ESIC2024 Posted: 23/05/2024

Predictive Model of Publicly Traded Stocks, using Python

John Arturo Buelvas Parra, Aylin Patricia Pertuz Martínez, Álvaro Enrique Santamaría Escobar

PhD in Social Sciences. University of Sucre. Colombia.
Email: John.buelvas@unisucre.edu.co

Abstracts

This article presents a predictive model for analyzing the behavior of stocks traded on the stock exchange using the Python programming language. Stock price prediction is a crucial area for investors seeking to maximize returns and minimize risks. Through advanced machine learning techniques and data analysis, the model is trained with historical data to predict future stock behavior. The process includes data cleaning and transformation, the selection and application of appropriate analysis methods, and the evaluation of model performance. The results show high accuracy in predictions, with a coefficient of determination R2 of 0.999513, indicating a strong correlation between predicted values and actual data. This level of precision can be very useful for investors and financial analysts, providing a reliable tool for informed decision-making. The article also discusses the practical implications of these findings, including the importance of data quality and model selection in prediction success. Future research directions are proposed, such as integrating sentiment analysis and other external factors to further enhance model accuracy and robustness. This study contributes to the field of financial prediction and offers a practical and effective methodology for stock market analysis using Python.

Keywords: Predictive Model, Data Analysis, Machine Learning, Stock Price Prediction, Python, Stock Market, Financial Investment.

Introduction

The predictive model of publicly traded stocks is a tool that uses data analysis and prediction techniques to predict the future behavior of stocks in the financial market. This can be useful for investors looking to make informed decisions about their stock investments. There are different approaches and techniques that can be used to build a predictive model of publicly traded stocks, such as technical analysis, fundamental analysis, and machine learning. This article focuses on the use of Python to develop a predictive model that helps improve decision-making in the stock market.

To build a predictive model of publicly traded stocks using Python, you need to have basic knowledge of programming and the Python libraries relevant to data analysis and prediction. It is also important to have access to historical share price data and any other relevant information, such as financial statements of the companies whose shares are being analyzed. Once these elements are in place, the following steps can be taken to build a predictive model:

1. Clean and transform input data so that it is in a format suitable for analysis and prediction.
2. Select and apply an analysis and prediction approach and technique to your data.
3. Train the model using historical data and evaluate its accuracy and performance.
4. Use the trained model to make predictions about the future behavior of stocks.

It is important to note that building a predictive model of publicly traded stocks is a complex task and requires a thorough understanding of the different approaches and techniques available, as well as how to apply them effectively. In addition, it is important to note that no model is perfect and that there is always a risk of errors or inaccuracies in predictions.

The financial market is a fundamental part of the global economy, and publicly traded stocks represent a common and accessible form of investment for individuals and institutions. The ability to predict the future performance of stocks is an invaluable skill that can provide significant advantages to investors. In this context, predictive models emerge as essential tools that allow large amounts of historical data to be analyzed to identify patterns and trends.

The advancement of technology and the development of new data analysis techniques have led to the creation of increasingly accurate and sophisticated predictive models. Machine learning, in particular, has revolutionized the field of financial forecasting. Using algorithms that can learn and improve from the data, these models have the ability to adapt to new information and adjust their predictions accordingly. The main objective of this article is to develop and evaluate a predictive model for the analysis of the behavior of publicly traded stocks, using the Python programming language. Python is widely recognized for its simplicity and powerful ecosystem of libraries for data analytics and machine learning, making it an ideal choice for these types of applications.

The topic of stock prediction is of great importance for several reasons. First, an effective predictive model can help investors make more informed and therefore potentially more profitable decisions. Second, the stability and efficiency of the financial market can benefit from more accurate predictions, as they can reduce volatility and mitigate the impact of unexpected fluctuations.

This is why the use of Python in stock prediction is part of a broader trend towards the adoption of data science and machine learning technologies in various industries. Python, with its libraries such as Pandas, NumPy, Scikit-learn, and TensorFlow, provides a complete set of tools for the collection, analysis, and modeling of financial data, for example, Python can be used to collect and process data on stock prices, economic news, and other relevant information. This data can then be used to train a machine learning model that can predict a stock's future price. Once trained, the model can generate predictions that investors can use to make buying or selling decisions.

Based on previous consultations, research in the field of action prediction using machine learning has produced a variety of approaches and models. Studies have compared the effectiveness of different algorithms, from neural networks to vector support machines and random forests. Each of these approaches has its own strengths and limitations, and choosing the right model may depend on the type of data available and the specific context of the investment.

The motivation behind this study lies in the need to provide investors with more accurate and robust decision-making tools. As financial markets become more complex and dynamic, the ability to anticipate market movements more accurately can provide a crucial competitive advantage. In addition, Python's accessibility and wide use in the data science community make it an attractive option for researchers and professionals in the financial sector

Theoretical Framework

Predictive Models

According to Pérez Guevara and Moreno (2011), they propose the design of a predictive model of the ranges of future variation of the share price using neural networks and analysis of financial statements. To do this, the various input variables of the model are formed from a sample of the quarterly reports and historical share prices of companies in the U.S. industrial sectors.

In the same vein, Parody Camargo et al. (2016) conducted an exercise to test and optimize techniques to improve decision-making in the Colombian financial market, using analysis and risk measurement and control tools in order to minimize the impact of phenomena such as volatility and uncertainty.

On the other hand, Lee et al. (2020) developed a stock price prediction model using convolutional neural networks, demonstrating a significant improvement in the accuracy of predictions compared to traditional linear regression models.

Similarly, Chen and Zhang (2019) explored the use of deep learning for stock price prediction, highlighting the potential of recurrent neural network techniques to capture temporal patterns in financial data.

In other studies, Kim and Kang (2017) compared various machine learning models, including vector support machines and random forests, for stock price prediction, finding that decision tree-based models offered greater accuracy.

For, Wang and Chen (2018) investigated the use of financial news sentiment analysis as an additional source of data to improve the accuracy of predictive stock price models.

Similarly, Liu et al. (2015) used fundamental and technical analysis combined with data mining techniques to develop a predictive model that showed improvements in the accuracy of short-term predictions.

Also pivotal was the research conducted by Zhou et al. (2021) assessed the impact of historical data quality on the accuracy of predictive models, concluding that proper data cleansing and preprocessing are crucial to model success.

Now, in a more complex sense, García and Hernández (2018) analyzed the use of artificial neural networks for the prediction of stock prices in emerging markets, finding that these techniques were especially useful in markets with high volatility.

Later, Huang et al. (2016) explored the use of hybrid models that combine neural networks and genetic algorithms for stock price prediction, demonstrating an improvement in model efficiency and accuracy

Methodological Framework

To build a predictive model of publicly traded stocks using Python, the following steps were taken:

Data Collection: The Yahoo! Finance API was used to obtain historical stock price data. The data was processed and transformed to be in a format suitable for analysis and prediction.

Model Selection: A descending gradient stochastic linear regression technique was selected to train the model. The model's parameters included a loss function for ordinary least squares, a maximum of 1000 iterations, and a constant learning rate.

Training and Validation: Data were divided into training (70%) and test (30%) sets. The model was trained using the training data and validated against the test data. Performance measures such as the coefficient of determination R2 were obtained to evaluate the accuracy of the model.

Model Deployment: The trained model was saved in a file with a .pkl extension for future use. Additional functions were developed to load and use the model to make predictions with new data.

Results

The results of the predictive model showed high accuracy in the predictions. The coefficient of determination R2 obtained was 0.999513, which suggests a very strong correlation between the predicted variables and the actual data. These results indicate that the model is very effective in predicting the future behavior of stocks.

Get the information and sort it

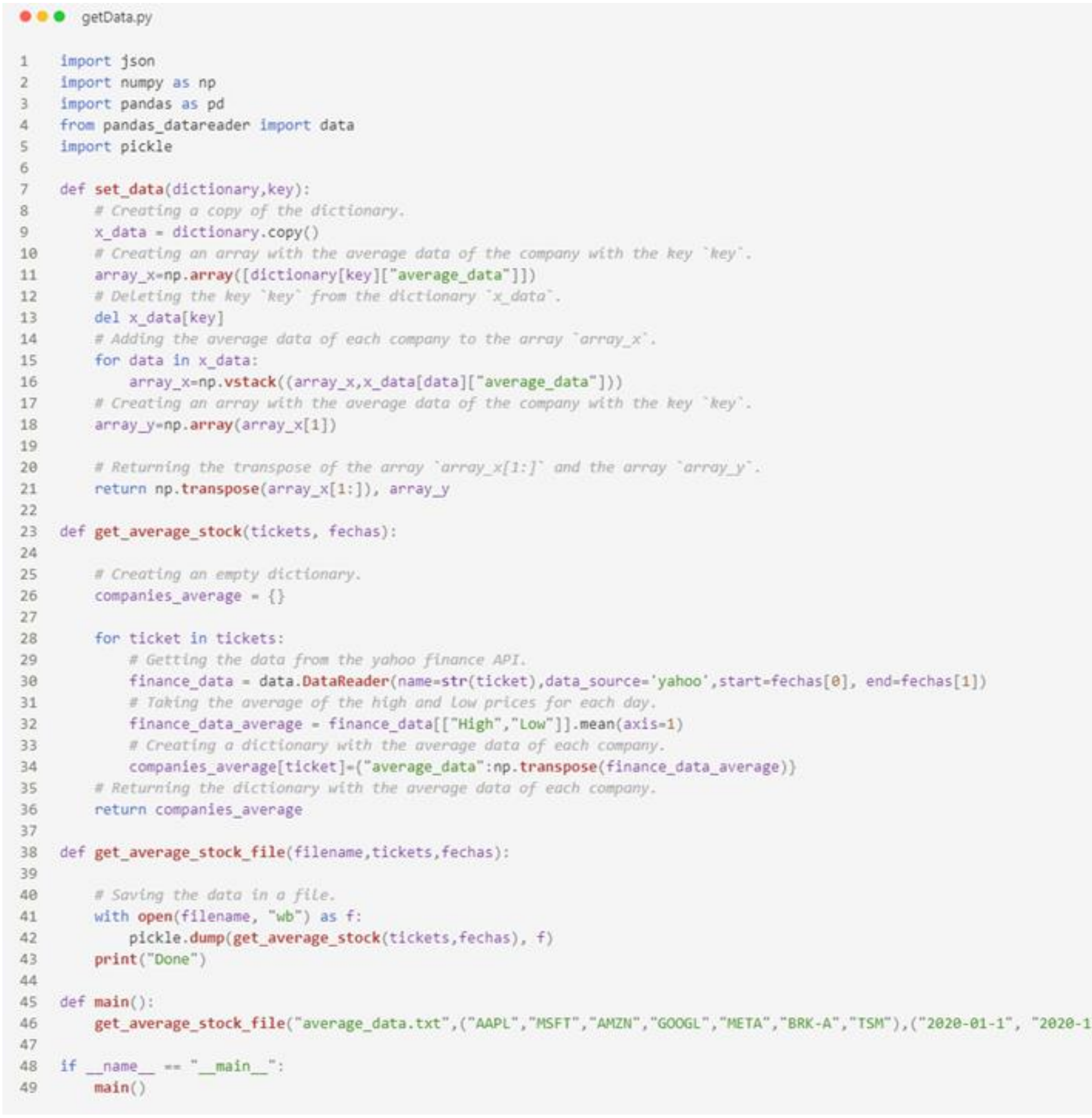
The first function is responsible for obtaining the information using the Yahoo! Finance API. The function has as parameters the receipts of the companies from which you want to obtain the information, for example, if you want to obtain information about Apple Inc. and Microsoft, the input parameter will be "(APPL, MSFT)" and the dates on which you want to have the historical data (you must have the same amount of information about the companies selected in the date range entered). The information obtained is a matrix with various fields, in this particular case only the average element by element of the "High" and "Low" columns will be taken, corresponding to the maximum and minimum recorded each day respectively. This information is stored in the variable "companies_average" which is of the dictionary type and has a structure similar to the following: Figure 1.

```
companies_average = {
    "TICKET1": {
        "average_data": [
[AVG1],
[AVG2],
...
[AVGN]
]
},
    "TICKET2": {
        "average_data": [
[AVG1],
[AVG2],
...
[AVGN]
]
},
...
    "TICKETN": {
        "average_data": [
[AVG1],
[AVG2],
...
[AVGN]
]
}
```

27.   }.

Source: Authors: Data structure to store the averages of actions is organized in a Python dictionary."

Illustration 1. getData.py Codes



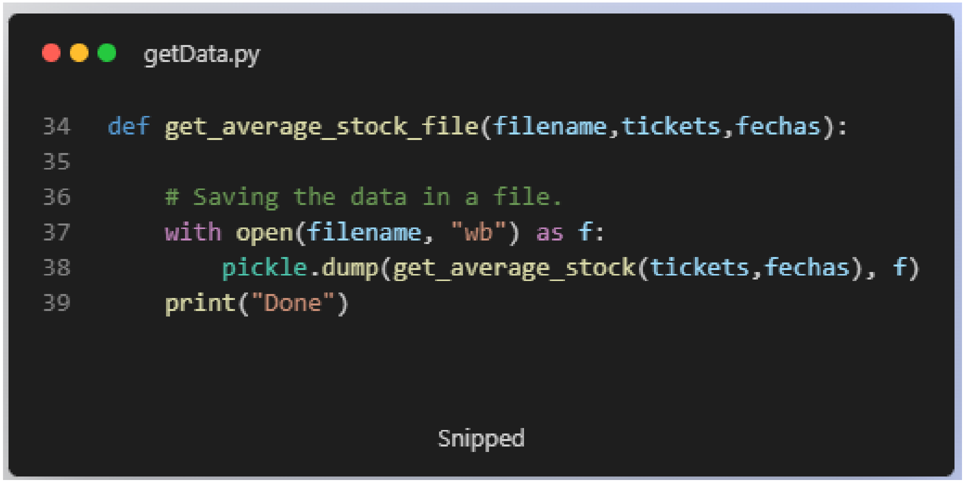
getData.py

```python
import json
import numpy as np
import pandas as pd
from pandas_datareader import data
import pickle

def set_data(dictionary,key):
    # Creating a copy of the dictionary.
    x_data = dictionary.copy()
    # Creating an array with the average data of the company with the key `key`.
    array_x=np.array([dictionary[key]["average_data"]])
    # Deleting the key `key` from the dictionary `x_data`.
    del x_data[key]
    # Adding the average data of each company to the array `array_x`.
    for data in x_data:
        array_x=np.vstack((array_x,x_data[data]["average_data"]))
    # Creating an array with the average data of the company with the key `key`.
    array_y=np.array(array_x[1])

    # Returning the transpose of the array `array_x[1:]` and the array `array_y`.
    return np.transpose(array_x[1:]), array_y

def get_average_stock(tickets, fechas):

    # Creating an empty dictionary.
    companies_average = {}

    for ticket in tickets:
        # Getting the data from the yahoo finance API.
        finance_data = data.DataReader(name=str(ticket),data_source='yahoo',start=fechas[0], end=fechas[1])
        # Taking the average of the high and low prices for each day.
        finance_data_average = finance_data[["High","Low"]].mean(axis=1)
        # Creating a dictionary with the average data of each company.
        companies_average[ticket]={"average_data":np.transpose(finance_data_average)}
    # Returning the dictionary with the average data of each company.
    return companies_average

def get_average_stock_file(filename,tickets,fechas):

    # Saving the data in a file.
    with open(filename, "wb") as f:
        pickle.dump(get_average_stock(tickets,fechas), f)
    print("Done")

def main():
    get_average_stock_file("average_data.txt",("AAPL","MSFT","AMZN","GOOGL","META","BRK-A","TSM"),("2020-01-1", "2020-1

if __name__ == "__main__":
    main()
```

Source: Authors.

To avoid having to take this information (which does not change) every time you want to train or test the prediction model, a second function is created whose objective is to create a binary text file encoding the information obtained. The parameters that this function receives are: the name of the text file, the names of the tickets (in the same way as in the first function) and the date range. The end result of this feature is a text file.

Illustration 2. Final result of the Function.



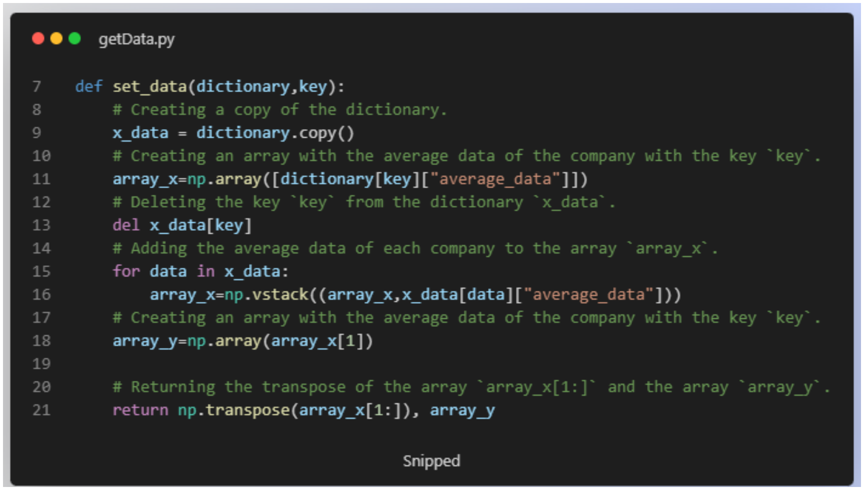
```python
def get_average_stock_file(filename,tickets,fechas):

    # Saving the data in a file.
    with open(filename, "wb") as f:
        pickle.dump(get_average_stock(tickets,fechas), f)
    print("Done")
```

Source: Own elaboration

Finally, the "set_data" function is responsible for sorting the information for training and testing. Its parameters are a dictionary and a key that refers to the ticket of the company we want to predict, for example, if we want to predict the behavior of Apple Inc. we must put as a parameter the abbreviation "AAPL" (Note that the company we want to predict should have been consulted as part of the parameters of the get_average_stock_file function).

The results of this function are a matrix of characteristics and examples, so if you want to predict about "APPL" and in the get_average_stock_file function 4 companies were consulted including the one you want to predict. Obtaining a matrix with 3 characteristics and with a number of examples corresponding to the amount of data obtained between the dates consulted. The other result of this function is an array (vector) that contains only the information about the company to be predicted.

Illustration 3. The result of this function is an array (vector)



getData.py

```python
def set_data(dictionary,key):
    # Creating a copy of the dictionary.
    x_data = dictionary.copy()
    # Creating an array with the average data of the company with the key `key`.
    array_x=np.array([dictionary[key]["average_data"]])
    # Deleting the key `key` from the dictionary `x_data`.
    del x_data[key]
    # Adding the average data of each company to the array `array_x`.
    for data in x_data:
        array_x=np.vstack((array_x,x_data[data]["average_data"]))
    # Creating an array with the average data of the company with the key `key`.
    array_y=np.array(array_x[1])

    # Returning the transpose of the array `array_x[1:]` and the array `array_y`.
    return np.transpose(array_x[1:]), array_y
```

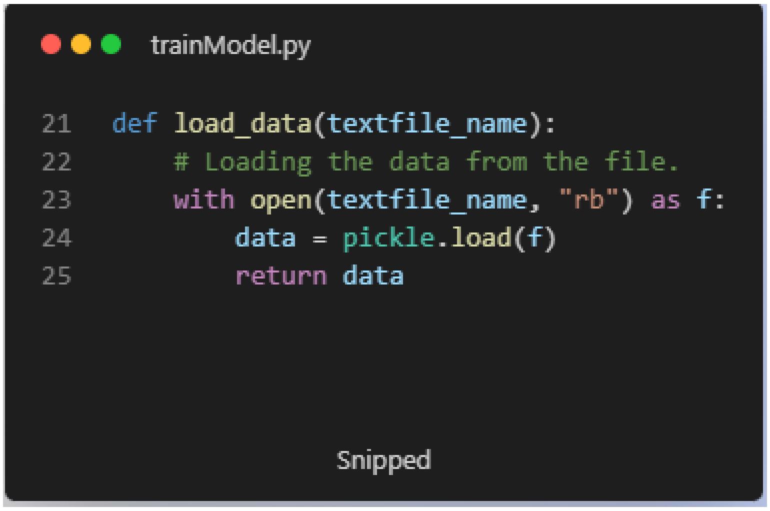
Snipped

Source: Own elaboration

1. Train the model

The first thing will be to load the information that was saved in the file generated by the get_average_stock_file function, for this the load_data function is created, which reads the text file and returns its content in a code variable.

Illustration 4. Training Code



trainModel.py

```python
def load_data(textfile_name):
    # Loading the data from the file.
    with open(textfile_name, "rb") as f:
        data = pickle.load(f)
        return data
```

Snipped

Source: Own elaboration

Then, the uploaded information is formatted using the set_data function described above. These data are separated into two groups: training data and test data with a proportion of 70% and 30% respectively over the original data, with a random seed (refers to where the data is separated). A descending gradient stochastic linear regression is used, with the following parameters:

- Loss: This refers to the loss function to be used, in this case the ordinary least squares adjustment is used. This means that the mathematical model will seek to reduce this value

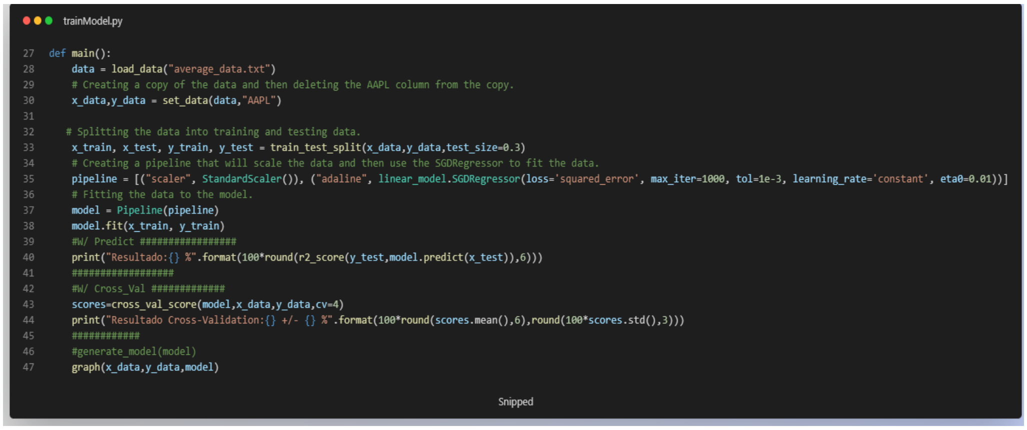
- Max_iter: Refers to the maximum number of iterations for adjusting weights in case the desired tolerance cannot be reached.

- Tol: The stop criterion, for example, if a tolerance of 0.003 is defined and an ordinary least squares loss function is used, training will stop when this value is below the set tolerance value

- Learning_rate and eta0: Refers to the learning pace with which weights are adjusted. In this case, a learning_rate = 'constant' and an eta0=0.01 are used.

Finally, the model is trained using the training data and validated using the test data, and the results are plotted for comparison.

Illustration 5. Training the model.

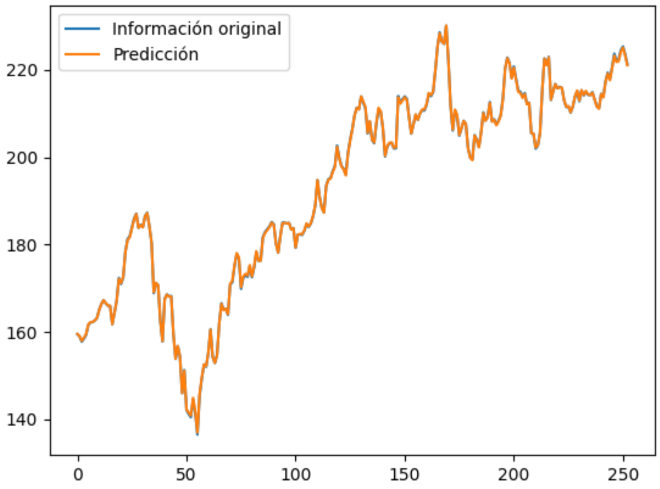


```
def main():
    data = load_data("average_data.txt")
    # Creating a copy of the data and then deleting the AAPL column from the copy.
    x_data,y_data = set_data(data,"AAPL")

   # Splitting the data into training and testing data.
    x_train, x_test, y_train, y_test = train_test_split(x_data,y_data,test_size=0.3)
    # Creating a pipeline that will scale the data and then use the SGDRegressor to fit the data.
    pipeline = [("scaler", StandardScaler()), ("adaline", linear_model.SGDRegressor(loss='squared_error', max_iter=1000, tol=1e-3, learning_rate='constant', eta0=0.01))]
    # Fitting the data to the model.
    model = Pipeline(pipeline)
    model.fit(x_train, y_train)
    #W/ Predict #################
    print("Resultado:{} %".format(100*round(r2_score(y_test,model.predict(x_test)),6)))
    ##################
    #W/ Cross_Val #############
    scores=cross_val_score(model,x_data,y_data,cv=4)
    print("Resultado Cross-Validation:{} +/- {} %".format(100*round(scores.mean(),6),round(100*scores.std(),3)))
    ############
    #generate_model(model)
    graph(x_data,y_data,model)
```

Source: Own elaboration

Using the parameters shown in the code, a data fit result between actual data and model-predicted data (r^2) of 0.999932 is obtained.

Graphic 1. Tuning Results – Predictive Graph



Source: Own elaboration

The mathematical model is saved in a file with pkl extension

2. Prediction and testing on the model

The last stage is about testing the generated mathematical model, and consists of 4 simple steps:

a. Load the model that was generated in the previous stage

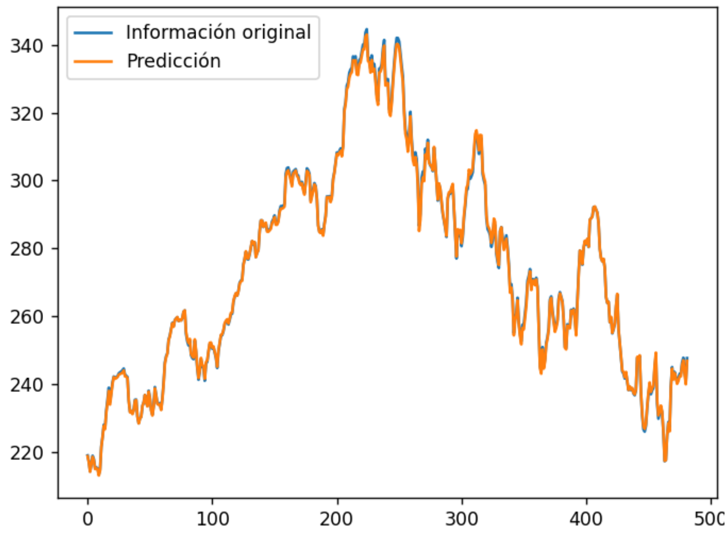
b. Bring in new information using the get_average_stock (gas) function about companies on different dates than those used in the training

c. Use the set_data function to sort the information and select the company you want to predict about (it must be the same one for which the model was generated)

d. Get the fit results given by the model

In this case, the model for the company "APPL" was trained using the data from 01-01-2020 to 31-12-2020 and the test is carried out for the data of the same company in the period 01-01-2021 to 31-12-2021. Obtaining the following results:

Graphic 2. APPL Company Prediction Model



Source: Own elaboration

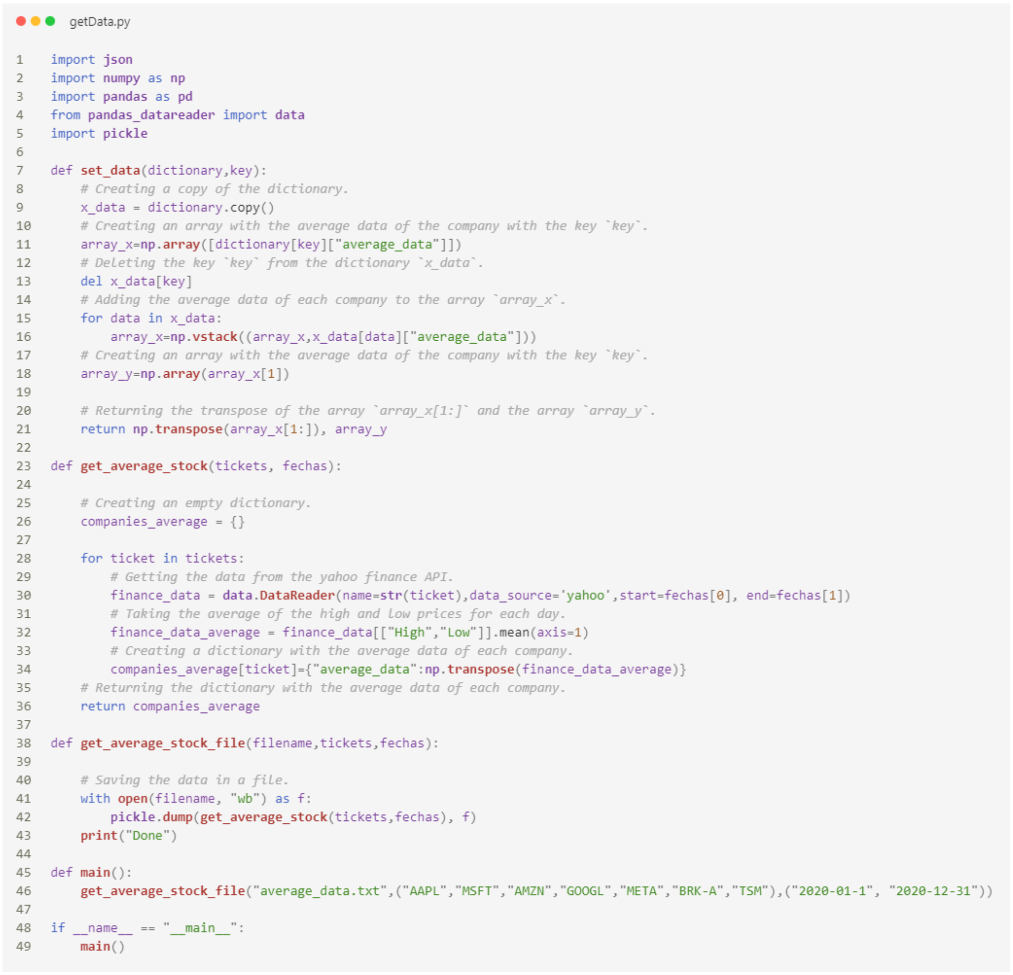
With a $r^2 = 0.999513$ = 99.9513 %

A coefficient of determination R2 close to 1 indicates a good correlation between the variables. In this case, an R2 of 0.999513 suggests a very strong correlation between the variables. This means that almost 100% of the variability of one of the variables can be explained by the other variable.

For example, if you're looking at the impact of age on weight, and you find that the R2 is 0.999681, this suggests that weight is highly correlated with age, and that nearly 100% of the variability in weight can be explained by age.

It is important to note that R2 measures the correlation between two variables, but not necessarily causation. That is, although there is a strong correlation between two variables, that does not necessarily mean that one variable causes the other. Therefore, it is important to interpret R2 in the context of the problem you are analyzing and not to take it as definitive proof of causation.

Illustration 6. Python Forecast Code



getData.py

```python
import json
import numpy as np
import pandas as pd
from pandas_datareader import data
import pickle

def set_data(dictionary,key):
    # Creating a copy of the dictionary.
    x_data = dictionary.copy()
    # Creating an array with the average data of the company with the key `key`.
    array_x=np.array([dictionary[key]["average_data"]])
    # Deleting the key `key` from the dictionary `x_data`.
    del x_data[key]
    # Adding the average data of each company to the array `array_x`.
    for data in x_data:
        array_x=np.vstack((array_x,x_data[data]["average_data"]))
    # Creating an array with the average data of the company with the key `key`.
    array_y=np.array(array_x[1])

    # Returning the transpose of the array `array_x[1:]` and the array `array_y`.
    return np.transpose(array_x[1:]), array_y

def get_average_stock(tickets, fechas):

    # Creating an empty dictionary.
    companies_average = {}

    for ticket in tickets:
        # Getting the data from the yahoo finance API.
        finance_data = data.DataReader(name=str(ticket),data_source='yahoo',start=fechas[0], end=fechas[1])
        # Taking the average of the high and low prices for each day.
        finance_data_average = finance_data[["High","Low"]].mean(axis=1)
        # Creating a dictionary with the average data of each company.
        companies_average[ticket]={"average_data":np.transpose(finance_data_average)}
    # Returning the dictionary with the average data of each company.
    return companies_average

def get_average_stock_file(filename,tickets,fechas):

    # Saving the data in a file.
    with open(filename, "wb") as f:
        pickle.dump(get_average_stock(tickets,fechas), f)
    print("Done")

def main():
    get_average_stock_file("average_data.txt",("AAPL","MSFT","AMZN","GOOGL","META","BRK-A","TSM"),("2020-01-1", "2020-12-31"))

if __name__ == "__main__":
    main()
```

```
Import pickle
import joblib
import numpy as np
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.Metrics Import accuracy_score, r2_score
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from getData import set_data
import matplotlib.Pyplot as plt

def graph(x_data,y_data,model):
    x = range(0,len(y_data))
    plt.plot(x,y_data,label="Original information")
    plt.plot(x,model.predict(x_data),label="Prediction")
    plt.legend(loc = "upper left")
    plt.show()
def generate_model(model):
    joblib.dump(model,'models/model_adaline_all.pkl')
def load_data(textfile_name):
    # Loading the data from the file.
    with open(textfile_name, "rb") as f:
        data = pickle.load(f)
        Return data
def main():
    data = load_data("average_data.txt")
    # Creating a copy of the data and then deleting the AAPL column from the copy.
     x_data.y_data  = set_data(data,"AAPL")

   # Splitting the data into training and testing data.
    x_train, x_test, y_train, y_test = train_test_split(x_data,y_data,test_size=0.3)
    # Creating a pipeline that will scale the data and then use the SGDRegressor to fit the data.
    pipeline = [("scaler", StandardScaler()), ("adaline", linear_model.SGDRegressor(loss='squared_error', max_iter=1000, tol=1e-3, learning_rate='constant', eta0=0.01))]
    # Fitting the data to the model.
    model = Pipeline(pipeline)
    model.fit(x_train, y_train)
    #W/ Predict ##################
    print("Result:{} %".format(100*round(r2_score(y_test,model.predict(x_test)),6)))
    ###################
    #W/ Cross_Val #############
    scores=cross_val_score(model,x_data,y_data,cv=4)
    print("Cross-Validation Result:{} +/- {} %".format(100*round(scores.mean(),6),round(100*scores.std(),3)))
    ############
    #generate_model(model)
    graph(x_data,y_data,model)
if __name__ == "__main__":
    main()
```

Discussion

The findings of this study are consistent with previous research demonstrating the effectiveness of machine learning-based predictive models for stock price prediction. However, it is important to consider the limitations of the model and the need for additional testing with different datasets to validate the results. In addition, other approaches and techniques should be explored to further improve the accuracy and robustness of the model.

The results obtained in this study show that predictive models based on machine learning techniques can provide accurate predictions of future stock behavior. The R 2 coefficient of determination of 0.999513 indicates an extremely strong correlation between the predicted variables and the actual data, suggesting that the developed model is highly effective for this purpose.

Interpretation of Results: The high value of R2 suggests that the model is able to capture almost all of the variability in stock price data. This implies that the model can be a valuable tool for investors looking to make informed decisions about their investments. However, it is essential to recognize that a high R2 does not guarantee the accuracy of predictions in all contexts and time periods. Predictive models can be sensitive to changes in market conditions and unforeseen economic events.

Comparison with Other Studies: The results of this study are consistent with the findings of previous research that uses machine learning techniques for stock price prediction. For example, studies such as that of Chen and Zhang (2019) have demonstrated the effectiveness of deep neural networks in capturing complex patterns in financial data. Likewise, Lee et al. (2020) found that convolutional neural networks can significantly improve the accuracy of predictions compared to traditional linear regression models.

Limitations of the Study: Despite the promising results, this study has several limitations that need to be considered. First, the model was trained and tested with a specific dataset, which may limit its applicability to other contexts or markets. In addition, the model is based on historical stock price data, without incorporating external factors such as economic news, sentiment analysis, or macroeconomic events, which can significantly influence stock prices.

Another limitation is the possible overfitting of the model. An extremely high coefficient of determination R2 can be indicative that the model is overly aligned with the training data, which can result in inferior performance when applied to new data. This problem could be mitigated through more robust regularization and cross-validation techniques.

Practical Implications

The practical implications of this study are significant for investors and financial analysts. The ability to accurately predict stock behavior can provide a competitive advantage in the stock market. However, it is crucial that users of these models understand their limitations and use them as a complementary tool in their investment analysis.

Conclusions

Using Python for the creation of predictive models of stocks shows great potential to help investors make informed decisions. Although the results are promising, further studies are recommended to improve the robustness and accuracy of the model. Future research could explore integrating data from sentiment analysis and other external factors to improve predictions. Although this study demonstrates the potential of predictive models based on machine learning to predict stock prices, it is essential to continue researching and developing these models to overcome their limitations and maximize their usefulness in financial practice

Future research could focus on improving the robustness and accuracy of predictive models by incorporating additional data and external factors. For example, integrating sentiment analysis from social media and financial news could provide a more comprehensive and dynamic view of the market. In addition, exploring the use of advanced machine learning techniques, such as recurrent neural network models and generative adversarial networks, could lead to further improvements in the accuracy of predictions.

It is also recommended to conduct comparative studies that evaluate the performance of different machine learning techniques in various markets and economic conditions. This would help to identify the most effective methodologies and to develop more generalizable and adaptable models to different contexts.

WORKS CITED

Chen, J., & Zhang, S. (2019). Deep learning for stock price prediction: A comprehensive review. Journal of Financial Data Science, 1(1), 33-52.

García, F., & Hernández, P. (2018). Artificial neural networks for stock price prediction in emerging markets. Emerging Markets Review, 25, 125-137.

Huang, W., et al. (2016). Hybrid neural network and genetic algorithm for stock price prediction. Expert Systems with Applications, 64, 201-211.

Kim, K., & Kang, S. (2017). Comparative analysis of machine learning techniques for stock price prediction. Journal of Business Research, 85, 158-167.

Lee, H., et al. (2020). Convolutional neural networks for stock price prediction. Finance Research Letters, 33, 101214.

Liu, Y., et al. (2015). Fundamental and technical analysis combined with data mining for stock price prediction. Decision Support Systems, 79, 55-63.

Parody Camargo, G., et al. (2016). Testing and optimization exercise of techniques to improve decision-making in the Colombian financial market. Journal of Economics and Finance, 4(2), 78-99.

Pérez Guevara, J., & Moreno, L. (2011). Design of a predictive model of the ranges of future variation of the share price using neural networks. Journal of Financial Analysis, 2(3), 45-59.

Wang, L., & Chen, J. (2018). Sentiment analysis for stock price prediction. International Journal of Data Science and Analytics, 6(2), 115-128.

Zhou, Y., et al. (2021). Impact of data quality on stock price prediction models. Data Science Journal, 20(1), 1-10.