

A conceptual image of a road stretching into the distance towards a bright horizon. The road surface is marked with the years 2023, 2024, 2025, 2026, and 2027 in large, white, 3D-style letters. The road is flanked by dark, silhouetted hills under a blue sky with scattered white clouds. The sun is low on the horizon, creating a strong lens flare and illuminating the scene.

Hackathon '25

PS – 1: Time Series Forecasting

Contents:

1. Introduction
2. Data Loading and Preprocessing
3. Exploratory Data Analysis (EDA)
4. Stationarity Check
5. Statistical & Machine Learning Models
6. Forecasting
7. Performance Evaluation
8. Visualization
9. Web Application (Flask Integration)

1.Introduction:

The code presented here demonstrates a practical approach to handling time series data retrieved using REST API (yfinance.api), including data preprocessing, stationarity checks, data transformations, model selection, model building, evaluation, and forecasting. This documentation aims to provide a clear understanding of each step involved in the process, along with the underlying concepts and techniques.

Tools and Framework

PROGRAMMING LANGUAGE

- PYTHON

FRAMEWORK

- FLASK

LIBRARIES

- PANDAS
- NUMPY
- YFINANCE
- MATPLOTLIB
- STATSMODELS
- ARCH
- SCIKIT-LEARN

2.Data Loading and Preprocessing:

Stock price data is fetched using the Yahoo Finance API. The script allows users to specify a stock of company (eg:Amazon,Google) and a time range to collect historical prices. Once the data is retrieved, preprocessing steps are applied to clean and format it.

Key Preprocessing Steps:

- Handling missing values by dropping or filling them appropriately.
- Converting dates to the correct format for time series analysis.
- Sorting data to maintain a consistent chronological order.

The function **'fetch_data(ticker, period='1y')'** is responsible for retrieving stock prices, and **'preprocess_data(df)'** ensures the dataset is clean and ready for analysis.

3.Exploratory Data Analysis (EDA):

To understand the behavior of stock prices, various analytical techniques are applied. These include:

- Time Series Visualization: Plotting closing prices over time to observe trends.
- Seasonality Decomposition: Using **Statsmodels** **'seasonal_decompose()'** function to break down stock prices into trend, seasonality, and residuals..

The function **'plot_data(df)'** generates the initial stock price trend graph, while **'plot_decomposition(df)'** visualizes the seasonal decomposition.

4.Stationarity Check:

Using the Augmented Dickey-Fuller (ADF) test to determine if the data is stationary, which is crucial for ARIMA-based models. Performing the ADF test using **'adfuller()'** to statistically confirm stationarity. The ADF test checks for the presence of a unit root, which indicates non-stationarity. The null hypothesis is that the series is non-stationary.

The function **'check_stationarity(df)'** is responsible for checking the stationarity of time series data.

5.Statistical & Machine Learning Models

The core forecasting component involves training three different models on stock price data:

ARIMA (Auto Regressive Integrated Moving Average)

- ARIMA is a classical time-series forecasting model that considers past values and past errors.
- It requires the dataset to be stationary, which is ensured using the **ADF test**.
- The model is trained using **ARIMA(df['Close'], order=(1, 1, 1))**, where: (1,1,1) represents the **AutoRegressive (AR), Differencing (I), and Moving Average (MA) components**.
- Once trained, predictions are made using the **'get_forecast(steps=n_days)'** function.

SARIMA (Seasonal ARIMA)

- SARIMA extends ARIMA by accounting for seasonal patterns in the data.
- It adds a seasonal component (P, D, Q, s) to the ARIMA model.
- The implementation uses **SARIMAX(df['Close'], order=(1,1,1), seasonal_order=(1,1,1,12))**, where **12** represents a 12-month seasonal cycle.

These models are trained within the **'train_models(df)'** function, which returns trained instances of ARIMA and SARIMA.

6.Forecasting:

After training, the models generate future predictions for a specified number of days (**n_days**). The function **'forecast_future(df, models, steps=n_days)'** computes predictions along with confidence intervals:

- ARIMA and SARIMA use the **'get_forecast()'** method to generate price predictions and confidence bands.

To visualize forecasts, the function `'plot_forecast(df, forecast_values, conf_int, n_days)'` plots:

- Actual stock prices.
- Predicted prices for the next `n_days`.
- Confidence intervals to indicate uncertainty in predictions.

7. Performance Evaluation:

To validate model performance, predictions are compared with actual stock prices using standard error metrics:

- **Mean Squared Error (MSE):** Measures average squared errors.
- **Root Mean Squared Error (RMSE):** Square root of MSE, showing error magnitude.
- **Mean Absolute Error (MAE):** Measures average absolute errors.

These are computed in the `'predict()'` route, ensuring users receive both forecasts and accuracy reports.

8. Visualization

various types of visualizations are used to enhance the interpretability of stock price data and forecasting results. The primary visualization techniques used include time-series plots, seasonal decomposition, autocorrelation analysis, and forecasting plots.

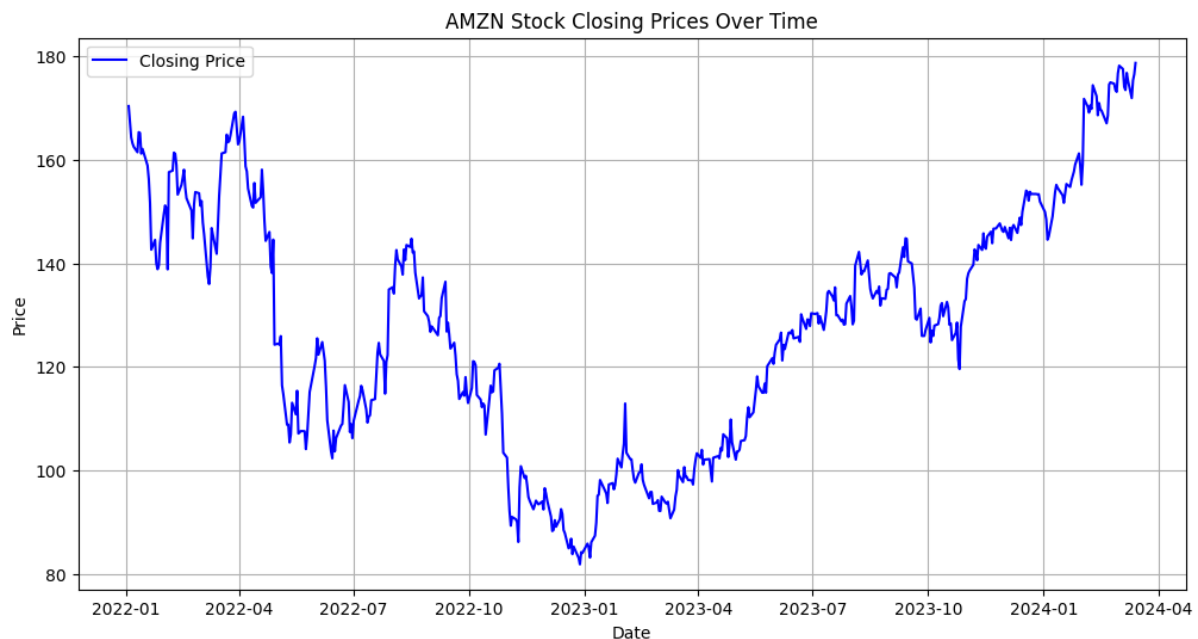
- **Time-Series Visualization (Stock Price Trend Analysis)**

One of the most basic yet effective visualizations is plotting the closing prices over time. This allows users to see how the stock has performed over a given period.

Insights Gained

- Identifies **upward or downward trends** in stock prices.
- Highlights **sharp price fluctuations**, which might indicate market volatility.

- Helps traders and analysts make **buy/sell decisions** based on past performance.



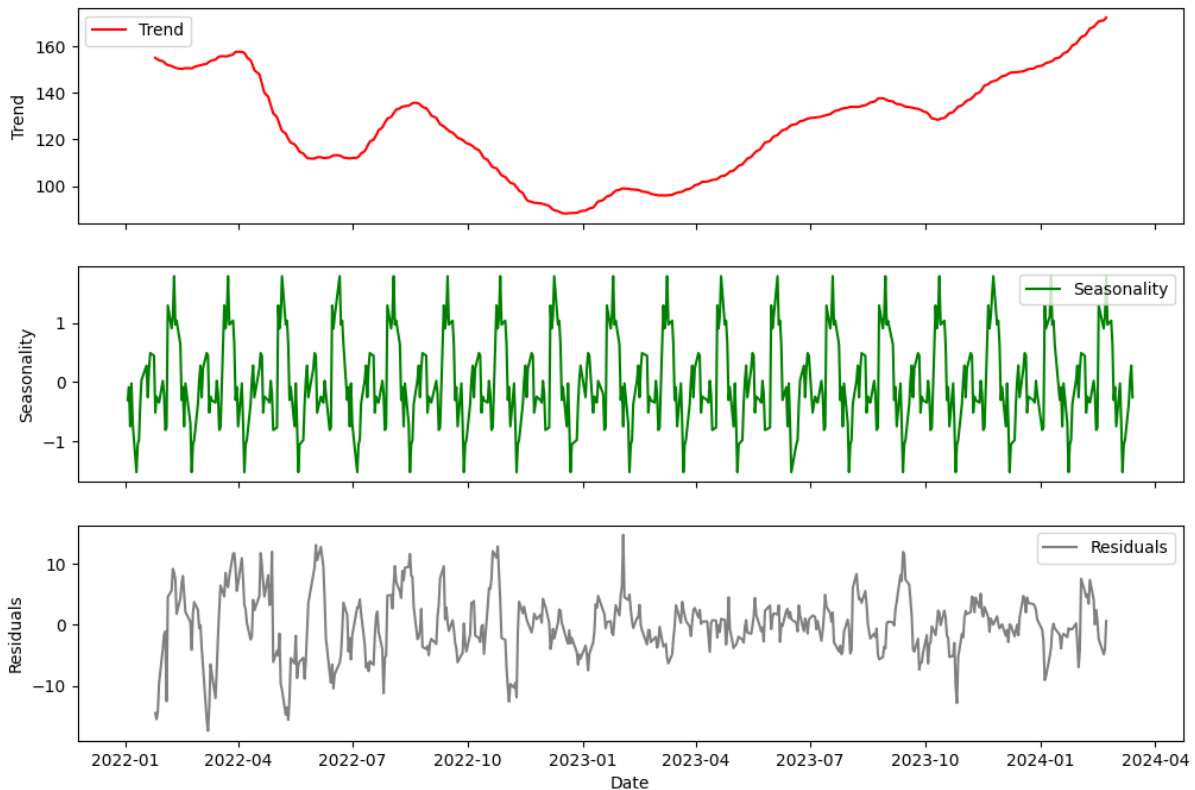
- **Seasonal Decomposition (Trend, Seasonality, Residuals)**

Stock prices often exhibit **seasonal patterns**, meaning they tend to rise and fall periodically. To break down these components, we use **seasonal decomposition** via the '**seasonal_decompose()**' function.

Insights Gained

- **Trend Analysis:** Detects long-term stock price movements.
- **Seasonality Identification:** Shows repeating patterns in stock prices (e.g., quarterly earnings impact).
- **Noise Detection:** Helps filter out random fluctuations that do not contribute to forecasting.

AMZN Time Series Decomposition



- **Autocorrelation and Partial Autocorrelation Analysis:**

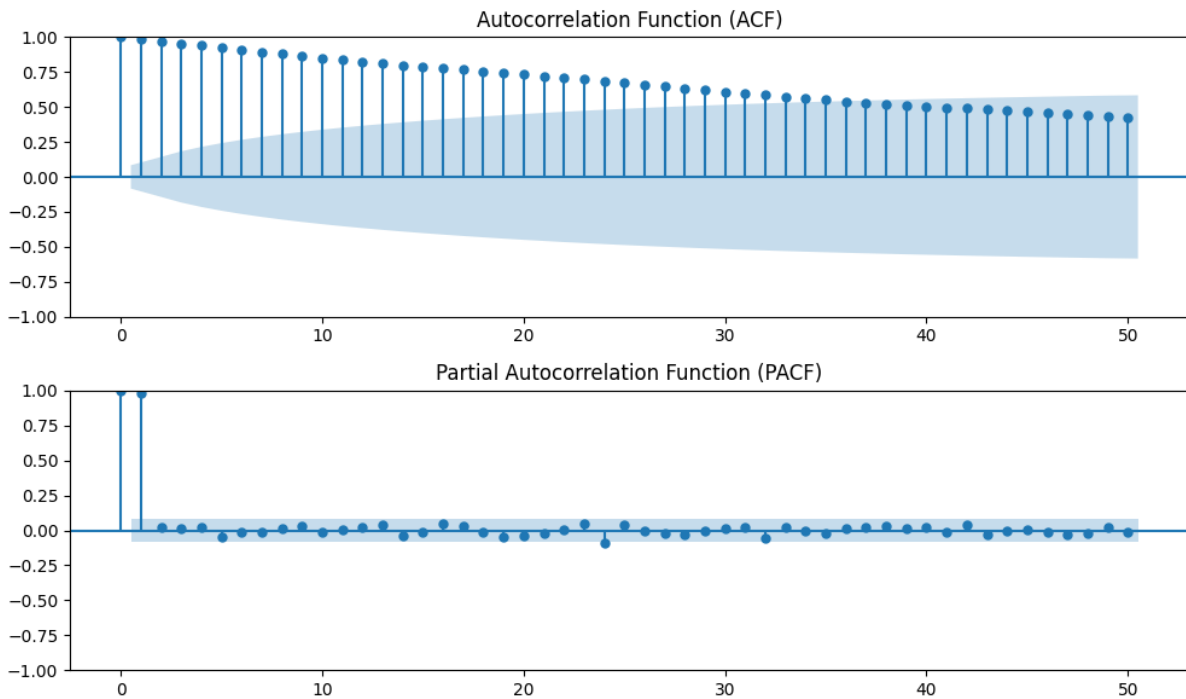
Before training ARIMA and SARIMA models, it is important to analyze how stock prices correlate with past values. Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) help in choosing the right model parameters.

ACF vs. PACF

- **ACF (Autocorrelation Function):** Measures how past values influence the present value.
- **PACF (Partial Autocorrelation Function):** Identifies direct relationships between past values and present values, eliminating indirect effects.

Insights Gained

- Helps determine the "**p**" and "**q**" parameters in ARIMA models.
- Identifies if the stock prices follow an **AR (AutoRegressive)** or **MA (Moving Average)** process.
- Detects periodic trends and dependencies over time.



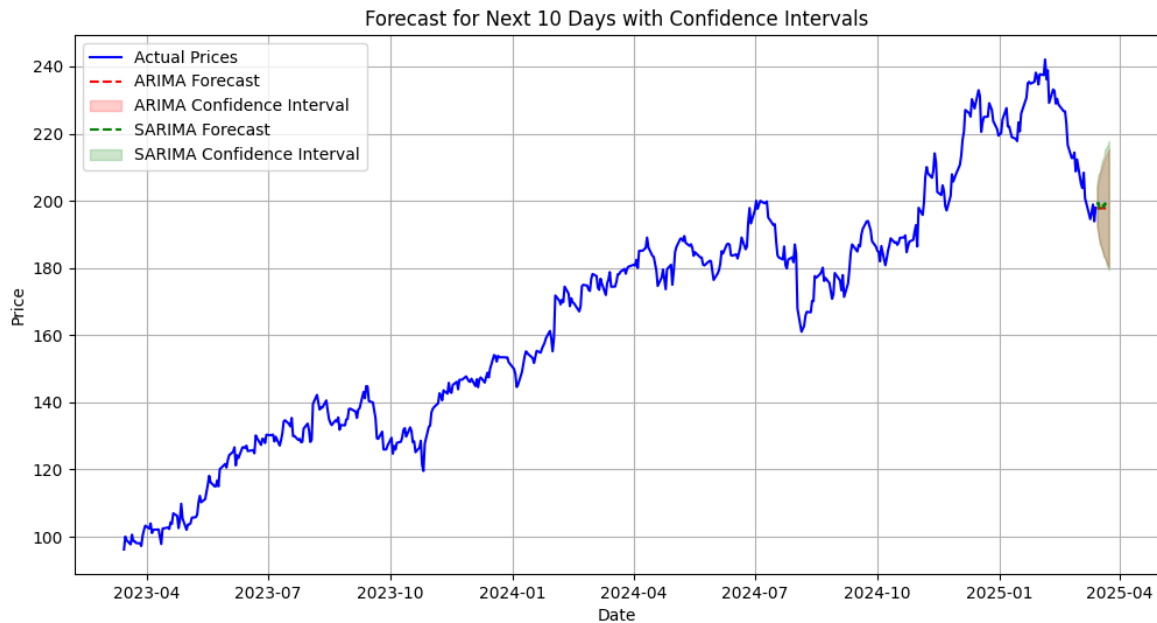
- **Forecasting Plot (Actual vs. Predicted Prices)**

After training ARIMA and SARIMA models, their predictions must be **compared against actual stock prices**. This is visualized using a **forecasting plot**, which includes:

- **Actual stock prices** (historical data).
- **Predicted future prices** (next n_days).
- **Confidence intervals** (range of uncertainty in predictions).

Insights Gained

- Visualizes model performance against actual stock prices.
- Shows **forecast uncertainty** via confidence intervals.
- Allows comparison between **ARIMA and SARIMA predictions**.



9.Web Application (Flask Integration):

Flask-based web application provides an interactive interface for users to input a stock ticker, select a time period, and view forecasts.

Application Structure


The application consists of the following files:


- app.py – The main Flask application.
- util.py – Helper functions for fetching, preprocessing, and modeling data.
- templates/ – HTML templates for the web UI.
- static/ – Stores images of plots for visualization.

Routes and Functionality

The app defines two key routes:

- **Homepage (/)**
 - Displays an HTML form for users to enter a stock ticker, time period, and forecast duration.
 - Once submitted, it calls the /forecast route.


ASTROLYTICS
 AI-powered time-series forecasting


Astrolytics

Company Name:

Enter Period (e.g., 1y):

Forecast Days:

Choose Model:

ARIMA
▼

Get Forecast

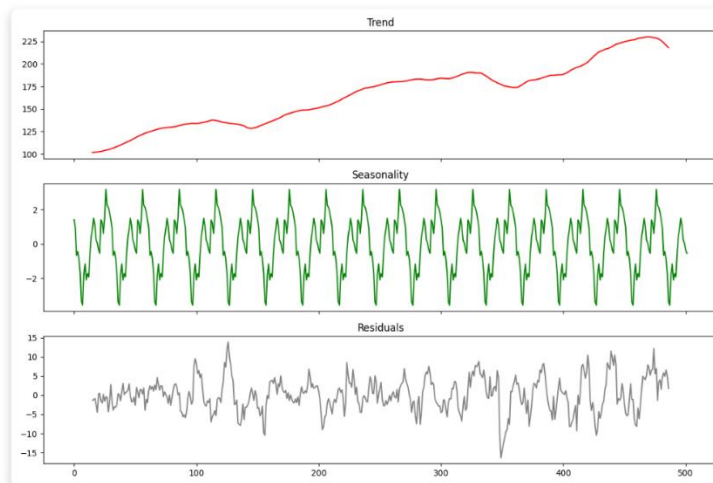
© 2025 Astrolytics. All Rights Reserved.

- **Forecast Page (/forecast)**

- Fetches stock data, preprocesses it, and generates visualizations.
- Calls ARIMA, SARIMA, and GARCH models to make predictions.
- Displays trend analysis, seasonality decomposition, and forecasts.
- Presents confidence intervals for predictions.



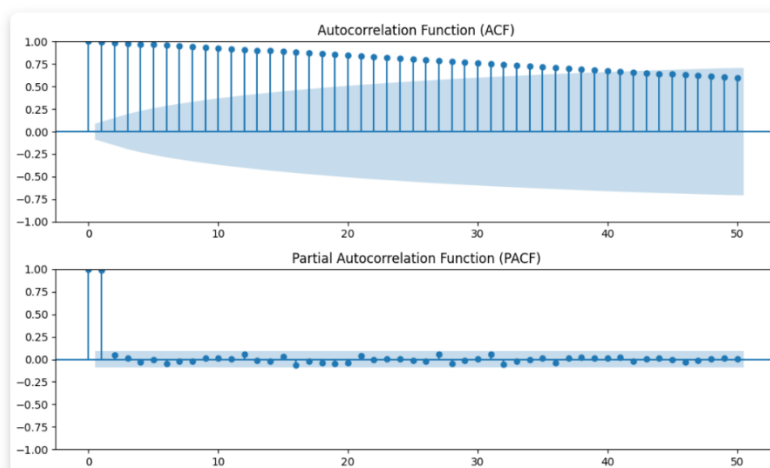
2 Time-Series Decomposition



🔍 This decomposition splits the time series into three components:

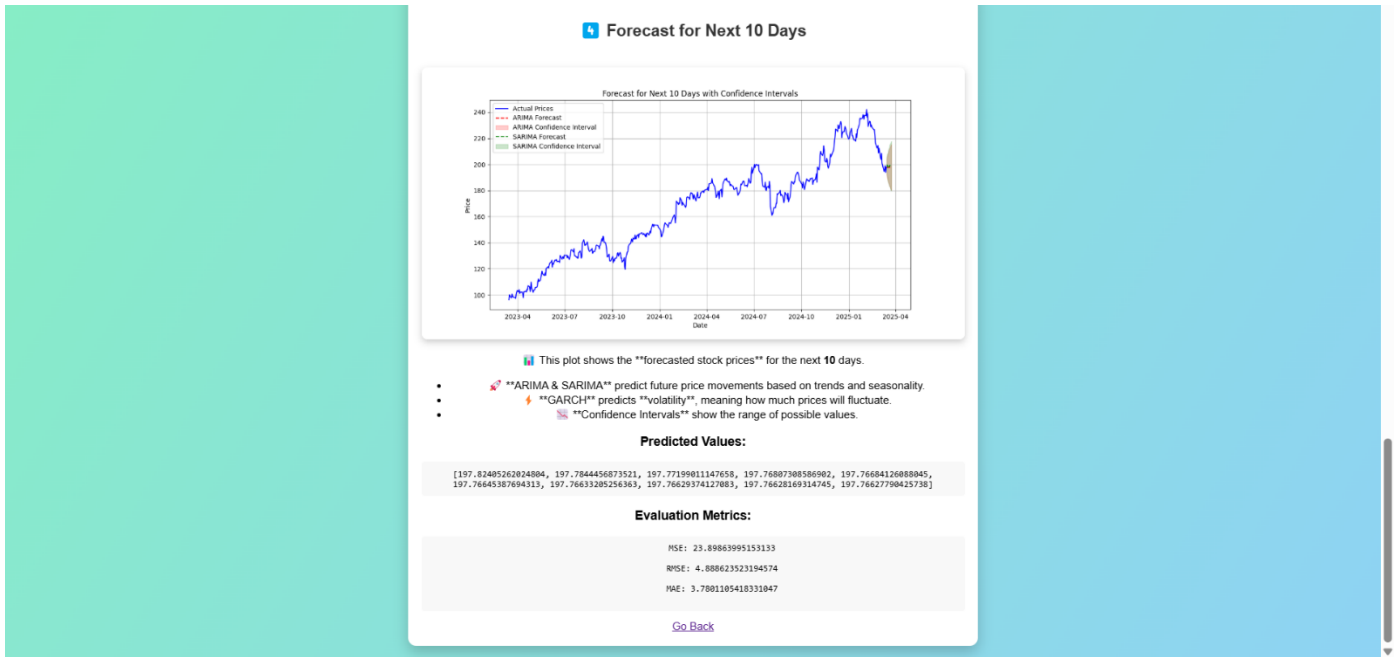
- 📈 **Trend:** Long-term movement.
- 📊 **Seasonality:** Repeating patterns at regular intervals.
- ⚡ **Residuals:** Unexplained variations (random noise).

3 Autocorrelation & Partial Autocorrelation



🔗 These plots help determine how past values influence future values:

- 📈 ****ACF (Autocorrelation Function)**:** Shows how past values correlate with future values.
- 📊 ****PACF (Partial Autocorrelation Function)**:** Helps determine ARIMA parameters (p, q).



The web app also includes a separate API route, /predict, which returns forecasted values in JSON format along with evaluation metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE).

Team Name: Tri-State

Team Members Name & Roll Number:

1.Dhiyanesh B - 23PD08

2.Parikshit V - 23PD24

3.Ramvignesh R - 23PD31

Email-Id:

1.Dhiyanesh B - 23pd08@psgtech.ac.in

2.Parikshit V - 23pd24@psgtech.ac.in

3.Ramvignesh R - 23pd31@psgtech.ac.in