**Previsão de Preços com LSTM e GRU: Uma Abordagem de Aprendizado Profundo para a Predição de Criptomoedas Ethereum**

| Carlos Mariano de Souza Rocha Neto  Engenharia de Software - ICEV  carlos\_mariano\_rocha@somosicev.com | Paulo Vinicius Alves Melo Engenharia de Software - ICEV  paulo\_vinicius.melo@somosicev.com | Mateus Mendes Caminha  Engenharia de Software – ICEV  mateus.caminha@somosicev.com |  |
| --- | --- | --- | --- |

**Resumo**

Este artigo apresenta um modelo de aprendizado profundo utilizando Redes Neurais de Longa Memória de Curto Prazo (LSTM) e Unidades Recurrentes Gated (GRU) para a previsão de preços da criptomoeda Ethereum. Além disso, o estudo inclui a aplicação do modelo Prophet como comparação. Os modelos foram treinados com um conjunto de dados de preços históricos e mostraram resultados promissores em termos de precisão na previsão de preços futuros. A abordagem emprega técnicas de normalização de dados e sequências temporais para alimentar as redes neurais, permitindo a captura de padrões complexos em séries temporais financeiras. A análise revela que o modelo GRU, em particular, oferece previsões mais alinhadas com os dados históricos em comparação com o Prophet, destacando sua eficácia na previsão de flutuações no mercado de criptomoedas.

1. **INTRODUÇÃO**

O mercado de criptomoedas tem experimentado um crescimento exponencial em popularidade como um veículo de investimento potencialmente lucrativo nos últimos anos. Profissionais de negociação e investimento estão continuamente em busca de maneiras de aprimorar seus processos de tomada de decisão devido à crescente complexidade e volatilidade desse mercado emergente. Maximizar os ganhos e minimizar os riscos exige a capacidade de prever com precisão as mudanças nas criptomoedas e nos mercados financeiros em geral. Nesse contexto, o uso de algoritmos avançados de aprendizado de máquina tem se destacado como uma abordagem promissora para prever as tendências do mercado. Este estudo se concentra na previsão do preço da criptomoeda Ethereum e avalia o desempenho de dois modelos distintos: Redes Neurais Recorrentes com Unidades Gated Recurrent Units (GRU) e Redes de Memória de Longo Curto Prazo (LSTM). O objetivo é comparar a eficácia desses modelos na previsão de preços e fornecer insights sobre suas respectivas capacidades de previsão.

1. **METODOLOGIA**
   1. **Coleta e Preparação dos Dados**

O conjunto de dados utilizado neste estudo consiste em preços diários de fechamento da criptomoeda Ethereum (ETH-USD). Os dados foram importados da biblioteca Yahoo Finance (yfinance). Inicialmente, os dados foram importados utilizando o seguinte algoritmo:

| import yfinance as yf  end\_date = '2024-06-30'  today = '2024-06-30' # Data final para previsão  df\_ethereum = yf.download('ETH-USD', '2016-01-01', end\_date)  df\_ethereum.reset\_index(inplace=True) |
| --- |

Listing 1: Importação dos dados de preços do Ethereum

Quando são apresentados os dados históricos da criptomoeda Ethereum (ETH-USD), como no caso deste estudo, são encontradas colunas que representam diferentes aspectos das negociações diárias. Especificamente, o conjunto de dados inclui as seguintes colunas:

* Open: Preço de abertura da criptomoeda no início do dia.
* High: Preço mais alto registrado durante o dia.
* Low: Preço mais baixo registrado durante o dia.
* Close: Preço de fechamento da criptomoeda no final do dia.
* Volume: Volume total de transações realizadas durante o dia.
* Adj Close: Preço de fechamento ajustado, que leva em conta eventos corporativos, como dividendos e desdobramentos, se aplicável.
  1. **Definição dos dados de treino e teste**

Após a coleta, inicia-se a fase de preparação dos dados. O objetivo desta etapa é adequar os preços de fechamento para que possam ser utilizados nos modelos, que esperam os dados na forma de matrizes (arrays NumPy) com uma estrutura específica (n samples, n features). Neste caso, n features é 1, pois estamos lidando com uma única característica: o preço de fechamento.

Para preparar os dados para os modelos Prophet e GRU, os seguintes passos são realizados:

* **Preparar dados:** O primeiro passo é reorganizar o DataFrame para o modelo Prophet e normalizar os dados para o modelo GRU.

| import pandas as pd  from sklearn.preprocessing import MinMaxScaler  from fbprophet import Prophet  df = df\_ethereum[["Date", "Adj Close"]]  df.rename(columns={'Date': 'ds', 'Adj Close': 'y'}, inplace=True) |
| --- |

Listing 2: Preparação de dados – GRU

| cripto = "ETH-USD"  inicio = "2016-01-01"  final = "2024-12-31"  dados\_cripto = yf.download(cripto, start=inicio, end=final)  print(dados\_cripto.head()) |
| --- |

Listing 3: Preparação de dados - LSTM

* 1. **Normalizar dados para os modelos:**

Para o modelo GRU, os dados são normalizados para melhorar o desempenho do treinamento.

| data = df['y'].values.reshape(-1, 1)  scaler = MinMaxScaler(feature\_range=(0, 1))  data\_scaled = scaler.fit\_transform(data) |
| --- |

Listing 4: Normalização - GRU

| scaler = MinMaxScaler(feature\_range=(0, 1))  df\_scaled = scaler.fit\_transform(df\_cripto\_fec) |
| --- |

Listing 5: Normalização - LSTM

* 1. **Preparar dados para os modelos:**

Após a preparação dos dados, a próxima etapa é a construção da arquitetura do modelo proposto. Utilizou-se a biblioteca Keras para criar um modelo sequencial composto por duas camadas GRU e camadas densas para a previsão dos valores.

O parâmetro Dense não é um parâmetro específico das camadas GRU, mas é uma camada utilizada em conjunto com o modelo. A camada Dense é caracterizada pela sua propriedade de que cada neurônio está conectado a todos os neurônios da camada anterior. Ela é usada para processar e combinar os recursos aprendidos pelas camadas GRU e produzir a saída final da rede.

| def criar\_sequencias(dados, janela):  X, y = [], []  for i in range(len(dados) - janela):  X.append(dados[i:i + janela])  y.append(dados[i + janela])  return np.array(X), np.array(y)  tamanho\_janela = 60  X, y = criar\_sequencias(data\_scaled, tamanho\_janela)  divisao = int(0.8 \* len(X))  X\_train, X\_test = X[:divisao], X[divisao:]  y\_train, y\_test = y[:divisao], y[divisao:] |
| --- |

Listing 6: Preparação dos dados para o modelo GRU

* 1. **Arquitetura dos modelos**

Após a preparação dos dados, a próxima etapa é a construção da arquitetura do modelo proposto. Utilizou-se a biblioteca Keras para criar dois modelos sequenciais distintos: um com camadas LSTM e outro com camadas GRU.

| model = Sequential()  model.add(LSTM(100, return\_sequences=True, input\_shape=(steps, 1)))  model.add(LSTM(100, return\_sequences=True))  model.add(LSTM(50))  model.add(Dropout(0.2))  model.add(Dense(1)) |
| --- |

Listing 7: Arquitetura LSTM

Neste modelo, as camadas LSTM são usadas para capturar padrões temporais nos dados. A primeira camada LSTM tem 100 unidades e retorna a sequência completa (return\_sequences=True), permitindo que a próxima camada LSTM receba a sequência completa. A segunda camada LSTM também tem 100 unidades e retorna a sequência completa. A terceira camada LSTM tem 50 unidades e retorna apenas o estado final. A camada Dropout é adicionada para ajudar a prevenir o overfitting, e a camada Dense é usada para gerar a saída final da rede.

O modelo GRU é configurado para prever os valores com a seguinte arquitetura:

| from keras.models import Sequential  from keras.layers import GRU, Dense  # Definir a arquitetura do modelo GRU  model\_gru = Sequential()  model\_gru.add(GRU(100, return\_sequences=True, input\_shape=(look\_back, 1))) # Primeira camada GRU  model\_gru.add(GRU(100, return\_sequences=False)) # Segunda camada GRU  model\_gru.add(Dense(25)) # Camada densa intermediária  model\_gru.add(Dense(1)) # Camada densa de saída  # Compilar o modelo  model\_gru.compile(optimizer='adam', loss='mean\_squared\_error')  # Resumir a arquitetura do modelo  model\_gru.summary() |
| --- |

Listing 8: Arquitetura GRU

Neste modelo, as camadas GRU são usadas para capturar padrões temporais nos dados de forma eficiente. A primeira camada GRU tem 100 unidades e retorna a sequência completa (return\_sequences=True), permitindo que a próxima camada GRU receba a sequência completa. A segunda camada GRU também tem 100 unidades, mas retorna apenas o estado final. Uma camada Dense intermediária com 25 unidades é adicionada antes da camada de saída final, que também é uma camada Dense com 1 unidade para gerar a previsão final.

* 1. **Elementos da arquitetura LSTM**
* **Sequential:** Inicializa um modelo sequencial, que permite a construção de uma pilha linear de camadas.
* **LSTM(100, return\_sequences=True, input\_shape=(steps, 1)):** Adiciona a primeira camada LSTM com 100 unidades. O parâmetro return\_sequences=True faz com que a camada retorne a saída completa para cada passo de tempo na sequência de entrada. Isso é necessário quando se deseja que a próxima camada LSTM receba toda a sequência, permitindo que ela aprenda padrões complexos e dependências de longo prazo nos dados temporais. input\_shape=(steps, 1) especifica a forma dos dados de entrada, onde steps é o número de passos de tempo e 1 é o número de características.
* **LSTM(100, return\_sequences=True):** Adiciona a segunda camada LSTM com 100 unidades. O parâmetro return\_sequences=True faz com que esta camada também retorne a saída completa, permitindo que a próxima camada LSTM receba a sequência completa.
* **LSTM(50):** Adiciona a terceira camada LSTM com 50 unidades. O parâmetro return\_sequences=False faz com que a camada retorne apenas a última saída da sequência. Isso é útil quando a camada anterior já processou e extraído as características relevantes da sequência completa, e agora se deseja obter uma única representação (ou vetor de características) que resume a informação de toda a sequência.
* **Dropout(0.2):** Adiciona uma camada Dropout com taxa de dropout de 0.2 para regularização, ajudando a prevenir o overfitting ao desligar aleatoriamente 20% dos neurônios durante o treinamento.
* **Dense(1):** Adiciona a camada densa de saída com 1 unidade, que prevê o valor final.
* **compile(optimizer='adam', loss='mean\_squared\_error'):** Compila o modelo, especificando o otimizador Adam e a função de perda Mean Squared Error.
  1. **Elementos da arquitetura GRU**
* **Sequential:** Inicializa um modelo sequencial, que permite a construção de uma pilha linear de camadas.
* **GRU(100, return\_sequences=True, input\_shape=(look\_back, 1)):** Adiciona a primeira camada GRU com 100 unidades. O parâmetro return\_sequences=True faz com que a camada retorne a saída completa para cada passo de tempo na sequência de entrada. Isso é necessário quando se deseja que a próxima camada GRU receba toda a sequência, permitindo que ela aprenda padrões complexos e dependências de longo prazo nos dados temporais. input\_shape=(look\_back, 1) especifica a forma dos dados de entrada, onde look\_back é o número de passos de tempo e 1 é o número de características.
* **GRU(100, return\_sequences=False):** Adiciona a segunda camada GRU com 100 unidades. O parâmetro return\_sequences=False faz com que esta camada retorne apenas a última saída da sequência. Isso é útil quando a camada anterior já processou e extraído as características relevantes da sequência completa, e agora se deseja obter uma única representação (ou vetor de características) que resume a informação de toda a sequência.
* **Dense(25):** Adiciona uma camada densa intermediária com 25 unidades, que atua para processar a saída da última camada GRU e aprender características adicionais.
* **Dense(1):** Adiciona a camada densa de saída com 1 unidade, que prevê o valor final.
* **compile(optimizer='adam', loss='mean\_squared\_error'):** Compila o modelo, especificando o otimizador Adam e a função de perda Mean Squared Error  
  1. **Treinando os modelos**

O modelo foi treinado utilizando os dados de treinamento e teste, com 120 épocas, um batch size de 30 para o modelo LSTM e 1 para o modelo GRU, e uma função de callback EarlyStopping para evitar o overfitting.

| validation = model.fit(  X\_train, Y\_train,  validation\_data=(X\_test, Y\_test),  epochs=120, batch\_size=30,  verbose=2  ) |
| --- |

Listing 9: Treinamento LSTM

O treinamento do modelo GRU foi configurado da seguinte forma:

| from keras.callbacks import EarlyStopping  # Adicionar early stopping  early\_stopping = EarlyStopping(monitor='loss', patience=10, restore\_best\_weights=True)  # Treinar o modelo GRU com early stopping  num\_epochs = 120 # Número de épocas para treinamento  history = model\_gru.fit(  X,  y,  batch\_size=1,  epochs=num\_epochs,  validation\_split=0.2,  callbacks=[early\_stopping]  ) |
| --- |

Listing 10: Treinamento GRU

* 1. **Previsões**

**Previsão com o Modelo Treinado:** Para obter previsões com o modelo treinado, utilizamos o seguinte código:

| prev = model.predict(X\_test)  prev = scaler.inverse\_transform(prev)  prev |
| --- |

Listing 11: previsões LSTM

* **model.predict(X\_test):** Utiliza o modelo treinado para prever os valores baseados nos dados de teste (X\_test).
* **scaler.inverse\_transform(prev):** Desfaz a normalização dos dados previstos, transformando os valores de volta ao seu intervalo original. Isso é necessário para interpretar as previsões no mesmo contexto dos dados originais.

**Previsão Futura:** Para prever valores futuros com base nos dados escalonados, utilizamos uma função personalizada que gera previsões para um número especificado de períodos futuros. O código da função é:

| def predict\_future(model, data\_scaled, look\_back, n\_future):  predictions = []  last\_data = data\_scaled[-look\_back:]  for \_ in range(n\_future):  X\_input = np.reshape(last\_data, (1, look\_back, 1))  next\_pred = model.predict(X\_input)  predictions.append(next\_pred[0, 0])  last\_data = np.append(last\_data[1:], next\_pred, axis=0)  return np.array(predictions) |
| --- |

Listing 12: previsões LSTM

* **data\_scaled:** Dados normalizados usados para previsões;
* **look\_back:** Número de períodos anteriores utilizados para gerar a entrada para o modelo.
* **n\_future:** Número de períodos futuros para os quais as previsões devem ser feitas.

**Funcionamento da Função:**

* **last\_data = data\_scaled[-look\_back:]:** Seleciona os últimos look\_back períodos dos dados escalonados para iniciar as previsões futuras.
* **for \_ in range(n\_future)::** Loop que itera n\_future vezes para gerar previsões para os períodos futuros desejados.
* **X\_input = np.reshape(last\_data, (1, look\_back, 1)):** Prepara os dados de entrada para o modelo, ajustando a forma para (1, look\_back, 1), onde 1 é o número de amostras e look\_back é o número de períodos anteriores.
* **next\_pred = model.predict(X\_input):** Faz uma previsão para o próximo período com base nos dados de entrada atuais.
* **predictions.append(next\_pred[0, 0]):** Adiciona a previsão à lista de previsões.
* **last\_data = np.append(last\_data[1:], next\_pred, axis=0):** Atualiza os dados de entrada, removendo o período mais antigo e adicionando a nova previsão para o próximo período.

A função retorna um array de previsões para os próximos períodos, permitindo avaliar a tendência futura dos dados com base no modelo treinado.

* 1. **Criação de dataframe de resultados**

Após realizar previsões com o modelo, o próximo passo é criar DataFrames que contêm os valores reais de fechamento das cotações e as previsões correspondentes. Esses DataFrames facilitam a visualização e a comparação dos resultados.

| datas = pd.to\_datetime(dados\_selecionados['Data'])  predict\_datas = pd.date\_range(list(datas)[-1] + pd.DateOffset(1), periods=30, freq='b').tolist()  df\_previsao = pd.DataFrame({'Data': predict\_datas, "preco\_fechamento": future\_previsoes})  df\_previsao['Data'] = pd.to\_datetime(df\_previsao['Data'])  df\_previsao.set\_index('Data', inplace=True) |
| --- |

Listing 13: dataframe LSTM

* **datas:** Converte a coluna de datas dos dados selecionados para o formato de data e hora.
* **predict\_datas:** Gera um intervalo de datas para os próximos 30 períodos úteis, começando um dia após a última data dos dados históricos.
* **df\_previsao:** Cria um DataFrame com as datas futuras e as previsões de preços de fechamento correspondentes.
* **df\_previsao['Data'] = pd.to\_datetime(df\_previsao['Data']):** Converte a coluna de datas para o formato de data e hora.
* **df\_previsao.set\_index('Data', inplace=True):** Define a coluna de datas como o índice do DataFrame para facilitar a visualização temporal.

1. **RESULTADOS**

A análise dos dados históricos e das previsões geradas pelos diferentes modelos foi realizada para avaliar a eficácia dos métodos utilizados na previsão dos preços de fechamento das criptomoedas. O gráfico abaixo ilustra a comparação entre a linha histórica dos dados, as previsões obtidas através do modelo Prophet, do modelo GRU e a previsão combinada.

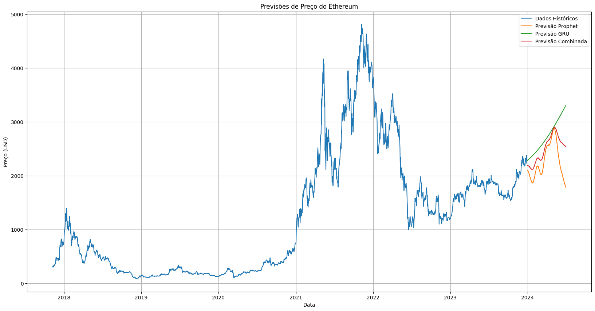


Figura 1: Comparação entre a linha histórica dos preços de fechamento, as previsões do Prophet, do GRU e a

previsão combinada.

O gráfico mostra a linha histórica dos preços de fechamento das criptomoedas ao longo do tempo, representada pela linha azul. As previsões geradas pelo modelo Prophet são exibidas pela linha laranja, enquanto as previsões do modelo GRU são apresentadas pela linha verde. A previsão combinada, que integra as previsões do Prophet e do GRU, é mostrada pela linha vermelha.

Esses resultados indicam que o modelo GRU pode ser mais eficaz na captura de padrões complexos e flutuações no mercado de criptomoedas, sugerindo uma vantagem na sua utilização para previsões mais precisas e detalhadas.

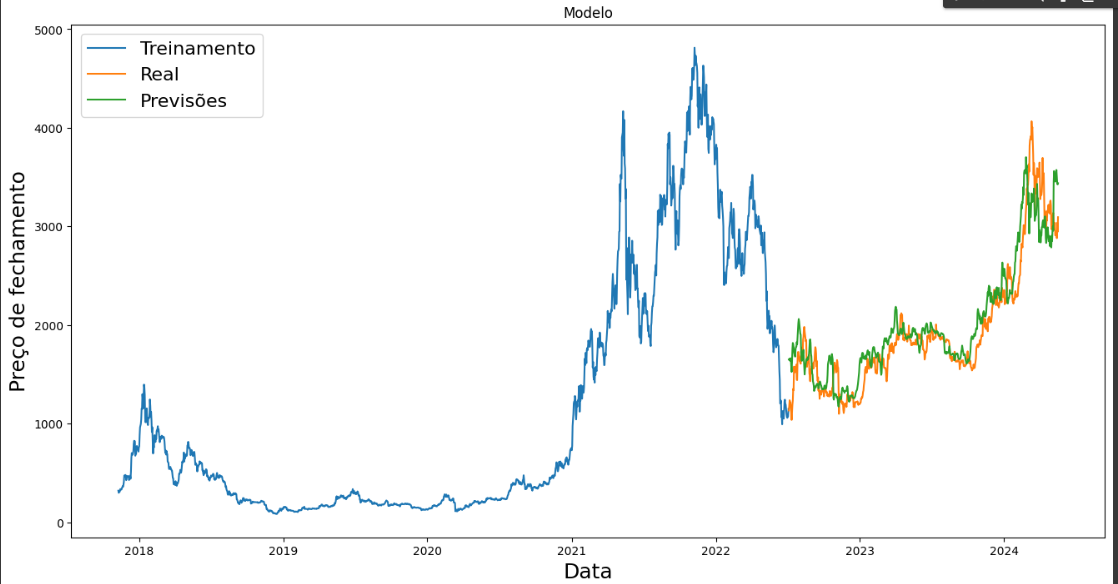


Figura 2: Comparação entre a linha histórica dos preços de fechamento e as previsões do modelo LSTM.

O gráfico mostra a previsão de preços usando um modelo LSTM para uma série temporal financeira. Ele é dividido em três partes: Treinamento representado pela cor azul são dados históricos usados para treinar o modelo, de 2017 a meados de 2022, com grandes flutuações nos preços. Real representado pela cor laranja são dados reais de preços de meados de 2022 a 2024, mostrando recuperação após uma queda significativa. Previsões representadas pela cor verde são as previsões do modelo a partir dos dados reais mais recentes, com boa precisão, apesar de alguns desvios.

Em resumo, o modelo LSTM foi treinado com dados históricos e, posteriormente, utilizado para prever preços futuros. As previsões feitas pelo modelo estão relativamente próximas dos preços reais, o que sugere que o modelo é eficaz para capturar a dinâmica da série temporal financeira apresentada.

1. **DISCUSSAO**

O uso de modelos LSTM e GRU para a previsão de dados de criptomoedas mostrou-se promissor, capturando eficazmente as tendências gerais dos preços e gerando previsões com um baixo RMSE. No entanto, a variabilidade nos erros de previsão e o risco de overfitting destacam a necessidade de abordagens complementares e contínuas melhorias nos modelos.

Enquanto os modelos LSTM e GRU se destacam na modelagem de séries temporais e na identificação de padrões complexos em dados históricos, é importante considerar que as análises baseadas exclusivamente em séries temporais podem ser significativamente impactadas por fatores externos não capturados pelos dados históricos. No contexto das criptomoedas, eventos como mudanças regulatórias, anúncios de grandes players do mercado, e outros fenômenos imprevisíveis podem causar grandes flutuações nos preços, desafiando a capacidade dos modelos de prever com precisão.

Além disso, o modelo Prophet, utilizado para previsões baseadas em componentes de sazonalidade e tendências, também enfrenta limitações semelhantes. Embora eficaz na modelagem de tendências e sazonalidades, o Prophet pode não capturar completamente as dinâmicas específicas e os choques de mercado que afetam as criptomoedas.

Portanto, incorporar técnicas que permitam a inclusão de fatores externos nas análises, como o uso de dados de notícias relevantes ou indicadores econômicos exógenos, pode melhorar a robustez e a eficácia das previsões. Integrar essas abordagens pode fornecer uma visão mais abrangente e realista das dinâmicas do mercado de criptomoedas, aumentando a capacidade dos modelos de lidar com eventos inesperados e melhorar a precisão das previsões.

1. **CONCLUSÃO**

Os resultados demonstram que tanto o modelo LSTM quanto o modelo GRU são eficazes na previsão dos preços de fechamento das criptomoedas, capturando bem as tendências gerais dos dados. Ambos os modelos apresentaram um RMSE relativamente baixo, o que indica uma boa precisão nas previsões. A distribuição dos erros, que está centrada em torno de zero, e a alta correlação entre os preços reais e previstos são evidências do bom desempenho desses modelos.

O modelo LSTM, em particular, destacou-se por sua capacidade de aprender padrões temporais complexos e oferecer previsões consistentes. Da mesma forma, o modelo GRU, com sua estrutura mais simplificada, também conseguiu capturar as principais tendências de forma eficaz.

Além disso, o modelo Prophet contribuiu significativamente ao fornecer uma visão geral das tendências e sazonalidades, oferecendo uma perspectiva complementar às previsões dos modelos LSTM e GRU.

Embora os modelos tenham mostrado um rendimento acumulado satisfatório ao longo do tempo, sugerindo uma boa capacidade de previsão, é importante destacar que a variabilidade nos erros de previsão e o risco de overfitting indicam a necessidade de melhorias contínuas e abordagens complementares. Incorporar técnicas que considerem fatores externos, como dados de notícias relevantes ou indicadores econômicos exógenos, pode aumentar a robustez das previsões e oferecer uma visão mais completa das dinâmicas do mercado de criptomoedas.

Em suma, a combinação de diferentes modelos e a inclusão de variáveis externas são estratégias promissoras para aprimorar a precisão e a eficácia das previsões de preços no mercado de criptomoedas, fornecendo uma ferramenta mais robusta para investidores e analistas.

1. **REFERENCIAS**

[1] Mohanty, S. (2023). An International Study of Application of Long Short-Term Memory (LSTM) Neural Networks for the Prediction of Stock And Forex Markets. International Journal for Multidisciplinary Research (IJFMR), 5(3), 1-9.

[2] He, B., et al. (2023). A Stock Price Prediction Method based on LSTM and K-Means. Frontiers in Science and Engineering, 3(6), 44-47.

[3] Tingley, D. (2019). yfinance: Download market data from Yahoo! Finance’s API. Retrieved from https://pypi.org/project/yfinance/

[4] McKinney, W. (2010). Data Structures for Statistical Computing in Python. In Proceedings of the 9th Python in Science Conference (pp. 51-56). Retrieved from https://pandas.pydata.org/

[5] Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90-95. Retrieved from https://matplotlib.org/

[6] Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830. Retrieved from https://scikit-learn.org/

[7] Chollet, F., et al. (2015). Keras. Retrieved from https://keras.io/

[8] Harris, C. R., et al. (2020). Array programming with NumPy. Nature, 585, 357-362. Retrieved from https://numpy.org/

[9] Waskom, M., et al. (2021). Seaborn: Statistical data visualization. Journal of Open Source Software, 6(60), 3021. Retrieved from https://seaborn.pydata.org/

[10] Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace. Retrieved from https://www.python.org/

[11] Lathkar, P. (2020). Mastering Matplotlib 2.x. Packt Publishing Ltd. ISBN: 9781788625177.

[12] NumPy Documentation. (2023). Retrieved from https://numpy.org/doc/stable/

[13] Cho, K., et al. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1724-1734. Retrieved from https://arxiv.org/abs/1406.1078