Applications of the Kalman Filter: Financial Derivatives

Isaiah Baksh

1 Introduction

Kalman filters are a mathematical tool that has been around for years and has been a very advanced mathematical way of handling time series data, and is used for systems that require updating quite often. For instance, Kalman filtering is widely used for devices such as GPS systems in order to approximate where a person is going by predicting its next location given its current one. Another form of time series data that can be explored is equity data, which is widely available, is being used in this instance to see how well or how bad a Kalman filter can predict the movement of equities in financial markets.

2 The Kalman Filter

The Kalman Filter is a linear estimator that was first published by Rudolf E. Kalman in 1960. The goal of the Kalman filter is to predict the next step of a system from data or a function, while also incorporating noisy measurements and errors from the data, which are assumed to be Gaussian.

The discrete version of the Kalman filter is divided into 5 different equations. Which can then be separated into two different categories. A prediction step, which starts after we have some sort of initial estimate, that are governed by equation (1) and (2), and then three equations which are considered the update equations, which are equations (3) - (5). Equations (6) and (7) are necessary components as well because they model how the true state of the system we are looking at changes over time. and defines how we observe the system, and accounts for any measurement inaccuracies with the v_k term, for example sensor inaccuracies in the system.

The point of the state extrapolation equation, is to be able to estimate the next state using the previous knowledge from the last state. The covariance extrapolation is there to track how uncertain one is about the next state and is heavily dependent on the Q matrix which is known as the process noise covariance matrix, which is there to account for disturbances that may not be accounted for in just a linear model. The larger the values of Q the more uncertain we are in our model and believe that there is outside forces disturbing our system. A larger Q would also indicate more weight on our new measurements whereas a smaller Q would lead to more of a reliance on the model itself.

The Kalman gain is the most important part of the entire Kalman filter algorithm, as the entirety of the update equation rely on the gain equation. The idea behind this gain to account for how much do we trust our measurement compared to our prediction, so the way it works is if we find that we have a very large value for P, so in other words we are quite uncertain about our model we will have a larger K value meaning we will rely more on our measurements from a sensor. On the other hand, if we have a larger R, which is introduced in this Kalman Gain equation, that will result in us trusting the value for our prediction more and result in a smaller value for K. Our R matrix is quite simply how much we trust our measuring devices, so how much uncertainty do we suggest there is in a sensor.

The State update equation also introduces the measurements equation, simply put we are computing the residual of how far off the prediction and the measurement are, and then applying the Kalman gain to see how much we trust our measurement and apply it to our prediction, the closer K is to one the larger the correction and the more we trust our measurement, while at K closer to 0 we trust our prediction more. The covariance update though controls how much uncertainty we have in our measurement and the idea is that the more our value for K increases, the more we trust our measurement reducing our uncertainty, whereas if it is a much smaller value for K then we get our uncertainty remaining the same nearly.

Table 1: Equations for Filter

Model	Equation	
State Extrapolation	$x_{k+1}^{-} = \phi_{k+1}\hat{x}_{k+1}$	(1)
Covariance Extrapolation	$P_{k+1}^{-} = \phi_{k+1}P_{k}^{-}\phi_{k+1}^{T} + Q_k$	(2)
Kalman Gain	$K_{k+1} = H_{k+1}^{T}P_{k+1}^{-}[H_{k+1}^{T}P_{k+1}^{-}H_{k+1} + R_{k+1}]^{-1}$	(3)
State Update	$\hat{x}_{k+1} = \hat{x}_{k+1}^{-} + K_{k+1}(z_{k+1} - H_{k+1}\phi_{k+1}\hat{x}_k)$	(4)
Covariance Update	$P_{k+1} = P_{k+1}^{-} - K_{k+1}H_{k+1}P_{k+1}^{-}$	(5)
True States	$x_{k+1} = \phi_{k+1}x_k + w_k$	(6)
Measurements	$z_k = H_k x_k + v_k$	(7)

3 Discrete Kalman Filter Derivation

We want to fit discrete-time problems into the format of

$$x_{k+1} = \phi_{k+1}x_k + w_k \tag{3.0.1}$$

where x_k is the state, ϕ_k is the matrix that relates x_k to x_{k+1} and w_k is white noise we expect. We assume a random process is modeled by the form in (1). The measurement of

our process also has a linear relationship given by

$$z_k = H_k x_k + v_k \tag{3.0.2}$$

Where H_k is the measurement matrix which draws a connection between the state x_k and the measurement at z_k, with measurement error v_k. It is important to note that between w_k and v_k there is no correlation. Using our original model we are going to derive the first Kalman filter equation for state extrapolation. Since Kalman filtering estimates the average value of a state, if we take the expected value of equation (1)

3.1 State Extrapolation

$$E[x_{k+1}] = E[\phi_{k+1} x_k + w_k] \tag{3.1.1}$$

Applying Linearity of Expectation we get

$$E[x_{k+1}] = \phi_{k+1} E[x_k] + E[w_k] \tag{3.1.2}$$

Where $E[w_k]$ is process noise which the has zero mean, resulting in it equaling zero, which leaves

$$E[x_{k+1}] = \phi_{k+1} E[x_k] \rightarrow \hat{x}_{k+1}^{-} = \phi_{k+1} \hat{x}_k \tag{3.1.3}$$

So the best estimate or prediction step at time k is given by equation (5). This is the state extrapolation equation and is the first Kalman filter equation used in the prediction step.

3.2 State Update

Now onto deriving the state update equation. Since the Kalman filter is a linear estimator, it updates its current estimate using a weighted average of the prior estimate and new observation, therefore the equation we can start with is a minimum variance estimator given by

$$\hat{x}_{k+1} = K_{k+1}^{'} \hat{x}_k + K_{k+1} z_{k+1} \tag{3.2.1}$$

from there find the error between the estimated and the true state of the system

$$\hat{x}_{k+1} - x_{k+1} = K_{k+1}^{'} \hat{x}_k + K_{k+1} z_{k+1} - x_{k+1} \tag{3.2.2}$$

from there substitute the measurement of the process, equation (2)

$$\hat{x}_{k+1} - x_{k+1} = K_{k+1}^{'} \hat{x}_k + K_{k+1}(H_{k+1} x_{k+1} + v_{k+1}) - x_{k+1} \tag{3.2.3}$$

where we now expand the equation, and replace x_{k+1} with our model equation (1) and expand once again we will arrive at a very large expanded form of the equation, where we can take the expectation value of the entire equation.

$$E[\hat{x}_{k+1} - x_{k+1}] = E[K_{k+1}' \hat{x}_k + (K_{k+1} H_{k+1} \phi_{k+1} - \phi_{k+1}) x_k + (K_{k+1} H_{k+1} - I) w_k + K_{k+1} V_{k+1}] \tag{3.2.4}$$

Through the use of identities in the appendix and some simple rearranging, the equation is reduced simply to

$$K'_{k+1} = (I - K_{k+1}H_{k+1})\phi_{k+1} \tag{3.2.5}$$

This equation can now be substituted back into equation (6), the minimum variance estimator which results in the equation

$$\hat{x}_{k+1} = \phi_{k+1}\hat{x}_k + K_{k+1}(z_{k+1} - H_{k+1}\phi_{k+1}\hat{x}_k) \tag{3.2.6}$$

Reduced even further using equation (4.1.3), results in the final form of the state update equation used.

$$\hat{x}_{k+1} = \hat{x}^-_{k+1} + K_{k+1}(z_{k+1} - H_{k+1}\phi_{k+1}\hat{x}_k) \tag{3.2.7}$$

3.3 Covariance Extrapolation

Error estimation is defined as $e_{k+1} = x_{k+1} - \hat{x}^-_{k+1}$, more simply said as the difference between the true state and the estimated state. From this equation along with equation (4.1.3) and our true state equation given by equation (4.0.1), the covariance extrapolation can be derived.

$$\begin{aligned} e_{k+1} &= x_{k+1} - \hat{x}^-_{k+1} \\ &= \phi_{k+1}x_k + w_k - \phi_{k+1}\hat{x}_k \\ &= \phi_{k+1}(x_k - \hat{x}_k) + w_k \end{aligned}$$

Covariance extrapolation is given by the $P^-_{k+1} = E[e_{k+1}e^T_{k+1}]$ which can be rewritten as the following

$$P^-_{k+1} = E[(\phi_{k+1}(x_k - \hat{x}_k) + w_k)(\phi_{k+1}(x_k - \hat{x}_k) + w_k)^T] \tag{3.3.1}$$

$$= P^-_{k+1} = \phi_{k+1}\phi^T_{k+1}E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] + E[w_k w^T_k] \tag{3.3.2}$$

After taking Linearity of Expectation, simply results in (4.3.2), and recognizing that the expected value of the first term is just P^-_k and the expected value of the white noise is covariance of the white noise results in a final equation given by (4.3.3) for covariance extrapolation.

$$P^-_{k+1} = \phi_{k+1}P^-_k\phi^T_{k+1} + Q_k \tag{3.3.3}$$

3.4 Covariance Update

Estimation error previously defined above, can be used along with the state update equation to derive this equation. Specifically from state update we can write the following

$$e_{k+1} = x_{k+1} - \hat{x}^-_{k+1} + K_{k+1}(Hx_{k+1} + v_{k+1} - H\hat{x}^-_{k+1}) \tag{3.4.1}$$

which simplifies to

$$e_{k+1} = (I - K_{k+1}H)(x_{k+1} - \hat{x}^-_{k+1}) - K_{k+1}v_{k+1} \tag{3.4.2}$$

Using identities found in the appendices for the expected value of the covariance we know that we will get left with

$$P_{k+1} = (I - K_{k+1}H)P_{k+1}^{-}(I - K_{k+1}H)^T + K_{k+1}RK_{k+1}^T \tag{3.4.3}$$

This is a form that we can obtain for our covariance update although it can also be simplified down further using the Kalman Gain equation. Specifically, if we expand this equation, and apply equation (3.5.5), we obtain

$$P_{k+1} = P_{k+1}^{-} - K_{k+1}HK_{k+1}^{-} \tag{3.4.4}$$

This is the final simplified form of the covariance update that is used in our simulations.

3.5 Kalman Gain

Using the covariance update equation not in its simplest form, we are able to rearrange and solve for the Kalman Gain. Begin by applying the transpose and P_{k+1}^{-} through the first brackets to obtain.

$$P_{k+1} = (P_{k+1}^{-} - K_{k+1}H_{k+1}P_{k+1}^{-})(I - K_{k+1}^T H_{k+1}^T) + K_{k+1}R_{k+1}K_{k+1}^T \tag{3.5.1}$$

We are able to reduce this model down to the terms

$$P_{k+1} = P_{k+1}^{-} - K_{k+1}^T H_{k+1}^T P_{k+1}^{-} - K_{k+1}H_{k+1}P_{k+1}^{-} + K_{k+1}(H_{k+1}^T P_{k+1}^{-} H_{k+1} + R_{k+1})K_{k+1}^T \tag{3.5.2}$$

After this we are able to take the trace of the entirety of equation (4.5.2), because we are trying to minimize the variance which is the diagonal entries of the P_{k+1} matrix, in other words we are trying to minimize the trace(P_{k+1}). After combining terms and using and using the property of trace matrices, the equation simplifies to the following.

$$trace(P_{k+1}) = trace(P_{k+1}^{-}) - 2trace(K_{k+1}H_{k+1}P_{k+1}^{-}) + trace(K_{k+1}(H_{k+1}^T P_{k+1}^{-} H_{k+1} + R_{k+1})K_{k+1}^T) \tag{3.5.3}$$

Now we can apply some matrix identities(see appendices) that differentiate the matrix, and result in setting the equation equal to zero since we are trying to minimize the gain.

$$-2(H_{k+1}P_{k+1}^{-})^T + 2K_{k+1}(H_{k+1}^T P_{k+1}^{-} H_{k+1} + R_{k+1}) = 0 \tag{3.5.4}$$

After some simple rearranging of equation (3.5.4), we are left with our Kalman Gain.

$$K_{k+1} = H_{k+1}^T P_{k+1}^{-}[H_{k+1}^T P_{k+1}^{-} H_{k+1} + R_{k+1}]^{-1} \tag{3.5.5}$$

4 Toy Model (1D Motion)

The simulating of the constant velocity and acceleration models provide a simple framework, to understand the core concepts and implementation of the Kalman Filter when it comes

to real world applications. The constant velocity model is given by equation (5.0.1), and although it is quite small, it can illustrate how useful the Kalman Filter really is.

$$x_{k+1} = x_k + v_k \Delta t \tag{4.0.1}$$

The results of simulating the system, can be seen in the plots below. As expected the Kalman gain starts off quite large since it is a random initial guess given to the system, but it begins to level off and correct itself accordingly as expected, make each prediction closer to the true state and measurements of the actual system.

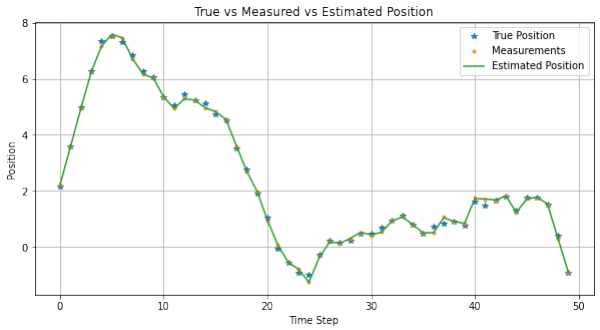
4.1 Constant Velocity

With the constant velocity model, from equation (5.0.1) the state extrapolation equation can be of the form

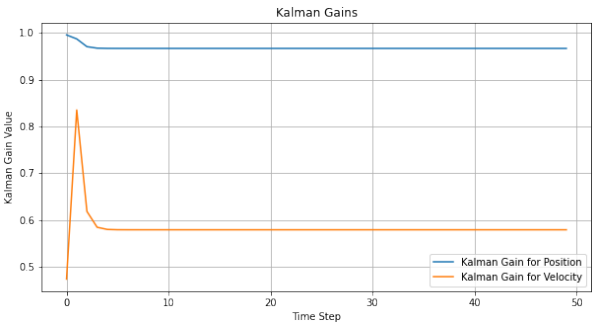
$$\begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix} \tag{4.1.1}$$

from this simple equation, we can implement it into code, and set our H matrix in our system to be [1 0], which would allow us to only observe the position of our system, in other words, you are watching a car drive by, and taking pictures of its position continuously, but have no measurement of its velocity, its velocity is simply inferred from its position at said time step. If the H matrix were to be [0 1] instead, the position would be inferred and the velocity would be captured by the sensor instead.

From this equation alone, it is possible to implement the Kalman filter algorithm, with all 5 equations and the measurements and true states of the system, and plot them in a position vs time graph, and measure the Kalman gain of the system over time.



(a) True vs Measured vs Estimated Position



(b) Kalman Gain for Position and Velocity

Figure 1: Comparison of estimated positions and Kalman gain evolution.

Above is the results of our simulation of the constant velocity model, just as stated in the first section the model for position has a gain near 1 which clearly means that we trust our measurements of the system more compared to our predictions, where the complete opposite is true for velocity, the model begins by giving more trust towards the measurements, but then begins to not trust it as much and gives a much more even weight to the predictions

as well. This actually makes quite a lot of sense, because the uncertainty given to the Q matrix, which is the amount of disturbances in the system is given to be 0.1 for both position and velocity while the R matrix which accounts for sensor measurements has a very low uncertainty of 0.01, which makes sense why we would have a result of a much higher gain for position and why we would trust our measurements much more as opposed to the predictions.

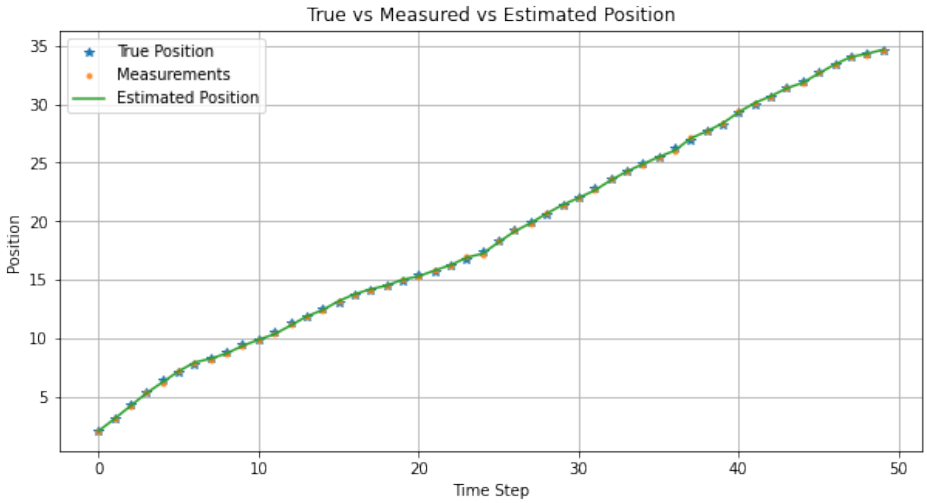


Figure 2: Q Matrix set to 0.01

Compared to figure 2, where the Q matrix is set to 0.01, we definitely have more trust in our model as opposed to our measurements, because the amount of disturbances is quite insignificant, so our predictions will be quite accurate, even in figure 2, it is visible that the position is much more linear, as opposed to figure 1 where there is clearly more disturbances in the path.

5 Forecasting Equities

Using this same idea of the toy model, we are going to apply to equities, specifically NVIDIA. NVIDIA is currently one of the largest companies in the world, and has its readily available in csv format to be used for this problem. Specifically, the thing that changes is instead of measuring position and velocity, we are measuring the price and its day to day change.

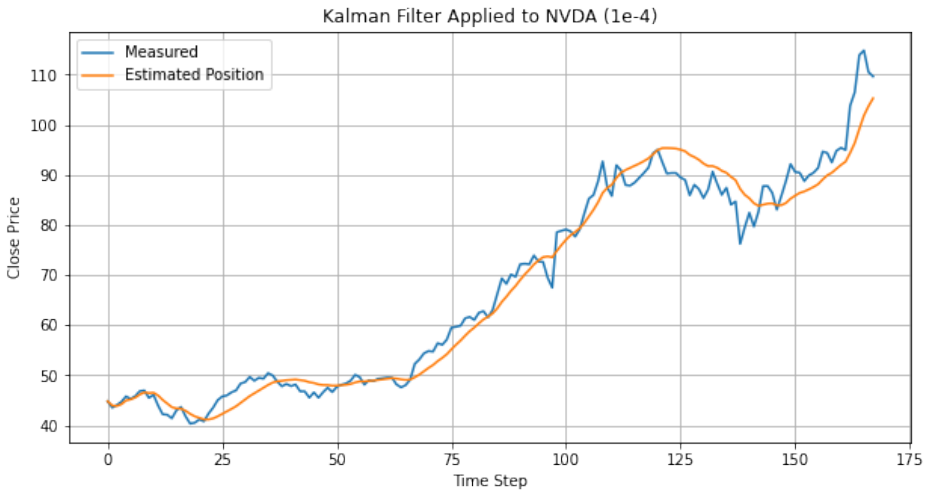


Figure 3: Very Low Q values

The estimated position clearly trails behind the actual measured price, which is the measured data from the csv file.

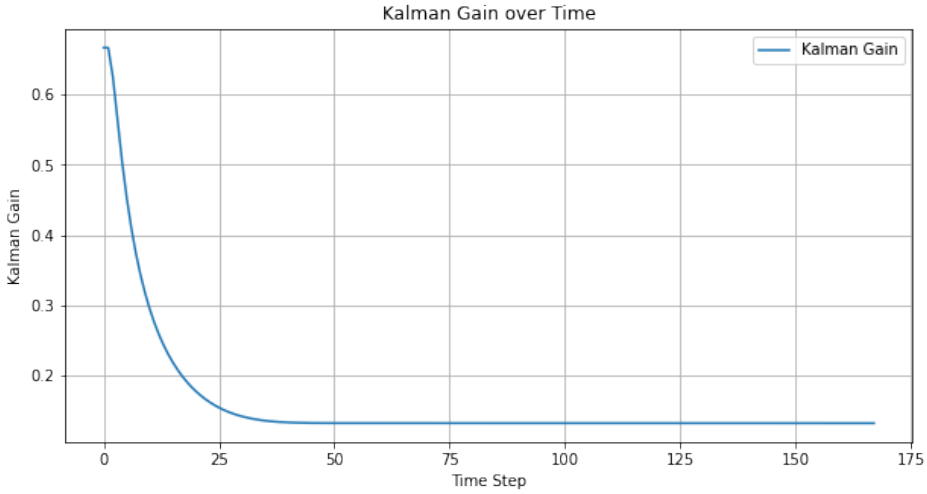


Figure 4: Kalman Gain for NVDA

Even when the Kalman Gain is plotted above, it starts off very high, and becomes very small very quickly, meaning that it begins to rely on the model, and predictions of thee future state as opposed to the true measurements of the state. From this point, we've created an extremely simplistic model for predictions of the stock price using the Kalman Filter, now we can attempt to forecast prices.

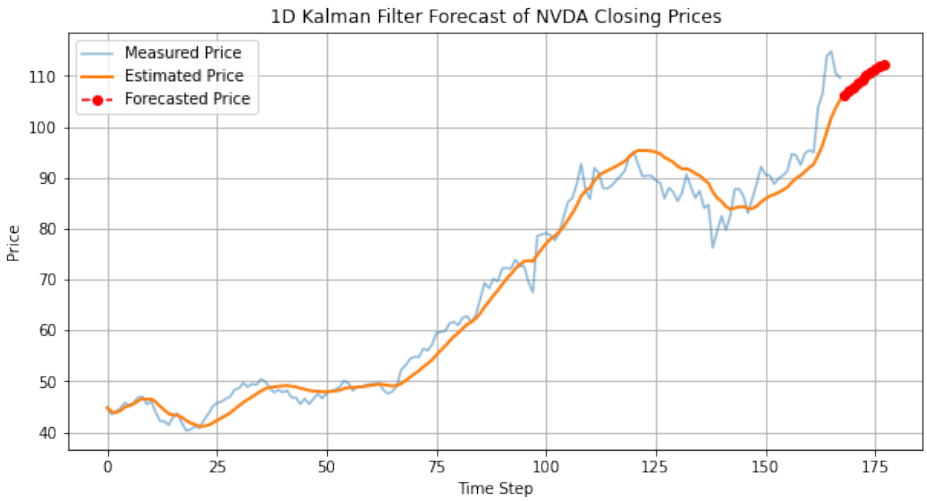


Figure 5: Forecast of Prices

The forecast of pricing here is quite simple, all that is introduced is the previous estimate used multiplied by the state transition matrix ϕ with a small amount of white noise introduced to the system, as seen in Figure 5, the next 10 days of prices are estimated, it is definitely a linear curve but not as sharp as the measured price and also seems to start to decline if you look veryy closely towards the last two days.

6 Conclusion

Kalman Filtering is definitely a very useful tool, for making estimates about the next time step of time series data, but there is still much more to be explored when implementing the algorithm, for instance the forecasting of data in NVIDIA stock price, is not that effective, and other more technical avenues should be sought out, such as introducing other data possibly or even, implementing higher level versions of the Kalman filter forecasting such as the extended Kalman filter or even the Unscented Kalman Filter. But as far as a simple model for Kalman filtering, it is very good at tracking previous data but as far as predicting the future price of something, more advanced methods must be applied.

7 Appendices

For all code snippets, please see github.com/isaiahbaksh28/undergraduateThesis

Table 2: Useful Identities

Identity	Explanation
$trace(A^T) = trace(A)$	Traces of matrices
$\frac{d(trace(AB)}{dA} = B^T$	Derivative of Matrices
$E[e_k e_k^T] = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T]$	Estimation Error Covariance (P_{k+1})
$E[w_k w_k^T] = Q_k$	Covariance of White Noise
$E[v_k v_k^T] = R_k$	Covariance Matrix of Measurement Error

References

[1] A. Becker, *Kalman Filter from the Ground Up*, KilmanFilter.NET (2023).

[2] R. Brown et Al, *Introduction to Random Signals and Applied Kalman Filtering*, John Wiley & Sons (1997).

[3] D. E. Caitlin, *Estimation, Control, and the Discrete Kalman Filter*, Springer-Verlag (1989).

[4] A. Gelb, *Applied Optimal Estimation*, M.I.T. Press (1986).

[5] A. Javaheri et Al, *Filtering in Finance*, RBC Capital Markets (2003).

[6] H. Qadri, *NVIDIA Corporation (NVDA) Stock — 2015 - 2024*, kaggle.com (2024).