

# Time Series Analysis

## JP Morgan Stock Price Analysis

by  
**Jay Jain**

Dataset: JP Morgan Stock Price

Dataset Link: [JPMorgan Chase & Co. \(JPM\) Stock Historical Prices & Data - Yahoo Finance](#)

## Table of Contents

Sr.No	Topic	Pg.No.
1.	Introduction	3
2.	Data Description	4
3.	Objective	5
4.	Data Cleaning	6
5.	Data Decomposition	7
6.	Smoothing Methods	9
7.	Testing Stationary	10
8.	Justification for Time Series	12
9.	Implementation and Interpretation for Forecast	13
10.	Reason for selecting the model	14
11.	Comparative Result Analysis	15
12.	Conclusion	16
13.	Future Scope	16
14.	References	17

# Introduction

**Time series analysis** can be useful to see how a given **asset, security**, or **economic** variable changes over time. Stock Market prices are something which change over time based on various factors such as Government Policies, Company News, Pandemics, Previous Prices etc

To study these **trends, irregularities**, and **volatility** in the stock prices, time series model(s) and techniques must be used. I chose data from one of the biggest financial firms in the world i.e. JP Morgan Chase and Co. to perform methods and **forecasting** using time series methods. This report includes the entire pipeline from data cleaning to fitting the model and forecasting the possible stock prices based on the **historical** data of stock currently available.

## Data Description

Currency in USD

[Download](#)

Date	Open	High	Low	Close*	Adj Close**	Volume
Dec 02, 2022	134.49	135.35	133.34	135.16	135.16	8,931,900
Dec 01, 2022	138.18	138.66	135.43	136.24	136.24	8,858,500
Nov 30, 2022	136.09	138.18	133.19	138.18	138.18	14,753,700
Nov 29, 2022	134.66	136.64	134.41	136.56	136.56	7,925,300
Nov 28, 2022	136.07	136.71	134.19	134.35	134.35	9,906,900
Nov 25, 2022	136.48	137.14	136.06	136.74	136.74	3,220,500
Nov 23, 2022	134.94	136.50	134.86	136.48	136.48	7,316,200
Nov 22, 2022	134.00	135.27	133.69	135.04	135.04	9,185,700

The dataset contains the following columns:

1. Date (Daily)
2. **Open** Price of Stock on that day
3. **High** Price of Stock on that day
4. **Low** Price of Stock on that day
5. **Close** Price of Stock on that day
6. **Adj Close** of Stock
7. **Volume** of Stock

The data changes based on **previous** values and other factors affecting companies' policies and follows some **trends** which can be **captured** using time series methods. Thus data used here is **Discrete** Time Series data.

# Objective

My objective here is to use multiple models on the dataset to perform analysis on the given dataset. The objective for my Time Series Data is:

1. Data **Preprocessing** and **Analysis**
2. Finding **Trends** in Data
3. Finding **Correlations** in Data
4. Finding if data is **stationary** or not
5. Checking **Seasonality**
6. **Smoothening** the data
7. Fitting all the models and **validating** their outputs
8. Which method is best for forecasting future values in data

## Data Cleaning

The data I am using is already **cleaned** and does not require any **imputation** or data handling. **Detrending** of data is required which can be done in the Data **Decomposition** section using various methods.

### ▼ JPM Dataset

```
[ ] series = pd.read_csv('JPM.csv', header=0, index_col=0)
series.head()
```

	Open	High	Low	Close	Adj Close	Volume
Date						
1980-03-17	0.0	5.129630	5.018519	5.037037	1.122904	62775
1980-03-18	0.0	5.111111	5.037037	5.074074	1.131161	64125
1980-03-19	0.0	5.166667	5.111111	5.148148	1.147674	40500
1980-03-20	0.0	5.148148	5.092593	5.111111	1.139418	18900
1980-03-21	0.0	5.222222	5.111111	5.222222	1.164188	97200

```
[ ] series.columns
```

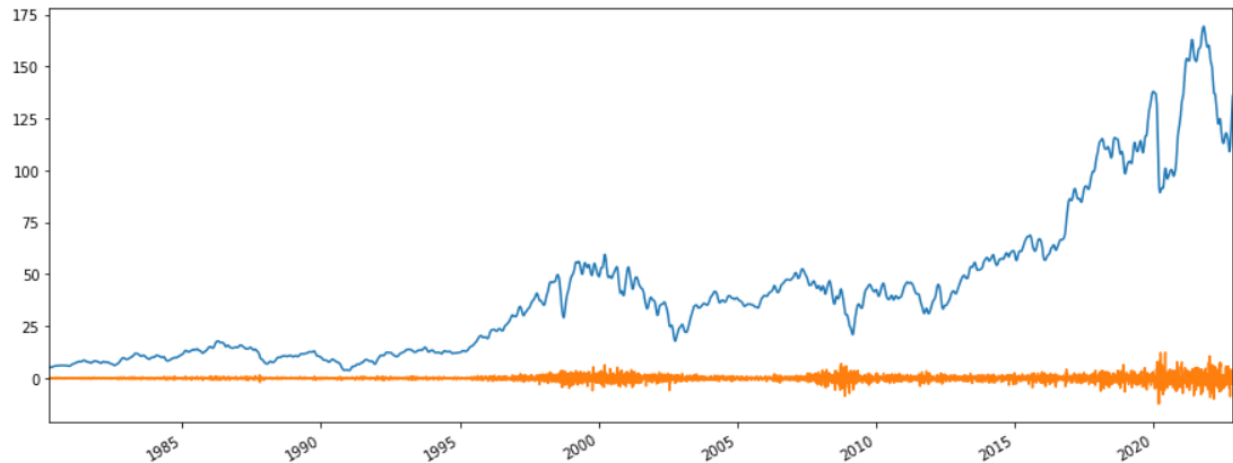
```
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

# Data Decomposition

Decomposition is a **statistical** job that involves breaking down Time Series data into many components or identifying **seasonality** and **trends** from a series of data.

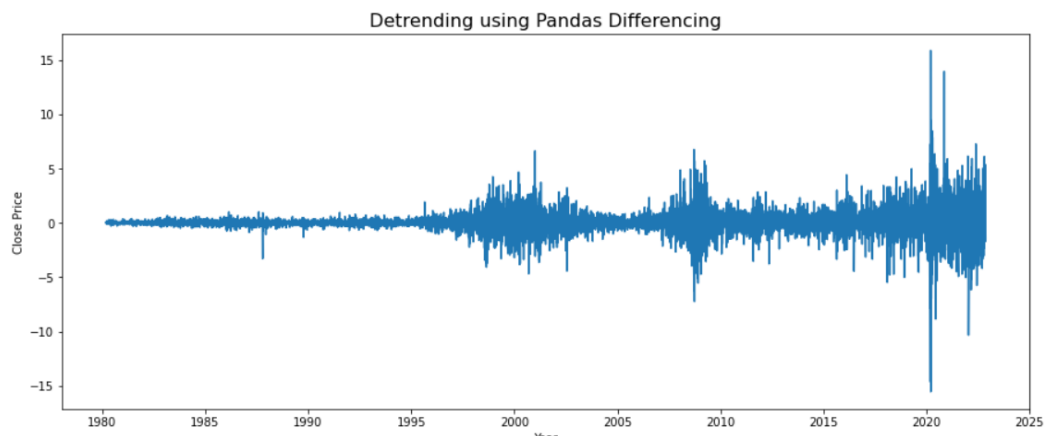
I can detect a trend using the **Hodrick-Prescott (HP)** Filter.

Trend on JPM Stock Data using **HP** Filter:

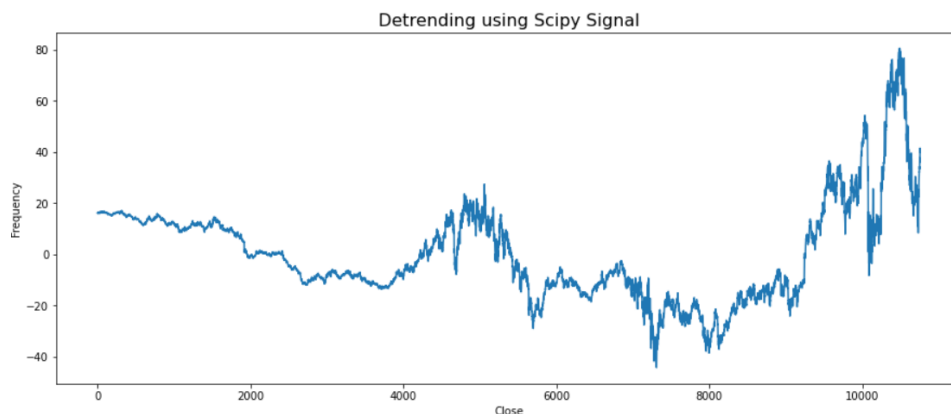


A trend from TS can be detrended using methods like

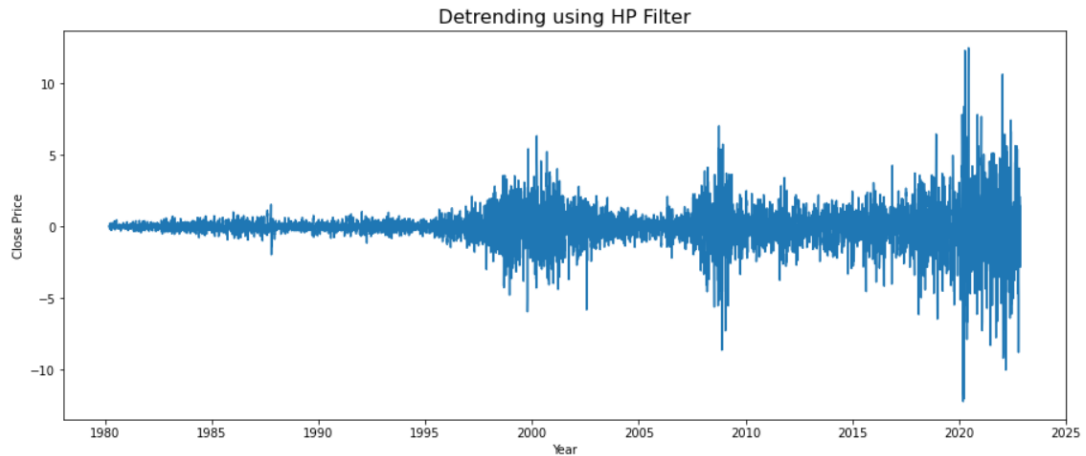
## 1. Pandas Differencing



## 2. SciPy Signal



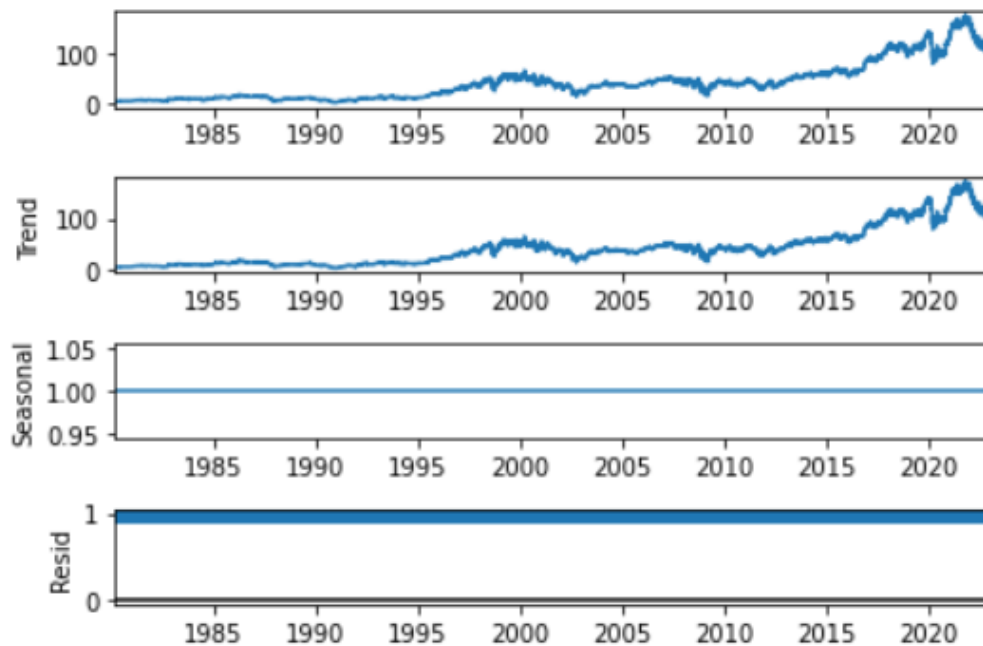
### 3. HP Filter



Time Series Data consists of 3 components:

1. **Trend**
2. **Seasonality**
3. **Residuals**

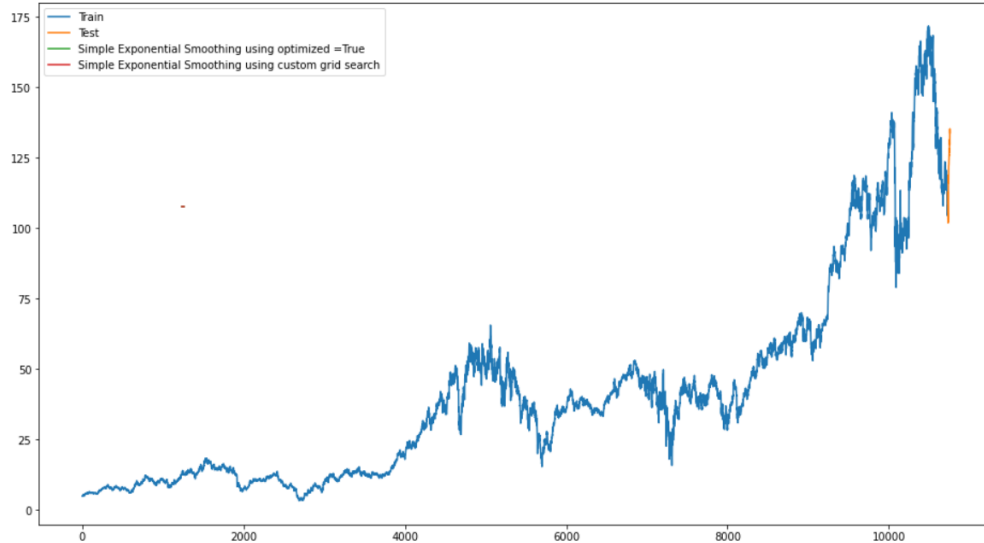
Data can be decomposed to find these components in data. This helps in Identifying seasonality & trends if present.



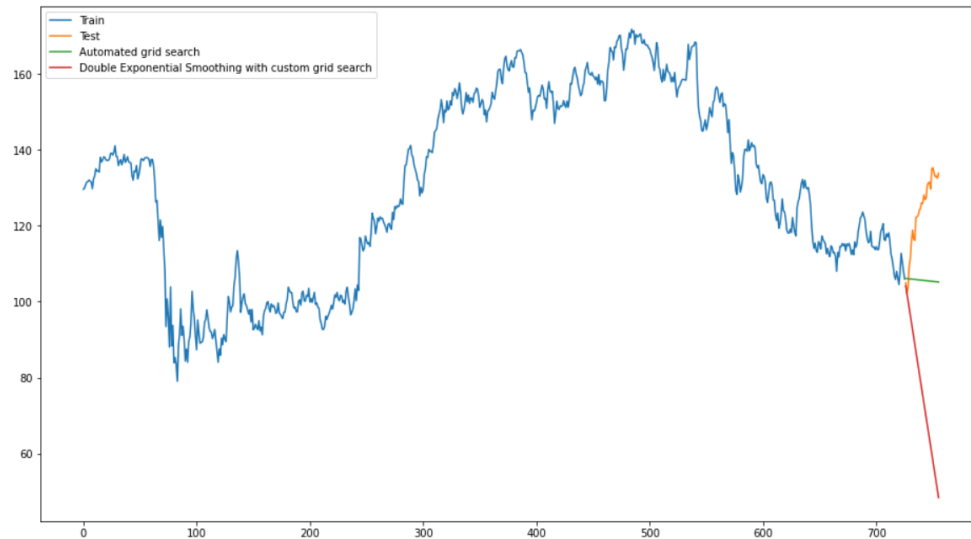
# Smoothing Methods

Smoothing is a statistical method I used to create an **approximation function** to **remove irregularities** in data and attempt to **capture significant patterns**. The smoothing technique is a family of time-series forecasting algorithms, which utilizes the **lighted averages** of a previous observation to **predict** or forecast a new value. Types of Smoothing Techniques are:

## 1. Simple Exponential Smoothing



## 2. Double Exponential Smoothing



# Testing Stationarity

When time-series data is **nonstationary**, it means it has **trends** and **seasonality patterns** that should be **removed**. By making data stationary, my data will have a **constant mean** and **variance**, and most of the statistical concepts can be applied. To **check** whether a time series is **stationary** or not, I used three methods.

1. Plots
2. Summary statistics
3. Statistics Tests

## Using Statistics **Unit Root** Tests

A statistical test makes a strong **assumption** about the data.

1. **Dickey-Fuller (DF) test**: This is based on **linear regression**. **Serial correlation** is a big issue of this method.
2. **Augmented Dickey-Fuller (ADF) test**: This solves the **serial correlation** problem of a DF test and handles big and **complex** models.

ADF test is conducted with the following **assumptions**:

1. **Null Hypothesis (Ho)**: Series is **non-stationary** or series has a unit root.
2. **Alternate Hypothesis(Ha)**: Series is **stationary** or series has no unit root.

Conditions to **Reject** Null Hypothesis(Ho):

If the **Test-statistic < Critical Value** and **p-value < 0.05** – **Reject** the Null Hypothesis(Ho) i.e., time series does not have a unit root, meaning it is stationary, otherwise.

```
def Augmented_Dickey_Fuller_Test_func(series , column_name):
    print (f'Results of Augmented Dickey-Fuller Test for column: {column_name}')
    dfctest = adfuller(series, autolag='AIC') #AIC: Method to use when automatically determining the lag length among the val
    #The t-value measures the size of the difference relative to the variation in your sample data.
    #T is simply the calculated difference represented in units of standard error.
    #The greater the magnitude of T, the greater the evidence against the null hypothesis.


    dfctest = pd.Series(dfctest[0:4], index=['Test Statistic','p-value','No. of Lags used','No. of Observations used'])
    for key,value in dfctest[4].items():
        dfctest['Critical Value (%s)'%key] = value #Critical values for the ADF test for 1%, 5%, and 10% significance levels
    print (dfctest)
    if dfctest[1] < 0.05:      # p-value =< 0.05
        print("Conclusion:====>")
        print("Null Hypothesis has been rejected.")
        print("The Data is Stationary.")
    else:
        print("Conclusion:====>")
        print("Failed to reject the Null Hypothesis.")
        print("Data is non-stationary.")
```


## On Non-Detrended Data:

```
[7] Augmented_Dickey_Fuller_Test_func(series['close'], 'close')
```

```
Results of Augmented Dickey-Fuller Test for column: Close
Test Statistic          -0.324347
p-value                 0.921985
No. of Lags used        39.000000
No. of Observations used 10719.000000
Critical Value (1%)     -3.430960
Critical Value (5%)     -2.861810
Critical Value (10%)    -2.566914
dtype: float64
Conclusion:====>
Failed to reject the Null Hypothesis.
Data is non-stationary.
```

## On Detrended Data:

```
 Augmented_Dickey_Fuller_Test_func(series['close'].diff().dropna(), '1')
```

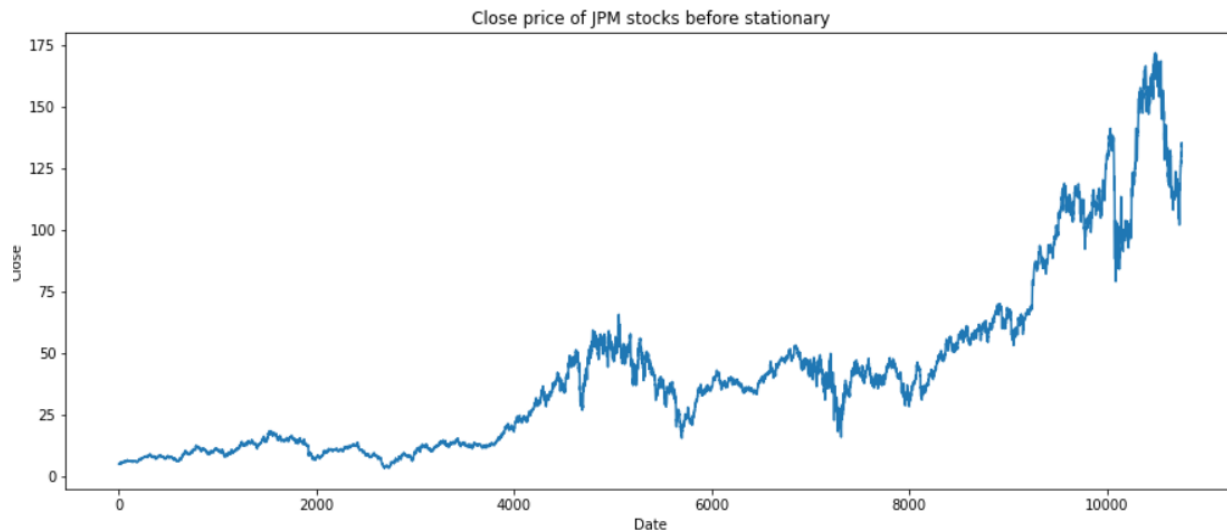
```
 Results of Augmented Dickey-Fuller Test for column: 1
Test Statistic          -1.551765e+01
p-value                 2.305185e-28
No. of Lags used        3.900000e+01
No. of Observations used 1.071800e+04
Critical Value (1%)     -3.430960e+00
Critical Value (5%)     -2.861810e+00
Critical Value (10%)    -2.566914e+00
dtype: float64
Conclusion:====>
Null Hypothesis has been rejected.
The Data is Stationary.
```

# Justification for Time Series

JP Morgan Stock Price Data is a Time Series Data because of the following reasons:

1. The data is **continuous** i.e data increases or decreases daily based
2. There is **seasonality** in some periods of data
3. Data can be **decomposed** into 3 components using seasonal decomposition
4. Rates of Stock are increasing with some **irregularities** based on factors affecting stock price and **market volatility**.





## Implementation and Interpretation of Forecast

From implementation, I can **interpret** that forecasts generated by models can **capture irregularity** in data and give somewhat **accurate predictions**.

Some methods implemented are

### 1. Autoregressive Integrated Moving Average (ARIMA) Model

Autoregressive integrated moving average—also called ARIMA(**p,d,q**)—is a forecasting equation that can make time series **stationary** with the help of **differencing** and **log** techniques when required. A time series that should be differentiated to be stationary is an **integrated** (d) (I) series. **Lags** of the stationary series are classified as **autoregressive** (p), which is designated in (AR) terms. **Lags** of the forecast **errors** are classified as **moving averages** (q), which are identified in (MA) terms.

### 2. Convolutional Neural Network (CNN) Model

I have seen examples of using CNN for **sequence prediction**. If I consider the **Dow Jones Industrial Average (DJIA)** as an example, I built a CNN with **1D convolution** for prediction. This makes sense because a 1D convolution on a time series is **roughly** computing its **moving average** or using **digital signal processing** terms, applying a filter to the time series. It should provide some clues about the trend.

### 3. Autoregressive Conditional Heteroskedasticity (ARCH) Model

Autoregressive models can be developed for **univariate** time-series data that is stationary (AR), has a trend (ARIMA), and has a **seasonal component** (**SARIMA**). However, these Autoregressive models do not model a change in the **variance** over time. The **error** terms in the **stochastic** processes generating the time series are **homoscedastic**, i.e. with constant variance. There are some time series where the variance changes consistently over

time. In the context of a time series in the **financial** domain, this would be called increasing and decreasing **volatility**.

#### 4. Generalized ARCH(GARCH) Model

Generalized Autoregressive Conditional Heteroskedasticity, or GARCH, is an **extension** of the ARCH model that incorporates a moving average component together with the autoregressive component.

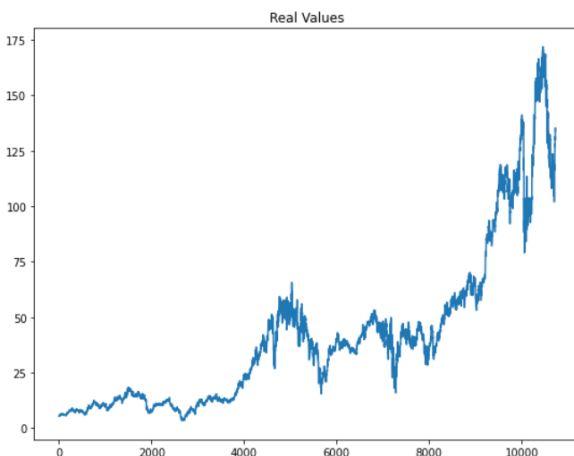
## Reason for Selecting the Model

For my dataset, I selected the **CNN** Model as it gives the best forecasts compared to other models like ARIMA, GARCH, ARCH, etc. Even though the data consists of financial stock prices CNN Model performs better than GARCH Model in this scenario. The **architecture** of used CNN Model

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(steps, features)))
model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())
model.add(Dense(100, activation="relu"))
model.add(Dense(1))
```

### Actual vs Predicted Forecast of Model



# Comparative Results Analysis

**Metrics** used for calculating **errors**:

1. MSE - Mean Squared Error
2. MAE - Mean Absolute Error
3. RMSE - Root Mean Squared Error
4. MAPE - Mean Absolute Percentage Error
5. R2 Score

From the output, it is evident that **CNN** performs way better compared to other algorithms for the given dataset. It can capture its trends and irregularities.

## **CNN Model Result:**

```
Evaluation metric results:-  
MSE is : 8.377951445922687  
MAE is : 1.9807257579852404  
RMSE is : 2.894469112967469  
MAPE is : 158.89644829338815  
R2 is : 0.993474059900318
```

## **GARCH Model Result:**

```
Evaluation metric results:-  
MSE is : 1514.5145875022415  
MAE is : 38.82838152072339  
RMSE is : 38.916764864287494  
MAPE is : 31.758735330487763  
R2 is : -14.401878805017143
```

## **ARIMA Model Result:**

```
Evaluation metric results:-  
MSE is : 254.64719636969028  
MAE is : 13.546780093520661  
RMSE is : 15.957668888960264  
MAPE is : 10.755199509648952  
R2 is : -1.5896384814567304
```

# Conclusion

After performing all the steps from preparing the data for implementation on the model to using various models like **ARCH, GARCH, and CNN** and using various error functions like **MSE, MAE, RMSE, MAPE**, and **R2** I concluded using **CNN** for my dataset.

The **loss function** used to compile my final model is **MSE** and the **optimizer** used is **'Adam'**

## My CNN Model:

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 29, 64)	192
max_pooling1d_1 (MaxPooling 1D)	(None, 14, 64)	0
flatten_1 (Flatten)	(None, 896)	0
dense_2 (Dense)	(None, 100)	89700
dense_3 (Dense)	(None, 1)	101
Total params: 89,993		
Trainable params: 89,993		
Non-trainable params: 0		

# Future Scope

I can use this method and model for further analysis of datasets related to finance or stock prices or any other materials with varying prices over time for example **petroleum, gold, silver**, etc. I can create a generalized **pipeline** consisting of all the steps involved for a time series analysis consisting of **Data preprocessing, Smoothing, Model Selection, Fitting, Comparison**, etc for any data by **auto-selecting** the best model for its **forecasting** and **analysis**.

# References

GARCH

[What Is the GARCH Process? How It's Used in Different Forms](#)

ARCH

[Autoregressive Conditional Heteroskedasticity \(ARCH\) Explained](#)

CNN

[Using CNN for financial time series prediction - MachineLearningMastery.com](#)

ADF

[Dickey-Fuller Test - an overview | ScienceDirect Topics](#)

Smoothing Methods

[Smoothing of time series | Statistical Software for Excel](#)

Loss functions

[Loss Functions in Time Series Forecasting](#)

Decomposition of data

[How to Decompose Time Series Data into Trend and Seasonality - MachineLearningMastery.com](#)