Trading Strategies

Riccardo Pondini

January 2024

Introduction

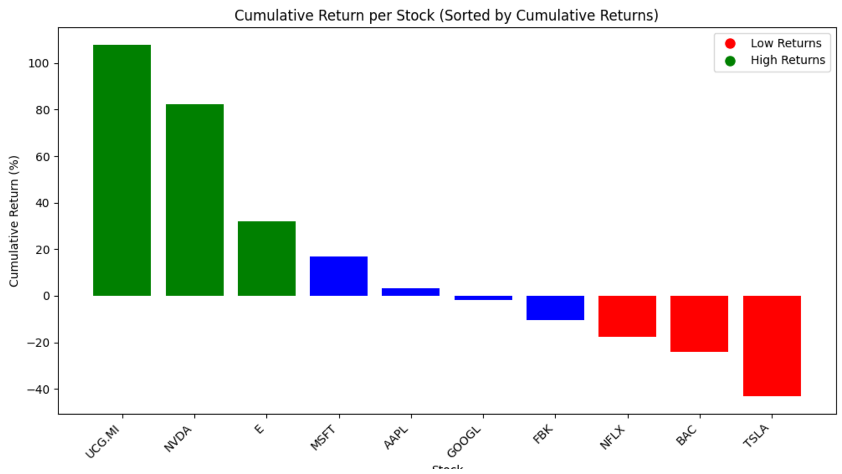
Within the expansive domain of quantitative trading, Kakushadze and Serur's "151 Trading Strategies" present a panorama of diverse tactics. This report meticulously analyzes selected strategies, each embodying a distinct facet in the intricate fabric of financial markets. The examination focuses on the practicality, mathematical coherence, and real-world adaptability of these chosen strategies, shedding light on the nuanced landscape of quantitative trading.

Stocks

There are a lot of strategies in the field of stocks, but the majority are the ones related to a single stock at time.

Price-momentum

This strategy is about the assumption that empirically there appears to be certain "inertia" in stock returns known as the momentum effect, whereby future returns are positively correlated with past returns. Momentum trading is a strategy that aims to capitalize on the continuance of existing trends in the market. Momentum traders usually buy or sell an asset moving intensely in one direction and exiting when this movement shows signs of reversing. They also seek to avoid buying or selling assets that are moving sideways. In the next example, the cumulative returns of several stocks are computed for the last two years. They are then sorted to identify the top quartile (stocks that should have been bought) and the bottom quartile (the short signal of the strategy). Finally, a histogram is plotted.



Code

```python
import pandas as pd
import numpy as np
import datetime as dt
from pandas_datareader import data as pdr
import matplotlib.pyplot as plt
import yfinance as yf
import math

# Set pandas display options
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# Import data
def getData(stocks, start, end):
    stockData = yf.download(stocks, start=start, end=end)

    # Extract adjusted closing prices for each stock
    adjClose = stockData['Adj Close']

    # Handle missing values before calculating returns
    adjClose = adjClose.dropna()

    # Calculate cumulative returns
    initial_price = adjClose.iloc[0]
    final_price = adjClose.iloc[-1]
    cumulative_returns = ((final_price - initial_price) / initial_price) * 100

    # Create a DataFrame directly with cumulative returns
    cumulative_returns_df = pd.DataFrame({'Cumulative_Return': cumulative_returns})

    return adjClose, cumulative_returns_df

# Specify stocks and date range
stocks = ['AAPL', 'MSFT', 'GOOGL', 'BAC', 'TSLA', 'NVDA', 'NFLX', 'E', 'UCG.MI', 'FBK']
start_date = '2022-01-01'
# Set current date
end_date = dt.datetime.now().strftime('%Y-%m-%d')

# Get data
adj_close_prices, cumulative_returns_df = getData(stocks, start_date, end_date)

# Determine deciles
quartile_10 = cumulative_returns_df['Cumulative_Return'].quantile(0.25)
quartile_90 = cumulative_returns_df['Cumulative_Return'].quantile(0.75)

# Create a column indicating the color for each stock
cumulative_returns_df['Color'] = np.where(cumulative_returns_df['Cumulative_Return'] <= quartile_10, 'red',
                                          np.where(cumulative_returns_df['Cumulative_Return'] >= quartile_90, 'green', 'blue'))

# Sort DataFrame by cumulative returns in descending order
cumulative_returns_df = cumulative_returns_df.sort_values(by='Cumulative_Return', ascending=False)

# Display a bar chart with colored columns
plt.figure(figsize=(12, 6))
plt.bar(cumulative_returns_df.index, cumulative_returns_df['Cumulative_Return'], color=cumulative_returns_df['Color'])
plt.title('Cumulative Return per Stock (Sorted by Cumulative Returns)')
plt.xlabel('Stock')
plt.ylabel('Cumulative Return (%)')
plt.xticks(rotation=45, ha='right')
legend_labels = ['Low Returns', 'High Returns']
legend_handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='red', markersize=10),
                  plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='green', markersize=10)]
plt.legend(legend_handles, legend_labels)

plt.show()
```

Volatility

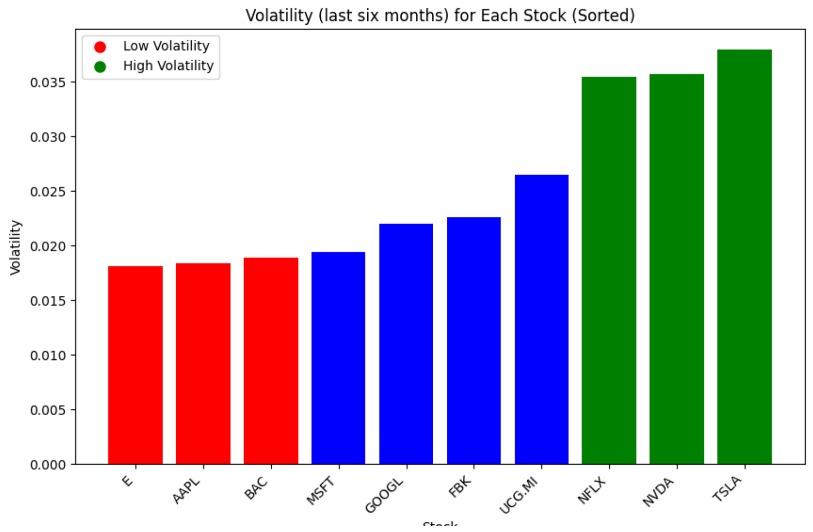
This strategy is based on the empirical observation that future returns of previously low-return-volatility portfolios outperform those of previously high-return-volatility portfolios. We can see that this goes counter to the expectation that higher risk assets should yield proportionately higher returns. That happens because we are evaluating on the risk-adjusted basis ("low-risk anomaly"). Similar to the previous example stocks are sorted, but based on volatility, specifically using standard deviation.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}} \tag{1}$$

Code

```
# Calculate historical volatility
def calculate_volatility(returns):
    # Calculate daily percentage returns
    returns = returns.pct_change().dropna()
    mean_returns = returns.mean()
    variance = (returns - mean_returns).pow(2).sum() / len(returns)
    volatility = math.sqrt(variance)

    return volatility

# Specify stocks and date range
stocks = ['AAPL', 'MSFT', 'GOOGL', 'BAC', 'TSLA', 'NVDA', 'NFLX', 'E', 'UCG.MI', 'FBK']
start_date = (dt.datetime.now() - pd.DateOffset(months=6)).strftime('%Y-%m-%d')  # Six months ago
end_date = dt.datetime.now().strftime('%Y-%m-%d')  # Set current date

# Calculate volatility for each stock
volatility_data = adj_close_prices[stocks].apply(calculate_volatility)

# Create a DataFrame for volatility and color
volatility_df = pd.DataFrame({'Volatility': volatility_data})

# Sort DataFrame by volatility in ascending order
volatility_df = volatility_df.sort_values(by='Volatility', ascending=True)

# Determine quartiles
quartile_10_vol = volatility_df['Volatility'].quantile(0.25)
quartile_90_vol = volatility_df['Volatility'].quantile(0.75)

# Create a column indicating the color for each stock
volatility_df['Color'] = np.where(volatility_df['Volatility'] <= quartile_10_vol, 'red',
                                  np.where(volatility_df['Volatility'] >= quartile_90_vol, 'green', 'blue'))

# Display a bar chart with sorted volatilities and colored deciles
plt.figure(figsize=(10, 6))
bars = plt.bar(volatility_df.index, volatility_df['Volatility'], color=volatility_df['Color'])
plt.title('Volatility (last six months) for Each Stock (Sorted)')
plt.xlabel('Stock')
plt.ylabel('Volatility')
plt.xticks(rotation=45, ha='right')

# Create a custom legend for colors
legend_labels = ['Low Volatility', 'High Volatility']
legend_handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='red', markersize=10),
                  plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='green', markersize=10)]
plt.legend(legend_handles, legend_labels)

plt.show()
```

An histogram is plotted, allowing for the possibility of buying stocks with lower volatility and selling those with higher volatility.



Residual Momentum

This is a stretegy similar to the price-momentum with the difference that considers the residual from the linear regression with the 3 Fama-French factors MKT(t), SMB(t), HML(t) in place of the returns. The Fama-French factors capture market performance, company size, and value-related factors, respectively. By utilizing residuals, the strategy aims to filter out the influence of these factors, emphasizing the unique patterns beyond general market trends and size or value considerations. Investors employing the Residual Momentum strategy may seek to capitalize on more refined signals for potential market outperformance or underperformance.

$$R_i(t) = \alpha_i + \beta_{1,i} MKT(t) + \beta_{2,i} SMB(t) + \beta_{3,i} HML(t) + \varepsilon_i(t) \tag{2}$$

Pairs trading

With pairs trading it is meant the goal of identifying a pair of historically highly correlated stocks and, when a mispricing (i.e., a deviation from the high historical correlation) occurs, shorting the “rich” stock and buying the “cheap” one. This is an example of a mean-reversion strategy.

$$R = \frac{P(t_2)}{P(t_1)} - 1 \qquad \bar{R} = \frac{1}{2}(R_A + R_B) \qquad \widetilde{R}_{A,B} = R_{A,B} - \bar{R} \tag{3}$$

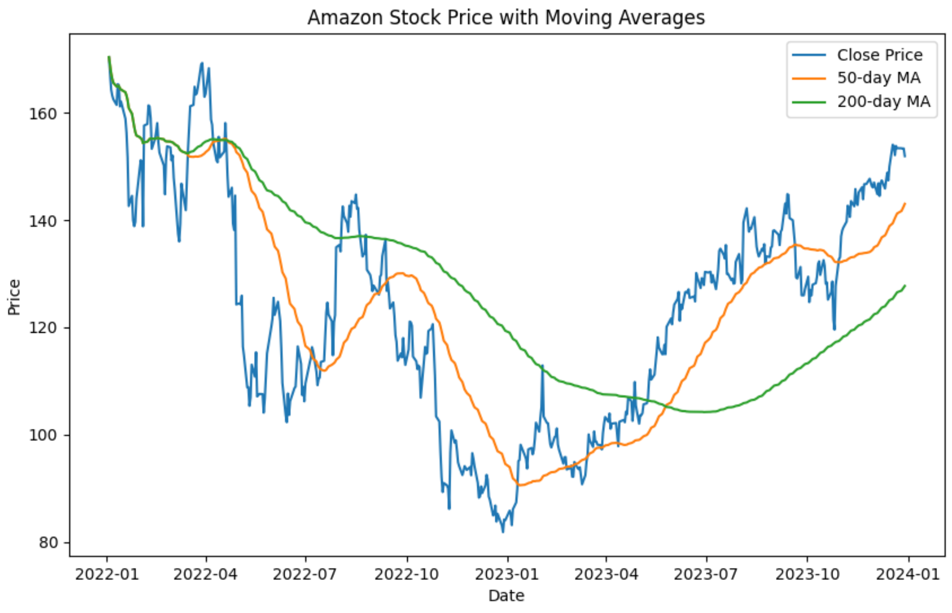
Let $\widetilde{R}_A$ and $\widetilde{R}_B$ be the demeaned returns, where $\bar{R}$ is the mean return. A stock is “rich” if its demeaned return is positive, and it is “cheap” if its demeaned return is negative. Demeaned returns help remove common market effects or factors that affect both stocks in a pair similarly. By subtracting the mean return (or other appropriate measure of central tendency) from the actual returns, you focus on the relative performance of each stock, independent of overall market movements. In fact a low demeaned return indicates that, relative to its historical average or the average of the pair, the stock is underperforming. This may be due to factors specific to that stock rather than general market conditions. The stock might be undervalued or experiencing a temporary dip in performance, presenting an opportunity for mean reversion.

Moving Avarages

This strategy is based on moving avarages (SMA) and their crossovers. In this practical example only two SMA are considered, the 50 and 200 periods, but generally more than only two are used in order to find a well behaved strategy. When the shorter SMA encounters the longer from above it means that the price is going bearish, while if it happens the opposite the sentiment is considered bullish.

Code

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

# Download historical data for Amazon stock
amazon_data = yf.download('AMZN', start='2022-01-01', end='2024-01-01')

# Calculate the 50-day and 200-day moving averages
short_window = 50
long_window = 200

# Utilizing function rolling in order to compute all the avarages
amazon_data['50_MA'] = amazon_data['Close'].rolling(window=short_window, min_periods=1).mean()
amazon_data['200_MA'] = amazon_data['Close'].rolling(window=long_window, min_periods=1).mean()

# Plot the data with moving averages
plt.figure(figsize=(10, 6))
plt.plot(amazon_data['Close'], label='Close Price')
plt.plot(amazon_data['50_MA'], label=f'50-day MA')
plt.plot(amazon_data['200_MA'], label=f'200-day MA')

plt.title('Amazon Stock Price with Moving Averages')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```



KNN alghoritm

The K-Nearest Neighbors (KNN) algorithm is a supervised learning method used for classification and regression problems. In the context of stock analysis, applying KNN involves several steps.

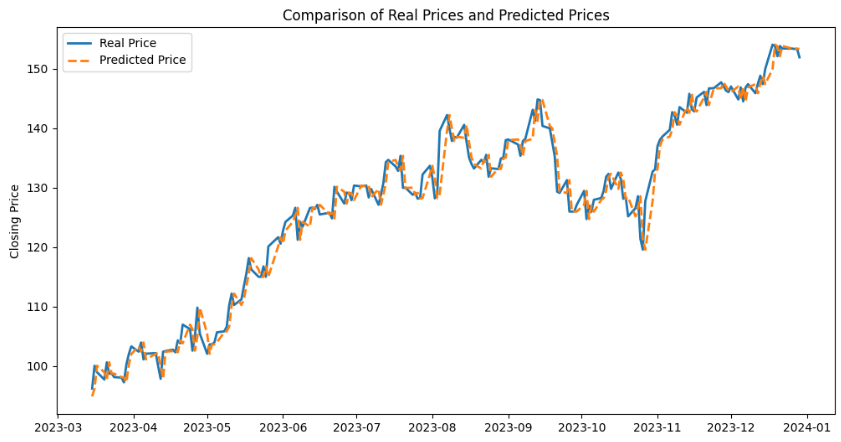
Firstly, historical data on stocks is collected, including daily closing prices, trading volumes, and other relevant indicators. This data is prepared by normalizing or standardizing features to ensure they carry equal weight. Next, the number of neighbors (K) to consider during the prediction phase is chosen. This represents the number of closest points that will influence the classification or prediction of a new data point.

The algorithm calculates the distance between the point of interest (the new observation) and all points in the training set using a distance metric like Euclidean distance. The K nearest neighbors are identified based on this distance. For a classification problem, the most common class label among the nearest neighbors is assigned to the point of interest. For a regression problem it can also be used the average of the output values of the nearest neighbors.

It's important to note that the dataset is commonly divided into a training set (60%) and a test set (40%). The training set is used to train the model, while the test set is used to assess the algorithm's ability to make predictions on data not seen during training. This helps evaluate the model's ability to generalize to new data and reduces the risk of overfitting.

Code

```python
import yfinance as yf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Retrieve data from Yahoo Finance
amazon_data = yf.download('AMZN', start='2022-01-01', end='2024-01-01')

# Use closing price as a feature
features = amazon_data[['Close']].values

# Label: 1 if the price increases, 0 otherwise
labels = (amazon_data['Close'].shift(-1) > amazon_data['Close']).astype(int).values

# Drop missing values
amazon_data.dropna(inplace=True)

# Create training and test sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.4, random_state=42)

# Data normalization
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the KNN model
k = 4
knn_model = KNeighborsClassifier(n_neighbors=k)
knn_model.fit(X_train_scaled, y_train)

# Predictions on the test data
predictions = knn_model.predict(X_test_scaled)

# Create an array of dates corresponding to the test data
dates = amazon_data.index[-len(predictions):]

# Plot the real prices
plt.figure(figsize=(12, 6))
plt.plot(dates, amazon_data['Close'][-len(predictions):], label='Real Price', linewidth=2)

# Plot the predicted prices
plt.plot(dates, amazon_data['Close'][-len(predictions)-1:-1].values, label='Predicted Price', linestyle='dashed', linewidth=2)

plt.title('Comparison of Real Prices and Predicted Prices')
plt.ylabel('Closing Price')
plt.legend()
plt.show()

# Accuracy Score
accuracy_test = accuracy_score(y_test, knn_model.predict(X_test))|
print ('Test_data Accuracy: %.2f' %accuracy_test)
```



This strategy has to be backtested out-of-sample and the process of diving the data set in a training and a test group is called cross-validation. The goodness in forecasting can be then evaluated by the minimization of the MSE, or also the out of sample error.

$$MSE = \sum (\hat{y}_i - y_i)^2 \tag{4}$$

In this case an accuracy test is performed:

```
Test_data Accuracy: 0.54
```

The first disadvantage is that equally weighting contributions of all k nearest neighbors could be suboptimal. The second is that since KNN is a lazy algorithm, it takes up more memory and data storage compared to other classifiers. This can be costly from both a time and money perspective.

ARIMA model

ARIMA (AutoRegressive Integrated Moving Average) is a statistical model used for time series forecasting. It combines AutoRegression (AR), Integration (I), and Moving Average (MA) components. The AR part involves predicting future values based on past values, while the I part deals with differencing to achieve stationarity. The MA part considers past error terms through a moving average. In an AR(p) model the future value of a variable is assumed to be a linear combination of p past observations and a random error together with a constant term. Mathematically the AR(p) model can be expressed as :

$$y_t = c + \sum_{j=1}^{p} \varphi_i y_{t-i} + \varepsilon_t = c + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \ldots + \varphi_p y_{t-p} + \varepsilon_t \tag{5}$$

Just as an AR(p) model regress against past values of the series, an MA(q) model uses past errors as the explanatory variables. The MA(q) model is given by :

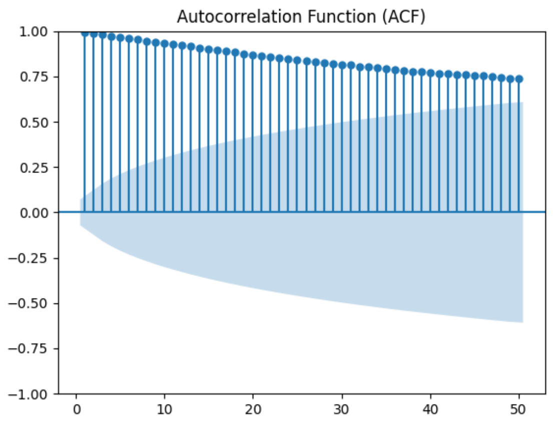
$$y_t = \mu + \sum_{i=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t = \mu + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_p \varepsilon_{t-p} + \varepsilon_t \tag{6}$$

The random shocks are assumed to be a white noise process, i.e. a sequence of independent and identically distributed (i.i.d) random variables with zero mean and a constant variance σ^2. Thus conceptually a moving average model is a linear regression of the current observation of the time series against the random shocks of one or more prior observations.

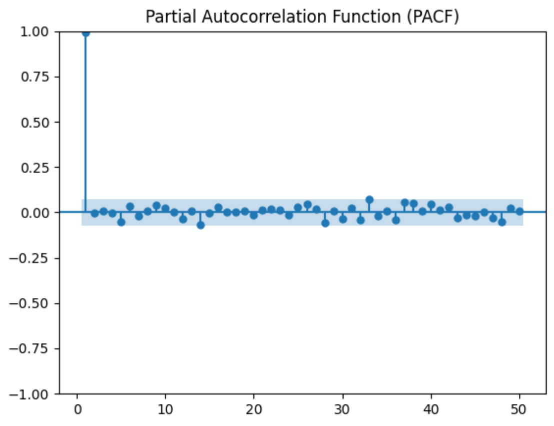
The formal notation is ARIMA(p, d, q), where p is the order of AutoRegression, d is the degree of differencing, and q is the order of Moving Average.

Autoregressive (AR) and moving average (MA) models can be effectively combined together to form a general and useful class of time series models, known as the ARMA models. The ARMA models, described above can only be used for stationary time series data. Thus from application view point ARMA models are inadequate to properly describe non-stationary time series, which are frequently encountered in practice. For this reason the ARIMA model is proposed, which is a generalization of an ARMA model to include the case of non-stationarity as well. In ARIMA models a non-stationary time series is made stationary by applying finite differencing of the data points. Another limit coming from ARIMA model is that it is for non-seasonal non-stationary data. Box and Jenkins have generalized this model to deal with seasonality. Their proposed model is known as the Seasonal ARIMA (SARIMA) model.

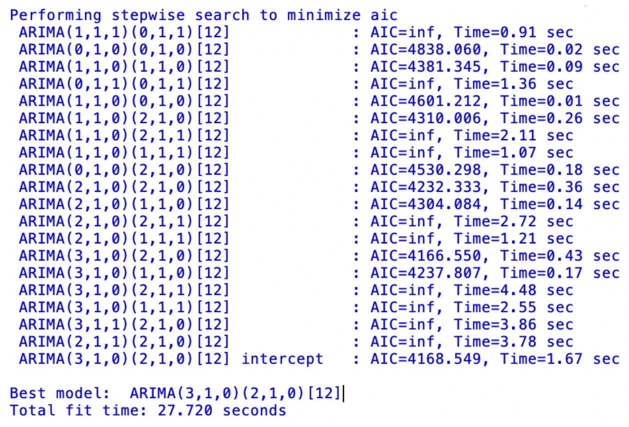
In this practical scenario, Amazon stock closing prices from January 1, 2021, to January 1, 2024, are considered for the application of the ARIMA model. To perform model identification, i.e., choosing the right parameter values, it can be beneficial to examine Autocorrelation (ACF) and Partial Autocorrelation (PACF). Specifically, while ACF indicates how many lagged observations could significantly influence the dependent variable, PACF is used to measure the correlation between an observation k periods ago and the current observation, after controlling for data points at intermediate lags. By observing ACF and PACF values outside the confidence interval, it is possible to deduce the proper values for both parameters, p and q.



Autocorrelation



Partial Autocorrelation



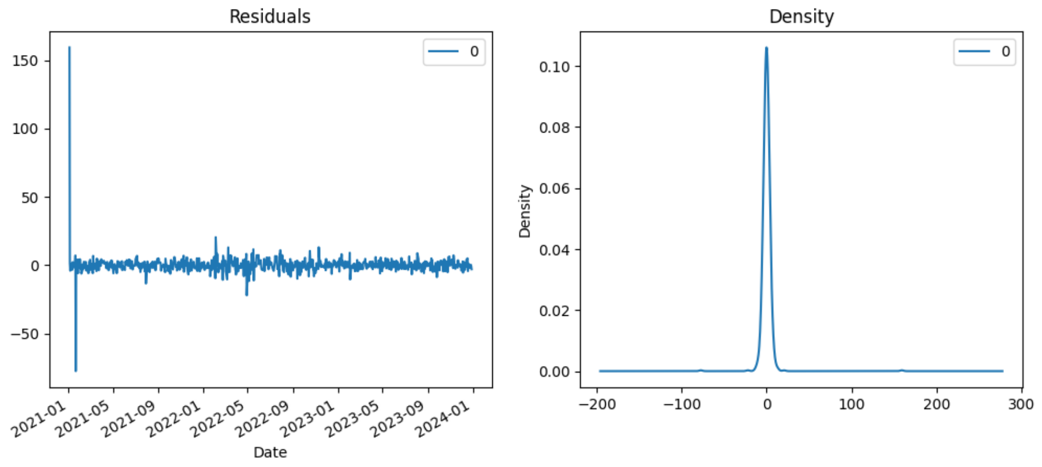
```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=inf, Time=0.91 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=4838.060, Time=0.02 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=4381.345, Time=0.09 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=inf, Time=1.36 sec
 ARIMA(1,1,0)(0,1,0)[12]             : AIC=4601.212, Time=0.01 sec
 ARIMA(1,1,0)(2,1,0)[12]             : AIC=4310.006, Time=0.26 sec
 ARIMA(1,1,0)(2,1,1)[12]             : AIC=inf, Time=2.11 sec
 ARIMA(1,1,0)(1,1,1)[12]             : AIC=inf, Time=1.07 sec
 ARIMA(0,1,0)(2,1,0)[12]             : AIC=4530.298, Time=0.18 sec
 ARIMA(2,1,0)(2,1,0)[12]             : AIC=4232.333, Time=0.36 sec
 ARIMA(2,1,0)(1,1,0)[12]             : AIC=4304.084, Time=0.14 sec
 ARIMA(2,1,0)(2,1,1)[12]             : AIC=inf, Time=2.72 sec
 ARIMA(2,1,0)(1,1,1)[12]             : AIC=inf, Time=1.21 sec
 ARIMA(3,1,0)(2,1,0)[12]             : AIC=4166.550, Time=0.43 sec
 ARIMA(3,1,0)(1,1,0)[12]             : AIC=4237.807, Time=0.17 sec
 ARIMA(3,1,0)(2,1,1)[12]             : AIC=inf, Time=4.48 sec
 ARIMA(3,1,0)(1,1,1)[12]             : AIC=inf, Time=2.55 sec
 ARIMA(3,1,1)(2,1,0)[12]             : AIC=inf, Time=3.86 sec
 ARIMA(2,1,1)(2,1,0)[12]             : AIC=inf, Time=3.78 sec
 ARIMA(3,1,0)(2,1,0)[12] intercept   : AIC=4168.549, Time=1.67 sec

Best model:  ARIMA(3,1,0)(2,1,0)[12]
Total fit time: 27.720 seconds
```

Auto ARIMA function

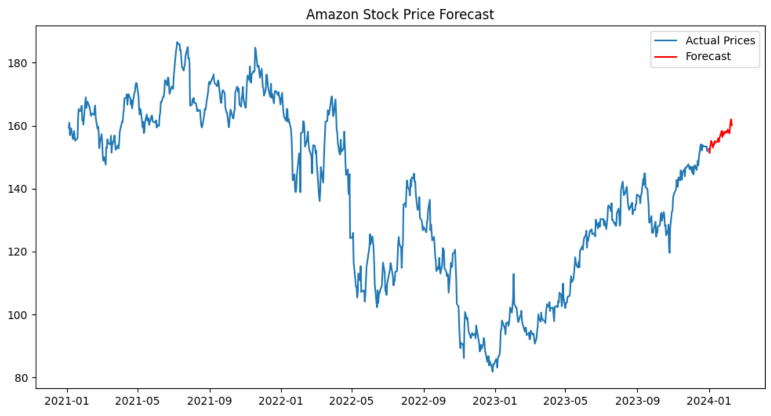
The process of model identification can also be done by an automated function (.auto_arima()) which leads to the best model without having to fit and try multiple values for p and q. This function is based on the minimization of AIC (Akaike Information Criterion), a statistical measure that quantifies the trade-off between the goodness of fit of a statistical model and its complexity (i.e. numbers of parameters p).

$$AIC = n \ln\left(\sigma^2/n\right) + 2p \tag{7}$$

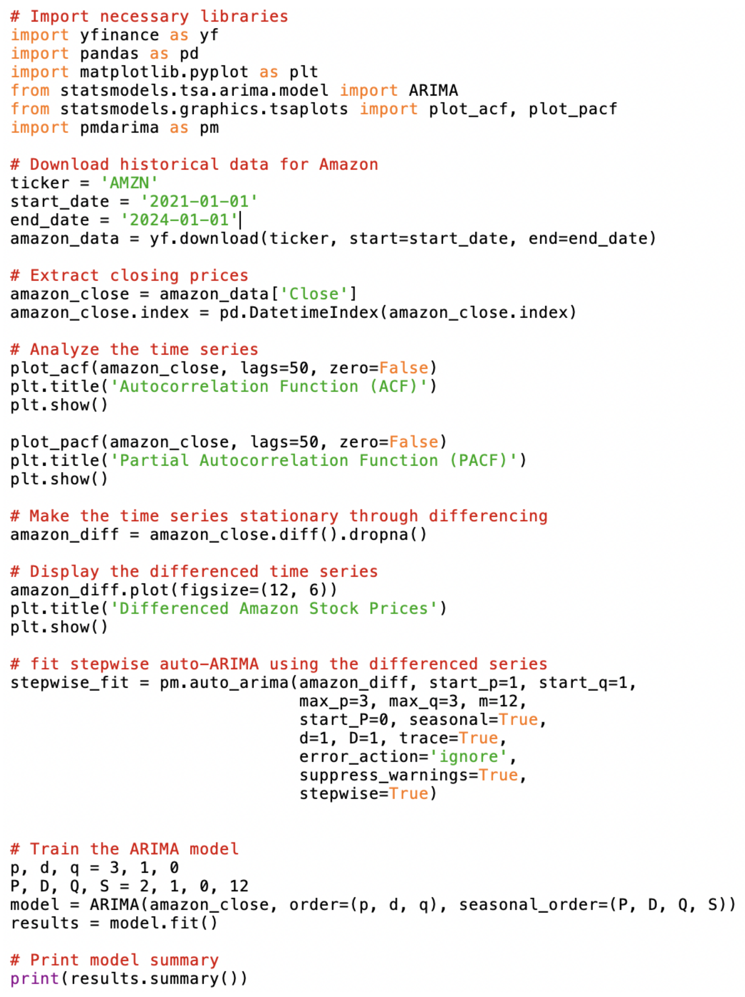
The model has been trained using the previously mentioned parameters, which include considerations for the seasonal influence of lagged observations. In this scenario, the initially contemplated ARIMA model has transformed into a SARIMA model. Examining the residual plot reveals that there is no discernible pattern indicating that our chosen model adequately captures the autocorrelation of the variable.



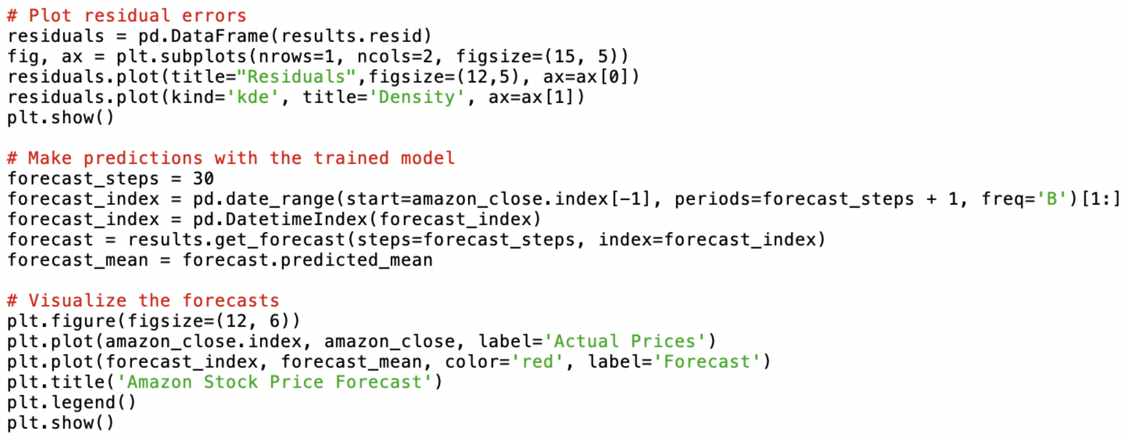
Following the training of the SARIMA model, it is employed to forecast the closing price of Amazon stock beyond the sample data, thereby providing predictions for future values. This out-of-sample forecasting allows us to assess the model's performance in predicting the behavior of the Amazon stock price beyond the time frame it was trained on.



Code



```
# Import necessary libraries
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import pmdarima as pm

# Download historical data for Amazon
ticker = 'AMZN'
start_date = '2021-01-01'
end_date = '2024-01-01'
amazon_data = yf.download(ticker, start=start_date, end=end_date)

# Extract closing prices
amazon_close = amazon_data['Close']
amazon_close.index = pd.DatetimeIndex(amazon_close.index)

# Analyze the time series
plot_acf(amazon_close, lags=50, zero=False)
plt.title('Autocorrelation Function (ACF)')
plt.show()

plot_pacf(amazon_close, lags=50, zero=False)
plt.title('Partial Autocorrelation Function (PACF)')
plt.show()

# Make the time series stationary through differencing
amazon_diff = amazon_close.diff().dropna()

# Display the differenced time series
amazon_diff.plot(figsize=(12, 6))
plt.title('Differenced Amazon Stock Prices')
plt.show()

# fit stepwise auto-ARIMA using the differenced series
stepwise_fit = pm.auto_arima(amazon_diff, start_p=1, start_q=1,
                             max_p=3, max_q=3, m=12,
                             start_P=0, seasonal=True,
                             d=1, D=1, trace=True,
                             error_action='ignore',
                             suppress_warnings=True,
                             stepwise=True)


# Train the ARIMA model
p, d, q = 3, 1, 0
P, D, Q, S = 2, 1, 0, 12
model = ARIMA(amazon_close, order=(p, d, q), seasonal_order=(P, D, Q, S))
results = model.fit()

# Print model summary
print(results.summary())
```



```
# Plot residual errors
residuals = pd.DataFrame(results.resid)
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))
residuals.plot(title="Residuals",figsize=(12,5), ax=ax[0])
residuals.plot(kind='kde', title='Density', ax=ax[1])
plt.show()

# Make predictions with the trained model
forecast_steps = 30
forecast_index = pd.date_range(start=amazon_close.index[-1], periods=forecast_steps + 1, freq='B')[1:]
forecast_index = pd.DatetimeIndex(forecast_index)
forecast = results.get_forecast(steps=forecast_steps, index=forecast_index)
forecast_mean = forecast.predicted_mean

# Visualize the forecasts
plt.figure(figsize=(12, 6))
plt.plot(amazon_close.index, amazon_close, label='Actual Prices')
plt.plot(forecast_index, forecast_mean, color='red', label='Forecast')
plt.title('Amazon Stock Price Forecast')
plt.legend()
plt.show()
```

Comment

Single-stock technical analysis strategies, such as those relying on moving averages, momentum, single-stock KNN and ARIMA model, are often considered "unscientific" by many professionals and academics. While moving averages are used in trend following/momentum strategies, applying them to individual stocks could bring to biased conclusions. In contrast, strategies involving a large cross-section of stocks introduce statistical elements, making mean-reversion more plausible due to correlations within industries. But it is important to remember that :" *stock market – an imperfect man-made construct – is not governed by laws of nature the same way as, say, the motion of planets in the solar system is governed by fundamental laws of gravity. The markets behave the way they do because their participants behave in certain ways, which are sometimes irrational and certainly not always efficient*". (Z. Kakushadze and J.A. Serur, 151 Trading Strategies)

References

[1] Momentum trading
https://www.5paisa.com/stock-market-guide/online-trading/what-is-momentum-trading#:~:
text=As%20per%20Momentum%20Trading%2C%20you,one%20direction%20for%20long%20periods.

[2] Pairs trading
https://www.fidelity.com/learning-center/trading-investing/trading/pairs-trading

[3] Volatility trading
https://deliverypdf.ssrn.com/delivery.php?ID=16308209308706907407410811900109211312500207106502503500
EXT=pdf&INDEX=TRUE

[4] Residual Momentum strategy
https://quantpedia.com/strategies/residual-momentum-factor/#:~:text=A%20residual%
20momentum%20strategy%20based,long%2Drun%20average%20Sharpe%20ratio.

[5] KNN alghoritm
https://www.ibm.com/it-it/topics/knn

[6] KNN alghoritm 2
https://medium.com/@tpreethi/introduction-to-k-nearest-neighbors-knn-algorithm-python-implementatio

[7] KNN alghoritm 3
https://blog.quantinsti.com/machine-learning-k-nearest-neighbors-knn-algorithm-python/

[8] Arima
https://www.performancetrading.it/Documents/VrAnalisi/VrA_Arima1.htm

[9] Arima 2
http://www.riccir.altervista.org/SerieStoriche/ssto6.html

[10] Ratnadip Adhikari and R. K. Agrawal , *An Introductory Study on Time Series Modeling and Forecasting*, LAP Lambert Academic Publishing, Germany, 2013.

[11] Z. Kakushadze and J.A. Serur, *151 Trading Strategies*, Palgrave Macmillan, 2018.