Epoch 99/100
40/40 [==============================] - 7s 164ms/step - LogLikelihood:
-84213.5938
Epoch 100/100
40/40 [==============================] - 7s 165ms/step - LogLikelihood:
-84073.3906
Epoch 1/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-149015.5312
Epoch 2/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-146185.3281
Epoch 3/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-143924.6719
Epoch 4/100
40/40 [==============================] - 8s 206ms/step - LogLikelihood:
-141966.6719
Epoch 5/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-140209.9688
Epoch 6/100
40/40 [==============================] - 8s 206ms/step - LogLikelihood:
```

```
-138592.2969
Epoch 7/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-136983.5469
Epoch 8/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-135459.5000
Epoch 9/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-133961.1562
Epoch 10/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-132453.1719
Epoch 11/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-130926.5234
Epoch 12/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-129445.4375
Epoch 13/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-127977.8125
Epoch 14/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-126576.6406
Epoch 15/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-125144.8125
Epoch 16/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-123852.3281
Epoch 17/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-122498.8516
Epoch 18/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-121245.7109
Epoch 19/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-120057.8906
Epoch 20/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-118894.4766
Epoch 21/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-117726.6484
Epoch 22/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
```

```
-116677.9609
Epoch 23/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-115687.5469
Epoch 24/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-114828.2109
Epoch 25/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-113865.2656
Epoch 26/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-112639.7344
Epoch 27/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-111154.6094
Epoch 28/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-109380.8594
Epoch 29/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-107598.9219
Epoch 30/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-106094.0625
Epoch 31/100
40/40 [==============================] - 8s 206ms/step - LogLikelihood:
-104883.6016
Epoch 32/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-103894.7656
Epoch 33/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-103130.0625
Epoch 34/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-102350.3672
Epoch 35/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-101900.4844
Epoch 36/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-101224.5781
Epoch 37/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-100760.7969
Epoch 38/100
40/40 [==============================] - 8s 207ms/step - LogLikelihood:
```

```
-100224.2578
Epoch 39/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-99733.6719
Epoch 40/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-99235.5156
Epoch 41/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-98783.2500
Epoch 42/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-98242.7109
Epoch 43/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-97644.0625
Epoch 44/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-96839.0547
Epoch 45/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-96097.2500
Epoch 46/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-95215.7656
Epoch 47/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-94567.0781
Epoch 48/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-93737.2188
Epoch 49/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-92969.7031
Epoch 50/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-92018.7031
Epoch 51/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-91177.6953
Epoch 52/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-90386.1719
Epoch 53/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-89932.1562
Epoch 54/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
```

```
-89576.0469
Epoch 55/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-89342.1406
Epoch 56/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-89185.0859
Epoch 57/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-88991.8516
Epoch 58/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-89073.9453
Epoch 59/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-88999.9219
Epoch 60/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88884.1562
Epoch 61/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88931.7109
Epoch 62/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88824.4297
Epoch 63/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-88915.9141
Epoch 64/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88797.4297
Epoch 65/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-88826.4922
Epoch 66/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88819.2188
Epoch 67/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88831.2812
Epoch 68/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88818.0625
Epoch 69/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-88821.1875
Epoch 70/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
```

```
-88792.8438
Epoch 71/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88773.8906
Epoch 72/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88692.5469
Epoch 73/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88709.4062
Epoch 74/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88634.1172
Epoch 75/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88602.8516
Epoch 76/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88543.9219
Epoch 77/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88460.8203
Epoch 78/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88313.4062
Epoch 79/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88156.1016
Epoch 80/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-88001.7656
Epoch 81/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-87863.2031
Epoch 82/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-87751.2969
Epoch 83/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-87634.7656
Epoch 84/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-87572.5391
Epoch 85/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-87360.3906
Epoch 86/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
```

```
-87128.9688
Epoch 87/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-86927.9375
Epoch 88/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-86778.7656
Epoch 89/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-86619.7656
Epoch 90/100
40/40 [==============================] - 8s 204ms/step - LogLikelihood:
-86377.8516
Epoch 91/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-86240.2656
Epoch 92/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-86126.8359
Epoch 93/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-86078.5234
Epoch 94/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-85971.1484
Epoch 95/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-85884.2031
Epoch 96/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-85833.1406
Epoch 97/100
40/40 [==============================] - 8s 206ms/step - LogLikelihood:
-85762.2344
Epoch 98/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-85799.4688
Epoch 99/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-85719.8672
Epoch 100/100
40/40 [==============================] - 8s 205ms/step - LogLikelihood:
-85717.0703
```

0.4 Interpretation

```
[4]: likelihoods = np.array(likelihoods)
     optimal_param = parameters[np.argmax(likelihoods)]
     optimal_encoder = encoders[np.argmax(likelihoods)]
     # print(np.array(optimal_param['mu']))
     # print(np.array(optimal_param['sigma']))
```

0.5 Evaluation

```
[5]: Y_test = np.array(test_data)[:,1:11]
     Y_hat = OOS_montecarlo(
         np.array(test_data)[:,0],
         train_data,
         optimal_encoder,
         optimal_param,
         100
     )
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0), HTML(value='')))

<ipython-input-2-5eb3a4405722>:20: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  X = (X - mu[None,:]) / sigma[None,:]

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))

<ipython-input-2-5eb3a4405722>:220: FutureWarning: Support for multi-dimensional
indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future
version.  Convert to a numpy array before indexing instead.
  z_prob = encoder(X[None,:])


HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))


HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))


HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))


HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

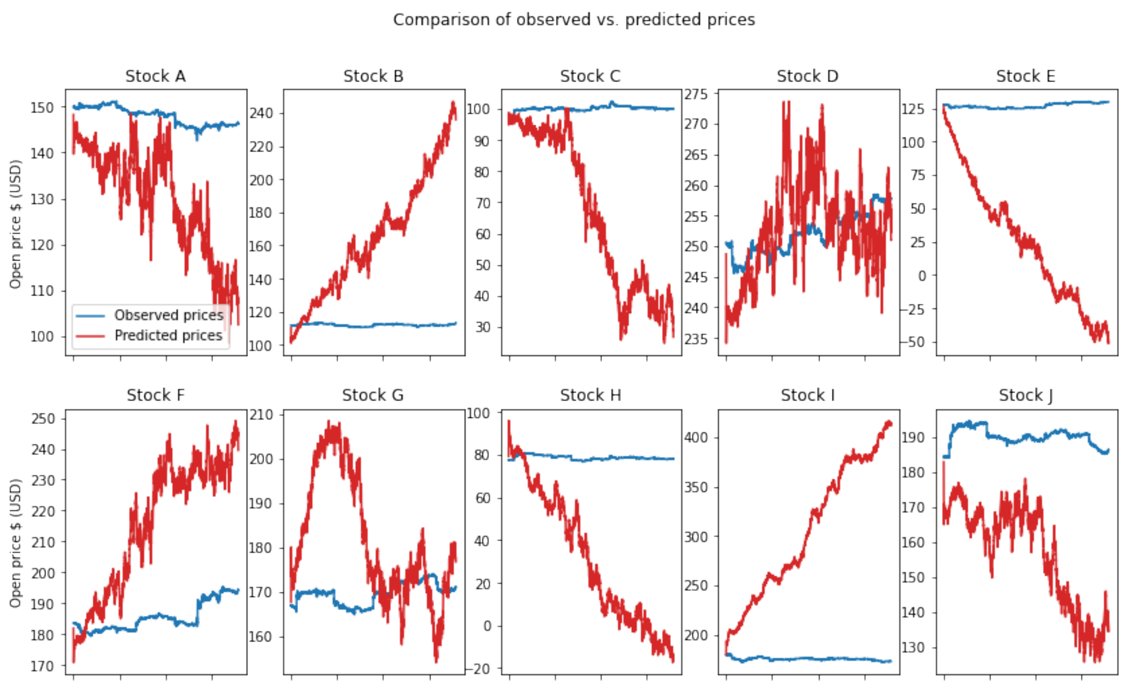
```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=35854.0),␣
 ↪HTML(value='')))
```

[9]:
```
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

headings = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
fig, axs = plt.subplots(2, 5, figsize=(14, 8), sharex=True)
for i in range(2):
    for j in range(5):
        index = 5 * i + j
        axs[i,j].plot(Y_test[:,index], color= 'tab:blue')
        axs[i,j].plot(Y_hat[:,index], color= 'tab:red')
        axs[i,j].set_title("Stock " + headings[index])
        axs[i,j].set(xticklabels=[])
blue_patch = Line2D([], [], color= 'tab:blue', label= 'Observed prices')
red_patch = Line2D([], [], color= 'tab:red', label= 'Predicted prices')
axs[0,0].legend(handles=[blue_patch, red_patch], loc= 3)
axs[0,0].set_ylabel("Open price $ (USD)")
axs[1,0].set_ylabel("Open price $ (USD)")
fig.suptitle("Comparison of observed vs. predicted prices")
```

[9]:
```
Text(0.5, 0.98, 'Comparison of observed vs. predicted prices')
```



```
[7]: MSE = np.mean((Y_test - Y_hat) ** 2)
     print(MSE)
```

```
4699.421195114543
```

```
[8]: # Attempt at multi-core processing
     #
     #from multiprocessing import Pool, cpu_count
     #
     #def f(i):
     #    return OOS_montecarlo(
     #        np.array(test_data)[:,0],
     #        train_data,
     #        encoder,
     #        param,
     #        2
     #    )
     #
     #store_res = []
     #
     #if __name__ == '__main__':
     #    with Pool(processes=20) as pool:         # start 20 worker processes
     #        store_res = pool.map(f, range(20))
```

```
[ ]:
```