Basic understanding of Time Series Modelling with Auto ARIMAX

Siddharth M
Last Updated : 30 Nov, 2021

This article was published as a part of the Data Science Blogathon.

Introduction

Data Science associates with a huge variety of problems in our daily life. One major problem we see every day include examining a situation over time. Time series forecast is extensively used in various scenarios like sales, weather, prices, etc…, where the underlying values of concern are a range of data points estimated over a period of time. This article strives to provide the essential structure of some of the algorithms for solving these classes of problems. We will explore various methods for time series forecasts. We all would have heard about ARIMA models used in modern time series forecasts. In this article, we will thoroughly go through an understanding of ARIMA and how the Auto ARIMAX model can be used on a stock market dataset to forecast results.

Understanding ARIMA and Auto ARIMAX

Traditionally, everyone uses ARIMA when it comes to time series prediction. It stands for 'Auto-Regressive Integrated Moving Average', a set of models that defines a given time series based on its initial values, lags, and lagged forecast errors, so that equation is used to forecast forecasted values.
We have 'non-seasonal time series that manifests patterns and is not a stochastic white noise that can be molded with ARIMA models.
An ARIMA model is delineated by three terms: p, d, q where,

- p is a particular order of the AR term
- q is a specific order of the MA term
- d is the number of differences wanted to make the time series stationary

If a time series has seasonal patterns, then you require to add seasonal terms, and it converts to SARIMA, which stands for 'Seasonal ARIMA'.
The 'Auto Regressive' in ARIMA indicates a linear regression model that employs its lags as predictors. Linear regression models work best if the predictors are not correlated and remain independent of each other. We want to make them stationary, and the standard approach is to differentiate them. This means subtracting the initial value from the current value. Concerning how complex the series gets, more than one difference may be required.
Hence, the value of d is the merest number of differences necessitated to address the series stationary. In case we already have a stationary time series, we proceed with d as zero.

"Auto Regressive" (AR) term is indicated by "p". This relates to the number of lags of Y to be adopted as predictors. "Moving Average" (MA) term is associated with "q". This relates to the number of lagged prediction errors that should conform to the ARIMA Model.

New Feature

We use cookies essential for this site to function well. Please click to help us improve its usefulness with additional cookies. Learn about our use of cookies in our Privacy Policy & Cookies Policy.
Show details
Accept all cookies
Use necessary cookies

Create My Path

An Auto-Regressive (AR only) model has Yt that depends exclusively on its lags. Such, Yt is a function of the 'lags of Yt'.

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} + \epsilon_1$$

https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

Furthermore, a Moving Average (MA only) model has Yt that depends particularly on the lagged forecast errors.

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q}$$

https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

The time series differencing in an ARIMA model is differenced at least once to make sure it is stationary and we combine the AR and MA terms. Hence, The equation becomes:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q}$$

https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

We have continued operating through the method of manually fitting various models and determining which one is best. Therefore, we transpire to automate this process. It uses the data and fits several models in a different order before associating the characteristics. Nevertheless, the processing rate increases considerably when we seek to fit the complicated models. This is how we move for Auto-ARIMA models.

Implementation of Auto ARIMAX:

We will now look at a model called 'auto-arima', which is an auto_arima module from the pmdarima package. We can use pip install to install our module.

```
!pip install pmdarima
```

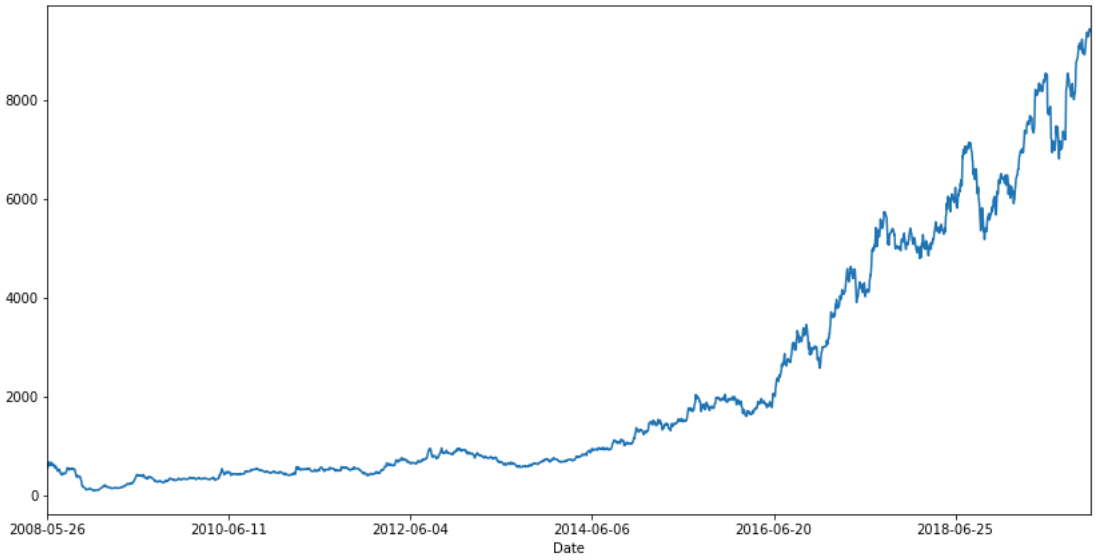
The dataset applied is stock market data of the Nifty-50 index of NSE (National Stock Exchange) India across the last twenty years. The well-known VWAP (Volume Weighted Average Price) is the target variable to foretell. VWAP is a trading benchmark used by tradesmen that supply the average price the stock has traded during the day, based on volume and price.

The dataset is available: https://www.kaggle.com/rohanrao/nifty50-stock-market-data.

```
df = pd.<a onclick="parent.postMessage({'referent':'.pandas.read_csv'}, '*')">read_csv("BAJAJFINSV.csv")
df.set_index("Date", drop=False, inplace=True)
df.head()
```

We use cookies essential for this site to function well. Please click to help us improve its usefulness with additional cookies. Learn about our use of cookies in our Privacy Policy & Cookies Policy.

Show details



Almost all time series problems will ought external characteristics or internal feature engineering to improve the model.

We add some essential features like lag values of available numeric features widely accepted for time series problems. Considering we need to foretell the stock price for a day, we cannot use the feature values of the same day since they will be unavailable at actual inference time. We require to use statistics like the mean, the standard deviation of their lagged values. The three sets of lagged values are used, one previous day, one looking back seven days and another looking back 30 days as a proxy for last week and last month metrics.

```
df.reset_index(drop=True, inplace=True)
lag_features = ["High", "Low", "Volume", "Turnover", "Trades"]
window1 = 3
window2 = 7
window3 = 30

df_rolled_3d = df[lag_features].rolling(window=window1, min_periods=0)
df_rolled_7d = df[lag_features].rolling(window=window2, min_periods=0)
df_rolled_30d = df[lag_features].rolling(window=window3, min_periods=0)

df_mean_3d = df_rolled_3d.mean().shift(1).reset_index().astype(np.<a onclick="parent.postMessage({'referent':'.numpy.float32'}
df_mean_7d = df_rolled_7d.mean().shift(1).reset_index().astype(np.<a onclick="parent.postMessage({'referent':'.numpy.float32'}
df_mean_30d = df_rolled_30d.mean().shift(1).reset_index().astype(np.<a onclick="parent.postMessage({'referent':'.numpy.float32

df_std_3d = df_rolled_3d.std().shift(1).reset_index().astype(np.<a onclick="parent.postMessage({'referent':'.numpy.float32'},
df_std_7d = df_rolled_7d.std().shift(1).reset_index().astype(np.<a onclick="parent.postMessage({'referent':'.numpy.float32'},
df_std_30d = df_rolled_30d.std().shift(1).reset_index().astype(np.<a onclick="parent.postMessage({'referent':'.numpy.float32'}

for feature in lag_features:
    df[f"{feature}_mean_lag{window1}"] = df_mean_3d[feature]
    df[f"{feature}_mean_lag{window2}"] = df_mean_7d[feature]
    df[f"{feature}_mean_lag{window3}"] = df_mean_30d[feature]

    df[f"{feature}_std_lag{window1}"] = df_std_3d[feature]
    df[f"{feature}_std_lag{window2}"] = df_std_7d[feature]
    df[f"{feature}_std_lag{window3}"] = df_std_30d[feature]

df.fillna(df.mean(), inplace=True)

df.set_index("Date", drop=False, inplace=True)
df.head()
```

We use cookies essential for this site to function well. Please click to help us improve its usefulness with additional cookies. Learn about our use of cookies in our Privacy Policy & Cookies Policy.

Show details

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	...	Turnover_mean_lag3
Date												
2008-05-26	2008-05-26	BAJAJFINSV	EQ	2101.05	600.00	619.00	501.0	505.1	509.10	548.85	...	4.316071e+13
2008-05-27	2008-05-27	BAJAJFINSV	EQ	509.10	505.00	610.95	491.1	564.0	554.65	572.15	...	1.726368e+14
2008-05-28	2008-05-28	BAJAJFINSV	EQ	554.65	564.00	665.60	564.0	643.0	640.95	618.37	...	2.107369e+14
2008-05-29	2008-05-29	BAJAJFINSV	EQ	640.95	656.65	703.00	608.0	634.5	632.40	659.60	...	2.350756e+14
2008-05-30	2008-05-30	BAJAJFINSV	EQ	632.40	642.40	668.00	588.3	647.0	644.00	636.41	...	2.508797e+14

During boosting models, it is very beneficial to attach DateTime features like an hour, day, month, as appropriate to implement the model knowledge about the time element in the data. For time sequence models, it is not explicitly expected to pass this information, but we could do so, and we will discuss in this article so that all models are analysed on the exact identical set of features.

```
df.Date = pd.<a onclick="parent.postMessage({'referent':'.pandas.to_datetime'}, '*')">to_datetime(df.Date, format="%Y-%m-%d")
df["month"] = df.Date.dt.month
df["week"] = df.Date.dt.week
df["day"] = df.Date.dt.day
df["day_of_week"] = df.Date.dt.dayofweek
df.head()
```

The data is split in both train and test along with its features.

train: We have 26th May 2008 to 31st December 2018 data.
valid: We have 1st January 2019 to 31st December 2019 data.

```
df_train = df[df.Date < "2019"]
df_valid = df[df.Date >= "2019"]

exogenous_features = ["High_mean_lag3", "High_std_lag3", "Low_mean_lag3", "Low_std_lag3",
                      "Volume_mean_lag3", "Volume_std_lag3", "Turnover_mean_lag3",
                      "Turnover_std_lag3", "Trades_mean_lag3", "Trades_std_lag3",
                      "High_mean_lag7", "High_std_lag7", "Low_mean_lag7", "Low_std_lag7",
                      "Volume_mean_lag7", "Volume_std_lag7", "Turnover_mean_lag7",
                      "Turnover_std_lag7", "Trades_mean_lag7", "Trades_std_lag7",
                      "High_mean_lag30", "High_std_lag30", "Low_mean_lag30", "Low_std_lag30",
                      "Volume_mean_lag30", "Volume_std_lag30", "Turnover_mean_lag30",
                      "Turnover_std_lag30", "Trades_mean_lag30", "Trades_std_lag30",
                      "month", "week", "day", "day_of_week"]

model = auto_arima(df_train.VWAP, exogenous=df_train[exogenous_features], trace=True, error_action="ignore", suppress_warnings
model.fit(df_train.VWAP, exogenous=df_train[exogenous_features])

forecast = model.predict(n_periods=len(df_valid), exogenous=df_valid[exogenous_features])
df_valid["Forecast_ARIMAX"] = forecast
```

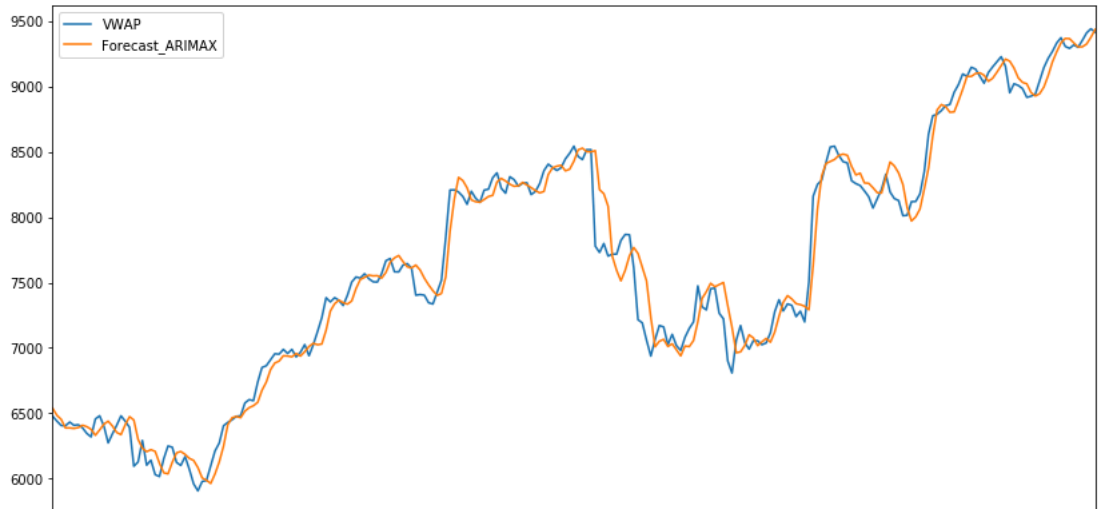
We use cookies essential for this site to function well. Please click to help us improve its usefulness with additional cookies. Learn about our use of cookies in our Privacy Policy & Cookies Policy.

Show details

```
Fit ARIMA: order=(2, 0, 2) seasonal_order=(0, 0, 0, 1); AIC=28695.940, BIC=28930.884, Fit ti
me=7.984 seconds
Fit ARIMA: order=(0, 0, 0) seasonal_order=(0, 0, 0, 1); AIC=29360.427, BIC=29571.876, Fit ti
me=4.414 seconds
Fit ARIMA: order=(1, 0, 0) seasonal_order=(0, 0, 0, 1); AIC=28786.662, BIC=29003.985, Fit ti
me=4.477 seconds
Fit ARIMA: order=(0, 0, 1) seasonal_order=(0, 0, 0, 1); AIC=29047.691, BIC=29265.014, Fit ti
me=7.146 seconds
Fit ARIMA: order=(0, 0, 0) seasonal_order=(0, 0, 0, 1); AIC=47416.468, BIC=47622.044, Fit ti
me=4.194 seconds
Fit ARIMA: order=(1, 0, 2) seasonal_order=(0, 0, 0, 1); AIC=28697.801, BIC=28926.871, Fit ti
me=7.625 seconds
Fit ARIMA: order=(2, 0, 1) seasonal_order=(0, 0, 0, 1); AIC=28693.883, BIC=28922.953, Fit ti
me=7.411 seconds
Fit ARIMA: order=(1, 0, 1) seasonal_order=(0, 0, 0, 1); AIC=28770.201, BIC=28993.398, Fit ti
me=7.153 seconds
Fit ARIMA: order=(2, 0, 0) seasonal_order=(0, 0, 0, 1); AIC=28702.843, BIC=28926.040, Fit ti
me=7.205 seconds
Fit ARIMA: order=(3, 0, 1) seasonal_order=(0, 0, 0, 1); AIC=28708.195, BIC=28943.138, Fit ti
me=7.804 seconds
Fit ARIMA: order=(3, 0, 0) seasonal_order=(0, 0, 0, 1); AIC=28694.776, BIC=28923.846, Fit ti
me=7.969 seconds
Fit ARIMA: order=(3, 0, 2) seasonal_order=(0, 0, 0, 1); AIC=30584.152, BIC=30824.969, Fit ti
me=7.986 seconds
Total fit time: 81.428 seconds
```

The most suitable ARIMA model is ARIMA(2, 0, 1) which holds the lowest AIC.

```
df_valid[["VWAP", "Forecast_ARIMAX"]].plot(figsize=(14, 7))
```



We use cookies essential for this site to function well. Please click to help us improve its usefulness with additional cookies. Learn about our use of cookies in our Privacy Policy & Cookies Policy.
Show details

```
RMSE of Auto ARIMAX: 147.0863858907692

MAE of Auto ARIMAX: 104.01942184247048
```

Conclusion:

In this article, we explored details regarding ARIMA and understood how auto ARIMAX was applied to a time series dataset. We implemented the model and got a score of about 147.086 as RMSE and 104.019 as MAE as the final result.

Reference:

1. https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/
2. https://www.kaggle.com/rohanrao/a-modern-time-series-tutorial/notebook

About Me: I am a Research Student interested in the field of Deep Learning and Natural Language Processing and currently pursuing post-graduation in Artificial Intelligence.

Image Source

1. Preview Image: https://medium.com/@fenjiro/time-series-for-business-a-general-introduction-50968346e660

Feel free to connect with me on:

1. Linkedin: https://www.linkedin.com/in/siddharth-m-426a9614a/
2. Github: https://github.com/Siddharth1698

The media shown in this article is not owned by Analytics Vidhya and are used at the Author's discretion

---

Siddharth M

Passionate about artificial intelligence, I am dedicated to advancing research in Generative AI and Large Language Models (LLMs). My work focuses on exploring innovative solutions and pushing the boundaries of what's possible in this dynamic and transformative field.

Time Series | Time Series Forecasting | Uncategorized

Free Courses

We use cookies essential for this site to function well. Please click to help us improve its usefulness with additional cookies. Learn about our use of cookies in our Privacy Policy & Cookies Policy.

Show details

Generative AI - A Way of Life

Explore Generative AI for beginners: create text and images, use top AI tools, learn practical skills, and ethics.

Getting Started with Large Language Models

Master Large Language Models (LLMs) with this course, offering clear guidance in NLP and model training made simple.

Building LLM Applications using Prompt Engineering

This free course guides you on building LLM apps, mastering prompt engineering, and developing chatbots with enterprise data.

Improving Real World RAG Systems: Key Challenges & Practical Solutions

Explore practical solutions, advanced retrieval strategies, and agentic RAG systems to improve context, relevance, and accuracy in AI-driven applications.

Microsoft Excel: Formulas & Functions

Master MS Excel for data analysis with key formulas, functions, and LookUp tools in this comprehensive course.

RECOMMENDED ARTICLES

Multivariate Time Series Analysis

Time Series Analysis: Definition, Components, M...

Build High Performance Time Series Models using...

Stock Market Price Trend Prediction Using Time ...

We use cookies essential for this site to function well. Please click to help us improve its usefulness with additional cookies. Learn about our use of cookies in our Privacy Policy & Cookies Policy.

Show details

Performing Time Series Analysis using ARIMA Mod...

Introduction to Time series Modeling With -ARIMA

10 Pandas One-Liners for Data Cleaning

Building an ARIMA Model for Time Series Forecas...

Responses From Readers

What are your thoughts?...

We use cookies essential for this site to function well. Please click to help us improve its usefulness with additional cookies. Learn about our use of cookies in our Privacy Policy & Cookies Policy.

Show details

tv schedule online

This is a great post! I have been working with time series data for a while now and this post has really helped me to understand the concepts better.

Write for us

Write, captivate, and earn accolades and rewards for your work

- Reach a Global Audience
- Get Expert Feedback
- Build Your Brand & Audience
- Cash In on Your Knowledge
- Join a Thriving Community
- Level Up Your Data Science Game

We use cookies essential for this site to function well. Please click to help us improve its usefulness with additional cookies. Learn about our use of cookies in our Privacy Policy & Cookies Policy.

Show details

GenAI Pinnacle Program | GenAI Pinnacle Plus Program | AI/ML BlackBelt Program | Agentic AI Pioneer Program

Free Courses

Generative AI | DeepSeek | OpenAI Agent SDK | LLM Applications using Prompt Engineering | DeepSeek from Scratch | Stability.AI | SSM & MAMBA | RAG Systems using LlamaIndex | Building LLMs for Code | Python | Microsoft Excel | Machine Learning | Deep Learning | Mastering Multimodal RAG | Introduction to Transformer Model | Bagging & Boosting | Loan Prediction | Time Series Forecasting | Tableau | Business Analytics | Vibe Coding in Windsurf | Model Deployment using FastAPI | Building Data Analyst AI Agent | Getting started with OpenAI o3-mini | Introduction to Transformers and Attention Mechanisms

Popular Categories

AI Agents | Generative AI | Prompt Engineering | Generative AI Application | News | Technical Guides | AI Tools | Interview Preparation | Research Papers | Success Stories | Quiz | Use Cases | Listicles

Generative AI Tools and Techniques

GANs | VAEs | Transformers | StyleGAN | Pix2Pix | Autoencoders | GPT | BERT | Word2Vec | LSTM | Attention Mechanisms | Diffusion Models | LLMs | SLMs | Encoder Decoder Models | Prompt Engineering | LangChain | LlamaIndex | RAG | Fine-tuning | LangChain AI Agent | Multimodal Models | RNNs | DCGAN | ProGAN | Text-to-Image Models | DDPM | Document Question Answering | Imagen | T5 (Text-to-Text Transfer Transformer) | Seq2seq Models | WaveNet | Attention Is All You Need (Transformer Architecture) | WindSurf | Cursor

Popular GenAI Models

Llama 4 | Llama 3.1 | GPT 4.5 | GPT 4.1 | GPT 4o | o3-mini | Sora | DeepSeek R1 | DeepSeek V3 | Janus Pro | Veo 2 | Gemini 2.5 Pro | Gemini 2.0 | Gemma 3 | Claude Sonnet 3.7 | Claude 3.5 Sonnet | Phi 4 | Phi 3.5 | Mistral Small 3.1 | Mistral NeMo | Mistral-7b | Bedrock | Vertex AI | Qwen QwQ 32B | Qwen 2 | Qwen 2.5 VL | Qwen Chat | Grok 3

AI Development Frameworks

n8n | LangChain | Agent SDK | A2A by Google | SmolAgents | LangGraph | CrewAI | Agno | LangFlow | AutoGen | LlamaIndex | Swarm | AutoGPT

Data Science Tools and Techniques

Python | R | SQL | Jupyter Notebooks | TensorFlow | Scikit-learn | PyTorch | Tableau | Apache Spark | Matplotlib | Seaborn | Pandas | Hadoop | Docker | Git | Keras | Apache Kafka | AWS | NLP | Random Forest | Computer Vision | Data Visualization | Data Exploration | Big Data | Common Machine Learning Algorithms | Machine Learning | Google Data Science Agent

Company

About Us

Contact Us

Careers

Discover

Blogs

Expert Sessions

Learning Paths

Comprehensive Guides

Learn

Engage

We use cookies essential for this site to function well. Please click to help us improve its usefulness with additional cookies. Learn about our use of cookies in our Privacy Policy & Cookies Policy.

Show details

Agentic AI Program

Podcasts

Contribute

Become an Author

Become a Speaker

Become a Mentor

Become an Instructor

Enterprise

Our Offerings

Trainings

Data Culture

AI Newsletter

Terms & conditions • Refund Policy • Privacy Policy • Cookies Policy © Analytics Vidhya 2025.All rights reserved.

We use cookies essential for this site to function well. Please click to help us improve its usefulness with additional cookies. Learn about our use of cookies in our Privacy Policy & Cookies Policy.

Show details