Generative Equity Returns

Arya Gadage[1,2], Courtney Harvey[1] and Madeline Peckham[1, 3]

[1] College of Natural Sciences, University of Massachusetts Amherst
[2] College of Humanities & Fine Arts, University of Massachusetts Amherst
[3] Isenberg School of Management, University of Massachusetts Amherst

Abstract :

Inspired by the groundbreaking work of Ruslan Tepelyan and Achintya Gopal in their paper titled "Generative Machine Learning for Multivariate Equity Returns", our project aims to leverage generative modeling techniques to predict future equity returns. Specifically, our goal was to focus on using normalizing flows in conjunction with factor models to enhance our predictive capabilities.

The motivation behind this project lies in the inherent unpredictability of stock prices and market factors. Generative modeling provides a powerful tool for financial analysis by transforming complex data distributions into realistic samples. Building upon the theoretical foundations covered in our coursework and recent advancements in generative modeling, we employ normalizing flows to learn the underlying probability distribution of financial data.

Our approach involves combining insights from financial data analysis with normalizing flow techniques. We integrate factor models, such as the well-known Fama-French Three-Factor Model, to predict outcomes for a carefully selected blue-chip stock. By training our model using historical data, we created a robust framework for predicting future returns in the equity market.

The Fama-French Three-Factor Model, developed by Eugene Fama and Kenneth French, extends the traditional Capital Asset Pricing Model (CAPM) by incorporating additional factors, market risk, size, and value. These factors capture important dimensions of stock returns beyond the market portfolio. Our project leverages these factors, with our own selection of important factors, to enhance the accuracy of our predictions.

Furthermore, we explored the use of the normalizing flow technique to model a complex financial dataset. This model did not allow us to capture the complexities in the data and generate realistic samples. In our endeavor to implement a factor model, we encountered a significant setback when attempting to scale up the complexity by introducing additional factors. The normalizing flows model was unable to accommodate this level of intricacy. Recognizing the limitations of this approach, we turned towards using a deep neural network, which proved resilient in handling the increased dimensionality and complexity inherent in the expanded factor model.

In our study, we constructed a neural network model focused on a 3-factor framework incorporating treasury yield, Gross Domestic Product (GDP) rates, and inflation rates as predictors. With two hidden layers, the network effectively learned the complex relationships between these factors and equity returns. By training on historical data, the model was able to make accurate predictions of equity returns. This approach showcases the power of neural networks in extracting meaningful insights from multi-factorial datasets, ultimately enabling more informed decision-making in the realm of financial forecasting.

In summary, our project contributes to the field of generative finance by combining Generative modeling techniques, such as normalizing flows and neural networks, and established factor models to predict equity returns.

Index Terms: Generative modeling, Normalizing flows, Neural Networks, Factor models, Equity returns, Blue-chip stocks.

1 Introduction

The unpredictability of stock prices creates uncertainty for investors and analysts. In recent years there has been an interest in applying generative modeling techniques to address this unpredictability. Inspired by the work of Tepelyan and Gopal, our project seeks to explore the potential of generative machine learning in predicting equity returns.

There are multiple motivations for this project. With our aim being to establish an understanding of how to use predictive modeling to forecast the outcome for stocks. This motivation is driven by the inherently unpredictable nature of stocks and their characteristics. This is due to the many factors impacting their movements and the associated risks they pose. Particularly, stocks carry business specific risk, or risks which are specifically related to their business model and internal functions. Additionally, equities are also correlated with external factors, such as global politics and the overall economic of the areas in which they operate. However, because stocks are known to have an unlimited return but limited liability (you can only lose the price of the stock, nothing more), they are at the center of much of the study of finance.

An interesting solution to this complex problem is Generative modeling. By transforming complex data distributions into simpler, more realistic data, more financial analysis can be done. This is particularly useful for equity returns, since traditional analysis methods can fall short capturing the underlying distributions and patterns. Our project recognizes the significance of generative modeling as predictive method for stocks, offering a way to model and forecast outcomes that would otherwise be extremely challenging to predict through traditional methods.

Within the academic literature, we will use the work of researchers who have explored various methods, including factor models that incorporate combined probability, Gaussian methods, normalizing flows, and neural networks to predict future outcomes in the financial markets. Factor models have emerged as pivotal tools for comprehending and forecasting stock returns. These models, rooted in statistical analysis and theory, aim to find and help analyze the underlying factors of market dynamics by finding key factors that influence asset prices. At the heart of factor models lies their ability to find the sources of variation in returns. Factors such as size, value, and market risks have been widely acknowl-

edged as fundamental determinants of stock returns.
Additionally, integrating normalizing flows and deep neural networks techniques into factor models represents a significant advancement in predictive modeling for financial markets. Normalizing flows in combination with deep neural networks has the ability to model complex probability distributions and generate realistic samples that can capture the dynamics of stock returns. The scalability of these techniques allows for the analysis of large-scale financial datasets, and therefore facilitates the development of predictive models that can accommodate the growing volume and complexity of financial data.
In conclusion, our project seeks to harness generative machine learning techniques to address the unpredictability of stock prices, inspired by the research of Tepelyan and Gopal. Through the integration of factor models, probability, Gaussian methods, neural networks, and normalizing flows, we aim to develop a predictive framework for forecasting equity returns. By leveraging the flexibility and scalability of these methods, our project aims to enhance our understanding of predictive modeling in finance and provide insights for investors and analysts.

2 Background

With this project, we aim to predict future returns of The Coca-Cola Company (KO or the Company). The Coca-Cola Compayy is considered to be a "Blue Chip Stock". Blue Chip Stocks are stocks issued by large, well-established corporations which have shown to have dependable earnings. These corporations regularly pay dividends to investors and have been doing so for many years. In addition to that, because these companies have a long history, they would be less prone to business risk based on their maturity, and more correlated to the data selected for the factor model. The Coca-Cola Company is a component of market indexes or averages, such as the Dow Jones Industrial Average, the Standard & Poor's (S& P) 500, and the Nasdaq-100 in the United States *(Chen, 2024.)*

There are essentially two ways to predict stock price - Fundamental Analysis and Technical Analysis. In this project, we have utilised Technical analysis methods along with normalising flows in order to predict returns. We then used the factor model and a deep neural network to produce a better fit model. Daily stock price makes an essential component of the robust dataset needed for this analysis. After collecting the daily prices, daily, monthly and annualised returns need to be calculated.

$$R_{\text{daily}} = \frac{P_{\text{today}} - P_{\text{yesterday}}}{P_{\text{yesterday}}}$$

P_{today} is the closing price of the stock today and $P_{\text{yesterday}}$ is the closing price of the stock yesterday.

$$R_{\text{monthly}} = \frac{P_{\text{end of month}} - P_{\text{start of month}}}{P_{\text{start of month}}}$$

$P_{\text{end of the month}}$ is the closing price of the stock at the end of the month and $P_{\text{start of the month}}$ is the closing price of the stock at the start of the month.

$$R_{\text{annualized}} = \left(1 + R_{\text{period}}\right)^{n} - 1$$

R_{period} is the return of the period and n is number of periods per year (for daily returns n would be the number of trading days in the year i.e, 252 and for monthly returns n would be 12.

Three primary components are employed by the FAMA-French factor model to explain stock market results. 1) Market risk; 2) Small-cap firms' higher earnings over large-cap companies; and 3) High book-to-market value companies' better performance over low book-to-market value companies. It depends on the observation that small-cap and high-value companies consistently outperform the market as a whole.

The expected rate of return on a stock or portfolio is given by the following equation:

$$r = r_f + \beta_1(r_m - r_f) + \beta_2(\text{SMB}) + \beta_3(\text{HML}) + \epsilon$$

Where:

- r = Expected rate of return
- r_f = Risk-free rate
- β = Factor's coefficient (sensitivity)
- $(r_m - r_f)$ = Market risk premium
- SMB (Small Minus Big) = Historic excess returns of small-cap companies over large-cap companies
- HML (High Minus Low) = Historic excess returns of value stocks (high book-to-price ratio) over growth stocks (low book-to-price ratio)

Factor models are common for stock prediction and can be changed to best fit to company. In the case of KO, as the business is more mature and has a large market capitalization, we selected our factors to be GDP, inflation rates, and the 10 year treasury yield. Larger companies typically do not see as sudden growth in earnings, so we chose to focus on publicly available data that ground the return expected from a market as a shareholder. Shareholders, the people who have a share of ownership in the companies stock, have a baseline expected rate of return. The intricacies of this are discussed further in section 4.1.

3 Methods

3.1 Normalizing Flows Technique

A common Generative model technique for complex distributions is normalizing flows. Normalizing flows maps between two latent-variable models, X and Z. This mapping is given by $f_\theta : \mathbb{R}^d \to \mathbb{R}^d$, where d is the dimension. This mapping results in $X = f_\theta(Z)$ and $Z = f_\theta^{-1}(X)$, which are both differentiable and invertible (Katsoulakis, 2024, Intro to Normalizing Flows). The steps to analyzing a distribution using normalizing flows are the following:

1. Defining the base distribution for $z_0 = x$.
2. Defining and applying a series of transformations to get to $x = z_K$, where K is the total number of transformations applied.
3. Compute the likelihood by change of variables.
4. Optimize using maximum likelihood over the dataset.
5. Once the model is trained using previous steps it can be used for efficient sampling and density estimations.

3.1.1 Defining the base distribution. We begin by defining our base distribution, denoted as $p(z_0)$. In most cases, we choose a simple distribution, such as a multivariate Gaussian. The variable

z_0 represents the latent space, and we want to transform it into the data space (x). The initial mapping is straightforward: $z_0 = x$, which means we assume that the latent representation is directly equal to the observed data.

3.1.2 Defining and applying a series of transformations. The next step involves applying a series of invertible transformations, also known as bijective functions, to modify the base distribution, enhancing its expressiveness. We initiate these transformations with the initial base distribution z_0 and continuing for all K steps:

$$z_1 = f_1(z_0),\ z_2 = f_2(z_1),...,\ z_K = f_K(z_{K-1})$$

Each function f_i is chosen to be invertible, ensuring that the data's likelihood under the transformed distribution can be accurately computed (Katsoulakis, 2024, Intro to Normalizing Flows). Notably, if the data is discrete, each transformation is discrete as well. For continuous data, unlike discrete transformations used with discrete data, continuous normalizing flows employ continuous transformations that act on the entire distribution rather than individual data points. This distinction is crucial in capturing the complex dependencies present in continuous data. The image below illustrates how the function evolves as it learns the data and undergoes a series of transformations.

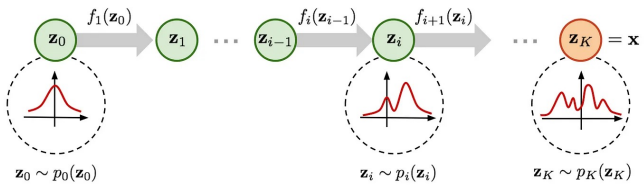

Figure 1. Illustration of a normalizing flow model, transforming a simple distribution to a complex one step by step. Adapted from Lil Log article by Lilian Weng, "Flow-based Deep Generative Models" **Lilian Weng**.

3.1.3 Compute the likelihood by change of variables. After we transform a simple distribution (usually a standard Gaussian) into a more complex distribution by applying a series of invertible transformations, we compute the likelihood. Given a data point x, we want to compute its likelihood under the target distribution. This is used to evaluate how effective the model is. We start with the likelihood of the transformed variable z and then apply the change of variables formula. Let $z = f(x)$ be the transformed variable, where f is the invertible transformation. The likelihood of (z) is given by the base distribution:

$$p(z) = \mathcal{N}(z; 0, I)$$

Here, $\mathcal{N}$ represents the Gaussian distribution with mean 0 and identity covariance matrix.
To compute the likelihood of the original data point x, we use the change of variables formula:

$$p(x) = p(z)\left|\det\left(\frac{\partial f}{\partial x}\right)\right|^{-1}$$

The term

$$\left|\det\left(\frac{\partial f}{\partial x}\right)\right|,$$

represents the determinant of the Jacobian matrix of the transformation f with respect to x (Katsoulakis, 2024, Intro to Normalizing Flows). Taking the determinant gives us a single value that represents the overall change in volume induced by the transformation. The determinant should be close to 1. If it is much greater than 1 then areas are being stretched out, resulting in larger volumes in the transformed space compared to the original (Papamakarios et al., 2021, March). On the other hand, if the determinant is less than 1, it implies areas are being compressed and the volume is much less than the original (Papamakarios et al., 2021, March). The change of volume affects the probability density as well. If the determinant were the only factor, the probability density might become distorted by over condensing or spreading out leading to invalid probabilities. Thus, we now take the inverse as a corrective measure. The inverse counteracts the stretching or condensing and ensures that the integral of the transformed density over the entire space remains equal to 1, thereby maintaining the integrity of the probability density (Papamakarios et al., 2021, March). This step essentially neutralizes the network.

3.1.4 Optimizing using maximum likelihood over the dataset To find the best parameters for a model based on the observed data, we optimize the likelihood using Maximum Likelihood Estimation (MLE) and Log-Likelihood. Given a dataset: $\mathcal{D} = \{x_1, x_2, \ldots, x_n\}$, where each x_i represents an observed data point, we want to find the model parameters θ that maximize the likelihood of the data.
The likelihood function $L(\theta)$ measures how well the model explains the observed data. It is defined as the joint probability of the data points given the parameters:

$$L(\theta) = p(x_1, x_2, \ldots, x_n|\theta),$$

The goal is to find θ that maximizes $L(\theta)$. We do this by log-likelihood to simplify computations. The log-likelihood function is:

$$\log L(\theta) = \textstyle\sum_{i=1}^{n} \log p(x_i|\theta)$$

Maximizing this log-likelihood function is equivalent to maximizing the likelihood function. In this optimization process, we may encounter complex transformations and computations, such as the expression:

$$\max_{\theta} \log(p_\theta(D)) = \sum_{x \in D} \log p_Z(f_\theta^{-1}(x)) + \log\left|\frac{\partial f_\theta^{-1}(x)}{\partial x}\right|$$

This expression encapsulates the optimization of the log-likelihood function over the dataset D, where $p_\theta(D)$ represents the likelihood of the dataset under the model parameterized by θ, f_θ^{-1} denotes the inverse transformation function, and p_Z is the probability density of the base distribution (Katsoulakis, 2024, Intro to Normalizing Flows). The determinant of the Jacobian matrix $|\frac{\partial f_\theta^{-1}(x)}{\partial x}|$, as dicussed earlier, accounts for the stretching or compression of volume in the transformation.

3.1.5 Using the model to sample and estimate. With the estimated parameters now in hand, we can perform inference tasks such as hypothesis testing, assess the effectiveness of your model to the data, or utilize the trained model to make predictions on new, unseen data.

3.2 Neural Networks

3.2.1 Defining Neurons A neural network is a highly parametrized model, inspired by the architecture of the human brain; learns from "experience" (Katsoulakis, 2024).

A "neuron" is a computational unit that takes as input x_1, x_2, x_3 (and a +1 intercept term), and outputs $h_{w,b}(x) = f(W^T x) = f\left(\sum_{i=1}^{3} W_i x_i + b\right)$, where $f : \mathbb{R} \to \mathbb{R}$ is called the activation function.

$$f(z) = \frac{1}{1 + \exp(-z)}$$

3.2.2 What is a neural network? Neural networks are computational frameworks designed to emulate the intricate operations of the human brain. These networks comprise interconnected units or neurons that process information and learn from data. Unlike systems with pre-programmed rules, neural networks develop their understanding by identifying features from data.

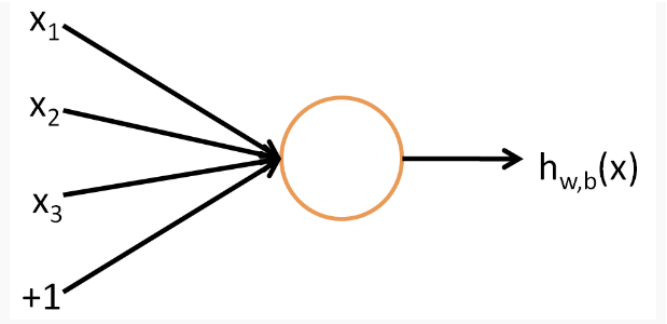

Figure 2. The simplest possible neural network, one which comprises of a single "neuron." **Stanford University**.

The components of a neural network include neurons, connections, weights, biases, propagation functions, and a learning rule. Neurons process inputs based on thresholds and activation functions, while connections use weights and biases to regulate the flow of information. The learning process involves three stages: computing the input, generating the output, and iteratively refining the process to improve the network's ability to perform various tasks.

The fundamental operations in neural networks are known as forward propagation and backpropagation:

- **Forward Propagation:**

1. **Input Layer:** Nodes in the input layer represent each feature and receive the corresponding data.
2. **Weights and Connections:** Weights signify the strength of connections, which are adjusted during training.
3. **Hidden Layers:** Neurons in the hidden layers process inputs by multiplying them by weights, summing them, and then applying an activation function to introduce nonlinearity. This allows the network to identify complex patterns.
4. **Output:** The process is repeated through the layers until the output layer produces the final result.

- **Backward Propagation:**

1. **Loss Calculation:** The output of the network is compared with actual target values using a loss function, such as Mean Squared Error (MSE) for regression tasks. The MSE is computed as

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

2. **Gradient Descent:** The network employs gradient descent to minimize the loss, adjusting weights based on the gradient of the loss with respect to each weight.
3. **Adjusting Weights:** This process of backpropagation involves adjusting weights across the network in a reverse direction iteratively.
4. **Training:** The entire cycle of forward propagation, loss calculation, and backpropagation is repeated with different data samples, allowing the network to learn and adapt to patterns in the data.
5. **Activation Functions:** Functions like the rectified linear unit (ReLU) or sigmoid determine whether a neuron should activate, contributing to the model's ability to handle nonlinear data.

3.2.3 Artificial Neural Networks Architecture The network architecture consists of three primary layers: the **input layer**, one or more **hidden layers**, and the **output layer**.

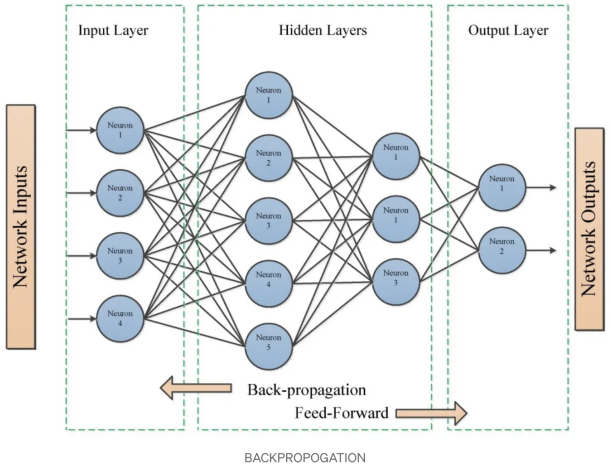

Figure 3. Illustration of an Artificial Neural Network. **Adbolrasol; et al 2020**.

The hidden layer or layers filters and extracts key patterns from the inputs, forwarding only the most pertinent information to subsequent layers for further processing. This selective extraction enhances the network's efficiency by focusing on essential data and omitting redundant inputs.

Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases. (Ghorakavi, 2024)
The activation function plays a crucial role for a couple of reasons: Firstly, it introduces non-linearity to the model, which is essential for capturing complex relationships between inputs. Secondly, it transforms the input into a format that is more suitable for further analysis and output generation, essentially shaping the data into a more actionable form.

- **Types of Activation functions**

1. **Binary Step function:** Depends on a threshold value that decides whether a neuron should be activated or not. The binary step function is defined as:

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

   The obvious drawback is that it cannot represent multi-value outputs.
2. **Linear Activation function:** It is called "no activation," or "identity function" the activation is proportional to the input. The identity function is defined as:

$$f(x) = x$$

   The last layer of the function will be a linear function of the first layer. Therefore, the derivative of the function is a constant.
3. **Non-linear Activation function:**They allow backpropagation because now the derivative function would be related to the input

Popular Non-linear Activation functions:

- **Sigmoid Function:** The function outputs values between 0 and 1, useful for binary classification problems. It is defined by the formula:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

  However, it is not often used in deep networks due to problems like vanishing gradients.
- **Hyperbolic Tangent Function (tanh)** Outputs values between -1 and 1. It is similar to the sigmoid but can provide stronger gradients since it centers the output, which helps accelerate the convergence. The formula is:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- **Rectified Linear Unit (ReLU)** It has become the default activation function for many types of neural networks because it allows models to converge faster and perform better. ReLU is defined as:

$$\text{ReLU}(x) = \max(0, x)$$

  It does not activate all the neurons at the same time, which means the network can easily benefit from sparsity.
- **Leaky ReLU:** It is a variant of ReLU that allows a small, positive gradient when the unit is not active. The formula is:

$$\text{Leaky ReLU}(x) = \max(0.01x, x)$$

- **Maxout** This activation is a generalization of ReLU and Leaky ReLU. It takes the maximum of a set of linear functions, which makes it more flexible and capable of learning non-linear decision boundaries. The Maxout function is represented as:

$$\text{Maxout}(x) = \max(w_1^T x + b_1, w_2^T x + b_2)$$

  where w_1, w_2 are weights and b_1, b_2 are biases.
- **Exponential Linear Unit (ELU)** It combines the benefits of ReLU and the ability to maintain mean activations closer to zero, which speeds up learning. The negative inputs are mapped to a function which ensures a smoother output than ReLU. It is defined as:

$$\text{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^{x} - 1) & \text{if } x < 0 \end{cases}$$

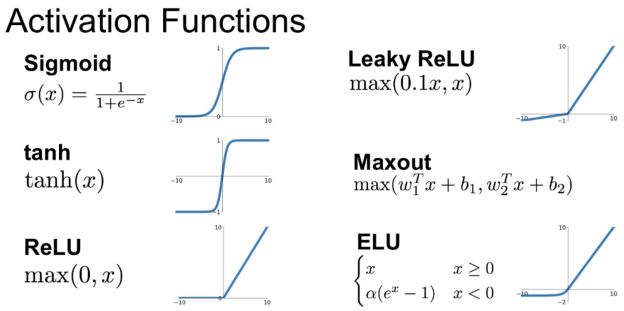

Figure 4. Activation functions **Singh** ; **2024**.

3.2.4 Deep Neural Networks Machine learning as a field is dedicated to automating the application of statistical models, through algorithms to enhance prediction accuracy. In ML, a model learns by adjusting its internal weights to minimize errors in its predictions, thus becoming more accurate over time.

The most basic form of neural network is the single layer perceptron model. This model has a single dense layer. A neural network with 2 or more layers qualifies as a "deep neural network".

A key component of building a successful model is figuring optimal weights that minimize prediction error. The backpropogation method takes care of that. The optimization approach uses the “gradient descent” technique to quantify prediction errors.

Deep neural networks take this concept further by layering multiple hidden layers on top of each other. This architecture builds on the premise that if a single hidden layer can effectively discern input significance and relationships for better predictions, multiple layers could significantly amplify these benefits, leading to more refined and accurate models.

4 Results

Our final code aims to use both a factor model and a neural network to predict equity returns of Coca Cola (KO).

4.1 Selecting the Data

We began by selecting a period of time to evaluate KO. We selected the stock performance from January 1, 1995 to December 31, 2004 (Yahoo Finance). This was an interesting period in time in that it includes the collapse of the technology bubble and the economic effects of the 9/11 terrorist attack. Following 9/11, there were immediate impacts on the economy, such as the immediate impact of the 9/11 attack was to reduce real GDP growth in 2001 by 0.5 percent and an increase to the unemployment rate by 0.11 percent (Roberts, 2009). Additionally, for better results with our data, we chose this time frame in order to have a holistic idea of the economy, with the future completely priced in. Meaning, during this time we have all necessary information to understand the performance in this period of time, as well as access to all economic data needed to support the model.

For the factors, we selected GDP rate, inflation, treasury yield. For the GDP rate, we used the annual real GDP, as this data is released in the US on an annualized basis. Real GDP is most frequently referenced to categorize economic health, as this is adjusted for inflation and determines the rate in which the US economy is growing each year (Hall). For inflation, we used the the Consumer Pricing Index (CPI). As described by the US Bureau of Labor Statistics, CPI "is a measure of the average change overtime in the prices paid by urban consumers for a market basket of consumer goods and services." Finally, for treasury yield we referred to the 10-year treasury yield. Often referred to as the risk free rate, this rate is an essentially market for knowing what shareholders expect for value generation of a company. Investing in the US treasury is viewed as "risk-less" granted it is unlikely for the government to collapse as they have the capability of taxing to fund debts. So, an investor in a stock would expect a return that of treasury, which for stocks as they have no maturity rates, is often compared to the 10-year treasury yield.

4.2 Attempting a Normalizing Flow Model

First, to begin our process we tried to create a model for KO that predicted its stock performance given only the price data. In this process, we came to realize that the normalizing flow method, was both difficult to integrate different factors into, and unlikely to be able to support type of model we needed without also incorporating additional modeling architecture.

As seen in Figure 5., there was instability of the normalizing flow model, even after adjusting dimensions and catering the model to the data. Due to both the complexity of the data and also the NF model in general, it was difficult to fit the data, whether over or under-fitting the KO stock performance. Also, for modeling a stock like KO, throughout volatile economic times, this produces natural outliers in the data. Here, the noise appears to also degrade the model's. For these reasons, we decided to seek alternatives to the project methods.

4.3 Creating a Deep Neural Network Model

Following our lack of success with the normalizing flow model, we decided to use the factor model with a deep neural network. Due to a large access to data for performance, a DNN is a reasonable and useful model for predicting stock performance. In this instance when we also wanted in intertwine other factors into our model, three factors, the DNN was a good architecture to apply.

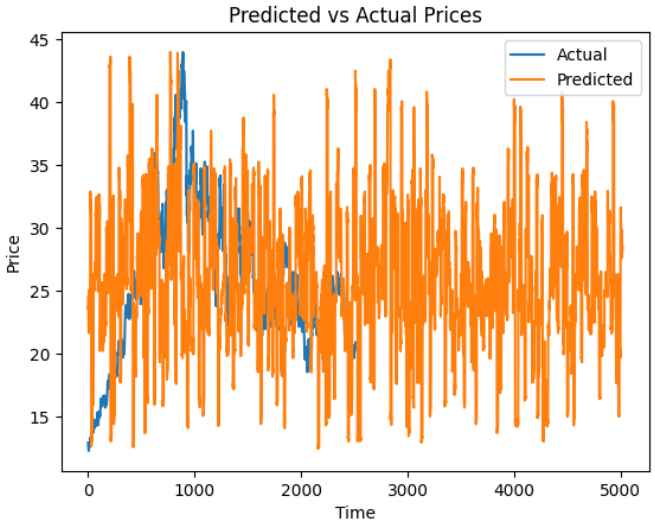

Figure 5. Outputs of the NF Model, with Time referencing days of stock performance metrics.

Particularly in this case, where there is a non-linear relationship between economic factors and stock performance; DNN can find these dependencies between variables to arrive at a stock prediction.

4.4 Merging the Time Frames

All three of these files were open and read in the code. Then to be able to merge the data based on dates, we first had to spilt up the dates, which was completed using string splits. The metrics used came in at different times frames, as KO and Treasury Yield were performed daily, subject to market closes. Likewise, GDP is annual and CPI is released monthly. Then using pandas' merge() function we were successfully able to merge the data based on the 'Year' and 'Mo nth' columns. For example, the function pd.merge(gdp_data, inflation_data, on='Year', how='left') merges the GDP data and inflation data based on the 'Year' column, retaining all rows from the left data-frame (GDP data) and matching rows from the right data-frame (inflation data). This process is then repeated for merging yield data and stock data. Now with the merged data we were able to drop any missing values using the dropna() function. It is also important to note that the target variable is selected, which in this case is the 'Close' price of KO stock shifted by one day to predict the next available day's price. This is done by the function target = factor_data['Close'].shift(-1).

4.5 Splitting the Data

Now that the data is cleaned and organized it is split into training and testing sets, with 80% of the data used for training and 20% for testing. This is done by "X_train, X_test, y_train, y_test = train_test_split(factor_data, target, test_size=0.2, random_state=42)", where

- X_train and y_train are used to train the the neural network.
- X_test and y_test are used to evaluate the performance of the trained model on unseen data.

4.6 *Standardizing the Input Data*

Now using standardization, or z-score normalization, we re-scaled the features of a dataset, namely X_train and X_test to have properties similar to a standard normal distribution. The first step in standardization is to calculate the mean value of each value across the entire dataset. For each value, the mean value is subtracted from all data points, resulting in a dataset where the mean of each value is centered around zero. After centering the mean, we scale to have a standard deviation of 1. This is achieved by dividing each value by its standard deviation, which is a measure of the spread or variability. Scaling the values in this way ensures that they have similar ranges and magnitudes, preventing values with larger scales from dominating the models learning process. By standardizing the data, all values will have a mean of approximately 0 and a standard deviation of 1. This will not change the shape of the data but rather just rescale it. In our code we use the function StandardScaler() to rescale our dataset.

4.7 *Building the Factor Model*

This is where a neural network is added by using the function "Model=Sequential()". Sequential from Keras is a linear stack of layers, where each layer has exactly one input tensor and one output tensor. Each neuron in one layer is connected to the ones in the previous creating a connected network. In this model there are two hidden layers followed by a Leaky ReLU activation layer. The final output layer uses a ReLU activation function to predict the next day's stock price.

4.8 *Training the Factor Model*

Before the model is trained the optimizer "Adam" is used to update the model's weights during training to minimize the loss function. This optimization is done with the function "Model.compile(optimizer='adam', loss='mse')". The second half of this function is naming the loss function as MSE or Mean Squared Error. The mean squared error loss function minimizes the squared differences between predicted and actual values. MSE is calculated as the average of the squared differences between predicted and actual values over all samples in the dataset.

Now that these steps are complete the model is ready to be trained. The is done on the line "Model.fit(X_train_scaled, y_train, epochs=50, batch_size=32, validation_data=(X_test_scaled, y_test))". During the training, the model is updating its weights iteratively to minimize the loss function. The model is trained for 50 epochs, meaning it will iterate over the entire training data 50 times. Additionally, a batch size of 32 is used, meaning the model updates its weights after processing 32 samples. The validation data (X_test_scaled and y_test) are provided during the iterations to evaluate the model's performance on unseen data after each epoch.

4.9 *Predicting Using the Model*

After training the neural network model on the training data, the next step is to use the trained model to make predictions on the test dataset. We use the predict() method to predict the next days stock based on the trained data. Then the predicted and actual values are combined into a DataFrame to directly compare. This comparison is allows us to assess the performance and accuracy of the model. The final step of the code saves the data to a CSV file to allow for further analysis of the data. The results, as shown in Figure 6., were much better at capturing the data. This supports our initial realization that a deep neural network would be better for developing a model with data complexities.

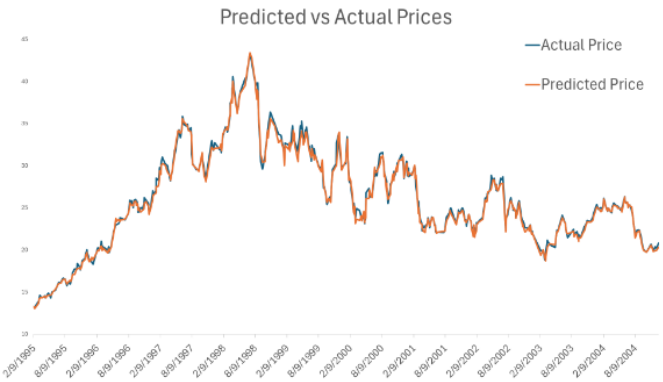

Figure 6. Outputs of the DNN Model

5 Discussion

Throughout creating a predictive model for stock performance, we noticed that there was nuance in integrating the economic data and other metrics given different time lines and merging. In addition to that, our factors, at only three, only represent a small amount of data available and metrics commonly used to support rationale on why a particular stock is performing in a certain fashion. With more time and data accessibility, we do see potential in further refining and augmenting our findings for a better fit model using a deep neural network model. There is always room to improve, especially as our model seemingly over-fits our data. We have continued to streamline the model and moreover taken our previous work to begin crafting a generative aspect to our stock.

Additionally, we note that our model does not necessarily generate new data, but predicts stock prices. This model was difficult for us to produce, and throughout our project we did attempt to create a generative model, not predictive. We began working on a future factor input set, in which after the timeline provided we input the factor data that we had, and in turn get the results of the stock prices. This proved to be difficult for us to do, but is something we are interested in continuing to explore in the future.

Finally, we recognize that our data selection was in a US-centric lenses. There is globality in general to the interconnections of the stock market, and in turn how that impacts a company's performance. We would be interested in supporting our model with global metrics and data, particularly when the performance of in this case our focus, KO, are also dependent on international markets. KO is a global company, with sales in over 200 countries of Coca-Cola soda alone (The Coca-Cola Company).

6 Conclusion

In conclusion, the modelling architecture of a deep neural network best support teaching and training a predictive model for the Coca-Cola equity prices. Through further optimization and adding more external factors, this model could be used as a baseline for pre-

dicting stock performance and generating new data. Stock performance and accompanying data is vast and ever-evolving in the field of finance.

Acknowledgements

This research received support during Math 456 course, instructed by Professor Markos Katsoulakis, Department of Mathematics and Statistics at the College of Natural Sciences. We also appreciate coding support from our peers, specifically James Peckham.

References

1. Abdolrasol, M. G. M., Hussain, S. M. S., Ustun, T. S., Sarker, M. R., Hannan, M. A., Mohamed, R., Ali, J. A., Mekhilef, S., & Milad, A. (2021, November 3). Artificial Neural Networks based optimization techniques: A Review. MDPI. https://www.mdpi.com/2079-9292/10/21/2689
2. Ghoraski, V. (2024, January 3). What is a neural network?. GeeksforGeeks. https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/
3. Hall, M. When do economists use real GDP instead of just GDP?. Investopedia. https://www.investopedia.com/ask/answers/030515/when-do-economists-use-real-gdp-instead-gdp.asp: :text=Real
4. Katsoulakis, M. (2024, Spring). Continuous Time Normalizing Flows [PowerPoint slides]. University of Massachusetts Amherst, Amherst, MA.
5. Katsoulakis, M. (2024, Spring). Intro to Normalizing Flows [PowerPoint slides]. University of Massachusetts Amherst, Amherst, MA.
6. Katsoulakis, M. (2024, Spring). More Advanced Normalizing Flows [PowerPoint slides]. University of Massachusetts Amherst, Amherst, MA.
7. Papamakarios, G., Nalisnick, E., Jimenez Rezende, D., Mohamed, S., & Lakshminarayanan, B. (2021, March). Normalizing Flows for Probabilistic Modeling and Inference. Journal of Machine Learning Research, 22, 1-64. Retrieved from https://www.jmlr.org/papers/volume22/19-1028/19-1028.pdf
8. Roberts, B. W. (2009, August). The Macroeconomic Impacts of the 9/11 Attack: Evidence from Real-Time Forecasting. https://www.dhs.gov/sites/default/files/publications/Macroeconomic
9. Singh, G. (2024, April 5). Introduction to artificial neural networks. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/09/introduction-to-artificial-neural-networks/
10. Weng, L. (2018, October 13) Flow-based Deep Generative Models. Lil Log. https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.
11. Katsoulakis, M. (2024, Spring). Neural Networks, Part II [PowerPoint slides]. University of Massachusetts Amherst, MA
12. Katsoulakis, M. (2024, Spring).Lecture - NNs-Examples [PowerPoint slides]. University of Massachusetts Amherst,
13. Zucchi, K. 10-year Treasury bond yield: What it is and why it matters. Investopedia. https://www.investopedia.com/articles/investing/100814/why-10-year-us-treasury-rates-matter.asp: :text=The